Information

FAQ（よくあるお問い合わせ）

ルネサス エレクトロニクスの半導体製品

本資料に記載のすべての情報は、本資料発行時点のものであり、ルネサス エレクトロニクスは、予告なしに、本資料に記載した製品または仕様を変更することがあります。
ルネサス エレクトロニクスのホームページなどにより、公開される最新情報をご確認ください。

ルネサス エレクトロニクス
www.renesas.com

Rev.4.00　2010.12

【 この FAQ の構成 】

製品共通

1. 品　　名 …4
2. 規格・信頼性 …12
   - 2.1 規　　格 …12
   - 2.2 規格外の電圧印加 …18
   - 2.3 信頼性 …19
   - 2.4 難燃性 …22
   - 2.5 サンプル …22
3. 設　　計 …23
   - 3.1 1ピン・マーク …23
   - 3.2 実装・組み立て …23
   - 3.3 未使用端子の処理 …28
   - 3.4 初期状態 …30
   - 3.5 熱設計 …30
   - 3.6 ノイズ対策 …33
   - 3.7 べし・べからず集 …36
   - 3.8 トラブル・シュート …40
4. 環境関連 …43
   - 4.1 環境負荷 …43
   - 4.2 鉛フリー …46
5. 購入・出荷 …49
   - 5.1 購　　入 …49
   - 5.2 出荷形態 …53
   - 5.3 相当品 …56
   - 5.4 輸　　出 …57
6. 基礎知識 …59
   - 6.1 電　　気 …59
   - 6.2 半導体 …65
   - 6.3 電気回路 …84
7. 会社・事業 …87

製品別

1. マイコン …89
   - 1.1 共　　通 …102
   - 1.2 78K デバイス …424
   - 1.3 V850 デバイス …646
2. プラットフォーム製品 …711
   - 2.1 ゲート・アレイ …711
3. 表示・画像処理 …714
   - 3.1 文字表示 IC（OSD） …714
   - 3.2 CCD …724
4. 通　　信 …736
   - 4.1 USB …736
   - 4.2 IEBus …750
5. メモリ …753
   - 5.1 同期SRAM …753
6. リニア IC …755
   - 6.1 オペアンプ／コンパレータ …755
   - 6.2 電源用 IC …785
7. ディスクリート …814
   - 7.1 共　　通 …814
   - 7.2 トランジスタ …819
   - 7.3 FET …825
   - 7.4 ダイオード …832
8. オプトエレクトロニクス …837
   - 8.1 共　　通 …837
   - 8.2 フォトカプラ …840
   - 8.3 光 MOS FET …848

用語集

1. 品質・信頼性用語 …850
2. 設計の予備知識 …855
3. 通信関連用語 …953

（50 音、アルファベット順）

ウェブ・コラムのご紹介









このドキュメントは、Web掲載のFAQを基に、おもなデバイス製品に関する情報を編集したダイジェスト版です。
旧NECエレクトロニクス社が所管していた製品に関する情報を収録しています。

Web FAQ：http://www.renesas.com/ja/faq/index.html

本文欄外の★印は、本版で改訂された主な箇所を示しています。
文中の青字（下線部）は、PDFでの該当箇所やウェブ・コンテンツへのリンク箇所です。

製品共通

1. 品　　名 …4
2. 規格・信頼性 …12
3. 設　　計 …22
4. 環境関連 …43
5. 購入・出荷 …49
6. 基礎知識 …59
7. 会社・事業 …87

1. 品　名

品名の構成

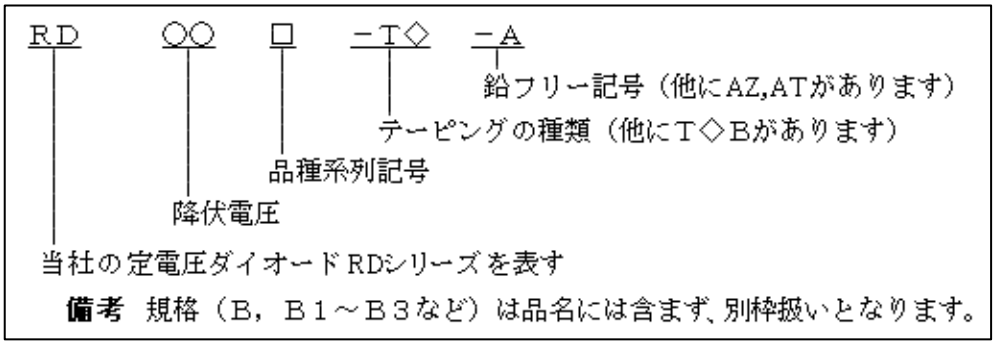
数字とアルファベットを組み合わせて構成します。左から順に、ファミリ、製品番号、改良区分、パッケージ区分、個別番号などです。各項目は、製品やファミリによって内容が異なります。

製品への表示

小型の製品は、品名のすべてを捺印できませんので、品名の一部や記号などで表示することがあります。
品名の他に、検査記号や製造番号などを表示することもあります。

ツェナー・ダイオード

RD シリーズを例に説明します（ツェナー・ダイオードの品名は、ルネサス エレクトロニクス独自の命名方式です）。



μとU

Q1. 梱包箱のラベルに「UPD***」と製品名が印字されていますが、正式名称は「μPD***」ではないでしょうか。

A1. データ・シートなどのドキュメントでは「μ」（品名では「ミュー」と読む）を使用していますが、一般文書では、PC のキーボードに「μ」がないので、代わりに「U」を使用することが慣習化し、定着しています。
なお、μPA、μPB、μPC および μPD は、NECエレクトロニクス株式会社の登録商標です。

★ パッケージ・コード

Q1. μPD7105x のように、パッケージ・コードが G と GB で流通しているものがありますが、違いは何ですか。

A1. 基本的に本体樹脂厚が違うだけで、ピン・ピッチや電気的特性は同じです。
もともと QFP のパッケージ・コードは G でした。1990 年ごろにリフロや VPS といった全体加熱の自動実装機の普及に伴って、パッケージ・クラックが発生するようになり、製品ごとの吸湿管理をするようになりました。このとき、uPD7105xGB なら本体樹脂厚を約 2 倍の 2.7mm に変更し、さらに 3 けたの細区分コードを付加してパッケージを特定できるようにしました。
なお、μPD7105xGB はすでに廃止品となっており、現状の流通品は模造の可能性があります。パッケージ・コードが G だけの製品は、特にご注意ください。

テーピング

リール出荷のテーピング品には、-E1、-E2、-T1、-T2 などの記号が付加され、1、2 でテープに対する製品の固定方向を示します。詳しくは出荷形態をご参照ください。

-E1、-E2(3 ピン・ミニモールド(SC-59)では-T1B、-T2B)はエンボス・テーピングで、樹脂テープのへこみ部分にデバイスを搭載し、樹脂フィルムでふたをしています。

もともと-T1、-T2 は粘着テーピングで、紙テープの粘着部にデバイスを固定しており、デバイスはテープに封入されていませんでした。現在、粘着テーピングは廃止しています。

ただし、初期のエンボス・テーピングには、-T1、-T2 が付加されたものがありました。

ダイオードとトランジスタ

最初の 2 文字か 3 文字は**ファミリ・タイプ**で、(社)電子情報技術産業協会(JEITA)の命名基準に従っています。

1 文字目：接合(ジャンクション)の数

2 文字目：S＝半導体(Semiconductor)の略

3 文字目：機能分類

タイプ	分　類	
1S	ダイオード	
2SA	バイポーラ・トランジスタ	PNP　高周波用
2SB		PNP　低周波用
2SC		NPN　高周波用
2SD		NPN　低周波用
2SJ	電界効果トランジスタ(FET)	P チャネル
3SJ		P チャネル(2 ゲート)
2SK		N チャネル
3SK		N チャネル(2 ゲート)
2SF	サイリスタ	

備考 1. 1N, 2N で始まる品名は、米国の命名方式です。

★ **2.** 次のコードで始まる品名は、当社が独自に命名しています。

RD、NNCD、NSAD ： ツェナー・ダイオード

N ： JEITA の命名基準に代わる汎用トランジスタ、FET の新型名

ファミリ・タイプ	耐圧ランク	ランク別連番	タイプ	改版	特殊規格	パッケージ細区分	テーピング	鉛フリー
N	06	00	P	A	( )	-ZK	-E1	-AY

”改版”以降の内容については、後述「2SJ600」の例をご参照ください。

耐熱ランク

コード	V_{CEO},V_{DSS}(V)	コード	V_{CEO}/V_{DSS}(V)	コード	V_{CEO},V_{DSS}(V)
01	10-19	07	70-79	40-49	400-499
02	20-29	08	80-89	50-59	500-599
03	30-39	09	90-99	60-69	600-699
04	40-49	10-19	100-199	70-79	700-799
05	50-59	20-29	200-299	80-89	800-899
06	60-69	30-39	300-399	90-99	900-999

タイプ

コード	タイプ
R	PNP Tr
S	NPN Tr
P	P-ch FET
N	N-ch FET

NP：自動車電装用途のパワーMOSFET

NE：RF/マイクロ波のトランジスタ、FET

ファミリ・タイプ	品番	パッケージ・コード	テーピング	鉛フリー
NE	3503	M04	-T2	-A

”テーピング、鉛フリー”の内容については、後述「2SJ600」の例をご参照ください。

パッケージ・コード：http://www2.renesas.com/microwave/ja/packages/index.html

NDL、NR、NX：通信用光ダイオード（NDL は廃止）

PS：フォトカプラ、光 MOS(SSR)デバイス

PH：受光素子

NV：可視光レーザダイオード（廃止）

当社の品名について、「2SJ600」を例に説明します。

ファミリ・タイプ	品番	改版	特殊規格	パッケージ細区分	テーピング	鉛フリー
2SJ	600		(0)	-Z	-E1	-AZ

フィールド	内　容
ファミリ・タイプ	前述のとおりです。
品番	数字で製品番号を表します。
改版	アルファベット順に、1 文字でチップの改版を表します。 最初(改版がない場合)は、この記号はありません。
特殊規格	個別対応の品質水準やリード成形などは、特殊規格として明示します。
パッケージ細区分	端子挿入品を標準リード成形で表面実装品にして、-Z で示す製品があります。
テーピング	テープ・リールでの出荷品では、-E1, -E2 などの方向記号を付加します。 SC-59 パッケージのばら品では、例外的に-L を付加します。
鉛フリー	鉛フリー品は、品名の最後に、-A, -AZ などの分類記号を付加します。

備考 1. 上記品名のあとに、″/JM″などが付く製品が一部にあります。これらは、製造国を表した記号です。/の次の 1 文字目はこのICの拡散工場のある国、2 文字目は組み立て工場のある国を表します。たとえば、Jは日本(Japan)、Mはマレーシア(Malaysia)です。/の前までが同じなら、製造国が異なっても、機能／性能（データ・シートなどに記載の電気的特性）やパッケージ形状などは同じです。

2. ツェナー・ダイオードの規格区分(B1, B2 など)やトランジスタの h_{FE}ランクは品名ではなく、別扱いです。

IC と複合素子

ルネサス エレクトロニクス独自の命名をしています。

「μPD78064」を例に説明します。

ファミリ・タイプ	品番	改版	機能区分	パッケージ・タイプ	品質水準	カスタム・コード／スピード区分など	パッケージ細区分	テーピング	鉛フリー
μPD	78064	B		GC	(A)	-011	-8EU		-A

「μPD78064」は**マスク ROM** タイプです。

「μPD78P064」のように「P」を挿入すると、**PROM** タイプになります。

フラッシュ ROM タイプでは、「μPD78F0xxx」のように「F」が挿入されます。

フィールド	コード	内　容	
ファミリ・タイプ	μPA	複合素子	
	μPB	バイポーラ	デジタル IC
	μPC		リニア IC
	μPD	MOS デジタル IC	
	μPG	化合物デバイス（GaAs など）	
品番	数字で製品番号を表します。 上記のような"P"や"F"などのアルファベットが挿入されることもあります。		
改版	アルファベット順に、1 文字でチップの改版を表します。 最初(改版がない場合)は、この記号はありません。		
機能区分	マイコンでは、I^2C 内蔵品に"Y"を付加します。		
パッケージ・タイプ	1～2 文字で、パッケージの種類を示します。		
	A	金属	
	B	セラミック	
	C, CW	プラスチック DIP, プラスチック SDIP	
	D	セラミック DIP	
	Fx	BGA, LGA	
	Gx	QFP, SOP, TSOP, SSOP	
	Hx	SIP	
	Mx	SSOP の一部	
	Nx	BGA	
	Sx	BGA, PGA	
品質水準	高品質水準は(A), (A1), (A2)などを付加します。特殊規格を明示する製品もあります。 品質水準の詳細は「半導体デバイスの品質水準」をご参照ください。		
カスタム・コード／スピード区分など	ROM や ASIC など、お客様の仕様に基づく製品には、3 文字のカスタム・コードが付加されます。汎用品には、このフィールドはありません。また、周波数やメモリのアクセス時間のような、スピード区分などを表す場合もあります。		
パッケージ細区分	QFP, SOP などのパッケージで、サイズやピン・ピッチの区分を示すことがあります。		
テーピング	テープ・リールでの出荷品では、-E1, -E2 などの方向記号を付加します。		
鉛フリー	鉛フリー品は、品名の最後に、-A, -AZ などの分類記号を付加します。		

備考　上記品名のあとに、"/JM"などが付く製品が一部にあります。これらは、製造国を表す記号です。
/の次の 1 文字目はこのICの拡散工場のある国、2 文字目は組み立て工場のある国を表します。
たとえば、Jは日本(Japan)、Mはマレーシア(Malaysia)です。/の前までが同じなら、製造国が異なっても、機能／性能（データ・シートなどに記載の電気的特性）やパッケージ形状などは同じです。

三端子レギュレータ(シリーズ・レギュレータ)

三端子レギュレータも基本的には「IC と複合素子」の命名方式に従っていますが、その特徴／性能を表現するために、品名に次のような特徴があります(μPC305、317、337 などは例外で、以下の説明に準じていません)。
なお、現在量産中の製品一覧を次のホームページに掲載しています。
http://www2.renesas.com/linear/ja/product/series.html

「μPD121W33AT1F-E1-AT」を例に説明します。

①ファミリ・タイプ	②品番	③改版	④パッケージ・タイプ	⑤テーピング	⑥鉛フリー
μPD	121W33	A	T1F	-E1	-AT

①μＰＤ１２１Ｗ３３ＡＴ１Ｆ－Ｅ１－ＡＴ
　ファミリ・タイプ

ファミリ・タイプ	構　造
μPC	バイポーラ
μPD	MOS

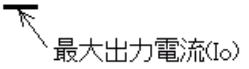
②μＰＤ１２１Ｗ３３ＡＴ１Ｆ－Ｅ１－ＡＴ
　品番

１２１Ｗ３３
シリーズ

シリーズ	特　徴
μPC78	標準タイプ
μPC79	
μPC35	
μPC24	低飽和タイプ
μPC29	
μPC29S	低飽和タイプ、ON/OFF 機能付き
μPC30	
μPC35	標準タイプ
μPC37	低飽和タイプ、2 出力
μPD120	低飽和タイプ、CMOS(低電圧、低消費電力)
μPD121	低飽和タイプ、CMOS(低電圧、低消費電力)、ON/OFF 機能付き

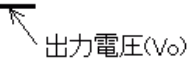
備考　製品によっては廃止品がありますので、販売店にご確認ください。

１２１Ｗ３３



記号	最大出力電流(Io max.)
L	0.1A
N	0.3A
M	0.5A
（なし）	1.0A
W	1.5A
A	2.0A

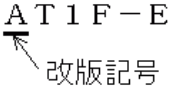
１２１Ｗ３３



番号	出力電圧（Vo typ.）
00	可変電圧
04	4.0V
05	5.0V
08	8.0V
09	9.0V
10	10.0V または 1.0V
12	12.0V
15	15.0V または 1.5V
18	18.0V または 1.8V
24	24.0V
25	2.5V
33	3.3V
78	7.8V
93	9.3V

備考　負電圧用（μPD79 シリーズなど）では、出力電圧はマイナスになります。

③μＰＤ１２１Ｗ３３ＡＴ１Ｆ－Ｅ１－ＡＴ



アルファベット順に、1 文字でチップの改版を表します。

最初(改版がない場合)は、この記号はありません。

④μＰＤ１２１Ｗ３３ＡＴ１Ｆ－Ｅ１－ＡＴ

パッケージ・タイプ

記号	パッケージ・タイプ
H	TO-220 または TO-126（MP-5 または 4 ピン MP-5）注
HB	TO-251（MP-3 または SC-99 注）
HF	TO-220 絶縁型（MP-45G）
J	TO-92
T	TO-252（MP-3Z）または SOT-89(SC-62) 注
TA	SOT-457（SC-74A）
TJ	TO-252（SC-98）
T1B	SOT-89（SC-62）
T1D	TO-252（MP-3Z または MP-3ZK 注）または TO-220 絶縁型（MP—45G）注
T1F	TO-252（5 ピン MP-3ZK）
C	DIP-8（8 ピン）
GR または G2	SOP-8（8 ピン、225mil）

注　製品よって異なりますので、データ・シートや次のホームページをご確認ください。

http://www2.renesas.com/linear/ja/product/series.html

パッケージの詳細については、次のホームページ（デスクリート製品のパッケージ情報）をご参照ください。

端子挿入：http://www2.renesas.com/discrete/ja/package/thd.html

表面実装：http://www2.renesas.com/discrete/ja/package/smd.html

⑤μＰＤ１２１Ｗ３３ＡＴ１Ｆ－Ｅ１－ＡＴ

テーピング包装（テープの種類と引き出し方向）

テーピング包装の記号の意味については、次のホームページをご参照ください。

http://www2.renesas.com/pkg/ja/pack/p_gaiyou_02.html

関連 FAQ：http://www2.renesas.com/ja/faq/f_name.html#0005

http://www2.renesas.com/ja/faq/f_ship.html#0001

⑥μＰＤ１２１Ｗ３３ＡＴ１Ｆ－Ｅ１－ＡＴ

鉛フリー記号

鉛フリー記号の意味については、次のホームページをご参照ください。

http://www2.renesas.com/pb_free/ja/identifi.html

関連 FAQ：http://www2.renesas.com/ja/faq/f_pb.html#0002

★

抵抗内蔵トランジスタ

FA1A4M を例に説明します（抵抗内蔵トランジスタの品名は、ルネサス エレクトロニクス独自の命名方式です）。

パッケージ	等価回路（耐圧）	R1 抵抗（有効数字）	R1 抵抗（指数 x10^n）	R2/R1 比
F	A1	A	4	M

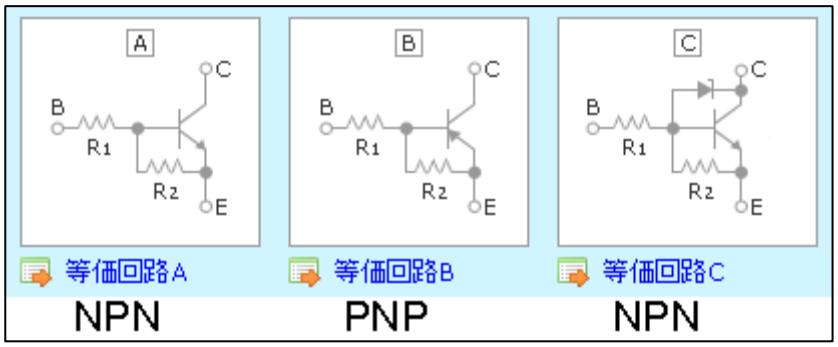
外形	記号
SC-59	F
SC-62	H
SC-70	G
SC-75	K

等価回路 (Vceo)	記号
等価回路A (20V)	A1～M1
等価回路B(-20V)	N1～X1
等価回路A (25V)	A1～M1
等価回路B(-25V)	N1～X1
等価回路A (60V)	A1～M1
等価回路B(-60V)	N1～X1
等価回路C (60V)	A2～M2
等価回路A(50V)	A4～M4
等価回路B(-50V)	N4～X4

抵抗値	記号
1.0	A
1.2	B
1.5	C
1.8	E
2.2	F
2.7	H
3.3	J
3.9	K
4.7	L
5.6	M
6.8	N
8.2	P

n	記号
1	1
2	2
3	3
4	4
5	5
6	6

比	記号
R1なし	A
1/100	E
1/50	F
1/20	H
1/10	J
1/5	K
1/2	L
1	M
2	N
5	P
10	Q
20	R
50	S
100	T
R2なし	Z



備考 R1 なしの場合、4,5けた目の記号は R2 の抵抗値を表します。

2. 規格・信頼性

2.1 規　格

規格の項目
絶対最大定格 一瞬たりとも、越えてはならない値です。また、各項目は独立しており、1 項目でも越えてはなりません。
動作（周囲）温度（TA、Topt） 正常動作を保証できる使用環境温度です。 ディスクリートは機能デバイスでないため、保存温度 Tstg が動作周囲温度と等価です。
ジャンクション温度（動作接合部温度）（TJ、Tj）、**チャネル温度**（Tch） デバイス内部のチップが許容する最高温度で、動作周囲温度と IC の自己発熱による上昇温度の和となります。 パワー・デバイスなどで、熱抵抗を基に、消費できる電力計算をするために必要となります。
保存温度（Tstg） 破壊を生じない、非バイアス時（どこにも電圧を印加していないとき）の周囲温度です。デバイス単体か基板実装状態かは問いません。通電していないため、ジャンクションやチャネルの部分も周囲温度と同じ温度になります。
熱抵抗 熱の伝わりにくさのことで、単位電力あたりの、パッケージの上昇温度で表します（熱設計参照）。 パワー・デバイスで、消費できる電力計算をするために必要となります。
接合部−ケース間熱抵抗（θj-c）、**チャネル部−ケース間熱抵抗**（θch-c） デバイス内部のチップから、パッケージ（ケース）表面までの熱抵抗です。デバイスに放熱板を付る場合に使用する値です。ケース温度にチップ上昇温度を加えて、チップ温度を計算します。 パワー・デバイスなどで、消費できる電力計算をするために必要となります。
接合部−周囲空気間熱抵抗（θj-a）、**チャネル部−周囲空気間熱抵抗**（θch-a） デバイス内部のチップから周囲空気までの熱の伝わりにくさを表した値です。デバイスを自立で使用する場合に使います。動作周囲温度にチップ温度上昇を加えて、チップ温度を計算します。 パワー・デバイスなどで、消費できる電力計算をするために必要となります。
DC 特性 デジタル IC で、入出力信号が変化しない領域(DC 状態)での、ハイやロウの電圧レベルに対する電圧、電流特性です。出力電流は、出力信号が規定の電圧レベルを確保できる限界値です。 リーク電流は、MOS デバイスにおいて、ゲート絶縁膜やオフ状態のチャネルを通して流れる微少電流です。
AC 特性 デジタル IC で、変化する(交流)入出力信号の時間特性です。周期や幅、遷移時間、また信号間の遅延と設定時間／保持時間を規定しています。負荷条件と測定点は、これらの時間を規定するための条件です。 測定点は一般に、DC 特性の出力電圧と入力電圧となっています。負荷条件を越える過負荷では、時定数が大きくなって、AC 特性を満足しないうえ、入力側が CMOS なら貫通電流で劣化を促進させる恐れがあります。また容量の過負荷では、突入電流への対策回路がないと、劣化を促進させる恐れがあります。
容　量 MOS デバイスの入出力端子−GND 間、またはダイオードの両端子間の静電容量です。DC 特性とともに、出力の負荷条件への適合性を検証したり、スイッチング動作の高速性を判断するために規定しています。
推奨動作条件 期待する特性や動作が得られやすいように、推奨している条件です。パワー・デバイスなどでのディレーティングは、推奨条件内なら、絶対最大定格に対してもできます。また、デバイス選定時の目安となります。

規格の見方や注意点

Q1. デジタル回路を設計したいのですが、規格の見方や注意点が分かりません。

A1. インフォメーション資料「電気的特性の考え方　マイコン編」を参考にしてください。タイトルはマイコン編ですが、測定点電圧が 50%の製品もあることなどを除けば、基本的には他のデジタル製品でも適用できます。

Q2. 端子の入出力電流で、＋と－の表記はどういう意味ですか。

A2. ＋は流れ込む電流方向で、－は流れ出す電流方向です。信号の入出力方向ではありません。

	ハイ・レベル	ロウ・レベル
入力信号	＋(リーク)	－(リーク)
出力信号	－	＋



参照資料:「電気的特性の考え方　マイコン編」

絶対最大定格

Q1. 入力電圧が、一瞬だけ絶対最大定格を越えても、問題ないですか。

A1. 信頼性は保証できません。破壊する可能性もあります。ルネサス エレクトロニクスでは、長期的な信頼性を保証するために、耐圧設計をしていますが、そのような場合には、外部に必ず保護回路を設けてください。

Q2. 全損失(許容損失)を越えて、パワー・デバイスを使えますか。

A2. だめです。全損失を越えると、チップの接合部(ジャンクション)温度が、絶対最大定格を越えてしまいます。

Q3. 絶対最大定格は使用温度の上限が 80℃となっています。83℃で使用したいのですが問題ないでしょうか。

A3. いいえ、絶対最大定格は、いかなる場合であっても越えることができません。
また、絶対最大定格を越えた製品は、正常に動作していても内部で劣化が生じていることもあります。
少しでも絶対最大定格を越えた場合には、使用を中止して新しい製品にお取り替えください。

Q4. 電源電圧 V_{DD}=−0.3～+7.0(V)、入力電圧 V_{IN}=−0.3～V_{DD}+0.3(V)であれば、入力端子には−0.3～+7.3(V)まで印加していいのですね。

A4. いいえ、だめです。絶対最大定格の項目は、独立した定義になっています。この場合、電気的特性(DC 特性)が、たとえば V_{DD}=5V±10%なら、V_{IN}=－0.3～5.8(V)です。動作温度範囲も同様に独立条件です。
また、正常動作の保証値は、あくまでも電気的特性(DC 特性)の範囲内です。これを越えると、絶対最大定格に近づくほど、劣化が進みます。

Q5. 保存温度 Tstg は、電源電圧 V_{DD} が電気的特性(DC 特性)の範囲内の定義ですか。

A5. いいえ、保存温度は電源や入力端子に電圧のかからない非バイアス状態での定義です。

Q6. DC 特性と絶対最大定格の間は、どのような意味を持つ範囲ですか。

A6. DC 特性の範囲を越えると、正常動作が保証されません。なお、破壊に至る製品もありますが、一般的には絶対最大定格の範囲内であれば、瞬時的な信頼性のダメージを受けることはありません。
ただし、絶対最大定格に近づくほどストレスが大きくなり、製品寿命を縮めてしまう恐れがあります。

参照資料:「電気的特性の考え方　マイコン編」

電気的特性

Q1. 電気的特性（AC,DC 特性）の規格範囲を越えて使用したいので、実力値を教えてください。

A1. そのような特定はできません。また、電気的特性の規格範囲を越えた場合の動作は保証していません。
半導体デバイスの個々の特性には、製造ばらつきがあります。出荷検査では、規格値を保証範囲として、個々の特性がその範囲内であることを確認しているだけです。
なお、この範囲を少しでも越えると破壊する製品もあります。電気的特性の規格の範囲内でご使用ください。

Q2. 電気的特性（AC,DC 特性）の規格をわずかに越える範囲で使用したいのですが、なんとかなりませんか。

A2. 動作保証の対象は電気的特性の規格の範囲内で、これを越える場合は、お客様の判断でご評価ください。
規格を広げられる場合もありますが、可否は各製品の項目によります。
これは選別基準を厳しくして、良品の中からさらに適合する特性のデバイスを抽出することによって実現するものです。この選別規格を特殊規格と呼んでおり、専用にカスタマイズしたテスト・パターンと専用の選別ラインが必要となるため、製品価格は相応に高額になります。価格を問題とされない場合、まず特殊規格の対応ができるかどうかを確認する必要がありますので、販売店にご相談ください。

Q3. 推奨動作条件を越えても、問題ないですか。

A3. 推奨動作条件は、製品によって意味が異なりますので、内容でご判断ください。

(1) AC, DC 特性より狭い範囲の場合

AC,DC 特性の範囲内であれば、特に問題はありません。

(2) 他に AC, DC 特性として規定していない場合

AC,DC 特性と同じ意味です。この場合には、推奨動作条件を越えると、動作が保証されません。
絶対最大定格を越える範囲では、信頼性も保証されません。
なお、パワー・デバイスでは、絶対最大定格に対してディレーティングができることをご確認ください。
これらが満足できなければ、IC 選定を見直してください。

(3) 動作の誤差に関連する場合

通信のクロック・レートなど、動作精度に関連して規定している場合、推奨動作条件を越えると、動作が保証されません。

Q4. 同じデジタル IC の電源投入で、ある端子からロウ・レベルを出力するものと、ハイ・レベルを出力するものがあります。どちらか不良品ではないでしょうか。

A4. 製造の拡散工程でドーピングされる不純物の分子量や拡散形状で初期状態が変わり、これらは工程管理できるレベルのものではありません。つまり初期状態がハイ／ロウどちらの出力でも、不良品ではありません。
通常、初期状態の固定が必要な出力は、リセット動作で初期化するようになっています。リセット機能のあるデバイスの電源投入時には、必ずリセットをしてください。

Q5. 負荷条件が C_L＝100pF の出力端子なら、入力容量 Ci＝80pF のメモリ・モジュールは直結できますか。

A5. 直結は困難です。負荷容量 C_L は、負荷側の入力容量 Ci と駆動側の出力容量 Co、基板の配線容量の合計です。この場合、たとえば Co＝10pF なら Co＋Ci＝90pF で、10pF の余裕しかないため、基板の配線パターンの容量が加わると、過負荷となります。
接続の可否については、ハイ・レベルとロウ・レベルそれぞれについて、電圧と電流のドライブ能力も確認する必要があります。規格を満足しない場合、ドライバの挿入などが必要となります。

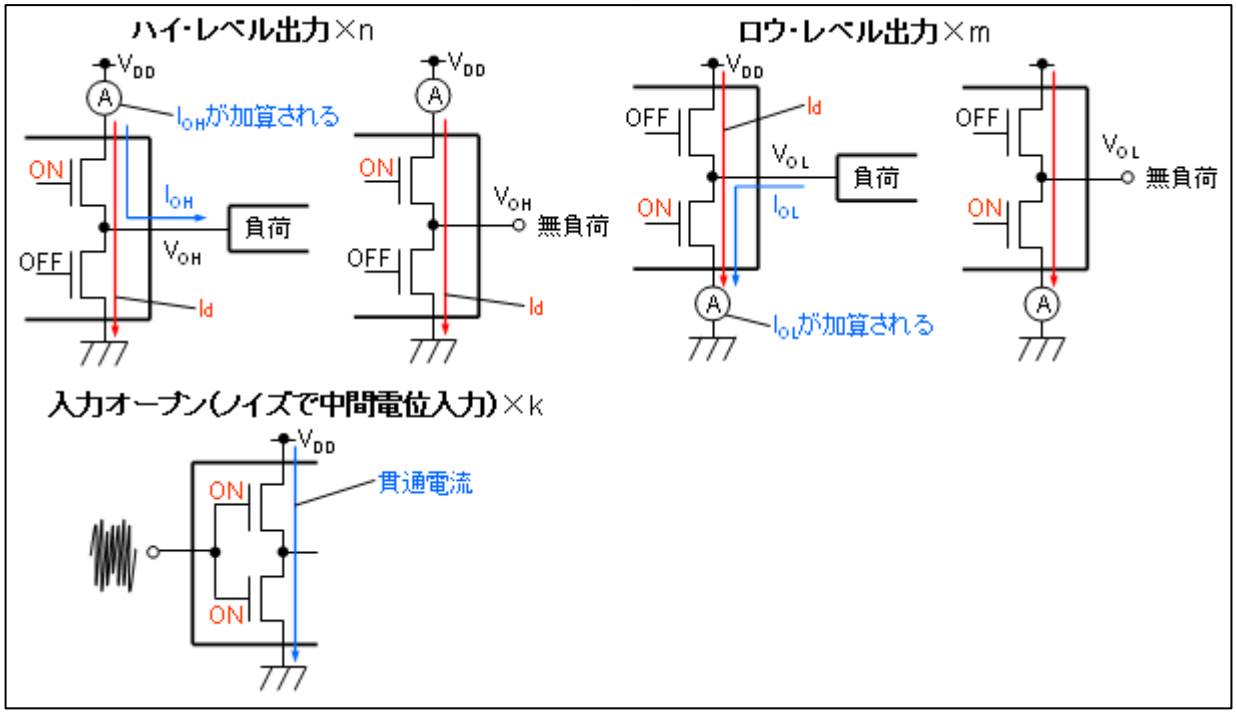
電圧については、プルアップ／プルダウンで対処できることもありますが、出力電流や立ち上がり／立ち下がり時間の規格を満足するような抵抗値を設定する必要があります。

Q6. デジタル IC で、それぞれ電源電流の規定はあるのですが、リセット入力中の電流値はどのくらいですか。

A6. リセット中という特定状態での規定はしておらず、通常動作の最大値以内であることが保証されるだけです。ただし入出力ポートを内蔵しているマイコンでは、リセットでポートが入力となるため、外部でポートの状態が固定されないと貫通電流が流れます。貫通電流は電源からグランドに流れるため、該当するポートすべての入力バッファの分だけ、余分に流れることになり、また電流値は特定できません。

Q7. 電源電流が規格を越えてしまいます。

A7. 電源電流の規格は、端子の出力電流を含みません。これは、負荷に依存して、単体デバイスで定義できないためです。負荷状態で電源やグランドに電流計を挿入すると、出力電流が加算されてしまいます。
また、入力がオープンでは貫通電流が流れますので、低消費電力モードなどでは、規格を越える可能性があります。出力がオープンで、入力がハイかロウに固定の状態が、電源電流の規格に相当する条件です。



Q8. デジタル IC の電源電流について、電気的特性に規定されている電圧や周波数と異なる条件では、どのように考えればいいですか。

A8. 一般に動作電圧範囲での電源電流は、電圧や周波数にだいたい比例します。
電流の要素としては、クロックが停止状態でも流れるリーク分と、クロック周波数に依存するスイッチング動作分があります。マイコンの STOP モードの特性から分かるように、リーク電流はけた違いに少ないので、全体の電源電流は周波数にだいたい比例します。たとえば 5V、10MHz で 10mA とすると、3V、5MHz の場合には、10×(3/5)×(5/10)=3 より、3mA 程度と考えられます。
ただし、レギュレータを内蔵したマイコンでは、電源電圧が変化してもレギュレータが内部回路(クロック・ジェネレータや LCD コントローラを除く)に一定電源を供給するので、電流はあまり変化しません。この場合は周波数だけ考慮してください。

Q9. デジタル出力の電流条件が、規格値と異なる場合の出力電圧は、どのように考えればいいですか。

A9. 段階的な電流値で規定している汎用ポートでは、オームの法則で類推(H 出力は V_{DD} からの降下、L 出力は GND からの上昇)してください。なお、マイコンの場合は、ホームページに特性グラフ(参考値)が掲載されている製品があります。

ただし、出力の電流条件(段階的な規定では最大値)は、最大負荷を想定していますので、それを越える電流は駆動させないでください。信頼性を損なう恐れがあります。

Q10. 発振子を接続しても、クロック発振回路の端子電圧が「入力電圧」の規格値にならず、微弱電圧なのですが、壊れているのでしょうか。

A10. いいえ、クロック発振回路の端子では、「入力電圧」の規格値は「クロック信号入力」に適用され、発振子接続の場合ではありません。なお、電圧や波形の観測でプローブをあてると、その容量などで発振動作に影響を与えてしまいます。

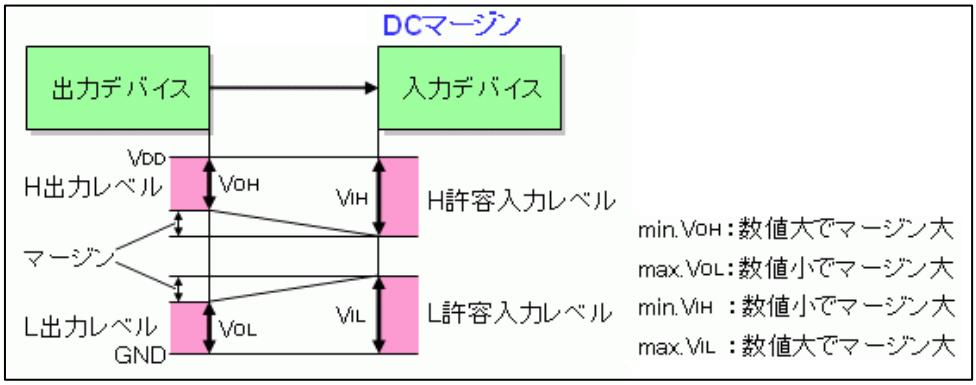
Q11. DC/AC 特性では min/max 規定をしていますが、あるデバイスで特性がふらつくということですか。

A11. いいえ、ちがいます。まず、入力と出力で意味が異なります。出力については、デバイスごとに固有の特性であり、製品ごとのばらつきを意味しています。入力については、許容値の範囲を意味します。

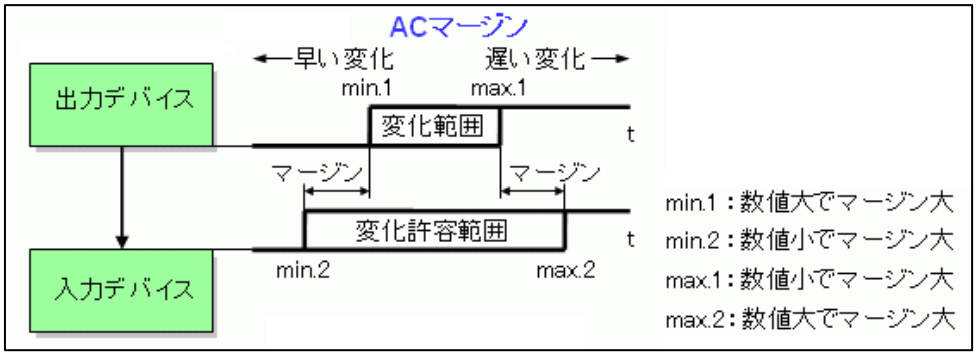
なお、あるデバイスについて、温度や電源／入力の電圧によって、これらの特性は変化しますが、動作保証範囲の条件なら、min/max 規定の範囲を越えることはありません。

Q12. DC/AC 特性の min と max について、どのようにとらえればいいのですか。

A12. それぞれ、出力側と入力側の相対値で、DC マージンと AC マージンが決まります。つまり、出力側の特性範囲が入力側を包含している必要があり、マージンが大きいほどノイズの影響による誤動作の可能性が低い良好なインタフェースといえます。このため、ある信号の接続を考えると、min/max 値の大小が望ましい方向は、入力側と出力側で逆になります。



DC マージンについては、ハイ・レベルで問題となることが多く、バッファ挿入や抵抗によるプルアップが一般的な対策です（AC マージンや電流駆動能力の確認が必要）。



関連資料：「電気的特性の考え方　マイコン編」

周囲温度

Q1. 周囲温度はどのように定義しているのですか。

A1. 当社では熱平衡状態を前提として、JEDEC を基に次のように定義しています。

無風の場合、基板を水平にします。デバイス中央の直下で、基板裏面から25mm 離れた位置を測定点とした温度とします。

強制空冷の場合、基板を水平か垂直にします。基板と水平の風向きで流速を 2m/s 以上とし、基板裏面から25mm 離れ、基板より風上 100～150mm の位置（デバイスの斜めうしろ方向）を測定点とした温度とします。

JEDEC の参照規定：JESD51-2（無風）、JESD51-6（流風）

セット・レベルの熱設計では、温度分布や筐体のサイズ、放熱特性などを考慮していただく必要があります。また、精度の高い熱設計は困難なので、余裕のある条件設定が好ましいといえます。

★

インピーダンス

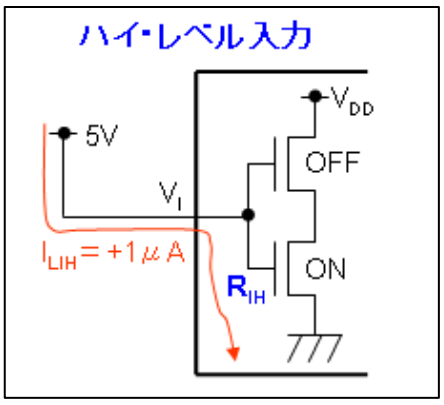
Q1. 入力や出力のインピーダンスを特性に表記している製品はほとんどありませんが、規定はないのですか。

A1. 半導体は電圧などによってインピーダンスが変化するので、一般に規定していません。実部について、規格で示されている特定条件では、オームの法則で算出できます。たとえば、CMOS の入出力について考えてみましょう。

入力については、V_I＝5Vのときにハイ・レベル入力リーク電流I_{LIH}＝+1 μA（max.）なら、次のとおりです。

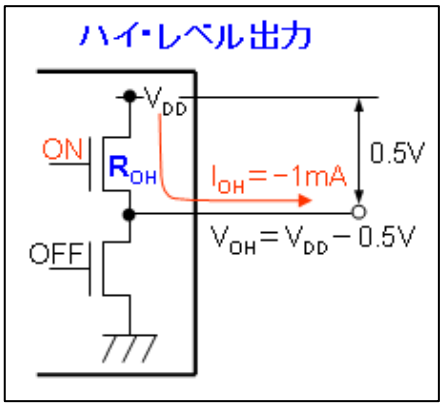
$R_I(min.)=5/(1\times10^{-6})=5(M\Omega)$

このような高抵抗になるのは、入力ゲートが酸化膜で絶縁されているためで、MOS入力の状態をハイ・インピーダンス(Hi-Z)と呼ぶ理由です。なお、一般にロウ・レベル入力リーク電流I_{LIL}は、V_I＝0Vの条件で、絶対値がI_{LIH}と同じなので、ロウ・レベルについても同じインピーダンスになります。



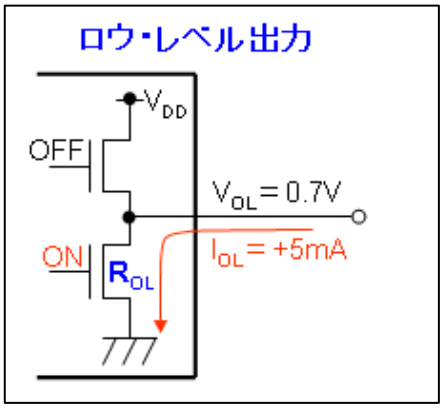
次に出力について、I_{OH}＝−1mAのときにハイ・レベル出力電圧V_{OH}＝（V_{DD}−0.5V）(min.)なら、V_{DD}からの降下電圧は0.5V以下ということになるので、Pチャネル（V_{DD}側）のオン抵抗は、次のとおりです。

$R_{OH}(max.)=0.5/(1\times10^{-3})=500(\Omega)$



また、I_{OL}＝+5mAのときにロウ・レベル出力電圧V_{OL}＝0.7V(max.)なら、Nチャネル（GND側）のオン抵抗は、次のとおりです。

$R_{OL}(max.)=0.7/(5\times10^{-3})=140(\Omega)$



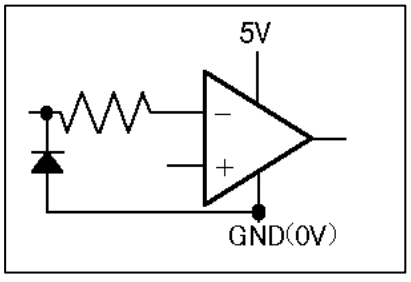
2.2 規格外の電圧印加

μPC177：入力に−0.5V のパルスが入る

Q1. μPC177 を+5 単電源で使っています。動作中、入力端子に約−0.5V のパルスが入ります。
データ・シートでは絶対最大定格が−0.3V～+36V となっていますが、いいでしょうか。

A1. だめです。絶対最大定格は、絶対に越えてはなりません。入力に保護回路を設けることをお勧めします。

通常、IC の入力端子には寄生ダイオードが生じます。入力端子に電源より高い(またはグランドより低い)電圧を引加すると、入力端子から寄生ダイオードを通して、電源(またはグランド)に電流が流れます。
寄生ダイオードに流れる電流が 5mA 程度までは、破壊に至ることはないと思われます。しかし動作が不安定になる恐れがありますので、いかなる場合でも絶対最大定格を越えないように、ご注意ください。
入力にマイナス電圧が引加される恐れがある場合は、右図のようにダイオードで保護するのが一般的です。このダイオードには、順方向電圧の小さいショットキ・バリア・ダイオードをお勧めします。
逆に電源電圧を越える電圧が引加される場合には、入力と電源の間にダイオードを挿入してください。



μPC494：端子に電源電圧以上の電圧を印加

Q1. 一般に、IC の端子に電源電圧以上の電圧を印加できませんが、μPC494 の規格では、C1、C2 出力に 41V を印加できると思えます。Vcc＝12V で、これらの端子に電源電圧以上の電圧を印加してもいいのですか。

A1. いいえ、電源電圧の範囲内でご利用ください。電源電圧以上の電圧が加わっても、すぐには破壊しないように設計してあります。しかし、このような状態での検査はしていません。

2.3 信頼性

半導体の寿命

Q1. 半導体には寿命がありますか。

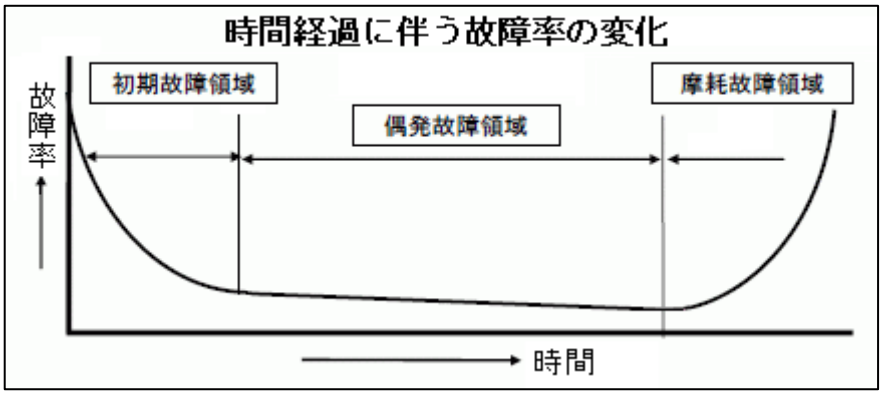
A1. 半導体デバイスの耐用年数は、デバイス構成素子としての配線、酸化膜およびトランジスタなどの耐久性によって決まります。
耐久性は、信頼性設計(下記資料の「第 1 章　1.3.2(2)」参照)により、半導体デバイスに造り込まれます。
耐用寿命は標準品の場合、推奨条件内でのご使用で少なくとも 10 年持つように設計されており、数十年と安定した動作をすることもあります。なお、寿命は単一製品そのもののライフです。

★ ただしフォトカプラでは、CTR半減期を寿命とする考え方が適用されます。定格内であっても、温度と電流の影響をかなり受けます。

参照資料：半導体　品質/信頼性ハンドブック

Q2. 10 年という寿命設計の根拠はどういうものですか。

A2. お客様のご要求に基づき、デバイスの磨耗故障領域が 10 年以降となるようにデバイス設計を行っています。



関連 FAQ：バスタブ・カーブ

Q3. 半導体デバイスの信頼性とはどのようなことですか。

A3. JIS によると、信頼性を次のように定義しています。
「アイテムが与えられた条件で規定の期間中、要求された機能を果たすことができる性質」
（JIS Z 8115「信頼性用語」）
簡単に言えば、「ある製品が使用期間中、故障しないで機能する性質」であり、製品の時間的品質を表しています。この信頼性をさらに定量的に計る尺度として、信頼度、故障率、平均故障寿命などがあります。

参照資料：半導体　品質/信頼性ハンドブック　第2章

Q4. 半導体の寿命は温度で変わりますか。

A4. はい、温度と電気的な特性の劣化は強く関係いたします。
一般に、温度が上がると劣化が促進される傾向にあります。
定格を越える温度では急激に劣化します。高温でご使用になる場合はご注意ください。

参照資料: 半導体　品質/信頼性ハンドブック

Q5. 半導体製品を製造直後に使用する場合と、一般室内で数年間保管後に使用する場合の違いはありますか。

A5. 通電すると劣化は進行します。
プラスチック・パッケージで実装していない場合、使用前に脱水のためのベーキングをしてください。
また、使用前にリード線の錆がないか、半田濡れ性はどうかなどのチェックをしてください。
一般に、製品の保証期間は、出荷後 2 年となっています。

参照資料: 半導体　品質/信頼性ハンドブック

Q6. X 線での透視検査は、半導体の寿命に影響を与えますか。

A6. 当社の半導体製品について、通常の X 線による透視検査は問題ありません。
「半導体　品質/信頼性ハンドブック」に記述のとおり、X 線透視検査は「非ストレス」のスクリーニング項目になっていますし、故障解析でも実施をします。

★ Q7. フォトカプラの寿命データ(経時変化データ)はありますか。

A7. フォトカプラでは、データ・シートに CTR 経時変化として掲載しています。
フォトカプラの寿命については、技術解説「汎用フォトカプラの電流伝達率(CTR)」(4 項)をご参照ください。

関連項目: CTR とは？

故障率

Q1. 製品ごとの故障率は、どこかに掲載されていますか。

A1. 温度や電源電圧によって条件が異なるため、製品ごとの故障率(fit)の掲載はしていません。
インフォメーション資料「半導体故障率」の第 2 章に算出手順、第 5 章に製品ごとの基礎故障率(λb)と活性化エネルギー(Ea)を掲載していますので、これらを元にご算出ください。
なお、この資料に掲載されていない製品のパラメータについては、製品の購入先を通じてご確認願います。

故障モード

Q1. 半導体製品が内部破壊した場合、どのような動作をしますか？

A1. 破壊の故障モードには、ショート、オープン、オーミック(抵抗性)の 3 通りありますが、いずれになるかを事前に特定することはできません。また、これらの故障モードと他の内部回路への影響は特定できません。

信頼性の推定

Q1. LnH(t)というのは、どういう値ですか。

A1. ワイブル分布によるハザード確率の推定パラメータです。Ln は自然対数、H(t)はワイブル分布を表します。
概要は、半導体 品質／信頼性ハンドブックの「付録 信頼性の推定」をご参照ください。
ワイブル分布の詳細については、日本規格協会発行の「ワイブル確率紙の使い方」等をご参照ください。

信頼性試験

Q1. 静電耐量を確認する ESD 試験がありますが、ごく短時間ならそのような高電圧印加は問題ないですか。

A1. いいえ、ESD 試験はあくまでも瞬時的な端子破壊の程度を確認するものであり、その後の製品の信頼性は保証できません。試験時に端子破壊がなくても、短期的に動作不具合が発生する可能性があります。
量産品に対して、絶対最大定格を越える電圧は印加しないでください。

外　観

Q1. パッケージ本体に近い端子部分で、外装されず地金の出ているものが納入されました。不良品ですか。

A1. パッケージによって、そのような製品がありますので、不良品ではありません。
ディスクリートやリニア IC の端子挿入タイプ（メッキでなく浸漬品）は、ストッパ（細い−太いの境）部分まで外装されていれば良品です。
また、3 ピン・ミニモールド(SC−59)では、リード付け根付近が少し太くなっている製品では、その付近にメッキされていない部分があります。たとえば 2SA812A などが該当します（2SA812 は全メッキ）。

使用環境

Q1. 旧 NEC エレクトロニクスの半導体製品は、耐放射線設計をされていますか。

A1. いいえ、一般に耐放射線設計はしておりません。セットでの被爆遮蔽を行ってください。
概要は、半導体 品質／信頼性ハンドブックで「半導体デバイスのご使用にあたって」の「使用環境に関する注意事項」をご参照ください。

Q2. 2000m を越えるような高山では、半導体デバイスの信頼性や特性に影響がありますか。

A2. 人間が快適に生活できる高度範囲なら、保存温度や動作温度の範囲を守っていただけば、そのような問題は考えられません。気圧についても、数千mの標高なら特に問題とはなりません。
ただし、高度が高くなるほど、高速中性子線量が増えるので、ソフト・エラーを心配される場合は、使用する高度と使用デバイスでの、ソフト・エラー評価と対策が必要となることがあります。

関連情報

Q1. ここに掲載しているFAQ以外の情報はないでしょうか。

A1. 半導体 品質/信頼性ハンドブックの「付録 D Q&A 集」をご参照ください。

2.4 難燃性

難燃グレード

Q1. 旧 NEC エレクトロニクス製の半導体製品は、UL 難燃グレード（UL94 V-0）を使用していますか。

A1. はい、UL 規格の難燃グレード「UL94 V-0」の認証を得ています。ただし、フォト・ディテクタ IC は除きます。

難燃材

Q1. 旧 NEC エレクトロニクス製の半導体製品は、パッケージの樹脂に無機リン系の難燃剤を使用していますか。

A1. いいえ、使用していませんし、今後も使用する予定はありません。

2.5 サンプル

サンプルの種類

Q1. 製品サンプルには、どのようなレベルのものがあるのでしょうか。

A1. サンプルには、開発時のものと、量産時のものがあります。
開発時のサンプルは、次のように区別されます。

DS(Design Sample)
動作検証を終える前のレベルで、メカ・サンプルと呼ばれることもあります。
最初にできたものはファースト・サンプルと呼ばれます。

ES(Engineering Sample)
ある程度、動作検証をしたレベルですが、不具合の内在することもあります。

CS(Commercial Sample)
動作検証と信頼性評価を完了したレベルで、量産品と同等です。

量産を開始したあとは、お客様のご要望に応じて、評価サンプルを提供します。
なお、お客様が信頼性試験をされる場合、**QT(Quality Test)サンプル**というものを提出します。

サンプルの入手

Q1. サンプルを入手するには、どうすればいいですか。

A1. 当社は直販をしておりませんので、サンプルについても通常の製品ご購入先にご相談ください。
開発中のサンプルは販社しか対応できませんが、量産品を少量お求めであれば、WEB ショップでご購入いただけるものもあります。

3. 設　計

3.1 1ピン・マーク

μPD780958

Q1. 端子接続図の1ピンの位置を示すマークが3つあります。本物はどれですか。

A1. 写真の一番下が1ピン・マークです。

左右の2つは、モールド(金型)に樹脂を注入する際にできるくぼみです。

1ピン・インデクスは二重丸で、他の 2 つとは異なる形状です。

小さくて分かりにくければ、ルーペなどで拡大してご確認ください。

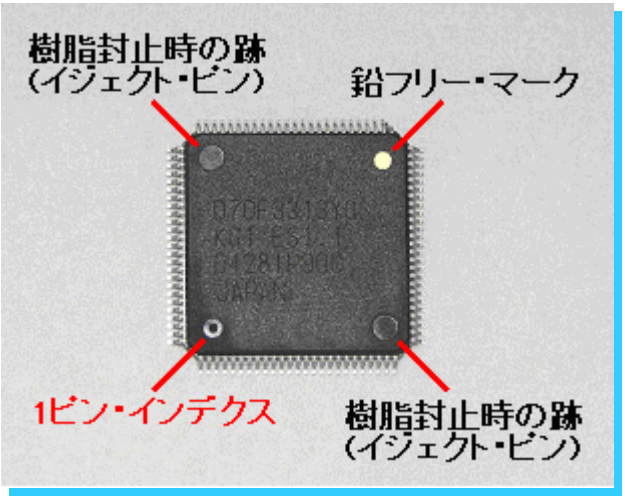
なお、製品によっては、樹脂注入でできるくぼみが写真とは違う位置にあるものもあり、1ピン・インデクスのくぼみを重ねているものなどもあります。



捺印と刻印

Q1. パッケージ表面に、円形の白い捺印と窪みのある刻印がありますが、どれが1ピンのマークですか。

A1. 二重円の刻印が1ピン・インデクスです。塗料やレーザによる捺印は、鉛フリーのマークです。



3.2 実装・組み立て

はんだ付け

Q1. N13T2 に関して、はんだ付け推奨条件はありますか。

A1. N13T2 は THD(リード線タイプ)と同様にお考えください。

具体的には、下記資料に記載していますので参考にしてください。

参考資料: 半導体デバイス実装マニュアル 3.3.3 各はんだ付け方式の推奨温度プロファイル

Q2. はんだ付け推奨条件で、実装回数が 2 回の場合、実装品をはがして再実装できるということですか。

A2. いいえ、はがすことは想定していません。
実装回数が 2 回というのは、通常の工程で実装機にかけられる回数が 2 回ということです。たとえば、基板の表面へ実装したあと、別のデバイスを裏面へ実装する工程に対応できるということです。この場合、実装回数が 1 回に制限されているデバイスは、後者の裏面実装に適用する必要があります。

Q3. はんだごてで、表面実装品を実装する場合の推奨条件はありますか。

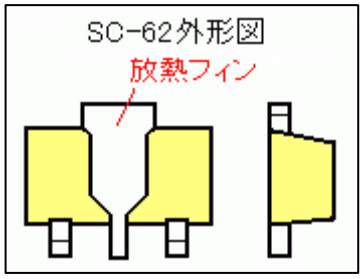
A3. 表面実装品の端子部分加熱は、熱が直接デバイスに加わるため、一部の製品を除いてレーザ加熱を推奨しています。なお、一辺あたり 3 秒としています。
はんだごてで、この条件を満足することが困難な場合、個別に購入先を通じてご相談ください。

Q4. はんだディップ方式の可否はどのように判断すればいいですか。

A4. 基板ごと、はんだ槽に浸漬させる方式は推奨していません。
ウエーブ・ソルダリングについては、SOJ や BGA などはんだの浸透などに問題があるパッケージや、ブリッジのおそれがあるリード・ピッチ 0.65mm 未満の表面実装品では推奨していません。
なお、端子外装にビスマスなどはんだ以外の成分をふくむ鉛フリー品では、全面的に推奨していません。

Q5. 表面実装品のフローはんだ付け（ウエーブ・ソルダリング）は可能ですか。

A5. SC-59 や SOP など放熱フィンがないパッケージでは可能です。SC-62 や TO-252、TO-220SMD など裏面に放熱フィンがあるものは、フィンと基板の接続が不十分になる可能性があるので、推奨していません。



プリベーク

Q1. 製品の"Moisture sensitivity level"を知りたいのですが、どこかに掲載されていますか。

A1. 旧 NEC エレクトロニクスの製品は、J-STD-020 による MSL の定義はしていません。
「4.3.1 耐熱温度プロファイル」の「(2)記号の意味」で、「ベーキング時間」と「ドライパック開封後の保管日数」を定義しています。

Q2. 半田付け推奨条件に「プリベーク」という表記のある製品がありますが、これはどういうことですか。

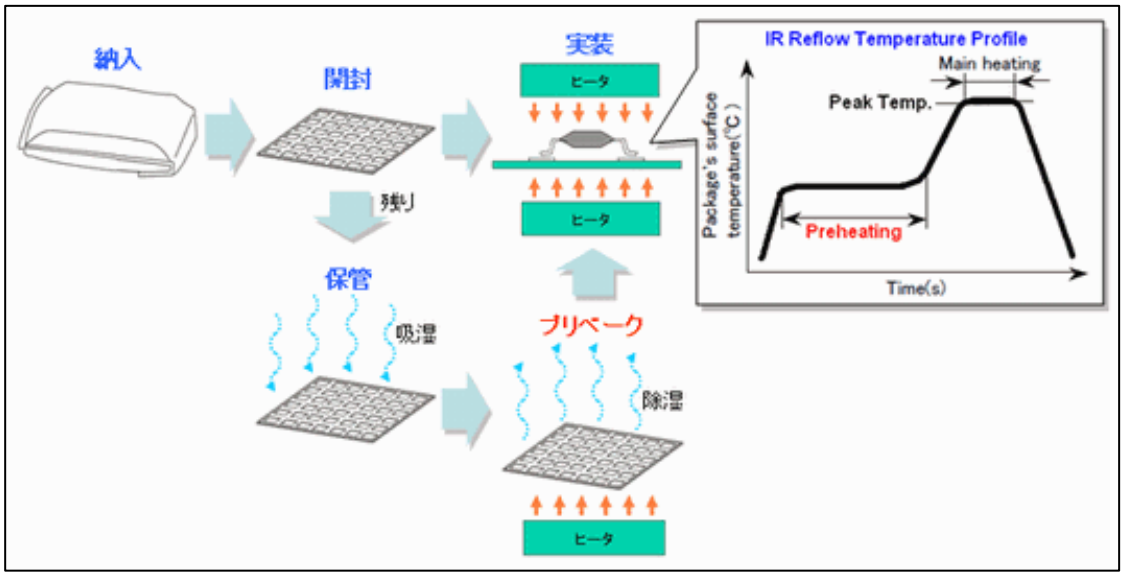
A2. プラスチック・パッケージの製品を放置していると、空気中の水分を吸収してしまいます。これを赤外線リフロや VPS で全体加熱して基板に実装すると、200℃を越える高温にさらされるため、内部の水分が一気に気化して膨張し、パッケージ・クラックを発生させる恐れがあります。
プリベークとは、これを回避するために実装直前に行う処理で、水分を放出するためのベーキング(加熱処理)のことです。
詳細については、半導体デバイス実装マニュアルの「4.2.3 ベーク処理」をご参照ください。
プリベークが必要となる製品は、防湿をするためのアルミ・コーティングしたドライパックに封入して出荷しています。ドライパックを開封後は、必要に応じて、各製品で推奨されているプリベークを行ってください。

Q3. 製品によって、プリベークの条件が異なっていたり、不要になっているのはなぜですか。

A3. パッケージ・サイズや内部のチップ・サイズが異なり、吸湿特性やクラックの発生しやすさが違うためです。

Q4. プリヒートとプリベークは違うのですか。

A4. プリヒートは実装時の予熱のことで、実装機で必ず行います。プリベークは実装前に水分を抜くための焼入れ(ベーキング)のことで、開封直後や吸湿管理の不要な製品では行いません。



実装の詳細については、半導体デバイス実装マニュアルをご参照ください。

洗　浄

Q1. 洗浄条件について教えてください。

A1. 洗浄に関しては、下記資料に記載していますので参考にしてください。

参考資料: 半導体デバイス実装マニュアル「3.4 洗浄プロセス」

Q2. RA タイプのロジン系フラックスは、腐食性が強いと言われていますが、無洗浄フラックスとして推奨しているフラックス・メーカもあります。RA タイプに種類があるのですか。

A2. ロジン系フラックスは、活性力の強い順に RA,RMA,R の 3 種に大別されます。
これらのうち RA、RMA は、ハロゲンを含んではんだ付け性を向上させている場合があります。
特に RA は、実装後にハロゲンが残留し、腐食、デンドライト等による信頼性劣化を引き起こします。

しかし最近は、ハロゲン等の活性元素を含まず、活性力を向上させて、無洗浄で信頼性の高いフラックスが多く開発されています。
フラックスの信頼性を示す指標の一つとして、塩素含有量/ハロゲン含有量があり、フラックスやペーストの製品特性として記載されていますので、お使いになるはんだペーストやフラックスの資料で御確認いただくか、メーカへお問い合わせください。

無洗浄の記載があるペーストやフラックスでも、半導体製品の品質レベルによっては問題となる場合がありますので、ご使用になる製品の品質レベルに応じた処置をお勧めします。

鉛フリー品

Q1. 鉛フリー品の実装条件は、従来品と違うのですか。

A1. 一般的に違いますが、従来品でも同条件をクリアしているものがあります。
鉛フリー品の実装条件については、半導体鉛フリー製品　パンフレットを参照してください。
なお、各製品の実装条件は、データ・シートを参照するか、販売店にご確認ください。

Q2. 鉛含有品と鉛フリー品を混在して基板に実装したいのですが、注意事項はありますか。

A2. (1) 鉛含有ペーストでの実装
鉛フリー品に対して、鉛含有品ほどの接合強度が得られない傾向があります。
お客様での用途や求められる品質等によって、問題であるかどうかの判断は異なります。

(2) 鉛フリー・ペーストでの実装
鉛含有品に対して、基板の反り等による接合部の剥離の可能性があります。
対策としては、フロー実装等の再加熱による該当部品の温度上昇を抑制することや、基板の反りを抑える治工具の使用などが挙げられます。

詳細については、半導体鉛フリー製品　パンフレットをご参照ください。

Q3. 従来はウエーブ・ソルダリングに対応していた製品で、鉛フリー化によって対応できなくなっているものがあるのはなぜですか。

A3. Sn-Bi めっき品などは、ビスマスなどはんだ以外の成分で、お客様の実装機を汚染するおそれがあるため、ウエーブ・ソルダリングを推奨していません。

ランド・パターン

Q1. IC のデータ・シートには、表面実装型のパッケージを実装する基板のランド・パターンが掲載されていませんが、確認する方法はありますか。

A1. IC パッケージのホームページの以下の場所に掲載しています。
パッケージ情報＞IC パッケージ一覧
目的パッケージの「Land pattern」のチェック・マークをクリックすれば、パターン図を閲覧できます。
なお、光デバイスと RF/マイクロ波デバイスについては、下記ページに掲載の各資料をご参照ください。

フォトカプラ　：「NEPOC セレクション・ガイド」の「10. 4 プリント基板の実装」
http://www2.renesas.com/opto/ja/document.html#Photocoupler

RF/マイクロ波デバイス：「高周波およびマイクロ波デバイス・パッケージ・ラインナップ」
http://www2.renesas.com/microwave/ja/document.html#pkg
CAD データ：http://www2.renesas.com/microwave/ja/mountpad.html

リード曲げ

Q1. THD(Through-hole device)のリード線を折り曲げて使っていいですか。

A1. リード線を折り曲げる場合は、リード根元に引張り応力が加わらないようにしてください。そのためには、冶具と本体が接触しないよう(間隔を 2mm 程度)に、本体の根元側を冶具でしっかり固定してください。その他、折り曲げは 90 度、1 回としてください。また、リード間隔を広げるような成形は絶対にしないでください。

防湿コーティング

Q1. 基板実装後、防湿のためのコーティング処理に関する注意事項や推奨内容はありますか。

A1. 当社半導体製品のパッケージは、エポキシ樹脂を使用しています。基板実装後のコーティング剤は、エポキシ樹脂の熱膨張係数と同等のものとして、機械的ストレスを回避してください。熱的，機械的ストレスの回避策としては、外装コート前にバッファ材(ラバー状のもの)を塗布することが一般的です。

注意 RF/マイクロ波デバイスでは、コーティング処理で特性劣化を生じる場合があります。

静電気対策

Q1. 半導体デバイスは静電気に弱いと聞きましたが、実装時にはどのような注意をすればいいでしょうか。

A1. 出荷梱包での包装材やトレイなどはすべて導電性で、静電気の影響を受けないようになっています。
開封後の実装では、すべての工程において、作業者や冶具などの帯電防止をしてください。
デバイスの端子に静電気を印加してしまうと、その時点で破壊にいたらなくても、重大なダメージによって、市場出荷後に不良となる可能性があります。

参考資料：「静電気放電(ESD)破壊対策ガイド」

メタル・マスク

Q1. デバイスをリフロ実装するときに、はんだペーストの塗布で使用するメタル・マスクの情報はありますか。

A1. いいえ、デバイス・メーカでは、メタル・マスクについての情報を管理していません。その開口寸法や厚さは、使用されるはんだペーストや実装機の特性などに応じて、実装工程で決定されるものです。

★ 幾何公差

Q1. 外形図に用いられている記号 ⌓ は、なにを表すのですか。

A1. 取り付け面 S(Seating plane)に対する端子最下面の均一性(コプラナリティ)です。詳細は、端子最下面の均一性 y をご参照ください。
以前は端子の平坦度 ⏥ で表していましたが、BGA での表現には適さないため、現在はこのような表現に統一しています。垂直方向の公差という点では、同じ意味です。

3.3 未使用端子の処理

一般的な未使用端子処理

Q1. 使わない端子はどうすれば良いのでしょうか。

A1. (1) MOS 型半導体(μPDxxx など)の入力端子

オープンにすることはできません。入力端子は、電源かグランドへ接続してください(内部でプルアップかプルダウンされている場合を除く)。

(2) 出力端子

通常オープンで構いません。ただしオープン・ドレーン形式の出力端子は、静電気による破壊防止のためトランジスタがオンした状態の電位に接続しておくことをお勧めします。

(3) 入出力端子(データ・バスやアドレス・バス、I/O ポートなど)

機能によって処理の方法が異なります。
出力モードに設定可能な場合は、出力モードにします。この場合は出力端子と同様です。
入力になる可能性がある場合は、適当な抵抗でプルアップ(またはプルダウン)してください。

なお、RF/マイクロ波/表示デバイスでは、製品によって処理方法が異なりますので、販売員にご相談ください。ただし、多チャネル内蔵のフォトカプラは、未使用チャネルの全端子をグランドなどに接続して同電位にするか、オープンでもかまいません。

Q2. NC 端子は、何も接続してはいけないという端子ですか。IC 端子との違いが分かりません。

A2. NC端子は、内部接続していない端子です。ただし信号線などを接続すると、内部のノイズ環境が悪くなりますので、何も接続しないか、電源／グランドに直接接続することを推奨します。
これに対して、IC端子は内部接続されているため、端子の状態が内部動作を定義したり、信号が出力される可能性があります。指示どおりの処理をしてください。

NC : Non-Connection(またはNo Connection)
IC : Internal Connection(またはInternally Connected)

★ 注意 旧来の製品には、例外的にNC端子が内部接続されているものがあります。
従来設計でのNC端子の処理で問題がなければ、設計変更は不要です。
新規設計では念のため、何も接続しないでください。

なお、RF/マイクロ波/表示デバイスでは、製品によって処理方法が異なりますので、販売員にご相談ください。

Q3. マイコンの入出力ポートでは、未使用端子の処理を入力時(抵抗でプルアップ／プルダウン)と出力時(オープン)に分けて推奨していますが、どちらでもいいですか。また、入力時はなぜ抵抗挿入が必要ですか。

A3. 入力と出力は、どちらでもけっこうです。ただし、入出力ポートはリセットで入力になるので、そのまま抵抗でプルアップ／プルダウンするのが、もっとも好ましい処理です。
部品点数の削減を望まれる場合、出力に設定することをお勧めしています。この場合、出力に設定するまでは貫通電流が流れる可能性がありますが、中間電位での固定ではなく、ノイズによるふらつき状態なので、継続的には流れません。リセット直後、すぐ出力に設定すれば、劣化などの問題はないと考えられます。

なお、入力のままにする場合でも、プログラムの暴走やノイズの影響で出力になる可能性を考慮して、外部抵抗でのプルアップ／プルダウンを推奨しています（内部プルアップが設定できる場合でも、誤動作時にその設定の保持が保証されません）。**電源やグランドに直結できるのは、出力機能のない入力専用ポートです。**

ただし、長年にわたって高信頼性を維持しなければならないシステムなどでは、入力専用ポートでも、抵抗を挿入することが最善の処理です。ごくまれなケースと言えますが、もし製品劣化で、ゲート入力に電流が流れるようになると、システム全体の電流が引っ張られ、システム・ダウンにいたることも想定できるからです。

Q4. 出力専用の汎用ポートは、未使用時にオープンを推奨されていますが、ハイ出力設定で電源、またはロウ出力設定でグランドに接続していいですか。また問題があれば、抵抗挿入で回避できますか。

A4. 通常状態では問題ありません。ただし、信号や電源・グランドのノイズ等の影響で、出力レベルが反転する可能性がないわけではありません。これによりショート状態となって、瞬時破壊にいたらなくても、劣化促進につながる恐れがあります。プルアップ／プルダウンの抵抗があればショートは回避できますが、電流が流れます。これを問題視されなければ、特に問題はありません。

コンパレータの空き端子処理

Q1. μPC339 の入力にアナログ・スイッチを使っており、切り替えのタイミングでハイ・インピーダンス状態が存在します。入力端子が一瞬オープン状態になりますが、いいでしょうか。

A1. 破壊しませんが、入力には Ib が流れるので、オープンにするとハイ・レベルになる可能性があります。
お尋ねの状態では、切り替え時間が短いと推測されるので問題はないと思いますが、この状態が動作に影響するかどうかは、使用状態によって異なります。お客様の使用条件で問題がないことをご確認ください。

未使用端子の処理

Q1. 2 回路入り MOS FET の 1 回路を未使用にする場合、空きピン処理は必要ですか。

A1. 特に、未使用の空きピン処理は必要ありません。強電界などの環境でご使用の場合は、空きピンをすべてまとめて GND に接続すれば、使用している素子への影響を確実に避けられます。

プルアップ／プルダウン抵抗の共用

Q1. 「個別に抵抗を介して」と記述している未使用端子をまとめて、プルアップ／プルダウンしていいですか。

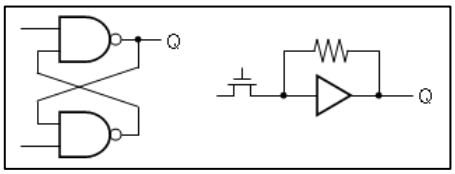
A1. だめです。この記述がある端子は、オープン・コレクタやオープン・ドレーンになっていない出力を含む入出力端子です。このため、出力の衝突を生じる可能性がありますので、ワイヤード OR 接続はできません。

3.4 初期状態

電源投入時の初期状態

Q1. IC のドキュメントには、電源投入時の初期状態が不定との注意書きがありますが、なぜ不定なのですか。

A1. 端子入力だけで論理の確定するロジック IC では、そのようなことはありません。内部にレジスタやカウンタ、SRAM などを内蔵する場合、それらの各ビットの等価出力は下図のような構成となっており、電源投入時の特性（Q が "1" か "0" か）が、製造条件や動作条件(電圧・温度など)によって異なります。リセット時の状態が定義されているビットについては、リセット動作で論理的に Q を所定の状態とするよう操作します。



3.5 熱設計

考え方の基礎

Q1. 半導体デバイスの熱設計について、基本的な考え方を教えてください。

A1. 半導体デバイスの消費する電力は熱に変わり、温度上昇の原因となります。
半導体デバイスが、正常に動作する温度（周囲温度や内部のジャンクション温度で規定）には上限があります。それを越えると、正常に動作しなくなったり、破壊したりします。そのため、発生熱をうまく放出させて、温度上昇を抑える必要があります。大電力を消費させたり、動作温度を高くする場合、熱設計が必須です。

この、放熱を検討するとき、**熱抵抗**という考え方をします。これは、熱の伝わり方は、電気の伝わり方と対比して考えることができるからです。詳細は省きますが、電流を熱＝電力、電圧を温度、抵抗を熱抵抗に置き換えれば、次のように電気の計算を適用できます。

電位差　＝　抵抗　×　電流
　↓　　↓　　↓
温度差　＝　**熱抵抗**　×　熱(電力)
熱抵抗　＝　温度差　／　熱(電力)

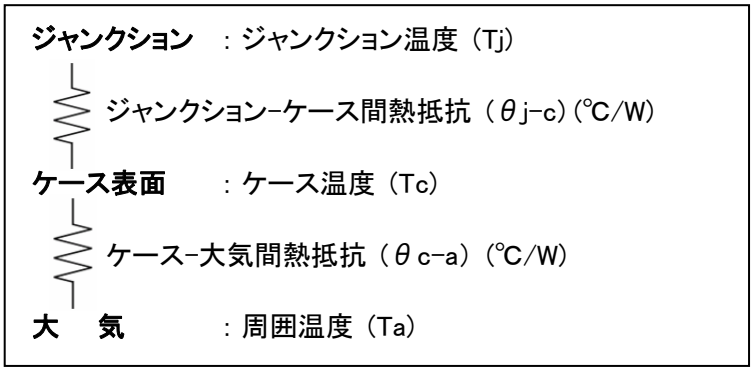
このことから、熱抵抗の単位は「℃/W」となります。また記号として θ を用います。どことどこの間の熱抵抗かを添え字で示します(θj-a:ジャンクションと大気間の熱抵抗)。

注意　熱の場合には、電気ほど良導体と絶縁体(断熱体)の差が大きくありません。
したがって、熱抵抗を使った細かな計算はあまり意味がありません。

モデリング

Q1. 熱設計は、どのようなモデリングに基づいているのですか。

A1. 半導体デバイスで電力を消費（＝熱を発生）している部分はジャンクション(MOS ではチャネル)です。
ここで発生した熱は、デバイスの内部構造を伝わって、デバイス表面（ケース）に来ます。
そこで、周囲の大気に放出されます。つまり、熱を発生しているジャンクションから、最終的に熱を吸収する大気までには、2 つの段階（熱抵抗)が直列に入ります。



各部の熱抵抗

Q1. 熱抵抗はどの部分に生じるのですか。

A1. ジャンクションとケース間の熱抵抗（熱の伝わりにくさ)は、デバイスの構造によって決まります。したがって、この部分の熱抵抗は固定値として扱うことになります。
ケースと大気間の熱抵抗は実装条件により大きく変化します。
放熱器を付けることは、ケース表面から放熱器を経由した放熱ルートを追加することになります。これは、デバイス本来のケース−大気間の熱抵抗と並列に、放熱器を経由した熱抵抗が入ると考えます。抵抗の並列接続なので、ケースと大気間の熱抵抗を放熱器の分だけ小さくすることができます。

ジャンクション温度と熱抵抗

Q1. ジャンクション温度と熱抵抗の関係を教えてください。

A1. 実際にデバイスを使う場合には、ジャンクション温度と熱の伝わりにくさを示す熱抵抗との関係が重要になります。ジャンクション温度（Tj）は、次の式で表すことができます。

$$Tj = Ta + \theta j\text{-}a \times P$$

Ta　：周囲温度
θj−a　：ジャンクション−大気間熱抵抗（＝ジャンクション−ケース間＋ケース−大気間）
P　：消費電力

このことから、次のようなことが言えます。
同じ消費電力の場合には、熱抵抗が小さいほどジャンクションでの温度上昇は小さくなります。したがって、より高い周囲温度でも使えるようになります。
逆に、同じ周囲温度で使用する場合には、熱抵抗が小さいほど大きな電力を消費させることが可能です。

事　例

Q1. 熱設計の事例はありますか。

A1. 特に事例集のような資料はありません。熱設計のポイントとしては、次の 3 点があります。

・ディレーティングによって全損失を抑える

・熱放射（放熱板・空冷ファン・水冷など）によって熱抵抗を抑える

・使用環境の温度範囲を下げる

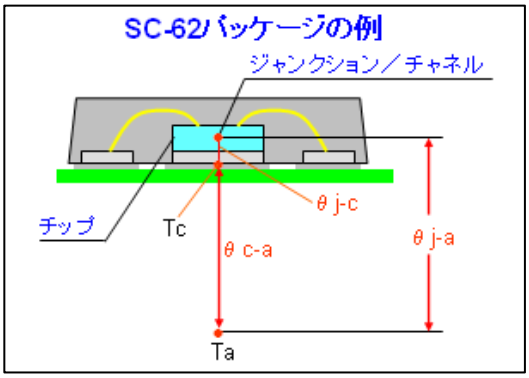
トランジスタの熱抵抗とジャンクション温度の算出については、「ディスクリート＞共通＞熱抵抗」と「ディスクリート＞トランジスタ＞熱設計」をご参照ください。

ここでは、次のようなデバイスについて、全損失を大きくする場合の熱抵抗の算出方法を紹介します。

(1) 最大ジャンクション温度が 150℃

(2) 単体では周囲温度 80℃のとき 2W の消費電力まで可能

(3) デバイスのジャンクション−ケース間の熱抵抗 θj−c が 5℃/W



このデバイス単体のジャンクション−大気間の熱抵抗 θj−a は 35℃/W(＝(150℃－80℃)/2W)です。

消費電力が 2W を越える場合には、熱抵抗をこれより抑える必要があります。たとえば、このデバイスを周囲温度 80℃までで 3W の電力を消費させる場合について考察しましょう。

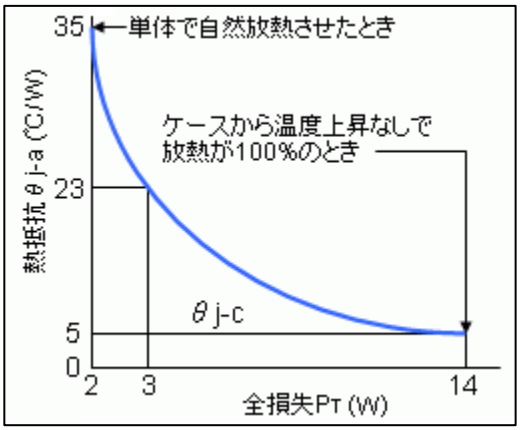
3W 分の発熱を 70℃(＝150℃－80℃) の温度差で伝えるには、θj−a が次の値以下でないといけません。

θj−a＝70℃/3W＝23.3℃/W　（桁数が多くても意味がないので、簡単に 23℃/W 以下と考えます）

(3)より、ケース−大気間の熱抵抗 θc−a を次の値以下にする必要があります。

θc−a＝θj−a－θj−c＝23℃/W－5℃/W＝18℃/W

ちなみに θc−a＝0℃/W とすると、θj−c に 70℃の温度差を生じるので、ケースからの放熱が「温度上昇なしで 100%」と仮定した場合の最大消費電力は 14W(＝70℃/5(℃/W))となります。



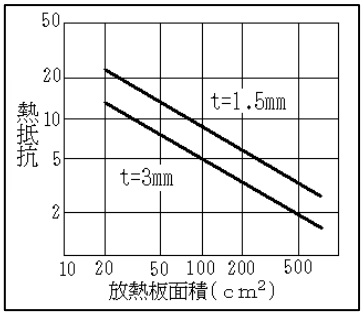
なお、ASIC はケース温度の規定をしていないので、このような熱設計は適用できません。ASIC で厳しい熱設計となる場合には、開発のご依頼先である販売店にご相談ください。

放　熱

Q1. どの程度の放熱板が必要になりますか。

A1. アルミニウムの平板の放熱を参考で示します。

前述の事例（18℃/W）では、厚さ 1.5mm のアルミニウムの板で、50cm² ほどの面積が必要です。実際は、放熱板の材質や空気の対流などで大きく変化します。



注意 取り付ける際は、シリコン・グリスなどで、部品と放熱板間の熱の伝達をよくする必要があります。

★ Q2. 許容損失の実装条件が、セラミック基板の製品とガラスエポキシ基板の製品がありますが、放熱特性は違うのですか。

A2. ガラスエポキシ基板(プリント基板)実装時の許容損失は、セラミック基板実装時の約40%となります。
ディレーティングについては、ディスクリート＞共通の電力定格をご参照ください。

IC パッケージの熱抵抗

Q1. IC で熱抵抗を規定していない製品があるのはなぜですか。

A1. マイコンや周辺デバイスなどは、パワー・デバイスのように消費電力を定格で制限するのではなく、電流条件などによって消費電力がある程度に落ち着きます。このため、動作周囲温度の範囲で使用すればいいので、パッケージの熱抵抗を規定していません。信号線の入出力電流が大きくなれば、発熱量は大きくなりますが、ハイ･レベルとロウ･レベルの限界電圧が電流を条件として規定されており、この電流値を越えると、劣化促進や破壊を生じる恐れがあります。このため、熱設計をする必要はなく、動作周囲温度が規定の範囲を越える場合に、冷却をすればいいことになります。
ディスクリートや電源用 IC、ASIC などは、使用方法が特定できないため、熱設計ができるように、パッケージの熱抵抗を規定しています。

3.6 ノイズ対策

基礎事項

Q1. ノイズ関連で、EMI や EMS というのはどういうことですか。

A1. EMI（Electro Magnet Interference）は、電子機器から漏れる電磁気（電波）雑音の放射や伝搬を規制して、他の機器への妨害を排除する電磁妨害放射規制です。欧米では法的に EMI を規制・規格化しており、欧州はひじょうに厳しくなっています。日本では業界の自主規制となっています。
EMS（Electro Magnetic Susceptibility）は、他の電子機器から漏れる電磁気雑音を受けても、電子機器が問題なく動作する電磁妨害耐量です。
EMI と EMS を併せたものが、EMC（Electro Magnet Compatibility）と呼ばれる電磁適合性です。

Q2. ノイズ発生には、どのような要因がありますか。

A2. 単位時間当たりの電圧／電流の変化(量や向き)が大きいときに、ノイズが発生しやすくなります。このため、大電力や高周波の回路は、ノイズ源となることがあります。ノイズの伝達経路は、電源／グランドや信号線といった配線の場合と、電磁波となって空中に放射される場合(クロストークなど)があります。
ノイズ発生の要因に、セットの構造や回路の電気特性(電流や動作速度、インダクタンス)などがあります。ブラシ・モータ、切片式のリレーやスイッチは、スパークによるサージ・ノイズ(雷と同様)や電源／グランド・ノイズの発生源となります。また、基板上の配線が直角や鋭角なら、ノイズが空中放射されやすくなります。

EMS 設計の基礎

Q1. 半導体デバイスの EMS 設計について、基本的な考え方を教えてください。

A1. ノイズは、経路によって種類があり、その対策が異なります。一般的には、以下のことが考えられます。
これらを組み合わせ、コストや実現性を考慮して、必要な対策を行ってください。

(1) 電源やグランドなどの共通配線から入るノイズについては、レギュレータを挿入して外部ノイズの影響を抑えるとともに、基板の入り口の近くを含めて、適所に周波数特性の良いセラミック・コンデンサをバイパス・コンデンサとして設置します。なお、積層セラミック・コンデンサで容量の大きいものは、誘電体の影響で周波数特性が悪いことがあるので、注意が必要です。基本は、基板の入り口で阻止することです。

(2) 入出力の信号に乗るノイズについては、フェライト・ビーズなどのノイズ対策部品を用いたり、ノイズ・フィルタを挿入することが考えられます。コンデンサでの対策は、コンデンサの電荷の影響で電源電圧以上の電圧になることが考えられるので、これだけでは問題があります。入出力の信号の場合にも、基板の入口から中に入れないように対策すべきです。そのためには、長く伸ばす信号線にはシールド線を用いたり、ツイスト・ペアで差動信号のやり取りをすることが効果的です。
ノイズ・フィルタを設置する場合は、ノイズを GND に逃がすことになるので、GND が十分に強化されている部分に配置してください。

(3) 最後に、直接配線に誘導するノイズについては、多層基板で信号のインピーダンスが下がるようにすることになります。それでも不足なら、基板そのものをシールド・ケースに入れることも考えられます。
そこまでは必要がない場合は、シールド板を立てて、基板をそこに近接して配置することも考えられます。

(4) ノイズ対策では、回路上に現れる部品も重要ですが、部品の配置や配線の引き回しも重要ですので、こちらにもご注意ください。

(5) マイコンの場合、さらに誤動作対策として、可能なところではリフレッシュ動作(定期的に、ポートなどに同じ値を設定して、書き換わっても修正できるようにする)を考慮してください。

Q2. 電源・グランドが層配置されている 4 層や 6 層の基板は、ノイズに強いと考えていいですか。

A2. 電源の供給能力が高く、主要デバイスに対するバイパス・コンデンサの配置が適切であれば、そのように考えられます。電源・グランドが配線状になっていないベタ基板では、電流が必要なルートだけをたどるのではないため、これらが不適切であれば、あるデバイスに流れる電流密度が高くなった場合、付近のデバイスで電圧降下を生じ、電流を十分に流せないデバイスを生じさせるおそれがあります。
電源電流の変化が大きいデバイスでは、インダクタンス分による電圧変動(電流変化を抑えようとする)を生じるので、バイパス・コンデンサの設置がより重要です(電源・グランドが配線状の基板でも同様)。

コンデンサの設置

Q1. 信号ノイズの対策で、グランドとの間にコンデンサを設置したいのですが、容量の制限はありますか。

A1. 一般に、MOS デバイスで問題となります。出力デバイスが許容するのは、配線容量や入力側の容量も含めて、負荷容量で規定する値です。これを越える場合、突入電流への対策回路が必要となります。
また、付加する容量が大きくなると、信号変化が緩慢になるため、入力デバイスでは、タイミングを満足して、貫通電流の影響が問題ないことを確認する必要があります（タイミング問題はアナログ・デバイスも同様）。

サージ／静電気対策

Q1. サージや静電気による半導体デバイス破壊の対策として、端子に接続する直列抵抗は有効ですか。

A1. いいえ、ダンピング抵抗は電流制限になるだけですから、過電圧印加は免れません。電圧を降下させるには、さらに抵抗と端子の間に接地コンデンサを接続する必要があります。ただし、サージや静電気を吸収できる大容量のコンデンサを接続すると、波形がなまること考慮する必要があります。また、MOS デバイスでは、電源遮断時の突入電流を回避するために、ダイオードを電源側に接続する必要があります。
このような回路と比較すれば、ツェナー・ダイオードをサージ・アブソーバとして、信号線とグランドの間に設置するほうが簡素です（「サージ・アブソーバ・デバイス（NSAD シリーズ）」のホームページ参照）。

Q2. リレーの動作時、デバイスへ入力しているクロックにノイズが発生します。
正常動作しているようですが、問題はありますか。また問題発生時は、どのような動作をしますか。

A2. リレーは特に電磁波（サージ・ノイズ）の発生源となるので、電磁遮蔽などの対策が必要と考えられます。
入力クロックの乱れによる動作は特定できません（クロック波形は、レベル幅と遷移時間が規定されているので、規格を満たさない）。また、絶対最大定格を越えるノイズ成分については、デバイスの劣化を促進させて信頼性に影響を与える可能性があります。
なお、ノイズはクロック以外の信号にも影響するので、リレー回路そのものの根本的な対策が必要です。

3.7 べし・べからず集

ハードウエア開発（早期開発・信頼性問題回避のために）

(1) 規格

◆規格の範囲を越えて設計／使用してはいけません。

規格外の状態では、誤動作や劣化促進や破壊により、市場不良を起こす原因になります。

また、特性は個々に異なるので、実力という考えは設計に適用できません。MIN/MAX 規定については、最悪値で設計してください。

なお、静電気やサージが混入するおそれがある箇所は、ツェナー・ダイオードによる保護が必要です。

★ デバイス同士を接続する場合、双方の絶対最大定格とAC特性、DC特性を比較することになります。ここで、DC特性の電圧は、定常状態を想定していることにご注意ください。出力側の過渡特性や配線のインダクタンスなどによって、アンダシュートやオーバシュートが発生し、入力側の絶対最大定格を越えることがあります。この場合、ノイズ・クリッピング・ダイオードなどでの対策がなければ、入力側のラッチアップや劣化促進の原因となります。

(2) 端子処理

◆CMOS デバイスの入力を未接続にしてはいけません（内部プルアップを使用する場合を除く）。

ノイズによる貫通電流が、誤動作や劣化促進の原因になります。

★ また、入出力を切り替えるポートにFETのゲートなどが接続されていて、そのポートが入力になっている期間（ともに入力Hi-Z)も、未接続と同じ状況です。この場合、双方が誤動作する可能性があるので、リセット後にそのポートを出力にするまでの期間、外部デバイスの状態を固定する必要がある場合には、外部でプルアップ／プルダウンしてください。

◆出力同士を直接接続してはいけません(オープン・ドレーンやオープン・コレクタを除く)。

極性の異なる信号が衝突すると、DC 特性を越える過電流が流れ、劣化促進や破壊の原因になります。

◆出力を電源やグランドに直結してはいけません。

★ 出力同士の接続と同様で、劣化促進や破壊の原因になります。

入出力端子を入力で使用する場合のレベル固定でも、ノイズの影響で出力となる可能性を考慮すると、抵抗挿入が理想的です。

◆MOS デバイスの信号線−グランド間にコンデンサ挿入の場合、突入電流や過電圧への対策が必要です。

ノイズ対策でよく検討されますが、コンデンサ付加だけでは、劣化促進や破壊の原因になります。これは、信号変化時に DC 特性を越える過電流が流れ、また電源遮断(0V)時に絶対最大定格を越える過電圧を印加することになるためです。時定数の変化が、入力側の AC 特性に対して問題がないことも要確認です。

★ **◆モータやコイルなどの誘導性負荷の場合、サージへの考察が必要です。**

誘導性負荷の駆動電流が変化すると、電流による磁束が変化して逆起電力を生じ、 変化を妨げる方向に大電流を流そうとします。これがサージとなって、駆動素子の劣化や破壊の原因と なります。駆動素子の定格を越えるサージが発生する場合は、ダイオードやスナバ回路などでの対策が必要です。なお、ダイオード内蔵型のトランジスタやFETでの駆動も有効です。

◆電源遮断時に外部機器から信号が入力される電源シーケンスの場合、MOS 入力なら過電圧への対策が必要です。

入力電圧の絶対最大定格で、最大値が「V_{DD}＋α」となっている場合、電源遮断(0V)時に信号を入力すると、劣化促進や破壊の原因になります。特に、この状態で電源を投入すると、ラッチアップが発生しやすくなります。なお、「＋6.5V」などの絶対値表記になっている端子では、その電圧までの入力は問題ありません。

◆外部入力クロックの周波数や供給元の切り替えで、ショート・パルスを発生させてはいけません。

規格を満さないパルスは(1)に該当するため、誤動作の原因になります。特に、信号と非同期な手動スイッチによる切り替えに注意が必要です。また、切り替え時にスパイク・ノイズが発生して、その電圧が DC 特性を越えると、劣化促進や破壊の原因にもなります。

◆デジタル回路での手動スイッチなどの入力について、チャタリングへの考察が必要です。

機械式のスイッチやキー、リレーなどは、作動によってチャタリングを生じて、ショート・パルスを発生させるため、回路によっては誤動作の原因になります。これらの入力をマイコンで認識する場合は、ソフト・タイマ（プログラム処理）でサンプリング待ちをして対策することができます。

なお、トグル・スイッチや回路切り替えスイッチの入力で、チャタリングが問題とならないサンプリング回路（サンプリング間隔がチャタリングより長い）なら、特に考慮する必要はありません。

◆マイコンでアナログ入力と兼用のポートを使用する場合、AV_{REF} 端子がポート電源なら電源供給をしなければいけません。

AV_{REF} が GND 接続では、ポートが機能しなかったり、内部に異常電流が流れて信頼性に影響したりすることがあります。アナログ・ポート電源が AV_{REF} か V_{DD} かは、端子の入出力回路でご確認ください（電源が V_{DD} なら、AV_{REF} は GND 接続可）。

★ **◆電源電圧の異なるデバイスの接続では、トレラント設計が必要です。**

たとえば、3.3V 動作と 5V 動作の CMOS デバイス同士を接続する場合、3.3V 動作デバイスの出力に 5V 耐圧以上のオープン・ドレーン・バッファを使用し、その出力を 5V 電源に抵抗でプルアップする必要があります。また、3.3V 動作デバイスの入力にも、5V 耐圧以上のバッファを使用する必要があります。いずれの場合も、5V 以上の耐圧がない通常のバッファでは、絶対最大定格を越えて信頼性問題の原因にもなります。

(3) 起動

◆発振回路を内蔵しているデバイスは、発振が安定する前に起動してはいけません。

不安定なクロックでは、どういう動作をするかを特定できません。外部発振回路からマイコンへクロックを供給する場合、パワーオン・リセット信号はクロックが安定してから解除してください。

なおマイコン内部の発振については、内部タイマで安定待ち時間を設定できたり、フラグで状態確認ができる製品があります。また、時間設定については、オプション・バイトで起動時に行ったり、起動後にプログラムで行う場合があります。

(4) 信号変化

◆入力信号の立ち上がり／立ち下がりは、あまり緩慢にしてはいけません。

一般に信号の遷移時間は規定されており、それを守る必要があります。しかし、マイコンのポート状態はプログラムで読み込まれるため、ランダム・ロジックに比べて時間がかかるため、立ち上がり／立ち下がり時間の規定をしていません。だからといって、あまり長い遷移状態では問題となることがありますので、大容量コンデンサで極端になまらせることなどは避けるべきです。

★ なお、電流や容量(突入電流)の過負荷は、波形のなまりだけでなく、信頼性にも影響を与えかねません。

(5) 電源供給

◆電源電圧の変動は、あまり急峻にしてはいけません。

動作電源電圧に「1.8～5.5V」などの幅がある製品では、その範囲内の変動があっても動作可能です。ただし、その変動は 0.1V/ms 以内を目安としてください。また、電源電圧に伴って、動作周波数範囲や DC 特性、絶対最大定格などが変動することにご注意ください。

★ **◆異なる電源で動作するデバイス間の信号接続をするには、グランドを同電位にする必要があります。**

グランド間に電位差があると、相対的な入出力電圧がずれるため、誤動作や破壊の原因となります。このような場合、基本的にはグランド同士を接続しますが、それができない回路では、信号線をフォトカプラで電気的に絶縁分離するなどの処置が必要です。

ソフトウエア開発（早期開発・信頼性問題回避のために）

(1) 操作

◆レジスタに禁止されている値を設定してはいけません。

禁止設定では、どういう動作をするかを特定できません。

◆禁止されているレジスタの書き込みや書き換えをしてはいけません。

禁止アクセスをすると、どういう動作をするかを特定できません。

ただし、書き換え禁止の場合、同じ値の書き込みは問題ありません。

なお、エミュレータによっては、リード・オンリのレジスタに書き込むプログラムの実行で、強制ブレークがかかってエラー終了することによって、不正アクセスを発見できることがあります。

(2) メモリ領域

◆マイコンのプログラムは、フェッチできないメモリ領域においてはいけません。

フェッチできないプログラムは実行できません。

◆マイコンで多重割り込みをする場合、想定されるネスティングに対して余裕のあるスタック領域を RAM 上に確保しなければいけません。

データやプログラムの書き換えが生じると、どういう動作をするかは特定できません。

(3) 初期化

◆オプション・バイトのあるフラッシュ・マイコンでは、これを設定しなければなりません。

マスク品ではマスク発注時に、コーディング・シートでオプション（ユーザ固定モード）のご指定をいただき、ハードウエアで固定します。フラッシュ・マイコンではソフトウエア指定なので、ユーザがプログラムとともにオプションを書き込む必要があります。

◆マイコンのプログラムを C 言語で記述すると、初期化をするスタートアップ・ルーチンが必要です。

スタートアップ・ルーチンがないと、周辺機能のレジスタ初期化や変数の初期化ができません。

(4) 起動

◆発振回路を内蔵しているデバイスでは、発振が安定する前にプログラムを起動してはいけません。

不安定なクロックでは、どういう動作をするかを特定できません。マイコンへの供給クロックとは別の発振回路からのクロックを供給している周辺デバイスがある場合、マイコンからそのデバイスへのアクセスは、その発振が安定してから行うようにプログラムしてください。なおマイコン内部の発振については、内部タイマで安定待ち時間を設定できたり、フラグで状態確認ができる製品があります。また、時間設定については、オプション・バイトで起動時に行ったり、起動後にプログラムで行う場合があります。

取り扱い（信頼性問題回避のために）

取り扱いの留意事項については、品質／信頼性ハンドブックの「5.2.5 使用環境に関する注意事項」「5.3 包装，保管，運搬，取り扱い上の注意事項」をご参照ください。

(1) 保管

◆保存温度を越えて半導体製品を保管してはいけません。

絶対最大定格オーバは、劣化促進や破壊の原因になります。

(2) 静電気

◆半導体製品に静電気を与えてはいけません。

絶対最大定格（電源電圧、入力電圧）オーバは、劣化促進や破壊の原因になります。

静電気対策については、静電気放電（ESD）破壊対策ガイドの第 6 章をご参照ください。

(3) 実装

◆半導体製品を基板にはんだ付け実装する場合、推奨条件（温度・時間・回数）を越えてはいけません。

絶対最大定格（チャネルやジャンクションの最高温度）を越えることになり、劣化促進や破壊の原因になります。なお、開封後の保管日数によってベーキング条件がある製品では、その日数を越えた場合、所定のプリベークをしてください。

(4) 接続

◆電源投入状態で、機器間のインタフェース・ケーブルを接続／取り外ししてはいけません。

挿抜時にスパイク・ノイズが発生して、その電圧が DC 特性を越えると、入力側と出力側ともに、劣化促進や破壊の原因になります。マイコン開発時のツール接続や、市場での装置接続などに配慮が必要です。

ただし、ホット・プラグインに対応しているインタフェースでは、所定の手順を守れば、特に問題ありません。

3.8 トラブル・シュート

回路が動作しない

(1) 各デバイスの電気的特性の範囲内で設計・使用していますか。

◆電源電圧、入出力電圧、動作タイミング、動作周囲温度（放熱）は適正ですか？

◆電源は各デバイスに正常に供給されていますか？

電源の電流容量は適正ですか？

電源の極性は正しいですか？

複数の電源・グランド端子がある場合、すべてを接続していますか？

電源が発振していませんか？（周辺に使用しているコンデンサの温度特性を考慮していますか？）

電源が電池の場合、低温時の電池の電圧／容量の低下を考慮していますか？

◆電源／グランドにノイズが乗っていませんか？

電源／グランドの配線、バイパス・コンデンサの設置は適正ですか？

◆信号線にノイズが乗っていませんか？

フィルタやボード・レイアウト、配線などのノイズ対策は適正ですか？

◆過負荷になっていませんか？

出力端子に接続している配線／入力端子の電流、容量の各合計は、出力の規格内ですか？

◆規定のリセットがかかっていますか？

◆クロック波形は正常ですか？

◆RF/マイクロ波でのインピーダンス整合は取れていますか？

(2) CMOS デバイスの入力が未接続になっていませんか。

(3) 出力同士が衝突していませんか（オープン・ドレーンやオープン・コレクタを除く）。

(4) 出力を電源やグランドに直結していませんか。

(5) デジタル設計の論理は合っていますか。

◆アクティブ・レベルやデコーダ、セレクタなどの回路の論理は適正ですか？

(6) IC の外付け回路などの定数は適切ですか。

◆特に RF/マイクロ波の場合は、インピーダンス整合のための定数の最適化が重要です。

(7) プログラマブル・デバイスのレジスタ設定は適切ですか。

◆誤ったモード設定や禁止状態の設定はないですか？

(8) 発振回路内蔵デバイスの発振安定時間を待っていますか。

(9) マイコンのプログラムは、フェッチ可能なメモリ・エリアに展開していますか。

(10) マイコンの多重割り込みで、スタックがオーバフローしていませんか。

◆スタック・ポインタの設定をご確認ください。

(11) マイコンの割り込みが、他の割り込みとの競合で保留されていませんか。

◆割り込み応答が遅くなると、処理が間に合わなくなり、その後の処理ができなくなることがあります。

この場合は、割り込みの優先度の見直しと多重割り込みをご検討ください。

(12) デバイスが壊れていませんか（規格越えや寿命）

◆MOS デバイスやその搭載基板の取り扱いで、静電気対策をされましたか？

◆端子に過電圧がかかっていませんか？

◆実装（はんだ付け）条件は守られましたか（温度破壊）。

◆保存温度を越えて保管していませんか？

◆製品寿命（劣化）ではないですか？

備考 内部ショート状態の破壊では、電源ダウンで誤動作デバイスの特定が困難な場合があります。
なお、ショートになるかオープンになるかは予測できません。

正常動作途中で回路が動作しなくなり、リセットが効かない

規定外の電圧（ノイズを含む）を CMOS デバイスに入力して、ラッチアップを起こしていませんか。

◆デバイスが壊れていなければ、正規の入力電圧にすれば「電源断 － 再投入 － リセット」で再起動します。
ただしラッチアップは劣化を促進させるので、当該システムを量産出荷されないようお勧めします。

シリアル通信で受信ができない（送信データは正常）

(1) 送信側と受信側の伝送速度の設定は一致していますか。

(2) 送信側の伝送速度誤差は、受信側の許容範囲内ですか。

(3) 受信データにノイズが乗っていませんか。

◆数十 ns のサンプリング間隔で波形観測をしてみてください。

マイコン・システムで、同じ動作を繰り返して先へ進まない

プログラムに無限ループとなる個所はありませんか。

マイコンのポートにハイ・レベルを出力できない

(1) ポート・モード・レジスタで出力モードに設定していますか。

ポート・レジスタに"1"を設定するだけでは、出力になりません。

(2) オープン・ドレインのポートなら、プルアップ抵抗を接続していますか。

プルアップがなければ、ハイ・レベル駆動要因がありません（プルアップ抵抗は、内部と外部のいずれも可）。

(3) アナログ入力兼用ポートなら、アナログ電源を供給していますか。

アナログ入力が兼用の場合、デジタル出力の電源がアナログ電源になっていることがあります。
その製品の入出力回路図をご参照ください。

★ **(4) 低電圧動作デバイスのCMOS出力を高電圧動作デバイスに直接入力していませんか。**

たとえば、3.3Vや2.5Vでの動作デバイスの出力を5V動作デバイスに入力して、5V電源に抵抗でプルアップしても、十分なハイ・レベルを確保できません。また、その出力バッファに5V以上の耐圧がなければ、劣化促進による市場不良の原因にもなります。
5V 耐圧以上のオープン・ドレーン・バッファでのトレラントが必要です。

端子の内部破壊が起こる

(1) 過電圧や過電流になっていませんか。

◆DC 特性や推奨動作条件を越えた設計になっていると、劣化を促進させる可能性があります。

◆規格内の設計であっても、ノイズ対策がされていないと、ノイズの影響で劣化する可能性があります。
また、アンダシュート／オーバシュートを生じる信号の入力でも、同様に劣化する可能性があります。

(2) 静電気を印加していませんか。

◆基板へのデバイス実装時に、静電気対策をしてください。
特に手で取り扱う場合、QFP などはピンに触れるため、対策が不十分なら破壊の可能性が高くなります。
参考資料：「静電気放電(ESD)破壊対策ガイド」

◆基板実装後でも、基板やコネクタの端子に触れず、導電素材で包装をしてください。

クロック発振回路が発振しない

発振回路の動作はデリケートです。発振子とのマッチングやノイズ環境などに、次のような問題はありませんか。

◆たまたま動作した、試作の発振定数で量産した（特性のばらつきを要確認）
発振回路はアナログで、特性のばらつきがあります。使用されるコンデンサの容量にも誤差があります。

◆試作と量産で、配線の特性が変わった（バラック配線とプリント配線では、長さ・配置・容量などが異なる）

◆配線が長い（容量が大きくなり、ノイズも拾いやすくなる）

◆他の信号線が交差している（ノイズを拾いやすくなる）

◆付近に大電流配線がある（グランド電位が変動する）

◆グランド線を他の回路と共用している（グランド電位が変動する）

◆発振信号線を取り出している（容量が大きくなり、グランド電位も変動する）
波形観測でプローブをあてると同様の状況です。また、発振は微弱振動なので、波形観測はできません。

◆フラッシュ・マイコンをマスク品に置き換えた（マッチング（コンデンサ容量）の再評価が必要）
ROM サイズの変更だけなら、再評価は不要です。

4. 環境関連

4.1 環境負荷

RoHS 指令

Q1. RoHS とはなんですか。

A1. "The restriction of the use of certain hazardous substances in electrical and electronic equipment"(電気電子機器における特定有害物質使用制限)のことで、「ローズ」または「ロス」と呼ばれます。
欧州連合の委員会による指令で、水銀、カドミウム、鉛、六価クロム、PBB、PBDE の 6 物質が規制されています。

Q2. ルネサス エレクトロニクス社の製品は、RoHS 指令の規制物質を含有していますか。

A2. 当社製品でRoHS規制の基準値を越える含有物質は、旧来のはんだめっき品の鉛だけです。したがって、鉛フリー品はRoHS指令に適合しています。
なお、当社はRoHS規制の対象6物質についても工程管理をしており、鉛フリー品は、鉛も基準値以下であることを確認しています。当社はグリーン調達目標に従い、鉛フリー化に対応しています。

Q3. 鉛フリー品の品名で「内部の高融点はんだは RoHS 規制の対象外」と記載されていますが、なぜですか。

A3. RoHS 指令の ANNEX 7 にて、「鉛 85%以上含有の高融点半田」が除外されています。
現状、パワー系のデバイスでは、高融点はんだが必須です。
これは今日の技術で代替がないためで、鉛を全面禁止すると、鉛バッテリを搭載せざるをえない車が、世の中から姿を消すことになることと同様です。

Q4. RoHS 指令の内容は、どこかに掲示されていますか。

A4. 欧州連合EUR-LexのOfficial Journalページで、次の指定で検索できます。

Year　　:2003
OJ series　:L
OJ number:037

32002L0095(2002/95/EC)が RoHS 指令の本体です。

Q5. 有鉛品の工程残留により、鉛フリー品に鉛が混入するのではないですか。

A5. RoHS 指令では、環境負荷物質の含有を最大 0.1wt%と指定しており、当社では、工程残留による鉛の混入をこの範囲となるよう、管理しております。

Q6. RoHS 指令について、どのような Phase に対応されていますか。

★ **A6.** RoHS 指令では Phase という定義をしていません。以前 JEITA が発行していた ETR-7021 規格(電子材料、電子部品、実装済基板の鉛フリー表示のガイドライン)に、Phase という表現が使われていました。
ETR-7021 規格では、完全鉛フリー品は Phase3 で、欧州 RoHS の適用除外に該当しない部分が鉛フリーの製品を Phase3A で表現していました。
当社の製品に当てはめると、オーダ品名末尾が-A や-AT、-AX の製品は Phase3 で、-AZ や-AY の製品は Phase3A に相当します。
その後、鉛だけでなく、RoHS 指令で規制している 6 物質の含有／非含有の表示を求める社会的要請から、ET-7001 規格(Phase という表現なし)が作られ、ETR-7021 規格は発展的に中止になっています。

環境への取り組み

Q1. 製品について、環境関連の規制と、それに対する取り組みはどうなっていますか。

A1. 環境関連については、欧州連合 EU が次のような指令を制定しており、当社の製品が該当する RoHS 指令について、対応を推進しています。

RoHS　:特定有害物質使用制限（2006.7 施行）
WEEE　:廃電気電子機器指令（2005.8 までに各国実施：部品は除外）
ELV　:廃自動車指令（2003.7 施行：電気部品は除外）

★ Q2. シリカゲルには環境負荷の規制があると聞きましたが、出荷梱包されている乾燥剤は問題ありませんか。

A2. はい、まったく問題ありません。
現在ではコバルト・フリーのシリカゲルもありますが、従来品は塩化コバルトを含み、EU指令(67/548/EEC、2003/34/EC)で規制されています（ただし、RoHS規制の対象ではありません）。
このため、当社は1993年に乾燥剤をシリカゲルから天然鉱物(吸湿性土)を原料とするDESI PAK(デシパック)に変更しました。DESI PAK原料の吸湿性能は、MIL規格(D-3464E)やJIS規格(Z-0701)の要求を大きく上回り、米国では政府調達品としてQPL(Qualified Products List)に登録されています。またその化学組成は、米国連邦規則で定めるFDA(食品医薬局)のGRAS(一般に安全と認められる物質)に適合し、誤って食べてしまっても無害です。

電子情報製品汚染抑制管理弁法（中国版 RoHS）

Q1. 電子情報製品汚染抑制管理弁法とは何ですか。

A1. 中国政府が 2006 年2月 28 日公布した、環境負荷になる特定有害物質規制に関する法律です（2007 年 3 月 1 日施行）。その対象物質と許容含有上限値は、欧州 RoHS の規制対象物質（水銀、カドミウム、鉛、六価クロム、PBB、PBDE の 6 物質）と同じで、中国版 RoHS とも呼ばれています。違いもありますので、当社ホームページの「中国版 RoHS への対応」をご参照ください。

JIG(Joint Industry Guide)

Q1. ルネサス エレクトロニクス社の製品は、JEDEC のレベル A の規制物質を含有していますか。

A1. 環境負荷物質の規制について、次の 3 者は合同規格 JIG-101 に統合されています。

・米国の工業団体 JEDEC(Joint Electron Device Engineering Council)
・米国の電子工業会 EIA(Electronic Industries Alliance)
・日本のグリーン調達調査共通化協議会 JGPSSI(Japan Green Procurement Survey Standardization Initiative)

JIG-101 で使用禁止されているレベル A の 15 物質群について、当社の半導体製品で使用しているのは、有鉛品の鉛だけです。当社の鉛フリー製品については、欧州 RoHS 指令で使用が認められている適用除外項目の用途以外には、鉛を使用していません。

Q2. JIG-101 の内容は、どこかに掲示されていますか。

A2. 和文は JEITA のホームページ、英文は JEDEC のホームページに、それぞれ掲示されています。

アスベスト

Q1. ルネサス エレクトロニクス社の製品は、アスベストを含有していますか。

A1. いいえ、製品や梱包材など、アスベストは含有していません。

Q2. 工場でアスベストを使用していて、製品に混入することはありませんか。

A2. ご安心ください。工場の建材にはアスベストを使用しておらず、熱処理設備等でのアスベストは非飛散性か、飛散防止処置をしており、法的な基準を十分クリアしています。製品への飛散混入の恐れはありません。

★ ガリウムひ素

Q1. フォトカプラやフォトMOSFETは、鉛フリー化によってガリウムひ素を使用しないようになりましたか。

A1. ガリウムひ素は発光ダイオード部に使用しています。
鉛フリー化はリード部のめっき材になります。
したがって、鉛フリー化によってガリウムひ素を使用しないということはありません。
オプト・RF＆マイクロ波デバイスの鉛フリー化については、次の URL ご参照ください。
http://www2.renesas.com/pb_free/ja/csd/contents.html

★ フマル酸ジメチル(DMF)

Q1. 製品、包装部材や乾燥剤などに、フマル酸ジメチル(DMF)を含有していますか。

A1. いいえ、含有していません。

4.2 鉛フリー

鉛フリー化

Q1. 全製品について、鉛フリー品に切り替えているのですか。

A1. 全面切り替えをしているわけではありませんが、基本的に鉛フリー化をしています。
車載や通信関連は RoHS 規制から除外されているため、お客様のセット生産に支障がないよう、これら製品の一部は鉛フリー化をしていません。
また、従来の有鉛品については、基本的に鉛フリー品と平行生産をしています。
ただし、新規開発は鉛フリー品だけです。
また、光デバイスと RF/マイクロ波デバイスでは、2006 年 10 月以降の生産は鉛フリーに切り替えています。
鉛フリー品については、鉛フリー製品リストをご参照ください。

はんだ材

Q1. 鉛フリー品のはんだ材は、何を使用しているのですか。

A1. 端子処理によって、次のように分類されます。

リード・メッキ	:	Sn-Bi（すず-ビスマス）または Sn（すず）、Ni-Pd-Au（ニッケル-パラジウム-金）（製品によって異なります）
リード・ディップ注	:	Sn-Ag-Cu（すず-銀-銅）
半田ボール	:	同上

注 ディスクリート製品の端子挿入品（ただし TO-251(MP-3)は Sn-Bi）

なお、他のパッケージ材質は、鉛はんだ品と同じです。

Q2. Sn 含有メッキの鉛フリー品の外装で、Sn 以外の含有量はどのくらいですか。

A2. 次のとおりです。
Sn-Bi :Bi が約 2wt%
Sn-Ag-Cu :Ag が約 3wt%、Cu が約 0.5wt%
詳細は、半導体鉛フリー製品パンフレットをご参照ください。

Q3. 鉛フリー品で、高融点はんだを使用しているのは、どの部分ですか。

A3. 鉛を含有する高融点はんだは、チップをリード・フレームに接着するダイアタッチ（ダイボンド）に使用されています。チップとリード・フレームの関係は、ホームページの半導体ができるまでをご参照ください。このページにリンクしているマウンティングの工程で、例として「銀ペースト」と記述している材料がダイアタッチです。
パワー系のデバイスでは、チップが高温となるため、溶融しないように高融点はんだを採用しています。代替材料の問題で、この部分の鉛含有は RoHS 規制の対象外となっています。

Q4. A2 や半導体鉛フリー製品パンフレットに「wt%」という表記がありますが、どういう意味ですか。

A4. 重量比率(weight per cent)の単位です。
ちなみに、「Sn-3Ag-0.5Cu」は Ag を 3wt%、Cu を 0.5wt%含有する、Sn-Ag-Cu ということです。
パンフレットでは、RoHS の許容上限値を「ppm」(part per million:1/1,000,000)で表していますが、これも重量比率です。つまり、1000ppm＝0.1wt%ということになります。

鉛フリー品の品名

Q1. 鉛フリー品は、品名に違いがあるのですか。

A1. ほとんどの製品では、チップやパッケージ形状が違うわけではないので、品名捺印は同じです。
捺印面では、●マークなどを印字しています（印字スペースのない小型パッケージは、違いがありません）。
オーダ品名は、末尾に分類記号が付加されますが、製品によって次のように分かれます。

リード外装	Sn-Bi, Sn-Ag-Cu	Sn	Ni/Pd/Au(Au が表面)
完全鉛フリー対応	-A	-AT	-AX または-G
端子部分だけ鉛フリー[注]	-AZ	-AY	なし

注　内部の高融点半田は RoHS 規制の対象外

/JM のような生産国表示のある製品では、その前に鉛フリーの分類記号が付加されます。
ごく一部のパッケージで形状が異なり、オーダ品名が異なりますので、個別には購入先にご確認ください。

Q2. 鉛フリー・コードにはいろいろありますが、どれを指定してもいいのですか。

A2. いいえ、製品によってコードが異なります。製品ごとのコードは、鉛フリー製品リストか、その製品の関連ドキュメントなどでご確認ください。
ディスクリートやリニア IC では、「-A と-AT」か「-AZ と-AY」の併記がありますが、新規購入では-AT か-AY(Sn めっき)をご指定ください。-A と-AZ(Sn-Bi めっき)は継続供給用です。ただし、Web ショップなど商社で一般向け在庫があればご購入いただけますので、具体的な供給については、購入先にご確認ください。
マイコンでは「-A と-AX」が併記されているものがありますが、いずれもご指定いただけます。
なお、はんだボール品(Sn-Ag-Cu)は、別の材質で生産する予定はありません。

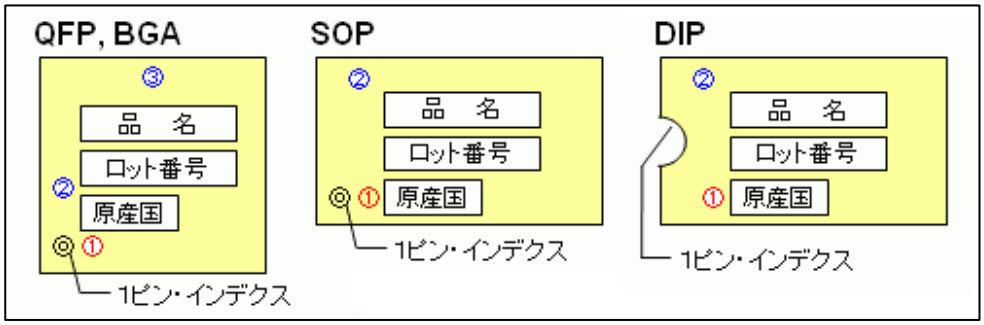
Q3 CCD センサの端子はもともと半田めっきではないようですが、-AZ が付加されている製品には、高融点半田が含まれているということですか。

A3. いいえ、CCD センサには、ガラス部分に鉛を含むものがあり、これを-AZ で表しています。ガラス部分に鉛を含まないものは-A です。

鉛フリーのマーク

Q1. 鉛フリー・マークには、標準的な位置がありますか。

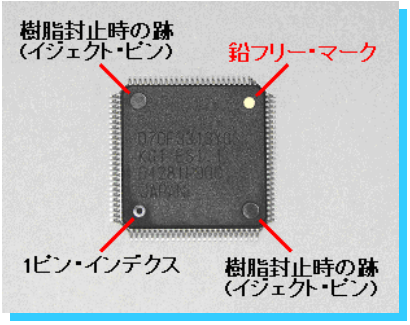
A1. 1ピン付近など、次の位置を標準としています。



①が基本形ですが、製品によっては、②③もあります。
また、これらの標準と異なる製品もありますので、鉛フリー・マークを1ピン・マークと誤認しないよう、基板への実装方向にご注意ください。なお、スペースの制約などで、鉛フリー・マークがない製品もあります。

Q2. 鉛フリーのマークは、統一しているのですか。

A2. 完全な統一はしていません。
捺印の手法は、塗料かレーザです(窪んでいません)。
形状は「○型」などです。位置は、1ピン・インデクス付近などを想定していますが、異なるものがあります。
右図は、**判別に注意を要する例**です。



Q3. 現品を梱包している状態で、鉛フリー品かどうか判別できますか。

A3. 梱包箱やテープ・リールのラベル表記で判別できます。鉛フリー品では、右上に「Pb-Free T.」の表示があります。また、オーダ・コード表記"INTP"では、鉛フリーを表すコードが表示されています。

鉛フリー品のラベル表記例



鉛フリー品の特性

Q1. 従来の有鉛品と鉛フリー品とで、機能／性能に違いがありますか。

A1. はんだと、そのはんだ付け推奨条件が異なるだけで、機能／性能に違いはありません。

鉛フリー品のデータ・シート

Q1. 鉛フリー品のデータ・シートはありますか。

A1. ほとんどが特性・外形とも有鉛品と同じなので、従来（有鉛品）のデータ・シートが共通資料になっています。

実装信頼性

Q1. 「半導体鉛フリー製品」パンフレットの温度サイクル寿命グラフ右側の LnH(t)はどういう値ですか。

A1. LnH(t)はワイブル分布によるハザード確率の推定パラメータで、Ln は自然対数、H(t)はワイブル分布です。

絶縁ブッシング

Q1. パワートランジスタ取付け部品の「絶縁ブッシング」は、鉛フリー対応済みですか。

A1. はい、絶縁ブッシングは、鉛を含む RoHS 規制 6 物質を含有していません。

5. 購入・出荷

5.1 購　入

納　期

Q1. 半導体製品の納期はどのくらいですか。

A1. 納期は製品や生産状況によって異なりますので、個々に販売店にご確認ください。
流通在庫に余裕があれば、短期に納入されると思われます。
流通在庫に余裕がなければ、新たに生産投入することになります。
この場合は、納期が数ヶ月注かかることになります。
このため、できるだけ早期に、販売店に所要をご提示ください。

注　製造と検査(長期寿命試験あり)の期間を含みます。

少量であれば、オンラインの Web ショップでご購入いただけるものもあります。

ランク指定

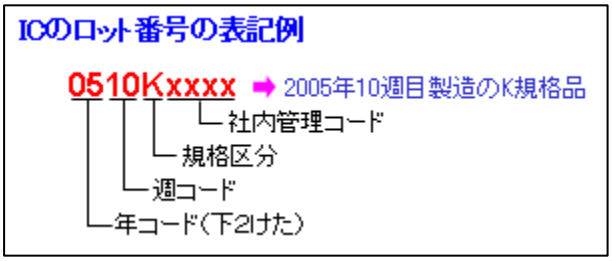
Q1. トランジスタやツェナー・ダイオードには規格区分がありますが、このランク指定をしないで発注すると、どういう特性のものが納入されますか。

A1. ランク指定をしない場合、オール・ランクという扱いになり、いずれかのランク品が納入されるというだけで、どのランクかは特定できません。
ツェナー・ダイオードでは、たとえば規格表に「B,B1,B2,B3」という規格区分があれば、「B」がオール・ランクで、「B1,B2,B3」を包含した区分です。この場合、ランク指定なしは B ランク指定とみなされ、「B1,B2,B3」のいずれかのランク品が納入されます。どのランクかは、梱包ラベルに表示されます。
現品捺印でも区別していますが、表示方法は製品やパッケージで異なります。

ロット番号

Q1. 納入された製品のロット番号は、どのような意味を持っているのですか。

A1. ロット番号は、製造日と規格区分、社内の管理コード(非公開)を表します。
製造年は、IC では西暦の末尾 2 けた、ディスクリートでは末尾 1 けたを表示します。
日付については、IC では週コード、ディスクリートでは月コードを表示します。



なお、製品捺印では、スペースの制約から、省略したり記号化したりしたものがあります。このため製品によって、捺印仕様の異なるものがあります。オペアンプ／コンパレータ、電源 IC やディスクリートについては、それぞれの FAQ をご参照ください。

★

捺　印

Q1. 製品によって現品捺印はかなり違いますが、どのような表示をされているのですか。

A1. 基本的には、社標ロゴ、品名、ロット番号を表記しますが、捺印面の制約で省略や記号化をしているものがあります。またディスクリートでは、規格区分があれば表記します。なお、IC では原産国を表示し、愛称があればそれを表記することがあります。さらに、鉛フリー品では、一般に●マークなどを明示しています。
ディスクリートとオペアンプ／コンパレータ、電源用 IC については、製品ごとの捺印情報をご参照ください。
なお、カスタム捺印をしている製品では、お客様のご指定による捺印をしますので、まったく異なった捺印になることがあります。また、装置に使用されているデバイスでは、その装置メーカのカスタム捺印であることもあり、社標ロゴが表示されていても、そのメーカが製造したものとは限りません。

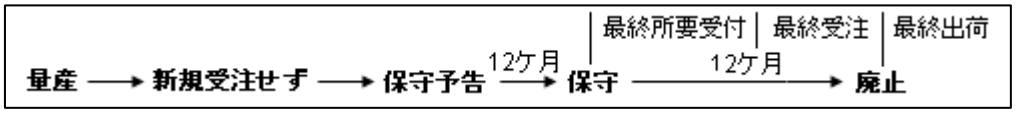
生産状況と供給

Q1. 販社から新規の商談を受けないと言われた製品があるのですが、どういうことですか。

A1. 集約方向で、以前から継続して使用されているセットへの供給しかしていない製品のことです。市場在庫(Web ショップを含む)がなければ、入手は困難です。

Q2. 販社から保守品と言われた製品があるのですが、生産中止したということですか。

A2. いいえ保守品は、お客様のセットの保守のためのスペアの位置づけで、それに必要とされる数量だけ生産するものです。通常、量産後は次のような生産状況に遷移します(製品個々には、販社にご確認ください)。



「保守予告」になると、継続使用されているお客様には、販社から保守・廃止予定の通知をお届けします。
「保守化」の段階で最終所要を販社にご提示いただき、その分だけ生産して終了となります。
最終所要の対応窓口は販社です。供給は販社の裁量となりますので、購入先にご相談ください。

★ **Q3.** スイッチング・ダイオードの生産状況を教えてください。

A3. 1S953 などガラス封止リード挿入品は廃止予定の保守品です(データ・シートは残っておりません)。
1SS220 など樹脂封止面実装品も集約方向で、新規受注をしていません。このため、当社に代替品はありません(データ・シートは Web 検索で閲覧できます)。概略仕様は次のとおりです。

【リード挿入品(ガラス封止)】

項　目	1S953	1S954	1S955	単位
せん頭逆電圧	35	75	100	[V]
サージ電流(1μs)	2000	4000	4000	[mA]
平均整流電流	100	200	200	[mA]
電力損失	500	500	500	[mW]
逆回復時間	MAX.3.0	MAX.3.0	MAX.3.0	[ns]

【面実装品(樹脂封止)】

項　目	1SS220	1SS221	1S2835	1S2836	単位
せん頭逆電圧	70	100	35	75	[V]
サージ電流(1μs)	2000	2000	4000	4000	[mA]
平均整流電流	100	100	100	100	[mA]
熱抵抗 Rth(j-a)	670	670	670	670	[℃/W]
逆回復時間	MAX.3.0	MAX.3.0	MAX.4.0	MAX.4.0	[ns]

購入先

Q1. 製品の購入は、どのようにすればいいのですか。

A1. 次のような販売チャネルがありますので、状況に応じてご判断ください。

(1) 店舗販売

電気パーツを扱っている一般店舗様での販売ですが、すべての製品がそろっているとは限りません。

(2) Web ショップ

ホームページから、販売会社が開設している Web を通じてご注文をいただきますが、一部製品の対応ですし、大量のご注文には対応できません。

(3) 販売店

当社と契約している販売店様(一部は Web 掲載)か、一般の商社様を通じてご注文をいただきます。大量のご注文や、需要予測に応じた継続供給などに対応できます。

Q2. どの販売店から購入すればいいでしょうか。

A2. 独占禁止法に基づいて、当社が特定の販売店様を指定することはできません。また、お客様の会社と販売店様との関係は、具体的には当社には分かりません。複数の販売店様と取引をされている場合もあり、最終的にはお客様の会社の購買部門の判断となりますので、そちらにご確認いただくのがいいと思われます。販売店様とのお取引がない場合、各販売店様にご相談ください。支店等のある販売店様がありますので、必ずしも近隣に本社がない販売店様は対応できないというわけではありません。

模造品

Q1. 形状や捺印が変な製品が入荷されましたが、問題ないでしょうか。

A1. 廃止品などの模造が、ネットなどで取引されているようなので、当社製品ではない可能性があります。他社品の模造で、チップが入っていなかった例もあるようです。偽ブランドや偽札と同様、模造品については、正規メーカである当社では対応できません。ご購入先から流通経路をさかのぼり、調査される必要があります。
偽札のように模造精度の向上を避けるため、詳細はお伝えできませんが、次のような確認事項があります。

- 廃止品が大量購入できることは疑問
- 捺印(社標ロゴ、品名、ロット表記など)は正規か(通常、廃止以降の製造年月ロットはない)
- 外形は外形図と一致するか
- ラベル表記の文字や内容は正規か
- 正しい設計で、現品が正常動作するか

なお、特にネット販売での問題が多発していますので、次のような状況もご確認ください。

- 当社が紹介している販売サイト(WEB ショップやサード・パーティ)でのご購入か
  または当社販売チャネルを経由していることが明確か
- その販売サイトには、お問い合わせを受け付ける(泣き寝入りをさせない)仕組みがあるか

Q2. ルネサス エレクトロニクスの製品が、他社のブランドで販売されていますが、模造品でしょうか。

A2. いいえ、模造品とはかぎりません。
トランジスタなどのディスクリート品は、同等性能であれば、他社が生産していいことになっています。
また、半導体製品全般に、当社製品にカスタム捺印で OEM ブランドとして出荷することもあります。ただし、カスタム捺印の製品が一般市場に流れることはありません。
なお、当社が他社とライセンス契約をして、その会社が自社製品（セカンド・ソース）として生産をすることもあります（一般に品名は変更される）。
模造品とは、当社との契約に基づかず、他社が製造したものです（同等のディスクリート品を除く）。
特に問題となっているのは、当社ロゴの捺印や梱包表示をしているものです。

用　途

Q1. 車載機器に使用するデバイスを Web ショップで購入する場合、信頼性の指定はどうすればいいですか。

A1. Web ショップでは、そのような高品質水準の製品を販売していません。輸送機器や交通システムなどには、特別水準の製品が必要ですが、製品によってオーダ・コードや保証レベルが異なるため、個別に対応をしています。特別水準や特定水準については、販売代理店を通じてご相談願います。

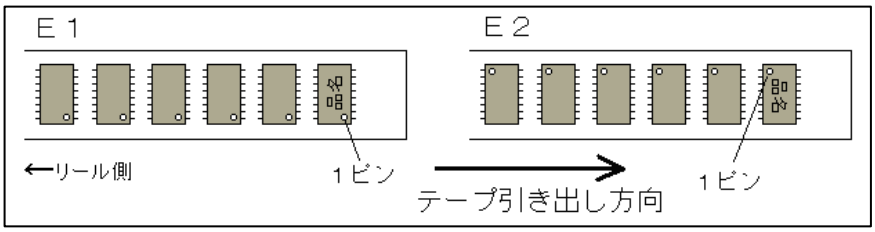
5.2 出荷形態

テーピング仕様

Q1. μPD43256BGU をリールで購入しますが、E1 と E2 の違いを教えてください。

A1. テープに張り付ける部品の方向を示しています。

E1 は引き出し方向から見て 1 ピンが手前、E2 は逆にリール側が 1 ピンです。



Q2. ディスクリート製品のリールで、E1、E2 のパッケージのキャリア方向はどのようになっているのですか。

A2. パッケージで異なります。下記Webページで該当パッケージ名の[テーピング >]をクリックし、ご参照ください。

http://www2.renesas.com/discrete/ja/package/smd.html

なお、光デバイスと RF/マイクロ波デバイスについては、下記ページに掲載の各資料をご参照ください。

フォトカプラ　　：テーピング仕様

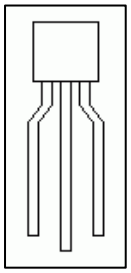
RF/マイクロ波デバイス：「ミニモールド・パッケージ 高周波トランジスタ製品のテーピング仕様／包装荷姿図」の「2.2」

http://www2.renesas.com/microwave/ja/document.html#pkg

Q3. TO-92 や SP-8 などのリード挿入品のオーダ品名で、"-T"はどういう意味ですか。

A3. ラジアル・テーピング品で、右図のようにリード先端部の間隔が通常品(バラ品)と異なります。

詳細寸法については、販売店にお問い合わせください。



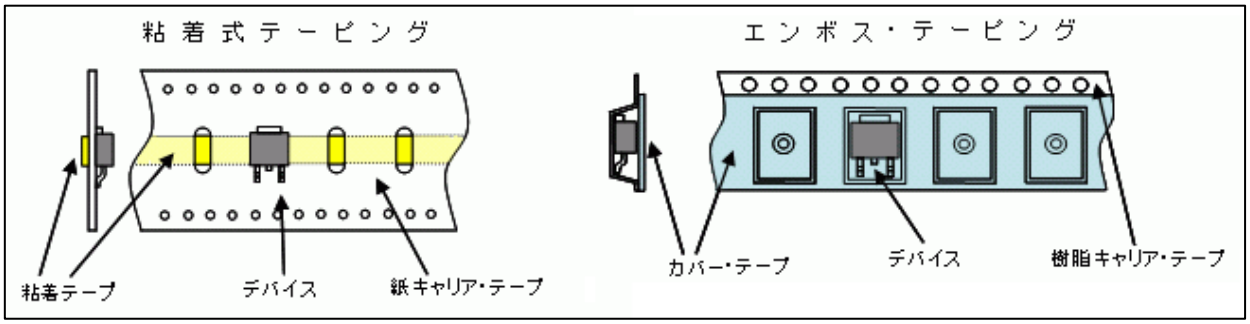
Q4. テーピング仕様のエンボス・テーピングと粘着式テーピングは、どのように違いますか。

A4. エンボス・テーピングは、キャリア・テープのくぼみ部分にデバイスを搭載して、フィルムでカバーしています。

粘着式テーピングは、テープ粘着部にデバイスを固定しており、デバイスは露出の状態です。

なお、粘着式テーピングは一部の小型パッケージに使用していましたが、現在は使用していません。

テーピング・イメージ図



包装材の静電気対策

Q1. テーピング、トレイ、マガジン、袋等の包装材は静電気面の配慮がなされていますか。

A1. プラスチック・マガジンは、表面に帯電防止剤を塗布しています。
トレイは、導電性トレイ(カーボン製練り込みプラスチック材)を使用しています。
袋は、静電気拡散性のアルミ・ラミネート袋または透明プラスチック袋を使用しています。
エンボス・テーピング品については、静電気拡散性キャリヤ・テープ、リールを使用しています。

包装材の表面抵抗率

当社では、上記包装材の表面抵抗率を $10^{12}\Omega$／□以下に規定しています。
これは、ANSI/EIA541 で規定している「静電気拡散性 10^{5}～$10^{12}\Omega$／□」に準拠しています。

Q2. 2SD1033 が納入時に静電用の袋に入っていませんでしたが、品質上の問題はないですか。

A2. 静電耐量の点では、電界効果トランジスタ(FET)よりバイポーラ・トランジスタが格段に優ります。このため、FETは帯電防止袋に入れて出荷しますが、バイポーラ・トランジスタは特に静電対策をしていません。

トレイ／テープ／リールの搭載個数

Q1. 1 つのトレイ／テープ／リールのデバイス搭載数は何個ですか。

A1. 概略は、次の手順でホームページからパッケージごとに分かります。正式には販売店にご確認ください。

[ICの場合]

①製品のデータ・シートなどで、パッケージ名やパッケージ・コードを調べます。
たとえばμPC4742G2 なら、パッケージは「8 ピン・プラスチック SOP」、コードは「8GM-50-255B-6」です。

②次のホームページでパッケージ名をクリックして、そのページに飛びます。
http://www2.renesas.com/pkg/ja/icpkg/pt_01.html
たとえばμPC4742G2 なら、P-SOP をクリックします。

③そのパッケージのページの上部にピン数の表がありますので、該当するピン数をクリックすると、そのピン数の表に飛びます。
たとえばμPC4742G2 なら、8PIN をクリックして、2 種類の「8 ピン・プラスチック SOP」の表に飛びます。

④パッケージ・コードの行のマガジン／テープ(リール)／トレイの欄をクリックして、表示内容を見ると、搭載個数が記述されています。
なお、パッケージによっては欄に 2 種類以上が表示されている場合があり、それは製品ごとに異なりますので、詳細は購入先にご確認ください。
★ パッケージ・コード「S8GM-50-255B-6」の行で、マガジンなら H5681-4 をクリックして、表示内容に 100 個と記述されています。Embossed Tape なら H5351-2 をクリックして、表示内容に 2500 個と記述されています。

[トランジスタやダイオードなどのディスクリート製品(表面実装)の場合]

①製品のデータ・シートなどで、パッケージ名を調べます。

②次のホームページで、そのパッケージ名の[テーピング >]をクリックして、表示内容を見ると、搭載個数が記述されています。
http://www2.renesas.com/discrete/ja/package/smd.html

リールの耐熱性

Q1. 半導体チップ部品のリール部品を、ベーキング（125 度，10 時間）するとリールが溶けて変形してしまいます。
リールの耐熱温度とベーキング条件、リールの変形対策などを教えてください。

A1. 粘着テープ、エンボステープ、リールは耐熱仕様ではありません。
ベーキング処理をする場合は、耐熱性容器に移し替えてください。その場合、テーピング実装できません。
ほとんどのテーピング品は耐湿包装をしていませんので、ベーキング条件は規定していません。
なお、包装の取扱注意事項をホームページに掲載していますので、「2.3.5 ベーキング」をご参照ください。
★ 製品情報 > パッケージ > パッケージ・包装関連情報 > 包装（旧 NEC エレクトロニクス） > ベーキング

ラベル

Q1. 梱包ラベルには、「サイズ、記載内容、文字サイズ」などの規格があるのですか。

A1. 個装箱、内装容器でのラベルの大きさ、文字の大きさなどは規定していません。なお、表示内容とバーコードの種類、フォーマット概要に関する標準と呼ばれるものは、国内外でいくつかあります。
日本：バーコード・ラベル・システム（C-3 ラベルなど）（JEITA）
米国：個装容器ラベルの標準（EIA624）
欧州：個装容器ラベルの標準（EDIFICE）

Q2. ルネサス エレクトロニクスでは、どのようにラベル・フォーマットを決めているのですか。

A2. JEITA（電子情報技術産業協会）などの定める標準を尊重しつつ、次のように独自に定めています。
■ラベルの大きさ
パッケージなどに合わせて、いくつかのラベルを用意しています。
・包装容器の最小のものに合わせたラベル
・小さなリールにあわせたラベル
・個装箱に貼るラベル
■記載内容
・当社グループ内管理用に必要な情報
・Pb-Free ナドの環境マーク
■バーコードの種類
一次元バーコードは、世界のデファクトである CODE39 を使用
二次元バーコードは、世界のどこでも読める PDF417 を使用（QR コードを使用していることもあります）

5.3 相当品

保守品／廃止品など

Q1. 手元にある製品が入手困難になりました。代わりに使えそうな製品はありますか。

A1. 保守化に伴う代替品のご相談は販売店でご相談を承ります。セールス部門の担当者へご相談ください。

参考 保守期間は通常 6 ヶ月程度です。保守期間を過ぎると廃止となります。
生産打ち切り時期は販売店へご確認ください。
該当製品をお使いの場合は、販売店とご相談の上、必要数量を確保願います。

Q2. μPD7105x シリーズ(マイコン周辺)の相当品はありますか。

A2. μPD71051,71054,71055 については、QFP パッケージの機能相当品があります。
詳しくは、上記リンク先の各データ・シートをご確認の上、販売特約店にお問い合わせください。
また、これらの DIP パッケージについては、サンハヤト社から機能互換モジュールが提供されています。
詳しくは、同社にお問い合わせください。

Q3. μPD7105x シリーズの QFP 相当品は、端子名称が uPD7105x と異なりますが、機能や正負の論理は同じですか。

A3. はい、機能相当品なので同じです。相当品の場合、負論理の端子名称では、末尾に"B"を付加しています(CLK 端子にはありませんが、立ち下がりエッジ・トリガになっています)。
なお、μPD7105x で NC 端子を外部信号のブリッジなどに使用していた場合、相当品では IC 端子なので短絡することになり、同一基板ではパターン・カットと信号線のバイパス接続が必要となります。

Q4. μPD72001(MPSC)の相当品はありますか。

A4. 当社には、代替できる相当品はありません。下記会社で、サブセット機能の MPSC モジュールが提供されています。詳細は同社にお問い合わせください。

株式会社テクノクリエート
http://www.techno-create.com/

他社相当品

Q1. 他社の製品〇〇に近い製品を製造していますか。

A1. 他社製品の情報は持ち合わせておりませんので、お客様が必要とする仕様をご提示ください。ご希望の仕様に近い商品をご案内します。

5.4 輸　出

★

法　令

Q1. 日本からの輸出について、関係法令はどこかで閲覧できますか。

A1. 経済産業省のホームページに掲載されています。
外郭団体である CISTEC(システック)の下記ページで、「輸出管理基本情報→関係法規」のリンク先に、サイト・リンク集が掲載されています。
http://www.cistec.or.jp/

非該当品リスト

Q1.非該当証明書が公開されていると聞きました。どこにあるのでしょうか。

A1. 非該当証明書については、販社経由で個別に対応しており、公開していません。
公開しているというのは、CISTEC 発行の「公表リスト」のことと思われます（下記で購入可）。

（財）安全保障貿易情報センター（http://www.cistec.or.jp/）
〒105-0001 東京都港区虎ノ門 1-1-21　新虎ノ門実業会館　（電話：03-3593-1147、FAX：03-3593-1136）

パラメータ・シート

Q1. 日本からの輸出で、パラメータ・シートが必要と聞きましたが、発行していただけますか。

A1. CISTEC「公表リスト」をご参照ください。ここに公表済みなら、個別のパラメータ・シート提出は不要です。
輸出時の通関には、インボイスに公表年月を表示してください。
CISTEC「公表リスト」に掲載されていない製品については、購入先を通じてパラメータ・シートを入手のうえ、これを通関業者にご提出ください。
なお、規制対象外の製品については証明書が不要なので、パラメータ・シートを用意していません。

ユーザ判定

Q1. 輸出貿易管理令によるマイコンの判定は、どのように考えればいいですか。

A1. マスク ROM 品と書き込み済み OTP 品は、プログラムの書き換えができないので、プログラムの仕様設計者であるお客様自身が該非判定をしなければなりません。お客様が判定する際は、専用集積回路（ある特定装置に使用するために設計）として判定されることになります。ご不明な点は、購入先にご相談ください。

フラッシュ・マイコン

Q1. 輸出貿易管理令による EPROM やフラッシュ・メモリ内蔵マイコンの判定は、どのようにするのですか。

A1. 書き換え可能なフラッシュ・メモリ品や EPROM 品は、本体（ハード）を汎用集積回路として判定します。これらのメモリに書き込まれたプログラムは、別に判定が必要です。ご不明な点は、購入先にご相談ください。

マイコンの性能

Q1. 日本の輸出規制で、デジタル電子計算機の性能尺度は「加重最高性能(APP:Adjusted Peak Performance)」ですが、ルネサス エレクトロニクスのマイコンの APP 値を教えてください。

A1. 当社では、該非判定の目安として「加重最高性能(APP)」の概算値の使用を推奨しています。
64 ビットと 32 ビットのマイコンの APP 概算値は 0.01WT(Weighted Tera FLOPS)です。
16 ビット以下のマイコンでは、浮動小数点オペランド計算を実行する機能がないので 0WT です。

Q2. APP 値について、なぜ概算値の使用を推奨しているのですか。

A2. CISTEC ジャーナル誌(2002.3 号)の Q&A の趣旨に基づきます。当社独自の判断によるものではありません。

6. 基礎知識

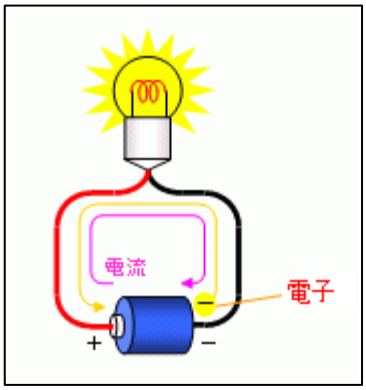
6.1 電　気

電気とは

Q1. 電気とはなんですか。

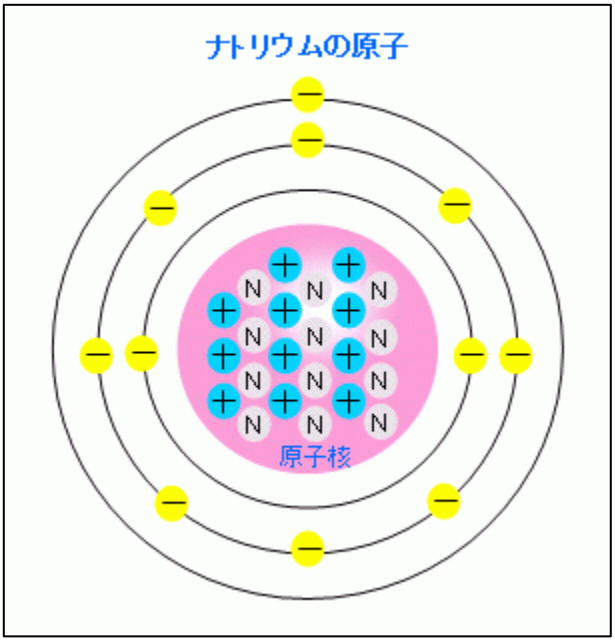
A1. 電気の実体は「電子」です。

電気はプラス(+)からマイナス(−)へ流れると言われますが、実際は電子がマイナスからプラスへ移動します。これが発見されたときには、すでに「プラスからマイナス」が世の中で定着していたので、混乱を避けるため、「電流はプラスからマイナス、電子はマイナスからプラスに流れる」という定義になってしまいました。

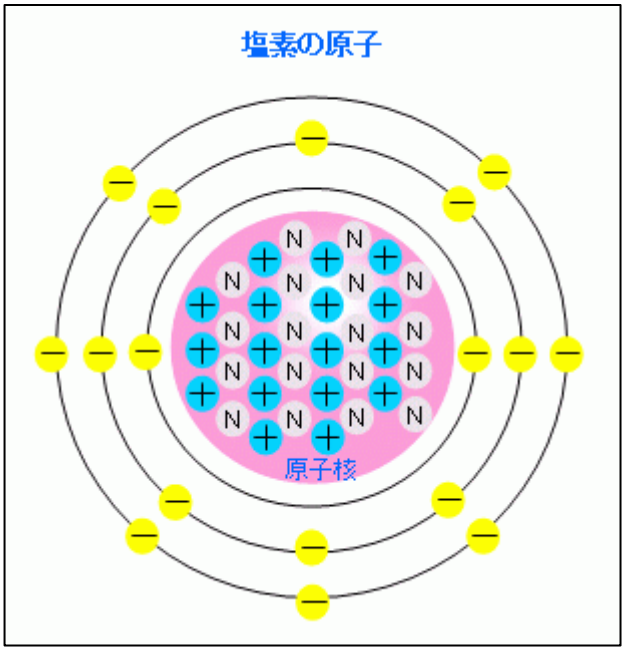


電子は、あらゆる物質（固体、液体、気体）に含まれています。物質を構成する原子そのものが、陽子(+)と中性子からなる原子核と、それを取り巻く電子(−)でできています。外力などによって、原子の電子が飛び出した場合、これが電気となります。本来、陽子と電子は同数ですが、電子(−)が飛び出した原子は、電気的には陽子(+)が過剰になるので、プラス・イオンとなります。逆に電子が飛び込んだ原子は、電気的には電子(−)が過剰になるので、マイナス・イオンとなります。

たとえば、ナトリウム原子（Na：原子番号 11）は、それぞれ 11 個の電子と陽子、12 個の中性子でできています。

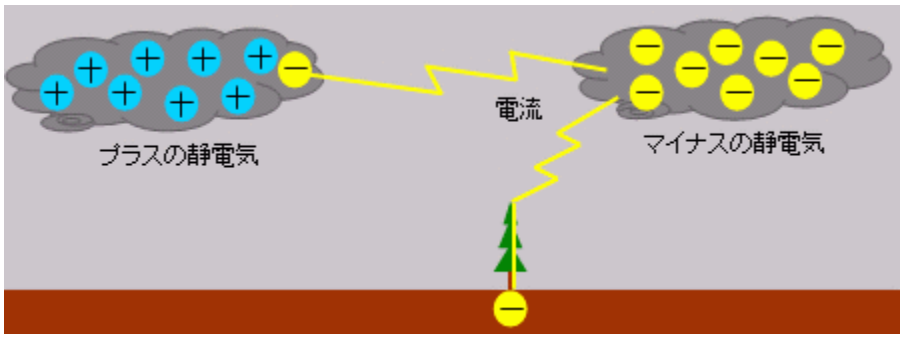


ここで、最外核の電子は1つなので、これが飛び出せば安定します。
また、塩素原子（Cl：原子番号 17）は、それぞれ 17 個の電子と陽子、18 個の中性子でできています。



ここで、最外核の電子は 7 つなので、ここに電子が飛び込めば安定します。
したがって、ナトリウム Na と塩素 Cl は容易にイオン結合して、NaCl（塩）になります。これを水に溶かすと、安定したままナトリウム・イオン Na^+と塩素イオン Cl^-に電離（イオン化）した塩水となります。ナトリウムはアルカリ金属の一種なので、ナトリウム・イオン Na^+を含む水は、アルカリ・イオン水と呼ばれます。
プラスやマイナスの性質を帯びることを「帯電する」と呼びますが、これはプラスの電荷とマイナスの電荷によるものです（電荷 Q の単位はクーロン[C]）。1つの電子の電荷量は $e^- = 1.6 \times 10^{-19}$[C]です。

塩のように原子が結合した場合は、電子が内部で安定してとどまりますが、電子が原子から飛び出した場合、自由電子となります。これが電子の足りないところへ流れると電流となり、流れる先がなくてとどまると静電気になります。
自然界の現象では、雷がその例です。これは、マイナスに帯電した雲（水蒸気）から電子が、地表やプラスに帯電した雲に放電されるのです。



★ **Q2.** 人工的には、どのように電気を発生させているのですか。

A2. ひとつには、磁界中でタービンに巻いた導線コイルを回転させ、フレミングの右手の法則に従う電流を発生させる方法があります。この原理は、発電所の発電機から、自動車や自転車のダイナモ、懐中電灯や携帯ラジオの手動発電機など、広く利用されています。発電所の動力源としては、水力・火力・原子力・風力などがあります。タービンの回転で起電方向が周期的に変わるので、交流発電となります。
もうひとつは、発光ダイオードの逆原理で、PN 接合の半導体に光をあてると、そのエネルギによって電子が空乏層を越えて導通するようになります。この原理は、ソーラ発電や光半導体の受光素子などに利用されている直流発電です。また、熱を電気エネルギに変換する例もあります。

なお、簡便に利用される電源として電池（バッテリ）があります。化学実験で習うボルタ電池（ガルバニ電池）が起源で、電解液に 2 種類の金属を浸し、化学反応で陰極にイオン化した電子を発生させる直流発電です。使いきりの一次電池と、充電して繰り返し使える二次電池があります。次表に、電池の例を示します（素材はメーカによって異なることがあります）。

分類	名称	陽極	陰極	電解液	公称電圧
一次電池（乾電池）	ボルタ電池(非商用)	銅	亜鉛	硫酸	
	マンガン電池	二酸化マンガン	亜鉛	塩化亜鉛	1.5V，9V
	アルカリ・マンガン電池	二酸化マンガン，黒鉛	亜鉛	水酸化カリウム，塩化亜鉛	1.5V，9V
二次電池（蓄電池）	鉛蓄電池(車載など)	二酸化鉛	鉛	希硫酸	2V/セル
	リチウム・イオン電池	酸化コバルト・リチウム	グラファイト	炭酸エチレン	3.7V
	ニッケル水素電池	水酸化ニッケル	水素吸蔵合金	水酸化カリウム水溶液	1.2V
	ニッケル・カドミウム電池	水酸化ニッケル	水酸化カドミウム	水酸化カリウム水溶液	1.2V

また、これらの金属反応ではなく、燃料と酸化剤（たとえば水素と酸素）で電気を生成する燃料電池があります。特に最近は、再生可能なクリーン・エネルギに対する需要が高まっているため、生体触媒（酵素）で食物（生ゴミのバイオマスなど）の糖やアルコールを分解して水素を生成し、電気エネルギを取り出すバイオ燃料電池の研究が進められています。

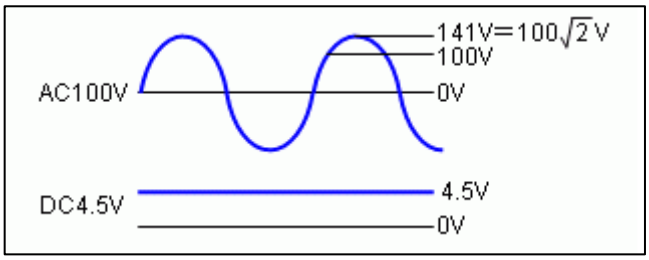
電気に関する単位

Q1. 電気に関する単位の意味を教えてください。

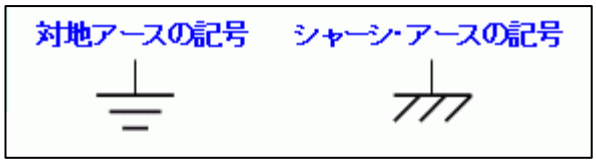
A1. 電気関連では、おもに電圧 V（単位[V]：ボルト）、電流 I（単位[A]：アンペア）、抵抗 R（単位[Ω]：オーム）、電力 P（単位[W]：ワット）、周波数f（単位[Hz]：ヘルツ）といった項目が基礎パラメータです。

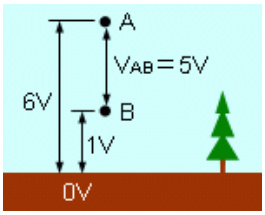
(1) 電圧

電圧とは、電荷の位置エネルギです。ですから、電圧は高い／低いと表現します。水にたとえると、水深の深いところで水圧が高くなったり、高低差の大きな水の流れのエネルギが大きくなるようなものです。
物質に電圧を加えることを「電圧をかける」とか「電圧を印加する」と言います。
なお電気には、時間の経過とともに極性（±）が変動する交流（AC：Alternative Current）と、極性変動のない直流（DC：Direct Current）があります。電灯線の AC100V というのは正弦波交流で、実効値が±100V、ピーク値は±141V です。これに対して、電源アダプタ出力の DC4.5V などは直流です。



電圧というのは、相対的な概念です。たとえば、半導体 IC の V_{DD}、GND 端子に電源を供給する場合、GND＝0V とみなして、V_{DD}＝3.3V というのはそれより 3.3V 高い電圧を印加することを意味します。絶対的な電圧は電位で定義され、地球（対地アース）の電位を 0V とします。先の例の GND は装置の筐体（シャーシ・アース）の電位で、0V 電位とは限りません。2 点間の電圧は電位差であり、一般に用いられている電圧というのは、この電位差のことです。



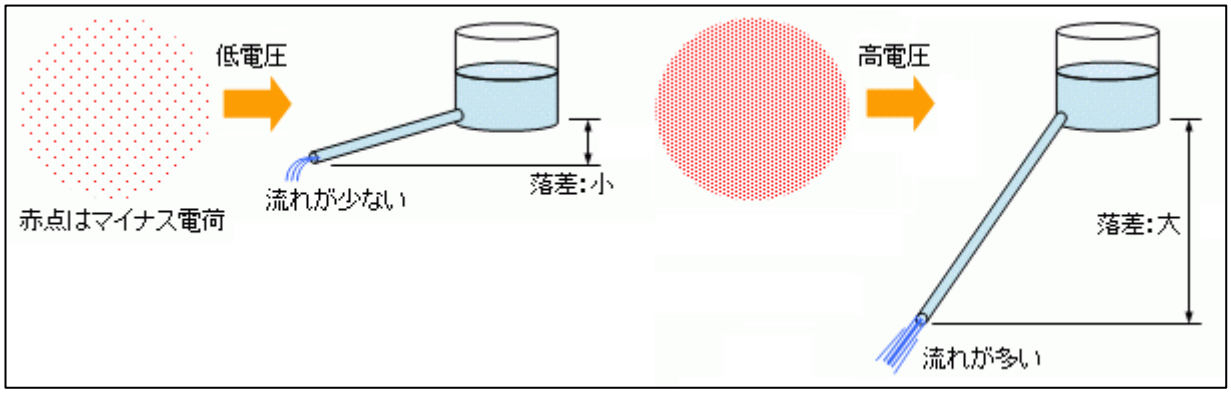


(2) 電流

電流は、単位時間に電気（電荷 Q）の流れる量を表します。

電流 I＝*dQ/ dt*

電圧が高いほど、電流が多くなります。



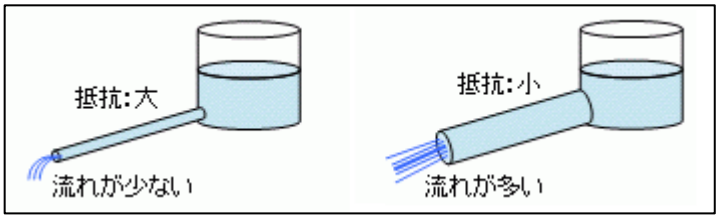
人体に 1A の電流が流れると死にいたると言われていますが、この 1A というのは次のような値です。

1m 離れて平行に 1A 流れる電流の間には、1m あたり 2×10^{-7}N（ニュートン）の力が働く

なお、「消費電流」とか「電流を消費」という表現は誤りです（「(4)電力」参照）。

(3) 抵抗

抵抗は、電気の流れにくさです。同じ電圧でも、抵抗が小さいほど、電流は多く流れます。



電気関係の基礎的な法則に、オームの法則があります。

オームの法則 ： 電圧 V(V)＝電流 I(A)×抵抗 R(Ω)

(4) 電力

電力は、電圧と電流の積です。

電力 P(W)＝電圧 V(V)×電流 I(A)

たとえば 40W の電灯線用電球には、40/100＝0.4A の電流が流れます。

電力会社の使用量明細では、kWh という単位を使用していますが、これは1時間当たりに消費する電力を累積した積算電力量です。40W の電球を1時間使用すると、40Wh という電力量になります。1500W の熱器具を 3 時間使用した場合の電力量は、4.5kWh になります。

ところで、電力は「消費する」という表現をします。「消費電流」という表現は誤りで、電流は消費してなくなることがなく、ループして戻るか、行き先に到達します。「消費」に対しては、「消費電力」でなければなりません。電流については、「定格電流」とか「電源電流」、「動作電流」などの表現が用いられます。

【ティー・ブレーク】

実効値というのは、抵抗負荷 R に正弦波交流の電気を供給したときに、直流と同じ平均電力（有効電力）を消費する電圧と電流の表現方法です。通常、交流の電圧と電流は実効値で表され、これによって、電力計算を直流と同様に行うことができます。電圧と電流のピーク値を Vp、Ip として、算出しましょう。

電流 $i(t)=Ip\cdot\sin\omega t$

電圧 $v(t)=R\cdot Ip\cdot\sin\omega t$

電力 $P(t)=R\cdot i(t)^2=R\cdot Ip^2\cdot\sin^2\omega t=R\cdot Ip^2\cdot(1/2)(1-\cos2\omega t)$

平均電力は、これを1周期分だけ積分して、周期 T で除算したものです。

$$平均電力Pr=\frac{1}{T}\int_0^T R\cdot Ip^2\frac{1}{2}(1-\cos2\omega t)dt=\frac{R\cdot Ip^2}{2T}\left[t-\frac{1}{2\omega}\sin2\omega t\right]_0^T$$

1周期分なので ωt＝2π より、第二項は積分すると 0 となるため、

$$平均電力Pr=R\left(\frac{Ip}{\sqrt{2}}\right)^2=\left(\frac{Vp}{\sqrt{2}}\right)\cdot\left(\frac{Ip}{\sqrt{2}}\right)$$

直流の電力計算式は、P＝V・I

そこで Ve、Ie（実効値）をそれぞれ Vp、Ip の「ルート 2 分の 1」と定義すると、直流と同じ計算式になります。

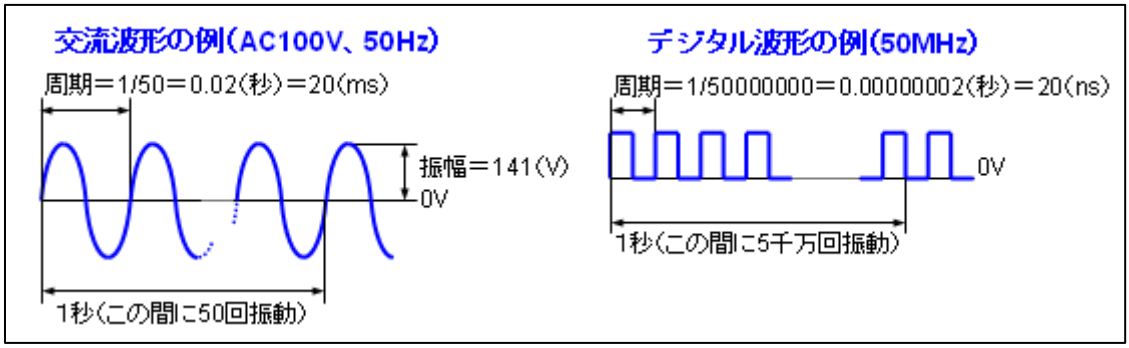
$$Ve=\frac{Vp}{\sqrt{2}}\qquad Ie=\frac{Ip}{\sqrt{2}}$$



(5) 周波数

周波数というのは、一定周期で変化する交流成分について、1 秒間に振動する回数を意味します。周波数と周期は逆数の関係です。

周波数f(Hz)＝1／周期 T(s)

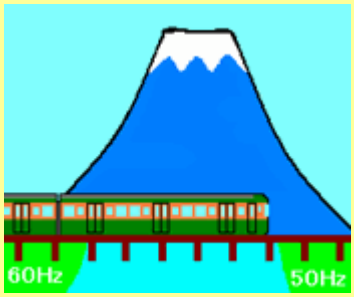


【ティー・ブレーク】

商用送電は、東日本が 50Hz、西日本が 60Hz ですが、これは発電機導入の歴史による文化の違いです。エジソンが世界初の電力供給システムを完成した翌年 1883 年（明治 16 年）に、東京電燈（現在の東京電力）が設立され、文明開化の波に乗って、日本でも一気に電化が進みました。当初、東京は直流、大阪は交流を支持していましたが、1895 年に東京も交流に切り替えることにして、浅草の火力発電所に、ドイツのアルゲネ社製 50Hz の交流発電機を導入しました。大阪電燈は米国のトムソン･ヒューストン社（のちに GE 社が吸収）製の 125Hz 発電機を導入しましたが、1897 年に GE 社製の 60 Hz 発電機に切り替えました。その後、神戸・京都・名古屋の電燈会社も GE 社製の 60 Hz 発電機を採用しました。このため、西日本は 60Hz になりました。

たとえば交流モータでは、回転数が周波数に依存して 20%の違いを生じるために、システムによっては切り替えが必要になります。東海道地域では、富士川をを境に周波数が分かれています。

最近は DC モータやインバータ制御の普及で、周波数を気にする必要性が少なくなっています。





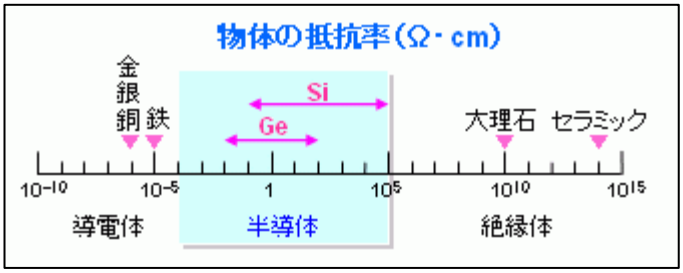
6.2 半導体

半導体とは …65
半導体デバイスの動作 …68
IC と LSI …73
MOS 製品と静電気 …74
半導体製品の種類 …75
化合物半導体 …82
光半導体 …83

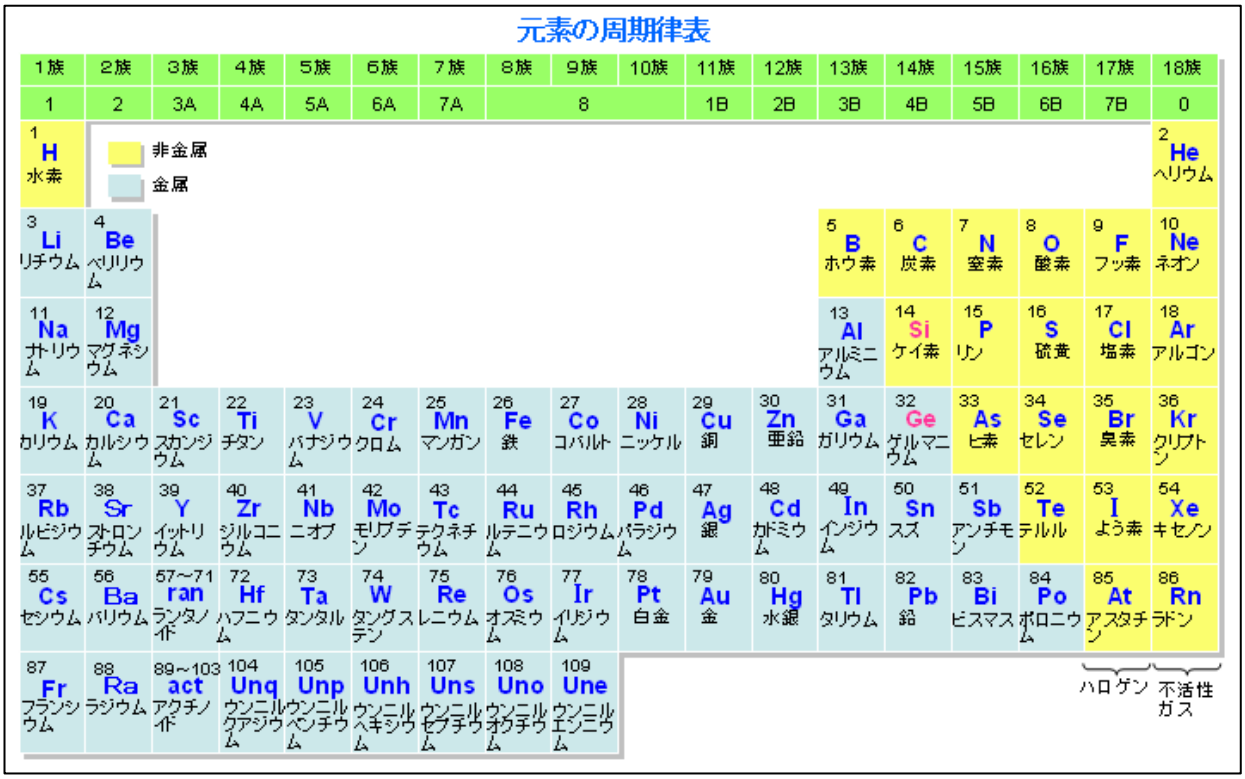
半導体とは

Q1. 半導体とはなんですか。

A1. 半導体(Semiconductor)というのは、導電体(Conductor)と絶縁体(Insulator)の中間の性質をもつ物質です。
導電体には電気が流れやすく、絶縁体には電気が流れません。絶縁体ではありませんが、半導体の元素にはほとんど電気が流れません。
金属などの導電体には自由電子が多く含まれ、これが容易に移動するため、電気が流れやすいのです。

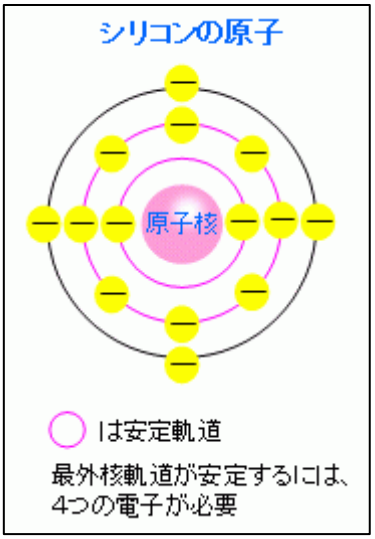


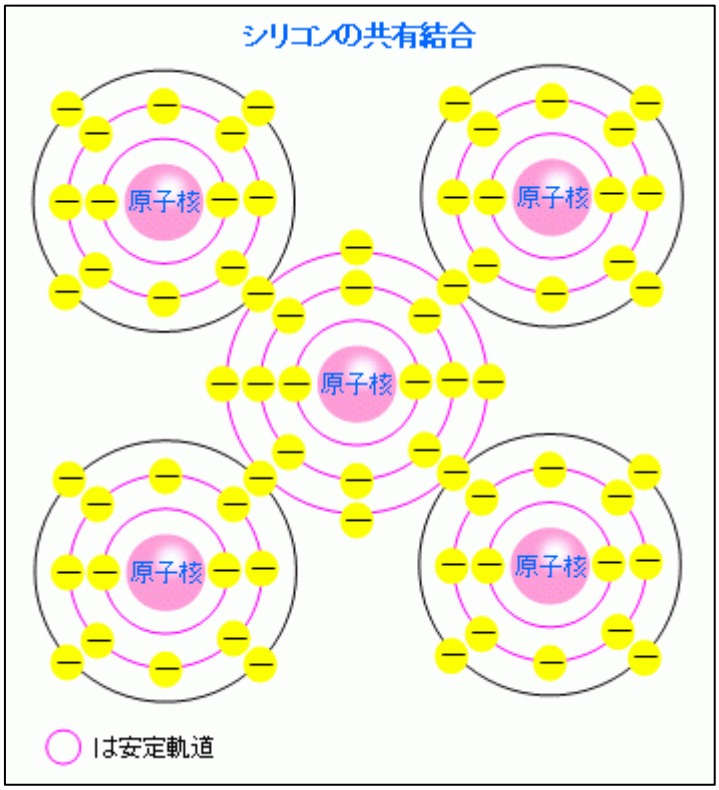
当初、半導体の元素としてゲルマニウム(Ge)を使用していましたが、現在はおもにシリコン(ケイ素:Si)が使用されています。

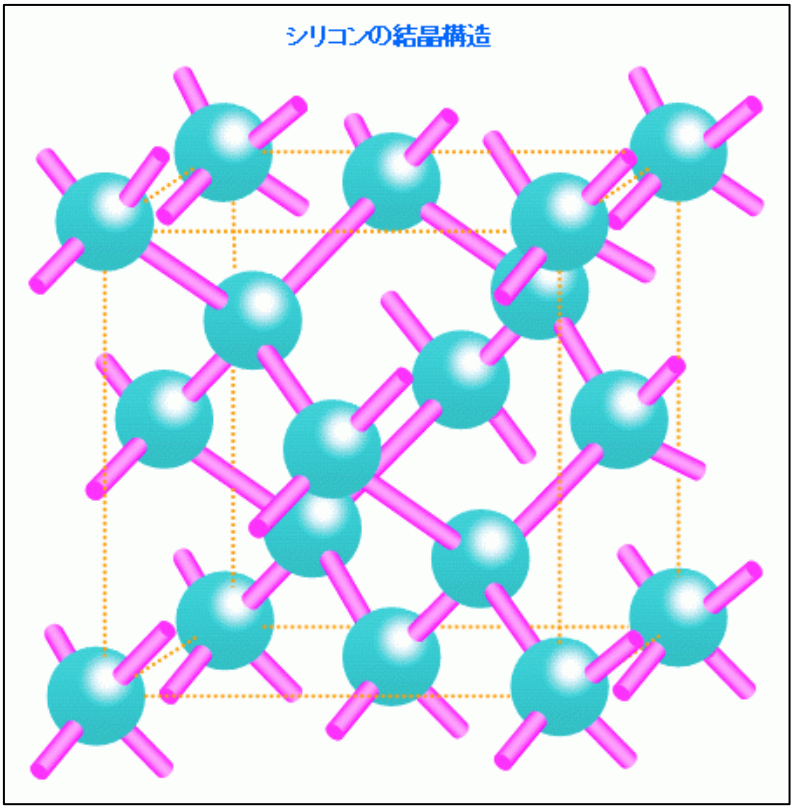


通常は半導体デバイスという表現をして、半導体を加工したものを指します。たとえば、純粋なシリコンにはほとんど電気が流れませんが、別の元素を混入(高温拡散によるイオン注入:ドーピング)することによって電気が流れるようになり、特性の異なるものを組み合わせて、いろいろな動作をさせることができます。半導体デバイスの製造工程については、ホームページの半導体ができるまでをご参照ください。

シリコン(Si)の原子は、最外核に電子(−)が 4 つあります。電子が 8 つで安定するため、周囲に 4 つのシリコン原子が共有結合すると、安定した結晶となります(電気は流れない)。





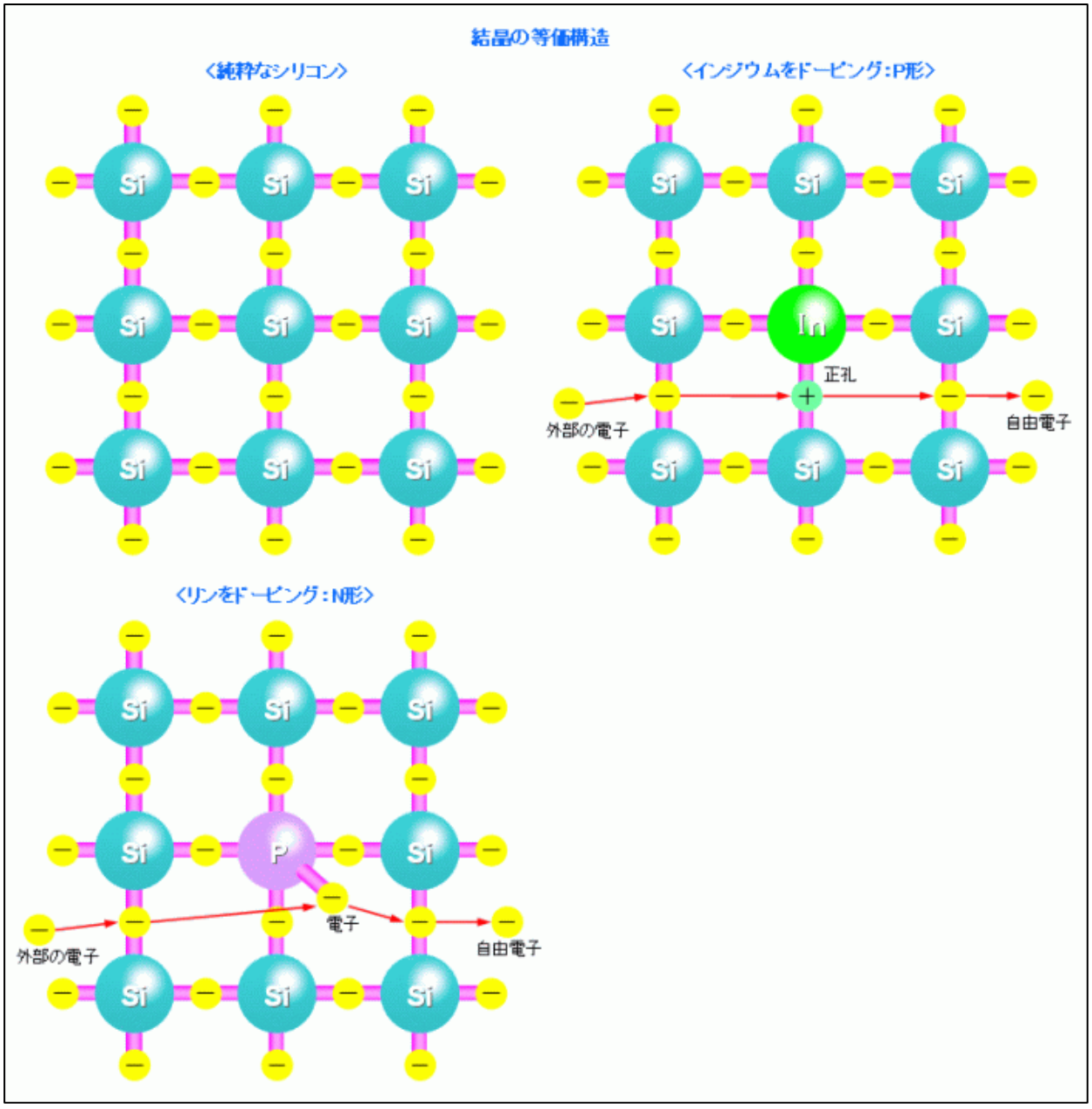


シリコンの原子は、最外核に電子(結合の手)が 4 つありますが、たとえばインジウム(In：結合の手が 3 本)をドーピングすると、シリコンに結合するための電子が 1 つ足りず、これを正孔(hole)と呼びます。ここに外部からの電子を取り込みやすく、またその電子を自由電子として他へ送りやすくなり、この性質を P 形(positive)と呼びます。

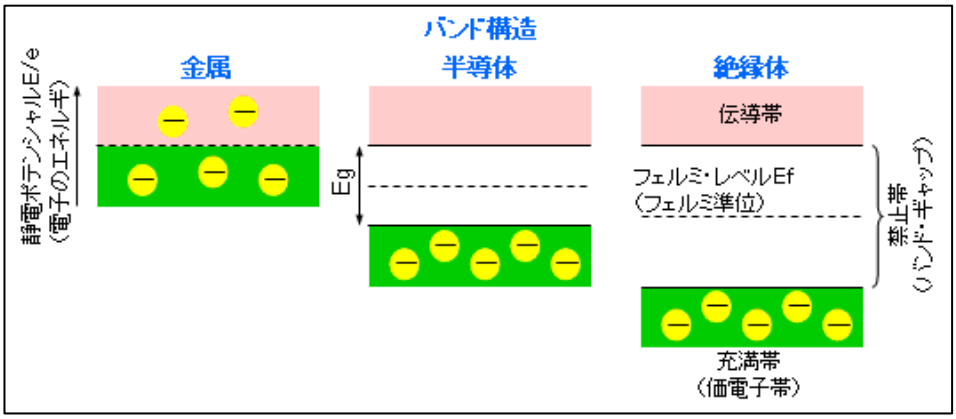
また、シリコンにリン(P：結合の手が 5 本)をドーピングすると、シリコンに結合しても電子が1つあまり、これを電子(electron)と呼びます。この電子を自由電子として他へ送りやすくなり、この性質を N 形(negative)と呼びます。

半導体デバイスは、これら P 形と N 形の組み合わせで構成されます。

なお、これら電子(−)や正孔(+)をキャリアと呼びます。

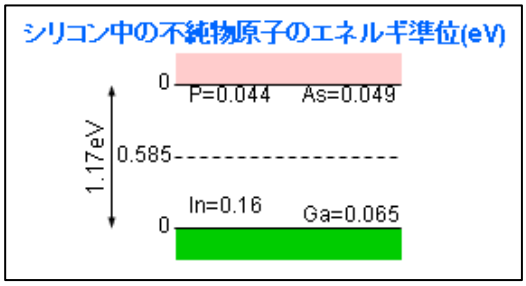


次図は、エネルギのバンド構造です。



金属では、充満帯の電子が価電子となって伝導帯へ自由に飛び出しますが、半導体や絶縁体では禁止帯（禁制帯）があるため、充満帯にとどまります。禁止帯の中心レベルはフェルミ・レベルと呼ばれ、電子が入りえる最大エネルギ・レベルです。

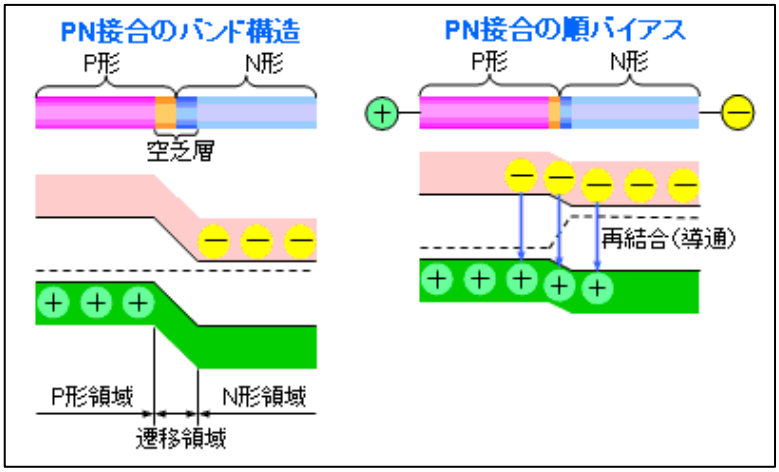
たとえばシリコン Si の禁止帯幅は 1.17eV（エレクトロン・ボルト:1.602×10^{-19}[J]）で、不純物をドーピングすることによってフェルミ・レベルをシフトさせます。半導体素子は、これをＰ形とＮ形で組み合わせ、電圧印加によって電子と正孔が再結合する原理を応用します。



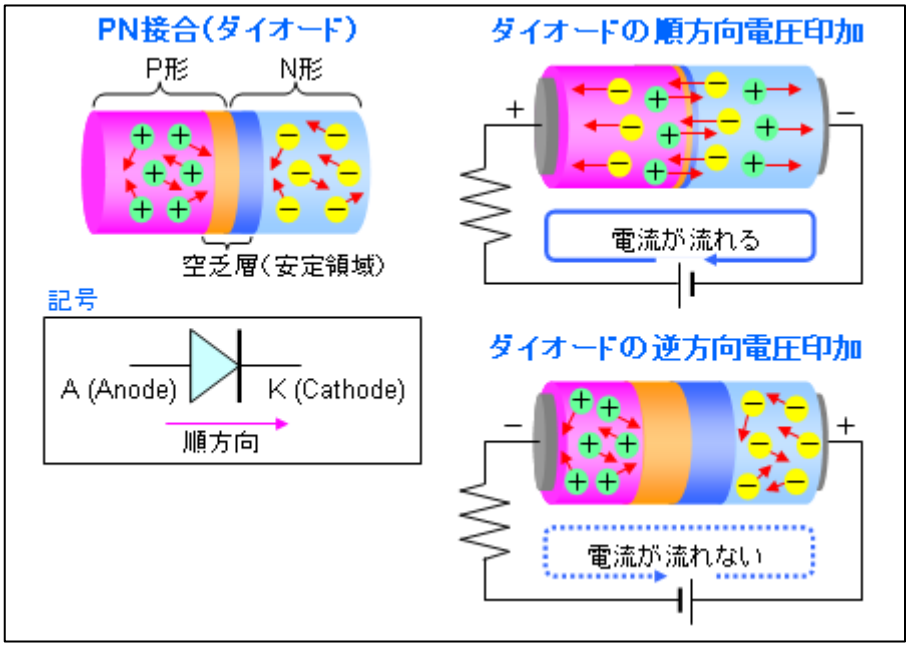
半導体デバイスの動作

Q1. 半導体デバイスの動作原理を教えてください。

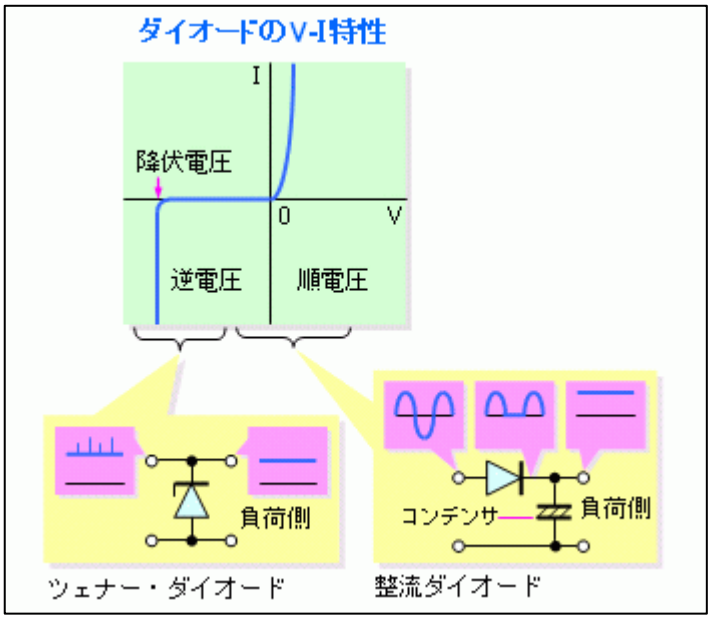
A1. Ｐ形とＮ形の半導体を接合すると、接合部分が中性の空乏層(depletion layer)という安定領域になります。この両端に、Ｐ型側に＋、Ｎ形側に－となるよう電池を接続すると、空乏層がせまくなって、キャリアが双方で移動できるようになり、電流が流れます。逆に、Ｐ 型側に－、Ｎ 形側に＋となるよう電池を接続すると、空乏層が広くなって、キャリアが双方で移動できず、電流が流れません。



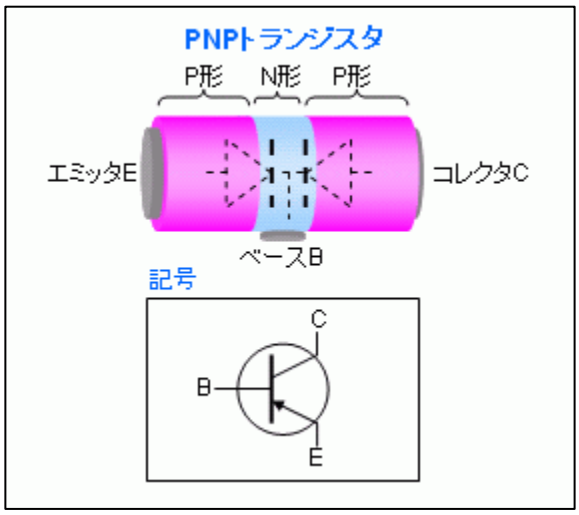
これが**ダイオード**の原理で、整流作用です。ダイオードというのは、Di-electrode(2 種類の電気の通り道)から命名されています。

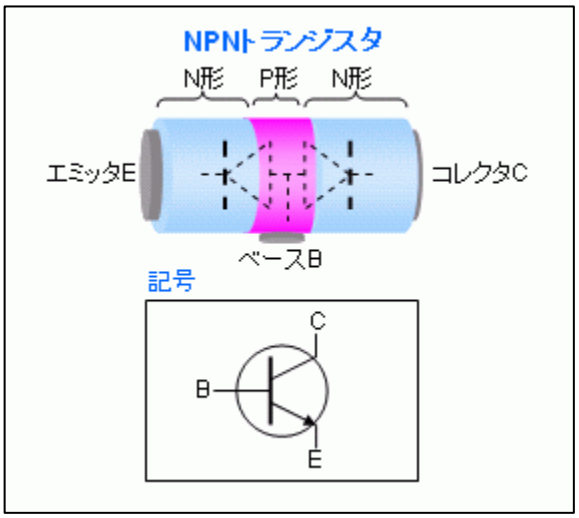


空乏層の両端の静電ポテンシャルの差を拡散電位(内蔵電位)と呼び、ここに不純物のエネルギ準位を差し引いた以上の順バイアス(順方向降下電圧)がかかると、電流が流れます。シリコン・ダイオードでは約 0.6～0.7V、ショットキィ・バリア・ダイオードで約 0.2V、発光ダイオードでは材料によって 1～5V 程度となります。
ただし逆方向に、空乏層を突き抜けるだけの電圧をかけると、電流が流れるようになります。この電圧は降伏電圧と呼ばれ、電流に依存せず一定(安定)になります。この電圧を定電圧として利用するのがツェナー・ダイオードです。したがって、ツェナー・ダイオードは対グランド間に逆方向に設置して、信号線の一定電圧以上のノイズを吸収するような利用をします。

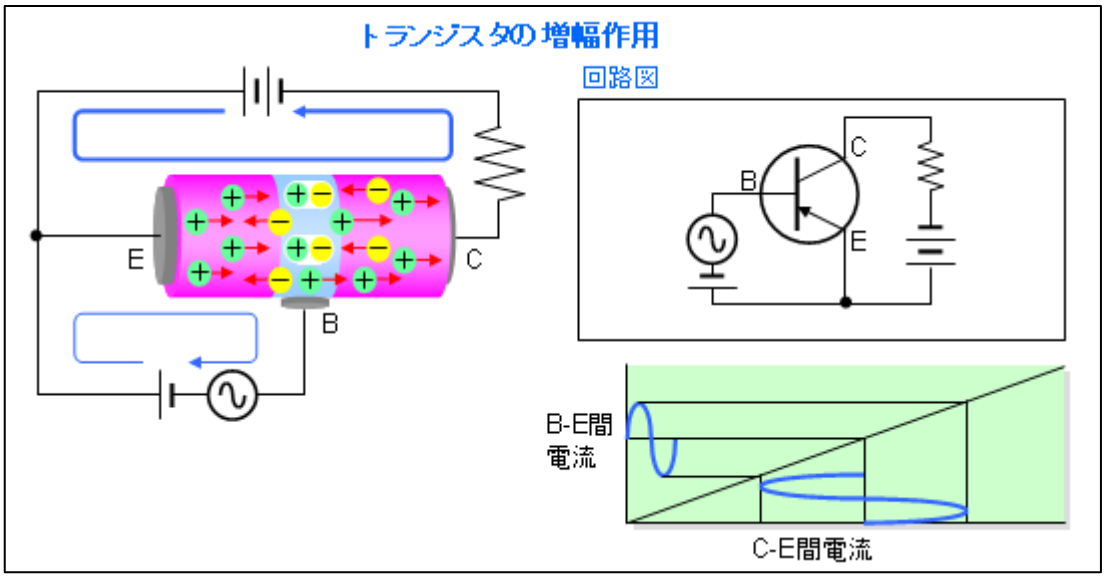


次に、P 形と N 形の半導体を交互に 3 つ接合すると、ダイオードを2つ向かい合わせ、または背中合わせにした構造になるので、両端の C-E 間には電流が流れません。これは**トランジスタ**の構造です。トランジスタとは、Transfer Resister から命名されています。また、キャリアが正孔と電子の両方なので、**バイポーラ・トランジスタ**と呼ばれます。バイポーラとは、2 つ(Bi)の極性(polar=polarity)にちなんでいます。

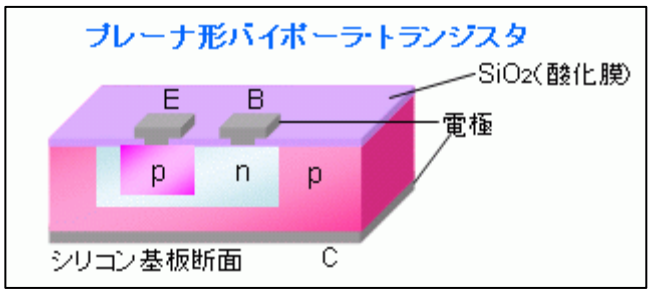




真ん中のベースの層は、ひじょうに薄くなっていて、キャリアと逆極性の電荷を加えると、容易に両端の C-E 間に電流が流れるようになります。このとき、能動状態にするために印加する電圧をバイアスと呼びます。ベース電流 I_B の変化が、C-E 間のコレクタ電流 I_C を大きく変化させます。これが、バイポーラ・トランジスタの電流増幅作用です。なお、エミッタ E を共通として接地（グランド接続）したときの、電流の増幅率（I_C/I_B）を h_{FE}（FE: Forward, grounded Emitter）で表します（昔は β だったが、現在は h パラメータのひとつとして統一）。

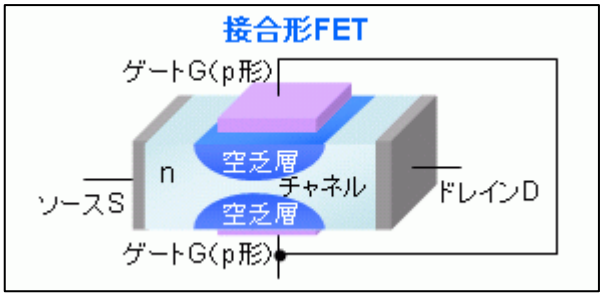


接合型トランジスタの例を紹介しましたが、現在では生産性や安定性などの点で、ICのようにシリコン基板上に順次拡散を繰り返して P 形と N 形を形成するプレーナ型が主流です。



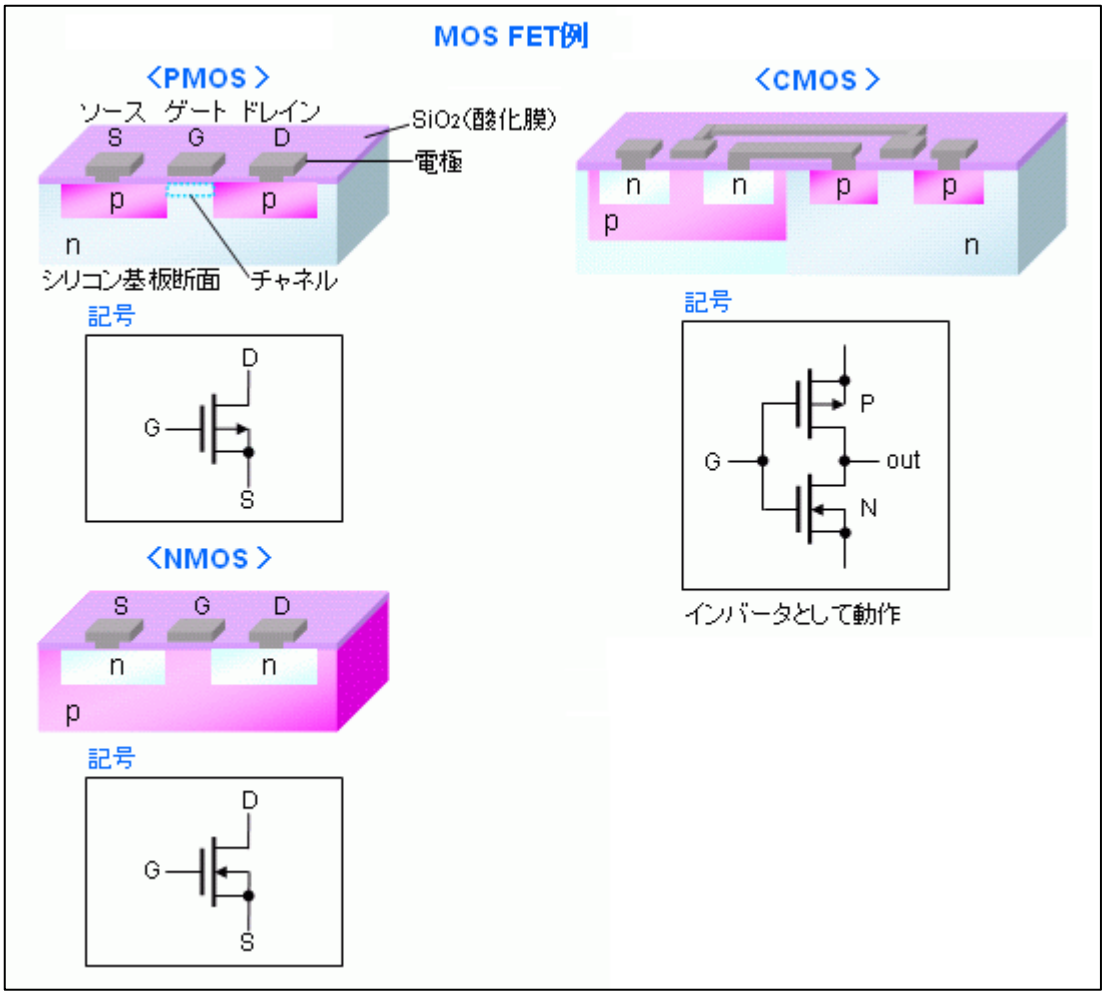
バイポーラに対して、ユニポーラ（1極性）のトランジスタもあります。つまり、キャリアが正孔か電子のいずれかのトランジスタです。このタイプのトランジスタは、電界効果トランジスタ（FET:Field Effect Transistor）と呼ばれ、接合型（J-FET）や絶縁型（MIS）などがあります。

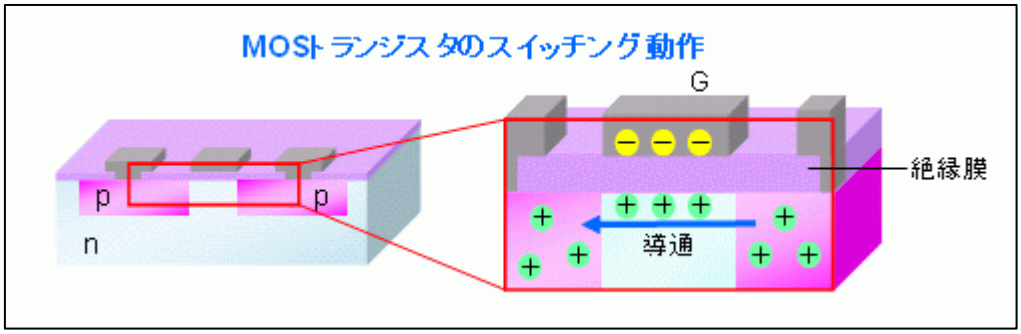
接合型 FET は、ゲート G で発生する電界によって空乏層の大きさが変化し、チャネル（通り道）の幅を変えて、ソース S-ドレーン D 間の電流が制御されます。電力用に向きますが、構造上、集積回路には不向きです。



絶縁型 FET では、ゲート電極が半導体に直接触れず、酸化膜（絶縁膜）をはさんで配置されます。このため、ゲート G には電流が流れず、またゲートと対向するチャネル部分が空乏層となって、S-D 間は導通しません。ゲート電位による酸化膜の電界によって、チャネルに電荷が蓄積されると、S-D 間が導通します。

この構造は、電極と半導体で絶縁体をはさんでいるため、MIS(Metal Insulator Semiconductor)構造と呼ばれ、絶縁体が酸化膜の場合には、MOS(Metal Oxide Semiconductor)といいます。したがって、このタイプのトランジスタを一般に、MOS FET または MOS トランジスタと呼びます。なお、チャネルが導通する極性によって、P チャネル MOS(PMOS)と N チャネル MOS(NMOS)があります。CMOS（Complementary MOS）というのは、PMOS と NMOS を組み合わせたものです。



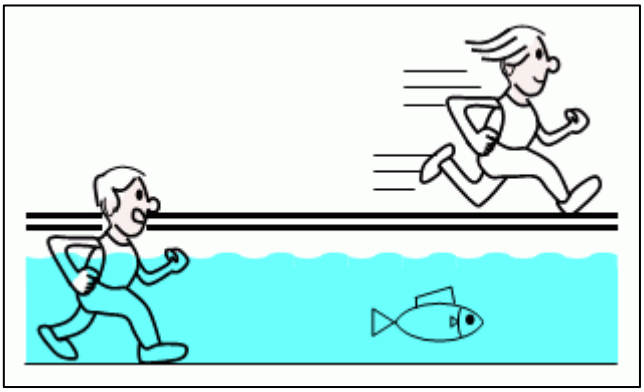


チャネル部分について、p-p または n-n の間隔を微細化の尺度となるチャネル長と呼びます。また、チャネルの奥行きの長さは、チャネル幅です。

Q2. 半導体デバイスで、高速のものと低速のものがありますが、どのような要因でちがうのですか。

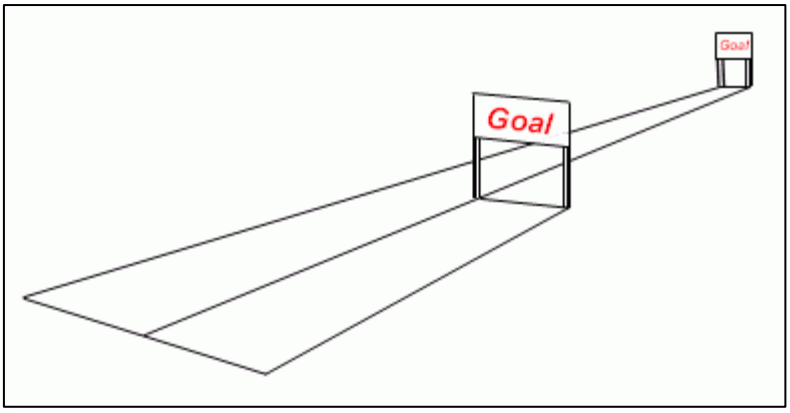
A2. デバイスの動作速度は、信号（電子）の伝達時間によって決まります。時間は速度に反比例し、距離に比例します。

同じ距離なら、速度の速いほうが、伝達時間が短くなります。伝達速度は、物性的な特性のちがいによって異なります。半導体デバイスは P 形と N 形で構成されますが、これらに含まれる元素の種類とその濃度によって、電子の**移動度（モビリティ）**が変わり、速度の違いを生じます。もちろん、配線や端子に使用される金属の抵抗率によっても移動度は変わりますが、デバイスの動作速度に影響するほどの差はありません。

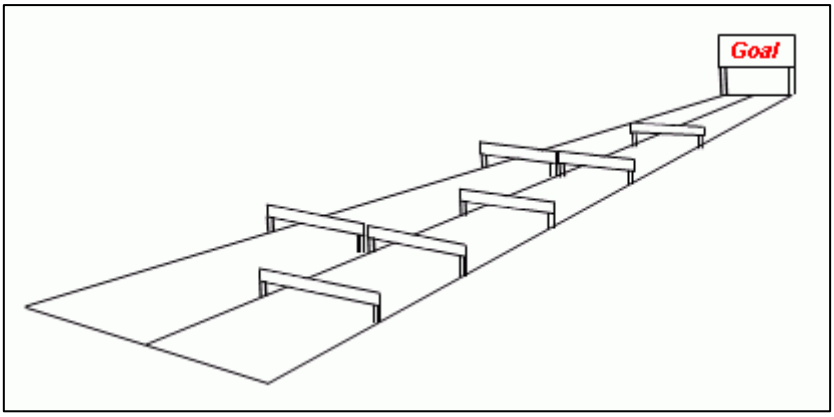


また同じ伝達速度なら、距離の短いほうが、伝達時間が短くなります。このため、**プロセス・テクノロジ**（0.35μm プロセスなどと表現され、この 0.35μm は**設計ルール**と呼ばれる）を**微細化**して、伝達距離を短縮します。設計ルールは、チャネル長（CMOS では P-ch と N-ch の間隔）または最小配線ピッチのことで、この微細化によって、全体のチップ・サイズが小さくなり、高速化とともに動作電圧が低くなる傾向があります。

ただし、最近のプロセスは 100nm（千万分の 1m）未満の領域となり、配線の容量や抵抗などが相対的に大きくなるため、単にプロセスの微細化による高速化は困難になっています。



さらに、**回路構成**によって動作速度は異なります。素子の段数が多くなれば、遅延が加算されるので、速度が低下します。特にクロック同期動作をする場合、クロック・サイクルごとにしか次の動作に移らないので、そのサイクル間に動作する回路規模によって動作速度が制限されます。



半導体デバイスの動作速度は、以上の要因が組み合わさって、製品ごとに異なります。

IC と LSI

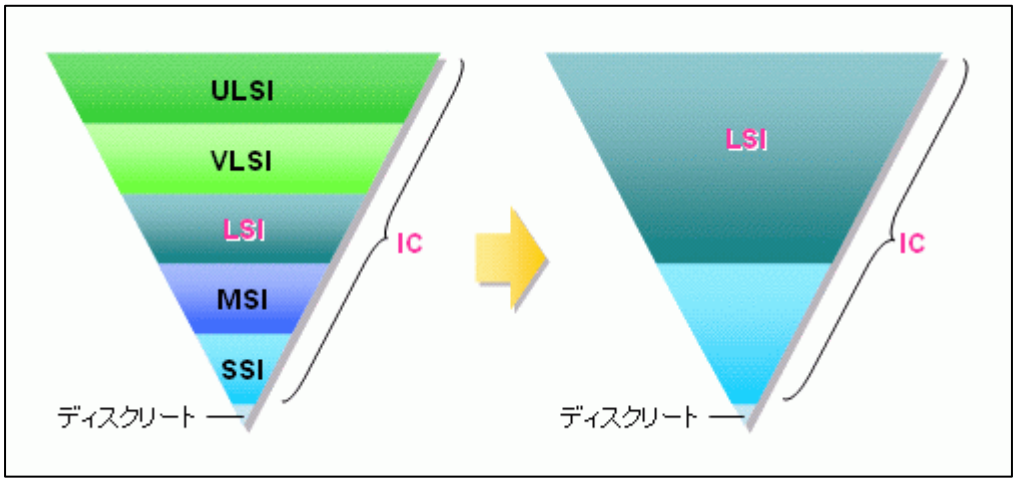
Q1. IC と LSI の違いはなんですか。

A1. 規模を表現するかしないかの違いだけで、機能や性能を表現しているのではありません。

IC は Integrated Circuit(集積回路)の略で総称であり、LSI は Large Scale Integration(大規模集積回路)の略です。

IC の大規模化が進み始めた 1970 年代に、規模によって SSI(Small Scale Integration)、MSI(Medium Scale Integration)、LSI、VLSI(Very Large Scale Integration)、ULSI(Ultra Large Scale Integration)などと分類されていましたが、高集積化はとどまるところを知らず、表現しきれなくなりました。今日では総称の IC と、数千トランジスタ以上を集積した LSI という表現が一般的に用いられています。

なお、集積化していない単体のトランジスタやダイオードは、ディスクリートと呼んでいます。

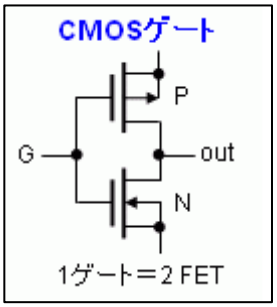


Q2. IC の集積度を高くすると、どのような利点があるのですか。

A2. まず、システム全体で見ると、部品点数が少なくなるので、**信頼性が向上**します。また、故障時の交換部品が特定しやすく、**保守が楽**になります。さらに、回路解析が困難になり、**機密性が高く**なります。
次にプロセス・テクノロジを微細化した場合、特性面では**高速化**が期待できます。なおこの場合、動作電圧は低くなる傾向があり、**消費電力や発熱の低減**（ひいては CO_2 削減）も実現できます。また、チップ・サイズが小さくなれば、小さなパッケージに収めることができ、**実装面積を縮小**できるので、**システムの小型化**に寄与します。
ただし、最近のプロセスは 100nm（千万分の 1m）未満の領域となり、配線の容量や抵抗などが相対的に大きくなるため、単にプロセスの微細化による高速化は困難になっています。また微細化すると、スタンバイ時のオフリーク電流などが増加するため、単にプロセスの微細化による低消費電力化も困難になっています。
当社では、High-K ゲート絶縁膜などの採用により、大幅な低消費電力化を実現しています。

Q3. 集積度を表すトランジスタ数とゲート数はどうちがうのですか。

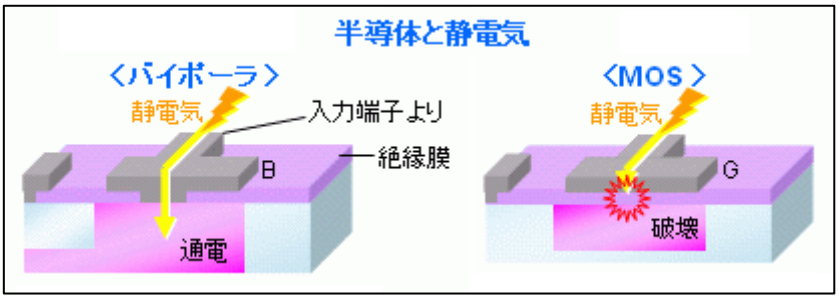
A3. トランジスタ数というのは、内蔵しているバイポーラ・トランジスタや MOS トランジスタ(FET)の総数です。これに対してゲート数というのは、MOS ゲート構造の数です（個々の FET のゲート端子数のことではない）。CMOS デバイスでは、基本的に 2 つの MOS トランジスタ（NMOS と PMOS）でゲートを構成するので、ゲート数はトランジスタ数の 1/2 です。ただし、オープン・ドレーン出力では、1 トランジスタ（NMOS だけ）でゲート構造になるので、その分だけ違いが生じることになります。
ゲート・アレイなどの「ユーザブル・ゲート数」というのは、ユーザ回路のロジック・ゲートとして使用できる、この CMOS 構造の数を意味します。



MOS 製品と静電気

Q1. なぜ MOS 製品は、静電気に弱いのですか。

A1. バイポーラ製品では、ベース入力が半導体を伝わって電流が流れるので、抵抗性で静電気を吸収します。ところが MOS 製品の場合は、ゲート入力が酸化膜で絶縁されていますので、電流が流れず、静電気の電圧がそのまま薄い酸化膜に印加されます。このため、酸化膜が破壊される可能性が高いのです。



半導体製品の種類

Q1. 半導体製品にはどのようなものがあって、どのように利用されるのですか。

A1. おもな半導体製品は、機能的には次のように分類されます。

当社では、通信デバイスや表示・画像処理デバイスなどの専用品を ASSP（Application Specific Standard Product：特定用途の標準製品）と呼んで、セミカスタム品の ASIC(Application Specific IC：特定用途 IC)と区別しています。

- ディスクリート（個別素子）
  - ダイオード
    - 整流ダイオード注
    - スイッチング・ダイオード
    - ツェナー・ダイオード
  - トランジスタ
    - バイポーラ・トランジスタ
    - FET
      - J-FET、MOS FET
  - サイリスタ
    - SCR、トライアック
- IC（集積回路）
  - マイコン
    - マイクロプロセッサ（MPU）
    - マイクロコントローラ（MCU）
  - メモリ
    - ROM
    - RAM
    - フラッシュ・メモリ注
  - 周辺デバイス
    - 通信デバイス
    - 表示・画像処理デバイス
    - A/Dコンバータ注、D/Aコンバータ注
  - オペアンプ、コンパレータ
  - 電源用IC
  - 標準ロジック
  - ASIC
    - ゲートアレイ
    - アナログマスタ
    - セルベースIC
    - ミックスト・シグナルASIC

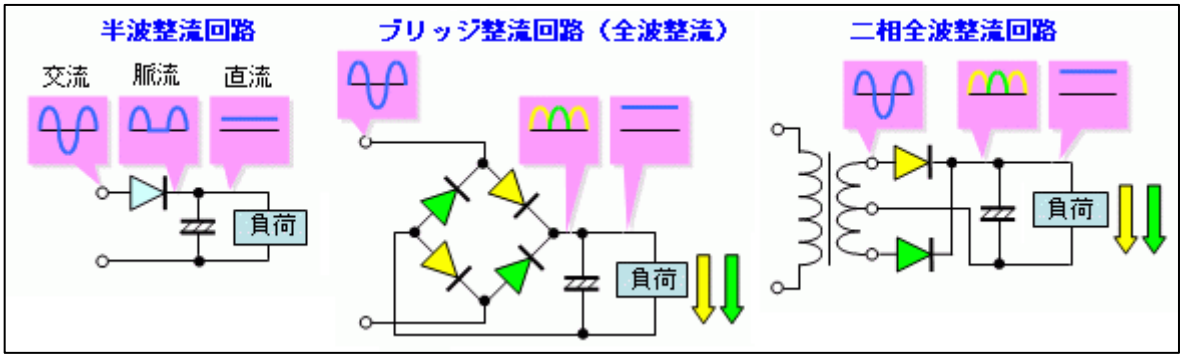
注 当社では、現在単体での生産をしていません。

(1) ダイオード

一方向にだけ電流を流す整流素子です。ただし、逆方向でも一定電圧（降伏電圧）を越えると、導通状態になって電流が流れます。ダイオードの動作については、半導体デバイスの動作原理をご参照ください。

(a) 整流ダイオード

整流機能を利用して、交流電源から直流を得るために使用されます。ここで、1 本のダイオードでは、マイナス側で出力しない半波整流のため、脈を打つような脈流を出力し、平滑コンデンサを設置しても、負荷に供給する直流電圧は低くなります。4 本のダイオードを使用したブリッジ整流回路では、対向の 2 本が組になって、交流のプラス側とマイナス側で、隣り合うダイオードが交互に順方向となって、負荷に供給する電圧が途切れない全波整流となり、平滑コンデンサの設置で高い直流電圧が得られます。また、トランスの中間タップをグランドとした二相全波整流回路でも、同様の直流電圧が得られます。



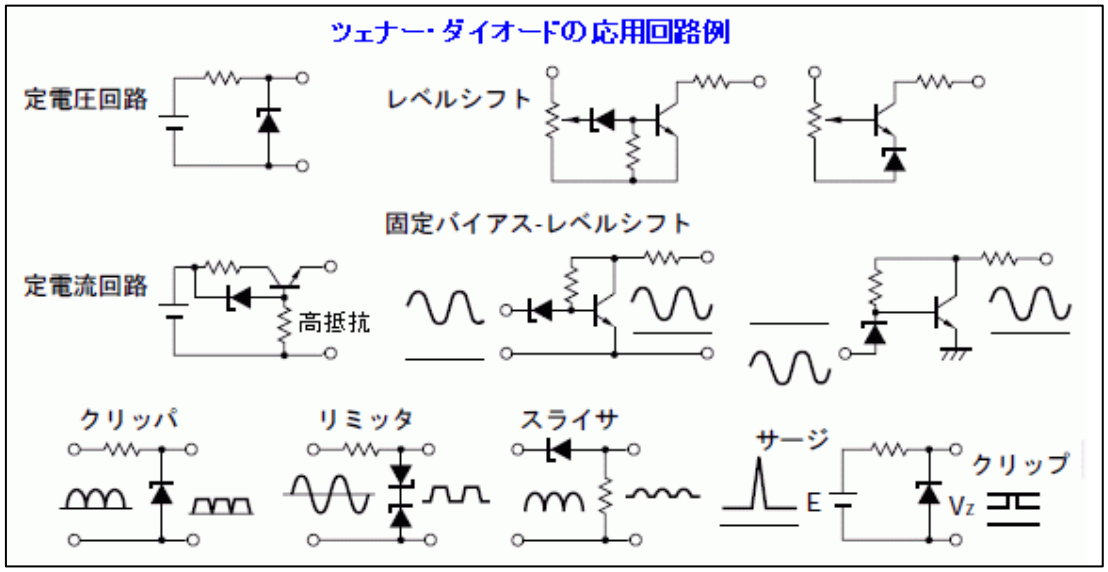
(b) スイッチング・ダイオード

電源のような大電流を流すのではなく、小信号の高速スイッチング素子として、整流機能が利用されます。

(c) ツェナー・ダイオード

逆極性での降伏電圧を越えると、導通状態になることを利用して、定電圧を得る回路や、過電圧やサージ、静電気による回路の破壊防止に使用されます。ノイズ除去を目的とする場合、その用途からノイズ・クリッピング・ダイオードとも呼ばれ、特にサージや静電気の吸収を意図する場合は、サージ・アブソーバと呼ばれることがあります。

なお、ノイズ･クリッピングをしない通常動作時には、端子間容量が信号変化の時定数に影響します。このため、高速信号線に使用する場合は、端子間容量の小さいものが必要となります。



(2) トランジスタ

入力信号の変化に応じて、増幅やスイッチング動作をする素子です。トランジスタの動作については、半導体デバイスの動作原理をご参照ください。

(a) バイポーラ・トランジスタ

おもに電流増幅に用いられます。エミッタ接地での電流増幅率(「出力／入力」の最大変化倍率)をh_{FE} で表し、製品ごとにh_{FE} ランクが規定されています。バイポーラ・トランジスタは、極性で PNP 型と NPN 型、特性で低周波用と高周波用に分類されます。

なお、バイポーラ・トランジスタには、小信号トランジスタと電力用のパワー・トランジスタがあります。またパワー・トランジスタには、ダーリントン接続で増幅率を大きくしたものもあります。

さらに、複数のバイポーラ・トランジスタを配列したトランジスタ・アレイもあります。

(b) FET

入力に電流があまり流れない電界効果トランジスタ(FET：Field Effective Transistor)です。

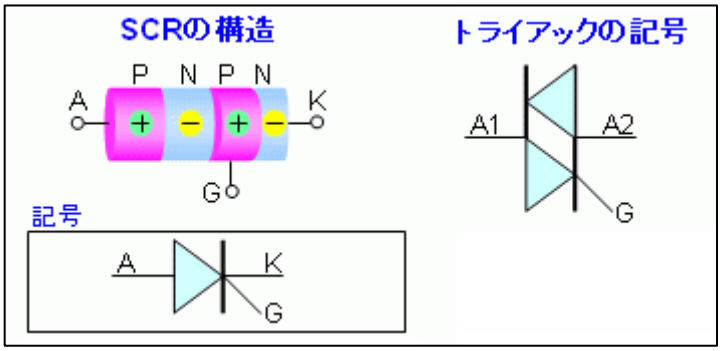
J-FET(Junction FET)は、ゲート-ソース間を逆バイアスにして、電流制御をすることにより、増幅やインピーダンス変換に利用されます。ゲートに電圧を印加しなくても、ドレイン-ソース間に常時電流が流れるデプレッション形です。ゲート電圧によって、ドレイン電流が制限されます。

MOS FET は、おもに回路のスイッチングに用いられます。また、極性で P チャネルと N チャネルに分類されます。ゲートに電圧を印加しないと、ドレイン-ソース間に電流が流れないエンハンスメント形です。

(3) サイリスタ

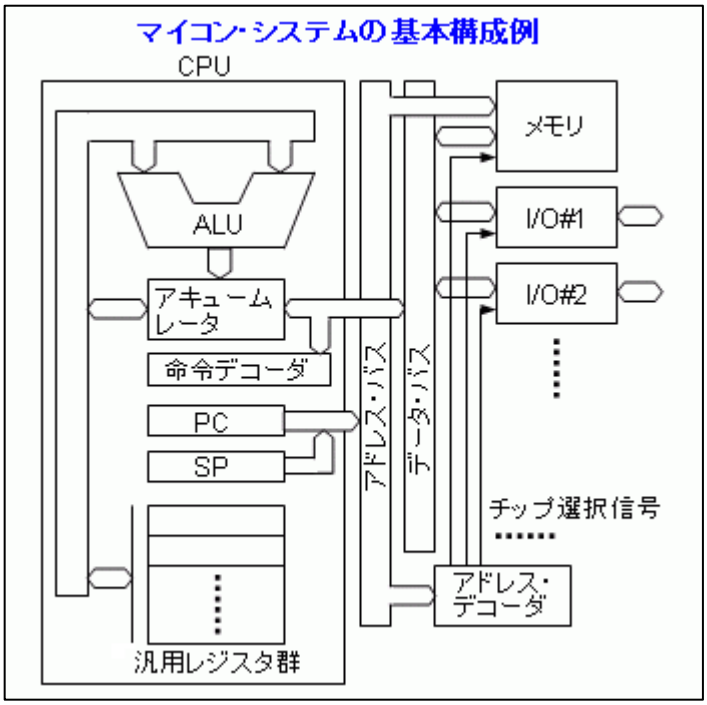
ダイオードは PN 接合で構成されますが、サイリスタはこれを 2 つ直列にした PNPN 構造になっています。

名称は、PNP 型と NPN 型を組み合わせてサイラトロン動作をするトランジスタ(Thyratron Transistor)に由来します。N 極にはさまれた P 極をゲートとして、ここに一定の電流を流すと、ダイオードとして機能するスイッチング素子です。単体のサイリスタを SCR (Silicon Controlled Rectifier)と呼びます。
これに対して、2 つの SCR を対向させて、交流制御に用いるものをトライアックと呼びます。

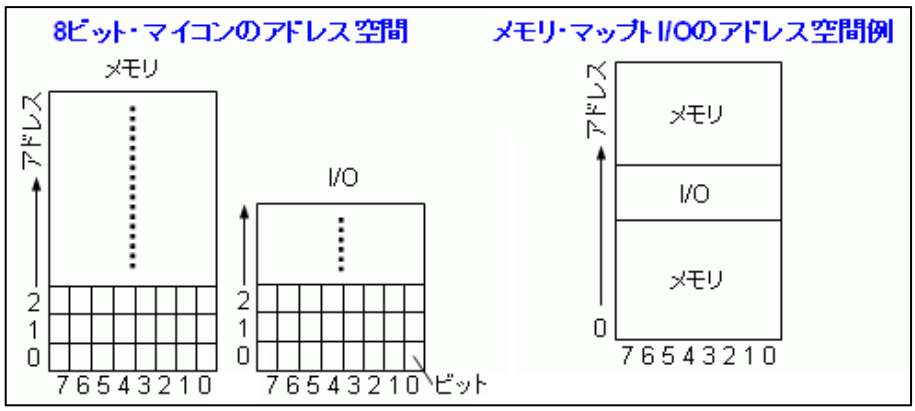


(4) マイコン

CPU(Central Processing Unit:中央処理装置)を内蔵するデバイスで、**プログラム処理**で動作が自由に定義できることが特徴です。マイコンの基本機能は、入出力／転送と演算です。演算には算術演算(加減算など)と論理演算(AND、OR など)があり、CPU にある ALU(Arithmetic Logic Unit)で実行します。入出力／転送の対象は周辺デバイスとメモリで、CPU 主体の **MPU**(Micro Processing Unit)に対して、最近はこれらを内蔵して組み込み用途などに利用される MCU(Micro Control Unit)が主流になっています。



CPU には**アーキテクチャ**という概念があって、データ・ビット幅や命令セット、アドレス空間、アドレシング・モード、周辺の資源割り当てなどが定義されています。現状のマイコンのデータ・ビット幅には、4、8、16、32、64 ビットがあります。データ・ビット幅を横方向とすると、アドレスは縦方向で、これらの積が管理できるデータ量になります。なお、メモリと I/O を別の空間として管理する方式と、I/O もメモリ空間と同様に割り当てる方式(メモリマップト I/O)があります。



周辺デバイスには、表示などの出力、キーなどの入力、ディスク装置などの入出力をする I/O デバイスと、タイマや割り込み制御、DMA 制御などをするサポート・デバイスがあります。

メモリは **ROM** と **RAM** で構成され、通常は ROM にプログラムが書き込まれますが、書き換えなどの目的で RAM にプログラムを置くこともあります。パソコンでは、起動プログラムが ROM にあるだけで、OS やアプリケーション・プログラムは、ハード・ディスクや CD-ROM などから読み出し、RAM に展開して実行するようになっています。また、プログラム領域には、内部メモリの場合と外部メモリの場合があります。内部 ROM については、**マスク ROM** から**フラッシュ・メモリ**に主流が移り変わっています

CPU は、まずメモリから命令コードを読み出し(オペコード・フェッチ)、命令デコーダで解析して、動作を決定します。このとき、PC(Program Counter)が、その命令コードのあるメモリの実行アドレスを指しており、プログラム実行ごとに更新して、次のアドレスを指定します。なお、最近のマイコンでは、命令実行ごとに次のオペコード・フェッチをするのではなく、プリフェッチ・キューに次々と命令コードを読み込んで、さらに実行もパイプラインで続けて行って、処理の効率化を図ったものがあります。

データの入出力や ALU での演算では、アキュームレータ(最近のマイコンでは表現されていません)がバッファとなって、一時的にデータを保持します。メモリとのデータ転送や I/O との入出力では、対象アドレスをアドレス・バスに出力して、これをアドレス・デコーダでデコードし、対象デバイスを選択して、データ・バスを通じてアキュームレータとのデータ授受をします。なお、最近の大量データを処理するマイコンでは、内蔵のキャッシュ・メモリにアクセス頻度の高いデータ群を置いて、アクセス効率の向上を図ったものがあります。

また演算の場合には、対象の 2 値をアキュームレータと汎用レジスタなどから ALU に入力して計算し、結果をアキュームレータに入力します。

なお、割り込みやサブルーチン・コールでは、現在のアドレス(PC の内容)を戻り先アドレスとして、メモリ(RAM)のスタック領域に退避させてから分岐しますが、その退避アドレスを SP(Stack Pointer)で管理していて、退避／復帰時にアドレス・バスを通じてその内容を RAM に出力します。

(5) メモリ

マイコン・システムなどで、データを記憶させるデバイスです。

(a) ROM(Read Only Memory)

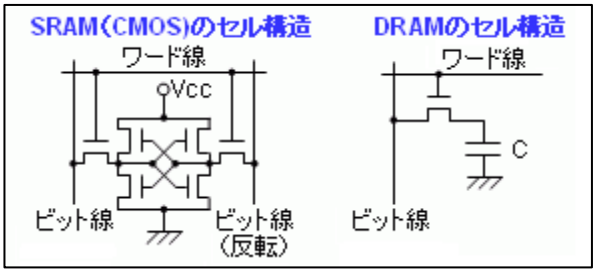
読み出し用のメモリで、電源を切っても書き込みデータは消えません(不揮発性)。固定的なプログラムやデータ・テーブルなどを格納するのに使用します。ROM には、製造工程でマスクによるデータ書き込みをするマスク ROM や、製造後にお客様が電気的に書き込みをする **PROM**(Programmable ROM)があります。また PROM は、書き換えができる **EPROM**(Erasable PROM)と書き換えができない(一度だけ書き込み)**OTP**(One Time PROM)に分けられます。EPROM については、かつては紫外線で消去してから書き込む **UVEPROM** でしたが、現在は電気的に消去したり書き換えができる **EEPROM** が一般的です。EPROM は、プログラムが未完成時の試作などで利用されます。

ROM
- マスクROM
- PROM
  - EPROM
    - UVEPROM
    - EEPROM
  - OTP

(b) RAM(Random Access Memory)

システム動作時に任意に書き込み／読み出しができるメモリで、電源を切ると書き込みデータは消失します（揮発性）。プログラムやデータの一時的な格納に使用します。データの書き換えは、上書きでできます。

RAM には、電源だけ供給しておけばデータを保持するフリップ・フロップ回路によるセル構造の SRAM (Static RAM)と、定期的にリフレッシュ信号を与えないと、電源を供給していてもデータが消失してしまうコンデンサ型セル構造の **DRAM**(Dynamic RAM)があります。SRAM は使いやすく高速性に優れています。DRAM は大容量化に向いていますが、リフレッシュ制御が必要です。SRAM のセルのアドレスは、A0～An のアドレス線で直線的に配置されています。DRAM のセルのアドレスは、行アドレス A0～Am と列アドレス A0～An でマトリクスに配置されていて、それぞれ RAS(Row Address Strobe)と CAS(Column Address Strobe)の信号でアクティブにします。RAS 信号がアクティブになると、選択されたワード線に接続されているメモリ･セルの電荷がビット線に伝わり、信号差がセンス・アンプで増幅されて、チャージされていたセルには再チャージされます。これが DRAM の**リフレッシュ動作**です。



(c) フラッシュ・メモリ

EEPROM の一種ですが、1 バイトごとの書き換えではなく、ブロック単位で消去・書き込みを行います。当初は書き込み用の電源端子(Vpp)がある 2 電源でしたが、最近は 1 電源でシステム上での書き換えもできるようになり、ROM や RAM の中間的な位置づけで、別に分類される傾向があります。

ルネサス エレクトロニクスの最近のマイコンは、All Flash 化によってフラッシュ・メモリの内蔵が一般的となっています。マイコンの内蔵メモリについては、マイコンの内部 ROM の種類をご参照ください。

(6) 周辺デバイス

マイコンの周辺で、マイコンが単独で処理できない信号の入出力やデータ処理をするデバイスです。

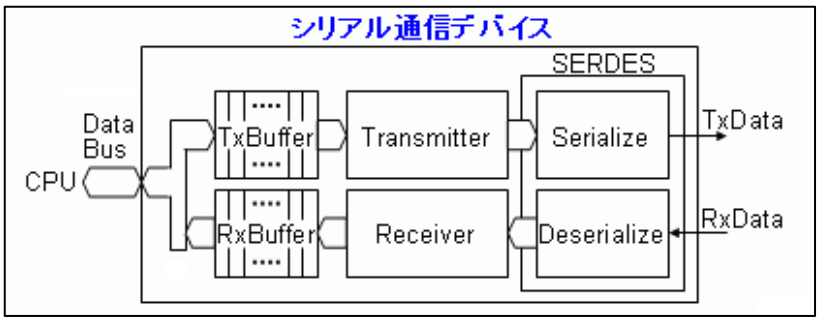
(a) 通信デバイス

システム間のデータ授受をするために、送信側と受信側の双方に出入り口としておかれます。そして双方の通信速度とプロトコル（通信の手順やフォーマット）を一致させることによって、相互の送受信が可能です。

通信は、ATM（Asynchronous Transfer Mode）のような基幹系と、USB のような機器間のインタフェースに大別されます。また、データの並びで分けると、データ・バスのイメージで送受信するパラレル通信と、送信側でシリアル・データに変換(Serialize)して、これを受信側でパラレル・データに再変換(Deserialize)するシリアル通信があります。パラレル通信ではビット数の分だけ回線が必要で、またシリアル通信では変換回路(SERDES：Serializer/Deserializer)が必要となります。高速インタフェースの例では、パラレルの PCI に対して、シリアルの PCI Express があります。

なお、シリアル通信では、ビット列に応じた通信時間が必要なので、最終ビットまで正確に送受信するために、送信側と受信側で同一クロックを基準にした同期式がよく用いられます。UART(Universal Asynchronous Receiver Transmitter)のようなビット同期式（データ列については非同期）も一般に使用されていますが、数百 kbps 以下で 2%以下の速度誤差の場合に限られます。

一般に通信デバイスでは、送信側の CPU が送信データを連続に書き込め、受信側の CPU 処理が遅くてもデータを連続に受信できるよう、双方に送信バッファと受信バッファとして、複数段の FIFO を内蔵します。



双方向の通信方式として、常時双方向で同時通信可能な全二重と、方向を切り替える半二重があります。

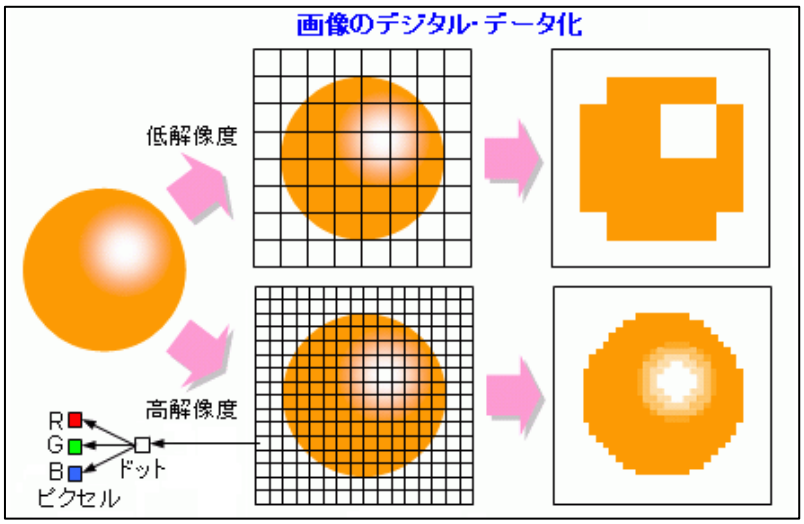
(b) 表示・画像処理デバイス

表示デバイスは、**OSD**(On-screen Display)**用 IC** での文字のような特定情報を出力します。OSD は、テレビの画面にリモコンの操作メニューを表示させたりする場合に使用されます。

また、画像処理デバイスは、画像データを出力したり、変形・変色などの加工や、**MPEG**(Moving Picture Experts Group)での**符号化／復号化**（CODEC：Coder/Decoder）をしたりします。ここで符号化／復号化は、伝送や保存のための情報量を少なくするための、圧縮・伸張のことです。プログラムや文字データなどは、1 ビットでも欠けるとまったく異なった結果になるので、可逆圧縮(Lossless Compression)が必要ですが、画像や音声のデータは少々のビット欠けがあっても、人間には認識できないので、大幅に圧縮する非可逆圧縮(Lossy Compression)が採用されます。ルネサス エレクトロニクスでは、MPEG コーデックなどを内蔵した **EMMA**(Enhanced Multimedia Architecture)シリーズなど、多彩な**デジタル AV 用 LSI** を用意しています。

なお、画像を入力するデバイスとして、**CCD センサ**（撮像素子）があります。CCD(Charge Coupled Device：電荷結合素子)は、フォト・ダイオードでの受光で電荷を蓄積させ、それを画像データとして転送するものです。

表示形式には、画像を格子状の画素（ドット）に分割した**ドット・マトリクス形式**と、デジタル時計のような固定形状を組み合わせた**セグメント形式**があります。ドット・マトリクス形式では、分割数を多くすると画素が小さくなって、形状の滑らかさや諧調度（グラデーション）の表現力が向上し、高精細な画像となります。ただし、データ量は膨大になるので、圧縮・伸張が重要になります。また、カラーの場合には、各ドットが光の三原色である RGB(Read, Green, Blue：赤緑青)に分解され、各データをピクセルとして処理されます。ピクセル・データを明るさで諧調表現（たとえば 8 ビットで 0～255）することによって、ドットの色調や濃淡を表現できます。

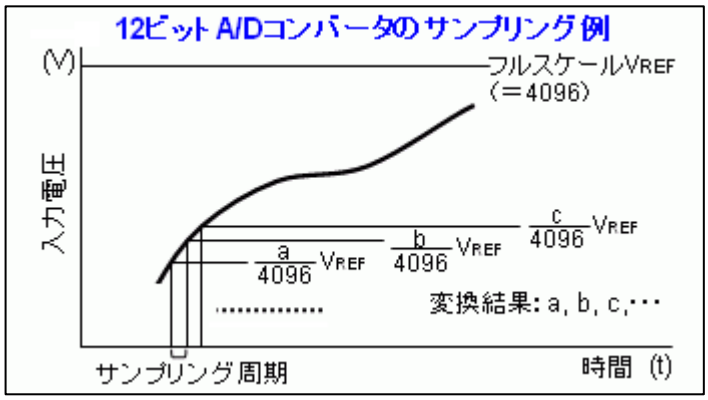


【ティー・ブレーク】
光の三原色である赤緑青を合わせると、白(無色)になります。ところが、絵の具の三原色は赤青黄で、合わせると黒になります。光の黒は、三原色がいずれもない状態です。
プリンタで紙に出力するにはインクを使用するので、色データの変換をして、シアン(C:水色)、マゼンタ(M:赤紫)、イエロー(Y:黄)と、深い黒を表現するためのブラック(K:黒)の 4 色で表現されます。
つまり、ディスプレイでは RGB で、また印刷では CMYK で、それぞれ色が表現されます。

(c) A/D コンバータ、D/A コンバータ

A/D コンバータ(ADC)は、アナログ信号をデジタル信号に変換します。これにより、各種センサなどの出力を A/D コンバータに入力すれば、アナログ・データをデジタルで処理／伝送したり保存したりできるようになります。逆に D/A コンバータ(DAC)は、デジタル信号をアナログ信号に変換します。これらを組み合わせることによって、たとえばデジタル放送や DVD などでの映像・音声の伝送／記録と再生が可能になります。最近はいずれも、マイコンや ASSP に内蔵される傾向にあります。

A/D コンバータでは、一定周期で入力電圧をサンプリングして、フルスケールのうちどのレベルかを順次数値化します。たとえば、12 ビットの A/D コンバータなら、基準電圧をフルスケール電圧として 4096 分割して、n/4096 から変換結果は n(000H-FFFH)となります。



D/A コンバータでは、デジタル・データをアナログ電圧に順次変換します。その出力間隔を A/D コンバータでのサンプリング周期に一致させれば、アナログ波形を再生できます。

(7) オペアンプ、コンパレータ

オペアンプ(Operational Amplifier:演算増幅器)は OP アンプとも表記され、アナログ入力と基準電圧を差動増幅する回路で、増幅やアナログ演算(加算・減算・微分・積分など)ができます。演算機能を利用して、A/D コンバータの入力段にも用いられます。詳細については、オペアンプの FAQ をご参照ください。

なおオペアンプには、電圧差動型と入力にエミッタ接地トランジスタを使用した電流差動型(ノートン・アンプ)があり、ルネサス エレクトロニクスは電圧差動型を製造しています。

コンパレータは、入力信号と基準電圧を比較する回路です。アナログ入力にもデジタル入力にも利用され、比較結果はデジタル出力となります。詳細については、コンパレータの FAQ をご参照ください。

(8) 電源用 IC

自動車用バッテリや家庭用の電灯線は、電圧が高すぎたり交流であったりして、そのまま電子回路の電源として使用できません。このため、電圧変換や整流((1)ダイオード参照)をする必要があります。このとき、リップルの除去や DC-DC 変換をするのが電源用 IC です。電子機器内部の電源や AC アダプタなどに応用されています。詳細については、電源用 IC の FAQ をご参照ください。

(9) 標準ロジック

インバータや AND、OR などのロジック IC で、論理回路の基本となります。現在、ユーザ・ロジックとしては、ゲートアレイなどで機能ブロックとして提供されています。

(10) ASIC

標準的な回路素子を下地に配列した IC です。お客様のシステム回路に応じた製造マスクで表面配線をして、チップを完成させます。ASIC によって、開発期間の短縮や機密保護、量産コストの低減などが実現できます。標準品とフルカスタム品の間に位置するため、セミカスタム品として分類されます。

(a) ゲートアレイ

MOS ゲートを格子状に配列したデジタル ASIC で、配線によって AND や OR などの標準ロジックとその組み合わせ回路を構成します。

(b) セルベース IC

CPU や通信デバイスなどの大規模マクロを IP コアとして用意した ASIC です。また、A/D コンバータや D/A コンバータなどのアナログ・マクロも用意されています。

(c) アナログマスタ

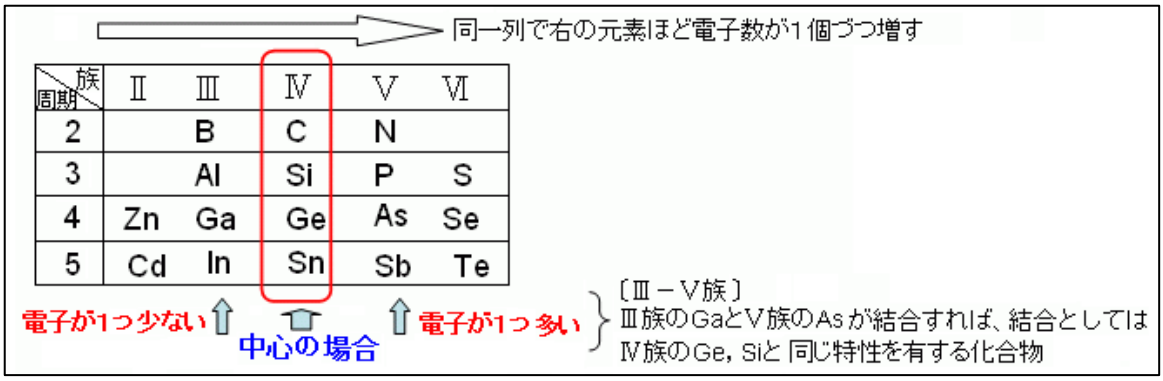
バイポーラ・トランジスタや抵抗、コンデンサを搭載したアナログ ASIC です。オペアンプやコンパレータも搭載しています。

(d) ミックスト・シグナル ASIC

ゲートアレイとアナログマスタの機能を搭載した、デジタル・アナログ混載の ASIC です。

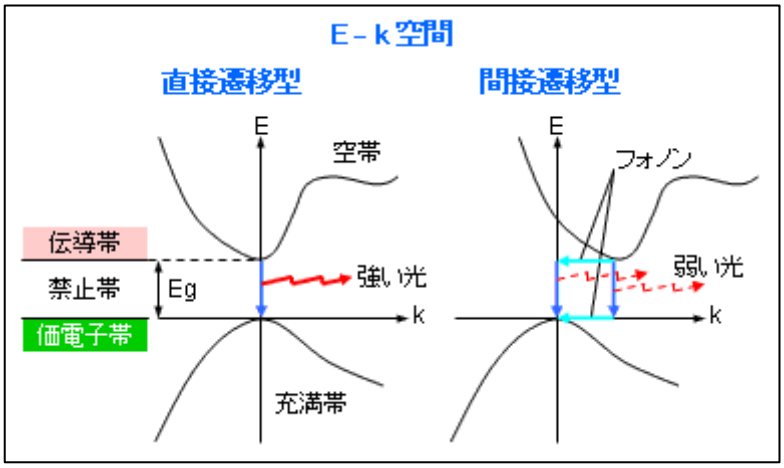
化合物半導体

半導体には、単体元素(シリコン Si など)の結晶と、2 種類以上の元素(ガリウム Ga とひ素 As など)を人工的に結合させた結晶があります。後者を化合物半導体と総称します。なお、ガリウム(Ⅲ族)とひ素(Ⅴ族)の化合物は、電子の移動度がシリコンの約 6 倍以上あるので、高周波(高速用)半導体デバイスの基板材料に用いられます。その他の化合物用元素(Al:アルミニウム、In:インジウム、N:窒素、P:リンなど)も発光ダイオード(LED:Light Emitting Diode)の基板材料として使用されます。このⅢ－Ⅴ族化合物のp形不純物として亜鉛(Zn)、n形不純物としてはテルル(Te)がよく用いられます。



発光ダイオードの発光は、ダイオードに順電流を流すと、正孔と電子が再結合して、それぞれが持っていたエネルギよりも小さなエネルギとなり、余分になったエネルギが光に変わることで起こります。このとき、単体元素では光が放出されにくく、化合物では放出されるので、化合物が発光ダイオードの基板材料に用いられます。

ここで、波数 k も考慮したバンド構造（E-k 空間）を考えると、ガリウムひ素 GaAs や窒化ガリウム GaN では、空帯（伝導体）の底と充満帯（価電子帯）の頂上が縦に並ぶ直接遷移型で、順バイアスによる電子と正孔の再結合で強い光を放出します。シリコン Si では、これらのピークが波数方向にずれていて、再結合エネルギは結晶の格子（原子）振動（フォノン）になる間接遷移型のため、熱や微弱光の発生にしかなりません。



発光の波長 λ は、次式で得られます。

λ[m]＝hc/Eg

h：プランク定数（6.626×10^{-34} [J・s]）

c：光速（3.000×10^{8} [m/s]）

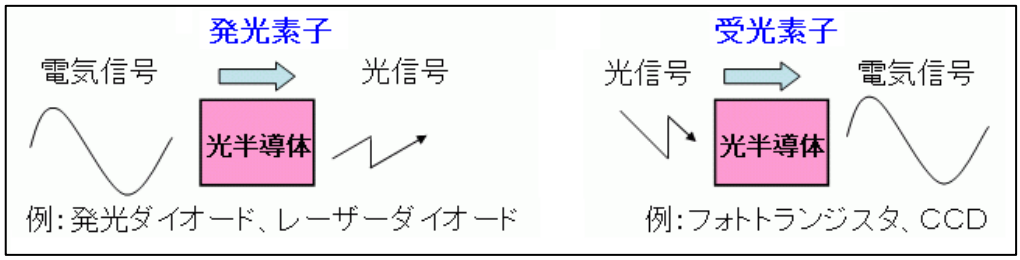
$1eV = 1.602 \times 10^{-19}$[J]より、λ は次のような値になります。

λ[nm]＝1241/Eg[eV]

たとえばガリウムひ素 GaAs の禁止帯幅 Eg は約 1.4eV で、波長は 886nm です。可視光は 360～830nm 程度なので、これは赤外線となります。また、窒化ガリウム GaN の禁止帯幅は約 3.4eV で、波長は 365nm で紫外線に近くなります。これに微量のインジウム In を加えて InGaN とし、青色発光ダイオードとして利用されています。

光半導体

身近な製品となったDVDプレーヤに使用されるレーザ・ダイオードは、電気信号を光信号に変換します。また、デジタル・カメラなどに使用されるCCD(Charge Coupled Device：電荷結合素子)は、逆に光信号を電気信号に変換します。このように、電気信号と光信号の間の変換をする半導体を光半導体といいます。

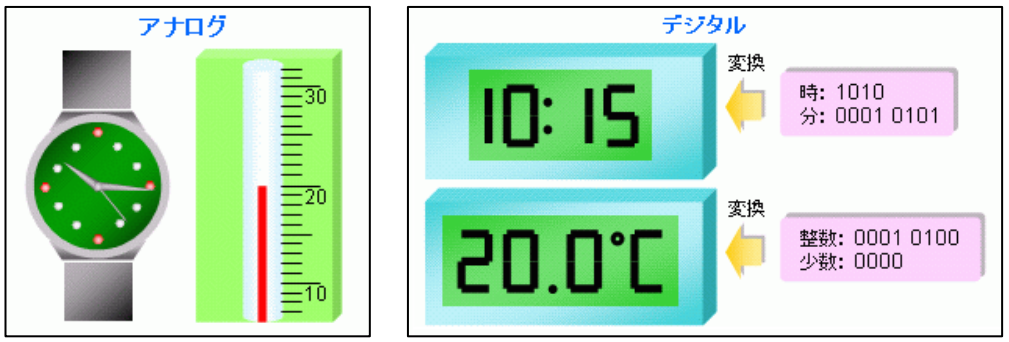


6.3 電気回路

アナログとデジタル

Q1. アナログとデジタルの違いはなんですか。

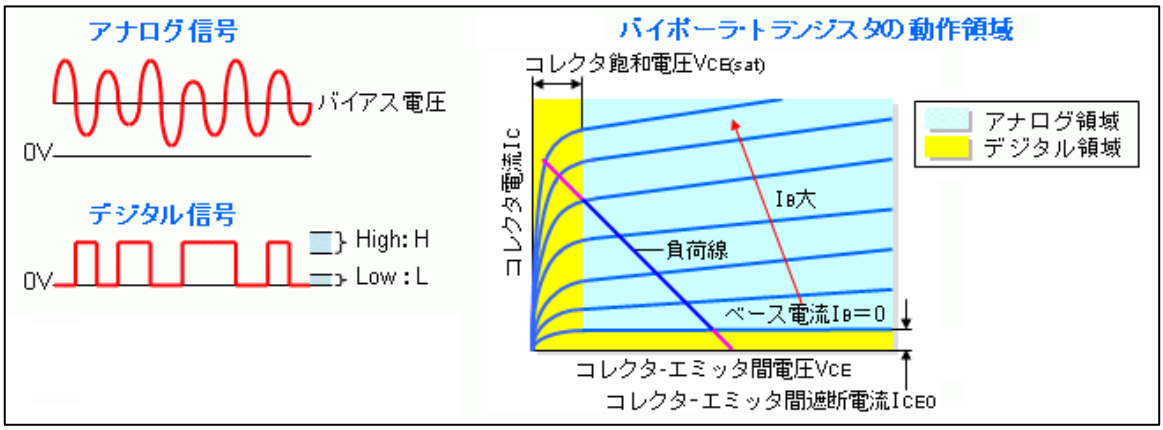
A1. アナログというのは、人が感覚的に直接とらえる量のことです。これに対してデジタルは、基本的に"1"か"0"の状態で変化する量で、数値変換することによって、人が視覚的にとらえることができます。



また、電気回路で扱う信号には、アナログ信号とデジタル信号があります。

アナログ信号は、連続的に電圧/電流が変化して、電圧/電流の値そのものに意味づけをします。

デジタル信号は、一定電圧以上を「ハイ・レベル」、一定電圧以下を「ロウ・レベル」と定義して、各レベルに対して"1"か"0"の不連続量で意味づけをします。たとえば、バイポーラ・トランジスタのベース電流I_Bを一定として、コレクタ電流I_Cの増加に伴って、コレクタ-エミッタ間電圧V_{CE}が直線的に増加する領域は、アナログ領域です。これに対して、コレクタ-エミッタ間電圧V_{CE}が非直線的に変化する領域では、負荷線が不連続のスイッチング特性を示し、デジタル領域となります。



なお、アナログ IC は線形(リニア)領域で使用するため、リニア IC と呼ばれます。

当社では、これらのプロセスによって、品名コードを分類しています。

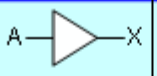
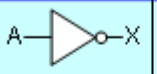
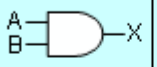
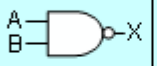
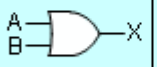
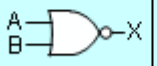
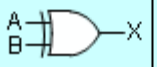
- μPB : バイポーラ・デジタル
- μPC : バイポーラ・アナログ(リニア)
- μPD : MOS(デジタル)

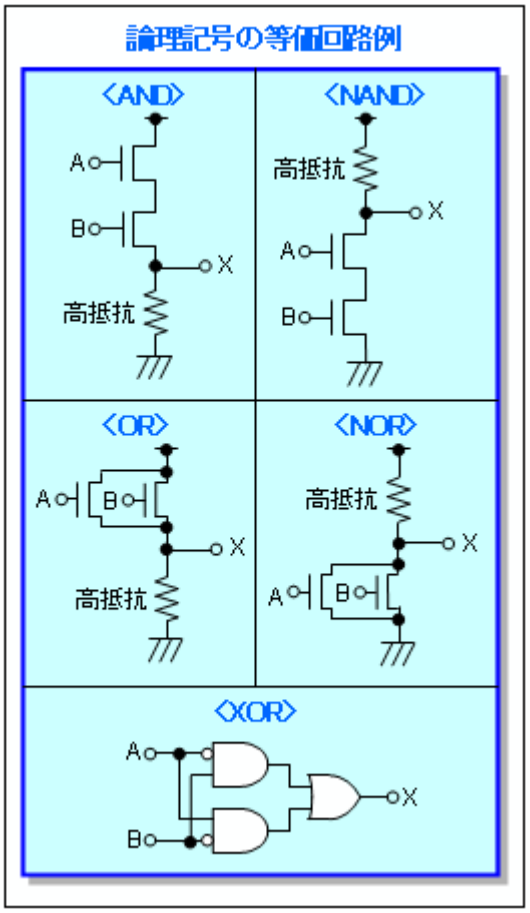
Q2. 論理記号の意味を教えてください。

A2. 論理記号の基本形は、1入力のドライバ、多入力の AND（論理積）、OR（論理和）です。また、これらを反転したインバータ（NOT）、NAND(Not AND：ナンド)、NOR(Not OR：ノア)があります。論理反転を負論理と呼び、論理回路の記号では、小さい「○」で表現します。なお、数式では論理反転を信号名の上線で表しますが、テキスト入力などで表現できない場合は、信号名の前か後ろに「/, !, #」などを付けて表します。

また、2 入力の排他論理和として、コンパレータなどで使用される XOR(Exclusive OR)があります。

なお、ドライバ（バッファ）は論理的には変化しませんが、回路的に電圧や電流、容量の駆動能力が不足する場合などに使用します。

名称	記号	論理式	真理値 A	B	X
ドライバ（バッファ）		X=A	0	–	0
			1	–	1
インバータ（NOT）		X=$\overline{A}$	0	–	1
			1	–	0
AND		X=A∧BまたはA∩B、A･B、A*B	0	0	0
			0	1	0
			1	0	0
			1	1	1
NAND		X=$\overline{A \wedge B}$または$\overline{A \cap B}$、$\overline{A･B}$、$\overline{A*B}$	0	0	1
			0	1	1
			1	0	1
			1	1	0
OR		X=A∨BまたはA∪B、A+B	0	0	0
			0	1	1
			1	0	1
			1	1	1
NOR		X=$\overline{A \vee B}$または$\overline{A \cup B}$、$\overline{A+B}$	0	0	1
			0	1	0
			1	0	0
			1	1	0
XOR		X=A⊻BまたはA⊕B	0	0	0
			0	1	1
			1	0	1
			1	1	0



★

リセット回路

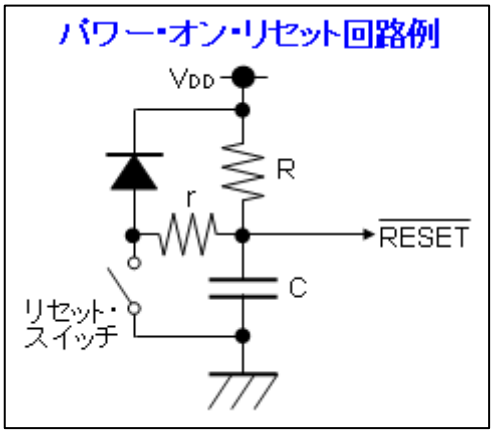
Q1. 電源投入時に IC へ入力するリセット信号は、どのように生成すればいいですか。

A1. リセット IC を利用されるのが、もっとも簡単です。

次に、RC の時定数でロウ・レベルを確保する方法があります。次図はリセット回路の例で、随時リセットをかけるためのリセット・スイッチも設置しています。なお、ダイオードは、電源遮断時に突入電流によって、接続している MOS IC を破壊することを避けるためのものです。一般に IC のリセット入力はシュミット・トリガなので、なだらかな立ち上がり波形でもリセットがかかります。

抵抗rは、大容量で低ESRのコンデンサを使用する場合に、ダイオードとコンデンサを保護する電流制限用に必要となります。スイッチON時のロウ・レベルを確保でき、リセット信号の入力バッファを保護できるような低抵抗で制限してください。

パワー・オン・リセット回路例



ここで、抵抗でのプルアップだけでコンデンサがない場合、電源電圧の上昇と同様にリセット入力も立ち上がるので、リセットがかかりません。コンデンサをバケツにたとえれば、水がバケツいっぱいになってからあふれるまでの時間を確保する働きをします。

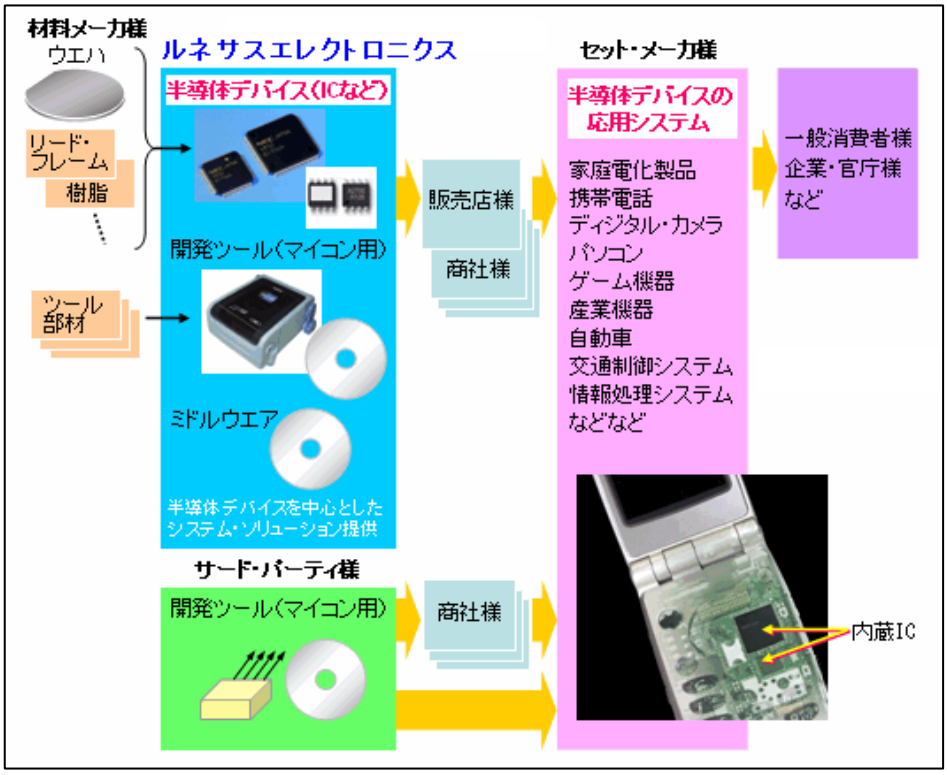
なお、最近のマイコンには POC（Power-on-Clear）回路を内蔵しているものがあり、その場合にはマイコン内部でリセット信号を生成するので、電源投入時のリセット入力は不要です。

7. 会社・事業

事業領域

Q1. ルネサス エレクトロニクス社の事業展開は、どのような位置づけになっていますか。

A1. おもな事業は、シリコンのウエハなどを中心とした材料を調達し、半導体デバイスを製造しています。
マイコンなど、お客様のシステム設計に開発ツールが必要な製品については、自社製のツールを製造するほか、サード・パーティ（協力会社）様にもツールをご提供いただいています。
なお、当社製品の販売については、販売店制度を導入しており、お客様には各販売店様を通じてご購入いただいています。価格や納期についてお問い合わせいただくことがありますが、たとえば電気製品を購入するときには、メーカではなく電気屋さんに確認されると思います。当社製品も同様で、価格や納期については販売店様が対応しています。特に価格は、価格公正法と独占禁止法に基づき、当社はお答えできません。



Q2. 会社案内に掲載されている「半導体ソリューション」というのは、どういう意味ですか。

A2. お客様がシステムを設計されるにあたって、より早くゴールに到達していただけるよう、用途に応じて機能や性能の最適化を図った半導体デバイス（IC やディスクリートなど）のことです。提供される機能は、制御、駆動、データ処理、信号処理などです。お客様は、システムに必要な半導体デバイスを組み合わせて、駆動機構や入出力部品などを接続したり、マイコンのプログラムを搭載したりすれば、そのシステムが完成します。
また、半導体デバイスだけでは実現困難な機能については、マイコンと組み合わせて使用する、用途ごとのミドルウエアやリアルタイム OS といったソフト製品も用意しています。
なお、マイコン製品などについては、お客様での開発を強力にサポートするため、自社だけでなく協力会社（パートナ）とともに、開発ツールや評価キット、各種ドライバなどの開発環境も提供しています。

製品別

1. マイコン …89
2. プラットフォーム製品 …711
3. 表示・画像処理 …714
4. 通　　信 …736
5. メモリ …753
6. リニア IC …755
7. ディスクリート …814
8. オプトエレクトロニクス …838

1. マイコン

1.1 共　通 …102
1.1.1 マイコンの種類 …102
マイコンの呼称 …102
マイコンのビット数 …104
マイコンの内部 ROM の種類 …104
マイコン内蔵のメモリ（データ記憶用） …104
マイコンの内蔵メモリ …105
表示 RAM やバッファ RAM を普通の RAM として使用 …105
All Flash …106
小ピン・マイコン …106
1.1.2 制御用マイコンと組み込み用マイコン …107
制御用マイコンと組み込み用マイコン …107
制御用マイコンや組み込み用マイコンとしての 78K0/78K0S …108
1.1.3 マイコンに内蔵された機能 …111
マイコンに内蔵された機能の使い方 …111
1.1.4 電源立ち上げ時の動作クロック切り替え …113
電源立ち上げ時の動作クロック切り替え［78K0/Kx2、78K0R］…113
1.1.5 ビット操作の注意点 …116
ビット操作上の共通注意事項 …116
ポートの出力異常 …117
ポート兼用端子の誤動作［μPD75P0016］…117
1.1.6 レジスタ設定順序 …118
レジスタ設定順序の注意点 …118
タイマ制御レジスタの設定順序［V853］…118
シリアルのレジスタ設定順序［78K0］…118
1.1.7 リモコン …119
NEC フォーマットの赤外線リモコン・フォーマット …119
NEC フォーマット …123
NEC フォーマットの使用 …123
カスタム・コードの取得 …123
キャリア周波数の許容誤差 …124
受信用のデバイス …124
NEC フォーマット用の標準デバイス …125
赤外線リモコンを受信するサンプル・プログラム …125
1.1.8 最小割り込み間隔 …125
最小割り込み間隔 …125
1.1.9 入力信号の立ち上がり／立ち下がり …126
端子に入力する信号の立ち上がり／立ち下がり …126
入力信号の立ち上がりエッジによる割り込み時の立ち上がりの規格 …127
1.1.10 シリアル通信 …128
マイコンに内蔵されているシリアル機能 …128
シリアル・インタフェースの選択 …129
シリアル通信の同期／非同期の違い …131
ボー・レート誤差 …133
ハンドシェーク …134
シリアルのバッファ構成 …138
全二重通信／半二重通信 …140

調歩同期の基本動作 [ほとんどのマイコン共通] …141
3 線式シリアル通信の基本動作 [ほとんどのマイコン共通] …144
1.1.11 タイマ／カウンタ …145
タイマ／カウンタの基礎 …145
16 ビット・タイマ／カウンタ 00 の使い方(1) [78K0、小ピン・マイコン共通] …153
16 ビット・タイマ／カウンタ 00 の使い方(2) [78K0、小ピン・マイコン共通] …157
内蔵タイマ／カウンタの使い分け [小ピン・マイコン] …159
16 ビット・タイマの PPG 出力を使った PWM 出力(1) [78K0、小ピン共通] …160
16 ビット・タイマの PPG 出力を使った PWM 出力(2) [78K0、小ピン共通] …169
タイマを組み合わせたパルス間隔測定 [78K0、小ピン・マイコン共通] …177
タイマを使った複数パルス出力(1) [78K0、小ピン・マイコン共通] …179
タイマを使った複数パルス出力(2) [78K0、小ピン・マイコン共通] …182
8 ビット・タイマ・カウンタ H のパルス発生への応用 [小ピン・マイコン共通] …186
8 ビット・タイマ・カウンタ H のパルス発生への応用プログラム例 [小ピン・マイコン共通] …190
1.1.12 ウッチドッグ・タイマ …211
ウォッチドッグ・タイマの基本 …211
1.1.13 I^2C …214
I^2C バスの概要 …214
I^2C バスのプルアップ抵抗値 …219
I^2C バスの初期設定で、STT0 と SPT0 を同時にセットするとうまく動作しない …219
外部の EEPROM に I^2C バスで接続する場合の処理例(マスタ) …219
EEPROM 読み出し後のスタート・コンディション発行ができない [共通] …223
78K0S 用ソフトウエア I^2C バス通信プログラム [78K0S/Kx1+] …224
ソフトウエアでの I^2C バスのスレーブ機能の検討 [78K0S/Kx1+] …247
ソフトウエアでの I^2C バスのスレーブ機能の実現(1) [78K0S/Kx1+] …249
ソフトウエアでの I^2C バスのスレーブ機能の実現(2) [78K0S/Kx1+] …270
1.1.14 スタンバイ動作 …284
スタンバイ動作 …284
割り込みでの解除 …288
1.1.15 マイコンとメモリ …289
マイコンとメモリ …289
1.1.16 マイコンとポート …290
マイコンとポート …290
データ・リード …291
リセット時の状態 …292
1.1.17 マイコンでのキー・スイッチ入力 …293
マイコンでのキー・スイッチ入力(1) …293
マイコンでのキー・スイッチ入力(2) …303
マイコンでのキー・スイッチ入力(3) …305
1.1.18 キー割り込みとチャタリング対応 …305
キー割り込みとチャタリング対応 …305
1.1.19 A/D コンバータ …306
A/D コンバータの簡単な使い方 …306
1.1.20 LED のドライブ …306
LED のドライブ方法 [78K0/Kx2、78K0S/Kx1+] …306
1.1.21 LED のダイナミック表示 …312
LED のダイナミック表示方法 [78K0/Kx2、78K0S/Kx1+] …312
1.1.22 EEPROM の制御 …318
CSI による SPI メモリの制御 [78K0/Kx2 など] …318

CSI による SPI メモリと EEPROM の制御プログラム例［78K0/Kx2］…322
SPI メモリと EEPROM の制御プログラム例［78K0S/Kx1+］…348
CSI によるマイクロワイヤ(Microwire)メモリの制御［78K0/Kx2 など］…345
CSI によるマイクロワイヤ EEPROM の制御プログラム例［78K0/Kx2］…363
汎用ポートによるマイクロワイヤ EEPROM の制御プログラム例［78K0S/Kx1+］…375
1.1.23 テーブル参照と補間処理 …393
テーブル参照と補間処理［プログラム例以外は共通、プログラム例は 78K0］…393
1.1.24 マイコンの開発 …399
マイコンのシステム開発の流れ …399
命　　令 …400
ROM 化 …401
ROM コード発注時のフォーマット(インテル・ヘキサ形式) …402
All Flash マイコンでのプログラム開発(IMS レジスタ設定)[78K0/Kx2] …403
C 言語でのマイコン開発［78K0、78K0S 共通］…404
プログラミング …406
1.1.25 開発ツール部品 …408
変換ソケット EV-9200G-80 の重量と寸法 …408
変換ソケット EV-9200G-80 の突起 …408
EV-9200G-80 のリフロ実装 …408
1.1.26 フラッシュ書き込み …408
フラッシュ書き込みの基本的事項 …408
エミュレータとフラッシュ品で動作が異なる …411
フラッシュ・メモリ・ライタとの接続 …411
書き込み中のポート状態 …411
フラッシュへの書き込み時のポート設定 …412
データ転送モード(3 線シリアルや UART)の選択 …412
オンボード書き込み時のクロック …412
書き込み時間とシリアル・クロック速度の関係 …412
UART のシリアル・クロックが 4.19MHz では、データ転送ができない …412
フラッシュ書き込み時のクロック結線(CLK→X1) …413
通常動作時の V_{PP} 端子接続 …413
オンボード書き込みでのイニシャライズ・エラー …413
書き込み時の“Target SIGNATURE Error” …413
フラッシュ ROM 品からマスク ROM 品への置き換え …414
内蔵フラッシュは 20 回以上書き換えできないか …414
書き込まれなかった領域のデータ値 …414
内蔵フラッシュ ROM に書かれた内容の読み出し …414
セルフ書き込みは可能か［μPD78F0058］…414
フラッシュ書き込み時間 …415
1.1.27 プリント基板 …416
基板設計上の注意点 …416
プリント基板パッド寸法 …416
1.1.28 パラメータ／デバイス・ファイルのダウンロード …417
パラメータ／デバイス・ファイルのダウンロード …417
1.1.29 オプション・バイトの設定 …417
オプション・バイト …417
オプション・バイトの設定［78K0S/Kx1+、78K0/Kx2］…418
1.1.30 スクリーニング …419
スクリーニング［フラッシュ・マイコンや OTP マイコン］…419

1.1.31 電　源 …419
A/D 機能を使用しない場合の V_{REF} 電源供給 …419
レギュレータ …419
低電圧検出回路 …420
マイコンの電源立ち上げ［マイコン共通］…420
動作電流と電源電圧の関係［マイコン共通］…422
1.1.32 発　振 …423
マッチング …423
発振誤差 …423
1.2 78K デバイス …424
1.2.1 A/D、D/A 変換 …424
8 ビット A/D と 10 ビット A/D の違い …424
μPD780021A(8 ビット A/D)の開発を μPD78F0034A(10 ビット A/D)で代行 …424
μPD78F9116 での 8 ビット A/D 変換［78K0/S］…425
10 ビット A/D の変換結果の格納 …425
10 ビット A/D 変換値の高速 6 ビット・シフト…426
μPD789445 の変換時間 …426
A/D 変換時間の違い［78K 共通］…426
A/D 変換は CH で指定した時間だけ待てば終了するか［78K 共通］…427
アナログ入力に指定しなかった端子をポートとして使用［78K 共通］…427
ADCS0=0 による直列抵抗ストリングのカットオフ …427
A/D 端子として使用する場合の PM レジスタの入出力設定 …427
A/D リファレンス電圧 AV_{REF}を 5V から 3.5V に変更したときの問題 …428
A/D 変換専用の電源端子(AV_{DD}と AV_{SS})を接続しない［78K 共通］…428
A/D 変換結果は 2 度読みしないと正常な値にならないか …428
A/D の完了用割り込みフラグ INTAD のセット／リセット条件 …428
A/D 入力端子のインピーダンス …429
A/D 端子に付けることができるコンデンサの容量 …429
変換処理時間［μPD780034A サブシリーズ］…429
リファレンス電圧の設定［μPD780034A サブシリーズ］…410
A/D 変換完了の割り込みが発生しない［78K 共通］…430
チャネルを切り替えると、変換結果が切り替え前チャネルに影響される［78K0、78K0S］…430
1.2.2 クロック …433
最小命令実行時間の切り替え …433
発振安定時間 …433
サブシステム・クロックの発振安定時間と確認 …433
サブシステム・クロックからメイン・システム・クロックへの切り替え …433
サブシステム・クロック動作中の HALT をタイマ割り込みで解除できない …434
STOP 解除後のクロック［78K0/Kx1］…434
クロック・モニタの動作［78K0/Kx1］…434
メイン・クロックが停止する …434
サブシステム・クロックの停止［78K0 全般］…435
クロック切り替え［μPD780958］…435
X1 発振回路で X1 端子電圧が 2.5V でいいか［78K0R/Kx3］…435
高速発振内蔵クロックでの動作［78K0/Kx2］…435
1.2.3 シリアル通信 …436
設定に起因したシリアル通信ポートの動作異常 …436
スレーブ・モード／マスタ・モード …436
初期設定時に不要データが送信されてしまう …437

不要データを送信しないための初期化 …437
シリアルのクロックにタイマ・レジスタを使用 …438
エラー発生時の設定による INTSER0 への影響 …438
送信データ書き込みタイミングはボー・レートから換算した時間後でいいか …438
SIO31 を 2 線式で使用する場合の設定 …438
シリアル内部クロック使用時に ASCK2 は入出力ポートとして使用可能か …438
PM1 モード設定 …439
送信データ書き込みから送信開始までの時間のばらつきをなくす …439
公称ボー・レートと実ボー・レートの差 …439
ボー・レートの許容誤差［78K0］…439
3 線式シリアルでの通信の手順［共通］…440
シリアル受信動作を停止させないで動作モードの切り替えができるか …440
SIO3 からデータを読み出さないと通信は起動しないか …440
UART 入力端子がロウ・レベルのままだと受信完了割り込みが発生し続ける …440
同期式通信でクロック異常によりビットがずれた場合の処置 …441
送信中に送信許可停止した場合の復帰処理 …441
シリアル通信設定の注意事項 …441
SPI への対応［78K0］…441
3 線式シリアルの動作［共通］…441
自動送受信機能付きシリアル(CSIA0)の使い方［78K0/Kx］…442
UART での 300bps 対応［78K0］…443
マイクロワイヤへの対応［ほとんどの CSI をもつデバイス共通］…444
シリアル・アレイの使い方［78K0R］…444
UART6 での PC との接続［78K0、一部の小ピン・マイコン］…446
マイコン内蔵の UART での制御信号処理［共通］…447
ソフトウエアによる UART 機能［78K0、小ピン・マイコン共通］…455
ソフトウエアによる UART プログラム例(1)［小ピン・マイコン］…461
ソフトウエアによる UART プログラム例(2)［小ピン・マイコン］…470
ソフトウエアでの 3 線式シリアルのスレーブ機能検討［78K0S/Kx1+］…478
ソフトウエアでの半二重 3 線式シリアル・スレーブ機能の実現(1)［78K0S/Kx1+］…485
ソフトウエアでの半二重 3 線式シリアル・スレーブ機能の実現(2)［78K0S/Kx1+］…508
3 線式シリアルの最後の不要クロック［78K0/Kx］…529
1.2.4 タイマ／カウンタ …529
インバータ制御用タイマをインターバル・タイマ割り込みに使用［μPD78098x］…529
タイマ出力 TO70～TO75 を停止［μPD78098x］…530
割り込みを使用しないタイマ待ち［共通］…530
カウント・クロックの誤差(割り込み周期がばらつくか)［共通］…530
タイマの出力を別のタイマに入力して割り込みを発生させる制御［共通］…530
キャプチャ機能の制限事項［μPD789407A、789417A サブシリーズ］…531
カウンタ初期値が 1 になってしまう［共通］…531
PPG 出力の ON と OFF の時間が反転する［78K0］…531
PPG 出力が出ない［78K0］…531
16 ビット・タイマでのインターバル［78K0S］…532
16 ビット・タイマのオーバフローでの割り込み［ほとんどの 78K0、78K0S 共通］…532
16 ビット・タイマの読み出し［78K0］…532
8 ビット・タイマでの PWM 出力［78K0］…532
タイマの出力周波数が期待どおりにならない［共通］…533
1.2.5 ウォッチドッグ・タイマ …533
ウォッチドッグ・タイマのカウント・クロック［共通］…533
タイマ割り込み処理でウォッチドッグ・タイマをクリア［共通］…533
ウォッチドッグ・タイマのオーバフロー時にリセットした場合の復帰時間［78K0S、78K0］…534
プログラムでウォッチドッグ・タイマを動作停止／開始できるか［共通］…534

STOP モード解除時のウォッチドッグ・タイマの動作［共通］…534
ウォッチドッグ割り込みからの初期化［78K0］…534
ウォッチドッグ・タイマがオーバフローしたときのモード［共通］…535
ウォッチドッグ・タイマの使い方［78K0/Kx2］…535
1.2.6 割り込み …542
割り込みの動作の基本［共通］…542
割り込みの優先度制御の注意点［78K0S、78K0、78K0R］…545
割り込み応答時間の変動を少なくする［78K0、78K0S 共通］…547
同じ割り込み要求を多重に受けてしまう［78K0］…548
多重割り込み処理［78K0］…548
割り込み要求フラグ待ちをしていると割り込みがかからない［共通］…550
サブシステム・クロック動作時、HALT 中の割り込みのクロック［共通］…550
リセット直後に割り込み要求を受け付けるか［共通］…550
割り込み要求フラグ・レジスタ IF0L のフラグをクリアしないようにする［共通］…551
割り込み要求フラグのクリアにより他の割り込み要求を受け付けなくなる［共通］…551
1.2.7 PWM …551
タイマ 0 を用いた 14 ビット PWM 出力周期とパルス幅(基本事項) …551
PWM を使用しない場合の PWMn への設定値 …552
PWM の出力をロウ・レベルに固定(PWM=0%) …552
PPG 出力と PWM 出力の違い［共通］…553
1.2.8 CAN …553
CAN の設定 …553
μPD780833Y に搭載されている J1850(CLASS2) …553
1.2.9 I^2C …554
I^2C バスが起動できない［共通］…554
I^2C バスをうまく制御できない［共通］…554
I^2C のクロック周波数が設定と異なる［共通］…555
簡易 I^2C バス［78K0R］…555
簡易 I^2C 機能［78K0R］…555
I^2C バスへの不要な信号出力［78K0/Kx］…556
1.2.10 LCD(液晶表示) …557
LCD 駆動電圧の発生方式 …557
LCD 駆動回路で、V_{DD}=5V 時に LCD 電圧 3V は得られるか …560
LCD 分割抵抗の値 …560
LCDM0 レジスタにある VAON0 の意味［78K0S］…560
LCD 接続用外付けコンデンサの値と接続 …561
LCD 電源電圧を上げたときの非表示部分の点灯 …561
LCD の制御［78K0/Lx2］…561
1.2.11 メモリ割り付け、ディレクティブ、LOCATION …562
通常のメモリ以外に変数やコードを割り付け …562
リンク時に「F206：Segment xxxxxx can't allocate to memory-ignored」が出る …565
ROM 内蔵品を外部 ROM で使う …565
ワード・データはメモリ上にどう格納されるか …566
レジスタはどこにマッピングされているか［78K0S］…566
CALLT 領域をプログラム領域で使う …566
16 ビットのスタック・ポインタが指す RAM 領域［78K0R］…566
1.2.12 メモリ …567
メモリ外部拡張の基本事項 …567
外部メモリ・アクセスのタイミング …569

SRAM を増設 …569
内部高速 RAM 上でプログラム実行は可能か［78K0］…569
マイコンに内蔵されている EEPROM［μPD78986x］…570
マイコン内蔵メモリの特徴と使い方［78K0、78K0S］…571
1.2.13 メモリ・バンク …574
メモリ・バンク［78K0/Kx2］…574
1.2.14 ROM 化 …581
ROM の未使用領域の充填値 …581
ROM の未使用領域を特定値で埋める …581
フラッシュ品で開発してマスク化にする場合のデバイス・ファイル …581
ROM 化のとき、IMS(メモリ・サイズ切り替えレジスタ)設定は残しておいていいか …581
最終的に使用するマスク品のメモリ・サイズと異なる IMS 設定としたときの問題 …582
OC78K の-R オプション …582
1.2.15 ROM コレクション …582
ROM コレクション［78K0］…582
1.2.16 フラッシュ書き込み …587
フラッシュ版の FLSPM0 の設定は省略していいか …587
オンボード書き込みのファームウエア …587
誤動作による書き込み動作を終了する方法 …587
フラッシュ書き込み時の UART 端子のプルアップはどうなるか …587
V_{PP} の立ち上がり時間が遅い場合の問題 …587
通常動作時の V_{PP} 端子接続 …588
フラッシュ書き込みに適切な CPU クロック …588
フラッシュにオンボードで書き込むためのツール …588
フラッシュの書き換え回数 …588
フラッシュ・メモリ・プログラミング特性のV_{DD} 電源電流の条件［78K0/Kx2、78K0/Lx2］…589
1.2.17 ROM 書き込み …589
変換アダプタの互換性［PA-78P078GF、PA-78P208GF］…589
1.2.18 電　源 …589
A/D 回路の電源分離 …589
電源のバックアップに AV_{DD} は必要ないか …589
リセット時の電源電流は STOP 時の電流と同じか …590
電源変動の許容値 …590
電源オフ時にも外部からポートに電圧をかけたい …590
V_{DD0} と V_{DD1} に別電源を供給 …590
電源電流を抑えられるモード(省電力) …590
1.2.19 リセット …591
発振停止状態でリセット信号入力時のポートの状態 …591
強制的にリセット処理を実行 …591
電源がなだらかに ON すると暴走する …591
リセット入力は長くてもいいか …591
リセット時、ポートに接続した LED が点灯してしまう …592
リセットしたときの RAM データは保持されるか …592
STOP モードの解除にリセットは使用できるか …592
1.2.20 端子処理 …592
入力・出力混在ポートで内蔵プルアップを ON にしていいか …592
未使用の入力端子とグランドの処理 …593
未使用端子はまとめて 1 つの抵抗で V_{DD0} か V_{SS0} に接続していいか …593
オープン・ドレーン端子の未使用時は、なぜ V_{SS} 接続ではなく V_{DD} 接続か …593

入力保護用ダイオード …593
テスト端子を 10kΩ でプルダウンしていいか …593
1.2.21 発　振 …594
発振回路の抵抗値 …594
XT1 は GND に接続していいか …594
フィードバック抵抗を使用しないとどうなるか …594
1.2.22 ノイズ …594
XT1 をオープンにするとノイズの影響を受けやすいか …594
LSI はどのようなノイズ対策がされているか …594
電源電流の高調波がクロック周波数と違う …595
ワンタイム PROM 品とマスク ROM 品とでノイズ耐量は異なるか …595
μPD78F0034A と μPD78P018F の割り込み要求端子の耐ノイズ性の違い …595
1.2.23 製品の違い・代替品 …595
μPD780034A から μPD780078 へ移行時の注意点 …595
μPD78P018F の代替品 …596
μPD780021A と μPD78P014 の違い …596
μPD78058 と μPD780058 の違い …596
μPD78P054 と μPD78F0058 の違い …596
μPD78P064GC の代替品…597
μPD789114 サブシリーズと μPD789114A サブシリーズの違い …597
μPD78F9177 と μPD78F9177A の違い …598
フラッシュ品の品名［78K0/Kx1］…598
μPD78064、78064B サブシリーズと μPD780308 サブシリーズの違い …599
8 ビット・マイコンの製品の違い［78K0/78K0S］…599
78K0/Kx2 と 78K0S/Kx1+の違い［78K0/Kx2、78K0S/Kx1+］…601
1.2.24 定　格 …602
絶対最大定格と DC 特性の違い …602
電源電圧と動作周波数範囲 …602
V_{DD}＝1.8V で高速モードは使用できるか［78K0S］…603
1.2.25 パッケージ …603
ドライパック …603
μPD789316 のパッケージ …603
1.2.26 プログラミング …603
最初に実行する内容［共通］…603
78K0S で 10 進補正を行うプログラム例 …604
未使用割り込み発生時の安全対策 …606
レジスタ・バンク割り付けの考え方 …607
LCD 表示用 RAM をビット・セット／リセット …607
ポートのビット・クリア …607
10 進補正命令がない CPU 用の四則演算のサンプル・プログラム …608
スタック・ポインタに直接数値を代入する命令［78K0S］…608
設定系のレジスタを一定周期で書き換えてもいいか …608
16 ビット空間でのテーブル参照 …608
スタック・ポインタに設定した値を読み出すと異なる［78K0/Kx1、78K0S］…609
スタック・ポインタの設定［78K 共通］…609
1.2.27 禁止事項 …610
暴走時に未定義命令を実行 …610
レジスタ未定義の SFR 領域の読み込み …610
1.2.28 リセット兼用端子 …610
リセットと兼用端子のポートとしての使用方法［小ピン・マイコン］…610

1.2.29 初期設定 …611
マイコンの電源投入からの動き …611
初期化処理［78K0S/Kx1+］…613
1.2.30 マイコンでの 10 進演算 …625
マイコンでの 10 進演算［共通］…625
1.2.31 乗除算 …627
乗除算［78K0］…627
1.2.32 ブート・スワップ機能 …644
ブート・スワップ機能［78K0/Kx2］…644
1.3 V850 デバイス …646
1.3.1 A/D、D/A 変換 …646
A/D 変換値が正しくない …646
A/D 変換結果の格納 …646
変換結果のふらつき(隣接チャネルの値になる)［V850/SF1、V850/SA、V850/SB、V850/SC］…646
A/D の CE ビットのクリア［V853、V850E/MS、V850E/MA など］…647
A/D 変換を開始するまでの時間［V853］…647
1.3.2 クロック …647
サブクロックから通常クロックへの切り替え時間［V850］…647
50MHz のクロック動作は可能か［V850E/MA1］…648
セラロック(発振子)を使えるのは PLL モードだけか［V850E/MS1］…648
ROM レス・モード 0 では、リセット期間中でも CLKOUT 出力が出るか …648
外部クロックと動作クロックの関係［V853］…648
PLL モードで X1 に 5MHz を入力していいか［V853］…648
外部からの 6.78MHz クロックで動作するか［V853］…649
BUSCLK と CLKOUT の使い分け［V850/MA1］…649
クロックのジッタやデューティの許容値［V850E/MA1］…649
消費電力低減のためにクロックを動的に変更［V850E/MA1］…649
サブシステム・クロックへの切り替え［V850/S］…650
PLL モードでのクロックの設定［V850ES/SG2、V850ES/SJ2］…651
リセットとクロック発振の関係［V850E］…651
サブクロックでの STOP モード［共通］…651
1.3.3 シリアル通信 …651
16 ビット長で同期式のシリアル通信 …651
シリアル・クロックに TO2/TO3 を選択すると、TO2/TO3 をタイマ出力に設定すべきか …652
TO2/TO3 をクロックとした場合、TO2/TO3 の出力は通常のポートとして使用可能か …652
データ出力後の通信割り込みが発生しない …652
2CPU 間で UART 通信を行う場合の電源投入時の注意事項 …653
送信時開始時にあらかじめデータを 2 バイト書き込めるか［V850E/MA1］…653
シリアル通信の受信が始まらない …653
DMA 起動にソフトウエア・トリガ・ビットを使用［V850E/MA1］…654
通信設定変更時に動作クロックを停止しないでいいか …654
同期式シリアル通信で、TRMDn=1 のときはデータ受信できないか［V850E/MA1］…654
シリアル通信で PM22 を 1 にするタイミング［V850E/MS1］…654
UART 入力端子がロウ・レベルを持続したことによるエラーからの復帰 …655
UART 非同期シリアルから入力されるブレーク信号を検出 …655
UART の送受信完了処理を割り込み以外で行う …655
UART 送信割り込みのタイミング［V850ES/Kx1 など］…655
I²C バスでスタート・コンディションが発行できない［V850ES/Kx1］…656
3 線式シリアルでデータを送信したいが、クロックが出ない［V850ES/Kx1］…656

自動送受信機能付きシリアルの使い方［V850ES/Kx1］…657
CSIB1 のクロック・ソースとしての TOP01［V850ES/Fx2］…658
1.3.4 タイマ／カウンタ …658
途中で CRn0/1 を変更した場合の出力波形［V850/S］…658
複数のタイマを同期(同時にカウント・スタート)［V850E］…658
インターバル・タイマの設定値(期待値－1) …658
タイマをカスケード接続したときのコンペア値の設定方法［V850/SB1］…658
リアルタイム・パルス・ユニットのカウント・クロックの上限［V853］…659
リアルタイム・パルス・ユニットを PWM 出力として使用［V853］…659
タイマ制御レジスタの設定順序［V853］…659
16 ビット・タイマをインターバル・タイマとして使用［V850ES/Kx1］…660
タイマ／カウンタ機能［V850ES/Kx1］…660
タイマ 2 の動作［V850E/IA］…660
8 ビット・タイマでのパルス幅測定［V850/SB］…661
INTP0n0/1 割り込みで STOP モードが解除できない［V850/SV2］…661
1.3.5 ウォッチドッグ・タイマ …661
ウォッチドッグ・タイマのタイマ割り込み処理でのクリア …661
ウォッチドッグ・タイマのタイムアウト後の動作［共通］…661
STOP モードに入る前のウォッチドッグ・タイマのクリア(RUN=1)［共通］…662
ウォッチドッグ・タイマのプログラムでの動作停止・開始［共通］…662
ウォッチドッグ・タイマ割り込みでの初期化処理［V850/SA、V850/SB］…662
1.3.6 割り込み …663
割り込みの動作の基本［共通］…663
割り込みがかからない(1)［共通］…666
割り込みがかからない(2)［V850/S］…667
割り込みがかからない(3)［共通］…667
割り込み要求非サンプル命令［共通］…667
割り込みが一度しか発生しない［共通］…668
割り込みハンドラ・アドレス［共通］…668
割り込み時のエッジ判断［共通］…668
外部割り込み端子のレベルで割り込み発生［共通］…669
割り込み要求フラグ(P02IF1)がセットされない［V850E/MA1］…669
不正命令コードの例外処理［共通］…669
UART 送信時の最初の割り込み［共通］…670
多重割り込み［共通］…670
割り込み処理中の割り込み［共通］…670
バス・アクセスと割り込み［共通］…670
割り込み制御レジスタの操作と割り込み［共通］…671
割り込みからの初期化［共通］…671
ソフトによる割り込みトリガ［共通］…671
1.3.7 PPG 出力と PWM 出力の違い …672
PPG 出力と PWM 出力の違い［共通］…672
1.3.8 DMA …672
DMA の使い方(リアルタイム出力機能と組み合わせ)［V850/Sxx］…672
DMA の使い方(UART での最初のデータ送信)［V850/Sxx］…673
DMA の転送モード［V850E/Sxx］…675
DMA 起動にソフトウエア・トリガ・ビットを使用［V850E/MA1］…675
DMA を繰り返し起動［V850E］…675
DMA 再起動で必要な設定［V850E/MA1］…675
DMA を一時的に停止［V850E/MS1］…676
DMA でのミス・アライン［V850E/MA1、V850E/MA2、V850E/ME2］…676

DMA でのエンディアン［V850E/MA1、V850E/MA2、V850E/ME2］…676
フライバイ転送でのバス幅［V850E/MA、V850E/MS、V850E/ME］…676
フライバイ転送時のウエイト［V850E/MA、V850E/MS、V850E/ME］…677
I²C との DMA 転送［V850/SB］…677
1.3.9 CAN …677
メッセージ送受信の完了の確認［V850/SF1］…677
内部レジスタのアドレスが複数ある理由［V850/SF1］…677
各レジスタの設定［V850/SF1］…678
バス・オフの発生と検出［V850/SF1］…678
受信メッセージが抜ける［V850/SF1］…678
MASK を有効にする設定［V850ES/Fx3］…679
1.3.10 IEBus …679
マスタ送信要求［V850/SB2］…679
スレーブ受信時、受信を行わず、送信相手に再送信を行わせる［V850/SB2］…679
オーバラン時のエラーの DR 読み込み［V850/SB2］…679
INTIE2 割り込み処理後、マスタ要求が通らないと ISR を消去［V850/SB2］…679
スレーブ受信時、受信データの送り先アドレスを知る方法［V850/SB2］…680
IEBus コントローラの「簡易版」…680
1.3.11 実行時間 …680
各命令の所要クロックの記述場所［V853、V850/S］…680
パイプラインと命令実行速度［V850E/IA1］…680
ビット操作命令の実行時間［V850/S、V850E］…681
内蔵周辺アクセスでの実行速度［共通］…681
最も高速処理が可能なプログラム・エリア［共通］…682
ほとんどの命令を 1 クロックで実行できるか［共通］…682
1.3.12 プログラミング …682
MOVEA 命令の符号拡張 …682
リセット・スタート後、最初に行うこと［共通］…683
1.3.13 メモリ …683
外部 8 ビット・データ・バスで、偶数番地も奇数番地も下位 8 ビットでアクセス …683
外部拡張メモリへのバイト・アクセスを行う場合の注意［V850/SA1］…684
内蔵 ROM を無効にして外部 ROM で使用［V850/SA1］…684
メモリ拡張で MM を設定する場合の、ポート 9 の入出力モード［V850/SA1］…684
SDRAM：アクセス・マルチプレクス幅の設定［V850E/MA1］…684
SDRAM：バンク信号に接続するアドレス信号［V850E/MA1］…685
SDRAM：直接制御できる SDRAM 容量の最大値［850E/MA1］…685
SDRAM：オート・プリチャージ時の AP 設定［V850E/MA1］…685
SDRAM：書き込みができない［V850E/MA1］…685
SDRAM：TWPRE、TWE の意味［V850E/MA1］…686
ROM レス・モード［V850E/MA1］…686
周辺 I/O レジスタのアドレス［V850E/MS1］…686
16 ビット・アクセス時の A0 出力［V850E/MS1］…686
DRAM：同ページ内アクセスではカラム・アドレスだけ出力［V850E/MS1］…686
DRAM：プログラマブル・ウエイト［V850E/MS1］…687
μPD703100 のアイドル状態［V850E/MS1］…687
プログラム・ウエイトの再設定［V850E/MS1］…687
外部メモリ拡張容量［V853］…687
8 ビット幅と 16 ビット幅の外部メモリの接続［V853］…687
外部メモリの接続［ほとんどの V850ES に共通］…688
1.3.14 ROM コレクション …689
ROM コレクション［V850/S］…689

1.3.15 フラッシュ書き込み …692
フラッシュ・メモリへの書き込み …692
フラッシュ・ライタを使用する際の CPU 動作周波数［V853］…692
内蔵フラッシュ・メモリに書きこむためのツール［V853］…692
1.3.16 セルフ・プログラミングによる書き換え［共通］…692
ライブラリ、ユーザーズ・マニュアルの入手方法［V850 All flash 共通］…692
セルフ・プログラミング実行時の割り込み処理記述方法［V850 All flash 共通］…693
セキュリティの設定変更［V850 All flash 共通］…693
FlashBlockIVerify 関数の機能［V850 All flash 共通］…694
セルフ・プログラミングのディバグ［V850 All flash 共通］…694
セルフ・プログラミング後のフラッシュ・ライタによるプログラム書き換え［V850 All flash 共通］…695
1.3.17 セルフ・プログラミングによる書き換え(Type 別）…696
エントリ RAM の配置領域［V850 All flash Type03］…696
エントリ RAM の設定方法［V850 All flash Type03］…696
セキュリティ・フラグの設定方法［V850 All flash Type03］…696
セキュリティの有効範囲［V850 All flash Type03］…697
FlashSetInfo 関数のエラー発生要因［V850 All flash Type03］…697
セキュリティ情報の読み出し［V850 All flash Type03］…697
ブート・スワップ・フラグの設定［V850 All flash Type03］…698
セルフ・プログラミング実行時のタイマ5の使用禁止［V850 All flash Type03］…698
EEPROM エミュレーションで書き込み中の電源瞬断［V850 All flash Type03］…698
フラッシュ書き込み環境での内蔵 ROM アクセス［V850 All flash Type04］…699
FLMD0 の設定タイミング［V850 All flash Type04, 04a］…699
FlashInit 関数の実行環境、実行タイミング［V850 All flash Type04, 04a ,02c］…699
フラッシュ関数のエラー発生要因［V850 All flash Type04, 04a ,02c］…699
FlashStatusCheck 関数のエラー発生後の再実行［V850 All flash Type04, 04a ,02c］…699
セルフ・プログラミング・ライブラリ・ファイルのリンク方法［V850 All flash Type04, 04a ,02c］…700
セルフ・プログラミング・ライブラリをリンクしたプロジェクトのビルド時のエラー発生要因
［V850 All flash Type04, 04a ,02c］…700
ブート領域に対するセキュリティ機能［V850 All flash Type04, 04a ,02c, 03］…700
1.3.18 電　源 …701
リセット中は低消費電力か …701
電源電流を抑えられるモード(省電力）…701
1.3.19 リセット …701
リセット解除のタイミング …701
ソフトウエアでリセット …702
リセット信号のパルス幅 …702
リセットしたときの RAM データは保持されるか …702
リセット解除前のクロック安定時間 …702
1.3.20 端子の DC 特性と処理 …703
プルアップ抵抗の上限値 …703
シュミット・トリガ入力のスレッシュホールド規格 …703
未使用端子の処理 …703
出力端子を 5V にプルアップしていいか［V850E/MA1］…703
P00～P07 は未使用時に V_{DD} に接続していいか［V850E/MA1］…704
空き端子とバス・ラインに使用するプルアップ抵抗値 …704
V_{DD} 電源なしのときに、ポートに外部からハイ・レベルをかけていいか …704
ポート／制御兼用端子の処理［V850E/MS1］…705

1.3.21 バス・インタフェース …706
16 ビット SRAM の接続 …706
マルチプレクス・バスでの制御 …707
1.3.22 その他周辺機能 …707
外部 I/O 接続の簡略化(アドレス・バス／データ・バス) [V850E/IA2] …707
I/O 接続のアドレス・デコード簡略化 [V850E/MS1] …707
初期値が不定のポートは、出力設定前に値を書き込めば不定状態を回避可能か …708
ポートをコントロール・モードに設定していても、ポート入力データを見られるか …708
WAIT 端子でウエイト制御をするとき、兼用ポートはアドレス出力可能か [V850/SB1] …708
システム・ウエイト・コントロール・レジスタ(VSWC) [V850E,V850ES] …708
1.3.23 製品の違い …709
A の有無の違い [V853] …709
1.3.24 定　格 …709
μPD703107 データ・シートの出力遅延時間 …709
CLKOUT 同期と CLKOUT 非同期の 2 つの規格 [共通] …709
1.3.25 その他 …710
内部周辺 I/O 領域の「イメージが見える」 …710

1.1 共　通

1.1.1 マイコンの種類

マイコンの呼称

Q1. マイコンの呼称がいろいろありますが、どうちがうのですか。

A1. マイコンというのは略称で、正式名はマイクロコンピュータです。

マイコンは、外部にメモリや周辺を接続することを前提としたマイクロプロセッサ(**MPU** または **CPU**)と、これらを内蔵した組み込み用途のマイクロコントローラ(**MCU**)に大別できます。

MPU は拡張性や性能等を重視しますが、**MCU** は用途別なので、ターゲットとするシステムに応じた仕様となっています。**MCU** は1チップでシステムを実現するため、シングルチップ・マイコン（シングルチップ・マイクロコンピュータ)とかワンチップ・マイコンなどとも呼ばれます。

【ティー・ブレーク】

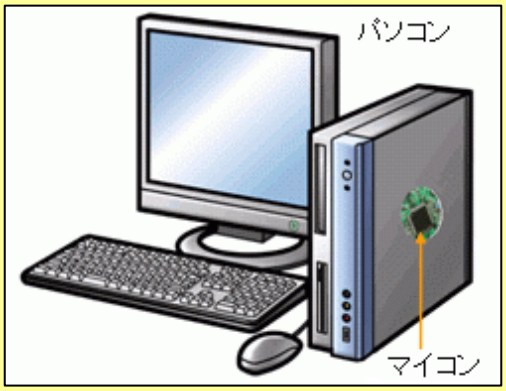
パソコンとマイコンを混同される方がいらっしゃいますが、パソコンは装置でマイコンは部品です。

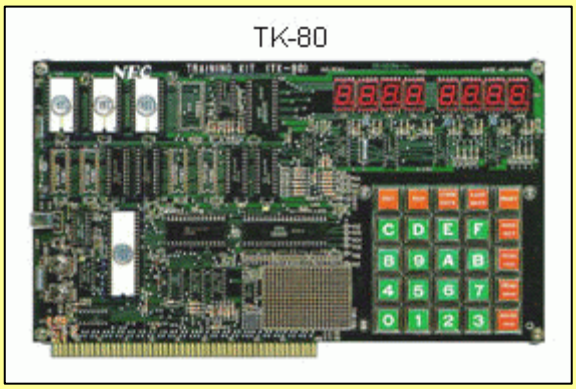
「マイコン」というのは日本での通称「マイクロコンピュータ」の略で、当初は正式には「マイクロプロセッサ」(MPU)と呼ばれていました。パソコンを例にあげると、Intel 社の Pentium(R)などがマイクロプロセッサです。その後、周辺を内蔵した組み込み用のマイクロコントローラ（MCU）も登場し、総称として「マイコン」が使用されています。なお、マイクロコントローラの出現当時は、プロセッサだけでなくコンピュータとしての機能をひと通り搭載していることを意図して、これを「マイクロコンピュータ」とか「マイコン」と呼ぶ風潮もありました。

「パソコン」というのは「パーソナル・コンピュータ」の略で、個人向けのコンピュータ・システムのことです。

NEC は 8 ビット・マイコン μPD8080 のトレーニング・キット TK-80（キー入力と 7 セグメント LED 表示）を 1976 年に発売し、それを小型汎用コンピュータとして発展させたパソコン PC-8001 が 1979 年に誕生しました。

TK-80 とその後継機 TK-85（μPD8085 搭載）は、マニアの間では、「マイ・コンピュータ」の意味で「マイコン」と呼ばれていましたので、パソコンに対しての誤解が生じているのかも知れません。





Q2. CISC と RISC はどうちがうのですか。

A2. まず、「シスク」「リスク」と読み、名称は次のとおりです。

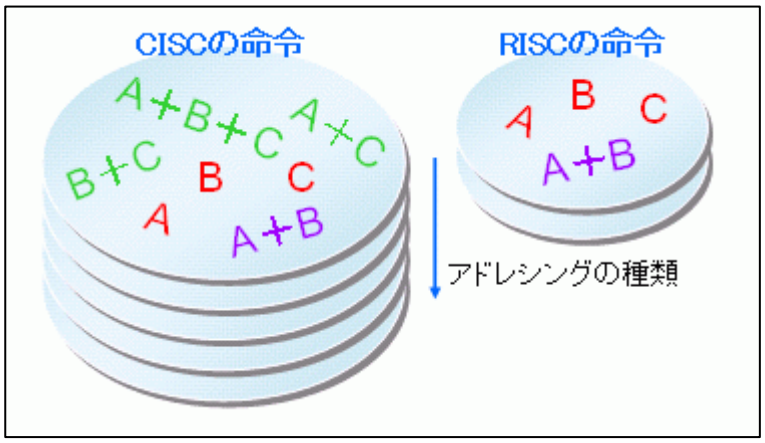
CISC : Complicated Instruction Set Computer（命令セットが複雑）

RISC : Reduced Instruction Set Computer（命令セットが簡素）

マイコンの進化に伴い、高速化とともに命令の複雑化が進みました。

たとえば A,B,C という基本機能の 3 命令があれば、1 命令で複数処理のできる A+B、B+C、A+C、A+B+C のように組み合わせた命令もあって、またアドレシングもたくさんあったほうが便利という発想です。アドレシングについては、オペランドにアクセス・アドレスを記述する直接アドレシングだけでなく、各種レジスタの内容をアクセス・アドレスとしたり、複数レジスタの内容を組み合わせてアクセス・アドレスとしたりする間接アドレシングなど、多様化しました。

こういった流れで発展したのが CISC です。でも、実際にすべての命令が多用されるわけではないのに、命令数が膨大になってしまいました。
命令数が増加すると、命令を実行する前に解析（なにをすべきか判断）するインストラクション・デコーダの負荷が大きくなってしまいます。ソフトでの解析では時間がかかりますし、ハードでの解析では回路規模が大きくなります。また回路の大規模化は、高速動作をさせるために、消費電力の増大も招く結果となりました。さらに、複雑なアドレシングについては、命令実行についての負荷も大きくなります。
これらの問題を見直して、実際に多用される基本命令だけに命令セットをしぼったマイコンが RISC です。
"Risk"ではないので、「危険なマイコン」というわけではありません。



Q3. レジスタ・オリエンテッド・マシンとメモリ・オリエンテッド・マシン（ポインタ・オリエンテッド・マシン）のちがいはなんですか。

A3. 比較的多くの汎用レジスタをもち、必要な演算をおもに A レジスタや B レジスタなどの汎用レジスタを用いて実行するのが前者です。汎用レジスタの数が少なく、メモリをインデクスするためのポインタ・レジスタをもち、おもにメモリとの間で演算を行うのが後者です。
前者では、演算に必要な変数が限られた数（8 つ程度）の汎用レジスタの範囲で処理できれば、コンパクトで高速処理が可能です。必要な変数が汎用レジスタに収まらないときには、メモリへのアクセスが必要になるので、処理効率が悪くなり、処理速度が低下します。
後者では、演算にメモリを使用するので、処理速度は汎用レジスタほど高速ではありませんが、変数用に多くのメモリが使用できるので、多くの変数が必要な場合でも速度の低下はありません。

【コーヒー・ブレーク】
昔のマイクロプロセッサで、8080 系は 8 つの汎用レジスタを搭載し、汎用レジスタで演算をしていました。6800 系はそんなに汎用レジスタを搭載しておらず、メモリを中心にした演算をしていました。
現在の 8 ビット MCU である 78K0 は 8 つの汎用レジスタで、しかも 4 つのレジスタ・バンクの合計で 32 の汎用レジスタを搭載しており、しかもショート・ダイレクト・アドレッシングで直接アドレスを指定できるメモリも使えるようになっています。
78K0S は 78K0 のサブセット版で、レジスタ・バンク機能はありませんが、8 つの汎用レジスタとショート・ダイレクト・アドレッシングで直接指定できるメモリも使えるようになっています。命令セットも 78K0S がサブセットになっていますが、命令コードの長さを見ると、汎用レジスタとショート・ダイレクト・アドレッシングで同じ命令長（HL レジスタをポインタとして使用すれば、もっと短い命令長）になっており、メモリ・オリエンテッド・マシンの性格が強くなっています。
どちらにしても、最近ではあまり気にする必要はありません。

マイコンのビット数

Q1. マイコンのビット数は、なにを意味するのですか。

A1. 一般に、マイコンの内部で、1度に処理できるデータの幅をビット数で表します。
たとえば 8 ビット・マイコンでは、1 度に 8 ビット演算しかできませんが、 32 ビット・マイコンなら 32 ビット演算ができます。つまり、32 ビット・マイコンの方が高速処理ができるのですが、それだけ大量のデータを1度に処理するシステムでなければむだな性能となるため、ビット数が大きければいいというわけでもありません。
なお、外部データ・バス幅が 16 ビットの 32 ビット・マイコンなど、入出力ビット数が内部データ・バス幅より少ない製品もあります。

マイコンの内部 ROM の種類

Q1. マイコンの内部 ROM の種類によって、用途が変わるのですか。

A1. MCU には、PROM やフラッシュ・メモリ、マスク ROM の内蔵品と、ROM なし(ROM レス)品があります。
PROM やフラッシュ・メモリには、ユーザがプログラミング・ツール(プログラマ)でプログラム書き込みを行い、試作や評価、少量生産などに利用されます。
PROM には、1 回しか書き込めない **OTP** と、消去・再書き込みのできる **EPROM** があります。
現在の主流は**フラッシュ・メモリ**です。
マスク ROM は、ユーザ・プログラムに応じた拡散マスク（LSI 製造に使用）を専用に準備し、製造工程でユーザ・プログラムを作りこんで、出荷時にはすでにプログラミングができているセミカスタム品です。
ROM なし品は、外部にプログラミングした ROM 等を接続する MCU で、 ROM だけの交換をする試作・評価や、複数 ROM の切り替えで機能切り替えをするシステムなどに利用されます。

マイコン内蔵のメモリ（データ記憶用）

Q1. マイコンには、データを記憶するメモリが内蔵されていますが、どのようなものがありますか。

A1. 通常の作業用 RAM と、電源が切れてもデータを保持できる EEPROM™ があります。

【ティー・ブレーク】
通常、マイコンに内蔵されたメモリは、次のように分類できます。



注 1. 以前は紫外線(UV)で消去できるように窓がついた EPROM や窓をなくして 1 回だけ書き込める PROM がありましたが、最近はフラッシュ・メモリになっています。
2. NMOS や PMOS のマイコンは、回路縮小のため内蔵の RAM やレジスタがダイナミック構造でしたが最近はスタティック構造になっています。このため、クロックが停止してもデータを保持できます。
3. 作業データ用としては、通常の汎用の RAM としての用途以外に表示用 RAM やバッファ RAM のような専用のものもあります。

Q2. EEPROM の使用上の注意点はありますか。

A2. EEPROM™ は、電源が切れてもデータを保持できますが、通常の RAM のように、簡単に書き込んだり読み出したりできません。専用の制御回路で書き込む必要があります。また読み出しについては、通常は電力消費を抑えるために、読み出せないようになっている場合があり、読み出す前に読み出し回路を動作させる必要があります。書き込みについては、「マイコン内蔵されている EEPROM について」もご参照ください。

マイコンの内蔵メモリ

Q1. フラッシュ・マイコンとは、どのようなものですか。

A1. フラッシュ・マイコンとは、マイコン内蔵のプログラム格納メモリとして、フラッシュ・メモリを内蔵したものです。

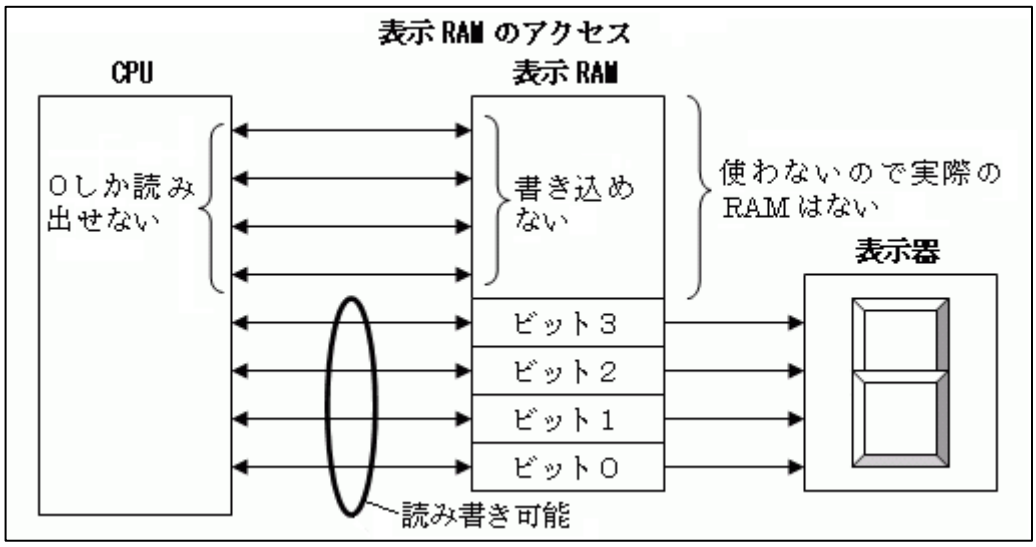
Q2. フラッシュ・マイコンとは、これまでのマイコンとどのような点が異なるのですか。

A2. もっとも大きな違いは、製品出荷後にプログラムを変更することができる点です。
フラッシュ・メモリでは、書き込んだ内容を電気的に消去して、新しい内容を書き込むことができます。
従来の 2 電源タイプのフラッシュ・メモリ(V_{DD} 以外に、書き込み用に V_{PP} 電源が必要)では、FlashPro などの専用のフラッシュ・ライタを用いることで、プログラムの書き換えが可能です。
単電源のフラッシュ・メモリ(書き込み用の別電源は不要)では、専用のフラッシュ・ライタを使用する以外にも、書き換えのためのプログラムと新しいデータを入力する手段を準備しておけば、書き換えが可能です。
また、出荷後のプログラムの書き換え以外にも、単電源タイプのフラッシュ・メモリを内蔵しているマイコンでは、フラッシュ・メモリの一部を EEPROM の代わりにデータ保存用に使用するような使い方もできます。
このように、使い方が大きく広がる可能性をもつのがフラッシュ・マイコンです。

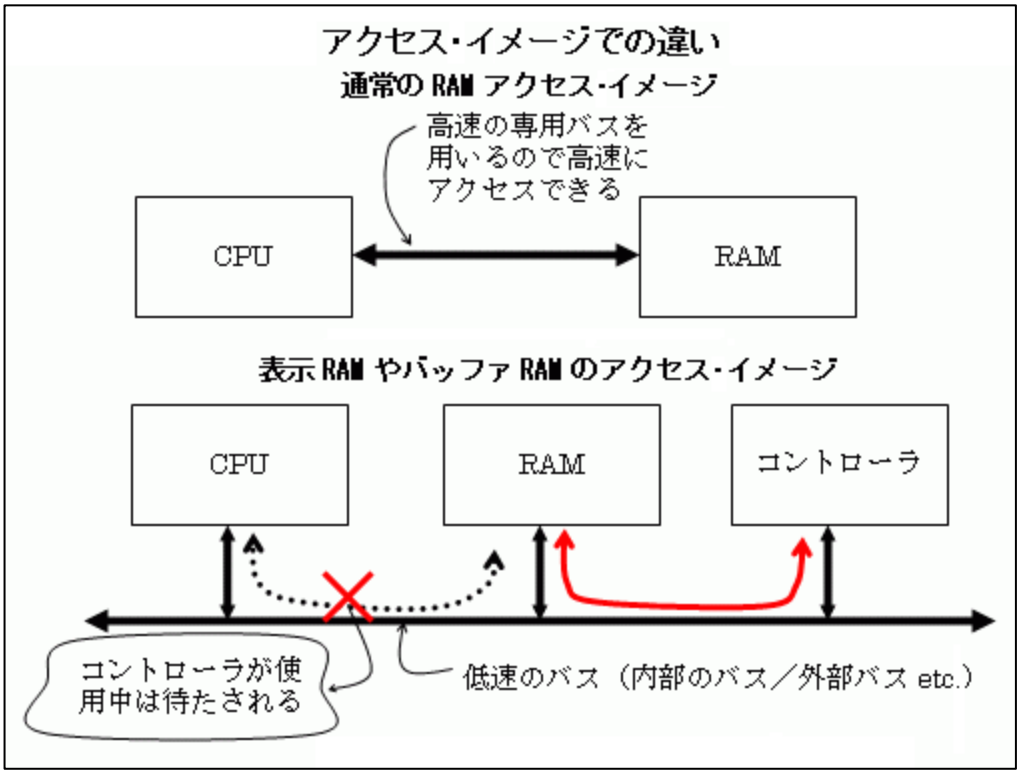
表示 RAM やバッファ RAM を普通の RAM として使用

Q1.　マイコン内蔵の表示用 RAM やバッファ RAM は、通常の RAM となにがちがうのですか。

A1. ビット構成に差があるのは表示用 RAM です。通常の RAM やバッファ RAM は、通常のデータを扱う(書き込んだり、読み出したりする)ため、データの幅が 8 ビットかその倍数が単位となっています。ところが、表示 RAM は、8 ビットでアクセスする場合でも、8 ビットすべてには RAM が存在しないことがあります。たとえば、78K0S のほとんどの LCD コントローラで、下 4 ビットには RAM が存在しますが、上位4ビットは 0 です。このような場合は、通常の RAM と同じように使うことはできません。



次に、RAM へのアクセス方法の差があります。通常の RAM は、CPU がアクセスするだけですが、表示 RAM の内容は表示コントローラが読んだり、バッファ RAM の内容はシリアル・コントローラが読み書きできます。このため、表示 RAM やバッファ RAM では、CPU とコントローラからのアクセスを調停する必要があります。そのため、表示 RAM やバッファ RAM に対して、CPU は通常の RAM のような高速アクセスはできません。また、一部のデバイスに内蔵されている CAN や IEBus コントローラでは、バッファ RAM が外部バスに接続されているものもあります。78K0/LG2 では、CPU が表示メモリを直接アクセスできず、I^2C バスを経由してアクセスする例もあります。



最後に、プログラム上の違いとして、CPU から見たアドレス領域が、通常の RAM 領域と異なっていて、標準状態では変数を配置できない点があります。通常、内蔵の高速 RAM では、特にメモリを宣言しなくても使えますが、表示 RAM やバッファ RAM では、リンク・ディレクティブでメモリの宣言をしないと、そこに変数を配置できない場合があります。リンク・ディレクティブについては、78K0 の「通常のメモリ以外に変数やコードを割り付け」をご参照ください。

All Flash

Q1. All Flash というのは、フラッシュ品しかない（マスク品や PROM 品がない）シリーズのマイコンのことですか。

A1. 基本的には、そのようなコンセプトです。ただし、マスク品や PROM 品がある従来品から派生のシリーズは、All Flash には分類していません。All Flash マイコンでは、マスク品相当以下の消費電力を実現しています。

小ピン・マイコン

Q1. 小ピン・マイコンというのは、どのような製品ですか。

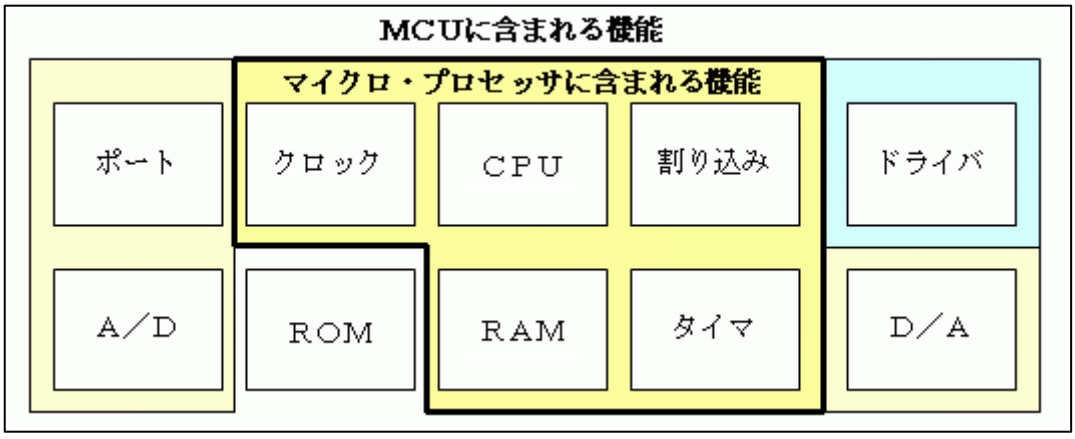
A1. All Flash マイコンの中で、約 30 ピン以下の 78K0S シリーズ（78K0S/Kx1+）をそのように呼んでいます。「小さい寸法のピン」ではなく、実際は"Low-Pin-Count"つまり「少ピン数」の意味です。

1.1.2 制御用マイコンと組み込み用マイコン

制御用マイコンと組み込み用マイコン

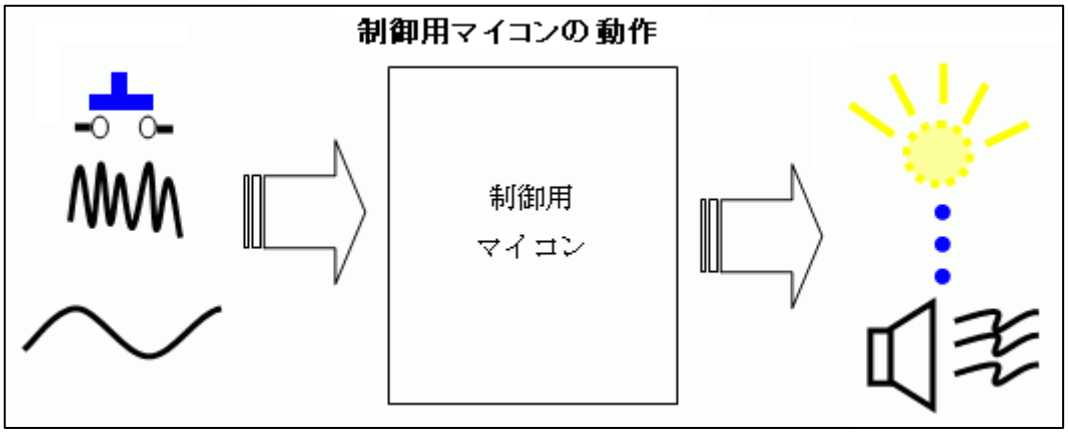
マイコンを区別するときに「制御用マイコン」や「組み込み用マイコン」といった呼び方をすることがあります。「普通のマイコン」とどこが違うのでしょうか。マイコンは自動車、PC、エアコン、ビデオ・レコーダ、携帯電話、TV、電気炊飯器など、何らかの機器の中に入って（組み込まれて）使用されます。マイコンを眺めているのはメーカの人か一部のマニアくらいでしょう。でも、これらの区別するのは、組み込みのための機能を強化していたり、組み込んで使い易いようになっていたりするからです。マイコン（マイクロコンピュータ）は、MPUやCPUと呼ばれるマイクロプロセッサと、MCU と呼ばれるマイクロコントローラに分けて考えることができます。

コンピュータは何らかのデータを入力して、データを加工（計算）して、結果を出力します。この中のデータ加工部分を主な対象にしたのがマイクロプロセッサで、そのためのCPU機能を中心に構成されています。
次図に、コンピュータを構成する機能の例を示します。最近のマイクロプロセッサは CPU 機能だけなく、中央の太線で囲まれた機能を集積しています。場合によっては、そのまわりのポート、A/D や D/A 等の機能を内蔵している場合もあります。しかし、マイクロプロセッサでは、外部に命令を格納した ROM を接続する必要があります。



マイクロコントローラ（MCU）の場合には、さらにROMを内蔵しており、ここに命令を書き込んでおくことで、1 つのLSIだけでも動作させることができます。このためによりコンパクトなシステムを構成できます。

これらの応用と「制御用」や「組み込み用」といった場合の違いに触れてみたいと思います。
「制御用途」では何らかの入力信号に対して必要な処理を行い、結果を出力します。入力信号は「データ処理用途」と異なり、スイッチの状態や、温度や磁気といった物理的な情報を変換するセンサからの信号（アナログ量）が主になります。これらの入力に応じて、たとえば、スイッチを押すと明かりを点灯するとか、温度がある値以上になったら警報音を鳴らすといったような処理を行うのが、制御用マイコンの用途です。



通常、スイッチの情報は 1 ビットです。つまり、制御用マイコンとは1ビットの情報を効率的に処理できる必要があります。また、自然界の物理量はアナログ量なので、センサの出力はアナログ信号です。これを扱うにはA/Dコンバータを内蔵する必要があります。制御用マイコンの出力は、明かりの ON/OFF を制御する信号のように1ビットのものがあります。また、明るさを制御したり、音を鳴らしたり、DCモータを制御したりするためのアナログ信号であったり、モータなどのアクチュエータをドライブするための信号や、LCD や蛍光表示管のような表示装置をドライブする信号だったりします。これらに対応した出力機能も、制御用マイコンに要求されます。

また、携帯用の装置に応用するには、コンパクトにシステム構成できる必要がありますし、それ以外に、低い電圧で動作できるとか、低消費電力であることも重要な要素となります。このように、細かな工夫をすることで、目的に合った機能を実現しているのが「制御用マイコン」「組み込み用マイコン」です。

制御用マイコンや組み込み用マイコンとしての 78K0/78K0S

[はじめに]

78K0 と 78K0S は、組み込み用途の 8 ビット MCU（マイコン）です。その特徴は使いやすさ、特にプログラムの作りやすさにあります。最近の All Flash 化や安価な開発環境の提供により、少量多品種への対応もより簡単になってきています。ここでは、その機能の特徴について、組み込み用途の観点で説明します。

[78K0 と 78K0S の差異]

78K0S の基本的なアーキテクチャは、78K0 とほぼ同じです。しかし、機能がいくつか削除され、78K0 のサブセットになっています。

そのため、レジスタ・バンク機能が削除されており、これに関連したレジスタ・バンクの設定命令を含めて、一部の命令が削除されています。これに関連して、78K0 ではメモリ・マップの高速 RAM 上に配置されていたレジスタが、メモリ・マップからはなくなり、高速 RAM はすべてメモリとして使用できるようになっています。

この点を除くと、以下の説明は 78K0 と 78K0S で共通となります。以下、特に断らない限り、78K0 の部分は 78K0S も含めてお考えください。

[78K0 での組み込み対応の基本的な考え方]

基本的な考え方としては、できるだけ内蔵ハードウエアで対応することで、外部の回路を少なくして、ソフトウエア作成の負担を軽くするとともに、わかりやすいソフトウエアが簡単に組めることを目指しています。

このため、78K0 は組み込み用途に必要なポート、割り込み、タイマ、シリアル・インタフェース、A/D コンバータなど、代表的な機能をハードウエア（内蔵周辺機能）として搭載し、それらのハードウエアを簡単に制御できるような命令をもっています。All Flash 化では、POC（パワーオン・クリア）、高速内蔵発振器を搭載し、ウォッチドッグ・タイマも機能強化されており、より低コストでシステムが構築できるようになってきています。

[78K0 のアーキテクチャ／命令の特徴]（一部 78K0S には含まれず）

78K0 での一般的な演算は、基本的にアキュームレータを介して実行するようになっています。また、データ転送も基本的にはアキュームレータを介する必要があります。こういった機能は、演算処理中心の MPU よりは劣ります。しかし、組み込み用途で周辺の制御に用いる内蔵周辺機能は、SFR 領域と呼ばれる領域に配置されており、この領域はアキュームレータを介することなく、直接アクセスできます。

また、ポートなどをビット単位で直接操作できます。これ以外にも、割り込み処理を効率化するためのレジスタ・バンクなどの機能を搭載しています。
これらの特長により、組み込み用途では、効率的なプログラミングが可能です。以下、その詳細を説明します。

[SFR 領域]

この領域は、内蔵周辺機能の配置専用です。この領域に対しては、直接周辺機能を指定して、値を設定することができます。たとえば、ポート 1 に 55H の値を出力するには、次のように直感的に理解可能な記述となります（（ ）内は C 言語記述の場合）。

```
MOV     P1,#55H
(P1 = 0x55;)
```

また、ポート 1 のビット 0 の出力を 1 から 0 にクリアしたい場合や、0 から 1 にセットしたい場合は、次のようにビット操作命令で簡単に記述可能です。

```
CLR1    P1.0            ; ポート 1 のビット 0 をクリア
(P1.0 = 0;)
SET1    P1.0            ; ポート 1 のビット 0 をセット
(P1.0 = 1;)
```

さらに、ポートからの入力をチェックしたい場合は、条件付分岐命令を用いて、次のような記述が可能です。

```
BF      P0.1,$$
(while(P0.1==0);)
```

この場合、ポート 0 のビット 1 が 0(False)なら、自分自身に分岐します。つまり、ポート 0 のビット 1 が 1 になるのを待つ処理となります。
このように、内蔵周辺機能を SFR 領域に配置すれば、アドレスやレジスタなどを意識することなく、外部の信号の制御が可能になります。

[saddr 領域]

78K0 で効率的にプログラムを作成できるように準備されたメモリ領域として、saddr 領域があります。これは、メモリのアドレスを指定するための専用のポインタを用いず直接アクセスできる 256 バイトの領域です。このうち、0FE20H～0FEDFH（78K0S では 0FEFFH）の RAM をレジスタとほぼ同じように使うことができます。
たとえば、次のように 16 ビットの変数 WORK0 と、8 ビットの変数 WORK1、WORK2 を定義します。

```
WORK00 DSEG     SADDRP          ; 16 ビット変数領域
WORK0: DS       2
WORK01 DSEG     SADDR           ;  8 ビット変数領域
WORK1: DS       1
WORK2: DS       1
```

これらの変数は、次のようにレジスタの代わりに使うことができます。

```
MOVW            WORK0,#1234H    ;  16 ビットの値の設定
XCH             AX,WORK0        ;  16 ビットデータの交換
MOV             WORK1,#33H      ;  8 ビットの値の設定
DEC             WORK1           ;  8 ビットの減算
SET1            WORK2.0         ;  ビットのセット
CLR1            WORK2.1         ;  ビットのクリア
BT              WORK2.3,$LOOP   ;  ビットのチェック
```

さらに、ループ・カウンタとして、次のような使い方もできます。

```
DBNZ            WORK1,$LOOP     ;
```

内蔵メモリの区分については、「マイコン内蔵メモリの特徴と使い方[78K0、78K0S]」もご参照ください。

[ビット操作]

78K0 が、組み込み用途に対応して内蔵している機能の 1 つに、ビット操作命令があります。これにより、組み込み用途で頻繁に使用される、1 ビットのデータを効率的に処理できます。

78K0 のデータ・アクセスは、基本的にリード・モディファイ・ライトとなります。この機能を使えば、内蔵周辺機能を特別なハードウエア構成にせず、ビット操作を行えます。

【コーヒー・ブレーク】

78K0 のデータ・アクセス・サイクルは、次の部分が必要に応じて組み合わせられます。

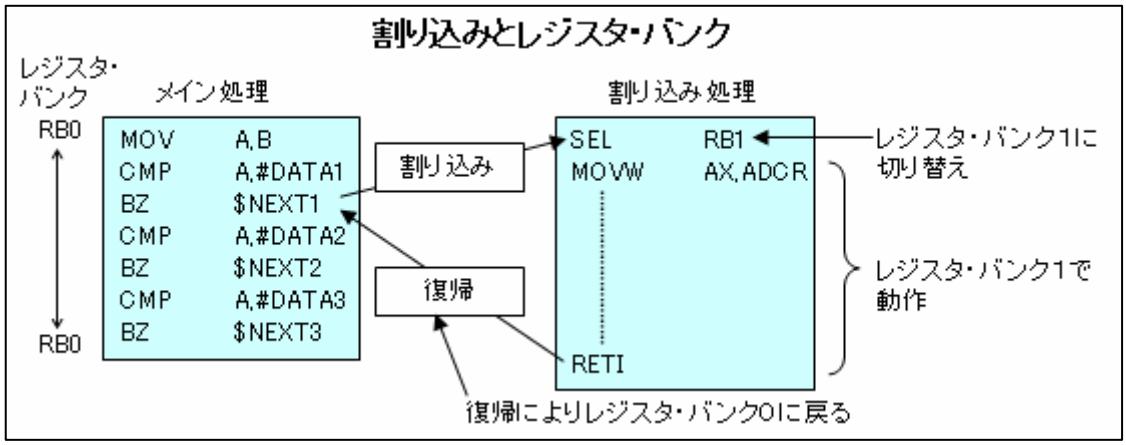
①アドレス指定
②データ読み出し
③データ書き込み

データ・リード：①と②の組み合わせとなり、①で指定したアドレスのデータを②で読み出します。
データ・ライト：①と③の組み合わせとなり、①で指定したアドレスに対して、③でデータを書き込みます。
ビット操作　：①～③のすべてを組み合わせます。①で対象のアドレスを指定し、②でそのアドレスのデータを読み出します。読み出したデータに演算を行い、③で同じアドレスに書き戻します。

[レジスタ・バンク]（78K0 だけ）

78K0 では、割り込み処理の効率化のために、汎用レジスタを 4 組内蔵しています。これを必要に応じて切り替えることで、割り込み処理に必要なレジスタの退避と復帰処理を簡素化できます。

たとえば、メイン処理ではレジスタ・バンク 0 を使用し、割り込み処理ではレジスタ・バンク 1 を使用するとします。この場合、割り込み処理の最初で、レジスタ・バンク 1 を選択します。すると、使用されるレジスタが、メイン処理で使用していたバンク 0 からバンク 1 に切り替わるので、レジスタ内容の破壊を気にせずプログラミングできます。割り込み処理が終了し、RETI 命令で割り込みを終了すると、PSW レジスタが元の状態に戻るので、メイン処理はレジスタ・バンク 0 で処理を継続します。



このように、レジスタ・バンクを使用すれば、割り込み処理を簡単にできます。

[スタンバイ]

組み込み用途では、その消費電力が重要なファクタとなることから、78K0 には HALT モードや STOP モードのような省電力機能が内蔵されています。スタンバイ機能については、FAQ の「スタンバイ動作」をご参照ください。

また、割り込みによるスタンバイの解除でも、CPU を割り込み禁止に設定すれば、ベクタ割り込み処理をしないで、スタンバイの解除だけすることができ、HALT モードと組み合わせれば、割り込み応答の高速化ができます。

1.1.3 マイコンに内蔵された機能

マイコンに内蔵された機能の使い方

[はじめに]

最近のマイコンには、各種の機能が内蔵されていて、どのように使いこなすか、迷うようなことがあります。特に、1 本の端子を複数の機能で兼用している場合には、各機能の間の関係や使い分ける方法に注意が必要です。

[機能設定]

1 本の端子に複数の機能があっても、それらをダイナミックに切り替えては使いません。通常は、ある機能に設定したまま、その機能で使用します。これは、端子に接続される相手との問題もあります。マイコンだけがかってに設定を変更するのは問題です。端子の機能を変更する場合には、十分に接続相手側の動きも考慮して、問題が発生しないようにしてください。

また、特定のときだけ端子の機能を使用することもあります。このときも、使用しないときにすぐ機能を停止すると、問題がある場合があります。たとえば、UART での送信では、ストップ・ビットの先頭や中央で完了のステータスが立ったり、割り込みが発生したりすることがあります。この状態ですぐ機能を停止し、直後に次の通信を開始しようとすると、ストップ・ビットが途中で切れることも考えられます。シリアル通信では、動作を許可していても、実際にデータを書き込まないと、通信そのものを実行しないので、一般には機能を停止する必要がないはずです。

[入力ポート]

端子の基本は、ポートとしての使い方です。ほとんどの場合、マイコンがリセットから起動すると、端子の状態は入力ポートに設定されています。このため、入力ポートとして使用する場合は、特に設定を変更する必要がありません。この状態で、ポート・レジスタの内容を読むことで、端子の状態を読み出すことができます。

なお、入力ポートに設定すると、プルアップ抵抗を使用できる場合があります。

[出力ポート]

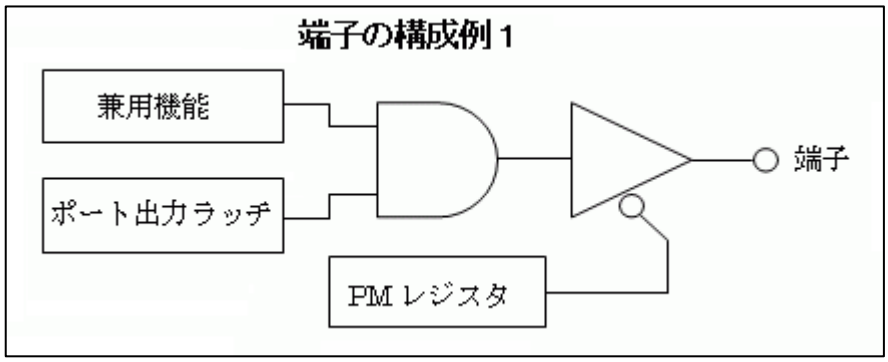
端子を出力ポートで使用する場合には、ポート・モードを入力から出力に変更します。この状態でポート・レジスタにデータを書き込むと、その内容が端子に出力されます。出力には、通常の CMOS 出力以外に、N-ch オープン・ドレーン出力などがあります。

リセット起動後にポート・モードを変更して、出力ポートに設定した場合は問題ないのですが、兼用機能を使ったあとでポートを使用すると、単純に兼用機能を停止するだけでは正常にデータを出力できないことがあります。

[兼用機能]

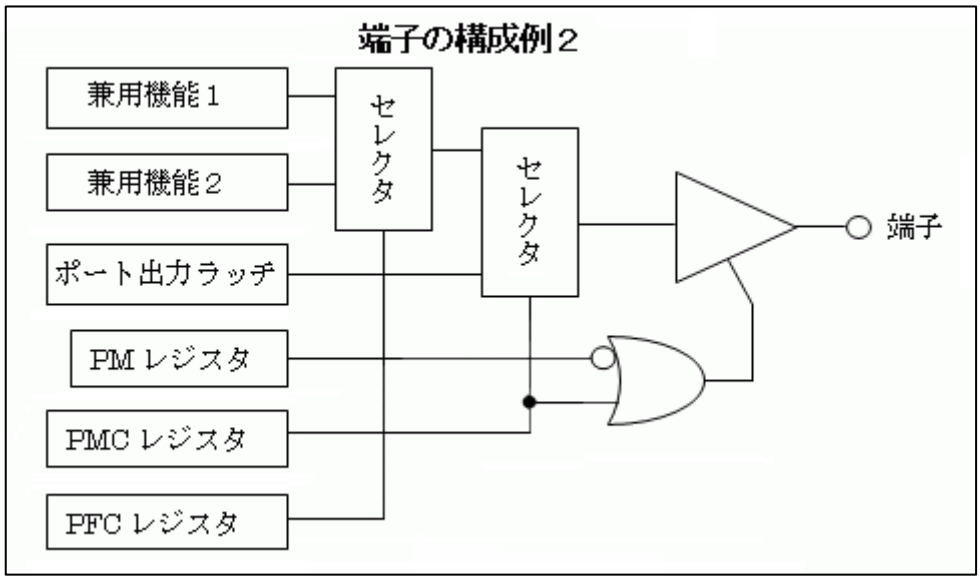
端子を兼用機能で使用する方法は、3 種類に大別されます。

まず、おもに 78K で使用されている方法で、ポートの設定を行う PM レジスタと、ポートの出力ラッチに設定するデータを用いる方法です。次に例を示します。この例では、端子の入出力設定をポートと同じ PM レジスタで行います。端子への出力データは、AND ゲートで結合されています。したがって、兼用機能を使用する場合は、ポートの出力ラッチに 1 を設定します。また、ポートを使用するとき（兼用機能を使用しないとき）は、兼用機能の出力は 1（ハイ・レベル）になっている必要があります。



もうひとつは、おもに V850 で使用されている方法で、端子の機能を切り替えるための専用レジスタを用いるものです。この場合、ポートか兼用機能かは、PMC レジスタで設定します。

さらに、端子に複数の兼用機能がある場合は、PFC レジスタや PFCE レジスタを用いて、使用する兼用機能を選択します。端子によっては、兼用機能のために専用のレジスタを持っていることもあります。次図に、兼用機能が 2 つの場合の例を示します。



[特殊な設定の端子]

以上で説明したような端子以外に、78K0 や 78K0S の All Flash の製品では、クロック発振回路の端子や RESET 端子をポートとして使用できるものがあります。これらの場合は、前記の制御方法とは異なる方法で設定を行います。

78K0/Kx2 の製品では、内蔵発振回路を使用する場合に、クロック発振回路の端子をポートとして使用できます。この場合、OSCCTL レジスタ（サブクロックは PCC レジスタも）で機能を指定します。リセット解除後、これらの端子は入力ポートとなります。したがって、クロックを外部から入力する場合や、クロックを発振させる場合には、端子機能の設定や必要な設定を行ってから、クロック入力を有効にしたり、発振を開始させる必要があります。

78K0S/Kx1+の製品では、オプション・バイトで指定します。オプション・バイトは、リセット解除直後（CPU が動作する前）に参照され、設定された内容が有効になります。

1.1.4 電源立ち上げ時の動作クロック切り替え

電源立ち上げ時の動作クロック切り替え [78K0/Kx2、78K0R]

[はじめに]

78K0/Kx2 や 78K0R シリーズのマイコンは、リセットからの起動時に高速内蔵発振回路のクロックで動作します。このため、水晶振動子やセラミック発振子を用いる場合は、発振回路を切り替える必要があります（このために、切り替え前に発振を開始させる）。マイコンは広い電源電圧範囲で動作します（「電源電圧と動作クロック」参照）が、使用する電源電圧により発振可能な周波数が異なります。以下、78K0/Kx2 を例に説明します。

電源電圧とX1発振周波数の例（78K0/Kx2）

電源電圧	発振周波数（MIN.）	発振周波数（MAX.）
$4.0V \leqq V_{DD} \leqq 5.5V$	1.0MHz	20.0MHz
$2.7V \leqq V_{DD} < 4.0V$	1.0MHz	10.0MHz
$1.8V \leqq V_{DD} < 2.7V$	1.0MHz	5.0MHz

このように、電源電圧で発振可能周波数が異なる場合には、発振周波数によっては処理に注意が必要です。最高周波数での発振を行い、POC（Power On Clear）を用いる場合には、特に注意が必要です。

[クロック発振開始前の手順]

そこで、電源が投入され、パワーオン・リセットや POC が解除されてマイコンが動作を開始したら、X1 の発振を開始する前に、低電圧検出回路を用いて、発振周波数に対応した電源電圧以上になっていることを確認します。低電圧検出回路の検出電圧としては、上記の表の低い方の電圧以上になっている必要があります。たとえば、X1 を 10MHz で発振させる場合には電源電圧が 2.7V 以上必要になります。2.7V 付近での、低電圧検出回路の検出レベルとしては、以下の2つが候補として考えられます。このうち、V_{LVI10} は低いほうにずれると 2.6V になるので、条件を満たさなくなります。そこで、確実に 2.7V 以上となる V_{LVI9} を選択します。

電圧選択	検出電圧
V_{LVI9}	2.85V±0.1V
V_{LVI10}	2.70V±0.1V

なお、発振させる周波数が 5MHz 以下であれば、電源電圧の立ち上がりを待つ処理は必要ありません。また、10MHz を越える場合は、検出電圧として V_{LVI0} を選択するだけで、処理は同じです。

(1) 具体的な処理

低電圧検出回路の割り込みをマスク（リセット解除時はマスク状態）し、LVIS レジスタに 9 を設定して検出電圧を設定します。次に、LVIM レジスタに 80H を設定して、割り込みモードで低電圧検出回路を起動します。起動後、10μs の動作安定時間を待ってから、LVIM レジスタを確認し、LVIF ビットが 0 になるのを待ちます。

注意 電源の立ち上がり待ち時間に、ウォッチドッグ・タイマがオーバフローして、リセットがかかることも考えられます。これを避けるため、ウォッチドッグ・タイマを使用する場合は、電源の立ち上がり待ちループの中に、ウォッチドッグ・タイマをクリアする処理（MOV WDTE,#0ACH）を追加してください。

(2) プログラム例

・アセンブラ記述の例

```
        SET1    LVIMK           ; 低電圧検出割り込みのマスク（念のため）
        MOV     LVIS,#9         ; 検出電圧(2.7V)を設定
        MOV     LVIM,#80H       ; 割り込みモードで低電圧検出を開始
        MOV     B,#6            ; ループ・カウンタを設定
LOOP1:
        DBNZ    B,$LOOP1        ; 10μs以上待つ
LOOP2:
        MOV     WDTE,#0ACH      ; ウォッチドッグ・タイマ使用時だけ入れる
        BT      LVIF,$LOOP2     ; 電源の立ち上がりを待つ
        CLR1    LVIIF           ; 割り込み要求フラグのクリア
```

・C言語記述の例

```
#pragma NOP
        :
        unsigned char work;
        :
        LVIMK = 1;              /* 低電圧検出割り込みのマスク */
        LVIS = 0b00001001;      /* 検出電圧（2.7V）を設定 */
        LVIM = 0x80;            /* 割り込みモードで低電圧検出を開始 */
        for (work = 0; work < 4; work1++){
                NOP();
        }                       /* 10μs以上待つ */
        while(LVIF){
                WDTE = 0x0AC;   /* ウォッチドッグ・タイマ使用時だけ */
        }                       /* 電源の立ち上がりを待つ */
        LVIIF = 0;              /* 割り込み要求フラグをクリア */
```

[クロック発振開始処理]

(1) 具体的な処理

電源電圧が必要なレベルになったら、X1 の発振を開始させるための処理を OSCCTL レジスタの設定で行います。まず、発振させるクロック周波数に応じた AMPH ビットの設定を行います。次に EXCLK と OSCSEL を設定します。これらの設定が終わったら、MSTOP ビットをクリアすることで、X1 の発振が開始します。

(2) プログラム例

・アセンブラ記述の例

```
        CLR1    AMPH            ; 10MHz を超える場合には SET1 を使用
        CLR1    EXCLK           ; 初期値は 0 なので必要はないが念のため
        SET1    OSCSEL          ; X1 発振モードを選択
        CLR1    MSTOP           ; X1 発振開始
```

・C 言語記述の例

```
        AMPH = 0;               /* 10MHz を超える場合には 1 にする */
        EXCLK = 0;              /* 初期値は 0 なので必要はないが念のため */
        OSCSEL = 1;             /* X1 発振モードを選択 */
        MSTOP = 0;              /* X1 発振開始          */
```

[X1 発振安定待ち]

X1発振開始を設定しても、すぐに発振ができるわけではありません。必ず、発振が安定するまで待つ必要があります。FAQ の「クロック発振安定時間」もご参照ください。

(1) 具体的な処理

発振安定までの待ち合わせは、OSTC レジスタで行います。X1 発振で使用する水晶振動子やセラミック発振子に応じて、OSTC レジスタのビット 4～0 のいずれかがセットされるのを待ちます。通常、セラミック発振子で発振回路のマッチングがとれていれば、100μs 程度で十分なので、ビット 4 がセットされるのを待ちます。

(2) プログラム例

・アセンブラ記述の例

```
LOOP3:
        MOV     WDTE,#0ACH      ; ウォッチドッグ・タイマ使用時だけ入れる
        BF      OSTC.4,$LOOP3   ; 発振安定を待つ
```

・C言語記述の例

```
        while(OSTC.4 == 0){
                WDTE = 0xAC;    /* ウォッチドッグ・タイマ使用時だけ */
        }                       /* 発振安定を待つ */
```

[クロックの切り替え]

X1 発振が安定したら、動作クロックを高速内蔵発振から X1 に切り替えます。切り替えが終わったら、使用しない高速内蔵発振器を停止します。これで、X1 クロックへの切り替えは完了です。あとは、PCC レジスタでクロックの分周比を設定してください。

(1) 具体的な処理

MCM レジスタの XSEL ビットと MCM0 ビットをそれぞれ1にセットすることで、動作クロックを切り替えます。切り替わったことは MCM レジスタの MCS ビットが1になることで確認します。その後、RCM レジスタの RSTOP ビットを1にセットすることで、高速内蔵発振器を停止します。

(2) プログラム例

・アセンブラ記述の例

```
        MOV     MCM,#00000101b  ; クロックを切り替える
LOOP4:
        BF      MCS,$LOOP4      ; 切り替わりを待つ
        SET1    RSTOP           ; 高速内蔵発振器の発振停止
```

・C言語記述の例

```
        MCM = 0b00000101;       /* クロックを切り替える */
        while(MCS == 0){
                NOP();
        }                       /* 切り替わりを待つ     */
        RSTOP =1;               /* 高速内蔵発振器の発振停止 */
```

1.1.5 ビット操作の注意点

ビット操作上の共通注意事項

Q1. ビット操作を行う上での共通注意事項を教えてください。

A1. これは 4 ビットの 75X から 32 ビットの V850 に共通の事項です。

・ビット操作命令

・ポートへのビット操作

マイコンのビット操作は専用のハードウエアで実現しているのではなく、リード・モディファイ・ライトにより実現しています。 次のような動作となります。

(1) データを 4/8 ビット単位で読み込む(読み込む長さは CPU で異なります)

(2) 読み込んだデータの目的のビットのセット/リセットを行う

(3) ビットを処理したデータを 4/8 ビット単位で書き戻します

通常のプログラム処理では、これらは特に意識する必要はないのですが、ポートのようなハードウエアに対してビット操作を行う場合には注意が必要です。
一部のデバイスを除いては、ポートへのビット操作を可能にするために、ポートを出力で使用している時には出力ラッチのデータを読み出し、入力で使用している時には外部端子の状態を読み出せるような回路構成となっています。

たとえば P20-P27 の 8 ビットの入出力ポートがあるとします。このポートへのアクセスは必ず 8 ビット単位で行われます。このポートに対するビット操作を行う場合、8 ビット分のデータが読み込まれます。このとき、出力に指定してあるビットでは出力ラッチのデータが読まれ、入力に指定してあるビットでは端子の状態が読まれます。

このようなビットごとに異なるところからのデータが組合わさった状態で読まれます。このデータの中の目的のビットに対して操作が行われ、結果が 8 ビットで出力ラッチに書き戻されます。つまり、入力ポートに指定されたポートの出力ラッチには外部端子の状態が書き込まれる事になります。 従って、入力に指定しているポートに関しては注意が必要になります。

以下の質問例のように、ポートを出力に指定しない状態でのポートに対するビット操作は、対象のビット以外に思いがけない影響を与えますので、注意が必要です。

Q2. 注意が必要なのは、ポートへのビット操作ということですが、内蔵周辺のモード・レジスタなどでは問題ないですか。

A2. はい、ビット操作でも問題はありません。
なお、内蔵周辺の動作中に、関連レジスタへの同じ値の書き込み可否を気にされることがあります。書き込み禁止(起動トリガになっているなど)が明記されていなければ、動作モードなどを変更するわけではないので、特に問題にはなりません(ビット操作、バイト操作とも)。

ポートの出力異常

Q1. ポート・モード(入出力設定)を設定したときに異常な出力がでないように、以下のように、あらかじめ各ビットのデータをポートのラッチに書き込んでからポート・モードを設定しましたが、意図した出力となりません。

```
P2.3 = 0;
P2.2 = 1;
P2.1 = 0;
PM2 = 0xf1;
```

先に PM を設定してから各ビットを設定すると、意図した出力になります。 PM を設定する前の各データ設定は無視されるのですか。

A1. ポートの出力データをビット単位で設定しているのが問題です。一括設定をしてください。

ビット単位の設定ではビット操作命令を使用しますが、ポート一括(全ビット)でデータを読み込んで、ビット操作を行って書き戻します。

リセット後のポートは入力のため、外部の端子状態を読み込むことになります。

つまり、P2.1=0 の実行で、その前にセットした P2.2 はクリアされてしまいます。

ポート兼用端子の誤動作 [μPD75P0016]

Q1. PORT22 をクロック出力に設定すると、電源を入れるたびにクロックが出たり出なかったりします。

オシロスコープのプローブを接続していると、95%以上の確率で動作しますが、たまにクロックが出ない場合もあります。初期化部分のソフトを次に示します。

```
MOV     A, #0011B
MOV     PCC, A
SET1    MBE
SEL     MB15
MOV     A, #0001B
MOV     CLOM, A
MOV     A, #0H
MOV     PORT2, A
MOV     PORT3, A
SET1    P2.3
MOV     PORT4, A
MOV     PORT5, A
MOV     PORT6, A
MOV     PORT7, A
MOV     XA, #10000100B
MOV     PMGB, XA
MOV     XA, #0
MOV     PMGC, XA
```

A1. ポート 2 のビット 3 のセット命令(SET1 P2.3)による操作が影響しています。

次のように、MOV 命令で操作してください。

```
MOV     A,#4H
MOV     PORT2,A
```

1.1.6 レジスタ設定順序

レジスタ設定順序の注意点

Q1. レジスタの設定順序の一般的な注意事項はありますか。

A1. マイコンに内蔵された周辺機能の設定は、ある順序に従って行う必要があります。
その順序が守られなかった場合には、その機能が動作しなかったり、外部に不正な出力が出たり、おかしな動作になることがあります。
各周辺機能で特に指定された順序がない場合、基本的には信号の伝わる順序に従って設定してください。

タイマ制御レジスタの設定順序 [V853]

Q1. タイマ制御レジスタの設定順序に、推奨（または規則）はあるですか。
対象は次のレジスタです。
- キャプチャ／コンペア・レジスタ（CC1n0-3）
- タイマ出力コントロール・レジスタ（TOC1n）
- タイマ・コントロール・レジスタ（TMC1n）
- タイマ・ユニット・モード・レジスタ（TUM1n）

A1. 先にモードを設定してから、コンペア・レジスタを設定してください。
①TMC でカウント・クロックのみを設定（CE は 0 のまま）
②TUM を設定
③TOC を設定
④CC を設定
⑤PMC を設定
⑥CE を 1 にして、TMC を設定して、タイマを起動

シリアルのレジスタ設定順序 [78K0]

Q1. μPD780034A の 3 線式シリアルで、スレーブ受信モードとマスタ送信モードを切り替えて使用していますが、マスタ送信に切り替えた直後のクロック出力ができません（続けて送信する場合は正常）。
マニュアルで P2n のラッチの指定が 0 になっているので、切り替え時に、ポート・モード(PM2)とラッチ(P2n)の設定を最初に行うと、ロウ・レベルが SCK30 から出力されます。

A1. これは当然の動作です。CSIM30 で送信動作を許可する前は、端子はポートとしての動作になります。
ポートが出力で、出力ラッチが 0 なら、端子にはロウ・レベルが出力されます。
これを避けるには、出力ラッチに 1 を設定した状態でポートを出力に設定し、CSIM30 の設定で送信動作を許可したあとに、出力ラッチを 0 に設定してください。

1.1.7 リモコン

NEC フォーマットの赤外線リモコン・フォーマット

Q1. 赤外線リモコンの信号は、どうなっているのですか。

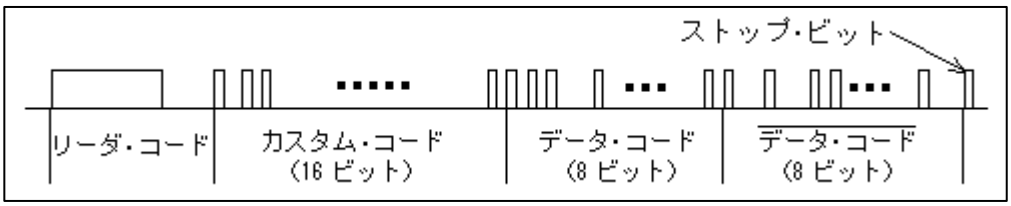
A1. [概要]

赤外線リモコンでは、約 950nm の波長の赤外線を使用して、数バイト分の情報を低速で送信します。
赤外線を用いて 0/1 のデータを送信しますが、単純に赤外線の有無でデータの 0/1 を表しているのではありません。NEC フォーマットを例にして説明します。

[概略フォーマット]

まず、赤外線のリモコン信号は、リーダ・コードから始まります。
そのあとに 16 ビットのカスタム・コード、8 ビットのデータ・コードと、その 0/1 を反転したコードが 8 ビット続き、最後にストップ・ビットがきます。
このあとに、フレーム・スペースと呼ばれる赤外線を出さない区間が続き、全体としては1フレーム(リーダ・コードからフレーム・スペースまで)は 108ms となります。

NEC フォーマットの例



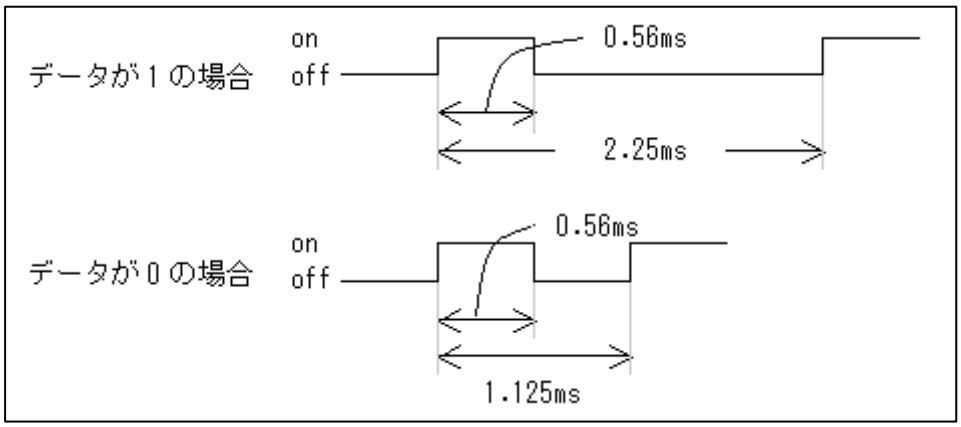
[リーダ・コード]

リーダ・コードでは、9ms の期間 on 状態が続き、そのあとに 4.5ms の期間 off 状態となります。
この部分は、その後のデータ部分とは波形(時間)が大きく異なるので、容易に識別できます。
リピートの場合、off 期間が 2.25ms で、その後のカスタム・コードやデータ・コードが省略され、すぐにストップ・ビットとなります。

[送信データ]

カスタム・コードやデータ・コードの部分が 0/1 のデータを含む部分です。
各部分はその下位ビットから順に送信されます(送信順は、後述の「データ転送順序」を参照)。
また、0/1 の区別は赤外線の有無ではなく、ビットの長さ（赤外線が出されていない期間）で区別するようになっています。したがって、カスタム・コード部分の長さは、そのデータによって変化します。
ただし、データ・コードについては、その反転データも送信するので、この部分でのトータルの 1 の数は 8 つとなり、長さは固定になります。

リモコン信号の 0/1 の違い

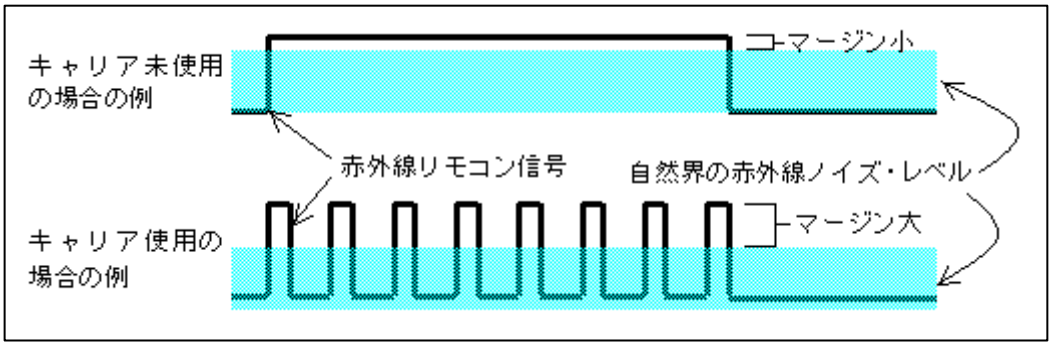


なお、on の期間は、ずっと連続して赤外線を出しているわけではありません。一定の周波数(キャリア周波数)で、赤外線を出している期間と出していない期間を繰り返しています。
このキャリア周波数は、38kHz が標準です。また、そのデューティ比は 1/3 となります。
このように設定されているのは、できるだけ消費電力を抑えるためです。

[キャリアでの変調理由]

通常の環境では、自然界にはいくつもの赤外線のノイズ源が存在しています。
これらのノイズの中で、正しく信号をやり取りするには、受信側でノイズ・レベル以上となるような強度で、赤外線を送信する必要があります。
ただし、単純に赤外線の強度を上げると、大きな電力が必要になります。このために、キャリア周波数で赤外線を on/off しています。こうすると、同じ電力でも赤外線の強度を強くできます。
この違いを模式的に表したのが、次の図です。
キャリアを用いないときにはノイズとほとんど変わらないような場合でも、キャリアを用いてピークの電力だけを大きくした場合には、トータルでは同じ電力でも、ノイズより信号を大きくできます。
また、キャリア周波数で on/off することで、受信時にその周波数成分以外にフィルタをかけることで、さらにノイズに対してマージンを大きくできます。

キャリアの有無でのリモコン信号とノイズの関係



このようなことができるのは、時間(間隔)でデータの 0/1 を表しているからです。

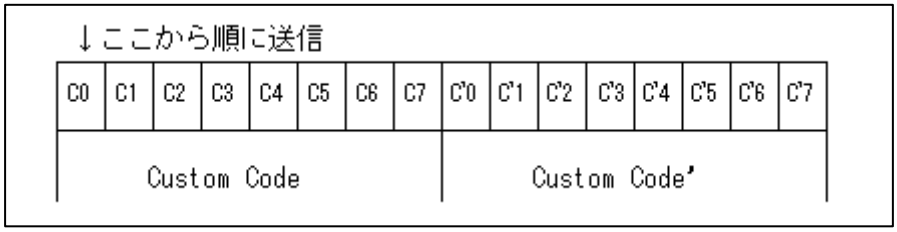
[データ転送順序]

このような方法で送信するリモコンのデータ構造は、カスタム・コードとデータ・コードから構成されています。
最初に送られるカスタム・コードは、16 ビット長ですが、2 つの 8 ビット部分に分けられます。
初期のリモコンでは、カスタム・コードは 8 ビット(C0～C7)だけで、続く 8 ビット(C'0～C'7)には、その論理を反

転したデータを送信していましたが、現在は C'0～C'7 の部分にもカスタム・コードを割り振って、16 ビットのカスタム・コードとなっています(Custom Code = □□、Custom Code' = □□ の計 16 ビットで指定)。

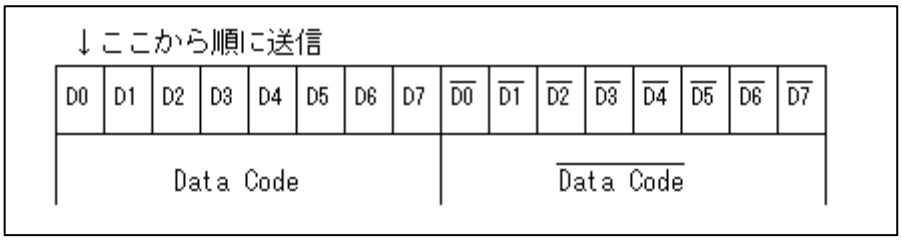
送信では Custom Code の下位(C0)から順に C7 まで出力し、その後 Custom Code'の下位(C'0) から C'7 まで出力します。

カスタム・コード部分の送信順



送信するデータ自体は 8 ビットです。
データの論理反転したデータを引き続き送信して、合計 16 ビットにしています。
受信の際には、必ず 2 つの 8 ビット・データの論理が反転していることを確認することで、エラー・チェックを行ってください。

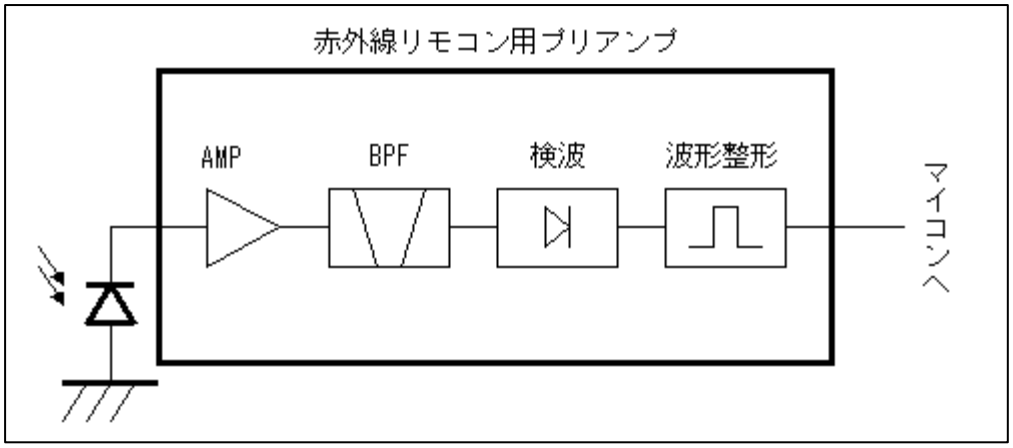
データ・コード部分の送信順



Q2. リモコン信号の受信(アナログ信号処理)は、どのように行うのですか。

A2. リモコン信号を受信する際には、赤外線信号は拡散して弱くなっています(受光ダイオードの出力で、p-p 値が数十 μV 程度)ので、赤外線受光素子の出力をプリアンプで増幅する必要があります。
また、帯域フィルタを通すことで、キャリア周波数成分だけを抽出して検波し、波形整形することで、本来のリモコン信号が得られます。この信号の on/off 期間を測ることで、どのようなリモコン信号かを判定します。

赤外線リモコン受信回路例



Q3. リモコン・コードの受信／解析は、どのように行うのですか。

A3. 受信したリモコン信号を解析するには、信号のエッジを用いて、信号の各期間の長さを測定します。
赤外線リモコン用プリアンプの出力データは負論理なので、負論理の入力信号として、以下に説明します。

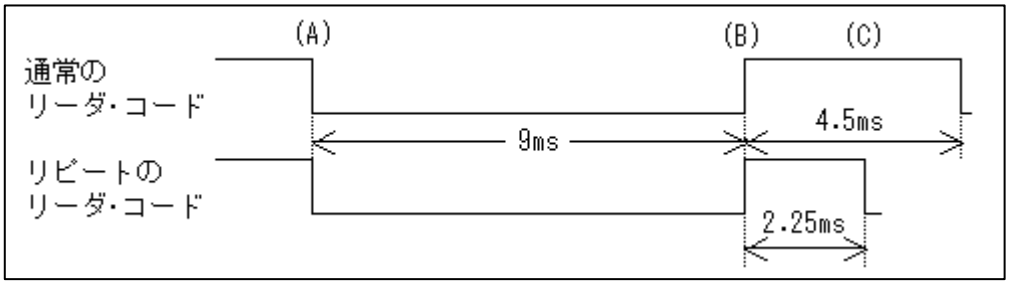
(1) リーダ・コード部

リーダ・コードは 9ms の on 期間ですから、次図の入力信号の立ち下がり(A)をエッジ検出割り込み機能を用いたり、プログラムによるポーリング処理で検出して、そこから立ち上がりまでの時間測定を行います。
このとき、タイマのキャプチャ機能を使用すると、立ち上がり(B)の検出と時間測定を同時に行えます。
このとき得られる時間が、どの範囲なら正常なリーダ・コードと判断するかは、誤動作対策を主に考えるか、リモコンの電池が消耗してもできるだけ受信できるようにするかで異なります。

他のフォーマットとして、この長さが 8.5ms 程度のものがありますので、それとヘッダ・コードだけで区別しようとすると、8.7ms 程度以上かどうかで判断する必要がありますが、フレーム全体で区別する場合には、もっと小さい値を用いて判別できます。

次に、立ち下がり(C)までの時間測定を行います。
基本的には、図のような 2 種類の時間がありえるので、得られた値が 4.5ms と 2.25ms の中間の 3.4ms 以上かどうかで、通常のコードかリピートのコードかを判別できます。
リピートの場合には、次はフレーム・スペースの確認となります。

リーダ・コードの時間関係



(2) カスタム・コードやデータ・コード部

リーダ・コードの検出が完了したあとは、カスタム・コードとデータ・コードの検出を行います。
厳密に行うには、on/off それぞれの期間を確認する必要がありますが、次の立ち下がりエッジまでの時間(1 ビットの時間)を確認するだけで、データの 0/1 を判定可能です。
2.25ms(データ＝1)か 1.125ms(データ＝0)の判定ですから、この中間値で判断するのが簡単です。
ただし、間隔が長すぎるときは、他のフォーマットでデータがすでに送り終えられた場合も考えられるので、上限値も必ず確認してください。ここでは、32 ビット分の信号がそろっているかどうかを確認してください 32 ビット分が確認できたら、次にフレーム・スペースを確認します。

(3) フレーム・スペース

ここでは、フレーム全体が 108ms になっているかどうかの確認を行いますが、それほど厳密な確認ではなく、32 ビットを越えるデータが来ないことを確認してください。
32 ビット分が受信できたら、カスタム・コードのチェック、データ・コードとその反転コードを確認してください。

NEC フォーマット

Q1. 赤外線リモコン通信に関係して、NEC 方式のフォーマットとは、どのようなものですか。

A1. NEC 方式の赤外線リモコンのフォーマットは、次のような送信をするものです。

- 38kHz のキャリア周波数
- 9ms＋4.5ms のリーダ・コード
- 16 ビットのカスタム・コード
- 8 ビットのデータ・コードとその反転コード

詳細については、リモコン用マイコンのデータ・シートをご参照ください。

「NEC フォーマットの赤外線リモコン・フォーマット」にも説明があります。

Q2. カスタム・コードとは何ですか。

A2. カスタム・コードは、赤外線リモコンを使用するセット・メーカを識別するためのもので、NEC フォーマットでは 16 ビットの長さで、データの前につけて送信するものです。

「NEC フォーマットの赤外線リモコン・フォーマット」にも説明があります。

NEC フォーマットの使用

Q1. カスタム・コードと NEC フォーマットだけで、全世界で混信や誤動作が起きないように検証していますか。また、何らかの形で保証していますか。

A1. 当社は、NEC フォーマットのコードに関しては、ダブルことがないように管理しているだけです。すべてのリモコンのコードを実際に管理しているわけではないので、混信や誤動作についての保証までは致しかねます。

Q2. NEC フォーマットを他社チップで実現するような例はありますか。

A2. 学習リモコンでは、他社のデバイスで NEC フォーマットを出力する例があるかと存じます。

Q3. NEC フォーマットを他社チップで実現した場合、著作権等についての問題はありますか。

A3. 著作権等について問題にすることは、特には考えていません。
いろいろな会社のチップで同じコードを使用した場合は、互換性を十分に評価される必要があります。

Q4. カスタム・コードを取得しないと、NEC フォーマットを使えないのですか。

A4. いいえ、カスタム・コードの 00xxH の部分は、自由に使えるようになっています。
ただし、他の機器が同じコードを使用している危険性があります。

カスタム・コードの取得

Q1. μPD6122 を使用し、カスタム・コードも取得していた製品がありましたが、生産を中止しました。新しい機種の計画があり、新しいカスタム・コードを取得する案と、従来のカスタム・コードを継承する案があります。新しいカスタム・コードは取得できますか。

A1. 基本的にカスタム・コードは、お客様 1 社に対して 1 コードとなっています。
コード数が限られるため、1 社で複数のコードの取得は、ご遠慮いただいています。
従来のカスタム・コードは有効ですので、そちらを継続してご使用ください。

Q2. リモコンを使用した機器の製造を企画しています。
カスタム・コードの申請をしたいのですが、どのようにしたらいいのですか。

A2. カスタム・コードの申請は、営業ルートで受け付けています。
お取り引きいただいている販売特約店にお申し付けください。
その際に、以下の情報をご準備ください。
- 対象デバイス名
- 貴社名称
- 所属
- 申請者氏名
- 連絡先
- ご使用装置について
  - 装置名称
  - 継続期間
  - 月々の生産予定数量

Q3. リモコンを作ろうとしていますが、製造は他社に委託する予定です。
カスタム・コードの申請はどうすればいいのですか。

A3. 実際にデバイスを購入されるルート、でお客様用のカスタム・コードの申請を行ってください。

キャリア周波数の許容誤差

Q1. リモコン用マイコンのデータ・シートに、キャリア周波数は 38kHz とありますが、誤差について記述がありません。各データ・コードの周期も、誤差の記述がありません。

A1. NEC フォーマットとしては、特に規定していません。
通常は、2kHz（5%）程度のばらつきはあるものと思われます。
なお、実際のご使用において、どこまで許すかは、受信側のマージンのとり方により決まります。
お客様のシステムとして、マージンをどの程度見るかでご判断ください。
また、データ・コードの周期の誤差に関しても、同じ比率とお考えください。

受信用のデバイス

Q1. NEC フォーマットのリモコン送信用のデバイスはあるようですが、受信用のデバイスはないのですか。

A1. ありません。
通常は、システムをコントロールしているマイコンが、リモコンの受信処理を行いますので、受信専用のデバイスはありません。
「NEC フォーマットの赤外線リモコン・フォーマット」もご参照ください。

NEC フォーマット用の標準デバイス

Q1. そのままで、NEC フォーマットを送信できる標準デバイスはないのですか。

A1. ありません。
参考となるプログラムは提供できますので、お取り引きいただいている販売店にご相談ください。

赤外線リモコンを受信するサンプル・プログラム

Q1. NEC フォーマットの赤外線リモコンを受信させる場合のサンプル・プログラムはありますか。

A1. マイコンのホームページの設計情報に、μPD780078 を用いたサンプル・プログラムがあります。
「78K0 シリーズ　サンプル・プログラム」をご参照ください。
また、受信処理については、「NEC フォーマットの赤外線リモコン・フォーマット」もご参照ください。

1.1.8 最小割り込み間隔

最小割り込み間隔

Q1. V850ES/HE2　で外部割り込みを立ち上がりエッジと立下りエッジの両エッジでかけます。立ち上がりエッジで割り込みがかかってから、次に立ち下がりエッジで割込みがかかるまで、どれだけの間隔が必要ですか。

A1. マイコンのエッジ検出機能では、500ns（アナログ・ノイズ除去）か　3　サンプル周期＋20ns（デジタル・ノイズ除去）以上の幅が必要です。
しかし、割り込み要求が発生してから、受け付けられるまでに次の割り込み要求が発生すると、先の割り込み要求は無視されます。これは、割り込み要求フラグが1ビットの情報しか保持できないためです。
つまり、割り込みの間隔として必要な時間は、個々のアプリケーション・システムやプログラムで異なります。アプリケーション的に、割り込み要求を受け付けられない時間の最悪値を見積もってください。割り込み要求を受け付けられない時間としては、以下の内容を考慮する必要があります
・マイコンが割り込み禁止状態にある時間
・マイコンは割り込み許可状態だが、より優先度の高い割り込み処理をしている時間
・割り込みを保留するような命令の実行
・割り込み要求を認識してから、受け付けるまでの時間
・DMAやバス・リクエストにより、CPUが命令を実行できない時間

【コーヒー・ブレーク】
基本的に割り込み要求は、命令と命令の区切り目で受け付けられます。このため、実行時間の長い命令を実行している間は、割り込み要求が受け付けられません。つまり、十数クロックもかかるような命令（通常は除算命令）の実行開始直後に発生した割り込み要求は、その期間は待たされてしまいます。
ただし、32 ビット RISC マイコンの V850 では、割り込み要求が発生すると、実行中の除算命令の実行を中断して、その要求を受け付けることで、割り込みへの応答時間を短くするようになっています。中断された除算命令は、割り込みから復帰後に再度（最初から）実行されます。



1.1.9 入力信号の立ち上がり／立ち下がり

端子に入力する信号の立ち上がり／立ち下がり

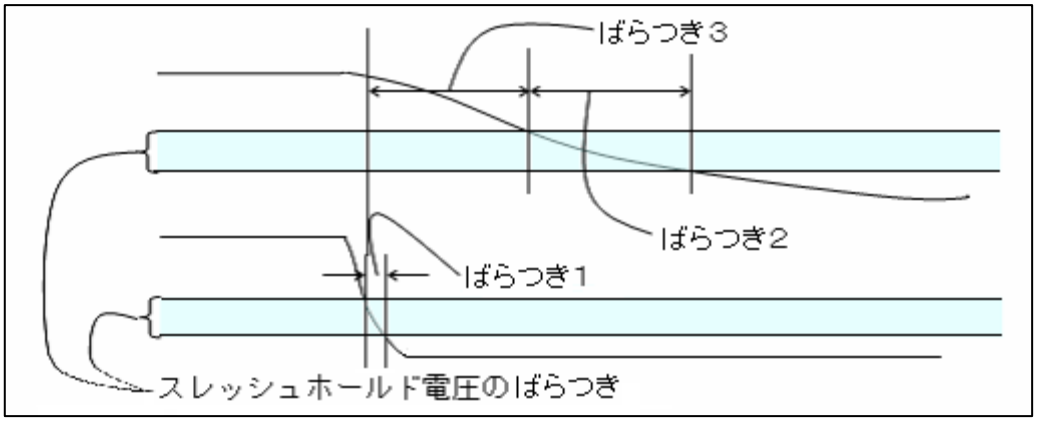
Q1. 入力する信号の立ち上がり／立ち下がりが緩やかな場合に何か問題がありますか。

A1. 入力信号の立ち上がり／立ち下がりが緩やかな場合には、大きく 3 つの問題が考えられます。

(1) タイミング変動の問題

半導体製品は、製造プロセスや使用条件で、スレッシュホールド電圧がばらつきます。このため、入力信号が緩やかに変化すると、その信号をハイ・レベル／ロウ・レベルと判断するタイミングが大きく変動します。同じスレッシュホールド電圧のばらつきに対して、変化時間が異なる信号入力の例を示します。立ち下がりが速い場合、ロウ・レベルと判定するタイミングはばらつき 1 程度なのに対して、ゆっくり立ち下がると、ばらつき 2 のように大きな差になります。さらに、本来の変化点から見ると、ばらつき 3 の遅延も加わります。

入力信号によるタイミングのばらつき

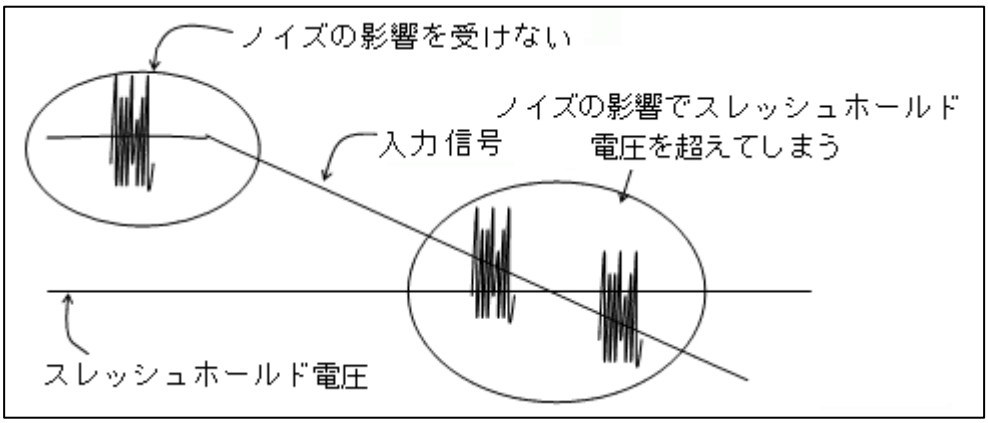


ほとんどのシステムが、信号のタイミングを元に設計されるはずです。入力信号を認識するタイミングが大きくばらつく場合は、そのばらつきが無視できる程度の遅いタイミング設計しかできなくなります。

(2) スレッシュホールド電圧付近でのノイズの問題

入力信号がスレッシュホールド電圧付近で緩やかに変化すると、そのノイズが重畳する可能性があります。このような状況では、小さなノイズでも、入力信号がスレッシュホールド電圧を越えることが考えられます。この場合、ノイズそのものが入力信号の変化とみなされ、複数の入力の順序が逆になったり、異常に短い信号が入力された状態となります。つまり、システム的にノイズ・マージンが非常に小さくなります。

ゆっくりした変化の信号に対するノイズの影響



(3) 貫通電流の問題

入力信号が中間電位では、入力バッファで貫通電流が流れることがあります。立ち上がり／立ち下がりが緩やかな入力信号では、中間電位に留まる時間が長くなるため、貫通電流の流れる可能性が高くなります。省電力のスタンバイモードを使用する場合や小さな動作電流のデバイスを使用する場合など、システムの消費電力を意識する場合には貫通電流の影響が無視できないことがあります。

いずれも、システムの設計や動作に大きな影響を与える可能性のある項目ですので、不必要に入力信号の立ち上がり／立ち下がりを緩やかにしないことをお勧めします。

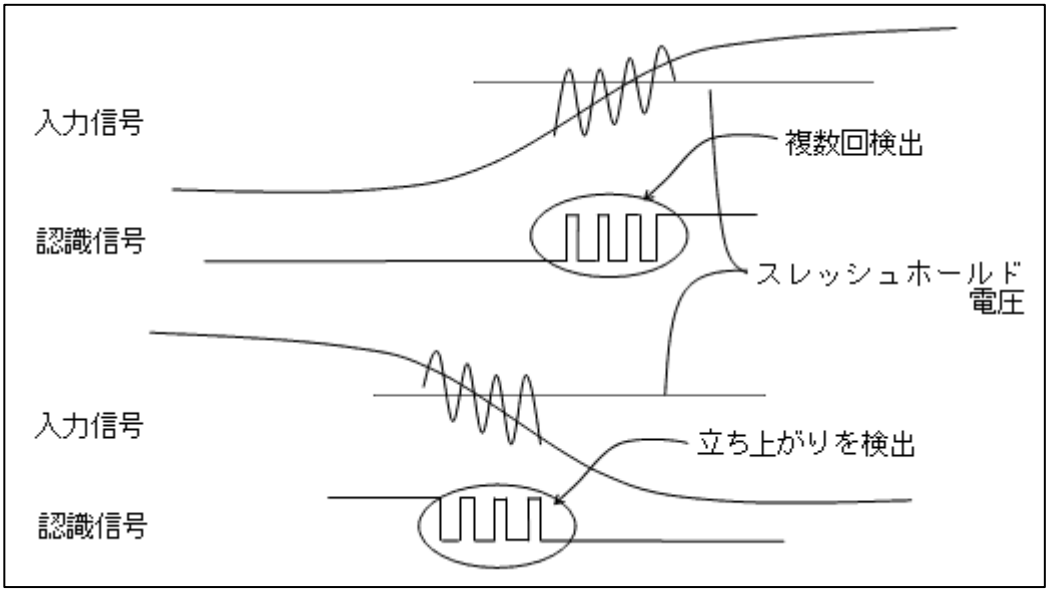
入力信号の立ち上がりエッジによる割り込み時の立ち上がりの規格

Q1. 入力信号のゆっくりした立ち上がりエッジで割り込みをかけたいのですが、立ち上がり時間についての規格はないのですか。

A1. デバイスとして、エッジ検出のための立ち上がり時間は規定していません。立ち上がりエッジの場合、ロウ・レベルと認識されている状態から、ハイ・レベルと認識される状態になれば結構です。
ただし、システムとしてみた場合、ゆっくりした立ち上がりでは、エッジを検出するタイミングがばらつきますので、そのばらつきが問題とならないかどうかをよくご検討ください。

また、ゆっくりした立ち上がりでは、ノイズの影響を受けやすくなります。次図のように、1 回の立ち上がりのつもりでも、ノイズの影響で複数回の立ち上がりエッジを検出することも考えられます。
また、本来はエッジを検出をしない立ち下がりエッジでも、ゆっくりした立ち下がりではノイズの影響で立ち上がりエッジができてしまい、予想外のタイミングで割り込みが発生することも考えられますので、ノイズの影響もご確認ください。

緩やかな変化の信号でのエッジ検出



1.2.10 シリアル通信

マイコンに内蔵されているシリアル機能

Q1. マイコンに内蔵されたシリアル・インタフェースには、どのようなものがありますか。

A1. マイコンに内蔵されたシリアル・インタフェースとしては、以下のものがあげられます。このうちで、おもに使用されるのは(1)～(3)となります。

(1)は信号レベルを変換することで、PC等で用いられてきた RS-232-C と接続することができます。

(2)、(3)はマイコン同士や、マイコンと外部の EEPROM とのボード内での接続でおもに使います。

(1) UART(Universal Asynchronous Receiver and Transmitter)

最も一般的なインタフェースです。転送できるデータ長は 7 または 8 ビットで、パリティによるエラー・チェックを行うことができます。マイコンによっては、次のような機能をサポートしたものもあります。

- **赤外線通信**
- **LIN(Local Interconnect Network)インタフェース**

また、個々のデータを送信する以外に、複数のデータを連続して送信する機能を持ったものもあります。

「シリアル I/F の選択について」や「調歩同期の基本動作」もご参照ください。

(2) 3 線式シリアル・インタフェース

シリアル・クロックと 2 本のデータの合計 3 本の信号線を用いて、比較的高速に 8 ビットのデータを双方向で転送できます。データの転送を下位ビットから行うか、上位ビットから行うかの切り替えができるものや、次のような機能をサポートしたものもあります。

- **SPI(Serial Peripheral Interface)に準拠**
- **複数データの転送**

「シリアル I/F の選択について」や「3 線式シリアル通信の基本動作」もご参照ください。

(3) I^2C バス・インタフェース

2 本の信号線を用いて、バス形式で複数の機器と通信を行うことができます。機能的には、シングル・マスタにだけ対応したものと、マルチ・マスタ機能まで対応したもの(最近のデバイスはほとんどこちら)があります。

「シリアル I/F の選択について」や「I^2C バス」もご参照ください。

(4) SMB(System Management Bus)

I^2C バスと同様に、2 本の信号線を用いたバスで通信を行うインタフェースで、設定によって、I^2C と互換性を持たせることができます。一部のデバイスに内蔵されています。

(5) CAN(Controller Area Network)

おもに車載分野で使用されているバスです。対応した規格により、何種類かのインタフェースがあります。

(6) IE-Bus

これも、おもに車載用で、CAN よりは低い転送速度をカバーしています。

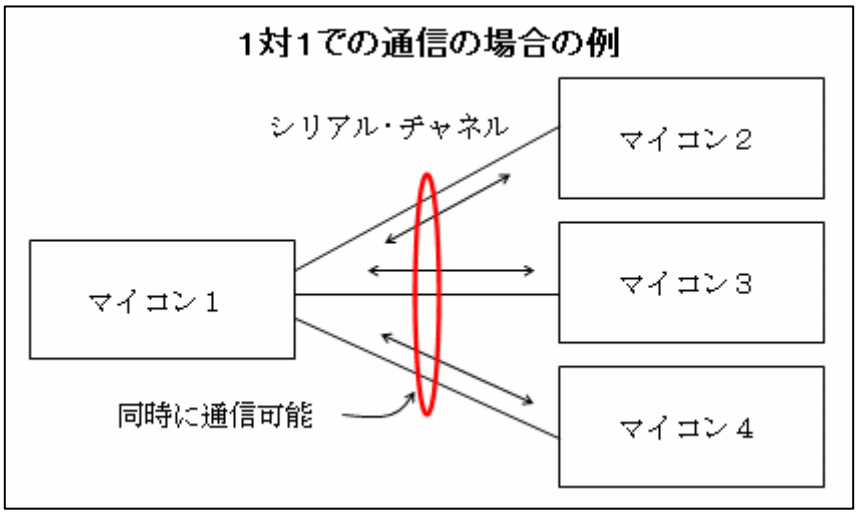
通常、マイコンに内蔵されているコントローラは、サブセット版となります。

シリアル・インタフェースの選択

マイコン(MCU:Micro Control Unit)は、デバイス単体をシステムに組み込むだけでなく、複数のマイコンを組み合わせて組み込むことも、多くの場面で見られます。また、マイコンだけでなく、外部の I/O やメモリと接続するようなアプリケーションもあります。
このような場合に、よく使用されるのがシリアル通信です。ここでは、マイコンと外部のマイコン、I/O、メモリとの通信に使用されるシリアル通信について説明します。

Q1. 複数のマイコンを使い、マイコン間をシリアル通信するシステムを考えていますが、マイコン間の通信にどのインタフェースを使うのが適していますか。

A1. まず考える必要があることは、シリアル通信の相手方は1つだけか、複数の相手が存在するかです。通常、シリアル通信は 1 対 1(Point to Point)の通信ですので、複数の相手と通信を行うには、その数だけシリアル・チャネルを準備する必要があります。たとえば、下記の例では、マイコン 1 がマイコン 2～4 と 1 対 1 での通信を行います。この場合、マイコン 1 には 3 チャネルのシリアル・インタフェースが必要で、マイコン 2～4 では、1チャネルのシリアル・インタフェースがあれば可能です。

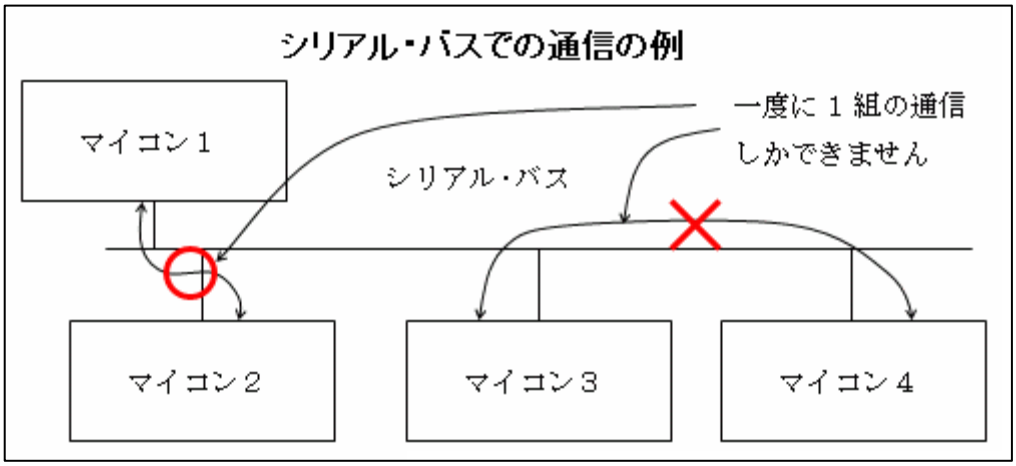


マイコン 2～4 の間でも通信を行わせる場合には、その分のシリアル・チャネルが必要になります。このような構成では、同時に通信を行えますが、通信相手が増えた場合には、その分のシリアル・チャネルが必要になってしまいます。したがって、接続する相手が少ない場合には、1 対 1 の通信をお勧めします。
この方式の代表的なシリアル通信としては、次のような機能があります。

- **調歩同期**（UART を使用）
- **3 線式シリアル通信**（SIO, CSI を使用）

調歩同期は比較的低速の通信を簡単に行え、3 線式シリアル通信は比較的高速の通信に使用できます。
調歩同期については「調歩同期の基本動作」、3 線式シリアル通信については「3 線式シリアル通信の基本動作」をご参照ください。

通信相手が多い場合には、バス形式のシリアル・バスの方が便利で簡単です。シリアル・バスを使用すれば、1 つのバスでデータのやり取りができるので、信号線やシリアル I/O の数が少なくて済みます。ただし、MCU で使用できるシリアル・バスでは、元々の転送速度が遅く、しかも 1 つのバスを共用するため、さらに転送速度は遅くなります。



この方式の代表的なシリアル通信としては、次のような機能があります。

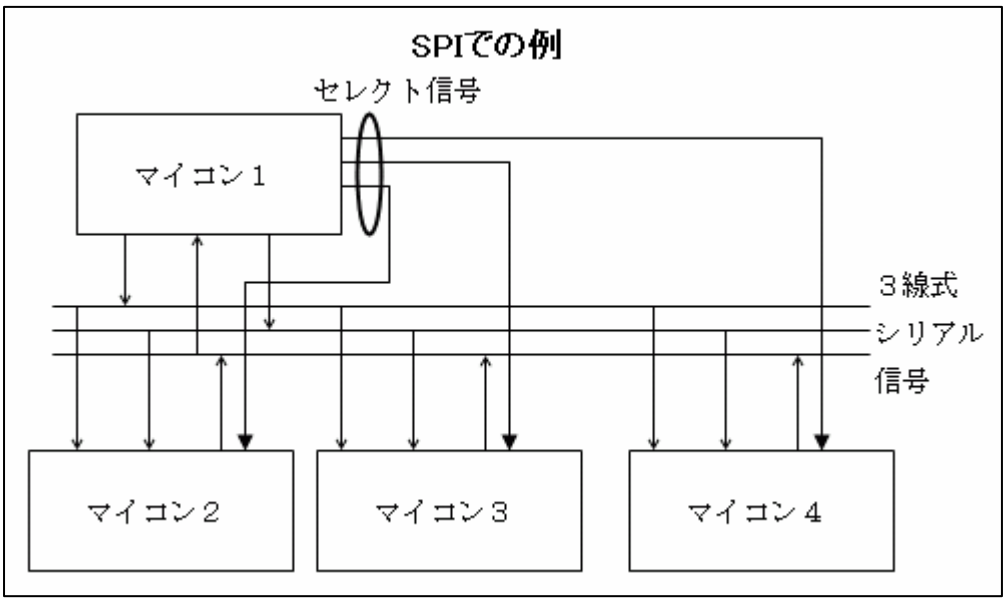
・**I²C バス**(I²C バス・コントローラを使用、特定分野では IE-Bus や CAN も使われています。)

I²C バスに関しては、「I²C バス」の FAQ もご参照ください。

これら以外に、本来は 1 対 1 の通信である 3 線式シリアルを機能拡張して、1 対複数の通信ができるようにた SPI もあります。この場合の基本的な考え方を示します。

実際のデータのやり取りを行うための信号は、3 線式シリアル通信と同じです。スレーブとなるマイコン 2～4 では、それぞれ同じ機能の信号を接続しておきます。そのままでは、シリアル出力同士で信号が衝突してしまいますので、どれか1つを選択するためのチップ・セレクト機能が追加されています。これにより、選択された1つ以外は、シリアル出力をハイ・インピーダンスにして、データの受信も送信も行わないようになります。

マスタであるマイコン 1 はポート等を使用して、通信相手をどれか1つ選択します。以降は、通常の 3 線式シリアルと同じ動作で通信を行います。

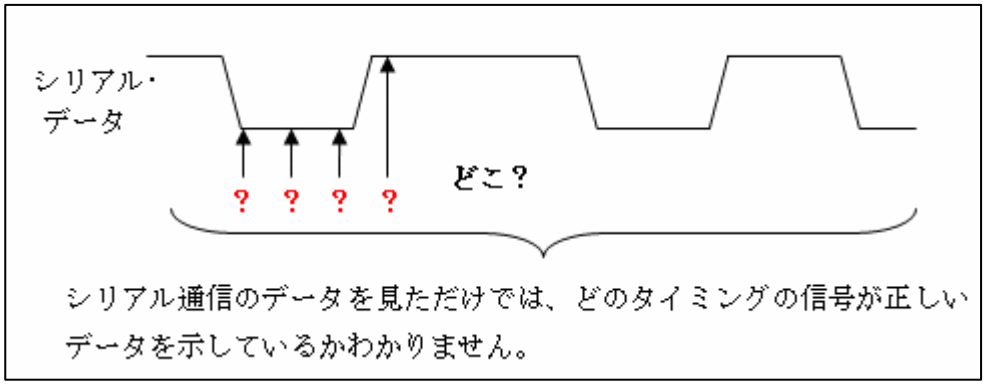


以上の、各シリアル通信方式の機能の違いと通信速度を考慮して選択してください。

シリアル通信の同期／非同期の違い

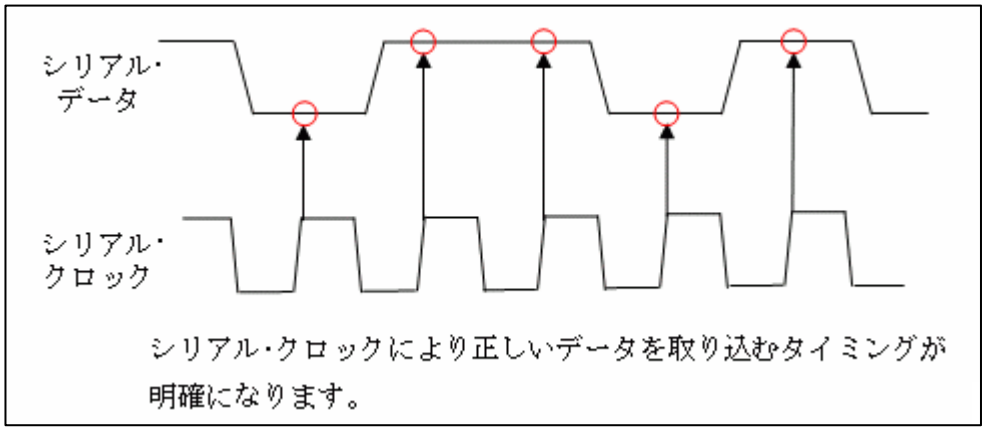
Q1. マイコンに内蔵されているシリアル通信機能で、クロックがあるものとないものがありますが、なぜこのような違いがあるのですか。

A1. シリアル通信では、たとえば 1 バイト・データをビット・データに分解して 1 ビットずつ送信します。このようにして送信されたデータを受信側では、元のデータに再構成する必要があります。この際、送られてきた信号のどこが有効なデータなのか、どこからどこまでがまとまり(バイト・データ)なのかが分からないと、元のデータに戻せません。



(a) 同期式通信

どこが有効なデータかを示す簡単な方法は、そのための信号を同時にやり取りすることです。このための信号が(シリアル)クロック信号です。下記の同期通信の例のように、クロックの立ち下がりでシリアル・データを変化させるようにしておけば、クロックの立ち上がりでは正しいデータが取り込めます。



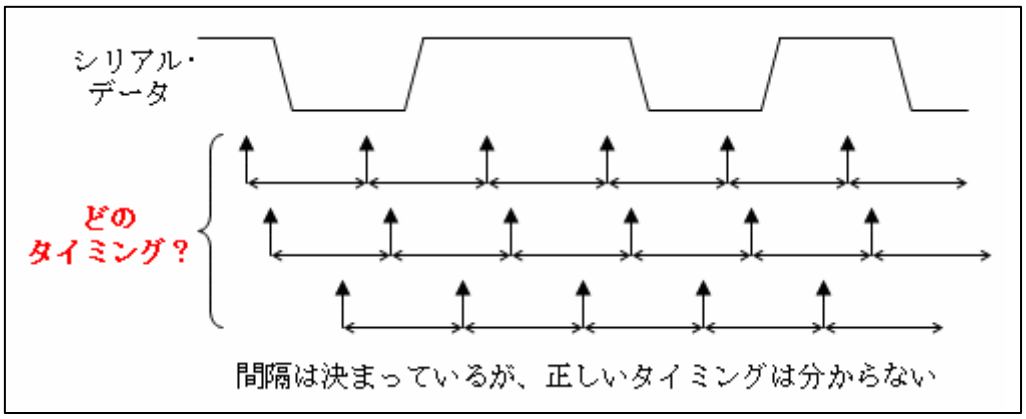
これで正しいデータ(ビット)の位置が明確になりましたが、これを「ビット同期が取れた」と表現します。このように、クロック信号をやり取りしてビット同期をとる通信方式を(クロック)同期式通信と呼びます。通常、マイコンに内蔵されている同期式通信機能では、データを転送するときだけクロックをやり取りします。このため、起動の順番を誤らなければ、どこがデータの先頭で、どこまでが 1 キャラクタのデータかが、明確になります(これをキャラクタ同期やバイト同期などと呼びます)。

この方式では、転送速度がデータと同時にやり取りされるクロック信号で決まるため、転送速度を比較的自由に高くできます。しかし、クロックをやり取りするため、クロックの受け取りを 1 つでもミスすると、データがずれてしまいます。このため、送信データの途中から受信を開始した場合などには、データのビット位置がずれて、その後は正しいデータの再生(受信)ができなくなります。これを避けるには、通信の途切れたところで、シリアル・コントローラを初期化する必要があります。「ハンドシェーク」もご参照ください。

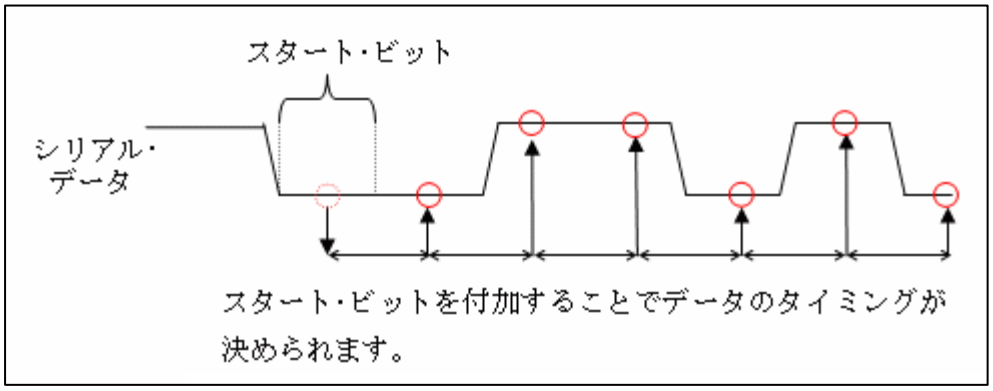
備考 一般的な通信方式としての同期式通信の場合、クロックは連続しています。したがって、どこからどこまでがデータなのかを判別することができません。この対策として、同期キャラクタをやり取りすることでキャラクタ同期をとり、どこからどこまでがデータかを明確にする方式があります。この方式では、データを転送していないときは、同期キャラクタが転送されています。このように、同じ同期通信でも動作が異なります。

(b) 調歩同期通信（非同期通信）

シリアル・クロックを用いることなく、データのやり取りを行う通信方式として、調歩同期通信(非同期通信)があります。この方式では、データ信号だけをやりとりするので、送信側と受信側で事前に転送速度を合わせておく必要があります。これで、データ(ビット)の間隔は決まりますが、これだけでは、どこが正しいデータかは分かりません。



データ信号そのものから正しいタイミングを再生するために、データ部以外にタイミングを決定するための部分が必要になります。このためにスタート・ビットやストップ・ビットといった制御部が信号に付加されてやり取りされます。



上図のように、スタート・ビットが頭に付加されることで、信号の先頭が明確になり、ビット同期とキャラクタ同期を取ることができます。また、各キャラクタのスタート・ビットを有効にするには、データの最後にストップ・ビットを付加して、その後いったんシリアル・データをハイにして、通信データがない状態ではハイを保持する必要があります。

受信側がスタート・ビットを正しいタイミングで検出するためには、ビット間隔よりも短い間隔でシリアル・データを監視する必要があります。このために、通信速度を同期通信ほど高速にすることができません。

なお、スタート・ビットとデータが 0 のビットとは、基本的に差がありません。そのため、送信されているデータの途中から受信を開始した場合には、誤ったデータを受信してしまいます。この場合には、1 キャラクタ分のデータがない部分が途中あれば、それ以降は正しく受信できるようになります。

備考　基本的に調歩同期通信では、送信側と受信側で転送速度を合わせておく必要があります。ただし、通信の頭で特定のデータ(通常は 55H)を送ることで、転送レートの合わせこみを行えます。また受信側で、シリアル・データの信号幅をタイマ・カウンタで測定して、転送速度を求めることもできます。また、9600/19200/38400 ボーのように、使用される転送速度がある程度限られている場合には、可能性のあるもっとも高速の転送速度で受信します。受信したデータの個数と値から、転送速度を類推できます。

★

ボー・レート誤差

Q1. データ転送速度の誤差は、どのように考えればいいですか。

A1. データ転送では、ジッタなどビットごとのゆらぎが発生しますが、これはボー・レート誤差という考え方をしません。実動作のボー・レートは、ボー・レート・ジェネレータなどの生成クロックの周波数で決定され、ターゲットとするボー・レートの**公称値との差**が誤差となります。ボー・レート・ファクタの設定で微調整できるシリアル・インタフェースでは、かなり誤差をしぼることができますが、2^n分周するだけの場合は、源クロックの周波数によっては、かなりの誤差が出ます。また、送信側と受信側で、早い／遅いという特性が逆の場合、相対的に誤差が大きくなることになります。クロック同期では双方が共通クロックでデータ転送するため、かなりの誤差を許容できますが、調歩同期(UART)では双方が独立したクロックで動作するため、許容誤差がかなり限定されます。

次の表は、いろいろな源クロックを 2^n分周した周波数を表します(32.768KHz は 1 秒の周期を得られるので、通信よりむしろ時計用タイマで使用されます)。たとえば 9600 ボーという公称値に対して、4.9152MHz を 512 分周すると実動作も 9600 ボーになります。ここで、送信側と受信側が 4.5MHz と 5MHz をそれぞれ源クロックにした場合、4.5MHz では 8789 ボー、5MHz では 9765 ボーが、もっとも 9600 ボーに近い値です。公称値との誤差は、8789 ボーで－8.4%、9765 ボーで＋1.7%なので、相対的に 10.1%の誤差となります。UART では、相対的に 4%程度の誤差しか許容できないため、この場合は正常な受信ができないことになります。また、実動作では**発振周波数の誤差**がありますので、それも影響します。

なお、クロック間の相対誤差比率は分周では変わらず、相対的には源クロックの周波数誤差が支配的です。

分周クロックの周波数(Hz)

分周比	4.9152MHz	4.19MHz	4.5MHz	5MHz	6MHz	32.768kHz
$1/2^0$(＝1/1)	4915200	4190000	4500000	5000000	6000000	32768
$1/2^1$(＝1/2)	2457600	2095000	2250000	2500000	3000000	16384
$1/2^2$(＝1/4)	1228800	1047500	1125000	1250000	1500000	8192
$1/2^3$(＝1/8)	614400	523750	562500	625000	750000	4096
$1/2^4$(＝1/16)	307200	261875	281250	312500	375000	2048
$1/2^5$(＝1/32)	153600	130937.5	140625	156250	187500	1024
$1/2^6$(＝1/64)	76800	65468.75	70312.5	78125	93750	512
$1/2^7$(＝1/128)	38400	32734.38	35156.25	39062.5	46875	256
$1/2^8$(＝1/256)	19200	16367.19	17578.13	19531.25	23437.5	128
$1/2^9$(＝1/512)	9600	8183.594	8789.063	9765.625	11718.75	64
$1/2^{10}$(＝1/1024)	4800	4091.797	4394.531	4882.813	5859.375	32
$1/2^{11}$(＝1/2048)	2400	2045.898	2197.266	2441.406	2929.688	16
$1/2^{12}$(＝1/4096)	1200	1022.949	1098.633	1220.703	1464.844	8
$1/2^{13}$(＝1/8192)	600	511.4746	549.3164	610.3516	732.4219	4
$1/2^{14}$(＝1/16384)	300	255.7373	274.6582	305.1758	366.2109	2
$1/2^{15}$(＝1/32768)	150	127.8687	137.3291	152.5879	183.1055	1

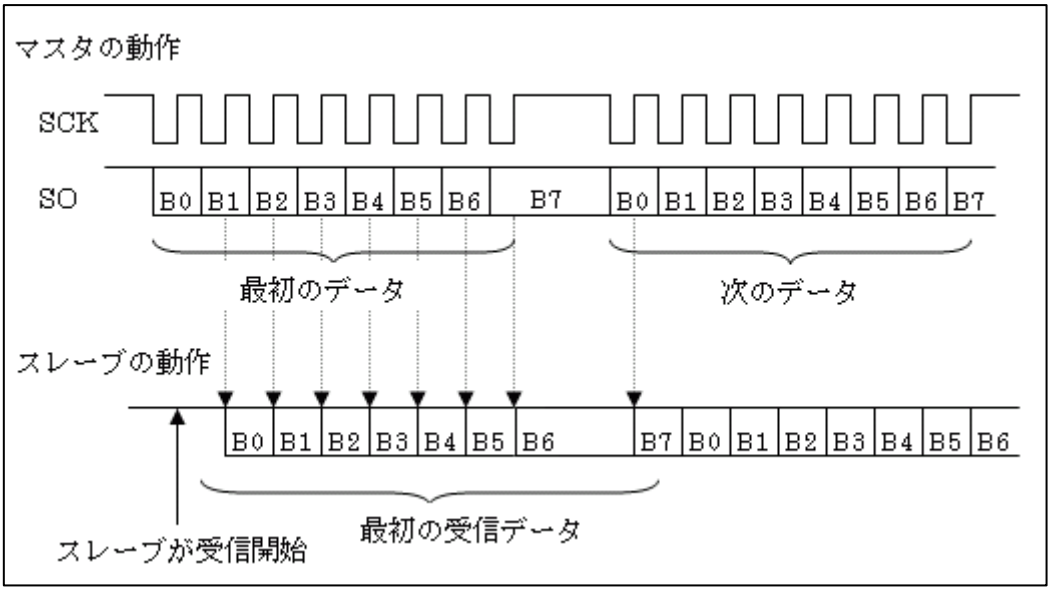
ハンドシェーク

マイコンに内蔵されたシリアル通信では、シリアル・バスを除いて、「通信の準備ができているか」といったハンドシェーク(プロトコル)は、アプリケーション側で準備する必要があります。ここでは、その結果発生する典型的な現象を Q1 に、その対策であるハンドシェークを Q2 に示します。

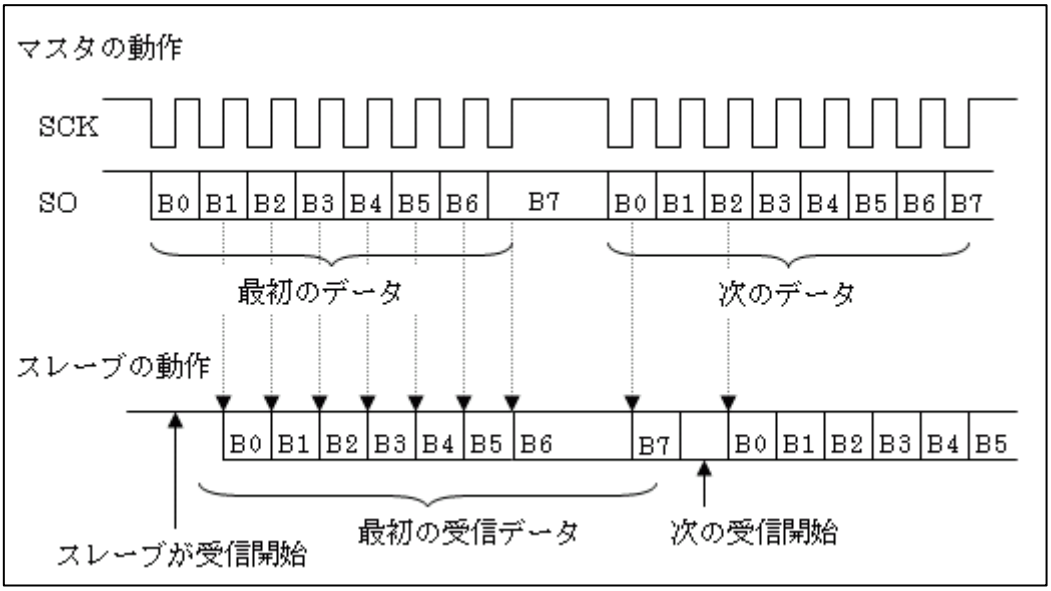
Q1. 2 つのマイコンの間で、3 線式シリアル通信を行おうとしているのですが、正しく通信できません。波形を確認すると、マスタからの送信では正常なデータが送られているのに、正しくデータを受信できません。スレーブからの送信では、意図したデータがでてきません。また、スレーブの転送完了割り込みが 1 回足りません。転送速度が高いと症状がひどく、転送完了割り込みの不足が多くなります。

A1. マスタが通信を開始した時点で、スレーブ側の通信準備ができていないためです。下記のタイミング図で、発生している現象を説明します。

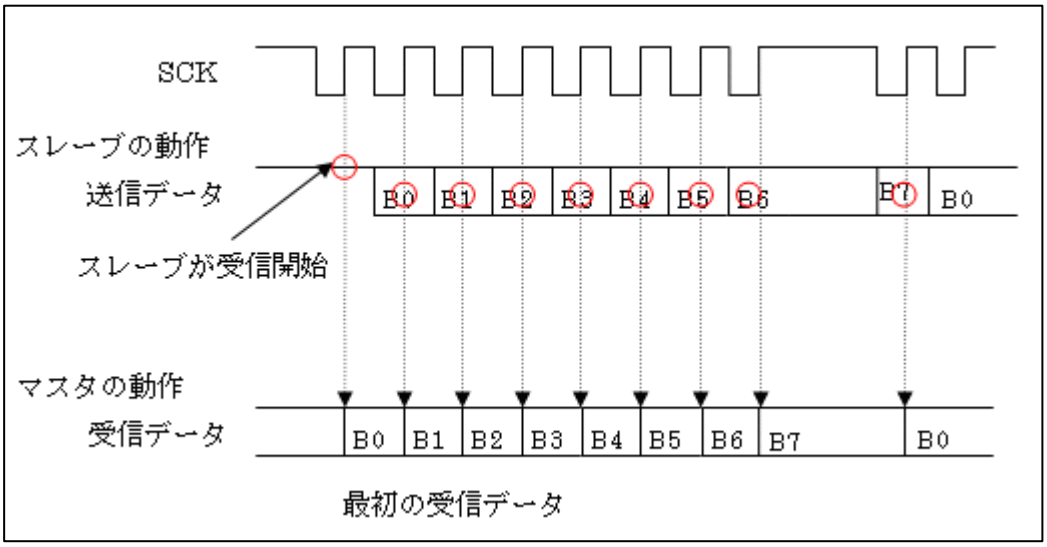
マスタは送信を開始して、クロック(SCK)を立ち下げ、最初のデータ(B0)を出力します。しかし、スレーブはまだ通信の準備が完了していないので、このデータを取りこぼしてしまいます。その後、スレーブが動作を開始しますが、下図の例では B1 を最初のデータと判断してしまいます。マスタは、B7 まで出力して最初のデータ送信を完了しますが、スレーブは、まだデータが揃っていないため、次のデータを待っています。マスタが次のデータの B0 を出力すると、スレーブはそこまでを最初のデータとして取り込み、転送完了の割り込みを発生します。このときスレーブが受信したデータは B1～B7,B0 となり、マスタが送信した B0～B7 と異なる結果となります。このように、マスタとスレーブで 1 ビット分のずれがあり、それが最後まで残ってしまうことになります。このため、マスタが n 個のデータの送信を完了しても、スレーブはまだ n 個目の受信中のため、転送完了の割り込みは n－1 回しか発生しません。



なお、スレーブがデータを受信して、次のデータの受信準備を行うまでの時間が、シリアル・データの転送速度よりも長くなると、ずれはさらに拡大していきます。この様子を下記タイミング図に示します。マスタからの次のデータの 1 ビット目(B0)で、スレーブは最初の受信を完了しますが、スレーブでの処理時間の関係で、マスタからのデータの 2 ビット目(B1)で次の受信を開始すると、3 ビット目(B2)を最初のビットとして受信します。この結果、マスタとスレーブでのビットのずれは増大していくことになります。



また、スレーブ送信の場合もビットずれが発生するので、正常なデータが送信されません。下記のタイミング図に示す例のように、マスタが受信動作を開始した時点では、スレーブは送信の準備ができていないため、マスタが受信した最初のビットは、そのときのスレーブの SO 端子の状態となります。その後、スレーブから送信データが出力されてきますが、マスタはそれを 2 ビット目(B1)以降として受信します。



Q2. 上記について、どのような対策をすればいいですか。

A2. ハンドシェーク信号を追加してください(データ自身にチェック用のデータを追加して、通信の信頼性を高める方法もありますが、補助的な手段であり、タイミングの差が発生しないようにすることが基本です)。
通常、データのやり取りを行う場合には、相手の準備ができているかどうかを事前に調べておく必要があります。しかし、(シリアル・バスを除く)シリアル通信信号自体には、そのような機能が準備されていないため、別に信号(これをハンドシェーク信号と呼びます)を準備する必要があります。

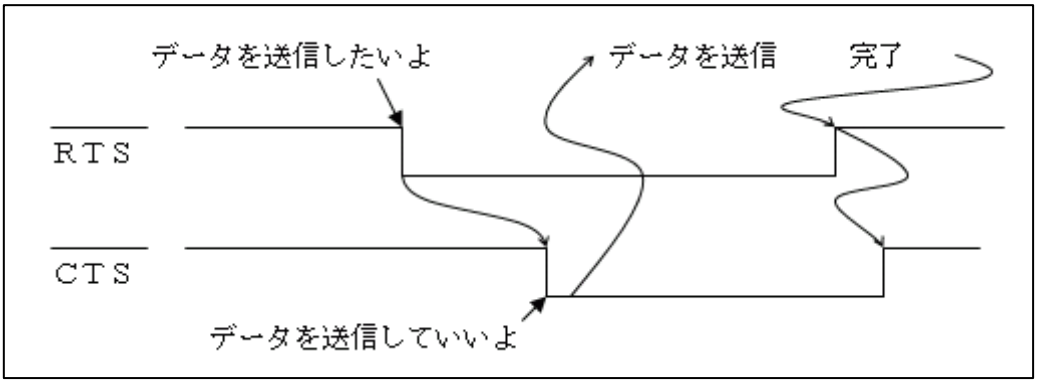
[制御信号の例]

このための信号として、モデムで使用されている次のような信号があります。

送信要求信号 RTS(Request To Send)

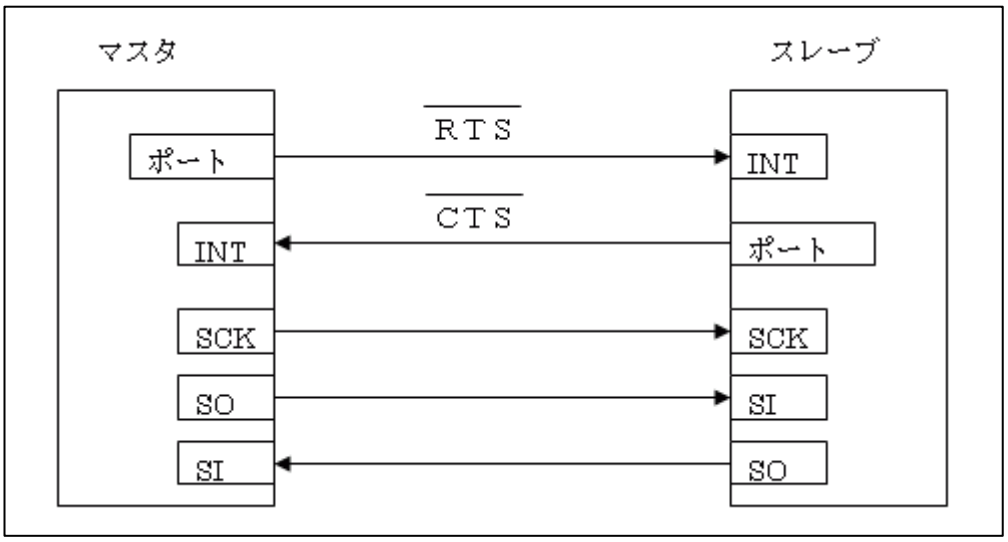
送信許可信号 CTS(Clear To Send)

送信したいときにRTS信号をアクティブにして、受信側に通知します。受信側は受信準備ができると、CTS信号をアクティブにして、送信していいことを送信側に通知します。このときのシーケンス図を下に示します。



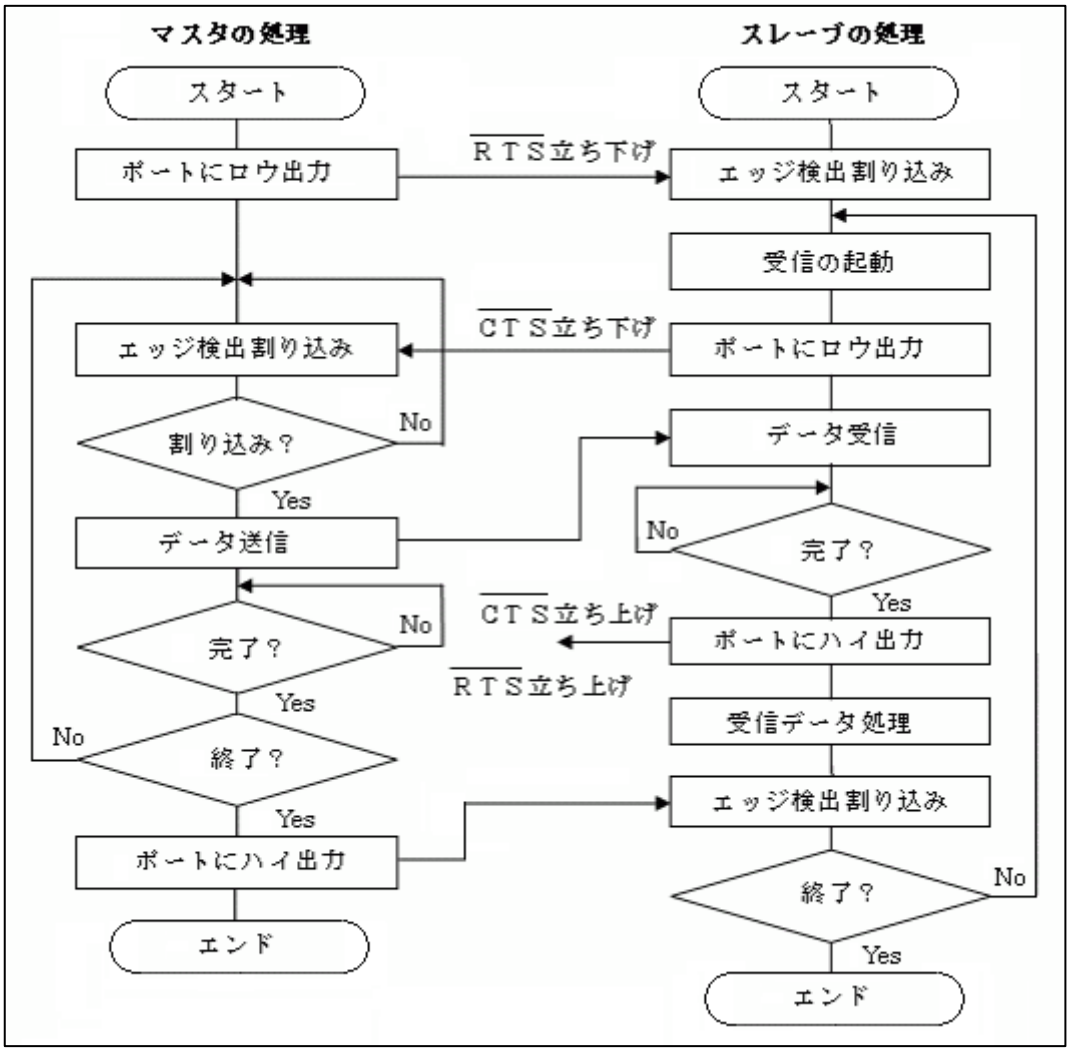
なお、モデムで使用しているこれらのハンドシェーク信号は、そのまま通信の細かな制御には使えません。そこで、これをマイコン間でのローカルな通信用に、定義の変更を考えます。次に新しい定義を示します。

信号	定　　義
RTS	マスタは、送信を行いたいときにロウを出力する。 スレーブは、その立ち下がりを送信要求と判断し、立ち上がりを送信完了と判断する。
CTS	スレーブは、受信可能になったら立ち下がりエッジを出力する(データを受信したら、いったんハイを出力し、次のデータの受信準備ができたら、立ち下がりエッジを出力する)。 マスタは、立ち下がりエッジを検出したら、1 キャラクタのデータを送信する。



これらの信号を処理するために、上記の接続を行います。マスタはポートを用いて、RTS 信号を出力し、スレーブはそれをエッジ検出割り込みで監視します。RTS 信号は立ち上がりと立ち下がりの両方のエッジを使用するので、エッジ検出割り込みとポートで、信号の状態を読み込みます。CTS 信号は立ち下がりエッジだけを使用するので、スレーブがポートで出力した信号をマスタが立ち下がりエッジ検出割り込みで監視します。

[制御信号を用いた送信処理手順]



上記処理手順に従って説明します。なお、マスタ、スレーブともに、ポートの初期状態はハイとします。

(1) マスタは、送信前にスレーブへ RTS 信号(立ち下がり)をポート出力し、スレーブの送信許可を待ちます。

(2) スレーブはエッジ検出割り込みと RTS 信号のレベル確認により、マスタからの送信要求を確認し、受信の準備(受信起動)を行います(スレーブで受信起動しても、マスタから SCK(シリアル・クロック)が来ないと、実際の転送はおこなわれません)。

(3) スレーブは受信起動したら、ポートにロウを出力することで、マスタに CTS 信号の立ち下がりエッジを送り、準備ができたことを知らせます。

(4) マスタは、スレーブからの CTS 信号の立ち下がりエッジを検出すると、データを送信します。

(5) スレーブは、マスタからのデータを受信します。

(6) マスタは送信完了の割り込みを待ち、送信完了すると、次のデータがあるかを判断して、次のデータがあれば、(3)の CTS 信号の立ち下がり待ちに戻ります。

(7) スレーブは受信完了したら、ポートにハイを出力して CTS 信号を立ち上げ、受信データを処理します。

(8) 送信が完了したら、マスタは RTS 信号を立ち上げ、スレーブに送信が完了したことを通知します。

(9) スレーブは受信したデータの処理を行い、マスタからの RTS 信号が立ち上がったことをエッジ検出割り込みとレベルで確認したら、転送を終了します。RTS 信号が立ち上がらない場合は、次の受信に戻ります。

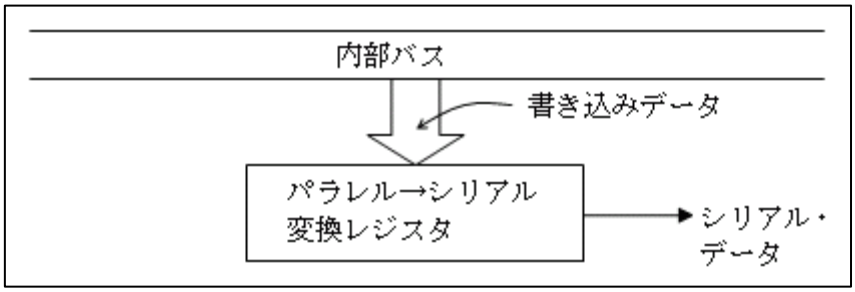
この例では RTS 信号を使用しましたが、スレーブがつねに受信準備をしていて問題がない場合には、RTS 信号は不要です。

スレーブが送信を行う場合も、同じような処理で実現可能です。この場合には、CTS 信号がデータの準備ができたことを意味して、RTS 信号はスレーブからのデータを要求する意味するようになります。

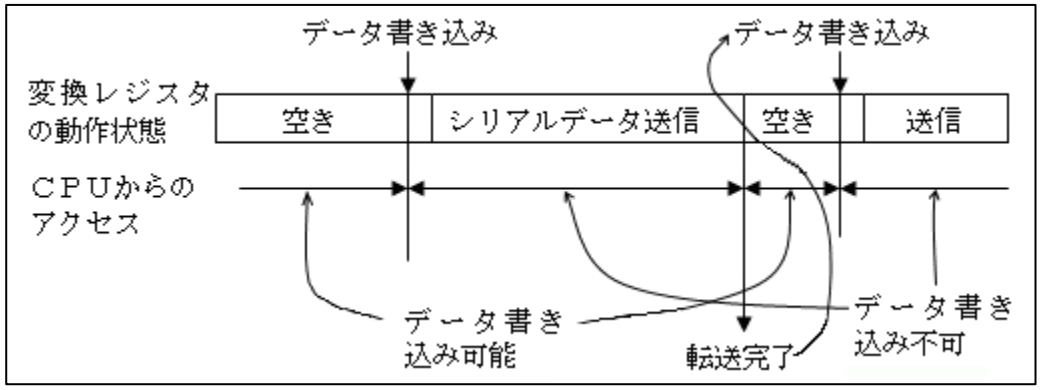
シリアルのバッファ構成

Q1. デバイスによっては、送信が完了する前に次のデータを書き込めるものと、送信が完了してからでないと書き込めないものがありますが、なぜ違いがあるのですか。

A1. これは、シリアル送信部の構成の違いによります。シリアル通信を行うには、パラレル・データとして書き込まれたデータを1ビットずつ出力するための、パラレル・シリアル変換レジスタが必要です。変換レジスタに直接データを書き込んで変換する構成では、構造は簡単ですみます。



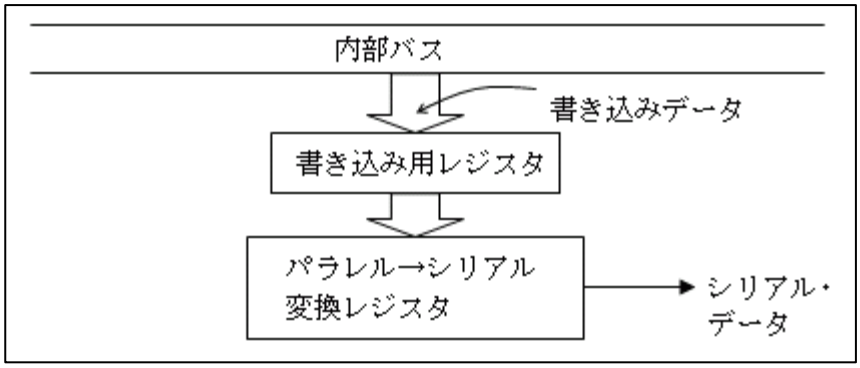
しかしこの構成では、シリアル・データ送信中に変換レジスタは占有されてしまうので、すべてのビットの送信を完了するまでは、次のデータは書き込めないことになります。



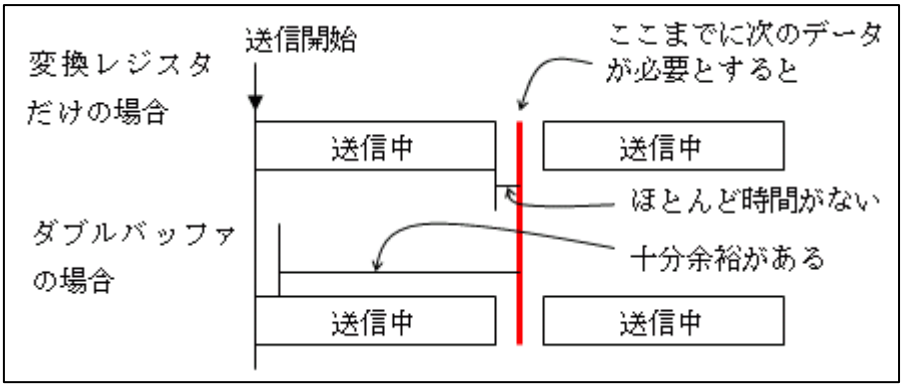
このため、複数のデータの送信を行う場合には、次のような時間が必要となります

実効的な転送時間＝実際の転送時間＋転送完了から次のデータ準備の時間

このように、実効的な転送速度を速くしようとすると、CPU の処理時間に対する制約が発生します。その対策として、変換レジスタとは別に書き込み用のレジスタを準備して、ダブル・バッファにしたものがあります。



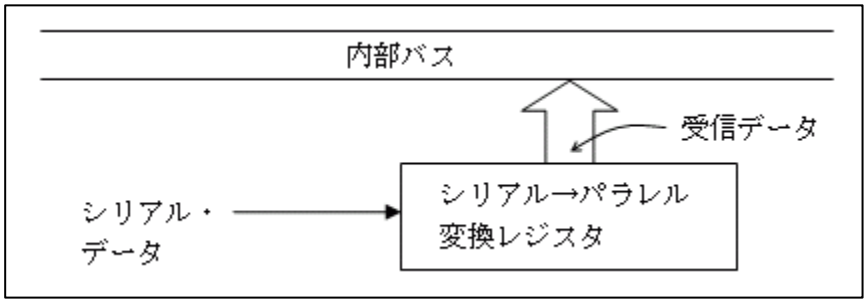
この構成では、CPU は書き込み用のレジスタにデータを書き込みます。書き込み用のレジスタに書き込まれたデータは、変換レジスタが空くとそこへ転送されます。これらの構成での、タイミングの差を次に示します。



変換レジスタだけの場合には、送信完了後に次のデータを書き込みますが、ダブル・バッファにすることで、データ送信中にも書けるので、CPU に十分な処理時間を割り当てることができます。

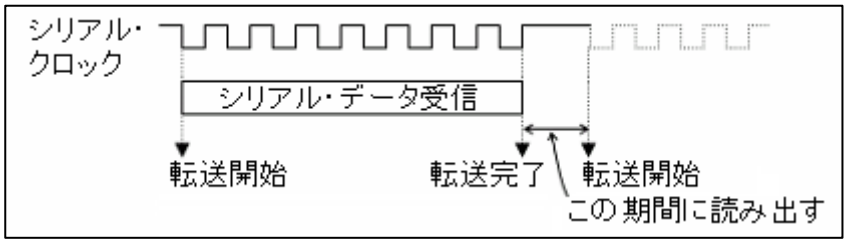
Q2. 連続したシリアルでのデータ転送を考えていますが、データ受信を完了してから、CPU がそれを読み出すまでには、どの程度の余裕がありますか。

A2. シリアル送信部の構成によって違います。一般的な 3 線式シリアルの、受信部のブロックを下記に示します。

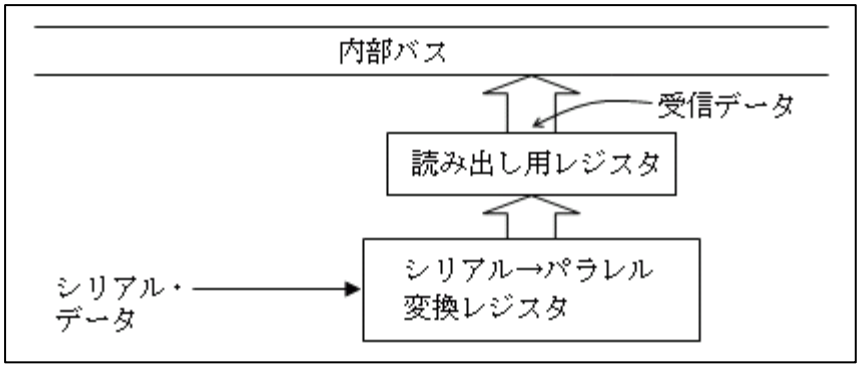


この構成では、受信したデータは変換レジスタに残ってしまうので、次のデータの転送までには、読み出す必要があります(通常の受信専用の動作では、読み出しが次の受信の起動条件になっています)。スレーブの受信では、ここまでに読み出さないと、次のデータの受信が正常にできません(ハンドシェークも参照)。

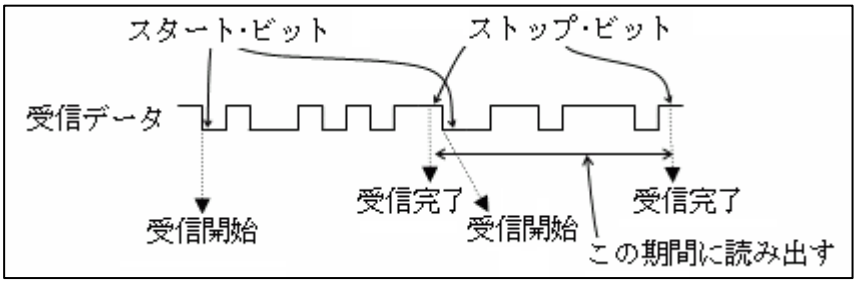
このように、「受信完了のシリアル・クロックの立ち上がりから、次の送信開始の立ち下がりまでの期間」という短い余裕しかありません。これで対応できない場合には、自動送受信機能をもった 3 線式シリアル・チャネルを使ってください。



一方、一般的な調歩同期通信用の UART の場合には、下記のようにシリアル・データをパラレル・データに変換する変換レジスタとは別に、読み出し用のレジスタを準備しており、ダブル・バッファ構成になっています。



この構成での、8 ビット・データ、パリティなし、ストップ・ビット 1 ビットの場合のタイミング例を次に示します。



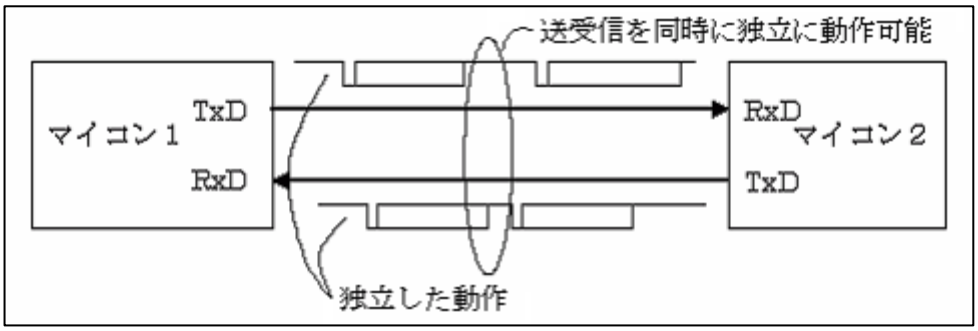
この例のように、受信したデータは、次の受信完了までの 1 キャラクタ分の時間に読み出せばよいので、処理には十分な余裕があります。

全二重通信／半二重通信

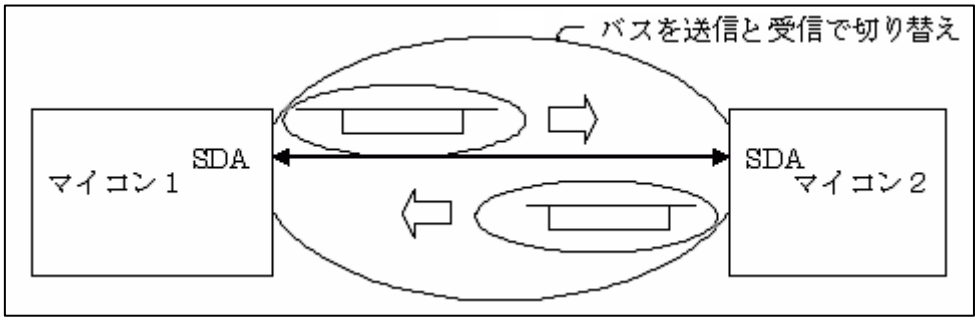
全二重通信や半二重通信については通信関連用語集をご参照ください。

Q1. 通信には全二重通信／半二重通信がありますが、マイコンに内蔵されたシリアル・インタフェースは全二重通信をサポートしていますか。

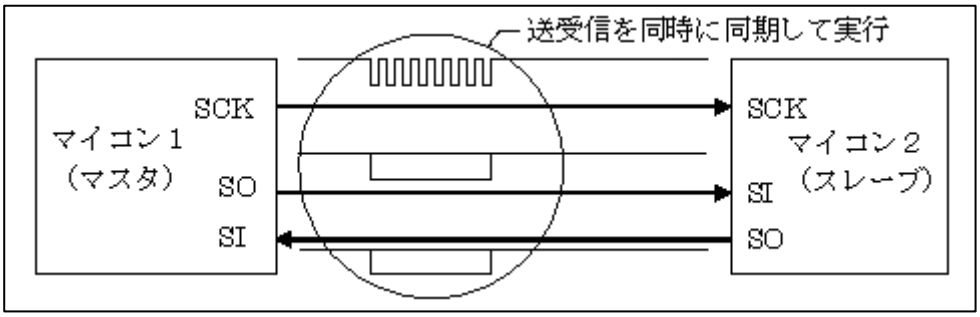
A1. マイコンに内蔵されたシリアル・インタフェースでは、UART は全二重通信が可能です。



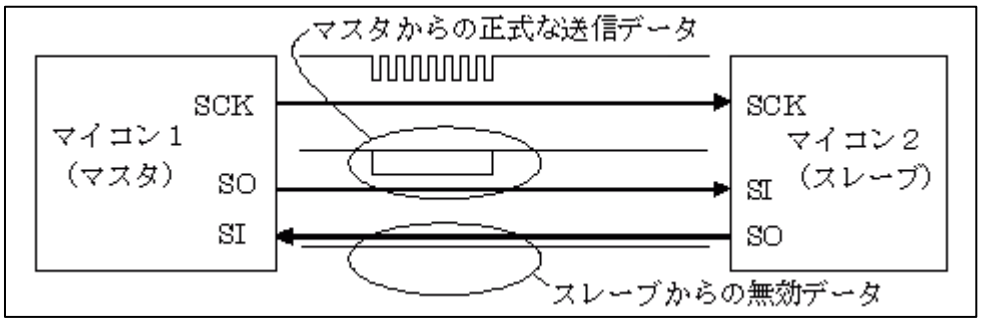
I²C バスのようなバス形式のシリアル・インタフェースでは、基本的に同時に片方向にしかデータを送れませんので、半二重通信となります。



3 線式シリアルの場合には、同時に双方向にデータを送ることができますので、全二重通信は可能です。



ただし、UART のようにデータの送受信が独立に動作できるわけではなく、送信と受信はシリアル・クロック(SCK)に同期して同時に行われます。つまり、送信と受信は全く同じタイミングで行う必要があります。そのため、送受信モードを使用して全二重通信を行う場合には、受信したデータが無効な場合が考えられます。
次図に、マスタが送信した際に、スレーブからの送信データなかった場合の例を示します。通常は、双方に送りたいデータが同時にある可能性は低いことから、このような動作になることは十分考えられます。これに対応するには、オール 1 やオール 0 のようなデータは、無効データとして無視する必要があります。



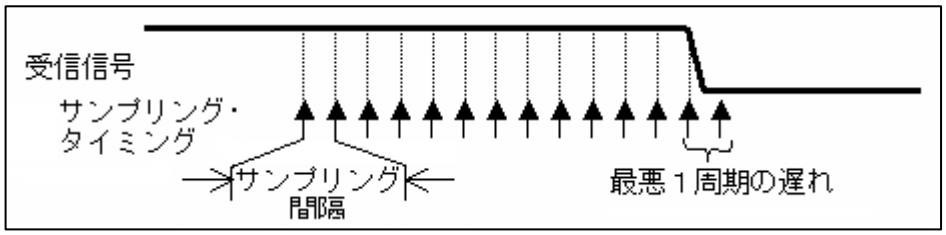
調歩同期の基本動作 [ほとんどのマイコン共通]

Q1. マイコン間で、UART のシリアル通信を行います。各マイコンの UART 受信で、スタート・ビットやキャラクタ・ビットなどの検出方法、タイミングなどを教えてください。

A1. UART 受信の場合、受信データのビット同期とキャラクタの同期をとるために、スタート・ビットを用います。
UART が受信を開始すると、次のような動作をします。

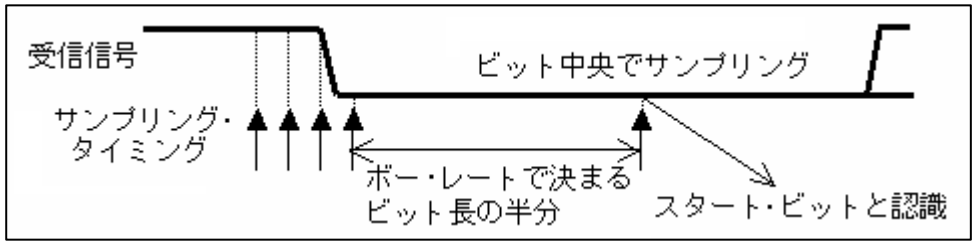
(1) スタート・ビットの始まりの検出

スタート・ビット検出のために、受信信号のサンプリングをします。このサンプリング・クロックとしては、ボー・レートを生成する元のクロック(システム・クロックやボー・レート・ジェネレータの入力)を使用します。
したがって、最悪はサンプリング・クロックの 1 周期分だけ、検出が遅れる可能性があります。
サンプリングした結果がロウ・レベルであれば、スタート・ビット検出に移ります。

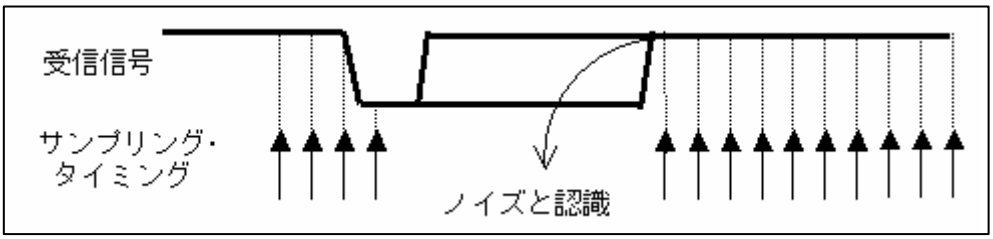


(2) スタート・ビットの検出

受信信号がロウ・レベルになったことを検出して、ボー・レート・ジェネレータのカウンタが動作を開始します。
スタート・ビット検出は、以下のようにボー・レートで決まるビット長の 1/2 の位置で行います。
このタイミングで再度サンプリングして、ロウ・レベルであれば、スタート・ビットであると判断します。

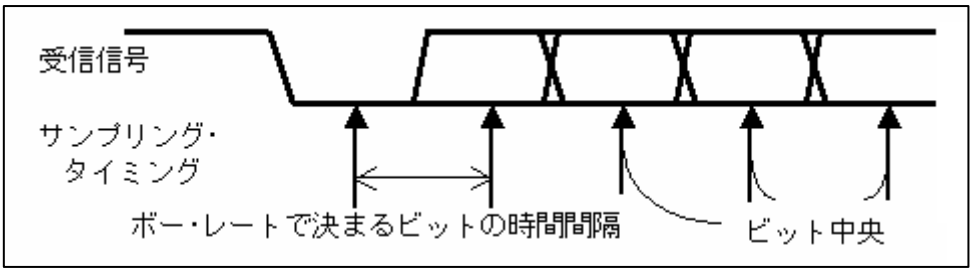


ハイ・レベルであった場合には、ノイズと判断して、(1)に戻ります。



(3)データのサンプリング

スタート・ビットの検出で、サンプリング・タイミングをスタート・ビットの立ち下りから 1/2 ビットずらすことにより、それ以後はボー・レートで決まるビット長ごとにサンプリングすることで、データ・ビットのデータの中央で受信信号のサンプリングを行うことができます。

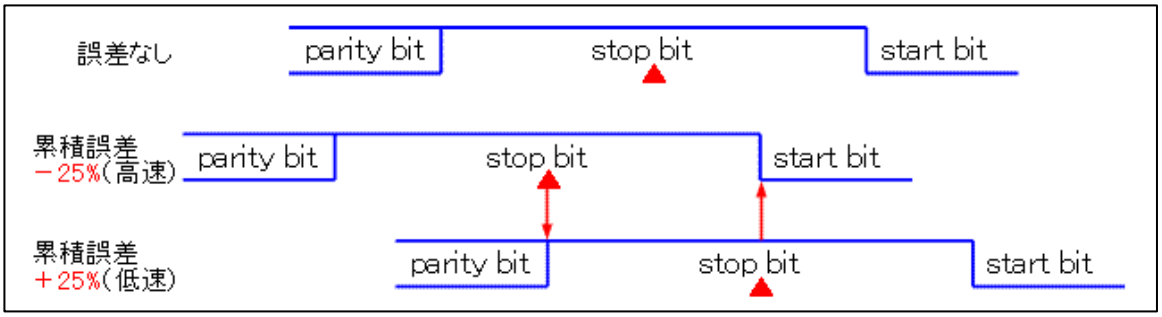


以上の流れで、受信信号からデータをサンプリングします。
このように、UART としてはビットの中央(実際にはサンプリング・クロックの関係で、正確に中央となりません)で受信信号を検出することで、伝送路での波形の歪みの影響をできるだけ避けるようになっています。

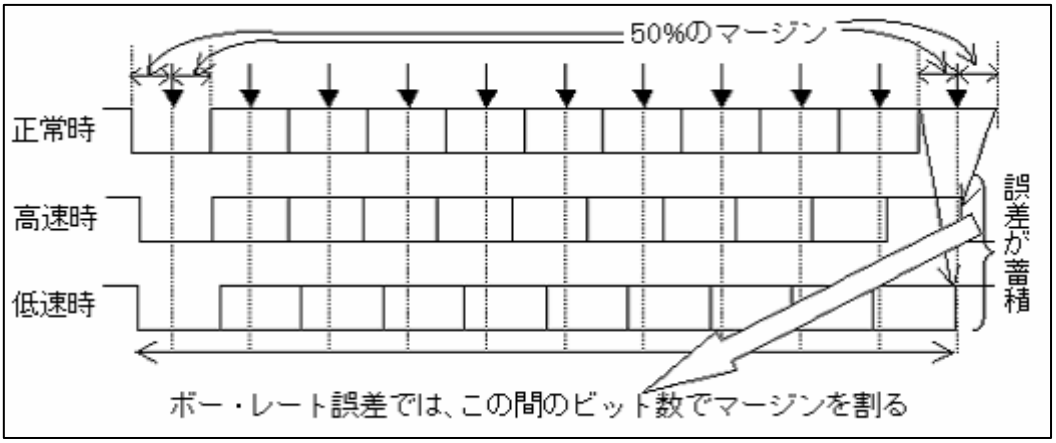
Q2. UART で受信する場合のボー・レートの許容誤差はどれくらいか、計算方法を教えてください。

A2. ボー・レートの許容誤差は、ビットの中央でサンプリングを開始して、ボー・レートの誤差によるサンプリング位置のずれが蓄積されていった場合に、ストップ・ビットの位置でどの程度の誤差になるかで判断します。

★ 送信側と受信側が+/−逆方向にずれている場合、1フレームの累積誤差の理論上の最悪値は、次図のように±25%です。つまり、相対的には50%が許容値となります。



理想的な場合(スタート・ビットの検出遅れがなく、サンプリングでのセットアップ時間やホールド時間が必要ない場合)には、スタート・ビット位置でのマージンは 50%(中央なので、前後に 50%ずれても問題なし)です。
これをストップ・ビットまでのビット数で割ることで、ボー・レート許容誤差が求められます。



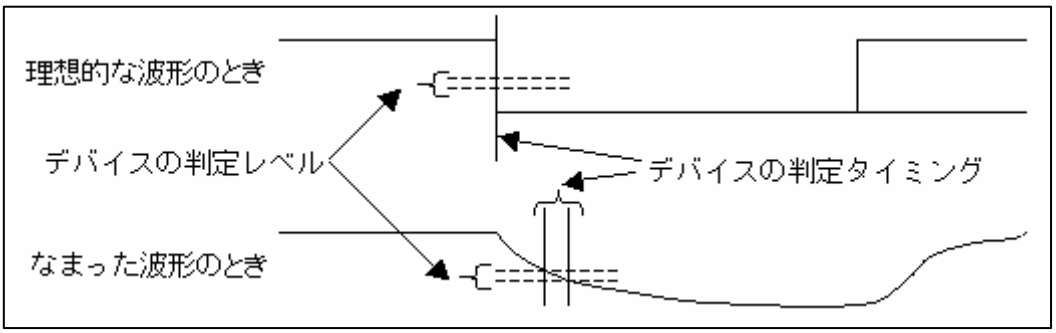
実際のケースでは、サンプリング周期やデータのサンプリングでの、セットアップ時間やホールド時間を加味する必要があり、50%が 40 数%になり、それをビット長で割ることで、ボー・レートの許容誤差が算出できます(どの程度マージンが少なくなるかは、個々のデバイスで異なります)。

このようにして得られた値は、送信側と受信側の相対的な許容誤差となります。

★ 図ではパリティ・ビットを含み、スタート・ビットの開始点からストップ・ビットのサンプリング点までが 11.5 ビットとなっています。この場合、40 数%の半分(1 局分)の 1/11.5、つまり 2%程度が許容ボー・レート誤差です。

なお、これらの値は、受信信号の波形が理想的な場合の値です。

実際の受信波形は、伝送路の影響で歪みます。このような場合には、デバイスがスタート・ビットを判定するレベル付近での信号の変化が緩やかになり、しかもノイズの影響も受けやすくなり、判定タイミングがばらつきます。このため、スタート・ビット位置でのマージンの 40 数%の値がさらに小さくなってしまい、その分ボー・レートの許容誤差は少なくなります。したがって、あまり許容誤差を大きく見積もらないようにしないと、誤動作の原因となります。



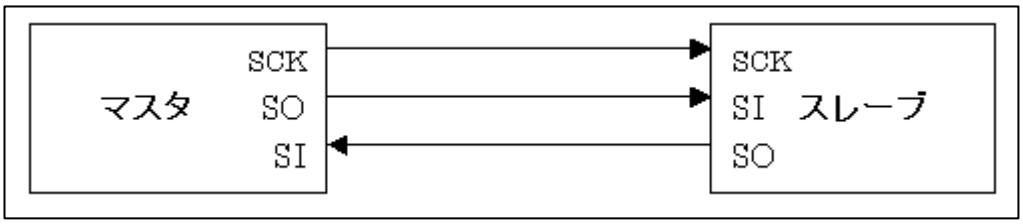
3 線式シリアル通信の基本動作 [ほとんどのマイコン共通]

Q1. 3 線式シリアル通信の基本動作について教えてください。

A1. 3 線式シリアル通信は、送信データ(SO)、受信データ(SI)と転送クロック(SCK)の3本の信号線を用いて通信を行う方式です。

この通信方式では、通信を制御する方をマスタと呼び、マスタは SCK 信号を出力して通信を制御します。

反対側をスレーブと呼び、こちらはマスタからの SCK 信号を受けて、データを送信/受信します。



送信と受信を同時に行えますので、効率的な転送を行うことができます。

また、調歩同期通信のような通信レートよりも高い周波数のクロックでのサンプリングが必要ありませんので、より高速の通信が可能です。

1.動作の概要

通常、通信を行っていない場合の SCK 信号は、H レベルになっています。

マスタは通信を開始すると、SCK を立ち下げます。

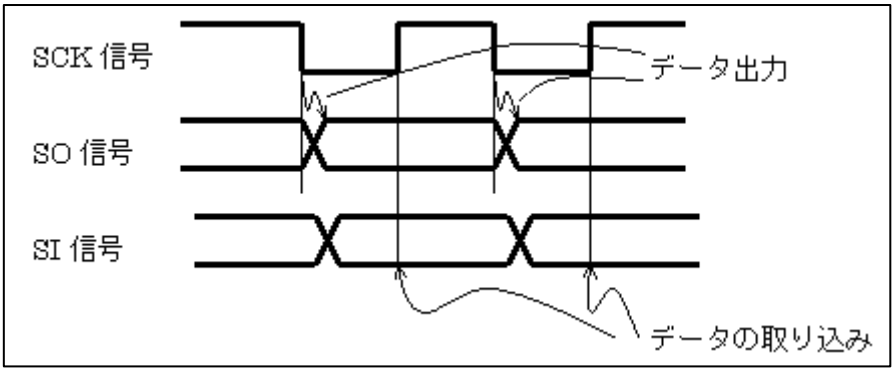
SCK 信号の立ち下がりで、マスタとスレーブは、それぞれの SO 端子に送信データを出力します。

出力されたデータは、相手の SI 端子に接続されていますので、マスタが SCK を立ち上げたときに取り込まれます。

この動作を 8 回繰り返すことで、1 バイトのデータを転送できます。

なお、これらの値は、受信信号の波形が理想的な場合の値です。

実際の受信波形は、伝送路の影響で歪みます。このような場合には、デバイスがスタート・ビットを判定するレベル付近での信号の変化が緩やかになり、しかもノイズの影響も受けやすくなり、判定タイミングがばらつきます。このため、スタート・ビット位置でのマージンの 40 数%の値がさらに小さくなってしまい、その分ボー・レートの許容誤差は少なくなります。したがって、あまり許容誤差を大きく見積もらないようにしないと、誤動作の原因となります。



この方式では、データだけでなくクロックもやりとりしますので、調歩同期通信のようなスタート・ビットやストップ・ビットを用いた制御は必要ありません。

また、データは必ず SCK 信号に同期して取り込みますので、送信側と受信側で転送レートが食い違って正常に受信できないといったことは発生しません。

このように、SCK の立ち下がりから全体の動作がスタートしますので、SCK が H レベルの期間に転送結果の処理を行い、次の転送の準備を完了する必要があります。

以上のような動作が基本ですが、一部のデバイスでは SCK の論理を変更できる(通信を行っていないときのレベルや送受信のエッジを変更できる)機能が追加されたり、外部からシリアルの動作を禁止させることができるものがあります。
また、データ長は 8 ビットが基本ですが、一部のデバイスでは 16 ビットまでの可変長のデータを通信できるもの、複数バイトのデータを転送する機能をもったものもあります。

2.使用する上での注意点

3 線式シリアル通信では、マスタが SCK 信号を出力する(L レベルに立ち下げる)前に、スレーブは送受信の準備を完了(スレーブがデータを送信したい場合には送信データを書き込み、受信の場合には前のデータの読み出しを完了)しておく必要があります。このためには、別途ハンドシェークのための信号を準備しておき、マスタは割り込みやポーリング等の手段により、スレーブ側のステータス(準備完了)を検出できるようにしておいてください。この順序が守られないと、正常なデータ転送はできません。

1.1.11 タイマ／カウンタ

タイマ／カウンタの基礎

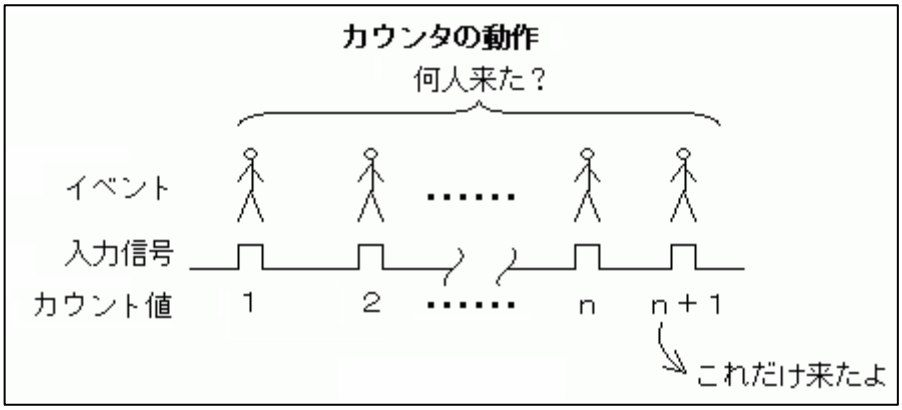
Q1. マイコンに内蔵されているタイマ／カウンタの動作について教えてください。

A1. タイマ／カウンタについての基礎的な内容を説明します。

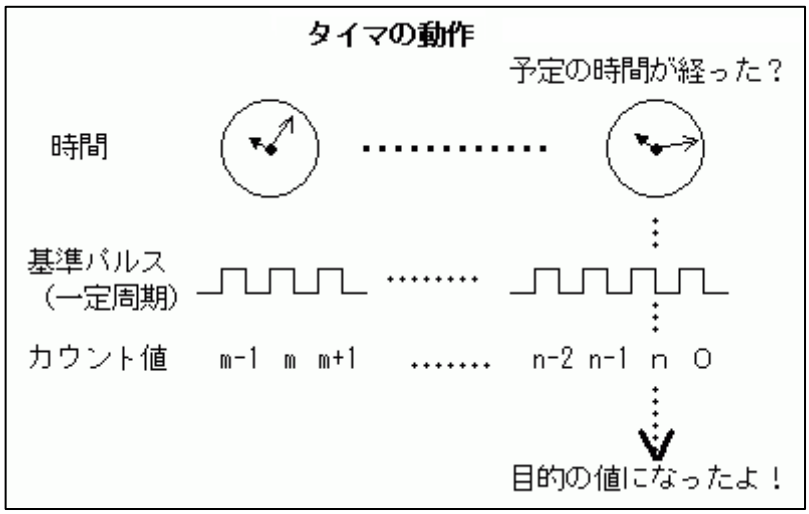
(1) はじめに

最近のほとんどのマイコンには、タイマ／カウンタが内蔵されています。
中にはインバータ制御用のように特定の目的専用のものもありますが、ここでは一般的なタイマ／カウンタについて説明します。
タイマ／カウンタは"タイマ"と"(イベント)カウンタ"の2つの機能を結合した名前となっています。
基本的に、タイマとカウンタを実現するための回路は同じもので、入力された信号をカウント(計数)するものです。タイマとカウンタの違いは、簡単には次のように表せます。

カウンタ ： 任意の間隔で入力された信号(イベント)の数をカウントし、個数を示す



タイマ ： 一定間隔で入力される信号をカウントすることで、時間の経過を示す



つまり、使用目的と入力される信号の違いとも言えます。
ほとんどの場合に、入力クロックとして内部のシステム・クロックを分周したものと外部からの信号の中から選択できるようになっていて、タイマとしてもカウンタとしても使えるようになっています。

(2) カウント動作による違い

タイマ／カウンタには、カウント動作で大きく分けると、次の 2 種類があります。

・単純にカウント動作だけを行うフリーラン・カウンタ

カウント動作だけを行います。場合によってはカウント動作を止めることができないものもあります。

・途中で停止したり、カウント周期を任意に設定してカウントできるカウンタ

ある値までカウントしたら信号を出したり、カウント値をクリアしたり、止めたりすることができます。

フリーラン・カウンタは回路が簡単なので、一部の安価なマイコンで採用されています。
また、時計用タイマは止めることは可能ですが、通常の動作では停止する必要がない(止めると時間が狂ってしまう)ので、動作的にはフリーラン・カウントとなります。
これに対して、マイコンに通常搭載されているタイマは途中で停止したり、任意のタイミングでクリアしたりといった動作が可能なので、いろいろと便利な使い方が考えられます。

(3) カウントへの付加機能

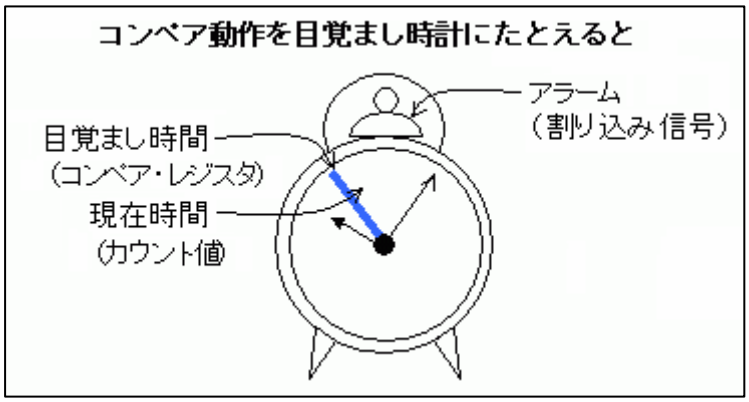
タイマ／カウンタでカウント動作するだけでは、使う場面が限られてきます。
タイマ／カウンタの機能を強化するために、タイマ／カウンタには1つか2つのレジスタが付加されています。
付加されたレジスタは、主に次のようなものです。

・コンペア・レジスタ
・キャプチャ・レジスタ

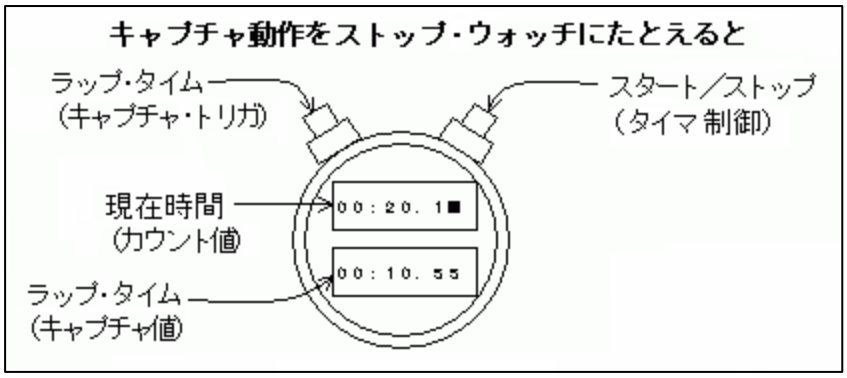
これらは、別のレジスタとして独立していることもありますが、通常は1つのレジスタで両方の機能を切り替えて使用することができるようになっています。
コンペア・レジスタは、タイマ／カウンタのカウント値と比較するための値を設定するレジスタです。
このレジスタを使用することで、設定した値とカウント値が一致したら、次のような動作を行います。

・割り込みを発生する
・出力を変化させる
・タイマをクリアする

この動作を利用すると、タイマ／カウンタのクロックの精度で、任意の時間間隔で割り込みを発生させたり、信号を変化させることが可能になります。



また、キャプチャ・レジスタは外部から入力された信号により、そのときのタイマ／カウンタのカウント値を取り込むために使用します。
この動作はハードウエアにより実現されるので、ソフトウエアによりタイマ／カウンタの値を読むよりも精度の高い値が得られます。



ソフトウエアで同じような動作をさせる場合、外部から入力された信号で割り込みを発生させて、割り込み処理プログラムの中でタイマ／カウンタの値を読み出すことになります。割り込みが発生して、実際に割り込みが受け付けられる時間は一般に変動します。さらに割り込み処理プログラムの中でタイマ／カウンタの値を読み出すまでに時間が必要です。そのため、外部から信号が入力されたタイミングの値からずれた値になることがあります。

(4) タイマ／カウンタの用途(動作)

(a) タイマ／カウンタのおもな用途(動作)

測定機能と出力機能があり、おもな用途(動作)として、次のようなものがあります。

①測定機能

・外部イベント・カウント
・パルス幅測定、時間間隔測定

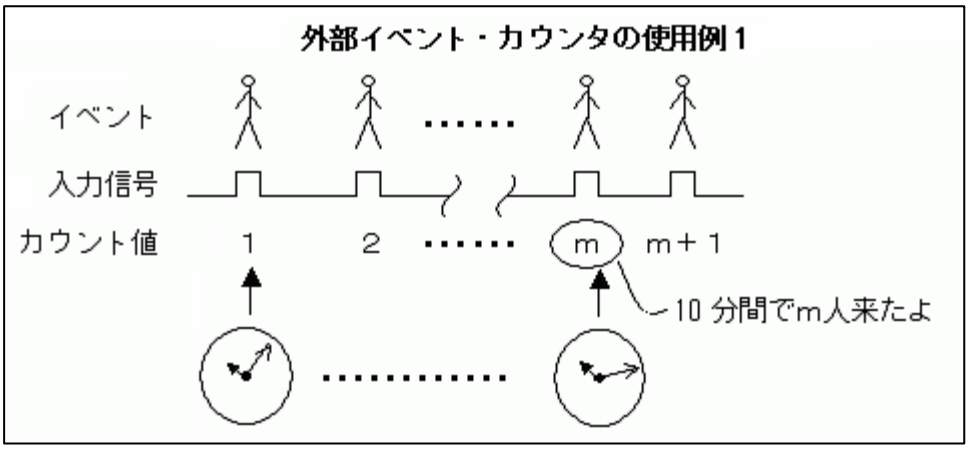
②出力機能

・インターバル・タイマ
・方形波出力
・パルス出力
・PWM 出力
・ワンショット・パルス出力
・キャリア・ジェネレーション

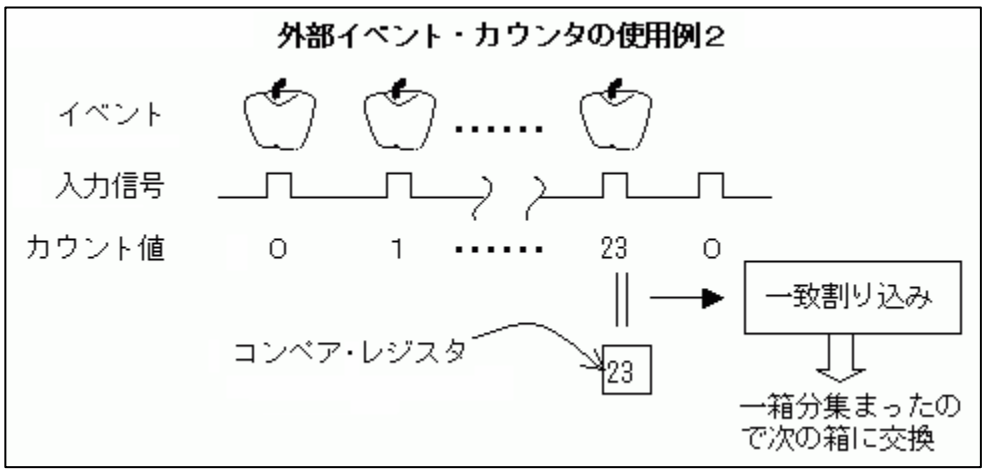
(b) 個々の機能の概要

(i) 外部イベント・カウント

外部からの信号(イベント)の数をカウントするする動作で、最も基本的な動作のひとつです。
信号がいくつ来たかをカウントしていて、プログラムでカウント値を定期的に読むような使い方が考えられます。
たとえば、次の図のように、10 分間に来た人を数える場合に使います。

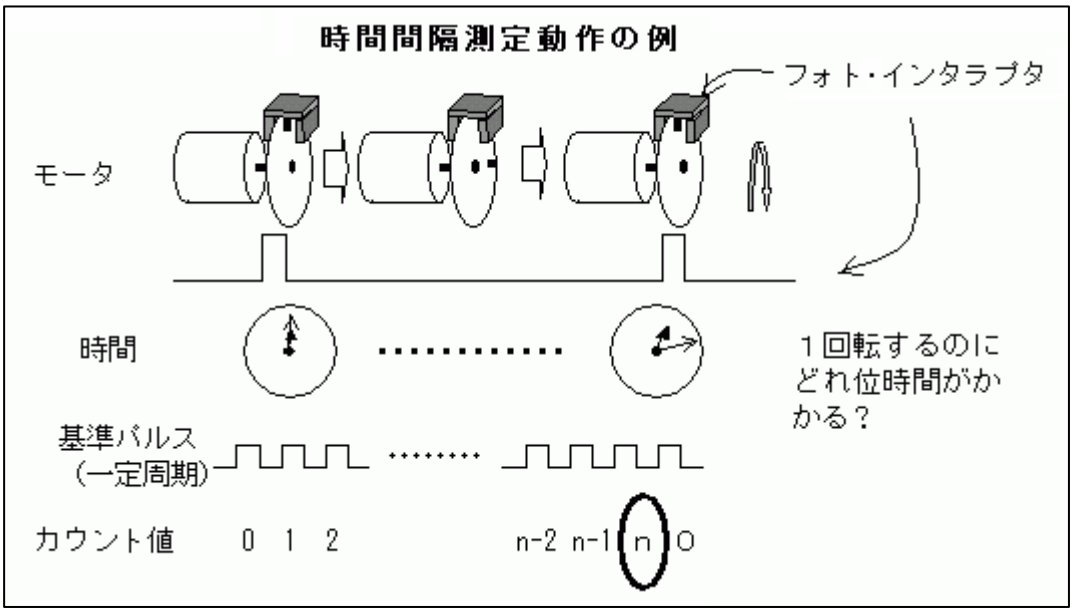


これ以外の使い方としては、コンペア・レジスタと組み合わせて、カウントした外部からの信号(イベント)が指定した数になったら、通知する(割り込みを発生する)動作を行わせることもあります。
たとえば、1 箱に 24 個のリンゴを入れるときに、24 個カウントしたら箱を交換するために使用します。
次の例のように、コンペア・レジスタに 24-1 を設定しておくと、24(0～23)個カウントして割り込みを発生します。割り込みが発生したら、次の箱に交換するような動作となります。



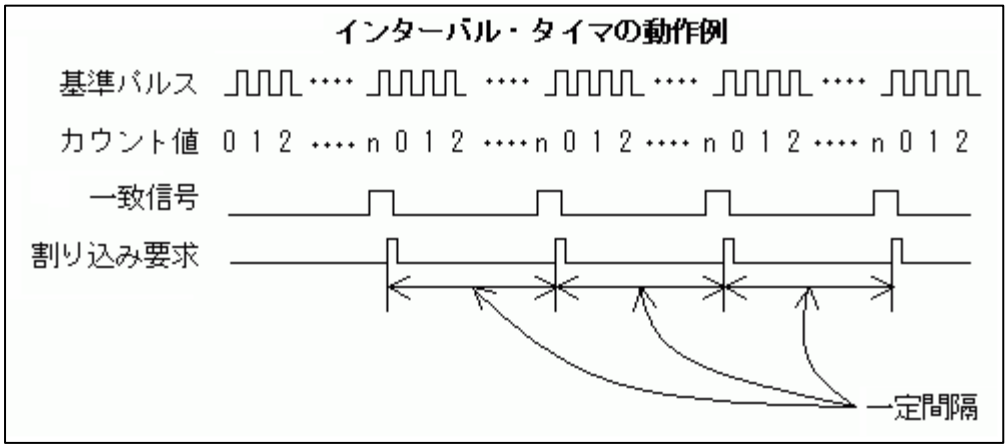
(ii) パルス幅測定、時間間隔測定

外部からの信号(イベント)の間隔を測定する機能で、キャプチャ・レジスタの基本的な使い方です。
下図ではモータの回転数を検出する場合の例を示します。
モータの軸にスロットの空いた円盤を取り付け、スロットをフォト・インタラプタで検出します。
フォト・インタラプタの出力を外部からの信号(イベント)として、キャプチャのトリガ入力にします。
トリガが入力されると、カウンタの値をキャプチャ・レジスタに取り込み、カウンタをクリアします。
キャプチャ・レジスタの値を読むことで、回転の周期を知ることができ、回転数を得ることができます。



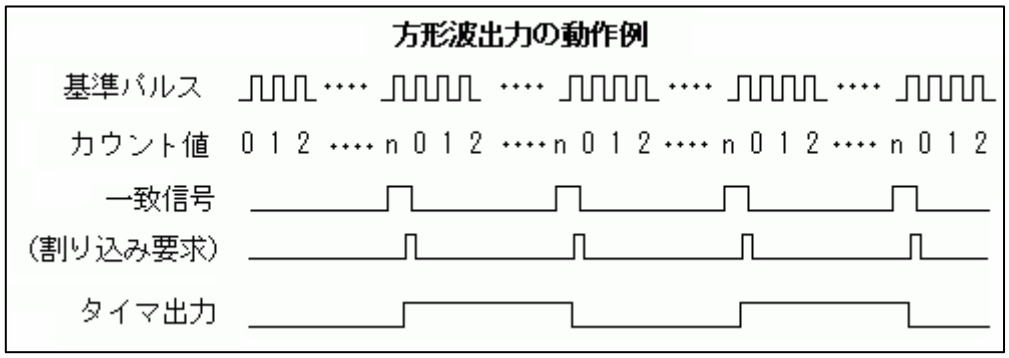
(iii) インターバル・タイマ

一定間隔(インターバル)で割り込みを発生させる機能で、タイマとしてもっとも基本的な動作です。
この動作では、コンペア・レジスタを使用します。コンペア・レジスタに設定した値とタイマ／カウンタの値が一致すると割り込みを発生し、タイマ／カウンタをクリアして再度 0 からカウントを行います。
この動作を繰り返すことで、一定間隔で割り込みを発生させることができます。



(iv) 方形波出力

方形波出力の基本的な動作は、インターバル・タイマと同じです。
違いは、波形そのものを出力することです。
下図の例のように、コンペア・レジスタとタイマ／カウンタが一致するたびに割り込み要求が発生し、タイマ出力が反転動作を繰り返します。



(v) パルス出力

パルスを出力する機能ですが、方形波出力と異なり、周期だけでなくデューティも変更できます。

この機能では、コンペア・レジスタを 2 つ使用し、1 つは周期を指定するために、もう 1 つはデューティを指定します。

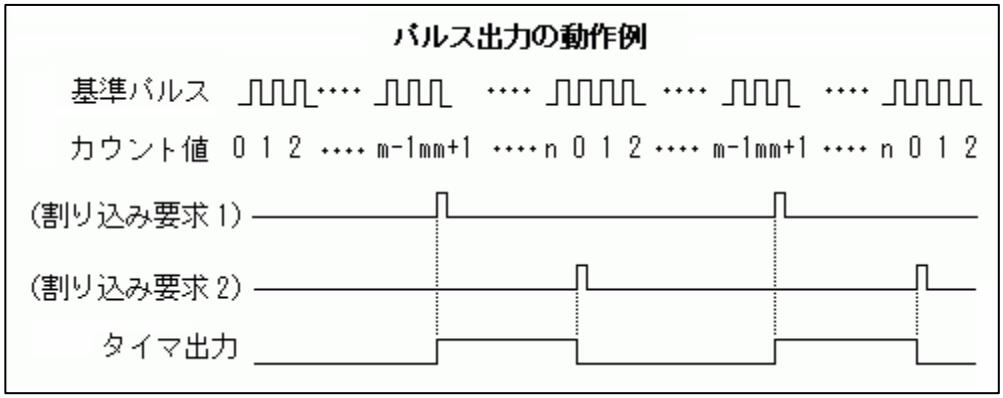
2 つのコンペア・レジスタは同等ではなく、周期を指定するために用いるコンペア・レジスタでは、一致によりタイマ／カウンタをクリアできますが、デューティを指定するためのコンペア・レジスタとの一致では、出力が変化するだけで、タイマ／カウンタをクリアしません。

また、各コンペア・レジスタとタイマ／カウンタとの一致で、割り込み要求を発生します。

パルス出力の動作例を次図に示します。

この機能を使用すると、次に説明する PWM 出力と同じような出力波形を発生させることができます。

なお、パルス出力では、デューティを変更するために、コンペア・レジスタの値を変更するタイミングが問題になることがあります。タイミングを誤ると、1 周期の中でデューティ設定用のコンペア・レジスタとの一致が 2 回発生したり、1 回も発生しなかったりして、タイマ出力が反転してしまうことがあります。



(vi) PWM 出力

パルス出力と同じような波形を発生させますが、基本的にデューティを変更することが目的です。

PWM 出力の周期は、通常固定されていることが多く、その場合にはタイマ／カウンタのビット数で決まる時間となります。

このとき、デューティを指定するために、コンペア・レジスタを使用します。

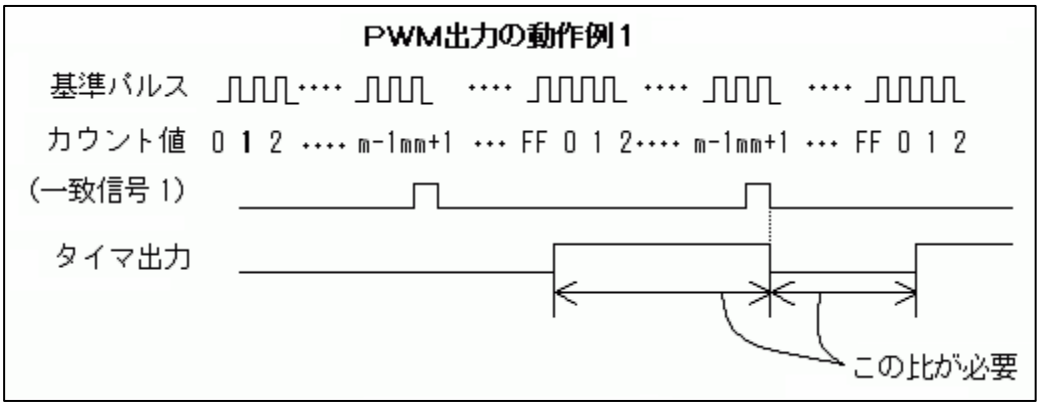
PWM 出力はパルス出力と異なり、目的が明確なので、デューティ変更のためのコンペア・レジスタの書き換えタイミングに制約はありません。

動作タイミングを次図に示します。この動作タイミングでは最初の起動時を示しています。

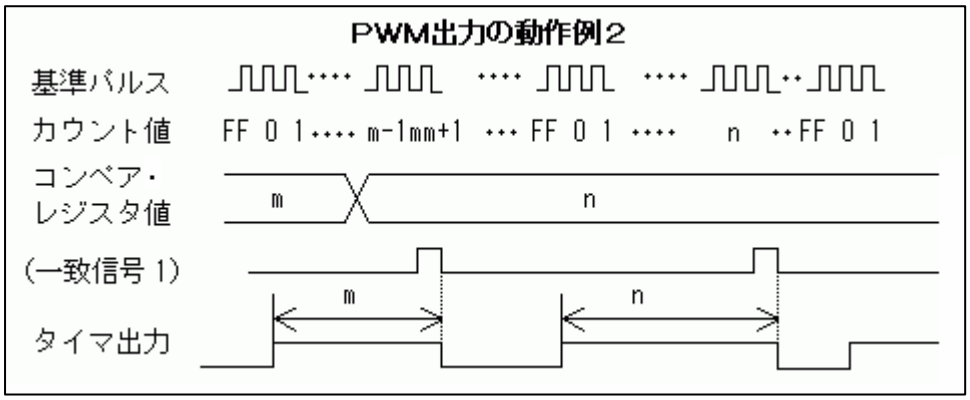
起動すると、最初の 1 周期間、出力はインアクティブ(図ではロウ・レベル)となります。

タイマ・カウンタがオーバフローすると出力がアクティブとなり、コンペア・レジスタとの一致でインアクティブになります。

以降は、この動作を繰り返すことで、PWM 信号を出力します。



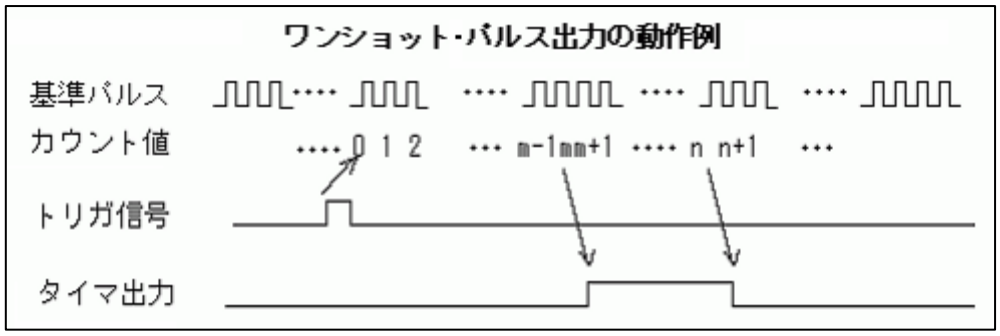
コンペア・レジスタの値を書き換えて、デューティを変更した場合の動作例を次の図に示します。
周期の途中で m から n に書き換えていますが、その結果が動作に反映されるのは、次のオーバフロー以降になります。



タイマ／カウンタによっては、周期も変更できるものもありますが、その場合は 1 周期の完了がタイマ／カウンタのオーバフローではなく、周期設定用のコンペア・レジスタとの一致になるだけです。

(vii) ワンショット・パルス出力

これまでの出力機能が、同じ動作を連続して繰り返すものであったのに対して、この機能は 1 パルスだけを出力するものです。
この機能では、2 つのコンペア・レジスタを使用します。
1 つはトリガがかかってからパルスを出力するまでの時間を指定し、もう 1 つはパルスの幅を指定します。
動作の概要を次のタイミング図に示します。
外部からトリガ信号が入力されると、タイマ・カウンタはクリアされ、0 からカウント動作を開始します。
タイマ／カウンタの値が、出力タイミングを指定するコンペア・レジスタの値(m)に一致すると、タイマ出力にパルスが出力され、次に幅を指定するコンペア・レジスタの値(n)に一致すると、パルスの出力を終了します。



(viii) キャリア・ジェネレーション

赤外線リモコン用の信号を生成するときに使用する機能で、2 つのタイマ／カウンタを組み合わせて使用します。

タイマ／カウンタそのものの動作よりは、その応用となりますので、他の機能とは性格が異なります。

赤外線リモコンでは、発光時には赤外線をあるキャリア周波数でオン／オフしています。そのため、1 つのタイマ／カウンタでは、このキャリア周波数に合わせた信号を発生します(パルス出力動作)。

もう1つのタイマ／カウンタでは、赤外線リモコン信号そのものの信号の長さ(期間)をカウントします(インターバル・タイマ動作)。

インターバル・タイマからの設定時間を経過したことを示す出力信号により、リモコンで出力したい信号をサンプリングし、1 ならキャリアを出力し、0 ならばキャリアを出力しないような動作になります。

この動作を次図で示します。キャリア信号は、すでに別のタイマ・カウンタで発生しているものとします。

インターバル・タイマ動作のコンペア・レジスタの値は、一致の割り込みで書き換えるものとし、k(オフ時間)、m(オン時間)、n(オフ時間)を順に設定していくものとします。

①k との一致で、出力データ値(1)がサンプリングされます。

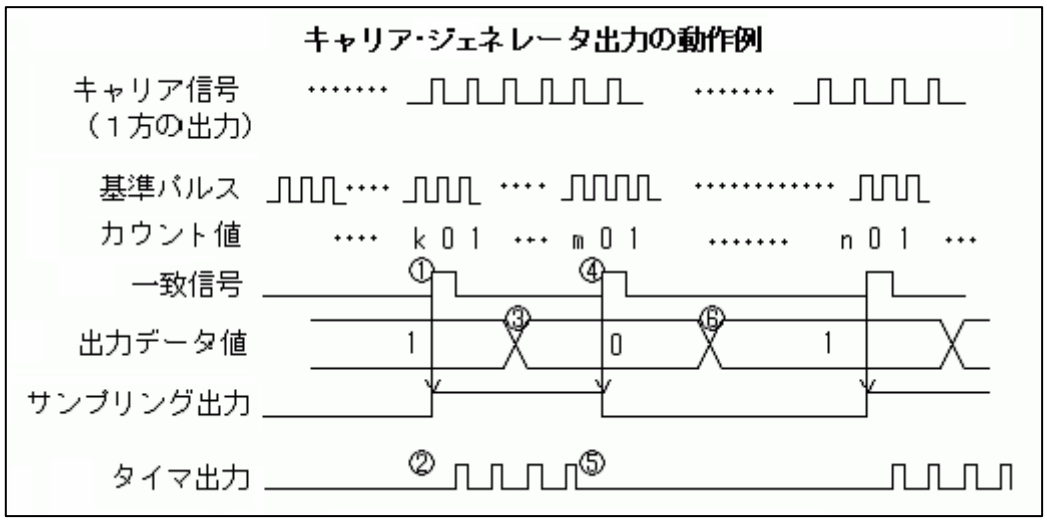
②タイマ出力から、キャリア信号が出力され始めます。

③一致による割り込み処理で、コンペア・レジスタに m、出力データに 0 を書きます。

④m との一致で、出力データ値(0)がサンプリングされ、⑤キャリア出力が停止します。

⑥一致割り込みで、コンペア・レジスタと出力データ値を更新します。

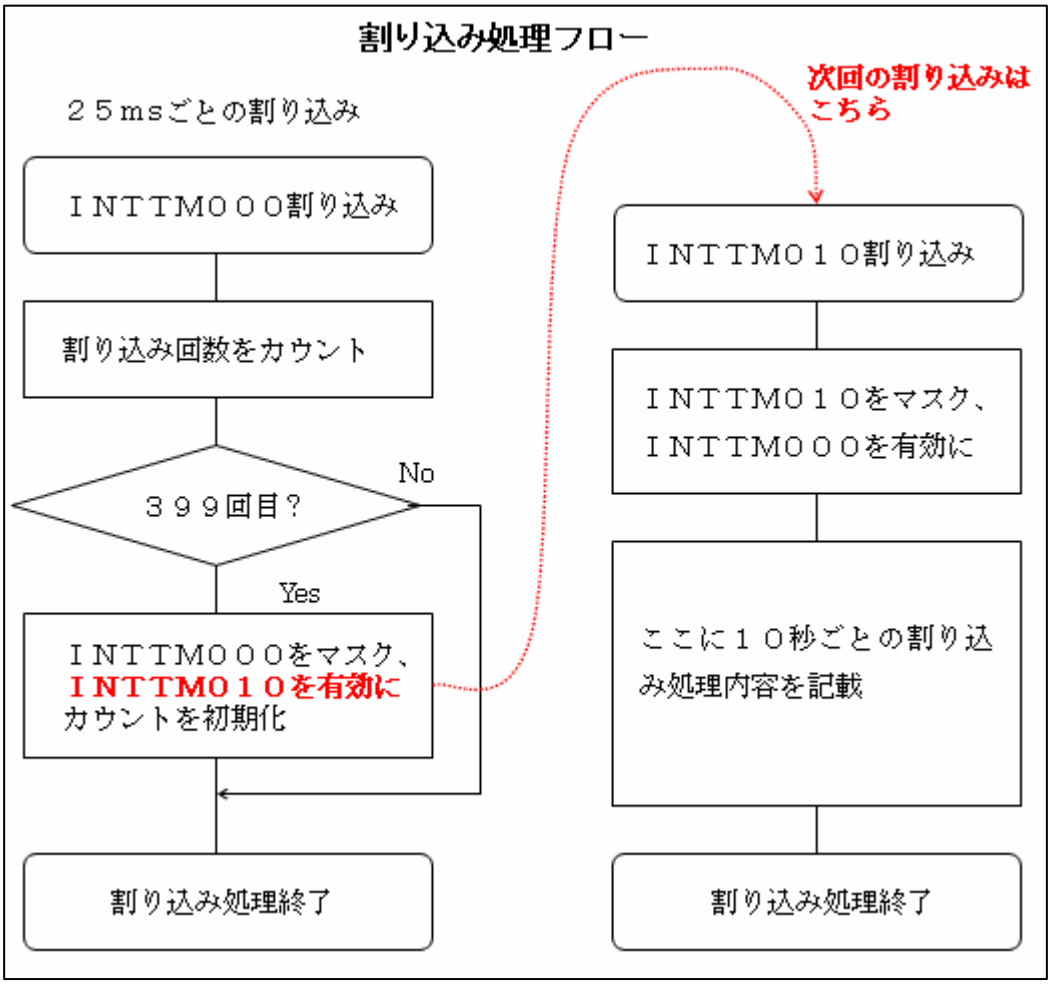
以上の動作を繰り返します。



16 ビット・タイマ／カウンタ 00 の使い方(1) [78K0、小ピン・マイコン共通]

Q1. 78K0S/KA1+の 16 ビットのタイマ 00 で、約 10 秒の間隔でインターバル・タイマ割り込みを実現したいのですが、タイマの設定だけではそんなに長くできません。ソフトウエアでのカウントは精度が悪いので、できるだけハードウエアでカウントさせたいです。

A1. 10MHz の周辺機能動作クロック(f_{XP})で、10 秒の時間間隔を発生させることを考えます。できるだけ誤差を小さくするために、f_{XP}を 1/4 分周にした 0.4μs をカウントします。そのうえで、CR000 を用いて、タイマ 00 で 25ms をカウントさせます。このままでは 25ms ごとの割り込みにしかなりませんので、割り込み処理の中で割り込み回数をカウントさせ、400 回目に INTTM010 が有効になるようにします。つまり、INTTM010 が 10 秒間隔で発生する割り込みとなります。この使い方では、時間はすべてタイマ 00 でカウントしますので、ソフトウエアの介在による時間の変動はありません(通常の割り込み受け付け時間の変動だけになります)。
なお、CR000 と CR010 には、同じ値(62500－1)を設定しておきます。



上記のフローで、各割り込みでの処理は次のようになります。

(1) INTTM000(25ms ごとの割り込み)

399 回分の割り込みをソフトウエアでカウントします。399 回のカウントが完了したら、自分自身の割り込みをマスクし、保留されている INTTM010 の割り込み要求をクリアしてからマスクを解除します。これで、400 回目には INTTM010 のベクタ割り込みが発生することになります。次の回の準備のため、ソフトウエアのカウンタを初期化します。

(2) INTTM010(10 秒ごとの割り込み)

ベクタ割り込みが発生したら、自分自身の割り込みをマスクします。保留されているINTTM000の割り込み要求をクリアし、マスクを解除します。その後に必要な処理を行ってください。割り込み制御部分の処理の分だけは実際の割り込みサービスは遅くなりますが、この遅延は一定ですから問題にはなりません。
また、この割り込みでの処理はタイマ00のカウント周期の2周期分の50ms未満で完了する必要があります。そうしないと、次の周期のカウントが正常にできません(割り込み要求フラグでは1回分の割り込みしか保留できないので、割り込み保留中にさらに同じ割り込み要求が発生しても1回分の割り込みしか認識できないためです)。

具体的な設定や処理は、次のようになります。
割り込みベクタ･テーブルの0EH番地と10H番地に、それぞれINTTM000とINTTM010の割り込み処理を行うアドレスをDW擬似命令でセットしておきます。

```
INTTMVECT       CSEG    AT      0EH
        DW      TM00INT         ;0E:INTTM000
        DW      TM01INT         ;10:INTTM010
```

また、25msごとのカウントを行うために、16ビットの変数を2つ準備しておきます。

```
        DSEG    saddrp
SOFTTIME:
        DS      2               ; ソフトウエア・タイマのカウント用
INITTIME:
        DS      2               ; 初期値用
```

この変数の初期化を含めて、タイマの初期化部分は次のようになります。

```
        TBASE   SET     40              ; 1秒当たりの割り込み回数
        ITIME   SET     10              ; 割り込み発生時間(秒)
        COUNT   EQU     TBASE*ITIME-1   ; 必要なカウント回数
        MOVW    AX,#COUNT
        MOVW    SOFTTIME,AX             ; ソフトウエア・タイマ初期化
        MOVW    INITTIME,AX             ; ソフトタイマ初期値
        MOV     CRC00,#00000000b        ; 両方をコンペアで使用
        MOV     PRM00,#00000001b        ; カウント・クロックは fxp/4
        MOVW    CR000,#62499            ; 62500 カウント
        MOVW    CR010,#62499            ; 62500 カウント
;
;   ここまでが初期設定、以下で実際にタイマ起動
;
        MOV     TMC00,#00001100b        ; タイマ起動
        CLR1    TMIF000                 ; 割り込み要求をクリアしておく
        CLR1    TMMK000                 ; INTTM000 のマスクを解除
        SET1    TMMK010                 ; INTTM010 はマスク
        EI                              ; 割り込み許可
```

割り込み処理部は次のようになります。INTTM000は、25msごとの処理にしか使用しない場合は、このような処理となりますが、他の処理もしたい場合は、割り込み処理の最初に追加してください。

```
;
;   INTTM000 割り込み処理部
;
TM00INT:
        PUSH    AX                      ; 使用するレジスタをセーブ
        MOVW    AX, SOFTTIME            ; ソフト・カウンタ読み出し
        DECW    AX                      ; カウント･ダウン
        MOVW    SOFTTIME,AX             ; 書き戻し
        OR      A,X                     ; カウント結果のチェック
        BNZ     $TM00EXIT               ; 完了しなければ抜ける
        MOVW    AX,INITTIME             ; 初期値の読み出し
        MOVW    SOFTTIME,AX             ; ソフトウエア・タイマ初期化
        SET1    TMMK000                 ; 自身の割り込みをマスク
        CLR1    TMIF010                 ; INTTM010 をクリア
        CLR1    TMMK010                 ; INTTM010 マスクを解除
TM00EXIT:
        POP     AX
        RETI                            ; 割り込みから復帰
```

10 秒ごとに発生する INTTM010 の割り込み処理は、次のようになります。この中で**赤字の部分**に、実際に処理させたい内容を記述します。

```
;
;   INTTM010 割り込み処理部
;
TM01INT:
        SET1    TMMK010                 ; 自身の割り込みをマスク
        CLR1    TMIF000                 ; INTTM000 をクリア
        CLR1    TMMK000                 ; INTTM000 マスクを解除

        PUSH    AX                      ; 使用するレジスタをセーブ
        PUSH    HL
;
;   ここに必要な処理を記述
;
        POP     HL                      ; セーブしたレジスタの復帰
        POP     AX
        RETI                            ; 割り込みから復帰
```

これをＣ言語で記述する場合は、次の宣言を最初に行い、値の定義とINTTM000 割り込みと関数 TM00INT、INTTM010 割り込みと関数 TM01INT を関連付けしておきます。

```
#define TBASE 40
#define ITIME 10
#pragma SFR
#pragma EI
#pragma DI
#pragma NOP
#pragma interrupt INTTM000 TM00INT
#pragma interrupt INTTM010 TM01INT
__interrupt void        TM00INT (void);
__interrupt void        TM01INT (void);
sreg static unsigned int SOFTTIME, INITTIME;
```

初期化部分は、次のようになります。

```
        CRC00 = 0b00000000;     /* CR000 をコンペアで使用          */
        PRM00 = 0b00000001;     /* カウント・クロックは fxp/4      */
        CR000 = 62499;          /* 62500 カウント                  */
        CR010 = 62499;          /* 62500 カウント                  */
/*
        ここまでが初期設定、以下で実際にタイマ起動
                                                                  */
        INITTIME = TBASE*ITIME-1;
        SOFTTIME = TBASE*ITIME-1;
        TMC00 = 0b00001100;     /* タイマ起動                      */
        TMIF000 = 0;            /* 割り込み要求をクリアしておく    */
        TMMK000 = 0;            /* INTTM000 マスクを解除           */
        TMMK010 = 1;            /* INTTM010 をマスク               */
        EI();                   /* 割り込み許可                    */
```

割り込み処理部分は、次のようになります。

```
__interrupt     void    TM00INT (void){
        SOFTTIME--;
        if(SOFTTIME == 0){
                SOFTTIME = INITTIME;
                TMMK000 = 1;            /* 自身の割り込みをマスク     */
                TMIF010 = 0;            /* INTTM010 をクリア          */
                TMMK010 = 0;            /* INTTM010 マスクを解除      */
        }
}

__interrupt     void    TM01INT (void){
        TMMK010 = 1;            /* 自身の割り込みをマスク         */
        TMIF000 = 0;            /* INTTM000 をクリア              */
        TMMK000 = 0;            /* INTTM000 マスクを解除          */
/*
   この部分に、実際に行いたい処理を記述
*/
}
```

周期は通常は固定されています。上記のプログラムでは、所定の周期が INTTM000 の割り込みの何回分かを指定するために、作業用として**変数 SOFTTIME** を使用しています。さらに、その初期値を**変数 INITTIME** に入れてあります。そのため、変数 INITTIME の値を書き換えれば、25ms 刻みで次回の周期を変更できます。

このプログラムを 78K0/Kx2 で動作させる場合は、次の点にご注意ください。

- 周辺機能動作クロック(f_{XP})が最大 20MHzなので、使用するf_{XP}に合わせてタイマ 00 でカウントさせる基本となる周期(25ms)を見直す。
- アセンブラ記述の場合は、INTTM000 や INTTN010 の割り込みベクタのアドレスを変更する。

16 ビット・タイマ／カウンタ 00 の使い方(2) [78K0、小ピン・マイコン共通]

Q1. 78K0/KB2 を 8MHz で動作させ、入力されるパルスの幅(280,000μs や 70,000μs)の測定はできますか。

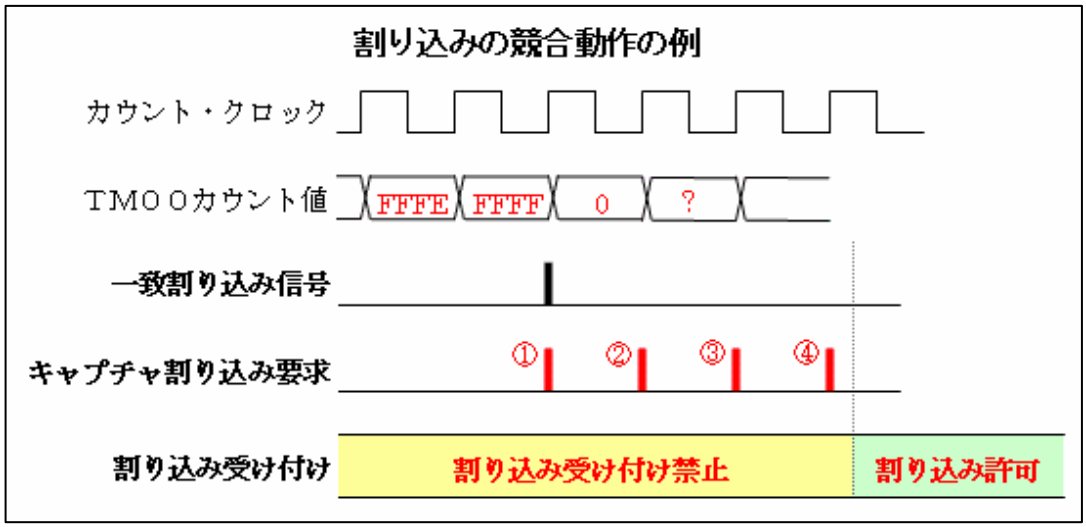
A1. 問題は測定精度ですが、16 ビット・タイマ／カウンタ(以下タイマ 00 と呼ぶ)を 8MHz のクロックを 1/256 に分周した 32μs のカウント・クロックで動作させると、2,097,152μs まで測定可能です。32μs の精度でよければ、タイマ 00 をキャプチャ動作で使用すれば測定できます。

Q2. Q1 の条件で1μs 程度の測定精度が必要な場合は、どうすればいいですか。

A2. 1μs 程度の精度にするには、8MHz のクロックを 1/4 に分周した 0.5μs のカウント・クロックに設定する必要があります。ところが、この場合には約 32ms までしか(OVF フラグを使用しても約 65ms)カウントできません。ここでは、さらに長いパルスの測定への対応方法を考えます。タイマ 00 がオーバフローしたら、上位のけたをカウントアップすれば、より長い時間の計測が可能になります。上位けたとして 8 ビットの変数を追加すれば、**8.3 秒**までの時間が計測可能です(カウント・クロックを 8MHz にしても、約 2 秒まで計測可能)。

しかし、タイマ 00 にはオーバフローを示すフラグはありますが、割り込み機能がありません。そこで、CR000 をコンペア・レジスタとして使用し、これをオーバフロー割り込み要求の代わりに使用します。つまり、CR000 レジスタに 0FFFFH を設定しておくと、タイマ 00 が FFFF→0000 になったときに割り込みが発生します。

ここで、CR000 レジスタの一致割り込み要求と CR010 レジスタのキャプチャ割り込み要求が、**同時に発生**することにご注意ください。多重割り込みを使わなければ、先に受け付けた割り込み要求が優先されます。ところが、ある割り込みを処理中に発生した割り込み要求は保留され、処理中の割り込みが完了したときにもっとも優先度が高いものが受け付けられます。このため、**割り込み処理の中で割り込みの発生順序を判断**する必要があります。割り込みがタイマ 00 の 1 周期分も保留されることはないとすると(そのような場合は元々正常な制御自体ができません)、次のような割り込み競合動作の例が考えられます。キャプチャの割り込みが①～割り込み許可になるまでのタイミングで発生した場合が競合となります(タイマ 00 を「TI000 端子の有効エッジでクリア＆スタート」モードに設定しておくと、①より前にキャプチャ割り込み要求が発生する場合は、タイマ 00 は FFFF にならないので、一致割り込みが発生しません)。



ここで、問題は①の場合だけです。この場合、上位のけたのカウントアップは不要です。しかし、②以降ではカウントアップが必要です。そこで、割り込みの優先度が高い CR000 レジスタの一致割り込み処理の中で、CR010 レジスタへのキャプチャ割り込み要求が発生しているかどうかを確認し、発生していなければカウントアップを行います。また、キャプチャ割り込み要求が発生している場合は、キャプチャ値を確認して、FFFF 以外の場合だけカウントアップを行います。これにより、上位のけたとキャプチャした値との、正しい組み合わせを得ることができます。

これを実際に C 言語で記述してみます。この場合には、最初に次の宣言をします。ここで、**変数 LENGTH** はキャプチャした値を取り込んでおくための 16 ビットの領域です。8 ビットの**変数 HIGHLENGTH** は、キャプチャした値に対する上位けたを取り込むみ、**HIGHCOUNT** は内部の作業用でカウントに用いる変数です。**変数 HIGHLENGTH** と**変数 LENGTH** を組み合わせて、24 ビットでキャプチャした値が得られます。

```
#pragma SFR
#pragma EI
#pragma DI
#pragma NOP
#pragma INTERRUPT INTTM000 TM00INT
#pragma INTERRUPT INTTM010 TM01INT
__interrupt void        TM00INT (void);
__interrupt void        TM01INT (void);
sreg static unsigned int LENGTH;
sreg static unsigned char HIGHLENGTH, HIGHCOUNT;
```

初期化部分は、初期設定部分と実際にタイマを起動する部分で、割り込み禁止で最初のエッジを待ちます。その後、割り込みを許可して測定を始めます。

```
        PM3.0 = 1;              /* TI000 兼用端子を入力に設定    */
        CRC00 = 0b00000100;     /* CR010 をキャプチャで使用      */
        PRM00 = 0b00110000;     /* 両エッジを指定                */
        CR000 = 0xFFFF;         /* CR000 はオーバフロー検出用    */
        TOC00 = 0b00000000;     /* タイマ出力は行わない          */
        MK0 = 0x0FF;            /* 最初は割り込みをマスク        */
        IF0 = 0x00;             /* 割り込み要求をクリア          */
/*
        ここまでが初期設定、以下で実際にタイマ起動
                                                              */
        HIGHCOUNT = 0;          /* 上位カウンタをクリア          */
        LENGTH = 0;             /* 結果の格納エリアをクリア      */
        TMC00 = 0b00001000;     /* タイマ起動                    */
        while(TMIF010==0){
                NOP();          /* 最初のキャプチャ完了を待つ    */
        }
        IF0 = 0;                /* 割り込み要求フラグをクリア    */
        TMMK000 = 0;            /* 割り込みマスクを解除          */
        TMMK010 = 0;            /* 割り込みマスクを解除          */

        EI();
```

INTTM000 では、タイマ 00 のオーバフローを監視しています。

```
__interrupt     void    TM00INT (void){
        unsigned int work;
        if (TMIF010){           /* キャプチャ割り込みが発生中    */
                work = CR010;   /* キャプチャ値を読み出し        */
                if (work != 0xFFFF){
                        HIGHCOUNT++;    /*同時でないならカウント*/
                }
        }
        else{
                        HIGHCOUNT++;    /* オーバフローだけならカウント*/
        }
}
```

INTTM010 で、実際にキャプチャした値と上位けたの値を読み出して、結果格納用の変数に書き込みます。
実際に使用する場合は、この部分（赤字）を書き換えてください。

```
__interrupt     void    TM01INT (void){
        LENGTH = CR010;         /* キャプチャした値を読み出し    */
        HIGHLENGTH = HIGHCOUNT; /* 上位桁を読み出し              */
        HIGHCOUNT = 0;          /* 上位桁をクリア                */
}
```

内蔵タイマ／カウンタの使い分け [小ピン・マイコン]

Q1. 78K0S/Kx1+に内蔵されているタイマは、どのように使い分ければいいですか。

A1. 78K0S/Kx1+に内蔵されているタイマの、おもな用途と動作モードは、次のように大きく分類できます。

主な用途	動作分類	タイマの機能	対応タイマ		
			TM00	TM80	TMH
時間計測	定周期タイマ	インターバル・タイマ	○	○	○
	計時処理	ワンショット・パルス出力	○	－	－
	パルス幅測定	パルス幅測定	○	－	－
パルス・カウント	パルス・カウント	外部イベント・カウンタ	○	－	－
パルス出力	連続パルス出力	方形波出力	○	－	○
		ＰＰＧ出力、ＰＷＭ出力	○	－	○
	単発パルス出力	ワンショット・パルス出力	○	－	－

タイマ機能の基礎的な内容については、「タイマ／カウンタの基礎的な内容[共通]」をご参照ください。
タイマの基本的な用途である**時間計測**には、次の 2 種類があります。

・ある一定の時間を待ち合わせる動作
・外部から入力されるパルスの（時間的な）長さを測る動作（**パルス幅測定**）

また、一定の時間を待ち合わせる動作には、単位となる時間を定周期タイマで計測（割り込みを発生）して、その割り込み回数で必要な時間を測定する場合（**定周期タイマ**）と、ある時点でタイマを起動して、そこからの時間を計測する場合（**計時処理**）が考えられます。これに使用できるタイマの機能は、「**インターバル・タイマ**」と「**ワンショット・パルス出力**」が考えられ、その機能をもったタイマが表の右側に示されています。
パルス・カウントは入力されたパルス数をカウントする動作で、16 ビットの TM00 しか対応していません。
また、**パルス出力**用途では、出力するパルスの種類に応じて、「**ワンショット・パルス出力**」、「**方形波出力**」、「**PPG 出力**」、「**PWM 出力**」などがあります。
なお、TM00 の PPG 出力と TMH の PWM 出力は似ていますが、デューティの変更は TMH の方が簡単です。違いは、78K デバイスの「PPG 出力と PWM 出力の違い」もご参照のうえ、使用目的に合わせて選択してください。

基本的に各タイマは、同時には 1 つの動作しかできないので、**タイマをどのように使用するか**を十分に吟味しておく必要があります。タイマの設定では、次の 2 点が大きな検討項目となります。

・使用するタイマの動作モード
・カウントするクロックの選択

また同じ機能でも、TM00 は 16 ビット長のカウントが可能で、他は 8 ビット長でのカウントなので、その使い分けも必要です。なお、カウントのビット長が直ちにカウント時間にはなりません。たとえば、TM00 は周辺機能動作クロック周波数を 1/256 分周したクロックを 16 ビット分カウントするので、最大で 16M カウントまで可能です。一方、8 ビットの TM80 は 1/65536 分周したクロックを 8 ビット分カウントするので、最大 16M カウントまで可能で、16 ビットの TM00 と同じです。違いは**分解能**（どれだけ細かな間隔でカウントできるか）だけです。つまり、測定したい時間とその分解能の両方を考慮して、使い分ける必要があります。

16 ビット・タイマの PPG 出力を使った PWM 出力(1) [78K0、小ピン・マイコン共通]

Q1. 16ビット・タイマのPPG出力機能を利用してPWM出力する場合、デューティの変更はどうすればいいですか。

A1. デューティ（アクティブ期間）を指定するためのコンペア・レジスタを書き換えるタイミングを現在の値より大きくするか小さくするかで、変更する必要があります。

Q2. 78K デバイスの PPG 出力の FAQ を参照して、割り込みを使い分けましたが、デューティが小さいと出力がおかしくなります。具体的なプログラム例を紹介してください。

A2. PPG 出力を用いた PWM 出力について、考え方とお問い合わせの現象から説明します。

PPG 出力による **PWM 出力**は、通常は 8 ビット以上の分解能で行います。PPG 出力では、CR000（周期を指定）と CR010（アクティブ期間を指定）の 2 つのコンペア・レジスタを使用します。PWM 出力では、出力は**ロウパス・フィルタ**を通して DC 成分を使用します。このとき周期が長いと、フィルタの時定数が大きくなり、部品の値が大きくなるなどの問題が発生することがあります。そこで、できるだけ周期を短くします。そのため**カウント・クロックはできるだけ高速**にします。

この使い方で一番問題になるのは、**CR010 の書き換え**による出力動作中のデューティの変更です。デューティを変更するために、任意のタイミングで CR010 を書き換えると、1 周期の期間で CR010 との一致が全く発生しなかったり、一致が 2 回発生したりすることが考えられます（こうなると、**デューティが逆転**します）。これを避けるには、現状の設定値と新しい設定値の関係により、INTTM000 の発生タイミング（周期が完了したことを示す）で書き換えたり、INTTM010 の発生タイミング（アクティブ期間が完了したことを示す）で書き換えたりする必要があります。使用方法の基本的な考え方としては、次のようになります。

INTTM000 ：設定値を大きくする場合に使用

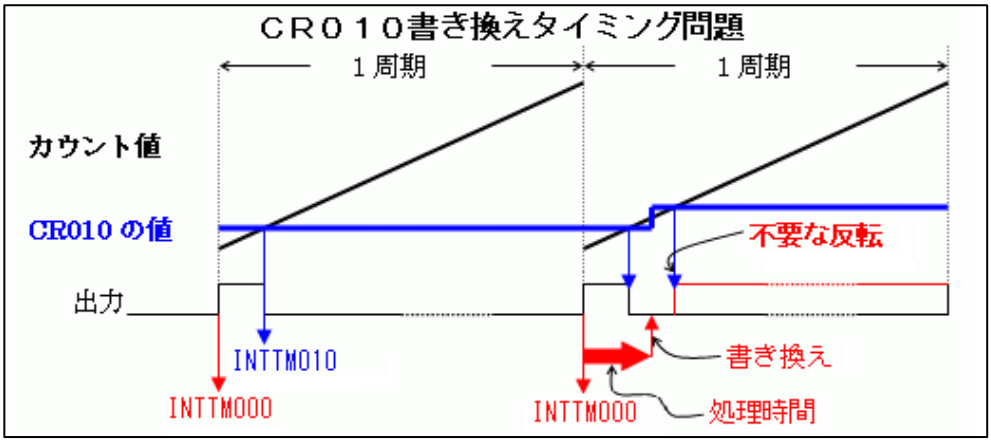
INTTM010 ：設定値を小さくする場合に使用

この書き換えのタイミングには、あまり余裕がなく、次のような制約があります。

①小さな設定値を大きくする場合、処理時間が厳しくなります。

②100%に近いデューティを 0%近くに変更する場合、書き換えタイミングに間に合わなくなります。

PWM 出力では、②のような処理はあまり考えられないので、ここでは特に対策しません。①について、詳しく説明します。INTTM000 の割り込み要求で書き換える場合、必ず処理時間（割り込み要求を受け付けるまでの時間と割り込みでの処理時間）が必要になります。その結果、実際に CR010 が書き換えられるタイミングが、次図のように、CR010 との一致タイミング（INTTM010）より遅くなることが考えられます。書き換えたとき、タイマ 00 が書き換えた値より小さいと、赤い波形のように 2 回目の一致が発生（デューティが逆転）します。



この問題に対応するため、INTTM010 で処理に必要な時間分を加味して、INTTM000 と INTTM010 の使い分けを見直します（INTTM010 での**処理が完了する時間**までの増加分まで INTTM010 で処理して、INTTM000 での処理遅れをカバーすることを考えます）。さらに、処理時間に余裕をもたせるため、INTTM010 での処理内容を見直します。具体的には、0%や 100%出力のための処理を INTTM010 での処理に追加し、INTTM010 で CR010 を書き換えるタイミングを数十クロックかかるようにします（以下、例として COFFSET で 70 クロックをしきい値に定義します）。これに伴い、INTTM010 で処理する条件を次のように変更します。

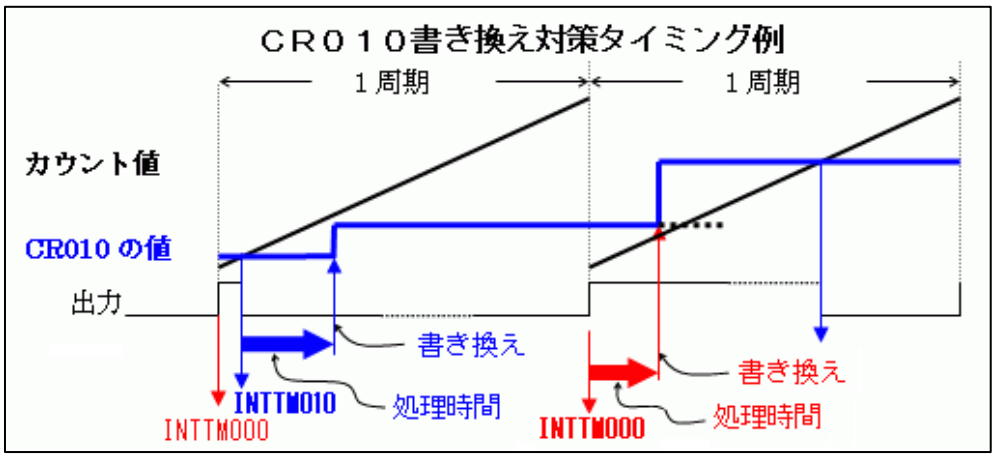
①設定値が、現在の値+70（COFFSET）以下の場合

②＋70 以上に変更する場合で、現在の設定値が 70（COFFSET）以下の場合

さらに②の場合は、いったん 70 に変更してから、INTTM000 割り込み処理で実際の設定値に変更します。これにより、INTTM000 の処理が 70 クロック以下で処理できれば、上記の問題はありません（実際の処理が 30 クロックで、割り込み応答は最悪の場合に 19 クロックとなるので、20 クロックの余裕があります）。

一方、INTTM010 での処理では、変更した条件により、CR010 の書き換えまで 70 クロック以上必要ですが、これを満足するような時間になっています。このときの書き換えの例を次図に示します。

この対策タイミング例のように、INTTM010 の最短時の処理時間より短く、IMTTM000 の最長時の処理時間より長い時間に、いったん設定することで逆転を防いでいます。



以上の動作を行うためのレジスタの設定は、次のようになります（**赤い太字**が指定部分）。

CRC00　：**000000x0**（CR000 と CR010 をコンペア・レジスタに）

PRM00　：**xxxx0000**（カウント・クロックは周辺機能動作クロック（f_{XP}））

TOC00　：**0001xx11**（コンペア・レジスタとの一致で出力を反転）

TMC00　：**0000110x**（TM00 と CR000 の一致でクリア＆スタート）

TOC00 については、出力を直接指定するときにビット 2、3 を使用します。また、0%出力や 100%出力のときには、コンペア・レジスタとの一致で出力を反転しないようにするためにビット 4 と 1 を 0 にします。

タイマ 00 の動作モードとしては、次の表に示す 5 つがあります。この表は、各モードでの出力の状態と、TMC00 レジスタや TOC00 レジスタの設定を示します。

MODEFLAG	動作状態	出力	TMC00	TOC00
0	停止状態（周期変更）	ロウ・レベル	00000000	00000101
1	0%出力	ロウ・レベル	00001100	00000101
2	0%＜出力＜100%	PWM	00001100	0001x011
3	100%出力	ハイ・レベル	00001100	00001001
4	70 以下から 70 以上増加	PWM	00001100	00010011
0xFF	処理完了	―	―	―

では、具体的なプログラムについて説明します。プログラムは、main 関数と初期設定を行う hdwinit 関数に加えて、次の 2 つの関数と 2 つの割り込み関数から構成されます。

- stopppg 関数（タイマを停止して周期を設定する）
- setppg 関数（デューティを変更する）
- INTTM000 関数（CR010 をデューティが増加する変更を行う）
- INTTM010 関数（モード変更やデューティを小さくする）

また、制御のために、8 ビットと 16 ビットの各 2 つずつの符号なし変数を saddr 領域に確保してあります。

PERIOD　　：周期を保持しておく
ACTIVETIME：2 つの割り込み関数に CR010 への設定値を渡す
MODEFLAG　：INT010 に次のモードを指定する（また、0xFF で指定された動作が完了したことを示す）
TEMPFLAG　：実際の動作モードを示す

これらを含めた宣言部分は、次のようになります。

```
#define IPTIME 0x3FF
#define IATIME 0x2
#define TOUTMODE PM3.1
#define COFFSET 70
```

最初の#define 文は、使用する定数を定義しています。IPTIME と IATIME は、周期とアクティブ期間の初期値です。TOUTMODE は、タイマ 00 の出力が兼用になっているポート・モードを定義しています。78K0S/KA1+、KB1+ではポート 3.1 が兼用なので、PM3.1 と定義しています。COFFSET は、INTTM010 での処理時間を示すオフセット値で、これまで説明してきた 70 クロックの大きさを定義しています。

その後は、必要な宣言や割り込み関数の宣言や、グローバルな変数を定義しています。

```
#pragma SFR
#pragma EI
#pragma DI
#pragma NOP
#pragma INTERRUPT INTTM000 TM00INT
#pragma INTERRUPT INTTM010 TM01INT
__interrupt void        TM00INT (void);
__interrupt void        TM01INT (void);
bit     setppg(unsigned int);
void    stopppg(unsigned int);
/*      CR010 や CR000 に設定するための新しい値の保持領域  */
sreg static unsigned int ACTIVETIME, PERIOD;
```

```
/*
        PWM 出力としての動作モード指定用フラグ
                0  : 動作停止（タイマ停止で 0 出力固定）
                1  : 0%出力（タイマは動作、出力は 0 固定）
                2  : 指定デューティ出力
                3  : 100%出力（タイマは動作、出力は 1 固定）
                4  : 大きく変更す場合の途中処理（MODEFLAG だけ）
                FF : 処理完了（MODEFLAG だけ）
*/
sreg static unsigned char MODEFLAG, TEMPFLAG;
```

次に、初期設定部分(hdwinit関数に記述)は、次のようになります。

```
        TOUTMODE = 0;           /* 兼用ポートを TO00 出力用に設定  */
        CRC00 = 0b00000000;     /* コンペア・レジスタで使用        */
        PRM00 = 0b00000000;     /* カウント・クロックは fXP        */
        CR000 = IPTIME;         /* CR000 は周期設定用              */
        CR010 = IATIME;         /* CR010 はアクティブ期間設定用    */
        TOC00 = 0b00000100;     /* 出力を 0 レベルで出力禁止       */
        IF0 = 0x00;             /* 割り込み要求をクリア            */
        MK0 = 0x0FF;            /* 最初は割り込みをマスク          */
```

ここでは、最初にタイマ出力の兼用ポートを出力に設定しています。次に、タイマを制御する CRC00 レジスタと PRM00 レジスタの設定を行います。他のレジスタの設定もしますが、まだタイマは起動しません。また、割り込み関係のレジスタも設定します(まだ、割り込みはマスクした状態にしておきます)。次に示す変数の初期化は、ここでは main 関数の最初に処理しています。

```
        PERIOD = IPTIME;        /* 変数を初期化しておく            */
        ACTIVETIME = 0;         /* 最初は出力をしない状態          */
        TEMPFLAG = 0;           /* 同じく作業用フラグをクリア      */
        MODEFLAG = 0xFF;        /* フラグを処理完了状態にセット */
```

実際の処理部分のうち、停止させる処理は、次の関数で処理します。この関数は、新しい周期のデータを引数として、呼び出されるようになっています。

```
/*
        PPG の停止処理
*/
void    stopppg(unsigned int    newtime){
        PERIOD = newtime;
        if( TEMPFLAG == 0 ){
                TOC00 = 0b00000101;     /* 出力を 0 に固定する    */
                CR000 = PERIOD;
                MODEFLAG = 0xFF;        /*完了フラグをセット     */
                TMIF000 = 1;
        }else{
                MODEFLAG = 0;           /* 以降で出力を 0 に固定 */
                TMIF000 = 0;
                TMIF010 = 0;
                TMMK010 = 0;
        }
}
```

関数が呼び出されて、そのときの出力デューティが 0%か 100%ならタイマを停止し、周期を変更してから停止状態(TEMPFLAG を 0)に設定して完了します。出力デューティが 0%と 100%の間なら、次の INTTM010 で、実際に処理するためにモードを指定(MODEFLAG を 0)して、NTTM010 の割り込みマスクを解除します。

次に、PWM のアクティブ期間を変更する関数を示します。この関数は割り込みと連携して動作し、処理が実際に完了する前に抜けてしまいます。これにより、ほかの必要な処理を平行して実行できるようにしています。そのため、処理が完了する前に再度呼び出されると、動作状態がおかしくなる可能性があるので、最初に処理が完了しているかどうかを確認しています(完了していなければ、エラーで戻る)。その後、新しいアクティブ期間と現在の動作モードや設定値に応じて、いくつかに場合分けして処理しています。

次の動作	現在の状態	処　理
0%出力	①タイマ停止中	出力を 0に固定し、反転禁止にしてタイマ動作再開
	②タイマ動作中	MODEFLAGを 1にして INTTM010 待ち
100%出力	③タイマ停止中	出力を 1に固定し、反転禁止にしてタイマ動作再開
	④タイマ動作中	MODEFLAGを 3にして INTTM010 待ち
PWM 出力	⑤PWM出力中以外注1	いったんタイマを停止し、出力を 1に指定し、反転許可にしてタイマ動作再開
小さくする注2	⑥PWM出力中	MODEFLAGを 2にして INTTM010 待ち
大きくする注2	⑦70以上で動作中注3	MODEFLAGを 2にして INTTM000 待ち
	⑧70以下で動作中注3	MODEFLAGを 4にして INTTM010 待ち

注 1.アクティブ期間を 0%出力から長くしたり、100%出力から短くしたりする場合も含みます。
2.現在の設定値＋COFFSET を判断基準として処理します。
3.現在の設定値が COFFSET 以上か以下かで判断します。

ここで、⑥～⑧が通常の PWM のデューティ変更に該当する処理です。以下に、実際のプログラムを示します。最初のチェック部分は、次のように処理中ではエラーで戻ります。

```
bit     setppg(unsigned int     newtime){
        if( MODEFLAG != 0xFF ) return 1;        /* 前回の処理中ならエラー    */
```

次は、上記表中の①②の 0%出力にする部分です。2 行目のif文で、①と②を分けています。タイマが停止中なら、タイマ出力を 0 に固定してタイマを起動します。タイマが動作中なら、フラグを設定して、INTTM010 割り込みで実際に設定変更の処理を行います。

```
    if( newtime == 0 ){
            if( TEMPFLAG == 0 ){
                    CR010 = COFFSE*2T;      /* COFFSET 値に設定しておく*/
                    TOC00 = 0b00000101;     /* 出力を 0 に固定する    */
                    TMC00 = 0b00001100;     /* タイマを起動する       */
                    TEMPFLAG = 1;
            }else{                          /* 出力中なら INTTM010 を待ち     */
                    MODEFLAG = 1;   /* 以降で出力を 0 に固定する      */
                    TMIF010 = 0;
                    TMMK010 = 0;
            }
```

次は、前記表中の③④の 100%出力にする部分です。3 行目の if 文で、③と④を分けています。

```
        }else{
        if( newtime >= PERIOD ){        /* 周期以上を設定した場合は   */
                if( TEMPFLAG == 0 ){
                CR010 = COFFSET*2;      /* COFFSET 値に設定しておく    */
                TOC00 = 0b00001001;     /* 出力を 1 に固定する         */
                TMC00 = 0b00001100;     /* タイマを起動する            */
                TEMPFLAG = 3;
        }else{                          /* 出力中なら INTTM010 を待ち */
                MODEFLAG = 3;           /* 以降で出力を 1 に固定する   */
                TMIF010 = 0;
                TMMK010 = 0;
                }
```

次の部分は、⑤のタイマとして PPG 出力を行っていない状態から、PPG 出力を開始する処理です。ここでは、レジスタの設定を直接行い、出力を 1 に設定してタイマを起動しています。

```
        }else{
        if( TEMPFLAG !=2 ){     /* 元が固定出力の場合は          */
        ACTIVETIME = newtime;   /* 直接タイマを操作して起動      */
        TMC00 = 0b00000000;     /* タイマを一旦停止する          */
        CR010 = ACTIVETIME;     /* アクティブ期間を設定する      */
        TOC00 = 0b00011011;     /* 出力は 1 から開始             */
        TMC00 = 0b00001100;     /* タイマを起動する              */
        TEMPFLAG = 2;
```

次の部分は、⑥の値を小さくする場合の処理です。現在の設定値＋COFFSET を判断基準として、それより小さくします。実際の処理のために MODEFLAG に 2 を設定して、INTTM010 を許可しています。

```
        }else{                  /* そうでなければ値で判定        */
        MODEFLAG = 2;           /* デューティ比変更モード        */
                                /* INTTM010 処理時間含め比較     */
        if( newtime < (ACTIVETIME + COFFSET)){
        ACTIVETIME = newtime;   /* 小さくする場合は              */
        TMIF010 = 0;
        TMMK010 = 0;            /* INTTM010 で処理する           */
```

次の部分は、⑦の値を大きくする場合の処理です。現在の設定値が COFFSET 以上であることを判断して、実際の処理は INTTM000 で行うために、INTTM000 を許可しています。

```
        }else{
        if( ACTIVETIME >= COFFSET ){
        ACTIVETIME = newtime;   /* 大きくする場合は      */
        TMIF000 = 0;
        TMMK000 = 0;            /* INTTM000 で処理する   */
```

最後が、⑧の COFFSET 以下から COFFSET 以上大きくする場合の処理です。ここでは、2 段階で処理するために、MODEFLAG に 4 を設定して、INTTM010 を許可しています。

```
        }else{
                MODEFLAG = 4;
                TMIF010 = 0;
                TMMK010 = 0;     /* INTTM010 で処理する          */
                ACTIVETIME = newtime;
        }
        }
        }
}
}
return 0;
}
```

以上の処理で、INTTM000 か INTTM010 が許可され、割り込みが発生したときに、以下の処理を行います。

INTTM000 の割り込み処理では、CR010 の設定値を大きい方向に変更する処理だけを行います。プログラム中の赤文字の部分までの処理時間に制限があります。

```
/*
                周期完了割り込み
        デューティを大きくする方向での変更処理
*/
__interrupt     void    TM00INT (void){
        TOC00 = 0b00000011;     /* 反転禁止に設定する          */
        CR010 = ACTIVETIME;     /* 新しい値を設定              */
        TOC00 = 0b00010011;     /* 反転を許可する              */
        TMIF010 = 0;            /* 割り込み要求をクリア        */
        TMMK000 = 1;            /* 割り込み要求をマスクする    */
        MODEFLAG = 0xFF;        /* 処理完了フラグをセット      */
}
```

次の部分は、**INTTM010 の処理**部分です。ここでは、MODEFLAG に設定されている値によって、その後の処理が異なります。そこで、swtch 文でその後の処理を判定しています。

MODEFLAG が 0 の場合には、タイマを停止して、出力を 0 に固定します。停止処理を行い、新しい周期を CR000 に設定し、処理完了します。

```
__interrupt     void    TM01INT (void){
        TMIF000 = 0;
        TMMK010 = 1;            /* INTTM010 をマスク           */
        switch ( MODEFLAG ){
        case 0 :                /* PWM 出力の停止処理          */
        TMC00 = 0b00000000;     /* タイマを停止する            */
        TOC00 = 0b00000100;     /* 出力を 0 レベルで出力禁止   */
        CR000 = PERIOD;         /* 周期の変更                  */
        TEMPFLAG = 0;
        MODEFLAG = 0xFF;        /* 処理完了フラグをセット      */
        TMIF000 = 1;
        break;
```

MODEFLAG が 1 の場合には、0%出力を行います。まだ 0%出力でない場合は、出力を 0 に固定して、タイマ出力を反転禁止にして処理を終了します。

```
        case 1 :          /* PWM 出力の 0 レベル固定処理 */
        if ( TEMPFLAG != 1 ){
                TMC00 = 0b00000000;
                TOC00 = 0b00000101;
                CR010 = COFFSET*2;
                TMC00 = 0b00001100;
                TEMPFLAG = 1;
        }
        MODEFLAG = 0xFF;          /* 処理完了フラグをセット*/
        break;
```

MODEFLAG が 2 の場合には、CR010 に新しい値を設定します。TOC00 への設定（赤文字部分）までに、タイマ 00 のカウント値が新しい設定（ACTIVETIME＋COFFSET 以下に制限）以上になる処理をしています。

```
        case 2 :                  /* 指定されたデューティでの出力 */
        TOC00 = 0b00000011;       /* 出力を反転禁止に設定          */
        CR010 = ACTIVETIME;
        TMIF010 = 0;              /* 割り込み要求をクリアする      */
        if ( TEMPFLAG == 1 ){
        TOC00 = 0b00010011;       /* 反転を許可する                */
        } else {
        if ( TEMPFLAG == 2 ){
                TOC00 = 0b00010011;       /* 反転許可に戻す        */
        } else {
                TOC00 = 0b00010111;
                TMC00 = 0b00001100;
        }
        }
        TEMPFLAG = 2;
        MODEFLAG = 0xFF;          /* 処理完了フラグをセット        */
        break;
```

MODEFLAG が 3 の場合には、100%出力を行います。まだ 100%出力でない場合には、出力を 1 に固定して、タイマ出力を反転禁止にして処理を終了します。

```
        case 3 :                  /* PWM 出力の 1 レベル固定処理    */
        if ( TEMPFLAG != 3 ){
        TMC00 = 0b00000000;
        TOC00 = 0b00001001;
        CR010 = COFFSET*2;
        TMC00 = 0b00001100;
        TEMPFLAG = 3;
        }
        MODEFLAG = 0xFF;          /* 処理完了フラグをセット        */
        TMIF000 = 0;
        break;
```

MODEFLAG が 4 の場合には、CR010 への設定値を COFFSET（すでにこの値以上にカウントしている）として、残りの処理は INTTM000 で行います。

```
        case 4 :                    /* 小さな値から大きく変更する場合*/
        TEMPFLAG = 2;
        TMMK000 = 0;                /* 残りの処理は INTTM000 で       */
        TOC00 = 0b00000011;         /* 出力を反転禁止に設定           */
        CR010 = COFFSET;            /* 作業用の値をセットする         */
        TMIF010 = 0;                /* 割り込み要求をクリアする       */
        TOC00 = 0b00010011;         /* 反転許可に戻す                 */
        break;
        }
}
```

以上が、タイマ 00 の PPG 出力で **PWM 出力**を行うプログラムの処理部分の例です。このプログラムでは、設定のために呼び出された関数からは、必要な設定が終わった段階で戻ってきます。関数コールから戻ってきた段階では、実際のタイマ 00 への設定が行われていないことがあるので、この例では設定完了を示すフラグを準備しています。それが **MODEFLAG** で、これが **0xFF** になったら、タイマ 00 への**設定は完了**したことになります。設定した値でタイマ 00 が **1 周期分動作したか**どうかは、**INTTM000** の発生の有無で判断します（タイマ 00 を停止すると1周期のカウントができないので、タイマ 00 を設定した段階で INTTM000 を設定しています）。

これらの関数を使って PWM 出力を行うには、次のような処理となります。この例では周期を設定しています。赤字の部分は、タイマ 00 での処理完了待ちをしている部分です。

```
        stopppg(IPTIME);            /* 周期を設定                 */
        while((MODEFLAG != 0xFF)||(TMIF000 == 0)){
        NOP();                      /* 処理の完了待ち             */
        }
```

次の例では、PWM のアクティブ期間を IATIME 刻みで増加させる処理をしています。プログラム全体は、アクティブ期間の指定値を for ループで IATIME 刻みに増加させながら、赤字の部分で PWM のアクティブ期間を設定する関数をコールしています。この例では、関数コールの結果を確認して、エラーが検出されていれば、青字の部分でループを回しています。

```
        for (work = IATIME; work < PERIOD; work += IATIME)
        if(setppg(work)){
        while(1){
                NOP();
        }
        }
        while((MODEFLAG != 0xFF)||(TMIF000 == 0)){
                NOP();                      /* 処理の完了待ち         */
                }
        NOP();
        }
```

★ 16 ビット・タイマの PPG 出力を使った PWM 出力(2) [78K0、小ピン・マイコン共通]

Q1. 「16 ビット・タイマの PPG 出力を使った PWM 出力 1」の A2 に「100%に近いデューティを 0%近くに変更すると、書き換えタイミングに間に合わない」という記述があります。タイマ 0 で 0～100%デューティの PWM 信号を出力するにはどうすればいいですか。

A1. 完全な対応方法はありませんが、次のような 2 通りの方法が考えられます。

(1) タイマをいったん停止し、CR010（アクティブ期間を指定）と出力レベルを設定してから再起動する

(2) CR010 に 50%程度の値を設定し、「100%付近→50%程度→0%付近」と 2 段階で変更する

(1)では、タイマの停止タイミングによって、中間にデューティ不定状態が 1 サイクル入ります。そのタイミングを CR010 との一致割り込みにすると、中間状態のデューティのばらつきを小さくできます。

(2)では、中間状態は必ず発生しますが、そのデューティにばらつきはありません。

PWM では、ロウパス・フィルタを通して、DC 成分を利用します。デューティを「100%付近→0%付近」に変更すると、必ず 100%付近に対応した電圧から中間的な電位を経て 0%付近に対応した電圧に遷移するので、中間のデューティが存在しても問題にはならないと考えられます。

Q2. どのようなプログラムになりますか。

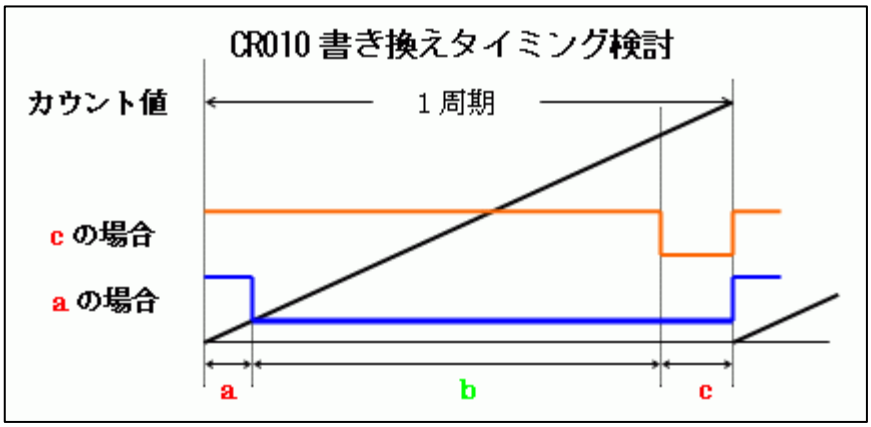
A2. タイマをいったん停止させ、出力反転禁止期間を考察し、デューティを変更するプログラム例を紹介します。

[タイミングの検討]

PPG による PWM 出力での、デューティ変更のための CR010 書き換えタイミングの問題については、「**16 ビット・タイマの PPG 出力を使った PWM 出力 1**」の「**CR010 書き換えタイミング問題**」もご参照ください。

タイマ 00 を用いた PPG で、CR010 を書き換えるタイミングの検討では、INTTM000 割り込み処理と INTTM010 割り込み応答時間に十分な注意が必要です。割り込み応答時間は、割り込み要求の発生からそれが受け付けられ、割り込みベクタに分岐するまでの割り込み要求受け付け時間と、プログラムを実行するための処理時間で構成されます。

78K/0 の割り込み要求受け付け時間は、割り込み許可状態なら、要求発生時に実行中の命令の実行時間＋7 クロック程度です。乗除算命令を使用していなければ、9～20 クロックで受け付けられます。これに割り込み処理プログラムでの、命令の処理時間を加算すれば、割り込み応答時間が求められます。次の「**CR010 書き換えタイミング検討**」は、現在の CR010 の設定値による場合分けを示したものです。



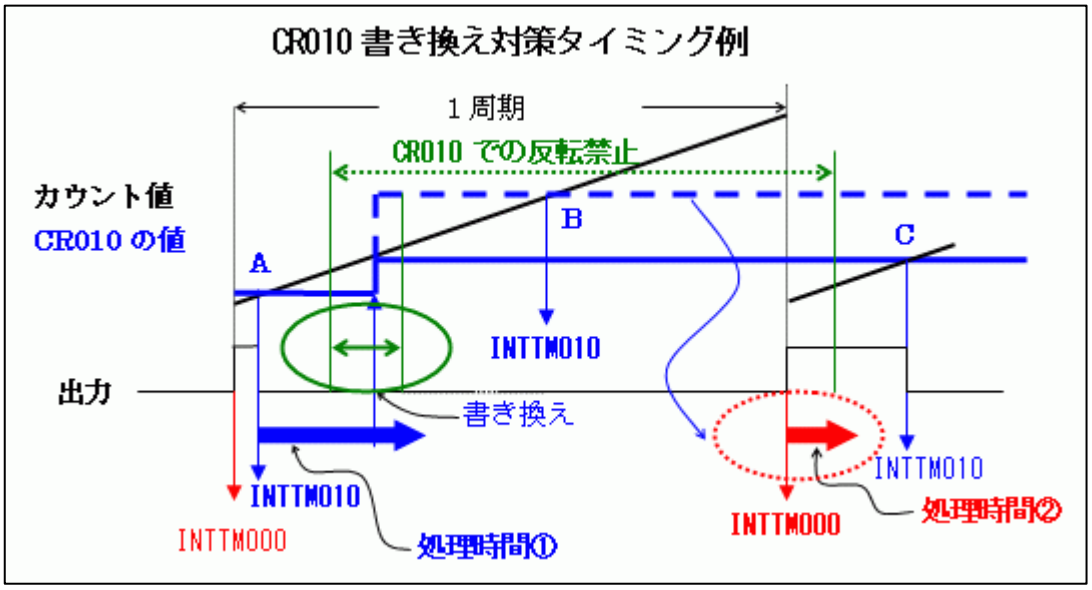
(1)「a」(デューティ比が 0%に近い)の部分

これは、INTTM000 の割り込み応答時間の影響を考慮すべき部分です。CR010 の値がこの範囲にあると、INTTM000 での処理では割り込み応答が間に合わない可能性があります。そのため、INTTM010 割り込みを使用する必要があります。ただし、CR010 の値を小さくする場合は問題ありませんが、大きくする場合は注意が必要です(「**16 ビット・タイマの PPG 出力を使った PWM 出力 1**」の「**CR010 書き換えタイミング問題**」参照)。この問題を解決するために、ここでは INTTM010 と INTTM000 の両方の割り込みを使い、処理を 2 つの部分に分けて対応します。

①**INTTM010 での処理**　:CR010 との一致での反転を禁止して、CR010 を書き換えます。
　　INTTM010 割り込みはマスクします。

②**INTTM000 での処理**　:CR010 との一致での反転を許可します(固定)。

②の処理時間の最大値が①の処理時間の最小値以下にする必要があるので、②はできるだけ簡単な処理で済ませるようにします。これにより、次のようなタイミングになります。



これは設定値を大きくする場合の例です。**A 点**でカウント値と CR010 値が一致して出力が反転し、**INTTM010** 割り込みが発生します。その処理の中で、CR010 との一致での出力反転を禁止します。その後、CR010 の値を書き換えます。新しい設定値に変わるまでにカウントが完了する(**青**の実線)場合、その周期の中でさらに一致することはないので、問題ありません。出力反転を許可して処理を終了します(**緑**の楕円の範囲内で、設定値を小さくする場合と同じ処理)。

新しい設定値に変わるまでにカウントが完了しない(**青**の破線)場合、出力反転禁止の状態で、CR000 との一致割り込みを許可し、INTTM010 割り込み処理を終了します。その周期の中で再度一致が発生(**B 点**)しますが、出力反転禁止になっているので、**出力は反転しません**。その後、CR000 との一致で、**INTTM000** 割り込みが発生します。ここで、CR010 との一致での出力反転を許可します。以降は通常の動作となり、**C 点**で CR010 との一致が発生すると、出力が反転して指定されたデューティの PWM 信号となります。

処理方法としては、次のような考え方です。

設定値を一定以上大きくする場合：**INTTM010** で反転禁止と CR010 の変更をする
　　INTTM000 で反転許可にする
その他の場合　　：**INTTM010** ですべて処理する

(2) 「b」の部分

基本的に、次のような割り込みの使い分けで対応できます（(1)の処理も可能）。

設定値を一定以上大きくする場合：INTTM000 で処理する
その他の場合　　：INTTM010 で処理する

(3) 「c」（デューティ比が 100%に近い）の部分

この部分から「0 に近い値」に変更するときだけが問題です。その他の場合は、「b」の部分と同様です。
「c」や「0 に近い値」の範囲は、INTTM000 と INTTM010 の割り込み処理時間で決まります。A1 で述べたように処理する必要があります。

[処理内容のまとめ]

以上の内容から、CR010 の現在の設定値と新しい設定値の関係で場合分けし、次のような処理になります。ここで、「CR000 付近」はデューティ比が 100%付近の状態、「0 付近」はデューティ比が 0%付近の状態を示します。オフセットは、INTTM000 割り込み要求発生から反転許可まで（(1)の②）の処理クロック数（MAX.）以上で、INTTM010 割り込み要求発生から反転許可まで（(1)の①＋②）の処理クロック数（MIN.）未満の値です。

処理の場合分け

CR010 の値		処理内容
現在の値	新しい値	
CR000 付近	0 付近	タイマを停止して再起動
	現在の値＋オフセット以上	INTTM010：反転禁止＋設定値変更＋INTTM000 許可 INTTM000：反転許可
	現在の値＋オフセット未満で 0 から遠い	INTTM010：反転禁止＋設定値変更＋反転許可
CR000 から遠い	現在の値＋オフセット以上	INTTM010：反転禁止＋設定値変更＋INTTM000 許可 INTTM000：反転許可
	現在の値＋オフセット未満	INTTM010：反転禁止＋設定値変更＋反転許可

[処理プログラム例]

以上の内容に基づくプログラム例について説明します。
このプログラムの主要部は、説明のように割り込み処理部です。この処理を正しく行うには、最初に現在の CR010 の設定値と新しい設定値を比較して、処理を決定する場合分け処理部が重要です。これらの処理が連携することで、プログラムが正しく動作します。

(1) 場合分け処理部

場合分けする部分は次のようになります。場合分けする関数と割り込み処理部で、全体の処理が完了します。この関数では、最初に変更されたかどうかを確認し、変更がなければ終了します。変更があったときの場合分けを簡単にするために、まずオフセットを加味して、大きくなるか小さくなるかを判定します。大きくなる場合は、g_ucMode に「02」を設定します（次のプログラムでピンクの背景部分）。小さくなる場合は、100%付近から 0%付近への変更かどうかを確認することで、判定を簡素化しています。また、フラグ（g_ucMode）が「00」になるのを確認することで、処理の完了まで待ちます。

```
void    setppg(unsigned int     p){
        olddata = g_unNextD;    /* 現在の設定値を旧データに      */
        g_unNextD = p;          /* 新しい設定値をセット          */
        if(g_unNextD == olddata) return;/* 同じ値は設定しない    */
if( g_unNextD > (olddata + C_Offset) ){
                                /* 現在値＋オフセット以上        */
                g_ucMode        = 02;
        }else{                  /* 現在値＋オフセット以下        */
                if( olddata < (g_unNextD + (CR000 - E_Limit))){
                                /* 現在値＋オフセット以下        */
                        g_ucMode        = 03;
                }else{          /* 100%付近から 0%付近に変更     */
                        g_ucMode        = 01;
                }
        }
        TMIF010 = 0;            /* これまでの割り込み要求クリア  */
        TMMK010 = 0;            /* INTTM010 を許可               */
        while(g_ucMode){        /* 処理完了を待つ                */
                NOP();
        }
}
```

ここで使用している定数 C_Offset、E_Limit については、(5) (6)で説明します。

(2) INTTM000 での処理例

単に TOC00 レジスタのビット 4(TOC004 ビット)をセットし、TMMK000 をセットするだけとなります。反転許可までの時間を短くする必要があるので、最初に反転許可を設定します。以下の処理ではレジスタを使用しないので、レジスタのセーブ処理は行われません。他の処理を追加した場合は、レジスタのセーブに時間が取られないようにする必要があります。このため、割り込み関数の宣言で、レジスタ・バンクの使用を宣言してください。

```
__interrupt     void    fn_inttm000(void){
        TOC00.4 = 1;            /* 反転許可                      */
        TMMK000 = 1;            /* INTTM000 マスク               */
        g_ucMode = ModeE;       /* 処理完了                      */
/*
        他の処理が必要ならここに記載する
*/
}
```

(3) INTTM010 での処理例

具体的なプログラム例を紹介します。ここで、変数 g_unNextD には次に CR010 に設定する値、g_ucMode には動作を指定する値が、それぞれ格納されているものとします。

```
__interrupt     void    fn_inttm010(void){
        TMIF000 = 0;            /* これまでの割り込み要求クリア */
        TOC00.4 = 0;            /* 反転禁止                     */
        TMMK010 = 1;            /* INTTM010 マスク              */
        CR010   = g_unNextD;    /* デューティ(パルス幅)変更     */
        switch( g_ucMode ){
                case 1:         /* 100%付近から 0%付近へ        */
                        TMC00   = 0b00000000;   /* タイマ停止   */
                        TOC00   = TOC_MODE;     /* 初期設定     */
                        LVS00   = 1;            /* 初期値 H     */
                        TOE00   = 1;            /* 出力許可     */
                        TMC00   = 0b00001100;   /* タイマ再起動 */
                        g_ucMode = ModeE;       /* 処理完了     */
                        break;
                case 2:         /* デューティを大きくする       */
                        TMMK000 = 0;    /* INTTM000 割り込み許可 */
                        break;
                case 3:         /* デューティを小さくする       */
                        TOC00.4 = 1;            /* 反転許可     */
                        g_ucMode = ModeE;       /* 処理完了     */
                break;
        }
/*
        他の処理が必要なら、ここに記載する
*/
}
```

(4) 初期設定

これらを使用するための初期設定を次に示します。hdwinit 関数では、クロックの設定と使用する端子に関係したポートの設定をしています。

```
void hdwinit(void){
        PCC = 0x0;                      /* 高速動作モード設定 */
                                        /* ポート設定         */
        P0 = 0b11111101;                /* P01 をクリアする   */
        PM0.1 = 0;                      /* P01 を出力に設定   */
        }
```

使用する変数やタイマは、main 関数の中で設定します。

```
void main(void){
/*
        使用する変数の初期化
*/
        g_ucMode        = 0;    /* デューティ変更なし状態に初期化        */
        g_unNextD       = active_time;

/*
        タイマ 00 の設定 (PPG モード、カウント・クロックは fx、初期値 H)
*/
```

```
        PRM00 = 0b00000000;     /* カウント・クロックは fx                    */
        CRC00 = 0b00000000;     /* CR000,CR010 をコンペア・レジスタに設定 */
        CR000 = intervalTM00;   /* 周期初期値を設定                          */
        CR010 = active_time;    /* アクティブ期間初期値設定                  */
        g_unNextD = active_time;    /* アクティブ期間初期値設定              */
        TOC00 = TOC_MODE;       /* CR000,010 との一致で反転                  */
        LVS00 = 1;              /* 初期値 H                                  */
        TOE00 = 1;              /* 出力許可                                  */
        TMC00 = TMC_MODE;       /* TM00 と CR000 の一致でクリア＆スタート */
        TMMK000 = 1;            /* 割り込みはまだマスクしておく              */
        TMMK010 = 1;            /* 割り込みはまだマスクしておく              */
        EI();
```

(5) オフセット値の決定

INTTM010だけで処理するかINTTM000も使用するかの判断は、INTTM010の処理時間と割り込み受け付け時間を加味して行います。判断基準となるのは、「**case3**」で反転許可にするまでの時間（オフセット）です。コンパイラ・オプションでアセンブラ・ソースを出力し、処理クロック数をカウントして、オフセット時間を算出します。次に、見やすく編集したプログラムを示します。

```
        sel     RB2             ;[INF] 2, 4
        clr1    IF0H.6          ;[INF] 3, 8  TMIF000    = 0
        clr1    TOC00.4         ;[INF] 3, 8  TOC00.4    = 0
        set1    MK0H.7          ;[INF] 3, 8  TMMK010    = 1
        movw    ax,_g_unNextD   ;[INF] 2, 6  CR010      = g_unNextD
        movw    CR010,ax        ;[INF] 2, 6
```

ここまでが共通の処理部分で、40クロックとなります。オフセットはINTTM010割り込み要求発生から反転許可までの処理クロック数（MIN.）未満の値なので、最小値として割り込み処理待ちをゼロとします。割り込み要求受け付けが最小7クロックなので、共通部分との計が47クロックです。

その後は、次に示す割り込み処理で、switch文で最初に判断分岐するので、?L0045への分岐まで14クロック、分岐先でのビット操作で8クロックの計22クロックが追加され、合計69クロックとなります。

これが最小なので、1クロックの余裕をみて、ここでは68クロック（**C_Offset**）とします。

```
        mov     a,_g_ucMode     ;[INF] 2, 4  switch( g_ucMode ){
        cmp     a,#03H  ; 3     ;[INF] 2, 4
        bz      $?L0045         ;[INF] 2, 6
        cmp     a,#02H  ; 2     ;[INF] 2, 4
        bz      $?L0044         ;[INF] 2, 6
        dec     a               ;[INF] 1, 2
        bnz     $?L0042         ;[INF] 2, 6
??bb00_fn_inttm010:             ;               case 1:
        mov     TMC00,#00H      ;[INF] 3, 7  TMC00      = 0b00000000
        mov     TOC00,#012H     ;[INF] 3, 7  TOC00 = TOC_MODE
        set1    TOC00.0         ;[INF] 3, 8  TOE00 = 1
        set1    TOC00.3         ;[INF] 3, 8  LVS00 = 1
        mov     TMC00,#0CH      ;[INF] 3, 7  TMC00      = 0b00001100
        mov     _g_ucMode,#00H  ;[INF] 3, 6  g_ucMode = ModeE
        br      $?L0042         ;[INF] 2, 6  break
?L0044:                         ;               case 2:
        clr1    MK0H.6          ;[INF] 3, 8  TMMK000    = 0
        br      $?L0042         ;[INF] 2, 6  break
```

```
?L0045:                         ;               case 3:
        set1    TOC00.4         ;[INF] 3, 8   TOC00.4     = 1
        mov     _g_ucMode,#00H  ;[INF] 3, 6   g_ucMode = ModeE
?L0042:
        reti                    ;[INF] 1, 6
```

(6) 100%付近から0%付近までの決定

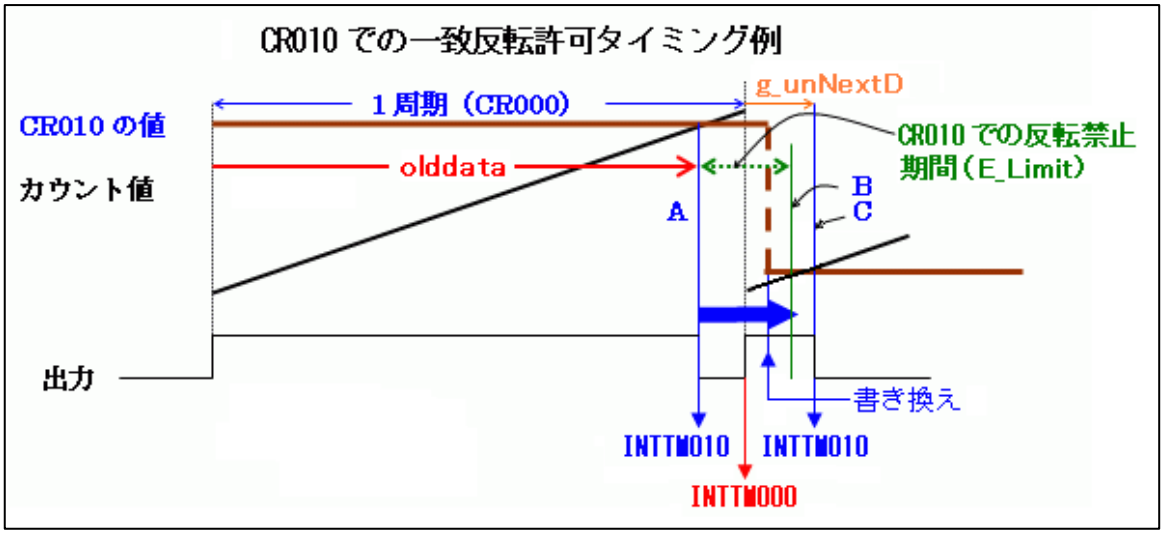
この判断には、INTTM010割り込み要求待ちから、INTTM000割り込みを許可（case2と同じ処理）し、反転許可（case3と同じ処理）までの最大値を使います。

(5)のプログラムの?L0044への分岐まで24クロック、分岐先の処理で14クロック、reti命令で6クロック、計44クロックです。

次に、INTTM000割り込み処理待ちとなります。待ち時間の最大値は、除算の33クロックですが、ここでは処理完了を待つ処理（ループ：最大6クロック）なので、割り込み要求受け付けの最大8クロックとの計が14クロックです（ここまで58クロック）。

さらに、反転許可（case3と同じ処理）への分岐まで14クロック、分岐先でのビット操作で8クロックの計22クロックが追加され、合計80クロックとなります。

これが最大なので、余裕をみて、ここでは100クロック（E_Limit）としています。



上図のように、INTTM010 発生(A)から、CR010 を書き換えて、CR010 での反転を許可(B)するまでの時間が、一致の発生(C)までより短ければ、通常のデューティを小さくする処理です。次に、これを式で表します。

CR000 ＋ g_unNextD ＞ olddata ＋ CR010 での反転禁止期間(E_Limit)

プログラムでは、次のような判定式となり、(1)のプログラムで緑の背景部分の if 文の判定条件です。

olddata ＜（g_unNextD ＋（CR000 － E_Limit））

(7) 定数や変数の定義

このプログラムで使用している変数や定数を次に示しておきます。

```
#define intervalTM00 300-1      /* CR000 にセットする周期データ初期値     */
#define active_time 01          /* CR010 にセットする初期値データ         */
#define TOC_MODE 0b00010010     /* CR000,010 との一致で反転、初期値 L     */
#define TMC_MODE 0b00001100     /* TM00 と CR000 の一致でクリア＆スタート */
#define ModeE 0                 /* 処理完了状態                           */
#define C_Offset 68             /* INTTM010 での反転許可までのクロック数  */
#define E_Limit 100             /* INTTM010 を大きくする場合の処理限界    */
```

```
sreg static unsigned int g_unNextD;     /* デューティ指定用      */
sreg static unsigned int olddata;       /* 前回の設定値用        */
sreg static unsigned char g_ucMode;     /* 動作モードフラグ
        00      : 処理完了(ModeE)
        01      : 100%近くから0%近くへ
        02      : デューティを大きく
        03      : デューティを小さく    */
```

(8) 使用方法

このプログラムでは、CR010 レジスタに設定する値を引数にして、**setppg** 関数を呼び出します。次に、実際の使用例を示します。なお、この例では、wait 関数を間に入れて、デューティを一度変更したら、数回は新しいデューティで動作させてから、次の変更を行うようにしています。

```
/* ここで、実際に PWM のデューティを変更します。                    */
/* デューティを変更する場合には、新しい CR010 の値を引数にして      */
/* setppg 関数を呼び出します。                                      */
/* 最初は最低値から最大値までインクリメント(+1)していきます。       */
                for ( i = active_time+1 ; i < intervalTM00 ; i++ ) {
                        wait(10);
                        setppg(i);
                }
/* 次は逆にデクリメントしていきます。                               */
/* これは INTTM010 だけの処理の確認です。                           */
                for( i = intervalTM00-1 ; i >= active_time ; i-- ){
                        wait(10);
                        setppg(i);
                }
/* 次は大きく (C_Offset 程度) していきます。                        */
                wait(10);
                j = C_Offset - 1;
                for( i = j ; i < intervalTM00 ; i += j ){
                        wait(10);
                        setppg(i);
                        j++;
                }
/* 次は 100%付近から 0%付近への変更とその逆の繰り返しです。        */
                for( j = 0 ; j < 120 ; j += 5 ){
                        wait(10);
                        setppg(intervalTM00 - 1);
                        wait(10);
                        setppg(j);
                }
```

タイマを組み合わせたパルス間隔測定 [78K0、小ピン・マイコン共通]

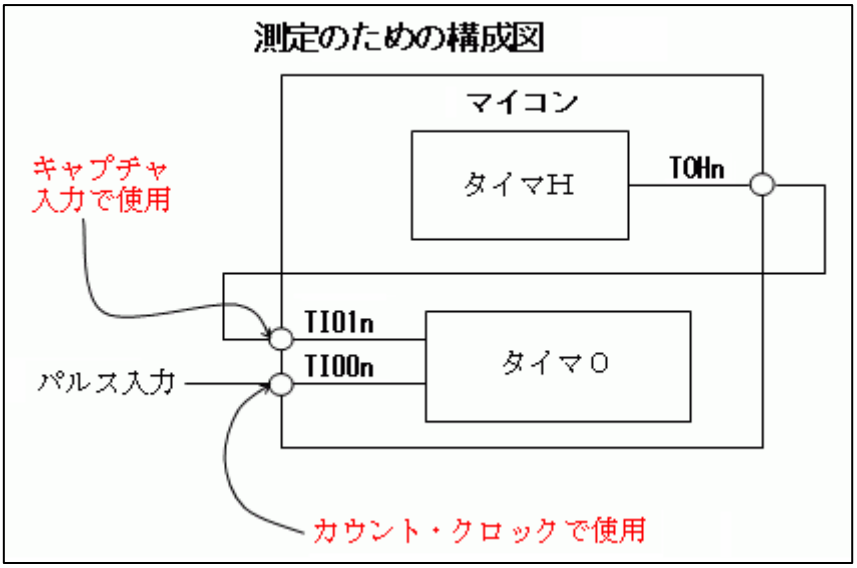
Q1. 約 1MHz の入力パルスの周波数を測定するには、どうすればいいですか。

A1. この周波数では、個々のパルスの間隔を測定するのは意味がありません。そこで、一定時間に入力されたパルスの数をカウントして、そこから平均周波数を求めることにします。

できるだけ誤差を小さくするために、2 つのタイマ／カウンタを組み合わせる方法を考えてみます。1 つは一定時間の測定に使用し、もう 1 つはパルスのカウントに使用します。さらに、CPU の割り込み応答時間の変動の影響を避けるために、パルスのカウント値をキャプチャすることを考えます。

[実現方法]

この方法では、時間計測にタイマ H を用い、その出力をタイマ 0 の TI01n に接続して、キャプチャに使用します。タイマ 0 の TI00n に、測定したいパルスを入力します。この構成を次図に示します（外部で TOHn と TI01n を接続します）。



次に、タイマ H で測定する時間を決定します。この時間が短いと、被測定パルスとの非同期による誤差の影響が出ますし、長すぎると変化の検出に時間がかかります。ここでは、1ms ごとに測定します。タイマ H で 8MHz のクロックから 1ms 周期の方形波を出力するには、1/32 に分周したものをカウント・クロックに使用します。その出力を 125 カウントで反転すれば、0.5ms ごとに反転するので、1ms の周期が得られます。

タイマ 0 はフリー・ランニング・モードで使用し、TI00n をカウント・クロックに指定します。また、TI01n をトリガ入力で使用して、CR00n にタイマ 0 のカウント値をキャプチャします。これで、キャプチャごとに INTTM00n 割り込みが発生するので、そこで CR00n の値を読み出して、前回の値を減算すれば、1ms 間のパルス数が分かります。

[タイマの設定例]

78K0S/KA1+を例に、具体的なタイマ関係の設定を示します。

タイマ H1 関係の設定は、次のようになります。

TMHMD1 : **00110001（10110001）**（インターバル・タイマで出力許可）
CMP01　: **124**（125 カウント）
PM4.2　: **0**（兼用端子を出力に設定）
P4.2　: **0**（兼用端子の出力ラッチは 0 に）

タイマ 00 関係の設定は、次のようになります。

PRM00　:**01000011**（TI010 は立ち上がり、TI000 をカウント）
CRC00　:**00000001**（CR000 はキャプチャで使用）
TOC00　:**00000000**（出力機能は使用しない）
TMC00　:**00000100**（フリー・ランニング・モードで使用）
PM3　　:**xxxxxx11**（対応する端子を入力に指定する）

[プログラム例]

前記の初期設定が完了し、測定を開始するには、TMHMD1 レジスタのビット 7(TMHE1)をセットします。これで時間計測が開始するので、最初のキャプチャ割り込み(INTTM000)を待ちます。最初のキャプチャでは、前回の値が分からないので、キャプチャした値を読み出すだけにします。2 回目以降の割り込みで、今回のキャプチャ値から前回のキャプチャ値を減算して、1ms 間のパルス数を求めます。

C 言語で記述する場合は、まず次の宣言をする必要があります。ここでは、SFR 名称を使うことや割り込みを使用することの宣言、内部で使用する変数を宣言します。

```
#pragma SFR
#pragma NOP
#pragma EI
#pragma DI
#pragma INTERRUPT INTTM000 TM00INT
__interrupt void        TM00INT (void);
sreg static unsigned int OLDDATA;
sreg static unsigned int FREQ;
sreg static unsigned int tempbuff;
```

次に、使用するタイマ関係の初期設定を行います。

なお、ここでは簡単のために、ウォッチドッグ・タイマは停止させておくものとします。

```
/*
        TMH1 mode set
*/
        TMHMD1 = 0b00110001;    /* set interval timer mode      */
        CMP01 = 124;            /* set interval time            */
        PM4.2 = 0;              /* set TOH1 enable              */
        P4.2 =0;
/*
        TM00 mode set
*/
        PM3 = 0b11111111;
        PRM00 = 0b01000011;             /* set TI010 edge and count CLK */
        CRC00 = 0b00000001;             /* use CR000 to capture mode    */
        TOC00 = 0b00000000;             /* no output is used            */
```

初期設定のあと、次のように 1 回目の INTTM000 をポーリングし、最初のキャプチャ値を変数 OLDDATA に保存します。その後、割り込み可能にすれば、1ms ごとにキャプチャ動作が起動し、INTTM000 が発生します。

```
        TMC00 = 0b00000100;             /* start TM00 as free running   */
        TMHE1 = 1;                              /* start 1msec timer    */
        TMIF000 = 0;                    /* clear INTTM00                */
        while(TMIF000 == 0){
                NOP();                  /* wait for the first capture   */
        }
        OLDDATA = CR000;                /* get initial value            */
        TMIF000 = 0;
        TMMK000 = 0;                    /* enable INTTM000              */
        EI();                           /* enable interrupt             */
```

キャプチャ割り込みが発生すると、次の割り込み処理関数が起動され、前回のキャプチャ値との差を変数 FREQ にセットします。

```
__interrupt void        TM00INT (void){
        tempbuff = CR000;               /* read captured value  */
        FREQ = tempbuff - OLDDATA;      /* get pulse number     */
        OLDDATA = tempbuff;             /* change TM00 value    */
}
```

【注　意】
タイマ 0 でパルスをカウントする場合、そのロウ・レベル／ハイ・レベルの幅に制限があります。内蔵周辺クロックの 2 周期＋100/200ns 以上の幅が必要です。
周辺クロックが 8MHz のとき、電源電圧が 4V 以上なら 350ns 以上、2.7V 以上なら 450ns 以上のレベル幅の信号しかカウントできません。

タイマを使った複数パルス出力(1) [78K0、小ピン・マイコン共通]

Q1. 16 ビット・タイマの PPG 出力機能で、複数の出力を制御できますか。

A1. 16 ビット・タイマの出力は 1 本だけです。複数の出力を制御するには、PPG 出力ではなく、タイマで時間を計測して、割り込みを発生させてください。割り込み処理の中で、出力したいデータ・パターンをポートに出力すれば、実現できることがあります。

【コーヒー・ブレーク】
複数パルスの出力は、許される出力タイミングの変動幅によって実現方法が異なります。以下に代表的な処理方法を示します。なお、①や②が必要な場合は、マイコンの選択から注意が必要です。

①リアルタイム出力機能
タイミング変動が許されない場合に使用されます。ただし、この機能を内蔵したマイコンは限られていますし、同時に出力できる数も限られます。

②DMAの利用
次にタイミングを正確にできる方法で、タイマで DMA をトリガして、出力したいデータをメモリからポートに転送します。この方法は、バスの調停分でのタイミング変動があるだけです。

③タイマ割り込みによる処理
最も一般的な方法です。この方法は、割り込みの受け付け時間分の変動が存在します。この中でも一番変動の小さいのが、HALT モードを割り込みで解除する方法です。ただしこの場合、他の割り込みは使えません。
ベクタ割り込みを使用する方法が一般的ですが、実行中の命令により、応答時間が変動します。また、他の割り込みを処理中で、割り込みを受け付けられない期間があると、その時間も変動要因になります。

要求される時間精度に応じて、適したものを選択する必要があります。

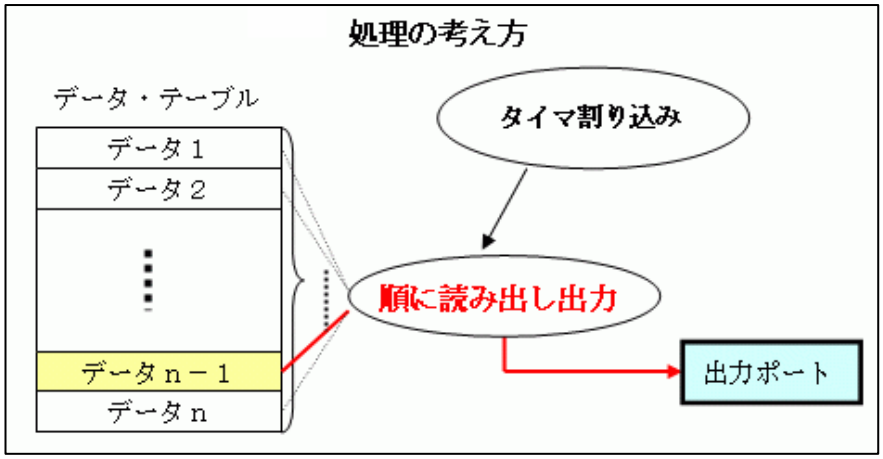


Q2. 数十 ms 程度の単位で出力を変更して、繰り返しが約 1 秒の場合、複数の出力制御はできますか。

A2. 時間の精度はあまり必要ないと考えられるので、割り込みで十分処理できます。

[処理の考え方]

インターバル・タイマ機能を用いて、数十 m 秒のインターバルで割り込みを発生させます。その処理の中で、1 秒分のデータ・パターンを順次出力していく方法が考えられます。この考え方を図に示します。タイマ割り込みが発生すると、データ・テーブルとして準備しておいた1秒分の出力データの中から、出力するデータを順に読み出して、出力ポートに出力します。



[実現方法]

78K0S/KY1+を例に、約 25ms 間隔でポート 4（8 ビット）に出力する場合の、具体的な実現方法を示します。8 ビット・タイマ H1 を用いて、8MHz の内蔵周辺クロックで約 25ms のインターバル割り込みを発生させます。このときのタイマ 80 の設定は、次のようになります。

TMHMD1：**01000000**（1/4096 に分周したクロックで動作）

CMP01　：**48**（49 カウント）

割り込み処理では、できるだけ処理時間が変動しないように、次に出力するデータを RAM に準備しておき、割り込みが起動されると、そのデータを出力ポートに出力します。その後、次に出力するデータを ROM から読み出し、RAM にセットします。このとき、テーブルの最後に達すると、テーブルの先頭に戻します（この部分は、処理時間が変動する可能性があります）。

[プログラム例]

C 言語で記述する場合には、まず次の宣言を行う必要があります。ここでは、SFR 名称を使うことや割り込みを使用することの宣言、内部で使用する変数を宣言します。

```
#define DATANUMBER 40
#pragma SFR
#pragma NOP
#pragma EI
#pragma DI
#pragma INTERRUPT INTTMH1 TMH1INT
__interrupt void        TMH1INT (void);
sreg static unsigned char NEXTDATA;
sreg static unsigned char OFFSET;
extern const unsigned char DATATABLE[DATANUMBER];
```

次に、使用するタイマ関係の初期設定を行います。ここでは、簡単にするために、ウォッチドッグ・タイマを停止させておきます。

タイマ 80 を 25ms 周期で割り込みが発生するように設定し、出力データの初期値を設定しておきます。

```
/*
        TMH1 mode set
*/
        TMHMD1 = 0b01000000;            /* set count clock      */
        CMP01 = 48;                     /* set interval time    */
        P4 = DATATABLE[0];              /* set initial data     */
        PM4 = 0;                        /* set port to output   */
        NEXTDATA = DATATABLE[1];
        OFFSET = 2;
```

あとは、タイマを起動するだけです。

```
        TMHE1 = 1;                      /* start 25ms timer     */
        TMIFH1 = 0;                     /* clear INTTMH1        */
        TMMKH1 = 0;                     /* enable INTTMH1       */
        EI();                           /* enable interrupt     */
```

インターバル・タイマ割り込みが発生すると、次の割り込み処理関数が起動され、出力を更新し、次のデータを準備します。

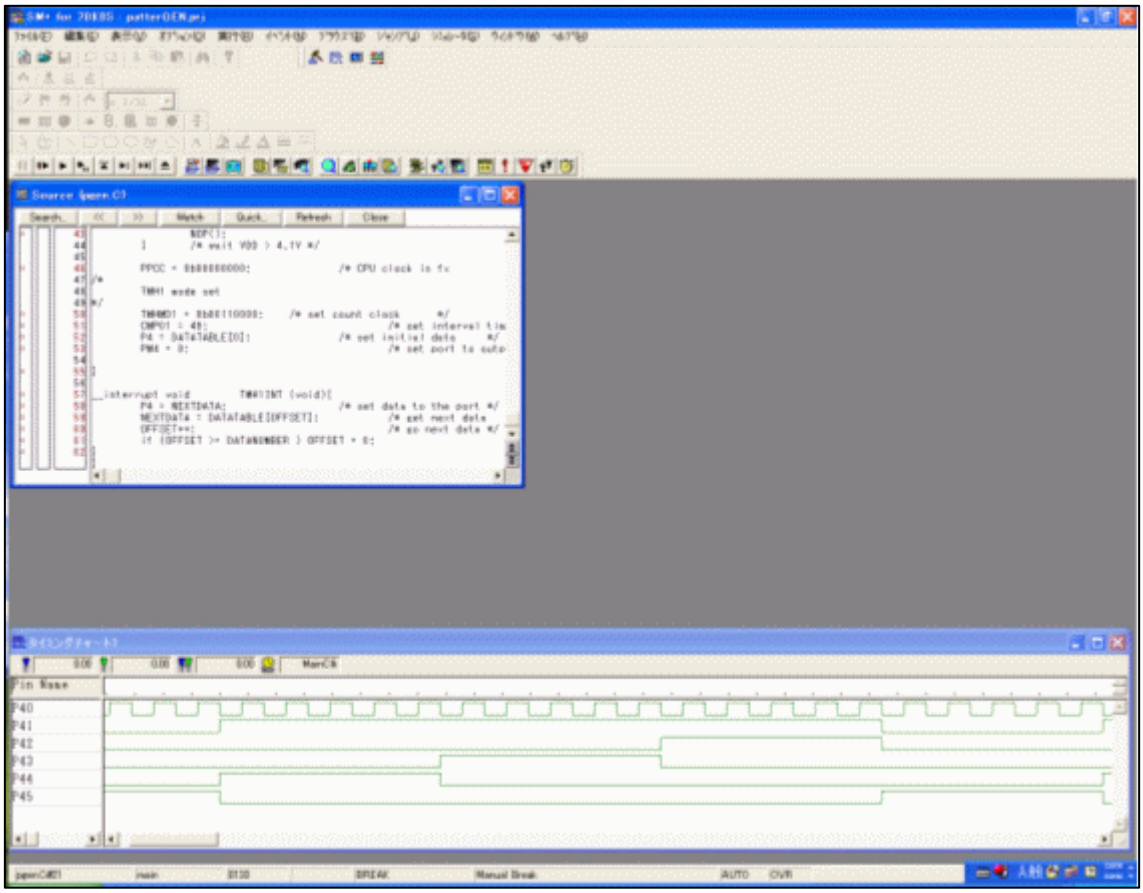
```
__interrupt void        TM H1INT (void){
        P4 = NEXTDATA;                  /* set data to the port */
        NEXTDATA = DATATABLE[OFFSET];   /* get next data        */
        OFFSET++;                       /* go next data */
        if (OFFSET >= DATANUMBER ) OFFSET = 0;
}
```

このプログラムで使用するデータは、次の例のように定義しています。

```
public  _DATATABLE
        cseg
_DATATABLE:
        DB      10100000b,10100001b,10100000b,10100001b
        DB      10100000b,10100001b,10100000b,10100001b
        DB      10100000b,10100001b,10010010b,10010011b
        DB      10010010b,10010011b,10010010b,10010011b
        DB      10010010b,10010011b,10010010b,10010011b
        DB      01001010b,01001011b,01001010b,01001011b
        DB      01001010b,01001011b,01001010b,01001011b
        DB      01001010b,01001011b,01000110b,01000111b
        DB      01000110b,01000111b,01000110b,01000111b
        DB      01000110b,01000111b,01000110b,01000111b
        END
```

このデータでは、ビット 0 が毎回反転し、ビット 1 は 75%がハイで 25%がロウ、ビット 2～5 は 25%がハイで 75%がロウの位相をずらした出力になります。この 6 ビットをシミュレータのタイミング・チャートで確認したときの画面イメージを次に示します（この画面では、25ms ごとでは時間が長すぎて波形の概要が見えないので、「TMHMD1 = 0b00110000;」として、時間を短縮しています）。

このイメージからも分かるように、このプログラムは 1ms ごと程度の速度の処理にも対応できます。



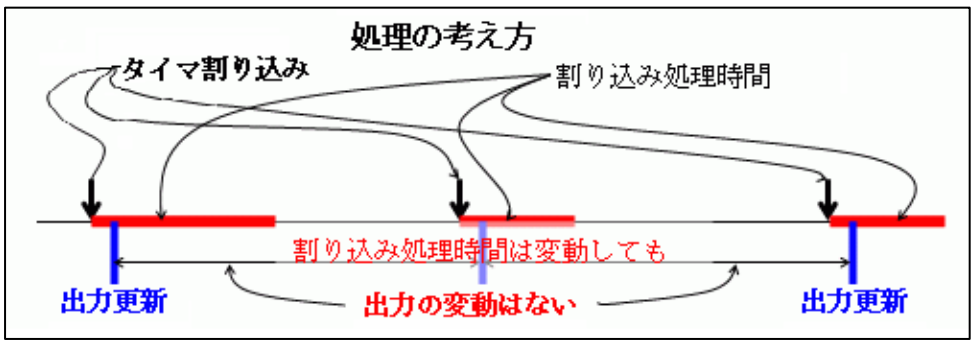
タイマを使った複数パルス出力(2) [78K0、小ピン・マイコン共通]

Q1. 繰り返しが 1 秒程度で、数十 ms 程度の単位で、周期やデューティを自由に変えられるパルスを複数出力できますか（タイミングのばらつきは、できるだけ小さく）。

A1. クロックの精度は、高速内蔵発振クロックで 5%以下、セラミック発振子で 0.5%以下となります（水晶振動子なら、さらに精度が高い）。これらを考慮すると、割り込み応答時間の変化範囲は、せいぜい数 μs 以下になりますので、割り込み禁止期間に注意すれば、割り込みを用いて十分処理できます。

[処理の考え方]

端子（ビット）ごとの出力データ・パターンは、**ON 期間**と **OFF 期間**で指定します（割り込みの回数で指定）。各端子には、この期間を実際に**カウントするための変数**を準備しておき、期間指定の 2 つの変数と合わせて、3 つの変数で制御します。実際にポートに出力する**データを準備**しておき、インターバル・タイマ機能を用いた数十 ms のインターバル**割り込み処理の先頭で、ポートに出力**します。出力後に、端子ごとに次のデータを生成して、割り込み処理を完了します（これにより、出力変化のタイミング変動をなくします）。



次のデータ生成では、前記の端子ごとのカウントを用い、それに応じて、端子単位に次の処理を行います。

- 0 なら何もしない（出力固定）
- 1 なら、次の期間をチェックする
  - 0 でなければ、その値をカウント変数に設定し、次のデータを反転する
  - 0 なら何もしない（反転を中止して出力固定）
- 0、1 以外なら、カウント変数を減ずる（出力はそのまま）

出力のディーティは、**ON 期間**と**OFF 期間**で決まりますが、それ以外に、**カウントするための変数**の初期値で出力の位相が決まります。また、出力の初期値は、次データの初期値で決めることができます。

[実現方法]

78K0S/KY1+を例に、約 25ms 間隔でポート 4（8 ビット）に出力する場合の、具体的な実現方法を示します。8 ビット・タイマ H1 を用いて、8MHz の内蔵周辺クロックから約 25ms のインターバル割り込みを発生させます。このときのタイマ 80 の設定は、次のようになります。

TMHMD1 :**01000000**（1/4096 に分周したクロックで動作）

CMP01　:**48**（49 カウント）

割り込み処理では、できるだけ処理時間が変動しないように、次に出力するデータをRAMに準備しておき、割り込みが起動されると、そのデータを出力ポートに出力します。その後、次に出力するデータを生成します（この部分は、処理時間が変動します）。

[プログラム例]

C 言語で記述する場合は、まず次の宣言を行う必要があります。ここでは、SFR 名称を使うことや割り込みを使用することの宣言、内部で使用する定数を定義します。

```
#define IDATA 0
#define I_ONTIME0 1
#define I_OFFTIME0 1
#define ITIME0 1
#define I_ONTIME1 2
#define I_OFFTIME1 2
#define ITIME1 1
#define I_ONTIME2 4
#define I_OFFTIME2 4
#define ITIME2 1
#define I_ONTIME3 8
#define I_OFFTIME3 8
#define ITIME3 1
#define I_ONTIME4 4
#define I_OFFTIME4 12
#define ITIME4 1
#define I_ONTIME5 12
#define I_OFFTIME5 8
#define ITIME5 1
#define I_ONTIME6 8
#define I_OFFTIME6 8
#define ITIME6 1
#define I_ONTIME7 8
```

```
#define I_OFFTIME7 8
#define ITIME7 8
#pragma SFR
#pragma NOP
#pragma EI
#pragma DI
#pragma INTERRUPT INTTMH1 TMH1INT
__interrupt void        TMH1INT (void);
```

IDATA は、出力の初期値の定義です。I_ONTIMEx（x=0～7）は ON 期間の長さ、I_OFFTIMEx は OFF 期間の長さ、ITIMEx は位相を指定するための最初の変化までの期間の定義です。

次に、各端子を制御するための変数を定義します。

```
sreg static unsigned char NEXTDATA = IDATA;
sreg static unsigned char ONTIME0 = I_ONTIME0;
sreg static unsigned char OFFTIME0 = I_OFFTIME0;
sreg static unsigned char COUNT0 = ITIME0;
sreg static unsigned char ONTIME1 = I_ONTIME1;
sreg static unsigned char OFFTIME1 = I_OFFTIME1;
sreg static unsigned char COUNT1 = ITIME1;
sreg static unsigned char ONTIME2 = I_ONTIME2;
sreg static unsigned char OFFTIME2 = I_OFFTIME2;
sreg static unsigned char COUNT2 = ITIME2;
sreg static unsigned char ONTIME3 = I_ONTIME3;
sreg static unsigned char OFFTIME3 = I_OFFTIME3;
sreg static unsigned char COUNT3 = ITIME3;
sreg static unsigned char ONTIME4 = I_ONTIME4;
sreg static unsigned char OFFTIME4 = I_OFFTIME4;
sreg static unsigned char COUNT4 = ITIME4;
sreg static unsigned char ONTIME5 = I_ONTIME5;
sreg static unsigned char OFFTIME5 = I_OFFTIME5;
sreg static unsigned char COUNT5 = ITIME5;
sreg static unsigned char ONTIME6 = I_ONTIME6;
sreg static unsigned char OFFTIME6 = I_OFFTIME6;
sreg static unsigned char COUNT6 = ITIME6;
sreg static unsigned char ONTIME7 = I_ONTIME7;
sreg static unsigned char OFFTIME7 = I_OFFTIME7;
sreg static unsigned char COUNT7 = ITIME7;
```

NEXTDATA は、次に出力する値を保持しておく変数です。ONTIMEx（x=0～7）は各端子の ON 期間を保持しておく変数、OFFTIMEx は OFF 期間を保持しておく変数です。この変数の値が COUNTx に転送されて、実際の時間計測に使用されます。ここで、各期間をすべて 8 ビットとしていますが、これで短い場合は、char を int に変更すれば 16 ビットに拡張できます。ONTIMEx や OFFTIMEx は参照するだけなので、途中でデューティや周期を変更する場合は、対応する端子のこれらの変数を変更するだけで済みます。

ここで、COUNTx に設定した初期値（ITIMEx）で、最初に変化するタイミング（位相）を指定しています。すぐ変更する場合は、1 を設定しておきます。0 を設定すると、出力は変化しません。

次に、使用するタイマ関係の初期設定をします。

なお、ここでは簡単のために、ウォッチドッグ・タイマは停止させておくものとします。

タイマ H を 25ms 周期で割り込みが発生するように設定し、出力データの初期値を設定しておきます。

なお、割り込み処理時間は、各端子で約 100 クロック、8 端子で 800 クロック程度と考えられます。そのため、そこまでは割り込み間隔を短くすることが可能です。

```
/*
        TMH1 mode set
*/
        TMHMD1 = 0b01000000;    /* set count clock      */
        CMP01 = 48;             /* set interval time    */
        P4 = 0;                 /* set initial data     */
        PM4 = 0;                /* set port to output   */
```

あとはタイマを起動するだけですが、最初に初期値を出力するため、割り込み要求フラグをセットしてすぐに割り込みを起動させます。

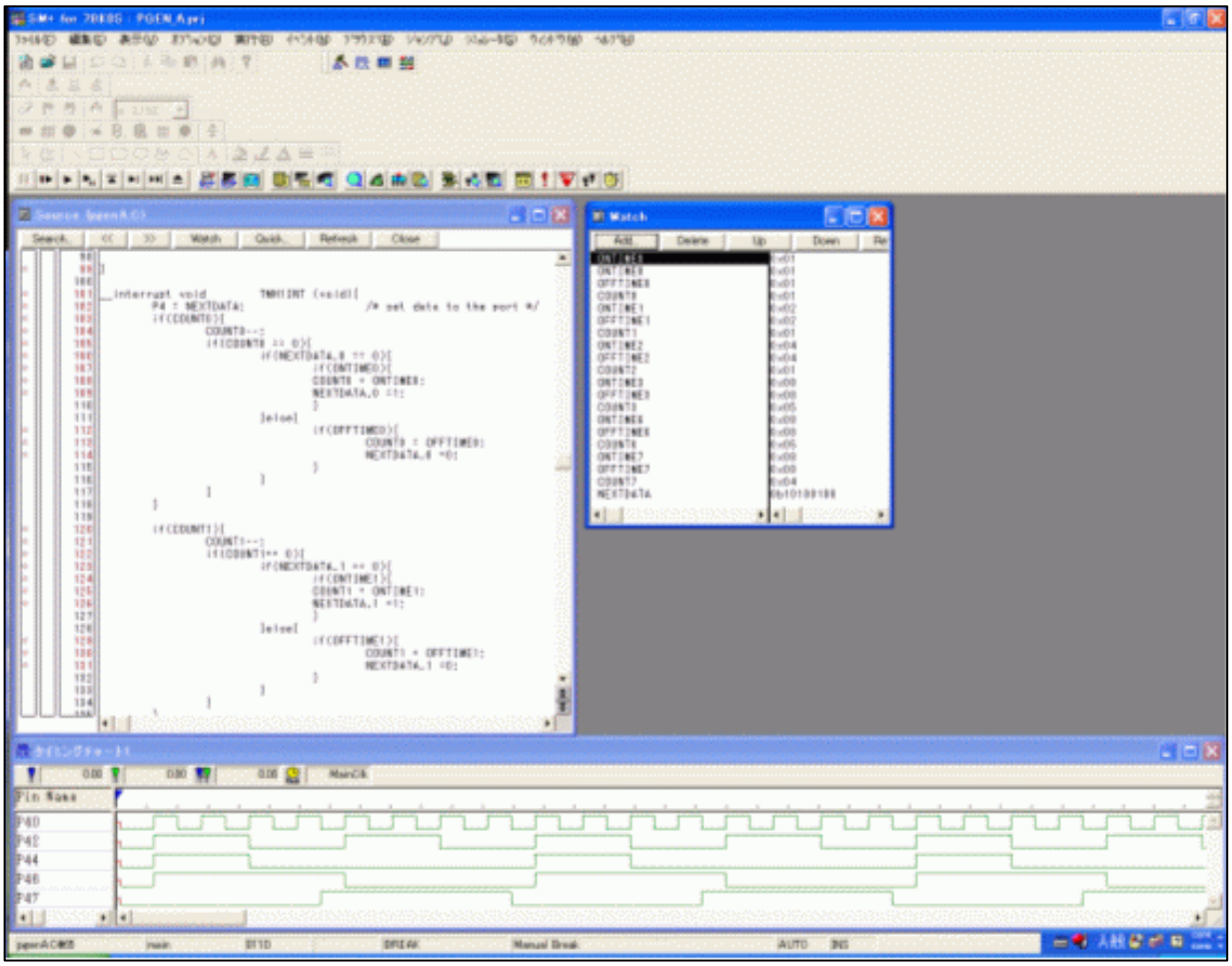
```
        TMHE1 = 1;              /* start 25msec timer   */
        TMIFH1 = 1;             /* set INTTMH1          */
        TMMKH1 = 0;             /* enable INTTMH1       */
        EI();                   /* enable interrupt     */
```

インターバルでの割り込みが発生すると、以下の処理が起動されます。ここでは、最初にポートに新しいデータが出力されます。以降は、各端子に対する次のデータを生成するために、1 ビットずつ同じ処理が繰り返されます。ビット番号が異なるだけなので、ここではビット 0 だけを示します。基本的な処理は[処理の考え方]で示しましたが、実際のプログラムでは、出力データが 0 か 1 かで処理が分かれます。

```
__interrupt void        TM H1INT (void){
        P4 = NEXTDATA;                  /* set data to the port */
        if(COUNT0){
                COUNT0--;
                if(COUNT0 == 0){
                        if(NEXTDATA.0 == 0){
                                if(ONTIME0){
                                COUNT0 = ONTIME0;
                                NEXTDATA.0 =1;
                                }
                        }else{
                                if(OFFTIME0){
                                        COUNT0 = OFFTIME0;
                                        NEXTDATA.0 =0;
                                }
                        }
                }
        }
/*
        ここに、上記の if(COUNT0){・・・・}の処理で、赤字部分を 1～7 に
        変更したものが並びます。
*/

}
```

このプログラムの実行結果をシミュレータのタイミング・チャートで確認したときの画面イメージを次に示します（この画面では、25ms ごとでは時間が長すぎて波形の概要が見えないので、「TMHMD1 = 0b00110000;」として、時間を短縮しています）。



P40 は毎回出力が反転し、P42 は 4 回ごとに反転、P44 は 12 回周期でデューティ 25%、P46 は 16 回周期でデューティ 50%、P47 は P46 の位相を変えたものとなります。

8 ビット・タイマ・カウンタ H のパルス発生への応用 [小ピン・マイコン共通]

Q1. 100kHz～0.1Hz を出力できるタイマ IC をマイコンに置き換えられますか。

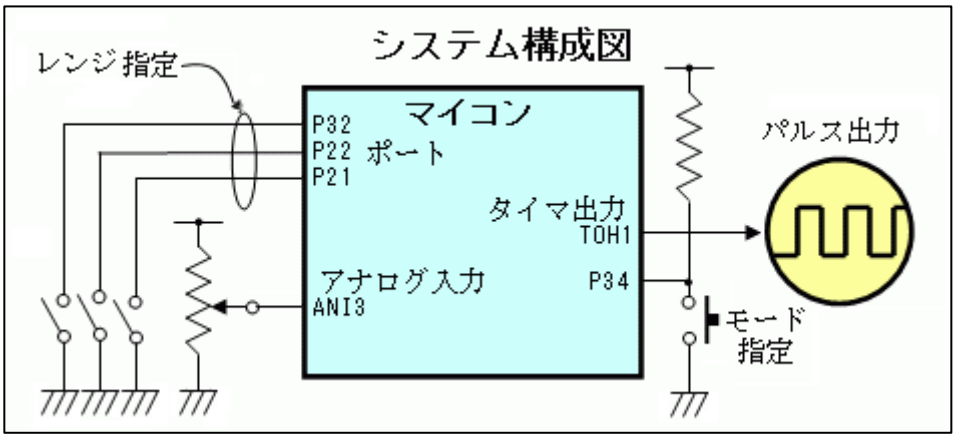
A1. タイマ IC の μPD5555 は、容易に幅広い周波数範囲のパルスを出力できます。その機能を小ピン・マイコンで実現することを考えてみます。プログラムは、8 ビット・タイマ・カウンタ H のパルス発生への応用プログラム例をご参照ください。

[概　要]

構成を簡単にするため、8MHz の高速内蔵発振器のクロックで動作させるものとします。このため、クロックの発振精度から 5%のばらつきが考えられますが、アナログ入力に半固定抵抗を接続して、トリミングすることで精度を高められるように考慮しています。ここでは、できるだけ短い周期から対応できるようにすることと、半固定抵抗での設定が微妙にならないように、3 ビットのポート入力により、出力パルスの周期を大きく 8 つのレンジに分割します。

レンジ	周　　期	分解能	処　理
1	1μs～17μs	0.125μs	ハードウエア
2	10μs～137μs	1μs	
3	96μs～1.112ms	8μs	
4	1ms～10ms	100μs	ソフトウエア
5	10ms～100ms	1ms	
6	100ms～1s	10ms	
7	1s～10s	100ms	
8	10s～100s	1s	

さらに、この中での細かな指定については、A/D コンバータで行います。これらの情報を定期的に確認して、その結果を反映できるようにします。



[処理方法]

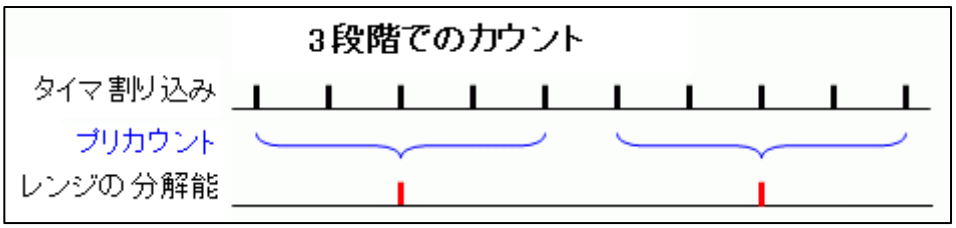
パルス出力方法を大きく 2 つに分けて考えます。1 つの方法は、短い周期に対応するために、タイマの出力機能を利用するものです。レンジ 1 はタイマの PWM 出力、レンジ 2 は方形波出力、レンジ 3 は PWM 出力を用いてパルス出力を行います。パルス出力中、CPU はタイマ割り込みを無視して、後述の変更タイミングのカウント動作だけを行います。

ハードウエア出力の設定一覧

周期	カウント・クロック	分解能	動作モード
1μs	f_{XP} (8MHz)	0.125μs	PWM 出力
10μs	f_{XP}/4 (2MHz)	1μs	方形波出力
96μs	f_{XP}/32 (250kHz)	8μs	PWM 出力

もう 1 つの方法は、長い周期の場合に使用するものです。タイマで基本的な間隔に割り込みを発生させて、それを**ソフトウエア**でカウントして、指定された間隔で**ポートを操作**してパルスを出力します。レンジ 4～8 は、この方法で実現します。この方法では、HALT モードでタイマ割り込みを待ち、割り込みで HALT を解除して、割り込み応答の変動を最低限に抑えてポートを操作します。時間計測は、3 段階に分けて処理します。最小の時間間隔は、タイマそのもののインターバル時間（カウント周期）です。このインターバル時間を計測するプリカウントを行うことで、インターバル時間と組み合わせて各レンジの時間差を吸収し、各レンジの分解能に相当する時間を得ます（実際は出力を反転する時間間隔なので、分解能の半分の時間）。

最後に、レンジ内での設定値に達するまでカウントします。各レンジで、A/D 変換結果の上位7ビット分の値を用いて、最小と最大の比を 1 対 13 程度に設定して、分解能の 10～135 倍の時間を指定できるようにしています。



ソフトウエア出力の設定一覧

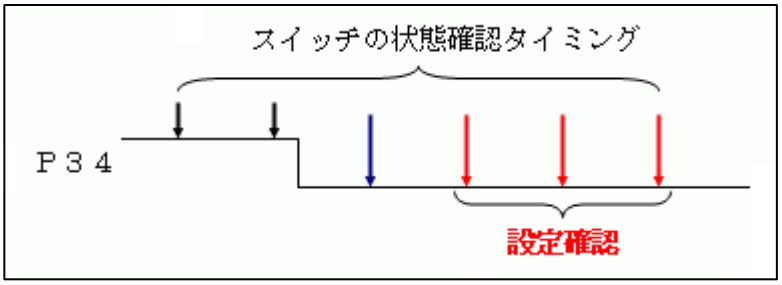
周期	カウント・クロック	カウント周期	プリカウント値	分解能
1ms	f_{XP}/4 (2MHz)	50μs	1	50μs
10ms	f_{XP}/16(500kHz)	0.5ms	1	0.5ms
100ms	f_{XP}/64(125kHz)	1ms	5	5ms
1s	f_{XP}/4096(2kHz)	50ms	1	50ms
10s	f_{XP}/4096(2kHz)	0.1s	5	0.5s

指定回数をカウントしたら、ポート出力を反転させ、ポート操作の回数をカウントして、設定の変更タイミングをチェックします。ポート操作後に、設定の変更を確認することで、変更確認の時間がパルス出力に影響しないようにしています。

ハード／ソフトいずれの場合も、パルス出力にベクタ割り込みは使用しません。

[設定モードの確認]

ハードウエア出力の場合、設定モードの確認タイミングは、ソフトウエア・タイマで時間を計測して決めます。18 クロックのループを 2^{16} 回ループさせて、約 150ms ごとにモード指定スイッチ（P34）の内容を確認します。
ソフトウエア出力の場合は、出力パルスを反転させた直後に、モード指定スイッチの内容を確認します。
モード指定スイッチの状態が、**2 回連続で ON（ロウ・レベル）**（レベル検出）なら、新しい設定の確認をします。



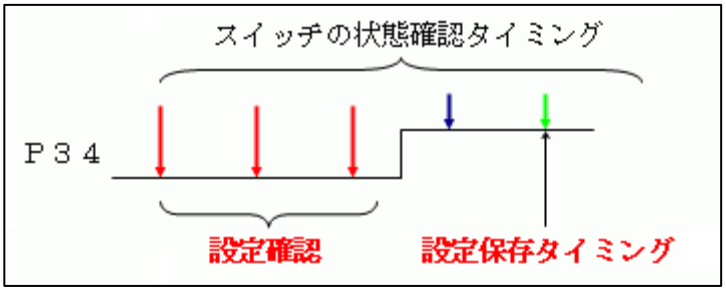
確認処理では、A/D コンバータを起動して 4 回の変換を行い、その平均を用いてノイズの影響をなくします。
ソフトウエア出力の場合、この間にタイマ割り込みが発生してもいいように、タイマ割り込みをベクタで処理して、出力タイミングの計測だけを行います。ソフトウエア出力処理は 1ms 周期（出力反転は 0.5ms ごと）以上なので、基本的に次の出力反転処理までの時間はかかりません。
以降は、モード指定スイッチの状態が 2 回連続で OFF（ハイ・レベル）になるまで、この処理を繰り返します。

[設定モードの変更]

4 回の A/D 変換の完了タイミングで、レンジ設定スイッチの状態も確認して、新しい設定値を取り込みます。その値（周期とその中での値）がこれまでと同じなら、何も変更しないでそれまでの処理を継続します。

ソフトウエア出力処理でレンジが同じ場合は、次のカウント周期からカウント値を変更すれば、出力の乱れがなくなります。ソフトウエア出力処理でレンジが切り替わった場合は、いったんタイマを停止して再設定するので、その処理分だけ変更時の周期のばらつきが発生してしまいます。これは、プリカウントを 16 ビットで行うと解消できますが、長い周期の場合に 2000 倍の頻度でタイマ割り込みが発生することになり、動作電流が大きくなってしまうことが考えられます。ここでは、そのようなことを考慮しています。

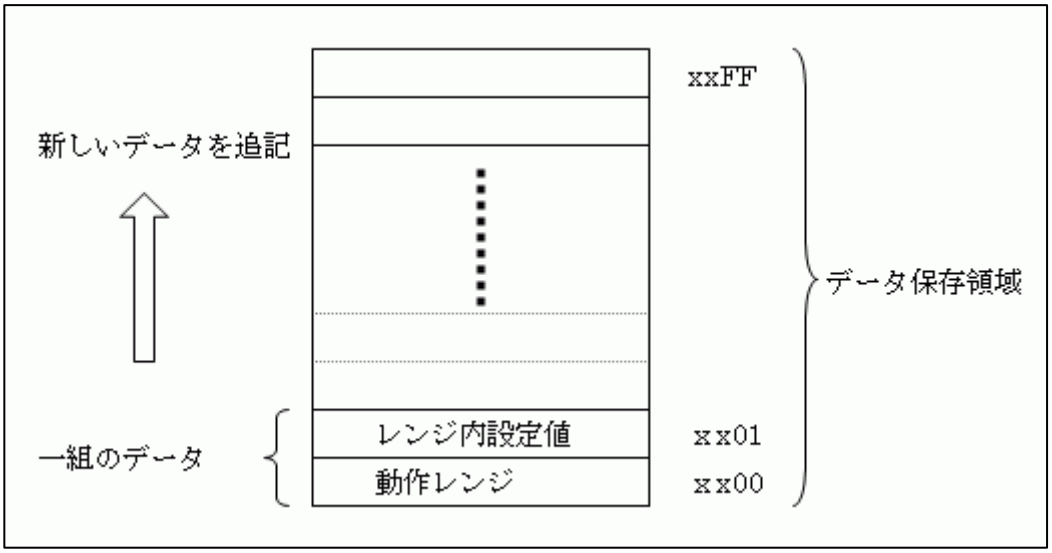
[設定されたモードの保存]



ロウ・レベルのあとに、**ハイ・レベルが 2 回続く**（エッジ検出）と、設定された内容の**保存タイミング**となります。**内蔵フラッシュの専用領域**に保存されている従来の（最新の）条件と、新しい設定値を比較して、変更されていたら、**その専用領域に新しい情報を保存**します。このとき、内蔵フラッシュに対してセルフ・プログラミングを行います。これにより、電源を切っても、次回は新しく設定した条件で動作するようになります。このため、フラッシュ・メモリの最後の 256 バイト分を EEPROM のように使います。

なお、ここでは書き換えやクリア中のエラーは考慮しません。内蔵フラッシュの ROM 容量が許せば、領域を複数準備して切り替えたり、エラー処理を追加したりすることも考えられます。

更新により、データ保存領域に新たなデータを書きこむ領域がなくなると、領域をクリアして、その最初から書き込みます。



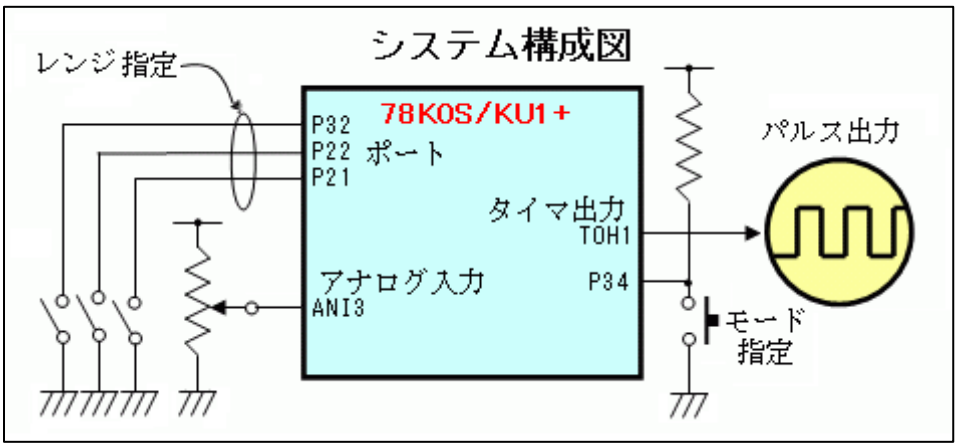
8 ビット・タイマ・カウンタ H のパルス発生への応用プログラム例 [小ピン・マイコン共通]

ここでは、「8 ビット・タイマ・カウンタ H のパルス発生への応用」で説明した処理の、具体的なプログラム例を紹介します。

[概　要]

指定された周期のパルスを出力するだけの単純な機能なので、できるだけ小さなマイコンで実現することを考えます。具体的なターゲットとして、10 ピンで内蔵 ROM が 1K バイトの μPD78F9200 での実現を例に取り上げます。電源電圧は 5V（4V 以上）として、高速内蔵発振器の 8MHz での動作とします。出力範囲は、次の 8 つのレンジで 1μs～100s をカバーします。

レンジ	周　　期	分解能	処　理
1	1μs～17μs	0.125μs	ハードウエア
2	10μs～137μs	1μs	
3	96μs～1.112ms	8μs	
4	1ms～10ms	100μs	ソフトウエア
5	10ms～100ms	1ms	
6	100ms～1s	10ms	
7	1s～10s	100ms	
8	10s～100s	1s	



[プログラム概要]

プログラムは、大きく分けて 10 の部分で構成されます。プログラム容量のうち、設定データの保存用に 1 ブロックを使用し、合計でも 1K バイトに収めるために、すべてアセンブラ記述とします。また、ベクタ・テーブル領域とオプション・バイトの間にもプログラムを配置して、できるだけメモリにむだが発生しないようにします。

(1) オプション・バイト部

動作クロックとして高速内蔵発振クロックを使用し、P34 端子はリセットでなくポートに指定します。また、ウォッチドッグ・タイマは使用しないので、低速内蔵発振器を停止可能に指定します。また、プロテクト・バイトはすべて書き込み／消去許可に設定しておきます。

(2) 初期化部

低電圧検出回路を用いて、電源が所定の電圧（4V）以上になるのを待ち、内蔵フラッシュのデータ領域に記録されている最新の設定データに基づいて、出力パルスを指定します。

(3) パルス出力部

指定された条件から、出力パルスの周期に応じた出力処理をします。これは大きく 2 つの処理に分けられます。

1 つは、1ms 以下の出力に対応するために、タイマ H の出力（**ハードウエア出力**）を直接使用するものです。プログラムでは、タイマを設定したあと、設定変更の確認タイミングを待つカウント処理をします。この間、タイマ H は指定された方形波出力／PWM 出力で動作し、割り込み要求を発生しますが、タイマ割り込みをマスクしておき、割り込み処理はしません。

もう 1 つの処理は、タイマ H をインターバル・タイマとして使用し、その割り込み回数をカウントして、出力を変化させるタイミングを得る（**ソフトウエア出力**）ものです。割り込み回数のカウントは 2 段階で処理します。これは、レンジごとの分解能（周期の最小時間の 1/10）を作る部分と、レンジの中での設定値を指定する部分に分けて、時間を計測するためです。これにより、1ms～100s の広い周期に対応できるようにします。出力を反転させたあとは、設定変更を確認するタイミングを計測します。

(4) 設定変更確認部

P3.4 に接続されたスイッチの状態を定期的に確認して、変数に取り込んでいきます。このプログラムでは、**2 回連続してロウ・レベル**と確認したら、設定の変更が有効と判断します（レベル検出）。また、**ロウ・レベルのあとに 2 回連続したハイ・レベル**を検出すると、設定変更結果のフラッシュ書き込み処理と判断します（エッジ検出）。

(5) 設定確認部

A/D コンバータを用い、半固定抵抗で入力される電圧値を読み取ることで、設定確認をします。A/D 変換は、割り込みで処理します。このとき、ノイズの影響を少なくするため、4 回の変換結果の合計を 1/8 にします。こうして得られた 0～127 の値を周期指定に使用します。同時に、レンジを指定するスイッチの内容も読み取ります。その結果を以前の設定と比較して、変化がなければゼロ・フラグで確認できるようにします。なお、この処理の期間だけベクタ割り込みを使用します。

(6) 設定の解析部

変更された新しい設定から、具体的なタイマ H の設定やソフトウエア出力のパラメータ設定をします。ソフトウエア出力では、レンジが同じなら、次の出力反転以降に新しい設定が反映されます。それ以外の変更では、いったんタイマ H を停止し、新しい設定に変更して動作を開始します。このため、**切り替え時に出力パルスの乱れが発生**してしまいます。

(7) フラッシュへ書き込み処理部

P3.4 に接続されたスイッチが OFF になると、フラッシュへ変更内容を書き込みます。以前の設定値と新しい指定値を比較して、変化がなければ何もしないで戻ります。このように、フラッシュへの不要な書き込みをしないことで、フラッシュの書き換え回数をできるだけ少なくします。フラッシュへの書き込み内容は、**レンジ情報**（0000xxx0 の 3 ビット分）と**レンジ内での指定値**の合計 2 バイトです。

書き込みは、フラッシュに対するセルフ・プログラミングを用いた、簡易的な EEPROM のエミュレーションで行います。簡素化のため、エラー処理は行いません。

(8) セルフ・プログラミング・ライブラリ部

小ピン・マイコンの内蔵フラッシュへの書き込みを行うためのライブラリで、基本的にユーザーズ・マニュアルに記載されたプログラムそのものです。

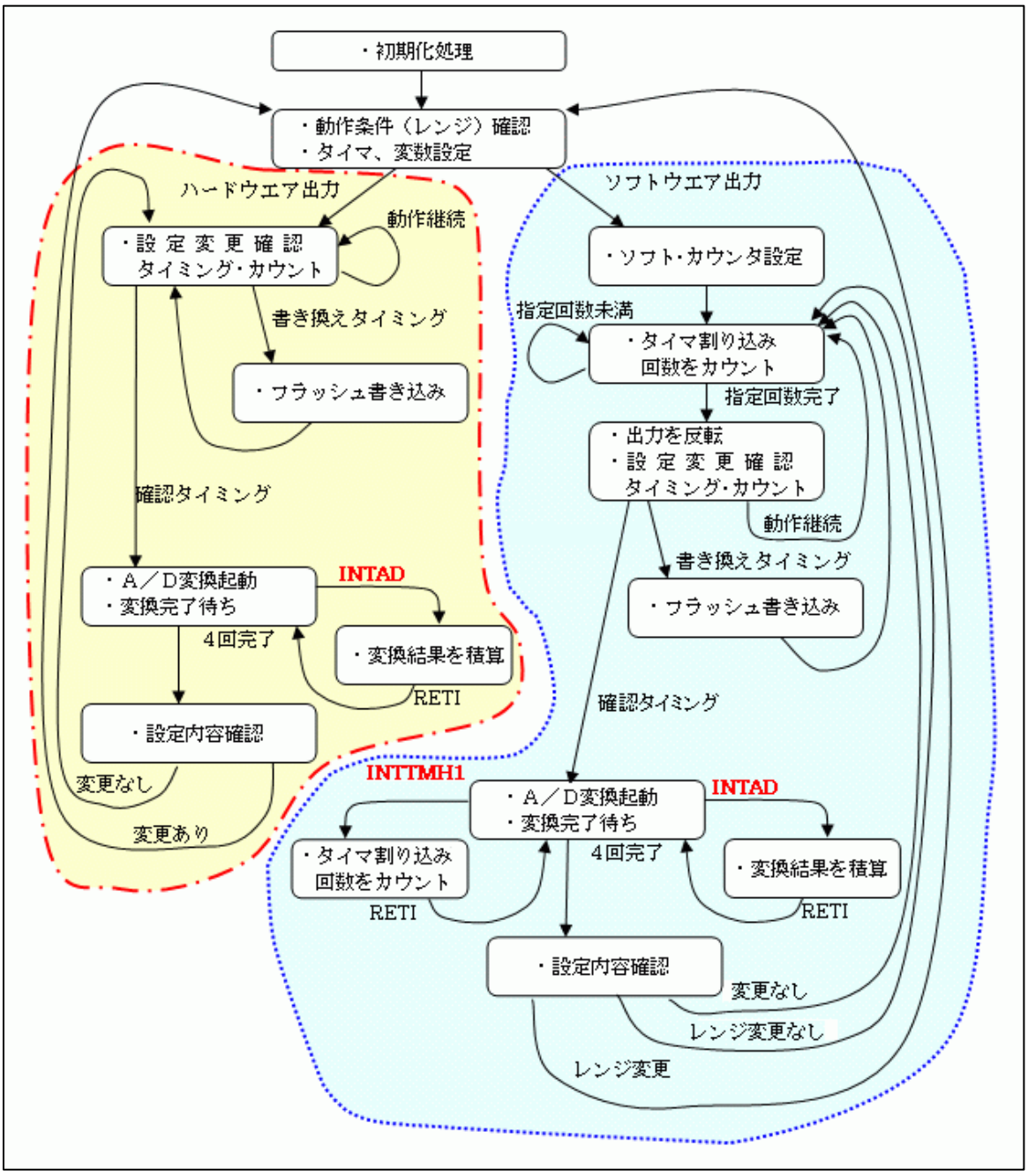
(9) タイマ H 割り込み処理部

設定確認処理の期間だけ有効になり、ソフトウエア出力処理のタイミング・カウントだけ行います。出力の直接の操作には使用しません。

(10) A/D 変換割り込み処理部

設定確認処理で、半固定抵抗を介して入力される電圧値を変換します。変換結果は、作業用の変数に積算していきます。

次に概略の状態遷移を示します。



[オプション・バイト]

ここで使用するオプション・バイトは、次のように設定します。

```
@@OPTB  CSEG    AT      0080H
        DB              10010100b       ; オプション・バイト
        DB              11111111b       ; プロテクト・バイト
```

[初期化処理]

POC 解除で起動して、パルス出力の準備を行います。

①動作準備部

スタック・ポインタ設定、ポート設定、ウォッチドッグ・タイマ停止、アナログ入力の設定を行い、低電圧検出機能を用いて、電源電圧が 4V 以上になるのを待ちます。

```
CSEG
;*********************************************************************
;       ハードウエアの初期化
;
;   ポートやアナログ入力の設定、電源電圧を 4V 以上に設定
;*********************************************************************

START:
        MOVW    AX,#STACKTOP
        MOVW    SP,AX                   ; スタックの設定
        MOV     PMC2,#0001000b          ; ANI3 だけアナログに設定
        CLR1    PM2.0                   ; タイマ出力用端子を出力に設定
        MOV     PU2,#00000110b          ; 入力ポートはプルアップを設定
        MOV     PU3,#00000100b          ;
        MOV     PU4,#00001001b          ; プルアップを設定
        MOV     LVIS,#00000000b         ; LVI=4.0V
        MOV     LVIM,#10000000b         ; 低電圧検出開始
        MOV     PCC,#00                 ; クロックを 1/4 分周に設定
        MOV     WDTM,#01110111b         ; ウォッチドッグ・タイマ停止
        SET1    LSRSTOP                 ; 低速発振器停止
        MOV     ADS,#00000011b          ; アナログ入力に ANI3 を選択
        MOV     A,#50
PONLOOP:
        DEC     A
        BNZ     $PONLOOP                ; 0.2ms 待つ
PONLOOP2:
        BT      LVIF,$PONLOOP2          ; 電圧が 4.1V 以上まで待つ
        MOV     PPCC,#00                ; クロックを fx に設定
```

②パルス出力設定部

フラッシュに格納されている最新の設定値から動作条件を確認し、対応したモードでの設定を行い、パルス出力動作を開始します。

```
        MOV     COUNTCHK,#00001110b     ; 変更タイミング初期化
;
;       ブロック 3 の有効データ・スキャン
;
        MOV     H,#EEPROM_BLOCK+1
        MOV     L,#0                    ; HL にスキャン用ポインタ設定
SCANLOOP:
        DECW    HL
        DECW    HL
        MOV     A,[HL]                  ; レンジを読み出し
        CMP     A,#0FFH                 ; 未書き込みかチェック
        BZ      $SCANLOOP               ; 次のアドレスを確認
        AND     A,#00001110b            ; 未使用ビットのマスク
        XCH     A,X                     ; レンジを X レジスタにセーブ
        MOV     A,[HL+1]                ; レンジ内での値を読み出し
        AND     A,#01111111b            ; 未使用ビットをマスク
        MOVW    FREQ,AX                 ; 設定値を作業領域にコピー
        XCHW    AX,HL                   ; 最新データのアドレスを
        MOVW    _Rd_Address,AX          ; 作業領域に保存
;
;  新しい設定でのタイマ起動
;
RESTART:
        MOV     A,FREQ+1                ; 出力のパルス周波数データ読み出し
        ADD     A,#10                   ; 設定値に 10 のオフセットをつけて、
        MOV     COUNTH,A                ; カウント値に設定
        MOV     B,A
;
;  動作パラメータ（レンジや設定値）を読み出して、その値に応じた
;  動作パラメータで処理ルーチンに分岐
;
        CALL    !SETTMH1MODE            ; 設定値から設定用パラメータに変換
        BC      $SOFTTIME               ; ソフトウエア処理なら分岐
```

[ソフトウエア出力処理]

1ms 以上の周期のパルスに対して、ソフトウエア出力処理（前記の状態遷移図で、青い破線で囲まれた部分）を行います。ソフトウエア出力処理では、基本となるインターバルで、タイマ H の割り込みを発生させ、その割り込みを HALT 状態の解除に使用して、時間測定を行います。タイマ H のインターバル時間は、次のようになります。

周期	カウント・クロック	タイマ H インターバル時間
1ms	f_{XP}/4 (2MHz)	50 μs
10ms	f_{XP}/16(500kHz)	0.5ms
100ms	f_{XP}/64 (125kHz)	1ms
1s	f_{XP}/4096 (2kHz)	50ms
10s	f_{XP}/4096 (2kHz)	0.1s

出力パルスは、指定された時間の 1/2 で、ハイ・レベルとロウ・レベルを切り替える必要があります。また、各周期の 1/10 の分解能を確保することを考慮すると、タイマ H で発生するインターバルは、周期の値の 1/20 に設定する必要があります。100ms や 10s の周期では、1/100 の時間しか生成できないので、5 カウントして分解能に相当する時間を得ます。そのため、次の 2 つのソフトウエアによるカウンタを使用します。

	メイン・カウンタ	カウント値保持レジスタ
プリカウンタ	PRECOUNT	PRECOUNTH
指定時間(周期)カウンタ	B レジスタ	COUNTH

また、設定変更の確認のためのタイミング計測用に、出力を反転させた回数を C レジスタでカウントします。これにより、設定変更スイッチのチャタリングの影響をなくします。

次に、実際のプログラム例を示します。

①ソフト・カウンタ設定

まず、カウント値保持レジスタから実際にカウントを行うレジスタに値をコピーします。その後、タイマ H を起動して割込みマスクを解除します(割り込みは禁止状態です)。

```
SOFTTIME:
        MOV     A,COUNT2H           ; チェック回数を読み出し
        MOV     C,A                 ; カウント・レジスタに設定
        MOV     A,PRECOUNTH         ; 設定値を読み出す
        MOV     PRECOUNT,A          ; 作業用にコピー
        MOV     A,COUNTH            ; 周期データ読み出し
        MOV     B,A                 ; カウンタにセット
        SET1    TMHE1               ; タイマの起動
        CLR1    TMIFH1              ; 割り込みフラグをクリア
        CLR1    TMMKH1              ; タイマ割り込みのマスク解除
        BR      $SOFTTIME2          ; 出力を反転しないでスタートする
```

②出力反転

タイマ H からの割込み要求を HALT 状態で待ち、**割り込み発生で HALT 状態が解除されると、すぐに出力を反転します。**その後、設定変更の確認カウンタを更新し、確認タイミングなら、設定モードの確認(CHKDATAIN をサブルーチン・コール)します。そこでは、A/D 変換を 4 回行って平均化処理をしますが、次の出力反転タイミングまでは最低でも 500μs 弱あるので、この処理では出力に影響を与えません。

スイッチの状態が変更なしなら、WAITLOOP に分岐して処理を継続します。変更モードなら、設定内容の確認(SETMAIN)に分岐します。変更モードの完了なら、変更内容を確認して、設定値が変更されていれば、フラッシュに新しい設定値を書き込み(サブルーチン CHANGEEXIT で処理)、処理を継続します。

フラッシュ書き込みでは全割り込みをマスクするので、書き込みから戻ると、タイマ H の割り込みマスクを解除します。なお、フラッシュ書き込みの最中はパルス出力処理ができないので、**パルスの周期が延びることがあります**。

```
;
;   前処理が完了して、パルス出力開始
;
SOFTTIME1:
        HALT                            ; タイマ割り込み待ち
        XOR     P2,#00000001b           ; 出力の反転
SOFTTIME2:
        DBNZ    C,$WAITLOOP             ; チェック・タイミング以外なら継続
        MOV     A,COUNT2H               ; チェック回数を読み出し
        MOV     C,A                     ; 再設定
        CALL    !CHKDATAIN              ; 設定モードか確認
        BNZ     $WAITLOOP               ; 変更なければ分岐
        BC      $SETMAIN                ; 変更モードなら分岐

;*********************************************************************
;       変更データのフラッシュへの書き込み
;
;   設定変更モードで設定された内容がフラッシュの最新のデータと同じか
;   確認し、変更されていれば、新しい設定値をフラッシュに書き込み
;*********************************************************************
        CALL    !CHANGEEXIT             ; 変更したデータの書き込み処理
        CLR1    TMMKH1                  ; タイマ割り込みのマスク解除
;
;   ここから下に抜ける場合、次の HALT はすぐ解除される
```

③インターバル時間の計測

ここでは、タイマ H の割り込みをカウントして、必要な時間を待ちます。いったん **PRECOUNT** で割り込みの回数をカウントして、必要な分解能分のタイミングを生成します。分解能のタイミングを **B** レジスタでカウントすれば、出力するパルスの周期の 1/2 の時間間隔（ロウ・レベルの期間／ハイ・レベルの期間）が分かります。②で割り込み要求フラグ（**TMIFH1**）をクリアしないので、出力を反転する間隔は **PRECOUNTH × COUNTH** となります。

```
;
;   ここで、インターバル・タイマ割り込みの回数をカウント
;   割り込みで出力を反転するまで待ち、次で反転するとき、前述の処理に分岐
;
WAITLOOP:
        HALT                            ; タイマ割り込み待ち
        CLR1    TMIFH1                  ; 割り込み要求フラグをクリア
        DBNZ    PRECOUNT,$WAITLOOP      ; プリカウント
        MOV     A,PRECOUNTH             ; 設定値を読み出す
        MOV     PRECOUNT,A              ; 作業用にコピー
        DBNZ    B,$WAITLOOP             ; タイム・アップしなければループ
        MOV     A,COUNTH                ; 周期データ読み出し
        MOV     B,A                     ; カウンタにセット
        BR      $SOFTTIME1              ; 次の割り込みで出力反転へ
```

[ハードウエア出力処理]

1ms 未満の周期では、ソフトウエア処理は間に合わないので、タイマ H の出力を利用して、パルスを出力します。この場合、カウント・クロックは 2 のべき乗で分周されるので、分解能は周期の 1/10 にはできません。できるだけ分解能を近づけるため、タイマ H の動作モードを PWM 出力と方形波出力で使い分けています。

①タイマ H の設定処理

各レンジに対するタイマ H の設定は、次のプログラムのようになります。ここで、変数 FREQ と FREQ+1 は、レンジ情報とそのレンジでの周期指定データ(0～127 の 7 ビット)が入っています。設定プログラムがレンジごとに分かれているのは、できるだけ処理時間を短くするためです。それでも、タイマ H を停止して、モード設定(下記のプログラムでは赤で示す命令)から再起動(青で示す命令)まで 12 クロックはかかります。そのため、同じレンジで設定値(周期)を変更するだけでも出力が乱れます。

```
;*********************************************************************
;         ハードウエア処理の場合のタイマ H の設定
;   TMH1 のモード設定（動作は停止）して、コンペアレジスタに周期に
;   応じた値を設定
;   キャリー・フラグをクリアして戻るので、ソフト処理と区別可能
;
;   CASE1（1μs 周期：0.125μs 分解能）
;   8MHz クロックの PWM モードで使用するので、分解能は 1/8μs
;*********************************************************************
CASE1:
        MOV     A,FREQ+1                ; 新しい設定値を読み出す
        ADD     A,#7                    ; 7(1μs)～134(17μs)
        MOV     TMHMD1,#00001001b       ; fXP 動作、PWM モード
        MOV     CMP01,A                 ; 周期の設定
        RORC    A,1                     ; デューティを約 1/2 に
        MOV     CMP11,A
        SET1    TMHE1                   ; 新しい設定値でタイマ起動
        CLR1    CY
        RET
;
;   CASE2（10μs 周期：1μs 分解能）
;   2MHz クロックのインタイバル・タイマで使用するので、インターバルは
;   1μs(タイマはハイ／ロウの期間を示すので、周期は倍)
;
CASE2:
        MOV     A,FREQ+1                ; 新しい設定値を読み出す
        ADD     A,#9                    ; 9(10μs)～136(137μs)
        MOV     TMHMD1,#00010001b       ; fXP/4 動作、インターバル・タイマ
        MOV     CMP01,A
        SET1    TMHE1                   ; 新しい設定値でタイマ起動
        RET
;
;   CASE3（100μs 周期：8μs 分解能）
;   125kHz クロックの PWM モードを使用するので、分解能は 8μs
;
```

```
CASE3:
        MOV     A,FREQ+1                ; 新しい設定値を読み出す
        ADD     A,#11                   ; 11(96μs)～138(1112μs)
        MOV     TMHMD1,#00111001b       ; fXP/64 動作、PWM モード
        MOV     CMP01,A                 ; 周期の設定
        RORC    A,1                     ; デューティを約 1/2 に
        MOV     CMP11,A
        SET1    TMHE1                   ; 新しい設定値でタイマ起動
        CLR1    CY
        RET
```

②出力中のソフトウエア処理

この動作でパルス出力中のソフトウエアは、単に設定変更の確認タイミングを待つだけの処理です。割り込みはマスク状態で動作し、HL レジスタでのソフトウエア・タイマを使用して確認タイミングを待ちます。確認タイミングになると、サブルーチン(CHKDATAIN)をコールして、設定スイッチの状態を確認します。タイマ H がハードウエア機能でパルス出力をしているので、ソフトウエアの実行時間の制限はありません。スイッチの状態が設定変更でなければ、処理を継続します。変更完了であれば、変更内容をフラッシュに書き込みます。設定変更なら、そのための処理(HWTIMERCHANGE)に分岐します。

```
;*****************************************************************
;
;   タイマ H の出力を直接使用する
;
;*****************************************************************
CASE1to3:
        CLR1    P2.0                    ; 兼用ポートをクリア
        SET1    TMHE1                   ; タイマの起動
        SET1    TMMKH1                  ; タイマ H 割り込みはマスクしておく
;
;       パルス出力
;
HWTIMERLOOP0:
        MOVW    HL,#0
HWTIMERLOOP1:
        DECW    HL                      ; 更新周期のカウント
        MOV     A,H
        OR      A,L
        BNZ     $HWTIMERLOOP1           ; 更新周期でなければ継続
        CALL    !CHKDATAIN              ; 通常動作かの確認
        BNZ     $HWTIMERLOOP0           ; 変更なければ処理継続
        BC      $HWTIMERCHANGE          ; 設定変更なら分岐
;*****************************************************************
;       変更データのフラッシュへの書き込み
;
;   設定変更モードで設定された内容が、フラッシュの最新のデータと
;   同じか確認して、変更されていれば、新しい設定値をフラッシュに
;   書き込み
;*****************************************************************
CALL    !CHANGEEXIT                     ; 変更したデータの書き込み処理
        BR      $HWTIMERLOOP0           ; 処理継続へ
```

③設定の変更処理

設定変更の場合、サブルーチン（SETMAINSUB）で指定内容を確認します。指定内容に変更がなければ、処理を継続します。変更があれば、指定内容を反映させるために、動作条件確認処理（RESTART）に分岐します。

```
HWTIMERCHANGE:
;
;       変更の確認処理
;
        CALL    !SETMAINSUB             ; 変更の確認
        BZ      $HWTIMERLOOP0
;
;       レンジ変更処理／ハードウエア変更処理
;
        BR      $RESTART
```

[設定変更確認処理]

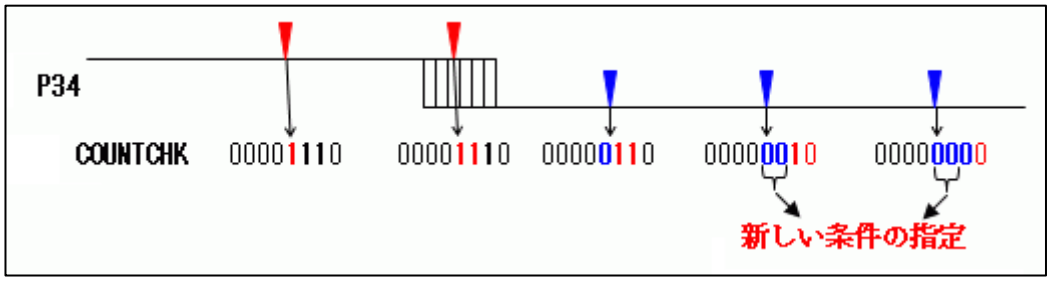
動作条件の設定変更は、P34 に接続されたスイッチが ON（P34 がロウ・レベル）の場合に行います。そのために、ある間隔で P34 を確認し、その変化を検出します。スイッチのチャタリング対策のため、同じ状態が 2 回連続したことを確認したら、有効な指定とします。そのため、P34 の状態を保持しておき、次の 3 つの状態を判定します。

- ・2 回連続でロウ・レベル　：新しい条件の指定
- ・ロウ・レベルのあと、2 回連続でハイ・レベル　：指定完了
- ・その他　：通常動作

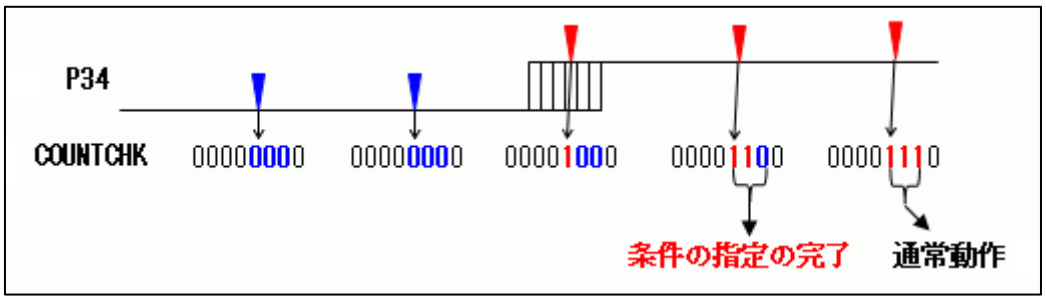
新しい条件の指定では、指定内容を確認するために A/D コンバータを起動し、半固定抵抗器で設定された電圧を 4 回読み取って平均化します。同時に、レンジを指定するスイッチの状態も読み取り、レンジを確認します。

①設定変更指示の確認処理

P34 の状態を確認するプログラムを示します。P34 の状態を変数 COUNTCHK に保存して、新たな状態をビット 4 に付け加えます。P34 が押されたときの変数 COUNTCHK の変化は、次のようになります。ここで、ビット 3、2 が 00 なら、**新しい条件の指定**として条件の読み出しを行います。



P34 が放されたときの変数 COUNTCHK の変化は、次のようになります。この場合、ビット 3～1 が 110 のときに、**条件の指定の完了**と判断します。このように、3 ビットのビット・パターンで判断して、エッジ検出を行います。



このように、最新の 3 回分を判定して、結果をキャリー・フラグとゼロ・フラグでもどします。

```
;*************************************************************************
;   P3.4 は設定モードの指定で、これが 2 回以上 0 を検出すると設定変更モード
;   また、0 に 1 が 2 回続くと、設定モードから抜けたとして、設定が変更されて
;   いた場合にフラッシュへ書き込み
;   このため、P3.4 の過去 2 回の履歴を COUNTCHK に保持し、P3.4 の変化を確認
;
;   CY    ZERO
;   0      1            : 設定完了で書き込みへ
;   1      0            : 設定変更しない
;   1      1            : 設定変更の確認
;*************************************************************************
CHKDATAIN:
        MOV     A,P3
        AND     A,#00010000b            ; 更新スイッチ(P3.4)をチェック
        OR      A,COUNTCHK              ; 履歴に追加
        AND     A,#00011100b            ; 3 回目以前をマスク
        ROR     A,1                     ; P3.4 がビット 3 に
        MOV     COUNTCHK,A              ; 履歴を更新
        CMP     A,#00001100b            ; 1 が 2 回続いたかチェック
        BZ      $CHKDATAEND             ; 設定完了でデータを書きこみ
        AND     A,#00001100b            ; 2 回 0 で設定と判断
        SET1    CY                      ; チェック完了
CHKDATAEND:
        RET
```

②指示内容の取り込み処理

新しい条件の指示では、具体的な指示内容を確認します。そのために、A/D コンバータを起動し、8 ビットのデータを得ます。A/D 変換はベクタ割り込みで処理します。タイマ H 割り込みがマスクされていない(ソフトウエア出力ではマスクされず)場合、ベクタ割り込みが発生します(ここだけベクタ割り込み)。4 回(変数 **ADEND** で指定)の A/D 変換が完了すると、割り込み処理で変数 **ADEND** を 0 にして、A/D 変換を停止するので、それを待ちます。

```
;*********************************************************************
;       変更の確認処理
;
;   A/D を起動して、ノイズ対策で 4 回分を蓄積
;   その値とスイッチ設定の状態を元に、新しい設定値を求めるとともに、
;   変化があったかフラグで戻す
;*********************************************************************
SETMAINSUB:
        MOV     ADM,#00110001b          ; A/D のモードを設定
        MOVW    AX,#0
        MOVW    ADDATASUM,AX            ; A/D 結果領域をクリア
        SET1    ADCS                    ; 1μs 待って A/D 変換開始
        CLR1    ADIF                    ; A/D 割り込みクリア
        MOV     ADEND,#4                ; A/D 変換は 4 回行う
        CLR1    ADMK                    ; A/D の割り込み許可
        CLR1    TMIFH1                  ; タイマ割り込み要求クリア
        EI
```

```
;       ソフト処理の時間計測を継続しながら、A/D 完了待ち
;   この処理の期間の時間計測は、割り込み処理
;   （出力制御のタイミングまではかからないのでカウントだけ）
;
SETLOOP1:
        MOV     A,ADEND                 ; 残り A/D 回数を読み出し
        AND     A,A                     ; 完了をチェック
        BNZ     $SETLOOP1               ; 完了していなければループ
```

③取り込み内容の加工と確認処理

4 回の A/D 変換結果は、累積されて 2 バイトの変数 **ADDATASUM** に格納されるので、その 1/4 で平均を得ます。ここでは、結果は7ビットで十分なので、さらに 1/2 にします。そして得られた 0～127 のデータは、16 ビット変数 **FREQ** の上位 8 ビット(FREQ＋1)に格納します。このとき、前の設定値との差をとっておきます。次に、レンジ指定用スイッチの値も、変更があったかどうかを確認します。変更がなければ、ゼロ･フラグがセットされます(X レジスタの内容は 0)。処理が完了したら、割り込みを禁止にして戻ります。

```
;   A/D 変換が完了したので、変更内容の確認を行う
;
;*****************************************************************
;       新しい設定結果の確認
;   新しい設定データは、変数 FREQ に設定される
;       FREQ    :スイッチの設定値（レンジ指定用）
;       FREQ+1  :A/D 変換結果（値指定用）
;   変更がなければ、ゼロ・フラグがセットされて戻る
;   ＜スイッチ入力の端子構成＞
;       P40 :スイッチの上位ビット
;       P22 :スイッチの中位ビット
;       P21 :スイッチの下位ビット
;*****************************************************************
CHKDATA:
        MOV     A,ADDATASUM             ; A/D 結果の下位桁を読み出し
        ROR     A,1                     ; 1/2
        ROR     A,1                     ; 1/4
        ROR     A,1                     ; 1/8
        AND     A,#00011111b            ; 上位ビットをマスク
        XCH     A,ADDATASUM+1           ; 上位桁読み出し
        ROR     A,1                     ; 1/2
        ROR     A,1                     ; 1/4
        ROR     A,1                     ; 1/8
        ADD     A,ADDATASUM+1           ; 下位桁とマージする
;
;       これで、変換結果は 0～127 の値になる
;
        XCH     A,FREQ+1                ; 新しいデータを保存
        SUB     A,FREQ+1                ; データの変化は？
        XCH     A,X
        MOV     A,P2                    ; レンジの読み出し
        AND     A,#00000110b            ; 未使用ビットのマスク
        XOR     A,#00000110b            ; スイッチ ON を 1 に反転
        BT      P3.2,$P32is0            ; 上位桁を確認
        SET1    A.3
```

```
P32is0:
        XCH     A,FREQ                  ; レンジを格納
        SUB     A,FREQ                  ; レンジの変更分
        XCH     A,X                     ; レンジ変更を X に保存
        OR      A,X                     ; 変更はあったか？
        DI                              ; 割り込み許可はここまで
        RET
```

[A/D 変換割り込み処理]

半固定抵抗器による設定を読み取るため、A/D 変換結果を 8 ビット精度で読み込みます。A/D 変換は、変数 **ADEND** で指定された回数実行します。変換結果は、16 ビットの変数 **ADDATASUM** に積算していきます。指定された回数の変換・累積が完了したら、A/D コンバータを停止します。

```
;*************************************************************************
;       interrupt for A/D
;  半固定抵抗での設定値を A/D コンバータで読み出して、その結果を 4 回分加算
;  4 回の変換が完了したら、変換を停止して処理を終了
;  変換結果は変数 ADDATASUM に格納され、変換完了で ADEND は 00
;*************************************************************************
ADINT:
        PUSH    AX
        MOV     A,ADCRH                 ; アナログ入力の変換結果読み出し
        ADD     A,ADDATASUM             ; 変換結果を積算
        MOV     ADDATASUM,A             ; 結果を保存
        ADDC    ADDATASUM+1,#0          ; 上位への桁上がり
        DBNZ    ADEND,$NOTEND           ; 変換回数をカウントダウン
        MOV     ADM,#00110000b          ; A/D 変換を終了
        CLR1    ADIF                    ; 要求フラグをクリア
NOTEND:
        POP     AX
DUMMY:
        RETI
```

[タイマ H 割り込み処理]

指定内容の変更中もパルス出力を正常に継続するため、ソフトウエア・タイマのカウントを継続します。指定内容の変更処理で必要な時間は、次に出力を反転する時間より短いので、単にカウントを継続するだけです。

```
;*********************************************************************
;       interrupt for TMH1
;  設定変更の確認処理の時間を計測するためだけに使用
;*********************************************************************
TMH1INT:
        DBNZ    PRECOUNT,$TMH1EXIT      ; プリカウント完了なら、
        PUSH    AX
        MOV     A,PRECOUNTH             ; 設定値を読み出す
        MOV     PRECOUNT,A              ; 作業用にコピー
        POP     AX
        DEC     B                       ; 周期をカウント･ダウン
TMH1EXIT:
        RETI
```

[動作条件確認及び設定処理]

フラッシュに格納された初期データや、設定変更結果から得た処理に必要なデータの設定を行います。
ハードウエア出力とソフトウエア出力で設定内容が異なるため、レンジによって処理方法を変更しています。

①レンジによる振り分け処理

ハードウエア出力の場合、基本的にタイマ H の設定だけなので、レンジごとの設定処理ルーチンのアドレス・テーブルを準備して、テーブル参照により**[ハードウエア出力処理]**で説明した CASE1～CASE3 の各サブルーチンへ分岐して処理します（各設定での処理時間をできるだけ短くするため、レンジごとに処理を分割しています）。
ソフトウエア出力の場合、処理時間からプログラムの大きさを考慮し、レンジごとに設定するパラメータをテーブルで準備して、共通の設定ルーチンで処理しています。

```
;*****************************************************************
;       動作条件に応じた初期設定
;
;  ハードウエア（タイマ H）でパルス出力する場合、設定プログラムの
;  アドレスをテーブル参照して、BR AX 命令で処理部に分岐
;  ソフトウエア処理の場合、設定するパラメータ・テーブルから読み出し、
;  その値を設定
;*****************************************************************

SETTMH1MODE:
        MOV     A,FREQ                  ; 新しいレンジ設定値を読み出す
        MOV     X,#0
        SUB     A,#3*2
        BNC     $SETTMH1MODE2           ; ソフトウエア処理なら分岐
        XCH     A,X                     ; AX にオフセット
        ADDW    AX,#SETTABLE+3*2+0FF00H ; アドレス・テーブルを参照
        MOVW    HL,AX
        MOV     A,[HL]                  ; 処理アドレスの下位を読み出し
        MOV     X,A                     ; X にプログラム・アドレス下位
        MOV     A,[HL+1]                ; AX に処理プログラム・アドレス
        BR      AX                      ; CASE1～3 処理プログラムに分岐
;
;       ハードウエア処理用のパラメータ・テーブル
;  初期設定サブルーチンのアドレスを示す
;
SETTABLE:
        DW      CASE1
        DW      CASE2
        DW      CASE3
```

②パラメータ設定処理

ソフトウエア出力ではパラメータを設定します。パラメータは、タイマ H の動作モード（各レンジでカウント・クロックが異なる）、インターバル・タイマの周期、プリカウント値、設定変更指示の確認周期の 4 つです。1ms 周期では、タイマ H で 50μs のインターバル時間を得るため、f_{XP}/4(2MHz)のカウント・クロックを 100 回計数させます。これで分解能 100μs となり、プリカウント値は 1 です。設定変更指示の確認周期出力反転 100 回ごとに行う設定です。

```
;
;       ソフトウエア処理用のパラメータ・テーブル
;   各レンジに対して、以下の 4 バイトのパラメータを定義
;   ・TMH1 のモード（カウント・クロックを指定、出力は禁止）
;   ・CMP01 への設定値
;   ・プリカウンタ（割り込み回数をカウント）
;   ・設定変更確認周期
;
;   TMH1 は指定されたクロックで、CMP01 への設定値カウントごとに割り込みを発生
;   それを PRECOUNTH で指定した回数カウントし、分解能の時間(周期の 1/20)を得る
;   分解能で決まる時間を A/D 変換で得た値(0～127)に 10 を加算した回数カウントして、
;   周期/2～周期×13.7/2 の時間を得る(1/2 にするのは、周期の半分の時間で出力を反転
;   させるため)
;
SETTABLE2:
        DB      00010000b,      99,1,100        ; 1ms レンジ（タイマは 50μs）
        DB      00100000b,      249,1,10        ; 10ms レンジ（タイマは 0.5ms）
        DB      00110000b,      124,5,1         ; 100ms レンジ（タイマは 1ms）
        DB      01000000b,      97,1,1          ; 1s レンジ（タイマは 50ms）
        DB      01000000b,      194,5,1         ; 10s レンジ（タイマは 100ms）
```

上記のパラメータを設定するプログラムは、次のようになります。設定が完了すると、キャリー・フラグを設定して戻るので、戻り先ではハードウエア出力かソフトウエア出力かを簡単に判定できます。

```
;
;       ソフトウエアによる処理の場合
;
SETTMH1MODE2:
        ADD     A,A                     ; テーブルのオフセットを計算
        XCH     A,X                     ; AX にオフセット
        ADDW    AX,#SETTABLE2           ; パラメータのアドレスを計算
        MOVW    HL,AX
        MOV     A,[HL]                  ; TMH のモードを読み出し
        MOV     TMHMD1,A                ; モードをセット（タイマは停止）
        MOV     A,[HL+1]                ; CMP01 への設定値を読み出し
        MOV     CMP01,A                 ; カウント値の設定
        MOV     A,[HL+2]                ; ソフトカウント値の読み出し
        MOV     PRECOUNTH,A             ; データ保持レジスタへの設定
        MOV     A,[HL+3]                ; 設定変更確認周期の読み出し
        MOV     COUNT2H,A               ; データ保持レジスタへの設定
        SET1    CY                      ; CY=1 でソフト処理を示す
        RET
```

[フラッシュ書き込み処理]

フラッシュを EEPROM の代わりに利用して、出力パルスの条件指定が完了したとき、設定内容がフラッシュに格納されている最新の値と異なっていた場合に、新しい設定内容をフラッシュに保存します。

①書き込み制御処理部

新しい設定値をフラッシュに格納されている最新の値と比較し、異なればフラッシュに新しい値を追記します。

```
;
;       更新完了
;
CHANGEEXIT:
        PUSH    HL
        MOVW    AX,_Rd_Address          ; EEPROM の最新データ・アドレス
        MOVW    HL,AX
        MOV     A,[HL]                 ; 現在の EEPROM データを読み出し
        SUB     A,FREQ                 ; 新しい設定値と比較
        XCH     A,X
        INC     L
        MOV     A,[HL]                 ; 2 つ目のパラメータも比較
        SUB     A,FREQ+1
        OR      A,X
        BZ      $END_ADINT             ; 変更になければ割り込み終了
        CALL    !WRITEDATA             ; 変更分をフラッシュに書き込み
END_ADINT:
        POP     HL
        RET
```

②書き込み処理部

フラッシュのセルフ･プログラミングを利用して、簡易 EEPROM 機能を実現します。使用するフラッシュは 1 ブロックだけなので、ここでは、フラッシュへの書き込みをできるだけ行わないようにしています。また、1K バイトですべて処理するため、エラー処理は省略しています。

```
;
;       Data Write to Flash
;
WRITEDATA:
        CALL    !SelfFlashModeOn       ; セルフ書き込みモードに
        INCW    HL                     ; 最新データの次アドレスへ
        MOVW    AX,HL
        CMP     A,#EEPROM_BLOCK
        BZ      $WRIREDATA2            ; 書き込み可能なら分岐
;
;       ブロックがいっぱいなので消去
;
        CALL    !BlockErase            ; ブロックを消去する
;
;       BC      $ERROR
;  エラーが起こるほど書き込みをしないのでエラーは無視
;
        MOVW    AX,#BROCK3             ; ブロックの先頭アドレス設定
WRIREDATA2:
        MOVW    _Wr_Address,AX         ; 新しいアドレスを設定
        MOVW    _Rd_Address,AX         ; 読み出し用にも設定
        MOVW    AX,FREQ                ; 書き込みデータをセット
        CALL    !EEPROMWrite           ; 1 バイト目書き込み
        XCH     A,X
        INC     _Wr_Address            ; 書き込みアドレス更新
        CALL    !EEPROMWrite           ; 書き込み
        CALL    !SelfFlashModeOff      ; セルフ書き込み終了
        RET
```

③データ部

データ部の例を示します。2FE 番地と 2FF 番地が初期データです。この直後 300 番地からの 1 ブロックが、新しい設定内容の書き込みに使用されます。ここはプログラムと分離し、データ部分だけの別ファイルとしています。

```
PUBLIC  EEPROM_BLOCK
PUBLIC  BROCK3

EEPROM_BLOCK    EQU     3       ; ここでは、1Kの最後のブロックを使用
;
;       Output Freq Initial Data Area
;
IDATA   CSEG    AT      2FEH
        DB              00000000b       ; highest range
        DB              00000000b       ; 1μs
;
;       Following Area is Block3 For EEPROM Emulation
;
BROCK3:
        DB      0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH   } 未書き込み時の
                                :                         } ダミー・データを
        DB      0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH   } 1ブロック分定義
        END
```

[セルフ・プログラミング処理部]

この部分は、独立したプログラムとして別に準備します。そのため、パルス出力プログラムの先頭で、次のように外部のサブルーチンとして宣言しておきます。

①外部参照の宣言

```
EXTRN   SelfFlashModeOn
EXTRN   SelfFlashModeOff
EXTRN   BlockErase
EXTRN   EEPROMWrite
EXTRN   _Wr_Address
EXTRN   _Rd_Address
EXTRN   EEPROM_BLOCK
EXTRN   BROCK3
```

②セルフ・プログラミング処理本体（別ファイル）

実際に μPD78F9200 でセルフ・プログラムを行うためのプログラム本体です。基本的な処理は、マニュアル記載の内容をプログラムに展開しただけなので、説明は省きます。

```
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; システム          :  78K/0S EEPROM エミュレーション・プログラム(簡易版)
; ファイル名        :  selfP.asm
; バージョン        :  1.00
; ターゲット CPU    :  μPD78F9200
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
PUBLIC SelfFlashModeOn
PUBLIC SelfFlashModeOff
PUBLIC __SelfFlashBlockErase
PUBLIC BlockErase
PUBLIC __FlashEEPROMWrite
PUBLIC EEPROMWrite
PUBLIC _Wr_Address
PUBLIC _Rd_Address
EXTRN EEPROM_BLOCK
```

```
;---------------------------------------------------------------------
; フラッシュ制御用定数の EQU 定義
;---------------------------------------------------------------------
CMD_INTERNAL_VERIFY     EQU     (00000001B)     ;内部ベリファイ
CMD_INTERNAL_VERIFY2    EQU     (00000010B)     ;内部ベリファイ 2
CMD_BLOCK_ERASE         EQU     (00000011B)     ;ブロック消去
CMD_BLANK_CHECK         EQU     (00000100B)     ;ブランク・チェック
CMD_BYTE_WRITE          EQU     (00000101B)     ;1 バイト書き込み
NO_ERR                  EQU     (00000000B)     ;エラーなし
ERR_FPRERR              EQU     (00000001B)     ;PFCMD 書き込みエラー
ERR_VCERR               EQU     (00000010B)     ;消去/書き込み/ベリファイ・エラー
ERR_WEPRERR             EQU     (00000100B)     ;プロテクト・エラー
FALSE                   EQU     (0FFH)          ;異常
TRUE                    EQU     (00H)           ;正常
WRITE_RETRY             EQU     (10)            ;書き込み時のリトライ回数
ERASE_RETRY             EQU     (20)            ;消去時のリトライ回数
;---------------------------------------------------------------------
; RAM 設定
;---------------------------------------------------------------------
EP_RAM1         DSEG    saddrp
_Wr_Address:    DS      2               ; EEPROM 書き込みアドレス設定用
_Rd_Address:    DS      2               ; EEPROM 読み出しアドレス設定用
EEPROM_PROG     CSEG
;---------------------------------------------------------------------
; 関数名  : __SelfFlashBlockErase
; 入　力  : なし
; 出　力  : CY = 0 (正常終了) / CY = 1 (異常終了)
; 概　要  : ブロック 3 の消去とブランク・チェックを行う
;---------------------------------------------------------------------
;bool   _SelfFlashBlockErase(void)
__SelfFlashBlockErase:
BlockErase:
        PUSH    BC
        PUSH    AX
        MOV     B,#ERASE_RETRY          ; 消去時のリトライ回数設定
        BR      $SF_BKER_ST             ; ブランク・チェックから開始
SF_BKER_RE:
        CALL    !FlashBlockErase        ; 消去処理
        BC      $SF_BKER_01             ; 異常終了
SF_BKER_ST:
        CALL    !BlankCheck             ;ブランク・チェック処理
        BNC     $SF_BKER_TR             ;ブランク・チェック正常終了
SF_BKER_01:
        DBNZ    B,$SF_BKER_RE           ;リトライ回数のチェック
SF_BKER_FL:
        SET1    CY                      ;戻り値 = 異常終了
SF_BKER_TR:
        POP     AX
        POP     BC
        RET
;---------------------------------------------------------------------
; 関数名  : SelfFlashModeOff
; Input   : なし
; Output  : なし
; 使用レジスタ : なし
; 概　要  : セルフ・プログラミング・モードを解除する
;---------------------------------------------------------------------
```

```
SelfFlashModeOff:
        MOV     FLCMD,#0
SF_MDOF_LP:
        MOV     PFS,#NO_ERR
        MOV     PFCMD,#0A5H             ; PFCMD レジスタ制御
        MOV     FLPMC,#00H              ; FLPMC レジスタ制御(設定値)
        MOV     FLPMC,#0FFH             ; FLPMC レジスタ制御(設定値の反転)
        MOV     FLPMC,#00H              ; 通常モード設定
        BT      PFS.0,$SF_MDOF_LP       ; 特定レジスタへの書き込み完了確認
        RET
;---------------------------------------------------------------------
; 関数名  : SelfFlashModeOn
; Input   : なし
; OutPut  : なし
; 使用レジスタ : なし
; 概　要  : セルフ・プログラミング・モードに設定する
;---------------------------------------------------------------------
SelfFlashModeOn:
        MOV     MK0,#11111111B
        DI
        MOV     FLCMD,#0
SF_MDON_LP:
        MOV     PFS,#NO_ERR
        MOV     PFCMD,#0A5H             ; PFCMD レジスタ制御
        MOV     FLPMC,#01H              ; FLPMC レジスタ制御(設定値)
        MOV     FLPMC,#0FEH             ; FLPMC レジスタ制御(設定値の反転)
        MOV     FLPMC,#01H              ; セルフ・プログラミング・モード設定
        NOP                             ;
        HALT                            ;
        BT      PFS.0,$SF_MDON_LP       ; 特定レジスタへの書き込み完了確認
        RET
;---------------------------------------------------------------------
; 関数名  : FlashBlockErase
; Input   : なし
; OutPut  : CY(CarryFlag) = 正常終了(0) / 異常終了(1)
; 使用レジスタ : なし
; 概　要  : ブロック 3 を消去する
;---------------------------------------------------------------------
FlashBlockErase:
        MOV     FLCMD,#CMD_BLOCK_ERASE  ;フラッシュ制御コマンド設定(ブロック消去)
        MOV     FLAPH,#EEPROM_BLOCK     ;ブランク・チェック・ブロック番号設定
        MOV     FLAPL,#00H
        MOV     FLAPHC,#EEPROM_BLOCK
        MOV     FLAPLC,#00H
;
;  引き続き、実際のフラッシュ・セルフ・プログラミング処理へ
;
;---------------------------------------------------------------------
; 関数名  : SubFlashSelfPrg
; Input   : なし
; Output  : CY(CarryFlag) = 正常終了(0) / 異常終了(1)
; 使用レジスタ : なし
; 概　要  : フラッシュ・セルフ・プログラミング機能を呼び出す
;---------------------------------------------------------------------
```

```
SubFlashSelfPrg:
SubSelfPrg:
        PUSH    AX
        MOV     PFS,#NO_ERR             ;フラッシュ・ステータス・レジスタのクリア
        HALT                            ;セルフ・プログラミング開始
        MOV     A,PFS
        SUB     A,#NO_ERR
        ADD     A,#0FFH
        POP     AX
        RET
;---------------------------------------------------------------------
; 関数名  : __FlashBlockBlankCheck
; Input   : なし
; OutPut  : CY(CarryFlag) = 正常終了(0) / 異常終了(1)
; 使用レジスタ : なし
; 概　要  : 指定ブロックのブランク・チェックを行う
;---------------------------------------------------------------------
__FlashBlockBlankCheck:
BlankCheck:
        MOV     FLCMD,#CMD_BLANK_CHECK  ;フラッシュ制御コマンド設定
; (ブロック・ブランク・チェック)
        MOV     FLAPH,#EEPROM_BLOCK     ;ブランク・チェック・ブロック番号設定
        MOV     FLAPL,#00H
        MOV     FLAPHC,#EEPROM_BLOCK
        MOV     FLAPLC,#0FFH
        BR      $SubSelfPrg
;---------------------------------------------------------------------
; 関数名  : __FlashEEPROMWrite
; Input   : _Wr_Address = 書き込み開始アドレス
;         : X = 書き込みデータ (8bit)
; 使用レジスタ  : なし
; OutPut  : CY(CarryFlag) = 正常終了(0) / 異常終了(1)
; 概　要  : EEPROM での書き込みとベリファイを行う
;---------------------------------------------------------------------
;bool   _FlashEEPROMWrite(unsigned int)
__FlashEEPROMWrite:
EEPROMWrite:
        PUSH    BC
        PUSH    AX
        MOV     A,_Wr_Address           ; 書き込みアドレスの読み出し
        MOV     C,A                     ; 作業用レジスタにセット
        MOV     FLCMD,#CMD_BYTE_WRITE   ;フラッシュ制御コマンド設定
; (バイト書き込み)
FL_WR_W1:
        MOV     B,#WRITE_RETRY
        MOV     FLAPH,#EEPROM_BLOCK     ; 書き込みブロックの設定
        MOV     A,C
        MOV     FLAPL,A                 ; 書き込み/ベリファイ・アドレス設定
        MOV     A,X
        MOV     FLW,A                   ; 書き込みデータ設定
FL_WR_W2:
        CALL    !SubSelfPrg
        BNC     $FL_WR_W3               ; 正常完了なら抜ける
        DBNZ    B,$FL_WR_W2             ; リトライ・カウンタ判定(リトライ 10 回)
        BR      FL_WR_E
```

```
FL_WR_W3:
;
; 書き込み完了してベリファイする
;
        MOV     FLCMD,#CMD_INTERNAL_VERIFY2 ;フラッシュ制御コマンド設定
; (内部ベリファイ 2)
        MOV     FLAPH,#EEPROM_BLOCK
        MOV     A,C                         ; 書き込みアドレスの読み出し
        MOV     FLAPL,A                     ; ベリファイ開始アドレス設定
        MOV     FLAPHC,#EEPROM_BLOCK
        MOV     FLAPLC,A                    ; ベリファイ終了アドレス設定
        CALL    !SubSelfPrg
FL_WR_E:
        POP     AX
        POP     BC
        RET
        END
```

[プログラムの配置とデータ領域]

①プログラムの配置

このプログラムはサイズに余裕が少ないため、**割り込み処理部と[設定変更確認処理]の②指示内容の取り込み処理～③取り込み内容の加工と確認処理**をベクタ･テーブルに続けて記述します。

```
RSTVECT CSEG    AT      0
        DW      START                   ;00:リセットスタート処理のベクタ
        DW      DUMMY                   ;02:
        DW      DUMMY                   ;04:
        DW      DUMMY                   ;06:LVI
        DW      DUMMY                   ;08:INTP0
        DW      DUMMY                   ;0A:INTP1
        DW      TMH1INT                 ;0C:INTTMH1
        DW      DUMMY                   ;0E:INTTM000
        DW      DUMMY                   ;10:INTTM010
        DW      ADINT                   ;12:INTAD
```

ここに 2 つの**割り込み処理部**を記述

さらに、続けて次のように記述すれば、0～7EH 番地に配置されます。

②指示内容の取り込み処理～③取り込み内容の加工と確認処理を記述

そのあと、残りのプログラムを**新たなセグメント**として"CSEG"を宣言して、次のように記述していきます。この部分は、オプション･バイトやプロテクト･バイトのあと(82H 番地)に配置されます。

```
        CSEG
;*********************************************************************
;       ハードウエアの初期化
;
;  ポートやアナログ入力の設定、電源電圧を 4V 以上に設定する
;*********************************************************************
START:
```

②データ領域

セルフ・プログラミングを除いた、このプログラムで使用するデータ（変数）領域は、次のとおりです。

```
;
;       データ領域
;
        DSEG
        DS              20H
STACKTOP:
        DSEG    saddrp
ADDATASUM:
        DS      2                       ; 4 回分の A/D 加算用
FREQ:
        DS      1                       ; レンジ指定用
        DS      1                       ; 値設定用
PRECOUNT:
        DS      1                       ; プリカウント作業用
PRECOUNTH:
        DS      1                       ; プリカウント用設定値
COUNT:
        DS      1                       ; カウント指定値用
COUNTH:
        DS      1                       ; カウント指定値設定値
COUNTCHK:
        DS      1                       ; 設定変更タイミング用
COUNT2H:
        DS      1                       ; 設定変更確認用カウンタ設定値
ADEND:
        DS      1                       ; A/D 変換カウント用
```

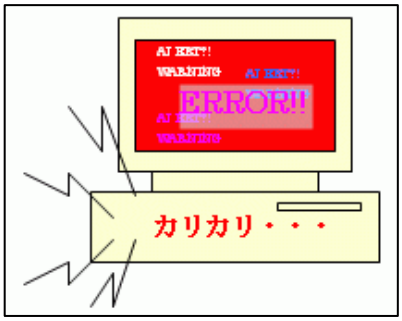
1.1.12 ウッチドッグ・タイマ

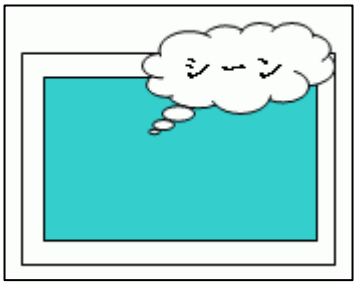
ウォッチドッグ・タイマの基本

(1) はじめに

パソコンでは、何らかの原因で動作がおかしくなった（異常動作を繰り返す、黙り込んで何の操作も受け付けない等）場合には、「暴走した」とか「フリーズした」として、再起動した経験があるでしょう。使用者は、パソコンが何をやっているかをある程度理解しており、異常を検知、対策ができるので、このような処理ができるわけです。

では、マイコンが通常使われている組み込み分野のアプリケーションではどうでしょうか。使用者は一般の人です。マイコンが使われていることも知らない可能性があります。異常を感じたら、コンセントから電源コードを抜くかも知れませんが、常に監視しているわけではありませんし、的確な判断や処理ができると期待できません。





その一方で、アプリケーション・プログラムはどんどん大きくかつ複雑になっています。そのため、設計者も全体を理解するのは困難になってきています。そのために、プログラムに何らかの対策を盛り込むのは無理（対策を考えるよりは、プログラムそのものの完成度を上げるのが先）でしょう。プログラムのみでの対策ができないとなると、なんらかのハードウエアの助けを借りる必要があります。プログラムが異常動作をしていないかどうかを監視するハードウエアの代表的なものに「ウォッチドッグ・タイマ」（番犬機能）があります。

(2) ウォッチドッグ・タイマとは

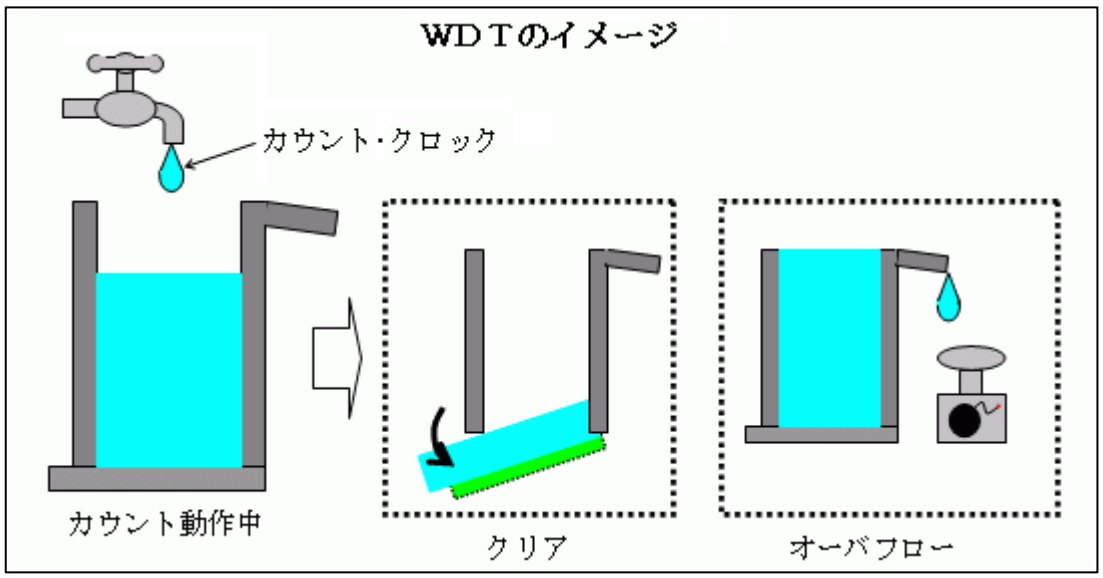
ウォッチドッグ・タイマ（以下 WDT）は、プログラムが異常動作していないかどうかを監視する番犬の機能を果たすものですが、その実体は、監視対象であるプログラムからはクリアできても、止めることができない単なるタイマです。このタイマはオーバフローすると、WDT 割り込みを発生したり、リセットをかけたりします。

通常は、オーバフローしないように定期的にクリアすることで、WDT 割り込みやリセットを発生させないようにします。これは、下記のように対応させると分かりやすいと思います。このように、**WDT は**他の機能と違って、**動作させない**（吠えさせない）ようにするのが、正常な使い方になります。

WDT	が	オーバフローして	リセットする前に	クリアして	リセットさせない
↕		↕	↕	↕	↕
番犬	が	空腹になって	吠え出す前に	餌をやって、	吠えさせない

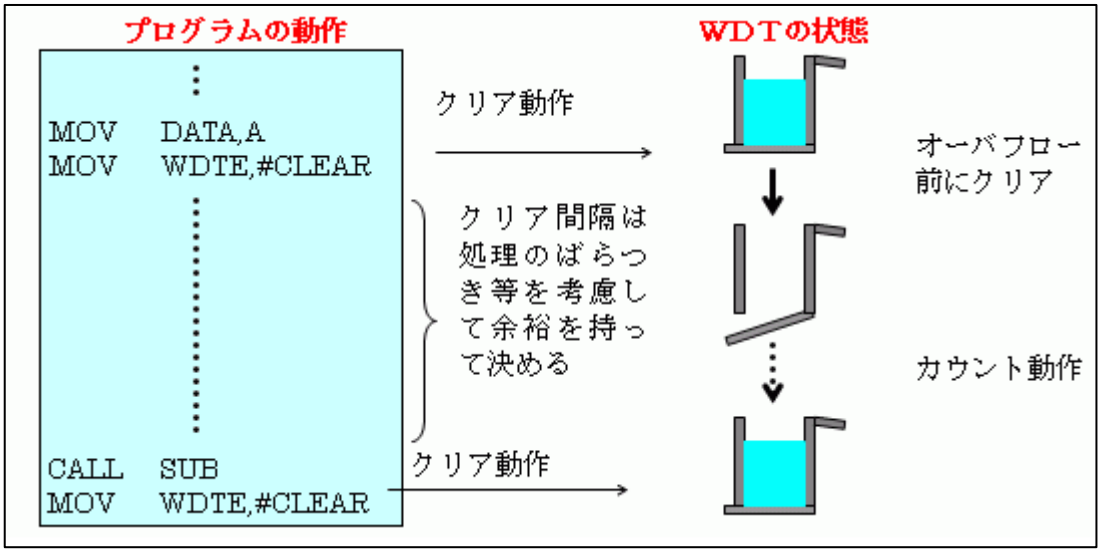
(3) WDT の構成と動作(概要)

WDT タイマは、クリアできるがとめられない（フリーランの）タイマで、起動すると、指定されたクロックをひたすらカウントし続けます。イメージとしては、下図のようになります。あふれる（オーバフロー）までの時間は、水滴（カウント・クロック）の頻度（周波数）で決まります。水滴を溜めていき（カウントして）、あふれる（オーバフロー）前に底をあけて水を捨て（クリア）れば、なにも起こりません。もし、水を捨てるのを忘れるか、間に合わなかった場合には、あふれて（オーバフロー）しまい、爆弾のスイッチが押される（リセットがかかる）ことになります。これで、プログラムの異常検出ができます。



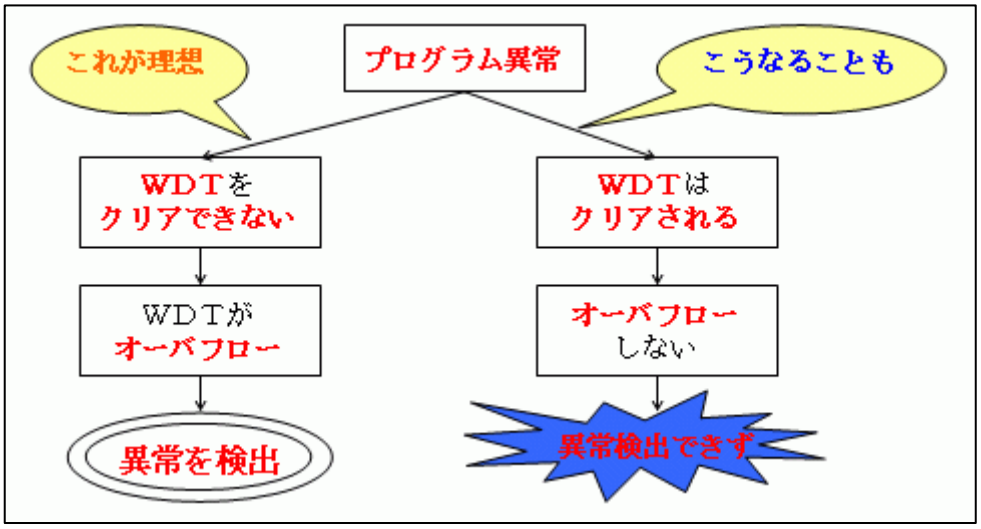
(4) WDT の使い方の概要

WDTは、プログラムが正常に動作しているときには、その中で、クリア処理を定期的に行うことで、WDT が動作しない（番犬が吠えない）ようにします。



(5) WDT の限界

プログラムの暴走等、異常な状態になったときには、WDT のクリア処理が定期的にできなくなります。そのため、WDT がオーバフローして、WDT割り込みが発生するかリセットがかかる（番犬が吠える）ので、異常を検出することができます。しかし、プログラムが異常になっても WDT が定期的にクリアされていれば、異常を検出することができません。



これは、どこで WDT をクリアしてもかまわないためで、マンガに出てくるダメな番犬が、侵入者から肉をもらって吠えるどころか尻尾を振っているのと似たようなものです。WDT も決して万能ではありません。

また、一部のマイコンに内蔵された WDT は、初期状態では停止していて（寝たままで）、プログラムの中で起動しないと（起こさないと）機能しない（番犬の役割を果たさない）ものもあります。

(6) 現状の WDT

前項では WDT の限界について説明しましたが、最近のマイコンに内蔵された WDT ではこれが改善されて、利口になってきています。主な改良は次のような点で、今後も改良が進むと思われます。

- 特定のデータ・パターンを書き込んでクリア、違う値を書くとリセット（いつものエサ以外を与えると噛み付く）
- クリアできるタイミングを制限、それ以外で書き込むとリセット（空腹でないときにエサをやると噛み付く）
- 止められない専用クロックを使い、最初から起動している（いつも起きていて、見張っている）

(7) WDT 使用上の注意点

WDT を使う上で一番の注意点は、オーバフローまでの時間をどうするかと、どのタイミングでクリアするかの問題です。オーバフローまでの時間を短くすると、頻繁にクリアする必要があり、システムとしての制約が増えてきます（番犬が直ぐに空腹になるようだと、頻繁にエサをあげないといけないので、遠くまで出かけられない）。
クリアするタイミングをぎりぎりまでにしてしまうと、カウント・クロックの変動のようなハードウエアの変動要因や、プログラムの処理内容による実行時間の変動や、非同期で発生する割り込み等のソフトウエア処理の変動要因により、間に合わなくなることがあります。これらを考慮して、最悪の場合でもオーバフローしないように設計する必要があります。また、スタンバイのように CPU が止まっているときに、WDT がどうなるのかの検討も必要です。
なお、よくある誤りが、「WDT のクリアが目的になってしまっているケース」です。そのために、タイマ割り込みの中で WDT をクリアするといった安易な処理を行うケースが見受けられます（エサをやるのが面倒で、番犬が空腹で吠えないように自動給仕機でエサをやるようなもの）。これは、まさに本末転倒です。

(8) WDT のまとめ

WDT は完全なものではありませんし、使いこなすためにはプログラム全体の動きを理解する必要がありますが、その動作と限界を理解すれば、プログラムの異常検出手段として使える機能です。

1.1.13 I²C

I²C バスの概要

はじめに

I²C バスは、フィリップス社が提唱しているシリアル・バスで、2 本の信号により、複数の機器の間でデータのやり取りをサポートします。
フィリップス社の特許が 2004 年 8 月に失効したので、以降は契約やロイアリティの支払いが不要となりました。

Q1. I²C バスの概略仕様を教えてください。

A1. I²C バスは、シリアル・クロックとシリアル・データの 2 信号で複数の機器を結んで、データをやり取りします。
次の 3 つの転送速度があり、速いモードの機器は、遅いモードのバスに接続しても使える必要があります。

標準モード	: ～100kbps
ファースト・モード	: ～400kbps
高速モード	: ～3.4Mbps

なお、高速モード・バスは、標準モードやファースト・モードのバスとは、ブリッジで分離する必要があります。
また、通信はシリアル・クロックに同期して行われますが、シリアル・クロックを出力し、通信相手を指定して通信を制御する機器(マスタと呼ぶ)は、1つのバスに複数存在することが許されています(マルチ・マスタ)。
このため、データのやり取りだけではなく、アドレスの指定やバス・アービトレーション等のプロトコルも含んで規定されています。

(1) 接続機器

I²C バスに接続される機器には、次のようなものがあります。

マスタ専用　　：通常、マイコンは、マスタとしての使い方が主です。
スレーブ専用　　：EEPROM や A/D、D/A 等があります。
マスタ／スレーブ共用　：マルチ・マスタ・システムで、両方の機能を持たせる場合です。

通常、マイコンで使用することが多いのはマスタ専用の機能で、この機能で EEPROM と接続します。アプリケーションによっては、スレーブとして使用することもあります。

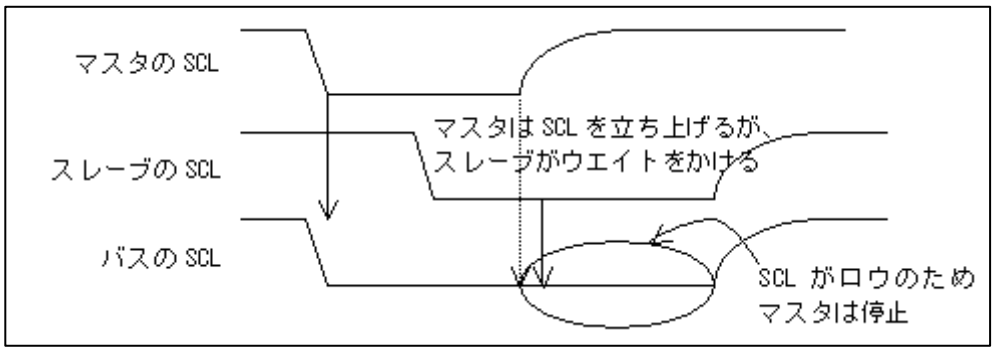
(2) インタフェース信号

インタフェース信号は、複数の機器の出力を接続するために、オープン・ドレーン出力でドライブするようになっており、ロウ・レベルは機器が出力して、ハイ・レベルはプルアップ抵抗で供給されます。
なお高速モードでは、立ち上がりの高速化のため、機器内部で電流ソース・プルアップが内蔵されています。
以降、特に断らない場合は、標準バス／ファースト・バスで話を進めます。

バス信号の電圧レベルとしては、2 つが規定されています。最近のマイコンの動作電源電圧は変更できますので、I²C バスのプルアップ抵抗を V_{DD} に接続し、ハイ・レベル入力電圧は $0.7V_{DD}$(MIN)となります。
古いデバイスや電源電圧が固定されているデバイスを接続する場合は、5V±5%に接続し、ハイ・レベル入力電圧は 3V(MIN)となります。

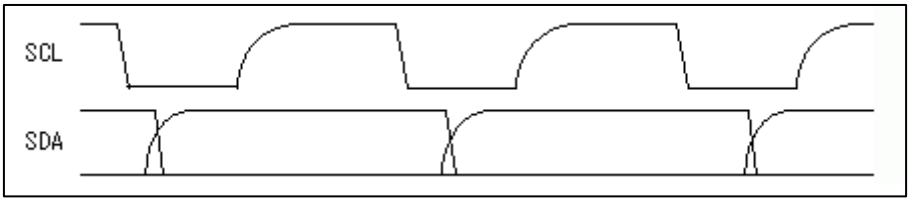
(a) シリアル・クロック(SCL)

通信は、SCL で同期を取りながら行います。通常、SCL はマスタが出力し、送信する機器は SCL がロウの期間にシリアル・データを変更し、SCL がハイの期間では保持することで通信を行います。
しかし、スレーブが通信準備できていないときなどに SCL をロウにすることで待ち合わせを行い、データの同期をとることができます。



(b) シリアル・データ(SDA)

SCL に同期して、データ転送に使用されます。マスタとスレーブのどちらからも、送信することがあります。
また、通信データに対してのアクノリッジ信号も SDA を通じて送信されるので、 8 ビットのデータとそれに対するアクノリッジでは、転送方向が逆になります。
なお、通常の通信では、SCL がロウのときに、必ず SDA は変化します。



SCL がハイの状態で、SDA が変化すると、スタート・コンディションやストップ・コンディションのような制御信号となります。

(3) 基本的な通信方法

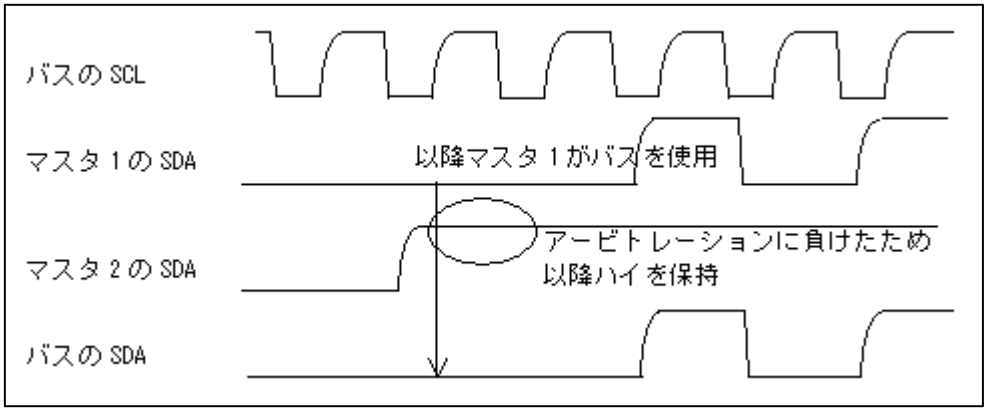
[スタート・コンディション]

マスタが通信をスタートするときには、最初にバスの使用権を獲得する必要があります。
このために、スタート・コンディションを発行します。これは、SCL がハイの状態で、SDA をロウに落とすことで実現します。
このとき、すでに SDA か SCL がロウになっていると、他のマスタがバスを使っている状態ですので、新たに通信を開始することができず、通信予約状態にします。また、バスを使えなかったマスタが、スレーブとしての機能をもっている場合には、スレーブとしての通信待ちにします。
ただし、同時に 2 つ以上のマスタがスタート・コンディションを発行した場合には、この段階ではどちらが優先かは区別できません。

なお、スタート・コンディションは、通信の最初にバスを確保するために使用する以外に、バスを確保したままでスレーブとの間のデータ送受信の方向を切り替えたりするためにも使用します。このとき使用するスタート・コンディションをリスタート・コンディションと呼びます。具体例は、後述の「リスタート・コンディション」をご参照ください。

[アドレスとアービトレーション]

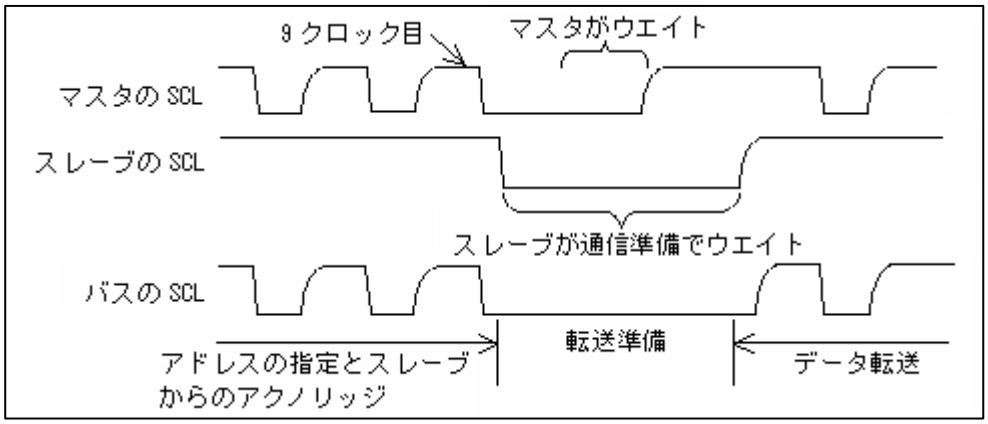
スタート・コンディションに続いて、スレーブを指定するために、7 ビットのアドレスと転送方向を示す 1 ビットの合計 8 ビットを送信します。
同時に 2 つ以上のマスタが、スタート・コンディションを発行した場合は、各マスタがスレーブのアドレスを送信することになります。I²C バスではロウが強いので、アドレスの小さいスレーブ(SDA をロウに保持していた方)への通信が優先となります。このようにして、バス使用権の調停が行われます。



アドレスと転送方向の送信に対して、SCL の 9 クロック目で選択されたスレーブからアクノリッジ(ロウ)が戻されます(このときに SCL をロウに引くことで、ウエイトがかかりますが、ウエイトについては別途説明します)。
指定したスレーブが存在しなければ、バスをロウに引くものが存在しないので、アクノリッジは戻らないことになります。アクノリッジが戻らなかったら、通信はそこで終わりとなり、マスタはストップ・コンディションを発行して、バスの使用を中止します。

[データ転送]

スレーブとの通信が確立したら、データの転送に移ります。このとき、スレーブは SCL をロウに引いてウエイトをかけた状態で、アドレスの 8 ビット目で指定された転送方向での通信準備を行い、準備が完了したら SCL をロウに引くのをやめます。これにより、マスタが SCL を制御できるようになり、マスタからの SCL でデータ転送を行えるようになります。以下のタイミングも参照してください。



[アクノリッジ]

データ転送は 8 ビット単位で行われ、データを受信した方はデータのあとにアクノリッジを戻します。

マスタからスレーブにデータを送信する場合、スレーブはデータを受け取ると、必ずアクノリッジを戻します。

データを受け取れなかったときには、アクノリッジを戻さないことで、マスタに知らせます。マスタが受信する場合、受信した最後のデータに対してアクノリッジを戻さないことで、スレーブに通信の終了を知らせます。スレーブは、アクノリッジがないことで通信の終了と判断し、以降の送信を中止して、SDA をハイにします。

このような処理を行わないで、スレーブが次のデータの送信を開始して、SDA をロウに引いてしまうと、マスタがストップ・コンディションを発行できなくなります。

[ウエイト]

スレーブとマスタは、SCL をロウに引くことで、通信を停止(ウエイト)させられます。通常、ウエイトは 8 クロック目のあと(データの転送終了後、アクノリッジを戻す前)か 9 クロック目のあと(アクノリッジ後、次の転送の前)にかけます。

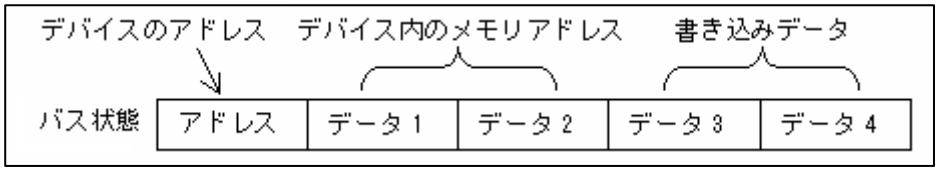
受信側でデータに対してアクノリッジを戻すかどうかを判断する場合は、8 クロック目でウエイトをかけます。

送信側は、アクノリッジのあと(9 クロック目)でウエイトをかけることになります(これでアクノリッジを確認して、その後の動作を決められます)。

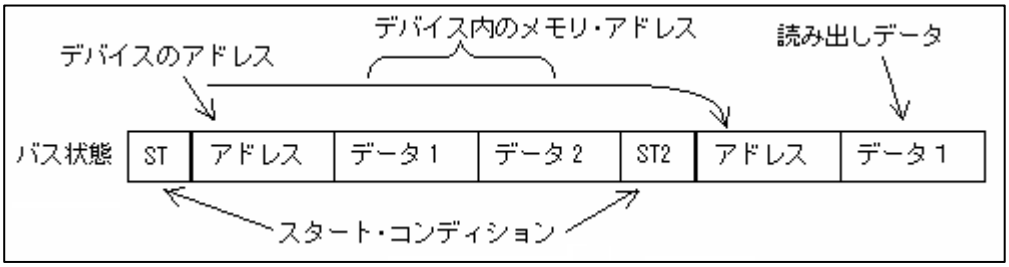
[リスタート・コンディション]

EEPROM のように、デバイスのアドレス以外に内部にアドレス情報をもったデバイスの場合には、I²C バスのアドレスでは、デバイスそのものを選択することしかできません。デバイスの中のどのアドレスのメモリ・セルかを指定するためのアドレス情報は、データとして送る必要があります。

書き込みの場合には、次の例のように、メモリ・アドレスに続いて書き込みデータを送信すればいいのですが、読出しでは、途中でデータの転送方向を切り替える必要があります。



以下に、読み出しの例を示します。
最初のスタート・コンディションでバスの使用を開始し、デバイスのアドレスを指定します(このときは、マスタからの送信を指定します)。その後、デバイス内のアドレスをデータとして送信します。このままでは読み出せないので、再度スタート・コンディション(ST2)を発行します。この ST2 が、リスタート・コンディションです。
その後、再度アドレスでデバイスを指定しますが、今度はスレーブの送信を指定します。
その後のデータは、スレーブがマスタに送信するので、読み出しができるようになります。



[ストップ・コンディション]

データ転送が終了すると、バスを開放します。このために使用するのが、ストップ・コンディションです。
マスタは、SCL がロウの状態で、SDA をロウにして、SCL をハイにします。その状態で、SDA をハイに立ち上げると、ストップ・コンディションとなります。EEPROM への書き込みでは、デバイス内の実際の書き込み動作は、ストップ・コンディションを検出してから行います。

Q2. その他の仕様や、仕様外の注意事項はありますか。

A2. (1) アドレス

スレーブのアドレスを指定するアドレス・フィールドは 7 ビットですが、そのすべてが自由に使えるわけではありません。最初の 4 ビットで、0000 と 1111 は用途が規定されています。0000 で始まるアドレスは、ジェネラル・コール・アドレス(0000 000)のような目的で使用します。1111 で始まるアドレスの内で、1111 0xx は、この xx の 2 ビットとそれに続く 1 バイトの合計 10 ビットで、スレーブのアドレスを指定できます。ただし、通常のマイコンでは、10 ビットのアドレスを使う可能性は低いと考えられます。

(2) バスの初期状態

途中から I²C バスのマスタとして参加するような場合、バスが使用中かどうかが、完全には判断できません。通常は、SCL や SDA がハイの状態を保持していれば、バスは開放状態と考えられますが、SCL のハイ期間については上限の規定がないので、どの程度待てばバスが開放状態と断定できるか分かりません。
したがって、この部分はコントローラではなく、上位のプログラムで判断する必要があります。

(3) 動作中のリセット

通信中にリセットがかかるような場合、バス上のすべてのデバイスがリセットを認識できればいいのですが、EEPROM のようにリセット信号をもたないデバイスも存在します。
もし EEPROM が SDA にロウを出力した状態であれば、そのままではスタート・コンディションもストップ・コンディションも発行できないため、バスを使用できなくなります。これは、コントローラでは対応できません。
この場合には、SCL を出力ポートでドライブして、出力ポートから SCL パルスを出力します。これにより、EEPROM は 1 バイトの残りのデータ出力を継続します。EEPROM がハイを出力するか、出力が終わってもアクノリッジが戻らないので、次の出力を止めれば、SDA がハイ状態となります。この状態になると、スタート・コンディションもストップ・コンディションも発行できますから、バスを使用できるようになります。

I²C バスのプルアップ抵抗値

Q1. μPD78P0308Y の I²C バスはオープン・コレクタ信号ですが、このラインのプルアップ抵抗値はいくつですか。

A1. I²C バスの規格は、フィリップス社が設定しています。詳細は、NXP 社のホーム・ページで仕様書をご覧ください(http://www.nxp.com/acrobat_download/literature/9398/39340011_jp.pdf)。p.39-40 に、プルアップ抵抗の最大値/最小値、バス容量、電源電圧、接続デバイス数などが説明されているようです。

I²C バスの初期設定で、STT0 と SPT0 を同時にセットするとうまく動作しない

Q1. μPD78F0034A の I²C の初期設定で、STT0 と SPT0 を同時にセットしていますが、うまく設定できません。

A1. 基本的な問題があります。STT0 と SPT0 を同時にセットすることは禁止なので、絶対に行わないでください。まず、ストップ・コンディションの生成を行い、その後にスタート・コンディションの生成に移ります。ストップ・コンディションの生成は、IICCL0 でクロックの選択を行ったあと、IICC0 で IICE0 と SPT0 をセットしてください。

外部の EEPROM に I²C バスで接続する場合の処理例(マスタ)

Q1. 外部の EEPROM に I²C バスで接続する場合の処理例(マスタとしての動作)を教えてください。

A1. [処理の概略手順]

通信を行うには、以下の手順となります。

(0) ポートの設定(PM32,33 を 1、P32,33 を 0 に設定)

(1) 転送クロックの設定

(2) コントロール・レジスタの設定

(3) PM32, 33 を 0 に設定

(4) ストップ・コンディションを発行し、バスが解放されていることを確認(割り込みか SPD0 のセットで確認)

(5) スタート・コンディションを発行し、バスを占有(STD0 のセットを確認)

(6) スレーブのアドレスを送信し、存在を確認(割り込みで ACKD0 を確認し、ACK がなければ転送を中止)

(7) EEPROM のアドレスを送信

書き込みの場合には、さらに次の手順となります。

(8) 書き込むデータを送信して、割り込みで ACK を確認(ACK がなければ、転送動作を終了)

(9) ページ・ライトで書き込むデータが残っていれば、(7)を繰り返し

(10) ストップ・コンディションを発行して、書き込みを終了

読み出しの場合には、さらに次の手順となります。

(11) スタート・コンディションを発行して、データ転送を読み出し方向に設定

(12) 8 クロック・ウエイト、自動 ACK 応答をセット

(13) ダミー・データ(0FFH)を書いて、転送をスタート

(14) 受信完了したら、データを読み出し

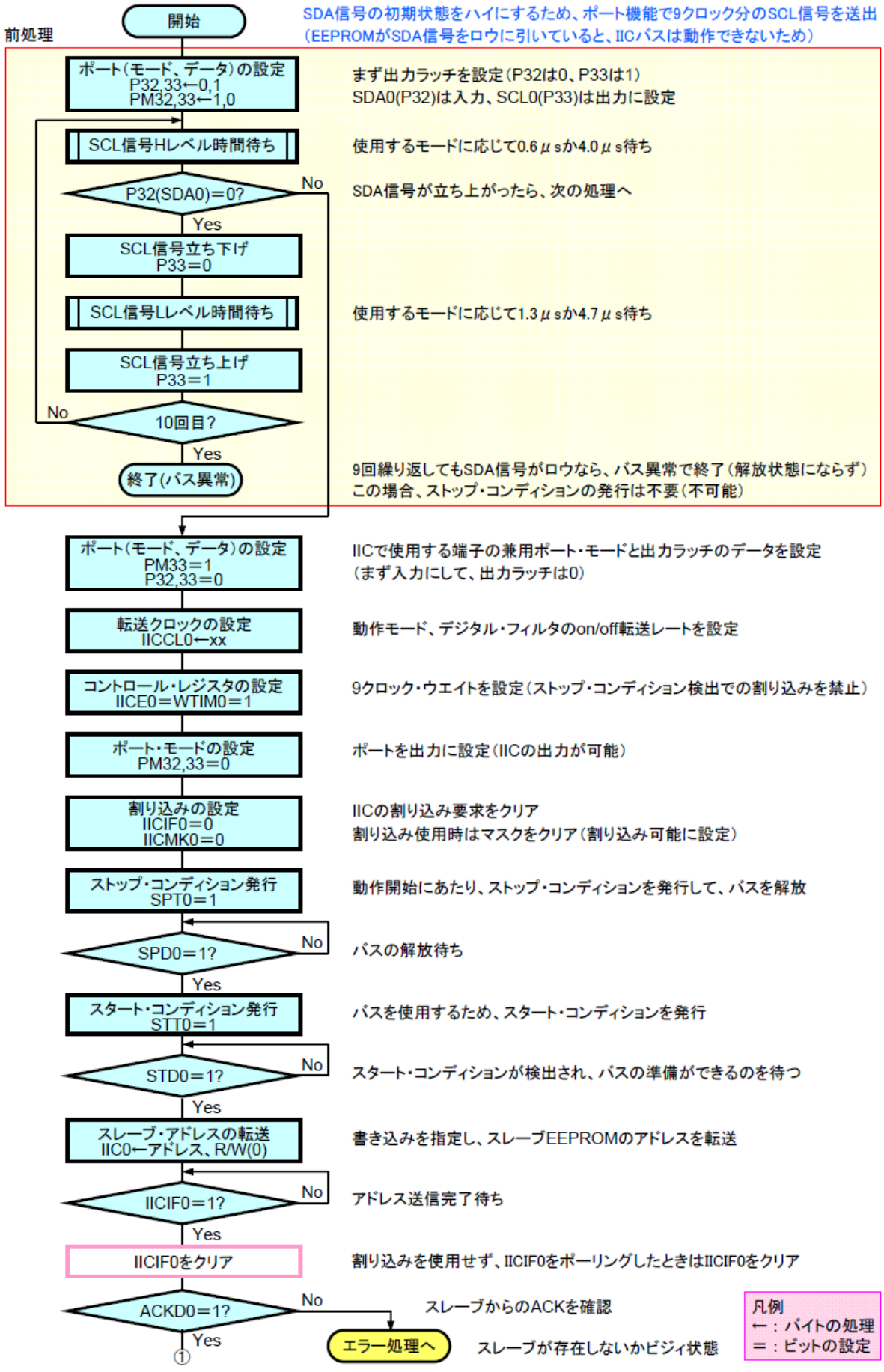
(15) 全データ受信完了で自動 ACK 応答を解除し、9 クロック・ウエイトにして、 8 クロック目でウエイト解除

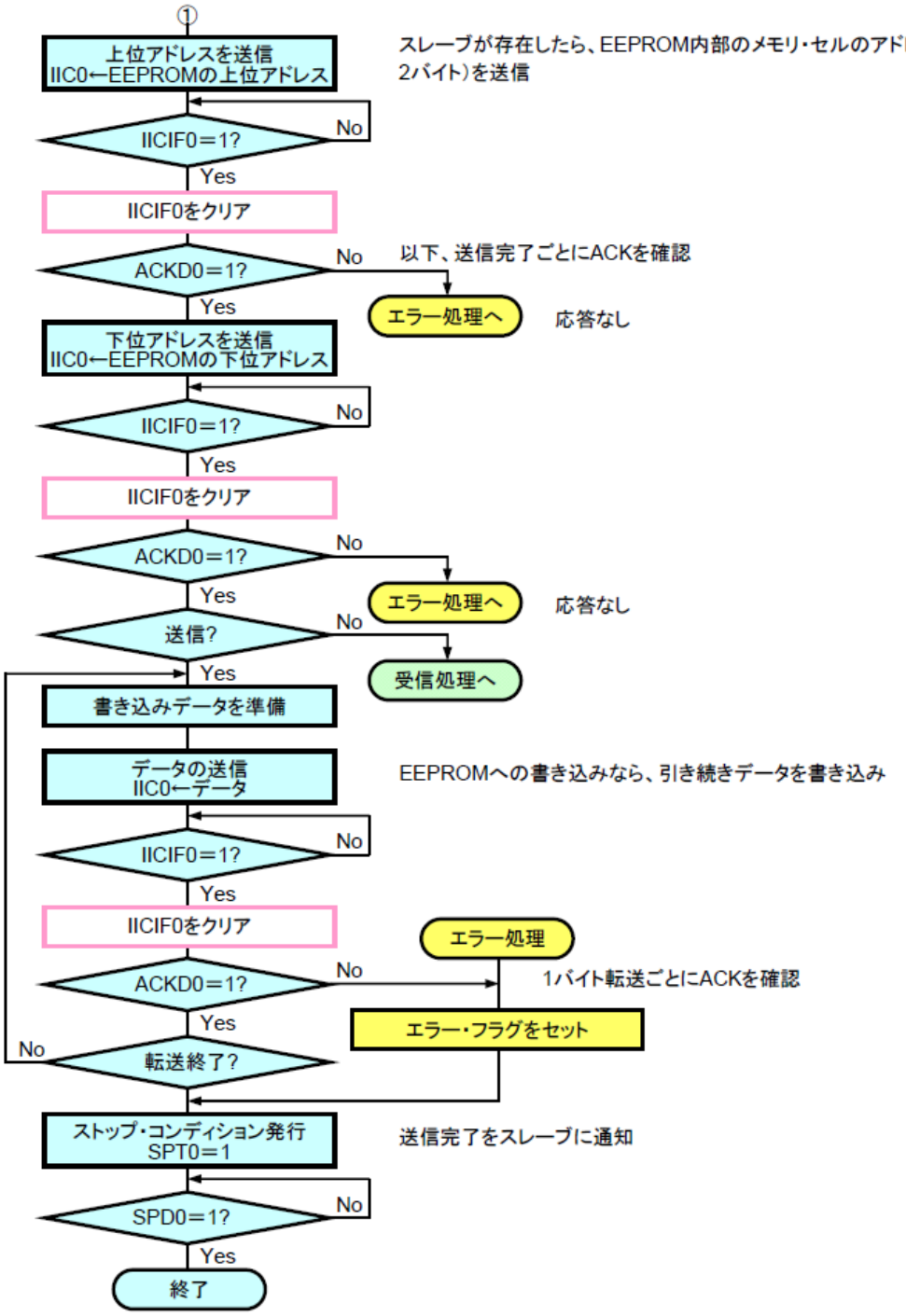
(16) ストップ・コンディションを発行

[処理フロー]

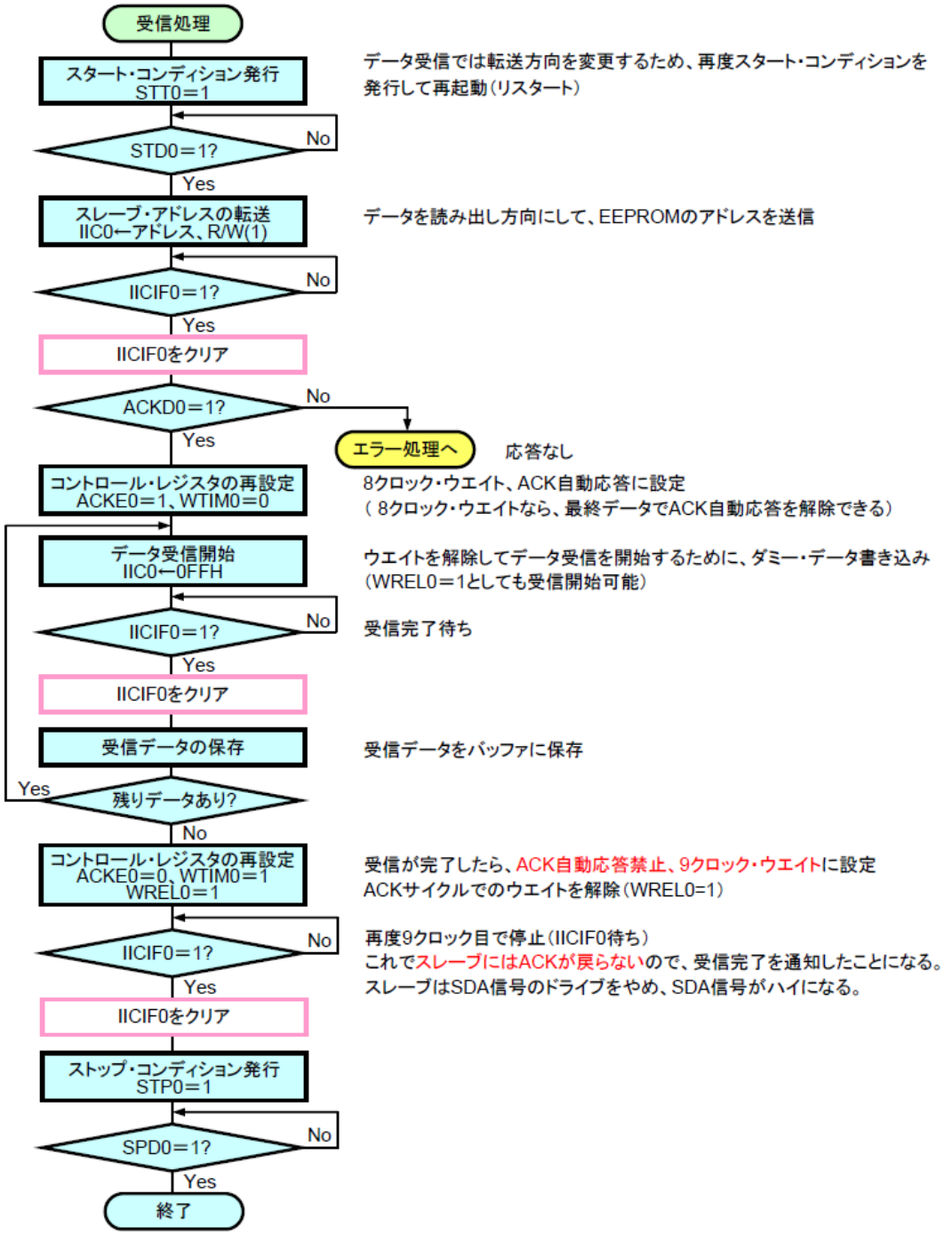
以上の処理フローを示します。簡略化し、データはすべて連続で読み出し／書き込みできるものとします。

μPD780034AY、780078Yサブシリーズを IICバスのマスタとし、スレーブのEEPROMなどを制御する手順





スレーブEEPROMは、ストップ・コンディションを検出したら、転送されたデータを書き込みます。
その間、自身のアドレスを受信してもACK応答しないので、書き込みに必要な時間を待ってから、次のブロックへのアクセスを行う必要があります。



[注意事項]

ピン数の制約で、通常シリアルEEPROMにはRESET端子がありません。電源投入時は、内部のパワー・オン・リセット回路で初期化され、バスを解放するので、問題は起こりません。通信中（EEPROMからの読み出し中）に、電源の変動などでマイコンにリセットがかかり、EEPROMはリセットがかからない場合などに、問題が発生します。このようなとき、EEPROMはデータを出し続けているので、たまたま0を出力していると、SDA信号はロウ・レベルに引かれた状態を保持します。IICバスでは、構造的に**マスタがデータ線を強制的にハイ・レベルにすることはできない**ので、ストップ・コンディションもスタート・コンディションも発行できなくなります。

この現象を回避する手段として、一部のEEPROMでは「マイコンのポートをクロック線に使用し、そこから**ダミー・クロック**を出力させ、9クロック入力することで、擬似的にEEPROMからの読み出しを進めて完了させる」といった対処方法が推奨されています。このときマスタは、EEPROMからの読み出し（受信）に対してACKを戻さないので、9クロックまでの間にEEPROMが出力をやめ、バスがハイ・レベルとなり、制御可能な状態になります。なお、このダミー・クロックを出力している間、データ線に対応したポートを必ずハイ・インピーダンス状態（ハイ・レベルを出力）にしておく必要があります。このための処理が、フロー・チャート冒頭の前処理です。

これと似たことが、マスタが受信を完了したときに、ACKを戻してしまった場合に発生します。

[注意点]

(1) ACK について

ACK の受信をループ待ちしてはいけません。I²C では ACK のタイミングが固定されており、そのタイミングで ACK が来ないと、ACK は認識されません。ACK が来なければ、通信は中止してください。

(1) 転送方向について

スレーブからデータを読み出す場合、スレーブ(EEPROM)に 2 バイトのアドレスをデータとして送信しますが、そのままデータの受信動作には移りません。I²C バスでは、このような動作はあり得ません。EEPROM のアドレスを指定したあと、リスタート処理を行って、データのリードで通信を再開する必要があります。使用される EEPROM の仕様をご確認ください。

EEPROM 読み出し後のスタート・コンディション発行ができない [共通]

Q1. μPD784031Y で、EEPROM の読み出しはできて、ストップ・コンディションを発行して処理を終了し、スタート・コンディションを発行しようとしましたが、SDA がロウ・レベルになりません。

A1. ストップ・コンディションを設定し、実際に発行されたことを確認せずに、スタート・コンディションを発行されているのではないかと思われます。ストップ・コンディションやスタート・コンディションは、実際に発行されたことを確認してから、次の処理に移ってください。

Q2. EEPROM の内容は正しく読み出せましたが、その後、ストップ・コンディションを発行できません。SDA 信号がロウ・レベルの状態で止まっているようです。

A2. EEPROM から最後のデータを読み出したときに、ACK を戻していないでしょうか。

スレーブ(EEPROM)は、マスタから ACK が戻るかどうかで転送を終了するか継続するかの判断を行います。そのため、マスタから ACK 応答があると、引き続き次のデータを送信しようとします。そのときの出力データがロウ・レベルであれば、マスタがストップ・コンディションを発行しようとしても、SDA がハイ・レベルにならないので、ストップ・コンディションを発行できません。

マスタで受信するときに、このような状態にならないようにするには、最後のデータを受信するときに、ACK 応答しないように設定しておく必要があります。この具体的な方法としては、次の 2 つが考えられます。

(1) **受信では 8 クロック・ウエイトを使う**(受信データで受信完了を判断するような場合)

- 最終データ以外の受信では ACK 応答に設定して、次の受信を起動します
- 最終データを受信したときには、ACK 応答しない設定で、9 クロック・ウエイトに変更して、8 クロック目でのウエイトを解除します。これで、SDA がハイ・レベルになり、9 クロック目でウエイト状態になるので、ストップ・コンディションを発行します。

(2) **受信でも 9 クロック・ウエイトを使う**

受信を起動する前に、残りデータ数をチェックして、次のような操作をします。

- 残りデータ数が 1 なら、ACK 応答しない設定にして、受信を起動します。
- 残りデータ数が 1 以外なら、ACK 応答する設定にして、受信を起動します。

データ受信が完了したら、全データの受信が完了したかどうかを確認します。完了していなければ、引き続き残りのデータ受信を継続します。全データの受信が完了していれば、通信を終了してストップ・コンディションを発行します。

78K0S 用ソフトウエア I²C バス通信プログラム [78K0S/Kx1+]

[はじめに]

I²C バスは、マイコンに EEPROM、表示コントローラ、A/D や D/A を接続するためのシリアル・バスとして、幅広く使用されています。マイコンの中には、78K0/Kx2 のように、専用のインタフェース回路を内蔵して、比較的容易に I²C バスを使用できる製品もあります。

ここでは、専用インタフェースを内蔵していない 78K0S で通常のポートを用いて、シングルマスタモードで I²C バスを制御するプログラムについて説明します。

[I²Cバスの概要]

I²C バスは、オープン・コレクタやオープン・ドレーン出力のクロック・ライン（SCL）とデータ・ライン（SDA）の 2 信号で通信を行います。2 つの信号線は抵抗でプルアップされており、未使用時はハイ・レベルになっています。I²C バスには、通信を制御するマスタと、マスタからの制御により通信を行うスレーブが存在します。

マスタが I²C バスを使用する手順は、次のとおりです。

①スタート･コンディションを発行してバスの使用を宣言
②最初にアドレスを送信して、通信相手のスレーブと通信方向を指定
③指定したスレーブが応答すると、通信を開始
④データを受信した方は、1 バイトごとに応答を戻す
⑤通信が完了したら、ストップ･コンディションを発行してバスを開放

I²C バスの詳細については、「I²C バスについての概要」をご参照ください。

ここでは、最大 400kbps までに対応するファースト・モードに準拠した動作を対象とします。

(1) 使用するポート

I²C バスは、オープン･ドレーン出力でドライブする必要があります。通常の CMOS 出力しかない場合は、ポートの使い方に工夫が必要です。具体的には、ポートの出力ラッチを 0 に固定しておき、入出力を切り替えることで処理します。つまり、次のような処理を行います。

- PM レジスタに 0 をセット → 出力ポート → ロウ・レベル
- PM レジスタに 1 をセット → 入力ポート → ハイ・レベル

このような処理により、バスへの出力と、ハイ・レベル出力状態でのバスの状態の読み出しが可能になります。

[I²C バスの主な動作／処理]

I²C バスのおもな動作を示します。ここで、送信はマスタからスレーブ、受信はスレーブからマスタへの通信です。

(1) バスの状態の確認

バス使用中にリセットがかかって、CPU が処理をやり直した場合など、スレーブがバス（SDA）をロウ・レベルに引いていることがあります。このような状態では、マスタはバスを使用できません。したがって、最初にバスが開放されているかどうかを確認する必要があります。

(2) バスの開放

バスが開放されていなければ、SDA がハイ・レベルになるまで、SCL にダミー・クロックを出力します。通常は 9 クロック（データ＋ACK 分）以下で、SDA はハイ・レベルになります。実際のプログラムでは、スレーブがウエイトをかけていた場合（ほとんどないと考えられます）、いつまでもバス開放を待つのはよくないので、通常のパルス幅のダミー・クロックを 256 クロック出力するまでに、SDA がハイ・レベルになるかどうかで判断します。SDA がハイ・レベルになったことを確認したら、ストップ･コンディションを発行して、バスを開放させます。
スレーブがハイ・レベルを出力しても、次のデータがたまたまロウ・レベルだった場合、SCL をロウ・レベルにすることで、スレーブが次のデータ（ロウ・レベル）を出力する可能性が考えられます。これを避けるため、ここでは SDA をロウ・レベルにしてから、再度ハイ・レベルに立ち上げます。SDA の立ち上げは、スタート･コンディションの発行と解釈されますが、その後ストップ･コンディションを発行するので、タイミング規格を満足します。

(3) スタート・コンディションの発行

バスが開放されたら、スタート･コンディションを発行して、バスの使用をスレーブに通知します。

(4) スレーブ・アドレスの送信

実際の通信の先頭では、スレーブのアドレス（7 ビット）と通信方向（1 ビット）の合計 8 ビットのデータを送信します。送信したアドレスに合致したスレーブが、ACK を戻さない場合は、該当するアドレスのスレーブが存在しないので、ストップ･コンディションを発行して通信を終了します。ACK が戻ると、通信方向に従って通信を継続します。

(5) サブアドレスの送信

I²C バスとしては、スレーブ・アドレスを送信して、通信相手のスレーブを指定できますが、スレーブにも内部のアドレス情報をもつものがあります。たとえば EEPROM では、内部のアドレスを指定するために、スレーブ・アドレスの送信に続いて、内部のアドレスを指定します。容量が 256 バイト以下なら、スレーブ内部のアドレスは、1 バイトの指定です。通常、ここで指定したアドレスは保持され、読み書きを行うごとに更新されます。

(6) データの送信

サブ・アドレスの送信に続いてデータを送信すると、指定したサブ・アドレスに、そのデータが書き込まれます。EEPROM では、ストップ･コンディションを受けてから、実際の書き込みが行われます。
サブ・アドレスとデータを送信し、スレーブがデータを正しく受け取ると ACK を戻してきます。ACK が戻れば、次の処理に移りますが、ACK が戻らない場合は通信を終了し、ストップ･コンディションを発行して通信を終了します。

(7) データの受信

通信方向で受信を指定した場合は、スレーブ・アドレスの送信の次からデータ受信を行います。データを受信したマスタは、続けてデータを受信したい場合、ACKを戻します。受信したデータが最後で、それ以上のデータを受信しない場合は NACK を戻して、受信が完了したことをスレーブに示す必要があります。

(8) ストップ･コンディションの発行

通信を完了してバスを開放する場合は、SCL がハイ･レベルの状態で SDA をハイ・レベルに立ち上げれば、ストップ･コンディションの発行になります。

(9) リスタート・コンディションの発行

バスの使用権を確保している状態で、データ転送方向を切り替える場合は、再度スタート・コンディションを発行します。これがリスタートで、この直後はスレーブ・アドレスの送信となり、通信方向を再指定できます。

一般的には、内部アドレスを指定して、スレーブからデータを読み出す場合に使用します。通常は、最初に送信方向でスレーブ・アドレスを送信して、スレーブ内のアドレスをサブ・アドレスとして指定します。次に、リスタート・コンディションを発行し、さらに受信方向を指定して、スレーブ・アドレスを送信します。これで、スレーブ内の指定したアドレスからの読み出しが可能となり、以降は指定したアドレスから順次読み出しができます。

[バスの状態(参考)]

バスのおもな状態は、次のようになります。

	状　　態	SCL	SDA
①	バスは未使用	H	H
②	データ変化時	L	データは変化
③	データは安定	H	H/L
④	スタート･コンディション発行後	H	L
⑤	アドレス送信直後(9 クロック・ウエイト状態)	L	ACK→H
⑥	9 クロック・ウエイト状態(次の通信待ち状態)	L	次データ
⑦	アドレス送信後(受信指定で通信開始)	L	受信データ
⑧	アドレス送信後(送信指定で通信開始)	L	送信データ
⑨	8 クロック・ウエイト状態	L	L＝ACK／H＝NACK
⑩	ストップ・コンディション発行後(バスは未使用)	H	H

- バスが使用されていない状態(①⑩)では、2 つの信号線 SCL と SDA はハイ・レベル状態です。
- 基本的に SDA は、SCL＝L のときに変化して、SCL＝H の状態では安定しています。
- スタート･コンディションの発行(SCL＝H の状態で、SDA を立ち下げる)後は、SCL＝H で、SDA＝L の状態となります。このあと、スレーブのアドレスを送信するには、まず SCL を立ち下げます。
- アドレスの送信やデータ通信で、SCL が 9 クロック送られたあとの⑤や⑥では、SCL＝L の状態で次の通信の準備を行います。次の通信がない場合、SDA＝Hとなっています。そうでない場合(⑦⑧)には、次のデータの先頭ビットが出力されます。
- 8 ビット・データの送受信後、⑨で SCL=L となり、ACK 応答待ちの状態となります。
- ⑥や⑨で、スレーブは SCL をロウ・レベルに引くことで、通信を待たせることができます。したがって、マスタは SCL がハイ・レベルになったことを確認してから、次の処理に移る必要があります。

[プログラム例]

基本的な制御は、アセンブラのプログラムで実現しますが、これらはC記述のプログラムから呼び出せるようにしてあります(あくまでも参考プログラムです)。

このプログラムでは、78K0S/KY1+を高速内蔵発振器で動作させ、P2.2 を SDA 信号用に、P2.3 を SCL 信号用に使用します。なお、P2 のほかのビットを操作すると、P2.2 と P2.3 の出力ラッチが書き換わってしまいます。P2 の他のビットを出力ポートとして使用する場合は、出力ラッチのデータのイメージを内蔵 RAM に準備しておきます。ポートの出力を操作する場合は、RAM のデータを操作し、結果をポートに書き込むようにしてください。

【ひとこと注意 1】
処理速度が要求される場合は、プログラムをアセンブラで記述し、大きな制御では C 言語で記述することがよくあります。この場合、C 言語のプログラムからアセンブラ記述のサブルーチンを関数としてコールします。そのときに、サブルーチンの名前の付け方に注意が必要です。アセンブラ記述のサブルーチンの名前は、__(2 つのアンダ・スコア)で始まる必要があります。C 言語のプログラムからこれを呼び出すには、_(1 つのアンダ・スコア)で始まるように指定します。
たとえば、I²C で使用するポートの初期化プログラムは、アセンブラ記述では「__setup_i2c_port」の名前が付けられていますが、これを C 言語のプログラムで呼び出すには「_setup_i2c_port();」と記述します。

また内蔵 RAM には、実行結果のエラー・ステータス用のバイト変数(i2c_error)と、スレーブに対して ACK を戻すかどうかを指定するビット変数(acke)を準備します。次に宣言の例を示します。これらの変数は、すべて専用のプログラムで制御するものとします。

```
comdata DSEG     SADDR      ; saddr 領域に変数を確保
i2c_error:  DS   1          ; エラー・フラグ
    BSEG                    ; ビット変数領域であることを示す
acke    DBIT                ; ACK 制御フラグ
```

(1) ポートの定義

PMSW_SDA が SDA 信号用に使用する名称です。これを 1 にすると端子は入力ポートとなり、PSW_SDA を読むことで、SDA 信号の状態を知ることができます。スレーブが出力していなければ、SDA 信号はハイ・レベルになります。これを 0 にすると出力ポートとなり、出力ラッチが 0 なので、SDA 信号をロウ・レベルにします。
PMSW_SCL が SCL 信号用に使用する名称です。これを 1 にすると端子は入力ポートとなり、PSW_SCL を読むことで、SCL 信号の状態を知ることができます。通常は、SCL をハイ・レベルにするときに、1 に設定します。スレーブがウエイトをかけてなければ、外部のプルアップ抵抗でハイ･レベルに引かれます。これを 0 にすると出力ポートとなり、SCL 信号をロウ・レベルにできます。

・アセンブラでの定義例(使用ポートの定義)

```
PMSW_SDA    equ PM2.2       ; SDA 信号制御で使用
PSW_SDA     equ P2.2        ; SDA 信号の読出しで使用
PSW_I2C     EQU P2          ; 使用するのはポート 2

PMSW_SCL    equ PM2.3       ; SCL 信号制御で使用
PSW_SCL     equ P2.3        ; SCL 信号の読出しで使用
```

実際のポートの操作(モード・レジスタの操作)では、分かりやすくするため、次のようなマクロを定義しています。

・ポート操作の定義

```
_scl_hi         macro       ; _scl_hi というマクロの定義を示す
        set1    PMSW_SCL    ; PM2.3 をセットして入力にする
        endm                ; マクロ定義の終了を示す

_scl_lo         macro
        clr1    PMSW_SCL    ; PM2.3 をクリアして SCL にロウ出力
        endm

_sda_hi         macro
        set1    PMSW_SDA    ; PM2.2 をセットして入力にする
        endm

_sda_lo         macro
        clr1    PMSW_SDA    ; PM2.2 をクリアして SDA にロウ出力
```

【ひとこと 1】
プログラムを作成する場合、マクロ機能を用いれば、定型的な処理を効率的に記述したり、プログラムを分かりやすくしたりできます。ここでは、処理そのものは、ポート・モード・レジスタに対するビット操作命令です。効率的な記述より、処理の分かりやすさのためにマクロ機能を使用しています。「CLR1 PMSW_SCL」と記述するより、「_scl_lo」と記述する方が、どのような処理を行うか、直感的に意味をとらえることができます。

(2) ポートの初期化(_setup_i2c_port)

使用するポートの出力ラッチを 0 にします(0 をバイト単位で書き込み)。ポートのモードは、入力にしておきます。
このプログラムは、C 言語記述部分から呼び出せるように、_で始まる名前をつけて public 宣言します。

```
    public  __setup_i2c_port    ; 外部から使えるように public 宣言する
__setup_i2c_port:
    MOV PSW_I2C,#0              ; 端子をオープン・ドレーンと等価に
                                ; 使用するのでデータは 0 に固定
    _scl_hi                     ; PM を入力にする
    _sda_hi
    RET
```

(3) バスのビジィ・チェック(_i2c_busy)

バスが空き状態かどうかをチェックします。結果はキャリー・フラグで戻ります。自分自身で使用中を含めて、SCL か SDA がロウ・レベルならビジィと判断し、キャリー・フラグはセット(戻り値は true)されます。バスが開放されていれば、キャリー・フラグはクリア(戻り値は false)されます。ポートの状態を変えるような処理は行いません。
ここではアセンブラ記述で、参照しやすいように専用に名前(i2c_busy)を追加します。

```
;bit    _i2c_busy(void)
    public  __i2c_busy
__i2c_busy:
i2c_busy:
    SET1    CY                  ; ビジィ・フラグをセットしておく
    BF  PSW_SDA,$busbusy        ; SDA ラインのチェック
    BF  PSW_SCL,$busbusy        ; SCL ラインのチェック
    CLR1    CY                  ; 開放されていればフラグ・リセット
busbusy:
    RET
```

【ひとこと 2】
78K0S で 1 ビットのポートの状態をチェックするときは、前述のプログラム例のように条件分岐命令（BF 命令や BT 命令）を使用します。BF 命令を使用すると、1 ビットのポートが 0 なら分岐し、1 なら分岐しません。このプログラムでは、PSW_SDA と PSW_SCL ともに1なら、2 つの条件分岐命令では分岐せず、CY（キャリー・フラグ）がクリアされます。

(4) バスの開放（_free_i2c_bus）

バスの状態を確認し、開放されてなければ、ダミー・クロックを送って、バスが開放されるのを待ちます。256 パルスを送ってもバスが開放されなければ、エラーとしてキャリー・フラグをクリア（戻り値は false）して戻ります。
SDA 信号と SCL 信号がハイ・レベルになれば、ストップ・コンディションを発行します。

【ひとこと注意 2】
この状態で SCL をロウ・レベルにすると、スレーブが次のデータとしてロウ・レベルを出力する可能性があります（スレーブ出力データが 1 で、次のデータが 0 の場合）。そこで、ここでは SCL がハイ・レベルの状態で、SDA をロウ・レベルにしてから、再度ハイ・レベルに立ち上げます。SDA の立ち下げはスタート・コンディションの発行と解釈されますが、あとで SDA を立ち上げてストップ・コンディションを発行するので、無視されます。

ストップ・コンディションを発行してバスを開放したら、キャリー・フラグをセット（戻り値は true）して戻ります。
ここで使用するダミー・クロック送信では、SCL のロウ・レベル幅 1.3 μs を確保するため、動作クロック 12 クロックの時間を確保しています。ハイ・レベル幅はサブルーチン・コールなどのオーバヘッドがあるので、意識して確保はしません。スレーブがウエイトをかけている場合は、マスタが立ち上げても、バス上ではロウ・レベルのままになってしまいます。この状態が継続すると、I²C バスは全く使えません。そのため、ここでは SCL の立ち上がりを待つことはしません。マージンをもたせた 256 回で処理を終了します。

```
;bit    _free_i2c_bus(void)
    public  __free_i2c_bus
__free_i2c_bus:
free_i2c_bus:
    _sda_hi                 ; 念のために SDA を立ち上げる
    _scl_hi                 ; 念のために SCL を立ち上げる
    CALL    !i2c_busy       ; バスの状態確認
    NOT1    CY
    BC  $free_exit1         ; バスが開放されていれば抜ける
    MOV A,#0                ; 限度を 256 回に設定
dumyclkloop:
    CALL    !scl_pulse      ; SCL にダミー・クロックを出力
    CALL    !i2c_busy       ; バスの状態確認
    NOT1    CY
    BC  $free_exit1         ; バスが開放されていれば抜ける
    DEC A                   ; 限度を越えていないか？
    BNZ $dumyclkloop        ; 限度以内なら継続
    RET                     ; エラーならキャリーをセットして戻る
free_exit1:
    PUSH    PSW             ; 念のためにフラグをセーブ
    CALL    !!I2C_STPR      ; ストップ・コンディション発行
    POP PSW
    RET
```

・I²C バスの規格に準拠したダミー・クロックを生成

```
scl_pulse:
    _scl_lo                 ; 6:SCL を立ち下げる
    NOP                     ; 2:ロウ・レベル幅 1.3μs を
    NOP                     ; 2:確保するために、11 クロック
    NOP                     ; 2:以上時間を確保して立ち上げる
    _scl_hi                 ; 6:SCL を立ち上げる
    RET                     ; 6:これだけでハイ・レベルは十分
```

(5) スタート・コンディションの発行(_I2C_STR)

スタート・コンディション発行の準備で、SDA を立ち上げてから、SCL を立ち上げます。すでに SCL が立ち上がっていれば、ストップ・コンディションと同じです。リスタートでは、SCL がロウ・レベルになっている必要があります。
その後、SCL が立ち上がっていることを BF 命令で確認してから、SDA を立ち下げることでスタート・コンディションを発行します。この処理は、スレーブが 9 クロック目でウエイトをかけていたときに、解除されるのを待つためで、ほとんどの場合は不要と考えられます。BF 命令を削除する場合は、2 つの NOP 命令に置き換えてください。

【ひとこと 3】
I²C バスでは、スレーブが SCL をロウ・レベルに引いて、通信を一時的に待たせる機能があります。そのため、マスタは SCL を立ち上げたあとで、SCL が立ち上がったことを確認します。そのため「BF PSW_SCL,$$」を使用しています。この命令により、SCL ライン(PSW_SCL で読める)がロウ・レベルであれば、自分自身に分岐して命令を繰り返します。これにより、SCL が立ち上がると、次に進むことができます。

```
    public  __I2C_STR
__I2C_STR:
I2C_STR:
    _sda_hi                 ; 6:準備のため SDA をハイに設定する
    NOP                     ; 2:
    _scl_hi                 ; 6:SCL をハイに設定する
    BF   PSW_SCL,$$         ;10 : SCL の立ち上がり検出
    _sda_lo                 ; 6:スタート・コンディション発行
    RET                     ; 6:
```

(6) ストップ・コンディションの発行(_I2C_STPR)

最初に SCL を立ち下げると、次の転送がスタートするかもしれないので、すでに SCL がロウ・レベルとして処理します。SCL がハイ・レベルなら、SDA をロウ・レベルにすると、スタート・コンディションの発行と同じです。でも、次にストップ・コンディションを発行することで、おかしな状態にはならないと考えられます。
ここでも、SCL が立ち上がっていることを BF 命令で確認してから、次の処理をします。これは、スタート・コンディションの発行と同じ理由によるものです。

```
    public  __I2C_STPR
__I2C_STPR:
I2C_STPR:
    _sda_lo                 ; 6:SDA を立ち下げる (これでスタート・
    NOP                     ; 2:コンディションと解釈されても、
    NOP                     ; 2:すぐにストップ・コンディションを
                            ;   発行するので構わない)
    _scl_hi                 ; 6:念のために SCL をハイに設定する
    BF   PSW_SCL,$$         ;10 : SCL の立ち上がり検出
    _sda_hi                 ; 6:ストップ・コンディション発行
```

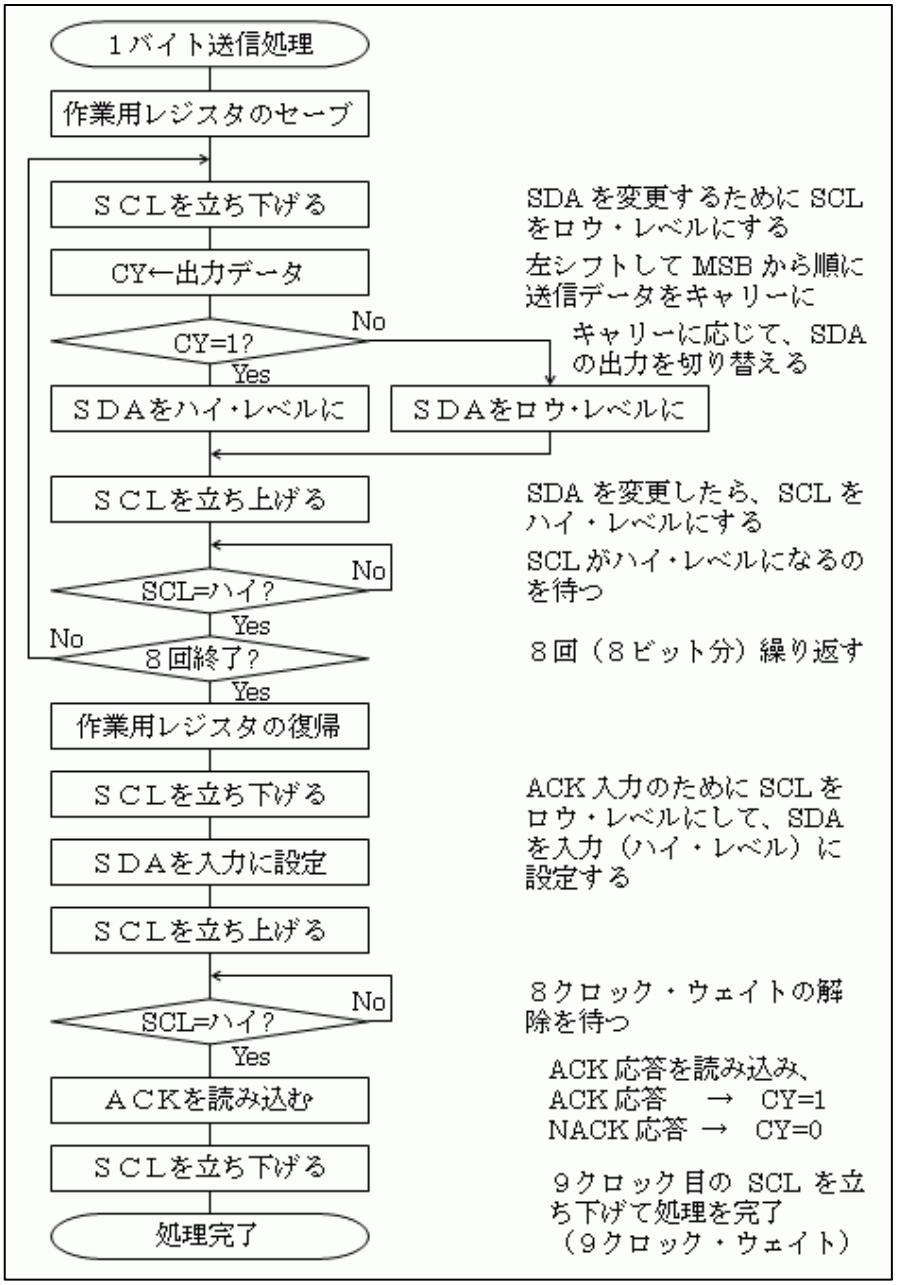
```
NOP                    ; 2:
NOP                    ; 2:
RET                    ; 6:
```

(7) I²C バスに 8 ビット・データを出力（_put_i2c）

C 言語のノーマル・モードで、X レジスタ（1 バイトの引数の受け渡しに使用）で渡された 8 ビット・データを I²C バスに出力するルーチンです。SCL がロウ・レベルのときに、MSB から順に出力して、ACK 応答を確認し、応答があればキャリー・フラグをセット（戻り値は true）して戻ります。ACK 応答がなければ、キャリー・フラグをクリア（戻り値は false）して戻ります。処理が終わった時点では、SCL はロウ・レベルになった状態です（9 クロック・ウエイト状態）。

【ひとこと 4】
I²C バスでは、スレーブが SCL をロウ・レベルに引いて、通信を一時的に待たせる機能があります。そのため、マスタは SCL を立ち上げたあとで、SCL が立ち上がったことを確認します。そのため「BF PSW_SCL,$$」を使用しています。この命令により、SCL ライン（PSW_SCL で読める）がロウ・レベルであれば、自分自身に分岐して命令を繰り返します。これにより、SCL が立ち上がると、次に進むことができます。

・プログラム・フロー



・プログラム例

```
;bit    _put_i2c(u8 data);
    public  __put_i2c
__put_i2c:
put_i2c:
    PUSH    BC              ; ループ・カウンタで使用のためセーブ
    MOV B,#8
    MOV A,X                 ; 入力データ(引数)は 8 ビット
;
;
OUTLOOP:
    _scl_lo                 ; 6：SCL を立ち下げてデータ出力準備
    NOP                     ; 2：立下りの時間調整用
    ROL A,1                 ; 2：MSB より CY にシフト・アウトする
    BNC $OUTLO              ; 6：データが 0 なら分岐
    _sda_hi                 ; 6：1 を出力
    BR  $CLKHI              ; 6：SCL の立ち上げへ
OUTLO:  _sda_lo             ; 6：0 を出力
    NOP                     ; 2：セットアップ時間確保
CLKHI:
      _scl_hi               ; 6：
    BF  PSW_SCL,$$          ;10：SCL の立ち上がり検出(念のため)
    DBNZ    B,$OUTLOOP      ; 6：

    POP BC                  ; 6：
    _scl_lo                 ; 6：8 クロック目の SCL を立ち下げる
    NOP                     ; 2：
    _sda_hi                 ; 6：SDA を入力に切替え
    NOP                     ; 2：
    NOP                     ; 2：
    NOP                     ; 2：
    _scl_hi                 ; 6：9 クロック目の SCL を立ち上げ
;
;   ウエイト解除を待つ
;
    SET1    CY              ; 2：

    BF  PSW_SCL,$$          ; SCL の立ち上がり検出待ち

    BF  PSW_SDA,$ACKOK      ;10：/ACK 信号を取得
    NOT1    CY              ; 2：
ACKOK:

    _scl_lo                 ; 6：SCL を立ち下げ(9 クロック目)
    NOP
    RET
```

(8) I²C バスに 8 ビット・データを出力、エラーでバスを開放(_put_i2c2)

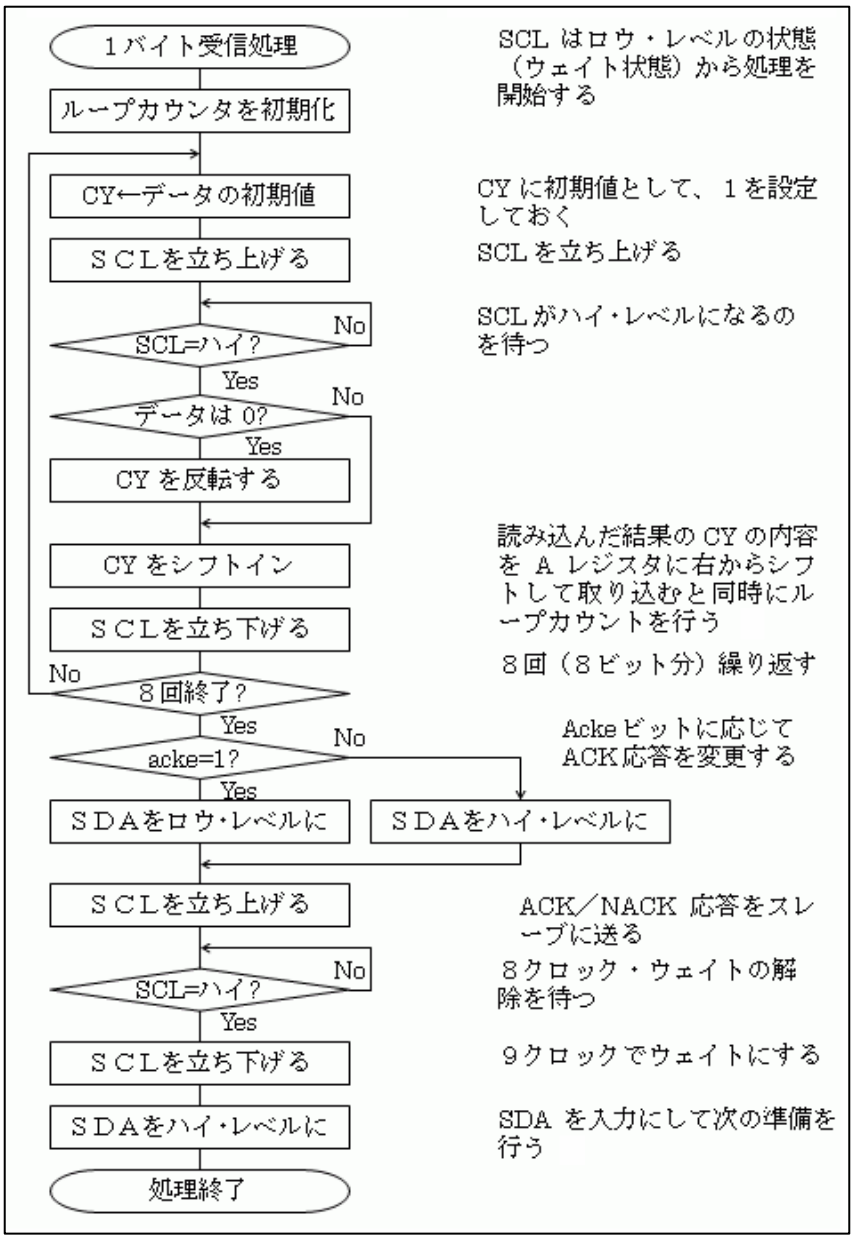
これは、(7)のルーチンでデータを出力し、エラーが発生したらストップ・コンディションを発行して戻る処理です。入出力条件は(7)と同じです。

```
;static bit  _put_i2c2(u8 data)
    public  __put_i2c2
__put_i2c2:
    CALL    !put_i2c            ; 1バイト出力
    BNC $PUTERROR               ; エラーなら終了処理へ
    RET                         ; 正常なら終了
PUTERROR:
    CALL    !I2C_STPR           ; ストップ・コンディション発行
    CLR1    CY                  ; falseにする(念のため)
    RET
```

(9) I²C バスから 8 ビットデータを入力（_get_i2c）

選択された I²C バスのスレーブから 8 ビット・データを受信し、Cレジスタに格納して戻ります。スレーブからのデータは、MSB から順に SCL が H のときに読み込みます。受信完了後に、acke ビットが 1 なら ACK 応答を行い、次のデータがあることをスレーブに通知します。acke ビットが 0 なら NACK 応答を行い、最後のデータを受信したことをスレーブに通知します。処理が終わった時点では、SCL はロウ・レベルの状態です（9 クロック・ウエイト状態）。

・プログラム・フロー



・プログラム例

```
;u8 _get_i2c(void);
    public  __get_i2c
__get_i2c:
    MOV A,#11111110b        ; シフトして CY=0 で終了
INLOOP:
    NOP                     ; 2:
    SET1    CY              ; 2:初期値として 1 をセット
    _scl_hi                 ; 6:SCL の立ち上げ
    BF  PSW_SCL,$$          ; SCL の立ち上がり検出

    BT  PSW_SDA,$DATAIS1    ;10:
    NOT1    CY              ; 2:データが 0 ならキャリーを反転
DATAIS1:
    ROLC    A,1             ; 2:読み込んだ結果を A レジスタにシフト・イン
    _scl_lo                 ; 6:SCL の立ち下げ
    BC  $INLOOP             ; 6:8 ビット分ループ

;*********************************************************
;*  ACK 応答の制御                                       *
;*********************************************************
    BF  _acke,$ACKEND           ;10:ACK 応答するかをチェック
    _sda_lo                     ; 6:ACK を戻す
ACKEND:
    _scl_hi                     ; 6: 9 クロック目の立ち上げ
;
;   ここではスレーブが 8 クロック・ウエイトしていたときの対応で
;   SCL 信号の立ち上がりを待っている。通常は不要。
;
    BF  PSW_SCL,$$              ; SCL の立ち上がり検出(念のため)
    NOP                         ; 2:
    _scl_lo                     ; 6:9 クロック目の立ち下げ

    MOV C,A                     ; 4:return (u8)
    _sda_hi                     ; 6:終わったら SDA は入力にしておく
    RET
```

【ひとこと 5】
ここでは、ループをカウントするのに A レジスタを使用しています。A レジスタへの受信データ(ビット)の取り込み(右からのシフト・イン)処理と、ループ・カウント動作を兼用させています。A レジスタに、初期値として 11111110b を設定しておくと、シフト・インごとに 1111110xb→111110xxb→11110xxxb と変化します。7 回目が完了した段階では 0xxxxxxxb となり、ここまでのシフト結果では、キャリー・フラグがセットされています。8 回目のシフトを行うと、初期値の最後のビットの 0 がキャリーに送られます。したがって、キャリーが 0 になったら 8 回が完了したことになります。

(10) エラー・フラグのクリア(_clear_i2c_error)

内部の変数をクリアするだけの処理です。エラー・フラグの状態を戻り値とします。

```
;static u8 _clear_i2c_error(void)
    public  __clear_i2c_error
__clear_i2c_error:
clear_i2c_error:
    MOV i2c_error,#I2C_NO_ERROR ;
    MOV C,#I2C_NO_ERROR
    RET
```

(11) エラー・フラグのセット(_set_i2c_error)

内部の変数にエラーの状態をセットするものです。エラーは、次のように定義しています。

```
I2C_NO_ERROR        = 0x00 : エラーなし
I2C_BUS_NOT_FREE    = 0x10 : バスが開放されていなかった
I2C_ADDR_NACK       = 0x11 : スレーブがアドレスに反応しなかった
I2C_RADDR_NACK      = 0x12 : スレーブが READ アドレスに反応しなかった
I2C_REG_ADDR_NACK   = 0x13 : スレーブから予期せぬ NACK が返って来た（レジスタ）
I2C_DATA_NACK       = 0x14 : スレーブから予期せぬ NACK が返って来た（データ）

;static u8 _set_i2c_error(u8 error)
    public  __set_i2c_error
__set_i2c_error:
set_i2c_error:
    XCH A,X                     ; エラー・データを取り込む
    MOV i2c_error,A             ; エラー・フラグにセット
    MOV C,A                     ; 戻り値をセット
    XCH A,X
    RET
```

(12) エラー・フラグの読み出し(_get_i2c_error)

エラー・フラグの内容を読み出します。エラーは、(11)のように定義しています。

```
;static u8 _get_i2c_error(void)
    public  __get_i2c_error
__get_i2c_error:
get_i2c_error:
    PUSH    AX
    MOV A,i2c_error             ; エラー・フラグの読み出し
    MOV C,A                     ; 戻り値にセット
    POP AX
    RET
```

(13) ACK 応答許可フラグのセット(_ACK_EN)

マスタ受信での ACK 応答を許可する設定をします。通常は許可にしておき、最後のデータ受信時に禁止します。

```
;void _ACK_EN(void)
    public  __ACK_EN
__ACK_EN:
ACK_EN:
    SET1    acke                ; ACK 応答フラグセット
    RET
```

(14) ACK 応答フラグの禁止セット(_ACK_DS)

マスタ受信の最後のデータで、NACK 応答させる場合に使用します。

```
;void _ACK_DS(void)
    public  __ACK_DS
__ACK_DS:
ACK_DS:
    CLR1    acke                ; ACK 応答フラグクリア
    RET
```

(15) マスタ送信処理(_write_i2c_block)

I²C バスを用いて、256 バイトの EEPROM への書き込みをします。このサブルーチンは、C 言語(ノーマル・モード)での利用を考慮して、必要なパラメータは C 言語の引数と同じ形式で渡すものとします。引き渡すパラメータは、次の形式となります。

第 1 の引数＝X レジスタ　:I2C バスのスレーブのアドレス

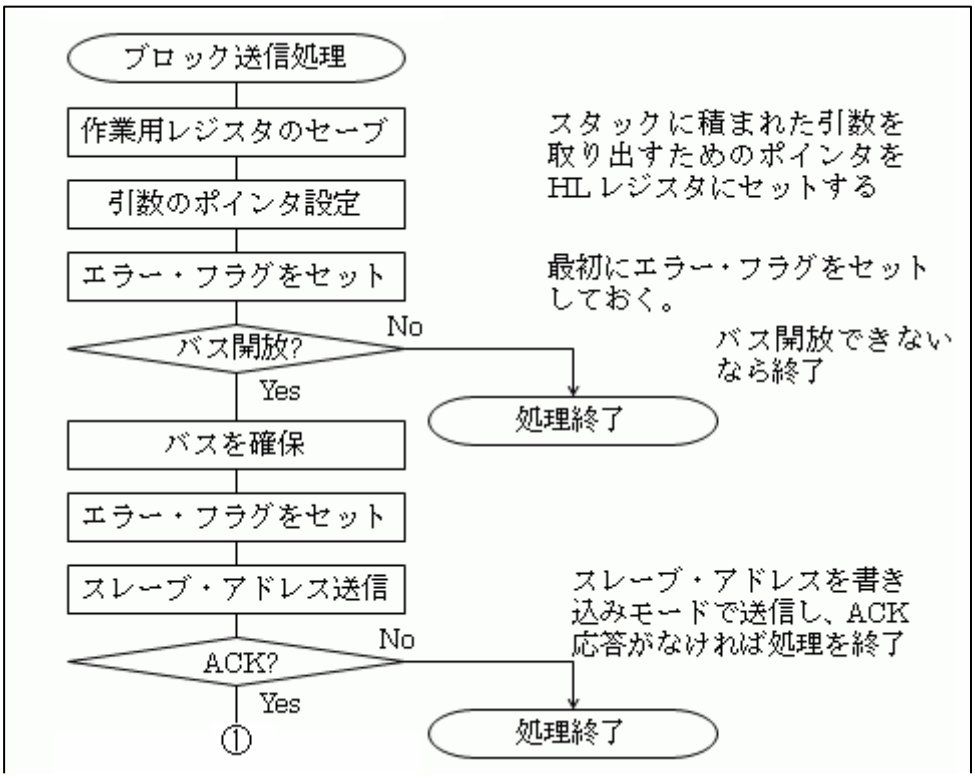
第 2 の引数＝スタックの 1 段目　:スレーブのアクセスしたい内部アドレス

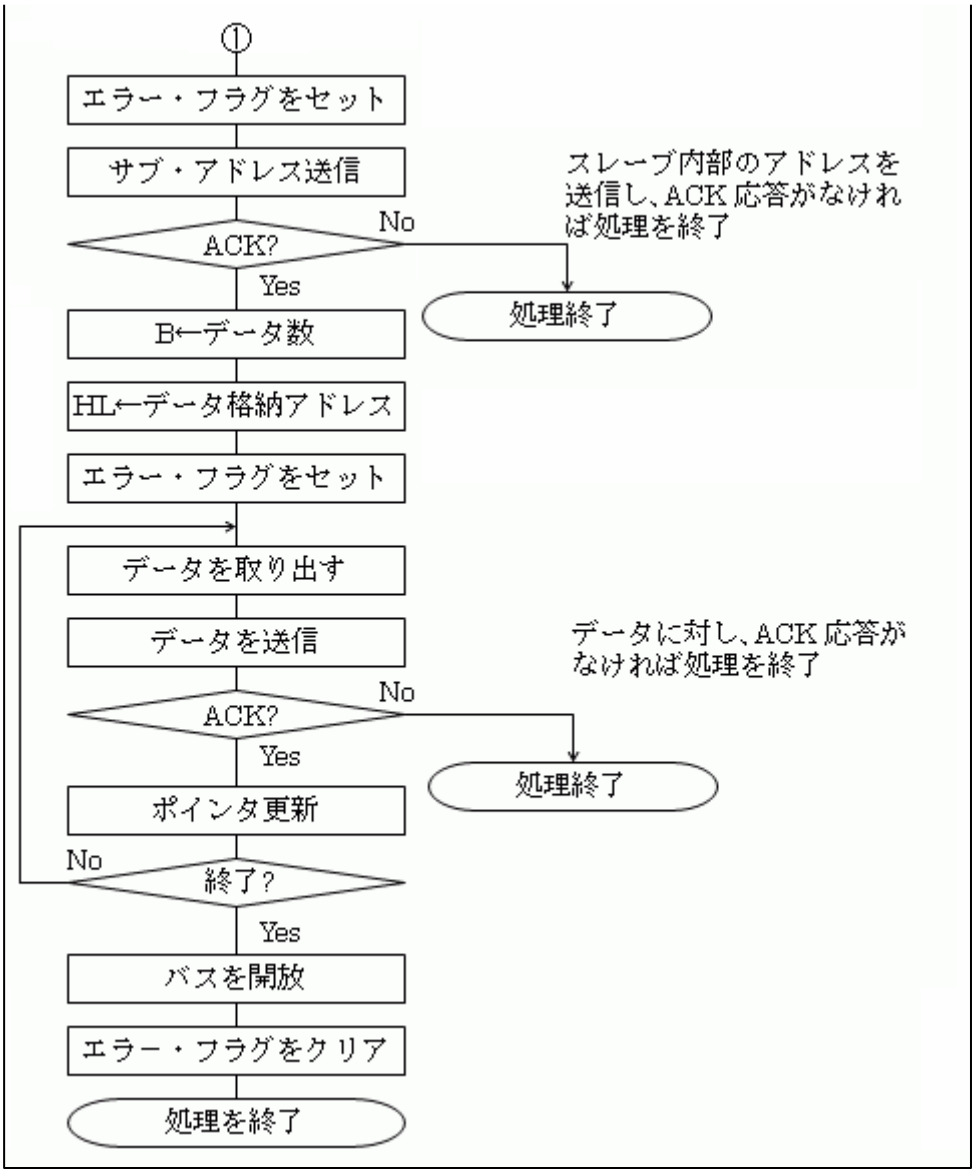
第 3 の引数＝スタックの 2 段目　:データを保存するバッファのアドレス

第 4 の引数＝スタックの 3 段目　:書き込むデータのバイト数

これらのパラメータを 2 バイト単位で、第 4 の引数～第 2 の引数をこの順にスタックにプッシュしてから、このサブルーチンをコールします。このサブルーチンは、第 1 の引数で示されたスレーブ・アドレスの EEPROM の、第 2 の引数で示された内部アドレス以降に、第 3 の引数で示されたバッファ・メモリから、第 4 の引数で示されたバイト数のデータを書き込みます。正常に書き込みが完了したら、バスを開放して、エラー・フラグをクリアして戻ります。エラー・フラグと同じ値を戻り値として、C レジスタに格納して戻るので、戻り値をチェックすることで、エラーを確認できます(エラーの内容は「**(11) エラー・フラグのセット**」参照)。

・プログラム・フロー





バイト単位の送信処理には、アセンブラ記述の_put_i2c2 関数を使用し、エラー発生の場合にはバスを開放して、発生状況に応じたエラーを戻します。エラーが発生しなかった場合は、バスを開放して、エラーなしを戻します。
EEPROM はストップ･コンディションを検出して、送信されたデータを内部のセルに書き込みます。
このプログラム例と同じ処理を行うプログラムをC言語で記述した例を**[プログラムの使用例]**の(1)に記載します。
細かな処理の順序は異なりますが、処理の内容は同じです。関数としての呼び出し方法と、戻り値も同じです。

・プログラム例

```
;*****************************************************************
;*                                                               *
;*  I2C デバイスにブロック・データ送信                           *
;*  対象は 256 バイト以下のサブアドレスもつデバイス              *
;*                  ADR(W), SUB_ADR, DATA...                     *
;*****************************************************************
;*  [i] : u8 i2c_adr    ... I2C スレーブのアドレス                *
;*  [i] : u8 reg_adr    ... スレーブ内部のアドレス                *
;*  [i] : u8 *data      ... 書き込みデータのアドレス              *
;*  [i] : u8 size       ... バイト数(0x00 = 256)                  *
;*  [r] : u8            ... error code                           *
```

```
;*                                                                  *
;*  実際の引数はスタックに 16 ビット単位で積まれて                  *
;*  渡されるので、SP の値を HL に持ってきて、HL 相対に              *
;*  より必要な引数を得る                                            *
;*                                                                  *
;******************************************************************
;u8 _write_i2c_block(u8 i2c_adr, u8 reg_adr, u8 *data, u8 size)

    public  __write_i2c_block
__write_i2c_block:
    PUSH    HL                  ; レジスタをセーブ
    PUSH    AX
;
;  このサブルーチンは C（ノーマル・モード）から関数コールされる
;  そのため、この段階でのスタックのデータは以下のようになっている
;
;   sp   : X レジスタ(第 1 引数)  : スレーブ・アドレス
;   sp+1 : A レジスタ             : 未使用（レジスタのセーブだけ）
;   sp+2 : L レジスタ             : 未使用（レジスタのセーブだけ）
;   sp+3 : H レジスタ             : 未使用（レジスタのセーブだけ）
;   sp+4 : PC の下位              : 戻りアドレス下位
;   sp+5 : PC の上位              : 戻りアドレス上位
;   sp+6 : 第 2 引数              : スレーブ内アドレス
;   sp+7 :                        : 未使用
;   sp+8 : 第 3 引数の下位        : 書き込みデータの格納アドレス
;   sp+9 : 第 3 引数の上位
;   sp+10: 第 4 引数              : 書き込みデータ数

    MOVW    AX,SP               ; 2 つ目以降の引数を取り出す準備
    MOVW    HL,AX
    MOV X,#I2C_BUS_NOT_FREE
    CALL    !set_i2c_error      ; ダミーでエラーをセットしておく
    CALL    !free_i2c_bus       ; バスが開放できるか
    BNC $EXITSUB                ; 開放できなければ戻る

    CALL    !I2C_STR            ; スタート･コンディション発行

    MOV X,#I2C_ADDR_NACK
    CALL    !set_i2c_error      ; ダミーでエラーをセットしておく
    MOV A,[HL]                  ; スレーブ・アドレスを取り出す
    AND A,#11111110b            ; 方向フラグを送信に
    MOV X,A
    CALL    !put_i2c2           ; 送信方向でスレーブ・アドレスを送信
    BNC $EXITSUB                ; スレーブから応答なければエラーで戻る

    MOV X,#I2C_REG_ADDR_NACK    ;
    CALL    !set_i2c_error      ; ダミーでエラーをセットしておく
    MOV A,[HL+6]                ; スタックから第 2 引数を取り出す
    MOV X,A
    CALL    !put_i2c2           ; スレーブ内アドレスを送信
    BNC $EXITSUB                ; スレーブから応答なければエラーで戻る
```

```
    MOV A,[HL+10]              ; スタックから第 4 引数を取り出す
    MOV B,A                    ; データ数を B レジスタに設定
    MOV A,[HL +8]              ; スタックから第 3 引数を取り出す。
    MOV X,A                    ; 第 3 引数（書き込みデータ格納アドレス）
    MOV A,[HL+9]               ; を
    MOVW    HL,AX              ; HL レジスタに設定する
    MOV X,#I2C_DATA_NACK       ; ダミーでエラー・フラグにデータへの
    CALL    !set_i2c_error     ; ACK 応答無しを設定
WBLOOP:
    MOV A,[HL]                 ; 書き込みデータをバッファから読み出す
    MOV X,A
    CALL    !put_i2c2          ; データを書き込む
    BNC $EXITSUB               ; エラーが発生したら戻る

    INCW    HL                 ; データ・アドレスを更新
    DBNZ    B,$WBLOOP          ; データ数分繰り返す

    CALL    !I2C_STPR          ; バスを開放
    CALL    !clear_i2c_error   ; エラー・フラグをクリア
EXITSUB:
    MOV B,#0                   ; 戻り値の上位をクリア（不要？）
    POP AX
    POP HL
    RET
```

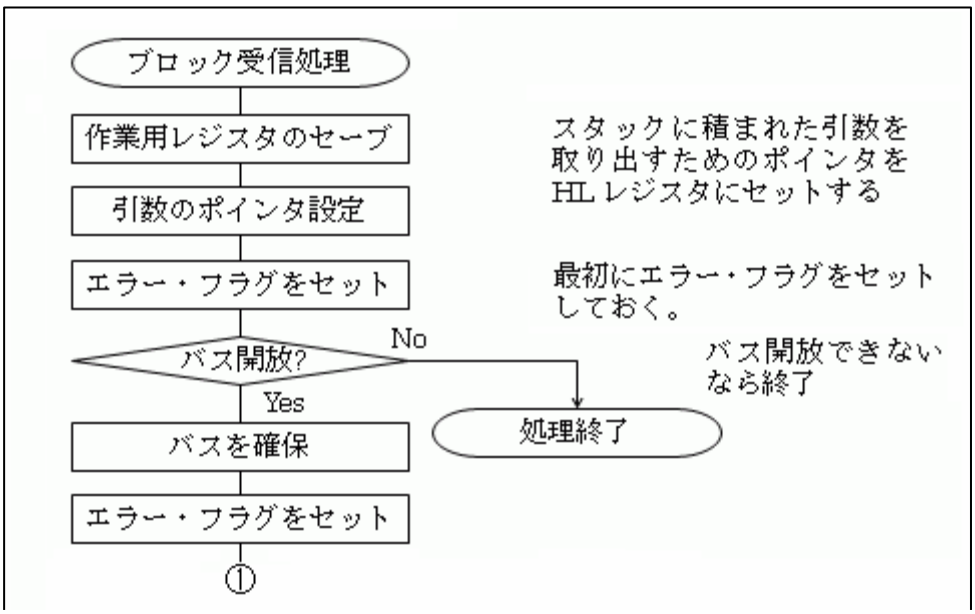
(16) マスタ受信処理(_read_i2c_block)

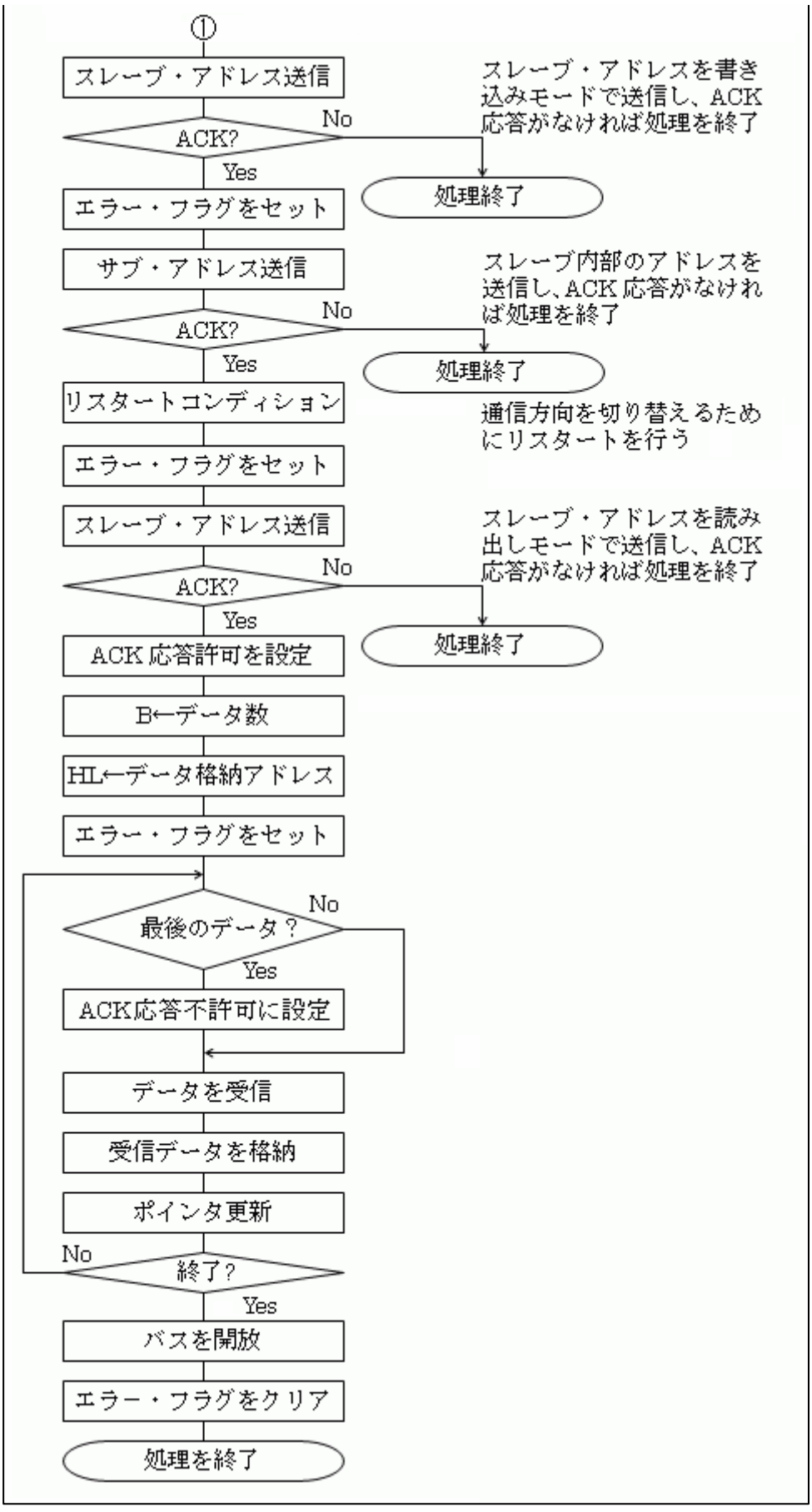
I²C バスを用いて、256 バイトの EEPROM からデータを読み出します。

第 1 の引数で示されたスレーブ・アドレスの EEPROM の、第 2 の引数で示された内部アドレス以降に、第 3 の引数で示されたバッファ・メモリから、第 4 のパラメータで示されたバイト数のデータを読み出します。

スレーブ・アドレスやスレーブ内部のアドレスの送信処理には、アセンブラ記述の_put_i2c2 関数を使用し、エラー発生の場合にはバスを開放して、発生状況に応じたエラーを戻します。エラーが発生しなかった場合は、バスを開放して、エラーなしを戻します。

・プログラム・フロー




①
スレーブ・アドレス送信
スレーブ・アドレスを書き込みモードで送信し、ACK応答がなければ処理を終了
ACK?
No
Yes
処理終了
エラー・フラグをセット
サブ・アドレス送信
スレーブ内部のアドレスを送信し、ACK応答がなければ処理を終了
ACK?
No
Yes
処理終了
リスタートコンディション
通信方向を切り替えるためにリスタートを行う
エラー・フラグをセット
スレーブ・アドレス送信
スレーブ・アドレスを読み出しモードで送信し、ACK応答がなければ処理を終了
ACK?
No
Yes
処理終了
ACK応答許可を設定
B←データ数
HL←データ格納アドレス
エラー・フラグをセット
最後のデータ？
No
Yes
ACK応答不許可に設定
データを受信
受信データを格納
ポインタ更新
終了？
No
Yes
バスを開放
エラー・フラグをクリア
処理を終了

・プログラム例

```
;*****************************************************************
;*                                                               *
;*  I2C デバイスからブロック・データを受信                       *
;*  対象は 256 バイト以下のサブ・アドレスをもつデバイス          *
;*                  ADR(W), SUB_ADR, DATA...                     *
;*****************************************************************
;*  [i] : u8 i2c_adr    ... I2C スレーブのアドレス               *
;*  [i] : u8 reg_adr    ... スレーブ内部のアドレス               *
;*  [i] : u8 *data      ... 書き込みデータのアドレス             *
;*  [i] : u8 size       ... バイト数(0x00 = 256)                 *
;*  [r] : u8            ... error code                           *
;*                                                               *
;*  実際の引数はスタックに 16 ビット単位で積まれて               *
;*  渡されるので、SP の値を HL に持ってきて、HL 相対に           *
;*  より必要な引数を得る。HL からのアクセス時の相対              *
;*  アドレスは書き込みの方を参照                                 *
;*****************************************************************
;u8 _read_i2c_block(u8 i2c_adr, u8 reg_adr, u8 *data, u8 size)

    public  __read_i2c_block
__read_i2c_block:
    PUSH    HL
    PUSH    AX
    MOVW    AX,SP               ; 2 つ目以降の引数を取り出す準備
    MOVW    HL,AX               ; ポインタを HL レジスタに設定
    MOV X,#I2C_BUS_NOT_FREE
    CALL    !set_i2c_error      ; ダミーでエラーをセットしておく
    CALL    !free_i2c_bus       ; バスが開放できるか
    BNC $EXITSUBR               ; 開放できなければ戻る
    CALL    !!I2C_STR           ; スタート･コンディション発行

    MOV X,#I2C_ADDR_NACK
    CALL    !set_i2c_error      ; ダミーでエラーをセットしておく
    MOV A,[HL]                  ; スレーブ・アドレスを取り出す
    AND A,#11111110b            ; 方向フラグを送信に
    MOV X,A
    CALL    !put_i2c2           ; 送信方向でスレーブ・アドレスを送信
    BNC $EXITSUBR               ; スレーブから応答なければエラーで戻る

    MOV X,#I2C_REG_ADDR_NACK
    CALL    !set_i2c_error      ; ダミーでエラーをセットしておく
    MOV A,[HL+6]                ; スタックから第 2 引数を取り出す
    MOV X,A
    CALL    !put_i2c2           ; スレーブ内アドレスを送信
    BNC $EXITSUBR               ; スレーブから応答なければエラーで戻る

    CALL    !!I2C_STR           ; リスタート･コンディション発行

    MOV X,#I2C_RADDR_NACK
    CALL    !set_i2c_error      ; ダミーでエラーをセットしておく
    MOV A,[HL]                  ; スレーブ・アドレスを取り出す
```

```
    OR   A,#00000001b           ; 方向フラグを受信に
    MOV X,A
    CALL    !put_i2c2           ; 受信方向でスレーブ・アドレスを送信
    BNC $EXITSUBR               ; スレーブから応答なければエラーで戻る

    SET1    acke                ; ACK 応答を設定
    MOV A,[HL+10]               ; データ数を
    MOV B,A
    MOV A,[HL+8]                ; スタックから第 3 引数を取り出す。
    MOV X,A                     ; 第 3 引数（書き込みデータ格納アドレス）
    MOV A,[HL+9]                ; を
    MOVW    HL,AX               ; HL レジスタに設定する

BRLOOP:
    MOV A,#1                    ; 最後のデータかをチェックする
    CMP A,B
    BNZ $BRNEXT                 ; 最後のデータ（残り 1 バイト）なら
    CLR1    acke                ; NACK 応答を設定
BRNEXT:
    CALL    !get_i2c            ; データをリード
    MOV A,C                     ;
    MOV [hl],a                  ; 読み込んだデータをバッファに書き込む

    INCW    HL                  ; ポインタ更新
    DBNZ    B,$BRLOOP           ; データ数分繰り返す

    CALL    !I2C_STPR           ; バスを開放
    CALL    !clear_i2c_error    ; エラー・フラグをクリア
EXITSUBR:
    MOV B,#00H                  ; 余分？
    POP AX
    POP HL
    RET
```

[プログラムの使用例]

以上に説明したアセンブラで記述したプログラムの使い方の例として、C言語で記述したプログラムから呼び出して使用する例を示します。

後記(1)、(2)の関数は、それぞれアセンブラでのプログラム例の(15)、(16)で示したプログラムと同じ処理をさせるものです。これらの例では、アセンブラで記述した基本的な制御ルーチンを使用して、I²C バスを用いた 256 バイトの EEPROM の制御を行います。また、(3)はアセンブラでのプログラム例の(15)、(16)や、後記の関数の例(1)、(2)を使用する例となります。

なお、以下の例を使用する際には次に示すようなアセンブラで記述されたプログラムを外部から参照するための宣言や" unsigned char"を"u8"と記述するための宣言などを行っておく必要があります。

```
#pragma SFR
#pragma  EI
#pragma  DI
#pragma  NOP
```

```
typedef unsigned char   u8, uint8,  /* same as "byte"   */
            *u8Ptr, *uint8Ptr;  /* same as "bytePtr"    */
/*  エラー・フラグ操作関係処理関数  */

extern      u8  _set_i2c_error(u8 error);
extern      u8  _get_i2c_error(void);
extern      u8  _clear_i2c_error(void);

/*  エラー・フラグ状態定義          */
enum{
    I2C_NO_ERROR        = 0x00, // エラー無し(0)
    I2C_BUS_NOT_FREE    = 0x10, // バスが開放されていなかった
    I2C_ADDR_NACK       = 0x11, // スレーブがアドレスに反応なし
    I2C_RADDR_NACK      = 0x12, // スレーブが READ アドレスに反応なし
    I2C_REG_ADDR_NACK   = 0x13, // 予期せぬ NACK が返って来た（レジスタ）
    I2C_DATA_NACK       = 0x14  // 予期せぬ NACK が返って来た（データ）
};

/*  ACK 応答制御関係処理関数      */

extern      void    _ACK_EN(void);
extern      void    _ACK_DS(void);

extern      void    _setup_i2c_port(void);
extern      bit _i2c_busy(void);
extern      bit _free_i2c_bus(void);

/*  I2C バス初期化関係処理関数    */

extern      void    _I2C_STR(void);
extern      void    _I2C_STPR(void);

/*  I2C バス・バイト転送基本関数    */

extern      bit _put_i2c(u8 data);
extern      bit _put_i2c2(u8 data);
extern      u8  _get_i2c(void);

/*  I2C バス・ブロック転送処理関数  */

extern      u8 _write_i2c_block(u8 i2c_adr, u8 reg_adr, u8 *data, u8 size);
extern      u8 _read_i2c_block(u8 i2c_adr, u8 reg_adr, u8 *data, u8 size);
```

【ひとこと注意 3】
これらのプログラムは、8MHz の内蔵発振器の出力を CPU のクロックとして動作させるものです。このため、78K0S/KY1+の POC が解除されても、そのままでは 8MHz での動作ができないので、電源電圧が 4V 以上になるのを待つ必要があります。そのためには、初期化のための hdwinit 関数に、次のような処理を記述しておく必要があります。

```
        LVIS = 0x00;          /* select VLI=4V */
        LVIM = 0b10000000;    /* Start VLI detection*/
        PCC = 0x00;           /* CPU clock is fx/4 */
        _clear_i2c_error();   /* I2C flag clear */
        _setup_i2c_port();    /* Initialize I2C port*/
    /*
        wait for 0.17ms(=0.2ms-(32+30)*52/1000)
    */
        for (work1 = 0; work1 < 10; work1++){
            NOP();
        }
        WDTM = 0b01110000;    /* Stop WDT */
        LSRSTOP = 1;          /* Stop Low Speed OSC */
        while ( LVIF ){
            NOP();
        }   /* wait VDD > 4V */

        PPCC = 0;         /* CPU clock is fx=fR */
        IF0 = 0x00;       /* Clear interrupt */
```

【ひとこと 6】

ここでは、WDT や低速内蔵発振器を停止させたり、X1,X2 端子をポートとして I²C バスで使用したりするために、オプション・バイトで指定する必要があります。そのため、次のようなアセンブラ記述の定義ファイルを作成し、リンクしておく必要があります。

```
    @@OPTB  CSEG    AT  0080H
    DB  10011100b
    DB  11111111b
    END
```

(1) マスタ送信処理(u8 write_i2c_block(u8 i2c_adr, u8 reg_adr, u8 *data, u8 size))

I²C バスを用いて、256 バイトの EEPROM へ書き込みを行う関数です。第 1 の引数で示されたスレーブの、第 2 の引数で示された内部アドレス以降に、第 3 のパラメータで示されたバッファ・メモリのデータを、第 4 のパラメータで示されたバイト数だけ書き込みます。バイト単位の送信処理では、アセンブラ記述の_put_i2c2 関数を使用して、エラー発生の場合にはバスを開放し、発生した状況に応じたエラーを戻します。エラーが発生しなかった場合は、バスを開放して、エラーなしを戻します。処理の手順は、次のとおりです。

- 最初に I²C バスが開放状態であることを確認して、バスを開放する。
- 開放できなければ、エラーで戻る。
- スタート･コンディションを発行して、バスの使用を開始する。
- EEPROM のアドレスを指定して、送信方向でアドレスを送信する。
- 書き込みたい EEPROM 内部のアドレスを送信する。
- 指定されたバイト数のデータを送信する。
- 送信が完了して、ストップ･コンディションを発行する(EEPROM はストップ･コンディションを検出し、送信されたデータを内部のセルに書き込む)。

```
u8 write_i2c_block(u8 i2c_adr, u8 reg_adr, u8 *data, u8 size)
{
    i2c_adr &= 0xfe;
    if (!_free_i2c_bus())          // バスを開放させる。
        return  _set_i2c_error(I2C_BUS_NOT_FREE); // バスが開放されない場合はエラー・
                                                  // フラグをセットして戻る

    _I2C_STR();                    // スタート・コンディション発行

    if (!_put_i2c2(i2c_adr))       // I2C アドレス送信
        return  _set_i2c_error(I2C_ADDR_NACK); // 対象デバイスがノー・リアクションなら
                                               // エラー・フラグをセットして戻る

    if (!_put_i2c2(reg_adr))       // レジスタ・アドレス送信
        return  _set_i2c_error(I2C_REG_ADDR_NACK); // サブアドレスで NACK ならエラー・
                                                   //フラグをセットして戻る

    do{
        if (!_put_i2c2(*data++))   // データ送信
            return  _set_i2c_error(I2C_DATA_NACK); // データ送信で NACK 発生
    }while(--size);
    _I2C_STPR();                   // 正常終了でバスを開放
    return  _clear_i2c_error();    // エラー・フラグをクリアして戻る
}
```

(2) マスタ受信処理(u8 read_i2c_block(u8 i2c_adr, u8 reg_adr, u8 *data, u8 size))

I²C バスを使用して、256 バイトの EEPROM からデータを読み出します。

第 1 の引数で示されたスレーブの、第 2 の引数で示された内部アドレス以降から、第 3 のパラメータで示されたバッファ・メモリに、第 4 のパラメータで示されたバイト数のデータを読み出します。

スレーブ・アドレスやスレーブ内部のアドレスの送信処理には、アセンブラ記述の_put_i2c2 関数を使用し、エラーが発生した場合にはバスを開放して、発生した状況に応じたエラーを戻します。エラーが発生しなかった場合は、バスを開放して、エラーなしを戻します。通信の手順は、次のとおりです。

- 最初に I²C バスが開放状態であることを確認して、バスを開放する。
- 開放できなければ、エラーで戻る。
- スタート･コンディションを発行して、バスの使用を開始する。
- EEPROM のアドレスを指定して、送信方向でアドレスを送信する。
- 読み出したい EEPROM 内部のアドレスを送信する。
- リスタートをかける。
- EEPROM のアドレスを指定して、受信方向でアドレスを送信する。
- 指定したアドレスからの、データの読み出しを行う。
- 最終データでなければ、acke をセットして ACK 応答するようにし、アセンブラの_get_i2c を呼び出して処理する。
- 最終データなら、acke をクリアして NACK 応答するようにし、_get_i2c を呼び出して処理する。

```
u8 read_i2c_block(u8 i2c_adr, u8 reg_adr, u8 *data, u8 size)
{
    u8  d;
    i2c_adr &= 0xfe;
    if (!_free_i2c_bus())
        return  _set_i2c_error(I2C_BUS_NOT_FREE);

    _I2C_STR();                     // スタート･コンディション発行

    if (!_put_i2c2(i2c_adr))        // I2C アドレス送信
        return  _set_i2c_error(I2C_ADDR_NACK); // 対象デバイスがノー・リアクションなら、
                                               // エラー・フラグを設定して戻る

    if (!_put_i2c2(reg_adr))        // レジスタアドレス送信
        return  _set_i2c_error(I2C_REG_ADDR_NACK); // サブ・アドレスで NACK ならエラー・
                                                   // フラグを設定して戻る

    _I2C_STR();                     // リスタート

    if (!_put_i2c2(i2c_adr | 0x01)) // I2C アドレス(READ)送信
        return  _set_i2c_error(I2C_RADDR_NACK); // 対象デバイスがノー・リアクションなら
                                                // エラー・フラグを設定して戻る

    _ACK_EN();              // ACK 許可
    do{
        if (size == 1)      // 最後のバイト？
            _ACK_DS();      // マスタ受信最後のバイトなら NACK に設定
        d = _get_i2c();     // データの読み込み
        *data++ = d;        // 読み込みデータをバッファにセット
    }while(--size);

    _I2C_STPR();            // バスを開放

    return  _clear_i2c_error(); // エラー・フラグをクリアして戻る
}
```

(3) EEPROM アクセス例

この使用例では受信用に 32 バイト、送信用に 16 バイトを準備してアセンブラのプログラム例(15)、(16)や前記の(1)**マスタ送信処理**や(2)**マスタ受信処理**を使用します。使用するのに必要な定義は次のとおりです。

```
u8      eep_read_buf[32];   // read buffer
u8      eep_write_buf[16];  // write buffer
u8      result;             // for result flag

#define     ADR_EEP     0xa0    // slave address
//              1010xxxyB
//              ||||||||
//              |||||||+--  R/W#
//              |||||++---  P1/P0 (fixed 00)
//              ||||+-----  A2 (fixed 0)
//              ++++-------  I2C Addr (fix)
```

・アセンブラ記述の関数を使用して、バッファ中の 16 バイトのデータを EEPROM の 00～0F 番地に書き込む。

```
result = _write_i2c_block(ADR_EEP, 0x00, eep_write_buf, 16);
```

・アセンブラ記述の関数を使用して、EEPROM の 00～1F 番地のデータをバッファに読み出す。

```
result = _read_i2c_block(ADR_EEP, 0x00, eep_read_buf, 32);
```

・C言語記述の関数を使用して、バッファ中の 16 バイトのデータを EEPROM の 00～0F 番地に書き込む。

```
result = write_i2c_block(ADR_EEP, 0x00, eep_write_buf, 16);
```

・C言語記述の関数を使用して、EEPROM の 00～1F 番地のデータをバッファに読み出す。

```
result = read_i2c_block(ADR_EEP, 0x00, eep_read_buf, 32);
```

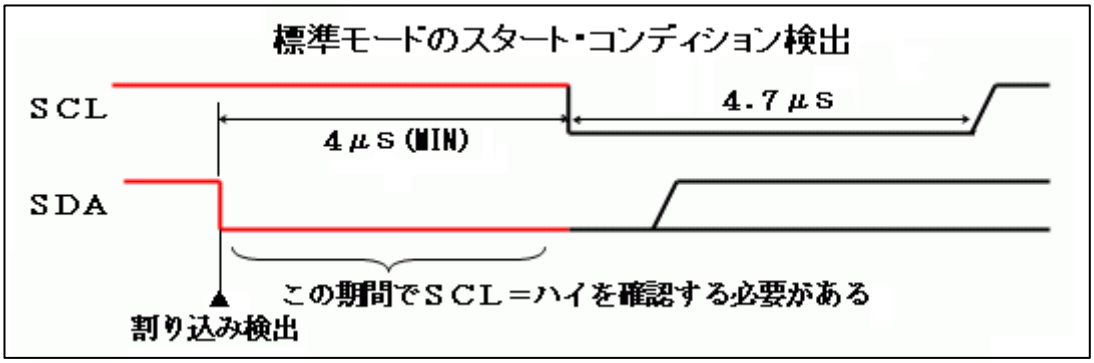
ソフトウエアでの I²C バスのスレーブ機能の検討 [78K0S/Kx1+]

ソフトウエアで I²C バスのスレーブ機能を実現する場合のハードルとして、次の点が考えられます。

①スタート・コンディション／ストップ・コンディションの検出
②SCL に対するリード／ライトのタイミング遅延
③ビット処理時間
④バイトの切れ目（9 クロック目のあと）での処理
⑤複数スレーブで非選択時のバスの無視

①については、専用のハードウエアがないので、常に監視することはできません。転送起動時にスタート･コンディションを検出するのと、処理途中のデータの切れ目（転送の 9 クロック目が完了したタイミング）で確認する程度になります。

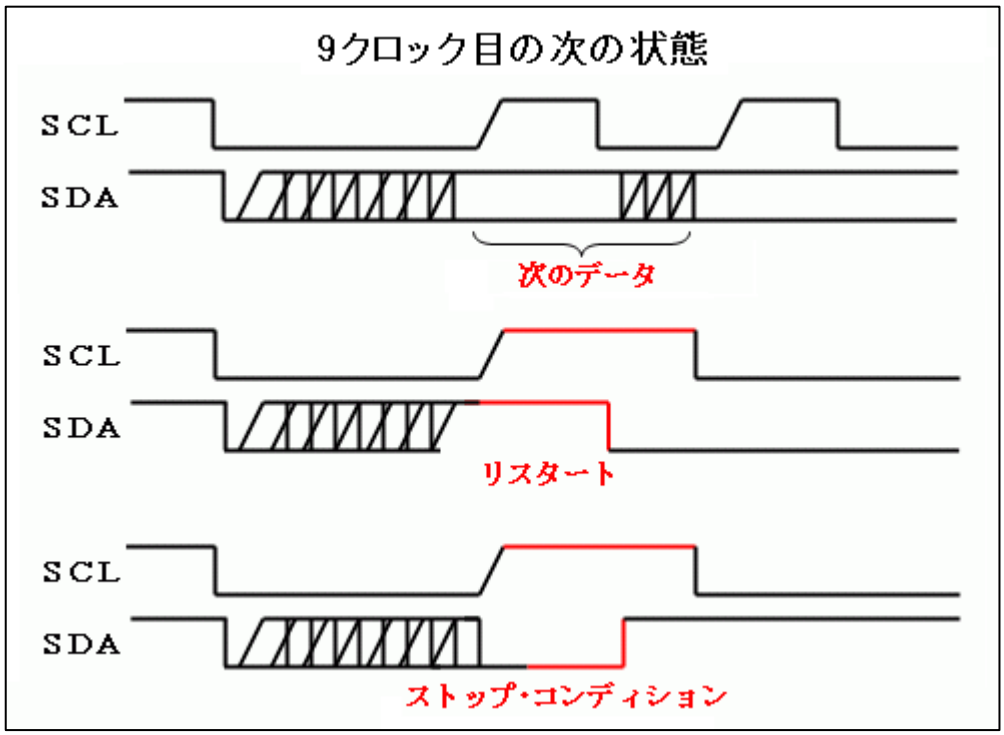
転送開始を検出する場合も、ベクタ割り込み応答になってしまうので、割り込み応答時間＋その後の確認までの時間を合わせて、最低 3μs（8MHz 動作時）程度必要になります。このことは、他の割り込みを処理していて 1μs 以上割り込みが保留になると、間に合わなくなることを示します。転送中のデータの切れ目（通常のストップ･コンディションやリスタートのタイミング）についても、ほぼ同じ時間が必要と考えると、**ファースト・モードの 0.6μs の規格には対応は不可能**で、**標準モードだけ**の対応となります。ファースト・モードに対応できる可能性があるのは、V850ES 以上の性能の製品と考えられます。



②については、SCL の立ち上がりからデータを取り込むまで時間の制限が厳しいと考えられます。同じポートの異なるビットで同時に読み出せば、SDA と SCL の関係は確保できますが、処理時間のネックが予想されます（1 回確認するループで「ポートのリード＋ビット判定での分岐」に 12 クロック必要で、2 回目で確認できたとすると 24 クロック(3μs)が必要）。これについては、通信速度を落として SCL の幅を確保する必要があります。さらに、スタート･コンディションだけ割り込みで検出し、その後のデータのやり取りはシーケンシャルに処理して、すべての処理が完了したら割り込みから復帰するような処置が必要です。これらの対策で、**標準モードには何とか対応可能**と考えられます。

③のビット･データの処理については、②の条件を満足すれば処理できる可能性があります。ただし、アドレスに対する ACK 応答を**ウエイト**で処理する可能性があり、スレーブでウエイト機能をサポートしていない 78K0R/Kx3 のシリアル･アレイ･ユニットの簡易 I²C 機能では、単純に接続できない可能性があります。この場合は、**ウエイトをかけなくて済む程度の転送速度（標準モードには何とか対応可能）**にしたり、必要な処理をいくつかに分割して、SCL の変化確認の間に分散処理する必要があります。

④については、次に示す**次のデータ**、**リスタート（スタート･コンディション）**、**ストップ･コンディション**の 3 つの状態が考えられます。これらの判定を行う必要があるので、一番のボトルネックになると考えられます。
ソフトウエア処理による信号確認で、確認間隔が長くて（10～12 クロック）リスタートと判断した場合は、その後にストップ･コンディションが来ることも考えられます。



これらを考慮して、その前の**ウエイト状態**（78K0R/Kx3 の簡易 I²C 機能でも通信の切れ目なので、次の通信を起動しないかぎりウエイト状態になるはず）で、どこまで前処理できるかが鍵になります。したがって、スレーブでのウエイト機能をサポートしていない 78K0R/Kx3 の簡易 I²C 機能でも、単純に**データとデータの間隔がスレーブからウエイトをかけなくて済む程度**にする必要があります。通常、マスタ側では、9 クロック目の SCL の立ち下がりで割り込みが発生し、そこから ACK を確認したり、受信データの処理をしたりするので、ここで 1 クロック分は間が空くと考えられます。

⑤は全体の処理のボトルネックになる可能性の高い部分です。SCL ごとに割り込み処理をすると、他の処理ができなくなる可能性が十分にあります。そのため、SCL での割り込みはせず、SDA の立ち下がりエッジ検出をスタート・コンディション検出とします。この場合、他のスレーブとの通信中のデータの変化で割り込みが入るので、通信状況をうまく管理できなくなります。そのため、**シングル・マスタ、シングル・スレーブでの動作に制限**する必要があります。

以上の結果をまとめると、次のようになります。

- **・標準モード（通信クロックは 100kHz 以下）**
- **・シングル・マスタ（想定外のタイミングではストップ・コンディションが出ない）**
- **・シングル・スレーブ（通信中以外で不要な信号変化がない）**
- **・通信途中（8 ビット目まで）にスタート・コンディションやストップ・コンディションが発行されない（通信前か 9 クロック目以降で発行可能）**
- **・他の割り込みは使用しない（割り込み応答の余裕が 1μs しかないため）**

これらの条件をつければ、実現可能と考えられます。これらを加味した具体的な検討については、「ソフトウエアでの I²C バスのスレーブ機能の実現(1) [78K0S/Kx1+]」をご参照ください。

ソフトウエアでの I²C バスのスレーブ機能の実現(1) [78K0S/Kx1+]

前述の「ソフトウエアでの I²C バスのスレーブ機能の検討」の結果でまとめた条件で、I²C バスのスレーブ機能をソフトウエアによるポート制御で実現してみます。

(1) 処理概要

プログラム例を示しながら説明します。ここで対象にするプログラムは、スタート・コンディションを割り込みで検出（SDA の立ち下がりで割り込みが発生するので、そのあとソフトで判定）して処理を開始します。その後、その割り込みの中で処理を行い、スレーブ送信ではマスタからの NACK 応答で、またスレーブ受信ではデータ受信後のストップ・コンディション検出で、通信を完了して割り込み処理から抜けます。

データ転送の方向（送信／受信）の判断は、I²C バスの規格に従って、アドレスのビット 0 で行います。I²C バスについては、「I²C バスについての概要」をご参照ください。データの送受信は、内部に用意された送信／受信用バッファに対して行います。このプログラムで使用する変数は、次のようになります。このプログラムを使用する場合は、これらを事前に設定しておく必要があります。

RXPOINT：受信データを格納するアドレスを保持しています。データを受信することで、自動的に更新されます。

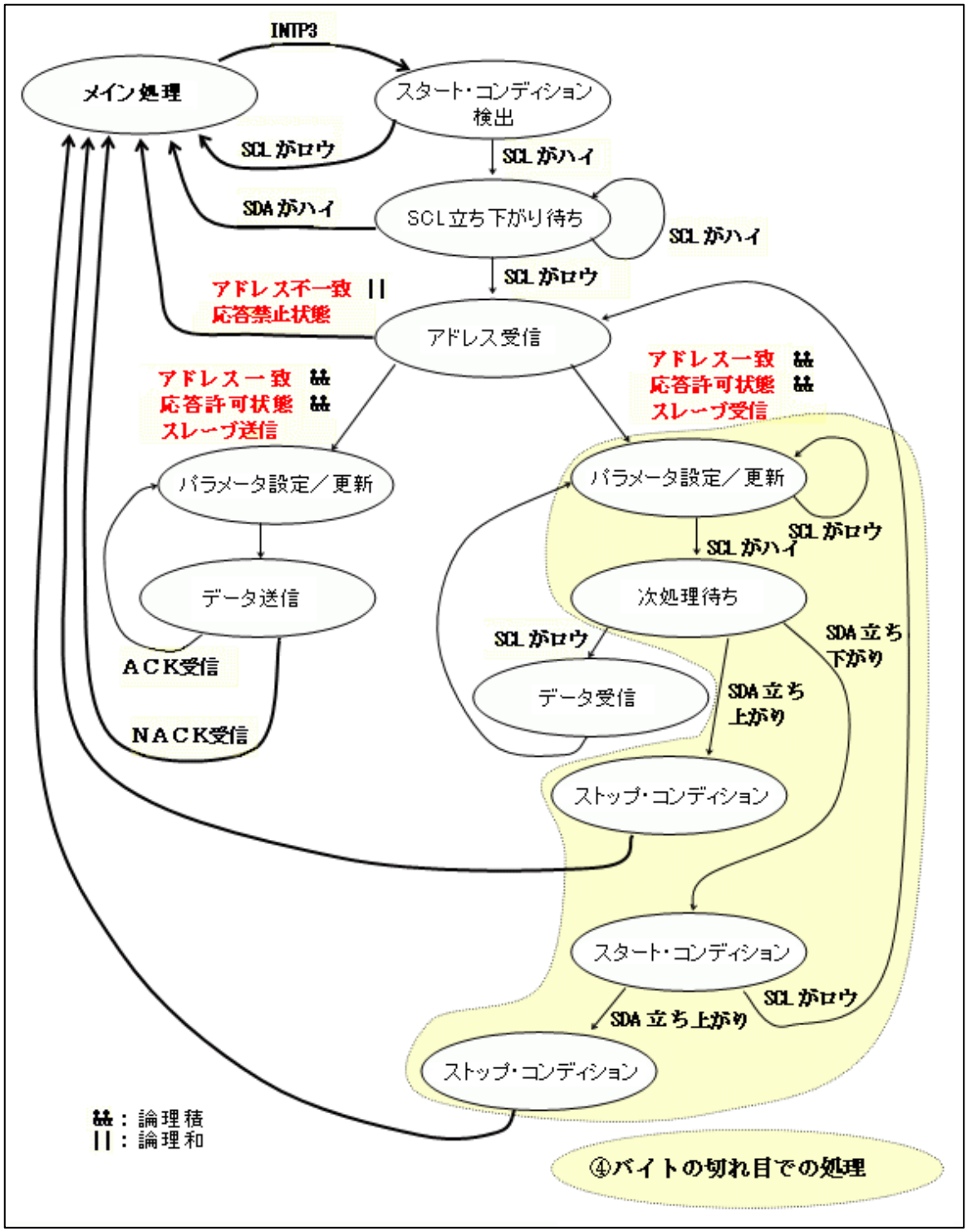
TXPOINT：送信するデータの格納アドレスを保持しています。データを送信することで、自動的に更新されます。

SAV0　：スレーブとしてのアドレスを保持しています。マスタからのアドレス（上位 7 ビット）がこの値と一致すると、通信を行います。ビット 0 は必ず 0 にしておく必要があります。アドレスが一致しないときは、IICC0 の ACKE をクリアすれば、ACK 応答をしなくなります。

IICC0　：ビット 2 が ACK 応答するかどうかのフラグ（**ACKE**）です。このビットが 1 のとき、マスタからのアドレスが SAV0 の値と一致したら ACK 応答します。マスタからデータを受信したとき、処理中に次の通信をしないように、フラグ（**ACKE**）をクリアします。そのため、受信データの処理が完了して、次のデータ転送が可能になったら、フラグ（**ACKE**）をセットする必要があります。

(2) 全体の処理

ここで考えている、通信処理全体の概略状態遷移図を次に示します。



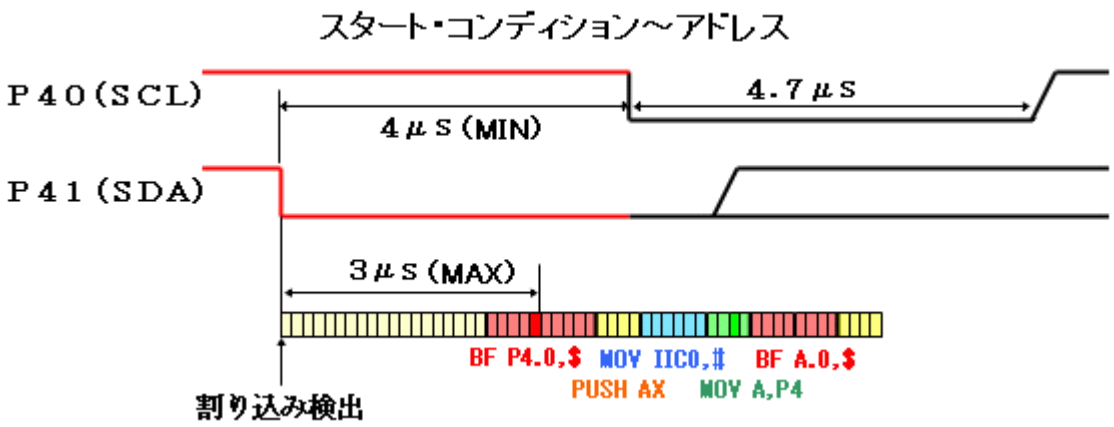
処理の開始は、SDA 信号の立ち下がり(**INTP3**)とします。SCL 信号のとの関係をチェックして、スレーブ・アドレス受信に移ります。受信したスレーブ・アドレスが一致して、応答許可状態のときに ACK 応答をします。内部処理中で通信したくないときは、応答禁止状態にしておき、アドレスが一致しても ACK 応答しないようにします。これにより、マスタに処理中であることを知らせます。スレーブ受信の場合は、受信準備をして、「バイト切れ目での処理」になります。ここで、ストップ・コンディションを検出すると、通信処理を完了します。スレーブ送信では、マスタから ACK 応答があれば次のデータを送信し、ACK 応答がなければ(NACK 応答)通信を完了します。

以下に、詳細な処理について説明します。

(3) 通信の起動(スタート･コンディション検出)

通信を開始するタイミング(スタート･コンディション)は、SDA 信号の立ち下がりエッジを検出したベクタ割り込みで検出するものとします。次のプログラム例では、**SCL 信号がハイでなければ処理しない**ようにしていますが、十分にスタート･コンディション検出が可能です。

```
__INTP3:
                BF      P4.0,$NOTSTART          ;SCL がハイか確認
                PUSH    AX                      ;AX レジスタをセーブ
                MOV     IICO,#00000000B         ;データ初期値を設定
LOOPO:
                MOV     A,P4                    ;SDA と SCL 読み込み
                BF      A.0,$ADDRESSO           ;SCL の立ち下がりで分岐
                BF      A.1,$LOOPO              ;SDA がロウなら継続
NOTACK:
                POP     AX                      ;AX レジスタを復帰
NOTSTART:
                RETI                            ;ストップ・コンディション検出
```



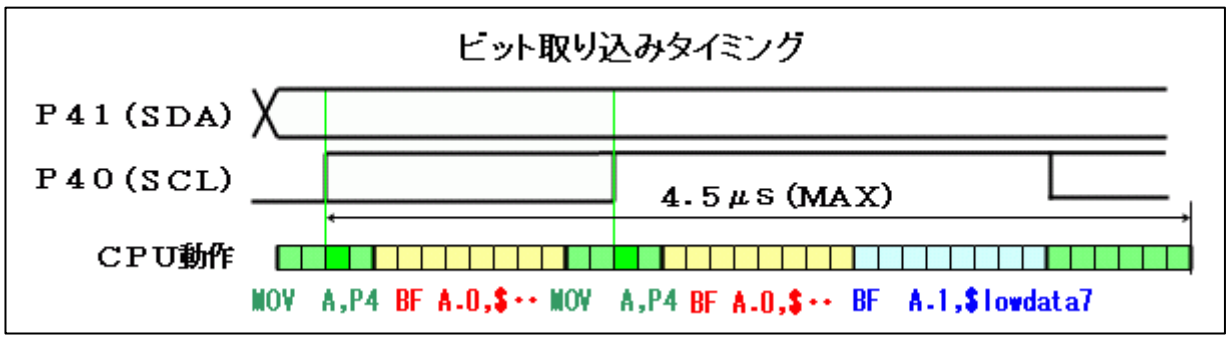
(4) アドレスの受信

スタート･コンディションを検出したら、**SCL 信号がいったんロウになったことを確認**してから、アドレス受信のために **SCL 信号の立ち上がりを待ちます**。このとき、MOV 命令で SCL 信号と SDA 信号を一緒に取り込めば、立ち上がりの確認間隔は 12 クロックで、BF 命令に比べて長くなります。しかし、SDA 信号も同時に取り込めるので、タイミングのずれを気にする必要がなくなります。

このときの処理プログラムは、次の例のようになります。SCL 信号がハイのときに SDA 信号を有効として、そのデータがハイの場合は、変数 **IICO** の該当するビットを 1 にします。

ここでの最悪のケースとして、読み込みの直後に SCL 信号が立ち上がったとしても、36 クロック(4.5μs)で、ビットの受信処理は完了できます。その後の SCL 信号の立ち下がり確認を含めて 46 クロック(約 6μs)となり、ビットの周期 10μs に対して十分なマージンで処理可能です。

```
ADDR7:          MOV     A,P4                    ;SDA,SCL 読み込み
                BF      A.0,$ADDR7              ;SCL の立ち上がり待ち
                BF      A.1,$lowdata7           ;SDA のチェック
                SET1    IICO.7                  ;SDA が 1 ならビットをセット
lowdata7:
                BT      P4.0,$$                 ;SCL の立ち下がり待ち
```



次のビット以降についても、同様に SCL 信号の立ち上がりを待って処理を行います。前述の処理は、ビット位置を変えて行う必要があるので、同じような処理を繰り返し記述することになります。ここでは、記述を簡単にするためにマクロ機能を利用し、次のように 1 ビットの受信処理をするマクロ sin1 を定義します。

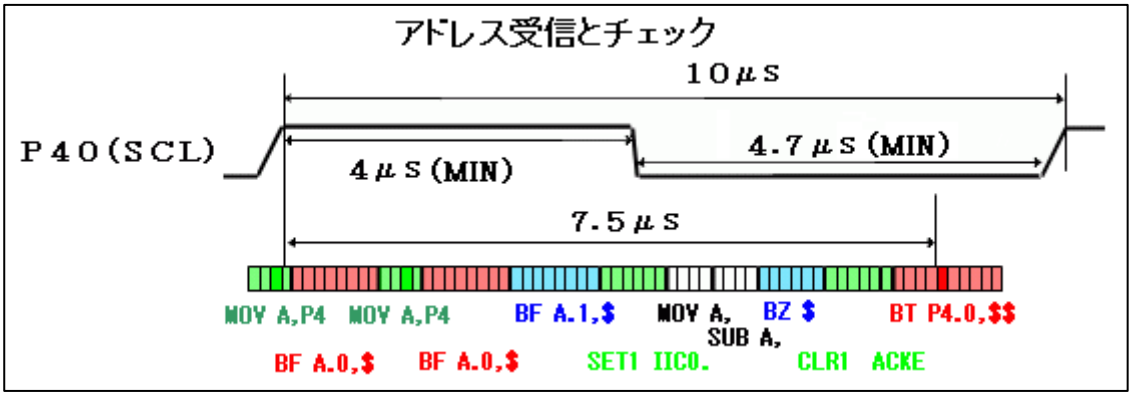
```
sin1    macro   bitno
        LOCAL   DATAis0,LOOP
LOOP:
                MOV     A,P4                    ;SDA,SCL 読み込み
                BF      A.0,$LOOP               ;SCL の立ち上がり待ち
                BF      A.1,$DATAis0            ;SDA のチェック
                SET1    IICO.bitno              ;SDA が 1 ならビットをセット
DATAis0:
                BT      P4.0,$$                 ;SCL の立ち下がり待ち
        endm
```

これで、マクロ名とビット番号(マクロの引数)を組み合わせて、次のようにアドレスの受信処理を記述できます。

```
ADDRESS0:
                PUSH    HL
ADDRESS:
                sin1    7                       ;ビット 7 受信
                sin1    6                       ;ビット 6 受信
                sin1    5                       ;ビット 5 受信
                sin1    4                       ;ビット 4 受信
                sin1    3                       ;ビット 3 受信
                sin1    2                       ;ビット 2 受信
```

これで、ビット 7～ビット 2 の受信処理を行います。ビット 1(アドレスの最終ビット)については、アドレス比較が必要なので、できるだけむだな待ちループが発生しないよう、最後の SCL 信号の立ち下がり待ちの前にアドレス比較を行います。以下の処理を実行しても、ウエイトをかけずに SCL 信号のロウ期間に処理を完了して、最後の転送方向受信処理に間に合います。

```
LOOP1:
                MOV     A,P4                    ;SDA,SCL 読み込み
                BF      A.0,$LOOP1              ;SCL の立ち上がり待ち
                BF      A.1,$bit1is0            ;SDA のチェック
                SET1    IICO.1                  ;SDA が 1 ならビット 1 をセット
bit1is0:
                MOV     A,SVA0                  ;アドレスを確認
                SUB     A,IICO                  ;結果をゼロ・フラグに
                BZ      $RESACK                 ;アドレスが一致しないと
                CLR1    ACKE                    ;ACK 応答禁止に設定
RESACK:
                BT      P4.0,$$                 ;SCL 立ち下がり待ち
```

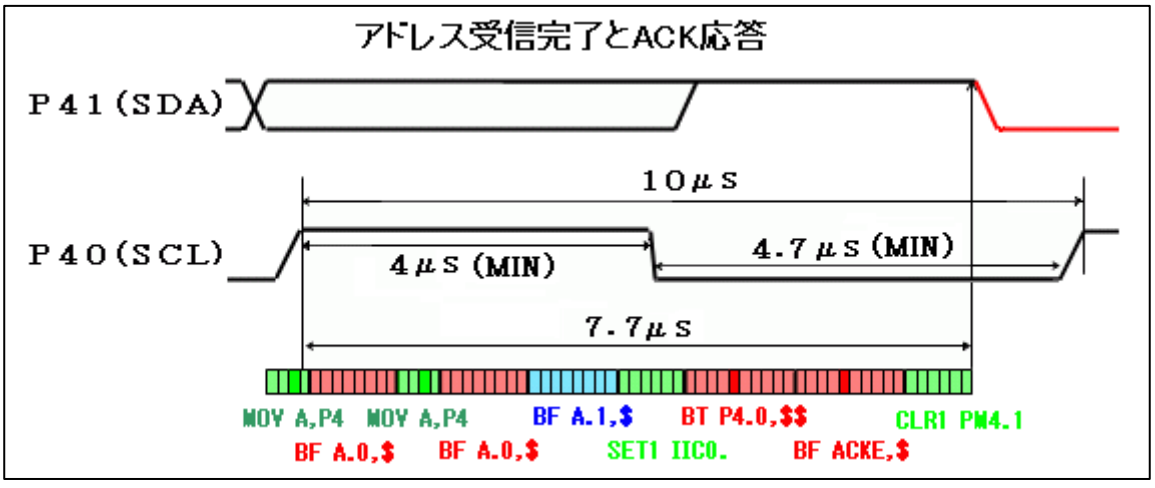


(5) アドレスの受信完了から ACK 応答

アドレスの 8 ビット目（転送方向）を受信したら、アドレスが一致して、ACK 応答可能なときに、SCL 信号の 9 クロック目で ACK 応答を行います。アドレスの一致は最初の 7 ビットでチェックが完了しているので、ここではその結果の **ACKE** フラグを参照して、応答するかどうかを判定します。アドレスが一致していても、**ACKE** フラグがクリアされている（別の処理中で応答できない）場合は、ACK 応答しません。したがって、スレーブから ACK 応答がなければ、マスタはある程度時間をおいて、再度スタート・コンディションから開始する必要があります。ACK 応答しないとき、スレーブの処理はこれで完了です。

転送方向を受信した SCL 信号の立ち上がりから約 7μs で、ACK 応答を戻すことができます。これは、次の SCL 信号の立ち上がりに対して、十分なマージンを確保できます。

```
sin1    0               ;ビット 0 受信
BF      ACKE,$NOTACK0   ;ACK 応答しないなら抜ける
CLR1    PM4.1           ;ACK 応答
```



(6) ACK 応答完了

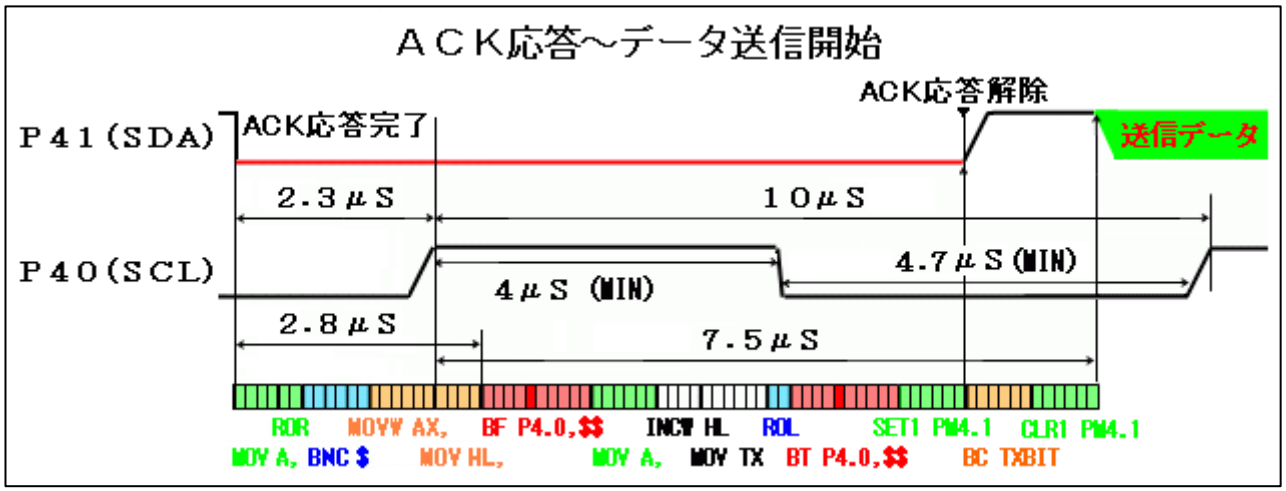
ACK 応答が完了したら、ACK の転送完了（SCL 信号の立ち上がりと立ち下がり）を待ち、ACK 応答を切ります。これでアドレス受信処理が完了したので、あとは転送方向に応じた処理となります。SCL 信号の変化待ちでは、処理に余裕があるので、事前に転送方向で処理を分けておきます。

```
MOV     A,IICO          ;アドレスを A レジスタに
ROR     A,1             ;CY=0:受信、1:送信
BNC     $RXDATA         ;受信なら分岐
```

(7) スレーブ送信でのアドレス 9 クロック目以降の処理(送信準備)

スレーブ送信では、事前にデータを準備する必要があります。その処理時間が必要なので、前サイクルの ACK 応答後に、すぐ処理を開始しておきます。SCL 信号の立ち上がり周期 10μs に対し、ACK 応答は 7.7μs で完了しているので、2.3μs の余裕があります。そこで、ACK 応答完了処理の MOV A,IIC0 命令から下記 MOVW HL,AX 命令までの実行後、SCL 信号が立ち上がったかどうかを確認します。これらの処理に 22 クロック(2.8μs)必要なので、SCL 信号の立ち上がりから 0.5μs で、立ち上がり確認の BF P4.0,$$命令を実行することになります。これで、SCL 信号の立ち上がりは 1 回だけの確認で次に進めます。この部分の処理は、7.5μs で完了します。

```
TXDATA:
                MOVW    AX,TXPOINT          ;データ・ポインタを読み出し
                MOVW    HL,AX               ;HL にポインタをセット
                BF      P4.0,$$             ;SCL 立ち上がり確認
TXLOOP:
                MOV     A,[HL]              ;送信データ読み出し
                INCW    HL                  ;ポインタ更新
                MOV     TXCOUNT,#8          ;ビット数をセット
TXBLOOP:
                ROL     A,1                 ;MSB をキャリーへ
                BT      P4.0,$$             ;SCL 立ち下がり待ち
                SET1    PM4.1               ;ACK 応答解除
                BC      TXBIT               ;データが 1 なら分岐
                CLR1    PM4.1               ;SDA をロウに
TXBIT:
```

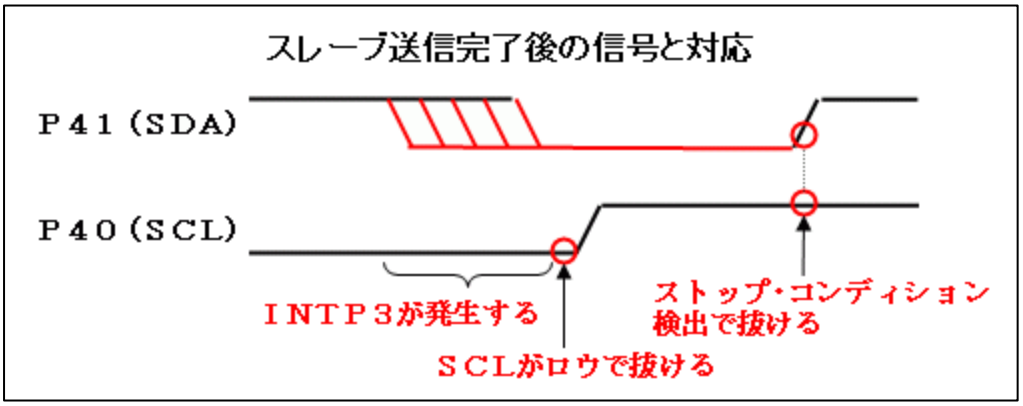


(8) データ送信処理～ACK 確認

スレーブ送信処理では、SCL 信号の変化を待つだけなので、ループでも十分に処理可能です。そこで、ループ・カウンタ(変数 TXCOUNT)を用いて制御します。8 ビットの送信が完了すると、SDA 信号を開放して ACK 応答を待ちます。マスタから ACK 応答があった場合、次のデータに進みます。ACK 応答がなかった場合、更新されたデータ・ポインタを保存して、処理を終了します。

```
TXBIT:
                BF      P4.0,$$             ;SCL 立ち上がり待ち
                DBNZ    TXCOUNT,$TXBLOOP    ;8 ビット分繰り返し
                BT      P4.0,$$             ;SCL 立ち下がり待ち
                SET1    PM4.1               ;SDA 開放
NOTACK0:
                BF      P4.0,$$             ;SCL 立ち上がり待ち
                BF      P4.1,$TXLOOP        ;ACK なら次データへ
                MOVW    AX,HL               ;ポインタを AX へ
                MOVW    TXPOINT,AX          ;ポインタを保存する
                SET1    STATUSFLAG.6        ;送信完了フラグをセット
                BT      P4.0,$$             ;SCL 立ち下がり待ち
                POP     HL                  ;レジスタを復帰
                POP     AX                  ;AX レジスタ復帰
                CLR1    PIF3                ;割り込み要求をクリア
                RETI                        ;処理完了
```

スレーブ送信処理では、マスタからの NACK 応答を確認した段階で処理を完了しています。その後、マスタはストップ･コンディションかスタート･コンディションを発行することになります。ここで抜けてしまうと、ストップ･コンディションを発行するときには、次のような信号の動きとなり、SDA の立ち下がりだけを見ているので、スタート･コンディションを誤検出します。しかし、INTP3 割り込み処理の中で、SCL がロウなら何もしないで抜けるようになっていることと、SCL がハイなら SDA の立ち上がり(ストップ･コンディション)で抜けるので、問題はありません。



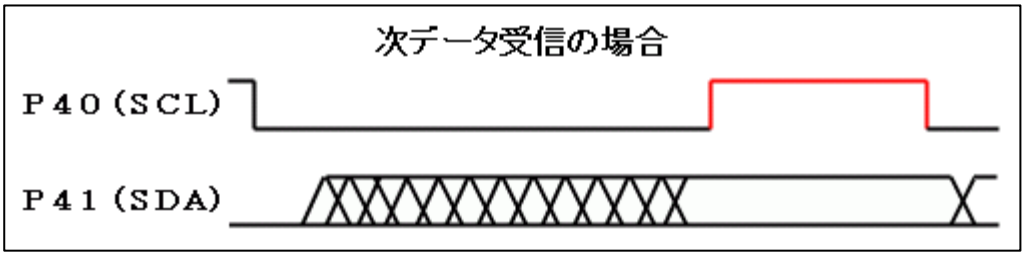
(9) スレーブ受信での 9 クロック目以降の処理

```
RXDATA:
                BF      P4.0,$$             ;SCL 立ち上がり待ち
                MOVW    AX,RXPOINT          ;データ・ポインタを読み出し
                MOVW    HL,AX               ;HL にポインタをセット
                BT      P4.0,$$             ;SCL 立ち下がり待ち
                SET1    PM4.1               ;ACK 応答解除
                MOV     IICO,#00000000B     ;データ初期値を設定
```

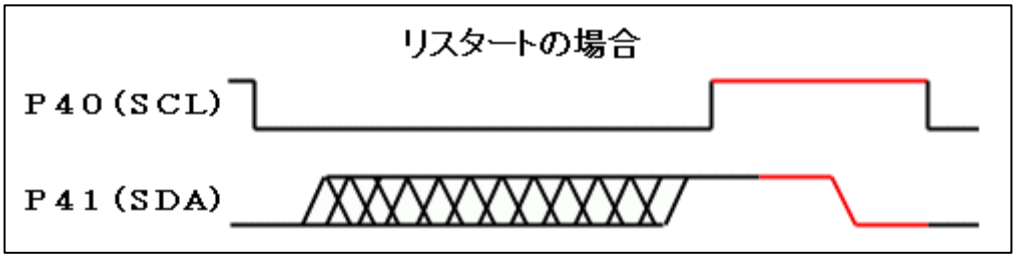
9 クロック目の SCL 信号が立ち下がったら、ACK 応答を解除して SDA 信号を解放します。次に、データの受信処理に移り、以降は繰り返し処理となります。

ここでは、次の 3 つの場合があり、タイミング的にもっとも厳しくなります。
①次のデータを受信する
②スタート･コンディション（リスタート）を受け付ける
③ストップ・コンディションを受け付ける

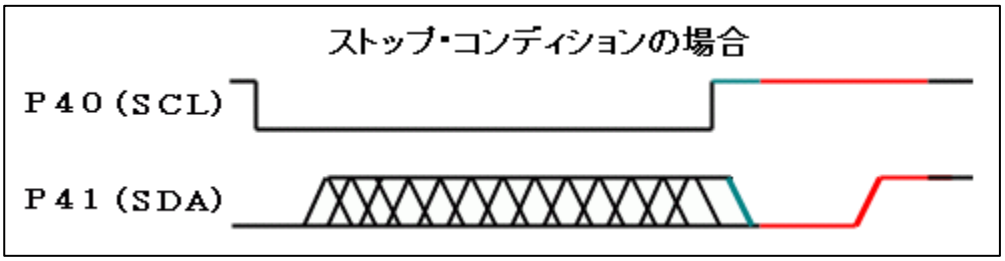
①では、SCL 信号がロウの状態で SDA 信号が変化して、SCL 信号が立ち上がります。その後、SDA 信号を保持したまま SCL 信号が立ち下がります。このとき、はじめてデータだと分かります。この場合、SCL 信号のハイ・レベル幅が 4μs 以上で、SDA 信号は 250ns のセット・アップ時間と 0ns のホールド時間で変化する可能性があります。これらの時間は、判別できる時間間隔よりはるかに短いので、**単独か、SDA 信号立ち下がりと同時に、SCL 信号の立ち下がりを検出できたらデータ受信**と判断することになります。



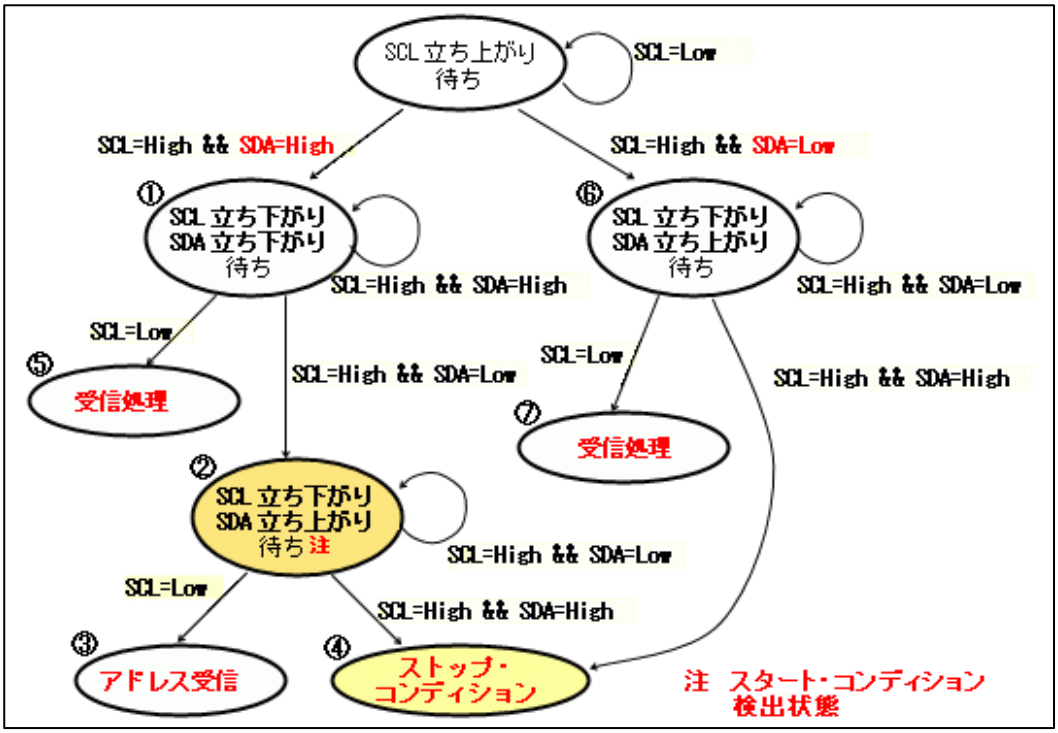
②では、SCL 信号がロウの状態で SDA 信号が立ち上がり、SCL 信号がハイの状態で SDA 信号が立ち下がります。SDA 信号の立ち下がりでスタート･コンディションであることは分かりますが、SCL 信号が立ち下がるまで次に移れません。



③では、SDA 信号をどのタイミングで立ち下げるかで、さらに 2 つの場合が考えられます。SCL 信号がロウの状態で、SDA 信号が立ち下がれば、単純にストップ･コンディションの発行だけとなります。**SCL 信号の立ち上げ以降に SDA 信号が立ち下がった場合**は、そこがスタート・コンディションと同じになります。その後、**ストップ･コンディションが発行**されることになります。つまり、スタート･コンディションを検出した場合は、SCL 信号が立ち下がるか、SDA 信号が立ち上がることを待つ必要があります。



次に、この部分の状態遷移図を示します。

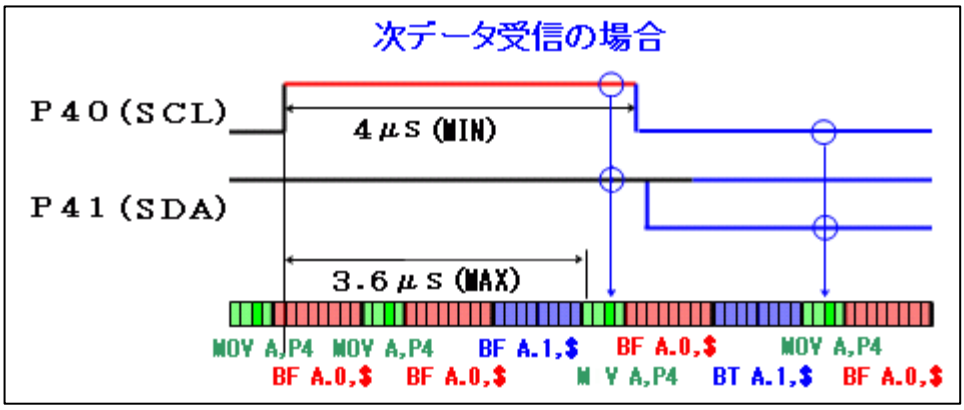


この処理のプログラム例を示します。

①SDA 信号がハイであったときの処理

まず、SCL 信号が立ち上がったときに、SDA 信号がハイの場合の処理を行います。**青がデータ受信の場合**で、**赤がスタート・コンディションの場合**です。SCL 信号の立ち上がりと SDA 信号の組み合わせでの場合分け（下記プログラムの LOOP21 以降の 3 命令）は、SCL 信号の立ち上がりから 30 クロック（3.7μs）以内に処理できるので、SCL 信号の周期が短い(4μs)データ転送の場合でも、タイミングに問題はありません。STCLOOP 以降の 3 命令で、データ受信開始かスタート・コンディション（リスタート）を待っています。データ受信では、SCL 信号の立ち下がりから 30 クロック（3.7μs）以内に検出できるので、SCL 信号がロウの期間（4.7μs）に処理可能です（以降の処理は⑤）。

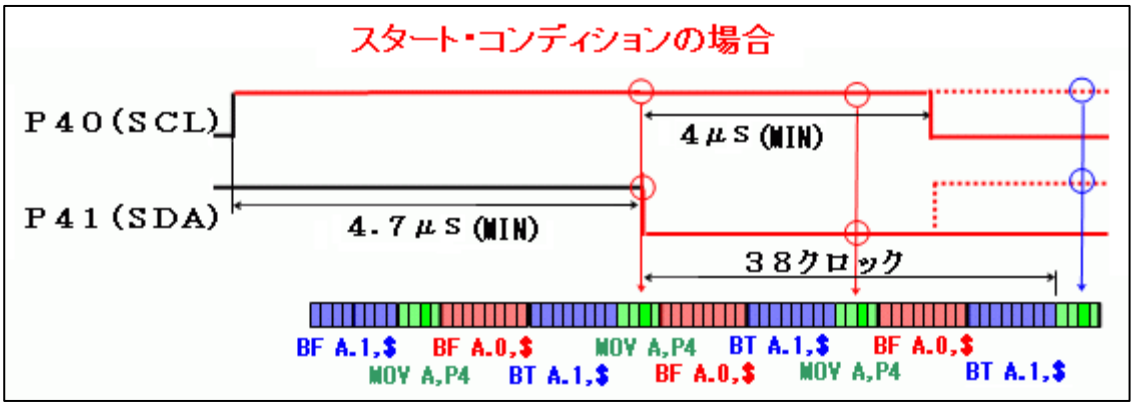
```
;
;       アドレス受信、データ受信したあとの処理の判断
;
LOOP21:
                MOV     A,P4                    ;SCL と SDA を読み込む
                BF      A.0,$LOOP21             ;SCL 立ち上がり待ち
                BF      A.1,$SDALOW             ;SDA がロウなら分岐
;
;       スタート･コンディションかデータ待ち
;   SCL が立ち下がれば通常データ受信（⑤）
;   SDA が立ち下がればスタート･コンディション（②）
;   ここは、SCL を確認してから SDA を確認する必要がある
;
STCLOOP:
                MOV     A,P4                    ;SCL と SDA を読み込む
                BF      A.0,$DATAREAD           ;SCL が立ち下がれば受信へ
                BT      A.1,$STCLOOP            ;変化なければループ
```



②リスタート

リスタートの場合、信号の読み取りが 20 クロックのループになり、1 回は確実に確認できますが、確認処理完了に最大 38 クロックかかります。このとき、次データのために SCL 信号が立ち下がっている可能性があります。その後、4 クロックで信号を確認します（次図の○）が、このときの状態で、次が何かを判断する必要があります。

SCL 信号が立ち下がれば（実線）、リスタート後のアドレス受信(③)の開始です。SCL 信号がハイのまま SDA 信号が立ち上がれば（破線）、ストップ･コンディション(④)です。ここでは、どちらかの信号が変化するまで待ちます。




```
;
;       スタート・コンディション検出
;   検出後、SCL の立ち下がり（通信開始）か SDA の立ち上がりを待つ
;
                MOV     IICO,#0                 ;受信ワークをクリア
STCLOOP2:
                MOV     A,P4                    ;SCL と SDA を読み込む
                BF      A.0,$ADDRESS2           ;SCL の立ち下がりでアドレス受信
                BF      A.1,$STCLOOP2           ;ストップ･コンディション以外
```

③アドレス受信

アドレス受信時の分岐先の ADDRESS2 は、本来 ADDRESS ですが、ここからの条件分岐では直接飛べず、また、いくつかの処理が必要なためのクッションです。RXDATA の直前に配置しておく必要があります。

```
;
;       リスタート時にアドレス受信へ戻る処理
;
ADDRESS2:
                MOVW    AX,HL
                MOVW    RXPOINT,AX              ;データ・ポインタを保存
                BR      ADDRESS
```

④ストップ・コンディション

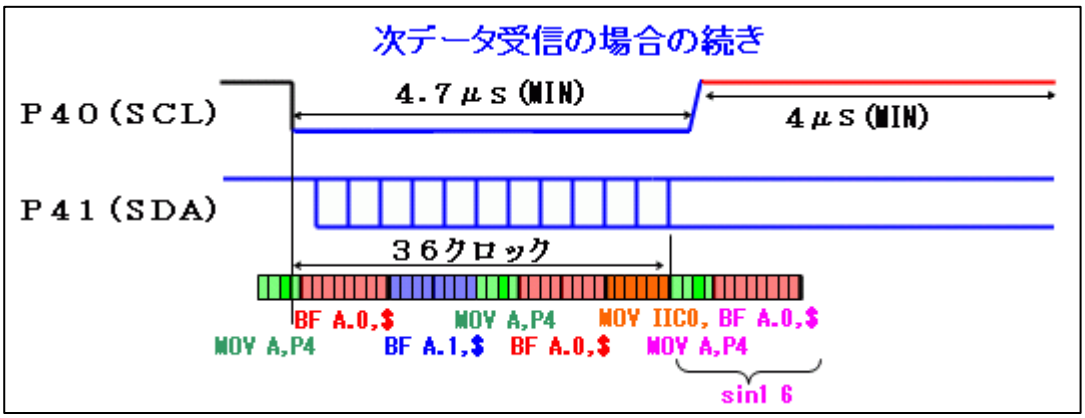
受信を完了して、ストップ･コンディションを検出したら、処理を完了します。ここでは、その後の受信データ処理を考慮して、ACKE をクリアして、以降はマスタが通信を開始し、そのアドレスが一致しても ACK 応答しないようにしています。受信データの処理が完了したら、ACKE をセットしておく必要があります（変数 IICC0 に 4 を設定）。

```
;
;       ストップ・コンディション検出
;   通信完了フラグをセット、ACK 応答禁止にして処理を完了する
;
STPCND:
                SET1    STATUSFLAG.7            ;通信完了フラグをセット
                CLR1    ACKE                    ;ACK 応答禁止に設定
                MOVW    AX,HL
                MOVW    RXPOINT,AX              ;データ・ポインタを保存
                POP     HL                      ;HL レジスタ復帰
                POP     AX                      ;AX レジスタ復帰
                CLR1    PIF3                    ;割り込み要求をクリア
                RETI                            ;処理完了
```

⑤データ受信処理（SDA＝H のとき）

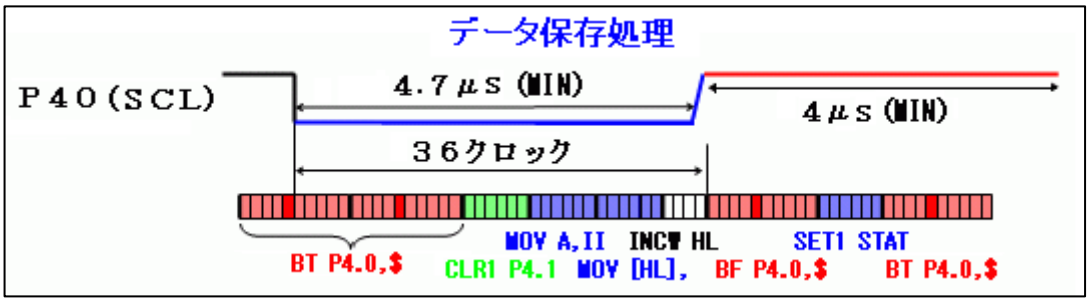
次データ受信の処理です。最初のビットは受信完了済みなので、残り7ビットの受信でACK 応答し、受信データをバッファに書き込みます。その後、**(9)スレーブ受信での 9 クロック目以降の処理**の最初に戻ります。

```
;
;       データ受信処理
;   最上位ビットは既に受信済みのため、残り 7 ビット分を受信する
;   受信が完了して、ACK 応答後 9 クロック目でウエイトして処理を抜ける
;
DATAREAD:
                MOV     IICO,#10000000B         ;ビット 7 は 1 であった
DATAREAD2:
                sin1    6                       ;ビット 6 を受信
                sin1    5                       ;ビット 5 を受信
                sin1    4                       ;ビット 4 を受信
                sin1    3                       ;ビット 3 を受信
                sin1    2                       ;ビット 2 を受信
                sin1    1                       ;ビット 1 を受信
                sin1    0                       ;ビット 0 を受信
                CLR1    PM4.1                   ;ACK 応答
                MOV     A,IICO                  ;受信データ読み出し
                MOV     [HL],A                  ;受信データの保存
                INCW    HL                      ;ポインタ更新
                BF      P4.0,$$                 ;SCL 立ち上がり待ち
                SET1    STATUSFLAG.6            ;受信フラグをセット
                BT      P4.0,$$                 ;SCL 立ち下がり待ち
                SET1    PM4.1                   ;ACK 解除
                BR      !LOOP21                 ;継続処理へ
```



上記のように、ビット 6 以降の処理へ分岐してきた場合も、SCL 信号の立ち上がり前に処理を開始できるので、タイミングの問題はありません。また、SDA 信号がロウの場合に、この処理を共通で使用しようとすると、分岐命令で 6 クロック分追加されますが、それでも十分に SCL 信号がハイの状態でサンプリング可能で、処理でのタイミングの問題もありません。

最後のビット(ビット 0)を受信したら、ACK 応答して、受信データをバッファに保存します。この処理も、SCL 信号の変化に対して十分にマージンがあります。



⑥SDA 信号がロウであったときの処理

ここでは、SCL 信号が立ち上がったときに、SDA 信号がロウの場合の処理を行います。このときの次の状態は、データ受信(⑦)かストップ･コンディション(④)かのどちらかです。

```
;
;         ストップ･コンディションかデータ待ち
;   SCL が立ち下がれば通常データ受信(⑦)
;   SDA が立ち上がればストップ･コンディション(④)
;    ここは、SCL を確認してから SDA を確認する必要がある
;
SDALOW:
STCLOOP3:
                MOV     A,P4                ;SCL と SDA を読み込む
                BF      A.0,$DATAREAD3      ;SCL が立ち下がれば受信へ
                BF      A.1,$STCLOOP3       ;変化なければループ
                BR      $STPCND             ;ストップ・コンディションへ
```

⑦データ受信処理(SDA＝Lのとき)

次データ受信の処理です。最初のビットは受信完了済みなので、残り7ビットの受信でACK応答し、受信データをバッファに書きこみます。この例では、最初のビットを0にし、⑤のビット6受信に分岐させています(共通の処理)。

```
DATAREAD3:
                MOV     IICO,#00000000B         ;ビット 7 は 0 であった
                BR      $DATAREAD2              ;受信処理継続
```

(10) 各種の宣言

このプログラムを使用する上で使用する変数や定数は以下のように定義しています。

```
;*************************************************************************
;                                                                        *
;               データ領域（変数やフラグ）の定義                         *
;                                                                        *
;*************************************************************************
PUBLIC          TXDATABUFF                      ;送信データ用
PUBLIC          RXDATABUFF                      ;受信データ用
PUBLIC          _RXDATABUFF                     ;受信データ用(C 言語用)
PUBLIC          TXPOINT                         ;送信ポインタ
PUBLIC          _TXPOINT                        ;送信ポインタ(C 言語用)
PUBLIC          RXPOINT                         ;受信ポインタ
PUBLIC          _RXPOINT                        ;受信ポインタ(C 言語用)
PUBLIC          SVAO                            ;アドレス
PUBLIC          _SVAO                           ;アドレス(C 言語用)
PUBLIC          IICCO                           ;動作指定
PUBLIC          _IICCO                          ;動作指定(C 言語用)
PUBLIC          STATUSFLAG                      ;動作結果フラグ
PUBLIC          _STATUSFLAG                     ;動作結果フラグ(C 言語用)
PUBLIC          __INTP3                         ;割り込み処理部

;
;       データ・バッファ
;   送信用、受信用で各 16 バイト分を確保しておく
;
        DSEG
TXDATABUFF:
                DS      16                      ;送信データ用バッファ
RXDATABUFF:
_RXDATABUFF:
                DS      16                      ;受信データ用バッファ
;
;       データ・ポインタ
;   送受信のデータ・バッファをアクセスするポインタの保存用
;
        DSEG            saddrp
TXPOINT:
_TXPOINT:
                DS      2                       ;送信用ポインタ
RXPOINT:
_RXPOINT:
                DS      2                       ;受信用ポインタ
```

```
;
;   作業用のフラグ、変数
;
        DSEG            saddr
IICO:
_IICCO:
                DS      1                       ;受信データ作業用
SVAO:
_SVAO:
                DS      1                       ;スレーブ・アドレス保持用
IICCO:
_IICCO:
                DS      1                       ;動作指定用
ACKE            EQU     IICCO.2                 ;ACK 応答制御用
_ACKE           EQU     IICCO.2                 ;ACK 応答制御用
        ;   1：ACK 応答許可、0：ACK 応答禁止
PUBLIC          ACKE
STATUSFLAG:
_STATUSFLAG:
                DS      1                       ;ステータス・フラグ
        ;
        ;ビット 76543210
        ;       00000000        :動作していない
        ;       01000000        :送信完了
        ;       10000000        :受信完了

TXEND           EQU     STATUSFLAG.6
RXEND           EQU     STATUSFLAG.7

TXCOUNT:
                DS      1                       ;送信のビット・カウント用
```

(11) プログラム例を使う方法(アセンブラ)

このプログラムを使用するには、初期設定が必要です。次のプログラムはその例で、初期化完了後はループしているだけです。実際は、**MLOOP** 以降に処理させたいプログラムを記述してください。この例の送信では、ラベル TXDATA に定義されたデータを送信します。受信では、RXDATABUFF に受信データが格納されます。なお、送信か受信かは、マスタから送られてくるアドレスの LSB で指定されるので、通信が始まるまでは分かりません。通信の開始は、**STATUSFLAG**(ビット 7, 6)で分かるので、このフラグを必要に応じて参照してください。

```
EXTRN           TXDATABUFF                      ;送信データ用
EXTRN           RXDATABUFF                      ;受信データ用
EXTRN           TXPOINT                         ;送信ポインタ
EXTRN           RXPOINT                         ;受信ポインタ
EXTRN           SVAO                            ;アドレス
EXTRN           IICCO                           ;動作指定
EXTRN           STATUSFLAG                      ;動作結果フラグ
EXTRN           __INTP3                         ;割り込み処理部
EXTBIT          ACKE                            ;ACK 応答許可
```

```
RSTVECT CSEG    AT 0
        DW              START
        ORG     18H
        DW              __INTP3                 ;割り込みベクタの設定
STKADR  DSEG    IHRAM
        DS              32                      ; スタック領域(32 バイト)
STACK:
        CSEG
TXDATA:                                         ;送信テスト用のデータ
        DB              055H,0AAH,0CCH,00,33H,01,02,03
        DB              04,05,06,07,08,10H,20H,40H,80H
        DB              0C0H,0E0H,0F0H,0F8H,0FCH,0FEH,0FFH
        DB              99H,66H,44H,22H,11H,08,04,02
START:
        MOVW            AX,#STACK
        MOVW            SP,AX
        MOV             PCC,#00000000B
        MOV             WDTM,#01110000b         ;ウォッチドッグ・タイマ停止
        SET1            LSRSTOP                 ;低速クロック停止
        SET1            LVIMK
        MOV             LVIM,#00000000b
        MOV             LVIS,#00000000b         ;電圧を 4V 以上に設定
        SET1            LVION                   ;低電圧検出回路起動
        MOV             B,#0
        DBNZ            B,$$
        BT              LVIF,$$                 ;電源立ち上がり待ち
        MOV             PPCC,#00000000b         ;8MHz 動作に
        MOV             SVA0,#10100000b         ;スレーブのアドレス設定
        SET1            ACKE                    ;ACKE をセット
        MOVW            AX,#TXDATA
        MOVW            TXPOINT,AX
        MOVW            AX,#RXDATABUFF
        MOVW            RXPOINT,AX
        MOVW            HL,AX
        MOV             B,#16
        MOV             A,#0
MEMCLRLOOP:                                     ;受信バッファのクリア
        MOV             [HL],A
        INC             L
        DBNZ            B,$MEMCLRLOOP
        MOV             PM4,#11111111b
        MOV             P4,#00000000b
        MOV             INTM1,#00000000b        ;INTP3 を立ち下がりエッジ検出に
        CLR1            PIF3
        CLR1            PMK3
        EI
;
;   初期設定を完了して通信可能に
;
MLOOP:                                          ;SDA 立ち下がりの割り込み待ち
        NOP
```

```
;
;   ここに必要な処理が入るが、この例では何も行わずループ
;   通信結果は STATUSFLAG のビット 7,6 で示される
;   受信では ACKE ビットがクリアされ、マスタからのアドレス送信に対して NACK 応答
;   するので、受信データ処理後に受信ポインタ(RXPOINT)を再設定
;   ACKE ビットのセット、STATUSFLAG のクリアが必要
;
        BR              $MLOOP
        END
```

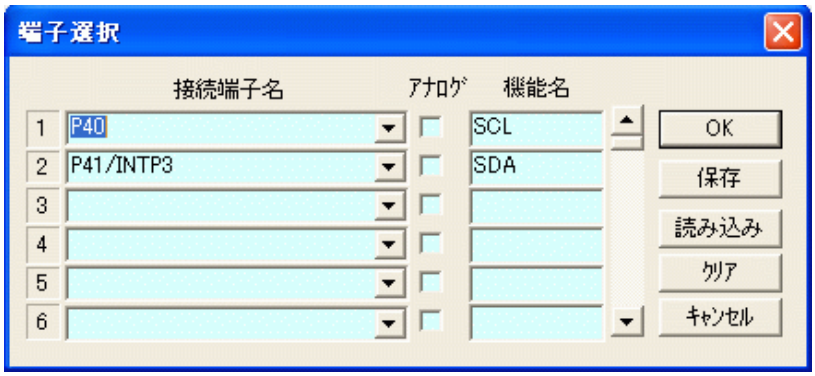
また、オプション・バイトは、次の例のように定義しておきます。

```
    @@OPTB  CSEG    AT      0080H
    DB      10011100b
    DB      11111111b
    END
```

(12) プログラム例の動作確認(アセンブラ)

このプログラムの動作確認やディバッグには、通信相手となる I²C バスのマスタが必要です。ここでは、簡単な手段として、SM+を利用した動作確認の例を示します。

SM+を起動したら、I²C バスの信号波形を確認するために、「**シミュレータ(S)**」のプルダウン・メニューから「**タイミングチャート(T)**」を選択します。その「**編集**」コマンドのプルダウン・メニューで「**端子選択(P)**」を選んで、次のように、確認したい端子を定義します。ここでは、P40 端子を SCL、P41/INTP3 端子を SDA の名前で定義します。



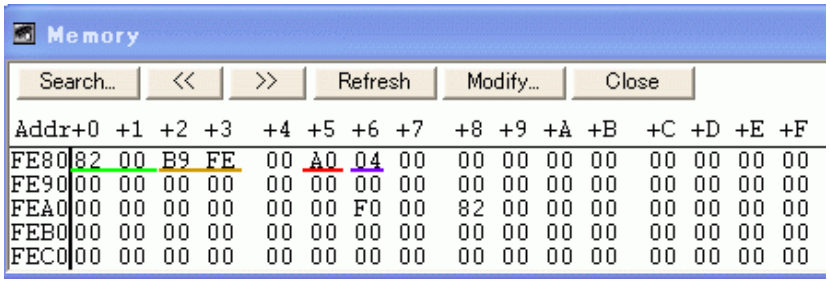
次に「**シミュレータ(S)**」のプルダウン・メニューから「**入出力パネル(P)**」を選択し、「**入出力パネルウィンドウ**」を開きます。「**入出力パネル**」がアクティブな状態で「**部品(P)**」コマンドのプルダウン・メニューから「**プルアップ／プルダウン設定(W)**」を選択し、プルアップ指定を行います。



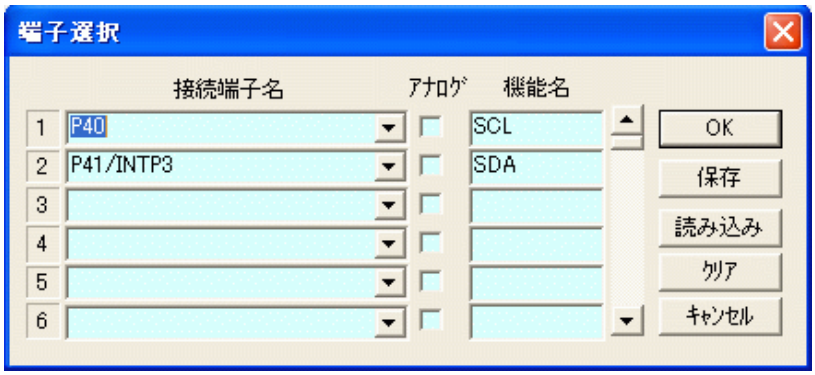
PM+で、リスト・コンバータを起動する設定でビルドすると、絶対的なアドレスを確認でき、変数の定義部分では、次のようなリストが得られます。

```
  75   75 ----                    DSEG↓
  76   76 FEA9                TXDATABUFF:↓
  77   77 FEA9                           DS      16                ;送信データ用バッファ↓
  78   78 FEB9                RXDATABUFF:↓
  79   79 FEB9                           DS      16                ;受信データ用バッファ↓
  80   80                             ↓
  81   81                             ;↓
  82   82                             ;       データ・ポインタ↓
  83   83                             ;□送受信のデータ・バッファをアクセスするポインタの保存用↓
  84   84                             ;↓
  85   85 ----                    DSEG            saddrp↓
  86   86 FE80                TXPOINT:↓
  87   87 FE80                           DS      2                 ;送信用ポインタ↓
  88   88 FE82                RXPOINT:↓
  89   89 FE82                           DS      2                 ;受信用ポインタ↓
  90   90                             ;↓
  91   91                             ;□作業用のフラグ、変数↓
  92   92                             ;↓
  93   93 ----                    DSEG            saddr↓
  94   94 FE84                IICO:↓
  95   95 FE84                           DS      1                 ;受信データ作業用↓
  96   96 FE85                SVAO:↓
  97   97 FE85                           DS      1                 ;スレーブ・アドレス保持用↓
  98   98 FE86                IICCO:↓
  99   99 FE86                           DS      1                 ;動作指定用↓
 100  100      R(0002.2)      ACKE              EQU     IICCO.2     ;ACK応答制御用↓
 101  101                                 ;□1：ACK応答許可、0：ACK応答禁止↓
 102  102                             PUBLIC          ACKE↓
 103  103 FE87                STATUSFLAG:↓
 104  104 FE87                          DS      1                 ;ステータス・フラグ↓
```

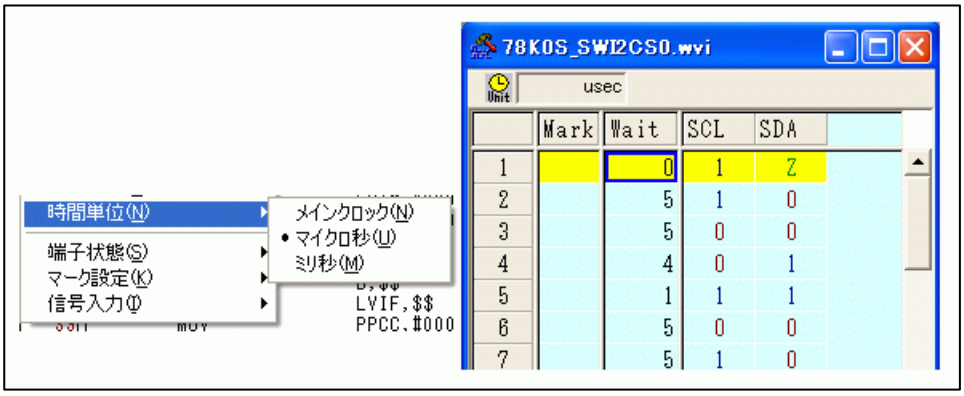
変数を確認するために、SM+の「**ブラウズ(B)**」コマンドのプルダウン・メニューから「**メモリ(M)**」を選択し、このリストで変数のアドレスになっている FE80 以降の領域が見えるようにしておきます。スタート･コンディションで割り込み処理に分岐すると、RAM 内容は次のようになります。**緑線**の部分が送信データのポインタ、**茶線**の部分が受信データの格納用ポインタ、**赤線**の部分はスレーブとしてのアドレス、**紫線**の部分が **ACKE** フラグです。



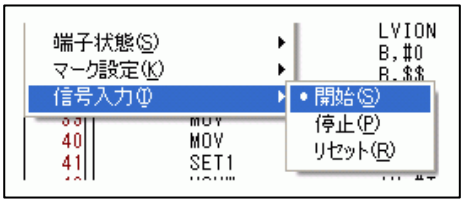
動作確認をするには、マスタからの信号をシミュレーションする必要があります。このためには、「**シミュレータ(S)**」のプルダウン・メニューで「**信号データエディタ(E)**」を選び、その「**編集**」のプルダウン・メニューで「**端子選択(P)**」を選択します。「**端子選択パネル**」で、次のように端子を定義します。



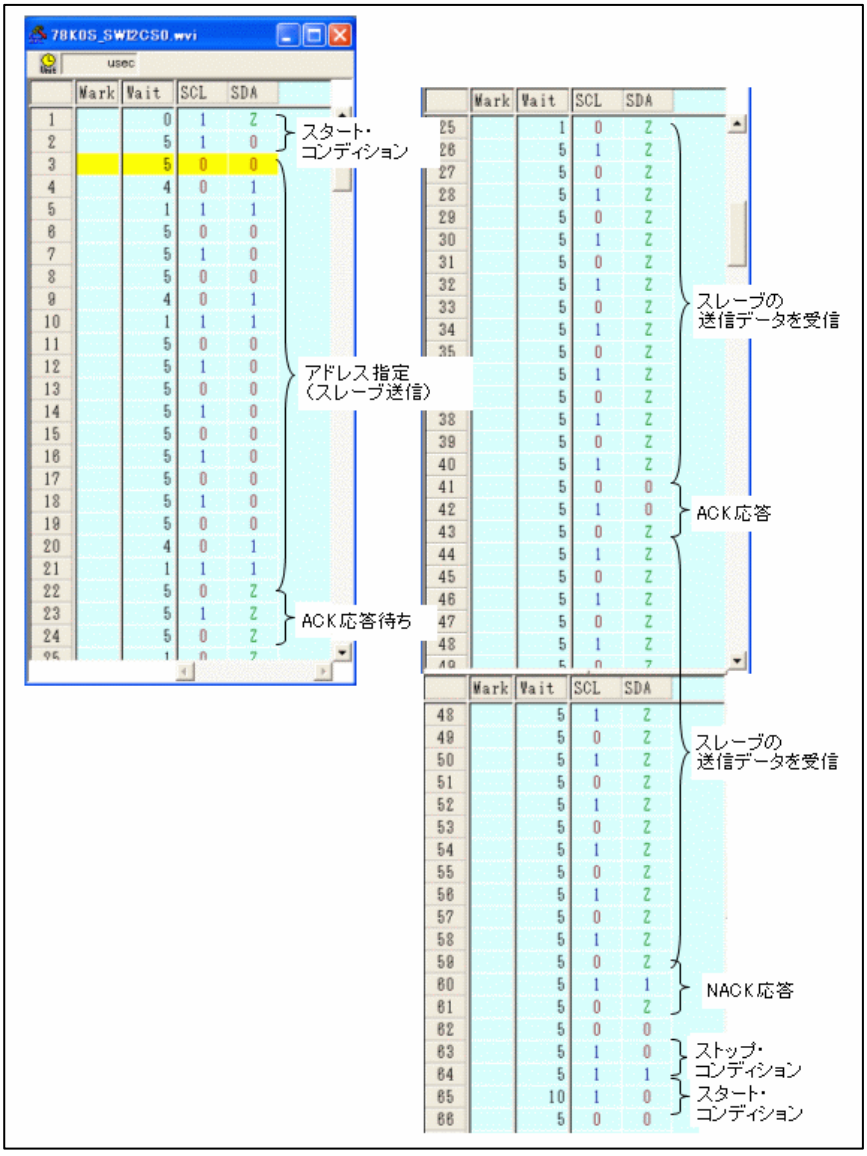
次に、時間の単位設定をします。「**編集**」のプルダウン・メニューで「**時間単位(N)**」を選ぶと、選択肢が表示されるので、「**マイクロ秒(U)**」を選択します。次に、信号パターンを入力します。初期値として、SCL がハイ(1)で SDA をハイ・インピーダンス(Z)にします。2 行目では、5 μs 経過したら SDA を 0 にします。これはスタート･コンディションになります。3 行目で、さらに 5 μs 経過したら、SCL を 0 にしてアドレス送信の準備を行います。標準モードを意識して、SCL のハイ期間とロウ期間を 5 μs にします。



次に、ここで準備した先頭部分のパターンを示します。このデータ・パターンを使用するには、メイン・ルーチンのループ部分にブレーク・ポイントを設定していったん実行させ、ブレークがかかったら、「**編集**」プルダウン・メニューの「**信号入力(I)**」で「**開始(S)**」を選択します。

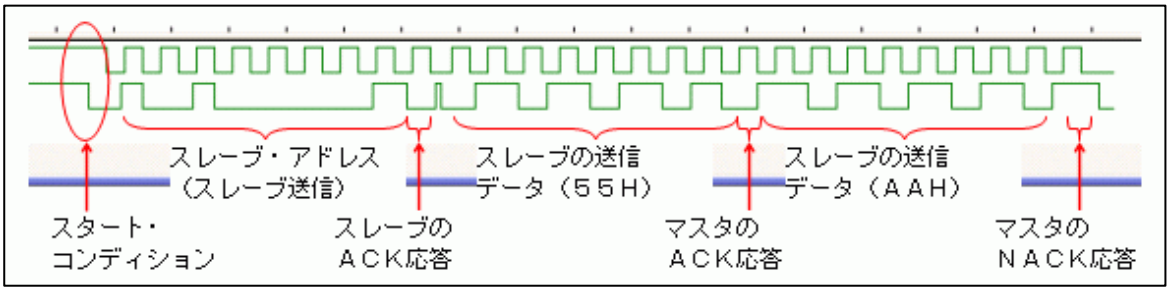


その後、ステップ実行を何回かクリックすると、スタート・コンディションが生成され、それを検出して割り込み処理に分岐します。このときのウィンドウの状態を次に示します。**黄色い**部分が出力中のデータです。

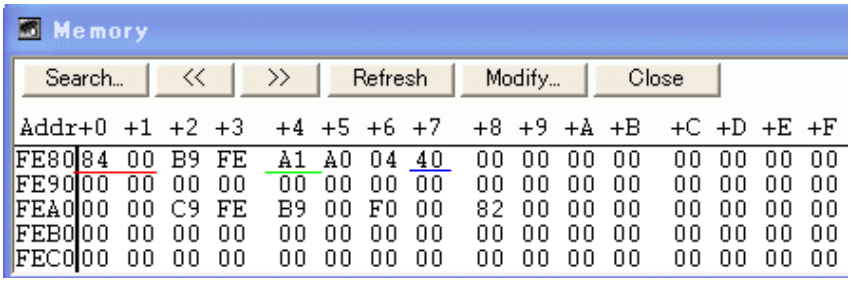


このデータ・パターンは、スレーブ送信でアドレス A0 を出力し、スレーブから 2 バイトのデータを受け取るものです。1 バイト目に ACK 応答すれば、続いてスレーブから 2 バイト目が送信されます。2 バイト目に NACK 応答すれば、スレーブは送信を止めて、通信処理を終了し、割り込みから復帰します。

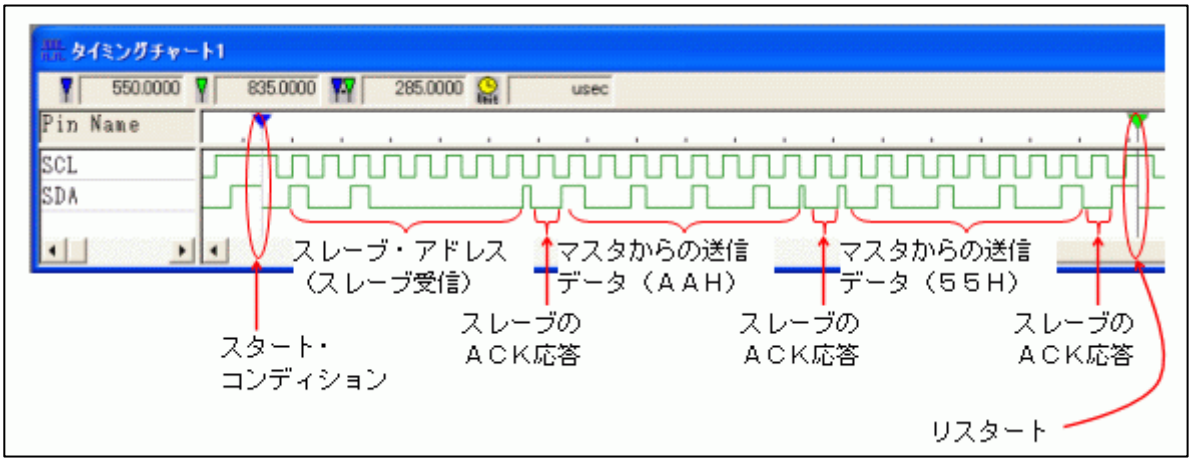
スタート・コンディション検出の割り込みが起動した状態で「**RUN**」させると、メイン処理に戻ってブレークがかかります。ここまでの波形を確認したのが、次のタイミング・チャートです。上の信号が SCL で、下の信号が SDA です。スタート･コンディションの生成から、スレーブ・アドレス転送、それに対するスレーブからの ACK 応答と、その後のスレーブからの送信データ、マスタからの ACK/NACK 応答まで見られます。これは、マスタからの NACK 応答の SCL の立ち下がりを確認して、プログラムが割り込み処理から復帰することと一致しています。



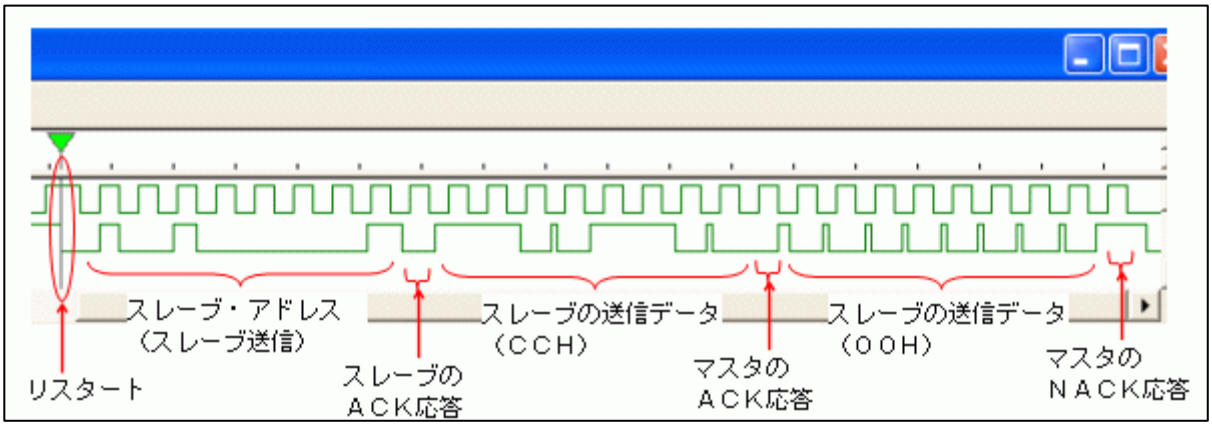
このときの RAM の内容を確認すると、**送信データ**のポインタが 2 バイト分更新されているのが分かります。また、FE87 番地の **STATUSFLAG** が、送信完了の状態になっています。



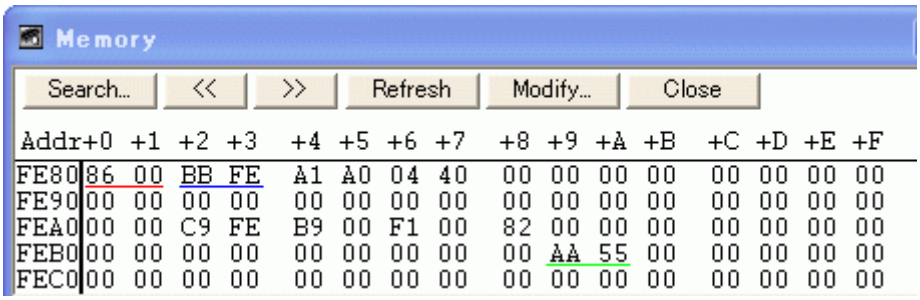
その後、ストップ･コンディションとスタート･コンディションのあとに、スレーブ受信させたときのタイミング・チャートを次に示します。ここでは、マスタからの AAH と 55H の送信後に、リスタート（スタート･コンディション）が発行されています。



次に、リスタート以降のタイミング・チャートを示します。ストップ･コンディションではなく、リスタートでスレーブ受信が完了しているため、続いて、アドレスに対する ACK 応答を行い、スレーブ送信を開始しています。



以上の処理が完了した段階で、RAM の内容は次のようになっています。**送信データ**のポインタが、さらに 2 バイト分更新されているのが分かります。また、**受信データ**用ポインタも更新され、**受信データ・バッファ**に AAH と 55H が格納されています。



(13) このプログラム例を使う方法（C 言語記述）

このプログラムは割り込みで起動し、そのときの I²C のスレーブ・アドレスの LSB の状態で転送方向が決まります。そのため、メイン処理は初期化だけとなります。また、オプション・バイトの定義部分に、使用する割り込み（INTP3）のベクタも合わせて定義しておきます。

```
EXTRN   __INTP3
RSTVECT CSEG    AT 18H
        DW              __INTP3
        @@OPTB  CSEG    AT      0080H
        DB      10011100b
        DB      11111111b
        END
```

C 記述のメイン処理のプログラム例は、次のようになります。

```
#pragma SFR
#pragma EI
#pragma DI
#pragma NOP

extern unsigned char *TXPOINT;
extern unsigned char RXDATABUFF[16];
extern unsigned char *RXPOINT;
extern unsigned char STATSFLAG;
extern unsigned char SVA0;
extern bit ACKE;
```

```
/*-----------------------------------------------------------------*/
/*                                                                 */
/*                                                                 */
/*      Function Body                                              */
/*                                                                 */
/*                                                                 */
/*-----------------------------------------------------------------*/
/*******************************************************************/
/*                                                                 */
/*                                                                 */
/*      Hardware initialize                                        */
/*                              (called by start up routine)       */
/*                                                                 */
/*******************************************************************/
void hdwinit(void)
{
        unsigned char   work1;
        DI();
        WDTM = 0b01110000;              /* stop WDT             */
        LVIMK = 1;
        LVIS = 0x00;                    /* select VLI=4V        */
        LVIM = 0b10000000;              /* Start VLI detection  */
        PCC = 0x00;                     /* CPU clock is fx/4    */
/*
        wait for 0.17ms(=0.2ms-(32+30)*52/1000)
*/
        for (work1 = 0; work1 < 10; work1++){
                NOP();
        }

        while ( LVIF ){
                NOP();
        }       /* wait VDD > 4V */

        PPCC = 0;                       /* CPU clock is fx=fR   */

        PM4 = 0b11111111;               /* set PORT4 as input   */
        P4 = 0b00000000;                /* clear port data to 0 */
        INTM1 = 0b00000000;             /* detect falling edge  */
        PIF3 = 0;
        PMK3 = 0;

        SVA0 = 0b10100000;              /* set slave address    */
        ACKE = 1;                       /* enable ACK           */
}
```

```
/*****************************************************************/
/*                                                               */
/*                                                               */
/*      Main                                                     */
/*                                                               */
/*                                                               */
/*****************************************************************/
void main(void)
{
        const unsigned char txdata[8] =
                {0x55,0xAA,0xCC,0x00,0x33,0x01,0x02,0x03};
        unsigned char   count;
        TXPOINT = txdata;
        RXPOINT = RXDATABUFF;
        for(count = 0 ; count<16 ; count++){
                RXDATABUFF[count] = 0;
        }
        EI();

        while(1){

                NOP();                  /* wait interrupt       */
                NOP();

        }

}
```

ソフトウエアでの I²C バスのスレーブ機能の実現(2) [78K0S/Kx1+]

「ソフトウエアで I²C バスのスレーブ機能の検討」の結果でまとめた、次の条件で I²C バスのスレーブ機能をソフトウエアによるポート制御で実現し、A/D 変換結果の送受信をしてみます。

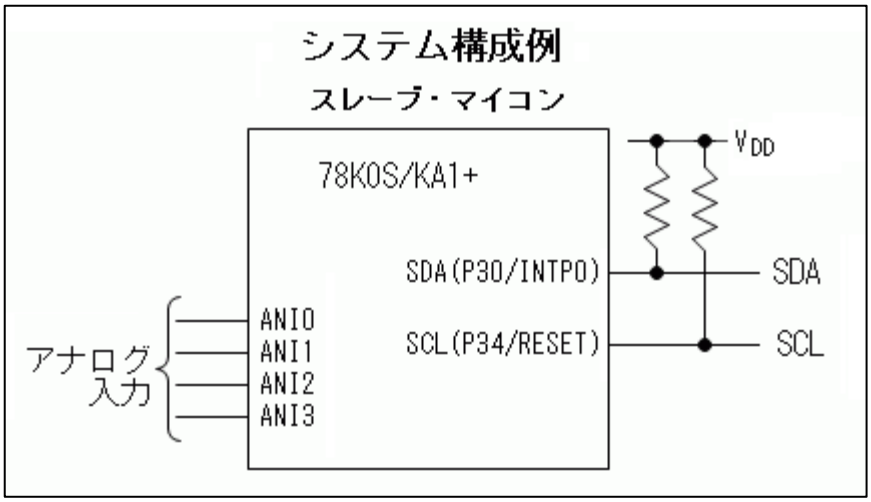
- **・標準モード（クロックが 100kHz 以下）**
- **・シングル・マスタ**
- **・シングル・スレーブ（通信中以外に不要な信号変化がない）**
- **・通信途中（8 ビット目まで）でスタート・コンディションやストップ・コンディションが発行されない（通信前か 9 クロック目以降で発行可能）**
- **・他の割り込みは使用しない（割り込み応答の余裕が 1μs しかないため）**

基本的な処理は、「ソフトウエアでの I²C バスのスレーブ機能の実現(1)」の内容をご参照ください。

(1) システム概要

この例は、アナログ入力を A/D コンバータでデジタル・データに変換して取り込み、その結果をマスタに I²C バスで転送するものです。マスタからは、変換したいアナログ入力チャネルを I²C バスのマスタ送信で指定します。スレーブ・マイコンは、マスタからのマスタ受信指示で変換結果を転送します。

このときのシステム構成例を示します。



この例では、78K0S/KA1+のポート 3（ビット 0 とビット 4）を利用して、I²C バスの通信を行います。同じポートの 2 ビットが使用でき、SDA で使用する端子にエッジ検出割り込み機能があれば、他の端子でも実現できます。

(2) 処理概要

プログラム例を示しながら説明します。ここで対象にするプログラムは、SDA の立ち下がりで発生した割り込みをスタート･コンディション検出に使用します。割り込み処理プログラムの先頭で、ソフトウエアで SCL の状態を確認し、スタート･コンディションなら処理を開始します。その後、通信完了まで、スタート･コンディションを検出した割り込みの中で処理を行います。スレーブ送信ではマスタからの NACK 応答で、またスレーブ受信ではデータ受信後のストップ･コンディション検出で、通信を完了して割り込み処理から抜けます。

データ転送の方向（送信／受信）の判断は、I²C バスの規格に従って、アドレスのビット 0(LSB)で行います。I²C バスについては、「I²C バスについての概要」をご参照ください。LSB の値により、次のような動作を行います。

LSB が 0 ：A/D 変換するチャネルの指定（初期値はチャネル 0）

LSB が 1 ：指定されたチャネルの変換結果を上位バイト･下位バイトの順で送信

ここで使用する A/D は、メイン処理の中で事前に起動しておきます。常時、変換動作を実行し、マスタからの送信要求で、最新の変換結果を送信します。

このプログラムで使用する変数は、次のようになります。このプログラムを使用する場合は、これらを事前に設定しておく必要があります。

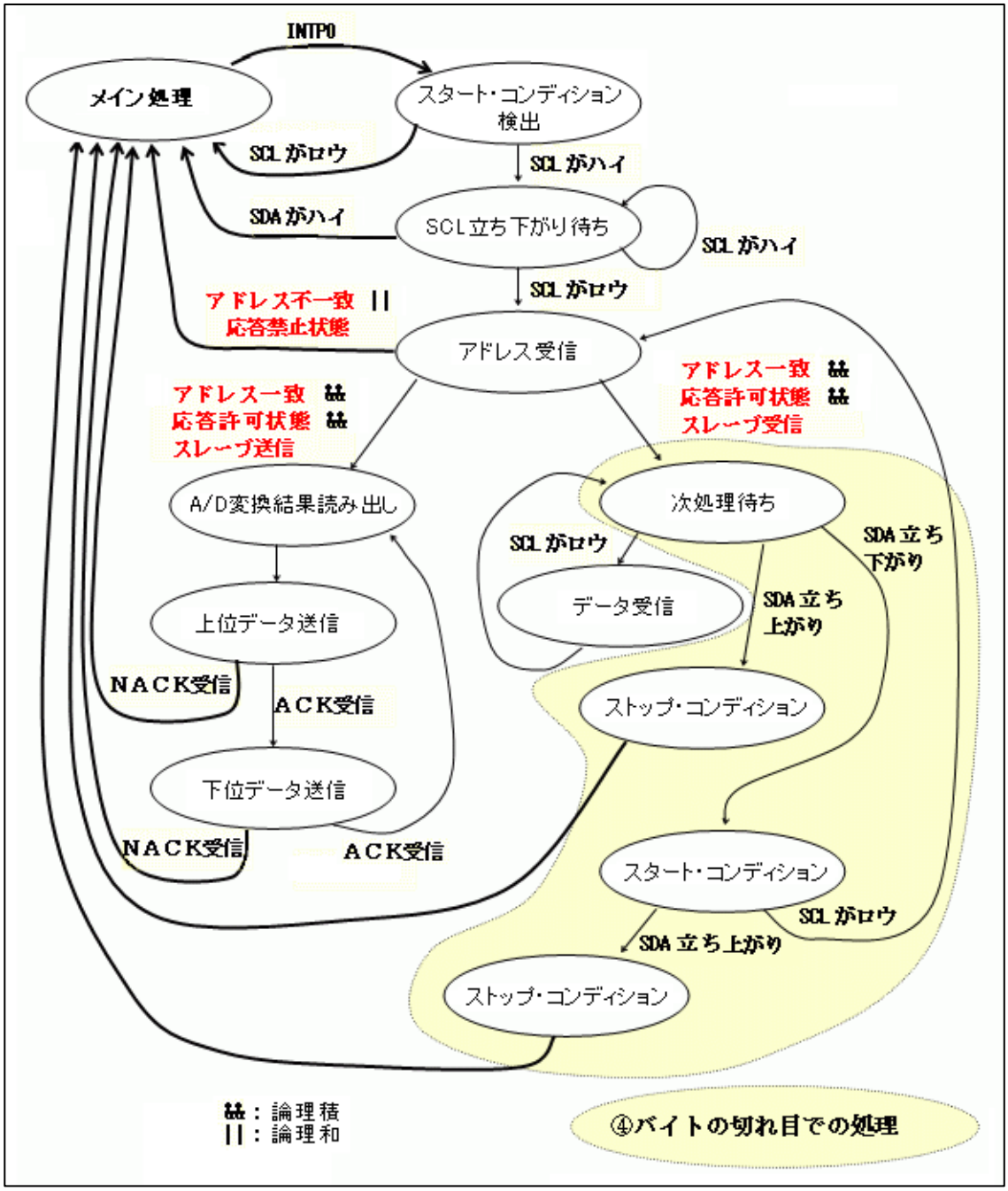
SAV0 ： スレーブとしてのアドレスを保持しています。マスタからのアドレス（上位 7 ビット）がこの値と一致すると、通信を行います。ビット 0 は必ず 0 にしておく必要があります。

IIC0 ： 受信データを構成するために使用します。初期値として、通常は 00 をセットしておきます。

IICC0 ： ビット 2 が ACK 応答するかどうかのフラグ（**ACKE**）です。このビットが 1 のとき、マスタからのアドレスが SAV0 の値と一致したら ACK 応答します。マスタから応答したくないとき、フラグ（**ACKE**）をクリアします。マスタからのアドレスが一致しなかったときにはクリアされるので、通信を許可するにはフラグ（**ACKE**）をセットする必要があります。

(3) 全体の処理

ここで考えている、通信処理全体の概略状態遷移を次に示します。



処理の開始は、SDA 信号の立ち下がり(**INTP0**)とします。SCL 信号のとの関係をチェックして、スレーブ・アドレス受信に移ります。受信したスレーブ・アドレスが一致して、応答許可状態のときに ACK 応答をします。内部処理中で通信したくないときは、応答禁止状態にしておき、アドレスが一致しても ACK 応答しないようにします。これによち、マスタに処理中であることを知らせます（ただし、ここではこの機能を使用していません）。

スレーブ受信の場合は、受信準備をして、「バイト切れ目での処理」になります。データが送信されてきたら、上位 6 ビットをマスクして、アナログ入力チャネルを指定します。複数のデータが送られてきた場合は、最後のデータが有効になります。ここで、スタート･コンディションが検出されると、リスタート処理として、アドレス受信かストップ･コンディション検出待ちをします。ここで、ストップ・コンディションを検出すると、通信処理を完了します。

スレーブ送信の場合は、A/D 変換結果を読み出し、上位バイトから順に送信します。マスタから ACK 応答があれば次のデータを送信し、ACK 応答がなければ（NACK 応答）通信を完了します。ACK 応答が続けば、新しい変換結果を順次送信します。

以下に、詳細な処理について説明します。

(4) 通信の起動(スタート･コンディション検出)

通信を開始するタイミング(スタート･コンディション)は、SDA 信号の立ち下がりエッジを検出したベクタ割り込みで検出するものとします。次のプログラム例では、**SCL 信号がハイでなければ処理しない**ようにしていますが、十分にスタート･コンディション検出が可能です。

```
__INTP0:
                BF      P3.4,$NOTSTART          ;SCL がハイか確認
                PUSH    AX                      ;AX レジスタをセーブ
                MOV     IIC0,#00000000B         ;データ初期値を設定
LOOP0:

                MOV     A,P3                    ;SDA と SCL 読み込み
                BF      A.4,$ADDRESS0           ;SCL の立ち下がりで分岐
                BF      A.0,$LOOP0              ;SDA がロウなら継続
NOTACK:
                POP     AX                      ;AX レジスタを復帰
NOTSTART:
                RETI                            ;ストップ・コンディション検出
```

(5) アドレスの受信

スタート･コンディションを検出したら、**SCL 信号がいったんロウになったことを確認**してから、アドレス受信のために **SCL 信号の立ち上がりを待ちます**。

このときの処理プログラムは、次の例のようになります。SCL 信号がハイのときに SDA 信号が有効として、そのデータがハイの場合は、変数 **IIC0** の該当するビットを 1 にします。この処理を次のようなマクロ **sin1** で定義して、プログラムを読みやすくします。

```
sin1    macro   bitno
        LOCAL   DATAis0,LOOP
LOOP:
                MOV     A,P3                    ;SDA,SCL 読み込み
                BF      A.4,$LOOP               ;SCL 立ち上がり待ち
                BF      A.0,$DATAis0            ;SDA のチェック
                SET1    IIC0.bitno              ;SDA が 1 ならビットをセット
DATAis0:
                BT      P3.4,$$                 ;SCL 立ち下がり待ち
        endm
```

これで、マクロ名とビット番号(マクロの引数)を組み合わせて、次のようにアドレスの受信処理を記述できます。

```
ADDRESS:
                sin1    7                       ;ビット 7 受信
                sin1    6                       ;ビット 6 受信
                sin1    5                       ;ビット 5 受信
                sin1    4                       ;ビット 4 受信
                sin1    3                       ;ビット 3 受信
                sin1    2                       ;ビット 2 受信
```

これで、ビット 7～ビット 2 の受信処理を行います。ビット 1(アドレスの最終ビット)については、アドレス比較が必要なので、できるだけむだな待ちループが発生しないよう、最後の SCL 信号の立ち下がり待ちの前にアドレス比較を行います。以下の処理を実行しても、ウエイトをかけずに SCL 信号のロウ期間に処理を完了して、最後の転送方向受信処理に間に合います。

```
LOOP1:
                MOV     A,P3                    ;SDA,SCL 読み込み
                BF      A.4,$LOOP1              ;SCL 立ち上がり待ち
                BF      A.0,$bit1is0            ;SDA のチェック
                SET1    IICO.1                  ;SDA が 1 ならビット 1 をセット
bit1is0:
                MOV     A,SVA0                  ;アドレスを確認
                SUB     A,IICO                  ;結果をゼロ・フラグに
                BZ      $RESACK                 ;アドレスが一致しないと
                CLR1    ACKE                    ;ACK 応答禁止に設定
RESACK:
                BT      P3.4,$$                 ;SCL 立ち下がり待ち
```

(6) アドレスの受信完了から ACK 応答

アドレスの 8 ビット目（転送方向）を受信したら、アドレスが一致して、ACK 応答可能なときに、SCL 信号の 9 クロック目で ACK 応答を行います。アドレスの一致は最初の 7 ビットでチェックが完了しているので、ここではその結果の **ACKE** フラグを参照して、応答するかどうかを判定します。アドレスが一致していても、**ACKE** フラグがクリアされている（別の処理中で応答できない）場合は、ACK 応答しません。したがって、スレーブから ACK 応答がなければ、マスタはある程度時間をおいて、再度スタート・コンディションから開始する必要があります。ACK 応答しないとき、スレーブの処理はこれで完了です。

転送方向を受信した SCL 信号の立ち上がりから約 7μs で、ACK 応答を戻すことができます。これは、次の SCL 信号の立ち上がりに対して、十分なマージンを確保できます。

```
                sin1    0                       ;ビット 0 受信
                BF      ACKE,$NOTACK0           ;ACK 応答しないなら抜ける
                CLR1    PM3.0                   ;ACK 応答
```

(7) ACK 応答完了

ACK 応答が完了したら、ACK の転送完了（SCL 信号の立ち上がりと立ち下がり）を待ち、ACK 応答を切ります。これでアドレス受信処理は完了したので、あとは転送方向に応じた処理となります。SCL 信号の変化待ちでは、処理に余裕があるので、事前に転送方向で処理を分けておきます。

```
                MOV     A,IICO                  ;アドレスを A レジスタに
                ROR     A,1                     ;CY=0:受信、1:送信
                BNC     $RXDATA                 ;受信なら分岐
```

(8) スレーブ送信でのアドレス 9 クロック目以降の処理（送信準備）

スレーブ送信では、A/D 変換結果（変換は完了している）を読み出し、2 回に分けて送信します。ここで、ループをすると処理が複雑になり、時間がかかる可能性があるので、8 ビットの送信処理を 2 つ並べておきます。

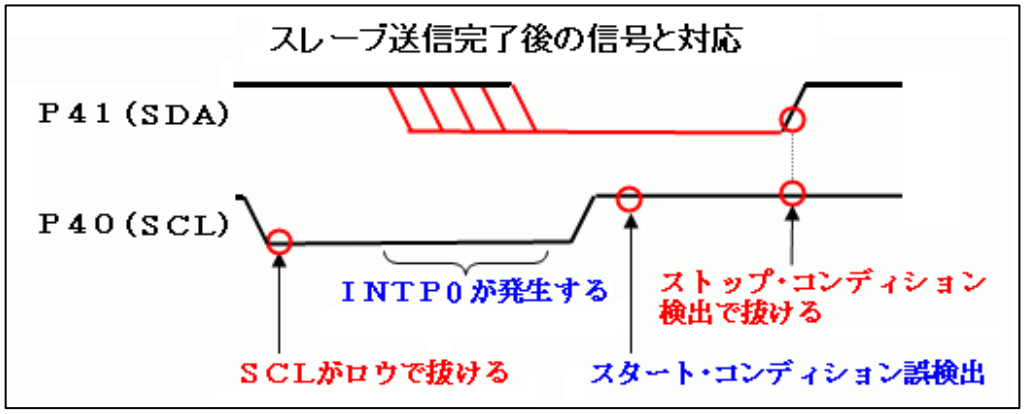
```
TXDATA:
                MOVW    AX,ADCR                 ;変換結果を読み出し
                BF      P3.4,$$                 ;SCL 立ち上がり確認
TXLOOP:
                MOV     TXCOUNT,#8              ;ビット数をセット
```

(9) データ送信処理～ACK 確認

スレーブ送信処理では、SCL 信号の変化を待つだけなので、ループでも十分に処理可能です。そこで、ループ・カウンタ(変数 TXCOUNT)を用いて制御します。8ビットの送信が完了すると、SDA 信号を開放して ACK 応答を待ちます。マスタから ACK 応答があった場合、次のデータに進みます。ACK 応答がなかった場合、処理を終了します。

```
;
;   上位バイト送信
;
TXBLOOP1:
                ROL     A,1                     ;MSB をキャリーへ
                BT      P3.4,$$                 ;SCL 立ち下がり待ち
                SET1    PM3.0                   ;ACK 応答解除
                BC      TXBIT1                  ;データが 1 なら分岐
                CLR1    PM3.0                   ;SDA をロウに
TXBIT1:
                BF      P3.4,$$                 ;SCL 立ち上がり待ち
                DBNZ    TXCOUNT,$TXBLOOP1       ;8 ビット分繰り返し
                BT      P3.4,$$                 ;SCL 立ち下がり待ち
                SET1    PM3.0                   ;SDA 開放
                MOV     A,X                     ;下位データ準備
;
;   ACK 応答確認
;
                BF      P3.4,$$                 ;SCL 立ち上がり待ち
                BT      P3.0,$NOTACK            ;NACK なら抜ける
                MOV     TXCOUNT,#8              ;ビット数をセット
;
;   下位バイト送信
;
TXBLOOP2:
                ROL     A,1                     ;MSB をキャリーへ
                BT      P3.4,$$                 ;SCL 立ち下がり待ち
                SET1    PM3.0                   ;ACK 応答解除
                BC      TXBIT2                  ;データが 1 なら分岐
                CLR1    PM3.0                   ;SDA をロウに
TXBIT2:
                BF      P3.4,$$                 ;SCL 立ち上がり待ち
                DBNZ    TXCOUNT,$TXBLOOP2       ;8 ビット分繰り返し
                MOVW    AX,ADCR                 ;次の変換結果を読み出し
                BT      P3.4,$$                 ;SCL の立ち下がり待ち
                SET1    PM3.0                   ;SDA 開放
;
;   ACK 応答確認
;
NOTACK0:
                BF      P3.4,$$                 ;SCL 立ち上がり待ち
                BF      P3.0,$TXLOOP            ;ACK なら次データへ
NOTACK:
;
;   9 クロック目完了待ち
;
                BT      P3.4,$$                 ;SCL 立ち下がり待ち
                POP     AX                      ;AX レジスタ復帰
                CLR1    PIF3                    ;割り込み要求をクリア
                RETI                            ;処理完了
```

スレーブ送信処理では、マスタからの NACK 応答を確認した段階（9 クロック目）で処理を完了しています。この後、マスタはストップ･コンディションかスタート･コンディションを発行することになります。ここで抜けてしまうと、ストップ・コンディションを発行するときには、次のような信号の動きとなり、SDA を立ち下がりだけを見ているので、**スタート･コンディションを誤検出**します。しかし、INTP0 割り込み処理の中で SCL がロウなら何もしないで抜けるようになっていることと、SCL がハイなら **SDA の立ち上がり（ストップ･コンディション）で抜ける**ので、問題はありません。



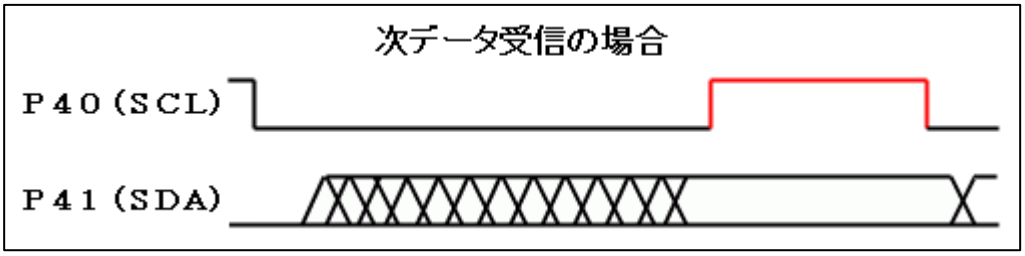
(10) スレーブ受信での 9 クロック目以降の処理

```
RXDATA:
;
;   9 クロック目(ACK 応答完了)待ち
;
                BF      P3.4,$$             ;SCL 立ち上がり待ち
                BT      P3.4,$$             ;SCL 立ち下がり待ち
                SET1    PM3.0               ;ACK 応答解除
                MOV     IICO,#00000000B     ;データ初期値を設定
```

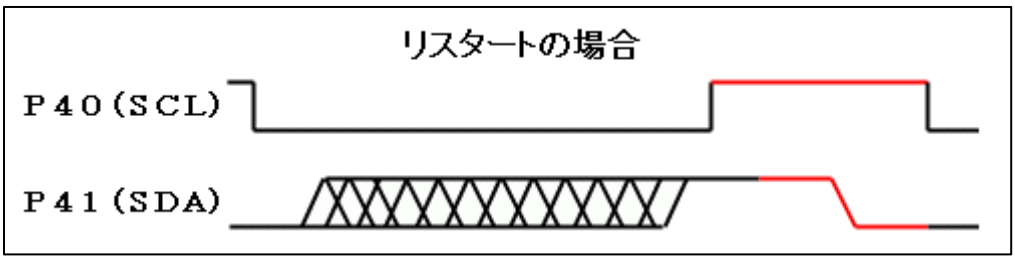
9 クロック目の SCL 信号が立ち下がったら、ACK 応答を解除して SDA 信号を解放します。次に、データの受信処理に移り、以降は繰り返し処理となります。繰り返しで処理する場合、スレーブ受信として次のデータが送られてくる以外に、以下の 3 つの場合が考えられます。これらをポートを用いて確認する必要があるため、処理が複雑になり、タイミング的に厳しくなります。

①次のデータを受信する
②スタート･コンディション（リスタート）を受け付ける
③ストップ・コンディションを受け付ける

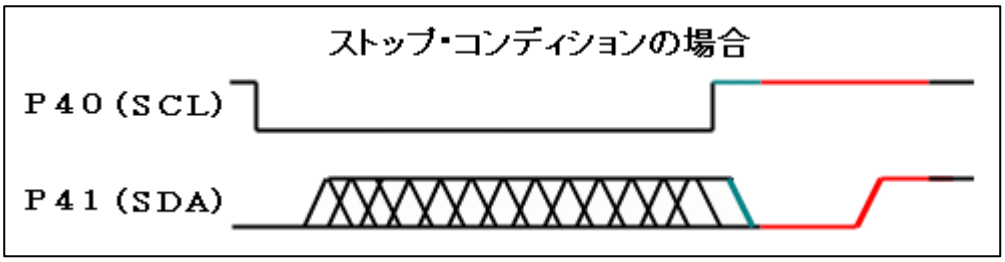
①では、SCL信号がロウの状態で SDA 信号が変化して、SCL 信号が立ち上がります。その後、SDA 信号を保持したまま SCL 信号が立ち下がります。このとき、はじめてデータだと分かります。この場合、SCL 信号のハイ・レベル幅が 4μs 以上で、SDA 信号は 250ns のセット・アップ時間と 0ns のホールド時間で変化する可能性があります。これらの時間は、判別できる時間間隔よりはるかに短いので、**単独か、SDA 信号立ち下がりと同時に、SCL 信号の立ち下がりが検出できたらデータ受信**と判断することになります。



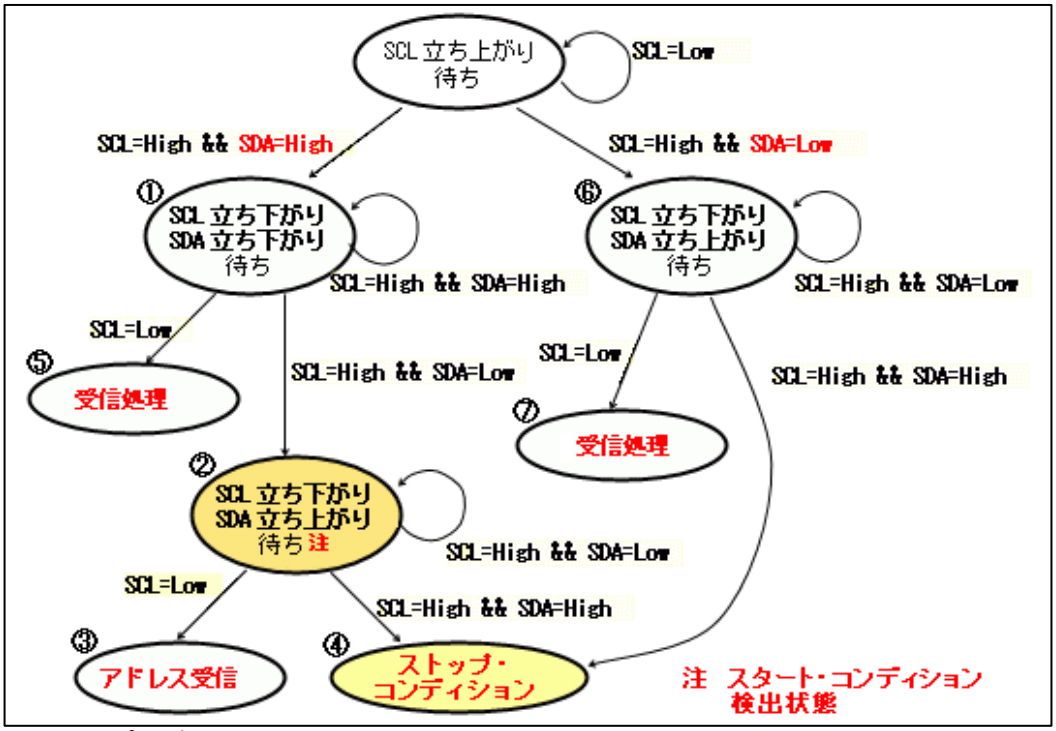
②では、SCL 信号がロウの状態で SDA 信号が立ち上がり、SCL 信号がハイの状態で SDA 信号が立ち下がります。SDA 信号の立ち下がりでスタート･コンディションであることは分かりますが、SCL 信号が立ち下がるまで次に移れません。



③では、SDA 信号をどのタイミングで立ち下げるかで、さらに 2 つの場合が考えられます。SCL 信号がロウの状態で、SDA 信号が立ち下がれば、単純にストップ･コンディションの発行だけとなります。**SCL 信号の立ち上げ以降に SDA 信号が立ち下がった場合**は、そこがスタート･コンディションと同じになります。その後、**ストップ･コンディションが発行**されることになります。つまり、スタート･コンディションを検出した場合は、SCL 信号が立ち下がるか、SDA 信号が立ち上がることを待つ必要があります。



次に、この部分の状態遷移図を示します。



この処理のプログラム例を示します。

①SDA 信号がハイであったときの処理

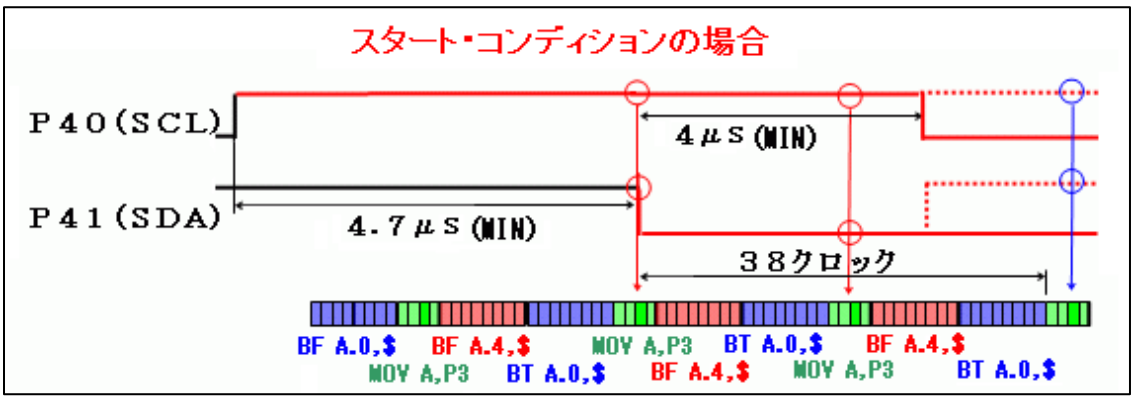
まず、SCL 信号が立ち上がったときに、SDA 信号がハイの場合の処理を行います。SCL 信号の立ち上がりと SDA 信号の組み合わせでの場合分け（下記プログラムの LOOP21 以降の 3 命令）は、SCL 信号の立ち上がりから 30

クロック(3.7μs)以内に処理できるので、SCL 信号の周期が短い(4μs)データ転送の場合でも、タイミングに問題はありません。**STCLOOP** 以降の 3 命令で、データ受信開始かスタート・コンディション(リスタート)を待っています。データ受信では、SCL 信号の立ち下がりから 30 クロック(3.7μs)以内に検出できるので、SCL 信号がロウの期間(4.7μs)に処理可能です(以降の処理は⑤)。

```
;
;       アドレス受信、データ受信したあとの処理の判断
;
LOOP21:
                MOV     A,P3                    ;SCL と SDA を読み込む
                BF      A.4,$LOOP21             ;SCL 立ち上がり待ち
                BF      A.0,$SDALOW             ;SDA がロウなら分岐
;
;       スタート･コンディションかデータ待ち
;   SCL が立ち下がれば通常データ受信(⑤)
;   SDA が立ち下がればスタート･コンディション(②)
;   ここは、SCL を確認してから SDA を確認する必要がある
;
STCLOOP:
                MOV     A,P3                    ;SCL と SDA を読み込む
                BF      A.4,$DATAREAD           ;SCL が立ち下がれば受信へ
                BT      A.0,$STCLOOP            ;変化なければループ
```

②リスタート

リスタートの場合、信号の読み取りが 20 クロックのループになり、1 回は確実に確認できますが、確認処理完了に最大 38 クロックかかります。このとき、次データのために SCL 信号が立ち下がっている可能性があります。その後、4 クロックで信号を確認します(次図の○)が、このときの状態で、次が何かを判断する必要があります。
SCL 信号が立ち下がれば(実線)、リスタート後のアドレス受信(③)の開始です。SCL 信号がハイのまま SDA 信号が立ち上がれば(破線)、ストップ・コンディション(④)です。ここでは、どちらかの信号が変化するまで待ちます。




```
;
;       スタート・コンディション検出
;   検出後、SCL の立ち下がり(通信開始)か SDA の立ち上がりを待つ
;
STCLOOP2:
                MOV     A,P3                    ;SCL と SDA を読み込む
                BF      A.0,$ADDRESS2           ;SCL の立ち下がりでアドレス受信
                BF      A.1,$STCLOOP2           ;ストップ･コンディション以外
```

③アドレス受信

アドレス受信時の分岐先の ADDRESS2 は、本来 ADDRESS ですが、ここからの条件分岐では直接飛べず、また、いくつかの処理が必要なためのクッションです。RXDATA の直前に配置しておく必要があります。

```
;
;       リスタート時にアドレス受信へ戻る処理
;
ADDRESS2:
                MOV     IICO,#0                 ;受信データをクリア
                BR      ADDRESS
```

④ストップ・コンディション

受信を完了して、ストップ･コンディションを検出したら、処理を完了します。これまでの SDA の変化で、PIF0 がセットされているので、クリアしてから割り込み処理を抜けます。

```
;
;       ストップ・コンディション検出
;   通信完了フラグをセット、ACK 応答禁止にして処理を完了する
;
STPCND:
                POP     AX                      ;AX レジスタ復帰
                CLR1    PIF0                    ;割り込み要求をクリア
                RETI                            ;処理完了
```

⑤データ受信処理（SDA＝H のとき）

次データ受信の処理です。最初のビットは受信完了済みなので、残り7ビットの受信で ACK 応答し、受信データをバッファに書き込みます。その後、**(9)スレーブ受信での 9 クロック目以降の処理**の最初に戻ります。

```
;
;       データ受信処理
;   最上位ビットは既に受信済みのため、残り 7 ビット分を受信する
;   受信が完了して、ACK 応答後 9 クロック目でウエイトして処理を抜ける
;
DATAREAD:
                MOV     IICO,#10000000B         ;ビット 7 は 1 であった
DATAREAD2:
                sin1    6                       ;ビット 6 を受信
                sin1    5                       ;ビット 5 を受信
                sin1    4                       ;ビット 4 を受信
                sin1    3                       ;ビット 3 を受信
                sin1    2                       ;ビット 2 を受信
                sin1    1                       ;ビット 1 を受信
                sin1    0                       ;ビット 0 を受信
                CLR1    PM3.0                   ;ACK 応答
                MOV     A,IICO                  ;受信データ読み出し
                BF      P3.4,$$                 ;SCL 立ち上がり待ち
                AND     A,#00000011b            ;受信フラグをセット
                MOV     ADS,A                   ;アナログ入力選択
                BT      P3.4,$$                 ;SCL 立ち下がり待ち
                SET1    PM3.0                   ;ACK 解除
                BR      !LOOP21                 ;継続処理へ
```

⑥SDA 信号がロウであったときの処理

ここでは、SCL 信号が立ち上がったときに、SDA 信号がロウの場合の処理を行います。このときの次の状態は、データ受信(⑦)かストップ･コンディション(④)かのどちらかです。

```
;
;       ストップ･コンディションかデータ待ち
;  SCL が立ち下がれば通常データ受信(⑦)
;  SDA が立ち上がればストップ･コンディション(④)
;   ここは、SCL を確認してから SDA を確認する必要がある
;
SDALOW:
STCLOOP3:
                MOV     A,P3                    ;SCL と SDA を読み込む
                BF      A.4,$DATAREAD3          ;SCL が立ち下がれば受信へ
                BF      A.0,$STCLOOP3           ;変化なければループ
                BR      $STPCND                 ;ストップ・コンディションへ
```

⑦データ受信処理(SDA＝Lのとき)

次データ受信の処理です。最初のビットは受信完了済みなので、残り7ビットの受信でACK応答し、受信データをバッファに書きこみます。この例では、最初のビットを0にし、⑤のビット6受信に分岐させています(共通の処理)。

```
DATAREAD3:
                MOV     IICO,#00000000B         ;ビット 7 は 0 であった
                BR      $DATAREAD2              ;受信処理継続
```

(11) 各種の宣言

このプログラムを使用する上で使用する変数や定数は以下のように定義しています。

```
;*****************************************************************************
;                                                                            *
;               データ領域（変数やフラグ）の定義                             *
;                                                                            *
;*****************************************************************************
PUBLIC          SVAO                            ;アドレス
PUBLIC          _SVAO                           ;アドレス(C 言語用)
PUBLIC          IICCO                           ;動作指定
PUBLIC          _IICCO                          ;動作指定(C 言語用)
PUBLIC          __INTPO                         ;割り込み処理部

;
;  作業用のフラグ、変数
;
        DSEG            saddr
IICO:
                DS      1                       ;受信データ作業用
SVAO:
                DS      1                       ;スレーブ・アドレス保持用
IICCO:
                DS      1                       ;動作指定用
ACKE            EQU     IICCO.2                 ;ACK 応答制御用
_ACKE           EQU     IICCO.2                 ;ACK 応答制御用(C 言語用)
;   1:ACK 応答許可、0:ACK 応答禁止
```

```
PUBLIC          ACKE
PUBLIC          _ACKE                           ;C 言語用

TXCOUNT:
                DS      1                       ;送信のビット・カウント用
```

(12) このプログラム例を使う方法(アセンブラ)

このプログラムを使用するには、オプション･バイトで高速内蔵クロックの使用、P34 を RESET 端子に使用、ウォッチドッグ･タイマを使わなくするために低速内蔵発振器の停止可能を設定します。オプション･バイトの設定例を次に示します。

```
        @@OPTB  CSEG    AT      0080H
        DB      10010100b
        DB      11111111b
        END
```

そのうえで、使用する内蔵周辺機能の初期設定をするメイン処理を作成する必要があります。次のプログラムはその例で、初期化完了後はループしているだけです。実際は、MLOOP 以降に処理させたいプログラムを記述してください。この例では、4 チャネルの A/D を使用可能にして、チャネル 0 を A/D 変換する状態で、通信待ちをしています。なお、送信か受信かは、マスタから送られてくるアドレスの LSB で指定されるので、通信が始まるまでは分かりません。

```
EXTRN           SVA0                            ;アドレス
EXTRN           IICC0                           ;動作指定
EXTRN           __INTP0                         ;割り込み処理部
EXTBIT          ACKE                            ;ACK 応答許可

RSTVECT CSEG    AT 0
        DW              START
        ORG     18H
        DW              __INTP0                 ;割り込みベクタの設定
STKADR  DSEG    IHRAM
        DS              32                      ;スタック領域(32 バイト)
STACK:
        CSEG
START:
        MOVW            AX,#STACK
        MOVW            SP,AX
        MOV             PCC,#00000000B
        MOV             WDTM,#01110000b         ;ウォッチドッグ・タイマ停止
        SET1            LSRSTOP                 ;低速クロック停止
        SET1            LVIMK
        MOV             LVIM,#00000000b
        MOV             LVIS,#00000000b         ;電圧を 4V 以上に設定
        SET1            LVION                   ;低電圧検出回路起動
        MOV             B,#0
        DBNZ            B,$$
        BT              LVIF,$$                 ;電源立ち上がり待ち
        MOV             PPCC,#00000000b         ;8MHz 動作に
        MOV             SVA0,#10100000b         ;スレーブのアドレス設定
        SET1            ACKE                    ;ACKE をセット
```

```
;
;   A/D 変換起動
;
        MOV             ADM,#00100001b          ;A/D 変換は 28μs
        MOV             PMC2,#00001111b         ;4 チャネル使用可能に
        MOV             PM2,#11111111b
        MOV             ADS,#00                 ;初期値はチャネル 0
        SET1            ADCS                    ;A/D 変換開始
;
;   ポート、割り込み初期化
;
        MOV             PM3,#11111111b          ;使用ポートを入力に
        MOV             P3,#00000000b           ;ポートデータをクリア
        MOV             INTM0,#00000000b        ;INTP0 を立ち下がりエッジ検出に
        CLR1            PIF0
        CLR1            PMK0
        EI
;
;   初期設定を完了して、通信可能に
;
MLOOP:                                          ;SDA 立ち下がりの割り込み待ち
        NOP
;
;   ここに必要な処理が入るが、この例では何も行わずループさせている。
;
        BR              $MLOOP
        END
```

(13) このプログラム例を使う方法(C 言語)

このプログラムを使用するには、オプション･バイトで高速内蔵クロックの使用、P34 を RESET 端子に使用、ウォッチドッグ・タイマを使わなくするために低速内蔵発振器の停止可能を設定します。さらに、INTP0 の割り込みベクタの設定を含めた、オプション・バイトの設定例を次に示します。

```
EXTRN   __INTP0
RSTVECT CSEG    AT 8H
        DW              __INTP0
        @@OPTB  CSEG    AT      0080H
        DB      10010100b
        DB      11111111b
        END
```

この機能(割り込み処理)を使用するために、初期化を行うC言語でのメイン処理は、次のようになります。**hdwinit** 関数で、必要な初期化処理を行います。**main** 関数では、A/D コンバータを起動し、割り込み許可にして INTP0 割り込みを待つだけです。この例は NOP 命令でループしているだけですが、必要に応じて処理を追加してください。

```
#pragma SFR
#pragma EI
#pragma DI
#pragma NOP

extern  unsigned char SVA0;
extern bit ACKE;
```

```
/*-------------------------------------------------------------*/
/*                                                             */
/*                                                             */
/*      Function Body                                          */
/*                                                             */
/*                                                             */
/*-------------------------------------------------------------*/
/*************************************************************/
/*                                                             */
/*                                                             */
/*      Hardware initialize                                    */
/*                              (called by start up routine)   */
/*                                                             */
/*************************************************************/
void hdwinit(void)
{
        unsigned char   work1;

        DI();
        WDTM = 0b01110000;              /* stop WDT            */
        LVIMK = 1;
        LVIS = 0x00;                    /* select VLI=4V       */
        LVIM = 0b10000000;              /* Start VLI detection */
        PCC = 0x00;                     /* CPU clock is fx/4   */
/*
        wait for 0.17ms(=0.2ms-(32+30)*52/1000)
*/
        for (work1 = 0; work1 < 10; work1++){
                NOP();
        }

        while ( LVIF ){
                NOP();
        }       /* wait VDD > 4V */

        PPCC = 0;                       /* CPU clock is fx=fR  */

        ADM = 0b00100001;               /* set A/D mode        */
        PMC2 = 0b00001111;              /* use 4-ch for A/D    */
        ADS = 0b00000000;               /* initial channel is 0 */

        PM3 = 0b11111111;               /* set port to input   */
        P3 = 0b00000000;                /* clear port data to 0 */
        INTM0 = 0b00000000;             /* detect falling edge */
        PIF0 = 0;
        PMK0 = 0;

        SVA0 = 0b10100000;              /* set slave address   */
        ACKE = 1;                       /* enable ACK          */
}
```

```
/*****************************************************************/
/*                                                               */
/*                                                               */
/*      Main                                                     */
/*                                                               */
/*                                                               */
/*****************************************************************/
void main(void)
{
        ADCS = 1;                       /* start A/D            */
        EI();

        while(1){

                NOP();                  /* wait interrupt       */
                NOP();

        }
}
```

1.1.14 スタンバイ動作

スタンバイ動作

(1) はじめに

製品設計において、省エネルギが一部の携帯機器だけの話ではなく、ほとんどすべての製品で叫ばれるようになって、かなりの歳月が経ちました。

半導体関係でも、デジタル回路が比較的作りやすかった PMOS、高速化に適した NMOS やバイポーラから、ほとんどの回路が CMOS に置き換わって、より少ない電流で動作できるようになってきました。

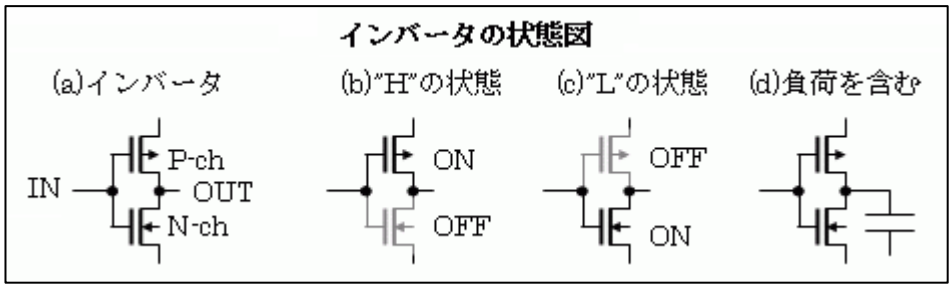
また、プロセスの進歩による回路の微細化により、LSI の動作電流も小さくなり、いくつもの IC やトランジスタで作ってきたものが、1個の LSI で実現できるようになりました。それに伴って、システムとしての消費電力も格段に小さくなりました(その分、システムがより複雑化していますが)。

しかし、世の中の省エネの流れに対して、LSI は単に微細化しても、簡単には動作電流を少なくできにくくなってきています。今後は使う立場からの対策が必要です。そのための手段の1つがスタンバイ機能です。

(2) 電力消費のメカニズム

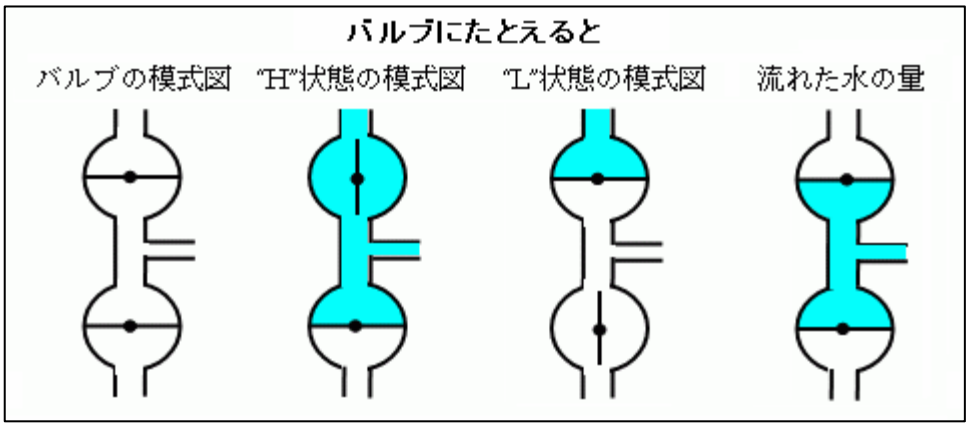
最近のマイコンは、ほとんどが CMOS 化されていますので、これに絞って説明します。中の回路の基本となるインバータは、下図の(a)に示すように、P-ch と N-ch のトランジスタが縦に並んでいます。(b)のように、上側の P-ch のトランジスタが ON、下側の N-ch のトランジスタが OFF で出力は"H"となります。(c)のように、上側の P-ch のトランジスタが OFF、下側の N-ch のトランジスタが ON で出力は"L"となります。これらの状態では OFF しているトランジスタは高抵抗状態となるので、電流はほとんど流れません。つまり、同じ状態が続いていれば電力消費は無視できます。

逆に考えると、信号が変化するときに電流が流れます。(d)に示すように、負荷容量がついた場合(実際には配線の容量が必ずつきます)には、この容量に充電するために余分に電流が流れます。



これを水の流れに置き換えて考えると、インバータをバルブと考えることができます。上から水が供給されていて、バルブでそれを制御します。インバータで"H"の状態は、上側のバルブが開いて下側のバルブが閉じた状態、また"L"の状態は、上のバルブが閉じて下のバルブが開いた状態にたとえることができます。ここで、"H"から"L"に変化した場合、右端に示した分の水が流れることになります。この分を消費した電力にたとえると、分かりやすいと思います。

上の(d)のように負荷容量が付いた場合には、バルブの右に水が溜まる部分が増えますから、流れる水の量はさらに多くなります。



(3) スタンバイ機能

殆どのアプリケーションでは、マイコンが常に動作している必要はなく、空いている時間（例えば、何かの入力を待っているとか、タイマにより時間が経過するのを待っているとか）があります。このときにまで通常動作をさせるのは無駄です。このため、最近のマイコンでは必要のない時には動作を停止させるスタンバイ機能をもっています。スタンバイとかスリープとか呼び名はいろいろありますが、どの機能まで止めるかでいくつかのモードに分けることができます。

基本的には、以下の 3 つのモードが主です。また、このうち(a)と(c)の機能をもつものが一般的です。

(a) HALT モード（CPU の命令実行だけを止める）

(b) IDLE モード（さらに、クロックの供給を止める）

(c) STOP モード（クロックの発振から止めてしまう）

従来は、動作したときの性能だけが問題にされてきました。しかし、いろんな機器の待機時の電力が問題になったり、アプリケーションとして携帯機器の増加により、動作していないときの性能も大きな問題になりました。その重要性は、現在でもますます大きくなってきています。

(4) HALTモード

昔のマイコンからあった機能ですが、本来は単に命令の実行を停止(＝HALT)するだけの命令です。古いマイコンでは、割り込みが入るのを待ったりするために使用するだけで、動作電流の削減とは無関係でした。ところが、CMOS 化により CPU の命令実行を停止すると、動作電流が小さくなることから、省エネのためのスタンバイ機能のひとつになりました。

このモードでは、CPU が停止しているだけで、他の回路は動作していますので、電流は数分の1程度までにしかなりません。それでも、内蔵された周辺機能は動作することができますし、スタンバイが解除されると、CPU は直ちに動作できる(寝起きがよい)ので、高速の応答が要求される場合などにメリットがあります。

(5) IDLEモード

HALT が CPU を停止するだけであったのに対して、その他の部分へのクロックの供給も停止してしまうものです。その結果、動作電流を HALT モードに対して比べても、さらに 1/10 以下にできます。

クロックの発振そのものは継続しているために、その分は STOP モードよりも動作電流は多くなります。しかし、クロックは発振しているので、外部からスタンバイ状態を解除するための信号が入力されると、直ちに動作を再開することができます。

(6) STOPモード

システムとしての動作クロックそのものの発振を停止してしまうので、動作電流は数十μAまで少なくなります。

スタンバイからの解除の際には、クロック発振を再開させる必要があり、クロックの発振が安定するまでは、動作を再開できない(血圧が低くて、起きてもすぐには動けない？)欠点があります。このため、スタンバイ解除から処理完了までの時間に制約がある場合には、使用できない場合があります。

(7) まとめ

以上をまとめると、各スタンバイ・モードの特徴は、一般的に次の表のようになります。

モード	停止部分	内蔵周辺	動作電流	起動
HALT	CPU だけ	動作可能	中	早い
IDLE	クロック供給	一部動作可能	少	早い
STOP	クロック発振	ほとんど動作停止	極少	遅い

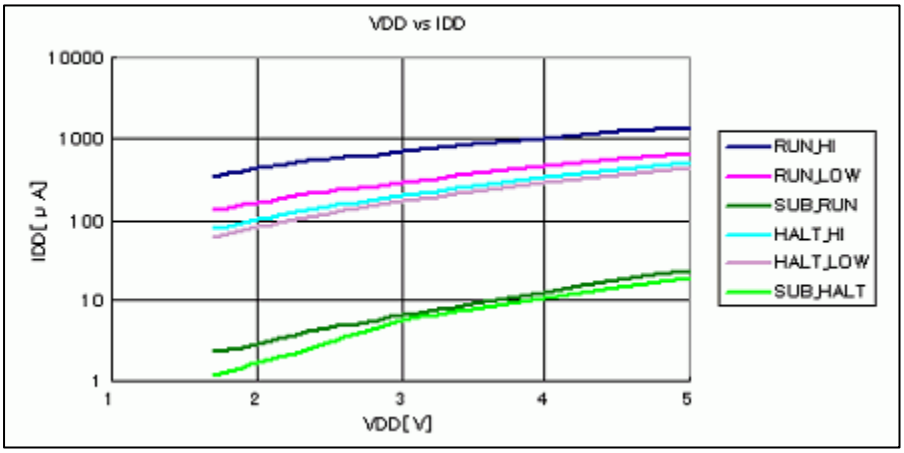
(8) その他の省電力のための手段

電力消費のメカニズムで触れたように、CMOS の動作電流が状態の変化により引き起こされることから、次のような省電力化の方法が考えられます。

- 動作クロック周波数を下げる
- サブクロック動作

マイコンの動作スピードは電源電圧に依存していますので、動作クロックが低い場合には低い電源電圧を用いることで、さらに動作電流を低減することが可能です。

次のグラフは、マイコンの動作モードごとの電源電圧と動作電流の関係例を示したものです。一番上が高速での動作時で、その次が低速(高速の 1/4 の速度)で動作したときのものです。下の2本は、サブクロック(このとき、システム・クロックの発振は停止している)での動作時とHALTモードを示しています。動作クロック周波数と電源電圧による、動作電流の変化を読み取ることができます。

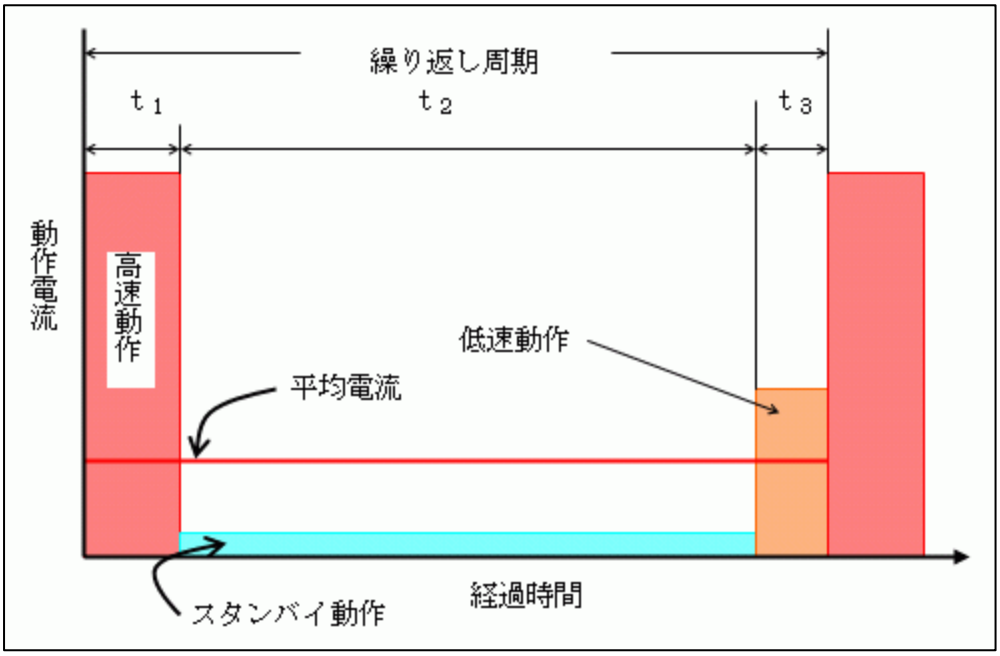


(9) 備　考

(a) 消費電力の計算方法

スタンバイのまとめとして、スタンバイ機能を使ったときの電力消費（動作電流）の見積もりについて検討します。基本的には、動作している各モードでの電流値の時間平均を求めることになります。ほとんどの場合に、それほど厳密に計算する必要はありません。これは、動作電流が個々のマイコン（デバイス）によって変動するためです。下記の例のように、高速動作が動作電流I_1 で時間がt_1、スタンバイ動作が動作電流I_2 で時間がt_2、低速動作では動作電流I_3 で時間がt_3 であったとすると、平均電流は次のようになります。

平均電流＝$(I_1 \times t_1 + I_2 \times t_2 + I_3 \times t_3) / (t_1 + t_2 + t_3)$



(b) 注意点／考慮すべき内容

・スタンバイから復帰時の電流

スタンバイ、特に STOP モードを使用したときに気をつける必要があるのが、クロック発振を開始するときの発振安定時間です。短い期間ですが、発振を開始するための電流を余分に流すので、通常動作と同じ程度の電流が流れると考える必要があります（低血圧で、朝はなかなか起きられない人も、努力はしています。マイコンも、早く起きるための努力をしています）。

・スタンバイ解除要因

スタンバイ状態から復帰（スタンバイを解除）するためには、割り込みを使用します。割り込みとしては、タイマ等の内部要因によるもの、外部からの割り込み要求信号によるものや、外部とのシリアル通信完了によるものなどが考えられます。スタンバイの解除に使える機能は、どのスタンバイ機能を使っているかで異なります。STOP モードでは、内部要因は使えないので、外部からの割り込み要求信号を使う必要がありますが、中にはクロックがないと使えない外部からの割り込み要求信号もあるので、注意が必要です。

・ウォッチドッグ・タイマ

スタンバイとの組み合わせで、一番問題になるのがウォッチドッグ・タイマです。これには、二重の意味で問題があります。1つは、CPU がスタンバイ状態に入っているのに、ウォッチドッグ・タイマが動作している場合です。この場合には、ウォッチドッグ・タイマのクリアに必ず間に合うタイミングで、スタンバイを解除する必要があります。

もう1つは、スタンバイにより、ウォッチドッグ・タイマのカウント・クロックも停止してしまう場合です。この場合には、タイミングの問題はそれほどありませんが、デッドロックを回避するためにウォッチドッグ・タイマの機能に頼り切っていると、問題があります。スタンバイ時にウォッチドッグ・タイマがどうなるかを前もって確認しておいてください。

・動作クロックとの組み合わせ問題

スタンバイとして、3 つのモードを説明してきましたが、新しいマイコンでは内蔵発振器によるクロック（デバイスにより複数あったり、発振を停止できたり、発振を停止できなかったりします）をもつものがあります。これらのマイコンでは、どのクロックで動作させるかにより、使用できるスタンバイ・モードと、その入り方が異なります。たとえば、メイン・クロックで動作しているときには STOP 命令でメイン・クロックの発振を停止できても、他のクロックで動作している場合には、STOP 命令は使用できず、レジスタの設定で制御する必要があります。また、クロックを切り替えるときには、発振の制御も合わせて行うことになります。サブクロックで動作している状態で、メイン・クロックの発振の再開を指示して、直ぐに動作クロックをメイン・クロックに切り替えると、発振が正常に行われていないクロックで動作しようとするために、暴走することが考えられます（STOP 命令でクロック発振が止まった場合には、割り込みによるスタンバイ解除時に発振安定時間がハード的に確保されますが、プログラムで発振を再開した場合には、発振安定時間をソフト的に確保してください）。

これらのことに十分に注意して、省エネ目標を達成してください。

★ **割り込みでの解除**

Q1. スタンバイへの移行直前に割り込み要因が発生するとどうなりますか。

A1. 基本的にスタンバイ状態になって、すぐに解除されます。マスクされていない割り込み要因であれば、割り込み禁止状態や優先順位が低くて保留されていても、スタンバイが解除されます。STOPモードでは発振がいったん停止するので、発振安定待ちをすることになります。

なお、V850ESシリーズで、命令ではなくPSMRレジスタでIDLE/STOPモードを設定する製品では、スタンバイ解除が許可状態で、マスクされていない割り込み要因があれば、これらのモードには移行しません。

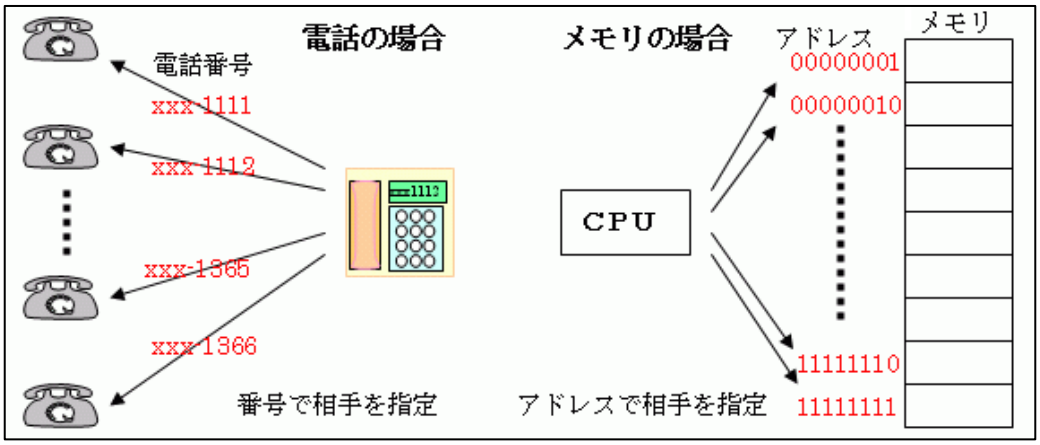
1.1.15 マイコンとメモリ

マイコンとメモリ

[はじめに]

通常、メモリにはプログラムやデータが記憶されており、マイコンが動作するには、必ずメモリをアクセスする必要があります。マイコンは、必要なときに記憶された内容を読み出したり、新たな内容を書き込んだりします。ところが、メモリには大量のプログラムやデータが記憶されているので、どのメモリの内容か、対象を正しく指定できないと必要なデータを得ることができません。このために、メモリには番地（アドレス）が割り当てられています。このため、アドレスを指定することで、対象のメモリを明確に指定することができるようになります。

これは、電話機に 1 台ごとに電話番号が付いていて、ある番号をダイヤルすると、必ず特定の電話機に接続されるのと同じと考えることができます。



78K0 シリーズのような 8 ビットのマイコンでは、レジスタは 8 ビットで、通常扱うデータの大きさは 8 ビットになります。8 ビットで区別（指定）できるメモリは 256 番地分です。通常は、1 番地あたりのメモリの大きさは 8 ビット（1 バイト）ですから、メモリの大きさ（容量）を表すときの単位としてはバイトを使用します。256 番地つまり 256 バイトのメモリでは、プログラムを格納するには不足します。そこで、8 ビットのマイコンでは、2 つのレジスタを組み合わせて、16 ビットでメモリを指定することで、65536 バイトのメモリを使うことができます。このため、8 ビットのマイコンでは、指定できるメモリは 65536 バイトまでが普通でした（65536 では数字の桁数が大きくなりすぎるので、10 ビットで指定できる 1024 バイトを補助単位の K（キロ）で表し、「2^{16} バイト＝2^{6}×2^{10} バイト＝64×1K バイト＝64K バイト」と表します。）

ほとんどの 8 ビット・マイコンの応用では、64Kバイトのメモリがあれば問題ありませんでした。実際、ほとんどの 78K0 や 78K0S では、数Kバイト～32Kバイト程度で、60Kバイト以上のメモリを内蔵している製品は限られます。ただし、一部の 8 ビット・マイコンでは、フォント・データのような大量のデータを扱うために、プログラムは 64Kバイトでも、データは 1M バイトまで指定できるものもありました。これは 78K2 と呼ばれていたマイコンでしたが、マイコンに 64Kバイト以上のメモリを内蔵しているのではなく、外部メモリとして 1M バイトのメモリを扱えるだけでした。最近の、64Kバイト以上のメモリを扱える 8 ビット・マイコン（78K0/Kx2）では、メモリそのものを内蔵しています。

[メモリのアクセス]

マイコンがメモリを使うために、メモリの番地を指定するときに、2 つのことを考慮する必要があります。1 つはプログラムでメモリを指定する方法です（これは論理的なメモリの指定方法です）。プログラムでメモリを指定するときにはレジスタを用います。実行するためのプログラムを読み出すためには、PC（プログラム・カウンタ）と呼ばれる専用のレジスタを使用します。また、データを書き込んだり読み出したりする場合には、2 つの 8 ビットのレジスタを組み合わせて、16 ビットのレジスタとして使います。

もう 1 つは、ハードウエアでの物理的なメモリの指定です。特定の番地を指定するには、そのための信号線を使います。つまり、16 ビットのレジスタを使い、16 本のアドレス信号線を使って、64K バイトの中から目的のメモリを指定します。その上で、制御信号により、データを書き込んだり、読み出したりします。
マイコンでメモリを指定するときに 16 ビットで指定しますが、すべての場合に 16 ビットを使うとは限りません。メモリは、いくつかの領域に分けられています。それにより、より簡単にメモリを指定することができるようになります。78K0 や 78K0S といった 8 ビット・マイコンで、SFR（周辺 I/O 機能が割り当てられた特殊機能レジスタ）領域や saddr 領域[注]と呼ばれるものです。これらの領域には、専用の命令コードが割り当てられており、8 ビットのアドレスだけで、その領域中の目的の周辺機能やメモリを指定することができます。これは、市内に電話をかける場合には、市外局番を指定する必要がないのと似ています。もちろん 16 ビットをフルに使って（市外局番からダイヤルして）、指定することも可能です。
以上のように、メモリが 64K バイトに収まっていれば特に問題ありませんが、8 ビット・マイコンで 64K バイト以上のメモリを指定させる場合は、そのままでは指定できません。このアドレスの拡張機能として、比較的簡単なバンク・スイッチング方式や、より高機能のアドレス・リロケーション方式があります。前者は 78K0/Kx2 で使用されており、後者は 16 ビットの V53 で使用されていました（「メモリ・バンク」もご参照ください。）

注　saddr 領域は、78K0 や 78K0S の 0FE20H～0FF1FH に配置された領域です。他の領域に比べて、より高速にアクセスできたり、アクセスに使用できる命令が豊富であるといった特徴があります。この領域は、レジスタが不足しているような場合に、拡張されたレジスタとしての使い方ができます。

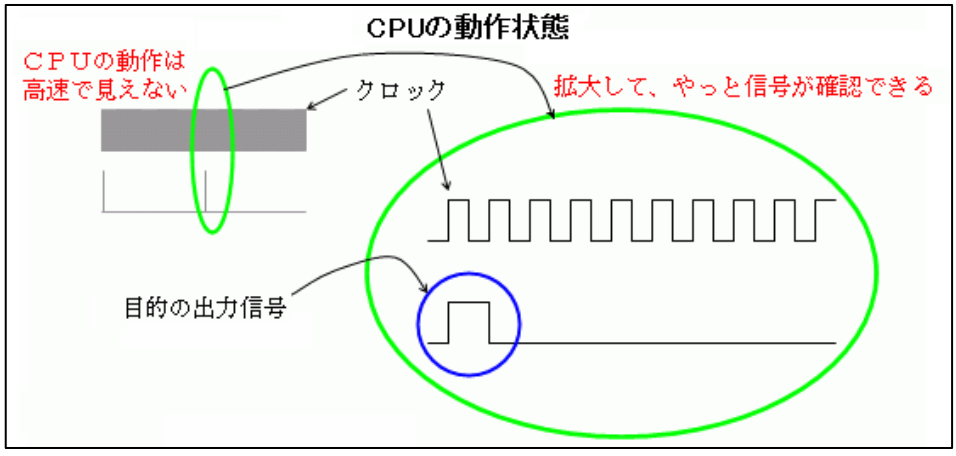
1.1.16 マイコンとポート

マイコンとポート

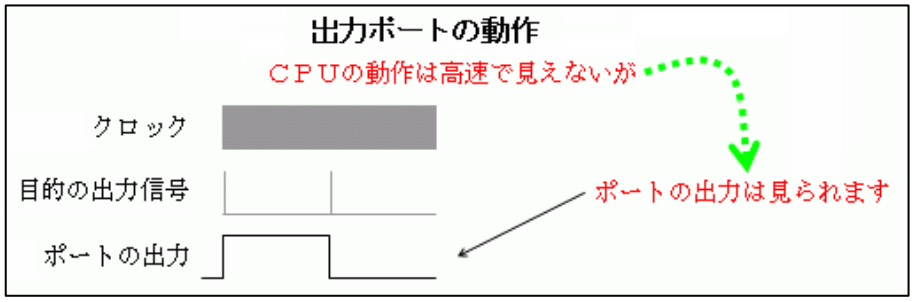
マイコンのポートは、なぜ必要なのでしょうか。また、入力ポートにラッチ機能がないのは、なぜでしょうか。

[出力ポート]

通常、マイコンの CPU は MHz 程度の周波数のクロックを用い、メモリから命令を読み出し（フェッチ）、それをデコードして動作内容を決定し、処理をします。CPU が外部に信号を出力しても、せいぜい μs の幅です。このため、次図の左側のように、波形を見ても分かりません。右側のように拡大すれば、目的の出力信号が確認できます。



これでは、とても制御に使えません。ポートは、CPU と外部を結ぶための港の機能です。上の例にポートを追加すると、次図のようになり、CPU の動作は見えませんが、その結果であるポート出力は見えるようになります。

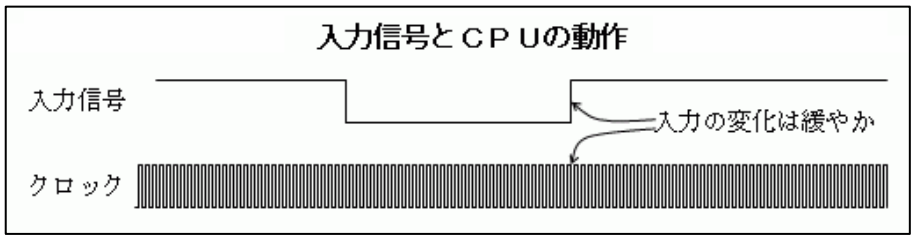


これで、CPU から制御に使える信号を取り出せるようになります。このように、出力ポートは、CPU の出力をラッチしてデータを保持することで、外部で使用できるようにするものです。

[入力ポート]

次に入力について触れます。CPU が外部の状態を取り込むために使用するのが入力ポートで、これは外部の信号を CPU が読むだけの機能です。出力ポートのように、外部の信号をラッチするわけではなく、CPU が必要なときに、入力ポートを介して外部の状態を確認するだけです。したがって、外部の状態が変化しても、CPU が入力ポートを介して読まなければ、変化を認識することはできません。

通常、ポートに入力される信号は、CPU から見て時間的な変化が緩やかです。信号をラッチして保持しておかないと扱えないような信号ではありません。



[入力ポートのエッジ検出]

入力信号の時間的な変化が緩やかであっても、CPU が常に入力ポートを監視するのは効率的ではありません。そのため、タイマを用いて、ある一定の周期で入力ポートを確認（ポーリング動作）するのが一般的です。ほとんどの場合はこれで十分ですが、これでは、変化の検出タイミングには不十分なこともあります。そこで、いくつかの入力はエッジ検出ができるようになっています。信号の変化をエッジ検出割り込みで検出、処理することで、周期的に監視する必要がなくなり、しかも入力信号の変化を早く検出することができます。

★

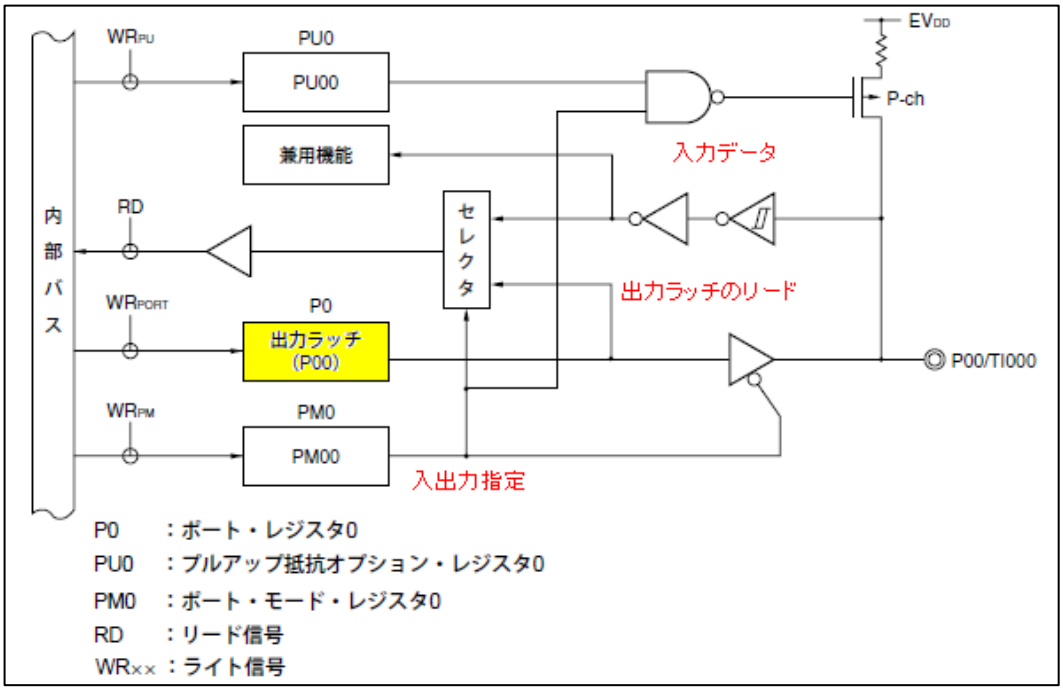
データ・リード

Q1. ポート・レジスタ Pxx の表で、入力モード時は出力ラッチ Pxx の状態を読み出すようになっていますが、外部端子の状態は読み込めないのですか。

Pmn	m = 0-3, 6, 12; n = 0-7	
	出力データの制御（出力モード時）	入力データの読み出し（入力モード時）
0	0を出力	ロウ・レベルを入力
1	1を出力	ハイ・レベルを入力

A1. いいえ、ポート・モード・レジスタ PMxx で入力モードを設定すると、外部端子の状態を読み込めます（入力データ）。このとき、出力ラッチ Pxx の状態は無関係です。

まぎらわしい表現ですが、この表でいう入力とは、ポート・モード・レジスタ PMxx で出力モードを設定しているとき、MOV 命令で出力ラッチ Pxx の状態を読み出せることを意味します。



★

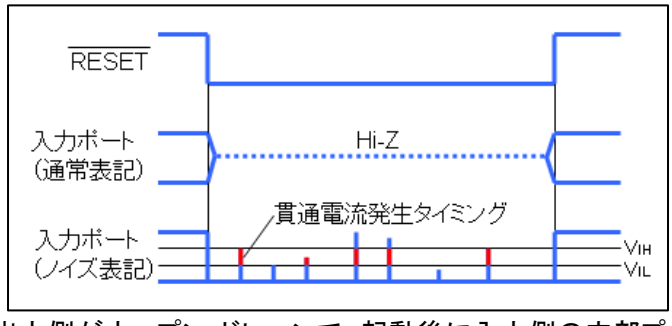
リセット時の状態

Q1. リセット時にポートは入力(Hi-Z)となりますが、貫通電流による劣化などの問題はありませんか。

A1. **入力で使用するポート:**

接続しているデバイスがリセット中でもハイ・レベルかロウ・レベルを出力するなら、中間電位を入力しないので問題ありません。

接続しているデバイスがリセット中に出力を遮断して Hi-Z になる場合、通常の一時的なリセットであれば問題ありません。貫通電流は中間電位の入力で流れますが、Hi-Z は中間電位の状態ではなく、信号源がないためノイズの影響によって、瞬時的に中間電位を遷移するだけです。通常のリセット期間なら信頼性に影響する程度ではありません。



出力側がオープン・ドレーンで、起動後に入力側の内部プルアップ抵抗を有効にする場合も同様で、初期化ですぐにプルアップ設定をすれば問題ありません。

通常はリセットをかけておいて、ある特定のタイミングだけ動作させるような応用では、長期的には劣化を促進させる可能性が考えられます。このような場合には、外部にプルアップ抵抗を接続するなどの処置が必要です。

出力で使用するポート:

リセット解除後の初期化ですぐに出力に設定すれば、入力ポートと同様に、特に問題はありません。

常時リセットをかけておくような応用では、入力ポートと同様に、外部プルアップなどの処置が必要です。

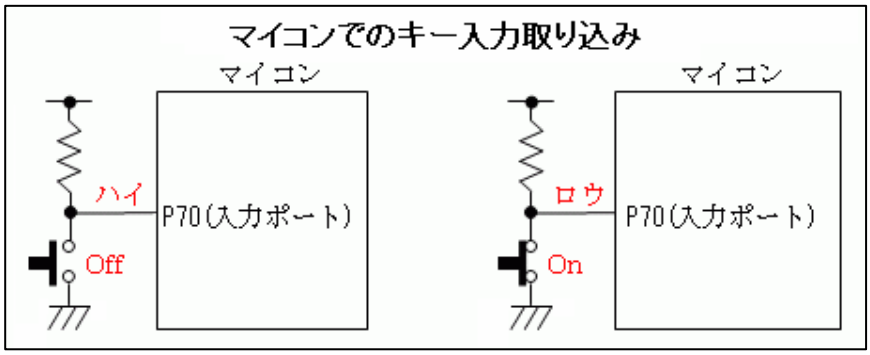
1.1.17 マイコンでのキー・スイッチ入力

マイコンでのキー・スイッチ入力(1)

キー・スイッチは、基本的なマイコン入力のひとつです。ここでは、マイコンでのキー入力の取り込みに触れます。

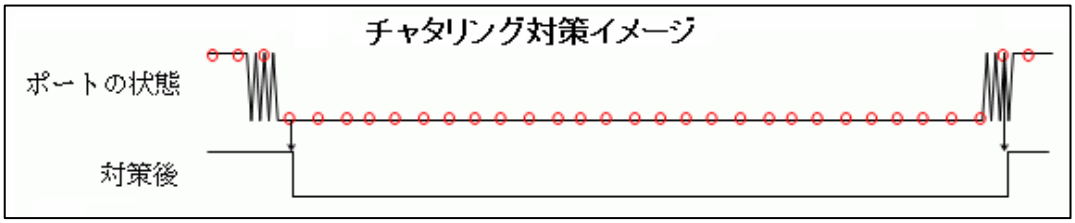
[はじめに]

マイコンでキー入力の取り込みには、入力ポートを用います。基本的な構成の例を次図に示します。この例で、キー・スイッチが押されていないとき(Off のとき)には、入力ポート P70 端子はハイ・レベルとなり、キー・スイッチが押されると(On のとき)、P70 端子はロウ・レベルとなります。したがって、プログラムにより P7 を読み出して、ビット 0 が 1 ならキー・スイッチは押されておらず、0 なら押されていることになります。



[チャタリング対策]

原理的には以上のようになりますが、一般にメカニカルな部品であるスイッチにはチャタリングがあるため、このままでは実用になりません。そこで、時間間隔をつけてポートを読み出します(サンプリング)。このサンプリングのイメージを次図に示します。ここで、丸はポートを読み出す(サンプリングする)タイミングで、定周期で読み出すことを示しています。サンプリングするタイミングは、チャタリングの発生する時間よりも長くとります。部品のばらつきや劣化を考慮すると、長めに設定する必要があります。サンプリングを行うことで、簡単に信号のエンベロープを得ることができ、チャタリング対策ができます。ただし、これだけでは、サンプリング時にノイズがあると、ノイズを信号と誤解する可能性があります。ノイズ対策も行うには、2 回以上連続して同じ状態をサンプリングしたら正しい値とします。具体的なプログラムは、**[チャタリング防止処理]**をご参照ください。

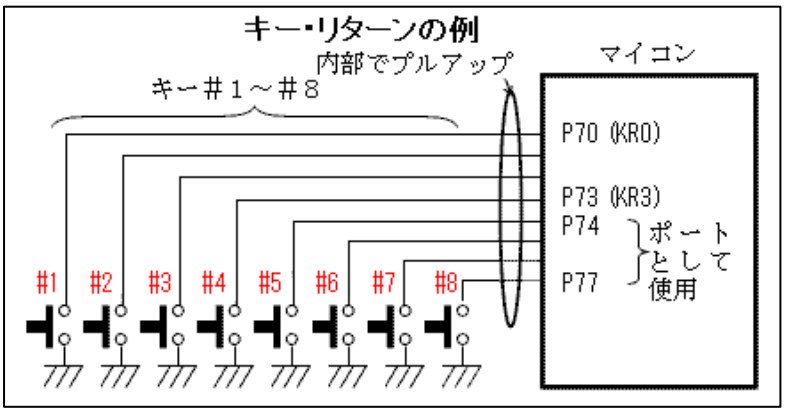


[キー・リターン割り込み／キー割り込み](78K0/Kx2)

以上のキー・スイッチからのデータ取り込みは、定周期でポートの状態を確認することで実現していました。実際のアプリケーションでは、電力消費を抑えるために、通常はスタンバイ状態で、キーが押されると動作状態に復帰するような例があります。このような場合に、割り込みを用いるのですが、通常のエッジ検出による割り込みとは別に、このような用途のために準備された割り込みがあります。それが、キー・リターン割り込みです。

前述の例で示したように、キー・スイッチは通常ハイ・レベルで、キーが押されるとロウ・レベルになります。キー・リターン割り込みは、このロウ・レベルを検出して、割り込みを発生させます。通常のエッジ検出による割り込みとの違いは、KR0～KR7 の 8 本の入力信号のどれか 1 本でもロウ・レベルになったことが検出できることです。

キー・リターン割り込みでキーが押されたことを検出して、スタンバイ解除し、CPU を動作させます。その後、ポート(P7)を読み出して、押されたキーを確認します。78K0/Kx2 のキー・リターン割り込みには、ビット単位でのマスク機能があるので、通常のキー入力用とスタンバイ解除もできるキーを組み合わせて構成できます。たとえば、次の構成では、8 個のキー・スイッチを使用し、マイコンに内蔵されたプルアップ抵抗を利用して、プルアップしてあります。また、P70(KR0)～P73(KR3)に接続した#1～#4 は、キー・リターン割り込みを使用し、#5～#8 は使用しない(KRM でマスクする)ものとします。この構成では、#1～#4 のキーを押すと、キー・リターン割り込みでスタンバイは解除されますが、スタンバイ状態で#5～#8 のキーを押しても、キー・リターン割り込みが発生しないため、スタンバイは解除されません。このようなキーの使い分けも可能です。

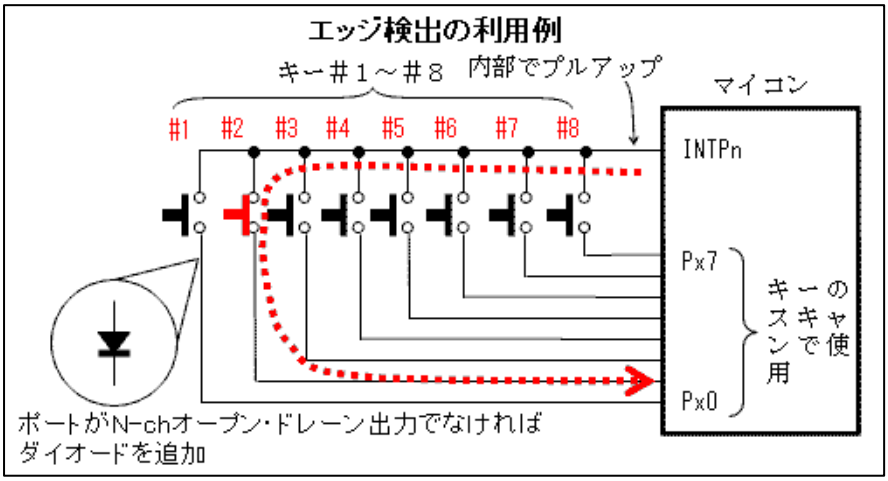


[通常の割り込みによるキー検出]

キー・リターン割り込みによるキーの検出ができない場合は、通常のエッジ検出割り込み入力を用います。ただし、エッジ検出割り込みの数は限られているので、使い方に工夫が必要です。具体的な例を次に示します。

この例で、キー・スキャンに用いる Px0～Px7 は、N-ch オープン・ドレーン出力ポートです。通常の CMOS 出力のポートを使用する際は、ダイオードを図の向きに挿入して、出力ポートから電流が流れ出さないようにします(「**こんな使い方が**」も参照)。

この方式では、キー・スキャン用の出力ポートのどれかひとつにロウ・レベルを出力し、そのときの INTPn 端子を監視することで、キーの状態を検出できます。次図の例で、#2 のキーが押されていたとすると、このキーに接続された Px1 がロウ・レベルのときだけ、INTPn 端子はロウ・レベル、それ以外の出力がロウ・レベルの場合には、INTPn 端子はハイ・レベルとなります。これで、どのキーが押されたかが判断できます。キーの確認を 1 つずつ行うため、キーの判定には最大で 8 回のスキャンを行う必要があります。



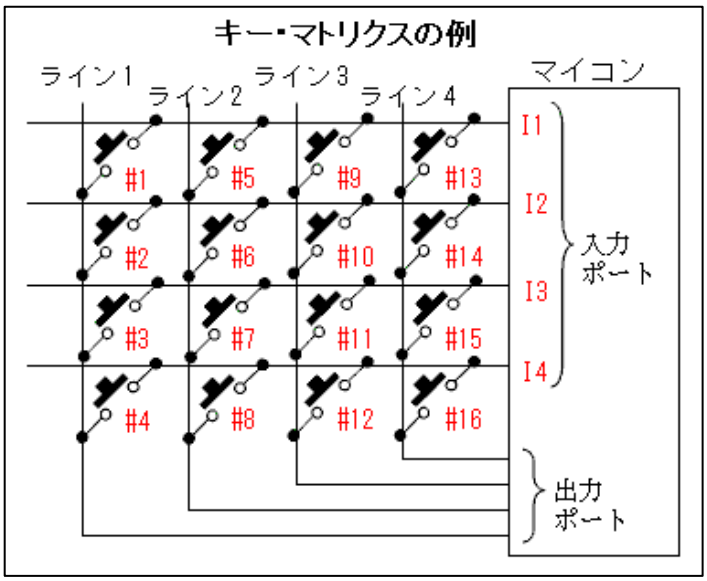
この方式で、キー・リターン割り込みのように、指定した複数のキーのどれかが押されたときに、割り込みが発生することを考えます。たとえば、#1～#3 のキーが押されて INTPn を発生させるには、対応した Px0～Px3 にロウ・

レベルを出力します。割り込みが発生したら、改めて Px0 からキー･スキャンを行い、押されたキーを確認します。

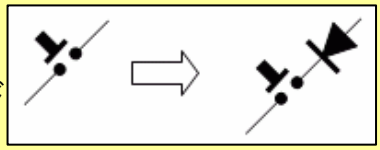
【こんな使い方が】
N-ch オープン・ドレーン出力がなくて、ダイオードも入れられない場合は、どうにもできないのでしょうか。
実は方法があります。ポートの出力ラッチには、すべて 0 をセットしておき、ロウ・レベルを出力したいポートだけ出力ポートに、その他を入力ポートに、PM レジスタで設定します。そうすれば、擬似的に N-ch オープン・ドレーン出力と等価にできます。また、この方法は、ラインをハイ・レベルに立ち上げることもできるので、スキャン時間を短くすることが可能です。

[キー・マトリクス]

使用するキーの数が少ない場合は、キーを直接ポートに接続するだけで済みます。しかし、キーが多くなると、キーの数だけポートを用いるのは効率的ではありません。そこで、出力ポートと組み合わせて、キー・マトリクスを構成することになります。次図に、4×4 のマトリクスの例を示します。入力ポートは、内蔵抵抗でプルアップしているものとします。また、出力ポートは N-ch オープン・ドレーン出力とします。



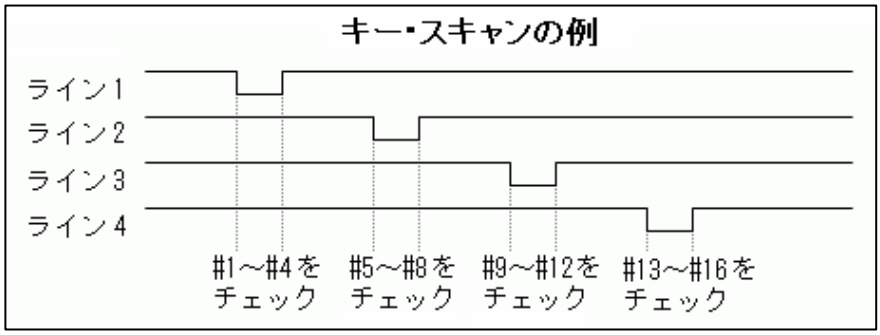
【ちょっと注意 1】
もし、#1 と#5 が同時に押されると、ライン 1 とライン 2 が、#1 と#5 のキー・スイッチを介して、接続された状態となります。このとき、出力ポートが通常の CMOS 出力であれば、ハイ・レベルの出力とロウ・レベルの出力がぶつかることになり、デバイスにダメージを与えたり、キー・スイッチに定格以上の電流が流れたりします。この問題を避けるには、キー・スキャンに用いる出力ポートは、N-ch オープン・ドレーン出力にする必要があります。「こんな使い方が」のように、CMOS出力ポートで入出力を切り替える方法も考えられます。
さらに、#1 と#5 に加えて、#6 が同時に押されると、別の問題が発生します。ライン 1 をスキャンしているとき、#1 と#5 を通してライン 2 もロウ・レベルとなり、この状態で、ライン 2 に接続された#6 が押されていると、I2 入力もロウ・レベルになってしまいます。つまり、押していない#2 まで押されたと認識されます。これは、#5 で、本来とは逆方向に電流が流れるためです。対策としては、ダイオードを追加して、電流を一方向だけにすることがあげられます。
ダイオードを追加すれば、出力ポートとして、N-ch オープン・ドレーン出力の必要性はなくなりますが、キー・スイッチごとにダイオードが必要になるので、コストに影響します。このため、他のキーと同時に押すキーだけにダイオードを付けることもあります。



以上は、オン抵抗が小さいメカニカル・スイッチの場合です。配線やスイッチのオン抵抗をある程度大きくすると、オン／オフの信号ではなく、抵抗分割された電圧の入力になります。こうすれば、上記の 3 つを同時に押した例でも、3 つのスイッチを経由すると、ロウ・レベルまで下がらないので、誤動作を避けられます。

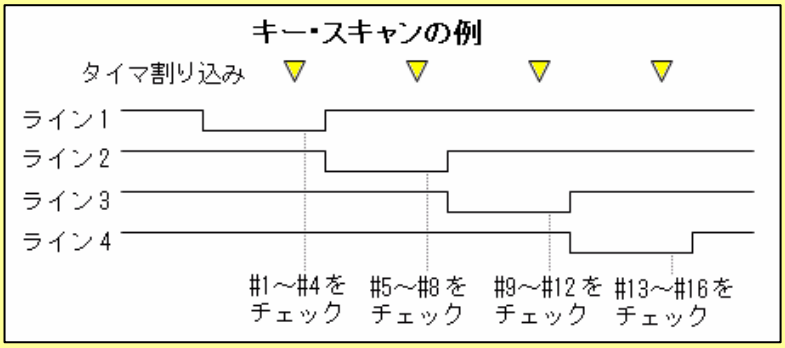
[スキャン]

キー・マトリクスを構成した場合、出力ポートでスキャン・ラインを順番に1ラインずつロウ・レベルにして、入力ポートから読めば、どのキーが押されたか判定できます。上記のキー・マトリクスを使用した場合の、キー・スキャン例を示します。プログラム例は、**[キー・スキャンでのチャタリング防止処理]**をご参照ください。



ライン 1 をロウ・レベルにした（スキャンした）ときに、I1の入力がロウになっていると、#1 が押されていることになります。同様に、ライン2のスキャン中に、I1の入力がロウになると、#5 が押されていることになります。このように、スキャン・ラインと入力の組み合わせで、押されたキーを判断します。

【ちょっと注意 2】
オープン・ドレーン出力の場合は、スキャンが完了して出力をハイ・レベルにしても、すぐに入力ポートの状態がハイ・レベルにはなりません。入力ポートをハイ・レベルに引き上げるのは、プルアップ抵抗の働きです。入力ラインの容量成分を抵抗で充電することになりますので、信号の立ち上がりに遅れが発生します。そのため、スキャンの間隔が短いと、ライン1のスキャンで、#1 のキーが押されてロウ・レベルになった入力が、ライン2のスキャンまで残ってしまい、押していない#5 が押されているように判断される場合があります。
そこで、キー・スキャンとチェックのタイミングを次の例のようにシフトします。スキャンのトリガとなるタイマ割り込みで、前回のスキャン結果をチェックし、その後、次のキー・スキャンを開始して、処理を終えます。これにより、スキャンの間隔分だけ、信号変化の時間に余裕がもてます。
これ以外に、「**こんな使い方が**」で触れた、通常の CMOS 出力で、ロウ・レベルを出力しないときに入力ポート設定する方法では、ロウ・レベルを出力してキー入力を確認したあとで、出力をハイ・レベルにすると、ラインの立ち上がり時間を短くできます。入出力の制御に加えて、出力データも制御するので、手間はかかりますが、スキャン間隔を短くできます。



[チャタリング防止処理]

チャタリング防止処理を行うためのプログラム例を 2 つ示します。どちらも、78K0/Kx2 の P7 をキー入力に用い、インターバル・タイマの定周期割り込みの中で、8ビットのキー入力の状態を確認して、チャタリングやノイズを除去しています。他のデバイスで利用するには、デバイスに合わせて、ポートを P7 から変更してください。また、8ビット以下の場合は、キー・データの読み込みのあとで、使用しないビットをマスクしてください。このプログラムは、定周期に実行されるタイマ割り込み処理の中に、組み込んで使用してください。

<第1のプログラム例>

(1) プログラムの概要

このプログラムは、2 回のスキャン結果を比較して、その結果に応じて次のような処理を行います。

①比較結果が異なるビットは、前回の有効キーのデータをそのまま利用する。

②比較結果が同じビットは、今回のキー・スキャン結果を利用する。

同じ値が 2 回連続しないと、スキャン結果を取り込まないので、ノイズを除去できます。複数のキーがランダムに押される場合を考慮して、変化の判定は条件分岐でなく、ビットごとの演算で処理します。

(2) 使用するデータ領域

このプログラムで得られた有効なキー入力が、KEYDATA に入ります。KEYDATA は、結果の出力用と前回までの有効なキー入力のデータとして、プログラムで参照します。また、前回のキー・データを保持しておくため、作業用に OLDDATA を用います。

(3) 処理手順

①読み込んだキー・データは、次のスキャンで使用するためデータ領域（OLDDATA）に保存し、さらに OLDDATA に保存されていた、前回のスキャン結果と比較（XOR 演算）して、キー・データの変化を検出します。

②XOR 演算結果の論理を反転して、変化したビットを0にし、それとスキャン結果の AND 演算をします。これで、変化したビットは0になり、変化しなかったビットにスキャン結果が入ることになります。

③得られた変化がなかったキー・データを保存し、変化したビットのデータとデータ領域（KEYDATA）に保存されていた、前回の有効なキー入力の AND 演算をします。これで、キー・データが変化しなかったビットは0になり、変化したビットに前回のデータが入ります。

④以上の演算で得られた2つの結果を OR 演算することで、最終的な結果が得られますから、これをデータ領域（KEYDATA）に保存します。

(4) プログラム例

```
KEYCHECK    DSEG    SADDR
OLDDATA:    DS      1           ;前回のキー・スキャン・データ （初期値 0FFH）
KEYDATA:    DS      1           ;前回の有効なキー入力データ （初期値 0FFH）
KEYSCAN:    PUSH    AX          ;AX レジスタをセーブ
        ①   MOV     A, P7       ;キー・データの読み込み
            XCH     A, OLDDATA  ;新データの保存、旧データを A レジスタへ
            XOR     A, OLDDATA  ;比較して変化を求める （変化なければ 0）
            MOV     X, A        ;変化データを保存
        ②   XOR     A, #0FFH    ;変化したビットを 0 にセット
            AND     A, OLDDATA  ;変化しないビットのデータを抽出
        ③   XCH     A, KEYDATA  ;作業中のデータと前回データを交換
            AND     A, X        ;変化したビットは前回データを使用
        ④   OR      A,  KEYDATA ;データを結合
            MOV     KEYDATA, A  ;有効なキー入力を保存
            POP     AX          ;AX レジスタの復帰
            RET
```

<第 2 のプログラム例>

このプログラムでは、キーの変化（押された／はなされた）も情報として得られます。このため、第1の例に変化検出処理を追加してあります。新たなキーが押されたら（入力データ＝0）PUSHKEY、はなされたら RELKEY の該当ビットがセットされるので、変化した情報を知ることができます。なお、この情報は保存されず、スキャンごとに更新されます。

```
KEYCHECK    DSEG    SADDR
OLDDATA:    DS      1           ;前回のキー・スキャン・データ （初期値 0FFH）
KEYDATA:    DS      1           ;前回の有効なキー入力データ （初期値 0FFH）
RELKEY:     DS      1           ;はなされたキー情報 （出力だけ）
PUSHKEY:    DS      1           ;押されたキー情報 （出力だけ）
KEYSCAN:    PUSH    AX          ;AX レジスタをセーブ
            MOV     A, P7       ;キー・データの読み込み
            XCH     A, OLDDATA  ;新データの保存、旧データを A レジスタへ
            XOR     A, OLDDATA  ;比較して変化を求める （変化なければ 0）
            MOV     X, A        ;変化データを保存
            OR      A, OLDDATA  ;変化したビットを 1 にセット
            XOR     A, X        ;変化しないビットのデータを抽出
            XCH     A, X        ;作業中のデータと変化データを交換
            AND     A, KEYDATA  ;変化したビットは前回データを使用
            OR      A, X        ;データを結合
            XCH     A, KEYDATA  ;新しい有効なキー入力を旧データと交換
            XOR     A, KEYDATA  ;有効データの変化情報を得る
            MOV     X, A        ;変化情報をセーブ
            AND     A, KEYDATA  ;はなされた（1 になった）キー情報を抽出
            MOV     RELKEY, A   ;はなされたキー情報を保存
            XOR     A, X        ;押されたキー情報を抽出
            MOV     PUSHKEY, A  ;押されたキー情報を保存
            POP     AX          ;AX レジスタの復帰
            RET
```

【ちょっと注意 3】

2 つのプログラムでは、5 行目までで、2 回のキー・スキャン・データの変化ビットが 1 になり、不変ビットは 0 になって、A と X レジスタに入ります。ここまでは同じで、次の 2 命令が異なっています。しかし、得られる結果は同じになります。

不変ビットには OLDDATA のビットを利用する（キーを取り込む）ため、変化ビット（A や X で 1）はクリアする必要があります。そこで、最初のプログラム例では、A レジスタのデータの補数をとり、OLDDATA との論理積をとることで、変化ビットをクリアしています。2 番目の例では、A レジスタと OLDDATA との論理和をとることで、変化のあった OLDDATA のビットを 1 にし、さらに X レジスタと排他的論理和をとることでクリアしています。この 2 つの演算の関係については、次の式をご参照ください。

$$(A+B) \oplus A = (A+B) \cdot \overline{A} + \overline{(A+B)} \cdot A = \cancel{A \cdot \overline{A}} + B \cdot \overline{A} + \cancel{\overline{A} \cdot \overline{B} \cdot A} = B \cdot \overline{A} = \overline{A} \cdot B$$

[キー・スキャンでのチャタリング防止処理]

(1) プログラムの概要

キー・マトリクスの例で示した、4×4 の 16 キーを対象として考えます。P7 の上位 4 ビットでスキャンを行い、下位 4 ビットで読み込みをします。P7 は通常の CMOS 出力なので、スキャン・ラインからポート出力にだけ電流が流れるように、ポート出力にダイオードを入れるか、「**ちょっと注意**」で紹介したように、各キーにダイオードを入れるものとします。使用するポートの構成やデバイスによって、読み込みやスキャンをするポートを変更してください。

(2) プログラムの概略動作

このプログラムは、キー・リターン割り込みとタイマ(タイマ 51)の 2 つの割り込みを利用しています。最初はキー・スキャンを行わず、ライン 1 を選択して、#1～#4 のキーのどれかが押されるまで、キー・リターン割り込みを待ちます。どれかが押されて、キー・リターン割り込みが発生したら、タイマを起動して、キー・スキャンを開始します。キー・スキャンを開始すると、約 10ms 周期で発生するタイマ割り込みで「ライン 1→ライン 2→ライン 3→ライン 4→ライン 1・・・・・・」とスキャンを行い、4 回のタイマ割り込みで、すべてのラインのスキャンを完了します。スキャンした結果は、ライン 1 から順に、KEYDATA1～KEYDATA4 に格納されます。

(3) 使用するデータ領域

基本的な処理は、前述のチャタリング防止処理の例 1 と同じです。スキャンごとに使用するデータ領域を次のように定義しておき、すべて 0FH で初期化します。ここでは、HL レジスタを用いてアクセスするので、SADDR 領域に配置する必要はありません。また、スキャン・ラインを指定するために、変数をひとつ(SCANLINE)準備します。

```
            DSEG
OLDDATA1:   DS      1           ;前回のライン 1 のデータ
KEYDATA1:   DS      1           ;前回のライン 1 の有効なデータ
OLDDATA2:   DS      1           ;前回のライン 2 のデータ
KEYDATA2:   DS      1           ;前回のライン 2 の有効なデータ
OLDDATA3:   DS      1           ;前回のライン 3 のデータ
KEYDATA3:   DS      1           ;前回のライン 3 の有効なデータ
OLDDATA4:   DS      1           ;前回のライン 4 のデータ
KEYDATA4:   DS      1           ;前回のライン 4 の有効なデータ
SCANLINE:   DS      1           ;スキャン・ライン番号(初期値 0)
STACKTOP    DSEG    IHRAM
            DS      20H         ;スタック領域として 32 バイト確保
STACKP:
```

(4) 初期化プログラム部

プログラムの処理は定周期に実行するため、インターバル・タイマの割り込みを使用します。そのため、初期化部分と実際の処理部分に分けて説明します。デバイス全体の初期化ルーチンに、この処理を組み込んでください。この初期化部分では、ポートのモード、タイマやキー・リターン割り込みの設定を行います。初期状態ではキー・スキャンを行わず、ライン 1(P77 に対応)の#1～#4 を押すと、キー・リターン割り込み要求が発生するように設定します。タイマは初期設定だけして、停止状態にしておきます。

```
RSTVECT     CSEG    AT      0
            DW      START           ;リセット・スタート処理のベクタ
INTVECT     CSEG    AT      2AH
            DW      TM51INT         ;タイマ 51 割り込みのベクタ
            DW      KRINT           ;キー・リターン割り込みのベクタ
;
;   全体の初期化処理
;
            CSEG
START:      MOVW    AX,#STACKP
            MOVW    SP,AX           ;高速 RAM にスタックを設定
;
;   ここには他の部分の初期化処理を入れてください
;
            CALL    !INITKEY        ;キー･スキャンのための初期化
;
;   ここには実行したい処理を入れてください
;
            CLR1    KRMK            ;キー・リターン割り込みマスクを解除
            EI                      ;割り込みを許可
;
;   ここには実行したい処理を入れてください
;
```

(5) キー･スキャン関係の初期化処理

スキャンポートの設定、変数初期化、キー･リターンやタイマの初期化処理をします。キー･リターンやタイマ機能は、まだ起動しません。

```
            CSEG
 INITKEY:   MOVW    HL,#OLDDATA1    ;変数を初期化
            MOV     B,#8            ;変数の数(8 個)をセット
            MOV     A,#0FH          ;初期値データを 0FH に設定
 LOOP0:     MOV     [HL],A          ;初期化
            INCW    HL              ;次のアドレスへ
            DBNZ    B,$LOOP0        ;すべて完了するまでループ
            SUB     A,A             ;A レジスタを 0 に
            MOV     [HL],A          ;スキャン・ライン番号を初期化
            MOV     P7,#01111111b   ;ライン 1 を選択
;
;   キー・リターンの初期化
;   初期化だけで、まだキー・リターン割り込みはマスクのまま
;
            SET1    KRMK            ;キー・リターン割り込みマスク
            MOV     PM7,#00001111b  ;ポート 7 の 0-3 を入力、4-7 を出力に
            MOV     PU7,#00001111b  ;P0-3 のプルアップ抵抗を有効に設定
            MOV     KRM,#00001111b  ;キー・リターンの KR0-3 を有効に設定
            CLR1    KRIF            ;キー・リターン割り込みをクリア
```

```
;
;    タイマの初期化(インターバルは約 10ms に設定)
;    タイマのインターバル設定だけで、まだ起動せず
;
             MOV     TCL51,#00000111b  ;カウント・クロックを 1.95kHz(@8MHz)に
             MOV     TMC51,#0          ;タイマはインタバル・モードで停止
             MOV     CR51,#19          ;インターバルを 20 に設定
             RET
```

(6) キー・リターン割り込みによるキー・スキャンの起動処理

キー・リターン割り込みが発生して、この処理が動作すると、キー・スキャンのタイマを起動します。以降はキー・リターン割り込みを禁止します。

```
KRINT:       SET1    KRMK              ;キー・リターン割り込みマスク
             CLR1    TMIF51            ;タイマ割り込みをクリア
             CLR1    TMMK51            ;タイマ割り込みのマスクを解除
             SET1    TCE51             ;タイマ 51 の起動
             RETI                      ;割り込み処理の終了
```

(7) タイマ割り込み処理

定周期に発生するタイマ割り込みの処理の中から、キー・スキャン処理をサブルーチンとして呼び出します。

```
TM51INT:                               ;タイマ割り込み処理
;
;    定周期タイマでのその他の処理を実行する
;
             CALL    !KEYSCAN          ;キー・スキャン処理
             RETI                      ;割り込み処理の終了
```

(8) キー・スキャン処理

定周期タイマ割り込みで起動され、実際にキー・スキャンをします。この処理は、大きく 3 つの部分に分かれます。最初に、スキャンした結果を保存するためのポインタを計算しています。できるだけ入力パラメータを少なくして、さらに条件分岐を使わないで済ませたいので、計算で必要な情報を得ています。

次に、実際にポートからスキャン・データを読み込み、チャタリングやノイズ対策をします。このような厳密な処理が不要で、単にサンプリングでのチャタリング対策だけで済ませる場合は、この部分はポートからスキャン・データを読み出して、データ領域に保存するだけでも結構です。

最後に、次のキー・スキャンの準備をします。スキャン・ラインを次のラインに変更し、新しいラインをスキャンするためのデータをポートに出力します。

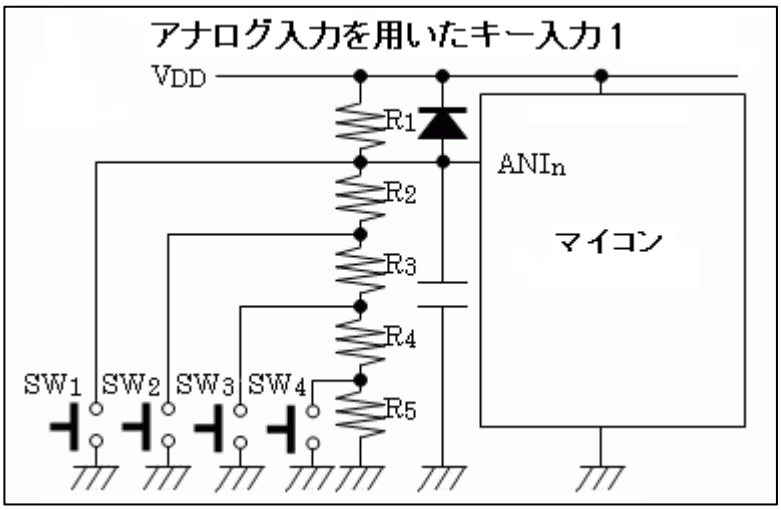
```
;
;    定周期タイマの割り込みで起動されるキー・スキャン処理ルーチン
;
             CSEG
KEYSCAN:     PUSH    AX                ;AX レジスタをセーブ
             PUSH    HL                ;HL レジスタをセーブ
```

```
;
;    スキャン・ラインに対応したデータ領域のアドレスを HL レジスタに設定
;
            MOV     A,!SCANLINE     ;スキャン・ラインの情報をリード
            ADD     A,A             ;ライン情報からオフセット情報を得る
            ADD     A,#LOW OLDDATA1 ;下位アドレスを得る
            MOV     L,A             ;L レジスタに下位アドレスをセット
            MOV     A,#0            ;A レジスタをクリア
            ADDC    A,#HIGH OLDDATA1;上位アドレスを得る
            MOV     H,A             ;HL レジスタにスキャン情報アドレス
;
;    スキャン・ラインのキー入力の取り込みとノイズ／チャタリング対策
;
            MOV     A,P7            ;キー・データの読み込み
            AND     A,#00001111b    ;未使用ビットのマスク
            XCH     A,[HL]          ;新データの保存、旧データを A レジスタへ
            XOR     A,[HL]          ;変化を求める(変化なければ 0)
            MOV     X,A             ;変化データを保存
            XOR     A,#0FFH         ;変化したビットを 0 に
            AND     A,[HL]          ;変化しないビットのデータを抽出
            INCW    HL              ;HL レジスタは有効データを指す
            XCH     A,[HL]          ;作業中のデータをセーブ
            AND     A,X             ;変化したビットは前回データを使用
            OR      A,[HL]          ;データを結合
            MOV     [HL],A          ;新しい有効なキー入力を保存
;
;    スキャン・ラインの更新
;
            MOV     A,!SCANLINE     ;スキャン・ラインの情報をリード
            INC     A               ;スキャン・ラインを更新
            AND     A,#00000011b    ;オーバフローをマスク
            MOV     !SCANLINE,A     ;新しいスキャン・ライン情報をセーブ
            SUB     A,#1            ;ライン 1 ならキャリーをセット
            NOT1    CY              ;キャリー・フラグを反転
```

【ちょっと注意 4】
スキャン・ラインの更新処理では、スキャン・ラインが 4 本(0～3 の 2 ビット)で表せることを利用しています。＋1 して、下位の 2 ビット以外をマスクすることで、新しいスキャン・ラインを得ることができます。
また、実際にポートをスキャンするデータについては、1 ビットだけ 0 にしたデータを用いますが、上位 4 ビットしか使用していないので、単に右回転するだけでは不十分です。ライン 4 までスキャンした段階で、ポートのデータは 1110XXX となっています。このまま右回転してもビット 7 を 0 にできません。そこで、このプログラムでは、ライン 1(スキャン・ライン)が 0 のときだけキャリーをクリアし、それ以外はキャリーをセットしておいて、キャリーを含めて右回転することで、ライン 1 のときにビット 7 を 0 にしています。
このような処理によって、条件分岐命令を不要にしています。

マイコンでのキー・スイッチ入力(2)

一般に、マイコンでキー入力を行うときにはポートを使用します。しかし、使えるポートの数が少ない場合は、キー入力のためだけにポートを何本も使うことはできません。そこで、考えられるのがアナログ入力を用いたキー入力です。次図に、アナログ入力を用いて、4 個のキー入力を行う例を示します。



[考え方]

SW1～SW4 のどのスイッチが押されるかで、抵抗分割されてアナログ入力端子（ANIn）に加わる電圧が異なります。その電圧を A/D 変換することで、どのキーが押されたかを判定できます。

[抵抗値の設定]

それでは、使用する抵抗値をどのようにすればよいのでしょうか。まず決める必要があるのは、「どのような電圧を基準として、入力を判定するか」です。たとえば、簡単のために、0～0.8V_{DD} を、0.2V_{DD} 刻みに分割する場合を考えます。このとき、入力電圧と押されたスイッチの関係は、次のようになります。

0.8V_{DD} 以上　　：どれも押されていない

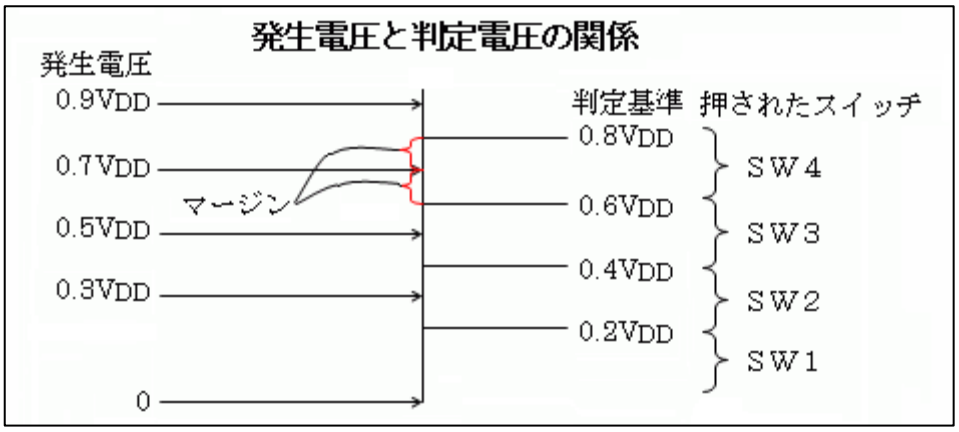
0.6V_{DD}～0.8V_{DD}　：SW4 が押された

0.4V_{DD}～0.6V_{DD}　：SW3 が押された

0.2V_{DD}～0.4V_{DD}　：SW2 が押された

0～0.2V_{DD}　　：SW1 が押された

そこで、この判定値のできるだけ中央になるように、0.1V_{DD} のマージンをもたせて、抵抗値を決めていきます。その際に、R1 を基準（＝100）にして考えます。



まず、SW2 が押されたときに $0.3V_{DD}$ になるようにR2 を決めると、R2＝R1×3/7 となるので、抵抗の E24 系列で近い値は 43 となります。次に SW3 が押されたときに $0.5V_{DD}$ に近くなるR3 を決めると、E24 系列では 56 が近い値となります。同様にして求めていくと、R4 は $0.7V_{DD}$ から 130、R5 は $0.9V_{DD}$ から 620 となります。

	R1 を元に求めた値
R1	100
R2	43
R3	56
R4	130
R5	620

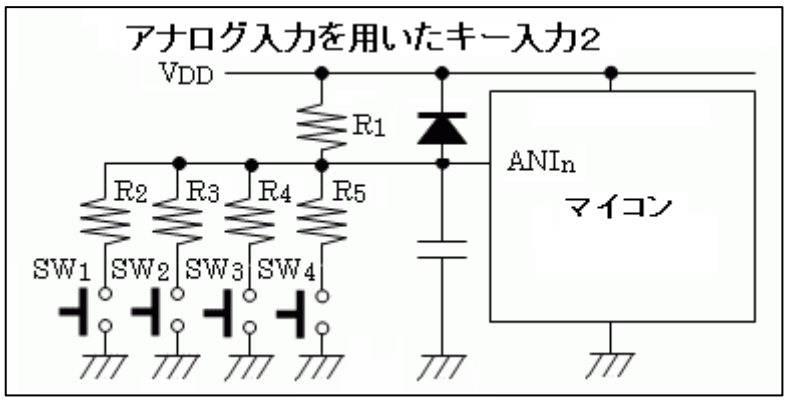
【コーヒー・ブレーク】
抵抗の値は、通常は中途半端です。これは、1～10 までを等比級数的に分割した値となっているためです。分割方法として一般的なのは、12 分割したものが E12 系列、24 分割したものが E24 系列と呼ばれています。E24 系列では **1.0**、1.1、**1.2**、1.3、**1.5**、1.6、**1.8**、2.0、**2.2**、2.4、**2.7**、3.0、**3.3**、3.6、**3.9**、4.3、**4.7**、5.1、**5.6**、6.2、**6.8**、7.5、**8.2**、9.1 の並びです。**太字**で示したものは、E12 系列にも存在する抵抗値です。



以上で抵抗比率が決まり、次に絶対値を決めます。A/D の変換精度をよくするには小さな値、電流を少なくするには大きな値にする必要があります。ここでは、A/D の変換精度はあまり必要ないので、電流を優先します。
R1 を元に求めた結果に、そのままkΩ を付けた場合には、合計で約 1MΩ、となり、待機時（スイッチが押されていない状態）に流れる電流は、5V 動作でも 5μA 程度となります。
以上の例では、間隔を広くとるため、20%刻みの設定ですが、待機時の電流を小さくするには、R5 をできるだけ大きくする（押されていないときの電圧を高くする）ことになります。たとえば、次のような設定を考えます。

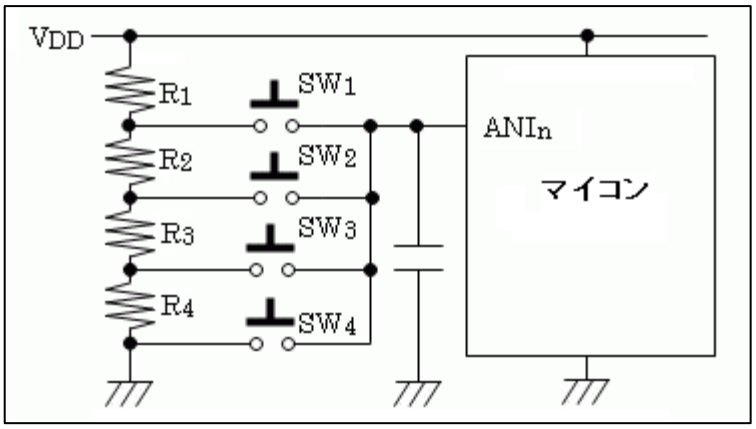
どれも押されていないとき　：$0.95V_{DD}$
SW4 が押されたとき　：$0.3V_{DD}$
SW3 が押されたとき　：$0.2V_{DD}$
SW2 が押されたとき　：$0.1V_{DD}$
SW1 が押されたとき　：0

ここで R1＝100kΩ とすると、R2＝11kΩ、R3＝15kΩ、R4＝16kΩ、R5＝1.8MΩ となります。合計は約 2MΩ で、待機電流を約半分にできます。この制御プログラムは、次々項「A/D コンバータ」をご参照ください。
これまで説明した回路は、キーの優先度が決まっており、二重押しがあっても必ず優先キーの処理ができます（優先度は SW1>SW2>SW3>SW4）。二重押しがない場合、次図のような回路も考えられます。この回路では、スイッチが 1 つしか押されない場合、抵抗の値は独立に設定できるので、計算は楽です。また、スイッチが全く押されていない場合は、電流が流れないので、その点でのメリットもあります。



マイコンでのキー・スイッチ入力(3)

Q1. マイコンでのキー・スイッチ入力(2)では、直列抵抗の各段とグランドの間にスイッチがあります。複数キーを同時に押さなければ、スイッチを各段から A/D 入力に接続した方が、抵抗値の計算が簡単でないですか。



A1. 通常、スイッチは人間が操作するので、人間の体を介したノイズの影響が考えられます。この回路は、計算（設計）は楽ですが、スイッチからノイズがのって、ノイズの影響が大きくなると考えられます。FAQ に掲載している回路では、スイッチの片側をグランドに接続することで、ノイズの影響を低減しています。
二重押し（複数のキー・スイッチを同時に押すこと）がない場合は、マイコンでのキー・スイッチ入力(2)の「**アナログ入力を用いたキー入力 2**」をご検討ください。

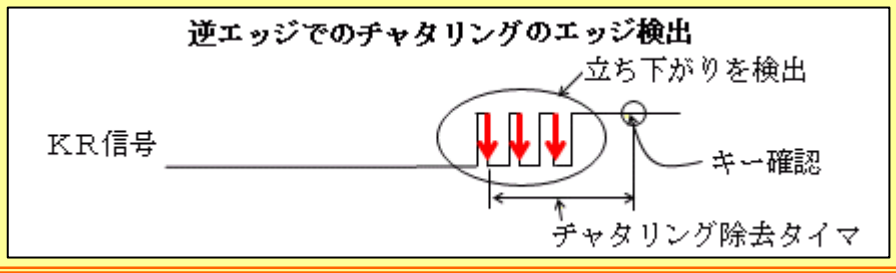
1.1.18 キー割り込みとチャタリング対応

キー割り込みとチャタリング対応

Q1. スイッチの入力をしたいのですが、キー割り込みはハードウエアでチャタリングに対応していますか。

A1. いいえ、ハードウエアでのチャタリング防止機能は内蔵されていません。マイコン内蔵のノイズ除去は、せいぜい数十ns（アナログ遅延を使用）から数 μs（サンプリングによる）程度です。通常、チャタリングのような機械的な振動は数 ms 続きますが、これはマイコンの動作から見ると、非常に長い時間です。このような長時間のノイズを除去ができるようなフィルタは内蔵されていません。チャタリング対策が必要な場合は、キー割り込み処理の冒頭で、ソフトウエア・タイマで、チャタリングが収まるまでの時間待ちをしてから、キーの確認をしてください。キー入力／キー割り込みについては、「マイコンでのキースイッチ入力(1)」もご参照ください。

【ちょっと注意】
キー割り込みでベクタに分岐したとき、CPU は割り込み禁止状態なので、多重割り込みはしません。しかし、チャタリングで割り込み要求フラグがセットされた状態となります。そこで、キー割り込みから戻る前に、割り込み要求フラグをクリアする必要があります。また、次図のように、キーが離されたときもチャタリングが発生します。ソフトウエア・タイマで、チャタリング除去をしていれば、この場合にも対応できます。



1.1.19 A/D コンバータ

A/D コンバータの簡単な使い方

[はじめに]

最近のマイコンは、かなり A/D コンバータを内蔵しています。そのほとんどは、8～10 ビット程度の分解能です。しかし、その分解能をフルに使う必要はありません。実際の応用では、8 ビット以下で十分な場合もあります。

[要求仕様例]

A/D コンバータの比較的簡単な応用に、少ないキーの入力があります。具体的なハードウエア構成として、前述の「マイコンでのキー・スイッチ入力(2)」を例に、押されたスイッチを A/D コンバータで検出します。押されたスイッチと、そのときの A/D コンバータの入力電圧は、次のようになっているものとします。

$0.8V_{DD}$ 以上	: どれも押されていない
$0.6V_{DD}$～$0.8V_{DD}$	: SW4 が押された
$0.4V_{DD}$～$0.6V_{DD}$	: SW3 が押された
$0.2V_{DD}$～$0.4V_{DD}$	: SW2 が押された
$0.2V_{DD}$ 以下	: SW1 が押された

[処理の考え方]

スイッチを押したり離したりする際、入力電圧の変化は緩やかです。付加コンデンサの容量を 1000pF とすると、電圧が安定するまでに数百 μs かかります。この方式では、その間に、入力電圧が他のキー入力に該当する領域を横切って変化します。このとき、誤検出をしないためには、単純に入力電圧だけで判断はできません。そこで、入力電圧が安定していることを判断する処理が必要です。ノイズ対策を兼ねて、16 回の変換分の平均を有効な変換結果として、その変化も監視します。このようにして得られた有効な変換結果から、キー入力を判断します。

【コーヒー・ブレーク】

使用するスイッチには、チャタリングが存在します。その影響を避けるには波形を鈍らせる必要があります。ここでは、電圧設定用の抵抗とアナログ入力のコンデンサによる時定数回路が、この機能を果たします。そのため、この回路の時定数をあまり小さくできません。

ノイズ対策のために A/D 変換結果を複数回平均することでも同様の効果は考えられます。しかし、A/D 変換の前段階でチャタリングの影響をできるだけ少なくしないと、変換結果は大きな誤差が発生する可能性があります。また、他のチャネルでも A/D 変換を行う場合に、その動作に影響を与える可能性があります。このため、できるだけ時定数回路でチャタリングの影響を避けます。



1.1.20 LED のドライブ

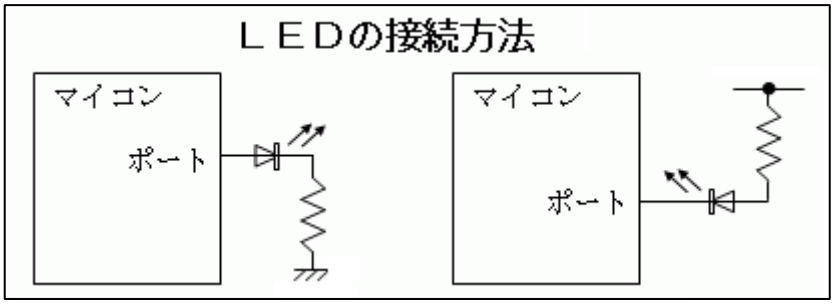
LED のドライブ方法 [78K0/Kx2、78K0S/Kx1+]

[はじめに]

マイコンの最も簡単な出力手段として、LED による表示があります。これは LCD や VFD(蛍光表示管)に比べて、専用の表示コントローラが不要で比較的簡単に使用でき、マイコンの動作電源程度で動作し、小型で小規模なものでも使えることから、幅広く使われています。

[ポートによる基本的なドライブ方法]

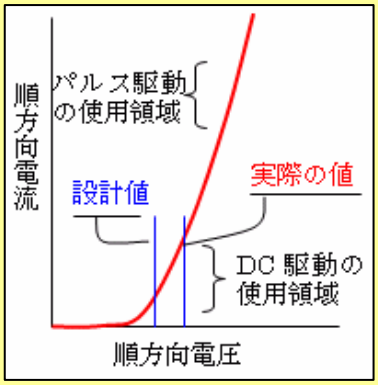
マイコンで LED をドライブする場合、普通はポート機能を使用します。基本的な接続方法としては、次図の左のように、ポートとグランドの間に接続し、ポートから電流を吐き出してドライブする方法と、右に示すように、ポートと正電源の間に接続して、ポートが電流を吸い込んでドライブする方法があります。どちらを使用するかは、ポートのドライブ能力と要求される輝度を得るために必要な電流で判断します。



【コーヒー・ブレーク】

上の接続方法で、LED と直列の抵抗は、ポートを過電流から保護する働きをします。LED は電圧に対して流れる電流が大きく変化し、ある電圧まではほとんど電流が流れず、それ以上になると急激に電流が大きくなります。その傾向を右側のグラフに示します。

マイコンのポートは、基本的に 0/1 のデジタル信号を出力するので、たとえば電流を 5mA 流しても、ポートでの電圧降下は 1V 以下と規定しています。特性がこの半分程度まで変動したとして、0.5V しか電圧が下がらなかった場合は、その分が LED にかかり、電流が約 10 倍になってしまいます。これを避けるために抵抗で電流を制限します。



[プログラムによる制御]

上記の接続でポート 40 を使用して LED を点灯するには、左側の接続例ではポートに 1、右側の接続例ではポートに 0 を出力します。1 個だけの制御を行う場合には、ビット操作命令を使うのが簡単です。

左側の接続例での点灯処理：SET1　　P4.0　（C では、P4.0=1;）

右側の接続例での点灯処理：CLR1　　P4.0　（C では、P4.0=0;）

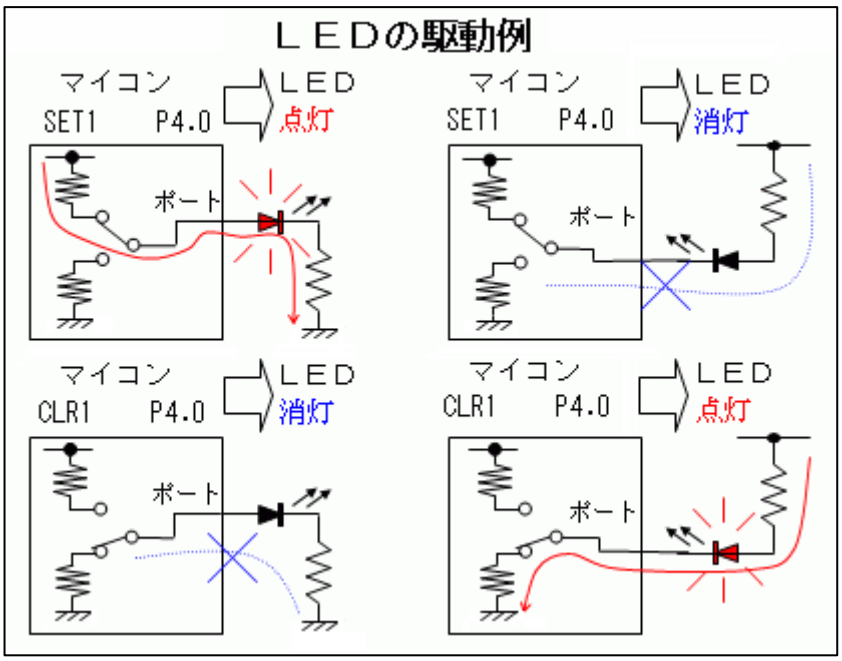
このように、ポートを操作することで、LED を点灯させることができます（次図の「LED の駆動例」参照）。

[はじめに]

マイコンの最も簡単な出力手段として、LED による表示があります。これは LCD や VFD（蛍光表示管）に比べて、専用の表示コントローラが不要で比較的簡単に使用でき、マイコンの動作電源程度で動作し、小型で小規模なものでも使えることから、幅広く使われています。

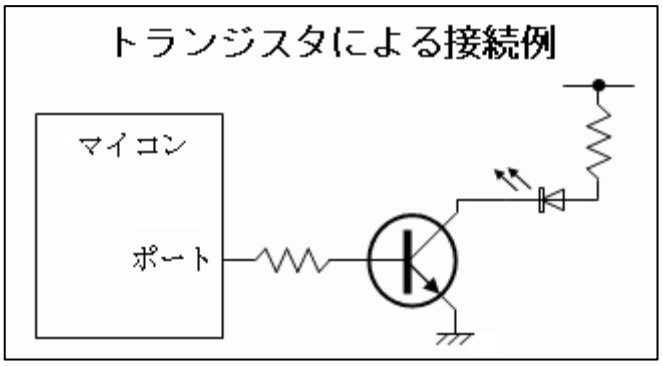
[ポートによる基本的なドライブ方法]

マイコンで LED をドライブする場合、普通はポート機能を使用します。基本的な接続方法としては、次図の左のように、ポートとグランドの間に接続し、ポートから電流を吐き出してドライブする方法と、右に示すように、ポートと正電源の間に接続して、ポートが電流を吸い込んでドライブする方法があります。どちらを使用するかは、ポートのドライブ能力と要求される輝度を得るために必要な電流で判断します。



[ドライブ電流の増大法]

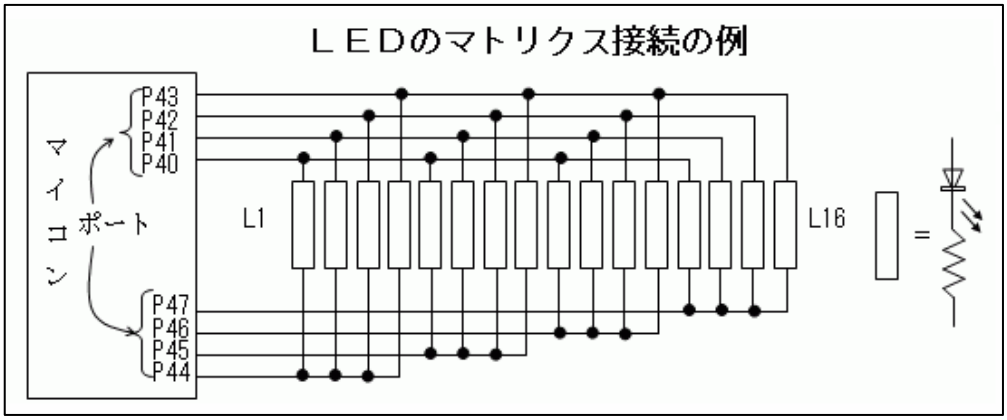
LED の点灯に必要な電流値が大きくて、マイコンで直接ドライブできない場合は、トランジスタを追加します。通常の LED の電流は、大きくても 100mA 以下なので、低周波増幅用の小信号用トランジスタで十分です。



この接続例では、トランジスタで信号が反転されるので、ポート出力を1にすることで、LED を点灯します。

[多くのLEDのドライブ例]

前の例では、1個の LED を点灯するのに、ポートを1つ使用しました。しかし、使用する LED が多くなると、各 LED にポートを割り当てることができなくなります。そこで、マトリクスを組んでダイナミック表示を行う方法が考えられます。LED の電流が 2mA 以下で、同時には 4 個までしか点灯しないとすると、次の構成が考えられます。



【ちょっと注意】
ポートのドライブ能力は、78K0S/Kx1+ではハイ側が 5mA、ロウ側が 10mA と、ロウ側が大きくなっています。78K0/Kx でも値は異なりますが、同じようにロウ側のドライブ能力が大きくなっています。LED に流す電流は弱い方のドライブ能力で決定します。

[プログラム例 1]

この接続では、点灯させたい LED に対して、上側のポートに 1、下側のポートに 0 を設定します。たとえば、L1 の LED を点灯させるには、P40 を 1、P41～P43 を 0 に設定し、P44 を 0、P45～P47 を 1 に設定します。

```
MOV     P4,# 11100001b  (C では  P4=0b11100001;)
```

各 LED に対するイミディエート値(即値)の設定はめんどうなので、通常は点灯させる LED(L1～L16)に対応したテーブルを準備し、その参照で制御します。次のテーブルは、0 なら消灯、1～16 なら対応した LED を点灯します。

```
LEDDATA:
        DB      11110000b       ; 消灯
        DB      11100001b       ; LED1 点灯データ
        DB      11100010b       ; LED2 点灯データ
        DB      11100100b       ; LED3 点灯データ
        DB      11101000b       ; LED4 点灯データ
        DB      11010001b       ; LED5 点灯データ
        DB      11010010b       ; LED6 点灯データ
        DB      11010100b       ; LED7 点灯データ
        DB      11011000b       ; LED8 点灯データ
        DB      10110001b       ; LED9 点灯データ
        DB      10110010b       ; LED10 点灯データ
        DB      10110100b       ; LED11 点灯データ
        DB      10111000b       ; LED12 点灯データ
        DB      01110001b       ; LED13 点灯データ
        DB      01110010b       ; LED14 点灯データ
        DB      01110100b       ; LED15 点灯データ
        DB      01111000b       ; LED16 点灯データ
```

このテーブルを用いて、LED のどれかひとつを点灯させるプログラムの例を次に示します。このプログラムでは、点灯させたい LED を X レジスタで指定します。

```
LEDON:
        PUSH    HL
        MOV     A,#LOW LEDDATA          ; テーブル・アドレスの下位
        ADD     A,X                     ; オフセットを加算
        MOV     L,A
        MOV     A,#HIGH LEDDATA         ; テーブル・アドレスの上位
        ADDC    A,#0                    ; 上位への桁上げ
        MOV     H,A                     ; HL をテーブルの読み出しポインタに設定
        MOV     A,[HL]                  ; データを読み出し
        MOV     P4,A                    ; ポートに出力
        POP     HL
        RET
```

[プログラム例 2]

この処理と同じことをC言語で記述する場合には、LEDの点灯データを配列として準備しておき、点灯させたいLEDを配列の番号として配列から読み出すだけです。以下にそのための記述例を示します。

配列は17個の要素(unsigned char:バイトデータ)から構成されたLEDSTATUSで定義しています。この構造の配列をdispと言う名前で定義しています。

```
typedef unsigned char LEDSTATUS[17];
const LEDSTATUS disp = {
        0b11110000,     // 消灯
        0b01111000,     // LED1 点灯データ
        0b01110100,     // LED2 点灯データ
        0b01110010,     // LED3 点灯データ
        0b01110001,     // LED4 点灯データ
        0b10111000,     // LED5 点灯データ
        0b10110100,     // LED6 点灯データ
        0b10110010,     // LED7 点灯データ
        0b10110001,     // LED8 点灯データ
        0b11011000,     // LED9 点灯データ
        0b11010100,     // LED10 点灯データ
        0b11010010,     // LED11 点灯データ
        0b11010001,     // LED12 点灯データ
        0b11101000,     // LED13 点灯データ
        0b11100100,     // LED14 点灯データ
        0b11100010,     // LED15 点灯データ
        0b11100001,     // LED16 点灯データ
        };
```

この配列を用いて実際のLEDを点灯する関数の例として下記のLEDONを示します。この関数は点灯したいLED(unsigned cha:バイト)を引数としてコールします。いったん、LEDを消灯した後で、配列からデータを読み出して、それをP4に書き込むことで、指定したLEDを点灯させるようになります。

```
void    LEDON(unsigned char LEDNUMBER){
        P4 = 0b11110000;                /* Clear LED             */
        P4 = disp[LEDNUMBER];           /* Turn on specified LED  */
}
```

【解　説】
アセンブラでのテーブル参照は、Cでは配列の参照となります。プログラムそのものより、配列(データ・テーブル)のデータ定義に手間がかかります。また、この配列はROMに入れておくことになるので、配列の宣言の先頭に定数であることを示すconstを宣言しておきます。

[プログラム例の使い方]

プログラム例2は、次のように使用します。タイマH1を用いて4MHz動作の場合、約1秒周期でLED1～LED16を順番に点灯します。

```
#pragma SFR
#pragma NOP
void    main(void){
        unsigned char work1;

        P4 = 0b11110000;                /* Clear All LED          */
        PM4 = 0x00;                     /* Port 4 is Output       */

        TMHMD1 = 0b01000000;            /* set interval timer mode */
        CMP01 = 60;                     /* Interval = 60ms        */
```

```
        IF0 = 0x00;
        work1 = 0x01;
        TMHE1 = 1;
        while(1){
                LEDON(work1++);          /* LED on                  */
                while(TMIFH1 == 0){
                NOP();                   /* wait timer interrupt    */
                }
                TMIFH1 = 0;
                if(work1 == 17) work1 = 1;
        }
}
```

【解　説】

ここでは、LED の切り替えタイミングをタイマで計測しています。タイマは動作モード（インターバル・タイマ）とカウント・クロックをモード・レジスタで設定し、カウントさせる値をコンペア・レジスタ（CMP01）に設定してから起動（TMHE1 に1をセット）しています。これで、タイマは設定時間ごとに割り込み要求フラグ（TMIFH1）をセットします。実際に割り込みを利用して処理を行ってもいいのですが、ここでは while(TMIFH1 == 0)でポーリングをしています。割り込み要求フラグがセットされると、ループを抜けます。
ループを抜けたら、割り込み要求フラグをクリア（TMIFH1 = 0）します。通常のベクタ割り込みでは、割り込み要求を受け付けて、ベクタに分岐するときに、割り込み要求フラグが自動的にクリアされるのですが、ポーリングではプログラムでクリアする必要があります。

ベクタ割り込みや、ここで使ったポーリング以外に、タイムアップを待つ方法があります。それは、省電力機能（スタンバイ機能）を用いて、CPU の実行を停止してしまう方法です。スタンバイ状態で、タイマからの割り込み要求で CPU を再起動して処理を継続します。このためには、スタンバイ状態を解除するための割り込みのマスクを解除しておく必要があります。上記の main 関数を次のように変更します。

```
#pragma NOP
        :
        :
IF0 = 0x00;
        :
        :
while(TMIFH1 == 0){
        NOP();
        }
        :
```

⇒

```
#pragma NOP
#pragma HALT      ← 追加
        :
IF0 = 0x00;
TMMKH1 = 0;       ← 追加
        :
        :
HALT();           ← 変更
        :
```

スタンバイには、HALT 以外に、クロック発振も停止してしまう STOP もあります。しかし、この場合はタイマで時間を計測する必要があるので、STOP は使えません。HALT を使用するには、HALT 命令を使用することを #pragma 指令で宣言する必要があります。そこで、最初に#pragma HALT 宣言を追加します。また、割り込みのマスクを解除するため、TMMKH1 = 0;も追加します。なお、ベクタ割り込みは使用しないので、CPU は割り込み禁止状態にしておきます。
そのうえで、タイマ割り込みを待つために、while 文を HALT()に変更します。

割り込みについては、「割り込みの動作の基本［共通］」もご参照ください。
また、スタンバイについては、「スタンバイ動作」もご参照ください。

1.1.21 LED のダイナミック表示

LED のダイナミック表示方法 [78K0/Kx2、78K0S/Kx1+]

[はじめに]

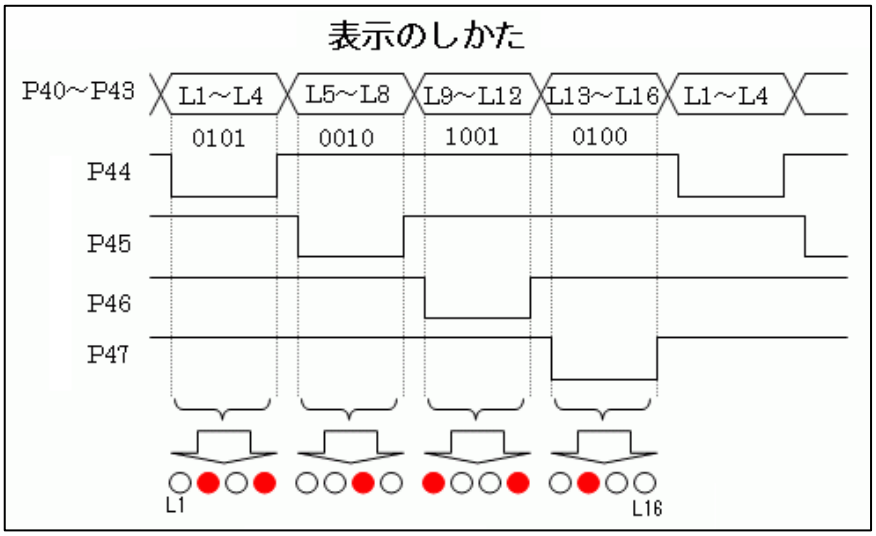
多くの情報を表示する場合、一般にダイナミック表示を行います。「ダイナミック動作とスタティック(Dynamic Operation and Static)」もご参照ください。ここでは、「LED のドライブ方法」の「LED のマトリクス接続の例」に掲載された回路を対象にした、ダイナミック表示方法について説明します。

LED は、次表のポート構成でグループを選択、その中から対象を選択します。

		グループ選択（アクティブ・ロウ信号）			
		P44	P45	P46	P47
		L1～L4	L5～L8	L9～L12	L13～L16
LED選択（アクティブ・ハイ信号）	P40	L1	L5	L9	L13
	P41	L2	L6	L10	L14
	P42	L3	L7	L11	L15
	P43	L4	L8	L12	L16

[表示のしかた]

ここでは、L1～L4、L5～L8、L9～L12、L13～L16 の 4 つの LED で構成された 4 つのグループに分けて表示することにします。P44～P47 で点灯したい LED のグループに対応したポートに 0、それ以外は 1 に設定、表示したい LED を P40～P43 で指定することで表示を行います。次図の表示のしかたに示すように、P44 から順に P45、P46、P47、また P44 と 0 を設定して、ポートからロウ・レベルを出力します。これに合わせて、P40～P43 に対応した LED の点灯パターンを出力します。



表示は 2.5ms で切り替え、10ms（100Hz）ですべての LED の点灯を完了するものとします。このために、システム・クロックとして高速内蔵発振器の 8MHz を 1/2 分周した 4MHz を使用します。これを 1/64 分周したクロックで 8 ビット・タイマを駆動し、1/156 分周することで約 400Hz のインターバル割り込みを発生させます。このインターバル割り込みの中で、表示を切り替えます。

[プログラム例 1]

ここでは、アセンブリ言語で記述したプログラム例を示します。C 言語記述は、プログラム例 2 で説明します。

表示の指定は、点灯する LED に対応した 2 つ（LEDSTATUS1、LEDSTATUS2）の 8 ビットの変数（LED ステータス）を準備して、そこに表示データを設定しておきます。タイマ割り込み処理では、点灯したい LED に対応したビットから 4 個の LED に対応したデータを抽出します。そのデータと表示グループの指定データを組み合わせて、ポート 4 に出力することで LED を点灯します。これを 4 回繰り返すことで、完全な表示となります。

(1) 表示データの作成

LED ステータスの情報を 4 つのグループに分けて点灯するため、何回目の表示かの情報を元に、LED ステータスから表示データ・パターンを抽出します。ここでは、どのグループかの情報を X レジスタ、どの LED かの情報を A レジスタで作成し、2 つを合成して表示データを作ります。処理時間を短くするために、グループを示す情報と、その中の点灯する LED の情報を並行に処理して、最後に結合します。次のプログラム例では、おもに X レジスタでグループの選択情報、A レジスタでグループ中の点灯する LED の情報を処理しています。

```
DATWORK         DSEG    SADDR
NEXTDATA:       DS      1               ; 次の表示データ格納用
LEDSTATUS1:     DS      1               ; LED1～8 の点灯情報（1 で点灯）
LEDSTATUS2:     DS      1               ; LED9～16 の点灯情報（1 で点灯）
COUNT:          DS      1               ; 何回目の表示か（0～3）
GETLED:
                MOV     A,LEDSTATUS1    ; LED1～8 のステータスを読み出し
                MOV     X,#11101111b    ; LED1～4 の選択信号
                BF      COUNT.1,$NEXT1  ; LED1～8 の表示なら次へ
                MOV     A,LEDSTATUS2    ; そうでなければ LED9～16 を
                MOV     X,#10111111b    ; LED9～12 の選択信号
NEXT1:
                BF      COUNT.0,$NEXT2  ; LED1～4/9～12 なら分岐
                ROR     A,1             ; 有効 LED ビットを下位 4 ビットに
                ROR     A,1
                ROR     A,1
                ROR     A,1
                XCH     A,X             ; グループ選択信号を A レジスタへ
                ROL     A,1             ; 選択信号を次のグループに
                XCH     A,X             ; A:LED ステータス、X:グループ情報
NEXT2:
                OR      A,#11110000b    ; グループ情報をマスク
                AND     A,X             ; 表示データを合成
                MOV     NEXTDATA,A      ; 次の表示データを保存
                RET
```

(2) データの表示（タイマ割り込み処理）

2.5ms ごとの割り込みで、LED をグループごとに点灯させます。表示するためのデータ生成は、前記のサブルーチンで行っています。ここでは、まず表示を消したあとで、前回計算しておいたデータをポートに出力することで、表示を行います。その後、次のグループを選択し、その表示データを準備します。
前回計算しておいたデータを使用するのは、できるだけ時間のばらつきの影響をなくすためです。

```
TMH1INT:
                PUSH    AX
                MOV     A,NEXTDATA      ; 表示データ情報を読み出し
                MOV     P4,#11110000b   ; LED をすべて消灯
                MOV     P4,A            ; 新しいデータを表示
```

```
                INC     COUNT           ; 次のグループを選択
                MOV     A,#00000011b    ; 表示グループ数のマスク
                AND     A,COUNT         ; すべての桁が終了していれば
                MOV     COUNT,A         ; 次のグループをセット
                CALL    !GETLED         ; 次の表示データを作成
                        :
        〈ここに、他に必要な処理を記述する〉
                        :
                POP     AX
DUMMY:          RETI                    ; 割り込み処理終了
```

(3) 初期化処理

初期化処理では、スタック・ポインタの設定、電源の立ち上がり待ち、メモリや使用する内蔵周辺のハードウエアを初期化します。その後、タイマを起動し、割り込みのマスクを解除してから、他の処理を行います。タイマ割り込みが発生すると、LED 表示を開始します。

なお、リセット・ベクタや割り込みベクタは、次のようにしておきます。ここで、使用しない割り込みには、何もしないで RETI 命令で戻るためのダミーのベクタを設定しておきます。

```
RSTVECT CSEG    AT      0
                DW      START           ;00:リセット・スタート処理のベクタ
                DW      DUMMY           ;02:
                DW      DUMMY           ;04:
                DW      DUMMY           ;06:INTLVI
                DW      DUMMY           ;08:INTP0
                DW      DUMMY           ;0A:INTP1
                DW      TMH1INT         ;0C:INTTMH1
                DW      DUMMY           ;0E:INTTM000
                DW      DUMMY           ;10:INTTM010
                DW      DUMMY           ;12:INTAD
                CSEG
START:          MOVW    AX,#STACKP
                MOVW    SP,AX
                CALL    !POWERON        ; 電源の立ち上がりを待つ
                CALL    !INITIO         ; ハードの初期化
                CALL    !INITRAM        ; RAM の初期化
                        :
                SET1    TMHE1           ; タイマをスタート
                CLR1    TMMKH1          ; タイマ割り込みマスク解除
                EI
                        :
        〈ここに、他に必要な処理を記述する〉
                        :
;
;       電源の立ち上がり待ち（2.7V 以上になるのを待つ）
;
POWERON:
                DI
                MOV     LVIS,#00000111b ; VLI=2.7V
                MOV     LVIM,#10000000b ; 低電圧検出開始
                MOV     PCC,#00         ; CPU clock is fx/4
                MOV     WDTM,#01110000b ; WDT を停止
```

```
                SET1    LSRSTOP         ; 低速発振器停止
                MOV     A,#50
PONLOOP:
                DEC     A
                BNZ     $PONLOOP        ; 0.2ms 待つ
PONLOOP2:
                BT      LVIF,$PONLOOP2  ; 電圧が 2.7V 以上になるまで待つ
                MOV     PPCC,#01        ; CPU clock is fx/2
                RET
;
;       RAM の初期化
;
INITRAM:
                MOV     COUNT,#3
                MOV     LEDSTATUS1,#0
                MOV     LEDSTATUS2,#0
                MOV     NEXTDATA,#11110000b
                RET
;
;       タイマ H1 やポート 4 などハードウエアの初期設定
;
INITIO:
                MOV     P4,#11110000b           ; 初期値（LED 全消灯）
                MOV     PM4,#00                 ; ポート 4 はすべて出力に
                MOV     TMHMD1,#00110000b       ; タイマ・モード設定
;                               |||++-----   インターバル・タイマ・モード
;                               +++-------   カウント・クロック fxp/64
                MOV     CMP01,#155              ; インターバル時間を設定
                MOV     IF0,#00                 ; 割り込みクリア
                        :
        <ここに、他で必要な初期化処理を記述する>
                        :
                RET
```

[プログラム例 2]

ここでは、C 言語で記述した LED 表示プログラム例について説明します。この例では、表示パターンを約 1 秒ごとに＋1 するような処理も追加しています。

(1) 宣言部分

このプログラムでは、内蔵周辺 I/O の操作以外に、割り込み制御を行うので、プログラムの先頭で#pragma 指令の宣言を行います。割り込み処理を行う関数としては、GETLED をプロトタイプ宣言しています。また、ハードウエアの初期化を行うための hdwinit もプロトタイプ宣言しています。

変数 LEDSTATUS は点灯するLEDの情報で、ここでは 16 ビットの変数として処理しています。SECTIMER は、1 秒の時間をカウントするのに使用します。NEXTDATA や COUNT は、アセンブラ記述のプログラムと同じ意味で使用しています。

```
#pragma SFR
#pragma EI
#pragma DI
```

```
#pragma NOP
#pragma INTERRUPT INTTMH1 GETLED
unsigned int LEDSTATUS,SECTIMER;
unsigned char NEXTDATA,COUNT;

__interrupt void        GETLED(void);
void    hdwinit(void);
```

【コーヒー・ブレーク】
78K シリーズ用のC言語では、標準的な C 言語に対して、組み込み用途で使用するための機能が拡張されています。この拡張機能を使用するには、宣言が必要となります。おもな宣言としては、ここで使用している内蔵周辺 I/O をキーワードとして使用するための「#pragma　SFR」、CPU のいくつかの命令をC言語で使用するための「#pragma EI」や「#pragma NOP」などの指定、CPU のベクタ割り込み（ここでは INTTMH1）と実際の処理関数（ここでは GETLED）を結びつける「#pragma INTERRUPT INTTMH1 GETLED」があります。



(2) 初期化処理部

78K のC言語では、プログラム本体の実行開始前に、内蔵周辺 I/O の初期化処理ができます。そのための専用の関数（hdwinit）が予約されています。早く実行させる内容は、hdwinit 関数に記述します。次に示す hdwinit 関数では、ウォッチドッグ・タイマの停止後、低電圧検出機能で、電源電圧の立ち上がりを待っています。低電圧検出機能は、起動後 0.2ms 待たないと正常な結果を得ないので、ソフトウエア・タイマで待ちます。電源電圧が 2.7V 以上になったら、CPU の動作クロックを 4MHz に設定して、使用するポートやタイマなどの初期化を行っています。

```
void    hdwinit(void){
        unsigned char work1;

        LVIS = 0b00000111;              /* select VLI=2.7V          */
        LVIM = 0b10000000;              /* Start VLI detection      */
        PCC = 0x00;                     /* CPU clock is fx/4        */
        WDTM = 0b01110000;              /* Stop WDT                 */
        LSRSTOP = 1;                    /* Stop Low speed OSC       */
/*
        wait for 0.2m秒
*/
        for (work1 = 0; work1 < 10; work1++){
                NOP();
        }

        while ( LVIF ){
                NOP();
        }                               /* wait VDD > 2.7V          */

        PPCC = 0b00000001;              /* CPU clock is fx/2        */
        P4 = 0b11110000;                /* Clear All LED            */
         PM4 = 0x00;                     /* Port 4 is Output         */

        TMHMD1 = 0b00110000;            /* set interval timer mode  */
        CMP01 = 155;                    /* Interval = 2.5ms         */
        IF0 = 0x00;
}
```

(3) 本体処理部

main 関数では、各種の変数に初期値を設定します。その後は、タイマを起動して、1 秒の時間をカウントする変数（SECTIMER）が 0 になるのを待ちます。1 秒経過後、表示データ（LEDSTATUS）を＋1 します。変数 SECTIMER は main 関数と割り込み関数の両方から書き込まれるので、本来は排他制御が考えられます。ここでは、割り込み間隔（2.5ms）以内に処理が完了してしまうので、排他制御（この場合は割り込みの禁止／許可）は行いません。

```
void    main(void){
        COUNT = 3;
        NEXTDATA = 0b1111000;
        LEDSTATUS = 1;
        SECTIMER = 400;
        TMHE1 = 1;
        TMMKH1 = 0;
        EI();
        while(1){
                while(SECTIMER){
                NOP();
                }
                SECTIMER = 400;
            LEDSTATUS++;
        }
}
```

(4) 割り込み処理部

タイマ H1 による 2.5ms ごとの割り込みで起動、実行されます。そのため、関数の先頭で割り込み処理関数を示す「_interrupt」が宣言されています。ほかは、通常の引数や戻り値のない関数と同じです。

処理は、ほとんど表示のためポートに出力するデータ・パターンの生成で、次の 2 つの処理で構成されます。

・表示グループの選択信号の生成（switch 文で、変数 COUNT に応じた値を NEXTDADA に設定）

・表示データの抽出（表示したいデータ（LEDSTATUS）を必要回数だけ右シフトし、下位 4 ビット以外をマスク）

これら 2 つのデータを結合して、表示のデータ・パターンとします。他の処理は、ポート出力（前回作成したデータ・パターン）や、次回のための表示グループの更新、1 秒を得るためのカウントです。

```
__interrupt     void    GETLED(void){
        unsigned int work1;
        unsigned char work2;
        P4 = 0b11110000;                /* Clear LED                */
        P4 = NEXTDATA;                  /* Turn on specified LED    */
        work1 = COUNT;                  /* Get Display Group Number */
        switch(work1){
                case 0:
                                NEXTDATA = 0b11100000;
                                break;
                case 1:
                                NEXTDATA = 0b11010000;
                                break;
                case 2:
                                NEXTDATA = 0b10110000;
                                break;
                case 3:
                                NEXTDATA = 0b01110000;
        }
```

```
        work1 *= 4;
        work2 = LEDSTATUS >> work1;
        work2 &= 0b00001111;
        work1 = (unsigned char) work2;
        NEXTDATA |= work1;
        COUNT = (COUNT+1) & 0b00000011;
        SECTIMER--;
    }
```

1.1.22 EEPROM の制御

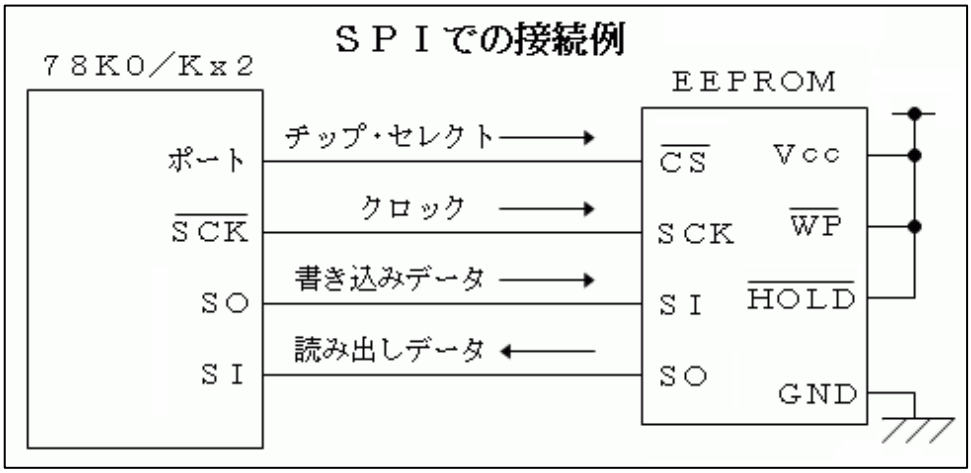
CSI による SPI メモリの制御 [78K0/Kx2 など]

[はじめに]

SPI は、モトローラ社（現フリースケール・セミコンダクタ社）が提唱した、マスタ・スレーブ方式のインタフェースで、基本的なデータの転送は 2 本のデータ（入力と出力）とクロックの 3 本の信号で行うクロック同期式シリアル通信方式です。データは 8 ビットの固定長でやりとりされます。基本的に、この部分は 78K0 シリーズなどに搭載されたシリアル・インタフェース(CSI)と互換性があります。SPI では、これ以外にチップ・セレクトでデバイスを選択する機能がサポートされています。

[EEPROM との接続]

SPI インタフェースをもつ EEPROM との接続例は、次のようになります。



EEPROMに対するチップ・セレクト信号は、ポートを用いて出力します。その他の信号は、3線式シリアルの信号をそのまま使用します。なお、EEPROM には、これ以外にステータス・レジスタへの書き込みを禁止するためのライト・プロテクト(WP)信号や、デバイスの選択中に動作を停止させるための、ホールド(HOLD)信号があるものもあります。ここでは、これらの信号は使用しないので、電源に接続しておきます。

[ステータス・レジスタ]

EEPROM は、デバイスの状態を示すためのステータス・レジスタを内蔵しています。ビットの意味は、次のとおりです。詳細は、実際に使用される EEPROM のマニュアルなどをご参照ください。

ビット 0　:書き込み中かどうかを示す。1 なら書き込み中で、新たなアクセスはできない。0 ならアクセス可能。
ビット 1　:書き込み許可ビット。後述のコマンドで操作し、1 なら書き込み可能、0 なら書き込み禁止。
ビット 2、3:ブロックごとの書き込み禁止ビット。個々のデバイスで異なる。00 なら、すべてのデータ領域に書き込み可能(デバイスによっては、3 ビット以上のものもあるようです)。
ビット 7　:ステータス・レジスタへの書き込み禁止ビット。このビットが 1 で、WP 端子がロウ・レベルなら、ステータス・レジスタへの書き込みは禁止される(デバイスによってはこの機能がないものもあるようです)。

[EEPROM のコマンド]

EEPROM にはいくつかのコマンドが定義されており、最初にこれを用いて動作を指定する必要があります。コマンドの詳細は、個々のデバイスのマニュアルをご参照いただくことにして、ここでは簡単に説明します。

コード	名　称	動　　作
0000 0110	WREN	書き込み許可ビットをセットして、書き込み可能にする
0000 0100	WRDI	書き込み許可ビットをクリアして、書き込み禁止状態にする
0000 0101	RDSR	ステータス・レジスタの読み出し
0000 0001	WRSR	ステータス・レジスタへの書き込み
0000 x011	READ	データの読み出し
0000 x010	WRITE	データの書き込み

WREN　:電源投入時には書き込み禁止状態なので、書き込みを行うにはこのコマンドを発行します。
WRDI　:このコマンドを発行すると、データとステータス･レジスタへの書き込みができなくなります。
RDSR　:コマンド発行の次のクロックから、ステータス・レジスタの内容を MSB から順に出力します。
WRSR　:ステータス・レジスタへの書き込みを行います。コマンド発行の次のクロックからステータス・レジスタへ書き込むデータを MSB から順に入力します。
READ　:EEPROM からデータを読み出します。コマンドに続いて、1 バイトか、メモリの容量によっては 2 バイトのアドレスを転送する必要があります。その後の転送クロックで、指定されたアドレスの内容を出力します。CS をハイ・レベルにすれば、読み出しは完了となります。コマンドのビット 3 は、アドレスが 9 ビット必要な場合のアドレスとして使用できます。
WRITE　:EEPROM へデータを書き込みます。コマンドに続いて、1 バイトか、メモリの容量によっては 2 バイトのアドレスを転送する必要があります。その後の転送クロックで、指定されたアドレスへの書き込みデータを転送します。転送が完了して CS をハイ・レベルにすることで、実際の書き込みを行います。書き込み中は、ステータス・レジスタのビット 0 が 1 となります。このビットが 0 になるまでは、ステータス・レジスタのリード以外のコマンドは発行しない方がいいでしょう。

[EEPROM の制御方法]

EEPROM へのアクセスは、すべてコマンドから始まります。

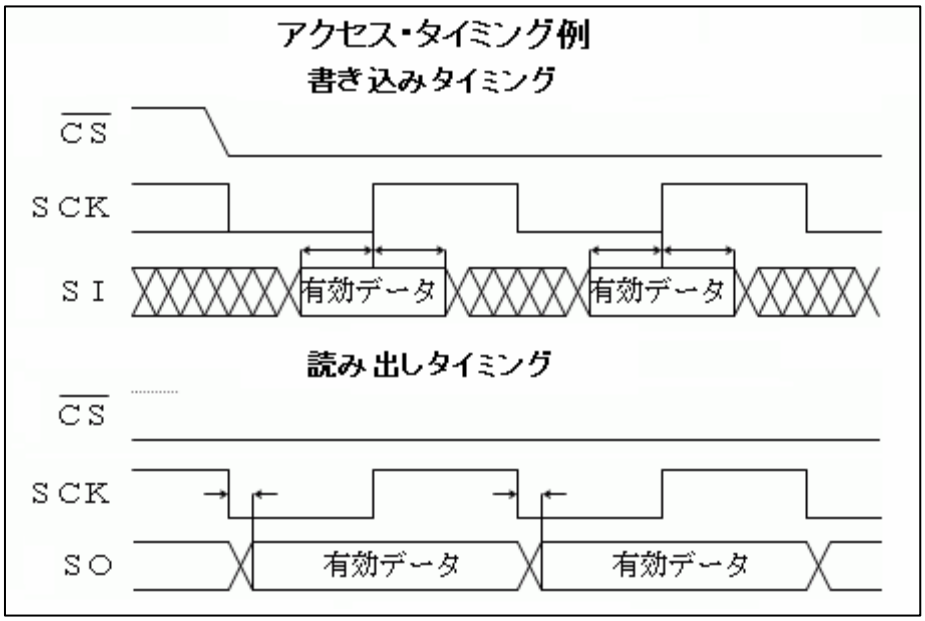

WRENコマンド
スタート
CSをロウ・レベルに
0000 0110 を送信
CSをハイ・レベルに
エンド
WRDIコマンド
スタート
CSをロウ・レベルに
0000 0100 を送信
CSをハイ・レベルに
エンド
RDSRコマンド
スタート
CSをロウ・レベルに
0000 0101 を送信
ステータスを受信
CSをハイ・レベルに
エンド
WRSRコマンド
スタート
WRENコマンド注1
CSをロウ・レベルに
0000 0001 を送信
ステータスを受信
CSをハイ・レベルに
RDSRコマンド
ビット0=0？
No
Yes
エンド
READコマンド
スタート
CSをロウ・レベルに
0000 x011 を送信注2
アドレスを送信注3
データを受信
完了？
No
Yes
CSをハイ・レベルに
エンド
WRITEコマンド
スタート
WRENコマンド注1
CSをロウ・レベルに
0000 x010 を送信注2
アドレスを送信注3
データを送信
完了？
No
Yes
CSをハイ・レベルに
RDSRコマンド
ビット0＝0？
No
Yes
エンド
注1. 電源投入後やWRDIコマンド実行後には必要。
2. 512バイトのメモリではビット3のxにA8が入る。
3. 1Kバイト以上ではアドレスは2バイトとなるので、上位→下位の順で指定する。

チップ･セレクト信号（CS）をロウ･レベルにすれば、EEPROM が選択され、最初に転送される内容がコマンドとなります。上記の 6 つのコマンドを用いて、EEPROM へのアクセス制御を行います。書き込みの場合は、アクセス制御が完了したら、CS をハイ･レベルにすれば、EEPROM はコマンドで指定された書き込みを実行します。ステータス･レジスタはいつでも読み出せ、これによって、書き込みの完了など、EEPROM の状態を確認できます。

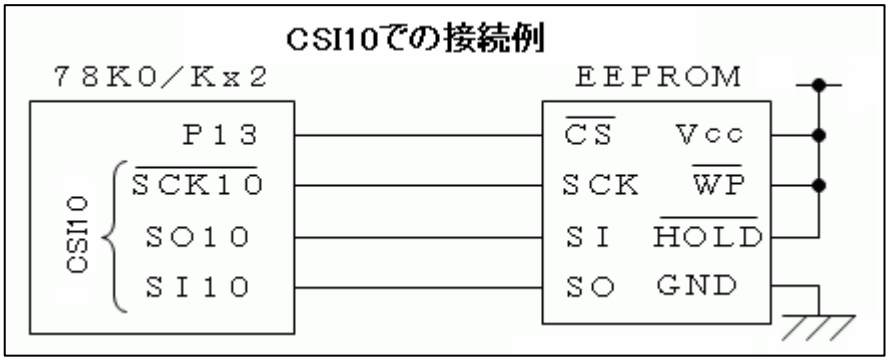
[78K0/Kx2 での制御モード]

EEPROM とのシリアル通信は、2.7V 以上の電源電圧であれば、シリアル･クロックが 5MHz までになります。そこで、78K0/Kx2 を高速内蔵発振器の 8MHz で動作させ、CSI10 のクロックにその半分の 4MHz を使用することにします。

SPI では、データとクロック(SCK)の位相の関係で 4 つのモードがありますが、モード 0 と 3 の 2 つのモードに対応した EEPROM を対象とします。この 2 つのモードはともに、SCK の立ち下りでデータを出力し、SCK の立ち上がりでデータを入力するものです。EEPROM のアクセス･タイミング例を次に示します。



78K0/Kx2 の CSI の転送タイプ 1 を使用します。これは、従来の 3 線式シリアル(SIO)と同じです。この場合、CSICI10 に 00（初期値と同じ）を設定して使用します。CSIM10 は、MSB ファーストの送受信モードに設定します。



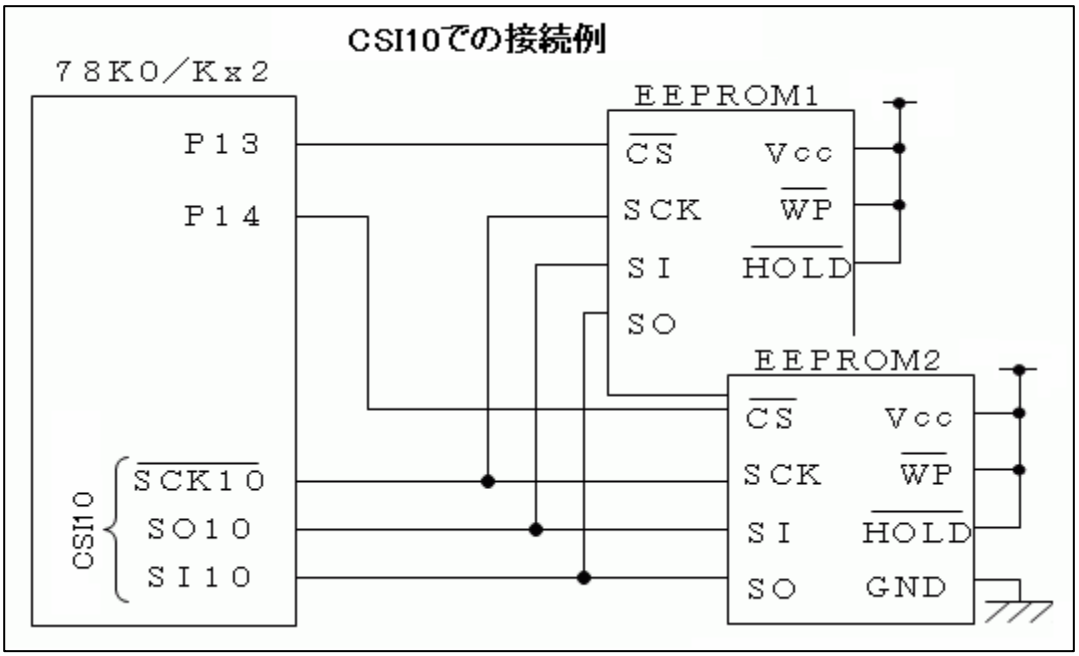
CSI10 での接続例を示します。CSI10 では P10～12 の端子を使用するので、CS 信号用に連続した P13 を使用します。ここでは EEPROM を 1 個接続していますが、2 個目の接続では P14、3 個目の接続では P15 をそれぞれ CS 信号に使用してください。

具体的な制御プログラムについては「CSI による SPI メモリと EEPROM の制御プログラム例 [78K0/Kx2]」をご参照ください。

CSI による SPI メモリと EEPROM の制御プログラム例 [78K0/Kx2]

[はじめに]

SPI の概要や SPI インタフェースの EEPROM については「CSI による SPI メモリの制御 [78K0/Kx2 など]」をご参照ください。ここでは、次の接続例に示す EEPROM を 2 個接続した場合に対応する、具体的な制御プログラムについて説明します。また、制御プログラムの使い方についても簡単に触れておきます。



[78K0/Kx2 の処理例]

78K0/Kx2 での EEPROM の制御処理は、以下の部分で構成します。内部で使用するサブルーチン以外は、C 言語からの使用を意識して、名前と入出力パラメータ(引数と戻り値)を決めています。たとえば引数として、読み出しや書き込みでは、次のパラメータを用います。

第 1 パラメータ(AX 渡し)　:EEPROM のアドレス
第 2 パラメータ(スタック渡し):バッファ・メモリのアドレス
第 3 パラメータ(スタック渡し):転送データ数で 1～255 と 0(256 バイト)

(1) 初期化部

ポート、CSI10、割り込みの初期化を行います。ここでは 2.7VPOC を使用することで、リセット解除後にはそのまま動作させます。基本的にベクタ割り込みは使用しないので、割り込みは禁止状態とします。

(2) EEPROM 選択処理

P13～17 を CS に使用することで、最大 5 つの中から一つを選択します。この処理では、単に作業領域に対象とする EEPROM の番号(X レジスタ渡しで、1～5 の値)から CS 信号に対応したビット・パターンをセットするだけで、その値を CS 信号の制御で用いるだけです。

(3) CS 信号出力処理

EEPROM への実際のアクセス時に、CS 信号を ON(ロウ・レベル)にしたり、OFF(ハイ・レベル)にしたりするための内部処理ルーチンです。P1 の内容を読み出し、CS 信号に該当するビットを反転させ、P1 に書き戻しています。

(4) ステータスの確認

EEPROM がデータ書き込み中かどうかをチェックします。内部で EEPROM への書き込みやデータの読み出しを行う前に使用します。

(5) 書き込み許可

対象の EEPROM の CS 信号を ON にして、「WREN」コマンドを送ります。コマンド転送完了後、CS 信号を OFF にします。このコマンドにより、EEPROM は書き込み可能状態になります。

(6) 書き込み禁止

対象の EEPROM の CS 信号を ON にして、「WRDI」コマンドを送ります。コマンド転送完了後、CS 信号を OFF にします。このコマンドにより、EEPROM は書き込み禁止状態になります。

(7) ステータス・レジスタの読み出し

対象の EEPROM の CS 信号を ON にして、「RDSR」コマンドを送り、ステータス・レジスタの状態を 1 バイトのデータで読み出します。読み出し完了後、CS 信号を OFF にします。ステータス・レジスタのビットの意味は「ステータス・レジスタ」をご参照ください。

(8) ステータス・レジスタへの書き込み

対象の EEPROM の CS 信号を ON にして、「WRSR」コマンドを送り、引き続いてステータス・レジスタにセットする、1 バイトのデータを書き込みます。書き込みデータの転送完了後 CS 信号を OFF にします。
通常は、ブロックごとのプロテクトの制御（ビット 2、3）に使用します。

(9) データの読み出し 1

512 バイト以下（アドレスは 1 バイトで、A8 はコマンドで指定）の容量の EEPROM から、256 バイト以下のデータを読み出します。EEPROM のステータスを確認し、書き込み中でなければ、指定されたパラメータでの読み出しをします。

(10) データの読み出し 2

1K バイト以上（ただしアドレスは 2 バイト）の容量の EEPROM から、256 バイト以下のデータを読み出します。EEPROM のステータスを確認し、書き込み中でなければ、指定されたパラメータでの読み出しをします。

(11) データの書き込み 1

512 バイト以下（アドレスは 1 バイトで、A8 はコマンドで指定）の容量の EEPROM に、256 バイト以下のデータを書き込みます。EEPROM のステータスを確認し、書き込み中でなければ、書き込み許可コマンドを発行し、指定されたパラメータで書き込みをします。
EEPROM へのデータ転送しかしないので、実際に EEPROM で書き込みが完了するまでは、並行して別の処理ができます。この間、EEPROM へのアクセスは、ステータス・レジスタの読み出し以外はしないでください。

(12) データの書き込み 2

1K バイト以上（ただしアドレスは 2 バイト）の容量の EEPROM に、256 バイト以下のデータを書き込みます。EEPROM のステータスを確認し、書き込み中でなければ、書き込み許可コマンドを発行し、指定されたパラメータ

での書き込みをします。アドレス長が異なる以外は「**データ書き込み 1**」と同じです。

(13) 書き込み完了待ち

データ書き込み後に、実際にEEPROMでの書き込み完了を待ちます。データ書き込み後は、この処理を実行するか、ステータス・レジスタを読み出して、ビット 0 が 0(書き込みが完了)になったことを確認してください。

(14) バイト転送完了待ち

1 バイトのデータ転送が完了するまで、割り込み要求フラグをポーリングすることで待ち合わせます。転送が完了して、割り込み要求フラグがセットされたことを確認したら、フラグをクリアして戻ります。これは、内部処理だけで使用します。

[アセンブラでのプログラム例]

このプログラム例では、他のプログラムからこのプログラムの処理を使えるように、以下の 10 のサブルーチンをパブリック宣言しています。これらは、アセンブラから使う場合や、C 言語のプログラムから使う場合も考慮しています。アセンブラから使用する場合は、大文字の名前を EXTRN 宣言してください。C 言語のプログラムから使う場合は、_で始まる名前を extern 宣言(たとえば、extern _selprom のように、_を 1 つにして宣言)してください。

```
        PUBLIC   __selprom,SELPROM          ; アクセスする EEPROM を選択
        PUBLIC   __writeenb,WRITEENB        ; 書き込み許可
        PUBLIC   __writedis,WRITEDIS        ; 書き込み禁止
        PUBLIC   __readst,READST            ; ステータス・レジスタのリード
        PUBLIC   __writest,WRITEST ; ステータス・レジスタへのライト
;
        PUBLIC   __read_spi_block1,READBLOCK1
                 ; 512 バイト以下の EEPROM からのデータ・リード
        PUBLIC   __read_spi_block2,READBLOCK2
                 ; 1K バイト以上の EEPROM からのデータ・リード
        PUBLIC   __write_spi_block1,WRITEBLOCK1
                 ; 512 バイト以下の EEPROM へのデータ・ライト
        PUBLIC   __write_spi_block2,WRITEBLOCK2
                 ; 1K バイト以上の EEPROM へのデータ・ライト
;
        PUBLIC   __waitwt,WAITWT            ; 書き込み完了待ち
```

また、内部で使用する作業領域の宣言や、値の定義を前もって行っています。

```
;
;       使用 EEPROM 数の定義
;
;       EEPROM は 5 つを最大として、P13 から P17 を順に CS 信号に
;       割り当てるものとする
;       ポートの有効利用のために使用する最大数を定義する
;       接続する EEPROM の個数に合わせて ROMNO の値を変更する
;
ROMNO   EQU      2                  ; ここでは 2 つとしておく
```

```
;
;       EEPROM のコマンド
;
WRENC  EQU      00000110b          ; 書き込み許可
WRDIC  EQU      00000100b          ; 書き込み禁止
RDSRC  EQU      00000101b          ; ステータス・レジスタのリード
WRSRC  EQU      00000001b          ; ステータス・レジスタへの書き込み
READC  EQU      00000011b          ; データ読み出し
WRITEC EQU      00000010b          ; データ書き込み

WDATA  DSEG     SADDR
CSFLAG:
       DS       1                  ; CS 信号用データ
RxCOUNT:
       DS       1                  ; 受信データ用カウンタ
TxCOUNT:
       DS       1                  ; 送信データ用カウンタ
```

(1) 初期化部

ポートや CSI10 の動作モードを設定します。ここではベクタ割り込みを使用しないので、割り込みはマスクしておきます。

```
;
;       ポートや CSI10 のイニシャライズ
;
        MOV     P1,#11111011b              ; 出力ラッチの設定
        MOV     PM1,#00000010b             ; SI10/P11 以外は出力に
        MOV     CSIC10,#00000000b          ; 初期値と同じ
;                       |||||+++--------: シリアル・クロックは fPRS/2
;                       |||++------------: タイプ 1
;                       +++--------------: 0 に固定
        MOV     CSIM10,#01000000b          ; MSB ファースト、送受信モード
;                       ||+|++++--------; 0 に固定
;                       || +------------; MSB ファースト
;                       |+--------------; 送受信モード
;                       +---------------; 動作禁止
;
        MOV     PCC,#0
        SET1    CSIMK10                    ; CSI10 の割り込みマスク
        CLR1    CSIIF10                    ; 割り込み要求フラグのクリア
        SET1    CSIE10                     ; CSI10 の動作許可
```

(2) EEPROM 選択処理

X レジスタで指定(1～ROMNO)された EEPROM に対応した CS 信号のビット位置パターンを作成します。接続している EEPROM の数を ROMNO で定義しておけば、それを越えて指定された場合にエラーを戻します(ただし、0 はチェックしていません)。

最初にこのルーチンで EEPROM を選択して、以降の処理で EEPROM の番号指定を省略します。EEPROM が 1 つだけの場合、この処理はなくてもかまいません。

以降のプログラム例では、外部から使用するサブルーチンはアセンブラでの使用（内部使用）に加えて、C 言語から使用するために、名前を 2 つ定義しています。

```
;
;       EEPROM の選択
;   <機能>
;   引数で指定された EEPROM をアクセス対象に設定する
;     ここでは CS 用のビット・イメージを抽出して作業領域に保存します
;     実際の CS 信号出力は CSOUT サブルーチンで行います
;   <入力パラメータ（引数）>
;   X    :選択する ROM の番号（1～ROMNO）を引数とする
;   <戻り値>
;   CF   :エラーならキャリー・フラグをセットして戻る
;bit   _ selprom(unsigned char spi_adr)

__selprom:
SELPROM:
        MOV     A,#ROMNO            ; 選択する番号を最大数と比較して、
        CMP     A,X                 ; 大きければキャリー・フラグをセットして
        BC      $EXITS              ; 処理を終了
        MOV     A,#00000100B        ; CS 信号ビットへの変換
SELLOOP:
        ROL     A,1                 ; CS 信号を上位ビットへシフト
        DEC     X                   ; ROM 番号を－1
        BNZ     $SELLOOP            ; 目的のビット位置まで繰り返す
        MOV     CSFLAG,A            ; CS フラグに結果を格納
EXITS:
        RET
```

(3) CS信号出力処理

EEPROM への CS 信号の ON/OFF 処理を行うプログラムです。ここで、上記の EEPROM 選択処理結果のビット・パターンを使用して P1 を制御し、実際に CS 信号を制御します。実際の処理は P1 のビットの反転処理ですので、必ず初期状態が決まっている必要があります。

EEPROM が 1 個の場合は、P1 を直接ビット操作した方が簡単です。

```
;
;       CS 信号の出力
;   <機能>
;   前もって作業領域に設定した内容に基づき、CS 信号出力の状態を反転する
;   <入力パラメータ（引数）>　なし
;   <戻り値>　なし
;
CSOUT:
        MOV     A,P1                ; ポートの状態をリード
        XOR     A,CSFLAG            ; CS ビットを反転
        MOV     P1,A                ; ポートから出力
        RET
```

(4) ステータスの確認

EEPROM のステータス･レジスタを読み出すことで、書き込み動作中かどうかをチェックします。EEPROM へのデータ転送開始時に、EEPROM のステータス確認のために内部で使用します。

```
;
;       EEPROM が書き込み処理中かどうかのチェック
;   <機能>
;   EEPROM のステータス・レジスタを確認し、書き込み処理中なら
;   キャリー・フラグをセットして戻る
;   <入力パラメータ（引数）>　なし
;   <戻り値>
;   CF　: 0 : 書き込み中ではない　　1 : 書き込み中
;

CHKRDY:
        PUSH    BC              ; レジスタをセーブ
        CALL    !READST         ; ステータス・レジスタのチェック
        MOV     A,C
        POP     BC
        ROR     A,1             ; 書き込み中フラグをキャリーへ
        RET
```

(5) 書き込み許可

対象の EEPROM に「WREN」コマンドを転送します。

```
;       書き込み許可コマンド発行
;   <機能>
;   指定されている EEPROM を書き込み許可に設定する
;   <入力パラメータ（引数）>　なし
;   <戻り値>　なし
;void  _writeenb(void)

__writeenb:
WRITEENB:
        CALL    !CSOUT          ; 対象の EEPROM を選択
        MOV     SOTB10,#WRENC   ; コマンドをセット

WRITEENBLOOP:                   ; 転送完了割り込みを待つ
        CALL    !CSIRDY         ; コマンド転送完了待ち
                                ; コマンド完了で CS 信号を解除
        BR      $CSOUT          ; EEPROM の選択解除
```

(6) 書き込み禁止

対象の EEPROM に「WRDI」コマンドを転送します。

```
;
;       書き込み禁止コマンド発行
;   <機能>
;   指定されている EEPROM を書き込み禁止に設定する
;   <入力パラメータ（引数）>   なし
;   <戻り値>   なし
;void  _writedis(void)

__writedis:
WRITEDIS:
        CALL    !CSOUT          ; 対象の EEPROM を選択
        MOV     SOTB10,#WRDIC   ; コマンドをセット
        BR      $WRITEENBLOOP   ; 完了待ち処理へ
```

(7) ステータス・レジスタの読み出し

対象の EEPROM に「RDSR」コマンドを転送し、EEPROM からのステータス・レジスタの内容を受け取ります。

```
;
;       ステータス・レジスタのリード・コマンド
;   <機能>
;   指定されている EEPROM のステータス・レジスタを読み出す
;   <入力パラメータ（引数）>   なし
;   <戻り値>
;   C   : 戻り値として、読み出したステータスが入る
;unsigned char _readst(void)

__readst:
READST:
        CALL    !CSOUT          ; 対象の EEPROM を選択
        MOV     SOTB10,#RDSRC   ; コマンドをセット
        CALL    !CSIRDY         ; コマンド転送完了待ち
        MOV     SOTB10,#0FFH    ; 受信起動用にダミー書き込み
        CALL    !CSIRDY         ; ステータス転送完了待ち
        MOV     A,SIO10         ; 受信データを読み出す
        MOV     C,A             ; 戻り値をセット
        BR      $CSOUT          ; CS 信号を解除
```

(8) ステータス・レジスタへの書き込み

対象の EEPROM に「WRSR」コマンドを送り、引き続いてステータス・レジスタに設定する値を転送します。

```
;
;       ステータス・レジスタへの書き込みコマンド
;   <機能>
;   引数で渡された値を、指定されている EEPROM のステータス・
; レジスタに書き込む
;   <入力パラメータ（引数）>
;   X    : 引数として書き込み値が入る
;   <戻り値>   なし
;void _writest(unsigned char st)

__writest:
WRITEST:
;         CALL    !WRITEENB          ; 書き込み許可(一度設定すれば個別設定は不要)
          CALL    !CSOUT             ; 対象の EEPROM を選択
          MOV     SOTB10,#WRSRC      ; コマンドをセット
          CALL    !CSIRDY            ; コマンド転送完了待ち
          MOV     A,X
          MOV     SOTB10,A           ; ステータスの送信
          CALL    !CSIRDY            ; ステータス転送完了待ち
          BR      $CSOUT             ; CS 信号を解除
```

(9) データの読み出し 1

512 バイト以下の容量の EEPROM から、256 バイト以下のデータを読み出します。

EEPROM が書き込み中でなければ、指定されたパラメータで書き込みを行います。この際、指定された EEPROM のアドレスが 256 番地より大きい場合は、リード・コマンドのビット 3 をセットします。コマンドに引き続いて、アドレスを 1 バイト転送し、以降は EEPROM からのデータを読み出します。データの読み出しでは、ダミー・データとして FF を書き込んで転送を起動します。読み出しが完了したら、CS 信号を OFF にすることで、EEPROM に読み出しの完了を通知します。

```
;
;       データの読み出し 1
;   <機能>
;   512 バイト以下の容量の EEPROM に対して、引数として渡された
;   パラメータで読み出しを行う
;   EEPROM が書き込み動作中でなければ指定された読み出しを実行し
;   キャリー・フラグをクリアして戻る
;   EEPROM が書き込み動作中なら、読み出しは行わず、キャリー・
;   フラグをセットして戻る
;   <入力パラメータ（引数）>
;   AX      : 第 1 の引数 :EEPROM 内のアドレス
;   SP+2,3: 第 2 の引数 :バッファ・メモリのアドレス
;   SP+4    : 第 3 の引数 :読み出しデータ数
;   <戻り値>
;   CF   : 0 : 正常終了       1 : 書き込み動作中で処理中止
;bit _read_spi_block1(int reg_adr, unsigned char *data, unsigned char size)
```

```
__read_spi_block1:
READBLOCK1:
        PUSH    HL
        PUSH    AX
        CALL    !CHKRDY         ; 書き込み中でないかチェック
        BC      $BUSYEXIT       ; 書き込み中なら処理を中止
        MOVW    AX,SP           ; 2 つ目以降の引数を取り出す準備
        MOVW    HL,AX           ; ポインタを HL レジスタに設定
        CALL    !CSOUT          ; CS 信号を出力
        MOV     A,[HL+1]        ; EEPROM のアドレス上位を読み出し
        AND     A,#1            ; 256 バイト以上かチェック
        ROL     A,1
        ROL     A,1
        ROL     A,1             ; A8 をビット 3 に移動
        OR      A,#READC        ; リード・コマンドを生成
        MOV     SOTB10,A        ; コマンドを送信
        CALL    !CSIRDY         ; コマンド転送完了待ち
READBLOCK:
        MOV     A,[HL]          ; アドレス下位を取り出し
        MOV     SOTB10,A        ; アドレスを送信
        MOV     A,[HL+8]        ; 転送データ数を取り出し
        MOV     RxCOUNT,A       ; データ数を作業領域にコピー
        MOV     A,[HL+6]        ; バッファ・アドレス下位を取り出し
        MOV     X,A
        MOV     A,[HL+7]        ; 上位アドレスを取り出し
        MOVW    HL,AX           ; HL レジスタにアドレスを設定
        CALL    !CSIRDY         ; アドレス転送完了待ち（不要）
;
;       データ受信処理
;
READLOOP:
        MOV     SOTB10,#0FFH    ; 受信用ダミー・データ書き込み
        CALL    !CSIRDY         ; 受信完了待ち
        MOV     A,SIO10         ; 受信データの読み出し
        MOV     [HL],A          ; 受信データをバッファへ
        INCW    HL              ; ポインタを更新
        DEC     RxCOUNT         ; 残りデータ数カウント
        BNZ     $READLOOP       ; データ数分繰り返す。
        CALL    !CSOUT          ; CS 信号をオフ
        CLR1    CY              ; エラー・フラグをクリア

BUSYEXIT:                       ; レジスタを復帰して戻る
        POP     AX
        POP     HL
        RET
```

(10) データの読み出し 2

1K バイト以上の容量の EEPROM から、256 バイト以下のデータを読み出します。

EEPROM が書き込み中でなければ、指定されたパラメータで書き込みを行います。この際、リード・コマンドに引き続いて、アドレスを上位から順に 2 バイト転送し、以降は EEPROM からのデータを読み出します。あとは「**データ読み出し 1**」と同じなので、プログラムを共通化します。

```
;
;       データの読み出し 2
;   <機能>
;   1K バイト以上の容量の EEPROM に対して、引数として渡された
;   パラメータで読み出しを行う
;   EEPROM が書き込み動作中でなければ、指定された読み出しを
;   実行し、キャリー・フラグをクリアして戻る
;   EEPROM が書き込み動作中なら、読み出しは行わず、キャリー・
;   フラグをセットして戻る
;   <入力パラメータ（引数）>
;   AX      : 第 1 の引数 :EEPROM 内のアドレス
;   SP+2,3: 第 2 の引数 :バッファ・メモリのアドレス
;   SP+4    : 第 3 の引数 :読み出しデータ数
;   <戻り値>
;   CF : 0 : 正常終了       1 : 書き込み動作中で処理中止
;bit _read_spi_block2(int reg_adr, unsigned char *data, unsigned char size)

__read_spi_block2:
READBLOCK2:
        PUSH   HL
        PUSH   AX
        CALL   !CHKRDY          ; 書き込み中でないかチェック
        BC     $BUSYEXIT        ; 書き込み中なら処理を中止
        MOVW   AX,SP            ; 2 つ目以降の引数を取り出す準備
        MOVW   HL,AX            ; ポインタを HL レジスタに設定
        CALL   !CSOUT           ; CS 信号を出力
        MOV    SOTB10,#READC    ; リード・コマンドを送信
        CALL   !CSIRDY          ; コマンド転送完了待ち
        MOV    A,[HL+1]         ; EEPROM のアドレス上位を読み出し
        MOV    SOTB10,A         ; アドレスの上位を送信
        CALL   !CSIRDY          ; アドレス転送完了待ち
        BR     $READBLOCK       ; 下位アドレス送信以降のデータ受信処理へ
```

(11) データの書き込み 1

512 バイト以下の容量の EEPROM に、256 バイト以下のデータを書き込みます。EEPROM のステータスを確認し、書き込み中でなければ、書き込み許可コマンドを発行します。書き込むアドレスが 256 番地を越える場合、コマンドのビット 3 をセットしたライト・コマンドを発行し、EEPROM のアドレスを 1 バイト転送します。以降は、指定されたパラメータで書き込み（転送）を行います。データ転送が完了すると、CS 信号を OFF にして、EEPROM に転送完了を通知します。EEPROM はステータス・レジスタのビット 0 を 1 にして、書き込み動作を開始します。このプログラムでは、書き込み完了を待つ処理はしません。このため、転送が完了し、EEPROM が書き込みを行っているときに、別の処理ができます。

```
;
;       データの書き込み 1
;   <機能>
;   512 バイト以下の容量の EEPROM に対して、引数として渡された
;   パラメータで書き込みを行う
;   EEPROM が書き込み動作中でなければ指定された書き込みを実行し
;   キャリー・フラグをクリアして戻る
;   EEPROM が書き込み動作中なら、書き込みは行わず、キャリー・
;   フラグをセットして戻る
;   <入力パラメータ（引数）>
;   AX      : 第 1 の引数 :EEPROM 内のアドレス
;   SP+2,3: 第 2 の引数 :バッファ・メモリのアドレス
;   SP+4   : 第 3 の引数 :書き込みデータ数
;   <戻り値>
;   CF : 0 : 正常終了      1 : 書き込み動作中で処理中止
;bit _write_spi_block1(int reg_adr, unsigned char *data, unsigned char size)

__write_spi_block1:
WRITEBLOCK1:
        PUSH  HL
        PUSH  AX
        CALL  !CHKRDY           ; 書き込み中でないかチェック
        BC    $EXITWRITE        ; 書き込み中なら処理を中止
;       CALL  !WRITEENB         ; 書き込み許可(一度設定すれば個別設定は不要)
        MOVW  AX,SP             ; 2 つ目以降の引数を取り出す準備
        MOVW  HL,AX             ; ポインタを HL レジスタに設定
        CALL  !CSOUT            ; CS 信号を出力
        MOV   A,[HL+1]          ; EEPROM のアドレス上位を読み出し
        AND   A,#1              ; 256 バイト以上かチェック
        ROL   A,1
        ROL   A,1
        ROL   A,1               ; A8 をビット 3 に移動
        OR    A,#WRITEC         ; ライト・コマンドを生成
        MOV   SOTB10,A          ; コマンドを送信
        CALL  !CSIRDY           ; コマンド転送完了待ち
WRITEBLOCK:
        MOV   A,[HL]            ; アドレス下位を取り出し
        MOV   SOTB10,A          ; アドレスを送信
        MOV   A,[HL+8]          ; 転送データ数を取り出し
        MOV   TxCOUNT,A         ; データ数を作業領域にコピー
        MOV   A,[HL+6]          ; バッファ・アドレス下位を取り出し
        MOV   X,A
        MOV   A,[HL+7]          ; 上位アドレスを取り出し
        MOVW  HL,AX             ; HL レジスタにアドレスを設定
        CALL  !CSIRDY           ; アドレス転送完了待ち（不要）
;
;       データ送信処理
;
```

```
WRITELOOP:
        MOV    A,[HL]          ; 4:送信データの読み出し
        MOV    SOTB10,A        ; 5:送信データ書き込み
        INCW   HL              ; 4:ポインタを更新
        CALL   !CSIRDY         ;   :受信完了待ち
        DEC    TxCOUNT         ; 4:残りデータ数カウント
        BNZ    $WRITELOOP      ; 6:データ数分繰り返し
        CALL   !CSOUT          ; CS 信号をオフ
        CLR1   CY              ; エラー・フラグをクリア
EXITWRITE:
        POP    AX
        POP    HL
        RET
```

(12) データの書き込み 2

1K バイト以上の容量の EEPROM に 256 バイト以下のデータを書き込みます。EEPROM のアドレスが 2 バイトになるだけで、あとの処理は「**データ書き込み 1**」と同じです。

```
;
;       データの書き込み 2
;   ＜機能＞
;   1K バイト以上の容量の EEPROM に対して、引数として渡された
;   パラメータで書き込みを行う
;   EEPROM が書き込み動作中でなければ指定された書き込みを実行し
;   キャリー・フラグをクリアして戻る
;   EEPROM が書き込み動作中なら、書き込みは行わず、キャリー・
;   フラグをセットして戻る
;   ＜入力パラメータ（引数）＞
;   AX     : 第 1 の引数 :EEPROM 内のアドレス
;   SP+2,3: 第 2 の引数 :バッファ・メモリのアドレス
;   SP+4   : 第 3 の引数 :書き込みデータ数
;   ＜戻り値＞
;   CF : 0 : 正常終了      1 : 書き込み動作中で処理中止
;bit _write_spi_block2(int reg_adr, unsigned char *data, unsigned char size)

__write_spi_block2:
WRITEBLOCK2:
        PUSH   HL
        PUSH   AX
        CALL   !CHKRDY         ; 書き込み中でないかチェック
        BC     $BUSYEXIT       ; 書き込み中なら処理を中止
;       CALL   !WRITEENB       ; 書き込み許可(一度設定すれば個別設定は不要)
        MOVW   AX,SP           ; 2 つ目以降の引数を取り出す準備
        MOVW   HL,AX           ; ポインタを HL レジスタに設定
        CALL   !CSOUT          ; CS 信号を出力
        MOV    SOTB10,#WRITEC  ; ライト・コマンドを送信
        CALL   !CSIRDY         ; コマンド転送完了待ち
        MOV    A,[HL+1]        ; アドレス上位を取り出し
        MOV    SOTB10,A        ; アドレス上位を送信
        CALL   !CSIRDY         ; 転送完了待ち
        BR     $WRITEBLOCK     ; 下位アドレス以降の処理へ
```

(13) 書き込み完了待ち

EEPROM のステータス・レジスタを読み出すことで、書き込み動作中かどうかをチェックします。書き込みが完了していなければ、完了するまで確認処理を繰り返します。データ書き込み処理のあと処理として使用します。

```
;
;       書き込み完了待ち
;   <機能>
;   EEPROM へのデータ転送後、CS 信号が立ち上がると、転送されたデータが
;   実際に書き込まれる（ここでは、完了するまで待つ）
;   <入力パラメータ（引数）>　なし
;   <戻り値>　なし
;void _waitwt(void)

__waitwt:
WAITWT:
        PUSH  BC                ; レジスタをセーブ

WAITLOOP:
        CALL  !READST           ; ステータス・レジスタのチェック
        MOV   A,C
        ROR   A,1               ; 書き込み中フラグをキャリー･フラグへへ
        BC    $WAITLOOP         ; 書き込み中ならループ
        POP   BC
        RET
```

(14) バイト転送完了待ち

1 バイトのデータ転送が完了するまで、割り込み要求フラグをポーリングすることで待ち合わせます。内部で転送完了の待ち合わせに使用します。

```
;
;       CSI の転送完了待ち
;   <機能>
;   CSI での転送完了を割り込み要求フラグをポーリングして待ち、
;   転送完了したら、割り込み要求フラグをクリアして戻る
;   <入力パラメータ（引数）>　なし
;   <戻り値>　なし
;

CSIRDY:
        NOP
        BF    CSIIF10,$CSIRDY
        CLR1  CSIIF10           ; 割り込み要求フラグをクリア
        RET
```

[C 言語でのプログラム例]

ここでは、C 言語で作成したプログラム例を示します。大きな構成や外部のプログラムとのインタフェースなどは、アセンブラ記述のプログラムと同じです。ただし、制御構造の記述の関係で変更した部分もあります。

(1) 宣言と定義部分

ここでは、関数のプロトタイプ宣言や SPI のコマンドの定義、変数の定義などを行っています。

```
//
//              このモジュールは SPI による EEPROM の
//              制御関数の例です（C 言語記述）
//
#pragma         SFR
#pragma         NOP
void putspi(unsigned char data);        // 1 文字送信
unsigned char getspi(void);             // 1 文字受信
bit selprom(unsigned char spi_adr);     // ROM 選択
bit chkrdy(void);                       // 書き込み完了チェック
void writeenb(void);                    // 書き込み許可設定
void writedis(void);                    // 書き込み禁止設定
unsigned char readst(void);             // ステータス・レジスタの読み出し
void writest(unsigned char st);         // ステータス・レジスタへの書き込み
bit read_spi_block1(int reg_adr, unsigned char *data, unsigned char size);
// 512 バイト以下の EEROM からの読み出し
bit read_spi_block2(int reg_adr, unsigned char *data, unsigned char size);
// 1K バイト以上の EEROM からの読み出し
bit write_spi_block1(int reg_adr, unsigned char *data, unsigned char size);
// 512 バイト以下の EEROM への書き込み
bit write_spi_block2(int reg_adr, unsigned char *data, unsigned char size);
// 1K バイト以上の EEROM への書き込み
void waitwt(void);                      // 転送データの書き込み完了を待つ

/*-------------------------------------------------------*/
/*                                                       */
/*     SPI 関連のコマンド定義                            */
/*                                                       */
/*-------------------------------------------------------*/

#define         WRENC   0b00000110      // 書き込み許可
#define         WRDIC   0b00000100      // 書き込み禁止
#define         RDSRC   0b00000101      // ステータス・レジスタのリード
#define         WRSRC   0b00000001      // ステータス・レジスタへの書き込み
#define         READC   0b00000011      // データ読み出し
#define         WRITEC  0b00000010      // データ書き込み

#define         ROMNO   2               // 接続した EEPROM の数

static sreg unsigned char       csflag; // CS 信号生成用データ
```

(2) 1 文字送信関数

内部で使用する関数で、1 文字分のデータをシリアル・チャネルを介して EEPROM に転送します。転送完了は、割り込み要求(CSIIF10)をポーリングして確認します。EEPROM への送信は、すべてこの関数を介して行います。

```
/*-------------------------------------------------------------*/
/*                                                             */
/*                      シリアル送信処理                       */
/*  <機能>                                                     */
/*  引数で渡された 8 ビットのデータをシリアルで転送する       */
/*  転送が完了したら戻る                                       */
/*  <入力パラメータ（引数）>    送信データ                     */
/*  <戻り値>   なし                                            */
/*                                                             */
/*-------------------------------------------------------------*/

void   putspi(unsigned char data)
{
        SOTB10 = data;                  // データをシリアルで送信
        while(CSIIF10 == 0){            // 転送完了待ち
                NOP();
        }
        CSIIF10 = 0;                    // 転送完了割り込みをクリア
}
```

(3) 1 文字受信関数

この関数は、EEPROM からの 1 文字分のデータをシリアル・チャネル経由で受信します。

```
/*-------------------------------------------------------------*/
/*                                                             */
/*                      シリアル受信処理                       */
/*  <機能>                                                     */
/*  8 ビットのデータをシリアルで転送(受信)する                  */
/*  転送が完了したら受信データを持って戻る                     */
/*  <入力パラメータ（引数）>     なし                          */
/*  <戻り値>    受信データ                                     */
/*                                                             */
/*-------------------------------------------------------------*/

unsigned char   getspi(void)
{
        unsigned char work;
        SOTB10 = 0xff;                  // 受信のためのダミー・データ送信
        while(CSIIF10 == 0){            // 転送完了待ち
                NOP();
        }
        CSIIF10 = 0;                    // 転送完了割り込みをクリア
        work = SIO10;                   // 受信データの読み出し
        return work;
}
```

(4) EEPROM の選択関数

複数の EEPROM を接続する場合に、どの EEPROM かを事前に設定する関数です。1～ROMNO の値を引数として関数をコールしてください。

```
/*-------------------------------------------------------------------*/
/*                                                                   */
/*                        EEPROM の選択                              */
/*  ＜機能＞                                                         */
/*  引数で指定された EEPROM をアクセス対象に設定する                 */
/*    ここでは CS 用のビット・イメージを抽出して作業領域に保存します*/
/*    実際の CS 信号出力は CSOUT 関数で行います                      */
/*  ＜入力パラメータ（引数）＞   EEPROM の番号                       */
/*  ＜戻り値＞                                                       */
/*  エラーならフラグをセットして戻る                                 */
/*                                                                   */
/*-------------------------------------------------------------------*/
bit    selprom(unsigned char spi_adr)
{
       unsigned char work;
       if(spi_adr > ROMNO)        return 1;
       work = 0b00000100;
       csflag = work << spi_adr;
}
```

(5) EEPROM が書き込み中かどうかのチェック関数

EEPROM が書き込み中でないかをステータス･レジスタで確認するために、内部で使用する関数です。

```
/*-------------------------------------------------------------------*/
/*                                                                   */
/*              EEPROM が書き込み処理中かのチェック                  */
/*  ＜機能＞                                                         */
/*  EEPROM のステータス・レジスタを確認し、書き込み処理中なら    */
/*  フラグをセットして戻る                                           */
/*  ＜入力パラメータ（引数）＞  なし                                 */
/*  ＜戻り値＞                                                       */
/*  0 : 書き込み中ではない      1 : 書き込み中                       */
/*                                                                   */
/*-------------------------------------------------------------------*/
bit    chkrdy(void)
{
       unsigned char work;
       work = readst();           // ステータス・レジスタの読み出し
       return work & 0b00000001; // 結果を戻り値にセット
}
```

(6) CS 信号の制御関数

EEPROM の選択関数で指定された EEPROM に対する、CS 信号の出力を制御するための内部関数です。

```
/*-------------------------------------------------------------*/
/*                                                             */
/*                      CS 信号の出力                          */
/*  <機能>                                                     */
/*  作業領域の設定内容に基づき、CS 信号出力の状態を反転する   */
/*  <入力パラメータ（引数）>  なし                             */
/*  <戻り値>  なし                                             */
/*                                                             */
/*-------------------------------------------------------------*/

void csout(void)
{
      P1 = P1 ^ csflag;          // CS 信号を反転
}
```

(7) 書き込み許可関数

選択されている EEPROM に WREN コマンドを転送します。EEPROM に書き込みを行う場合は、最初にこの処理をする必要があります。

```
/*-------------------------------------------------------------*/
/*                                                             */
/*                      書き込み許可コマンド発行               */
/*  <機能>                                                     */
/*  指定されている EEPROM を書き込み許可に設定する            */
/*  <入力パラメータ（引数）>  なし                             */
/*  <戻り値>  なし                                             */
/*                                                             */
/*-------------------------------------------------------------*/

void writeenb(void)
{
      P1 = P1 ^ csflag;          // CS 信号を反転
      putspi(WRENC);             // 書き込み許可コマンドを転送
      P1 = P1 ^ csflag;          // CS 信号を反転
}
```

(8) 書き込み禁止関数

選択されている EEPROM に WRDI コマンドを転送し、書き込み禁止にします。

```
/*-------------------------------------------------------------*/
/*                                                             */
/*                      書き込み禁止コマンド発行               */
/*  <機能>                                                     */
/*  指定されている EEPROM を書き込み禁止に設定する            */
/*  <入力パラメータ（引数）>  なし                             */
/*  <戻り値>  なし                                             */
/*                                                             */
/*-------------------------------------------------------------*/
```

```
void writedis(void)
{
        P1 = P1 ^ csflag;          // CS 信号を反転
        putspi(WRDIC);             // 書き込み禁止コマンドを転送
        P1 = P1 ^ csflag;          // CS 信号を反転
}
```

(9) ステータス・レジスタの読み出し関数

選択されている EEPROM から、ステータス・レジスタの内容を読み出します。ステータス・レジスタのビットの意味は「ステータス・レジスタ」をご参照ください。

```
/*-------------------------------------------------------------*/
/*                                                             */
/*              ステータス・レジスタのリード                   */
/*  <機能>                                                     */
/*  指定されている EEPROM のステータス・レジスタを読み出す     */
/*  <入力パラメータ（引数）>  なし                             */
/*  <戻り値>                                                   */
/*  読み出したステータス                                       */
/*                                                             */
/*-------------------------------------------------------------*/

unsigned char readst(void)
{
        unsigned char work;
        P1 = P1 ^ csflag;          // CS 信号を反転
        putspi(RDSRC);             // コマンドを転送
        work = getspi();           // ステータスの読み出し
        P1 = P1 ^ csflag;          // CS 信号を反転
        return work;
}
```

(10) ステータス・レジスタへの書き込み関数

選択されている EEPROM のステータス・レジスタへの書き込みを行います。書き込むデータは、個々の EEPROM で異なります。通常は、領域ごとのライト・プロテクトを設定するために使用します。

```
/*-------------------------------------------------------------*/
/*                                                             */
/*              ステータス・レジスタへの書き込み               */
/*  引数で渡された値を、指定されている EEPROM のステータス・   */
/*  レジスタに書き込む                                         */
/*  <入力パラメータ（引数）>                                   */
/*      :書き込み値                                            */
/*  <戻り値>  なし                                             */
/*                                                             */
/*-------------------------------------------------------------*/

void writest(unsigned char st)
{
```

```
//      writeenb();                     // 書き込み許可に設定
        P1 = P1 ^ csflag;               // CS 信号を反転
        putspi(WRSRC);                  // コマンドを転送
        putspi(st);                     // 設定データを転送
        P1 = P1 ^ csflag;               // CS 信号を反転
}
```

(11) データ読み出し関数 1

選択されている EEPROM(容量が 512 バイトまでのもの)からデータを読み出します。読み出すデータ数は最大 256 バイトまでです。デバイスの容量より多いデータ数を指定したときの動作は、EEPROM に依存します。

```
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*                          データの読み出し 1                             */
/*  <機能>                                                                 */
/*  512 バイト以下の容量の EEPROM に対して、引数として渡された             */
/*  パラメータで読み出しを行う                                             */
/*  EEPROM が書き込み動作中でなければ指定された読み出しを実行し           */
/*  エラー・フラグをクリアして戻る                                         */
/*  EEPROM が書き込み動作中なら、読み出しは行わず、エラー・                */
/*  フラグをセットして戻る                                                 */
/*  <入力パラメータ（引数）>                                               */
/*  第 1 の引数 :EEPROM 内のアドレス                                       */
/*  第 2 の引数 :読み出したデータを格納するメモリのアドレス                */
/*  第 3 の引数 :読み出しデータ数                                          */
/*  <戻り値>                                                               */
/*   0 : 正常終了     1 : 書き込み動作中で処理中止                         */
/*                                                                         */
/*-------------------------------------------------------------------------*/
bit read_spi_block1(int reg_adr, unsigned char *data, unsigned char size)
{
        unsigned char work;
        int count;                              // 受信データ数のカウンタ
        if(chkrdy())      return 1;             // 書き込み中ならエラーで戻る
        P1 = P1 ^ csflag;                       // CS 信号を反転
        if(reg_adr > 0xff){
                work = READC | 0b00001000;
        }else work = READC;
        putspi(work);                           // コマンドを転送
        work = reg_adr & 0xff;                  //
        putspi(work);                           // アドレスを転送
        if(size){
                count = size;
        }else count = 0x100;                    // 入力が 0 なら 100H に設定
        do{
                work = getspi();                // データの受信
                *data++ = work;
        }while(--count);
        P1 = P1 ^ csflag;                       // CS 信号を反転（選択解除）
}
```

(12) データ読み出し関数 2

選択されているEEPROM(容量が 1K～64K バイト)からデータを読み出します。読み出すデータ数は、最大 256 バイトです。デバイスの容量より多いデータ数を指定したときの動作は、EEPROM に依存します。

```
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*                          データの読み出し 2                             */
/*  ＜機能＞                                                               */
/*  1K バイト以上の容量の EEPROM に対して、引数として渡された              */
/*  パラメータで読み出しを行う                                             */
/*  EEPROM が書き込み動作中でなければ指定された読み出しを実行し            */
/*  エラー・フラグをクリアして戻る                                         */
/*  EEPROM が書き込み動作中なら、読み出しは行わず、エラー・                */
/*  フラグをセットして戻る                                                 */
/*  ＜入力パラメータ（引数）＞                                             */
/*  第 1 の引数 :EEPROM 内のアドレス                                       */
/*  第 2 の引数 :バッファ・メモリのアドレス                                */
/*  第 3 の引数 :読み出しデータ数                                          */
/*  ＜戻り値＞                                                             */
/*  0 : 正常終了      1 : 書き込み動作中で処理中止                         */
/*                                                                         */
/*-------------------------------------------------------------------------*/
bit read_spi_block2(int reg_adr, unsigned char *data, unsigned char size)
{
        unsigned char work;
        int count;                                  // 受信データ数のカウンタ
        if(chkrdy())        return 1;               // 書き込み中ならエラーで戻る
        P1 = P1 ^ csflag;                           // CS 信号を反転
        putspi(READC);                              // コマンドを転送
        work = reg_adr / 0x100;                     // 上位アドレスを抽出
        putspi(work);                               // アドレスを転送
        work = reg_adr & 0xff;                      // 下位アドレスを抽出
        putspi(work);                               // アドレスを転送

        if(size){
                count = size;
        }else count = 0x100;                        // 入力が 0 なら 100H に設定
        do{
                work = getspi();                    // データの受信
                *data++ = work;
        }while(--count);
        P1 = P1 ^ csflag;                           // CS 信号を反転（選択解除）
}
```

(13) データ書き込み関数 1

選択されている EEPROM(容量が 512 バイトまで)にデータを書き込みます。読み出すデータ数は、最大 256 バイトです。デバイスの容量より多いデータ数を指定したときの動作は、EEPROM に依存します。

```
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*                          データの書き込み 1                             */
/*  <機能>                                                                 */
/*  512 バイト以下の容量の EEPROM に対して、引数として渡された             */
/*  パラメータで書き込みを行う                                             */
/*  EEPROM が書き込み動作中でなければ指定された書き込みを実行し           */
/*  エラー・フラグをクリアして戻る                                         */
/*  EEPROM が書き込み動作中なら、書き込みは行わず、エラー・                */
/*  フラグをセットして戻る                                                 */
/*  <入力パラメータ（引数）>                                               */
/*  第 1 の引数 :EEPROM 内のアドレス                                       */
/*  第 2 の引数 :書き込みデータの格納されたバッファ・メモリのアドレス      */
/*  第 3 の引数 :書き込みデータ数                                          */
/*  <戻り値>                                                               */
/*  0 : 正常終了      1 : 書き込み動作中で処理中止                         */
/*                                                                         */
/*-------------------------------------------------------------------------*/
bit write_spi_block1(int reg_adr, unsigned char *data, unsigned char size)
{
        unsigned char work;
        int count;                                  // 送信データ数のカウンタ
        if(chkrdy())        return 1;               // 書き込み中ならエラーで戻る
        P1 = P1 ^ csflag;                           // CS 信号を反転
        if(reg_adr > 0xff){
                work = WRITEC | 0b00001000;
        }else work = WRITEC;
        putspi(work);                               // コマンドを転送
        work = reg_adr & 0xff;                      //
        putspi(work);                               // アドレスを転送

        if(size){
                count = size;
        }else count = 0x100;                        // 入力が 0 なら 100H に設定
        do{
                putspi(*data++);                    // 送信データ書き込み
        }while(--count);
        P1 = P1 ^ csflag;                           // CS 信号を反転（選択解除）
}
```

(14) データ書き込み関数 2

選択されているEEPROM(容量が 1K～64K バイト)にデータを書き込みます。読み出すデータ数は、最大 256 バイトです。デバイスの容量より多いデータ数を指定したときの動作は、EEPROM に依存します。

```
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*                          データの書き込み 2                             */
/*  ＜機能＞                                                               */
/*  1K バイト以上の容量の EEPROM に対して、引数として渡された              */
/*  パラメータで書き込みを行う                                             */
/*  EEPROM が書き込み動作中でなければ指定された書き込みを実行し            */
/*  エラー・フラグをクリアして戻る                                         */
/*  EEPROM が書き込み動作中なら、書き込みは行わず、エラー・                */
/*  フラグをセットして戻る                                                 */
/*  ＜入力パラメータ（引数）＞                                             */
/*  第 1 の引数 ： EEPROM 内のアドレス                                     */
/*  第 2 の引数 ： 書き込みデータの格納されたバッファ・メモリのアドレス    */
/*  第 3 の引数 ： 書き込みデータ数                                        */
/*  ＜戻り値＞                                                             */
/*  0 ： 正常終了       1 ： 書き込み動作中で処理中止                      */
/*                                                                         */
/*-------------------------------------------------------------------------*/
bit write_spi_block2(int reg_adr, unsigned char *data, unsigned char size)
{
        unsigned char work;
        int count;                              // 送信データ数のカウンタ
        if(chkrdy())      return 1;             // 書き込み中ならエラーで戻る
        P1 = P1 ^ csflag;                       // CS 信号を反転
        putspi(WRITEC);                         // コマンドを転送
        work = reg_adr /0x100;                  // 上位アドレスを抽出
        putspi(work);                           //
        work = reg_adr & 0xff;                  //
        putspi(work);                           // アドレスを転送

        if(size){
                count = size;
        }else count = 0x100;                    // 入力が 0 なら 100H に設定
        do{
                putspi(*data++);                // 送信データ書き込み
        }while(--count);
        P1 = P1 ^ csflag;                       // CS 信号を反転（選択解除）
}
```

(15) 書き込み完了待ち関数

EEPROM に書き込みを行ったときに、書き込みが完了するまで待つための関数です。ステータス・レジスタを読み出して、書き込みが完了するまで待ちます。

```
/*------------------------------------------------------------------------*/
/*                                                                        */
/*                          書き込み完了待ち                              */
/*  ＜機能＞                                                              */
/*  EEPROM へのデータ転送後、CS 信号が立ち上がると転送されたデータが      */
/*  実際に書き込みが行われる（ここでは、完了するまで待つ）                */
/*  ＜入力パラメータ（引数）＞　なし                                      */
/*  ＜戻り値＞　なし                                                      */
/*                                                                        */
/*------------------------------------------------------------------------*/

void waitwt(void)
{
        while(readst() & 0b00000001);
}
```

[プログラム例の使用方法 1]

ここでは、アセンブラ記述での使用方法の例を示します。

(1) 宣言部

ここでは、前述のプログラム例のサブルーチンを使用するための宣言をします。また、データを格納するバッファ・メモリ領域も定義します。受信用は DS 擬似命令で領域を確保し、送信用は固定データを準備しています。

```
;
;       このプログラムは SPI 制御ルーチン・モジュールを使用して
;       EEPROM をアクセスする例を示します
;
;       参照する制御サブルーチン
;
        EXTRN   SELPROM         ; アクセスする EEPROM を選択
        EXTRN   WRITEENB        ; 書き込み許可
        EXTRN   WRITEDIS        ; 書き込み禁止
        EXTRN   READST          ; ステータス・レジスタのリード
        EXTRN   WRITEST         ; ステータス・レジスタへのライト
;
        EXTRN   READBLOCK1      ; 512 バイト以下の EEPROM からのデータ・リード
        EXTRN   READBLOCK2      ; 1K バイト以上の EEPROM からのデータ・リード
        EXTRN   WRITEBLOCK1     ; 512 バイト以下の EEPROM へのデータ・ライト
        EXTRN   WRITEBLOCK2     ; 1K バイト以上の EEPROM へのデータ・ライト
;
        EXTRN   WAITWT          ; 書き込み完了待ち
```

```
;
;       作業領域の定義
;
        DSEG
RxBUFF1:                                ; 受信用データ・バッファ
        DS      16
RxBUFF2:
        DS      16
RxBUFF3:
        DS      16
;
;       送信テスト用データ
;
        CSEG
TxDATA1:
        DB      0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
TxDATA2:
        DB      0,1,2,4,8,16,32,64,128,64,32,16,8,4,2,1
```

(2) 読み出し処理

EEPROM からデータを読み出すには、EEPROM を選択し、スタックやレジスタに必要なパラメータをセットして、READBLOCK1（512 バイト以下の EEPROM の場合）か READBLOCK2（1K～64K バイトの EEPROM の場合）をサブルーチン・コールします。エラー（EEPROM が書き込み中）の場合は、キャリー・フラグがセットされて戻ってくるので、エラー処理に分岐させてください。

```
        MOV     X,#1
        CALL    !SELPROM
;
;       512 バイト以下の EEPROM の読み出し例
;
RxTEST:
        MOVW    AX,#16                  ; 読み出しデータ数セット
        PUSH    AX
        MOVW    AX,#RxBUFF1             ; バッファ・アドレスをセット
        PUSH    AX
        MOVW    AX,#33H                 ; EEPROM 内部アドレス
        CALL    !READBLOCK1
        POP     AX
        POP     AX
        BC      $ERRORLOOP              ; エラーならエラー処理へ
```

(3) 書き込み処理

EEPROM にデータを書き込むには、まず EEPROM を選択し、書き込み許可状態にする必要があります。その後、スタックやレジスタに必要なパラメータをセットして、WRITEBLOCK1（512 バイト以下の EEPROM の場合）か WRITEBLOCK2（1K～64K の EEPROM の場合）をサブルーチン・コールします。サブルーチンから戻ってきた段階でエラーが発生していなければ、EEPROM は転送されたデータを実際に書き込んでいます。この間に必要に応じて、他の処理を行ってください。書き込みの完了は WAITWT をコールすることで待ち合わせます。

```
        MOV     X,#1
        CALL    !SELPROM
        CALL    !WRITEENB               ; 書き込み許可
;
;       256～512 の EEPROM への書き込み例
;
        MOVW    AX,#16                  ; データ数のセット（16 バイト）
        PUSH    AX
        MOVW    AX,#TxDATA2             ; バッファ・アドレスのセット
        PUSH    AX
        MOVW    AX,#0155H               ; EEPROM 内部アドレス
        CALL    !WRITEBLOCK1
        POP     AX
        POP     AX
        BC      $ERRORLOOP              ; エラーならエラー処理へ

        〈ここに必要に応じて他の処理を入れる〉

        CALL    !WAITWT                 ; 書き込み完了待ち
```

[プログラム例の使用方法 2]

ここでは、C 言語記述での使用方法の例を示します。使用する制御プログラムが、アセンブラ記述か C 言語記述かは、呼び出す関数の名前が異なるだけです。関数の名前の先頭にある'_'を削除するだけで、上記の C 言語記述の制御プログラム例を使用できます。

(1) 宣言部

ここでは、前述のプログラム例(アセンブラ記述分)のサブルーチンを使用するための宣言をします。

```
#pragma         SFR
extern bit _selprom(unsigned char spi_adr);         // ROM 選択
extern void _writeenb(void);                        // 書き込み許可
extern void _writedis(void);                        // 書き込み禁止
extern unsigned char _readst(void);                 // ステータス・レジスタの読み出し
extern void _writest(unsigned char st);             // ステータス・レジスタへ書き込み
extern bit _read_spi_block1(int reg_adr, unsigned char *data, unsigned char size);
// 512 バイト以下の EEROM からの読み出し
extern bit _read_spi_block2(int reg_adr, unsigned char *data, unsigned char size);
// 1K バイト以上の EEROM からの読み出し
extern bit _write_spi_block1(int reg_adr, unsigned char *data, unsigned char size);
// 512 バイト以下の EEROM への書き込み
extern bit _write_spi_block2(int reg_adr, unsigned char *data, unsigned char size);
// 1K バイト以上の EEROM への書き込み
extern void _waitwt(void);                          // 転送データの書き込み完了を待つ
```

これ以外に、このファイル中にある関数のプロトタイプ宣言や、読み出しデータ用のバッファ・メモリ、固定データを書き込む際のデータ・パターンなどを宣言しています。

```
void hdwinit(void);
bit rxtest(void);
bit    txtest(void);
unsigned char   eep_read_buf1[16];        // read buffer
unsigned char   eep_read_buf2[16];        // read buffer
unsigned char   eep_read_buf3[16];        // read buffer

typedef unsigned char data16[16];
const data16 txdata1 = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
const data16 txdata2 = {0,1,2,4,8,16,32,64,128,64,32,16,8,4,2,1};
```

(2) 読み出し処理

ここでは、ステータス・レジスタの読み出しとデータの読み出しの例を示します。rxtest として関数の形にしてあり、制御プログラムでエラーが発生した場合は、そのままエラーで戻るようになっています。

この例では、1 番の EEPROM を選択し、ステータス・レジスタの内容を読み出し、その後、256 番地未満のアドレス(0x33)と 256 番地以上のアドレス(0x1cc)から 16 バイトのデータを読み出し、その後 EEPROM を切り替えて、512 番地以上(0x33aa)のアドレスから読み出しています。

```
bit rxtest(void)
{
       unsigned char work;
       if(_selprom(1) == 1) return 1;
       work = _readst();
       if(_read_spi_block1(0x33, eep_read_buf1, 16)) return 1;
       if(_read_spi_block1(0x1cc, eep_read_buf2, 16)) return 1;
       if(_selprom(2) == 1) return 1;
       if(_read_spi_block2(0x33cc, eep_read_buf3, 16)) return 1;
}
```

(3) 書き込み処理

書き込みも関数の形にしています。対象 EEPROM を選択後、書き込みを許可して、データを書き込んでいます。

```
bit    txtest(void)
{
       if(_selprom(1) == 1) return 1;
       _writeenb();                                          // 書き込み許可
       if(_write_spi_block1(0xaa, txdata1, 16))return 1;

//     ここに他の処理を入れてもよい

       _waitwt();
       if(_write_spi_block1(0x155, txdata2, 16))return 1;

//     ここに他の処理を入れてもよい

       _waitwt();
       if(_selprom(2) == 1) return 1;
       _writeenb();                                          // 書き込み許可
       if(_write_spi_block2(0x55aa, txdata1, 16))return 1;

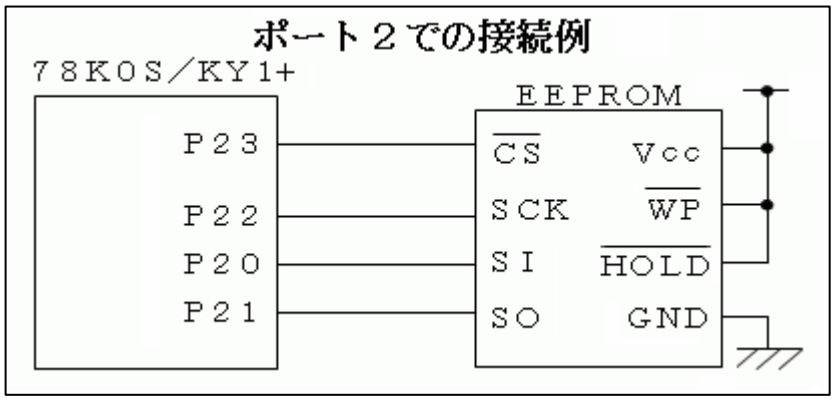
//     ここに他の処理を入れてもよい

       _waitwt();
}
```

SPI メモリと EEPROM の制御プログラム例 [78K0S/Kx1+]

[はじめに]

SPI の概要や SPI インタフェースの EEPROM については「CSI による SPI メモリの制御 [78K0/Kx2 など]」をご参照ください。78K0S/Kx1+には 3 線式のシリアル機能が内蔵されていません。そこで、ポート 2 を用いて、下記の例のように EEPROM を接続して、ソフト的に制御する場合の、具体的な制御プログラムについて説明します。また、制御プログラムの使い方についても、簡単に触れておきます。



[78K0S/KY1+の処理例]

78K0S/KY1+（以下、小ピンと呼ぶ）での EEPROM の制御処理は、以下の部分から構成します。内部で使用するサブルーチン以外は、C 言語からの使用を意識して、名前と入出力パラメータ（引数と戻り値）を決めています。たとえば引数として、読み出しや書き込みでは、次のパラメータを用います。

第 1 パラメータ（AX 渡し）　：EEPROM のアドレス
第 2 パラメータ（スタック渡し）：バッファ・メモリのアドレス
第 3 パラメータ（スタック渡し）：転送データ数で 1～255 と 0（256 バイト）

(1) 初期化部

ポート、割り込みの初期化をします。基本的にベクタ割り込みは使用しないので、割り込みは禁止とします。また、8MHz で動作させたいので、電源電圧が 4V 以上になるまで待ちます。

(2) シリアル出力処理

小ピンには、CSI のようなシリアル機能がないので、ソフトウエアによるパラレル・シリアル変換と出力処理をします。ソフト処理では、通信速度が 200kbps 程度まで低下することが考えられます。ここでは、ループさせないで、できるだけ速度を低下させないようにします。

(3) シリアル入力処理

出力ポートを制御して SCK を出力しながら、入力ポートからシリアル・データを読み込み、パラレル・データに変換します。ここでも、ループさせないで、通信速度の低下を防いでいます。

(4) ステータスの確認

EEPROM がデータ書き込み中かどうかをチェックします。内部で EEPROM への書き込みやデータの読み出しをする前に使用します。

(5) 書き込み許可

対象の EEPROM の CS 信号を ON にして、「WREN」コマンドを送ります。コマンド転送完了後、CS 信号を OFF にします。このコマンドにより、EEPROM は書き込み可能状態になります。

(6) 書き込み禁止

対象の EEPROM の CS 信号を ON にして、「WRDI」コマンドを送ります。コマンド転送完了後、CS 信号を OFF にします。このコマンドにより、EEPROM は書き込み禁止状態になります。

(7) ステータス・レジスタの読み出し

対象の EEPROM の CS 信号を ON にして、「RDSR」コマンドを送り、ステータス・レジスタの状態を 1 バイトのデータで読み出します。読み出し完了後、CS 信号を OFF にします。ステータス・レジスタのビットの意味は「ステータス・レジスタ」をご参照ください。

(8) ステータス・レジスタへの書き込み

対象の EEPROM の CS 信号を ON にして、「WRSR」コマンドを送り、ステータス・レジスタにセットする 1 バイトのデータを書き込みます。書き込みデータの転送完了後、CS 信号を OFF にします。
通常は、ブロックごとのプロテクトの制御（ビット 2、3）に使用します。

(9) データの読み出し1

512 バイト以下（アドレスは 1 バイトで、A8 はコマンドで指定）の容量の EEPROM から、256 バイト以下のデータを読み出します。EEPROM のステータスを確認し、書き込み中でなければ、指定されたパラメータで読み出しをします。

(10) データの読み出し2

1K バイト以上（ただし、アドレスは 2 バイト）の容量の EEPROM から、256 バイト以下のデータを読み出します。EEPROM のステータスを確認し、書き込み中でなければ、指定されたパラメータで読み出しをします。

(11) データの書き込み1

512 バイト以下（アドレスは 1 バイトで、A8 はコマンドで指定）の容量の EEPROM に、256 バイト以下のデータを書き込みます。EEPROM のステータスを確認し、書き込み中でなければ、書き込み許可コマンドを発行し、指定されたパラメータで書き込みをします。
EEPROM へのデータ転送しかしないので、実際に EEPROM で書き込みが完了するまでは、並行して別の処理ができます。この間、EEPROM へのアクセスは、ステータス・レジスタの読み出し以外はしないでください。

(12) データの書き込み2

1K バイト以上（ただし、アドレスは 2 バイト）の容量の EEPROM に、256 バイト以下のデータを書き込みます。EEPROM のステータスを確認し、書き込み中でなければ、書き込み許可コマンドを発行し、指定されたパラメータで書き込みをします。
アドレス長が異なる以外は「**データ書き込み 1**」と同じです。

(13) 書き込み完了待ち

データ書き込み後に、実際に EEPROM でのデータ書き込み完了を待ちます。データ書き込み後は、この処理をするか、ステータス・レジスタを読み出して、ビット 0 が 0(書き込み完了)になったことを確認してください。

[アセンブラでのプログラム例]

このプログラム例では、他のプログラムからこの処理を使えるように、以下の 10 のサブルーチンをパブリック宣言しています。これらは、アセンブラからの使用や C 言語プログラムからの使用も考慮しています。アセンブラから使用する場合は、大文字の名前を EXTRN 宣言してください。C言語のプログラムから使う場合には_で始まる名前を extern 宣言(たとえば、extern _writeenb のように'_'を 1 つにして宣言)してください。

```
        PUBLIC   __writeenb,WRITEENB          ; 書き込み許可
        PUBLIC   __writedis,WRITEDIS          ; 書き込み禁止
        PUBLIC   __readst,READST              ; ステータス・レジスタのリード
        PUBLIC   __writest,WRITEST ; ステータス・レジスタへのライト
;
        PUBLIC   __read_spi_block1,READBLOCK1
                 ; 512 バイト以下の EEPROM からのデータ・リード
        PUBLIC   __read_spi_block2,READBLOCK2
                 ; 1K バイト以上の EEPROM からのデータ・リード
        PUBLIC   __write_spi_block1,WRITEBLOCK1
                 ; 512 バイト以下の EEPROM へのデータ・ライト
        PUBLIC   __write_spi_block2,WRITEBLOCK2
                 ; 1K バイト以上の EEPROM へのデータ・ライト
;
        PUBLIC   __waitwt,WAITWT              ; 書き込み完了待ち
```

また、内部で使用する値の定義を事前に行っています。

```
;
;       EEPROM のコマンド
;
WRENC  EQU      00000110b          ; 書き込み許可
WRDIC  EQU      00000100b          ; 書き込み禁止
RDSRC  EQU      00000101b          ; ステータス・レジスタのリード
WRSRC  EQU      00000001b          ; ステータス・レジスタへの書き込み
READC  EQU      00000011b          ; データ読み出し
WRITEC EQU      00000010b          ; データ書き込み
```

(1) 初期化部

オプション・バイトは次のように設定し、高速内蔵発振クロックを使うようにしておきます。

```
        @@OPTB  CSEG    AT      0080H
        DB      10011100b       ; オプション・バイト
;                 || |||+-----:低速クロック停止可
;                 || |++------:高速内蔵発振クロック使用
;                 || +--------:RESET 端子を使用
;                 ++----------:発振安定時間は最短
;
        DB      11111111b       ; プロテクト・バイト
        END
```

ポートの動作モードを設定します。ここではベクタ割り込みを使用しないので、割り込みはマスクしておきます。また、ソフトウエアでの変換処理のため、できるだけ高速で動作させるために、電源電圧が 4V 以上になるまで、低電圧検出回路を用いて待ちます。

```
;
;       ポートや割り込みのイニシャライズ
;
        MOV     P2,#11111111b           ; 出力ラッチの設定
        MOV     PM2,#11110010b          ; P20、P22、P23 は出力に
        MOV     MK0,#0FFH
        DI
;
;       電源の立ち上がり待ち
;
        MOV     LVIS,#00000000b         ; VLI=4.0V
        MOV     LVIM,#10000000b         ; 低電圧検出開始
        MOV     PCC,#00                 ; CPU clock is fx/4
        MOV     WDTM,#01110000b         ; WDT を停止
        SET1    LSRSTOP                 ; 低速発振器停止
        MOV     A,#50
PONLOOP:
        DEC     A
        BNZ     $PONLOOP                ; 0.2ms 待つ
PONLOOP2:
        BT      LVIF,$PONLOOP2          ; 電圧が 4.0V 以上まで待つ
        MOV     PPCC,#0                 ; CPU clock is fx=fR
```

(2) シリアル出力処理

記述を簡単にするため、次のマクロを定義して、これを 8 回繰り返して、1 バイトのデータをシリアルに変換して、ポートに出力します。受信側は SCK の立ち上がりでデータを取り込むので、SCK の立ち下げとデータの出力を同時に行って、処理速度をできるだけ落とさないようにしています。

```
;
;       シリアル出力処理（マクロ定義）
;
sout1   macro
        ROL     A,1             ; 2:MSB より CY にシフト・アウト
        MOV     X,A             ; 4:
        AND     A,#00000001b    ; 4:データ以外をマスク
        MOV     P2,A            ; 4:SCK 信号立ち下げと同時にデータ出力
        MOV     A,X             ; 4:
        SET1    P2.2            ; 6:SCK 信号を立ち上げ
        endm

;
;       1 文字送信
;   <機能>
;   A レジスタの内容をシリアルで送信する
;   <入力パラメータ（引数）>
;   A : 送信データ
;   <戻り値>  なし
;
```

```
PUTSPI:
        sout1                   ;bit7 出力
        sout1                   ;bit6 出力
        sout1                   ;bit5 出力
        sout1                   ;bit4 出力
        sout1                   ;bit3 出力
        sout1                   ;bit2 出力
        sout1                   ;bit1 出力
        sout1                   ;bit0 出力
        RET
```

(3) シリアル入力処理

SCK を出力し、立ち上げたあとで、シリアル・データを読み込み、A レジスタにシフトしながら蓄積してデータを取り込みます。ループを回さないで処理する際の記述を簡単にするために以下のマクロを定義しておき、このマクロを 8 回実行させることで処理します。

```
;
;       シリアル入力処理（マクロ定義）
;
sin1    macro   bitno
        LOCAL   DATAis1
        MOV     P2,#11110011b     ; 6 : SCK 信号を立ち下げ
        SET1    P2.2              ; 6 : SCK 信号を立ち上げ
        BT      P2.1,$DATAis1     ;10 :
        CLR1    A.bitno           ; 2 : データが 0 ならデータを反転
DATAis1:
        endm

;
;       １文字受信
;   ＜機能＞
;   シリアルからデータを受信する。
;   ＜入力パラメータ（引数）＞　なし
;   ＜戻り値＞
;   A : 受信データ
;

GETSPI:
        MOV     A,#11111111b      ; 初期値として 1 を設定
        sin1    7                 ; MSB から順に受信処理
        sin1    6
        sin1    5
        sin1    4
        sin1    3
        sin1    2
        sin1    1
        sin1    0                 ; 最後に LSB を受信処理
        RET
```

(4) ステータスの確認

EEPROM のステータス･レジスタを読み出して、書き込み動作中かどうかをチェックします。EEPROM へのデータ転送開始時に、EEPROM のステータス確認のために内部で使用します。

```
;
;       EEPROM が書き込み処理中かのチェック
;   ＜機能＞
;   EEPROM のステータス・レジスタを確認し、書き込み処理中なら
;   キャリー・フラグをセットして戻る
;   ＜入力パラメータ（引数）＞　なし
;   ＜戻り値＞
;   CF　：0：書き込み中ではない　　1：書き込み中
;

CHKRDY:
        PUSH    BC              ; レジスタをセーブ
        CALL    !READST         ; ステータス・レジスタのチェック
        MOV     A,C
        POP     BC
        ROR     A,1             ; 書き込み中フラグをキャリー・フラグへ
        RET
```

(5) 書き込み許可

対象の EEPROM に「WREN」コマンドを転送します。

```
;       書き込み許可コマンド発行
;   ＜機能＞
;   指定されている EEPROM を書き込み許可に設定する
;   ＜入力パラメータ（引数）＞　なし
;   ＜戻り値＞　なし
;void  _writeenb(void)

__writeenb:
WRITEENB:
        MOV     A,#WRENC ; コマンドをセット
WRITEENBLOOP:                   ; 転送完了割り込みを待つ
        CLR1    P2.3            ; CS 信号を立ち下げ
        CALL    !PUTSPI         ; コマンド転送
        SET1    P2.3            ; コマンド完了で CS 信号を解除
        RET
```

(6) 書き込み禁止

対象の EEPROM に「WRDI」コマンドを転送します。コマンド以外は、「**書き込み許可**」と同じ処理です。

```
;
;       書き込み禁止コマンド発行
;   ＜機能＞
;   指定されている EEPROM を書き込み禁止に設定する
;   ＜入力パラメータ（引数）＞　なし
;   ＜戻り値＞　なし
;void  _writedis(void)
```

```
__writedis:
WRITEDIS:
        MOV     A,#WRDIC            ; コマンドをセット
        BR      $WRITEENBLOOP       ; 完了待ち処理へ
```

(7) ステータス・レジスタの読み出し

対象の EEPROM に「RDSR」コマンドを転送し、EEPROM からのステータス・レジスタの内容を受け取ります。

```
;
;       ステータス・レジスタのリード・コマンド
;   <機能>
;   指定されている EEPROM のステータス・レジスタを読み出す
;   <入力パラメータ（引数）>　なし
;   <戻り値>
;   C　：戻り値として、読み出したステータスが入る
;unsigned char _readst(void)

__readst:
READST:
        CLR1    P2.3                ; CS 信号を立ち下げ
        MOV     A,#RDSRC            ; コマンドをセット
        CALL    !PUTSPI             ; コマンド転送
        CALL    !GETSPI             ; データ受信
        MOV     C,A                 ; 戻り値をセット
        SET1    P2.3                ; CS 信号を解除
        RET
```

(8) ステータス・レジスタへの書き込み

対象の EEPROM に「WRSR」コマンドを送り、引き続いてステータス・レジスタに設定する値を転送します。

```
;
;       ステータス・レジスタへの書き込みコマンド
;   <機能>
;   指定されている EEPROM のステータス・レジスタに、引数で渡された値を書き込む
;   <入力パラメータ（引数）>
;   X　：引数として書き込み値が入る
;   <戻り値>　なし
;void _writest(unsigned char st)

__writest:
WRITEST:
        PUSH    AX                  ; ステータスをセーブ
        CLR1    P2.3                ; CS 信号を立ち下げ
        MOV     A,#WRSRC            ; コマンドをセット
        CALL    !PUTSPI             ; コマンド転送完了待ち
        POP     AX                  ; ステータスの復帰
        MOV     A,X
        CALL    !PUTSPI             ; ステータス転送
        SET1    P2.3                ; CS 信号を解除
        RET
```

(9) データの読み出し 1

512 バイト以下の容量の EEPROM から、256 バイト以下のデータを読み出します。

EEPROM が書き込み中でなければ、指定されたパラメータで書き込みをします。この際、指定された EEPROM のアドレスが 256 番地より大きい場合は、リード・コマンドのビット 3 をセットします。コマンドに続いて、アドレスを 1 バイト転送し、以降は EEPROM からのデータを読み出します。データの読み出しでは、ダミー・データとしてFFを書き込んで転送を起動します。読み出しが完了したら、CS 信号を OFF にして、EEPROM に通知します。

```
;
;       データの読み出し 1
;   ＜機能＞
;   512 バイト以下の容量の EEPROM に対して、引数として渡された
;   パラメータで読み出しを行う
;   EEPROM が書き込み動作中でなければ、指定された読み出しを実行し
;   キャリー・フラグをクリアして戻る
;   EEPROM が書き込み動作中なら、読み出しはせず、キャリー・
;   フラグをセットして戻る
;   ＜入力パラメータ（引数）＞
;   AX      : 第 1 の引数 :EEPROM 内のアドレス
;   SP+2,3: 第 2 の引数 :バッファ・メモリのアドレス
;   SP+4    : 第 3 の引数 :読み出しデータ数
;   ＜戻り値＞
;   CF  : 0 : 正常終了      1 : 書き込み動作中で処理中止
;bit _read_spi_block1(int reg_adr, unsigned char *data, unsigned char size)

__read_spi_block1:
READBLOCK1:
        PUSH    HL
        PUSH    AX
        CALL    !CHKRDY             ; 書き込み中でないかチェック
        BC      $BUSYEXIT           ; 書き込み中なら処理を中止
        MOVW    AX,SP               ; 2 つ目以降の引数を取り出す準備
        MOVW    HL,AX               ; ポインタを HL レジスタに設定
        PUSH    BC
        CLR1    P2.3                ; CS を立ち下げる
        MOV     A,[HL+1]            ; EEPROM のアドレス上位を読み出し
        AND     A,#1                ; 256 バイト以上かのチェック
        ROL     A,1
        ROL     A,1
        ROL     A,1                 ; A8 をビット 3 に移動
        OR      A,#READC            ; リード・コマンドを生成
        CALL    !PUTSPI             ; コマンド転送
READBLOCK:
        MOV     A,[HL]              ; アドレス下位を取り出し
        CALL    !PUTSPI             ; アドレスを送信
        MOV     A,[HL+8]            ; 転送データ数を取り出し
        MOV     B,A                 ; データ数を作業領域にコピー
        MOV     A,[HL+6]            ; バッファ・アドレス下位を取り出し
        MOV     X,A
        MOV     A,[HL+7]            ; 上位アドレスを取り出し
        MOVW    HL,AX               ; HL レジスタにアドレスを設定
```

```
;
;       データ受信処理
;
READLOOP:
        CALL    !GETSPI             ; 受信起動
        MOV     [HL],A              ; 受信データをバッファへ
        INCW    HL                  ; ポインタを更新
        DBNZ    B,$READLOOP         ; データ数分繰り返す
        SET1    P2.3                ; CS 信号を解除
        POP     BC
        CLR1    CY                  ; エラー・フラグのクリア
BUSYEXIT:                           ; レジスタを復帰させて戻る
        POP     AX
        POP     HL
        RET
```

(10) データの読み出し 2

1K バイト以上の容量の EEPROM から 256 バイト以下のデータを読み出します。

EEPROM が書き込み中でなければ、指定されたパラメータで書き込みを行います。この際、リード･コマンドに引き続いて、アドレスを上位から順に 2 バイト転送し、以降は EEPROM からのデータを読み出します（あとは「**データ読み出し 1**」と同じなので、プログラムを共通化します。

```
;
;       データの読み出し 2
;   ＜機能＞
;   1K バイト以上の容量の EEPROM に対して、引数として渡された
;   パラメータで読み出しを行う
;   EEPROM が書き込み動作中でなければ指定された読み出しを実行し
;   キャリー・フラグをクリアして戻る
;   EEPROM が書き込み動作中なら、読み出しは行わず、キャリー・
;   フラグをセットして戻る
;   ＜入力パラメータ（引数）＞
;   AX     : 第 1 の引数 :EEPROM 内のアドレス
;   SP+2,3: 第 2 の引数 :バッファ・メモリのアドレス
;   SP+4   : 第 3 の引数 :読み出しデータ数
;   ＜戻り値＞
;   CF  : 0 : 正常終了      1 : 書き込み動作中で処理中止
;bit _read_spi_block2(int reg_adr, unsigned char *data, unsigned char size)

__read_spi_block2:
READBLOCK2:
        PUSH    HL
        PUSH    AX
        CALL    !CHKRDY             ; 書き込み中でないかチェック
        BC      $BUSYEXIT           ; 書き込み中なら処理を中止
        MOVW    AX,SP               ; 2 つ目以降の引数を取り出す準備
        MOVW    HL,AX               ; ポインタを HL レジスタに設定
        PUSH    BC
        CLR1    P2.3                ; CS 信号を立ち下げ
        MOV     A,#READC            ; リード・コマンドをセット
        CALL    !PUTSPI             ; コマンド転送
        MOV     A,[HL+1]            ; EEPROM のアドレス上位を読み出す
        CALL    !PUTSPI             ; アドレス転送
        BR      $READBLOCK          ; 下位アドレス送信以降のデータ受信処理へ
```

(11) データの書き込み 1

512 バイト以下の容量の EEPROM に、256 バイト以下のデータを書き込みます。EEPROM のステータスを確認し、書き込み中でなければ、書き込み許可コマンドを発行します。書き込むアドレスが 256 番地を越える場合は、コマンドのビット 3 をセットしたライト・コマンドを発行し、EEPROM のアドレスを 1 バイト転送します。以降は、指定されたパラメータで書き込み(転送)を行います。データの転送が完了すると、CS 信号を OFF にして、EEPROM に転送完了を通知します。EEPROM は、ステータス・レジスタのビット 0 を 1 にして、実際の書き込み動作を開始します。このプログラムでは、書き込み完了を待つ処理はしません。このため、転送が完了し、EEPROM が書き込みをしているときに、別の処理を行うことができます。

```
;
;       データの書き込み 1
;   <機能>
;   512 バイト以下の容量の EEPROM に対して、引数として渡された
;   パラメータで書き込みを行う
;   EEPROM が書き込み動作中でなければ指定された書き込みを実行し
;   キャリー・フラグをクリアして戻る
;   EEPROM が書き込み動作中なら、書き込みは行わず、キャリー・
;   フラグをセットして戻る
;   <入力パラメータ（引数）>
;   AX     : 第 1 の引数 :EEPROM 内のアドレス
;   SP+2,3: 第 2 の引数 :バッファ・メモリのアドレス
;   SP+4   : 第 3 の引数 :書き込みデータ数
;   <戻り値>
;   CF  : 0 : 正常終了     1 : 書き込み動作中で処理中止
;bit _write_spi_block1(int reg_adr, unsigned char *data, unsigned char size)

__write_spi_block1:
WRITEBLOCK1:
        PUSH    HL
        PUSH    AX
        CALL    !CHKRDY             ; 書き込み中でないかチェック
        BC      $EXITWRITE          ; 書き込み中なら処理を中止
        MOVW    AX,SP               ; 2 つ目以降の引数を取り出す準備
        MOVW    HL,AX               ; ポインタを HL レジスタに設定
        PUSH    BC
        CLR1    P2.3                ; CS 信号を立ち下げ
        MOV     A,[HL+1]            ; EEPROM のアドレス上位を読み出す
        AND     A,#1                ; 256 バイト以上かのチェック
        ROL     A,1
        ROL     A,1
        ROL     A,1                 ; A8 をビット 3 に移動
        OR      A,#WRITEC           ; ライト・コマンドを生成
        CALL    !PUTSPI             ; コマンド転送
WRITEBLOCK:
        MOV     A,[HL]              ; アドレス下位を取り出す
        CALL    !PUTSPI             ; アドレスを送信
        MOV     A,[HL+8]            ; 転送データ数を取り出す
        MOV     B,A                 ; データ数を作業領域にコピー
        MOV     A,[HL+6]            ; バッファ・アドレス下位を取り出す
        MOV     X,A
        MOV     A,[HL+7]            ; 上位アドレスを取り出す
        MOVW    HL,AX               ; HL レジスタにアドレスを設定
```

```
;
;       データ送信処理
;
WRITELOOP:
        MOV     A,[HL]          ; 送信データの読み出し
        MOV     X,A
        CALL    !PUTSPI         ; データを送信
        INCW    HL              ; ポインタを更新
        DBNZ    B,$WRITELOOP    ; データ数分繰り返す。
        SET1    P2.3            ; CS 信号を解除
        POP     BC
        CLR1    CY              ; エラー・フラグのクリア

EXITWRITE:
        POP     AX
        POP     HL
        RET
```

(12) データの書き込み 2

1K バイト以上の容量の EEPROM に 256 バイト以下のデータを書き込みます。EEPROM のアドレスが 2 バイトになるだけで、その後の処理は「**データ書き込み 1**」と同じです。

```
;
;       データの書き込み 2
;   <機能>
;   1K バイト以上の容量の EEPROM に対して、引数として渡された
;   パラメータで書き込みを行う
;   EEPROM が書き込み動作中でなければ指定された書き込みを実行し
;   キャリー・フラグをクリアして戻る
;   EEPROM が書き込み動作中なら、書き込みは行わず、キャリー・
;   フラグをセットして戻る
;   <入力パラメータ（引数）>
;   AX     : 第 1 の引数 :EEPROM 内のアドレス
;   SP+2,3: 第 2 の引数 :バッファ・メモリのアドレス
;   SP+4   : 第 3 の引数 :書き込みデータ数
;   <戻り値>
;   CF  : 0 : 正常終了     1 : 書き込み動作中で処理中止
;bit _write_spi_block2(int reg_adr, unsigned char *data, unsigned char size)

__write_spi_block2:
WRITEBLOCK2:
        PUSH    HL
        PUSH    AX
        CALL    !CHKRDY         ; 書き込み中でないかチェック
        BC      $BUSYEXIT       ; 書き込み中なら処理を中止
        MOVW    AX,SP           ; 2 つ目以降の引数を取り出す準備
        MOVW    HL,AX           ; ポインタを HL レジスタに設定
        PUSH    BC
        CLR1    P2.3            ; CS 信号を立ち下げる
        MOV     A,#WRITEC       ; ライト・コマンドをセット
        CALL    !PUTSPI         ; コマンド転送
        MOV     A,[HL+1]        ; アドレス上位を取り出す
        CALL    !PUTSPI         ; 上位アドレスの転送
        BR      $WRITEBLOCK     ; 下位アドレス以降の処理へ
```

(13) 書き込み完了待ち

EEPROM のステータス・レジスタを読み出して、書き込み動作中かどうかをチェックします。書き込みが完了していなければ、完了するまで確認処理を繰り返します。データ書き込み処理の、あと処理に使用します。

```
;
;       書き込み完了待ち
;   <機能>
;   EEPROM へのデータ転送後、CS 信号が立ち上がると転送されたデータが
;   実際に書き込みが行われる（ここでは、完了するまで待つ）
;   <入力パラメータ（引数）>　なし
;   <戻り値>　なし
;void _waitwt(void)

__waitwt:
WAITWT:
        PUSH    BC              ; レジスタをセーブ

WAITLOOP:
        CALL    !READST         ; ステータス・レジスタのチェック
        MOV     A,C
        ROR     A,1             ; 書き込み中フラグをキャリー・フラグへ
        BC      $WAITLOOP       ; 書き込み中ならループ
        POP     BC
        RET
```

[プログラム例の使用方法 1]

上記の SPI の制御プログラム例をアセンブラ記述のプログラムから使用する方法の例を示します。

(1) 宣言部

ここでは、前述のプログラム例のサブルーチンを使用するための宣言をします。また、データを格納するためのバッファ・メモリ領域も定義します。受信用は DS 擬似命令で領域を確保し、送信用は固定データを準備しています。

```
;
;       このプログラムは、SPI 制御ルーチン・モジュールを使用して
;       EEPROM をアクセスする例を示します
;
;       参照する制御サブルーチン
;
        EXTRN   WRITEENB        ; 書き込み許可
        EXTRN   WRITEDIS        ; 書き込み禁止
        EXTRN   READST          ; ステータス・レジスタのリード
        EXTRN   WRITEST         ; ステータス・レジスタへのライト
;
        EXTRN   READBLOCK1      ; 512 バイト以下の EEPROM からのデータ・リード
        EXTRN   READBLOCK2      ; 1K バイト以上の EEPROM からのデータ・リード
        EXTRN   WRITEBLOCK1     ; 512 バイト以下の EEPROM へのデータ・ライト
        EXTRN   WRITEBLOCK2     ; 1K バイト以上の EEPROM へのデータ・ライト
;
        EXTRN   WAITWT          ; 書き込み完了待ち
```

```
;
;       作業領域の定義
;

        DSEG
RxBUFF1:                        ; 受信用データ・バッファ
        DS      16
RxBUFF2:
        DS      16
RxBUFF3:
        DS      16
;
;       送信テスト用データ
;
        CSEG
TxDATA1:
        DB      0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
TxDATA2:
        DB      0,1,2,4,8,16,32,64,128,64,32,16,8,4,2,1
```

(2) 読み出し処理

EEPROM からデータを読み出すには、必要なパラメータをスタックやレジスタにセットして、READBLOCK1（512 バイト以下の EEPROM の場合）か READBLOCK2（1K～64K バイトの EEPROM の場合）をサブルーチン・コールします。エラー（EEPROM が書き込み中）の場合は、キャリー・フラグがセットされて戻ってくるので、エラー処理に分岐させてください。

```
;
;       512 バイト以下の EEPROM の読み出し例
;
RxTEST:
        MOVW    AX,#16                  ; 読み出しデータ数セット
        PUSH    AX
        MOVW    AX,#RxBUFF1             ; バッファ・アドレスをセット
        PUSH    AX
        MOVW    AX,#33H                 ; EEPROM 内部アドレス
        CALL    !READBLOCK1
        POP     AX
        POP     AX
        BC      $ERRORLOOP              ; エラーならエラー処理へ
```

(3) 書き込み処理

EEPROMにデータを書き込むには、書き込みを許可状態にする必要があります。その後、必要なパラメータをスタックやレジスタにセットして、WRITEBLOCK1（512 バイト以下の EEPROM の場合）か WRITEBLOCK2（1K～64K の場合）をサブルーチン・コールします。サブルーチンから戻ってきた段階で、エラーが発生していなければ、EEPROM は転送されたデータを実際に書き込んでいます。この間に必要に応じて、他の処理を行ってください。書き込みの完了は、WAITWT をコールすることで待ち合わせます。

```
        CALL    !WRITEENB                   ; 書き込み許可
;
;       256～512 の EEPROM への書き込み例
;
        MOVW    AX,#16                      ; データ数のセット（16 バイト）
        PUSH    AX
        MOVW    AX,#TxDATA2                 ; バッファ・アドレスのセット
        PUSH    AX
        MOVW    AX,#0155H                   ; EEPROM 内部アドレス
        CALL    !WRITEBLOCK1
        POP     AX
        POP     AX
        BC      $ERRORLOOP                  ; エラーならエラー処理へ

        〈ここに必要に応じて他の処理を入れる〉

        CALL    !WAITWT                     ; 書き込み完了待ち
```

[プログラム例の使用方法 2]

ここでは、C 言語記述での使用方法の例を示します。

(1) 宣言部

ここでは、前述のプログラム例（アセンブラ記述分）のサブルーチンを使用するための宣言を行います。

```
#pragma         SFR
extern void _writeenb(void);                    // 書き込み許可
extern void _writedis(void);                    // 書き込み禁止
extern unsigned char _readst(void);             // ステータス・レジスタの読み出し
extern void _writest(unsigned char st);         // ステータス・レジスタへ書き込み
extern bit _read_spi_block1(int reg_adr, unsigned char *data, unsigned char size);
// 512 バイト以下の EEROM からの読み出し
extern bit _read_spi_block2(int reg_adr, unsigned char *data, unsigned char size);
// 1K バイト以上の EEROM からの読み出し
extern bit _write_spi_block1(int reg_adr, unsigned char *data, unsigned char size);
// 512 バイト以下の EEROM への書き込み
extern bit _write_spi_block2(int reg_adr, unsigned char *data, unsigned char size);
// 1K バイト以上の EEROM への書き込み
extern void _waitwt(void);                      // 転送データの書き込み完了を待つ
```

これ以外に、このファイル中にある関数のプロトタイプ宣言や、読み出しデータ用のバッファ・メモリ、固定データを書き込む際のデータ・パターンなどを宣言しています。

```
Void   hdwinit(void);
bit    rxtest(void);
bit    txtest(void);
unsigned char   eep_read_buf1[16];// read buffer
unsigned char   eep_read_buf2[16];// read buffer
unsigned char   eep_read_buf3[16];// read buffer
typedef unsigned char data16[16];
const data16 txdata1 = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
const data16 txdata2 = {0,1,2,4,8,16,32,64,128,64,32,16,8,4,2,1};
```

(2) 読み出し処理

ここでは、ステータス・レジスタの読み出しとデータの読み出しの例を示します。rxtest として関数の形にしてあり、制御プログラムでエラーが発生した場合は、そのままエラーで戻るようになっています。

この例では、1 番の EEPROM を選択し、ステータス・レジスタの内容を読み出し、その後、256 番地未満のアドレス(0x33)と 256 番地以上のアドレス(0x1cc)から 16 バイトのデータを読み出し、その後 EEPROM を切り替えて、512 番地以上(0x33aa)のアドレスから読み出しています。

```
bit rxtest(void)
{
        unsigned char work;
        if(_selprom(1) == 1) return 1;
        work = _readst();
        if(_read_spi_block1(0x33, eep_read_buf1, 16)) return 1;
        if(_read_spi_block1(0x1cc, eep_read_buf2, 16)) return 1;
        if(_selprom(2) == 1) return 1;
        if(_read_spi_block2(0x33cc, eep_read_buf3, 16)) return 1;
}
```

(3) 書き込み処理

書き込みについても関数の形にしてあります。対象とする EEPROM を選択した後に、書き込み許可状態にして、データの書き込みを行っています。

```
bit    txtest(void)
{
        if(_selprom(1) == 1) return 1;
        _writeenb();                                          // 書き込み許可
        if(_write_spi_block1(0xaa, txdata1, 16))return 1;

//      ここに他の処理を入れてもよい
        _waitwt();
        if(_write_spi_block1(0x155, txdata2, 16))return 1;

//      ここに他の処理を入れてもよい
        _waitwt();
        if(_selprom(2) == 1) return 1;
        _writeenb();                                          // 書き込み許可
        if(_write_spi_block2(0x55aa, txdata1, 16))return 1;

//      ここに他の処理を入れてもよい
        _waitwt();
}
```

CSI によるマイクロワイヤ(Microwire)メモリの制御 [78K0/Kx2 など]

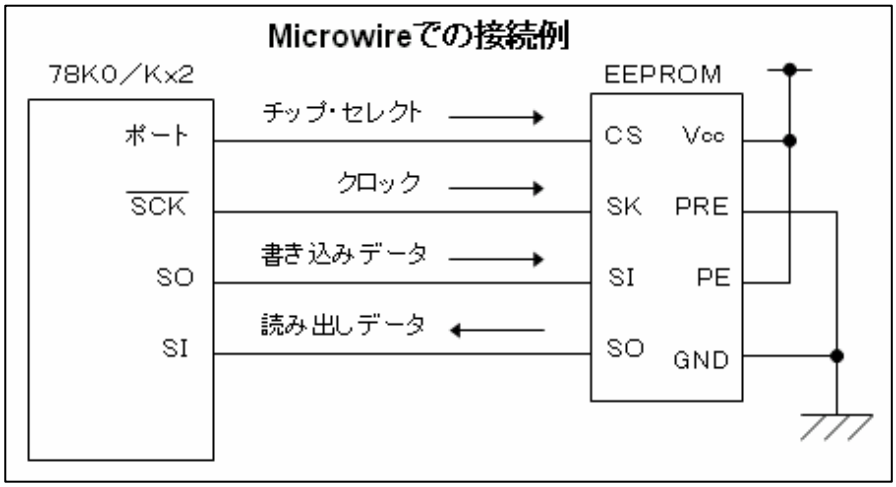
[はじめに]

Microwire はナショナルセミコンダクタ社が提唱したマスタ／スレーブ方式のインタフェースで、基本的なデータの転送は 2 本のデータ（入力と出力）とクロックの 3 本の信号で行うクロック同期式のシリアル通信方式です。これ以外に、チップ・セレクトとして CS 信号が使用され、合計 4 本で制御します。

データは、基本的に 16 ビットでやりとりされます。この部分は、78K0 シリーズなどに搭載されたシリアル・インタフェース（CSI）と、基本的に互換性があります。Microwire では、データ以外に 1 ビットのスタート・ビットと、2 ビットのコマンド、デバイスの容量によってビット長の異なるアドレスが使用されます。

[EEPROM との接続]

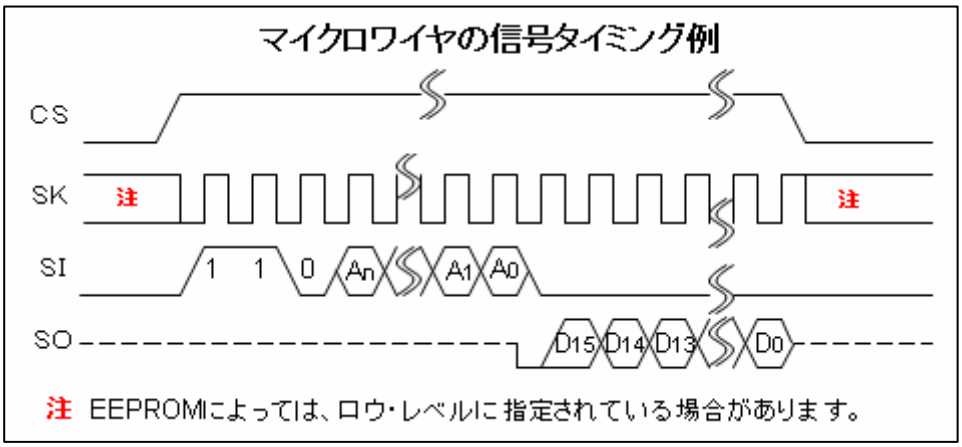
Microwire インタフェースをもつ EEPROM との接続例は、次のようになります。



EEPROMに対するチップ・セレクト信号は、ポートを用いて出力します。その他の信号は、3線式シリアルの信号をそのまま使用します。なお、EEPROM には、これ以外に書き込みを禁止するための信号（PRE や PE）があるものもあります。ここでは、これらの信号は使用しないので、電源に接続しておきます。

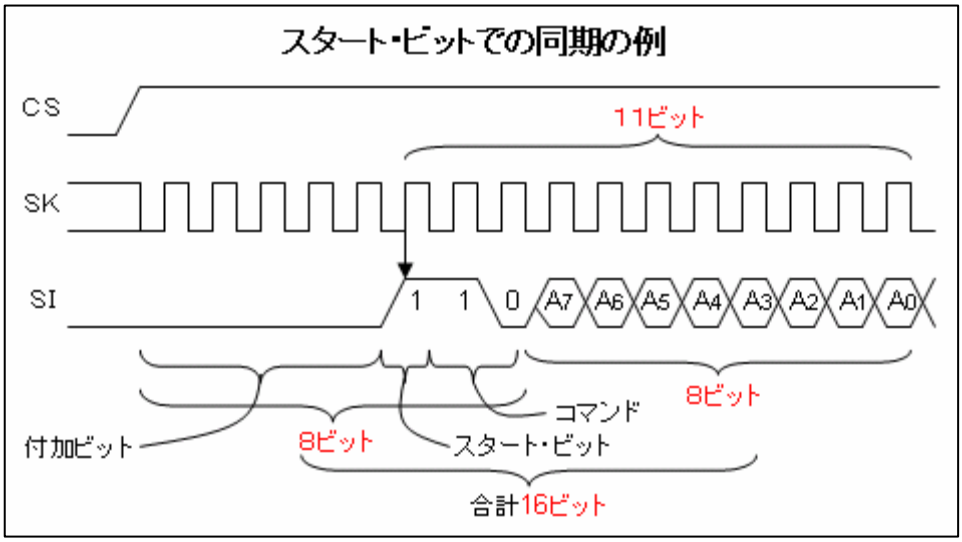
[信号のやり取りタイミング]

Microwire インタフェースをもつ EEPROM は、CS 信号がアクティブの状態で、シリアル・クロック（SK）の立ち下がりに同期して送られてきたコマンドやデータを SK の立ち上がりで受け取ります。ところがリードでは、SK の立ち上がりに同期してデータを出力するので、立ち下がりに同期して読み出す必要があります。これは、3 線式シリアルや SPI とは異なるので注意が必要です。78K の CSI では、送信（書き込み）方向ではタイプ 4、受信（読み出し）方向ではタイプ 3 の通信フォーマットを使用します。



[スタート・ビットでの同期]

Microwire インタフェースをもつ EEPROM では、その容量によってアドレスのビット数が異なります。そのため、アドレス転送までに必要なビット数が、CSI のような 8 ビット固定では合わなくなります。そこで、スタート・ビットが定義されています。つまり、最初に受信した1のビットまでは、CS がアクティブになっていても、SK は無視されます。これを利用すれば、全体の長さは 8 ビットや 16 ビットの倍数になります。たとえば、容量が 256 ワードの場合には、アドレスが 8 ビットになります。スタート・ビットとコマンドで 3 ビットですから、スタート・ビットの前に 5 ビットの 0 を付加すると、全体で 16 ビットにすることができます。この例を次図に示します。



[EEPROM のコマンド]

EEPROM にはいくつかのコマンドが定義されており、最初にこれらで動作を指定する必要があります。デバイスによって、使用できるコマンドの種類が異なるようなので、コマンドの詳細は個々のデバイスのマニュアルをご参照いただくことにして、ここでは概要を説明します。

コード	名　称	動　　作
00+00	WDS	書き込み禁止
00+01	WRALL	全空間に同じデータを書き込む
00+10	ERAL	すべてのアドレスのデータを消去
00+11	WEN	書き込み許可
01	WRITE	指定されたアドレスに 16 ビットのデータを書き込む
10	READ	指定されたアドレスから順にデータを読み出す
11	ERASE	指定されたアドレスのデータを消去

WDS　:デバイスへの書き込み禁止状態にします。コマンドが 00 で、アドレス部分の最初の 2 ビットが 00 のときに、このコマンドとなります。残りのアドレス部分は何でも構いません。
アドレスが 8 ビット時のスタート・ビット以降のビット・パターンを示します。
SI:　1 00 00xxxxxx

WRALL　:デバイスのすべてのアドレスに、同じデータを書き込みます。コマンドが 00 で、アドレス部分の最初の 2 ビットが 01 のときに、このコマンドとなります。残りのアドレス部分は何でも構いません。
アドレスが 8 ビット時のスタート・ビット以降のビット・パターンを示します。
SI:　1 00 01xxxxxx D15～D0

ERAL　:デバイスのすべてのアドレスのデータをクリアします。コマンドが 00 で、アドレス部分の最初の 2 ビットが 10 のときに、このコマンドとなります。残りのアドレス部分は何でも構いません。
デバイスによっては、このコマンドがない場合もあるようです。
アドレスが 8 ビット時のスタート・ビット以降のビット・パターンを示します。
SI:　1 00 10xxxxxx

WEN　:デバイスへの書き込みを可能状態にします。電源立ち上げ時、デバイスは書込み禁止状態になっているので、書き込みを行う場合には、必ずこのコマンドで書き込みを許可する必要があります。コマンドが 00 で、アドレス部分の最初の 2 ビットが 11 のときに、このコマンドとなります。残りのアドレス部分は何でも構いません。
アドレスが 8 ビット時のスタート・ビット以降のビット・パターンを示します。
SI:　1 00 11xxxxxx

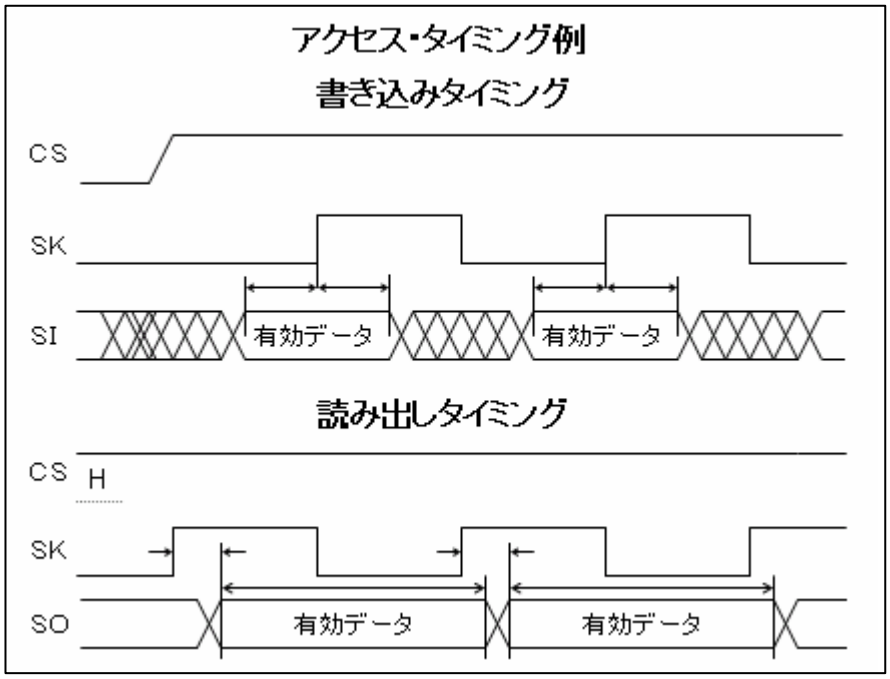
WRITE　:EEPROM へデータを書き込みます。コマンドに続いて、書き込みたいメモリのアドレスを転送します。その後、指定されたアドレスへの書き込みデータ(16 ビット)を転送します。転送完了後、次の SK の立ち上がりより前に、CS をロウ・レベルにすれば、実際の書き込みを行います。書き込んでいる最中には、再度 CS をハイ・レベルにして、DO 端子を確認することで、書き込みを完了したかどうかが分かります。0 なら書き込み中で、1 になったら書き込みは完了です。なお、書き込み完了を確認する際は、SI をロウ・レベルにする必要があるようです。
アドレスが 8 ビット時のスタート・ビット以降のビット・パターンを示します。
SI:　1 01 A7～A0 D15～D0

READ　:EEPROM からデータを読み出します。コマンドに続いて、アドレスを転送すれば、その後の SK の立ち上がりで、指定されたアドレスの内容を出力します。その後、SK を入力すれば、次のアドレスの内容を読み出せます。CS をロウ・レベルにすれば、読み出しは完了となります。
アドレスが 8 ビット時のスタート・ビット以降のビット・パターンを示します。
SI:　1 10 A7～A0

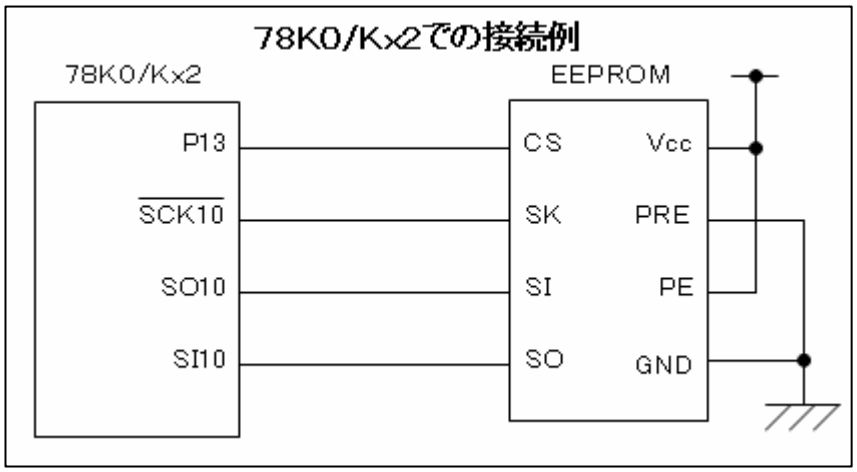
ERASE　:EEPROM の指定したアドレスのデータをクリアします。このコマンドがないデバイスもあるようです。
アドレスが 8 ビット時のスタート・ビット以降のビット・パターンを示します。
SI:　1 11 A7～A0

[78K0/Kx2 での制御]

EEPROM は、電源電圧が 2.7V 以上なら書き込み可能で、シリアル・クロックは約 250kHz までになります。そこで、78K0/Kx2 を高速内蔵発振器の 8MHz で動作させ、CSI10 のクロックは 125kHz に設定します。クロック周波数の変動を考慮すると、250kHz のシリアル・クロックに対して、250kHz の設定は問題があります。
マイクロワイヤでは、データとクロック(SK)の位相の関係で 2 つの状態があります。そのため、CSI の通信フォーマットを送信時と受信時で切り替える必要があります。そこで、送信時には 78K0/Kx2 の CSI の転送タイプ 4、受信時にはタイプ 3 を使用します。なお、CSIM10 は MSB ファーストの送受信モードに設定して使用します。
次に、EEPROM のアクセス・タイミング例を示します。



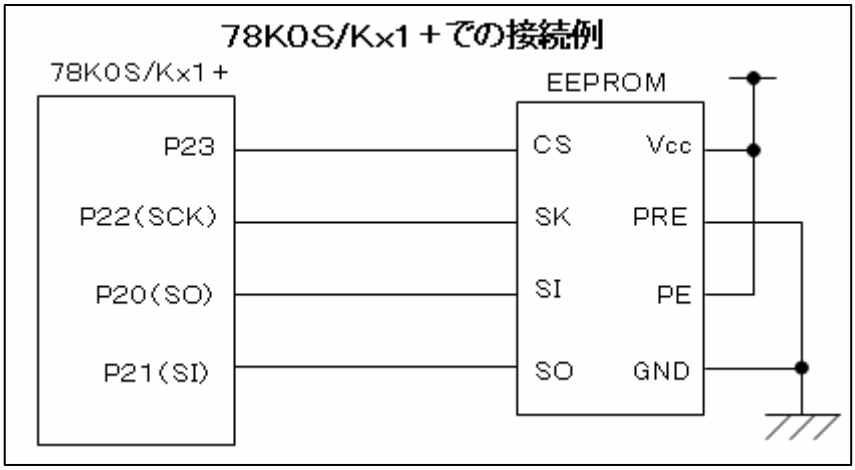
CSI10 での接続例を示します。CSI10 では P10～12 を使用するので、CS 信号には、連続した P13 を使用します。



具体的なプログラムは「CSI によるマイクロワイヤ EEPROM の制御プログラム例［78K0/Kx2］」をご参照ください。

［78K0S/Kx1+での制御］

78K0S/Kx1+には CSI 機能がないので、汎用ポートをソフトウエアで制御します。次に P2 での接続例を示します。

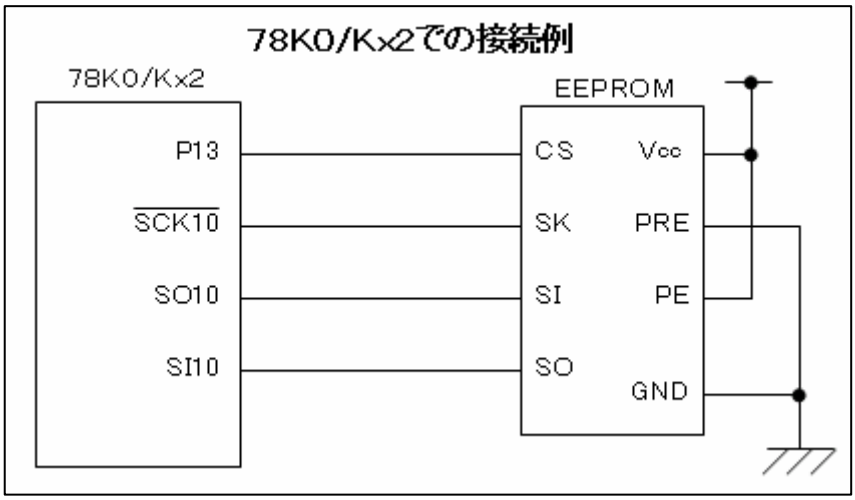


具体的なプログラムは「汎用ポートによるマイクロワイヤ EEPROM の制御プログラム例［78K0S/Kx1+］」をご参照ください。

CSI によるマイクロワイヤ EEPROM の制御プログラム例 [78K0/Kx2]

[はじめに]

マイクロワイヤの概要やマイクロワイヤ・インタフェースの EEPROM については「CSI によるマイクロワイヤ（Microwire）メモリの制御 [78K0/Kx2 など]」をご参照ください。ここでは、下記の接続例に対応した具体的な制御プログラムについて説明します。また、制御プログラムの使い方についても簡単に触れておきます。



[78K0/Kx2 の処理例]

78K0/Kx2 での EEPROM の制御処理は、以下の部分で構成します。マイクロワイヤ方式では、あまりスピードが要求されないので、制御プログラムはすべて C 言語で記述します。制御関数は引数として、読み出しや書き込みでは、次のパラメータを用います。たとえば、1ワード・データの読み出しでは、第 1 引数で読み出すアドレスを指定します。1ワード・データの書き込みでは、第 1 引数で書き込むアドレスを指定し、第 2 引数で書き込みデータを指定します。第 3 引数まで使用するのは、ブロック単位での読み書きの場合です。

第 1 引数：EEPROM のアドレス
第 2 引数：バッファ・メモリのアドレス／ワード書き込みのデータ
第 3 引数：転送データ数で 1～255 と 0（256 バイト）

(1) 初期化部

ポート、CSI10、割り込みの初期化を行います。ここでは 2.7VPOC を使用して、リセット解除後にそのまま動作させます。基本的にベクタ割り込みは使用しないので、割り込みは禁止状態のままとします。

(2) EEPROM アドレス長選択処理

この制御プログラムでは、6～13ビットのアドレス長の EEPROM に対応できるように、アドレス長を指定する処理を付け加えてあります。これで、指定したアドレス長に対応して、以下で使用するコマンドとアドレスは、必要なだけ頭に 0 を追加して、合計で 16 ビットになるようにします。

(3) 書き込み許可

EEPROM の CS 信号を ON にして、「WEN」コマンドを送ります。コマンドの転送完了後、CS 信号を OFF にします。このコマンドにより、EEPROM は書き込み可能状態になります。

(4) 書き込み禁止

EEPROM の CS 信号を ON にして、「WDS」コマンドを送ります。コマンドの転送完了後、CS 信号を OFF にします。このコマンドにより、EEPROM は書き込み禁止状態になります。

(5) ワード・データの読み出し

第 1 引数で示した EEPROM のアドレスから、1 ワードのデータを読み出し、戻り値を返します。

(6) ブロック・データ読み出し

第 1 引数で示した EEPROM のアドレスから、第 2 の引数で示されたアドレスのバッファに、第 3 の引数で示されたデータ数（ワード）のデータを読み出します。

(7) ワード・データ書き込み

第 1 引数で示した EEPROM のアドレスに、第 2 の引数で示されたワード・データを書き込みます。書き込み（転送）終了後、CS を OFF にすると、EEPROM は実際の書き込みを開始します。書き込みが完了するまでの待ち合わせはしません。次に EEPROM を操作する前に、「**(9)書き込み完了待ち**」を実行して、書き込みが完了したことを確認する必要があります。

(8) ブロック書き込み

第 1 引数で示した EEPROM のアドレスに、第 2 の引数で示されたアドレスのバッファのデータを第 3 の引数で示されたデータ数（ワード）だけ書き込みます。EEPROM は 1 ワードの書き込みにしか対応していないので、ここでは、ワード・データの書き込みを必要な回数だけ繰り返すことで、必要な書き込みを行います。ワードごとに書き込みコマンドを繰り返し発行する必要があるので、書き込み完了待ちまで処理しています。

(9) 書き込み完了待ち

EEPROM は書き込みデータを転送後、CS 信号を OFF にして、実際の書き込みを開始します。書き込みが完了したかどうかは、再度 CS 信号を ON にして、SO 端子の状態をチェックし、ハイ・レベルになることで確認できます。

(10) バイト・データ受信

CSI10 を使って、実際に EEPROM からバイト・データを受信します（内部で使うだけ）。

(11) バイト・データ送信

CSI10 を使って、EEPROM にコマンドやアドレス、バイト・データを送信します（内部で使うだけ）。

[C 言語でのプログラム例]

ここでは、C 言語でプログラムを作成した場合のプログラム例を示します。

初期化の前に、まず宣言と定義をしておきます。ここでは、関数のプロトタイプ宣言、EEPROM に対するコマンド（最も小さな 6 ビットのアドレスに対応した値）をディフォールト値として定義しています。また、第 1 引数のアドレス値を EEPROM のアドレスに丸めるための、マスク・データもビット長に応じて準備しています。

```
#pragma       SFR
#pragma       NOP

bit selrom(unsigned char leng);                          // アドレス長設定
void putmw(unsigned char data);                          // 1 文字送信
unsigned char getmw(void);                               // 1 文字受信
void writeenb(void);                                     // 書き込み許可
void writedis(void);                                     // 書き込み禁止
unsigned int read_mw(int reg_adr);                       // ワード読み出し
void read_mw_block(int reg_adr, unsigned int *data, unsigned char size);
                                                         // ブロック読み出し
void write_mw(int reg_adr, unsigned int data);           // ワード書き込み
void write_mw_block(int reg_adr, unsigned int *data, unsigned char size);
                                                         // ブロック書き込み
void waitwt(void);                                       // 書き込み完了待ち

/*----------------------------------------------------------*/
/*                                                          */
/*    MW 関連のコマンド定義                                 */
/*                                                          */
/*----------------------------------------------------------*/
#define       READC   0b0000000110000000      // データ読み出し
#define       WRITEC  0b0000000101000000      // データ書き込み
#define       WENC    0b0000000100110000      // 書き込み許可
#define       WDSC    0b0000000100000000      // 書き込み禁止

#define       LENGTH  6                       // アドレスのビット長
#define       Cssignal        P1.3            // CS 信号の定義

static        unsigned char adrlen = LENGTH;  // アドレス長の保存用
typedef unsigned int mask[8];                 // アドレス・マスク・データ
const mask mask1 = {0b0000000000111111,0b0000000001111111,
                    0b0000000011111111,0b0000000111111111,
                    0b0000001111111111,0b0000011111111111,
                    0b0000111111111111,0b0001111111111111};
```

(1) 初期化部

使用するポートと CSI の初期化を行う部分を hdwinit 関数として記述した例を以下に示します。実際の通信を行うときに、CSIM10 のビット 7(CSIE10)をセットしてください。

```
Void  hdwinit(void)
{
      P1              = 0b11110011;             // 出力ラッチの設定
      PM1             = 0b00000010;             // SI10/P11 以外は出力に
      CSIC10  = 0b00011101;                     //
//              |||||+++--------: シリアル・クロックは fPRS/64
//              |||++------------: タイプ 4
//              +++--------------: 0 に固定
      CSIM10  = 0b01000000;                     // MSB ファースト、送受信モード
//              ||+|++++--------; 0 に固定
//              || +------------; MSB ファースト
//              |+--------------; 送受信モード
//              +---------------; 動作禁止
}
```

(2) EEPROM アドレス長選択処理

EEPROM のアドレス長を指定します。アドレス長は引数で指定し、6～13 ビットの範囲とします（これで、スタート・ビット～アドレスを 16 ビットで処理）。この範囲以外は、エラーとして処理します。ここでは、単に変数に記録するだけです。

```
Bit   selrom(unsigned char length)
{
      if((length < 6) || (length > 13))       return 1;
      adrlen = length;
}
```

(3) 書き込み許可

CS 信号を ON にして、アドレス長に合わせて「WEN」コマンドをシフトし、頭の 0 の個数を調整して、EEPROM のコマンドを作成します。作成したコマンドを上位バイト、下位バイトの順で送信します。送信が完了したら、CS 信号を OFF にして、処理を終了します。

```
Void writeenb(void)
{
      unsigned int work;
      Cssignal = 1;                          // CS 信号をセット
      work = WENC << (adrlen-6);             // コマンドをデバイスに合わせる
      putmw(work / 0x100);                   // コマンドを転送
      putmw(work & 0x0ff);
      Cssignal = 0;                          // CS 信号をリセット
}
```

(4) 書き込み禁止

CS 信号を ON にして、アドレス長に合わせて「WDS」コマンドをシフトし、頭の 0 の個数を調整して、EEPROM のコマンドを作成します。作成したコマンドを上位バイト、下位バイトの順で送信します。送信が完了したら、CS 信号を OFF にして、処理を終了します。

```
Void writedis(void)
{
      unsigned int work;
      Cssignal = 1;                          // CS 信号をセット

      work = WDSC << (adrlen-6);             // コマンドをデバイスに合わせる
      putmw(work / 0x100);                   // コマンドを転送
      putmw(work & 0x0ff);

      Cssignal = 0;                          // CS 信号をリセット
}
```

(5) ワード・データの読み出し

引数として渡された EEPROM のアドレスから、必要な部分以外をマスクし、コマンドを必要なビットだけシフトさせて、コマンド＋アドレスを生成します。CS 信号を ON にし、スタート・ビット～アドレスを送信したら、いったん CSI を停止させします。動作モードをモード 3 に変更して再起動します。2 バイトのデータを読み出したら、CS 信号を OFF にして、通信モードを元に戻します。読み出した2バイトを結合して、ワードの戻り値とします。

```
Unsigned int read_mw(int reg_adr){
      unsigned int work;

      work = reg_adr & mask1[adrlen-6];       // アドレスの上位をマスク
      work |= (READC << (adrlen-6));          // コマンドを合成

      Cssignal = 1;                           // CS 信号をセット

      putmw(work / 0x100);                    // コマンドを転送
      putmw(work & 0x0ff);

      CSIE10 = 0;                             // CSI を動作停止
      CSIC10.3 = 0;                           // モード 3 を選択
      CSIE10 = 1;                             // CSI 動作許可
      work = getmw();                         // 上位データの読み込み
      work = work << 8;                       // 上位にシフト
      work |= getmw() ;                       // 下位データの読み出し

      Cssignal = 0;                           // CS 信号をリセット

      CSIE10 = 0;                             // CSI を動作停止
      CSIC10.3 = 1;                           // モード 4 を選択
      CSIE10 = 1;                             // CSI 動作許可

      return work;
}
```

(6) ブロック・データ読み出し

EEPROM とのやり取りは、ワード・データの読み出しと同じですが、連続に読み出すと、次のアドレスの内容が読み出せることを利用した処理です。読み出しデータ数の処理後、コマンド送信までの処理は同じです。その後、さらにデータを読み出して、それをバッファ・メモリに書き込む処理を指定された回数だけループします。

```
Void read_mw_block(int reg_adr, unsigned int *data, unsigned char size)
{
      unsigned int work;
      int count;                              // 受信データ数のカウンタ

      if(size){
            count = size;
      }else count = 0x100;                    // 入力が 0 なら 100H に設定

      work = reg_adr & mask1[adrlen-6];       // アドレスの上位をマスク
      work |= (READC << (adrlen-6));          // コマンドを合成

      Cssignal = 1;                           // CS 信号をセット

      putmw(work / 0x100);                    // コマンドを転送
      putmw(work & 0x0ff);

      CSIE10 = 0;                             // CSI を動作停止
      CSIC10.3 = 0;                           // モード 3 を選択
      CSIE10 = 1;                             // CSI 動作許可
```

```
        do{
                work = getmw();                 // 上位データの読み込み
                work = work << 8;               // 上位にシフト
                work |= getmw() ;               // 下位データの読み出し
                *data++ = work;                 // データの保存
        }while(--count);

        Cssignal = 0;                           // CS 信号をリセット

        CSIE10 = 0;                             // CSI を動作停止
        CSIC10.3 = 1;                           // モード 4 を選択
        CSIE10 = 1;                             // CSI 動作許可
}
```

(7) ワード・データ書き込み

コマンド転送までの処理は読み出しと同じです。第 2 の引数として渡されたワード・データを送信する部分が異なるだけです。なお、この関数の処理は書き込みの起動までで、書き込みの完了は待ちません。

```
Void write_mw(int reg_adr, unsigned int data)
{
        unsigned int work;

        work = reg_adr & mask1[adrlen-6];       // アドレスの上位をマスク
        work |= (WRITEC << (adrlen-6));         // コマンドを合成

        Cssignal = 1;                           // CS 信号をセット

        putmw(work / 0x100);                    // コマンドを転送
        putmw(work & 0x0ff);

        putmw(data / 0x100);                    // データを転送
        putmw(data & 0x0ff);

        Cssignal = 0;                           // CS 信号をリセット
}
```

(8) ブロック書き込み

1 ワード書き込みのループを回すので、コマンド処理もそれだけ繰り返す必要があります。そのたびに、コマンドのシフト処理やアドレスとの結合をすると、処理能力が低下するので、前処理として EEPROM に書き込むコマンドとアドレスを結合し、変数(command)にしておきます。ループでの EEPROM のアドレス更新は、この変数で処理します。そのため、EEPROM のアドレスがオーバフローするような場合には対応できないので、ご注意ください。

```
Void write_mw_block(int reg_adr, unsigned int *data, unsigned char size)
{
        unsigned int work, command;
        int count;                              // 送信データ数のカウンタ

        if(size){
                count = size;
        }else count = 0x100;                    // 入力が 0 なら 100H に設定

        command = reg_adr & mask1[adrlen-6];    // アドレスの上位をマスク
        command |= (WRITEC << (adrlen-6));      // コマンドを合成
```

```
        do{
                Cssignal = 1;                   // CS 信号をセット

                putmw(command / 0x100);         // コマンドを転送
                putmw(command & 0x0ff);

                work = *data++;                 // 送信データのセット

                putmw(work / 0x100);            // 上位バイト書き込み
                putmw(work & 0xff);             // 下位バイト書き込み

                Cssignal = 0;                   // CS 信号をリセット

                command++;                      // アドレス更新

                waitwt();                       // 書き込み完了を待つ

        }while(--count);
}
```

(9) 書き込み完了待ち

書き込み後に、この関数で EEPROM への書き込み完了を待つことができます。CS 信号を ON にしても、すぐには EEPROM から正しいステータスが出ないので、4 命令の NOP()で待ち合わせてから、ステータス確認をします。

```
Void waitwt(void)
{
        Cssignal = 1;                           // CS 信号をセット

        NOP();                                  // ステータス出力待ち
        NOP();
        NOP();
        NOP();

        while(P1.1 == 0);                       // 書き込み完了待ち

        Cssignal = 0;                           // CS 信号をリセット

}
```

(10) バイト・データ受信

CSI10 を直接制御して、バイト・データの受信処理をします。受信完了は、割り込み要求フラグをポーリングして確認します。割り込み要求フラグがセットされたらフラグをクリアし、受信データを読み出して戻ります。

```
unsigned char getmw(void)
{
        unsigned char work;
        SOTB10 = 0x00;                          // 受信のためのダミー・データ送信
        while(CSIIF10 == 0){                    // 転送完了待ち
                NOP();
        }
        CSIIF10 = 0;                            // 転送完了割り込みをクリア
        work = SIO10;                           // 受信データの読み出し
        return work;
}
```

(11) バイト・データ送信

CSI10 を直接制御して、バイト・データの送信処理をします。送信完了は、割り込み要求フラグをポーリングして確認します。割り込み要求フラグがセットされたら、フラグをクリアして戻ります。

```
void  putmw(unsigned char data)
{
      SOTB10 = data;                          // データをシリアルで送信
      while(CSIIF10 == 0){                    // 転送完了待ち
            NOP();
      }
      CSIIF10 = 0;                            // 転送完了割り込みをクリア
}
```

[プログラムの使用例]

ここでは、これまで説明してきたプログラムの使用方法について説明します。

(1) 宣言部

ここでは、これまで説明した関数を使用するために extern 宣言をしています。また、ハードウエアの初期化関数のプロトタイプ宣言も行っています。

```
#pragma       SFR
extern bit selrom(unsigned char spi_adr);     // ROM のアドレス長指定
extern void writeenb(void);                   // 書き込み許可
extern void writedis(void);                   // 書き込み禁止
extern unsigned int read_mw(int reg_adr);     // ワード・データ読み出し
extern void read_mw_block(int reg_adr, unsigned int *data, unsigned char size);
// ブロック読み出し
extern void write_mw(int reg_adr, unsigned int data);
// ワード・データの書き込み
extern void write_mw_block(int reg_adr, unsigned int *data, unsigned char size);
// ブロック書き込み
extern void waitwt(void);                     // 転送データの書き込み完了待ち
void hdwinit(void);
```

以下の部分は、関数を用いて読み書きを行う際のデータ・バッファと書き込むデータを定義しています。

```
unsigned int  eep_read_buf1[16];              // read buffer
unsigned int  eep_read_buf2[16];              // read buffer
unsigned int  eep_read_buf3[16];              // read buffer
unsigned int  eep_write_buf[16];              // write buffer
typedef unsigned int data4[4];
const data4 txdata1 = {0x1234,0x5678,0x9ABC,0xDEF0};
const data4 txdata2 = {0x55AA,0x33CC,0xf00f,0xff00};
```

(2) 読み出し処理

ここでは、これまで説明した関数を使用して、実際に EEPROM からデータを読み出します。最初に、アドレス幅を 8 ビットに指定します。なお、書き込みはしないので、書き込みは禁止にして読み出します。この例では、0x33 番地と 0xAA 番地からワード・データを読み出し、0xCC 番地から 8 ワードのデータをバッファ・メモリに読み出す処理を行っています。

```
void rxtest(void)
{
      unsigned int work;
      selrom(8);                              // アドレス幅を設定
      writedis();                             // 書き込み禁止
      work = read_mw(0x33);                   // 0x33 からリード
      work = read_mw(0xAA);                   // 0xAA からリード
      read_mw_block(0xcc, eep_read_buf1, 8);  // ブロック・データ・リード
//  第 1 引数で EEPROM アドレス（0xcc）を指定、
//  第 2 引数でリード・バッファ・メモリのアドレス（eep_read_buf1）を指定
//  第 3 引数でデータ数（8 ワード）を指定
}
```

(3) 書き込み処理

ここでは、これまで説明した関数を使用して、実際に EEPROM にデータを書き込みます。最初に、アドレス幅を 8 ビットに指定し、書き込みを許可状態にしてから、書き込みを行います。その後、ワード・データの書き込みでは、実際の書き込みが完了するまでの待ち合わせをしますが、完了待ちの前に、他の処理を入れることも可能です。

```
void  txtest(void)
{
      selrom(8);                              // アドレス幅を変更
      writeenb();                             // 書き込み許可
      write_mw(0xaa, 0x1234);                 // ワード・データ書き込み

//  ここに他の処理を入れて構いません。
      waitwt();                               // 書き込み完了待ち
      write_mw(0xaa, 0x55aa);

//  ここに他の処理を入れて構いません。
      waitwt();
      write_mw_block(0xaa, txdata1, 4);       // ブロック書き込み
}
```

汎用ポートによるマイクロワイヤ EEPROM の制御プログラム例 [78K0S/Kx1+]

[はじめに]

マイクロワイヤの概要やマイクロワイヤ・インタフェースの EEPROM については「CSI によるマイクロワイヤ（Microwire）メモリの制御 [78K0/Kx2 など]」をご参照ください。78K0S/Kx1+には、CSI 機能がないので、汎用ポートで制御する必要があります。ここでは、下記の接続例に対応した具体的な制御プログラムについて説明します。また、制御プログラムの使い方についても、簡単に触れておきます。



[78K0S/Kx1+の処理例]

78K0S/Kx1+での EEPROM の制御処理は、以下の部分から構成します。78K0S/Kx1+は、マイクロワイヤとのインタフェースに使用する CSI を内蔵していないので、汎用ポートをソフトウエアで制御して、必要なインタフェースを行います。プログラムでの処理は時間がかかるので、シリアルとパラレルのデータ変換処理と、入出力処理は、アセンブラでも記述します。この処理を C 言語記述のプログラムから利用する例と、すべてをアセンブラ記述した例を示します。

C 言語記述の例では、78K0/Kx2 の CSI を用いた制御例にできるだけ合わせるために、シリアルの処理は 8 ビット単位で行います。アセンブラ記述の例では、EEPROM のアドレスに合わせたビット長での処理としています。ただし、すべてアセンブラで記述した制御プログラムも、C 言語記述のプログラムから使用できるようにします。

制御関数は引数として、読み出しや書き込みでは、次のパラメータを用います。たとえば、1 ワード・データの読み出しでは、第 1 引数で読み出すアドレスを指定します。1 ワード・データの書き込みでは、第 1 引数で書き込むアドレスを指定し、第 2 引数で書き込みデータを指定します。第 3 引数まで使用するのは、ブロック単位での読み書きの場合です。引数の定義と使用できる機能は、できるだけ SPI の制御と合わせてあります。

第 1 引数：EEPROM のアドレス

第 2 引数：バッファ・メモリのアドレス／ワード書き込みのデータ

第 3 引数：転送データ数で 1～255 と 0（256 バイト）

なお、内蔵高速発振クロックを使用し、ウォッチドッグ・タイマは使用しないものとして、オプション・バイトは、次のように宣言しておくものとします。

```
    @@OPTB  CSEG    AT      0080H
    DB      10011100b       ; オプション・バイト
;             || |||+-----:低速クロック停止可
;             || |++------:高速内蔵発振クロック使用
;             || +--------:RESET 端子を使用
;             ++----------:発振安定時間は最短
;
    DB      11111111b       ; プロテクト・バイト
    END
```

(1) 初期化部

ポートの初期化を行います。さらに、8MHz で使用するために LVI を用いて、電源電圧が 4.0V 以上になるまで待ちます。基本的にベクタ割り込みは使用しないので、割り込みは禁止状態のままとします。また、ウォッチドッグ・タイマも停止します。

(2) EEPROM アドレス長選択処理

この制御プログラムでは、6 ビット～13 ビットのアドレス長の EEPROM に対応できるように、アドレス長を指定する処理を付け加えてあります。これで指定したアドレス長に対応して、使用するコマンドとアドレスは、必要なだけ頭に 0 を追加して、合計で 16 ビットになるようにします。アセンブラ記述の例では、この処理は行いません。

(3) 書き込み許可

EEPROM の CS 信号を ON にして、「WEN」コマンドを送ります。コマンドの転送完了後、CS 信号を OFF にします。このコマンドにより、EEPROM は書き込み可能状態になります。

(4) 書き込み禁止

EEPROM の CS 信号を ON にして、「WDS」コマンドを送ります。コマンドの転送完了後、CS 信号を OFF にします。このコマンドにより、EEPROM は書き込み禁止状態になります。

(5) ワード・データの読み出し

第 1 引数で示した EEPROM のアドレスから、1 ワードのデータを読み出し、戻り値を返します。
コマンドとアドレスの転送後、SK とデータの位相関係が変化することに対しては、アドレス転送後に SK を立ち下げ、その後のバイト・データ受信処理で、SK 立ち下がりでデータ読み込んで対応します。

(6) ブロック・データ読み出し

第 1 引数で示した EEPROM のアドレスから、第 2 の引数で示されたアドレスのバッファに、第 3 の引数で示されたデータ数（ワード）のデータを読み出します。

(7) ワード・データ書き込み

第 1 引数で示した EEPROM のアドレスに、第 2 の引数で示されたワード・データを書き込みます。書き込み（転送）終了後、CS を OFF にすると、EEPROM は実際の書き込みを開始します。書き込みが完了するまでの待ち合わせはしません。次に EEPROM を操作する前に、「**(9)書き込み完了待ち**」を実行して、書き込みが完了したことを確認する必要があります。

(8) ブロック書き込み

第 1 引数で示した EEPROM のアドレスに、第 2 の引数で示されたアドレスのバッファのデータを第 3 の引数で示されたデータ数（ワード）だけ書き込みます。EEPROM は 1 ワードの書き込みにしか対応していないので、ここでは、ワード・データの書き込みを必要な回数だけ繰り返すことで、必要な書き込みを行います。ワードごとに書き込みコマンドを繰り返し発行する必要があるので、書き込み完了待ちまで処理しています。

(9) 書き込み完了待ち

EEPROM は書き込みデータを転送後、CS 信号を OFF にして、実際の書き込みを開始します。書き込みが完了したかどうかは、再度 CS 信号を ON にして、SO 端子の状態をチェックし、ハイ・レベルになることで確認できます。

(10) バイト・データ受信

汎用ポートを使って、EEPROM からバイト・データを受信します。C 言語記述でも、ここはアセンブラで記述します。

(11) バイト・データ送信

汎用ポートを使って、EEPROM にコマンドやアドレス、バイト・データを送信します。C 言語記述の場合でも、ここはアセンブラで記述します。

[C 言語でのプログラム例]

ここでは、C 言語でプログラムを作成した場合のプログラム例を示します。EEPROM へのアクセスで、あまり速度に制約がない場合に参考にしてください。
初期化の前に、まず宣言と定義をしておきます。ここでは、関数のプロトタイプ宣言、EEPROM に対するコマンド

（最も小さな 6 ビットのアドレスに対応した値）をディフォールト値として定義しています。また、第 1 引数のアドレス値を EEPROM のアドレスに丸めるための、マスク・データもビット長に応じて準備しています。

```
#pragma       SFR
#pragma       NOP
extern void _putmw(unsigned char data);       // 1 文字送信
extern unsigned char _getmw(void);            // 1 文字受信
bit selrom(unsigned char leng);               // アドレス長設定
void writeenb(void);                          // 書き込み許可
void writedis(void);                          // 書き込み禁止
unsigned int read_mw(int reg_adr);
void read_mw_block(int reg_adr, unsigned int *data, unsigned char size);
void write_mw(int reg_adr, unsigned int data);
void write_mw_block(int reg_adr, unsigned int *data, unsigned char size);
void waitwt(void);
/*-------------------------------------------------*/
/*                                                 */
/*    MW 関連のコマンド定義                        */
/*                                                 */
/*-------------------------------------------------*/
#define       READC   0b0000000110000000      // データ読み出し
#define       WRITEC  0b0000000101000000      // データ書き込み
#define       WENC    0b0000000100110000      // 書き込み許可
#define       WDSC    0b0000000100000000      // 書き込み禁止
#define       LENGTH  6                       // アドレスのビット長
#define       CSsignal        P2.3            // CS 信号の定義
#define       SKsignal        P2.2            // SK 信号の定義
#define       SIsignal        P2.1            // SI 信号の定義
static        unsigned char adrlen = LENGTH;  // アドレス長の保存用
typedef unsigned int mask[8];                 //  アドレス・マスク・データ
const mask mask1 =    {0b0000000000111111,0b0000000001111111,
                       0b0000000011111111,0b0000000111111111,
                       0b0000001111111111,0b0000011111111111,
                       0b0000111111111111,0b0001111111111111};
```

(1) 初期化部

使用ポートの初期化や電源の立ち上がり待ち、クロック設定を hdwinit 関数として記述した例を次に示します。

```
void  hdwinit(void)
{
      int work1;
      P2      = 0b11110011;                   // 出力ラッチの設定
      PM2     = 0b00000010;
      LVIS    = 0x00;                         // select VLI=4V
      LVIM    = 0b10000000;                   // Start VLI detection
      PCC     = 0x00;                         // CPU clock is fx/4
      WDTM    = 0b01110111;                   // Stop WDT
      LSRSTOP = 1;                            // Stop Low Speed OSC
//    wait for 0.17ms(=0.2ms-(32+30)*52/1000)
      for (work1 = 0; work1 < 10; work1++){
            NOP();
      }
      while ( LVIF ){
            NOP();
      }                                       // wait VDD > 4V
      PPCC = 0;                               // CPU clock is fx=fR
}
```

(2) EEPROM アドレス長選択処理

EEPROM のアドレス長を指定します。アドレス長は引数で指定し、6～13 ビットの範囲とします（これで、スタート・ビット～アドレスを 16 ビットで処理）。この範囲以外はエラーします。ここでは、単に変数に記録するだけです。

```
Bit   selrom(unsigned char length)
{
      if((length < 6) || (length > 13))       return 1;
      adrlen = length;
}
```

(3) 書き込み許可

CS 信号を ON にして、アドレス長に合わせて「WEN」コマンドをシフトし、頭の 0 の個数を調整して、EEPROM のコマンドを作成します。作成したコマンドを上位バイト、下位バイトの順で送信します。送信が完了したら、CS 信号を OFF にして、処理を終了します。

```
Void writeenb(void)
{
      unsigned int work;
      Cssignal = 1;                           // CS 信号をセット
      work = WENC << (adrlen-6);              // コマンドをデバイスに合わせる
      _putmw(work / 0x100);                   // コマンドを転送
      _putmw(work & 0x0ff);
      Cssignal = 0;                           // CS 信号をリセット
}
```

(4) 書き込み禁止

CS 信号を ON にして、アドレス長に合わせて「WDS」コマンドをシフトし、頭の 0 の個数を調整して、EEPROM のコマンドを作成します。作成したコマンドを上位バイト、下位バイトの順で送信します。送信が完了したら、CS 信号を OFF にして処理を終了します。

```
Void writedis(void)
{
      unsigned int work;
      Cssignal = 1;                           // CS 信号をセット
      work = WDSC << (adrlen-6);              // コマンドをデバイスに合わせる
      _putmw(work / 0x100);                   // コマンドを転送
      _putmw(work & 0x0ff);
      Cssignal = 0;                           // CS 信号をリセット
}
```

(5) ワード・データの読み出し

引数として渡された EEPROM のアドレスから、必要な部分以外をマスクし、コマンドを必要なビットだけシフトさせて、コマンド＋アドレスを生成します。CS 信号を ON にし、スタート・ビット～アドレスを送信したら、SK 信号をロウにします。2 バイトのデータを読み出したら、CS 信号を OFF にして、通信モードを元に戻します。読み出した 2 バイトを結合して、ワードの戻り値とします。

```
unsigned int read_mw(int reg_adr){
      unsigned int work;
      work = reg_adr & mask1[adrlen-6];       // アドレスの上位をマスク
      work |= (READC << (adrlen-6));          // コマンドを合成
      CSsignal = 1;                           // CS 信号をセット
      _putmw(work / 0x100);                   // コマンドを転送
      _putmw(work & 0x0ff);
```

```
        SKsignal = 0;                           // SK 信号の位相を変更する
        work = _getmw();                        // 上位データの読み込み
        work = work << 8;                       // 上位にシフト
        work |= _getmw() ;                      // 下位データの読み出し
        CSsignal = 0;                           // CS 信号をリセット
        SKsignal = 1;                           // SK 信号の位相を変更する
        return work;
}
```

(6) ブロック・データ読み出し

EEPROM とのやり取りは、ワード・データの読み出しと同じですが、連続に読み出すと、次のアドレスの内容が読み出せることを利用した処理です。読み出しデータ数の処理後、コマンド送信までの処理は同じです。その後、さらにデータを読み出して、それをバッファ・メモリに書き込む処理を指定された回数だけループします。

```
void read_mw_block(int reg_adr, unsigned int *data, unsigned char size)
{
        unsigned int work;
        int count;                              // 受信データ数のカウンタ
        if(size){
                count = size;
        }else count = 0x100;                    // 入力が 0 なら 100H に設定
        work = reg_adr & mask1[adrlen-6];       // アドレスの上位をマスク
        work |= (READC << (adrlen-6));          // コマンドを合成
        CSsignal = 1;                           // CS 信号をセット
        _putmw(work / 0x100);                   // コマンドを転送
        _putmw(work & 0x0ff);
        SKsignal = 0;                           // SK 信号の位相を変更
        do{
                work = _getmw();                // 上位データの読み込み
                work = work << 8;               // 上位にシフト
                work |= _getmw() ;              // 下位データの読み出し
                *data++ = work;                 // データの保存
        }while(--count);
        CSsignal = 0;                           // CS 信号をリセット
        SKsignal = 1;                           // SK 信号の位相を変更
}
```

(7) ワード・データ書き込み

コマンド転送までの処理は読み出しと同じです。第 2 の引数として渡されたワード・データを送信する部分が異なるだけです。なお、この関数の処理は書き込みの起動までで、書き込みの完了は待ちません。

```
void write_mw(int reg_adr, unsigned int data)
{
        unsigned int work;
        work = reg_adr & mask1[adrlen-6];       // アドレスの上位をマスク
        work |= (WRITEC << (adrlen-6));         // コマンドを合成
        CSsignal = 1;                           // CS 信号をセット
        _putmw(work / 0x100);                   // コマンドを転送
        _putmw(work & 0x0ff);
        _putmw(data / 0x100);                   // データを転送
        _putmw(data & 0x0ff);
        CSsignal = 0;                           // CS 信号をリセット
}
```

(8) ブロック書き込み

1 ワード書き込みのループを回すので、コマンド処理もそれだけ繰り返す必要があります。そのたびに、コマンドのシフト処理やアドレスとの結合をすると、処理能力が低下するので、前処理として EEPROM に書き込むコマンドとアドレスを結合し、変数(command)にしておきます。ループでの EEPROM のアドレス更新は、この変数で処理します。そのため、EEPROM のアドレスがオーバフローするような場合には対応できないので、ご注意ください。

```
void write_mw_block(int reg_adr, unsigned int *data, unsigned char size)
{
      unsigned int work, command;
      int count;                              // 送信データ数のカウンタ
      if(size){
            count = size;
      }else count = 0x100;                    // 入力が 0 なら 100H に設定
      command = reg_adr & mask1[adrlen-6];    // アドレスの上位をマスク
      command |= (WRITEC << (adrlen-6));      // コマンドを合成
      do{
            CSsignal = 1;                     // CS 信号をセット
            _putmw(command / 0x100);          // コマンドを転送
            _putmw(command & 0x0ff);
            work = *data++;                   // 送信データのセット
            _putmw(work / 0x100);             // 上位バイト書き込み
            _putmw(work & 0xff);              // 下位バイト書き込み
            CSsignal = 0;                     // CS 信号をリセット
            command++;                        // アドレス更新
            waitwt();                         // 書き込み完了を待つ
            }while(--count);
}
```

(9) 書き込み完了待ち

書き込み後に、この関数で EEPROM への書き込み完了を待つことができます。CS 信号を ON にしても、すぐには EEPROM から正しいステータスが出ないので、4 命令の NOP()で待ち合わせてから、ステータス確認をします。

```
Void waitwt(void)
{
      Cssignal = 1;                           // CS 信号をセット
      NOP();                                  // ステータス出力待ち
      NOP();
      NOP();
      NOP();
      while(SIsignal == 0);                   // 書き込み完了待ち
      Cssignal = 0;                           // CS 信号をリセット
}
```

(10) バイト・データ受信

汎用ポートを制御し、てバイト・データの受信処理を行います。この処理は、アセンブラ記述となります。

```
;
;              シリアル受信処理
; ＜機能＞
;  8 ビットのデータをシリアルで転送（受信）する
;  転送が完了したら受信データを持って戻る
;    SK 信号のデューティを変えることで、周期を守りながら全体の転送レートを
;    下げすぎないようにしています
;    汎用ポート(P2.1)のデータをキャリー・フラグを介して取り込みます
```

```
;     新しいデータの取り込みとループ・カウントを 1 つの命令で兼用するため
;     に A レジスタに設定する初期値を工夫しています
;   <入力パラメータ（引数）>   なし
;   <戻り値>
;       受信データ
;unsigned char _getmw(void)
        public  __getmw

__getmw:
        MOV     A,#11111110b    ; 4: 初期値を設定
RXLOOP:
        SET1    P2.2            ; 6: SK 信号を立ち上げ
        NOP                     ; 2:
        NOP                     ; 2:
        SET1    CY              ; 2: 入力ディフォールト値設定
        CLR1    P2.2            ; 6: SK 信号を立ち下げ
        BT      P2.1,$RXNEXT    ;10: 入力をチェック
        CLR1    CY              ; 2: データ 0 用
RXNEXT:
        ROLC    A,1             ; 2: 入力ビットを組み入れ
        BC      $RXLOOP         ; 6: 8 回ループ
        MOV     C,A             ; 戻り値に設定
        RET
```

(11) バイト・データ送信

汎用ポートを直接制御して、バイト・データの送信処理をします。

```
;
; <機能>
;  引数で渡された 8 ビットのデータをシリアルで転送する
;  転送が完了したら戻る
;  処理は 250kbps 以下にするため、SK の制御間隔を 2μs 以上
;  確保させる
;  <入力パラメータ（引数）>
;      送信データ
;  <戻り値> なし
;void _putmw(unsigned char)
        public  __putmw

__putmw:
        PUSH    BC
        MOV     B,#8            ; ビット・カウンタ設定
TXLOOP:
        MOV     A,X             ; 4: 出力データを持ってくる
        SET1    P2.2            ; 6: SK 信号を立ち上げ
        ROL     A,1             ; 2: 出力ビットを LSB にセット
        MOV     X,A             ; 4:
        AND     A,#1            ; 4: その他のビットをマスク
        OR      A,#8            ; 4: CS ビットをセット
        MOV     P2,A            ; 4: SO 信号の出力と SK 信号の立ち下げ
        NOP                     ; 2:
        DBNZ    B,$TXLOOP       ; 6: 8 回繰り返し
        POP     BC              ; 6:
        SET1    P2.2            ; 6: SK 信号を立ち上げ
        RET
```

[プログラムの使用例 1]

ここでは、これまで説明してきたプログラムの使用方法を説明します。

(1) 宣言部

ここでは、これまで説明した関数を使用するために extern 宣言をしています。また、ハードウエアの初期化関数のプロトタイプ宣言も行っています。

```
#pragma       SFR
extern bit selrom(unsigned char leng);          // ROM のアドレス長指定
extern void writeenb(void);                     // 書き込み許可
extern void writedis(void);                     // 書き込み禁止
extern unsigned int read_mw(int reg_adr);       // ワード・データ読み出し
extern void read_mw_block(int reg_adr, unsigned int *data, unsigned char size);
// ブロック読み出し
extern void write_mw(int reg_adr, unsigned int data);
// ワード・データの書き込み
extern void write_mw_block(int reg_adr, unsigned int *data, unsigned char size);
// ブロック書き込み
extern void waitwt(void);                       // 転送データの書き込み完了待ち
void hdwinit(void);
```

以下の部分は、関数を用いて読み書きを行う際のデータ・バッファと書き込むデータを定義しています。

```
unsigned int  eep_read_buf1[8];       // read buffer
typedef unsigned int data4[4];
const data4 txdata1 = {0x1234,0x5678,0x9ABC,0xDEF0};
```

(2) 読み出し処理

ここでは、これまで説明した関数を使用して、実際に EEPROM からデータを読み出します。最初に、アドレス幅を 8 ビットに指定します。なお、書き込みはしないので、書き込みは禁止にして読み出します。この例では、0x33 番地と 0xAA 番地からワード･データを読み出し、0xCC 番地から 8 ワードのデータをバッファ･メモリに読み出す処理を行っています。

```
void rxtest(void)
{
      unsigned int work;
      selrom(8);                                // アドレス幅を設定
      writedis();                               // 書き込み禁止
      work = read_mw(0x33);                     // 0x33 からリード
      work = read_mw(0xAA);                     // 0xAA からリード
      read_mw_block(0xcc, eep_read_buf1, 4);    // ブロック・データ・リード
//  第 1 引数で EEPROM アドレス（0xcc）を指定、
//  第 2 引数でリード・バッファ・メモリのアドレス（eep_read_buf1）を指定
//  第 3 引数でデータ数（4 ワード）を指定
}
```

(3) 書き込み処理

ここでは、これまで説明した関数を使用して、実際に EEPROM にデータを書き込みます。最初に、アドレス幅を 8 ビットに指定し、書き込みを許可状態にしてから、書き込みを行います。その後、ワード･データの書き込みでは、実際の書き込みが完了するまでの待ち合わせをしますが、完了待ちの前に、他の処理を入れることも可能です。

```
void  txtest(void)
{
      selrom(8);                              // アドレス幅を変更
      writeenb();                             // 書き込み許可
      write_mw(0xaa, 0x1234);                 // ワード・データ書き込み

//  ここに他の処理を入れて構いません。
      waitwt();                               // 書き込み完了待ち
      write_mw(0xaa, 0x55aa);

//  ここに他の処理を入れて構いません。
      waitwt();
      write_mw_block(0xaa, txdata1, 4);       // ブロック書き込み
}
```

[アセンブラでのプログラム例]

ここでは、アセンブラ記述でのプログラム例を示します。

初期化の前に、まず各種宣言と作業領域の定義をしておきます。プログラムの大きさを小さくするため、ここではよく使用する 3 つのサブルーチンを CALLT 命令で呼べるように、CALLT テーブルを定義します。

また、マクロを定義して、意味のある名前を使用して使えるようにしています。

汎用ポートの制御もマクロを使用して、たとえば CS 信号を ON にする処理は、CSsignalON と直感的に処理が分かる記述ができるようにしています。

```
PUBLIC __selrom                         ; ROM のアドレス長を指定
PUBLIC __writeenb                       ; 書き込みを許可
PUBLIC __writedis                       ; 書き込みを禁止
PUBLIC __read_mw                        ; 1 ワード読み出し
PUBLIC __read_mw_block                  ; ブロック読み出し
PUBLIC __write_mw                       ; 1 ワード書き込み
PUBLIC __write_mw_block                 ; ブロック書き込み
PUBLIC __waitw                          ; 書き込み完了待ち
;
;     作業領域
;
WDATA DSEG    SADDR
ADRLEN        DS      1                 ; アドレス長パラメータ
COMMAND       DS      2                 ; コマンド生成用ワーク
;
;     シリアル処理マクロ定義
;
CSsignalON    macro
      SET1    P2.3
      endm
CSsignalOFF   macro
      CLR1    P2.3
      endm
SKsignalON    mcro
      SET1    P2.2
      endm
SKsignalOFF   mcro
      CLR1    P2.2
      endm
```

```
putmw macro
      CALLT   [48H]
      endm
getmw macro
      CALLT   [4AH]
      endm
putbits       macro
      CALLT   [4CH]
      Endm
RTX   CSEG    AT      48H
      DW      PUTMWS
      DW      GETMWS
      DW      TXBIT
```

(1) 初期化部

使用するポートの初期化や電源の立ち上がり待ち、クロック設定などを行う部分です。オプション・バイトの設定で低速発振クロックを停止可能に設定してあるので、ウォッチドッグ・タイマは停止させておきます。

```
;
;     ポートのイニシャライズ
;
      MOV      P2,#11110000b          ; 出力ラッチの設定
      MOV     PM2,#11110010b          ; P20、P22、P23 は出力に
;                   |||+-----: シリアル出力 (SO)
;                   ||+------: シリアル入力 (SI)
;                   |+-------: シリアル・クロック (SK)
;                   +--------: チップ・セレクト (CS)
;
;     電源の立ち上がり待ち (4.0V 以上になるのを待つ)
;
POWERON:
            DI
            MOV     LVIS,#00000000b ; VLI=4.0V
            MOV     LVIM,#10000000b ; 低電圧検出開始
            MOV     PCC,#00         ; CPU clock is fx/4
            MOV     WDTM,#01110111b ; WDT を停止
            SET1    LSRSTOP         ; 低速発振器停止
            MOV     A,#50
PONLOOP:
            DEC     A
            BNZ     $PONLOOP        ; 0.2ms 待つ
PONLOOP2:
            BT      LVIF,$PONLOOP2  ; 電圧が 4.0V 以上まで待つ
            MOV     PPCC,#0         ; CPU clock is fx=fR
```

(2) EEPROM アドレス長選択処理

アドレス長を 6～13 ビットの範囲で設定します。指定値に問題がなければ、指定されたビット長に対応したパラメータを保存します。指定値が範囲外の場合、エラーとしてキャリー・フラグ(戻り値)をセットして戻ります。

```
;/*-------------------------------------------------------------*/
;/*                                                             */
;/*          アドレス長設定                                     */
;/*  <機能>                                                     */
;/*  引数で指定された値をアドレス長として設定する               */
;/*    EEPROM に対するコマンドとアドレスは合わせて 16 ビットで   */
;/*    処理します                                               */
```

```
;/*     ここで設定した値を元にしてアドレスの有効桁を抽出したり、*/
;/*     コマンドのビット位置を調整します                        */
;/*  <入力パラメータ（引数）>        アドレス長                 */
;/*  <戻り値>                                                   */
;/*   エラーをキャリー・フラグにセットして戻る                  */
;/*-------------------------------------------------------------*/
;bit _selrom(unsigned char length)

__selrom:
SELROM:
      MOV     A,X             ; 6～13
      ADD     A,#(255-13)     ; 249～255
      SUB     A,#(255-7)      ; 0～7
      BC      $SELROMEXIT
      INC     A               ; 1～8
      MOV     ADRLEN,A        ; パラメータを保存
SELROMEXIT:
      RET
```

(3) 書き込み許可

CS 信号を ON して「WEN」コマンドを転送し、指定されたアドレス長に対応した数（アドレス長－2）の 0 を送ります。送信が完了したら、CS 信号を OFF にして、処理を終了します。

```
;/*-------------------------------------------------------*/
;/*                                                       */
;/*          書き込み許可コマンド発行                     */
;/*  <機能>                                               */
;/*  EEPROM を書き込み許可に設定する                      */
;/*  <入力パラメータ（引数）>  なし                       */
;/*  <戻り値>  なし                                       */
;/*                                                       */
;/*-------------------------------------------------------*/
;void _writeenb(void)

__writeenb:
WRITEENB:
      CSsignalON                ; CS 信号をオン
      MOV     A,#01001100b      ; 書き込み許可コマンド
WRITENEXT:
      MOV     B,#6              ; 書き込みは 6 ビット分
      putbits                   ; コマンドの書き込み
      MOV     A,ADRLEN
      ADD     A,#3              ; 残りアドレス数を計算
      MOV     B,A
      MOV     A,#0              ; 残りアドレスは 0 に
      putbits                   ; アドレスの残りに 0 を出力
      SKsignalOFF               ; SK 信号をオフ
      CSsignalOFF               ; CS 信号をオフ
      RET
```

(4) 書き込み禁止

CS 信号を ON にして「WDS」コマンドを転送し、指定されたアドレス長－2 の 0 を送ります。送信が完了したら CS 信号を OFF にして、処理を終了します。コマンド以外は、書き込み許可と同じです。

```
;/*-------------------------------------------------------*/
;/*                                                       */
;/*          書き込み禁止コマンド発行                     */
;/*  ＜機能＞                                             */
;/*  EEPROM を書き込み禁止に設定する                      */
;/*  ＜入力パラメータ（引数）＞  なし                     */
;/*  ＜戻り値＞  なし                                     */
;/*                                                       */
;/*-------------------------------------------------------*/
;void _writedis(void)
__writedis:
WRITEDIS:
      CSsignalON                    ; CS 信号をオン
      MOV     A,#01000000b          ; 書き込み許可コマンド
      BR      $WRITENEXT            ; コマンド書き込みへ
```

(5) ワード・データの読み出し

CS 信号を ON にし、コマンドに続いて、引数として渡された EEPROM のアドレスを必要なビット数だけ転送します。その後、SK 信号をロウにして、2 バイトのデータを読み出し、CS 信号を OFF にします。読み出したデータは、戻り値として BC レジスタに設定して戻ります。

```
;/*-------------------------------------------------------*/
;/*                                                       */
;/*          データの読み出し 1                           */
;/*  ＜機能＞                                             */
;/*  EEPROM の引数で指定したアドレスから 1 ワードの       */
;/*  データを読み出す                                     */
;/*  ＜入力パラメータ（引数）＞                           */
;/*  AX :EEPROM 内のアドレス                              */
;/*  ＜戻り値＞                                           */
;/*  BC :読み出したデータ                                 */
;/*                                                       */
;/*-------------------------------------------------------*/
;unsigned int _read_mw(int reg_adr)

__read_mw:
READ_MW:
      CALL    !RDCMD                ; コマンドとアドレス送信
      SKsignalOFF                   ; SK 信号を立ち下げ
      SOsignalOFF                   ; SO 信号をロウ・レベルに設定
      getmw                         ; 上位バイト受信
      MOV     B,A
      getmw                         ; 下位バイト受信
      MOV     C,A
      CSsignalOFF                   ; CS 信号をオフ
      RET
```

(6) ブロック・データ読み出し

EEPROM とのやり取りは、ワード・データの読み出しと同じですが、連続に読み出すと、次のアドレスの内容が読み出せます。読み出しデータ数の処理後、コマンド送信までの処理は同じです。その後、さらにデータを読み出し、それをバッファ・メモリに書き込む処理を指定回数だけループします。引数がスタックを用いて渡されるので、

コマンド転送後に、そこから必要なパラメータを抽出します。具体的な処理は、ブロック書き込みと共用したサブルーチン（GETPARA）で行います。データの読み出しは、上位バイト＞下位バイトの順なので、バッファ・メモリに格納する際は、格納アドレスに注意が必要です（「エンディアン」をご参照ください）。

```
;/*-------------------------------------------------------*/
;/*                                                       */
;/*          データの読み出し（ブロック）                 */
;/*  ＜機能＞                                             */
;/*  引数として渡されたデータ数だけ EEPROM から           */
;/*  データを読み出す                                     */
;/*  ＜入力パラメータ（引数）＞                           */
;/* AX     : 第 1 の引数 :EEPROM 内のアドレス             */
;/* SP+2,3: 第 2 の引数 :バッファ・メモリのアドレス       */
;/* SP+4   : 第 3 の引数 :読み出しデータ数（ワード単位）  */
;/*  ＜戻り値＞          なし                             */
;/*                                                       */
;/*-------------------------------------------------------*/
;void _read_mw_block(int reg_adr, unsigned int *data, unsigned char size)

__read_mw_block:
READ_MWB:
      CALL    !RDCMD                  ; コマンドとアドレス送信
      SKsignalOFF                     ; SK 信号を立ち下げ
      PUSH    HL                      ; レジスタをセーブ
      PUSH    AX
;
;  このサブルーチンは C（ノーマル・モード）からの関数コールに対応
;  そのため、この段階でのスタックのデータは次のようになっている
;
;     sp   : X レジスタ（第 1 引数）    : EEPROM アドレス（下位）
;     sp+1 : A レジスタ（第 1 引数）    : EEPROM アドレス（上位）
;     sp+2 : L レジスタ                 : 未使用（レジスタのセーブだけ）
;     sp+3 : H レジスタ                 : 未使用（レジスタのセーブだけ）
;     sp+4 : PC の下位                  : 戻りアドレス下位
;     sp+5 : PC の上位                  : 戻りアドレス上位
;     sp+6 : 第 2 引数の下位            : バッファ・メモリのアドレス（下位）
;     sp+7 : 第 2 引数の上位            : バッファ・メモリのアドレス（下位）
;     sp+8 : 第 3 引数の下位            : 読み出しデータ数
;     sp+9 :                            : 未使用
;
      CALL    !GETPARA                ; 引数を作業領域（HL、DATANO）に
READLOOP:
      getmw                           ; 上位バイト読み出し
      MOV     [HL+1],A                ; 読み出したデータをセーブ
      getmw                           ; 下位バイト読み出し
      MOV     [HL],A                  ; 下位バイトの保存
      INCW    HL
      INCW    HL                      ; メモリ・ポインタ更新
      DBNZ    DATANO,$READLOOP        ; 指定回数繰り返す
      CSsignalOFF                     ; CS 信号をオフ
      POP     AX                      ; レジスタの復帰
      POP     HL
      RET
```

(7) 共通処理部 1

EEPROM へのコマンド転送と、その後のアドレス転送までの内部処理を行う部分です。これらの処理は、リード／ライトで共通となるので、プログラムの大きさを小さくするために共通化しています。特にアドレス部分は、長さが変わるので、共通化によるプログラム・サイズ削減に大きく影響します。

リードとライトでは入り口が異なり、対応するコマンドを設定したら、後の処理は共通になります。

```
;
;       リード・コマンド発行
;
RDCMD:
        MOVW    ADDR,AX                 ; アドレスをワークに保存
        MOV     A,#01100000b            ; リード・コマンド
        BR      $SENDCMD                ; コマンド送信へ
;
;       ライト・コマンド発行
;
WRCMD:
        MOVW    ADDR,AX                 ; アドレスをワークに保存
WRCMD2:
        MOV     A,#01010000b            ; ライト・コマンド
;
;       実際にコマンドの転送を行います（ここからはリードとライトは共通処理）
;
SENDCMD:
        CSsignalON                      ; CS 信号をオン
        PUSH    BC
        MOV     B,#4                    ; コマンドのビット数セット
        putbits                         ; コマンド転送
;
;       アドレス処理
;   EEPROM のアドレスが 8 ビット未満／8 ビット／9 ビット以上で分けて処理する
;
        MOV     A,ADRLEN
        SUB     A,#3                    ; アドレス長をチェック
        BZ      $LOWADR                 ; 8 ビットなら分岐
        BC      $LESTHAN8               ; 8 ビット未満なら分岐
;
;       アドレスの上位を処理する
;
        MOV     C,A
        MOV     B,A
        MOV     A,ADDR+1                ; 上位アドレスを読み出す
ADORLOOP:
        ROR     A,1                     ; 1 ビット右シフト
        DBNZ    C,$ADORLOOP
;       MOV     X,A                     ; 上位アドレスをセット
        putbits                         ; 上位アドレス出力（可変長）
;
;       アドレスの下位を処理する
;
LOWADR:
        MOV     B,#8
        MOV     A,ADDR                  ; アドレスを読み出す
        putbits                         ; 下位アドレス出力
        POP     BC
        RET
```

(8) 共通処理部 2

共通に使用する引数を扱う内部ブロック処理です。引数はスタックを介して渡され、データ数を作業用メモリ(DATANO)に、バッファ・メモリのアドレスを HL レジスタに設定します。これらを用いて、以降の転送を行います。

```
;
;       ブロック・アクセスでのパラメータ引き取り処理
;
;   EPROM 制御関数は C（ノーマル・モード）からの関数コールに対応
;   そのため、この段階でのスタックのデータは次のようになっている
;
;       sp    : PC の下位                   : 戻りアドレス下位
;       sp+1  : PC の上位                   : 戻りアドレス上位
;       sp+2  : X レジスタ（第 1 引数）     : EEPROM アドレス（下位）
;       sp+3  : A レジスタ（第 1 引数）     : EEPROM アドレス（上位）
;       sp+4  : L レジスタ                  : 未使用（レジスタのセーブだけ）
;       sp+5  : H レジスタ                  : 未使用（レジスタのセーブだけ）
;       sp+6  : PC の下位                   : メイン・ルーチンへの戻りアドレス下位
;       sp+7  : PC の上位                   : メイン・ルーチンへの戻りアドレス上位
;       sp+8  : 第 2 引数の下位             : バッファ・アドレスタ（下位バイト）
;       sp+9  : 第 2 引数の上位             : バッファ・アドレスタ（下位バイト）
;       sp+10: 第 3 引数の下位              : 書き込みデータ数
;       sp+11:                              : 未使用
;
GETPARA:
        MOVW    AX, SP                      ; 2 つ目の引数を取り出す準備
        MOVW    HL, AX                      ; ポインタを HL に設定

        MOV     A, [HL+10]                  ; データ数（ワード）の取り込み
        MOV     DATANO, A                   ; ワーク領域にコピー
        MOV     A, [HL+8]                   ; バッファ・メモリのアドレスを
        MOV     X, A
        MOV     A, [HL+9]                   ; 読み出して作業用に使用するため
        MOVW    HL, AX                      ; HL レジスタにセット
        RET
```

(9) ワード・データの書き込み

CS 信号を ON にし、コマンドに続いて、引数として渡された EEPROM のアドレスを必要なビット数だけ転送します。その後、2 バイトのデータを転送します。転送が完了したら CS 信号を OFF にし、ここから EEPROM では実際の書き込みが開始します。書き込み中に他の処理ができるような、書き込み完了の待ち合わせ処理は行いません。別途、書き込み完了待ちを行ってください。

```
;/*-------------------------------------------------------*/
;/*                                                       */
;/*          1 ワード・データの書き込み                   */
;/*  ＜機能＞                                             */
;/*  ワード・データを指定されたアドレスに書き込む         */
;/*  ＜入力パラメータ（引数）＞                           */
;/* AX     :第 1 の引数 :EEPROM 内のアドレス              */
;/* SP+2, 3:第 2 の引数 :書き込みデータ（ワード）         */
;/*  ＜戻り値＞  なし                                     */
;/*                                                       */
;/*-------------------------------------------------------*/
;void _write_mw(int reg_adr, unsigned int data)
__write_mw:
WRITE_MW:
        CALL    !WRCMD                      ; コマンド発行
        PUSH    HL
        PUSH    AX
```

```
;
;   このサブルーチンは C（ノーマル・モード）からの関数コールに対応
;   そのため、この段階でのスタックのデータは次のようになっている
;
;       sp    : X レジスタ（第 1 引数）      : EEPROM アドレス（下位）
;       sp+1  : A レジスタ（第 1 引数）      : EEPROM アドレス（上位）
;       sp+2  : L レジスタ                   : 未使用（レジスタのセーブだけ）
;       sp+3  : H レジスタ                   : 未使用（レジスタのセーブだけ）
;       sp+4  : PC の下位                    : 戻りアドレス下位
;       sp+5  : PC の上位                    : 戻りアドレス上位
;       sp+6  : 第 2 引数の下位              : 書き込みデータ（下位バイト）
;       sp+7  : 第 2 引数の上位              : 書き込みデータ（下位バイト）
;
        MOVW    AX,SP                   ; 2 つ目の引数を取り出す準備
        MOVW    HL,AX                   ; ポインタを HL に設定
        MOV     A,[HL+7]                ; 上位バイトを読み出す
;       MOV     X,A
        putmw                           ; 上位バイト送信
        MOV     A,[HL+6]                ; 下位バイトを読み出し
;       MOV     X,A
        putmw                           ; 下位バイト送信
        SKsignalOFF                     ; SK 信号を立ち下げ
        CSsignalOFF                     ; CS 信号をオフ
        POP     AX
        POP     HL
        RET
```

(10) ブロック・データの書き込み

1 ワード書き込み処理をコマンド処理から必要な回数だけ繰り返します。このため、書き込み完了待ちまでの処理を内部で行うので、別途書き込み待ちを行う必要はありません。

```
;/*-------------------------------------------------------------*/
;/*                                                             */
;/*          データ（ブロック）の書き込み                       */
;/*  ＜機能＞                                                   */
;/*  引数として渡されたアドレスに、指定された数のデータを書き込む */
;/*  ＜入力パラメータ（引数）＞                                 */
;/* AX     : 第 1 の引数 :EEPROM 内のアドレス                   */
;/* SP+2,3: 第 2 の引数 :バッファ・メモリのアドレス             */
;/* SP+4,5: 第 3 の引数 :書き込みデータ数（ワード単位）         */
;/*  ＜戻り値＞  なし                                           */
;/*                                                             */
;/*-------------------------------------------------------------*/
;void _write_mw_block(int reg_adr, unsigned int *data, unsigned char size)

__write_mw_block:
WRITE_MWB:
        PUSH    HL
        PUSH    AX
        MOVW    ADDR,AX                 ; EEPROM アドレスを設定
;
;   このサブルーチンは C（ノーマル・モード）からの関数コールに対応
;   そのため、この段階でのスタックのデータは次のようになっている
;
;       sp    : X レジスタ（第 1 引数）      : EEPROM アドレス（下位）
;       sp+1  : A レジスタ（第 1 引数）      : EEPROM アドレス（上位）
;       sp+2  : L レジスタ                   : 未使用（レジスタのセーブのみ）
;       sp+3  : H レジスタ                   : 未使用（レジスタのセーブのみ）
;       sp+4  : PC の下位                    : 戻りアドレス下位
;       sp+5  : PC の上位                    : 戻りアドレス上位
```

```
;       sp+6 : 第 2 引数の下位              : バッファ・アドレスタ（下位バイト）
;       sp+7 : 第 2 引数の上位              : バッファ・アドレスタ（下位バイト）
;       sp+8 : 第 3 引数の下位              : 書き込みデータ数
;       sp+9 :                              : 未使用
;
        CALL    !GETPARA                    ; 引数を作業領域（HL,DATANO）に
WRITELOOP:
        CALL    !WRCMD2                     ; コマンド発行
        MOV     A,[HL+1]                    ; 上位データの読み出して
;       MOV     X,A
        putmw                               ; それを送信
        MOV     A,[HL]                      ; 下位データを読み出して
;       MOV     X,A
        putmw                               ; それを送信
        SKsignalOFF                         ; SK 信号を立ち下げ
        CSsignalOFF                         ; CS 信号をオフ
        MOVW    AX,ADDR                     ; 次アドレスに更新
        INCW    AX
        MOVW    ADDR,AX
        INCW    HL
        INCW    HL
        CALL    !WAITW;                     ; 書き込み完了を待つ
        DBNZ    DATANO,$WRITELOOP
        POP     AX
        POP     HL
        RET
```

(11) 書き込み完了待ち

1 ワード書き込み処理を用いたとき、書き込み完了までの待ち合わせのために使用します。

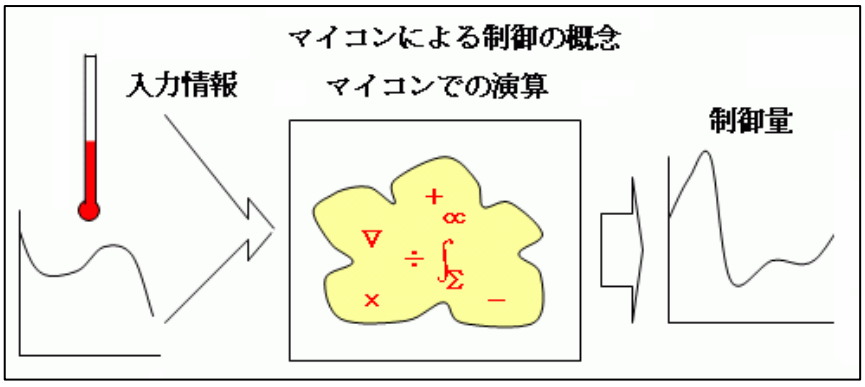
```
;/*-------------------------------------------------------*/
;/*                                                       */
;/*                    書き込み完了待ち                   */
;/*  ＜機能＞                                             */
;/*  EEPROM へのデータ転送後、CS 信号が立ち下がると       */
;/*  転送されたデータが実際に書き込みが行われる           */
;/*  ここでは、完了するまで待つ                           */
;/*  ＜入力パラメータ（引数）＞  なし                     */
;/*  ＜戻り値＞  なし                                     */
;/*                                                       */
;/*-------------------------------------------------------*/
;void _waitwt(void)

__waitw:
WAITW:
        CSsignalON                          ; CS 信号をオン
        NOP                                 ; ステータス出力待ち
        NOP
        NOP
WAITLOOP:
        NOP
        BF      P2.1,$WAITLOOP              ; 書き込み完了待ち
        CSsignalOFF                         ; CS 信号をオフ
        RET
```

1.1.23 テーブル参照と補間処理

テーブル参照と補間処理 [プログラム例以外は共通、プログラム例は 78K0]

通常、マイコンを制御用に使用する場合、入力情報に何らかの加工（入力情報から計算）をして、制御量を決めて出力することになります。

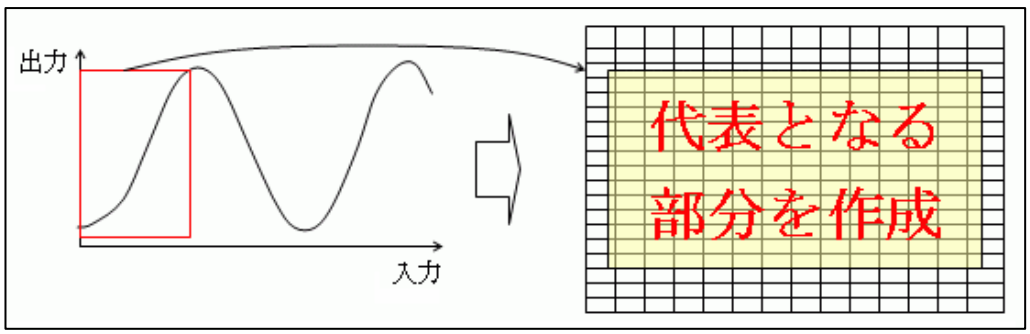


[はじめに]

制御を行うマイコンで必要とされる演算（計算）処理は、簡単なものからひじょうに複雑なものまで、用途に応じて大きく変わります。演算だけで、複雑な計算を決められた時間内に処理するには、マイコンにとても高い演算能力を要求されることがあります。その対応のためだけに、演算能力を高くするのは、あまり実用的ではありません。そこで、用いられるのがテーブル参照です。

[テーブル参照の考え方]

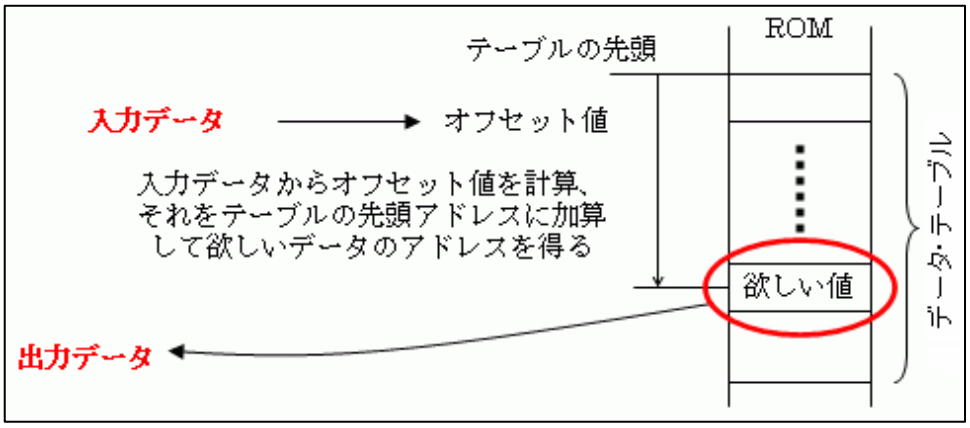
テーブル参照は、与えられた入力（x）に対して、あらかじめ結果を計算しておき、表（テーブル）で準備しておくものです。計算そのものは、ある程度の範囲の入力値に対して実行しておく必要があるので、手間はかかります。しかし、PCやその他のコンピュータを利用することで、比較的簡単にできます。



テーブルを作成する場合は、その大きさに注意する必要があります。上の例のような繰り返しの場合は、すべての入力値の範囲でテーブルを作成する必要はなく、繰り返される部分を代表する範囲だけ作成すれば十分です。その上で、入力される値を繰り返しの周期で割った余りやその等を用いてテーブルを参照すれば済みます。

[テーブル参照の使い方]

テーブル参照では、入力データを元に、テーブルの中で、その先頭から欲しいデータがある場所までのアドレスの差（オフセット値）を計算します。欲しいデータがバイト幅のデータであれば、入力データはそのままオフセット値として使えますし、16 ビット幅のデータなら入力データを 2 倍すれば、オフセット値となります。このオフセット値をテーブルの先頭アドレスに加算すれば、欲しい値が格納されたメモリのアドレスが得られます。そのアドレスのメモリを読み出せば、欲しい演算結果が得られます。



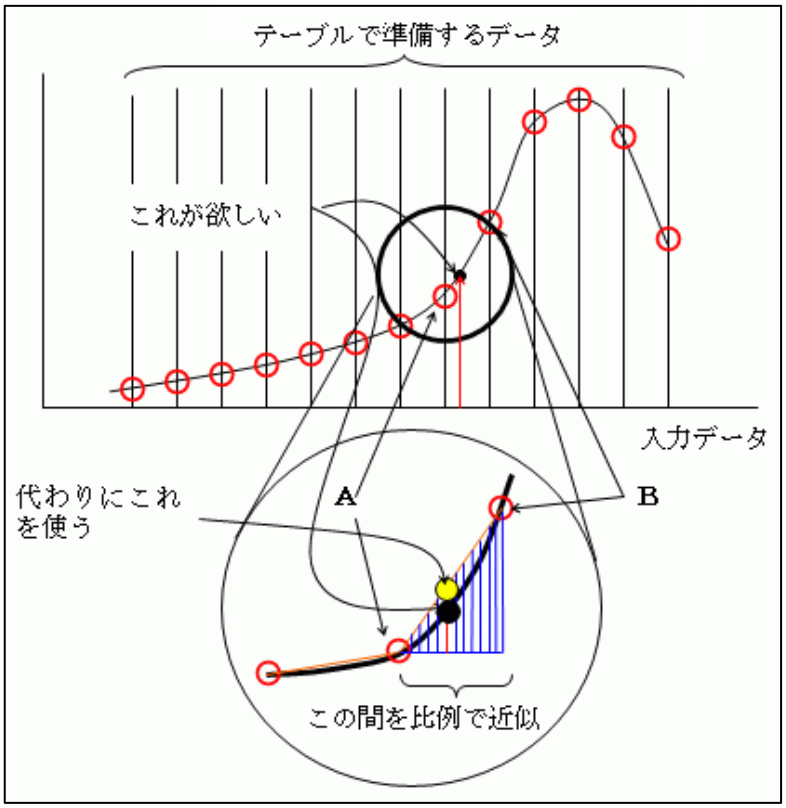
【テーブル参照命令】
4 ビット・マイコンの 75X では、テーブル参照専用命令（MOVT XA,@PCDE や MOVT XA,@PCXA）がありました。78K0 ではテーブル参照命令と明記された命令はありませんが、ROM も RAM も同一のメモリ空間に配置されているので、わざわざ専用の命令を使用する必要がないのです。テーブル参照で便利な命令としては、ベースト・インデクス・アドレッシング命令があります。これは MOV　A,[HL+B]や MOV　A,[HL+C]などで、HL レジスタにテーブルの先頭アドレス、B または C レジスタにオフセット値を設定して実行すれば、先頭アドレスにオフセット値が加算され、そのメモリの値をAレジスタに読み込みます。

[テーブル参照の注意点]

テーブル作成の際に問題になるのは計算の精度や、準備するデータ数です。あまり精度を欲張りすぎるとテーブルが大きくなり、メモリを圧迫してしまいます。すべてをテーブル参照で処理するのではなく、最後に補間処理を行うことで、テーブルを小さくできます。「**補間処理**」をご参照ください。

[補間処理]

テーブル参照で大まかな結果を得て、細かな部分は簡単な比例計算で求めるものです。テーブルとして準備するデータは、ある程度大きな間隔の値ですむので、テーブルの大きさを数分の 1 に抑えられます。



たとえばデータ量を 1/16 にすることを考えると、入力データの下位 4 ビットが 0000 に対応するデータだけをテーブルとして準備しておきます。そこで、次の手順で補間処理ができます。

①入力データを 1/16（右に 4 ビット・シフト）した値をオフセット値としてテーブルを参照する。前図の A 点のデータが得られる。

②テーブル上の次のデータ（前図の B 点）との差を計算する。

③これを 1/16 して、入力データの下位 4 ビットと乗算すれば、A 点からの増分が得られる。このままの順序で計算すると、丸め誤差が発生するので、計算の順序を変えて、乗算を先に計算する必要がある。

④この増分をテーブルから得られた A 点の値に加算する。

基本的には、加算と乗算やシフト程度ですみ、あまり時間がかからずに処理できるはずです。ただし、②で計算した誤差には符号を含むので、その後の計算がめんどうになります。そこで、実際の処理では、加重平均を取れば簡単です。これらの処理が等価であることは、次のように式を変形できることから分かります。ここで、m は分割数で、nが入力データの下位ビットで示される数値とします。

$$A+(B-A)\times n/m=\{mA+(B-A)n\}/m=\{(m-n)A+nB\}/m$$

この最後の式でわかるように、分割数から入力の下位を引いたものとテーブルから得られたデータ A を乗算したものと、入力の下位とテーブルの次のデータ B を乗算したものを加えて分割数で割っても同じ結果になります。上の例では、下位 4 ビットで示されるオフセット分だけテーブル上の次のデータ（B のデータ）を加算し、残りは A 点のデータを加算して 1/16 することになります。

プログラム例では、テーブルのデータを 1/32 に圧縮した場合を例示しています。通常は、ここまでのデータ圧縮は必要なく、1/2 に圧縮しても十分な場合もあります。1/2 圧縮では、補間処理としては単純で、前後の値の平均で処理でき、プログラムを簡単にできるので、処理速度は速くなります。トータルのメモリ使用量も、単なるテーブル参照よりはるかに小さくできます。ただし、テーブルのデータ量が多くなるので、1/2 圧縮の例は省きます。

【補間による誤差について】
補間処理での誤差を考えてみます。sin0～sin（π/2）の範囲について、10 ビット分解能（1024 分割）で、結果を最大 1023（＝sin(π/2)）とします。詳しい計算結果は省きますが、次のようになります。
256 分割で 2 ビット分の補間処理を行った場合、誤差は最大で 0.005
128 分割で 3 ビット分の補間処理を行った場合、誤差は最大で 0.019
64 分割で 4 ビット分の補間処理を行った場合、誤差は最大で 0.077
32 分割で 5 ビット分の補間処理を行った場合、誤差は最大で 0.31
いずれも、十分小さな誤差（0.5 ビット以下）に収まっています。10 ビット分のテーブルを用いると、2048 バイトの容量が必要ですが、それを 64 バイトまで小さくしても、十分な精度が得られることになります。

[プログラム例]

sin0～sin（π/2）を 10 ビット精度で求めるプログラム例を示します。入力値としては、0～sin（π/2）の範囲を 1024 分割して、0～1023 の 10 ビットで示します。なお、入力パラメータからオフセット値を計算する部分で、#00111110b との論理積をとる部分を#01111110b に変更し、テーブルを拡張すれば、0～2047 の 11 ビットまで対応できるようになり、sin 0～sinπ までの値を得ることができます。

このプログラムでは、入力パラメータと結果の格納用に 16 ビットの変数(CONVDATA)、それに続く 8 ビット変数(WORK1)を作業用に使用し、これらをショート・ダイレクト・アドレッシング領域に確保します。

```
WORK00          DSEG    SADDRP
CONVDATA:
        DS      2               ;入力パラメータと結果の戻り領域
WORK01          DSEG    SADDR
WORK1:  DS      1
```

プログラムはサブルーチンとしており、大きくは 4 つの部分（3 つはプログラム本体で、1 つは内部のサブルーチン）から構成されます。

(1) 入力パラメータからのテーブル参照

最初の部分で、テーブルとして 1/32 に圧縮したデータを格納しておきます。そのため、テーブルのオフセット値を求めるために、入力データを 1/32 しますが、データ長が 10 ビットで 2 バイト必要なので、これを 2 倍にすることになります。つまり、入力パラメータを 1/16 すれば、テーブル参照のオフセット値となります。ここでは、入力の下位 10 ビットだけが意味をもちます。そこで、上位 4 ビットは無視して、78K0 のデイジット・ローテート命令（ROR4 [HL]）を使用して、4 ビット右回転することで 1/16 しています。このようにして得られたオフセット値（8 ビット以下の値）にテーブルの先頭アドレスを加算して、テーブル上のデータのアドレスが得られます。テーブルを 200H のように、下位アドレスが 00 から配置してあれば、加算する必要はなく、単にオフセット・アドレスを下位アドレスに代入するだけでも処理できます（この例では一応、加算して求めています）。ここまでの部分は、1/16 する処理を除けば、通常のテーブル参照と同じです。

```
CONVSUB:
        PUSH    AX
        PUSH    HL
        MOVW    AX, CONVDATA        ;A に上位、X に下位
        MOVW    HL, #CONVDATA
        ROR4    [HL]                ;A の下位 4 ビットと[HL]を 1/16 に変換
        MOV     A, [HL]
        AND     A, #00111110b       ;オフセット値を得る
        ADD     A, #LOW TABLE       ;下位アドレスを計算
        MOV     L, A
        MOV     A, #0
        ADDC    A, #HIGH TABLE      ;上位アドレスを計算
        MOV     H, A                ;HL はテーブルの対象を指す
        MOV     A, [HL]             ;下位バイトを読み出し
        MOV     CONVDATA, A         ;下位バイトの初期値としてセット
        MOV     A, [HL+1]
        MOV     CONVDATA+1, A       ;上位バイトのセット
```

(2) 補間のための計算

次に、加重平均をとるため、テーブルの値と次の値に重みを付けて積算します。補間が不要な場合は、すでに出力パラメータとしてテーブル参照した結果がセットされているので、処理を終了します。補間処理では、補間データだけ次のデータを加算（実際は乗算）し、32 から補間データを引いた数だけ、テーブルのデータを加算します。実際の計算は、サブルーチンで処理します。ここでは、計算に必要なパラメータを準備しておき、必要な回数だけサブルーチンをコールします。計算は 16 ビット×8 ビット＋16 ビットなので、処理を簡単にするため、必要なパラメータはスタックに入れておき、HL レジスタを積算データのインデックスで使用しています。なお、積算した結果は 16 ビットに収まりますが、処理の共通化のため、1 バイトを作業用で使用しています。この作業用の 1 バイトは、データとしては意味がないのですが、初期値として 0 を入れてあります。

```
        MOV     A,X
        AND     A,#00011111b    ;補間のためのデータを得る
        BZ      $EXITS          ;補間不要なら終了
;
;       補間（加重平均）のための準備
;       （スタックに計算データを入れておく）
;
        MOV     WORK1,#0        ;上位データを初期化
        MOV     X,A             ;補間データを X レジスタにセット
        MOV     A,[HL+3]        ;次のデータの上位をセット
        PUSH    AX              ;計算データをセーブ
        MOV     A,[HL+2]        ;次のデータの下位をセット
        PUSH    AX              ;計算データをセーブ
        MOV     A,#31
        SUB     A,X             ;32 に対する残り数を計算
        MOV     X,A
        MOV     A,[HL+1]        ;テーブルの上位データをセット
        PUSH    AX              ;計算データをセーブ
        MOV     A,[HL]          ;テーブルから下位データを
        MOVW    HL,#CONVDATA    ;ポインタを設定
;
;       テーブルの検索データ分の積算を行う
;
        CALL    !SEKIWA         ;下位データ演算
        POP     AX
        INCW    HL
        CALL    !SEKIWA         ;上位の計算と結果の積算
;
;       補間データから次のデータ重み分の積算を行う
;
        POP     AX
        DECW    HL
        CALL    !SEKIWA
        POP     AX
        INCW    HL
        CALL    !SEKIWA
```

(3) 積算した結果を 1/32 して最終的な結果を計算

ここまでの結果は、10 ビットと 5 ビットの乗算と同じで、16 ビット以下になります。結果を切り捨てでなく四捨五入するため、積算結果の 1/2 の値を加算後、1/32 処理をします。これは、これまでの結果で、HL が CONVDATA+1 を指していることを利用しています。また、1/32 するのに、4 ビットの左回転を行い、バイトをずらして考えることで 1/16 にして、さらに 1/2 処理を行います。

```
        MOV     A,CONVDATA      ;結果の下位をロード
        ADD     A,#00010000b    ;四捨五入のために 0.5 分を加算
        BNC     $NEXT
        INC     CONVDATA+1      ;上位に桁上げ
NEXT:   AND     A,#11100000b    ;未使用ビットをマスク
        ROR     A,1             ;結果を下位 4 ビットにセット
        ROR     A,1             ;A=00XXX000、CY=0
        ROR     A,1             ;A=000XXX00、CY=0
        ROR     A,1             ;A=0000XXX0、CY=0
```

```
        ROL4    [HL]            ;結果を 1/16 に変換
        RORC    A,1             ;さらに 1/2 に変換
        XCH     A,[HL]          ;上位 3 ビットと下位を交換
        RORC    A,1             ;下位も 1/2 に変換
        MOV     CONVDATA,A      ;結果を保存

EXITS:  POP     HL
        POP     AX
        RET
```

(4) 積算のための積和演算

内部のサブルーチンで、A、X レジスタに乗数と被乗数を設定し、HL で積算されるデータを指した状態でコールします。A、X レジスタの内容を乗算した結果を[HL]と[HL+1]に積算していきます。

```
;       8 ビットの積和演算
;   (結果は 16 ビットに収まるものとする)
;
SEKIWA:
        MULU    X               ;重みの計算
        XCH     A,X             ;結果の上位と下位を交換
        ADD     A,[HL]          ;下位を積算
        MOV     [HL],A          ;結果をセーブ
        MOV     A,X             ;結果の上位をもってくる
        ADDC    A,[HL+1]        ;上位への桁上げ計算
        MOV     [HL+1],A        ;上位バイトをセーブ
        RET
```

(5) 0(0)～1024(＝π/2)に対応したテーブル

テーブルのデータは 1024 個ではなく、1/32 に圧縮した(32 ずつのデータだけを取り出した)データです。

対象範囲を 0(0)～2048(＝π)に広げたい場合は、このテーブルに 0～1022 のデータを逆の順で追加します。

```
TAREA   CSEG    AT      200H
;
;       sin0～sin (π/2) を 32 分割したときの値を 1023 倍した値のテーブル

TABLE:  DW      0,      50,     100,    150
        DW      200,    249,    297,    345
        DW      392,    437,    482,    526
        DW      568,    609,    649,    687
        DW      723,    758,    791,    822
        DW      851,    877,    902,    925
        DW      945,    963,    949,    992
        DW      1003,   1012,   1018,   1022
        DW      1023
```

【2048(＝π)以上への対応】

π以上は、テーブルやオフセットのマスクだけで対応できません。これは、結果の範囲(負の値をどう表現するか)を見直す必要があるからです。このプログラムでは、結果を 0～1023 の範囲で扱えるようになっています。このため、sin(π/2)～sin(3π/2)を 0～1023 で表す必要があります。具体的には、1(sin(π/2))を 1023、0(sinπ)を 512、－1(sin(3π/2))を 1 で表現すれば対応できます。負の数を 2 の補数で表現する場合、このプログラムでは対応できませんので、外で符号を付ける(0 から引く)ような処理が必要となります。

1.1.24 マイコンの開発

マイコンのシステム開発の流れ

Q1. マイコンのシステム開発は、どのように行うのですか。

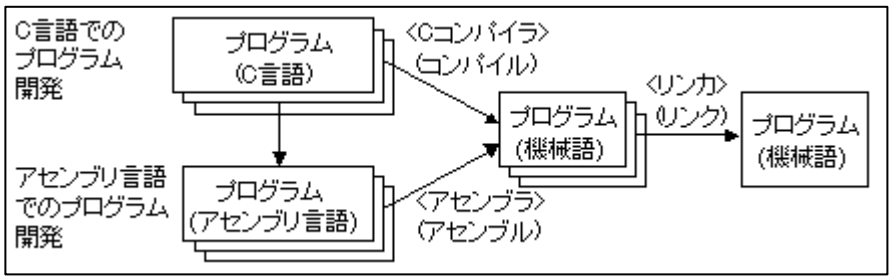
A1. 一般には、次のような開発の流れになります。

(1) システム設計
(2) 部品調達
(3) 試作ハードウエア開発
(4) プログラム開発
(5) IE やシミュレータ、評価ボード等での動作検証とデバッグ
(6) ROM 書き込み
(7) 実チップでの動作検証
(8) 量産ボード開発
(9) マスク ROM 発注／フラッシュ・メモリ書き込み
(10) プログラム品での動作検証

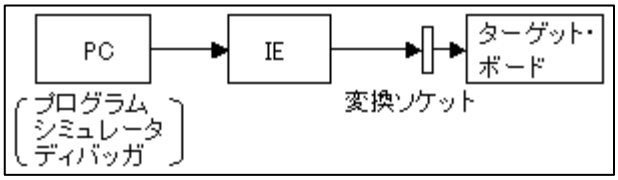
(1)では、システム要件に適したマイコンを選定し、各内部機能の動作モードを定義したり、外部接続やアドレスの割り当て、処理フローの作成などを行います。

(3)と(4)は、ハードウエアとソフトウエアの分担で、同時進行することがあります。(3)では、(1)の設計内容に従って、ボードのハードウエア設計と部品(マイコン以外)の実装をします。試作なので、一般にソケット実装でのラッピング配線やはんだ付けによる配線をします。(4)では、パソコンでプログラム作成(プログラミング)をします。開発言語は C 言語や、 CPU のアーキテクチャに依存するアセンブリ言語です。

プログラムが完成すると、アセンブリ言語ならアセンブラで、またC言語なら C コンパイラで、それぞれ機械語(マシン語)に変換します。C コンパイラによっては、C 言語をアセンブリ言語に落としてから、アセンブラで機械語に変換する場合もあります。これらが分割した機械語プログラムなら、その後リンカで結合します。



ターゲット・システムのマイコンの代わりに、(5)で IE(インサーキット・エミュレータ)とパソコンを接続し、 IE 上に展開したプログラムを実行します。問題があれば、シミュレータやデバッガで、回路やプログラムのディバグをします。想定するマイコンのパッケージと、IE のアダプタの形状が異なる場合、変換ソケットが必要です。

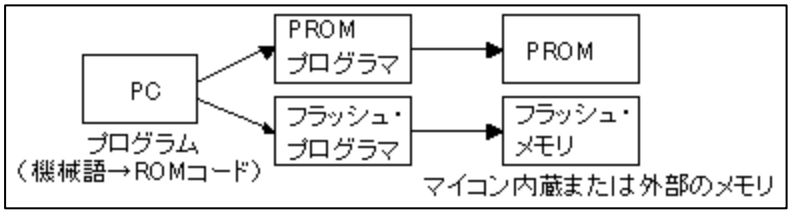


(6)では、 PROM(内蔵または外部)では PROM プログラマで、またフラッシュ・メモリではフラッシュ・プログラマで、パソコンを通じて生成した機械語を ROM コードに変換して書き込みます。

プログラムを書き込んだ ROM とマイコンをターゲット・ボードに実装して、(7)でプログラム動作を確認します。

(8)で、プリント基板を作成します。

マスク品で大量生産の場合は、(9)で作成します。フラッシュ品なら、ここで完成プログラムを書き込みます。
マスク品では、プログラムの提示だけでなく、マスク・オプションによる機能指定ができます。発注方法はROM コードの発注方法をご参照ください。
(10)では、(9)で完成したマイコンと必要なデバイスを(8)の基板に実装して、最終的な動作確認を行います。



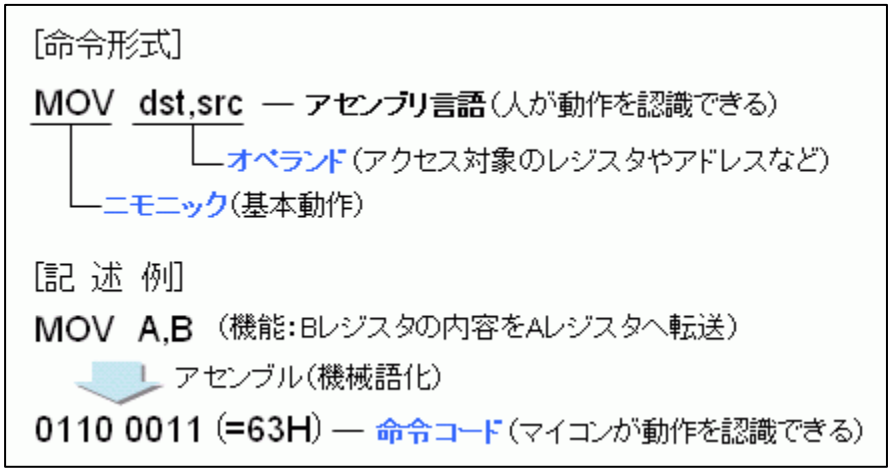
Q2. マイコン・システムの開発で、PROM 品とフラッシュ品ではどのように違いますか。

A2. プログラム開発に違いがあります。
まず、プログラムの書き込みに使用するライタが、PROM プログラマかフラッシュ・プログラマかの違いです。書き込み方法が異なるので、互いに代用することはできません。
次に、PROM 品はプログラマ上だけ書き込めませんが、フラッシュ品はターゲット・システムにプログラマを接続してオンボード書き込みもできます。このためフラッシュ品では、セット出荷後のプログラム変更が比較的容易です。
さらに All Flash マイコンでは、フラッシュ・メモリを書き換えながらプログラムのディバグを行うオンチップ・ディバグ機能を内蔵しています。
なお、オプション・バイトがあるフラッシュ品では、ユーザ・オプションとしてこれを設定する必要があります。
フラッシュ品の書き込みについては、フラッシュ書き込みをご参照ください。

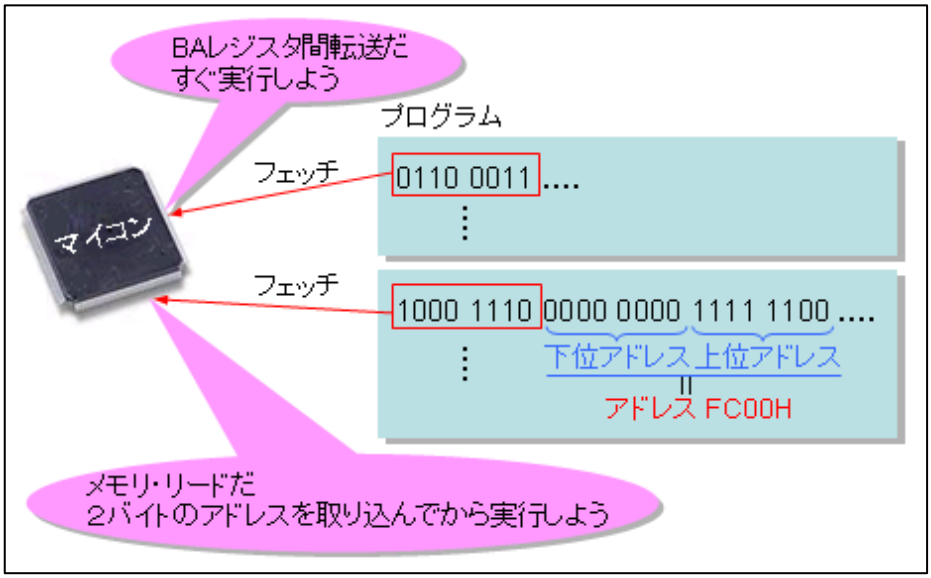
命　令

Q1. マイコンの命令は、どういう構成になっているのですか。

A1. マイコンの動作をレジスタ・レベルで直接定義する命令(インストラクション)記述は、アセンブリ言語です。アセンブリ言語は、基本動作を定義するニモニックと、アクセス対象の内部レジスタやアドレス(メモリ、I/O)を指定するオペランドで構成されます。
C 言語では、マイコンの命令を意識せず、演算や転送、判断などの概念を記述します。C 言語とマイコンの命令セットは、C コンパイラが関連付けます。このため、C コンパイラによって、コード生成効率が異なります。
また、C コンパイラやアセンブラが生成する機械語は、マイコンが直接実行できる形式の命令コードです。



内部レジスタ間アクセスを1バイトの命令コードで記述するマイコンであっても、転送先(dst)や転送元(src)に、メモリや I/O のアドレスの値を記述する場合は、命令コードが2バイト以上になります。命令コードの1バイト目をオペコード(オペレーション・コード)と呼び、マイコンがオペコードを読み取ることをオペコード・フェッチといいます。マイコンは、フェッチしたオペコードを解析して、何バイト命令かを認識し、残りの命令コードがあるかどうか(さらに読み取ってから実行するかどうか)を判断します。



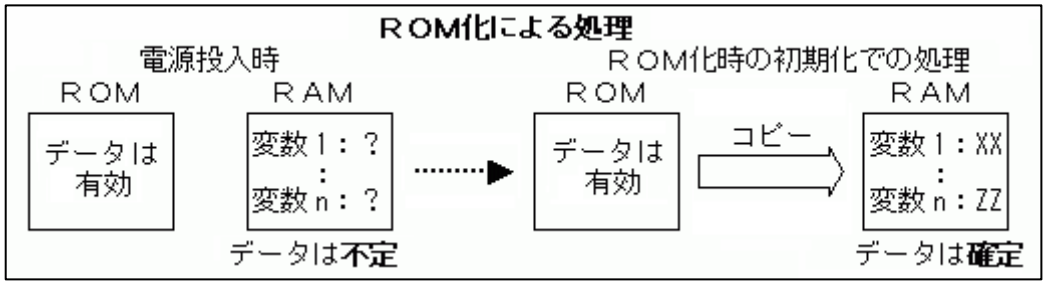
★ **Q2.** 転送命令では、転送元にデータが残らないのですか。

A2. いいえ残ります。命令の動作としては"dst←src"と表記しますが、実際は転写です。転送後にそのデータを使用する場合でも、コピーや退避などの操作は不要です。

ROM 化

Q1. ROM 化とはなんですか。また、なぜ ROM 化が必要なのですか。

A1. マイコンのアプリケーションプログラムは、最終的に内蔵の ROM(マスク ROM やフラッシュ)に書き込み、応用システムに組み込みます。このため、C コンパイラでプログラムを作るだけでは不十分です。プログラムや固定データを内蔵 ROM へ配置するだけでなく、初期値のある変数への対応などが必要です。たとえば、初期値のある変数は、アプリケーションプログラムの動作時には RAM に配置されていて、参照や書き換えができる必要があります。しかし、マイコンの電源投時は、RAM の内容が不定です。そのため、アプリケーション・プログラムの起動前に、変数の初期値を設定する必要があります。この処理が ROM 化と呼ばれ、ROM 中に初期値データを事前に格納しておき、起動時にそのデータを対応する RAM にコピーします。



78K シリーズでは標準で ROM 化するようになっていて、気にする必要はありませんが、V850 ではオプションで指定する必要があります。さらに、ROM から RAM へのコピーは、アプリケーションプログラムの中で実行(_rcopy 関数を使用)する必要があります。

Q2. フラッシュ品を使ってデバッグ(評価)を行う場合にも、ROM 化は必要ですか。

A2. はい、ROM 化は、フラッシュ品を用いた評価の場合にも必要です。

Q3. エミュレータを用いたデバッグでは ROM 化をしないで、ROM 発注のときだけ ROM 化すればいいですか。

A3. いいえ、エミュレータを用いたデバッグ段階から ROM 化しておいてください。

ROM 化してないと、デバッグでリセットからやり直した場合に、初期値のある変数の初期値がおかしくなっている可能性があります。

エミュレータでは、初期値のある変数も、直接 RAM にオブジェクトをダウンロードするので、初期値が入ります。しかし、デバッグ中に変数が書き換えられると、その状態が残ってしまいます。再度、プログラムをリセットからやり直した場合には、書き換えた値が新しい初期値となるので、プログラムの動作が変わってくることが考えられます。ROM 化してあると、リセットからやり直すことで、初期値のある変数に初期値がコピーされるため、正しい初期値で動作します。

ROM コード発注時のフォーマット(インテル・ヘキサ形式)

Q1. ROM コード発注時に INTEL HEX フォーマットを使いますが、どのようなものですか。

A1. INTEL HEX フォーマットは、オブジェクト(機械語)をやり取りするためのフォーマットのひとつで、1 バイトを ASCII コードで 2 桁の 16 進数(HEX)に変換して表現したものです。

1 行(レコード)で最大 255 バイト分のデータを表現でき、これを必要な回数だけ繰り返します。

具体的なフォーマット例を以下に示します。

① ":"がレコードの開始を示します

② 次の 16 進数 2 けた(10)が、レコードのデータ数を示します

③ 次の 16 進数 4 けた(0000)が、データのアドレスを示します

④ 次の 16 進数 2 けた(00)が、レコード・タイプで、データ・レコードであることを示します
(00 はデータ・レコード、01 はエンド・レコード、02 はセグメント・レコード等)

⑤ ここがデータそのもので、データは"00","01","02",・・・,"0E","0F"です

⑥ 最後がチェクサムで、データ数からチェックサムまでを加算すると、下位 8 ビットは 00 になるようになっていて、エラー・チェックに使用します。

以上の情報に改行がついて、1 行が 1 つのレコードとなります。

2 行目がエンド・レコードで、データ数②が 00、レコード・タイプ④が 01 になります。

フォーマットの例

```
: 10 0000 00 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 7D
①② ③ ④                    ⑤                          ⑥

: 00 0000 01 FF
①② ③ ④ ⑥
```

備考　説明のために、意味のある単位で区切っています

これだけでは、アドレスは 16 ビット(64K バイト)しか表せません。それ以上の指定には、セグメント・レコード(レコード・タイプ 02)を使用します。次の例は、前例の前にセグメント・レコードを追加したものです。セグメント・レコードで指定したセグメントの値を 16 倍した値が、以降のレコードのアドレスに加算されます。ここでは⑤の 1000H がセグメントとして指定され、以降のデータは 10000H～1000FH に格納されることを示します。

６４Ｋバイトを越える場合の例

```
: 02 0000 02 1000 EC
①② ③ ④ ⑤ ⑥
:100000000001020304050607080900A0B0C0D0E0F7D
:00000001FF
```

Q2. ROM 発注には、INTEL HEX フォーマットであれば、どのようなものでもいいですか。

A2. いいえ、マスク ROM 処理では、チェックのための制限を設けています。たとえば、INTEL HEX フォーマットでは、レコードのアドレスの順番に制限はありませんが、マスク ROM 処理では、アドレスは連続して大きくなる必要があります。その他にも、厳密なチェックを行っていますので、ROM 発注では必要なオプションを指定して、オブジェクト・コンバータを通した結果をご使用ください。

All Flash マイコンでのプログラム開発(IMS レジスタ設定)[78K0/Kx2]

Q1. 78K0/KB2 で最終的なプログラム・サイズを 16K バイト以下にするので、μPD78F0501 をターゲットにします。ユーザーズ・マニュアルに、IMS レジスタの設定をディバグ対象の製品に合わせるように記述されています。開発段階で 16K バイトを越えることも考えられますが、IMS レジスタをどう設定すればいいですか。

A1. 16K バイトを越える可能性があれば、IMS レジスタは開発対象製品の値(04)でなく、より大きな ROM 容量の製品の値(たとえば、オンチップ・ディバグに使用するμPD78F0503D の値(C8H))に設定してください。開発終了時に、プログラムが 16K バイトに収まれば、開発対象製品の値に変更してください。

Q2. μPD78F0501 をターゲットにして、μPD78F0503D 用の値を IMS レジスタに設定して、オンチップ・ディバグで開発します。IMS をそのままで量産していいですか。プログラムはμPD78F0501 の内蔵 ROM に収まります。

A2. 2 つ問題があるので、IMS レジスタは開発対象品の値に変更してください。

まず、内蔵 RAM へのアクセスです。アドレスの**直接指定**(**ダイレクト・アドレシング**や**ショート・ダイレクト・アドレシング**など、変数名で指定し、MAP 情報でアドレスが明確)だけでなく、**レジスタ間接指定**(レジスタの内容が RAM のアドレスを示し、最終的なアドレスは命令実行まで不明)などでアクセスすることがあります。そのとき、開発対象品にないアドレスでは、実行をすれば問題が発生します。ところが、オンチップ・ディバグ用のμPD78F0503D では、そのアドレスにも RAM があるので、ディバグでは分かりません。

もうひとつは、ROM の内容をデータとして参照する場合です。このとき、開発対象製で存在しないアドレスを参照すると、アクセス結果が異なることになります。

IMS レジスタを開発対象製品の値に設定すると、本来は存在しない ROM アドレスの参照が、ウォッチドッグ・タイマの機能で検出できます(ただし、内部高速 RAM 領域での異常アドレス・アクセスは検出できません)。

Q3. 小ピン・マイコン(78K0S)は、製品ごとに内蔵メモリ容量が異なるのに、なぜ IMS レジスタがないのですか。

A3. 小ピン・マイコンでは、オンチップ・ディバグ専用のデバイスでなく、対象製品でディバグができます。このため、IMS レジスタのような機能を用いなくても、メモリ構成を対象の製品と同じにできるからです。

C 言語でのマイコン開発 [78K0、78K0S 共通]

Q1. C 言語でのプログラム開発で、CC78K0 を用いるには、ポートの制御をどのように記述すればいいですか。また、ユーザーズ・マニュアルのレジスタの説明で、ビット番号が□で囲まれたものとそうでないものの制御は、どうちがいますか。

A1. CC78K0 や CC78K0S では、デバイスのユーザーズ・マニュアルに記載されたレジスタ名称を C 言語のプログラム中で使えます。そのためには、プログラムの頭の部分で、「#pragma sfr」と宣言してください。そうすると、内蔵周辺レジスタを変数のように使用できます。次のプログラムは、ポート制御などの記述例です。

```
① port_data = P2;          /* ポート 2 の状態を変数 port_data に読み込み */
② PM1 = 0b00000000;        /* ポート 1 の 8 ビットｽﾍﾞてを出力に設定 */
③ P1 = 0x55;               /* ポート 1 に 0x55 を出力 */
④ P1.0 = 0;                /* ポート 1 のビット 0 を"0"に設定 */
⑤ P1.7 ^= 1;               /* ポート 1 のビット 7 を反転 */
⑥ while(P2.0 == 0);        /* ポート 2 のビット 0 が"1"になるのを待つ */
⑦ while(TMIFH1 == 0);      /* タイマ H1 のインターバル割り込みを待つ */
   TMIFH1 = 0;             /* タイマ H1 の割り込み要求をクリア */
```

①では、ポート 2 の端子の状態を読み込んで、その値を変数"port_data"に代入します。
このような記述で、ポート(端子)の状態を取り込めます。
②は、ポート 1 を出力に設定するために、ポート・モード・レジスタ 1 の各ビットを 0(出力)に設定しています。
③は、全ビットを出力に設定してあるポート 1 に対して、16 進数で 55 というデータを出力します。
これで、ビット 0, 2, 4, 6 はハイ・レベルになり、ビット 1, 3, 5, 7 はロウ・レベルになります。
④は、ポート 1 のビット 0 の値だけを"0"にして、出力をロウ・レベルにします。
⑤は、ポート 1 のビット 7 の出力だけを反転させます。
⑥では、ポート 2 のビット 0 がロウ・レベルの間は while でループするので、ポート 2 のビット 0 の入力がハイ・レベルになるのを待つことになります。
⑦は、タイマ H1 を用いたインターバル・タイマの時間経過を割り込みでなくポーリングで待つ処理です。
事前に指定されたインターバルが経過すると、タイマ H1 は割り込み要求フラグをセットします。ここでは、割り込み要求フラグが"0"の間は while でループするので、インターバルの経過までを持つことになります。
割り込み要求が発生してループを抜けると、割り込み要求をクリアします(この 2 つのステートメントをセットで使用します)。
この例の⑥と⑦の while 文のループ条件を比べてください。
⑥では、"P2.0"と「"ポート番号"＋"."＋"ビット番号"」で特定のビットを指定しています。ところが⑦では、"TMIFH1"とビット名称を直接記述しています。これがデバイスのユーザーズ・マニュアルでレジスタの説明中のビット番号が□で囲まれたものと、そうでないものの使い方のちがいです。"TMIFH1"は IF0H レジスタのビット 3 なので、"IF0H.3"と記述してもいいのですが、"TMIFH1"の方が何を指しているか分かりやすいです。
ところが、"P2.0"のような記述では、ビットに機能ブロックの意味がないので、□はついていません。たとえば、"P2.0"がモータの電源オン／オフの確認に使っているような場合は、次のように目的に応じて名前を定義する方が、"P2.0"を"P20"とするより分かりやすいです。

```
#define MOTORONOFF      P2.0    /* モータ電源確認ポート */
while(!MOTORONOFF);
```

Q2. C 言語でのプログラム開発で、ほかに注意点はありますか。

A2. CC78K0 や CC78K0S はオプション・バイトに対応していないので、オプション・バイトについてはアセンブラで記述してください。詳細は、オプション・バイトの設定 [78K0S/Kx1+,78K0/Kx2]をご参照ください。

Q3. CC78K0 を使用する場合、割り込み処理の記述はどうすればいいですか。

A3. 前述のタイマ H1 の割り込みを処理する場合について紹介します。ただし、タイマ H1 の設定そのものは省略して、割り込みに関係した部分だけを説明します。

割り込みを使用するには、プログラムの頭の部分で、そのための宣言をする必要があります。通常は、次のような宣言となります。この 2 行目と 3 行目は、割り込みを使う場合に必須となる、割り込みの受け付けの許可や禁止を行う命令を使用可能にするための宣言です。4 行目が割り込みの宣言です。**#pragma interrupt** の部分が、割り込み関係の宣言であることを示します。次の **INTTMH1** の部分は、タイマ H1 の割り込みに関する宣言であることを示します。**fn_intTimerInterval** の部分は、**INTTMH1** の割り込みが発生したときに、実際に起動する割り込み処理関数の名前を示します。

```
#pragma      sfr
#pragma      ei
#pragma      di
#pragma      interrupt INTTMH1 fn_intTimerInterval
```

その上で、割り込み処理関数の本体部分は以下のように記述します。

```
__interrupt void fn_intTimerInterval(void){
ここに、割り込みの中で実行する処理を記述
}
```

これで、割り込み処理そのものの記述は終わりです。あとは、メイン関数の中で、タイマ H1 を起動したあとで次のように記述すれば、割り込み受け付けの準備は完了です。

```
TMIFH1 = 0;        /* 割り込み要求フラグをクリア */
TMMKH1 = 0;        /* 割り込みマスクの解除 */
ei();              /* 割り込み受付許可 */
```

Q4. V850 の多重割り込みでは「__multi_interrupt」修飾子を使用しますが、CC78K0 にはありません。多重割り込みはできないのですか。

A4. いいえ、多重割り込みはできます。

78K0 や 78K0S では、割り込み要求を受け付けると、プログラム・カウンタや PSW の値をスタックにセーブします。ところが、V850 では割り込みを受け付けると、プログラム・カウンタや PSW の値は特定のレジスタにセーブされます。そのため、多重に割り込み要求を受け付けさせるには、この特定のレジスタにセーブされた内容をスタック領域に退避させる必要があります。

この処理を指定するのが「__multi_interrupt」です。CC78K0 では、このような処理が必要ないので、この修飾子はありません。多重割り込みを行いたい場合、それを許す割り込み処理の中で割り込みマスクを制御して、割り込みを許可する（ei();を記述する）だけです。多重割り込みについては、割り込みの優先度制御の注意点[78K0S、78K0、78K0R]をご参照ください。

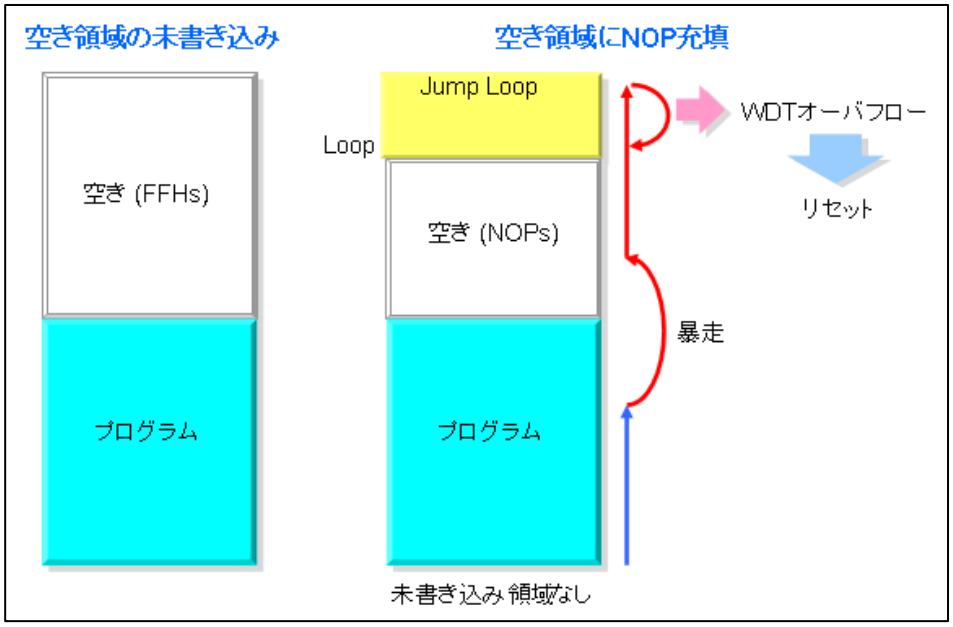
★

プログラミング

Q1. プログラム書き込みをしない空き領域は、どのように処理すればいいですか。

A1. 空き領域はアクセスしないので、特にどうしなければならないということはありません。ただし、PROM やフラッシュ・メモリの初期状態は FFH になっているので、プログラムを書き込まない空き領域は FFH のままです。なお、当社でのマスクROM受注やフラッシュ書き込み受注の場合、未使用領域が不定ではベリファイ工程で問題が発生する可能性があるので、未使用領域を PROM やフラッシュ・メモリの初期値と同じ FFH で埋めていただくように決めています。ただし、未使用領域を NOP 命令のコードで埋めれば、そこもプログラムの一部と考えられ、未使用領域はなくなるので、「未使用領域を FFH で充填した」とみなせます。

この領域のコードによって、CPU の動作に違いを生じる可能性があるのは、ノイズによるプログラム・カウンタの内容化けやプログラム・ミスによる暴走のときです。これによって、どこに飛んでしまうかは特定できませんが、飛び先がプログラムの空き領域であれば、その内容を命令コードとみなして実行しようとします。
78K/0 シリーズでは、FFH が CALLT 命令(addr5=007EH)なので、エラー処理ルーチンを用意しておいて、007EH,007FH 番地にそのアドレスを書き込めば、エラー処理へ分岐することになります。
他の 78K シリーズや V850 シリーズでは、FFH が未定義コードだったり例外トラップ対象だったりするので、空き領域を NOP 命令で埋めて、末尾にエラー処理ルーチンを用意（全域をプログラム化）する処置が考えられます（78K/0 でも有効）。このとき、V850、78K0/Kx2 シリーズなどでは、特にエラー処理ルーチンを用意しなくても、プログラム領域を越えてフェッチしようとしたときにリセットがかかります（この機能がない 78K シリーズでは、フェッチ・コードが特定できず動作不定）。



なお、誤動作（内部各ブロックの状態を特定できない）が原因なので、エラー処理での表示（データ出力）動作などは保証されません。最終的にはリセットによる再起動が必要なので、エラー処理の最後は無限ループにして、ウォッチドッグ・タイマでリセットすれば、デッドロックは回避できます。

以上は、たまたま暴走時の飛び先がプログラムの空き領域の場合の動作です。アプリケーション・プログラム書き込み領域や、78K/0、78K/0S シリーズでプログラム領域でない領域に飛べば、動作は特定できません。ウォッチドッグ・タイマでのリセットは効果的ですが、ノイズの影響を受けないハードウエアやミスのないソフトウエアといった設計品質が重要です。

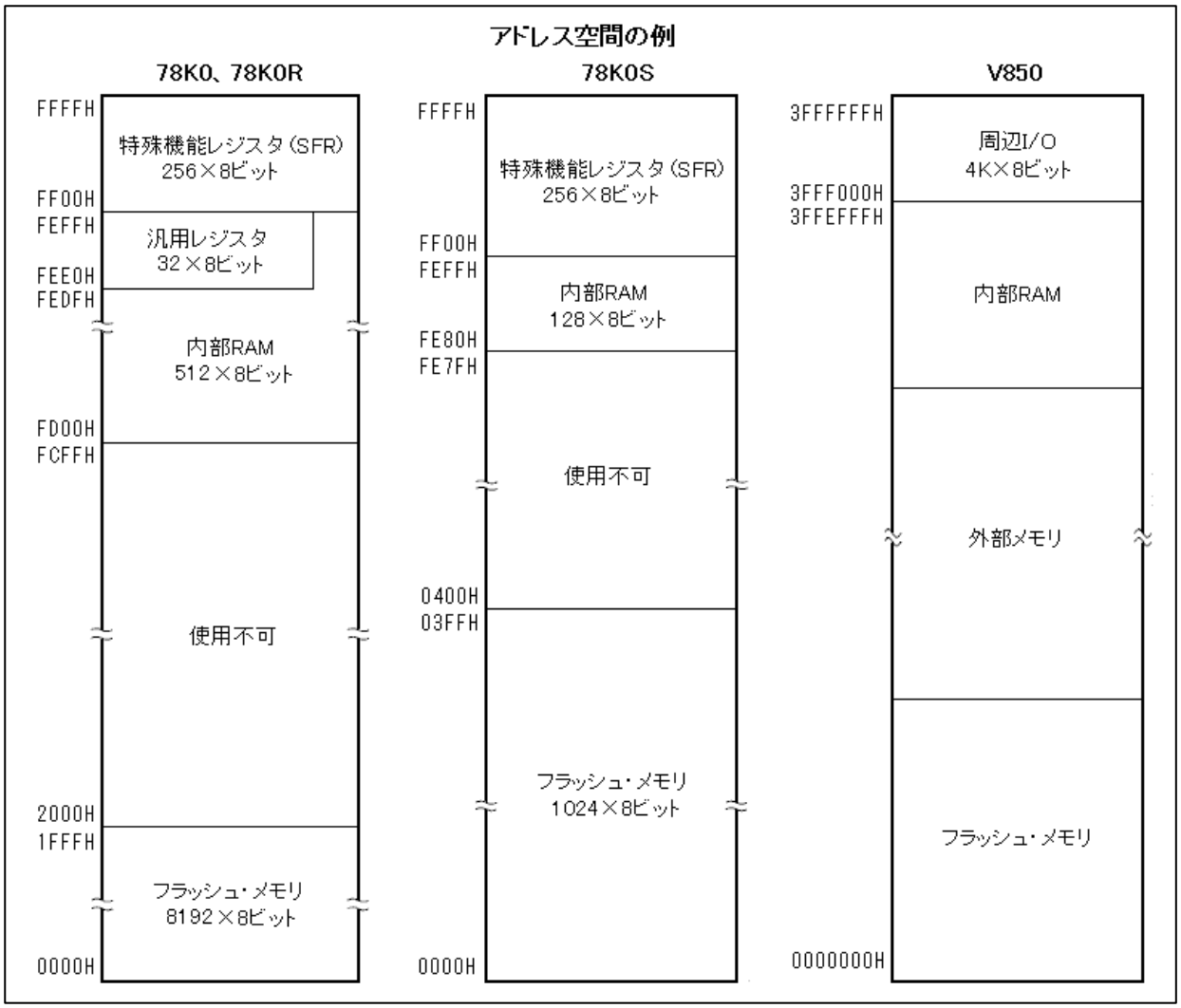
Q2. スタック操作では SP−1 以前の領域をアクセスするので、スタック・ポインタの初期値を内部 RAM の次のアドレスに設定してもいいですか。

A2. 78K0 と 78K0R では、内部 RAM の末尾がレジスタ領域になっているので、スタック領域を割り当てられません。SP の初期値をレジスタ領域の先頭アドレス(FEE0H)に設定すれば問題ありません。

78K0S でも、スタック操作には問題がありません。内部 RAM の末尾にスタック領域を割り当てられますが、次が SFR 領域になっており、SP の初期値を FF00H に設定しても、スタック領域 1K バイトの先頭 FB00H に変換されます。ここで PUSH 命令を実行すると、データを退避させる FAFFH が内部 RAM 領域でないため、末尾の FEFFH に再変換され、FF00H に設定した通常のアドレシングと同じ結果になります。ただし、スタック領域の読み出しをすると、FB00H に変換されたアドレスをアクセスするため、正しいデータが得られません。このため、78K0S でスタック領域の読み出しをするプログラムでは、スタック領域を FF00H に割り当てられません。

78K シリーズのスタック設定については、スタック・ポインタの設定をご参照ください。

V850 では、内部 RAM の次が周辺 I/O 領域ですが、SP の初期値をその先頭アドレスに設定しても、内部 RAM 領域の設定と同じ動作をします。



1.1.25 開発ツール部品

変換ソケット EV-9200G-80 の重量と寸法

Q1. 変換ソケット EV-9200G-80 の重量と寸法を教えてください。

A1. 実測値として以下のとおりです。

重量　：　80g

寸法　：　132×95×27（mm）

なお、この測定値はあまり厳密なものではありません。

変換ソケット EV-9200G-80 の突起

Q1. ソケットに 2 本の突起がありますが、基板に穴を空けたくないので、切断してもいいですか。

A1. この突起は、基板への位置合わせ用のものです。

この突起をなくした場合には、はんだ付けのときに位置がずれて、隣同士のピンがショートすることが考えられますので、位置合わせや位置固定に十分ご注意ください。

EV-9200G-80 のリフロ実装

Q1. EV-9200GC-80 の基板実装はリフロ可能ですか。

A1. EV-9200GC-80 は、リフロには対応できません。

この部分だけは、手付けでのはんだ付けをお願いします。

1.1.26 フラッシュ書き込み

フラッシュ書き込みの基本的事項

Q1. フラッシュ書き込みの基本的事項を教えてください。

A1. (1) 概　要

マイコン内蔵のフラッシュへの書き込みには、大きく分けて次の2種類があります。

・チップ単体での書き込み

・ターゲット・システム上のデバイスへの書き込み

これらは、ターゲット・デバイスの状態の違いだけで、基本的な書き込み方法は共通です。

これ以外に、一部のデバイスでは自分自身で書き込みを行うセルフ書き込みをサポートしていますが、ここでは触れません。

チップ単体での書き込みでは、ライタに加えて、ターゲット・デバイスをセットするためのソケット・アダプタが必要になります。この部分は（株)内藤電誠町田製作所のアダプタ(FA シリーズ)を用いると簡単です。

ターゲット・システム上のデバイスの書き込み（オンボード書き込み)では、電源の供給されたターゲット・システムとライタの間で必要な信号をやりとりするだけです。

(2) 必要なもの

書き込みに必要なものは、次のとおりです。

・ライタ（例　ルネサス エレクトロニクス：PG-FP4、内藤電誠町田製作所：FL-PR4）
・パラメータ・ファイル（開発ツール・ダウンロードよりダウンロード）
・制御するための PC
・必要な配線の施された FA アダプタ（チップ単体書き込みの場合）
　→オンボード書き込みではターゲット・システム

(3) デバイスとライタのインタフェース

書き込むためには、デバイスとライタの間で信号をやりとりを行います（基本的には、次の 3 種類）。

・電源／制御関係（V_{DD}, V_{PP}, RESET）
・シリアル信号
・動作クロック（必要な場合）

電源／制御関係信号の中で V_{PP} は、書き込みの電源として使用する他に、デバイスとライタの間のシリアル通信にどの方法を使用するかを指定するためにも用います。
また、V_{DD} については、チップ単体での書き込みの場合にはライタ側から供給できますが、オンボード書き込みの場合には、ボード全体を駆動するだけの容量はありませんので、別途電源が必要です。

シリアル信号としては、実際に使用する通信方法に合わせて信号を選択してください。
オンボード書き込みでは、空いているシリアル・ポートのどれかを使用することになります(ターゲット・デバイスによってサポートされている通信方法が異なりますので、個々のデバイスに合わせて選択してください)。
なお、最も書き込み時間を短くできるのは、ハンドシェイク付きの 3 線式シリアル通信です。
ただし、これは FL-PR2 ではサポートされていません。

デバイス単体での書き込みでデバイス側にクロック発生機能がない場合、動作クロックは、ライタ側から供給できます。ただし、周波数はある程度制限されますので注意が必要です。オンボード書き込みでは、基本的にクロックはターゲット・システムで発生しますので、ライタ側から供給する必要はありません。

(4) デバイスとライタの基本的なやりとり(初期化)

書き込みを行う場合に、ライタはまず、デバイスに V_{DD} を供給します。
電源の立ち上がり時間ののちに V_{PP} を供給し、立ち上がり時間を待ったあとで RESET を解除します。
その後、クロック発振安定時間待ちを行います。

デバイスは、RESET 解除とクロック供給(発振)により、フラッシュ書き込みモードとなります。 このあと、ライタは V_{PP} をパルス状に変化させることで、どのようなインタフェースで通信を行うかをデバイスに通知します。
この処理は限られた時間以内に行わないといけないため、ライタが RESET を解除してから、実際にデバイスの RESET が解除されるまでの時間を大きく遅延させるような回路が入らないようにしてください。
インタフェース条件の設定が完了すると、ライタはシリアル信号を使って、デバイスに初期化コマンドを最大 16 回送信します。
16 回以内にデバイスから ACK 信号が戻れば、デバイスの通信準備ができたことになります。
ここまでが正しくできなかった場合には、エラー・メッセージとして“Target initialize error”が表示されます。

(5) 書き込みとベリファイ

通常、書き込みには EPV コマンドを使用しますが、このV(ベリファイ)処理はデバイス内部のベリファイで、個々のデータのベリファイはしません。書き込みデータを確実に確認するには、書き込み後に単独のベリファイ・コマンドを実行してください。

なお、フラッシュの書き込みデータは、プログラム保護の観点から外部に読み出せません。

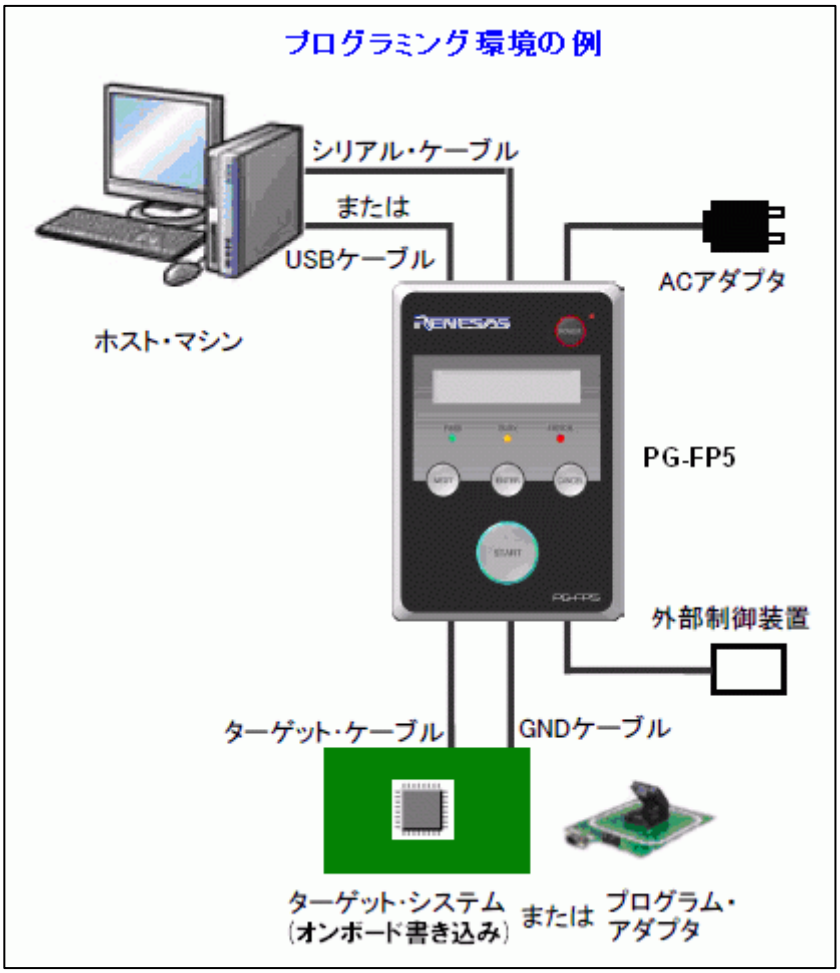
また同じ観点から、アドレスを指定するベリファイはできません。

(6) パラメータ・ファイル

パラメータ・ファイルは、デバイスごとに準備されています。また、デバイスのバージョンや使用するクロック、インタフェース条件によって異なることがありますので、書き込みデバイスに合ったものをご使用ください。

Q2. 高価なプログラマを購入しないで、フラッシュ・メモリにプログラミングをする方法はありませんか。オンボード書き込みならできますか。

A2. 書き込み制御が対象デバイスによって異なり、容易にはできないため、専用のプログラマやプログラミング機能付きオンチップ・ディバグ・エミュレータ MINICUBE2 のご利用をお勧めします。これは、オンボード書き込みでも同様です。また、マイコンのセルフプログラミング機能では、その機能を実現するための最初の書き込みにプログラマが必要となるので、やはりプログラマが不可欠です。



サード・パーティからも各種プログラマが提供されています。

なお、小ピンマイコンの 78K0S/KA1+については、サンハヤト社から簡易ライタ・パーツ・セットが発売されています。

エミュレータとフラッシュ品で動作が異なる

Q1. エミュレータ上で動作するロード・モジュール・ファイルと同時に生成された HEX ファイルをフラッシュに書き込むと、全く動作しません(リセット解除しても、全く何も動作せず)。

A1. 同じオブジェクトがエミュレータでは動作して、フラッシュで動作しない場合、フラッシュの書き込みに問題がなければ、以下の点に問題があることが考えられます。

(1) 初期値ありの変数が、正しく初期化されていない(ROM 化が正しく行われていない)。
(2) クロックが正しく発振していない。
(3) パワーオン・リセットがおかしい(時間が短すぎる)。
(4) 本来デバイスにないリソースまで使用している
(5) 電源の容量に問題がある(エミュレータでは、電源を監視するだけで、ターゲットの電源は使用せず)。

これらに加えて、フラッシュの場合には、次の要因もあります。

(6) フラッシュに正しく書き込まれていない。
(7) V_{PP} 端子が正しく処理されていない(マスク品では、IC 端子が正しく処理されていない)。

(2)、(3)のハードウエア問題については、エミュレータでは評価できない項目です。
ハードウエアの要因としては、以上の点をご確認ください。
簡単なプログラムが動作したのであれば、ハードウエア問題の可能性は少ないと思われます。
(4)はエミュレータの設定にも関係しますが、プログラムのサイズがデバイス以上になっており、エミュレータではエミュレーション・メモリが割り当てられているということはないでしょうか。

フラッシュ・メモリ・ライタとの接続

Q1. μPD78F0058 に、オンボードで書き込むための接続について教えてください。

A1. 次のような端子処理をしてください。

端　子	処　　理
V_{PP}	スイッチかジャンパで、書き込み時にはライタと接続、 通常動作時にはグランドに接続できるようにする。
RESET	ボード上のリセットとライタ側の、どちらからでもリセットがかけられるようにする。 または、書き込み時にはライタ側に、通常動作時にはボード上の回路に切り替えられるようにする。
X1、X2	ボード上のクロックが使える状態のままにしておく(何もしない)。
シリアル系	書き込みに使用するシリアル・チャネルをライタと接続できるようにする。
V_{DD}、V_{SS}	ライタと接続できるようにする。

その他の信号は、通常使用状態のままでけっこうです。

書き込み中のポート状態

Q1. オンボードでフラッシュ書き込み中、書き込み関係以外のポート状態はどうなっていますか。

A1. フラッシュ書き込みに使用しないポートは、リセット後の状態、つまり**入力ポート**になっています。

フラッシュへの書き込み時のポート設定

Q1. フラッシュへの書き込みの際、シリアルなどのポート設定は必要ですか。

A1. 設定は不要です。フラッシュ書き込みモードにすると、内蔵のファームウエアが書き込みに必要な設定を行いますので、書き込みに使用する信号をライタと結線するだけでけっこうです。

データ転送モード(3 線シリアルや UART)の選択

Q1. μPD78F0034A でデータ転送方法には、3 線シリアルと UART がありますが、マイコン側はどういう方法で、データ転送方法を判断するのですか。

A1. フラッシュ書き込み動作の最初に、V_{PP} のパルス数で指定しています。
詳細については、データ・シートの「6 .フラッシュ・メモリ・プログラミング」をご参照ください。

オンボード書き込み時のクロック

Q1. ターゲット・システムのボード上に水晶振動子が実装された状態（X1、X2 に接続）で、フラッシュ・ライタから書き込みができますか。

A1. できます。この場合には、水晶の有無を気にする必要はありません。
ただし、All Flash マイコンの小ピン・マイコンには適用できません。

書き込み時間とシリアル・クロック速度の関係

Q1. フラッシュへ書き込む際に、シリアル・クロック周波数を 1MHz から 2MHz に変更しましたが、書き込みの時間はほとんど変わりませんでした。

A1. クロック周波数はデータ転送時間についてのみ影響しますが、書き込み時間を決定する要因としては、ほかに、内部の処理速度やパラメータ・ファイルで設定された書き込み条件が関係します。
シリアル・クロックを倍にしても、半分の書き込み時間にはなりません。

UART のシリアル・クロックが 4.19MHz では、データ転送ができない

Q1. μPD78F9116 の UART による書き込みが、4.19MHz ではうまくできません。5MHz にすると書き込めます。

A1. ライタ側からの 00 のデータ送信信号のパルス幅を測定して、ボー・レートの設定に必要なパラメータを決定しています。このとき、CPU はファームウエアで時間測定を行い、ボー・レートのパラメータを BRGC レジスタに設定することになります。
このとき設定する値に対して、ボー・レートと BRGC の設定値、各クロックでの誤差は次のようになります。

ボー・レート	BRGC の値	誤　差（%）		
		5.0MHz	4.9152MHz	4.1943MHz
9600	40H	1.73	0	−14.6
19200	30H	1.73	0	−14.6

つまり、4.1943MHz のクロックは、誤差が大きくて使用できません。

フラッシュ書き込み時のクロック結線(CLK→X1)

Q1. μPD78F0058 のデータ・シート p.23～p.24 に、CLK→X1 の接続が記載されていますが、水晶振動子による発振の場合、この接続は不要ですか。

A1. 不要です。これは単に、ライタ側からクロックを供給する場合の例を示しています。
したがって、通常の使い方で、デバイス自身がクロックを発振する場合は、この結線は不要です。

通常動作時の V_{PP} 端子接続

Q1. 通常動作時、V_{PP} 端子は Vss に直接接続でなく、抵抗で接続することは可能ですか。

A1. フラッシュ品の場合、ライタのドライブ能力と、デバイスのノイズ耐性の両方を考慮すると、 10kΩ 程度の抵抗で Vss にプルダウンすることは可能です。ただし、実装時のノイズ耐性の確認評価が必要です。

マスク品の場合、V_{PP} に相当する端子はテスト端子となっているため、V_{PP} は必ず Vss に直接接続する必要があります。直接接続しない場合には、動作保証ができません。

オンボード書き込みでのイニシャライズ・エラー

Q1. FL-PR3 を使用した μPD78F9488 のオンボード書き込みで、イニシャライズ・エラーが発生します。

A1. イニシャライズ・エラーは、デバイスとライタが全く通信できない状態であることを示しています。
これは通常、結線に問題がある場合や、デバイスが動作してない場合に発生します。

ユーザーズ・マニュアルの p.281 の結線は、通常の書き込みの時の信号の結線を示したものです。
オンボードでの書き込みでは、電源やクロックはボード上で供給し、ライタからのクロックは配線しません。
また、このとき FL-PR3 の TARGET VDD SWITCH は、IN 側に設定してください。

そのほか、デバイスの V_{PP} や リセットは、書き込み時には FL-PR3 に切り替えてください。
この状態で、CLK 以外の信号がマニュアルの記述のように結線してあり、クロックの発振に問題なければ、書き込めるはずです。

書き込み時の“Target SIGNATURE Error”

Q1. FlashProIII を使用した μPD78F4216A の書き込みで、Target SIGNATURE Error が出ます。

A1. パラメータ・ファイルは、正しいものをお使いでしょうか？
μPD78F4216 と μPD78F4216A（さらに規格によって）では書き込みの方法が異なり、必要なパラメータ・ファイルが異なります。デバイスと書き込み方法に合ったパラメータ・ファイルをご使用ください。
パラメータ・ファイルについては、マイコン開発ツールのバージョンアップのホームページ（URL : https://www5.necel.com/micro/tool_reg/OdsListTop.do?lang=ja）で、78K4 を選び、78F4216A のパラメータ・ファイルの中から、コメントを参照して、目的に合ったものをダウンロードしてご使用ください。

フラッシュ ROM 品からマスク ROM 品への置き換え

Q1. フラッシュ品の μPD78F9177Y を使って開発し、マスク ROM 品への置き換え時に、プログラム上修正が必要な点はありますか。

A1. 特に修正が必要なところはありません。
アセンブルなどを行う場合は、μPD789177Y のデバイス・ファイルを使用して、フラッシュ品で動作確認を行ってください。

内蔵フラッシュは 20 回以上書き換えできないか

Q1. フラッシュ・マイコンのデータ・シートで、書き換え回数が MAX.20 回 になっていますが、デバッグ段階でも 20 回以上の書き換えはできないのですか。

A1. いいえ、違います。「20 回」は、書き込み後に 10 年間のデータ保持を保証できる回数です。
デバッグ段階など、このような保証が必要ない場合には、20 回を越えて書き込んでも問題ありません。

書き込まれなかった領域のデータ値

Q1. μPD78F0034A に FlashPro で書き込む(EPV)と、書き込まれなかった領域のデータはどうなりますか。

A1. 0xFF となります。

内蔵フラッシュ ROM に書かれた内容の読み出し

Q1. 内蔵フラッシュ ROM に書き込まれたデータを読み出す方法はありますか。

A1. ライタを使って、内蔵フラッシュに書かれた内容を読み出すことはできません。
書き込まれたプログラムをセキュリティ保護するために、このようになっています。

セルフ書き込みは可能か [μPD78F0058]

Q1. μPD78F0058 の内蔵フラッシュ・メモリに、ソフトウエアから書き込みはできますか。

A1. 自分自身のプログラムでの、フラッシュの書き換えはサポートしていません。
ファームウエアの書き換えは、外部にライタ(FlashPro)をシリアル通信で接続して、ライタで行ってください。

フラッシュ書き込み時間

Q1. 書き込み時間とベリファイ時間を教えてください。

A1. 概略時間を紹介します（CPU クロックと通信は最高速）。

V850 シリーズ

愛称	品名	メモリ・サイズ（バイト）	条件		書き込み時間（参考値）	
			CPU クロック	通信方式	消去→書き込み[注]コマンド	ベリファイ・コマンド
V850E/Ix3	μPD70F3454	256K	8MHz×8 逓倍	CSIB0+HS：5MHz	7.5 秒	3.5 秒
	μPD70F3453	128K	8MHz×8 逓倍	CSIB0+HS：5MHz	4.5 秒	2 秒
V850ES/Hx2	μPD70F3712	512K	5MHz×4 逓倍	CSIB0+HS：5MHz	14.5 秒	7 秒
	μPD70F3710	256K	5MHz×4 逓倍	CSIB0+HS：5MHz	8 秒	4 秒
V850ES/Hx3	μPD70F3757	512K	6MHz×8 逓倍	CSIB0+HS：5MHz	7.5 秒	5 秒
	μPD70F3755	256K	4MHz×8 逓倍	CSIB0+HS：5MHz	5.5 秒	3.5 秒
V850ES/Ix2	μPD70F3714	128K	2.5MHz×8 逓倍	CSIB0+HS：5MHz	5.5 秒	2.5 秒
	μPD70F3713	64K	2.5MHz×8 逓倍	CSIB0+HS：5MHz	3 秒	1.5 秒
V850ES/Jx2	μPD70F3724	640K	5MHz×4 逓倍	CSIB0+HS：5MHz	15 秒	8.5 秒
	μPD70F3722	384K	5MHz×4 逓倍	CSIB0+HS：5MHz	9.5 秒	5.5 秒
V850ES/Jx3-L	μPD70F3738	256K	5MHz×4 逓倍	CSIB0+HS：5MHz	7.5 秒	4.5 秒
	μPD70F3737	128K	5MHz×4 逓倍	CSIB0+HS：5MHz	3.5 秒	2.5 秒
V850ES/Jx3	μPD70F3746	1M	4MHz×8 逓倍	CSIB0+HS：5MHz	16 秒	11 秒
	μPD70F3744	512K	4MHz×8 逓倍	CSIB0+HS：5MHz	8 秒	6 秒
V850ES/Kx2	μPD70F3734	256K	5MHz×4 逓倍	CSI00+HS：5MHz	25 秒	4 秒
	μPD70F3733	128K	5MHz×4 逓倍	CSI00+HS：5MHz	13 秒	2.5 秒
V853	μPD70F3003A	128K	33MHz	ハンドシェーク3線式	14～25 秒	5～6 秒
	μPD70F3025A	256K	33MHz	3 線式	2 分	30 秒
V850/SA1	μPD70F3017A	256K	20MHz	HS：1MHz	25 秒	5～6 秒
V850/SV1	μPD70F3040	256K	20MHz	ハンドシェーク3線式	16 秒	14 秒

78K0 シリーズ

愛称	品名	メモリ・サイズ（バイト）	条件		書き込み時間（参考値）	
			CPU クロック	通信方式	消去→書き込み[注]コマンド	ベリファイ・コマンド
78K0/Kx2	μPD78F0547	128K	内蔵発振回路	CSI10：2.5MHz	10 秒	4 秒
	μPD78F0545	60K	内蔵発振回路	CSI10：2.5MHz	5 秒	2 秒
78K0/Lx2	μPD78F0397	128K	内蔵発振回路	CSI10：2.5MHz	10.5 秒	4 秒
	μPD78F0395	60K	内蔵発振回路	CSI10：2.5MHz	5 秒	2 秒
78K0/Lx3	μPD78F0495	60K	内蔵発振回路	CSI10：2.5MHz	8.5 秒	4.5 秒
	μPD78F0493	32K	内蔵発振回路	CSI10：2.5MHz	5.5 秒	2.5 秒
78K0	μPD78F0818A	59.5M	8MHz	SIO：1MHz	26 秒	25 秒
				UART：76.8Kbps	27 秒	51 秒
	μPD78F0833Y	60K	4.19MHz	3 線式	1 分 10 秒	20 秒

注 コマンド・オプション「消去前ブランク・チェック」を有効にして、ブランク品で測定

：2 電源（旧タイプ）のフラッシュ・メモリ内蔵品

78K0S シリーズ

愛称	品名	メモリ・サイズ（バイト）	条件		書き込み時間（参考値）	
			CPU クロック	通信方式	消去→書き込み[注] コマンド	ベリファイ・コマンド
78K0S/Kx1+	μPD78F9234	8K	8MHz×1 逓倍	UART:115.2Kbps	3 秒	未サポート
	μPD78F9232	4K	8MHz×1 逓倍	UART:115.2Kbps	2 秒	未サポート
78K0S	μPD78F9046	16K	5MHz	3 線式	15 秒	15 秒
	μPD78F9136B	16K	5MHz	3 線式	15 秒	15 秒
	μPD78F9177Y	24K	5MHz	3 線式	30 秒	10 秒
	μPD78F9850A	16K	8MHz	SIO：1.25MHz	8 秒	8 秒
				UART：76Kbps	17 秒	15 秒

78K0R シリーズ

愛称	品名	メモリ・サイズ（バイト）	条件		書き込み時間（参考値）	
			CPU クロック	通信方式	消去→書き込み[注] コマンド	ベリファイ・コマンド
78K0R/Kx3	μPD78F1168	512K	内蔵発振回路	UART：1Mbps	23.5 秒	14 秒
	μPD78F1166	256K	内蔵発振回路	UART：1Mbps	13 秒	8 秒

78K4 シリーズ

愛称	品名	メモリ・サイズ（バイト）	条件		書き込み時間（参考値）	
			CPU クロック	通信方式	消去→書き込み[注] コマンド	ベリファイ・コマンド
78K4	μPD78F4938A	256K	12.58MHz	ハンドシェーク：1Mbps	1 分 20 秒	10 秒
				通常モード：1Mbps	3 分	30 秒
				ハンドシェーク 3 線式：1Mbps	2 分半	30 秒

注　コマンド・オプション「消去前ブランク・チェック」を有効にして、ブランク品で測定

：2 電源（旧タイプ）のフラッシュ・メモリ内蔵品

1.1.27 プリント基板

基板設計上の注意点

Q1. 回路(パターン)設計上の注意点などありますか。

A1. クロック回路については、誤動作を避けるために最短で配線し、ポートの配線と交差させないでください（ユーザーズ・マニュアルの「システム・クロック発振回路」など参照）。ノイズ・レポートなど、データ・シートやユーザーズ・マニュアルなどに記載されていない情報が必要な場合、お取り引きいただいている販売店へご相談願います。またノイズには、EMS と EMI の観点があります。ノイズに関するご相談の際は、このような具体的な項目を明確にしてください。

プリント基板パッド寸法

Q1. マイコンのプリント基板パッド寸法を教えてください。

A1. 当社ホームページのメイン・メニュー「製品情報/IC　パッケージ」で、ここの左コラムに「パッケージ情報」があります。ここの「IC パッケージ一覧」から参照でき、「Land pattern」に基板パッド寸法が載っています。

1.1.28 パラメータ／デバイス・ファイルのダウンロード

パラメータ／デバイス・ファイルのダウンロード

Q1. パラメータ・ファイルやデバイス・ファイルのダウンロード・サイトで、使用する製品名をどのように探せばいいですか。

A1. 数製品のパラメータ・ファイルやデバイス・ファイルをまとめて、1 つのファイルにしてあり、その中の代表品名がファイル名になっています。したがって、バージョンアップ・サービスのページの「マイコンから選択」の品名別のページで、パラメータ・ファイルやデバイス・ファイルを見つけてください。
前述のとおり、そのファイル名がご使用になる製品名と一致していない場合がありますが、ダウンロードしたファイルを解凍すれば、数製品分のパラメータ・ファイルやデバイス・ファイルを確認できます。

1.1.29 オプション・バイトの設定

オプション・バイト

[はじめに]

マイコン(MCU)では、基本的な機能（命令セットやアーキテクチャ）は各シリーズで共通ですが、パッケージ（端子数）により内蔵機能が異なったり、同じパッケージでも内蔵メモリ（フラッシュ・メモリや RAM）の大きさが異なったりします。さらに、細かな機能（ポートのプルアップ抵抗や POC 電圧など）については、従来のマイコンではマスク・オプションとして、プログラムを ROM に書き込むときに指定できました。でも、PROM 品やフラッシュ・メモリ品では、マスクでのプログラム書込み工程がないので、マスク・オプションがありません。

★ ポートのプルアップ抵抗は、ほとんどソフトウエアで設定できるようになりましたが、POC電圧などソフトウエアで設定できないものも残っています。こういった機能をプログラムとは別のフラッシュ・メモリ領域に、オプション・バイトとして設定しておきます。マイコンは起動時にそれらを参照して、動作モードを確定させてから、プログラムの実行を開始します。

【コーヒー・ブレーク】
ポートのプルアップ抵抗の設定をソフトウエアでできるようにすると、使用時の自由度が大きくなります。特にフラッシュ・メモリを内蔵した製品では、出荷後の変更も可能です。その一方で、マスク・オプションのように、電源投入時から有効というわけではありません。プルアップ抵抗を設定するソフトウエアが実行されるまで、抵抗は有効になりません。
また、POC 電圧は、ソフトウエアを実行可能にするための電源電圧を指定するものなので、これをソフトウエアで設定することは、原理上不可能です。このように、ソフトウエアで設定できるものと、そうでないものの違いを認識するのは重要なことです。



[フラッシュ品での対応]

78K0/Kx1 のように、マスク品とフラッシュ・メモリ品がある製品では、マスク・オプションに対応した個別の製品がラインアップされたものがあります。FAQ の「フラッシュ品の品名 [78K0/Kx1]」をご参照ください。
この場合は、マスク・オプション分だけ製品が必要で、選択を誤ると目的の機能が得られません。特に、途中で関連した仕様を変更すると、使用できるデバイスが違ってしまい、再設計する必要がありました。
All Flash マイコンになって、78K0/Kx2 や 78K0S/Kx1+の 8 ビット・マイコンでは、新たな対応方法が採用されました。これが、オプション・バイトです。指定できる内容や指定フォーマットは、個々のデバイスで異なっています。

[オプション・バイト]

オプション・バイトは、マイコンに内蔵されたフラッシュ・メモリの特定アドレスに割り付けられています（一般のプログラム領域に使用される、先頭のアドレスである 80H 番地から始まる領域が使用されます）。

電源投入やリセット処理によりマイコンが起動すると、最初にオプション・バイトの内容が参照され、指定された設定がマイコンの機能に反映されます。その後、アプリケーション・プログラムが、0 番地のリセット・ベクタで示されたアドレスからスタートします

Q1. μPD78F0533 で、予約された 0x0080-0x0085 に、通常のプログラムを記述したらどうなりますか。

A1. プログラムを記述したつもりでも、デバイスとしては、このアドレスはオプション・バイトの指定と解釈します。その内容が、本来意図されている機能と異なる設定であれば、正常な動作ができなくなります。

78K0/Kx2 や 78K0S/Kx1+のマイコンでは、起動時の動作は次のようになります。

①オプション・バイト領域の内容を参照します。その内容に応じて、対応オプション機能が選択されます。この動作はハードウエアの機能として、必ず自動的に実行され、禁止や変更はできません。

②リセット・ベクタとして指定された番地から、プログラムが実行されます。以降は、ユーザ記述のプログラム実行なので、オプション・バイト領域に配置されたプログラムも、アドレスが指定されると実行されます。

【コーヒー・ブレーク】

78K0/Kx2 や 78K0S/Kx1+は、78K0 や 78K0S のアーキテクチャに含まれるマイコンです。このアーキテクチャにはオプション・バイトがありません（78K0/Kx2 や 78K0S/Kx1+で導入）。オプション・バイトはサポートされていないので、オプション・バイト領域でプログラムが指定されれば、オプション・バイトを命令コードと解釈して命令を実行しようとしてしまいます。また、RA や CC では、まだオプション・バイトをサポートしていません。そこで、オプション・バイトだけを指定したファイルを別途準備しておき、ビルド（アセンブルやコンパイルしてリンク）するときにそれを組み込むことで、簡単に対応できます。具体的な方法は、FAQ の「オプション・バイトの設定 [78K0S/Kx1+,78K0/Kx2]」をご参照ください。こうすれば、オプション・バイトを特に意識せずにプログラムを組むことができます。



オプション・バイトの設定 [78K0S/Kx1+、78K0/Kx2]

Q1. C 言語でオプション・バイトをどのように指定すればいいですか。

A1. C 言語ではオプション・バイトを記述できません。またオプション・バイトは、リセット解除後、プログラムの実行前に、ハードウエアで内容が参照されます。このため、プログラムでは書き換えられません。

オプション・バイトは、事前にフラッシュ・メモリの 80H番地へ書き込んでおく必要があります。そのためには、プログラムの一部としてコードを準備します。一番簡単な方法は、オプション・バイトの定義部分だけを「DB 擬似命令で設定」したアセンブラ記述のファイルで準備しておき、それをソースとして設定しておくことです。

たとえば 78K0S/Kx1+なら、次のように記述したファイルを準備します。この名前を opt.asm とすると、PM+の「プロジェクト」メニューの「ソース・ファイルの追加(S)」メニューで、opt.asm をソース・ファイルとして追加するだけです。新規にプロジェクトを作る場合、初期設定でソース・ファイルを追加する際でも指定できます。

```
@@OPTB   CSEG     AT       0080H
DB       11111001b                ; オプション・バイト
DB       11111111b                ; プロテクト・バイト
END
```

オプション・バイトやプロテクト・バイトについては、使用される各デバイスのマニュアルをご参照ください。

1.1.30 スクリーニング

スクリーニング [フラッシュ・マイコンや OTP マイコン]

Q1. OTP マイコンでは、プログラムの書き込み後にスクリーニングが必要でしたが、フラッシュ・マイコンではどうすればいいですか。

A1. フラッシュ・マイコンでは、プログラムの書き込み後のスクリーニングは必要ありません。

Q2. OTP マイコンではスクリーニングが必要で、フラッシュ・マイコンでは不要なのはなぜですか。

A2. OTP マイコンでは、組み立て工程の前までは、書き込んでも消すことができるので、PROM 部分のテストは行えます。ところが、組み立ててしまうと、PROM の内容を消すことができなくなるので、組み立て工程以降は PROM の書き込みテストができません。そこで、お客様がプログラムを書き込んだ後に、スクリーニングを行っていただくことで、組み立て工程での PROM の不良発生を除こうとしています。

一方、フラッシュ・マイコンでは、組み立て工程以降でも、PROM 部分に書き込んで消去ができます。つまり、製造工程で組み立て工程での不良はすでに除去されています。そのため、お客様がプログラムを書き込んだ後のスクリーニングは必要ありません。

1.1.31 電　源

A/D 機能を使用しない場合の VREF 電源供給

Q1. A/D 機能を使用しないで、兼用端子をポートとして使用する場合、VREF 端子に電源を供給しないでいいですか（オープンかグランド接続）。

A1. A/D 入力端子と兼用しているポート端子の電源が、VREF 端子から供給されている製品があります（ユーザーズ・マニュアルの「端子の入出力回路」を参照）。この場合は、VREF 端子に必ず電源を供給してください。ただし、特別なアナログ電源でなく、他のデジタル電源と共通に接続しても問題ありません。

また、内部で他の電源と寄生回路を挟んで接続されている製品もあります。この場合、VREF 端子をグランドへ接続すると、大電流が流れる可能性がありますので、それは絶対におやめください。

レギュレータ

Q1. レギュレータ内蔵のマイコンで、外部コンデンサの容量は、推奨値より大きければいいのですか。

A1. いいえ、容量が大きいと、電源投入時に周辺に対して内部動作の起動が遅れるため、動作保証ができません。

なお、容量や ESR,ESL が小さいコンデンサでは、動作が不安定になって発振する可能性があります。特性にご注意のうえ、ご使用のコンデンサで十分にご評価・判断ください。

低電圧検出回路

Q1. 低電圧検出回路では、検出電圧に幅がありますが、電源電圧がその範囲内で一定の場合、リセットと起動を繰り返すのですか。

A1. いいえ、そのような繰り返し動作はしません。

MIN.と MAX.の間で検出電圧の特性が変動するのではなく、他の規格と同様、個体差があるだけで、個々の製品では固定的な特性です。

ただし、その検出電圧をはさんで電源電圧が上下する場合、検出電圧未満でリセット状態となり、検出電圧を越えるとリセット解除で起動します。このため、低電圧検出回路を利用するセットでは、リセット解除ですぐ起動させないことをお勧めします。

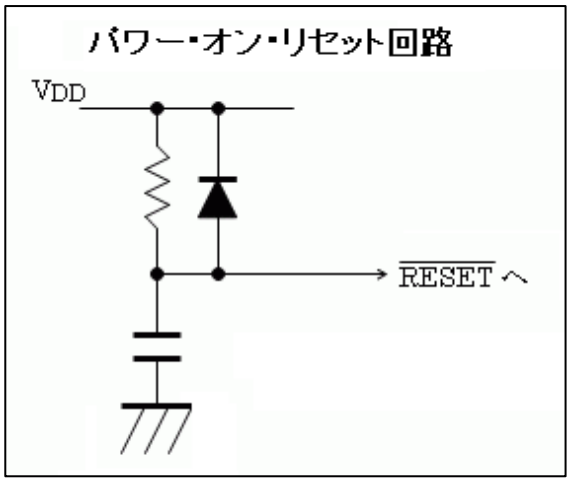
マイコンの電源立ち上げ [マイコン共通]

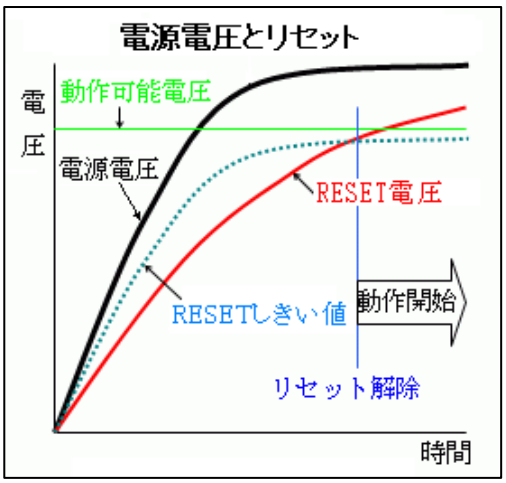
Q1. マイコンの電源について、注意事項はありますか。

A1. 電源電圧の立ち上がり時間に制限があります。

(1) 遅い立ち上がり

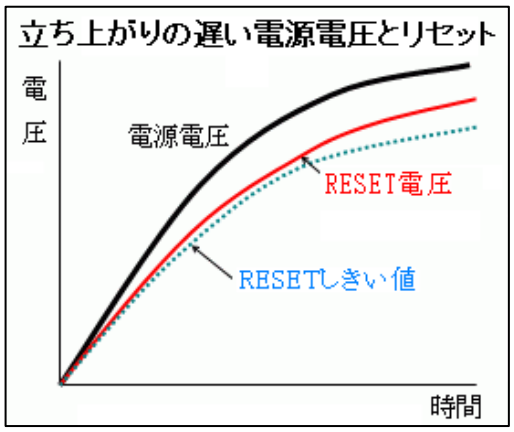
1 つはパワー・オン・リセットとの関係です。電源投入時にリセットをかけるための簡単な方法は、次の回路例のように、CR の時定数で立ち上がりを鈍らせて RESET 信号を生成するものです。このときの電源電圧と RESET 電圧の関係が、その右のグラフです。このグラフのように、電源電圧が立ち上がると、それに従って、マイコンの RESET 入力のしきい値が変化します。マイコンに入力する RESET 電圧は、CR の時定数で立ち上がりが鈍って、このように緩やかに立ち上がります。RESET 電圧が RESET 入力のしきい値より低いとき、マイコンのリセット状態が継続します。しきい値を越えるとリセットが解除され、マイコンは動作を開始します。





電源電圧の立ち上がりがさらに緩やかになった場合を考えてみます。RESET 入力のしきい値は電源電圧に比例して変化します。電源電圧の立ち上がりが緩やかになると、すぐに RESET 電圧がこのしきい値を上回った状態になります(次のグラフ)。こうなると、マイコンにはリセットがかからなくなるので、正常に起動できなくなります。これが電源電圧の立ち上がりが緩やか過ぎる場合の問題点です。

この問題を避けるには電源電圧を監視して、ある電圧以下では、RESET 信号を出力するような専用 IC を使う必要があります。

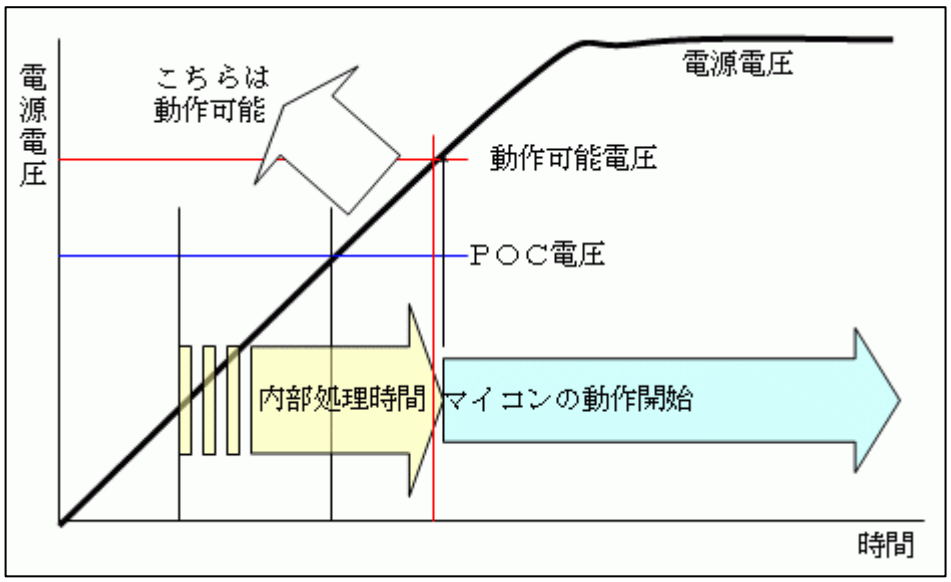


もう一つは、マイコンに内蔵された POC（パワー・オン・クリア）回路との関係です。
最近のマイコンでは、内部に POC 回路と呼ばれる電源電圧を監視する機能を内蔵したものがあります。この場合、電源電圧がどのような値になったらリセットを解除するかが問題となります。最近のマイコンは、幅広い電源電圧範囲で動作が可能となっています。

参考 FAQ：予備知識「電源電圧と動作クロック（Operation Voltage and System clock）」

低い電源電圧での動作を考えると、できるだけ POC 電圧を低くする必要があります。また、電源投入からの起動時間をできるだけ短くする（起動を早くする）ことも重要です。このため、リセット時には外部発振によるクロックでなく、より早く発振可能な内蔵発振器を使用したクロックを利用するものもあります。この場合にも、POC 電圧はできるだけ低くすることが重要です。

この点を意識したのが 78K0/Kx2 で、POC 電圧が選択できるようになっています。78K0/Kx2 の 1.59V POC は、デバイスの動作可能電圧（1.8V）より低い値です。普通に考えると、動作可能電圧より低い電圧で POC によるリセットが解除されると、正常に動作しないことになります。ところが、実際のデバイスでは、電源電圧が POC 電圧を越えても、すぐに動作するわけではありません。POC 電圧を越えたことを検出するための時間や、内蔵レギュレータで電圧を生成するなどの内部処理の時間（POC 電圧と独立）が必要になります。これらの時間を考慮すると、POC 電圧を 1.59V にしても、これらの準備処理中に電源電圧が動作可能電圧になれば、正常に動作可能です。このために規定されたのが、電源電圧の立ち上がりの傾きです。この規格値より電源電圧が早く立ち上がれば、マイコンの準備が完了して動作可能になったときには、電源電圧は動作可能電圧なので、高速の起動が可能となります。



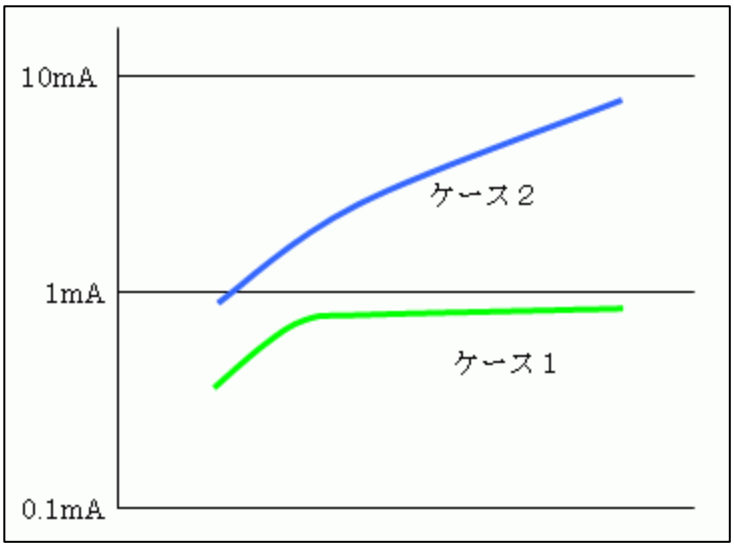
(2) 電源の急激な立ち上がり

逆に電源の立ち上がりが急峻な場合にも問題があります。極端に立ち上がりが早いと、クロック発振ができなくなる可能性が考えられます。ただし、通常の電源ではこのような立ち上げはほとんど不可能なので、無視しても問題はありません。

なお、レギュレータ内蔵のマイコンでは、供給電源の立ち上がりが極端に早かったり遅かったりすると、内部が非同期問題で正常動作できなくなる可能性があり、接続コンデンサの容量を調整する必要が生じるため、そのような電源供給はお勧めできません。

動作電流と電源電圧の関係［マイコン共通］

Q1. ホームページ掲載のマイコンの動作電流のグラフで、ある電源電圧以上で電流がほとんど変わらないデバイス（ケース 1）と、電源電圧が高くなると電流が増加するデバイス（ケース 2）がありますが、どちらかあやまりでないですか。



A1. いいえ、どちらも正しいグラフです。

供給された電圧がマイコン内部でそのまま回路に供給されて動作する場合、電源電流は電源電圧にほぼ比例すると考えられます。これがケース 2 の場合で、古いマイコンはほとんどこの傾向を示します。

一方、最近のマイコンでは、内部に**レギュレータを内蔵**しているものが増えています。このようなマイコンでは、供給された電圧がレギュレータで低い電圧に変換されて、内部回路に供給されます。この場合、レギュレータが正常に動作する電源電圧範囲では、内部回路に供給される電圧が一定となります。このため、外部から供給する電源電圧が変動しても、動作電流はほとんど変化しません。外部から供給する電源電圧が低くなり、レギュレータが内部回路に安定した電圧を供給できなくなると、供給した電源電圧に引きずられて、内部回路に供給される電圧が低下します。このため、ある電圧以下では、電源電圧が低くなると動作電流が少なくなります。

1.1.32 発　振

マッチング

Q1. フラッシュ品とマスク品が用意されているマイコンで、試作にフラッシュ品を使用して、マスク品で量産しようと考えています。この場合、発振回路のマッチング評価は、フラッシュ品だけすればいいですか。

A1. いいえ、フラッシュ品とマスク品では、製造プロセスが異なりますので、マスク品でもマッチング評価をしてください。ただし、回路そのものを変更する必要はなく、コンデンサの値を調整する程度でけっこうです。
なお、フラッシュ品かマスク品で、メモリ・サイズが異なるだけの場合は、再評価をする必要はありません。

発振誤差

Q1. 発振周波数の誤差は、動作クロックの誤差にどのように影響しますか。

A1. クロック・ジェネレータでは、源発振の誤差比率がそのまま出力誤差の比率として逓倍／分周されます。
たとえば、20MHz に対して発振周波数が 20.1MHz（誤差＋0.5%）として 4 分周すると、本来の 5MHz に対して 5.025MHz となり、誤差は＋0.5%です。

1.2 78K デバイス

1.2.1 A/D、D/A 変換

8 ビット A/D と 10 ビット A/D の違い

Q1. 1つのフラッシュ品で、8 ビット A/D 内蔵マイコンと 10 ビット A/D 内蔵マイコンをサポートしていますが、どのように使い分けるのですか？

A1. 対象となるのは、以下のサブシリーズです。

μPD780024A、780034A サブシリーズ（μPD78F0034A）
μPD789104A、789114A サブシリーズ（μPD78F9116A）
μPD789124A、789134A サブシリーズ（μPD78F9136A）
μPD789167、789177 サブシリーズ（μPD78F9177）
μPD789407A、789417A サブシリーズ（μPD78F9418A）
μPD789426、789436 サブシリーズ（μPD78F9436）
μPD789446、789456 サブシリーズ（μPD78F9456）

1 つのフラッシュ内蔵マイコンで、8 ビット A/D を内蔵した製品と 10 ビット A/D を内蔵した製品の評価を兼用します。このために、10 ビット A/D の変換結果の見え方に工夫がなされています。
10 ビットの変換結果は、16 ビットの転送命令で読み出しますが、このデータは 16 ビットの上位 10 ビットに格納されます。つまり、変換結果のレジスタが、sfr のメモリ・マップ上で addr0 と addr0＋1 に配置されていたとすると、次のような内容になります。

addr0 ：上位 2 ビットに、変換結果 10 ビットの下位 2 ビット
addr0＋1 ：変換結果 10 ビットの上位 8 ビット

このため、10 ビットの場合は addr0 をワードで読み出し、8 ビットの場合は addr0＋1 をバイトで読み出します。

ハード的には以上のようになっていますが、実際に使う時には、対象のデバイスを指定すると、8 ビット A/D か 10 ビット A/D かが決まってしまいます（これはデバイス・ファイルで決まります）。
これで、変換結果レジスタ(ADCR0)を指定すると、アドレスとワード／バイトが自動的に決定されますので、どのアドレスをアクセスするかは、意識する必要がなくなります。単に、10 ビットのときの変換結果が、16 ビットの上位 10 ビットに格納されることを知っていれば十分となります。

μPD780021A(8 ビット A/D)の開発を μPD78F0034A(10 ビット A/D)で代行

Q1. 開発を μPD78F0034A で行い、マスク化は μPD780021A で行います。 μPD78F0034A は A/D が 10 ビット、μPD780021A は 8 ビットと違いがありますが、ソフトウエア完全互換で開発できますか。

A1. はい、可能です。

μPD78F0034A で μPD780021A を評価する場合には、μPD780021A としてプログラムを作成してください。これを μPD78F0034A に書き込んで実行した場合は、A/D の変換結果は 8 ビットで読み出せます。
μPD78F0034A の 10 ビット A/D は、上位 8 ビットと下位 2 ビットに分かれており、8 ビットの変換結果が欲しい場合には、上位 8 ビットを読めばいいようになっており、8 ビット A/D と同じ結果を得ることができます。
なお、μPD78F0034A での評価では、必ず IMS レジスタでメモリ構成を μPD780021A の設定にしてください。

μPD78F9116 での 8 ビット A/D 変換 [78K0/S]

Q1. 製品には μPD789102(マスク品)、開発には μPD78F9116(フラッシュ品)を使用します。μPD78F9116 でアセンブルすると、「MOV A, ADCR0」の命令でエラーが発生します。

A1. μPD78F9116 では、ADCR0 は 10 ビット A/D コンバータ用に、16 ビット・メモリ操作命令だけ使用できるように定義されています。μPD789102(8 ビット A/D コンバータを内蔵)の評価で μPD78F9116 を使用する場合には、プログラムの作成段階から μPD789102 用のデバイス・ファイルを使用してください。

参照 「μPD789104A,789114A,789124A,789134A サブシリーズ」ユーザーズ・マニュアル
第 18 章 μPD78F9116A, 78F9136A フラッシュ・メモリ製品とマスク ROM 製品の違い

注意 8 ビット A/D コンバータ(μPD789104A, 789124A サブシリーズ)として使用するときは、8 ビット・メモリ操作命令で、10 ビット A/D コンバータ(μPD789114A, 789134A サブシリーズ)として使用するときは、16 ビット・メモリ操作命令で、それぞれ A/D 変換結果レジスタ 0(ADCR0)を操作します。
ただし、μPD78F9116A や μPD78F9136A を μPD789101A, 789102A, 789104A や μPD789121A, 789122A, 789124A のフラッシュ・メモリ製品として使用するときは、ADCR0 を 8 ビット・メモリ操作命令で操作できます。その場合、μPD789101A, 789102A, 789104A 用、または μPD789121A, 789122A, 789124A 用にアセンブルしたオブジェクト・ファイルで行ってください。

Q2. μPD789111A と μPD789101A に同じプログラムを入れて、入力端子で切り替えたいのですが、「MOV A, ADCR0」と「MOVW AX, ADCR0」を共存させられません。

A2. ADCR0 というレジスタを 8 ビットと 16 ビットの両方の意味で使うことはできません。
そのような使い方が必要な場合には、ADCR0 を明示的に使用するのでなく、0xFF15 番地のバイト・データを指すポインタと、0xff14 番地のワード・データを指すポインタの2つを準備しておき、8 ビットの変換結果の場合に 0xFF15 番地のポインタでアクセスし、10 ビットの変換結果では 0xFF14 番地のポインタでアクセスすることが考えられます。
それよりは、μPD789111A だけでプログラムを作成し、8 ビットの値が欲しい場合には、変換結果の上位 8 ビットを使うことを考える方が容易です。

Q3. 評価は μPD78F9116A、量産はマスク品の μPD78102A を使用する予定ですが、A/D コンバータは 10 ビットと 8 ビットの違いがあります。これを切り替えるような処理を行えばいいですか。

A3. そのような処理は、全く必要ありません。プログラムは、μPD78102A 用に作成してください。
それをそのまま、μPD78F9116A に書き込むだけでけっこうです。

10 ビット A/D の変換結果の格納

Q1. μPD78F9116 の 10 ビット A/D について、変換結果はどのように入りますか。

A1. 変換結果の 10 ビットは、16 ビット読み出しのビット 15 から配置されます。下位の 6 ビットには、0 が入ります。したがって、読み出した 16 ビットの変換結果は、次のようになります。

レジスタ・ビット位置	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
変換結果データ	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	0	0	0	0	0	0

10 ビット A/D 変換値の高速 6 ビット・シフト

Q1. μPD78F9116A で、10 ビット A/D 変換結果は上位 15 ビット目から順に格納されますが、高速にデータを右に 6 ビット・シフトさせる方法はありますか。

A1. 基本的にアセンブラで記述して、それをCから関数コールすることになります。

ノーマル・モデルに対応したアセンブリ言語記述の本体部は、次のようになります。

```
_funcsub:  XCH     A,X     ;GET B1,0 IN A
           ADD     A,A     ;SET CARRY WITH B1
           XCH     A,X     ;GET B9-B2 AND SAVE B0
           ADDC    A,A     ;ADD B1 TO B8-B2 AND SET CARRY WITH B9
           XCH     A,X     ;GET B0 AND SAVE B8-B1
           ADDC    A,A     ;GET B9 AND SET CARRY WITH B0
           XCH     A,X     ;GET B8-B1 AND SAVE B9
           ADDC    A,A     ;ADD B0 TO B7-B1 AND SET CARRY WITH B8
           MOV     C,A     ;SAVE B7-B0 TO C RETURN VALUE
           MOV     A,X     ;GET B9
           ADDC    A,A     ;ADD B8 TO B9
           MOV     B,A     ;SAVE B9,B8 TO B
           RET
```

AXレジスタの入力をシフトして、下から詰め直したものがBCレジスタに入って戻ります。ad_buffに値があり、戻りも ad_buff に格納する場合、C 言語は、この関数を次の形でコールすることになります。

```
ad_buff = funcsub(ad_buff);
```

μPD789445 の変換時間

Q1. μPD789445 のユーザーズ・マニュアルで、電気的特性にある A/D コンバータ特性の変換時間ですが、上段の条件は「AVDD＝2.7～5.5V」と記述されていて、下段は空欄です。下段は「AVDD＝1.8～2.7V」ですか。

A1. はい、そうです。

条件の欄に記述がない場合、一般には表の上に記述している条件を適用します。この考え方をしてもいいのですが、条件欄が空白の特性は緩和されていますので、「上の条件欄以外の条件」と考えても同じです。

A/D 変換時間の違い [78K 共通]

Q1. μPD780058 のユーザーズ・マニュアルで、A/D 機能説明中の ADM で選択する変換時間(p.269)と、電気的特性の変換時間(p.628)の関係を教えてください。

A1. データ・シートの記述を満足する範囲で、システムに適した変換時間を ADM レジスタで選択してください。

[ADM の記述]

どのような設定ができるのか**機能**を示しています。 動作クロック周波数に対応して、最適な値を選択できるようになっていますので、 お客様のシステムの要件にしたがって、最適な値を選択してください。

[電気的特性の記述]

保証値を示しています。 つまり、デバイスとして、どこまで高速に変換できるか**性能**を示しています。 この範囲外では、変換精度が保証できません。

A/D 変換は CH で指定した時間だけ待てば終了するか [78K 共通]

Q1. μPD789445 で、ADMO=88H(fx=4MHz)での変換時間は 30μs なので、CH の指定から 30μs 待てば、A/D 変換は正しく終わっていますか。

A1. CH の指定をしても、すぐ A/D 変換は開始されません。ソフト・タイマで変換終了を待たず、INTAD0 の発生(IF1 レジスタの ADIF0 をセット)を待ってください(当然、ADIF0 がセットされたのを確認後、ADIF0 はクリア)。

Q2. FR02, FR01, FR00 を 48/fcc に設定して、1MHz 動作で A/D 変換を行っています。変換時間をオシロスコープで測定すると、約 80μs かかっています。48/fcc なら、1 回の A/D 変換時間は 48μs 程度ではないですか。

A2. 変換開始後、FRC02-00 で指定した時間で終了します。しかし、CPU のクロックと A/D の動作クロックが異なるので、CPU が A/D 変換を起動しても、すぐには開始しません。また、変換終了割り込みにより、プログラムでポートを操作しますが、変換終了割り込み発生から処理開始までと、プログラム処理の時間が必要です。お問い合わせの内容では、クロック周波数が低いので、1 命令の実行に最低でも 2μs は必要になります。このため、変換に約 80μs かかっているように見えていると思われます。

アナログ入力に指定しなかった端子をポートとして使用 [78K 共通]

Q1. アナログ入力チャネル指定レジスタで指定した端子以外は、入力ポートとして使用できますか。

A1. はい、アナログ入力しない端子を入力ポートに使用できます。ただし変換中はデータを読まないでください。

Q2. μPD789166 の A/D 変換について、「変換中にポートの入力命令を実行すると、変換分解能が低下することがある」と記述があります。どの程度低下するのですか。

A2. デバイスとして保証している精度の、最下位ビットが影響を受ける程度とお考えください。

Q3. μPD789166 の A/D 変換について、「変換中にポートの入力命令を実行すると、変換分解能が低下することがある」と記述があります。これは、入力命令を実行したときの変換、1 回分だけですか。

A3. はい、入力命令を実行したときの変換結果だけが対象です。

Q4. 変換中にポートの入力命令を実行すると、なぜ分解能が低下するのですか。

A4. アナログ信号に比べ、デジタル信号は振幅が大きく、周りの回路に影響を与えます。デジタル信号同士なら、発生ノイズがスレッシュホールド電圧を越えなければ、影響はありません。しかしアナログ信号は、ノイズの分だけ直接影響を受けます。このため、A/D の精度が、デジタル信号からのノイズで低下します。

ADCS0=0 による直列抵抗ストリングのカットオフ

Q1. ADCS0=0 とすれば、A/D コンバータの直列抵抗ストリングは、オフになるのですか。

A1. はい、そうです。A/D 停止時(ADC0=0)、直列抵抗ストリングに流れる電流を切り、消費電力を低減できます。

A/D 端子として使用する場合の PM レジスタの入出力設定

Q1. 兼用端子を A/D 用に使用するとき、モード設定は入力と出力のどちらでもいいですか。

A1. いいえ、PM レジスタで入力に指定し、出力バッファを OFF にしてください。出力では、A/D コンバータは動作しますが、アナログ入力とマイコンの出力が衝突します。なお、PMC レジスタや MM レジスタは設定不要です。

A/D リファレンス電圧 AV_{REF}を 5V から 3.5V に変更したときの問題

Q1. μPD780988 を電源電圧 5V、リファレンス電圧 5V で使用していました。リファレンス電圧を 3.5V に変更したいのですが、問題や注意点はありますか。

A1. AV_{REF}を 3.5V で使用する場合には、5V 時に比べて、A/V 変換の総合誤差が 50%悪くなり(TYP:0.2%FSR が 0.3%FSR)、その他の変換誤差特性も悪くなります。
詳細は、ユーザーズ・マニュアルの「A/D コンバータ特性」をご参照ください。
また、変換時間が、14μs(MIN)から 19μs(MIN)と長くなってしまいます。

A/D 変換専用の電源端子(AV_{DD}とAV_{SS})を接続しない [78K 共通]

Q1. A/D コンバータの電源端子(AV_{DD}とAV_{SS})がありますが、これらを接続しないとどうなりますか。

A1. そのような場合には、正常な A/D 変換結果を得ることが保証できません。
内部の保護ダイオードを経由して、V_{DD}からAV_{DD}に電流が流れます。したがって、あたかも動作しているかのように見えることは考えられますが、つねに同じ結果が得られるかどうかは保証できません。

Q2. μPD789104A の P60/ANI0～P63/ANI3 端子のうち、P60/ANI0～P62/ANI2 を A/D 入力、P63/ANI3 を入力ポートとして使用しています。P63 は、抵抗で V_{DD} にプルアップしていますが、端子の電圧が上がりません。なお AV_{DD} は、ANI0～ANI2 を使い始めるときに供給しています。

A2. ANI 端子には、保護のためのダイオードが入っており、これによって AV_{DD} より高い電圧が入力されないようになっています。ご指摘の現象は、P63/ANI3 に抵抗を介して入力された電流が、このダイオードにより AV_{DD} 側に流れて発生するものです。A/D 変換をしないときでも、AV_{DD} には V_{DD} と同じ電圧を印加してください。

A/D 変換結果は 2 度読みしないと正常な値にならないか

Q1. μPD78F0078 で、A/D 変換結果は 2 度読みで正常値になるということですが、ハード的な対策はありますか。

A1. ハード的な回避策はありません。また、2 度読みが対策ではありません。A/D 変換を開始して、1 回目の変換結果が正しくないことがありますので、これは捨てて、次の変換結果を使ってください。
なお、いったん A/D 変換を開始すると、2 回目以降の変換結果は正常な精度となりますので、A/D 変換動作許可にして、最初の 1 回だけ無視すれば、あとはチャネルを切り替えても、1 回読むだけでけっこうです。

A/D の完了用割り込みフラグ INTAD のセット／リセット条件

Q1. μPD780058 で、A/D の完了用割り込みフラグ INTAD のセット/リセット条件を教えてください。

A1. セットされるのは、A/D 変換が完了したときか、ソフトでセットしたときです。リセットされるのは、CPU がリセットされたときか、INTAD の割り込みが受け付けられたとき、ソフトでクリアをしたときです。

Q2. A/D の動作完了直前に CS=1 を書き込むと、INTAD フラグはどうなりますか。

A2. A/D 変換動作中に CS を再度 1 にすると、最初から変換動作をやり直します。

A/D 入力端子のインピーダンス

Q1. A/D コンバータの入力端子のインピーダンスは、どの程度ですか。

A1. この端子は、時間とともにインピーダンスが変化します。サンプリング・コンデンサの容量は、数 pF 程度です。サンプリング時には、コンデンサを充電するために電流が流れますが、通常はリーク電流程度しか流れません。したがってインピーダンスは、その状態によって大きく変化します。

なお A/D 端子は、数 kΩ 以下の低いインピーダンスでドライブしてください。これは、サンプリング時間中に、サンプリング・コンデンサを A/D の精度内に充電するためで、インピーダンスが高いと精度が得られません。

A/D 端子に付けることができるコンデンサの容量

Q1. A/D 入力ポートに直付けできる、電解コンデンサの容量を教えてください。

A1. アナログ入力に、大容量の電解コンデンサをつけることには問題ありません。しかし、コンデンサを付けると、そこに蓄えられた電荷の影響で、電源オフ時に、端子電圧が電源電圧を越える可能性があります。

これを避けるために、電荷を逃がすためのダイオードを必ず追加してください。通常のポートも同様です。

変換処理時間 [μPD780034A サブシリーズ]

Q1. HALT モードの場合の処理時間はどの位になりますか。

A1. 次のような、処理と時間が考えられます。

処　　理	クロック数	条　　件
割り込み受け付けまでの時間	約 10	
A/D 起動処理	7	
A/D 変換 1 回の時間	144	μPD78F0034B, 12MHz, 4.5V
	120	8.38MHz, 4.0V
	96	5MHz, 2.7V
2 回の変換に必要な時間	288	μPD78F0034B, 12MHz, 4.5V
	240	8.38MHz, 4.0V
	192	5MHz, 2.7V
CPU と A/D の非同期による処理待ち（最悪1周期として）	144	μPD78F0034B, 12MHz, 4.5V
	120	8.38MHz, 4.0V
	96	5MHz, 2.7V
2 回目変換完了割り込み待ち注	10	
変換値の読み出し	8	

注　1回目でも必要な処理ですが、2 回目の変換動作と平行処理なので、2 回目だけに適用します。

これらを合計すると、次のような時間となります。

クロック数	処理時間	条　　件
467	約 40 μs	μPD78F0034B, 12MHz, 4.5V
395	約 50 μs	8.38MHz, 4.0V
323	約 65 μs	5MHz, 2.7V

リファレンス電圧の設定 [μPD780034A サブシリーズ]

Q1. AV_{SS} に 1V、AV_{REF} に 4V を入力して、この間の 3V を 256 分解能で変換できますか。

A1. いいえ、できません。AV_{SS}＝V_{SS} の条件があります。AV_{SS} は、外部アナログ入力の基準電位ですが、内部ではデジタル回路とのインタフェースの基準電位です。このため、AV_{SS} を V_{SS} と異なる電位にはできません。

★ Q2. 78K0/KC2-A などでは、一側アナログ基準電圧入力が AV_{REFM} か AV_{SS} かを選択できますが、AV_{REFM} では 1V などの入力で、AV_{REFP} との差を少なくして変換誤差を小さくできますか。

A2. いいえ、電気的特性で「V_{SS}=AV_{SS}=AV_{REFM}=0V」と条件付けているので、そのような入力はできません。AV_{SS} は A/D コンバータとオペアンプのグランドも兼用しているので、グランド・ノイズの状況によっては、単独の基準電圧入力として AV_{REFM} を選択できるようにしています。

A/D 変換完了の割り込みが発生しない [78K 共通]

Q1. 次のプログラムで、μPD789177Y の A/D 変換をさせようとしても、割り込み要求フラグがセットされません。

```
ADM0 = 0x28;
ADS0 = 0x00;
ADMK0 = 0;
ADCS0 = 1;
EI();
while(ADIF0 == 0) ;
```

A1. A/D 変換完了割り込みのマスクが解除され、CPU が割り込み許可状態になります。これでは、割り込み要求が受け付けられ、割り込み要求フラグ ADIF0 がクリアされます。次のように、割り込みをマスクしてください。

```
ADM0 = 0x28;
ADS0 = 0x00;
ADMK0 = 1;
ADCS0 = 1;
EI();
while(ADIF0 == 0) {
  NOP();
}
ADIF0 = 0;
```

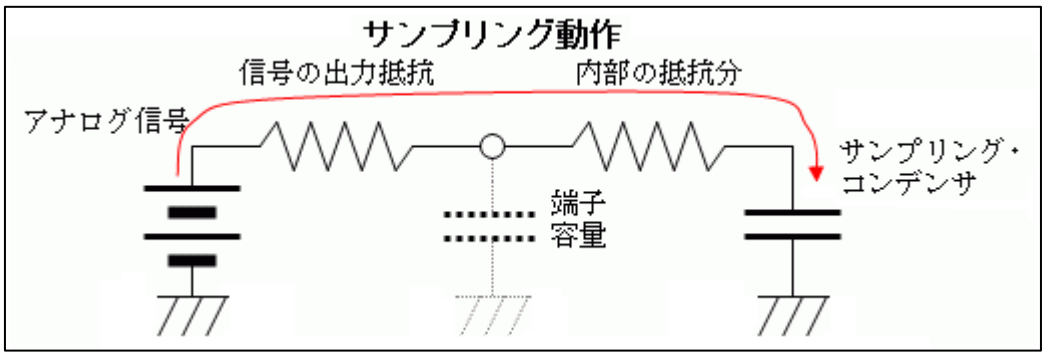
チャネルを切り替えると、変換結果が切り替え前チャネルに影響される [78K0、78K0S]

Q1. 複数のアナログ入力を処理したいので、ADS レジスタで変換するチャネルを切り替えながら、A/D 変換を行っています。ところが、変換結果が、切り替え前の変換結果に影響されているようです。

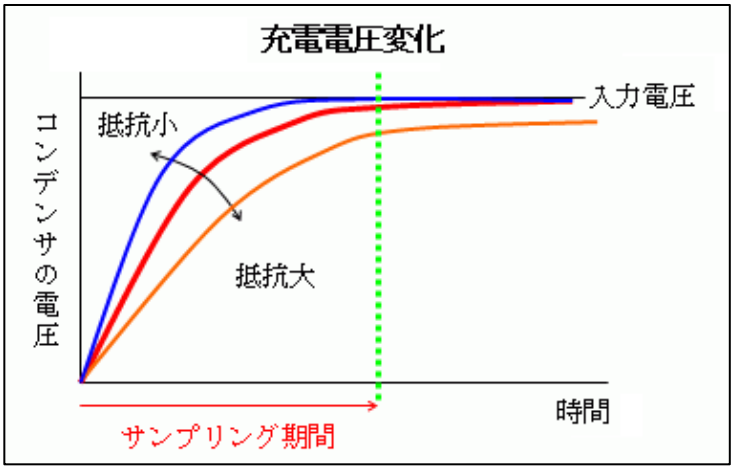
A1. 2 つの要因が考えられます。

(1) ハードウエア要因

78K マイコンの A/D コンバータは、ほとんどが逐次変換型で、アナログ入力を内部コンデンサにサンプリング（チャージ）して、その電圧を変換します。このチャージされた電圧は、次のサンプリングまで保持されます。サンプリングでは、次図のように抵抗を介してコンデンサをチャージします。

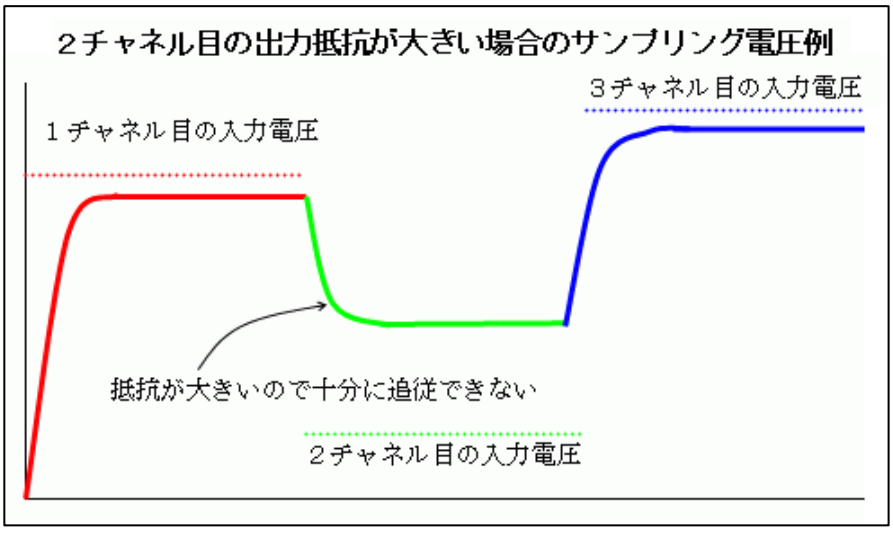


この動作は抵抗を介したコンデンサの充電なので、コンデンサの電圧は次のグラフのような曲線（充電カーブ）を描いて入力電圧に近づいていきます。



直列抵抗の値が大きいと、この曲線はなだらかになります。逆に抵抗値が小さいと、立ち上がりが俊敏になります。この抵抗の大きさによって、サンプリング期間でコンデンサの電圧がどこまで入力電圧に近づくかが変化します。入力電圧に近づいても、等しくなることはありませんが、A/D コンバータの精度以下の誤差なら、使用上は問題ありません。

問題はこの直列抵抗、特に信号の出力抵抗（信号源インピーダンス）が大きい場合です。この抵抗が大きいと十分に充電できず、充電開始前の電圧と入力電圧の間の電圧となってしまいます。これが、チャネルを切り替えたときに、前のチャネルの影響を受ける原因となります。この様子を模式的に示した例を次に示します。この例では、2 チャネル目の出力抵抗が大きいために、1 チャネル目の値から 2 チャネル目の値に変化できない状態を示しています。



この対策として、次の 3 つがあげられます。

①信号源のインピーダンスを下げる

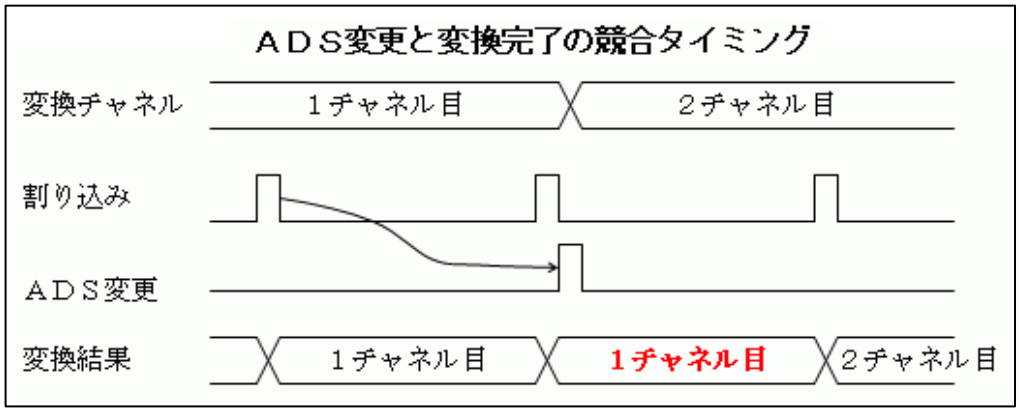
②端子にコンデンサを付加して、信号源のインピーダンスの影響を少なくする

③サンプリング時間を長くする

基本は①ですが、かなり困難です。そこで、②か③の対策となります。②はハードウエアでの対策で、③はソフトウエアでの対策です。③では、変換時間を長くすればサンプリング時間も長くできますが、他のチャネルの変換時間も遅くなるので、2 チャネル目の変換だけ複数回行うことが有効です。変換回数を増やすことで、サンプリングの回数が増加して、トータルのサンプリング時間が長くなります。

(2) ソフトウエア要因

A/D 変換処理は、プログラムの実行と独立です。問題となるのは、ADS レジスタでチャネルを変更した直前のタイミングで、変更前のチャネルの A/D 変換が完了した場合です。このときの割り込みを新しいチャネルでの変換完了の割り込みと解釈して処理しても、切り替え前のチャネルの値を得てしまいます。この場合は、ハードウエアの問題と異なり、切り替え前のチャネルの変換結果となります（下記の **1 チャネル目**）。



この場合の対応方法は、**不要な割り込み要求フラグ**（ADIF ビット）**をクリア**することです。

もし、ベクタ割り込みを使用していたなら、チャネルを切り替える前に割り込みをマスク（ADMK ビットをセット）して、新たな割り込みでベクタに分岐しないようにします。そのあとで ADS を変更し、割り込み要求フラグ（ADIF ビット）をクリアしてから、割り込みマスクを解除（ADMK ビットをクリア）します。

なお、A/D 変換完了のベクタ割り込み処理の中で ADS レジスタを書き換えている場合、多重割り込みを許可しないかぎり、新たな A/D 変換完了割り込みを受け付けることはないので、ADS レジスタを書き換えたあとで割り込み要求フラグ（ADIF ビット）をクリアするだけで済みます。

この場合の派生問題として、ADS レジスタを書き換える前に ADCS ビットをクリアして、**A/D 変換動作を停止する場合**があげられます。A/D 変換を停止すると、変換結果が不定になります。その状態で ADS レジスタを変更し、A/D 変換を再開して割り込み要求を待つと、すでに要求フラグはセットされているので、すぐに割り込み要求を受け付け、**不定となった変換結果を読み出し**てしまいます。この場合も、ADS レジスタを変更したら、割り込み要求フラグ（ADIF ビット）をクリアしてください。

いずれも現象は似ており、対策として、A/D 変換時間を長くしたり、A/D 変換を複数回実行したりすることで、正しい結果が得られます。しかし、本来の原因は全く別物です。原因を良く見極めた対策が必要です。

1.2.2 クロック

最小命令実行時間の切り替え

Q1. μPD780031A で、最小命令実行時間が高速(0.4μs)と低速(1.6μs)に変更可能ですが、どのように最小命令実行時間を変更させるのですか。

A1. CPU の動作クロックを PCC レジスタで切り替えれば、命令実行時間が変化します。

発振安定時間

Q1. μPD780024 のメイン・クロック(セラミック発振子)の発振安定時間についてですが、発振周波数が 8.38MHz、リセット入力の場合、どのくらいになりますか。

A1. OSTS レジスタにより 15.6ms となります。

Q2. 78K 全般の発振安定時間について、電気的特性ではセラミック発振子使用時に 4ms、リセット機能の説明では発振安定時間経過後 (2^{17}/fx)などと記載されていますが、なぜ異なっているのですか。

A2. 電気的特性に記載された値は、発振回路の特性としての規格です。つまり、セラミック発振子を用いて、マッチングが取れた状態であれば、内蔵の発振回路では 4ms 以内に発振が安定します。
2^{17}/fx の方は、マイコン全体として、発振が安定するまで CPU 動作を開始させない時間を意味しています。
こちらの値の方が、実際の発振回路の安定時間より長ければ、CPU が動き出したときには、すでにクロックの発振が安定していますので、発振安定時間に起因する誤動作の可能性がなくなります。

サブシステム・クロックの発振安定時間と確認

Q1. μPD789488 のパワーオン・リセットで、メインとサブのシステム・クロックが起動します。サブシステム・クロックの安定時間も含めて、発振安定時間(2^{15}/fx)経過後にプログラムを実行開始するのですか。

A1. いいえ、この発振安定時間には、サブシステム・クロックの発振安定時間を考慮していません。
サブシステム・クロックの発振安定時間が必要な場合は、このあとにソフトウエアで確保してください

Q2. サブシステム・クロックの発振安定は、ソフトウエアでどのように確認できますか。

A2. 時計用タイマのインターバル・タイマ機能を使う方法が考えられます。
WTM レジスタで、クロック・ソースにサブシステム・クロックを選択し、時計用タイマを動作させて、9 ビット・カウンタのオーバフローを INTWT1 の割り込み要求フラグ(WTIIF)がセットされるのを待つ方法が考えられます。

サブシステム・クロックからメイン・システム・クロックへの切り替え

Q1. μPD78011 で、サブシステム・クロックからメイン・システム・クロックにすぐ切り替え可能ですか。

A1. すぐに切り替えてはいけません。通常、メイン・システム・クロックを安定発振させるには、4ms 必要です。
プログラムですぐに切り替えると、せいぜい 0.6ms 程度となり、発振が安定する前にメイン・システム・クロックに切り替えてしまいますので、異常なクロックで動作して暴走する可能性があります。
OSTO レジスタで指定した時間だけ、遅らせてから切り替えるようにしてください

サブシステム・クロック動作中の HALT をタイマ割り込みで解除できない

Q1. μPD789046 で、メイン・システム・クロック(4MHz)からサブシステム・クロック(32.768kHz)へ切り替えて HALT モードに入り、16 ビット・タイマ(サブシステム・クロック動作)の割り込要求で HALT モードを解除させたいのですが、HALT が解除されません。

A1. これは、デバイスの制限事項に該当する内容と思われます。ブザー出力許可に設定してみてください。
なお、この制限事項については、販売店から文書をお取り寄せください。

STOP 解除後のクロック [78K0/Kx1]

Q1. メイン・クロック・レジスタ MCM のビット 0(MCM0)が 1(X1 入力クロック)の状態で STOP モードに入り、割り込みで STOP モードが解除すると、次の命令実行は内部発振器での動作ですか。

A1. いいえ、元の状態(X1 入力クロック)での動作となります。
STOP モードをリセットで解除した場合だけ、内部発振器での動作となります。

Q2. STOP モード解除後、X1 入力クロックの発振安定時間経過まで、内部発振器で動作させたい場合は、STOP モードに入る前に、メイン・クロック・レジスタ MCM のビット 0(MCM0)を 0(内部発振クロック)にしなければいけないのですか。

A2. はい、そうです。この場合、メイン・クロックの停止は、MOC レジスタの MSTOP ビットで制御してください。

クロック・モニタの動作 [78K0/Kx1]

Q1. X1 を接続しない状態で立ち上げた場合、クロック・モニタの動作を許可してもリセットがかかりません。

A1. クロック・モニタは、X1 に 1 クロックも入力しない状態では動作しません。つまり、X1 を接続しない状態では、クロック・モニタ機能は使えません。
また、X1 に入力していない状態では、X1 での動作を指定しても X1 に切り替わらず、内部発振器での動作を継続します。

Q2. X1 入力が全くない場合、クロック(X1)異常はどのようにして検出すればいいのですか。

A2. 最初から X1 が入力されないことを検出するには、タイマ H を内部発振器で動作させ、ある時間までに OSTC が一度もセットされなかったら、X1 の発振異常とご判断ください。

メイン・クロックが停止する

Q1. STOP 命令を実行しないのに、メイン・クロック発振が停止します(プログラムはエミュレータで確認済み)。

A1. 発振回路のマッチングはとれていますか。マッチングがとれていない場合、正常な発振が得られません。

Q2. μPD78F0034A の X1, X2 に指定定数での発振回路を接続していますが、数秒後に発振が停止します。

A2. 発振回路のマッチングに問題がなく、プログラムで止めていないのにクロック発振が停止するのなら、リセット入力にノイズがのっている可能性があります。リセット中はメイン・クロックの発振が停止しますので、リセット・ラインをご確認ください。

サブシステム・クロックの停止 [78K0 全般]

Q1. μPD78P058 で、サブシステム・クロックからメイン・システム・クロックに切り替えても、サブシステム・クロックの発振が止まりません。

A1. 78K0 では、サブシステム・クロックの発振を制御できません。サブシステム・クロックを使用しない場合、消費電力を低減するには、フィードバック抵抗を切り離すことができるだけです。
なお 78K0S では、サブシステム・クロックの発振を制御できる製品があります。

クロック切り替え [μPD780958]

Q1. サブシステム・クロックからメイン・システム・クロックへ変更時、MCC 設定後、何クロック待てばいいですか。

A1. 待つ必要はありません。CR 発振の場合は、サブシステム・クロック動作で、MCC をクリアして発振可能にしたあと、クロックの切り替えまでには発振が安定しています。つまり、発振安定時間をプログラムで待たず、発振可能にした直後に、クロックを切り替えられます。

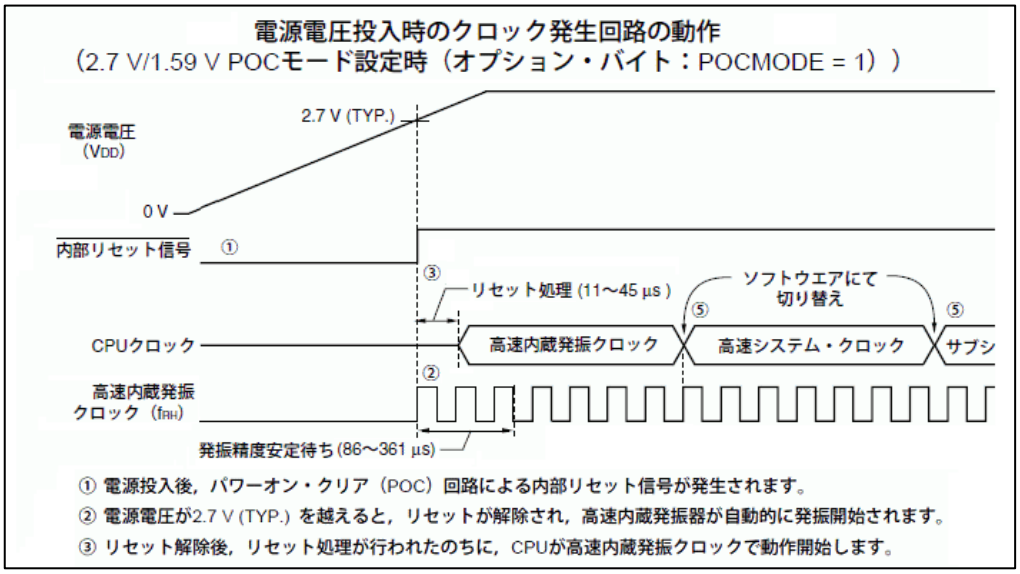
X1 発振回路で X1 端子電圧が 2.5V でいいか [78K0R/Kx3]

Q1. 78K0R/Kx3 の 3.3V 動作で、セラミック発振子を接続すると、X1 端子が 2.5V しかありません。

A1. 78K0R/Kx3 では、発振回路は内蔵レギュレータの電圧での振幅となりますので、だいじょうぶです。

★ 高速発振内蔵クロックでの動作開始 [78K0/Kx2]

Q1. ユーザーズ・マニュアルの「電源投入時のクロック発生回路の動作（2.7V/1.59V POC モード設定時）」の図で、電源電圧が 2.7V に達したあと、高速内蔵発振が安定（②：86～361 μs）する前に、CPU のリセット（③：11～45 μs）が終了していますが、いいのですか。



A1. 発振精度（±5%）以内に安定する前の高速内蔵発振クロックが CPU に供給されて、CPU は動作を開始し、プログラム実行を開始します。このとき、タイマなど時間のずれが容認できない周辺ハードウエアを動作させなければ、特に問題ありません。

1.2.3 シリアル通信

設定に起因したシリアル通信ポートの動作異常

この内容は 78K0 シリーズ、78K0S シリーズ、μPD78421X サブシリーズに共通の事項です。これらのマイコンでは、シリアル通信ポートを使用する場合に、ポートの設定が必要です。
シリアル・ポートと I/O ポートが、兼用となっています。シリアル通信を実行するには、ポートのモード、出力ラッチのデータを設定しておく必要があります。特に出力となっている信号については、ポートを出力モードに設定して、ポートの出力ラッチに決められたデータ（デバイスや信号によって異なる)をセットしておく必要があります。これをしないと、送信完了の割り込みは発生しても、データが送信されていないといった現象が発生します。

Q1. μPD780034A の UART で受信はできますが、送信ができません。

A1. TxD 端子と兼用の端子の設定が原因と思われます。P23 と P24 について、次のように設定してください。
P23：入力モード（PM23=1、P23=0）
P24：出力モードで出力ラッチに 0 を設定（PM24=0, P24=0）

Q2. μPD789022 の UART で受信を行っていますが、受信バッファ・レジスタ(RXB00)にデータが取得できません。

A2. PM20 に 1、PM22(RxD 端子)に 1(入力)を設定されていますか。PM22 が 0 になっていると、端子が出力となり、正常なデータが受信できません。

Q3. μPD78F0034A で、3 線式シリアル通信を使用しますが、シリアルのクロックやデータが動作しません。

A3. SCK30, SO30 と兼用になっている端子の出力ラッチを 0 にセットしていませんか。
この場合、外部に信号は出力されませんが、内部動作は終了して、割り込み要求フラグがセットされます。

Q4. μPD784216A で、シリアル・チャネル 0, 1 の 3 線式シリアル通信で、シリアル・クロックが出力されません。

A4. SCK 出力と兼用の端子(P22 か P27)の出力ラッチのデータが、1 になっていませんか。
ポートの出力ラッチのデータと兼用の信号(ここではクロック)は、OR 論理で出力されます。このため、出力ラッチが 1 になっていると、端子は H レベルに固定されます。P2 出力を 0 に設定してください。

Q5. 78K4 の 3 線式シリアル通信で、データが出力されません。

A5. シリアル出力と兼用の端子(P21)の出力ラッチのデータが、0 になっていませんか。
シリアル出力と P21 の出力ラッチの出力は、AND 論理で出力されます。このため、P21 は 1 でなければなりません。P21 の初期状態は 0 なので、端子の状態はロウ・レベルになります。P21 を 1 にしてください。

スレーブ・モード／マスタ・モード

Q1. μPD780078 で、シリアル動作モード・レジスタ 1(CSIM1)のビット 5(SSE1)を次のように設定すれば、スレーブ・モード／マスタ・モードを切り替えられますか。
SSE1＝1：スレーブ・モード
SSE1＝0：マスタ・モード

A1. いいえ違います。CSIC1 レジスタの CKS12-0 ビットで外部クロックを選択するとスレーブで、それ以外ではマスタになります。ユーザーズ・マニュアルの図 17-3 に、このレジスタについての記述があります。
なお、転送クロックを出力する方(クロック・ソースが内部の方)、つまり通信を制御する方がマスタであり、クロックを外部から入力する方がスレーブです。

Q2. スレーブ・モードで SSE1＝0(使用しない)にしたときの受信動作は、通常の動作ですか。
STOB1 への書き込みで、送信動作をスタートできますか。

A2. SSE1＝0 にしたときは、通常の 3 線式の動作になります。
なお、通常動作でも、スレーブが単にデータを STOB1 に書いただけでは、送信はスタートしません。
その状態で、マスタからクロックが入力されて、初めて送信動作を開始します。

初期設定時に不要データが送信されてしまう

Q1. シリアル・インタフェース・チャネル 2 を UART で使用しますが、リセット後、最初に 0xFF が出力されます。

A1. 初期化の手順に問題があります。
TxD 信号はポート 71 と兼用端子になっており、リセット後は入力ポートとして機能します。このとき、PM71 を出力に設定すると、その時の出力ラッチの内容が出力されます。
リセット後は出力ラッチが 0 になっているので、そのまま出力ポートに設定すると、L レベル(スタート・ビットと同じ)が出力されます。この状態は、出力ラッチに 1 をセットするまで続きます。この期間が、ボー・レートで決まる 1/2 ビットの時間より長い場合には、この状態がスタート・ビットと見なされ、このような現象になります。
初期化の手順を「P71 の出力ラッチへの 1 のセット、PM71 を出力ポートにセット」の順にしてください。

不要データを送信しないための初期化

Q1. μPD784218 のアシンクロナス・シリアル・インタフェースの送信説明に、次の注意書きがありますが、最後の「先にポートを出力にすると、誤動作する可能性があります」と、3 と 4 の手順が矛盾していると思えます。

> **注意** UART で送信する場合、1 バイト目は次の手順で行ってください。
> (1) ポートを入力モード(PM21=1 または PM71=1)にし、ポート・ラッチに 0 を書き込みます。
> (2) アシンクロナス・シリアル・インタフェース・モード・レジスタ n(ASIMn)のビット 7(TXEn)をセット(1)して送信許可状態にします。
> (3) ポートを出力モード(PM21=0 または PM71=0)にします。
> (4) TXSn に送信データを書き込み、送信動作を起動します。
> 先にポートを出力モードにすると、端子から 0 が出力されてしまうために、誤動作する可能性があります。

A1. この手順に誤りはありません。(2).の段階での、状態の変化を誤解されています。
(1)では、UART は動作していないため、その出力は 0 であり、端子はポートとして機能します。このため、ここでは入力ポートにしたままの状態で、出力ラッチに 0 をセットします。
(2)で UART の出力が許可されるので、**「出力が 1」**となります。(1)で出力ラッチが 0 なので、この段階でポートの出力ラッチと UART の出力を OR 演算した値(この値をバッファを介して端子に出力)は 1 となります。
つまり、UART を送信許可にすれば、UART の出力は STOP ビット極性(マーク状態 : 1)になります。
(3)でポートを出力モードに設定すれば、(2)の OR 演算値(＝1)が端子に出力されます。つまり、端子はハイ・インピーダンスから 1 出力となります。このあと、TXSn にデータを書き込めば、送信データが出力されます。

シリアルのクロックにタイマ・レジスタを使用

Q1. μPD78054 の 8 ビット・タイマ・レジスタ 2 を使用するには、シリアル・インタフェース・コントロール・レジスタ 0 でクロックをどのように設定するのですか。CSIM0 を 9AH に設定しても、SCK0 からクロックが発生しません。

A1. CSIM0 の設定だけでは、SCK0 からクロックは出力されません。次の初期設定をしておく必要があります。

・SCK0 からクロックが出るには、CSIM0 の設定以外に、PM27 に 0(出力)、P27 に 1 をセット

・この CSIM0 の設定では、SB1, SCK0 端子を抵抗でプルアップ

この状態で、SIO0 に送信ならデータを書き込むか、受信なら FFH を書き込むと、通信が開始され、SCK0 が出力されるはずです。

エラー発生時の設定による INTSER0 への影響

Q1. μPD780022 で、UART0 の ISRM0 を 1 に設定して、エラー発生時に受信割り込み要求を発生させないようにすると、INTSER0 の割り込みは可能ですか。

A1. はい、可能です。ISRM0 は受信完了割り込みの発生を制御するだけで、INTSER0 には影響しません。

送信データ書き込みタイミングはボー・レートから換算した時間後でいいか

Q1. 「1 ビット相当のクロック・パルス幅×8 ビット」の時間後に、転送終了として、次のデータを書き込めますか。

A1. いいえ、転送完了フラグの変化を待ってから、次のデータを書き込んでください。

CPU の動作と SIO の動作は非同期なので、SIO にデータを書き込んでも、すぐ送信を開始するわけではありません。同期をとるために、最悪で約 1.5 シリアル・クロック分の時間が経過後に、送信が開始することがあります(内部クロックで動作する場合)。なお、実際に転送が行われている時間は、ご考察のとおりです。

SIO31 を 2 線式で使用する場合の設定

Q1. μPD780024 で、SIO31 を 2 線式で送信専用、SI31 端子を出力ポートとする場合、SIO31 送信モードで、PM34=0 とするだけでいいですか。

A1. はい、SIO31 は送信専用モードがないので、送受信モードに設定し、PM34=0(ポート 34 出力)に設定すれば、SI31 端子を出力ポートで使用できます。なお、送信時の無意味な受信データは無視してください。

シリアル内部クロック使用時に ASCK2 は入出力ポートとして使用可能か

Q1. μPD784218 で、SIO2 を内部クロック動作の UART として使用する場合、ASCK2 端子は通常の入出力ポートとして使用できますか。

A1. はい、できます。

PM1 モード設定

Q1. μPD784935 で、PMC1 に 00011000b を設定して、RXD2 を入力モード、TXD2 を出力モードにする場合、PM1 で P13, P14 を入力に設定する必要はありますか。

A1. いいえ、必要ありません。ユーザーズ・マニュアルの図 6-10、図 6-11 に記述されているように、PMC の PMC13 や PMC14 を1に設定すると、それだけで P13 は RXD2 入力に、P14 は TxD2 出力になります。

送信データ書き込みから送信開始までの時間のばらつきをなくす

Q1. μPD78P083 の UART で、送信シフト・レジスタ TXS にデータを書き込んでから、データ送信が始まるまでは、0～数百μs のばらつきが出ます。これをなくすには、どうすればいいですか。

A1. これは、UART のボー・レート・クロックと CPU のクロックが、非同期のために発生する問題です。
これを避けるには、ボー・レート・クロックを CPU の動作に同期させる必要があり、このためには、送信しないときには ASIM の TXE を 0 にしておき、送信開始する前に TXE をセットすることが考えられます。
これにより、ばらつきが抑えられる可能性はありますが、この動作は保証されたものではありません。

公称ボー・レートと実ボー・レートの差

Q1. 内部クロックを分周してボー・レートが決定されるため、ボー・レートの公称値とある程度ずれてしまいます。たとえば、16MHz、32MHz のクロックで 0.16%ずれます(公称 9600bps、実際 9615bps)が、問題ありませんか。

A1. 通常は、ほとんど問題がない範囲の誤差です。
通常の UART は、マージンがビット長の 45%程度あります。これを全体の長さで割ると 4%弱になり、それを送信側と受信側で半分ずつ消費したとしても 2%弱になります。0.16%は全く問題とならない値です。

ボー・レートの許容誤差 [78K0]

Q1. 非同期シリアル通信で、相手から送られてくる信号のボー・レートのずれは、何%まで許容できますか。

A1. 78K0, 78K4 の UART0 の例を説明します。
データの構成(データ長、パリティの有無)、ボー・レート・ジェネレータの設定で異なりますが、ほぼ **4.5%**(8 ビット、パリティあり、k=0)～**5.9%**(7 ビット、パリティなし、k=14)程度の相対的な誤差とお考えください。
具体的な許容誤差は、次のように表されます。この値は、送信側と受信側の相対差です。

許容誤差＝±((16＋k)－1.5)/(2(16＋k)×n)　k：BRGC の設定(0～14)、n：データ長＋パリティ＋1

マイコンの動作クロックにより、ボー・レート誤差が存在した場合、それに対する値と考えてください。

3 線式シリアルでの通信の手順 [共通]

Q1. μPD784031 で、SO0 と SCK、SI0(3 線式)を使用して、マスタ-スレーブ CPU 間の通信を行います。
受信側は、マスタから送信されてくることがわかっている必要がありますか。

A1. はい、その必要があります。マスタからクロックが来ても、スレーブが転送可能状態でないと動作しません。
正常に転送するには、スレーブの転送可能状態を確認してから、マスタが転送を開始する必要があります。

Q2. マスタ側(μPD78F9488)からデータを送信すると、マスタ側では送信完了割り込みが発生しますが、スレーブ側(μPD78F9306)では受信完了割り込みが発生しません。ただし、スレーブ側で SIO10 に何かデータを書き込んでから、マスタ側よりデータを送信すると、スレーブ側で受信完了割り込みが起こります。

A2. スレーブの通信起動(SIO10 へのダミー・データ書き込み)が、マスタの送信前に行われていますか。通信の手順としては、スレーブが通信の準備をしてから、マスタが通信を起動します。これにより、マスタからシリアル・クロックが出力され、そのクロックでスレーブのデータがマスタに送られると同時に、マスタのデータがスレーブに送られます。この順序が逆の場合、マスタからクロックが入力されても、スレーブは動作しません。

シリアル受信動作を停止させないで動作モードの切り替えができるか

Q1. UART のモード・レジスタ説明で、「動作モードの切り替えは、シリアル送受信動作を停止させたあとに行ってください」と記載があります。最初に受信(RXEn＝1)して、そのまま TXEn＝1 として送受信できますか。

A1. いいえ、一度停止しないと、動作は保証されません。

SIO3 からデータを読み出さないと通信は起動しないか

Q1. μPD784935 の SIO3 で受信する場合、SIO3 からデータを読み出さないと、通信は起動しないのですか。

A1. はい、そうです。マニュアルに記載された条件(この場合、SIO からの読み出し)で受信動作がスタートします。この前に SCK が入力されても、無視されます。したがって、マスタから SCK が来る前に、SIO から読み出し(ダミーのデータ・リード)をして、受信状態にしておく必要があります(受信状態にしておいても、マスタから SCK が来なければ、受信は行われないので問題ありません)。データを受信すると、受信データの読み出しが、次の受信の起動となります。

UART 入力端子がロウ・レベルのままだと受信完了割り込みが発生し続ける

Q1. UART 受信中に、データ入力が ロウ・レベルのままだと、8 ビット受信完了とみなされ、受信完了割り込みが発生し続けます。

A1. UART ではストップ・ビット(ハイ・レベル) が必要なので、ロウ・レベルのままにするのは**異常な状態**です。
一般に UART は、スタート・ビット(ロウ・レベル)の検出で、データが送られて来たと解釈し、受信動作を開始します。ロウ・レベルが続く状態では、連続してデータを受信することになるので、**異常な状態**となります。このよな状態では、ストップ・ビットが検出できないので、フレーミング・エラーが発生します。ステータスを調べることで、通信異常を認識することができます。
なお、78K0/Kx1 の UART は、受信完了後に、必ず信号がハイ・レベルになったことを確認し、次のスタート・ビットを検出します。このため、上記のような状態でも、受信完了の割り込みは連続発生しません。

同期式通信でクロック異常によりビットがずれた場合の処置

Q1. μPD789101A が、クロック同期式 CSI のスレーブとして動作中、クロック異常で欠けや増加があると、データと同期しなくなります。シリアル・アイドル時に、シリアル動作モード・レジスタ(CSIM20)の CSIE20 を 0 にして CSI を停止させ、再び 1 にして動作再開すると、最初のクロックで復帰（先頭ビットから送受信開始）しますか。

A1. はい、復帰します。ビットずれが発生したとき、いったんシリアル動作を停止させ、その後再開させてください。これで、ビット・カウンタは初期化されて、先頭ビットからの送受信となります。なお、転送バッファの内容はクリアされませんが、新たなデータに置き換わるので問題はありません。

送信中に送信許可停止した場合の復帰処理

Q1. μPD78P9014 のユーザーズ・マニュアルに、次のような UART の送信中の停止、復帰の手順があります。

(A) 次の送信を行う前に、送信シフト・レジスタに FFH を設定し、TXE00 に 1 を設定してください。

(B) 送信動作中は、ASIM00 の書き換えは行わないでください。それ以降の送信動作ができなくなる場合があります。

(B)の書き換えをしても、(A)の設定をすれば、送信ができなくなる状況にはならないのですか。

A1. いいえ、ちがいます。(A)は、あくまでも TXE00 をクリアして、送信を停止した場合の再開手順です。

(B)は、他の書き換えも含めて、送信できなくなることがあるということです。(B)の状態が、すべて(A)の手順で復帰できるわけではありません。送信中に、送信条件を変える操作はしないでください。

シリアル通信設定の注意事項

Q1. マイコンで、シリアル通信速度を変更するときの注意事項はありますか。

A1. 通信条件を変更するには、必ずデータ通信が終わっている状態で切り替えてください。送受信の途中で通信条件(ボー・レートやモード・レジスタなど)を操作すると、それ以降の送信ができなくなる可能性があります。

SPI への対応 [78K0]

Q1. μPD78F0034A の SIO3 で、3 線式の通信でマイクロ・ワイヤか SPI を使用して、シリアル EEPROM と通信したいのですが、どちらが使えますか。

A1. μPD78F0034A の SIO3 は、どちらも対応しません。μPD78F0078 や 78K0/Kx1 の CSI は、SPI に対応します。

3 線式シリアルの動作 [共通]

Q1. μPD78F0034A の 3 線式シリアル I/O(MSB 先頭固定)で、「8 ビット転送終了により、SIO3n の動作は自動的に停止」とあります。これは、送信時は送受信できるが、8 ビット・データの送信に対する受信データが数バイトある場合や、最初に受信をする場合は、クロックを出力しないので受信できないということですか。

A1. いいえ、3 線式シリアル通信では、送信でも受信でも転送時に起動操作が必要です。UART のように、設定だけしても自動的に受信はしません。たとえば、最初から受信する場合や、引き続きデータを受信する場合は、送受信モードであれば、ダミー・データを書き込んで、シリアルの転送をスタートさせる必要があります。

自動送受信機能付きシリアル(CSIA0)の使い方 [78K0/Kx]

Q1. 自動送受信機能付きシリアル(CSIA0)で、データを送信する方法がよく分かりません。

A1. 自動送受信機能付きシリアル(CSIA0)で、マスタとしてデータを連続送信する手順は、次のとおりです。

(1) CSIMA0 レジスタの CSIAE0、ATE0、MASTER0 ビットをセットし、CSIA0 を動作許可状態(バッファ RAM にアクセス可能)として、自動転送モードでのマスタ動作を設定します。

(2) TXEA0 ビットをセットして、送信可能状態にします。

(3) CSIS0 レジスタでシリアル・クロックを選択します。

(4) BRGCA0 レジスタでシリアル・クロックの分周比を設定し、通信速度を指定します。

(5) ADTI0 レジスタに送信インターバル時間を設定します。

(6) P142 に 1、P144 に 0 をセットして、PM142,PM144 に 0 をセットします。

(7) バッファRAMに、FA00H から順に、送信したいデータを書き込みます。

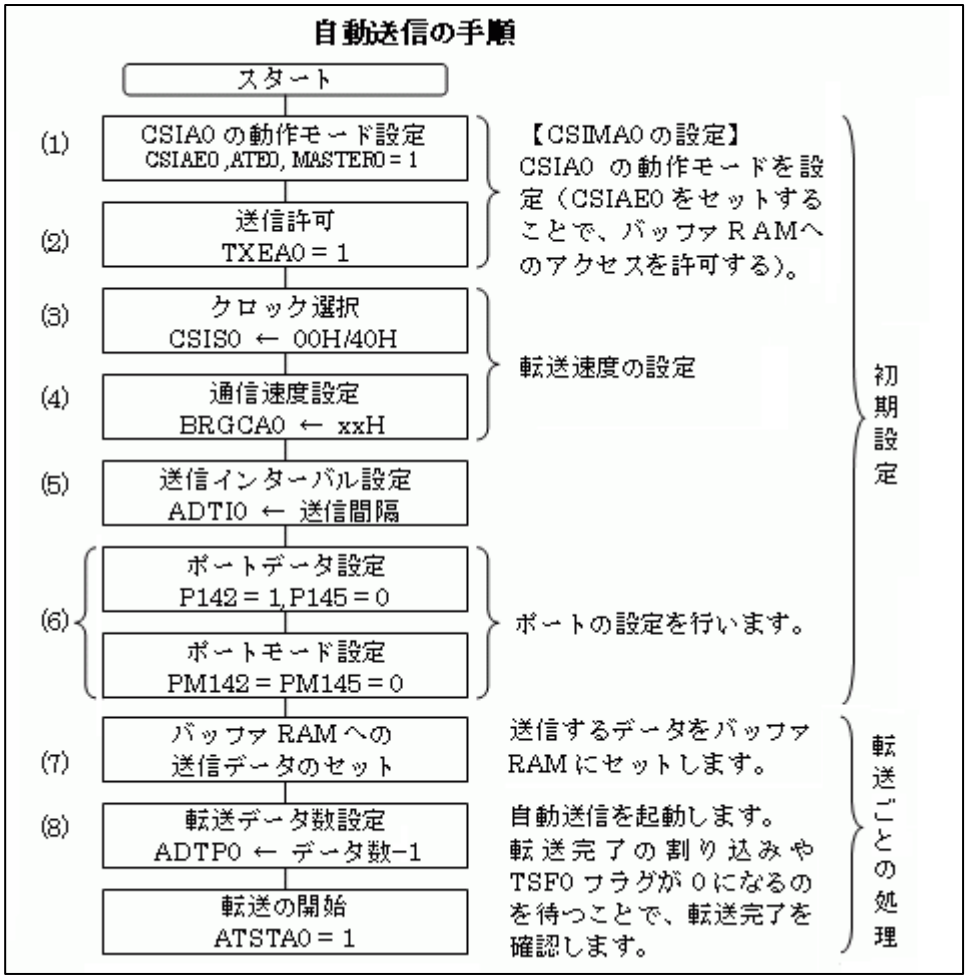
(8) ADTP0 レジスタに「送信したいデータ数－1」を設定します。

これで、必要な設定は完了です。CSIT0 レジスタの ATSTA0 ビットをセットすれば起動します。すると、CSIS0 レジスタの TSF0 ビットがセットされ、バッファ RAM から順に読み出したデータを送信します。ADTP0 で指定されたデータを送信完了すると、割り込み要求(INTACSI)を発生して、CSIS0 レジスタの TSF0 ビットがクリアされます。次のデータを送信したい場合は、新しいデータをバッファ RAM にセットし、ADTP0 レジスタへデータ数を設定してから、CSIT0 レジスタの ATSTA0 ビットをセットしてください((7)からを繰り返す)。

送受信の場合は、(2)で RXEA0 ビットもセットし、(6)で PM143 を 1 にセットします。

転送が完了すると、バッファRAMに格納されている受信データを読み出します。

送信の場合の、フローチャートとプログラム例を次に示します。



次にプログラム例を示します。初期設定をサブルーチン CSIAINIT で、また転送ごとの処理をサブルーチン TXBLK で実現します。TXDATA に準備した 8 バイトのデータを 4 バイトずつ 2 回に分けて送信する例です。

```
        CSEG
START:  MOVW   SP,#STACK
        CALL   !CSIAINIT                ; CSIA0 の初期設定
        MOVW   HL,#TXDATA               ; 送信データの先頭を指す
        MOV    B,#4                     ; 転送データ数を設定
        CALL   !TXBLK                   ; 送信データの設定と転送起動
LOOP1:  BF     CSIS0.0,$LOOP1
LOOP2:  BT     CSIS0.0,$LOOP2           ; 転送完了(TSF0=0)を待つ
        CALL   !TXBLK                   ; 次の転送を起動
LOOP3:  BF     CSIS0.0,$LOOP3
LOOP4:  BT     CSIS0.0,$LOOP4
;
LOOP:   NOP                             ; 転送終了でのループ
        BR     $LOOP
;
;       CSIA0 を使用するための初期設定
;
CSIAINIT:
        MOV    CSIMA0,#11011000B        ; 動作モードの設定
        MOV    CSIS0,#01000000B         ; クロック・ソースを選択
        MOV    BRGCA0,#00000010B        ; ボー・レートの設定
        MOV    ADTI0,#00000001B         ; インターバルの設定
        MOV    P14,#00000100B           ; ポートの設定
        MOV    PM14,#11101011B
        RET
TXBLK:
MOVW    DE,#0FA00H                      ; バッファ RAM のアドレス設定
        PUSH   BC                       ; データ数のセーブ
TXLOOP:
        MOV    A,[HL]                   ; バッファ RAM にデータをセット
        MOV    [DE],A
        INCW   HL
        INCW   DE
        DBNZ   B,$TXLOOP
        POP    BC
        MOV    A,B
        DEC    A
        MOV    ADTP0,A                  ; 転送数の設定
        SET1   ATSTA0                   ; 転送スタート
        RET
TXDATA:
        DB     33H,55H,0AAH,0CCH
        DB     0FH,0F0H,66H,99H
```

UART での 300bps 対応 [78K0]

Q1. μPD789176 の UART で、300bps の通信をしたいのですが、どうすればいいですか。

A1. タイマ 80 で通信用のクロックを発生させ、それを UART の ASCK20 入力に接続して、内部のクロックではなく、BRGC20 で ASCK20 を選択してください。メイン・クロックを 1.2288MHz にすれば可能です。

Q2. μPD78941x を 4.91MHz で使用していますが、UART を 300bps にしたいのに、1200bps 以下になりません。16 ビット・タイマ出力(T05)から 4.8KHz を生成して、ASCK 端子に入力することはできますか。

A2. 16 ビット・タイマでは困難なので、8 ビットの TM02 を使ってください。

マイクロワイヤへの対応 [ほとんどの CSI をもつデバイス共通]

Q1. 78K0/Kx2 に、マイクロワイヤ方式の EEPROM を接続できますか。

A1. はい、可能です。マイクロワイヤでの基本的な信号のやり取りは、シリアル・クロックに同期して行われます。書き込んだデータは、シリアル・クロックの立ち上がりで取り込まれ、読み出すデータも、シリアル・クロックの立ち上がりに同期して出力されてきます。この位相関係を合わせることで、転送は可能です。

また、アドレス長がデバイスによって異なりますが、マイクロワイヤではスタート・ビットが規定されており、その前に任意ビット数の 0 のデータを付加すれば、固定長のシリアル・インタフェースでも対応できます。

たとえば、アドレスが 8 ビットなら、スタート・ビット（1 ビット）とコマンド（2 ビット）を合わせて 11 ビットなので、これを 8 ビット長固定で対応するには、最初に 0 を 5 ビット追加すればいいことになります。

リードの場合は、コマンドは 10 なので、00000110 を送ればいいことになります。

マイクロ・ワイヤ方式の EEPROM 接続については、「CSI によるマイクロワイヤ(Microwire)メモリの制御 [78K0/Kx2 など]」もご参照ください。プログラム例については、「CSI によるマイクロワイヤと EEPROM の制御プログラム例 [78K0/Kx2]」もご参照ください。

シリアル・アレイの使い方 [78K0R]

Q1. シリアル・アレイを 3 線式シリアルで使う場合、ユニット 0 の 4 チャネルは、すべて独立に使用できますか。

A1. デバイスで異なります。78K0R のシリアル・アレイの 3 線式シリアルは、次のようなチャネルが使用可能です。

ユニット	チャネル	78K0R/KE3	78K0R/KF3	78K0R/KG3	78K0R/KH3	78K0R/KJ3
0	0	CSI00	CSI00	CSI00	CSI00	CSI00
	1	—	CSI01	CSI01	CSI01	CSI01
	2	CSI10	CSI10	CSI10	CSI10	CSI10
	3	—	—	—	CSI11	CSI11
1	0	—	CSI20	CSI20	CSI20	CSI20
	1	—	—	—	CSI21	CSI21
	2	—	—	—	—	—
	3	—	—	—	—	—

この表のように、78K0R/KE3～78K0R/KG3 では、ユニット 0 のシリアル・アレイのチャネル 3（78K0R/KE3 はチャネル 1 も）は 3 線式シリアルで使用できず、2 チャネル（78K0R/KE3）か 3 チャネル（78K0R/KF3、KG3）が使用可能です。

なお、これらは 3 線式シリアル(CSI)としては独立で使用できますが、調歩同期通信の UART と組み合わせて使用する場合は注意が必要です。UART では、偶数チャネルを送信用、奇数チャネルを受信用として、2 チャネル（チャネル 0 と 1、またはチャネル 2 と 3 の組み合わせ）を占有します。これにより、送信と受信を独立に処理でき、全二重通信をサポートしています。

78K0R/KG3 の場合について、ユニット 0 の各チャネルの使い方についての一覧を基に説明します。UART を 1 チャネル使用するとしましょう。

もし、UART0 を使うなら、兼用になっている 3 線式シリアルの CSI00 と CSI01 の 2 チャネルが使えなくなり、CSI10 だけが使用できます。もし、UART1 を使うとすると、CSI10 が使えなくなるだけで、CSI00 と CSI01 の 2 チャネルが使えます。このように、組み合わせる機能を十分に考慮する必要があります。

ユニット	チャネル	CSI で使用	UART で使用	簡易 I²C で使用
0	0	CSI00	UART0	—
	1	CSI01		—
	2	CSI10	UART1	IIC10
	3	—		—

Q2. シリアル・アレイを UART で使い、データ受信時の応答送信をする場合、最初にデータ受信用にレジスタを設定しておき、送信が必要になったらデータ送信用にレジスタを設定変更すればいいですか。

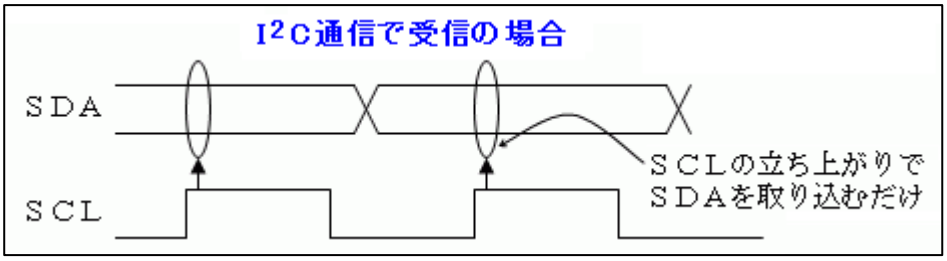
A2. いいえ、違います。78K0R のシリアル・アレイでは、UART を構成する場合に、ふたつのチャネルを使用します。たとえば、UART0 として使用する場合にはチャネル 0 とチャネル 1 を組み合わせます。その上で、偶数チャネル（この場合にはチャネル 0）を送信で、奇数チャネル（この場合にはチャネル 1）を受信で使用します。このように 2 つのチャネルを組み合わせれば、受信と送信を同時に処理（全二重通信）できます。
受信と送信を切り替えながら通信を行う（半二重通信）こともできますが、あまり一般的ではありません。

Q3. シリアル・アレイを 3 線式シリアル（CSI10）で使いたいのですが、SPS0＝0x0000、SDR01＝0x0000（動作クロック＝f_{CLK}/2）に設定すると、正常動作しません。SDR01＝0x0200（f_{CLK}/4）では正常に動作します。

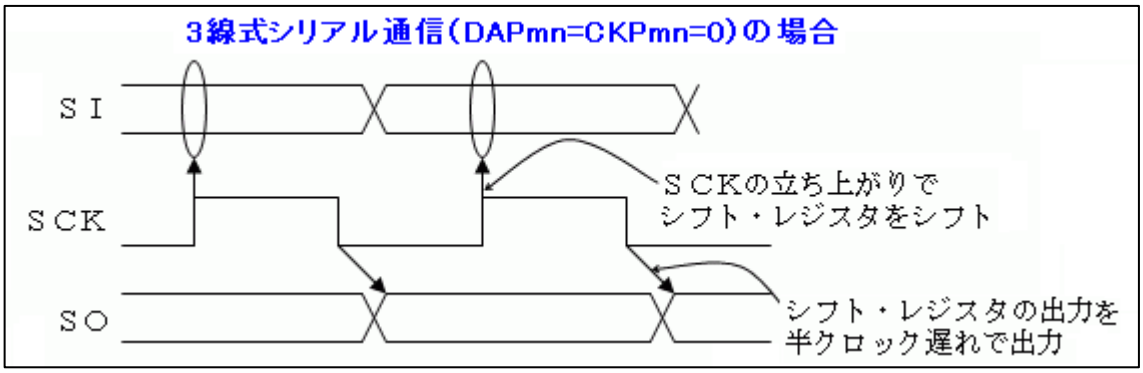
A3. 3 線式シリアルとしての転送レート（＝シリアル・クロック）の最大値はf_{CLK}/4 です。それ以上の周波数のシリアル・クロック設定では、正常動作を保証できません。マニュアルの「マスタ送信」または「マスタ受信」の表に記載された「転送レート」の範囲でご使用ください。

Q4. 3 線式シリアルや簡易 I²C は 1 チャネルで送受信可能なのに、なぜ UART は 2 チャネル必要なのですか。

A4. I²C は半二重通信で、ある時点ではマスタかスレーブのどちらかが送信を行い、他方は受信をするだけです。そのため、シフト・レジスタは 1 つで十分です。



また、3 線式シリアル通信は、UART のように送信と受信を同時に処理できますが、通信のためのクロックを共有しており、マスタとスレーブが同じクロック・エッジでデータを出力し、逆のエッジでデータを取り込みます。つまり、全く同じタイミングで動作するので、1 チャネル（シフト・レジスタ）で送信と受信を処理できます。

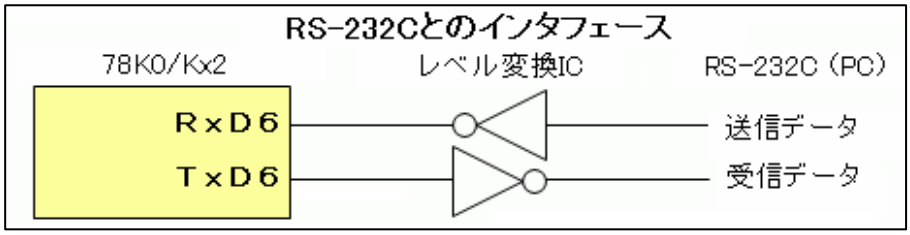


これに対して、UART の全二重通信では、受信と送信は全く独立に動作します。このために、独立した 2 つのシフト・レジスタが必要になります。このため、UART では 2 チャネルを必要とします。

UART6 での PC との接続 [78K0、一部の小ピン・マイコン]

Q1. 78K0/Kx2 の UART6 で、パソコンの RS-232C と通信させるには、どのように設定すればいいですか。

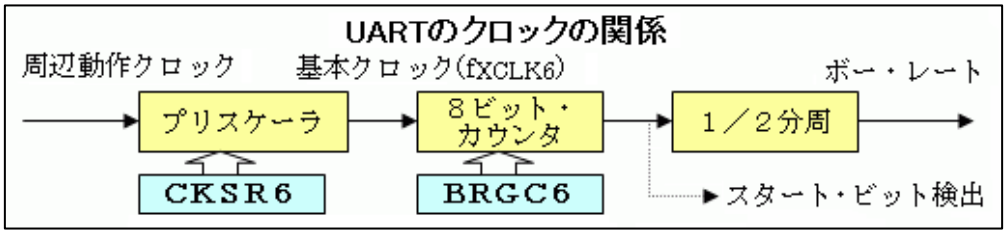
A1. 78K0/Kx2 は、そのままでは RS-232C の電圧レベルに対応できないので、RS-232C 用のレベル変換 IC でレベル変換をします。データ信号線の接続だけを次図に示します。



これで電圧レベルは合いますが、調歩同期通信では、送信側と受信側のフォーマットや通信速度を合わせる必要があります。このため、UART6 では ASIM6 レジスタでデータ・フォーマット（データ長やパリティなど）を選択し、専用ボー・レート・ジェネレータで通信速度を合わせてください。

Q2. 78K0/Kx2 を 8.38MHz で動作させ、RS-232C は 9600bps で通信する場合、専用ボー・レート・ジェネレータの設定値をどのように決めればいいですか。

A2. 通常は、送信側と受信側の合計で約 4%以下の誤差しか許されないので、その半分の 2%以下の誤差に設定する必要があります。高速内蔵発振クロックを使う場合は、クロック周波数に 3～5%の誤差があるので、そのままでは使用できません。通信の最初にダミー・データをやり取りするなどして、誤差を補正する必要があります。UART6 で、通信速度を指定するボー・レート・クロック発生部の構成を次に示します。



受信信号（RxD6）からデータの始まりであるスタート・ビットを検出する（「調歩同期の基本動作(ほとんどのデバイスで共通的な内容)」参照）ために、上図の基本クロックを使用します。精度よく検出するには、基本クロックの周波数は高いほうがよく、そのため 8 ビット・カウンタでの分周比をできるだけ大きくするように、全体の分周動作を設定します。

具体的には、周辺機能動作クロック周波数を使用したいボー・レートで除算し、さらに 1/2 倍することで、必要な分周比を求めます。次に、**8 ビット・カウンタでの分周比が 255 以下でできるだけ大きくなる**ように、プリスケーラで分周します。その結果を CKSR6 レジスタと BRGC6 レジスタに設定します。

8.38MHz を 9600bps で除算すると 872.9 となり、これをさらに 1/2 倍すると 436.4 となります。これは 255 より大きいので、さらに 1/2 倍して 218.2 となります。この結果、**プリスケーラで 1/2 に分周し、8 ビット・カウンタで 1/218 に分周**することになります。レジスタ設定は、次のようになります。

```
MOV     CKSR6,#1        ; プリスケーラで 1/2 分周
MOV     BRGC6,#218      ; 1/218 分周する
```

C 言語記述では、次のようになります。

```
CKSR6 = 0x01;           /* プリスケーラで 1/2 分周 */
BRGC6 = 218;            /* 1/218 分周する          */
```

このようにして得られたボー・レートの誤差は 0.229/218 で、約 0.1%の大きさ（一般的なセラミック発振子の周波数精度と同程度）となります。

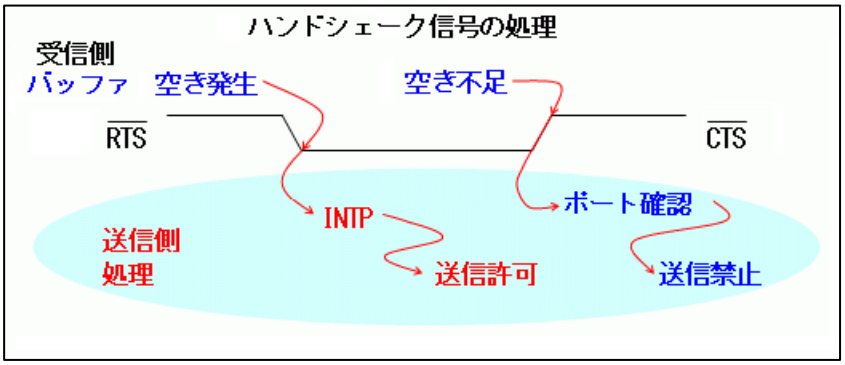
マイコン内蔵の UART での制御信号処理 [共通]

Q1. MPSC をマイコン内蔵の UART に置き換えて調歩同期通信をする場合、$\overline{\text{CTS}}$ 信号をどうすればいいですか。

A1. UART で $\overline{\text{RTS}}$ や $\overline{\text{CTS}}$ のような制御信号を利用するには、出力ポートと外部割り込み入力を利用します。

受信では、受信バッファに空きができたら $\overline{\text{RTS}}$ 信号を立ち下げて、送信許可状態であることを示します。逆に受信バッファの空きが少なくなると、送信を止めるように、$\overline{\text{RTS}}$ 信号を立ち上げます。

送信では、$\overline{\text{CTS}}$（送信許可）入力信号は、外部割り込みと入力ポートで処理します。その立ち下がりエッジ（送信許可）を割り込みで検出して、バッファにたまったデータを送信し、そのハイ・レベルをポートで読むことで、新たな送信を停止します。この関係を次図に示します。

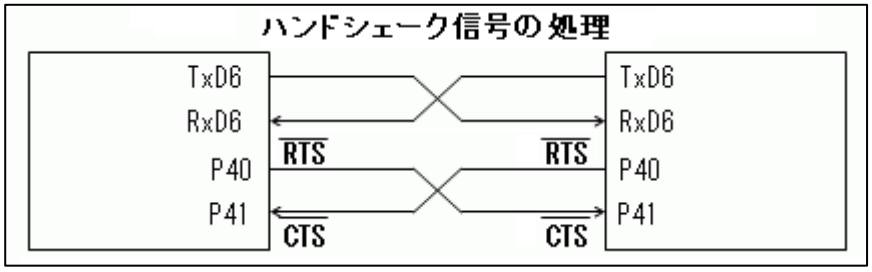


バッファの制御の概念については、「予備知識」に掲載されているバッファもご参照ください。

Q2. マイコン内蔵の UART で、$\overline{\text{CTS}}$ などの制御信号を用いた調歩同期通信の実現方法を教えてください。

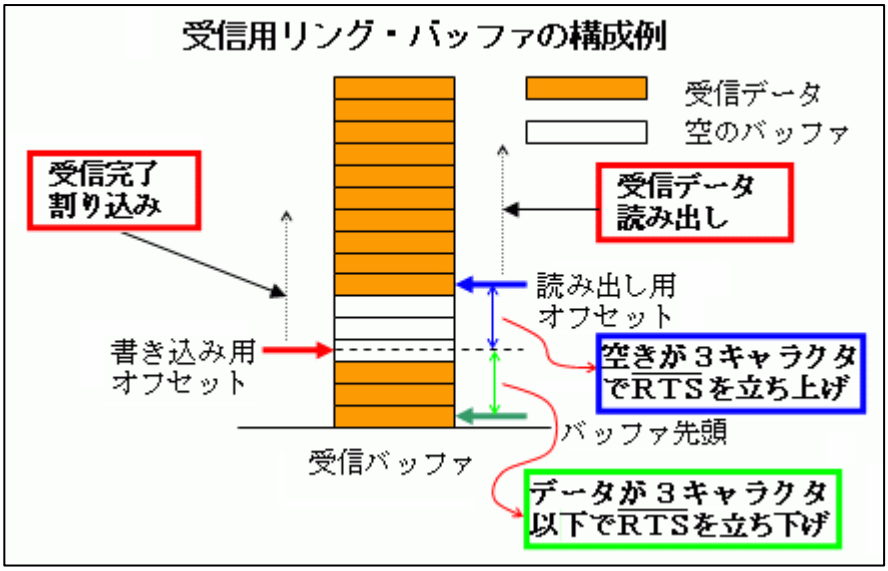
A2. ここでは、78K0S/KA1+に内蔵された UART6 を用いた例を考えてみます。UART6 はシフト・レジスタとバッファ・レジスタの 2 段バッファ構成になっていますが、複数キャラクタを送受信するには、別に送信用と受信用のバッファ・メモリが必要です。また、通常の処理と送受信処理を並行に行えるよう、割り込みで制御します。

ここでは、受信バッファの状態を送信側に知らせるために、P40 を $\overline{\text{RTS}}$（送信要求）信号、P41（INTP3）を $\overline{\text{CTS}}$（送信許可）信号に使用します。次図はこの接続例で、対応した信号同士をクロスで接続します。



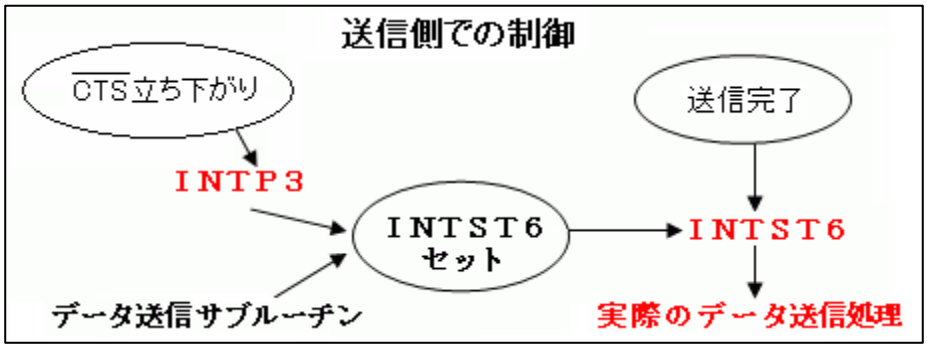
バッファ・メモリとして、送受信ともに 16 バイトの容量を**リング・バッファ**で確保し、16 バイト中のどれかを示すには、**4 ビットのオフセット**を使用します。書き込み用と読み出し用の 2 つのオフセットを比較すれば、**0**（**バッファが空**）から **15**（**バッファに 15 バイトのデータを蓄積**）までの状態を表せます。バッファの確実な制御のため、**16 バイトが蓄積**された状態は**オーバフロー**として扱います（そのような状態にしない）。

受信側でのリング・バッファの構成例を次図に示します。左側に受信処理を示し、右側が受信バッファからの読み出しを示します。受信データのバッファへの書き込みにより、受信バッファに 13 バイトのデータが埋まる（読み出し用オフセットから書き込み用オフセットを減算した値（空き）が 3 キャラクタになる）と、送信要求（$\overline{\text{RTS}}$ 信号）をオフにします。バッファの空きが 3 キャラクタ以下になると送信要求をオンにします。$\overline{\text{RTS}}$ 信号の出力は単純なポート出力として制御します。



送信側では送信要求を送信許可（$\overline{\text{CTS}}$ 信号）として扱い、オフしたら新たな送信を中止します。$\overline{\text{CTS}}$ 信号は、立ち下がりエッジ検出による割り込み要求を送信許可（通信再開）のトリガで使用するとともに、次の送信データの書き込み時に、ポート入力として確認で使用する方法を組み合わせます。

この制御を行う際の、送信側の処理を次に示します。送信処理では、送信するデータをいったん送信バッファに蓄積します。その後、送信可能なら送信処理を起動します。なお、**起動処理**は、単に**送信完了割り込み要求フラグをセットするだけ**です。また、受信側からの $\overline{\text{CTS}}$ 信号の立ち下がり割り込み要求でも、**起動処理**を行います。**実際のデータ送信は、送信完了割り込み（INTST6）処理の中で行います。**INTST6 処理では、送信が可能で送信するデータがあれば、次のデータを UART6 の送信バッファに書き込んで、送信をします。また、全データの送信が完了すると、**送信中フラグ**（これは起動処理の中でセット）をクリアして、処理を終了します。



プログラムは大きく分けて、次の 5 つの部分から構成されます。

①送信バッファへの送信データの追加（**putTxDATA**）
②受信バッファからの受信データの読み出し（**getRxDATA**）
③$\overline{\text{CTS}}$立ち下がり割り込み処理（**CTSONINT**）
④受信完了割り込み処理（**RXRDYINT**）
⑤送信完了割り込み処理（**TXRDYINT**）

これ以外に、受信エラー割り込み処理と、バッファの状態をチェックするサブルーチンがあります。

①送信バッファへの送信データの追加（putTxDATA）

Aレジスタに送信したいデータをセットし、サブルーチン・コールすることで使用します。**バッファ内のデータが 15 キャラクタ**になると、それ以降は**キャリーをセットしてエラー**を示します。

15 キャラクタ未満の場合は、書き込み用の**オフセット**（下位 4 ビットだけ使用）で書き込むアドレスを算出して、そこに送信データを格納します。オフセットを更新（+1）してバッファ処理が完了し、UART6 の制御を行います。このとき割り込み要求が入ると、余分な送信完了割り込みが発生する可能性があるので、以降は**排他制御**のために**割り込み禁止で処理**します。$\overline{\text{CTS}}$ 信号がハイ･レベルなら、新たな送信は起動しません。そうでないときには、**起動中フラグをセット**し、**送信完了割り込み要求フラグをセット**して、処理を終了します。

```
;
;   送信バッファへのデータの書き込み
;
putTxDATA:
        PUSH    HL
        PUSH    AX
        CALL    !CHKTXDATA      ; バッファをチェック
        ADD     A,#0F1H         ; バッファ・フルでキャリー
        BC      $TXEXIT         ; エラーで終了
        MOV     A,#0FH
        AND     A,TXWTPT        ; 下位 4 ビットを抽出
        ADD     A,#LOW TXBUFF   ; オフセットを加算
        MOV     L,A
        MOV     H,#HIGH TXBUFF
        POP     AX              ; データを復帰
        PUSH    AX
        MOV     [HL],A          ; データをバッファにセット
        INC     TXWTPT          ; 書き込みオフセット更新
        DI                      ; フラグ操作の排他処理のため
;
;   送信が止まっているので、送信可能なら再開する
;
        BT      P4.1,$TXEXIT    ; CTS 信号がハイなら再開しない
        SET1    TXFLAG.0        ; 送信中フラグをオン
        SET1    STIF6           ; 起動のための割り込み要求セット
TXEXIT:
        EI
        POP     AX
        POP     HL
        RET
```

オフセットは下位の 4 ビットだけが有効なデータですが、更新は 8 ビットまで行います。オフセットから実際のバッファのアドレスを算出するときに、下位の 4 ビットだけを使用するようにしています。

②受信バッファからの受信データの読み出し（getRxDATA）

受信バッファが空なら、**ゼロ･フラグ**をセットして戻ります。データがあり、**残りが規定値（BUFFEMP）以下**なら、$\overline{\text{RTS}}$ **信号をロウ･レベル**にして受信可能であることを示します。読み出し用のオフセット（下位ビット）で読み出すデータのアドレスを算出して、バッファからデータを読み出します。その後、読み出し用オフセットを更新して処理を終了します。

```
;
;   受信バッファからデータの読み出し
;
;出力
```

```
;データがあるとき
;  A レジスタ        : 読み出しデータ
;  ZERO              : 0
;データがないとき
;  A レジスタ        : 0
;  ZERO              : 1
;
getRxDATA:
        CALL    !CHKRXDATA      ; データ数チェック
        BZ      $GETEXIT        ; 空なら分岐
        PUSH    PSW             ; フラグをセーブ
        CMP     A,#BUFFEMP      ; 残りをチェック
        BNC     $GETNEXT        ; 指定値以下なら
        CLR1    P4.0            ; RTS 信号を立ち下げ
GETNEXT:
        PUSH    HL
        MOV     A,#0FH
        AND     A,RXPOINTER     ; オフセット計算
        ADD     A,#LOW RXBUFF   ; 下位アドレスの計算
        MOV     L,A
        MOV     H,#HIGH RXBUFF
        MOV     A,[HL]          ; データの読み出し
        INC     RXPOINTER       ; 読み出しポインタ更新
        POP     HL
        POP     PSW             ; フラグを復帰
GETEXIT:
        RET
```

③$\overline{\text{CTS}}$ 立ち下がり割り込み処理(CTSONINT)

$\overline{\text{CTS}}$ 信号の立ち下がり(**受信側のバッファに空き**ができた)で発生する割り込み処理です。$\overline{\text{CTS}}$ 信号を入力ポートP41から読んで、ロウ・レベルである(ノイズ入力でない)こと確認します。ロウ・レベルなら、**送信許可の処理**をします。すでに送信が起動されている(送信中フラグをセット)か、送信するデータがなければ、何もせず終了します。送信するデータが残っていれば、送信中フラグをセットし、送信完了割り込みを起動します。

```
;
;  INTP3(立ち下がり検出)割り込み処理
;
;  CTS 信号が立ち下がり、送信可能になったときの処理
;
CTSONINT:
        BT      P4.1,$CTSONEXIT ; ノイズ対策
        BT      TXFLAG.0,$CTSONEXIT
;
;  送信中ではないときの処理
;
        PUSH    AX
        CALL    !CHKTXDATA      ; バッファをチェック
        POP     AX
        BZ      $CTSONEXIT
        SET1    TXFLAG.0        ; 送信中フラグをオン
        SET1    STIF6           ; 起動のための割り込み要求セット
CTSONEXIT:
        RETI
```

④受信完了割り込み処理(RXRDYINT)

受信完了割り込みで起動され、受信バッファへの書き込み用オフセットで書き込むアドレスを算出し、受信したデータを保存します。オフセットを更新後、バッファ内のデータ数が規定値(BUFFFULL)以上なら、$\overline{\text{RTS}}$ 信号をハイ・レベルに立ち上げます。

```
;
;  UART6 データ受信処理
;
RXRDYINT:
        PUSH    AX
        PUSH    HL
        MOV     A,RXWTPT        ; 受信データ書き込みオフセット
        AND     A,#0FH
        ADD     A,#LOW RXBUFF
        MOV     L,A
        MOV     H,#HIGH RXBUFF  ; ポインタをセット
        MOV     A,RXB6          ; 受信データ読み出し
        MOV     [HL],A          ; データをバッファに保存
        INC     RXWTPT          ; オフセットを更新
        POP     HL
        CALL    !CHKRXDATA      ; バッファの状態をチェック
        CMP     A,#BUFFFULL
        BC      $RXRDYEXIT      ; 13 バイト以下なら戻る
        SET1    P4.0            ; RTS 信号を立ち上げ
RXRDYEXIT:
        POP     AX
        RETI
```

⑤送信完了割り込み処理(TXRDYINT)

実際のデータ送信処理を行う部分です。データ送信は、必ずこの送信完了割り込み処理の中で行われます。処理を開始すると、TXB6 が空かどうかを確認します(空のはずだが念のため)。次に、CTS 信号がハイ・レベルになっているか、リング・バッファが空になっていれば、新たなデータの送信は行いません。さらに、**TXS6 が空**なら**送信中フラグをクリア**します。このとき、UART6 の送信完了の割り込みは、それ以上発生しません。次の起動は、プログラムによる putTxDATA での新しい送信データの書き込みでのセットか、その後の **INTP3** 割り込み処理の中でのプログラムでのセットによるものです。

送信可能状態で、次に送信すべきデータがリング・バッファに残っていれば、そのデータを UART6 に書き込んで、読み出し用のオフセットを更新します。

```
;
;  UART6 データ送信完了処理
;
TXRDYINT:
        PUSH    AX
        PUSH    HL
        MOV     A,ASIF6
        BT      A.1,$TXRDYEXIT  ; 送信中で抜ける
;
;  TXB6 が空なら、次データ送信のために CTS 信号をチェックする
;  CTS 信号がオンなら残りデータの送信を継続し、オフなら現在の
;  データの送信完了で、送信中ステータスをクリアする
;
```

```
        BT      P4.1,$CTSOFF
        CALL    !CHKTXDATA      ; 残りデータをチェック
        BZ      $NODATA
        MOV     A,TXPOINTER     ; 出力データ用オフセット読み出し
        AND     A,#0FH
        ADD     A,#LOW TXBUFF
        MOV     L,A
        MOV     H,#HIGH TXBUFF
        MOV     A,[HL]          ; データ読み出し
        MOV     TXB6,A          ; 送信
        INC     TXPOINTER       ; 読み出し用オフセット更新
        POP     HL
        POP     AX
        RETI
CTSOFF:
NODATA:
                BT      A.0,$TXRDYEXIT
; 送信動作を完了
        CLR1    TXFLAG.0
TXRDYEXIT:
        POP     HL
        POP     AX
```

⑥その他の処理

UART6 の受信でエラーが発生した場合の処理や、送受信のリング・バッファの状態をチェックする処理です。

受信でエラーが発生した場合は、そのときの ASIS6 の内容をエラー・ステータスとしてセットし、受信したデータは読み飛ばします。

リング・バッファの状態のチェックは、書き込みと読み出しの 2 つのオフセットを比較(書き込み用から読み出し用を減算)し、結果の下位 4 ビットに書き込まれたデータ数を得ています。

```
;
;  UART6 受信エラーの処理
;  ASIS6 レジスタの内容をフラグにコピーするだけ
;
RXERRORINT:
        PUSH    AX
        MOV     A,ASIS6         ; ステータスの読み出し
        MOV     RXSTATUS,A
        MOV     A,RXB6          ; データのダミー読み出し
        POP     AX
        RETI
;
;  受信データ数確認
;
;出力
;  Ａレジスタ    ：バッファされたデータ数
;  ZERO          ：空なら 1
;
```

```
CHKRXDATA:
        MOV     A,RXWTPT        ; 書き込みオフセットの読み出し
        SUB     A,RXPOINTER     ; 書き込み用－読み出し用
        AND     A,#0FH          ; 有効ビットだけ抽出
        RET
;
;  送信待ちデータ数確認
;
;出力
;  Ａレジスタ    ：送信待ちデータ数
;  ZERO          ：空なら１
;
CHKTXDATA:
        MOV     A,TXWTPT        ; 書き込みオフセットの読み出し
        SUB     A,TXPOINTER     ; 書き込み用－読み出し用
        AND     A,#0FH          ; 有効ビットだけ抽出
        RET
```

リング・バッファ関係のメモリの定義は、次のとおりです。初期化で、これらの内容は 0 にクリアしておきます。

```
;
;  送受信用 16 バイトのリング・バッファ領域
;
        DSEG    SADDRP
RXBUFF:
        DS      16              ; 受信バッファ
TXBUFF:
        DS      16              ; 送信バッファ
;
;  リング・バッファのアクセス用オフセットは各バッファで読み出しと
;  書き込みを１バイトずつ準備し、その下位 4 ビットを有効とする
;
RXPOINTER:
        DS      1               ; 受信データ読み出しオフセット
RXWTPT:
        DS      1               ; 受信データ書き込みオフセット
TXPOINTER:
        DS      1               ; 送信データ読み出しオフセット
TXWTPT:
        DS      1               ; 送信データ書き込みオフセット
RXSTATUS:
        DS      1               ; 受信エラー・ステータス
TXFLAG:
        DS      1               ; 送信動作モード・フラグ
;  ビット 0 が 1 なら送信動作中、0 なら送信していない
```

また、ベクタ･テーブル関係とプログラムの起動処理関係の例は以下のようになります。使用していない割り込みのベクタには、ダミーのアドレスを設定しています。

```
        DSEG    IHRAM
        DS      20H             ; スタック領域確保
STACKTOP:
RSTVECT CSEG    AT      0
        DW      START           ; 00:リセット・スタート処理のベクタ
```

```
INTVECT CSEG    AT      18H     ; ここでは以下の割り込みを定義
        DW      CTSONINT        ; 18:INTP3
        DW      DUMMY           ; 1A:INTTM80
        DW      RXERRORINT      ; 1C:INTSRE6
        DW      RXRDYINT        ; 1E:INTSR6
        DW      TXRDYINT        ; 20:INTST6
@@OPTB  CSEG    AT      0080H
        DB      10011100b
        DB      11111111b
START:
        MOVW    AX,#STACKTOP
        MOVW    SP,AX           ; RAM の先頭部分にスタックを設定
```

このあと、UART6 を使用するための初期設定が、次の例のように続きます。

```
;
;  メモリの初期化
;
        MOVW    HL,#RXBUFF
        MOV     A,#0
MEMLOOP:
        MOV     [HL],A
        INC     L
        BNZ     $MEMLOOP
        MOV     CKSR6,#1                ; プリスケーラで 1/2 分周
        MOV     BRGC6,#218              ; 1/218 分周する
        MOV     ASIM6,#10010000b        ; 受信エラーで INTSRE6 発生
        MOV     ASICL6,#00000010b
        SET1    TXE6                    ; 送信許可に設定
        SET1    RXE6                    ; 必要なら受信許可も設定
        MOV     P4,#00001000b           ; 兼用のポートをハイに設定
        CLR1    PM4.3                   ; TxD6 端子を出力モードに設定
;
;  ハンドシェイク関係の初期化
;
        CLR1    PM4.0                   ; RTS 端子を出力モードに設定
        MOV     INTM1,#00               ; 立ち下がりエッジに設定
;
;  割り込み関係の初期化
;
        MOV     IF0,#00                 ; 割り込み要求クリア
        MOV     IF1,#00
        MOV     MK0,#0FFH
        MOV     MK1,#10001011b          ; シリアルと INTP3 のマスク解除
        EI                              ; 割り込み許可
```

⑦使い方

送信する場合は、A レジスタに送信データをセットして、putTxDATA をコールします。バッファがいっぱいなら、キャリーを見てループさせています。同様に受信では、getRxDATA をコールします。データがなければ、ゼロ・フラグで確認します。

```
;
;   送信の例
;
        MOV     A,#41H
TX15LOOP:
        CALL    !putTxDATA              ; データ送信
        BC      $TX15LOOP               ; エラーなら繰り返し
;
;   受信の例
;
RXCHARLOOP:
        CALL    !getRxDATA              ; データ読み出し
        BZ      $RXCHARLOOP             ; データがなければループ
```

ソフトウエアによる UART 機能 [78K0、小ピン・マイコン共通]

Q1. 内蔵の UART だけでは不足なので、汎用のポートを用いて UART を実現するにはどうすればいいですか。

A1. ホーム・ページにソフトUARTとしてサンプル・プログラムが掲載されていますので、そちらをご参照ください。

Q2. サンプル・プログラムでは 16 ビット・タイマを使用していますが、これは別の用途に使用しています。8 ビットのタイマ H を使用するか、タイマを使用しないで実現はどうすればいいですか。

A2. 78K0S/KU1+(UART は内蔵せず)を例に実現方法を示します。なお、UART での受信データのスタート・ビットがいつ来ても対応できるように、外部割り込みを使用します。また、受信データ信号に P32/INTP1 を使用し、送信データ信号に P40 を使用するものとします。

なお、動作クロックは、高速内蔵発振器の出力である 8MHz を使うものとします。

ここでは、必要な考え方だけを示します。具体的なプログラムについては、「ソフトウエアによる UART プログラム例 1」(タイマHを使用したもの)、「ソフトウエアによる UART プログラム例 2」(タイマを使用しないもの)をご参照ください。

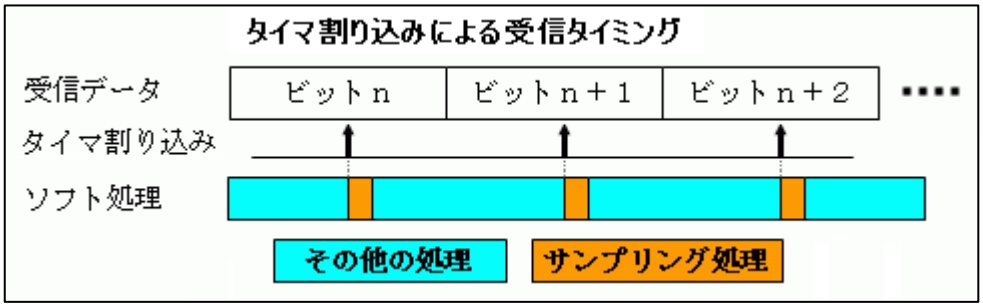
[対応する通信フォーマット]

ここでは、調歩同期通信で 9600～19200 ボーの通信速度、データ長は 8 ビット、パリティなし、ストップ・ビットは 1 ビット、LSB ファーストの通信に対応するものとします。また、通信は半二重で行うものとします。

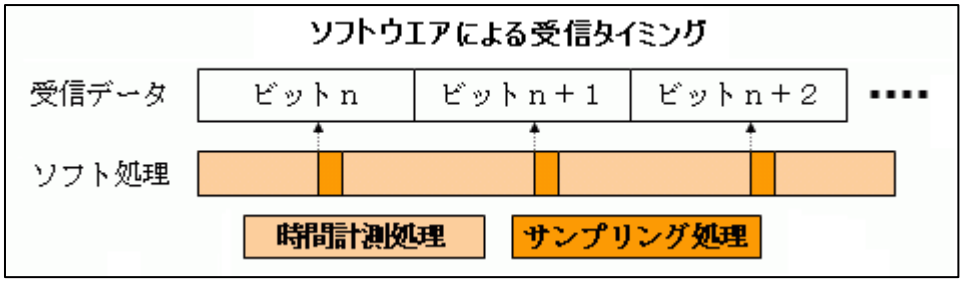
[サンプリング・タイミングの発生]

ソフトウエアで UART 機能を実現する場合、パラレルからシリアルへの変換や、その逆の変換以上に重要なのがタイミングで、特に受信(サンプリング)タイミングです。UART が対応している調歩同期通信では、やり取りするデータにスタート・ビットとストップ・ビットを付加して、受信した信号がどこから始まるかが分かるようになっています。スタート・ビットのあとは、送信側と受信側があらかじめ約束した速度でデータをやり取りするので、受信側は自分で受信タイミングを生成する必要があります(タイミングについては、「シリアル通信の同期／非同期の違い」をご参照ください)。

このタイミングを得るには、2 つの方法が考えられます。ひとつはタイマ機能を利用する方法です。この場合、タイミングはタイマ機能でハードウエア的に発生するので、ソフトウエアの処理はかなり自由度が大きくなります。通信速度が遅い場合は、割り込みを使うことで、**その他の処理**を行いながら並行して**サンプリング処理**を実現できます。このようすを次のタイマ割り込みによる受信タイミングに示します。



もうひとつは、命令の実行クロック数からタイミングを測る方法です。この方法では、タイマのようなリソースが不要ですが、命令の実行時間が正確に管理されている必要があります（UART では送信側と受信側の差を 4%程度以下にする必要がある）。そのため、割り込みを含めて、他の処理は一切実行できません。



なお、この場合も、受信するデータがいつ来るか分からないので、スタート・ビットの立ち下がりの検出には、エッジ検出の割り込みを使用します。

調歩同期通信については、「調歩同期の基本動作」をご参照ください。

[ボー・レートのキャリブレーション]

高速内蔵発振器からのクロックは、誤差が大きいので、そのまま使用すると、通信エラーが発生する可能性があります。そこで、使用する前にキャリブレーションを行って、誤差を小さくする必要があります。このための簡単な方法は、通信相手から特定のデータ（できるだけ同じ状態が長く続くデータ）を送ってもらい、その継続時間を計測して、相手の通信速度を測定し、それに合わせた通信速度に設定することです。ここでは、処理を簡単にするために、8 ビット分の連続したロウ・レベル（データは 80H）を受信して、ロウ・レベルの時間を計測します。具体的な処理は、次のようになります。

①RxD端子がハイ・レベルになるのを待つ。

②カウントの準備をして、HALT モードでスタート・ビットの立ち下がりを待つ。

③HALT モードが解除されたら、受信データの立ち上がりまで、ソフト・タイマで時間計測をする。

測定精度をできるだけ高くするには、カウントの刻みをできるだけ小さくする必要があります。しかし、最速の19200ボーの場合でも、8ビット分の時間は417μsで、30クロック程度の測定誤差でも1%にもならないので、特に気にする必要はありません。具体的なプログラム例を以下に示します。

```
CAL:    DI                              ; 割り込みは禁止
        MOV     INTM0,#00110000b        ; INTP1 は両エッジ検出に設定
        MOV     MK0,#11110111b          ; INTP1 のマスク解除
        MOVW    HL,#0
PRELOOP:
        BF      RxDPIN,$PERLOOP         ; 端子がハイになるのを待つ
        CLR1    PIF1                    ; 擬似検出をクリア
        HALT                            ; 2：立ち下がりエッジ待ち
;   割り込み受け付けには 3～5 クロック必要
        CLR1    PIF1                    ; 6:割り込み要求をクリア
```

```
MLOOP:
        NOP                             ; 2:時間調整用
        INCW    HL                      ; 4:時間計測
        BF      PIF1,$MLOOP             ;10:割り込み要求待ち
```

この時間計測のループは、エッジ検出から測定開始までが 9～11 クロック、その後のカウントは 16 クロック(2.0μs)の分解能です(余分に NOP 命令を追加すれば、分解能は低くなりますが、ここでは、その後の計算を楽することを優先させます)。受信データの立ち上がりを検出したらループを抜けます。このときの標準的なカウント数は、次のようになります。

9600 ボー　：417 カウント

19200 ボー：208 カウント

ここで得た回数をクロック数(1 カウント＝16 クロック)に変換し、測定したビット数(8)で除算すれば、(最終的に 2 倍すると)実際のデータ送受信タイミング(間隔)のクロック数を得ることができます。

この時間をタイマ H で計測する場合は、得られた値が上記の 2 倍になることから、そのままでは、8 ビットのタイマ H ではカウントできません。そこで、カウント・クロック数を周辺機能動作クロックの 1/4 に(上記カウント値を単純に 1/2 に)設定すれば、タイマ H でのカウント数が得られます。この値は、1 ビットあたりのカウント数になります。

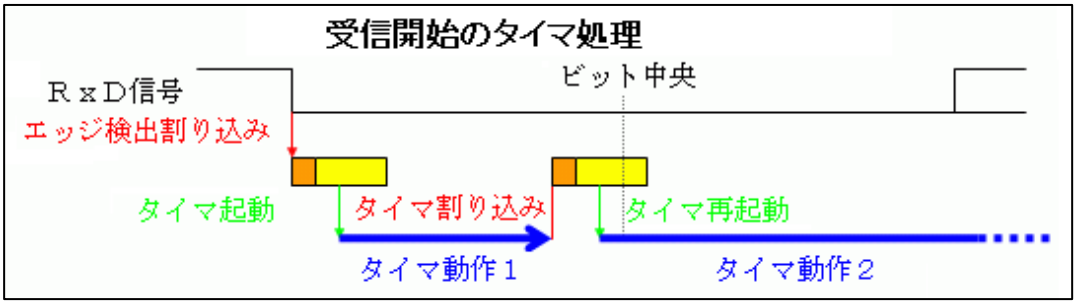
ソフトウエアで受信タイミングを生成する場合は、前に掲載したタイミング図のサンプリング処理の部分を除いた時間を計測プログラムのループ時間で除算します。

UART での送信では、1 ビットの時間間隔だけでいいのですが、受信の場合は 1/2 ビット分の時間も必要になります。次図のように、スタート・ビットの立ち下がりから 1/2 ビット経過したタイミングで、スタート・ビットの確認を行い、以降は 1 ビット時間の間隔で信号のサンプリングをします。ここで注意を要するのが、割り込み処理での処理時間による遅延です。



小ピン・マイコンでは、割り込み受け付けで 9～19 クロックかかります。さらに割り込み処理でも、実際に端子のレベルを読むまでに時間がかかります。スタート・ビット検出では、立ち下がりエッジ検出と 1/2 ビット分の時間を計測したタイマの割り込みを処理する必要があります。これらの時間の合計を補正しないと、ビットの中央でサンプリングできなくなります。そのため、立ち下がりエッジ検出割り込みで計測を開始する 1/2 ビット時間に補正を施します(必要な処理時間分だけ短くします)。

この補正値の関係を詳細に示したのが、次の図です。**エッジ検出割り込み**が発生しても、その受け付けには**割り込み応答時間**が必要です。**割り込み処理**が起動すると、その中で時間計測のために**タイマ**を**起動**します。タイマHが起動され、**タイマ動作 1** で必要な時間を計測しますが、ここではピッタリ中央にはしません。

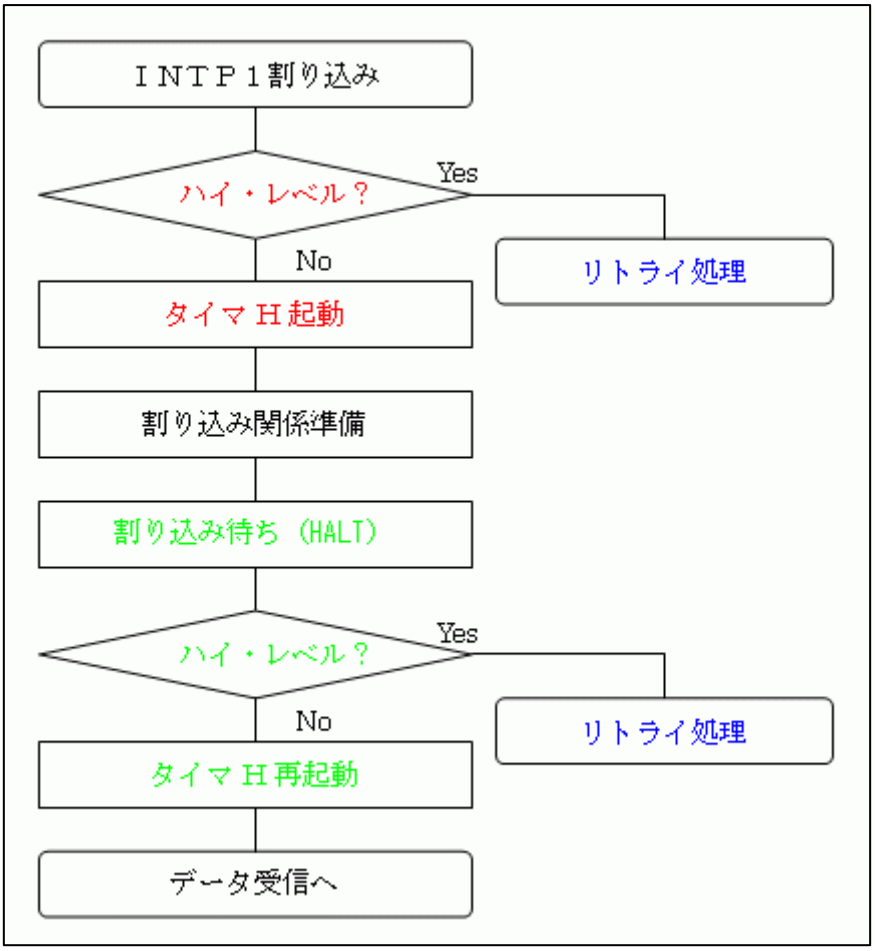


実際のデータのサンプリングが、ビットの中央になるように、**タイマ動作 2** がビット中央の少し前で完了できるタイミングで、**タイマ動作 2** を起動するようにします。**タイマ動作 1** が完了して、**割り込み応答時間**の経過で**割り込み処理**が起動して、実際のサンプリングを行うように、ビット周期に合わせて**タイマ**を**再起動**します。これらの時間（実行クロック数）は、プログラムで決まるので、定数として定義しておきます。プログラムを変更して、処理時間が変わった場合は、この定数の定義を変えるだけで済みます。

[受信割り込み処理]

実際にデータの受信を行う場合は、まず、INTP1 のエッジ検出割り込みを許可して、信号の立ち下がりを待ちます。割り込み要求が受け付けられたら、信号がロウ・レベルになっていることを確認します。確認できたら、タイマを起動してスタート・ビットの中央を待ちます（このときは、ここまでの処理時間と、スタート・ビット中央でデータ・サンプリングのタイミング用にタイマ H を再起動するまでを補正（その分短く）した時間となります）。

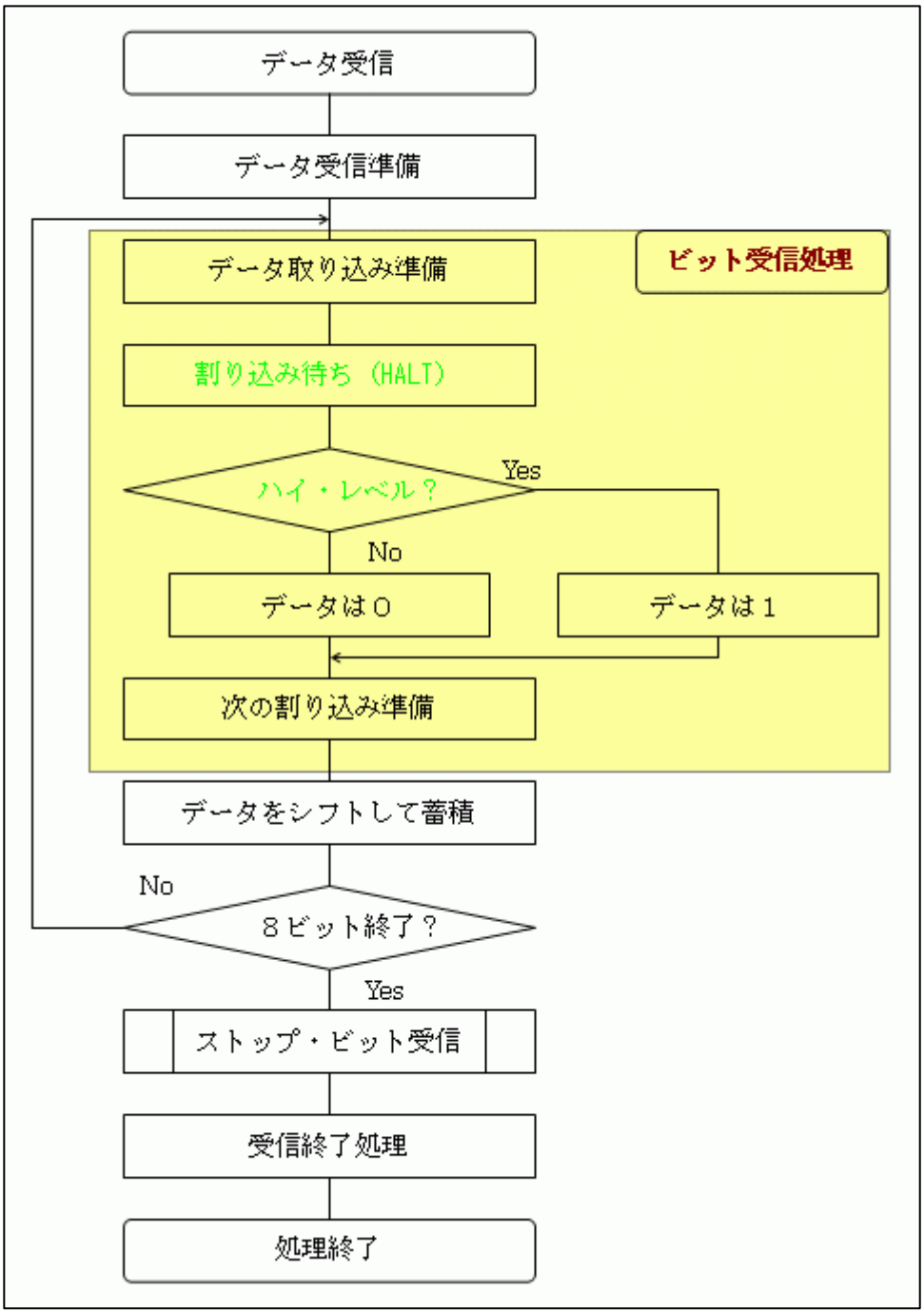
次のフロー・チャートで、**赤字の処理**がその後の時間計測に影響する部分です。タイマ H を起動してしまえば、あとはタイマ H が時間測定を行うので、命令の処理時間をあまり気にする必要がありません。その後、スタート・ビットの中央確認タイミング待ちの処理を行い、HALT 命令でタイマ H 割り込み（INTTMH1）待ちをします。HALT 命令での割り込み（ここでは、HALT 状態の解除だけでベクタ処理はしない）待ちは、応答時間が速いので、スタート・ビットの中央確認からデータ・サンプリング用のタイマ H 再起動までの時間を短くできます。フロー・チャートの**緑字の処理**が、スタート・ビット中央検出から、データ・サンプリングのためのタイマ H 再起動までの処理となります。



もし、端子のレベルがハイ・レベルなら**青字の処理**となり、再度 INTP1 割り込み待ちの準備をして、割り込み処理から抜けます。
スタート・ビットの確認ができたら、引き続きデータ受信処理となります。

[データ受信処理]

データの受信は、タイマH割り込み（INTTMH1）を HALT 命令の解除に用いることによりタイミングを計っています。データ受信処理は1ビットずつ HALT 命令でタイマ H 割り込み待ちをしてタイミングを合わせて、RxD 端子の状態を取り込みます。



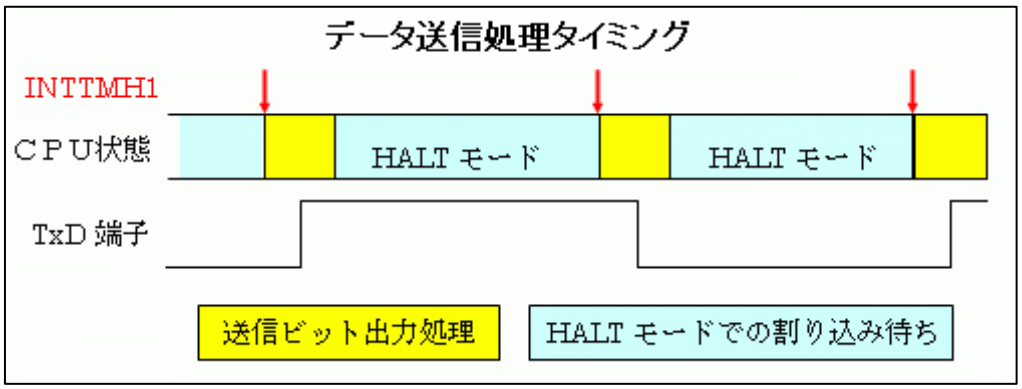
1 ビット分の受信処理はサブルーチン化してあり（上記フロー・チャートの色が付いた部分）、この処理を 8 ビット分繰り返します。この 1 ビット分の受信処理では、**緑字**で示された**タイマ H の割り込み待ちの HALT 処理**（HALT モードの解除時間部分）と **RxD 端子のレベル判定処理**が処理時間を考慮すべきところです。

このようにして取り込まれたビットは、シフトして 8 ビット分が蓄積されます。8 ビット分の取り込みが完了すると、次にストップ・ビットを受信して、結果をフレーミング・エラーに格納します。ストップ・ビットまでの受信が完了すると、タイマ H を停止し、フラグをセットして処理を終了します。

[データ送信処理]

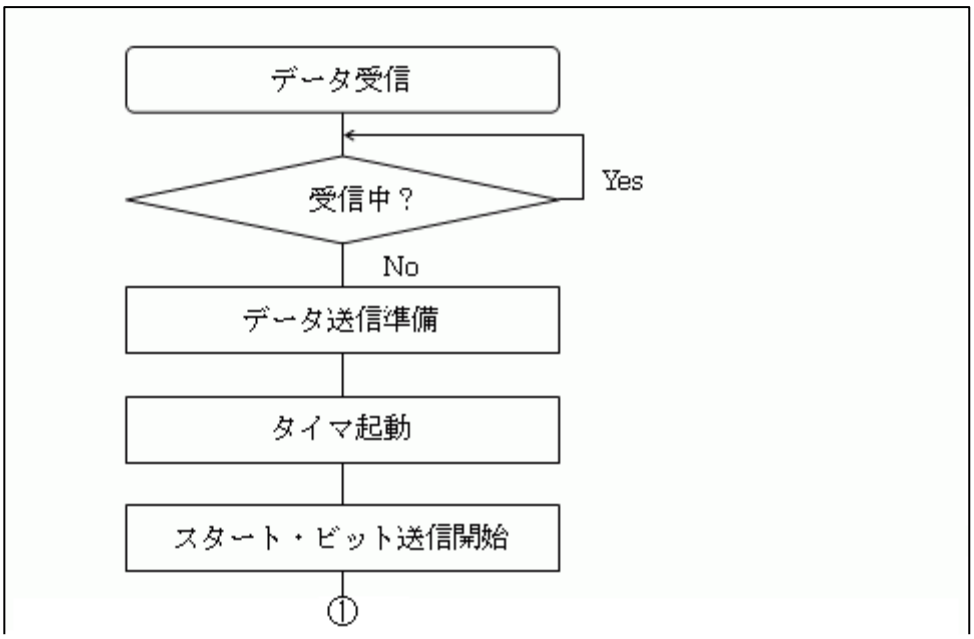
基本的に、ソフトウエア処理では半二重通信しかできないので、受信処理中（受信処理はスタート・ビットを割り込みで処理しているので、スタート・ビット待ちの状態でもメイン処理ができる）には、処理が終わるまで待ち、受信終了後に送信を行います。

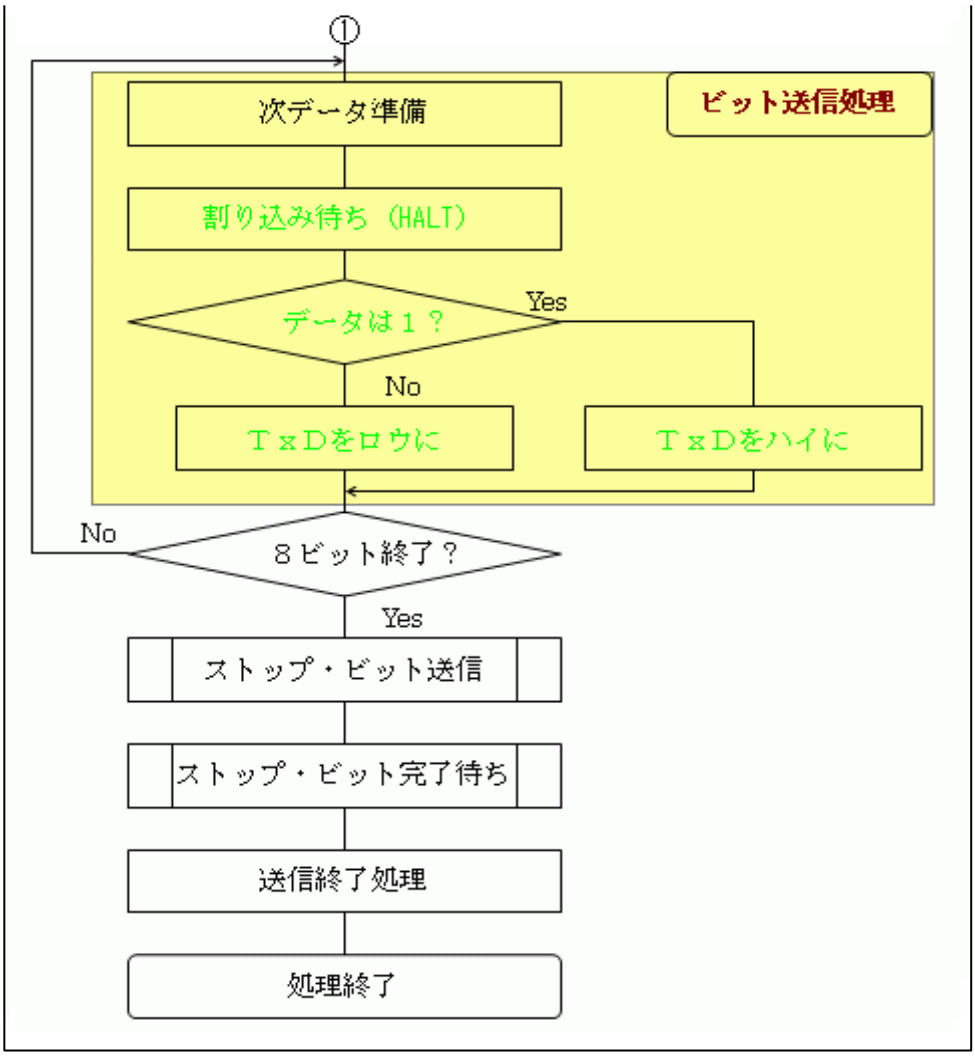
送信処理は、基本的に、タイマ H の割り込みタイミングで、TxD 端子として使用するポートを送信したいデータに合わせて操作するだけです。タイマ H 割り込みからポート操作までの時間については、その遅れの大きさをあまり気にする必要はなく、そのばらつきだけに注意が必要です。半二重通信では、送信中に受信処理を実行できないので、タイマ H の割り込みは、ベクタ処理でなく HALT モードの解除だけに使用します。次にデータ送信処理タイミングを示します。



送信が可能になったら、必要なパラメータを設定して、タイマ H を起動し、TxD 端子をロウ・レベルにすることで、スタート・ビットを送信開始します。

送信処理の処理フローを次に示します。**ビットの送信処理**（フロー・チャートの色が付いた部分）はサブルーチン化しておきます。ここでは、タイマHの割り込みでの処理時間をできるだけ短くして、処理時間にばらつきが発生しないようにしています（フロー・チャートの**緑字**で示した部分）。





ビット送信処理は、必要な時間を待ち、送信データを出力開始するだけです。実際にビットの送信が完了するのは、次のタイマH割り込みが発生するときです。そのため、1ビット目の送信処理でスタート・ビットの送信完了を待ち、ストップ・ビット送信でデータの 8 ビット目の送信完了を待つことになります。そこで、ストップ・ビットの送信完了を待つために、ストップ・ビットと同じ1を出力するように、ビット送信処理をコールします。これで、ストップ・ビットのうしろに少しむだな時間が入りますが、問題になることはありません。

ソフトウエアによる UART プログラム例(1) [小ピン・マイコン]

Q1. 8 ビットのタイマ H と汎用のポートを用いてUARTを実現するには、どうすればいいですか。

A1. ここでは、調歩同期通信で 9600～19200 ボーの通信速度、データ長は 8 ビット、パリティなし、ストップ・ビットは1ビットで、LSB ファーストの通信に対応するものとします。また、通信は半二重で行うものとします。
ソフトウエアによる UART を実現する考え方は、「ソフトウエアによる UART 機能」をご参照ください。

[プログラムの概要]

このプログラム例は、大きく分類して、次の部分で構成されています。

①ボー・レート・キャリブレーション部
②INTP1 によるデータ受信処理部
③受信処理起動部
④データ送信処理部

①のボー・レート・キャリブレーション部は、高速内蔵発振クロックを使用する際の補正をする部分です。外部クロックやセラミック発振子などを用いる場合には必要ありません。
②がスタート・ビットの立ち下がりによる INTP1 で起動される処理で、調歩同期通信の受信処理を行う部分です。基本的に割り込み処理なので、結果はレジスタではなく、RAM に受信したデータと結果のフラグを格納します。半二重通信なので、つねにスタート・ビットを待っているわけではありません。必要なときに、③の処理で起動する必要があります。1 キャラクタのデータを受信完了すると、INTP1 割り込みとタイマ H 割り込みはマスクされ、タイマ H は停止します。
③では、受信（実際にはスタート・ビット待ち）を起動するために、必要なフラグをセットして INTP1 割り込みを許可します。ここでは、受信処理の起動だけを行うので、スタート・ビットが来るまでは、送信以外の処理を実行可能です。
④はデータ送信処理を行います。送信タイミングをタイマ H の割り込みで得ていますが、ベクタ割り込みは使用していません。そのため、他の処理と並行して実行はできません。

[サンプリング・タイミングの発生]

このプログラムでは、受信でのサンプリング・タイミングや送信のためのタイミング生成用にタイマ H を用いています。ただし、受信の場合に実際のサンプリング・タイミングのばらつきを少なくするために、タイマ H の割り込みを HALT 命令で待ち合わせて処理しています。

[ボー・レートのキャリブレーション]

高速内蔵発振器からの、誤差が大きいクロックを使用することを考慮して、通信動作の前にキャリブレーションを行います。このための簡単な方法として、通信相手から 8 ビット分の連続したロウ・レベル（データは 80H）を受信して、その時間を計測します。考え方については、「ソフトウエアによる UART 機能」をご参照ください。

```
CAL:    DI                              ; 割り込みは禁止
        MOV     INTM0,#00110000b        ; INTP1 は両エッジ検出に設定
        MOV     MK0,#11110111b          ; INTP1 のマスク解除
        MOVW    HL,#0
PRELOOP:
        BF      RxDPIN,$PERLOOP         ; 端子がハイになるのを待つ
        CLR1    PIF1                    ; 擬似検出をクリア

        HALT                            ; 2：立ち下がりエッジ待ち
;  割り込み受け付けには 3～5 クロック必要
        CLR1    PIF1                    ; 6:割り込み要求をクリア
MLOOP:
        NOP                             ; 2:時間調整用
        INCW    HL                      ; 4:時間計測
        BF      PIF1,$MLOOP             ;10:割り込み要求待ち
```

ここで得られた回数をクロック数(1 カウント＝16 クロック)に変換し、測定したビット数（8）で除算すれば、（最終的に 2 倍すると）実際のデータ送受信タイミング（間隔）を得ることができます。

この時間をタイマ H で計測する場合には、カウント・クロックを周辺機能動作クロックの 1/4 に設定し、得られたカウント値を単純に 1/2 にすれば、タイマ H でのカウント数が得られます。このための具体的なプログラムを紹介します。なお、スタート・ビット検出用に使用する補正用の定数 SOFFSET は、EQU 擬似命令で次のように定義してあります。カッコの中の第 1 項は INTP1 の割り込み応答時間、第 2 項はタイマ H 起動までの時間、第 3 項はスタート・ビット中央検出でのタイマ再起動までの時間、第 4 項は実際にデータをサンプリングするまでの時間（クロック数）です。タイマ H は 1/4 分周したクロックをカウントするので、合計のクロック数を 4 で除算して補正値を算出しています。

```
SOFFSET EQU     (14+34+29+10)/4 ; スタート・ビット検出補正用
```

以上で得られたパルス幅（クロック数）と補正値を使って、実際にタイマ H で時間測定を行うプログラム例は、次のようになります。除算や乗算のような手間のかかる処理は使わず、代わりにシフト処理で済ますようにしています。

```
;
; 19200～9600 なら、1/2 の値がマージンをみて 90～255 の範囲を正常とみなす
;
        MOVW    AX,HL           ; 計測結果を AX レジスタへ
        CLR1    CY
        RORC    A,1             ; 結果を 1/2 に変換
        XCH     A,X
        RORC    A,1
        XCH     A,X
        ADD     A,#0FFH         ; 上位が 0？
        BC      $CALEXIT        ; 遅すぎるなら抜ける
        XCH     A,X
        CMP     A,#90           ; 下限のチェック
        BC      $CALEXIT
CALNEXT:                        ; CY=0
        MOV     CMPDATA1,A      ; CMP01 設定値を保存領域に格納
        DEC     CMPDATA1        ; -1 補正
        RORC    A,1
        SUB     A,#SOFFSET+1    ; スタート・ビット処理補正
        MOV     CMPDATA0,A      ; スタート・ビット処理用に
CALEXIT:
        RET
```

このようにして得られたタイマ H での時間測定に使用する値が変数 CMPDATA1（ビット間隔計測用）と変数 CMPDATA0（スタート・ビット検出用）に格納されます。

[受信割り込み処理]

実際の受信は、スタート・ビットの先頭での、INTP1 のエッジ検出割り込みで起動します。割り込み要求が受け付けられたら、信号がロウ・レベルになっていることを確認します。確認できたら、タイマを起動してスタート・ビットの中央を待ちます（このときは、ここまでの処理時間と、スタート・ビット中央でデータ・サンプリングのタイミング用にタイマ H を再起動するまでを補正（その分短く）した時間となります）。次のプログラムで、コメント欄の頭に記載した数値が命令の処理クロック数で、INTP1 処理の先頭からタイマ起動まで（**赤字の命令**）が、その後の時間計測に影響する部分です。

タイマ H を起動してしまえば、あとはタイマ H が時間測定を行うので、命令の処理時間をあまり気にする必要はありません。その後、スタート・ビットの中央確認タイミング待ちのための処理を行ない、HALT 命令でタイマ H 割り込み(INTTMH1)待ちをします。HALT 命令での割り込み(ここでは、HALT 状態の解除だけでベクタ処理は行わない)待ちは応答時間が速いので、スタート・ビットの中央確認からデータ・サンプリング用のタイマ H 再起動までの時間を短くできます。次のプログラムで緑字の命令が、スタート・ビット中央の検出からデータ・サンプリングのためのタイマ H 再起動までの処理となります。

```
P1INT:  BT      RxDPIN,$P1INTEND        ;10:ノイズなら抜ける
        PUSH    AX                      ; 4:レジスタをセーブ
        CLR1    TMHE1                   ; 6:タイマ H 停止
        MOV     A,CMPDATA0              ; 4:設定値の読み出し
        MOV     CMP01,A                 ; 4:ビット中央までをセット
        SET1    TMHE1                   ; 6:タイマ H を再起動
        CLR1    TMIFH1                  ; タイマ割り込みクリア
        CLR1    TMMKH1                  ; タイマ割り込みマスク解除
        SET1    PMK1                    ; エッジ検出をマスク
        MOV     A,CMPDATA1
        HALT                            ; 3:スタート・ビット中央待ち
        CLR1    TMHE1                   ; 6:タイマ H 停止
        BT      RxDPIN,$NOTSTART        ;10:スタート・ビット未検出なら抜ける
        MOV     CMP01,A                 ; 4:新しい比較値をセット
        SET1    TMHE1                   ; 6:タイマ H 再起動
        CLR1    TMIFH1                  ; タイマ割り込みクリア
        MOV     A,#10000000b            ; 初期データをセット
```

もし、スタート・ビットの中央で検出に失敗したときには、タイマ H を停止して、再度 INTP1 の割り込み待ちに戻ります。

```
NOTSTART:SET1   TMMKH1
         CLR1   PIF1
         CLR1   PMK1
         POP    AX
         RETI
```

スタート・ビットを検出したら、引き続きデータ・ビットの受信処理となります。

[データ受信処理]

データの受信は、タイマ H 割り込み(INTTMH1)を HALT 命令の解除に用いることにより、タイミングを計っています。ここでは、受信のためのビット単位の処理をサブルーチン化し、CALLT 命令で参照できるように、次のように定義してあります。

```
TABLESUB        CSEG    CALLTO
TXSTARTSUB:     DW      TXSTART
TXDATASUB:      DW      TXDATA
RXSTARTSUB:     DW      RXSTART
RXDATASUB:      DW      UARTRX
TXBITSUB:       DW      TXBIT
RXBITSUB:       DW      RXBIT
```

ここで、CSEG CALLT0 の部分が CALLT 命令で使用するサブルーチンのアドレスを格納する領域へ配置することを指定します。あとは、CALLT 命令のオペランドとして使用したいラベルを定義して、DW 擬似命令で対応したサブルーチンのアドレスを記述します。一番下が、1 ビットのデータ受信処理を行うためのサブルーチンのアドレスを定義しています。サブルーチンの本体は、次のようなプログラムです。タイマ H 割り込み（INTTMH1）から 13 クロック以内（BF 命令の途中では、すでにデータを読み込んでいるので、これよりは早くなります）にデータを確認して、キャリーに反映します。同じサブルーチンをストップ・ビット確認でも使用するので、ここの処理では受信結果はキャリーに反映するだけにしています。

```
RXBIT:
        CLR1    CY                      ; キャリーをクリアしておく
        HALT                            ; 3:割り込み待ち
        BF      RxDPIN,$RXDLOW          ;10: データ確認
        SET1    CY
RXDLOW:
        CLR1    TMIFH1
        RET
```

以上で説明したサブルーチンを CALLT 命令（具体的には CALLT [RXBITSUB]）で呼び出し、得られたキャリーを RORC A,1 命令で A レジスタに左から取り込みます（これで LSB ファーストのフォーマットになります）。このときの受信ビット数は、A レジスタに設定されている初期値で決まります（これで、新しいビットの取り込みと、ビットのカウントを同時に実行しています）。

8 ビット分の受信が完了すると、次にストップ・ビットを受信して、結果をフレーミング・エラーに格納します。ストップ・ビットまでの受信が完了すると、タイマ H を停止し、フラグをセットして処理を終了します。

```
RXDATALOOP:
        CALLT   [RXBITSUB]              ; ビット受信
        RORC    A,1                     ; 新しいビットを左から取り込む
        BNC     $RXDATALOOP             ; 8 ビット分繰り返す
        MOV     RXDATA,A                ; 受信データを保存
        CALLT   [RXBITSUB]              ; ストップ・ビット受信
        NOT1    CY                      ; CY:フレーミング・エラー
        MOV     A,#01000000b
        ROLC    A,1                     ; 結果をエラーにセット
        MOV     RXSFLAG,A               ; フラグにセット
        CLR1    TMHE1                   ; タイマを停止
        SET1    TMMKH1                  ; タイマ割り込みをマスク
        MOV     TRXSTAT,#0              ; 受信中フラグをクリア
        POP     AX
P1INTEND:
DUMMY:  RETI
```

[受信起動処理]

受信処理そのものは、これまで説明したように、INTP1 のエッジ検出割り込みで起動されますが、通常は INTP1 のエッジ検出割り込みは禁止状態になっています。調歩同期の受信開始時に割り込みを許可したり、必要なフラグをセットしたりするのが受信起動処理です。このプログラムは、次のようになります。単に受信処理を起動するだけなので、受信完了は RXSFLAG のビット7がセットされたかどうかで確認します。

```
RXSTART:MOV     RXSFLAG,#0              ; エラー・フラグをクリア
        MOV     TRXSTAT,#SRXMODE        ; 受信起動をセット
        CLR1    PIF1                    ; 割り込みをクリア
        CLR1    PMK1                    ; マスク解除
        EI                              ; RxD 信号の立ち下がり待ち
        RET
```

[受信処理サブルーチン]

簡単に使うために、次のような、受信起動から受信完了までを処理するサブルーチンを準備しています。このサブルーチンは、CALLT のテーブルに RXDATASUB の名前で登録してあるので、CALLT [RXDATASUB] と記述すれば使用できます。受信したデータは A レジスタに格納され、X レジスタとゼロ・フラグにエラー・ステータスが入ります。

```
UARTRX: CALLT   [RXSTARTSUB]            ; 受信起動
RXLOOP0:MOVW    AX,RXDATA               ; 受信ステータスを読み出す
        ROL     A,1                     ;
        BNC     $RXLOOP0                ; 完了まで待つ
        AND     A,#00000010b            ; 完了フラグをマスク
        XCH     A,X
        DI
        RET
```

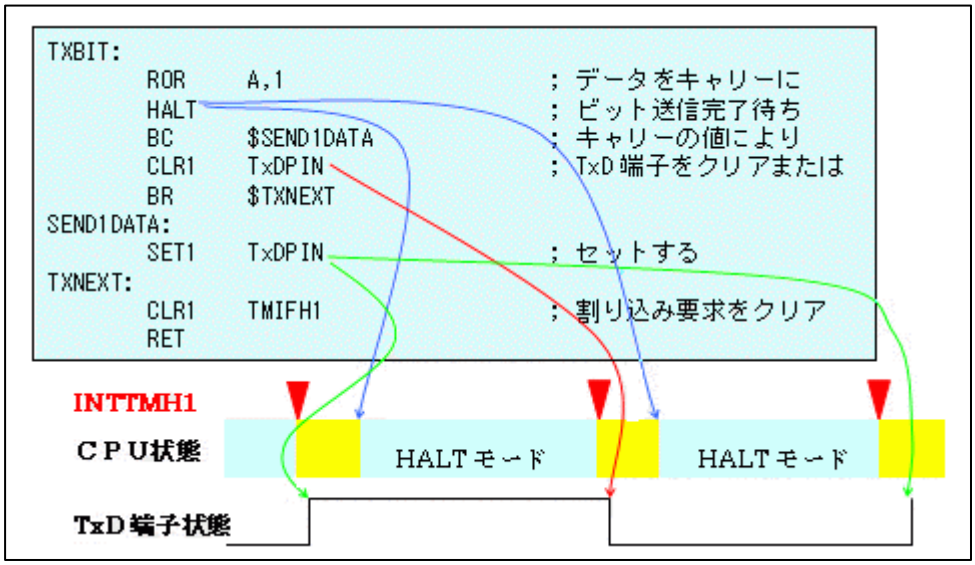
[データ送信処理]

基本的に半二重通信しかできないので、受信処理中（受信処理はスタート・ビットを割り込みで処理しているので、スタート・ビット待ちの状態でもメイン処理ができる）には、処理が終わるまで待ち、受信終了後に送信を行います。

送信が可能なら開始します（TXSTART サブルーチンで必要なパラメータを設定して、タイマを起動し、TxD 端子をロウ・レベルにしてスタート・ビットを送信）。ここまでが、次のプログラムの**赤字**で示した処理です。その後、次のビット送信のために 1 ビット送信サブルーチンを呼び出して、送信中のビットの転送完了をタイマ H 割り込み(INTTMH1)で待ちます。ストップ・ビット送信まで完了すると、タイマ H を停止して処理を完了します。

```
TXDATA: PUSH    PSW
UARTREADYCHK:
        BT      CHKSRXMODE,$UARTREADYCHK ; 受信完了を待つ
        DI                               ; 割り込みを禁止しておく
        CALLT   [TXSTARTSUB]             ; 送信開始
TXDATALOOP:
        CALLT   [TXBITSUB]               ; 1 ビット送信
        DEC     TXBCOUNT                 ; ビット数をカウント
        BNZ     $TXDATALOOP
        MOV     A,#0FFH                  ; ストップ・ビットをセット
        CALLT   [TXBITSUB]
        CALLT   [TXBITSUB]               ; ストップ・ビット完了待ち
        CLR1    TMHE1                    ; タイマ停止
        SET1    TMMKH1                   ; 割り込みマスク
        POP     PSW
        RET
```

ここで呼び出している 1 ビット送信サブルーチンは、次のプログラムに示すように、送信すべきビット（A レジスタの LSB）をキャリーにセットした状態で、前のビットの送信完了を HALT 命令で待ちます。タイマ H 割り込み（INTTMH1）で HALT 状態が解除されると、キャリーをチェックし、ビット操作で TxD 端子にデータ出力します。



[送信起動処理]

送信起動処理は、データ送信処理から呼び出されるサブルーチンで、データ送信で必要な**パラメータ**や**割り込み**設定を行い、タイマ H を起動し、**スタート・ビットの送信を開始**してもどります。

```
TXSTART:
        PUSH    AX                      ; 送信データをセーブ
        MOV     TXBCOUNT,#BLENGTH       ; ビット長を作業用に設定
        MOV     A,CMPDATA1              ; ビット長時間をセット
        MOV     CMP01,A
        SET1    PMK1                    ; 送信時には INTP1 をマスク
        CLR1    TMIFH1
        CLR1    TMMKH1
        SET1    TMHE1                   ; タイマ H 起動
        CLR1    TxDPIN                  ; スタート・ビット送信
        POP     AX                      ; 送信データを戻す
        RET
```

[通信を行うための準備]

以上のプログラムを使用するため、定数を次のように定義します。ここでは、78K0S/KU1+や 78K0S/KY1+を高速内蔵発振クロックで使う場合に合わせています。

```
RxDPIN   EQU     P3.2
RxDMODE  EQU     PM3.2
RxDPUP   EQU     PU3.2
TxDMODE  EQU     PM4.0
TxDPIN   EQU     P4.0
INTP1FALL        EQU     00000000b
BLENGTH  EQU     8               ; ビット長
SOFFSET  EQU     (14+34+29+10)/4 ; スタート・ビット検出補正用
B9600    EQU     207             ; 9600 ボーのタイマ値
HB9600   EQU     104-SOFFSET-1   ; 9600 ボーでの 1/2 ビット時間
```

また、リセットや割り込みのベクタ・テーブルは、次のように定義しています。基本的には、INTP1 だけベクタ割り込みを使用し、INTTMH1 はスタンバイの解除にだけ使用します。

```
RSTVECT CSEG    AT      0
        DW      START           ;00:リセット・スタート処理のベクタ
        DW      DUMMY           ;02:
        DW      DUMMY           ;04:
        DW      DUMMY           ;06:LVI
        DW      DUMMY           ;08:INTP0
        DW      P1INT           ;0A:INTP1
        DW      DUMMY           ;0C:INTTMH1
        DW      DUMMY           ;0E:INTTM000
        DW      DUMMY           ;10:INTTM010
        DW      DUMMY           ;12:INTAD
```

使用する RAM 領域は、次のような定義になっています。

```
                DSEG
                DS      20H
STACKTOP:
                DSEG    saddrp          ; 16 ビット・アクセスのため
RXDATA:         DS      1               ; 受信データ(ステータスとペア)
RXSFLAG:        DS      1               ; 受信ステータス・フラグ
;                  76543210
;                  |      +----: フレーミング・エラー
;                  +-----------: 受信完了
                DSEG    saddr
TRXSTAT:        DS      1               ; 受信中ステータス
;                  76543210
;                  +-----------: 1:受信中
SRXMODE         EQU     10000000b       ; 受信中
CHKSRXMODE      EQU     TRXSTAT.7
TXBCOUNT:       DS      1               ; 送信用ビット・カウンタ
CMPDATA0:       DS      1               ; スタート・ビット確認用
CMPDATA1:       DS      1               ; ビット間隔用設定値
```

端子などは次の例のように初期化します。TxD 端子をハイ・レベルにして出力に設定し、RxD 端子は内蔵プル・アップ抵抗を有効に設定(**青字の部分**)します。また、タイマ H に設定する時間データの初期値を変数に設定(**緑字の部分**)します。

高速内蔵発振器を使用する場合は、キャリブレーションが必要なので、**赤字**のように処理を呼び出しておきます。

```
        MOV     CMPDATA1,#B9600 ; タイマの初期値設定
        MOV     CMPDATA0,#HB9600
        SET1    RxDPUP
        SET1    TxDPIN          ; TxD 信号をハイに設定しておく
        CLR1    TxDMODE         ; TxD 端子を出力に設定
        CALL    !CAL            ; キャリブレーション
        MOV     TMHMD1,#00010000b
```

[使用方法]

①UART 機能を使用するための初期化処理の例です。この例では、タイマ H のパラメータ領域に 8MHz クロックを用いて、9600 ボー用の初期値を設定し、ポート関係の初期化を行っています。その後、キャリブレーション処理を行い、タイマ H のモードを設定しています（ただし、タイマ H はまだ起動しない）。

```
        MOV     CMPDATA1,#B9600 ; タイマの初期値設定
        MOV     CMPDATA0,#HB9600
        SET1    RxDPUP
        SET1    TxDPIN          ; TxD 信号をハイに設定しておく
        CLR1    TxDMODE         ; TxD 端子を出力に設定
        CALL    !CAL            ; キャリブレーション
        MOV     TMHMD1,#00010000b
```

②次に、8 キャラクタのデータを受信するための例を示します。ここでは、HL レジスタをデータ格納用の RAM のポインタとして使用し、B レジスタでデータ数を指定します。

```
                DSEG
;
;  データ・バッファ
;
RXDATABUFF:     DS      8       ; 受信用のデータ・バッファ

                CSEG
        MOVW    HL,#RXDATABUFF
        MOV     B,#8
RXLOOP:
        CALLT   [RXDATASUB]     ; データ受信
        MOV     [HL],A          ; 受信データをバッファへ
        INC     L               ; 次のデータへ
        DBNZ    B,$RXLOOP       ; 8 回繰り返し
```

③同様に、送信は次のようになります。

```
                DSEG
;
;  データ・バッファ
;
TXDATABUFF:     DS      8       ; 送信用のデータ・バッファ

                CSEG
        MOVW    HL,#TXDATABUFF
        MOV     B,#8
TXLOOP:
        MOV     A,[HL]          ; 送信データの読み出し
        CALLT   [TXDATASUB]     ; データ送信
        INC     L               ; 次のデータへ
        DBNZ    B,$TXLOOP       ; 8 回繰り返し
```

ソフトウエアによる UART プログラム例(2) [小ピン・マイコン]

Q1. 汎用のポートだけを用いて、UART を実現するプログラムはありますか。

A1. ここでは、調歩同期通信で、9600～19200 ボーの通信速度、データ長は 8 ビット、パリティなし、ストップ・ビットは 1 ビットで、LSB ファーストの通信に対応するものとします。また、通信は半二重で行うものとします。非同期での受信データに対応するために、スタート・ビットの検出に INTP1 割り込みを使用するものとします。
ソフトウエアによる UART を実現する考え方は、「ソフトウエアによる UART 機能」をご参照ください。
タイマ H も用いる場合の例は、「ソフトウエアによる UART プログラム例 1」をご参照ください。

[プログラムの概要]

このプログラム例では、データ受信の起動だけ割り込みを使用しますが、それ以外は割り込み禁止状態で動作します。処理は大きく分類して、次の部分で構成されています。

①ボー・レート・キャリブレーション部
②INTP1 によるデータ受信処理部
③受信処理起動部
④データ送信処理部

①のボー・レート・キャリブレーション部は、高速内蔵発振クロックを使用する際の補正を行う部分です。外部クロックやセラミック発振子などを用いる場合には必要ありません。
②がスタート・ビットの立ち下がりによる INTP1 で起動される処理で、調歩同期通信の受信処理を行う部分です。基本的に割り込み処理なので、結果はレジスタではなく、RAM に受信したデータと結果のフラグを格納します。半二重通信なので、つねにスタート・ビットを待っているわけではありません。必要なときに、③の処理で起動する必要があります。データの受信を開始すると、以降の INTP1 割り込みを含めて、すべての割り込みは禁止されます。
③では、受信（実際にはスタート・ビット待ち）を起動するために、必要なフラグをセットして INTP1 割り込みを許可します。ここでは、受信処理の起動だけを行うので、スタート・ビットが来るまでは、送信以外の処理を実行可能です。
④はデータ送信処理を行います。送信タイミングは、命令の実行時間をカウントすることで実現しているので、他の割り込みは使用できません。

[サンプリング・タイミングの発生]

このプログラムでは、受信でのサンプリング・タイミングや送信のためのタイミング生成用にソフトウエア・タイマを用いています。そのため、送受信の処理中は、処理時間に影響するような割り込みは、一切受け付けられません。

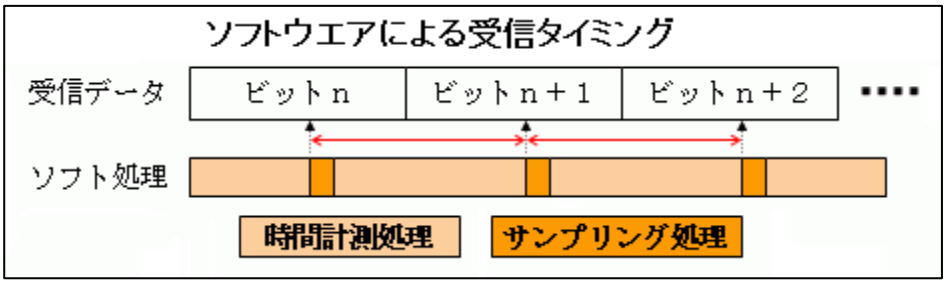
[ボー・レートのキャリブレーション]

高速内蔵発振器からの、誤差が大きいクロックを使用することを考慮して、通信動作の前にキャリブレーションを行います。簡単な方法として、通信相手から 8 ビット分の連続したロウ・レベル（データは 80H）を受信して、その時間を計測します。考え方については、「ソフトウエアによる UART 機能」をご参照ください。
ここでは、INTP1 割り込みで立ち下がりエッジを検出して、割り込み処理に分岐し、そこでキャリブレーション・フラグがセットされていたら、RxD 端子の立ち上がりまでの時間を計測します。INTP1 割り込み受け付けからカウント開始までの時間は、カウントに用いる HL レジスタの初期値を 3 にして補正しておきます。

データ受信もキャリブレーションも、ともに INTP1 割り込みで起動するようにしています。そこで、どちらの処理かを判別できるようにフラグを準備しておきます。このフラグが変数 TRXSTAT のビット 0 で、これがセットされていればキャリブレーションであると定義しています。

```
P1INT:
        BT      RxDPIN,$P1INTEND        ;10:ノイズなら抜ける
        BT      TRXSTAT.0,$CALIBSTART   ;10:キャリブレーション開始
                :
                :
CALIBSTART:
        MOVW    HL,#3                   ;  ここまでの時間補正用
MLOOP:
        NOP                             ;  2:
        INCW    HL                      ;  4:時間計測
        BF      RxDPIN,$MLOOP           ;10:割り込み要求待ち
```

ここで得られた回数をクロック数(1 カウント＝16 クロック)に変換し、測定したビット数(8)で除算すれば、(最終的に 2 倍すると)実際のデータ送受信タイミング(間隔)を得ることができます。この時間間隔は、次図では赤い矢印となります。



この時間間隔でサンプリングするには、サンプリング処理にかかる時間を減算した残りの部分をソフトウエアで計測処理すればいいことになります。実際の時間計測処理は、「変数を減算してゼロになるまでループ」することになるので、次の関係となります。

サンプリング間隔＝サンプリング処理＋計測準備処理＋計測ループ

計測ループはできるだけ短い方がいいのですが、処理(計算)を簡単にするために、「DBNZ saddr」命令を用います。これでループは 8 クロックとなります。そこで、サンプリング処理と計測準備処理の合計が、8 クロックの倍数になるようにしておくと、ループ回数は次の式で求められます。

ループ回数＝サンプリング時間/8 ―(サンプリング処理＋計測準備処理)/8

つまり、前記プログラムで測定した値を 1/4 にして、この式の第 2 項を減算すればいいことになります。この第 2 項は、あとで説明するプログラム例では 6 となります。また、スタート･ビット検出のための 1/2 ビット時間については、定数 SOFFSET として次のように設定しています。

```
SOFFSET EQU     (14+20+12+10)/8 ; スタート・ビット検出補正用
```

以上で得られたパルス幅(クロック数)と補正値を使って、実際にソフト･タイマのループ回数を求める処理プログラム例は、次のようになります。除算や乗算のような手間のかかる処理は使わず、代わりにシフト処理で済ますようにしています。

```
;
; 19200～9600 なら、1/2 の値がマージンをみて 90～255 の範囲を正常とみなす
;
```

```
        MOVW    AX,HL           ; 計測結果を AX レジスタへ
        CLR1    CY
        RORC    A,1             ; 結果を 1/2 に変換
        XCH     A,X
        RORC    A,1
        XCH     A,X
        ADD     A,#0FFH         ; 上位が 0？
        BC      $CALEXIT        ; 遅すぎるなら抜ける
        XCH     A,X
        CMP     A,#90           ; 下限のチェック
        BC      $CALEXIT
CALNEXT:
        RORC    A,1             ; これで 1/4 に変換
        MOV     CMPDATA1,A      ; CMP01 設定値を保存領域に格納
        SUB     CMPDATA1,#6     ; 送受信の 48 クロック分を補正
        RORC    A,1
        SUB     A,#SOFFSET      ; スタート・ビット処理補正
        MOV     CMPDATA0,A      ; スタート・ビット処理用に
CALEXIT:
        MOV     TRXSTAT,#0      ; キャリブレーション済みに
        RETI
```

このようにして得られたソフト・タイマのループ回数が変数 CMPDATA1（ビット間隔計測用）と変数 CMPDATA0（スタート・ビット検出用）に格納されます。8MHz の外部クロックやセラミック発振子を用いて、9600 ボーでの通信を行う際は、CMPDATA1＝98、CMPDATA0＝45 となります。

[受信割り込み処理]

データ受信は、INTP1 のエッジ（スタート・ビットの先頭）検出割り込みで起動します。割り込み要求が受け付けられたら、信号がロウ・レベルになっていることを確認します。その後、**キャリブレーションかどうかを判断**して、受信処理の場合は、スタート・ビットの中央を待ちます。次のプログラムで、コメント欄に記載した数値が命令の処理クロック数です。スタート・ビットの中央が確認できたら、データ受信のための準備をします。

```
P1INT:
        BT      RxDPIN,$P1INTEND        ;10:ノイズなら抜ける
        BT      TRXSTAT.0,$CALIBSTART   ;10:キャリブレーション開始
        PUSH    AX                      ; 4:レジスタをセーブ
        MOV     A,CMPDATA0              ; 4:設定値の読み出し
        MOV     TIMECOUNT,A             ; 4:ビット中央までをセット
STBLOOP:
        DBNZ    TIMECOUNT,$STBLOOP      ; 8:スタート・ビット中央待ち
        BT      RxDPIN,$NOTSTART        ;10:スタート・ビット未検出なら抜ける
;
;  スタート・ビット検出で受信処理をスタート
;
        NOP                             ; 2:タイミング調整用
        MOV     A,#10000000b            ; 6:初期データをセット
```

スタート・ビットの検出ができなかったら、なにもしないで割り込み処理から抜けます。

```
NOTSTART:
        POP     AX
        RETI
```

スタート・ビットを検出したら、引き続きデータ・ビットの受信処理となります。

[データ受信処理]

データの受信は、受信のためのビット単位の処理をサブルーチン化し、それを用いて処理しています。サブルーチン化したのは、メインの処理を見やすくするためと、データにより処理時間が変動するのを避けるためです。ビット受信の具体的なプログラムを次に示します。

```
RXBIT:
        NOP                             ; 2:送信処理との時間調整用
        MOV     TIMECOUNT,A             ; 4:データをセーブ
        MOV     A,CMPDATA1              ; 4:受信時間カウント値をゲット
        XCH     A,TIMECOUNT             ; 6:カウント値セット、データ復帰
BITLOOPR:
        DBNZ    TIMECOUNT,$BITLOOPR     ; 8*n:時間待ち
        BF      RxDPIN,$RXDLOW          ;10: データ確認
        SET1    CY                      ; 2:
        RET                             ; 6:
RXDLOW:
        CLR1    CY                      ; 2:
        RET                             ; 6:
```

ここで、最初の NOP 命令は、受信でのサンプリング処理と送信処理の時間を合わせる(ループ回数を同じにする)ためのものです。**オレンジの部分**が時間待ちのための準備で、**薄いオレンジ部分**が時間調整用のループです。**赤い部分**が実際にデータを確認して、結果をキャリ・フラグに反映している部分です。このようにサブルーチン化して RET 命令で抜けることで、データの 0/1 で処理時間が変わらないようにしています。

このビット受信処理を使い、次のプログラムのように処理して、8 ビットのデータとストップ・ビットを受信します。このプログラムで、**赤い部分**が時間の制限が厳しい部分です。

```
RXDATALOOP:
        CALL    !RXBIT                  ; 6:ビット受信
        RORC    A,1                     ; 2:新しいビットを左から取り込み
        BNC     $RXDATALOOP             ; 6:8 ビット分繰り返し
        CALL    !RXBIT                  ; 6:ストップ・ビット受信
        MOV     RXDATA,A                ; 4:受信データを保存
        NOT1    CY                      ; 2:CY:フレーミング・エラー
        MOV     A,#01000000b
        ROLC    A,1                     ; 結果をエラーにセット
        MOV     RXSFLAG,A               ; フラグにセット
        MOV     TRXSTAT,#0              ; 受信中フラグをクリア
NOTSTART:
        POP     AX
P1INTEND:
DUMMY:
        RETI
```

受信したデータは変数 RXDATA に格納し、受信ステータスを変数 RXSFLAG にセットし、変数 TRXSTAT の受信中フラグをクリアして処理を終了します。

[受信起動処理]

受信処理そのものは、これまで説明したように、INTP1 のエッジ検出割り込みで起動されますが、通常は INTP1 のエッジ検出割り込みは禁止状態になっています。調歩同期の受信開始時に割り込みを許可したり、必要なフラグをセットしたりするのが受信起動処理です。このプログラムは、次のようになります。単に受信処理を起動するだけなので、受信が完了は RXSFLAG のビット7がセットされたかどうかで確認します。

```
RXSTART:MOV     RXSFLAG,#0              ; エラー・フラグをクリア
        MOV     TRXSTAT,#SRXMODE        ; 受信起動をセット
        CLR1    PIF1                    ; 割り込みをクリア
        CLR1    PMK1                    ; マスク解除
        EI                              ; RxD 立ち下がり待ち
        RET
```

[受信処理サブルーチン]

簡単に使うために、次のような、受信起動から受信完了までを処理するサブルーチンを準備しています。

```
UARTRX: CALL    !RXSTART                ; 受信起動
RXLOOP0:
        MOVW    AX,RXDATA               ; 受信ステータスを読み出し
        ROL     A,1                     ;
        BNC     $RXLOOP0                ; 完了まで待つ
        AND     A,#00000010b            ; 完了フラグをマスク
        XCH     A,X
        DI
        RET
```

ここで、作業に使用する変数は、次のように定義されているものとします。特に、RADATA と RXSFLAG は 16 ビットでアクセスできるようにしています。saddrp 属性で RAM に配置しています。

```
        DSEG    saddrp          ; 16 ビット・アクセスのため
RXDATA:
        DS      1               ; 受信データ（ステータスとペア）
RXSFLAG:
        DS      1               ; 受信ステータス・フラグ
;
;   76543210
;   |       +----: フレーミング・エラー
;   +------------: 受信完了
;
        DSEG    saddr
TRXSTAT:
        DS      1               ; 受信中ステータス
;
;  受信動作状態を示すステータス・フラグ
;  76543210
;  x0000000
;  |       +--------: 1:未キャリブレーション
;  |
;  +----------------: 1:受信中
```

なお、このサブルーチンは、次のプログラムで示すように、CALLT 命令のテーブルに RXDATASUB の名前で登録してあります。

```
;
;  サブルーチンを CALLT で使用するためのテーブル領域
;
TABLESUB        CSEG    CALLT0
TXDATASUB:
        DW      TXDATA          ; バイト送信
RXDATASUB:
        DW      UARTRX          ; バイト受信
```

そこで、CALLT [RXDATASUB]と記述すれば、1バイトのコール命令で使用できます。受信したデータは、A レジスタに格納され、X レジスタとゼロ・フラグにエラー・ステータスが入ります。

[データ送信処理]

基本的に半二重通信しかできないので、処理が終わるまでは割り込み禁止にしています。

送信を開始します（TXSTART サブルーチンで処理ビット長を 10 に設定、TxD 端子をロウ・レベルにしてスタート・ビットを送信）。ここまでが、次のプログラムの**赤字**で示した処理です。その後、次のビット送信のために 1 ビット送信サブルーチンを呼び出して、送信中のビットの転送完了をソフトウエア・タイマで待ちます。ストップ・ビットの送信まで完了すると、処理を完了します。

```
TXDATA: PUSH    PSW
        PUSH    BC
        DI                          ; 割り込みを禁止しておく
        CALL    !TXSTART            ; 送信開始
TXDATALOOP:
        CALL    !TXBIT              ; 6:1 ビット送信
        DBNZ    B,$TXDATALOOP       ; 6:出力ビット数をカウント
        POP     BC
        POP     PSW
        RET
```

ここで呼び出している1ビット送信サブルーチンは、次のプログラムに示すように、送信すべきビット（A レジスタの LSB）をキャリーにセットした状態で、前のビットの送信完了をソフトウエア・タイマにより待ちます。指定された時間が経過したら、キャリーをチェックし、TxD 端子をビット操作してデータを出力します。

このプログラムでは、**キャリーをセット**した状態で、キャリーを含めてデータを右回転させると、送信したいビットをキャリーに転送すると同時に、データに 1 を埋めるようになっています。そのため、データ送信後に、このサブルーチンをコールすれば、ストップ・ビットを送信したり、ストップ・ビットの送信完了待ちができます。

```
TXBIT:
        MOV     TIMECOUNT,A         ; 4:データをセーブ
        MOV     A,CMPDATA1          ; 4:送信時間カウント値を取得
        XCH     A,TIMECOUNT         ; 6:データを
        SET1    CY                  ; 2:出力後にデータを 1 に設定
        RORC    A,1                 ; 2:データをキャリーにセット
BITLOOPT:
        DBNZ    TIMECOUNT,$BITLOOPT ; 8*n:時間待ち
        BC      $SEND1DATA          ; 6:キャリーの値により
        CLR1    TxDPIN              ; 6:TxD 信号をクリアまたは
        RET                         ; 6:
SEND1DATA:
        SET1    TxDPIN              ; 6:セットする
TXNEXT:
        RET                         ; 6:
```

[送信起動処理]

送信起動処理は、データ送信処理から呼び出されるサブルーチンで、データ送信で必要な**パラメータ（送信処理を行うスタート・ビットからストップ・ビットまでのビット数）**の設定を行い、**スタート・ビットの送信を開始**してもどります。ビット送信処理と処理時間の整合をとるために、サブルーチン化しています。

```
TXSTART:
        MOV     B,#BLENGTH+2            ; ビット長を作業用に設定
        CLR1    TxDPIN                  ; スタート・ビット送信
        RET                             ; 6:
```

[通信を行うための準備]

以上のプログラムを使用するため、定数を次のように定義します。ここでは、78K0S/KU1+や 78K0S/KY1+を高速内蔵発振クロックで使う場合に合わせています。

```
RxDPIN  EQU     P3.2
RxDMODE EQU     PM3.2
RxDPUP  EQU     PU3.2
TxDMODE EQU     PM4.0
TxDPIN  EQU     P4.0
INTP1FALL       EQU     00000000b
BLENGTH EQU     8               ; ビット長
SOFFSET EQU     (14+20+12+10)/8 ; スタート・ビット検出補正用
B9600   EQU     104-6           ; 9600 ボーのカウント値
HB9600  EQU     52-SOFFSET      ; 9600 ボーでの 1/2 ビット時間
```

また、リセットや割り込みのベクタ・テーブルは、次のように定義しています。基本的には、INTP1 だけベクタ割り込みを使用し、INTTMH1 はスタンバイの解除にだけ使用します。

```
RSTVECT CSEG    AT      0
        DW      START           ;00:リセット・スタート処理のベクタ
        DW      DUMMY           ;02:
        DW      DUMMY           ;04:
        DW      DUMMY           ;06:LVI
        DW      DUMMY           ;08:INTP0
        DW      P1INT           ;0A:INTP1
        DW      DUMMY           ;0C:INTTMH1
        DW      DUMMY           ;0E:INTTM000
        DW      DUMMY           ;10:INTTM010
        DW      DUMMY           ;12:INTAD
```

使用する RAM 領域（タイミング・カウント関係）は、次のような定義になっています。

```
                DSEG    saddr
TXBCOUNT:       DS      1               ; 送信用ビット・カウンタ
CMPDATA0:       DS      1               ; スタート・ビット確認用
CMPDATA1:       DS      1               ; ビット間隔用設定値
```

端子などは次の例のように初期化します。TxD 端子をハイレベルにして出力に設定し、RxD 端子では内蔵プル・アップ抵抗を有効に設定（**青字の部分**）します。また、タイマ H に設定する時間データの初期値を変数に設定（**緑字の部分**）します。

高速内蔵発振を使用するにはキャリブレーションが必要なので、**赤字**のように処理を呼び出しておきます。

```
        MOV     CMPDATA1,#B9600 ; タイマの初期値設定
        MOV     CMPDATA0,#HB9600
        SET1    RxDPUP
        SET1    TxDPIN          ; TxD 信号をハイに設定しておく
        CLR1    TxDMODE         ; TxD 端子を出力に設定
        CALL    !CAL            ; キャリブレーション
```

実際のキャリブレーションは、すでに説明したように、INTP1 割り込みで起動しますが、INTP1 割り込みを有効にするための処理が必要です。そのための処理を行うのがサブルーチン CAL です。具体的なプログラム例を次に示します。処理内容は、変数 TRXSTAT にキャリブレーション・モードを示すフラグをセットして、INTP1 割り込みを許可します。これで、キャリブレーション用のデータ(80H)を待ちます。キャリブレーションが完了すると、変数 TRXSTAT のキャリブレーション・モードを示すフラグがクリアされるので、それを待ちます。

```
CAL:
        MOV     IF0,#0                  ; 割り込み要求をクリア
        MOV     TRXSTAT,#00000001b
        MOV     MK0,#11110111b          ; INTP1 のマスク解除
        EI
CALLOOP:
        NOP
        BT      TRXSTAT.0,$CALLOOP      ; キャリブレーション完了待ち
        RET
```

[使用方法]

①UART 機能を使用するための初期化処理の例です。この例では、パラメータ領域に 8MHz クロックを用いて、9600 ボー用の初期値を設定し、ポート関係の初期化を行っています。その後、キャリブレーション処理を行います。

```
        MOV     CMPDATA1,#B9600 ; タイマの初期値設定
        MOV     CMPDATA0,#HB9600
        SET1    RxDPUP
        SET1    TxDPIN          ; TxD 信号をハイに設定しておく
        CLR1    TxDMODE         ; TxD 端子を出力に設定
        CALL    !CAL            ; キャリブレーション
```

②次に、8 キャラクタのデータを受信するための例を示します。ここでは、HL レジスタをデータ格納用の RAM のポインタとして使用し、B レジスタでデータ数を指定します。

```
                DSEG
;
;  データ・バッファ
;
RXDATABUFF:     DS      8       ; 受信用のデータ・バッファ
                CSEG
        MOVW    HL,#RXDATABUFF
        MOV     B,#8
RXLOOP:
        CALLT   [RXDATASUB]     ; データ受信
        MOV     [HL],A          ; 受信データをバッファへ
        INC     L               ; 次のデータへ
        DBNZ    B,$RXLOOP       ; 8 回繰り返し
```

③同様に、送信は次のようになります。

```
                DSEG
;
;  データ・バッファ
;
TXDATABUFF:     DS      8       ; 送信用のデータ・バッファ
```

```
                CSEG
        MOVW    HL,#TXDATABUFF
        MOV     B,#8
TXLOOP:
        MOV     A,[HL]          ; 送信データの読み出し
        CALLT   [TXDATASUB]     ; データ送信
        INC     L               ; 次のデータへ
        DBNZ    B,$TXLOOP       ; 8 回繰り返し
```

ソフトウエアでの 3 線式シリアルのスレーブ機能検討 [78K0S/Kx1+]

3 線式シリアル通信は、ボード内の IC 同士の通信に広く利用されていますが、78K0S/Kx1+のようにその機能を内蔵していないマイコンもあります。ここでは、ソフトウエアでポートを制御して、3 線式シリアルのスレーブ機能を実現することを検討します。この場合に考慮すべき内容としては、次の 5 点が考えられます。

①SCK の有効エッジに対するデータ取り込みの遅延（ソフトウエア対応特有）

②バイトの切れ目での処理時間によるデータ間隔

③CSI のデータ・クロック位相選択に対する対応

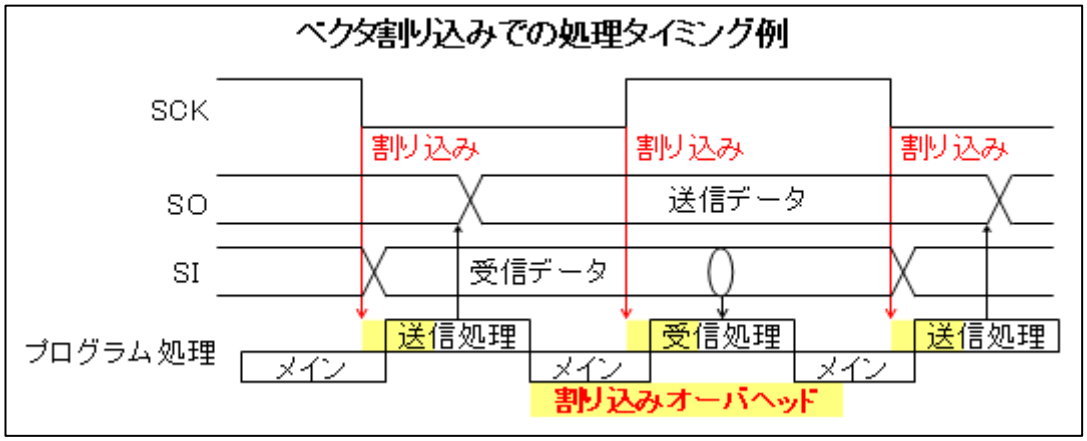
④チップ・セレクト（SPI）への対応

⑤ハンドシェーク信号

1. SCK の有効エッジからの遅延

これが通信速度の上限を決めます。3 線式シリアル通信では、スレーブからマスタに対してウエイトをかけられないため、通信クロック（SCK）を遅くすることで対応する必要があります。以下、単純化のため、SCK の立ち下がりで送信、立ち上がりで受信する（タイプ 1）として説明します。

一番自由度が高いのは、1 ビットごとに SCK の有効エッジを検出して、ベクタ割り込みで処理する方法です。

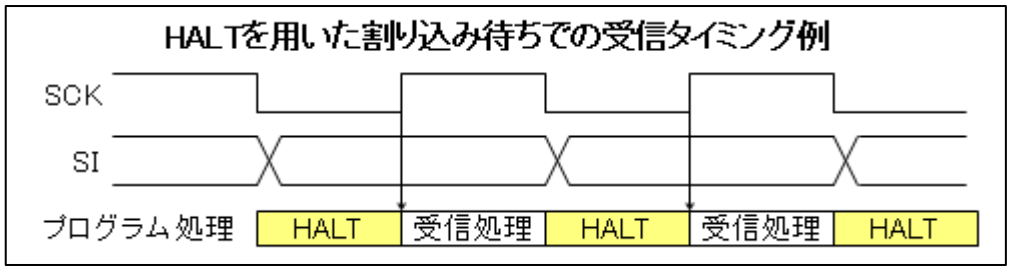


この場合は、各ビットの処理に必ず割り込みのオーバヘッドが付加されるため、通信速度が遅くなってしまいます（割り込みからデータ読み込みまで、78K0S/Kx1+では最大 23 クロック必要です。その後の処理を含めると、1 ビットを処理するのに約 70 クロック必要と考えられます。これは、他の処理方法の約半分の転送速度になります）。

通信速度をできるだけ高くするには、通信を開始（たとえば、チップ・セレクトを割り込みで検出するなどして確認）したら、その後はベクタ割り込みではなく**ポーリングで処理**するか、HALT モードに設定して、**割り込みで HALT モードを解除**することが考えられます。さらに、**送受信でなく、送信だけ／受信だけ**の制限が必要になります。

(1) 受信処理の検討

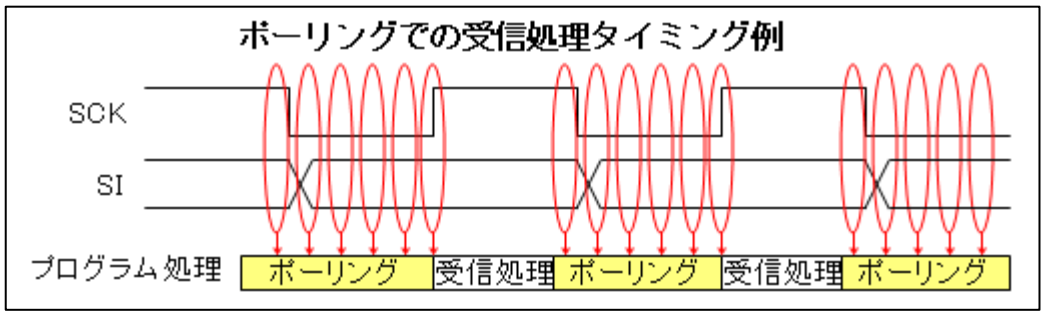
HALT モードを用いた受信を考えると、1 ビットに対して、次の例のような処理になります。そうすると、1 ビット当たりの単純な処理時間は 28 クロックとなり、高速内蔵発振クロックを使用した 8MHz での動作で考えると 3.5μs となるので、約 280kbps の通信速度となります。



このときのプログラムとして、SI 入力からビット 7 を取り込む例を次に示します。

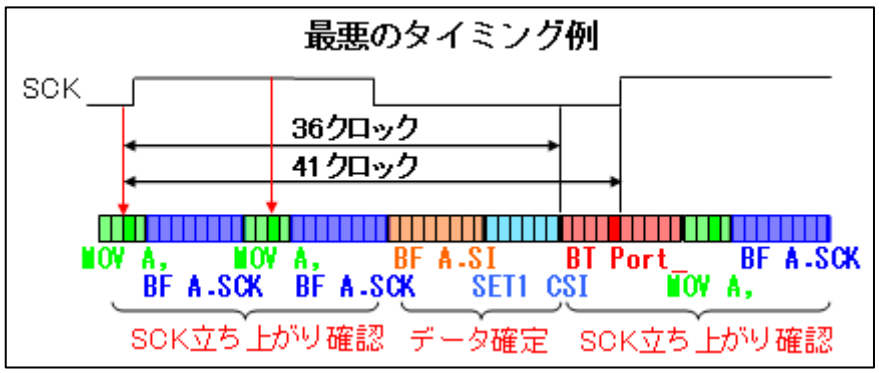
```
    HALT                                ;2+4:SCK のエッジ検出待ち
    BF      Port_SI,$Datais0            ; 10:データ確認
    SET1    CSI.7                       ;  6:ビット 7 受信
Datais0:
    CLR1    PIF0                        ;  6:割り込みフラグ・クリア
```

これをポーリング処理すると、SI 信号と SCK 信号を同じポートに割り振れば、次のプログラム例が考えられます。

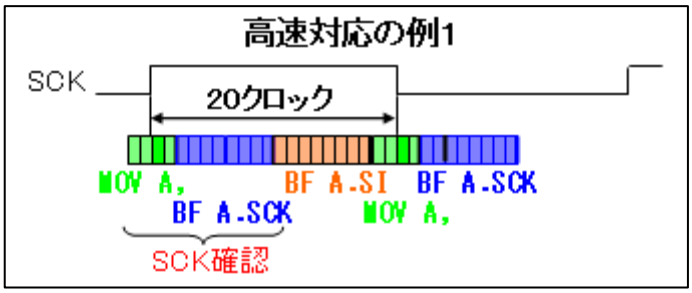



```
LOOP:
    MOV     A,Port_S                    ; 4:端子データの読み込み
    BF      A.SCKbit,$LOOP              ; 8:SCK の立ち上がり待ち
    BF      A.SIbit,$Datais0            ; 8:SI データの確認
    SET1    CSI.7                       ; 6:ビット 7 受信
Datais0:
    BT      Port_SCK,$$                 ;10:SCK がロウを確認
```

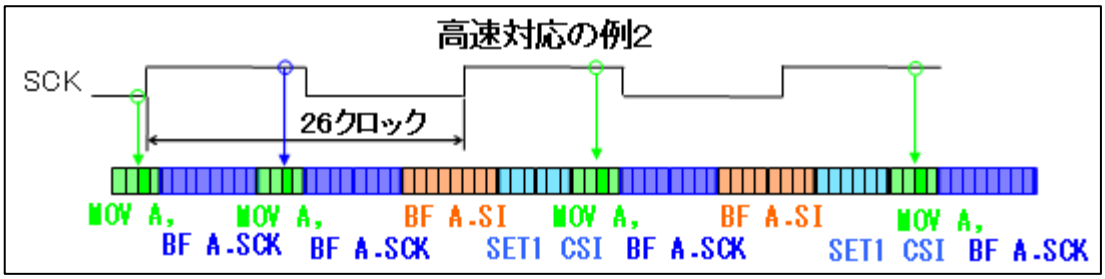
この例では、受信データの反映(**Datais0**:の前)までの処理は、単純計算で 26 クロックとなり、約 300kbps の通信速度です。しかし、SCK の立ち上がりとその確認の同期関係がないため、最悪 2 回ループすると考えると 36 クロック必要です。その後、SCK がロウ・レベルであることを確認すると、1 ビットあたり 41 クロック必要になります。これは 200kbps 弱の転送速度になります。これが安定して転送可能な値です(次の最悪タイミング例参照)。



より高速の転送が要求される場合は、転送速度に下限を設け、SCK のロウ・レベル状態の確認を省略して、ある程度は対応できます。SCK のハイ・レベル幅が 20 クロック未満（転送速度が約 160kbps 以上）の条件があれば、SCK の立ち上がり直後にハイ・レベルを確認しても、次のビットでの確認までに、SCK は確実に立ち下がっているので、SCK がロウ・レベルであることの確認が省略できます。



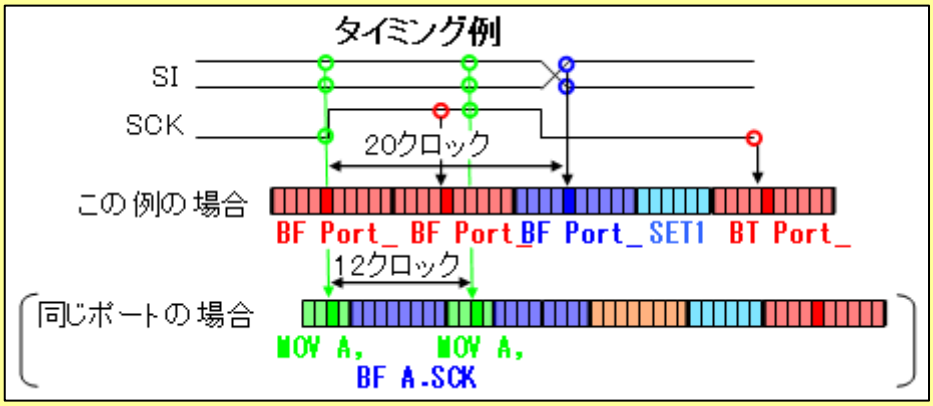
この場合の動作タイミング例を次に示します。この例のように、最初の SCK のチェックでハイ・レベルが確認できなくても、**2 回目のチェック**（12 クロック後）で確認できます。そこから 26 クロックで、**次のビットの SCK のチェック**（SI データの取り込み）をしますが、その後は 1 回で済むので、転送速度は 300kbps 程度まで対応可能になります。



このように、ポーリングの場合には条件をつけることで、より高速の転送が可能になります。

【コーヒー・ブレーク】
次のように記述しても同様の処理ができます。このプログラムなら、1 ビットのデータを同じクロック数で処理でき、しかも信号を同じポートに配置する必要はありません。ただし、SCK の立ち上がり確認と SI の確認（読み込み）の間が、必ず 1 命令分（10 クロック）空きます。SCK の立ち上がりのタイミングと確認のタイミングが非同期のため、SCK 立ち上がりから最悪 10 クロック遅れで確認され、さらに 10 クロック遅れて SI が読まれることになります。そうなると、SCK のハイ・レベル幅（実際は SCK の立ち下がりに対するデータの保持時間）が 20 クロック必要になり、SCK のデューティが 50%なら、1 ビットの処理に最悪 40 クロックかかる場合が考えられます。この場合は、条件をつけても高速化はできません。

```
LOOP:
    BF      Port_SCK,$LOOP          ;10:SCK の立ち上がり待ち
    BF      Port_SI,$Datais0        ;10:SI データの確認
    SET1    CSI.7                   ; 6:ビット 7 受信
Datais0:
    BT      Port_SCK,$$
```

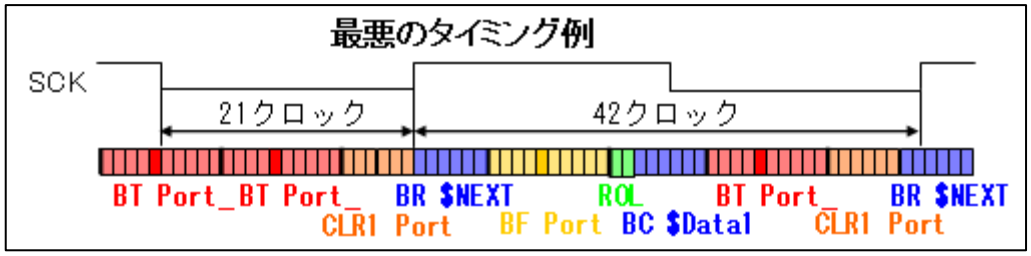


(2) 送信処理の検討

次に送信について考えます。この場合、次のような処理が考えられます。

```
    ROL     A,1                     ; 2:出力データをキャリーへ
    BC      $Data1                  ; 6:データが 1 なら分岐
    BT      Port_SCK,$$             ;10:SCK 立ち下がり待ち
    CLR1    Port_SO                 ; 6:送信
    BR      $NEXTbit                ; 6:次のビットへ
Data1:
    BT      Port_SCK,$Data1         ;10:SCK 立ち下がり待ち
    SET1    Port_SO                 ; 6:送信
NEXTbit:
    BF      Port_SCK,$$             ;10:SCK 立ち上がり待ち
```

ここで、単純に時間を計算すると、最悪の場合は 1 ビットあたり 42 クロック必要になります。データのセット・アップ時間を考慮すると、約 180kbps（44 クロック）の転送速度となります。



以上の考察より、**送信／受信で対応できる最大の転送速度は 180kbps 程度**と考えられます。

なお、125kbps 以上の制限をつけると、250kps 程度までは対応できそうです。

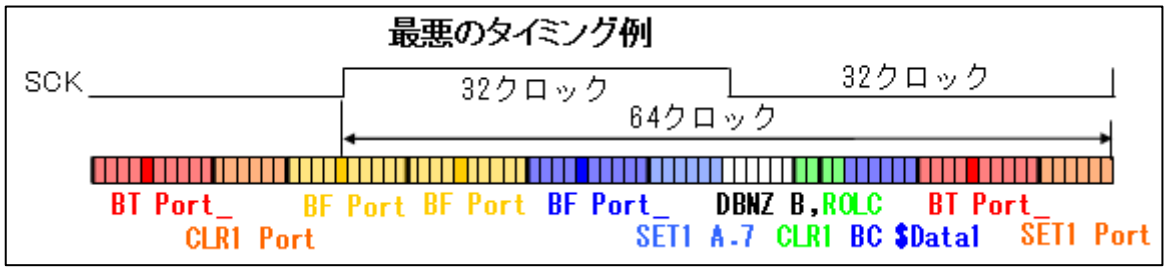
(3) 送受信処理の検討

送受信を行う必要性は低いと考えられますが、一応検討だけしておきます。これまでの検討結果より、送信だけで 1 ビットあたり約 44 クロックなので、受信処理はコーヒー・ブレークで紹介した方法でも可能です（これにより、A レジスタは送信専用に使用可能）。この場合、1 バイトの処理に必要なプログラムは、次の例のようになります。ここでは、プログラムの大きさを考慮して、8 回ループすることで 1 バイトのデータを処理しています。

```
RTXBYTE:
    MOV     B,#8                    ;ビット数をセット
LOOP:
    CLR1    CY                      ; 2:次のビット用にクリア
    ROLC    A,1                     ; 2:出力データをキャリーへ
    BC      $Data1                  ; 6:データが 1 なら分岐
    BT      Port_SCK,$$             ;10:SCK 立ち下がり待ち
    CLR1    Port_SO                 ; 6:送信
    BF      Port_SCK,$$             ;10: SCK 立ち上がり待ち
    BF      Port_SI,$RDataisO       ;10:SI データの確認
    SET1    A.0                     ; 6:ビット 7 から順に受信
    DBNZ    B,$LOOP                 ; 6:8 ビット分繰り返し
    BR      $RTXEND
Data1:
    BT      Port_SCK,$Data1         ;10:SCK 立ち下がり待ち
    SET1    Port_SO                 ; 6:送信
    BF      Port_SCK,$$             ;10:SCK 立ち上がり待ち
    BF      Port_SI,$RDataisO       ;10:SI データの確認
    SET1    A.0                     ; 6:ビット 7 から順に受信
```

```
RDatais0:
    DBNZ    B,$LOOP                 ; 6:8 ビット分繰り返し
RTXEND:
```

この例では、必ず立ち下がりと立ち上がりのエッジを待って処理を行います。そのため、受信から次の送信までの処理が最悪 64 クロック必要（次のタイミング例参照）になり、これが全体のボトルネックになります。命令の並びを変更して、CLR1 CY と ROLC A,1 を送信の直後に移動させれば、SCK の立ち上がり検出から次のビット送信までの時間を 4 クロック短くできます。



これだけでは効果が少ないので、実行命令数を削減する手段を考えます。これまでのプログラム例は、SCK の立ち上がりを確認してデータを取り込んでいました。送信では SCK の立ち下がり検出後にデータを出力しましたが、その直後にデータを取り込んでしまいます。**マスタからの送信は SCK の立ち下がりからせいぜい 500ns** で完了します。したがって、SCK の立ち下がり検出後の**データ出力の直後にデータを取り込んでも、正しいデータが読める**ことになります。これを反映すると、次の例のようになります。

```
RTXBYTE:
    MOV     B,#8                    ;ビット数をセット
LOOP:
    CLR1    CY                      ; 2:次のビット用にクリア
    ROLC    A,1                     ; 2:出力データをキャリーへ
    BC      $Data1                  ; 6:データが 1 なら分岐
    BT      Port_SCK,$$             ;10:SCK 立ち下がり待ち
    CLR1    Port_SO                 ; 6:送信
    BF      Port_SI,$RDatais0       ;10:SI データの確認
    SET1    A.0                     ; 6:ビット 7 から順に受信
    BF      Port_SCK,$$             ;10:SCK 立ち上がり待ち
    DBNZ    B,$LOOP                 ; 6:8 ビット分繰り返し
    BR      $RTXEND
Data1:
    BT      Port_SCK,$Data1         ;10:SCK 立ち下がり待ち
    SET1    Port_SO                 ; 6:送信
    BF      Port_SI,$RDatais0       ;10:SI データの確認
    SET1    A.0                     ; 6:ビット 7 受信
RDatais0:
    BF      Port_SCK,$$             ;10:SCK 立ち上がり待ち
    DBNZ    B,$LOOP                 ; 6:8 ビット分繰り返し
RTXEND:
```

この変更により、SCK の立ち上がり待ちで不要なループをしないようにします。これで 10 クロック分は処理を短くできます。これ以上短縮するには、**転送速度の下限を設けて、SCK の立ち上がり待ちそのものをなくす**必要があります（SCK の立ち上がり待ちは、次のビットの SCK の立ち下がりを検出するための処理です。転送速度の下限を設ければ必要なくなります）。そうすると、**約 170kbps** までは対応できる可能性があります。

2. バイト切れ目での処理

マスタが連続転送機能を内蔵している場合、データ転送の間隔を指定できます。スレーブは、必要な間隔を明確にするだけで済みます。そのような機能を内蔵していない場合、転送完了時に次の転送を起動することになり、切れ目のない転送はできません。このため、1 転送クロック分は間隔が空くと考えられます。
送受信データは上位ソフトの処理なので、ここではハンドシェークで対応します(「**5. ハンドシェーク信号**」参照)。

3. CSI のデータ・クロック位相

ひとつのプログラムで、複数の通信タイプに対応するのは不適切で、いずれかに絞り込む必要があります(たとえば従来の SIO と互換の**タイプ 1**)。
タイプ 1 とタイプ 3 は、SCK の論理が反転しているだけなので、タイプ 3 への変更は比較的容易です。
タイプ 2 とタイプ 4 は、データ受信のタイミングがタイプ 3 やタイプ 1 と同じなので、問題はないと考えられますが、タイプ 2 とタイプ 4 の送信では注意が必要です。

4. チップ・セレクト(SPI)への対応

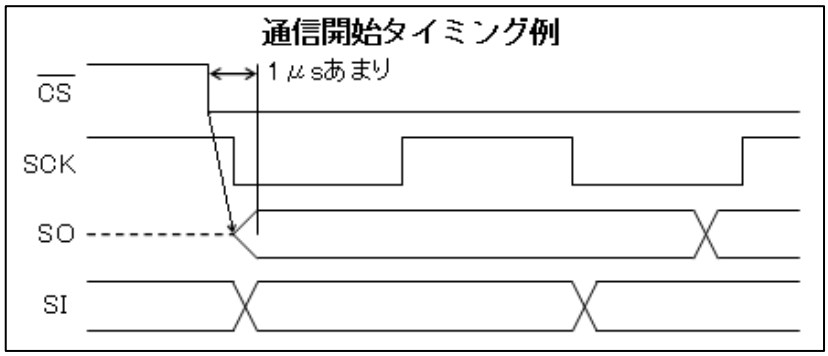
SPI のスレーブでは、チップ・セレクトの制御が必要で、これがアクティブになってから通信を開始するまでの時間に制限があります。その検出方法で変動しますが、ベクタ割り込みを用いて、そこで準備をすべて行うプログラム例を次に示します。この例では、**通信開始までに約 10μs 必要**です。**SPI** のスレーブでは、この時間に注意が必要です(ただし、ハンドシェーク信号を利用すれば回避できます)。

```
__INTP1:
            PUSH    AX                  ; 4:使用レジスタをセーブ
            PUSH    HL                  ; 4:
            SET1    P4.5                ; 6:SO を 1 に
            CLR1    PM4.5               ; 6:SO を出力に
            BF      TXFLAG,$RXDATA      ;10:フラグが 0 なら受信処理へ
            MOVW    AX,TXPOINTER        ; 6:送信ポインタ読み出し
            MOVW    HL,AX               ; 4:
TXLOOP:
            MOV     A,[HL]              ; 6:送信データ読み出し
            INCW    HL                  ; 4:ポインタ更新
            MOV     TXCOUNT,#8-1        ; 6:ビット長を設定
;
;   送信準備
;
LOOP0:
            ROL     A,1                 ; 2:MSB をキャリーへ
```

この時間を短くするには、事前に通信(特に送信のデータ)準備を行い、ベクタ割り込みではなく、ポーリングなどでチップ・セレクトの立ち下がりを待つ必要があります。通信の手順やタイミング、通信タイプが事前に決まっている場合は、そのような制限をつければ、**2～3μs 程度**に短縮できます。
たとえば、SO に出力する送信データの最初のビットを事前に出力ラッチに設定し、チップ・セレクトの立ち下がりを検出したら、すぐポートを出力に設定します。すると、通信準備(実際は最初のビットの送信)まで、割り込みオーバヘッド＋6 クロックで処理できます。HALT と組み合わせると、さらに応答時間を **1μs あまり**まで短くできます。

```
;
; 事前に送信データの最初のビットを P4.5 にしておく
;
__INTP1:
            CLR1    PM4.5               ; 6:SO を 1 に
```

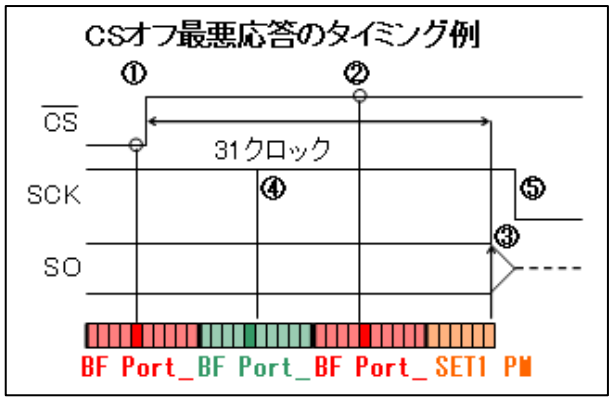
```
        PUSH    AX                      ; 4:使用レジスタをセーブ
        PUSH    HL                      ; 4:
```



このように、チップ・セレクトからの動作時間（出力）を通信動作に影響を与えないレベルまで短くすることは可能ですが、チップ・セレクトが切れてから SO をハイ・インピーダンスにするまでの時間は、チップ・セレクトと次のデータ転送 SCK の確認が必要なので、**3～4μs 程度**になります。この部分のプログラム例を次に示します。

```
LOOP1:
        BF      P4.0,$TXSTART0          ;10:SCK の立ち下がりで送信へ
        BF      P4.3,$LOOP1             ;10:CS がロウなら継続
;
;   送信処理完了
;
TXEXIT:
        SET1    PM4.5                   ;6:SO を Hi-Z に
```

この処理での最悪のタイミングを次に示します。チップ・セレクトの立ち上がり確認（①でロウ、②でハイ）のループ（20 クロック）が遅延量を決定します。そのあと、ポートを入力に変更（③）します。④以降なら、SCK は無視されますが、③までは SO が出力状態なので、⑤以降でないと変化させられません。



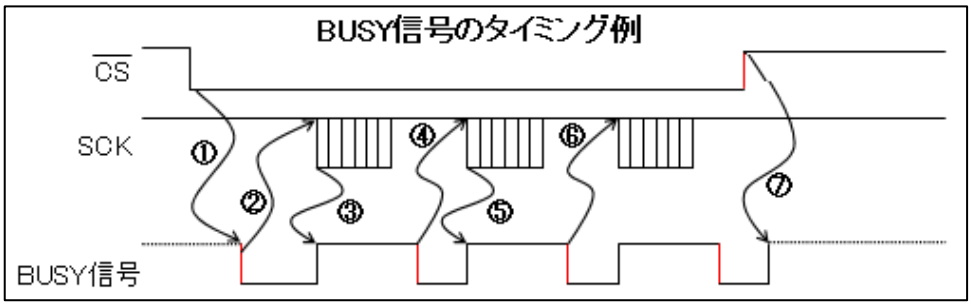
通常、SPI のマスタは、チップ・セレクト端子としてポートを利用すると思われるので、こういったタイミングをあまり厳密に検討する必要はないと考えられます。

5. ハンドシェーク信号

3 線式シリアルでは、マスタが通信を開始（SCK を出力）する前に、スレーブが動作可能になっている必要があります（動作可能でないと、マスタとスレーブでビットがずれてしまいます（「ハンドシェーク[共通的な内容]」参照）。マスタがスレーブの状態を確認するには、スレーブが状態を示すハンドシェーク信号を出力する必要があります。SPI での複数のスレーブ接続を考慮すると、オープン・ドレーンの BUSY 信号を転送許可に使用することになります。チップ・セレクト信号で選択されたスレーブが通信準備を完了したら、BUSY 信号をロウ・レベルにし、マスタに通信準備ができたことを知らせます。このときのタイミング例を示します。

① マスタがチップ・セレクト信号でスレーブを選択します。スレーブがそれを認識するまで、BUSY 信号はハイ・インピーダンス状態です。
② スレーブが選択されたことを認識して通信可能になると、BUSY 信号を立ち下げます。マスタは、BUSY 信号が立ち下がったことを確認したら、通信を開始します。
③ マスタからの通信がスタートすると、スレーブは BUSY 信号を立ち上げます。
④ スレーブは、通信されたデータを処理して、次の通信準備が完了すると BUSY 信号を立ち下げます。
⑤ ③④の動作を繰り返します。
⑥ チップ・セレクト信号が切れると、スレーブは BUSY 信号を立ち上げて、ハイ・インピーダンス状態にして通信を終了します。

このハンドシェーク信号が利用できれば、転送速度を除いて、基本的にタイミングの問題は緩和されます。



注意 BUSY 信号は、③や⑤のタイミングで、通信完了よりはるかに早いタイミングで立ち上がりますが、マスタの制御によっては、通信開始前に次の通信準備をすることも考えられます。この場合、BUSY 信号の立ち下がりエッジを確認してから、次の通信を行う必要があります。

ソフトウエアでの半二重 3 線式シリアル・スレーブ機能の実現(1) [78K0S/Kx1+]

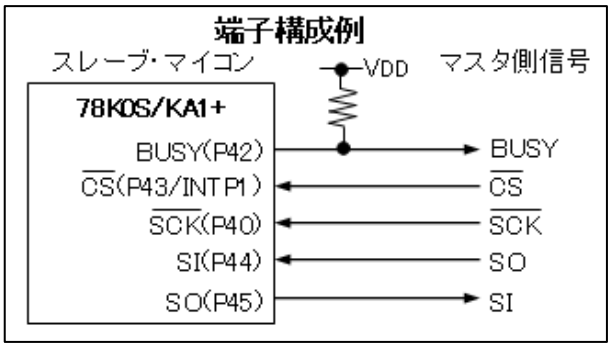
「ソフトウェアでの 3 線式シリアルのスレーブ機能検討」の結果から、次の条件の 3 線式シリアルのスレーブ機能をソフトウエアによるポート制御で実現してみます。

- スレーブとして送信動作か受信動作（半二重通信）だけ
- CS はエッジ検出割り込みを使用（データ転送まで約 10μs 必要）
- CS に対して、BUSY 信号をサポート
- 通信タイプ 1（SCK 立ち下がりで送信、立ち上がりで受信）
- 転送速度は 180kbps 程度までに対応

[仕様検討]

1. 接　続

ここでは、次の端子構成例のように、2 本のハンドシェーク信号を含めて 5 本のポートを使用するものとします。



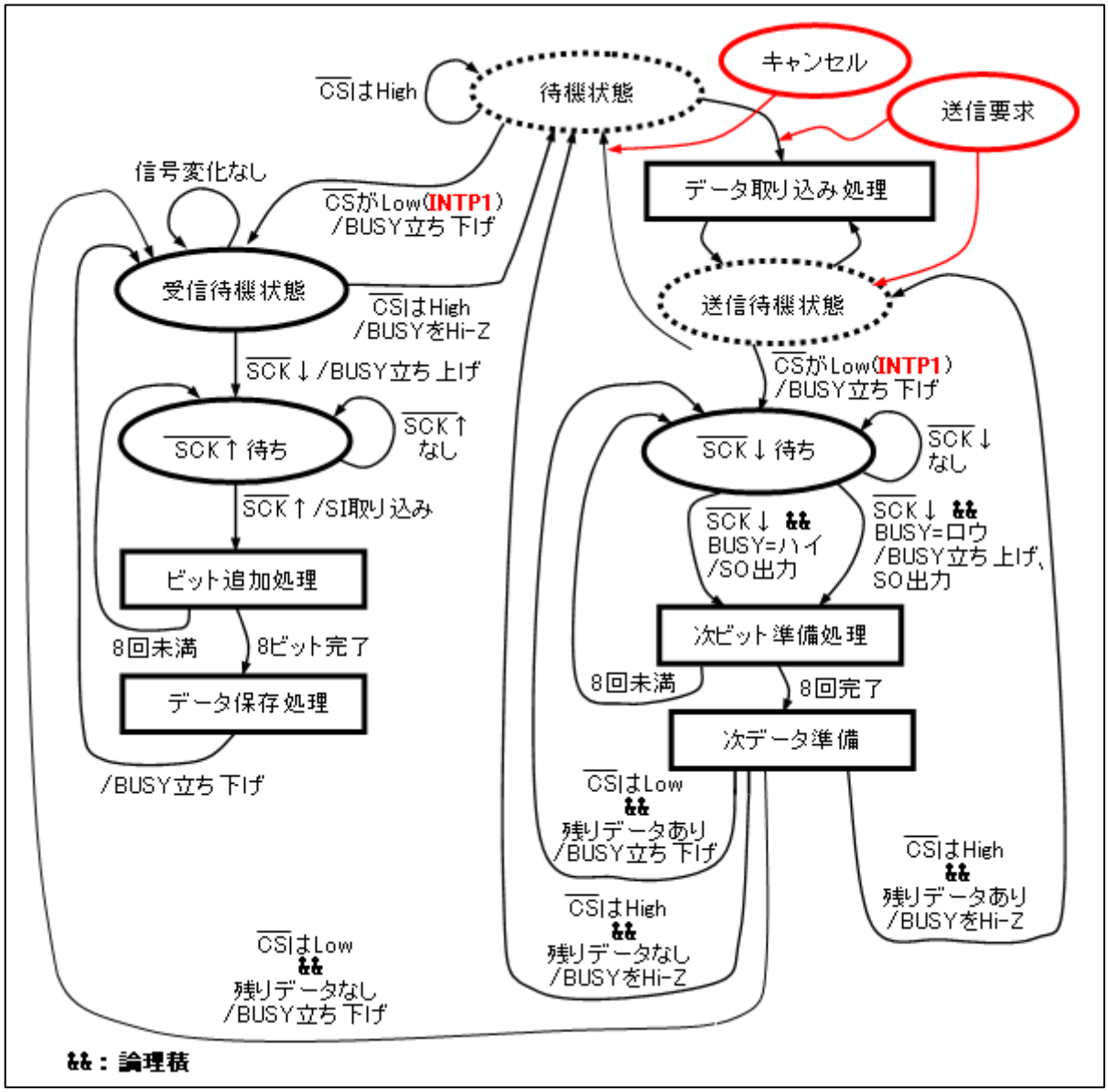
2. 動作概要

SPI マスタからの $\overline{CS}$ 信号の立ち下がりをエッジ割り込みで検出し、ベクタ割り込みで通信を起動します。通信の方向は、事前にフラグに設定しておきます。受信の場合は、ベクタ割り込み処理で、$\overline{SCK}$ 信号の立ち上がりで SI 端子の状態を読み込みます。8 ビットのデータを読み込んで、その結果をバッファに保存します。そのとき、$\overline{CS}$ 信号を確認して、ハイ・レベルなら処理を終了します。ロウ・レベルのままなら、受信処理を継続します。送信の場合は、送信バッファの内容を $\overline{SCK}$ 信号の立ち下がりで SO 端子から出力します。

なお、データはMSBファーストで転送します。また、バッファは16バイトの容量を確保しておき、先頭から使用するだけとし、リング・バッファ構造にはしません。

3. 全体の処理

ここで考えている、通信処理全体の概略状態遷移図を次に示します。



初期状態では、受信モードで待機します（待機状態）。その状態で $\overline{CS}$ 信号が立ち下がると、ベクタ割り込みを受け付けてスレーブ受信となり、BUSY 信号を立ち下げて、$\overline{SCK}$ 信号の立ち下がりを待ち、BUSY 信号を立ち上げます。その後、$\overline{SCK}$ 信号の立ち上がりに同期して、SI 信号を読み込みます。8 ビット分の受信が完了すると、データをバッファに保存して、BUSY 信号を立ち下げて次の通信準備ができたことをマスタに通知し、次のデータか通信完了（$\overline{CS}$ 信号の立ち上がり）を待ちます。

待機状態(プログラムは何もしない)で、メイン処理から送信要求(送信したいデータをバッファに書き込む)があると、送信待機状態となります(これも単にバッファに送信データがあるだけで、プログラム処理はしない)。この状態で $\overline{CS}$ 信号が立ち下がると、割り込み要求を受け付けてスレーブ送信となり、BUSY 信号を立ち下げます。$\overline{SCK}$ 信号の立ち下がりに同期して出力データを SO 端子に出力し、BUSY 信号を立ち上げます。1 バイトの送信が完了したら、次のデータを準備します。このあとは、4 つの条件に分かれます。

①次のデータが残っている状態で、$\overline{CS}$ 信号ロウ・レベルのままなら、次データ送信のための BUSY 信号を立ち下げて、通信準備ができたことをマスタに通知し、$\overline{SCK}$ 信号の立ち下がり待ちとなります。

②次のデータがあるのに、$\overline{CS}$ 信号がハイ・レベルになった場合は、BUSY 信号を Hi-Z にしてから割り込み処理を抜け、送信待機状態になります。

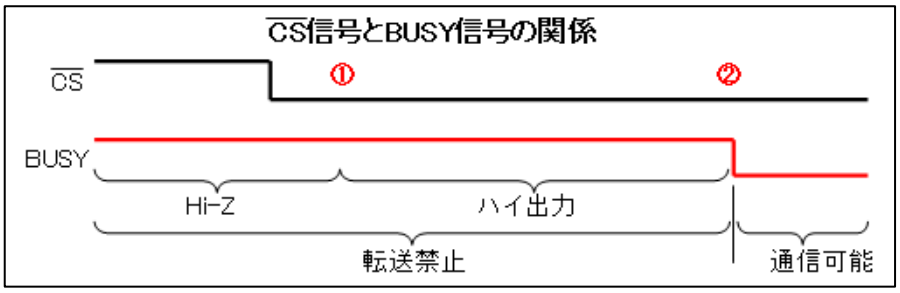
③次のデータがなくなって、$\overline{CS}$ 信号がハイ・レベルになった場合は、通信完了として、BUSY 信号を Hi-Z にしてから割り込み処理を抜けて、待機状態に戻ります。

④ 次のデータがなくなって、$\overline{CS}$ 信号がロウ・レベルのままなら、BUSY 信号を立ち下げて、受信待機状態に移行します。

[制御プログラム検討]

1. 通信の起動($\overline{CS}$ 信号の立ち下がり検出)

通信開始のタイミングは、$\overline{CS}$ 信号の立ち下がりをエッジ検出したベクタ割り込みで確認することにします。このとき、送信データがあるかどうかのフラグで、受信処理か送信処理かを判断します。送信の場合は、バッファからデータを読み出し、送信の準備を行います。$\overline{CS}$ 信号の立ち下がりから、実際にデータを送受信(特に送信)できるまでは「58 クロック＋割り込み応答時間(最大 19 クロック)」で、8MHz 動作とすると 10μs が必要です。ただし、$\overline{CS}$ 信号の立ち下がりから「6 クロック＋割り込み応答時間(最大 19 クロック)」で BUSY 信号が出力されますし、それ以前の BUSY 信号は Hi-Z(ハイ・レベル)となっているので、BUSY 信号を利用すれば、この部分の応答速度をカバーできます。



次に、この起動部分のプログラム例を示します。BUSY(P42 の出力ラッチ)には初期状態で 1 がセットされているので、1 行目で PM4 を設定した段階(①)でハイ・レベルが出力されます。通信可能(次の例では送信可能)になると(②)、BUSY 信号を立ち下げて、通信準備ができたことをマスタに知らせます。なお、データ送信のための $\overline{SCK}$ 信号確認までは、さらに 13 クロックかかります。最初のビットの処理を 2 ビット目以降と分けると、この時間を 5 クロックまで短縮できますが、マスタでの BUSY 信号の立ち下がり確認から $\overline{SCK}$ 出力までの時間を考えると、あまり効果的ではありません。

```
__INTP1:
            MOV     PM4,# 11011011b         ; 6:①So(P45) と BUSY(P42) を出力に
            PUSH    AX                      ; 4:使用レジスタをセーブ
            PUSH    HL                      ; 4:
            BF      TxFlag,$RxData          ;10:フラグが 0 なら受信処理へ
            MOVW    AX,TxPointer            ; 6:送信ポインタ読み出し
            MOVW    HL,AX                   ; 4:
```

```
TxLoop:
            MOV     A,[HL]                  ; 6:送信データ読み出し
            MOV     TxBitCount,#8           ; 6:ビット長を設定
            CLR     P4.2                    ; 6:②BUSY を立ち下げ
; 送信準備完了
```

2. 送信処理

送信待機状態で $\overline{CS}$ 信号の立ち下がりを検出したら、送信処理に移ります。送信処理がスタートすると、$\overline{SCK}$ 信号の立ち下がりを検出して、データを SO 端子に出力します。このとき、同時に BUSY 信号を立ち上げて、送信処理を開始したことを示します(③)。この処理では、SO 信号を出力する **MOV 命令**で、BUSY 信号の立ち上げも同時に行って、処理時間の増加をなくしています。その部分を含めた処理は、次のようになります。ここでは、$\overline{SCK}$ 信号の立ち下がりからの遅延をできるだけ小さくするため、送信データが 0 か 1 かを事前に判定しています。また、8 ビットのデータ送信が完了するまでは、$\overline{CS}$ 信号の確認をしません。

```
TxLoop2:
            ROL     A,1                     ; 2:MSB をキャリーへ
            BC      $Data1                  ; 6:送信データ判定
;
;   出力データが 0 の場合の処理
;
TxLoop3:
            BT      P4.0,$TxLoop3           ;10:SCK の立ち下がり待ち
            MOV     P4,#00000100b           ; 6:③0 を出力、BUSY 立ち上げ
            BR      $NexrBit                ; 6:次のビット処理へ
;
;    出力データが 1 の場合の処理
;
Data1:
TxLoop4:
            BT      P4.0,$TxLoop4           ;10:SCK の立ち下がり待ち
            MOV     P4,#00100100b           ; 6:③1 を出力、BUSY 立ち上げ
NextBit:
;
;   1 ビットの送信完了
;
TxLoop5:
            BF      P4.0,$TxLoop5           ;10:SCK の立ち上がり待ち
            DBNZ    TxBitCount,$TxLoop2     ; 8:8 ビット分繰り返す
            INCW    HL                      ; 4:ポインタ更新
            BT      P4.3,$TxExit            ;10:CS がハイなら抜ける
            DBNZ    TxCount,$TxLoop         ; 8:残データがあればループ
```

変数 **TxBitCount** をループ･カウンタとし、**TxLoop2** までを 8 回ループすれば、1 バイトのデータ送信が完了します。次にデータ･ポインタを更新してから $\overline{CS}$ 信号の確認を行い、$\overline{CS}$ 信号がロウ･レベルのままなら、次のデータ処理に移ります。

このプログラムでは、通信速度をできるだけ高速化するために、**$\overline{CS}$ 信号は 8 ビット目の SCK の立ち上がりでしかチェックしていません。**そのため**マスタは、最後のデータの転送が始まったら(8 ビット目の転送までに)、$\overline{CS}$ 信号を切る必要があります。このタイミングが不可能なら、1 ビット目と 2 ビット目以降の転送で、処理を変える必要があります。**このプログラム例は、あとで示します。

3. 送信完了処理

$\overline{CS}$ 信号が切れる(ハイ・レベルになる)と、以降の通信は行わなくなります。また、送信するデータがなくなると、送信完了となります。次の 3 つの場合、送信を継続しません。

①送信データはなくなったが、$\overline{CS}$ 信号はロウ・レベルのまま　→　受信処理へ

②送信データがなくなり、さらに $\overline{CS}$ 信号も切れた　→　通信完了

③送信データはあるが、$\overline{CS}$ 信号が切れた　→　通信中断

(1) 送信完了して受信へ移行

①の場合の処理プログラムは、次のようになります。この処理の直前で、変数 TxCount のカウントにより、送信が完了していることを確認できます。そのため、送信関係のフラグやポインタを初期化してから、$\overline{CS}$ 信号を確認します。この結果、ロウ・レベルのままなら、**受信処理に分岐します**。ハイ・レベル(②)なら、割り込み処理から抜けますが、$\overline{CS}$ 信号のタイミングでは、ここで割り込みから抜けることはほとんど考えられません。

```
;
;   データ送信は完了
;
            CLR1    TxFlag                  ; 6:送信フラグをクリア
            MOVW    AX,#TxBuff              ; 6:ポインタを初期化
            MOVW    SetPointer,AX           ; 8:セット・ポインタ初期化
            MOVW    TxPointer,AX            ; 8:送信ポインタ初期化
            BF      P4.3,$RxData            ;10:送信完了で CS がロウなら受信
            MOV     PM4,#0FFH               ; 6:ポートを Hi-Z に
            POP     HL
            POP     AX
            CLR1    PIF1
            RETI
```

(2) 全データ送信完了で通信完了

②の場合の処理プログラムは、次のようになります。この処理は、$\overline{CS}$ 信号がハイ・レベルなっていることを確認して開始します。最初に、SO 信号と BUSY 信号出力を Hi-Z にしています。$\overline{SCK}$ 信号の立ち上がりを確認してから、SO 信号が Hi-Z になるまでの時間は約 33 クロックです。マスタの受信データ保持時間がそれ以上必要なら、1 行目の **MOV** 命令の前に NOP 命令などを追加して、実行時間を延ばしてください。2 行目の **DBNZ** 命令で、残りデータ数を確認し、0 なら②として処理を完了して、割り込みから復帰します。

```
;
;   CS 立ち上がりでの処理中断
;
TxExit:
            MOV     PM4,#0FFH               ; 6:ポートを Hi-Z に
            DBNZ    TxCount,$TxExit2        ; 6:残りデータあれば分岐
;
;   CS が off し、データ送信も完了
;
            MOVW    AX,#TxBuff
            MOVW    SetPointer,AX           ;セット・ポインタ初期化
            MOVW    TxPointer,AX            ;送信ポインタ初期化
            CLR1    TxFlag                  ; 6:送信フラグクリア
            POP     HL                      ; 6:レジスタを復帰
            POP     AX                      ; 6:レジスタ復帰
            CLR1    PIF1                    ; 6:割り込み要求クリア
            RETI
```

(3) 残りデータありの状態で通信終了

③の場合の処理プログラムは、次のようになります。ここでは、次の送信のためのポインタを保存し、割り込みから復帰します。

```
;
;   残りデータがあるのに CS が off したので、次の準備を行う
;
TxExit2:
            MOVW    AX,HL                   ; 4:ポインタを保存
            MOVW    TxPointer,AX            ; 8:メモリに保存
            POP     HL                      ; 6:レジスタを復帰
            POP     AX                      ; 6:レジスタ復帰
            CLR1    PIF1                    ; 6:割り込み要求クリア
            RETI                            ; 6:処理完了
```

4. 送信開始でも $\overline{CS}$ 信号を確認する処理例

これまで説明した処理では、バイトの送信完了時だけ $\overline{CS}$ 信号を確認していましたが、さらに、バイトの送信開始待ちでも確認するプログラム例を次に示します。「**1. 通信の起動**」の **TxLoop** から **TxLoop2** の前までの部分が、次のようになります。**赤**と**青**の部分が変更点です。基本的に、最初のビット(ビット 7)の $\overline{SCK}$ 信号の立ち下がり待ちが追加になり、残りの 7 ビット分だけループで処理します。

```
;
;    1 バイトの送信処理
;
TxLoop:
            MOV     A,[HL]                  ; 6:送信データ読み出し
            MOV     TxBitCount,#7           ; 6:ビット長-1 を設定
            CLR     P4.2                    ; 6:BUSY を立ち下げる
;
;    最初のビット処理（BUSY のチェックを含む）
;
            ROL     A,1                     ; 2:ビット 7 をキャリーへ
            BC      $B7Data1                ; 6:データが 1 なら分岐
TxB7Loop:
            BT      P4.3,$TxExit            ;10:CS がハイなら終了
            BT      P4.0,$TxB7Loop          ;10:SCK 立ち下がり待ち
            MOV     P4,#00000100b           ; 6:送信&BUSY セット
            BR      $TxBit6                 ; 6:残り 7 ビット処理へ
B7Data1:
            BT      P4.3,$TxExit            ;10:CS がハイなら終了
            BT      SCKPort,$B7Data1        ;10:SCK 立下り待ち
            MOV     SIOPort,#00100100b      ; 6:送信&BUSY セット
TxBit6:
            BF      P4.0,$TxBit6            ;10:SCK 立ち上がり待ち
```

5. 受信処理

送信するデータがない状態で、$\overline{CS}$ 信号がロウ・レベルになったら受信処理を行います。

(1) マクロの定義

ここでは、ビット毎の受信処理用にマクロを定義することで、プログラムの記述を簡素化(sin1 6 のように記述するだけ)します。このマクロは受信データを変数 SIO のオペランドで指定されたビットにセットするものです。サブルーチンとは異なり、マクロを参照したところにはマクロの内容そのものが展開されます。つまり、ビット毎の受信処理が並びます。なお、最初のビットについては $\overline{CS}$ 信号の確認を行うためにマクロは使用しません。

```
;*************************************************************************
;                                                                        *
;    ビット受信処理用マクロ                                              *
;                                                                        *
;*************************************************************************
sin1        macro   bitno
    LOCAL   Datais0,Loop
    BT      P4.0,$$                     ;SCK 立ち下がり待ち
Loop:
    MOV     A,P4                        ;SCK と SI 読み出し
    BF      A.0t,$Loop                  ;SCK がロウならループ
    BF      A.4,$Datais0                ;SI を確認
    SET1    SIO.bitno
Datais0:
    endm
```

(2) 受信起動処理と完了処理

受信準備が完了したら、BUSY 信号を立ち下げて、$\overline{SCK}$ 信号の立ち下がりを待ってループします。$\overline{SCK}$ 信号が立ち下がったなら、実際の受信処理に移行します。SCK 信号の立ち下がりの前に $\overline{CS}$ 信号が立ち上がったら、受信完了となり、受信データを格納するためのポインタを更新して、割り込みから復帰します。

```
RxData:
            MOVW    AX,RxSetPointer         ; 6:受信用ポインタ読み出し
            MOVW    HL,AX                   ; 4:HL にポインタを設定
RxLoop:
            MOV     SIO,#0                  ; 6:受信初期値を 00 に
            CLR1    P4.2                    ; 6:BUSY を立ち下げ
RxLoop2:
            MOV     A,P4                    ; 4:ポートのリード
            BF      A.0,$RxBit7             ;10:SCK がロウなら受信へ
            BF      A.3,$RxLoop2            ;10:CS がロウなら継続
;
;   CS の立ち上がりで、受信処理完了へ
;
RxExit:
            SET1    P4.2                    ; 6:BUSY を立ち上げ
RxExit2:
            MOV     PM4,0FFH                ; 6:ポートの出力禁止
            MOVW    AX,HL                   ; 4:ポインタを作業領域に
            MOVW    RxSetPointer,AX         ; 6:コピーする
            CLR1    PIF1
            POP     HL
            POP     AX
            RETI
```

(3) 受信処理本体

SCK 信号が立ち下がって通信がスタートしたら、BUSY 信号を立ち上げます。次に、SCK 信号の立ち上がりを待って、読み込んだ SI 信号を変数 SIO のビット 7 に取り込みます。ビット 6 以降はマクロで処理して、変数 SIO で得た受信データをバッファに格納し、受信データ数を＋1 して、次のデータ受信に移ります。

```
;
;    SCK 立ち上がりを待って、データの読み出し開始
;
RxBit7:
            SET1    P4.2                ;BUSY を立ち上げる
RxLoop3:
            MOV     A,P4
            BF      A.0,$RxLoop3        ;SCK の立ち上がり待ち
            BF      A.4,$RxDatais0
            SET1    SIO.7               ;ビット 7 をセット
RxDatais0:
            sin1    6                   ;ビット 6 を受信
            sin1    5                   ;ビット 5 を受信
            sin1    4                   ;ビット 4 を受信
            sin1    3                   ;ビット 3 を受信
            sin1    2                   ;ビット 2 を受信
            sin1    1                   ;ビット 1 を受信
            sin1    0                   ;ビット 0 を受信
            MOV     A,SIO               ;受信データ読み出し
            MOV     [HL],A              ;バッファに格納
            INCW    HL
            INC     RxCount
            BT      CSPort,$RxExit3
            BR      RxLoop
RxExit3:
            BR      RxExit2
```

[周辺プログラム検討]

以上で説明した制御プログラムを使用するための、サポート・プログラムを紹介します。ここで触れるのは、次のような処理です。なお、プログラムはサブルーチン化して、C 言語記述のプログラムからも関数呼び出しできるようにしてあります。

①初期化処理
②送信ステータス確認
③受信ステータス確認
④1 バイトのデータを送信バッファに追加
⑤送信バッファ設定
⑥受信バッファから 1 バイトのデータ読み出し
⑦受信バッファ設定

1. 初期化処理

制御プログラムを使用するために、ポインタ類の初期化や受信データ・バッファのクリア、割り込みの設定を行います。この処理は、バッファを制御するためのポインタや変数を設定するため、割り込み禁止で処理させます。通常は、起動後に 1 回使用します。

```
;*******************************************************************************
;                                                                              *
;    送信／受信（バッファ）の初期化                                            *
;                                                                              *
;   送信／受信で使用するバッファを内部で定義したバッファに設定し、受信   *
;   バッファはデータを0クリアしておきます                                      *
;   また、CS 信号の検出で使用する INTP1 に立ち下がりエッジ検出を有効       *
;   に設定し、割り込みマスクの解除までの処理を行います                         *
;   入力データ          : なし                                                 *
;   戻り値              : なし                                                 *
;   使用レジスタ        : なし                                                 *
;                                                                              *
;    extern void _init_buff(void);                                             *
;                                                                              *
;*******************************************************************************
; PUBLIC    __init_buff,INIT_BUFF
__init_buff:
INIT_BUFF:
    PUSH    PSW                     ;IE フラグをセーブ
    DI                              ;以下の処理は割り込み禁止で行う
    PUSH    HL
    PUSH    AX

    MOVW    AX,#TxBuff
    MOVW    TxPointer,AX            ;送信データ読み出しポインタ初期化
    MOVW    SetPointer,AX           ;送信データ設定ポインタ初期化
    MOV     TxCount,#0              ;データ数を初期化
    CLR1    TxFlag                  ;送信フラグをクリア

    MOVW    AX,#RxBuff              ;受信バッファの初期化初期化
    MOVW    RxPointer,AX            ;受信データ読み出しポインタ初期化
    MOVW    RxSetPointer,AX         ;受信データ格納用ポインタ初期化
    MOV     RxCount,#16             ;データ数を初期化
    MOVW    HL,AX
    MOV     A,#0
Init_Loop:
    MOV     [HL],A                  ;受信バッファをクリア
    INCW    HL
    DBNZ    RxCount,$Init_Loop
    POP     AX
    POP     HL
    MOV     P4,#00100100b           ;SO と BUSY 出力をハイにする
    MOV     INTM1,#00000000b        ;INTP1 を立ち下がりエッジ検出に
    CLR1    PIF1
    CLR1    PMK1
    POP     PSW
    RET
```

2. 送信ステータス確認

送信データ･バッファの未送信のデータ数を C レジスタに格納します。また CY フラグで、バッファが空かどうかを確認できます。通常は、送信データ･バッファにデータをセットしたあと、送信完了かどうかの確認に使用します。

```
;*******************************************************************************
;                                                                              *
;    送信データ・バッファのステータス確認                                      *
;   入力データ          : なし                                                 *
;   戻り値              : C レジスタ(空なら CY=0)                              *
;   使用レジスタ        : なし                                                 *
;   バッファの残りデータ数を戻します                                          *
;    extern unsigned char _check_txbuff(void);                                 *
;                                                                              *
;*******************************************************************************
PUBLIC      __check_txbuff,CHECK_TXBUFF
__check_txbuff:
CHECK_TXBUFF:
            PUSH    AX
            MOV     A,TxCount               ;バッファ中のデータ数読み出し
            MOV     C,A                     ;戻り値にセット
            SUB     A,#1                    ;空なら CY をセット
            NOT1    CY                      ;CY を反転
            POP     AX
            RET
```

3. 受信ステータス確認

受信データ･バッファの中のデータ数をCレジスタに格納します。データがあれば、**1バイト読み出し**などでバッファから読み出してください。

```
;*******************************************************************************
;                                                                              *
;    受信データ・バッファのステータス確認                                      *
;   入力データ          : なし                                                 *
;   戻り値              : C レジスタ、CY フラグ(バッファ空)                    *
;   使用レジスタ        : なし                                                 *
;   バッファ中のデータ数を戻します。データが存在するときにデータを読み        *
;   出してください                                                            *
;    extern unsigned char _check_rxbuff(void);                                 *
;                                                                              *
;*******************************************************************************
PUBLIC      __check_rxbuff,CHECK_RXBUFF
__check_rxbuff:
CHECK_RXBUFF:
            PUSH    AX
            MOV     A,RxCount               ;バッファ中のデータ数読み出し
            MOV     C,A                     ;戻り値にセット
            SUB     A,#1                    ;データ数０なら CY をセット
            POP     AX
            RET
```

4. 1 バイトのデータを送信バッファに追加

X レジスタにセットされたデータを送信データ･バッファに追加します。なお、バッファの大きさは管理していないので、送信バッファのサイズ以上にデータを追加しないようにする必要があります。送信バッファ内のデータ数は、**送信ステータス確認**で判断してください。

```
;*******************************************************************************
;                                                                              *
;    送信データ１バイトをバッファに追加                                        *
;   入力データ          : X レジスタ                                           *
;   戻り値              : なし                                                 *
;   使用レジスタ        : なし                                                 *
;   バッファのデータがすべて送信されると、TxFlag が 0 になります              *
;    extern void _add_data(unsigned char);                                     *
;                                                                              *
;*******************************************************************************
PUBLIC      __add_data,ADD_DATA
__add_data:
ADD_DATA:
            PUSH    HL
            PUSH    AX
            MOVW    HL,AX                   ;データをセーブ
            MOVW    AX,SetPointer           ;セット・ポインタ読み出し
            XCHW    AX,HL
            MOV     A,X
            MOV     [HL],A                  ;データをバッファにセット
            INCW    HL                      ;ポインタ更新
            XCHW    AX,HL                   ;H:データ、AX:ポインタ
            MOVW    SetPointer,AX           ;ポインタ保存
            PUSH    PSW                     ;PSW のセーブ
            DI                              ;割り込み禁止
            INC     TxCount                 ;データ数を+1
            SET1    TxFlag                  ;送信要求フラグをセット
            MOV     P4,#00100100b           ;SO と BUSY 出力をハイにする
            POP     PSW
            POP     AX
            POP     HL
            RET
```

5. 送信バッファ設定

複数バイトのデータを送信する場合、1 **バイトのデータ送信**を必要な回数使用しても実現できますが、バッファを切り替えたほうが便利です。ここでは、AX レジスタに送信データが格納されたバッファのアドレスを、またスタックにデータ数を設定して、サブルーチン･コールします。このパラメータの渡し方は、CC78K0S の引数の受け渡しと合わせるためです。アセンブラで使用するだけなら、データ数もレジスタで渡す方が、処理が簡単になります。
送信が完了したかどうかは、**送信ステータス確認**で判別できます。

```
;*******************************************************************************
;                                                                              *
;    送信データＮバイトをバッファにセット                                      *
;   入力データ          : AX レジスタ（バッファの先頭アドレス）    : 第一引数 *
;                       : スタック（データ数）                     : 第二引数 *
;   戻り値              : なし                                                 *
;   使用レジスタ        : なし                                                 *
```

```
;   バッファのデータがすべて送信されると、TxFlag が 0 になります          *
;                                                                        *
;    extern void _set_data(unsigned int,unsigned char);                  *
;                                                                        *
;************************************************************************
PUBLIC      __set_data,SET_DATA
__set_data:
SET_DATA:
            PUSH    PSW                 ;PSW をセーブ
            DI                          ;割り込み禁止
            PUSH    HL                  ;使用レジスタをセーブ
            PUSH    AX
            MOVW    TxPointer,AX        ;送信ポインタを変更
            MOVW    SetPointer,AX       ;送信セット・ポインタを変更
            MOVW    AX,SP               ;スタック・ポインタ読み出し
            MOVW    HL,AX
            MOV     A,[HL+7]            ;データ数読み出し
;
;   ここでのスタックの内容は、以下のとおり
;    HL(SP)  ---> X
;    HL+1    ---> A
;    HL+2    ---> L
;    HL+3    ---> H
;    HL+4    ---> PSW
;    HL+5    ---> 戻り番地(下位)
;    HL+6    ---> 戻り番地(上位)
;    HL+7    ---> データ数
;    HL+8    ---> ダミー・データ
;
            MOV     TxCount,A           ;データ数を設定
            CLR1    TxFlag              ;フラグをクリアしておく
            AND     A,A                 ;データ数チェック
            BZ      $Set_Data_End       ;データ数 0 ならポインタ設定だけ
            SET1    TxFlag              ;送信要求フラグをセット
            POP     HL
            PUSH    HL
            ADD     A,L
            MOV     X,A
            MOV     A,H
            ADDC    A,#0
            MOVW    SetPointer,AX       ;送信セット・ポインタを変更
Set_Data_End:
            MOV     P4,#00100100b       ;SO と BUSY 出力をハイにする
            POP     AX                  ;レジスタの復帰
            POP     HL
            POP     PSW
            RET
```

6. 受信バッファから 1 バイトのデータ読み出し

受信データ・バッファから、もっとも古いデータを読み出し、C レジスタに格納します。もし、受信データ・バッファにデータがない場合は、C レジスタに 0 が格納され、CY フラグがセットされます。読み出してデータがなくなったら、ポインタを初期値に戻します。

```
;*******************************************************************************
;                                                                              *
;    受信データ 1 バイトをバッファから読み出す                                 *
;   入力データ          : なし                                                 *
;   戻り値              : C レジスタ(データなければ CY=1)                      *
;   使用レジスタ        : なし                                                 *
;   バッファから最も古いデータを読み出し、残りデータ数を-1 にします           *
;    extern unsigned char _get_data(void);                                     *
;                                                                              *
;*******************************************************************************
PUBLIC      __get_data,GET_DATA
__get_data:
GET_DATA:
            PUSH    AX
            MOV     A,RxCount               ;データ数を読み出し
            CMP     A,#1                    ;データ数チェック
            MOV     C,A                     ;ダミーの戻り値をセット
            BC      $GetEnd                 ;データなければ戻る
            PUSH    HL
            MOVW    AX,RxPointer            ;リード・ポインタ読み出し
            MOVW    HL,AX                   ;HL レジスタにセット
            INCW    AX                      ;ポインタ更新
            MOVW    RxPointer,AX            ;ポインタ保存
            MOV     A,[HL]                  ;データ読み出し
            MOV     C,A                     ;戻り値をセット
            POP     HL
            DEC     RxCount                 ;データ数を-1
            BNZ     $GetEnd                 ;残りデータありなら分岐
            PUSH    PSW
            DI                              ;以下の処理は割り込み禁止
            MOVW    AX,#RxBuff              ;受信バッファの初期化
            MOVW    RxPointer,AX            ;データ読み出しポインタ初期化
            MOVW    RxSetPointer,AX         ;受信データ格納用ポインタ初期化
            POP     PSW
GetEnd:
            POP     AX
            RET
```

7. 受信バッファ設定

受信バッファをパラメータとして、渡されたアドレスに変更します。

```
;*******************************************************************************
;                                                                              *
;    受信バッファの設定                                                        *
;   入力データ          : AX レジスタ(受信用バッファの先頭アドレス)            *
;   戻り値              : なし                                                 *
;   受信データを格納するバッファを指定します                                  *
;   また、受信データ数を 0 に設定します                                       *
;   受信完了確認は受信データ・バッファのステータスで行います                  *
;    extern void _set_rxbuff(unsigned int);                                    *
;                                                                              *
;*******************************************************************************
PUBLIC      __set_rxbuff,SET_RXBUFF
__set_rxbuff:
SET_RXBUFF:
    PUSH    PSW
    DI
    MOVW    RxPointer,AX                ;リード・ポインタをセット
    MOVW    RxSetPointer,AX             ;セット・ポインタをセット
    MOV     RxCount,#0                  ;データ数をクリア
    POP     PSW
    RET
```

8. データ領域の定義

内部で使用している変数やポインタ、バッファは、次のように定義してあります。これらを直接操作することはありませんが、作業用にこれらの領域を確保しています。そのため、メモリの使用量には注意が必要です。

```
;
;    データ・ポインタ領域
;
            DSEG    saddrp
TxPointer:
            DS      2       ;送信用データ読み出しポインタ
SetPointer:
            DS      2       ;送信データ設定用ポインタ
RxPointer:
            DS      2       ;受信データ読み出しポインタ
RxSetPointer:
            DS      2       ;受信データ格納用ポインタ
            DSEG    saddr
;
;    データ・カウンタ領域
;
TxCount:
            DS      1       ;送信データ数カウンタ
RxCount:
            DS      1       ;受信データ数カウンタ
BitCount:
            DS      1       ;ビット数カウント用
;
;    データ領域
;
TxBuff:
            DS      16      ;送信データ・バッファ領域
RxBuff:
            DS      16      ;受信データ・バッファ領域
SIO:
            DS      1       ;受信作業用
;
;    フラグ領域
;
            BSEG
TxFlag              DBIT    ;送信要求フラグ
```

[プログラムの使用方法]

前述の周辺プログラムを介して通信を行います。このとき、サブルーチン群は PUBLIC 宣言されているので、次のように EXTRN 宣言すればリンクできます。

```
EXTRN               INIT_BUFF                   ;バッファ初期化
EXTRN               CHECK_TXBUFF                ;送信バッファ確認
EXTRN               CHECK_RXBUFF                ;受信バッファ確認
EXTRN               ADD_DATA                    ;送信データをバッファに
EXTRN               SET_DATA                    ;送信バッファを設定
EXTRN               GET_DATA                    ;受信データ読み出し
EXTRN               SET_RXBUFF                  ;受信バッファ設定
```

1.1 バイト送信の使用例

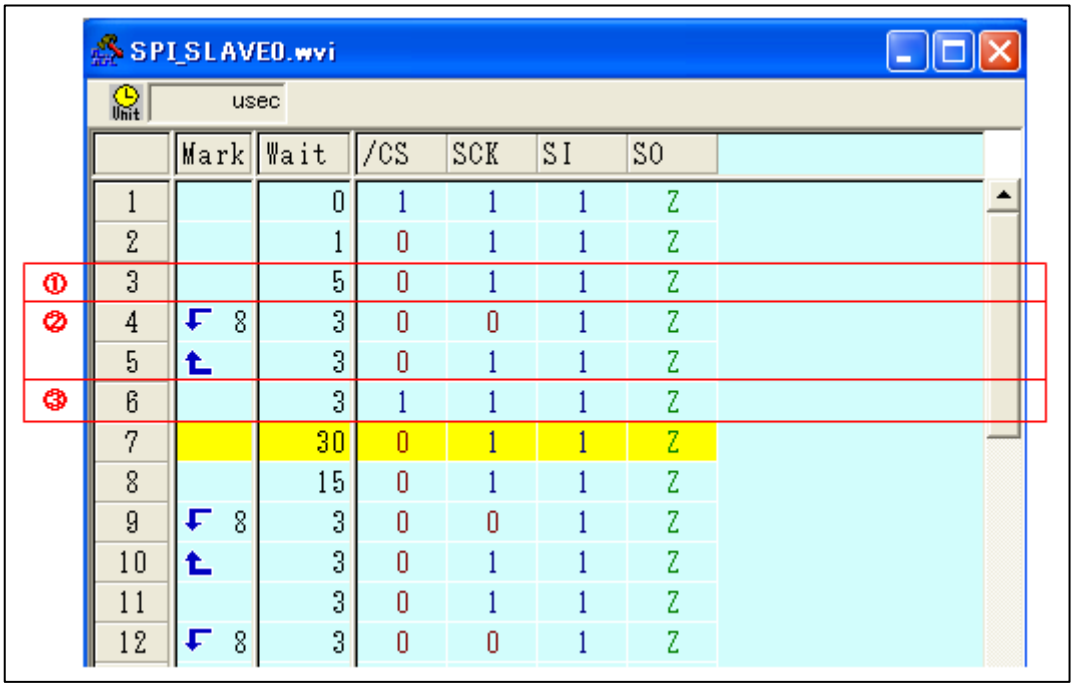
(1) プログラム例

1 バイトのデータを送信する例を紹介します。最初にサブルーチン **INIT_BUFF** を用いて、バッファ領域の初期化を行います。次にサブルーチン **ADD_DATA** を用いて、データ(55H)をバッファにセットします。これで送信準備完了で、割り込みを許可して INTP0 割り込みを待ちます。送信完了は、サブルーチン **CHECK_TXBUFF** で確認します。

```
;
;    送信例 1
;
;   標準の 16 バイトのバッファを使用して 1 バイト単位で送信
;
      CALL            !INIT_BUFF              ;バッファ初期化
      MOV             X,#055H
      CALL            !ADD_DATA               ;送信データをセット
      EI
TXLOOP1:
      CALL            !CHECK_TXBUFF           ;送信バッファを確認
      BC              $TXLOOP1                ;バッファが空になるまで待つ
```

(2) SM+での 1 バイト送信時のタイミング検証

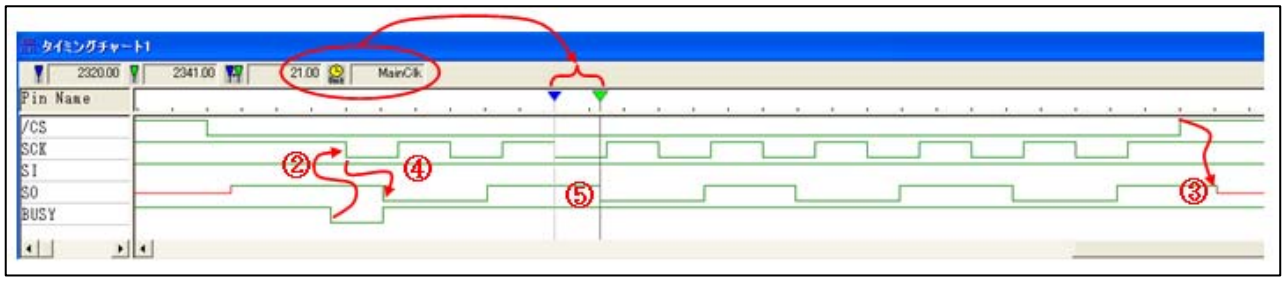
プログラムを動作させて、タイミングを確認します。外部からの信号をシミュレートするため「信号データ・エディタ」を利用して、$\overline{CS}$ などの入力信号のタイミングを定義します。また、時間の単位として μs を選択します。リセット解除後からのデータを準備するのは大変なので、上記プログラムの **EI** 命令にブレーク・ポイントを設定して、プログラムの実行をここで止めた状態で、信号データ・エディタからの入力を開始するようにします。このときのデータの先頭部分を次に示します。$\overline{CS}$ 信号を入力して(次図の①)8 μs 後に、$\overline{SCK}$ 信号を立ち下げて通信を開始し、以降は 3 μs ごとに $\overline{SCK}$ 信号を 8 回反転させます(②)。これは、周期が 6 μs(約 170kbps)に相当します。その後、$\overline{CS}$ 信号を立ち上げます(③)。



	Mark	Wait	/CS	SCK	SI	SO
1		0	1	1	1	Z
2		1	0	1	1	Z
① 3		5	0	1	1	Z
② 4	8	3	0	0	1	Z
5		3	0	1	1	Z
③ 6		3	1	1	1	Z
7		30	0	1	1	Z
8		15	0	1	1	Z
9	8	3	0	0	1	Z
10		3	0	1	1	Z
11		3	0	1	1	Z
12	8	3	0	0	1	Z

次に、このときの信号波形を示します。$\overline{CS}$ 信号が立ち下がると、SO 信号は Hi-Z 状態(赤い表示)からハイ・レベル出力に推移しますが、BUSY 信号は抵抗でプルアップされているので、最初からハイ・レベルになっています。

BUSY 信号が立ち下がってから、$\overline{SCK}$ 信号が立ち下がります(②)。それを確認して、BUSY 信号が立ち上がると同時に、送信データ 55H の MSB が 0 なので、SO 信号が立ち下がります(④)。この波形で一番タイミングの苦しいのが 3 ビット目です(⑤)。この部分の遅れは 21 クロック(約 2.6 μs)です。8 ビットのデータの送信が完了すると $\overline{CS}$ 信号が立ち上がり、それを受けて、SO 信号が Hi-Z 状態になります(③)。



2. 複数バイト送信の使用例

(1) プログラム例

この例では、ROM(CSEG)領域に **DB** 疑似命令で定義したデータを送信します。このデータが、格納領域を送信バッファとすることで、データのコピー作業を省略しています。

```
    CSEG
TXDATA:
    DB              0AAH, 0CCH, 33H, 00, 0FFH, 01, 02, 03
```

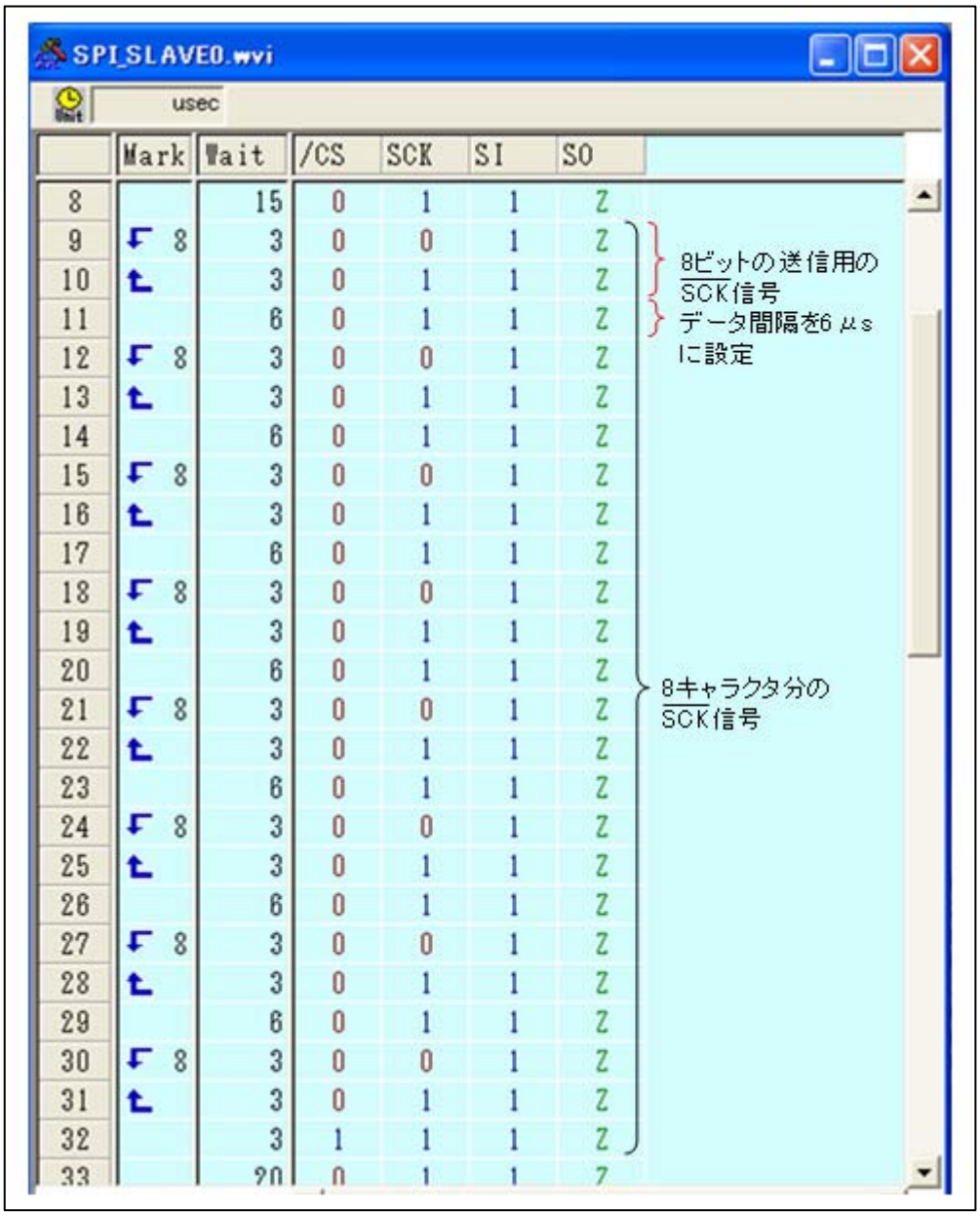
実際のプログラムでは、送信したいデータ数(この例では 8 バイト)を X レジスタに設定して、**PUSH** 命令でスタックにセットします。その後、データの先頭アドレス(この例では **TXDATA**)を AX レジスタにセットして、サブルーチン **SET_DATA** を呼びます。これで、送信に必要な設定が完了するので、**EI** 命令で割り込みを許可し、INTP0 割り込みを待ます。送信データの設定が完了したら、サブルーチン **CHECK_TXBUFF** を用いて、送信完了を待ちます。

```
;
;    送信例 2
;
;   データ・テーブル TXDATA から 8 バイトを送信
;
    DI
    MOV             X, #8                   ;送信データ数は 8 キャラクタ
    PUSH            AX                      ;スタックにセット
    MOVW            AX, #TXDATA             ;バッファの先頭アドレス設定
    CALL            !SET_DATA
    POP             AX                      ;スタックを元に戻す
    EI
TXLOOP2:
    CALL            !CHECK_TXBUFF           ;送信バッファを確認
    BC              $TXLOOP2                ;バッファが空になるまで待つ
```

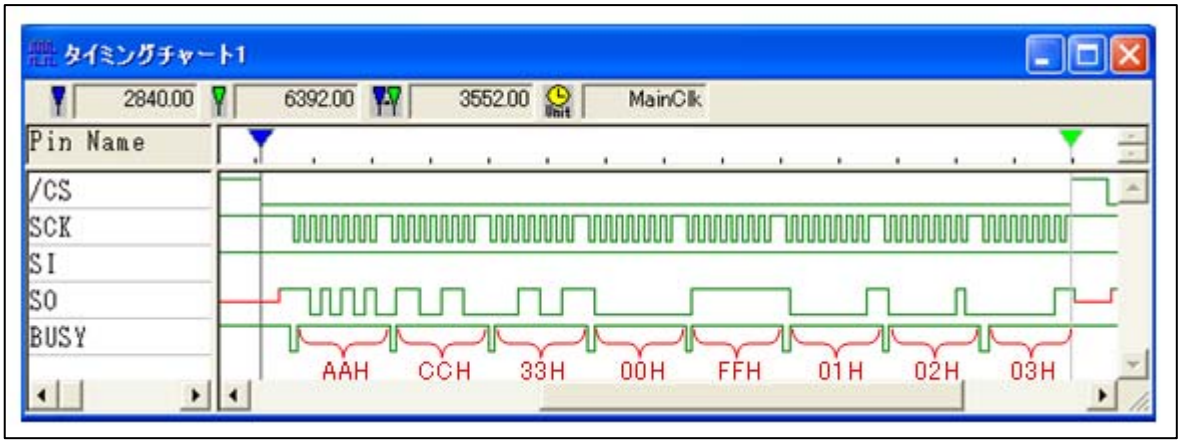
【注意】
サブルーチン **SET_DATA** は、排他制御(割り込みの禁止を制御)しているので、このようにメイン処理での制御は必要ないのですが、ここでは分かりやすくするために、メイン処理でも割り込みの制御を行っています。全体の動作としては、上位側で割り込みを制御した方が、**EI** や **DI** だけで済むので、処理は単純になります。

(2) SM+での複数バイト(8 バイト)送信時のタイミング検証

複数バイト送信を確認するためのタイミング・データは、次のようになります。3 μs ごとに $\overline{SCK}$ 信号をロウ／ハイと 8 回切り替え、1 バイト(1 キャラクタ)の転送タイミング($\overline{SCK}$ 信号)を生成しています。1 キャラクタの最後の SCK 信号と、次のキャラクタの $\overline{SCK}$ 信号の間に 6 μs の間隔を設定し、8 キャラクタ分の信号を生成します。



このときの信号波形は、次のようになります。指定されたデータが、順に SO 信号として出力されています。また、BUSY 信号も 1 キャラクタごとにハイ／ロウが変化しています。



3.1 バイト受信の使用例 1

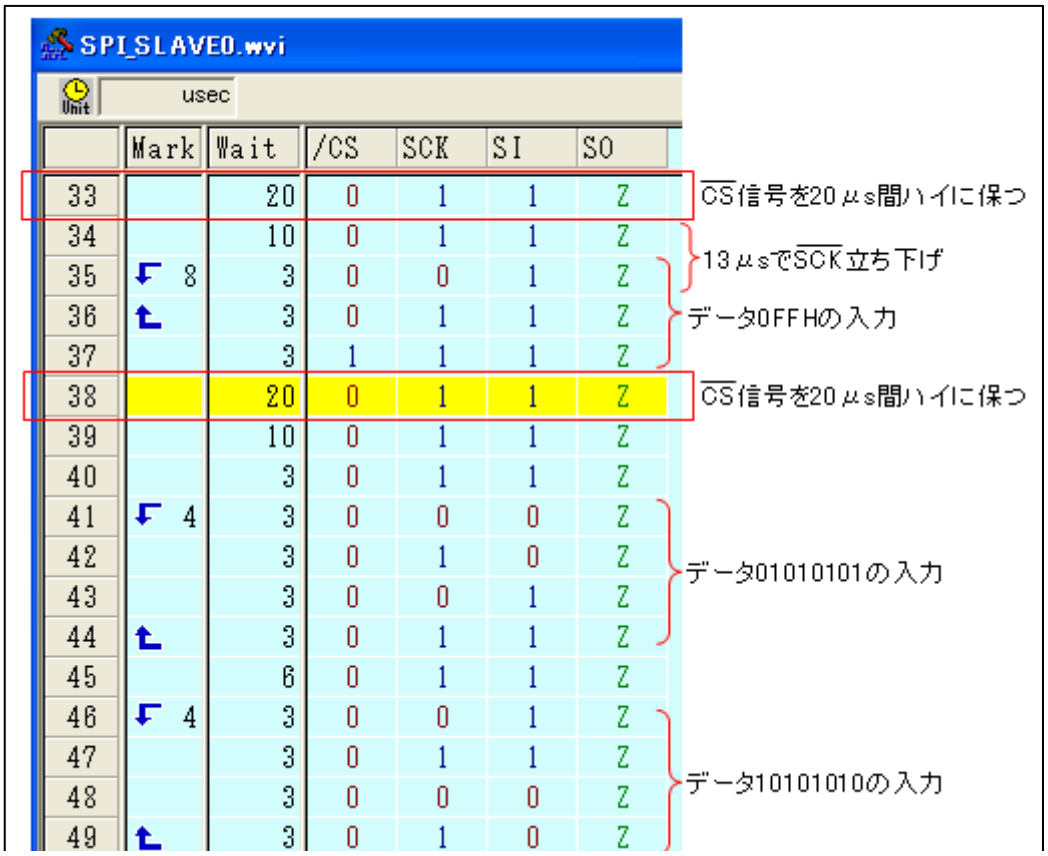
このプログラム例は、受信ステータスを確認し、受信データがあれば、バッファから読み出します。受信データが 03(ETX のコード)になるまで継続します。なお、この受信処理は、かなりの時間が必要になるため、1 キャラクタのデータをバッファから読み出している間に、複数のデータが割り込み処理で受信バッファに格納されます。

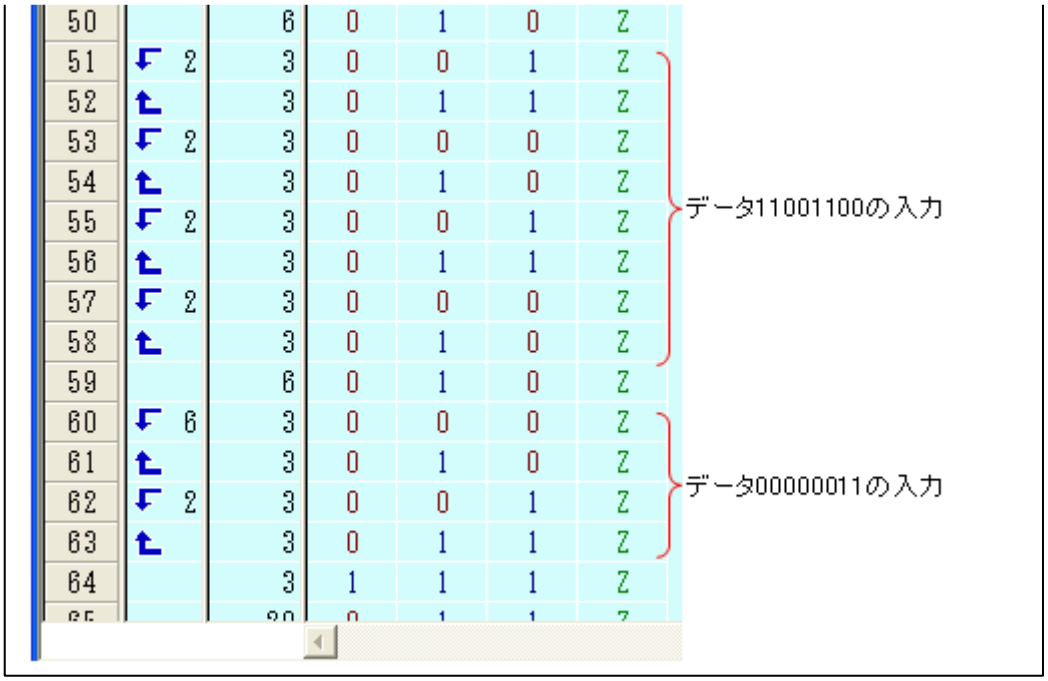
(1) プログラム例

```
;
;    受信例 1
;
;   標準のバッファに受信する。受信したデータが ETX(03) なら終了する
;
    EI
RXLOOP1:
    CALL        !CHECK_RXBUFF       ;受信ステータス確認
    BC          $RXLOOP1            ;受信バッファ空ならループ
    CALL        !GET_DATA           ;受信データ読み出し
    MOV         A,#03               ;ETX
    CMP         A,C                 ;受信データを比較
    BNZ         $RXLOOP1            ;ETX を受信するまで継続
    DI
```

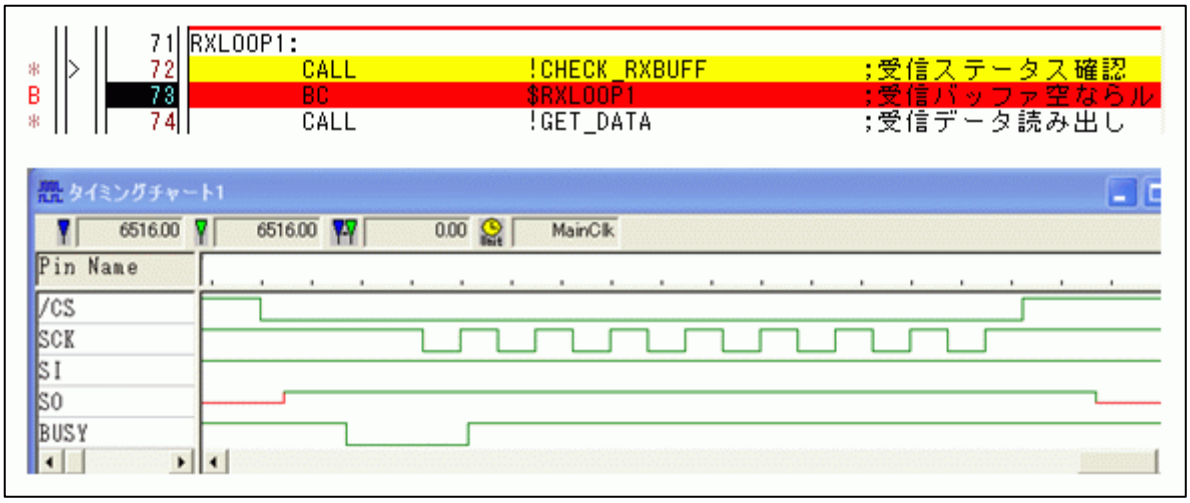
(2) SM+での受信時のタイミング検証

「信号データエディタ」の入力パターンを次に示します。ここでは、8 ビットのデータ(0FFH)を入力してから、$\overline{CS}$ 信号を立ち上げています(38 行目の黄色で示された部分まで実行)。以降は、01010101(55H)、10101010(AAH)、11001100(CCH)、00000011(03)を SI 信号に入力しています。

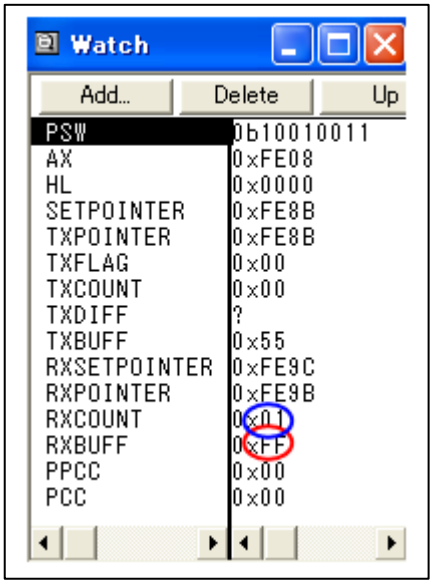




このとき、プログラムは受信バッファを確認し、まだ受信完了していないため、ループでブレークした状態です。実際は次のタイミング・チャートのように、すでに受信は完了しているので、2 回目のチェックでループから抜けます



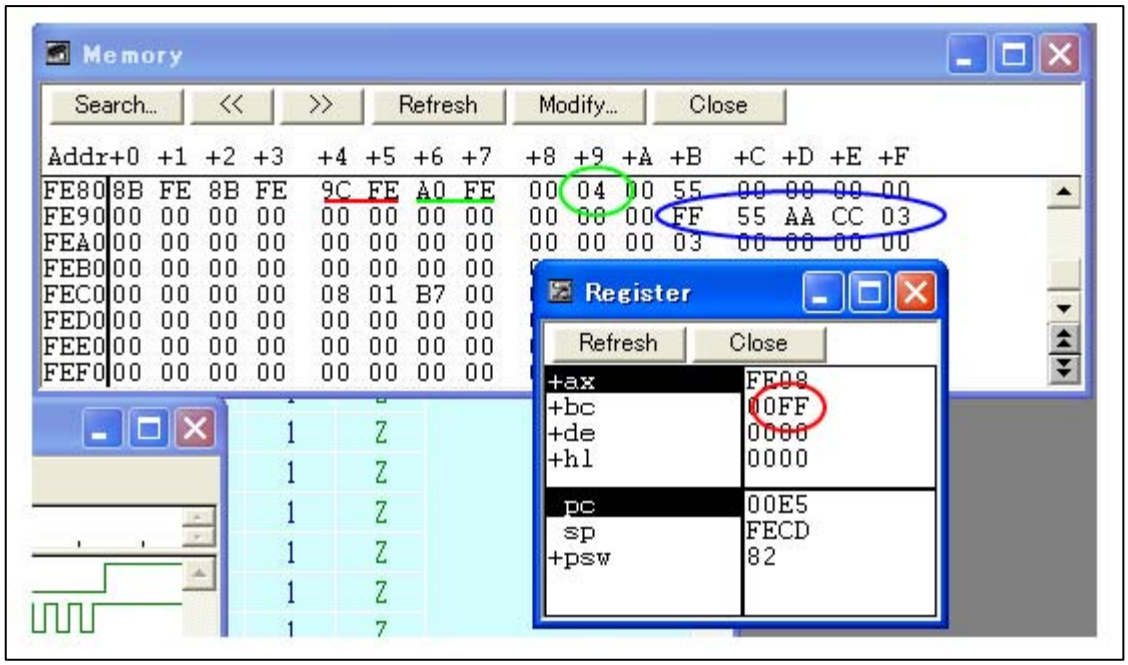
このときの変数を確認すると、下図のように、RXBUFF に 0FFH(**赤丸**)が入っています。また、変数 RXCOUNT のデータ数が 1(**青丸**)になっています。



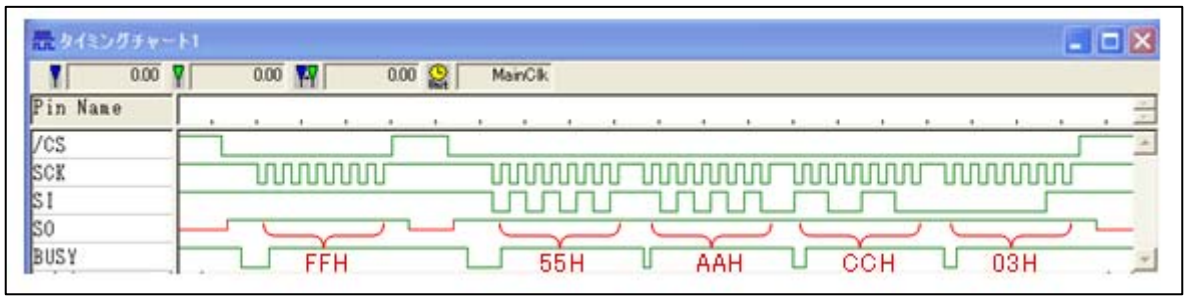
処理を継続して、受信データの読み出しまで実行させたのが、次図の状態です。

```
71 RXLOOP1:
72         CALL    !CHECK_RXBUFF       ;受信ステータス確認
73         BC      $RXLOOP1            ;受信バッファ空ならル
74         CALL    !GET_DATA           ;受信データ読み出し
75         MOV     A,#03               ;ETX
76         CMP     A,C                 ;受信データを比較
77         BNZ     $RXLOOP1            ;ETXを受信するまで継続
78         DI
```

この状態では、プログラムは最初のデータ(FFH)を読み出しただけで、その値が C レジスタに格納されています(次図の**赤丸**)が、受信バッファには 5 バイト(FFH、55H、AAH、CCH、03)のデータが格納されています(**青丸**)。受信バッファには5バイトのデータが格納されていますが、すでにプログラムで最初の受信データが読み出されているので、バッファに残ったデータ数は 4(**緑丸**)になっています。**赤い下線部**が読み出し用のポインタで、2 バイト目のデータ(55H)が格納された FE9CH 番地を示しています。**緑の下線部**が、次の受信データを格納するアドレスを示しています。



次に、ここまでの信号波形を示します。



このように、受信は割り込みで処理され、受信したデータはバッファに格納されていくので、受信バッファがあふれないかぎり、受信処理を意識する必要はありません。

4. 1 バイト受信の使用例 2

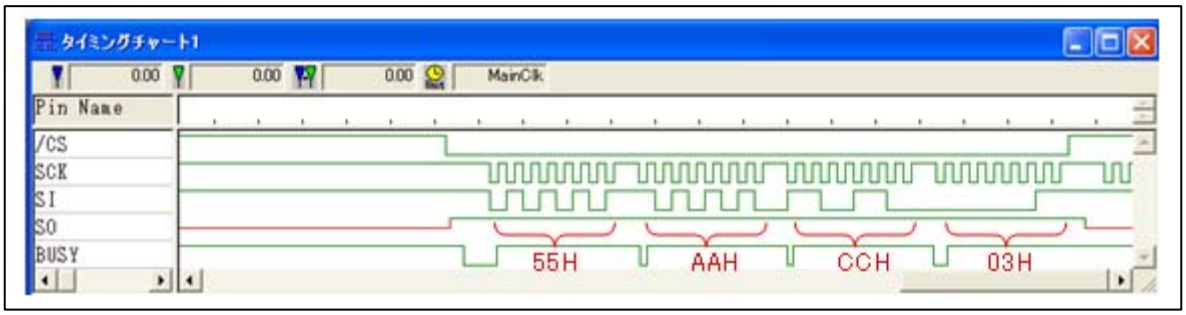
次の使用例は、サブルーチン GET_DATA だけで処理を行います。アセンブラ記述では、サブルーチンの戻り値として、CY フラグ(受信データの有無)と C レジスタの両方を利用できるので、このような処理が可能です。

(1) プログラム例

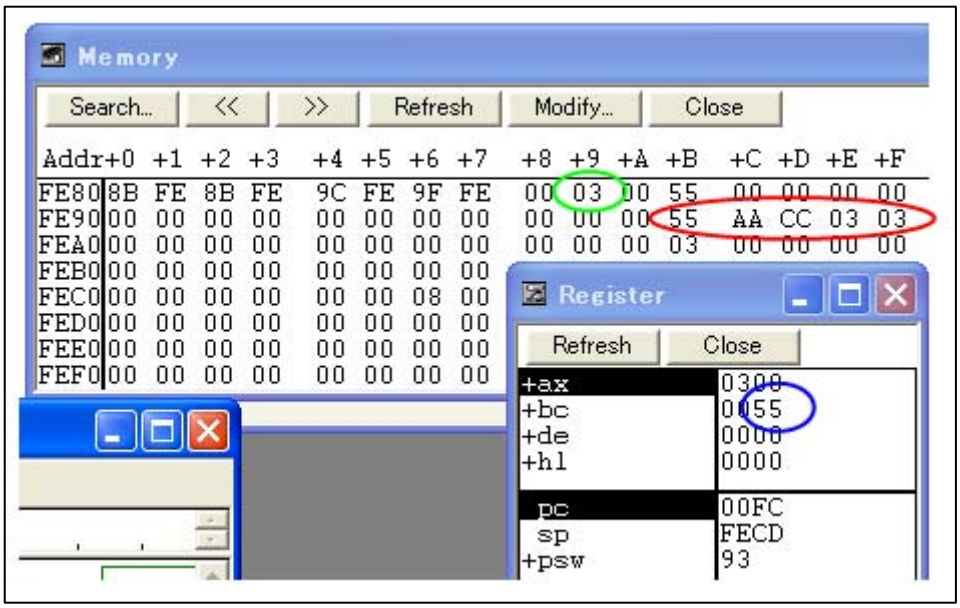
```
    EI
RXLOOP2:
    CALL        !GET_DATA           ;受信データ読み出し
    BC          $RXLOOP2            ;受信バッファ空ならループ
    MOV         A, #03              ;ETX
    CMP         A, C                ;受信データを比較
    BNZ         $RXLOOP2            ;ETX を受信するまで継続
    DI
```

(2) SM+での検証

上記のプログラムでサブルーチン GET_DATA を 1 回実行したあと、次のように SI 信号にデータを入力しています。03 を受信すると $\overline{CS}$ 信号が切れるので、$\overline{SCK}$ 信号の変化は無視されます。



次に、プログラムが最初の受信バッファから受信データ 55H を読み出したときの状態を示します。



5. ベクタ・テーブルなどの設定

このプログラムを使用するために、次のように、ベクタ・テーブルやスタック領域を確保しておきます。

```
RSTVECT     CSEG    AT 0
    DW              START
    ORG     0AH
    DW              __INTP1
STKADR      DSEG    IHRAM
    DS              32                  ; スタック領域(32 バイト)
STACK:
```

[C言語プログラムでの使用方法]

ここでは、C 言語のプログラムから「**周辺プログラム**」で説明した処理を介して通信を行います。CC78K0S では、最初に pragma 指令で、次のように宣言します。

```
#pragma     SFR
#pragma     EI
#pragma     DI
#pragma     NOP
```

サブルーチン群（関数）は PUBLIC 宣言されているので、次のようにプロトタイプ宣言に extern 宣言すれば、リンクできます。

```
/*-------------------------------------------------------------*/
/*                                                             */
/*  使用する関数の定義                                         */
/*                                                             */
/*-------------------------------------------------------------*/

/*  バッファ初期化処理関数          */
extern      void    _init_buff(void);

/*  送信ステータス確認関数          */
extern      unsigned char   _check_txbuff(void);

/*  受信ステータス確認関数          */
extern      unsigned char   _check_rxbuff(void);

/*  1バイト送信関数                 */
extern      void    _add_data(unsigned char data);

/*  ブロック送信関数                */
extern void _set_data(unsigned char *data,unsigned char size);

/*  1バイト受信関数                 */
extern unsigned char        _get_data(void);

/*  受信バッファ設定関数            */
extern void _set_rxbuff(unsigned char *data);
```

1. 割り込みベクタの設定

CC78K0S の場合、スタックはスタートアップ･ルーチンが自動的に設定するので、アセンブラ記述の割り込みを使用するには、ベクタを設定するだけです。ここでは、簡素化のため、次のようなアセンブラ記述のプログラムをリンクしておきます。

```
;
;   割り込みベクタ・テーブル定義
;
EXTRN               __INTP1                         ;割り込み処理部
RSTVECT     CSEG    AT 0AH
    DW              __INTP1
    END
```

2. 1 バイト送信の使用例

使用するにはまず、_init_buff()関数を呼び出して、バッファの初期化や割り込み関係の設定を行います。

```
main(void)
{
    unsigned char   RxData;
    _init_buff();           /* バッファの初期化             */
    RxData = 0;             /* 受信データ初期化             */
/*********************************************************************

    1バイトのデータ送信

*********************************************************************/
    _add_data(0x55);        /* 1バイトのデータ(55H)送信     */
    EI();                   /* 割り込み許可                 */
/*
    送信完了確認
*/
    while(_check_txbuff() != 0){
            NOP();
    }
    DI();
```

初期化が完了したら、送信データを引数として、_add_data()関数を呼び出します。これで、送信バッファにデータがセットされるので、あとは割り込みを許可して転送可能にします。転送完了待ちのため、_check_txbuff()関数の戻り値(＝残り送信データ数)が 0 になるまで while 文で待ちます。while 文を抜けたら、送信は完了です。

3. 複数バイト送信の使用例

送信データを設定する関数が異なるだけで、全体的な処理の流れは 1 バイトの送信と同じです。使用する関数は _set_data()です。この関数では、第 1 引数で送信データの格納された配列のアドレス、第 2 引数でデータ数を示します。

```
/*********************************************************************

    ブロック（8バイト）データ送信

*********************************************************************/
    _set_data(TxData1,8);   /* 送信データのセット           */
    EI();
/*
    送信完了確認
*/
    while(_check_txbuff() != 0){
            NOP();
    }
    DI();
```

このプログラムでは、次のように定義した定数の配列 TxData1 の 8 バイトのデータを送信します。

```
    const   unsigned char   TxData1[8] =
            {0xAA,0xCC,0x33,0x00,0xFF,01,02,03};
```

4. 1 バイト受信の使用例

受信は割り込みで処理されるので、データは受信バッファに格納されます。そのため、プログラムは受信バッファの制御をするだけです。具体的には、_check_rxbuff()関数で受信バッファのデータ数をチェックし、データがあれば、そのデータを_get_data()関数で読み出します。

```
/************************************************************************

    データ受信

************************************************************************/
    EI();
    while(RxData != 03){
            while(_check_rxbuff() == 0){
            NOP();
            }
            RxData = _get_data();
    }
```

このプログラムでは、受信データが 03(ETX)になるまで、while 文で待つ処理にしています。

★

ソフトウエアでの半二重 3 線式シリアル・スレーブ機能の実現(2) [78K0S/Kx1+]

「ソフトウェアでの 3 線式シリアルのスレーブ機能検討」の結果から、次の条件の 3 線式シリアルのスレーブ機能をソフトウエアによるポート制御で実現してみます。

- スレーブとして送受信動作か受信動作を行う
- CS はエッジ検出割り込みを使用(データ転送まで約 12μs 必要)
- CS に対して、BUSY 信号をサポート
- 通信タイプ 1(SCK 立ち下がりで送信、立ち上がりで受信)
- 転送速度は 160kbps 程度(83kbps 以上)までに対応

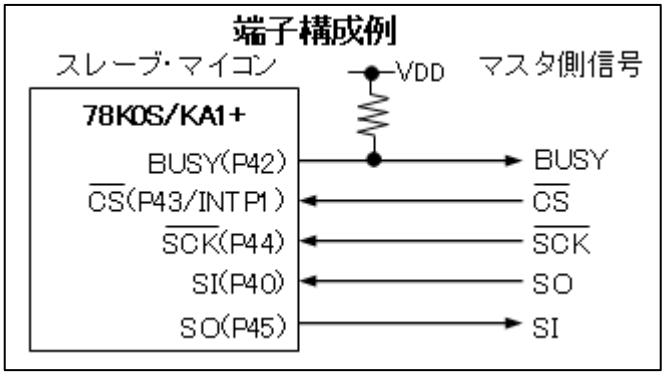
通信はループ処理をすることで、プログラムのサイズを小さくします。

[仕様検討]

1. 接　続

ここでは、次の端子構成例のように、2 本のハンドシェーク信号を含めて 5 本のポートを使用するものとします。

「ソフトウエアでの半二重 3 線式シリアル・スレーブ機能の実現(1)」とは、SI 信号と SCK 信号に使用する端子が入れ替わっています。



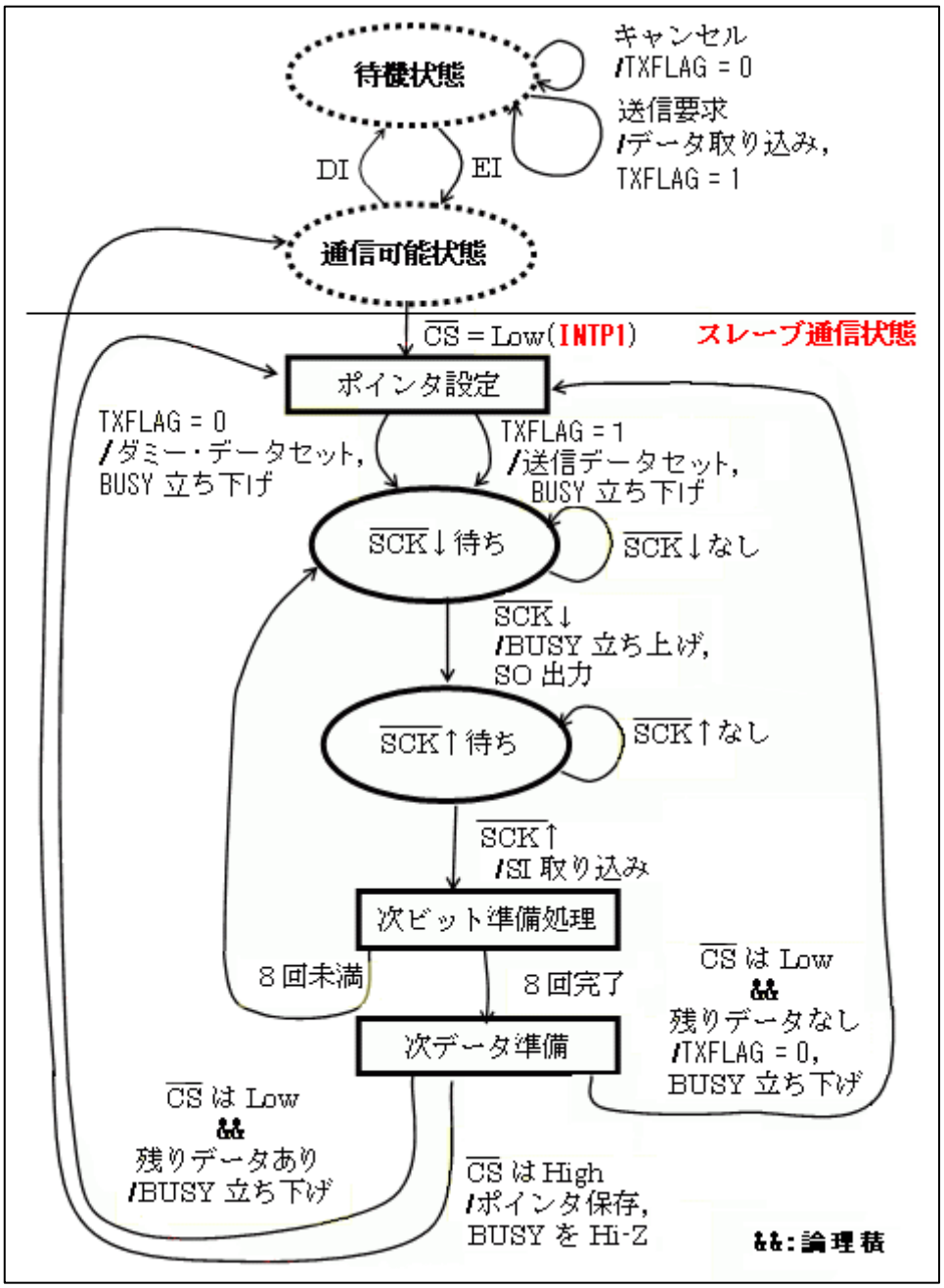
2. 動作概要

SPIのマスタからの$\overline{CS}$信号の立ち下がりをエッジ割り込みで検出し、ベクタ割り込みで通信を起動します。送信データの有無は、事前にフラグに設定しておきます。$\overline{SCK}$信号の立ち下がりでデータをSO端子から送信しますが、フラグの状態で送信データがない場合は、ダミー・データ(0FFH)をSO端子から送信します。次に、$\overline{SCK}$信号の立ち上がりで、SI端子の状態を読み込みます。8ビットのデータを読み込むと、その結果をバッファのに保存します。その後、残り送信データがなくなれば、フラグをクリアして、以降は受信で動作(ダミー・データを送信)することになります。CS信号を確認して、ハイ・レベルなら処理を終了します。ロウ・レベルが継続していれば、処理を継続します。

なお、データはMSBファーストで転送します。また、バッファは16バイトの容量を確保しておき、先頭から使用するだけで、リング・バッファ構造にはしません。

3. 全体の処理

ここで考えている、通信処理全体の概略状態遷移図を次に示します。



初期状態（待機状態）では、割り込みは禁止で、送信データの設定が可能です。その状態で、EI 命令で割り込みを許可すると、通信が可能（状態遷移の通信可能状態）になります。その状態で、$\overline{CS}$ 信号が立ち下がると、ベクタ割り込みを受け付けて、スレーブ通信状態となります。

スレーブ通信状態になると、通信に使用する BUSY 信号や SO 信号を出力に設定します。必要なポインタの設定を行ってから BUSY 信号を立ち下げて、$\overline{SCK}$ 信号の立ち下がりを待ちます。$\overline{SCK}$ 信号が立ち下がると、SO 信号に出力するのと同時に、BUSY 信号を立ち上げます。このとき、送信するデータがなければ、ダミー・データ（0FFH）を出力するので、SO 端子はハイ・レベルになります。その後、SCK 信号の立ち上がりに同期して、SI 信号を読み込みます。8 ビット分の受信が完了すると、データをバッファに保存し、送信データの残り数を確認して、BUSY 信号を立ち下げ、次の通信準備ができたことをマスタに通知して、次のデータ通信か通信完了（$\overline{CS}$ 信号の立ち上がり）を待ちます。

このあとは、3 つの条件に分かれます。

①次の送信データが残っている状態で、$\overline{CS}$ 信号がロウ・レベルのままなら、次データ送信のための BUSY 信号を立ち下げて、次の通信準備ができたことをマスタに通知し、$\overline{SCK}$ 信号の立ち下がり待ちとなります。

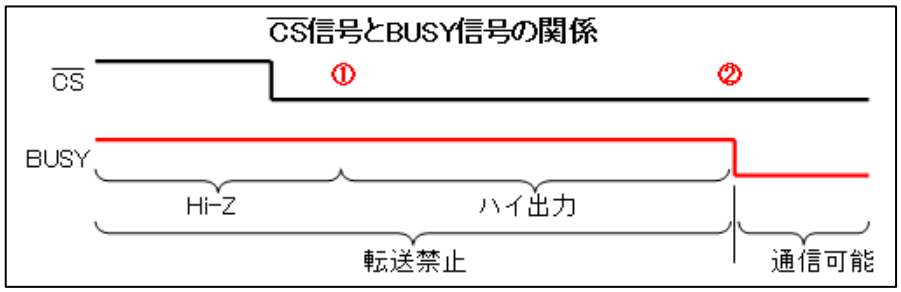
②送信データがなくなって、$\overline{CS}$ 信号がロウ・レベルのままなら、BUSY 信号を立ち下げて、次の通信準備ができたことをマスタに通知し、$\overline{SCK}$ 信号の立ち下がり待ちとなります。以降は、ダミー・データを送信して、受信処理だけを行います。

③$\overline{CS}$ 信号がハイ・レベルになった場合は、BUSY 信号を Hi-Z にしてから割り込み処理を抜け、通信可能状態になります。

[制御プログラム検討]

1. 通信の起動（$\overline{CS}$ 信号の立ち下がり検出）

通信を開始するタイミングは、$\overline{CS}$ 信号の立下りをエッジ検出したベクタ割り込みで検出することにします。このとき、送信データがあるかどうかのフラグで、データを送信するかダミー・データを送信するかを判断します。データ送信の場合は、バッファからデータを読み出し、送信の準備を行います。$\overline{CS}$ 信号の立下りから、実際にデータを送受信(特に送信)できるまでは「80 クロック＋割り込み応答時間（最大 19 クロック）」で、8MHz 動作とすると 12 μs が必要です。ただし、$\overline{CS}$ 信号の立が下りから「6 クロック＋割り込み応答時間（最大 19 クロック）」で BUSY 信号が出力されますし、それ以前の BUSY 信号は Hi-Z（ハイ・レベル）となっているので、BUSY 信号を利用すれば、この部分の応答速度をカバーできます。



次に、この起動部分のプログラム例を示します。BUSY（P42 の出力ラッチ）には初期状態で 1 がセットされているので、1 行目で PM4 を設定した段階(①)でハイ・レベルが出力されます。通信可能（次の例では送信可能）になると(②)、BUSY 信号を立ち下げて、通信準備ができたことをマスタに知らせます。なお、データ送信のための $\overline{SCK}$ 信号確認までは、さらに 13 クロックかかります。最初のビットの処理を 2 ビット目以降と分けると、この時間を 5 クロックまで短縮できますが、マスタでの BUSY 信号の立ち下がり確認から $\overline{SCK}$ 出力までの時間を考えると、あまり効果的ではありません。

```
__INTP1:
                MOV     PM4,# 11011011b     ; 6:①So(P45) と BUSY(P42) を出力に
                PUSH    AX                  ; 4:使用レジスタをセーブ
                PUSH    HL                  ; 4:
                PUSH    DE                  ; 4:
                PUSH    BC                  ; 4:
                MOVW    AX,TxPointer        ; 6:送信ポインタ読み出し
                MOVW    HL,AX               ; 4:ポインタをセット
                MOVW    AX,RxPointer        ; 6:受信ポインタ読み出し
                MOVW    DE,AX               ; 4:ポインタをセット
RTxByte:
                MOV     A,#0FFH             ; 6:ダミー・データの設定
                BF      TxFlag,$RxData      ;10:フラグが 0 なら受信処理へ
RTxLoop:
                MOV     A,[HL]              ; 6:送信データの読み出し
Loop1:
                BT      P4.3,$RTxExit       ;10:CS の立ち上がりで抜ける
RxData:
                MOV     B,#8                ; 6:ビット数をセット
                CLR     P4.2                ; 6:②BUSY を立ち下げる
;   通信準備完了
```

青字は、送信データの有無で送信データを選択する部分です。ここで、データの間隔が 2μs 長くなります。この処理をなくすには、送信データ・バッファに、最初からダミー・データをセットしておけばいいことになります。

2.1 キャラクタ(8 ビット)の通信処理

通信可能状態で $\overline{CS}$ 信号の立ち下がりを検出したら、スレーブ通信状態に移ります。通信処理がスタートすると、$\overline{SCK}$ 信号の立ち下がりを検出して、データを SO 端子に出力します。このとき、同時に BUSY 信号を立ち上げて、送信処理を開始したことを示します。この処理では、SO 信号を出力する **MOV 命令**で、BUSY 信号の立ち上げも同時に行って、処理時間の増加をなくしています。その部分を含めた処理は、次のようになります。ここでは、$\overline{SCK}$ 信号の立ち下がりからの遅延をできるだけ小さくするため、送信データが 0 か 1 かを事前に判定しています。

```
                CLR1    CY                  ; 2:次のビット用にクリア
Loop2:
                ROLC    A,1                 ; 2:出力データをキャリーへ
                BC      $Data1              ; 6:データが 1 なら分岐
;
;   出力データが 0 の場合の処理
;
Data0:
                BT      P4.4,$Data0         ;10:SCK 立ち下がり待ち
                MOV     P4,#00000100b       ; 6:送信&BUSY セット
                MOV     SIO,A               ; 4:データをセーブ
                CLR1    CY                  ; 2:次のビット用にクリア
Loop3:
                MOV     A,P4                ; 4:ポートの読み出し
;               BF      A.4,$Loop3          ; 8: SCK 立ち上がり待ち
                AND     A,#0000001B         ; 4:SI データ以外をマスク
                OR      A,SIO               ; 4:ビットを追加
                DBNZ    B,$Loop2            ; 6:8 ビット分繰り返し
                BR      $RTxEnd
```

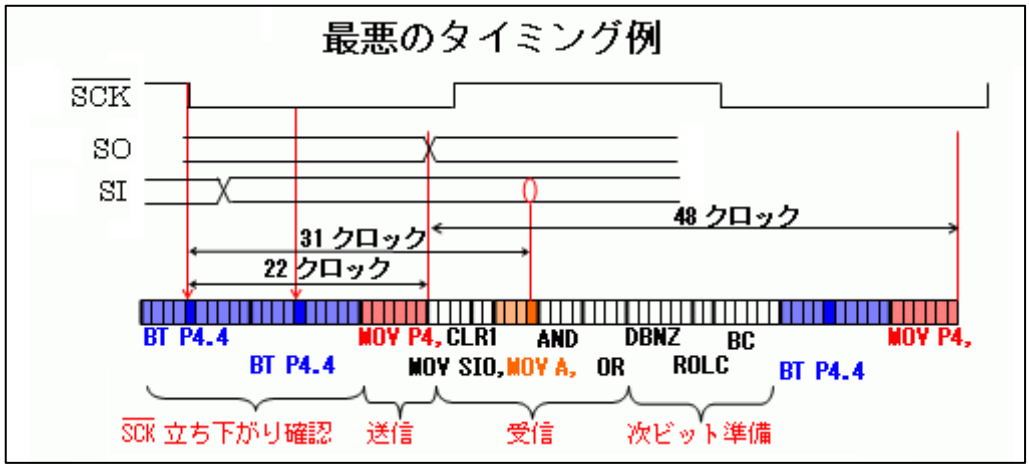
```
;
;   出力データが 1 の場合の処理
;
Data1:
                BT      P4.4,$Data1             ;10:SCK 立ち下がり待ち
                MOV     P4,#00100100b           ; 6:送信&BUSY セット
                MOV     SIO,A                   ; 4:データをセーブ
                CLR1    CY                      ; 2:次のビット用にクリア
Loop4:
                MOV     A,P4                    ; 4:ポートの読み出し
;               BF      A.4,$Loop4              ; 8: SCK 立ち上がり待ち
                AND     A,#0000001B             ; 4:SI データ以外をマスク
                OR      A,SIO                   ; 4:ビットを追加
                DBNZ    B,$Loop2                ; 6:8 ビット分繰り返し
```

データのビット送信が完了したら受信処理を行います。本来は $\overline{SCK}$ 信号の立ち上がりを確認して、SI 信号を取り込みます(プログラムで緑字のコメント部)。ここでは、通信速度をできるだけ低下させないように、この確認を省略(コメント・アウト)しています。転送速度が遅い場合は、$\overline{SCK}$ 信号の立ち上がりより前に SI 信号を取り込む可能性があります。ほとんどの場合、$\overline{SCK}$ 信号の立ち下がりで出力されることになり、送信処理の期間にはデータが確定していると考えられるので、コメント・アウトしています。

この送受信処理では、共通の変数(**変数 SIO**)でデータを処理しています。送信のために送信データを左シフトして、ビット 0 を受信用に空けておき、SI 信号(ビット 0)を OR 演算して、新たなビットを取り込んでいきます。

B レジスタをループ・カウンタとして、**Loop2** までを 8 回ループすることで、1 バイトのデータの送受信が完了します(**B** レジスタを使用すれば、2 クロック分短くなります)。

実際の送受信タイミングは、次のようになります。このプログラムは、1 ビット分の送信と受信に 48 クロック必要です。つまり 1 ビットの時間は 6μs となります。$\overline{SCK}$ 信号の立ち上がりをチェックしていないので、次ビットの **BT** 命令の実行まで $\overline{SCK}$ 信号が連続していなければ(6μs 以上の長さで、83kbps 以下の転送レートでなければ)対応でき、最大で 166kbps の転送速度まで可能です。



3. キャラクタ完了時の処理

1 キャラクタの送信、受信が完了したら、受信データを受信バッファに保存し、送信バッファに次の送信データがあるかどうかを確認します。次に、このプログラム例を示します。

```
;
;       バイト・データの送受信完了
;
RTXEND:
                MOV     [DE],A                  ; 6:受信バッファに格納
                INCW    DE                      ; 4:受信ポインタ更新
                INC     RxCount                 ; 4:受信データ数更新
                MOV     A,#0FFH                 ; 6:ダミー・データをセット
                BF      TxFlag,$Loop1           ;10:送信データなしで分岐
                INCW    HL                      ; 4:送信ポインタ更新
                DBNZ    TxCount,$RTxLoop        ; 8:残り送信データがあれば分岐
                CLR1    TxFlag                  ; 6:送信フラグをクリア
                BR      Loop1                   ; 6:受信だけ(0FFH 送信)
```

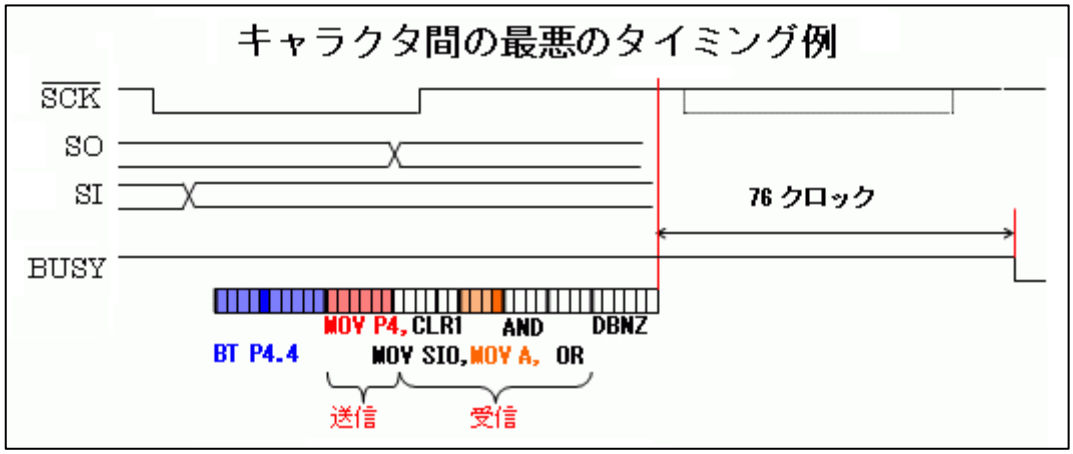
ここでは、処理が次のように 3 つの場合に分けられます。

・まだ送信データがある　　：次のデータを読み出して通信処理へ

・最後のデータ送信を完了　：送信フラグをクリアして受信処理へ

・受信だけを処理中　　：ダミー・データをもって受信処理へ

このうち最も処理時間のかかるのは、最終データの送信が完了したときです。この場合、全命令を実行することになり、それだけで 54 クロックです。これに通信準備の 22 クロックを加算して、76 クロックで BUSY 信号の立ち下げとなります。



BUSY 信号が立ち下がるのは、最終ビットの受信完了から 76 クロックとなり、9μs(1.5 ビット分の間隔)が空くことになります。これを 1 ビット分にするには、ダミー・データを事前に送信バッファに準備しておき、送信フラグを使用しなくても済むようにします。

4. 転送完了時の処理(CS 信号の立ち上がり)

上記のプログラムでは、次のキャラクタの転送開始処理で CS 信号を確認し、ロウ・レベルのままであれば転送処理を継続します。

なお、このプログラム例では、**CS 信号は 8 ビット目の受信処理完了から 35 クロックでしかチェックしていません。そのため、最後のデータの転送が完了したら、マスタは CS 信号を切るようにしてください。どうしても、35 クロックまでに不可能な場合は、転送処理を 1 ビット目と 2 ビット目以降で変える必要があります。**

```
;
;       CS の立ち上がりで、受信処理完了へ
;
RTXEXIT:
                SET1    BUSYPort                ;BUSY を立ち上げ
RTXEXIT2:
                MOV     PM4,#0FFH               ;ポートを Hi-Z に
                MOVW    AX,HL
                MOVW    TXPOINTER,AX            ;送信ポインタを更新
                MOVW    AX,DE
                MOVW    RXSETPOINTER,AX         ;受信ポインタを更新
                CLR1    PIF1
                POP     BC
                POP     DE                      ;レジスタの復帰
                POP     HL
                POP     AX
                RETI
```

[周辺プログラム検討]

以上で説明した制御プログラムを使用するための、サポート・プログラムを紹介します。ここで触れるのは、次のような処理です。なお、プログラムはサブルーチン化して、C 言語記述のプログラムからも関数呼び出しできるようにしてあります。

①初期化処理

②受信バッファの初期化

③送信ステータス確認

④受信ステータス確認

⑤1 バイトのデータを送信バッファに追加

⑥送信バッファ設定

⑦受信バッファから 1 バイトのデータ読み出し

⑧受信バッファ設定

1. 初期化処理

制御プログラムを使用するために、ポインタ類の初期化や受信データ・バッファのクリア、割り込みの設定を行います。この処理は、バッファを制御するためのポインタや変数を設定するため、割り込み禁止で処理させます。通常は、起動後に1回使用します。

```
;*****************************************************************************
;                                                                             *
;       送信／受信（バッファ）の初期化                                        *
;                                                                             *
;  送信／受信で使用するバッファを内部で定義したバッファに設定し、受信        *
;  バッファはデータを 0 クリアしておきます                                    *
;  また、CS 信号の検出で使用する INTP1 を立ち下がりエッジ検出に設定し、       *
;  割り込みマスクの解除までの処理を行います                                   *
;  入力データ     : なし                                                      *
;  戻り値         : なし                                                      *
;  使用レジスタ   : なし                                                      *
;                                                                             *
;       extern void _init_buff(void);                                         *
;*****************************************************************************
```

```
PUBLIC  __init_buff,INIT_BUFF
__init_buff:
INIT_BUFF:
        PUSH    PSW                     ;IE フラグをセーブ
        DI                              ;以下の処理は割り込み禁止で行う
        PUSH    HL
        PUSH    AX

        MOVW    AX,#TxBuff
        MOVW    TxPointer,AX            ;送信データ読み出しポインタ初期化
        MOVW    SetPointer,AX           ;送信データ設定ポインタ初期化
        MOV     TxCount,#0              ;データ数を初期化
        CLR1    TxFlag                  ;送信フラグをクリア

        MOVW    AX,#RxBuff              ;受信バッファの初期化初期化
        MOVW    RxPointer,AX            ;受信データ読み出しポインタ初期化
        MOVW    RxSetPointer,AX         ;受信データ格納用ポインタ初期化
        MOV     RxCount,#16             ;データ数を初期化
        MOVW    HL,AX
        MOV     A,#0
Init_Loop:
        MOV     [HL],A                  ;受信バッファをクリア
        INCW    HL
        DBNZ    RxCount,$Init_Loop
        POP     AX
        POP     HL
        MOV     P4,#00100100b           ;SO と BUSY 出力をハイにする
        MOV     INTM1,#00000000b        ;INTP1 を立ち下がりエッジ検出に
        CLR1    PIF1
        CLR1    PMK1
        POP     PSW
        RET
```

2. 受信バッファの初期化

このプログラムは、送信時に受信も行うため、半二重通信では無意味なデータを受信バッファに取り込みます。このデータを1バイト単位で読み出して確認するのはむだです。そこで、明らかに送信時に受信したデータだけの場合に、受信バッファをクリアする処理を準備しておきます。

```
;*******************************************************************************
;                                                                              *
;       受信(バッファ)の初期化                                                 *
;                                                                              *
;  受信で使用するバッファを内部で定義したバッファに設定し、データを            *
;  0クリアしておきます                                                         *
;       extern void _init_rxbuff(void);                                        *
;                                                                              *
;*******************************************************************************
PUBLIC  __init_rxbuff,INIT_RXBUFF
__init_rxbuff:
INIT_RXBUFF:
        PUSH    PSW                     ;IE フラグをセーブ
        DI                              ;以下の処理は割り込み禁止で行う
        PUSH    HL
        PUSH    AX
```

```
        MOVW    AX,#RxBuff              ;受信バッファの初期化
        MOVW    RxPointer,AX            ;受信データ読み出しポインタ初期化
        MOVW    RxSetPointer,AX         ;受信データ格納用ポインタ初期化
        MOV     RxCount,#16             ;データ数を初期化
        MOVW    HL,AX
        MOV     A,#0
Init_Loop2:
        MOV     [HL],A                  ;受信バッファをクリア
        INCW    HL
        DBNZ    RxCount,$Init_Loop2
        POP     AX
        POP     HL
        POP     PSW
        RET
```

3. 送信ステータス確認

送信データ･バッファの未送信のデータ数を C レジスタに格納します。また CY フラグで、バッファが空かどうかを確認できます。通常は、送信データ･バッファにデータをセットしたあと、送信完了かどうかの確認に使用します。

```
;*******************************************************************************
;                                                                              *
;       送信データ・バッファのステータス確認                                   *
;  入力データ     : なし                                                       *
;  戻り値         : C レジスタ(空なら CY=0)                                    *
;  使用レジスタ   : なし                                                       *
;  バッファの残りデータ数を戻します                                           *
;       extern unsigned char _check_txbuff(void);                              *
;                                                                              *
;*******************************************************************************
PUBLIC  __check_txbuff,CHECK_TXBUFF
__check_txbuff:
CHECK_TXBUFF:
                PUSH    AX
                MOV     A,TxCount               ;バッファ中のデータ数読み出し
                MOV     C,A                     ;戻り値にセット
                SUB     A,#1                    ;空なら CY をセット
                NOT1    CY                      ;CY を反転
                POP     AX
                RET
```

4. 受信ステータス確認

受信データ･バッファの中のデータ数をＣレジスタに格納します。データがあれば、**1 バイト読み出し**などでバッファから読み出してください。

```
;*******************************************************************************
;                                                                              *
;       受信データ・バッファのステータス確認                                   *
;  入力データ     : なし                                                       *
;  戻り値         : C レジスタ、CY フラグ(バッファ空)                          *
;  使用レジスタ   : なし                                                       *
;  バッファ中のデータ数を戻します。データが存在するときにデータを読み         *
;  出してください                                                             *
;       extern unsigned char _check_rxbuff(void);                              *
;                                                                              *
;*******************************************************************************
```

```
PUBLIC  __check_rxbuff,CHECK_RXBUFF
__check_rxbuff:
CHECK_RXBUFF:
                PUSH    AX
                MOV     A,RxCount               ;バッファ中のデータ数読み出し
                MOV     C,A                     ;戻り値にセット
                SUB     A,#1                    ;データ数 0 なら CY をセット
                POP     AX
                RET
```

5. 1 バイトのデータを送信バッファに追加

X レジスタにセットされたデータを送信データ･バッファに追加します。なお、バッファの大きさは管理していないので、送信バッファのサイズ以上にデータを追加しないようにする必要があります。送信バッファ内のデータ数は、**送信ステータス確認**で判断してください。

```
;*******************************************************************************
;                                                                              *
;   送信データ 1 バイトをバッファに追加                                        *
;  入力データ          : X レジスタ                                            *
;  戻り値              : なし                                                  *
;  使用レジスタ        : なし                                                  *
;  バッファのデータがすべて送信されると、TxFlag が 0 になります               *
;   extern void _add_data(unsigned char);                                      *
;                                                                              *
;*******************************************************************************
PUBLIC      __add_data,ADD_DATA
__add_data:
ADD_DATA:
            PUSH    HL
            PUSH    AX
            MOVW    HL,AX                   ;データをセーブ
            MOVW    AX,SetPointer           ;セット・ポインタ読み出し
            XCHW    AX,HL
            MOV     A,X
            MOV     [HL],A                  ;データをバッファにセット
            INCW    HL                      ;ポインタ更新
            XCHW    AX,HL                   ;H:データ、AX:ポインタ
            MOVW    SetPointer,AX           ;ポインタ保存
            PUSH    PSW                     ;PSW のセーブ
            DI                              ;割り込み禁止
            INC     TxCount                 ;データ数を+1
            SET1    TxFlag                  ;送信要求フラグをセット
            MOV     P4,#00100100b           ;SO と BUSY 出力をハイにする
            POP     PSW
            POP     AX
            POP     HL
            RET
```

6. 送信バッファ設定

複数バイトのデータを送信する場合、**1 バイトのデータ送信**を必要な回数使用しても実現できますが、バッファを切り替えたほうが便利です。ここでは、AX レジスタに送信データが格納されたバッファのアドレスを、またスタックにデータ数を設定して、サブルーチン･コールします。このパラメータの渡し方は、CC78K0S の引数の受け渡しと合わせるためです。アセンブラで使用するだけなら、データ数もレジスタで渡す方が、処理が簡単になります。

送信が完了したかどうかは、**送信ステータス確認**で判別できます。

```
;*********************************************************************
;                                                                    *
;       送信データＮバイトをバッファにセット                         *
;  入力データ        : AX レジスタ（バッファの先頭アドレス）  : 第一引数 *
;                    : スタック（データ数）                  : 第二引数 *
;  戻り値          : なし                                            *
;  使用レジスタ  : なし                                              *
;  バッファのデータがすべて送信されると、TxFlag が 0 になります      *
;                                                                    *
;       extern void _set_data(unsigned int,unsigned char);           *
;                                                                    *
;*********************************************************************
PUBLIC  __set_data,SET_DATA
__set_data:
SET_DATA:
                PUSH    PSW                     ;PSW をセーブ
                DI                              ;割り込み禁止
                PUSH    HL                      ;使用レジスタをセーブ
                PUSH    AX
                MOVW    TxPointer,AX            ;送信ポインタを変更
                MOVW    SetPointer,AX           ;送信セット・ポインタを変更
                MOVW    AX,SP                   ;スタック・ポインタ読み出し
                MOVW    HL,AX
                MOV     A,[HL+7]                ;データ数読み出し
;
;  ここでのスタックの内容は、以下のとおり
;       HL(SP)  ---> X
;       HL+1    ---> A
;       HL+2    ---> L
;       HL+3    ---> H
;       HL+4    ---> PSW
;       HL+5    ---> 戻り番地(下位)
;       HL+6    ---> 戻り番地(上位)
;       HL+7    ---> データ数
;       HL+8    ---> ダミー・データ
;
                MOV     TxCount,A           ;データ数を設定
                CLR1    TxFlag              ;フラグをクリアしておく
                AND     A,A                 ;データ数チェック
                BZ      $Set_Data_End       ;データ数 0 ならポインタ設定だけ
                SET1    TxFlag              ;送信要求フラグをセット
                POP     HL
                PUSH    HL
                ADD     A,L
                MOV     X,A
                MOV     A,H
                ADDC    A,#0
                MOVW    SetPointer,AX           ;送信セット・ポインタを変更
Set_Data_End:
                MOV     P4,#00100100b           ;SO と BUSY 出力をハイにする
                POP     AX                      ;レジスタの復帰
                POP     HL
                POP     PSW
                RET
```

7. 受信バッファから 1 バイトのデータ読み出し

受信データ・バッファから、もっとも古いデータを読み出し、C レジスタに格納します。もし、受信データ・バッファにデータがない場合は、C レジスタに 0 が格納され、CY フラグがセットされます。読み出してデータがなくなったら、ポインタを初期値に戻します。

```
;*******************************************************************************
;                                                                              *
;    受信データ 1 バイトをバッファから読み出す                                 *
;   入力データ          : なし                                                 *
;   戻り値              : C レジスタ (データなければ CY=1)                     *
;   使用レジスタ        : なし                                                 *
;   バッファから最も古いデータを読み出し、残りデータ数を-1 にします           *
;    extern unsigned char _get_data(void);                                     *
;                                                                              *
;*******************************************************************************
PUBLIC      __get_data,GET_DATA
__get_data:
GET_DATA:
            PUSH    AX
            MOV     A,RxCount               ;データ数を読み出し
            CMP     A,#1                    ;データ数チェック
            MOV     C,A                     ;ダミーの戻り値をセット
            BC      $GetEnd                 ;データなければ戻る
            PUSH    HL
            MOVW    AX,RxPointer            ;リード・ポインタ読み出し
            MOVW    HL,AX                   ;HL レジスタにセット
            INCW    AX                      ;ポインタ更新
            MOVW    RxPointer,AX            ;ポインタ保存
            MOV     A,[HL]                  ;データ読み出し
            MOV     C,A                     ;戻り値をセット
            POP     HL
            DEC     RxCount                 ;データ数を-1
            BNZ     $GetEnd                 ;残りデータありなら分岐
            PUSH    PSW
            DI                              ;以下の処理は割り込み禁止
            MOVW    AX,#RxBuff              ;受信バッファの初期化
            MOVW    RxPointer,AX            ;データ読み出しポインタ初期化
            MOVW    RxSetPointer,AX         ;受信データ格納用ポインタ初期化
            POP     PSW
GetEnd:
            POP     AX
            RET
```

8. 受信バッファ設定

受信バッファをパラメータとして、渡されたアドレスに変更します。

```
;*******************************************************************************
;                                                                              *
;    受信バッファの設定                                                        *
;   入力データ          : AX レジスタ（受信用バッファの先頭アドレス）          *
;   戻り値              : なし                                                 *
;   受信データを格納するバッファを指定します                                   *
;   また、受信データ数を 0 に設定します                                        *
;   受信完了確認は受信データ・バッファのステータスで行います                   *
;    extern void _set_rxbuff(unsigned int);                                    *
;                                                                              *
;*******************************************************************************
PUBLIC      __set_rxbuff,SET_RXBUFF
__set_rxbuff:
SET_RXBUFF:
```

```
    PUSH    PSW
    DI
    MOVW    RxPointer,AX                ;リード・ポインタをセット
    MOVW    RxSetPointer,AX             ;セット・ポインタをセット
    MOV     RxCount,#0                  ;データ数をクリア
    POP     PSW
    RET
```

9. データ領域の定義

内部で使用している変数やポインタ、バッファは、次のように定義してあります。これらを直接操作することはありませんが、作業用にこれらの領域を確保しています。そのため、メモリの使用量には注意が必要です。

```
;
;   データ・ポインタ領域
;
            DSEG    saddrp
TxPointer:
            DS      2       ;送信用データ読み出しポインタ
SetPointer:
            DS      2       ;送信データ設定用ポインタ
RxPointer:
            DS      2       ;受信データ読み出しポインタ
RxSetPointer:
            DS      2       ;受信データ格納用ポインタ
            DSEG    saddr
;
;   データ・カウンタ領域
;
TxCount:
            DS      1       ;送信データ数カウンタ
RxCount:
            DS      1       ;受信データ数カウンタ
BitCount:
            DS      1       ;ビット数カウント用
;
;   データ領域
;
TxBuff:
            DS      16      ;送信データ・バッファ領域
RxBuff:
            DS      16      ;受信データ・バッファ領域
SIO:
            DS      1       ;受信作業用
;
;   フラグ領域
;
            BSEG
TxFlag              DBIT    ;送信要求フラグ
```

[プログラムの使用方法]

前述の周辺プログラムを介して通信を行います。このとき、サブルーチン群は PUBLIC 宣言されているので、次のように EXTRN 宣言すればリンクできます。

```
EXTRN               INIT_BUFF               ;バッファ初期化
EXTRN               INIT_RXBUFF             ;受信バッファ初期化
EXTRN               CHECK_TXBUFF            ;送信バッファ確認
EXTRN               CHECK_RXBUFF            ;受信バッファ確認
EXTRN               ADD_DATA                ;送信データをバッファに
EXTRN               SET_DATA                ;送信バッファを設定
EXTRN               GET_DATA                ;受信データ読み出し
EXTRN               SET_RXBUFF              ;受信バッファ設定
```

1.1 バイト送信の使用例

(1)プログラム例

1 バイトのデータを送信する例を紹介します。最初にサブルーチン INIT_BUFF を用いて、バッファ領域の初期化を行います。次にサブルーチン ADD_DATA を用いて、データ(55H)をバッファにセットします。これで送信準備完了で、割り込みを許可して INTP0 割り込みを待ちます。送信完了は、サブルーチン CHECK_TXBUFF で確認します。送信と同時に受信するので、そのデータを読み出しておきます。

```
;
;       送信例 1
;
;   標準の 16 バイトのバッファを使用して 1 バイト単位で送信
;
    CALL        !INIT_BUFF          ;バッファ初期化
    MOV         X,#055H
    CALL        !ADD_DATA           ;送信データをセット
    EI
TXLOOP1:
    CALL        !CHECK_TXBUFF       ;送信バッファを確認
    BC          $TXLOOP1            ;バッファが空になるまで待つ
;
;   同時に受信もしているので、受信データを読み出す
;
    CALL        !GET_DATA           ;受信データ読み出し
```

(2) SM+での 1 バイト送信時のタイミング検証

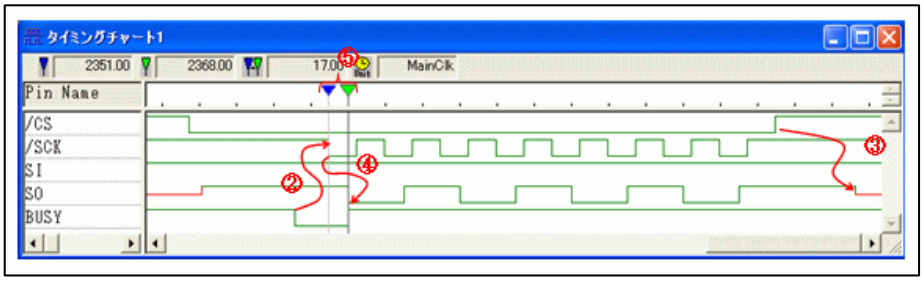
プログラムを動作させて、タイミングを確認します。外部からの信号をシミュレートするため「信号データ・エディタ」を利用して、$\overline{CS}$ などの入力信号のタイミングを定義します。また、時間の単位として μs を選択します。リセット解除後からのデータを準備するのは大変なので、上記プログラムの EI 命令にブレーク・ポイントを設定して、プログラムの実行をここで止めた状態で、信号データ・エディタからの入力を開始するようにします。このときのデータの先頭部分を次に示します。$\overline{CS}$ 信号を入力して(次図の①)15 μs 後に、$\overline{SCK}$ 信号を立ち下げて通信を開始し、以降は 3 μs ごとに $\overline{SCK}$ 信号を 8 回反転させます(②)。これは、周期が 6 μs(約 170kbps)に相当します。その後、$\overline{CS}$ 信号を立ち上げます(③)。

78K0S_SPI_SLAVE0.wvi

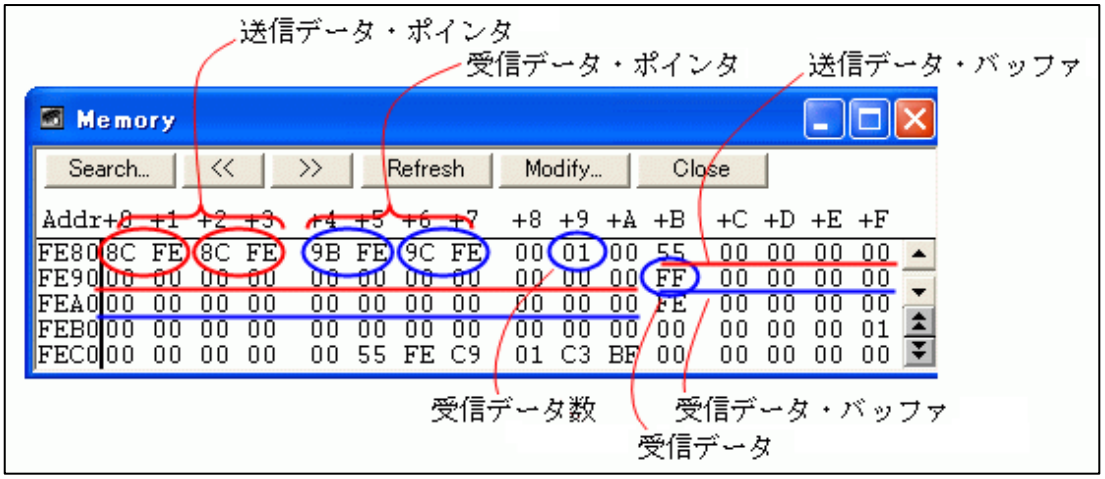
Unit: usec

	Mark	Wait	/CS	/SCK	SI
1		0	1	1	1
① 2		10	0	1	1
② 3		12	0	1	1
4	8	3	0	0	1
5		3	0	1	1
③ 6		3	1	1	1
7		30	0	1	1
8		10	0	1	1
9	8	3	0	0	1
10		3	0	1	1
11		12	0	1	1
12	8	3	0	0	1

BUSY 信号が立ち下がってから、$\overline{SCK}$ 信号が立ち下がります（②）。それを確認して、BUSY 信号が立ち上がると同時に、送信データ 55H の MSB が 0 なので、SO 信号が立ち下がります（④）。この波形で一番タイミングの苦しいのが 3 ビット目です（⑤）。この部分の遅れは 17 クロック（約 2μs）です。8 ビットのデータの送信が完了すると $\overline{CS}$ 信号が立ち上がり、それを受けて、SO 信号が Hi-Z 状態になります（③）。

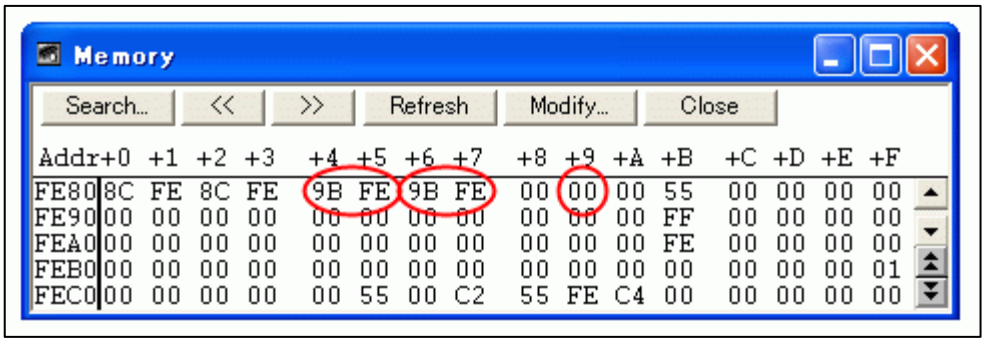


このときのメモリの状態は次のとおりです。



ここで、FE80,1 番地と FE82,3 番地が送信データ・ポインタ（TxPointer と SetPointer）です。その次に受信データ・ポインタ（RxPointer と RxSetPointer）が配置されています。FE8B～FE9A 番地が送信データ・バッファ（TxBuff）で、FE9B～FEAA 番地が受信データ・バッファ（RxBuff）になります。送信データ・バッファには、送信したデータ 55 が残っています。受信データ・バッファには、送信時に受信した無効データ FF が格納され、受信データ数（RxCount）が 1 になっています。

この無効データを読み出すことで、受信データ・バッファをクリアします（ここでは、単にデータを読み出すためにサブルーチン GET_DATA をコール）。この結果、メモリの状態は次のようになります。受信データ・ポインタ（RxPointer と RxSetPointer）が初期値に戻され、受信データ数（RxCount）が 1 になります。受信データ・バッファはリング・バッファ構造にしていないので、データがなくなったらポインタを初期化して、バッファ・オーバ・フローが発生する可能性を少なくしています。



2. 複数バイト送信の使用例

(1) プログラム例

この例では、ROM(CSEG)領域に DB 疑似命令で定義したデータを送信します。このデータが、格納領域を送信バッファとすることで、データのコピー作業を省略しています。

```
    CSEG
TXDATA:
    DB              0AAH, 0CCH, 33H, 00, 0FFH, 01, 02, 03
```

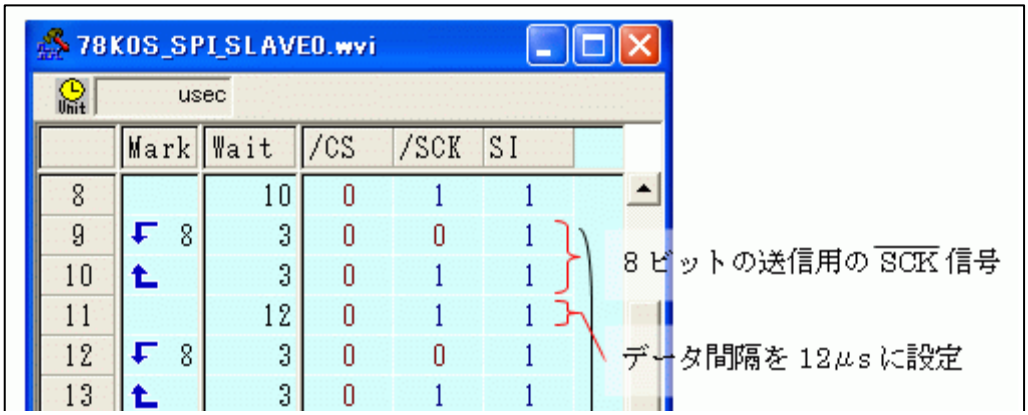
実際のプログラムでは、送信したいデータ数(この例では 8 バイト)を X レジスタに設定して、PUSH 命令でスタックにセットします。その後、データの先頭アドレス(この例では TXDATA)を AX レジスタにセットして、サブルーチン SET_DATA を呼びます。これで、送信に必要な設定が完了するので、EI 命令で割り込みを許可し、INTP0 割り込みを待ます。送信データの設定が完了したら、サブルーチン CHECK_TXBUFF を用いて、送信完了を待ちます。

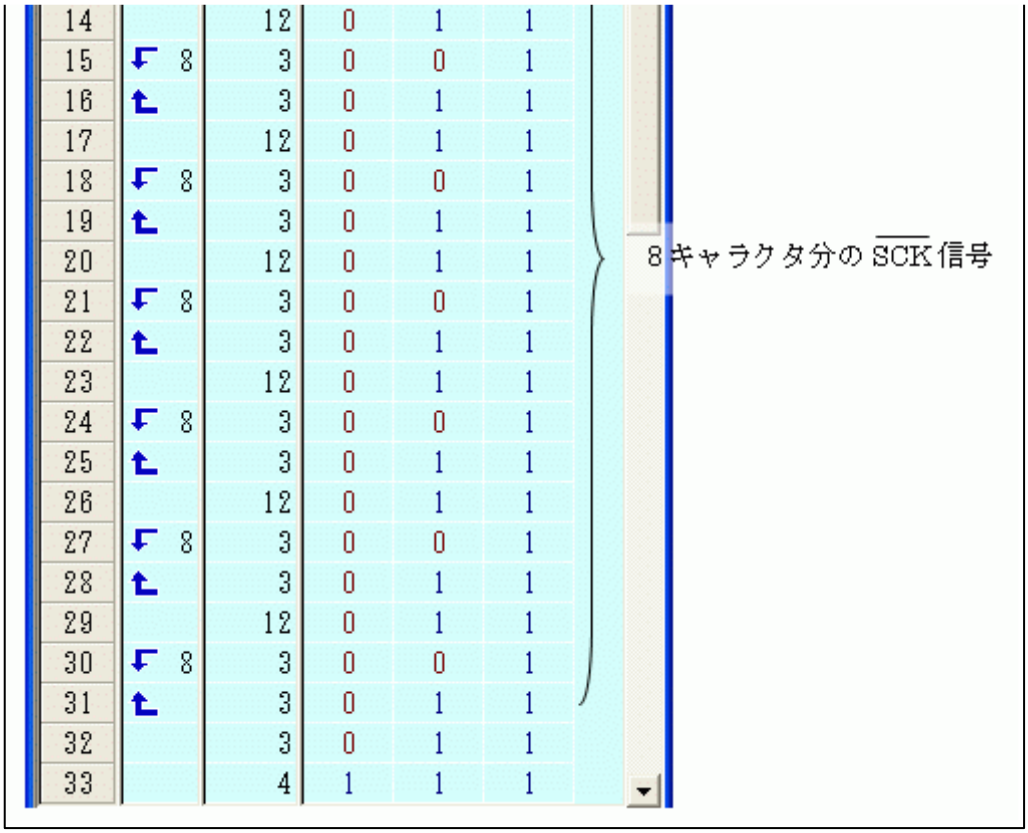
```
;
;    送信例 2
;
;   データ・テーブル TXDATA から 8 バイトを送信
;
    DI
    MOV             X, #8               ;送信データ数は 8 キャラクタ
    PUSH            AX                  ;スタックにセット
    MOVW            AX, #TXDATA         ;バッファの先頭アドレス設定
    CALL            !SET_DATA
    POP             AX                  ;スタックを元に戻す
    EI
TXLOOP2:
    CALL            !CHECK_TXBUFF       ;送信バッファを確認
    BC              $TXLOOP2            ;バッファが空になるまで待つ
```

【注意】
サブルーチン SET_DATA は、排他制御(割り込みの禁止を制御)しているので、このようにメイン処理での制御は必要ないのですが、ここでは分かりやすくするために、メイン処理でも割り込みの制御を行っています。全体の動作としては、上位側で割り込みを制御した方が、EI や DI だけで済むので、処理は単純になります。

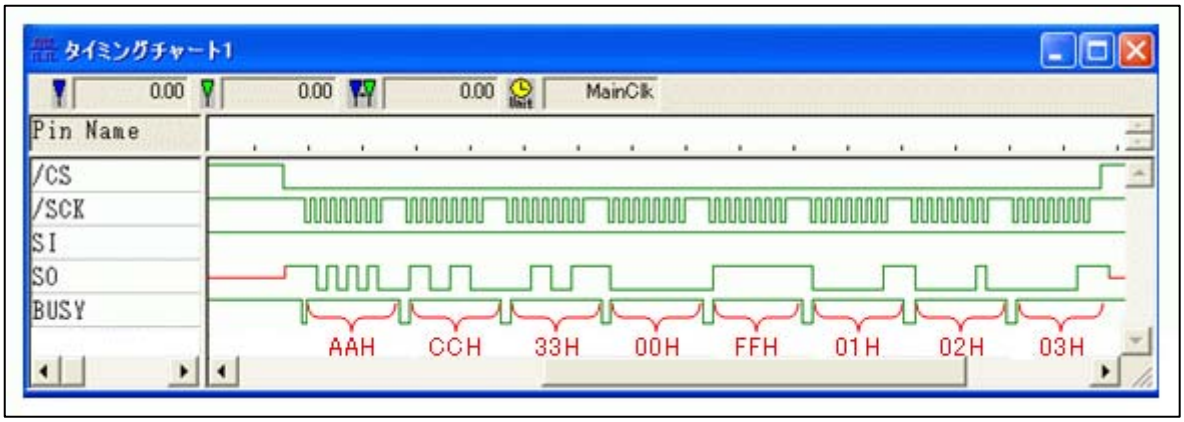
(2) SM+での複数バイト(8 バイト)送信時のタイミング検証

複数バイト送信を確認するためのタイミング・データは、次のようになります。3μs ごとに $\overline{SCK}$ 信号をロウ／ハイと 8 回切り替え、1 バイト(1 キャラクタ)の転送タイミング($\overline{SCK}$ 信号)を生成しています。1 キャラクタの最後の SCK 信号と、次のキャラクタの $\overline{SCK}$ 信号の間に 12μs の間隔を設定し、8 キャラクタ分の信号を生成します。





このときの信号波形は、次のようになります。指定されたデータが、順に SO 信号として出力されています。また、BUSY 信号も 1 キャラクタごとにハイ／ロウが変化しています。



3. 1 バイト受信の使用例

このプログラム例は、受信ステータスを確認し、受信データがあれば、バッファから読み出します。受信データが 03（ETX のコード）になるまで継続します。なお、この受信処理は、かなりの時間が必要になるため、1 キャラクタのデータをバッファから読み出している間に、複数のデータが割り込み処理で受信バッファに格納されます。

最初に、送信例 2 で送信時に受信した無効データ（受信データ･バッファ）をクリアします。それまでの受信データは消失するので、これからデータが送られてくる場合だけ、受信データ･バッファのクリア処理を行ってください。

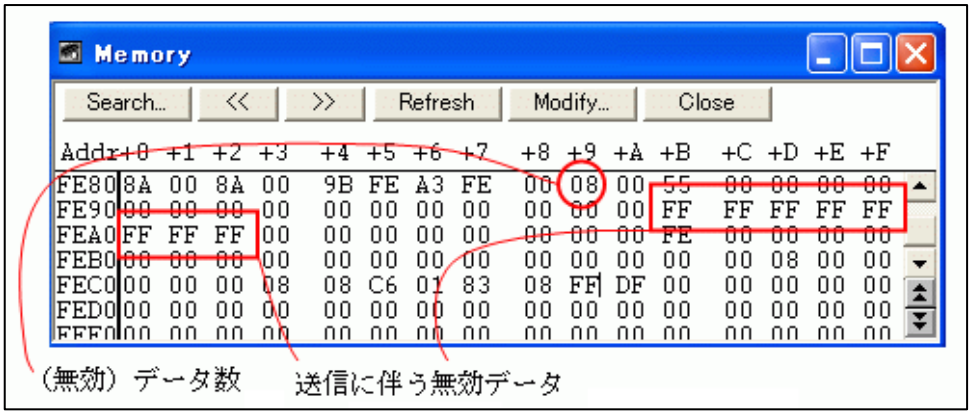
(1)プログラム例

```
;
;          受信例 1
;
;   標準のバッファに受信する。受信したデータが ETX (03) なら終了する
;
```

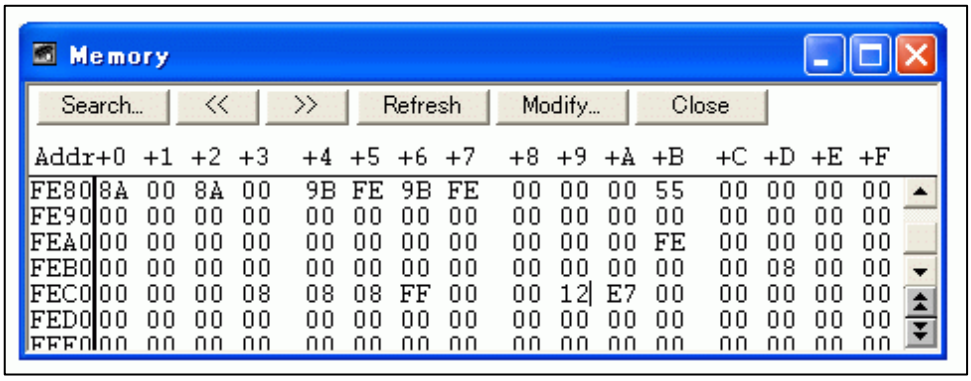
```
        CALL        !INIT_RXBUFF        ;受信バッファをクリア
        EI
RXLOOP1:
        CALL        !CHECK_RXBUFF       ;受信ステータス確認
        BC          $RXLOOP1            ;受信バッファ空ならループ
        CALL        !GET_DATA           ;受信データ読み出し
        MOV         A,#03               ;ETX
        CMP         A,C                 ;受信データを比較
        BNZ         $RXLOOP1            ;ETX を受信するまで継続
        DI
```

(2) SM+での受信時のタイミング検証

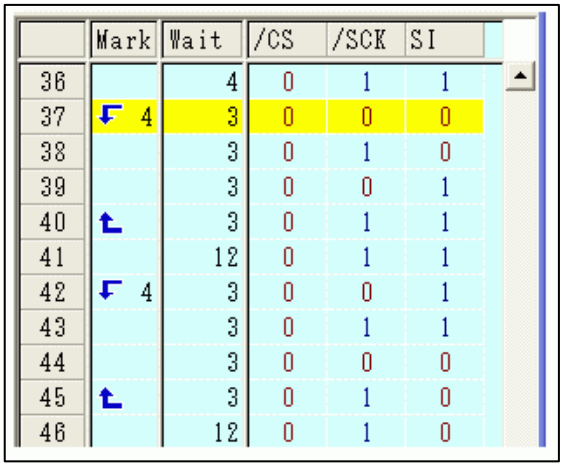
送信例 2 を実行した段階で、8 バイトの送信と同時に無効データを受信しています。この受信例が動作する段階での、メモリの状態は次のとおりです。



このままでは、無効データの処理も行うことになるので、最初に受信データ・バッファをクリアして、次のような状態にしておきます。

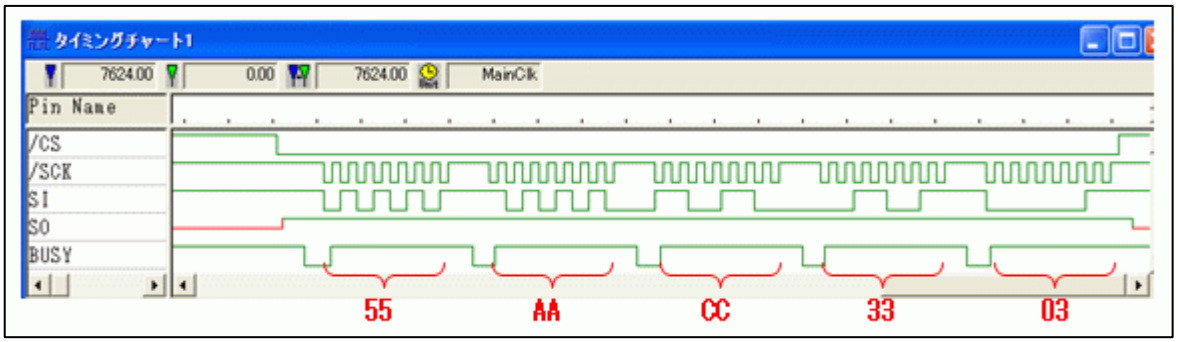


その後、次のようなパターンで信号を入力し、55H、AAH、CCH、33H、03H のデータを設定します。

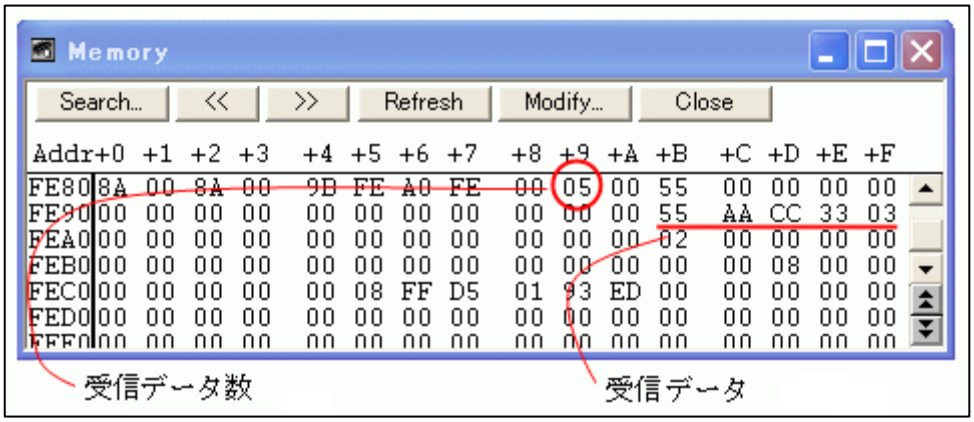


	Mark	Wait	/CS	/SCK	SI
36		4	0	1	1
37	4	3	0	0	0
38		3	0	1	0
39		3	0	0	1
40		3	0	1	1
41		12	0	1	1
42	4	3	0	0	1
43		3	0	1	1
44		3	0	0	0
45		3	0	1	0
46		12	0	1	0

このときの信号波形は、次のようになります。



また、メモリの状態は次のようになり、5 バイトのデータが受信されているのが分かります。



このように、受信は割り込みで処理され、受信データはバッファに格納されていくので、受信バッファが溢れないかぎり、受信処理を意識する必要はありません。

4. ベクタ・テーブルなどの設定

このプログラムを使用するために、次のように、ベクタ・テーブルやスタック領域を確保しておきます。

```
RSTVECT     CSEG    AT 0
            DW      START
            ORG     0AH
            DW      __INTP1
STKADR      DSEG    IHRAM
            DS      32                  ; スタック領域(32 バイト)
STACK:
```

[C言語プログラムでの使用方法]

ここでは、C 言語のプログラムから「**周辺プログラム**」で説明した処理を介して通信を行います。CC78K0S では、最初に pragma 指令で、次のように宣言します。

```
#pragma     SFR
#pragma     EI
#pragma     DI
#pragma     NOP
```

サブルーチン群(関数)は PUBLIC 宣言されているので、次のようにプロトタイプ宣言に extern 宣言すれば、リンクできます。

```
/*-------------------------------------------------------------*/
/*                                                             */
/*      使用する関数の定義                                     */
/*                                                             */
/*-------------------------------------------------------------*/

/*      バッファ初期化処理関数          */
extern  void    _init_buff(void);

/*      受信バッファ初期化処理関数      */
extern void _init_rxbuff(void);

/*      送信ステータス確認関数          */
extern  unsigned char   _check_txbuff(void);

/*      受信ステータス確認関数          */
extern  unsigned char   _check_rxbuff(void);

/*      1バイト送信関数                 */
extern  void    _add_data(unsigned char data);

/*      ブロック送信関数                */
extern void _set_data(unsigned char *data,unsigned char size);

/*      1バイト受信関数                 */
extern unsigned char    _get_data(void);

/*      受信バッファ設定関数            */
extern void _set_rxbuff(unsigned char *data);
```

1. 割り込みベクタの設定

CC78K0S の場合、スタックはスタートアップ･ルーチンが自動的に設定するので、アセンブラ記述の割り込みを使用するには、ベクタを設定するだけです。ここでは、簡素化のため、次のようなアセンブラ記述のプログラムをリンクしておきます。

```
;
;   割り込みベクタ・テーブル定義
;
EXTRN               __INTP1                         ;割り込み処理部
RSTVECT     CSEG    AT 0AH
    DW              __INTP1
    END
```

2. 1 バイト送信の使用例

使用するにはまず、_init_buff()関数を呼び出して、バッファの初期化や割り込み関係の設定を行います。

```
main(void)
{
    unsigned char   RxData;
    _init_buff();           /* バッファの初期化             */
    RxData = 0;             /* 受信データ初期化             */
```

```
/*****************************************************************

    1バイトのデータ送信

*****************************************************************/
    _add_data(0x55);          /* 1バイトのデータ(55H)送信      */
    EI();                     /* 割り込み許可                  */
/*
    送信完了確認
*/
    while(_check_txbuff() != 0){
            NOP();
    }
    DI();
```

初期化が完了したら、送信データを引数として、_add_data()関数を呼び出します。これで、送信バッファにデータがセットされるので、あとは割り込みを許可して転送可能にします。転送完了待ちのため、_check_txbuff()関数の戻り値（＝残り送信データ数）が 0 になるまで while 文で待ちます。while 文を抜けたら、送信は完了です。

送信が完了すると、同時に実行した受信処理により、無効データが取り込まれているので、必要に応じてデータの空読み処理や受信バッファの初期化処理などを行う必要があります。

3. 複数バイト送信の使用例

送信データを設定する関数が異なるだけで、全体的な処理の流れは 1 バイトの送信と同じです。使用する関数は _set_data()です。この関数では、第 1 引数で送信データの格納された配列のアドレス、第 2 引数でデータ数を示します。

```
/*****************************************************************

    ブロック（8バイト）データ送信

*****************************************************************/
    _set_data(TxData1,8);    /* 送信データのセット           */
    EI();
/*
    送信完了確認
*/
    while(_check_txbuff() != 0){
            NOP();
    }
    DI();
```

このプログラムでは、次のように定義した定数の配列 TxData1 の 8 バイトのデータを送信します。

```
    const   unsigned char   TxData1[8] =
            {0xAA,0xCC,0x33,0x00,0xFF,01,02,03};
```

ここでも、**送信と同時に受信処理を行うので、無効データが 8 バイト分取り込まれます。無効データを空読みするか、受信データ・バッファをクリアする必要があります。**

4.1 バイト受信の使用例

受信は割り込みで処理されるので、データは受信バッファに格納されます。そのため、プログラムは受信バッファの制御をするだけです。具体的には、_check_rxbuff()関数で受信バッファのデータ数をチェックし、データがあれば、そのデータを_get_data()関数で読み出します。

```
/************************************************************************

    データ受信

************************************************************************/
    EI();
    while(RxData != 03){
            while(_check_rxbuff() == 0){
            NOP();
            }
            RxData = _get_data();
    }
```

このプログラムでは、受信データが 03(ETX)になるまで、while 文で待つ処理にしています。

3 線式シリアルの最後の不要クロック [78K0/Kx]

Q1. シリアル・インタフェース CSI1n をマスタ・モードの受信で使用する際、最後のデータを受信後、SIO1n レジスタからリードすると、次の不要なクロックが出力されます。一方、先に CSIE1n ビットを 0 にして、シリアル動作を停止すると、必要な SIO1n レジスタの内容がリセットされます。SIO1n をリードしても、不要なクロックが出力されない方法はありますか。

A1. 受信モードで使用する場合、最後のデータ受信後にダミー・クロックが 1 回出力されると考えたシステム設計をお願いします。それが許容できないシステムでの根本的な対策は、送受信モードを利用することです。送受信モードで、受信起動のためにダミー・データとして、SOTB1n レジスタに 0x00 を書き込んでください。

受信だけの起動　　：SOTB1n に 0x00 を書き込む
送信か送受信の起動　：SOTB1n に送信データを書き込む

こうすれば、受信データの読み出しで SIO1n をリードしても、次の通信は起動しません。また、0x00 を書き込んで通信が起動しても、SO1n の出力はロウ・レベルのままです(外部から見ると、受信モードと同じ)。

1.2.4 タイマ／カウンタ

インバータ制御用タイマをインターバル・タイマ割り込みに使用 [μPD78098x]

Q1. μPD780984 で、10 ビット・インバータ制御用タイマをインターバル・タイマ割り込みに使用できますか。
可能な場合、TO70-75 の出力を停止する方法を教えてください。

A1. 可能です。TM7 のアンダフローごとに、INTTM7 を発生させることができます。
また、TO70-75 の出力を禁止するには、TMM7 レジスタの TOEDG ビットで有効エッジを指定して、TOSPP ビットをセットした上で、TOFF7 端子に有効エッジを入力すれば、TO70-75 を出力停止（ハイ・インピーダンス）にできます。TOSPW ビットをセットして、INTWDT を発生させても、出力停止できます。

タイマ出力 TO70～TO75 を停止 [μPD78098x]

Q1. μPD78F0988 の 10 ビット・インターバル・タイマで、TO70～TO75 出力を常時停止する方法はありますか。

A1. TMM7 レジスタの TOSPP ビットを 1 にしておき、タイマ動作後、TOFF7 端子にアクティブなエッジ(INTP0 の有効エッジを定義する EGP、EGN のビット 0 で指定)を入力してください(P00 から有効エッジを出力しても可)。

割り込みを使用しないタイマ待ち [共通]

Q1. 割り込みをしないで、タイマで一定時間だけ待つ処理は、次のような手順でいいですか。

①TMMK50=0(割り込み許可)

②タイマ関連レジスタ・セット

③タイマ・スタート

④割り込み要求 TMIF50==1 になるまで無限ループ

A1. 問題があります。タイマを割り込みで処理しない場合には、その割り込みをマスクしてください。その上で、TMIF50 がセットされるのを無限ループで待つのですが、ループに入る前に、TMIF50 をあらかじめクリアしておいてください。また、TMIF50 がセットされてループから抜け出したら、そこでも TMIF50 をクリアしてください。

Q2. 上記で、割り込み処理は必要ないので、対応する割り込み関数の定義はしなくていいですか。

A2. はい、割り込み関数の定義は必要ありません。

カウント・クロックの誤差(割り込み周期がばらつくか) [共通]

Q1. タイマなどに使用されるクロックは、どの程度の誤差が生じますか。

A1. fxx を元に、内部で生成されるクロックは、計算値に対して誤差を生じません。誤差があるとすれば、元クロックの立ち上がり／立ち下がり時間が長い場合に、いつスレッシュホールド・レベルを越えるかのばらつきですが、周波数誤差の原因にはなりません。むしろ、発振クロック(メイン・システム・クロック)の変動の影響が大きいと思われます。

なお、クロックの誤差ではなく、タイマ割り込みタイミングはばらつきます。これは、実行中の命令によって、割り込み発生から処理開始までが、最低 7 クロックから最大 32 クロックと変動します(もちろん他の割り込みとの関係で、割り込み禁止状態があれば、さらに長くなります)。

タイマの出力を別のタイマに入力して割り込みを発生させる制御 [共通]

Q1. μPD784215 のインターバル・タイマ(INTTM3)を動作させ、それを 250ms カウントして、INTTM00 を発生させます。INTTM0 待ちの HALT モードに移行する前に許可する割り込みは、TMMK0 だけでいいですか。TMMK3 も許可しないと、カウント動作しないのですか。また、TMIF3 は毎回クリアしなければならないのですか。

A1. TMMK3 を許可状態にする必要はありません。この場合、INTTM3 は単にタイマ 0 のカウント・クロックに使用しているだけで、これで割り込みを発生させる必要はありません。TMMK3 は、マスクをセットしてください。

また、マスクされていれば、TMIF3 は他に影響しません。この場合、INTTM3 の割り込みが受け付けられるたびに、TMIF3 はクリアされますので、操作する必要はありません。

キャプチャ機能の制限事項 [μPD789407A、789417A サブシリーズ]

Q1. μPD789407 のマニュアルの TMC50 の説明で、次の記述があります。
カウント・クロックを fx に設定した場合、キャプチャ機能は使用できません。
読み出す場合は、CPU クロックをメイン・システム・クロックの高速に設定してください。
ここで、「読み出す場合」とは、インプット・キャプチャのことですか。

A1. いいえ、ちがいます。これは、16 ビット・カウンタ・リード・バッファを介した、タイマ・レジスタ 50(TM50)の読み出しの説明です。前文は、キャプチャ機能に対する制限です。この場合、キャプチャ機能は使えないので、読み出しはタイマ・レジスタ 50 に対して行うことになります。後文は、読み出しの制限です。

カウンタ初期値が 1 になってしまう [共通]

Q1. μPD78F9026A の 8 ビット・タイマを外部イベント・カウンタとして、立ち上がりエッジでカウント動作する場合、外部入力レベルをハイに固定してあると、動作許可(TCE00＝1)に設定直後、TM00 の値が 1 になります。

A1. 外部イベント・カウントで、外部信号がアクティブなエッジの極性の状態でタイマを起動すると、有効なエッジと見なして 1 カウントします。これを避けるには、TI0 端子をアクティブなエッジと異なる極性にしてください。

PPG 出力の ON と OFF の時間が反転する [78K0]

Q1. μPD78F0034A で、周波数一定でデューティを変更したいのですが、PPG 出力で波形確認すると、ON と OFF の時間が反転するときがあります。

A1. CR01 への書き込みタイミングの問題と思われます。タイマ 0 の出力は、CR00 か CR01 との一致で反転します。CR01 と一致したあとで、CR01 を大きな値に書き換えると、1 周期の中で再度 CR01 との一致が発生し、出力が反転して元に戻ることが考えられます。
また、タイマの値が CR01 より小さいときに、そのタイマ値より小さい値を書き込むと、その周期では CR01 との一致が発生せず、CR00 との一致が連続して、結果的に出力状態が反転することが考えられます。
次の「PPG 出力が出ない」もご参照ください。

PPG 出力が出ない [78K0]

Q1. 安定した PPG 出力が出ません。PPG 出力設定以外に、注意点はありますか。

A1. CR01 の書き換えタイミングについて、注意する必要があります。このため、次のような処理が必要です。
(1) CR00 との一致割り込み(周期の切り替わりタイミング)で、CR01 の書き換えを行う。
(2) CR01 の新しい設定値と従来の値を比較し、値を小さくする場合は CR01 との一致割り込みで書き換え、大きくする場合は CR00 との一致割り込みで書き換える。
(1)の方が簡単ですが、CR01 への書き込み値が小さい場合に、タイマの動作クロックが高いと、 CR01 書き込みまでにタイマがカウント・アップしてしまい、間に合わない可能性があります。
(2)ではそのようなことはありませんが、処理が複雑になります。
タイマのカウント・クロック周波数と割り込み時間、CR01 の最小値の関係でご判断ください。

関連 FAQ：16 ビット・タイマの PPG 出力を使った PWM 出力 [78K0、小ピン共通]

16 ビット・タイマでのインターバル [78K0S]

Q1. 16 ビット・タイマでコンペア・レジスタに値を設定しても、期待したインターバルが得られません。

A1. コンペア・レジスタに値をセットしたら、その値にインターバル分のカウント値を加算した値をメモリに保存してください。一致割り込みが発生したら、メモリに保存した値をコンペア・レジスタに書き込み、インターバル分のカウント値を加算して、またメモリに保存してください。
これを繰り返すことにより、一定のインターバルの信号を得られます。
なお、コンペア・レジスタに値を書き込む際には、マニュアル等に記載された条件に従って行ってください。

<参考>
16 ビット・タイマは、フリーランのカウンタです。したがって、コンペア・レジスタに一度値を設定した場合、その値とタイマ値が比較され、一致すると一致信号が出力されますが、タイマは動作を続けます。
次に一致するのは、タイマが一回りしたあとになりますので、以降のインターバルは、16 ビット分のカウントごとになります。
これを避けるには、上記のようにコンペア・レジスタにセットする値をダイナミックに更新する必要があります。

16 ビット・タイマのオーバフローでの割り込み [ほとんどの 78K0、78K0S 共通]

Q1. 78K0/KB1 の TM00 をフリーランで使用すると、カウント値オーバフローの割り込みは発生しませんか。

A1. はい、オーバフローではフラグがセットされるだけで、割り込みは発生しません。

Q2. オーバフロー割り込みを発生させるには、どうすればいいですか。

A2. コンペア・レジスタ CR010 に 0FFFFH をセットすれば、INTTM010 をオーバフロー割り込みの代わりに使用できます。
なお、フリーランで使用されていて、すでにコンペア・レジスタとの一致割り込みを利用されている場合には、その割り込み処理の中で、オーバフロー・フラグをご確認ください。

16 ビット・タイマの読み出し [78K0]

Q1. μPD780078 の 16 ビット・タイマ TM0 で、「カウント値を読み出し時にカウント・クロックを一時停止」とありますが、カウント値の読み出しでカウントがずれるのですか。

A1. いいえ、読み出しを行っても、カウントはずれません。これは、単に競合時に読み出し動作が優先されることを示しているだけです。

8 ビット・タイマでの PWM 出力 [78K0]

Q1. 78K0/KC1 のタイマ H0,H1 で PWM 出力を行いたいのですが、CMP00 と CMP01 を同じ値に設定すれば、100%出力は可能ですか。

A1. いいえ、それはできません。CMP00 と CMP01 に設定する値は、CMP01＜CMP00 である必要があり、同じ値は設定できません。

タイマの出力周波数が期待どおりにならない [共通]

Q1. μPD780148 を 8.192MHz で動作させ、8 ビット・タイマ(TM51)で 2kHz の方形波出力を考えています。
8.192MHz を 2kHz で除算すると 4096 なので、TCL51 に 07 を設定して、タイマの入力クロックとして fx/4096 を選択し、CR51 に 1 を設定すると、出力は 500Hz になります。

A1. 2 つの点で誤解があります。

(1) タイマ(TM51)は 0～CR51 の設定値までカウントするので、CR51 にはカウントさせたい期間のカウント数－1 の値を設定する必要があります。

(2) タイマでカウントするのは、ハイ・レベルかロウ・レベルの期間です。出力の周期は、タイマのカウント時間の 2 倍になります。

設定されている内容では、2kHz の入力クロックの 2 周期分(CR51 の設定値＋1)カウントして出力が反転するので、出力の 1 周期は入力クロックの 4 周期分で、500Hz になります。

2kHz の出力にするには、最低でもその 4 倍以上の周波数の入力クロックを選択してください。TM51 では、fx/256 の 32kHz を選択してください。2kHz はこの 16 クロック分の周期なので、ハイ・レベルとロウ・レベルの期間は、それぞれ 8 クロック分になります。

TM51 で 8 クロック分の時間をカウントするには、CR51 に 7(＝8—1)を設定してください。

1.2.5 ウォッチドッグ・タイマ

ウォッチドッグ・タイマのカウント・クロック [共通]

Q1. ウォッチドッグ・タイマのカウント・クロックは、発振子のクロックがそのまま入るのですか。

A1. いいえ、ちがいます。ウォッチドッグ・タイマの動作については、各デバイスのマニュアルをご参照ください。

<備考>

ウォッチドッグ・タイマのカウント・クロックは、通常はメイン・システム・クロック(fx)を分周したものです。

例 μPD780034A サブシリーズでは、fx/256
μPD784216A サブシリーズでは、fx/128 かサブシステム・クロック
78K0/Kx1 では、X1 発振クロックを 1/16 に分周したものか、内部発振器の 1/4 分周

タイマ割り込み処理でウォッチドッグ・タイマをクリア [共通]

Q1. CPU の暴走対策として、ウォッチドッグ・タイマを使用します。プログラムの簡略化のため、定期的にかかるタイマ割り込みの中で、ウォッチドッグ・タイマをクリアしていいですか。

A1. タイマ割り込みで、ウォッチドッグ・タイマをクリアするのはお勧めできません。これでは、メイン・ルーチンが暴走しても、タイマ割り込みが受け付けられれば、ウォッチドッグ・タイマはクリアされてしまい、暴走を検出できません。より厳密な暴走検出を行うには、一定時間内に必ず処理しないといけない部分に、個別にフラグを準備しておき、その処理を行ったらそのフラグをセットします。

タイマ割り込みのように、定期的に動作するプログラムでフラグを確認して、必要なフラグがすべてセットされていたら、そこでウォッチドッグ・タイマをクリアするようなことを考えてください。

ウォッチドッグ・タイマのオーバフロー時にリセットした場合の復帰時間 [78K0S、78K0]

Q1. μPD78F9116A のウォッチドッグ・タイマを使用して、オーバフロー時にリセット信号を発生させた場合、 CPU が再度動き出すには、どのくらいの時間がかかりますか。

A1. CPU が動作を再開するまでには、次のような時間が必要となります。

内部のリセット時間

実際にクロックが発振するまでの時間

発振安定待ち時間

μPD78F9116A の場合、**内部のリセット時間**は約 10μs で、**発振安定待ち時間**は 6.55ms となります。

なお、78K0/Kx1 では、リセット解除後は内部発振器出力の 17 クロック後に、動作を再開します。

プログラムでウォッチドッグ・タイマを動作停止／開始できるか [共通]

Q1. ウォッチドッグ・タイマは、プログラム実行中に、1 秒タイマと同様に動作停止・開始できますか。

A1. いいえ、ウォッチドッグ・タイマを起動すると、ソフトで停止できません。このため、ソフト監視に使用できます。

備考 78K0/Kx1 では、リセット解除時にウォッチドッグ・タイマは動作しています。
内部発振器が停止可能の場合に限り、止めることが可能です(再開は不可)。

STOP モード解除時のウォッチドッグ・タイマの動作 [共通]

Q1. STOP モードでは、ウォッチドッグ・タイマを停止しますが、STOP 状態を解除するとどうなりますか。

A1. STOP モードではクロックがなくなるので、タイマは停止しています。
STOP モードが解除されたときには、ウォッチドッグ・タイマはクリアされた状態から再スタートします。

備考 78K0/Kx1 では、STOP モード中は停止し、解除後には継続してカウントします(クリアされません)。

ウォッチドッグ割り込みからの初期化 [78K0]

Q1. ウォッチドッグ割り込みでリスタートをしていますが、リスタート後に割り込み要求を受け付けません。
PSW もマスク・レジスタも確認しましたが、割り込みは許可状態になっています。

A1. ウォッチドッグ割り込にノンマスカブル割り込みを使用されていませんか。この場合、リスタートすると、その後の動作はノンマスカブル割り込み処理の実行となります。このため、割り込み要求を受け付けません。
これを回避するには、必ず RETI 命令を実行して、ノンマスカブル割り込み状態から抜ける必要があります。
具体的な方法としては、PSW と分岐先アドレスをスタックに積み、RETI 命令で分岐先アドレスに分岐します。
次の例では、0 番地からベクタを読んで、そこに分岐しています。

```
MOV     PSW,#02H  ; ダミーの PSW をセット
MOVW    AX,!0     ; リセットのベクタ値を取り込み
PUSH    PSW
PUSH    AX        ; RETI のためのスタックを準備
RETI              ; リセット・ベクタに分岐
```

ウォッチドッグ・タイマがオーバフローしたときのモード [共通]

Q1. μPD784218AGF のウォッチドッグ・タイマは、オーバフロー割り込みで停止(RUN ビットが 0)しますか。

A1. いいえ、ウォッチドッグ・タイマが停止するのは、リセットがかかったときだけです。

ウォッチドッグ・タイマの使い方 [78K0/Kx2]

[概要]

78K0/Kx2 には、従来のウォッチドッグ・タイマを機能強化したものが搭載されています。そのうちで、次の機能は全く新規のもので、ウォッチドッグ・タイマとして特に設定をしなくても動作します。

・命令をフェッチできないアドレスからフェッチした場合にリセット発生

・データをアクセスできないアドレスをアクセスした場合にリセット発生

これ以外に、次のような特徴があります。

・特定の期間以外で、ウォッチドッグ・タイマをクリアしようとするとリセット発生(ウィンドウ機能)

ウォッチドッグ・タイマの基本的な内容については、「1.1.12 ウォッチドッグ・タイマ」をご参照ください。

[使い方の基本]

プログラムが正しく動作しているかどうかを確実に把握することは困難です。ウォッチドッグ・タイマを使う場合にも、「どうなっていれば正しく動作しているか」を最初に定義することが必須です。

このためには、まずプログラムが正常な動作をしているときにどのような処理フローで動作しているかを把握する必要があります。いろんな場合分けが存在するような場合には大変かもしれませんが、これがきちんと把握できないと、仕様の追加が必要になったときや評価項目設定等が大変になります。通常、プログラムを開発する場合には、ここまでは問題なくできるはずです。逆に、この段階で引っかかってしまうようだと、プログラムの開発方法に問題が考えられ、プログラムが完成しても、デバッグ段階でトラブルが多発する可能性があります。

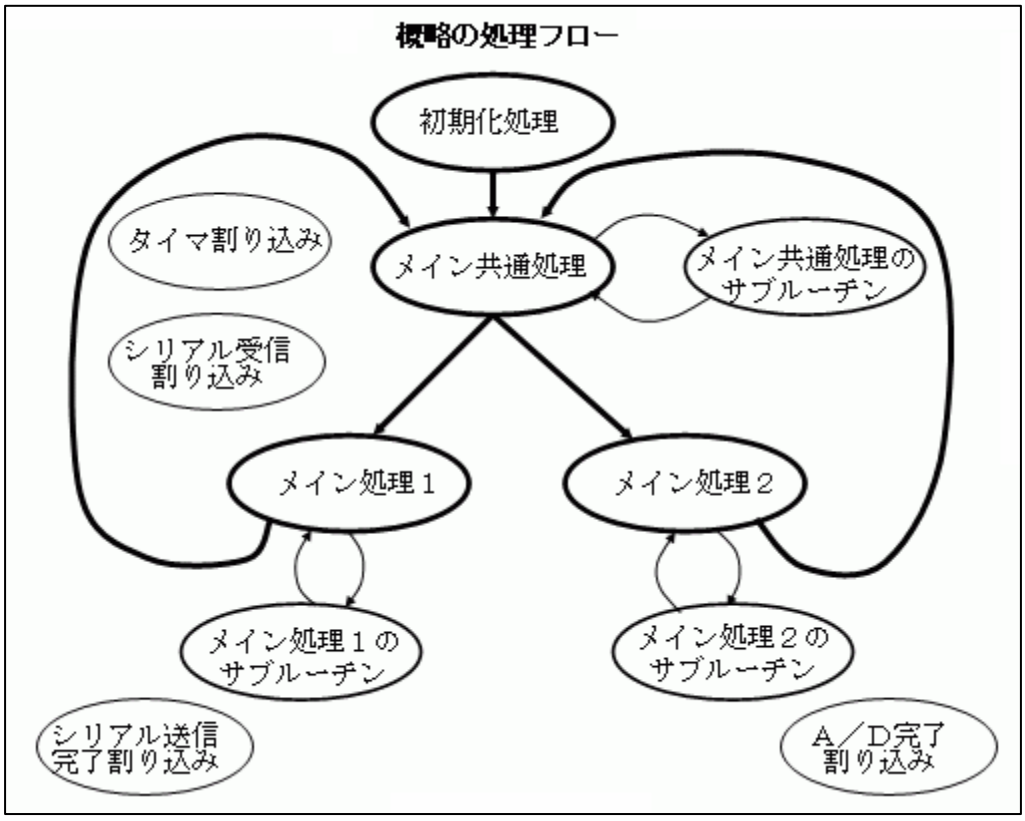
次に、正常な動作時の処理時間やタイミングの見積もりを行います。特に、非同期に発生する割り込み処理がある場合には、全体の処理時間が変動しますので注意が必要です。その比率が大きい場合には、ウォッチドッグ・タイマのウィンドウ機能は使えないと考えた方がいいでしょう。

[処理時間の解析]

以下に、処理時間解析の例を示します。

処理内容のリスト

処理番号	処理内容	処理時間
1	初期化処理	100μs
2	メイン共通処理	1ms／100μs
3	メイン処理1	20ms
4	メイン処理2	30ms
5	共通のサブルーチン	100μs
6	メイン処理1のサブルーチン	200μs
7	メイン処理2のサブルーチン	300μs
8	シリアル受信割り込み	10μs
9	シリアル送信完了割り込み	5μs
10	タイマ割り込み(1ms 周期)	50μs
11	A/D完了割り込み	5μs



初期化が完了したら、メイン共通処理を行い、そこでタイマを起動して、その割り込みを許可しておきます。また、シリアル通信の割り込みを許可しておきます。

メイン共通処理は、その後の処理要求を待つ部分です。通常はタイマ割り込みにより起動されて、その処理後、メイン処理 1 の処理要求確認、メイン処理 2 の起動タイミング確認を行い、何も処理要求がなければ、100μs の処理後に HALT モードに入って、次のタイマ割り込み待ちになるものとします。処理要求はシリアル通信により送られてくるものとし、シリアル受信割り込みでは、送られてきたデータをリング・バッファに保存するだけとします。シリアル通信では、最短で 520μs 間隔（19.2kbps 相当）に受信割り込みが発生することがあり、メイン共通処理では、HALT モード中にこの割り込みで HALT が解除された場合、再度 HALT に入るものとします。タイマ割り込みでの起動により処理要求が認識され、その解析を行い、1ms の処理をして、メイン処理 1 かメイン処理 2 に分岐します。処理要求の解析では、サブルーチンを 2 回コールするものとします。

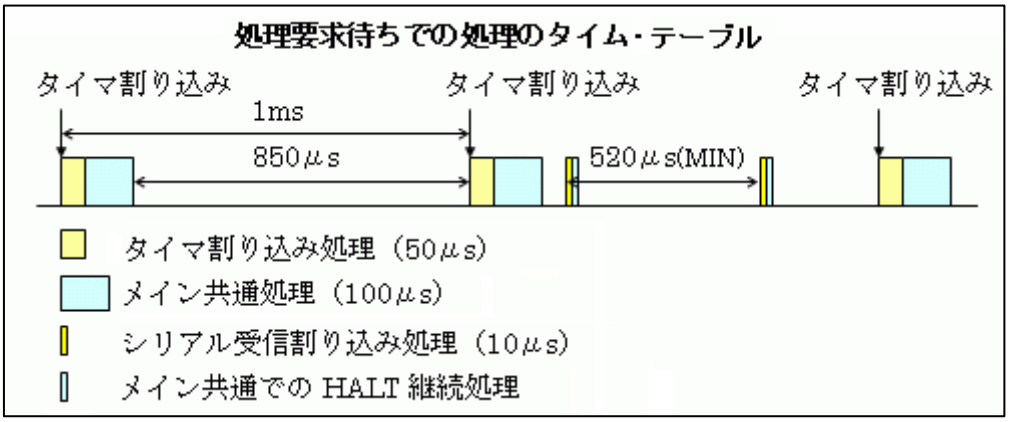
メイン処理 1 では必要な処理を行い、その結果のステータス 10 バイト分をシリアル通信によって送信するものとします。メイン処理 1 ではサブルーチンを 10 回コールし、シリアル送信は割り込みを使用することで、メイン処理 1 と並行して実行できるものとします。

メイン処理 2 は一定の周期で起動され、8 チャネルのアナログ信号を読み込んで、A/D 変換を処理するために、サブルーチンを 8 回コールします。サブルーチンでは、A/D の割り込み処理時間を含めて、300μs の処理時間がかかるものとします。また、このサブルーチン以外では、A/D コンバータは動作しないものとします。

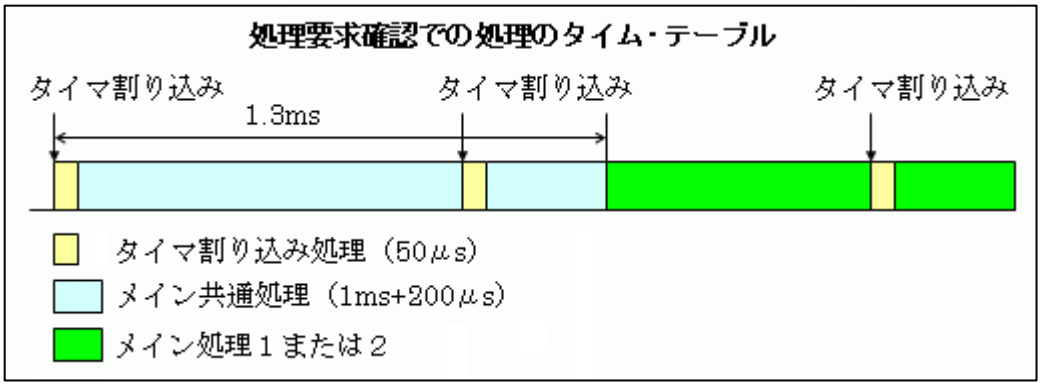
このような場合に、動作としては大きく次の 3 つに分けられます。これらは排他的に動作するものとします。

(1) 処理要求待ち

メイン共通処理での処理要求待ちで、1ms のタイマ割り込みで起動されます。タイマ割り込み処理を含め、150μs の期間に処理を行い、残りの期間は HALT モードで停止します。



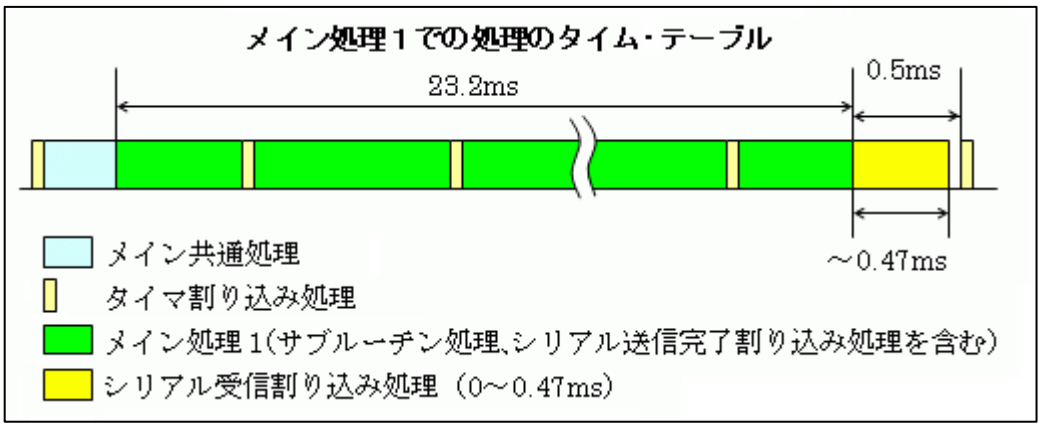
処理要求があった場合、タイマ割り込みで起動されたメイン共通処理では、1ms とサブルーチンでの処理（100μs が 2 回）、次のタイマ割り込み処理の 50μs が必要になります。これは、タイマ割り込みの頭からみると、1.3ms となります。シリアル受信割り込みは、この間に 0 回～最大で 3 回入る可能性がありますので、これを含めると最大では 1.33ms かかることになります。シリアルの処理時間比率は小さいので、それを無視した場合の処理要求確認から、実際の処理がスタートするまでのタイム・テーブルは次のようになります。



(2) メイン処理 1 の実行

メイン処理 1 では、その処理時間 20ms とサブルーチンの処理時間（200μs が 10 回）、シリアル送信完了割り込み処理（5μs が 10 回）で、合計 22.05ms の処理時間となります。この間に、タイマ割り込みは 22 回発生しますので、その処理時間である 1.1ms だけ処理が長くなります。この 1.1ms の間にも、さらにもう 1 回のタイマ割り込みが発生しますから、合計では 23.2ms かかることになります。

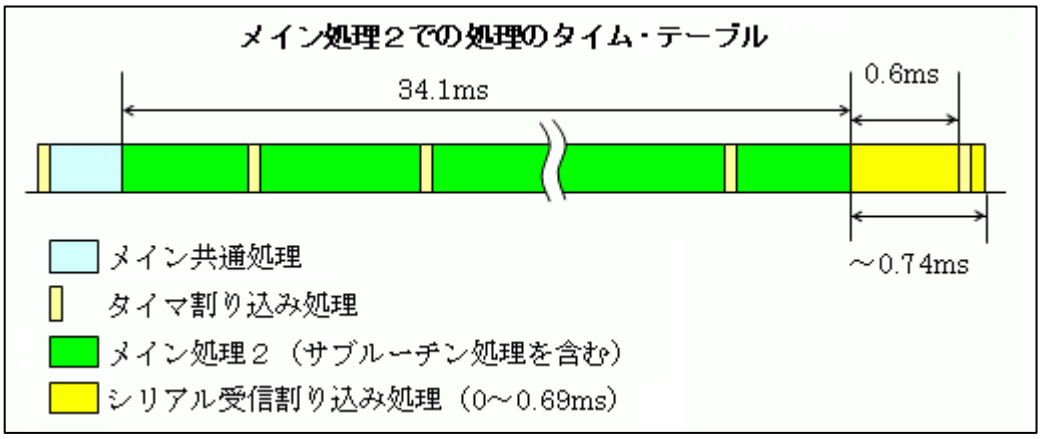
処理要求確認のタイマ割り込みからは、24.5ms で完了することになります。もし、この期間にもシリアル受信割り込みが発生していたとすると、最大で 47 回のシリアル受信割り込みが発生することになり、その処理時間は 0.47ms で、それでも次のタイマ割り込みまでには完了できることになります。



(3) メイン処理 2 の実行

メイン処理 2 では、その処理時間 30ms とサブルーチンの処理時間（300μs が 8 回）で、32.4ms の処理時間となります。この間にタイマ割り込みが 32 回発生しますが、それにより実行時間が 1.6ms 長くなるので、さらに 2 回のタイマ割り込みが発生し、全体では 34.1ms で処理が完了となります。

処理要求確認のタイマ割り込みからは、35.4ms で完了することになります。もし、この期間にもシリアル受信割り込みが発生していたとすると、最大で 69 回のシリアル受信割り込みが発生することになり、その処理時間は 0.69ms となります。シリアル受信処理に 0.6ms 以上かかると、次のタイマ割り込みも発生するので、最悪では 36.14ms となります。



以上をまとめると、3 つの場合について、以下の実行時間となります。

処理要求待ちは、1ms ごとに 150～160μs

メイン処理 1 の実行は、24.5～24.97ms

メイン処理 2 の実行は、35.4～36.14ms

ここまでは、各処理における実行時間の見積もりです。個々の処理時間が与えられているものとして計算しましたが、実際に命令の実行時間から処理時間を求めるのは困難です。そのため、デバッグを進める中で、インサーキット・エミュレータの時間計測機能を用いて、処理時間を求めることになります。ここで注意する必要があるのは、「測定できるのは、ある特定パターンで動作している状態での処理時間」ということです。それが、どのような動作であり、同期／非同期で発生してくる割り込みがどのように影響しているかを確認しておく必要もあります。その上で、必要と思われる変動要素を加味した処理時間を求めてください。

これは、ウォッチドッグ・タイマの検討に必要な情報の一部ですが、むしろ処理性能の評価です。場合によって、システムの要求仕様に、メイン処理 1 は 20～30ms で完了し、メイン処理 2 は 25～50ms で処理完了するといった条件が与えられることもあります。しかし、このときの値はほとんどの場合にかなりの幅をもちますし、最大値しか示されない場合もあり、ウォッチドッグ・タイマの検討ではあまり使えない可能性があります。

また、ウォッチドッグ・タイマの動作の基準クロックにはばらつきが存在します。そのばらつき以上の精度で処理時間を求めても、通常はあまり意味はありません。せいぜい 10%程度の精度で求めれば十分です（この例のように比較的単純な場合はいいのですが、複雑な処理では厳密に求めるのは大変です）。処理時間の精度が要求されるのはウォッチドッグ・タイマのウィンドウ機能を用いるときです。もっとも厳しい条件の場合には、約 8%しかクリアできる期間がありませんので、4%以下の誤差で処理時間を求める必要があります。

[動作パターンと検出条件]

実際の検討を進めるには、メイン処理 1,2 に対する要求がどのような頻度やタイミングで発生するかを明確にする必要があります。その上で、どのような条件で検出を行うかを決めます。

たとえば、次のような要求条件で検討してみます。

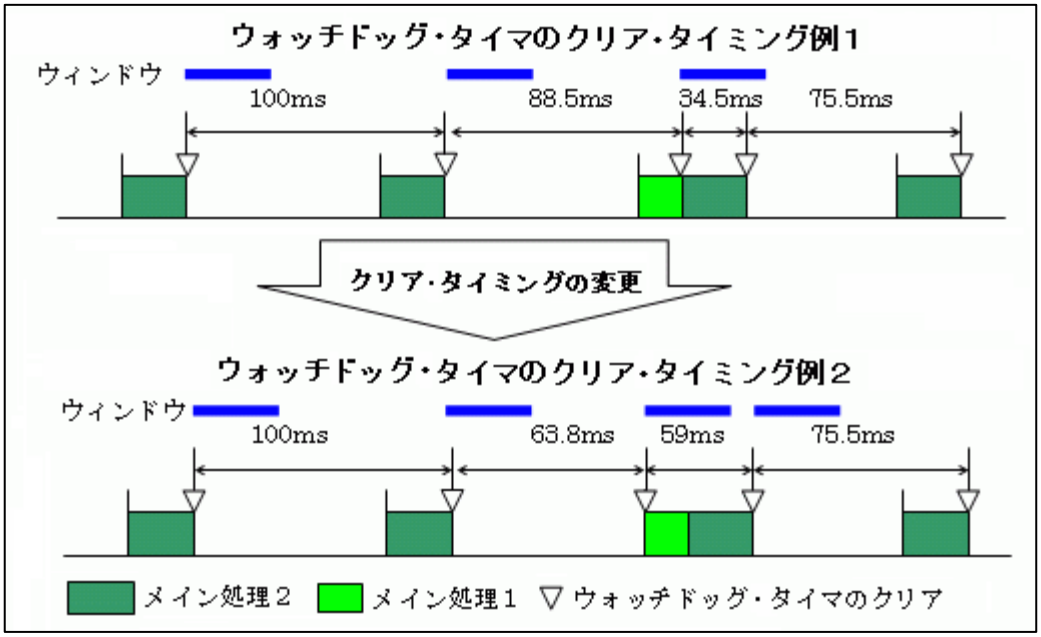
<例 1>

・100ms に 1 回、メイン処理 2 が起動される。
・1 時間に1回、メイン処理 1 が 200ms ごとに 20 回要求される。
・メイン処理1実行は、メイン処理 2 より優先。

この場合、どのような状態を正常とするかを規定する必要があります。ここでは、次の 3 種類の基準で考えます。

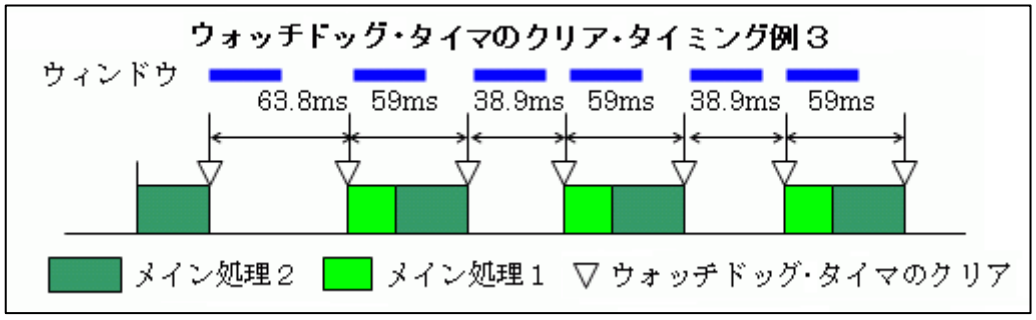
(1) メイン処理 2 やメイン処理 1 を実行すれば正常と判断
(2) メイン処理 2 やメイン処理 1 を定められた時間以内で実行すれば正常と判断
(3) メイン処理 2 やメイン処理 1 を定められた時間範囲で実行すれば正常と判断

(1)の場合、メイン処理 2 やメイン処理 1 だけで、ウォッチドッグ・タイマをクリアすればいいことになります。処理のループとしては最低が 100ms ですから、ウォッチドッグ・タイマのオーバフローまでの時間をこれ以上に設定して、メイン処理 1 とメイン処理 2 で、ウォッチドッグ・タイマをクリアすればよいことになります。78K0/Kx2 では、この条件を満たすオーバフロー時間は $2^{15}/f_{RL}$(min.124.12ms)、$2^{16}/f_{RL}$(min.248.24ms)、$2^{17}/f_{RL}$(min.489.48ms)です。
さらに、(2)の条件を加味すると、$2^{15}/f_{RL}$(min.124.12ms)を選択することになります。この場合、メイン処理 1 とメイン処理2が競合して、メイン処理 2 の実行が遅らされた場合には、メイン処理2だけでクリアしていると、ここの間隔はメイン処理 1 の分だけ伸びてしまい、124.5ms 以上になってオーバフローが発生するので、メイン処理 1 でのクリアも必要になることがわかります。
次に、(3)の例として、ウォッチドッグ・タイマのウィンドウ機能を使う場合について考えてみます。メイン処理 1 とメイン処理 2 の最後でクリアするものとした場合の、最短のクリア・タイミングは次図の例 1 のようになります。ウォッチドッグ・タイマのクロックが遅いほうにずれた場合、オーバフロー時間は 151.7ms となり、最もウィンドウを開いた場合でも、オーバフロー時間の 25%の期間(約 38ms)はウィンドウが閉じています。次図の例 1 では、最短で 34.5ms 間隔となるので、このままでは、ウィンドウが閉じているときにクリアをすることになります。そこで、メイン処理 1 でのクリア・タイミングを処理の頭に変更すると、最短の間隔でも 59ms となるので、クリア・タイミングはすべてウィンドウが開いている期間におさまります。



<例 2>

前記の例 1 では、メイン処理1が 200ms ごとに起動することにしていましたが、これを 100ms ごとに変更した場合には、どうなるでしょうか。今度は、メイン処理2でのクリアから、メイン処理 1 でのクリアまでの時間が厳しくなり、次の例 3 のように、約 38.9ms となります。



この場合には、最悪でもウィンドウに納まるので、このままのタイミングでもいいのですが、マージンを大きくしたい場合には、メイン処理 1 の頭から 10ms 程度のタイミングでクリアしてください。このように、ウィンドウ機能を用いる場合には、条件のちょっとした変更でも影響を受けます。ここで重要な点は、次のとおりです。

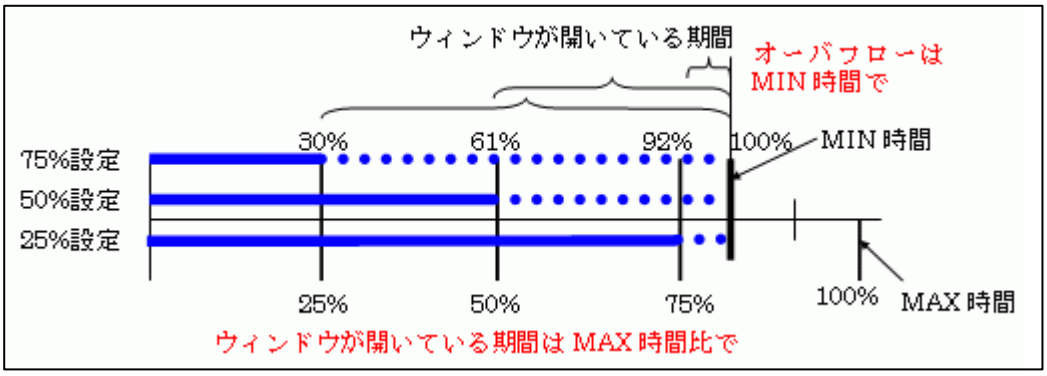
・オーバフロー時間を計算する場合には、カウント・クロックの最大周波数で計算する。

・ウィンドウ期間を見積もるときには、カウント・クロックの最小周波数を使う。

ウォッチドッグ・タイマのカウント・クロックは、規格上−10%～+10%の範囲でばらつきます。検討を簡単にするために、カウント・クロックとして最大周波数だけを用いて見積もる場合には、ウィンドウが閉じている期間は 1.22 倍して見積もる必要があります。つまり、次の期間に確実にウィンドウが開いています。

- 25%閉じる（75%開く）設定では、約 30%閉じていて、残り 70%開いている
- 50%閉じる設定では、約 61%閉じていて、残り 39%開いている
- 75%閉じる設定では、約 92%閉じていて、残り 8%開いている

これを元に、処理時間も最悪の組み合わせで考えてください。



[高度な検出条件設定]

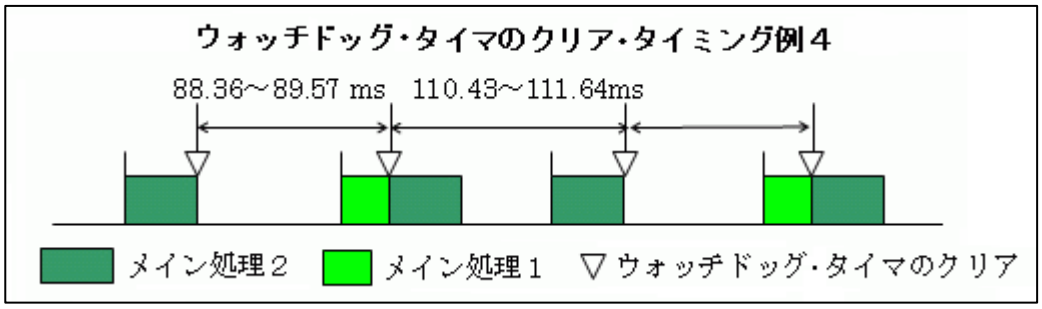
<例 3>

これまでは、単純にウォッチドッグ・タイマをクリアするタイミングだけで検討してきました。これでは、メイン処理 1 が実行されなくても、ウォッチドッグ・タイマでは検出できません（1 時間に数回しか実行されない処理を考慮することが問題ですが）。そこで、プログラムの処理の状況に応じたフラグを準備しておき、これを利用してメイン処理 1 の実行を監視することにします。

具体的には、まず次のフラグを準備します。

- メイン処理 1 間隔フラグ（カウンタ）

このフラグは、メイン処理 1 を実行してからの時間を 100ms 単位でカウントします。メイン処理 1 はこのカウンタをクリアし、メイン処理 2 でカウント処理を行います。メイン処理 2 では、このカウンタをチェックして、1 時間分のカウント以上であれば、ウォッチドッグ・タイマをクリアしないようにします（実際には、1 時間／100ms=36000 カウントから、メイン処理 1 の実行回数×繰り返しの間隔を引いた値となります。例1のような 200ms ごとでは 20×2、例2のような 100ms ごとでは 20 を 36000 から引きます）。これにより、1 時間以上メイン処理 1 が実行されなかったことを検出できます（タイミング例 2 やタイミング例 3 と同じです）。
また、メイン処理 1 では、最後に必ずウォッチドッグ・タイマをクリアするようにし、メイン処理 2 では、このフラグが 0 のときもウォッチドッグ・タイマをクリアしないようにしておくことが考えられます。この場合のクリアのタイミング例を次図に示します。このときのクリアの最大間隔は 111.64ms、最小間隔は 88.36ms となり、オーバフロー時間を $2^{15}/f_{RL}$（min.124.12ms）と設定した場合には、ウィンドウの開いている期間を 50%（確実にウィンドウが開いているのは約 80ms 以降）に狭めても問題がなくなります（ただし、メイン処理 1 の直後にメイン処理 2 を実行したかどうかの確認はできません）。



さらに、ウォッチドッグ・タイマのクリアを処理の頭で行うようにすれば、処理時間のばらつきの影響をさらに小さくできるので、ウィンドウの開いている期間を 25%にすることも可能です（この場合も、メイン処理1の直後にメイン処理2を実行したかどうかは確認できません）。どちらのチェックを優先するかにより、処理方法を決めてください。

ただし、これだけでは、1 時間にメイン処理 1 を 20 回実行したかどうかを確認するには不十分です。
そこで、メイン処理1実行フラグを追加することを考えます。

・メイン処理 1 実行フラグ（カウンタ）

メイン処理 1 は、このフラグをカウントアップします。メイン処理 2 では、メイン処理 1 間隔フラグの値をカウントアップして、2 か 3（メイン処理 1 の間隔による）になったときにこのフラグを確認して、0 か 20（メイン処理1の連続した実行回数）以外の場合には、ウォッチドッグ・タイマをクリアしないようにします。これでメイン処理 1 の実行回数も確認することはできます。
このように、ウォッチドッグ・タイマをクリアする際の条件を追加することで、より細かなチェックを行うことができるようになります。しかし、あまり複雑にすると、本来実行すべき処理の制限が厳しくなったり、処理が複雑になって誤りを起こすなど、別のところで問題が発生する可能性があります。

1.2.6 割り込み

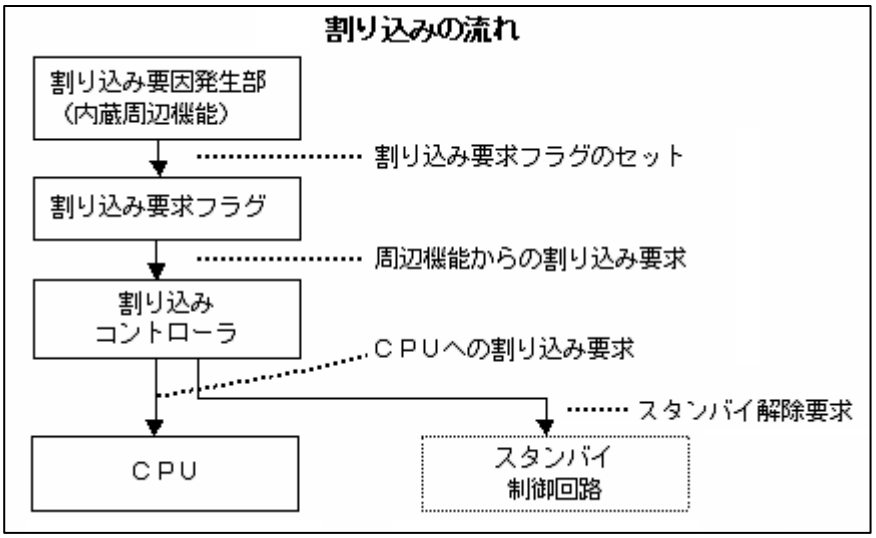
割り込みの動作の基本 [共通]

Q1. 割り込み処理について、どのような動作になり、どのような点に注意すればいいですか。

A1. [はじめに]

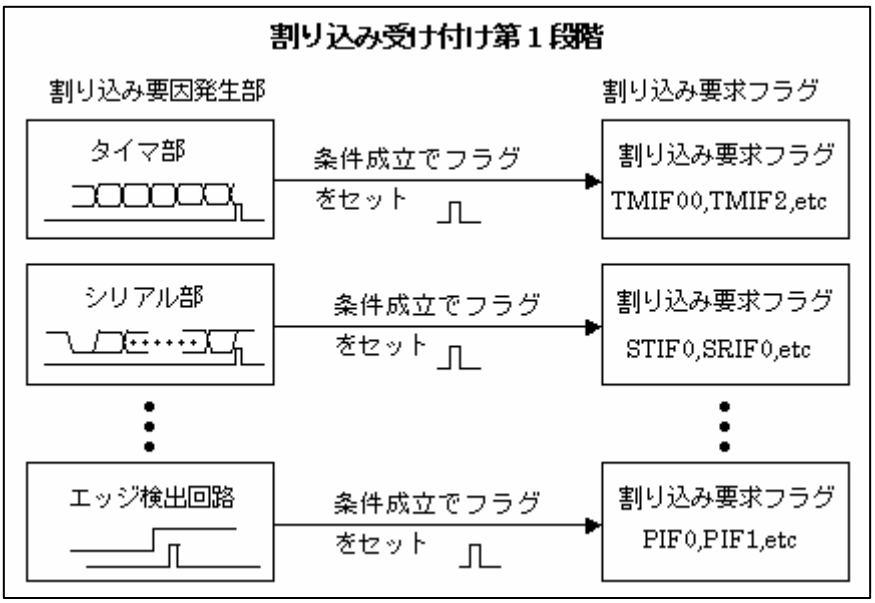
割り込み処理の基本的な動作は、大きく4つの部分に分けて考えることができます。

最初は割り込みの要因を発生させる部分で、2つ目が割り込み要求フラグ、3 つ目が割り込みの制御を行う割り込みコントローラ部、最後が実際に割り込みを処理する CPU 部です。



[割り込みの要因発生部]

割り込みの要因を発生させる部分には、タイマ部やシリアル部、外部信号のエッジ検出回路等があります（デバイスで異なる）。各部では、タイマの一致やシリアル・データの受信完了など、割り込み発生条件を満足すると、対応した割り込み要求フラグをセットします。



割り込みの第１段階は、各周辺機能部で指定された条件が満足されることです。タイマ部ではコンペア・レジスタとの一致等で、シリアル部では通信完了等で、外部割り込みではエッジ検出により、対応した割り込み要求フラグをセットします。また、プログラムで割り込み要求フラグをセットすることもできます。

[割り込み要求フラグ]

このように、各割り込み要因は対応した割り込み要求フラグをセットするだけで、割り込みの要因を発生させる部分と割り込みコントローラ部の動作は独立しています。この間を結ぶのが、割り込み要求フラグです。

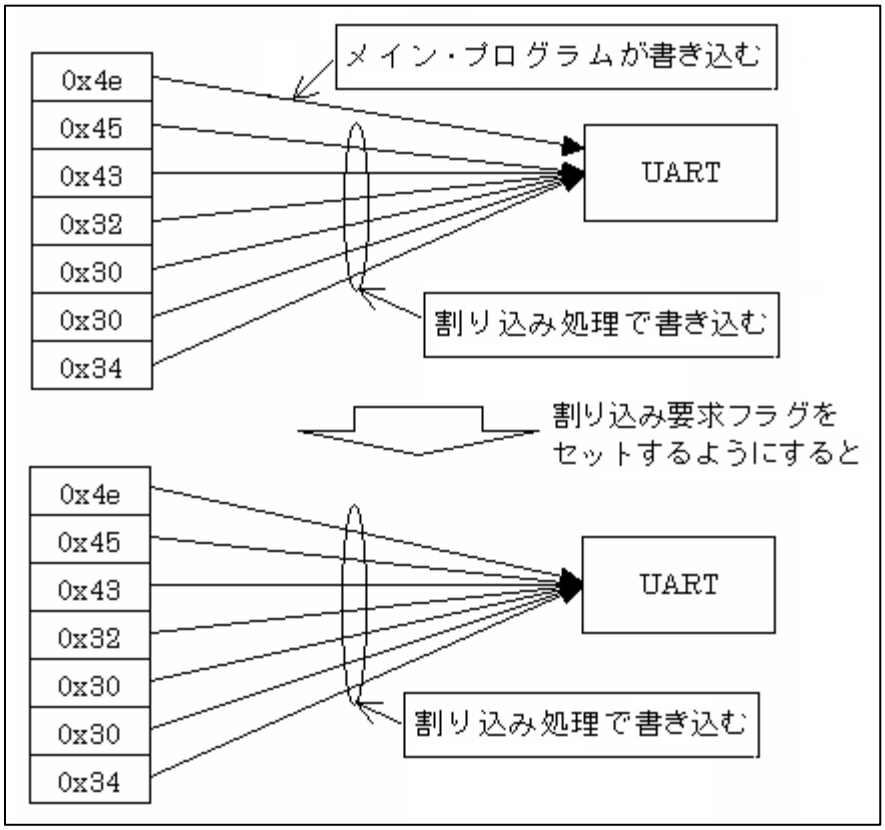
なお、各割り込み要因に対応した割り込み要求フラグは、1 つ(1 ビット)しかありません。したがって、割り込み要求が受け付けられる前に、同じ割り込み要求が複数回発生しても、それは 1 回としか認識できません。

また、割り込み要求フラグは、プログラムでセットしたりクリアしたりすることもできます。これにより、UARTでの送信処理を、最初のデータを含めて、すべて割り込みで実行させることができます。ただしこの場合には、必ずビット操作命令を用いて、1 命令で割り込み要求フラグをセットする必要があります。

<UART 送信処理例>

複数バイトのデータを送信する場合、通常はメイン・プログラムが出力データの 1 バイト目を UART に書き込むとともに、2 バイト目以降のデータの情報を割り込み処理プログラムに引き渡します。

割り込み処理プログラムは、メイン・プログラムから渡された情報を元にして、2 バイト目以降のデータを UART に書き込みます。このように、1 バイト目と 2 バイト目以降では、処理が異なります。



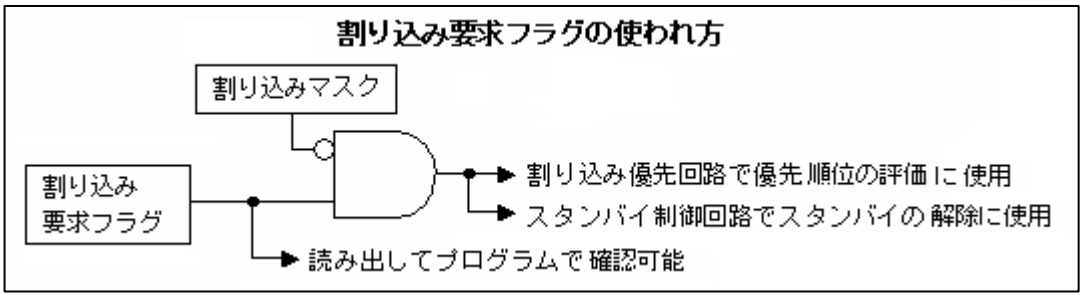
メイン・プログラムは、割り込み処理プログラムに送信データの情報を渡し、割り込み要求フラグをセットするだけとなります。実際の UART への書き込みは、すべて割り込み処理プログラムが処理しますので、全体の処理が見やすくなります。

[割り込みコントローラ部]

割り込み要求フラグ以降が、割り込みコントローラ部の動作となります。割り込みコントローラ部では、まず割り込み要求フラグが割り込みマスク・レジスタで有効にする(マスクしない)か無効にする(マスクする)かが決められます。マスクして無効に設定された割り込み要求フラグは、以降の動作で無視されます。

ただし、要求フラグはクリアされないので、マスク解除時点で、割り込み要求が有効になってしまいます。

マスクされなかった割り込み要求は、スタンバイの制御回路に送られ、スタンバイの解除に使用されます(78K4 のマクロ・サービスを除く)。また、この割り込み要求は、割り込み優先回路を介して、CPU への割り込み要求となります。CPU が割り込み要求を受け付けると、対応した割り込み要求フラグはクリアされます。
CPU が割り込み禁止状態では、割り込みは単にスタンバイ解除に使用されます(スタンバイ状態でない場合には保留されるだけです)。割り込み許可状態では、スタンバイ解除後、ベクタ割り込み処理を起動します。
なお、78K4 のマクロ・サービスでは、CPU が割り込み禁止でも受け付け、マクロ・サービス処理を行います。



割り込み要求フラグには、次の 3 つの使い方があります。

(1) プログラムでポーリング(読み出し)を行い、割り込み要求元の処理が完了したかどうかを確認

この場合、割り込みマスクをセットしておき、CPU への割り込みは発生させません。
確認後は、必ずフラグをクリアしてください。

(2) スタンバイの解除に使用

スタンバイ状態で、マスクされていない割り込み要求フラグがセットされると、スタンバイが解除されます。このとき、CPU が割り込み禁止状態であれば、スタンバイ解除後はベクタ処理をせず、スタンバイ命令の次から実行を再開します(78K4 でマクロ・サービスを実行する場合、スタンバイは解除されません)。
確認後は、必ずフラグをクリアしてください。

(3) CPU へのベクタ割り込み要求として使用

この場合には、他の割り込み要求フラグの状態と、CPU が実行中の割り込みの状態が、割り込み優先回路でチェックされ、最も優先度が高ければ、CPU への割り込み要求となります(フラグは自動クリア)。

割り込み優先回路は、他に発生している割り込み要求や、処理中の割り込みの優先度を比較します。また、CPU が実行中の割り込みの状態を監視し、 RETI 命令の実行までは、割り込み処理中と判断します。
また 78K4 や 78K0 でも、NMI の実行中は、たとえ CPU が EI 命令を実行しても、優先度の低い割り込み要求を受け付けないことになります。

[CPU 部]

CPU が割り込み許可状態なら、優先度がもっとも高い割り込み要求(割り込み制御回路で決定)を受け付け、PC や PSW の値をスタックに保存し、割り込みベクタに分岐します。このとき、該当する割り込み要求フラグはクリアされ、CPU は割り込み禁止状態になります。
なお、CPU が割り込み要求を受け付け、ベクタに分岐するタイミングは、命令の実行状態で異なります。割り込み要求を保留する命令や、割り込み禁止期間があると、大きく変動します。このため、細かなタイミングが問題になる場合は、割り込み処理ではなく、タイマの出力機能やリアルタイム出力機能などのハードウエア機能を使用してください。
割り込み処理が完了したら、RETI 命令で元の処理に戻ります。このとき、PSW は割り込み受け付け前の値に書き戻されるので、 RETI 命令の直前に EI 命令を実行する必要はありません。

割り込みの優先度制御の注意点 [78K0S、78K0、78K0R]

Q1. 割り込み処理について、優先順位に注意点はありますか。

A1. マイクロコンピュータで複数の割り込みを使用する場合、割り込みの優先度の制御や多重割り込みの制御が必要になります。これらの制御を誤ると、割り込みの優先度が逆転して、余分な割り込み要求を受け付けたり、必要な割り込み要求を受け付けないという問題が発生します。

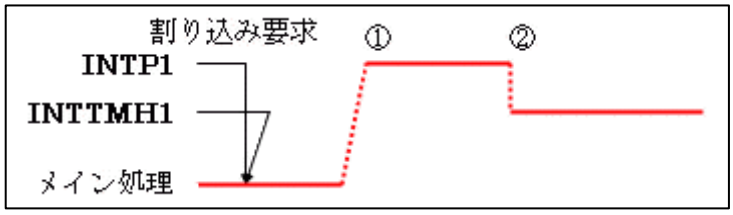
(1) ディフォールト優先度

78K0 などで扱う割り込みは、割り込み要求の要因ごとにディフォールト優先度が設定されています。この優先度は、複数の割り込み要求が**同時**に発生した場合に、どの割り込みを優先するかの判断に利用されるだけです。そのため、**ディフォールト優先度を多重割り込みの制御に用いることはできません。**

78K0S ではディフォールト優先度しかないので、78K0S で考えてみます。ここで、次のような割り込み要因を扱います（左側のディフォールト優先度が高い）。

INTP0 ＞ INTP1 ＞ INTTMH1

INTP1 や **INTTMH1** では、より優先度の高い割り込みを処理できるように、多重割り込みを許可するものとします。**INTP1** と **INTTMH1** が同時に発生した場合、CPU は割り込み可能になると、ディフォールト優先度の高い **INTP1** を受け付けます（①）。この状態で、**INTP0** 割り込み要求を受け付けるために割り込みを許可すると、保留された **INTTMH1** が受け付けられてしまいます（②）。



このような動作を避けるには、**INTP1** 割り込み処理の中で割り込みを許可する前に、割り込み要求を受け付けたくない **INTTMH1** をマスクします。また、**INTP1** 割り込み処理の最後で割り込みを禁止して、**INTTMH1** のマスクを解除します。

【注意】
割り込みの説明で、「**同時**」という言葉がよく使用されます。でも、ほとんどは、厳密な意味で「全く同じ時間（タイミング）」の意味ではありません。この「**同時**」は、CPU が割り込み要求を受け付けるタイミングを意味しています。つまり、CPU が割り込み可能なら、1 命令の実行時間が「**同時**」の対象になりますが、CPU が割り込み禁止状態で動作している場合は、CPU が割り込み要求を受け付け可能になるまでの期間（数十～数百命令以上になることもある）が「**同時**」の対象になります。



(2) ソフトウエアによる優先度指定

78K0 は 2 レベル、78K0R は 4 レベルの割り込み優先度（優先順位）を指定できます。この指定はディフォールト優先度と異なり、**同時に発生した割り込みの優先度判定以外**でも使用されます。

78K0R を例にすると、PSW レジスタの 2 ビットを用いて、4 つの割り込みレベル（レベル 0～3）をとり得ます。また、個々の割り込みの優先レベルも、PR レジスタの 2 ビットで、4 つのレベルのどれかに設定できます。これらの関係は、次の表のようになっています。割り込み要求を受け付けると、そのときの PSW の値はスタックにセーブされ、受け付けた割り込み要求の優先度レベルに対応した値が ISP0、ISP1 ビットにセットされます。

これらのビットの値は、RETI 命令を実行する（または PSW を書き換える）までは保持され、優先度の低い割り込み要求の受け付けを禁止します。

CPU の割り込みレベル		受け付け可能な割り込み（割り込み優先度）			
レベル (ISP0,1)	処理中の割り込みレベル	0	1	2	3
レベル 0 (0,0)	レベル 0 またはレベル 1	○	×	×	×
レベル 1 (0,1)	レベル 2	○	○	×	×
レベル 2 (1,0)	レベル 3	○	○	○	×
レベル 3 (1,1)	処理中の割り込みなし	○	○	○	○

ここで注意する必要があるのは、割り込み優先度レベル 0 の割り込み処理中（CPU の割り込み優先度がレベル 0）に割り込み許可すると、同じ優先度レベル 0 の割り込み要求を受け付けてしまうことです。これは、最優先の割り込みを処理中は、それ以上の割り込みを処理する必要がないので、「割り込み**優先度レベル 1** の割り込み処理中は多重割り込みを**許可**し、割り込み**優先度レベル 0** の割り込み処理中は多重割り込みを**許可しない**」ことで、レベル 0～3 の割り込みの優先処理が可能です。

【参考】
V850 は 8 レベルまでの割り込み優先度の制御が可能です。V850 は PSW ではなく、8 ビットの ISPR レジスタ（読み出し専用）で、処理中の割り込み優先度を保持します。割り込み要求を受け付けると、割り込み優先度に対応したビットをセットします。RETI 命令の実行時、ISPR レジスタにセットされているビットの中で、もっとも優先度の高い割り込みに対応したビットがクリアされます。この処理によって、割り込み優先度を制御しています。

(3) 割り込み優先度とレジスタ・バンク

78K0 と 78K0R には 4 組のレジスタ・バンクがあります。これを利用すれば、割り込み処理を高速化できます。レジスタ・バンクは、多重で処理すべき割り込みの優先度に応じて設定してください。ただし 4 組では、メイン処理を含めた 5 つのレベルには不足です。そこで、次の例のように、最低優先度の割り込みではレジスタ・バンクを使用せず、使用するレジスタはプログラムでスタックにセーブします。

- メイン処理、レベル 3 の割り込み　：レジスタ・バンク 0 を使用
- レベル 2 の割り込み　：レジスタ・バンク 1 を使用
- レベル 1 の割り込み　：レジスタ・バンク 2 を使用
- レベル 0 の割り込み　：レジスタ・バンク 3 を使用

このような使い分けができるのは、同じ優先度の割り込みが多重で処理されることがないからです。

(4) 多重割り込み

78K0R で多重割り込みをするには、単に優先度の低い割り込み処理の中で、割り込みを許可するだけです。78K0 では、割り込み優先度の制御は 2 つのレベルだけです。優先度の高い割り込みの処理中に、低い優先度の割り込み要求の受け付けを禁止できるだけなので、78K0R のように簡単には実現できません。78K0 は、個々の割り込みのマスク機能を使って制御する必要があります。78K0 で 4 レベル以上の優先度を使う例を次に示します。

優先レベル	ISP の設定	割り込み処理でのマスクの制御など
レベル 0（最優先）	0（高優先）	制御しない（割り込み禁止のまま）
レベル 1	0（高優先）	レベル 1 の分のマスクをセット／クリア
レベル 2	1（低優先）	PSW の ISP ビットをクリアする
レベル 3（低優先）	1（低優先）	レベル 3 の分のマスクをセット／クリア

たとえば、レベル 1 の割り込み処理では、同じレベル 1 の優先度の割り込み（自分自身を含む）のマスク・ビットをセットしてから割り込み許可にして、自身の割り込み処理を行います。割り込み処理が完了したら、割り込み禁止にして、セットした割り込みマスク・ビットをクリアします。

レベル 2 の割り込み処理では、PSW の ISP ビットをクリアして割り込みを許可にし、自身の割り込み処理を継続します。処理が完了したら、RETI 命令を実行します（これにより、PSW は元の値に復帰）。

レベル 3 の割り込み処理では、同じレベル 3 の割り込みをマスクしてから割り込みを許可します（マスクする対象が異なるだけで、処理はレベル 1 と同じ）。

78K0S の多重割り込みの場合は、割り込みマスクで優先度を制御することになります。この場合、どの割り込みのマスクをセットし、またクリアするかに注意が必要です。このため、多重割り込みを許す割り込み処理では、割り込み処理を開始した時点での MK0、MK1 レジスタの値を読み出して、スタックにセーブしてから書き換えます。その後、割り込み許可にして自身の割り込み処理を行います。処理が完了したら、割り込みを禁止し、スタックからセーブしておいた MK0、MK1 の値を元に戻して、割り込みを完了します。

割り込み応答時間の変動を少なくする [78K0、78K0S 共通]

Q1. 割り込み応答時間の幅は大きいようですが、ばらつきを少なくできますか。

A1. 割り込み要求のチェック・タイミングは、実行中の命令の最後で、実行後に受け付けます。このため、応答時間は、その命令の実行時間や、実行中のどのタイミングで割り込みを要求されたかで変動します（「最小割り込み間隔について」参照）。この変動を小さくするには、実行する命令について、次の注意が必要です。

- できるだけ**実行時間の短い命令**だけにする
- 割り込みに関係した SFR（特殊機能レジスタ）はアクセスしない
- 割り込み許可／禁止の操作を行わない

最適な例は、**HALT 命令**実行での割り込み要求待ちです（変動は、78K0 で 1 クロック、78K0S で 2 クロック）。

Q2. 同じ割り込み要求でも、状況によって異なる処理をさせます。割り込み処理でフラグを参照し、プログラムを分岐させると、処理ごとに開始時間が違います。割り込みベクタの値をダイナミック書き換えできませんか。

A2. 割り込みベクタを書き換えることはできませんが、次のような処理ができます。ショート・ダイレクト領域に、実際の処理部のアドレスを設定する変数を準備しておき、次の割り込みで処理する処理部のアドレスを事前に格納しておきます。その状態で、割り込みが受け付けられたら、その変数を読み出して、そのアドレスに分岐します。こうすると、割り込み要求の受け付けから処理開始までの時間は、いつも同じになります。具体的には、次のように saddr 属性で DSEG を宣言して、2 バイトの領域（ここでは Intaddr とします）を確保します。

```
        DSEG    saddrp
Intaddr:
        DW      2               ;割り込み処理アドレス格納領域
```

次に、割り込み処理の先頭で、次のような処理を実行します。

```
INTxxx:
      PUSH    AX                ;AX レジスタの内容をスタックにセーブ
```

```
;
;   必要に応じて割り込みで使用する他のレジスタをスタックにセーブします。
;
        MOVW    AX, Intaddr         ;処理アドレスの読み出し
        BR      AX                  ;処理アドレスへ分岐
```

これで、AX と他のレジスタがスタックにセーブされた状態で、割り込み処理部へ分岐できます。この処理は、割り込み処理の内容によらず一定なので、要求を満足します。

フラグでテーブルを参照し、処理アドレスを得て分岐する方法もあります。でも、できるだけ割り込み発生からの遅れを小さく抑えるには、処理アドレスを事前に変数に設定しておく前述の方法が有利です。

同じ割り込み要求を多重に受けてしまう [78K0]

Q1. 優先順位が低い割り込み処理で、すぐ EI にすると、同じ割り込み要求を多重に受けてしまいます。

A1. 単に割り込みを許可すると、同じ割り込みも許可されます。自身の割り込み要求をマスクしてください。

多重割り込み処理 [78K0]

Q1. 3 レベル以上の多重割り込みをするには、どのようにすればいいですか。

A1. 78K0 の割り込みは、2 レベルの優先度を設定できます。これとは別に、既定のディフォールト優先度があり、この 2 つで各割り込みの優先度が決まります。CPU が受け付け可能になった時点で、同時に発生する割り込み要求では、優先順位指定の高い方が受け付けられます。優先順位指定が同じなら、ディフォールト優先度の高い方が受け付けられます。この優先順位指定機能を使用すれば、2 レベルまでの多重割り込みは簡単に実現できます。3 レベル以上の多重割り込みでは、処理に注意が必要です。基本的には、割り込みマスク機能を使用することになります。**INTP0～INTP3** の 4 つの割り込み要因を使って説明します。

これら 4 つの優先度を「**INTP2＞INTP3＞INTP0＞INTP1**」として使用する場合を考えます。このとき、PPR0, PPR1 は 1 に設定し、PPR2, PPR3 は 0 に設定します。各割り込みで必要な処理は、次のようになります。

(1) INTP2 受け付け中の処理 [処理フロー(1)]

この割り込みが最優先ですので、一切の割り込みを受け付ける必要はありません。

したがって、この処理中には、割り込みを禁止したままにします。

(2) INTP3 受け付け中の処理 [処理フロー(2)]

この割り込みの処理中にも、**INTP2** は受け付ける必要があります。

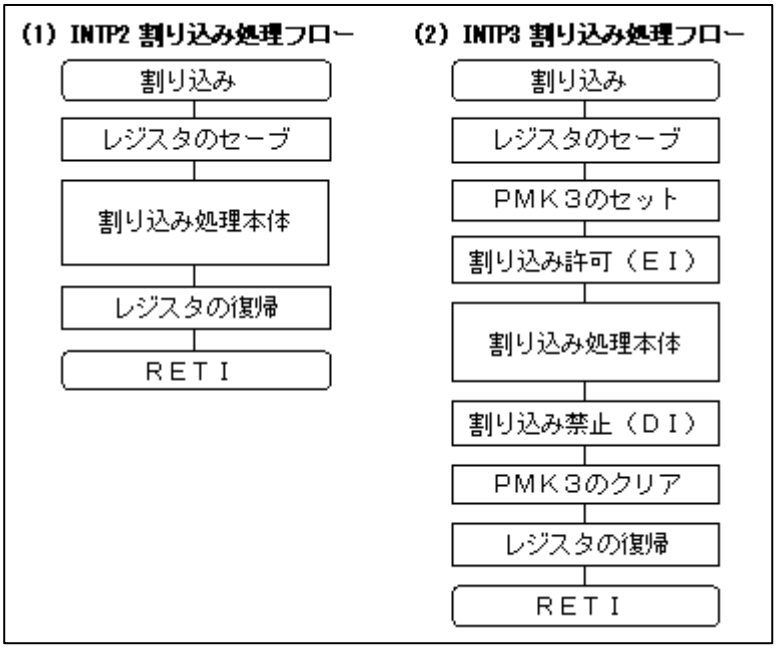
また、**INTP0**, **INTP1** は受け付けないようにする必要があります。

INTP3 は高優先順位の割り込みに指定してあるので、その実行中は PSW の中の ISP フラグが 0 になり、低優先順位の割り込みに指定してある **INTP0**, **INTP1** は自動的に禁止されます。

したがって、単純に割り込みを許可すればよいと考えられます。しかし、**INTP3** 割り込み処理中に **INTP3** が再度入る可能性があるときには、**INTP3** も禁止する必要があります。

このためには、割り込みマスク・フラグ PMK3 をセットしてから、割り込みを許可する必要があります。

また、割り込みから抜ける前に、割り込みを禁止して、PMK3 をクリアします。



(3) INTP0 受け付け中の処理 [処理フロー(3)]

この割り込み処理中は、**INTP2, INTP3** だけを受け付ける必要があります。また、自分自身と **INTP1** は受け付けないようにする必要があります。このためには、割り込みを許可する前に、PMK1 の値を保存しておき、PMK0, PMK1 をセットしておく必要があります。

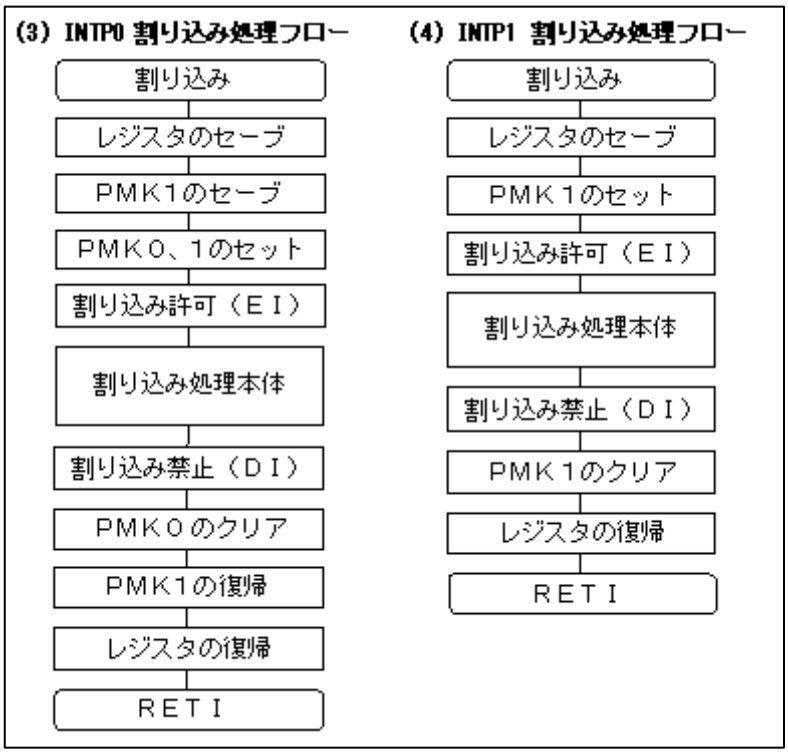
また、割り込みから抜ける前に、割り込みを禁止して、PMK0 をクリアします。

さらに、前もって保存しておいた PKM1 の状態を復帰させます。

(4) INTP1 受け付け中の処理 [処理フロー(4)]

この割り込み処理中は、自分自身以外(**INTP2, INTP3, INTP0**)を受け付ける必要があるため、割り込みを許可する前に、PMK1 をセットする必要があります。

また、割り込みから抜ける前に、割り込みを禁止して、PMK1 をクリアします。



割り込み要求フラグ待ちをしていると割り込みがかからない [共通]

Q1. UART 送信中、8ビットの割り込み要求フラグが立っているのに、次のプログラムで8ビット・タイマの割り込みが発生しません。

```
do{
} while (STIF == 0);
```

A1. STIF がセットされるのを待っているところの命令に問題があります。
この do～while 文は、BF 命令に置き換えられ、自分自身への分岐処理となります。
一方、割り込み要求フラグに対するアクセスが行われていると、割り込み要求が保留されてしまいます。
つまり、STIF がセットされるのを待っているところでは、割り込み要求が保留されて受け付けられません。
do { } の中に、何かダミーの命令を書いて、そこで割り込み要求を受け付けられるようにしてください。
たとえば、プログラムの頭に「#pragma NOP」と書いて、次のように変更してみてください。

```
do{
  NOP();
  NOP();
} while (STIF == 0);
```

サブシステム・クロック動作時、HALT 中の割り込みのクロック [共通]

Q1. サブシステム・クロック動作中、メイン・システム・クロック停止状態で HALT モードに入ったあと、それを割り込みで解除した場合、ベクタ割り込み処理は、そのままサブシステム・クロックで行われますか。

A1. はい、そのままサブシステム・クロックで動作します。
なお、HALT モードをマスクされていない割り込みで解除しても、ベクタ割り込み処理が行われるとはかぎりません。CPU が DI 状態であれば、単に HALT モードが解除されるだけで、ベクタ処理は行いません。

Q2. このベクタ割り込みをメイン・クロックで動作させるためには、ベクタ割り込みの最初に CPU クロックをメイン・クロックにするという命令を実行すればいいですか。

A2. いいえ、それだけではだめです。HALT モードが解除されたあとで、メイン・クロックの発振を開始させ、発振が安定してからメイン・クロックに切り替える必要があります。

リセット直後に割り込み要求を受け付けるか [共通]

Q1. リセット後にシステムが起動してから、ユーザ・プログラムを実行するまでの間に、外部割り込み要求が発生した場合、それを受け付けることはできますか。

A1. いいえ、受け付けません。リセット後のレジスタは、割り込み要求なし、割り込み処理禁止の状態にイニシャライズされますので、外部割り込み要求は、受け付けられない状態になっています。
各製品のユーザーズ・マニュアルで、リセット入力後の状態をご確認ください。

割り込み要求フラグ・レジスタ IF0L のフラグをクリアしないようにする [共通]

Q1. μPD78F0034A で、IF0L をソフトで書き込む際、処理ビット以外のフラグをクリアしないようにするには、どうすればいいですか。IF0L に書き込む直前に、PSW の IE を禁止にするだけでいいですか。

A1. いいえ、PSW の操作では対応できません。
ビット操作命令(clr1 や set1)を用いて、目的のビットだけを操作するようにしてください。
これにより、IF0L の他のビットに配置された割り込み要求フラグは、影響を受けなくて済みます。

割り込み要求フラグのクリアにより他の割り込み要求を受け付けなくなる [共通]

Q1. 「IF0 &= 0xfe;」の C 言語記述で要求フラグをクリアしていたら、リードとライトの間で、同一レジスタの他の要求フラグがセットされ、その後、ライト動作で再クリアされて割り込み要求を受け付けなくなりました。

A1. ビット操作命令を用いることで問題はなくなります。
クリアをビット操作命令「__asm("clr1 IF0,0");」に変更してください。

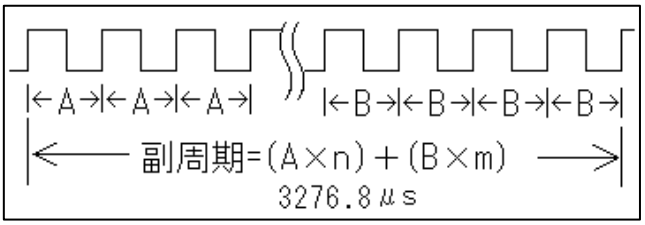
1.2.7 PWM

タイマ 0 を用いた 14 ビット PWM 出力周期とパルス幅(基本事項)

Q1. μPD780308 の PWM 出力の周期とパルス幅について、次の例で基本事項を教えてください。

条　件	・メイン・システム・クロック周波数:5.0MHz ・16 ビット・タイマのカウント・クロック:5.0MHz(MCS=1, TCL06=0, TCL05=1, TCL04=0 に設定) ・16 ビット・キャプチャ/コンペア・レジスタ CR00=N
PWM 出力	・カウント・クロック:1/5MHz=0.2us ・周期:0.2us×2^{16}=13.1ms ・パルス幅:0.2us×(N+1)

A1. いいえ、1 周期の中で、指定したデューティが出力される構成ではありません。
この PWM 出力は、64 個のパルスを使って、14 ビット分の分解能を得る構成となっています。
CR00 の上位 14 ビットで、アクティブ期間の合計(**副周期**)を指定します。
この**副周期**は、64 個の基本パルスで構成します。基本パルスは 2 種類あって、基本 A を n 個、基本 B を m 個、合計 64 個の基本パルスを使います（A と B については、あとで説明します)。



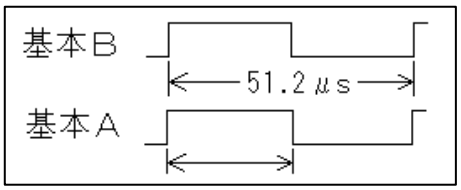
1 つの副周期に含まれる A を n 個とすれば、B は m(＝64－n)個となります。
m＋n＝64　（n の数は、14 ビットのうち下位 6 ビットで指定します）
1 つの基本パルスの周期(基本周期 A と基本周期 B)は、上位 8 ビット(256 クロック)で指定します。

この例では、基本周期は 51.2μs です。したがって、この例では副周期が次のようになります。

副周期＝基本周期(51.2μs)×64＝3276.8μs

なお、実際に出力される波形は、A と B が適度に混ざった波形です。

しかし、副周期に含まれる合計は、上で述べたとおりの数になっています。



さて、A(attribute)と B(basic)の違いはなにでしょう。

基本パルスのデューティは、上位 8 ビットで指定します。上位 8 ビットで指定した波形は、**基本 B** です。

基本 A は、基本 B より 1 クロックだけデューティを多くしたパルスです。つまり、上位 8 ビットで指定した値より、1 つ大きい値のパルスとなります。

CR00 の下位 6 ビットが 0 であれば、副周期はすべて基本 B だけで構成されます。

下位 6 ビットの値が 1 なら、基本 A が一つと基本 B が 63 個になり、下位 6 ビットが 63(オール 1)であれば、基本 A が 63 個、基本 B が 1 個で構成されます。

CR00 の上位 8 ビットで基本周期(クロック 256 個分の期間)を決めますが、これだけでは 8 ビット分の精度しか得られません。そこで、下位 6 ビットで示される数だけ、基本 A を混ぜた副周期を採用することで、14 ビット分の PWM 分解能が得られるようになっています。

<補足>

このようにパルスを細かく分割することで、全体(3276.8μs)の期間の中のパルス数を増やすことができます。すなわち、PWM 出力波形の周波数成分を上げることが可能となります。

この端子には、ローパス・フィルタを付加しますが、周波数が高くなることでフィルタの時定数を短くできます。その分、部品(コンデンサ)が小さくて済むように配慮した設計となっています。

PWM を使用しない場合の PWMn への設定値

Q1. μPD784038 で、PWM を使用しない場合の PWMn への設定値は、00H でいいですか。

A1. PWMC レジスタで出力禁止に設定していれば、モジュロ・レジスタの設定は必要ありません。設定するなら、0FFF0H にされることをお勧めします。

PWM の出力をロウ・レベルに固定(PWM=0%)

Q1. μPD789177 で、PWM 出力を 0 固定したいのに、ユーザーズ・マニュアルに「CR8n に 00H を設定しないでください。PWM が正常に出力されないことがあります」という注意書きがあります。どうすればいいですか。

A1. TOE8n をリセットして、ポート・モードにしてください。CR8n に1をセットしておけば、PWM のフリップ・フロップがロウの状態で INTTN8n が発生すれば、TOE8n を制御できます。

Q2. 「PWM が正常に出力されない」とは、どのような意味ですか。

A2. 出力のフリップ・フロップをセットする信号(カウント 00 で発生)とリセットする信号(CR8n との一致の次に発生)が、同時に発生してしまい、フリップ・フロップがどのような状態になるかを規定できないことを意味します。

PPG 出力と PWM 出力の違い [共通]

Q1. タイマ出力機能に PPG 出力と PWM 出力があり、同じような信号を出力していますが、違いはなにですか。

A1. PPG 出力は、「周期と幅」を指定してパルスを出力します。この機能では、波形(タイミング)が重要です。このため、パラメータ(パルス幅と周期)の変更は、すぐ動作に反映されます。そのため、変更のタイミングによっては、出力が反転してしまうことがあります。

PWM 出力は、アナログ量を出力するために、指定された周期の中で、アクティブな期間を指定します。この機能では、波形よりアクティブ期間の比率が重要です。そのため、アクティブ期間を複数に分割して出力する機能をもったマイコンもあります。また PWM 出力では、任意のタイミングでパラメータ(周期やアクティブ期間)を変更しても、周期の切り換わりのところで反映され、おかしな出力にはなりません。そのかわり、パラメータが変更されても、すぐには動作に反映されません。

Q2. PPG 機能を用いて、PWM 信号を出力できますか。また、そのときの注意点はありますか。

A2. はい、PPG 機能を用いて PWM 信号を出力できます。その場合は、周期やアクティブ期間の書き換えに注意が必要です。周期の書き換えは、必ず 1 周期の終了を示す割り込みタイミングで行ってください。

使用するタイマによっては、カウント動作を停止してから書き換える必要があります。アクティブ期間の変更は、カウントを停止せずにできますが、書き換えタイミングには注意が必要です。

アクティブ期間を指定する比較レジスタとの一致が、1 周期の中で 2 回発生したり、全く発生しなかったりすると、出力が反転してしまうタイマもあります。

1.2.8 CAN

CAN の設定

Q1. μPD780701Y で、次の CAN バス・タイミング・パラメータのときの設定とクロック周波数を教えてください。

Time Quanta＝1000ns、Synchronization Jump Width＝3000ns、Sampling mode＝1 回

A1. 6MHz のシステム・クロックで、BRPRS＝2(＝1/6)に設定すれば、1000ns の Time Quanta が選択できます。SYNC0 と SYNC1 で、データ・ビット時間＝8、同期ジャンプ幅＝3、サンプル点＝5 に設定してください。

Q2. 拡張フレームに対応した CAN2.0 (Bosch 仕様 2.0 パート B)ですか。

A2. はい、そうです。

Q3. 連続 11 ビットのリセッシブを 128 回検出した場合、送信を自動で再開できますか。初期化は必要ですか。

A3. はい、エラー・アクティブ状態で自動的に送受信可能となるので、初期化は不要です。

μPD780833Y に搭載されている J1850(CLASS2)

Q1. J1850 (CLASS 2)と表記されていますが、他社製品では SAE-J1850 プロトコル準拠(CLASS B)と表記されています。呼称のちがいだけで、機能は同じですか。

A1. CLASS A～C はスピード分類の区分で、CLASS B が J1850 です。これは通信方式で、GM 用(CLASS2)、Ford 用、クライスラー用(CLASS2＋IFR)にさらに区分できます。つまり、μPD780833 内蔵の CAN は、CLASS B の中の一部の規格です。このため、CLASS B がご希望なら、カバーできる範囲が異なる可能性があります。簡単には置き換えられませんので、必要な仕様を明確にしたうえで、利用できるかどうかご検討ください。

1.2.9 I²C

I²C バスが起動できない [共通]

Q1. μPD780034AY を次のようにプログラムしましたが、ストップ・コンディションを検出できません。

```
PM3.2 = 1;          /* 兼用ポートの設定 */
PM3.3 = 1;
P3.2= 0;
P3.3= 0;
IICCL0.3 = 0;       /* 動作モードの設定 */
IICCL0.2 = 0;
IICCL0.0 = 0;
IICE0 = 1;          /* I2C 制御レジスタ 0 の設定 */
WTIM0 = 1;          /* 割り込みタイミング設定  */
PM3.2 = 0;
PM3.3 = 0;
SPT0 = 1;           /* ストップ・コンディション生成 */
while (SPD0 != 1) {  /* ストップ・コンディション検出 */
}
```

A1. ビット操作で兼用ポートの設定をされているのが原因です。P3.3=0; とされたときに、P3.2 が外部端子の状態(入力ポートでプルアップされていれば 1)である 1 になり、SDA0 端子(P3.2)が 1 のままとなります。このため、ストップ・コンディション(SCL0 端子が 1 のときの SDA0 の立ち上がりエッジ)が生成できません。
ポートの設定はバイト単位で行うか、PM レジスタを出力に設定したあとに P3.3=0;を実行してください。
FAQ の共通項目の「ビット操作の注意点」もご参照ください。

I²C バスをうまく制御できない [共通]

Q1. μPD780034AY で I²C バスを制御しようとしていますが、アドレス送信から先に進めません。

```
do{
  IIC0 = 0B10100000;     /* アドレス＆方向設定 */
  do { NOP();
  } while ( IICIF0 == 0 );
      IICIF0 = 0;
} while ( ACKD0 == 0 );
```

A1. ACKD0 確認のループが問題です。I²C バスでは、アドレス送信でアクノリッジが戻らないと、そのアドレスのスレーブは存在しないと認識します。ACKD0 確認でループし、アドレスを再送していますが、アドレスと解釈されず、アクノリッジは戻りません。、アドレスの再指定には、再度スタート・コンディションを発行してください。

Q2. アドレス送信でアクノリッジが戻らなかった場合、ACKD0 でのループがいけないのはなぜですか。

A2. I²C バスでは、スタート・コンディションの直後がアドレスの指定と定義されており、アクノリッジが戻らなくて再送信しても、スレーブはそれをアドレスとは解釈しません。スレーブにアドレスと認識させるには、再度スタート・コンディションを発行して、アドレスを再送してください。または、ストップ・コンディションを発行して、いったんバスを開放し、時間をおいてから、再度スタート・コンディションを発行して、アドレスを送信してください。

備考 存在するスレーブからアクノリッジが戻らない場合、スレーブが処理中でビジィ状態であることが考えられます。これは、個々のデバイスで異なるので、各デバイスのマニュアルでご確認ください。

I²C のクロック周波数が設定と異なる [共通]

Q1. μPD784031Y で、プリスケーラ設定のクロック周波数と、バスで実測した周波数が異なります。

A1. 設定値は、信号の立ち上がり時間が 0 のときの周波数に該当します。実際のバスでは、バスの容量により立ち上がりが鈍ります。その分は、余分に時間が必要なので、設定した周波数よりは遅くなります。

簡易 I²C バス [78K0R]

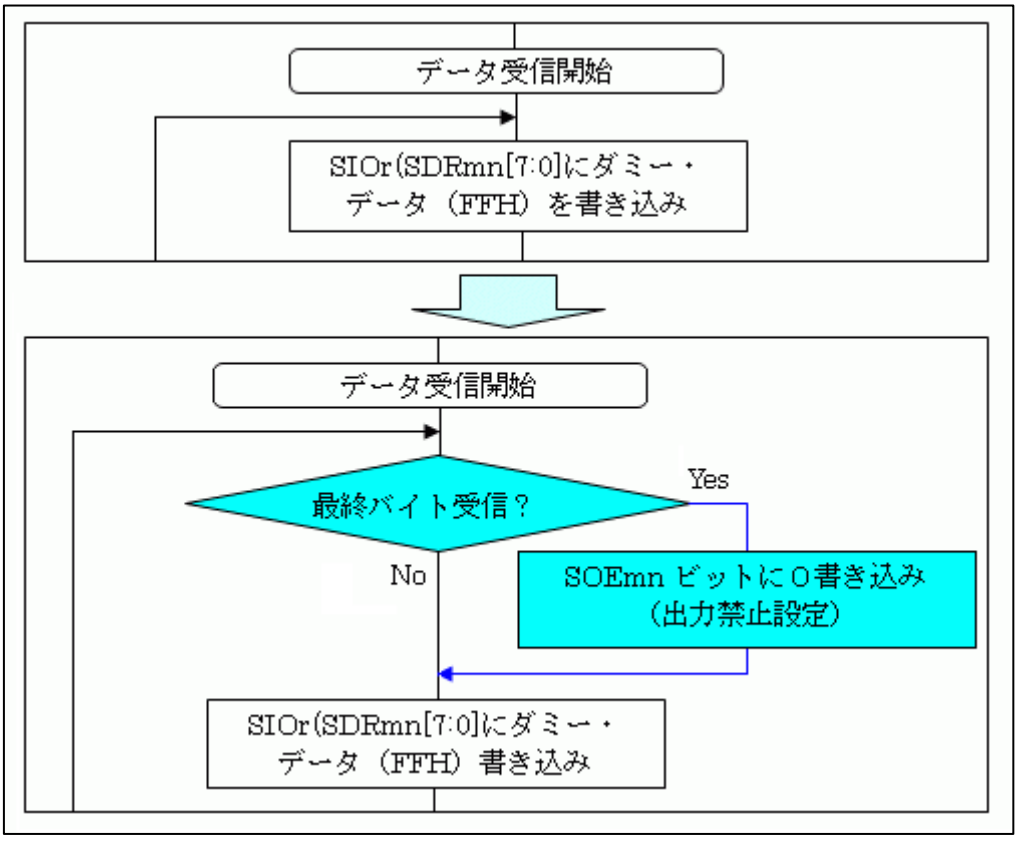
Q1. 78K0R/Kx3 の簡易 I²C バスとは、どういうものですか。

A1. シングルマスタ・モードのマスタ機能だけをもったI²Cバスです。ウエイト機能をサポートしていないので、遅いスレーブは接続できません。

簡易 I²C 機能 [78K0R]

Q1. マニュアル掲載のフローを参考にして、78K0R/KG3 に搭載されたシリアル・アレイを簡易 I²C で使用して EEPROM を制御します。EEPROM の書き込みと読み出しはできますが、その後の動作がおかしくなることがあります。原因と対策を教えてください。

A1. EEPROM から最後のデータを読み出したときに、ACK を戻しているため、EEPROM が読み出し継続と判断して次のデータを出力し、そのデータが SDA 信号をロウ・レベルにしているのではないでしょうか。
この対策としては、マニュアル記載のフローチャートのデータの受信をスタートする前に、残りデータ数が 1 になったかどうかの判断を追加して、残りが 1 で SDA への出力を禁止することが考えられます。次のフローの**青の部分の処理を追加**してください。



I²C バスへの不要な信号出力 [78K0/Kx]

Q1. 78K0/Kx2 のユーザーズ・マニュアルの I²C のフローチャートを基に、次のようにプログラムしましたが、最初に SCL0 信号や SDA0 信号がロウ・レベルに同時出力されてしまいます。

```
/* ポートの設定 */
  P6 = 0x00;     /* 兼用端子を OR 回路を通して IIC 用にするために、ポート側を 0 に設定 */
  PM6 = 0xFC;    /* 兼用端子を出力モードに */

/* I2C の初期設定 */
  IICX0 = 0x00;     /* 転送クロックの選択 */
  IICCL0 = 0x01;    /* 標準モードの転送クロックの選択 */
  SVA0 = 0x02;      /* 自局アドレスの設定 */
  IICF0 = 0x00;     /* 開始条件の設定 */

/* 通信開始準備 */
  IICC0 = 0x9C;        /* I2C 通信許可 */
  SPT0 = 1;            /* ストップ・コンディション生成 */
  while (SPD0 != 1){
  }                    /* ストップ・コンディション検出 */
  STT0 = 1;            /* スタート・コンディション生成 */

/* 通信開始 */
  IIC0 = 0x04;      /* マスタ送信開始 */
    :
```

A1. 次のように、ポートの設定では P60 と P61 を 1(ハイ)にしておいて、IICE0 ビットを 1(I²C 通信許可)にしてから、P60 と P61 を 0(ロウ)にしてください。IIC0 機能が動作許可(IICC0 レジスタの IICE0 ビットを 1 にする)まで、ポート機能として動作するので、ポートの設定で P60 と P61 を 0(ロウ)にして出力設定すると、SCL バス上と SDA バス上にロウ・レベルを出力してしまいます。

```
/* ポートの設定 */
  P6 = 0x03;     /* 兼用端子を OR 回路を通して I2C バス上 1 を出力しておくために、
                    ポートを 1 に設定 */
  PM6 = 0xFC;    /* 兼用端子を出力モードに */

/* I2C の初期設定 */
  IICX0 = 0x00;     /* 転送クロックの選択 */
  IICCL0 = 0x01;    /* 標準モードの転送クロックの選択 */
  SVA0 = 0x02;      /* 自局アドレスの設定 */
  IICF0 = 0x00;     /* 開始条件の設定 */

/* 通信開始準備 */
  IICC0 = 0x9C;        /* I2C 通信許可 */
  P6 = 0x00;     /* 兼用端子を OR 回路を通して IIC 用にするために、
                    ポート側を 0 に設定 */
  SPT0 = 1;            /* ストップ・コンディション生成 */
  while (SPD0 != 1){
  }                    /* ストップ・コンディション検出 */
  STT0 = 1;            /* スタート・コンディション生成 */

/* 通信開始 */
  IIC0 = 0x04;      /* マスタ送信開始 */
    :
```

1.2.10 LCD(液晶表示)

LCD 駆動電圧の発生方式

Q1. LCD コントローラ／ドライバについて、LCD 駆動電圧の発生方式の種類と端子接続を教えてください。

A1. 78K0, 78K0S シリーズ内蔵の LCD コントローラ／ドライバは 、LCD 駆動電圧の発生方式により、いくつかに分類されます。主流は、従来の抵抗分割方式から、内部昇圧方式に変わってきています。

[電圧発生方法による分類]

(1) 抵抗分割方式

回路が簡単で、従来より使われてきた方式です。内部電源か外部から供給した電圧を内蔵抵抗で分割し、必要な電圧を発生します(マスク版以外のマイコンでは、抵抗は内蔵せず外付けになることがあります)。
表示方式に応じて外部でショートし、1/3 バイアスや 1/2 バイアスに切り替えます(デバイスによっては固定)。
通常、LCD 駆動電圧は電源電圧より低くなりますが、μPD789327 サブシリーズのように、電源電圧以上の LCD 駆動電圧を入力できるものもあります。

(2) 容量分割方式

分割抵抗の代わりに、外付けコンデンサで電圧を分割する方式です。抵抗のように常時電流は流れないので、消費電力を削減できます。この方式も、低電圧動作では LCD 駆動電圧が低くなってしまいます。

(3) 内部昇圧方式

外付けコンデンサを接続し、基準電圧から昇圧して、LCD 駆動電圧を発生する方式です。通常、CAPx 端子の間と、VLCx 端子とV_{SS}の間にコンデンサを接続します。この方式は、CPU の電源電圧が低い場合にも、3V や 4.5V といった LCD 駆動電圧を発生できます。また、抵抗を使用しないため、動作電流が小さくできるので、電池動作に適しています。新しいデバイスでは、この方式が増えてきています。

[各ファミリの電圧発生方式]

サブシリーズ名	方　式	回路構成	接続例
μPD780308	抵抗分割	内蔵抵抗＋外部回路	図 1
μPD780318/28/38	内部昇圧	外部コンデンサ接続 (抵抗で電圧調整可能)	図 6
μPD780958	容量分割	外部にコンデンサ接続	図 5
μPD780344/54	内部昇圧	外部にコンデンサ接続	図 7
μPD789306/16	内部昇圧	外部にコンデンサ接続	
μPD789327	抵抗分割	V_{LC0}を供給、分割は固定	図 2
μPD789407A/17A	抵抗分割	内蔵抵抗＋外部でショート	図 3
μPD789426/36/46/56	内部昇圧	外部にコンデンサ接続	図 7
μPD789467	内部昇圧	外部にコンデンサ接続	
μPD789478	抵抗分割	外部回路	図 4
μPD789488	内部昇圧	外部にコンデンサ接続	図 7
μPD789830	抵抗分割	内部回路だけで固定	必要なし
μPD789835	内部昇圧	外部にコンデンサ接続	図 8

図 1　抵抗分割 1(μPD780308 サブシリーズ)のおもな接続例 (LEPS＝0、LIPS＝1 に設定)

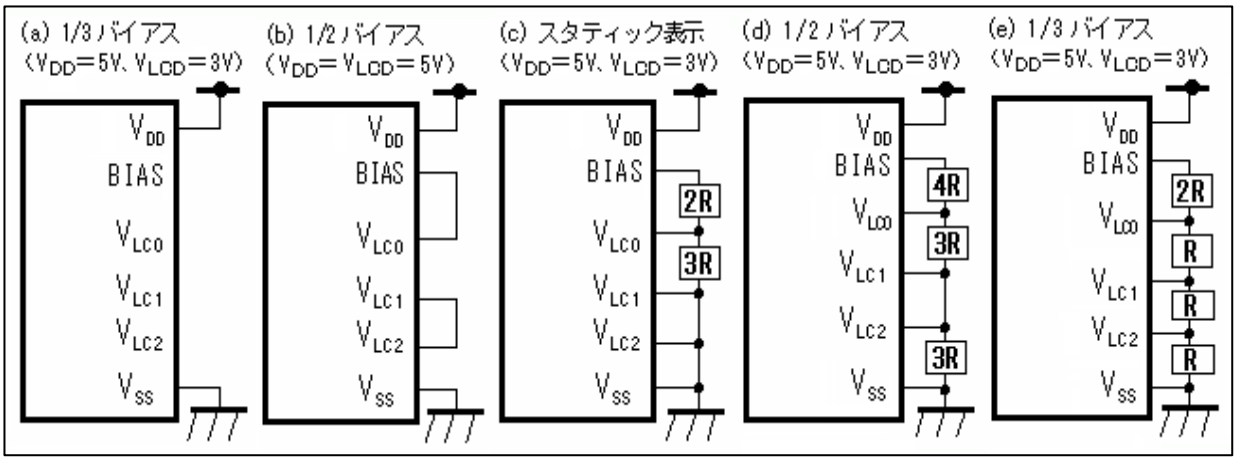


図 2　抵抗分割 2(μPD789327 サブシリーズ)の接続例

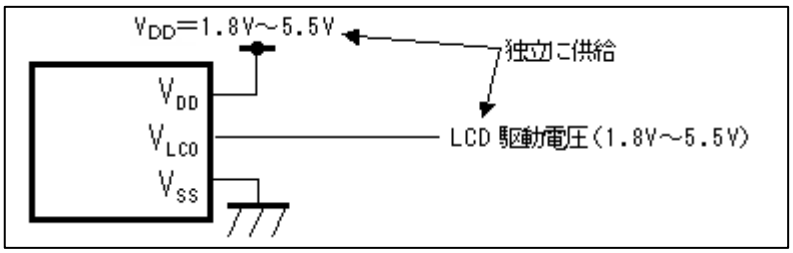


図 3　抵抗分割 3(μPD789407, 789417 サブシリーズ)の接続例

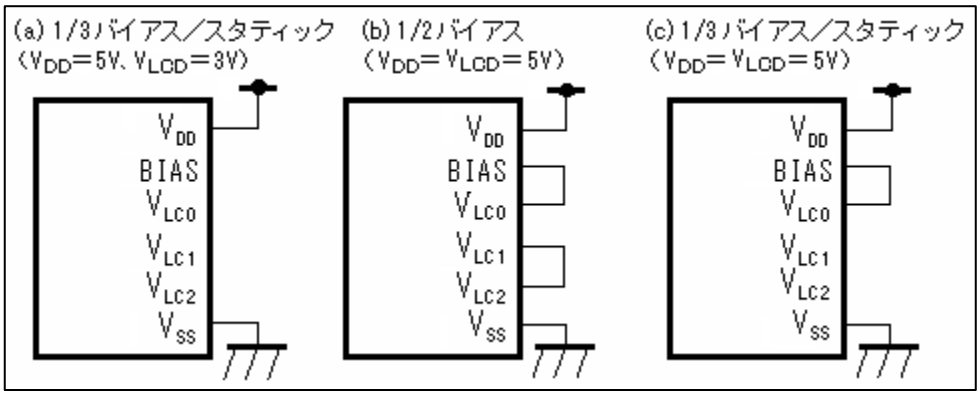


図 4　抵抗分割 4(μPD789478 サブシリーズ)の接続例

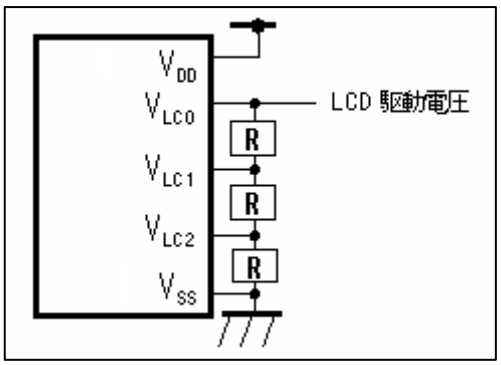


図 5 容量分割(μPD780958 サブシリーズ)の接続例

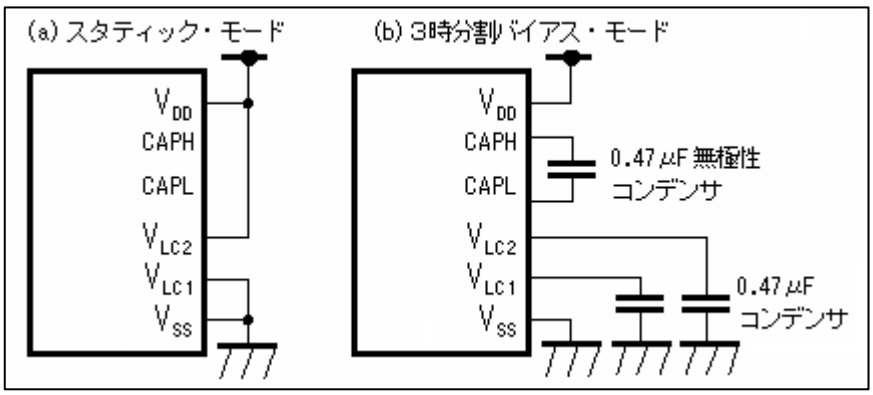


図 6 内部昇圧 1(μPD780318, 780328, 780338 サブシリーズ)の接続例

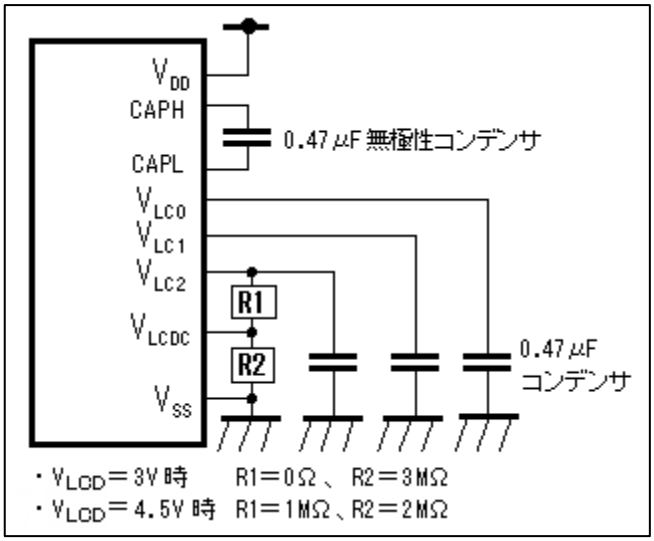


図 7 内部昇圧 2(μPD780344/45, 789306/16, 789426～67, 789478/88 サブシリーズ)の接続例
(LCD ゲイン調整レジスタ 0 で V_{LCD}＝4.5V(GAIN＝0)、V_{LCD}＝3V(GAIN＝1)を設定)

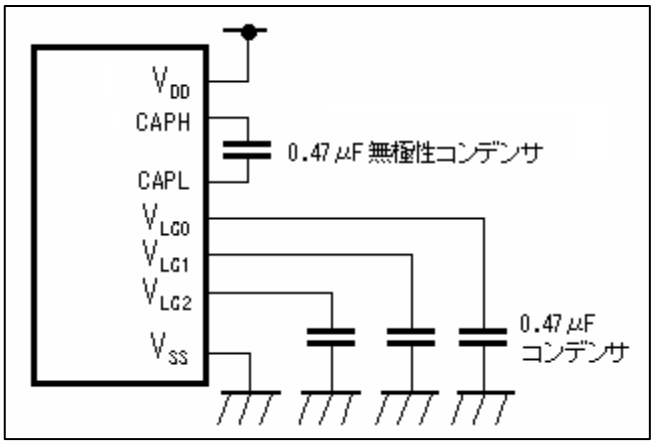
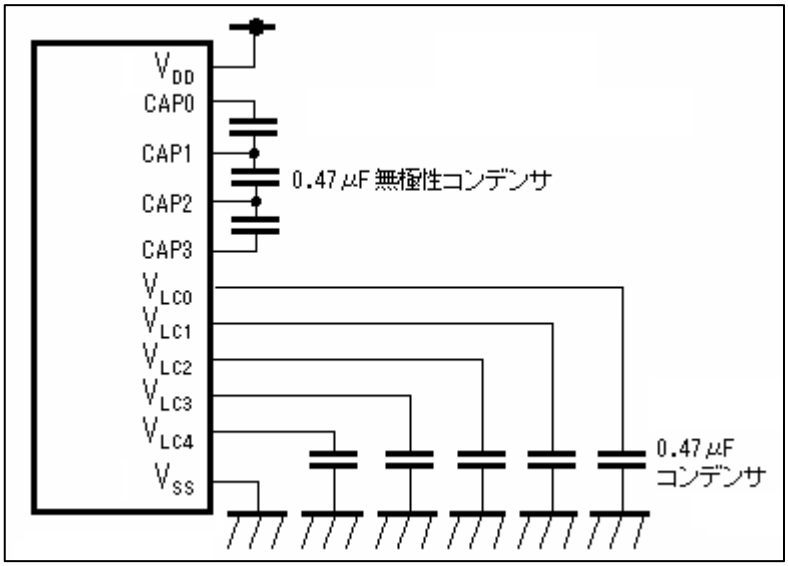


図 8 内部昇圧 3(μPD789835 サブシリーズ)の接続例 (V_{LCD} は LCD 昇圧レベル設定レジスタ 00 で 4.0V～5.5V に変更可能)



LCD 駆動回路で、V_{DD}＝5V 時に LCD 電圧 3V は得られるか

Q1. μPD789304 の LCD 電圧は、V_{DD} の 1 倍か 1.5 倍ですか。次のような、2 種類の使い分けはできますか。
1/3 バイアス、V_{DD}＝5V、V_{LCD}＝3V
1/3 バイアス、V_{DD}＝3V、V_{LCD}＝3V

A1. できます。電源電圧が 1.8V～5.5V で、LCD 昇圧制御レジスタ 0 を 1 にすれば、3V が得られます。昇圧回路は、レギュレータで V_{DD} を 1V か 1.5V に落とし、その 2 倍か 3 倍の LCD 用電圧を作ります。安定化する電圧は、LCD 昇圧制御レジスタで 1.0V か 1.5V に設定できます。この電圧は、CAPH-CAPL 間のコンデンサに充電され、V_{LC2} と GND 間には同じ電圧が、V_{LC1} と GND 間には 2 倍の電圧が、V_{LC0} と GND 間には 3 倍に昇圧された電圧が得られます。

LCD 分割抵抗の値

Q1. μPD789405A で、LCD 分割抵抗を各 10kΩ としていいですか。

A1. ご使用になる LCD パネルや配線により異なります。10kΩ であれば、問題ないと推測されます。

LCDM0 レジスタにある VAON0 の意味 [78K0S]

Q1. μPD789407A の LCDM0 レジスタ中の、VAON0 について教えてください。

A1. 低い電源電圧まで使うことを優先する場合、1 を設定してください。低電圧が不要な場合や、LCD が低電圧動作を推奨していない場合は、0 を設定してください。ユーザーズ・マニュアルの電気的特性で、LCD 特性をご覧ください。VAON0 の設定により、LCD 駆動電圧の最小値が異なります。

低電圧モード(VAON0＝1)　：min.2.2V
通常モード(VAON0＝0)　：min.2.7V (1/3 バイアス)
通常モード(VAON0＝0)　：min.3.0V (1/2 バイアス)

LCD 接続用外付けコンデンサの値と接続

Q1. μPD789446 に LCD を接続する場合の推奨外付け回路を教えてください。

A1. LCD 用の昇圧に、4 個のコンデンサ(いずれも 0.47μF)が必要です。次のように接続してください。

CAPH-CAPL 端子間　:0.47μF (C1)

V_{LC0} 端子-GND 間　:0.47μF (C2)

V_{LC1} 端子-GND 間　:0.47μF (C3)

V_{LC2} 端子-GND 間　:0.47μF (C4)

V_{LCn}-GND 間のコンデンサは、電解タイプでもかまいませんが、リークの小さいものを選んでください。

なお、CAPH-CAPL 間のコンデンサは、無極性であることが必要です。

LCD パネルとマイコンの接続(COM0-COM3、S0-S14)は、ユーザーズ・マニュアルの図 13-12 のとおりです。

LCD 電源電圧を上げたときの非表示部分の点灯

Q1. μPD789478 サブシリーズの外部抵抗分割で、LCD の点灯を濃くしようと V_{LCD}(LCD 電源)の電圧を上げると、非表示部分も点灯するようになってしまいました。対策はありますか。

A1. 本製品は時分割 1/3 バイアスを採用しており、コモン信号(COM0-COM3)とセグメント信号(S0-27)の間の電位差で、LCD の点灯／消灯が決まります。点灯(表示部分)はその電位差が $\pm V_{LCD}$ で、消灯(非表示部分)は $\pm(1/3)V_{LCD}$ です。使用する LCD の点灯と消灯のしきい値が、その間に入るように、V_{LCD} を決めてください。ただしこれは、V_{LC0}、V_{LC1}、V_{LC2}、と GND 端子間に付ける抵抗値が同じ場合です。V_{LCD} の電圧を上げるなら、非表示部の電位差が LCD の点灯／消灯のしきい値以下となるように、各抵抗の比率を変更してください。

LCD の制御 [78K0/Lx2]

Q1. 78K0/LF2のI²Cインタフェースで LCD表示をしたいのですが、I²Cバスでアドレスを正しく指定しても、LCDコントローラ／ドライバからACKが戻ってきません。

A1. 内蔵のLCDコントローラ／ドライバが正しく動作するには、動作クロックを供給し、リセットが解除されている必要があります。LCDコントローラ／ドライバに対するリセット信号は、P130を介して出力されています。この信号は、リセット解除後はロウ・レベルなので、使用するにはP130に1をセットして立ち上げてください。

また、動作クロックはP140から供給されますが、リセット後は供給されません。そこで、CKSレジスタで規格に合う動作クロックを選択し、PM140に0を設定後、CKSレジスタのCLOEビットを1にセットして、クロックを供給してください。その後、I²CバスからLCDコントローラ／ドライバのアドレスを指定すれば、応答があります。

1.2.11 メモリ割り付け、ディレクティブ、LOCATION

通常のメモリ以外に変数やコードを割り付け

Q1. 内部拡張 RAM やバッファ RAM、外部拡張 ROM/RAM に、変数を割り付ける方法を教えてください。

A1. デバイスには、内部拡張 RAM やバッファ RAM のように、特定用途のメモリ領域がいくつかあります。また、デバイスの外部に ROM/RAM を増設する場合もあります。Cコンパイラを使用してプログラムを作成する際、これらのさまざまなメモリ領域に、特定の変数やコードを配置することがあります。このとき、変数やコードをメモリに配置する手順を以下に示します。基本的には、次の 2 手順です。

(1) 変数やコードの宣言前に#pragma section 指令を置き、配置指定する変数やコードのセクション名を変更します。分ける場合、別のセグメント名で #pragma section 指令を行い、そのあとに変数を宣言します。

(2) リンク・ディレクティブにより、具体的なアドレスと上記セグメントを結びつけます。

以下に、具体的な 2 例を紹介します。

例1. μPD780058Y の各種内部 RAM に変数を割り当て

メモリの種類と、そこに割り付けたい変数とセグメント名を次のように仮定します。

メモリの種類	サイズ	アドレス	変数名	セグメント名
(1)内部高速 RAM	1K バイト	(FB00～FE1F)	a, b, c	@@DATA (default)
同 saddr 領域		(FE20～FEFF)	x, y, z	@@DATA (default)
(2)バッファ RAM	32 バイト	(FAC0～FADF)	serbuf[32]	BUFSEG
(3)内部拡張 RAM	1K バイト	(F400～F7FF)	extdbuf[1024]	EXTDSEG

通常のソースには、内部高速 RAM に割り付ける変数を定義します。 sreg 宣言すると、saddr 領域に割り当てられます。

main.c

```
extern char serbuf[32];
extern char extdbuf[1024];
int a, b, c;
sreg int x, y, z;
void main()
{
}
```

次のソースで#pragma section が先頭にない場合、CC78K では直接 REL ファイルを出力できず、ASM ファイルを出力してしまうため、data.c のコンパイラ・オプションで、「アセンブラ・ソース・モジュール・ファイルの出力」をオンにします(-a)。

data.c

```
#pragma section @@DATA BUFSEG
char serbuf[32];
#pragma section @@DATA EXTDSEG
char extdbuf[1024];
```

リンカ・オプションで、リンク・ディレクティブを指定(-d linkdir.dr)し、作成します。
プロジェクト・マネージャをご使用の場合は、リンカ・オプションの設定の「その他」の中で指定できます。

linkdir.dr

```
merge   BUFSEG  : =LRAM          (a)
merge   EXTDSEG : =IXRAM         (b)
```

(a) セグメント BUFSEG をバッファ RAM 領域に割り当てます。
「LRAM」は、バッファ RAM(32 バイト)の**定義済みメモリ領域名**です。
(b) セグメント EXTDSEG を内部拡張 RAM 領域に割り当てます。
「IXRAM」は、内部拡張 RAM (1K バイト)の**定義済みメモリ領域名**です。

定義済みメモリ領域名は、デバイス・ファイルの添付文書(ディフォールトのリンク・ディレクティブ情報)**をご参照ください。**載っていない場合、リンク・マップの「MEMORY=XXX」の項目をご参照ください。
変数 a, b, c, x, y, z は、ディフォールトの領域に割り付けるため、指定不要です。
以上で、ビルド結果のリンク・マップ・ファイルより、希望のメモリ配置をされたことが確認できます。
リンカ・オプションの設定で、「パブリック・シンボル・リストの出力」をオンにしておいてください(-kp)。

s0l.map

```
*** Public symbol list ***
MODULE  ATTR       VALUE    NAME
(他のシンボルは省略)
main      ADDR       FB78H    _a
main      ADDR       FB7AH    _b
main      ADDR       FB7CH    _c
main      ADDR       FE20H    _x
main      ADDR       FE22H    _y
main      ADDR       FE24H    _z
data2     ADDR       F400H    _extdbuf
data1     ADDR       FAC0H    _serbuf
```

例 2. μPD780056Y に外部 ROM/RAM を接続して、コードと変数を割り当て

メモリの種類と、そこに割り付ける変数/コードとセグメント名を次のように仮定します。HEX ファイルは、内部 ROM 用と外部 ROM 用を別々に出力するようにします。

メモリの種類	サイズ	アドレス	変数名	セグメント名
(1)内部 ROM	48K バイト	(0000～BFFF)	main()	@@CODE
(2)外部 ROM	4K バイト	(C000～CFFF)	sub()	GROMSEG
(3)外部 RAM	1K バイト	(D000～D3FF)	gaibubuf[1024]	GRAMSEG

通常のソースには、内部 ROM に割り付ける変数を定義します。

main.c

```
void sub();
void main()
{
   sub();
}
```

外部 ROM に割り付ける関数は、ソースの先頭に#pragma section を記述したソース中に記述します。

sub.c

```
#pragma section @@CODE GROMSEG
extern char gaibubuf[1024];
void  sub()
{
    gaibubuf[1] = 123;
}
```

外部 RAM に割り付ける関数は、ソースの先頭に#pragma section を記述したソース中に記述します。

data.c

```
#pragma section @@DATA GRAMSEG
char gaibubuf[1024];
```

リンカ・オプションで、このリンク・ディレクティブを指定(-d linkdir.dr)し、作成します。
プロジェクト・マネージャをご使用の場合は、リンカ・オプションの設定の「その他」の中で指定できます。

linkdir.dr

```
memory  GAIROM : (0C000H,1000H) /EX1    (a)
memory  GAIRAM : (0D000H,0400H)         (b)
merge    GROMSEG  : =GAIROM /EX1        (c)
merge    GRAMSEG  : =GAIRAM             (d)
```

(a) 外部 ROM 領域を定義します。
領域 0C000H から 1000H(4K バイト)に、「GAIROM」というメモリ領域名を付けます。
内部 ROM とは別のメモリ空間/EX1 として定義することで、別の HEX ファイルを出力できます。

(b) 外部 RAM 領域を定義します。
領域 0D000H から 0400H(1K バイト)に、「GAIRAM」というメモリ領域名を付けます。

(c) セグメント GROMSEG を外部 ROM 領域に割り当てます。
ここでも、別のメモリ空間/EX1 であることを記述します。

(d) セグメント GRAMSEG を外部 RAM 領域に割り当てます。

以上で、ビルド結果のリンク・マップ・ファイルより、希望のメモリ配置をされたことが確認できます。
リンカ・オプションの設定で、「パブリック・シンボル・リストの出力」をオンにしておいてください(-kp)。

s01.map

```
*** Public symbol list ***
MODULE    ATTR       VALUE    NAME
(他のシンボルは省略)
main      ADDR       0111H    _main
sub       ADDR       C000H    _sub
data      ADDR       D000H    _gaibubuf
```

内部 ROM 用 HEX ファイルです(スタートアップ・ルーチンと main.c)。

main.hex

```
:0200000080007E
:1000800061D0EE1C20FE9AEC001000000354FB032C
:1000900040FB0344FB400342FB1058FB0356FB1696
:1000A00080001478FBC6EA8000AD0687958684FA46
:1000B000F41678FBC6EA78FBAD06A1009786FAF441
:1000C0001680001420FEC6EA8000AD06879586845F
:1000D000FAF41620FEC6EA20FEAD06A1009786FAC5
:1000E000F49A11011000009AED00FAFEAF0240FBF5
:1000F0003061013027D667616EAD129696B7C6CAD9
:1001000000FBD68730AE01B69A0F01FAE9FAFE314C
:010110009856
:040111009A00C0AFE1
:00000001FF
```

外部 ROM 用 HEX ファイルです(sub.c)。

main.H1

```
:06C00000A17B9E01D0AF00
:00000001FF
```

リンク時に「F206:Segment xxxxxx can't allocate to memory-ignored」が出る

Q1. 78K0 で、ROM:0000H-7FFFH、外部 RAM:8000H-F7FFH に設定して使用します。アセンブル時に、次のプログラムで、「対象外のメモリ領域のため無視する」といった内容のエラー・メッセージが出ます。

```
data2   dseg  8000h
tmp:    ds    2
```

外部メモリ領域へのアクセスは、この方法ではできないのですか。

A1. リンク・ディレクティブで、外部 RAM があることを宣言していないためのエラーです。

エディタを使用して、リンク・ディレクティブの中に、次のメモリ・ディレクティブを記述してください。

```
MEMORY RAMEX : (8000H,7800H)
```

ここで、RAMEX は追加した外部メモリ領域の名前で、すでに使用している名前以外の名前を指定します。このように作成したリンク・ディレクティブのファイル名をリンカのオプション-D で指定してください。

詳細については、RA78K0 操作編マニュアルの「第 6 章 リンカ」の「6.4 リンク・ディレクティブ」「6.6 リンカ・オプション」をご参照ください。

ROM 内蔵品を外部 ROM で使う

Q1. μPD784218 を外部 ROM (ROM なし品と同じ方法)で使えますか。

A1. 可能ですが、お勧めできません。外部 ROM 動作には、ROM なしの μPD784031 をお勧めします。ROM 内蔵品で外部 ROM のプログラムを実行させると、一般に内部 ROM での実行に比べて、実行速度がかなり遅くなります。また、内部 ROM を介して割り込み処理を行うので、割り込みのレスポンスが悪くなることを考慮する

必要があります。この 2 点を考慮すれば、以下の方法(すべてを外部へジャンプさせる)で実現できます。

(1) 内部 ROM には、必要最小限の処理だけを入れる。
つまり、リセット解除後に初期化処理(外部メモリでは必須)だけ行い、外部の特定番地に分岐させる。

(2) 他の割り込みベクタについても、要因ごとに、外部の特定番地への分岐処理を内部 ROM に書いておく。
これにより、内部 ROM を介して、すべて外部 ROM へジャンプさせます。この場合、当然ですが、リセットの分岐先や、割り込みの分岐先は、事前に定義した番地となります。

ワード・データはメモリ上にどう格納されるか

Q1. 78K0 で MOVW AX,0FE20H の命令を実行すると、A、X レジスタには、それぞれ何番地の値が入りますか。

A1. A レジスタに FE21 番地、X レジスタに FE20 番地の内容が入ります。
基本的に、この CPU はリトル・エンディアンであり、下位データが低いアドレスに対応しています。
ただし、MOVW AX,saddrp の場合には、偶数番地しか指定できませんので、ご注意ください。

レジスタはどこにマッピングされているか [78K0S]

Q1. 78K0S の汎用レジスタは、何番地にマッピングされているのですか。

A1. 78K0S では、レジスタはメモリ・マップ上には配置されていません。
このため、マニュアルのメモリ・マップにも記載していません。

CALLT 領域をプログラム領域で使う

Q1. μPD789012 の CALLT テーブル領域は 0040H～007FH ですが、プログラム領域に使用できますか。

A1. 次の例のように、ソース中で配置したいセクション名(ここでは SEG1)のアドレスを AT で指定してください。
これで、40H から使えるようになります。

```
SEG1    CSEG    AT  40H
```

なお、実際には 40H ではなく、もっと前のベクタのないアドレスからも使用できます。
ただし、このように使用するより、CALLT 命令を有効に使った方が、プログラムが短くなると思われます。

16 ビットのスタック・ポインタが指す RAM 領域 [78K0R]

Q1. 78K0R/Kx3 のマニュアルで、スタック・ポインタのビット数が 16 ビットと記載されています。スタック・ポインタにアドレスをセットする場合、実際の RAM のアドレスの下位 16 ビットを設定ればいいですか。μPD78F1144 の RAM 領域は FDF00H～FFEFFH(20 ビット相当)なので、上位 4 ビットは必ず FH になりますか。

A1. はい、78K0R のメモリ空間は 20 ビット・アドレス空間で、スタックを割り当てられる RAM 領域は、製品で異なりますが、30K バイトある製品でも F8700H～FFEDFH です。
したがって、上位 4 ビットは必ず FH なので、スタック・ポインタに下位 16 ビットを設定してください。

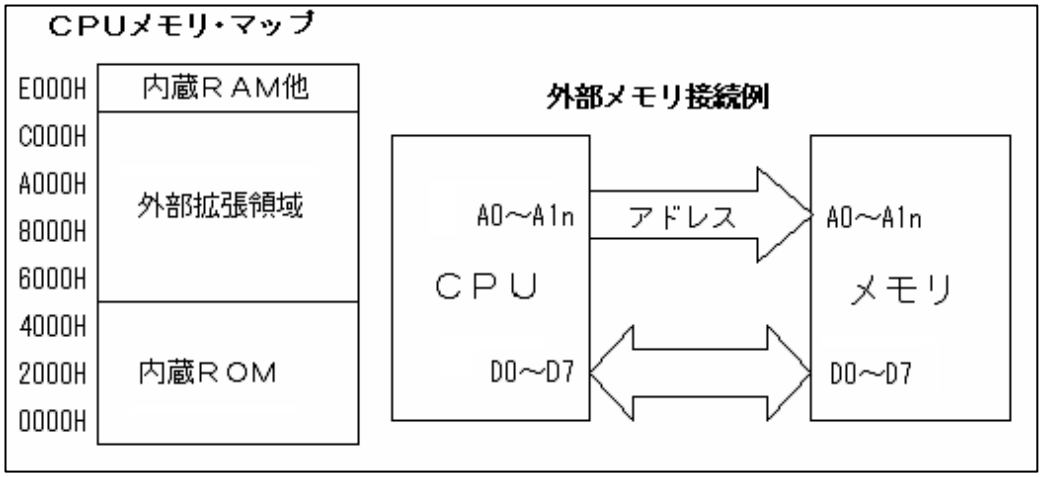
1.2.12 メモリ

「1.2.11 メモリ割り付け、ディレクティブ、LOCATION」もご参照ください。

メモリ外部拡張の基本事項

Q1. メモリの外部拡張について、基本的なことを教えてください。

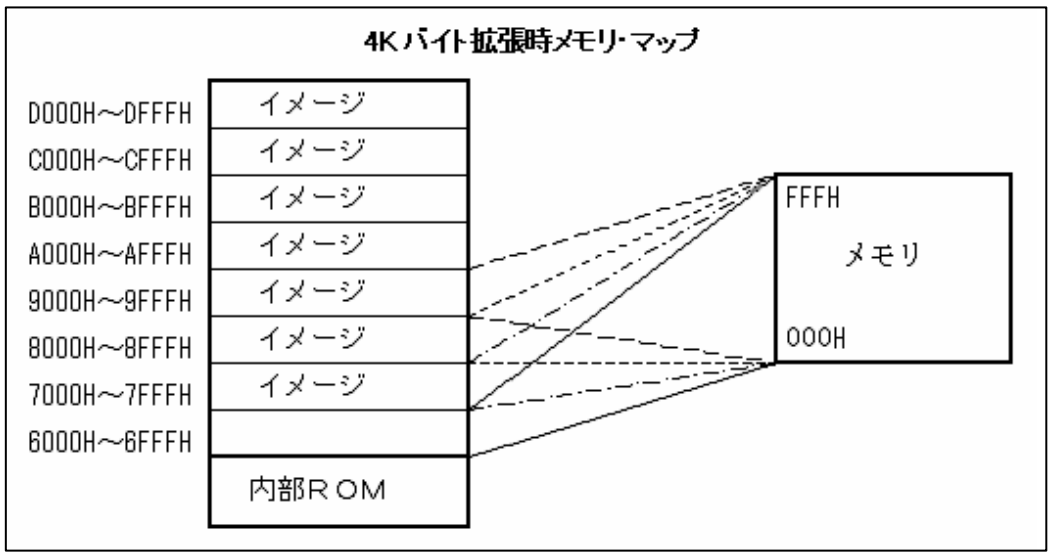
A1. いくつかのマイコンは、内部の ROM や RAM 以外に、外部にメモリを拡張できるようになっています。
その内部アドレスと外部アドレスの関係を下の図に示します。
なお、デバイスや設定によっては、マイコンが使用できるアドレス情報の一部しか外部に出力されない場合があります。



CPU メモリ・マップに示すように、CPU が 16 ビットのアドレス空間を扱え、内部 ROM が 24K バイトある場合について説明します。
この場合、6000H 以降の部分が、外部に拡張領域として使用できることになります。
簡略化のために、外部拡張領域は、DFFFH までの 32K バイトとして説明します。

(1) 外部に 12 ビットのアドレス出力(4K バイト拡張)を指定した場合

この場合、外部に物理的に拡張できるメモリは、2^{12} = 4K バイトになります。
CPU から 6000H をアクセスすると、外部にはその下位 12 ビット分のアドレス(000H)が出力されます。
同様に、7000H や 8000H, 9000H をアクセスしたときにも、下位の 12 ビットは同じく 000H となります。
このように、同じアドレスがアクセスされることになり、イメージが 4K バイトごとに現れます。



(2) 外部に 14 ビットのアドレス出力(16K バイト拡張)を指定した場合

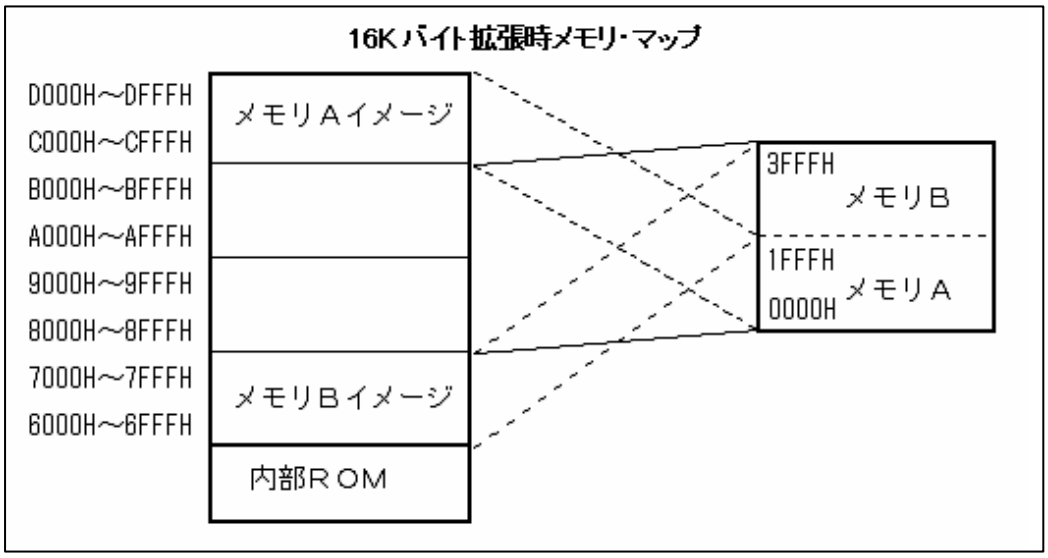
この場合、外部に物理的に拡張できるメモリは2^{14} = 16K バイトになります。

CPU から 6000H をアクセスすると、外部にはその下位 14 ビット分のアドレス(2000H)が出力されます。

同様に、7000H では 3000H、8000H では 0000H、9000H では 1000H、A000H では 2000H、B000H では 3000H が、それぞれ出力されます。

したがって、6000H から使用した場合には、内部のアドレスの並びと外部のアドレスの並びが逆転してしまうことになります。外部に RAM を拡張した場合には問題はありませんが、PROM やフラッシュを接続する場合はご注意ください。これを避けるには、8000H 以降のアドレスだけを使用することをお勧めします。

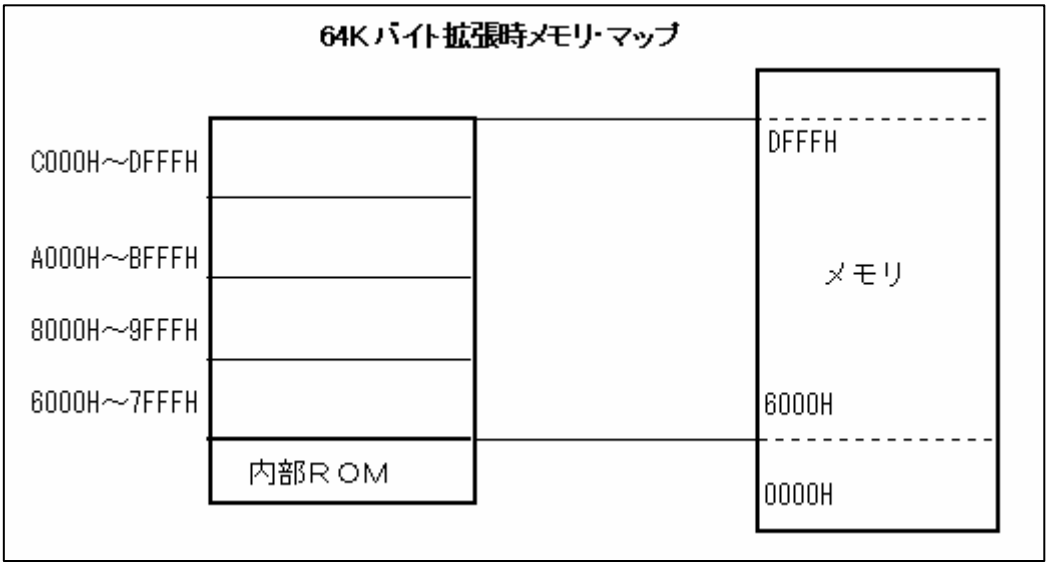
また、複数のアドレスで同じ外部アドレスが出力されるので、RAM を拡張したときには、異なるアドレスからアクセスしないように管理する必要があります。



(3) 外部に 16 ビットのアドレス出力(64K バイト拡張)を指定した場合

この場合、外部に 64K バイトの空間が使用できるようになります。ただし、CPU そのものが 64K バイトの空間しか扱えないため、外部に拡張できるのは、64K バイトから CPU 内蔵分を除いた領域だけです(デバイスによっては、さらに少なくなることがあります)。このとき、外部のアドレスは内部のアドレスと同じ値になります。

このため、内部 ROM と重なった部分(この場合には 0～5FFFH)がむだになる可能性があります。



関連 FAQ：1.2.11 の「通常のメモリ以外に変数やコードを割り付け」

外部メモリ・アクセスのタイミング

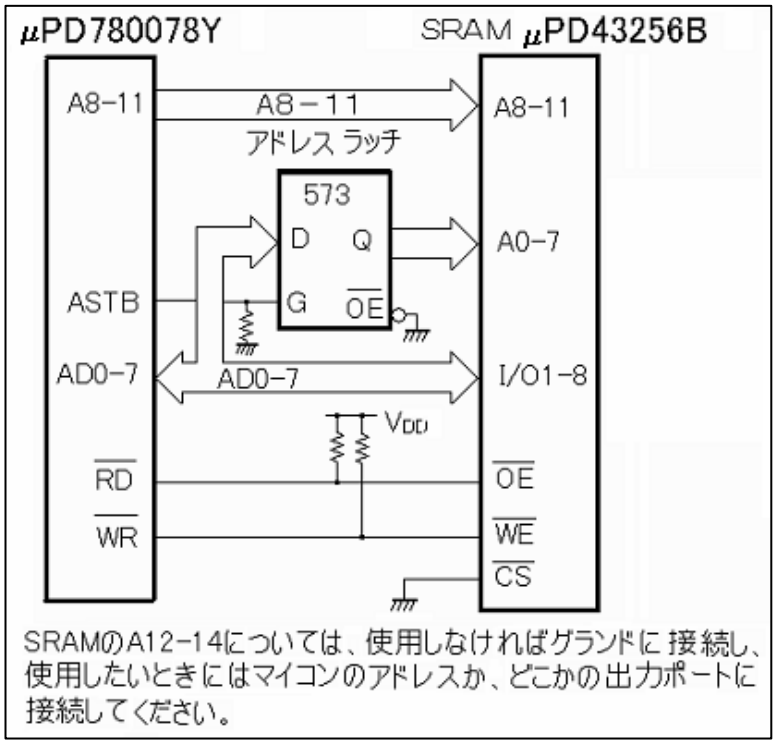
Q1. μPD780031A の電気的特性で「外部データ・アクセス」のタイミング・チャートに、RD, WR 信号が順次アクティブとなるように表現されています。これは、リード・アクセスかライト・アクセスでも、リード・モディファイ・ライトのアクセスをすることを表しているのですか。

A1. いいえちがいます。ユーザーズ・マニュアルの「第 20 章　外部デバイス拡張機能」の図 20-5、図 20-6 に記載されているように、リード時にはライト動作が、またライト時にはリード動作が、それぞれ発生しません。
ただし、この分の時間だけは必要とします(リード・モディファイ・ライトに必要な部分だけ、信号がアクティブになるとお考えください)。

SRAM を増設

Q1. μPD780078 に SRAM 等を外付けする場合の、接続方法を教えてください。

A1. 接続例としては、次図の回路かユーザーズ・マニュアルの図 20-8 をご参照ください。
次図の回路では、外部 RAM の容量は 32K バイトですが、4K バイト・モードでの接続例を示しています。
これ以上の容量を使用したい場合には、16K バイト・モードや 64K バイト・モードを利用するか、内部 ROM をできるだけ確保したい場合には、4K バイト・モードを使用して、出力ポートを RAM の A12-14 に接続すること(バンク・スイッチ)が考えられます。



内部高速 RAM 上でプログラム実行は可能か [78K0]

Q1. 内部高速 RAM で、プログラムを実行することは可能ですか。

A1. いいえ、内部高速 RAM からのフェッチはできません。

マイコンに内蔵されている EEPROM [μPD78986x]

Q1. μPD78E9860 に内蔵された EEPROM の値は、初期状態ではどうなっていますか。

A1. 現状、工場出荷時の EEPROM の値は"00"です。あくまでも現状の値で、変更される可能性もあります。

Q2. μPD789860 の内蔵 EEPROM(0xF800-0xF81F)領域へのデータの書き込みができません。

A2. 以下に、RAM 上のデータを EEPROM に書き込む手順を示します。ここでは、INTEE0 割り込みを使用せず、フラグ(EWST10)のポーリングで処理しています。

(1) EEWC10 レジスタの設定により、書き込みタイマと ERE10 ビットをセット(EEPROM の動作開始)する

(2) EEWC10 レジスタの EWE10 ビットをセット(書き込みモードを設定)する

(3) 1ms 以上ウエイトする(書き込み可能になるのを待つ)

(4) 1 バイト書き込む(これにより EWST10 は 1 になる)

(5) EWST10 が 0 になる(1バイトの書き込みが完了する)のを待つ

(6) 書き込みたいデータのアドレスを更新する

(4)～(6)を必要バイト数だけ繰り返す。

(7) EWE10 をクリアする(書き込みモードから抜ける)

以上の処理をアセンブリ言語で記述した、書き込みプログラムの例を示します。

このプログラム例では、HL に書き込みたい EEPROM のアドレスをセット、DE に書き込みたいデータのアドレスをセット、B に書き込みたいデータ数がすでにセットされている場合のサブルーチンになっています。必要に応じて、書き込んだデータを読み出して確認します。

```
;
;       Write EEPROM
;       Input parameter
;       HL        Data Pointer for EEPROM
;       DE        Data Pointer for Read
;       B         Data number
;

WRITE:  MOV       A,#4CH
        MOV       EEWC10,A          ; Set ERE10 & Set Timer to 3.71ms
        SET1      EWE10             ; Set EWE10
        PUSH      DE
        MOVW      DE,#279           ; 1ms
LOOP2:  DECW      DE
        MOV       A,D
        OR        A,E
        BNZ       $LOOP2
        POP       DE
LOOP3:  MOV       A,[DE]            ; Read Data
        MOV       [HL],A            ; Write Data to EEPROM
        INCW      HL
        INCW      DE
LOOP4:  BT        EWST10,$LOOP4
        DBNZ      B,$LOOP3
        CLR1      EWE10             ; Write end
        RET
```

このプログラムの考え方は、μPD789862 サブシリーズでも同じです。

マイコン内蔵メモリの特徴と使い方 [78K0、78K0S]

78K0 や 78K0S の 8 ビットのマイコン、特に All Flash の製品では、通常のメモリの中に、特殊なアドレスが予約されています。ここでは、そのちがいと使い方について簡単に説明します。

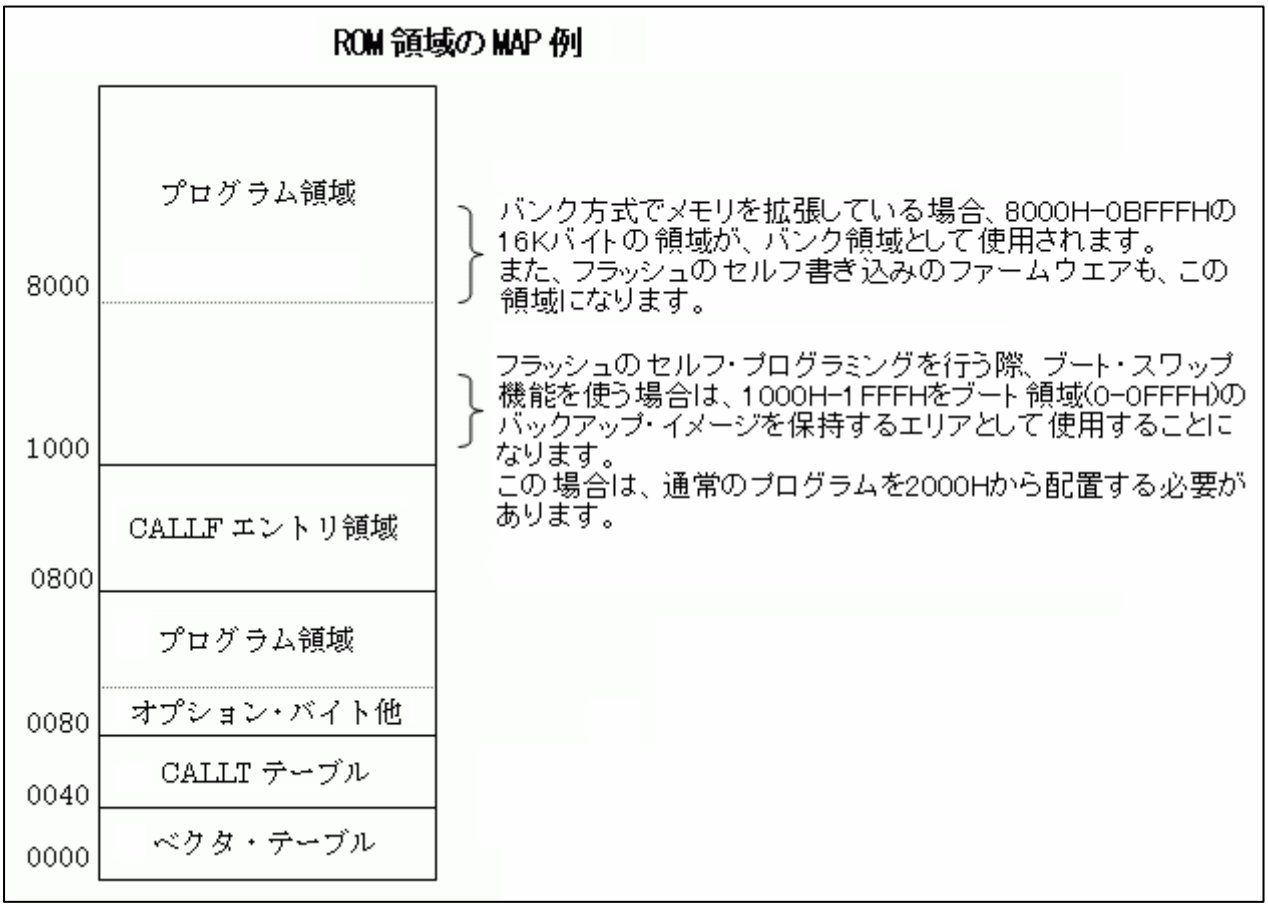
[内蔵メモリ領域]

これらのマイコンに内蔵され、通常使用するメモリとしては、プログラムによって次の 3 つがあります。

メモリ領域	特徴／機能
ROM	命令や固定データ用で、マスクかフラッシュで構成
高速 RAM	データ・アクセス専用のメモリ
内部拡張 RAM	データや命令を格納

(1) ROM

通常のプログラムや固定データに用いられますが、マイコンのアーキテクチャにより、特殊な領域が存在します。



ベクタ・テーブル [0000～003FH]

この領域には、ベクタ・テーブルが並んでいます。特に、0000 と 0001 番地には、リセット解除後に実行開始する命令のアドレス情報をベクタとしてセットします。残りは、使用する割り込みに対応して、割り込み処理プログラムの開始アドレス情報をベクタとしてセットします。通常は、下記の例のように、DW 擬似命令により、処理部のアドレスを記述します。ここで、START は実際のプログラムの実行開始アドレスです。使用しない割り込みのベクタには、RETI 命令を記述したダミー・エントリを準備しておき、そのアドレスを記述します。

```
VECT    CSEG    AT      0
                DW      START   ;00:リセット・エントリ
                DW      DUMYINT ;02:未使用割り込み用ダミー・エントリ
                DW      LVIINT  ;04:低電圧
                DW      POINT   ;06:エッジ検出割り込み
                DW      DUMYINT ;08:エッジ検出割り込み（未使用）
```

CALLT テーブル［0040H～007FH］

この領域には、1 バイトの CALL 命令である CALLT 命令を使用するときに、各 CALLT 命令に対応した処理プログラムの、開始アドレス情報をベクタとしてセットします。

たとえば、サブルーチン SUBADDR0, SUBADDR1 へのベクタを次のようにして設定しておくと、CALLT [SUB1] や CALLT [SUB2]と記述することで、SUBADDR0 や SUBADDR1 をサブルーチンとしてコールできます。C コンパイラでは、コード効率を良くするために、この機能を利用しています。

```
CALTTBL CSEG    CALLT0
SUB1:           DW      SUBADDR0        ;サブルーチン 1
SUB2:           DW      SUBADDR1        ;サブルーチン 2
```

オプション・バイト他［0080H～］

All Flash マイコンで、各種の設定情報やセキュリティ情報のための領域です。通常、プログラムで参照することはありません。設定したいデータは、DB 擬似命令で記述します。

```
OPTIONB CSEG    AT      80H
        DB      01111111B               ;WDT は最低速で動作
        DB      00000001B               ;POC は 2.7V 動作
        DB      0
        DB      0
        DB      0
```

プログラム領域［～07FFH］

通常のプログラムで使用できる領域です。プログラムは,、通常 CSEG で指定します。

CALLF エントリ領域［0800H～0FFFH］

2 バイトの CALL 命令である CALLF 命令で、実際に処理するプログラム（ハンドラ）のための領域です。CALLT とは異なり、ここには実際の処理部分が入ります。

CALLF 命令を使わない場合、通常のプログラム領域として使うこともできます。このとき、通常のプログラムと同様に、サブルーチン・プログラムを記述するだけです。そのように記述したサブルーチンをコールする際は、CALL 命令ではなく CALLF 命令を使用します。

プログラム領域［1000H～］

この領域は、個々のデバイスにより、大きさが異なります（アドレスは最大でも 0EFFF まで）。また、8000H 以降の領域は、フラッシュのセルフ・プログラミング時には、書き込み用のファームウエアに置き換わります。

(2) 高速 RAM

最大 1K バイトの領域が割り当てられています。この領域は、その最後のアドレスが固定されていて、内蔵される高速 RAM の容量により、開始アドレスが異なります。代表的なマイコンの高速 RAM を次表に示します。

シリーズ	デバイス	アドレス	容量(バイト)
78K0S/Kx1+	μPD78F9210～9212	0FE80H～0FEFFH	128
	μPD78F9221	0FE80H～0FEFFH	128
	μPD78F9222	0FE00H～0FEFFH	256
	μPD78F9232、9234	0FE00H～0FEFFH	256
78K0/Kx2	μPD78F0500	0FD00H～0FEFFH	512
	μPD78F0501	0FC00H～0FEFFH	768
	μPD78F0502、0503	0FB00H～0FEFFH	1024
	μPD78F0511	0FC00H～0FEFFH	768
	μPD78F0512～0515	0FB00H～0FEFFH	1024
	μPD78F0521	0FC00H～0FEFFH	768
	μPD78F0522～0525	0FB00H～0FEFFH	1024
	μPD78F0531	0FC00H～0FEFFH	768
	μPD78F0532～0535	0FB00H～0FEFFH	1024
	μPD78F0544～0547	0FB00H～0FEFFH	1024
78K0/Lx2	μPD78F0361	0FC00H～0FEFFH	768
	μPD78F0362～0397	0FB00H～0FEFFH	1024

なお、高速 RAM はスタックを含めたデータ専用領域です。ここからプログラムをフェッチすることはできません。また、高速 RAM はいくつかに分かれているので、スタック領域を設定する際には注意が必要です。

そのうちの 1 つは、**ショート・ダイレクト・アドレッシング(saddr、saddrp)領域**と呼ばれる領域です。これは 0FE20H～0FF1FH の 256 バイトの領域で、そのうち 0FE20H～0FEFFH が高速 RAM になります。この領域にバイト長の変数を配置するには、次のように SADDR 属性で DSEG を定義し、そこで DS 擬似命令で変数(領域)を定義します。

```
DO          DSEG    SADDR
DATA1:      DS      1
DATA2:      DS      1
DATA3:      DS      1
```

このように定義した変数は、次のように、汎用レジスタを介さずに、値を代入したり演算をしたり、汎用レジスタと同じように使用することもできます。

```
MOV         DATA1,#1
INC         DATA2
CMP         DATA3,#10
```

また、上記のようなバイト単位の変数ではなく、16 ビットの変数として使用する場合には、再配置属性を saddr ではなく saddrp とします。これにより saddrp に指定した変数は 16 ビットアクセスができるようになります。なお、この再配置属性を指定すると、変数は必ず偶数番地に配置されます。

```
DP          DSEG    SADDRP
DATA1P: DS  2
DATA2P: DS  2
DATA3P: DS  2
```

このように、ショート・ダイレクト・アドレッシング領域は便利に使えるので、スタックのような単純な機能に使うのはもったいないです。
もう 1 つは、0FEE0～0FEFFH の 32 バイトの領域です(78K0 の仕様で、78K0S では該当しません)。この領域には、78K0 の**汎用レジスタ**が配置されています。78K0 では、割り込み処理を高速化するために、割り込みで使用するレジスタを切り替える機能(レジスタ・バンク機能)があります。この機能を利用すると、割り込み処理で、レジスタをスタックに退避させたり、逆に復帰させたりする手間が必要なくなります。
また、高速 RAM は、セルフ・プログラミングでは、エントリ RAM やデータ・エリアとしても使われます。
なお、SADDR 領域に限定せずに、高速 RAM にデータ領域を配置する場合には、DSEG 指定で、再配置属性を IHRAM に指定します。次に例を示します。

```
RAMD        DSEG    IHRAM
DATA4:      DS      2
```

(3) 内部拡張 RAM

プログラムの実行が可能な RAM 領域です。通常のデータやプログラムをこの領域に配置することはできますが、スタックを設定することはできません。この領域には、通常は、それほどアクセス頻度が高くない変数や、データ量が大きな変数(配列など)を配置します。
また、一部のデバイスでは、ROM コレクション機能で、プログラムを実行するためにも使用されます。
この領域は DSEG 指定で、再配置属性を IXRAM に指定します。次に例を示します。

```
LSPRAM2     DSEG    IXRAM
DATA_AREA1:
            DS      20H
DATA_AREA2:
            DS      20H
DATA_AREA3:
            DS      20H
```

1.2.13 メモリ・バンク

メモリ・バンク [78K0/Kx2]

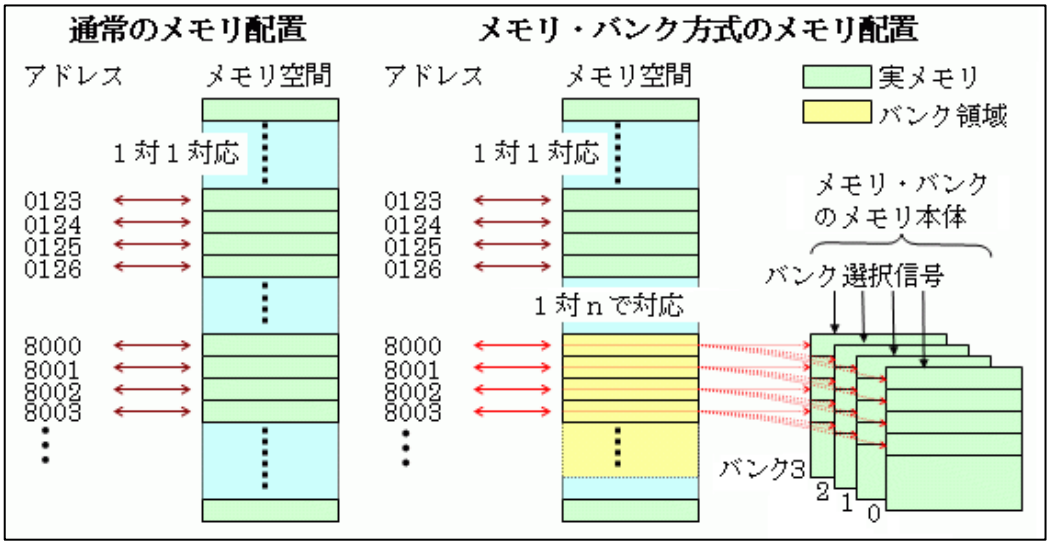
78K0/Kx2 で導入されたメモリ・バンクの機能について紹介します。

【はじめに】

8ビット・マイコンでは、そのアーキテクチャから、16 本のアドレス信号で、64Kバイトまでのメモリを使うのが一般的です。マイコンの基本的なアーキテクチャ(命令体系)を変更すれば、64kバイト以上のメモリを使うことができますが、それでは全く別のマイコンになってしまいます(FAQの「マイコンとメモリ」もご参照ください)。

【メモリ・バンク】

従来のマイコンの仕様を大きく変更することなく、64Kバイト以上のメモリをアクセスするために、バンク・スイッチングやメモリ・バンクと呼ばれる方法が用いられます。これは、マイコンが直接扱えるメモリの範囲(この場合は 64K バイト)はそのままで、一部の領域にメモリをいくつか重ねているものです。次にイメージを示します



通常のメモリ配置では、アドレスに対してメモリが 1 対 1 に対応していますが、右側に示されたメモリ・バンク方式の場合、バンク領域（ここでは 8000-BFFF の 16K バイトとします）では、1 つのアドレスに複数のメモリ（この図では 4 バンク）が対応しています。1 アドレスに対応するメモリの数を増やせば、扱えるメモリを増やすことができます。ただし、1 つのアドレスに対して複数のメモリが対応しているといっても、すべてがアクセスできる状態になっているわけではありません。このバンク領域は、前もってバンク選択信号で使用する（アクセスできる）メモリ・バンクを 1 つ指定しておきます。この状態で、バンク領域をアクセスすると、指定しておいた 1 つのメモリ・バンクを選択してアクセスできます。このときの手順は、次のようになります。

（1）使用したいバンクを選択 → バンク選択信号で、メモリ・バンクの 1 つが選択される

（2）バンク領域をアクセス → (1)で指定したメモリ・バンクの、該当アドレスのメモリをアクセスする

このように、2 つの手順を組み合わせれば、任意のメモリ・バンクの、任意アドレスのメモリをアクセスできます。ただしこれは、プログラムが通常のメモリで動作中に、メモリ・バンクをアクセスする方法です。プログラムがメモリ・バンクで動作中には、別のメモリ・バンクを直接アクセスすることができません。別のメモリ・バンクをアクセスするには、プログラムを通常のメモリに移して、メモリ・バンクを切り替えてから、アクセスする必要があります。

【メモリ・バンクの使い方】

メモリ・バンクは、マイコンの仕様をあまり変更せずにメモリを拡張する方法ですので、マイコンを使う上では、頻繁に使うものではありません。特にメモリ・アクセスの効率を考えると、通常のメモリとメモリ・バンクの用途を明確にしておき、頻繁にバンクを切り替えないようにしてください。メモリの使い分けの目安は、次のとおりです。

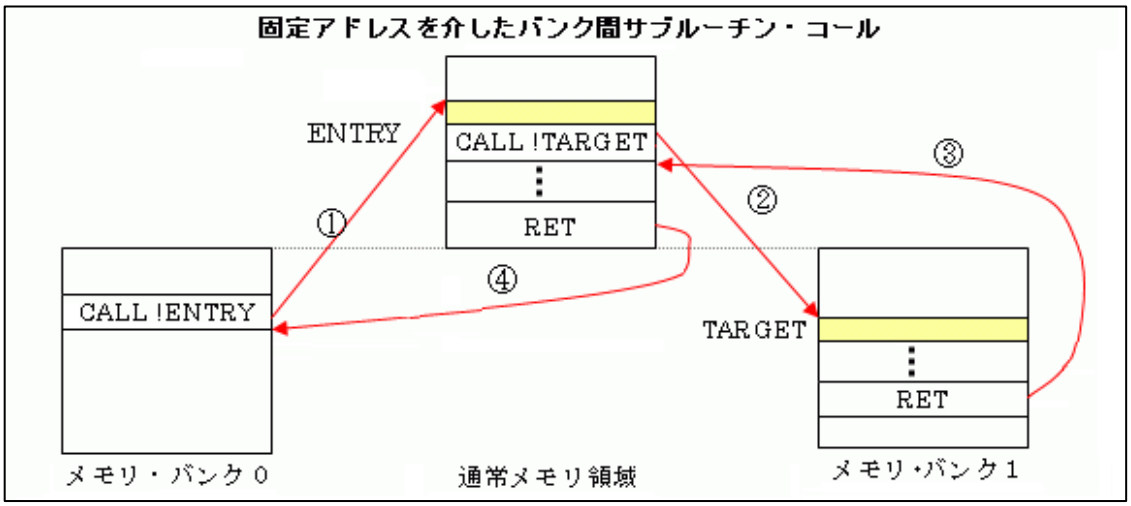
通常メモリ領域 ：プログラム本体、共通的に使用するルーチンやデータ
割り込みのように、プログラム実行とは非同期の処理
処理時間に制限のある処理

バンク領域 ：バンク領域内でクローズする処理とそのデータ
使用頻度の低い処理やデータ
処理時間に余裕がある処理
アクセス時間に制限が少ないデータ

この考え方に従うと、メモリ・バンクをまたぐような処理は限られてきます。そのため、メモリ・バンクをまたぐ処理が必要なときだけ、通常メモリの固定されたアドレスにバンクを切り替える処理を準備して、そこを使って切り替えます。この方法は、比較的単純なプログラムで済みます。この方式を用いてメモリ・バンク0からメモリ・バンク1のサブルーチンをコールする例について、処理の概要を次図に示します。

この例では、メモリ・バンクを越えて、メモリ・バンク 1 の TARGET というサブルーチンをコールします。通常メモリ

領域に、このサブルーチンを呼びたすための踏み台になる処理として、ENTRY というサブルーチンを用意します。ここでメモリ・バンクを切り替え、サブルーチン TARGET をコールし、メモリ・バンクを元に戻してリターンします。
この状態で、①メモリ・バンク 1 から通常メモリ領域の ENTRY をサブルーチン・コールすると、そこから②メモリ・バンク1の TARGET がコールされます。TARGET での処理が完了すると、③いったん ENTRY に戻ってきて、そこから④メモリ・バンク0に戻ります。このように、サブルーチンごとに、通常メモリに踏み台を準備すれば、他のメモリ・バンクのサブルーチンを使用できます。



実際に使用する場合、通常メモリに準備する具体的なプログラム例は以下のようになります。なお、作業用（データ保持やデータ受け渡し）に saddr 領域のメモリを使用します。saddr 領域については、「マイコンとメモリ」をご参照ください。

(1) プログラム例 1

このプログラムでは、レジスタの内容を破壊しないように、saddr 領域の RAM1 バイト（R_BNKN）を作業用に使用しています。

```
;
;       入力パラメータ
;       なし
;       出力パラメータ
;       なし
;
ENTRY:
        MOV     R_BNKN, A               ;A レジスタを保存
        MOV     A, #BANKNUM TARGET      ;サブルーチンのバンク番号を指定
        XCH     A, BANK                 ;戻り先のバンク読み出しとサブルーチンの
                                        ;バンク番号セット
        XCH     A, R_BNKN               ;A レジスタ復帰、元バンク・セーブ
        CALL    !TARGET                 ;実際のサブルーチンをコール
        XCH     A, R_BNKN
        MOV     BANK, A                 ;戻り先バンク設定
        MOV     A, R_BNKN
        RET
```

(2) プログラム例 2

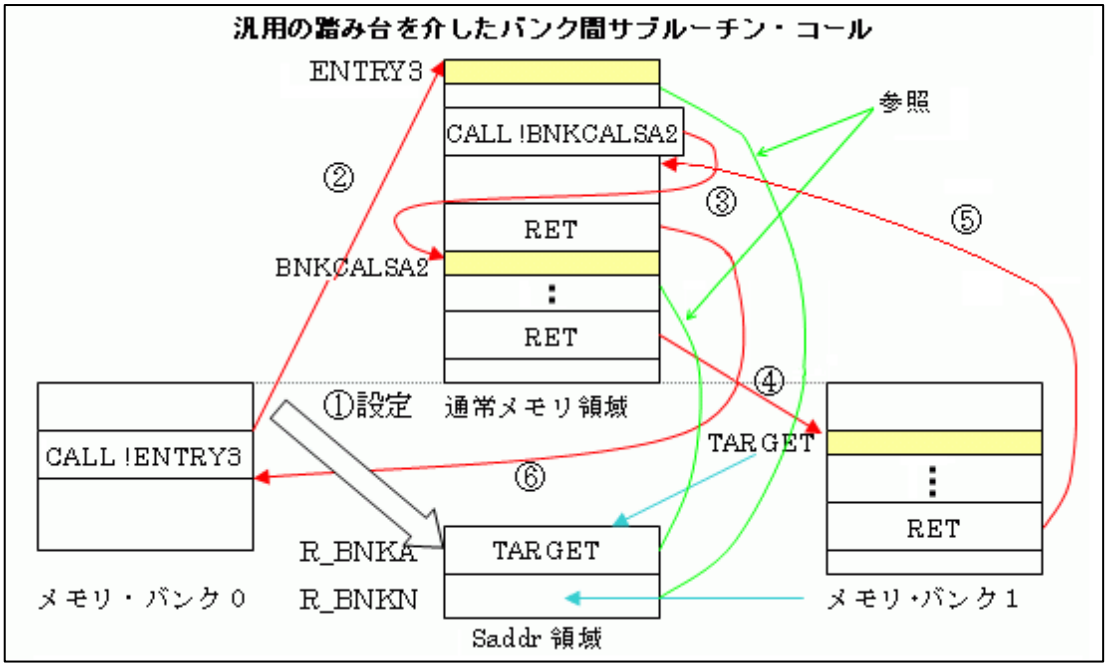
どうしても、複数のメモリ・バンク間を遷移する必要がある場合、上記のプログラムを複数並べる必要があります。

しかし、例 1 のプログラムは、別バンクのサブルーチンをコールする際に、戻り先のバンク番号を saddr 領域の作業用メモリ(R_BNKN)に保存するので、バンク間の多重の呼び出しには対応できません。このような場合には、以下に示すように、破壊したくない値(戻り先のバンク番号)をスタックに退避させて、処理を変更してください。今回は AX レジスタをワークとして使用するので、作業用に saddr 領域に 2 バイトの RAM(RSAVEAX)を準備します。

```
;
;       入力パラメータ
;       なし
;       出力パラメータ
;       なし
;
ENTRY2:
        MOV     RSAVEAX, A              ;A レジスタを保存
        MOV     A, #BANKNUM TARGET
        XCH     A, BANK                 ;戻り先のバンク読み出しとサブルーチンの
                                        ;バンク番号セット
        PUSH    AX                      ;戻り先バンクをスタックにセーブ
        MOV     A,  RSAVEAX             ;A レジスタ復帰
        CALL    !TARGET                 ;実際のサブルーチンをコール
        MOVW    RSAVEAX, AX             ;AX レジスタセーブ
        POP     AX
        MOV     BANK, A                 ;戻り先バンク設定
        MOVW    AX, RSAVEAX             ;AX レジスタ復帰
        RET
```

(3) プログラム例 3

メモリ・バンク間を行き来するサブルーチン・コールが多くなると、その数だけ踏み台となるプログラムを準備するのは、効率的ではありません。その場合には、以下のような汎用のプログラムを使います。このプログラムは、saddr 領域の R_BANK にコールするサブルーチンのバンク番号、R_BNKA にアドレスを設定した状態でコールします。また、処理の中で、作業用に 2 バイトの RAM(RSAVEAX)を使用します。このプログラムは、サブルーチンのコールでも、逆にサブルーチンからの復帰でも、レジスタの値は破壊しません。そのために、ダミー・コールを用いて、2 段階でサブルーチン・コールを処理しています。



①で、コールするサブルーチンのバンク番号やアドレスを saddr 領域に設定しておきます。その後②で、踏み台プログラムをコールします。踏み台プログラムでは、saddr 領域のバンク番号を BANK レジスタにセットしたあとで、③あと処理部（BNLCALSA2）をサブルーチン・コールします。あと処理部では、サブルーチンのアドレスをスタックに設定し、④RET 命令により、指定されたサブルーチンに分岐します。
サブルーチンでは必要な処理を行い、⑤RET 命令により、踏み台プログラムに戻ります。踏み台プログラムでは、BANK レジスタにコール元のバンク番号を戻して、⑥RET 命令により、コール元のプログラムに戻ります。
具体的なプログラムは、以下のとおりです。

```
;
;           入力パラメータ
;           R_BNKN：参照先のバンク番号
;           R_BNKA：参照先のアドレス
;           出力パラメータ
;           なし
;
ENTRY3:                                 ;メモリ・バンク間コール処理ルーチン
★          MOVW    RSAVEAX, AX         ;AX レジスタをセーブ
           MOV     A, R_BNKN           ;コール先バンク番号を取得
           XCH     A, BANK             ;バンク変更と戻り先バンクを取得
           PUSH    AX                  ;戻り先のメモリ・バンク番号をスタックに
           CALL    !BNKCALSA2          ;コール先に分岐するためのダミー・コール
           MOVW    RSAVEAX, AX         ;AX レジスタをセーブ
           POP     AX                  ;戻り先バンク番号を取得
           MOV     BANK, A             ;戻り先のメモリ・バンク番号を指定
           MOVW    AX, RSAVEAX         ;AX レジスタを元に戻す
           RET                         ;コール元にリターン
BNKCALSA2:
           MOVW    AX, R_BNKA          ;コール先のアドレス指定
           PUSH    AX                  ;コール先をスタックにセット
           MOVW    AX, RSAVEAX         ;元の AX レジスタの値を復帰
           RET                         ;コール先に分岐
```

このままでは、この踏み台を使うたびに、サブルーチンのバンク番号とアドレスを設定する処理を記述しないといけません。これを避けるには、以下のようにマクロを定義します。

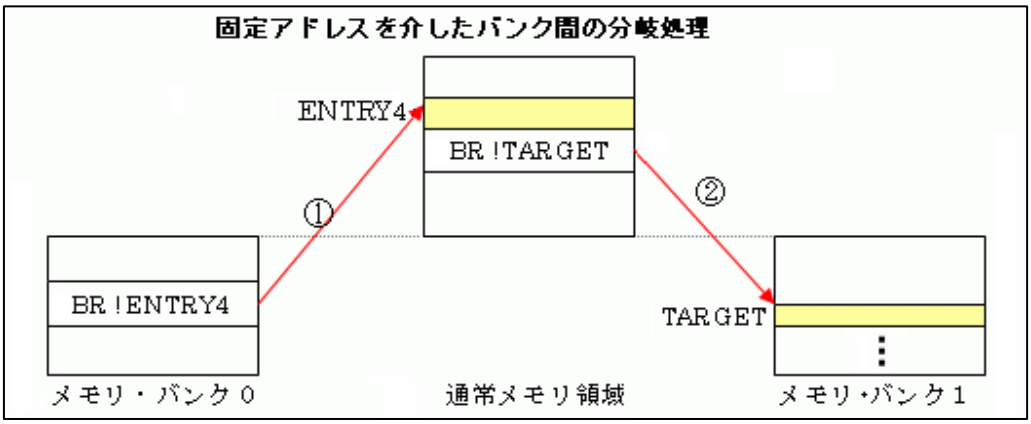
```
BNKCAL     MACRO    PARA1
           MOV      R_BNKN, #BANKNUM PARA1 ;参照先のメモリ・バンク番号を格納
           MOVW     R_BNKA, #PARA1         ;参照先のアドレスを格納
           CALL     !ENTRY3
           ENDM
```

これで、3 命令を記述する替わりに、新たにマクロ定義した BNKCAL 命令を用いて、次のようにプログラムを記述するだけで、簡単にこの踏み台プログラムを使うことができます。

```
BNKCAL     TARGET
```

(4) メモリ・バンク間のプログラム分岐処理

以上、メモリ・バンク間のサブルーチン・コールを説明しましたが、次に、分岐のプログラム例を示します。



固定アドレスへの分岐処理では、通常メモリ領域に、単純に次の 2 命令を準備します。単純な処理なので、これを必要なだけ並べても実用的です。分岐元のプログラムは、単純にこの踏み台プログラムに分岐するだけです。

```
;
;       入力パラメータ
;       なし
;       出力パラメータ
;       なし
;
ENTRY4:
        MOV     BANK,#BANKNUM TARGET
        BR      !TARGET
```

参考に、踏み台としての汎用のメモリ・バンク間の分岐プログラムの例を以下に示します。かなりの命令が必要なことと、これを使用するには、事前に分岐先のバンク番号とアドレスを saddr 領域に設定する必要があることから、とても効率的とは言えないので、あくまでも参考にとどめてください。

```
;
;       入力パラメータ
;       R_BNKN：参照先のバンク番号
;       R_BNKA：参照先のアドレス
;       出力パラメータ
;       なし
;
ENTRY5:
        MOVW    RSAVEAX,AX              ;AX レジスタをセーブ
        MOV     A,R_BNKN                ;
        MOV     BANK,A                  ;分岐先のメモリ・バンク番号を指定
        MOVW    AX,R_BNKA               ;分岐先のアドレス指定
        PUSH    AX                      ;分岐先アドレスをスタックにセット
        MOVW    AX,RSAVEAX              ;分岐元の AX レジスタ復帰
        RET                             ;分岐先に分岐
```

この場合にも以下のマクロを定義しておけば、「BNKBR TARGET」と記述するだけで済みます。

```
BNKBR   MACRO   PARA1
        MOV     R_BNKN,#BANKNUM PARA1   ;参照先のメモリ・バンク番号を格納
        MOVW    R_BNKA,#PARA1           ;参照先のアドレスを格納
        BR      !ENTRY5
        ENDM
```

(5) メモリ・バンク間のデータ参照処理

次に、メモリ・バンクをまたいだ、データ参照処理のプログラム例を示します。この例では、レジスタを用いて、必要なパラメータを受け渡しています。

```
;
;       入力パラメータ
;       A レジスタ：参照先のバンク番号
;       HL レジスタ：参照先のアドレス
;       出力パラメータ
;       A レジスタ：参照結果
;
ENTRY6:
        XCH     A,BANK              ;参照元と参照先のバンク番号の交換
        MOV     R_BNKN,A            ;参照元のメモリ・バンク番号を退避
        MOV     A,[HL]              ;目的の値をリード
        XCH     A,R_BNKN            ;参照元のメモリ・バンク番号を取得
        MOV     BANK,A              ;参照元のメモリ・バンク番号を指定
        MOV     A,R_BNKN            ;参照結果を A レジスタに復帰
        RET
```

汎用のサブルーチン・コールや、分岐と入力パラメータを統一した場合の例も、参考に示しておきます。

```
;
;       入力パラメータ
;       R_BNKN：参照先のバンク番号
;       R_BNKA：参照先のアドレス
;       出力パラメータ
;       A レジスタ：参照結果
;
ENTRY7:                             ;メモリ・バンク間参照用サブルーチン
        PUSH    HL                  ;HL レジスタの内容を退避
        MOV     A,R_BNKN            ;
        XCH     A,BANK              ;参照元と参照先のバンク番号の交換
        MOV     R_BNKN,A            ;参照元のメモリ・バンク番号を退避
        XCHW    AX,HL               ;X レジスタをセーブ
        MOVW    AX,R_BNKA           ;
        XCHW    AX,HL               ;参照先のアドレスを指定
        MOV     A,[HL]              ;目的の値をリード
        XCH     A,R_BNKN            ;参照元のメモリ・バンク番号を取得
        MOV     BANK,A              ;参照元のメモリ・バンク番号を指定
        MOV     A,R_BNKN            ;
        POP     HL                  ;HL レジスタの内容を復帰
        RET                         ;復帰
```

1.2.14 ROM 化

ROM の未使用領域の充填値

Q1. マスク化したとき、プログラムの未使用領域に、何らかのコードが書き込まれる可能性はありますか。

A1. はい、あります。

万一、バグなどで未使用領域のデータを読んだ場合は、何らかの値が読まれます。その領域がコードとして生成されていない領域の場合には、デバッグ時の値とマスク ROM になった場合の値が、異なることが考えられます。この場合、読み込まれた値とプログラムの動作が異なると、マスク ROM とデバッグ時で、動作が異なります。

これは**絶対に避ける**必要があります。念のために、コード領域以外も、一定の値で充填しておけば、マスク ROM とデバッグを同じ動作にできます。マスク発注後のトラブルを避けるためにも、-U オプションで充填して評価、発注されることをお勧めします。

ROM の未使用領域を特定値で埋める

Q1. プログラムの最終アドレスから ROM の終了アドレスまでを特定データで埋めようとしています。
DB 疑似命令で次の記述をすると、エラーになってしまいました。

```
DB ( 7FFFH - !$ )
```

A1. アセンブルの段階で、このようなことはできません。

その代わり、未使用領域を特定のコードで埋める機能が、オブジェクト・コンバータに準備されています。

この機能(-U オプション)を使用すれば、未使用領域を任意のコードで埋めることができます。

通常、マスク ROM を発注される場合には、この機能を使用して、未使用領域を埋めることをお勧めします。

詳細は、操作編マニュアルの「オブジェクト・コンバータ」の章をご参照ください。

フラッシュ品で開発してマスク化する場合のデバイス・ファイル

Q1. 開発時には μPD78F0034 を使用し、μPD780021 でマスク化します。IMS でメモリ容量を変更していますが、デバイス・ファイルを μPD78F0034 にすると、RAM のメモリ・マッピングが μPD780021 と同じになりません。このような場合の、デバイス・ファイルの選び方を教えてください。

A1. フラッシュ品のデバイスで、マスク品のプログラムを開発するときには、マスク品のデバイス・ファイルを使用してください。その上で、IMS を対象のマスク製品に合わせてください(この場合は 42H)。

ご質問の場合は、μPD780021 のデバイス・ファイルをご使用ください。

ROM 化のとき、IMS(メモリ・サイズ切り替えレジスタ)設定は残しておいていいか

Q1. フラッシュ品で開発してマスク化する場合、IMS レジスタへの設定命令を残したままでいいですか。

A1. 問題ありません。マスク品には、IMS レジスタが存在しませんので、設定をしても何も影響を受けません。

最終的に使用するマスク品のメモリ・サイズと異なる IMS 設定としたときの問題

Q1. フラッシュ品で開発する場合、最終的に使用するマスク品のメモリ・サイズと違う IMS 設定で動作させていいですか(IMS 設定を初期値のまま使用)。
プログラム・サイズは、マスク品の ROM に収まる範囲です。

A1. はい、問題があります。プログラムで明示的に表現された部分は問題がなくても、想定外の状態になって、本来はないメモリ領域を参照した場合などに、動作が異なることが考えられます。
プログラムが完全で、そのような動作が全く発生しないのであれば、特には問題ありませんが、通常はお勧めできません。
また、外部拡張での動作にも影響を与えますし、禁止設定は、デバイスとしてもテストしているわけではありませんので、どのような動作になるかは保証ができません。本来のデバイスに合った設定をしてください。

OC78K の-R オプション

Q1. オブジェクト・コンバータの-R オプションの必要性・動作原理は、どういうものですか。

A1. -R オプションは、リンカにでアドレスが決定されたオブジェクトを HEX ファイルに出力する際、アドレス順に並べ直すものです。
ROM 受注システムでは、お客様に依頼されたオブジェクトのチェックの1つとして、アドレスが順に並んでいるかどうかを確認しており、並んでなければ、エラーとして処理を受け付けません。これに対応するためのオプションが-R です。

Q2. -R オプションがなくても、マスク発注はできますか。

A2. たまたまアドレスが昇順で出力される可能性はありますが、不確実ですので、-R オプションは必ず指定してください。

1.2.15 ROM コレクション

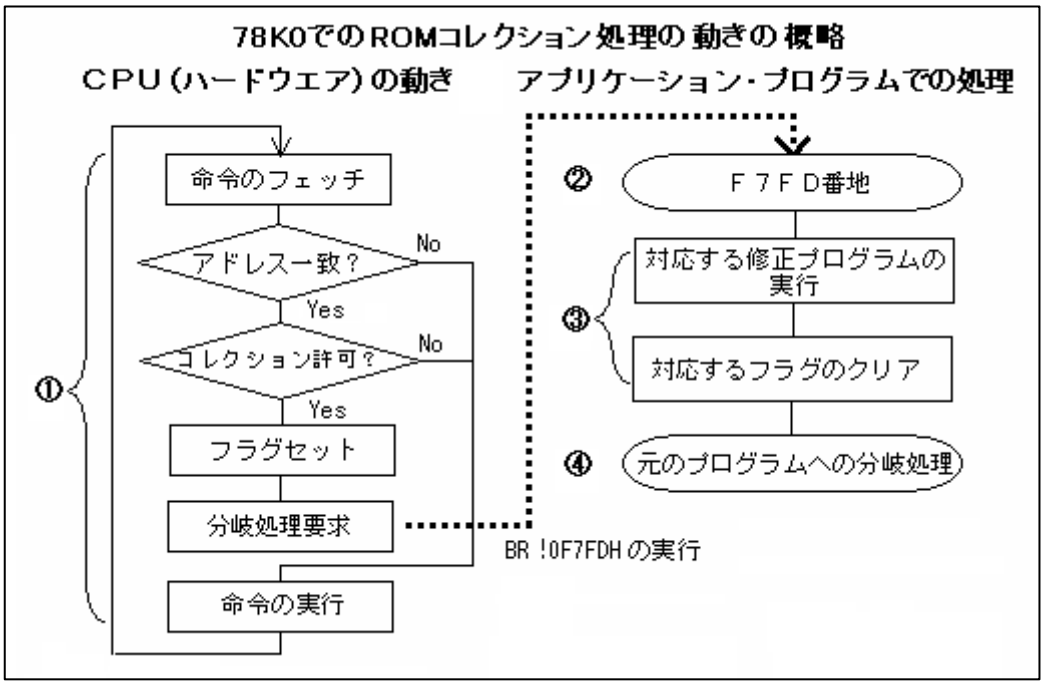
ROM コレクション [78K0]

Q1. 78K0 の ROM コレクションは、どのような機能ですか。

A1. すでにマイコン内の ROM に書き込まれたプログラムにパッチを当てて、プログラムを修正する手段として、ROM コレクション機能があります。この機能は、既存のプログラムを修正できるという便利な機能ですが、いつでも無条件で使用できるものではありません。
この機能を使用するには、事前に、そのための処理の一部をプログラム中に埋め込んでおく必要がありますし、代わりに実行させるプログラムを読み込むための、外部メモリも必要になります。
ROM コレクションの動作や、そのために必要なことなどについて、以下に説明します。

(1) ROM コレクションの基本的な動作

ROM コレクションでは、CPU (ハードウエア)が自動的に実行する部分と、アプリケーション側で対応をすべき部分とが混在しているので、これらを区別して考える必要があります。
すでに、必要な設定が行われた状態での、ROM コレクションの動きを次に示します。



①プログラム実行中、CPU が命令をフェッチする際に、そのフェッチ・アドレスと、CORADn レジスタに設定されている ROM コレクション・アドレスとの比較を行って、一致した場合には本来の命令ではなく、分岐命令（BR !0F7FDH 命令）を実行します。
置き換えられた分岐命令の実行により、それまでの処理を中断し、0F7FDH 番地に分岐します。このとき、一致が発生したコレクション・アドレスに対応する CORCN レジスタのビット(CORST0, 1)をセットします。
ここまでが、ROM コレクションのハードウエアで処理される内容で、これ以降は、アプリケーション・プログラムで処理する内容です。

②BR !0F7FDH 命令により、内部拡張 RAM のうしろに分岐してきますので、 実際に ROM コレクション処理を行うアドレスへ分岐させる必要があります。

③ROM コレクションは、2 箇所まで設定できるので、どのアドレスでの一致による処理かを CORST0, 1 ビットで判断して、対応したプログラム(パッチ処理)を実行します。
対応した CORST ビットをクリアすることを忘れないでください。

④パッチ処理が完了したら、元のプログラムに分岐します。

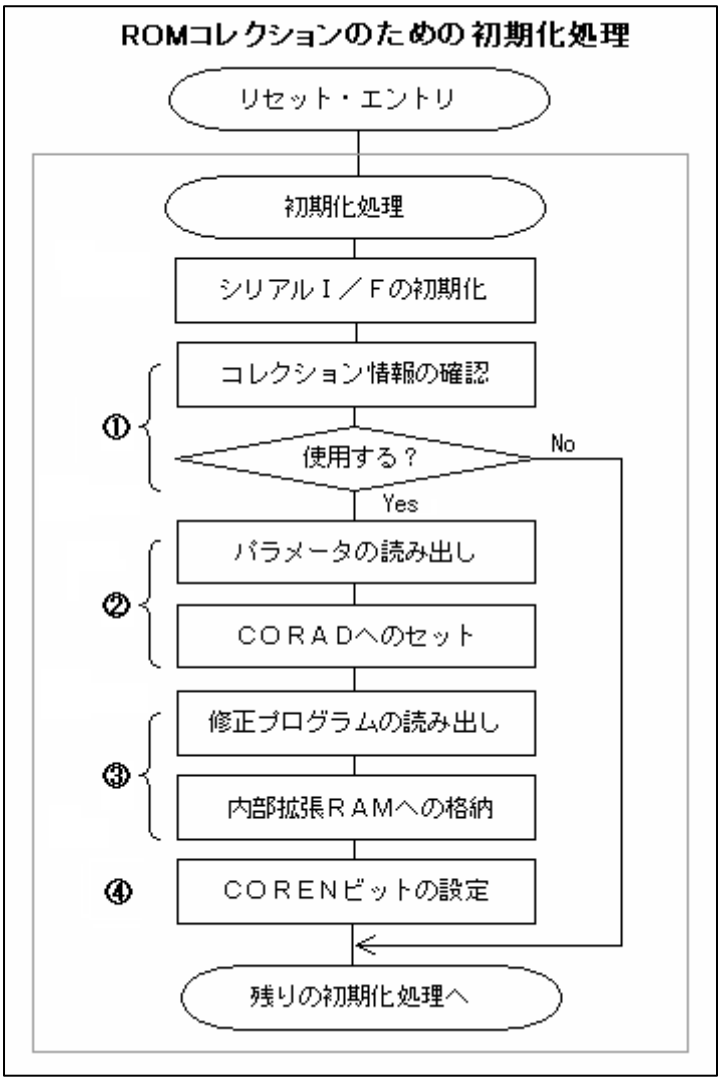
以上のように、ROM コレクション用のハードウエアと、事前にアプリケーション・プログラムに組み込んである処理プログラムを用いて、修正用プログラムを実行させることで、パッチ処理を行えるようになっています。

(2) ROM コレクションを行うために準備するハードウエア

ROM コレクションを行うには、修正プログラムを読み込むためのハードウエアが必要になります。
シリアル・インタフェースで制御する EEPROM を接続し、その中に修正用プログラムを書き込んでおきます。
この場合、初期化処理の中で、そのプログラムをシリアル・インタフェースを介して読み出し、内部拡張 RAM に書き込んでおく必要があります。

(3) 初期化プログラムでの対応

ROM コレクションを使用するには、上述のハードウエアを準備し、初期化プログラムで必要な設定をしておく必要があります。



ROM コレクションを使用するには、初期化処理で、上記フローの処理を行う必要があります。

以下、このフローに沿って説明します。なお、外部に接続したメモリを使用できるように、最初にシリアル・インタフェースの初期化を行っておいてください。

①コレクション情報の確認

入力ポートを準備するか、外部に接続したメモリの特定のアドレスに ROM コレクションを使う必要があるかどうかのフラグを準備しておき、それを確認することで処理を行います。

ROM コレクションを使用する必要がない場合には、②～④の処理は飛ばして、次の処理に移ります。

②パラメータの設定

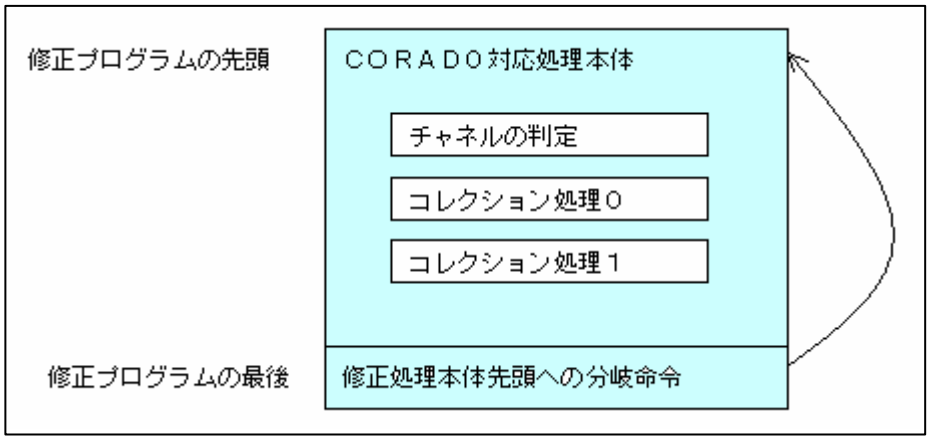
ROM コレクションで必要なパラメータ(コレクション・アドレス)を外部に接続したメモリから読み込み、CORAD レジスタへセットします。

③修正プログラムの読み込み

引き続き、ROM コレクションで代わりに実行させたいプログラムを外部に接続したメモリから読み込み、実際にプログラムを実行させる内部拡張 RAM へ書き込みます。

実際にプログラムを実行するのは、内部拡張 RAM なので、内部拡張 RAM のうしろ数十バイトを ROM コレクション用として、事前に確保しておいてください。

なお、②の処理と前後することもあり得ます。



④COREN の設定

最後に、使用したい ROM コレクション・アドレスに対応した CORCN レジスタの COREN0, 1 をセットすれば、ROM コレクション処理が有効になります。

以上の処理で、②～④の部分を具体的にプログラムにすると、以下の例のようになります。
このプログラム例では、シリアル・インタフェースからの1バイトの読み出しをサブルーチン GET1CH で実行して、 結果は A レジスタに入るものとします(HL レジスタは影響を受けないものとします)。
また、ROM コレクション用の内部拡張 RAM の先頭アドレスを PATCH とします。
ただし、ROM コレクションでは、内部拡張 RAM を 256 バイト以下しか使用しないものとします。

```
;
; ROM コレクション関係初期化
;
            MOV     CORCN,#0          ; ROM コレクション禁止
            MOVW    HL,#PATCH         ; コピー先アドレスを設定
LOOP1:
            CALL    !GET1CH           ; 1 バイトを読み出す
            MOV     [HL],A            ; 内部拡張 RAM への書き込み
            INC     L                 ; アドレス更新
            BNZ     $LOOP1            ; 0F7FFH まで完了するまでループ
            MOVW    AX,!PATCH         ; チャネル 0 の設定アドレスをリード
            MOVW    CORAD0,AX         ; アドレス設定
            PUSH    AX                ; チャネル 0 のアドレスをセーブ
            MOVW    AX,!PATCH+2       ; チャネル 1 の設定アドレスをリード
            MOVW    CORAD1,AX         ; アドレス設定
            OR      A,X               ; アドレス・チェック
            BZ      $NEXT1            ; アドレスが 0000 なら無視
            SET1    COREN1            ; チャネル 1 を有効に
NEXT1:
            POP     AX                ; チャネル 0 のアドレスを復帰
            OR      A,X
            BZ      $NEXT2            ; アドレスが 0000 なら無視
            SET1    COREN0            ; チャネル 0 を有効に
NEXT2:
; ROM コレクション初期化完了
```

(4) ROM コレクションの前処理

アドレスが一致して、修正したい命令の代わりに BR !0F7FDH 命令が実行され、0F7FDH 番地へ分岐します。

0F7FFH 番地までしか命令はおけませんので、ここには分岐命令を配置し、実際の ROM コレクションで実行する処理の先頭に分岐させます。

(5) ROM コレクションの修正用プログラムの構成

ROM コレクションで使用する修正プログラムや、パラメータについて説明します。
外付けのメモリには、ROM コレクションを設定するアドレスを2つ、修正用のプログラムとエントリ用の分岐命令のために用意する必要があります。
簡略化のために、領域として 64 バイトを確保したとします。
この 64 バイト分のデータを内部拡張 RAM の最後の部分(0F7C0H～0F7FFH)に転送するものとします。
このとき、ROM コレクションしたいアドレスの 1 つ目が PATCH0 で戻り番地を RETADR0、2 つ目を PATCH1 で戻り番地を RETADR1 とすると、以下のようなプログラムを準備すればいいことになります。

DW 擬似命令で、PATCH0 を ADDR0 に、PATCH1 を ADDR1 に定義します。
ROM コレクションの初期設定で、ADDR0 と ADDR1 の内容を確認し、その内容が 0000 なら対応したチャネルの ROM コレクションを使用せず、0000 以外なら対応したチャネルの ROM コレクションを使用する(CORCN レジスタの該当するビットをセットする)ようにします。

```
PATCH       CSEG    AT   0F7C0H
ADDR0:      DW      PATCH0          ; コレクションを設定したい番地
ADDR1:      DW      PATCH1          ; コレクションを設定したい番地

STARTADR:
            BT      CORCN.0, $PROCE0
PROCE1:
            CLR1    CORCN.2         ; CONST1 に対応した処理
;           <ここに必要な処理を記述>
            BR      !RETADR         ; 処理終了してメインに分岐
PROCE0:
            CLR1    CORCN.0         ; CONST0 に対応した処理
;           <ここに必要な処理を記述>
            BR      !RETADR0        ; 処理終了してメインに分岐

ENTRY       CSEG    AT   0F7FDH
            BR      !STARTADR
            END
```

(6) ROM コレクションの競合について

通常は、ROM コレクションが同時に複数発生することはありません。
しかし、割り込み処理にも ROM コレクションを設定した場合には、ROM コレクション処理中に割り込みが発生し、割り込み処理の中で別の ROM コレクションが発生することが考えられます。
このような場合には、ROM コレクション対象のプログラムの優先度の高いチャネルの CORST ビットから順に確認を行うことで、ROM コレクションのチャネルに優先度を設定できます。
このように、処理に優先度をつけることで、割り込みにも ROM コレクション処理を行うことができます。

1.2.16 フラッシュ書き込み

フラッシュ版の FLSPM0 の設定は省略していいか

Q1. 同じプログラムで、HEX ファイルをフラッシュ用(μPD78F0983)と、マスク用(μPD780983)に作成しています。プロセッサの指定を(0983)とし、FLSPM0 の設定は省略(通常)してアセンブルした HEX ファイルを、フラッシュ品に書き込んだ場合、動作上の問題はありますか。

A1. いいえ、問題ありません。FLSPM0 は、μPD78F0988 でフラッシュのセルフ書き込みを行うためにだけ、使用するものです。セルフ書き込みを行わない場合には、リセット後の初期値から変更する必要はありません。

オンボード書き込みのファームウエア

Q1. オンボード書き込みについて、マイコン自体には、どのようなファームウエアが組み込まれていますか。

A1. オンボード書き込みは、単体で書き込む動作をターゲット上で行うだけです。デバイスから見ると、単体書き込みと差がありません。これらの書き込みに必要な処理(通信条件の設定、クリア、書き込み、ベリファイなど)は、すべて内蔵のファームウエアが行っています。

誤動作による書き込み動作を終了する方法

Q1. 誤動作(誤って V_{PP} 端子に信号が入力された場合など)で、フラッシュ・メモリ書き込み動作に入ってしまった場合、抜け出す方法を教えてください。

A1. 書き込み状態から抜ける方法は、V_{PP} 端子をロウ・レベルにした状態で、リセットをかけて解除するだけです。これは、リセットによる通常の動作開始と同じです。

フラッシュ書き込み時の UART 端子のプルアップはどうなるか

Q1. フラッシュ書き込み時の UART 端子で、通常ポート使用時には内蔵プルアップを設定しています。フラッシュの書き込み時に影響はありますか。

A1. 内蔵のプルアップ抵抗は、ソフトで設定したときだけ有効(入力モード時だけ有効)になりますので、フラッシュの書き込みでは有効にはなりません。
なお、3 線式の方が高速で通信できますので、書き込み時間を短縮できます。

V_{PP} の立ち上がり時間が遅い場合の問題

Q1. V_{PP} の立ち上がり時間が遅い(ゆっくり 5V から 10V に変化)場合、誤作動の可能性はありますか。

A1. V_{PP} の立ち上がり時間そのものについての規定はありませんが、あまりにも時間がかかる場合には、問題が発生します。V_{PP} のパルスによるモードの設定は、約 2ms 以内 に、すべて完了する必要があります。
立ち上がりに時間がかかり過ぎると、モード設定が正常にできなくなります(書き込みが正常に行えない)。

通常動作時の V_{PP} 端子接続

Q1. 通常動作時、V_{PP} 端子は Vss に直接接続でなく、抵抗で接続することは可能ですか。

A1. フラッシュ品の場合、ライタのドライブ能力と、デバイスのノイズ耐性の両方を考慮すると、10kΩ 程度の抵抗で Vss にプルダウンするのは可能です。ただし、実装時のノイズ耐性が問題ないかどうか評価が必要です。
マスク品の場合、V_{PP} に相当する端子はテスト端子となっているため、V_{PP} は必ず Vss に直接接続する必要があります。直接接続しない場合には、動作保証はできません。

フラッシュ書き込みに適切な CPU クロック

Q1. μPD78F9418 A、78F0988 の書き込み時の、適切なクロック・スピードを教えてください。

A1. CPU のクロックは、できるだけ高速にされることをお勧めします。
基本的には、動作可能クロック範囲であればかまいませんが、高速にした方が書き込みが速く完了します。
また、UART 接続の場合には、μPD78F9418A のクロックは、5MHz か 4.91MHz にしてください。

フラッシュにオンボードで書き込むためのツール

Q1. μPD78F0034A のシリアル・インタフェースで、オンボード・プログラミングができるプログラマはありますか。

A1. PG-FP4(ルネサス エレクトロニクス製)、FL-PR4(内藤電誠町田製作所製)といったライタが、オンボード書き込みにも使用できます。ライタとの具体的な接続については、デバイスのドキュメントに記載されているように、ライタとの接続に使用するシリアル信号線、電源(ライタでは電源供給の確認に使用)、グランド、リセット信号、V_{PP} を使用します。CPU のクロックについては、オンボードのクロックをご使用ください。
なお、ライタ側からの信号は、山一電機製の FAS-1001-2101 コネクタが付いたケーブルで出力されていますので、これに対応したコネクタを基板に準備してください。接続する信号の詳細については、PG-FP4 などライタのマニュアルに記載されていますので、そちらをご参照ください。
ライタと接続するリセット信号や V_{PP} 信号は、通常の動作状態でも使用しますので、リセットについては、通常のリセットに追加して、ライタからもリセットがかけられるようにしてください。
ただし、リセット解除から書き込みモードへの切り替わりには、時間的な制限があります。ライタ側がリセットを解除してから、デバイスのリセットが解除されるまでの時間が、長くならないようにしてください。
また、通常 V_{PP} は V_{SS} に接続するので、書き込み時はライタに、動作時は V_{SS} に接続できるように、スイッチかジャンパで切り替えられるようにしてください。書き込みで使用するその他のシリアル信号は、未使用なら未使用時の端子処理を行い、通常動作で使用するなら、書き込み時に切り替えられるようにしてください。

フラッシュの書き換え回数

Q1. μPD78F0034A のフラッシュ ROM は、何回書き換え可能ですか。

A1. デバイスのバージョンによって異なります。P 規格以降は、20 回の書き換えを保証しています。
なおこの保証は、最初の書き込みから 10 年間のデータ保持を想定しています。したがって、20 回以上書き換えた場合は、10 年のデータ保持は保証できません。
データの保持を気にされない場合は、これ以上書き換えることが可能です。

フラッシュ・メモリ・プログラミング特性のV_{DD}電源電流の条件 [78K0/Kx2、78K0/Lx2]

Q1. 電気的特性で、フラッシュ・メモリ・プログラミング特性の V_{DD} 電源電流の.値は、どのような条件ですか。

A1. この値は、プログラム・ライタを使用して、書き込む際の電源電流値で、次のようになります。

f_{XP}＝10MHz(TYP.)：4.5mA(TYP.)、f_{XP}＝20MHz(MAX.)：11.0mA(MAX.)

一方、セルフプログラミングの場合、動作周波数によらず、11.0mA(MAX.)となります(TYP.値は提示していません)。これは、外部クロック使用時でも、フラッシュ・プログラミングの書き込みでは、内蔵 8MHz から特定のクロックを生成しているからです。

なお、このフラッシュ・メモリ・プログラミング特性の V_{DD} 電源電流は、フラッシュ・マクロだけに流れる値です。したがって、フラッシュ・メモリ・プログラミング中に CPU、A/D、WDT、LVI(,LCD)などを動作させる場合には、その電流値も加算する必要があります。

1.2.17 ROM 書き込み

変換アダプタの互換性 [PA-78P078GF、PA-78P208GF]

Q1. PA-78P078GF と同形状の PA-78P208GF を使用して、μPD78P078GF にプログラムを書き込めますか。

A1. いいえ、できません。パッケージは同じですが、端子接続が異なります。

1.2.18 電　源

A/D 回路の電源分離

Q1. μPD780054 で、アナログとデジタルの電源・グランド同士を接続すると、ADC の特性を出せませんか。

A1. アナログとデジタルの電源関係を共通にすると、ご質問のように、デジタル側の影響を受けます。しかし、別電源を用意すると、コストがかかります。どの程度の精度を要求するかで、電源回路は変わります。お客様の仕様に合わせてご判断ください。

Q2. μPD780024A の電源を V_{DD}(5V±10%)と AV_{DD}、AV_{REF}(5V±2%)の 2 系統にしてもいいですか。
その場合、V_{DD}＋0.3 を越えないように、ショットキ・バリア・ダイオードで対策しようと考えています。

A2. はい、その使い方で問題ありません。V_{DD} が低くなると、V_{DD} 変動の影響がショットキ・バリア・ダイオードを介して、AV_{DD} や AV_{REF} に伝搬される可能性があるので、ダイオードに電流が流れないようにしてください。

電源のバックアップに AV_{DD} は必要ないか

Q1. μPD78F0034A で、電源をバックアップする場合、V_{DD1} の供給だけでいいですか。

A1. いいえ、V_{DD1} だけではだめです。AV_{DD} と V_{DD0}、V_{DD1} は、必ず同じ電圧にする必要があります。
これらはノイズ対策のため別端子になっているだけで、内部的には寄生回路でつながっているので、そこに電流が流れて、バックアップ効果が低下します。

リセット時の電源電流は STOP 時の電流と同じか

Q1. μPD78F0058 で、リセット端子を ロウ固定での電源電流は、STOP モードの低消費電力動作と同等ですか。

A1. いいえ、同じような動作電流にはなりません。リセット時の状態によって異なるので、そのような保証できません。リセットがかかると、出力で使用しているポートも入力になります。このため、入力でオープンと同じ状態になることもあります。このときは貫通電流が流れ、STOP 時より電流が増加することが考えられます。

電源変動の許容値

Q1. μPD789012 の電源電圧の規格は 1.8V～5.5V ですが、変動がある場合の、変動率の規格を教えてください。

A1. 規格はなく、以下の条件がめやすです。なお、動作周波数で許された電源電圧条件を厳守してください。

- 電源電圧の変動率に関しては、0.1V/ms 以内の変動に抑えてください。
- 電源電圧のリップル電圧に関しては、動作電源電圧範囲であれば許されます。

電源オフ時にも外部からポートに電圧をかけたい

Q1. μPD789025 で、CPU の電源がオフの場合に、あるポートに外部から電圧がかかってもいいですか。

A1. そのようなことは許されていませんので、絶対に避けてください。μPD789025 の各ポートには、V_{DD}＋0.3V 以上の電圧はかけられません。これを守らないと、**絶対最大定格違反**となり、デバイスにダメージを与える可能性があります。特に、この状態で電源投入すると、**ラッチアップ破壊**を起こすことも考えられます。
また、各ポートから V_{DD} に保護用のダイオードが入っていますので、この状態ではポート経由で V_{DD}（この場合はグランド電位)に向かって、電流が流れてしまいます。

V_{DD0} と V_{DD1} に別電源を供給

Q1. μPD789417 で、V_{DD0}（ポート用電源）と V_{DD1}（ポート以外用電源）へ、別の電源電圧を供給できますか。

A1. V_{DD0} と V_{DD1} に、異なる電源電圧を供給する使い方はできません。必ず同じ電圧を供給してください。
電源端子を分けているのは、デジタル・ノイズがアナログ回路に影響しないように対策するためです。

電源電流を抑えられるモード(省電力)

Q1. μPD789046 で、STOP モードとサブクロック動作のどちらが、電流を低減できますか。

A1. 基本的に、どの程度のプログラムを実行する必要があるかと、クロックの発振安定時間で決まります。
STOP モードでは、次の 3 つの状態と考えて、各電流値とその時間の積の和を求めます。

STOP モードで待機中　：STOP モードの電流（I_{DD5}）
クロックの発振待ち時間　：HALT モードと同じ電流（I_{DD2}）
通常動作状態　：動作時の電流（I_{DD1}）

サブクロック動作では、次の各電流値と時間の積の和を求めます。

サブクロック HALT 状態　：32.768kHz での HALT での電流（I_{DD4}）

サブクロック動作状態　：32.768kHz での動作電流（I_{DD3}）

両者を比較して、決定してください。処理するプログラムの実行時間が短いとサブクロック動作の方が有利になり、クロックの発振安定時間が短ければ STOP モードの方が有利になります。

1.2.19 リセット

発振停止状態でリセット信号入力時のポートの状態

Q1. μPD78P014 で、発振が停止した状態でリセット信号が入ったときの、ポートの状態はどうなりますか。

A1. 正常なリセット信号が入力されると、ポートは入力モードになります。

強制的にリセット処理を実行

Q1. プログラム中で、リセット処理を(スタートアップ・ルーチンへのジャンプ)実行する方法を教えてください。

A1. ウォッチドッグ・タイマの時間切れを使用するのが一番簡単です。これ以外で、ソフトウエアだけで処理する場合は、アセンブラでの記述となります。この場合、単にリセット・ベクタに分岐するだけでは、リセットと同じ状態にはなりません。

なお、NMI 機能があるマイコンのソフトウエア・リセットの場合は、NMI 処理中にリセット・ベクタに分岐しても、ハードウエア・リセットと同じ状態にはなりませんので、次の処理を最後に追加する必要があります。

```
MOV    PSW,#2   ;PSW の初期値
MOVW   AX,!0    ;リセット・ベクタ読み出し
PUSH   PSW      ;PSW をスタックにセット
PUSH   AX       ;リセット・ベクタをスタックにセット
RETI            ;リセット・ベクタへ分岐
```

なお、リセット時の初期値が定義されているレジスタについては、使用したものの初期値を事前にプログラムで設定しておく必要があります。

電源がなだらかに ON すると暴走する

Q1. 電源が 50ms 程度で立ち上がる場合、CPU が暴走してしまうことがあります。

A1. 電源電圧の安定時間に比べて、リセットの時間が短過ぎるためと思われます。リセットの時定数が電源の立ち上がり時間より短いと、電源が立ち上がらないうちに、リセットが解除されてしまいます。動作開始しようとしても、電源電圧が動作周波数に対して低すぎるために、異常動作になっていると考えられます。

リセットは、少なくとも電源電圧が安定するまでの時間は、保持できるように時定数を選んでください。

リセット入力は長くてもいいか

Q1. リセット信号を入力している期間は、何秒間でもいいですか。

A1. リセットが長い分には、問題ありません。短い方については規定していますが、長い方は規定していません。

Q2. CPU を複数使用し、最初に動作を停止しておく CPU のリセットを継続入力していいですか。

A2. リセット中、ほとんどの端子がハイ・インピーダンス状態です。この状態で、端子が中間電位になると、入力ポートで貫通電流が流れてしまいます。個々の端子の貫通電流は少なくても、全体では大電流となる可能性があります。各端子が中間電位にならないように、外部抵抗でのプルアップなどをしてください。

リセット時、ポートに接続した LED が点灯してしまう

Q1. リセット時、ポートに接続した LED が点灯してしまいます。原因と対策を教えてください。

A1. リセット時には、全ポートがハイ・インピーダンス状態の入力モードになります。外部バッファに TTL を使うと、この状態はハイ・レベルと判断される可能性が高くなります。ポート出力がハイ・レベルで LED が点灯する回路では、点灯してしまいます。対策としては、ロウ・レベル出力で LED を点灯するようにしてください。

リセットしたときの RAM データは保持されるか

Q1. CPU をリセットしたとき、内部 RAM のデータは保持されますか。

A1. 通常動作中のリセットでは、RAM のデータの保持は保証していません。リセットは、通常の動作とは非同期でかかるので、RAM の書き込み中に、リセットがかかることも考えられます。この場合、RAM への書き込みが、正常に行われない可能性があります。これは、リセット・タイミングの問題です。RAM への書き込みをしていないときにリセットがかかれば、問題は発生しません。しかし、たまたま RAM への書き込み中にリセットがかかると、書き込み信号の幅が規定未満だったり、書き込み信号がオンの最中にアドレスが変動することもあり、書き込みアドレスのデータ破壊はもちろん、他アドレスのデータにも書き込みが起こることがあります。

STOP モードの解除にリセットは使用できるか

Q1. μPD78F9116A で、リセットかマスクされていない割り込みで STOP モードが解除できますが、STOP モードでは発振が止まってしまうため、リセットがかからないのではないですか。

A1. リセットはクロックがなくても受け付けられるので、リセットも STOP モードの解除に使用できます。
リセット信号の遅延時間は、アナログ・ディレイによるものなので、問題なく使用できます。

1.2.20 端子処理

入力・出力混在ポートで内蔵プルアップを ON にしていいか

Q1. μPD780308 で、1つのポートに入力端子と出力端子を混在させ、内蔵プルアップ抵抗を ON とした場合、出力端子側と干渉しますか。

A1. いいえ、何も影響はありません。ポートのブロック図に示すとおり、ポートのプルアップ抵抗は、PUO レジスタに 1 を設定し、かつ PM レジスタに 1(入力ポート・モード)を設定したときにしか有効になりません。
したがって、出力に指定した端子には、プルアップ抵抗は接続されません。

未使用の入力端子とグランドの処理

Q1. 入出力ポートについて、未使用時には「個別に抵抗を介して」と処理の記述がありますが、AV_{REF}、AV_{SS} などポート以外の端子は、抵抗を介して接続か、直接接続か、どのようにすればいいですか。

A1. どちらでも問題ありません。通常は、処理が簡単になることか、ら直接接続することになると思います。
なお、NC 端子は内部ではオープンですので、どのように接続してもかまいません。

未使用端子はまとめて 1 つの抵抗で V_{DD0} か V_{SS0} に接続していいか

Q1. 未使用端子の処理で「個別に抵抗を介して、～に接続してください」と書かれていますが、空きポートをまとめて V_{DD0} か V_{SS0} に接続してはいけないのですか。

A1. はい、だめです。電源投入からリセットが有効になるまでの期間や、プログラムの暴走などでポートのモードが書き変わってしまった場合に、ポートが出力になることが考えられます。このとき、未使用端子をまとめていると、ハイとロウの出力端子同士がショート状態となる可能性があります。
マイコンがダメージを受けることが考えられますので、個別に処理する必要があります。

オープン・ドレーン端子の未使用時は、なぜ V_{SS} 接続ではなく V_{DD} 接続か

Q1. μPD78F9116 で、未使用時のポート処理の推奨について、P50-P53 ポート（オープン・ドレーン）だけ、入力時に「抵抗を介して V_{DD} 接続」となっていますが、なぜ V_{SS} 接続ではないのですか。

A1. P5 の中耐圧ポートには、リードしたときのレベルを安定させるための回路が内蔵されており、リード時だけ、内部の回路を介して端子を V_{DD} 方向にドライブするようになっています。プルダウンすると、この内部回路と外部のプルダウンの引き合いにより、電流が増加したり、中間レベルになってしまう可能性が考えられます。
このようなことを避けるために、プルアップを推奨していると思われます。
V_{SS} に接続して P5 のリードを行ったとき、最悪で 3μA の電流が流れてしまうことが考えられます。
これ以外には、特に問題はありません。

入力保護用ダイオード

Q1. 入力過電圧保護用のダイオード(ポート- GND、ポート-V_{DD} 間)は存在しますか。

A1. はい、ダイオードは存在します。しかし、過電圧を許しているわけではありません。デバイスのご使用時は、規格(絶対最大定格)を厳守してください。規格を満足されなかった場合、一切の保証はありません。

テスト端子を 10kΩ でプルダウンしていいか

Q1. μPD780988 のテスト端子を抵抗(10kΩ)でプルダウンしていいですか。

A1. システムのノイズが大きくて、テスト端子の電圧がロウ・レベルを維持できなくなると、テスト・モードに入ってしまい、正常に動作しなくなる可能性があります。
テスト端子は、指定された処理(直接 V_{SS} に接続)を行ってください。

1.2.21 発　振

発振回路の抵抗値

Q1. μPD78F0058 の発振回路定数は、ユーザ評価か発振子メーカ評価の依頼になっているのは、なぜですか。

A1. これらの定数は、水晶によって変化しますので、一概に決められません。
なお、サブシステム・クロックの抵抗 R2 は、水晶振動子の高調波発振を防止するためのものです。したがって、基本波で発振する水晶振動子をご利用の場合には、ほとんど必要性がありません。R2 を大きくすると、目的の周波数で発振しやすくなりますが、大き過ぎると発振そのものが停止するおそれがあります。以前は 220～330kΩ 程度を推奨していたこともありますので、参考にしてください。

XT1 は GND に接続していいか

Q1. μPD780058 のユーザーズ・マニュアルに「XT1 は V_{DD} に接続」と記載されていますが、GND に接続すると問題ありますか。

A1. XT1 を GND に接続すると、内部的なリーク電流が、V_{DD} に接続した場合より大きくなります。
それ以外の、機能的な面については、特に問題ありません。

フィードバック抵抗を使用しないとどうなるか

Q1. μPD789306 のサブ発振モード・レジスタ(SCKM)で、フィードバック抵抗を使用しない(FRC＝1)選択とし、外部にも抵抗を付けなかった場合、どのような動作になりますか。

A1. 正常にサブシステム・クロックを発振できません。仮に発振したとしても、安定な発振は期待できません。
したがって、サブシステム・クロックを用いた動作はできなくなります。

1.2.22 ノイズ

XT1 をオープンにするとノイズの影響を受けやすいか

Q1. μPD78F0058 の XT1 をオープンのまま使用していますが、ノイズの影響を受けやすいですか。

A1. はい、ノイズの影響を受けやすくなります。

LSI はどのようなノイズ対策がされているか

Q1. μPD7800xx シリーズは、EMI ノイズ低減をされているそうですが、どのようなことをされていますか。

A1. 次のような対策をしています。
(1) 回路のドライブ電流を低減
(2) 大電流が流れるポートについて、GND や電源端子を分離

電源電流の高調波がクロック周波数と違う

Q1. スペクトラム・アナライザでグランド・パターンの電流を観測すると、2.5MHz 間隔の高調波を観測できます。CPU クロックは分周せず 5MHz で使っているのに、電流の高調波が 2.5MHz 間隔なのはなぜですか。

A1. CPU は、複数のクロックを1つの単位として処理します。たとえば、NOP 命令を連続で実行すると、2 クロック単位で命令フェッチを行って、命令フェッチの瞬間以外は、ほとんど仕事をしません。この場合、2 クロック、つまり 5MHz の半分の 2.5MHz 間隔で、大電流が流れてノイズとなります。

ワンタイム PROM 品とマスク ROM 品とでノイズ耐量は異なるか

Q1. ワンタイム PROM 品とマスク ROM 品で、内部構成のちがいなど、内蔵 WDT のノイズ耐量に差が出るようなことは考えられますか。

A1. はい、製造プロセスが異なりますので、ノイズ耐量にも差が出ると思われます。

μPD78F0034A と μPD78P018F の割り込み要求端子の耐ノイズ性の違い

Q1. μPD78F0034A と μPD78P018F の割り込み要求端子のノイズ耐性にちがいがあるようですが、なぜですか。

A1. **μPD78P018F** の INTP0 は、クロックでのサンプリングをしており、割り込み要求を確実に受け付けるには、2サンプリング周期以上のパルス幅が必要です。これにより、1サンプリング周期未満のノイズに対しては、完全に除去できます。ただし、その分だけ受け付けタイミングが遅れることになり、また STOP モードの場合は使用できないなどの制限があります。

一方、**μPD78F0034A** では、クロックでのサンプリングをしていないため、割り込み要求信号が入ると、すぐに受け付けます。そのため、シュミット・トリガ入力のスレッシュホールド・レベルを越えるノイズが入力されると、割り込み要求として受け付けてしまう可能性があります。ノイズが多い環境の場合には、割り込み処理の中で P00 を読み込んで、ノイズかどうかの判定を行うか、外部で何らかの対策が必要となります。

1.2.23 製品の違い・代替品

μPD780034A から μPD780078 へ移行時の注意点

Q1. μPD780034A にプログラムが入らないので、μPD780078 に変更したいのですが、プログラムの変更点を教えてください。

A1. 16 ビット・タイマが追加されたので、16 ビット・タイマ関係の SFR 名称が変更になっています。

TMC0→TMC00、CR01→CR010 のように、名前に“0”を追加してください。

3 線式シリアルについて、SIO3 が1つだけになりましたので、SIO30 関係の SFR は、SIO30→SIO3 のように“0”を取ってください。

SIO31 については、CSI1 と全く機能が変わっていますので、プログラムを完全に見直してください。

デバイスの違いについては、比較表(http://www2.renesas.com/faq/ja/mi78k/mi-78dif.pdf)をご覧ください。

μPD78P018F の代替品

Q1. μPD78P018F の代替品を教えてください。また、置き換え時の注意事項はありますか。

A1. 機能的には、μPD78F0078 が代替品です。ただし、ピン配置や内蔵されている機能が一部異なります（一部追加・削除）。また、SFR 名称が異なるものもありますので、それらの見直しも必要です。
A/D コンバータのように、使い方で注意が必要になる機能もありますので、これらの機能をご使用の場合には、プログラムの変更が必要です。
デバイスの違いについては、比較表(http://www2.renesas.com/faq/ja/mi78k/mi-78dif.pdf)をご覧ください。

μPD780021A と μPD78P014 の違い

Q1. μPD780021AGC-AB8 と μPD78P014GC-AB8 の相違点(ピン配置、ソフトなど)はなにですか。

A1. μPD780024A/34A のユーザーズ・マニュアルの付録 A をご参照ください。機能的には、シリアルを強化した程度ですが、ピン配置は全く異なります。また、A/D コンバータの動作が、一部変更になっています。
サブファミリの製品としては、PROM 版はなく、フラッシュ品となります。
デバイスの違いについては、比較表(http://www2.renesas.com/faq/ja/mi78k/mi-78dif.pdf)をご覧ください。

μPD78058 と μPD780058 の違い

Q1. μPD78058 と μPD780058 のピン配置、機能、ソフト面等の相違点はなにですか。

A1. μPD780058 サブシリーズのユーザーズ・マニュアルの付録 A をご参照ください。
ピン配置に関しては、次の点が異なります(14、15 ピンは、機能が追加されています)。

ピン番号	μPD78058	μPD780058
67	P06/INTP6	Vss0
74	AVDD	VDD0
14	P23/STB	P23/STB/TxD1
15	P24/BUSY	P24/BUSY/RxD1

Q2. μPD78058 から μPD780058 への移行時に、ソフトウエア上の注意点はありますか。

A2. μPD780058 サブシリーズには、A/D 変換開始後の 1 回目のデータの精度が悪いという制限事項がありますので、その部分についてはソフトの変更が必要となります。
また、ポートと外部割り込みが 1 つ少なくなっていりますので、その部分に関しては見直しが必要です。

μPD78P054 と μPD78F0058 の違い

Q1. μPD78P054GC から μPD78F0058 へ移行するにあたって、注意点はありますか。

A1. 大きな違いは、P06/INTP6 が Vss0 になったことですが、V_{DD} や V_{SS} 関連で、一部の機能が変わっています。
たとえば AV_{REF0} 端子は、従来の A/D コンバータの基準電圧に加えて、アナログ電源の機能も持っています。
従来の AV_{DD} 端子は、V_{DD0} としてポート部の電源になっていますので、配線パターンにはご注意ください。
また、電源電圧範囲が 1.8V～5.5V と低くなっています。

Q2. メモリ・サイズが異なっていますが、これはメモリ・サイズ・レジスタを操作することで解決できますか。

A2. はい、IMS レジスタと IXS レジスタの設定で、同一にできます。
ただし μPD78F0058 では、A/D コンバータを起動すると、1 回目の変換結果の精度が確保できず、読み飛ばす必要があります。このため、A/D コンバータを使用している場合は、プログラムの変更が必要です。

μPD78P064GC の代替品

Q1. μPD78P064GC の代替品種を教えてください。

A1. 代替品種としては、μPD78P0308GC があります。

μPD789114 サブシリーズと μPD789114A サブシリーズの違い

Q1. μPD789114 から μPD789114A へ切り替えると、なにが変わるのですか。

A1. おもな変更点は、以下のとおりです。

(1) 制限事項の修正

・ポート 5 入力時の制限事項

・10 ビット A/D コンバータの制限事項

(2) 動作速度の改善

最高動作クロック周波数が、次のように改善されています。

μPD789114　:5MHz

μPD789114A　:5MHz(K 規格)、10MHz(K 規格以外)

Q2. μPD78F9116, μPD78F9116A, μPD78F9116B の違いは何ですか。

A2. おもな変更点は、以下のとおりです。

(1) 制限事項への対応（修正）

・ポート 5 入力時の制限事項（μPD78F9116B）

・10 ビット A/D コンバータの制限事項（μPD78F9116A/B）

(2) 動作速度の改善

最高動作クロック周波数が、次のように改善されています。

μPD78F9116/A　　:5MHz

μPD78F9116B/B(A)　:10MHz (ただし、μPD78F9116B(A1)/(A2)は 5MHz です)

(3) フラッシュ信頼性の向上

μPD78F9116　:1 回の書き換えだけを保証

μPD78F9116A　:20 回の書き換えを保証

μPD78F9116B　:20 回の書き換えを保証,、高信頼性の(A)品、(A1)品、(A2)品を準備

μPD78F9177 と μPD78F9177A の違い

Q1. μPD78F9177 から μPD78F9177A に切り替えると、なにが変わるのですか。

A1. おもな変更点は、以下のとおりです。

(1) 制限事項の修正

・ポート 5 入力時の制限事項

(2) 動作速度の改善

最高動作クロック周波数が、次のように改善されています。

μPD78F9177　　:5MHz

μPD78F9177A/A(A)　:10MHz (ただし、μPD78F9177A(A1)は 5MHz です)

(3) フラッシュ信頼性の向上

高信頼性の(A)品、(A1)品を準備

フラッシュ品の品名 [78K0/Kx1]

Q1. 78K0/Kx1 のフラッシュ品を使いたいのですが、μPD78F0138M1GC～μPD78F0138M6GC と 6 種類あって、どれを選んでいいのか分かりません。

A1. マスクオプションについて、フラッシュ品の品名に次のフィールドがあり、組み合わせで **M1～M6** が追加されています。

POC(パワーオン・クリア)　:使用の有無と使用時の電圧 (2.7V か 3.3V)

内蔵発振器　　:ソフトウエアで発振を停止できるかどうか

記号	POC	ソフトウエアでの内蔵発振器発振停止
M1	使用不可	不可
M2	使用不可	可能
M3	2.7V	不可
M4	2.7V	可能
M5	3.3V	不可
M6	3.3V	可能

使い分けとしては、以下のようになります。

(1) 最低動作電源電圧を 4V 程度以上か 2.7V～3V で使用する場合や、外部リセット IC と組み合わせて使用する場合は、**M1** か **M2** となります。

・内蔵発振器を止めなければ **M1** です(たとえば μPD78F0138M1GC)。

・内蔵発振器を止めたければ **M2** です(たとえば μPD78F0138M2GC)。

(2) 最低動作電源電圧を 3V 程度で使用する場合は、**M3** か **M4** となります。

・内蔵発振器を止めなければ **M3** です(たとえば μPD78F0138M3GC)。

・内蔵発振器を止めたければ **M4** です(たとえば μPD78F0138M4GC)。

(3) 最低動作電源電圧を 3.7V 程度で使用する場合は、**M5** か **M6** となります。

・内蔵発振器を止めなければ **M5** です(たとえば μPD78F0138M5GC)。

・内蔵発振器を止めたければ **M6** です(たとえば μPD78F0138M6GC)。

μPD78064、78064B サブシリーズと μPD780308 サブシリーズの違い

Q1. μPD78064 サブシリーズ、μPD78064B サブシリーズから μPD780308 サブシリーズへ移行する場合、注意点はありますか。

A1. 機能的に大きく変わったのは、1チャネルの 3 線式シリアルと内部拡張 RAM が追加されています。デバイスの違いについては、比較表(http://www2.renesas.com/faq/ja/mi78k/mi-78dif.pdf)をご覧ください。
なお、電気的特性も多少違います。

8 ビット・マイコンの製品の違い [78K0/78K0S]

Q1. 8 ビットのマイコンを選ぶにあたって、簡単な比較表はありませんか。

A1. 「78K マイクロコントローラ セレクション・ガイド」をご覧ください。

Q2. 8 ビットのマイコンをどのように選べばいいですか。

A2. どのような応用に用いるかで、その実現に要求される機能があります。その**機能が内蔵されているか**どうかが、判断基準のひとつになります。分かりやすい例としては、サーミスタで温度を検出して何かしたい場合、サーミスタの両端の電位差を測るための A/D コンバータが内蔵されている必要があることが判断基準です。
LCD の表示が必要な場合は、LCD コントローラの内蔵が判断基準のひとつとなります。また、表示データ量が多い場合は、外部の表示コントローラと組み合わせて使用することになり、同期式シリアル通信機能(CSI)などの通信機能があることが判断基準となります。CAN を内蔵しているか、内蔵しているタイマの機能や数も判断基準のひとつとなります。
必要な機能が内蔵されていることの確認が済んだら、必要なポートの数や ROM/RAM の容量、動作電源電圧、処理速度(動作クロック周波数)などでの検討を行ってください。
最新の 78K0 や 78K0S の ALL Flash マイコンでは、使用されるパッケージのピン数によって、次のように品名が付けられています(抜粋)。

- 16 ピン　:Y (たとえば、78K0S/KY1+)
- 20 ピン　:A (たとえば、78K0S/KA1+)
- 30 ピン　:B (たとえば、78K0/KB2、78K0S/KB1+)
- 44 ピン　:C (たとえば、78K0/KC2、78K0/FC2)
- 52 ピン　:D (たとえば、78K0/KD2)
- 64 ピン　:E (たとえば、78K0/KE2、78K0/LE2、78K0/FE2)
- 80 ピン　:F (たとえば、78K0/KF2、78K0/LF2、78K0/FF2)
- 100 ピン :G (たとえば、78K0/LG2)

ここで、78K0 や 78K0S は使用されている CPU コアの名前を示します。また、/の後の K、L、Fは対応分野を示します。K は汎用、L は LCD 制御用、Fは電装用(CAN 対応)となります。

Q3. 8 ビットの ALL Flash マイコンで、78K0 と 78K0S の 2 つのコアの違いはなんですか。

A3. 78K0S は 78K0 のサブセットになっており、コアとして、一部の機能が削除されています。削除された主な機能は次の通りです（内蔵される周辺機能は、個々のマイコンで異なるので、ここでは触れません）。

- レジスタ・バンク機能
- 乗除算機能
- 10 進補正機能
- 割り込みの優先度処理機能
- ソフトウエア割り込み機能
- ニブル（4 ビット）単位でのローテート機能
- 内蔵フラッシュのセルフ・プログラミングでのブート・スワップ機能

また命令セットでは、削除された機能に関連した命令を含めて、使用頻度が低いと思われる次のような命令やアドレシングが削除されています。

- 8 ビット転送命令や演算命令での「ベースト・インデクスト・アドレシング」（MOV A,[HL+B]や ADD A,[HL+C]など）
- 8 ビットのイクスチェンジ命令での「ダイレクト・アドレシング」（XCH A,!addr16）
- 16 ビット転送命令での SFR 領域に対するアクセス（そのような SFR がマッピングされていないので影響はありません）
- 「ショート・ダイレクト・アドレシング」（saddrp）領域へのイミーディエト・データ設定（MOVW saddrp,#word）や、「ダイレクト・アドレシング」での AX レジスタとの転送（MOVW AX,!addr16 など）
- 8 ビットの演算では、演算結果の格納先は A レジスタに限定（ADD C、A などは削除）
- 乗除算命令
- ニブル（4ビット）単位でのローテート命令（ROR4 [HL]など）
- 10 進補正命令（補正については、FAQ の「78K0S で 10 進補正を行うプログラム例」をご参照ください）
- ビット操作関係の転送や演算命令（MOV1 CY,A.bit や AND1 CY,saddr.bit など）
- CALLF 命令
- ソフトウエア割り込みに関した命令
- スタック・ポインタに直接値を設定する命令（AX レジスタから転送することで対応）
- ビット判定命令での「レジスタ・インダイレクト・アドレシング」（BT [HL].bit,$addr16 など）
- セマフォ制御命令（BTCLR 命令）
- レジスタ・バンク選択命令

他の違いとして、割り込み処理中（多重割り込み禁止）に他の割り込み要求が発生したとき、78K0 では割り込みから復帰後に、メイン処理の命令を 1 命令実行して次の割り込み処理に入りますが、78K0S では割り込みから復帰後、メイン処理の命令を実行せずに次の割り込み処理に入ります。

78K0/Kx2 と 78K0S/Kx1+の違い [78K0/Kx2、78K0S/Kx1+]

Q1. 8 ビットの ALL Flash マイコンで、78K0 と 78K0S はピン数以外どのように違い、どう使い分ければいいですか。

A1. 78K0/Kx2 と 78K0S/Kx1+は同じ 8 ビットの ALL Flash マイコンですが、使用されているマイコンのコアが異なります。コアの違いについては「8 ビット・マイコンの製品の違いについて」の A3 をご参照ください。

ピン数（それに伴うポート数）やコアの違い以外にも、機能面で次の点が異なります。

	７８K０／Kｘ２	７８K０S／Kｘ１＋
動作電源電圧	１.８～５.５Ｖ	２.２注1～５.５Ｖ
最大動作周波数	２０MHｚ	１０MHｚ
ＲＯＭ容量	８～１２８Ｋバイト（６４K以上はバンク使用）	１～８Ｋバイト
ＲＡＭ容量	５１２～７Ｋバイト（高速ＲＡＭは１Ｋまで）	１２８～２５６バイト（高速ＲＡＭだけ）
１６ビット・タイマ	１、２チャネル	０、１チャネル
８ビット・タイマ注2	４チャネル	１、２チャネル
キャリア・ジェネレーション	１チャネル	なし
時計タイマ	一部製品にあり	なし
ブザー出力、クロック出力	一部製品にあり	なし
ウォッチドッグ・タイマ	１チャネル	１チャネル
ウィンドウ機能	あり（４段階）	なし
不正メモリアクセス検出	あり	なし
カウント・クロック	低速内蔵クロック固定	低速内蔵クロック／システム・クロック
Ａ／Ｄコンバータ（１０ビット）	８チャネル	なし、４チャネル
シリアル機能		
ＵＡＲＴ	２チャネル	０、１チャネル
ＣＳＩ	１、２チャネル	なし
自動送受信機能付きＣＳＩ	一部製品にあり	なし
I^2Cバス	１チャネル	なし
キー・リターン割り込み	一部製品にあり	なし
乗除算回路	一部製品にあり １６×１６、３２÷１６	一部製品にあり ８×８
ＰＯＣ	１.５９Ｖ／２.７Ｖ	２.１Ｖ固定
内蔵レギュレータ	あり	なし
ＬＶＩ	電源電圧／外部端子	電源電圧のみ
リセット端子機能	リセット端子	リセット／入力ポート
システム・クロック	高速内蔵クロックで起動、その後プログラムで選択可能	オプション・バイトで指定
周辺クロック	システム・クロック／高速内蔵クロック	システム・クロックから分周して選択
フラッシュメモリ		
メモリサイズ指定機能	あり	なし
ライタでの書き込み	ＵＡＲＴ／ＣＳＩ	Ｘ１、Ｘ２使用
ブロック・サイズ	１Ｋバイト	２５６バイト
ブート・スワップ	あり	なし
モード制御切り替え	専用のＦＬＭＤ０端子	Ｘ１と兼用
セキュリティ機能	セキュリティ・コマンドやライタでの指定により、消去禁止や書き換え禁止	プロテクト・バイトで消去、書き換え禁止
セルフ・プログラミング	ＦＬＭＤ０端子で起動	特殊シーケンスで起動
周辺アクセスの調停	ウエイトで調停	なし

注 1. POC 電圧のばらつき(2.0～2.2V)を含みます。
2. 搭載されているタイマの機能は異なります。

備考 この表はおもな違いだけを掲載しています。タイマで選択できるソース・クロックが異なったり、Kx2 のアナログ入力との兼用端子では初期状態がアナログ入力だったり、個々の設定レベルでは、ほかにも異なっているものがあります。また、特性でも高速内蔵発振クロックの精度が±5%(Kx2)と±5%(温度制限で±3%)(Kx1+)と異なっているものがあります。

必要な周辺機能、処理速度、プログラムの大きさと将来の製品展開などを加味して使い分けてください。

1.2.24 定　格

絶対最大定格と DC 特性の違い

Q1. 規格に、絶対最大定格 I_{OL}、I_{OH} がありますが、DC 特性にも同じ規定があります。その違いは何ですか。

A1. 絶対最大定格と DC 特性の規格は、全く意味が異なります。
絶対最大定格は、デバイスとして破壊等の恒久的なダメージを受けない条件を記述したものです。
一方、DC 特性は、動作を保証している値です。
通常、両方とも記載しており、片方しかないことはありません。

Q2. ハイ・レベル出力電流(1 端子)は、絶対最大定格－10mA、DC 特性－1mA(max)となっています。電源電圧 5V で－3mA を駆動していいですか。絶対最大定格値と DC 特性の max 値が大きく異なるのはなぜですか。

A2. 問題があります。そのような動作は保証できません。
両者の値の違いは、デバイスの特性が大きくばらつくためです。
デバイスに恒久的なダメージを与える(破壊する)可能性のあるレベルと、実際にご使用いただくレベル(特性を保証)とは、当然ながら差があります。この例では、－10mA まで電流が流せることを保証しているわけではありません。動作を保証(流せると保証)しているのは、あくまでも－1mA までです。

電源電圧と動作周波数範囲

Q1. μPD780023 は、動作電源電圧が 1.8V～5.5V(サイクル・タイム 1.6us 以上)で、8.38MHz の発振子使用時は 4.0～5.5V ですが、電源電圧 1.8V 時に 8.38MHz の発振子を使用して、サイクル・タイム 1.6us 以上での動作は保証されないのですか。

A1. はい、保証されません。CPU のサイクル・タイムと発振回路の特性は、別の特性であり、両方を満足しないとデバイスの動作は保証できません。

Q2. それでは、TDK 製 8.0MHz, 8.38MHz 発振子での電源電圧が、2.0～5.5V となっているのは、なぜですか。

A2. これは、発振子メーカが示している発振特性です。マイコンが、この電圧まで保証するわけではありません。

Q3. 発振子メーカで発振の保証をすれば、マイコンでの使用は問題ないですか。

A3. 発振子メーカが、マイコンの特性ばらつきを含めて保証できるかどうかは疑問ですが、基本的にはそのようにお考えください。なお、発振子の保証条件については、発振子メーカへご確認ください。

V_{DD}=1.8V で高速モードは使用できるか [78K0S]

Q1. μPD789124 の RC 周波数特性の表に、2MHz で V_{DD}=1.8V 以上の場合、発振周波数は 0.5～2.5MHz と記載されています。また、サイクル・タイム(最小命令実行時間)の動作保証範囲が、1.6～8us になっています。2MHz、1.8V で使用する場合、サイクル・タイムが動作保証範囲を越えてしまうのではないですか。

A1. 1.8V では、高速モードは使えないことになります。低速モードでご使用ください。
発振回路の規格は、あくまでも発振回路の特性を示しているだけで、CPU の動作は保証していません。

1.2.25 パッケージ

ドライパック

Q1. 湿度インジケータの 20%が"WARNING(警告)"、40%が"EXAMINE(調査)"ですが、どう考えればいいですか。

A1. これはあくまで湿度の目安を示しており、40%がピンクになると、プリベークが必要になります。20%でも、その時間が続くと吸湿は進みますので、保管期間が長くなった場合は、プリベークを行うことをお勧めします。
なお、このインジケータは、未開封状態での湿度を確認するためのものです。開封後の表示は無効です。

μPD789316 のパッケージ

Q1. μPD789316GK-XXX-9ET のパッケージは、JEITA(EIAJ) type、JEDEC type、その他のうちどれですか。

A1. JEDEC type です。

1.2.26 プログラミング

最初に実行する内容 [共通]

Q1. C 言語で 78K0 のプログラムを開発したいのですが、リセットでの起動からアプリケーションの処理(main 関数)を実行するまでの流れが分かりません。

A1. その部分は、スタートアップ・ルーチンが処理します。マイコンのアプリケーション・プログラムは、内蔵 ROM (マスク ROM やフラッシュなど)に書き込み、応用システムに組み込みます。事前に、内蔵の周辺機能を初期設定しておく必要があります。また、C 言語の標準関数やランタイム・ライブラリを使用するための設定を行う必要があります。起動直後、スタートアップ・ルーチンでは、以下の処理を行っています。

(1) 前処理
標準ライブラリを使用するために、必要な処理を行います。

(2) ハードウエアの初期設定
マイコンに内蔵された周辺機能の初期化を行います。実際の処理は、hdwinit 関数で定義してください。

(3) ROM 化処理
変数の初期値がある場合、その初期値を ROM から実際の変数領域(RAM)にコピーします。

(4) main 関数の起動
main 関数をコールして、実際のプログラムを実行します。

Q2. 内蔵周辺の初期化は、hdwinit 関数に記述するようになっていますが、他の部分に記述できないのですか。

A2. 基本的には、実際に使用するまでに内蔵周辺を初期化すればいいので、あとで記述してもかまいません。ただし、出力に設定したいポートの初期化が遅くなったりすると、ノイズの影響が増えることが考えられます。また、初期状態では、動作クロックが低い周波数に設定されています。電源電圧と動作クロック周波数の関係を考慮し、できるだけ早めに初期化を完了されることをお勧めします。さらに、ウォッチドッグ・タイマの設定は、処理プログラムが起動してから操作することは、あまりお勧めできません。

Q3. hdwinit 関数に記述する初期化の順序について、注意することはありますか。

A3. 一番注意すべきなのは、電源電圧と動作可能クロック・サイクルや動作可能な内蔵周辺の関係です。最近は、POC(パワーオン・クリア)回路が内蔵されている製品が多くなっています。このような製品では、電源が ON になって、POC によるリセットがかかってから起動する電圧は、低い電圧になっています。また、リセット解除後の起動時間が短い最近のマイコンでは、メイン・システム・クロックの発振が安定する前に、内蔵発振器で動作を開始します。このような場合には、電源電圧の立ち上がり時間を意識して、初期化を行う必要があります。

次に注意すべきなのは、信号の流れに沿って、信号の入口側から順に設定していくことです。こうしないと、異常な出力をしたり、誤動作を起こすことも考えられます。

それ以外には、シリアル系のクロックなどのように、初期設定時にパルスが出力されないような順序で設定する必要があるものもあります。

78K0S で 10 進補正を行うプログラム例

Q1. 78K0S に 10 進の補正命令がありませんが、どのようにプログラムすればいいですか。

A1. (1) 78K0 の 10 進補正命令(ADJBA 命令と ADJBS 命令)を ADJBAS(加算補正)か ADJBSS(減算補正)への、サブルーチン・コールに置き換えれば、10 進補正を行うことができます。

(2) プログラムの書き換えが多い場合、次のようにマクロを定義して、サブルーチン・コールに置き換えます。

```
ADJBA     MACRO
          CALL      ADJBAS
          ENDM
ADJBS     MACRO
          CALL      ADJBBS
          ENDM
```

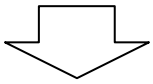
(3) コール命令を多用して、プログラムが大きくなりすぎる場合、CALLT 命令を使えば短くできます。マクロの定義を以下のように変更して、CALLT テーブルを定義してください。

```
;
;         マクロの定義
;
ADJBA     MACRO
          CALLT     [ADJBAT]
          ENDM
```

```
ADJBS    MACRO
         CALLT     [ADJBST]
         ENDM
;
;        CALLT テーブルの定義
;
CALTTBL  CSEG    AT      40H
ADJBAT:  DW      ADJBAS
ADJBST:  DW      ADJBSS
;
;        10 進補正（加算）
;
ADJBAS:  PUSH    BC
         MOV     C, #0
         BNC     $NEXT0          ; IF C=1 ADJUST 60H
         MOV     C, #0B0H
NEXT0:   BT      PSW.4, $NEXT1   ; JUMP IF AC=1
         MOV     B,A             ; SAVE DATA
         AND     A, #0FH         ; MASK UPPER NIBBLE
         CMP     A, #0AH
         MOV     A, B            ; RESTORE DATA
         BC      $NEXT3          ; JUMP IF NOT ADJUST
NEXT1:   ADD     A, #6           ; ADJUST
         MOV     B, A            ; SAVE DATA
NEXT2:   BNC     $NEXT3
         MOV     C, #0B0H
NEXT3:   CMP     A, #0A0H
         BC      $NEXT4
         MOV     C, #0B0H
NEXT4:   ADD     A, C
         ADD     A, C
         POP     BC
         RET
;
;        10 進補正（減算）
;
ADJBSS:  PUSH    BC
         MOV     C, #0
         BNC     $NEXT5
         MOV     C, #0D0H
NEXT5:   BF      PSW.4, $NEXT6
         SUB     A, #6
NEXT6:   BNC     $NEXT7
         MOV     C, #0D0H
NEXT7:   ADD     A, C
         ADD     A, C
         POP     BC
         RET
```

(4) 以上の補正ルーチンを使えば、78K0 のアプリケーション・ノートⅡの 10 進加算(符号付き)が使えます。
ただし、1 ビットの転送命令がないので、次のように変更します。

```
MOV1    CY, SFLAG
NOT1    CY
MOV1    SFLAG, CY
```



```
MOV     A, FLAG
XOR     A, #1
MOV     FLAG, A
```

その上で、このサブルーチンを呼び出す側で、以下のように定義してください。

```
FLAGS   DSEG    SADDR
FLAG:   DS      (1)
SFLAG   EQU     FLAG.0
```

未使用割り込み発生時の安全対策

Q1. バグや CPU の故障等で、使用していない割り込みが発生した場合、CPU はどのような動作になりますか。

A1. CPU は、割り込み要求を受け付けると、割り込みごとに決まっているアドレスから、分岐先アドレス(ベクタ)を読み、そこにプログラムを分岐します。あとは、そのアドレスに書かれているコードを命令として実行しますので、どのような動作になるかは、読み込んだコードにより変わります。つまり、動作の特定はできません。

Q2. 未使用の割り込みが発生したとき、@cstart に飛ばしたいのですが、どのようにすればいいですか。

A2. C ソース中に割り込み関数を定義して、その処理関数名を@cstart にするだけです。たとえば、INTP0 をそのようにする場合、次のように、使用しない割り込みすべてについて、noint で処理するように宣言してください。

```
#pragma interrupt INTP0 noint rb1
```

これで、未使用の割り込みベクタは、すべて noint になります。そこで noint として、以下の関数定義をします。

```
void noint()
{
  #asm
        :
      割り込み関係のフラグの初期化処理
        :
      movw  ax , !0
      br  ax
  #endasm
}
```

実際の処理は、どうしてもアセンブラ記述になるので、このように#asm で、インラインでアセンブラを使用してください。

レジスタ・バンク割り付けの考え方

Q1. 割り込み関数の数が 3 つ以下なら、各割り込みをレジスタ・バンク RB1-3 に割り当てればいいと思いますが、それ以上の数の割り込みに対応するには、どのようにすればいいですか(レジスタ退避など)。

A1. すべての割り込みに、異なるレジスタ・バンクを割り当てる必要はなく、多重割り込みの有無で異なります。
割り込みをいくつ使っても、同時には受け付けなければ、メイン処理と割り込み処理で、レジスタ・バンクを 1 つずつ使用するだけです(割り込みを多重処理しないので、1 つのレジスタ・バンクを共用)。
多重割り込みで、割り込み処理中に、他の割り込み要求を受け付ける場合は、別のレジスタ・バンクを使用する必要があるので、割り込みの組み合わせで、矛盾がないようにレジスタ・バンクを配分してください。
多重のレベルが深くて、レジスタ・バンクで対応できない場合には、処理の優先度が低いものからレジスタ・バンクを使わないで、スタックに待避するようにしてください。
これは割り込みの定義を行う、#pragma interrupt 指令において、レジスタ・バンク指定を行わない(NOBANK も指定しない)ことにより、自動的に行われます。
詳細は CC78K0 言語編のマニュアルの「第 11 章拡張機能」の「11.5 拡張機能の使用方法」の「(10)割り込み関数」に説明されていますので、そちらをご参照ください。なお、簡単な例も記述されています。

LCD 表示用 RAM をビット・セット／リセット

Q1. μPD789488 で、LCD 表示用 RAM 内容をビット・セット／リセットすることは可能ですか。

A1. アセンブラでのビット・セット/リセットはしないでください。ビット操作命令は、リード・モディファイ・ライト方式のためです。LCD 表示 RAM には、上位 4 ビットがないので、読みだした時に不定となるため、この上位 4 ビット部分には、0 を設定する必要があります(ユーザーズ・マニュアルの「13.5 LCD 表示データ・メモリ」末尾)。
C 言語では、ビット・フィールドを 4 ビットに設定して、ビット単位の AND 演算子を利用すれば、記述可能と思われます。お手元の C 言語のマニュアルを参照して、プログラムを作成してください。

ポートのビット・クリア

Q1. μPD789012 の P2.0 は、どのようにクリアするのですか。「AND P2,#11111110B」では、SFR が使えません。

A1. 対象の全ポートをポートとして使用する場合は、単に「CLR1 P2.0」命令を使用してください。
兼用機能を使用する場合、ポートのビット操作に要注意です。μPD789012 は、ポート出力ラッチと兼用機能の AND を端子に出力します。ポートのビット演算では、端子出力の状態をバイト読み出しをして、ビット演算結果をバイト単位でポートへ戻します。このため、P20 の演算では、TxD(P21)がロウ・レベルなら、バイト読み出しで P21 に 0 が読み出されるので、P21 の出力ラッチに 0 が書き込まれ、以降 TxD として機能しません。
このため、ポートに書き出す値を RAM に保存しておいて、ポートの内容を書き換えるときに、この値を修正(この場合はビット 0 をクリア)して、RAM に書き戻すとともに、P2 に出力してください。次の例は、saddr 領域に PORT2 の名前の変数領域を確保しておき、その値を P2 に出力するものです。

```
MOV   A, PORT2
AND   A, #11111110B
MOV   PORT2, A
MOV   P2, A
```

10 進補正命令がない CPU 用の四則演算のサンプル・プログラム

Q1. μPD789415 には 10 進補正命令がないので、10 進演算が複雑です。加算、減算、掛け算、割り算などの、10 進演算のサンプル・プログラムはないですか。

A1. このマイコンは制御用です。10 進演算を考慮した、サンプル・プログラムは準備していません。
10 進補正が必要な場合には、ビット・テスト命令で PSW の AC をチェックすれば、プログラムで対応可能と思われます(ただし、十数ステップは必要)。78K0S で 10 進補正を行うプログラム例をご参照ください。

スタック・ポインタに直接数値を代入する命令 [78K0S]

Q1. μPD789415 のマニュアルには、SP に直接値を設定する命令がないようですが、「MOV SP,#0FEFFH」の命令は使用できますか。

A1. いいえ、SP には直接設定できません。いったん AX レジスタに設定してから、SP に転送してください。

```
MOVW    AX, #0FF00H
MOVW    SP, AX
```

設定系のレジスタを一定周期で書き換えてもいいか

Q1. μPD789022 で、一定周期で下記のレジスタをリフレッシュしていいですか。
PMn、PUO、TCL2、WDTM、TMC20、CR20、IF0、MK0、IF1、MK1

A1. いいえ、問題があります。
タイマ動作中に CR20 の設定を行うと、その時点で割り込みが発生する可能性があります。
また、IF0 や IF1 では、再設定を行うと、発生した割り込みが、その時点で消失する可能性がありますので、不用意に操作しないでください。
基本的に、必要のないレジスタ類の再設定は、トラブルの原因になりかねませんので、お勧めできません。

16 ビット空間でのテーブル参照

Q1. 16 ビット空間でのテーブル参照は、どのようにプログラミングすればいいですか。

A1. テーブルを参照するためのパラメータが 8 ビット(A レジスタ)で、結果も 8 ビットで A レジスタに格納する例を以下に示します。

```
MOV     X, A            ; X=Input data(offset)
SUB     A, X            ; A=0
ADDW    AX, #table      ; get Table Address
MOVW    HL, AX
MOV     A, [HL]         ; Get data from Table
```

スタック・ポインタに設定した値を読み出すと異なる [78K0/Kx1、78K0S]

Q1. μPD78F9116 で、スタックをできるだけ確保したく、スタック・ポインタに 0FF00H をセットしましたが、読み出すと 0FB00H になっていました。

A1. スタック・ポインタは、高速 RAM 領域だけを指すので、実際に設定できるのは、下位の 10 ビットだけです。0FF00H は高速 RAM 領域ではなく、SFR 領域ですので、高速 RAM 領域に変換されて 0FB00H になります。なお、実際にスタックに値を PUSH する場合には、0FB00H は−1 で 0FAFFH になり、これは高速 RAM 領域ではないので、変換されて 0FEFFH となり、スタック・ポインタに 0FF00H を設定したときと同じ動作になります。

Q2. 78K0/KC1 で、スタック・ポインタに 5555H をセットして、読み出すと 0D555H になっていました。

A2. 78K0/Kx1 では、スタック・ポインタは高速 RAM 領域にしか設定できません。5555H は高速 RAM 領域でないので、上位 6 ビットが高速 RAM 領域に変換されて 0D555H になります。

スタック・ポインタの設定 [78K 共通]

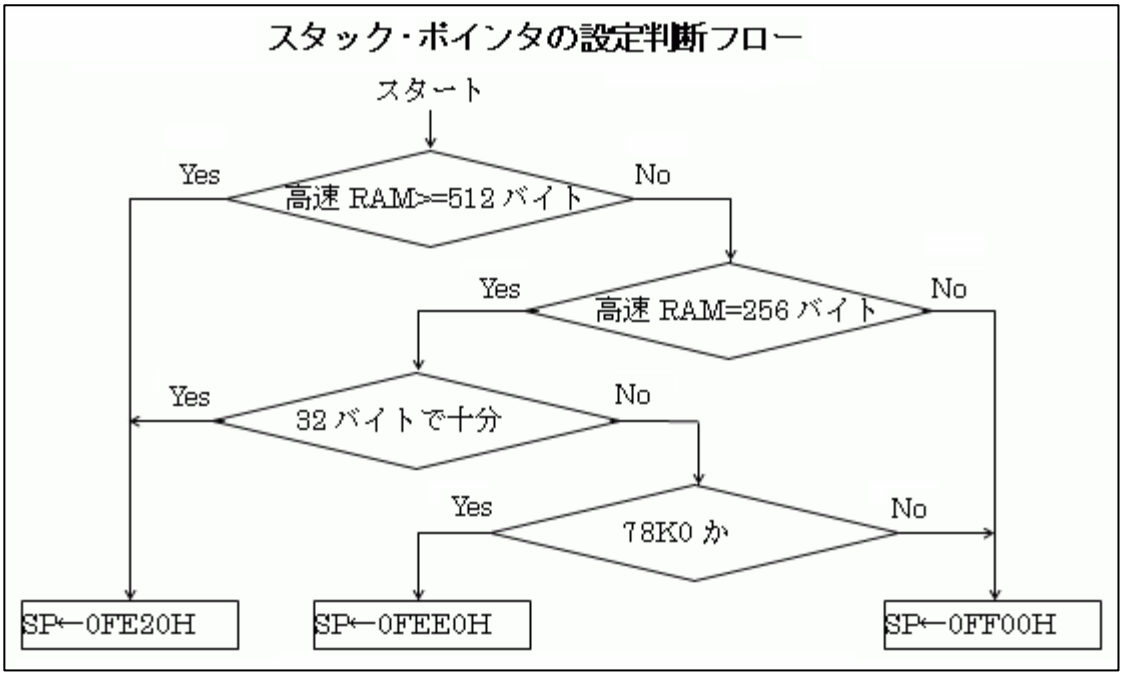
マイコンでは、サブルーチン・コールや割り込み、レジスタの一時的な退避で、スタックを使用します（「スタック」もご参照ください）。Cコンパイラを使用する場合には、コンパイラが自動的に、空き領域の中で最も大きい部分を探して、スタックとして設定してくれるので、あまり気にする必要はありません。ただし、コンパイラの実行中に、スタックがどれだけ消費されるかは分かりませんので、全く気にする必要がないわけではありません。

[使用できるメモリ]

78K0 や 78K0S には、内蔵 RAM に何種類かあります。この中で、高速 RAM だけがスタックに利用できます（「マイコン内蔵メモリの特徴と使い方」もご参照ください）。

[高速 RAM へのスタックの設定]

高速 RAM のどこにでも、スタックは設定できます。スタック・ポインタに設定する値については、次の判定フローに示す、いくつかのパターンが考えられます。



(1) 高速 RAM が 512 バイト以上の場合

saddr 領域の直前をスタックで使用するため、0FE20H を設定します。

(2) 高速 RAM が 256 バイトで、スタックには 32 バイトで十分な場合

saddr 領域を避けて、0FE20H を設定します。

つまり、78K0 の All Flash の製品では、0FE20H が標準的な設定といえます。

78K0S/Kx1+で、高速 RAM が 256 バイトの場合も同様です。

(3) 高速 RAM が 256 バイトで、スタックが 32 バイトでは不足する場合

0FE00H から、スタックに必要な容量を確保したアドレスを設定します。その見積もりが難しい場合には、自由に使える RAM の最後(78K0S は 0FF00H、78K0 は 0FEE0H)を設定します。

(4) 高速 RAM が 128 バイトの場合(78K0S)

0FF00H を設定します。

1.2.27 禁止事項

暴走時に未定義命令を実行

Q1. μPD789116 で、プログラムの空き領域を FFH で埋めた場合、マイコンが暴走してこれを実行すると、どのような動作をするのですか。

A1. FF という命令コードは、78K0S では未定義のコードです。未定義のコードの実行については、CPU がどのような動作を行うかは不明です(テストも行いませんし、保証も一切できません)。

レジスタ未定義の SFR 領域の読み込み

Q1. SFR 領域内で、レジスタが定義されていないアドレスを読み込んでもいいですか。

A1. レジスタが定義されていない SFR 領域のアドレスをアクセスした場合、動作は一切保証されません。

通常、未定義のエリアをリードした場合には、不定値が読み出されます。

そのアドレスに SFR 自体が存在しなければ、書き込みは無意味な動作になるだけですが、そこに未公開のテスト回路などが存在した場合は、デバイスの動作がおかしくなる(暴走する)可能性があります。

1.2.28 リセット兼用端子

リセットと兼用端子のポートとしての使用方法 [小ピン・マイコン]

Q1. 78K0S/KA1+で、リセット端子をポートとして使用する方法と、注意点を教えてください。

A1. リセット端子の兼用ポートを使用する場合、電源投入時に POC 機能でリセットすることになりますが、次の点に注意が必要です。

(1) オプション・バイト(80H 番地)の RMCE ビットを 0 に設定しておきます。

(2) POC によるリセットが解除されるまで、兼用の P34 端子は、ハイレベルに固定しておく必要があります。

これは、78K0S/KY1+でも同様です。詳細については、各デバイスのマニュアルで、オプション・バイトの章をご参照ください。

1.2.29 初期設定

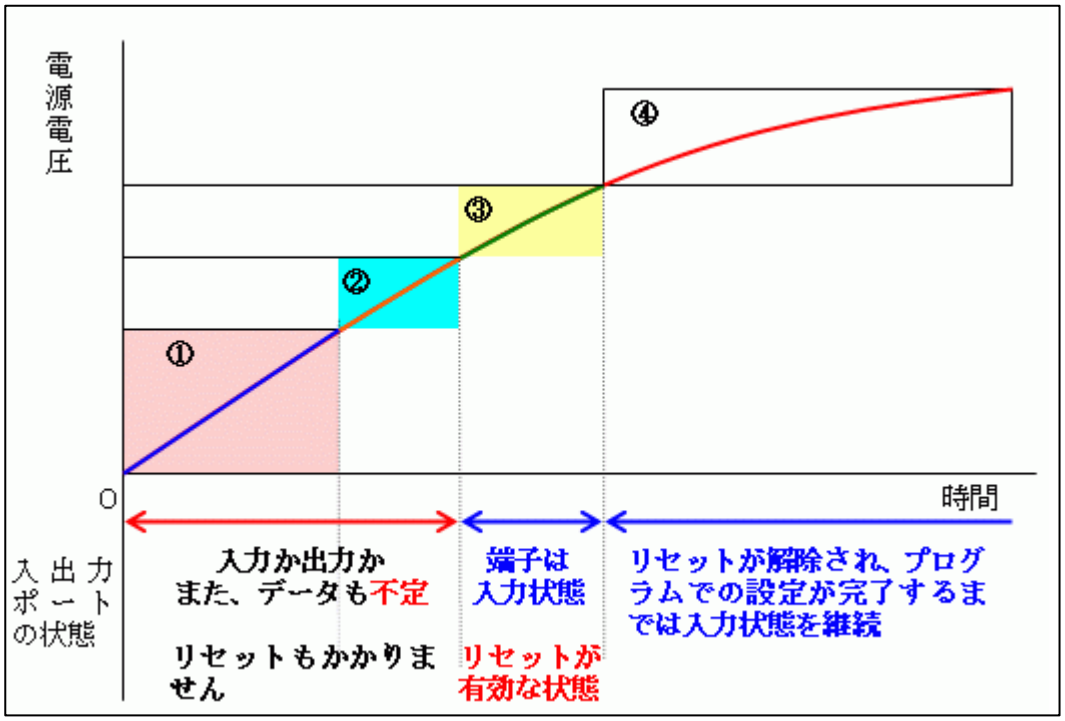
マイコンの電源投入からの動き

[はじめに]

マイコンを使用する場合、起動後の動きは比較的よく理解されていますが、そこまでの動きはあまり知られていません。ここでは、起動前の動きを中心に説明します。

[電源投入後]

マイコンに電源が投入されても、すぐに動作可能にはなりません。電源が投入され、徐々に電圧が上昇していきますが、約 1V（この電圧は製造プロセスにより異なります）までは半導体として動作しません（次図の①の領域）。そこから、内部のトランジスタが動作できるようになります。この状態では、最初はあくまでもいくつかのトランジスタがバラバラに動作可能なだけです（次図の②の領域）。一般に、フリップフロップを含む論理回路は、初期化されるまではその状態が不定となります（FAQ の「クロックとリセット」もご参照ください）。このとき**端子の状態は不定**で、**異常な信号が出る**ことがあります。また、リセットも受け付けられません。さらに電源電圧が上昇していき、デバイス全体の回路が動作可能になると、**リセットが受け付けられる**ようになります（次図の③の領域）。



複数の電源をもったマイコンでは、電源の投入順序によっては、リセットが受け付けられる前にポートが動作を開始し、端子に予想外の信号が出ることがあります。

[リセット期間]

POC や外部からのリセット信号により、マイコンが初期化されます。これにより、内部のフリップフロップ類やレジスタ類は初期化され、入出力ポートは入力ポートになります。また、78K0 や 78K0S のマイコンでは、メイン・システム・クロックの発振は停止します。通常は、この期間に電源電圧をマイコンの動作可能電圧以上にあげる必要があります。

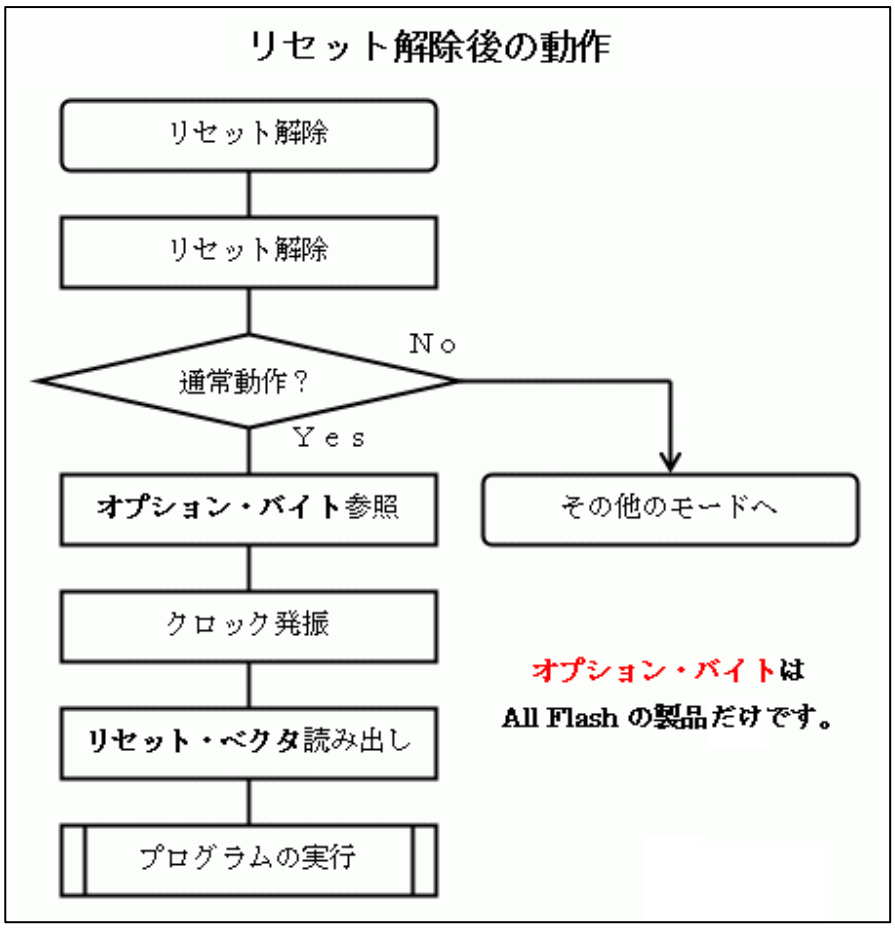
[リセット解除]

さらに電源電圧が上昇して、デバイスとしての動作可能領域（前図の④の領域）となります。ここで、リセットが解除されても、すぐにマイコンがプログラムを実行できるわけではありません。動作に必要な内部の電源供給を開始するといった処理が行われます。また、フラッシュの書き込みモードで起動するか、通常動作モードで起動するかの判定も行われます。

通常動作モードで起動すると、**オプション・バイト**が読み出され、その指定による設定が有効になります。

これら一連の処理が完了して、クロックの発振が安定すると、マイコンは 0 番地のリセット・ベクタを読み込み、そのアドレスに分岐することでプログラムの実行を開始します。

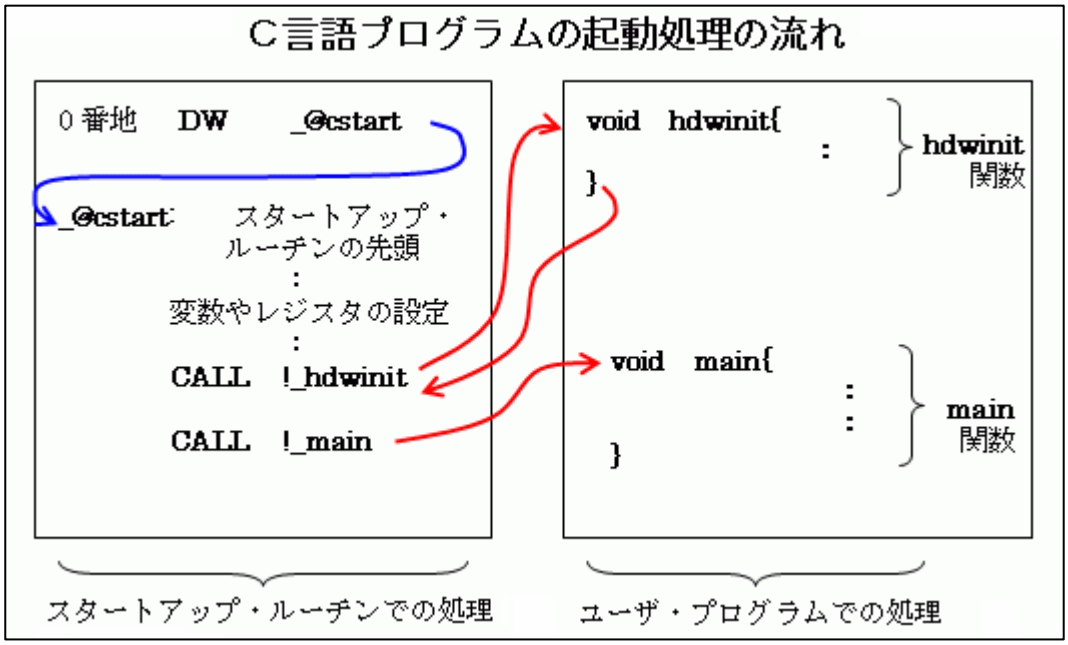
78K0 や 78K0S では、リセット解除からプログラムの実行開始までの時間は、使用するマイコンやクロックによって異なります。



[プログラムの起動]

リセット・ベクタで示されたアドレスからプログラムの実行を開始したあとは、フラッシュ・メモリに書き込まれたプログラムに従って処理を行います。アセンブラでプログラムを記述している場合は、**リセット・ベクタ**はプログラムの先頭のアドレスを指定する必要があります。プログラムがスタートすると、通常は、作業領域として使用するメモリやレジスタと内蔵周辺機能の初期化を行います（内蔵周辺機能の初期化については、「初期化処理」もご参照ください）。

C 言語でプログラムを記述する場合は、**リセット・ベクタ**は**スタートアップ・ルーチン**のアドレスが設定され、これらの初期化処理は**スタートアップ・ルーチン**で行われます。内蔵周辺機能の使い方は個々のアプリケーションで異なるので、内蔵周辺機能の初期化は hdwinit 関数として作成しておきます。**スタートアップ・ルーチン**は、変数の初期化などを行った後、main 関数を呼び出す前に hdwinit 関数を実行するようになっています。



hdwinit 関数の処理内容については、「最初に実行する内容について[78K 共通]」もご参照ください。

初期化処理 [78K0S/Kx1+]

[はじめに]

78K0S/Kx1+に内蔵された周辺機能は、1 端子に複数の機能が割り当てられていることがあります。そのために、初期化段階でどの機能を使用するかを指定する必要があります。また、内蔵された周辺機能もいくつかのモードをもち、目的に応じて初期設定で指定する必要があるものがあります。ここでは考え方だけを説明します。具体的な設定は**初期化処理プログラム**をご参照ください。マイコンの起動については、「マイコンの電源投入からの動き」もご参照ください。

また、内蔵 RAM や汎用レジスタも、電源投入時にはその内容が不定になっています。したがって、初期設定の中で RAM の初期化も行う必要があります。これらがきちんと行われていないときに、値が不定の内蔵 RAM や汎用レジスタを参照するようなプログラムを作成すると、その値で処理が変わってしまい、不安定な動きをすることになります。

C言語を使用する場合は、通常はスタートアップ・ルーチンと呼ばれるプログラムがまず起動して、使用する変数(内蔵 RAM に配置されている)を初期化するので、特に意識する必要はありません。また、スタック・ポインタについても、適当なところに設定してくれるので意識する必要はありません。

[初期設定の考え方]

78K0S/Kx1+を用いたシステムを設計する際、最初に行うのはどのような機能を使用するか、特にポート以外のどの内蔵周辺機能を使用するかの決定です。ポートについては、入力ポートと出力ポートの数が確保できれば、まずは問題ありません。ドライブ能力や出力の構成(オープン・ドレーンかどうか)、入力レベル(耐圧)など、ポートについての検討項目は、78K0S/Kx1+では該当しません。

78K0S/Kx1+では、ポート以外の内蔵周辺機能として、以下の機能があります(内蔵周辺機能で使用しない端子は、ポートとして使用できます)。

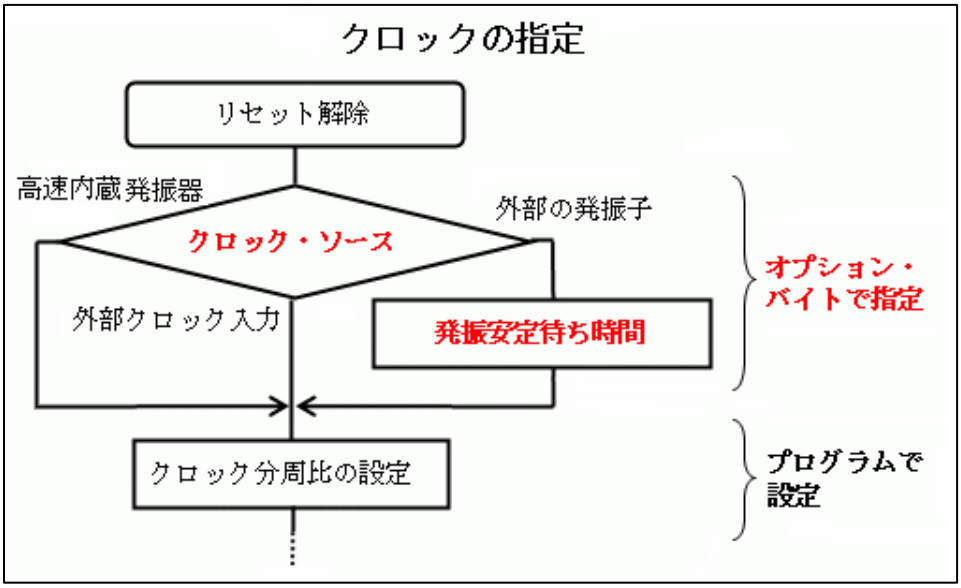
①クロック機能（外部の発振子／外部クロック入力／内蔵高速発振器）
②ウォッチドッグ・タイマ
③RESET 端子機能
④シリアル(UART)機能（内蔵しないデバイスもある）
⑤タイマ機能
⑥外部割り込み機能
⑦アナログ入力機能（内蔵しないデバイスもある）
⑧低電圧検出機能

これらの中で①③は、起動時にオプション・バイトで指定することになります。その他は、通常はプログラムの初期設定で指定します。これらの指定は、携帯機器で特に省電力モードでの電流の制限が厳しい場合を除くと、起動時の初期設定で指定しておくことをお勧めします。
なお、1端子に複数の機能が割り当てられていても、端子の機能を切り替えて使用するのは問題があるので推奨できません。バスのようにデータの方向が切り替わったりするものを除いて、すべての信号には出力元と入力先が存在し、通常はその間の関係が固定されています。そのような接続関係にある信号の端子の機能を切り替えるには、お互いの間で問題が生じないように、十分な検討をする必要があります。どのような問題が発生するかは、個々のケースで異なるので、これでいいという画一的な対策はありません。

【コーヒー・ブレーク】
データ・バスなど信号の方向が切り替わるバスでは、ハイ・インピーダンス状態に注意が必要です。バスが発振状態になることがあり、十分なアクセス時間を確保したつもりでも、発振によりバスが安定せず、正常なアクセスができないことがあります。この場合、バスを抵抗でプルアップすれば、発振は収まります。
また、ハイ・インピーダンス状態では、端子が長期間にわたって中間電位にとどまることがあり、これが貫通電流の原因になることもあります。詳細は FAQ の貫通電流をご参照ください。

[動作クロックの設定]

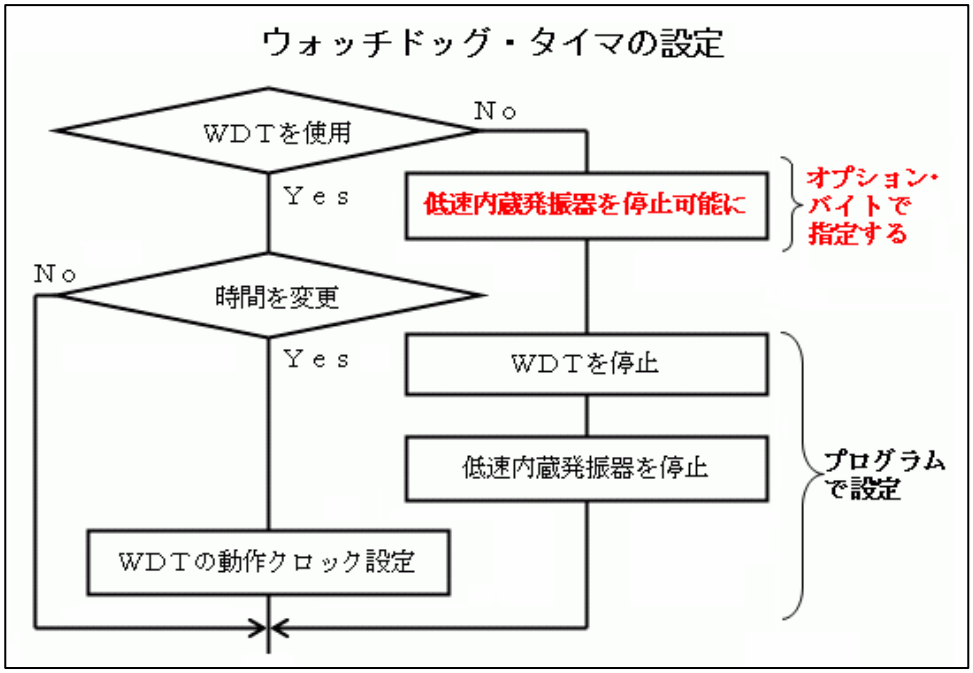
マイコン動作の基本となるのが、動作クロックです。通常は最初に設定します。使用するクロック・ソース（外部の発振子／外部クロック入力／高速内蔵発振器）はオプション・バイトで指定するので、初期設定では CPU の動作クロックの分周比や周辺回路のクロックの分周比を指定することになります。この関係を次図に示します。



78K0S/Kx1+では、CPU の動作クロックや A/D の変換時間が、電源電圧の影響を受けます。そこで、クロック周波数を電源電圧に応じてダイナミックに変更するか、固定で使用するかを決める必要があります。
電源電圧に応じて動作クロックを変更するには、低電圧検出機能などを用いて、電源電圧を監視する処理を追加して、その中でクロックの分周比を変更する必要があります。電源電圧の低下に対しては割り込みで処理し、上昇に対してはポーリングで処理します。
また、動作クロック周波数を固定するときは、低電圧検出機能を用いて、動作クロック周波数に対応した電圧まで電源電圧が上昇するのを待ちます。この場合、前述のクロック指定の「クロック分周比の設定」の前に、電源電圧が規定の電圧になったかどうかの判断分岐を追加します。

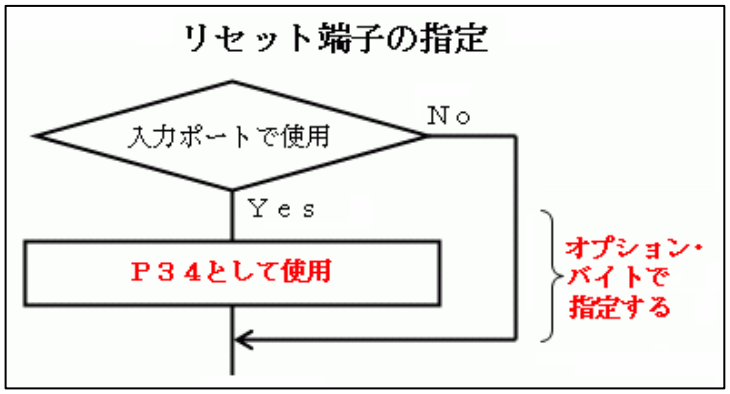
[ウォッチドッグ・タイマの設定]

ウォッチドッグ・タイマの指定には、低速内蔵発振器の設定も関連します。ウォッチドッグ・タイマを使用しないときは、オプション・バイトで「低速内蔵発振器を停止可能」に設定してから、初期設定プログラムで WDT を停止して、さらに低速内蔵発振器を停止します。
ウォッチドッグ・タイマ(WDT)は、何も設定しないと、最長の期間でオーバ・フローするモードで動作を開始します。その状態で使用するなら、プログラムでの設定は不要です。オーバ・フローする時間を変更する場合には、動作クロックを変更しますが、動作クロックを低速内蔵発振器でなくシステム・クロックに変更するときは、オプション・バイトで内蔵低速発振器を停止可能に設定しておく必要があります。
動作クロックの変更は 1 度しかできませんので、必ず初期設定の中で処理してください。



[リセット端子の設定]

78K0S/Kx1+では、外部からのリセット入力端子も入力ポートとして使用でき、その設定はオプション・バイトで行います。
オプション・バイトでの指定は、POC(パワー・オン・クリア)解除後に有効になり、それまでこの端子はリセット端子として機能します。そのため、電源投入時(正確には POC 解除時点)に端子にロウ・レベルが入力されていると、リセットがかかり続けてしまいます。そのため、通常はハイ・レベルで、必要なときだけロウ・レベルとなる信号しか入力できません。



[シリアル(UART)機能の設定](78K0/KY1+、KU1+にはない)

78K0S/Kx1+の UART では、LIN バスに対応した機能を内蔵していますが、ここでは通常の UART 機能についてだけ説明します。UART 関係の設定は、すべてプログラムで行います。

UART 機能を使用するには、通信速度以外にフォーマットを通信に合わせする必要があります。つまり、データのビット長(7／8 ビット)、パリティの有無、ストップ・ビット長(1/2 ビット)とビット順序を指定する必要があります。

これ以外に、送信許可や受信許可の制御がありますが、動作電流が特に気になる場合以外、初期設定の段階でこれらを許可するのがいいでしょう。

UART6 について設定が完了したら、対応したポートの設定を行って、初期設定は完了です。あとは、割り込みを使用するために、対応した割り込み要求フラグをクリアして、マスクを解除します。通信完了をベクタ割り込みで処理するには、通信時に CPU の割り込みを許可します。ベクタ割り込みを使用しない場合は、CPU を割り込み禁止状態で使用してください(このとき、UART 関連の割り込みでスタンバイを解除することはできます)。他で割り込みを使用し、UART をポーリングで処理する(スタンバイを解除しない)場合、割り込みはマスクしたままで、割り込み要求フラグをポーリングしてください。割り込みについては、「割り込みの動作の基本」をご参照ください。



[タイマ機能の設定]

タイマの働きについては、「タイマ・カウンタの基礎的な内容」もご参照ください。

タイマの設定は、どのような使い方をするかで大きく異なります。用途は、以下の 4 つに大きく分けられます。各用途のうしろのタイマの名称は、その使い方ができるタイマを示します。実際のマイコンでは、同じタイマを複数内蔵することを意識して、TM00 のように、ここで示したタイマ名称のうしに数字が追加されます。

(1) 時間測定（インターバル・タイマ）：TM0、TMH、TM8
(2) 外部イベント・カウント：TM0
(3) パルス幅／間隔測定：TM0
(4) パルス出力：TM0、TMH

(1)のインターバル・タイマはもっとも基本的な使い方で、タイマと割り込みの設定だけで使用できます。

8 ビット・タイマ TM8、TMH について、初期設定の例を次に示します。ここでは、タイマはまだ起動せず、モードの設定までです。実際にタイマを使用するとき、タイマ動作を許可に設定してください。

TM8、TMHでのインターバル・タイマの設定

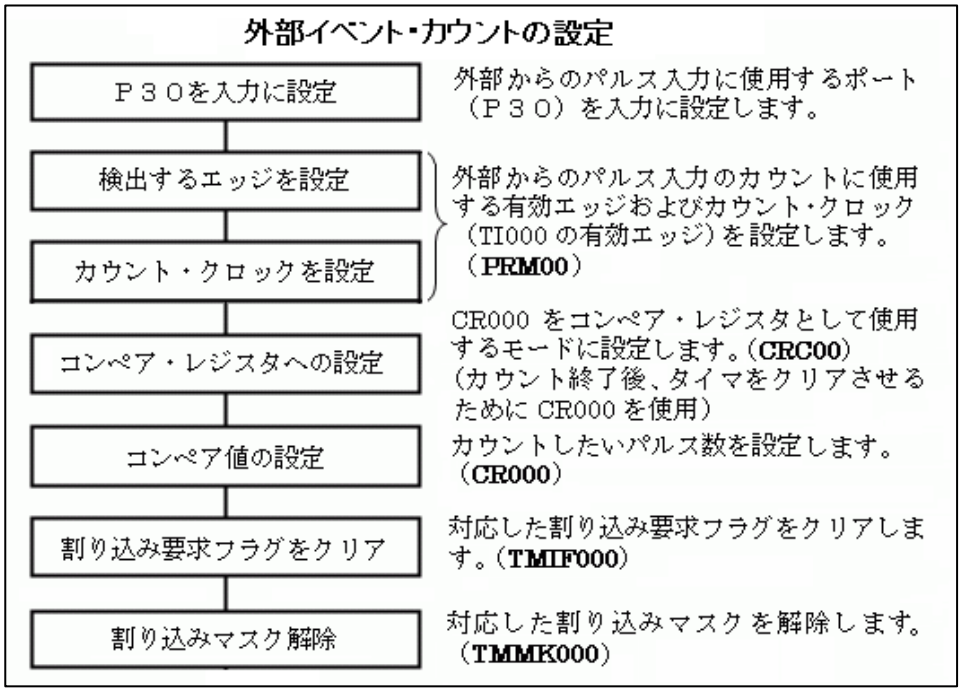
カウント・クロックを設定	設定したい時間間隔と、必要な精度からカウント・クロックを選択します。（TMC80） TMH の場合には同時にタイマ動作モードも設定します（出力は行わないので禁止に）。（TMHMD1）
コンペア値を設定	カウント・クロックと時間間隔からコンペア・レジスタの値を求めて設定します。（CR80、CMP01）
割り込み要求フラグをクリア	インターバル・タイマの割り込みに対応した割り込み要求フラグをクリアします。（TMIF80、TMIFH1）
割り込みマスク解除	インターバル・タイマの割り込みに対応した割り込みマスクを解除します。（TMMK80、TMMKH1）

16 ビット・タイマ TM0 について、初期設定の例を次に示します。16 ビット・タイマでインターバル・タイマを構成するには、CR000 レジスタをコンペア・レジスタとして使用し、割り込みとして INTTM000 を使用します。ここでは、タイマ関連のモード設定と割り込みの初期設定までで、タイマの起動は行いません。必要になったタイミングで「TM00 と CR000 の一致でクリア＆スタート」モードを TMC00 レジスタに設定します。

TM0でのインターバル・タイマの設定

カウント・クロックを設定	設定したい時間間隔と、必要な精度からカウント・クロックを選択します。（PRM0）
コンペア・レジスタへの設定	CR000 をコンペア・レジスタとして使用するモードに設定します。（CRC00）
コンペア値の設定	カウント・クロックと時間間隔からコンペア・レジスタの値を求めて設定します。（CR000）
割り込み要求フラグをクリア	インターバル・タイマの割り込みに対応した割り込み要求フラグをクリアします。（TMIF000）
割り込みマスク解除	インターバル・タイマの割り込みに対応した割り込みマスクを解除します。（TMMK000）

(2)の**外部イベント・カウント**は外部からパルスを入力して、その数をカウントします。カウントした結果が、あらかじめ設定した数に達したら割り込みを発生するような使い方になります。そのため、入力するパルスのどのエッジでカウントするかと、対応したポートを設定する必要があります。



ここでは、タイマ関連のモード設定と割り込みの初期設定までで、タイマの起動は行いません。必要になったタイミングで「**TM00 と CR000 の一致でクリア＆スタート**」モードを TMC00 レジスタに設定します。

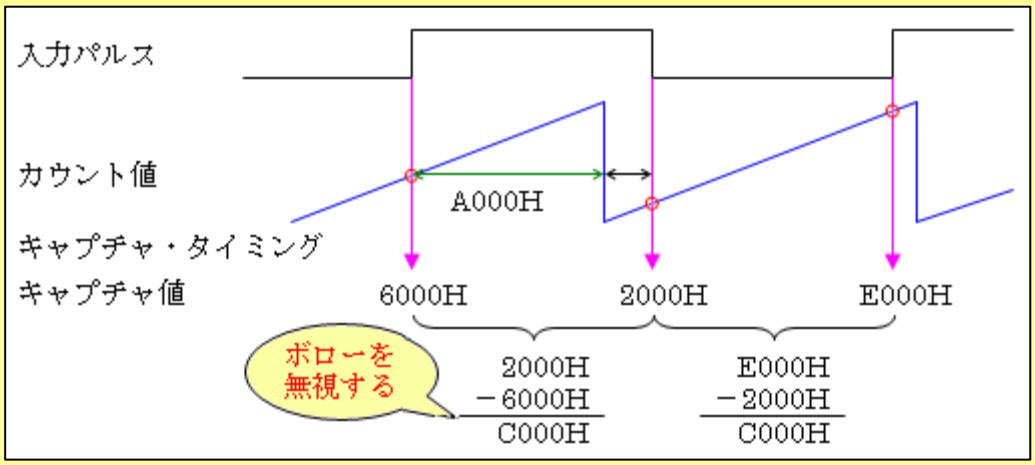
(3)の**パルス幅測定**はタイマのキャプチャ機能を利用して実現します。キャプチャに使用できるレジスタは 2 本内蔵されているので、最大 2 つのパルス幅を測定できます。この場合、タイマはフリーランで使用します。1 つのパルス幅測定では、TI000 端子を使用して「**TI000 の有効エッジでクリア＆スタート**」で使用するのが便利です。
2 つのパルス幅を測定する場合の初期設定の例を次に示します。



これで初期設定は完了し、測定開始時に「**フリー・ランニング**」モードを TMC00 レジスタに設定します。これで、2 つのパルス入力に対して、エッジ検出時のタイマの値が CR000 か CR010 にキャプチャされ、それぞれ対応した割り込みが発生します。そこで、キャプチャされた値と前回の値との差で、パルス幅を測定できます。

【コーヒー・ブレーク】

2 つのパルスの幅を測定する場合、測定できる最大幅はタイマのビット幅である 16 ビットまでです。この範囲なら、割り込み処理の中で、今回キャプチャされた値から前回のキャプチャ値を単純に 16 ビット幅で引き算（ボローは無視）するだけで、幅を計算できます。次に例を示します。

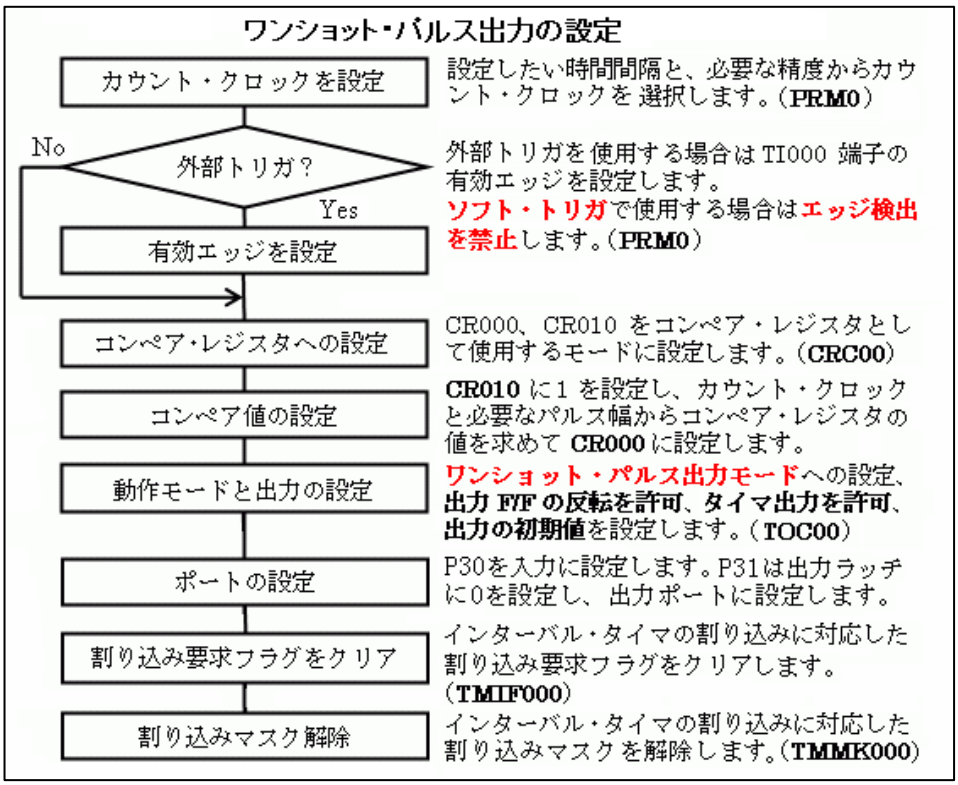


また、立ち上がりと立ち下がりの両エッジを有効にした場合は、割り込みだけではどちらのエッジかを判断できません。でも、パルスのハイ・レベルの期間かロウ・レベルの期間かは、割り込み処理の最初に入力パルスの状態をポートから読めば判断できます。このとき、ポートから読んだ値が1なら、検出したエッジは立ち上がりですから、そのとき計算されるパルス幅はロウ・レベルの幅になります。



(4)のパルス出力には、連続パルスの出力と単発パルスの出力があります。また、連続パルス出力には、方形波出力と PPG 出力、PWM 出力があります。

単発のパルス出力（ワンショット・パルス出力）は、TM0 だけの機能です。2 つのコンペア・レジスタを使って、トリガがかかってから、指定された時間後に指定された幅のパルスを出力します。次に初期設定の例を示します。ここでは、まだタイマは起動させません。必要になったタイミングで「**フリー・ランニング**」か「**TI000端子の有効エッジでクリア＆スタート**」を設定して、TM0 を起動してください。その後、トリガをかけます。



連続パルス出力のうち**方形波出力**は、インターバル・タイマでタイマから出力をするようにしたものです。タイマの動作モードは、インターバル・タイマ・モードと同じ設定にします。TMH で方形波を出力する場合の初期設定例を次に示します。ハイ・レベル期間とロウ・レベル期間はインターバル時間と同じになり、デューティ比 50%の出力となります。

実際に出力を行うには、この初期設定のあと、起動したいタイミングで TMH をスタートさせてください。なお、この設定では、割り込みを使える設定をしていますが、割り込みを使用しない場合は、割り込みのマスクをセットしたままにしておきます。

TMHでの方形波出力の設定

カウント・クロックを設定	設定したい時間間隔と、必要な精度からカウント・クロックを選択します。(**PRM0**)
動作モードと出力の設定	同時に、タイマ動作モードも**インターバル・タイマ・モード**に設定します。出力は許可に設定し、出力の初期値を指定します。(**TMHMD1**)
コンペア値を設定	カウント・クロックと時間間隔からコンペア・レジスタの値を求めて設定します。(**CMP01**)
ポートの設定	P42の出力ラッチを0に設定し、出力ポートに設定します。
割り込み要求フラグをクリア	TMH1 の割り込みに対応した割り込み要求フラグをクリアします。(**TMIFH1**)
割り込みマスク解除	TMH1 の割り込みに対応した割り込みマスクを解除します。(**TMMKH1**)

TM0 で方形波を出力する場合の初期設定の例を次に示します。実際に出力を行うには、この初期設定のあと、起動したいタイミングで「**TM00 と CR000 の一致でクリア＆スタート**」モードでタイマを起動します。CR000 をコンペア・レジスタに使用しますが、CR010 をコンペア・レジスタにする場合は、「CR000＜CR010」に設定するか「CR010 と TM00 の一致でタイマ出力が反転しない」ように設定してください（CR010 と TM00 の一致でタイマ出力の反転を許可し、CR000＞CR010 に設定すると、PPG 出力になります）。

TM0での方形波出力の設定

カウント・クロックを設定	設定したい時間間隔と、必要な精度からカウント・クロックを選択します。(**PRM0**)
コンペア・レジスタへの設定	CR000 をコンペア・レジスタとして使用するモードに設定します。(**CRC00**)
コンペア値の設定	カウント・クロックと時間間隔からコンペア・レジスタの値を求めて設定します。(**CR000**)
動作モードと出力の設定	**連続パルス出力モード**への設定、**出力F/Fの反転を許可、タイマ出力を許可、出力の初期値**を設定します。(**TOC00**)
ポートの設定	P31は出力ラッチに0を設定し、出力ポートに設定します。
割り込み要求フラグをクリア	インターバル・タイマの割り込みに対応した割り込み要求フラグをクリアします。(**TMIF000**)
割り込みマスク解除	インターバル・タイマの割り込みに対応した割り込みマスクを解除します。(**TMMK000**)

PPG 出力は、アクティブ期間と非アクティブ期間を指定して、パルスを出力する機能です。基本的な初期設定は方形波出力と同じで、違いは次のような点です。

CR010 もコンペア・レジスタで使用する。

CR010 と TM00 の一致でもタイマ・フリップフロップが反転する。

CR010 に設定する値を CR000 に設定する値より小さくする。

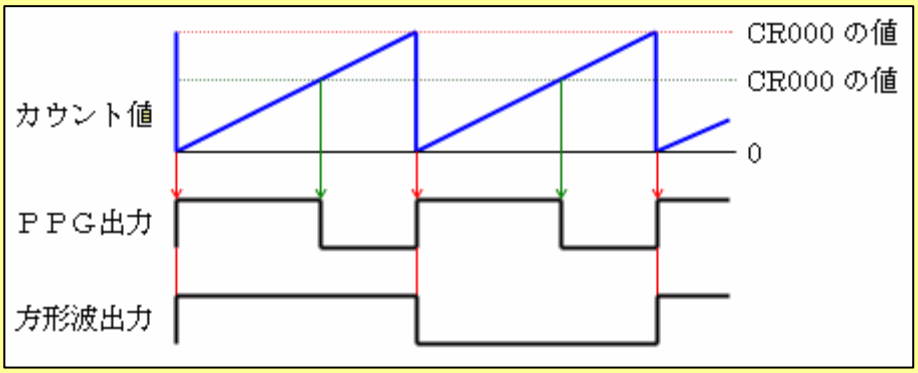
これにより、「**TM00 と CR000 の一致でクリア＆スタート**」モードでタイマを起動すると、TM00 の出力は、CR010 との一致でアクティブから非アクティブに変化します。その後もカウントを続け、CR000 と一致すると、出力はアクティブに変化して、TM00 はクリアされます。この動作波形は、PWM 出力と類似になります。16 ビットの PWM 出力が必要な場合は、PPG 出力を使用します。

TM0でのPPG出力の設定

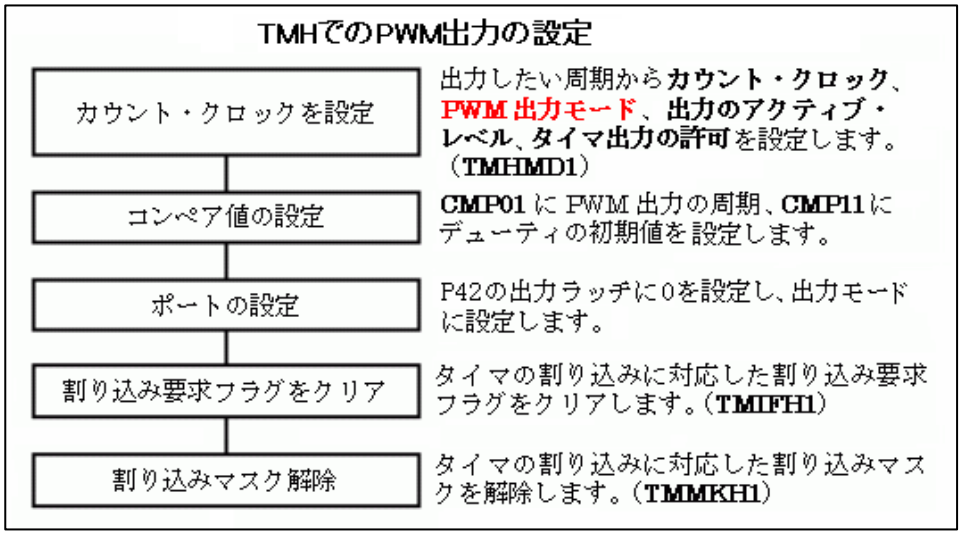
カウント・クロックを設定	設定したい時間間隔と、必要な精度からカウント・クロックを選択します。(**PRM0**)
コンペア・レジスタへの設定	CR000、CR010 をコンペア・レジスタとして使用するモードに設定します。(**CRC00**)
コンペア値の設定	PPG出力の周期から決定した値を **CR000** に設定します。**CR010** にはPWM出力のアクティブ期間の初期値を設定します。
動作モードと出力の設定	**連続パルス出力モード**への設定、**出力F/Fの反転を許可**、**タイマ出力を許可**、**出力の初期値**を設定します。(**TOC00**)
ポートの設定	P31は出力ラッチに0を設定し、出力ポートに設定します。
割り込み要求フラグをクリア	インターバル・タイマの割り込みに対応した割り込み要求フラグをクリアします。(**TMIF000**)
割り込みマスク解除	インターバル・タイマの割り込みに対応した割り込みマスクを解除します。(**TMMK000**)

【参　考】

方形波出力では、CR000 との一致での出力反転と TM00 のクリアが、1 周期に 2 回発生します。このため、出力は CR000 で指定した期間の 2 倍の周期となります。PPG 出力では、CR010 は出力を反転するだけで、CR000 で指定した期間が周期となります。波形の例を次に示します。



PWM 出力はアナログ情報を出力する場合に用います。TMH の PWM 出力は、カウント・クロックと CMP01 で周期を指定し、CMP11 でデューティを制御します。TMH1 で PWM 出力を行う場合の初期設定の例を次に示します。このあと、必要なタイミングで TMH1 を起動してください。



[タイマの使用モード（まとめ）]

タイマは、使用目的に応じて動作モードを設定する必要があります。その一覧を次にまとめます。

用途＼タイマ	TM00	TMH1	TM80
インターバル・タイマ	TM00とCR000の一致でクリア&スタート	インターバル・タイマ・モード	指定不要
イベント・カウント	TM00とCR000の一致でクリア&スタート	－	－
パルス幅測定	フリー・ランニングかTI000端子の有効エッジでクリア&スタート	－	－
方形波出力	TM00とCR000の一致でクリア&スタート	インターバル・タイマ・モード	－
ＰＰＧ出力	TM00とCR000の一致でクリア&スタート	－	－
ＰＷＭ出力	－	ＰＷＭ出力モード	－

また、一部の機能では、入出力のために関連するポートの設定をする必要があります。

用途＼タイマ	TM00	TMH1
インターバル・タイマ	－	－
イベント・カウント注	PM30／PM20＝1	－
パルス幅測定注	TI000 ： PM30／PM20＝1 TI010 ： PM31／PM21＝1	－
方形波出力注	P31／P21 ＝ 0 かつ PM31／PM21 ＝ 0	P42／P20 ＝ 0 かつ PM42／PM20 ＝ 0
ＰＰＧ出力注	P31／P21 ＝ 0 かつ PM31／PM21 ＝ 0	－
ＰＷＭ出力注	－	P42／P20 ＝ 0 かつ PM42／PM20 ＝ 0

注　タイマの入出力端子と兼用しているポートは、デバイスにより異なります。表中の"／"の左側は KB1+や KA1+の場合で、右側は KY1+や KU1+の場合です。

[外部割り込みの設定]

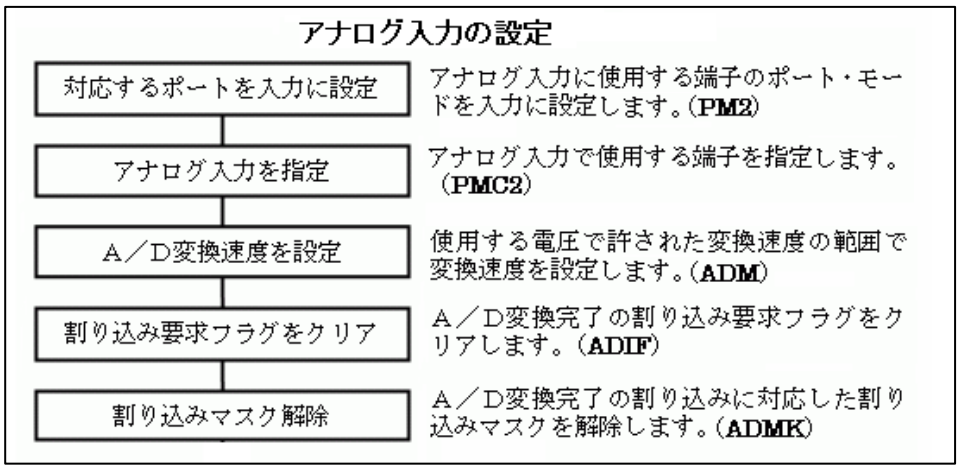
78K0/Kx1+のデバイスでは、2～4 本の外部割り込み要求信号をもっています。これらでの割り込みは、入力のエッジ検出を行い、有効エッジを検出したら割り込み要求フラグをセットします。このため、外部割り込みを使用するには、初期設定で有効エッジの指定を行う必要があります。外部割り込みを使用する場合の初期設定の例を次に示します。これで、EI 命令を実行して CPU が割り込み受け付け可能になると、外部から有効エッジの入力があれば、ベクタ割り込みを発生します。CPU が割り込み受け付け禁止の場合、ベクタ割り込みは発生しませんが、STOP モードのようなスタンバイ状態を解除できます。



【注　意】
INTP0 やINTP2 は、TM0 のキャプチャ入力と端子が共通になっています。しかも、エッジ検出で割り込みが発生するところまで同じです。でも、STOP モード時の動作が大きく異なります。キャプチャ入力信号で使用する場合は、TM0 のカウント・クロックでサンプリングされてノイズ除去されます。このため、TM0 のカウント・クロックが停止する STOP モードでは、キャプチャ入力のエッジ検出ができません。一方、外部割り込みは、クロックによるサンプリングを行わないので、STOP モード中でも機能し、STOP モードの解除に使用できます。

[アナログ入力機能の設定]（一部に内蔵していない製品があります）

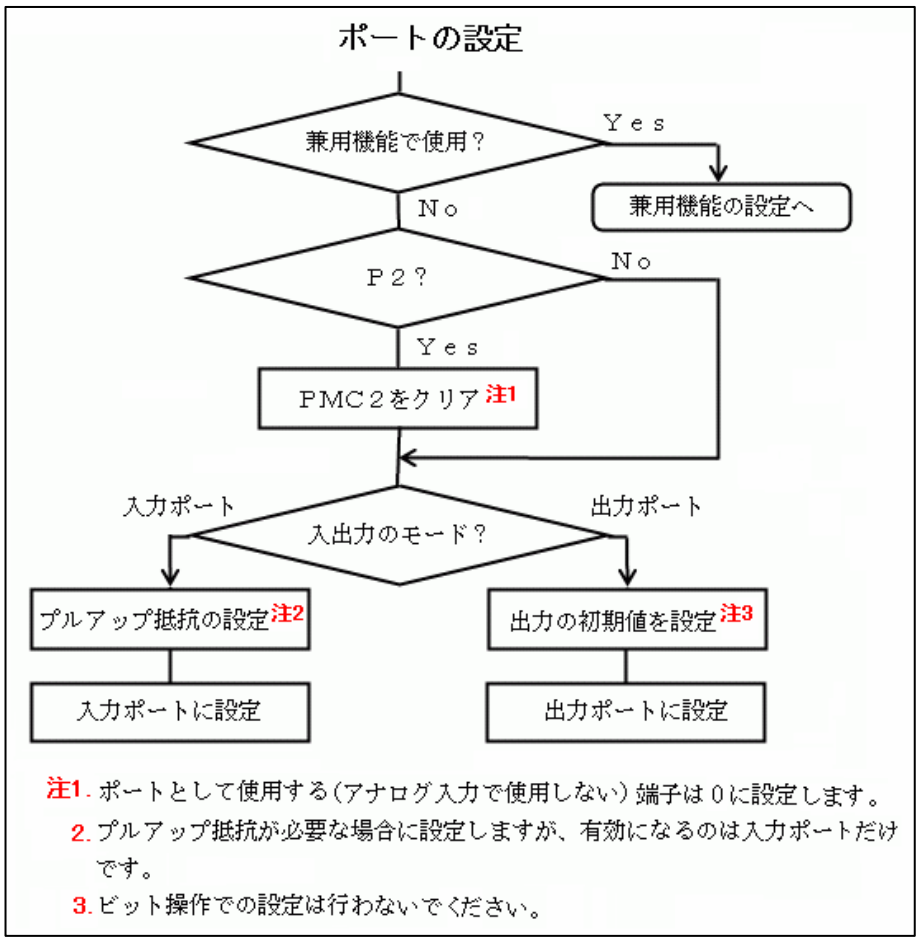
アナログ入力を扱うには、A/D変換の変換速度をデバイスの動作電圧に応じて設定する必要があります。また、端子を兼用しているポートを入力に設定するだけでなく、PMC2 レジスタの設定を行います。これで、実際にアナログ入力が必要なタイミングで ADCE ビットを 1 にセットし、ADS レジスタで入力を指定して、A/D 変換を起動（ADCS ビットを 1 にセット）すれば、変換がスタートします。



【注　意】
A/D 変換を行うには、通常は動作していないコンパレータを動作させる必要があります。ADCE ビットを 1 にすれば、コンパレータが動作するようになりますが、1μs の待ち時間が必要です。そのため、ADCE ビットをセットして 1μs 以上経過してから、ADCS ビットを 1 にセットして変換動作をスタートさせます。
ADCE ビットをセットしておくと、A/D 変換を停止してもコンパレータが動作するので電力を消費します。STOP モードで省電力にするには、ADCE ビットをクリアして、コンパレータでの電力消費を抑えます。
ADCEビットを用いないで、ADCSビットを1にセットするだけも、コンパレータを動作させ、A/D 変換できます。しかし、この場合は最初の変換結果の精度が得られません（低い入力電圧のときに、正規の値より大きくなることがあります）。これを避けるには、最初の変換結果を無視して、2 回目以降の結果を用いる必要があります。この制限は ADCS ビットをセットしたあとの最初の 1 回だけであり、その後はコンパレータが動作しているので、ADS レジスタで変換する入力を変更しても問題ありません。

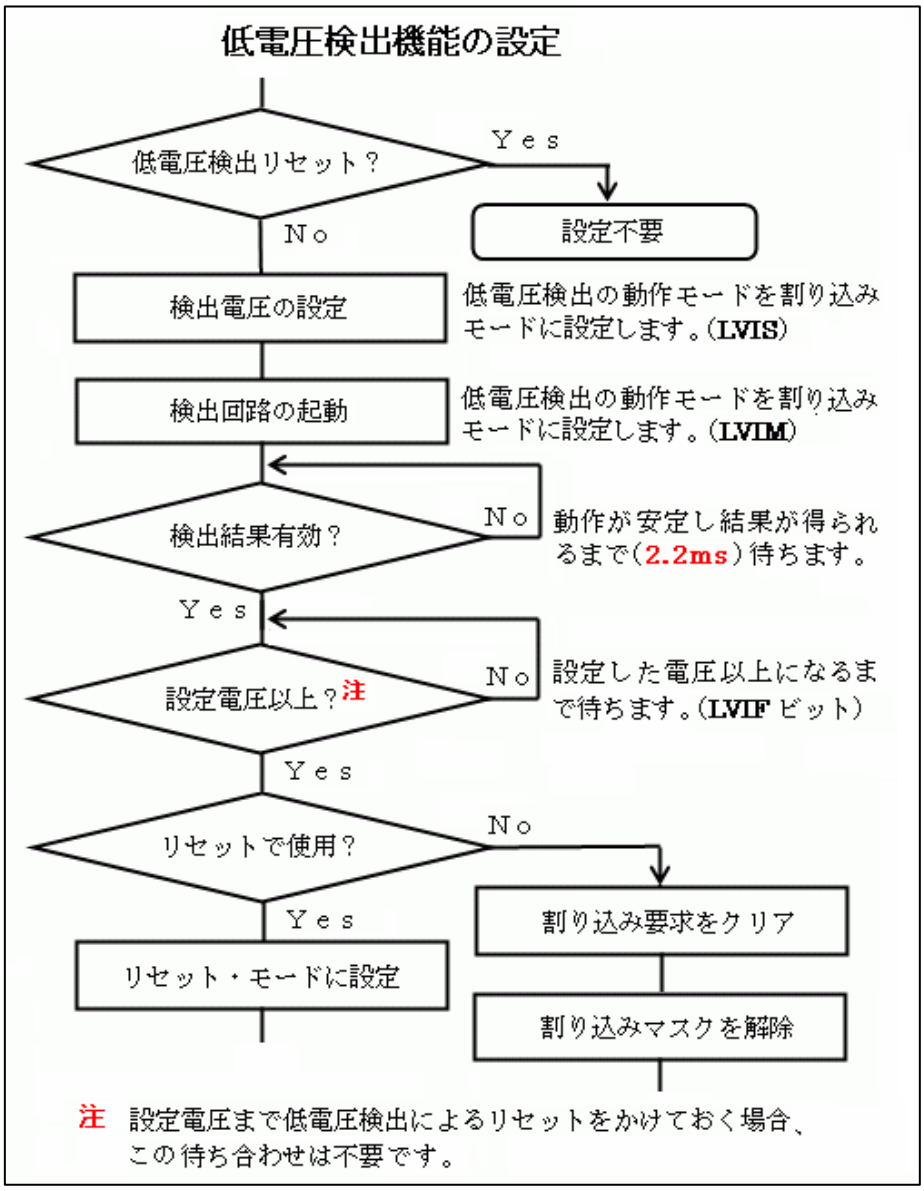
[ポートの設定]

兼用機能で使用しない端子は、ポートとして使用できます。ポートにする初期設定の流れを次に示します。なお、78K0S/KA1+と 78K0S/KB1+だけに存在する P130 は出力専用なので、初期設定は不要です。



[低電圧検出機能]

低電圧検出機能では、初期設定に注意が必要です。低電圧検出機能の設定は、低電圧検出によるリセットでは保持されますが、それ以外の要因（ウォッチドッグ・タイマ、POC、リセット端子）によるリセットではクリアされてしまいます。しかし、どちらの場合にも、リセット解除後は同じベクタ・アドレスからプログラムが起動するので、リセット・コントロール・フラグ・レジスタでどの要因かを判断し、低電圧検出以外の要因によるリセットの場合には、低電圧検出機能の初期設定を行う必要があります。



1.2.30 マイコンでの10進演算

マイコンでの 10 進演算 [共通]

Q1. マイコンで 2 進数から 10 進数に変換するとき、10 進補正命令を使用するのですか。

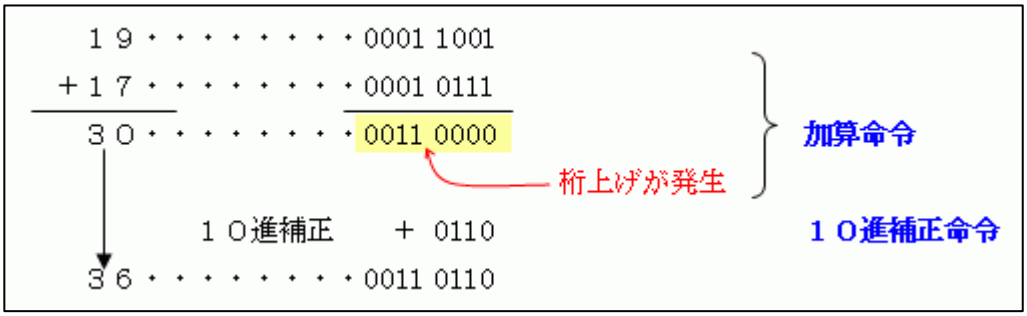
A1. はい、そうです。2 進数で演算を行った結果を 10 進数に変換するときに、10 進補正命令を使用します。

Q2. 2 進数の数値を A レジスタにセットして 10 進補正をしましたが、正しい結果が得られません。

A2. 10 進補正命令は、2 進数を 10 進数に変換する命令ではありません。この変換をするには、そのためのプログラムを作成する必要があります。

Q3. 10 進補正命令は、どのように使用するのですか。

A3. 10 進補正命令は、**10 進数同士で演算**(2 進数での加減算)を行って、その結果で得られた 2 進数の値を 10 進数にするときに、補正するための命令です。たとえば、19＋17 を計算する場合を考えてみます。加算そのものは 2 進数で処理され、30 が得られます。30 自体は 10 進数としておかしな値ではないですが、この加算では下位の 4 ビットからけた上げが発生しています。本来の 10 進数では 10 でけた上げしますが、この場合には 16 でけた上げが発生します。そこで 10 進補正を行うと、下位 4 ビットに 6 が加算され、その結果 36 という正しい 10 進加算の結果が得られます。



このように、4 ビットごとのけた上げや値によって補正するのが 10 進補正の動作で、演算の直後に実行する必要があります。

【コーヒー・ブレーク】
10 進補正では、A レジスタの内容とキャリー・フラグ、補助キャリー・フラグを用います。実際の補正では、A レジスタの内容を上位 4 ビットと下位 4 ビット(10 進数のけたごと)に分けて考えます。加算の場合は、結果の値が A～F になったときか、けた上げが発生(キャリーか補助キャリーをセット)したときに、対象の 4 ビットに 6 を加算(そうでないときは 0 を加算)します。結果は A レジスタとキャリー・フラグに反映されます。
もともと、10 進数同士の演算なので、加算の結果が A～F になり、同時にけた上げが発生することはありません。
10 進補正の詳細な動きについては、78K0S で 10 進補正を行うプログラム例をご参照ください。

Q4. 78K0R/Kx3 には 10 進補正命令がありませんが、10 進補正はできないのですか。

A4. いいえ、できます。

10 進補正は、前記の計算例でも分かるように、演算結果(4 ビットごとのけた上げや値)によって補正値を加算(減算)することで処理されます。つまり、演算結果に応じて、補正する値を発生できれば、その発生した値を加算(減算)すれば同じ結果が得られます。

このように、わざわざ命令として 10 進補正命令がなくても同じことができるので、78K0R では命令セットから 10 進補正命令を削除しています。

【コーヒー・ブレーク】
10 進の計算には、いくつかの方法があります。命令として、10 進加算命令や 10 進減算命令をもつ考え方もありますし、補正で済ませる考え方もあります。ほとんどの場合に 10 補正が用いられています。
初期のマイコンの 8080 では、10 進補正命令を加算でしか使えませんでした。このため、10 進の減算を行うには、かなりの処理が必要でした。その点を改良したマイコンでは、「加算した／減算した」のフラグを準備し、1 つの補正命令で加算と減算に対応できるようにしたものがありました。後者の方が使いやすいのは明らかですが、市場では前者が使用されていたために、互換性の点で後者はなかなか受け入れられませんでした。その後、Z80 が出て、後者の方式が一般的になりました。このフラグを追加する方法は、従来とのソフトの継承性から出てきたものです。継承性を考えなければ、現在の 78K0 のような加算用の補正命令と減算用の補正命令の方が、フラグを必要としないので構成は簡単です。

★ 1.2.31 乗除算

乗除算［78K0］

四則演算、特に乗除算を行う場合、入力データの桁数が問題になります。これは、78K0 のハードウエアがどこまでサポートしているかが影響するからです。78K0 での乗除算では、次のような場合が考えられます。

乗算

被乗数＼乗数	8 ビット以下	16 ビット以下	16 ビット以上
8 ビット以下	乗算命令を使用[注1]	乗除算回路を使用	プログラム処理[注2]
16 ビット以下	乗除算回路を使用	乗除算回路を使用	プログラム処理[注2]
16 ビット以上	プログラム処理[注2]	プログラム処理[注2]	プログラム処理[注2]

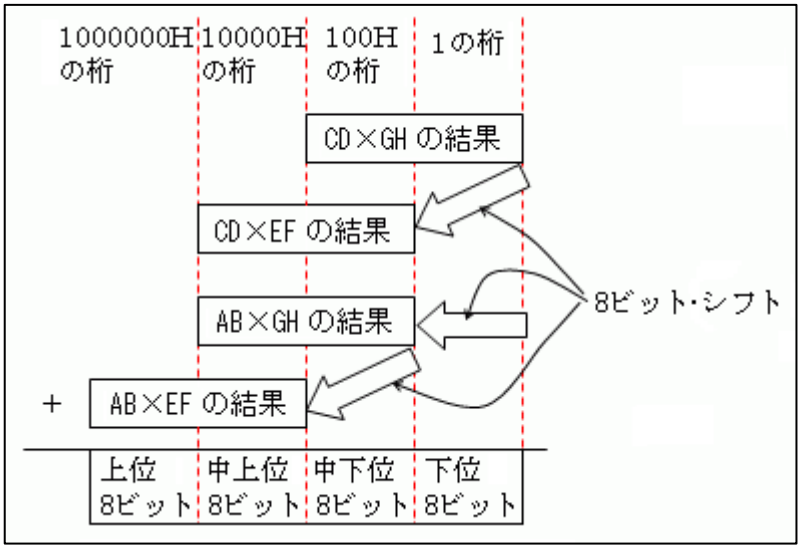
除算

被除数＼除数	8 ビット以下	16 ビット以下	16 ビット以上
16 ビット以下	除算命令を使用[注1]	乗除算回路を使用	―
32 ビット以下	乗除算回路を使用	乗除算回路を使用	プログラム処理[注4]
32 ビット以上	プログラム処理[注3]	プログラム処理[注3]	プログラム処理[注4]

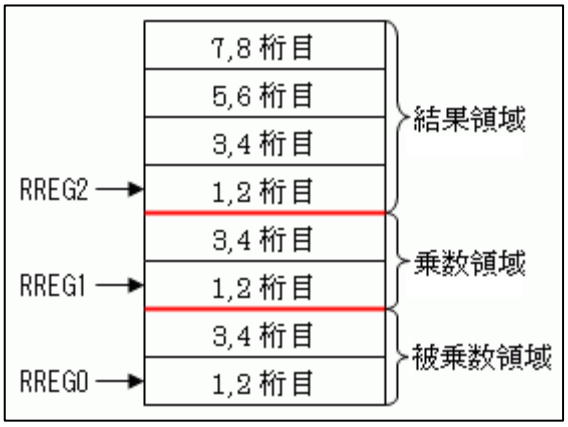
注 1. 乗除算回路でも計算はできますが、命令の方が速く処理できます。
2. 乗算命令／乗除算回路を使用し、何回かに分けて計算を行います。
3. 除算命令／乗除算回路を使用し、何回かに分けて計算を行います。
4. 除算命令／乗除算回路は使用できないので、プログラムで減算を繰り返す必要があります。

(1) 乗算の考え方(基礎)

大きいデータを小さい桁に分けて計算する(通常の筆算と同じ)場合は、位取り(桁)に注意が必要です。ここでは、16 進数の ABCD×EFGH の計算を考えてみます。計算方法は、「MULU X 命令」を使い、CD×GH、CD×EF、AB×GH、AB×EF の 2 桁(8 ビット)ずつ、4 回に分けて計算(乗算)し、これらの結果を桁に注意して加算します。この様子を図に示します。このように、4 回の計算結果の重み(桁)を考慮して加算すれば答がえられます。ここでは、被乗数や乗数は 4 桁(16 ビット)ですが、5 桁以上でも同様に、2 桁ずつの計算結果の桁を考慮して加算すれば計算できます。

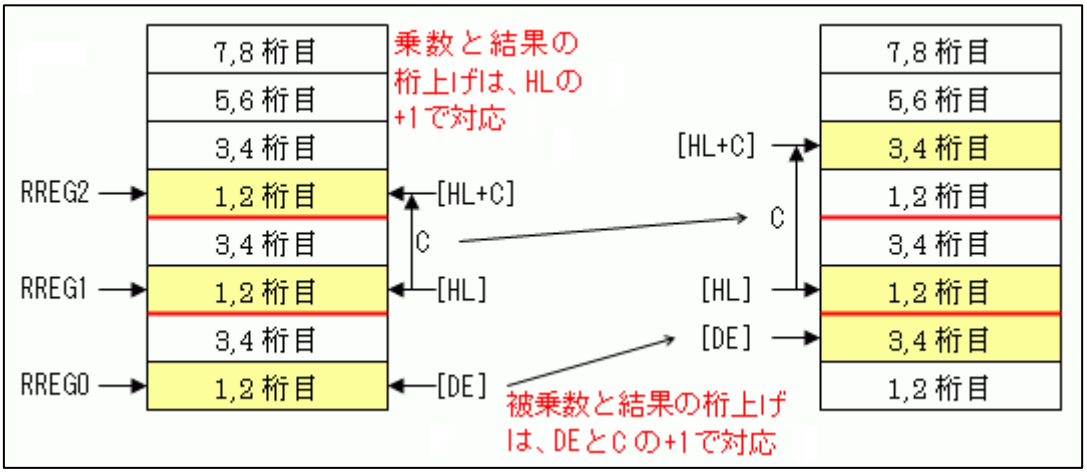


これを具体的なプログラムで考えてみます。なお、データ領域の例を次に示します（乗除算器を使用する場合を考慮して、各領域が偶数番地になるように配置）。




```
;桁数定義
DNMUL           EQU     16/8    ;乗数は 16 ビット
DNMULC          EQU     16/8    ;被乗数は 16 ビット
DNRES           EQU     DNMUL + DNMULC
;データ領域の定義
RREG            DSEG    SADDRP
;下位、上位の順（リトル・エンディアン）
RREG0:          DS      DNMULC  ;被乗数
RREG1:          DS      DNMUL   ;乗数
RREG2:          DS      DNRES   ;結果
CLOOP0:         DS      1       ;ループ・カウンタ
CLOOP1:         DS      1
```

各領域をインデックスするためのポインタの使用方法を工夫すると、効率的にアクセスできるようになります。ここでは、被乗数(**RREG0**)用と乗数(**RREG1**)用にポインタを使用し、結果(**RREG2**)は **RREG1** をベースにしてオフセットを付けてアクセスします。



この図で、DE に **RREG0**、HL に **RREG1** の各アドレスを設定し、C には **RREG2** と **RREG1** の差を設定します。この状態で、被乗数（[DE]）と乗数（[HL]）の乗算結果の下位を[HL+C]に格納し、C を＋1 して、上位を格納します。実際は単に格納するのでなく、すでに格納されているデータに加算して、桁上げをしていきます。乗数の桁上げには、HL を＋1 することで対応します。次に、被乗数の桁上げは、DE を＋1 すると同時に、C も＋1 することで対応できます。

プログラムは、計算結果領域をクリアしてから、被乗数の下位桁と乗数の下位桁を乗算し、乗数を上位に移して計算するという手順で処理していきます。

```
;
;  16 ビット×16 ビット演算
;  入力：被乗数（RREG0）、乗数（RREG1）
;  出力：計算結果（RREG2）
;
M16bitX16bit:
        MOVW    HL, #RREG2-1            ; 結果領域の設定
        MOV     B,# DNRES               ; 領域の大きさ
        MOV     A,#0                    ; クリアの値を設定
CLRLOOP:
        MOV     [HL+B],A                ; メモリのクリア
        DBNZ    B,$CLRLOOP              ; 領域分を繰り返す。
;
;  領域のクリア完了
;
        MOVW    DE,#RREG0               ; 被乗数のアドレス設定
        MOVW    HL,#RREG1               ; 乗数のアドレス設定
        MOV     CLOOP1,#DNMULC          ; 被乗数桁カウンタ
        MOV     C,#DNMUL                ; 結果格納オフセット
;
;  ポインタの使い分けは以下の通り
;  被乗数には DE レジスタ間接（[DE]）でアクセス
;  乗数には HL レジスタ間接([HL])でアクセス
;  結果にはベース･インデックス([HL+C])でアクセス
;
LOOP1:
        PUSH    HL                      ; 乗数、結果のベースをセーブ
;
;  被乗数桁を固定して計算
;
        MOV     CLOOP0,#DNMUL           ; 乗数桁カウンタ
LOOP2:
;
;  被乗数と乗数をセットして計算
;
        PUSH    BC                      ; オフセットをセーブ
        MOV     A,[DE]                  ; 被乗数の読み出し
        MOV     X,A                     ; 被乗数を X にセット
        MOV     A,[HL]                  ; 乗数の読み出し
        MULU    X                       ; A×X を計算
        XCH     A,X                     ; A レジスタに下位をセット
        ADD     A,[HL+C]                ; 新しい結果を累積する
        MOV     [HL+C],A                ; 結果を保存
```

```
;
;   計算結果の加算処理
;
LOOP3:
        MOV     A,#0                    ; 上位桁用にクリア
        INC     C
        XCH     A,X                     ; 上位桁を取り込み
        ADDC    A,[HL+C]                ; 上位桁の加算
        MOV     [HL+C],A                ; 結果を保存
        BC      $LOOP3                  ; 桁上げがあれば、ループ
;
;   1 回分の計算完了
;
        POP     BC                      ; オフセットを復帰
        INCW    HL                      ; 乗数と結果の領域を桁上げ
        DBNZ    CLOOP0,$ LOOP2          ; 乗数を桁上げして次の桁を計算
;
;   1 桁分の被乗数の計算完了
;
        POP     HL                      ; 乗数、結果のベースを復帰
        INCW    DE                      ; 被乗数の桁上げ
        INC     C                       ; 結果保存用オフセットを桁上げ
        DBNZ    CLOOP1,$LOOP1           ; 被乗数の次の桁の計算に
;
;   必要な桁の計算完了
;
```

同じ処理を乗除算器で処理してみます。こちらは 1 回の処理で完了します。

```
M16bitX16bit2:
        MOVW    AX,RREG0                ; 被乗数の取り込み
        MOVW    MDAOL,AX                ; 被乗数のセット
        MOVW    AX,RREG1                ; 乗数の取り込み
        MOVW    MDBO、AX                ; 乗数のセット
        MOV     DMUC0,#10000001B        ; 乗算を開始
        BT      DMUE,$$                 ; 演算完了待ち
        MOVW    AX,MDAOL                ; 結果の下位を読み出し
        MOVW    RREG2,AX                ; 結果の格納
        MOVW    AX,MDAOH                ; 上位の読み出し
        MOVW    RRE2+2,AX               ; 結果の格納
```

(2) 桁数が大きくなった場合の応用

桁数が増えた場合の乗算は、基本的に説明してきた内容を繰り返すだけです。特に、「MULU X 命令」を用いたプログラムは、定数「**DNMULC**」と「**DNMUL**」を変更するだけで、乗算の桁数を変更できます。

乗除算器を用いる場合は、「MULU X 命令」の場合と同様に、乗算の結果を累積すれば対応できます。ただし、結果が 4 バイトになるので、結果の累積部分のループに注意が必要です（AX レジスタだけでなく、たとえば DE レジスタを使って、「00」→「D」→「E」→「A」→「X」とシフトしていけば処理できます。ただし、4 バイト分は必ず処理する必要があります）。また作業用に、**RREG2** のうしろに余分に 2 バイトの領域を確保します。

```
RREGW:          DS      2               ; 作業用
```

さらに、定数「DNMULC」と「DNMUL」を「32/8」に変更して、32 ビット×32 ビットの計算とします。
次に、プログラム例を示します。

```
;************************************************
;  32 ビット×32 ビット演算部本体
;  入力：被乗数（RREG0）、乗数（RREG1）
;  出力：計算結果（RREG2）
;************************************************
M32bitX32bitB:
        MOVW    HL, #RREG2-1            ; 結果領域の設定
        MOV     B,# DNRES               ; 領域の大きさ
        MOV     A,#0                    ; クリアの値を設定
CLRLOOP:
        MOV     [HL+B],A                ; メモリのクリア
        DBNZ    B,$CLRLOOP              ; 領域分を繰り返す。
;
;  領域のクリア完了
;
        MOVW    DE,#RREG0               ; 被乗数のアドレス設定
        MOVW    HL,#RREG1               ; 乗数のアドレス設定
        MOV     CLOOP1,#DNMULC/2        ; 被乗数桁カウンタ
        MOV     C,#DNMUL                ; 結果格納オフセット
        MOV     B,#4-1                  ; 乗除算器結果の桁数－1 を設定
;
;  ポインタの使い分けは次のとおり
;  被乗数には DE レジスタ間接（[DE]）でアクセス
;  乗数には HL レジスタ間接([HL])でアクセス
;  結果にはベースト･インデックス([HL+C])でアクセス
;
LOOP1:
        PUSH    HL                      ; 乗数、結果のベースをセーブ
;
;  被乗数桁上げ
;
        MOV     CLOOP0,#DNMUL/2         ; 乗数桁カウンタ
;
;  被乗数のセット
;
        MOV     A,[DE]                  ; 被乗数の下位 8bit 読み出し
        MOV     X,A                     ; 被乗数を X にセット
        INCW    DE
        MOV     A,[DE]                  ; 被乗数の上位 8bit 読み出し
        MOVW    MDB0,AX                 ; 被常数を乗除算器にセット
LOOP2:
        PUSH    BC                      ; オフセットをセーブ
;  乗数のセット
        MOV     A,[HL]                  ; 乗数の下位 8bit 読み出し
        MOV     X,A                     ; 下位を X レジスタへ
        MOV     A,[HL+1]                ; 乗数の上位 8bit 読み出し
        MOVW    MDA0L,AX                ; 乗数のセット
```

```
;   計算処理開始
        MOV     DMUC0,#10000001B        ; 乗算開始
        BT      DMUE,$$                 ; 演算完了待ち
;   計算処理完了
        MOVW    AX,MDA0H                ; 結果の上位 16 ビット読み出し
        MOVW    RREGW,AX                ; 結果を作業領域に保存
        MOVW    AX,MDA0L                ; 結果の下位 16 ビット読み出し
;
;   今回の結果を計算結果領域に加算
;
        XCH     A,X                     ; A レジスタに下位をセット
        ADD     A,[HL+C]                ; 新しい結果を累積する
        MOV     [HL+C],A                ; 結果を保存
;
;   計算結果の加算処理
;
LOOP3:
        MOV     A,#0                    ; 上位桁用にクリア
        XCH     A,RREGW+1               ; 最上位桁にシフト
        XCH     A,RREGW                 ; 次の桁にシフト
        XCH     A,X                     ; 次の桁を A レジスタに
        INC     C
        ADDC    A,[HL+C]                ; 上位桁の加算
        MOV     [HL+C],A                ; 結果を保存
        DBNZ    B,$LOOP3                ; 4 バイト分は繰り返す
        INC     B                       ; 次回以降必ず下に抜けるためのダミー
        BC      $LOOP3                  ; 桁上げがあればループ
;
;   1 回分の計算完了
;
        POP     BC                      ; オフセットを復帰
        INCW    HL                      ; 乗数と結果の領域を桁上げ
        INCW    HL                      ; 16 ビット分桁上げ
        DBNZ    CLOOP0,$ LOOP2          ; 乗数を桁上げして、次の桁を計算
;
;   1 桁分の被乗数の計算完了
;
        POP     HL                      ; 乗数、結果のベースを復帰
        INCW    DE                      ; 被乗数の桁上げ
        INC     C                       ; 結果保存用オフセットを桁上げ
        INC     C                       ; 16 ビット分桁上げ
        DBNZ    CLOOP1,$LOOP1           ; 被乗数の次の桁の計算に
;
;   必要な桁の計算完了
;
```

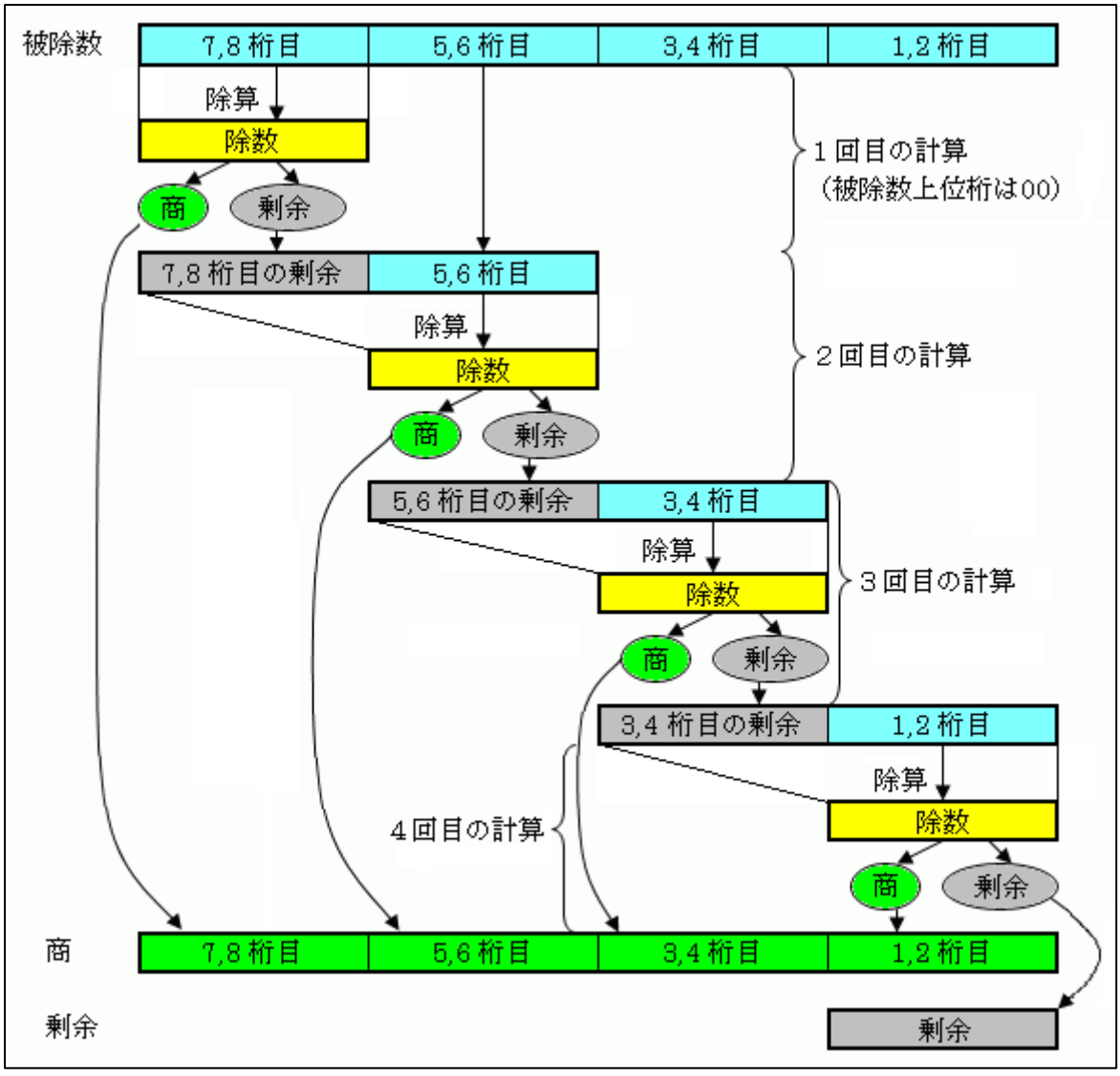
このプログラムでは、コンパクトにするために、乗除算器の演算結果を加算する部分をループで処理しています。そのため、4 バイト分までの加算は「DBNZ B」命令でループしますが、その後は「BC」命令でループします。そこで、一度「DBNZ B」命令を下に抜けると、その後は必ず下に抜けるように「INC B」命令を実行して、B レジスタを 1 にしています。ここを分かりやすく記述するには、「INC B」命令の代わりに、次の赤字の 5 命令が必要です。

```
        DBNZ    B,$LOOP3                ; 4 バイト分は繰り返す
LOOP4:
        BNC     $NEXT
        MOV     A,#0
        INC     C                       ; 次の桁へ
        ADDC    A,[HL+C]                ; 桁上げ部分の計算
        MOV     [HL+C],A                ; 結果の保存
        BR      $LOOP4                  ; ループ
NEXT:
```

(3) 除算の考え方(基礎)

除算の除数は分割できないので、除数に応じた計算方法が必要になります。除数が、使用するハードウエア(除算命令を含む)で対応するビット長を越えると、基本的にそのハードウエアは使用できません。被除数については、桁数が大きくても、ハードウエアで処理できるビット長以下に分割すれば処理できます。ハードウエアで処理できるビット長以下ならいいので、そのビット長に合わせる必要はありません。これは、実際の筆算の処理と同じように、除数の桁に合わせると、処理が簡単です。

桁の管理については、計算結果の商は最大で被除数と同じ桁になるので、被除数の大きい方の桁から埋めていけば良いことになります(結果そのものは、値が 0 になっていることがあります)。8 桁(32 ビット)を 2 桁(8 ビット)で除算する場合の考え方の例を示します。



これをプログラムにすると、次のようになります。まず定数の定義は、32 ビット割る 8 ビットに対応した値です。「DNdivD」の値を変化させれば、被除数の桁数を変更できます。「32/8」の部分を「40/8」にすると 40 ビット割る 8 ビット、「64/8」にすると 64 ビット割る 8 ビットになります。

```
;************************************************
;       定数や変数の定義
;************************************************
;   計算回数の定義（データのバイト長で定義）
DNdivS          EQU     8/8             ; 除数は 8 ビット
DNdivD          EQU     32/8            ; 被除数は 32 ビット
```

これを用いて、作業用のデータ領域を次のように定義します。被除数と商は、同じ領域に配置します。

```
;
;データ領域の定義
;
RREG            DSEG    SADDRP
;
;   データは下位、上位の順（リトル・エンディアン）とする
;
;   データ領域
RREG0:          DS      DNdivD          ; 被除数／商
RREG1:          DS      DNdivS          ; 剰余
RREG2:          DS      DNdivS          ; 除数
```

プログラム本体は、次のような簡単な処理です。LOOP1 以降が、実際の計算をする部分です。その直前で C レジスタを 0 にして、1 回目の計算での被除数の上位を 0 にしています。これにより、ひとつの繰り返しループだけで処理できます（除算が 4 回必要なので、処理時間が長くなりますが、毎回得る商は必ず下位 8 ビットに収まるので、処理が簡単になります）。

```
        CSEG
;************************************************
;   32 ビット÷8 ビット演算部本体
;   入力：被除数（RREG0）、除数（RREG2）
;   出力：商（RREG0）、剰余（RREG1）
;************************************************
D32bit_8bit:
        MOVW    HL,#RREG0+DNdivD-1      ; 被除数のアドレス設定
        MOV     B,#DNdivD               ; 被除数数桁カウンタ
;
;   ポインタの使い分けは以下の通り
;   被乗数／商には HL レジスタ間接（[HL]）でアクセス
;
        MOV     C,#0                    ; 剰余（被除数の上位）をクリア
LOOP1:
;
;   被除数、除数のセット
;
        MOV     A,RREG2                 ; 除数の読み出し
        XCH     A,C                     ; 除数を C レジスタにセット
        MOV     X,A                     ; 前回の剰余を X に退避
        MOV     A,[HL]                  ; 被除数の読み出し
        XCH     A,X                     ; 被除数を AX にセット;
```

```
;
        DIVUW   C                       ; 除算の実行
        MOV     A,X                     ; 商の読み出し
        MOV     [HL],A                  ; 商をセット
        DECW    HL                      ; ポインタを下の桁へ
        DBNZ    B,$LOOP1                ; 必要回数ループ
;
        MOV     A,C                     ; 剰余の読み出し
        MOV     RREG1,A                 ; 剰余のセット
;
;   必要な桁の計算完了
;
```

今度は、同様に乗除算器を用いた例を考えます。32 ビット割る 16 ビットは 1 回の処理で完了してしまうので、上記の除算命令使用時との比較のために参考で示します。

```
        CSEG
;************************************************
;   32 ビット÷8 ビット演算部本体
;   入力：被除数（RREG0）、除数（RREG2）
;   出力：商（RREG0）、剰余（RREG1）
;************************************************
D32bit_8bitB:
;
;   被除数、除数のセット
;
        MOVW    AX,RREG0                ; 被除数の下位データ読み出し
        MOVW    MDAOL,AX                ; 乗除算器へのセット
        MOVW    AX,RREG0+2              ; 被除数の上位データ読み出し
        MOVW    MDAOH,AX                ; 乗除算器へのセット
        MOV     A,RREG2                 ; 除数の読み出し
        MOV     X,A                     ; 下位バイトにセット
        MOV     A,#0                    ; AX レジスタに除数をセット
        MOVW    MDB0,AX                 ; 除数を乗除算器にセット
;
;   計算処理開始
        MOV     DMUC0,#10000000B        ; 除算開始
        BT      DMUE,$$                 ; 演算完了待ち
;   計算処理完了
        MOVW    AX,MDAOL                ; 商の下位読み出し
        MOVW    RREG0,AX                ; 商領域に設定
        MOVW    AX,MDAOH                ; 商の上位読み出し
        MOVW    RREG0+2,AX              ; 商領域に設定
        MOV     A,SDR0L                 ; 剰余の下位を読み出し
        MOV     RREG1,A                 ; 剰余領域に設定
;
;   必要な桁の計算完了
;
```

(4) 除数が 16 ビットのときの処理

この場合、乗除算器で計算することになります。乗除算器を内蔵していないデバイスの場合は、「**(5) 除数が 16 ビットを越える場合の処理**」と同じ処理となります。

このプログラムの基本的な考え方は、「**(3) 除算の考え方(基礎)**」の除算命令を用いた除算と同じです。違いは、1 回に処理するのが 16 ビット単位になることと、除算命令でなく乗除算回路を使用することです。

プログラム例を次に示します。このプログラムでは、定数「**DNdivS**」が変更され、除数や剰余の桁数が 16 ビットになったこと以外は、変数領域での違いはありません。

```
DNdivS          EQU     16/8            ; 除数は 16 ビット
DNdivD          EQU     32/8            ; 被除数は 32 ビット

        CSEG
;************************************************
;  32 ビット÷16 ビット演算部本体
;  入力：被除数（RREG0）、除数（RREG2）
;  出力：商（RREG0）、剰余（RREG1）
;************************************************
D32bit_16bit:
        MOVW    HL,#RREG0+DNdivD-2      ; 商のアドレス設定
        MOV     B,#DNdivD/2             ; 被除数桁カウンタ
;
;  ポインタの使い分けは次のとおり
;  被乗数／商には HL レジスタ間接([HL])でアクセス
;
        MOVW    AX,RREG2                ; 除数の読み出し
        MOVW    MDB0,AX                 ; 除数を乗除算器にセット
        MOVW    DE,#0                   ; 前回の剰余を 0 にしておく
LOOP1:
;
;  被除数のセット
;
        MOV     A,[HL]                  ; 被除数の下位バイト読み出し
        MOV     X,A
        MOV     A,[HL+1]                ; 上位バイト読み出し
        MOVW    MDA0L,AX                ; 乗除算器にセット
        MOVW    AX,DE                   ; 前回剰余を読み出し
        MOVW    MDA0H,AX                ; 上位桁には前回剰余をセット
;
;  計算処理開始
        MOV     DMUC0,#10000000B        ; 除算開始
        BT      DMUE,$$                 ; 演算完了待ち
;  計算処理完了
        MOVW    AX,MDA0L                ; 商の読み出し
        MOV     [HL+1],A                ; 商の上位バイトを格納
        MOV     A,X                     ; 商の下位バイトをセット
        MOV     [HL],A                  ; 商の下位バイト格納
        MOVW    AX,SDR0                 ; 剰余の読み出し
        MOVW    DE,AX                   ; 剰余をセーブ
```

```
        DECW    HL                      ; ポインタを下の桁へ
        DECW    HL
        DBNZ    B,$LOOP1                ; 必要回数ループ
;
        MOVW    RREG1,AX                ; 剰余を格納
;
;   必要な桁の計算完了
;
```

(5) 除数が 16 ビットを越える場合の処理

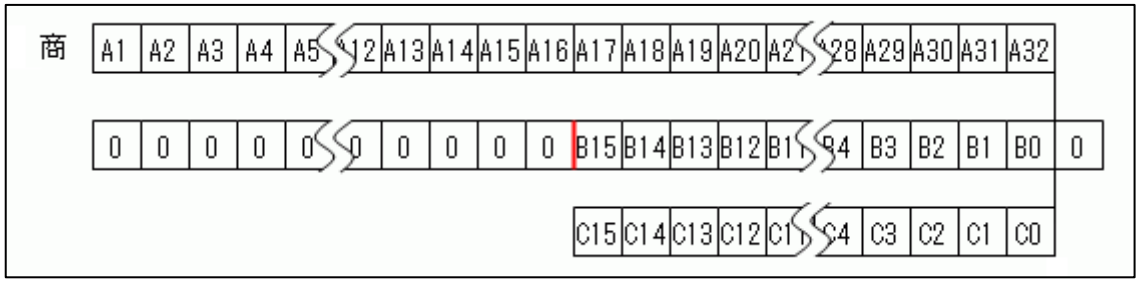
この場合、1 ビットずつずらしながら、被除数から除数を引いていき、引ければ商のそのビットが 1 になります。引けなければ 0 になります。このような計算では、商の桁位置(ビット位置)に注意する必要があります。ビットをずらしながら引いていくので、商 1 回の引き算ごとに 1 ビットずらして、毎回の結果を付け加えていくことになります。たとえば、32 ビットの被除数を 16 ビットで割る場合を考えます。引き算は上位ビットから計算していきますが、除数が最低で 1 の場合もあるので、次のような被除数と除数の関係で、計算を開始する必要があります。この状態で引けた場合は、その結果(1 ビット)に、2 の 31 乗の重み(桁)があります。



次の桁を計算するには、被乗数を左にシフトして、同様に引き算をすればよいことになります。また、同時に前回の商も左シフトすれば、今回の結果との桁の関係が整合します。

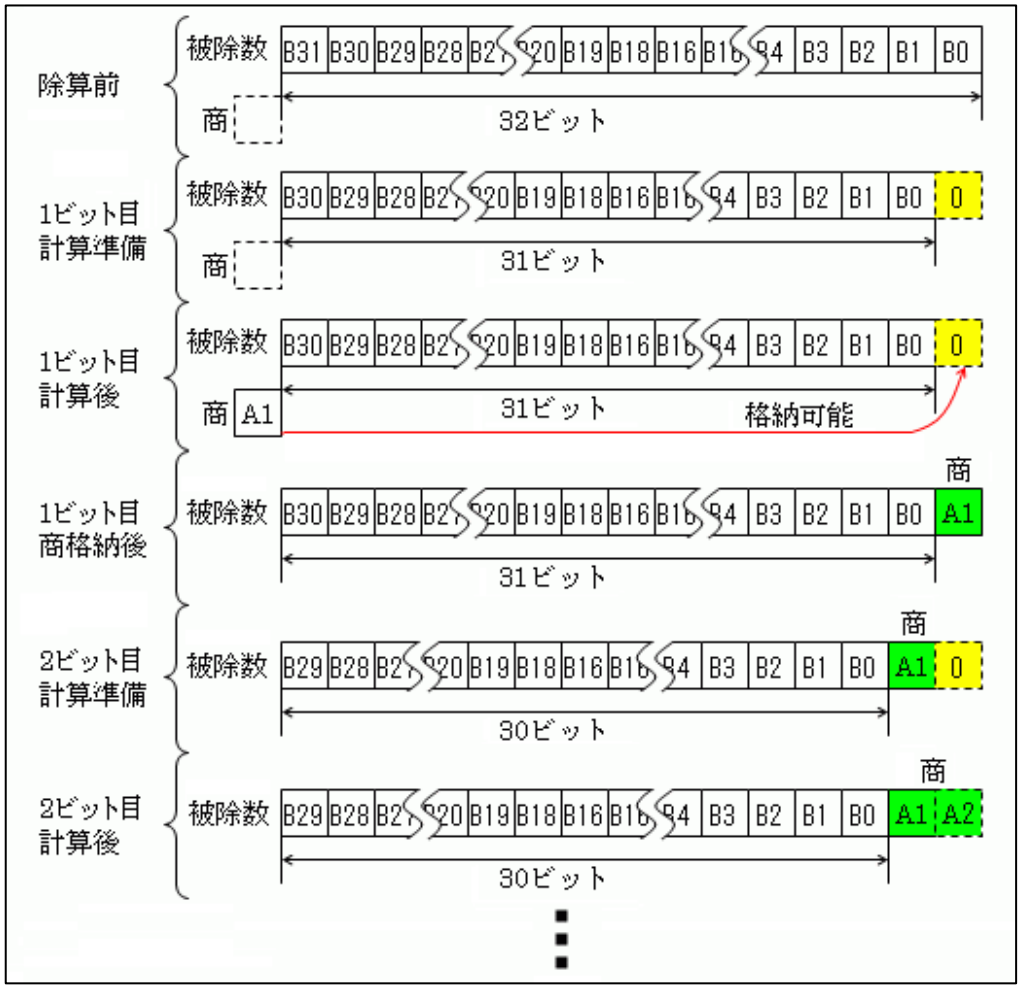


この処理を被除数の桁数分だけ繰り返すと、最終ビットの桁の計算時には、次のようになります。これをまとめると、被除数の上位に除数－1 個の 0 の桁を付加し、被除数を左シフトしながら引き算を行い、商のビットも左シフトし、それを被除数の桁数分だけ繰り返せば除算できます。

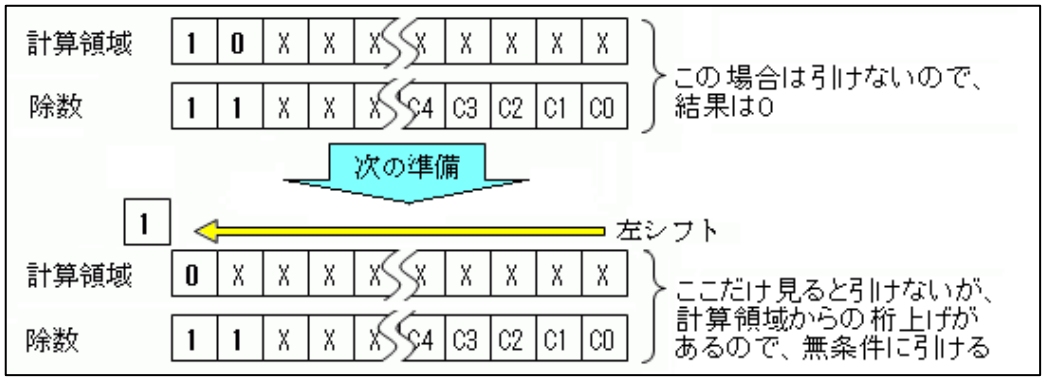


プログラムでは、変数の使い方と並びに注意が必要です。被除数と商、引き算用の作業領域は、1 ビットの計算ごとに左シフトする必要があります。これらのシフトを連続的に処理すれば、効率的なプログラムになります。

また、商の桁数は、最大で被除数の桁数と同じになるので、同じ領域を兼用できれば、必要なメモリ（RAM）領域を少なくできますし、同時に左シフトができます。除算する前は、商は当然なくて、被除数が 31 ビットあるだけです。次図に示すように、計算を進めるに従って、被除数の下位から未使用のビットが追加されていきます。ビットごとの計算完了後に、そのビットの商を未使用のビットに格納すれば、商のための領域を独立して準備する必要はなく、被除数の左シフトで商も同時に左シフトできます。このように、被除数領域と商領域を共用すれば、メモリ領域も演算処理も効率的なプログラムになります。



次に計算領域について考えてみます。計算のための領域として、除数と同じビット長の領域を確保する必要があり、被除数は MSB から順に、計算領域の右からシフトインしてきます。ここで、単純に計算領域から除数を引いていくだけでは問題があります。たとえば、除数の上位 2 ビットが 11 の場合などに注意が必要です。計算領域の上位 2 ビットが 10 の場合、除数を引くことができないので、そのビットの商は 0 になります。次の計算のために、計算領域を左シフトすると、計算領域の MSB の 1 がなくなくなります（実際はキャリー・フラグに入る）。ところが、実際の値は、16 ビットでなく 17 ビットです。除数は 16 ビットなので、その商のビットは必ず 1 になります。



つまり、計算領域からの桁上がりを商の領域（被除数に使用していた領域）の LSB に反映させれば、対応できます。この桁上がりがないときは、当然ながら商は、計算領域からの除数の減算結果で決まります。このような処理をすれば、正しい計算ができます。

実際のプログラムでは、次のようになります。桁数として、32ビットの被除数を16ビットの除数で除算する例です。

```
;************************************************
;       定数や変数の定義
;************************************************
;
;   計算回数の定義（データのバイト長で定義）
DNdivS EQU      16/8            ; 除数は 16 ビット
DNdivD EQU      32/8            ; 被除数は 32 ビット
DNQUO  EQU      DNdivD*8        ; 計算回数
```

これを用いて、作業用のデータ領域を次のように定義します。被除数と剰余は、同じ領域に配置します。

```
;
;データ領域の定義
;
RREG   DSEG     SADDR
;
;   データは下位、上位の順（リトル・エンディアン）とする
;
;   データ領域
RREG0:DS        DNdivS          ; 計算領域（剰余）
RREG1:DS        DNdivD          ; 被除数（商）
RREG2:DS        DNdivS          ; 除数
```

これらの定数や変数を用いた計算処理部分のプログラムは、次のようになります。

```
                CSEG
;************************************************************
;       32bit÷16bit 演算本体部
;       入力：被除数(RREG1)、除数(RREG2)
;       出力：商(RREG1) 、剰余(RREG0)
;************************************************************
D32bit_16bit:
;
; 作業領域のクリア
;
        MOV     A,#0
        MOV     C,#DNdivS               ; 作業領域の大きさ設定
        MOVW    HL,#RREG0-1
;
;   ベースを RREG0-1 にすることで、C の値（DNdivS～1）で[HL+C]が
;   RREG0 の最後（RREG0+DNdivS-1）から RREG0 の先頭までを指せる。
;
CLOOP0:
        MOV     [HL+C],A                ; メモリのクリア
        DBNZ    C,$CLOOP0               ;
        MOV     B,#DNQUO                ; ビット演算回数設定
```

```
JDIV120:
;
;   1 桁(ビット)の計算のための準備
;
;*********************************************
;作業領域に被除数の MSB を 1 ビット・シフト移動
;*********************************************
        CLR1    CY                      ; キャリー情報のクリア
        MOV     C,#DNdivS+DNdivD        ; 演算回数(作業領域＋被除数分)
        MOVW    HL,#RREG0               ;
SLLOOP:
        MOV     A,[HL]                  ; RREG1 からの桁上げ
        ROLC    A,1                     ; 用に LSB は 0 にして
        MOV     [HL],A                  ; 次の桁演算準備で
        INCW    HL                      ; 左シフトしておく。
        DBNZ    C,$SLLOOP               ;
        MOV1    RREG0.0,CY              ; 被除数-->RREG0 の LSB
;
;   RREG0 から RREG1 を引くことで、1 桁分の演算をする
;   ここまでで RREG1 の LSB が 1 なら、被除数は 17 ビットとなり、
;   除数より大きいので、減算結果のボローを確認しなくてよい
;
;*********************************************
;被除数から除数を引く
;*********************************************
        CLR1    CY                      ; 初期値として CY クリア
        MOVW    HL,#RREG2               ; 除数先頭アドレス
        MOVW    DE,#RREG0               ; 被除数先頭アドレス
        MOV     C,#DNdivS               ; 演算回数(除数バイト分)
SUBLOOP:
        MOV     A,[DE]                  ;
        SUBC    A,[HL]                  ; 被除数から除数を減算
        MOV     [DE],A                  ; 演算結果を戻す
        INCW    DE                      ; 被除数次アドレス
        INCW    HL                      ; 除数次アドレス
        DBNZ    C,$SUBLOOP              ; 演算回数分繰り返す
        BT      RREG1.0,$JDIV220        ; 被除数が 17 ビットなら次へ
        BNC     $JDIV210                ; 引ければ分岐
;*********************************************
;被除数を元に戻す
;*********************************************
JDIV180:
        CLR1    CY                      ; キャリー情報のクリア
        MOVW    HL,#RREG2               ; 除数先頭アドレス
        MOVW    DE,#RREG0               ; 被除数先頭アドレス
        MOV     C,#DNdivS               ; 演算回数(除数バイト分)
ADJLOOP:
        MOV     A,[DE]                  ;
        ADDC    A,[HL]                  ; 被除数に除数を加算
        MOV     [DE],A                  ; 演算を元に戻す
        INCW    DE                      ; 除数次アドレス
        INCW    HL                      ; 被除数次アドレス
        DBNZ    C,$ADJLOOP              ; 演算回数分繰り返す
```

```
JDIV210:
        NOT1    CY                      ; キャリーを反転
        MOV1    RREG1.0,CY              ; 商のビットに反映
JDIV220:
        DBNZ    B,$JDIV120              ; 演算を DNCNT 桁分完了するまで
                                        ; 繰り返す
```

このプログラムは、次の定数の定義部分で、赤字の値を変えれば、除数のビット長を 8 ビット単位で変更できます。また、青字の値を変えれば、被除数のビット長も 8 ビット単位で変更できます。

```
DNdivSEQU       16/8            ; 除数は 16 ビット
DNdivDEQU       32/8            ; 被除数は 32 ビット
```

(6) 除数が 16 ビットを越える処理プログラムの考察

以上のプログラムは約 70 バイトで、かなりサイズを意識しています。このような計算プログラムでは、サイズ以外に処理時間も問題になります。このプログラムでは、除数が 1 で被除数が全ビット 1 のときに、最も処理時間が短くなります(64 ビット÷32 ビットで 26,318 クロック)。「除数＞被除数」の場合は、引きすぎた分を毎回戻す処理が発生するので、処理時間が一番長く(64 ビット÷32 ビットで 34,126 クロック)なり、約 7,800 クロックの差があります。通常のリアルタイム性が要求される組み込み制御では、この最悪値を処理時間とする必要があります。

これに対し、次に示すプログラムのように、引き過ぎ分を戻す必要がないように変更すると、同じ条件の計算で 28,372～28,474 クロックと、最短時間は約 2,000 クロック長くなり、プログラムも 16 バイト長くなります。しかし、最悪の計算時間は約 5,600 クロックも短くなります。

除数桁数を 16 ビット～32 ビットと考え、0/1 の確率が 50%とします。64 回の計算中で最初の 8 回は結果が 0 で残り、56 回のうちの 50%で結果が 0 になるとすると、36 回は結果が 0 となります。前述のプログラムの平均計算時間の予測値(約 31,000 クロック)に対して、以下のプログラムは 28,400 クロックで、1 割以上高速に処理できることになります。つまり、リアルタイム性があまり必要なくても、以下のプログラムの方が高速に処理できます。どちらが望ましいかは、プログラムに対する要求項目の優先度に依存します。

```
;*********************************************************************
;*      64bit÷32bit サブルーチン                                     *
;*      入力：被除数(RREG1)、除数(RREG0)                             *
;*      出力：商(RREG1)、剰余(RREG2)                                 *
;*                                                                   *
;*  RREG1 の上位に RREG2 を加えて、96 ビットにしてから除数を         *
;*  引く。実際の減算は、RREG2 から RREG0 を引く。                    *
;*  引ければ、その回(ビット)の答は 1 となり、RREG1 の LSB に         *
;*  結果が MSB から順に得られる。                                    *
;*  さらに RREG1 と RREG2 を合わせて左シフトして、被除数と答         *
;*  のビットを 2 倍し、次の桁(ビット)の計算に移る。                  *
;*      +←(RREG2/RREG3)←(RREG1)←+   ・・・左シフト                  *
;*      |                        |                                   *
;*      +------(結果のビット)----→+                                  *
;*  なお、減算結果の RREG2 の MSB が 1 の場合は、次の桁の計算       *
;*  結果が無条件に 1 となるので、シフトした結果の RREG1 の LSB       *
;*  の値"1"が、そのままその桁の結果となる。                          *
;*                                                                   *
;*********************************************************************
;*******************************************************
;       定数や変数の定義
;*******************************************************
```

```
;
;計算回数の定義（データのバイト長で定義）
;
DNdivSEQU     (32/8)                  ; 除数(32 ビット)
DNdivDEQU     (64/8)                  ; 被除数(64 ビット)
DNQUO EQU     DNdivD*8                ; 演算回数
;
;データ領域の定義
;
DREG  DSEG    SADDRP
RREG0:DS      DNdivS                  ; 除数領域
RREG1:DS      DNdivD                  ; 被除数領域、商領域
RREG2:DS      DNdivS                  ; 作業領域、剰余領域
RREG3:DS      DNdivS                  ; 減算領域
RCOUNT:       DS      1               ; 演算回数カウンタ
Offset0       EQU     RREG3-RREG0     ; 減算結果用オフセット
PUBLICD64bit_32bit
PUBLICRREG0
PUBLICRREG1
PUBLICRREG2

      CSEG
;*********************************************************
;      64bit÷32bit 演算本体部
;      入力：被除数(RREG1)、除数(RREG0)
;      出力：商(RREG1)、剰余(RREG2)
;*********************************************************
D64bit_32bit:
;
; 作業領域のクリア
;                                     ↓:クロック数
      MOV     A,#0                    ;4:
      MOV     B,#DNdivS               ;4:作業領域の大きさ設定
      MOVW    HL,#RREG2-1             ;6:
;                                      (20)
;
; ベースを RREG2-1 にすることで、C の値（DNdivS～1）で[HL+B]が
;RREG2 の最後（RREG2+DNdivS-1）から RREG2 の先頭までを指せる。
;
CLOOP0:
      MOV     [HL+B],A                ;6:メモリのクリア
      DBNZ    B,$CLOOP0               ;6:
                                      ;(12*4=48)
      MOV     RCOUNT,#DNQUO           ;6:bit 演算回数設定
      MOV     B,#DNdivS               ;4:
;                                     (10)
JDIV120:
;
;  1 桁(ビット)の計算のための準備
;
;*****************************************************
;作業領域に被除数の MSB を 1 ビット・シフト移動(270)
;*****************************************************
```

```
        CLR1    CY                      ;2:キャリー情報のクリア
        MOV     C,#DNdivD               ;4:被除数桁分カウンタ設定
        MOVW    HL,#RREG1               ;6:被除数領域ポインタ設定
SLLOOP:
        MOV     A,[HL]                  ;4:作業領域からの桁上げ用に
        ROLC    A,1                     ;2:回転して LSB は 0 に
        MOV     [HL],A                  ;4:次の桁演算準備で被除数
        INCW    HL                      ;4:領域を左シフトしておく。
        DBNZ    C,$SLLOOP               ;6:被除数のバイト数分ループ
                                        ;(12)+(20*8=160)
        MOV     C,#DNdivS               ;4:作業領域分カウンタ設定
SLLOOP2:
        MOV     A,[HL+B]                ;6:減算／作業領域のデータを
        ROLC    A,1                     ;2:次の桁演算準備で左シフト
        MOV     [HL],A                  ;4:して、作業領域にセット。
        INCW    HL                      ;4:
        DBNZ    C,$SLLOOP2              ;6:
;                                       (4)+(22*4=88)
        MOV1    RREG1.0,CY              ;6:被除数-->RREG1 の LSB
;
;  RREG2 から RREG0 を引くことで、1 桁分の演算をする
;  ここまでで RREG1 の LSB が 1 なら、被除数は 17 ビットとなり、
;  除数より大きいので、減算結果のボローを確認しなくてよい
;
;****************************************
;被除数から除数を引く
;****************************************
        CLR1    CY                      ;2:初期値として CY クリア
        MOVW    HL,#RREG0               ;6:除数先頭アドレス
        MOVW    DE,#RREG2               ;6:作業領域先頭アドレス
        MOVW    BC,#DNdivS*100H+Offset0 ;6:B:演算回数(除数バイト分)
                                        ; C:減算結果用オフセット
;                                       (20)
SUBLOOP:
        MOV     A,[DE]                  ;4:
        SUBC    A,[HL]                  ;4:被除数から除数を減算
        MOV     [HL+C],A                ;6:演算結果を減算領域に
        INCW    DE                      ;4:被除数次アドレス
        INCW    HL                      ;4:除数次アドレス
        DBNZ    B,$SUBLOOP              ;6:演算回数分繰り返す
;                                       (28*4=112)
        MOV     B,#DNdivS               ;4:次回オフセット設定
        BT      RREG1.0,$JDIV220        ;10:被除数が 17bit なら次へ
        NOT1    CY                      ;2:キャリーを反転
        MOV1    RREG1.0,CY              ;6:商のビットに反映
        BC      $JDIV220                ;6:引けたら分岐する
        MOV     B,#0                    ;4:前回結果は使わない
;                                       (32)
JDIV220:
        DBNZ    RCOUNT,$JDIV120         ;8:演算を DNQU0 桁分完了するまで
                                        ;  繰り返す
```

```
;****************************************
;   最後の減算結果を元に戻す（引けた場合）
;****************************************
        BF      RREG1.0,$JDIV310        ;10:引けなかったら終了
        MOVW    HL,#RREG2               ;6:剰余先頭アドレス
        MOVW    BC,#DNdivS*101H         ;6:演算回数(除数バイト分)
ADJLOOP:
        MOV     A,[HL+B]                ;6:減算結果読み出し
        MOV     [HL],A                  ;4:剰余領域にコピー
        INCW    HL                      ;4:結果を次アドレスに
        DBNZ    C,$ADJLOOP              ;6:剰余桁回数分繰り返す
;                                       (22)+(20*4=80)
JDIV310:
        RET                             ;6:
```

1.2.32 ブート・スワップ機能

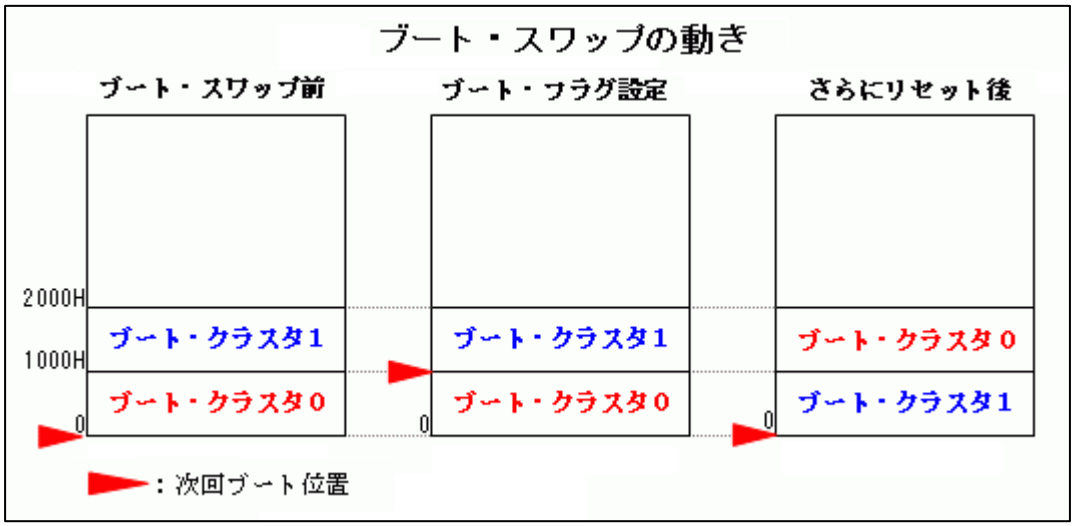
ブート・スワップ機能 [78K0/Kx2]

Q1. ブート・スワップ機能は、どのような機能で、どのようなときに使うのですか。

A1. 78K0 では、リセットから起動すると、内蔵フラッシュ・メモリの 0 番地から始まるメモリ領域（ブート領域）から、実行するプログラムのアドレス（ベクタ）を読み出して、そのアドレスから命令の実行を開始（ブート）します。通常は、この領域のフラッシュメモリの内容をライタで書き込んだら、あまり内容を書き換えないはずです。しかし、プログラムで内蔵フラッシュ・メモリに書き込む（セルフ・プログラミング）際に、このブート領域を含めて書き換えをしたい場合があります。そのときはいったん、対象ブロックの内容を消去してから、新しいプログラムを書き込む必要があります。ところが、消去後でプログラムを書き込む前に、電源の瞬断などでマイコンにリセットがかかってしまうと、ブート領域の内容が消去されているので、正常に起動できません（リセット・ベクタは FFFFH となっており、FFFFH 番地は内蔵周辺機能(SFR)領域なので、ここからは命令フェッチできず、不定なコードを命令として読み込み、以降の動作は不定となります。つまり、リセットによる起動ができなくなります。この問題に対応したのが、ブート・スワップ機能です。

ブート・スワップ機能は、物理的なメモリ領域を入れ替える機能です。物理的なメモリ領域としてブート・クラスタ 0 とブート・クラスタ 1 があり、内蔵フラッシュ・メモリが消去された状態では、ブート・クラスタ 0 がフラッシュ・メモリの先頭 4KB(0000-0FFFH:ブロック 0～3)に割り当てられ、ブート・クラスタ 1 が続く 4KB(1000H-1FFFH:ブロック 4～7)に割り当てられています。

ここでブート・フラグを設定すると、次にリセットがかかったときに、ブート領域として使用される領域は**ブート・クラスタ 1** となります（図中の**赤い三角**）。この段階（ブート・フラグを設定し、リセットが発生する前）では、メモリ・マップはまだ変わりません（図の**ブート・フラグ設定**）。次に、ブート・フラグが設定された状態でリセットがかかると、0～0FFFH の領域に**ブート・クラスタ 1** が、1000H～1FFFH の領域に**ブート・クラスタ 0** が配置されます（図の**さらにブート後**）。つまり、CPU からみた 0 番地は**ブート・クラスタ 1** となり、リセット解除後には**ブート・クラスタ 1** から起動することになります（CPU としては 0 番地から起動することに変わりはありません）。**ブート・クラスタ 1** にブートに必要なプログラムをコピーしておけば、**ブート・クラスタ 0** が消去された状態でも正常に起動できます。このように、**リセットで起動するクラスタを切り替えるのがブート・スワップ機能です**。



このように、**0～0FFFH 番地からのブート領域をセルフ・プログラミングにより書き換える場合**にだけ使用する機能です。したがって、ブート領域を書き換えない場合には、使用する必要はありません。

また、単にブート・フラグを操作しても、次のブート位置が変わるだけで、メモリの実際の配置が切り替わるわけではありません。リセットがかかって初めてメモリの配置が切り替わるので、ライタで書き換えを行う場合には、次の点に注意が必要です。

ブート・フラグがセットされた状態のマイコンをライタで消去すると、ブート・フラグはクリアされてしまいます。ところが、この状態ではメモリの配置は変わらないので、そのまま書き込みを行うと、0 番地の内容が**ブート・クラスタ 1** に書かれることになります。その後、動作させようとすると、ブート・フラグはクリアされているので、**ブート・クラスタ 0** が使用され、0 番地のつもりで書き込んだ**ブート・クラスタ 1** は 1000H 番地となってしまいます。

このようなトラブルを避けるには、フラッシュを消去したあといったんリセットし、メモリの配置をブート・フラグと整合させてから書き込みます（具体的には、EPV コマンドではなく、単独でのイレーズ・コマンドを実行し、そのあとで書き込みを行います）。

または、ブート・スワップ機能を使用するのであれば、最初から 2 つのブート・クラスタに同じ内容を書き込んでおけば、どちらでもブート可能になり、1 回の EPV コマンドで書き込みが可能です。

1.3 V850 デバイス

1.3.1 A/D、D/A 変換

A/D 変換値が正しくない

Q1. ADM1 レジスタの ADCS を設定(A/D 変換開始)すると、1回目だけ変換値が正しくないことがあります。

A1. コンパレータが、A/D 変換停止時に停止しており、変換開始時にまだ起動していないと思われます。ADPS＝0 (コンパレータ・オン)に設定して安定時間を待つか、ADPS＝1(安定時間込みの変換)に設定してください。なお、ADCS＝1(変換動作許可)のまま、チャネル切り替えをして変換を継続した場合は、すでにコンパレータが起動しているので、必ず正常値が得られます。

<参考>

この注意事項は、初期の V850/S と、一部の 78K0、78K0S に存在するものです。

V850E の A/D コンバータには該当しません。

V850ES/Kx1 のような最近のデバイスや、V850/SA1、V850/SB1 でも、最新バージョンではこのような制御ビットを設置しています。

一定時間以上前に、そのビットをセットしておけば、A/D 変換動作をスタートした直後の最初の A/D 変換値の精度も規格を満たせるようになっています。

A/D 変換結果の格納

Q1. V850 の A/D 変換結果は、ADCR にどのように入りますか。

A1. 変換結果の 10 ビットは、16 ビット読み出しのビット 15 から配置されます。下位の 6 ビットには、0 が入ります。したがって、読み出した 16 ビットの変換結果は、次のようになります。

レジスタ・ビット位置	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
変換結果データ	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	0	0	0	0	0	0

変換結果のふらつき(隣接チャネルの値になる) [V850/SF1、V850/SA、V850/SB、V850/SC]

Q1. V850/SF1 で、5 チャネル分のアナログ入力を連続で A/D 変換する動作を一定周期で繰り返しています。ときどき、変換結果が正しくない値(隣接チャネル値)になってしまいます。

A1. 次の 2 点が考えられます。

(1) ソフトウエア上の問題

5chの変換終了で、毎回 A/D を停止していませんか。A/D 変換の最初(ADCS ビットを 0→1 にした直後)の結果は、精度の保証ができません。A/D 変換開始後の最初のデータを無視するか、A/D を停止させないでください。

(2) ハードウエア上の問題(V850 全般に該当)

アナログ入力をドライブする回路のインピーダンスが高くないですか。この問題の可能性が高いです。

この A/D 変換回路は、サンプリング・コンデンサでアナログ入力をサンプリングして、その電圧と基準電圧を逐次比較する動作を行っています。

したがって、アナログ入力端子をドライブするドライブ・インピーダンスが高いと、サンプリング・コンデンサを十分充電することができないので、入力電圧を正確にサンプリングできない(前回変換時のサンプリング値が残ってしまう)可能性があります。
この問題を解決するには、数 kΩ 以下の低いインピーダンスで、アナログ入力端子をドライブしてください。
それができない場合には、アナログ入力端子にコンデンサを挿入してみてください。
この際、ユーザーズ・マニュアルのノイズ対策に記載された回路を参照して、コンデンサの電荷を逃がすためのダイオードも入れてください。

A/D の CE ビットのクリア [V853、V850E/MS、V850E/MA など]

Q1. A/D コンバータ・モード・レジスタの CE ビットは、A/D 変換が終了すると自動的にクリアされますか。

A1. いいえ、A/D 変換が終了しても、**ADM0** レジスタの **CE** ビットはクリアされません。
A/D トリガ・モードでは、CE ビットへの 1 の書き込み自体が、A/D 変換の起動トリガとして使用されますので、(A/D トリガ・モードで動作させる場合には) **CE** ビットが 1 でも 0 でも関係ありません。
CE は、セットしたらそのまででけっこうです。
なお、お問い合わせの動作に近い動きをするのが、**ADM0** レジスタの **CS** ビットです。
このビットは、実際に A/D 変換動作が開始すると 1 になり、変換終了で 0 となります。
タイマ・トリガなどのように、繰り返し A/D 変換動作が行われる場合、**CS** ビットは変換中には 1 になり、変換動作を行っていないときには 0 になります。

A/D 変換を開始するまでの時間 [V853]

Q1. ADM0 レジスタに **CE**＝1(A/D スタート)をセットしてから、**CS**(A/D ステータス)がセットされるまでの時間を教えてください。

A1. **CE** ビットをセットしてから、**CS** ビットがセットされるまでは、CPU クロックで 3 クロック必要です。

1.3.2 クロック

サブクロックから通常クロックへの切り替え時間 [V850]

Q1. V850/SA1 で、サブクロック動作モードから、通常動作への移行にかかる時間を教えてください。

A1. この時間は、大半がメイン・クロックの発振安定時間で、デバイスの特性より、発振子の特性で決まります。
デバイスがクロックの切換えに必要な時間は、最大 2 命令分必要です(ユーザーズ・マニュアルの「6.3.1 制御レジスタ」の項をご参照ください)。
通常、サブクロック動作時には、メイン・クロックの発振は停止していると思います。
その場合には、この切り換えに必要な時間は、通常のクロック発振の安定時間より短いので、上記のように発振安定時間で決まるとお考えください。

50MHz のクロック動作は可能か [V850E/MA1]

Q1. V850/MA1 のクロック発生機能について、ユーザーズ・マニュアルに「5×fxx、2.5×fxx、1×fxx のいずれかで使用する場合は、4～6.6MHz の周波数を使用することができます」とありますが、1×fxx だけ使用する場合には、50MHz のクロックを入力できますか。

A1. いいえ、そのような使い方はできません。PLL モードで X1 に入力できるのは 6.6MHz までです。
50MHz のクロックを入力することはできません。

セラロック(発振子)を使えるのは PLL モードだけか [V850E/MS1]

Q1. V850E/MS1 ユーザーズ・マニュアルに「ダイレクト・モードで使用する場合、外部発振子は接続しないでください」とありますが、セラロックを使用できるのは PLL モードだけということですか。

A1. はい、セラロック(発振子)を使えるのは PLL モードだけです。
ダイレクト・モードの場合には、外部の発振器で発生させたクロック信号を入力してください。

ROM レス・モード 0 では、リセット期間中でも CLKOUT 出力が出るか

Q1. V850E/MA1 ユーザーズ・マニュアルの「2.3 端子機能の説明」にある CLKOUT の説明で、「シングルチップ・モード 0 のときは、リセット期間中はポート・モードとなっているため、CLKOUT 端子からの出力は行われません」と記述されていますが、ROM レス・モード 0 では、リセット期間中でも CLKOUT 端子から 50MHz クロックが出力されるのですか。

A1. はい、ROM レス・モード 0 では、リセット期間中でも CLKOUT 端子からクロックが出力されます。
CLKOUT 端子の各種状態での動作については、ユーザーズ・マニュアルの「2.2 端子状態」の表をご参照ください。

外部クロックと動作クロックの関係 [V853]

Q1. V853 で、ベース・クロックに外部クロック 6.78MHz を使用したいのですが、ダイレクト・モードで使用の場合、CKC レジスタが(0,0)だけ設定可能なので、システム・クロックは 1/2 の 3.39MHz になるのですか。

A1. はい、ダイレクト・モードの場合の動作クロック周波数は、入力クロック周波数の 1/2 となります。
したがって、外部クロックとして 6.78MHz を入力した場合には、3.39MHz での動作となります。

PLL モードで X1 に 5MHz を入力していいか [V853]

Q1. μPD70F3003A の PLL モードで、X1 に 5MHz を入力してもいいですか。

A1. はい、PLL モードで 5MHz を入力しても問題ありません。動作周波数は 5～33MHz(A/D 使用時)です。
ただし、CKC レジスタで内部動作クロック周波数をクロック入力周波数の 1/2 に設定した場合、A/D コンバータは使用できません。

外部からの 6.78MHz クロックで動作するか [V853]

Q1. ベース・クロックとして、外部発振器からの 6.78MHz は使用できますか。

A1. これは、デバイスと動作モードによって異なります。

μPD703003、70F3003 の場合には、ダイレクト・モードで A/D コンバータを使用しないなら、動作可能です（PLL モードでは、起動時に 33MHz の最高動作周波数を越えてしまうので、使用できません）。

その他の V853(品名のうしろに A が付く製品)の場合、PLL モードで MODE＝0 に設定したときか、ダイレクト・モードで MODE＝1 に設定したときだけ、動作可能です。なお、PLL モードで MODE＝0 に設定した場合は、その後の設定で CKC レジスタを 02(fxx で動作)に設定したときだけ、A/D コンバータが使用可能です。

BUSCLK と CLKOUT の使い分け [V850/MA1]

Q1. V850E/MA1 のバス制御機能について、クロック同期型でバス制御を行う場合、BUSCLK と CLKOUT の使い分けについて教えてください。

A1. CLKOUT を出力に設定すると、常にクロックを出力します。BUSCLK は、バス・アクセスが発生したときだけ出力され、通常はロウ・レベルです。一般には、CLKOUT をクロック出力にご使用になることをお勧めします。

Q2. パワー・セーブ・モードではバス駆動をしないので、BUSCLK は出力されませんが、CLKOUT はどうですか。

A2. CLKOUT 信号は、内蔵周辺への供給クロックです。このため、HALT モードではクロック出力となりますが、IDLE モードと STOP モードではロウ・レベル出力となります。

クロックのジッタやデューティの許容値 [V850E/MA1]

Q1. EMI 対策で、周波数拡散デバイスからのクロックを入力したいのですが、外部より 4.8MHz（周波数変動約 6%）のクロックを入力して、48MHz で動作させることはできますか。

A1. 4.8MHz で約 6%の変動は、X1 の入力規格範囲内で問題ありません。入力クロックは、周期、ロウ・レベル幅、ハイ・レベル幅、立ち上がり時間、立ち下り時間が規定されています。これらを満足すれば問題ありません。

Q2. X1 に 4.88MHz のクロックを入力して、PLL モードで 10 逓倍していますが、クロック・デューティは 4/9 や 5/9 になります。50%でなくてもいいですか。

A2. デバイスとしては、デューティではなく、ロウ・レベル幅とハイ・レベル幅、立ち上がり時間や立ち下がり時間で規定しています。ロウ・レベル幅とハイ・レベル幅は 50ns 以上になりますので、この点では問題ありません。

消費電力低減のためにクロックを動的に変更 [V850E/MA1]

Q1. V850E/MA1 を 50MHz で使用していますが、高速処理が必要ないとき、PLL の倍率を変更して、動作クロック周波数を 5MHz にすることを考えています。CKC レジスタを動的に切り替えていいですか。

A1. はい、CKC レジスタを動的に切り替えて問題ありません。なお、目的の設定にする前に、いったん fxx 動作に切り替える必要があります。この場合は、fxx への変更なので、この点も問題はありません。

Q2. [V850E 共通]

そのほかに、クロック切り替え時に注意すべき点はありますか。

A2. CKC レジスタは特定レジスタですので、ユーザーズ・マニュアルの「9.3.4 クロック・コントロール・レジスタ」に記載されいる、次の設定手順で変更してください。

(1) 割り込みの禁止（NP ビットをセットして NMI も禁止）する。

(2) 汎用レジスタに CKC レジスタへセットする値をセットする。

(3) PHCMD に(2)のレジスタ内容を転送する。

(4) CKC レジスタに(2)のレジスタ内容を転送する。

(5) NOP 命令を挿入する。

(6) 割り込み禁止を解除する。

Q3. [V850]

リセット時やソフトウエア・ストップ・モードからの復帰時に、PLL がロックされるまで待つ必要がありますが、PLL の倍率を変更しただけでも、待つ必要はありますか。

A3. その必要はありません(クロックを切り替えても、PLL はアンロックしません)。

Q4. [V850/SB]

CPU の内部クロックをディフォールトの fxx/8 から fxx に変更する手順は、どうすればいいですか。

A4. CPU のクロックを制御する PCC レジスタは、**特定レジスタ**ですので、ユーザーズ・マニュアルの「3.4.9 特定レジスタ」に記載された手順で PCC レジスタの値を変更してください。

サブシステム・クロックへの切り替え [V850/S]

Q1. 時計機能をサブクロックで動作させ、CPU をメイン・クロック動作からサブクロック動作に切り替えます。

マニュアルに、「サブクロック動作するまでには(設定前の CPU クロック周波数／サブクロック周波数)×2 の命令数がかかる」とありますが、メイン・クロック動作中にサブクロックが発振している場合も、ウエイト時間は必要ですか。

A1. はい、必要です。この時間は、サブクロックの発振を待っているのではなく、切り替える2つのクロックの間の位相が非同期なので、これを調停するための時間です。

Q2. この時間をソフトウエアでウエイトさる場合、早期にクロックが切り替わると、サブクロックで動作する時間が長くなり、その間はメイン・クロックが発振を続けるので、電力がむだになります。

このウエイト時間の前に、メイン・クロックを停止させることはできますか。

A2. 無条件に、ウエイト時間より前にメイン・クロックを停止することはできません。基本的には、サブクロックで 2 クロック分だけ待てばよいわけで、次のようなハードウエアでのウエイトが考えられます。

(1) サブクロックで動作するタイマで、2 クロック分のカウントを行う。

(2) そのためのタイマが空いていなければ、時計用タイマのインターバル・タイマ機能を使用する。

PLL モードでのクロックの設定 [V850ES/SG2、V850ES/SJ2]

Q1. V850ES/SG2 を PLL モードに設定し、20MHz の CPU クロックで動作させたいのですが、外部に接続するクロックは何 MHz で、どのレジスタをどのように設定すればいいですか。

A1. PLL の倍率により、外部に接続するクロック周波数は異なります。PLL を 4 逓倍で使用する場合は 5MHz のクロックとなり、8 逓倍で使用する場合は 2.5MHz となります。レジスタの操作手順は次のとおりです。(3)で操作している **CKC** レジスタは**特定レジスタ**ですので、この手順で実行してください。

(1) **PLLCTL** レジスタの **SELPLL** ビットを 0(クロック・スルー・モード)に設定します(初期値は 0)。

(2) **PLLCTL** レジスタの **PLLON** ビットを 0(停止)に設定します。

(3) **CKC** レジスタの **CKDIV0** ビットで逓倍率を設定します(4 逓倍なら 0x0A、8 逓倍なら 0x0B)。

(4) **PLLCTL** レジスタの **PLLON** ビットを 1(PLL 動作)に設定します。

(5) **LOCKR** レジスタの **LOCK** ビットが 0(ロック状態)になるのを待ちます。

(6) **PLLCTL** レジスタの **SELPLL** ビットを 1(PLL モード)に設定します。

リセットとクロック発振の関係 [V850E]

Q1. 電源電圧低下時に STOP モード(RAM 保持)にして、電力消費を抑えようとしましたが、電圧低下でリセットがかかると、クロックが発振を再開してしまいます。

A1. V850E では、立ち上げ時間を最適に設定できるように、電源立ち上げ時のメイン・クロックの発振安定時間をリセットで確保するようになっています。つまり、リセットでメイン・クロックを発振させるので、ご指摘の動作となります。電源電圧を低下させた状態で RAM 保持をするには、リセットがかからないようにしてください。

備考 V850/S や V850ES コアの製品では、リセッでメイン・クロックの発振を停止する構成となっています。

サブクロックでの STOP モード [共通]

Q1. サブクロックで動作させている状態では、STOP モードに入れないとありますが、どうすればいいですか。

A1. サブクロックで動作している場合は、**PCC** レジスタの **MCK** ビットの設定で、メイン・クロックの発振を停止してください。その状態で、HALT モードに入ってください(サブクロックの発振は停止できません)。

1.3.3 シリアル通信

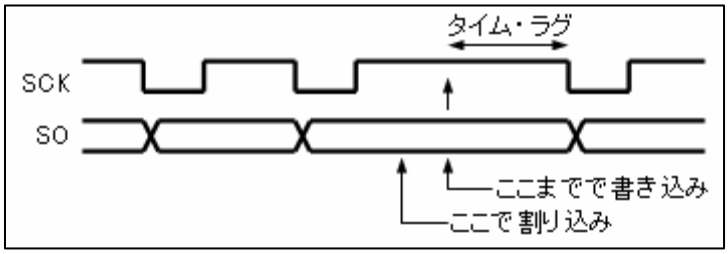
16 ビット長で同期式のシリアル通信

Q1. V850 から、16 ビット長で同期式のシリアル信号を出すことはできますか。

A1. バイト間の期間が、どの程度まで許されるかによります。一般の V850 シリーズでは、8 ビットの通信を 2 回繰り返すことになります。このときの 1 バイト目と 2 バイト目の間隔で、CPU の処理が間に合えば使えます。CPU 処理がむりな場合、V850/SB1 には、ハードウエアで 16 ビット長までの同期通信ができるシリアル通信機能が内蔵されていますので、こちらをご使用ください。

Q2. 1 バイト目と 2 バイト目のタイム・ラグは、どのくらいになりますか。

A2. 内部クロックで動作させる場合、2 バイト目の書き込みから最低 1.5 クロック後に、次の送信を開始します。これは、最低でも 1 ビット分の間が空くことを意味します。



最終ビットでのクロックの立ち上がりで、2 バイト目のデータの書き込みがトリガされますが、書き込みが次のクロックの立ち下がりタイミング(上記 SCK の ↑ 部、実際はクロックは立ち下がらない)より遅れると、その分は追加されます。スレーブ(外部クロック)の場合は、8 回目の SCK の立ち上がりでの通信完了割り込みから、SCK がハイ・レベルの期間に、次の通信の準備を完了しておく必要があります。

Q3. DMA を使用して送信した場合と、DMA を使用せず SIO だけで 2 バイトの送信(1 バイト送信後、SIO の完了時点で 2 バイト目を書き込む)した場合と、どちらがタイム・ラグが少ないですか。

A3. DMA の方が、書き込みの遅れは少ないです。前記 A2 の書き込みタイミングの問題ですが、そこまでに書き込みが完了していれば同じです。これらは、CPU のクロック周波数、伝送周波数、プログラムで変わります。

シリアル・クロックに TO2/TO3 を選択すると、TO2/TO3 をタイマ出力に設定すべきか

Q1. シリアル・インタフェース 0-3 では、クロック・ソースに TO2/TO3 を選択できますが、これを使用する場合は、TO2/TO3 を必ずタイマ出力に設定しなければならないのですか。

A1. いいえ、タイマ出力に設定する必要はありません。

TO2/TO3 をクロックとした場合、TO2/TO3 の出力は通常のポートとして使用可能か

Q1. シリアル・インタフェースで TO2/TO3 をクロックとして使い、その出力端子を通常のポートとして使えますか。

A1. はい、タイマ 2、3 をシリアル・クロックに使用しても、タイマ出力をしなければ、端子をポートに使用できます。

データ出力後の通信割り込みが発生しない

Q1. SIO2 を使用してデータを出力すると、送信完了割り込みが発生しません。

A1. 以下の点をご確認ください。

(1) CPU が割り込み禁止なっていませんか。

問題が割り込みかシリアルかを判別するために、CSIC2 のビット 7(CSIF2)の状態をご確認ください。これがセットされていればシリアルは動作しており、セットされていなければシリアルの動作に問題があります。

(2) CSI2 のクロックは設定されていますか。

送信では CSIM1 の設定をコピーして、クロック選択部分だけは流用されます。

(3) シリアルの設定と起動条件の関係は問題ないですか。

CSI2 の送受信モードなら、転送起動条件は SIO2 へのダミー・データ書き込みです(SIO2 読み出しでない)。

2CPU 間で UART 通信を行う場合の電源投入時の注意事項

Q1. UART の通信で、電源投入時の RXD, TXD の状態が不安定となり、通信に影響を与えませんか。

A1. 通信前に、シリアルの信号線がハイ・レベルを保持できれば問題ありません。もしロウ・レベルになると、その幅によってはスタート・ビットと認識され、誤動作の原因になりえます。通常、これらのシリアル信号は、シリアル設定以前にポートとして動作するようになっています。このときに、ロウ・レベルとならないようにしてください。たとえば V850/SA1 では、**P14** や **P24** が TxD 信号と兼用になっています。シリアルの初期設定時に、これらがハイ・レベルを保持するよう、次のように設定します。

(1) **P14**(P24)に 1 をセットする

(2) **PM14**(P24)に 0(出力モード)をセットする

(3) UART の設定を行う

(4) **P14**(P24)に 0 をセットする

上記(1)の設定をビット操作で行う場合、他のビットの出力ラッチの内容も書き換わることが考えられます。

また、これ以前に入力ポートの状態をハイ・レベルに保持するには、端子を抵抗でプルアップしておくことをお勧めします。

送信時開始時にあらかじめデータを 2 バイト書き込めるか [V850E/MA1]

Q1. クロック同期式シリアル通信で送信を開始するとき、シフト・レジスタ(SIOn)と送信バッファ・レジスタ(SOTBn)に 1 バイトずつ、合わせて 2 バイト書き込むことはできますか。

A1. いいえできません。V850E/MA1 の **CSI** はダブル・バッファとして機能しないため、最初に 2 バイト分のデータを設定することはできません。**SIOn** は読み出し専用で、書き込みは許されていません。

シリアル通信の受信が始まらない

Q1. シリアル転送で、データが受信できません。

A1. 受信側を起動していますか。クロック同期式シリアルは、動作許可にしただけでは、送信も受信もしません。送信はデータを書き込めば起動しますが、受信にも起動トリガが必要です。受信オンリ・モードの場合は **SIO** の読み出し、送受信モードの場合は **SOTB** へのダミー・データの書き込みが必要です。

Q2. 双方「送受信モード」に設定後、送信側の SOTB1 に送信データを書き込めば転送待ちの状態で、受信側が SOTB1 にダミー・データを書き込んで、データ転送が行われるのですか。

A2. いいえ、マスタである送信側は、**SOTB1** に送信データを書き込めば送信開始します(受信側と無関係)。
一方、スレーブである受信側は、転送の起動(送受信モードなら、SOTB にダミー・データの書き込み)がかかっていないと、マスタからシリアル・クロックが来ても無視してしまい、転送は行いません。
つまり、スレーブ側で転送の起動がかかった状態で、マスタ側が転送を起動する必要があります。

Q3. 書き込んだデータが、ダミー・データか本当のデータかは、プログラムで判断しなければならないのすか。

A3. はい、上位の通信の取り決め(プログラム)で判断することになります。3 線式シリアルでは、基本的に送信と同時に受信も行います。このため、送信完了時に受信したデータを使うかどうかは、プログラムしだいです。
なお、通常は受信起動時のダミー・データとして、FFH を書き込みます。したがって、通常のデータに FFH を使用しないようになっていれば、受信データ自体でも確認できます。

Q4. 電力低減のため、シリアルを常時動作させたくありません。受信側が受信起動をせずに受信できますか。

A4. ハンドシェークのための信号を別に設ければ実現できます。シリアル信号以外にハンドシェーク信号を準備しておき、転送前にマスタからスレーブに転送要求を出して、スレーブはそれを割り込みで受け、転送起動をかけたら準備完了をマスタに通知するような手順を追加してください。
また、スレーブの転送要求でも、転送起動後、ハンドシェーク信号でマスタに要求するようにしてください。

DMA 起動にソフトウエア・トリガ・ビットを使用 [V850E/MA1]

Q1. V850E/MA1 の DMA を使用した UART の送信処理で、チャネル・コントロール・レジスタのソフトウエア・トリガ・ビットの設定は、データ送信のトリガとなりますか。また、このビットをクリアしておく必要はありますか。

A1. はい、DMA チャネル・コントロール・レジスタのソフトウエア・トリガ・ビットを設定すれば、UART の送信を起動できます。なお、このビットは単にトリガのためのビットなので、わざわざクリアする必要はありません。

通信設定変更時に動作クロックを停止しないでいいか

Q1. シリアル・ポートの設定のうち、フォーマットだけ変更する場合、動作クロックを停止する必要はありますか。

A1. いいえ、動作クロックの停止は必要ありません。ただし、動作は停止(TxEn、RxEn＝0)する必要があります。

同期式シリアル通信で、TRMDn=1 のときはデータ受信できないか [V850E/MA1]

Q1. V850E/MA1 ユーザーズ・マニュアル ハードウエア編の「11.3.3 制御レジスタ」で、次の項目に CSIMn レジスタの TRMDn＝0 のときの動作についての記述があります
(3) シリアル IO シフト・レジスタ(SIO0-SIO2)
(4) 受信専用シリアル IO シフト・レジスタ(SIOE0-SIOE2)
TRMDn＝1 のときの記述はありませんが、このときは受信ができないのですか。

A1. いいえ、TRMDn＝1 のときも、送信と同時に受信動作ができます。この記述は、TRMDn＝1(送受信モード)のときの通信の起動が SOTBn への書き込みで、SIOn を読み出しても起動せず、TRMDn＝0(受信専用モード)のときに SIOn の読み出しで通信が起動することについてです。
また、受信専用モードで、受信データを読み出しても、次の通信を起動させたくないときのために、SIOEn が用意されています。TRMDn＝1(送受信モード)では、SIOEn を使用する必要がないので記載していません。

シリアル通信で PM22 を 1 にするタイミング [V850E/MS1]

Q1. シリアル・インタフェースを使用する場合、PM22 をどのタイミングで 1 にすればいいですか。

A1. V850E/MS1 のリセット解除後に、PM2 は FF になっています。したがって、初期値のままであれば、操作する必要はありません。どうしても設定をされるのなら、P2 に値を設定したあと、シリアルの設定を行う前に、PM22 を設定されることをお勧めします。他のビットで出力ポートに設定した端子に、不定値を出力しないようにするため、P2 の設定後とします。

UART 入力端子がロウ・レベルを持続したことによるエラーからの復帰

Q1. 受信データにロウ・レベルが持続すると、受信エラー(パリティ・エラー、ストップ・ビットなし)が発生しますが、プログラムで受信エラー処理(受信バッファ(RXB0)を読み出し)を行えば、正常に次の受信ができますか。

A1. いいえ、それでは不十分です。V850E/MS1 の UART は、入力がロウ・レベルの間、ボー・レートに従って連続的にデータを受信します。このため、この間は受信エラーが連続発生します。
なお、途中で入力がハイ・レベルに立ち上がれば、最後の部分では、正常受信と解釈されることも考えられます。この場合、異常なデータを受信してもエラーにならないことがあります。
これらを考慮した処理が必要です。たとえば、フレーミング・エラーと受信データ 00 の状態が連続発生したら、入力のハイ・レベルを外部割り込み信号で検出し、UART の動作を停止する処理が考えられます。

UART 非同期シリアルから入力されるブレーク信号を検出

Q1. 非同期シリアルの接続開始時に、端末側から送られて来るブレーク信号(180～250ms のマーク状態)を検出する方法はありますか。

A1. 該当するポートの PM レジスタが入力として設定されていれば、RXD0～2 の端子状態をポートとして読むことが可能です。これをプログラムでチェックしてください。

UART の送受信完了処理を割り込み以外で行う

Q1. V850/SF1 のシリアル・インタフェース(UART)の送受信完了処理を割り込み以外で行う方法はありますか。

A1. 割り込みをマスクして、STIC0 か STIC1 のビット 7 をプログラムで確認(ポーリング)し、セットされていたら送信完了と判断してください。受信については、CSIC1 か CSIC3 を確認してください。
どちら場合も、セットされていたら完了で、プログラムでフラグをリセットしてください。

UART 送信割り込みのタイミング [V850ES/Kx1 など]

Q1. 送信割り込みを使用した UART の半二重通信で、送信開始後に最初の割り込みは、どのタイミングで発生しますか。

A1. UART では送信完了割り込みなので、ストップ・ビットのタイミングで発生します。

備考 マイコンに内蔵されている UART の送信関係の割り込みは、ほとんどが「送信完了割り込み」です。
ただし、V850ES/SG2, SJ2 の UARTAn では、「送信許可割り込み」です。
いずれの場合にも、送信を許可しただけでは、割り込みは発生しません。最初のデータを書き込み、その送信が完了したときに、割り込みが発生します(V850ES/SG2, SJ2 では、送信シフト・レジスタにデータが転送されたとき)。
2 バイト目以降の送信データは、割り込みのタイミングで書き込むことになります。
最初のデータの送信と、2 バイト目以降では、処理が異なりますのでご注意ください。
なお、V850E/ME2 では、FIFO を内蔵しているため、送信の割り込みは特殊な動作となるので、ここでは触れません。

Q2. 送信許可時に最初の送信割り込みが発生し、以後、送信完了ごとに割り込みを発生させられますか。

A2. はい、できます。送信許可にしたあとで、送信完了の割り込み要求フラグをプログラムでセットしてください。これにより、送信完了の割り込みが発生したのと同じ動作をさせることができますので、データの送信処理を共通の割り込み処理ルーチンで行うことができます。

初期化時に送信完了の割り込み要求フラグをプログラムでセットすれば、このフラグを送信可能フラグとして使用できます。この場合、送信するデータがないときは、送信完了割り込みをマスクしておいてください。これにより、不要な割り込みの発生を抑えられます。

I²C バスでスタート・コンディションが発行できない [V850ES/Kx1]

Q1. IICF0 設定値が 0x01 では、マスタ通信スタート関数(動作許可、ストップ・コンディション検出後、スタート・コンディション生成)のスタート・コンディション・チェックにかかり、通信できません。IICF0= 0x03 なら動作します。

A1. I²C バスのコントローラ段階では、初期状態でバスが使用されているかどうかを示す情報が得られません。そのため、従来のコントローラでは、起動後にストップ・コンディションを検出して、初めてバスを使用できる(スタート・コンディションを発行できる)ようになっていました。したがって、上位のプログラムでバスが空いているかどうかを判断して、ストップ・コンディションを発行し、その後にスタート・コンディションを発行すれば、I²C バスを使用できるようになります。**IICF0** が 0x01 の場合は、これに該当します。

一方、V850ES/Kx1 のような新しい製品は、ストップ・コンディションを検出せずに、スタート・コンディションを発行できる機能が追加されています。この機能を使用する(**IICF0**=0x3 の場合が、これに該当します)ことで、ただちにスタート・コンディションを発行して、I²C バスを使用できるようになります。

3 線式シリアルでデータを送信したいが、クロックが出ない [V850ES/Kx1]

Q1. CSI00 の転送クロックにタイマ 50 の出力を使ってデータを転送したいのですが、データを書き込んでもクロックが出力されません。設定は次のとおりです。

```
(1) __DI();
(2) TCL50   = 0x04;
(3) CR50    = 0x11;
(4) TMC50   = 0x80;
(5) CSIC0   = 0x06;
(6) CSIM00  = 0xd0;
(7) CSIOIC0 = 0x05;
(8) PMC4    = 0x07;
(9) __EI();
```

A1. タイマ 5 の設定で、**TMC50** レジスタのビット 1(**TMC501**)が 0 になっているのが原因です。**TMC50** には、0x80 ではなく、0x82 を設定してください。

CSI00 のクロックは、あくまでも **TO50** です。**TMC501** がセットされていないと、出力フリップ・プロップが反転せず、**TO50** が変化しません。**TMC50** レジスタの設定では、実際に **TO50** を外部に出力する／しないにかかわらず、出力フリップ・プロップは反転動作するようにしてください。

自動送受信機能付きシリアルの使い方 [V850ES/Kx1]

Q1. 自動送受信機能付きシリアルで、データを送信する方法が分かりません。

A1. 自動送受信機能付きシリアル(CSIAn)で、マスタとしてデータを連続送信する手順は、次のようになります。

(1) CSIMAn レジスタの CSIAEn ビットを設定(CSIAn を動作許可)します。
　これで、バッファ RAM にアクセス可能になります。

(2) CSISn レジスタで、シリアル・クロックを選択します。

(3) BRGCAn レジスタで、シリアル・クロックの分周比を設定し通信速度を指定します。

(4) バッファ RAM の FFFFFE00H から順に送信したいデータを書き込みます。

(5) ADTPn レジスタに、「送信したいデータ数－1」を設定します。

(6) CSIMAn レジスタの ATEn ビットと MASTERn ビットで、自動転送モードでのマスタ動作を設定します。

(7) CSIMAn レジスタの TXEAn ビットを設定(送信可能)します。

(8) ADTIn レジスタに、データの送信間隔を設定します。

(9) CSIA0 の場合、PFC5 レジスタのビット 4,5 を 0 に、PMC5 レジスタのビット 4,5 を 1 に設定します。
　CSIA1 の場合、PFC9 レジスタのビット 11,12 を 1 に、PMC9 レジスタのビット 11,12 を 1 に設定します。

これで、自動送受信機能付きシリアルで、自動送信を行う場合の設定は完了しました。CSIT0n レジスタの ATSTAn ビットを設定すれば、自動送信処理が起動されます。すると、CSISn レジスタの TSFn ビットを設定し、バッファ RAM の先頭から順に読み出したデータを送信します。ADTPn で指定された数＋1 のデータを送信すると、割り込み要求(INTCSIAn)を発生し、 CSISn レジスタの TSFn ビットをクリアします。
引き続き、次のデータを送信したい場合は、新しいデータをバッファ RAM にセットし、(5)の ADTPn レジスタへのデータ数の設定を行ったあとに、CSIT0n レジスタの ATSTAn ビットを設定します。
送受信を行う場合は、(7)で RXEAn ビットも設定して、(9)で PFC5 レジスタのビット 3 も 0、また PMC5 レジスタのビット 3 も 1(CSIA1 では PFC9 レジスタのビット 10 も 1、また PMC9 レジスタのビット 10 も 1)に設定します。
転送が完了すると、バッファ RAM に受信したデータが格納されているので、読み出します。

Q2. 次のような設定を行いましたが、SCKA0 が出力されません。

```
CSIMA = 0xDC;
CSIS0 = 0x40;
CSIT0 = 0x01;
BRGCA0 = 0x00;
ADTP0 = 0x07;
ADTI0 = 0x03;
P5 = 0x07;
PM5 = 0xc8;
PMC5 = 0x38;
PF5 = 0x00;
PFC5 = 0x00;
PU5 = 0x00;
```

A2. 設定の順序や方法に問題があります。
この手順では、CSIMA0 レジスタのビット 7 と 6 だけを設定したあとに、残りのビットを設定されています
ATSTA0 ビット以外すべての設定が完了したあと、最後に CSIT0 レジスタの ATSTA0 を設定してください。

CSIB1 のクロック・ソースとしての TOP01 [V850ES/Fx2]

Q1. 3 線式シリアル・インターフェース CSIB1 の入力クロックとして TOP01 を使用する場合、TOP01 端子の出力を禁止できますか。

A1. TOP01 端子をポート・モードにしてください。

・TOP01 端子はポート・モードとして使用

・SCKB1 に TOP01 を入力クロックとして選択

・タイマ P のレジスタ設定後、タイマ P を動作

なお、タイマ出力を禁止にしてしまうと、TOP01 出力を CSIB1 のクロック・ソースとして使用できません。

1.3.4 タイマ／カウンタ

途中で CRn0/1 を変更した場合の出力波形 [V850/S]

Q1. 16 ビット・タイマの「PPG 出力機能」を動作させている状態で、任意のタイミングで CRn1 でパルス幅の変更を行うと、中途半端な延び縮みは起こりますか。

A1. はい、書き換えタイミングによっては、意図しない波形になることがあります。
これを避けるには、CRn1 の書き換えを CRn0 の一致割り込みで行うなどの処理が必要です。

Q2. 周期を変更する場合、タイマ(PPG 機能)を停止せずに、CRn0 の値を変更してもいいですか。

A2. はい。タイマを停止せずに変更できます。

Q3. 周期を変更する際、注意点はありますか。

A3. 「CRn0 の書き込み」と「CRn0 とタイマが一致するタイミング」が競合すると、一致検出が正しく行われないので、一致タイミング付近では、値を書き込まないでください。

複数のタイマを同期(同時にカウント・スタート) [V850E]

Q1. 3 個のタイマを同期させたいのですが、同時にカウント・スタートはできますか。

A1. できません。各タイマは、TMCCE0 をプログラムで設定すればスタートし、これは各タイマ個別に行う必要があります。したがって、命令の実行時間の差と、カウント・クロックの周期の影響分の差は発生します。

インターバル・タイマの設定値(期待値－1)

Q1. 外部クロックによるインターバル・タイマで、インターバルが少しずつ遅れてしまいます。

A1. インターバル時間は「n＋1」となるため、コンペア・レジスタには「インターバルの値－1」を設定してください。

タイマをカスケード接続したときのコンペア値の設定方法 [V850/SB1]

Q1. 8 ビット・タイマ TM2 と TM3 をカスケード接続して、16 ビット・タイマとして使用する場合、コンペア値は、CR20、CR30 に設定するのですか。それとも CR23 に設定するのですか。

A1. 16 ビット・レジスタの CR23 に設定してください。

Q2. CR23 にコンペア値を設定した場合、INTTM2、INTTM3 はどうなるのですか。

A2. カスケードで使用した場合、16 ビットの一致で INTTM2 が発生します。したがって、カスケード時の割り込みには INTTM2 をご使用ください。
なお、上位 8 ビットの一致で INTTM3 の割り込みが発生しますので、カスケードで使用されるときは、INTTM3 をマスクして、不要な割り込みがかからないようにしてください。

リアルタイム・パルス・ユニットのカウント・クロックの上限 [V853]

Q1. タイマ 1 のカウント・クロックを外部入力とした場合、カウント可能周波数はどのように決まるのですか。

A1. タイマ 1 の外部クロックは、エッジ検出のため、CPU クロック周波数の影響を受けます。確実にエッジ検出を行うには、ハイ・レベル、ロウ・レベルとも最低でも CLKOUT の 3 クロック分の幅が必要です。これに、内部で必要な時間を加味して決まります。規格では、最低 3T＋18ns 必要です(T は CLKOUT の周期)。

Q2. システム・クロックと同一周波数のクロックは、カウントできますか。

A2. いいえ、できません。上記 A1 のように、ハイ／ロウのレベルとも 3CLKOUT＋18ns 以上の時間が必要です。

リアルタイム・パルス・ユニットを PWM 出力として使用 [V853]

Q1. V853 のリアルタイム・パルス・ユニットで PWM 出力をします。任意の PWM 周期を設定するため、カウント値がレジスタ設定値に達したらカウンタをリセットしたいのですが、途中でリセットするには、TCLR 端子にリセット信号を入力する必要があります。タイマ自身のパルス出力をリセット信号に使用できますか。

A1. そのような使い方で問題ありません。ただし、最初から ECLRn をセット(外部入力によるクリアを許可)していると、TCLR1n が入力されないとタイマが起動しません。したがって、最初は ECLRn をセットしない状態で、CE1n をセットしてタイマを起動し、その後に ECLRn をセットして、外部クリアを有効にしてください。
または、起動時にタイマ出力端子をポートとしておき、TCLR1n 入力の有効エッジを出力(タイマを起動)したあとに、タイマ出力に切り替えてください。なお TCLR1n 入力は、外部での遅延は不要です。

タイマ制御レジスタの設定順序 [V853]

Q1. 次のタイマ制御レジスタの設定順序に、推奨や規則はありますか。

- キャプチャ／コンペア・レジスタ（CC1n0-3）
- タイマ出力コントロール・レジスタ（TOC1n）
- タイマ・コントロール・レジスタ（TMC1n）
- タイマ・ユニット・モード・レジスタ（TUM1n）

A1. 次のように、最初にモードを設定してから、コンペア・レジスタを設定してください。

(1) TMC でカウント・クロックのみを設定(CE は 0 のまま)
(2) TUM を設定
(3) TOC を設定
(4) CC を設定
(5) PMC を設定
(6) CE を 1 にしてから、TMC を設定してタイマを起動

16 ビット・タイマをインターバル・タイマとして使用 [V850ES/Kx1]

Q1. 16 ビット・タイマをインターバル・タイマ CR0n0 だけ使用すると、CR0n1 は初期値のままで、割り込みは発生しませんか。

A1. 初期状態では、割り込み制御レジスタの割り込みマスク・ビットが設定されているので、CR0n1 の一致信号が発生しても、割り込みが発生することはありません。

Q2. CR0n0 と一致すると、カウント・レジスタはクリアされますが、クリア状態と CR0n1(初期値 0000)を比較してもいいですか。

A2. CR0n1 との比較は行われます。このため、タイマ出力や CR0n1 の一致割り込みを使用すると、0000 でタイマ出力が反転したり、CR0n1 の一致割り込み要求フラグがセットされたりします。

タイマ／カウンタ機能 [V850ES/Kx1]

Q1. マニュアルに、「タイマ動作中に CR0n0, CR0n1(16 ビット・タイマ・キャプチャ/コンペア・レジスタ)を書き換えたとき、その値がタイマ値に近く、かつタイマ値より大きい場合、一致割り込み要求の発生やクリア動作が正常に行われない可能性があります」とあります。任意の時間間隔で、出力信号を反転(トグル出力)させる場合、タイマを停止せずに切り替えることはできますか。

A1. はい、できます。通常、このようなカウンタは、「CR0n0 の一致でクリア＆スタート」で使用し、CR0n0 との一致割り込みの中で、必要なら次の設定値に書き換えます。これにより、この注意事項を回避できます。

Q1. 上記 Q1 のマニュアル記述について、「その値がタイマ値に近く」というのは、どれだけの差のことですか。

A1. 具体的に、どれ位とは言えません。次の値を書き込むまでに、カウンタがその値を越えることが問題です。

タイマ 2 の動作 [V850E/IA]

Q1. マニュアルで「9.3.4 制御レジスタ (8)CMSE120」の BFEEn の説明に、キャプチャ・レジスタ・モードのバッファ使用のときに、「CPU が主レジスタ(CVPEn0)のリードを行うと、主レジスタは CPU がリード動作を行う直前の副レジスタ(CVSEn0)の値に更新されます。キャプチャ・イベントが発生すると、その時のタイマ／カウンタの値が常に副レジスタに保存されます」とあります。
外部信号(INTP24 端子)により、サブチャネル 4 がキャプチャを行ったタイミングで発生した INTCC24 割り込み内で、CPU が主レジスタのリードを行うと、どういう値が読めますか。

A1. TM20 のキャプチャ値が読めます。

Q2. 複数のキャプチャが発生して、CPU が INTCC24 割り込み内で副レジスタをリードすると、TM20 のキャプチャ値が読めるのですか。

A2. はい、そうです。ただし、副レジスタを読み出しても次のキャプチャを行わないので、主レジスタも読み出す必要があります。

Q3. バッファとして使用しない(BFEEn ビット＝0)とき、主レジスタをリードしないと次のキャプチャを行わないのはなぜですか。

A3. カスケード接続で両レジスタの同期をとる制御回路が内蔵されていて、副レジスタ、主レジスタの順に読み出せば、両者の値が同じタイミングで更新されます。この制御が、カスケード接続以外でも有効だからです。

8 ビット・タイマでのパルス幅測定 [V850/SB]

Q1. 8 ビット・タイマをカスケード接続して、DMA でタイマの値を読み込んで、パルス幅を測定できますか。

A1. 外部のパルスを **INTP6** か **INTP0** に入力して、これを DMA 起動トリガに指定すれば可能です。

Q2. 上記 A1 の場合、16 ビット・タイマでキャプチャするデータとは、異なるのですか。

A2. ほかに優先度の高い DMA がなければ同じです。ただし、**INTP6** を使用した場合、ノイズ除去クロックが低い周波数なら、大きくずれる可能性はあります。ノイズ除去クロックを fxx に設定してください。

Q3. 上記 A2 で異なる場合、その理由と、ずれるクロック数が一定かどうかを教えてください。

A3. 優先度の高い DMA の影響で、カウント値の読み出しタイミングがずれるのは当然ですが、ノイズ除去の影響では、外部の入力信号とノイズ除去クロックが非同期のため、サンプリング周期分は変動します。いずれの場合も、その影響は一定にはなりません。

INTP0n0/1 割り込みで STOP モードが解除できない [V850/SV2]

Q1. STOP モードの解除条件に、外部割り込み **INTP040, 041, 050, 051** を設定しましたが、割り込みがかからず、STOP モードが解除できません。通常動作時は、これらの外部割り込みは正常にかかります。

A1. タイマのキャプチャ信号に使用される外部割り込み信号は、タイマが停止すると動作しません。STOP モードでは、タイマのクロックが停止（タイマ停止）するので、これらの外部割り込みでの解除はできません。

1.3.5 ウォッチドッグ・タイマ

ウォッチドッグ・タイマのタイマ割り込み処理でのクリア

Q1. CPU の暴走対策でウォッチドッグ・タイマを使用します。プログラムの簡略化のため、定期的にかかるタイマ割り込みで、ウォッチドッグ・タイマをクリアしていいですか。

A1. それはお勧めできません。この方式では、メイン・ルーチンが暴走しても、タイマ割り込みが受け付けられれば、ウォッチドッグ・タイマがクリアされてしまい、暴走を検出できません。
より厳密な暴走検出を行うには、一定時間内に必ず処理しないといけない部分に、個別にフラグを準備しておき、その処理後にフラグをセットします。タイマ割り込みなど、定期的に動作するプログラムでそのフラグを確認し、必要なフラグがすべて設定されていたら、ウォッチドッグ・タイマをクリアするなどしてください。

ウォッチドッグ・タイマのタイムアウト後の動作 [共通]

Q1. ウォッチドッグ・タイマのタイムアウト後、CPU はどのような動作をするのですか。

A1. ウォッチドッグ・タイマのタイムアウトでノンマスカブル割り込み要求(**INTWDT**)が発生します。NMI 処理中でなければそれを受け付け、例外コード 0020H の割り込みとなり、ハンドラ・アドレス(00000020H)に分岐します。
詳細は V850/SA1, SB などのユーザーズ・マニュアルで「5.2 ノンマスカブル割り込み」をご参照ください。

備考 V850ES/Kx1, SG2, SJ2, SA2, SA3 では、ウォッチドッグ・タイマがタイムアウトとすると、リセットするかノンマスカブル割り込みを発生するかを指定できます。

STOP モードに入る前のウォッチドッグ・タイマのクリア(RUN=1) [共通]

Q1. STOP モードで WDT は停止するのに、STOP モードに入る前に RUN＝1 にする必要があるのはなぜですか。

A1. STOP モードに入る直前に、オーバフローが考えられるからです。このとき、STOP モードに入れません。
これを避けるため、STOP モードに入る前に WDT をクリアしてください。

ウォッチドッグ・タイマのプログラムでの動作停止・開始 [共通]

Q1. ウォッチドッグ・タイマは、プログラム実行中に動作停止・開始ができますか。

A1. ウォッチドッグ・タイマは、一度起動するとソフト的に停止させられません。このため、ソフトの監視に使用できます。なお、V850ES/Kx1 のウォッチドッグ・タイマ 2 では、動作が異なります（下記 Q2 参照）。

Q2. V850ES/Kx1 などで、ウォッチドッグ・タイマ・イネーブル・レジスタ(WDTE)に ACH を書き込んだあと、ウォッチドッグ・タイマ・モード・レジスタ 2(WDTM2)に動作停止を設定すれば、ウォッチドッグ・タイマは停止しますか。

A2. はい、停止します。ユーザーズ・マニュアルの「11.2.1 機能」の注 1 に、次のように記載されています。
ウォッチドッグ・タイマ 2 は、リセット解除後に自動的にリセット・モードでスタートします。
ウォッチドッグ・タイマ 2 を使用しない場合は、この機能によるリセットが発生する前に停止するか、一度ウォッチドッグ・タイマ 2 をクリアし、次のインターバル時間内で停止してください。

Q3. なぜ、ウォッチドッグ・タイマは、プログラム実行中に動作停止／再開ができないのですか。

A3. ウォッチドッグ・タイマは、プログラムの暴走を検出する手段です。プログラムがどのような状態でも動作しないと、暴走の監視ができません。プログラムで停止できると、正常な処理での停止ならいいですが、プログラムの異常動作で停止する可能性があります。最近のマイコンでは、動作クロックを停止するスタンバイ機能が搭載されています。その影響を受けないクロックで、動作可能なウォッチドッグ・タイマもあります。

ウォッチドッグ・タイマ割り込みでの初期化処理 [V850/SA、V850/SB]

Q1. V850 で、ウォッチドッグ割り込み時にリセット処理をすると、割り込みが正常に動作しなくなります。

A1. ウォッチドッグ・タイマ割り込み中は、他の割り込み要求を受け付けません。NMI か通常の割り込みかで、状況が異なります。通常の割り込みの場合は、ISPR がセットされているので、次の処理を行ってください。

(1) EIPC と EIPSW にダミー値(EIPC には次の ISPR をチェックしているアドレス、EIPSW には 20H)を設定
(2) ISPR をチェックして、"0"でなければ、RETI 命令を実行
(3) ISPR が"0"になったら、リセット処理部に分岐

NMI の場合は、PSW の NP ビットがセットされ、他の割り込み要求を受け付けない状態です。また、NMI がかかった時点で処理中の割り込みがあれば、それに対する ISPR レジスタがセットされています。まず、NP ビット・クリアしてから、上記(1)～(3)の処理を行ってください。次に、具体的なプログラム例を示します。

```
            MOV     #loop,r10
            LDSR    r10,0
            MOV     0x20,r10
            LDSR    r10,1
Loop:       LD.BU   ISPR,r10
            CMP     r0,r10
            BZ      next
            RETI
Next:       JMP     [r0]
```

1.3.6 割り込み

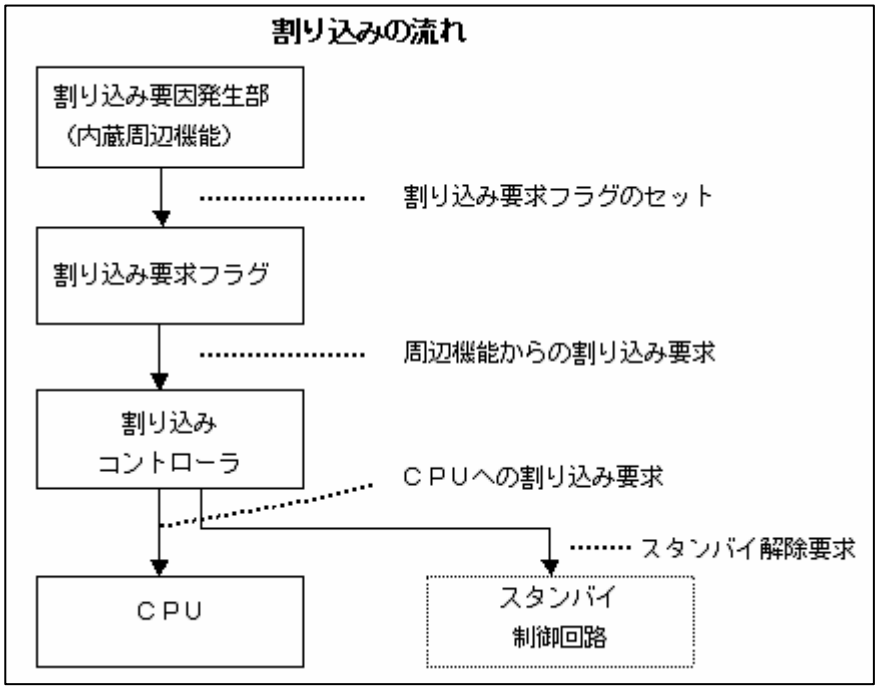
割り込みの動作の基本 [共通]

Q1. 割り込み処理について、どのような動作になり、どのような点に注意すればいいですか。

A1. [はじめに]

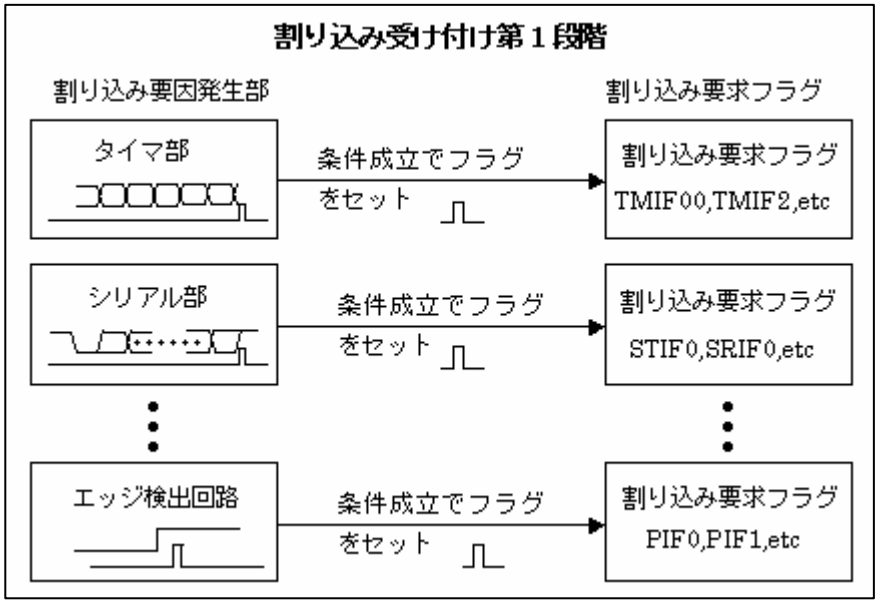
割り込み処理の基本的な動作は、大きく 4 つの部分に分けて考えることができます。

最初は割り込みの要因を発生させる部分で、2 つ目が割り込み要求フラグ、3 つ目が割り込みの制御を行う割り込みコントローラ部、最後が実際に割り込みを処理する CPU 部です。



[割り込みの要因発生部]

割り込みの要因を発生させる部分には、タイマ部やシリアル部、外部信号のエッジ検出回路等があります（デバイスで異なる）。各部では、タイマの一致やシリアル・データの受信完了など、割り込み発生条件を満足すると、対応した割り込み要求フラグをセットします。



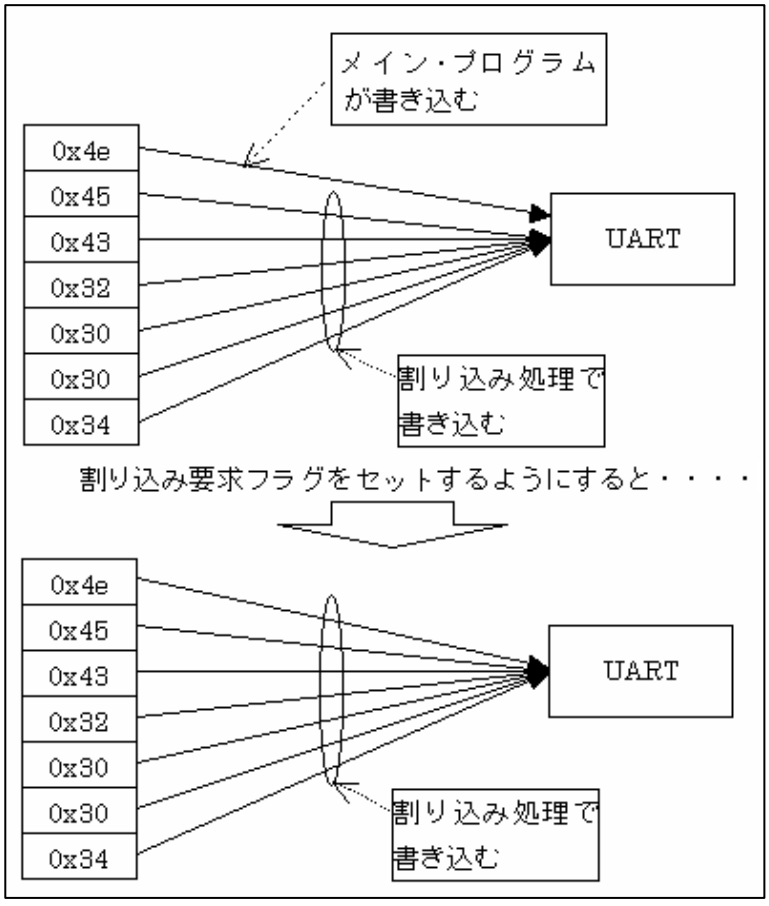
割り込みの第1段階は、各周辺機能部で指定された条件が満足されることです。タイマ部ではコンペア・レジスタとの一致等で、シリアル部では通信完了等で、外部割り込みではエッジ検出により、対応した割り込み要求フラグをセットします。また、プログラムで割り込み要求フラグをセットすることもできます。
なお、割り込み要因をDMAの起動要因として使用する場合は、この割り込み要求フラグとは別のフラグが使用されます。DMA転送でも、この割り込み要求フラグはセットされますので、不要な割り込み処理を行わないように、マスクをセットしておく必要があります。

[割り込み要求フラグ]

このように、各割り込み要因は対応した割り込み要求フラグをセットするだけで、割り込みの要因を発生させる部分と割り込みコントローラ部の動作は独立しています。この間を結ぶのが、割り込み要求フラグです。
なお、各割り込み要因に対応した割り込み要求フラグは、1 つ(1 ビット)しかありません。したがって、割り込み要求が受け付けられる前に、同じ割り込み要求が複数回発生しても、それは1回としか認識できません。
また、割り込み要求フラグは、プログラムでセットしたりクリアしたりすることもできます。これにより、UARTでの送信処理を、最初のデータを含めて、すべて割り込みで実行させることができます。ただしこの場合には、必ずビット操作命令を用いて、1 命令で割り込み要求フラグをセットする必要があります。

<UART 送信処理例>

複数バイトのデータを送信する場合、通常はメイン・プログラムが出力データの1バイト目を UART に書き込むとともに、2 バイト目以降のデータの情報を割り込み処理プログラムに引き渡します。
割り込み処理プログラムは、メイン・プログラムから渡された情報を元にして、2 バイト目以降のデータを UART に書き込みます。
このように、1 バイト目と 2 バイト目以降では、処理が異なります。



メイン・プログラムは、割り込み処理プログラムに送信データの情報を渡し、割り込み要求フラグをセットするだけとなります。実際の UART への書き込みは、すべて割り込み処理プログラムが処理しますので、全体の処理が見やすくなります。

[割り込みコントローラ部]

割り込み要求フラグ以降が、割り込みコントローラ部の動作となります。

割り込みコントローラ部では、まず割り込み要求フラグが割り込みマスク・レジスタで有効にする(マスクしない)か無効にする(マスクする)かが決められます。

マスクして無効に設定された割り込み要求フラグは、以降の動作で無視されます。

ただし、要求フラグ自体がクリアされるのではないため、マスクを解除したら、その時点で割り込み要求が有効になってしまいます。

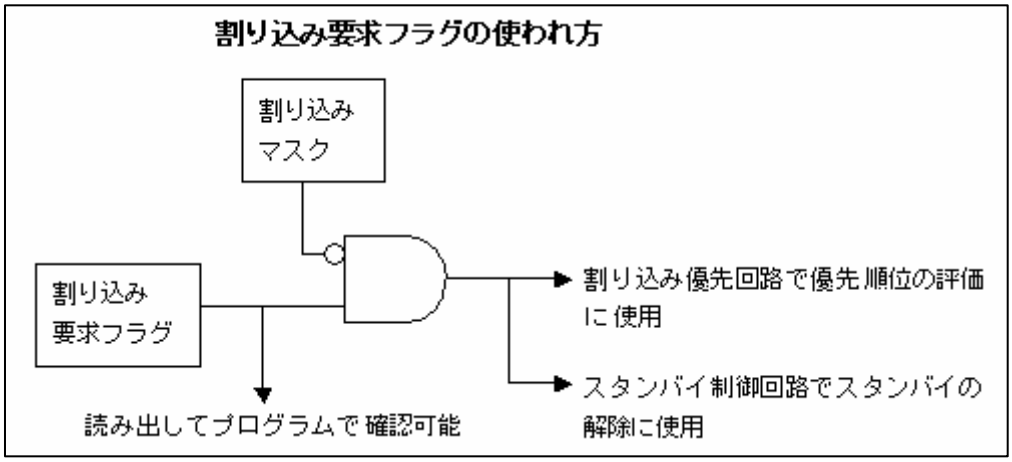
特に、DMAの起動要因に使用していた割り込み要求フラグは、DMA完了時にクリアする必要があります。

マスクされなかった割り込み要求は、スタンバイの制御回路に送られ、スタンバイの解除に使用されます。

また、この割り込み要求は、割り込み優先回路を介して、CPU への割り込み要求となります。CPU が割り込み要求を受け付けると、対応した割り込み要求フラグはクリアされます。

CPU が割り込み禁止状態では、割り込みは単にスタンバイ解除に使用されます(スタンバイ状態でない場合には保留されるだけです)。

割り込み許可状態では、スタンバイ解除後、割り込み処理を起動して、対応したハンドラ・アドレスに分岐します。



割り込み要求フラグには、次の 3 つの使い方があります。

(1) プログラムでポーリング(読み出し)を行い、割り込み要求元の処理が完了したかどうかを確認

この場合、割り込みマスクをセットしておき、CPU への割り込みは発生させません。

確認後は、必ずフラグをクリアしてください。

(2) スタンバイの解除に使用

スタンバイ状態で、マスクされていない割り込み要求フラグがセットされると、スタンバイが解除されます。

このとき、CPU が割り込み禁止状態であれば、スタンバイ解除後はベクタ処理をせず、スタンバイ命令の次から実行を再開します(78K4 でマクロ・サービスを実行する場合、スタンバイは解除されません)。

確認後は、必ずフラグをクリアしてください。

(3) CPU へのベクタ割り込み要求として使用

この場合には、他の割り込み要求フラグの状態と、CPU が実行中の割り込みの状態が、割り込み優先回路でチェックされ、最も優先度が高ければ、CPU への割り込み要求となります(フラグは自動クリア)。

割り込み優先回路は、同時に発生している他の割り込み要求や、現在処理中の割り込みとの優先度比較を行います。このため、ISPR レジスタに CPU が実行している割り込みの状態を記憶しています。
割り込み処理を完了して、RETI 命令を実行すると、ISPR レジスタの最も優先度の高い割り込みに対応したビットをクリアして、処理中の割り込みの完了となります。なお、ISPR レジスタは、プログラムで書き込むことはできません。
V850/S では、CPU が割り込み禁止状態のときには、優先度比較回路の動作を停止しており、割り込み禁止状態から許可にしても、すぐには CPU への割り込み要求信号が発生せず、優先度の比較が完了してから発生します。このため、割り込み許可期間が7クロック以下では、割り込み要求は受け付けられません。

注意 割り込み優先度や割り込みマスクは、割り込み要求フラグと同じレジスタ内に配置されています。そのため、割り込みの優先度やマスクの操作にビット操作命令を使用しないと、セットされた割り込み要求フラグがクリアされることがあります（FAQ「割り込み制御レジスタの操作と割り込み［V850 共通］」もご参照ください）。

[CPU 部]
CPU が割り込み許可状態なら、優先度がもっとも高い割り込み(割り込み制御回路で決定)を受け付け、PC や PSW の値をスタックに保存して、割り込みハンドラに分岐します。このとき、該当する割り込み要求フラグはクリアされ、CPU は割り込み禁止状態になります。
EIPC や EIPSW レジスタは 1 組しかないので、多重割り込みをさせたい場合には、割り込みを許可する前に、これらの内容をスタックなどに保存しておきます。さらに、割り込み処理から復帰する前に、多重割り込みの受け付けを禁止して、スタックなどに保存した EIPC や EIPSW の値を元に戻しておく必要があります。

なお、CPU が割り込み要求を受け付けて、ハンドラに分岐するタイミングは、命令の実行状態によって異なり、いつも同じタイミングにはなりません。割り込み要求を保留する命令や、割り込み禁止期間があると、大きく変動します。このため、細かなタイミングが問題になる場合には、割り込み処理ではなく、タイマの出力機能やリアルタイム出力機能等のハードウエア機能を使用してください。

割り込み処理が完了したら、RETI 命令で元の処理に戻ります。
このとき、PSW は割り込み受け付け前の値に書き戻されますので、 RETI 命令の直前に EI 命令を実行する必要はありません。

割り込みがかからない(1) [共通]

Q1. PSW の ID ビットが 0 で、割り込み要求がセットされ、割り込み処理も許可していますが、外部割り込み要求が受け付けられません。

A1. 次のような原因が考えられます。

- 割り込みにマスク(割り込み制御レジスタのビット 6)がセットされている
- より優先度の高い割り込みの処理中(ISPR レジスタで御確認ください)である
- 割り込みを受け付けない命令が連続している

ISPR レジスタ、割り込み制御レジスタをご確認ください。

割り込みがかからない(2) [V850/S]

Q1. INTP0 を次のように設定していますが、割り込みが発生しません。

```
P0    = 0x00;
PM0   = 0x02;
EGP0 = 0x00;   /* 割り込み↑エッジ指定 */
EGN0 = 0x00;   /* 割り込み↓エッジ指定 */
```

A1. この設定では、エッジ検出で立ち上がり/立ち下がりともに、割り込みを発生しないようになっています。
INTP0 を使用するには、EGP0 か EGNP0 のどちらかのビット 1 を設定してください。

割り込みがかからない(3) [共通]

Q1. マスカブル割り込みをさせたいのに、処理ルーチンに分岐しません。EI 命令の実行で、NP＝1、ID＝1 となります。

A1. NP＝1 の状態(NMI を処理中を示す)では、マスカブル割り込み要求を受け付けません。したがって、EI 命令を実行しても、割り込み可能にはなりません。
NP＝1 となる原因は、次のいずれかが考えられます。

- NMI が受け付けられている
- LDSR 命令で PSW を書き換えている
- マスカブル割り込みからの復帰時に、EIPSW にセットされていた値(PSW に戻す値)がおかしい

割り込み(NMI)処理の中から初期化処理に分岐して、再初期化をされてはいないでしょうか。割り込み処理中には、その状態をフラグ(NP ビットや ISPR レジスタ)に記憶しており、この状態は RETI 命令を実行するまで保持されます。これらのフラグをそのままにしておくと、割り込み要求を受け付けなくなります。

参考 FAQ

割り込みからの初期化 [V850 共通]
ウォッチドッグ・タイマ割り込みでの初期化処理 [V850/SA, V850/SB]

割り込み要求非サンプル命令 [共通]

Q1. ユーザーズ・マニュアルに「割り込み要求非サンプル命令とその次の命令の間では、割り込みを受け付けません」とありますが、この期間(7 クロック)は、割り込みが禁止されてしまうのですか。

A1. いいえ、「非サンプル命令とその次の命令の間」は、CPU が割り込み要求の受け付け(サンプリング)をしない期間です。「非サンプル命令と次の命令は、必ず連続して実行される」と解釈してください。
なお、7 クロックは、DI 状態で EI 命令を実行してから、CPU が割り込み要求を受け付けられるようになるまでの時間です。これらは、独立な項目です(V850/S 共通)。

割り込みが一度しか発生しない [共通]

Q1. TM0 をインターバル・タイマとして使用します。割り込みの発生は、起動後 1 度だけで、あとは発生しません。

A1. 割り込み終了時に、RETI 命令が実行されていないためです。

V850 では、割り込みに 8 レベルの優先度が設定でき、これを ISPR レジスタで管理しています。この 8 ビットのレジスタは、割り込みの優先度にビットが対応しており、割り込み要求を受け付けると、優先度に対応したビットがセットされます。ISPR レジスタのビットがセットされると、そのビットに対応した優先度より高い優先度の割り込みしか受け付けません。

割り込み処理の最後で、RETI 命令を実行して割り込み処理から抜けると、ISPR レジスタのセットされているビットの中で、もっとも高い優先度の割り込みに対応したビットをクリアします。

このようにして、割り込みの優先度の管理を行っていますので、割り込み処理を終了するときは、RETI 命令を実行する必要があります。

割り込みハンドラ・アドレス [共通]

Q1. 割り込み／例外テーブルは、ROM 上の固定アドレス 00000000H-00000272H を占有しますが、これは個々の割り込みが発生すると、それに応じた処理プログラムをその番地に置く必要があるということですか。

A1. はい、そのような使い方になります。ただし、ほとんどの場合は 16 バイトで処理ができないので、ここに分岐命令を置いて、実際に割り込みを処理する割り込みハンドラにジャンプさせて処理することになります。

Q2. ハンドラ・アドレス部は、16 バイトまで使用していいですか。

A2. 各ハンドラ・アドレスで 16 バイト分は確保されていますので、その分は使用できます。

16 バイト以下で済む処理ならそれで終了ですし、不足する場合は残りの処理を別のところに配置しておき、そこに分岐させれば問題はありません。ただし、この場合は、レジスタを 1 つ使用します。

Q3. 使用しない割り込みに対応するテーブル領域も、空けておく必要はありますか。

A3. いいえ、このメモリ領域は、割り込みで使用するために確保されており、その割り込みを使わなければ、対応する部分は、通常のメモリ領域として使うことができます。

Q4. V850E/MA1 のシングルチップ・モード 0 で、割り込みハンドラ・アドレスは 0～0x360 ですが、シングルチップ・モード 1 ではどうなるのですか。

A4. ハンドラ・アドレスは同じです。外部に接続した ROM の、該当するアドレスに、割り込みハンドラを書き込んでおいてください。

割り込み時のエッジ判断 [共通]

Q1. V850E/MA1 の割り込み機能で、立ち上がり／立ち下がりの両エッジで割り込みがかかるように設定します。どちらのエッジなのかは、割り込み処理の中で、ポートの状態を確認して判断すればいいですか。

A1. はい、それで結構です。V850 では、端子を兼用機能で使用する場合にも、入力ポートとして、端子の状態を読み出すことができます。ただし、その場合は、PM レジスタで入力ポートに設定されている必要があります。

なお、V850/S では、外部割り込み機能を使用する際に、必ず入力に設定されているはずなので、追加設定は必要ありません。

Q2. PORT0 のビット 7 を外部割り込み要求入力とします。割り込み処理で、この入力を P07 から読み出すために、PMC07 を入出力モードにしてから、PM07 を入力モードに設定しても、P07 から正常値が読み出せません。

A2. 先に PMC をポート・モードにしたため、出力バッファが ON になり、端子で信号が衝突したと思われます。
外部割り込み端子の状態を確認する場合は、単に PM07 を入力に設定するだけで結構です。
PMC07 は、割り込み側の設定のままで、ポートを読んでください。

外部割り込み端子のレベルで割り込み発生 [共通]

Q1. 外部割り込みで、レベルを検出して、割り込みを連続して発生させることはできますか。

A1. V850E/MA1 では、ロウ・レベルでのレベル検出機能があります。
ほとんどの V850 のデバイスは、エッジによる外部割り込みしかサポートしていません。ただし、割り込み入力信号の状態をポートを介して読み込むことで、レベル検出に近いことが実現できます。ロウ・レベルで割り込み処理をする場合、次の設定をしておきます。

(1) PM レジスタの設定を入力ポートにする
(2) エッジ検出を立ち下がりに設定する

そして、エッジ検出の割り込み要求を受け付けたときに、次の処理を行います。

(3) 割り込み時のプログラム処理を行う
(4) 割り込み処理を抜ける前に、割り込み要求入力端子の状態をポートを読んで確認する
(5) 端子の状態がハイ・レベルなら、割り込み処理を抜け出す
(6) 端子がロウ・レベルのままなら、対応した割り込み要求フラグをセットして、割り込み処理を抜け出す

(6)で割り込み要求フラグがセットされれば、割り込みから復帰後に、再度割り込み要求を受け付けます。
V850E/MA1 以外の V850 の場合、これで、レベル検出割り込みに近い処理が実現できます。

割り込み要求フラグ(P02IF1)がセットされない [V850E/MA1]

Q1. ポートの P22(INTP021)を立ち下がりエッジの外部割り込みに設定していますが、割り込み要求フラグ(P02IF1)がセットされません。

A1. INTP021 を使用するには、TMCC20 レジスタの TMCCAE2 をセットする必要があります。ユーザーズ・マニュアルの「7.3.9 割り込みトリガ・モードの選択」に注意事項として記載されています。

不正命令コードの例外処理 [共通]

Q1. V850 の不正命令コードの例外処理について教えてください。

A1. V850 では、命令の割り当てられていない命令コードの中で、特定の組み合わせを不正命令と判断します。
不正命令と判断するコードについては、CPU コアによって異なりますので、使用されるコアのアーキテクチャ編のマニュアルの例外トラップの項目をご参照ください。
これらのコードに該当する命令を実行しようとすると、例外トラップが発生して、60H 番地のハンドラ・アドレスに分岐します。そこに、不正命令に対する処理を準備し、未使用メモリ領域を不正コードで埋めておくことで、システムの誤動作を検出する一手段として使用できます。
なお、ほとんどの未定義コードを網羅していますが、すべてをカバーしているわけではありません。

UART 送信時の最初の割り込み [共通]

Q1. 送信割り込みを使用した UART の半二重通信で、送信開始の割り込みはいつ発生しますか。

A1. そのようなタイミングでの、割り込みは発生しません。シリアルの割り込みは、基本的に通信完了(この場合は STOP ビットの先頭)で発生します(ほとんどの V850)。
送信開始を割り込みで行う場合、プログラムで、該当する割り込み要求フラグをセットすることが考えられます。なお、このとき、データを送信中でないことを確認してから、フラグをセットする必要があります。
「シリアル通信」の「UART 送信割り込みのタイミングについて [V850ES/Kx1 他]」もご参照ください。

多重割り込み [共通]

Q1. V850E/IA2 で、タイマ割り込み中に、高優先度の割り込みを処理します。タイマ割り込み関数は、多重割り込みの宣言(_multi_interrupt)をしていますが、タイマ割り込み中は、高優先度の割り込みが発生しません。

A1. 多重割り込み宣言は、あくまでも多重割り込みのための準備(通常の割り込み処理に加えて、EIPC と EIPSW に保存されている戻りアドレスなどの情報をセーブ)です。どのタイミングで多重で割り込み要求を受け付けるかは、各システムで異なるので、多重割り込みを許可するタイミングで「__EI();」を実行してください。
★ また、割り込み関数から抜けるときには、セーブしていた情報をEIPC,EIPSWにロードしますが、多重割り込み許可の状態で割り込みが発生すると、EIPC,EIPSWを上書きしてしまいます。このため、多重割り込み宣言した割り込み関数から抜けるときには、割り込み禁止状態である必要があります。__EI();を実行したら、必ず__DI();を実行してから、EIPC,EIPSWをロードして、割り込みを終了してください。

Q2. 割り込み A, B の優先度は同一で、その処理中は割り込みを許可しない設定とします。この場合、割り込み A の処理中に、割り込み要求 B が発生すると、割り込み要求 B は保留されますか。

A2. はい、割り込み要求 B は保留されます。

Q3. Q2 の設定で、割り込み A の処理中に割り込み要求 B が発生し、さらに割り込み要求 A が発生したとします。割り込み A の処理完了後の処理順序は、発生順に割り込み B、割り込み A ですか。

A3. いいえ、割り込み要求の発生順序とは無関係です。ディフォールト優先度の高い方が、先に処理されます。

割り込み処理中の割り込み [共通]

Q1. 割り込み処理中に、同一の割り込み要求が 2 回発生すると、処理は合計 3 回行われますか。

A1. いいえ、実行は 2 回です。割り込み要求は 1 ビットのフラグなので、要求の有無しか記憶できません。
このフラグは、割り込み要求を受け付けるとクリアされ、割り込み処理中には、あとの要求の有無しか分かりません。つまり、割り込み処理中に複数の同一割り込み要求が発生しても、2 回目以降は無視されます。

バス・アクセスと割り込み [共通]

Q1. V850E/MA1 で、外部に I/O を拡張しています。外部をワード・アクセスすると、2 回の 16 ビット・アクセスになります。このとき、1 回目のアクセスと 2 回目のアクセスの間に、割り込みが入ることはありますか。

A1. いいえ、1 つの命令の実行の間に、割り込みが入ることは、基本的にありません。2 回の 16 ビット・アクセスの間は、バスがロックされますので、他のサイクルが入ることもありません。

割り込み制御レジスタの操作と割り込み [共通]

Q1. 次の処理を行い、PIC1 レジスタで INTP1 のマスクと優先度を設定すると、割り込みが抜けることがあります。

```
if(条件) {
    pic1 = 0x02;    /* 通常状態 */
}
else {
    pic1 = 0x47;
}
```

A1. 割り込み要求フラグがマスク・フラグ、優先度と同じ PIC1 レジスタに割り当てられており、PIC1 レジスタの書き込みで、同時に PIF1(割り込み要求フラグ)ビットにも書き込み(0 クリア)が行われるためです。このために、書き込み時に保留されていた(セットされていた)割り込み要求がクリアされ、割り込みが抜けます。
これを避けるには、PIC1 レジスタの操作を割り込みが発生しないタイミング(たとえば、INTP1 の割り込みが発生した直後など)で行ってください。または、ビット操作命令により、1 ビットずつ変更してください。

Q2. 割り込み制御レジスタのマスク・ビットで、割り込み禁止／許可をした場合、禁止～許可中に割り込み要求があると、割り込みを保留して、許可後に実行しますか。

A2. マスク解除を 8 ビット・アクセスで行う(C言語ベースのプログラムで、割り込み制御レジスタを直接ビット操作せず、複数の命令に展開された)場合は、それまでの割り込み要求は消えてしまいます。
マスク解除をビット操作で行う場合、その時点までの割り込み要求は、マスク解除により受け付けられるようになります。

割り込みからの初期化 [共通]

Q1. 割り込み内からプログラムの先頭(初期化)にジャンプする方法として、EIPC レジスタ、EIPSW にダミー・データをセットして、RETI でジャンプさせています。以前の割り込みレベルを保持して、他の割り込みがかからないことがあります。

A1. 割り込みが多重にかかっていないでしょうか。この場合には、1 回の RETI 命令の実行で、最優先の割り込みに対応した ISPR レジスタのビットがクリアされるだけです。ISPR レジスタが 00 になるまで、RETI を繰り返す必要があります。
「ウォッチドッグ・タイマ割り込みでの初期化処理 [V850/SA, V850/SB]」をご参照ください。

ソフトによる割り込みトリガ [共通]

Q1. INTPn 端子をポートとして使用し、ソフトで INTPn 割り込みを起動(割り込み要求フラグをセット)しています。外部入力によるトリガはかけません。このような使い方で、INTPn 割り込みは誤動作しませんか。

A1. INTPn 端子のエッジ検出を行わないようにすれば、その変化が原因となる割り込みは発生しません。つまり、この場合には、割り込み要求フラグを操作して、ソフトで割り込みをかけても問題ありません。

1.3.7 PPG 出力と PWM 出力の違い

PPG 出力と PWM 出力の違い [共通]

Q1. タイマ出力機能に PPG 出力と PWM 出力があり、同じような信号を出力していますが、違いはなにですか。

A1. PPG 出力は、「周期と幅」を指定してパルスを出力します。この機能では、波形(タイミング)が重要です。このため、パラメータ(パルス幅と周期)の変更は、すぐ動作に反映されます。そのため、変更のタイミングによっては、出力が反転してしまうことがあります。

PWM 出力は、アナログ量を出力するために、指定された周期の中で、アクティブな期間を指定します。この機能では、波形よりアクティブ期間の比率が重要です。そのため、アクティブ期間を複数に分割して出力する機能をもったマイコンもあります。また PWM 出力では、任意のタイミングでパラメータ(周期やアクティブ期間)を変更しても、周期の切り換わりのところで反映され、おかしな出力にはなりません。そのかわり、パラメータが変更されても、すぐには動作に反映されません。

Q2. PPG 機能を用いて、PWM 信号を出力できますか。また、そのときの注意点はありますか。

A2. はい、PPG 機能を用いて PWM 信号を出力できます。その場合は、周期やアクティブ期間の書き換えに注意が必要です。周期の書き換えは、必ず 1 周期の終了を示す割り込みタイミングで行ってください。

使用するタイマによっては、カウント動作を停止してから書き換える必要があります。アクティブ期間の変更は、カウントを停止せずにできますが、書き換えタイミングには注意が必要です。

アクティブ期間を指定する比較レジスタとの一致が、1 周期の中で 2 回発生したり、全く発生しなかったりすると、出力が反転してしまうタイマもあります。

1.3.8 DMA

DMA の使い方(リアルタイム出力機能と組み合わせ) [V850/Sxx]

Q1. V850 で、あまり CPU に負担をかけない(リアルタイム出力を使用)モータ制御方法を教えてください。

A1. モータ制御のようにタイミング制限が厳しい出力では、タイマ機能、DMA 機能をリアルタイム出力機能と組み合わせます。4 ビットのリアルタイム出力(**RTP0～RTP3**)を考えましょう。この場合、出力タイミングはタイマ 4 (**INTTM4**)で制御することになります。また、同じタイミング(**INTTM4** を DMA 転送のトリガにも使用)で、次の出力データを **RTBL** に転送するので、DMA はチャネル 0 かチャネル 1 を使用します。事前に、出力パターンを内蔵 RAM の下位 4 ビットに作成して、次の設定を行ってください。

(1) タイマ 4 は、インターバル・タイマ機能により、一定周期で割り込み要求信号を発生します。割り込み周期は、**TCL4** レジスタでタイマのカウント・クロックを選択して、「コンペア・レジスタ **CR40** の値＋1」カウント分の時間で設定します。ただし、まだタイマはスタートしません。

(2) DMA は、次のように設定します。

DIOA0/1 ：RTBL のアドレスの下位 10 ビットである 3A0H を設定

DRA0/1 ：データを格納する RAM の下位アドレスを設定(有効ビット数は CPU の内蔵 RAM 容量による)

DBC0/1 ：データ数－1 を設定

DCH0/1 ：動作モードとして 19H を設定

内蔵 RAM アドレスはインクリメント、内蔵 RAM→周辺 I/O で 8 ビット・データを転送するように設定されます。

(3) リアルタイム出力は、まず **RTP0～RTP3** と兼用になっているポート(P100～P103)を設定します。

P100～103　：初期値

PM100～103　：xxxx0000B(下位 4 ビットを出力)を設定

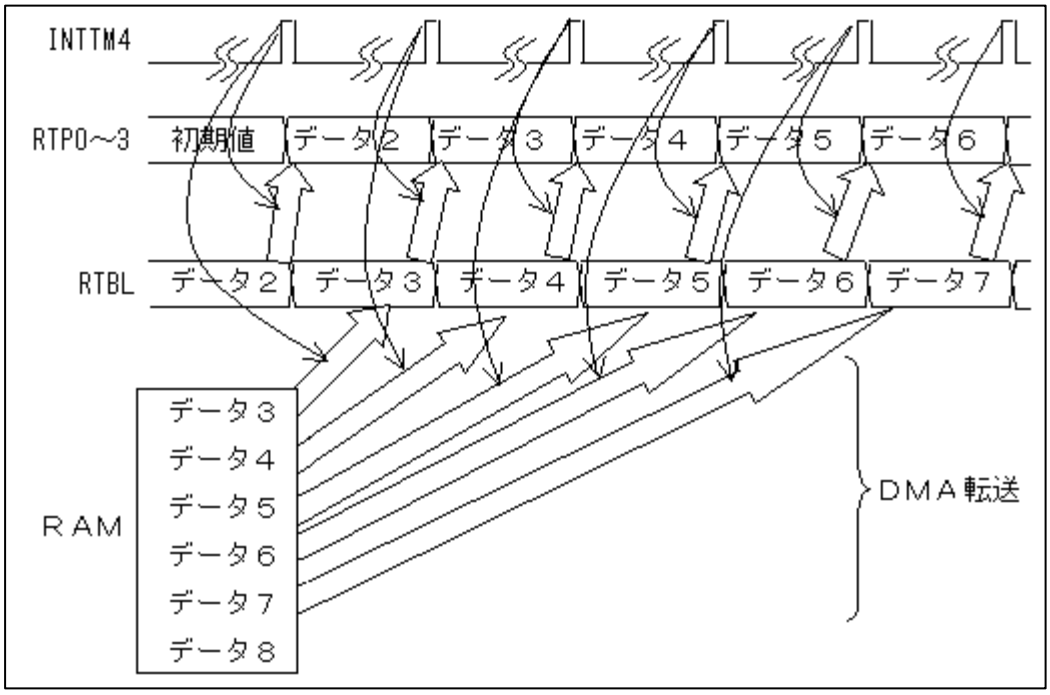
その後、次のように設定します。

RTPM　：0FH(下位 4 ビットをリアルタイム出力)を設定

RTPC　：80H(4 ビット 2 チャネル・モードで動作許可)を設定

RTBL　：次のデータを設定

(4) これで設定が完了しました。**TMC4** に 80H(**TM4** と **CR40** の一致でクリア＆スタート、タイマ出力は使用しない)を設定して、タイマ 4 をスタートします。これで、タイマ 4 で指定した間隔で、**RTP0～3** に出力パターンが出てきます。なお、初期値と次のデータは設定済みなので、DMA 転送するのは、その次のデータからです。タイミングを次に示します。



あとは、**INTDMA0**/1 の割り込みごとに、DMA の再設定と再起動を行うことで、連続した出力パターンを発生させることができます。

DMA の使い方(UART での最初のデータ送信) [V850/Sxx]

Q1. V850/SA1 の DMA で UART 送信をします。最初のデータ送信完了の割り込みが発生しても、DMA が起動しません。UART の送信が完了したあとで、DMA の設定を行っています。

A1. DMA の起動要因と、割り込み要求フラグの状態は全く別です。DMA の設定が完了してから、起動要因(送信完了)が入るようにしてください。

Q2. UART 送信で、最初のデータをプログラムで書き込まず、最初から DMA で転送できますか。

A2. できません。この DMA の起動要因は、UART の送信完了信号だけです。他の方法では、起動できません。

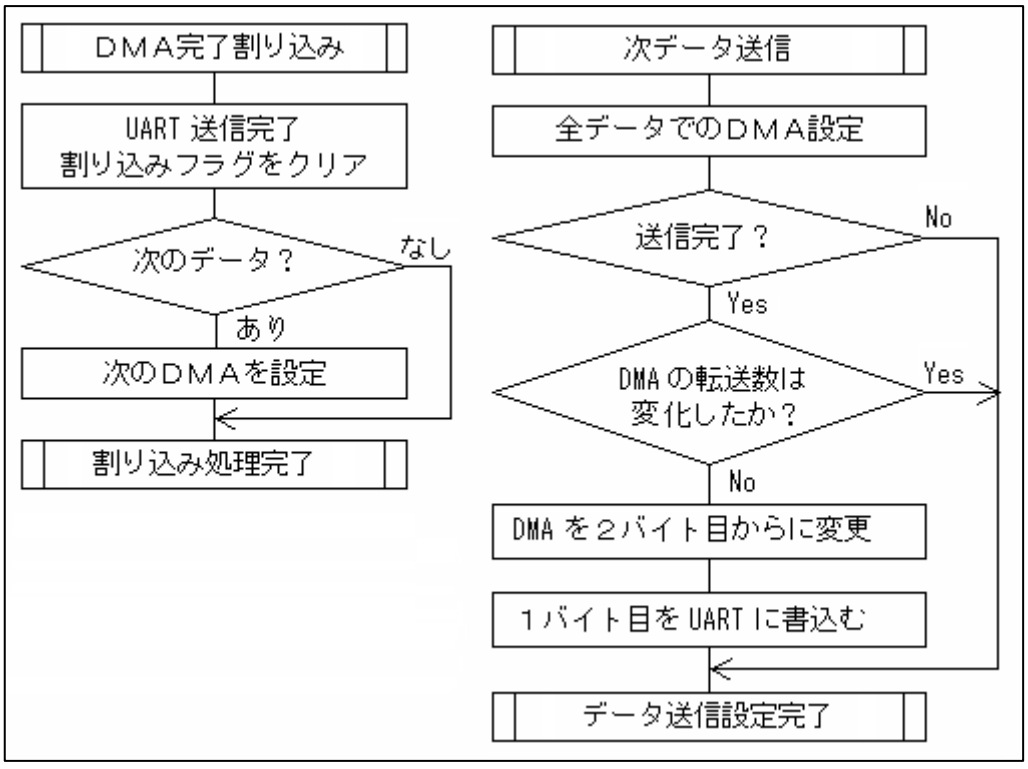
Q3. 一連のデータ送信が終わり、次のデータを送信する場合、最初のデータはプログラムで書き込むのですか。

A3. 前回のデータが送り終わっているかどうかで異なります。これを判断して DMA の設定を行う場合の処理は、次のようになります。ここでは、UART の転送速度が、割り込み応答に対して速くない場合の処理です。

最終データを UART に転送したら、DMA 完了の割り込みが発生するので、UART の送信完了割り込み要求フラグをクリアしておいてください。この段階で次のデータがあれば、DMA を再設定します。ここまでの処理は当然ながら、最後のデータが送信完了する前に行えるものとします。
この段階で次のデータがない場合、そのまま処理を完了します。
次のデータを送信するときに、最初から DMA を開始するように、DMA を設定してください。

DMA の設定が終わったら、UART の送信完了割り込みフラグが立っているか(前回の最後のデータの送信が完了したか)どうかを確認します。フラグが立っていなければ、前回の最後のデータの送信完了で、そのまま DMA は起動できるので、設定は完了です。DMA 完了を待ってください。
フラグが立っていれば、すでに送信は完了しているので、DMA バイト・カウント・レジスタを読み出し、設定値と同じかどうかをチェックします。値が変化していれば、すでに DMA が動作しているので、そのまま DMA の完了を待ってください。変化していなければ、DMA の設定が間に合わなかったことになるので、DMA 設定を変更(2 つ目のデータから送信するようにアドレスとカウント値を変更)して、その後、UART に最初のデータを書き込んでください。これで、最初のデータの送信完了後に、DMA が起動されます。

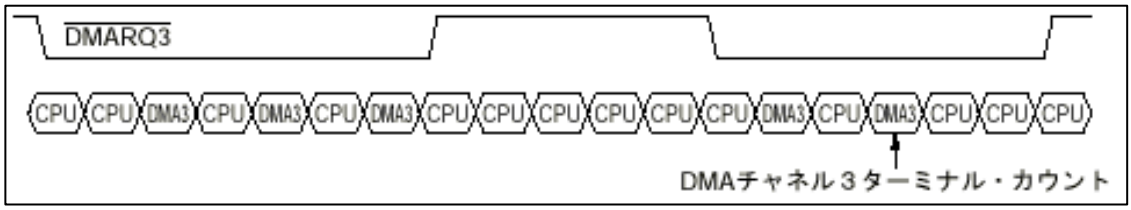
次に概略フローを示します。



DMA の転送モード [V850E/Sxx]

Q1. DMA の転送モードは、どういう動作ですか。

A1. V850/Sx の DMA モードは、V850E の「シングル転送」に相当します。
1 サイクルごとにバスを解放して、DMA 要求があるたびに、TC に至るまでデータ転送をします。



DMA 起動にソフトウエア・トリガ・ビットを使用 [V850E/MA1]

Q1. V850E/MA1 の DMA による UART の送信処理で、チャネル・コントロール・レジスタのソフトウエア・トリガ・ビットをセットすれば、データ送信のトリガとなりますか。また、このビットをクリアしておく必要はありますか。

A1. はい、DMA チャネル・コントロール・レジスタのソフトウエア・トリガ・ビットをセットすれば、UART の送信を起動させることができます。なお、単にトリガのためのビットなので、わざわざクリアする必要はありません。

DMA を繰り返し起動 [V850E]

Q1. DMA を同じ条件で繰り返したいのですが、1 回しか起動しません。

A1. DMA が開始するには、次の条件が必要です。

ENn ビット＝1

TCn ビット＝0

DMA 実行が終了すると、**TCn** ビットが 1 になりますので、この状態で DMA を起動しようとしてもできません。

DMA が終了したら、必ず DMA チャネル・コントロール・レジスタを読み出して、**TCn** ビットを 0 にしてください。

DMA 再起動で必要な設定 [V850E/MA1]

Q1. 同じ条件で DMA を繰り返す場合、なるべくレジスタの再設定を行わない方法がありますか。

A1. V850E シリーズでは、DMA ソース・アドレス・レジスタ、DMA デスティネーション・アドレス・レジスタ、DMA 転送カウント・レジスタは、マスタ・レジスタとスレーブ・レジスタの 2 段構成になっています。

通常の DMA では、スレーブ・レジスタの値を更新しながら動作していますが、ターミナル・カウントが発生すると、マスタ・レジスタの内容がスレーブ・レジスタに転送され、次の DMA 転送は、それ以前にマスタ・レジスタに設定されていた条件で行うことになります。

DMA チャネル・コントロール・レジスタの **MLEn** ビットをセットすれば、この機能を利用して、同じ条件での DMA を自動的に繰り返し実行させることができます。

これらのレジスタを前回と同じ条件で使用する場合、再度設定する必要はなく、DMA チャネル・コントロール・レジスタの再設定を行う(読み出して、**TCn** ビットをクリアしたあとで)だけです。

Q2. DMA 転送完了後、同じ設定で再度転送するには、どのレジスタを再設定すればいいですか。

A2. ソフトウエア・トリガで DMA を使うのなら、**DCHC** 以外は再設定の必要がありません。ユーザーズ・マニュアルの「6.9 ネクスト・アドレス設定機能」に記載されているように、**DSA**、**DDA**、**DBC** はマスタ・スレーブの構成になっています。DMA がターミナル・カウントに達すると、その前に設定していた値に自動的に戻ります。
DMA の条件を変更したい設定だけを変更してください。その後、**DCHC** を設定してください。

Q3. **DCHC** の TCn ビット・ターミナル・カウントは、再転送時、DCHC の設定直前に読み出す必要がありますか。

A3. はい、必要です。実際は、DMA が完了したあとになりますが、必ず **DCHC** レジスタを読み出してください。
DMA の開始条件に **TC**＝0 がありますので、次の DMA 開始までに、**DCHC** を読み出す必要があります。

DMA を一時的に停止 [V850E/MS1]

Q1. DMA 要求後、転送を一時停止させる場合は、**DCHC** レジスタの **ENn** ビットをリセットすればいいですか。

A1. はい、シングル転送モードなら、**ENn** ビットをリセットすれば、その後の DMA を停止させられます。

DMA でのミス・アライン [V850E/MA1、V850E/MA2、V850E/ME2]

Q1. V850E/MA1 で、16 ビット・バスの DMA 転送では、ソース・アドレスとデスティネーション・アドレスは、偶数アドレスだけが有効ですか。

A1. はい、そうです。ユーザーズ・マニュアルの「6.17 注意事項」の「(2) ミス・アライン・データの転送」に記載されているように、奇数アドレスを指定すると、A0 が強制的に 0 に設定されて、転送が行われます。

DMA でのエンディアン [V850E/MA1、V850E/MA2、V850E/ME2]

Q1. V850E/MA1 のユーザーズ・マニュアルに、リトル・エンディアンからビッグ・エンディアンへの DMA 転送が可能という記載があります。しかし、DMA の制御レジスタの中には、エンディアンの設定がありません。

A1. DMA 転送の対象メモリの CS 空間ごとに、**BEC** レジスタで設定されているエンディアンに従い、自動的に処理されます。

フライバイ転送でのバス幅 [V850E/MA、V850E/MS、V850E/ME]

Q1. V850E/MS1 のユーザーズ・マニュアル「6.7.1 の注意 2」にある「転送先と転送元のデータ・バス幅を同じにしてください」というのは、どのような意味ですか。

A1. フライバイ転送では、8 ビットと 16 ビットの間のデータ変換をしないことを意味します。プログラムによる CPU 転送を例に考えましょう。16 ビット幅での転送の場合、16 ビット幅の空間に対しては、16 ビットで1回のバス・サイクルでアクセスします(データは偶数アドレスにあるとします)。そのデータを 8 ビット幅の空間に対してアクセスする場合、2 回のバス・サイクルでのアクセスになります。フライバイ DMA 転送では、このようなことができないということです。
なお、V850E/MS1 では、2 サイクル転送でも、バス幅の異なる空間同士の DMA は保証されていませんが、V850E/MA や V850E/ME ではサポートされています。

フライバイ転送時のウエイト [V850E/MA、V850E/MS、V850E/ME]

Q1. 外部 I/O と外部 SRAM の DMA フライバイ転送時のウエイト挿入は、メモリ・ブロックが異なる場合、どちらの設定が有効ですか。

A1. フライバイ転送の場合、メモリ側のブロックの設定が有効です。

Q2. 上記 Q1 について、外部 I/O→SRAM、SRAM→外部 I/O についてどうなりますか。

A2. A1 の回答のように、転送方向によらず、メモリ(SRAM)側の設定に従います。

I^2C との DMA 転送 [V850/SB]

Q1. V850 の DMA 機能で、外部バスから(内蔵 RAM 経由で)I^2C バスへデータを転送することはできますか。

A1. 内蔵 RAM と I^2C バスの間の DMA 転送は可能です。ただし、外部バスから内蔵 RAM へは転送できません。また、I^2C バスの場合には、転送方向などの制御の問題(送信では、受信側からの ACK 確認で、転送を継続するか中止するかを判定)があり、DMA 転送はお勧めできません。

Q2. I^2C バスへの DMA 転送速度は、最大どの程度ですか。

A2. I^2C バスの転送速度が支配的なので、このような考察は不要です。DMA 転送は、I^2C バスよりはるかに高速です(4 クロックで 1 バイト転送可能で、転送ごとに CPU が動作するので、全体ではこの半分の転送速度)。

1.3.9 CAN

メッセージ送受信の完了の確認 [V850/SF1]

Q1. V850/SF1 の FCAN コントローラで、メッセージ送受信の完了時に、INTCTn/INTCRn の割り込みを発生させるには、CnIE の E_INT0 と E_INT1、さらに M_CTRLn の IE のすべてに、割り込み許可の設定が必要ですか。

A1. FCAN 部分からの割り込み要求(INTCTn、INTCRn)発生については、これらすべての設定が必要です。
さらに、割り込み制御レジスタ(CANIC3、CANIC2 など)のマスク(CANMK3、CANMK2 など)をクリアする必要があります。これで、CPU が割り込み許可状態になると、FCAN の送受信完了で INTCTn や INTCRn の割り込みが発生して、CPU が割り込みハンドラに処理を移します。

Q2. 割り込みを使わずに、送受信の完了を確認できますか。

A2. 上記 Q1、A1 の設定のうち、M_CTRLn の IE をクリアしたり、CANIC のマスクをセットしたりすると、割り込みを発生しません。CnINTP の CnINT1/0 ビットで、割り込み要求の保留を調べてください。いずれかのビットが立っていたら、それをクリアしてください。この場合、CnIE の E_INT0 と E_INT1 は、許可しておく必要があります。

内部レジスタのアドレスが複数ある理由 [V850/SF1]

Q1. V850/SF1 ユーザーズ・マニュアルで、内部レジスタのアドレスが xxnFF804H(n＝3,7,B)と表現されていますが、n の値は何を表すのですか。

A1. 単に、FCAN のレジスタが、アドレスとして n＝3, 7, B のアドレスに見えるというだけです。通常の使用では、n＝3 とお考えください。これは、FCAN のインタフェースが外部バスを経由して接続されており、外部アドレスとしてすべてが準備されていないために、イメージが見えるものです。

各レジスタの設定 [V850/SF1]

Q1. 次の条件で、各レジスタ(CGCS, CnBRP, CnSYNC)の設定について教えてください。

クロック　:16MHz
ボー・レート　:500kbps
サンプル・ポイント :75%程度

A1. (1) CGCS レジスタ

MCPn フラグで、CAN メモリ・アクセス・コントローラのベース・クロック f_{MEM} を設定します。たとえば、MCPn＝0(f_{MEM}＝fxx)で 16MHz を入力すれば、f_{MEM}＝16MHz となります(fxx:メイン・クロック周波数)。

f_{MEM} は、ボー・レート・ジェネレータ部とグローバル・タイマ・システム・クロック生成部に供給されます。CAN の基本ボー・レート設定に関係するのは、前者のボー・レート・ジェネレータ部です。後者は、タイム・スタンプ機能という、CAN プロトコルを利用したオプション機能です。

(2) CnBRP レジスタ

CAN ビット・レート・プリスケーラ出力の CAN モジュール・クロック f_{BTL} を設定します。

このレジスタで、バスのタイプ(ハイ・スピード／ロウ・スピード)と、プロトコルで定義される TQ(Time Quanta)の設定をします。500kbps の場合、データ・ビット幅を 8TQ とします。よって、1TQ＝500kHz×8=4MHz=f_{BTL} です。

上記 MCPn の設定で、16MHz をベース・クロックとして供給としたとき、BRPn＝1(f_{MEM}/4)を設定すれば、4MHz になります。同時に BTYPE＝1 を設定して、ハイ・スピード・モード(＞125kbps)にしておきます。

(3) CnSYNC レジスタ

ビット時間や SP(Sample Point)、SJW(同期ジャンプ幅)を設定して、CAN グローバル周波数を定義します。

500kbps の場合、(2)でデータ・ビット幅を 8TQ としましたので、DBTn＝7(f_{BTL}×8)となります。

SP＝75%であれば、6TQ ですので、SPTn＝5(f_{BTL}×6)を設定します。

なお、プロトコル上、SJW は 4TQ まで設定できますが、この例では(DBT－SPT)＝2 なので、上限は 1 です。

バス・オフの発生と検出 [V850/SF1]

Q1. CAN で、バス・オフの状態を検出するには、どうすればいいですか。

A1. バス・オフの発生には、トランシーバなどとの接続を切り離し、マイコンの CANRX 端子をハイ・レベルにプルアップして送信する方法があります。検出は、割り込みハンドラ(INTCE1/2)処理ルーチン内で、CANn コントロール・レジスタのバス・オフ・フラグ(BOFF)をチェックすればできます。

受信メッセージが抜ける [V850/SF1]

Q1. 一定周期で非同期に送受信を行い、受信前に相手の送信が連続すると、受信メッセージが抜けます。

A1. 受信メッセージ・バッファへのオーバライトが発生していると思われます。受信処理で DN ビットをクリア後に、メッセージ受信があると、M_STATn レジスタの DN ビットがセットされて、オーバライトが発生します。これは、M_CTRLn レジスタの MOVR ビットで確認できます。

受信処理の最後に DN ビットのチェックを行い、0 でなければ再度受信データを取得するようにしてください。

また、CnCTRL レジスタの OVM ビットを 0 に設定して、新規メッセージのオーバライトを許可してください。

★

MASK を有効にする設定 [V850ES/Fx3]

Q1. CAN を受信させるチャンネルの MASK レジスタを次のように設定しても、ID レジスタで設定した ID しか受信しません。MASK を有効にするには、どのように設定すればいいですか。

C0MASK1L : 0xffff

C0MASK1H : 0x1fff

A1. 対象メッセージ・バッファの C0MCONF レジスタに、0x11 を設定してください。

1.3.10 IEBus

マスタ送信要求 [V850/SB2]

Q1. IEBus で、マスタ送信の要求中(DR、MSTRQ などセット)に、スレーブ受信要求(SLVRQ セット)を受けた場合、スレーブ受信開始の処理をしてから、マスタ送信要求を続行してもいいですか。

A1. はい、スレーブ受信処理をしてから、マスタ送信要求を行ってください。ただし、競合での待機後、61.7 μs(強制リセット期間)は、MSTRQ のセットができません(マニュアル p.463 の注意 1 参照)。

スレーブ受信時、受信を行わず、送信相手に再送信を行わせる [V850/SB2]

Q1. スレーブ受信時、受信をせず(DMA 転送で RAM を上書きせず)に、相手に再送信をさせたい場合、INTIE2, STARTF, SLVRQ の割り込み時に、なにも処理をしないでいいですか。

A1. ISR フラグのクリアだけは、必ず行ってください。

オーバラン時のエラーの DR 読み込み [V850/SB2]

Q1. オーバラン発生時に、同報通信で IEERR 割り込み、または個別通信で ENDFRAM 割り込みが起きたとき、DR を空読みするなら、char dummy = DR でいいですか。

A1. はい、そのフレームを破棄するなら、ダミー読み込みで結構です。

Q2. 上記 Q1 場合、マスタ送信側では、個別通信なら ENDFRAM 割り込みを受けて再送信処理を行い、同報通信ならあきらめるという処理でいいですか。

A2. はい、そのようになります。

INTIE2 割り込み処理後、マスタ要求が通らないと ISR を消去 [V850/SB2]

Q1. INTIE2 割り込みの処理後、マスタ要求が通らなかったとき、ISR レジスタを全消去したいのですが、 STARTF や ENDTRNS フラグは、ENSLVRX フラグなどを立てることで消去すればいいですか。

A1. STARTF は ISR へのバイト書き込み、ENDTRNS は ENSLVRX フラグのセットなどでクリアしてください。

Q2. マスタ要求が通らなかった場合、ISR レジスタを消去しないと、STARTF と ENDTRNS が同時に立ちますか。

A2. はい、その可能性がありますし、そもそも ISR が 0 でない状態では、次の INTIE2 が発生しません。

スレーブ受信時、受信データの送り先アドレスを知る方法 [V850/SB2]

Q1. スレーブ受信時、受信データの送り先アドレスは分かりますか。
また同報受信時、全同報かグループ同報かを区別できますか。

A1. スレーブ受信時は、マスタのアドレスは分かりますが、それ以外のアドレスは分かりません。
また、同報受信か個別受信かは区別できますが、それ以上の区別はできません。

IEBus コントローラの「簡易版」

Q1. IEBus コントローラが「簡易版」と表記されていますが、何が違うのですか。

A1. 完全版の IEBus コントローラは、送受信バッファが 32 バイト分ありますが、簡易版は 1 バイト分だけです。
また外部には、送受信データ・ラインに差動ドライバが必要です（当社では製品化していません）。

1.3.11 実行時間

各命令の所要クロックの記述場所 [V853、V850/S]

Q1. 各命令の所要クロックは、どのマニュアルに記述されているのですか。

A1. 命令の実行時間は、「V850 ファミリ アーキテクチャ編」のマニュアルに記載されています。なお、実行時間は、パイプラインの状態によって異なります。

パイプラインと命令実行速度 [V850E/IA1]

Q1. 命令フェッチとメモリ・アクセスにかかる時間を教えてください。

A1. V850E/IA1 のメモリ・アクセス時間は、ハードウエア編マニュアルの「4.5.1 アクセス・クロック数」で記載されています。外部メモリ・アクセスは、最低 3 クロックです。また、データ・アクセスと競合した場合、内蔵 RAM からのフェッチは 2 クロックで、内蔵 ROM からのデータ・アクセスは 5 クロックです。

Q2. MEM ステージで内蔵 RAM へ書き戻す場合、命令の実行サイクルは何クロックですか。

A2. 実行サイクルは、次のようになります。

IF(下位)＋IF(上位)＋ID＋EX1＋MEM

外部からの命令フェッチ（IF）については、バスの空き時間に先読みキューへ取り込みますので、単純に外部メモリのアクセス時間（最低 3 クロックずつ）とはなりません。
また、この先読みキューから CPU のパイプラインに転送されるので、IF-ID も 3 クロックずつにはなりません。
なお、SET1 や CLR1 命令は、EX1-MEM-EX2-MEM と続くので、その分は長くなります。
これらを考慮すると、最低で 7 クロック以上となります(WB を含む)。
ただし、先読みキューが完全に埋まるとも考えられないので、さらに 2、3 クロックかかると思われます。
なお、これらの動きは前の命令に依存しますので、何クロックと規定することはできません。

ビット操作命令の実行時間 [V850/S、V850E]

Q1. ビット操作命令は、リード・モディファイ・ライト動作となりますか。また、その実行中は割り込み可能ですか。

A1. はい、リード・モディファイ・ライト動作となります。このため、ビット操作命令の実行では、他の命令より余分にクロック数を必要とします。また、次の命令の IF と ID の間に、アイドル・サイクル(V850/SA1 は 3 クロック、V850E/MS1 は 2 クロック)が挿入されます。

詳細な動作については、アーキテクチャ編のマニュアルの「第 8 章　パイプライン」でビット操作命令をご参照ください。なお、リード・モディファイ・ライトの間に割り込みが入ることはありません。

内蔵周辺アクセスでの実行速度 [共通]

「その他周辺機能：システム・ウエイト・コントロール・レジスタ(VSWC) [V850E,V850ES 共通]」もご参照ください。

Q1. P1 端子へパルスを出力するのに約 200ns かかります。ユーザーズ・マニュアルには「ST.B 命令は 1 クロックで処理が行われる」とあり、40ns 程度ではないですか。

```
MOV     FF,r10
ST.B    r10,-BFE[r0]
MOV     0,r10
ST.B    r10,-BFE[r0]
MOV     FF,r10
ST.B    r10,-BFE[r0]
```

A1. これは、内蔵周辺(P1)のアクセス時間が、最低 7 クロックは必要となるためです。このアクセス時間の分だけ実行時間が延びます。内蔵周辺をアクセスしなければ、このように時間がかかることはありません。

Q2. I/O アクセスには何クロックかかりますか。

A2. I/O ポート(周辺 I/O レジスタ)に対するアクセス自体は、**VSWC** レジスタの設定で決まり、50MHz での推奨値 12H を使用すると、6 クロック必要になります。

ビット操作命令ではリード・モディファイ・ライトとなるので、アクセスは 13 クロック(アクセス 2 回＋演算)必要となります。

Q3. 出力ポートに「1, 0」を順次ライトしたときのサイクル数を教えてください。

A3. 連続した命令で、単純に「0, 1」を出力すると、12 クロックの周期となります。

また、ビット操作の連続では、26 クロックの周期となります。これは、ポートのリードが必要になるためです。

Q4. 次のプログラムで 20MHz 動作をさせると、P20 の反転が 50ns ごとでなく、400ns ごとになります。

逆アセンブルの結果、「P2.0 = ~P2.0」は1つの NOT1 命令に変換されて、余分な実行はしていません。

```
while(1)
{
    P2.0 = ~P2.0;
        :
        :
    P2.0 = ~P2.0;
}
```

A4. 2 つの要因があります。まず、NOT1 命令自体が、ポート 2 に対してリード・モディファイ・ライトの 3 つの動作の組み合わせで実行されます。次に、ポート 2 に対するアクセスは、3 クロック必要です。

備考 ほとんどの V850 シリーズでは、マニュアルの「バス・アクセス」の最初に、内蔵メモリのアクセス速度(クロック数)が記載されています。デバイスによっては、内蔵周辺へのアクセス速度が記載されているものもあります。また、内蔵周辺のアクセスにウエイトを指定する VSWC レジスタを内蔵しているデバイスがあり、この場合には、動作周波数に応じたウエイト数を設定する必要があります。

最も高速処理が可能なプログラム・エリア [共通]

Q1. 最も高速にプログラム処理ができるのは、どのエリアですか。

A1. 内部 ROM(フラッシュを含む)にプログラムを配置したときです。内部メモリでは、最少 1 クロックで 32 ビットの命令をフェッチできます。外部からのフェッチは 16 ビット幅で、しかも 1 クロックではフェッチできません。

備考 V850E/ME2 には内部 ROM(フラッシュを含む)はありませんが、大容量の RAM を内蔵しており、そこからは 1 クロックで 32 ビットの命令をフェッチできます。また、外部バスの幅も 32 ビットです。

ほとんどの命令を 1 クロックで実行できるか [共通]

Q1. V850E/MA1 で、プログラム動作の時間を測定しました。マニュアルには、「パイプライン制御によりほとんどの命令を 1 クロックで実行できる」と記述されていますが、結果は記載されている時間と異なります。

A1. 内蔵 ROM から命令をフェッチして、パイプラインのハザードがなく動作している場合は、ほとんどの命令は 1 クロックで実行可能です。しかし、外部メモリからのフェッチでは、1 ワード(4 バイト)の命令をフェッチするのに、最低でも 4 クロック必要となります(実際には、メモリのアクセス・タイムの関係で、さらに長くなります)。これだけで、命令の実行は 4 倍以上の時間となります。
また、全命令の実行時間が 1 クロックではなく、2 クロック必要なものもあります。その実行結果を次の命令で参照するような場合(「mov 3, r10」に「st.b r10, 11[r29]」が続くようなとき)には、実行クロックが余分に必要になります。このような遅延要因が入り交じっていることから、命令の実行時間が長くなります。
これらの動きの詳細は、アーキテクチャ編マニュアルの「5.4 命令実行クロック数」をご参照ください。
なお、パイプラインの動きを完全に把握することはかなり困難です。

1.3.12 プログラミング

MOVEA 命令の符号拡張

Q1. 0x718002 番地のワード・データと 0x718000 を加算したアドレスに分岐したいのですが、MOVEA 命令の実行後、r25 の内容が 0x718000 でなく 0x708000 となってしまいます。

A1. 16 ビットのイミディエート値を符号拡張して、32 ビットで計算しているためです。つまり、MOVEA で 0x8000 を符号拡張すると、MSB が 1 なので 0xFFFF8000 となります。これを加算するので、0x708000 となります。
この影響(上位 16 ビットがマイナス 1 になる)を避けるには、あらかじめセットしておく上位の値を 0x0071 ではなく、0x0072 としてください。

リセット・スタート後、最初に行うこと [共通]

Q1. V850 のプログラムで、やりたい処理以外に、どのような処理を最初に行う必要がありますか。

A1. お客様のプログラムを実行させる前に、次のような処理が必要になります。

(1) デバイスとして必要な、共通的な初期設定

- VSWC レジスタによる内蔵周辺アクセス時間の設定
- 動作クロック(PLL モード／ダイレク・モードと分周比)の設定
- スタンバイ関係の設定
- ウォッチドッグ・タイマの設定

(2) お客様の使用方法に合わせた、内蔵周辺 I/O の初期設定

- ポートの初期設定
- 使用したい内蔵周辺 I/O に関連した、兼用ポートの初期設定(具体的な内容は、マニュアルのポートの章で、兼用機能を使用する場合の設定をご参照ください)
- 使用したい内蔵周辺 I/O の初期設定

Q2. マイコンはリセット・ハンドラから実行しますが、main プログラムの実行まで、どのようにつながるのですか。

A2. リセット・ハンドラから main プログラムまでの部分は、「**スタートアップ・ファイル**」が処理します。具体的には、リセット・ハンドラでスタートアップ・ルーチンへの分岐命令が生成されて、最初にスタートアップ・ルーチンが動作します。そこでは、各種変数などの設定や定義が行われます。変数などの初期化が完了すると、ユーザ・プログラムの main 関数をサブルーチン・コールすることで、main プログラムの実行を行います。
通常、起動して最初に実行させる処理(前述 **A1** の(1)の部分)は、main 関数のサブルーチン・コールの前に記述しておきます。

1.3.13 メモリ

外部 8 ビット・データ・バスで、偶数番地も奇数番地も下位 8 ビットでアクセス

Q1. 外部に 8 ビットのデータ・バスしかない製品で、偶数番地も奇数番地も下位 8 ビットでアクセスできますか。

A1. できますが、方法は次の 2 つに分かれます。

(1) V850E シリーズ (V850E/MS、V850E/MA、V850E/IA など)

これらのデバイスには **BSC** レジスタが準備されており、その設定により各メモリ・ブロックに対するデータ・バスの幅を 8/16 ビットのどちらかに指定できます。8 ビットに設定すれば、そのメモリ・ブロックに対しては、データ・バスの下位 8 ビットを使用したアクセスとなります。

(2) V850/S シリーズ (V850/SA、V850/SB、V850/SF、V850/SV など)

これらデバイスのデータ・バスは、16 ビット幅に固定です。設定だけで 8 ビットにはできません。
8 ビットのデバイスを接続した場合、外部にバス切り替え回路を追加して、奇数番地へのアクセスでは、8 ビットのデータを上位 8 ビット側に切り替えてください。具体的には、**UBEN** と **LBEN** を用います。
UBEN のアクティブ時は、外部デバイスと AD8-15 の間のバッファを選択し、**LBEN** のアクティブ時は、外部デバイスと AD0-7 の間のバッファを選択してください。
なお、当然ながら 16 ビットでのアクセスはできませんので、バイト・アクセスだけにしてください。

外部拡張メモリへのバイト・アクセスを行う場合の注意 [V850/SA1]

Q1. 外部拡張機能を使用して、1バイト幅の外部拡張メモリへバイト・アクセスをする場合の、周辺デバイスとの接続方法を教えてください。

A1. 外部拡張メモリに対するアクセスが、8 ビット固定であれば、そのまま接続できます。
しかし、16 ビット・アクセスが発生すると、正しく動作しませんのでご注意ください。
命令による 16 ビット・アクセス以外に、命令フェッチやスタック動作が、これらの外部拡張メモリに対して発生しないように管理する必要があります。

内蔵 ROM を無効にして外部 ROM で使用 [V850/SA1]

Q1. 内蔵のマスク ROM を無効にして、外部の ROM で動作させることができますか。

A1. いいえ、できません。この製品は、携帯機器を低消費電力で実現することを目的に開発しました。
消費電力を増やしてしまいますので、外部に ROM を付加することができない設計となっています。

メモリ拡張で MM を設定する場合の、ポート 9 の入出力モード [V850/SA1]

Q1. ユーザーズ・マニュアルで、MM(メモリ拡張モード・レジスタ)の設定に「ポート 9 の P93, P94 ビットを 1 に設定してください」とありますが、ポート 9 は、出力か入力のどちらに設定すればいいのですか。

A1. どちらでもかまいません。
MM レジスタを設定することで、入出力は自動的に決まります。したがって、メモリ拡張モードを使用する場合には、モード設定レジスタ P9 は入力／出力どちらの設定でもけっこうです。

SDRAM:アクセス・マルチプレクス幅の設定 [V850E/MA1]

Q1. V850/MA1 で、SDRAM コンフィギュレーションのアクセス・マルチプレクス幅の設定方法を教えてください。
SDRAM の構成は、次のとおりです。

ロウ・アドレス幅	:11
カラム・アドレス幅	:8
SDRAM 容量	:512K × 16 ビット × 2 バンク
CPU の動作周波数	:50MHz

A1. SDRAM を接続するブロックに対応した SCRn レジスタで、次のように設定してください。

SSO1n, SSO0n	:01 （16 ビット）
RAW1n, RAW0n	:00 （11 ビット）
SAW1n, SAW0n	:00 （8 ビット）

SDRAM：バンク信号に接続するアドレス信号 [V850E/MA1]

Q1. 128M ビットか 256M ビットの SDRAM を使用する場合、バンク信号にどのアドレス信号を接続するのですか。

A1. SDRAM のバンク構成に依存します。ユーザーズ・マニュアルに記載の 64M(1M ワード×16 ビットが 4 バンク)の構成では、1 バンクが 1M ワード=2M バイトとなります。このため、この容量をアクセスするには、アドレスが 21 本(A0-A20)必要になり、そのすぐ上位アドレス信号が、バンクの指定に使用されます。つまり、A21 と A22 で、4 つのバンクを切り替えることになります。

この考え方は、使用する SDRAM が変わっても同じで、1 バンクの容量が 4M バイトなら、A22 以降がバンク指定に使用されます。

なお、V850E/MA1 では、1 バンクの容量は最大 4M バイトまでしか対応していません。

128M や 256M の場合でも、この範囲である必要があります。

SDRAM：直接制御できる SDRAM 容量の最大値 [850E/MA1]

Q1. V850E/MA1 が直接制御できる SDRAM 容量の最大値はどれだけですか。

A1. デバイスとしては、全空間を使用すれば、256M バイトから内部メモリや内部周辺 I/O を除いた分ですが、1 本の CS 信号に接続できる容量が制限されるので、ここまで直接は制御できません。

バンクあたり最大 4M バイトまでアクセスできます。つまり、メモリのバンクが 4 つなら 16M バイト、8 つなら 32M バイトとなります。これが最大で 4 つ分ということになります。

このように、使用する SDRAM により容量は異なりますので、使用される SDRAM に合わせてお考えください。

SDRAM：オート・プリチャージ時の AP 設定 [V850E/MA1]

Q1. バンクを V850 の A11 で制御し、オート・プリチャージとする場合、SDRAM はどのように接続するのですか。

A1. SDRAM にもよりますが、次のようになります。

	V850	SDRAM
アドレス	A[10:1]	A[9:0]
プリチャージ選択	A11	A10/AP
BANK 選択	A20	BA

SDRAM：書き込みができない [V850E/MA1]

Q1. V850E/MA1 で、SDRAM(1M×16 ビット)に書き込みができません。接続方法は、ユーザーズ・マニュアル記載のとおりです。

・CBR リフレッシュを許可すると、アドレス、データ、その他の信号が出力されません。

・CBR リフレッシュを禁止すると、初期化後の最初の書き込みだけ正常に各種信号（DQM、RAS、CAS など）が出力されます。読み出したデータは、すべて 0xff になっています。

A1. PMCCD を設定(SDCLK 出力モード、SDCKE 出力モード)されていますか。また、SELFREF 端子が H レベルになっていると、CPU は、SDRAM がセルフ・リフレッシュ状態にあると認識します。

SDRAM：TWPRE、TWE の意味 [V850E/MA1]

Q1. ユーザーズ・マニュアルで SDRAM のライト・タイミングにある TWPRE，TWE の意味を教えてください。
また、これらはライト・アクセスで、必ず 1 クロック分ずつあるのですか。

A1. ライトのプリチャージと終了サイクルを表しており、必ず 1 クロック分ずつ付きます。

ROM レス・モード [V850E/MA1]

Q1. μPD703106 の MODE(0-2)端子を使用して、ROM レス・モードに設定できますか。

A1. はい、できます。詳細な設定は、ユーザーズ・マニュアル「3.3.2 動作モード指定」をご参照ください。

周辺 I/O レジスタのアドレス [V850E/MS1]

Q1. V850E/MS1 のデータ空間は、ユーザーズ・マニュアルの「3.4.4 メモリ。マップ」では X0000000H-X3FFFFFFH ですが、「3.4.8 周辺 I/O レジスタ」では FFFFFxxxH になっています。メモリ空間は、どうなっていますか。

A1. 基本的に同じアドレスを示しています。V850E/MS1 の論理的なメモリ空間は 4G バイトですが、すべて独立して存在するのではありません。実際にアクセスできるのは、64M バイトの物理空間だけです。
4G バイトの論理空間では、この 64M バイトの物理空間のイメージが、64 個並んでいるように見えます。
したがって物理的には、X3FFF000-X3FFFFFFH と FFFFF000-FFFFFFFFH のアドレスは全く同じになります。
プログラム・カウンタは 26 ビットしかありませんので、これを使うと p.74 のように-X3FFFFFFH と表せます。
特に p.74 の図は、各モードでのメモリ・マップの違いを表していますので、ブート領域である 0 番地部分の違いを明確にするために、このようになっています。
一方、レジスタは 32 ビットですので、レジスタを用いてデータ空間をアドレシングするときには、4G 空間のどこでもアドレスできますが、内蔵 ROM を含めて LD/ST disp16 命令でのアクセスを考えると、0 番地を中心にして 32K バイトの範囲に収まるように考えた方が都合がいいので、FFFFF000 以降のアドレスとしています。

16 ビット・アクセス時の A0 出力 [V850E/MS1]

Q1. 16 ビット・モードのリード／ライト時、A0 出力はどのようになりますか。

A1. ユーザーズ・マニュアルの「4.5.3 バス幅」に記載のように、アクセスするアドレスに応じて変化します。

- 偶数番地への 8/16 ビット・アクセス　：A0=0
- 奇数番地への 8 ビット・アクセス　：A0=1
- 奇数番地への 16 ビット・アクセス　：1 回目は A0=1、2 回目は A0=0（各 8 ビット・アクセス）

DRAM：同ページ内アクセスではカラム・アドレスだけ出力 [V850E/MS1]

Q1. ロウ・アドレスが変化しない DRAM アクセスの場合、DRAM コントローラは、カラム・アドレスだけを出力するのですか。

A1. はい、V850E/MS1 では、メモリ・アクセスの高速化のために、DRAM のページ機能をサポートしています。
2 サイクル目のアクセスがページ内の場合には、カラム・アドレスだけを出力するようになっています。

DRAM:プログラマブル・ウエイト [V850E/MS1]

Q1. μPD703100A の CS3 と CS4 について、2 ウエイトしか入りません。

A1. DRAM の場合、**DWC1, 2** ではウエイト制御ができません。DWC は、SRAM や ROM などに対するウエイト時間を制御するレジスタで、DRAM については **DRC0-3** レジスタで制御します。**DRC0-3** のどれかに、3 種類のウエイト時間を設定しておいて、**DTC** レジスタで、各ブロックと **DRC** の対応を指定してください。
SRAM の場合には、**BCT** レジスタの設定が正しいか(該当するビットが 00 になっているか)どうかをご確認ください。

μPD703100 のアイドル状態 [V850E/MS1]

Q1. アイドル・ステート TI を挿入すると、CS 出力はハイ・レベルに戻りますか。

A1. はい、TI サイクルでは、必ず CS 出力はハイ・レベルになります。

プログラム・ウエイトの再設定 [V850E/MS1]

Q1. 電源投入後にプログラム・ウエイトを設定していますが、アプリケーションに処理が移ったあと、プログラム・ウエイトの変更は可能ですか。

A1. いいえ、**DWC1, 2** レジスタ(プログラム・ウエイト)の設定は、リセット後だけに行って、その後は値の変更をしないでください。

外部メモリ拡張容量 [V853]

Q1. アドレス・バスが A0～A19 なので、ハーフワード・アクセスなら、1M ワード(2M バイト)まで拡張できますか。

A1. いいえ、違います。V853 では、外部に拡張できるのは最大 1M バイトです。
アドレスは 20 本で、データ・バスは 16 ビットですが、A0 は上位バイト/下位バイトの切り替えに使用します。したがって、最大で 1M バイトになります。
ただし、実際に外部のデバイスをアクセスするには、**/LBEN** や **/UBEN** 信号を用います。

8 ビット幅と 16 ビット幅の外部メモリの接続 [V853]

Q1. 外部接続する SRAM のデータ・バス幅が 8 ビットと 16 ビットの 2 種類ある場合、接続方法を教えてください。

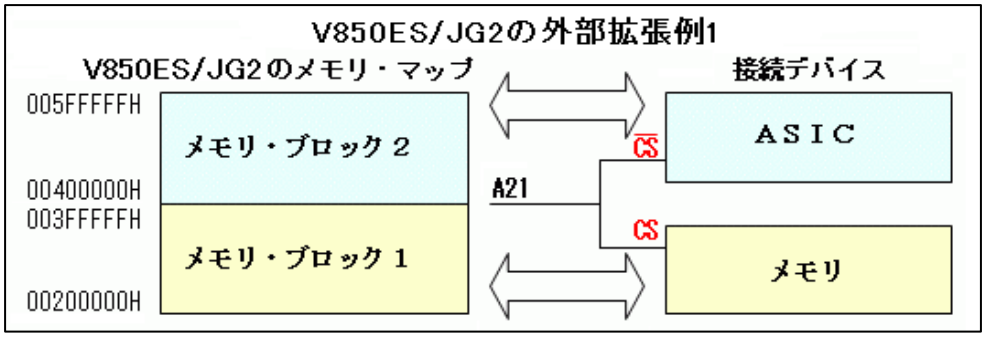
A1. まず、16 ビット SRAM ですが、これにはバイトごと(上位と下位 8 ビット)の WE 信号(WRL, WRH)を使い分ける(SRAM 側にこれらに対応した WE 信号も)必要があります。この信号がないと、CPU からの 8 ビットでの書き込みで、関係ない方の 8 ビットにも不定のデータが書き込まれてしまいます。
次に、8 ビット SRAM ですが、これだけでは偶数番地のデータだけで、奇数番地のデータを処理できません。CPU は、D8-D15 で奇数番地のデータをアクセスします。このため、奇数番地の場合には、SRAM のデータ・バスを CPU の D8-D15 に接続する切り替え回路が必要です。このとき、必ず 8 ビットでアクセスする必要があります。バス・アクセスの際のデータ・バスの動作は、マニュアルの「4.3.2 バス幅」をご参照ください。

外部メモリの接続 [ほとんどの V850ES に共通]

Q1. V850ES/JG2 で外部メモリと ASIC を制御したいのですが、CS 端子がありません。どうすれば、メモリと ASIC を接続できますか。外部でアドレスをデコードしないといけませんか。

A1. ファミリ製品である 144 ピンの V850ES/JJ2 には $\overline{CS}$ 端子がありますが、100 ピンの V850ES/JG2 は端子数が少ないので $\overline{CS}$ 信号をサポートしていません。基本的には、使用するマイコンを V850ES/JJ2 に変更するか、外部にアドレス・デコーダを追加して、必要な $\overline{CS}$ 信号を生成することになります。

接続するデバイスが、メモリと ASIC の各 1 個なら、アドレス・デコーダを使用しなくても接続できます。たとえば次の例のように、メモリ・ブロック 1 の 2M バイトをメモリの接続用に使用し、メモリ・ブロック 2 の 4M バイトの下位 2M バイト（00400000H～005FFFFFH）を ASIC の接続に使用することを考えます。



この場合には、A21 を $\overline{CS}$ 信号として利用できます。この例では、メモリの場合に A21 が"1"に、ASIC の場合に A21 が"0"になっています。つまり、A21 は次のような信号となります。

ASIC の $\overline{CS}$ 信号

メモリの $\overline{CS}$ 信号

もし、メモリのチップ・セレクトがアクティブ・ロウなら、接続を逆にすれば対応できます（ASIC のチップ・セレクトがアクティブ・ハイの必要がある）。

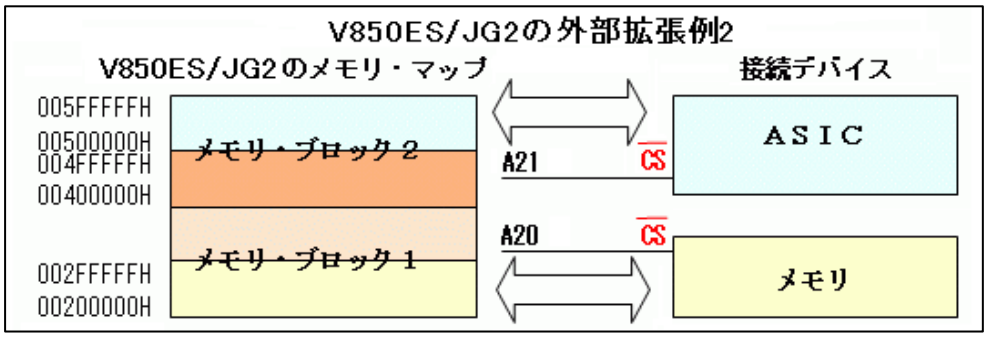
メモリと ASIC のチップ・セレクトがともにアクティブ・ロウの場合は、メモリ・ブロック 1 に接続するデバイスでは A21 を反転させてください。

もし、メモリや ASIC に必要なアドレスが各 1M バイト以下なら、A20 も $\overline{CS}$ 信号として利用することが考えられます。ここで、メモリと ASIC は、次のように割り付けることになります。

メモリ　：00200000H～002FFFFFH（**A20=0,A21=1**）

ASIC　：00500000H～005FFFFFH（**A20=1,A21=0**）

次図にそのメモリ・マップを示します。



このように異なるアドレス信号を $\overline{CS}$ 信号として使用する場合、アクセスするアドレスに注意が必要です。00400000H～004FFFFFH の領域は絶対にアクセスしないでください（それ以外でも **A20=0,A21=0** となるアドレスは使えません）。

1.3.14 ROM コレクション

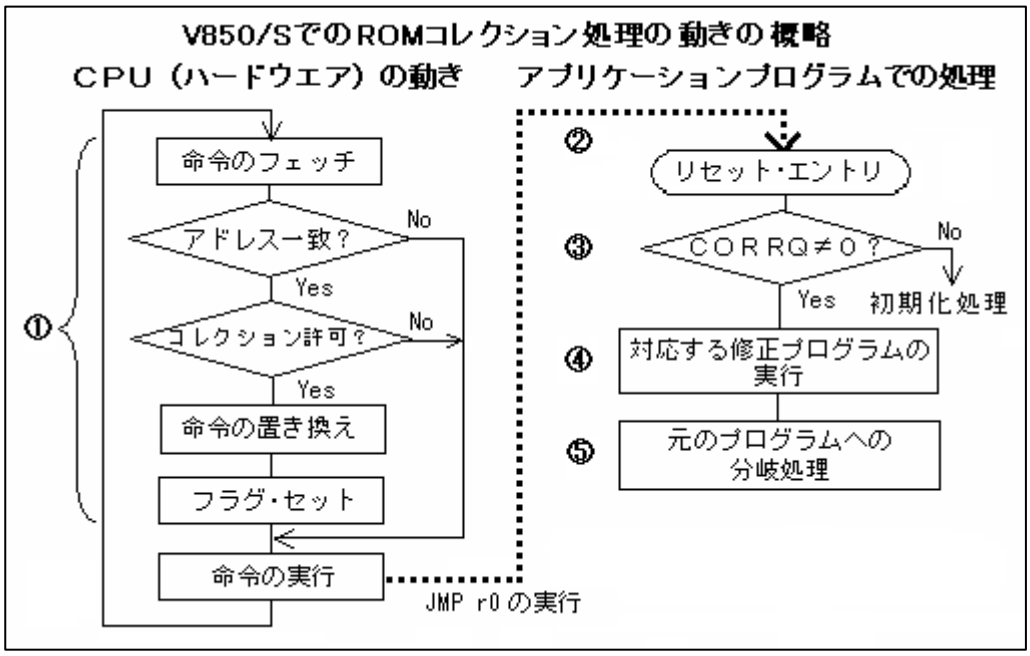
ROM コレクション [V850/S]

Q1. V850/SB1 の ROM コレクションは、どういう機能ですか。

A1. マイコン内の ROM に書き込まれたプログラムにパッチを当て、プログラムを修正する手段に ROM コレクション機能があります。既存のプログラムを修正できるという便利な機能ですが、いつでも無条件で使用できるのではありません。この機能を使用するには、事前にそのための処理の一部をプログラム中に埋め込んでおく必要がありますし、代わりに実行させるプログラムを読み込むための、外部メモリも必要になります。
ROM コレクションの動作や、そのために必要なことなどについて、以下に説明します。

(1) ROM コレクションの基本的な動作

ROM コレクションでは、CPU (ハードウエア)が自動的に実行する部分と、アプリケーション側で対応をすべき部分が混在していて、これらを区別する必要があります。すでに、必要な設定が行われた状態での、ROM コレクションの動きを次に示します。



①プログラム実行中、CPU が命令のフェッチ時に、フェッチ・アドレスと、CORADn レジスタに設定されている ROM コレクション・アドレスを比較して、一致するとフェッチ命令を分岐命令(JMP r0)に置き換えます。
②置き換えられた分岐命令を実行し、それまでの処理を中断して 0 番地に分岐します。このとき、一致したコレクション・アドレスに対応する CORRQ レジスタのビット(CORRQ0-3)をセットします。
ここまでが、ROM コレクションのハードウエアで処理される内容で、これ以降は、アプリケーション・プログラムで処理する内容です。
③JMP r0 命令で分岐するので、通常のリセット時のエントリからプログラムを実行することになります。このため、最初にリセットか ROM コレクションかを CORRQ レジスタで判定し、リセットなら初期化処理を行います。
④通常、ROM コレクションでは複数箇所に設定できるので、 どのアドレスでの一致による処理かを CORRQ0-3 ビットで判断して、対応したプログラム(パッチ処理)を実行します。
⑤パッチ処理が完了したら、元のプログラムに分岐します。
以上のように、ROM コレクション用のハードウエアと、事前にアプリケーション・プログラムに組み込んである処理プログラムを用いて、修正用プログラムを実行させることで、パッチ処理を行えるようになっています。

(2) ROM コレクションを行うために準備するハードウエア

ROM コレクションを行うには、修正プログラムを読み込むためのハードウエアが必要になります。

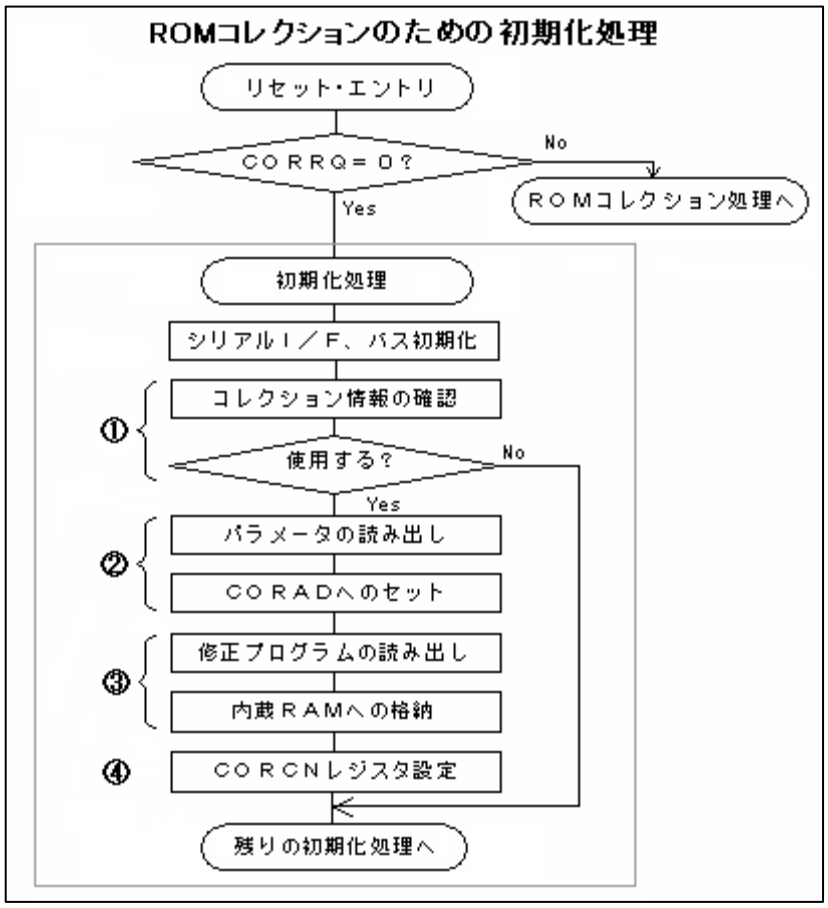
比較的大きなシステムで、バスを使用して外部に拡張用のフラッシュ・メモリなどを接続するようになっていれば、そこに修正プログラムを格納します。この場合は、プログラムそのものを外部に拡張したメモリ上で実行させることができます。

バスによる拡張を行っていない、比較的小さなシステムでは、シリアル・インタフェースで制御する EEPROM を接続し、その中に修正用プログラムを書き込んでおきます。この場合、初期化処理の中で、そのプログラムをシリアル・インタフェースを介して読み出し、内蔵 RAM に書き込んでおく必要があります。

(3) 初期化プログラムでの対応

ROM コレクションを使用するには、上述のハードウエアを準備し、初期化プログラムで必要な設定をしておく必要があります。また、ROM コレクションが発生すると、リセット・エントリからプログラムが起動するので、初期化処理の前に、ROM コレクションが発生したかどうかを CORRQ レジスタの内容で判定する処理が必要になります(上述の③)。

通常のリセットによる起動では、CORRQ レジスタはクリアされており、ROM コレクションが発生した場合は、CORRQ レジスタの該当するビット(CORRQ0-3)がセットされているので、CORRQ レジスタが 0 なら、初期化処理を行うようにします。



ROM コレクションを使用するには、初期化処理で上記フローの処理を行う必要があります。

以下、このフローに沿って説明します。

なお、外部に接続したメモリを使用できるよう、最初にシリアル・インタフェースやバス・インタフェースの初期化をしておいてください。

①ROM コレクションを使用するかどうかの判定処理

外部に接続したメモリの特定のアドレスに ROM コレクションを使う必要があるかどうかのフラグを準備しておき、 それを確認することで処理を行います。

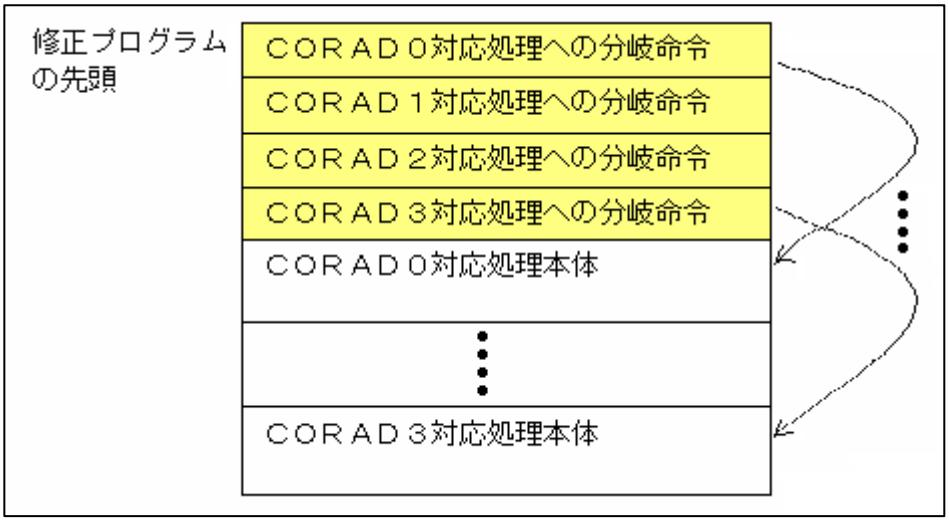
ROM コレクションが必要でない場合は、②～④の処理を飛ばして、次の処理に移ります。

②パラメータの設定

ROM コレクションに必要なパラメータ(コレクション・アドレス)を外部に接続したメモリから読み込み、CORAD レジスタへセットします。

③修正プログラムの読み込み

ROM コレクションで、代わりに実行させたいプログラムを外部に接続したメモリから読み込み、実際にプログラムを実行させる内蔵 RAM へ書き込みます。なお、外部メモリを接続し、そのプログラムを直接実行可能なら、内蔵 RAM へ書き込む必要はありません。内蔵 RAM または外部メモリの修正プログラムのアドレスは固定し、 下記の例のように先頭部分に分岐命令を並べておくと ROM コレクション処理が簡単になります。



④COREN の設定

COREN レジスタの **CORENn** ビットをセットすれば、対応する **CORADn** での ROM コレクション処理が有効になります。

(4) ROM コレクションの前処理

アドレスが一致して、修正したい命令の代わりに JMP r0 命令が実行され、リセット・エントリへ分岐します。

そこでは、(3)で説明したように、通常のリセットか ROM コレクション処理かの判定を行います。

ROM コレクションの場合には、どのアドレス(チャネル)での ROM コレクションが発生したかを **CORRQ** レジスタの **CORRQn** ビットで判定し、対応処理への分岐命令で分岐します。

各チャネルの ROM コレクション対応処理では、関連の **CORRQn** ビットを必ずクリアしてください。

(5) ROM コレクションの競合について

通常は、ROM コレクションが同時に複数発生することはありません。しかし、割り込み処理にも ROM コレクションを設定した場合には、ROM コレクション処理中に割り込みが発生し、割り込み処理の中で別の ROM コレクションが発生することが考えられます。

このような場合には、ROM コレクション対象プログラムの優先度の高いチャネルの **CORRQ0～3** ビットから順に確認を行うことで、 ROM コレクションのチャネルに優先度を設定できます。

このように処理に優先度をつけることで割り込みにも ROM コレクション処理を行うことができます。

1.3.15 フラッシュ書き込み

フラッシュ・メモリへの書き込み

Q1. 内部フラッシュ・メモリへの書き込み制御方法を教えてください。

A1. お客様が信号を制御するプログラム書き込みはできません。内部フラッシュ・メモリには、**専用ライタ**とパラメータ・ファイルで書き込みます（書き込み用プログラムは不要）。デバイスのマニュアル掲載のインタフェース回路で、ライタとデバイスを接続してください。書き込み方法は、ライタのマニュアルをご参照ください。

フラッシュ・ライタを使用する際の CPU 動作周波数 [V853]

Q1. UART 方式で専用フラッシュ・ライタを使用するとき、V853 の動作周波数は特定の値にするのですか。

A1. フラッシュ書き込みでは、10MHz 以上のクロックにする必要があります。その他では、UART ボー・レート誤差を小さくできる周波数(19.6608MHz など、伝送周波数の整数倍)をご使用になることをお勧めします。

内蔵フラッシュ・メモリに書きこむためのツール [V853]

Q1. V853(μPD703003)の内蔵フラッシュ・メモリにプログラムを書き込むには、どのようなツールが必要ですか。

A1. フラッシュ・ライタ(PG-FP4(ルネサス エレクトロニクス製)か FL-PR4(内藤電誠町田製作所製))と、プログラム・アダプタ(FA-100GC(内藤電誠町田製作所製))をご利用ください。なお、オンボード書き込み(システムに V853 のフラッシュ品を実装した状態での書き込み)では、ライタだけで、プログラム・アダプタは不要です。

内藤電誠町田製作所

TEL：042-750-4172　ホームページ：http://www.ndk-m.co.jp/asmis/

1.3.16 セルフ・プログラミングによる書き換え［共通］

ライブラリ、ユーザーズ・マニュアルの入手方法 [V850 All flash 共通]

Q1. セルフ・プログラミング・ライブラリ、ユーザーズ・マニュアルの最新バージョンを教えてください。

A1. **バージョンアップ・サービス**(下記 URL)に掲載のものが最新バージョンです。必ず最新版をご使用ください。なお、セルフ・プログラミング・ライブラリを公開していない製品については、販売店にお問い合わせください。お客様の製品に組み込まれた場合は、その製品の将来の保守のために、そのバージョン番号を記録するとともに、ドキュメント、ソフトの設計書を保管してください。

開発ツール・ダウンロードとユーザ登録：http://www2.renesas.com/micro/ja/ods/index.html

●ライブラリ、ユーザーズ・マニュアルをダウンロードするには ID/パスワードが必要です。

①次のどちらかの ID/パスワードを使用してください。

・当社製開発ツール製品をご購入された際にオンライン・ユーザ登録した ID/パスワード。

・フリーツールをダウンロードする際に、オンライン・ユーザ登録した ID/パスワード。

②ユーザ登録されていない方は、登録をしてください。

前記 URL ページの「> **マイコン登録ユーザ向けサービス**」で登録してください（ツール未購入でも可能）。

●フラッシュ・セルフ・プログラミング・ライブラリ ダウンロード手順

・手順 1

前記 URL のページで「> **バージョンアップ・サービス**」をクリックし、バージョンアップにアクセスしてください。

・手順 2

マイコンから選択 で、使用するマイコンを選択してください。

・手順 3

マイコン製品名 で、使用するマイコンを選択してください。

・手順 4

品名 または **添付資料** で、ダウンロードするファイルを選択してください。

・手順 5

・使用許諾の同意
・ID/パスワードの入力
・品名または添付資料の選択
・ダウンロード実行
・作業完了

セルフ・プログラミング実行時の割り込み処理記述方法 [V850 All flash 共通]

Q1. セルフ・ライブラリ・ユーザーズ・マニュアルに、「セルフ・プログラミング実行時に割り込み処理を行う場合は、**内蔵 RAM の先頭領域に記述**」とありますが、ジャンプ命令は**アセンブリ言語**で記述するのですか。

A1. はい、**アセンブリ言語記述**が必要です。内蔵 RAM の先頭領域にジャンプ命令を配置し、その先で割り込み要因を判別して、目的の割り込み処理へジャンプさせてください。

同マニュアルの「**フラッシュ環境時の割り込み処理**」の章に、次の例を掲載していますのでご参照ください。

- **内蔵 RAM 先頭アドレスで実行するソース（ノンマスカブル割り込み）**
- **内蔵 RAM 先頭アドレス＋4 バイト目で実行するソース（マスカブル割り込み）**

セキュリティの設定変更 [V850 All flash 共通]

Q1. FlashSetInfo 関数で、セキュリティ情報の設定を変更できますか。

A1. セルフ・プログラミングでは、セキュリティ情報を許可から禁止にすることはできますが，禁止から許可にすることはできません。禁止から許可にする場合は、プログラマでチップ消去の実行が必要です。ただし、チップ消去を禁止に設定した場合、プログラマでもチップ消去が実行不可となります。

Q2. セルフ・プログラミングで、セキュリティ情報をチップ消去禁止とブート・ブロック・クラスタ書き換え禁止に設定してしまいました。許可に設定することはできますか。

A2. セルフ・プログラミング、プログラマとも、**許可に設定できません。**チップ消去禁止とブート・ブロック・クラスタ書き換え禁止を設定すると、プログラマでのチップ消去もできず、セキュリティ設定許可に変更できません。

FlashBlockIVerify 関数の機能 [V850 All flash 共通]

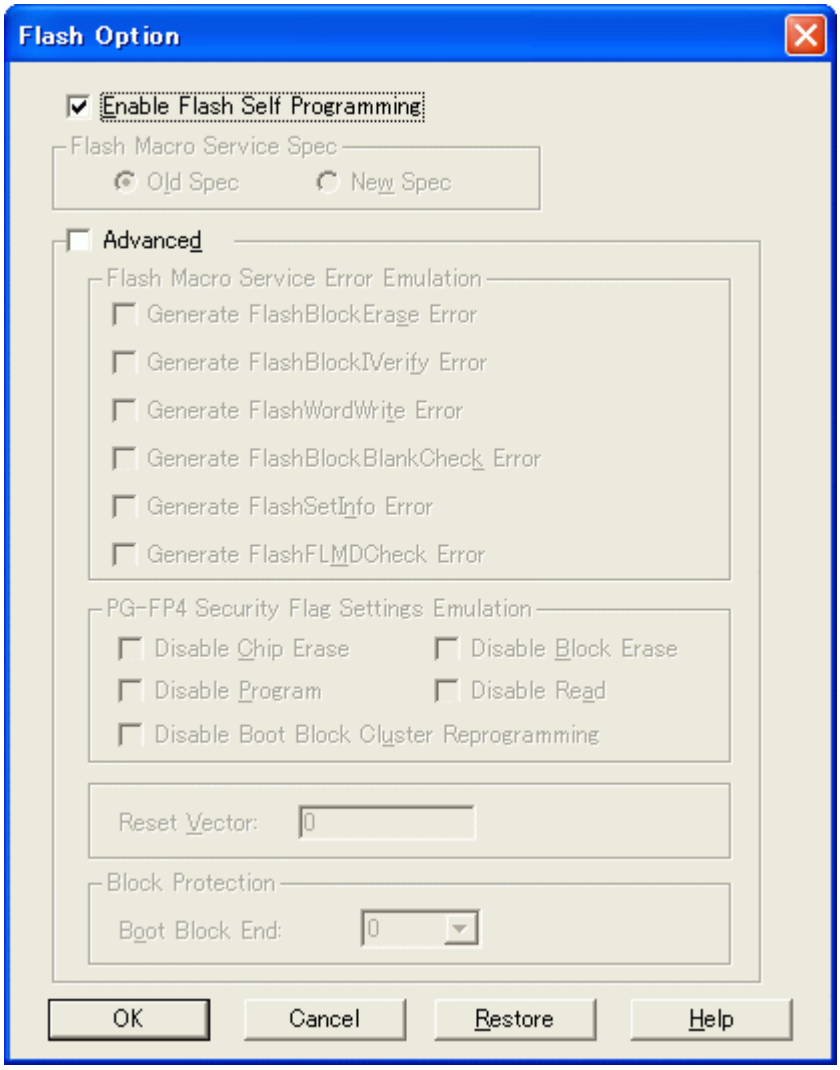
Q1. FlashBlockIVerify 関数は、**データ比較**をしないのですか。

A1. はい、**データ比較**をしません。プログラムでデータを比較（ベリファイ）してください。

★ セルフ・プログラミングのディバグ [V850 All flash 共通]

Q1. IECUBE でディバグ中、フラッシュ関数を実行すると意図しないアドレスに分岐しません。

A1. ID850QB の"**フラッシュ・オプション設定ダイアログ**"の"Enable Flash Self Programming"をチェックしているか確認してください。IECUBE でセルフ・プログラミングのディバグをするには、この設定が必要です（フラッシュ・セルフ・プログラミング・エミュレーション機能）。



Q2. IECUBE でディバグ中、フラッシュ関数内をアセンブラでステップ実行すると、意図しないアドレスに分岐します。

A2. IECUBE で、フラッシュ関数内をアセンブル・モードでステップ実行することはできません。この場合、実際のデバイスが実行するコードと異なるコードが実行されます。ディバグ時は、ソース・モードでステップ実行してください。

Q3. ID850QB でブート・スワップは実行できますか。

A3. いいえ、未サポートです。

Q4. MINI CUBE2 でセルフ・プログラミングを実行中、ハングアップします。

A4. 次の禁止事項に該当していないか確認してください。

(1)フラッシュ環境中にブレーク

(2)フラッシュ環境へ移行するプログラムのステップ・イン

(3)ディバグ用モニタ・プログラムが配置された空間をフラッシュ・セルフ・プログラミングで書き換え

Q5. IECUBE でフラッシュ・セルフ・プログラミング・エミュレーション機能を有効にしてプログラム実行すると、意図しないアドレスに分岐します。

A5. この機能を有効にすると、ディバッガが 0x0 番地からの 4 バイト分を予約領域として使用します。これによって、プログラムが不正に実行されている可能性があります。リセット・ハンドラの処理を次のようなコードにしてください。

```
# RESET handler（0 番地の場合）
.section "RESET", text
jr _start
jr _start --IECUBE 用に追加
```

Q6. IECUBE でセルフ・プログラミングをディバグ中、フラッシュ関数の実行で意図しないブレークが発生します。

A6. ID850QB を使用する場合、フェイルセーフ・ブレーク設定ダイアログの内蔵 RAM 欄の”Non Map“のチェックをはずしてください。

★

セルフ・プログラミング後のフラッシュ・ライタによるプログラム書き換え [V850 All flash 共通]

Q1. ブート・スワップを実行したデバイスに、フラッシュ・ライタでプログラムを書き込みました。電源を入れても、デバイスが動作しません。

A1. チップ消去で**ブート・スワップ・フラグ**の設定は初期化されますが、ブート・スワップ・クラスタ 0, 1 の内容が入れ替わるのは、ハードウエア・リセットの発生時です。チップ消去後、リセットせずに書き込むと、**ブート・スワップ・フラグ**とブート・スワップ・クラスタ 0, 1 が非同期の状態での書き込みとなります。この場合、プログラム書き込み後のリセットのタイミングでブート・スワップ・クラスタ 0, 1 が入れ替わることになり、プログラムが正常に動作しません。

ブート・スワップ実行後、フラッシュ・ライタによる書き込みは、次の手順で行ってください。

(1)チップ消去（プログラム消去、**ブート・スワップ・フラグ**の初期化）

(2)ハードウエア・リセット（**ブート・スワップ・フラグ**とブート・スワップ・クラスタ 0, 1 の同期化）

(3)プログラムの書き換え

1.3.17 セルフ・プログラミングによる書き換え(Type 別)

エントリ RAM の配置領域 [V850 All flash Type03]

Q1. エントリ RAM の配置領域に制限はありますか。

A1. セルフ・プログラミング実行中に割り込み処理を実行しない場合は、配置に対する制限はありません。
割り込み処理を使用する場合、下記のジャンプ命令がお客様自身によって組み込まれますので、内蔵 RAM の先頭 4 バイト以降か 8 バイト以降に、エントリ RAM を設置しなければなりません。

- **内蔵 RAM の先頭 4 バイトには「ノンマスカブル割り込み(NMI)処理へのジャンプ命令」**
- **内蔵 RAM の先頭 4 バイトからの 4 バイト領域には「マスカブル割り込み処理へのジャンプ命令」**

Q2. エントリ RAM は、グローバル配列とローカル配列のどちらでも問題ないですか。

A2. どちらでも問題ありません。グローバル配列の場合、その配置場所は内蔵 RAM 領域、外部 RAM 領域のどちらでもかまいません。関数で使用するスタック上に確保することも可能です。ただしその場合、フラッシュ関数を利用している間は上位の関数によって開放されないという条件が必要です。

エントリ RAM の設定方法 [V850 All flash Type03]

Q1. ユーザーズ・マニュアルに記載の「エントリ RAM を 4 バイト境界にする」とは、どのような設定ですか。

A1. プロジェクト・マネージャ・ツール PM+では、「**ツール＞コンパイラオプションの設定＞出力コード＞構造体パッキングの設定**」で「**ディフォールト[8 バイト]**」か「**4 バイト[Xpack=4]**」に設定してください。
その他の設定をする場合は、エントリ RAM に対して部分的にパッキング値を指定することも可能です。
次にサンプル・プログラムを紹介します。

```
#pragma pack (4)
int    entry_ram[ENTRY_RAM_SIZE/sizeof(int)];
#pragma pack ()
/* #pragma pack (4)で構造体のパッキングを指定し、
#pragma pack ()でオプションのパッキング値に戻します。*/
```

セキュリティ・フラグの設定方法 [V850 All flash Type03]

Q1. セキュリティ情報の「**ブート・ブロック・クラスタの書き換え禁止フラグ**(ビット 5)」を[1:書き換え許可]に設定すると、「**ブート・ブロック・クラスタの末尾ブロック番号**(ビット 24-31)」は無効ですか。

A1. はい無効です。[0:書き換え禁止]設定時に有効となります。その場合、「**末尾ブロック番号**(ビット 24-31)」で書き換え禁止領域を設定してください。ブロック 0 から「**設定ブロック番号**」までが書き換え禁止領域です。

セキュリティの有効範囲 [V850 All flash Type03]

Q1. セキュリティ情報で禁止した機能は、セルフ・プログラミング実行時も関数での実行ができないのですか。

A1. 実行できる関数とできない関数があります。セルフ・プログラミングでの機能の対応を次に示します。

セキュリティ・フラグ	フラグの設定値	書き換え対象	関　　数	実行可否
ブロック消去許可 チップ消去許可 書き込み許可 読み出し許可	0：禁止	全ブロック	ブロック消去関数(FlashBlockErase)	○
			書き込み関数(FlashWordWrite)	
			読み出し関数(FlashWordRead)	
ブロック消去許可 チップ消去許可 書き込み許可 読み出し許可	1：許可		ブロック消去関数(FlashBlockErase)	
			書き込み関数(FlashWordWrite)	
			読み出し関数(FlashWordRead)	
ブート・ブロック・クラスタ書き換え許可	0：書き換え禁止	ブート・ブロック・クラスタ	ブロック消去関数(FlashBlockErase)	×
			書き込み関数(FlashWordWrite)	
			読み出し関数(FlashWordRead)	○
	1：書き換え許可		ブロック消去関数(FlashBlockErase)	
			書き込み関数(FlashWordWrite)	
			読み出し関数(FlashWordRead)	
	0：書き換え禁止 1：書き換え許可	ブート・ブロック・クラスタ以外	ブロック消去関数(FlashBlockErase)	
			書き込み関数(FlashWordWrite)	
			読み出し関数(FlashWordRead)	

FlashSetInfo 関数のエラー発生要因 [V850 All flash Type03]

Q1. FlashSetInfo 関数の戻り値の定義を教えてください。

A1. 次のような定義になっています。

シンボル	戻り値	内　　容
FE_OK	0x00	正常終了
FE_PARA	0x05	パラメータ設定エラー • 指定されたブート領域の最終ブロック番号が製品のブロック数を越えた • 禁止されているセキュリティ情報を許可にした
FE_PROTECT	0x10	プロテクト・エラー • ブート・ブロック・クラスタ書き換え禁止状態で、そのサイズを変更しようとした(最終ブロック番号の変更) • ブート・ブロック・クラスタ書き換え禁止状態で、ブート・スワップ・フラグ(ビット0)を変更しようとした
―	その他	その他エラー • デバイス不良(フラッシュ・メモリ劣化など)が考えられます

セキュリティ情報の読み出し [V850 All flash Type03]

Q1. FlashGetInfo 関数以外で、セキュリティ情報を読み出せますか。

A1. はい、**プログラマの[シグネチャ・リード]コマンドの実行**により、現在のセキュリティ情報を確認できます。
セキュリティ情報は、シグネチャ情報に反映されます。

ブート・スワップ・フラグの設定 [V850 All flash Type03]

Q1. ブート・スワップ・フラグの設定方法を教えてください。

A1. ブート・スワップ・フラグの設定は、次の手順①～④とサンプル・プログラムを参考に実行してください。

①セキュリティ情報の取得

②ブート・スワップ・フラグの取得

③セキュリティ情報とブート・スワップ・フラグの合成

④ブート・スワップ設定の実行

```
//サンプルプログラム-------------------------------
 void err(int error, int phase):/*エラー処理関数*/
 unsigned long boot_flag;/*ブート・スワップ・フラグ*/
 unsigned long get_info;/*取得するセキュリティ情報*/
 unsigned long set_info;/*設定するセキュリティ情報*/
 unsigned long error;/*エラー情報*/
 /* ①セキュリティ情報の取得 */
 get_info = FlashGetInfo(entry_ram, 3) ;
 get_info &= 0xfffffffe;
 /* ②ブート・スワップ・フラグの取得 */
 boot_flag = FlashGetInfo(entry_ram, 4);
 boot_flag &= 0x01;
 /* ③セキュリティ情報とブート・スワップ・フラグの合成*/
 set_info = (get_info|boot_flag);
 /* ④ブート・スワップ設定の実行 */
 if(FE_OK != (error=FlashSetInfo(entry_ram, set_info))){
  err(error);
 }
//-----------------------------------------
```

セルフ・プログラミング実行時のタイマ5の使用禁止 [V850 All flash Type03]

Q1. セルフ・プログラミングを行う場合、ユーザ・アプリケーションでの**タイマ 5(TM50 と TM51)**の使用を禁止していますが、セルフ・プログラミング実行時だけ使用しなければいいですか。

A1. いいえ、セルフ・プログラミングを実行する場合は、ソフトウエア全体で**タイマ 5** を使用できません。

EEPROM エミュレーションで書き込み中の電源瞬断 [V850 All flash Type03]

Q1. EEPROM エミュレーションで EEPROM_Write 関数の実行中、電源瞬断で書き込みデータはどうなりますか。

A1. 正しく書き込まれません。EEPROM_Write 関数では、書き込みデータの終端に、終端を示す「デリミタ」を書き込み、その書き込み完了で正常終了となり、戻り値として**正常終了値 0x00** を返します。

Q2. EEPROM エミュレーションで書き込み中に電源が瞬断したあと、EEPROM_Read 関数でデータを読み出すと、書き込み不十分だったデータが読み出されますか。

A2. いいえ、そのようなデータは読み出されません。「デリミタ」が正しく書き込まれた最新データを読み出します。

★ フラッシュ書き込み環境での内蔵 ROM アクセス [V850 All flash Type04]

Q1. フラッシュ書き込み環境で、内蔵 ROM に書き込まれたデータを読み出せますか。

A1. はい、オプション・ライブラリ・ファイル「**selflibv850_type04r_vxxx_j/e.zip**(xxx：バージョン)」でできます。このファイルは、V850 セルフ・プログラミング・ライブラリ Type04 に読み出し機能を追加します。バージョンアップ・サービス(https://www5.renesas.com/micro/tool_reg/OdsListTop.do?lang=ja)で対象デバイスを選択し、フラッシュ・セルフ・プログラミング・ライブラリの関連ファイルから取得してください。

★ FLMD0 の設定タイミング [V850 All flash Type04, 04a]

Q1. FLMD0 端子は、FlashEnv 関数の前後どちらのタイミングでハイ・レベルに設定するのですか。

A1. どちらでもけっこうです。

★ FlashInit 関数の実行環境、実行タイミング [V850 All flash Type04, 04a, 02c]

Q1. FlashInit 関数は、内蔵 ROM 以外(RAM、外部 ROM)で実行する必要がありますか。

A1. いいえ、不要です。FlashInit 関数実行時はフラッシュ書き込み環境に移行していないので、内蔵 ROM 上で実行できます。

Q2. フラッシュ書き込み開始の FlashEnv 関数(引数 1)を実行するたびに、FlashInit 関数の実行は必要ですか。

A2. はい、必要です。

★ フラッシュ関数のエラー値の発生要因 [V850 All flash Type04, 04a, 02c]

Q1. フラッシュ関数実行後、エラー"SELFLIB_ERR_FLASHPROCx"(ハードウエア障害により処理が失敗)が発生するのは、どのような要因ですか。

A1. 次のような要因が考えられます。

・FLMD0 端子がハイ・レベルでない。

・未消去ブロックへの書き込みをしている。

・ブランク・チェックを実行したブロックが、ブランク状態でない。

また、何らかの要因で、戻り値が破壊されていることも考えられます。スタックなど、RAM 配置に問題がないか確認してください。

Q2. ユーザーズ・マニュアルに記載されていない、フラッシュ関数の戻り値でした。

A2. そのような戻り値はありません。戻り値が破壊されていると思われるので、スタックなど、RAM 配置に問題がないか確認してください。

★ FlashStatusCheck 関数のエラー発生後の再実行 [V850 All flash Type04, 04a, 02c]

Q1. FlashStatusCheck 関数でエラーが発生し、再度実行すると正常終了します。どちらが正しいですか。

A1. エラーです。戻り値のエラーは一度だけ返り、その時点で、フラッシュ関数は処理完了です。そのあと FlashStatusCheck 関数を実行すると、ハードウエア処理が完了状態なので、"**SELFLIB_OK**"が返ります。

★ セルフ・プログラミング・ライブラリ・ファイルのリンク方法 [V850 All flash Type04, 04a, 02c]

Q1. PM+でのセルフ・プログラミング・ライブラリ・ファイルのリンク方法を教えてください。

A1. PM+のメニュー[**ツール**]→[**リンカオプションの設定**]ダイアログを開いて、[**ライブラリ**]タブをクリックします。[**ライブラリ**]テキスト・ボックスに**セルフ・プログラミング・ライブラリ・ファイル名**(a ファイルの lib から.a の間に配置されている文字列)、[**ライブラリのパス**]テキスト・ボックスにセルフ・プログラミング・ライブラリ・ファイルが格納されているフォルダの**パス名**を入力してください。

例　libf.a の場合、“f”
libf_type1.a の場合、“f_type1”
libf_type3.a の場合、“f_type3”

★ セルフ・プログラミング・ライブラリをリンクしたプロジェクトのビルド時のエラー発生要因 [V850 All flash Type04, 04a, 02c]

Q1. CA850 でビルドすると、次のようなエラーが発生します。

```
ld850: CA850 error F4452: undefined symbol.
_sSelfLib_ToRamUsrInt.text ・・・
_eSelfLib_ToRamUsrInt.text ・・・
_sSelfLib_ToRamUsr.text ・・・
_eSelfLib_ToRamUsr.text ・・・
```

A1. SelfLib_ToRamUsrInt.text、SelfLib_ToRamUsr.text セクションが定義されていないのが原因です。
セルフ・プログラミング・ライブラリは、ライブラリ内部でこの 2 つのユーザ・セクション名を参照するので、ユーザが定義する必要があります。

★ ブート領域に対するセキュリティ機能 [V850 All flash Type04, 04a, 02c, 03]

Q1. 誤ったプログラム操作によるセルフ・プログラミングで、ブート領域を破壊することを防ぐ方法はありますか。

A1. **ブート・ブロック・クラスタの書き換え禁止設定**で防げます。
ブート・ブロック・クラスタとは、プログラマ、セルフ・ライブラリによる消去／書き換えを禁止できるブロックの集合(クラスタ)です。プログラマのセキュリティ設定か、セルフ・ライブラリの FlashSetInfo 関数で、次の 2 つの設定が可能です。
・ブート・ブロック・クラスタ領域の範囲
・ブート・ブロック・クラスタ領域を書き換え「**許可**」から「**禁止**」へ変更

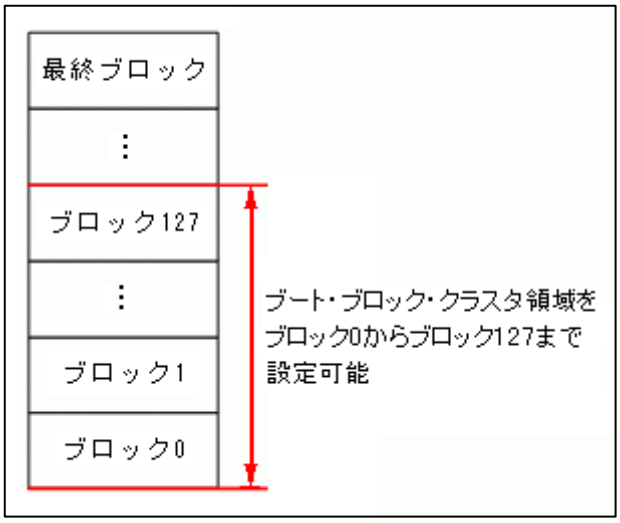
(1)領域の範囲

ブート・ブロック・クラスタの設定可能な範囲は、製品ごとに異なります。対象デバイスのユーザーズ・マニュアルを参照してください。また、開始ブロック番号は 0 固定(可変リセット・ベクタ対応製品ではリセット・ベクタ・アドレスを含むブロックの番号)で、末尾ブロック番号だけ設定可能です。

(2)FlashSetInfo 関数の注意事項

FlashSetInfo 関数の実行時は、ブート・ブロック・クラスタの書き換え許可／禁止にかかわらず、ブート・ブロック・クラスタの末尾ブロック番号を設定可能な範囲内の値に設定してください。範囲外の値で実行すると、パラメータ・エラーが発生します。

例　指定可能な最大ブロックがブロック 127 までの場合



1.3.18　電　源

リセット中は低消費電力か

Q1　V850E/MA1 のリセット信号を入れ続けると、電源電流は HALT や STOP モードより少ないですか。

A1.　いいえ、HALT 時と同程度と考えられます。リセット中でもクロックは発振しますし、各回路に供給されます。

電源電流を抑えられるモード(省電力)

Q1.　V850/SB1 の停止状態で時間計数をする場合、一番省電力のモードは HALT、IDLE、STOP のどれですか。

A1.　モードで電源電流が一番少ないのは STOP モードですが、時計のカウント動作を考えると、サブクロック動作とサブクロックでの IDLE モードを組み合わせた方が、トータルでは電流を少なくできます。

1.3.19　リセット

リセット解除のタイミング

Q1.　V850/SB1、SA1 のリセット解除タイミングは、立ち上がりエッジだけの判断でいいですか。

A1.　いいえ、リセットはエッジでなく**レベル**で認識します。インアクティブになることが、リセットの解除です。

ソフトウエアでリセット

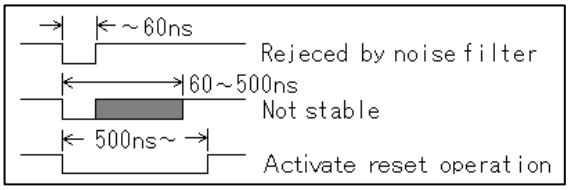
Q1. V850E/MS1 でソフト・リセットをかけるには、PC=0x00000000 にジャンプするコードを書けばいいですか。

A1. 単に、割り込み内からプログラムの先頭にジャンプしても、その後は正常実行できません。 EIPC レジスタと EIPSW レジスタを設定し、RETI 命令を実行してください。これで、その後の割り込み処理が可能になります。その上で、プログラム先頭の初期化部分に分岐させてください。ただし、多重割り込みを使用している場合、インサービス・プライオリティ・レジスタ(ISPR)の内容が 00 になるまで RETI 命令を繰り返す必要があります。なお、リセット時の初期値が定義されているレジスタについては、使用したものの初期値を事前にプログラムで設定しておく必要があります。

リセット信号のパルス幅

Q1. RESET 端子のロウ・レベルのパルス幅は、最低どの程度必要ですか。

A1. リセットが正常に行われるために必要な時間は 500ns です。電源電圧が動作範囲内で、クロック発振が安定した状態で、リセット入力が 500ns 以上ロウ・レベルであれば、リセットがかかります。
約 60ns 以下の短いパルスでは、アナログ・ディレイによるノイズ除去が行われます。
60～500ns では、リセットされない内部ブロックのある可能性があります。この状態は、個々のデバイスや使用条件などで変わります。その動作は確定しませんので、60～500ns のパルスは入力しないでください。
確実にリセットするには、500ns 以上のパルスを入力してください。



リセットしたときの RAM データは保持されるか

Q1. CPU をリセットしたとき、内蔵 RAM のデータは保持されますか。

A1. 通常動作中のリセットでは、RAM のデータの保持は保証していません。
リセットは、通常の動作とは非同期でかかります。このため、RAM への書き込み中にリセットがかかることも考えられます。この場合には、RAM への書き込みが正常に行われないことが考えられます。このため、通常動作中でのリセットがあった場合の RAM のデータは保証できません。
これは、リセット・タイミングの問題です。RAM への書き込みを行っていないときにリセットがかかれば、問題はありません。しかし、たまたま RAM への書き込み時にリセットがかかると、リセット信号の幅が規定値未満だったり、書き込み信号がオンのときにアドレスが変動したりするため、書き込み中だったアドレスのデータはもちろん、他のアドレスのデータにも書き込みが起こることがあり、RAM の内容は保証されなくなります。

★ リセット解除前のクロック安定時間

Q1. クロック入力をする場合、マイコンの発振安定時間はなくなると考えられますが、リセット解除時に、クロックを安定供給していればいいですか。

A1. CPUのリセットを解除する500ns以上前から、安定したクロックを入力しておく必要があります。各製品のドキュメントに掲載している電気的特性で、「リセット・タイミング」をご参照ください。

1.3.20 端子の DC 特性と処理

プルアップ抵抗の上限値

Q1. V850/SA1 を外部バス仕様で使っています。できるだけ消費電力を低減したいので、バスのプルアップ抵抗を高抵抗にしたいのですが、設計基準のようなものはありますか。

A1. 基本的には、バスがハイ・インピーダンスになったときに、中間レベルとならない値とします。このためには、バスのリーク電流がプルアップ抵抗に流れても、ハイ・レベルを保持できる値にしてください。
通常は、50kΩ～100kΩ 程度と考えられます。
抵抗値を大きくすると、ノイズに弱くなります。これを回避するには、抵抗値を小さくする必要があります。環境は個々のシステムで異なるので、どの程度にするかは、お客様の要求仕様に合わせて決定されます。

シュミット・トリガ入力のスレッシュホールド規格

Q1. 「シュミット・トリガ入力スレッシュホールド電圧」と「ハイ・レベル入力電圧」、「ロウ・レベル入力電圧」が別々に規定されていますが、入力電圧が V_T^+～$0.75V_{DD}$ のとき、論理はどのように決定されるのですか。

A1. 入力電圧でお考えください。シュミット・トリガ入力スレッシュホールド電圧は参考値で、規格ではありません。

未使用端子の処理

Q1. V850E/MA1 ユーザーズ・マニュアルの「2.4」で、ポート端子の推奨接続が、次のように記述されています。
入力状態：抵抗を介して V_{DD} または V_{SS} に接続
出力状態：オープン
リセット時はほとんどのポートが入力状態となりますが、実際の回路設計では次のいずれですか。
(1) リセット時の状態を無視して、出力ポートで使用する端子はオープンで良い。
(2) リセット時に入力となる端子は、必ず抵抗を介して V_{DD} などに接続する必要がある。

A1. 一概には言えません。出力として使用する端子では、どのような使われ方をするか、接続相手の状況も考慮が必要です。未使用端子は理想的には(2)ですが、システムの動作電流に余裕がない場合(電池駆動など)を除いては、システムをコンパクトにするために(1)の処理も使えると考えられます。

出力端子を 5V にプルアップしていいか [V850E/MA1]

Q1. アプリケーション・ノートの第2章の冒頭の(5)で、5V 動作メモリでも DC 特性を満たせば接続可能と記述されていますが、CPU の出力ポートを 5V にプルアップしていいですか。

A1. いいえ、だめです。電気的特性で出力電圧を「－0.5V～V_{DD}＋0.5V」と明確に規定しています。外部メモリで DC 特性を満たすというのは「3V でのインタフェースが可能なら」という意味です。

P00～P07 は未使用時に V_{DD} に接続していいか [V850E/MA1]

Q1. μPD70F3017A で、未使用時の端子処理の記述に、P00～P07 は「入力時：V_{SS} に接続してください」となっています。I/O ポートの入力端子設定(初期値)の場合、V_{DD} にプルアップしてはいけませんか。

A1. いいえ、V_{DD} でも問題ありません。古い製品で、抵抗でプルアップしていたとき、電源の立ち上がり時に端子の電圧の立ち上がりが遅くて、ロウからハイと認識されて、割り込みが発生したことがあったようです。これを避けるため、そのような処置がされたものです。このデバイスの初期状態では、NMI の有効エッジが指定されていませんので、このような問題が発生することはありません。

空き端子とバス・ラインに使用するプルアップ抵抗値

Q1. 空き端子処理とバス・ラインなどに使用するプルアップ抵抗の推奨値はありますか。また、何を基準に決めればいいのですか。

A1. 特に推奨値はありませんが、未使用端子については、ハイ・レベルのリーク電流が流れてもハイ・レベル入力電圧を維持できる値にしてください。たとえば V_{DD}＝3.0V の場合、10μA のリーク電流が流れても、$0.75V_{DD}$(＝2.25V)を維持するには 75kΩ 以下となるので、75kΩ 以下の抵抗で V_{DD} にプルアップしてください。
また、バス・ラインに関しては、バス・ラインに接続されているデバイスのハイ・レベル・リーク電流の合計でもハイ・レベルが維持できる値が目安ですが、接続されたデバイスのロウ・レベルのリークや入力電流とプルアップ抵抗の電流を合計したものと、最もドライブ能力の低いデバイスのロウ・レベルのドライブ能力を比較して、ハイ・レベルからの値とロウ・レベルからの値の中間程度の抵抗値で、V_{DD} にプルアップしてください。
なお、出力を絶対最大定格以上にはできませんので、3.3V 以下での動作デバイスの出力は、5V にプルアップしないでください。

V_{DD} 電源なしのときに、ポートに外部からハイ・レベルをかけていいか

Q1. CPU の電源が供給されてないとき、ポートに外部からハイ・レベル信号が入力されてもいいですか。

A1. だめです。正規の電源電圧が印加された状態でしか、ポートに電圧を印加してはいけません。電源電圧が印加されていない状態でポートにハイ・レベルを入力し、その状態で電源を投入すると、デバイスが破壊することが考えられます。なお、ポートから V_{DD} の方向には、入力保護ダイオードは入っていません。

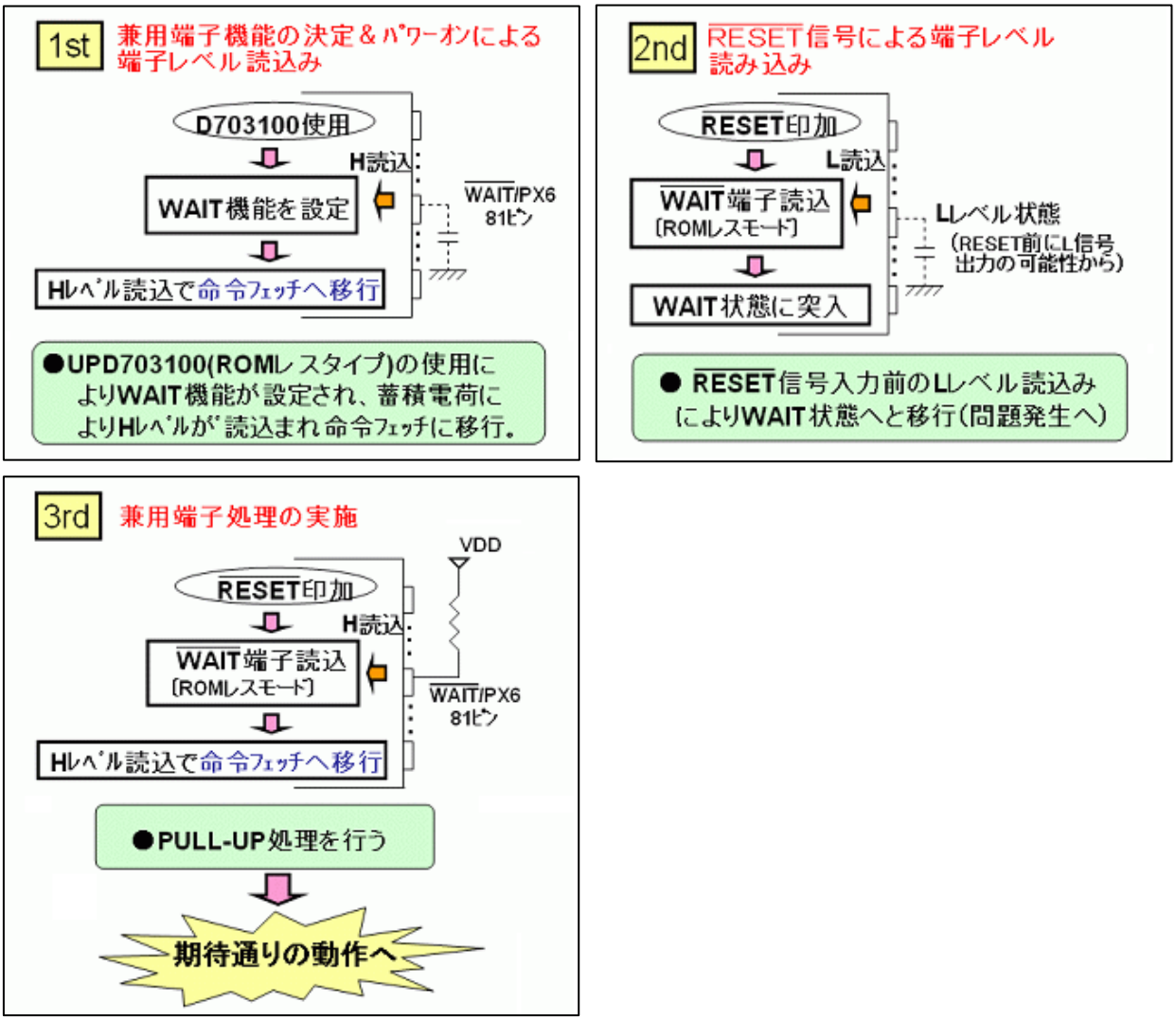
ポート／制御兼用端子の処理 [V850E/MS1]

Q1. V850E/MS1 の ROM レス品で、2 回目以降のリセットでは、アドレス／データ・バスが動作しません。

A1. $\overline{\text{WAIT}}$ 端子の未処理(オープン状態)が原因です。$\overline{\text{WAIT}}$ や $\overline{\text{HLDRQ}}$ は、正しくプルアップしてください。

シングルチップ・モード 0 では、この端子の初期状態はポートですが、ROM レス・モードでは $\overline{\text{WAIT}}$ 端子です。パワーオン後の最初の命令をフェッチする際に、$\overline{\text{WAIT}}$ 端子のレベル確認が行われます。このとき $\overline{\text{WAIT}}$ 端子がオープンなら、パワーオン時の電荷蓄積でたまたまハイ・レベルと認識され、ウエイト挿入をせずに正常に動作します。しかし、その後のリセット入力では、$\overline{\text{WAIT}}$ 端子の信号レベルがリセット入力以前の状態に依存するため、ロウ・レベルの場合はリセット解除後、すぐにウエイト状態に移行し、バス動作が停止します。

兼用端子処理対応の流れ（μPD703100 での事例）



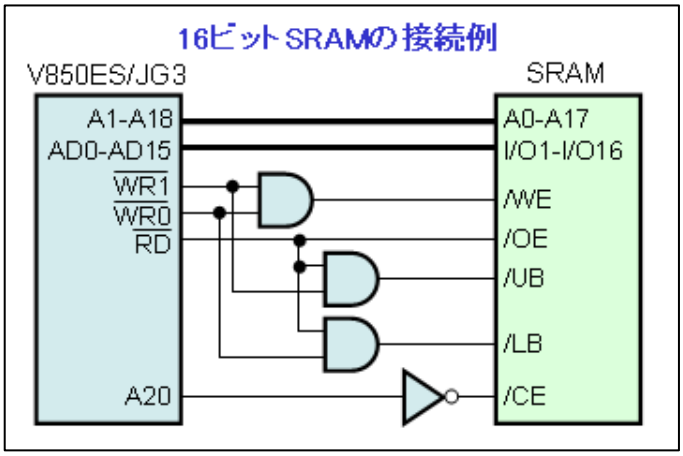
★ 1.3.21 バス・インタフェース

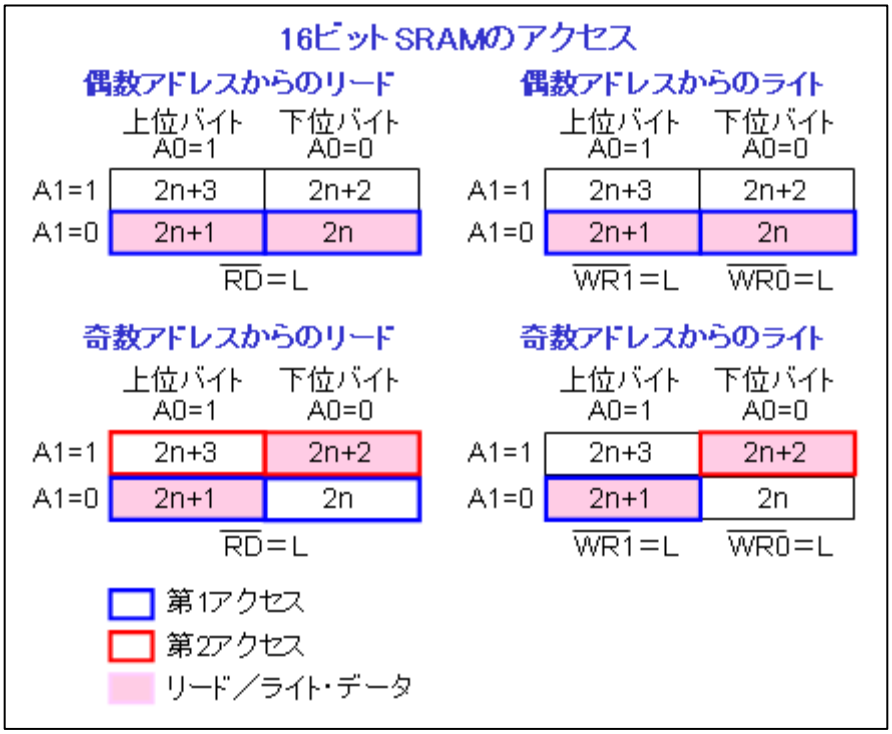
16 ビット SRAM の接続

Q1. リード・ストローブ信号は $\overline{RD}$ の 1 本だけですが、なぜライト・ストローブ信号は $\overline{WR0}$,$\overline{WR1}$ の 2 本あるのですか。また、16 ビット・データのライト・アクセスをする場合、$\overline{WR0}$ の 1 本だけ使用すればいいですか。

A1. 奇数番地からの 16 ビット・ライトでは、$\overline{WR}$ 信号を分けなければ、アクセス対象でないデータの書き込みをしてRAM 内容を破壊します。リードでは、CPU がアクセス対象でないデータの読み込みをしなければいいだけなので、特に $\overline{RD}$ は分けていません。

偶数番地からのライト・アクセスしかしなければ、$\overline{WR0}$ だけでのアクセスは可能です。通常は、CPU の A0 を使用せず（16 ビット・アクセス）、奇数番地からのライト・アクセスでは上位と下位を $\overline{WR1}$,$\overline{WR0}$ で切り替えます。





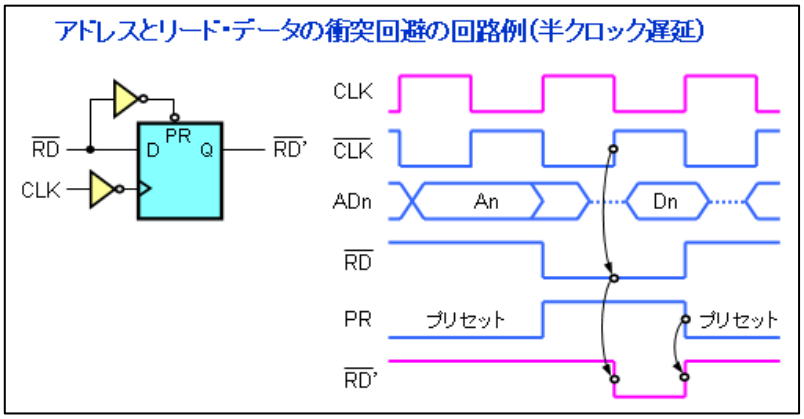
マルチプレクス・バスでの制御

Q1. マルチプレクス・バスでメモリや周辺デバイスをアクセスするには、どのような接続をすればいいですか。

A1. マルチプレクス・バスでは、バス・サイクル内でアドレスとデータが時分割で出力されます。このため、データ・アクセスまでアドレスを外部でラッチして、データと分離する必要があります。ASTB出力の立ち下がりが、そのトリガとなります。

注意 デバイスによっては、リード・サイクルで、$\overline{RD}$の立ち下がりからのアドレス・フロート遅延が長いため、アドレスとリード・データが衝突することがあります。タイミングをご確認のうえ、このような場合は、$\overline{RD}$の立ち下がりを遅延させる回路を追加してください（立ち上がりは遅延させず）。

アドレスとリード・データの衝突回避の回路例（半クロック遅延）



1.3.22 その他周辺機能

外部 I/O 接続の簡略化(アドレス・バス／データ・バス) [V850E/IA2]

Q1. 外部に I/O デバイスを接続する際、簡略化のため/CS を I/O ポートで代用しています。そのため、アドレス・バスの上位ビット用のポート(ポート DH)は不要です。そこで DH を入出力ポート設定とした場合、他のポートを外部バス用の設定にすれば、データ・バス、制御端子は正常に動作しますか。

A1. はい、他の外部アクセスのバス、制御信号などは正常に動作します。

I/O 接続のアドレス・デコード簡略化 [V850E/MS1]

Q1. V850E/MS1 の外部に I/O デバイスを接続する場合、アドレス・デコードをせずに直接アクセスできますか。

A1. アドレスについては、V850E/MS1 ではメモリ領域が 7 つのブロックに分かれています。そして、各ブロックにどのような外部デバイスを接続するか指定できます。このため、外部に接続するメモリや I/O が少なければ、各ブロックに 1 つのデバイス(メモリや I/O)を接続できます(/CS0-7 信号を外部 I/O の CS に接続)。
これは、どれだけのデバイスを外部に拡張するかの問題で、V850E/MS1 だけでは決まりません。アドレスは外部のデバイスを区別するための信号です。区別する必要がなければ(I/O が 1 つしかない場合)、アドレスをデコードする必要はありません。

Q2.外部 I/O に直接アクセスできる場合、8 ビット・データ・バスの I/O にアクセス可能ですか。

A2. V850E/MS1 は、バス幅が 8 ビットか 16 ビットかをメモリ・ブロックごとに指定できます(バス・サイジング機能)。これを使用すれば、8 ビット幅のデバイスを使用できます。

初期値が不定のポートは、出力設定前に値を書き込めば不定状態を回避可能か

Q1. 各ポートの初期値は不定ですが、ポート・モードの出力モードとして動作させるとき、設定前にポートに値を書き込めば、不定状態を回避できますか。

A1. はい、これにより、モード設定からデータ設定までの端子の不定状態を回避できます。ただし、ポートを出力に設定したときに、出力バッファがオンとなるときのゲート遅延によるヒゲ出力の可能性があります。

ポートをコントロール・モードに設定していても、ポート入力データを見られるか

Q1. I/O 兼用端子について、ポートのモード・レジスタで入力に設定して、モード・コントロール・レジスタでコントロール・モードを使用するように設定した場合、その端子がコントロール・モードで動作していても、ポート入力データを知ることができますか。

A1. はい、モード・レジスタで入力に設定されていれば、たとえコントロール・モードで使用していても、ポートを読めば、そのときの端子の状態を知ることができます。

WAIT 端子でウエイト制御をするとき、兼用ポートはアドレス出力可能か [V850/SB1]

Q1. μPD70F3032A で、P110/A1/WAIT 端子のウエイト機能を使用する場合、ポート 3,10,11(P110/A1/WAIT 以外)はアドレス出力として使用できますか。

A1. そのような動作はできません。ウエイト機能を使用する場合、マルチプレクス・バス・モードをご利用ください。

システム・ウエイト・コントロール・レジスタ(VSWC) [V850E、V850ES]

FAQ の「実行時間：内蔵周辺アクセスでの実行速度 [V850 共通]」もご参照ください。

Q1. VSWC の機能について教えてください。

A1. 一般に、CPU を高速で動作させる場合、内蔵周辺まで含めて高速動作させるのは、デバイス設計上、またパワーやノイズの面でも問題があります。そのため、バスを分離して、高速で動作させる必要のある部分と低速で動作させてもよい部分に分けることになります。通常、内蔵周辺は低速で動作させるバスに接続されます。また、各種の内蔵周辺をいろいろなデバイスで共通に使用することを考えると、速度の異なる CPU に対応できるようにする必要があります。つまり、CPU の速度に応じてウエイト数を加減し、適切なアクセス時間を確保する必要があります。

これを考慮して、V850 では **VSWC** レジスタで、CPU とその動作周波数に応じて、アクセス時のウエイト数を最適に設定できるようにしています。**VSWC** レジスタは、上位 4 ビットと下位 4 ビットで構成され、アクセスの準備に関係するウエイト数と、アクセス自体に関係するウエイト数を独立に指定できます。

VSWC レジスタの初期値は 77H で、内蔵周辺のアクセス時に「7+7」の計 14 クロックのウエイトがかかります。したがって、ノー・ウエイトで 3 クロックの場合には、17 クロックのアクセス時間となります。

1.3.23 製品の違い

A の有無の違い [V853]

Q1. μPD70F3003 と μPD70F3003A との相違点はなにですか。

A1. 基本的には、ROM レス・モードの有無、CKC レジスタの初期値設定と、これに伴って MODE 端子の機能が異なります。

	μPD703003	μPD703003A
ROM レス・モード	あり MODE=0 : ROM レス・モード MODE=1 : シングルチップ・モード	なし
CKC レジスタ初期値	00	MODE 端子で選択 MODE=0 : 03 MODE=1 : 00
パッケージの厚さ	1.45mm	1.40mm

1.3.24 定　格

μPD703107 データ・シートの出力遅延時間

Q1. μPD703107 の規格で、次のようにクロック分の差異がありませんが、正しいですか。
データ出力遅延時間（対 CLKOUT ↓）<36> : 2～13ns
データ出力遅延時間（対 CLKOUT ↑）<37> : 2～13ns

A1. はい、正しいです。
<36>は出力バッファが出力を開始するタイミングで、<37>は出力データが確定するタイミングです。
したがって、<36>と<37>の間は複数の線で表していますが、これはデータが不定となることを示します。
<36>はその直前のデータ・リード後のバスのフローティングと、次のデータ・ライトでのバスの出力の衝突を確認するための規格です。ライト動作の場合には、<37>の規格を適用してください。

CLKOUT 同期と CLKOUT 非同期の 2 つの規格 [共通]

Q1. 外部バスに、CLKOUT 同期と CLKOUT 非同期の規格がありますが、両方を満足させる必要がありますか。

A1. いいえ、どちらか一方の規格を満足するだけで結構です。

Q2. 外部バスの CLKOUT 同期と CLKOUT 非同期の規格は、どのように使い分けるのですか。

A2. 外部バスにSRAMのような、アドレスとデータ、RDやWRなどの簡単な制御信号だけで制御できるデバイスしか接続しないのであれば、CLKOUT 非同期の規格をご使用ください。CLKOUT 非同期の規格では、SRAM とのタイミング設計が比較的簡単にできます。
外部バスに接続するデバイスが、CPU の出力信号だけでは制御できず、もっと細かいタイミングでの制御が必要なときは、CLKOUT 信号でタイミングをとるために、CLKOUT 同期の規格をご使用ください。

Q3. 外部バスにSRAMを接続してウエイトをかける場合、WAIT 入力を制御する必要があるので、CLKOUT 同期の規格を使わないといけないのですか。

A3. プログラマブル・ウエイト機能を利用すると、最大7クロック分はバス・サイクルを延長できます。この範囲のウエイトで済むなら、CLKOUT 非同期の規格が使用できるので、簡単にバス・タイミングの設計ができます。

1.3.25 その他

内部周辺 I/O 領域の「イメージが見える」

Q1. ユーザーズ・マニュアルに「V850/SB1、V850/SB2 は、物理内部周辺 I/O として FFF000H-FFF3FFH 番地の1K バイトに割り当てており、残りの領域(FFF400H-FFFFFFH)にはそのイメージが見えます」と記述されていますが、「イメージが見える」とはどういうことですか。

A1. アドレスを完全にデコードしていない場合、複数のアドレスで同じ内容が見え、それをイメージと呼びます。この場合には、A11,10 がデコードされていないので、次のように複数のアドレスで同じ内容が見えます。

FFF000H のイメージ：FFF400H、FFF800H、FFFC00H
FFF001H のイメージ：FFF401H、FFF801H、FFFC01H
FFF002H のイメージ：FFF402H、FFF802H、FFFC02H
：
FFF3F8H のイメージ：FFF7F8H、FFFBF8H、FFFFF8H

2. プラットフォーム製品

2.1 ゲート・アレイ

シリーズ共通（CPU 周辺、RAM、Compiled RAM）

Q1. 故障検出率を上げるには、RAM やメガマクロの扱いをどのようにすればいいですか。

A1. マクロ単体は気にせず、ユーザ回路のカバレージ・アップに留意して、テスト・パタンを作成してください。
メガマクロについては、次の FAQ もご参照ください。

シリーズ共通（CPU 周辺）

Q1. メガマクロのテスト手法について教えてください。

A1. メガマクロの内部は、当社で用意するテスト・パタンでテストします。そのための簡単なテスト回路（デコーダなど）を作成していただく必要があります。

Q2. メガマクロを複数（2 つ以上）搭載することはできますか。

A2. 可能ですが、希望マスタに搭載できるだけの面積があるかどうかは、見積もりツールでご確認ください。

シリーズ共通（BIST、Compiled RAM、RAM、Test pattern）

Q1. RAM のテスト手法について教えてください。

A1. 設計フローでテスト回路(BIST:Built-In Self Test)回路を組み込みますので、ユーザ側でのテスト・パタン作成は不要です。詳細は設計マニュアル（EA-9HD はメモリマクロ編）をご参照ください。

シリーズ共通（パッケージ）

Q1. FPBGA とプラスチック FBGA は、同じものですか。

A1. はい、同じものです。次の表記は、すべて同じものを指しています。

QFP ファインピッチと QFP(FP)
QFP ヒートスプレッダと QFP(HSP)、QFP(HS),
テープ BGA と TBGA
プラスチック BGA と PBGA
アドバンスト BGA と ABGA

EA-9HD（インタフェース・バッファ）

Q1. 3.3V と 5V の両方にインタフェース可能なゲート・アレイはありますか。

A1. EA-9HD で対応可能です（5V フル・スイング・バッファ内蔵）。
CMOS-9HD や CMOS-N5 では、5V インタフェースの部分に外付け抵抗を追加すれば実現できます。

EA-9HD、CMOS-9HD(インタフェース・バッファ)

Q1. 5V 耐圧と failsafe 機能を備えたバッファはありますか。

A1. 技術的に実現困難で、ありません。failsafe 機能を使う場合は、外部で 5V→3V に変換してください。

EA-9HD、CMOS-9HD(一般)

Q1. EA-9HD, CMOS-9HD の最高動作周波数は何 MHz ですか。

A1. 100MHz です。消費電力がパッケージの許容値以内であることと、同時動作制限が設計マニュアルの基準を満たしていることが条件です。また逓倍後の周波数が 100MHz を越える場合も、原則として対応できません。なお、100MHz以上の強いご希望がある場合は、販売店を通じて、個別にご相談ください。

EA-9HD、CMOS-9HD(ROM)

Q1. ROM を搭載できるシリーズを教えてください。

A1. CMOS-9HD,EA-9HD です。

EA-9HD、CMOS-9HD(発振回路)

Q1. CMOS-9HD, EA-9HD の発振器の周波数は何 MHz ですか。

A1. 4MHz から 48MHz です。この範囲を外れた場合、低周波ではノイズが、高周波ではマイグレーションが問題になる可能性があります。

EA-9HD、CMOS-9HD(テスト・パターン、ROM)

Q1. ROM のテスト手法について教えてください。

A1. お客様にて、テスト・パタンを作成してください。

CMOS-9HD(一般、PLL、CPU 周辺、NEC_BSCAN)

Q1. CMOS-9HD で、CMOS 3.0V 対応はできますか。

A1. はい、できます。ただし PLL、BSCAN、メガマクロは使用できません。

EA-9HD、CMOS-9HD、CMOS-10HD(PLL)

Q1. 複数の PLL を搭載できますか。

A1. はい、できます。ただし、複数の PLL 相互の位相調整はできません。

CMOS-10HD(インタフェース・バッファ)

Q1. LVDS、SSTL2 などの高速バッファはありますか。

A1. CMOS-10HD に内蔵しています。詳細は販売店にお問い合わせください。

CMOS-N5(インタフェース・バッファ)

Q1. CMOS-N5 で、3.3V　TTL インタフェースは可能ですか。

A1. はい、可能です。

Q2. CMOS-N5 を低電圧(V_{DD}=3.0V±10%～5.0V±10%)で使用したとき、5V 信号とインタフェースできますか。

A2. 入力バッファに failsafe 機能付きバッファを使い、出力バッファに failsafe 機能付きの NchOpenDrain を使えば可能です(通常タイプの NchOpenDrain バッファでは、リーク電流が流れるため不可)。出力バッファは NchOpenDrain バッファですので、外部でプルアップする必要があります。また高速動作には向きません。

CMOS-N5(NEC_BSCAN)

Q1. CMOS-N5 の 3.3V 動作回路に、BSCAN 搭載は可能ですか。

A1. はい、可能です。

CMOS-N5(一般)

Q1. CMOS-N5 の最高動作周波数は何 MHz ですか。

A1. 5V では 60MHz です。それ以上は、マイグレーションなどの問題があり保証できません。
また、3.3V では 33MHz、3.0V では 25MHz が目安です。消費電力がパッケージの許容値以内であることと、同時動作制限が設計マニュアルの基準を満たしていることが条件です。

CMOS-N5(一般、RAM、CPU 周辺)

Q1. CMOS-N5 で、CMOS 3.3V 対応はできますか。

A1. はい、できます。ただしメガマクロ、RAM は使用できません。NEC_SCAN,NEC_BSCAN は使用できます。

3. 表示・画像処理

3.1 文字表示 IC(OSD) …714
3.1.1 OSD 全般 …714
3.1.2 ビデオ系 OSD(μPD6464A, 6465) …717
3.1.3 RGB 系 OSD(μPD6461, 6462, 6466, 6467) …721
3.2 CCD …724

3.1 文字表示 IC(OSD)

3.1.1 OSD 全般

外部クロック入力

Q1. OSC_{IN} 端子に、外部クロック入力は可能ですか。

A1. 外部クロック入力モードなら可能です。次の場合、内部破壊の恐れがあるため、クロックは入力できません。

- μPD6466, 6467 の初期状態設定コマンドで、LC 発振を指定
- μPD6461, 6462 のマスク・オプションで、LC 発振を指定
- μPD6464A, 6465 を使用

OSD 用 LSI ユーザーズ・マニュアルの「3.3 LC 発振回路部への外部クロック強制入力」をご参照ください。

シリアル・データと Hsync、Vsync の同期

Q1. シリアル・データと Hsync,Vsync は、同期させる必要がありますか。

A1. シリアル・データと Hsync, Vsync は非同期で結構です。なお、VRAM 書き込みデータ転送時は、コマンド連続入力許可時間を守ってください。シリアル・データ転送後、そのデータが VRAM 書き込み関連なら、ドット・クロックで VRAM へデータが書き込まれ、これに要する時間が表示オフ時には「12/fosc × 文字サイズ」(μPD6461, 6462)となります。また、表示オン時には、Hsync 期間中はドット・クロックが停止し、次に再発振するまでに数クロックが必要なので、「21/fosc × 文字サイズ + t_{HWL}」(μPD6461, 6462)の時間が必要です。
コマンド連続入力許可時間については、各製品のデータ・シートで「コマンド連続入力許可時間について」(μPD6461, 6462, 6466, 6467)、「コマンド入力の BUSY 期間について」(μPD6464A, 6465)をご参照ください。

パワーオン・リセット

Q1. PCL 端子を V_{DD} に接続(電源投入時にハイ・レベル、t_{PCLL} 満足せず)し、電源投入後に PCL 端子をロウ・レベルに切り替えると、パワーオン・クリア実行までの出力はどうなりますか。

A1. パワーオン・クリアが実行されていないとき、何が出力されるかは予測できません。次に例を示します。

例 1. NTSC 使用時で、PAL の内部モードに設定していると、白黒斜め縞模様になる可能性があります。
2. VTR 系 OSD で外部映像信号モードに入った場合、表示オンになると、クリアされていない VRAM の内容に従って、異常出力をする可能性があります(背景についても同様)。

ただし、PCL 端子がロウ・レベルになれば、外部映像信号モードは表示オフとなり、異常出力は停止します。

Q2. 電源投入時、パワーオン・クリアまで不正データが出力されないように、内部スイッチなどはありますか。

A2. いいえ、OSD 内部にはスイッチのようなものはありません。
当社の OSD では、電源投入と同時にパワーオン・クリアをかけることを前提としており、パワーオン・クリアを実行すれば異常な出力をすることもないため、スイッチは必要ないと判断しています。

LC 発振定数

Q1. LC 発振定数の決め方を教えてください。

A1. OSD 用 LSI ユーザーズ・マニュアルの「2.1.1 ドット・クロック発振回路」をご参照ください。

初期化の違い

Q1. ビデオ RAM 一括解除コマンドとパワーオン・クリアによる初期化の違いはなんですか。

A1. パワーオン・クリアは、電源投入時に、IC 内部の不定状態(ハードウエア)を初期化します。
ビデオ RAM 一括解除コマンドは、パワーオン・クリア後に、他のコマンドで設定するレジスタ、カウンタやビデオ RAM の初期化などをします。なお、ビデオ RAM 一括解除コマンドは、テスト・モード解除はしません。

フィールド判別

Q1. フィールド判別は行っていますか。

A1. 当社の OSD では、V, ODD, EVEN などのフィールド判別は行っていません。
内部での表示文字の更新は、Hsync のタイミングで行います。

ビデオ RAM に設定されたデータ

Q1. ビデオ RAM に設定されたデータは、設定を変更しない限り保持されますか。

A1. OSD の内部にリフレッシュ・タイマを内蔵しているため、パワーオン・クリア、ビデオ RAM 一括解除コマンド、表示キャラクタ制御コマンド、電源オフのいずれかを実行しない限り、データは保持されます。

2 バイト連続コマンド

Q1. 2 バイト連続コマンドはどのような機能ですか。

A1. 各製品のデータ・シートで「コマンドの転送方法」の「2 バイト連続コマンド」をご参照ください。

Hsync とドット・クロックのタイミング

Q1. Hsync とドット・クロックのタイミングには、制限事項はありますか。

A1. いいえ、コマンド連続入力許可時間を満足していれば、特に制限事項はありません。

2 バイト連続コマンドの転送

Q1. 表示オン時のコマンド連続入力許可時間は、なぜ Hsync 同期信号幅を含むのですか。

A1. OSD では、表示オン時に Hsync で内部タイミングをリセットして、表示位置などを制御します。このため、LC 発振で Hsync 期間中は、内部回路へのドット・クロック供給を停止するため、ビデオ RAM へのデータ書き込みが行われないからです。Hsync 期間中でもコマンドは受け付ける（データ入力シフト・レジスタはドット・クロックを使用しない）ので転送できますが、コマンド連続許可時間を守ってください。

NTSC 信号入力時と同等に PAL 信号を入力

Q1. NTSC のように、水平・垂直表示開始位置とドット・クロック周波数を設定して、PAL 信号を入力すると、キャラクタの垂直サイズが縮んで、表示画面下部にスペースが発生します。

A1. 図 1 のように、NTSC と PAL の走査線数の違いが原因です。PAL 信号入力では、水平・垂直表示開始位置とドット・クロック周波数を調整して、表示領域を移動させる必要があります(信号方式単位に調整)。
なお、フレームあたり、NTSC で 53 本(最小文字のキャラクタで約 1.5 行に相当)、PAL で 143 本(最小文字のキャラクタで約 4 行に相当)のキャラクタ非表示領域が発生(図 2)するため、同等の表示は不可能です。

図 1　NTSC, PAL 信号入力時のキャラクタ表示領域イメージ(表示開始位置、ドット・クロック周波数が同等)

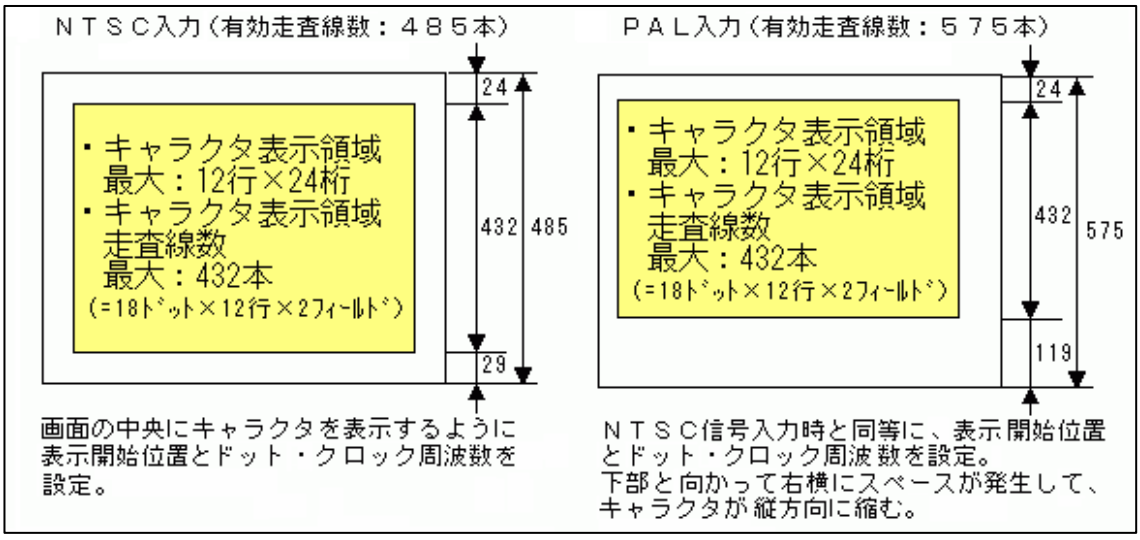
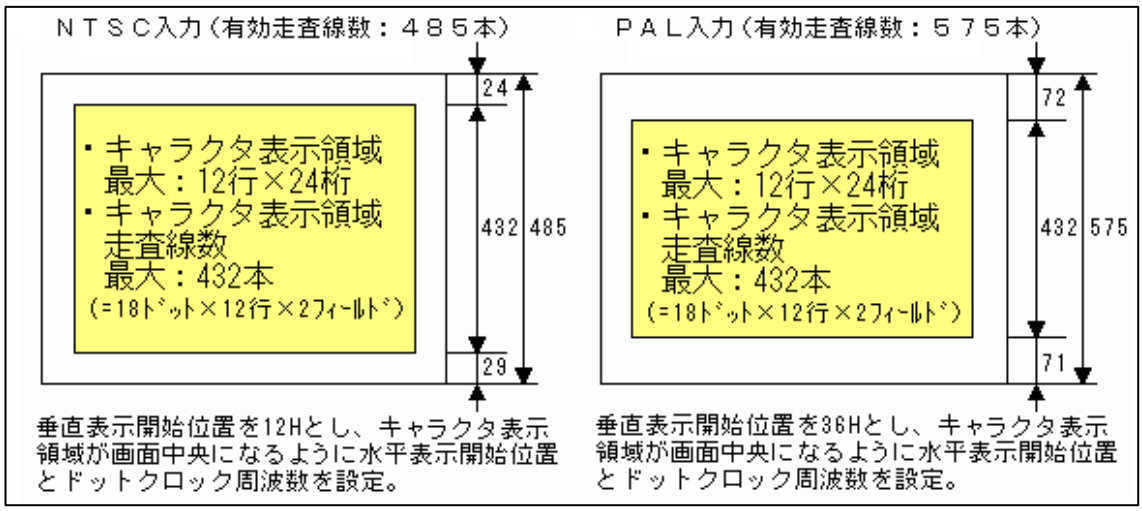


図 2　NTSC、PAL 信号入力時のキャラクタ表示領域イメージ(センタ表示)



標準品

Q1. μPD645x/6x シリーズ OSD 製品で、標準品とはなんですか。

A1. 内蔵 ROM(キャラクタ・ジェネレータ ROM)に、データ・シート記載の標準キャラクタ・パターンが書き込まれている製品です。お客様がパターンを作成する必要はありません。標準品の ROM コードは次のとおりです。

μPD6450GT-102　μPD6464AGT-101　μPD6465GT-101　μPD6466GS-001　μPD6467GR-001

Q2. すべての OSD 製品で、標準キャラクタ・パターンは同じですか。

A2. いいえ、多少違いますので、各製品のデータ・シートをご確認ください。

Q3. ユーザ作成のキャラクタ・パターンを内蔵 ROM に書き込んでもらうことはできますか。

A3. 現在は、μPD6467GR だけ対応できます。

VRAM(ビデオ RAM)書き込みコマンド

Q1. 電気的特性の「コマンド連続許可時間について」という規格に、VRAM 書き込みコマンドという記述がありますが、コマンド一覧には載っていません。どういうコマンドですか。

A1. VRAM 書き込みコマンドとは、ビデオ RAM に書き込むコマンドの総称で、ビデオ RAM 一括解除コマンドと表示キャラクタ制御コマンドの 2 つのコマンドを指します。VRAM 書き込みでドット・クロックを使用するので、コマンドを連続転送する時間が、ドット・クロックで規定されています。特に、表示キャラクタ制御コマンドは 2 バイト連続コマンドなので、この規格に注意が必要です(「2 バイト連続コマンドの転送」参照)。

3.1.2 ビデオ系 OSD(μPD6464A, 6465)

水晶発振子の必要性

Q1. 水晶振動子は必要ですか。

A1. 発振方式制御コマンドで、4fsc 水晶発振を選択すると必要です。これは、振動子と VCO で生成される 4fsc 信号を 4 分周した fsc 信号が、同期信号分離回路(外部映像信号モード時)と内部映像信号発生用の基本クロックににになるためです。キャラクタ表示時、水晶発振制御コマンドは必ず水晶発振 ON としてください。

μPD6464A:必要な周波数精度

Q1. 14.31818MHz の周波数の精度は、どのくらい必要ですか。

A1. ±500Hz 程度なら十分です。放送規格のサブキャリア周波数 3.579545MHz は±20Hz 程度ですが、一般民生用では±100Hz ぐらいです。ただし、サブキャリアが問題となるのは、内部映像信号を利用する場合です。外部映像信号モードでは、同期分離回路のクロックとするだけなので、さらに甘くなり、±数百 Hz です。

Q2. 外部映像信号とキャラクタを重ねて表示すると、キャラクタの位置がずれます。映像信号は正常です。

A2. 同期信号に問題があると推測されます。μPD6464A を外部映像信号モードで使用する場合は、映像信号に同期した複合同期信号を CSYIN 端子(17pin)に入力する必要があります。このとき、外部映像信号とキャラクタ信号の同期がずれる原因として、次の 2 つが考えられます。

(1) CSYIN に入力する**同期信号自体**に不具合がある

・CSYIN 信号を観測して、レベルや位相などに異常がないかご確認ください。

・CSYIN 信号にノイズが乗っていないかどうかご確認ください。

μPD6464A は、0.2us 以上のパルスを Hsync と認識します。したがって、0.2us 以上のノイズ・パルスが CSYIN 信号に混入すると、Hsync と解釈するので、垂直位置カウンタが進んでしまいます。

(2) μPD6464A 内部での同期分離動作ができない。

外部映像信号モードの場合、CSYIN に入力した複合映像信号を同期信号分離回路ブロックで、水平同期信号と垂直同期信号に分離します。分離用のクロックには、4fsc 発振回路でカウント・ダウンしたクロックを使用します。したがって、4fsc 発振回路動作に異常があると、同期分離動作が異常となります

不具合現象発生時の 4fsc 発振に、異常がないかどうかをご確認ください。

Q3. 実験で 1Vp-p 信号を入力すると、そのまま出力されるので、このまま使っていいですか。

A3. いいえ、だめです。外部映像信号モードで入力信号を 1Vp-p にする場合、内部映像信号(キャラクタ信号)のレベルも合わせる必要があるので、次の設定をしてください。

(1) 内部映像信号振幅制御ビット：VPD＝0

(2) V_{CNT} 端子印加電圧：2.5V

(1)はコマンド変更ですが、(2)はハード的な対応が必要で、次の 2 つの方法が考えられます。

・外部から強制的に V_{CNT} 端子を 2.5V にする。

・V_{CNT}-V_{DD}(+5V)間に抵抗を挿入し、V_{CNT} 端子電圧が 2.5V となるよう調節する。

この端子(キャラクタ信号)のレベルは、V_{CNT} 端子と GND 間に設置する分圧抵抗で決まります。

参考資料：OSD 用 LSI ユーザーズ・マニュアル の「2.2.3 映像信号出力部」(図 2-9)

fsc 入力の必要性

Q1. fsc 入力は必要ですか。

A1. 発振方式制御コマンドで 4 逓倍発振を選択すると必要です。これは、fsc 入力から生成される 4fsc 信号を 4 分周した fsc 信号が、同期信号分離回路(外部映像信号モード時)と内部映像信号発生用の基本クロックになるためです。キャラクタ表示時、水晶発振制御コマンドは、必ず水晶発振 ON としてください。

μPD6464A, 6465 は 4 逓倍回路を内蔵し、水晶発振でなく、LC 発振で低価格化と実装面積の縮小化を図れます。LC 発振は Q が小さく、安定度は低いですが、PLL への fsc 入力で、4fsc で安定発振ができます。

fsc 入力を使用しないときの FSCI 端子の処理

Q1. fsc 入力を使用しないとき、FSCI 端子(9 ピン)はどのように処理すればいいですか。

A1. GND か V_{DD} に接続してください。また、発振制御コマンドは 4fsc 水晶発振を選択してください。

13～18 ピンを使用しない場合の端子の処理

Q1. 13～18 番ピンを使用しない場合、どのように処理すればいいですか。

A1. オープンにしてください。

SECAM 非使用時の SECAM 端子の処理

Q1. 映像信号方式制御コマンドで SECAM を選択しないとき、SECAM 端子(22 ピン)はどうすればいいですか。

A1. SECAM 端子は、SECAM 以外では内部回路と切り離されるため、何かの接続でもオープンでもけっこうです。
何か接続する場合は、入力電圧を絶対最大定格(V_{IN}=−0.3～V_{DD}+0.3V)の範囲内にしてください。

内部映像信号だけ使用

Q1. μPD6464A, 6465 で内部映像信号だけを使用する場合、複合同期信号(CSYNC)の入力は必要ですか。

A1. 複合同期信号(CSYNC)の入力は必要ありません。内部映像信号モード時は、4fsc 水晶発振か 4 逓倍発振で生成される 4fsc 信号を用いて、同期信号を自己生成します。

内部映像信号モード時の走査方式

Q1. 内部映像信号モード時の走査方式は、どのようになりますか。

A1. ノンインタレースです。水平走査線は、フィールドあたり NTSC/PAL-M で 263 本、PAL/PAL-N で 312 本です。

CSYIN 端子へ入力する複合同期信号(Csync)のレベル

Q1. 外部映像入力モード時に、CSYIN 端子(17 ピン)に入力する複合同期信号(Csync)は何 V ですか。

A1. CSYIN 端子は通常のロジック入力です。したがって、入力レベルは次のような CMOS 入力の規定です。
入力ハイ・レベル電圧：MIN.0.7V_{DD}　　入力ロウ・レベル電圧：MAX.0.3V_{DD}

表示 OFF での VBSI→VBSO の状態

Q1. 表示 OFF にすると、VBSI→VBSO の状態はデータ・スルーとなりますか。

A1. 外部映像信号モード時は、表示 OFF でデータ・スルーとなります。
また、内部映像信号モード時は、画面背景色として指定した色の、単一画面色表示となります。

外部映像信号モード時の HSYO、VSYO の出力状態

Q1. 外部映像信号モード時に、CSYNC 端子(17 ピン)がハイ・レベルの場合、HSYO 端子(15 ピン：水平同期信号出力)、VSYO 端子(16 ピン：垂直同期信号出力)の出力状態はどうなりますか。

A1. HSYO 端子、VSYO 端子は、ともにロウ・レベルのままです。

Q2. Q1 の状態で 2 バイト連続コマンドを転送する場合、コマンド連続入力許可時間(TB2、TB2')を守れば、ビデオ RAM に正しく書き込まれますか。

A2. いいえ、2 バイト連続コマンド転送する場合、CSYNC 端子がハイ・レベルでは、コマンド連続入力許可時間の T_{HWL1} が無限大となり、表示 ON 時は書き込みできません。これは、LC 発振が停止し、書き込み用のクロックが供給されないからです。ビデオ RAM への書き込みは、表示 ON 時に停止し、表示 OFF 時に動作します。

VBSO 端子のドライブ能力

Q1. VBSO 端子(21 ピン:複合映像信号出力)のドライブ能力は、どのくらいですか。

A1. VBSO 端子は、VBSI 端子(複合映像信号入力)に入力された映像信号をアナログ・スイッチでスルー出力するので、ドライブ能力は高くありません。トランジスタのエミッタ・フォロワ回路などを付加してください。

参考資料：OSD 用 LSI ユーザーズ・マニュアルの「3.1.1 μPD6464A/6465 応用回路例」

バラクタ・ダイオードの選択

Q1. バラクタ・ダイオードは、どのように選択すればいいですか。

A1. 1～4V 程度の電圧変動で、2,3～10pF くらいの範囲で容量が変化する程度のものを選んでください。
なお、バラクタ・ダイオード変更時には、12 ピンに接続するコンデンサの容量を十分にご評価ください。

μPD6464 と μPD6464A の違い

Q1. μPD6464 と μPD6464A は、なにが違うのですか。

A1. μPD6464A は μPD6464 の対応映像信号方式(NTSC、PAL、PAL-M、SECAM)に PAL-N を追加しています。

PCL 端子のプルアップ抵抗の必要性

Q1. PCL 端子(7 ピン)は、プルアップ抵抗が必要ですか。

A1. いいえ、不要です。内部に約 50kΩ のプルアップ抵抗を内蔵しています。

画面上にキャラクタが表示されない

Q1. パワーオン・クリア後、表示 ON にして各種コマンドを転送しても、画面上にキャラクタが全く表示されません。

A1. 以下の内容をご確認ください。

(1) ドット・クロック発振の有無

OSD では、ドット・クロックを用いてビデオ RAM へのデータ書き込みを行います。ドット・クロックの発振が停止状態では、転送したはずのデータがビデオ RAM に書き込まれず、キャラクタが表示されないことがあります。

(2) 複合同期信号(CSYNC)入力の有無

外部映像信号モードでは、タイミング・ジェネレータが複合同期信号 CSYNC を分離した水平同期信号と垂直同期信号を用い、水平制御部、垂直制御部、出力コントロール部のリセット、カウント用の基準信号を生成します。CSYNC 入力がないと、これらの基準信号が生成されないため、キャラクタが表示されないことがあります。なお、内部映像信号モードでは、内部で水平同期信号と垂直同期信号を生成するため、CSYNC 入力がなくても、キャラクタは表示されます。

(3) 4fsc 発振の有無

4fsc 水晶発振か 4 逓倍発振で生成された 4fsc、fsc 信号を用いて、複合同期信号 CSYNC の分離(外部映像信号モード時)か、内部映像信号と内部同期信号の生成(内部映像信号モード時)が行われます。4fsc 発振が行われていないと、これらの動作が行われないため、キャラクタが表示されないことがあります。

コマンド連続入力許可時間

Q1. μPD6464A、6465 のデータ・シートの「2 バイト連続コマンド」の説明で、コマンド連続入力許可時間の「TB1'＋(21/fosc)×S_1+T_{HWL1}」の S_1 はどのような値ですか。

A1. この時間は、2 バイト連続コマンド入力のビジィ期間で、「TB1＋(21/fosc)×S_1+T_{HWL1}」の S_1 や、「(21/fosc)×S_2+T_{HWL2}」の S_2 はキャラクタ・サイズで、1(倍)か 2(倍)のどちらかです。

なお、電気的特性の推奨動作条件で、クロック周期(t_{TCK})は min.1 μs になっており、8 クロックでは min.8 μs 必要になります。TB2 や TB2'の計算結果が「TB1＋(CLK 入力の7クロック周期：min.7 μs)」より大きければ、TB2 や TB2'の計算結果をそのままご使用ください。ただし、計算結果が「TB1＋(CLK 入力の7クロック周期 : min.7 μs)」より小さい場合は、この値をご使用くださいます。

3.1.3 RGB 系 OSD(μPD6461, 6462, 6466, 6467)

μPD6466, 6467 でのドット・クロック確認

Q1. μPD6466 では、ドット・クロックをどのように確認すればいいですか。

A1. μPD6461, 6462 は、ドット・クロックを CLKout 端子で観測できますが、μPD6466 は専用端子がありません。TEST 端子を V_{DD} に接続し、次のテスト・モード・コマンドを転送すれば、BLK1 端子に出力できます。

2 バイト・コマンド(MSB ファースト)

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
1	0	1	1	0	0	0	0	0	0	0	0	1	1	0	0

2 バイト・コマンド(LSB ファースト)

D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14	D15
0	0	0	0	1	1	0	1	0	0	1	1	0	0	0	0

注意 ドット・クロックの確認以外には、これらのコマンドを使用しないでください。

μPD6466, 6467：チャネルを切り替えると、文字位置がずれる

Q1. 2 台のカメラを切り替えると、個別に OSD で表示している文字位置がずれます。合わせられますか。

A1. できますが、カメラの同期状況で方法は異なります。

(1) カメラ間が同期している場合

水平同期信号に同期した外部クロックを各デバイスのドット・クロックにすれば、チャネルを切り替えてもキャラクタ位置はずれません。クロック入力にするには、初期状態設定コマンドでドット・クロック制御ビット(OSC)を"1"にします。クロックは OSCIN 端子に入力し、OSCOUT 端子はオープンにしてください。

(2) カメラ間が非同期の場合

チャネルごとに、VCXO などを用いて水平同期信号を基準とした PLL を構成し、水平同期信号に対するドット・クロックの位相と周波数をチャネル間でそろえる回路が必要となります。回路規模は大きくなります。

外部クロック入力タイミングで t_{H-C} が 30ns より小さい場合

Q1. 外部クロック入力タイミングで、t_{H-C} が MIN.の 30ns より小さい場合(たとえば Hsync の立ち上がりエッジと外部クロックの立ち下がりエッジが重なる：t_{H-C}≒0ns)、どのような現象になりますか。

A1. 当社 OSD は、Hsync を水平制御カウンタのリセットに、外部クロックを水平制御カウンタのクロックに使用しています。t_{H-C}＝MIN. 30ns を満足しないと、Hsync とクロックが水平制御カウンタに到達する時間差がなく、水平制御カウンタ内で両エッジが重なる可能性があります。このとき、Hsync でのカウンタ・リセットの解除とカウント・アップのタイミングが重なり、クロック・エッジの計数が不確定です。Hsync もクロックもジッタがあり、エッジ・カウントの有無を繰り返して、水平の表示位置が 1 クロック分ずれます(水平方向に 1 ドット分のジッタ)。

同期保護回路の動作

Q1. 同期保護回路は、どのような動作をするのですか。

A1. OSD 用 LSI ユーザーズ・マニュアルの「2.1.3 同期保護回路」をご参照ください。

ドット・クロックを外部入力に設定したときの LC 発振

Q1. ドット・クロックに外部クロック入力を選択して、LC 発振はできますか。

A1. いいえ、できません。外部クロック入力オプションを選択すると、LC 発振回路の発振段が端子から切り離され(OSD 用 LSI ユーザーズ・マニュアルの図 2-2(b)参照)、LC 発振子を外付けしても発振しません。

外部クロック選択時に入力するクロック

Q1. 外部クロックとして、どのようなクロックを入力すればいいですか。

A1. 次のようなクロックを目安にしてください。

入力ハイ・レベル電圧 : MIN. 0.7V_{DD}

入力ロウ・レベル電圧 : MAX. 0.3V_{DD}

デューティ比 : TYP. 50% (ばらつきは 40～60%以内)

なお、次の値については、データ・シートで規格をご参照ください。

外部クロック立ち下がり→同期信号立ち上がり時間(t_{C-H})

同期信号立ち上がり時間→外部クロック立ち下がり(t_{H-C})

立ち上がりスルーレート(ts)

各出力端子の構成

Q1. 各出力端子は、どのような構造ですか。

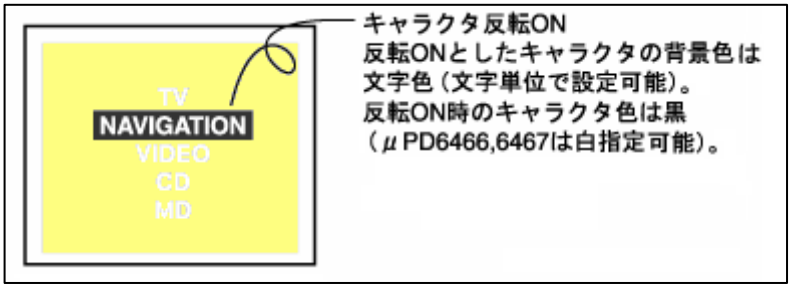
A1. CMOS 構造です。出力インピーダンスは約 100Ω 以下です。

特定領域の背景色を変更

Q1. 特定領域の背景色を変更することはできますか。

A1. はい、できます。背景色の設定は画面単位なので、キャラクタ(色)反転 ON/OFF 指定コマンドと表示キャラクタ制御コマンド(キャラクタ単位で指定)のキャラクタ(色)反転指定ビットを ON に設定してください

キャラクタ反転使用表示例



キャラクタの表示が乱れたり、行われなかったりする

Q1. キャラクタの表示が乱れたり、行われなかったりします。

A1. 以下の内容をご確認ください。

(1) ドット・クロック発振の有無

OSD では、ドット・クロックを用いてビデオ RAM へのデータ書き込みを行います。ドット・クロックの発振が停止状態では、転送したはずのデータがビデオ RAM に書き込まれず、キャラクタが表示されないことがあります。

(2) Hsync,Vsync 入力の有無

タイミング・ジェネレータは、入力した Hsync と Vsync を用いて、水平制御部、垂直制御部、出力コントロール部のリセット、カウント用の基準信号を生成します。Hsync、Vsync が入力されないと、タイミング・ジェネレータで基準信号が生成されないので、キャラクタが表示されないことがあります。

(3) コマンド連続入力許可時間の厳守

コマンド連続入力許可時間を守らないと、転送したはずのデータがビデオ RAM に書き込まれず、キャラクタの表示が乱れることがあります。T2 の時間をご確認ください。

キャラクタ書き込みコマンド(2 バイト連続コマンド)のコマンド連続入力許可時間は、T1、T2 が適用されます。T2 の値は、モード(表示 ON／OFF)、文字サイズ、ドット・クロック周期によって異なります。

参考資料：μPD6461,6462 データ・シート p.54「コマンド連続入力許可時間について」

μPD6467:プログレッシブの映像信号、1125HD 方式への対応

Q1. μPD6467 は、プログレッシブの映像信号に対応できますか。

A1. はい、インターレスとノンインターレスのどちらでも対応できます(フィールド判別をしていません)。

ただし、倍速スキャンでは、使用するドット・クロック周波数が推奨動作周波数範囲を越えることがあります。

実力的には、15MHz 付近まで延びていますが、使用に際しては十分にご評価ください。

Q2. μPD6467 は 1125HD 方式に対応していますか。

A2. 動作はしますが、次のような制限があります。

(1) ドット・クロック周波数が、推奨動作範囲を越えないかどうか？

ドット・クロック推奨周波数は最大 8MHz です。実力は 15MHz 近辺ですが、出荷検査は 8MHz で行っておりますので、これ以上での保証はできません。

(2) キャラクタの不自然さ(縦横比)

キャラクタのドット構成などは、通常のテレビ信号である NTSC, PAL 方式(EIA, CCIR)を想定しています。

1125HD は NTSC/PAL と走査線構造が異なるため、そのまま出力すると、縦方向につぶれた形になります。

μPD6466, 6467 には、キャラクタ・サイズ縦横独立設定機能(1～4 倍)がありますので、水平方向ドット・クロックに応じて縦サイズ設定を行えば、対策できる場合もあります。

キャラクタ出力は、OSD に入力する Hsync、Vsync に同期しますので、基本的には走査線数によって動作が制限されることはありませんが、1125HD 方式は本来の目的とは異なるため、事前に十分ご確認ください。

3.2 CCD

μPD3728:画素の不均一性

Q1. 均一な光を CCD に入射すると、偶数(EVEN)か奇数(ODD)画素の出力が、3 画素ほど連続して低くなることがあります。連続した不均一な画素は、最大何画素続くかの規定はありますか。

A1. 連続した不均一な画素数の規定はありません。データ・シート記載のとおり、出力電圧の不均一性(PRNU)は、全有効画素内で不均一な画素の最大か最小のレベルを平均出力レベルの百分率で規定しています。

デバイス自体で PRNU を悪化させる要因は、オンチップ・フィルタの塗布むらなどですが、これらは PRNU の規格で選別しており、不均一レベルが規格外のものは出荷しません。だたし、連続画素の規定がないので、不均一画素の出力レベルが規格内なら、連続する可能性はあります。

デバイスの外的要因として、ガラス・キャップにキズがついたり、ゴミなどが付着した場合が考えらます。キズやゴミの大きさにより、連続して出力が不均一になる可能性があります。

μPD3734A:センサ・カバー

Q1. 保護シールのようなカバーが、センサ部分に貼り付けられていますが、使用時は取り外すのですか。

A1. はい、外してください。プラスチック・パッケージの CCD には、保護シールがついています。μPD3734ACY は保護シールなしも受注できますが、傷がつきやすいので保護シール付きをお勧めします。

μPD3734A:ブルーミング

Q1. スポット状の高輝度画像が入力されると、周辺画素まで飽和するブルーミングが発生します。飽和露光量の範囲を越えると、どの程度でブルーミングは発生しますか。飽和露光量の最大値の規定はありますか。

A1. どのレベルでブルーミングが発生するかは分かりません。出力電圧が飽和しないレベル(飽和出力電圧 MIN 以下)でご使用ください。飽和露光量の最大値は規定していません。

Q2. 露光時間の短縮など、感度を落とさないでブルーミングを低減することはできますか。

A2. いいえ、ブルーミングを防ぐには、光源の光量を低減するか、蓄積時間を短くして、CCD が飽和しない範囲(飽和出力電圧 MIN 以下)でご使用ください。

μPD3734A:画素ゆらぎ

Q1. 画素ゆらぎの特性の定義と最大値を教えてください。

A1. 「画素ゆらぎ」はランダム・ノイズの最大偏差を表しており、5 倍のランダム・ノイズ(標準偏差)で定義します。

画素揺らぎ(BN)≒5×ランダム・ノイズ(σ)

ランダム・ノイズについては、データ・シートの p.12 をご参照ください。「ランダム・ノイズ」と「画素ゆらぎ」は、測定条件(計算以外)が同じです。画素ゆらぎの規格は参考値で、TYP 値だけ記載しています。MAX 規定はありません。ばらつきの実力値は、約 9～11mV_{P-P} です。

μPD3734A:パルス停止

Q1. CCD の電源を入れたまま、シフト・レジスタ・クロックなどのパルスを止めてもいいですか。
また、パルスを復帰させたあとの動作は、どうなりますか。

A1. 入力パルスを止めても、デバイス破壊やラッチアップは起きません。
ただし、パルスを入力していないと、CCD は電荷が溢れた状態になります。したがって、パルスの復帰後は、十数～数十走査分の駆動パルスを与えてからでないと、正常な出力が得られません。有効な出力が得られるまでの走査回数は、パルス入力が止まってから復帰するまでに、製品が置かれていた状態(光入力有無、時間)によるため、規定はできません。十分にご評価のうえ、余裕のある設計をしてください。

μPD3734A:φTG 信号がハイ・レベルの期間の φRB、φSHB

Q1. φTG 信号がハイ・レベルの期間は、φRB、φSHB 信号を与える必要がありますか。

A1. はい、回路動作を正常に動作させるには、φTG 信号がハイ・レベルの期間も、φRB、φSHB が必要です。
φRB、φSHB は、基本的に φTG 信号がハイ・レベルの期間も連続した波形にしてください(データ・シートに記載のとおり、常に同じ信号が連続)。タイミングについては、φTG 信号のハイ・レベル期間もデータ・シート記載のタイミングを守ってください(φ1 と φ2 とのタイミング(φ1、φ2 停止前後)も満足)。

Q2. φTG 信号がハイ・レベルの期間は、φRB、φSHB に 3 パルス以上を与えると、正常に動作しませんか。

A2. いいえ、データ・レートの範囲内の周波数なら、何パルスでも問題ありません。

μPD3734A:出力の画素番号の基準

Q1. 出力の画素番号の基準は、なにで決まりますか。

A1. データ・シート p.7 のタイミング・チャート記載のように、有効フォト・セル 2660 画素の最初が、画素番号の基準になります。物理的な第一画素の位置は、データ・シート p.15 の外形図に記載しています。

μPD3739：温度-受光出力特性データ

Q1. 温度-受光出力の特性データはありますか。

A1. 2 つのサンプルの評価データを参考に示します。保証値ではありません。

	サンプル 1			サンプル 2			単位
パッケージ温度	25	45	61	25	45	61	℃
感度	8.98	9.07	9.19	8.75	8.94	9.02	V/lx・s
感度変化率注	100.0	101.0	102.3	100.0	102.2	103.1	%

条件：データ・レート＝2MHz(1MHz/ch)

V_{OD}＝12V

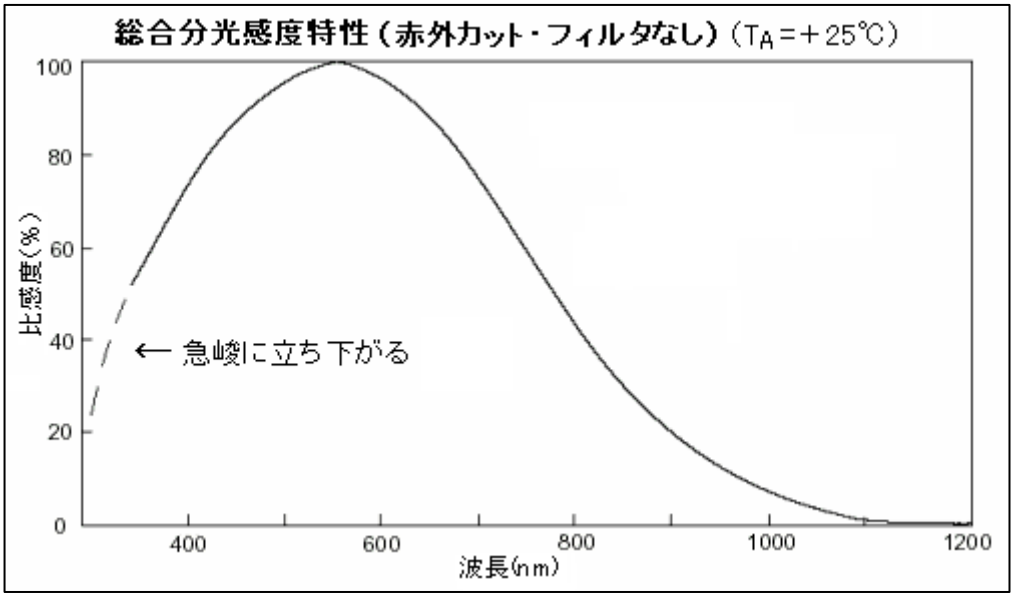
入力クロック＝$5V_{P-P}$

昼光色蛍光灯感度

注　25℃での値を 100%とした値。

Q2. データ・シートの感度特性のグラフは、波長が 350nm までですが、これ以下ではどのようになりますか。

A2. 分光感度特性は、光の波長に対する信号出力の特性を示し、波長が長いほど CCD の深部で吸収され、短いほど表面付近で吸収されます。不可視光は十分に電荷に変換できず、一般的な構造の CCD では感度がなくなります。一般的な立ち下がりは、次図のように、長波長側は緩やかで、短波長側は急峻です。



μPD3739：ランダム・ノイズ

Q1. ランダム・ノイズは、光遮断（暗）時より光入射（明）時の方が増加しますか。

A1. はい、光入射時のランダム・ノイズは、次の(1)と(2)に起因するため、遮断時より増加します。

(1) アンプ・ノイズ成分(アンプを構成するMOSトランジスタのノイズ)

(2) 信号電荷の揺れ成分(光電変換される電子数のばらつき)

暗時は(1)、明時は(1)＋(2)で、光入射時の方が増加します。なお、(1)は信号量によらず一定です。また、(2)は信号電荷量(信号量)のルートに比例します。次に、参考データを示します(定義はデータ・シート参照)。

信号量(V)	0	0.3	0.5	1.0
ランダム・ノイズ(mV)	0.7	3.3	4.1	5.0

暗時は(1)の成分だけですが、明時は(2)の成分が支配的になります。各値は、測定系のノイズによっても変わります。また、(2)の成分には当然、光源の揺らぎ成分が加わります。
なお、蓄積時間は、信号量と関係があります。したがって、(2)は蓄積時間に関係します。蓄積時間が 2 倍になれば、信号量が 2 倍になりますので、(2)の成分はルート 2 倍になります。

μPD3739:クロックのクロス・ポイントのずれの影響

Q1. シフト・レジスタ・クロック φ11、φ21 などのクロス・ポイントがずれると、どのような影響がありますか。
また、微調整は抵抗だけで行うのですか。

A1. 低い電位でのクロスは、CCD の転送効率(TTE)の悪化や、転送したキャリアの逆戻りなどの原因となります。
また、Vout1(奇数)の出力と Vout2(偶数)の出力電位差の原因ともなります。

①TTE 不良の原因

クロックのハイ・レベル幅が短くなって、転送が間に合わなくなり、実効的に周波数が制限を越える場合があります。この場合、TTE 不良となる可能性があります。十分に低い周波数であれば、問題ありません。

②逆戻りの原因

CCD レジスタは、疑似三相を実現するため、バリアとストレージの 2 枚のゲートで各ビットを構成しています。このバリアとストレージの電位差は、ロウ・レベルの方が小さくなるので、飽和出力に近い信号量があると、電荷が逆方向にこぼれる可能性があります。もちろん、先送りの可能性もあります。

③先送りの原因

CCD シフト・レジスタには、最終段に OG(アウトプット・ゲート)というゲートがあります。ラスト・ゲートとその前のゲート(たとえばφ2 とφ1L)のクロス・ポイントがロウ・クロスになると、φ2 の下にたまっている電荷の一部がφ1L を通過して、OG を通って出力段の Floating Capacitor に入る先送り現象が生じます(本来はφ1L にたまるべき電荷)。

微調整は、抵抗(直列)だけで行うことを推奨しています。

μPD3739:出力電位の差

Q1. CCD の出力 Vout1(奇数)と Vout2(偶数)に、約 100mV の電位差(対 GND)があるのは、なぜですか。

A1. CCD の出力 $V_{OUT}1$ と $V_{OUT}2$ の電位差 100mV というのが、オフセット・レベルということなら、あり得ます。
しかし、信号レベルなら、レジスタ・インバランスという規格(V_{OUT}=500mV で Max4%)があり、100mV の差は大きすぎます。レジスタ・インバランスは、データ・シートの p.14 に定義を記載していますが、一様な光を当てたときの偶数、奇数画素出力の平均値の差と、全有効画素の平均出力電圧との比です。
レジスタ・インバランスの不良は考えられませんが、シフト・レジスタ・クロックのクロス・ポイントの状態で、悪化する可能性があります。

μPD3739:リセット・ゲート・クロック φR1 信号

Q1. リセット・ゲート・クロック φR1 信号は、何に使用しているのですか。

A1. φR1(リセット・パルス)は、CCD の動作上不可欠なパルスで、その名のとおり、光により変わる各画素の信号レベルをリセットするパルスです。このパルスが細くなると、リセット不良となって、信号レベル、リニアリティの悪化や、解像度(MTF)の劣化などが生じる恐れがあります。

Q2. φR1 のパルス幅が 12.5ns しかないので、立ち上がりエッジを前に動かしてもいいですか。

A2. いいえ、だめです。φR1 の立ち上がりを前にすると、リセット期間が信号出力期間にかかってしまって、信号出力がとれなくなる可能性があります。立ち下りエッジ側を規格の 15ns 以上になるよう広げてください。

μPD3739:φR レベル適正化回路

Q1. φR レベル適正化回路とは、どのようなものですか。

A1. φR レベル適正化回路は、リセット・ゲート・クロック入力(φR) を CCD の内部でリセットをかけるための、適正レベルにシフトする回路です。

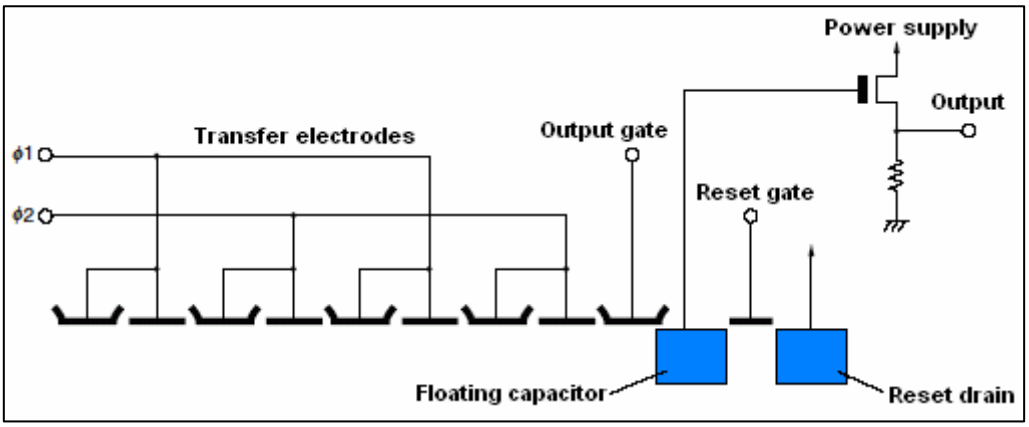
μPD3739:出力を一方しか使用しない場合

Q1. 奇数と偶数の 2 つの出力がありますが、一方しか使用しない場合、もう一方の出力のシフト・レジスタには、クロックを入力しなくてもいいですか。

A1. いいえ、クロックを入力してください。μPD3739 は、1 列のフォトセルを 2 つの CCD アナログ・シフト・レジスタに分けて出力する構成です。未使用な出力側のクロックを入力しないと、使用しないフォトセルに、常に電荷が溜まります。その結果、電荷がフォトセルから溢れて、隣接画素に流れ込むなど悪影響が考えられます。したがって、片側の出力だけ使用する場合でも、データ・シート記載のとおり、全信号を入力してください。

Q2. 1 出力が未使用でも、出力のシフト・レジスタにクロックを入力すれば、電荷はあふれないのですか。

A2. 出力部は下図のように、φ1、φ2 のクロックで転送されてきた電荷を電圧に変換して出力しています。電圧に変換後、電荷はリセット・ゲートのクロックにリセットされるので、電荷が蓄積されることはありません。したがって、出力はオープンでかまいません。



μPD3739:一部の素子で使用可能か

Q1. リフレッシュのレートを上げるため、1500 素子程度で使用できますか。

A1. 具体的なタイミングが不明なので断言できませんが、5000 画素の転送を終える前に、φTG を入力して 1 ラインの時間を短くすることは推奨できません。このような場合、使用していない画素の出力と、使用する 1500 画素の出力が混ざるためです。

μPD3739:φ1L2,φ2L1 端子の駆動

Q1. データ・シート p.17 の備考 2 に記述されている「17,6 番端子(φ1L2, φ2L1)はそれぞれ、10, 13 番端子(φ11, φ12)および 9,14 番端子(φ21, φ22)とは別のインバータで単独に駆動することをお勧めします」とは、別 IC で駆動するという意味ですか。同一 IC の別ゲートでもいいですか。

A1. 別 IC で駆動してください。シフト・レジスタ・クロック 1,2(φ11, φ12, φ21, φ22)の端子は、入力端子容量が大きいのでドライブ電流が多くなり、これらの端子のドライバと最終段シフト・レジスタ・クロック(φ1L2, φ2L1)のドライバを同一パッケージにすると、 CCD 出力の安定期間を決める最終段シフト・レジスタ・クロックが、シフト・レジスタ・クロックの影響を受けるため好ましくありません。
なお、応用回路例ではインバータを点線で囲っていますが、便宜上、インバータの電源、GND の接続を示すためです。

Q2. 全シフト・レジスタ・クロック 1,2(φ11, φ12, φ21, φ22)を同一パッケージのドライバで駆動していいですか。

A2. 規格を満たしていれば、CCD としては特に問題ありません。
なお、シフト・レジスタ・クロック端子は CCD パッケージの両側に分かれているため、シフト・レジスタ・クロック 1,2 を単一パッケージのドライバで駆動した場合、インバータ出力の基板パターン引き回しによっては、特に高周波において、クロスポイントの規格を満たすことが厳しくなる可能性があります。

Q3. シフト・レジスタ・クロック 1,2(φ11, φ12, φ21, φ22)と φTG, φR1, φR2 も、別パッケージとすべきですか。

A3. いいえ、上記 A1 以外は、特に推奨はしていません。シフト・レジスタ・クロック 1,2 と同一パッケージにするかどうかは、基本的にはシフト・レジスタ・クロックからのノイズの影響度合いによります。つまり、お客様の設計されたパターンなどの環境にもよりますので、一概には言えません。

μPD3739:データ・シート上の「50%, 90%, 10%」の電圧

Q1. データ・シート p.8 のタイミング・チャート 2 などで、50%, 90%, 10%とは、2.5V, 4.5V, 0.5V の意味ですか、それとも、実際の駆動波形の H レベルと L レベル間の電位差に対する%ですか。

A1. 後者の、実際の駆動波形(推奨動作範囲内)の H レベルと L レベル間の、電位差に対する比率です。

μPD3739:t1 などの MAX 規定がない

Q1. データ・シート p.9 に記載されている t1 などは、MIN, TYP だけの記載で、MAX の記載がないものは、TYP を越えて使用していいですか。

A1. 基本的に問題ありません。ただし、最終段シフト・レジスタ波形の立ち下りが遅いと、タイミング・チャート 2 のように、CCD 出力の安定期間が短くなるので、極力なまらないようにしてください。

μPD3739：蓄積時間

Q1. 蓄積時間は、タイミングの図でどの部分ですか。

A1. φTG の繰り返し周期が、蓄積時間です。厳密には、φTG の立ち下がりから、次の立ち下がりまでです。

Q2. 蓄積された電荷が、放電するタイミングはいつですか。

A2. φTG がハイ・レベルの期間です。

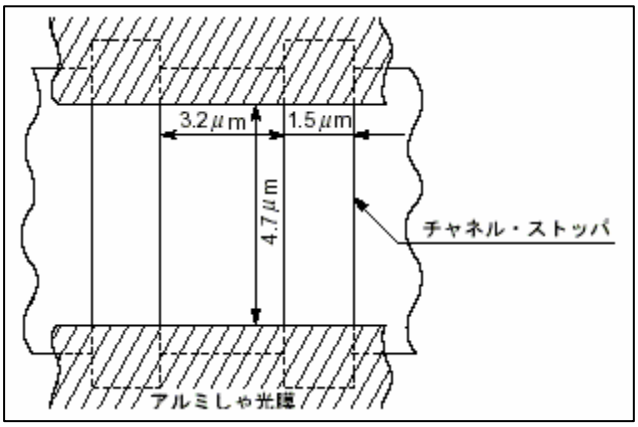
Q3. 蓄積時間が 100ms でも、問題はないですか。

A3. 蓄積時間だけでの規格はないですが、飽和出力電圧の規格があり、これは露光量、つまり蓄積時間×照度に依存します。飽和出力電圧を越えると、感度の直線性が失われます。また、暗時出力不均一性(DSNU値)は、蓄積時間に比例して大きくなります。電気的特性の暗時出力不均一性(DSNU値)の規格値は、蓄積時間10msでの値になります。

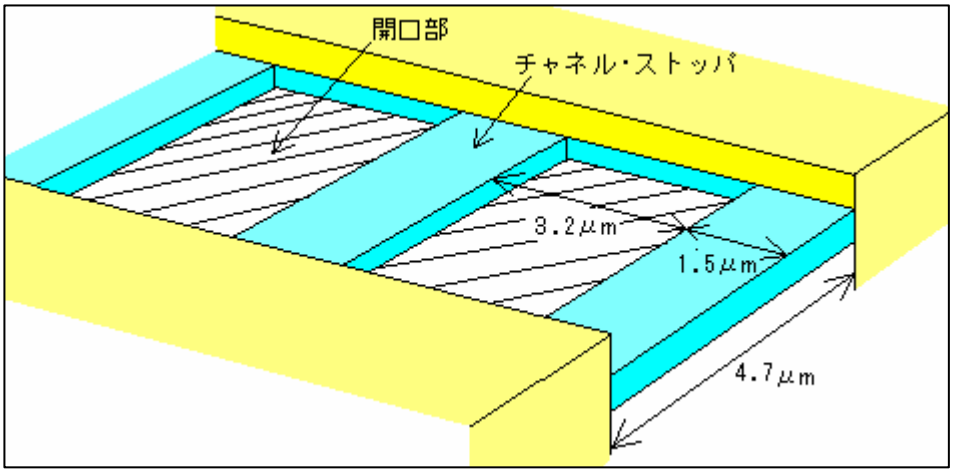
ところで、評価や設計の際には、広いダイナミックレンジを得るために、この飽和出力電圧ぎりぎりに最大の露光量になるように、照度(と蓄積時間)を調整することを推奨します。

μPD3747：フォトセルの立体図

Q1. データ・シートに、次のようなフォトセルの平面図がありますが、立体的にはどのような構造ですか。



A1. 次のとおりです。



μPD3747:リセット・フィードスルー・レベル・クランプ・クロック

Q1. リセット・フィードスルー・レベル・クランプ・クロック(φCP)とは、どのようなものですか。
また、画像クランプ時とライン・クランプ時では、信号出力に違いがありますか。

A1. φCP は、出力を所定のレベルにクランプするためのタイミング・パルスです。画素クランプとライン・クランプの2種類がありますが、機能的には同じです。画素クランプとライン・クランプでは、ランダム・ノイズ特性が異なります。データ・シートの電気的特性にも記載していますが、ライン・クランプ時のランダム・ノイズは、画素クランプ時の約4倍(TYP)となります。ライン・クランプの方が、比較的容易にクランプ・パルスを生成できますが、ノイズ特性を重視して、画素クランプを推奨します。
なお、画素クランプでは、クランプ・パルスのタイミングで、出力にノイズが重畳します(データ・シート p.8, 9 の CCD 出力波形参照)。

μPD3753:2 つの GND

Q1. GND がアナログとデジタルで分かれていますが、応用回路例のようにまとめていいのですか。

A1. はい、まとめてもかまいません。雑音などに配慮した基板設計を想定して、分けています。

μPD3753:分光特性

Q1. 分光特性のグラフの測定条件は、照度一定とエネルギー一定のどちらですか。

A1. 測定条件は、エネルギー一定です。

μPD3753:φ1,φ2 のパルス幅

Q1. データ・レート max.2MHz の記載がありますが、φ1, φ2 のパルス幅についての記載がありません。φ1 と φ2 の 1 サイクルで、V_{OUT} 出力が 1 回なので、周波数は max.2MHz ですか。また、パルス幅はどのくらいですか。

A1. はい、φ1, φ2 の最大周波数は 2MHz(周期 500ns 以上)です。立ち上がり、立ち下がりの規定はありますが、レベル幅は規定していません。ただし、クロックのデューティは、基本的に 50%でご使用ください。
なお、データ・シート p.7 の φTG とのタイミングでは、t5=min.650ns なので、φ1, φ2 の半クロックが 500ns 以上ですが、t5 は φTG のパルス期間に限定されます。φTG パルス前後の期間だけ、φ1 のハイ・レベルと φ2 のロウ・レベルが、それぞれ長くなるように入力する必要があります。

μPD3753:カバー材質

Q1. 上部窓キャップの材質はなんですか。また、アルコールなどの有機溶剤で、汚れを除去してもいいですか。

A1. キャップの材質は、アートンというものです。 拭き取りには、次の溶剤であれば問題ありません。

エチル・アルコール(EtOH)、メチル・アルコール(MeOH)、イソプロピル・アルコール(IPA)、
Nメチル・ピロリドン(NMP)

これら以外の溶剤を使用した場合は、キャップに物理的または光学的な傷害を引き起こす恐れがあります。
また、拭き取りには、傷をつけないよう、ゴミなどの付着していない、やわらかい布をご使用ください。

μPD8827A：電源の意味

Q1. μPD8827A には、デジタル電圧、ヒートシンク電圧、出力部ドレーン電圧の 3 つの電源がありますが、デジタル電圧とヒートシンク電圧は、どの部分に供給されるのですか。

A1. デジタル電圧は、クロック・ドライバなど内部デジタル回路用の電源端子です。ヒートシンク電圧は、金属製ヒートシンクに熱を逃がすために接続された、CCD チップの裏面の電位を保つのに必要な電源です。

μPD8827A：各クロックをドライブするバッファの数

Q1. データ・シートの応用回路例で、φ1A、φ2A などをドライブするバッファを 3 つ並列に使用するように記述されていますが、φTG1 は入力端子容量が typ.400pF と大きいのに、なぜバッファは 1 つでいいのですか。

A1. φ1A、φ2A などの信号にはクロス・ポイントの規格があり、他信号との関係が比較的厳しいですが、φTG1 は他信号との関係が緩やかなため、あまり立ち上がり／立が下りが厳しくありません。また、φTG1 のパルス幅 t3 は、min.1μs～max.5μs と緩やかな規格であることもその理由です。あくまでも応用回路例なので、規格を満足するように、お客様の配線容量なども考慮し、設計と評価をお願いします。

μPD8670A：マイクロ・レンズは付いているか

Q1. マイクロ・レンズは付いていますか。

A1. いいえ、マイクロ・レンズは搭載していません。

μPD8670A：光入射角

Q1. 約 60 度の角度で光が入射した場合、検知可能ですか。

A1. 斜めの光入射は、垂直光入射に対して CCD 受光面の光量が厳しくなるため、出力信号減少となります。
この減少に対して、60 度の角度で検知可能かどうかは、システム設計に依存するため、一概に言えません。

μPD8670A：レジスタ・ダーク DC レベル

Q1. データ・シートで、「3. 出力電圧の不均一性(PRNU)」の図の「レジスタ・ダーク DC レベル」とはなんですか。

A1. タイミング・チャートに記載されている「レジスタ画素（32 画素）」期間の V_{OUT} の DC レベルのことです。

μPD8861：電源投入順序

Q1. CCD の電源投入順序に、制限はありますか。

A1. CCD 電源 12V を先に投入し、その後 5V 系の電源を立ち上げることを推奨します。

μPD8861：出力電圧の変動量

Q1. 光照射時、有効フォトセルの同一画素を複数回読み出すと、出力電圧の変動はどのくらいですか。

A1. 参考値として、概略は次のとおりです。

出力電圧	変動（ショット・ノイズ）
0.0V(暗時)	1.5mV
0.5V	9.0mV
1.0V	12.5mV
1.5V	15.5mV
2.0V	18.0mV

μPD8861：ノイズ除去方法

Q1. CCD 信号のノイズ除去方法について、次のどちらの方がいいですか。

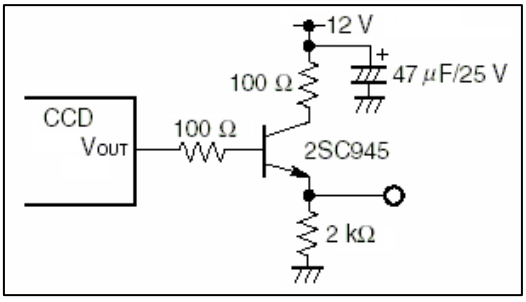
(1) 隣接する 4 画素を平均する方法

(2) 同一画素を 4 回読み出して、平均する方法

A1. お客さまの信号処理にも依存するので一概には言えませんが、CCD としては画素ごとの出力差(デバイスは PRNU, DSNU で規定)があるため、(2)の同一画素の平均の方が好ましいです。

μPD8861：出力を受けるトランジスタ回路

Q1. データ・シートの応用回路例で、次図の B1-B3 等価回路(CCD 出力部)がありますが、他社ではトランジスタ(2SC945)のベース、コレクタに抵抗(100Ω)のないものがあります。これらは省略してもいいですか。



A1. 標準的なエミッタ・フォロワのバッファ回路で、100Ω の抵抗はなくても動作します。この抵抗は、電源電圧が 12V と高いため、μPD8861 か 2SC945 の一方しか電源が供給されない場合や、回路のショートで流れる電流を制限する保護のためです。データ・シートの回路は参考例なので、ご評価のうえ決定してください。

μPD8884A：出力を受ける A/D コンバータ

Q1. μPD8884Aの出力を受けるA/Dコンバータは、何ビットのものが必要ですか。

A1. 当社CCDでは、一般的に8-10ビットのA/Dコンバータで受けます。出力に対する揺らぎやノイズのレベルで、システム的に決定できます。たとえば、データ・シート記載のランダム・ノイズは平均1.2mVで、飽和出力電圧が2.7Vなので、白が飽和レベルに近い露光量の設定をしても約2000倍ですから、11ビットの分解能です。ただし、1.2mVは平均値（偏差値）で、また出力電圧の不均一性なども考慮すると、分解能は低下します。

μPD8884A: データ転送時間

Q1. μPD8884A のデータ転送時間について、データ・レート 5MHz ということは、42720 画素のデータを 5MHzで読み出すということですか。

A1. はい、そうです。

データ・シートp.6の推奨動作範囲に、データ・レートのmax.5MHzが$f_{\phi RB}$と記述されていて、φRBの周波数を表しています。そして、p.2のブロック図で、φRBが供給される部分は、各色に1箇所です。また、各色には2列のCCDアナログ・シフタがあり、4列のフォトセルがあります。その構造は、1列のCCDアナログ・シフタを2列のフォトセルがはさんでいます。つまり、2列のフォトセルからの電荷が、φTGのタイミングで転送ゲートを通り、1列のCCDアナログ・シフタに転送されます。そして、φ1-2のクロックでCCDアナログ・シフタをシフトしていき、φRBが供給される部分を通り、出力端子V_{OUT}1-3から出力されます。ここで、各色のCCDアナログ・シフタは2列ですが、φRBが供給されている部分は1箇所なので、2回に分けて出力されます。φRBは5MHzで動作しますが、2回に分けられるので、φTGが2周期分必要になります。なお、21360画素というのは有効フォトセルの数で、p.9では省略されており、時間計算には無効フォトセルやオプティカル・ブラックなどを出力するクロック時間、φTGのハイの時間なども必要です(p.10のタイミング・チャート参照)。

CCD 全製品:クランプ電圧(オフセット・レベル)と平均暗時出力(ADS)電圧との違い

Q1. クランプ電圧と ADS 電圧は違うのですか。

A1. CCDセンサ受光部は、入射光量に比例した電荷を発生し、それを電圧に変換して電気信号で出力します。しかし、暗時でもPN接合のリーク電流成分によるエレクトロン発生などがあり、これがADS電位になります。このため、クランプ電圧からADS電圧分だけ下がります。

CCD 全製品:センタ・ピクセルの位置

Q1. レンズの中心を CCD のセンタ・ピクセルに合わせたいのですが、センタ・ピクセルはどこになりますか。

A1. まず、第 1 有効ピクセルからセンタ・ピクセルまでの距離は、次のように求めます。

①CCD の有効画素数を A(偶数の場合)とすると、A/2 画素目と A/2+1 画素目の中心がセンタです。

②データ・シートにフォトセル・ピッチが記述されていますが、たとえば B[μm]とします。

③第 1 有効ピクセルからセンタ・ピクセルまでの距離は**(A/2−1+0.5)×B[μm]**です。

ここで−1 は、第 0 有効ピクセル(実際はない)からでなく、第 1 有効ピクセルからの距離を計算するための補正値です。また+0.5 は、A/2 画素目と A/2+1 画素目の中心をセンタにするための補正値です。

次に、1番ピンから第一有効ピクセルまでの距離がデータ・シートに記述されています。たとえば、C[mm]とすると、1番ピンからセンタ・ピクセルまでの距離は、**((A/2-1+0.5)×B)×0.001+C[mm]**になり、実際に1番ピンからの距離として位置決めできます。

ただし、データ・シートの外形図で、第1有効ピクセルからパッケージのセンタまでの距離に誤差が記述してあるように、このセンタ・ピクセルまでにも同程度の誤差があります。また、1番ピンから第1有効ピクセルまでの距離にも誤差が記述されていますので、総合的に両方の誤差を加算した誤差があります。

CCD 全製品:感度の比較

Q1. ルネサス エレクトロニクスの CCD センサの感度の単位は V/lx*s を使用していますが、他社は V*cm^2/μJ を使用しています。どのように比較するのですか。

A1. 単位の関係を考えましょう。まず、人が最も明るく感じる555nm波長で、光出力1W(ワット)は683lm(ルーメン)です。そして、1W・s(ワット・秒)は1J(ジュール)で、1lmは1lx・m^2(ルクス・平方メートル)です。これらの関係から、お問合せの感度の関係は次のようになりますので、これで比較してください。

1(V/lx・s)=683/100(V・cm^2/μJ)=6.83(V・cm^2/μJ)

ただし当社では、光の波長が約400～約800nm(可視光)の最大値などで感度を表示しているため、各製品のデータ・シートで総合分光感度特性を参照し、555nmでの比感度を用いて、補正する必要があります。

たとえばμPD8827Aでは、この感度の最大値が赤の55.44V/lx・sで、総合分光感度特性から赤の最大点(約620nm)に対する555nmでの比感度は約80%ですので、約303(55.44×0.8×6.83)V*cm^2/uJに相当します。

CCD 全製品:S/N 比

Q1. μPD8894 のランダム・ノイズ(CDS)は 1.2mV ですが、Signal の電圧は飽和出力電圧から 2.8Vとして、S/N=2.8V/1.2mV と考えていいですか。S/N 比(=有効信号出力電圧値/暗時出力電圧値)を知りたいです。

A1. S/N比(=有効信号出力電圧値/暗時出力電圧値)について、一般にはS/N比を最大にするために、有効信号出力電圧値に飽和出力電圧(2.8V)を使用しますが、実使用のS/N比の計算では、お客様の光学設計も含めた有効信号出力電圧値を使用します。また、暗時出力電圧は、ADSかランダム・ノイズのどちらかを指します。通常、ランダム・ノイズの方が大きいのでこちらを使います。

CCD 全製品: 応用回路例での out 端子後段のトランジスタ

Q1. μPD3747 のデータ・シートの応用回路例で、Vout 端子の後段にトランジスタの 2SA1005 と 2SC945 が記述されていますが、生産集約品のようです。代替品はありますか。

A1. 2SA1005の代替は2SA1461、2SC945の代替は2SC1623Aです。ただし、パッケージと定格/特性が多少異なりますので、各データ・シートをご確認ください。

Q2. μPD8670A のデータ・シートの応用回路例で、Vout 端子の後段にトランジスタの 2SA1206 と 2SC1842 が記述されていますが、生産集約品のようです。代替品はありますか。

A2. 2SA1206の代替は2SA1462、2SC1842の代替は2SC1623Aです。ただし、パッケージと定格/特性が多少異なりますので、各データ・シートをご確認ください。

4. 通　信

4.1 USB …736
4.1.1 USB 全般 …736
4.1.2 ホスト・ペリフェラル・コントローラ …738
4.1.3 ホスト・コントローラ …741
4.1.4 USB2.0 HUB コントローラ …744
4.1.5 ファンクション・コントローラ …748
4.1.6 IDE ブリッジ …749
4.2 IEBus …750

4.1 USB

4.1.1 USB 全般

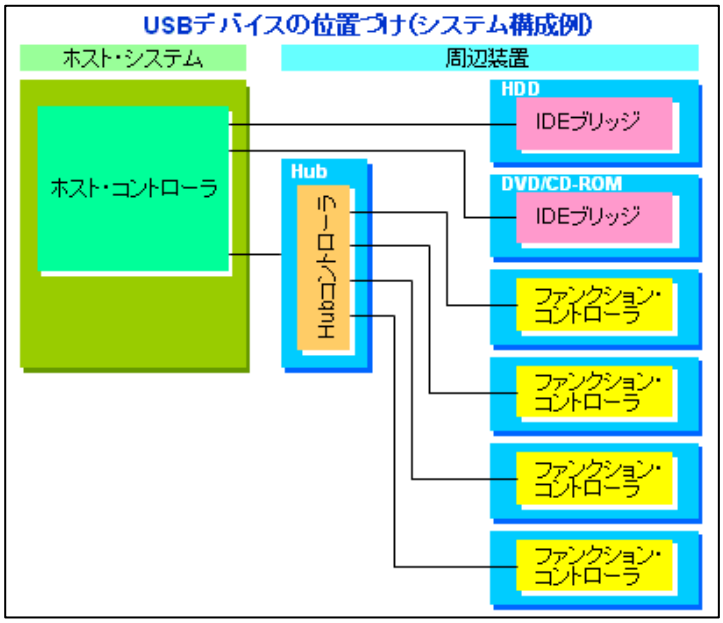
USB の概要

Q1. USB とはなんですか。

A1. 業界団体によるインタフェース規格のひとつで、「Universal Serial Bus」の略です。規格化された共通コネクタで、各種周辺機器を接続できます。また、Hub(ハブ)と呼ばれる分岐点を追加すれば、最大 127 台の機器を接続でき、Plug&Play で機器を認識できるホット・プラグインに対応しているなどの、使いやすさが特徴です。現在、パソコンではほとんど USB インタフェースが搭載されていますが、プリンタやスキャナなどの周辺機器にも対応するため、より高速な USB 2.0 が規格化されました。USB 2.0 でインタフェースするには、ホストと周辺機器の両方が USB 2.0 規格に対応していることが必要です。

Q2. USB 関連製品は、それぞれどのような位置づけですか。

A2. ホスト・システムにホスト・コントローラを内蔵し、周辺機器にファンクション・コントローラを内蔵して、相互にインタフェースします。ホスト 1 台に複数の周辺機器を接続する場合、USB ポートの増設が必要です。このとき、USB ハブで増設可能です。ハブに内蔵されるハブ・コントローラは、ホストと周辺機器の USB 接続を中継します。なお、DVD/CD-ROM ドライブなど IDE インタフェースの周辺装置は、IDE ブリッジで接続します。



Q3. USB のホスト・コントローラやハブ・コントローラのドライバは、どのように入手すればいいですか。

A3. Windows や Linux 環境では、基本的に OS でサポートされています。
μITRON や VxWorks などの組み込み用リアルタイム OS のドライバは、ソフトウエア・ベンダが提供しています。詳細は、USB サイトから、各製品のページ下部をご参照ください。

USB の仕様

Q1. USB の仕様を教えてください。

A1. USB そのものの仕様については、USB の仕様書をご参照ください。
http://www.usb.org/developers/docs/usb_20.zip
USB の概要については、下記リンクで説明しています。
http://www2.renesas.com/usb/ja/about_usb/index.html

Q2. USB1.1 と USB2.0 の違いはなんですか。

A2. 転送レートが、それぞれ次のように規定されています。
USB 1.1 : Low-Speed (1.5 Mbps:LS)、Full-Speed (12 Mbps:FS)
USB 2.0 : Low-Speed (1.5 Mbps:LS)、Full-Speed (12 Mbps:FS)、Hi-Speed (480 Mbps :HS)
一般に、USB 1.1 は LS/FS を指し、USB 2.0 は HS を指すかのように表現されますが、これは正しくありません。現在、開発されている USB デバイスは、すべて USB 2.0 に準拠する必要があります。転送レートを示すには、LS/FS/HS のいずれかを使います。

Q3. USB の IC デバイスを購入すれば、すぐ使えますか。

A3. 当社の USB LSI は、すべて USB-IF による認証試験をパスしていますが、お客様が LSI 製品を使用して機器を開発・販売される際は、機器ごとに個別に USB-IF による認証試験をパスさせ、ライセンスを受ける必要があります。詳細は、USB 仕様の策定管理団体である USB Implementers Forum(USB-IF)でご確認ください。

Q4. USB の周辺機器が動作しないのですが、IC メーカでサポートしてもらえますか。

A4. 機器の動作は、USB の LSI だけでなく、LSI が搭載されている基板や部品、その他の回路、さらにはソフトウエアの設計や特性に大きく依存します。これらは IC メーカでは分かりませんので、その周辺機器のメーカでないとサポートできません。

データ転送速度

Q1. USB2.0 で、どのくらいの転送レートになりますか。

A1. データ自体は、480Mbps(60MByte/s)で転送します。他の通信規格と同様、USB では送信するパケットの形式が決まっており、アドレスや CRC などの情報を付加して転送します。またパケット間に Idle 期間があったりして、全期間で 60MByte/s の純粋なデータ転送となるのではありません。転送レートはシステムにも依存し、PCI 側の制限や USB デバイスの制限、PC の性能やドライバによっても、大きく異なります。PC 環境で USB2.0 ストレージ・デバイス(HDD)を使った当社の評価では、約 20～25MByte/s の転送レートです。

★ 4.1.2 ホスト・ペリフェラル・コントローラ

基本仕様

Q1. USB ホスト・ペリフェラル・コントローラとは、どのようなものですか。

A1. USB のホスト機能とペリフェラル機能を持ったコントローラです。USB ポートは 2 つありますが、一方はホスト専用で、他方はホストかペリフェラルのいずれかに使用できます。機能の切り替えは、レジスタで行います。

Q2. USB On-The-Go 規格に対応していますか。

A2. いいえ、対応していません。

Q3. ホスト機能(ポート No.1)とペリフェラル機能(ポート No.2)を同時に使用する場合、1つのパイプ・レジスタを両機能で共有することはできますか。

A3. いいえ、できません。いずれか一方で使用することになります。

Q4. レジスタへのバイト・アクセスはできますか。

A4. CPU によるバイト・アクセスには対応していません。
ただし、DMA によるデータ転送では、バイト・アクセスができます。バイト・アクセスをするには、レジスタ設定が必要です。

Q5. どのようなデバイス・クラスに対応できますか。

A5. ソフトウエアを開発すれば、Mass storage、HID、Audio など、さまざまなデバイス・クラスに対応できます。

Q6. INT 信号や DREQ、DACK 信号のアクティブ極性を変えることはできますか。

A6. はい、レジスタ設定でアクティブ・レベルを変えられます。ディフォールトはアクティブ・ロウですが、アクティブ・ハイにすることもできます。

システム・クロック

Q1. 水晶振動子の代わりに、クロック発振器を使用できますか。

A1. PLL やクロックの停止処理(PWDOWN と GSUSPEND ビットを 1 に設定)をしなければ、外部クロック発振器を使用することができます。

ホスト機能： μPD720150

Q1. EHCI や OHCI に準拠していますか。

A1. いいえ、準拠していません。当社オリジナルの仕様です。

Q2. High-speed 機能を無効にして、強制的に Full-speed で転送することはできますか。

A2. はい、レジスタ設定で High-speed の機能を無効にでき、Full-speed での転送ができます。

Q3. USB Control 転送の各ステージ(Setup、Data、Status)の管理は自動的に行われますか。

A3. いいえ、uPD720150 では管理していないので、ソフトウエアで管理していただくことになります。制御方法は、ユーザーズ・マニュアルをご参照ください。

Q4. DMA 自動継続機能とは、どういう機能ですか。

A4. TBT（Total Byte to Transfer）分のデータ転送が完了するまで、uPD720150 が DMA 転送の要求を継続する機能です。この機能により、システムの負荷を低減できます。詳細は、ユーザーズ・マニュアルをご参照ください。

Q5. SCRATCHPAD レジスタは、ホスト機能でも使用できますか。

A5. ポートNo.2をペリフェラルに設定すれば使用できます。つまり、2ポートともホストに設定する場合は使用できません。

ペリフェラル機能： μPD720150

Q1. USB ペリフェラル機能は、Low-speed 転送をサポートしていますか。

A1. いいえ、High-speed と Full-speed だけをサポートしています。

Q2. High-speed 機能を無効にして、強制的に Full-speed で転送することはできますか。

A2. はい、レジスタ設定で High-speed の機能を無効にでき、Full-speed での転送ができます。

Q3. DMA 自動継続機能は、ペリフェラルでもサポートされていますか。

A3. いいえ、サポートしていません。

ボード設計（ホスト、ペリフェラル共通）：μPD720150

Q1. 未使用端子は、どのように処理すればいいですか。

A1. データ・シートの 2.2 節をご参照ください。

Q2. RREF 端子に接続される 1.6KΩ 抵抗は、なぜ±1%の精度が必要なのですか。

A2. RREF 端子は、AV_{SS}(R) 端子とともに、内部アナログ回路の基準電圧の生成に使用されます。その精度を上げるため、接続される抵抗には、±1%の精度が要求されます。また、ボード設計でも、これらの端子がノイズの影響を受けないよう、レイアウトに注意する必要があります（USB2.0 ボード作成ガイド参照）。

Q3. USB 信号線に、終端抵抗や 15kΩ のプルダウン抵抗（ホスト時）、1.5kΩ のプルアップ抵抗（ペリフェラル時）は必要ですか。

A3. いいえ、不要です。これらの抵抗は、すべてμPD720150 が内蔵しています。

Q4. VDD15OUT 端子付近のセラミック・コンデンサ（4.7μF）は必要ですか。

A4. はい、内蔵レギュレータの出力特性上、このコンデンサが必須です。必ず、VDD15OUT 端子の近傍に配置してください。

Q5. デジタル電源（V_{DD}、V_{DD15}）とアナログ電源（AV_{DD33}、AV_{DD15}）は、分離する必要がありますか。

A5. AV_{DD33}、AV_{DD15} は内部アナログ回路用の電源です。したがって、アナログ電源のノイズを極力低減するため、デジタル電源と分ける必要があります。

Q6. ボード・レイアウト上で注意すべき事項はありますか。

A6. USB2.0 ボード作成ガイドをご参照ください。アナログ回路周辺や USB の差動信号の配線には、特にノイズ対策などの注意が必要です。

Q7. 参考アプリケーション回路図はありますか。

A7. はい、あります。当社の特約店が配付していますので、製品購入先を通じてご相談ください。

ボード設計（ホスト用）：μPD720150

Q1. Downstream ポートに、過電流保護機能の搭載は必要ですか。

A1. はい、USB2.0 規格で規定されているので必要です。詳細は、USB2.0 規格の 7.2.1.2.1 節をご参照ください。

Q2. 過電流検出のためのパワー・スイッチは必要ですか。

A2. システムに依存します。
USB2.0 規格では、過電流(Over-current)を検出して、システムに報告するように規定していますが、その手段は問いません。パワー・スイッチを使わずに、ポリヒューズなどで構成することもあります。なお、パワー・スイッチは、メーカによって極性が異なりますので、ご注意ください。

Q3. VBUS 入力に設置するコンデンサは、どのくらいの容量にすればいいですか。

A3. 120μF 以上が必要です。詳しくは USB2.0 規格の 7.2.4.1 節をご参照ください。

ボード設計（ペリフェラル用）：μPD720150

Q1. USB コネクタの VBUS 端子を uPD720150 の OCI20/VBUS 端子に接続するとき、なぜ抵抗分割で 3.0V にして入力する必要があるのですか。

A1. OCI20/VBUS 端子には 5V 耐圧バッファを内蔵していますが、uPD720150 の電源が OFF のときに、PC などの USB ホストから VBUS に 5V をそのまま入力されると、uPD720150 がダメージを受けてしまうためです。必ず分圧して入力してください。

Q2. VBUS 入力に設置するコンデンサは、どのくらいの容量にすればいいですか。

A2. 合計で 1～10μF のコンデンサを実装する必要があります。USB2.0 規格の 7.2.4.1、7.2.4.2 をご参照ください。

開発環境

Q1. サンプル・ファームウエアはありますか。

A1. ホスト・コントローラとペリフェラル・コントローラそれぞれについて、サンプル・ファームウエアを用意しています。当社の特約店が配付していますので、製品購入先を通じてご相談ください。

4.1.3 ホスト・コントローラ

概　　要

Q1. USB2.0 ホスト・コントローラとは、どのようなものですか。

A1. ホスト・コントローラは、接続されるUSB デバイスすべてを制御します。その制御方法はUSB の規格で定義されており、USB2.0 ホスト・コントローラは、USB2.0 規格すべてを満足した形で、USB デバイスを制御します。

Q2. ホスト・コントローラとドライバの関係はどのようになりますか。

A2. ホスト・コントローラは USB デバイスを制御しますが、その制御方法はソフトウエアが決めた一定のルールに従っています。つまり、ソフトウエアで用意された手順書を読みながら、そのとおりに行動して、その結果を手順書にフィードバックするような働きをします。この手順書を管理するソフトウエアがデバイス・ドライバです。ドライバがなければ、ホスト・コントローラは動作しません。

Q3. OHCI や EHCI とは、どういうものですか。

A3. ホスト・コントローラとドライバの間で決められた手順(ルール)が、Host Controller Interface(HCI)です。OHCI (Open Host Controller Interface) は USB1.1 のホストの規格で定義されており、ほかに UHCI があります。EHCI (Enhanced Host Controller Interface) は、USB2.0 の規格化とともに Intel 社が作成した規格です。当社のホスト・コントローラは、OHCI と EHCI に準拠しています。

基本仕様

Q1. ルネサス エレクトロニクス社の USB2.0 ホスト・コントローラの主な仕様を教えてください。

A1. USB のポートを 5 ポートまで用意しています。各ポートとも、HS(480Mbps), FS(12Mbps), LS(1.5Mbps)の異なるスピードの USB デバイスを接続できます。また、EHCI を 1 つと OHCI を 2 つ搭載しています。
システム側は PCI バス(PCI2.2 規格に準拠)になっており、PCI バスに直結できます。

Q2. ルネサス エレクトロニクス社の USB2.0 ホスト・コントローラの特徴はなんですか。

A2. μPD720100A は、世界初の USB2.0 対応ホスト・コントローラです。当初は USB2.0 対応のホスト・コントローラがこれだけだったため、Microsoft 社の XP 用ドライバ開発に、当社のホスト・コントローラが使用されました。また、その他組み込み系 OS ベンダでも、ドライバ開発環境のプラットフォームとなっています。さらに、市場に広く普及しているので、周辺機器メーカが、当社製のホスト・コントローラを評価対象とすることが多く、もっとも問題を起こしにくい標準的なホスト・コントローラと言えます。
また、他社製品と比較して、低消費電力と高い転送能力、小型パッケージも特徴です。

Q3. なぜ OHCI が 2 つあるのですか。

A3. ホスト・コントローラは、その下につながる USB デバイスの管理をします。多くの USB デバイスが接続されると、各デバイスが使用できるバンド幅が制限されます。たとえば、カメラとスピーカを接続した場合、カメラの画像がコマ落ちしたり、スピーカの音声が途切れたりします。この場合、複数のホスト・コントローラに個別に接続すると、ホスト・コントローラとデバイスが 1 対 1 になるため、バンド幅を考慮する必要がなくなります。
従来の 12Mbps という低速デバイスを管理する OHCI では、このバンド幅の調整が必要なので、当社のホスト・コントローラでは、2 つの OHCI を搭載しています(ポート 1, 3, 5 は OHCI1 で、ポート 2, 4 は OHCI2 で制御)。

Q4. EHCI と OHCI は、どのように切り替わるのですか。

A4. 当社のホスト・コントローラは、接続されたデバイスの転送速度を自動認識します。接続前は、EHCI が各ポートの所有者(Owner)です。USB デバイスが接続されると、USB2.0 の規格で定義されたシーケンスで、接続されたデバイスの転送速度を自動認識します。
接続されたデバイスが HS なのか、FS/LS なのかをシステムに通知して、FS/LS の場合には OHCI に切り替えます(実際には、ドライバと共同でこの制御をします)。つまり、接続デバイスを認識して、HS の場合は EHCI が、FS/LS の場合は OHCI が、それぞれポートの所有者になります。

クロック

Q1. 30MHz 発振動作と 48MHz クロック入力動作は、どのように切り替えるのですか。

A1. ホスト・コントローラによって、次のように仕様が異なります。

	30MHz 発振	48MHz クロック入力
μPD720100A	CLKSEL 端子＝L	CLKSEL 端子＝H
μPD720101	Clock_sel ビット＝0	Clock_sel ビット＝1

Clock_sel ビット：EXT2 レジスタのビット 5

Q2. システム・クロック用に、48MHz の発振子は使用できますか。

A2. いいえ、使用できません。48MHz 動作の場合は、外部発振器等からのクロック入力が必要です。

Q3. システム・クロック用に、30MHz のセラミック発振子は使用できますか。

A3. いいえ、使用できません。入力クロックには、±500ppm の精度が必要です。当社では、±100ppm の精度の発振子を推奨しており、基本的にセラミック発振子は推奨していません。

端子処理

Q1. 未使用ポートは、どのように処理すべきですか。

A1. それぞれ、次のように処理してください。

DPx/DMx　：ロウ・レベル固定
RSDPx/RSDMx：オープン
PPONx　：オープン(パワー・スイッチ等に接続する Vbus 制御用端子)
OCIx　：ハイ・レベル固定

Q2. INTA、INTB、INTC 端子を共通結線していいですか。

A2. 結線に問題はありませんが、OHCI1、OHCI2、EHCI の各割り込み要求の出力先を示す PCI Configuration 空間の Interrupt Pin の値は、固定値なので変更できません。システム側で、それを考慮する必要があります。

Q3. CRUN0 端子は、どのように使用するのですか。

A3. この端子は、PCI Mobile Design Guide Version 1.1 で定義されています。おもに、モバイル PC などで、PCI クロックの ON/OFF 制御に使用します。この機能を使用しない場合は、ロウ・レベルに固定してください。

Q4. LEGC 端子は、どのように使用するのですか。

A4. デスクトップ PC などで、USB キーボード・マウスを PS2 のキーボード・マウスにエミュレーションするために使います。この機能を使用しない場合、LEGC 端子は使用しませんので、ロウ・レベルに固定してください。

Q5. RREF 端子に接続される 9.1kΩ の抵抗は、なぜ±1%の精度が必要なのですか。

A5. RREF 端子は、AV_{SS}(R) 端子とともに、内部アナログ回路の基準電圧の生成に使用されます。その精度を上げるため、接続抵抗には、±1%の精度が要求されます。またボード設計でも、これらの端子がノイズの影響を受けないよう、レイアウトに注意する必要があります。詳細は、USB2.0 ボード作成ガイドをご参照ください。

Q6. RSDP/RSDM 端子に接続される 36Ω の抵抗は、なぜ±1%の精度が必要なのですか。

A6. USB2.0 の規格では、DP/DM 端子を 45Ω±10%で終端することになっています。当社のホスト・コントローラは、RSDP/RSDM 端子の内部抵抗と外付けの 36Ω の抵抗の合計でこの規格を満足するため、±1%の精度が要求されます。この抵抗値の精度が低かったり、値が異なると、USB の波形に影響することがあります。

外部デバイス

Q1. シリアル ROM は必須ですか。

A1. ご使用のシステム・ソフトウエアが、PCI Configuration 空間の EXT レジスタにライトできれば、シリアル ROM は不要です。その場合、SRMOD/SRCLK/SRDTA 端子はオープンで結構です。ただし、システム・ソフトウエア(BIOS)から設定できない PCI カードや PC カードのようなシステムでは、シリアル ROM が必須です。

Q2. パワー・スイッチは必須ですか。

A2. システムに依存します。USB2.0 のスペックでは、過電流(Over-current)を検出して、システムに報告することが必要ですが、その手段は問いません。パワー・スイッチを使わずに、ポリヒューズ等で構成する場合もあります。なお、パワー・スイッチは、メーカによって極性が異なりますので、ご注意ください。

基板設計

Q1. AV_{DD} は V_{DD} と分離する必要がありますか。

A1. AV_{DD} は内部アナログ回路用の電源です。ノイズを極力低減するため、V_{DD} と分けてください。

Q2. ボード・レイアウト上で、注意事項はありますか。

A2. USB2.0 ボード作成ガイドをご参照ください。アナログ回路周辺や USB の差動信号の配線には、特にノイズ対策などの注意が必要です。

μPD720101：製品の違い

Q1. μPD720100A との違いはなんですか。

A1. μPD720101 の基本的な回路は μPD720100A と等価で、安定性を継承しつつ、Note PC などで必要とされる低消費電力化、パッケージ小型化をしています(消費電力を約 35%削減、12×12mm FPBGA パッケージ)。また、EHCI 規格の Rev1.0 に対応して、転送速度も約 20%向上しています。

μPD720101：ポート

Q1. ポート数を変更できますか。

A1. はい、EXT1 レジスタのビット(2:0)で、2～5 ポートを指定できます。

Q2. ポートの電源制御を行わない設定は、どのようにするのですか。

A2. EXT1 レジスタのビット 21 を"0"にしてください。

μPD720101：HCI

Q1. EHCI を使用しない設定は、どのようにするのですか。

A1. EXT2 レジスタのビット 0 を"1"に設定すれば、EHCI をマスクできます。この場合、全ポートが OHCI で制御され、HS デバイスも FS デバイスとして認識されます。

4.1.4 USB2.0 HUB コントローラ

概　要

Q1. USB2.0 HUB コントローラとは、どのようなものですか。

A1. USB HUB（ハブ）コントローラは、USB ホストと USB デバイスの間に置かれ、USB 信号の分岐を実現するためのデバイスです。USB ホストに多くの USB デバイスを接続したくても、ホストのポート数が足りない場合や、1つのセットで複数の機能を持ちながら、USB のポート数は 1 つにしたい場合などに使用されます。
USB2.0 HUB コントローラは、ダウンストリームに Full-Speed（12Mbps:FS）や Low-Speed（1.5Mbps:LS）のデータも効率よく転送できるように、Transaction Translator（TT）と呼ばれる回路を内蔵しており、ダウンストリーム・ポートに接続される USB デバイスのスピードと無関係に、アップストリーム・ポートが USB2.0 ホスト（EHCI）と接続されているときは、つねに High-Speed で通信を行っています。

Q2. Transaction Translator(TT)とはなんですか。また、"Single"と"Multi"とはなんですか。

A2. TT は、HUB のダウンストリーム・ポートに接続される FS や LS のデバイスのデータ転送をアップストリーム・ポートに対して HS 転送に変換するため、一時的にデータを格納するバッファです。HUB コントローラへの搭載バッファの数によって、"Single"と"Multi"とがあり、"Single"は、HUB コントローラにバッファを 1 つ内蔵しています。"Multi"には、ダウンストリーム・ポートごとにバッファがあります。このため、"Single"を"per Hub"、"Multi"を"per Port"とも呼びます。またこれにより、"Multi"では論理上、ダウストリーム・ポートごとに、12Mbps のバンド幅を実現できます。
なお、"Single" Transaction Translator HUB コントローラのダウンストリーム・ポートに、FS Camera、USB Speaker や FS/HS Storage Device などを組み合わせて接続して、デバイスを同時に動作させても、画像や音の途切れは発生せず、使用上の問題は起こりません。

Q3. USB2.0 HUB コントローラを USB1.1 に準拠したホスト・コントローラに接続できますか。

A3. はい、できます。USB2.0 HUB コントローラは、USB 1.1 のホスト・コントローラ(UHCI か OHCI)に接続すると、Full-Speed Mode で動作します。
また当然ながら、USB2.0 の Host Controller(EHCI)に接続すると、Hi-Speed Mode で動作します。

Q4. ルネサス エレクトロニクス社の USB2.0 HUB コントローラの主な仕様を教えてください。

A4. μPD720110A, 720112, 720114 は 4 ダウンストリーム・ポート、μPD720113 は 7 ダウンストリーム・ポートを持っています。各ダウンストリーム・ポートに、HS(480Mbps)、FS(12Mbps)、LS(1.5Mbps)の異なる転送速度の USB デバイスを接続できます。μPD720110A は、パッケージが 100 ピン 14×14mm の LQFP で、Self-Powered HUB の動作をサポートしています。

μPD720112, 720114 は、低消費電力の実現により、Self-Powered に加えて、Bus Powered もサポートしています。μPD720112 のパッケージは 80 ピン 12×12mm、μPD720114 のパッケージは 48 ピン 9×9mm の TQFP です。おもに、Box HUB や PC モニタ HUB などに利用されます。

μPD720113 のパッケージは、80 ピン 12×12mm の TQFP です。おもに、4 ポート以上に拡張が必要な Port Replicator や Docking Station などに利用されます。

Q5. ルネサス エレクトロニクス社の USB2.0 HUB コントローラの特徴はなんですか。

A5. μPD720110A は、世界初の USB2.0 対応 HUB コントローラです。Microsoft 社の Windows USB2.0 ドライバは、本製品をリファレンスとして開発されています。市場に広く普及している HUB コントローラですので、周辺機器メーカの接続評価でも利用され、標準的な HUB コントローラといえます。

μPD720112, 720113 は、μPD720110A の基本的な回路を踏襲しつつ、消費電力の削減やパッケージの小型化をした第 2 世代製品です。外部 Serial ROM インタフェースを備え、ユーザ独自の Vendor ID、Product ID、String Descriptor やシステム構成をカスタマイズすることができます。

μPD70114 は地球環境に配慮し、さらに小型化した低消費電力、低 BOM コストの第 3 世代製品です。コア・ロジック用 2.5V 電源レギュレータや USB 終端抵抗などを内蔵することで、外付け部品の点数を大幅に削減しています。また、端子機能でシステム構成をカスタマイズでき、外付け Serial ROM が不要になっています。

Q6. HUB の電源供給や管理について教えてください。

A6. HUB の電源形態には、自分で電源を持っている Self-Powered と、アップストリームから電源の供給を受ける Bus-Powered があります。Self-Powered HUB では、ローカルでの電源供給が必要ですが、各ダウンストリーム・ポートへ最大 500mA の電流供給ができます。Bus-Powered HUB では、ローカル電源は不要ですが、最大 100mA しか供給できず、High Powered USB デバイス(Bus-Powered で消費電力が 100mA 以上必要なデバイス : PC カメラや HS Bus-Powered ドライブなど)を接続できません。

	Self-Powered HUB	Bus-Powered HUB
最大ポート数	制限なし	4
最大供給電流	500mA	100mA
ローカル電源	必要	不要
接続可能デバイス	High Powered USB デバイスを含むすべて	Low powered/Self-Powered USB デバイス
おもな用途	Box HUB、PC モニタ、 ドッキング・ステーション、 ポート・リプリケータなど	Box HUB、キーボード、 ポート・リプリケータなど

また電源管理には、ダウンストリーム・ポートの電源制御を一括で行う Gang 方式と、個別に行う Individual 方式があります。Gang 方式では、部品点数を減らせますが、過電流の発生が他のポートに影響します。Individual 方式は個別制御なので、過電流の発生は他のポートに影響しません。

Q7. Non-Removable デバイスと Removable デバイスは、なにが違いますか。

A7. Non-Removable デバイスは、HUB のダウンストリーム・ポートに接続される USB デバイスのうち、取り外しができないデバイスです。たとえば、HUB 付きキーボードのキーがその代表です。Box HUB のように、ダウストリーム・ポートに接続するデバイスが取り外し可能なものは Removable です。

HUB コントローラの Hub Descriptor の bDeviceRemovable 値で、ダウンストリームに接続されるデバイスの属性を設定できます(詳細は、デバイスのユーザーズ・マニュアルをご参照ください)。

Non-Removable デバイスの場合は、Plug の抜き差しなどのイベントが発生しないため、コネクタはもちろん、過電流検出やパワー・スイッチなどの部品を削減することができます。

基本仕様

Q1. μPD720114 は、クロック入力で動作可能ですか。

A1. いいえ、μPD720114 ではクロック入力を禁止しています。Suspend Mode 時に内部の電源とクロックの供給が停止します。このときにクロック入力をしていると、内部発振回路にダメージを与えます。
なお、Suspend Mode はホストに制御されるため、HUB コントローラ側では対処できません。
4 ポートの HUB コントローラで、クロック入力を検討されるなら、μPD720112 をお勧めします。

Q2. HUB コントローラのダウンストリーム・ポート数を変更することは可能ですか。

A2. はい、可能です。μPD720114 は、PPB4,PPB3 端子によって、2～4 ポートに設定ができます。
μPD720112, 720113 は、外部 Serial ROM に bNbrPorts 値を記述することで、ダウンストリーム・ポート数を設定できます。μPD720110A は、外部 Serial ROM で設定できますし、端子機能（NUMPORT）で 3 ポートに設定することもできます。詳細については、各ユーザーズ・マニュアルをご参照ください。

端子処理

Q1. RREF 端子に接続される抵抗は、なぜ±1%の精度が必要なのですか。

A1. RREF 端子は、AV_{SS}(R)端子とともに、内部アナログ回路の基準電圧の生成に使用されます。その精度を上げるため、接続抵抗（μPD720110A は 9.1kΩ で、μPD720112, 720113, 720114 は 2.43kΩ）には、±1%の精度が要求されます。また、ボード設計でも、これらの端子がノイズの影響を受けないよう、レイアウトにご注意ください。詳細は、USB2.0 ボード作成ガイドをご参照ください。

Q2. AV_{DD} は V_{DD} と分離する必要がありますか。

A2. AV_{DD} は内部アナログ回路用の電源です。ノイズを極力低減するため、V_{DD} と分けてください。

外部デバイス

Q1. 外付け Serial ROM は必須ですか。

A1. μPD720110A, 720112, 720113 は、外付け Serial ROM がなくても動作します。
汎用 Box HUB を規定値の Descriptor で構成する場合には、Serial ROM がなくてもかまいません。それ以外の場合、たとえばポート数の変更や、Vendor ID、Product ID の変更などを行う場合には、外付け Serial ROM が必要になります。μPD720114 には、外付け Serial ROM インタフェースがありません。外部端子処理によりポート数などの変更ができます。詳細については、各ユーザーズ・マニュアルをご参照ください。

Q2. HUB コントローラ用の Serial ROM（バス）を他の IC と共有で使えますか。

A2. いいえ、他の IC と、HUB コントローラ用の Serial ROM（バス）を共有して使うことはできません。

Q3. 外付け Serial ROM のデータ書き込みは、実装後でもできますか。

A3. μPD720110A には、Serial ROM の書き込み機能がないため、実装前にデータを書き込んでおくか、ボードから書き込めるようにパターンを設ける必要があります。uD720112, 720113 は、HUB のアップストリーム・ポート経由で、Serial ROM の書き込み機能を備えていますので、実装してからデータ書き込むことが可能です。ROM への書き込みツールとして、Windows 版と DOS 版を用意しています。

Q4. 部品コスト削減のため、パワー・スイッチの代わりにポリヒューズを使用できますか。

A4. システムに依存します。Bus-Powered HUB では、USB 仕様上、パワー・スイッチが必須です。

Self-Powered HUB では、過電流(Over-current)を検出して、システムに報告する義務がありますが、その方法はパワー・スイッチ、ポリヒューズのいずれも利用できます。

	Self-Powered HUB	Bus-Powered HUB
パワー・スイッチ	オプション	必須
過電流検出	必須	オプション

備考　パワー・スイッチは、過電流検出機能を内蔵しています。

基板設計

Q1. ボード・レイアウト上で、注意事項はありますか。

A1. USB2.0 ボード作成ガイドをご参照ください。

アナログ回路周辺や USB の差動信号の配線には、特にノイズ対策などの注意が必要です。

μPD720112、720113：端　子

Q1. LPWRM 端子は、どのように使用するのですか。

A1. LPWRM は、Local power を監視する端子です。通常は、Box HUB やモニタに内蔵される HUB で、この端子を"High"にクランプしてください。バッテリなどを使われるモバイル PC の Docking Station に使用された場合、この端子を監視すれば、AC アダプタが外されたときに、USB HUB 経由で Host PC に通知できます。

LPWRM 端子が"High"から"Low"に変化すると、全 Port Status は、Power-Off State へ遷移します（詳細は、ユーザーズ・マニュアルをご参照ください）。

Q2. 未使用ポートは、どのように処理すべきですか。

A2. それぞれ、次のように処理してください。

	μPD720112	μPD720113
DPx/DMx	ロウ・レベル固定	
CSBx	ハイ・レベル固定	
PPBx	オープン	
LPWRM	ハイ・レベル固定	
SCL	オープン	
LEDx	オープン	-（端子なし）
AMBER	オープン	
GREEN	オープン	

4.1.5 ファンクション・コントローラ

ファンクション・コントローラ全般

Q1. ファンクション・コントローラとは、どのようなものですか。

A1. ファンクション・コントローラは、USB ケーブルで接続されたホスト・コントローラとデータを送受信するためのものです。このため、ファンクション・コントローラ同士の通信はできません。また、HUB コントローラを介してホスト・コントローラと通信できますが、HUB コントローラにつながった他のファンクション・コントローラとは直接通信できません。
周辺機器にファンクション・コントローラを搭載すれば、パソコンなどホスト・コントローラを搭載した機器と通信できるようになります。

Q2. ファンクション・コントローラとドライバは、どういう関係ですか。

A2. USB Function は、Function の種類によって Device Class で分類され、制御方法などが仕様化されています。この Class 仕様に準拠して機器のファームウエアを作られた場合、独自にドライバを開発しなくても、Windows などの OS にあらかじめインストールされている Class Driver で、その機器が動作する可能性が高くなります。

μPD720122：基本仕様

Q1. どのような特徴がありますか。

A1. μPD720122 は、Hi-/Full-Speeed を自動的に切り換える USB PHY コントローラと、Control Endpoint(×1)、Bulk Endpoint(×2)、INT Endpoint(×1)を搭載しています。このため、ストレージ機器やプリンタ機器を USB Hi-Speed 対応とするために最適なデバイスです。

Q2. Full-Speed デバイスとして使用できますか。

A2. ホスト・コントローラが Full-Speed で動作している場合、Full-Speed として動作します。
ホスト・コントローラが Hi-Speed で動作している場合は Hi-Speed で動作し、Full-Speed に固定できません。

Q3. Low-Speed デバイスとして使用できますか。

A3. いいえ、Low-Speed デバイスとしては使用できません。

μPD720122：割り込み

Q1. 割り込み要求信号は、3 本とも使用する必要がありますか。

A1. いいえ、INT_ALL 信号 1 本でも使用可能です。

μPD720122：バス・パワード

Q1. バス・パワード・デバイスの機器として設計できますか。

A1. Unconfigured 時の電流規格を満たすことが困難なため、推奨できません。

4.1.6 IDE ブリッジ

IDE ブリッジ全般

Q1. IDE ブリッジ とは、どのようなものですか。

A1. IDE ブリッジは、IDE インタフェースを持つ HDD や DVD ドライブと、USB ホスト・コントローラを搭載したパソコンとの通信をするために、USB ケーブルでの接続を可能とするデバイスです。

Q2. USB 2.0 IDE ブリッジをサポートするドライバはありますか。

A2. 当社の USB 2.0 IDE ブリッジは、USB Mass Storage Class Bulk Only Transport 仕様に準拠しています。この仕様のドライバは、Windows2000 以降と MAC OS 9 以降の OS で標準サポートされています。なお、Hi-Speed 仕様で動作させるには、EHCI ドライバが別途必要です。

μPD720130、720133：基本仕様

Q1. どのような特徴がありますか。

A1. μPD720130, 720133 は、USB 2.0 と ATA/ATAPI デバイスを接続するブリッジ機能を実現するために、USB 2.0 規格に準拠した USB PHY、USB SIE ブロック、USB Bulk-Only Transport 仕様に準拠するための Endpoint Controller、16 ビット CISC プロセッサ、ATA/ATAPI-6 準拠の IDE コントローラを搭載しています。

μPD720130、720133：バス・パワード

Q1. バス・パワード・デバイスの機器として設計できますか。

A1. IDE 機器に流れる電流が小さい場合、設計可能です。

μPD720130、720133：外部デバイス

Q1. シリアル ROM は必須ですか。

A1. はい、ベンダ ID、プロダクト ID やシリアル・ストリングなど、お客様の固有情報をシリアル ROM で設定していただく必要がありますので、シリアル ROM は必要です。

4.2 IEBus

IRQ(割り込み要求)の検出

Q1. IRQ(割り込み要求)の検出方法を教えてください。

A1. IRQ には、次の 2 通りの検出方法があります。

(1) ハード検出: IRQ 端子をホスト CPU の INT 端子に接続する。

(2) ソフト検出 : ポーリングで IRQ(FLG レジスタのフラグ)を検出する。

ただし(2)では、データ通信が高速か大量になると、取りこぼす可能性あるので、(1)を推奨します。

IRQ(割り込み要求)の発生

Q1. IRQ(割り込み要求)は、どのような場合に発生しますか。

A1. IRQ の発生条件は、次の 2 通りです。

(1) LSI が暴走した場合(FLG レジスタの RAW フラグが"1"となる)。

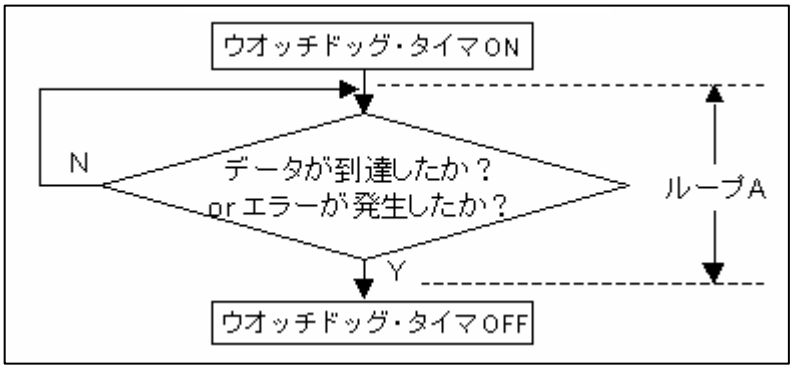
(2) RCR レジスタのリターン・コードが変化した場合。

リターン・コードは、ユニットがマスタかスレーブとして通信するときに、開始や終了の状態を表すものです。

暴走フラグ(ウォッチドッグ・タイマ)

Q1. 暴走フラグ(ウォッチドッグ・タイマ)について教えてください。

A1. μPD72042,72042B 内部で通信処理をしているファームウエアが、IEBus 上のデータ到達まで命令実行を保留させている間に、ウオッチドッグ・タイマ(WDT)を起動し、無限ループに陥ることを防止するフラグです。



データが到達したか、到達せずにエラーが発生すると、上図のループ A を抜けます。WDT には、その最大時間より大きな値を設定しており、外的な要因では暴走が発生しません。暴走の発生要因として、内部のハード／ファームウエアが、何らかの原因で異常をきたす場合を想定しています。

また、電源投入後、リセットをかけずに起動すると、タイマ値が不定で暴走が発生する可能性もあります。

バス・ラインの接続部品

Q1. バス・ラインに、ノイズ・フィルタなどのインダクタンス成分の部品を接続してもいいですか。

A1. いいえ、バス・ラインには、インダクタンス成分をもつ部品を接続しないでください。

インダクタンス成分があると、タイミング・エラーが発生して、通信エラーを誘発する恐れがあります。

マスタが NAK を受け付けて伝送が中止された場合

Q1. マスタが NAK を受け付けて、伝送が中止された場合、ホスト側はどのように認識するのですか。

A1. 通常、通信でスレーブが正しくデータを受け付けなかった場合は、マスタに NAK を返します。これは、4 通りの場合があり、スレーブ・アドレス・フィールド、コントロール・フィールド、電文長フィールド、データ・フィールドの、それぞれの最後です。マスタが NAK を受け付けて伝送を中止すると、この通信は途中終了となり、リターン・コードが変化します。このとき IRQ を発生して、ホストに通知します。

CMR レジスタへのコマンド書き込み

Q1. 通信中に、CMR レジスタにコマンドを書き込んだ場合、実行までにどのくらい待つ必要がありますか。

A1. 通信中に CMR レジスタへコマンドを書き込むと、レジスタは書き替えられますが、通信が終了するまで、そのコマンド実行を待機しなければなりません。したがって、コマンドの実行までには、最大で 1 通信フレーム時間だけ待つ必要があります。たとえば、モード 1 では 15.2ms です。

スタンバイ・モードの要求

Q1. 通信中に、スタンバイ・モードを要求してもいいですか。

A1. 通信中には、スタンバイ・モードを要求しないことを推奨します。通信が終了してキャリア・センス状態にならないと、スタンバイ・モードには入れません。また、通信中にスタンバイ・モード要求した場合、スタンバイ・モードに入るまでは、シリアル I/O を介した内部レジスタの読み出しを禁止しています。

μPD6708 と μPD72042B の混在使用

Q1. μPD6708 と μPD72042B を混在で使用する場合、注意点はありますか。

A1. μPD72042B は外付けの保護抵抗を考慮して設計されていますが、μPD6708 では考慮されていません。
混在で使用する場合、終端抵抗と保護抵抗の推奨値は、次のようになります。

終端抵抗	: 両端 120Ω または片端 60Ω
μPD72042B の保護抵抗	: 180Ω
μPD6708 の保護抵抗	: 0Ω

受信エラー: μPD6708

Q1. 10 バイト程度の通信では問題ありませんでしたが、26 バイトの受信でエラーが発生します。
エラーの内容は、データ・フィールドの ACK エラーで、21 バイト目のようです。

A1. μPD6708 の受信バッファは 20 バイト分なので、20 バイトの受信時点でバッファ・フルの割り込み要求が発生します。このとき、規定時間内にそのデータを読み出さないと、21 バイト目のデータを取り込めず、ACK を返さずスレーブ受信途中終了となります。、受信バッファ・フル割り込み要求の発生後、規定時間(約 390μs)以内にバッファのデータを読み出してください。このとき、1 バイトしか読み出さないと、次の 21 バイト目のデータを受信した時点で、再度受信バッファ・フルの割り込み要求が発生します。

IEBus のモニタ方法

Q1. μPD6708 か μPD72042、72042B で、IEBus のバス・ラインをモニタすることはできますか。

A1. いいえ、できません。

通信フラグ : μPD72042B

Q1. μPD72042B のデータ・シートに「6.1 通信フラグ」がありますが、通信フラグとはなんですか。

A1. メイン・ルーチンと割り込み処理ルーチンの間で、引数かグローバル変数として定義したものです。データ・シートの 6 章は、マイコンで μPD72042B を制御する処理フロー例で、通信フラグの例として命名しています。

転送速度 : μPD72042B

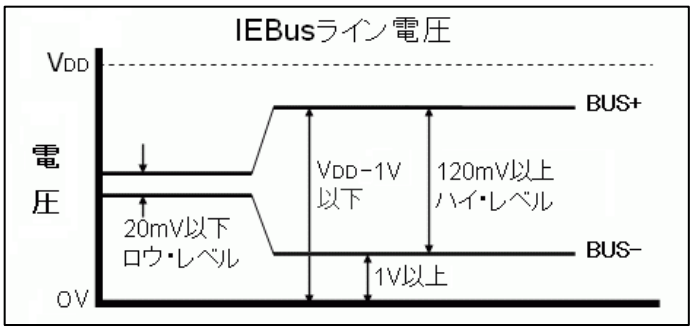
Q1. μPD72042B のデータ・シートに「通信速度が 6MHz、モード 1 であれば、実行伝送速度が 17kbps」と記載されていますが、これはデータ・フィールド内のデータ・ビット（実際のデータ）だけの転送速度ですか。

A1. いいえ違います。17kbps は、ヘッダ部やパリティ・ビット、アクノリッジ・ビットなども含めた値です。
データ・フィールド 1 バイト分（パリティ・ビット、アクノリッジ・ビット含む）の時間は、モード 1 で約 410 μs です。

BUS+端子と BUS-端子の入力電圧

Q1. IEBus ドライバ／レシーバの電気的特性で、入力電圧と同相入力電圧の違いはなんですか。

A1. ハイ・レベル入力電圧／ロウ・レベル入力電圧は、BUS+と BUS−間の電位差（電圧）を規定しており、120mV 以上でハイ・レベル、20mV 以下でロウ・レベルと認識されます。
一方、ハイ・レベル同相入力電圧／ロウ・レベル同相入力電圧は、BUS+とグランド間／BUS−とグランド間の電位差を規定しており、ハイ・レベルの場合は BUS+と BUS−が 1.0V から V_{DD}−1.0V までにあり、かつ BUS+と BUS−間の電位差が 120mV 以上ある必要があります。またロウ・レベルの場合は、BUS+と BUS−が 0V から V_{DD} までにあり、BUS+と BUS−間の電位差が 20mV 以下である必要があります。



IEbus 上でのハイ／ロウ・レベルとロジックの 0/1 の関係

Q1. IEbus 上でのハイ・レベル／ロウ・レベルと、ロジックの 0/1 の関係はどのようになっていますか。

A1. ロウ・レベル（バスの線間（BUS＋、BUS－）の電位差が 20mV 以下）はロジックの 1 になり、ハイ・レベル（電位差が 120mV 以上）はロジックの 0 になります。

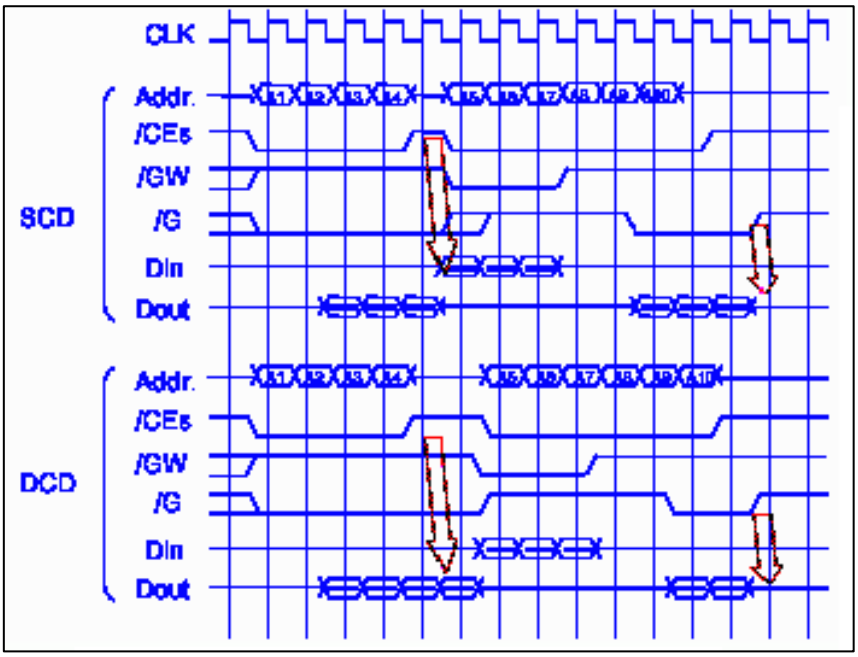
5. メモリ

5.1 同期 SRAM

パイプライン SRAM の仕様

Q1. 同期型のパイプライン SRAM の仕様で、SINGLE CYCLE DESELECT（SCD）、DOUBLE CYCLE DESELECT（DCD）、PIPLINED BURST（PB）、FLOW THROUGH（FT）の違いについて教えてください。
また、リニアとインタリーブ・バーストの違いについても教えてください。

A1. 基本的に、PB/FT は出力レジスタのある／なしの違いです。
SCD/DCD は、/CE 信号で出力をディスエーブルにした場合、データがハイ・インピーダンスになるのが 1 クロック後か、2 クロック後かの違いです。



PB/FT と SCD/DCD は、別次元の話です。
PB には SCD/DCD の両方が存在しますが、FT は SCD だけとなります。
PB/FT については、ライトでは同じで、アドレス(外部／内部)とデータを同一サイクルで取り込みます。
リードでは、FT はアドレス入力の次のサイクルでデータが出力されますが、PB は 2 クロック後のサイクルで出力されます。
FT の方が 1 クロック後にデータが出てくるので良いように思えますが、FT より PB の方が高い動作周波数に対応できますので、一長一短です。
用途は特に限定されず、上記の違いによって、お客様が都合の良いものを使っていただくことになります。

リニアとインタリーブ・バーストについては、もともと PB/FT が CPU のキャッシュの仕様として発展した物で、CPU のアドレス・インクリメントの方法を選べるようにしてあるだけであり、都合の良い方をご使用ください。

Late Write シンクロナス SRAM

Q1. Late Write シンクロナス SRAM とはなんですか。

A1. 同期式メモリで、アドレスを入力するサイクルの、次のサイクルで書き込みデータを入力する動作が Late Write 動作で、この動作を基本にしている同期式 SRAM を特に「Late Write シンクロナス SRAM」といいます。

Q2. 標準シンクロナス SRAM と、Late Write シンクロナス SRAM の動作の違いはなんですか。

A2. 回路構成上の大きな違いとして、Late Write シンクロナス SRAM に内蔵する「Write Address Register」があります。これにより、書き込みデータを入力するサイクルで、リード動作のアドレスと制御信号が入力されても、いったん Write Address Register にアドレスを退避させ、リード動作が完了してから、そのアドレスを使って書き込み動作を完了させることができます。

これにより、「リード→ライト→リード・・・」の繰り返しで、ウエイト・サイクルを少なくすることができます。

ZEROSB SRAM

Q1. ZEROSB SRAM とはなんですか。

A1. リード／ライト切り替え時にデッド・サイクルが生じない、ネットワーク機器のバッファ・メモリに最適な高速同期式 SRAM です。

ルネサス エレクトロニクスでは ZEROSB という商標ですが、IDT 社は ZBT SRAM、三星電子は NtRAM という商標で販売されています。

Q2. 標準シンクロナス SRAM と ZEROSB SRAM の動作の違いはなんですか。

A2. 標準シンクロナス SRAM では、アドレス入力サイクルと同じサイクルで、書き込みデータを入力するのが基本動作なので、「リード→ライト→リード・・・」の動作が続く場合には、リード・サイクルとライト・サイクルの間にウエイト・サイクルを必要とします。

ZEROSB SRAM は、Late Write シンクロナス SRAM と同様に、Write Address Register を持っています。この点で ZEROSB SRAM は、Late Write シンクロナス SRAM の考え方を発展させた仕様です。

ZEROSB が持つ Write Address Register は、フロー・スルー品で 1 アドレス分、パイプライン品で 2 アドレス分あり、その機能で、「リード→ライト→リード・・・」でのウエイト・サイクルを完全になくすことができます。

メモリの動作は、必ずしも「リード→ライト→リード・・・」と交互に続くとは限らないため、標準シンクロナス SRAM と ZEROSB SRAM は、目的に応じて使い分けられます。

6. リニア IC

6.1 オペアンプ／コンパレータ …755
6.1.1 オペアンプ …755
6.1.2 コンパレータ …770
6.1.3 相違点と共通事項 …776
6.2 電源用 IC …785
6.2.1 電源用 IC の種類 …785
6.2.2 電源の構成 …790
6.2.3 三端子レギュレータ …791
6.2.4 シャント・レギュレータ …804
6.2.5 スイッチング・レギュレータ …806
6.2.6 製品照会 …810
6.2.7 共通事項 …811

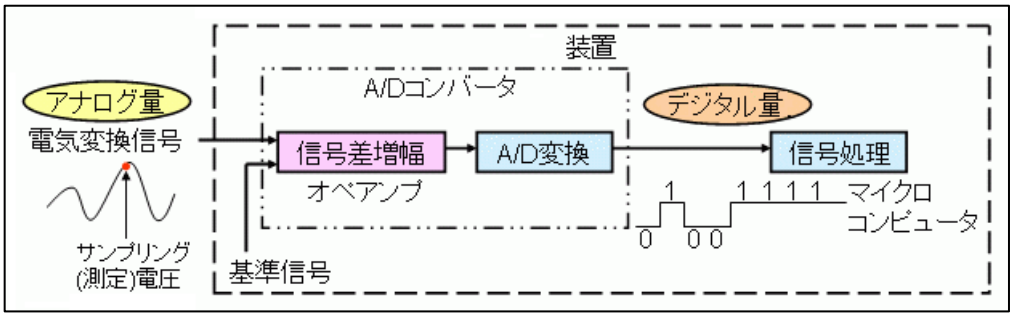
6.1 オペアンプ／コンパレータ

6.1.1 オペアンプ

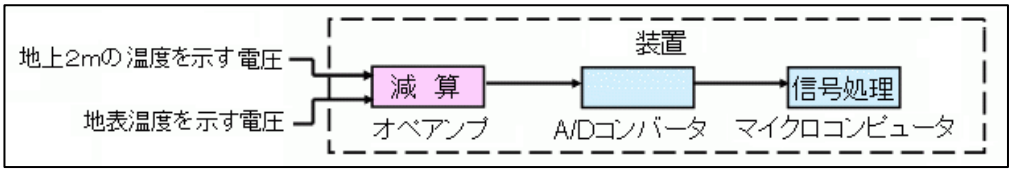
基　　礎

Q1. オペアンプの役割は、どのようなことですか。

A1. 我々を取り巻く環境は、風・光・音・気温・湿度など、連続的な変化量（アナログ量）です。一方、我々が使用する装置では、デジタル処理がほとんどなので、装置の入り口ではアナログ信号処理が必要です。この主役がオペアンプ(演算増幅器)です。一定時間ごとに地表面温度を測定して、装置に記憶する応用で、具体的なアナログ信号処理の事例を説明します。まず、地表面温度を温度センサで電圧（アナログ値）に変換します。この電圧が微小な場合、その変化を読み取れないおそれがあり、電圧を増幅する必要があります。この増幅をオペアンプで実現できます（電圧に変換された信号と基準信号の差を増幅）。その後、増幅された電圧は A/D 変換でデジタル値に変換され、マイコンで処理されます。



他のオペアンプ応用事例として、2 点間(地表面と地上 2m)の温度差を測定、記憶する応用を考えます。2 つの温度センサを用い、この 2 点の温度差を計測する(減算)ことが、オペアンプで次のようにできます。



オペアンプでは、このほか加算、微分、積分の演算機能も実現できます。

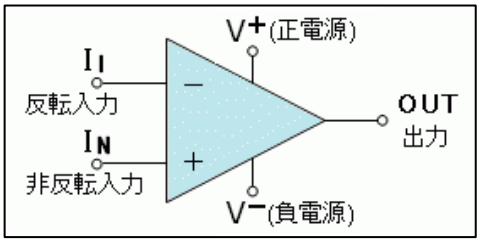
なお、デジタル回路とのインタフェース(やり取り)には、コンパレータ(比較器)が使われます(コンパレータの基礎参照)。

Q2. オペアンプとはなんですか。

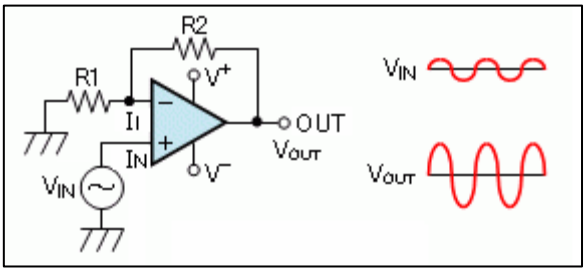
A2. Operational Amplifier(演算増幅器)の略で、理想的には、次のような差動増幅器です。

増幅率(ゲイン)　：無限大
入力インピーダンス　：無限大
出力インピーダンス　：0
入力から出力の遅延　：0

負帰還[注]で使用し、増幅や多様なアナログ信号処理(加算・減算・微分・積分など)が行えます。＋入力(I_N:非反転入力)、－入力(I_I:反転入力)、出力(OUT)、電源(V^+, V^-)の 5 本の端子があり、三角形で表します。



最も一般的な使用例は、次のような増幅回路です。

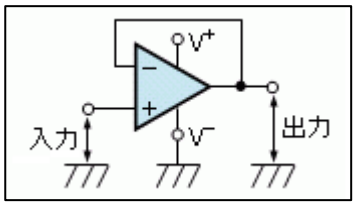


この例では、入力信号を(1＋R2／R1)倍に増幅して出力します。
R1=1kΩ, R2=10kΩ なら、増幅率 11 倍のアンプとなります。このため、抵抗値を変えるだけで、自由に増幅率を設定できます。図の回路では、＋端子に入力をしているので、入力と出力は同相ですが(非反転増幅器)、R1 側に入力をすると、出力は反転します(反転増幅器)。

注　負帰還とは、出力と－入力を抵抗やコンデンサ、ダイオード、トランジスタなどを介して接続し、出力信号を－入力に「フィードバック」することをいいます。

Q3. 単電源オペアンプとはなんですか。

A3. 単電源で使用したときに GND 入力できる、つまり V^-を同相入力電圧範囲に含んでいるオペアンプです。



Q4. 単電源用と 2 電源用の違いはなんですか。

A4. 厳密な区分はありませんが、一般に**同相入力電圧**の範囲が異なります。V^-と同じ電圧までの入力で動作する製品が単電源用です。従来の 2 電源用は、V^-から約 1.5V の範囲では使用できません。

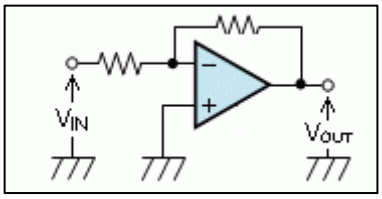
Q5. 入力電圧と出力電圧を電源電圧の範囲まで使用できる、レール・トゥ・レールのオペアンプはありますか。

A5. 残念ながら、当社にはレール・トゥ・レール品はありません。

Q6. イマジナリ・ショートとはなんですか。

A6. オペアンプで負帰還をかけて、＋入力と－入力の端子間に電位差がない状態をいいます。イマジナリ・ショートと呼ぶのは、両端子が同電位であることから、あたかもショート状態であると見なせるからです。

次図は**反転増幅回路**の例で、マイナス入力は、プラス入力と同じ GND レベルとなります。

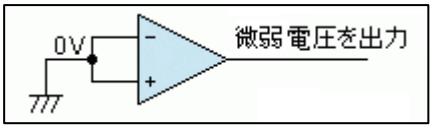


オペアンプは、イマジナリ・ショート状態を常に保とうとする帰還特性を利用し、増幅動作を行っています。

特　　性

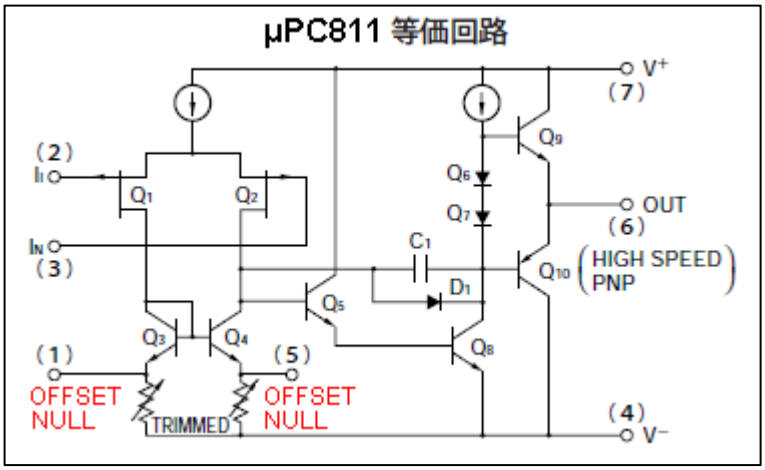
Q1. オフセット電圧とはなんですか。

A1. オペアンプ入力端子間の電圧差が 0V のとき、出力電圧は 0V となるはずです。しかし、入力段差動増幅回路の＋入力と－入力のトランジスタの V_{BE} に差があるので、実際には微弱電圧が出力されます。これをオフセット電圧と言います。相違点と共通事項の「入力オフセット電圧（V_{IO}）とは？」をご参照ください。

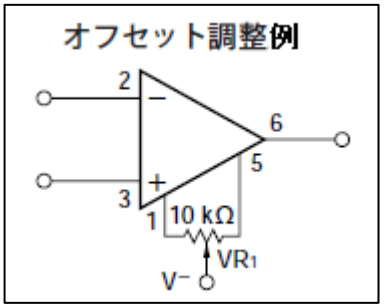


Q2. OFFSET-NULL によるオフセット調整とは、どのようなことですか。

A2. 入力段差動増幅回路の V_{BE} 差を平衡させる調整です（μPC811 などは調整用の OFFSET-NULL 端子あり）。

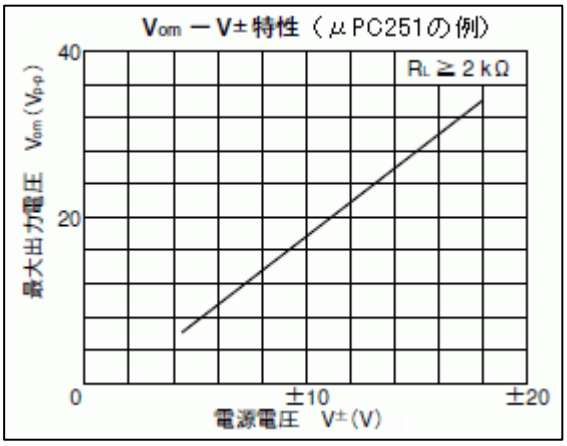


この端子に可変抵抗を外付けし、入力段の平衡を取ることで、オフセット電圧の調整を行います。



Q3. 最大出力電圧(Vom)とは、どのような特性ですか。

A3. 規定の電源電圧でオペアンプの出力が飽和する限界値です。正電源側と負電源側の両方の値があります。条件として、出力端子とGND間の負荷抵抗(2kΩ、10kΩ)か出力電流を設定します。出力が正方向か負方向(内部回路による)へ飽和するまでのダイナミック領域を意味し、負荷抵抗にも制限されます。オペアンプの種類にもよりますが、通常、最大出力電圧はV^+より1～1.5V下がった電圧と、V^-から1～1.5V上がった電圧の範囲までとれます。したがって、電源電圧$V^{\pm}$が変化すると、最大出力電圧も変化します。



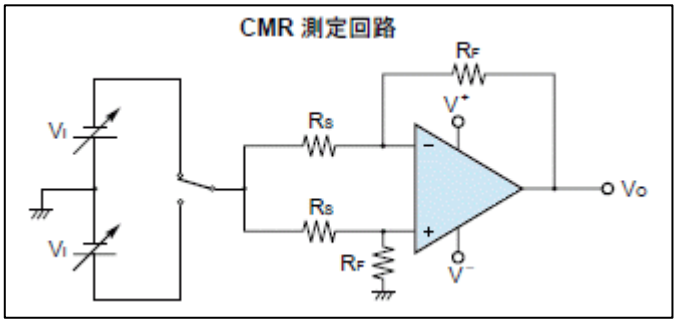
Q4. 出力印加電圧とは、どのような特性ですか。

A4. 特性劣化や破壊を起こさずに、出力端子に印加できる電圧範囲です。電源 ON/OFF 時の過渡状態も含めて、定格を越えないようにご注意ください。なお、オペアンプとして得られる出力電圧は、電気的特性の最大出力電圧の範囲内です。

Q5. CMR(Common Mode Rejection：同相信号除去比)とは、どのような特性ですか。

A5. 入力電圧を変化させた場合に、変動した入力オフセット電圧との変動比を示します。

$$CMR = \Delta V_I / \Delta V_{IO}$$



通常は、DC 変動分が対象です。この値が大きければ、ΔV_Iが同じでも、ΔV_{IO}が小さいことになります。つまり、2 入力に対して、同一条件で影響を与えるモードの信号(ノイズ、ドリフトなど)が、どれだけ抑圧されるかの程度を表します。なお、CMR は出力の誤差に影響する一因ともなります(インフォメーション「オペアンプの用語と特性」の「2.7　同相信号除去比の影響」をご参照ください)。

Q6. SVR(電源変動除去比)とは、どのような特性ですか。

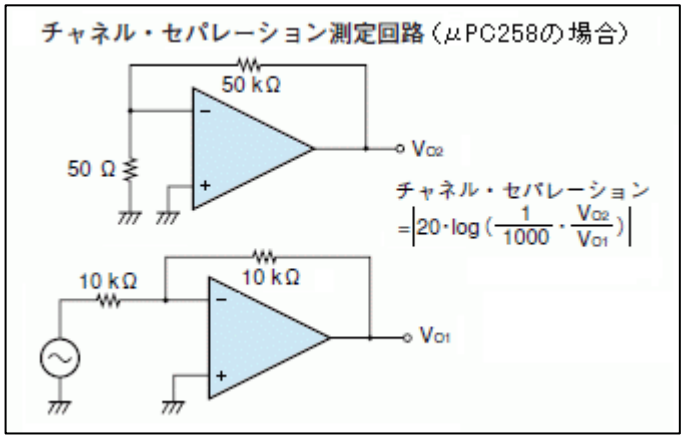
A6. 電源電圧を変化させた場合、変動する入力オフセット電圧との変動比です。電源のリップルやノイズに対して、デバイスとしてどの程度抑えられるかを示します。

$$SVR = \Delta(V^+ - V^-) / \Delta V_{IO}$$

例　μPC801 の SVR は typ.76dB です。これより、SVR は 6.3×10^3 倍であり、たとえば電源電圧が外乱によって 0.5V 変動した場合、V_{IO} は 79μV の変動となります。

Q7. チャネル・セパレーションとは、どのような特性ですか。

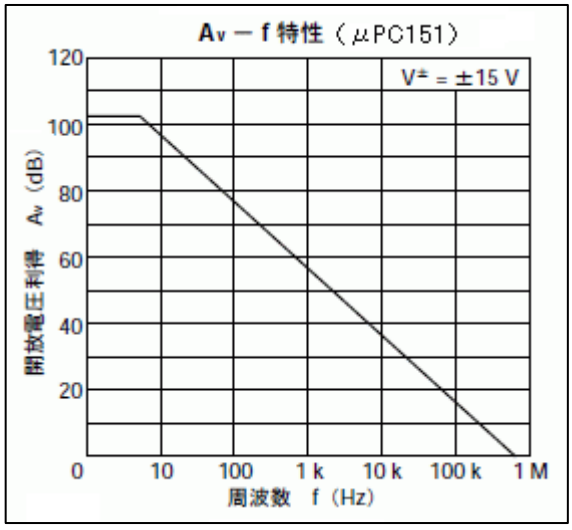
A7. オペアンプがパッケージ中に複数(2、4 チャネル)内蔵されている場合、動作をさせたオペアンプが、他のオペアンプに及ぼす影響度合いを示します。



Q8. 大振幅電圧利得(Av)とは、どのような特性ですか。

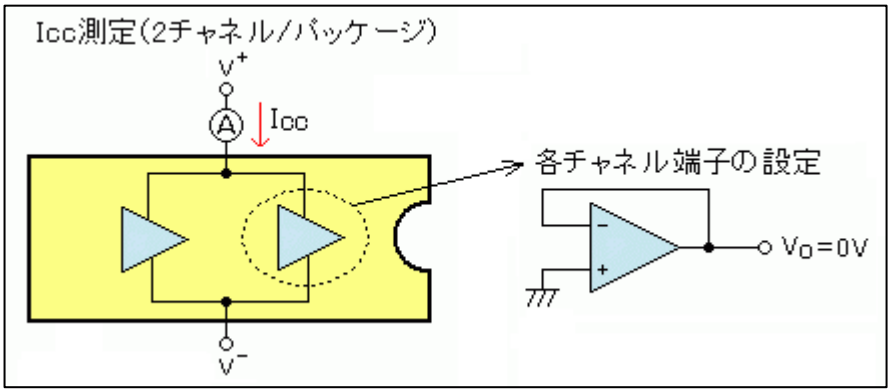
A8. 開放電圧(負帰還をかけない)利得ともいわれ、データ・シートでは、直流での差動電圧利得を示します。100dB(10 万倍)前後が、オペアンプとして標準的な値です。

次に、μPC151 の Av–f 特性を示しますが、周波数が高くなるに伴って、Av は減少する傾向があります。



Q9. 回路電流(Icc)は、何チャネル分を示すのですか。

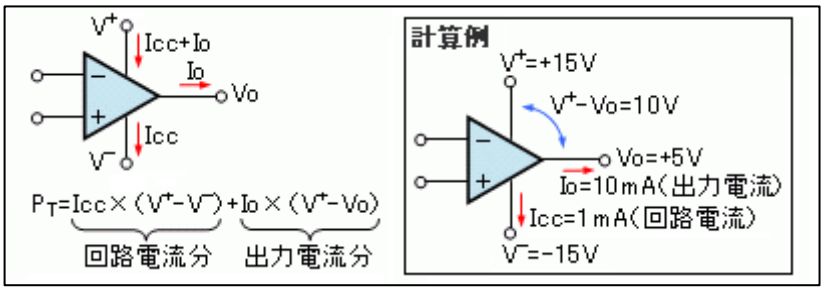
A9. 回路電流は、1 パッケージに内蔵される全チャネル(2 チャネル：デュアル形、4 チャネル：クワッド形)の合計です。なお、各チャネルに負荷がない状態（出力電流が 0A）での、電源端子に流れる電流を規定しています。



Q10. オペアンプの消費電力は、どのように決まるのですか。

A10. オペアンプの回路電流と、出力の負荷電流で計算します。出力電流の流れ出しと流れ込みで異なります。

(1) 出力電流が流れ出しの場合



たとえば、上の図の場合は次のとおりです。

①オペアンプ自身の回路電流による電力消費

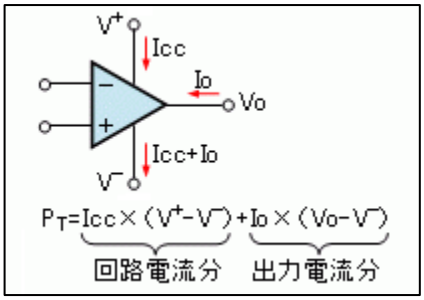
$I_{CC} \times (V^+ - V^-) = 1(mA) \times 30(V) = 30(mW)$

②出力負荷電流による電力消費

$I_O \times (V^+ - V_O) = 10(mA) \times 10(V) = 100(mW)$

①＋②＝130(mW)

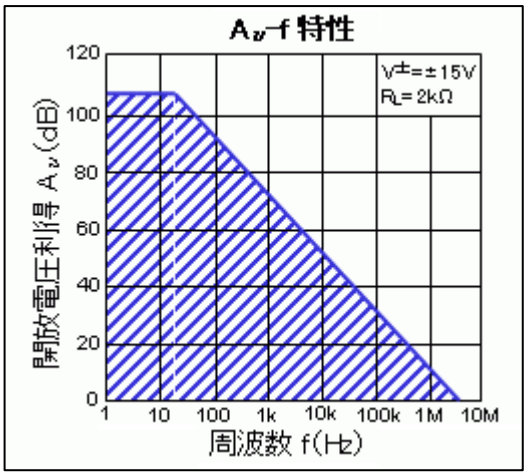
(2) 出力電流が流れ込みの場合



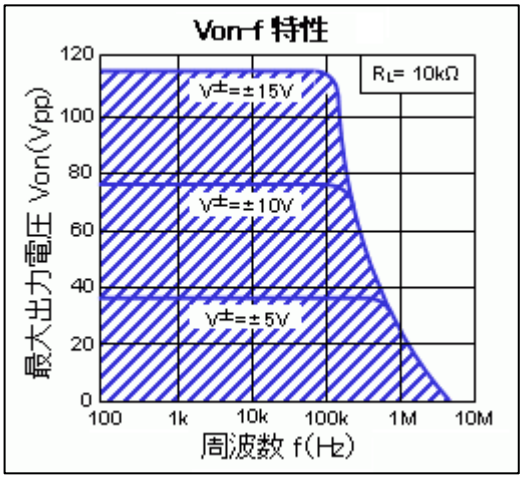
Q11. オペアンプの特性は、どのように判断すればいいのですか。

A11. 「利得帯域幅積(G･B 積)」、「ゼロクロス周波数」、「スルーレート」の 3 項目で判断できます。

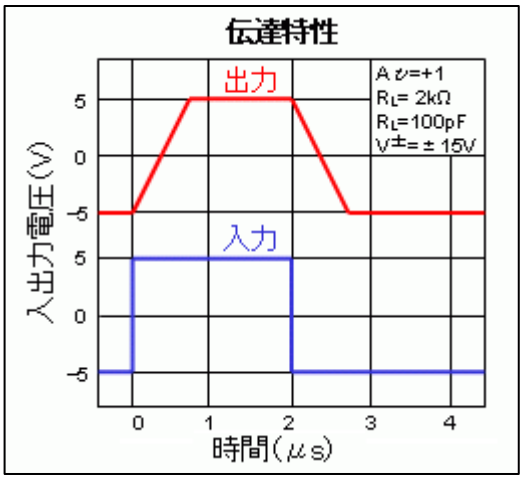
各規格値が大きいと高速･広帯域ですが、処理信号の周波数に応じ、適切なオペアンプを選択する必要があります。これらの項目が電気的特性表にない場合、特性曲線の次の 3 つのグラフで判断してください。



オペアンプの差動ゲインと周波数の関係を示します。オペアンプは、DC から低周波域では高ゲインですが、高周波域ではゲインが低下します(斜線部分が設計可能範囲)。



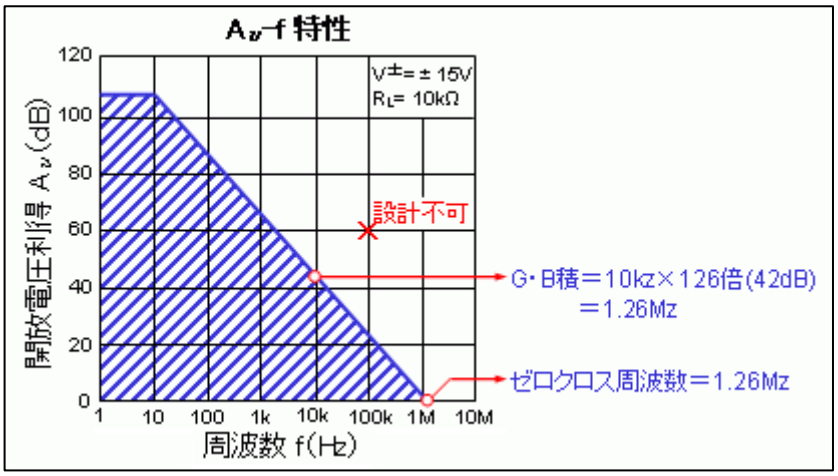
出力が歪まずに振動できる最大周波数（サイン波）を示します（斜線部分が設計可能範囲）。



パルス入力に対する、出力応答特性を示します。

Q12. G･B 積とは、どのような特性ですか。

A12. 利得（Gain）と帯域（Band Width）の積です。通常は f＝10kHz で、Aυ–f 特性の範囲内で規定できます。



斜線部分が設計可能範囲です。

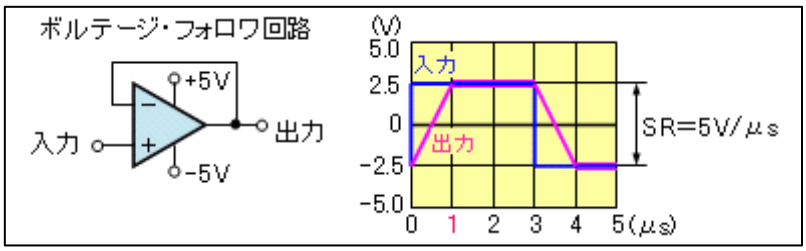
たとえば、この例のオペアンプでは、100kHz の周波数で、60dB の増幅回路は実現できません。

Q13. ゼロクロス周波数とは、どのような特性ですか。

A13. 開放電圧利得が 0dB（1 倍）となる周波数のことで、Aυ–f 特性から求められます。

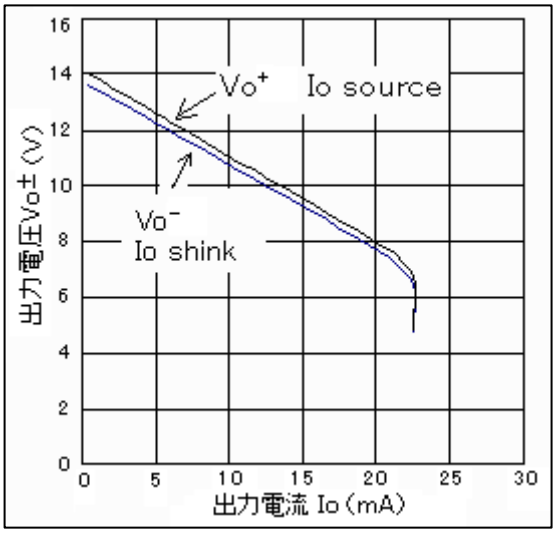
Q14. スルー・レートとは、どのような特性ですか。

A14. スルー・レート（SR）というのは、パルス入力応答性を規定するもので、ボルテージ・フォロワ回路（ゲイン=1）での出力応答を 1μs あたりの電圧振幅で測定した値です。これが大きいほど、高速のオペアンプです。



Q15. μPC803 の出力特性を教えてください。

A15. たまたまあった製品でデータをとってみました。なお参考で、数値を保証するものではありません。



測定温度：常温（約 25℃）

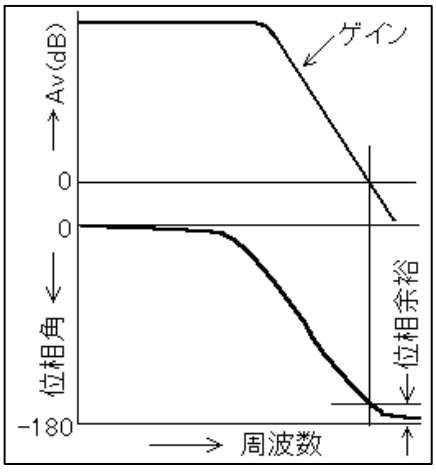
電源電圧：±15V

Q16. μPC1251 は 15V での「V_o vs I_o（shink）特性」が示されていますが、5V のときの特性を教えてください。

A16. 5V では電流値が少し下がりますが、大きな変化はありません。15V での特性を参考にしてください。

Q17. 負荷に容量をつけると、発振しやすいのはなぜですか。

A17. 発振は、出力から入力へ帰還がかかることで生じます。ゲインが 1 以上（右図では 0dB）のときに、出力を入力へ正帰還させると、外部から信号がなくても自分自身で信号を増幅することになり、発振を開始します。負帰還アンプでは、入力と出力の位相が逆（180 度）なので、負帰還ループでの位相が 180 度遅れると、全体の遅れは 360 度で、正帰還と同じです。この状態で「**帰還量×増幅率**」が 1 を上回ると、発振を開始します。出力端子に容量をつけると、位相の遅れが生じるので、抵抗負荷より発振しやすくなります。



発振しない程度は、次図のようにゲインが 1（0dB）の周波数の位相差（位相余裕）、または位相が 180 度となる点のゲイン（ゲイン余裕）で示されます。

理想的な増幅器では、浮遊容量や GND 配線の影響がありませんが、実際はこれらの影響で帰還がかかってしまいます。このため、発振が起こります。詳細は資料「オペアンプの用語と特性」をご参照ください。

Q18. オペアンプのデータ・シートに「入力抵抗」が記載されていませんが、規定はないのですか。

A18. 入力抵抗（Ω）は、電気的特性に記載の入力バイアス電流（A）の逆数で、単位電圧（1V）あたり R＝V/I で求められます。μPC151 を例にすると、次のとおりです。

	MIN.	TYP.	MAX.	単位
入力バイアス電流	―	80	500	nA
入力抵抗	2	12.5	―	MΩ

Q19. 入力換算電圧性ノイズ(en)/入力換算電流性ノイズ(in)とは、どのような特性ですか。

A19. 特にオーディオのプリアンプなどでは、小信号のゲインを高くとるために、この特性が重要視されます。オペアンプの中で発生するノイズにも、オフセットと同様に雑音電圧と雑音電流の2種類があります。入力オフセット電圧に乗るのが入力換算電圧性ノイズ、入力バイアス電流に乗るのが入力換算電流性ノイズです。入力換算値なので、入力オフセット電圧と同様に増幅率倍されて、出力に現れます。

入力換算電流性ノイズは、信号源抵抗との積で入力換算電圧性ノイズと見なされ、電圧性ノイズに加算されます。したがって、信号源抵抗が高くなるほど、出力されるノイズは大きくなります。

Q20. デュアルオペアンプで、チャネル間の電気的特性のペアー性はどうですか。

A20. 一般に、デュアルオペアンプの電気的特性で、ペアー性を問題とするのは、入力オフセット(V, I)です。入力オフセットの発生要因は、入力作動段の素子のばらつきによります。入力オフセットのばらつきは、データ・シートの電気的特性に記載しています。

Q21. 複数(4、2)チャネル内蔵する製品で、推奨動作条件の負荷容量 C_L は、1 チャネルに適用されるのですか。

例　μPC842

推奨動作条件

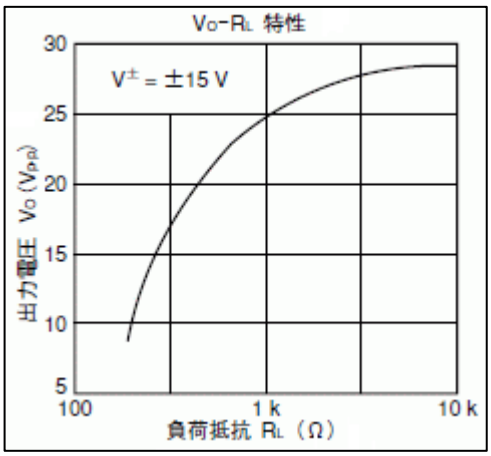
項　目	略　号	MIN.	TYP.	MAX.	単　位
電源電圧（V^- = GND）	V^+	+3	+5～+30	+32	V
電源電圧（両電源）	$V^\pm$	±1.5		±16	V
出力電流	Io			±10	mA
負荷容量（A_v = +1のとき）	C_L			1 000 注	pF

注 フィードバック抵抗（R_f）＝0Ωのときの値です。

A21. はい、1 つのパッケージに内蔵される複数チャネルのうち、1 チャネルに適用します。

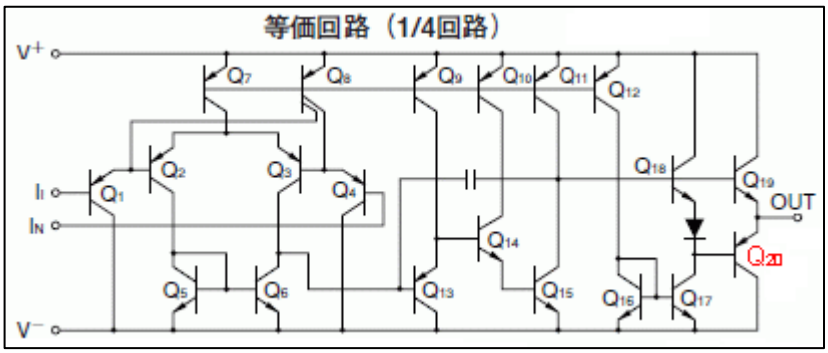
Q22. μPC4558 の出力電流はどの程度流すことができますか。

A22. μPC4558 は出力電流を規定していません。その代わりに、データ・シートに V_O-R_L 特性（参考）を記載しています。お客様の負荷抵抗(R_L)に対応した出力電圧(Vo)から逆算して、電流値を求めてください。

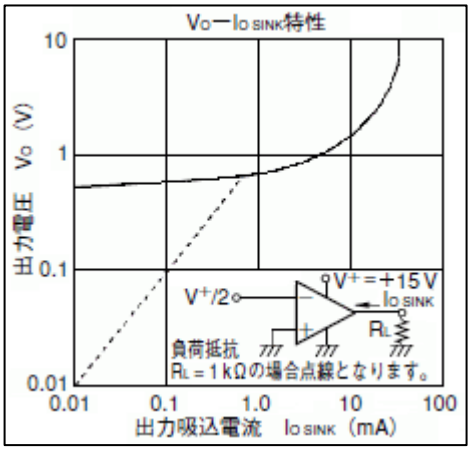


Q23. μPC844 は,、出力電圧が V⁻＋0.7V 以下で、出力電流吸い込み動作をしなくなるのはなぜですか。

A23. 等価回路の出力段トランジスタ Q20 はエミッタ・フォロワ構成です。本回路が動作するには、Q20 のエミッタ−ベース間電圧に必要なオン電圧の約 0.7V を与える必要があります。このため、出力電圧の制御可能範囲は約 0.7V 以上となり、オペアンプ自身で 0.7V 以下の電圧を出力することができません。



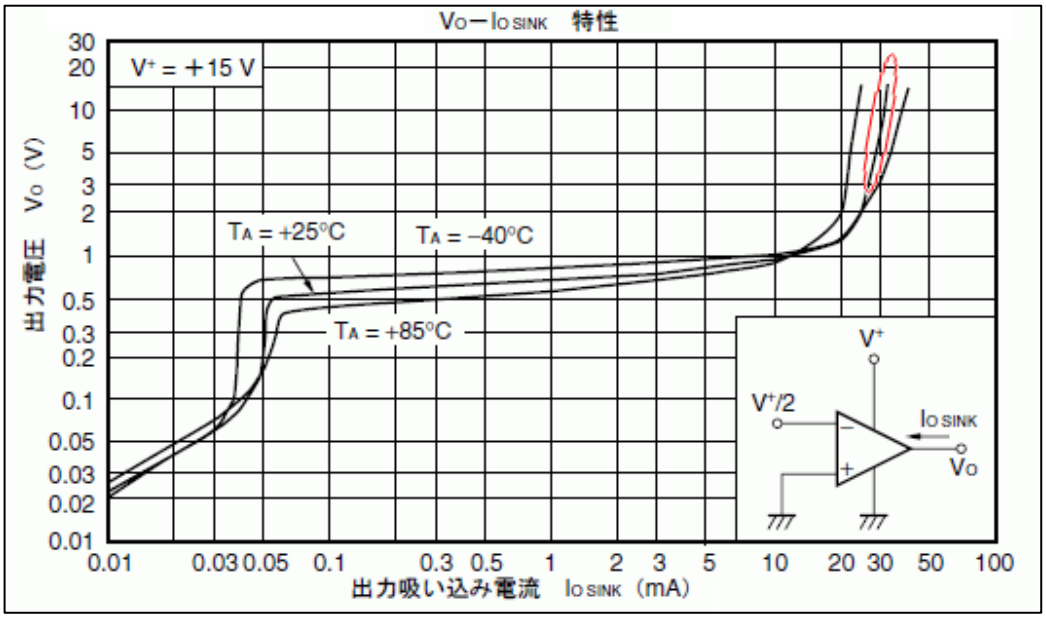
参考データとして「Vo－IoSINK 特性」をデータ・シートに掲載しています。



Q24. μPC324 の出力を短絡すると、出力電流は制限されますか。また、品質的ストレスになりますか。

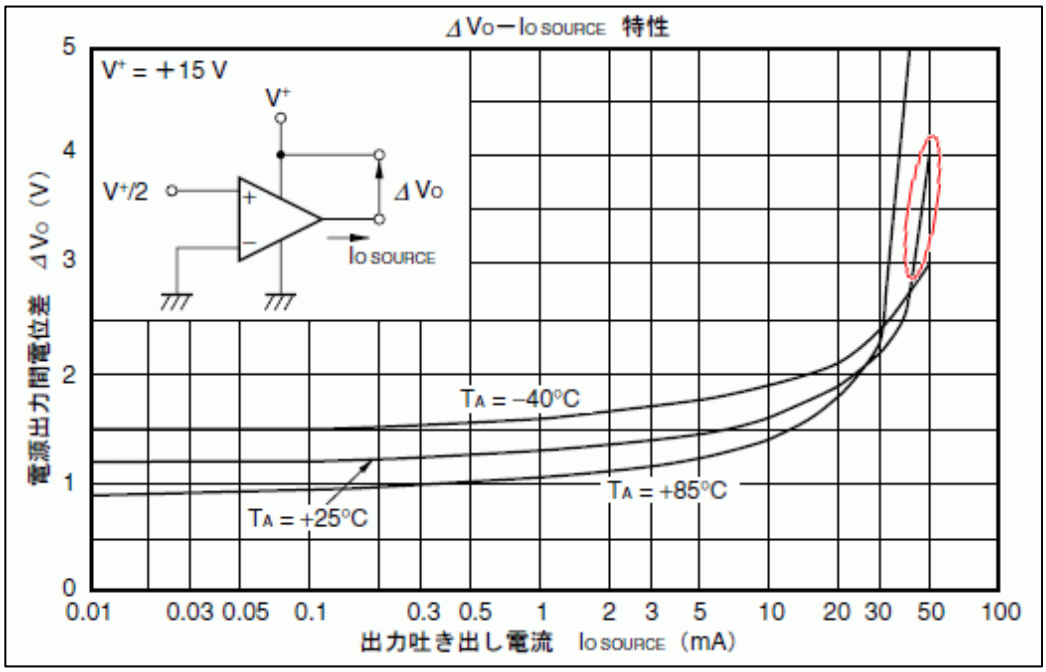
A24. μPC324 は出力短絡保護回路を内蔵しており、出力電流は次図(参考)のような制限を受けます。

■V⁺へ短絡



T_A＝＋25℃の曲線で、$I_{O\ SINK}$ は 30mA程度に制限されています。

■V⁻（GND）へ短絡



T_A＝＋25℃の曲線で、$I_{O\ SOURCE}$ は 50mA程度に制限されています。

なお、「短絡状態が継続」でも、絶対最大定格の「全損失 P_T」以内なら、品質的ストレスにはなりません。

絶対最大定格（T_A = 25°C）

項　目	略　号	μPC451C, 451C (5)	μPC451G2, 451G2 (5)	μPC324C, 324C (5)	μPC324G2, 324G2 (5)	単　位
全損失	P_T	570 注4	550 注5	570 注4	550 注5	mW
出力短絡時間 注6	ts	無限大				s

注4. T_A≦＋50°C での値です。T_A>50°C では−7.6 mW/°C でディレーティングしてください。

5. T_A≦＋25°C での値です。T_A>25°C では−5.5 mW/°C でディレーティングしてください。

6. V⁺≦＋15 V，任意の 1 チャネルのみ。また全損失および 注4,5 のディレーティング以下で使用してください。

応　用

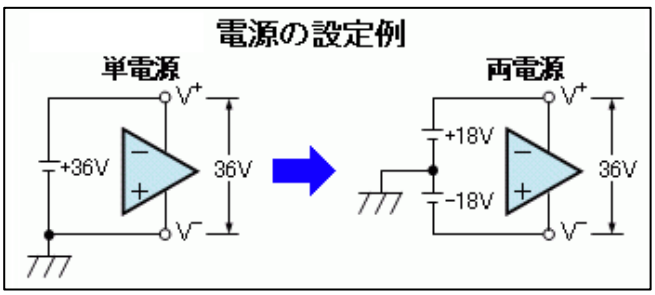
Q1. 2 電源のオペアンプを単電源で使えますか。

A1. ほとんどのオペアンプは単電源でも使えますが、GND 入力、GND 出力ができないので、入力にバイアスをもたせる必要があります。また、入力と出力の電圧範囲に制限があります。

入力トランジスタの形式によって、入力電圧の範囲が変わります。利用可能な電圧範囲を「同相入力電圧」に記載しています。製品ごとに、データ・シートでご確認ください。なお、規格は 2 電源で記載しています。

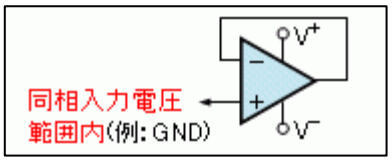
Q2. 単電源オペアンプを両電源オペアンプとして使用することは可能ですか。

A2. 使用は可能です。ご使用に際しては電源電圧範囲、同相入力電圧範囲をご確認ください。

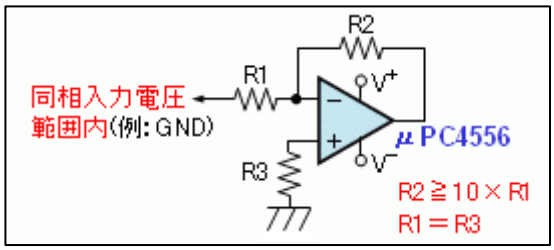


Q3. 複数チャネル内蔵で、使用しないオペアンプは、どのように処理すればいいですか。

A3. 入力端子をオープンのままにしても破壊はしませんが、空チャネルの動作が不安定となって、IC 内部の電流バランスがくずれる可能性があります。このため、入力端子はボルテージ・フォロワとして、＋入力を同相入力電圧範囲内（たとえば GND）の電位に接続してください。



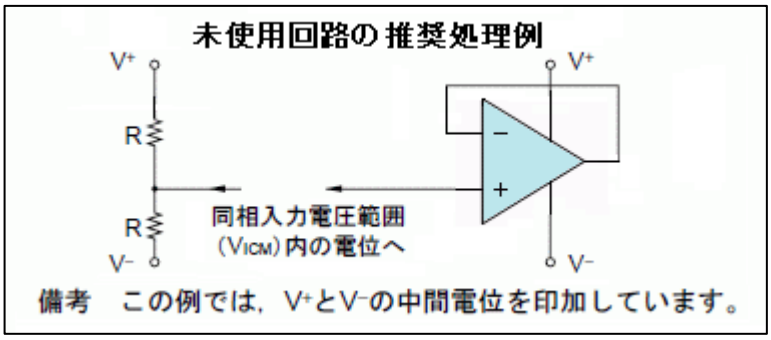
ただし、μPC4556 の場合には、増幅率 10 倍以上の増幅回路にする必要があります。



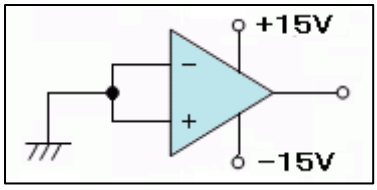
なお、空き端子の処理方法は、電源供給形態に依存しません。たとえば、単電源オペアンプ μPC324 では、次のような電源供給形態が設定できます。



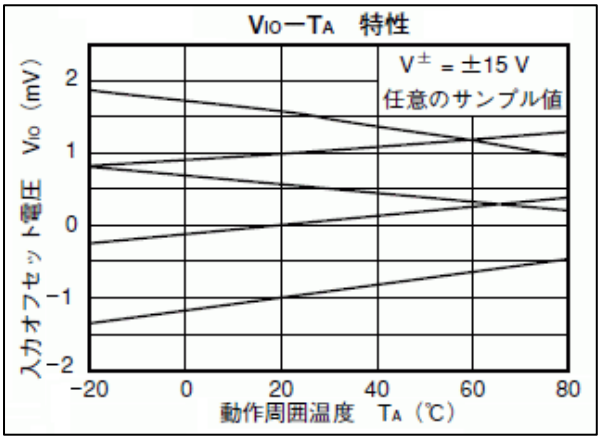
このように、電源供給形態が異なっても、推奨接続のボルテージ・フォロワで空き端子の処理をします。このとき、＋入力端子は同相入力電圧範囲内の電位に設定する必要があります。



Q4. デュアルオペアンプで、1チャネルを使用しません。$V^{\pm}$＝±15V で、空きチャネルの±入力端子を 0V に接続し、出力端子をオープンにすると、問題はありますか。



A4. 出力を入力に戻さないので、利得がひじょうに大きくなります。この状態で、たとえば入力オフセット V_{IO} などの小さな変化でも、出力がハイ・レベルかロウ・レベルに振り切れる動作をします。この V_{IO} は、動作周囲温度 T_A の変化に伴って変化します。次図は μPC4560 の V_{IO}–T_A 特性曲線ですが、V_{IO} が 0V を横切ることで、出力変化が生じることを示しています。この出力変化による影響が、使用チャネルに対して考えられます。
空きチャネルについては、推奨処理（ボルテージ・フォロア）をしてください。



Q5. オペアンプはコンパレータとして使えますか。

A5. 使えますが、次の 2 点にご注意ください。

- 位相補正コンデンサを内蔵しているので、スピードが遅い。
- 出力レベルが、デジタル・インタフェースに適していない。

Q6. 単電源用のオペアンプを使っていますが、出力電圧が 1V 以下になりません。

A6. 単電源では、出力電圧が 1V 以下になりません。0V 付近の出力が必要なら、出力端子を抵抗（2～5kΩ）でプルダウンすると、低い電圧での特性を改善できます。これは、出力トランジスタの飽和電圧によって生ずる特性です。ゼロボルト付近の信号を扱う場合は、プラス・マイナスの 2 電源が必要となります。
マイナス側の電源（単電源ではグランド電位）から、0.6V～1.5V 程度が最低の出力電圧となります。
この電圧以下では、出力インピーダンスが極めて大きな値となりますのでご注意ください。

Q7. 単電源のオペアンプを同相入力電圧で使用すると、出力電圧は 0V になりますか。

A7. いいえ、0V 出力にはならず、電源の 1/2 程度の電圧を出力します。
このため、0V 出力が必要であれば、2 電源用のオペアンプで、出力を±電源のセンタとしてください。

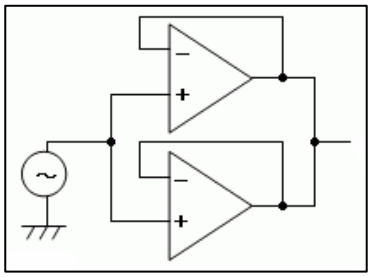
Q8. μPC358 で音声信号の増幅をすると、出力にクロスオーバ歪みが出ます。

A8. はい、そのようになります。クロスオーバ歪みが問題となる場合は、μPC1458 や、オーディオ用の μPC4572、クロスオーバの出ない μPC4742 などをご利用ください。

Q9. 超低ノイズ広帯域デュアルオペアンプ（μPC4570）の推奨電源電圧は±4V～±16V ですが、単電源で使用の場合、電源電圧推奨値は 4V～16V ですか。

A9. いいえ、単電源での電源電圧範囲は 8～32V となります。

Q10. 2 つのオペアンプをボルテージ・フォロワーとし、並列に接続できますか。



A10. 2 つを並列接続すると、安定動作が期待できませんので、このような使用はしないでください。この接続では 2 つの入力電圧が同じでも、オフセット電圧による電位差がボルテージ・フォロワー出力に生じます（これらを 0Ω で結ぶこととなります）。この状態では、電圧が高い方は電圧を高めようとし、低い方は電圧を下げようと電流を吸い込む動作となり、入力電圧が変化しても追従しないので、動作は安定しません。

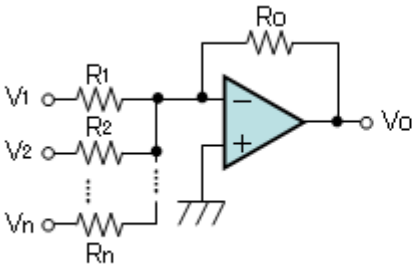
★

Q11. オペアンプを利用した演算回路は、どのような構成ですか。

A11. それぞれ、次のような構成です。

(1) 加算回路

入力電圧の和を出力する回路です(出力電圧の定格範囲内)。



$$Vo = -Ro \times \left[\frac{V1}{R1} + \frac{V2}{R2} + \cdots\cdots + \frac{Vn}{Rn} \right]$$

R1 = R2 = ····· = Rn = R のとき

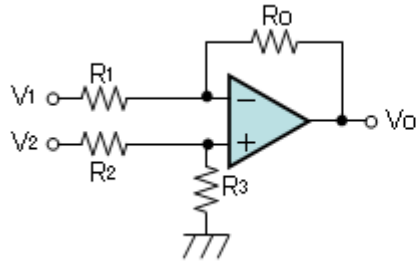
$$Vo = -\frac{Ro}{R}(V1 + V2 + \cdots\cdots + Vn)$$

Ro = R のとき

$$Vo = -(V1 + V2 + \cdots\cdots + Vn)$$

(2) 減算回路(差動増幅回路)

入力電圧の差を出力する回路です。



$$Vo = -\frac{Ro}{R1} V1 + \frac{R3}{R1} \times \frac{R1 + Ro}{R2 + R3} V2$$

R1 = R2、Ro = R3 とすると

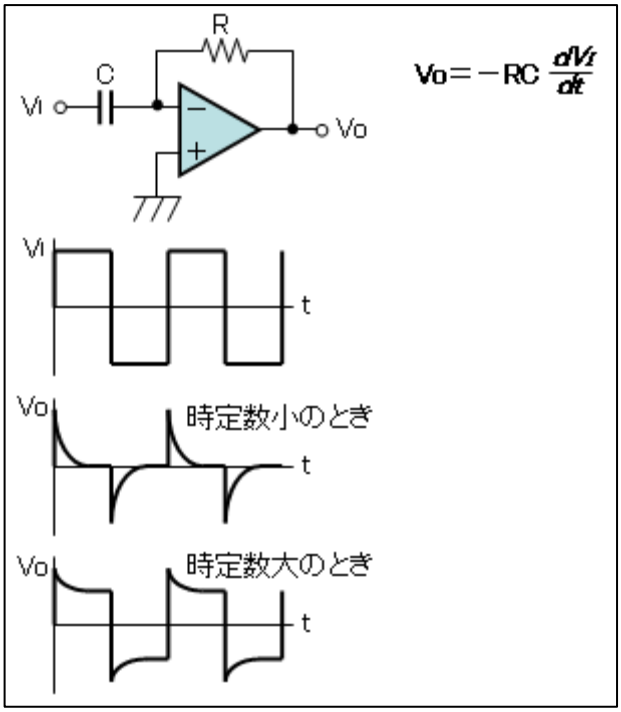
$$Vo = \frac{Ro}{R1}(V2 - V1)$$

Ro = R1 のとき

$$Vo = V2 - V1$$

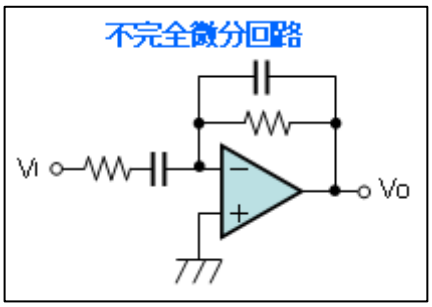
(3) 微分回路

入力電圧の微分値を出力する回路です。



$$Vo = -RC\frac{dVi}{dt}$$

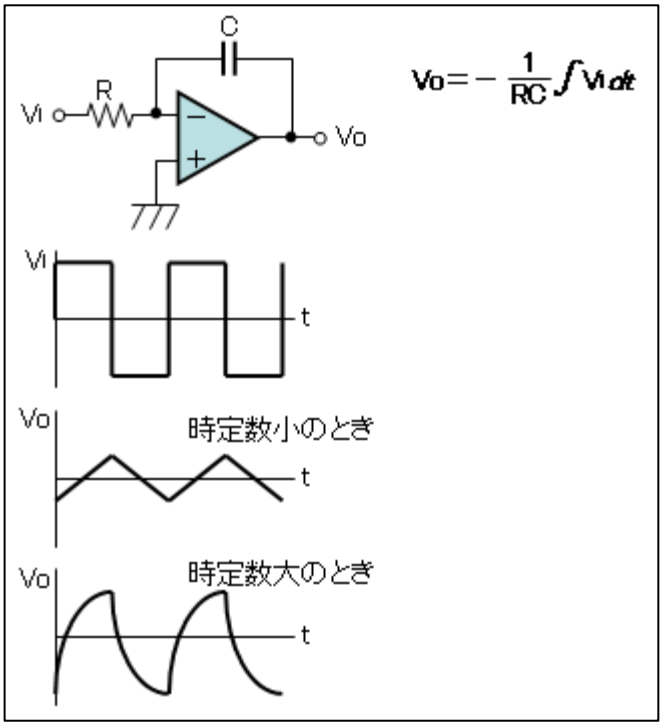
おもにアクティブ・ハイパス・フィルタとして利用されますが、周波数に比例して利得が増加し、高周波ノイズ成分も増幅します。このため、出力の立ち下がりをなだらかにする不完全微分回路がよく用いられます。



不完全微分回路は、高周波成分に対して積分特性を示します。

(4) 積分回路

入力電圧の積分値を出力する回路です。



おもにアクティブ・ロウパス・フィルタとして利用されます。

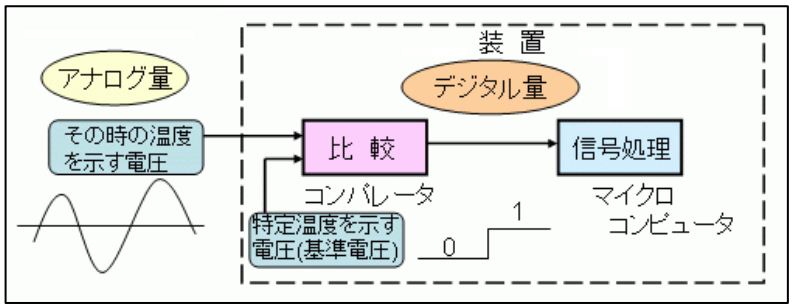
6.1.2 コンパレータ

基　礎

Q1. コンパレータの役割はどのようなことですか。

A1. 我々を取り巻く環境は、風・光・音・気温・湿度など、連続的な変化量（アナログ量）です。一方、我々が使用する装置では、デジタル処理がほとんどです。このアナログ信号と装置内のデジタル回路のインタフェースに、コンパレータ(比較器)が使われます。つまりコンパレータは、基準とする信号と入力信号の比較を行い、その結果に基づいて、出力が「0」か「1」に変化します。

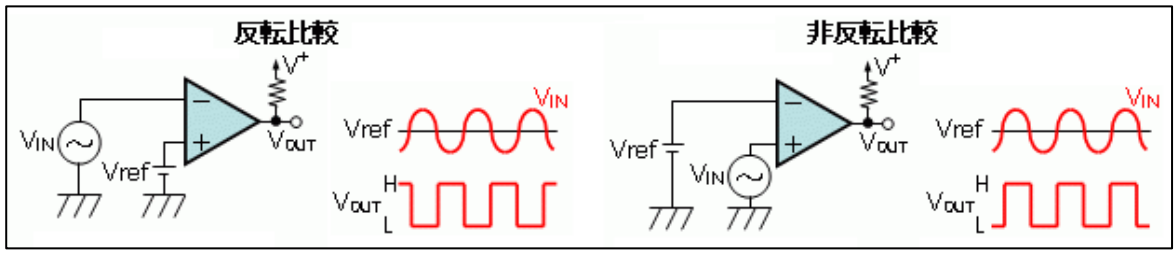
一例として、温度がある値(基準温度)を越えたかどうかの判断を考えましょう。コンパレータは、その時点の温度と基準温度との比較(両温度を電圧値に変換したあと比較)を行い、その結果をデジタル値(例.基準値以下：0、基準値以上：1)として出力します。この 2 値を受け、装置内ではデジタル信号の処理が行われます。



なお、アナログ信号を処理するには、オペアンプが使われます（オペアンプの基礎参照）。

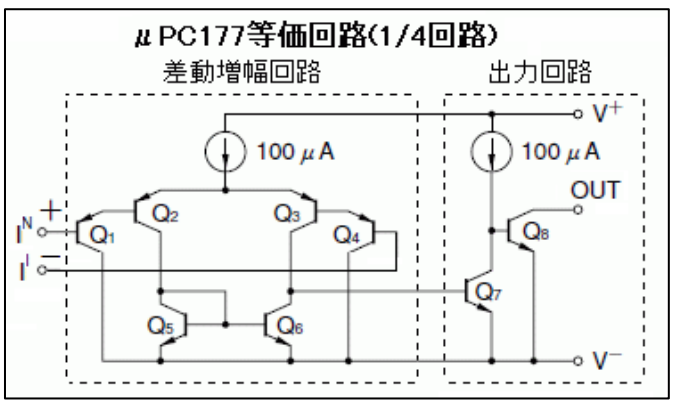
Q2. コンパレータとはなんですか。

A2. Comparator（比較器）のことで、基準電圧（Vref）に対して入力信号（アナログ）の電圧が高いか低いかの比較結果をハイかロウで出力します。 1 ビットの A/D コンバータとも解釈できます。 オペアンプと同様に、5 本の端子があり、三角形で表します。 一般的な使用例は、次のとおりです。



Q3. コンパレータは、どのような機能部分から構成されていますか。

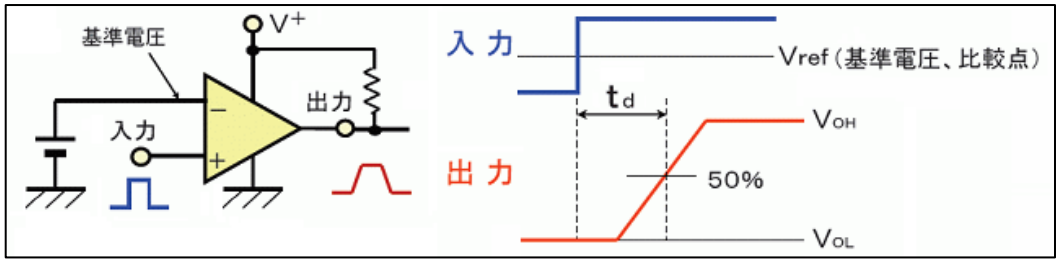
A3. 基本的な回路構成は「差動増幅回路(電圧比較)」と「出力回路」です。「差動増幅回路」では、基準端子に印加された基準電圧と他端子に印加された電圧との比較が行われます。その比較結果が「出力回路」より出力されます。次に、μPC177 の回路構成を示します。



特　　性

Q1. コンパレータの電気的特性はオペアンプとほぼ同様（V_{io}, I_b など）ですが、コンパレータの「応答時間」というのは、なにを規定しているのですか。

A1. コンパレータの入力信号が基準電圧 V_{ref} を越えてから、出力の振幅が 50%になるまでの時間です。



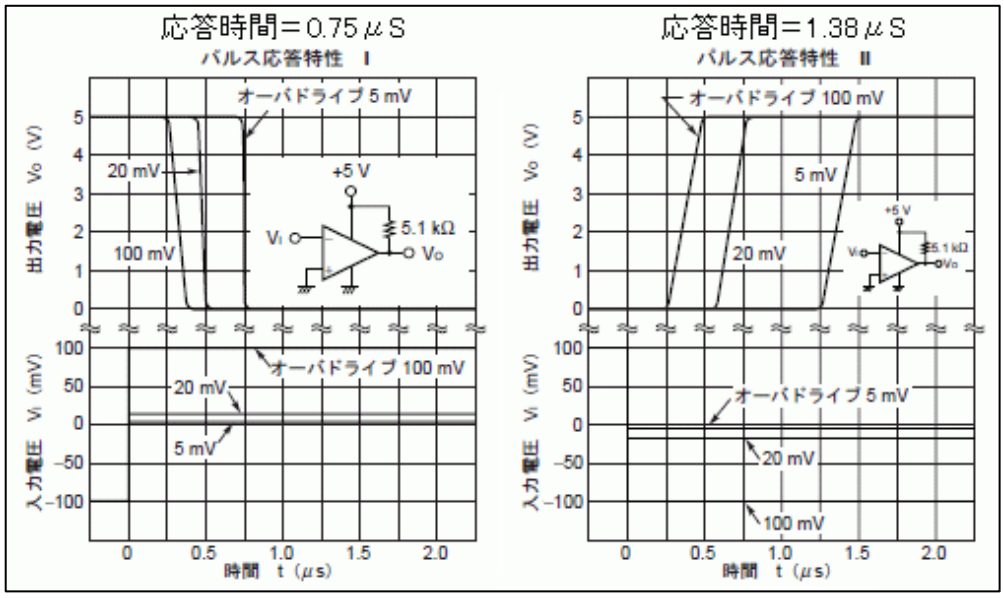
Q2. μPC277 の電気的特性で、応答時間が 1.3μs(TYP.)と規定されています。

μPC277 電気的特性（T_A = 25 ℃, V^+ = +5 V, V^- = GND）

項　目	略 号	条 件	MIN.	TYP.	MAX.	単 位
応答時間 注8		R_L = 5.1 kΩ, V_{RL} = 5 V		1.3		μs

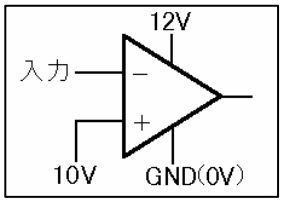
注8. 入力振幅100 mV，オーバドライブ5 mVの時の値です。
オーバドライブ量を増加すれば応答時間を短縮できます。

一方、特性曲線では、パルス応答特性Ⅰ（出力 H⇒L）とパルス応答特性Ⅱ（出力 L⇒H）の時間が異なっています。応答時間 1.3μs の規定をどのように解釈すればいいですか。



A2. パルス応答特性ⅠとⅡでは、パルス応答特性Ⅱの方が時間を要します。パルス応答特性として電気的特性に記載する場合、悪い特性値(応答時間であれば長い方)を提示して、お客様の危険を回避しています。
このため、出力がロウからハイになるパルス応答特性Ⅱのケースで規定をしています。

Q3. μPC177 の同相入力電圧範囲は 0～V^+－1.5V なので、単電源で V^+＝12V なら、10.5V まで入力できると思います。右の回路で基準電圧が 10V の場合、入力電圧（比較する電圧）が 10V 以上になりますが、動作しますか。



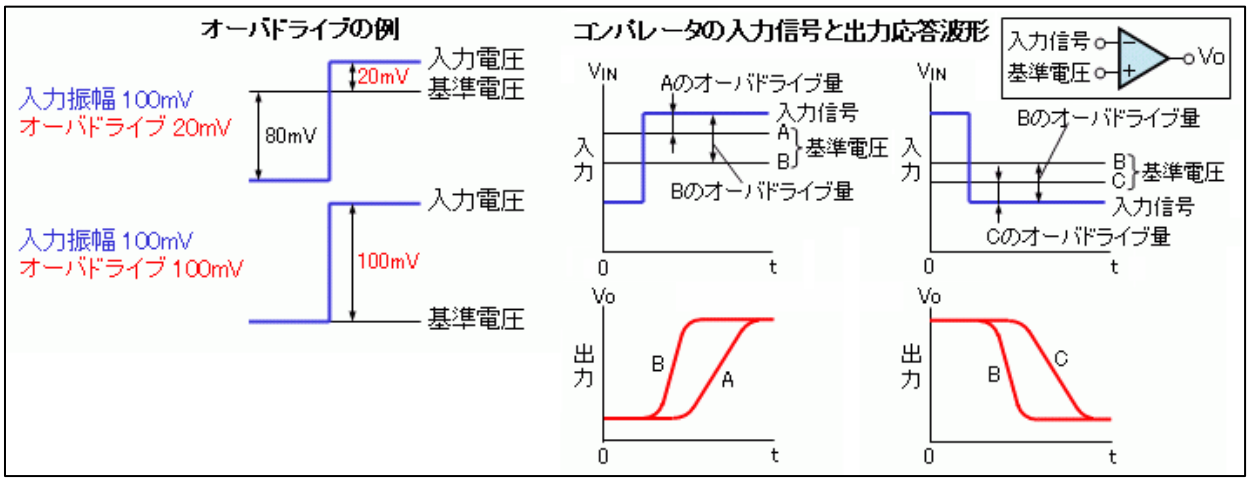
A3. はい、一方の入力（この場合は非反転入力）の端子電圧が「同相入力電圧範囲」なら、正常に動作します。
この場合、もう一方の端子は、0～36V の範囲で正常に動作します。

Q4. μPC272 の一方の入力が 0V で、他方に 9V をかけると、入力に大電流が流れます。壊れているのですか。

A4. μPC272 の差動電圧（VID）は 5V です。このため、入力端子間の電圧差を 5V 以下にしなければなりません。この製品は、高速動作をするため、入力の差動アンプに通常のトランジスタを使っており、差動入力電圧に制限があります。この値を越えると、入力トランジスタの E－B 間が逆耐圧を越えて破壊します。
一度でもこの電圧を越えると、正常に動作しているように見えても、内部では破壊（入力トランジスタの h_{FE} 低下など）が始まっていますので、ご使用にならないでください。

Q5. オーバドライブとは、どのような特性ですか。

A5. 伝達特性の測定条件で、入力電圧が基準電圧を越えて、余分に変化させられる差動入力電圧です。

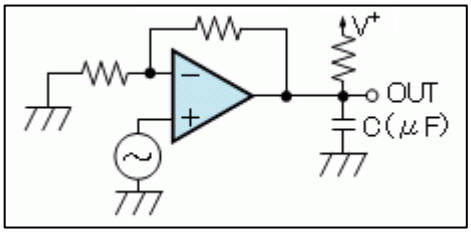


オーバドライブ量によって出力遅延時間が異なるため、設計時には注意が必要です。また特性比較時には、オーバドライブ量の条件をそろえることが重要です。

応　用

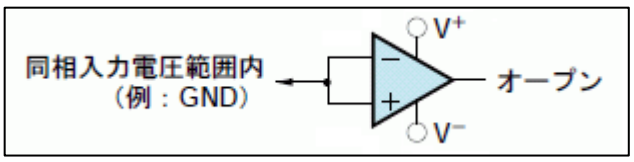
Q1. コンパレータはオペアンプとして使えますか。

A1. 使えますが、位相補正コンデンサを内蔵していないので、負帰還をかけると発振します。このため、大容量のコンデンサを出力に接続する必要があり、スピードが極端に遅くなるので、実用的ではありません。



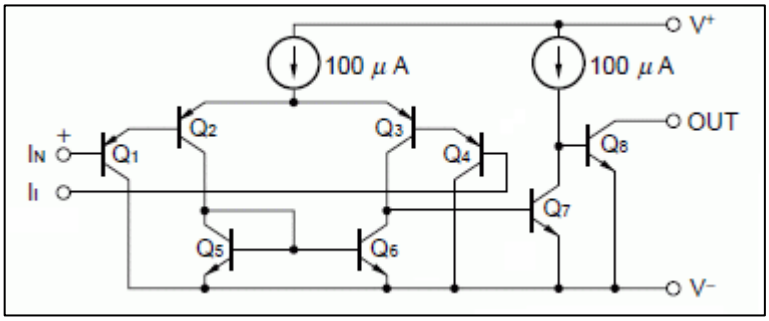
Q2. 複数チャネル内蔵で、使用しないコンパレータは、どのように処理すればいいですか。

A2. 入力端子をオープンのままにしても破壊はしません。しかし、未使用回路の動作は不定なので、安定させるために、入力端子は次のように接続してください。



Q3. μPC393 の 1 チャネルを使用せず、その空き端子は推奨処理（5V 単電源で、－入力と＋入力を GND へ接続し、出力はオープン）とします。この場合、コンパレータ内部はどのような動作となりますか。

A3. 入力端子を短絡すると、入力差動増幅回路のオフセット状態（－入力と＋入力につながるトランジスタの V_{BE} 差による増幅電圧を出力）によって、出力トランジスタ Q_8 が ON か OFF の状態となります。
なお、出力がオープンなので、空きチャネルの出力電流が、他チャネルに影響することはありません。

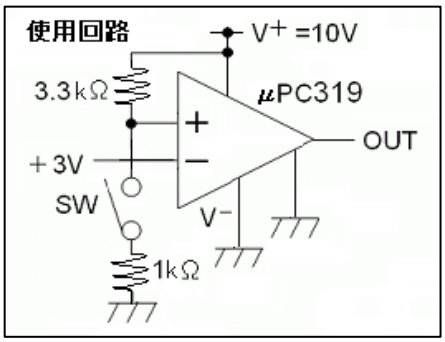


関連 FAQ：オペアンプ>特性>オフセット電圧とはなんですか

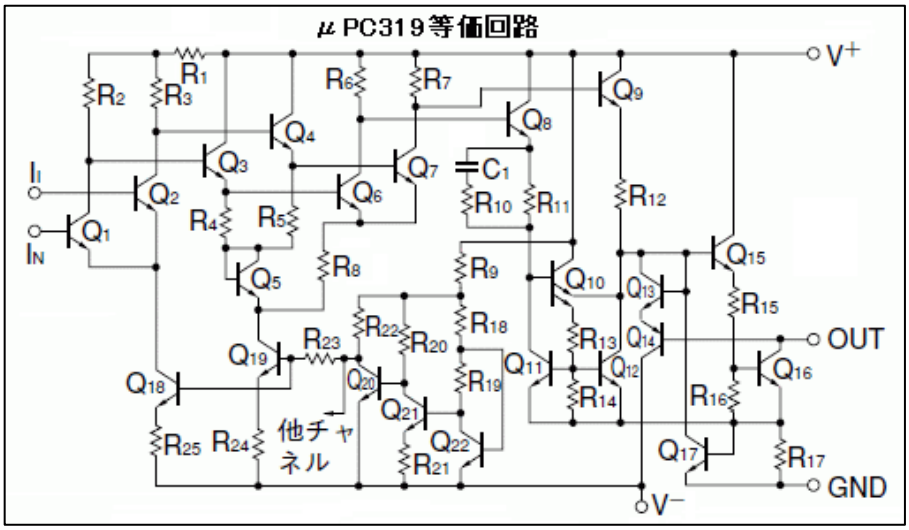
Q4. μPC393 を V^+＝15V、V^-＝GND と設定し、＋24V へ接続された抵抗性負荷を直接駆動します。
このとき、μPC393 の出力端子（オープンコレクタ）–GND 間に印加できる、最大電圧値はどれだけですか。

A4. オープン・コレクタタイプの出力耐圧（Vo）は、絶対最大定格で V^-－0.3V～V^-＋36V と規定しています。
V^-が GND に接続されているので、印加できる最大電圧値は＋36V です。

Q5. スイッチの ON/OFF で比較動作をして、OFF のとき、＋入力に約 1mA の電流が流れ込むのは正常ですか。



A5. 正常ではありません。入力差動段 Q1、Q2 の V_{EB} 耐圧を越えているために起っている現象と思われます。
μPC319 の±入力端子間の絶対最大定格は、|±5V|＝5V です。

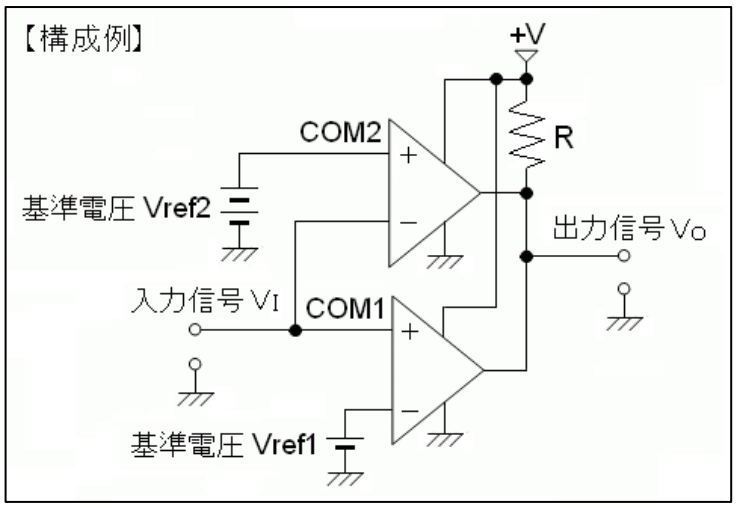


対策として、3.3KΩ を V^+＝10V より低い電圧（たとえば＋5V）にプルアップしてください。

Q6. ウィンドウ・コンパレータとは、どういうものですか。

A6. ウィンドウ・コンパレータは 2 つの基準電圧を持ち、入力信号の電圧がこの間にあるときに、出力が変化します(次の構成例では H レベル)。

以下に、出力がオープン・コレクタ・タイプの単電源コンパレータを用いた例で、その動作を説明します。

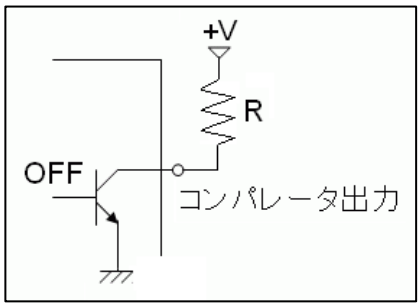


【動　作】

入力信号V_Iの変化に伴う出力信号 V_O の変化を次表に示します。

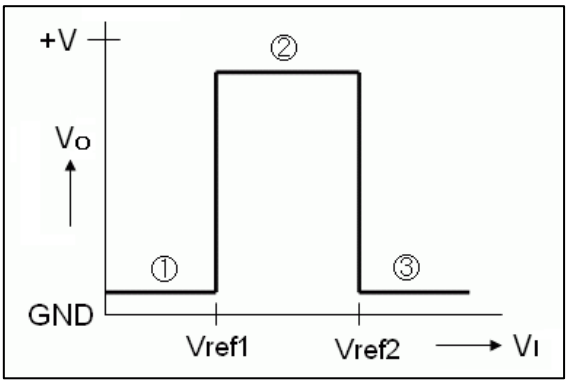
入力信号(VI)	COM1出力 COM2出力	出力信号(VO)注2	状態
VI＜Vref1	L H注1	L	①
Vref1＜VI＜Vref2	H注1 H注1	H	②
Vref2＜VI	H注1 L	L	③

注 1. コンパレータの出力がオープン・コレクタなので、出力トランジスタが OFF のときは、次図のように、コンパレータ出力は外付け抵抗 R に引っ張られて H レベルとなります。



2. 構成例の回路では、コンパレータの 2 出力がワイヤード OR になっています。これにより、両出力が H レベルのときに V_O が H レベルとなり、いずれかの出力が L レベルのときに V_O が L レベルとなることを応用しています。

入力信号V_Iと出力信号V_Oとの関係は、次図のとおりです。



Q7. μPC393 のデータ・シートに掲載の「応用回路 2（ヒステリシス付き）」は、どのような動作をするのですか。

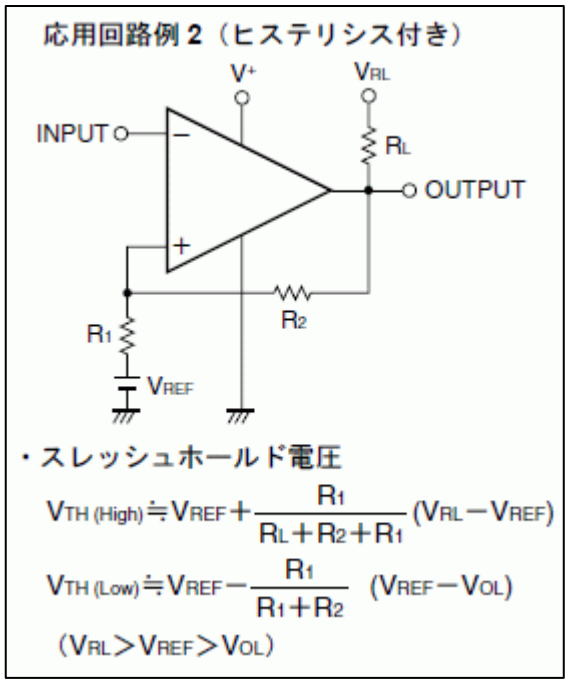
A7. この回路は、入力信号に多少のノイズが含まれていても、出力に影響しないことを特徴としています。現在の入力条件だけでなく、過去の入力条件も出力に影響することをヒステリシスと呼びます。

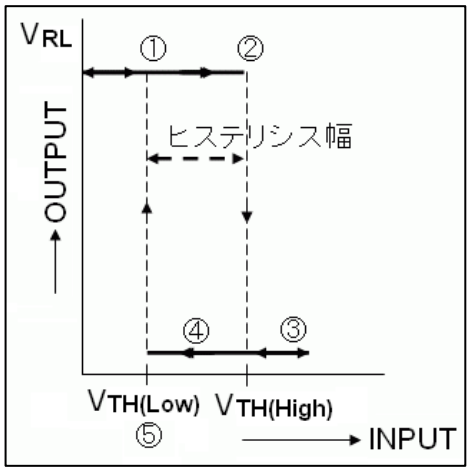
【動　作】

OUTPUT と V_{REF}の電位差を R_2、R_1で分割して＋入力端子に正帰還することで、しきい値電圧にヒステリシスを与えることができます。これによって、ヒステリシス幅以内の電圧のノイズを除去できます。

具体的な動作では、INPUT が十分に低い状態から電圧が上昇(①)した場合、 OUTPUT はハイ・レベルを出力するため、 R_1には正の電圧降下（R_1 の「V_{REF} につながる端子」より「＋入力につながる端子」の方が電位が高い）が生じ、＋入力の電圧は $V_{TH(High)}$となり、INPUT がこの電圧に達するまでこの状態を保ちます。INPUT が $V_{TH(High)}$ を越えた瞬間(②)、OUTPUT はロウ・レベルとなり、同時に R_1の電圧降下は負（R_1 の「V_{REF} につながる端子」より「＋入力につながる端子」の方が電位が低い）となり、＋入力の電圧は $V_{TH\ (Low)}$となります。

このあと、INPUT を上昇させても、OUTPUT はロウ・レベルのままです(③)。次に、 INPUT の電圧を徐々に下げると、 $V_{TH\ (Low)}$になるまで OUTPUT はロウ・レベルを保ちます(④)が、$V_{TH\ (Low)}$以下になる(⑤)と、再び OUTPUT はハイ・レベルに反転して、最初の状態に戻ります。



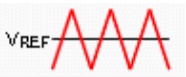
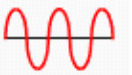


6.1.3 相違点と共通事項

相違点

Q1. オペアンプとコンパレータは、どう違うのですか。

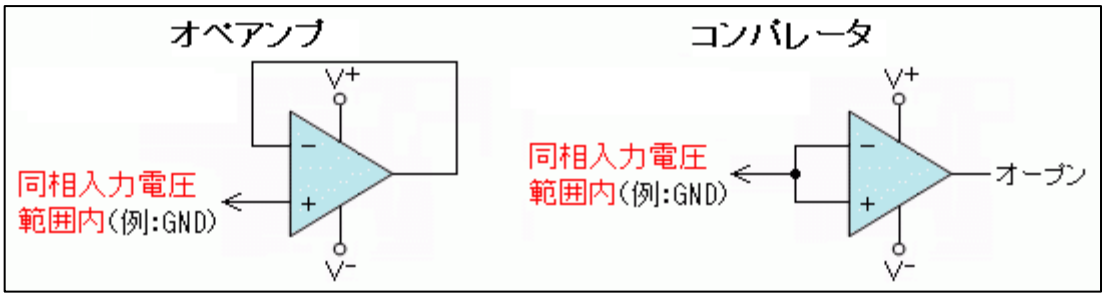
A1. 基本構成は同様ですが、次のような相違があります。

	オペアンプ	コンパレータ
基本動作	増幅	比較
入力信号	アナログ 	アナログ 
出力信号	アナログ 	デジタル 
出力段の構成	プッシュプル	オープン・コレクタ／ドレーン
特徴	発振しにくい	増幅動作では発振する
位相補正コンデンサ	内蔵(負帰還で発振させないため)	内蔵せず(負帰還で使用せず)
パルス応答特性比較	遅い(スルー・レートで定義)	速い(50%出力遅延で定義)

Q2. オペアンプとコンパレータでは、使用しない端子の処理方法が異なるのはなぜですか。

A2. 使用しないオペンプやコンパレータの動作を安定させ、使用ブロックの動作に影響を及ぼさないようにする目的は、両者とも同じです。回路の違いから、その安定化方法が異なります。

オペンプでは、負帰還で安定化が可能なボルテージ・フォロワ形式を推奨しています。コンパレータでは、入力電圧の固定で出力の安定化が図れるので、入力電圧の固定化を推奨しています。



関連 FAQ　オペアンプ　：複数チャネル内蔵で、使用しないオペアンプの処理

コンパレータ：複数チャネル内蔵で、使用しないコンパレータの処理

特　　性

Q1. オペアンプ／コンパレータの「一般用」と「通信工業用」の違いはなんですか。

A1. 「一般用」は民生機器など、「通信工業用」は産業機器などでの使用をそれぞれ推奨しています。「通信工業用」は、搭載セットが厳しい環境下で使用されると考えられますので、動作周囲温度を「一般用」より拡張しています。さらに「通信工業用」は、検査項目によっては全数検査を行っています。

例　単電源デュアルオペアンプ

一般用(μPC358G2)　　：動作周囲温度＝−20～+80℃

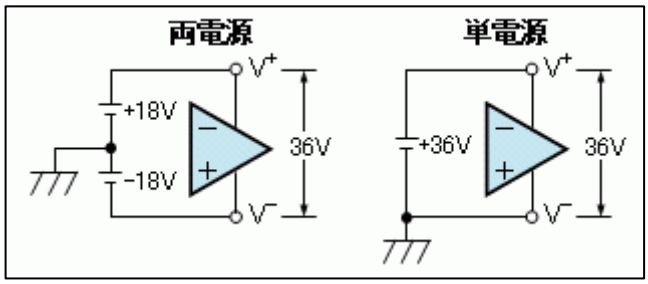
通信工業用(μPC1251G2)：動作周囲温度＝−40～+85℃

Q2. 定格電源電圧（$V^+ - V^-$）とは、どのような特性ですか。

A2. V^+端子と V^-端子の間に印加できる電源電圧の最大値です。たとえば、定格電源電圧が最大 36V の場合、次の電圧まで印加できます。

2 電源で使用時　：±18V（非対称でもよい）

単電源で使用時　：＋36V



オペアンプには GND 端子がないので、この場合は、2 電源／単電源いずれでも、V^+と V^-の端子間の電圧は 36V です。ただし、電源の逆接続は、破壊を起こす可能性があります。

Q3. 定格入力電圧とは、どのような特性ですか。

A3. 特性劣化や破壊を起こさずに、入力端子に印加できる電圧範囲です。

電源 ON/OFF 時などの過渡状態も含めて、定格を越えないようにする必要があります。

また、オペアンプやコンパレータが正常動作する入力電圧は、電気的特性の同相入力範囲内です。

なお、通常のオペアンプやコンパレータの定格は、$V^- - 0.3V \sim V^+ + 0.3V$ ですが、単電源の場合には、電源 OFF 時にも入力電圧の正電圧印加が可能です。個別データ・シートで、絶対最大定格をご参照ください。

Q4. 全損失（P_T）とは、どのような特性ですか。

A4. 常温（T_A=25℃）で正常動作を保証しながら、IC が消費できる最大電力です。全損失は、チップのジャンクション温度の最大値（通常は保存温度 T_{stg} の上限と同一）と、パッケージの熱抵抗で決まります。動作周囲温度が高くなると、全損失が低下するので、P_T–T_A 特性（ディレーティング・カーブ）を参考にご検討ください。

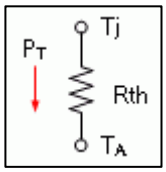
Q5. 熱抵抗とは、どのような特性ですか。

A5. 熱の伝わりにくさのことで、単位電力あたりの、パッケージの上昇温度で表します（熱設計参照）。規定がない場合、チップのジャンクション（PN 接合部）温度（Tj）は保存温度の最大値です（オペアンプやコンパレータでは 125℃）。ジャンクションからパッケージ周囲までの熱抵抗 Rth（j-A）は、データ・シートの絶対最大定格の値から、次の例のように求められます。

8ピン SOP の場合（T_A=25 で P_T=440 mW、Tj=125℃）

$$Rth = \frac{Tj - T_A}{P_T} = \frac{125 - 25}{0.44} \fallingdotseq 227\ (℃/W)$$

なお、熱伝導を電気の等価回路に置き換えると、次の図のようになります。

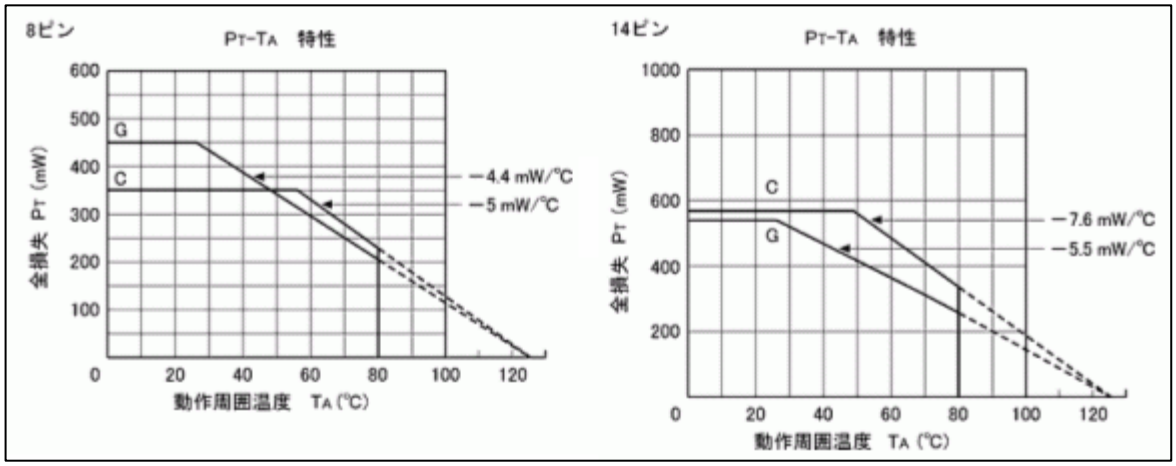


Q6. ディレーティング・カーブとは、どのような特性ですか。

A6. 周囲温度の上昇に従い、消費できる電力の制限が必要で、この熱設計をディレーティングと呼びます。

ディレーティング・カーブとは、「全損失（P_T）−動作周囲温度（T_A）」特性曲線のことです。

8ピン SOP で T_A=＋80℃の例（T_A=25 で P_T=440mW, Rth（j−A）=227℃/W）：P_T=440−（80−25）/0.227=198mW

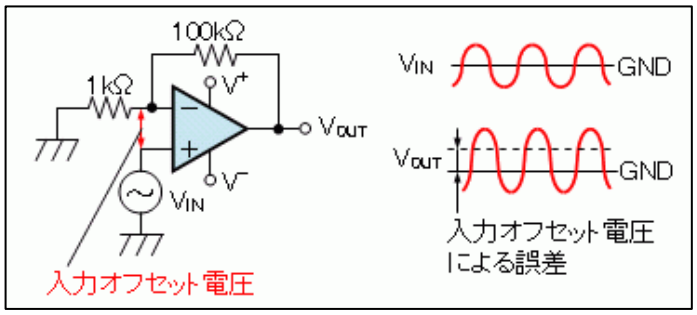


備考　ディレーティング・カーブの斜線を延長すると、P_T＝0 でジャンクション温度の最大値（オペアンプやコンパレータでは 125℃）となります。

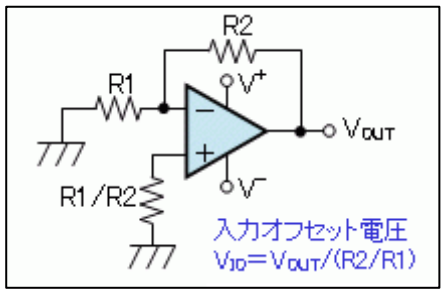
Q7. 入力オフセット電圧（V_{IO}）とは、どのような特性ですか。

A7. オペアンプでは、負帰還をかけたときの、＋入力と−入力の端子間の電圧差です。またコンパレータでは、比較電圧（スレッシュホールド電圧）の誤差となります。0V が理想ですが、オペアンプでは入力段差動増幅回路の＋入力と−入力のトランジスタの V_{BE} に差があり、これが入力オフセット電圧となります。

実使用では、たとえば入力オフセット電圧 $V_{IO} \leqq \pm 5$mV のオペアンプで、100 倍のアンプを作った場合、出力に最大**±5×100＝±500（mV）**の誤差が出ます。このため、微小入力信号を扱う回路では、入力オフセット電圧の小さい製品を選ぶか、外部でオフセット調整を行う必要があります。



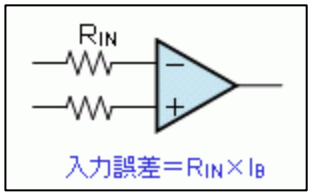
測定方法：上記回路で V_{IN} を 0 として、出力を 100 で割れば測定できます。このため測定回路は、次の図のようになります。



備考　2 端子間の電位差なので、最大値は±の値になります。ただし、一般の参考文献などでは、±符号を省略して記載している場合もあります。

Q8. 入力バイアス電流（I_B）とは、どのような特性ですか。

A8. 入力端子に流れ込むか、流れ出す電流のことです。入力端子は、トランジスタのベース（FET ではゲート）となっています。この入力トランジスタが動作するには、ベース電流（I_B）が流れる必要があり、これが入力バイアス電流です。実使用では、入力に高抵抗（R_{IN}）を入れた場合に、入力信号の誤差となって表れます。



たとえば、R_{IN}=1MΩ，I_B=100nA では、0.1V の誤差となります。このため、信号源のインピーダンスが大きいときには、入力バイアス電流の小さい製品を選ぶ必要があります。また、＋入力と－入力のバイアス電流の差が入力オフセット電流です。流れ込みか、流れ出しかは、入力段のトランジスタによって異なります。

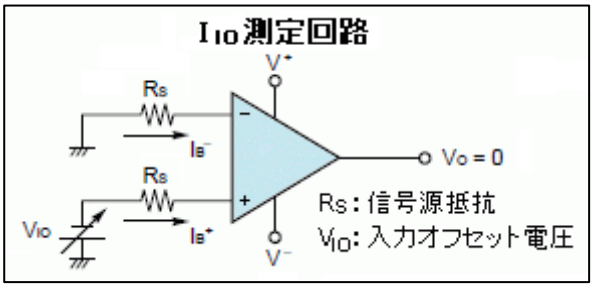
入力段Tr	バイアス電流の方向	バイアス電流のオーダ
PNP	流れ出し（μPC358, 324, 4558, 4570, 393, 339など）	nA
NPN	流れ込み（μPC741, 1458など）	nA
Pch JFET	流れ込み（μPC406x, 407x, 811～814など）	pA 注

注　Pch JFET の入力のバイアス電流は、ゲート・リーク電流に相当するため、pA 単位と小さいのが特長ですが、温度が 10℃上がるとバイアス電流が 2 倍になります。

Q9. 入力オフセット電流(I_{IO})とは、どのような特性ですか。

A9. 差動入力トランジスタのペア特性（h_{FE},V_{BE}）が一致しないことから、オペアンプの出力電圧を 0V としても、それぞれの入力端子に流入または流出する入力電流に差が生じます。この差が入力オフセット電流（I_{IO}）です。

$$I_{IO} = | I_B^+ - I_B^- |$$



この異なる電流が、オペアンプの入力に接続される信号源抵抗（R_S）を流れることで生じる電圧降下は、オフセット電圧を増加させる一因となります。

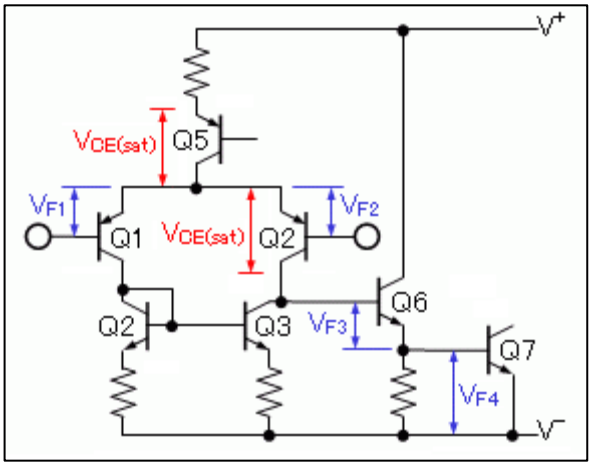
なお、入力オフセット電流が入力オフセット電圧と同じく、出力値への誤差として影響する事例をインフォメーション「オペアンプの用語と特性」の「2.3　入力誤差項の影響」に示していますのでご参照ください。

Q10. 同相入力電圧範囲とは、どのような特性ですか。

A10. オペアンプやコンパレータが、正常動作をする入力電圧の範囲です。同相入力電圧範囲を越えると、出力が V^+か V^-に飽和して、正常動作ができなくなります。μPC4558 の例を紹介します。

同相入力電圧範囲の考え方

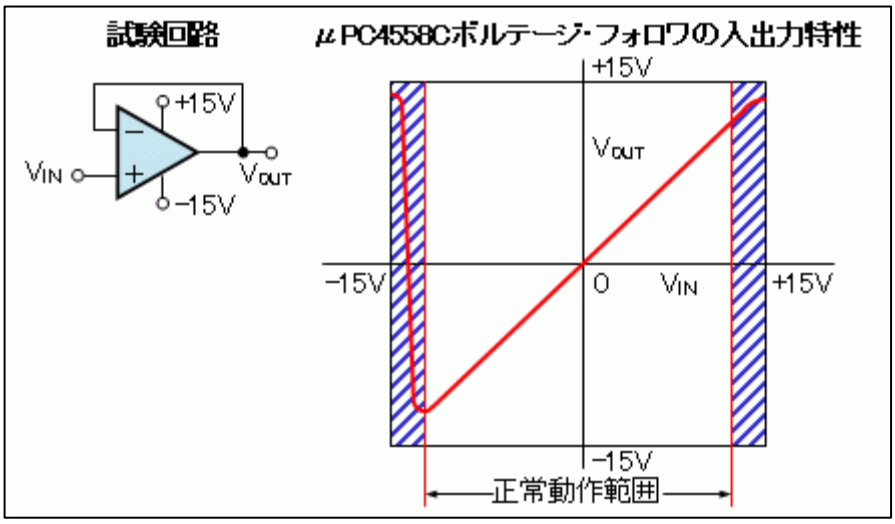
トランジスタが動作するには、V_{BE}（約 0.7V）が必要です。



$V^+ - V_{ICM(+)} = V_{F1} + V_{CE(sat)} \leqq 1V$ （T_A=25℃）

$V^- - V_{ICM(-)} = V_{F4} + V_{F3} + V_{CE(sat)} - V_{F2} \leqq 1V$ （T_A=25℃）

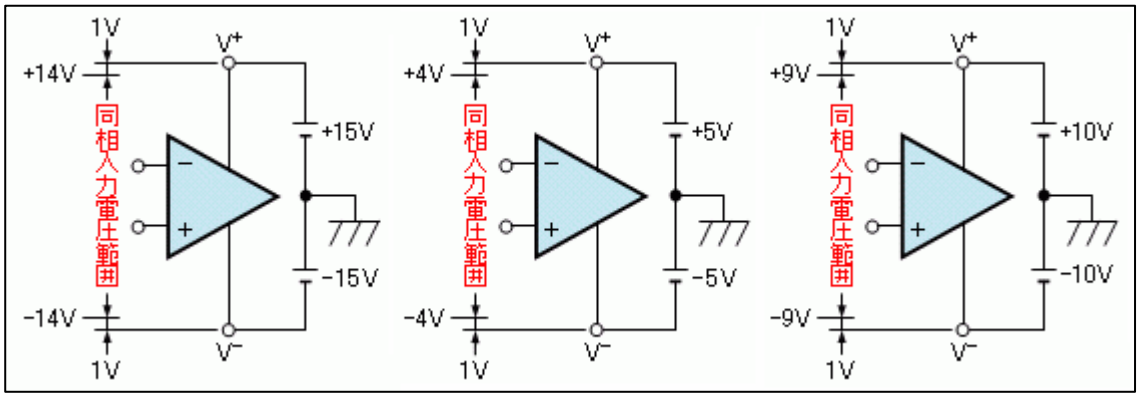
(1) 同相入力電圧範囲を越えた場合



斜線部分が同相入力電圧範囲を越えた範囲で、μPC4558 では出力が V^+へ飽和します。

(2) 電源電圧が変化した場合

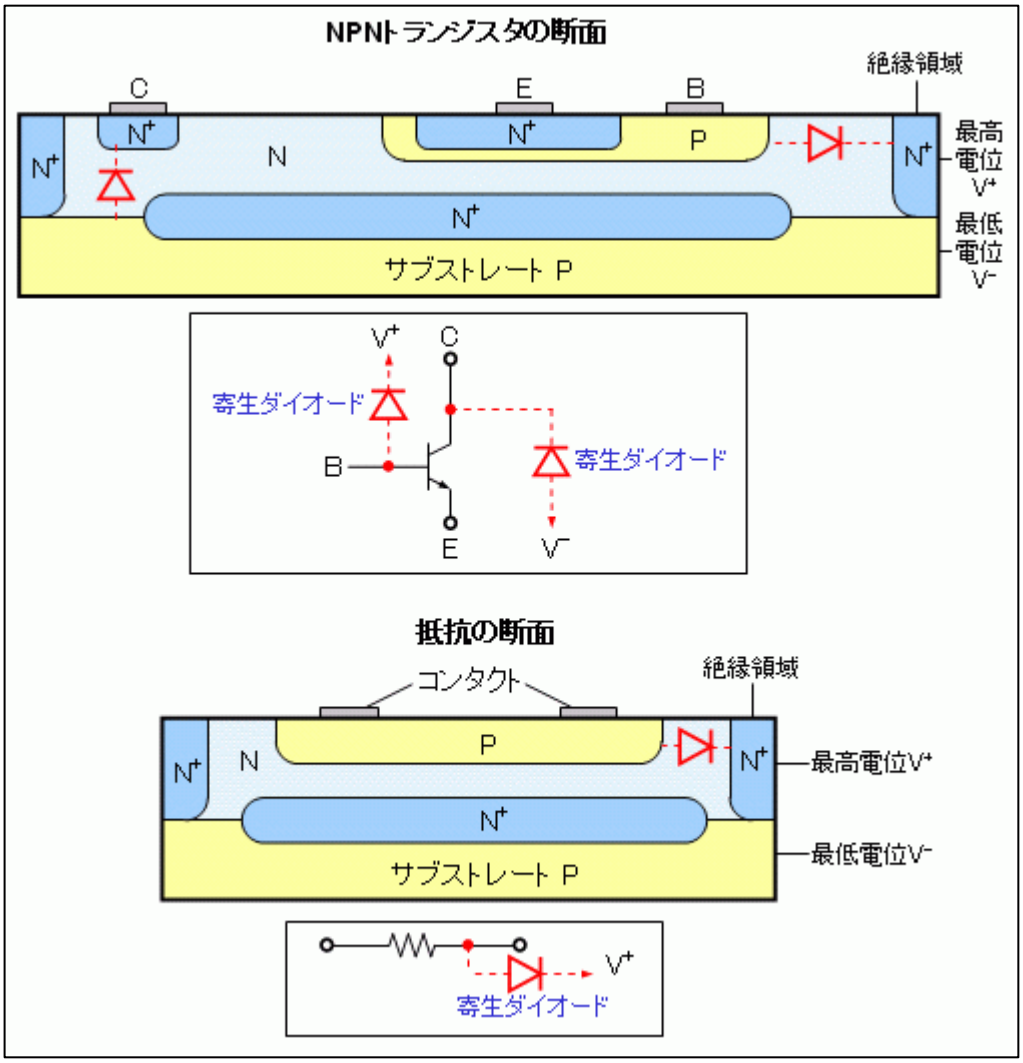
データ・シートでは、$V^{\pm}$=±15V での同相入力電圧範囲が TYP.±14V となっていますが、V^-＋1V～V^+－1V の範囲のことなので、電源電圧を下げれば同相入力電圧範囲も狭くなります。



Q11. 寄生ダイオードとは、どういうものですか。

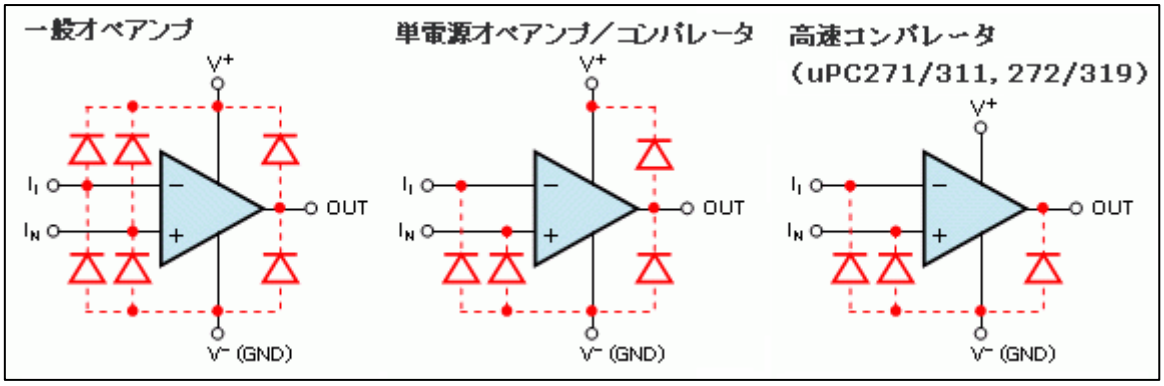
A11. 等価回路には表しませんが、IC の構造上、PN 接合で構成されるダイオードです。

バイポーラ IC 内部の、NPN トランジスタと抵抗の断面構造の例を次に示します。



サブストレート（P 層）は電源の最低電位 V^-に、絶縁領域（N 層）は最高電位 V^+に、それぞれ接続されます。このため、各素子の電位が V^-と V^+の範囲を越えると、寄生ダイオードに電流が流れ、誤動作や特性劣化、破壊の原因となります。スパイク・ノイズなどに対しては、寄生ダイオードに逆電流が流れて保護ダイオードの働きをしますが、特性は保証していませんので、必ず絶対最大定格の範囲でご使用ください。

オペアンプとコンパレータの内部では、次のように寄生ダイオードが構成されます。



Q13. 「標準品」と「DC項目選別品：オーダ名称の末尾に(5)を付加」は、なにが違うのですか。

A13. DC項目選別品は、入力オフセット電圧、入力バイアス電流などの特性で、標準品より厳しい基準で選別した製品です。詳細は各製品のデータ・シートで、電気的特性を比較してください。

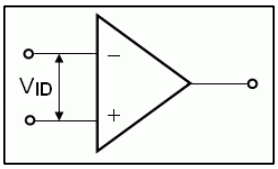
注意 (5)に対する内容は、当社のオペアンプ／コンパレータに適用します。他製品では、別内容で定めていますので、個別にご確認ください。

Q14. 基板に応力が加わると、特性は変動しますか。

A14. はい、特に入力オフセット電圧が変動します。半導体の外部から応力が加わると、ピエゾ効果で半導体の比抵抗が変化します。このため、基板に応力が加わると、実装している IC チップにも応力が加わり、オペアンプやコンパレータの入力部の差動増幅回路の特性が変動します。変動量は、基板の大きさや実装条件、応力の加わり方で異なるため、IC 単体での評価基準はなく、実機での十分な評価が必要です。

Q15. 差動入力電圧（V_{ID}）とは、どのような特性ですか。

A15. 入力 2 端子間に印加できる最大電圧です。どちらの端子を基準にするかで、極性が反対になるので、絶対値の最大を±て規定します。これを越えると、初段トランジスタの劣化や破損の原因となります。



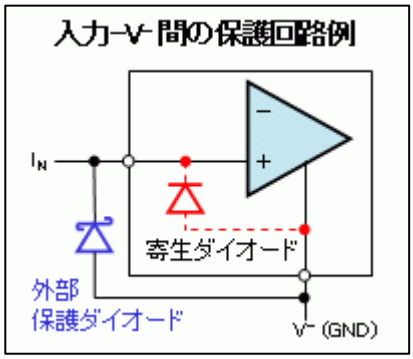
絶対最大定格（TA = 25°C）

項　目	略　号	μPC1251GR-9LG, μPC1251GR(5)-9LG	μPC1251MP-KAA, μPC1251MP(5)-KAA	μPC358GR-9LG, μPC358GR(5)-9LG	単　位
電源電圧 注1	V+−V−	−0.3～+32			V
差動入力電圧	VID	±32			V

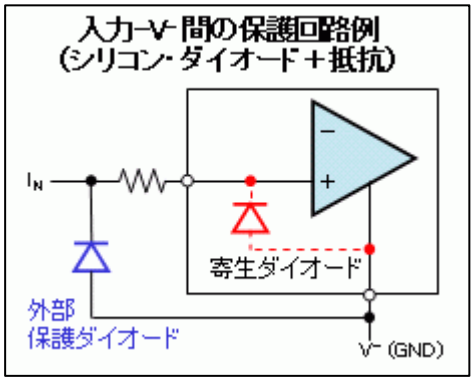
回路設計

Q1. 過電圧入力に対する保護回路は、どのように設計すればいいのですか。

A1. 入力電圧が V+より高くなる場合と、V−より低くなる場合を考慮して、入力端子と V+、V−間をそれぞれ順方向電圧の小さいショットキィ・バリア・ダイオードなどでクランプしてください。寄生ダイオードの順方向電圧は、通常のシリコン・スイッチング・ダイオードと同等の約 0.6V です。この寄生ダイオードに電流が流れないように、順方向電圧の小さいダイオードで保護する必要があります。



通常のシリコン・スイッチング・ダイオードを外付け保護ダイオードとして使用する場合には、入力にシリーズ抵抗を挿入して、寄生ダイオードに流れる電流を約 5mA 以下に制限すれば、破壊は防止できます。ただし、誤動作の可能性がありますので、ショットキィ・バリア・ダイオードなどの順方向電圧の小さいダイオードの使用を推奨します。



パッケージ

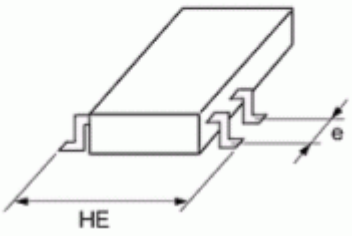
Q1. SOP のパッケージ記号 G と G2 の違いはなんですか。

A1. 外形寸法が異なります。

JEITA（技術情報技術産業協会：旧 EIAJ）の標準外形に準拠するために、1983 年（昭和 58 年）に外形寸法を変更し、区分のためにパッケージ記号を G から G2 に変更しました。なお、1984 年 4 月以降に生産している製品は、すべて G2 です。

外形寸法の相違点は、次のとおりです。

	G	G2
リード取り付け幅（HE）	6.0mm	6.5mm
リード・ピッチ（e）	1.25mm	1.27mm



なお、G，G2 の区分は、オペアンプとコンパレータだけに適用しています。

Q2. マガジン・ケースとテーピング品の包装仕様を教えてください。

A2. 包装をご参照ください。

Q3. オペアンプ、コンパレータのばら品は、どのような包装ですか。

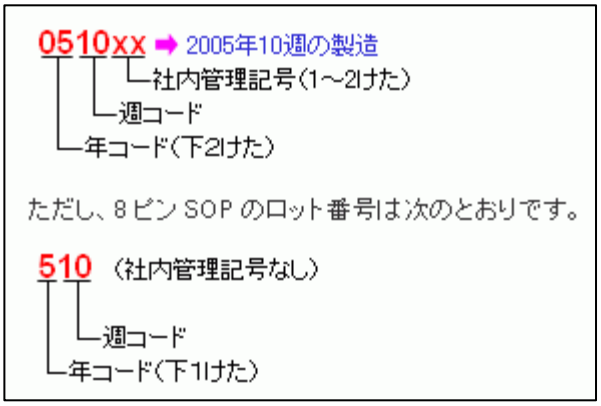
A3. パッケージが SOP か DIP の製品は、マガジン・ケースでの包装をしています。

TSSOP の製品は、バラ品での提供はしておらず、テープ包装だけです。

なお、マガジン・ケースとテープについては、包装をご参照ください。

Q4. ロット番号の意味を教えてください。

A4. 製造時期と社内管理記号です。



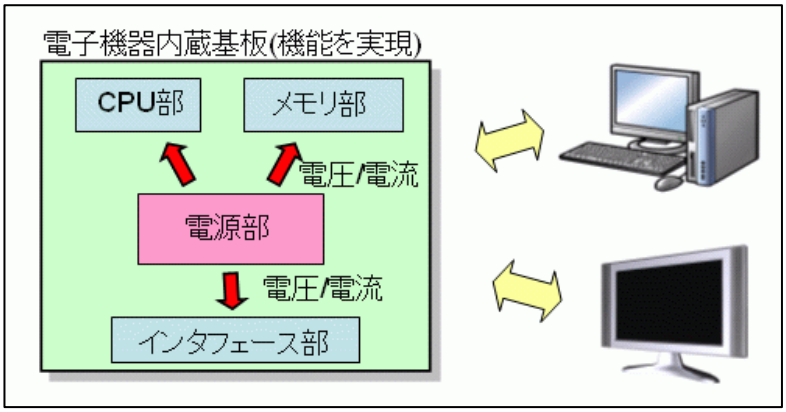
6.2 電源用 IC

6.2.1 電源用 IC の種類

電源とは

Q1. 電源とはなんですか。

A1. 我々の身近にあるパソコン、AV 機器などには、その電子機器の機能を実現させるため、多くの電子部品が搭載されています。電源とは、これら電子部品を動作させるのに必要な電圧/電流を供給する機能部です。



さらに、その機能は次の2つに大別できます。

■交流から直流への変換

■直流から直流への変換

(1) 交流から直流への変換

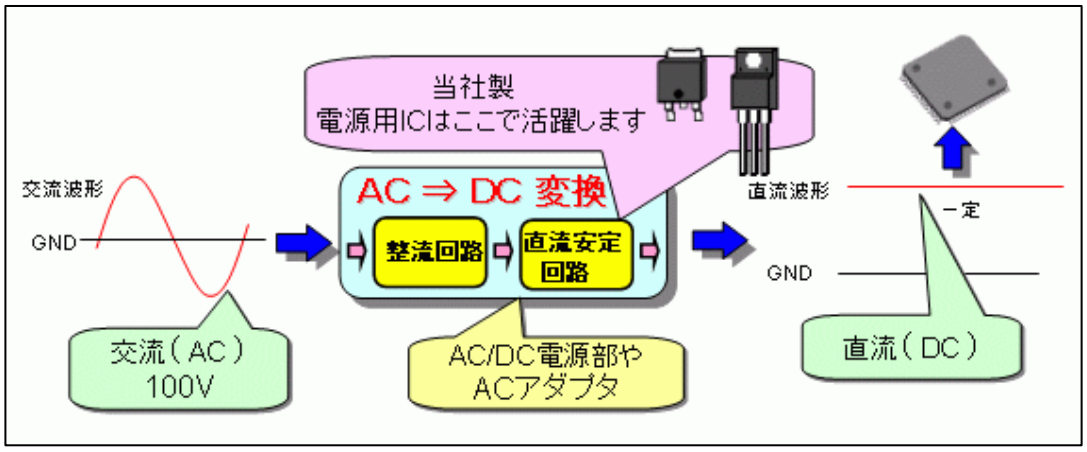
[変換の必要性]

この電源部に対し、電気は大元の発電所からオフィス／家庭のコンセントに供給されます。このコンセントに供給される電気は交流 100V です。一方、電子機器を構成する電子部品の大半は直流で低電圧(例:3.3V、1.8V)で動作します。つまり、電子機器を動作させるには、コンセントからの交流を電子部品用の直流に変換する必要があります。

なお、交流/直流に関しては、FAQ の製品共通の情報－基礎知識－電気「電気に関する単位」をご参照ください。

[役割]

電源部は、コンセントの交流電圧を電子部品が必要とする、直流電圧に変換する役割を担います。



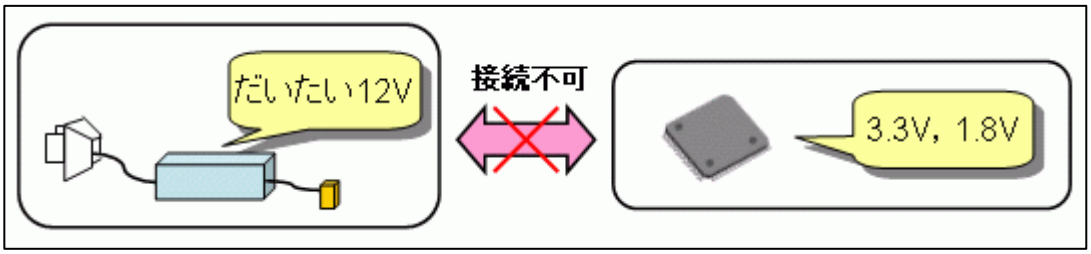
(2) 直流から直流への変換

[変換の必要性]

電子機器の小型化や省エネ化の要求から、電源部に対してコンパクト化、発熱の低減が強く求められています。このような要求を満たす方法に、スイッチング電源があります。

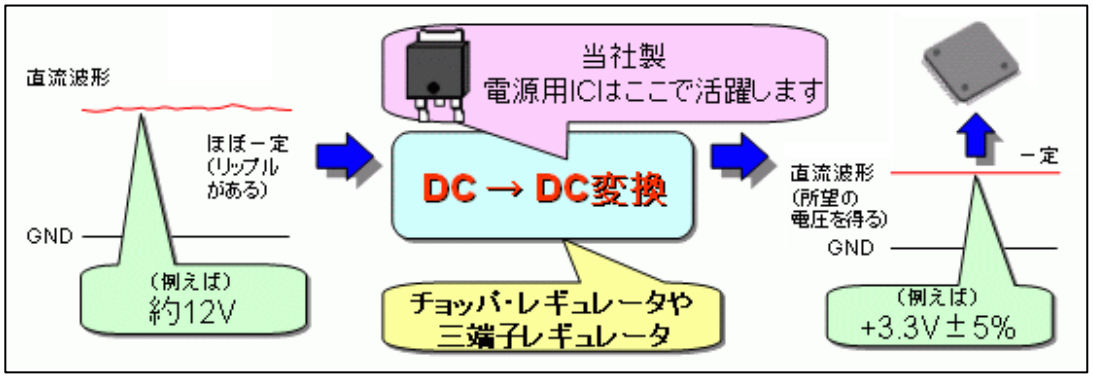
他にバッテリや AC アダプタなどから供給される電気は、ある特定の直流電圧(例:12V)だけで、しかも不安定な供給状況が見られます。一方、電子機器を構成する電子部品は、異なる電圧(例:3.3V, 1.8V)と精度、電流を必要とし、電子部品が要求する電源仕様を満足しません。したがって、ここでも変換が必要となります。

なお、スイッチング・レギュレータに関しては、FAQ の「方式の違い」をご参照ください。



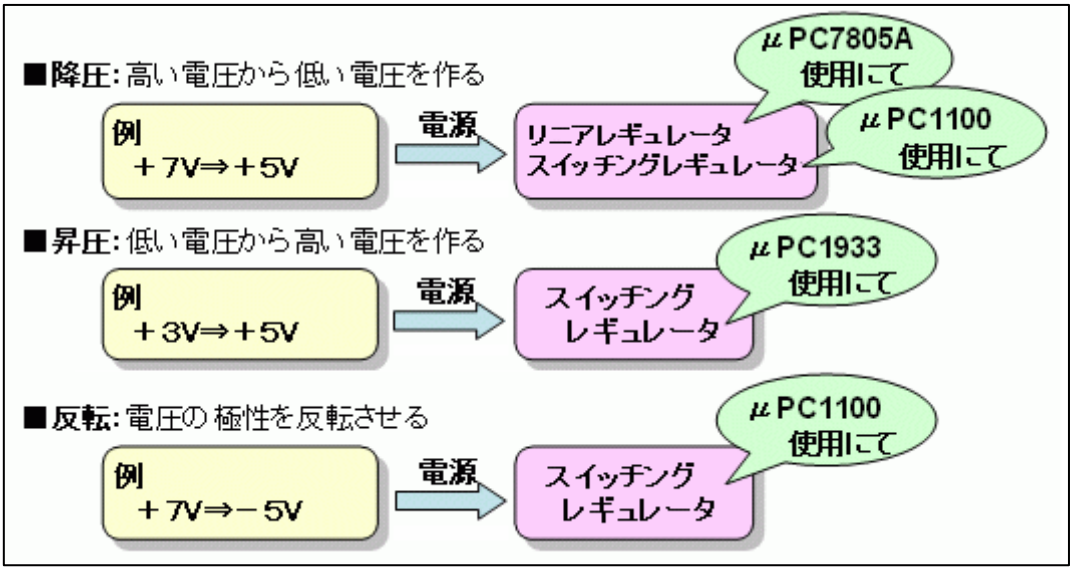
[役割]

スイッチング動作による発熱の低減、さらには電源部はバッテリや AC アダプタから供給される不安定な電圧を電子部品が必要とする安定した電圧へと変換する役割を担います。



Q2. 電圧変換には、どのような種類がありますか。

A2. 電圧変換には 3 種類あります。



関連 FAQ 「方式の違い」「基本構成例」「基礎」

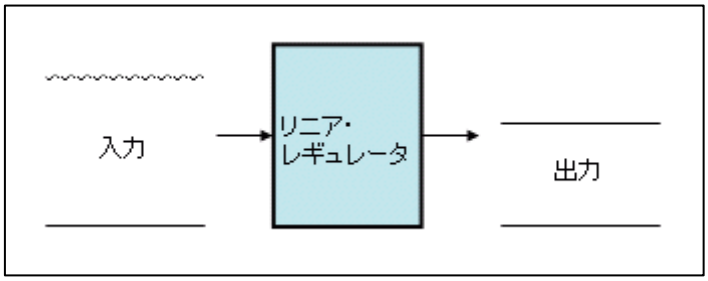
方式の違い

Q1. 電源用 IC は、種類によってどのように違うのですか。

A1. 電源は、方式によって次のように分類され、各用途に応用する IC があります。

[リニア・レギュレータ]

機能：入力－出力間の電力を連続的に制御し、不安定な入力電圧を電圧降下させて、安定した出力電圧を取り出します。

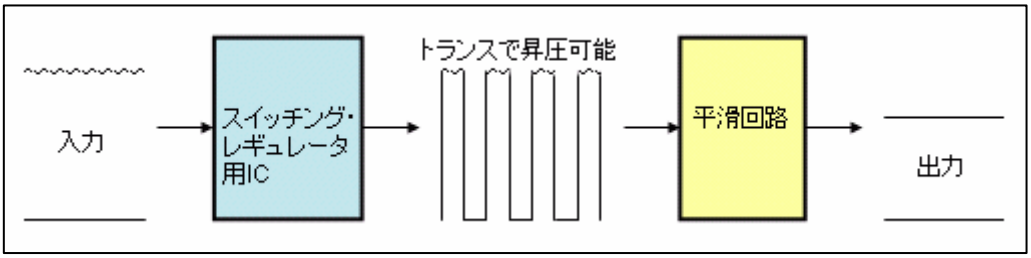


種類：シリーズ・レギュレータ(三端子レギュレータなど)、シャント・レギュレータがあります。入出力の電位差が 1V 程度でも動作するシリーズ・レギュレータを低飽和型(LDO タイプ)と呼びます。

特徴：電源がほとんど 1 チップ化されシンプルです。

[スイッチング・レギュレータ]

機能：入力を ON/OFF 制御して、平滑回路で一定電圧を出力します（PWM 制御）。



種類：絶縁型（AC－DC コンバータ）、非絶縁型（DC－DC コンバータ）があります。

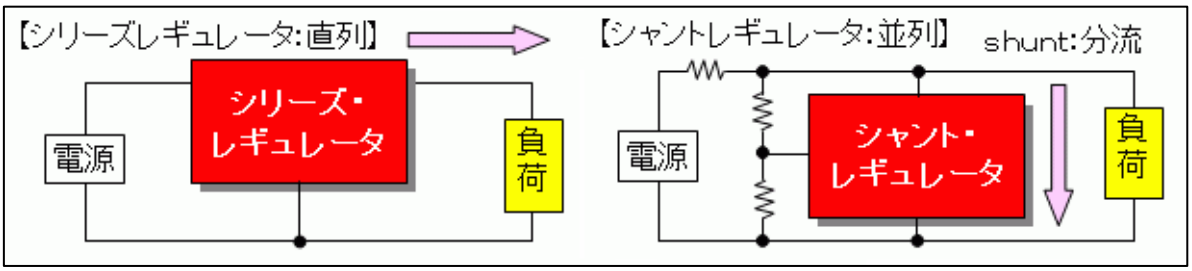
用途：昇圧/降圧/反転の AC－DC コンバータ、DC－DC コンバータに利用されます。

特徴：発熱が少なく高効率で、大電力電源に対応可能です。

シリーズ・レギュレータとシャント・レギュレータの違い

Q1. シリーズ・レギュレータとシャント・レギュレータの大きな違いはなんですか。

A1. 負荷との接続方法が異なります。負荷に対して、シリーズ・レギュレータは直列(シリーズ)、シャント・レギュレータは並列に接続することで、それぞれ出力電圧の安定を図ります。



シリーズ・レギュレータの種類

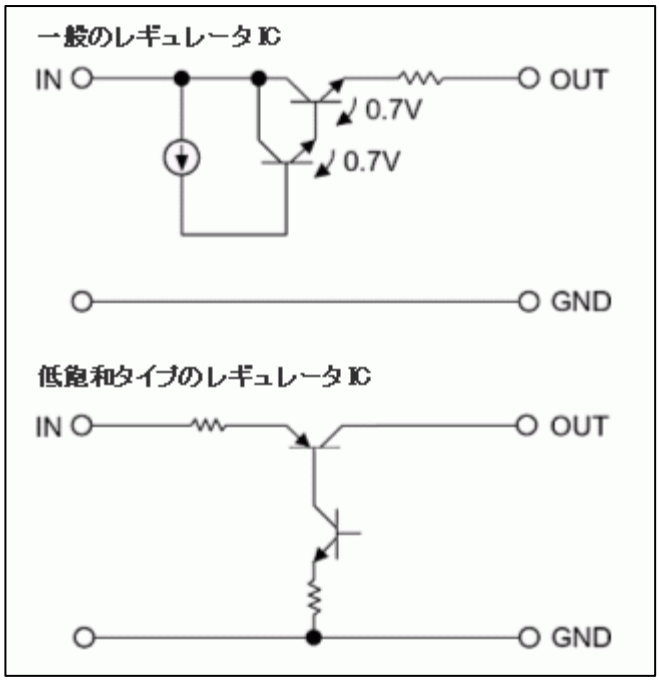
Q1. 三端子レギュレータとはなんですか。

A1. シリーズ・レギュレータの基本形で、端子が入力、GND、出力の 3 種類なので、一般にこのように呼ばれています。

Q2. 低飽和三端子レギュレータとはなんですか。

A2. 一般の三端子レギュレータ（μPC78L00 シリーズなど）は、出力段に NPN 型のトランジスタをダーリントン接続しています。このため、動作に必要となる入出力間電圧差は、ダーリントン接続された出力段トランジスタのベース−エミッタ間電圧（0.7V×2＝1.4V）より低くできません。

低飽和タイプのレギュレータでは、出力段のトランジスタに PNP トランジスタを使用しており、小さな入出力間電圧差で使用できます。



三端子レギュレータの損失 Pd は、次式で表されます。

$Pd \fallingdotseq (V_{IN} - V_O) \cdot I_O$

$(V_{IN} - V_O)$：入出力間電圧差

I_O：出力電流

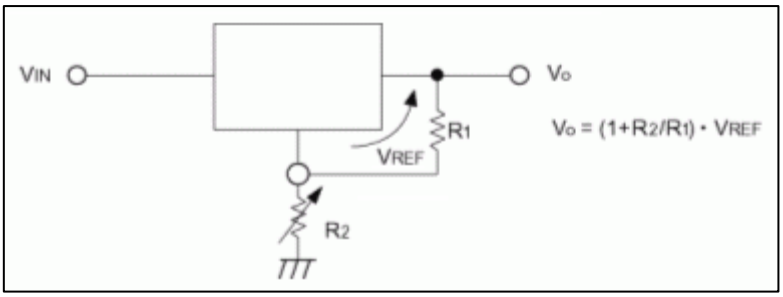
入出力間電圧差 $V_{IN}-V_O$ が小さいほど、損失（発熱）が小さくなります。

注意 異常発振を防止するため、低飽和三端子レギュレータの出力には、大容量のコンデンサが必要です（47μF 以上など）。

このため、起動時などの低入力時に、比較的大きな回路動作電流が流れます。出力電圧の立ち上がりに、このピークが出てしまいますので、GND に抵抗を挿入して、出力電圧をシフトアップするアプリケーションは推奨できません。

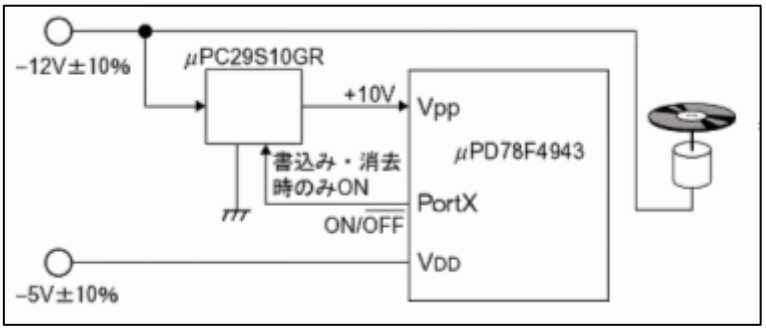
Q3. 可変出力三端子レギュレータとはなんですか。

A3. 外部に付加する抵抗の値によって、出力電圧を変化させることができるレギュレータです。固定出力の三端子レギュレータにない電圧が必要なときに使用します。出力電圧の微調整もできるので、高精度な電圧が必要なときにも利用できます。



Q4. ON/OFF 機能付きシリーズ・レギュレータとはなんですか。

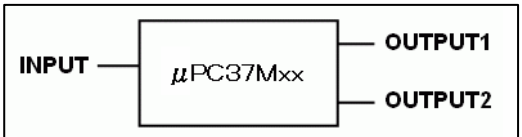
A4. 三端子レギュレータに制御信号を追加して、外部のロジック信号で、電圧を出力させたり、停止させたりする機能を持ったシリーズ・レギュレータです。



Q5. 2 出力機能付きシリーズレギュレータとはなんですか。

A5. 1 入力電圧から、2系統の電圧を出力するシリーズ・レギュレータです(1パッケージ化)。

μPC37Mxx シリーズの出力は、OUTPUT1(3.3V か 2.5V)＝1A、OUTPUT2(1.8V か 2.5V)＝0.5A です。



	OUTPUT1	OUTPUT2
μPC37M21	2.5V	1.8V
μPC37M31	3.3V	1.8V
μPC37M32	3.3V	2.5V

Q6. シリーズ・レギュレータの品名には、μPC78L05 のように、真ん中にアルファベットのつくものがありますが、どのように違うのですか。

A6. この記号は、およそ次のような出力電流特性を表します。詳細は、ホームページで、シリーズ・レギュレータの一覧表をご参照ください。

L：0.1A

N：0.3A

M：0.5A

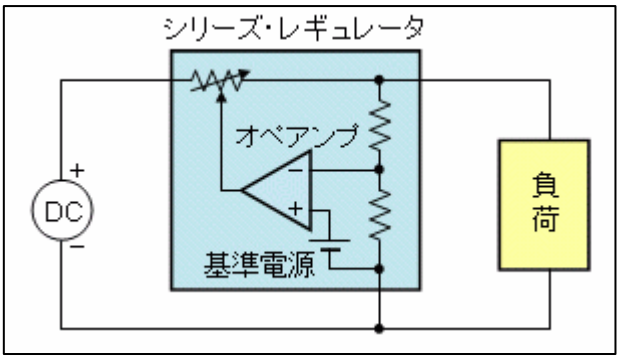
(記号なし)：1.0A

6.2.2 電源の構成

基本構成例

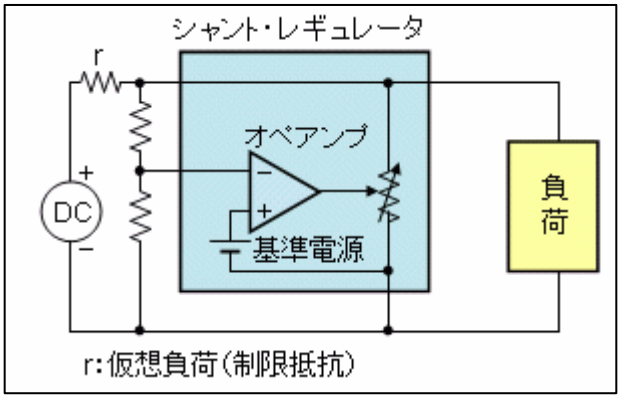
Q1. シリーズ・レギュレータはどのように使うのですか。

A1. シリーズ・レギュレータは、負荷に対して直列（シリーズ）に接続します。
出力電流や入力電圧の変動をオペアンプで検出して、負荷と直列の可変抵抗を制御することで、出力電圧を一定に保ちます。



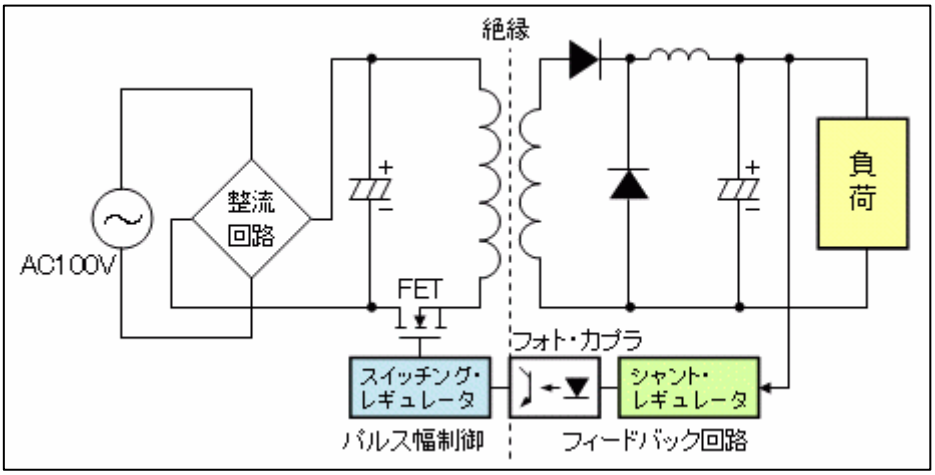
Q2. シャント・レギュレータはどのように使うのですか。

A2. シャント・レギュレータは負荷に対して並列に接続します。このため、電流は分流（shunt）となります。
出力電流や入力電圧の変動をオペアンプで検出して、負荷と並列の可変抵抗を制御することで、出力電圧を一定に保ちます。



Q3. スイッチング・レギュレータで、AC－DC コンバータをどのように構成するのですか。

A3. パワーMOS FET をスイッチング制御して、できたパルスをトランスで変圧し、平滑回路で安定電圧を得ます。

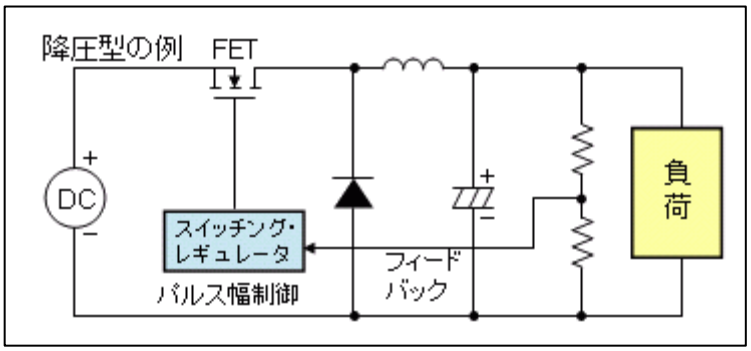


Q4. スイッチング・レギュレータで、DC－DC コンバータをどのように構成するのですか。

A4. DC-DC コンバータには、チョッパ型とチャージ・ポンプ型があります。

[チョッパ型]

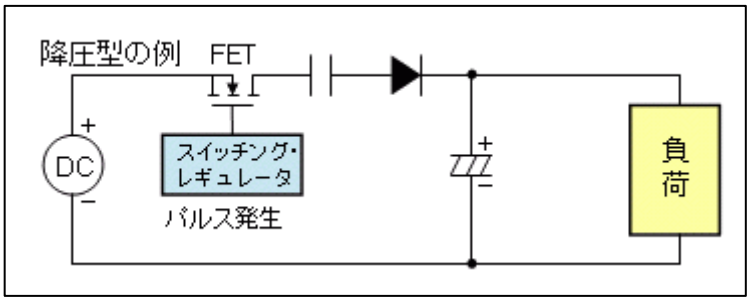
入力電圧をスイッチング制御して、平滑回路で安定電圧を得ます。



[チャージ・ポンプ型]

入力電圧をスイッチング制御して、コンデンサを充放電しながら平滑回路で安定電圧を得ます。

この方式では、フィードバックをしません。



6.2.3 三端子レギュレータ

用　語

出力電圧

三端子レギュレータの特性のうち、もっとも重要な特性です。接合部温度、入力電圧、出力電流（負荷電流）を条件として、上限と下限が規定されています（これが出力電圧の精度です）。

設計時は、これらの条件を充分確認してください。なお、ルネサス エレクトロニクスの三端子レギュレータでは、品名が出力電圧を表しています。

例 μPC2405AHF： μPC2400AHF シリーズで出力電圧が 5V のもの。

参考資料：三端子レギュレータの使い方

入力安定度

接合部温度（通常 T_j＝125℃）、出力電流の条件を一定としたとき、入力電圧をある範囲で変化させた場合の、出力電圧の変化を最大値で示します。この値が小さいほど、IC の出力電圧の安定度が高いことになります。

参考資料：三端子レギュレータの使い方

負荷安定度(出力安定度)

接合部温度(通常 T_j=125℃)、入力電圧の条件を一定としたとき、出力電流(負荷電流)をある範囲で変化させた場合の、出力電圧の変化を最大値で示します。この値が小さいほど、IC の出力電圧の安定度が高くなります。

参考資料：三端子レギュレータの使い方

回路動作電流

IC が動作するのに必要な電流です。入力端子から GND 端子に流れる電流として規定します。

リップル除去率

入力電圧に含まれる交流成分(AC 成分)が、出力電圧で減衰する比率を dB で示します。この値が大きいほど、入力電圧の変化が出力に現れにくく、よい性能といえます。

リップル除去率＝20log(ΔV$_{IN}$/ΔV$_O$)[dB]

リップル除去率は、交流成分の周波数に依存します。ルネサス エレクトロニクスは、通常 120Hz で規定します。

出力雑音電圧

IC が動作すると、内部で雑音(ノイズ)が発生します。出力雑音電圧は、IC 内部で発生した雑音が、出力に現れる電圧を実行値で示します。この値が小さいほど、オーディオ用や微小電圧を扱う回路の電源に適します。

最小入出力間電圧差

安定な出力電圧を得るために、IC に最低限必要な入力電圧と出力電圧との電圧差です。

入力電圧と出力電圧の差と、出力電流の積が、三端子レギュレータの損失になります。この値が小さいほど、低損失で使用できます。

参考資料：三端子レギュレータの使い方

全損失

三端子レギュレータの全損失 P_d は、入出力間の損失と回路の損失の合計で、次式で表されます。

$P_d=(V_{IN}-V_O)\cdot I_O+V_{IN}\cdot(I_{bias}+\Delta I_{bias1})$

($V_{IN}-V_O$)：入出力間電圧差

I_O：出力電流

I_{bias}：回路動作電流

ΔI_{bias1}：回路電流変化量

ピーク出力電流

三端子レギュレータは、過電流制限回路を内蔵しています。ピーク出力電流とは、過電流制限回路が動作を開始する出力電流値です。この値を越える電流を流そうとすると、出力電圧が低下します。

また通常、安全動作領域制限回路と組み合わされていて、入出力電圧差が大きい(入力電圧が高い)ときや、チップ温度が高いときには、より少ない電流で制限がかかるようになっています。たとえば、1A 出力の μPC2900 シリーズの場合でも、入力電圧が高かったり、チップ温度が高かったりすると、1A 未満でも制限回路が動作して、出力電圧が低下することがあります。

参考資料：三端子レギュレータの使い方

出力短絡電流

事故などで、出力端子がグランドと短絡(ショート)状態になったとき、出力端子から流出する電流です。

この値は、過電流制限回路の特性で決まります。つまり、入力電圧が高くて、チップ温度が高いほど、電流を制限します。

注意 信頼性を保証する項目ではないので、連続的に出力を短絡させないでください。連続的な出力短絡に対する保証が必要な場合は、外部回路で対応してください。

出力電圧温度変化

出力電圧は、チップ温度によって変化します。出力電圧温度変化は、チップ温度が 0～125℃の範囲での、出力電圧の変化量です。この値は小さいほどよいのですが、保証値ではないので、出力電圧の温度変化を要求する用途には通常使えません。温度変化を保証しなければならない用途には、次の製品を推奨します。

μPC1060：2.5V 基準電圧源

μPC1093：可変出力シャント・レギュレータ

μPC1943, 1944, 1945：低電圧対応シャント・レギュレータ

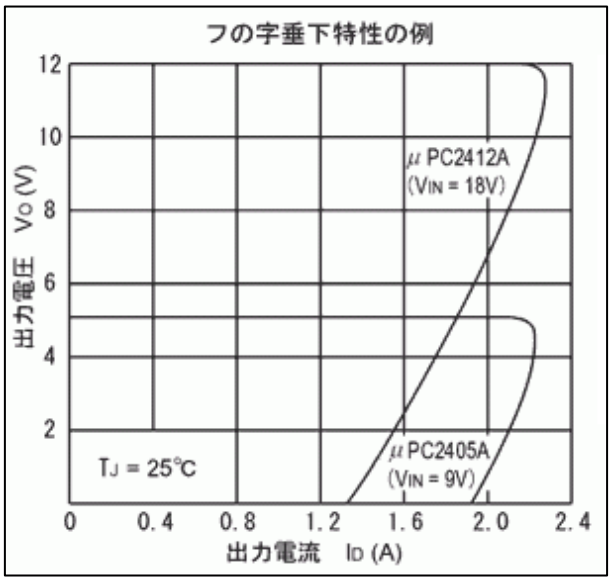
起動時回路動作電流（起動時突入電流）

μPC2400 シリーズなどの低飽和型レギュレータは、出力段トランジスタに PNP 型トランジスタを使用しています。このため、入力電圧が低いときにこのトランジスタが飽和して、入力から GND に大電流が流れます。起動時回路動作電流は、この電流の最大値です。この値を越える電流が流れないと、レギュレータは正常に動作しません。

フの字垂下（フの字負荷）特性

出力電流を徐々に増加させたときの、出力電圧の変化を負荷特性で表します。IC 内部の過電流制限回路が動作して、出力電圧が低下する直前の出力電流をピーク出力電流（I_{Opeak}）といいます。また、過電流制限回路の作用で、出力電圧が 0V になるときの出力電流（I_{Oshort}）を出力短絡電流といいます。

ピーク出力電流が出力短絡電流より大きいと、カタカナの「フ」の字形の「フの字垂下特性（フの字負荷特性）」となり、等しければ、逆「L」字形の「定電流垂下特性（垂下特性）」となります。



基　礎

Q1. 正電圧から負電圧が得られますか（たとえば「＋5V→－12V」）。

A1. いいえできません。三端子レギュレータは、正電圧入力では正電圧の出力しかできません。

Q2. 負電圧から正電圧が得られますか（たとえば「－12V→＋5V」）。

A2. いいえできません。三端子レギュレータは、負電圧入力では負電圧の出力しかできません。

Q3. 低い電圧から高い電圧が得られますか（たとえば「＋3V→＋5V」や「－3V→－5V」）。

A3. いいえできません。三端子レギュレータの出力電圧は、絶対値が入力電圧より小さくなければなりません。

★ **Q4.** 規格値より出力電圧を下げて動作させるには、入力電圧を出力規格より下げればいいですか。

A4. V_O–V_{IN} 特性が示すように、入力電圧が下がれば出力電圧も下がる傾向はありますが、安定動作は保証されません。出力電圧 V_O に最小入出力間電圧差 V_{DIF} を加えた電圧以上を入力してください。

なお、出力規格より低い電圧での回路動作をさせる場合、ツェナー・ダイオードによる定電圧回路の追加で対応できます。RD シリーズは温度係数が、電圧標準ダイオードとするには好ましくありませんが、特性グラフをご参照のうえ、許容できるのであれば、そちらの採用をお勧めします。定電圧回路については、半導体製品の種類の「ツェナー・ダイオードの応用回路例」の冒頭をご参照ください。

Q5. 過熱保護回路は、どのような条件で動作しますか。

A5. 接合部温度（T_j）が絶対最大定格（一般に 150℃）を越えると動作します。T_j が 150～200℃の間で、IC の出力を遮断します。

Q6. 過熱保護回路は、どのような条件で解除されますか。

A6. 接合部温度（T_j）が過熱保護回路動作温度（150℃～200℃）より低くなると、自動的に解除されます。

Q7. 過熱保護回路が動作すると、製品は劣化しませんか。

A7. 過熱保護回路が動作するのは、接合部温度（T_j）が絶対最大定格を越えたためですから、製品が劣化する可能性は高いです。

Q8. 過熱保護回路の目的は、レギュレータ IC 自身と周辺のいずれの保護ですか。

A8. IC 自身が破壊寸前に暴走して、周辺にも影響する場合があります。過熱保護回路は、IC 自身を保護することによって、周辺を含めた安全を確保するものです。

Q9. 過電流制限回路のばらつきは、どのくらいですか。

A9. ピーク出力電流（I_{Opeak}）が、過電流制限回路の動作開始点を表わします。そのばらつきをご参照ください。

Q10. μPC78L05J のような自立タイプ・パッケージ製品の、ジャンクション-ケース間熱抵抗を教えてください。

A10. μPC78L05J は、自立タイプのパッケージの中でも、TO-92 という放熱板を取り付けられないタイプです。したがって、ジャンクション-周囲温度間の熱抵抗値を基に設計してください。放熱板を取り付けられるタイプでは、ジャンクション-ケース間の熱抵抗をデータ・シートに掲載しています。

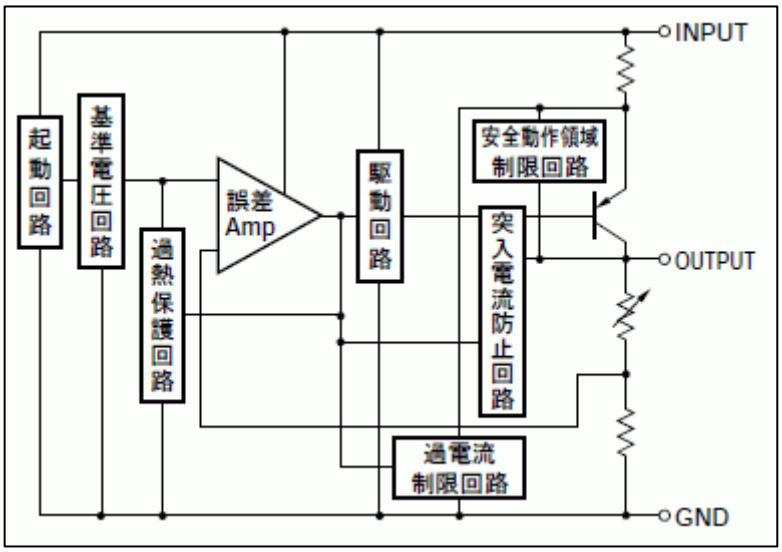
Q11. μPC7800A シリーズの電気的特性で、出力雑音電圧(Vn)の条件 10Hz≦f≦100kHz は何を示しますか。

項目	略号	条件	MIN.	TYP.	MAX.	単位
出力雑音電圧	Vn	T_J=25℃, 10Hz≦f≦100kHz		40	200	μVrms

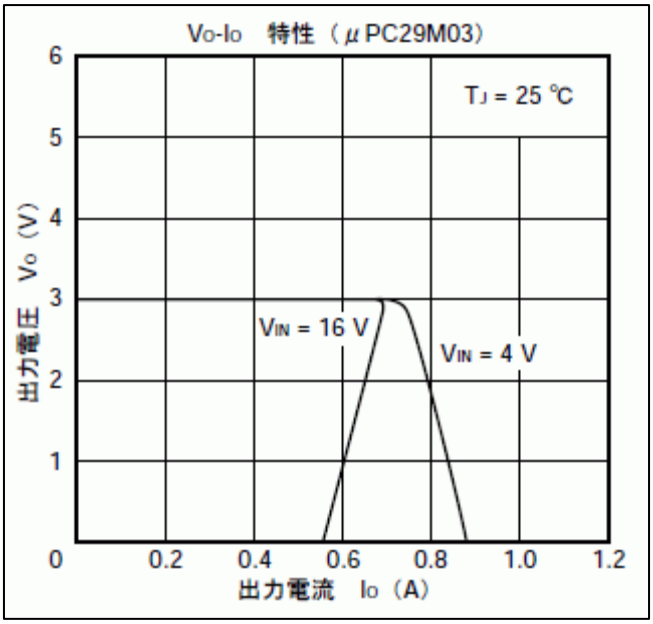
A11. 出力端子にバンドパス・フィルタを設置して、その出力を出力雑音電圧の実効値として測定しています。10Hz≦f≦100kHz は、このバンドパス・フィルタのバンド幅を示しています。単位は μVrms が一般的です。

関連 FAQ：電気に関する単位

Q12. データ・シートに過電流制限回路、過熱保護回路、安全動作領域制限回路が記載されていますが、シャット・ダウン機能はないのですか。また、出力短絡状態では、過電流制限回路だけが動作しているのですか。



A12. 出力電流をゼロとする保護回路（シャット・ダウン機能）は内蔵していません。データ・シート記載の Vo-Io 特性のように、出力電圧が 0V でもある程度の出力電流が流れます。



なお、出力短絡時は過電流制限回路が動作して、そのまま放置すると、熱上昇に伴って過熱保護回路や安全動作領域制限回路も動作します。したがって、過電流制限回路だけが動作するのではありません。

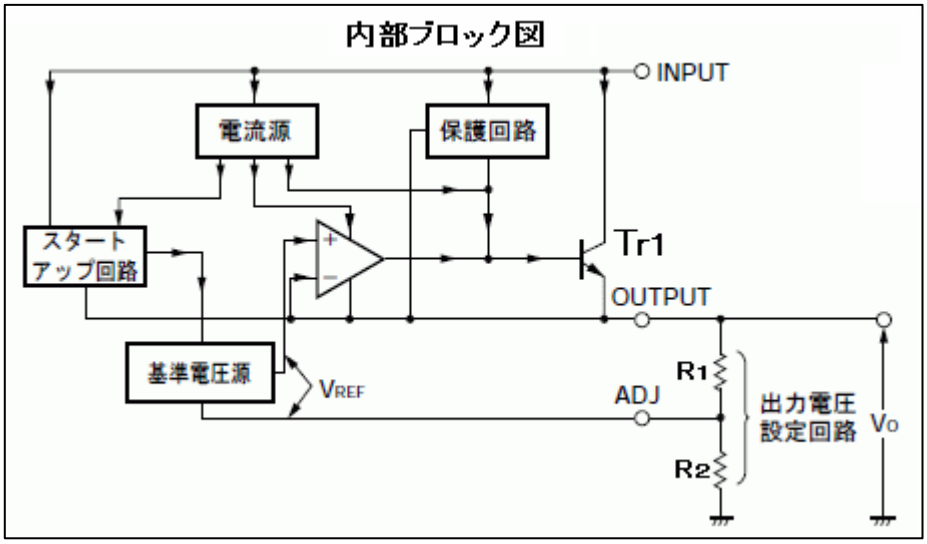
関連 FAQ：用語

Q13. 可変正出力電圧安定化電源 IC(μPC317)で規定されている基準電圧(V_{REF})は、どういう電圧ですか。

電気的特性（特に指定のないかぎり、VIN – Vo = 5 V, Io = 0.5 A, 0°C≦TJ≦+125°C）

項　目	略　号	条　件	MIN.	TYP.	MAX.	単 位
基準電圧	VREF	10 mA≦Io≦1.5 A, PT≦15 W	1.20	1.25	1.30	V

A13. 外部で出力電圧を任意に設定できるので、基準電圧を規定しています。
これを元に、出力電圧を R_1 と R_2 で簡単に求められます。以下に、その動作を説明します。



各ブロックに必要な回路動作電流は、OUTPUT 端子から吐き出され、R_1,R_2 を通じて外部 GND に流れます。一方、ADJ 端子からの流出電流（I_{ADJ}）は、最大でも 100 μA とわずかですが、安定した出力電圧を得るために、I_{ADJ} と V_{REF}/R_1 の電流比を十分に大きくとります。

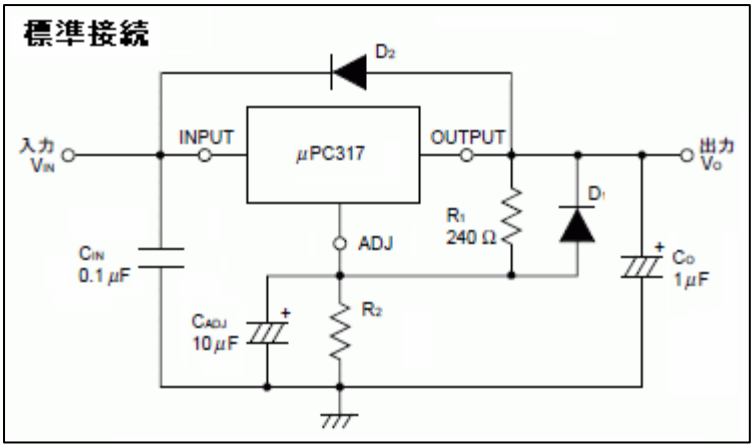
R_1 の推奨値は 240Ω で、V_{REF}/R_1＝1.25V/240Ω＝5.2mA です。この場合の I_{ADJ} との電流比は約 52 です。設定の詳細は後述しますが、これだけの電流比が得られていれば、外付け抵抗比だけで安定した出力電圧設定が可能です。

出力電圧が安定している状態では、R_1 の両端電圧が（誤差増幅器でイマジナル・ショートが成立している）V_{REF} と等しくなっています。

このとき、何らかの要因で OUTPUT の電圧が低下すると、R_1 両端の電圧が V_{REF} よりも小さくなることから、非反転端子には正の電圧が印加され、Tr1 を深くバイアスして OUTPUT 電圧を上げる動きをします。逆に OUTPUT が上がると、R_1 両端の電圧が V_{REF} より大きくなるので、非反転端子には Tr1 を浅くバイアスして、OUTPUT の電圧を下げる動きとなる電圧が印加されます。

このように V_{REF} は、OUTPUT の電圧を一定に保つための基準の働きをします。

さらに、出力電圧が一定となる動作をすることを利用し、外部回路で出力電圧を任意に設定できる構成とすることも可能です。次の標準接続を例に、これを説明します。



ADJ 端子を基準にすると、基準電圧 V_{REF} は ADJ 端子と OUTPUT 端子間の電圧として確認できます。この端子間を流れる電流(1.25V/240Ω＝5.2mA)を利用して、R_2 の抵抗値を変えることにより、ADJ 端子の電位を設定(かさ上げ)できます。

そして出力電圧は、次式によって決定されます。

$$Vo = R_2(I_{ADJ} + V_{REF}/R_1) + V_{REF}$$

$$\fallingdotseq R_2 \times V_{REF}/R_1 + V_{REF}$$　（ADJ 端子の流出電流 I_{ADJ} を無視）

$$= (1 + R_2/R_1) \times V_{REF}$$

この式から V_{REF} が基準電圧となり、出力電圧を設定できることが分かります。

R_1＝240Ω の場合の出力電圧 Vo は、次のとおりです。

出力電圧 Vo(V)	R2（Ω：TYP.）
2.5	240
5.0	720

R_1 が 240Ω 以外の大きな値の場合、次の注意が必要です。

R_1＝240Ω では、無負荷でも 5.2mA（＝1.25V/240Ω）が流れます。ADJ 端子流出電流(I_{ADJ})は Max.100μA で、5.2mA よりかなり小さい値なので無視できます。

しかし、R_1＝1kΩ などにすると、R_1 に流れる電流が少なくなり、I_{ADJ} の値が無視できなくなるため、出力精度が悪くなることが考えられます。

Q14. μPC317 の全損失(P_T)は、次式で求められますか。

$P_T = (V_{IN} - V_O) \cdot I_O + V_{IN} \cdot I_{bias}$

$(V_{IN} - V_O)$：入出力間電圧差

I_O　：出力電流

V_{IN}：入力電圧

I_{bias}：回路動作電流

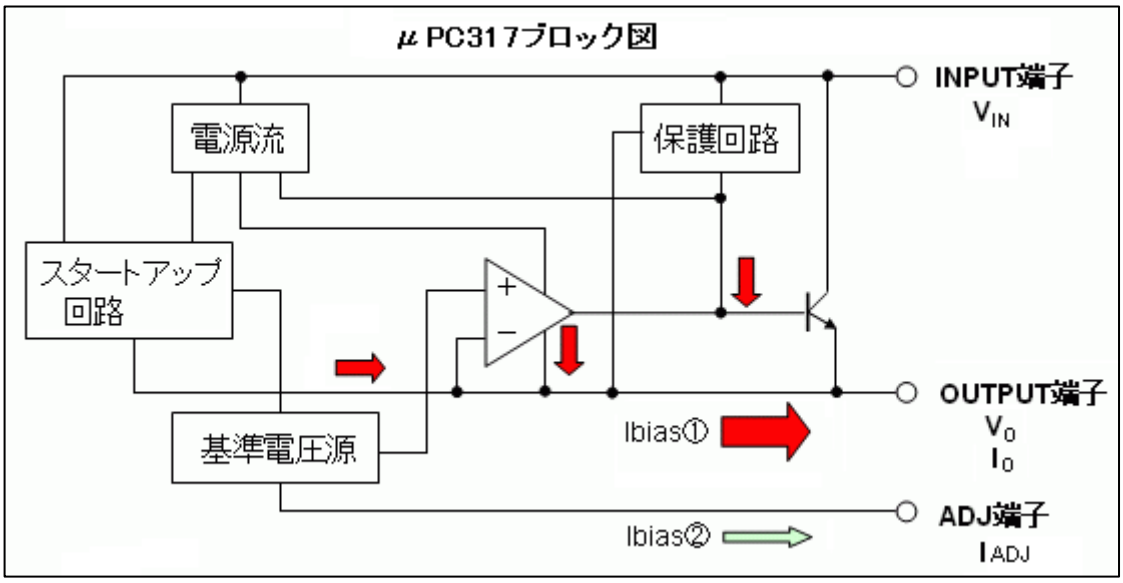
A14. μPC317 は通常の三端子レギュレータとは回路構成が異なっています。具体的には下記ブロック図で、回路動作電流のほとんど(I_{bias}①)が出力電流(I_O)に含まれ、OUTPUT 端子から流出する構成となっています。そして残る回路動作電流(I_{bias}②)は、ADJ端子からI_{ADJ}として流出します。

このことから、全損失 P_T は次のようになります。

$P_T = (V_{IN} - V_O) \cdot I_O + V_{IN} \cdot I_{ADJ}$　ただし、I_O は I_{bias}①を含む

ここで、I_{ADJ}は電気的特性に記載のように、μA なので無視でき、全損失 P_T は次式で算出できます。

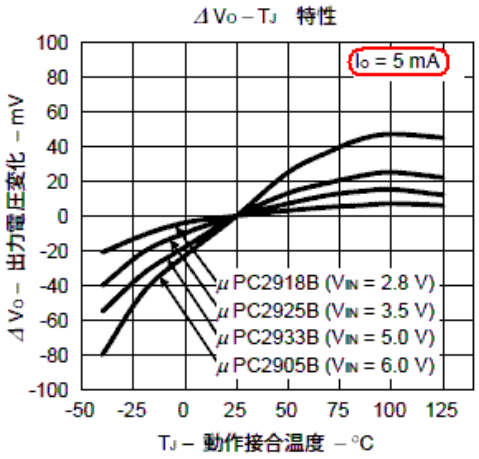
$P_T = (V_{IN} - V_O) \cdot I_O$



★ **Q15.** 三端子レギュレータの「出力電圧の温度変化」特性で、条件が Io＝5mA になっていますが、出力電流が変化するとどのようになりますか。

μPC2925B（特に指定のないかぎり，T_J = 25°C，V_{IN} = 3.5 V，I_O = 0.5 A，C_{IN} = 0.1 μF，C_{OUT} = 10 μF）

項　目	略　号	条　件	MIN.	TYP.	MAX.	単　位
出力電圧	V_{O1}		2.45	2.5	2.55	V
	V_{O2}	3.5 V≦V_{IN}≦12 V, 0 A≦I_O≦1 A	(2.425)	–	(2.575)	V
入力安定度	REG_{IN}	3.5 V≦V_{IN}≦12 V	–	5.5	25.0	mV
負荷安定度	REG_L	0 A≦I_O≦1 A	–	3.5	40.0	mV
回路動作電流	I_{BIAS}	I_O = 0 A	–	1.8	4.0	mA
出力電圧温度変化	$\Delta V_O/\Delta T$	0°C≦T_J≦125°C, I_O = 5 mA	–	0.2	–	mV/°C



A15. 温度特性は動作接合温度 T_J で決まるだけで、出力電流での変化はありません。
自己発熱をおさえて、周囲温度で T_J に近似できるように Io＝5mA で測定しただけで、本来は表記する必要のない条件です。

応　用

Q1. 入力側と出力側のコンデンサは、必要ですか。

A1. はい、必要です。低飽和型レギュレータや負出力レギュレータは、コンデンサの容量や種類によっては発振しやすいので、必ずデータ・シートに記載されているものをご使用ください。また、入力側のコンデンサは、温度特性や周波数特性の良いものを推奨します。

Q2. 外部容量の推奨はタンタル・コンデンサやフィルム・コンデンサですが、他のコンデンサは使えませんか。

A2. 温度特性、周波数特性が良いコンデンサであれば、どのようなコンデンサでも結構です。
コンデンサには、温度特性と周波数特性が劣る製品がありますので、ご注意ください。

(1) 電解コンデンサは、低温(0℃以下)で著しく容量が小さくなる製品があります。
ご使用の温度で、所定の容量が確保できる必要があります。

(2) 一般に電解コンデンサは、100～120Hz という低い周波数で容量を測定しています。
周波が上がると、著しく容量が低下する製品があります。
三端子レギュレータは 1～3MHz 付近で発振します。したがって、1～3MHz 付近で、所定の容量が確保できる必要があります。周波特性が悪いコンデンサを使用する場合、並列に高周波特性の良いコンデンサを入れるなどして、発振を抑える対策が必要です。

電解コンデンサでは、特に注意が必要ですが、セラミック・コンデンサなどでも、特性の悪い製品がありますので、特性にご注意ください。タンタル・コンデンサやフィルム・コンデンサは、一般に特性が安定しているので、データ・シートではこれらの使用をお勧めしています。
なお、ESR(等価直列抵抗)や ESL(実効直列インダクタンス)が小さいコンデンサでは、発振することがありますので、ご使用になるコンデンサでご評価ください。ESR,ESL は温度や周波数の影響を受けます。

Q3. 並列接続で、駆動電流を増やすことはできますか。

A3. だめです。電流容量が不足の場合は、放熱対策をするか、出力にトランジスタを接続するなどしてください。
参考資料：三端子レギュレータの使い方

Q4. 低飽和レギュレータの GND に抵抗を挿入して、出力電圧を上げることはできますか。

A4. いいえ、できません。低飽和型レギュレータでは、定常時の回路動作電流より大きな起動時回路動作電流が流れますので、そのような回路では、起動時や停止時に出力電圧が上昇してしまいます。
参考資料：三端子レギュレータの使い方

Q5. データ・シートに記載している推奨出力電流を越えてもいいですか。

A5. 出力電流の最大値(μPC2900 シリーズでは 1A)を越えても、破壊することはないと思われます。
なお、三端子レギュレータでは、出力電流を規定していません。過電流制限回路を内蔵しており、過負荷では出力電流を制限します。ただし、推奨条件を越えた出力電流では、特性が保証されていませんので、他の IC を選定されることをお勧めします。

Q6. 出力の垂下特性を教えてください。

A6. $I_{Opeak}-(V_{IN}-V_O)$ 特性から、簡易的に求めることができます。

Q7. 推奨条件より低い V_{IN} での使用を続けると、どうなりますか。

A7. 入力電圧にほぼ等しい電圧を出力します。ただしこのような条件では、電気的特性や各種保護回路の動作が保証されません。また、低飽和型三端子レギュレータでは、レギュレーション開始電圧より V_{IN} が低いと、大きな回路電流が流れて高温になることがあります。

Q8. 推奨条件より高い V_{IN} で使用すると、どうなりますか。

A8. 絶対最大定格を越えなければ、破壊することはありません。ただし、入出力間電圧差が大きくなるため、温度上昇に対する配慮が必要です。また、絶対最大定格の入力電圧との差が少なくなることに、ご注意ください。

Q9. 無負荷で使用した場合、破壊や温度上昇といった問題は生じますか。

A9. 無負荷でも、基本的に問題ありません。ただし、発振する可能性が高いので、セットで十分ご評価ください。
なお、可変出力型は、最小出力電流を規定しています。無負荷にしないでください（トラブル・シュート参照）。

Q10. 接合部温度が絶対最大定格を越えなければ、動作周囲温度が絶対最大定格を越えてもいいですか。

A10. いいえ、信頼性は保証できません。絶対最大定格は、1 項目でも越えてはなりません。

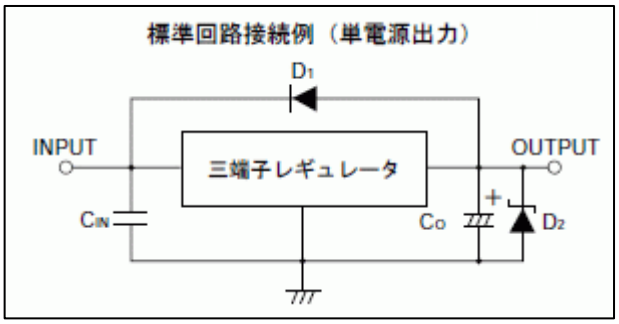
Q11. 可変出力型の三端子レギュレータで、5V から 3.3V を出力できますか。

A11. いいえ、できません。μPC317 や μPC7800 シリーズは、入出力電圧差が 3V 以上必要です。入出力電圧差が 1V 以上で動作する μPC2900 シリーズのうち、μPC2933（3.1V 出力）をお使いください。

Q12. μPC7805A の V_O が3項目あります。2 つは温度条件が記載されていますが、1 つは記載されていません。条件が記載されていない項目は、T_j =0～125℃と考えていいですか。

A12. はい、そのとおりです。条件が記載されていない項目は、欄外の条件（共通条件）が適用されます。

Q13. 「標準接続」にある D1(OUTPUT が INPUT より高電圧を想定)には、どのようなダイオードがいいですか。



A13. 電源 IC に対するダメージを軽減する目的から、ショットキ・バリア・ダイオードを推奨します。なお、INPUT−OUTPUT の電圧差以上の耐圧が必要です。

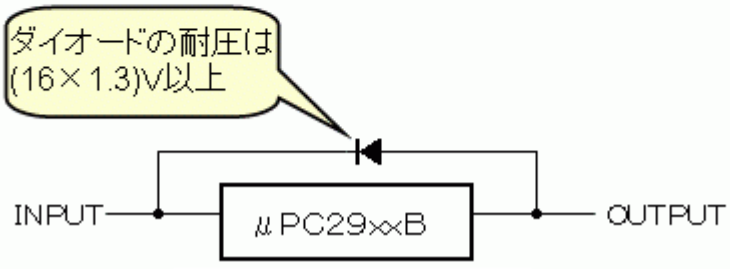
Q14. 三端子レギュレータ(正電源用)で、逆流防止用ダイオード(上記 D1)の耐圧は、何Vのものがいいですか。

A14. この場合のダイオードの耐圧は、絶対最大定格の入力電圧の 1.2～1.3 倍程度以上としてください。

例 μPC29xxB シリーズ

絶対最大定格（特に指定のないかぎり，T_A = 25°C）

項　目	略　号	定　格		単　位
		μPC29xxBHF	μPC29xxBHB μPC29xxBT μPC29xxBT1D	
入力電圧	V_{IN}	−0.3～+16.0		V



Q15. μPC7805AHF の入力電圧を 30V に設定したいのですが、推奨動作条件の入力電圧は Max.25V です。30V で使用すると、どのような問題がありますか。

A15. 入力電圧を 30V としても、特性悪化や破損などは生じません。ただし、絶対最大定格の内部消費電力(P_T)を越えない範囲でご使用ください。

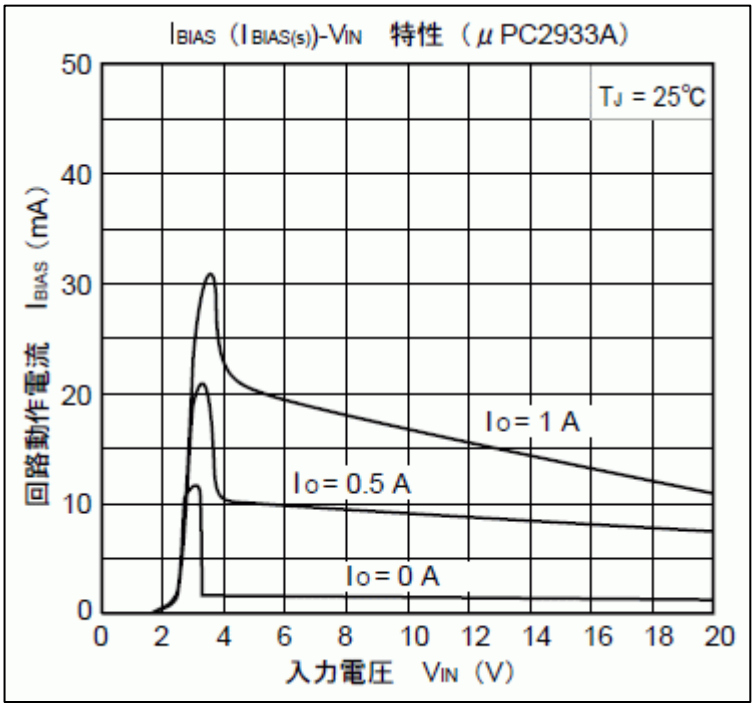
Q16. μPC2933A の回路設計で、回路電流(I_{BIAS})の規格に対して、どのように注意すればいいですか。

A16. I_{BIAS} は、定常状態で電源用 IC 内部の各ブロックが動作するのに必要な電流です。入力端子から GND 端子へ流れる電流を規定しています。

このほかに、起動時回路動作電流($I_{BIAS(s)}$)についても注意が必要です。μPC2900 シリーズのような低飽和型レギュレータは、出力段トランジスタにPNPトランジスタを使用しているため、起動時の入力電圧が低いときにこのトランジスタが飽和して、入力から GND に大電流が流れます。起動時回路動作電流は、この電流の最大値です。この電流値を越える電流が流れないと、レギュレータは正常に動作しません。

【μPC2933A の例】

項　目	略号	条　件	MIN.	TYP.	MAX.	単位
回路動作電流	I_{BIAS}	Io=0A		2.0	3.0	mA
起動時回路動作電流	$I_{BIAS(s)}$	V_{IN}=3.1V,Io=0A		10	30	mA



この例の回路電流は、定常状態で動作するのに 2.0mA を要するデバイスが平均的であり、最大で 3.0mA が流れるデバイスが含まれることを示します。

同様に起動時では、最大で 30mA の回路電流が流れるデバイスが含まれていることを示します。

以上から、回路負荷に流す電流に加え、30mA 以上の電流がデバイスの入力端子に供給できる外部電源が必要となります。

トラブル・シュート

Q1. 2 電源回路に三端子レギュレータを使うと、どちらか一方しか動作しないのですが、なぜですか。

A1. 出力端子に外部から電源が供給されると、IC は動作しなくなります。2 電源方式の場合は、どちらか一方が先に立ち上がると、負荷を通してもう一方の IC に逆の電圧が供給されます。この状態で IC は動作しません。
なお、いったんこの状態になると、すべての端子に加わる電圧がゼロになるまで回復しません。
出力端子に逆電流が流れないように、順方向電圧の小さなダイオード（ショットキ・バリア・ダイオードなど）で保護してください。

Q2. 出力端子を抵抗でプルアップしたら、動作が不安定になりました。

A2. 三端子レギュレータは、負荷へ電流を供給することを前提に設計してあります。
外部から出力端子へ電流を供給すると制御不能になり、正常な動作ができなくなります。
なお、いったんこの状態になると、電源を切るまで回復しないこともあります。プルアップはしないでください。

Q3. 可変出力型の μPC317 で、出力をオープンにすると、出力電圧が規定より高くなります。

A3. 可変出力型では、出力電流の最小値が規定されていますので、出力はオープンにできません。
μPC317 は、3mA 以下の負荷電流では起動しません。この状態では、出力電圧が規定より高くなります。

Q4. 正負電源を構成すると、出力電圧が上昇しません。

A4. 正電源の立ち上がりが負電源より遅い場合、正電源の GND から OUTPUT に電流が流れるので、正電源が立ち上がらなくなります。両端子間にダイオードを接続して、電流をバイパスさせてください。

Q5. 負荷を接続すると、出力電圧が低下します。

A5. 次のような、原因と対応が考えられます。

原因：出力が発振している。
対応：入力と出力に接続しているコンデンサの接続を確認する。

原因：ピーク出力電流を越える電流が流れている。
対応：入力電圧を下げる。放熱を良くして、チップ温度を下げる。
注意　モータやランプなどが負荷の場合は、起動時に大きな電流が必要です。

原因：チップ温度が 150℃を越えている。
対応：入力電圧を下げる。放熱を良くして、チップ温度を下げる。

原因：入力と出力を逆に接続している。
対応：端子接続をご確認ください。

原因：前段の電源の容量が不足している（低飽和型レギュレータの場合）。
対応：低飽和型レギュレータでは、出力電流に加え、起動時の回路動作電流を供給できる電源を用意してください。

Q6. 出力が発振します。

A6. 次のような、原因と対応が考えられます。

原因：入力と出力に、データ・シートで指定していないコンデンサが接続されている。

対応：入力側にアルミ電解コンデンサしか接続されていない場合は、指定のコンデンサも接続してください。
また、(積層)セラミック・コンデンサを接続されている場合は、指定のコンデンサに替えてください。

原因：コンデンサと IC の距離が長い。

対応：配線の長さは、1cm 以下にしてください。

注意　低飽和型／負出力型の三端子レギュレータでは、等価直列抵抗の小さい(低インピーダンス)コンデンサを出力に使用してください。

Q7. 端子の根元に、地金が出ています。

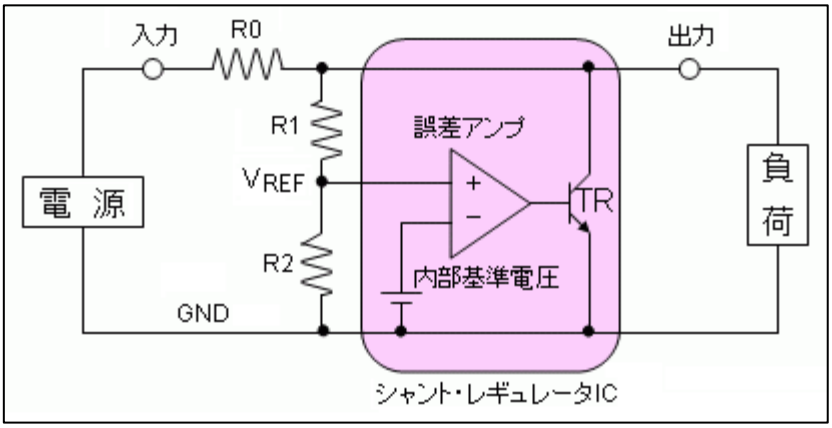
A7. 問題ありません。リードの細い部分まで、めっきされていれば良品です。

6.2.4 シャント・レギュレータ

基　　礎

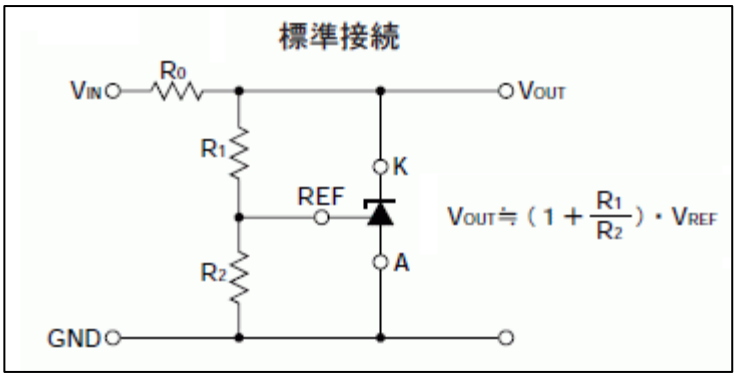
Q1. シャント・レギュレータは、どのように動作するのですか。

A1. シャント・レギュレータは、外付け抵抗を介して出力電圧の変動を誤差アンプで検出し、負荷に対して**並列**に接続されたトランジスタを制御することで、出力電圧を一定に保ちます。つまり、シャント・レギュレータは電源に対する負荷の変動を小さくするのが特徴です。なお、基本的な制約事項として、シャント・レギュレータに流せる電流の Min. と Max. の差分までの負荷電流の変化までしか対応できません。
簡易的に表現した次の回路で、基本動作を説明します。

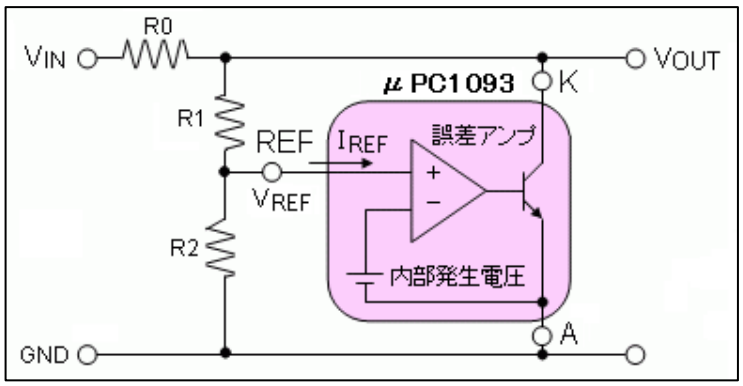


負荷変動で**出力電圧が低下した場合**、抵抗 R_1,R_2 に加わる電圧が低下します。これにより、出力電圧を R_1,R_2 で分割した V_{REF} の電圧が低下します。つまり、誤差アンプの非反転(＋)端子の入力電位も低下(内部基準電圧より低下)します。この結果、誤差アンプはトランジスタ TR のベース電位を低下させ、TR のコレクタ電流が抑えられます。これによって、出力電圧が上昇して、電圧を安定させます。
負荷変動で**出力電圧が上昇した場合**は、V_{REF} が上昇するので、誤差アンプはトランジスタ TR のベース電位を上昇させ、TR のコレクタ電流増加させます。これによって、出力電圧が低下して、電圧を安定させます。
つまり、V_{REF} の電位と内部基準電圧とをつねに等しくさせる動作をします。

Q2. μPC1093 の標準接続で、V_{REF} はどのような電圧を表し、これを用いて V_{OUT} はどのように導かれますか。



A2. V_{REF} は REF 端子の入力電圧です。これが基準電圧（リファレンス電圧）となり、外部抵抗によって出力電圧 V_{OUT} を設定します。



次の関係式が成り立ちます。

$V_{OUT}=V_{REF}+(R_1$ の電圧降下$)=V_{REF}+(V_{REF}/R_2+I_{REF})\times R_1=V_{REF}(1+R_1/R_2)+I_{REF}\times R_1$

ここで、I_{REF} は μA オーダの電流なので、$I_{REF}\times R_1$ を省略して次式のように表せます。

$V_{OUT}\fallingdotseq V_{REF}(1+R_1/R_2)$

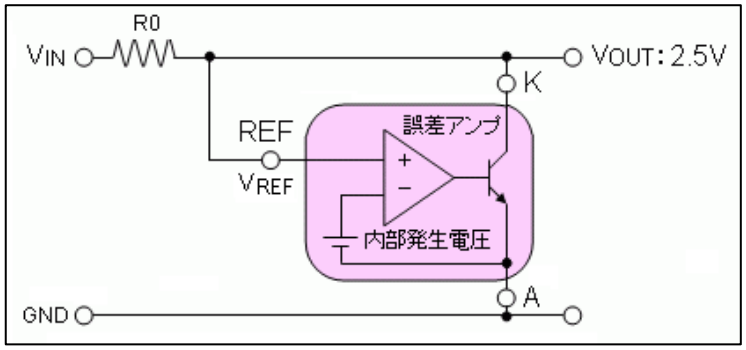
また、分圧の法則で簡単に考えることもできます。

$V_{OUT}：V_{REF}=(R_1+R_2)：R_2$

$V_{OUT}\times R_2=V_{REF}\times(R_1+R_2)$

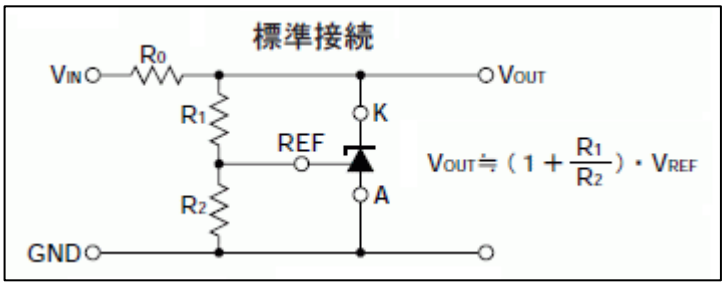
$V_{OUT}=V_{REF}\times(R_1+R_2)/R_2=V_{REF}\times(1+R_1/R_2)$

V_{REF} は 2.495V(TYP.)で安定するので、R_1 と R_2 の比率を変えれば、出力電圧 V_{OUT} が変化します(推奨動作条件：MIN.V_{REF}～MAX.36V)。なお、μPC1093 を 2.5V の定電圧電源として使用する場合は、次のような接続で R_1=0、R_2=∞(オープン)とします。



応　用

Q1. μPC1093 の外付け抵抗の値は、どのように決定すればいいですか。



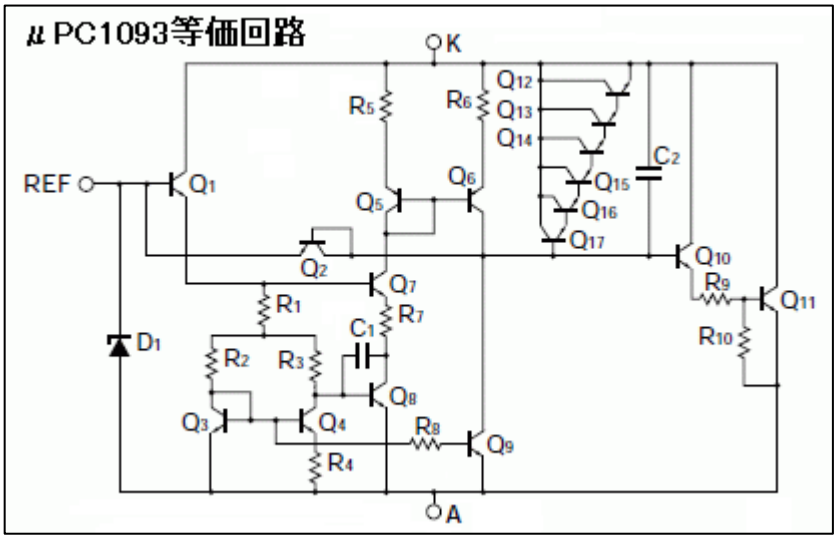
A1. R_1 と R_2 は、REF 端子に流れるリファレンス電流を十分確保できるように設定します。ただし、これらの値を小さくする、と無駄な電力を「V_{IN}→R_0→R_1→R_2→GND」の経路で消費します。また、値が大きすぎると、I_{REF} の電流が確保できません。この 2 点を考慮して、適当な数値を決定してください。

なお R_0 は、V_{IN} と V_{OUT} の電圧差と、K→A に流す電流値で決定します。

Q2. μPC1093 の電気的特性に回路動作電流が掲載されていませんが、応用回路設計の注意点はありますか。

A2. アノードを基準に考えると、μPC1093 は回路構造上、カソード電流とリファレンス電流の 2 電流が、アノードに流れます。つまり、カソード電流とリファレンス電流が、回路電流に相当します。

このため、応用回路設計では、カソード電流とリファレンス電流が絶対最大定格を越えないようにする必要があります。また、推奨動作条件に記載しているように、カソード電流の推奨最小値は 1mA です。カソード電流を 1mA 以上確保できない場合、μPC1093 の起動に必要な電流が不足し、起動できなくなります。



規格区分

Q1. μPC1093 には、－1、－2 などの製品があるようですが、どのような製品ですか。

A1. 一般品から Vref を選別した、特殊規格の製品です。Vref は次のとおりですが、特殊規格の内容は変更されることがありますので、購入先にご確認ください。

品名	Vref（単位：V）
μPC1093J	2.440 ―― 2.550
μPC1093J－1	2.475 ―― 2.525
μPC1093J－2	2.455 ―― 2.505

6.2.5 スイッチング・レギュレータ

用　語

出力電圧

出力端子に印加できる、最大の電圧です。三端子レギュレータの出力端子には、外部から電圧を印加できませんが、制御 IC では、0V から定格値までの電圧を印加できます。

出力電流

内蔵の出力トランジスタに流すことができる電流です。

低電圧誤動作防止回路

電源電圧の立ち上がり時に、動作開始電圧に達するまで、IC の動作を停止させておく回路です。また、電源電圧が下がる場合にも、ある電圧で動作を停止させます。

スタンバイ電流

低電圧誤動作防止回路の機能によって、動作が停止している状態で、IC に流れる電流です。少ない方が好ましいです。

回路(動作)電流

動作状態で、IC に流れる電流です。少ない方が好ましいです。

スレッシュホールド電圧

内部で発生する三角波の、上限値と下限値のことです。スレッシュホールド電圧で、出力のデューティが 0%か 100%(μPC494 では最大値)になります。

マスタ・スレーブ動作

あるスイッチング・レギュレータ IC が、内蔵する発振器で、複数のスイッチング・レギュレータ IC を同期運転させることです。

基　礎

Q1. μPC1909 のデータ・シートの冒頭に「アクティブ・クランプ型回路に使用する際は、Vicor 社の許諾が必要」と記載されています。特許の番号などを教えてください。

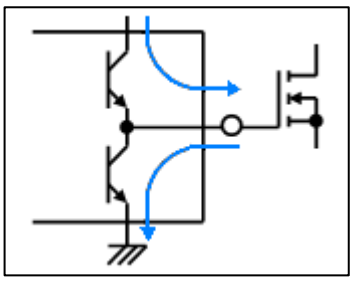
A1. アクティブ・クランプの特許番号は「USP No.4,441,146」です。Vicor 社では Reset Patent と呼んでいるようです。詳細は Vicor 社のホームページをご参照ください。
なお、特許使用に関する交渉は、お客様と Vicor 社の間で直接行われますので、当社は関与していません。

Q2. スイッチング・レギュレータの出力回路は、どのようになっているのですか。

A2. 出力には、トーテムポール、オープン・コレクタ／オープン・ドレーン、プッシュプルの3タイプがあります。

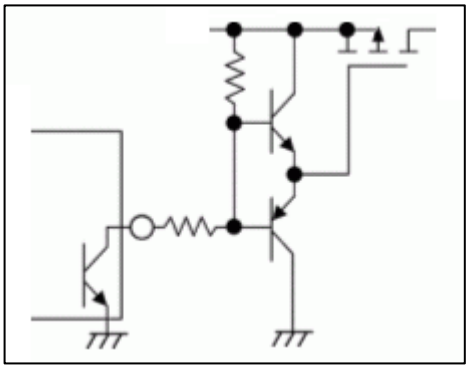
(1) トーテムポール

スイッチング電源のメイン・スイッチング素子には、パワーMOS FET を使用することが一般的です。パワーMOS FET を高速に駆動するには、そのゲート容量を急速に充放電する必要があります。
トーテムポール回路は、この急速充放電をする、大きなピーク電流を供給できるプリドライバ回路です。
この回路を制御 IC に内蔵することで、シンプルなスイッチング電源回路になります。



(2) オープン・コレクタ／オープン・ドレーン

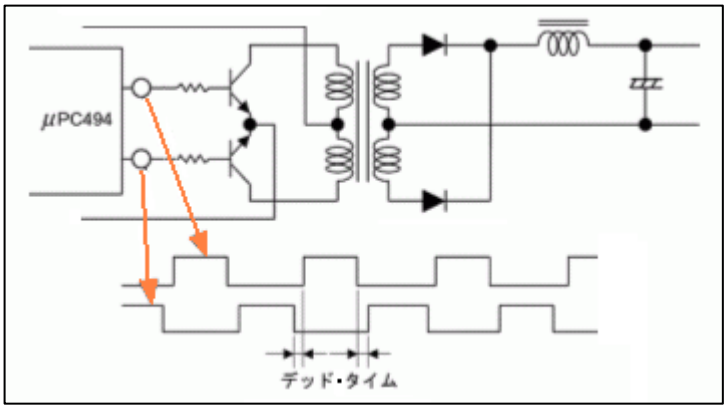
メイン・スイッチング素子の特性に応じて、外付けで最適な駆動回路を設計できるメリットがあります。



(3) プッシュプル

プッシュプル方式とは、スイッチング電源の回路方式の名称です。メイン・スイッチング素子を 2 つ使用して、スイッチング・トランスを交互に駆動します。トランスの利用率が上がるので、大きな出力のスイッチング電源に適する方式です。

プッシュプル方式のスイッチング電源では、2つのトランジスタが同時にオンしないように、デッド・タイムを設けます。μPC494 では、このデッド・タイムの設定が容易です。



Q3. 正電圧から負電圧が得られますか。

A3. はい、できます。回路構成によっては、次のような出力ができます。

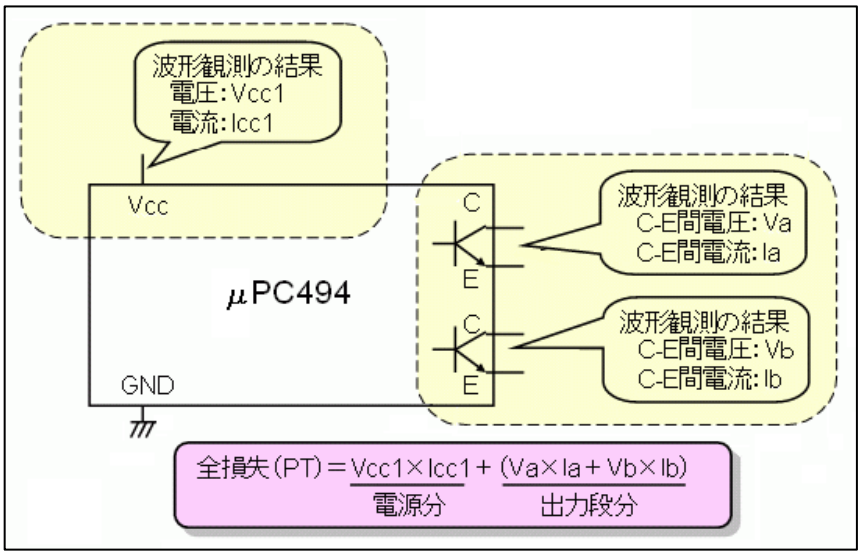
・入力電圧より低い出力電圧（降圧）

・入力電圧より高い出力電圧（昇圧）

・入力電圧と逆極性の出力電圧（反転）

Q4. μPC494 の全損失算出方法を教えてください

A4. μPC494 の動作時の消費電力は、Vcc 端子の電圧と流入する電流を観測して、損失を計算してください。
出力段はオープンコレクタですので、さらに出力電圧と出力電流による損失も同じく波形観測により計算し、前述の損失値に加算することで全損失とします。



応　用

Q1. μPC494 は、デッド・タイム・コントロール入力（4 番ピン）をハイにすれば、出力を遮断できますか。

A1. はい、できます。
この端子に約 3.3V 以上（V_{REF} 以下）の電圧を与えれば、出力段トランジスタがオフとなります。
μPC494 のデット・タイム・コンパレータには、約 0.1V のオフセット電圧が与えられています。
ただし、タイミング・コンデンサの容量が増加すると、デッド・タイム・デューティが減少して、0%になる可能性があります。そのため、プッシュプル・モードでご使用の際は、注意が必要です。
詳細は、アプリケーション・ノートの「4.5 発振周波数－PWM コンパレータのスレッシュホールド電圧（デッド・タイム・コントロール）」をご参照ください。

Q2. μPC1099 の『スタンバイ電流』と『オフ電流』の違いを教えてください。

A2. スタンバイ電流：電源電圧が 8V 以下に低下すると出力を遮断します。 この状態での電源電流をスタンバイ電流と呼びます。
オフ電流　　：通常動作時に、ON/OFF 端子からオフにしたときの電源電流をオフ電流と呼びます。

Q3. μPC1100 のタイマ・ラッチ電圧が 2.25V を越えると作動するようですが、三角波の電圧が 1.45～2.05V のため、 2.25V では出力が ON 状態（直流）となって、MOS などドライブ用の素子を破損してしまうと思われます。そこで、出力短絡状態でも、あるデューティ比で出力したいのですが、どうすればいいですか。

A3. ご心配のとおり、直流状態にならないよう設定する必要があります。
この設定には、デッド・タイム・コントロールを用います。デッド・タイム・コントロール端子の電圧を三角波の山（2.05V）より低い電圧に設定すれば、直流状態を避けることができます。
ユーザーズ・マニュアルの「5.ブロック図および動作波形」の「図 3 動作波形」を参考に、直流にならないようデッド・タイム・コントロール端子（6, 11 番ピン）の電圧を調整してください。

Q4. μPC1100 の出力が、電源投入時にオーバシュートします。

A4. 一般には、ソフト・スタート（6 番ピンにコンデンサを付加）機能の利用をお勧めしています。

Q5. μPC1935 が発振しているかどうか、確かめる方法を教えてください。

A5. 残念ながら、発振しているかどうかを確認する方法はありません。
もし、出力波形が出ないのなら、タイマ・ラッチ回路がノイズなどで動作してしまい、出力を遮断していることが考えられます。そこで、DLY 端子を GND に短絡して、タイマ・ラッチが働かないようにしてご確認ください。
なお、この状態では、出力の短絡保護が働きませんので、実験の際には出力の短絡にご注意ください。

トラブル・シュート [製品共通]

Q1. ドキュメントの記述どおりに回路設計をしても、動作しません。

A1. 次の事項をご確認ください。
- 誤配線はありませんか。
- GND の引き回しは、誤動作の原因となります。信号系とパワー系の GND を分離して、それぞれ 1 点接地にしてください。
- V_{REF}（REFERENCE、REF）端子と GND 端子の間に、コンデンサが接続されていますか。コンデンサは、フィルムかタンタル、セラミックなどのタイプをご使用ください。

トラブル・シュート [μPC494]

Q1. プッシュプル動作で、パルスが連続して出ることがあります。

A1. GND 端子にノイズが混入して、誤動作している可能性があります。
グランド配線を見直してください。また、Ref Out 端子（14 ピン）と GND 端子（7 ピン）の間に、コンデンサが接続されていますか。コンデンサは、フィルムかタンタル、セラミックなどのタイプをご使用ください。

トラブル・シュート [μPC1100, 1150]

Q1. 動作する場合と、しない場合があります。

A1. タイマ・ラッチ回路が、誤動作している可能性があります。次の事項をご確認ください。
- 誤配線はありませんか。
- GND の引き回しは、誤動作の原因となります。信号系とパワー系の GND を分離して、それぞれ 1 点接地にしてください。
- REF OUT 端子（16 ピン）と GND 端子（8 ピン）の間に、コンデンサが接続されていますか。コンデンサは、フィルムかタンタル、セラミックなどのタイプをご使用ください。
- DLY 端子（15ピン）に接続するディレイ用コンデンサ C_{DLY} の GND 側は、GND 端子（8ピン）に極力近づけて接地してください。
- ON/OFF コントロール端子（4 ピン）の使用時は、この入力の立ち上がりを 0.07V/μs 以下にしてください。

Q2. μPC1100 で 1 つの出力が短絡すると、その出力しか遮断しません（両出力の遮断にならない）。

A2. タイマ・ラッチ回路が、誤動作している可能性があります。上記 A1 に記載の事項をご確認ください。

6.2.6 製品照会

最も低い電圧を出せる電源用 IC

Q1. μPC2900 シリーズを使っています。3.3V 以下の製品はありますか。

A1. 2.5V の製品ができましたのでご利用ください。なお、2.0V 以下は、μPC317 をご検討ください。
μPC317 は、1.3V から使用できます。ただし、μPC317 の入出力間電圧差は、3V 以上必要です。

低飽和型の負電圧三端子レギュレータ

Q1. 負電圧を出せる、低飽和型の三端子レギュレータはありますか。

A1. 残念ながらありません。負電圧を出せる製品は、μPC7900 シリーズと μPC337 です。

負の高精度基準電圧発生器

Q1. 正電圧は μPC1060 を使っています。負電圧を出せる、高精度基準電圧器はありますか。

A1. 残念ながらありません。なお、μPC1060 で発生させた電圧を、ゲイン 1(0dB)の反転増幅器を用いてマイナスの電圧に変換する方法があります。この方法をご検討ください。

低飽和三端子レギュレータの製品の違い

Q1. μPC2905, 2933 と μPC 2905A, 2933A、μPC2905B, 2933B は、なにが違うのですか。

A1. おもな相違点を下表にまとめます。
その他の違いや詳細は、各データ・シートをご参照ください。

項　目	条　件	μPC2905/2933	μPC2905A/2933A	μPC2905B/2933B	単位
出力電圧(V_O)精度	I_O＝0.5A	±4	±3.5	±2	%
入力電圧(V_{IN})	最大定格	20		16	V
回路動作電流(I_{BIAS})	I_O＝0A, TYP.	2.2/2.0		1.8	mA
出力付加コンデンサ		47		10	μF
リップル除去率(R・R)	f＝120Hz	61/64		57/58	dB

★ Q2. μPC29M05, 29M33 と μPC29M05A, 29M33A は、なにが違うのですか。

A2. おもな相違点を下表にまとめます。
その他の違いや詳細は、各データ・シートをご参照ください。

項　目	条　件	μPC29M05	μPC29M05A	μPC29M33	μPC29M33A	単位
出力電圧(V_O)精度	I_O＝0.35A	±4	+3.6/−3.4	±3.9	±3.6	%
回路動作電流(I_{BIAS})	I_O＝0A, MAX.	4.0	3.0	4.0	3.0	mA
起動時回路動作電流($I_{BIAS(s)}$)	V_{IN}＝4.5V, MAX.	30	20	30	20	mA
最大変化量(ΔI_{BIAS})	$T_J \leqq 125°C$, $V_{IN} \leqq 16V$	20	15	20	15	mA

リセット信号出力機能付き製品

Q1. マイコンに対して、リセット信号の出力機能を持った電源用 IC はありますか。

A1. 残念ながらありません。なお、電源電圧の出力を ON/OFF できる安定化電源用 IC はあります。

6.2.7 共通事項

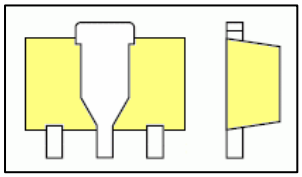
捺印品名

Q1. 電源用 IC の中で、捺印品名が記号化されているのはどの製品ですか。

A1. パワー・ミニ・モールド(SOT-89/SC-62)、5 ピン・ミニ・モールド(SC-74A)のパッケージを採用している製品で記号化をしています。記号の内容は、各製品のデータ・シートをご参照ください。

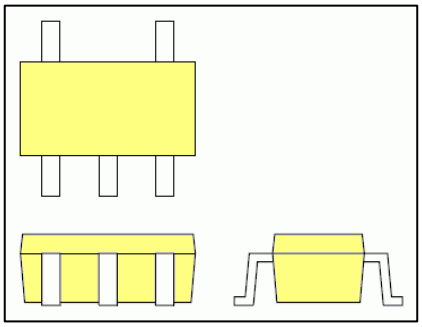
<SOT-89>

μPC1093T(捺印:93)
μPC1943T
μPC1944T
μPC78LxxT シリーズ
μPC29LxxT シリーズ
μPD120NxxT1B シリーズ



<SC-74A>

μPC1093TA(捺印:K93)
μPC1945TA
μPD120NxxTA シリーズ

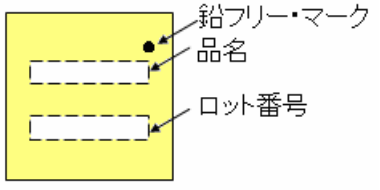
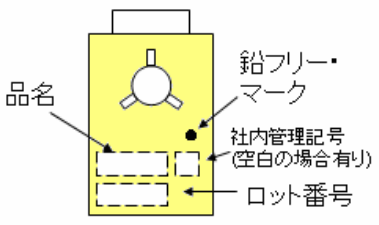


捺印表示

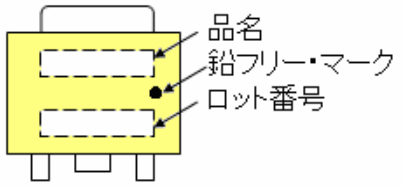
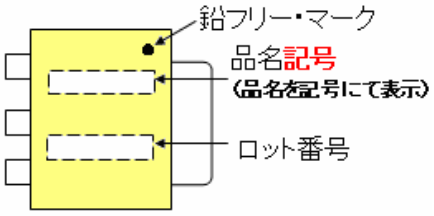
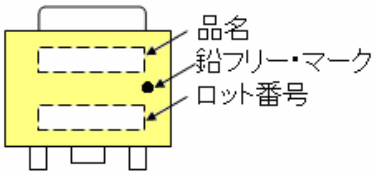
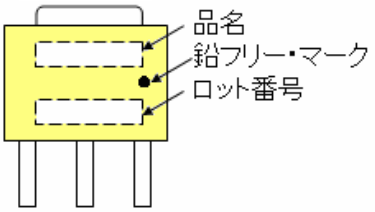
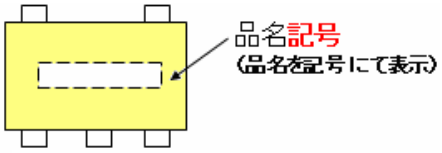
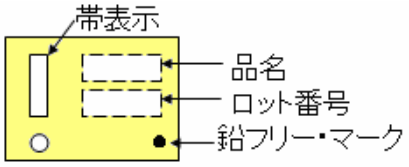
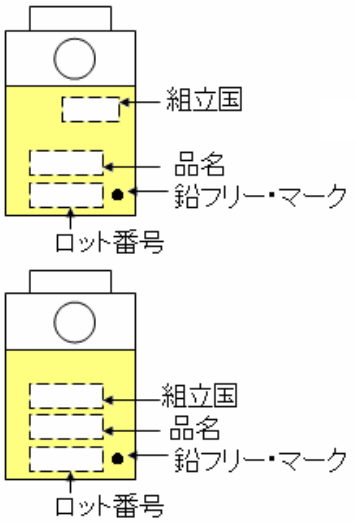
Q1. 現品の捺印表示を教えてください。

A1. パッケージごとの表示内容を以下に示します。

(1/2)

パッケージ	捺印表示	備　　考
TO-92		ロット番号の構成 □ □□ □ 社内管理記号 週コード(数字2桁) 西暦末尾(数字1桁)
TO-126 (MP-5)		ロット番号の構成 □ □□ □ 社内管理記号 週コード(数字2桁) 西暦末尾(数字1桁)

(2/2)

パッケージ	捺印表示	備　　考
TO-252 (MP-3ZK)		ロット番号の構成 □ □□ □ 社内管理記号 週コード(数字2桁) 西暦末尾(数字1桁)
SOT-89 (Power Mini Mold)		ロット番号の構成 □ □ 月コード(英数字1文字) 1~9月 → 1~9 10月 → X 11月 → Y 12月 → Z 西暦末尾(数字1桁) 〔関連 FAQ〕 捺印品名
SC-63 (MP-3Z)		ロット番号の構成 □ □□ □ 社内管理記号 週コード(数字2桁) 西暦末尾(数字1桁)
SC-64 (MP-3)		ロット番号の構成 □ □□ □ 社内管理記号 週コード(数字2桁) 西暦末尾(数字1桁)
SC-74A (Mini Mold)		〔関連 FAQ〕 捺印品名
8 ピン SOP (225mil)		ロット番号の構成 □ □□ 週コード(数字2桁) 西暦末尾(数字1桁)
MP-45G		ロット番号の構成 □□ □□ □ 社内管理記号 週コード(数字2桁) 西暦末尾(数字2桁)

パッケージ

Q1. パッケージ記号の H と HF の違いはなんですか。

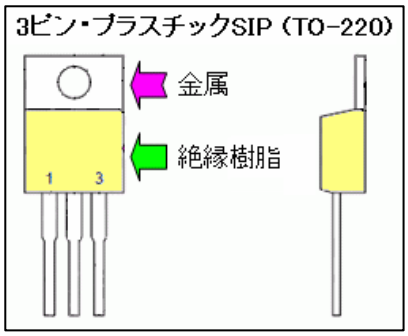
A1. シリーズ・レギュレータを例に説明します。品名に H や HF がつくものには、次の製品があります。

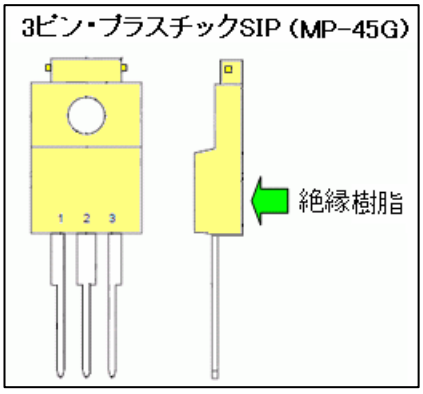
μPC79M12H （廃止品）⇒ パッケージ名は TO-220

μPC79M12HF ⇒ パッケージ名は MP-45G

なお、TO-220 は JEDEC の呼称で、MP-45G は旧 NEC エレクトロニクスの愛称です。

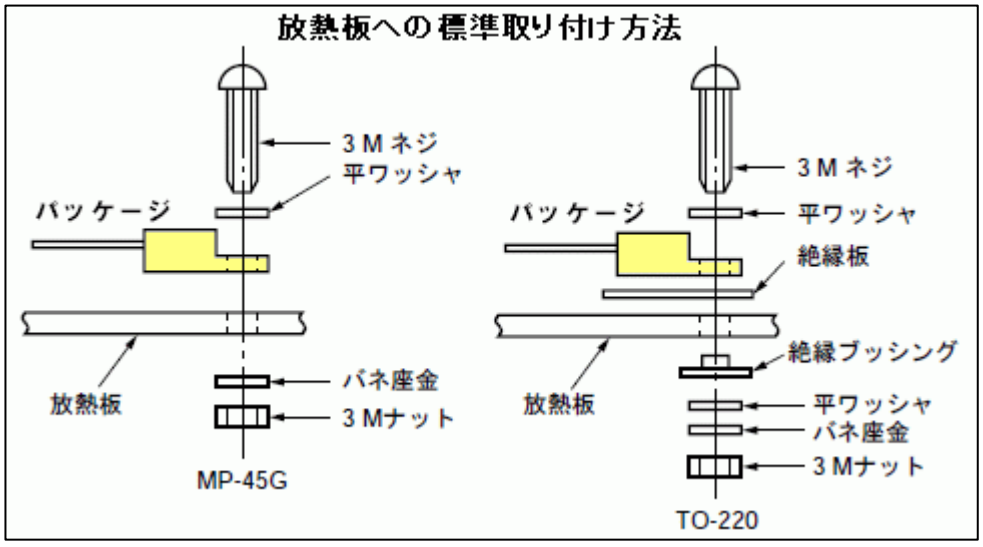
F はフルモールドを示し、TO-220 の金属部分が、MP-45G では樹脂被覆で絶縁されています。





■フルモールド(MP-45)の金属露出(TO-220)に対するメリット

フルモールド製品のメリットは、絶縁樹脂被覆が絶縁板の代替となります。また、絶縁ブッシングも必要ないので、取付け工数の削減にもなります。



7. ディスクリート

7.1 共　通

外形図

Q1. 3 ピン・パワー・ミニ・モールド(SC-62)の外形図は、Top view なのか Bottom view なのか分かりません。

A1. 中央のコレクタやソースの端子形状が分かるように、Bottom view で記載しています。

Q2. 4 ピンと 6 ピンのパッケージで、1ピンの位置が分かりません。

A2. パッケージを捺印が読める向きにして、左下が1ピンです。
なお、RF/マイクロ波デバイスは製品によって位置が異なります。各製品のデータ・シートをご参照ください。

Q3. SC-62 と SOT-89 は同じものですか。

A3. SC-62 は EIAJ（日本電子機械工業会）、SOT-89 は JEDEC（電子部品の標準化を推進する米国業界団体）の呼称です。呼称は異なりますが、形状に違いはありません。

動作温度範囲

Q1. トランジスタやダイオードの動作温度範囲を教えてください。

A1. 集積回路製品は回路が機能しなければ用を成さないため、(回路)動作温度範囲を規定しています。
トランジスタやダイオードなどディスクリート品は、回路機能がないので、動作温度範囲を規定していません。
保存温度範囲内でご使用ください。下限は保存温度範囲ですが、上限は接合部温度で制限されます。
たとえばトランジスタなどの絶対最大定格で接合部温度は 150℃です。接合部温度が 150℃を越えないよう、ディレーティングしてご使用ください。

絶縁耐圧

Q1. TO-220 などのパッケージは、放熱フィンなどに装着して使用することがあると思います。その際、放熱フィンに電圧が印加される状況があるため、絶縁耐圧を教えてください。

A1. TO-220 などの絶縁耐圧は測定していません。お手数ですが、販売店経由でご要求願います。

付加記号

Q1. ラベルに記載の型名 2SD1701-AZ/JM の末尾「JM」の意味を教えてください。

A1. 「JM」は生産国（拡散国、組立国）を表します。これも標準品であり、形状や特性に違いはありません。
1 けた目 ： 拡散国（J＝日本）　　2 けた目 ： 組立国（M＝マレーシア）

Q2. 2SD1694 の「K」規格を購入しましたが、「KM」と表示されています。

A2. 「KM」という表示品は、K 規格品です。「M」は社内管理記号で、情報公開していません。

Q3. 3ピン・ミニモールド(SC-59)品のオーダ品名で、「L」はなにを表す記号ですか。

A3. 「L」はマガジン・ケース包装を表します。この製品では、テーピング品の「T1B」や「T2B」などもあります。
オーダ品名には、包装形態の記号も含みます。
なお、RF/マイクロ波デバイスではマガジン・ケース包装でなく、テーピング品をデバイス50個ごとに切断したものになっています。

Q4. トランジスタやFETなどのオーダ品名に、(0)が付いたものがあります。これはなにを表す記号ですか。

A4. (0)は電装対応を表す記号です。自動車用途に対応した高品質、高信頼性の製品です。

Q5. ディスクリート製品の電装仕様品と一般品は、なにが違うのですか。

A5. 電装仕様品は、一般品の製造管理の上に、製造設備や工程の管理を特別に定め、信頼性水準を一般品より高く設定し、高品質、高信頼性の製品を提供するものです。

★

捺　印

Q1. パッケージ捺印はどのようになっていますか。

A1. 基本的には、品名、規格(区分がある場合だけ)、ロット番号を表記しますが、捺印面の制約で省略や記号化をしているものがあります。表記方法は、パッケージで異なる場合があります(ロット番号の数字と2進ドットなど)。

	トランジスタ (バイポーラ、FET)	ツェナー・ダイオード[注2]	サイリスタ/トライアック
表面実装	**品名**(記号:バイポーラは h_{FE} で細分類[注1]) **ロット番号**(数字かドット)	**品名**(省略形) **規格**(RDの一部:数字(1など)かドット) **カソード・マーク**(2ピンだけ)	**品名**(記号(SOT-89)か省略形(MP-3Z)) **規格**(P)、**ロット番号**(数字)
端子挿入	**品名**:記号("2S"は省略) **規格**(h_{FE}区分)、**ロット番号**(数字)	**品名**(省略形) **規格**(B1など) **カソード・マーク**	**品名**(省略形) **規格**(P)、**ロット番号**(数字)

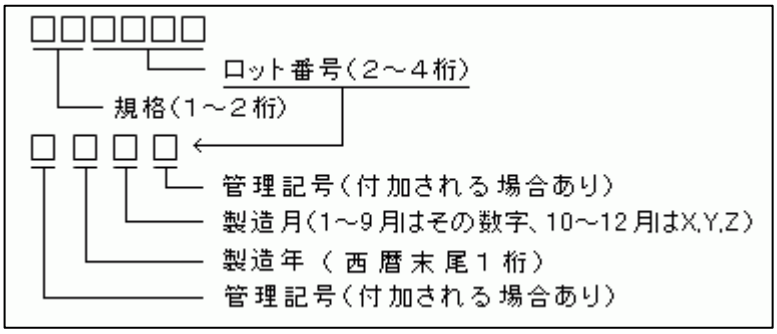
注 1. 2SC1623の例

捺印品名	L4	L5	L6	L7
h_{FE}	90～180	135～270	200～400	300～600

表面実装トランジスタの捺印品名は、データ・シートに記載されています。

2. 端子挿入型はガラス・パッケージです。

次図は、端子挿入パッケージのトランジスタとサイリスタ/トライアックでの、規格とロット番号の表記です。



管理記号の内容は公開していません(表面実装パッケージでは表記なし)。

ツェナー・ダイオードについては、製品ごとの捺印情報をご参照ください。

電力定格

Q1. パワー・トランジスタやパワーMOSFET の周囲温度−全損失の特性グラフに記載されている「無限大放熱板付」というのは、どういう意味ですか。

A1. 無限大放熱板付≒Tc＝25℃とみなします。つまり、きわめて大きな放熱器をつけて、接合部の熱を外部に放散させ、ケース温度を 25℃にした状態です。

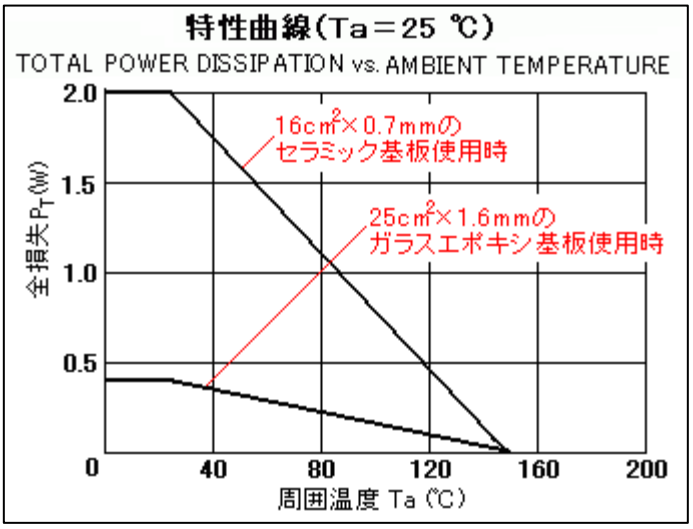
Q2. パワー・トランジスタやパワーMOSFET の電力定格である全損失は、T_A＝25℃とTc＝25℃の 2 通りの条件で規定していますが、どのような違いがありますか。

A2. T_A＝25℃での規定は、周囲(Ambient)温度 25℃の雰囲気中での、素子単体の最大電力消費量を表します。この場合、熱発生源から周囲空間までの熱抵抗は、Rth(j−a)で表します。Tc＝25℃での規定は、素子(Case)そのものを強制冷却して、パッケージ表面を 25℃とした場合に、許容できる最大電力消費量を表します。「無限大放熱板付」と表記してある場合もありますが、実使用ではパッケージ表面温度を 25℃にすることは困難で、ディレーティングも考慮すると、許容電力は T_A＝25℃と Tc＝25℃の条件値の中間程度となります。

関連項目：熱設計

Q3. 2SD1000(SC−62)は、セラミック基板実装時の全損失として 2W と規定されていますが、ガラスエポキシ基板に実装した場合はどのようになりますか。

★ **A3.** この全損失 2W は、そのときのジャンクション－周囲間熱抵抗(62.5℃/W)から規定されています。ガラスエポキシ基板実装時のジャンクション－周囲間熱抵抗は約 310℃/W で、これを基に計算すると、周囲 25℃のときは 0.8W(＝(150℃－25℃)/310℃/W)程度となります。全損失−周囲温度の特性は、データ・シートの P_T−T_A グラフに、25℃で 0.4W と 150℃で 0W を線で結んでご参照ください。



Q4. SC−62 製品の絶対最大定格全損失の条件に、セラミック基板使用時とありますが、どのようなものですか。

A4. セラミック基板とは、陶磁器などの材料を板状にし、高温で焼き固めたものです。ガラス・エポキシ基板に比べ放熱性に優れているため、比較的パワーの大きな IC などに使用されます。

選定のための確認事項

Q1. トランジスタや MOSFET を選定する際、どのような確認事項がありますか。

A1. 選定のための最低限の確認事項として、次のようなものがあります。

V_{CEO}、V_{DSS} 定格	→	電源電圧＋マージン
I_C、I_D 定格	→	負荷電流＋マージン
パッケージ	→	サイズや実装方法
P_T 定格	→	使用時の最大周囲温度
h_{FE},ゲート駆動電圧	→	回路設計による h_{FE} やゲート駆動電圧
V_{CE}(sat),R_{DS}(on)	→	消費電力に影響
NF,Ga,f_T	→	高周波特性に影響

パッケージ

Q1. TO-220 の表面実装品で、TO-220SMD(MP-25Z)と TO-263(MP-25K,MP-25ZP)はどのように違いますか。

A1. 外形上は大差ありません。MP-25ZP は、大電流(110A)を保証したものです。
TO-220SMD(MP-25Z)は、JEITA(電子情報技術産業協会)の標準パッケージで、日本国内向けです。
TO-263(MP-25K, MP-25ZP)は、JEDEC(アメリカの電子部品業界団体)の標準パッケージです。

Q2. 古いトランジスタで、樹脂が緑色のものがありますが、過去にこのようなものを製造していしましたか。

A2. 1994年12月までは、PNPタイプに緑色の樹脂を使用していました。
したがって、PNPタイプの1994年12月までのロットでは、緑色の樹脂のものが存在します。
1995年1月以降は、トランジスタ全般を黒色へ統一しています。

Q4. ディスクリートのパッケージでルネサスエレクトロニクス独自の呼称はありますか。

A4. パワー系パッケージの MP-**は、当社独自の呼称です。

締め付けトルク

Q1. パワー・トランジスタを放熱板に取り付ける際の、ネジ締めトルクの許容値を教えてください。

A1. 「パワートランジスタの取り付け方法と取り付け部品一覧表」の表-1 で、パッケージごとにご参照ください。
なお、RF/マイクロ波デバイスについては、各製品のデータ・シートでの記載をご参照ください。

熱抵抗

Q1. データ・シートに熱抵抗の記載がありません。どのように算出するのですか。

A1. 熱抵抗は、データ・シートの全損失(P_T)を基に、次の式で算出できます。

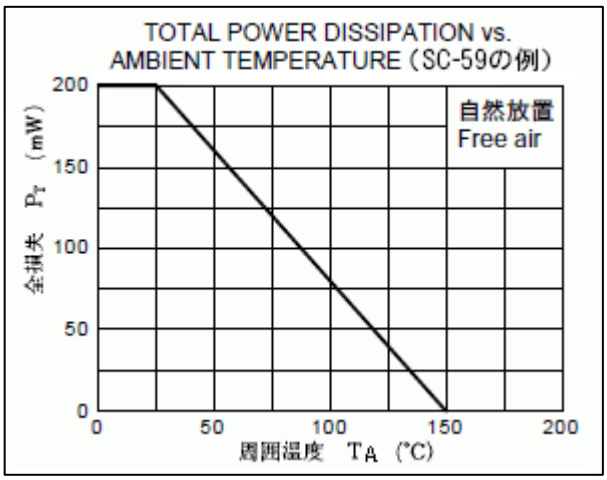
- ジャンクション・周囲間　：$R_{th(j-a)}=(T_{jmax}-T_a)/P_T$
- ジャンクション・ケース間：$R_{th(j-c)}=(T_{jmax}-T_c)/P_T$

例として 2SD1588 の場合は、次のようになります。

- $R_{th(j-a)}=(150[℃]-25[℃])/2[W]=62.5[℃/W]$
- $R_{th(j-c)}=(150[℃]-25[℃])/30[W]=4.17[℃/W]$

Q2. SC-59（ミニモールド）や SC-62（パワー・ミニモールド）のジャンクション－ケース間熱抵抗を教えてください。

A2. いずれもジャンクション－ケース間熱抵抗は規定していません（Tc＝25℃での全損失を規定していません）。
データ・シートに記載の「全損失−周囲温度（T_A）」の特性グラフで、使用の可否をご判断ください。



SC-62 のガラスエポキシ基板実装時における「全損失−周囲温度(T_A)」の特性グラフは、電力定格の A-4 をご参照ください。

小信号系のリード挿入型製品

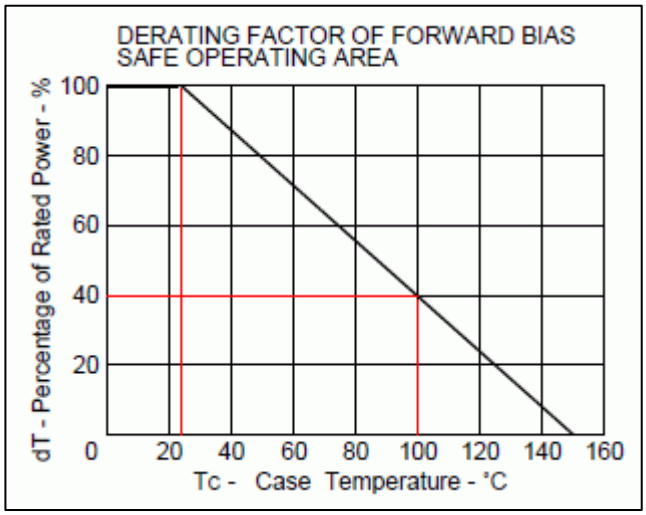
Q1. 2SC945A は廃止品ではないが、新規受注をしていないとのことですが、相当品はありますか。

A1. 小信号系のリード挿入型製品はすべて集約方向にあり、新規セットへのご採用は控えていただいています。
したがって、相当品は表面実装型製品の 2SC1623 や 2SC4177 などでご検討ください。

ディレーティング

Q1. 動作温度範囲の FAQ で「接合部温度が 150℃を越えないよう、ディレーティングしてご使用ください」と記述されていますが、このディレーティングとは具体的にどのようなことをするのですか。

A1. 次図でたとえばケース温度が 100℃なら、安全動作領域は 25℃のときの消費電力の 40%となります。ディレーティングとは、このように素子の周囲温度と自己発熱による温度上昇を基に、使用条件を制限することです。この例で 40%以上の領域で使用すると、接合部温度が絶対最大定格の 150℃を越えることになります。



Q2. 周囲温度 55℃のとき、2SC4553 のジャンクション温度を 150℃以下に抑えるには、ケース温度を何℃にする必要がありますか。

A2. ケース温度でジャンクション温度を推測する場合、ジャンクション−ケース間熱抵抗を使用します。
2SC4553 のジャンクション−ケース間熱抵抗は、4.2℃/Wです。
たとえば、2SC4553 を全損失 10Wで使用する場合、ジャンクションとケースの温度差は、熱抵抗 4.2℃/Wより 4.2℃×10W＝42℃となります。つまり、10Wで使用した場合のジャンクション−ケース間の温度差は 42℃なので、ケース温度は 150℃−42℃＝108℃以下としなければなりません。
以上のように、ケース温度測定でジャンクション温度の推測をするので、周囲温度と直接関係しません。
ただし、周囲温度が上昇すれば、その分ケースの温度も上昇するので、ジャンクション温度を 150℃以下に抑えるには、全損失を下げてジャンクション温度が 150℃を越えないようにしなければなりません。
関連項目：熱設計

パッケージ温度

Q1. フィン付きのパワー・パッケージのケース温度は、どの部分で測定すればいいですか。

A1. 簡便な方法としては、チップに近いフィンと樹脂の界面に、熱電対を付けて測定します。

絶対最大定格略号の見方

Q1. トランジスタや FET の絶対最大定格の略号の見方を教えてください。

A1. $V_{①②③}$として、次の意味です。
① 被測定端子
② 基準となる端子
③「②の基準端子」と「①と②以外の端子」との接続状態
S：Short（短絡）、**O**：Open（開放）、**X**：指定バイアス印加、**R**：指定直流抵抗接続

7.2 トランジスタ

熱設計

Q1. 2SD1899-Z(MP-3)の全損失が 1.0W のとき、ケース表面が 80℃なら、ジャンクション温度は何度ですか。

A1. 2SD1899-Z(MP-3)のジャンクション−ケース間熱抵抗 Rth(j-c)は、「ディスクリート＞共通＞熱抵抗」で示す算出式より 12.5℃/W です。したがって、ケース温度 80℃、全損失 1.0W のときのジャンクション温度は、次の計算式で求められます。

Tj＝80[℃]＋12.5[℃/W]×1.0[W]＝92.5[℃]

温度特性

Q1. V_{BE} の温度特性を教えてください。

A1. V_{BE} の温度特性は、PN ジャンクションの温度係数(−2mV/℃)で決まります。

Q2. コレクタ遮断電流 I_{CBO} の温度による変化量は、どの程度になりますか。

A2. I_{CBO} の温度特性は、10℃上昇するごとに、ほぼ 2 倍の変化を示します。

Q3. h_{FE} の温度特性を教えてください。

A3. h_{FE} の温度特性は、ほぼ 0.5～1.5%/℃です。回路設計の際には、特に低温での h_{FE} 減少を考慮してください。

捺　　印

Q1. 「L4」と捺印されている製品はなんですか。

A1. 2SC1623 です。表面実装品では、品名を h_{FE} ランクで分類した記号を現品に表示しています。

例 2SC1623 の h_{FE} ランク

捺印	L4	L5	L6	L7
h_{FE}	90～180	135～270	200～400	300～600

ベース電流の最大定格

Q1. 小信号用トランジスタの絶対最大定格では、ベース電流を規定していませんが、どのくらい流せますか。

A1. I_B の最大定格は、一般に $I_B \leqq (1/10)Ic$ とみなします。コレクタ電流の絶対最大定格を元に、ご確認ください。

表面実装用マウント・パッド

Q1. 表面実装用トランジスタの推奨マウント・パッドはありますか。

A1. 下記Webページで、該当パッケージ名のマウント・パッドをクリックしてご参照ください。

http://www2.renesas.com/discrete/ja/package/smd.html

なお、RF/マイクロ波デバイスについては、下記ページに掲載の「高周波およびマイクロ波デバイス・パッケージ・ラインナップ」をご参照ください。

http://www2.renesas.com/microwave/ja/document.html#pkg

CAD データ：http://www2.renesas.com/microwave/ja/mountpad.html

コンプリメンタリ使用

Q1. コンプリメンタリ使用を保証する素子としての要件と、その保証値や許容値を教えてください。

A1. コンプリメンタリを推奨していても、特に保証項目は決めていません。一般に要求されるコンプリメンタリ特性は、直流電流増幅率(h_{FE})の電流特性です。その他、設計される回路から、要求項目は異なると考えます。

周波数特性

Q1. トランジスタの特性で、コレクタ容量(C_{OB})やトランジション周波数(f_T)は、h_{FE} ランクと相関があるのですか。

A1. 一般に、h_{FE} と C_{OB} の相関はありません。h_{FE} が下がれば電流利得が下がるので、f_T は低くなります。

関連項目：トランジション周波数

スイッチング特性

Q1. 増幅用トランジスタでは、ターンオン時間とターンオフ時間はどのくらいですか。

A1. 増幅用のトランジスタはスイッチング用ではないので、ターンオン時間とターンオフ時間を規定していません。
スイッチング特性が必要な場合は、スイッチング用のトランジスタをご使用ください。

V_{CEO} 逆電圧

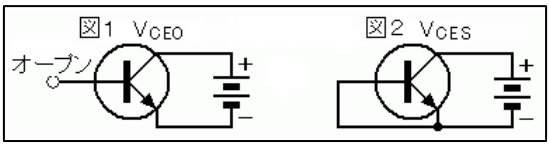
Q1. 2SC1623 で V_{CEO} の逆電圧の耐圧は、何 V ですか。

A1. V_{CEO}の逆方向電圧は規定していませんが、目安として絶対最大定格Vebo程度となります。
2SC1623 の場合は約 5V(エミッタ基準のコレクタ電圧は−5V)です。

V_{CEO} と V_{CES}

Q1. 2SC4176 の絶対最大定格で V_{CES}＝40V、V_{CEO}＝15V とありますが、どう違うのですか。

A1. ベース端子の状態で規定しています。V_{CEO}(図 1)はベース・オープン、V_{CES}(図 2)はベース−エミッタ間ショートでのコレクタ−エミッタ間の定格です。V_{CEO}＜V_{CES} の関係で、低い方の V_{CEO} を守れば安全です。



周波数による分類

Q1.低周波増幅用と高周波増幅用の周波数範囲は具体的にどのようになりますか。

A1. 厳密な区分はありませんが、低周波用素子とは一般に音声や楽器などの数十 Hz から数十 kHz 程度の周波数領域の増幅用を言います。高周波増幅用は数十 kHz から数百 MHz 程度の信号増幅を想定しています。
ちなみに、1GHz 以上では超高周波増幅用と呼ばれます。

h_{FE} 劣化

Q1. 長時間使用中に h_{FE} の劣化が起きることがあると聞きました。その要因と対策を教えてください。

A1. トランジスタのエミッタ・ベース間に、逆バイアス・ブレークダウン電圧(エミッタ・ベース間電圧 V_{EBO})近くか、それ以上の電圧が瞬時的にも繰り返し印加される回路では、長時間使用中に h_{FE} 劣化をもたらします。回路設計では、このエミッタ・ベース間逆バイアス・ブレークダウンを起こさない設計が必要です。

静電気耐量

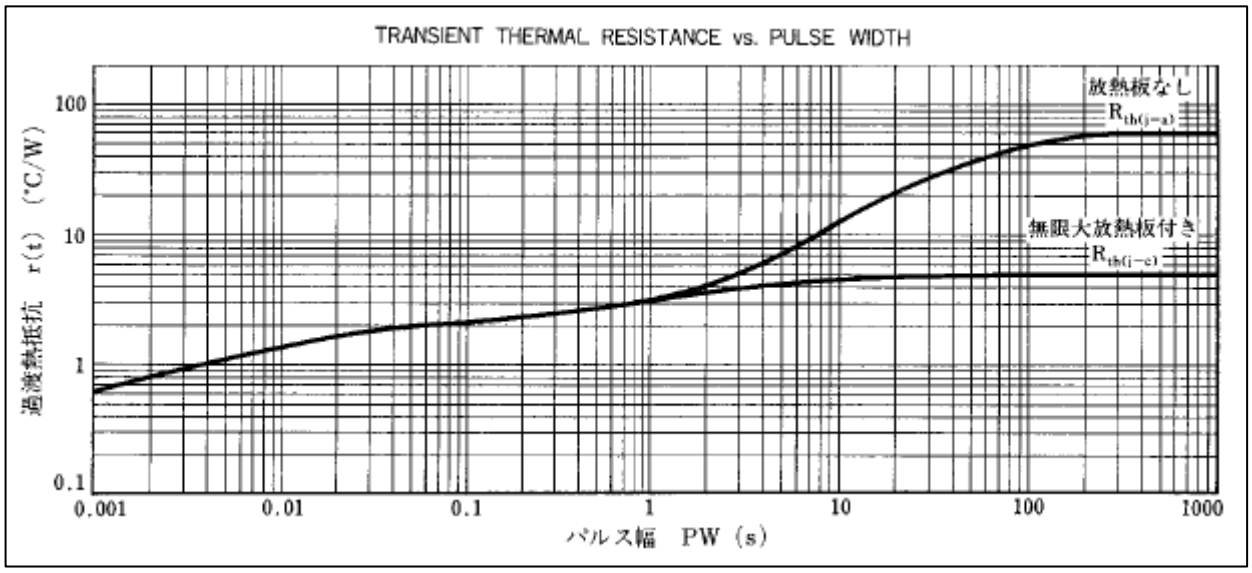
Q1. バイポーラ・トランジスタの静電気耐量(人体モデル HBM、機械モデル MM)を教えてください。

A1. トランジスタの静電気耐量は規定していません。
必要な場合は、販売店に「品質/信頼性試験報告書」をご要求ください。
HBM、MM については、半導体　品質/信頼性ハンドブックの「6.4 ESD に対する注意事項」をご参照ください。

過渡熱抵抗特性グラフ

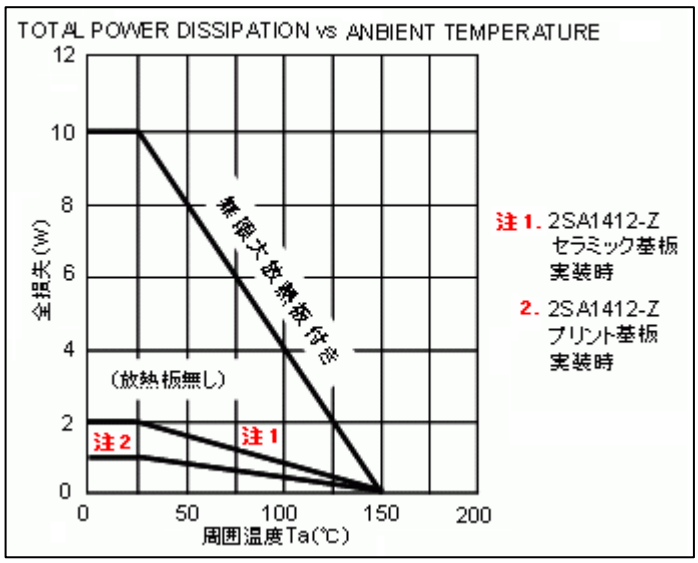
Q1. 過渡熱抵抗対パルス幅の特性グラフで、「単体（放熱板なし）」と「無限大放熱板付き」のちがいは何ですか。

A1. 「単体」はジャンクション−周囲間の過渡熱抵抗で、「無限大放熱板」はジャンクション−ケース間の過渡熱抵抗です。特性曲線の右端の値が、定常熱抵抗値になります。



放熱板

Q1. 2SA1412 のデータ・シートに、放熱板付きの損失特性がありますが、取り付け可能な放熱板の型名、取り付け方法などを教えてください。



A1. 放熱板の型名、取り付け方法などは、当社で指定していません。ご使用条件を基に、放熱板メーカにご相談ください。なお、下記技術資料に MP−3 パッケージの放熱フィン取り付け例や注意事項が記載されています。

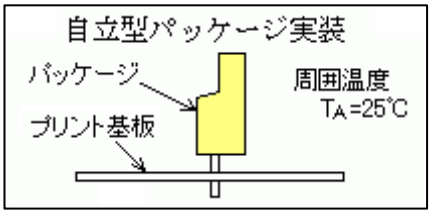
技術資料：表面実装用 MP−3 形パワーデバイス(D17347JJ4V0AN00)

関連項目：電力定格

放熱板なしの実装状態

Q1. 2SA1396 の過渡熱抵抗グラフの全損失の条件で、「T_A＝25℃ 放熱板なし」はどのような実装状態ですか。

A1. 「T_A＝25℃ 放熱板なし」は、図のように基板に自立（縦差し）実装し、放熱板を付けない状態での規定です。
なおこの場合、基板への放熱は実質的に少ないため、基板サイズは規定していません。



コレクタ電流の最大定格

Q1. 2SA1611、2SC4177 の絶対最大定格で、コレクタ電流 Ic＝100mA とありますが、これは直流電流とパルス電流のどちらにも当てはまりますか。

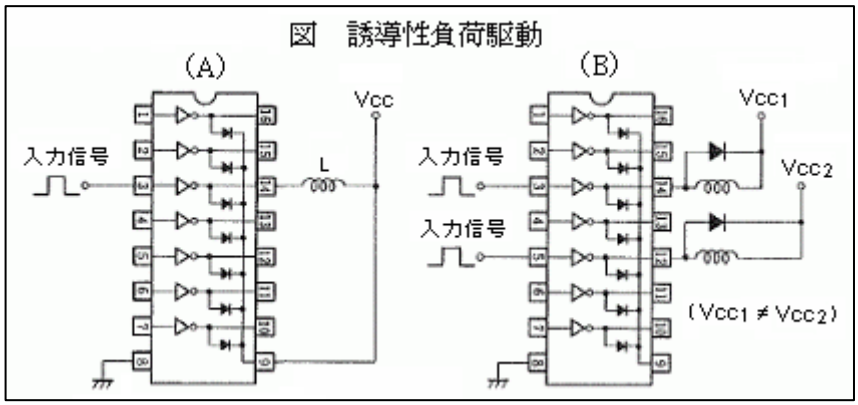
A1. これは直流電流での定格です。パルスのコレクタ電流についても、この定格内でご使用ください。

L負荷時の外付けダイオード

Q1. μPA2000 シリーズで、出力側にリレーや電磁弁を接続する場合、そのコイル両端にサージ吸収用のダイオードを付けた方がいいですか。

A1. μPA2000 シリーズには、次図(A)のようにサージ吸収用ダイオードが内蔵されていますので、外部には基本的に必要ありません。ただし、内蔵ダイオードの絶対最大定格を越えるサージが想定される場合は、損失容量の大きいサージ吸収用ダイオードの外付けが必要です。
また、図(B)のように、電源電圧が異なる場合や、一定時間に渡って回生電流が生じるパルス・モータ駆動など（内蔵ダイオードはスパイク電圧吸収が目的ではない）では、外付けダイオードが必要となります。

図　誘導性負荷駆動



同一パッケージで全損失が異なる

Q1. 2SB1430、2SB1431、2SB1432 などは同一パッケージ(TO-220)なのに、なぜ全損失 P_T(T_C＝25℃)が 20、25、30W と違うのですか。また、全損失 P_T(T_A＝25℃)はなぜ同じ 2.0W なのですか。

A1. チップ・サイズの違いによります。
T_C＝25℃の場合、チップ・サイズの大きい方が放熱しやすく、全損失は大きくなります。
T_A＝25℃では、ほとんどパッケージから周囲への放熱で、チップ・サイズは全損失にほとんど影響しません。

抵抗内蔵トランジスタのベース電流値

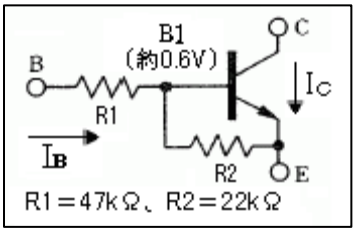
Q1. FA1L4L のベース端子に 4V を印加時のベース電流値は、どのように考えればいいですか。

A1. ベース電流 I_B はベース端子 B に流れる電流で、内部トランジスタのベース B1−エミッタ E 間に流れる電流は算出できません。一般にトランジスタのベース−エミッタ間の直流ベース電圧 V_{BE} は約 0.6V なので、次のような値になります。

ベース抵抗 R1(TYP.47kΩ)より、$(V_B-V_{B1})/R1=(4V-0.6V)/47k\Omega=72\mu A$ 程度となります。

なお、B1−エミッタ間接続の抵抗 R2(TYP.22kΩ)に流れる電流は、$V_{B1}/R2=0.6V/22k\Omega=27\mu A$ 程度です。

ちなみに、コレクタ電流 I_C は、抵抗内蔵であってもベース電流 I_B の h_{FE} 倍流れることになります。



動作領域

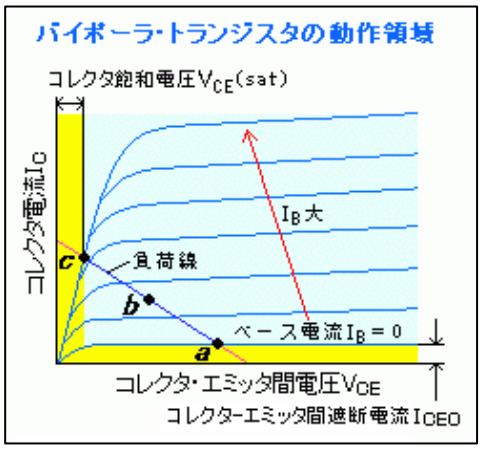
Q1. トランジスタの動作領域である飽和領域、活性領域、遮断領域とは、それぞれどのような動作状態ですか。

A1. ［飽和領域］

ベース電流を大きくしていくと、負荷抵抗の両端電圧（V_{RI}=Ic×RI）は大きくなります。右図でいえば負荷線上 a→b へ移動します。電源電圧が一定であれば、ベース電流をいくら増やしても、負荷線上 c から先はコレクタ電流が増えません。コレクタ・エミッタ間の電圧は最小となります。

この最小電圧をコレクタ飽和電圧 V_{CE}(sat)といいます。

この領域は、スイッチ動作としてのオン状態です。



［活性領域］

図の薄いブルーの領域で、負荷線上 b を中心に適切なベース電流である入力信号の変化を、大きなコレクタ電流の変化として、ひずみなく増幅することができる領域です。

［遮断領域］

ベース端子をオープンにしても、コレクタ・エミッタ間にはコレクタ遮断電流 I_{CEO} が流れます。この領域を遮断領域（コレクタ遮断領域）といいます。この領域は、スイッチ動作のオフ状態です。

関連項目：コレクタ飽和電圧

差動増幅用

Q1. μPA604T は NPN トランジスタが 2 回路内蔵されていますが、この製品は差動増幅に使用できますか。

A1. 差動増幅用は電気的、熱的に平衡でなければなりませんが、μPA604T を含め、当社に差動増幅用に設計した製品はありません。

7.3 FET

ゲート抵抗

Q1. MOSFET のゲートに抵抗を実装した図が、アプリケーション・ノートに載っていますが、なぜ抵抗が必要なのですか。また、抵抗値はどのように決めるのですか。

A1. FET のゲート抵抗には、次のような役割があります。

- 寄生振動の防止
- ゲート部のサージ保護
- スイッチング速度の抑制
- 前段の電流ドライブ能力との整合

主目的は寄生振動(共振)の防止です。寄生振動は、寄生容量と寄生インダクタンスによる共振回路の Q が大きいために発生します。

そこで、ゲートに数 10～数 100Ω の抵抗を付加することで Q を低下させ、発振やリンギングを減少させます。

なお、サージ保護のためには、抵抗をさらに大きくしますが、これにより、動作速度は著しく低下します。

お客様の仕様によって、抵抗値をご決定ください。

ゲート保護

Q1. FET のゲート保護は必要ですか。

A1. FET のゲート-ソース間に内蔵されている保護ダイオードは、取り扱い時の静電破壊の保護のためです。

回路動作で定格電圧を越える恐れがある場合は、定電圧ダイオードなどでゲート保護をしてください。

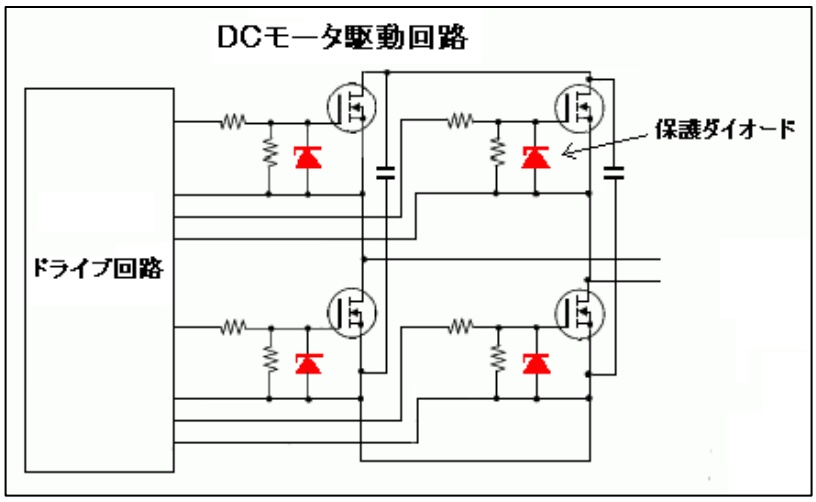
注意 RF/マイクロ波デバイスでは、ゲート保護ダイオードを付加すると、特性劣化を生じることがあります。

2SK1288 の保護ダイオード

Q1. パワーMOSFET 2SK1288 の静電破壊保護で、外付け定電圧ダイオードの使用例はありますか。

A1. 2SK1288 での使用例ではありませんが、一般的な応用事例を次図に示します。

2SK1288 の絶対最大定格 V_{GSS} は 20V なので、ゲート保護用定電圧ダイオードは V_{GS} が最大 16V(絶対最大定格の 80%を想定、推奨は 10V 程度)以下となるよう選定してください。



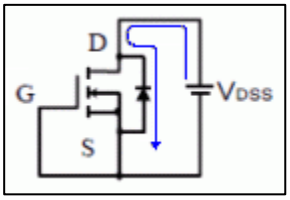
遮断電流値の規定

Q1. I_{DSS} や I_{GSS} などの遮断電流は、どのような規定の電流値ですか。

A1. MOSFET がゲート・オフ状態のときの、各端子間の漏れ電流です。

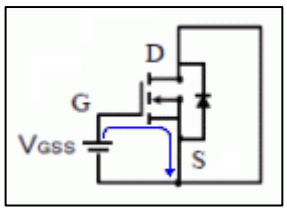
I_{DSS} はドレーン・ソース間遮断電流で、V_{GS}＝0 のときのドレーン・ソース間の漏れ電流値です。

ドレーン・ソース間に最大定格 V_{DSS} を印加して規定します。



I_{GSS} はゲート・ソース間遮断電流で、V_{DS}＝0 のときのゲート・ソース間の漏れ電流値です。

ゲート・ソース間に最大定格 V_{GSS} を印加して規定します。



寄生ダイオード

Q1. ドレーン-ソース間の内部ダイオードの最大定格を教えてください。

A1. 内部ダイオード単体での規定はしていません。順電流と耐圧は、絶対最大定格に記載のとおりです。

Q2. ドレーン-ソース間の寄生ダイオードをフライホイール・ダイオードとして利用できますか。

A2. 寄生ダイオードは逆回復時間が長く、高速制御には適しませんので、このような使用はしないでください。

絶対最大定格 V_{GS}

Q1. 2SK3062 などのゲート-ソース間電圧の絶対最大定格で、AC と DC の規格が違うのはなぜですか。

AC：±20V、DC：＋20V,－10V

A1. AC の場合、ゲートは電荷の充放電を繰り返します。このため、ゲートへのストレスは、DC より少なくなります。

このようなことから、AC と DC で区分して決めています。

全損失

Q1. 全損失 P_T が、「ドレーン-ソース間電圧 V_{DSS}」と「ドレーン電流 I_D」の積よりかなり小さいのはなぜですか。

A1. 「ドレーン-ソース間電圧 V_{DSS}」は OFF 時（V_{GS}＝0V）の耐圧であって、ON 時のドライブ電圧ではありません。

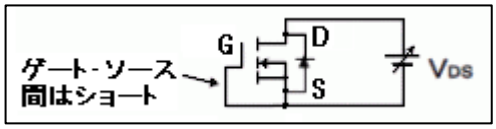
オン抵抗が小さいので、大電流が流れることになり、そのような高電圧でのドライブはできません。「ドレイン電流 I_D」が絶対最大定格を越える場合は、制限抵抗などが必要となります。

詳細は「パワーMOS FET の安全動作領域について」をご参照ください。

電圧定格の定義

Q1. V_{DSS} や V_{GSS} の定義を教えてください。

A1. V_{DSS} はドレーン・ソース間に加えることができる最大電圧値で、ゲート・ソース間はショート状態です。



V_{GSS} はゲート・ソース間に加えることができる最大電圧値で、ドレーン・ソース間はショート状態です。



略号の見方は、絶対最大定格略号の見方をご参照ください。

熱抵抗

Q1. 2SK3061 のジャンクションからドレーン端子までの熱抵抗値を教えてください。

A1. 端子での定義はしていません。ジャンクション−ケース間の熱抵抗 Rth(ch−c)なら、次の式で求めます。

$Rth(ch-c)=(Tch - Tc)/P_T$

2SK3061 の場合：$Rth(ch-c)=(150℃-25℃)/35W=3.57℃/W$

Q2. μPA1764 など 8 ピン SOP で、過渡熱抵抗の $Rth(ch-_A)$を規定していますが、Rth(ch−c)の値はありますか。

A2. チャネル−ケース間熱抵抗 Rth(ch−c)は、放熱器に取り付けて使用する構造のパッケージで規定しています。8 ピン SOP は、放熱器に取り付ける構造ではないので、Rth(ch−c)は規定していません。

放熱フィンの配線

Q1. μPA2700TP(8 ピン SOP)の裏面放熱フィンはどこかにつながっていますか。

A1. データ・シートの外形図に記載しているように、ドレーンにつながっています。

オン電圧

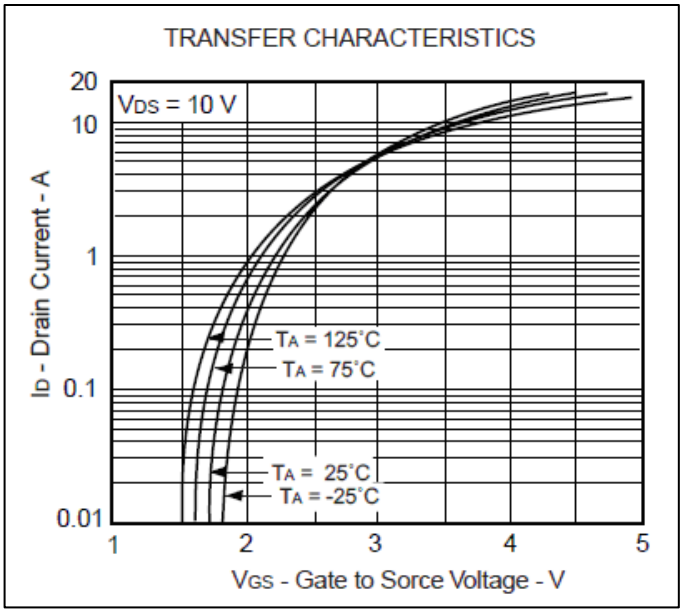
Q1. FET の D−S 間がオンする電圧は、ゲート・カットオフ電圧($V_{GS(off)}$)と同じですか。

★ A1. ゲート・カットオフ電圧は、次の特性表にもあるような条件をオフと定め、その時点のゲート・ソース間電圧です(I_D が条件値より少なくなる VGS ではオフ)。

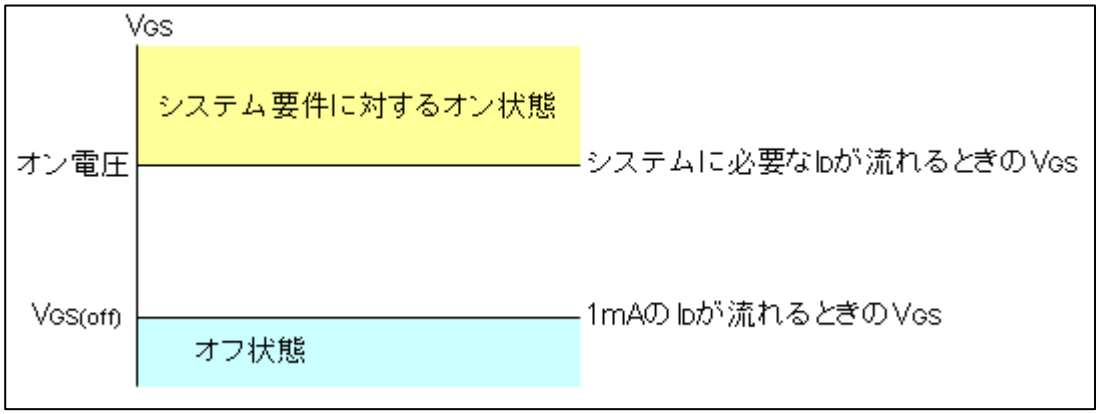
電気的特性（T_A = 25°C）

項　目	略号	条　件	MIN.	TYP.	MAX.	単位
ドレインしゃ断電流	I_{DSS}	V_{DS} = 60 V, V_{GS} = 0 V			10	μA
ゲート漏れ電流	I_{GSS}	V_{GS} = ±20 V, V_{DS} = 0 V			±10	μA
ゲート・カットオフ電圧	$V_{GS(off)}$	V_{DS} = 10 V, I_D = 1 mA	1.0	1.4	2.0	V
順伝達アドミタンス	$\|y_{fs}\|$	V_{DS} = 10 V, I_D = 1 A	1			S

一方、オン電圧は、お客様がどの程度のドレイン電流(I_D)をオンとするかによって、伝達特性(グラフ参照)で決まるゲート・ソース間電圧です(I_D が設定値より多くなる VGS ではオン)。



オン電圧の条件として、システムで必要なドレイン電流が、ゲート・カットオフ電圧の条件と一致すれば、両者は同じ電圧です。通常は一致しない(オン状態は大電流)ので違う値になります。



安全動作領域

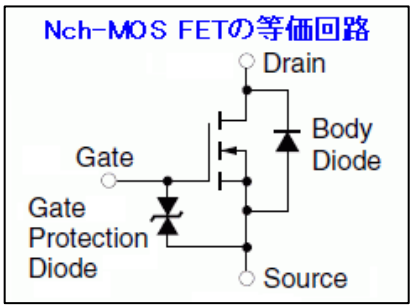
Q1. パワーMOS FET の順方向安全動作領域のグラフで、$R_{DS(on)}$Limited の点線はどのような制限ですか。

A1. $R_{DS(on)}$Limited の点線は、オン抵抗の制限による特性の限界線で、ここを越えることはあり得ません。
つまり、V_{DS} が低くなると、オン抵抗による I_D の制限が大きくなります。

逆方向特性

Q1. FET でソース-ドレイン間に逆方向の電流をながしても、同じスイッチング特性になりますか。

A1. いいえ、FETにはソース-ドレイン間にボディ・ダイオードを内蔵しており、逆方向ではダイオードの順方向となります。そのため、オン抵抗が大きい領域ではダイオードに電流が流れて、スイッチング動作ができません。

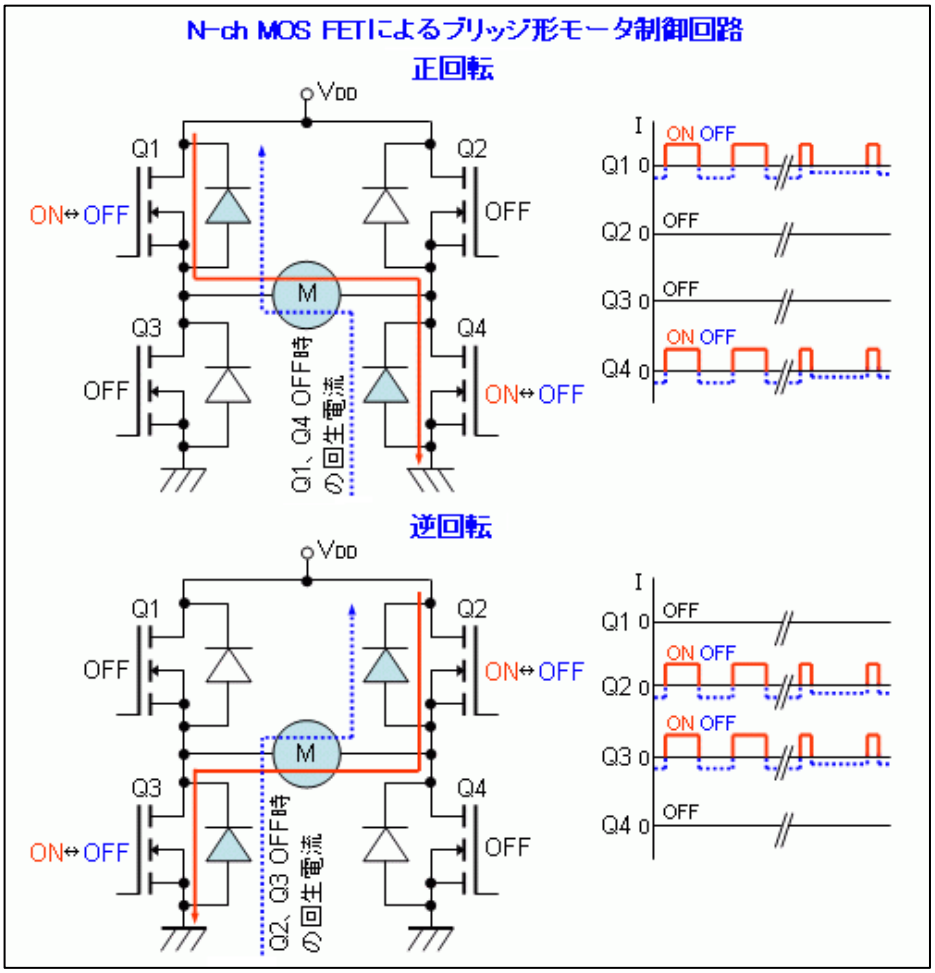


Q2. Nch-MOS FET のソースからドレーンへ電流を流し、整流ダイオードとして使うことはできますか。

A2. はい、このような用途は、MOS FET の一般的な使用方法です。
ただし、特に trr などを配慮した設計となっていませんので、SBD/FRD 注などの特殊なダイオードの特性が必要な場合には、相当する単品ダイオードをご使用ください。

注 SBD：Schottky Barrier Diode、FRD：Fast Recovery Diode

ダイオードを整流に使用する例として、モータのブリッジ制御回路があります。このとき、PWM でパルス幅を制御しますが、全 FET がオフ状態で、慣性による回転をしているモータに生じる電位差により、ボディ・ダイオードを通じて、回生電流がグランドから電源に流れます。



Q3. ソースからドレーンへ電流を流す場合、許容損失やオン抵抗などはデータ・シートの値と同じですか。

A3. 電力定格は、順方向と同等です。オン抵抗は規定していません。ゲート順電圧を印加すると、構造上、寄生ダイオードと MOS のオン抵抗が並列接続される構成となります。MOS のオン抵抗による電圧降下が V_F よりかなり低いと、ダイオードには電流がほとんど流れず、オン抵抗に流れます。

パルス電流定格

Q1. FET のパルス電流定格で、条件(10us)のパルス幅が短くなれば(たとえば数 ns)、もっと電流を流せますか。

A1. いいえ、たとえパルス幅が違っても、パルス電流定格は変わりません。パルス電流定格は、P_W≦10us、Duty Cycle≦10%の条件で一義的に決めており、数 ns の幅でも、この値を越える使い方はできません。
なお、安全動作領域の各パルス幅による斜線部は、電力定格で制限される領域です。

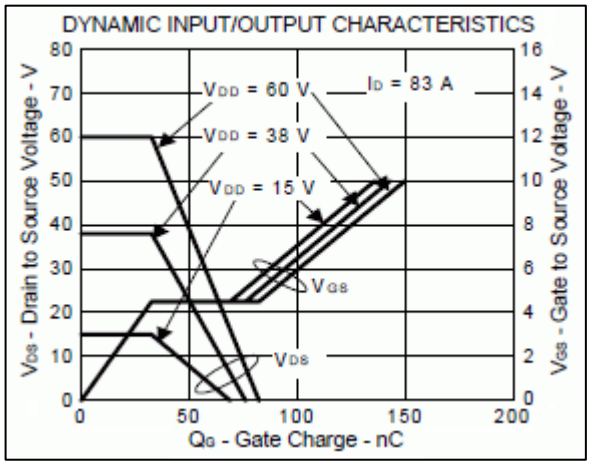
Q2. 2SK2857 のドレイン電流(直流)は最大定格値±4A になっています。直流でなくパルス波形で使用する場合、周期＝1ms、デューティ＝45%のパルスでは、実力的に何 A まで流せますか。

A2. 電流定格の実力値というものはありません。この場合、ドレイン電流(直流)定格±4A を適用してください。

電荷量規格

Q1. 2SK3510 のデータ・シートに、Qg＝typ.150nC、Qgs＝typ.30nC、Qgd＝typ.52nC と掲載されています。
なぜ、Qg＝Qgs＋Qgd でないのですか。

A1. データ・シートの「DYNAMIC INPUT/OUTPUT CHARACTERISTICS」特性グラフをご参照ください。



Qgs は、FET がオン開始までの電荷量です。それまでは、ゲート-ソース間容量を充電します。グラフでは、V_{GS} の最初のランプアップ期間が、この状態にあたります。グラフから読み取ると、約 30nC です。
Qgd は、FET がオン開始後、ドレーン-ソース間電圧が低下して、ゲート-ドレーン間の容量を充電する電荷量です。グラフでは、V_{GS} がランプアップ後に、平たんになっている期間が、この状態にあたります。グラフから読み取ると、約 50nC です。
ゲート-ドレーイン間容量の充電が完了すると、再びゲート電圧はランプアップします。
Qg は、スペック条件にあるように、ゲート-ソース間電圧を 10V とするのに必要なゲート総電荷量です。

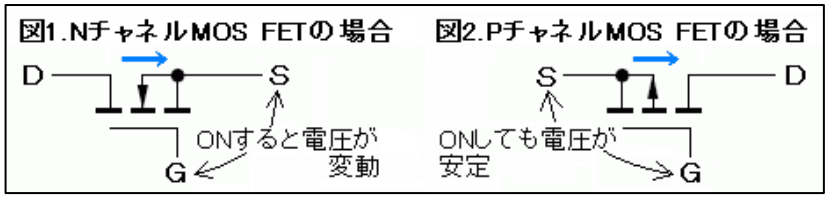
Qg＝Qgs＋Qgd＋[平坦後のランクアップで、V_{GS} が 10V になるまで充電するのに必要な電荷量]

グラフから、V_{DD}＝60V の条件では約 150nC です。
したがって、スペック上の Qg は、Qgs＋Qgd にはなりません。

ハイサイド・スイッチへの応用

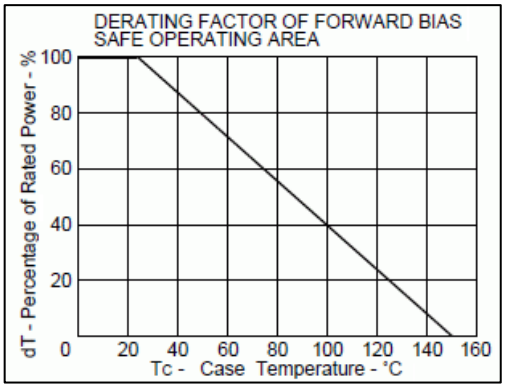
Q1. 2SK1583 で電源の ON/OFF 回路を作りたいのですが、ドレーン側に供給元の電源を接続し、ソース側を基板全体の電源として、ゲート端子のハイ／ロウの切り換えで、基板の電源 ON/OFF はできますか。

A1. 通常、ハイサイド・スイッチとしてＰチャネル(2SJ)FETを使用し、ソース側からドレーン側へ電源を供給するのが一般的です。**図 1** のようなＮチャネル(2SK)FET では、ON するとドレーン側からソース側に供給された電圧で、ゲート・ソース間電圧が変動して動作が不安定となります。この対策で、回路が複雑になります。**図 2** のＰチャネル・タイプでは、ゲート・ソース間電圧が固定されるので、簡素な回路で安定した動作となります。



ドレーン電流定格のディレーティング

Q1. ドレーン電流(直流)は、ケース温度に対するディレーティング・ファクタ dT(%)の制約を受けますか。



A1. ドレーン電流がディレーティング・ファクタの制約を受けるのではなく、温度が上昇すると内部抵抗が大きくなり、結果として消費電力のかたちでドレーン電流がディレーティング・ファクタの制約を受けることになります。ドレーン・ソース間電圧はこの制約を受けませんが、定格の 80%以内でのご使用を推奨します。

ゲート・カットオフ電圧

Q1. 2SJ601-Z のゲート・カットオフ電圧が、データ・シートの規格値と異なるのはなぜですか。

A1. 2SJ601-Z のゲート・カットオフ電圧規格は、V_{DS}＝－10V、I_D＝－1mA をオフ条件と定め、そのときのゲート電圧を規定しています。オフ条件や測定温度などが違えば、ゲート・カットオフ電圧は異なります。

2 素子入りの電流定格

Q1. μPA2755 は 2 素子入りですが、絶対最大定格のドレイン電流 8A は、2 素子の合計ですか。

A1. いいえ、1 素子あたりの定格です。2 素子の合計電流は、全損失によって制限されます。

7.4 ダイオード

ツェナー・ダイオードの順方向特性

Q1. ツェナー・ダイオードは逆方向の特性しかありません。順方向はどのようになりますか。

A1. 一般のダイオードと同じ特性を示します。シリコンの場合は、約 0.6V で電流が流れ始めます。
ただし、ツェナー・ダイオードは順方向特性を測定していませんので、順方向特性の保証値はありません。

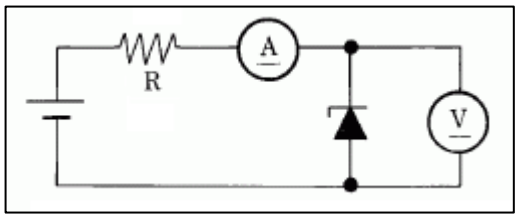
低電圧のツェナー・ダイオード

Q1. 1mA で 2.5V のツェナー・ダイオードはありますか。

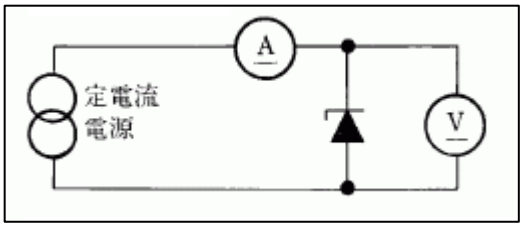
A1. ありますが、低電圧(約 5V 以下)のツェナー・ダイオードは、低電流領域で安定した電圧を得ることが困難です(電流が変化すると、電圧も変化する)。
μPC1093(シャント・レギュレータ)を使用されてはいかがでしょうか。この IC は、1mA で安定した電圧が得られます。ただし、μPC1093 はアクティブ素子です。ツェナー・ダイオードと違って、外部に発振止めのコンデンサが必要です。詳しくは、データ・シートなどでご確認ください。

ツェナー電圧の測定方法

Q1. ツェナー電圧は、次図のように抵抗を介して測定すればいいですか。



A1. 抵抗を介すると、抵抗値が小さすぎたり電源電圧の変動などによる過電流があると、ツェナー・ダイオードの破壊を招く恐れがあります。次図の定電流源を使用した方法をお勧めします。
なお、測定時間によっては、温度上昇により電圧規格との誤差が生じることもありますのでご留意ください。



ツェナー電流の決め方

Q1. 電圧を安定して出力できるツェナー電流(Iz)の値を教えてください。

A1. データ・シート記載のツェナー電圧(Vz)測定条件のツェナー電流(Iz)値を目安としてください。
電流を大きくすると発熱量が増大し、温度によるツェナー電圧変動が大きくなります。
また、電流を小さくすると、ダイオード自体から発生するノイズが増大します。

ツェナー・ダイオードの電圧電流の限界

Q1. ツェナー電流は、どれだけ流せますか。

A1. ツェナー電流の許容値は、ツェナー電圧、通電条件(使用条件)、実装条件により異なります。基本的には、使用条件で接合温度を算出して、絶対最大定格接合温度(Tj)を越えないことをご確認ください。
考え方と接合温度の算出方法は、ホームページ掲載の下記資料をご参照ください。
「定電圧ダイオードおよびノイズ・クリッピング・ダイオードの使用上の諸注意」

Q2. ツェナー電圧は、どれだけ印加できますか。

A2. 次の2種類の場合が考えられ、それぞれ異なります。

(1) 静電気による電圧印加の場合

静電気耐量を問題とされる場合は、静電気耐量を保証している NNCD シリーズをお勧めします。

(2) DC 印加、パルス電圧の繰り返し印加の場合

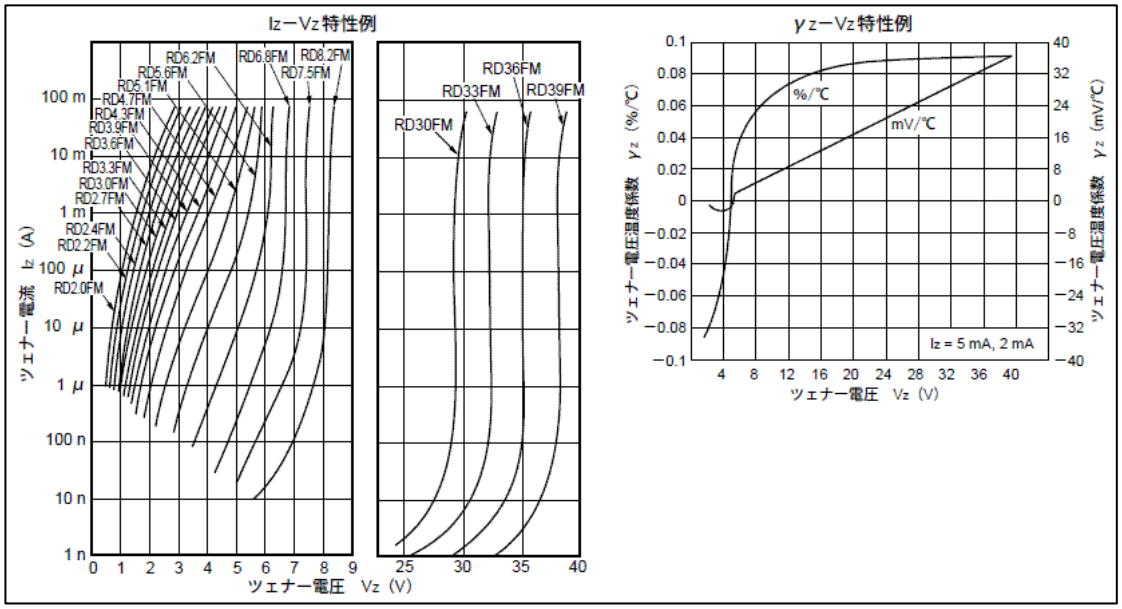
L負荷の解放サージ吸収などの応用で、最も多いのがこのケースです。この場合も、上記 A1 と同様に、使用条件で接合温度を算出して、絶対最大定格接合温度(Tj)を越えないことをご確認ください。
考え方と接合温度の算出方法は、ホームページ掲載の下記資料をご参照ください。
「定電圧ダイオードおよびノイズ・クリッピング・ダイオードの使用上の諸注意」

★

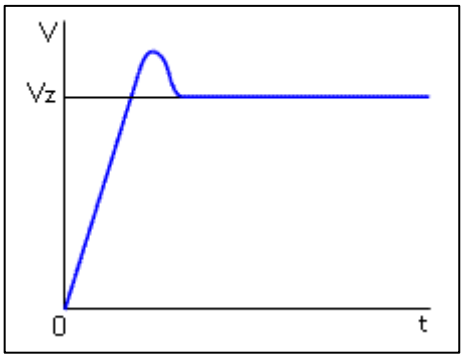
温度特性

Q1. ツェナー・ダイオードに 10uA を流して 39V にクランプしたいのですが、低温になると、一瞬ツェナー電圧を越えてから定電圧に下がるのはなぜですか。

A1. ツェナー・ダイオードは、トンネル効果とアバランシェ効果（電子なだれ）で機能する素子です。
トンネル効果は、PN 接合への逆バイアスで広がった空乏層を電子が突き抜ける現象です。
アバランシェ効果は、シリコン結晶に衝突した自由電子が、共有結合の電子をはじき出して、連鎖的に自由電子が増加する現象です。
ツェナー電圧が約 5.1V までの素子は、トンネル効果が支配的です。それを越えると、アバランシェ効果が支配的です。ツェナー電圧の温度係数が、約 5.1V までは負で、それを越えると正になるのはこのためです。



熱エネルギーも、シリコン結晶から自由電子を放出させる要因になります。ツェナー電圧が 39V では、10uA でもトンネル効果を生じる電圧に達していますが、さらにアバランシェ効果が必要です。低温で電流量が少ないと、アバランシェ効果が生じるまでブレークダウンせず、いったん電圧が上昇することがあります。電圧上昇による電流量の増加や、通電でのチップ温度上昇も、アバランシェ効果の要因になります。



ツェナー・ダイオードの応答性

Q1. ツェナー・ダイオードの応答性は、どのように考えればいいですか。

A1. 応答性は、端子間容量に依存します。端子間容量と応答性の関係については、ホームページ掲載のノイズ・クリッピング・ダイオード「NNCD シリーズ」をご参照ください。

ツェナー・ダイオードの端子間容量

Q1. RD シリーズの端子間容量を教えてください。

A1. RD シリーズの用途は、基準電圧、電圧検出、レベル・シフトなどなので、端子間容量を規定していません。端子間容量を規定している、高速信号線の ESD 保護、サージ吸収用途の NNCD シリーズや NSAD シリーズをご使用ください。

ツェナー・ダイオードの並列接続

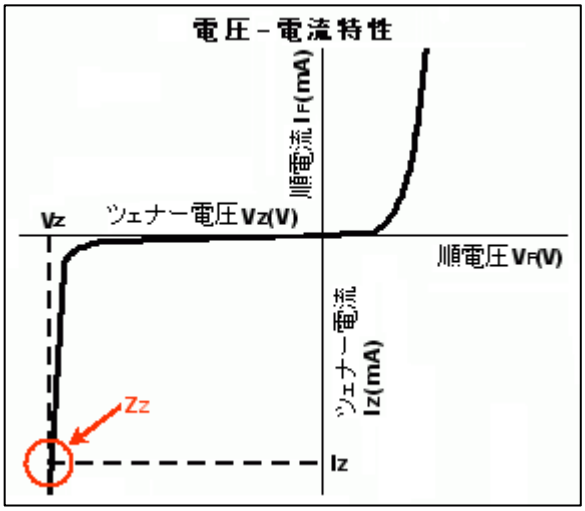
Q1. 損失の分散を図りたいのですが、複数のツェナー・ダイオードを並列接続していいですか。

A1. いいえ、ツェナー・ダイオードの並列接続はしないでください。ツェナー電圧のわずかなばらつきでも、電流の大きなアンバランスの要因となり、部分的に負荷が集中します。
ツェナー電圧のばらつきが大きい組み合わせの場合、1 つしかオンしない場合もあります。

動作抵抗

Q1. 定電圧ダイオードの特性項目に動作抵抗 Zz がありますが、どのような抵抗ですか。

A1. 動作抵抗 Zz は、次図の○の領域の微分抵抗です。ツェナー電流を流した状態で微小交流信号を重畳して、微分抵抗として測定しています。その値は、Vz＝6～7V 附近で最小となり、これより低い Vz のツェナー・ダイオードでも、高い Vz のツェナー・ダイオードでも、Zz は大きくなります。



温度係数の小さい製品

Q1. 温度係数が保証されている電圧標準ダイオードの 1SZ シリーズは、近い将来廃止されると聞きました。温度係数を保証している代替製品はありますか。

A1. 他のツェナー・ダイオードで、温度係数を保証している製品はありません。高精度可変シャント式安定化電源回路の μPC1093 をお勧めします。

ESD 保護用ツェナー・ダイオード

Q1. データ・ラインの ESD 保護用ダイオードはありますか。

A1. はい、NNCD シリーズと NSAD シリーズがあります。特に NSAD シリーズは、USB2.0、IEE1394a、DVI や 100B のような、高速データ・ラインの保護用に開発されたダイオードです。
各シリーズの詳細は、下記ホームページをご覧ください。
- NSAD シリーズ特徴
- NNCD シリーズ特徴

Q2. 5V 電源のノイズ除去に使用する NNCD の降伏電圧は、何 V を選定すればいいですか。

A2. 降伏電圧を決める場合、NNCD の下限電圧が、電源ラインの上限電圧以上のものをお選びください。
5V 電源の場合、6.8V 品あたりの NNCD が適当と考えます。

Q3. RS-232C などの信号ラインの ESD 保護に、2 個の NNCD を向かい合わせで使用すればいいですか。

A3. はい、それでけっこうです。
信号ラインに乗る ESD は、正負のいずれも可能性があります。このため、2 個のカソード同士を接続し、一方のアノードをライン側、もう一方のアノードをグランド側へ、それぞれ接続してください。

破壊用途

Q1. ツェナー・ダイオードを過電流でヒューズのように確実に焼き切って絶縁状態にする条件はありますか。

A1. ツェナー・ダイオードは、逆バイアスで定電圧を確保するための素子なので、そのような破壊用途は想定していません。そのような使い方はしないでください。そのような特性データもありません。なお、半導体製品の破壊モードは、オープン、ショート、オーミック接触のいずれもあり得るため、特定はできません。

パッケージ

Q1. 同じダイオードで、灰色のものと透明ガラスのものがあり、色帯は同じですが、特性が違うのですか。

A1. 特性は同じです。2002 年 4 月に、灰色の下地を廃止しました。

表面実装用マウント・パッド

Q1. 表面実装用ダイオードの推奨マウント・パッドはありますか。

A1. パッケージごとに異なりますので、下記Web ページで該当パッケージ名の[マウント・パッド >]をクリックして参照してください。

http://www2.renesas.com/discrete/ja/package/smd.html

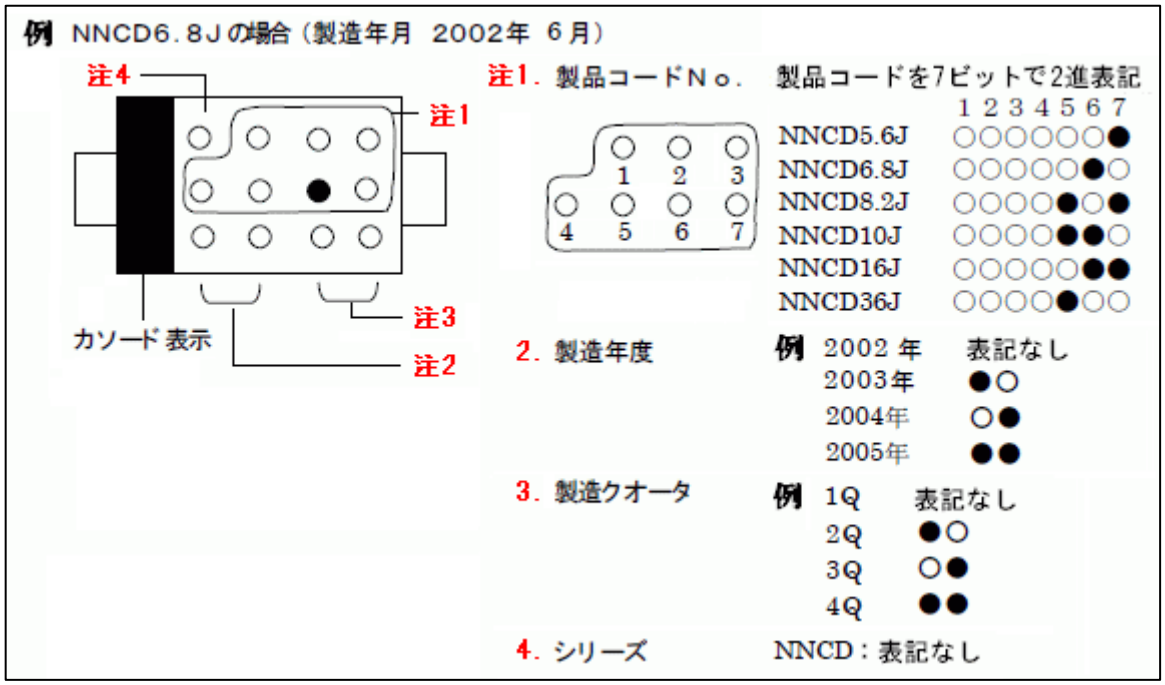
捺印仕様

Q1. ツェナー・ダイオードの捺印仕様を教えてください。

A1. 技術資料「ノイズクリッピングダイオード，定電圧ダイオードの捺印仕様」をご参照ください。

Q2. NNCD[ ]Jは上記の技術資料に掲載されていませんが、捺印仕様はどのようになっていますか。

A2. 次のようになります。



Q3. RD6.2S で、B1 の規格表示は「1」を捺印してあるのですが、B 規格品はどのような表記になりますか。

A3. B 規格品のご要求には、B1、B2、B3 のいずれかの規格でお応えしています。
したがって、B 規格品といった表記はなく、「1」,「2」,「3」のいずれかの捺印となります。

Q4. RD27S は、データ・シートで B 規格だけですが、現品には"B1"規格を示す捺印があります。実際には、B1,B2,B3 の区分があるのですか。

A4. いいえ、RD27S は B 規格だけです。便宜的に、B1 規格を示す"1"を捺印しています。

ノイズ・クリッピング・ダイオード（NNCD）のクリッピング特性

Q1. サージ保護用ダイオード NNCD5.6LH に 8kV の静電気を印加すると、サージは何 V に抑えられますか。

A1. IEC61000-4-2 試験での参考結果では、8kV の印加に対して 2.5kV 前後でクリッピングしています。

RD[ ]E（ガラス・タイプ）

Q1. RD□E の L 規格はどのようなものですか。B 規格品に置き換える場合、どの規格に相当しますか。

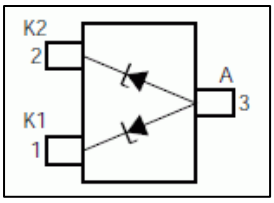
A1. L 規格品は Iz＝1mA で電圧を保証したもので、現在は販売していません。
B 規格品（Iz＝20mA で電圧を保証）への置き換えは、Iz 値で電圧が違ってきますので、単純に同じ電圧品への置き換えはできません。技術検討を伴いますので販売店にご相談ください。

★ Q2. RD□E の代替品はありますか。

A2. 相当品としてHZシリーズがあります。特性に少し差がありますので、代替についてはご評価願います。

2 素子入り製品の全損失

Q1. RD[ ]MW シリーズは 2 素子入りで、両ダイオード・トータルで 200mW と記載されています。この損失は、1 素子あたり 100mW ですか。それとも、1 素子が 0mW なら、他方は 200mW まで使用できますか。



A1. 全損失はパッケージで規定しています。
したがって、2 素子入り RD[ ]MW シリーズで1素子使用の場合、全損失は1素子で 200mW となります。

許容損失と逆電力定格

Q1. RD2.0FM～RD120FM の許容損失最大定格 P＝1W の意味を教えてください。

A1. RD2.0FM～RD120FM の許容損失 P＝1W は、DC 動作時に許容できる最大の消費電力です。この許容損失を越えると、接合部温度が定格の 150℃を越えることになり、破壊事故などに至ります。ツェナー・ダイオードの損失は、ツェナー電圧 Vz とツェナー電流 Iz の積で求めます。
サージ的な電力定格については、非繰り返しを条件として、データ・シートにサージ逆電力定格を記載しています。資料「定電圧ダイオードおよびノイズクリッピングダイオードの使用上の諸注意」もご参照ください。

8. オプトエレクトロニクス

8.1 共　通

安全規格

Q1. フォトカプラと光 MOS FET で、多くの海外安全規格を取得しているのはなぜですか。

A1. フォトカプラや光 MOS FET を搭載するセット(電源、FA機器など)では、セットに対して 1 次側と 2 次側の分離(アイソレーション)に関する規定があります。この機能をフォトカプラが代行します。これらの製品がその規定に準拠していれば、セットとしての認定試験が免除されることもあるので、当社では海外安全規格の認定取得を進めています。なお、製品ごとの海外安全規格の認定取得状況は、下記 URL で確認できます。

フォトカプラ　：http://www2.renesas.com/opto/ja/safety.html

光 MOS FET：http://www2.renesas.com/opto/ja/safetyocmos.html

Q2. 海外安全規格認定証は。Web に掲示されていますか。

A2. ホームページからダウンロードできませんが、認定は取れていますので、販社経由でご請求ください。

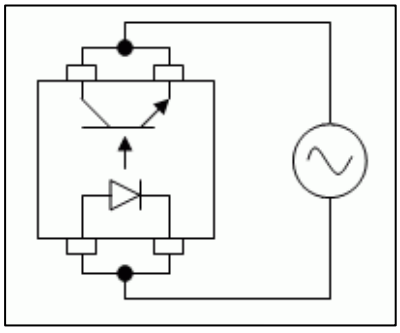
特　性

Q1. 「絶縁耐圧」、「沿面距離」、「絶縁物厚」、「空間距離」とは、どういう意味ですか。

A1. それぞれ、次のような意味です。

■**絶縁耐圧**(Isolation Voltage)

入力端子－出力端子間に印加できる、交流電圧の最大値を実効値(r.m.s.:Root　Mean　Square)で表示し、絶縁耐量を示します。測定条件は、通常 Ta=25℃、RH=60%で、入力側全端子と出力側全端子の間に、規定の交流電圧を 1 分間印加することとしています。



■**沿面距離**(Creepage　Distance)

絶縁物質に沿う 2 つの導体(入力端子－出力端子)間の、最小距離を示します

■**絶縁物厚**(Isolation　Thickness)

2 つの導体(入力－出力)間にある絶縁物の、最小の厚みを示します

■**空間距離**(Clearance)

空気中における 2 つの導体(入力端子－出力端子)間の、最小距離を示します

詳細は、次の URL をご参照ください。

http://www2.renesas.com/opto/ja/parameter_p/index.html

★ **Q2.** フォトカプラや光 MOS FET は最高ジャンクション温度が規定されていませんが、どのように考えればいいのですか。

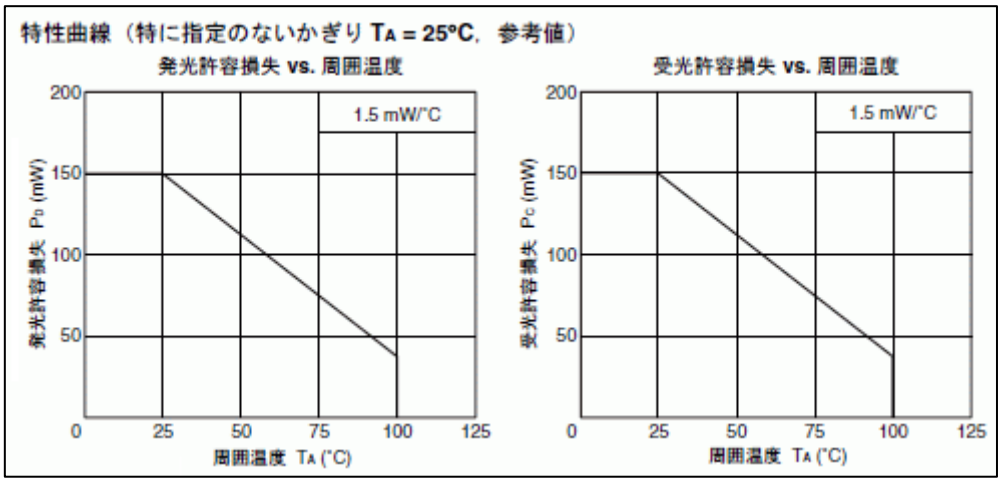
A2. フォトカプラや光 MOS FET は、非通電時と通電時で異なります。

非通電時では、一般的な半導体デバイスと同様の最大定格「保存温度」の上限値です。

通電時には、光電デバイス特有の高温劣化現象を生じるため、最高ジャンクション温度は非通電時より低くなります。その値は、特性曲線「許容損失 vs.周囲温度」で示されます。

特性曲線の傾斜部を延長すると、およそ 125℃で許容損失が 0mW になります。

「動作周囲温度」の許容範囲内で、この傾斜線近傍の内部損失があったとき、ジャンクション温度は 125℃付近になっていると推定されるため、この 125℃が「通電時」の「最高ジャンクション温度」です。



詳細は、次の Web ページをご参照ください。

http://www2.renesas.com/opto/ja/technology/radiation/index.html

★ **Q3.** フォトカプラの熱抵抗を教えてください。

A3. フォトカプラや光 MOS FET では、最大定格の「低減率（mW/℃）」が規定されていることがあります。その逆数を「ジャンクション−周囲間の熱抵抗：$R_{th}(j-a)$」と考えることができます。

「低減率」が規定されていない場合は、一般のシグナル・デバイスと同様、特性曲線「許容損失 vs.周囲温度」の傾斜部の傾きが $R_{th}(j-a)$になります。

絶対最大定格（特に指定のないかぎり TA = 25°C）

項　目		略　号	定　格		単　位
			PS2501-1, PS2501L-1	PS2501-2,-4, PS2501L-2,-4	
発　光	逆電圧	VR	6		V
	順電流（DC）	IF	80		mA/ch
	低減率	ΔPD/°C	1.5	1.2	mW/°C
	許容損失	PD	150	120	mW/ch
	ピーク順電流 注1	IFP	1		A/ch

★ **Q4.** データ・シートに AC での絶縁耐圧が載っていますが、DC での絶縁耐圧を教えてください。

A4. フォトカプラやのフォト MOSFET の絶縁耐圧は、一般的に入力端子ー出力端子間に印加できる交流電圧の最大値を実効値(r.m.s.:Root Mean Square)で表示し、絶縁耐量を保証しています。

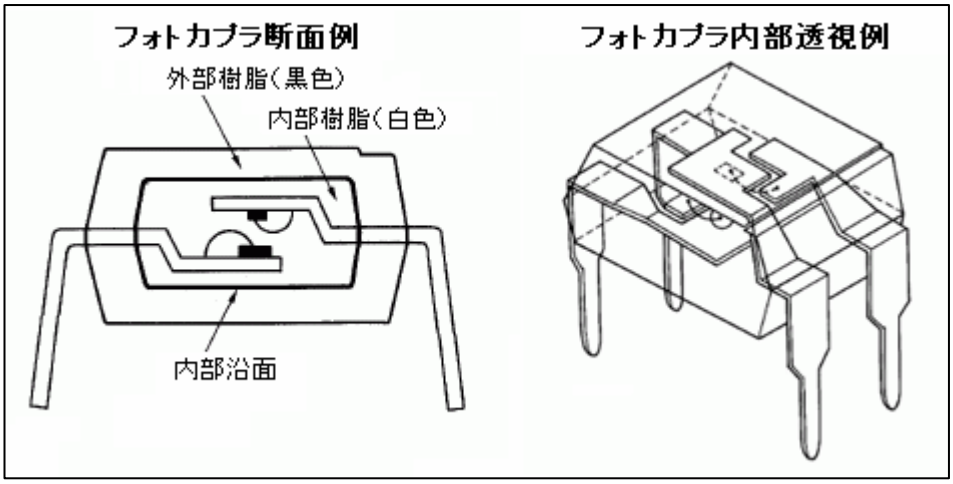
DC での規定はありませんが個別に対応させていただきますので、販社経由でご相談ください。

8.2 フォトカプラ

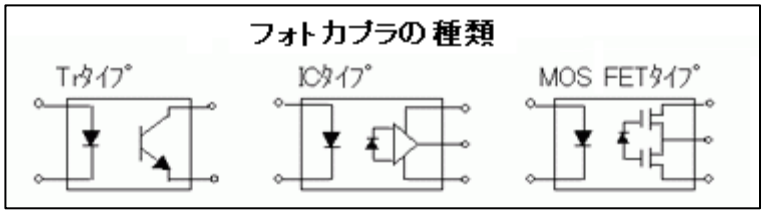
基　礎

Q1. フォトカプラとは、どういうものですか。

A1. 光結合素子の一種で、発光素子と受光素子とを 1 つのパッケージ内で一体化したものです。



一般に発光素子としては LED が用いられ、受光素子にはフォトトランジスタ、フォト MOS FET、フォト IC などが用いられます。その使用される受光素子により、フォトトランジスタ出力、光 MOS FET、フォト IC 出力などの種類があります。



[特徴]

フォトカプラは、信号を光で伝達するため 次のような特徴があります。

①入出力間を電気的に絶縁分離することが可能

②直流から高周波まで、広帯域の電気信号を一方向に伝達

③リレーなどに比べて小型

特　性

Q1. PS2501 の発光側で、順電流 I_F の規定はありますが、アノード-カソード間の供給電圧に規定はありますか。

A1. 発光側のアノード・カソード間に供給する電圧は、発光側の許容損失(順電流×順電圧)で制限されます。

絶対最大定格（特に指定のないかぎり T_A = 25°C）

項　目		略　号	定　格		単　位
			PS2501-1, PS2501L-1	PS2501-2,-4, PS2501L-2,-4	
発　光	逆電圧	V_R	6		V
	順電流（DC）	I_F	80		mA/ch
	低減率	ΔP_D/°C	1.5	1.2	mW/°C
	許容損失	P_D	150	120	mW/ch
	ピーク順電流 注1	I_{FP}	1		A/ch

★ **Q2.** フォトカプラの入力（発光）の順電圧は 2V 以下ですが、5V で駆動する場合、抵抗を直列につないで、フォトカプラにかかる電圧を下げればいいですか。

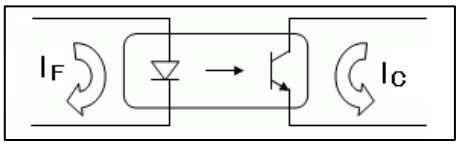
A2. 順電圧は入力電圧ではありません。発光側ダイオードのアノード−カソード間に規定の電流を流したときに生じる電位差です。順電圧の最大値以上を加えれば、フォトカプラは確実に動作します。
ただし、入力電圧が高ければ順電流が大きくなり、入力側損失が増大するので、直列に電流制限用の抵抗を挿入し、順電流、許容損失定格を越えないような設計が必要となります。

電気的特性（T_A = 25°C）

項目		略号	条件	MIN.	TYP.	MAX.	単位
発光	順電圧	V_F	I_F = 10 mA		1.17	1.4	V
	逆電流	I_R	V_R = 5 V			5	μA
	端子間容量	C_t	V = 0 V, f = 1.0 MHz		50		pF
受光	コレクタしゃ断電流	I_{CEO}	V_{CE} = 80 V, I_F = 0 mA			100	nA

Q3. CTR とは、どのような特性ですか。

A3. 「(Current Transfer Ratio」の略で、フォトカプラの入力と出力間の電流効率を表します。
出力側電流 I_C と入力側電流 I_F の比率（CTR＝I_C/I_F × 100）(%)です。
電流伝達率とも呼び、通常は汎用フォトカプラの最も重要な性能指標です。



なお、CTR には次のような固有の特性があります。
①I_F(LED に流す順電流)に依存する
②周囲温度の影響を受ける
③経時的（経年的）変化がある
詳細は、次の URL をご参照ください。
http://www2.renesas.com/opto/ja/technology/ctr/index.html

Q4. PS2801-4-F3 の CTR ランクは、PS2801-1 と同じですか。

A4. いいえ、CTR ランクの規定は PS2801-1 だけに適応しており、PS2801-4 には適応していません。

電気的特性（T_A = 25°C）

項目		略号	条件	MIN.	TYP.	MAX.	単位
発光	順電圧	V_F	I_F = 5 mA		1.1	1.4	V
	逆電流	I_R	V_R = 5 V			5	μA
	端子間容量	C_t	V = 0 V, f = 1.0 MHz		15		pF
受光	コレクタしゃ断電流	I_{CEO}	V_{CE} = 80 V, I_F = 0 mA			100	nA
伝達特性	電流伝達率 (I_C/I_F) 注1	CTR	I_F = 5 mA, V_{CE} = 5 V	80		600	%

注1. CTR ランク（PS2801-1 のみ実施）
K : 300～600 (%)
P : 150～300 (%)
L : 100～300 (%)
N : 80～600 (%)

★ **Q5.** PS2503-1 の CTR のランク（K, L, M）のどれが、生産量的に問題なさそうですか。

A5. ランクごとの生産量は、受注状況や製造段階での変動があるので、まったく分かりません。
むしろ、どのランクでもかまわないのなら、ランク指定をしないオール・ランクでのご購入をお勧めします。

★ **Q6.** 汎用型製品には特性表の MIN.～MAX.値で規定した N ランク品がありますが、これは他の P,L,M ランク品のいずれかで出荷されるのですか。

電気的特性（T_A = 25°C）

項目		略号	条件	MIN.	TYP.	MAX.	単位
発光	順電圧	V_F	I_F = 5 mA		1.2	1.4	V
伝達特性	電流伝達率 (I_C/I_F) 注1	CTR	I_F = 5 mA, V_{CE} = 5 V	50		300	%
	コレクタ飽和電圧	$V_{CE(sat)}$	I_F = 10 mA, I_C = 2 mA		0.13	0.3	V
	立ち下がり時間 注2	t_f			7		

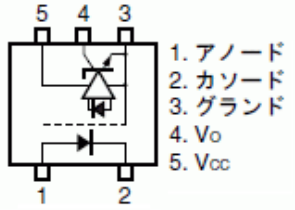
注1. CTR ランク
N ： 50～300 (%)
P ：150～300 (%)
L ：100～300 (%)
M ： 50～150 (%)

A6. N ランクは、製品規格の MIN.～MAX.値範囲内のオールランク品です。
N ランクとして、P,L,M ランク品を出荷するのではありません。

Q7. PS9117A(IC出力、超高速 10Mbps)の電気的特性に、CTR 特性が掲載されていないのはなぜですか。

A7. PS9117A には、デジタル出力機能があります。入力とこの出力の関係は、次の真理値表で表されます。

端子接続図 (Top View)



1. アノード
2. カソード
3. グランド
4. Vo
5. Vcc

真理値表

LED	出力
ON	L
OFF	H

出力を「H」から「L」へ変化させるには、伝達特性の「スレッシュホールド入力電流(H→L):I_{FHL}」以上の入力電流を流す必要があります。なお、出力を「L」から「H」へ変化させるには、入力電流をI_{FHL}以下とします。

電気的特性（特に指定のないかぎり T_A = −40～+85°C）

項目		略号	条件	MIN.	TYP. 注1	MAX.	単位
伝達特性	スレッシュホールド入力電流 (H→L)	I_{FHL}	V_{CC} = 5 V, V_O = 0.8 V, R_L = 350 Ω		2	5	

このように、入力電流を制御して信号伝達をするため、設計に CTR 特性が不要なので、規定していません。

Q8. フォトカプラの寿命は電流伝達率 CTR で確認されますが、高速フォトカプラ（アナログ出力タイプを除く）では、CTR が掲載されていない製品があります。この場合、寿命はどのように確認できますか。

A8. 高速フォトカプラの通電劣化の目安として、電気的特性（伝達特性）のスレッシュホールド入力電流I_{FHL}で確認できます。劣化が進めば、I_{FHL}の規格値以上の電流を流しても、出力が「H→L」の変化をしなくなります。この現象で寿命に至ったと判断できます。詳細については、購入先にご確認ください。

【PS9115 の例】

項目		略号	条件	MIN.	TYP. 注1	MAX.	単位
伝達特性	スレッシュホールド入力電流 (H→L)	I_{FHL}	T_A = 25°C		2.3	5	mA
			V_{CC} = 5 V, V_O = 0.6 V			6	

Q9. フォトカプラの納入品で、CTR ランクはどのように確認できますか。

A9. デバイスにはランク捺印がありません。梱包ラベルで、「(Z)S/NO:」に CTR ランクを記載しています。次の例では、最初のアルファベット K が「K ランク」です。他のアルファベットや数字は社内管理記号です。

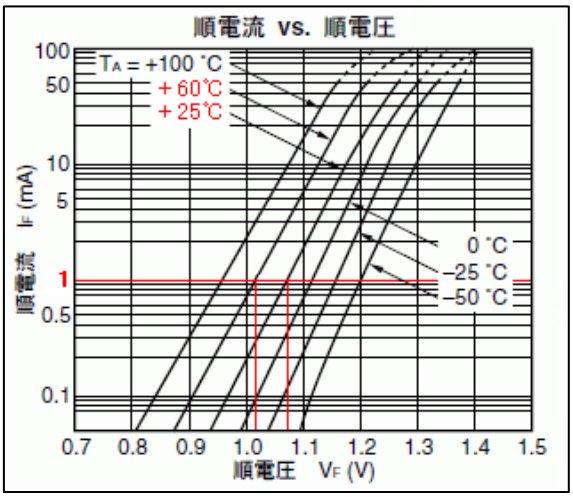


Q10. PS2802-4-Aの一次側ダイオードで、V_Fの温度係数はどのくらいですか。

A10. データ・シートの特性曲線「順電流 vs. 順電圧」で参考値を確認できます。まず、使用する順電流値 I_F を定めます。次に、その値と各温度曲線の交点で順電圧差が求められ、その温度差で除算してください。

例 1mA を流したときの、25℃から 60℃の温度変化による V_F変化。

$\Delta V_F = (1.12V - 1.17V) / (60℃ - 25℃) = -0.05V/35℃ = -1.4mV/℃$

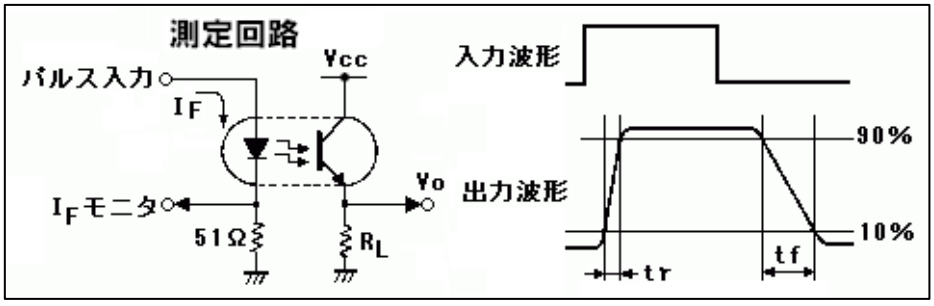


Q11. 立ち上がり／立ち下がり時間(t_r, t_f)とは、どのような特性ですか。

A11. 発光側にパルスで順電流 I_F を流して、出力が ON/OFF するときの、出力電圧の過渡的変化時間です。

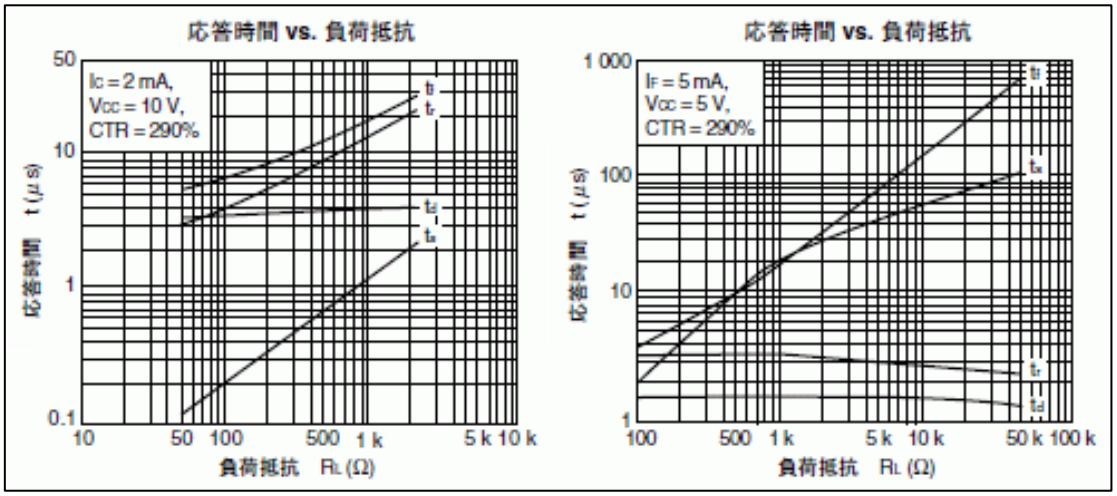
t_r : 出力波形が 10%から 90%に遷移する時間

t_f : 出力波形が 90%から 10%に遷移する時間



★ **Q12.** PS2501 の入力に比べて、2 次側の波形の立ち上がりが約 25μs 遅れます（負荷抵抗1kΩ、1 次側の電流 20mA）。

A12. 「応答時間 vs. 負荷抵抗」特性に対して、負荷抵抗が大きいためと思われます。
高速応答が必要でしたら、IC 出力タイプをご検討ください。

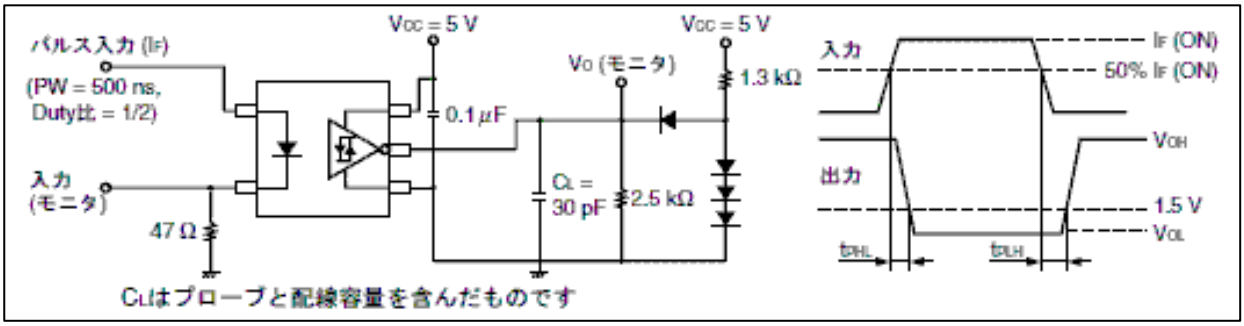


★ **Q13.** フォトカプラの応答速度と周囲温度の関係はどうなりますか。

A13. 周囲温度が高くなると応答速度は遅くなります。
製品個別のデータはありませんが、温度特性の傾向は次の URL をご参照ください。
http://www2.renesas.com/faq/ja/f_pcpl.html#0601

Q14. 伝達遅延時間(H→L)(t_{PHL})、伝達遅延時間(L→H)(t_{PLH})とは、どのような特性ですか。

A14. 発光側にパルスで順電流 I_F を流したときの、順電流変化から出力電圧変化までの遅れ時間（入力と出力の相関）です。「立ち上がり時間t_r／立ち下がり時間t_f」のような、出力だけの遷移ではありません。
次の例は、PS9115 のデータ・シート掲載の伝達遅延時間と測定回路です。



Q15. パルス幅ひずみ（$|t_{PHL}-t_{PLH}|$）とは、どのような特性ですか。

A15. 発光側から入力/送信したパルス信号が、受信側で変化する程度のことです。その変化を伝達遅延時間を用いて「$|t_{PHL}-t_{PLH}|$」で表します。
この値が小さいフォトカプラはひずみが小さい、つまりパルス幅の変化量が少ない特性といえます。

Q16. 低減率（ΔP_D/℃、ΔP_C/℃）とは、どのような特性ですか。

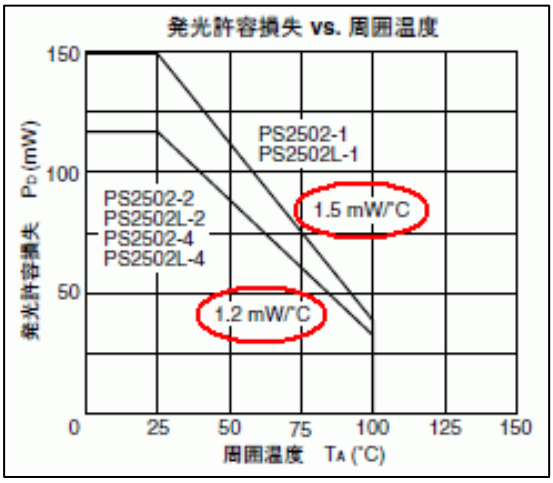
A16. 許容損失（発光側：P_D、受光側：P_C）について、周囲温度が 1℃上昇したときの許容損失の低減率です。

例として、PS2502 の絶対最大定格で低減率（発光側）として、次のように規定されています。

絶対最大定格（特に指定のないかぎり T_A = 25°C）

項目		略号	定格 PS2502-1, PS2502L-1	定格 PS2502-2,-4, PS2502L-2,-4	単位
発光	逆電圧	V_R	6		V
	順電流（DC）	I_F	80		mA/ch
	低減率	ΔP_D/°C	1.5	1.2	mW/°C

この発光側の低減率（ΔP_D/℃）は、特性曲線で 25℃以上での直線の傾きとして示されます。



受光側についても、同様にデータ・シートに掲載されています。

Q17. PS2502-4 の絶対最大定格に、コレクタ電流 Ic が 160mA/chと記載されています。これは、4ch 同時にコレクタ電流を 160mA まで流せるということですか。

絶対最大定格（特に指定のないかぎり T_A = 25°C）

項目		略号	定格 PS2502-1, PS2502L-1	定格 PS2502-2,-4, PS2502L-2,-4	単位
受光	コレクタ・エミッタ間電圧	V_{CEO}	40		V
	エミッタ・コレクタ間電圧	V_{ECO}	6		V
	コレクタ電流	I_C	200	160	mA/ch

A17. はい、各チャネルに 160mA まで流せます。なお、許容損失 Pc＝160mW/chに対する確認も必要です。

★ **Q18.** 入力側の ON 電流は何 mA ですか。

A18. 次の URL にある技術解説(フォトカプラの「ON と OFF」)をご参照ください。

http://www2.renesas.com/opto/ja/technology/onoff/index.html

Q19. PS2701A-1 のデータ・シートで、オーダ情報に海外安全規格の記載がありますが、「PS2701A-1-V」、「PS2701A-1V-F3」、「PS2701A-1-V-F4」は UL 認定を取得していないのですか。

オーダ情報

品　名	オーダ名称	メッキ仕様	包装形態	海外安全規格	申請品名注
PS2701A-1	PS2701A-1-A	鉛フリー	マガジン・ケース　100個	標準品 （UL，CSA認定品）	PS2701A-1
PS2701A-1-F3	PS2701A-1-F3-A		エンボス・テーピング 3 500個/リール		
PS2701A-1-F4	PS2701A-1-F4-A				
PS2701A-1-V	PS2701A-1-V-A		マガジン・ケース　100個	DIN EN60747-5-2 (VDE0884 Part2) 認定品（オプション）	
PS2701A-1-V-F3	PS2701A-1-V-F3-A		エンボス・テーピング 3 500個/リール		
PS2701A-1-V-F4	PS2701A-1-V-F4-A				

注 海外安全規格申請は申請品名で行ってください。

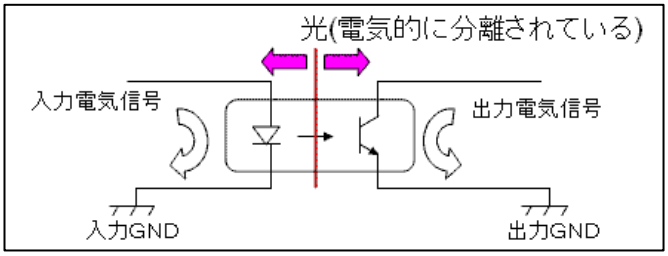
A19. いいえ、当社のフォトカプラは、基本的に UL 認定を取得しています。オーダ名称に「-V」が付くと、さらに VDE 認定も取得した製品です。フォトカプラの海外認定規格取得状況は、次の URL をご参照ください。

http://www2.renesas.com/opto/ja/safety.html

応　用

Q1. フォトカプラは、どのようなセットに応用できますか。

A1. フォトカプラの大きな特徴として、入出力間を電気的に絶縁分離することができます。



このため、次のような利点があります。

・GND電位が異なる回路間を結ぶことができる。

・別々のGNDを共通とした場合、GNDノイズの影響を防ぐことができる。

これらの利点を活かして、エアコン、スイッチング電源などに応用されています。

具体的な応用事例は、次のURLをご参照ください。

http://www2.renesas.com/opto/ja/application.html

Q2. フォトトランジスタの受光で発生する、汎用フォトカプラのノイズについて、現象と対策を教えてください。

A2. 汎用フォトカプラのフォトトランジスタは、ベースに回路配線がない浮いた状態で動作させるので、一般のトランジスタに比べて、意外な場面でノイズが発生することがあります。現象としては次のものがあります。

[現象]

①発光側－受光側間電位差変動によるノイズ

②受光側電源の立ち上がりによるノイズ

[対策]

①回路間の電位差の急激な変動を抑える工夫をする

②電源の立ち上がりを緩やかにし、電圧が安定するまでの時間帯は、出力信号を使わないように工夫する

詳細は、次のURLをご参照ください。

http://www2.renesas.com/opto/ja/technology/noise/index.html

★

構　造

Q1. PS2533L-1 の絶縁距離（構造パラメータ）を教えてください。

A1. PS2533L-1 の構造パラメータは次のようになります。

- 絶縁物厚：0.4mm
- 外部沿面距離：7.0mm
- 内部沿面距離：4.0mm
- 空間距離：7.0mm

フォトカプラ絶縁距離（構造パラメータ）については、次のURLをご参照ください。

http://www2.renesas.com/opto/ja/parameter_p/index.html

★

生産動向

Q1. フォトカプラに 1 素子（-1）と 4 素子（-4）の 2 タイプありますが、4 素子タイプは生産中止になりやすいですか。

A1. そのような傾向はありません。需要と供給の関係です。

8.3 光 MOS FET

特　性

Q1. CR 積とは、どのような特性ですか。

A1. リレーに応用したときに、高周波信号に対する特性の良し悪しを示すものです。

C：光 MOS FET がオフ時の出力側端子間容量

この容量が大きい場合、リレー・オフ状態でも高周波信号が通過してしまいます。

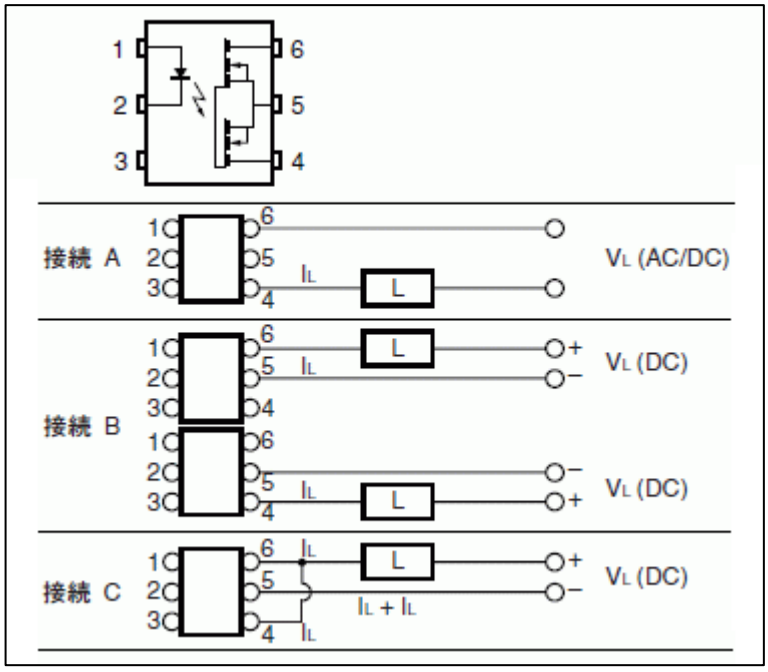
R：光 MOS FET がオン時の出力端子間抵抗

この抵抗が大きい場合、信号の損失などが生じます。

CR 積の小さい方が、高周波信号に対する特性が優れていることになります。

Q2. PS7122A-1B で、データ・シート記載のオン抵抗について教えてください。

①4.5Ω(TYP.)は、次のどの接続での値ですか。



②①に該当する接続以外での接続における値を教えてください。

A2. ①接続Aの場合です。

②接続BはAの 1/2、接続CはAの 1/4　となります。

Q3. PS7160-1A のオン抵抗には 2 種類ありますが、違いはなんですか。

電気的特性（T_A = 25°C）

項　目		略　号	条　件	MIN.	TYP.	MAX.	単　位
伝達特性	LED オン電流	I_{Fon}	I_L = 90 mA			2.0	mA
	オン抵抗	R_{on1}	I_F = 10 mA, I_L = 10 mA		42	50	Ω
		R_{on2}	I_F = 10 mA, I_L = 90 mA, t ≤ 10 ms		33	50	

A3. オン抵抗の大きい製品では、負荷電流I_Lが大きいとオン抵抗が下がる性質があります。このため、小電流域と大電流域の 2 条件で規定しています。PS7160-1A では、小電流域(I_L＝10mA)でのオン抵抗を Ron1、大電流域(I_L＝90mA、t≦10ms)でのオン抵抗を Ron2 としています。

詳細は「光 MOS　FET 規格の解説－オン抵抗：Ron(Ω)」をご参照ください。

用語集

1. 品質・信頼性用語 …850
2. 設計の予備知識 …855
3. 通信関連用語 …953

（50 音、アルファベット順）

1. 品質・信頼性用語

アレニウス・モデル(Arrhenius Model)

半導体デバイスの寿命は、電気的・熱的・機械的ストレスの中で、温度に対して最も敏感です。
温度ストレスによる反応速度の依存性は、アレニウスの化学反応論がよくあてはまることが実証されており、寿命予測に用いられます。

$L = A \cdot \exp(Ea/k \cdot T)$

L　:寿命
A　:定数
Ea　:活性化エネルギー［eV］
k　:ボルツマン定数（8.6159×10^{-5}［eV/K］）
T　:絶対温度［K］

詳細は半導体 品質/信頼性ハンドブックをご参照ください。

エージング(Aging)

製品の外観や特性を検査して不良品を除去しますが、残ったものには、本来良品として出荷すべきものだけではなく、潜在不良が内在していて経時劣化で早期に不良となるものが混入している可能性があります。
エージングというのは、良品として出荷すべきものには劣化や損傷を生じない程度のストレスを加えて、潜在不良品の劣化だけを進行させる加速試験の総称です。

エレクトロ・マイグレーション(Electro Migration)

半導体デバイスのファイン・パターン化により、配線電流密度が増加し、顕在化してきた故障モードです。
配線材（大半はAl）は多結晶構造で、その金属原子は、電子との衝突により電子の流れ方向に運動量を得て、質量輸送を起こします。
その結果として断線不良、またはヒロック（突起）やウィスカ（ヒゲ）により、短絡不良が起こります。

詳細は半導体 品質/信頼性ハンドブックをご参照ください。

温度サイクル試験(Temperature Cycle Test)

バイアスをかけずに、高温と低温での保管を一定時間ごとに繰り返す加速試験です。

詳細は半導体 品質/信頼性ハンドブックをご参照ください。

故障率(Failure Rate)(λ)

単位に Fit を用い、Failure in time の略称です。
単位時間当りに故障する確率で、「1Fit = 1×10^{-9}／時間」と表すことができます。
故障率は、あくまで予測値であり、使用条件(電圧、デューティなど)と使用環境(温度、湿度など)により、値は変ります。

詳細は半導体 品質/信頼性ハンドブックをご参照ください。

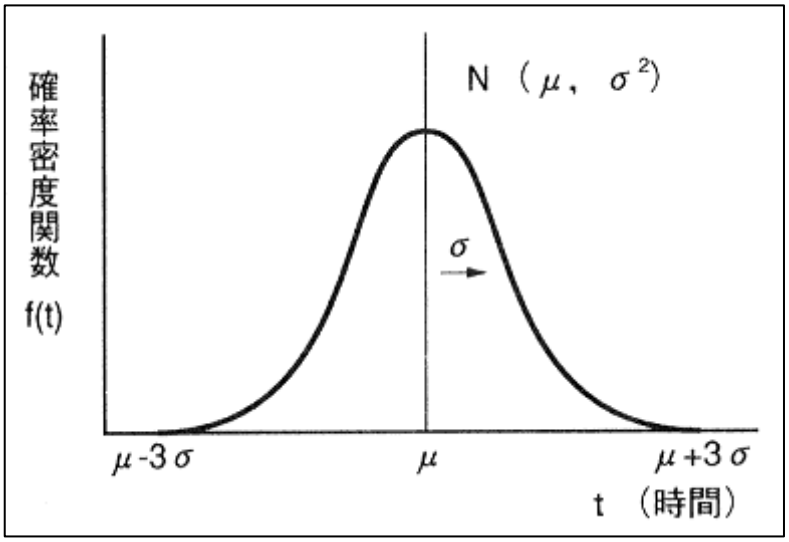
スクリーニング(Screening)

初期不良品や偶発不良品を選別して除去することです。製造工程でストレスを加えずに行う外観検査と、製造後に良品として出荷すべきものには劣化や損傷を生じない程度のストレスを加えて、潜在不良品だけを劣化させて除去する加速試験(エージング)があります。

詳細は半導体 品質/信頼性ハンドブックをご参照ください。

正規分布(Normal Distribution)

品質管理で最も多く使用する分布特性です。
正規分布は平均値 μ を中心とした左右対称のつり鐘形分布を示し、ほぼ 6σ(σ:標準偏差)を持ちます。



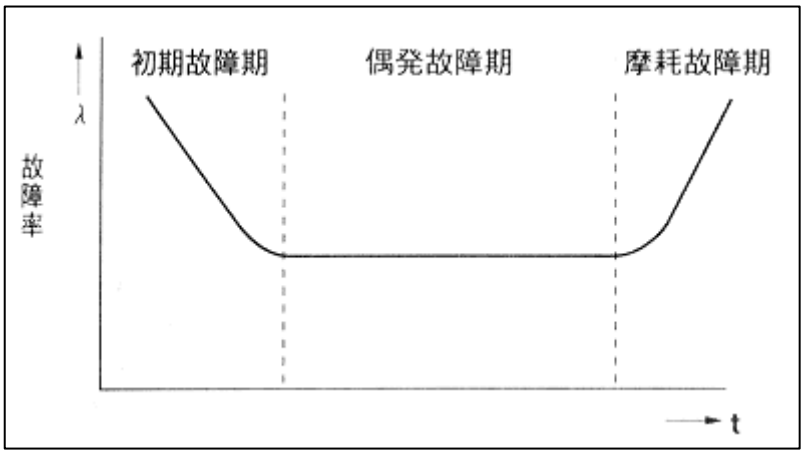
詳細は半導体 品質/信頼性ハンドブックをご参照ください。

バーンイン(Burn-in)

半導体デバイスの潜在欠陥を除去する目的で、100～125℃程度の高温状態で、一定時間電圧を印加することです。

バスタブ・カーブ(Bath-tub Curve)

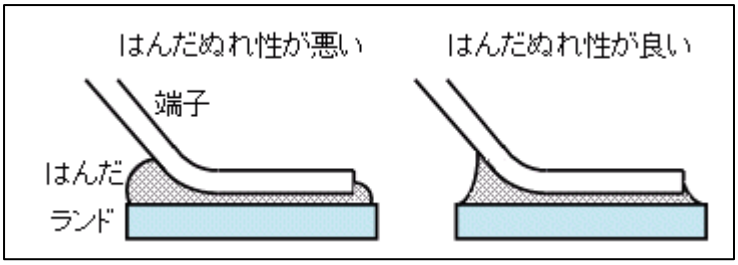
縦軸に故障率、横軸に時間をとると、故障率が西洋の浴槽に似た形となります。



詳細は半導体 品質/信頼性ハンドブックをご参照ください。

はんだぬれ性(Solder Wettability)

半導体デバイスの端子と基板実装面、それらを接続するはんだの間の付着の良さのことです。



ポアソン分布(Poisson Distribution)

ポアソンという学者が考えた、故障の発生確率理論による分布で、抜取検査の基礎です。

詳細は半導体 品質/信頼性ハンドブックをご参照ください。

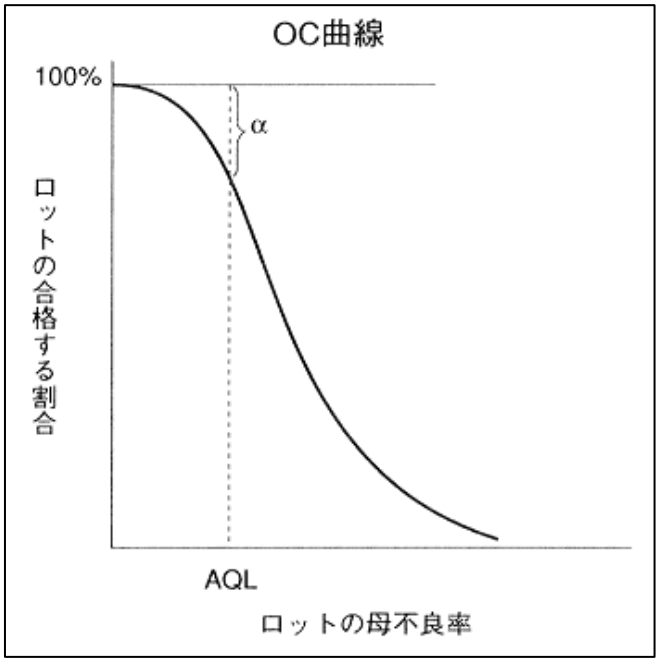
ワイブル分布(Weibull Distribution)

ワイブルという学者が、鋼の破壊強度などの分布を表すために使用した分布で、半導体デバイスの寿命解析にも多用されます。

詳細は半導体 品質/信頼性ハンドブックをご参照ください。

AQL (Acceptable Quality Level)(合格品質水準)

抜取り検査で、積極的に合格とする最低品質値（上限不良率）を表します。



詳細は半導体 品質/信頼性ハンドブックをご参照ください。

EMC (Electro Magnetic Compatibility)

電磁環境両立性を表します。

動作中の電気機器の内部で、電流が断続するとノイズが発生し、他の電気機器に影響して誤動作を起こす場合があります。これを電磁環境が悪化したといい、このような状況を防ぐようにするのが EMC です。

ESD (Electro Static Discharge)

静電気放電のことです。

半導体デバイスの静電気放電による破壊の要因としては、次の4つのモデルが提唱されています。

(1) 人体モデル

人体に帯電した電荷が、デバイスに触れたときに、放電を起こし発生する破壊モデル

(2) マシン・モデル

人体より大容量を有し、放電抵抗が小さい金属製機器とデバイスが接触したときに、発生する破壊モデル

(3) デバイス帯電モデル

デバイスのパッケージやリード・フレームが摩擦などにより帯電し、この電荷がデバイスの端子を通して放電され、発生する破壊モデル

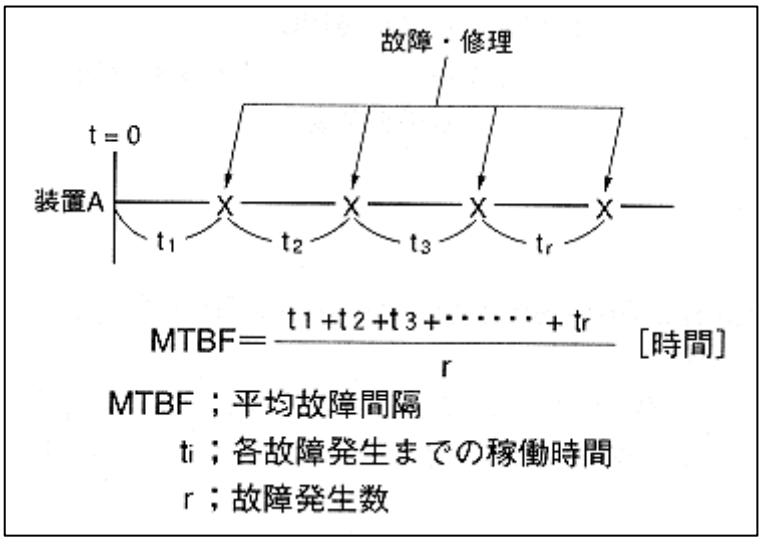
(4) 電界誘導モデル

MOS 構造を有するデバイスが、高電界中にさらされたときに、酸化膜破壊を起こすモデル

詳細は半導体 品質/信頼性ハンドブックをご参照ください。

MTBF (Mean Time between Failures)

コンピュータなど、装置機器の修理系の平均故障間隔のことです。



$$MTBF = \frac{t_1 + t_2 + t_3 + \cdots\cdots + t_r}{r} \quad [時間]$$

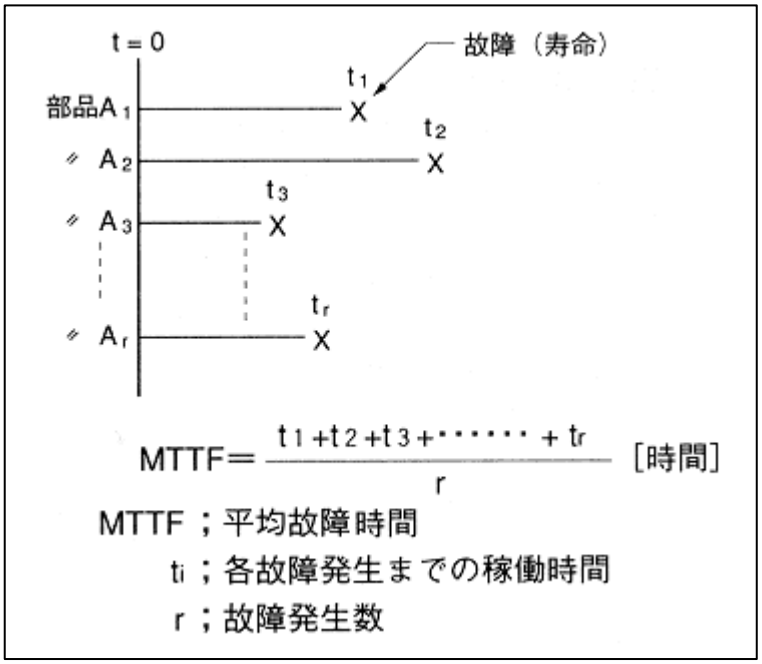
MTBF ；平均故障間隔

t_i ；各故障発生までの稼働時間

r ；故障発生数

MTTF (Mean Time to Failure)

部品等の非修理系の平均故障時間のことです。



$$MTTF = \frac{t_1 + t_2 + t_3 + \cdots\cdots + t_r}{r} \quad [時間]$$

MTTF ；平均故障時間

t_i ；各故障発生までの稼働時間

r ；故障発生数

2. 設計の予備知識

アーキテクチャ(Architecture)

「建築様式」の意味で、コンピュータでは基本設計を指します。マイコンでは、次のような項目が含まれます。

・データ・ビット幅
・アドレス空間とバンク構成
・命令セットとアドレシング・モード
・資源の割り当て(メモリやレジスタなど)
・パイプライン構成
・内部バス構成
・キャッシュ・メモリ構成

バンク構成は、アドレス空間を多面的に管理するものです。
アドレシングは、リード／ライトするデータのアドレスを指定する方法で、次のようなものがあります。

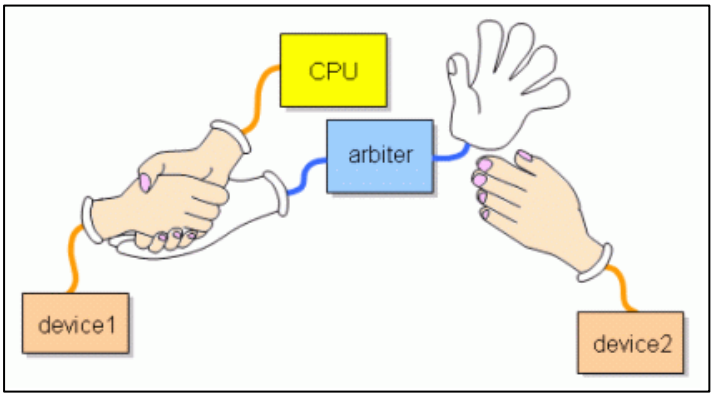
直接アドレシング:アドレス値を直接指定
相対アドレシング:ディスプレースメント(オフセット)を指定して、現アドレスとの相対アドレスを指定
間接アドレシング:レジスタの値で直接／間接にアドレシング

パイプラインは、処理ステージを数段用意して、命令フェッチやデータ処理を順次行う機構で、ある命令の処理が終わらないと次の命令を処理できないというボトルネックを解消するものです。
また、スーパスカラというパイプライン構成では、パイプラインを複数本用意して、並列処理を可能にします。
内部バス構成の特徴的なものとして、ハーバード・アーキテクチャがあります。これは命令フェッチとデータ転送のためのバスを独立させたもので、双方の待ちサイクルがパイプラインで発生することを回避します。
キャッシュ・メモリは、頻繁に使用するアドレス空間のデータだけをコピー管理する機構です。広大なアドレス空間を常時管理せずに、効率化を図るためのものです。キャッシュ領域でヒットしなければ、データをリプレース(入れ替え)します。

アービトレーション(Arbitration)

複数の能動素子がある資源を共有する場合などに、同時アクセスによる衝突を避けるために行う調停のことです。これにより、優先されたアクセスが終わってから、保留されていたアクセスが行われるようになります。

バス･アービトレーションという表現がよく用いられますが、これはマイコン・システムで、バスの使用権を調停することです。たとえば DMA では、DMA コントローラと CPU とのハンドシェークで、CPU をホールド状態にして、DMA コントローラがバスの使用権を得てデータ転送を行います。
3 つ以上のデバイス間では、バス・アービタがアービトレーションを行います。

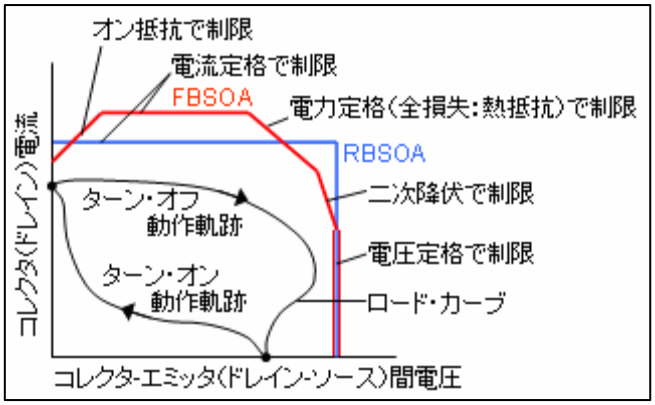


アイドル状態(Idle State)

システム的には、CPU が周辺デバイスにアクセスをしていない状態のことですが、双方では意味が異なります。
CPU 側では、周辺デバイスのリカバリ時間を確保するために、アイドル・サイクルを挿入して、そのデバイスをアクセスしないようにする期間のことです(関連用語:リカバリ時間)。
周辺デバイス側では、CPU がリードやライトのアクセスをしておらず、アクセス待ちの状態のことをアイドル状態といいます。この状態のとき、CPU はアクセスをすることができます。
なお、通信デバイスが送受信を行っておらず、回線にデータがない状態のこともアイドル状態といいます。このとき、回線はマーク状態(ハイ・レベル)やアイドル・パターンの送出になっています。

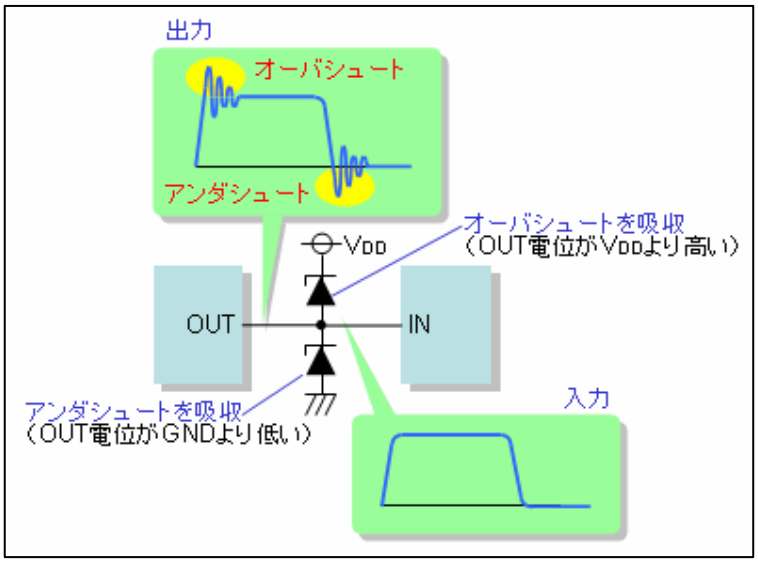
安全動作領域(Safe Operation Area:SOA)

バイポーラ・トランジスタや MOS FET で、スイッチング動作をさせる場合に、動作軌跡が電圧、電流、電力の制限領域内であることを確認するための定格です。
SOA には、順バイアス SOA(FBSOA)と逆バイアス SOA(RBSOA)があります。



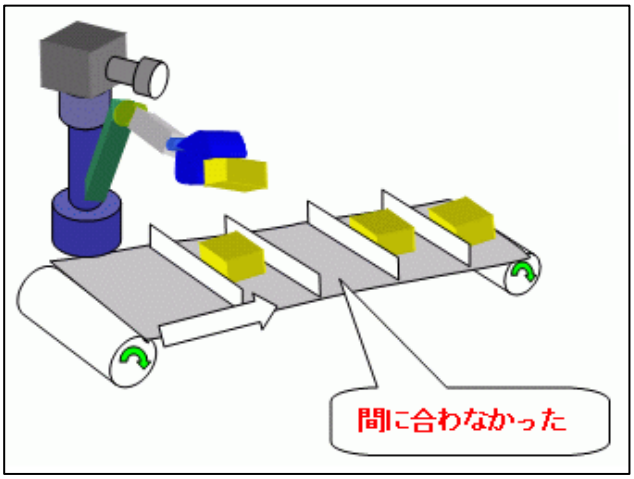
アンダシュート(Undershoot)

信号の立ち下がりで、定常的なロウ・レベルを瞬時的に下回る現象です。スイッチング時間の短い、高速デバイスで発生しやすくなります。アンダシュートやオーバシュートが絶対最大定格を越えると、ラッチアップや破壊を生じることがあります。通常は、ノイズ・クリッピング・ダイオードで吸収します。



アンダラン(Underrun)/バッファ・アンダラン(Buffer Underrun)

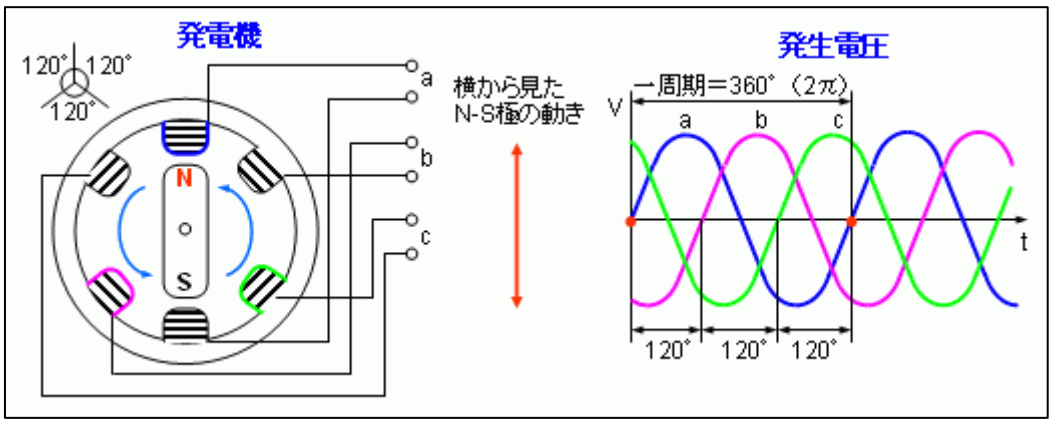
オーバランと逆に、必要なタイミングにデータが間に合わない状態がアンダランです。出力するデータをバッファに蓄え、あるタイミングで取り出す場合、順次データを取り出してバッファが空になって、次のデータが取り出せない状態をバッファ・アンダランと呼びます。
アンダランやバッファ・アンダランは、発生してはいけないエラー状態です。→オーバラン



なお、単にバッファが空になったのはバッファ・エンプティで、これはエラー状態ではありません。
アンダラン／バッファ・アンダランは、CD-R の書き込みのように、決められたタイミングでデータが必要な場合に発生して問題となります。

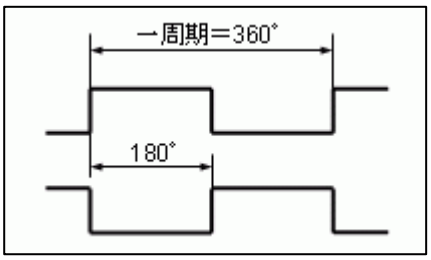
位相(Phase)

信号間のずれを意味します。一周期を 360° とし、相対的に何度ずれているかという表現をします。
これは、発電機のタービンが 1 回転して、一周期の正弦波が発生することに由来します。タービンの回転を横から見ると、磁石の先端が上下に往復し、これに伴って発生電圧も上下します。これを時間変化でタイミング・チャートにすると、正弦波となります。ここで、コイルの巻き方向が互い違いになっており、発生電圧の位相には 120° ずつのずれがあります。

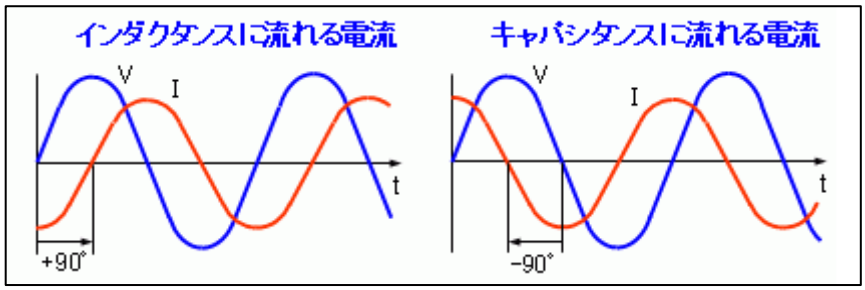


【ティー・ブレーク】
3 相交流で発電をしているので、送電線は 3 本一組になっており、家庭の電灯線はこのうちの 1 相を使用しています。工場などの動力系では、3 相を使用している場合があります。

周期的に同じ信号の変化をするというのは、このような回転系の概念から来るものですが、デジタル信号などにも位相の考え方が適用されています。たとえば、デューティ比 50%のクロック波形を反転させると、位相差は 180° になります。



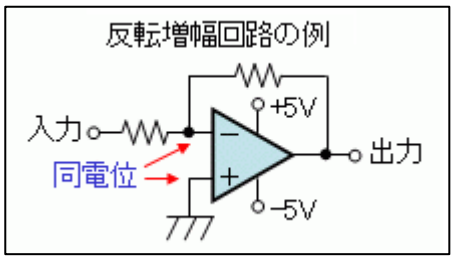
なお、電圧変化の位相について紹介してきましたが、電圧と電流の間にも位相が発生します。インダクタンスで電流が電圧より、位相は 90° 遅れ(＋)ます。またキャパシタンスでは、電流が電圧より、位相は 90° 進み(－)ます。



このため、合成抵抗であるインピーダンスによって、電圧と電流の位相が変化します。

イマジナリ・ショート(Imaginary Short)

オペアンプを利用した負帰還増幅器では、＋入力と－入力が同電位となるように動作するため、両端子間が短絡しているようにみなされ、この状態をイマジナリ・ショート(またはバーチャル・ショート：仮想短絡)と呼びます。
なお、イマジナル・ショートと呼ばれることもあります。



インストラクション・サイクル（命令サイクル）(Instruction Cycle)

マイコンが命令を実行するときのシーケンス全体のことで、次のような動作で構成されます。

命令フェッチ ：命令コードの読み取り
命令デコード ：命令コードの解析
命令実行 ：解析結果に基づく命令の実行

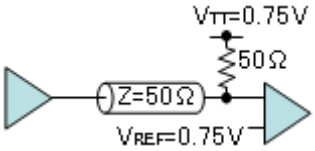
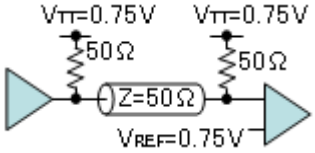
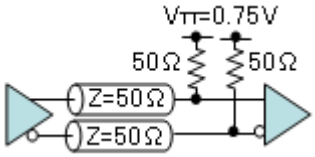
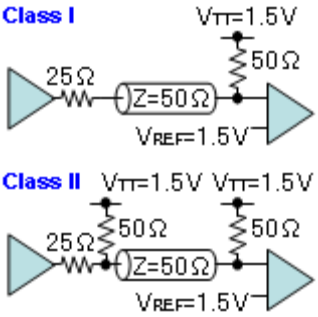
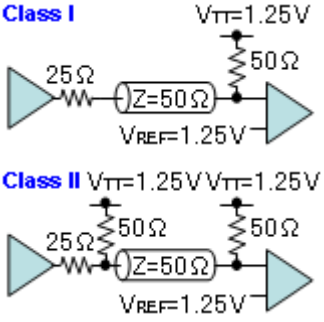
各動作はクロック・サイクルを基に進められるので、インストラクション・サイクルは各動作を合計したクロック数で表現されることがあります。ただし現在の RISC マイコンは、プリフェッチ・キューに命令コードを先読みしておいたり、パイプラインで複数の命令を並行処理したりすることが多く、命令のフェッチとデコードが外部で見えないため、実行サイクルだけのクロック数を表記することが一般的となっています。
また、命令実行についても、データのライトを待たずに、次の命令を実行するようなマイコンもあります。

インタフェース(Interface)

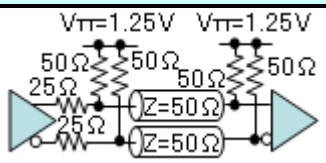
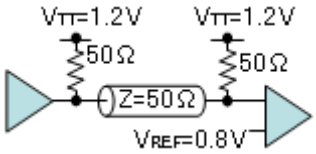
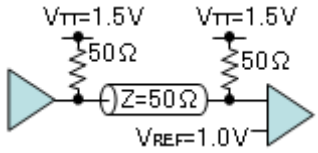
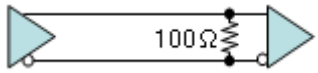
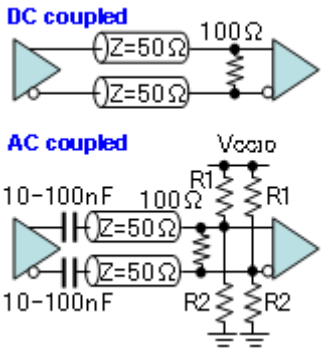
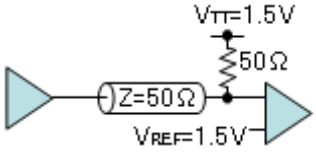
「つながり」に対する、送り側と受け側の整合性のことです。たとえば、人がことばで伝える場合、相手が聞こえる声量で、聞いているときに話さないと伝わりません。ちなみに、「マン・マシン・インタフェース」というのは、人が認識できる表示や音声・操作性などを装置が提供することです。

半導体デバイス同士を接続する場合、双方のシグナリングについて、電圧や変化タイミングを整合させる必要があります。シグナリングについては、TTL(Transistor Transistor Logic)レベルや CMOS レベル(「プルアップ／プルダウンのティー・ブレーク」参照)のほかに、次のような小信号・差動による高速インタフェースの例があります。

(1/2)

インタフェース	電圧	ターミネーション	規格	スピード例	応用例
LVTTL/LVCMOS (Low Voltage TTL/ Low Voltage CMOS)	3.3V		JEDEC JESD8-B	350MHz	
	2.5V		JEDEC JESD8-5	350MHz	
	1.8V		JEDEC JESD8-7	250MHz	
LVCMOS	1.5V		JEDEC JESD8-11	225MHz	
HSTL Class I (High-Speed Transceiver Logic)	1.5V		JEDEC JESD8-6	150-225MHz	QDRSRAM
HSTL Class II	1.5V			150-250MHz	QDRSRAM
Differential HSTL	1.5V			150-225MHz	CLK interface
SSTL-3 Class I, II (Stub Series Terminated Logic)	3.3V		JEDEC JESD8-8	167MHz	SDRAM
SSTL-2 Class I, II	2.5V		JEDEC JESD8-9A	160-400MHz	DDR I

(2/2)

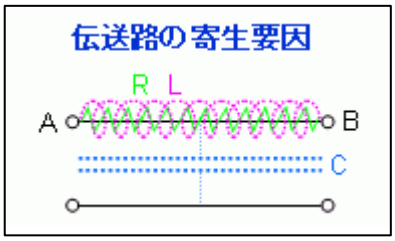
インタフェース	電圧	ターミネーション	規格	スピード例	応用例
Differential SSTL-2	2.5V			160-400MHz	DDR I SDRAM
GTL (Gunning Transceiver Logic)	2.5V 3.3V		JEDEC JESD8-3	200MHz	Backplane
GTL+	5V			133-200MHz	Backplane
LVDS (Low Voltage Differential Signaling)	3.3V		ANSI/TIA/ EIA-644	840Mbps	通信
LVPECL (Low Voltage Positive Emitter Coupled Logic)	3.3V 2.5V		ANSI/TIA/ EIA-612	840Mbps	SONET/SDH, Ethernet,1394
CTT (Center-Tap-Terminated)	2.5V 3.3V		JEDEC JESD8-4	200MHz	Backplane-bus
PCI (Peripheral Component Interconnect bus)	3.3V			66MHz	PC
PCI-X 1.0	3.3V			133MHz	PC
AGP 1x, 2x (Accelerated Graphics Port)	3.3V			66-133MH	画像処理

なお、装置間の接続では、識別やシーケンス、チェックなど通信手順(プロトコル)の整合も必要となります。

また、インタフェースにはシリアル(直列)とパラレル(並列)があります。シリアル・インタフェースは 1 ビットずつの転送で待ち行列が長くなりますが、パラレル・インタフェースでは同時に複数のビット・データを転送できます。たとえば 133MHz で 64 ビット・パラレルの PCI では、8.5Gbps(133M×64)もの帯域(Bandwidth)になります。ISDN では 64kbps 以上でブロードバンド(広帯域)と呼んでいますが、パラレル・インタフェースでは、圧倒的に単位時間当たりの転送データ量が多くなります。ただし、複数本の信号線が必要ですし、それらの同期化も課題となります。

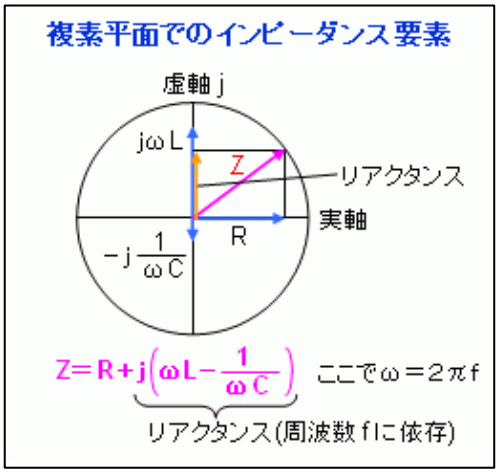
インピーダンス(Impedance)

直流に対する電気抵抗をR(Resistance：Ω)で表しますが、交流に対する電気抵抗はZ(Ω)で表し、これをインピーダンスと呼びます。ある信号線について、進行方向にはRだけでなくインダクタンスL(H)も寄生し、またグランドや他の信号線に対してはキャパシタンスC(F)が、それぞれ寄生します。

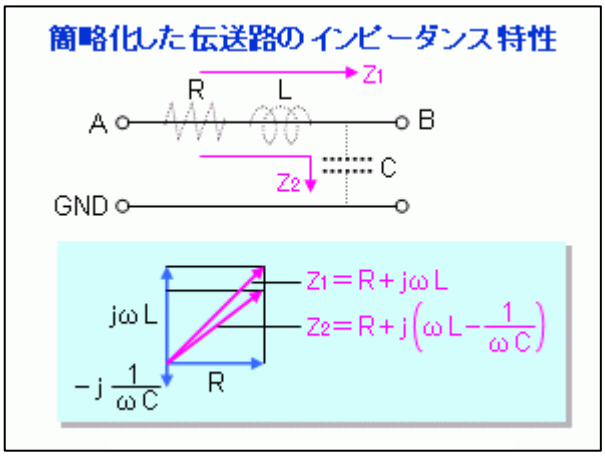


虚数j($j^2=-1$)を縦軸とする複素数で表現すると、Lのインピーダンスは$j\omega L$、Cのインピーダンスは$-j(1/\omega C)$となります($\omega=2\pi f$)。このため、直流(f=0Hz)に対しては、Lのインピーダンスは0、Cのインピーダンスは無限大となります。また交流に対しては、周波数fが高くなるにつれて、Lのインピーダンスが大きく、Cのインピーダンスが小さくなります。

なお、LとCの合成インピーダンス$j(\omega L-1/\omega C)$をリアクタンスと呼びます。一般にインピーダンスと呼んでいるのは、実数部であるRと虚数部であるリアクタンスを合成したものです。



AB間の伝送線と対グランド間の寄生要因を簡略化して下図のように考えると、進行方向には$(R+j\omega L)$、グランドに対しては$(R+j(\omega L-1/\omega C))$のインピーダンス特性となります。



このため、直流については、AB間のインピーダンスが実数部のRだけで、グランドに対して絶縁状態となります。また、交流については周波数が高いほど、AB間の抵抗が大きくなり、さらにグランドに対するCの抵抗が小さくなって、信号が減衰しやすくなります。

インピーダンス整合(Impedance Matching)

電気信号の伝送路で、出力側のインピーダンスと、これを受ける入力側のインピーダンスを一致させることで、この操作をインピーダンス変換と呼びます。反射をなくして効率のよい伝送をするには、重要なことです。入出力双方の実数部 R を合わせ、伝送路を含めて虚数部j(ωL−1/ωC)をゼロにすれば、もっとも効率の良い整合となります。
インピーダンス変換には、次のような手法があります。

(1) 抵抗による

高周波回路で LC による整合をさせる場合、フィルタ回路となるため、共振周波数付近でしか整合できなかったり、過度の利得で不安定になる可能性があります。このようなときには、抵抗による整合が行われます。
低周波回路では、反射が問題になることはなく、低インピーダンスからの出力を高インピーダンスの入力へ接続してもかまいません。逆に、高インピーダンスからの出力を低インピーダンスの入力へ接続すると、過電流による破損やひずみを生じる可能性があります。このようなときには、入力側に直列に抵抗を挿入して、整合が行われます。ただし、エネルギ消費の許容範囲を考慮する必要があります。

(2) コイルとコンデンサによる

LC を組み合わせて整合させるもので、L と C の接続形態で、L 型、T 型、π 型などがあります。

(3) トランジスタによる

トランジスタは接地形態によって、入力側と出力側のインピーダンスが異なるので、これを利用してインピーダンス変換ができます。オーディオ・アンプの最終段では、スピーカ(4-32Ω ほどの低インピーダンス)を接続するために、コレクタ接地回路(エミッタフォロア回路)がよく採用されます。

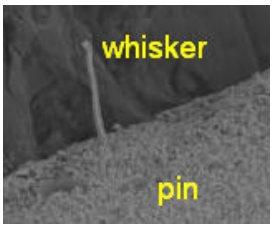
接地形態	入力インピーダンス	出力インピーダンス
コレクタ接地	高	低
エミッタ接地	中	中
ベース接地	低	高

(4) トランスによる

トランス(変圧器)は、一般に交流電圧の変換に使用されますが、変成器としてインピーダンス変換にも使用できます。ただし、おもに真空管オーディオ・アンプ(出力インピーダンスが数kΩ と高い)にスピーカを接続するために用いられ、最近は見かけなくなりました。

ウィスカ(ホイスカ)(Whisker)

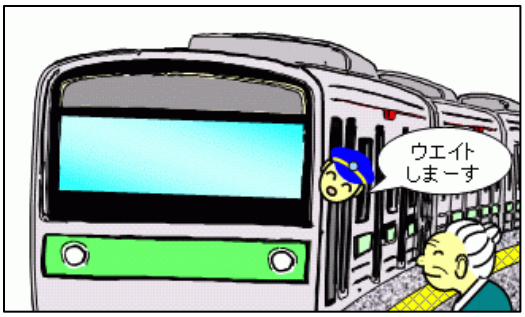
Sn(すず)などのメッキ材が、ヒゲ状に成長する現象で、経時変化によってピン間ショートの原因となることがあります。最近は、鉛フリー化による Sn-Bi(すず-ビスマス)の採用で、問題となる場合があります。
当社では、鉛フリー品も含めて、ウィスカが発生しない条件で、製品の製造をしています。



ウエイト(Wait)

CPU がメモリや I/O デバイスをアクセスするときに、リード／ライトできる状態になるまで待ち合わせをすることです。CPU が高速で、メモリや I/O デバイスが低速の場合に、ウエイト制御が必要になります。
つまり、CPU が一方的にリード／ライトしようとしても、メモリや I/O デバイスのアクセス時間が経過するまではアクセスできないため、ウエイト・サイクルを挿入して、ストローブ信号のアクティブ期間を延長させます。

電車にたとえると、ドアの開いている期間が短すぎると、乗り降りができないのと同様です。電車の発車時刻だといって、一方的にドアを閉めると、乗り遅れる人が取り残されますが、乗客の乗り降りが終わるまでドアを開けておけば、取り残される人はいなくなりますね。この「ドアの開いている間」は、リード・ストローブ信号やライト・ストローブ信号がアクティブの期間に相当します。

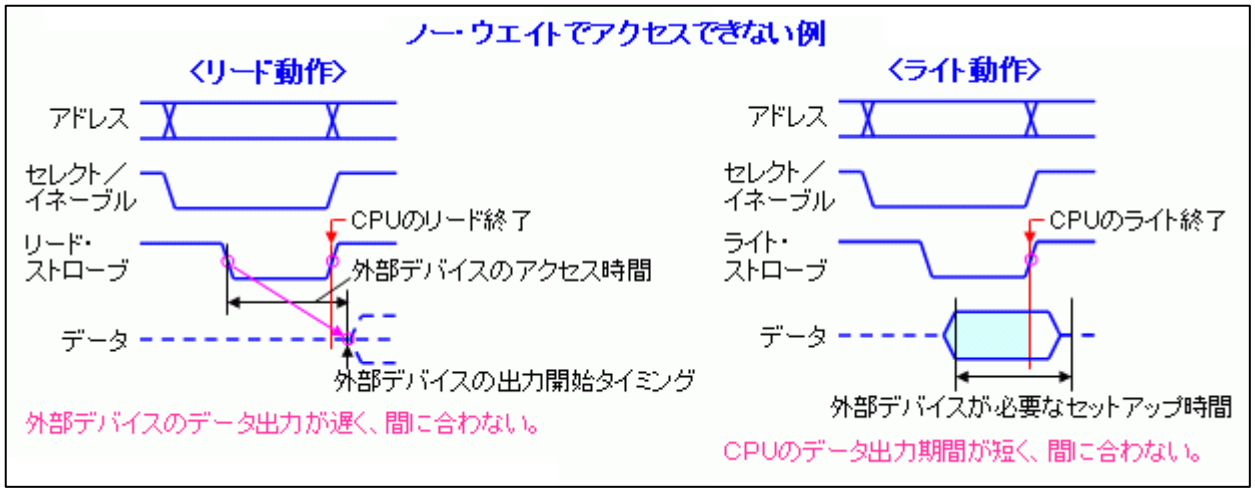


リード・ストローブ信号がアクティブになると、メモリや I/O デバイスからデータがデータ・バスに出力されますが、それを待たずにリード・ストローブ信号をインアクティブにすると、CPU はデータをリードできません。
また、ライト・ストローブ信号がアクティブになると、CPU がデータ・バスに出力しているデータがメモリや I/O デバイスに入力されますが、それを待たずにライト・ストローブ信号をインアクティブにすると、メモリや I/O デバイスにデータをライトできません。

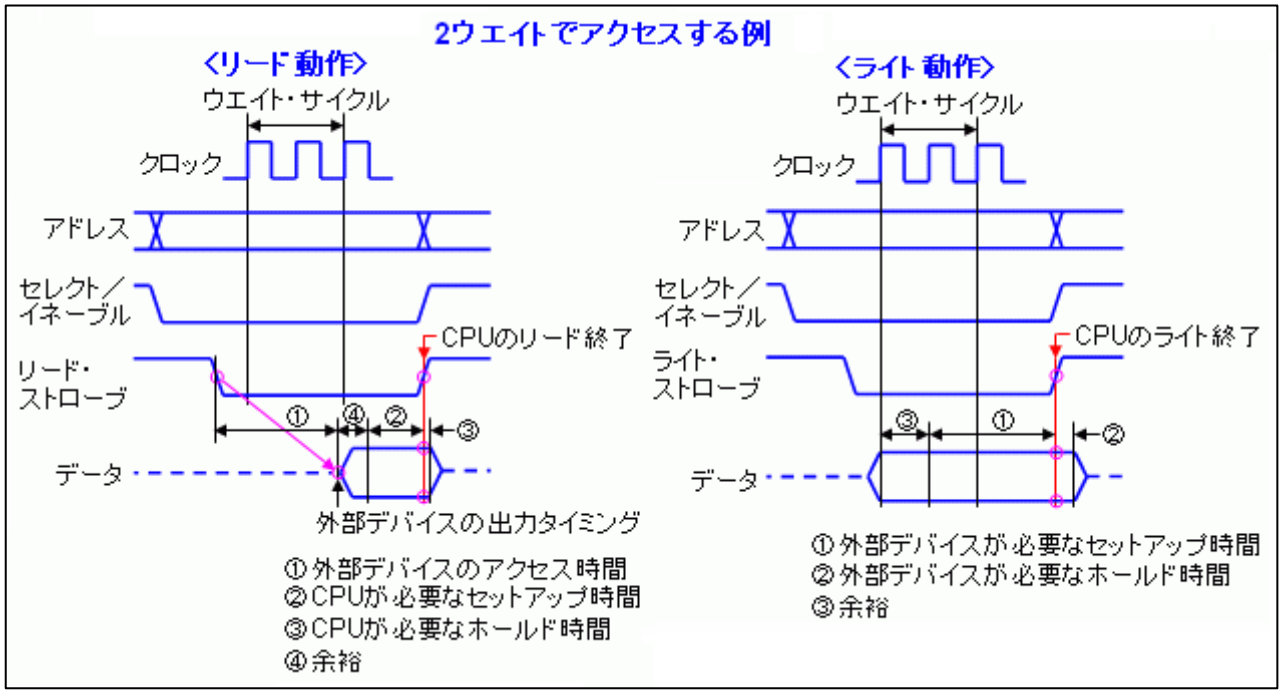
実際のリード動作では、ストローブ信号がインアクティブになるタイミングに対して、事前にデータが確定している必要があり、これをセットアップ（設定）時間と呼びます。外部デバイスのアクセス時間と、CPU のセットアップ時間を合計した期間だけ、リード・ストローブ信号をアクティブにしておく必要があります。
ライト時には、外部デバイスのセットアップ時間だけ、ライト・ストローブ信号をアクティブにする必要があります。

余分なウエイト・サイクルを挿入すると、リード／ライト・サイクルが無意味に長くなって、システム速度が低下しますので、各外部デバイスのアクセス時間に合わせたウエイト制御の最適化が、システムの効率化に必要です。

余談ですが、ストローブ信号がインアクティブになってから、短時間ですが、データを固定しておく必要があり、これをホールド（保持）時間と呼びます。ただし、ホールド時間はデバイスの特性で決まっており、ウエイト制御とは無関係です。



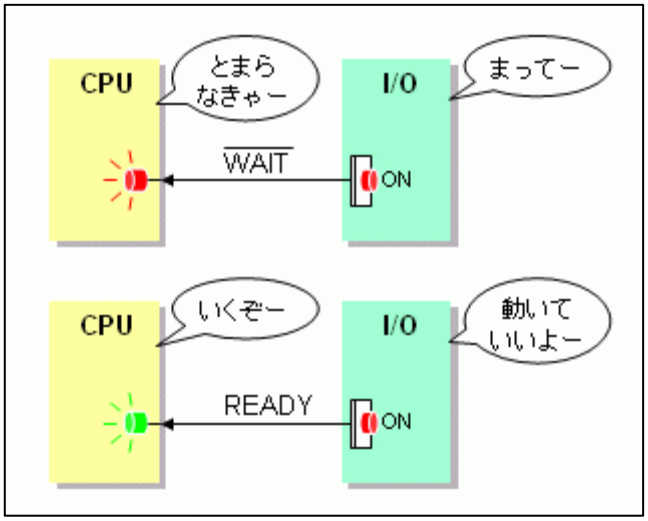
メモリや I/O デバイスのアクセス・スピードに応じて、挿入するウエイト・サイクル数を制御します。このサイクルは、クロック・サイクルで決定されます。また、ウエイト制御には、WAIT 信号によるハードウエア制御（WAIT がアクティブの間、ウエイト・サイクルを挿入）と、レジスタ設定によるソフトウエア制御（設定クロック数だけ、ウエイト・サイクルを挿入）があります。



なお、ウエイトに似た機能にレディがあります。通常、ハードウエア制御で入力をチェックする場合、ウエイトでは負論理の WAIT(−)信号がアクティブ（ロウ・レベル）なら CPU がウエイト状態になり、レディでは正論理の READY 信号がインアクティブ（ロウ・レベル）なら CPU がウエイト状態になります。つまり、CPU の動作は同じですが、ウエイトは CPU が待機する考え方で、レディは外部 I/O デバイスが動作可能状態を通知する考え方です。

ウエイトは周辺に合わせてできるだけ早く動作させることに主眼を置き、レディはシステムを確実に動作させることに主眼を置いていると言えます。

CPU の WAIT(−)信号と READY 信号は同義の場合もありますが、もともとレディ制御は、周辺の準備ができたら動作を開始するという、ハンドシェークのような待ち合わせ手順です。

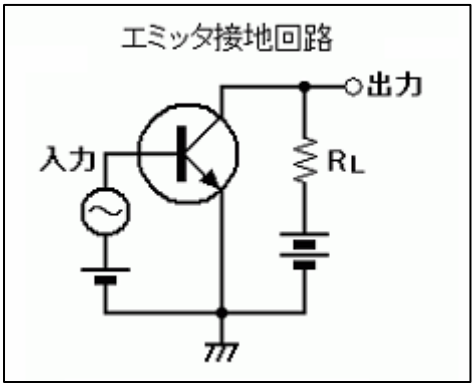


エミッタ遮断電流(Emitter Cut-off Current)

コレクタをオープンとして、エミッタ・ベース間の PN 接合に逆方向電圧をかけた時の逆方向電流のことです。
電気的特性の略号表記は I_{EBO} です。

エミッタ接地回路(Emitter Common Circuit)

エミッタを共通接地として、入力をベースに加え、コレクタに負荷抵抗を付け、その両端から出力を取り出します。
トランジスタは電流増幅素子なので、電圧増幅で動作させるときは、**Vo=RL×Ic** として出力電圧を得ます。
最も一般的に使われる回路で、入力インピーダンスも比較的高く、入出力の位相が反転します。
動作バイアス点と負荷抵抗の選び方が設計のポイントです。



関連項目 コレクタ接地回路(Collector Common Circuit/Emittor Follower Circuit)
ベース接地回路(Base Common Circuit)

エンディアン(Endian)

16ビット幅以上のデータを1バイト(8ビット) ごとに区切って、メモリ・アクセスやデータ転送する場合に、アドレスと対比させる順序のことで、バイト・オーダとも呼びます。
アドレスの小さい方から、「上位バイト→下位バイト」の順に配置する場合をビッグ・エンディアン(Big Endian)、「下位バイト→上位バイト」の順に配置する場合をリトル・エンディアン(Little Endian)と呼びます。

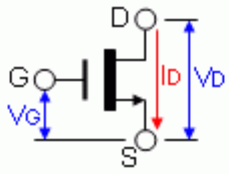
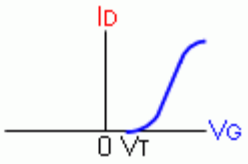
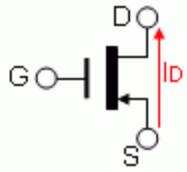
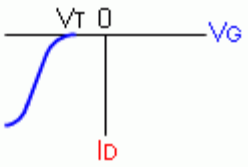
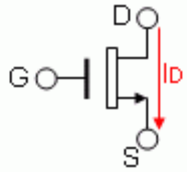
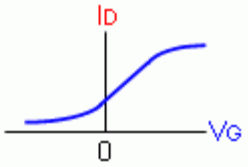
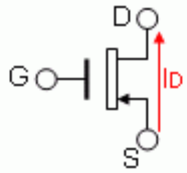
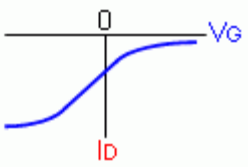
たとえば 0x12345678 という 32 ビット・データでは、それぞれ次のような定義になります。

アドレス	0	1	2	3
ビッグ・エンディアン	0x12	0x34	0x56	0x78
リトル・エンディアン	0x78	0x56	0x34	0x12

インテルの 86 系 CPU はリトル・エンディアンですが、モトローラの 68 系 CPU はビッグ・エンディアンです。エンディアンの切り替えができるデバイスもあります。
Endian という名称は、ガリバー旅行記の中の、ゆで卵をどちらの端から割るかという議論に由来しています。この物語では、大きい方の端から割る人々を Big End-ian、小さい(とがった)方の端から割る人々を Little End-ian と呼んでいました(-ian は「～な人々」という接尾辞)。

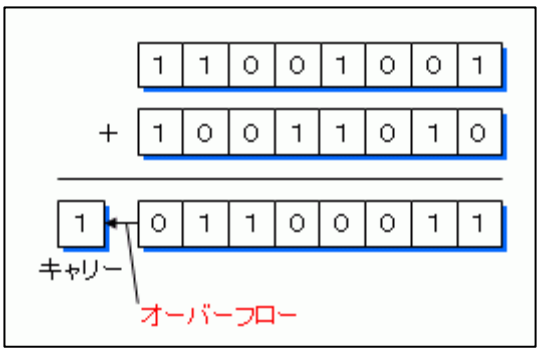
エンハンスメント(Enhancement)

MOS FET の分類のひとつで、ゲート電圧を印加しないと、ドレーン電流が流れない特性を持ちます。ドレーン電流が流れはじめるゲート電圧をスレッシュホールド(しきい値)電圧 V_T で規定します。
今日では、エンハンスメント形が一般的です。

タイプ	記号	入出力特性
Nチャネル・エンハンスメント		
Pチャネル・エンハンスメント		
Nチャネル・デプレッション		
Pチャネル・デプレッション		

オーバフロー(Overflow)

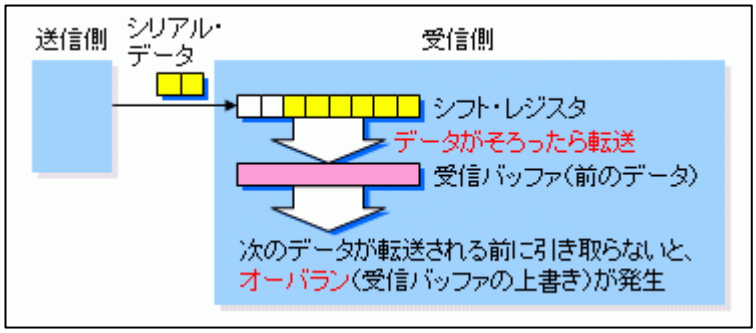
①計算の結果が、桁あふれをおこすことをいいます。桁あふれが発生すると、キャリー・フラグがセットされます。



②それ以外に、バッファがいっぱいになった(あふれた)場合も、「(バッファが)オーバフローした」といった使い方をします。
計算でのオーバフローは、桁があふれただけなので、桁上げして、引き続きキャリー・フラグを利用して計算をすればいいのですが、バッファがオーバフローした場合は、データがなくなってしまいます。
関連 FAQ:バッファ・オーバフロー、アンダラン/バッファ・アンダラン

オーバラン(Overrun)

データをやりとりする場合、送り側が受け側の処理速度を越えて送ってくると、受け側は正しいデータを受け取れません。この状態をオーバランと呼びます。たとえば、シリアル通信 UART では、1 ビットずつシリアルに送られてきたデータはシフト・レジスタに取り込まれ、パラレル・データに変換され、必要なビットが揃うとバッファ・レジスタに転送されます。バッファ・レジスタからこのデータが読み出される前に、次のデータがバッファ・レジスタに転送されると、前のデータが上書きされます。このとき UART では、フラグをセットしてオーバランを通知します。



参考 FAQ：オーバフロー②

オーバシュート(Overshoot)

信号の立ち上がりで、定常的なハイ・レベルを瞬時的に上回る現象です。スイッチング時間の短い、高速デバイスで発生しやすくなります。オーバシュートやアンダシュートが絶対最大定格を越えると、ラッチアップや破壊を生じることがあります。通常は、ノイズ・クリッピング・ダイオードで吸収します。

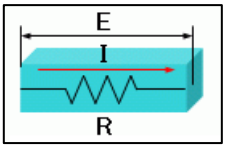
オームの法則(Ohm's Law)

ドイツのゲオルク・ジーモン・オームが発表した、電圧と電流の関係を表した法則です。電気工学で最も有名な法則のひとつで、エジソンの発明にもよく利用されました。ある材料の両端の電圧を E(V)、その両端間の電気抵抗を R(Ω)、そこに流れる電流を I(A)とすると、オームの法則の関係式は次のとおりです。

E＝IR

I＝E/R

R＝E/I

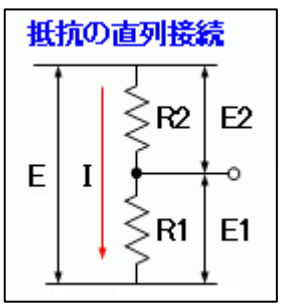


これらの式から分かるように、電流が一定であれば、電圧は抵抗に比例します。

これを利用すれば、直列抵抗の両端の電圧は、次のように算出できます。

E1＝E・R1/(R1＋R2)

E2＝E・R2/(R1＋R2)



これを分圧と呼びます。これより、全体の電圧は、次のように算出できます。

E＝E1＋E2＝I･R1＋I･R2＝I･(R1＋R2)

R＝E/I なので、合成抵抗は次のとおりです。

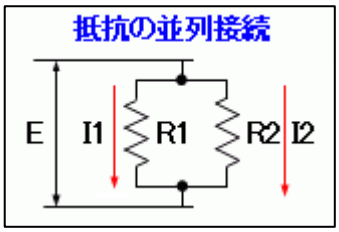
R＝R1＋R2

なお、全体の電圧が各電圧降下の総和になるという定理を「**キルヒホッフの第二法則**」(電圧則：KVL)と呼びます。これは、後述の第一法則とともに、ロシアのグスターヴ・キルヒホッフが発見したものです。

また、電圧が一定であれば、電流は抵抗に反比例します。これを利用すれば、並列抵抗に流れる電流は、次のように算出できます。

I1＝E/R1

I2＝E/R2



これを分流と呼びます。これより、全体の電流は、次のように算出できます。

I＝I1＋I2＝E(R1＋R2)/R1R2

R＝E/I なので、合成抵抗は次のとおりです。

R＝R1R2/(R1＋R2)

このことは、次の式からも検証できます。

I＝E/R＝E/R1＋E/R2　これを E で除算する

1/R＝1/R1＋1/R2＝(R1＋R2)/R1R2

R＝R1R2/(R1＋R2)

なお、3 つ以上の抵抗の並列接続では、合成抵抗の逆数が、次のようにすべての逆数の和になります。

1/R＝1/R1＋1/R2＋1/R3＋･･･

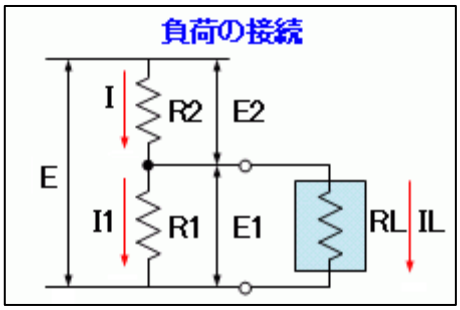
なお、分流があっても、流入電流の総和と流出電流の総和が等しいという定理を「**キルヒホッフの第一法則**」(電流則：KCL)と呼びます。

ここで、抵抗分割による分圧で、負荷に電圧を供給することを考えて見ましょう。

負荷がなければ、先に述べたとおり、**E1＝E･R1/(R1＋R2)**となります。

でも、負荷を接続すると負荷電流も流れるため、R1と負荷抵抗RLが並列接続で**R0＝R1RL/(R1＋RL)**となります。

つまり、R2とR0の直列接続となり、このように電源と抵抗の直列接続で、回路を等価変換することを「**鳳(ほう)テブナンの定理**」と呼びます。



並列接続のため、負荷に供給される電圧は、次のようになります。

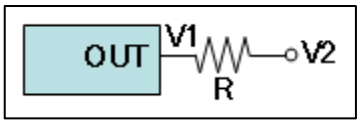
E1＝E･(R1RL/(R1＋RL))/(R1RL/(R1＋RL)＋R2)＝E･R1RL/(R1RL＋R2(R1＋RL))

＝E･R1/(R1＋R2(R1/RL＋1))

ここで、RLがほとんど電流の流れない高抵抗であれば、R1/RL≒0となり、R1とR2の分圧でE1が得られます。

でも、RLの抵抗が小さいほど、E1が低下することになります。

★ 「**オームの法則**」は、あくまでも抵抗に電流が流れた場合に、その両端の電位差（電圧降下）を表します。**E＝IR** なので、電流が流れない（**I＝0**）ときには、電位差がありません。つまり、次図のように R の一端がオープン状態では、**V2＝V1** となります。これに負荷が接続されると、R と負荷の直列抵抗となり、負荷に応じて V2 が変化します。



「**オームの法則**」は簡単な関係式ですが、当然、半導体デバイスにも、この法則が適用されます。デバイスの内部抵抗は、電流と電圧の規格から算出できます。また、外部にプルアップ抵抗を接続する場合には、出力のドライブ能力（電流と電圧の規格）から、抵抗の最小値が算出できます。

ただし、半導体の抵抗は一定ではなく、印加電圧によって変化します。このことは、PN接合で構成されるダイオードのV-I特性からお分かりいただけると思います（抵抗が一定なら直線グラフになる）。

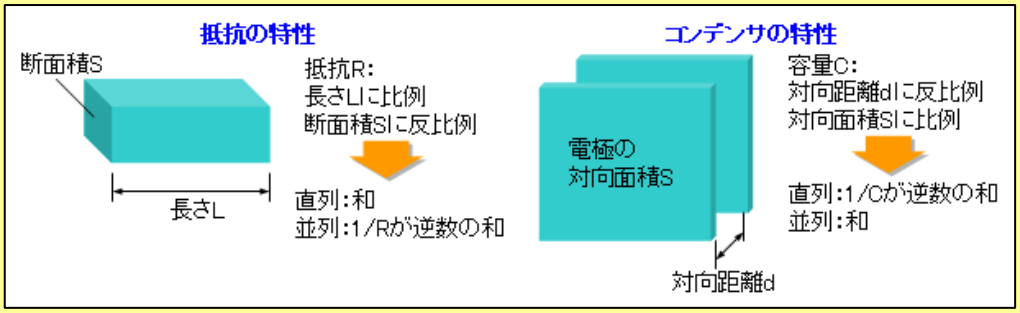
★ **【ティー･ブレーク 1】**

直列抵抗の合成は和、並列抵抗の合成の逆数は各逆数の和となることを紹介しましたが、コンデンサの容量については、逆の関係になります。つまり、次のとおりです。

直列容量：**1/C＝1/C1＋1/C2＋1/C3＋･･･**

並列容量：**C＝C1＋C2＋C3＋･･･**

これは、コンデンサに蓄えられる電荷 Q が **Q＝CV** で表されるため、**V＝(1/C)･Q** となり、**E＝IR** と対比して、R が 1/C に置き換えられるためです。これは、抵抗とコンデンサそれぞれの構造的な特性によるものです。



【ティー･ブレーク 2】

電気抵抗の逆数で、電気の伝わりやすさの表現があり、実部と虚部、合成部分について、それぞれ次のように定義されています。

	単位	実部	虚部	合成部
抵抗	オーム(Ω)	レジスタンス:R	リアクタンス:X	インピーダンス:Z
伝わりやすさ(1/抵抗)	ジーメンス(S)	コンダクタンス:G	サセプタンス:B	アドミタンス:Y

MOS FET では、アドミタンスで伝達特性を表します。

なお、時間の秒を表す単位は"s"（小文字）または"sec"です。ジーメンスは大文字の"S"で表しますので、混同しないよう、ご注意ください。

音声／音楽コーデック(Speech/Music Codec)

音声／音楽のアナログ信号をデジタル化して、符号(Code)化して情報量を少なくすることを**エンコード(Encode)**と呼びます。逆に符号化されたデータから元のアナログ信号に戻す復号化を**デコード(Decode)**と呼びます。また、音声／音楽に対するエンコードとデコードを合わせて、**音声／音楽コーデック**といいます。

符号化すれば、デジタル・データを圧縮することによって、音質をあまり落とさず、通信（有線／無線）で転送したり媒体（HDD／フラッシュ・メモリなど）に記録するデータ量を削減できます。

このとき、人の聴覚の特性を次のように利用します。

- 可聴周波数は20Hz～20kHz。また、人の声のほとんどの周波数成分は、さらに狭く3～4kHz以下。
  電話では話者の識別と内容が理解できればよく、周波数が4kHz以下の音でも十分です。そこで、それ以上の周波数のデータを削除できます。
- 音程や音量には敏感だが、音の位相には鈍感なので、位相成分のデータを削減できる。
- 大きい音があると、小さい音が聞き取りにくくなる（マスキング効果）ので、大きな音と時間的／周波数的に近接した、小さい音のデータを削減できる。
- 中音域より低音域や高音域の感度が低いので、低音域や高音域のデータを削減できる。

これらの方法はデータを削減するので、復号化で正確に元の音には復元できません（可逆性なし）が、人が聞いても、音質の劣化はほとんど目立ちません。

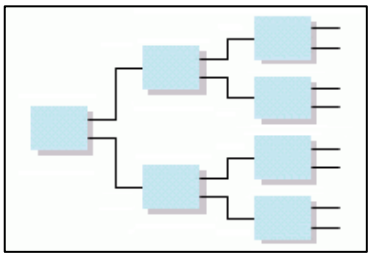
ところで、符号化した機器と異なる機器（通信装置、パソコン、携帯型機器）でも復号化が可能なように、これらの方法を組み合わせた音声／音楽の符号化方式と復号化方式は、一般的に規格化されており、IP電話などでよく使用されているADPCM方式、MDで使用されているATRAC方式、DAP（デジタル・オーディオ・プレーヤ）でよく使用されているMP3方式、AAC方式などがあります。

カスケード（縦続）接続(Cascade Connection/Cascading)

複数チャネルを持ったあるデバイスの各チャネルに、同種のデバイスを直列（出力を次段の入力）に接続する接続形態です。チャネル数をネズミ算式に拡張できます。

以前は、割り込みコントローラでの割り込み要求の拡張で使用されていました。

現在では、Ethernet や USB、IEEE1394 などで、ストリーム・ポートの増設に使用されます。



トランジスタを使用した増幅回路では、エミッタ接地とベース接地の組み合わせをカスケード接続と呼びます。

画像コーデック(Video Codec)

映像／画像のアナログ信号をデジタル化して、符号(Code)化して情報量を少なくすることを**エンコード(Encode)**と呼びます。逆に符号化されたデータから元のアナログ信号に戻す復号化を**デコード(Decode)**と呼びます。また、映像／画像に対するエンコードとデコードを合わせて、**画像コーデック**といいます。

符号化すれば、デジタル・データを圧縮することによって、画質をあまり落とさず、通信（有線／無線）で転送したり媒体（HDD／フラッシュ・メモリなど）に記録するデータ量を削減できます。

このとき、人の視覚の特性を次のように利用します。

・輝度より色に対して鈍感なので、色の情報量を減らせる。

・止まっている物には鈍感なので、一部のフレームでは動いている物の差分だけを符号化する。また、物の動きは一定方向が多く、動きが予想できるので、その予想値だけを符号化して、情報量を削減する。

これらの方法はデータを削減するので、復号化で正確に元の画像には復元できません（可逆性なし）が、人が見ても、画質の劣化はほとんど目立ちません。

ところで、符号化した機器と異なる機器（通信装置、パソコン、携帯型機器）でも復号化が可能なように、これらの方法を組み合わせた映像／画像の符号化方式と復号化方式は、一般的に規格化されています。たとえば、デジタル・スチル・カメラなどで使用されてる静止画の符号化方式には JPEG 方式、デジタル TV や DVD などで使用されている動画の符号化方式には MPEG2 方式（当社製品 EMMA2 に搭載）などがあります。また、最近では次世代 DVD 用に高圧縮率の H.264 方式（当社製品 EMMA3 に搭載）などがあります。

過渡応答(Transient Response)

一般に、外部から制御系に変化を与えて、その結果が定常状態に達するまでの経過を過渡応答といいます。電気回路では、入力の変化に対して、出力の変化が定常状態になるまでの時間のことです。

たとえば、バイポーラ・トランジスタのスイッチング時間や、ダイオードの逆回復時間は、過渡応答の例です。

トランジスタのスイッチング時間は、「ターンオン時間 t_{on}」と「ターンオフ時間 t_{off}」で規定され、それぞれ入力と出力の電圧/電流が最大振幅の 10%、90%の点で測定します。

また、これらを決定するパラメータは、次のとおりです。

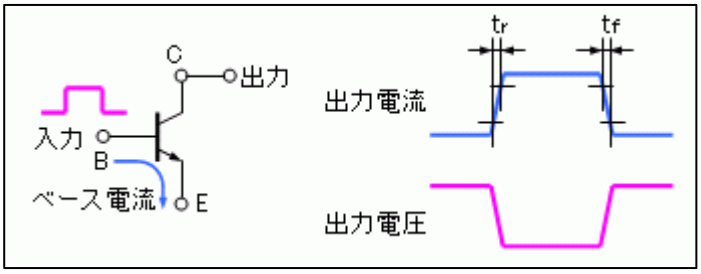
遅延時間 t_d ：パルス入力が 10%になってから、出力電流が 10%に増加（電圧が 90%に降下）するまでの遅延

上昇時間 t_r ：出力電流が 10%から 90%へ増加（電圧が 90%から 10%に降下）するまでの時間

蓄積時間 t_{stg} ：パルス入力が 90%になってから、出力電流が 90%に減少（電圧が 10%に上昇）するまでの時間

下降時間 t_f ：出力電流が 90%から 10%へ減少（電圧が 10%から 90%へ上昇）するまでの時間

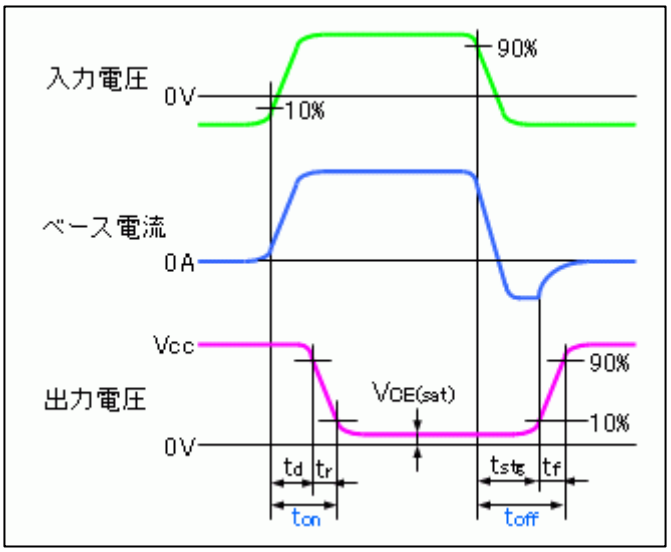
これらのパラメータは、電流で定義されていますが、入出力波形は慣例的に電圧で表現されるため、出力の概念が反転します（電流が増加すると、電圧が低下）。



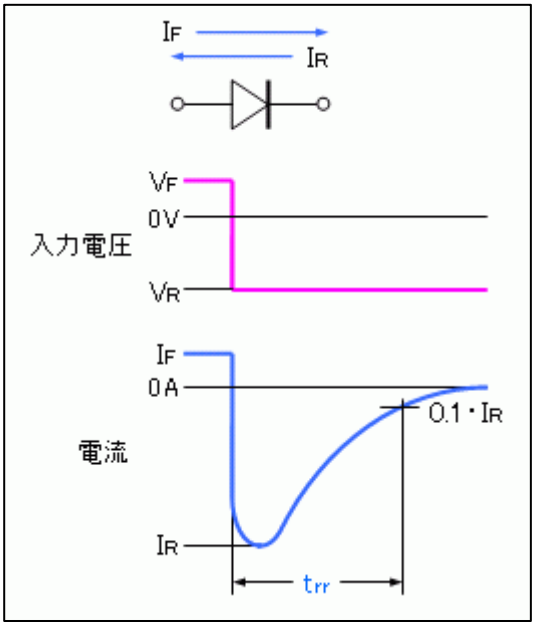
これらによって、スイッチング時間は次の定義となります（t_r 、t_f が電流定義であることに注意）。

ターンオン時間 $t_{on}=t_d+t_r$

ターンオフ時間 $t_{off}=t_{stg}+t_f$

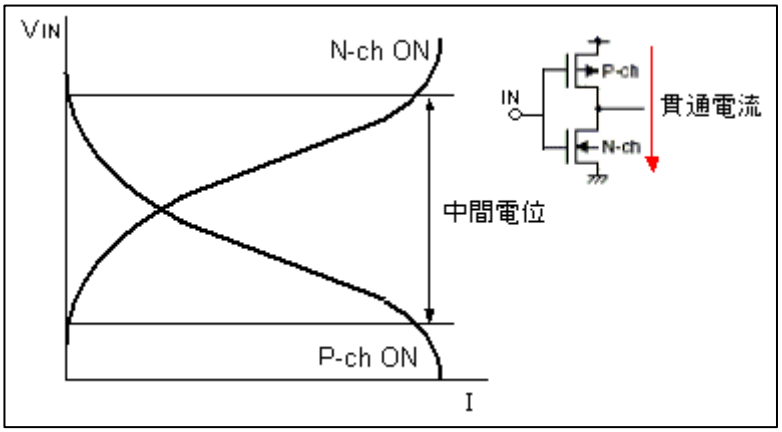


ダイオードの逆回復時間は、入力電圧が順バイアス V_F から逆バイアス V_R に反転してから、逆電流の最大振幅の10%まで回復するまでの時間 t_{rr} で規定されます。



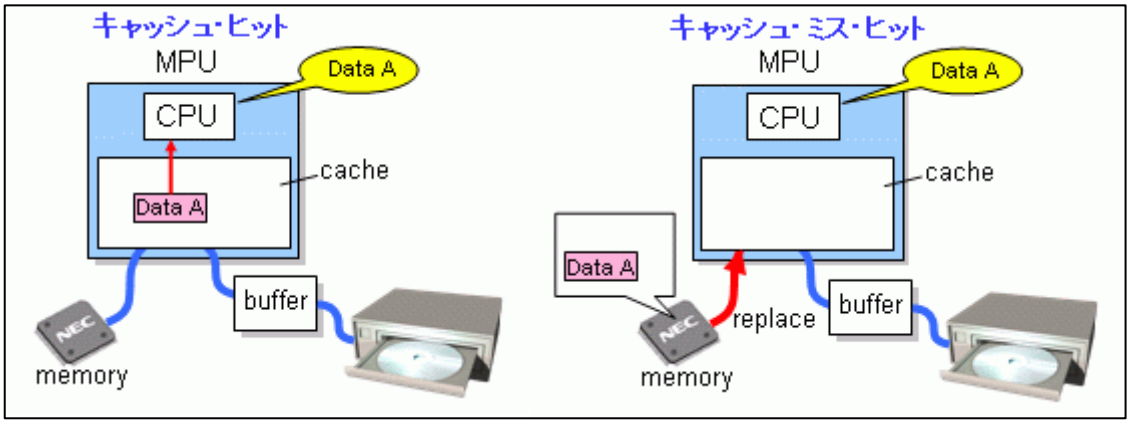
貫通電流(Through Current)

CMOS デバイスに、ハイとロウの中間の電圧を入力すると発生します。CMOS は PMOS と NMOS の組み合わせですが、ともにある電圧で完全に OFF となるのではなく、中間電位では両方が ON 状態となります。このため、電源からグランドへ電流が突き抜けてショート状態となり、この電流を貫通電流と呼びます。
立ち上がりや立ち下がりの遅い入力では、中間電位の入力時間が長いため、貫通電流が発生して内部電源が弱くなり、誤動作につながる恐れがあります。
また、未使用の CMOS 入力がオープンになっている場合も、入力がアンテナとなってノイズを拾い、貫通電流が流れる要因となります。貫通電流はショート状態で発生しますから、劣化を促進させます。



キャッシュ・メモリ(Cache Memory)

CPU が、メモリだけでなく、磁気ディスクや CD-ROM などの外部記憶装置も含めた広大なデータ領域を頻繁にアクセスすると、長い時間を要して非効率的です。そこで、キャッシュ・メモリという高速アクセスできる内部メモリを用意して、アクセス頻度の高いデータを外部から読み込んで、ここに展開しておきます。通常は、ユーザ・プログラムがキャッシュ・メモリの存在を意識する必要はありません。CPU はまずキャッシュ・メモリをアクセスして、目的のデータがあれば（キャッシュ・ヒット）そのアクセスを行い、目的のデータがなければ（キャッシュ・ミス・ヒット）外部から該当データのあるブロックを読み込んでリプレースし、そのデータをアクセスします。

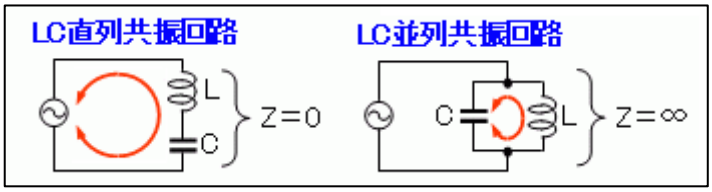


また、キャッシュ領域でも、さらにアクセス頻度の高いブロックを一次キャッシュとして独立させ、二次キャッシュとの 2 段構成で、より効率化を図ったマイコンもあります。
キャッシュ・システムではリプレースが発生するので、シングルチップ・マイコンのような、ある程度の容量の内部メモリでの高速アクセスが基本的な目的のデバイスでは、かえって処理効率が低下するので採用されません。

共振(Resonance)

振動する物体間でエネルギの授受をするとき、ある振動数で、授けるエネルギと受けるエネルギが等しくなって、エネルギ保存の法則で外部からエネルギが与えられなくても振動を持続するようになり、この現象が**共振**です。このとき、外部からエネルギが与えられると、振動が大きくなります。また、そのときの振動数を**固有振動数**(Eigenfrequency)と呼びます(Eigen はドイツ語で own の意味)。
振り子やブランコは、運動エネルギと、振れたときに高くなる位置エネルギの間で、エネルギの授受が生じます。固有振動数に合わせてブランコをこぐと、ゆれを持続したり大きくしたりできます。このとき、振幅が大きくなると、ブランコは高い位置まで上がりますが、速度も速くなり、乗っている人の体重にかかわらず、振動数は一定です。この固有振動数は、支点からの距離で決まります。

電気回路では、水晶振動子やセラミック発振子の圧電効果を利用した発振回路や、コイルとコンデンサを用いた LC 共振回路が代表的な例です。LC 共振回路では、コンデンサ内部に電界として蓄えられたエネルギと、コイル内部に磁界として蓄えられたエネルギが、固有周波数で等しくなって、相互にエネルギ授受ができるので、外部からエネルギ供給がほとんどなくても、電気エネルギの授受(振動)が持続するようになります。このとき、外部から見たエネルギ消費は、配線抵抗 R(≒0)での発熱と電磁波の放射だけです。この状態で、外部から LC 共振回路に固有振動数(周波数)の電気振動を与えると、大きな振動(**発振**)をするようになります。この原理は、テレビやラジオの同調回路(設定周波数に共振する電波を拾う選局)などに利用されています。
コイルとコンデンサを直列に接続(直列共振回路)すると、固有振動数で互いの電位が相殺され、外部からはインピーダンスが 0 に見えます。また、並列に接続(並列共振回路)すると、互いの電流が相殺され、外部からはインピーダンスが無限大に見えます。LC フィルタは、並列共振回路で、特定の周波数ノイズを選択的に除去します。

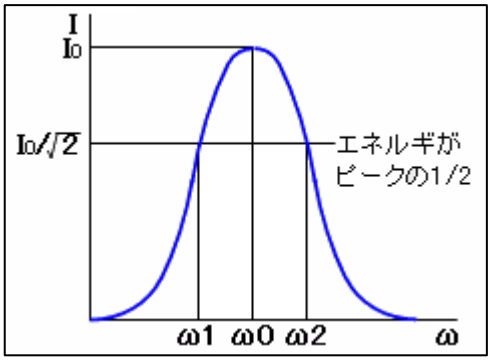


ここで、インピーダンスが最小(リアクタンスが 0)のときの共振周波数は次のようになります。

$$\omega L - 1/\omega C = 0$$
$$\omega = 2\pi f_0 = 1/\sqrt{LC}$$
$$f_0 = 1/2\pi\sqrt{LC}$$

この共振の鋭さを Q 値(Quality Factor/Value)で表します。

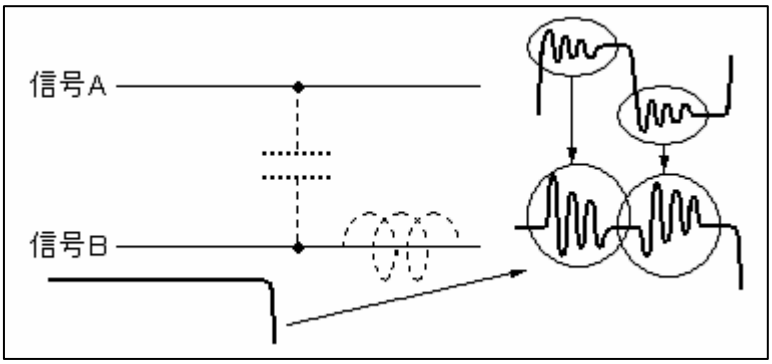
$$Q = \frac{\omega 0}{\omega 2 - \omega 1} = \frac{1}{R}\sqrt{\frac{L}{C}}$$



発振回路ではこれが大きいほど、正確で安定します(LC 発振は数十、水晶は数百万)。ただし、Q 値が大きくなるにつれて、応答性が悪く、起動に時間がかかります。

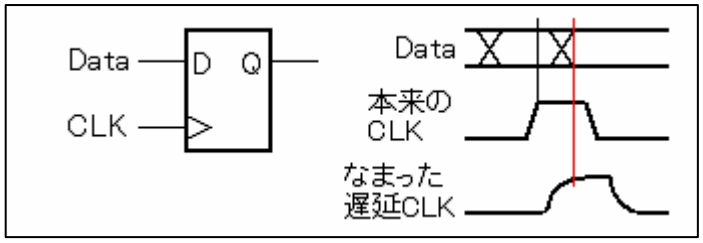
クロス・トーク(Cross Talk)

信号線の変化がノイズとなって、他の信号線に乗ってしまう現象です。
たとえばオーバシュートやアンダシュートなどのリンギングが発生すると、信号線間のキャパシタンスによって、ノイズが伝播しやすくなり、これが信号線のインダクタンスの影響で大きくなります。



クロック・スキュー(Clock Skew)

多段回路などでクロック同期を取る場合、クロック線の引き回しが長くなるクリティカル・パスや、論理ゲートがクロック線に設置されているゲーテッド・クロックがあると、クロック遅延を生じて、セットアップ時間やホールド時間を満足できずに、クロックが正規のタイミングでデータをサンプリングできなくなる現象です。

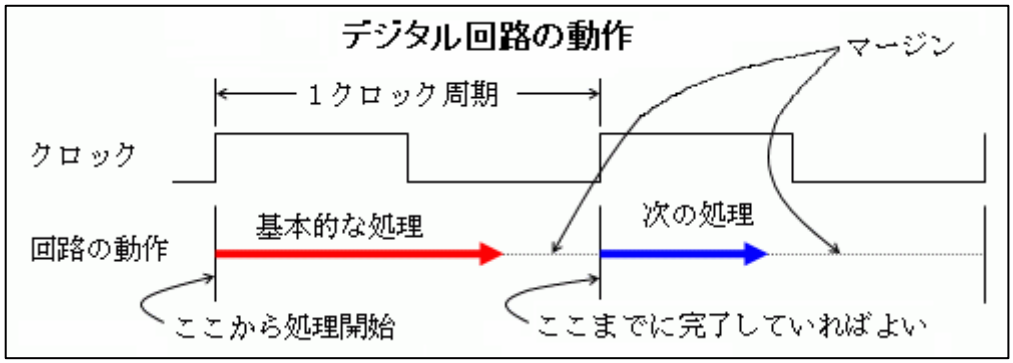


クロックとリセット(Clock And Reset)

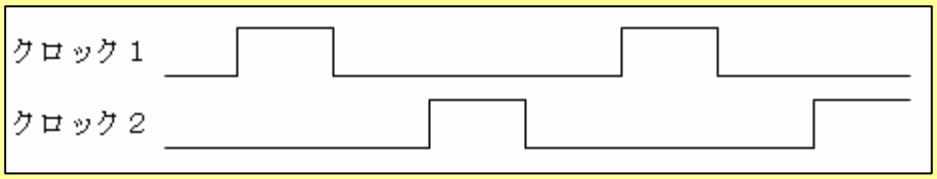
クロックとリセットはマイコンを含めて、デジタル回路では基本的な信号です。

[クロック]

簡単なゲート回路を除いて、デジタルの回路はクロックに同期して動作します。1クロックの間に基本的な処理を行い、これを繰り返すことで必要な動作をします。このように動作をクロック単位に分ければ、複雑な動作を安定に実行させることができます。



【コーヒー・ブレーク】
LSIの中では、上のような単純なクロックを用いることはまれで、1周期の中で位相が異なる複数のクロックを使い分けるのが一般的です。下図に、2相の場合のクロック例を示します。このように、2 つのクロックが重ならないようになっています。

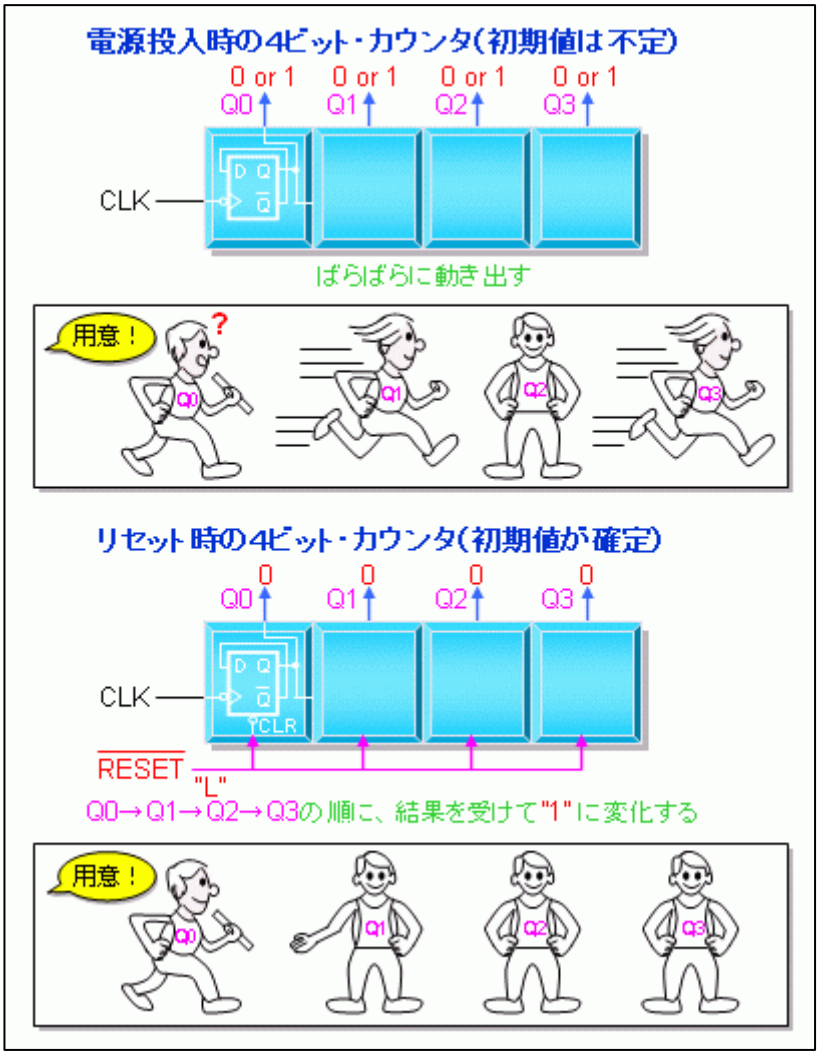


このように、クロックは動作の基準タイミングとなる信号ですが、それ以外にも、次のような使われ方をします。

- タイマ・カウンタでカウントして、必要な時間間隔（インターバル）を得る
- 入力された信号からノイズをとる
- 入力された信号のエッジ検出を行う

[リセット]

クロックと同様に、基本的な信号としてリセットがあります。内部に F/F や順序回路をもったデジタル回路では、電源投入時に F/F や順序回路の状態が不定となります。これをある状態に初期化するために、使用するのがリセット信号です。そのため、リセット信号がきちんとしていないと、その後の正常な動作は期待できません。

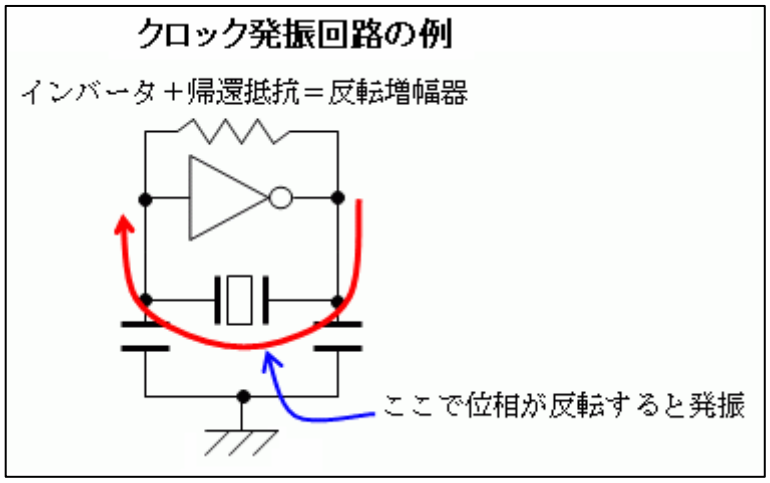


古いデバイスではリセット信号をクロックでサンプリングしていて、リセット信号の幅として 2～3 クロック分必要などの規格が定められていたものもありました。そのため、電源投入時に電源電圧が動作電圧になり、クロック発振が安定してから規定された時間経過してからリセットを解除するような手順が必要でした。
最近のマイコンではこのような規格はなくなりました。これは、省電力モードとして、クロックの発振そのものを停止するような STOP モードをサポートするようになったからです。STOP モードを解除する手段の一つがリセットですので、クロックがないと受け付けられないようでは使い物になりません。
それどころか、78K0 や 78K0S などのマイコンでは、リセットがかかるとクロック発振を止めてしまいます。
リセットは、おもに電源投入時の初期化のために使用されます。このため、電源電圧を監視して、ある電圧になるまではリセットをかけるようなリセットICと呼ばれる専用のICもありましたが、最近では POC（パワー・オン・クリア）回路として、マイコンに内蔵されるようになってきています。
それ以外では、プログラムの実行が異常になった場合に、正常な動作状態に復帰させるためのウォッチドッグ・タイマでもリセットがかけられます。FAQ の「ウォッチドッグ・タイマの基本」もご参照ください。

クロック発振安定時間(Clock Oscillation Stabilization Time)

[はじめに]

マイコンの動作に、クロックは欠かせないものです。通常は、マイコンに内蔵された発振回路に、水晶振動子やセラミック発振子とコンデンサを接続して発振させるか、外部で発振させたクロックを供給します。
水晶振動子やセラミック発振子を用いた発振は、一般にインバータに帰還抵抗を付加することで、構成した反転増幅器の出力を水晶振動子やセラミック発振子で、位相を反転(180 度ずらす)して入力に戻すことで実現します。



[発振安定時間]

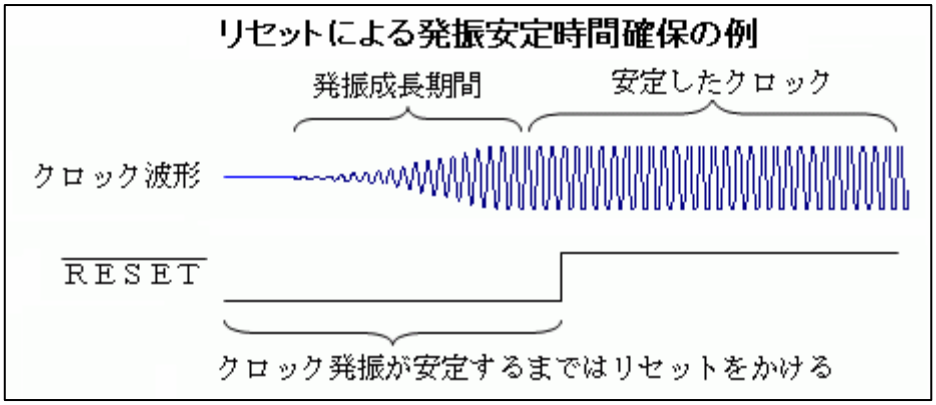
発振回路でのクロック発振は、発振を開始してもすぐに安定するわけではありません。水晶振動子やセラミック発振子で決まる周波数が徐々に大きくなり、やがて安定した発振となります。この発振成長期間を発振安定時間と呼びます。この間は振幅が小さく、高い周波数の成分がかなり多く含まれており、マイコンを動作させるクロックの規格を満足できません。そこで、発振が安定するまではクロックを使用しないようにする必要があります。



【コーヒー・ブレーク】
発振安定時間は、使用する発振子の種類や発振周波数、回路のマッチングの状況で異なります。数 MHz の発振の場合、正しくマッチングしていれば、水晶振動子では ms のオーダで、セラミック発振子では 100μs のオーダで、発振は安定します。周波数が低い時計用の 32.768kHz の発振では、発振安定時間が秒のオーダになることがあります。

[発振安定時間の確保]

発振安定時間の確保には2つの方法があり、マイコンによりどちらかになります。1つはマイコンが自動的に確保する場合で、もう1つは使う側で意識して確保する必要がある場合です。78K0 や 78K0S のような 8 ビットのマイコンでは、ほとんどマイコン自身で発振安定時間を確保するようになっています。一方、V850 では、電源投入時やリセットで STOP モードを解除するときは、発振安定時間はリセット信号の幅で確保する必要があります。



リセット時の発振安定時間をマイコンが自動的に確保する方式は、使用するうえでは簡単ですが、自由度が低くなります。電源立ち上げやリセット解除後のシステムの立ち上がり時間が問題となるようなときは、発振安定時間をできるだけ短くする場合があります。また、外部からクロックを供給するときのように、発振安定時間を確保する必要がない場合もあります。このような場合には、リセット信号で発振安定時間を確保する方法が適しています。

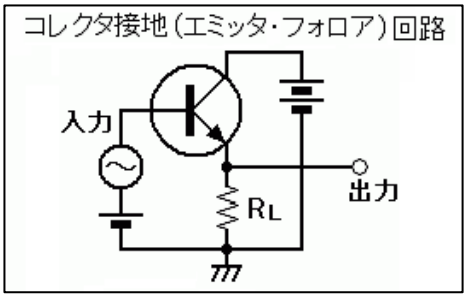
【コーヒー・ブレーク】
78K0/Kx2 などのマイコンでは、起動を早くするために、発振安定時間の短い発振器を内蔵し、起動時にはその内蔵した発振器で起動しています。
78K0S/Kx1+では、使用するクロックを選択できる機能があり、外部からクロックを供給する場合や、内蔵の発振器を利用する場合は、発振安定時間を確保しないようになっています。
V850 で PLL 内蔵のマイコンでは、発振安定時間以外に、PLL がロックするまでの時間が必要です。

コレクタ遮断電流(Collector Cut-off Current)

エミッタをオープンとして、コレクタ・ベース間の PN 接合に逆方向電圧をかけたときの逆方向電流のことです。電気的特性の略号表記は I_{CBO} です。

コレクタ接地回路(Collector Common Circuit/Emitter Follower Circuit)

エミッタと共通接地の間に、負荷抵抗をつなぎます。コレクタを直接電源につないで、交流的には共通接地です。エミッタ・フォロア回路とも呼ばれ、入力インピーダンスが高く、出力インピーダンスは低く、電流利得が大きいので、負荷に電流を多く流しても出力電圧の変動がほとんどないため、重い負荷をドライブするのに向いています。



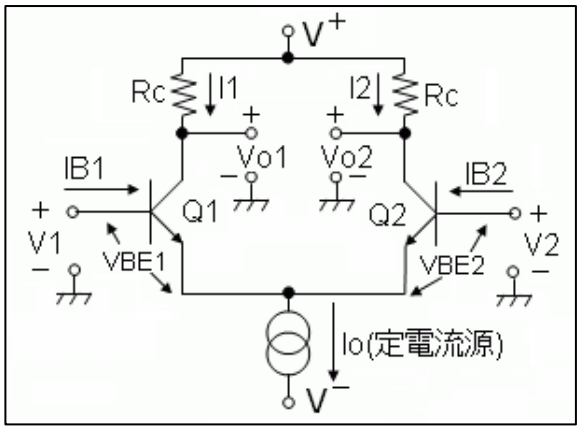
関連項目：エミッタ接地回路(Emitter Common Circuit)、ベース接地回路(Base Common Circuit)

コレクタ飽和電圧(Collector Saturation Voltage)

トランジスタのベース電流を大きくしていくと、コレクタ電流はその h_{FE} 倍流れます。ところが、いくらコレクタ電流を流そうとベース電流を大きくしても、コレクターエミッタ間の電圧はゼロにならず、いくらかの電圧が残ります。
これがコレクタ飽和電圧 $V_{CE}(sat)$ です。コレクタ飽和電圧 $V_{CE}(sat)$が大きいと、電源電圧の一部が $V_{CE}(sat)$にとられ、スイッチング動作では、オン状態での残留電圧となります。
特に大電力動作では効率が悪くなり、発熱の要因となるので、$V_{CE}(sat)$はなるべく小さいことが望まれます。
コレクタ飽和電圧 $V_{CE}(sat)$については、共通 FAQ の「アナログとデジタルの違い」で図示しています。

差動増幅回路(Differential Amplification Circuit)

オペアンプなどの入力回路に内蔵される差動増幅回路は、2 つの入力信号の差だけを増幅するもので、次図のように、全く対称的な構成となっています。



2 つの入力電圧 V1、V2 が等しく、各抵抗とトランジスタが全く等価な特性だと仮定すると、コレクタ電流 I1、I2 は等しく、出力電圧 Vo1、Vo2 は同じ値となり、差動回路は全くバランスした状態となります。

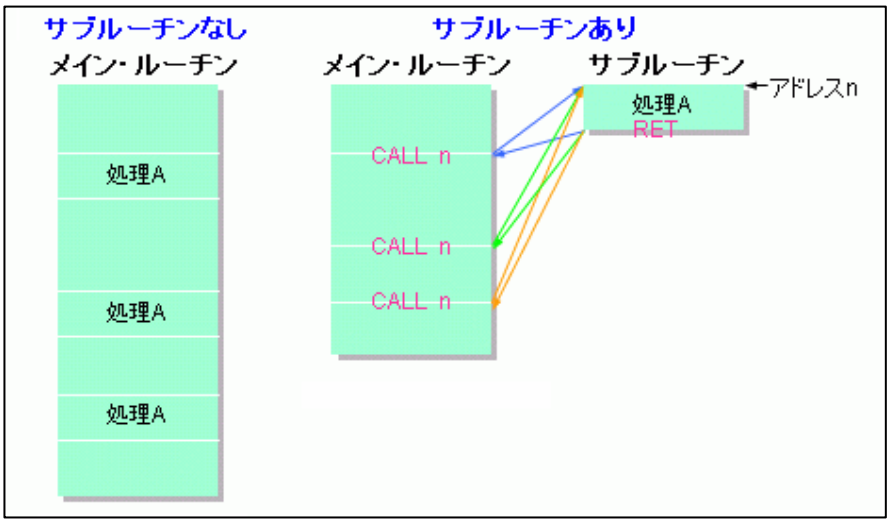
$$Vo1 = Vo2 \fallingdotseq V^{+} - Io \cdot Rc/2$$

次に V1 が増大し、Q1 の入力電流 IB1 が増大すれば、それに従って I1 も増加するので、出力電圧 Vo1 は低下する方向となります。ところが、I1 と I2 の総和は常に定電流源 Io の値に等しいため、必然的に I2 は減少し、Vo2 は増加する方向となるので、Vo1≠Vo2 となってバランスがくずれます。このとき、差動出力電圧 Vo1－Vo2 が生じ、この電圧は差動入力電圧 V1－V2 に比例します。
また、V2 も V1 の増加量と等しく増大すれば、再び回路はバランス状態となり、差動出力電圧は生じません。
このように差動増幅回路には、入力 2 端子間の電圧差(差動成分)だけに感応するという性質があります。

サブルーチン(Subroutine)

マイコンのプログラミング・テクニックのひとつで、あるまとまった処理プログラムをメイン・ルーチンから参照するように切り出したものです。同じ処理が何度も行われる場合に有効です。サブルーチンの利点は、プログラム・サイズが短縮できることと、その処理プログラムの変更が1ヶ所で済むことです。なお、サブルーチン・コールの命令には、プログラムを参照する CALL 命令のほかに、テーブル参照の CALLT 命令などがあります。

実際のサブルーチンの参照では、CALL 命令で分岐して、処理の最後に RET 命令でメイン・ルーチンに戻ります。このとき、CALL 命令でプログラム・カウンタ PC の内容をスタックに退避させ、RET 命令でそれをスタックから PC に復帰させます。なお、V850 シリーズでは、ハード的なスタック領域の概念がなく、内部のレジスタが退避領域として割り当てられています(JARL(Jump and register link)命令で退避して、JMP(Jump register)命令で復帰)。

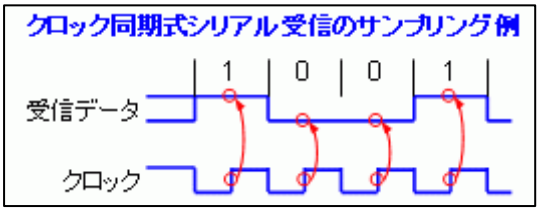


こうした一連の動作は割り込みに似ていますが、次のような違いがあり、サブルーチンはメイン・ルーチンの流れの一部(PSW を共有)であるのに対して、割り込みではメイン・ルーチンと独立したまったく無関係な別の処理が行われます。通常、サブルーチンでは、メイン・ルーチンから何らかの情報(引き数)を受けて処理を行い、その結果(戻り値)をレジスタやフラグ、メモリに戻します。

	サブルーチン	割り込み
分岐条件	CALL 命令(プログラム同期)	要求入力受け付け(非同期)
スタック退避/復帰内容	PC	PC+PSW
引き数/戻り値	あり	なし
復帰条件	RET 命令	RETI 命令

サンプリング(Sampling)

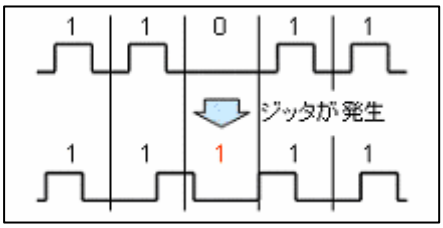
「標本化」と訳され、あるタイミングでの信号を抽出することです。デジタル・サンプリングは、デジタル信号がハイ・レベルかロウ・レベルかを判定して、回路動作の制御のトリガにしたり、通信では受信データが"1"か"0"かを認識したりするために行われます。



アナログ・サンプリングは、A/D コンバータでアナログ信号の電圧をデジタルの数値に変換する場合などに行われます。

ジッタ(Jitter)

パルス信号が、時間軸方向に揺らぐ現象です。原因としては、ノイズやディスク装置の回転ムラ、デバイスの応答性のばらつきや、光ファイバ内の反射による位相ずれなどがあります。
サンプリング点を越えるジッタが発生すると、ビット列を正常に再生できなくなります。



スキューは回路遅延によって固定的に発生しますが、ジッタは環境等によって変動的に発生します。

時定数(Time Constant)

回路定数によって決定される時間的な特性のことで、τ（タウ）で表します。また、時間特性なので、t で表すこともあります。一般には、抵抗 R と容量 C の直列回路(ロウパス・フィルタ)に信号の変化を与えてから、RC 接続部分の電圧の変化量が定常(最終)値の 63.2%に達するまでの過渡応答（遅延）を意味します。

抵抗 R(Ω)と容量 C(F：ファラッド)の直列回路にかかる電圧が 0V から Vo(V)に変化すると、ある時間 t に流れる電流は i(t)(A)、容量に蓄えられている電荷量は q(t)(C:クーロン)で、このときの電圧 Vo は、次のように表せます。

$$Vo = R \cdot i(t) + q(t)/C \quad \text{ここで、} i(t) = dq(t)/dt$$

これを電荷量 q(t)についての微分方程式にすると、次式が得られます。

$$Vo = R(dq(t)/dt) + q(t)/C$$

これを解くと、q(t)は次のようになります。

$$q(t) = CVo(1-\exp(-t/RC))$$

ここで、容量 C の両端の電圧を $V_A(t)$とすると、$q(t) = CV_A(t)$なので、次のようになります。

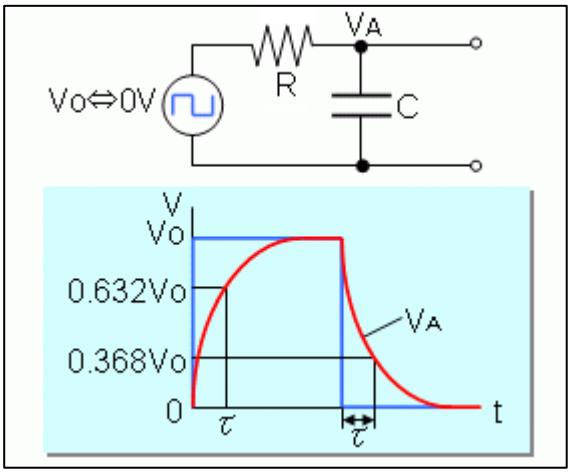
$$CV_A(t) = CVo(1-\exp(-t/RC))$$

$$V_A(t) = Vo(1-\exp(-t/RC))$$

ここで、「$\tau = RC$」と定義すると、$t = \tau$ のときには、V_A が次の値になります。

$$V_A(\tau) = Vo(1-e^{-1}) \fallingdotseq 0.632Vo$$

したがって時定数 τ は、V_A が Vo の 63.2%に達するまでの時間で、抵抗 R や容量 C が大きいと、それらに比例して長くなります。ちなみに $t = 2.2\tau$ のとき、$V_A \fallingdotseq 0.9Vo$ となります。



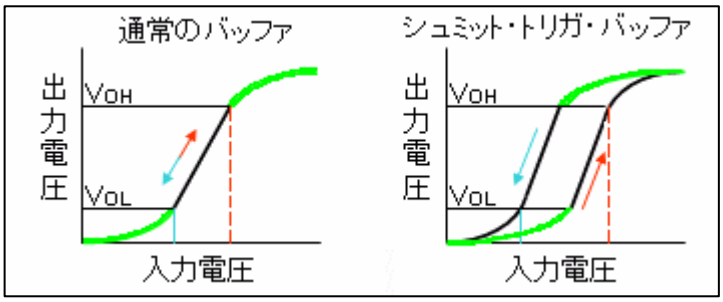
リセット用の IC などは、この時定数を利用した遅延回路です。

ジャンクション(接合部)温度(Junction Temperature)

動作状態で許容される、接合部の温度のことです。
絶対最大定格の略号表記は T_j です。

シュミット・トリガ(Schmitt Trigger)

ヒステリシス特性を持つバッファのタイプです。
ヒステリシスとは、入力の立ち上がりと立ち下がりに対するスレッシュホールド(スレッショルド)電圧が異なる特性のことです。通常のバッファより立ち上がり／立ち下がりが急峻で、しかも両者のスレッシュホールドの幅が広くなります。したがって、このタイプのバッファは、入力電圧が変動しても出力に影響が少ないため、ノイズに強くなっています。
次図では、入力電圧の赤線(破線)が立ち上がりに対するスレッシュホールド、青線(実線)は立ち下がりに対するスレッシュホールドで、波形の緑線(太線)が入力ノイズの許容電圧範囲です。



進数(n 進数:n-adic Number)

数の表記で、どの値で「けた上げ」が生じるかの定義で、これによって使用する数字が決まります。

通常の生活で、わたしたちは 10 進数(decimal code:通称「デシマル」)を使用しています。これは、0 から 9 の数字を使い、9 の次はけたが上がって、10(じゅう)になります。生活上の例外には、時計(12 進、60 進)があります。

デジタルの情報処理では、各けたが 0 か 1 の 2 進数(binary code:通称「バイナリ」)を使用します。これは、デジタル信号がロウ・レベル(L=0)とハイ・レベル(H=1)の電圧で定義されるからです。
2 進では、1 の次にけたが上がって、10(イチ・ゼロ)になります。10 進の「じゅう」と区別する場合、"10b"または"10B"と表記します。

またマイコンなどの場合、2 進数ではけた数が多くなるので、表記する場合は 16 進数(hexadecimal code:通称「ヘキサ」)が一般的です。2 進数の 4 けたが、16 進数の 1 けたに相当します。
16 進では、9 以降にアルファベットの A から F をあてて、F(10 進の 15)の次にけたが上がって、10(イチ・ゼロ)になります。10 進の「じゅう」と区別する場合、"10h"または"10H"、"0x10"と表記します。

10 進	2 進(バイナリ:b)	16 進(ヘキサ:h)
00	0000 0000	00
01	0000 0001	01
02	0000 0010	02
03	0000 0011	03
04	0000 0100	04
05	0000 0101	05
06	0000 0110	06
07	0000 0111	07
08	0000 1000	08
09	0000 1001	09
10	0000 1010	0A
11	0000 1011	0B
12	0000 1100	0C
13	0000 1101	0D
14	0000 1110	0E
15	0000 1111	0F
16	0001 0000	10

10 進では、1 の位、10 の位、100 の位....というように定義しますが、それぞれ 10^0、10^1、10^2....というけたに相当しています。同様に、2 進なら 2^0、2^1、2^2....、16 進なら 16^0、16^1、16^2....というけた取りをします。

たとえば、10 進数の"**123**"($1\times10^2+2\times10^1+3\times10^0$)は、それぞれ次のような表記になります。

16 進: $7\times16^1+11\times16^0$ → **7B**

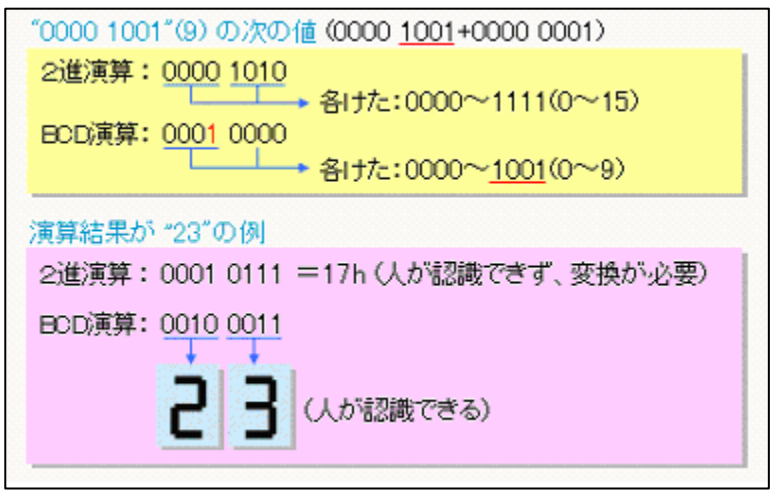
2 進 : $1\times2^6+1\times2^5+1\times2^4+1\times2^3+0\times2^2+1\times2^1+1\times2^0$ → **1111011**

デジタルの分野では、128 や 256 といった数を目にすることがありますが、これらは 2^7、2^8 に相当しています。

つまり、8 ビット(けた)で 1 度に処理できる数は 256 通り(0～255)です。

なお、デジタル時計や電卓のような、10 進数を扱うための演算を「けたごと」に 2 進で処理する場合があり、これを 2 進化 10 進(BCD:Binary Coded Decimal)と呼びます。

たとえば 2 進演算の結果が"00010111b"(17h)の場合、10 進表記では"23"です。ですから、10 進表示をするには変換が必要です。これに対して BCD 演算では、けた上げを管理して、各けたの演算を 0 から 9 (1001b)の数で行い、"2"と"3"を結果として出力します。したがって BCD 演算では、2 進数から 10 進数への変換が不要です。



数詞(Telwoord)

通常の単位系では、k(キロ:小文字)や M(メガまたはミリオン)といった数詞を使用します(例:1km)。これらは 10 進で、それぞれ 10^3(千)、10^6(百万)を意味します。
これに対し、情報処理(2 進数)で扱う数詞の K(キロ:大文字)や M(メガ)は「2 のべき乗」なので、1K=1024(2^{10})、1M=1048576(2^{20})となります(例 : 1K バイト)。

数値(Number)

デジタル回路で使用する数値は、通常 n ビットの 2 進数となります。たとえば、次のような値です。

8 ビット :00000000～11111111(16 進表記で 0～FF)

16 ビット:0000000000000000～1111111111111111(16 進表記で 0～FFFF)

これらを 2 進数や、4 ビット分をまとめた 16 進数で表現する場合には問題ないのですが、10 進数で表す場合などに注意が必要です。問題となるのは、数値の範囲で、具体的には負の数の表現方法です。2 進数そのものには、正の数や負の数の概念はありません。しかし、自然界に存在するものは、負の数値で表現する必要があるものもあります。
一般的に用いられる負の数の表現としては、(2 進数の)2 の補数表現があります。これは、0 から表現したい値を引くことで計算できます。たとえば、－100、－10、－1 の場合には、次のようになります。

－１００の場合	－１０の場合	－１の場合
0000 0000	0000 0000	0000 0000
－0110 0100	－0000 1010	－0000 0001
1001 1100	1111 0110	1111 1111

2 の補数表現では、加減算はできても乗除算はできません。このときには、符号と値の絶対値として、符号同士と絶対値部分同士で演算する必要があります。このとき、数値を「符号ビット＋絶対値」で表現することになります。負の数をどのように表現するかは、個々のケースで異なります。特に、アナログ信号とデジタル信号の間で変換する場合は、使用する変換器で負の数の定義が異なることがあります。

スキャン(Scan)

一連のデータに対して、順次状態の入出力をすることです。
たとえば、キー入力状態(入力／非入力)を順次取り込むことをキー・スキャンと呼びます。また、CRT や CCD で、画像を横向きの直線データの集合に分割し、これを走査して順次、表示出力/読み取り入力することもスキャンといいます。A/D コンバータで複数の入力を順次取り込むこともスキャン動作です。

スタック(Stack)

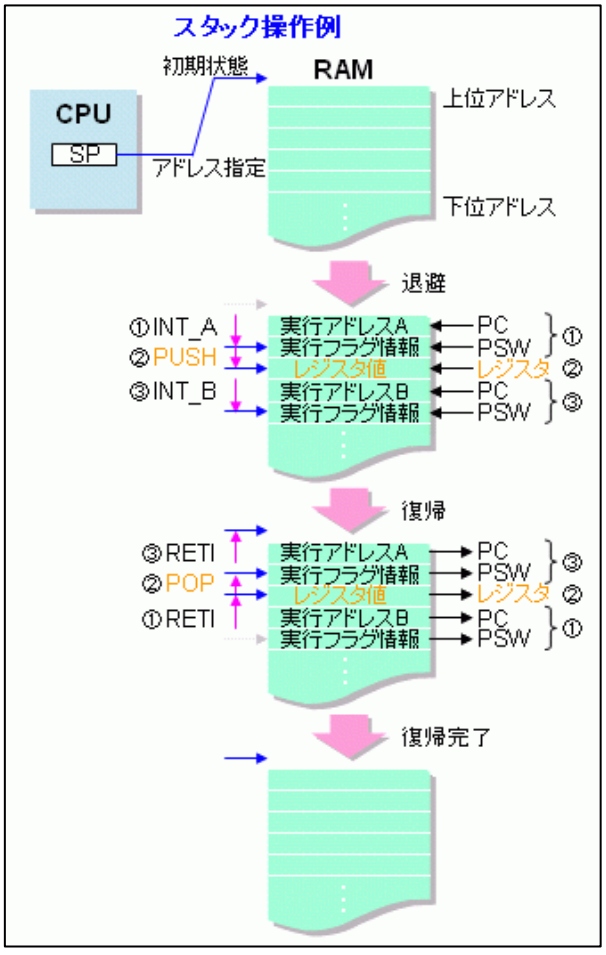
レジスタの内容を上書きで消失させないように、一時的に退避させる領域で、メモリ(RAM)上に割り当てます。
CPU のサブルーチン・コールのように、プログラム処理が別アドレスへ分岐する際、元のアドレスへ戻ることを前提にする場合があり、その戻り先のアドレス(プログラム・カウンタ(PC)の値)をスタックに退避させます。この分岐をするのは、割り込みとサブルーチン・コール(CALL 命令)のときで、割り込みでは PSW(Program Status Word:プログラム実行情報のフラグ・レジスタ)の内容も退避させます。各処理の最後に次のような復帰命令を実行すると、スタックに退避させたアドレスが PC に復帰して、自動的に元の処理へ戻ります。

割り込み:RETI (Return from Interrupt)命令(PSW も復帰)

サブルーチン・コール(CALL 命令):RET (Return)命令

なお、ジャンプ命令やブランチ命令の分岐は、戻ることを前提としないので、このようなスタック操作をしません。
また、演算などでレジスタを使用する際、現状のレジスタ値をあとで使用する場合に、PUSH 命令でレジスタ値をスタックに退避させます。PUSH 命令に対しては、スタックに退避させたレジスタ値が必要となったときに POP 命令を実行して、元のレジスタに復帰させます。

CPU にはスタック・ポインタ(SP)があり、スタック領域のどこをアクセスするかを管理しています。スタック・ポインタの値をデクリメント(－1)したアドレスにデータがストアされるため、一般にスタック・ポインタの初期値は、RAM の「最高アドレス＋1」に設定されます。
スタックには、汎用レジスタより多くのデータを格納できますが、読み出しには条件があります。この条件は LIFO(Last-in First-out)と呼ばれる制御で、最後に書き込んだデータが最初に読み出されます。つまり、最初に書き込んだデータの読み出しは最後になります。

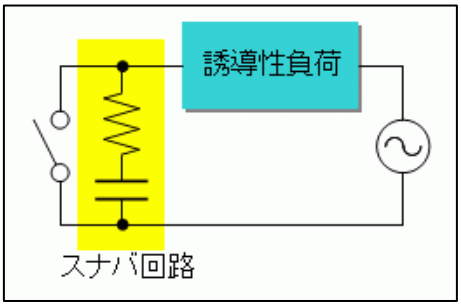


スナバ回路(Snubber Circuit)

スパイク・ノイズを吸収する回路で、抵抗とコンデンサを直列に接続したものです。
スパイク・ノイズは、回路のインダクタンス分によって、電流変化時に起こる逆起電力が原因となって発生します。
電流によって、右ねじの法則(アンペールの法則)に従った磁界が発生し、電流変化でその磁束の変化が起こると、次のような逆起電力による電圧が発生します。

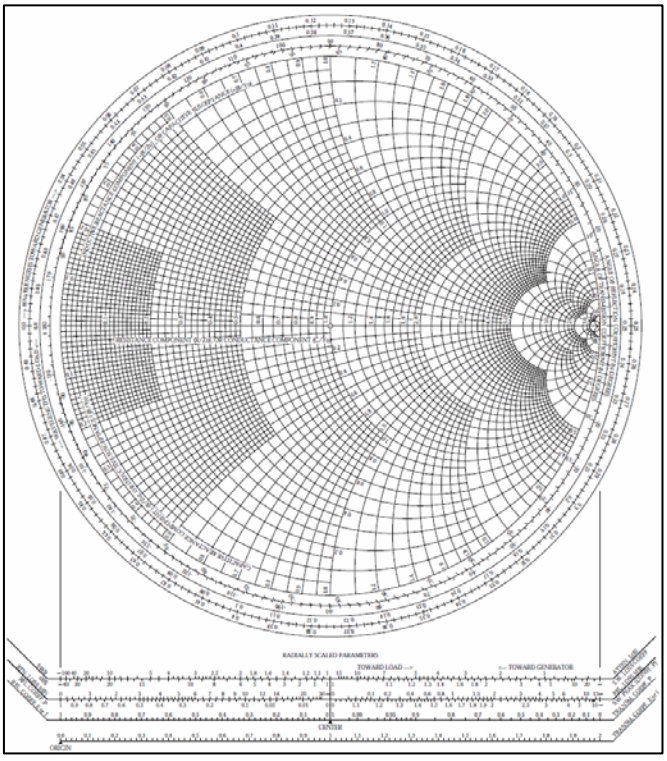
$dV = -d\Phi/dt$ (Φ:磁束)

電球が、消灯時に切れやすいのは、このためです。誘導性負荷の電源の ON/OFF では、スパイク・ノイズが発生しやすいので、スイッチやリレーにスナバ回路を接続して吸収します。

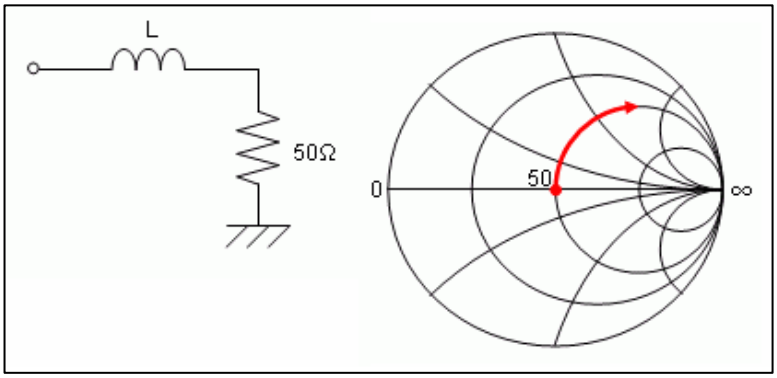


スミス・チャート(Smith Chart)

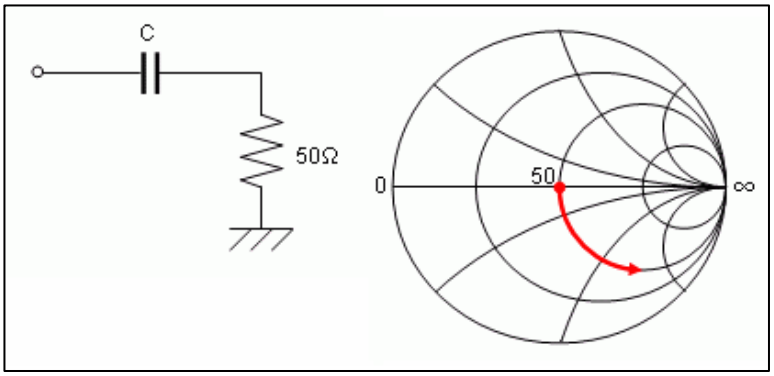
RF(Radio Frequency)やマイクロ波のインピーダンス整合の設計に用いられる、円形のインピーダンス・チャートです。横軸はインピーダンス Z の実部(抵抗成分)で、縦軸が虚部(リアクタンス成分)を表します。左端は Z=0(全透過)、右端は Z=∞(全反射)に相当します。また、周囲に表示している角度は、電圧反射係数の**位相 θ (ANG.: angle)**です。中心 Z_0 は、負荷と伝送路が整合した状態に相当し、通常は 50Ω ですが、下図のインピーダンス・チャートでは、入力インピーダンスを基準に正規化して 1.0 で表現しています。
中心からの**距離 ρ (MAG.: magnitude)**が、反射量の大きさです。



たとえば、50Ω の負荷に直列にコイル L を接続すると、L のインダクタンスが大きくなるにつれて、軌跡が∞に近づきます。

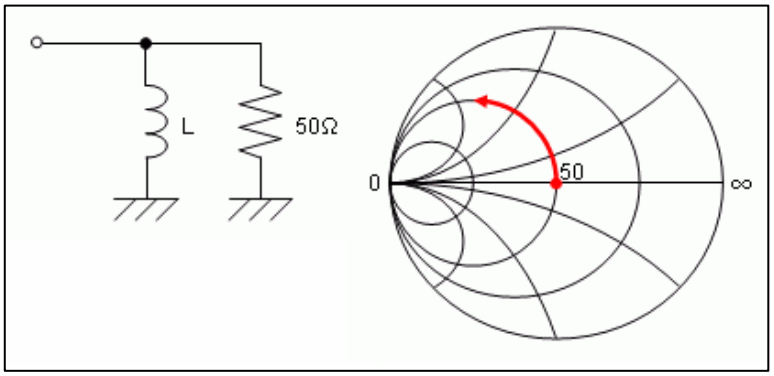


また、50Ω の負荷に直列にコンデンサ C を接続すると、C の容量が小さくなるにつれて、軌跡が∞に近づきます。

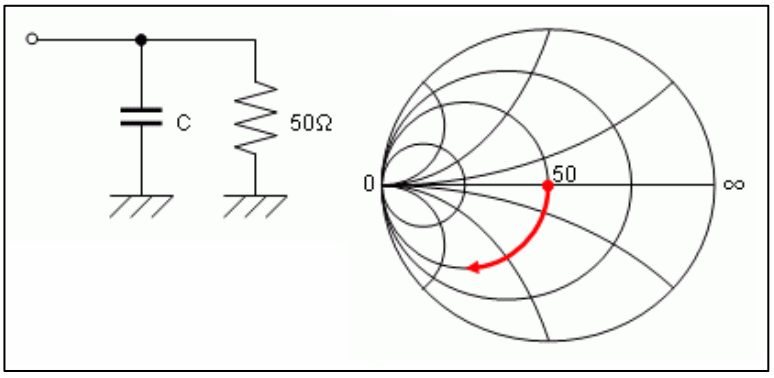


なお、スミス・チャートには、インピーダンス・チャートのほかに、横軸のスケールはそのままで、左右反転（180°逆位相）してアドミタンス Y（インピーダンスの逆数）を表現する**アドミタンス・チャート**や、両方を重ねた**イミタンス・チャート**があります。並列回路では、アドミタンス・チャートが見やすくなります。

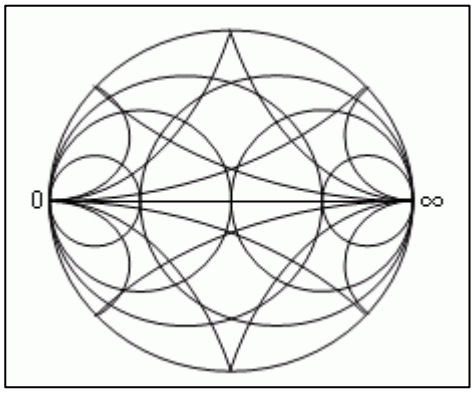
たとえば、50Ω の負荷に並列にコイル L を接続すると、L のインダクタンスが大きくなるにつれて、アドミタンス・チャートの軌跡が 0 に近づきます。



また、50Ω の負荷に並列にコンデンサ C を接続すると、C の容量が大きくなるにつれて、軌跡が∞に近づきます。



イミタンス・チャートは、直列と並列を両方表現する場合に利用されます。



なお、リアクタンスは$j(\omega L-1/\omega C)$なので、$\omega=2\pi f$よりインピーダンスは周波数 f によって変化します。

スミス・チャートは、S パラメータの周波数特性を表すのに用いられ、周波数の変化に対するインピーダンスの軌跡がプロットされます。

スリー・ステート(Three State)

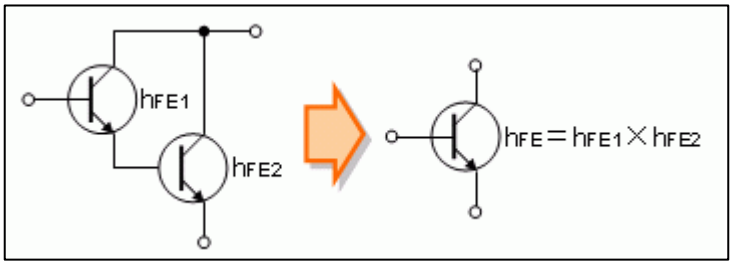
ハイ・レベル、ロウ・レベル、ハイ・インピーダンスの3つの状態を可能とするデジタル出力です。

ハイ・インピーダンスには、出力を遮断するだけの場合と、入力になる場合があります。

トライステートは同義語ですが、ナショナルセミコンダクター社の商標なので、当社では使用していません。

ダーリントン接続(Darlington Circuit)

2 つのバイポーラ・トランジスタのコレクタ同士を接続し、さらに一方のエミッタを他方のベースに接続した回路です。これによって、全体を 1 つのトランジスタのように動作させることができ、電流増幅率 h_{FE}は、2 つのトランジスタの h_{FE}の積となります。さらに多段接続をすれば、それにともなって電流増幅率を大きくすることができます。



ダイナミック動作とスタティック(Dynamic Operation and Static)

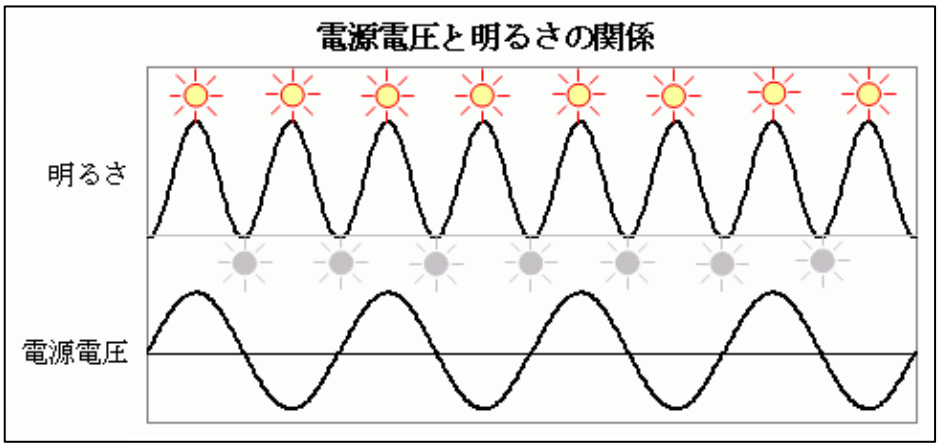
一般に存在するものは状態が安定して、ほとんどはいつも同じように確認できます(スタティック)。ところが、人間の目は高速に変化するものに追従できません。一般に 70Hz 以上の変化では、違いを認識できないようです。



[ダイナミック動作]

この特性を積極的に利用したのが、蛍光灯などの照明やテレビなどで用いられるダイナミックな動作（ダイナミック表示）です。

蛍光灯は商用電源の周波数が 50Hz の地区ではその倍の 100Hz で点滅を繰り返しています。これは、次のグラフで模式的に示すように、＋のピークと－のピークで明るくなり、0V 付近では消えるからです。人間の目が、一様に光っているように感じるのは、速い変化を認識できないからです。



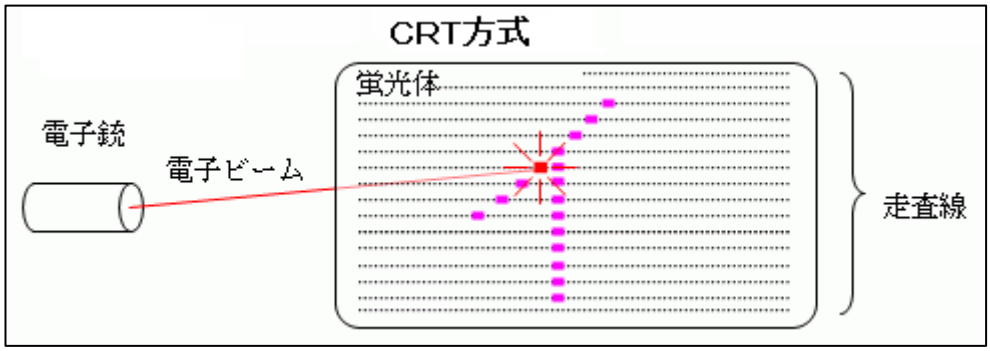
【コーヒー・ブレーク】

人の感覚には個人差があります。また、年齢により変化します。そのため、蛍光灯のちらつきが気になる人もいます。そこで、蛍光体の残光時間を長くして、ちらつきを抑えたものもあります。中には、電気を切ってもしばらくは光っているホタルックのような製品もあります。

最近は商用電源周波数ではなく、20～50kHz でのスイッチングを行って、高い周波数で点灯させるインバータ照明も多く使われています。可聴周波数以上の周波数で点灯させるので、ちらつきはなく、従来の蛍光灯のような点灯の遅さもありません。

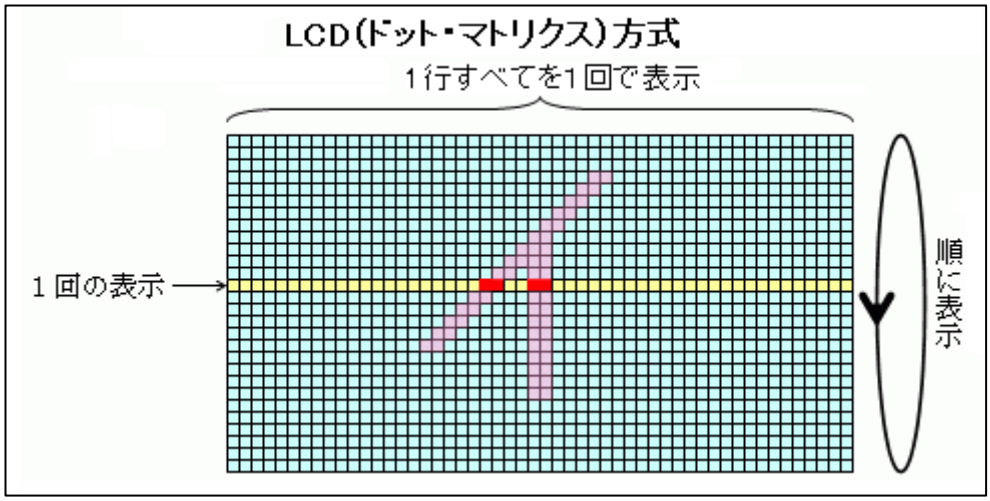
[表示器でのダイナミック動作]

CRT 方式の TV では、1 本の電子ビームが画面前面の蛍光体を横方向（走査線）に移動しながら、画面を表示させています。通常の TV 放送（SD：Standard definition）では、1 画面（フレーム）分 525 本の走査線の半分を約 1/60 秒で表示しています（これをフィールドと呼び、奇数フィールドと偶数フィールドで、1 つの完全な画面を構成します。

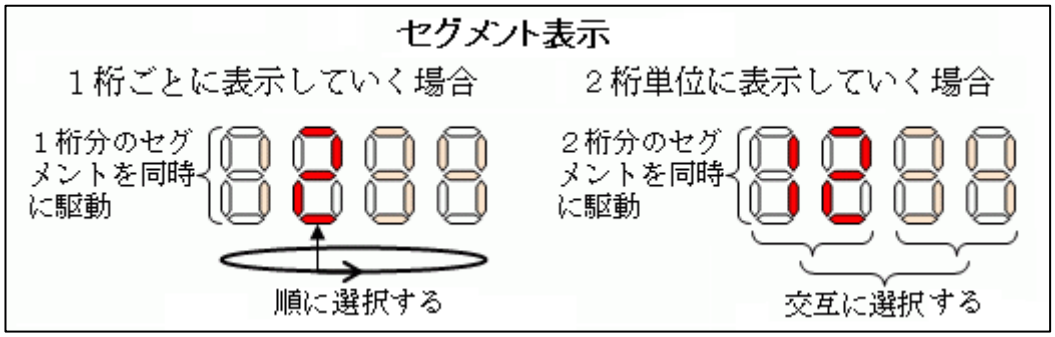


瞬間的に電子ビームが当たっているのは、1 点（上図で光っている部分）だけですが、蛍光体がしばらくは光り続ける（上図で光っていない点）ので、ちらつきが感じられません。（カメラやビデオで撮影すると、暗くなっているところがわかります。）

LCD の TV では、CRT 方式のような点（ドット）でなく、横方向の複数のドットが同時に表示されます。これを縦方向に順番に切り替え（ダイナミック動作）て、画面を表示させます。横 640 ドットで縦が 480 ラインの画面では、これをスタティックに表示させるには約 32 万本の信号線が必要です。これをライン単位でダイナミックに表示させれば、1120 本（640 本＋480 本）の信号線で済みます。



TV のようなドット表示でなく、たとえば電卓のように、決まりきった数字を表示するだけのセグメント形式の表示では、セグメント信号と桁を示す信号の組み合わせで制御します。次図左の例は、桁ごとに表示するものです。この例では、セグメントの駆動に 7 本、桁の選択に 4 本で、合計 11 本の信号線を使用します。右の例は、2 桁ずつ表示するもので、セグメントの駆動に 14 本の信号線を使用します。

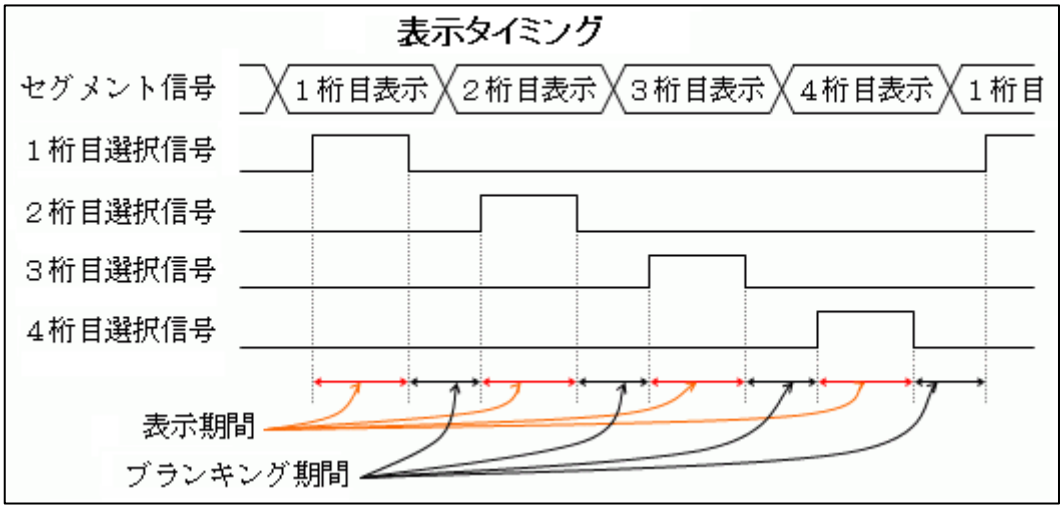


[ダイナミック表示の制御]

上記のセグメント表示の例について、その制御をタイミング図で示します。

まず、1 桁目のセグメント表示データをセグメント信号に出力し、その後、1 桁目選択信号を出力します。これで一桁が表示されます。次に、2 桁を表示するため、1 桁目選択信号を切り、セグメント信号のデータを 2 桁目のセグメント表示データに変更します。その後、2 桁目選択信号を出力して、2 桁目を表示します。

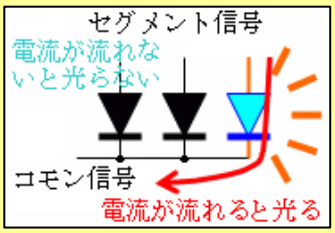
このように、桁の切り替えの際に選択信号が出力されない期間が存在するのは、表示のにじみをなくすためで、この期間をブランキング期間と呼びます。ブランキング期間は、LED や蛍光表示管では必須ですが、LCD 表示では表示速度が遅く、追従できないので必要ありません。



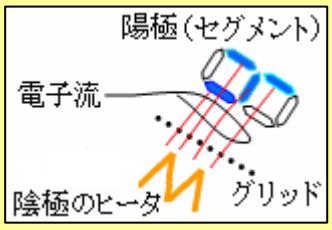
このように、時間ごとに表示する桁を順次切り替えて表示することを時分割表示と呼びます。分割の数を増やすと、信号線の数を少なくできますが、表示している期間が短くなります。このため、表示しているところと表示していないところの差（コントラスト）が低くなります。これを避けるためには、大きな電流を流して、できるだけ表示を明るくする必要があります。

【コーヒー・ブレーク】

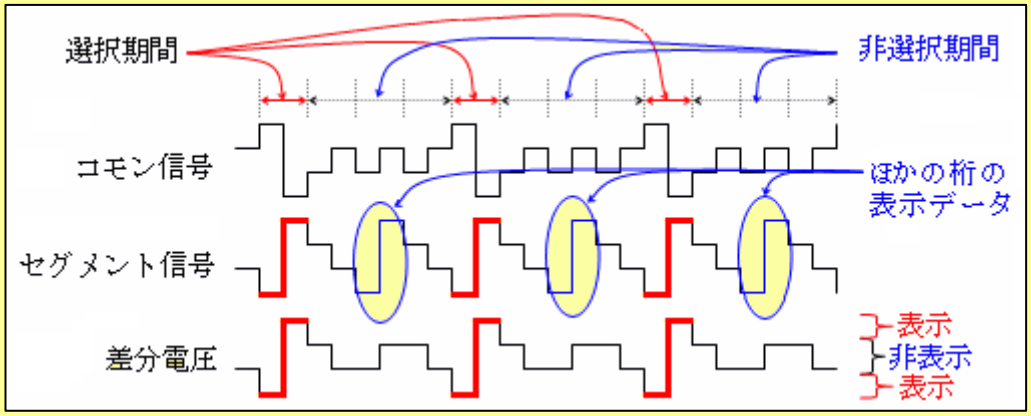
前記の表示タイミングは考え方としてはいいのですが、実際の信号波形しては注意が必要です。セグメント信号を正論理とすると、LEDの場合にはカソード・コモン型を使用して、桁選択信号（コモン信号）は負論理にする必要があります。つまり、セグメント信号からコモン信号に電流が流れ点灯します。



蛍光表示管では、セグメント（光る部分）は陽極になります。桁信号になるのがグリッド電極で、電流はカソード（陰極）とセグメントの間を流れます。電流が流れるには、陽極とグリッド電極が、ともに陰極に対してプラスになる必要があります。両方がプラスになると、陰極のヒータから電子が出て、セグメント電極に向かい、そこで発光します。

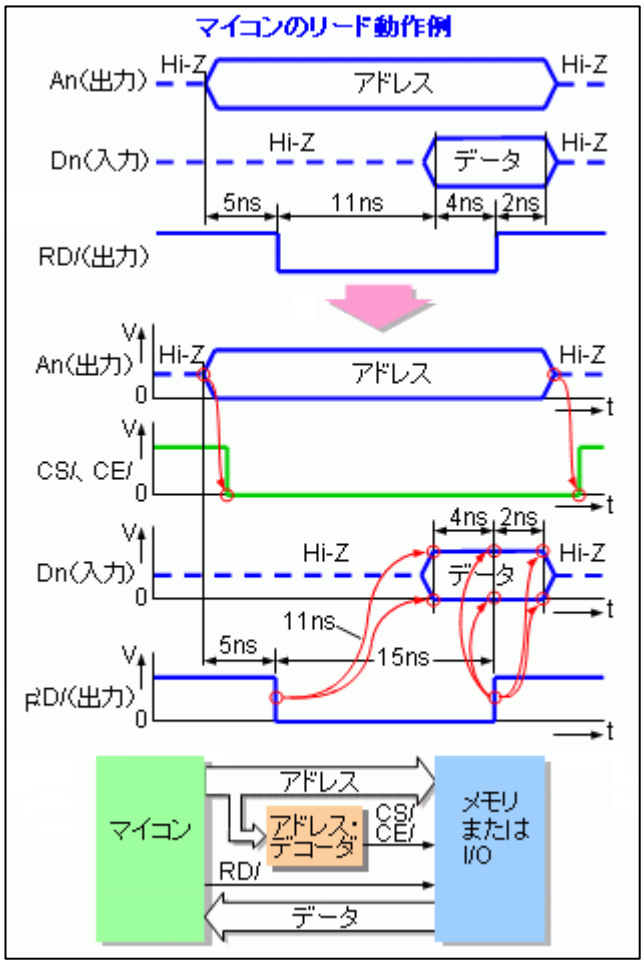


LCD（液晶）の場合には、電極の間に電圧がかかれば、液晶の配列がねじれて、光の透過率が変化します。LED は逆方向の電流が流れませんが、かける電圧の極性が変わっても同様の変化があるので、非選択状態で注意が必要です。そのため、フル振幅の電圧以外に、中間的な電圧を加えられるようにしておきます。表示させる場合はフル振幅の電圧がかかり、表示しない場合は中間的な電圧がかかるようにします。このように、LCDをドライブするには、2 値の電圧では不足するため、通常のポートでドライブすることは困難です。また、電圧をかける方向がつねに同じではなく、逆転させる必要もあります。駆動波形の例を次に示します。差分電圧で、赤く（太く）示された部分が表示される部分です。



タイミング・チャート(Timing Chart)

電気信号の電圧や電流が、時間とともにどのように変化するかを表すグラフで、通常は電圧のタイミング・チャートが用いられます。表現については、オシロスコープの波形をイメージすれば、分かりやすいです。たとえば次図のような、マイコンがデータをリードするというデジタルのタイミング・チャートの場合、次のような動作を表します。



(1) マイコンがアドレスを出力すると、このアドレス信号と、これをデコードして生成した選択信号(CS/やCE/)が、メモリかI/Oデバイスに入力される。

(2) マイコンがリード・ストローブ信号RD/をアクティブ(ロウ・レベル)にする(SRAMではOE/入力)。

(3) アクセス対象のメモリかI/Oデバイスが、データを出力して11ns後に確定する(メモリではアクセス時間)。

(4) マイコンがRD/をアクティブにしてから15ns後に、インアクティブに戻す。

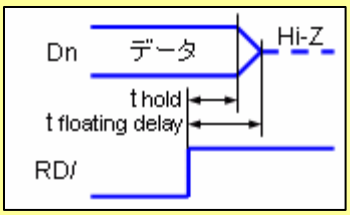
(5) マイコンは、データがハイ・レベル(V_{IH}の範囲)かロウ・レベル(V_{IL}の範囲)かをサンプリングして取り込む。

(6) アクセス対象のデバイスが、RD/のインアクティブから2ns後にデータ出力を遮断する。

以上の動作により、マイコンのリード動作に対して、データのセットアップ時間が4ns、ホールド時間が2nsになります。マイコンの各規格が、これらの時間以内(MIN.規定)なら、正常にデータをリードできることになります。

【ティー・ブレーク】

データの保持については、ホールド時間とは別に、バス・フロート遅延時間があります。これは、リード・ストローブ信号がインアクティブになってから、データ線がフローティング(Hi-Z)状態になるまでの時間で、高速システムでは次のバス・サイクルとの衝突の原因となることがあります。高速のマイコンでは、このような状態を避けるために、バス・サイクルの最後にアイドル・サイクルを挿入することがあります。

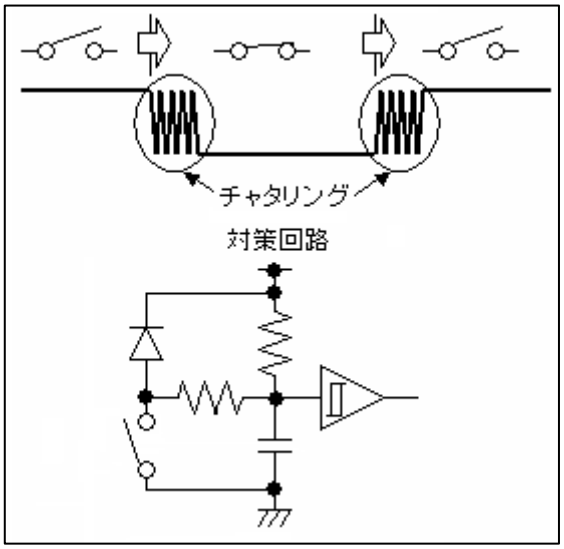


チェック・サム(Check Sum)

デジタル通信やコンピュータでの誤り(エラー)検出手法のひとつです。nビットごとのデータを加算し、その合計を誤り検出用データとして追加します。デジタル通信では、その誤り検出用データを含めて全データを送信し、受信側でnビットごとのデータを加算して、その結果と誤り検出用データを照合することにより、誤りを検出します。
また、合計を誤り検出用データとするのではなく、その合計の 2 の補数を誤り検出用データとすることによって、受信側でこの誤り検出用データまで加算して、その結果の全ビットが"0"かどうかで誤りを検出します。
たとえば、"0xF001F102F203F304F405F506F607F708"の 16 バイトのデータを 8 ビットごとに加算する(オーバフローは無視する)と、"0xC0"になります。そして、その 2 の補数を取ると"0x40"になるので、これを追加した"0xF001F102F203F304F405F506F607F70840"の 17 バイトのデータにして送信します。受信側で 8 ビットごとに加算し、その結果を確認して、"0x00"なら誤りがなかったことになります。
このチェック・サムは、ROM 内蔵マイコンの ROM コード発注時のフォーマットの中で使用しています。

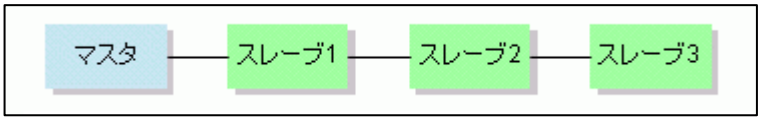
チャタリング(Chattering)

手動スイッチによる入力で、接点の振動によって、短時間にON/OFFを繰り返す現象です。
リセット入力やカウンタ入力などでは、誤動作の原因となることがあります。
通常は、RC回路とシュミット・トリガ・バッファの組み合わせなどで除去します。次の対策回路のダイオードは、電源OFF時にバッファ入力の絶対最大定格を越えないよう、コンデンサのチャージを放電するためのものです。
ダイオードとコンデンサの間の抵抗は、大容量で低ESRのコンデンサを使用する場合に、ダイオードとコンデンサを保護する電流制限用に必要となります。スイッチON時のロウ・レベルを確保でき、バッファを保護できるような低抵抗で制限してください。



デイジ・チェーン(Daisy Chain)

あるデバイスがマスタとなり、これに同種のスレーブ・デバイスを直列(出力を次段の入力)に接続する接続形態です。ひな菊(デイジ)をつなぐ、子供の遊びに由来しています。
デイジ・チェーンでは、各デバイスの割り込みの優先順位を簡易的に決定できます。
ただし、信号が直列に伝わるので遅延が大きいため、低速のシステムに使用されます。



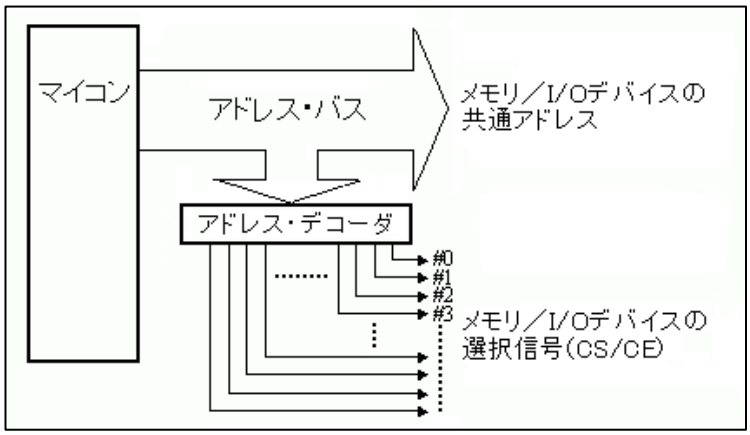
デジタルTV放送(Digital Television Broadcast)

50 年間続いたアナログ信号によるテレビ放送に対し、デジタル信号で行うテレビ放送のことです。
通信衛星を使った BS/CS デジタル放送は、それぞれ 2000 年 12 月/2002 年 7 月にスタートし、地上デジタル放送は 2003 年 12 月に関東、中京、近畿の一部でスタートしました。
日本では、2011 年 7 月にアナログ放送を終了させて、完全デジタル化に移行しようとしています。
デジタル・テレビジョン放送の規格概要の一覧を次表に示します。

地域名	日本				米国など	欧州など
放送形式	地上デジタル TV 放送 (ISDB-T)	BS デジタル放送 (ISDB-S)	110°CSデジタル放送 (ISDB-S)	124/8°CSデジタル放送 (DVB-S)	地上デジタル TV 放送 (ATSC)	地上デジタル TV 放送 (DVB-T)
映像符号化方式	MPEG2 Video					
音声符号化方式	MPEG2 Audio(AAC)			MPEG2 Audio(BC)	AC-3	MPEG2 Audio(BC)
多重化方式	MPEG2 Systems					
限定受信方式	MULTI2/B-CAS			MULT2/独自	DES	DVB-CSA
変調方式	OFDM 64QAM,16QAM,QPSK	TC8PSK,QPSK,BPSK		QPSK	8VSB	OFDM 64QAM,16QAM, QPSK

デコーダ(Decoder)

デコーダはエンコード(コード化)で圧縮した情報を元に戻す機能です。
ソフト的には、エンコードした JBIG(FAX)や MPEG のような画像データを再現する技術があります。
ハード的には、マイコンが出力するアドレスに従って、アクセスするメモリや I/O デバイスを選択するアドレス・デコーダが、例としてあげられます。



マイコン・システムでは、使用するメモリや I/O デバイスが必要とするアドレスをマッピング(割り振り)しますが、各アドレス空間を定義するのがアドレス・デコーダです。
たとえば 64K のアドレス空間を持つマイコンなら、65536 本の選択信号が必要となり、現実的ではありません。
そこで 16 本(A15-A0)の"0,1"の組み合わせにエンコードして、これを選択信号に戻すのがデコーダです。
たとえば前図で、マイコンに接続するデバイス#0 のアドレス空間が 0000H-03FFH(1K)で、デバイス#1 のアドレス空間が 0400H-07FFH(1K)とします。デコーダは各領域のアドレスに対して、それぞれ#0、#1 をアクティブにします。たとえば、アドレス 0200H なら#0、アドレス 0600H なら#1 がアクティンブになります。

ここで、各デバイスが持っている物理アドレスは 0000H-03FFH です。
これを 2 進数で表すと 0000 00xx xxxx xxxx (x=0 or 1)ですから、必要なアドレス線は A9-A0 となり、A10 をデコードすれば#0、#1 を選択でき、A10=1 で#1 をアクティブにすれば、論理的には 0400H-07FFH のアドレス空間に割り当てることになります。つまり、A10 とその反転信号で、#0 と#1 のアクセスを選択できることになります。
このように、接続デバイスの物理アドレスを共通アドレスとして、それより上位のアドレス線をデコードして、各デバイスを選択します。このため、回路の規模と構成は、接続デバイスとその数によって異なります。
メモリと I/O を区別するマイコンでは、識別信号でメモリと I/O を分けます。
また、リード信号かライト信号で、選択しているデバイスのアクセスを決定します。

デシベル(Decibel:dB)

電気工学や音響工学などで、入出力の関係を表すのに用いられる無次元単位です。具体的には、入出力比の常用対数を 10 倍して、利得(ゲイン)を表します。

$G = 10\log_{10}$(出力／入力)

たとえば、電力利得が 1000 なら、次のようになります。

$G_P = 10\log_{10}1000 = 10\log_{10}10^3 = 30$(dB)

ただし、電力は電圧の 2 乗に比例($10\log_{10}A^2$)するので、電圧利得の場合には 20 倍で表します。

$G_V = 20\log_{10}$(出力電圧／入力電圧)

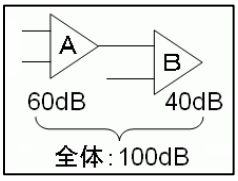
常用対数で表すため、増幅の場合は正となり、減衰の場合は負となります。
なお音の場合には、音圧レベルの比を使用します。たとえば防音効果を示す場合、音源(入力)と集音側(出力)の音圧レベルが 1000:1 とするなら、次のようになります。

$G = 10\log_{10}(1/1000) = 10\log_{10}(1/10^3) = 10\log_{10}10^{-3} = -30$(dB)

入出力の電圧の比をデシベルで表示すれば、表示けた数を次のように圧縮できます。

電圧の入出力比[倍]	デシベル[dB]
1000	60
100	40
0.1	−20
0.01	−40

さらに、デシベルを用いると、複数段の増幅率を次図のように加算で簡単に計算できます。



【コーヒー・ブレーク】
人間の感覚は、変化に対しては敏感で、変化しないものに対しては鈍くなります。
また、自然界の現象も相対的な変化が主です。このため、リニアな目盛りよりは、対数で表した方が比較的この感覚に合います。そのため、デシベルでの表記がいろいろなところで使用されています。
なお、デシベル自体はあくまでも数値の比ですが、絶対的な値を表したいときには、dB のあとに数詞をつけます。たとえば dBm で電力(mW 単位)、dBV や dBμ(μV 単位)で電圧を表します。
例 10dBm＝10mW、20dBV＝10V、20dBμ＝10μV

デッドロック(Dead-lock)

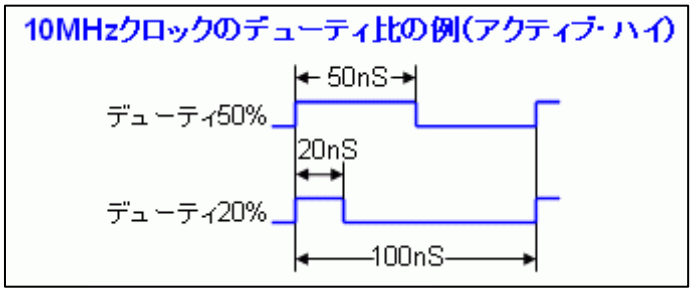
プロセッサが暴走したり、あるリソースに対する複数のアクセスが競合したりして、外部からの制御ができなくなる状態で、ハングアップ(Hung-up)とも呼ばれます。
複数のタスクで構成されたソフトウエアでは、あるタスクが必要なリソースが確保できないときに、開放待ち状態になります。この場合、リソースが開放されれば、そのタスクを実行できる状態となるので、これだけではデッド・ロックが発生しません。デッド・ロックになるのは、次のケースです。
タスク 1 とタスク 2 がある処理をするときに、ともにリソース A とリソース B を確保するとします。このとき、リソースを確保する順番が、タスク 1 は「リソース A→リソース B」、タスク 2 は「リソース B→リソース A」で、2 つのタスクが同時に処理を行えば、デッド・ロックが発生します。
この場合、タスク 1 はリソース A を確保した状態でリソース B を確保しようとしますが、タスク 2 がすでにリソース B を確保した状態でリソース A を確保しようとしている状態が考えられます。このとき、タスク 1 とタスク 2 は、1 つのリソースを確保した状態で、もうひとつのリソースが開放されるのを待ちます。ところが、待ち状態になった 2 つのタスクが確保しているリソースを開放しないかぎり、次のリソースを確保することはできませんので、2 つのタスクは永久にリソース待ち状態から抜け出すことができません。

デプレッション(Depletion)

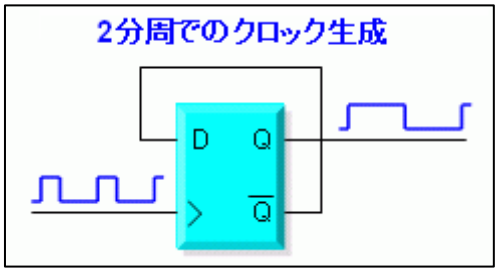
MOS FET のタイプのひとつで、ゲート電圧が 0V でドレーン電流が流れる特性を持ちます。
今日では、エンハンスメント形が一般的です。ゼロ・バイアス時の電流で、製品をランク分けします。

デューティ比(Duty Cycle)

クロックなど一定周期の信号について、周期に対するアクティブ・レベル幅の比(t_{ACT}/t_{CY})のことです。パーセンテージで表し、デューティ 50%は、アクティブ・レベルとインアクティブ・レベルの幅が等しいことを意味します。



クロックなどで回路を駆動する場合、デューティ比が 0%や 100%に近いと、一般に正常動作ができないので、ハイ・レベルとロウ・レベルの最小幅を規定しています。なお、デューティ 50%のクロック信号は、必要な周波数の倍の周波数のクロックをフリップ・フロップで 2 分周すれば得られます。

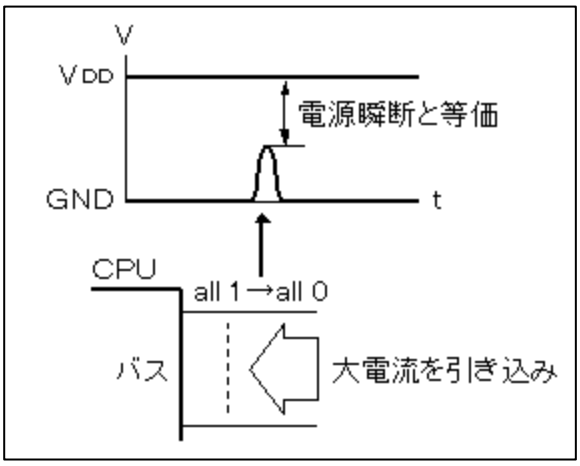


電気角と機械角(Electric Angle and Mechanical Angle)

ステッピング・モータ用ドライバが、ステッピング・モータのコイルに流す交流電流の角度のことを電気角と言い、それによって、実際にモータが回転する角度を機械角と言います。
たとえば、8 極 2 相で、36 枚の小歯を持つステッピング・モータの場合、ある極に交流電流を一周期分(電気角を 360 度)流したとき、実際にモータは 360 度÷36 枚＝10 度(機械角)回転します。

電源／グランド・ノイズ(Noise on Power/Ground)

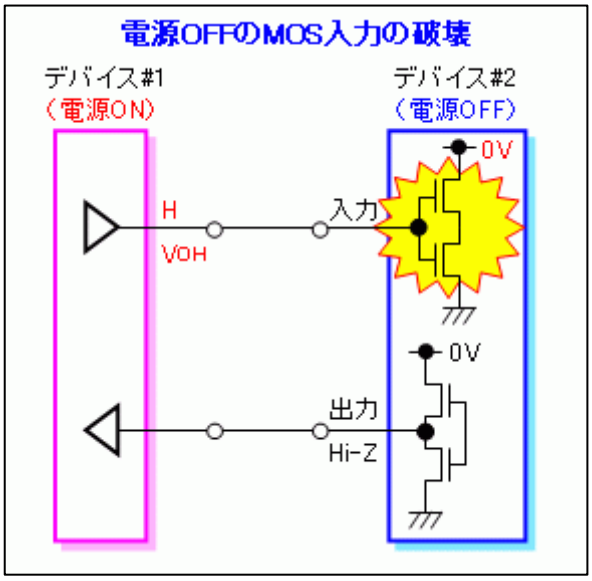
基板の電源／グランドの配線パターン上の電流分布は均一ではありません。
電源供給部から遠い個所や、配線パターンが細い個所では、デバイスの動作によって電荷密度が低くなることがあります。通常は電源供給部や IC の電源とグランドの間にバイパス・コンデンサ(パスコン)を設置して、電圧降下を回避します。
これが不十分だと、バスの同時動作(all 1 → all 0) など大電流が流れるタイミングで電源の瞬断が発生して、誤動作の原因となることがあります。



電源シーケンス／パワー・シーケンス(Power Sequence)

電源の投入／遮断の順序や時間間隔といった手順のことです。異なる電源電圧で動作するデバイスが基板上に混在する場合や、複数の装置を接続する場合にかかわります。時間間隔については、デバイスによっては規定されていて、基板上の異電源デバイスの電源投入で問題となることがあります。
順序について一般的に、電源投入はメインフレームが先で周辺があと、電源切断は周辺が先でメインフレームがあとです(入力信号の機能によっては、逆になることもあります)。つまり、制御側が被制御側の状態を確定させておく手順にします。
なお、アナログ・デバイスでは特に問題となりませんが、MOS などのデジタル・デバイスでは注意が必要です。それは、一般に MOS デバイスの絶対最大定格で、入力電圧の最大値が「V_{DD}＋0.3V」などと規定されているからです。一方のデバイスの電源だけが ON の状態では、その出力信号がハイ・レベルの場合に、接続されている電源 OFF(V_{DD}＝0V)のデバイスの定格を越えるので、破壊や劣化の原因となります。また、この状態で電源 OFF のデバイスの電源を ON にすると、ラッチアップによるデッドロックが発生する可能性が高くなります。ラッチアップも劣化促進の要因になります。

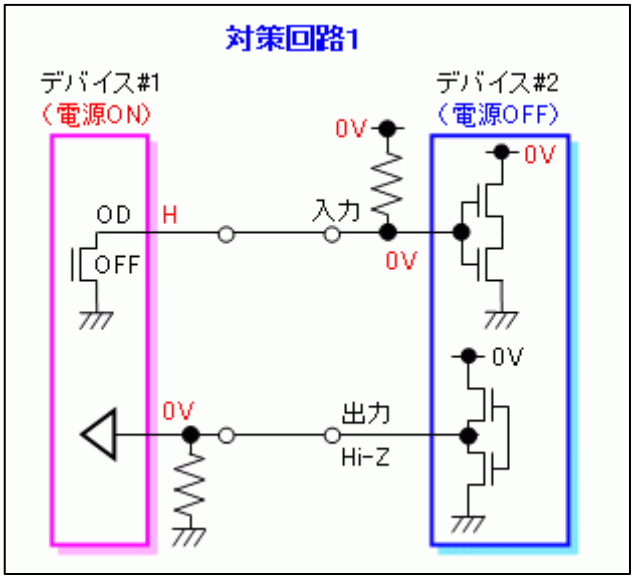
ただし、入力電圧の最大値が「＋6.5V」などの絶対値で定義されている高耐圧バッファの場合、いずれの問題もありません。また、入力の定格電圧以下のロウ・レベルが出力側で保証されている場合も問題もありません。



インタフェース規格で電源シーケンスや回路が定義されていれば、こういった問題に対応しています。ホット・プラグインに対応した USB などもそういった例です。規格で定義されていないインタフェース接続をする場合、出力側をオープン・ドレーンにして、入力側を終端抵抗でプルアップやプルダウンするなどの対策があります。

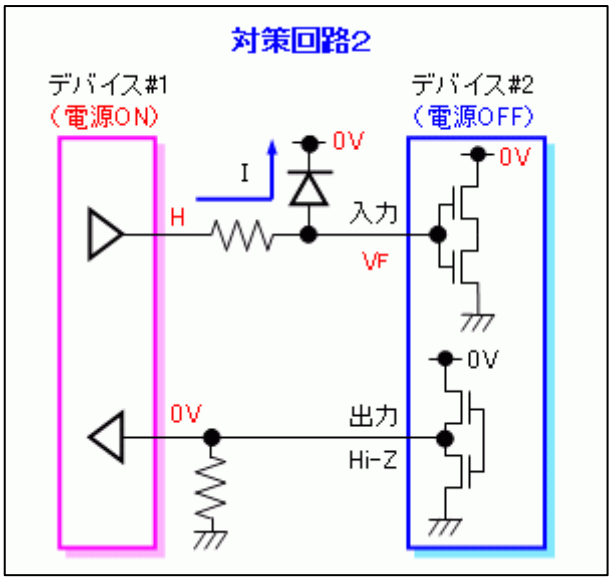
このとき、電源 OFF のデバイスの出力はフローティングなので、ノイズの影響で、電源 ON のデバイスの入力バッファに貫通電流が流れる可能性があります。このため、このラインは抵抗でプルダウンすることが理想的です。

ここで、プルアップでは、電源 OFF のデバイスの出力がダメージを受けます。電源 ON のデバイスへの入力の初期値をハイ・レベルにする必要があれば、電源 OFF のデバイスの出力にプルアップを許容する高耐圧バッファを選定してプルアップする必要があります。



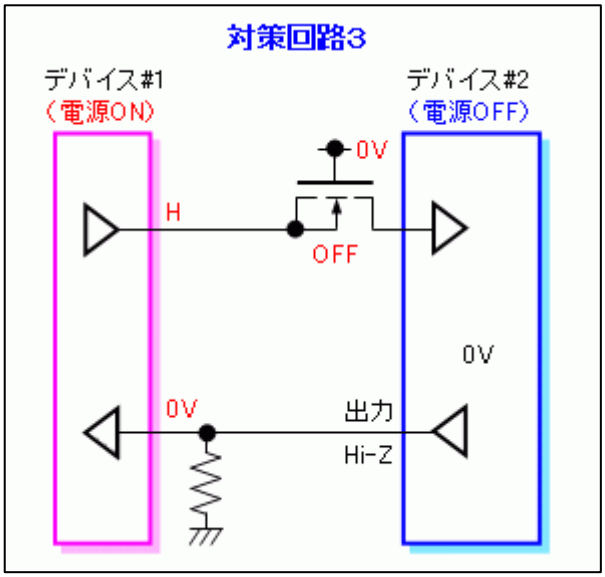
なお、オープン・ドレーン出力は電圧を発生させないので、この対策は電源電圧が異なるデバイスの接続でのトレラントとしても有効です。

また、オープン・ドレーンでなく CMOS 出力を受ける場合、ショットキィ・バリア・ダイオードのように順方向電圧 V_F の低いダイオードと電流制限抵抗で OFF の電源(0V)へ電流を流して保護することができます。ノイズ対策として、基板上の信号線とグランドの間にコンデンサを設置する場合、突入電流への対策としてこの回路が用いられますので、共用することができます。



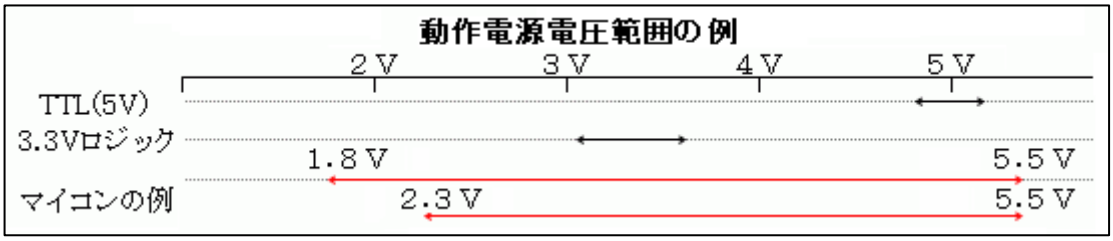
両デバイスの電源が OFF の状態では、MOS の入力と出力がハイ・インピーダンスなので、配線がアンテナとなります。このため、ノイズの多い環境では、デバイスの劣化を促進させる可能性があります。したがってこのような環境では、単一電源の MOS デジタル回路でも、こういった対策回路が有効となります(ノイズは電源の立ち上がりより、かなり時定数が短いので、抵抗よりダイオードの方が効果的)。
ただし当然ながら、正常動作をさせるためのノイズ対策も必要です。

また、ノイズ環境が良好な場合には、MOS FET を利用して、あとで起動するデバイスの電源でインタフェース線をスイッチングすれば、電気的に分離することができます。



電源電圧と動作クロック(Operation Voltage and System clock)

電子回路は使用する電源電圧により動作速度が変化します。一般に、同じ製造プロセスで製造されていれば、電源電圧が高いほど高速で動作できます(これまでは、製造プロセスが微細化するほど高速動作が可能でした)。
昔のロジック回路は、長らく 5V での動作が大半でしたが、その後 3.3V となり、現在はさまざまな電源電圧で用いられています。
最近のマイコンは CMOS 化されたことで、1.8V や 2.3V から 5.5V まで、広い電圧範囲で動作するものもあります。



そのため、マイコンによっては、電源電圧に応じて最高動作周波数が、何通りか決められているものもあります。また、最高動作周波数と別に、発振回路の発振可能周波数も電源電圧により異なる場合があります。

マイコンの動作クロックとしては、入力された(または発振した)クロックを複数の分周比で分周したものの中から切り替えて選べます。通常は低い周波数のクロックで起動するようになっているので、電源電圧を検出できれば、電源電圧に応じて適した周波数をソフトウエアで選択すればよいことになります。ところが、発振周波数は使用する発振子で固定されます。つまり、発振周波数に対応した電源電圧になるまでマイコンが動作しないようリセットをかけておくなど、ハードウエアで対策しておく必要があります。「クロックとリセット」もご参照ください。

【ティー・ブレーク】

IC の電源端子には、V_{CC} や V_{DD} があります。また、グランド端子には、GND や V_{SS} があります。

V_{CC} はバイポーラ IC の内部で、コレクタ同士を接続(C-C)していたことから、このように呼ばれています。

また、V_{DD} は NMOS IC の内部で、ドレーン同士を接続(D-D)していたことから、このように呼ばれています。同じく NMOS IC の内部で、ソース同士を接続(S-S)していたことから、グランドは V_{SS} と呼ばれています。これらは 2 電源 IC だったころのなごりですが、1 電源の CMOS になっても、NMOS での表記を継承しています。グランドについては、バイポーラ IC と同じ GND 表記が一般的となっています。

なお、メモリについては、JEDEC(米国の業界団体)で端子名を定義しており、V_{CC} が使用されているので、CMOS であってもこれに準拠して、慣例的に電源端子を V_{CC} と表記しています。

つまり、製品によって表記を変えているだけで、電源とグランドとしての位置づけは同じです。

電磁シールド(Electromagnetic Shield)

金属箱や金属板、金属メッシュによって電磁波を遮蔽し、EMI や EMS を回避する方法です。電磁波がこれらのシールド材を通過しようとすると渦電流が発生し、電磁波を打ち消そうとします。

一般には、シールド材を接地することによってグランド電位とし、遮蔽効果を高めます。テレビとアンテナの接続などに使用される同軸ケーブルは、芯線が導線メッシュに覆われているので、シールド線になっています。

なお、電磁シールドには、次のような留意点があります。

- ノイズ発生源のシールドは、ノイズを受ける側のシールドより重要だが難しい。
- 完全にシールドしないで弱い部分があると、そこにエネルギが集中する。
- 金属箱や金属板は、金属メッシュより放熱効果が低い。

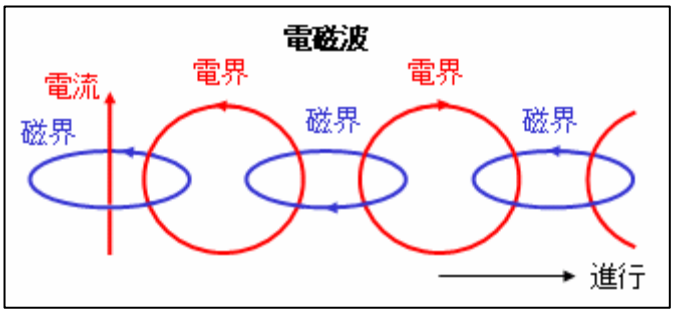
したがって、電磁シールドは絶対的なものではなく、ノイズ対策の一手段なので、ノイズが発生しにくい根本的な対策や、他の方法と組み合わせることが望ましいといえます。

【ティー・ブレーク】

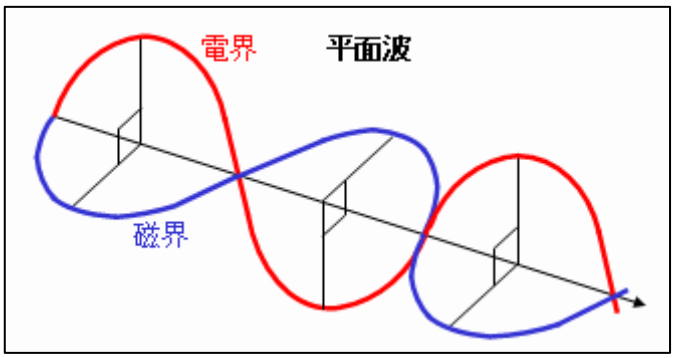
ノイズ・エネルギを分散させて集中を避けるには、全体の遮蔽効果を下げて緩和する方法や、複数要因に時間差をつけて緩和する方法などがあります。ルネサス エレクトロニクスでは、クロック・ジェネレータに時間差機能を内蔵した SSCG(Spread Spectrum Clock Generation)をアナログ・コアとして提供しています。32 ビット・マイコン V850E/ME2 や V850E2/ME3 などにこれを搭載しており、IC での EMI 対策を実現しています。

電磁波と電波（Electromagnetic Wave and Electric Wave）

音や光と同様に、電気の振動は、水面の波紋のように、波となって広がります。このとき、電流によって生じる磁界が、直交する電界を発生させ、この電界によって直交する磁界が生じ、これらが連鎖して空気中に放射されたものが電磁波です。



電波は、赤外線より波長が長い電磁波のことです。電波法では、3THz(=3000GHz)以下の周波数の電磁波を電波と定義しています。電波の単純なものは平面波で、電界と磁界の波が、互いに直交しています。



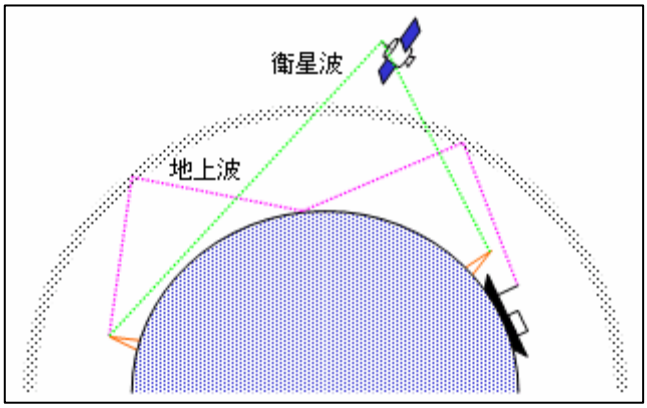
自由空間での電波の伝搬速度は、光に等しく 30 万 km/s です。つまり、周囲 4 万 km の地球を1秒に7周半する速さです。なお、電波は周波数によって、次のように分類されています。

周波数	分類	俗称	波長
3kHz	ELF	長波	100km
30kHz	VLF		10km
300kHz	LF	中波	3km
3MHz	MF	中短波	1km
30MHz	HF	短波	200m
300MHz	VHF	超短波	50m
3GHz	UHF	極超短波	10m
30GHz	SHF	マイクロ波	1m
300GHz	EHF	ミリ波	10cm
3THz		サブミリ波	1cm
			1mm
			0.1mm

種　　類	用途の例
ELF(Extremely Low Frequency)	(大規模な設備が必要で伝送量が少なく、あまり利用されない)
VLF(Very Low Frequency)	オメガ(電波航法)、潜水艦無線
LF(Low Frequency)	長波ラジオ、LORAN-C、電波時計、船舶無線
MF(Medium Frequency)	AM ラジオ、アマチュア無線、NDB/ADF(無線航法)、船舶気象通報
HF(High Frequency)	短波ラジオ、アマチュア無線、市民バンド、船舶無線、航空無線
VHF(Very High Frequency)	FM ラジオ、テレビ、アマチュア無線、MKR/LOC/VOR(無線航法)、民間航空無線
UHF(Ultra High Frequency)	テレビ、アマチュア無線、ATC/GS/DME/TCAS/TACAN(無線航法)
SHF(Super High Frequency)	衛星テレビ／通信、アマチュア無線、無線 LAN、電子レンジ、携帯電話
EHF(Extra High Frequency)	レーダ、衛星通信
サブミリ波	光通信、宇宙電波天文、非破壊検査

実際には、光も電磁波の 1 種です。このため、光の波長ごとの屈折率の違いで虹が見られるように、電波も波長によって伝達特性が異なります。

たとえば、ラジオで使用している MF や HF では、地表と電離層の間で反射を繰り返して、地球を回りこんで伝わります(地上波)。これに対して、衛星放送で使用するマイクロ波では、直線性が強いため、地平線や水平線の先には回りこみません。また、電離層を突き抜けるので衛星に届け、衛星から地球の裏側へ伝えることができます。



【ティー・ブレーク】

AM ラジオは、数百 kHz 程度の搬送波に音声を載せて送信しています。つまり、この程度の周波数でも、電波として利用されています。マイコン・システムでは、数 MHz 以上(早いものは数百 MHz)のクロック・スピードで動作していますので、電波発生の可能性があり、これはノイズの原因となります。

こういったノイズ源となる例はスパーク・ノイズにも多く見られ、ヘア・ドライヤのモータや、2輪車のエンジン、雷などの影響で、AM ラジオからノイズ音が出たり、テレビの画像にちらつきが出たりします。また、電池点火式の石油ストーブが、2輪車のスパイク・ノイズで点火してしまい、火災を起こす事故なども起きます(電池ははずして収納しましょう)。

電磁誘導(Electromagnetic Induction)

磁束が変動する環境にある導体では、その変化に比例して電位差(起電力)が生じる現象です。これは、容量の単位**ファラッド**の語源になった、イギリスのマイケル・ファラデーが発見したものです。この起電力は、磁束変化を妨げる電流を生じる向きに発生(**レンツの法則**)し、その大きさは**ノイマンの定理**で、次のように表されます。

$V = -d\Phi/dt$

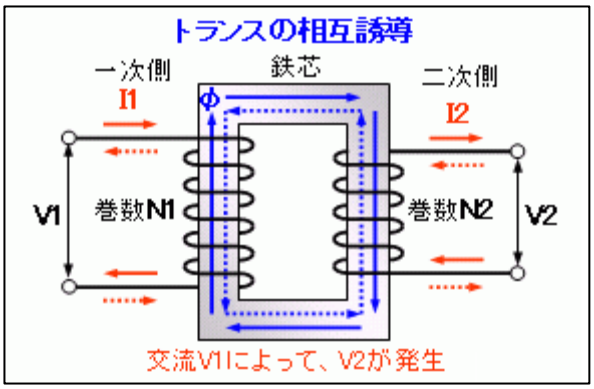
電気回路では、この起電力がノイズ源となります。
また、この磁界をもとに電磁波が発生し、離れた場所にもノイズの影響を与えます。
なお、誘導性負荷を駆動する電圧が変化すると、それに伴う電流変化によって磁束変化が生じ(アンペールの法則)、電圧変化とは逆向きの逆起電力がサージとなって発生します(インダクタンスが大きく、電流変化が急激なとき、10倍程度になることもある)。これが誤動作やデバイス破壊の原因となる程度のものであれば、ダイオードなどによるサージ吸収回路が必要です。

電磁誘導を利用した部品にコイルがあります。
コイルに大電流を流すと、強力な磁場が発生します。ここに電気を通しやすい鉄やステンレスなどの金属を置くと、電磁誘導で金属内に**渦電流**が発生して、金属の抵抗によって発熱(**ジュール熱**)します。この原理は**電磁誘導加熱**(IH:Induction Heating)と呼ばれ、IHクッキング・ヒータなどの調理器に応用されています。

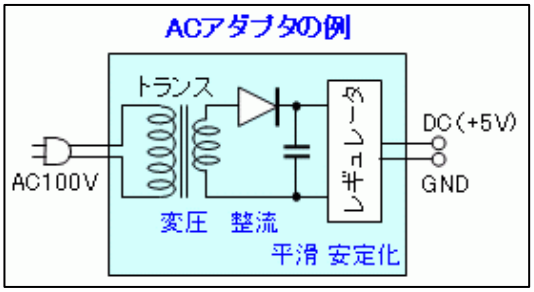
【ティー・ブレーク】
抵抗での電力消費(W)による発熱をジュール熱と呼びますが、その単位は J(ジュール)です。
物理的な熱量の単位は cal(カロリ)ですが、1 カロリとは 1ml の水の温度を 1℃上昇させる熱量です。
ここで、両者には次の関係があります。
1cal=4.2J または 1J=0. 24cal
ダイエット運動で汗を流している人は、熱放射によって確実にカロリ消費をしているのです。



またチョーク・コイルは、自己誘導インダクタンスを大きくして、直流を通すけれど、交流を通さないようにしたもので、フィルタや EMC 対策に利用されます。
トランス(変圧器)では、相互誘導インダクタンスを利用して、あるコイルの電流の変化で、隣接するコイルに誘導起電力を生じさせています。これにより、相互の巻数に比例して、交流電圧の昇圧や降圧ができます。



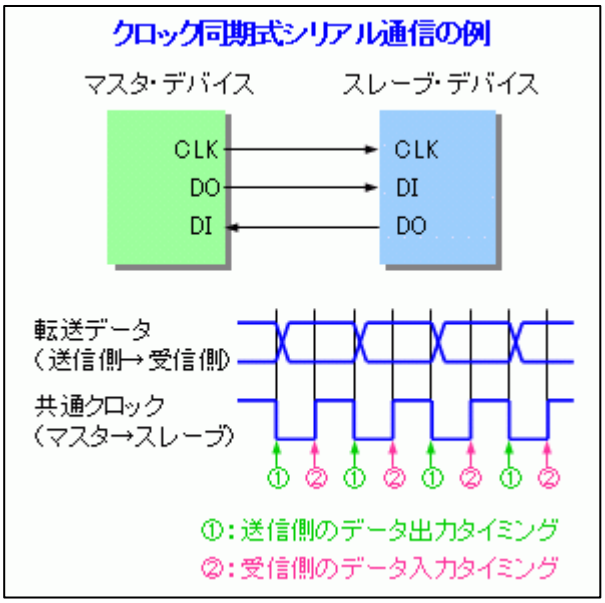
身近なトランスでは、携帯機器などの電源 AC アダプタや充電器に内蔵したものや、電柱の送電(AC6600V)から家庭へ引き込む(AC100/200V)ため変圧をする柱上トランス(電柱に設置されている円筒形のもの)があります。



なお、電流容量が小さい AC アダプタでは、シリーズ・レギュレータやスイッチング・レギュレータによる安定化回路がない場合があります。

同期(Synchronization)

待ち合わせ制御や順序制御によって、複数の事象が同時に発生したり変化したりすることです。たとえばクロック同期では、複数の機能ブロックが共通のクロックを基に動作することによって、ブロック間を同期化させます。
シリアル通信を例にあげると、クロック同期やキャラクタ同期で、送信側と受信側を同期化してデータを転送しています。クロック同期では、たとえばマスタ・デバイスが基準クロックをスレーブ・デバイスに出力して、双方でこれに同期した送受信動作をします。



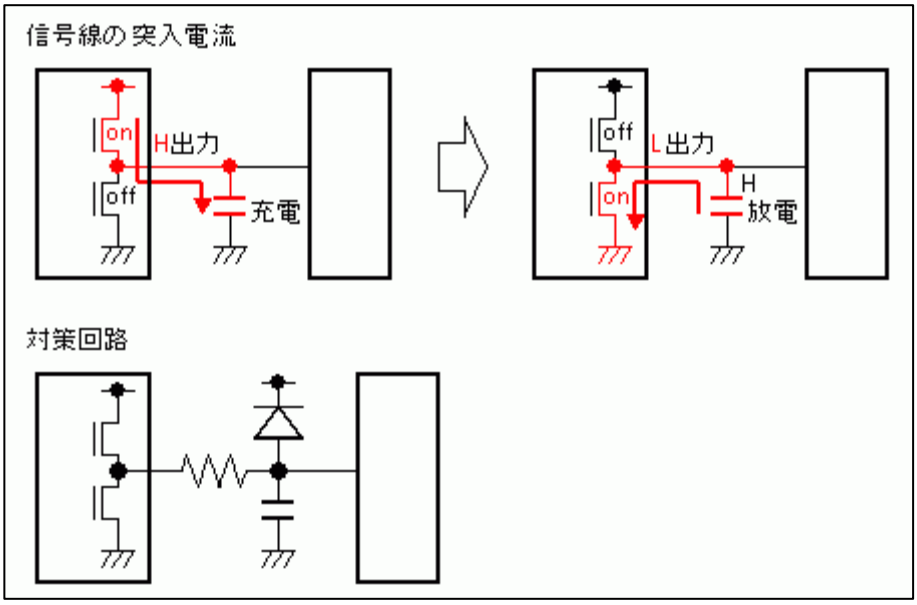
またキャラクタ同期では、送信側と受信側の転送レートを合わせておき、送信側が特定のキャラクタ・パターンを送信して、受信側がこれをフレームの開始と認識し、双方でそれに続くデータ列を送受信します。
なお、UART(Universal Asynchronous Receiver/Transmitter)の通信方式は調歩同期と呼ばれますが、あくまでもAsynchronous なので、本来の同期式とはいえません。いわばビット同期で、スタート・ビットでフレームの開始、ストップ・ビットでフレームの終了と認識するだけですから、ボー・レート誤差やジッタによるビットずれが原因で、誤動作する可能性が高いといえます。
ただし LIN バスでは、基本動作は UART ですが、通信開始時にシンク・フィールド(55H)を送受信して、転送レートを合わせるキャラクタ同期の機能があります。
なお I^2C バスでは、スレーブがクロック信号(SCL)をロウ・レベルに固定し、マスタにクロックをハイ・レベルに立ち上げさせないようにして、ビット同期をとることがあります。

同時動作(Simultaneous Operation)

アドレス／データ・バスのような多ビットのラインが、同時に同じ状態変化をすることです。
特に全ビットが"1→0"の変化をする場合、瞬時的に駆動デバイスのグランドに電流を引き込むことになり、ノイズ性の誤動作を起こすことがあります。
そのデバイスの電源-グランド間にバイパス・コンデンサを設置して、影響を軽減するのが一般的です。
デバイス内部でも、各信号の変化タイミングを細かくずらして回避している製品もあります。
ゲート・アレイなどでは、ユーザ仕様に応じてポート数を定義できてしまうため、同時動作のビット数を制限することがあります。

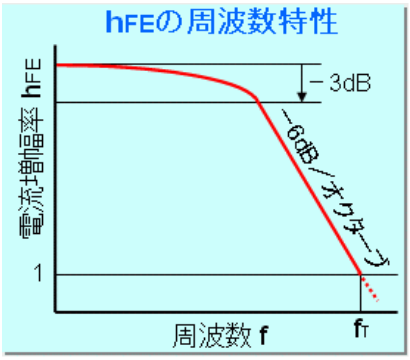
突入電流(Rush Current)

コンデンサの充放電によって、短絡的に流れる電流のことです。
ひとつには、電源投入時に、平滑コンデンサやバイパス・コンデンサへの充電で、電源ユニットから流れ出す大電流があります。
もうひとつは、デジタル信号線のノイズ対策として、グランドとの間にコンデンサを設置する場合で、信号の変化時に駆動側が短絡状態となって大電流が流れ、特性の劣化や誤動作の原因となります。
これを抑制するには制限抵抗の挿入が有効となります。
また、電源遮断時には、充電されたコンデンサによって入力が電源(0V)より高くなり、絶対最大定格を越えるため、電源へ放電する保護ダイオードも必要となります。
このため、むしろプルアップ抵抗などの方が簡易なノイズ対策と言えます。



トランジション周波数(Transition Frequency)

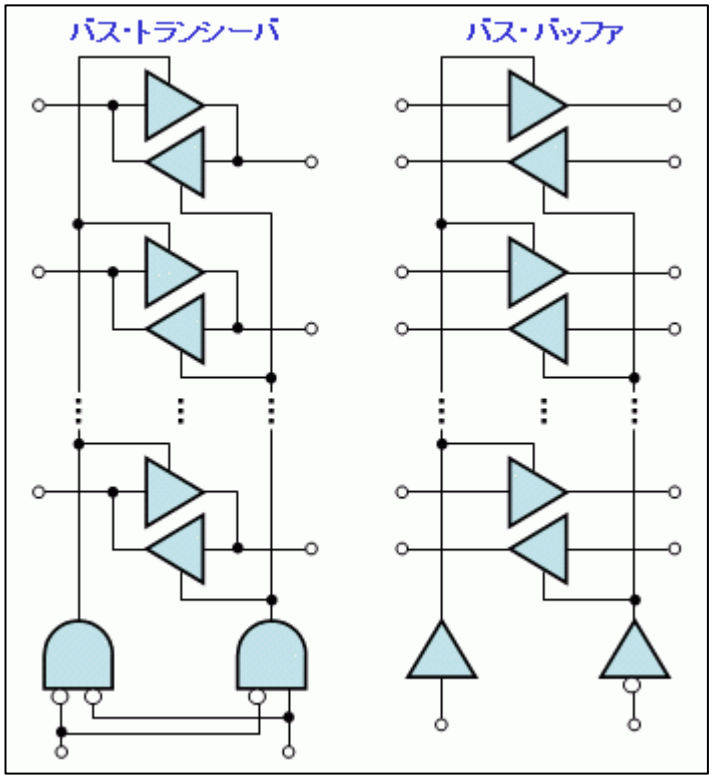
トランジスタのエミッタ接地での増幅率が1になる周波数のことです。そのトランジスタが、どのくらいの高周波信号までを増幅できるかという限界を示します。一般には利得帯域幅積(fT)と呼びます。



関連項目 周波数特性

トランシーバ(Transceiver)

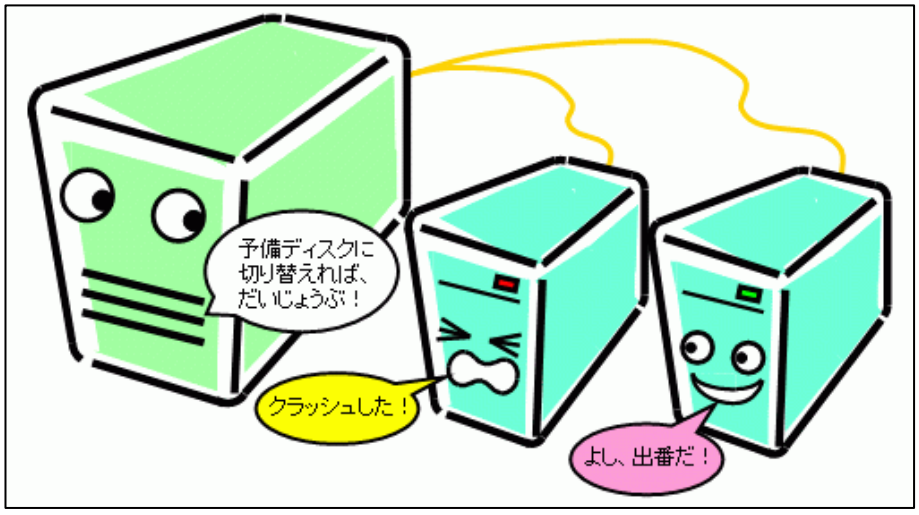
トランスミッタ・レシーバの略称で、送信機能と受信機能を組み合わせたものです。一般には無線機のトランシーバを連想しますが、デジタル回路では双方向バスの入出力切り替えを行うバス・トランシーバが利用されます。これにより、バス・アービトレーションができます。バス・バッファとの違いは、バッファの入力と出力が双方向で共通になっていることです。



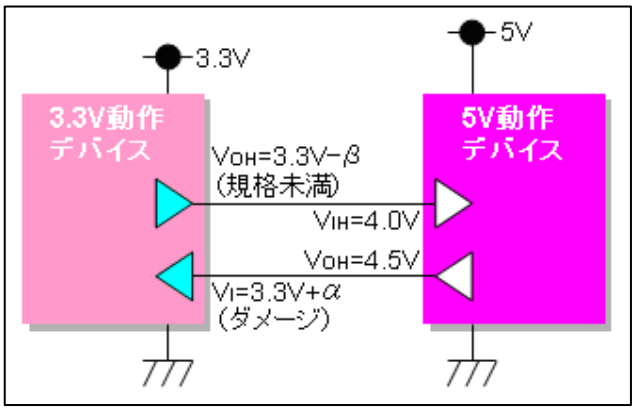
トレラント(Tolerant)

障害に対応して、システムを円滑に運用するしくみのことです。たとえば、「フォールト・トレラント」とか「5V トレラント」というしくみがあります。

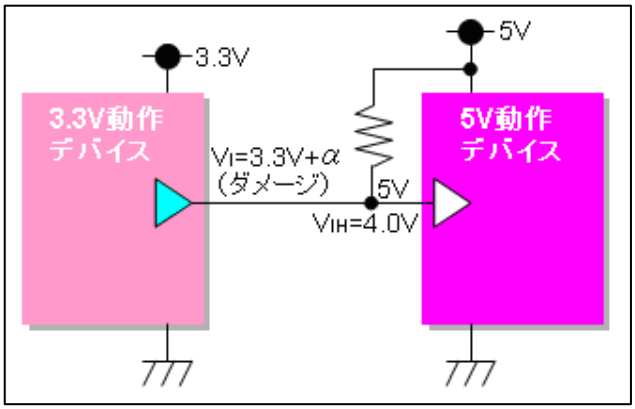
フォールト・トレラントとは、システムに障害が発生しても、停止させずに継続運転ができるようにすることです。コンピュータ・システムの例では、電源ユニットや RAID のハード・ディスクなどのように二重化しておいて切り替えたり、ハード・ディスクやメモリなどのデータ・エラーを ECC などの技術でエラー訂正したりするしくみがあります。



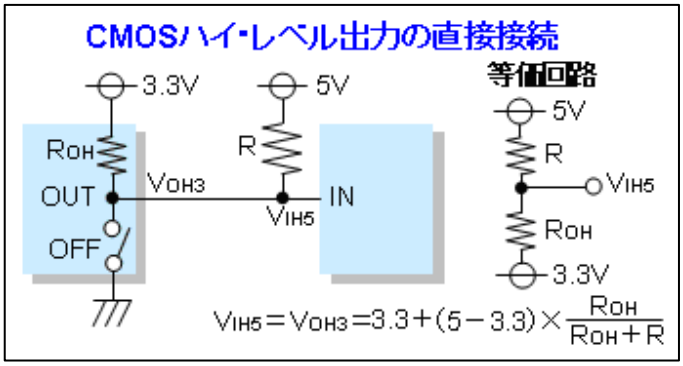
★ また、5V トレラントというのは、電源電圧の異なる半導体デバイス（この場合、5V デバイス）との接続を可能にする内部 I/O バッファの特性のことです。たとえば通常の 3.3V 動作デバイスでは、入力の絶対最大定格の最大値が「3.3V＋α」なので、5V デバイスからのハイ・レベル入力は、これを越えるため許容できません。
出力についても最大値が「3.3V－β」なので、これを入力する 5V デバイスが CMOS なら、ハイ・レベル入力に満たないため許容できません（TTL では V_{IH}＝2V なので問題ありません）。



また、5V電源へのプルアップ抵抗で、3.3V動作デバイスの通常のCMOS出力を引き上げようとすると、3.3Vバッファの出力に約5Vが印加され、絶対最大定格を越えるため許容できません。



そこで、3.3V動作デバイスの入出力に中耐圧のI/Oバッファを使用することにより、信頼性の問題を回避できます。
ただし、「**CMOS出力バッファ**」の場合、外部で5Vにプルアップしても、出力電圧を十分に引き上げることができない可能性があります。これは、ハイ・レベル出力（3.3V電源に接続されているPチャネル・トランジスタがON）の電圧が、双方の電源電圧差を出力Pチャネルのオン抵抗（出力インピーダンス：電圧／電流比）とプルアップ抵抗で分割した値となるためです。

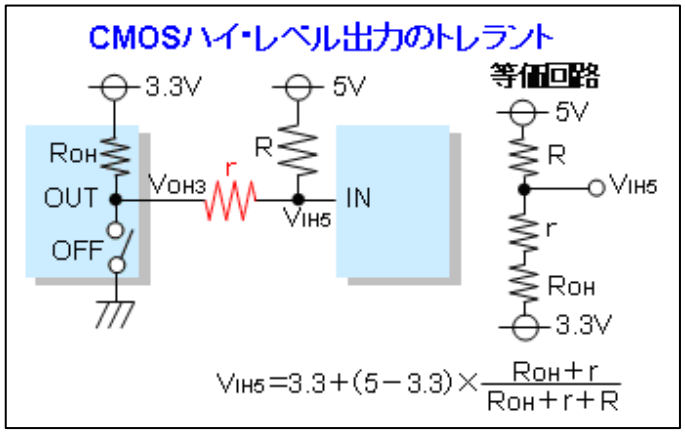


たとえば、プルアップ抵抗R＝1kΩで、Pチャネルのオン抵抗R_{OH}＝500Ωだとすると、5Vバッファの入力電圧は次のとおりです。

V_{IH5}＝3.3＋(5－3.3)×0.5/(0.5＋1)＝3.9(V)

ハイ・レベル入力電圧V_{IH6}＝0.8V_{DD}(min.)の規格であれば、V_{DD}＝5Vのときに4.0V以上の入力が必要となり、この場合にはインタフェースできないことになります。

中耐圧CMOSバッファの出力を使用する場合は、出力端子とプルアップ抵抗の間に直列抵抗を挿入してください。

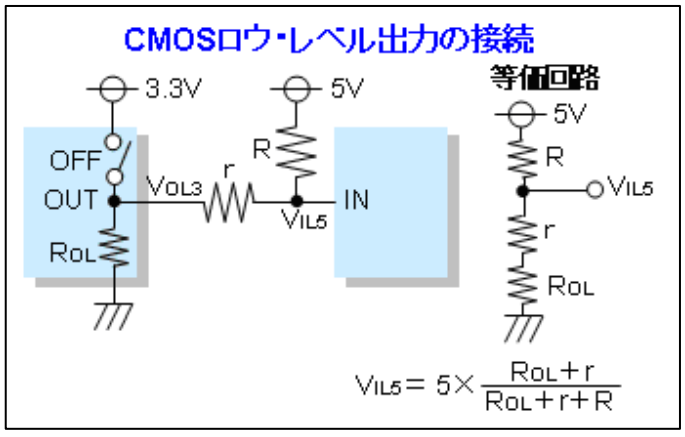


前述の例で、r＝200Ωの抵抗を挿入すると、5V バッファの入力電圧は次のようになります。

V_{IH5}＝3.3＋(5－3.3)×(0.5＋0.2)/(0.5＋0.2＋1)＝4.0(V)

これで、理論的にはインタフェースできることになります。ただし、実際にはノイズ・マージンや、算出できる出力インピーダンスが最大値であることを考慮する必要があります。

また、ロウ・レベル出力についても確認が必要です。



前述の例で、出力 N チャネルのオン抵抗 R_{OL}＝100Ωだとすると、5V バッファの入力電圧は次のようになります。

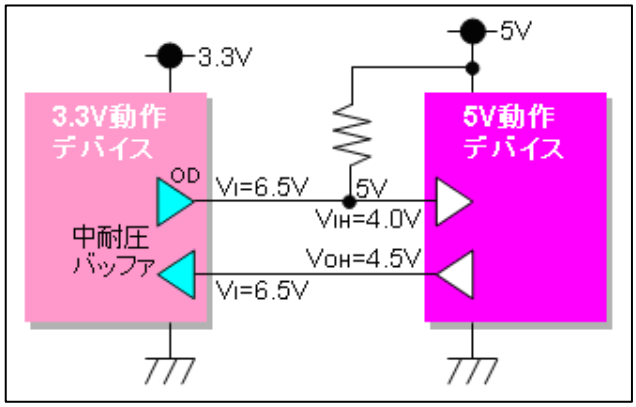
V_{IL5}＝5×(0.1＋0.2)/(0.1＋0.2＋1)＝1.2(V)

ロウ・レベル入力電圧V_{IL5}＝0.3V_{DD}(max.)の規格であれば、V_{DD}＝5Vのときに1.5V以下の入力でよいことになり、この場合にはインタフェースできます。もし、TTLレベルの0.8V以下の入力が必要であれば、インタフェースできないことになります。

プルアップ抵抗値Rが小さければハイ・レベルに余裕ができ、大きければロウ・レベルに余裕ができます。また、直列抵抗は遅延の要因になります。これらの点も含めて、挿入抵抗値rの検討が必要です。

ただし、適正化を図った設計でも、電流がプルアップ抵抗に常時流れ、しかもそれが低抵抗でなければならないので、CMOS出力のプルアップはお勧めできません。

設計を容易にするため、通常は中耐圧の「**N チャネル・オープン・ドレイン・バッファ**」を出力に使用します。この場合は、外部で 5V にプルアップするだけで済みます。



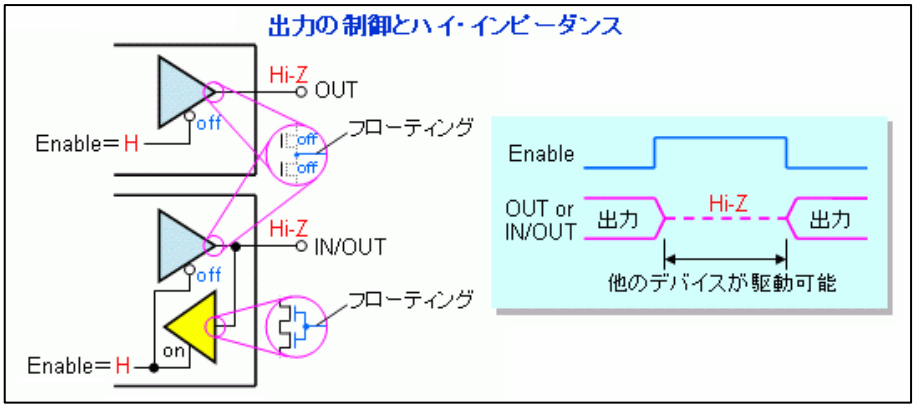
5V 動作デバイスの入力バッファに内部プルアップ機能があれば、それも利用可能で、デバイス間の外部部品は不要になります。

熱暴走(Thermal runaway)

トランジスタの入力電力から、信号として出力電力を引いた分は、トランジスタ内部で消費され、接合部の温度が上昇しますが、発熱と熱放散が平衡していれば、特に問題はありません。
しかし、この発熱と熱放散が平衡しないと、温度上昇により I_{CBO},V_{BE},h_{FE} などが、すべてコレクタ電流を増加させる変化をして、接合部温度をさらに上昇させます。これが循環して、接合部が溶融して破壊することが熱暴走です。

ハイ・インピーダンス(High Impedance)

電気信号に対して高抵抗を示す特性のことで、「Hi-Z」で表します。具体的には、MOS デバイスのデジタル動作などで、出力信号を遮断して外部と切り離したり、入出力兼用端子を入力に切り替えたときに発生する状態です。
これらの状態は、1 本の信号線を複数の出力が共用するときに、信号の衝突を避けるために発生させます。内部では、その信号線に対してわずかなリーク電流しか流れず、浮いた状態(フローティング)となます。なお、MOS デバイス同士の接続で Hi-Z 状態では、他に駆動源がないと入力(出力端子の場合、接続先の入力)がフローティングとなり、入力バッファに貫通電流が流れる恐れがありますので、プルアップなどの処置が必要となります。



バイアス(Bias)

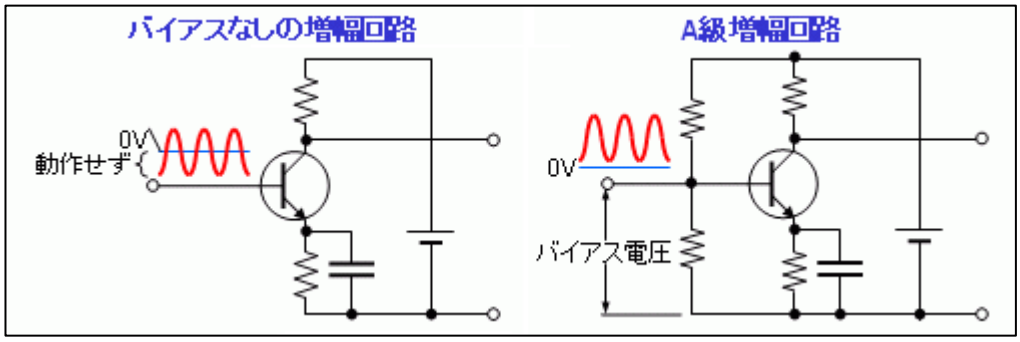
処理ができる状態になるように行う「かさ上げ」のことです。

たとえばプログラムの数値処理で、−128～+127 を扱う場合、バイアスとして 128(80H)のオフセットを定義すれば、0～255(0～FFH の 8 ビット)で負数も表現できます。この場合、00H～80H が−128～0、81H～FFH が 1～127 に相当し、演算結果が 80H より小さければ、負数ということになります。

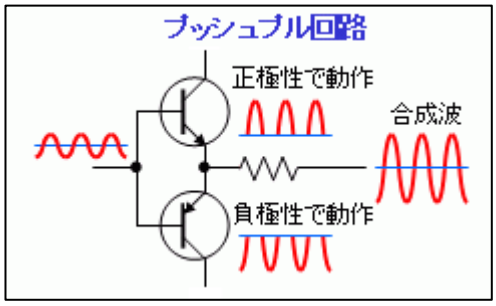
【ティー・ブレーク】

負数表現には、MSB を符号とみなす 2 の補数もあります。この場合も−128～+127 は 8 ビットで扱われ、MSB であるビット 7 が 1 なら負数を表し、00H～7FH が 0～127、80H～FFH が−128～−1 に相当します。

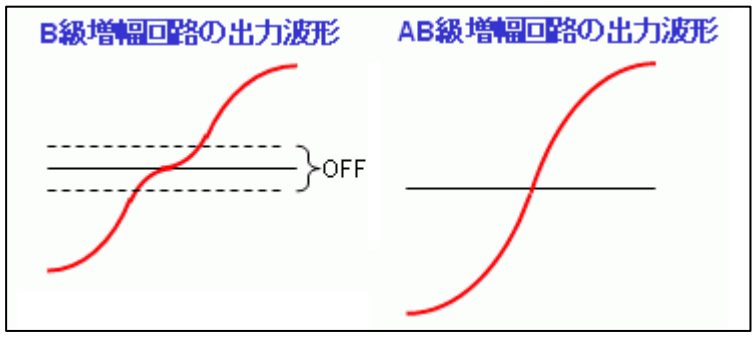
電気回路でのバイアスは、動作可能となるよう、直流電圧をかけることです。増幅回路では、交流信号の負電圧部分で動作するように、バイアス電圧でオフセットをかけて正電圧側に持ち上げます。たとえばバイポーラ・トランジスタの「A 級増幅回路」は、ベースにバイアス電圧をかけ、0V 以下に振動しないよう、常に能動状態にします。



A 級増幅回路は、バイアスの分だけ消費電力が多くなります。そこで、NPN トランジスタと PNP トランジスタを組み合わせたプッシュプル回路で、正負交互に動作させて出力を合成し、消費電力を低減したのが「B 級増幅回路」です。プッシュプル回路には、コンプリメンタリ(特性が同等で逆極性)の 2 つのトランジスタが使用されます。



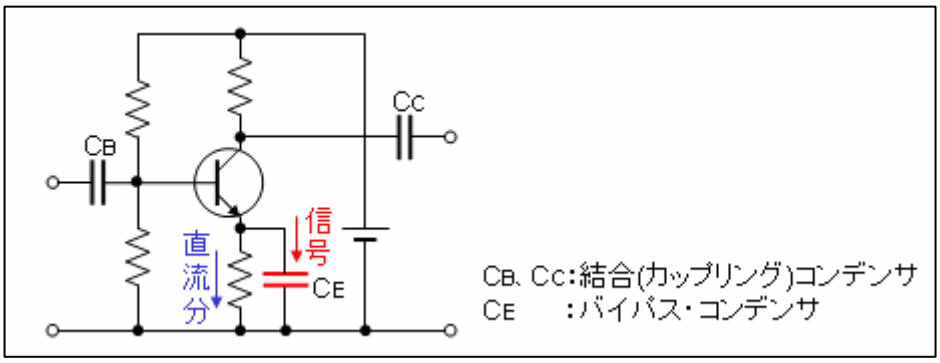
各トランジスタのベース−エミッタ間のオン電圧(0.6V 程度)より 0V に近いベース信号では回路が動作しないため、B 級増幅回路の 0V 付近では、合成出力にひずみが生じます。「AB 級増幅回路」では、オン電圧相当のバイアスをかけて、合成波がひずまないように動作させます。



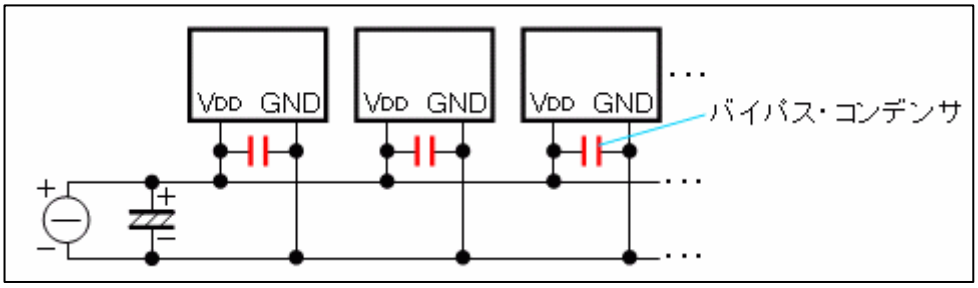
【ティー・ブレーク】
トランジスタや真空管の増幅回路では、ほかに C 級と D 級の増幅回路があります。
「C 級増幅回路」は素子にバイアスを与えないで、入力電圧が高い部分だけを増幅するもので、高調波成分をフィルタで抽出して、周波数逓倍(ていばい)回路などに応用されます。
「D 級増幅回路」は素子の飽和領域を使用して、デジタルのスイッチング動作をするもので、純粋な増幅回路ではありません。

バイパス・コンデンサ(Bypass Capacitor)

AC 成分を迂回させるコンデンサの使用方法で、パスコンとも呼ばれ、2通りの形態があります。
1つはアナログの増幅回路に使用され、信号を電流制限抵抗で減衰させないよう、抵抗と並列に接続されます。



もう1つは基板上の要所の電源−グランド間に設置され、基板上の電源ノイズを吸収したり、電流密度の低い部分を補う平滑コンデンサの役目をしたりします。大電流の流れるデバイスや高速動作をするデバイスの付近では、特に重要になります。



必要なコンデンサの容量は、システム(基板のパターンや電源容量、ノイズ環境など)によって異なりますが、次のような目安です。
LSI の各電源端子の近くに、セラミック・コンデンサ(0.数 uF)を1つと、ゲート IC などの SSI や MSI には、数個に1つ同様のセラミック・コンデンサを付けます。また、基板全体ではその大きさにもよりますが、数個の電解コンデンサ(数十μF)を付ければいいのではないでしょうか。
セラミック・コンデンサは、できるだけ周波数特性のよいものをお勧めします。大容量の積層セラミック・コンデンサには、周波数特性の悪いものがあるので、あまり大きい容量のものは避けてください。
また、クロックのような高い周波数の信号を扱う IC については、LSI でなくても近くにセラミック・コンデンサを付けることをお勧めします。
大電流を扱う部分には、その近くに電解コンデンサを付けてください。

パイプライン(Pipeline)

CPU が命令の実行を複数のステージに分けて、各ステージを処理するユニットごとに、順次処理を進める機構です。これにより、命令の実行終了を待たずに次の命令を実行できるので、プログラム実行を高速化できます。インストラクション・サイクルが次のような 5 ステージに分かれている場合、5 段パイプラインと呼びます。

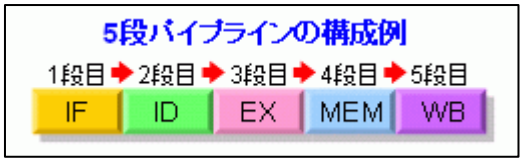
IF : 命令フェッチ

ID : 命令デコード

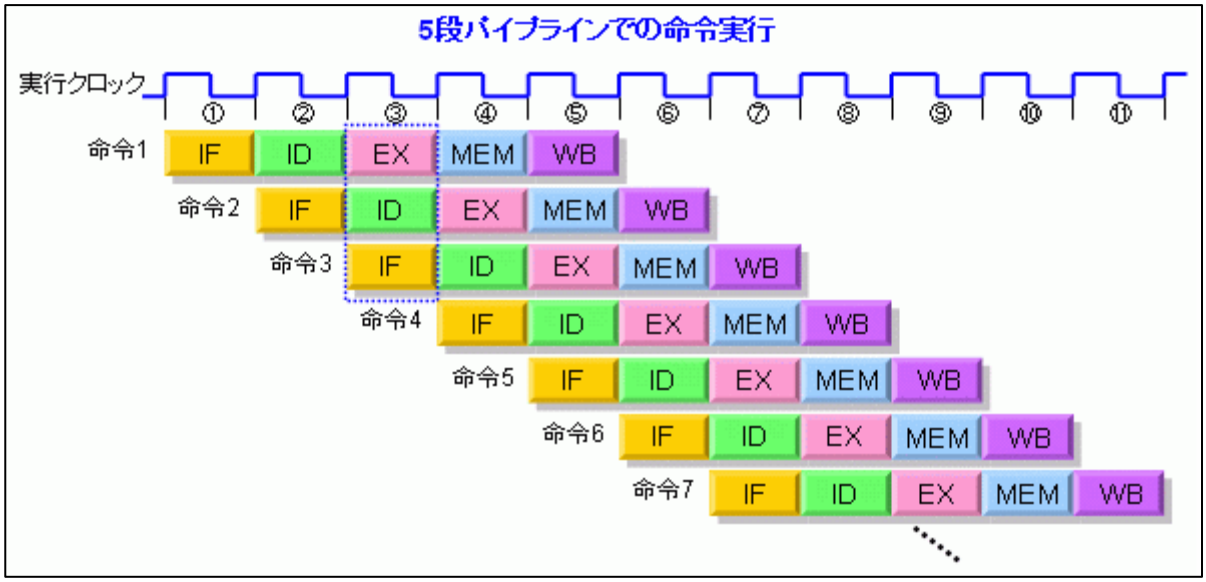
EX : ALU などでの演算

MEM : メモリ・アクセス

WB : 実行結果をレジスタに書き込み



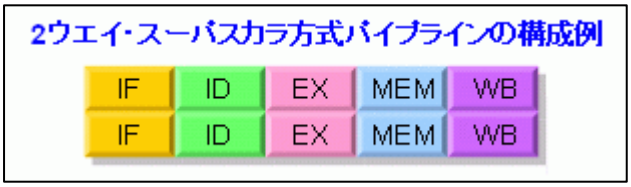
各ステージでは処理手順が進むだけで、データそのものを受け渡すとはかぎりません。ID では IF のデータを受けて、それを解析しますが、EX 以降は ID での解析結果そのものを受けるのでなく、それに対応する処理をします。命令フェッチは 1 度に 1 命令で、実行クロックごとに各ステージの処理が進みますので、各命令のパイプライン処理は、1 クロックずつずれることになります。



クロック・サイクルごとのパイプライン処理は、上図でそのクロック・サイクルを縦に見た内容となります。たとえば 3 番目のクロック・サイクルでは、破線で囲んだように、IF で命令 3 のフェッチ、ID で命令 2 の解析、EX で命令 1 の実行を行います。

なお、命令間の依存性やハードウエア資源の競合、分岐命令などがあると、停止（ストール）や再実行どが必要となり、かえって処理速度が低下することになり、これを「パイプライン・ハザード」と呼びます。命令間の問題については、コンパイラがあるていど命令の順序を操作して対応しています。分岐については、分岐予測機能によって、分岐先の命令を先に実行しておくことによって、あるていど回避されます。

より高速化を図るために、複数のパイプラインを並列に組み合わせて、並列処理をさせる構成をスーパスカラ方式と呼びます。たとえば、2 ウエイのスーパスカラ方式では、同時に 2 命令をフェッチします。



バス・アービトレーション(Bus Arbitration)

複数のデバイスが共通のバスを使用して、他のデバイスをアクセスしようとした場合に、調停してバスの使用権を割り当てることです。この調停回路をバス・アービタと呼びます。
このとき、使用権を得たデバイスが、バス・マスタとなります。
バス・マスタがアクセスを終えてバスを解放すると、バス・アービタは、バスの使用権を要求している他のデバイス間で、再び調停を行います。

バス・サイクル(Bus Cycle)

マイコンが命令実行をするときに、インストラクション・サイクルのうちバスを駆動する期間のことです。CPU の基本機能には、演算と I/O やメモリに対する入出力がありますが、このうち入力と出力でバスを駆動し、それぞれリード・サイクルとライト・サイクルとなります。
最近のマイコンでは、内部処理が高速化されており、内部バスと外部バスで処理速度が異なります。マイコン内部のメモリにある命令の実行では、1バス・サイクルで命令フェッチができます。

【ティー・ブレーク】
バスはひとかたまりのデータを同時に伝える束線のことで、乗り物のバスと同じイメージなので、このように呼ばれています。通常はアドレス・バスとデータ・バスを指し、これらによって、複数のデバイスの接続が容易になります。

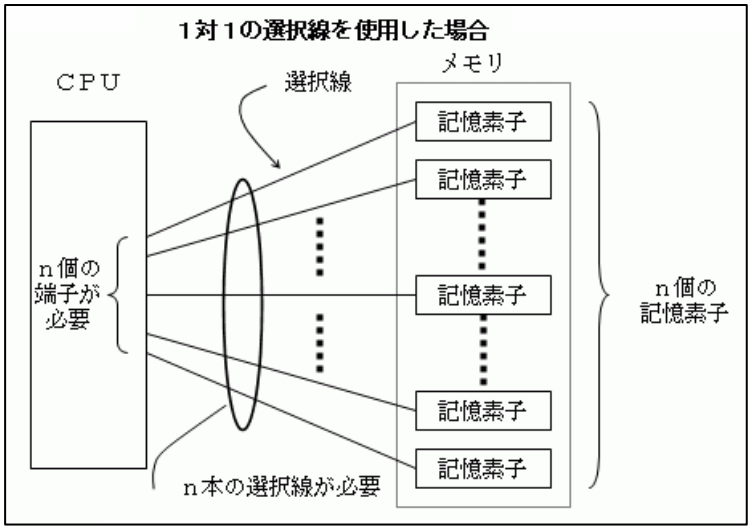


バス・サイジング(Bus Sizing)

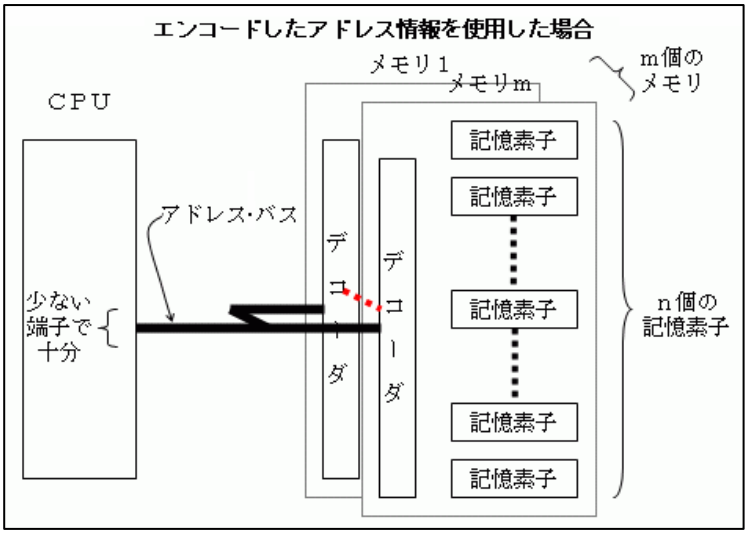
アクセス対象デバイスのデータ・バス幅に応じて、バス・マスタ・デバイスの外部データ・バス幅を設定または変更することです。たとえば、外部 32 ビット・データ・バスの CPU で、メモリや I/O をアクセスする場合、そのままでは 32 ビット・データ・バスのメモリや I/O でなければ接続できません。しかしバス・サイジング機能で、CPU の外部データ・バスを 16 ビットに設定できれば、16 ビット・データ・バスのメモリや I/O を接続できるようになります。
バス・サイジングには、起動時に端子電位で設定するだけのスタティック・バス・サイジングと、動作中に端子電位やレジスタ設定で変更できるダイナミック・バス・サイジングの2種類があります。
また、メモリ空間をいくつかに分割して、それぞれに異なるバス幅を指定できる CPU もあります。

バス接続と 1 対 1 接続(Bus Connection and Point to Point Connection)

基本的に信号は出力(point)から入力(point)に接続します。したがって、メモリのように多くの記憶素子から構成されたパーツで、個々の記憶素子を1対1接続で選択しようとすると、配線が記憶素子の数だけ必要になります。

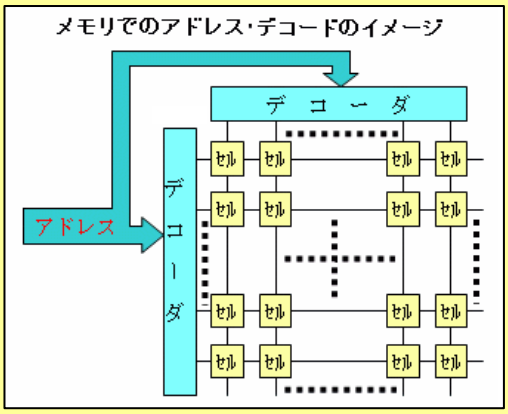


そこで、選択信号をエンコードして配線数を減らす必要があります。選択信号をエンコードすると、アドレス情報となります。このアドレス情報は選択信号と異なり、1 つの入力に対して出力してやればいいのでなく、複数の入力に対して出力する必要があります。そのためバス構成となり、アドレス・バスとなります。

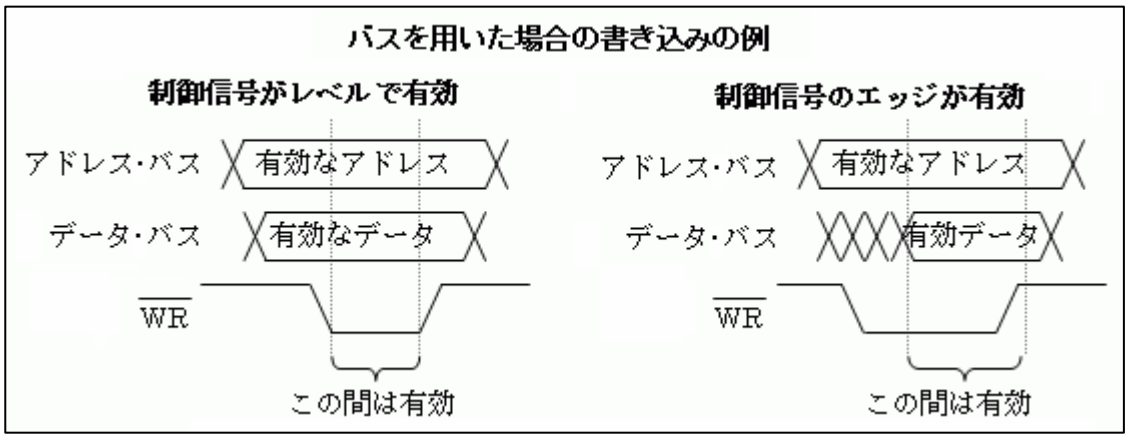


このようにアドレス・バスを用いることにより、少ない本数の信号線で必要な指定ができますが、最終的には個々の記憶素子の選択信号が必要になります。このため、アドレス情報をデコードする手間が追加されます。デコードについては、FAQ のデコーダもご参照ください。

【ちょっとおまけ】
メモリの中では個々の記憶素子(メモリ・セル)を選択するためにアドレス情報をデコードしますが、単純にデコードしているわけではありません。
アドレスをロウとカラムに分けて、マトリス状に配置したメモリを 2 本の選択線の組み合わせで選択しています。これにより、選択信号の本数を減らし、集積度の上を図っています。
また、選択線をデータの読み出し線と兼用させ、より配線を少なくできるようにもなっています。

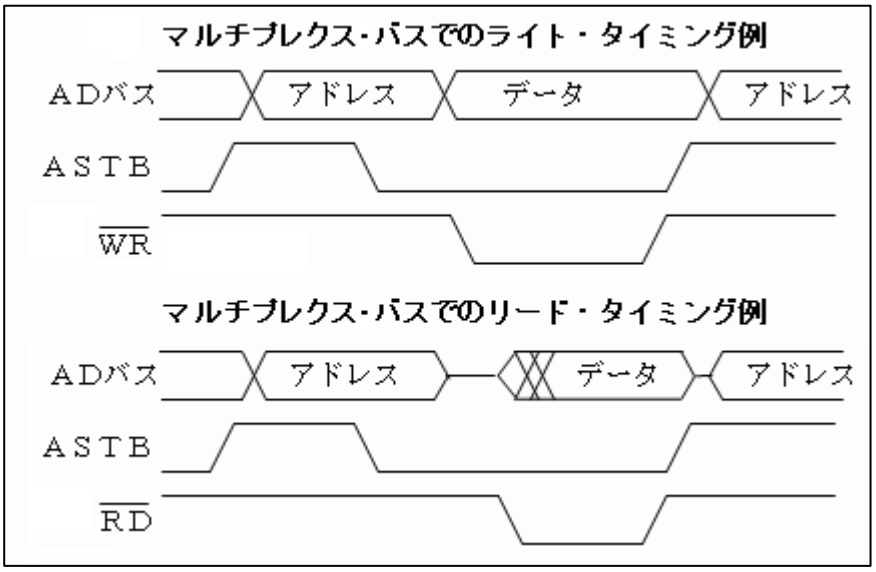


同様に、データについてもバス構成とし、しかも双方向での転送を行えば、必要な信号の本数を少なくできます。なお、データ・バスの場合には、通常はデコードを行う必要はありませんが、いつデータが有効であるかを明確にする必要があります。このため、アドレス・バスとデータ・バス以外に、制御用の信号が別途必要になります。バスを用いて書き込みを行う場合の例を次図に示します。この例では、WR 信号がアクティブ（ロウ・レベル）のときに、バス上のデータとアドレスが有効であることを示します。



さらに、バスの信号線の数を減らすため、アドレスとデータを同じバスで兼用することもあります。この場合には、バスを時間で区切って、ある期間はアドレス・バスとして使用し、その後はデータ・バスとして使用します。この方式は、アドレスとデータをマルチプレクスすることからマルチプレクス・バスと呼ばれます。マルチプレクス・バスを使用する場合は、信号の受け側で正しいアドレス情報を取り出すために、専用の制御信号が必要になります。アドレスの制御信号としては、ASTB などと呼ばれる信号が使われ、この信号を用いることで、マルチプレクス・バスからアドレス情報を取り出します（アドレスを生成します）。マルチプレクス・バスからのアドレスの生成には、ラッチが使用されます。FAQ のラッチをご参照ください。

なお、マルチプレクス・バスでは、1つのバスを切り替えながら使用するため、切り替え部分で信号が衝突しないようなタイミングが重要になります。特にリードでは、データの転送方向が切り替わるので、より注意が必要です。このため、リード後にアイドル・サイクルと呼ばれる、バスをアクセスしないサイクルを入れるマイコンもあります。

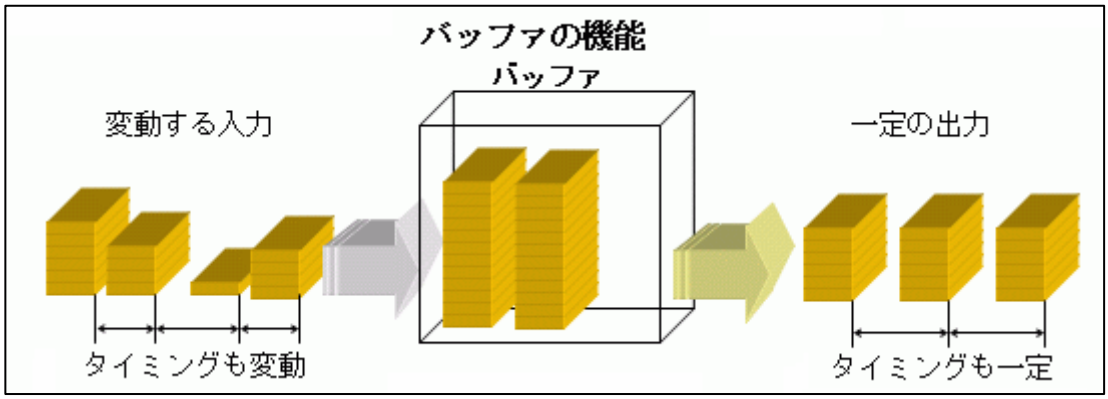


このようにバスは、信号線の本数、配線を少なくするのに有効で、今日のマイコンの発展を支えた重要な技術のひとつです。しかし、高速化ではボトルネックとなる面もあります。

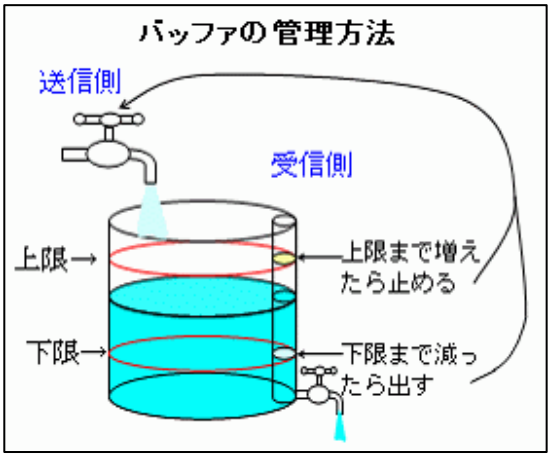
バッファ(Buffer)

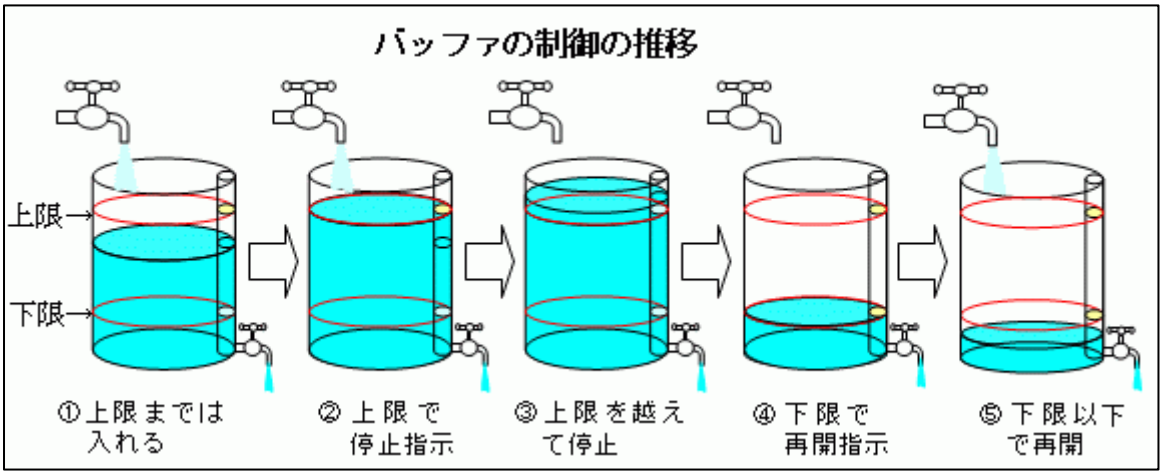
緩衝器のことですが、電気関係では、出力側の影響を入力側に与えないためのバッファ・アンプ（緩衝増幅器）や、やり取りするデータを一時的に溜めておくためのバッファ・メモリがあります。以下、バッファ・メモリを単にバッファと呼びます。

バッファは変動して入力されるデータを一定間隔や一定の量で取り出したり、逆に一定間隔や一定の量で入力されるデータを不定期に取り出したりする場合に使用されます。バッファを使用することで、転送速度が異なるものの間で、スムースなデータ転送を可能にします。



通常、バッファは FIFO 方式で、リング・バッファ構成として実現します。また、その大きさは、取り扱うデータの入出力の速度差とタイミングを元に決めます。なお、バッファがいっぱいになってから、新しいデータの入力を止めても遅すぎますし、バッファが空になってから新しいデータを要求すると、空の状態がしばらく続いてしまいます。このため、バッファは入っているデータの数に応じた細かな管理が必要です。通常は上限(たとえば 70～80%)までデータが溜まったら、新しいデータ入力をとめるような指示を出します。その後、データを取り出して、残りのデータが下限(たとえば 20～30%)以下になったら、新しいデータの入力を許可します。このように、上限／下限と制御の幅(ヒステリシス)をもたせることで、頻繁に禁止／許可を繰り返すのを防ぎます。

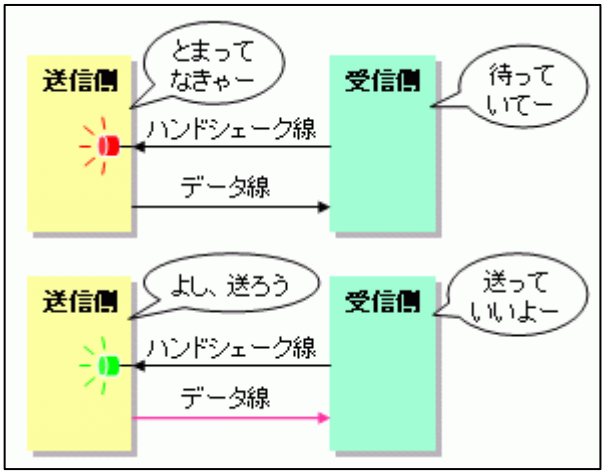




バッファ・オーバフロー(Buffer Over-flow)

データを溜めておくためのバッファが溢れることです。データを通信する場合に、処理速度の変動で発生するオーバランを避けるための手段として、バッファを利用しますが、その持ち方により、3 つの方法があります。ただし、いずれの方法も、基本的に受信側の処理能力が送信速度を上回っている必要があります。

ひとつはハンドシェークによる制御です。この方法では、受信側が送信側にデータを受け取れることを示す信号を送ることで送信を待ってもらうものです。この場合は、送信側にバッファが必要です。この方法では、受信側にバッファがないので、受信側から受信できないという通知がきたら、送信側はすぐ送信を止める必要があります。そのため、ハンドシェークのタイミングが重要です。



2 つ目は、受信側でバッファを準備するものです。送られてきた全データをバッファに格納しておき、読み出す準備ができてから読み出すようにするものです。この方式では、データを受信してから読み出すまでの時間的な余裕を大きくできます。この場合も、バッファに空がある間は受信できますが、いっぱいになると(バッファがオーバフロー)、それ以上の受信はできなくなります。



3 つ目は、両方にバッファを準備するものです。受信側では、バッファ管理を行い、ある程度データがたまると、送信側にデータの送信を控えるように通知します。送信側は、通知を受けると送信を停止します(停止すると、送信側のバッファにデータが溜まります)。ただし、すぐに止める必要はなく、タイミング的には緩やかです。受信側で、バッファからデータを読み出して、残りがある程度少なくなると、送信再開を通知します。→バッファ参照

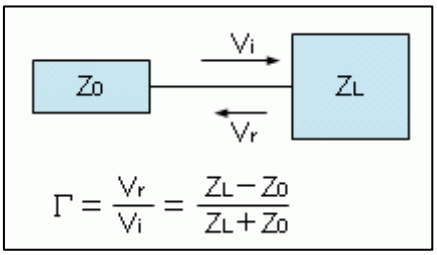
パリティ(Parity)

デジタル通信やコンピュータでの誤り検出(エラー検出)手法の一つです。nビットごとのデータに1ビット(パリティ・ビット)加え、その 1 ビットを"0"か"1"にすることによって、全ビットの"1"の数をあらかじめ決めておいた偶数(偶数パリティ)か奇数(奇数パリティ)にそろえます。デジタル通信では、これを送信して、受信側で、全ビットの"1"の数を数え、決めておいた偶数か奇数かを確認することによって、通信中のデータの誤りを検出します。たとえば、"1011001"の7ビットの送信データの"1"の数は 4 つなので、偶数パリティの場合は追加の 1 ビットを"0"にして、"10110010"にします。奇数パリティの場合は"1"を追加して、"10110011"にします。パリティによる誤り検出は、通信中の奇数個のビットの誤りは検出できますが、偶数個の誤りは検出できません。このため、多くても 1 ビットしか誤りが起きない通信手段や記憶手段に用いられたり、他の誤り検出手法と一緒に用いられます。

反射(Reflection)

日常生活では、光の反射が馴染み深いものですが、電気信号の伝送路でも、インピーダンスの不整合があると反射が起こります。このとき、反射波と入射波が合成されて、一定周期の定在波となります。なお、定在波の電圧の最大値と最小値の比は、**電圧定在波比 VSWR**(Voltage Standing Wave Ratio)と呼ばれます(**定在波比 SWR** と略されることもあります)。VSWR＝1 のとき、振動しないで伝搬効率が最大となります。

電圧反射係数 Γ (ガンマ)は、反射波の電圧と入射波の電圧の比で定義されます。



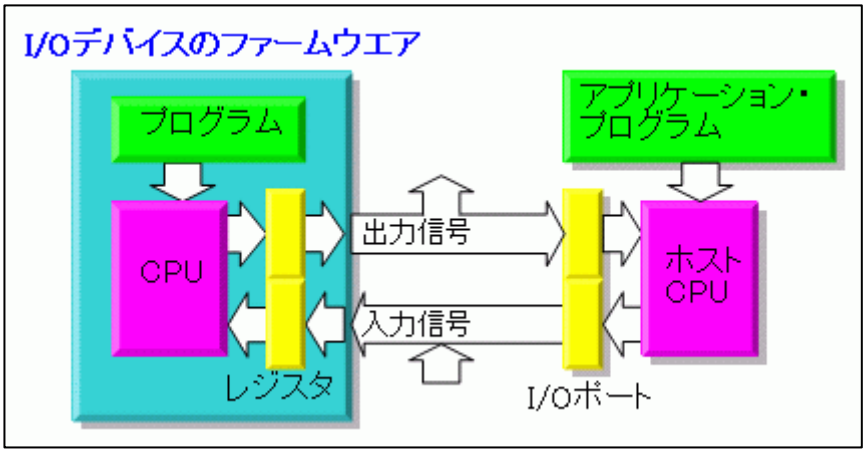
ここで、インピーダンス整合によって $Z_L=Z_0$ にすれば、Γ＝0(全透過)となり、電力がそのまま伝わります。また、$Z_L=\infty$(終端オープン)のとき Γ＝1(全反射)となり、電力はまったく伝わりません。

実際の回路にはリアクタンスがあるので、入射波を基準に実数とすると、透過波と反射波は位相の異なる複素信号になります。スミス・チャートでは、この位相 θ と反射量 ρ を表すことができます。

ファームウエア(Firmware)

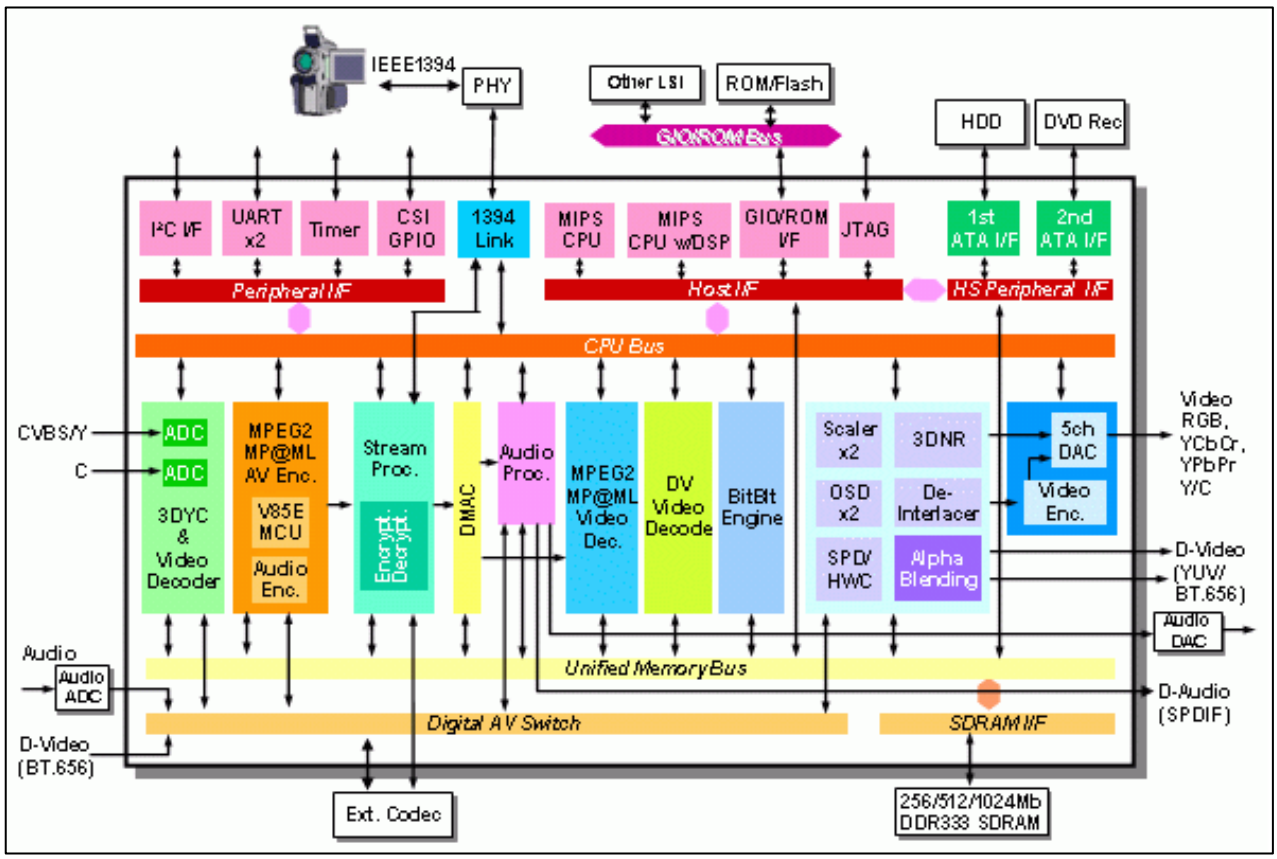
ソフトウエア、つまりプログラムの実行状態が、そのままハードウエアの機能(信号の状態)として表されるシステムのことです。順序制御などを行う半導体デバイスでは、内部にファームウエアが組み込まれていますが、ユーザはその存在を直接認識することはできません。

実際には、CPUを内蔵したデバイスで、内部のプログラム実行ごとにレジスタ内容を更新して、各ビットの状態を信号として出力したり、入力信号をこのレジスタのビット状態として取り込んだりするもので、ハードウエアとソフトウエアを結合したシステムです。ここで言うプログラムは、デバイス内部で動作を定義するもので、外部のホスト CPU が実行するユーザ・プログラム(アプリケーション・ソフト)のことではありません。



次図は MPEG2 AV コーデックの、μPD61181(EMMA2RL)の内部ブロック図です。

ここでは、V85E がコア CPU となって、ファームウエアのプログラムを実行します。



ファンアウト(Fan-out)

ロジック IC が、複数の同一特性の負荷を駆動する場合に、接続できる負荷数を表したものです。1対複数の接続が、扇(ファン)を広げたような形状なので、出力に対する特性としてファンアウトと呼びます。

通常、回路素子はそれぞれ異なった特性のため、今日ではあまりファンアウトの考え方を適用しませんが、ゲート・アレイなどで用いられています。

ドライブ能力を制限する要因には、電流(ハイ・レベル出力時、ロウ・レベル出力時)と容量があり、これらのうちの最小値でファンアウトが決定されます。出力電流がドライブ能力を越えると、ハイ・レベルやロウ・レベルの出力電圧が維持できなくなったり、破壊を引き起こしたりします。また、負荷容量がドライブ能力を越えると、信号の立ち上がり、立ち下がりが遅くなり、AC 特性を満足できなくなったり、ノイズの影響を受けやすくなったりします。

以下に、MOS デバイス同士の接続の場合について、説明します。

(1) ハイ・レベル出力の電流ファンアウト

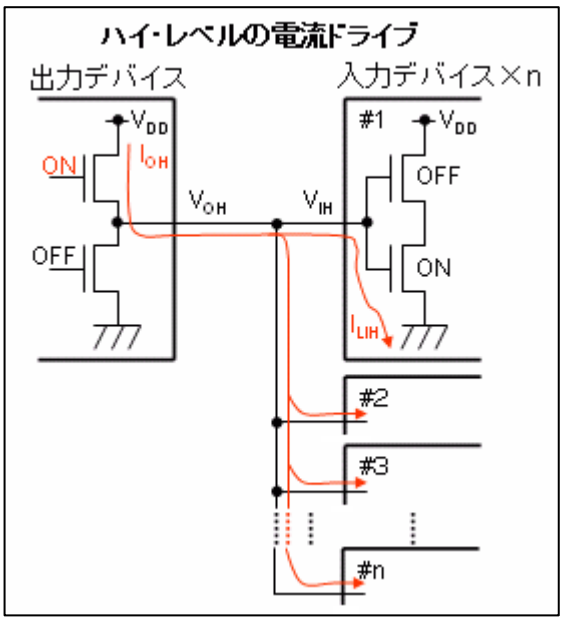
出力のドライブ電流 I_{OH} と、入力デバイスのリーク電流 I_{LIH} で算出します。

ファンアウト＝I_{OH}／I_{LIH}

たとえば、I_{OH}＝－100 μA、I_{LIH}(入力デバイス)＝5 μA のとき、ファンアウトは次のとおりです。

n＝100/5＝20

つまり、ハイ・レベルについては、20 個のデバイスをドライブできます。



(2) ロウ・レベル出力の電流ファンアウト

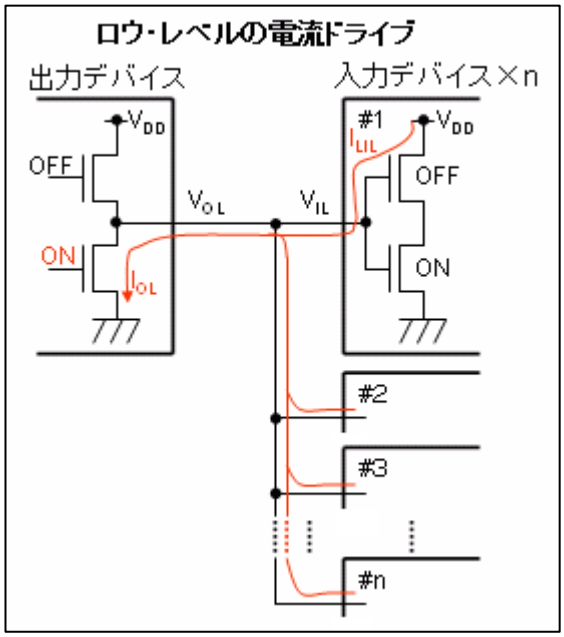
出力のドライブ電流 I_{OL} と、入力デバイスのリーク電流 I_{LIL} で算出します。

ファンアウト＝I_{OL}／I_{LIL}

たとえば、I_{OL}＝1mA、I_{LIL}(入力デバイス)＝5μA のとき、ファンアウトは次のとおりです。

n＝1000/5＝200

つまり、ロウ・レベルについては、200 個のデバイスをドライブできます。



(3) 容量のファンアウト

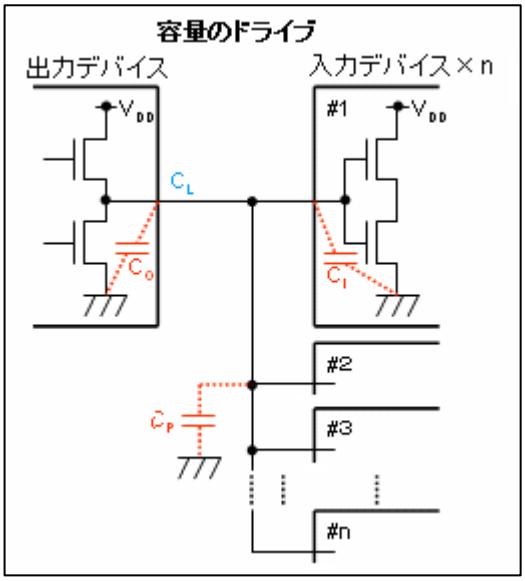
負荷条件 C_L と出力容量 $C_O(C_{IO})$、配線容量 C_P、入力デバイスの入力容量 $C_I(C_{IO})$で算出します。

ファンアウト＝(C_L－C_O－C_P)/C_I

たとえば、C_L＝100pF、C_O＝10pF、C_P＝50pF、C_I(入力デバイス)＝10pF のとき、ファンアウトは次のとおりです。

n＝(100－10－50)/10＝4

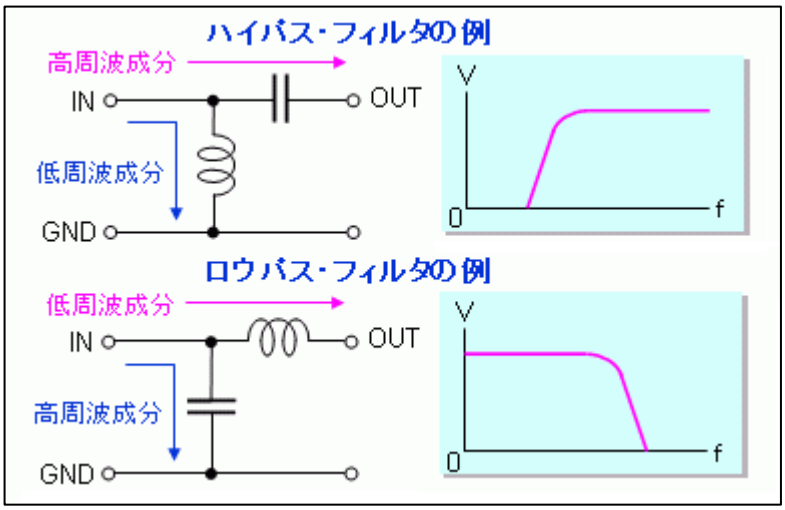
つまり、容量については、4 個のデバイスをドライブできます。



このように、MOS デバイスの場合には、ファンアウトに対して容量が支配的となります。

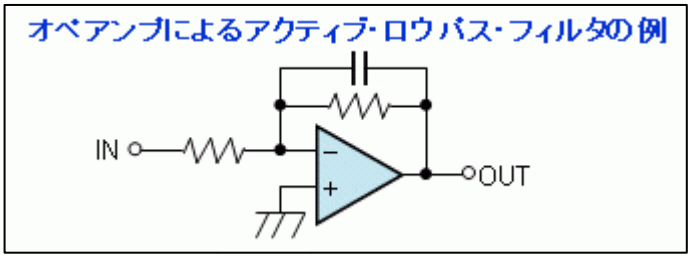
フィルタ(Filter)

液体や気体、光などに対するフィルタと同様、電気では特定の周波数成分だけ透過させる回路がフィルタです。通常は、コンデンサやコイルのインピーダンスの周波数依存性を利用します。コンデンサは高周波を通しやすく、低周波を遮断するので、ハイパス・フィルタとなります。また、コイルは逆に低周波を通しやすく、高周波を遮断するので、ロウパス・フィルタとなります。さらにこれらを組み合わせると、特定の周波数帯域だけ透過させるバンドパス・フィルタとなります。



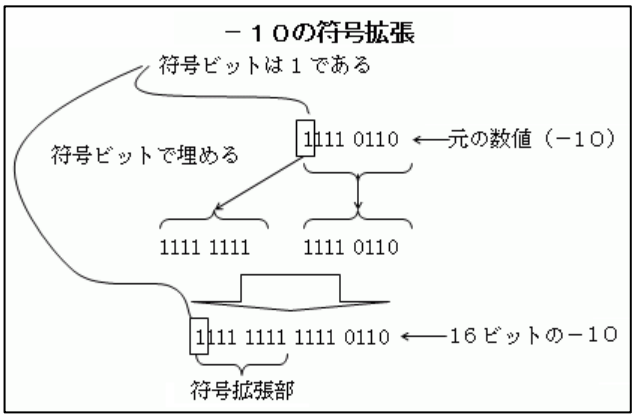
ただし、大きなコイルの使用は実用的でないため、抵抗を代用することが一般的です。

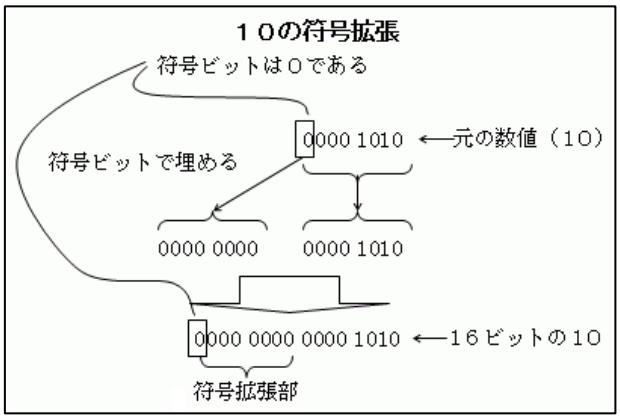
このように、コンデンサやコイル、抵抗だけで構成するフィルタをパッシブ・フィルタと呼び、オペアンプやトランジスタなどの能動素子も組み合わせたものをアクティブ・フィルタと呼びます。



符号拡張(Sign Extension)

負の数を 2 の補数で表現する場合に、異なる桁数の値の加減算を行うときに必要になる前処理です。
大きい方の桁数に合わせて、小さい桁数の方に新しい桁(符号拡張部)を追加しますが、そのとき、元の符号の値で符号拡張部を埋めます。例として、8 ビットの値(−10 と 10)を 16 ビットに拡張する場合を以下に示します。



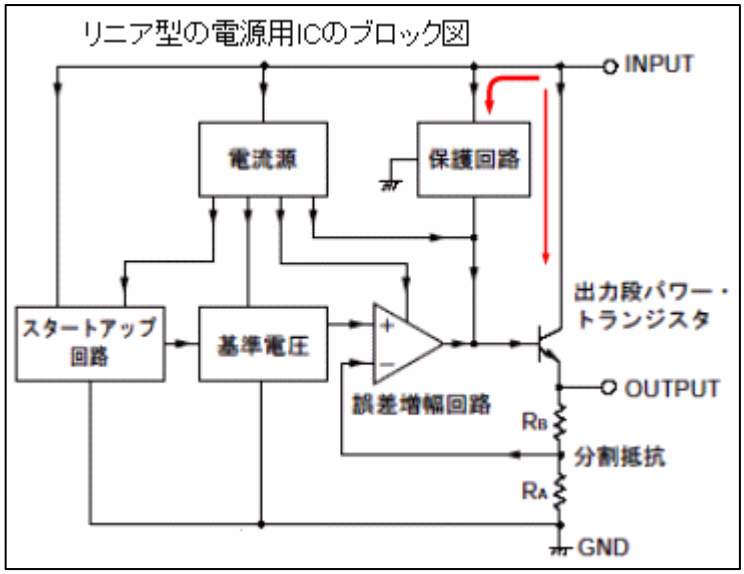


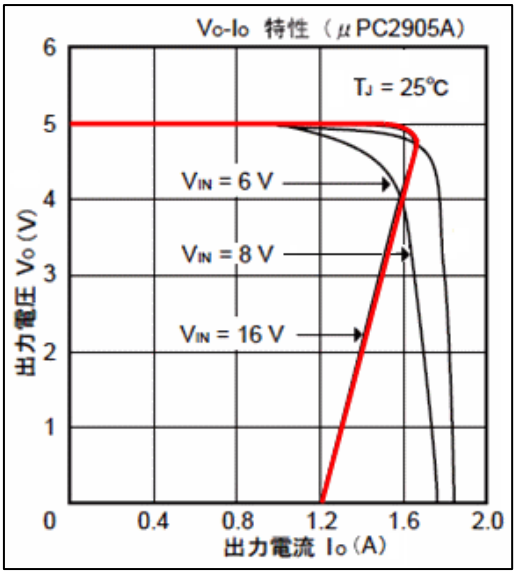
不定状態(Undefined State)

デジタル IC にて、信号出力レベルやレジスタ内容が特定できないことを指します(電源投入時の初期状態参照)。
出力については、ハイかロウのいずれかで、中間電位のことではありません。また、レジスタ内容は 1 か 0 のいずれかです。

フの字特性(Fold-back Type Drooping Characteristic)

リニア型の電源で、入力過電圧による安全動作領域越えをしないように、保護回路が働いている出力特性です。
「出力電圧 vs.出力電流」特性が、カタカナの「フ」の字に似ているので、このように呼びます。
このタイプの電源では、入出力間電圧(出力段パワー・トランジスタのコレクタ−エミッタ間電圧)が大きくなり、出力段パワー・トランジスタの安全動作領域を越えようとしたとき、保護回路側に電流が流れて出力電流を絞るため、このような特性になります。





プラグ＆プレイ(Plug & Play)

つないですぐ遊べるという発想で、接続した機器を認識して、自動的に ID とリソースを割り当てることをいいます。テレビに接続するゲーム機ではじまり、今では、パソコンに接続するだけですぐ使用できる周辺機器などに発展しています。

プラグ＆プレイには、ケーブルの挿抜時に電源を切るコールド・プラグインと、電源を入れたままでいいホット・プラグインがあります。

プリチャージ(Pre-charge)

DRAM がアクセス可能状態となるために、内部コンデンサに電荷をチャージしておくことです。DRAM では、メモリ・セルからデータを読み出すと、その電荷が失われてしまうため、読み出したデータを一時的に記憶回路に保持して、これに基づいて、電荷を失ったメモリ・セルにチャージすることが必要となるわけです。

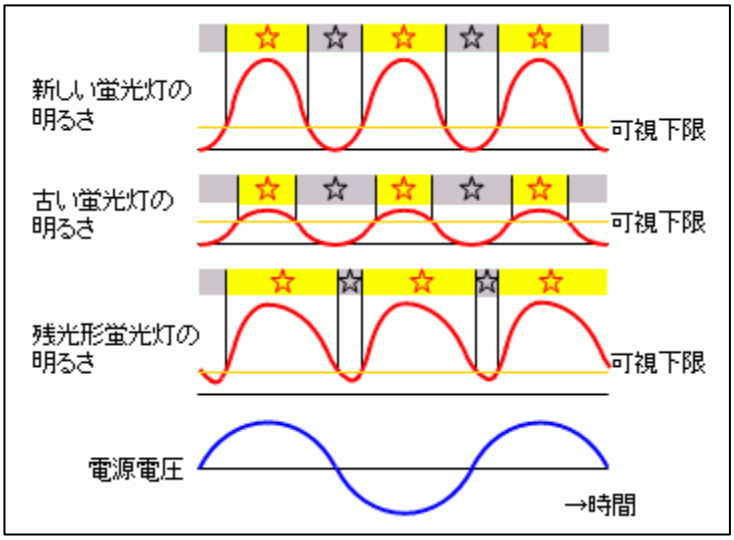
このため、次のアクセスが可能になるまでに時間を要し、DRAM ではリカバリ時間が長くなります。

フリッカ(Flicker)

「ゆらぎ」とか「明滅」という意味で、ダイナミック動作での「ちらつき」を表す用語です。

おもに光学系で用いられ、眼精疲労の原因となります。たとえば、蛍光灯のちらつきをフリッカ現象と呼びます。これは商用送電の周波数が 50/60Hz のため、1 秒間に倍の 100/120 回だけ、光る／光らないを繰り返すため生じる現象です。周波数が低いほど、消える時間帯を認識しやすくなります。

蛍光灯の寿命が近くなり、暗くなると、光る時間帯が短くなって顕著に現れます。逆に残光時間の長い蛍光体を使用したものでは、フリッカが軽減されます。このことは、熱放射で発光する白熱電灯でも同様です。



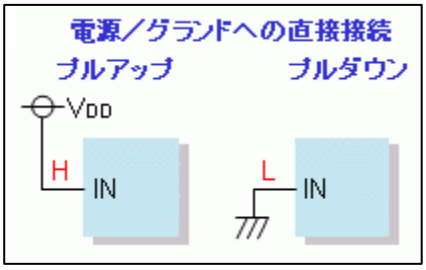
ダイナミック動作の周波数が高くて、通常は認識できなくてもフリッカが発生するのは、動画撮影をした場合で、フリッカ・ノイズと呼ばれます。テレビやパソコンの画面に黒い横線が縦に流れたり、電車や駅の LED 表示パネルがちらついたりするのは、いずれもフリッカ・ノイズで、表示タイミングとカメラのシャッタ・スピード(コマ数)との非同期が原因です。

プルアップ／プルダウン(Pull-up/Pull-down)

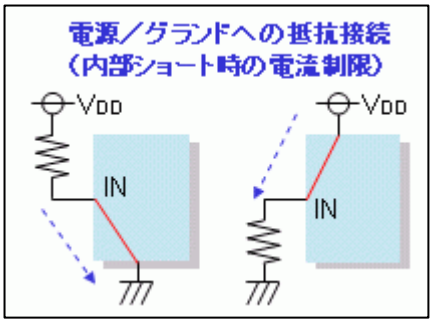
デジタル信号線の処理で、電源側に接続することがプルアップ、グランド側に接続することがプルダウンです。プルアップ／プルダウンには、電源やグランドに直接接続する場合と、抵抗を介して接続する場合があります。

(1) 直接接続

直接接続するのは、入力専用端子の入力レベルをハイかロウに固定する場合です。抵抗を介すると、部品が増えるうえ、信号ノイズを拾う恐れがあるので、一般には直接接続をします。

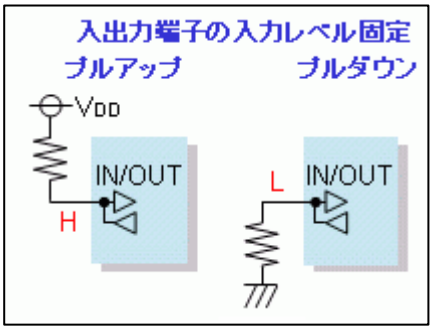


ただし高信頼性システムなどでは、内部劣化で入力が電源やグランドとショート状態になることを想定すると、システム全体のダウンを避けるためには、抵抗を介しておいた方がいいと言えます。したがって、入力固定の場合の抵抗の有無は、設計品質のターゲットをどこまでにするかで判断すべきです。

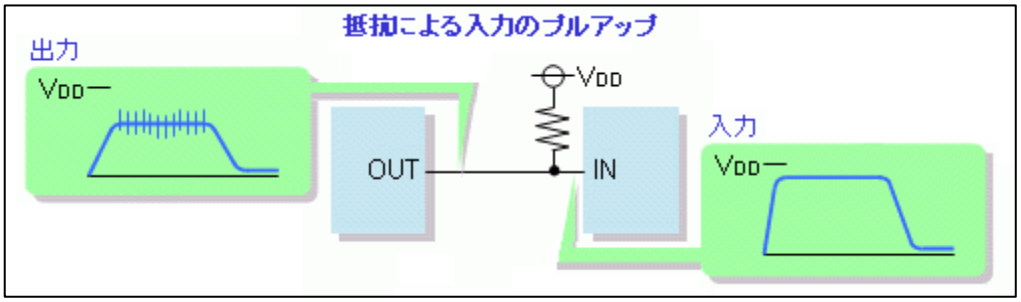


(2) 抵抗接続

一般に抵抗を介するのは、2 つのケースがあります。ひとつは、入出力兼用端子で、出力になる可能性がある入力レベルを前述のようにハイかロウに固定する場合です。この場合、電源やグランドに直接接続すると、出力時に電源／グランドとショート状態になるため、制限抵抗を挿入します。プログラムで意図的に出力に設定しなくても、ノイズによって出力になってしまうことを想定すると、抵抗を挿入することが望ましいといえます。

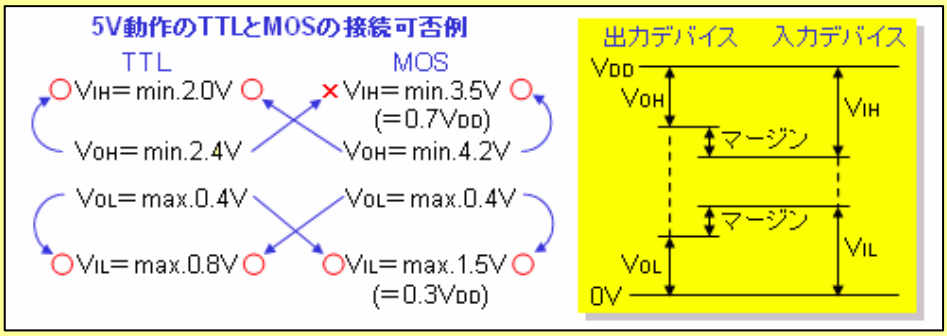


もうひとつは、入力レベルが規定に満たない場合や、信号線のインピーダンスを低減して信号ノイズや反射波を吸収したり、立ち上がり／立ち下がりを俊敏にする目的で、プルアップ／プルダウン抵抗を設置するケースです。特に装置間をケーブルで接続する規格では、ターミネータ(終端抵抗)が定義されていることがあります。

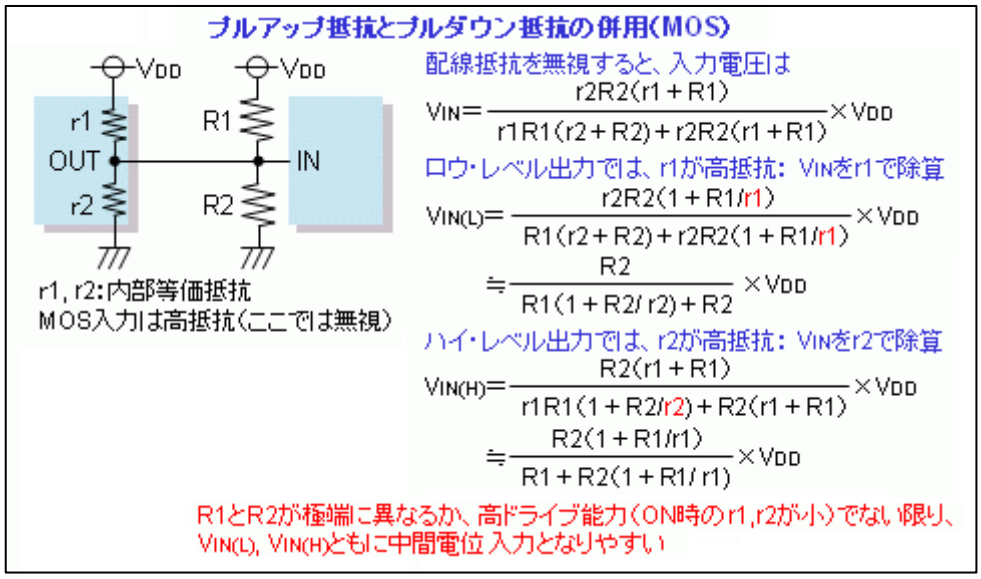


【ティー・ブレーク】

MOS 同士の接続では、電圧レベルがマッチングしないことはあまりありませんが、MOS が登場したころは、TTL と MOS を接続することがありました。このとき、TTL 出力と MOS 入力の接続では、ハイ・レベルを満足することができず、プルアップで対処することがありました。



なお、プルアップ抵抗とプルダウン抵抗の両方を設置すると、抵抗分割で中間電位の入力になりやすいので、一般的な MOS デバイスの入力では、必要に応じてプルアップ抵抗だけを設置します。



ハイ・レベルとロウ・レベルをともに改善するには、バッファの挿入が効果的ですが、遅延が許容されるかどうかを確認する必要があります。

(3) 抵抗値

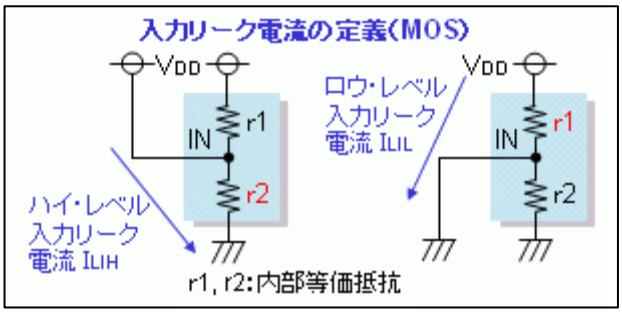
では、プルアップの抵抗値は、どのように設定すればいいでしょうか。

インタフェース規格で定義されている場合は、それに従います。

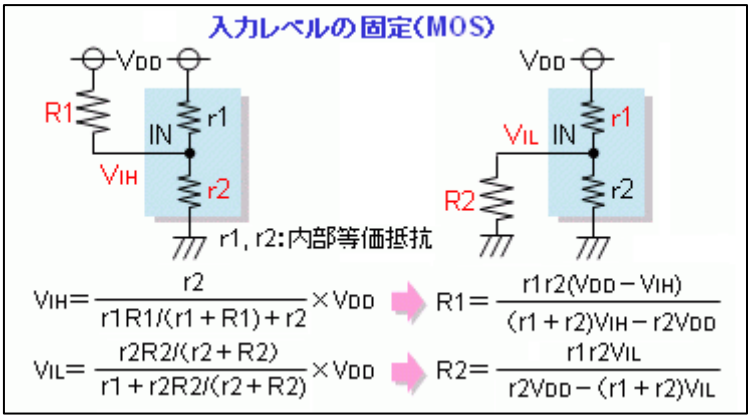
一般には数ｋΩ～数十ｋΩ程度の抵抗が使用され、あまり高抵抗では存在しないに等しいですし、かえってノイズを拾う要因となってしまいます。

(a) 入力の固定

MOS の入力レベルを固定する場合は、負荷電流が入力リーク電流となります。



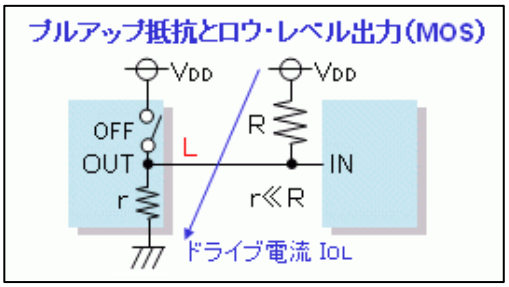
たとえば、V_{DD}＝5V で I_{LIH}＝＋5μA、I_{LIL}＝－5μA の場合、**r1＝r2＝5/(5×10^{-6})＝1MΩ** となります。



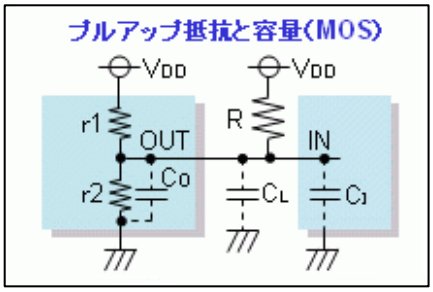
V_{IH}＝0.7V_{DD}, V_{IL}＝0.3V_{DD} だとすると、V_{DD}＝5V で r1＝r2＝1MΩ の場合には、上図の式より、**R1＝R2＝750kΩ** となります。つまり、これらの値を越えると、プルアップ側やプルダウン側の電位差が大きくなって、ハイ・レベルやロウ・レベルを確保できず、中間電位となってしまいます。

(b) 入力信号の処理

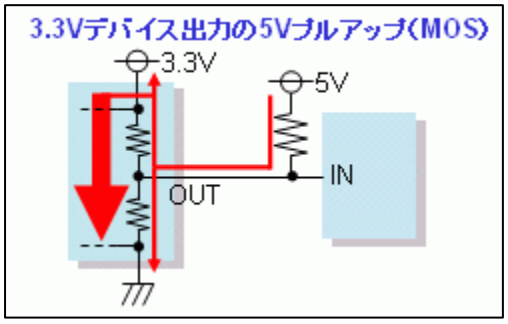
MOS デバイスの入力信号では、プルダウンではなく、プルアップが一般的なので、プルアップの説明をします。信号の補正をする場合、反転出力(プルアップではロウ・レベル、プルダウンではハイ・レベル)で抵抗に電流が流れるので、消費電力の低減には、入力特性を満足する範囲で、ある程度の高抵抗が望まれます。



ただし、出力容量と入力容量、配線の浮遊容量があるので、抵抗が大きいと RC の時定数で波形がなまることも考慮する必要があります。

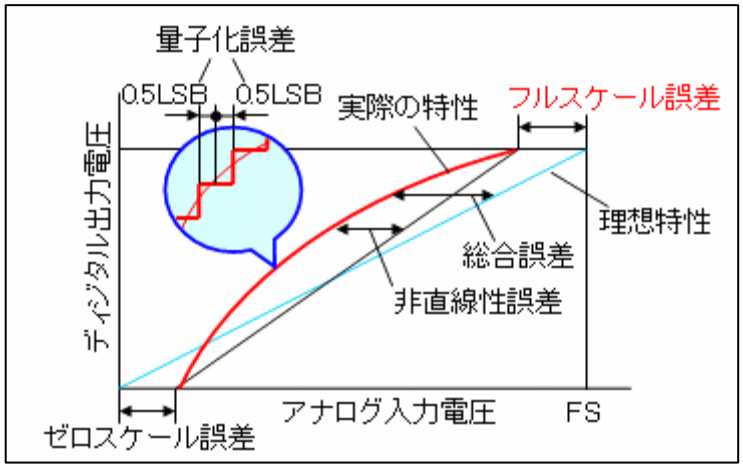


次に、信号の補正をする場合の下限値は、おもに出力デバイスのドライブ能力で決まります。
デジタル出力は、ハイ・レベル V_{OH} の最小値とロウ・レベル V_{OL} の最大値が、それぞれドライブ電流 I_{OH}、I_{OL} を条件として定義されます。
低抵抗では、プルアップでロウ・レベルが上昇し、プルダウンでハイ・レベルが下降します。このため、中間電位出力になってしまったり、ドライブ能力を越えた電流で劣化を促進させてしまったりすることになります。
たとえば、5V 動作時に I_{OL}＝5mA で V_{OL}＝MAX.0.5V という出力を 5V 電源にプルアップする場合を考えましょう。
抵抗値の下限は、$(V_{DD}-V_{OL})/I_{OL}=(5-0.5)/0.005=900(\Omega)$ となります。
なお、プルアップ抵抗を内蔵しているデバイスをドライブする場合、外部にプルアップ抵抗を設置すると、抵抗の並列接続で抵抗値が小さくなるので、注意する必要があります。
また、動作電圧の異なるデバイスがあって、高い方の電源電圧にプルアップすると、低い動作電圧のデバイスの絶対最大定格を越えたり、電流が回り込んで、デバイスや電源装置の破壊の原因となります。したがって、低い方の電源電圧にプルアップする必要があります。



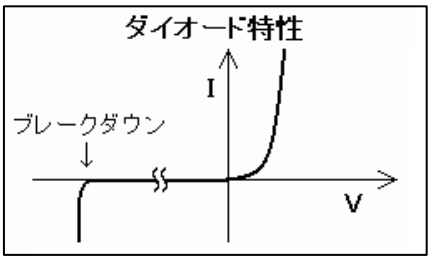
フルスケール誤差(Full-scale Error)

A/D コンバータで、変換値が最大になるときのアナログ入力と、フルスケール(FS)のアナログ入力(V_{REF})のオフセットを誤差(%FSR)として表したものです。フルスケール誤差とゼロスケール誤差、非直線性(線形)誤差を合わせたものが総合誤差で、デジタル変換で必然的に発生する量子化誤差は、これには含まれません。



ブレークダウン(Breakdown)

PN 接合のダイオードで逆方向に定格以上の電圧を印加していくと、あるところで電流が急増します。この現象を降伏あるいはブレークダウンと呼んでいます。この領域では、わずかの電圧の変化で電流が急増します。

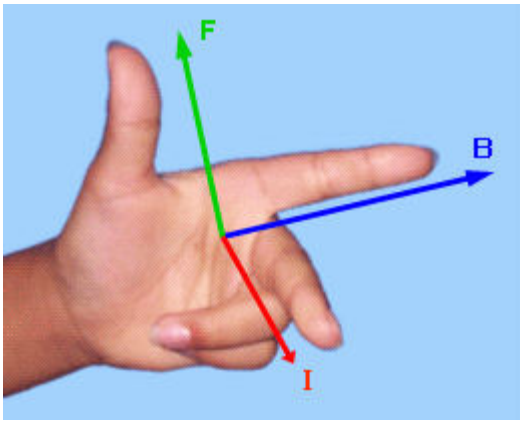


フレミングの法則(Fleming's Rule)

2 極真空管を発明したイギリスのジョン・アンブローズ・フレミングが考案した、電流と磁界、運動(力)の方向の定義で、「**左手の法則**」と「**右手の法則**」があります。

(1) フレミングの左手の法則

電子の電荷 q は、電界 E や磁界 B からローレンツ力($F=q(E+v\times B)$)という力を受けます(v は速度(velocity)、×はベクトル積)。この式から、電荷 q が運動する(電流になる)と、磁界 B からローレンツ力を受けることが分かります。「**左手の法則**」は、このローレンツ力の方向を示します。左手の親指と人差し指、中指を互いに直角に伸ばすと、親指から順に「**力(F)-磁界(B)-電流(I)**」の方向となります。「**FBI**」と覚えれば、分かりやすいですね。

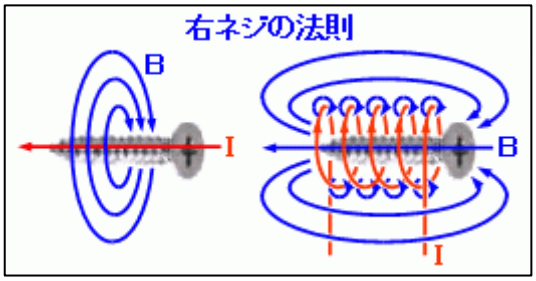


これは、磁界中で電流が受ける力の向きを示し、モータ(電動機)の回転原理に応用されています。
また、テレビなどのブラウン管にも応用され、水平方向と垂直方向の偏向コイルによる電磁石で強磁界を縦横に制御して、電子ビーム(電子の流れ)を走査線上の蛍光体に順次当てています。
MR センサ(磁気抵抗素子)では、パーマロイやニッケル・コバルト合金の強磁性体の薄膜を形成し、周囲の微弱な磁界の変化の影響で電流の流れが変わって、抵抗値が変化することを利用しており、非接触での回転数計測や位置検出(携帯電話、ノート PC の開閉検出など)、給湯器やガス・メータなどの流量検出などに応用されています。

では、携帯端末のような小型機器を磁石で別装置の金属筐体に固定する場合、電流が磁界の影響で流れが変わり、誤動作するでしょうか。この場合は、ブラウン管の偏向ヨークのように、N 極と S 極の間を電子が通過するのではなく、小型機器の外部の磁石からのもれ磁束が回路を横切るだけですから、影響はほとんどないと考えられます。たとえばブラウン管の例では、テレビの設置方向によって、地球の磁気の影響で、色が少しにじむことがあるくらいです。でも、強磁界を発生する機器をテレビの上に置くと、その周辺の画面の色相が変わってしまいます。強力な磁石の近くで微弱電流を扱うなど、影響がある場合、必要に応じて磁気シールドなどの対策をしてください。

これは、回路電流自体が発生する磁界についても言えます。導線に電流が流れると、その向きに右ネジが進むように回転するネジと同じ向き（右手の親指を電流の向きとしたときに、握った他の指の向き）に回転磁界が発生します。これは、電流の単位**アンペア**の語源になった、フランスのアンドレ・マリー・アンペールが発見した「**アンペールの法則（右ねじの法則）**」です。

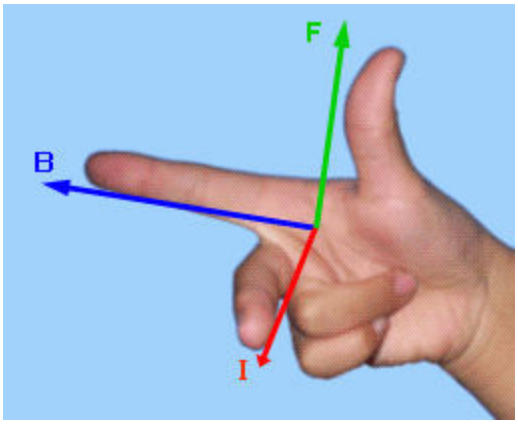
なお、コイルに電流が流れる場合は「右ねじの法則」による磁界がコイルの筒の向きに発生するので、コイルの中では、電流の向きを右ネジの回転の向きとしたときの、右ネジが進む方向に磁界が発生します。



電流から半径ｒだけ離れた点での磁場の強さは**H＝I/2πｒ**で表されますが、大電流でない限り、他の導体に流れる電流に影響を及ぼすほどの強さではありません。ただし、交流成分があって電流値が時間とともに変化する場合、これによって発生する磁束の変化で電磁誘導が起こり、他の回路に影響を与えます。

(2) フレミングの右手の法則

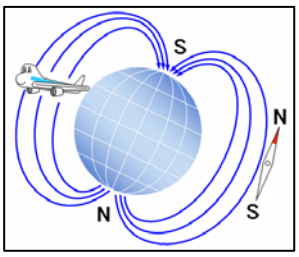
磁界中で導体が移動すると、導体内に起電力を生じて、電流が流れます。「**右手の法則**」は、この電流の方向を示します。「左手の法則」と同様に、右手の親指と人差し指、中指を互いに直角に伸ばすと、親指から順に「**力(F)−磁界(B)−電流(I)**」の方向となります。ここで言う「力」は、力の働きによる移動（運動）の方向を表します。



これは、磁界中で導体に流れる電流の向きを示し、ジェネレータ（発電機）の発電原理に応用されています。

「gather right and generate(右と発電をまとめる)」などと覚えれば、「左手の法則」と区別しやすいですね。

では、地球は磁気を帯びていますが、車や飛行機などの移動体に搭載している電気回路は、影響を受けて誤動作するでしょうか。地磁気は微弱ですし、たとえロケットでも電子の速度（抵抗がなければ、光と同じ 30 万 km/s）には遠く及びませんので、ほとんど影響はないと言えます。



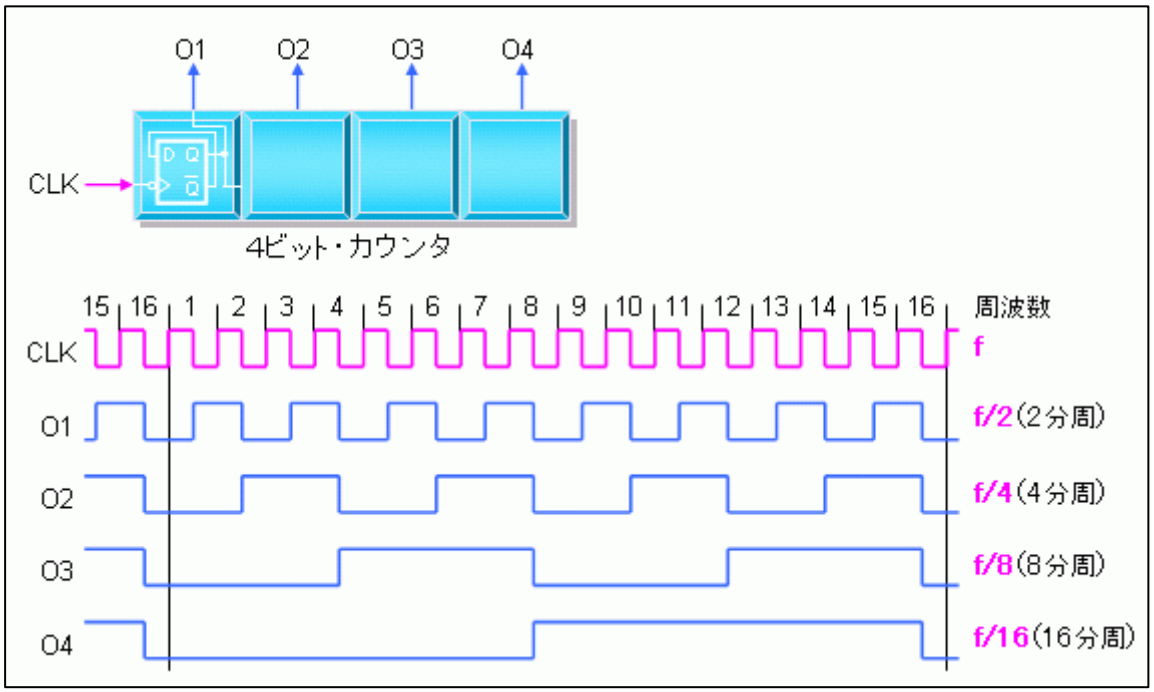
【ティー・ブレーク】

磁気ネックレスなどで、磁気の強さ（磁束密度 B）を「ガウス」で表しますが、現在使用されている SI 単位系では「テスラ:T」（＝10^4ガウス）となっています。「テスラ」を使うと、0 が 4 つも減って宣伝インパクトが弱くなるので、「ガウス」にこだわっているのでしょうか。なお、磁束 Φの単位はウェーバ（Wb）です（T＝Wb/m^2）。

分周（Divide）

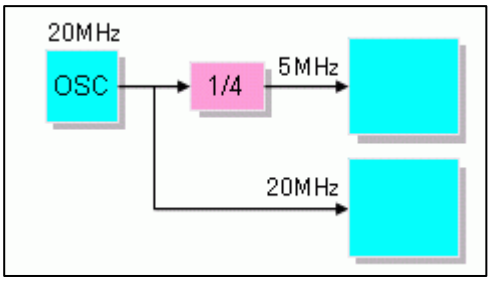
クロックなどのデジタル信号の周波数を 1/N に低減させて変換することで、その回路を分周回路または分周器(divider)と呼びます。また、この周波数変換を N 分周するといいます。

カウンタは、$1/2^n$（nはビット数）の分周器です。



分周器は、デバイスのシステム・クロックや通信の転送速度を生成するときなどに利用されます。

たとえば、同一基板に20MHz 動作のデバイスと5MHz 動作のデバイスがあって、外部発振器からシステム・クロックを供給する場合、20MHz のクロックを 4 分周すれば、発振器を共用できます

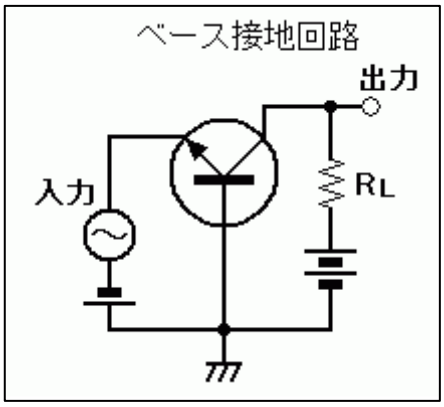


また、4.9152MHz のクロックを分周すれば、たとえば次のような、一般的なデータ伝送で使用するボー・レートが得られます。

分周比	ボー・レート
1/256 （8 ビット）	19200
1/512 （9 ビット）	9600
1/1024 （10 ビット）	4800
1/2048 （11 ビット）	2400
1/4096 （12 ビット）	1200

ベース接地回路(Base Common Circuit)

ベースは共通接地とし、入力をエミッタに加え、コレクタに負荷抵抗を付けて、その両端から出力を取り出します。入力抵抗が低く、帰還容量の影響を受けにくいことから、高周波電力増幅に使われましたが、最近は、高周波用にもエミッタ接地回路が多用され、ベース接地回路はあまり見かけません。



関連項目 エミッタ接地回路(Emitter Common Circuit)
コレクタ接地回路(Collector Common Circuit/Emitter Follower Circuit)

ベリファイ(Verify)

一連のデータをスキャンして、期待値と一致するかどうかを判定することです。
たとえば、メモリに書き込んだデータが正常かどうか確認するために、そのデータを読み出して、書き込みデータと比較することがベリファイです。また、その読み出し動作をベリファイ・リードと呼びます。

ポーリング(Polling)

デバイス内部のフラグや、外部からの入力の状態を定期的に確認することで、御用聞きのような作業です。スキャンでは状態を取り込むだけですが、ポーリングではその状態を判別する処理を含みます。
たとえば、入力がある状態になったときに、特定の処理をするようなシステムでは、ポーリングで定期的に入力を期待値と照合し、一致したときにその処理をさせることができます。

保護ダイオード(Protecting Diode)

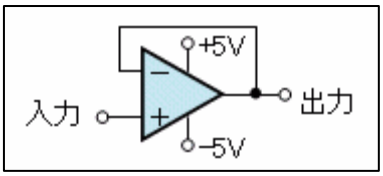
IC 内部の入力回路に、定格を越える電圧印加で、電源やグランドに電流をバイパスさせるよう設置しているダイオードです。
これにより、静電気やノイズの影響が軽減されます。
ただし通常、保護ダイオードの特性は規格保証していませんので、定格を越えても良いという訳ではありません。
必要に応じて、ノイズ・クリッピング・ダイオードを外部に設置してください。

ホット・プラグイン(Hot Plug-in)

たとえばUSB(Universal Serial Bus)やIEEE1394、パソコンの外部ハード・ディスク装置を接続するeSATA(External Serial ATA)のように、電源を入れたまま、インタフェース・ケーブルのコネクタの脱着ができる活線挿抜のことで、現在では機器交換のための差し替えを意味するホット・スワップ(Hot Swap)と同様の用語として用いられます。
これによって、動作中のシステムに周辺機器を追加接続できますし、ポート数の制限があっても、必要なときに周辺機器の交換ができるので、リソースへのアクセスの自由度が高くなります。
なお、ホット・プラグインに対応していない通常のインタフェース・ケーブルで活線挿抜をすると、発生するスパイク・ノイズが入出力バッファの絶対最大定格を越え、デバイス破壊を起こす可能性があります。

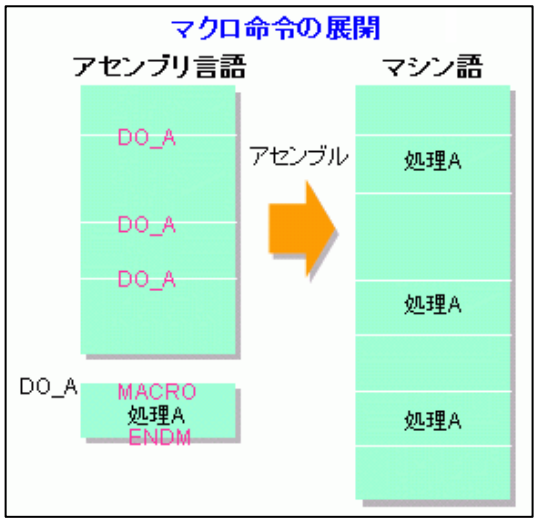
ボルテージ・フォロワ(Voltage Follower)

オペアンプを利用した非反転増幅器で、出力信号をマイナス側へ 100%帰還した回路で、全帰還とも呼ばれます。その名は、出力電圧が入力電圧に追従する(ゲイン＝1)ことに由来します。この回路では、ゲインはありませんが、入力インピーダンスが高くて出力インピーダンスが低いので、インピーダンス変換に利用されます。



マクロ命令(Macro Instruction)

マイコンのプログラミング・テクニックのひとつで、アセンブリ言語などで、使用頻度の高い処理プログラムなどに名前を定義して、この名前で記述を簡略化する手法です。マイコンが実行するマシン語（機械語）命令ではなく、それらをまとめた擬似的な命令を定義するので、マクロ（巨視的）命令と呼ばれます。
プログラムのアセンブル時にマクロ命令があると、アセンブラはこれを定義されたマシン語の命令コード列に展開します。まとまった処理を定義する点はサブルーチンに似ていますが、サブルーチンがマシン語レベルの呼び出しをするのに対し、マクロ命令は個々に命令コードに展開される点が異なります。そのため、サブルーチンの使用よりはプログラム・サイズが大きくなりますが、特定の命令列の記述が簡素化でき、処理プログラムの変更が1ヶ所で済むという利点は同じです。



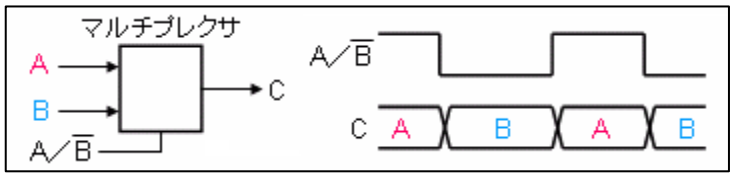
なお、マクロ命令の実行では、スタックのような退避領域が不要で、またその操作や分岐などがないので、サブルーチンの処理よりは実行時間を短縮できます。また、マクロ命令ではパラメータが定義できるので、パラメータを変更するだけで、類似処理プログラムを容易に定義できます。たとえば、マクロ命令を用いた 78K0S のシリアル入力のプログラムを次に説明します。ここで、P2.1 でシリアル入力ポート(SI)を、P2.2 でシリアル・クロック・ポート(SCK)を代用します。SCK を立ち上げてから読み込んだシリアル・データに応じて、A レジスタの対応するビットをセット(クリア)します。ループさせないで処理する記述を簡単にするため、マクロ sin1 でビットをパラメータ bitno として定義しておき、このマクロを 8 回展開し、実行させています。

```
;
;       シリアル入力処理（マクロ定義）
;
sin1    macro   bitno
        LOCAL   DATAis1
        MOV     P2,#11110011b   ; 6：SCK を立ち下げる
        SET1    P2.2            ; 6：SCK の立ち上げ
        BT      P2.1,$DATAis1   ;10：
        CLR1    A.bitno         ; 2：データが 0 ならデータを反転
DATAis1:
        endm
;
;       １文字受信
;   ＜機能＞
;   シリアルからデータを受信する。
;
;   ＜入力パラメータ（引数）＞　なし
;   ＜戻り値＞
;   A ： 受信データ
;
GETSPI:
        MOV     A,#11111111b    ; 初期値として 1 を設定
        sin1    7               ; MSB から順に受信処理
        sin1    6
        sin1    5
        sin1    4
        sin1    3
        sin1    2
        sin1    1
        sin1    0               ; 最後に LSB を受信処理
        RET
```

マルチプレクサ(Multiplexer)

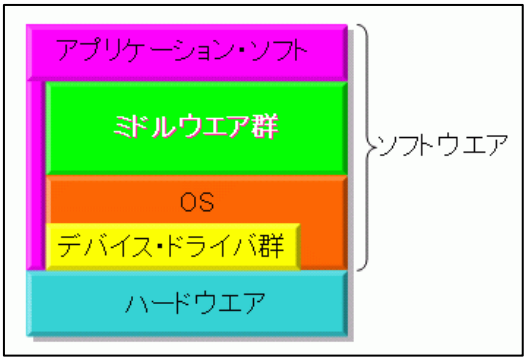
複数の信号を入力して、制御信号で切り替え出力をする回路です。

同時には使用しない信号をマルチプレクスすることによって、端子数を削減することができます。



ミドルウエア(Middleware)

基本ソフト(OS: Operating System)とユーザ・アプリケーション・ソフトの間に位置することから、このように呼ばれるソフトウエアで、ソフトウエアIP(Intellectual Property)とも呼ばれます。かつては、アプリケーションの一部としてユーザが設計していましたが、あまりにも規模が大きくなったため、分野ごとに共通部分を汎用パッケージとしたものです。ミドルウエアの採用によって、MPEGやGPSなどの分野技術を容易にアプリケーション・システムに取り入れることができます。

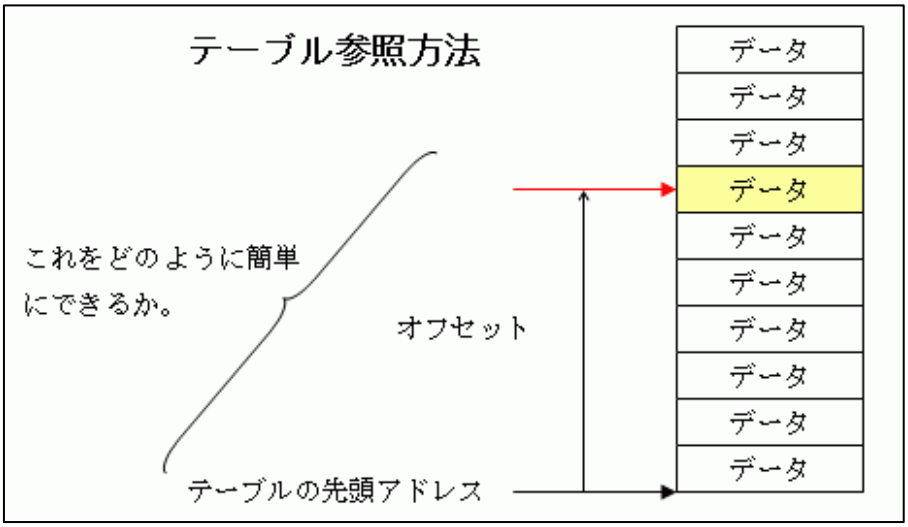


命令セットとアドレシング・モード(Instruction Set and Addressing Mode)

マイコンに処理を実行させるための命令が用意されています。一般に、命令数が多いほどできることも多いですが、できることは命令数だけでは議論できません。たとえば、RISCマイコンは比較的単純な命令しか実行できません。しかし、個々の命令を高速に実行させ、命令を組み合わせて同じことを実現できるようにしています。なお、命令の組み合わせはコンパイラが行うので、プログラムでは細かなことを意識する必要はありません。



命令数のほかに、各命令のアドレッシング・モードも重要です。アドレッシング・モードは、命令の対象を指定する方法を示します。たとえば、複数のデータで構成されたテーブルで、データを参照する場合を考えてみましょう。

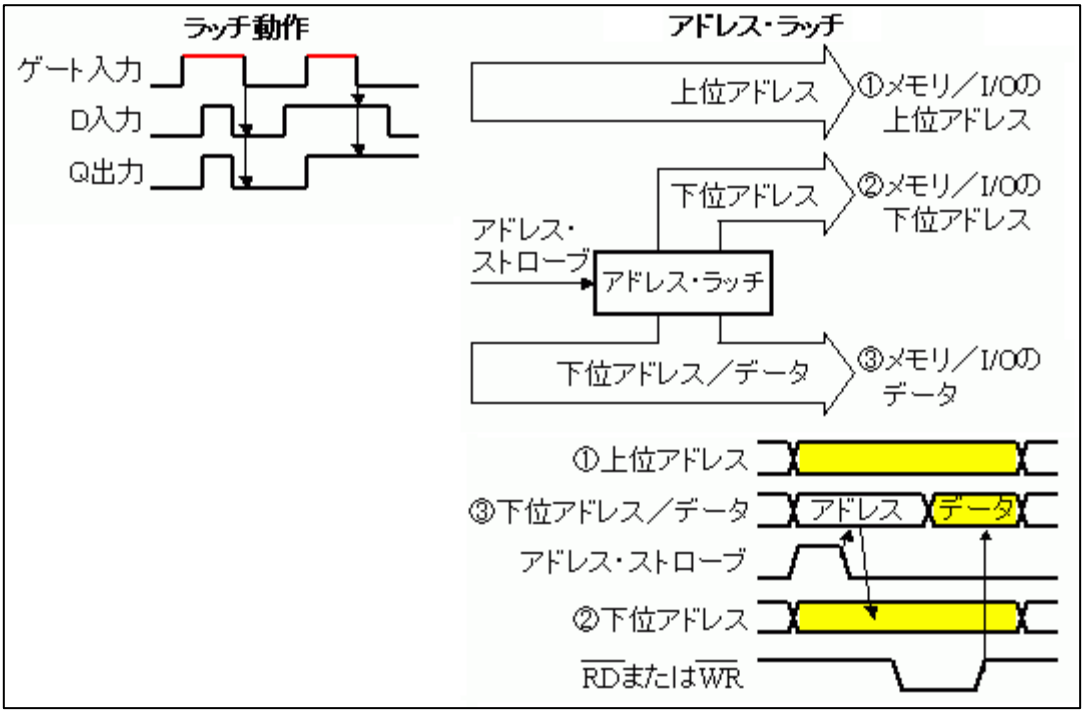


この場合、目的はデータの転送ですから、最終的に使用する命令は MOV 命令です。
①基本的なアドレッシングであるレジスタ・インダイレクト・アドレッシングでは、目的のデータを指すアドレスをテーブルの先頭アドレスと、そこからのオフセットで計算します。その結果をレジスタにセットし、レジスタを使って目的のデータを指し示します。つまり、MOV 命令以外にアドレス計算が必要になります。
②少し高度なアドレッシングであるベース・アドレッシングではレジスタにはテーブルの先頭アドレスをセットして、命令のオペランドでオフセットを指定します。このアドレッシング・モードは比較的よく使われますが、オフセットを自由に変更することはできません。
③オフセットを自由に変更できるアドレッシング・モードとして、ベースト・インデクスト・アドレシングがあります。このアドレッシングでは、オフセットもレジスタで指定します。
④さらに、複雑なアドレシングとして、②と③を組み合わせたようなアドレッシングもありました。

一時は、どれだけ高度な（複雑な）アドレッシング・モードを使えるかを競った時期もありましたが、ハードウエアが複雑になり、性能のネックとなるので最近はあまり複雑なアドレッシング・モードは少なくなっています。それでも、アドレッシング・モードは重要な要素であることには違いありません。
また、すべての命令で全アドレッシング・モードが使えるわけではありません。命令とアドレッシング・モードの組み合わせにも注意が必要です。

ラッチ(Latch)

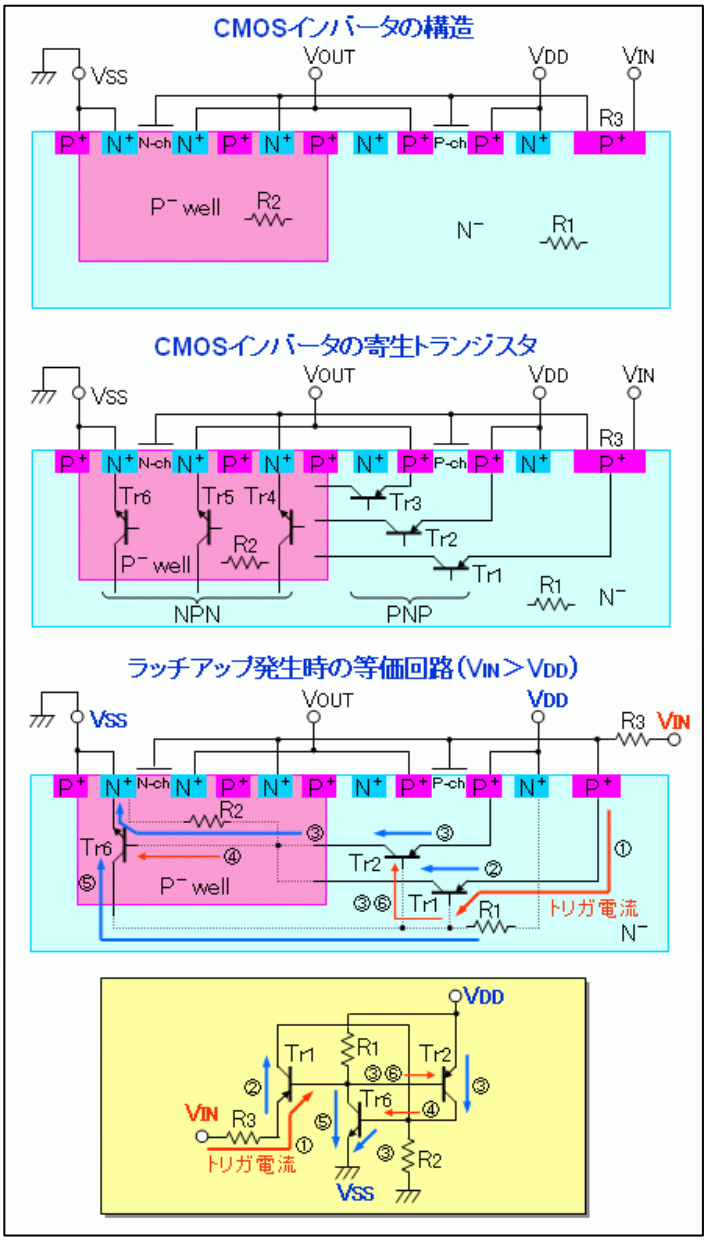
ゲート入力がアクティブな間は、入力データをそのまま出力して、ゲート入力がインアクティブになると、そのときの入力データを保持して出力を固定する回路です。
たとえば、アドレスとデータが時分割の兼用端子になっているマイコンでは、アドレスをデータから分離して保持（アドレス・ラッチ）する機能として利用されます。この場合、データ・アクセス（リード／ライト）のタイミングでは、すでに兼用端子のアドレスが出力されていないため、ラッチが必要となります。
この例では、アドレス・ストローブ信号がゲートに入力されます。



ラッチアップ(Latch-up)

CMOS デバイスに、絶対最大定格を越える(V_{DD} より高いか、GND/V_{SS} より低い)電圧を入力すると、ラッチアップが発生することがあります。これは、ノイズなどの影響で、CMOS デバイスの V_{DD}–V_{SS} 間が、構造的にショート状態となる現象です。

本来、CMOS 構造では、P–ch と N–ch にしか電流が流れません。ところが CMOS 構造は、寄生トランジスタによってサイリスタ構造(PNPN)と等価なので、高電圧を入力すると、電流がチップの下層に流入して寄生トランジスタを ON にし、電源からグランドへ突き抜けて、ショート状態となります。



① V_{IN} が V_{DD} より高くなる。

② Tr_1 が ON になる。

③ Tr_2 が ON になる。

④ R_2 に電流が流れて、両端が Tr_6 を ON にする電位差となる。

⑤ Tr_6 が ON になる。

⑥ R_1 に電流が流れて、両端の電位差で Tr_2 の ON を継続させる。

⑦ ③～⑥で V_{DD}–V_{SS} 間に電流が流れ続け、ラッチアップが発生する。

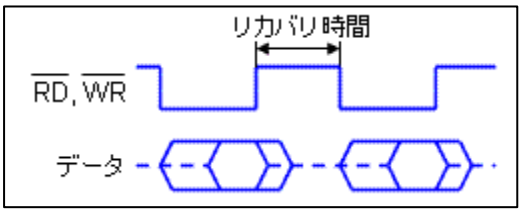
ラッチアップの原因としては、静電気やノイズ、入力端子への電源電圧範囲外の電圧印加(オーバシュートなどを含む)などがあります。ラッチアップが発生すると、これらの原因がなくなっても、上記のメカニズムで V_{DD}-V_{SS} 間に電流が流れ続けます。ラッチアップはショート状態ですから、劣化を促進させたり、破壊を引き起こしたりします。もちろん、絶対最大定格は、一瞬でも越えてはならない規格です。
なお、ラッチアップ発生時は、構造的にショートしているためリセットが利かず、電源を遮断・再投入しないと復帰しません。ただし、ラッチアップが発生すると、デバイスの信頼性を損ないますので、そのまま使用を続けることは好ましくありません。通常、ラッチアップ対策としては、ノイズ・クリッピング・ダイオードでノイズなどの吸収をします。

リード・モディファイ・ライト(Read Modify Write)

メモリ・セルやレジスタの内容を読み出し、データの変更後に書き込みをする一連の処理です。デバイスの動作モードを変更するため、モード設定レジスタの特定ビットの内容だけを変更して、再設定する場合などに行います。

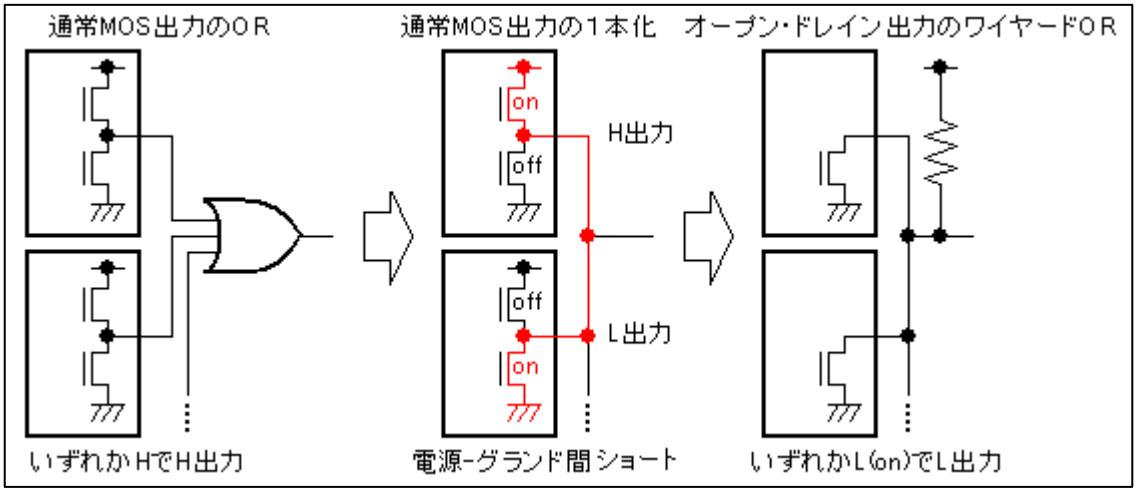
リカバリ時間(Recovery Time)

デバイスに対して、CPU がリードやライトのアクセスをしたあと、そのデバイスが次のアクセスへの準備が完了して、リードやライトが可能になるまでの時間のことです。
この時間以内にアクセスすると、正規のデータが読み出せなかったり、正常にデータ書き込みが行えません。



ワイヤード OR(Wired OR)

複数の信号のうち、いずれかの変化をとらえる場合、通常は OR ゲートを使用しますが、これを削減して1本の信号にまとめる手法です。現実的には、負論理の論理回路で有効です。
通常のバイポーラや MOS の出力を1本にまとめると、いずれかの反転でショート状態になってしまいます。
ところが、オープン・コレクタ(バイポーラ)やオープン・ドレーン(MOS)の出力なら、電源へのショート・パスがないため、プルアップ抵抗を1本つけることで、論理和がとれます(負論理)。
ここで抵抗値は、1信号だけアクティブになるワースト・ケースのロウ・レベル駆動を想定する必要があります。



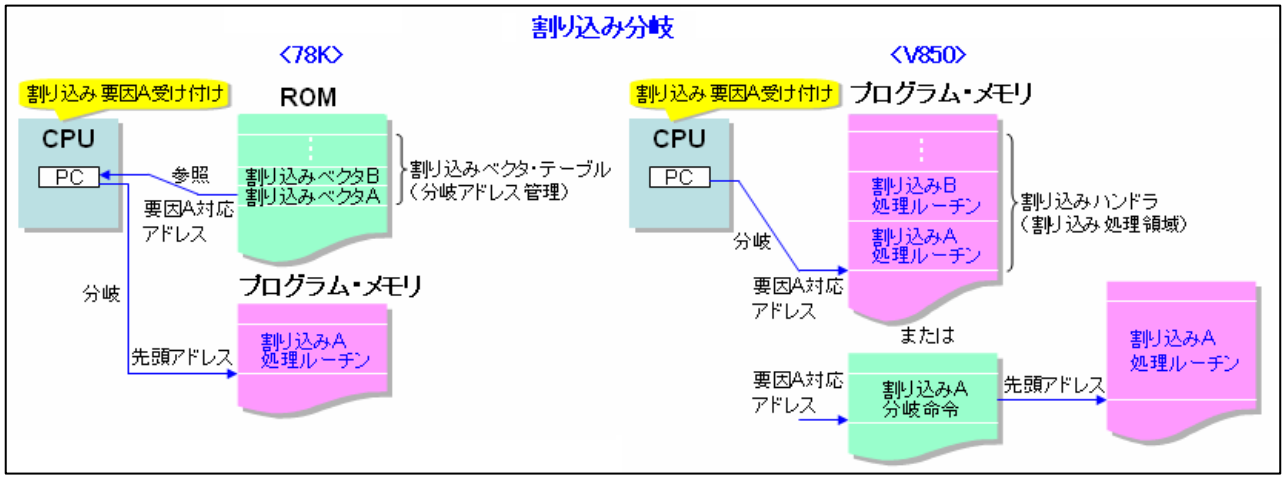
割り込み(Interrupt)

(1) 基本動作

割り込みとは、外部からの要求信号などによって、CPU がプログラム処理を中断して、他のプログラム処理をすることです。つまり、プログラム処理とは非同期に、別の処理を行う手法です。このとき、プログラム処理を中断して他のプログラムに分岐するアドレスは、割り込み要因ごとに定義されています。

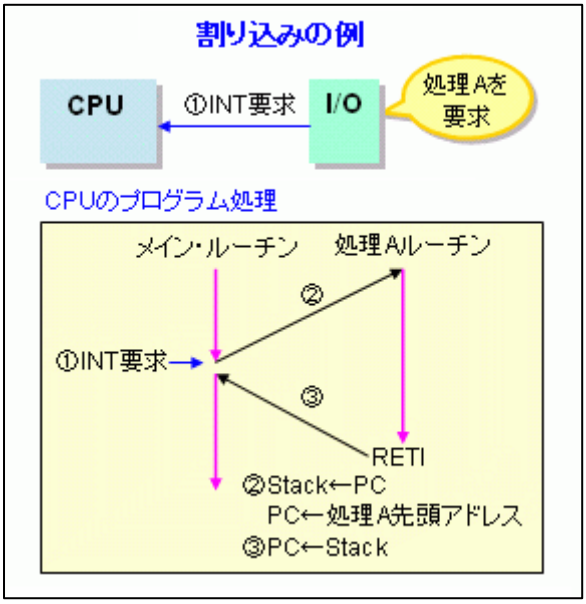
78K 系のマイコンでは、割り込み処理ルーチンの先頭アドレスをベクタと呼び、割り込み要因ごとのベクタをまとめて ROM 上でテーブルにしています。割り込みが受け付けられると、ベクタ・テーブルから要因に対応する情報を参照し、それをアドレスとして分岐します。割り込みベクタ・テーブルには、ユーザが要因ごとに用意している割り込み処理ルーチンの、先頭アドレスを書き込んでください。

V850 系のマイコンでは、割り込みの分岐先の領域(16 バイト)を割り込みハンドラと呼びます。割り込みが受け付けられると、要因に対応するハンドラの開始アドレスに分岐します。割り込み処理ルーチンがハンドラ領域に収まらない場合、処理ルーチンは別の領域におき、ハンドラにはその先頭アドレスへの分岐命令をおきます。



割り込み処理が終了すると、元の処理に戻ります。

78K 系のマイコンでは分岐するときに、中断したプログラムの実行情報(アドレス(プログラム・カウンタ(PC)値)と状態フラグ(PSW:Program Status Word)の値)が、RAM 上のスタック領域に自動的に退避させられます。割り込み処理ルーチンの最後で RETI(Return from Interrupt)命令を実行すると、退避させたプログラムの実行情報が PC と PSW に復帰するので、元の処理へ戻ることができます。



その他の汎用レジスタの内容は、割り込み時に自動的にはスタックなどに保存されません。必要に応じて、プログラムで退避(PUSH 命令)や復帰(POP 命令)の処理をしてください。ただし、78K 系のマイコン(78K0S を除く)では、汎用レジスタがレジスタ・バンク形式(複数組)なので、割り込み処理の先頭でレジスタ・バンクの指定命令を実行すれば、PUSH/POP は不要です。

なお、V850 系のマイコンの割り込みでは、退避用のレジスタに実行情報を自動的に退避させるだけです。そのため、多重割り込みをさせるには、退避用レジスタの内容を上書きで消失させないように、スタック領域に退避させる必要があります。

リセット直後は割り込み禁止(DI)状態なので、割り込みを許可するには EI(Enable Interrupt)命令を実行する必要があります。DI 状態のとき、78K は IE フラグ(PSW)＝0、V850 は ID フラグ(PSW)＝1 となっています。

なお、具体的なマイコン製品の割り込み機能については、78K シリーズと V850 シリーズの FAQ もご参照ください。

(2) 優先順位

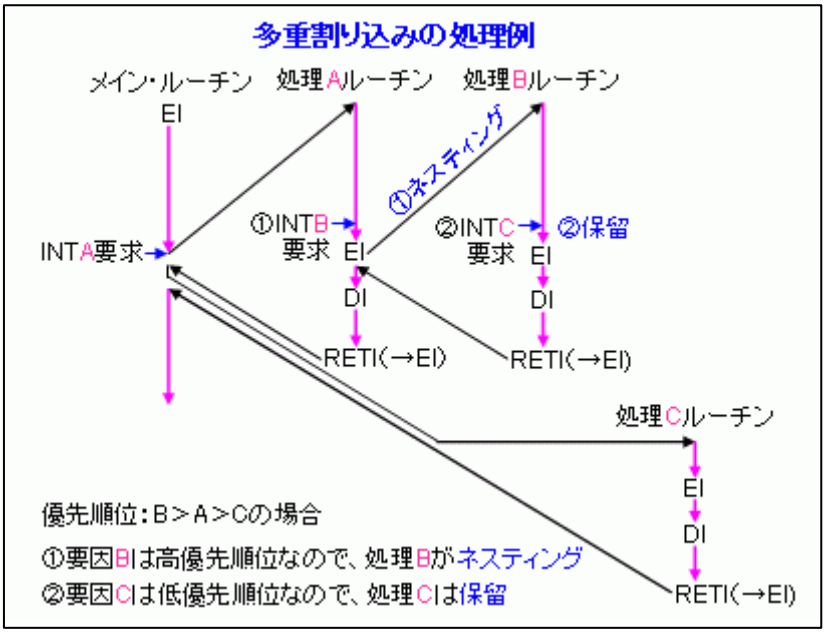
割り込み要因が複数ある場合、通常は優先順位(プライオリティ)があり、割り込みコントローラが処理すべき要因を決定します。低優先順位の割り込み要求は保留されます。
優先順位の最も高い要因に対する処理が終わると、次の優先順位(その時点では最高優先順位)の要因に対する処理をします。

(3) 割り込みのマスク

割り込みには、マスカブル割り込みとノンマスカブル割り込みがあり、マスカブル割り込みは割り込みを禁止することができます。割り込みの禁止には DI (Disable Interrupt)命令、許可には EI (Enable Interrupt)命令を使用します。また、レジスタ設定で割り込み要求をマスクして、割り込み要求を発生させないマイコンもあります。
ノンマスカブル割り込みは禁止できず、最高優先順位となっています。

(4) 多重割り込み

割り込み処理中に、より高優先順位の割り込み要求が発生した場合、その割り込み処理をしたい場合があります。このときは、さらに分岐して多重割り込みとなります。これをネスティング(「巣ごもり」の意味)と呼びます。
ただし一般に、割り込み要求を受け付けると DI(Disable Interrupt)状態となるため、ネスティングを許可するには EI(Enable Interrupt)命令を実行する必要があります。なお、EI 命令を実行したのち、ネスティングを許可するタイミングが経過すると、DI 命令を実行してネスティングを禁止させます。
また、RETI 命令で割り込み処理を完了すると、自動的に EI 状態に戻り、次の割り込み要求を受け付けられるようになります。

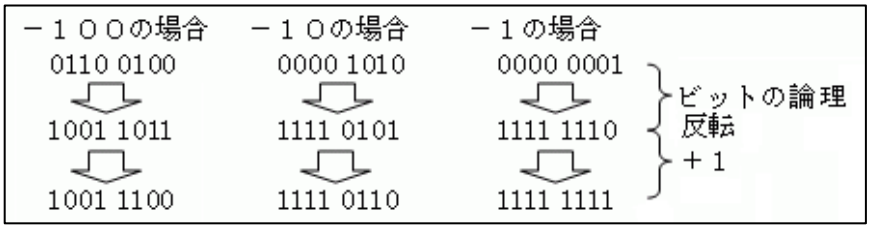


2 の補数(2's Complement)

負の数を表現するのによく用いられる表現が 2 の補数表現です。ここでは、簡単な例として 8 ビットで表現できる数値(－128～＋127)のを下表に示します。

2 進数	10 進数表現	2 進数	10 進数表現
00000000	0	10000000	－128
00000001	＋1	10000001	－127
00000010	＋2	10000010	－126
00000011	＋3	10000011	－125
⋮	⋮	⋮	⋮
01111100	＋124	11111100	－4
01111101	＋125	11111101	－3
01111110	＋126	11111110	－2
01111111	＋127	11111111	－1

この例でもわかるように、最上位ビット(MSB)が 0 なら正の数、1 なら負の数を表します。実際に 2 の補数を求めるには、0 から減算するのが簡単ですが、減算できない場合は、各ビットの論理を反転して＋1 しても求められます。以下に求め方の例として、－100、－10、－1 を示します。



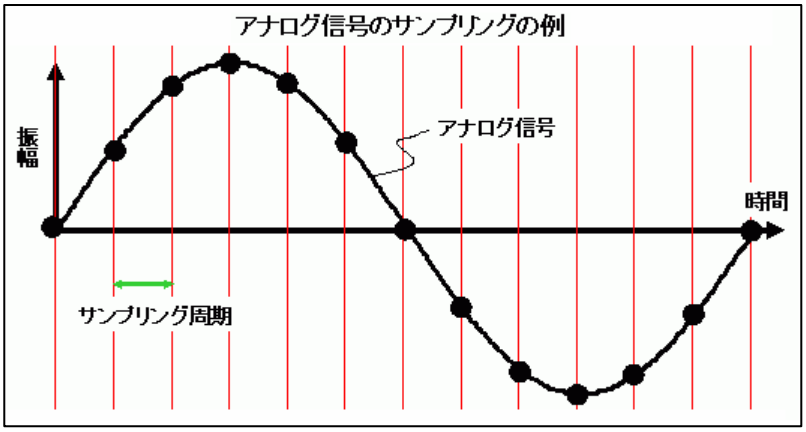
2 の補数表現は、結果の数値が溢れない範囲では、符号を気にすることなく加減算を行えます。特に、負の数を 2 の補数で表すと、加算だけで引き算も行うことができます。たとえば、4 から 5 を引くと、次のようになります。

$$4 - 5 = 4 + (-5) = 00000100 + 11111011 = 11111111 = -1$$

2 の補数で桁数が異なる数値の加減算を行う場合、先に符号拡張をする必要があります。

A/D のサンプリング(Sampling of A/D Converter)

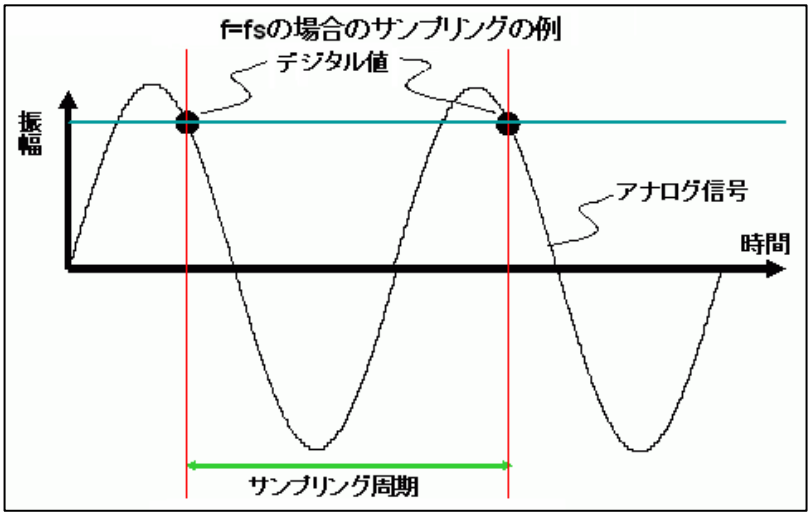
自然界のアナログ信号をマイコンなどのデジタルで処理する場合、その間に A/D(アナログ･トウ･デジタル)コンバータが必要で、最近のほとんどのマイコンに内蔵しています。A/D コンバータは、時間軸方向に一定間隔でアナログ信号を取り込み(サンプリング ： 標本化)、その一定時間をサンプリング周期と呼びます(逆数はサンプリング周波数)。なお、振幅軸方向で、あらかじめ決めておいた離散値に近似することを量子化といいます。



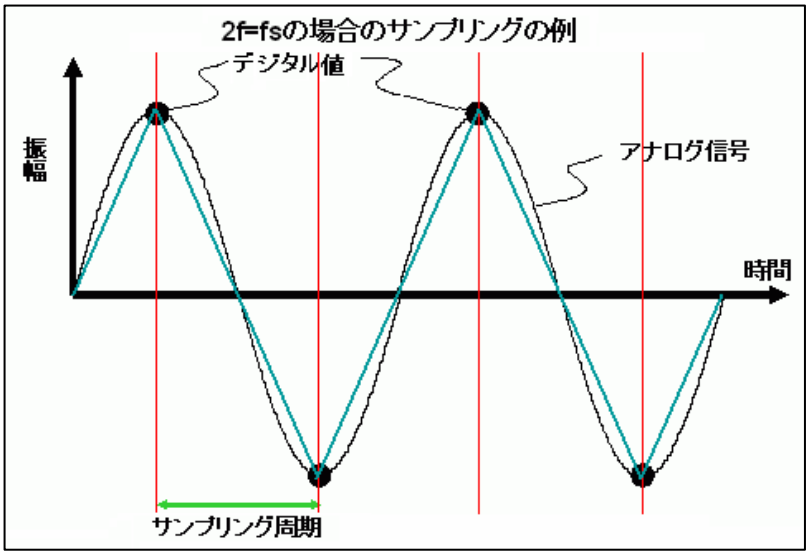
A/D 変換する場合、サンプリングについて次のような注意が必要です。

(1) サンプリング周波数とアナログ信号の周波数の関係

サンプリング周波数(fs)は、アナログ信号の周波数(f)の 2 倍より大きくしなければなりません(シャノンのサンプリング定理)。たとえば次図のように、アナログ信号の周波数とサンプリング周波数が同じ場合には、デジタル化した値が一定になり、アナログ波形をデジタル波形で近似できません。



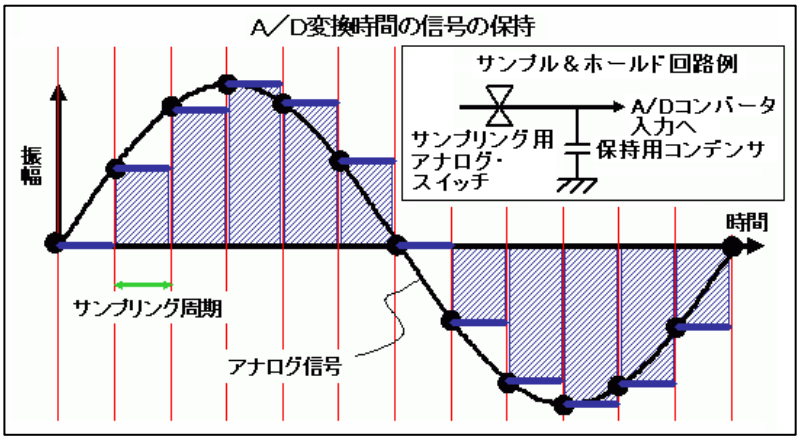
次に、サンプリング周波数(fs)をアナログ信号の周波数(f)の 2 倍にした場合を次図に示します。2 倍で、デジタル波形がやっと同じ周波数の三角波になります。位相がずれれば、振幅が小さくなってしまうので、サンプリング周波数をアナログ波形の周波数の 2 倍よりもっと大きくしなければ、誤差が大きくなってしまいます。
このように、デジタル化したいアナログ信号の上限周波数の 2 倍以上でサンプリングできる(変換速度のある)A/D コンバータを選ばなければなりません。



A/D コンバータに入力しようとする実際のアナログ波形は、単一の周波数成分(f)ではなく、色々な周波数成分を持っており、サンプリング周波数の 2 分の 1 以上の成分も含まれています。そこで、A/D コンバータの前段に、サンプリング周波数の 2 分の 1 以上の成分をカットするロウパス・フィルタを使用します。ただ、一般的にロウパス・フィルタは、カットしたい周波数以上を急峻にカットできないので、A/D コンバータの入力信号がサンプリング周波数の 2 分の 1 以上の周波数成分を十分カットした信号になるように、カットオフ周波数を設計する必要があります。なお、ロウパス・フィルタのコンデンサやコイルによって、電源遮断時に、A/D コンバータの入力の絶対最大定格を超える可能性があるので、電源へ放電する保護ダイオードも必要になります。

(2) A/D 変換時間

A/D コンバータは、アナログ信号をデジタル信号に変換するのに一定時間を費やしますが、アナログ信号は常に変化しています。そこで、正確に変換するために、変換中にアナログ信号を一定に保持しておくサンプル＆ホールド回路を、A/D コンバータの直前に設ける必要があります。最近のマイコンは、この回路を内蔵しています。



A/D の分解能と精度(Resolution and Precision of A/D Converter)

A/D コンバータの変換結果を評価する指標として、分解能と精度があります。どちらも同じような使い方をされますが、区別する必要があります。

[A/D の分解能]

A/D の分解能は、A/D コンバータでアナログ入力電圧を変換した結果が何ビットのデータで得られるかを示します。通常のマイコンに内蔵されている A/D コンバータは、8～10 ビットの分解能のものがほとんどです。一般に、分解能が高いほど正確な結果が得られ、また、分解能以上の結果は得られません。分解能が高くなると、A/D 変換では避けられない量子化誤差の影響も小さくなります。それでは、分解能を高くするだけでいいかというと、そう簡単ではありません。分解能を高くしようとすると、それだけ変換時間が余分に必要となります。また、デジタル信号が高速でやり取りされているマイコンの中では、その影響もあり、分解能に応じた精度が得られません(クロス・トークを参照)。

【コーヒー・ブレーク】
デジタル・オーディオの分野では、ポピュラーな CD では 16 ビット分解能の PCM でデータが記録されています。また、DVD Audio は最大 24 ビット分解能の PCM で記録されています。
一方、デジタル画像分野では、画像そのものは圧縮されているので、直接の比較はできません。映像出力 DAC では、RGB で 6～8 ビットのものが普及していますが、高級機と呼ばれるものでは、14 ビット程度のものを使用している機器も見られます。

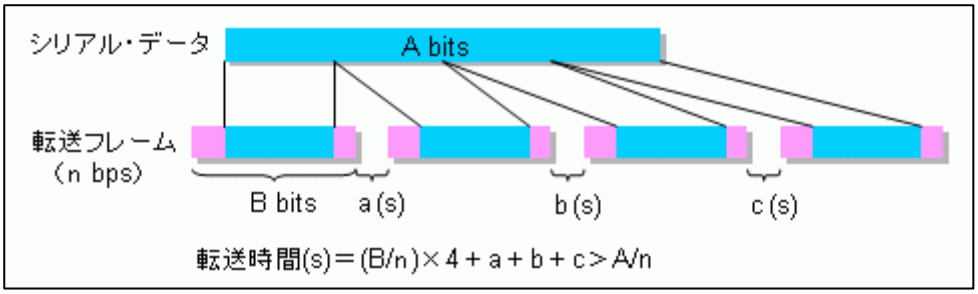
[A/D の精度]

A/D コンバータの変換結果の正確さを示す指標として精度があります。同じ分解能の A/D コンバータであっても、いつも同じ変換結果が得られるとはかぎりません。たとえば、マイコン内蔵の 10 ビット分解能の A/D コンバータでも、動作している電源電圧が低くなると、変換誤差が 0.4%から 0.6%、1.2%と大きくなります。その分、精度が悪くなります。

bps(Bits Per Second)

シリアル通信での、1 秒あたりの転送ビット数を表す、データ転送速度の単位です。UART などでは「ボー」を用い、データ転送速度を「ボー・レート」と呼びます。

なお、データ部分の転送は定義された速度で行われますが、データ通信は一定データ量ごとに区切ったフレームで行われるので、各通信プロトコルのフレーム処理がオーバヘッドとなり、転送に要する時間は、データ量を転送ビット速度で除算した時間より長くなります。また、そのデータ転送より優先度の高い処理が介入すると、より長い時間を要します。

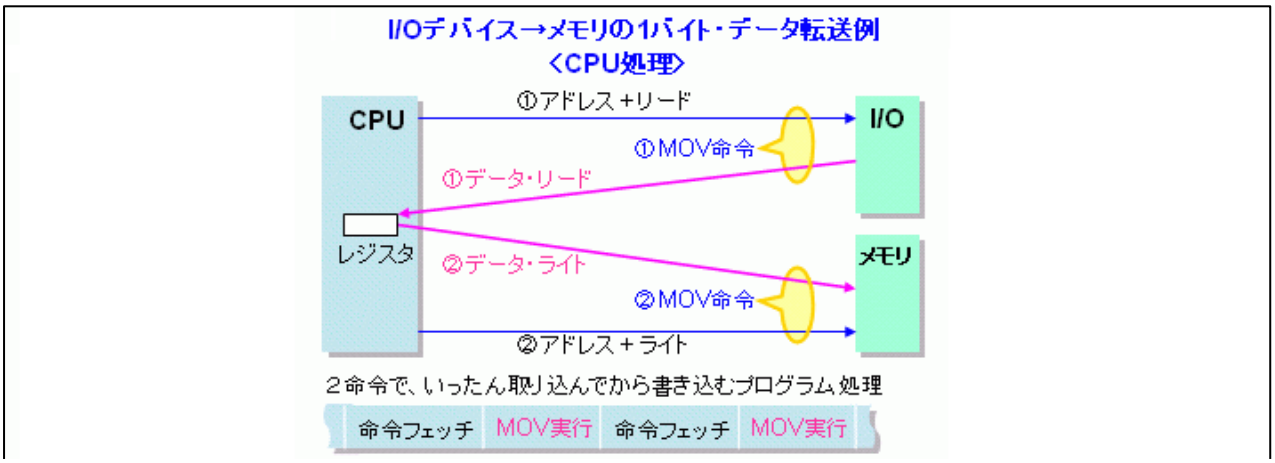


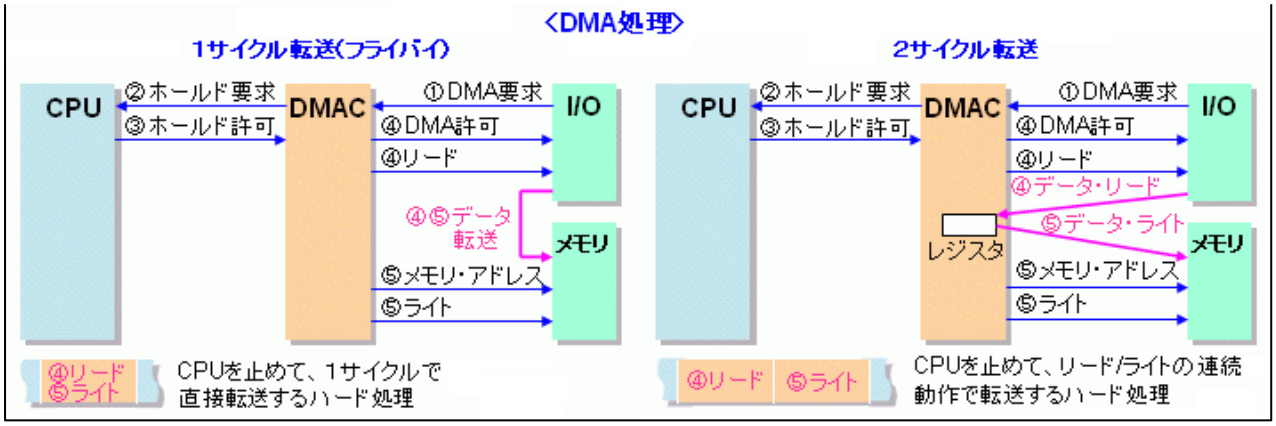
DMA(Direct Memory Access)

マイコンの応用システムで、I/O デバイスやメモリの間でデータを転送する場合、通常のアクセスでは、CPU が転送元と転送先のアドレスをそのつど（1 バイトごとに）指定して、いったん取り込んだデータを出力します。このため、アドレスの更新管理が必要ですし、プログラム処理なので、転送に時間がかかり、大量データの処理では大きな時間差になります。特にリアルタイムの動画データや音声データ、ディスク装置とのデータ転送など、CPU 処理では間に合わなくなります。

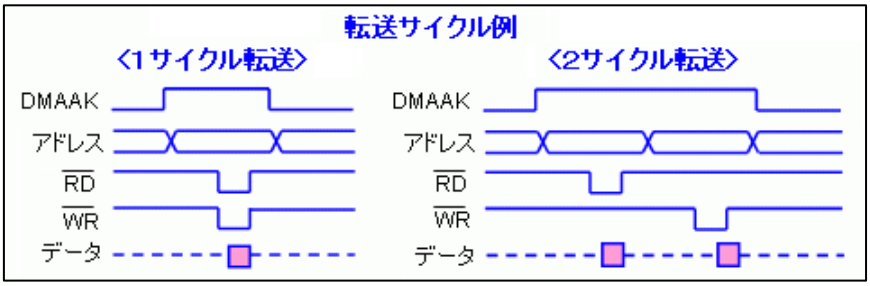
CPU を介さずに、このデータ転送を高速化するのが DMA です。DMA コントローラが DMA を実行しますが、データは取り込まず、バスの駆動だけします（ただしメモリ-メモリ間の転送では、両アドレスの切り替えが必要なので、いったんデータを取り込みます）。I/O デバイスに対して DMA コントローラは、DMA 許可信号をセレクト信号としてアクセスするので、I/O のアドレスは出力しません。

DMA 転送に先立って、CPU は DMA コントローラに動作モードを設定するだけです。データ転送時には、外部から DMA 要求信号が DMA コントローラに入力され、CPU とのハンドシェークで CPU をホールド(停止)状態にして、データ転送を行います。





ここで、DMA コントローラによっては、リードとライトの 2 サイクルで転送するモードと、リードとライトを同時にして 1 サイクルで転送するモード（V850E/Mx シリーズに内蔵のフライバイなど）を持っている場合があります。



データ転送後、DMA 要求信号がアクティブなら、DMA コントローラは、事前に設定された転送バイト数(ターミナル・カウント：TC)に達するか、停止信号(TC/END)が入力されるまで、データ転送を継続します。実際は、データ転送ごとに TC をデクリメント(−1)して、0 になると次の転送を停止させます。なおオートロード・モードのように、TC＝0 になっても停止しない機能を持つ DMA コントローラがあります。このモードは、転送メモリ・アドレスと TC の初期値を自動的に再設定し、同じメモリ領域のデータを繰り返し転送でき、画像データの表示などに利用されます。

DMA 転送はモードによって、1 転送ごとに CPU へバスを開放する(ホールド要求を解除する)かどうかなど、動作が異なります。次に、転送モードごとのバス・サイクル例を紹介します。DMA コントローラによって、モードの名称や動作、サポートするモードが異なる場合があります。

転送モード例

転送モード	タイミング・チャート	動　作
シングル	DMARQ HLDRQ HLDAK DMAAK データ	DMARQを受け付けると、1回のDMA転送を行い、バスの使用権をCPUに開放します。 引き続きDMARQ入力があると、再度DMA転送を行います。TC/ENDに達しないうちは、DMARQ入力がある間、この動作を繰り返します。
シングル・ステップ	DMARQ HLDRQ HLDAK DMAAK データ DMA要求がなくなっても転送を継続 TC/END	DMARQを受け付けると、TC/ENDに達するまで、DMA転送とCPUのバス・サイクルを交互に繰り返します。
ディマンド	DMARQ HLDRQ HLDAK DMAAK データ バス開放せず	シングル転送と似ていますが、DMARQ入力がある間は、DMA転送だけを行い、バスの使用権をCPUに開放しません。 DMARQ入力が切れると、はじめてバスの使用権をCPUに開放します。
ブロック(バースト)	DMARQ HLDRQ HLDAK DMAAK データ DMA要求がなくなっても転送を継続 バス開放せず TC/END	シングル・ステップ転送と似ていますが、DMARQ入力を受け付けると、TC/ENDに達するまで、DMA転送だけを行い、バスの使用権をCPUに開放しません。

DMARQ：DMA要求(周辺→DMAC)
HLDRQ：ホールド要求(DMAC→CPU)
HLDAK：ホールド許可(CPU→DMAC)
DMAAK：DMA許可(DMAC→周辺)

EEPROM と Flash メモリ(Difference Between EEPROM and Flash Memory)

マイコンとともに使用されたり、マイコンに内蔵されたりする、電気的に書いたり消したりできる PROM（書き込み可能なメモリ）としては、EEPROM と Flash メモリがあります。両者はどのように使い分けするのでしょうか。

[構成による差]

Flash メモリは EEPROM の一種ですが、消去はブロック単位で行うことで、構造を簡素化しています。その結果、メモリ容量を大きくできますし、製造コストも安くなります。そのため、マイコンに内蔵されると、EEPROM は数十バイト程度が普通ですが、Flash メモリは数十 K～数百 K バイトになります。

[書き込み機能による差]

EEPROM は、必要なアドレスに対して、新しいデータを書き込むだけで、古いデータを書き換えることができます。一方、Flash メモリは、消去された状態から書き込むこと（"1"の状態から"0"にする）はできますが、逆方向の書き込み（"0"から"1"にする）はできません。この場合には、ブロック単位での消去をする必要があります。

[データの書き換え回数の差]

通常、EEPROM は 10 万回以上の書き換えが保証されています。それに対して、Flash メモリでは、そこまでの書き換え回数は保証されていません。78K の All Flash 製品では、デバイスにより異なりますが、100～1000 回が保証されています。この回数は少ないように思われますが、量産後のプログラムは、そんなに頻繁に書き換えることはないためと、むしろきちんとデータを保持することの方が重要なためです。

なお、開発段階では、この回数を大きく越えて書き換えることが可能です。この場合には、頻繁にプログラムを書き換えるので、長い間データを保持する必要がないので、書き換え回数を増やせます。

以上をまとめると、次のようになります。

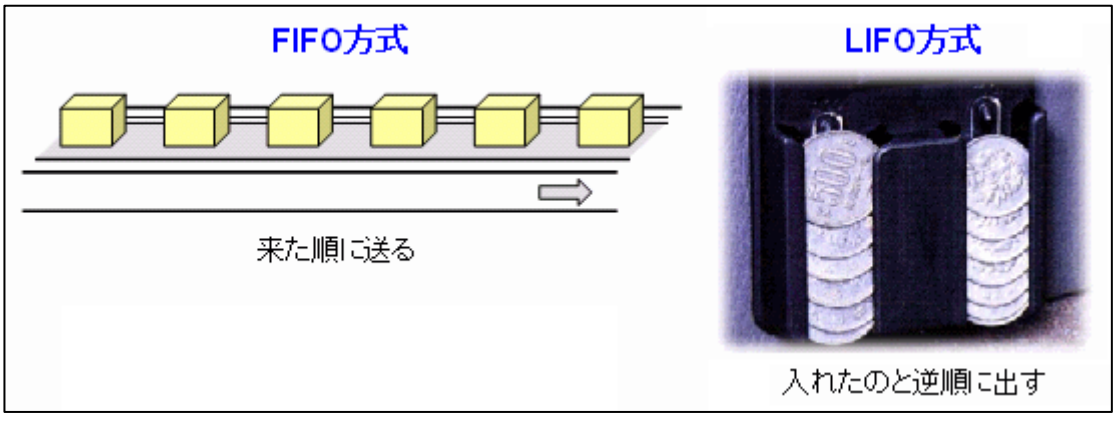
EEPROM　：RAM の延長で、電源断でも保持したい少ないデータの記憶用で、比較的頻繁に書き換える用途

Flash メモリ：ROM としてプログラムや大量の（固定／半固定）データ用で、頻繁に書き換えが必要ない用途

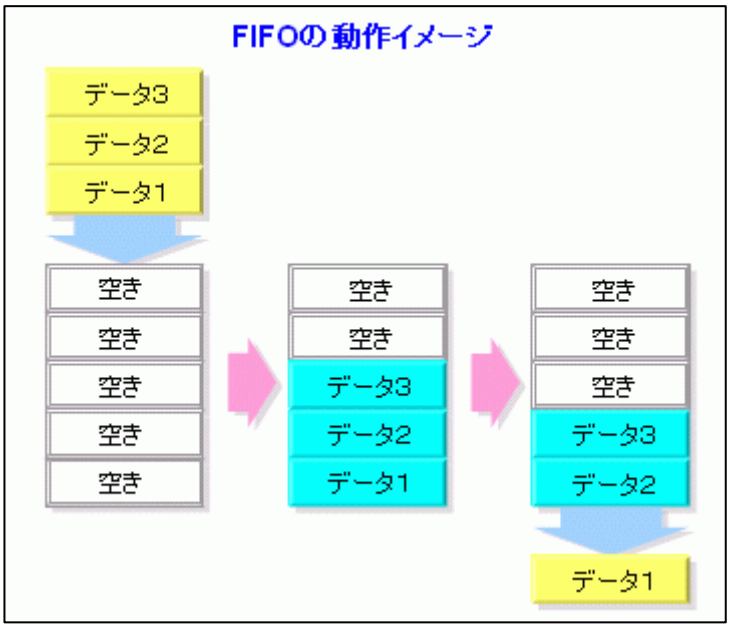
FIFO(First-in First-out)

一般に「ファイフォ」と読みます。

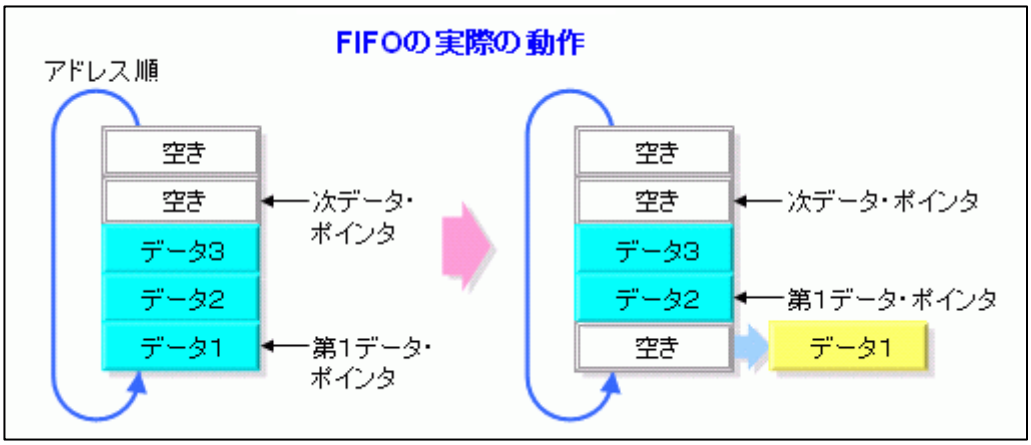
レジスタなどの格納データを、格納した順に取り出す方式のことで、この方式をユニット化したものも FIFO と呼びます。逆に、最後に格納したデータを先に取り出す方式を LIFO(Last-in First-out)と呼びます（スタック参照）。



キューと呼ばれるデータ構造や、通信データの送受信部分では、データ授受の時間差吸収に、一時格納領域として FIFO が利用されます。FIFO の段数が多いほど、吸収できる時間差が大きくなり、「欲しいときに取り出す」という自由度が高くなります。FIFO の格納データが満杯になると、プロセッサや送信側に信号で通知して、次のデータ格納を待たせます。逆に FIFO が空の状態も信号で通知します。
一般に FIFO は、入力データを最終段から格納してゆく概念で表現します。また、データを出力すると、残りのデータが最終段以降にシフトするイメージです。



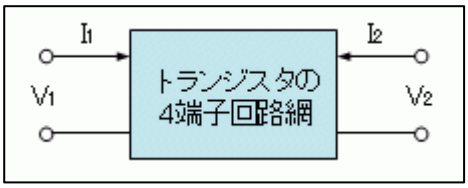
実際には、リング・バッファになっていて、内部でデータがシフトするのではなく、ポインタが格納位置を更新管理します。



ここで、次データ・ポインタのアドレスが、「第1データ・ポインタのアドレス－1」のときに、1 アドレスしか空きがないことをアラーム信号で、プロセッサや送信側に知らせます。
この状態で次のデータを格納すると、第1データ・ポインタと次データ・ポインタのアドレスが一致して、満杯の状態になります。また、格納データを順次取り出して、第1データ・ポインタと次データ・ポインタのアドレスが一致すると、空の状態になります。

h パラメータ(h-parameter:Hybrid Parameter)

トランジスタ回路で用いられる、Z パラメータ(open-circuit impedance parameters)と Y パラメータ(short-circuit admittance parameters)を組み合わせた回路定数です。トランジスタの 4 端子回路網では、下図のように、出力電流 I_2 の向きが、一般の 4 端子回路網とは逆になっています。



ここで、入力側の電圧 V_1 と出力側の電流 I_2 は、h パラメータによって次のように表されます。

$V_1 = h_{11}I_1 + h_{12}V_2$

$I_2 = h_{21}I_1 + h_{22}V_2$

パラメータ h_{11}、h_{12}、h_{21}、h_{22} は、それぞれ h_I、h_R、h_F、h_O で表され、エミッタ接地回路では h_{IE}、h_{RE}、h_{FE}、h_{OE} です。

h_I:入力インピーダンス(Ω)

h_R:電圧帰還率

h_F:電流増幅率

h_O:出力アドミタンス(S)

つまり、トランジスタの規格 h_{FE} は、エミッタ接地での電流増幅率のことです。

mil

「ミル」と呼ばれ、米国防総省の MIL(Military)規格で使用されている長さの単位で、1/1000 インチを表します。

たとえば、300mil は 7.62mm(=300×25.4/1000)です。

半導体デバイスでは、DIP や SOP のパッケージ本体幅を表現するのに用いられていましたが、メートル法施行により、現在は mm 表記が一般的になっています。

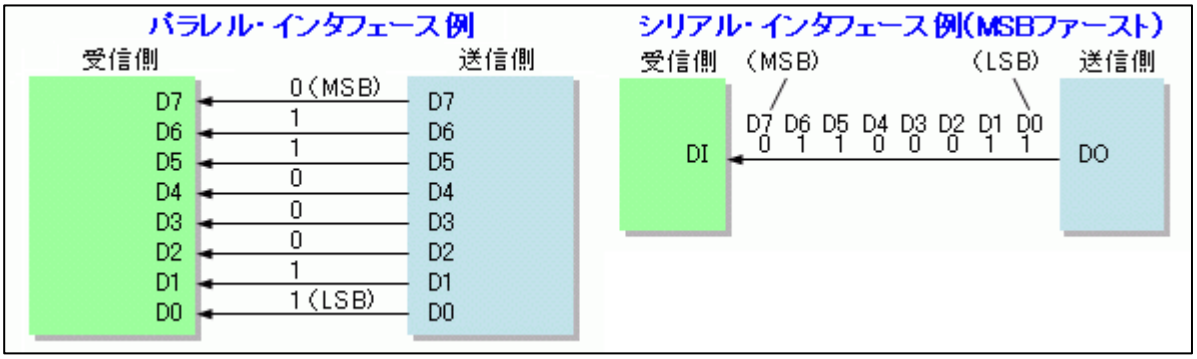
なお、mil 表記に慣れた方への配慮で、当社の外形図では「7.62mm(300)」などの表記をしています。

MSB/LSB(Most Significant Bit/Least Significant Bit)

デジタル・データのビット列について、MSB は最上位ビット、LSB は最下位ビットのことです。

デジタル・データは 2 進数で、通常の数の表記と同様に、左端が最上位けた、右端が最下位けたです。たとえば、10 進の 99 という数は、2 進で「(MSB)01100011(LSB)」と表されます。この場合、MSB は 0 で LSB は 1 です。

ブロック間のインタフェースを考えるとき、パラレル・インタフェースでは、送信側と受信側の各ビットが対応しているので、特に問題はありません。ところが、シリアル・インタフェースでは、1 本のデータ線で全ビットを送受信するので、送信側と受信側のプロトコルでビット順序の定義を一致させておく必要があります。上記データの場合、MSB ファーストでは″01100011″、LSB ファーストでは″11000110″の順に送受信されます。受信側も MSB ファーストか LSB ファーストかを送信側に合わせておけば、受信データのシリアル-パラレル変換によって、パラレル・データを再生できます。



【ティー・ブレーク】

ビットというのは 2 進数の最小単位で、2 進数は「けた(digit)」の二者択一状態(1 か 0)で構成されます。このため、2 進で数を扱うことを「けた」の派生語である「デジタル(digital)」と表現します。

ただし、2 進表記ではけた数が多くなるので、4 ビットを単位とした 16 進(ヘキサ)表記がよく用いられます。プログラムで 16 進を記述する場合、数の前に"0x"をつけたり、数のうしろに"H"をつけたりします(C 言語やアセンブラで定義)。昔は 3 ビットを単位とした 8 進数がよく使われていました。

なお、ビット数表記では、8 ビットを単位とした「バイト」をよく用います。「ワード」というのは 16 ビット以上を単位とした表記ですが、なんビットかという共通的な定義はありません。

ただしメモリでは、データ・ビットの構成によって容量が変わるので、アドレス数をワードと呼び、×n ビットでデータ・ビットの構成を表して、「2M ワード×8 ビット」(=16M ビット)などと表現します。SRAM でこの構成の場合、アドレス線が 21 本(A0～A20)でデータ線が 8 本(D0～D7)となります。

NF(Noise Figure)

NF は、入力信号源抵抗が発生する熱雑音(Ni)と、増幅素子が発生する雑音を入力換算し、その比を電力デシベルで表した値です。装置としての増幅器では、入力 S/N 比と出力 S/N 比の比を電力デシベルで表したものです。

以下、S/N 比から NF を導き出してみましょう。

増幅器に信号を通した場合、出力側の S/N 比は入力側での S/N 比より必ず悪化します。それは増幅器自体が雑音を発生するからであって、この S/N 比の悪化の度合いを電力デシベルで表し、NF(雑音指数)と呼びます。

So/No の方が Si/Ni より小さくなるため、NF で表すと必ず 1 より大きく(dB で表すと 0dB より大きく)なります。

S/N 比は大きいほど良いのに対して、NF は小さいほど良いということになります。

入力	→ 増幅器(増幅素子) →	出力
信号=Si	(A倍で N を発生)	信号=So=A*Si
雑音=Ni		雑音=No=A*(Ni+N)
(Ni:信号源抵抗 Rg から発生する熱雑音)		(N:増幅器の入力換算雑音)
S/N 比=Si/Ni		S/N 比=So/No

出力端でのノイズ(No)は、入力端でのノイズ(Ni:信号源抵抗 Rg から発生する熱雑音)の増幅倍率(A)に、増幅器が発生するノイズ(N:増幅器の入力換算雑音)が加わります。

信号=So=A*Si　　雑音=No=A*(Ni+N)

$$\frac{Si/Ni}{So/No}=\frac{Si/Ni}{A*Si/A*(Ni+N)}=A*Si*(Ni+N)/A*Si*Ni=1+N/Ni$$

$$NF(dB)=10\log_{10}(1+N/Ni)=10\log_{10}1+10\log_{10}N/Ni$$

ここで、$10\log_{10}1$ は 0 なので、$10\log_{10}N/Ni$ の項だけとなります。

$$NF(dB)=10\log_{10}\frac{N\ \{増幅器(増幅素子)の入力換算雑音\}}{Ni\ (信号源抵抗\ Rg\ から発生する熱雑音)}$$

OS(Operating System)

基本ソフトとも呼ばれ、コンピュータ・システムでアプリケーションが必要とするハードウエア・インタフェースの基本機能や、時間と手順の管理機能の集合体です。OS にはパソコン用の Windows、Mac OS や、UNIX 系の Linux、産業機器組み込みマイコン用のμITRON などがあります。

複数の処理を実行する場合の OS の処理形態には、次の 3 種類があります。

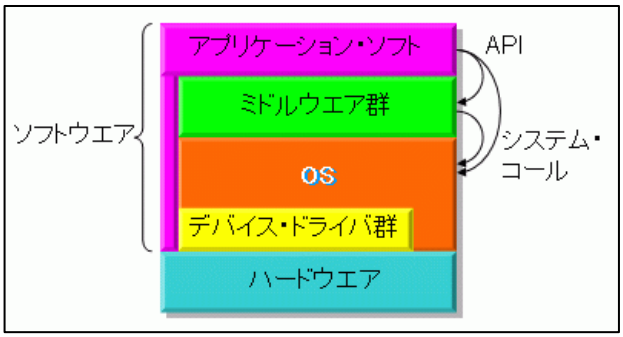
リアルタイム処理：随時の実行

TSS(Time Sharing System)処理：各処理の時間を区切って順次実行

バッチ処理：処理がひと段落するまで実行を続け、ほかを保留

事務処理にたとえると、大量のコピーに付きっきりで、終わってから電話の応対をするのがバッチ処理、一定時間ごとにコピーの進捗を確認しながら、合間に電話の応対をするのが TSS 処理、コピーが終わるまで平行して随時電話の応対をするのがリアルタイム処理です。

当社はμITRON 系のリアルタイム OS(RTOS)として、RX シリーズを提供しています。OS では各種のタスクが定義されており、それらを実行する処理モジュールが個々に用意されています。RTOS の場合、システム・コールとしてアプリケーションやミドルウエアから呼び出し関数が発行されるのを待っています(イベント・ドリブン)。ここで、アプリケーションが OS やミドルウエアに発行する関数は、API(Application Program Interface)と呼ばれます。



システム・コールが発行されると、ハンドラが割り込みによって、対応するタスクを起動します。OS にはセマフォという排他制御フラグがあり、起動タスクはセマフォを取得して、これが返却されるまで他のタスクは起動できなくなります。RTOS やマルチタスク OS では、複数のイベントが発生したときに、スケジューラがタスク制御のプロセスに優先度を割り当て、処理の順序をスケジューリングします。

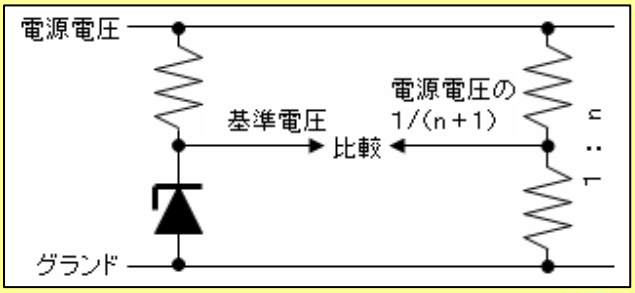
ハードウエアごとのインタフェースは、各デバイスドライバが行います。たとえばパソコンでは、マウス、キーボード、ディスプレイ、ディスク装置や通信回線などのハードウエアが接続されており、BIOS(Basic Input/Output System)が入出力を行います(最近の OS では、BIOS 機能を提供するものもあります)。

POC(Power-on-clear)

マイコンを含む論理回路は、電源投入時の内部の状態は不定となっています。このため、電源投入直後に初期化のために、リセットをかける必要があります(FAQの「クロックとリセット」をご参照ください。

通常は、リセットをかけるために専用の端子(RESET 端子)を使用しますが、これとは別に、デバイス内部に電源電圧を比較／検出する機能を内蔵して、検出した電圧が規定値以下の場合にリセットをかけることがあります。この機能をパワー・オン・クリア(Power On Clear)または POC(ポック)といいます。この機能を利用すれば、外部回路を削減できますから、システムのコスト削減や小型化ができます。

【コーヒー・ブレーク】
電源電圧を比較／検出というと、内部に昇圧回路をもっていて、そこで POC で指定された電圧を発生して、電源電圧と比較していると考える人がいます。実際には、そのようなことを行う必要はありません。電源電圧から指定された電圧の数分の 1 の基準電圧を発生します。その電圧と電源電圧を抵抗で数分の 1 に分割し、得られた電圧と比較するだけです。これでも、電源電圧を基準電圧と比較するのと同じ結果となります。



通常、POC は一定の電圧に固定されていますが、中には使用電圧に応じて、マスク・オプションなどで変更できるデバイスも存在します。

注意

POC は、個々のデバイスでの電源電圧を検出して初期化を行うものです。1 つのシステム内に POC を内蔵したデバイスが複数存在する場合は、検出電圧のばらつきにより、あるデバイスは初期化され、別のデバイスは初期化されないケースが発生することがあり、システムの誤動作につながることが考えられます。このような場合には、リセット信号を用いるなど、全体で整合をとる必要があります。

I^2C バスを用いた EEPROM では、リセット端子がないので、この問題が発生することがあります。そうなると、そのときの状態（データ・リードで SDA がロウ・レベルなど）によっては、I^2C バスが動作しなくなることがあります。このときには、ポートを用いて SCL にダミー・クロックを出力して、SDA がハイ・レベルになるのを待つしかありません。

r. p. m. (rpm：Revolutions Per Minute)

1 分あたりの回転数の単位で、モータやエンジンなどで用いられます。たとえば 3000 r. p. m. というのは、1 秒間に 50 回転（3000/60）する回転速度です。1 秒あたりの回転数を rps で表すこともあります。

【ティー・ブレーク】
回転系の出力(kW)は、「回転数 ×トルク(N・m)」に比例します。トルクは回転モーメント、N（ニュートン）は力の単位で 1N＝1kg・m/s^2 です。このため、車が坂を上るときに、同じエンジン出力でもギア比を下げれば、トルクが大きくなって登りやすくなりますが、回転数が下がるので、スピードはダウンします。
エンジン性能で ps という単位を見かけますが、これは馬力（仕事率）を表します。馬力は、ジェームス・ワットが馬 1 頭の仕事を基準に、蒸気機関のエネルギを定義したものです。このとき、ヤード・ポンド法に基づいて 1hp≒745.7W としました（hp：horse power：英馬力）。その後、フランスが用いていたメートル法に基づいて 1ps ≒735.5W が定義されました（ps：pferdestärke(ドイツ語で「馬の力」)：仏馬力）。1ps≒0.986hp です。

S/N 比(SNR：Signal to Noise Ratio)

信号伝達品質の尺度のひとつで、信号と雑音の比をデシベルで表したものです。この値が大きいほど、伝達品質がよいことになります。なお、磁気テープや磁気ディスク、光ディスクなどの記録媒体でも、再生特性のひとつとして S/N 比が用いられます。たとえば、カセット・テープでは約 50dB（信号電圧がノイズ電圧の 300 倍）、CD では約 100dB（信号電圧がノイズ電圧の 10 万倍）です。

S パラメータ(S-parameter：Scattering Parameter)

高周波の電子回路や部品について、反射／透過の特性を表す回路網パラメータのひとつで、**散乱パラメータ**とか**散乱行列(S 行列)**とも呼ばれます。

通常のトランジスタ回路では h パラメータが使われますが、一般に SHF 帯(3G-30GHz)以上の周波数では、電圧や電流の測定が困難なので、インピーダンスを基にした S パラメータが用いられます。

入力と出力がn対になった回路で、入力方向の信号の振幅を a_1～a_n、出力方向の信号の振幅を b_1～b_n とすると、回路網は次式で表されます。

$b_1 = S_{11}a_1 + S_{12}a_2 + \cdots + S_{1n}a_n$

$b_2 = S_{21}a_1 + S_{22}a_2 + \cdots + S_{2n}a_n$

$\cdots\cdots$

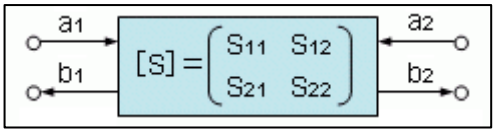
$b_n = S_{n1}a_1 + S_{n2}a_2 + \cdots + S_{nn}a_n$

これらの式は、行列で次のように表されます。

$$\begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix} = \begin{pmatrix} S_{11} \cdots\cdots S_{1n} \\ \vdots \quad \vdots \quad \vdots \\ S_{n1} \cdots\cdots S_{nn} \end{pmatrix} \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix}$$

この S_{11}～S_{nn} が S パラメータです。これらを複素数で表記すれば、反射の位相についても記述できます。

4 端子回路網では、S パラメータは次のようになります。



$S_{11} = (b_1/a_1)\ (a_2 = 0)$：入力反射係数

$S_{21} = (b_2/a_1)\ (a_2 = 0)$：順方向透過係数

$S_{12} = (b_1/a_2)\ (a_1 = 0)$：逆方向透過係数

$S_{22} = (b_2/a_2)\ (a_1 = 0)$：出力反射係数

なお、これらの特性は、次のように反射量の大きさ(MAG.)と位相(ANG.)で表されます。

Sパラメータ参考値：μPC2710TB (T_A = ＋25℃, V_{CC} = V_{out} = 5.0 V, I_{CC} = 22 mA)

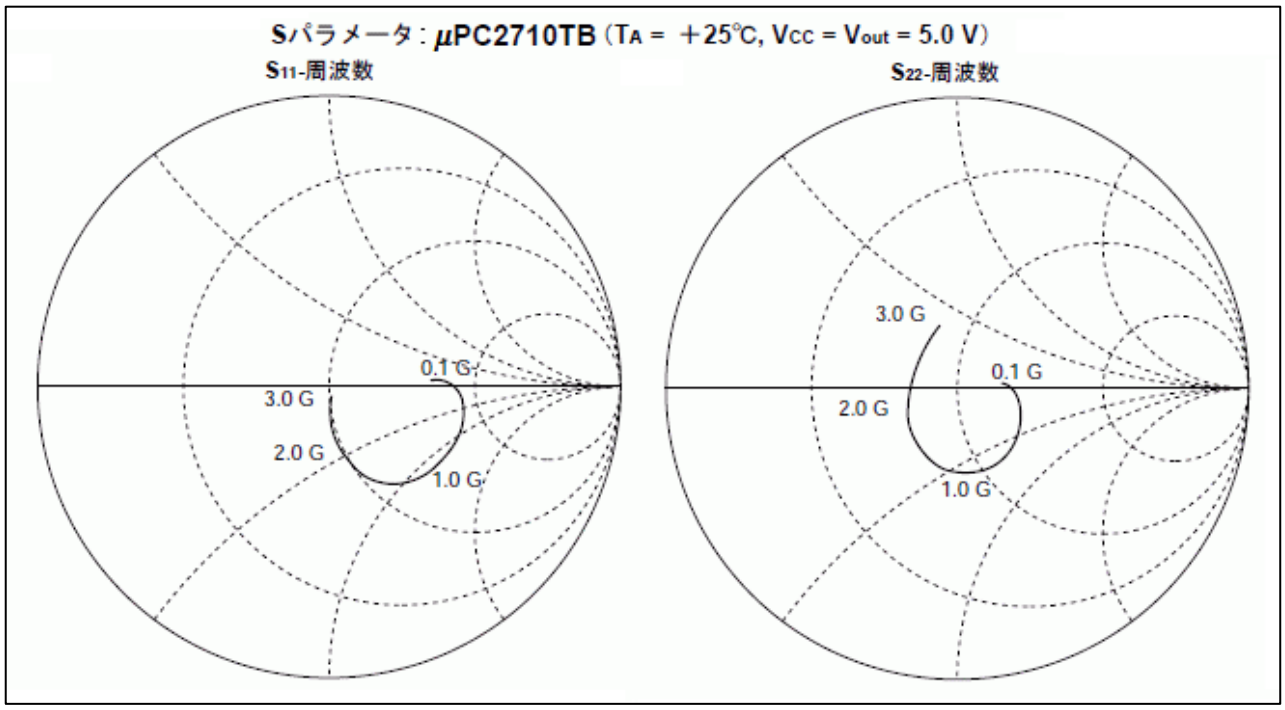
FREQUENCY	S_{11}		S_{21}		S_{12}		S_{22}		K
(MHz)	MAG.	ANG.	MAG.	ANG.	MAG.	ANG.	MAG.	ANG.	
100.0000	0.306	2.5	43.072	−8.4	0.012	15.2	0.156	2.7	1.08
200.0000	0.324	5.2	43.517	−17.1	0.010	10.7	0.164	2.1	1.17
300.0000	0.356	5.3	44.432	−26.5	0.010	20.2	0.185	0.3	1.10
400.0000	0.400	2.5	45.513	−36.9	0.012	26.9	0.225	−5.5	0.92
500.0000	0.439	−3.3	45.679	−48.1	0.012	27.0	0.255	−15.4	0.85
600.0000	0.469	−10.2	45.670	−59.7	0.013	31.3	0.283	−27.6	0.77
700.0000	0.481	−17.9	44.793	−71.8	0.014	34.9	0.301	−40.2	0.74
800.0000	0.488	−26.7	43.016	−84.3	0.014	27.9	0.312	−54.9	0.74
900.0000	0.479	−34.5	40.519	−96.0	0.013	26.6	0.316	−67.7	0.78
1000.0000	0.465	−41.2	37.946	−107.3	0.016	30.8	0.311	−79.5	0.79
1100.0000	0.448	−49.3	35.122	−117.9	0.016	26.6	0.307	−92.2	0.85
1200.0000	0.417	−54.9	32.108	−128.0	0.015	39.5	0.282	−104.6	0.99
1300.0000	0.387	−61.2	29.221	−137.0	0.015	39.7	0.270	−115.5	1.12
1400.0000	0.350	−65.2	26.656	−145.8	0.015	50.2	0.248	−127.0	1.27
1500.0000	0.316	−70.8	23.895	−153.9	0.013	50.8	0.236	−136.2	1.56
1600.0000	0.292	−74.0	21.576	−161.6	0.016	56.6	0.215	−145.3	1.49
1700.0000	0.256	−76.9	19.567	−168.1	0.015	69.0	0.200	−155.2	1.71

1800.0000	0.245	−80.5	17.743	−174.4	0.018	61.7	0.196	−162.5	1.59
1900.0000	0.215	−82.9	16.040	179.6	0.017	70.0	0.180	−173.4	1.88
2000.0000	0.201	−85.6	14.717	173.5	0.021	71.2	0.175	−178.1	1.71
2100.0000	0.177	−84.4	13.475	168.8	0.020	83.0	0.166	172.0	1.94
2200.0000	0.161	−88.8	12.327	163.1	0.021	76.7	0.171	167.7	1.99
2300.0000	0.145	−88.7	11.154	158.7	0.022	87.9	0.159	159.1	2.08
2400.0000	0.124	−90.3	10.262	154.4	0.023	81.4	0.164	154.0	2.15
2500.0000	0.113	−89.8	9.490	150.4	0.025	91.9	0.158	147.0	2.19
2600.0000	0.107	−91.9	8.793	146.4	0.028	88.7	0.166	141.8	2.06
2700.0000	0.091	−92.2	8.149	142.4	0.030	93.4	0.175	135.7	2.13
2800.0000	0.081	−94.9	7.652	138.9	0.031	92.1	0.183	131.6	2.13
2900.0000	0.067	−97.4	7.134	135.1	0.031	93.0	0.191	123.4	2.26
3000.0000	0.055	−103.8	6.726	131.5	0.039	88.3	0.200	118.9	1.97
3100.0000	0.039	−95.6	6.295	128.4	0.039	89.6	0.203	111.5	2.08

ここで、K は安定度係数で、大きいほど安定することになります。K の理論式は次のとおりですが、上記は実測値のため一致はしていません。

$$K = \frac{1 - |S_{11}|^2 - |S_{22}|^2 + |S_{11}S_{22} - S_{12}S_{21}|^2}{2 \cdot |S_{12}S_{21}|}$$

S パラメータの測定は、一般にネットワーク・アナライザで行われ、測定結果はスミス・チャートや周波数グラフなどで出力されます。次図に、スミス・チャートでの S パラメータの表記例を示します。



3. 通信関連用語

【 あ 】

アイソクロナス転送（Isochronous Transfer）

一定周期で転送する優先度の高い転送方式。
音声データなどに使用され、リアルタイム性を保証する。USB や IEEE1394 の転送モードの1つ。

アイソシンクロナス動作（Isosynchronous Operation）

アシンクロナス（調歩同期）フォーマットの送信データを同期 MODEM で混成モード制御すること。
調歩同期でクロック・レートを×1とした場合など、基本処理を変更せずに動作速度を高めることができる。

アカウント（Account）

ネットワーク・プロバイダのサーバに接続（ログイン）する権利で、一般的にはユーザIDを指す（PPPアカウント）。
メール・サーバを利用するときのユーザIDをメール・アカウントとも呼び、電子メール・アドレスのエイリャス(″@″の前の部分)に割り当てることもある。

アクセス・ポイント（Access Point）

パソコン通信サービスで、ユーザが電話をかける拠点。ノード局とも言う。アクセス・ポイントとホスト・コンピュータは専用回線で接続されており、ホストから離れているユーザでも電話料金の負担増にならない。

アシンクロナス転送（Asynchronous Transfer）

定期的には転送せず、すべてのアイソクロナス転送が終了してから、数バイトのヘッダとデータを転送し、受信側がアクノリッジを返す方式。IEEE1394 の転送モードの1つ。

アップリンク（Up-Link）

衛星通信のような双方向同時通信を行う場合、双方の通話周波数が同じだと混信するため、必ず送信と受信の周波数を別々に設定し、衛星を介して双方向通信を行う。
この場合、地上局から衛星に送る通話路をアップリンクと呼び、衛星から地上局に送る通話路をダウンリンクと呼ぶ。
＜反意語＞ダウンリンク

アドレス・フィールド（Address Field）

フレームの先頭フラグなどの次のフィールドで、端末の識別に使用する。

アプリケーション層（Application Layer）

→ OSI 参照。

【 い 】

イーサネット（Ethernet）

Intel、DEC、Xerox の共同開発による LAN（IEEE802.3）。
1本の回線に複数の装置を接続し、個々が通信制御を行う方式。″Ethernet″は Xerox 社の登録商標。

位相変調（Phase Modulation）

アナログ信号でビット・データを伝送する変調方法の1つ。
キャリア（搬送波）の位相を変化させ、異なる位相に″0″、″1″をそれぞれ割り当てる。

一芯コネクタ(One-Piece Optical Connector)

光伝送リンクで、単方向通信ができるように1本の光ファイバを組み込んだコネクタ。

一次群インタフェース(Primary Rate Interface)

PBX の設置やテレビ会議用などの高速チャネルを必要とする、大口顧客向けのインタフェース。
23 本の B チャネル(64kbps)と1本の D チャネル(16kbps)を束ねたインタフェースで、同期フレーム・ビットを1ビット付加して、計 1.544Mbps の転送レートとする。NTT ではこの一次群の商用サービスとして、INS ネット 1500 を運用している。
＜類語＞基本インタフェース

移動体衛星通信(Mobile Satellite Communication)

一般の移動体通信と異なり、中継器として通信衛星を使用。地上波を使った移動体通信では、多数の基地局を設置して移動体の追跡が必要だが、衛星の場合は1本のトラポンで広大なエリアをカバーでき、移動体の追跡も不要。
インマルサットという船舶対象のサービスに始まり、自動車対象に発展した。

移動体通信(Mobile Communication)

移動する端末との通信。自動車・新幹線などの車両や携帯電話などが対象。

インターネット(Internet)

パソコン通信、商用、学術関連の3種のネットワーク群を相互接続した世界最大のグローバル・ネットワーク。

インターネット・プロバイダ(Internet Provider)

→ NSP(Network Service Provider)参照。

インタネットワーク(Inter-network)

一般にネットワーク同士を接続することをいう。米国で大学や研究機関のネットワーク同士を接続したものを最初に「インタネットワーク」と称したため、狭義には学術機関のネットワークの相互接続を指す。
企業などでの LAN の普及により、遠隔地の LAN 同士の接続が新たなインタネットワークの形態となった。

インタラプト転送(Interrupt Transfer)

割り込みを使って、一定周期で少量データを転送する方式。キーボード入力データや装置のステータスのやりとりに使用され、伝送遅延時間が保証される。USB の転送モードの1つ。

イントラネット(Intranet)

Web を中心とした企業内の情報ネットワーク。

【 え 】

エコー・キャンセラ(Echo Canceller)

電話回線の2線－4線変換部で発生するエコーを予測し、受信信号から予測したエコー成分を除去して、相手の送信信号だけ取り出す機能。9600bps 以上のモデムでは、エコー・キャンセラで送信チャネルと受信チャネルを分離する。

【 お 】

応用層(Application Layer)

→ OSI 参照。

【 か 】

カスケード接続(Cascade Connection/Cascading)

複数チャネルを持った回路の出力を次段の同種回路に入力し、ネズミ算式にチャネル数を増やす多段の縦続接続。

仮想端末(Virtual Terminal:VT)

ホスト・システムに直結している端末と同様のサービスを、ネットワーク上の遠隔地端末(ホスト・システム介在もあり)に行うシステム。

【 き 】

ギガビット・イーサネット(Giga-bit Ethernet)

1Gbps 対応のイーサネットで、 ATM しか実現できないとされている帯域幅をプラグ&プレイで実現する(1000BASE)。

【 く 】

クライアント(Client)

ネットワークに接続して、サービスを利用する側(依頼者)のコンピュータ。

<反意語>サーバ

クライアント/サーバ型 LAN(Client/Server Type LAN)

LAN の構築をサーバで行い、これにクライアントのパソコンを接続する形態の LAN。

<関連語>ピア・ツー・ピア型 LAN

クライアント/サーバ・モデル(Client/Server Model)

インターネットなどの、おもなネットワークで使用されている分散処理型ネットワークの形態。サーバを中心に、複数のクライアントが接続されるスター型トポロジとなる。

クライアントは、サーバの機能を利用して、サービスを受ける側のコンピュータで、エンド・ユーザが操作するパソコンなど。サーバは、情報を集中管理して、他にサービスを提供するコンピュータ。

クライアントがサーバにサービスの処理を要求し、サーバがその応答としてクライアントに処理結果を返す通信プロトコルによって、ひとつの処理が行われる。

クロス・トーク(Cross Talk)

隣接する伝送路間の干渉。配線間の容量的・誘導的な結合(リアクタンス)などによって生じる。

【 け 】

ゲートウエイ(Gate-way)

LAN と LAN を接続する装置。

【 こ 】

コネクション型(Connection Oriented:CO)

通信のためのネゴシエーションをして、接続を確立・監視・保持するプロトコル。

コネクションレス型(Connectionless:CL)

送信先アドレスを指定して、接続の確立をせずに送信するプロトコル。

通信経路はダイナミックに最適経路を確保する。
＜関連語＞イーサネット

コントロール転送（Control Transfer）
ホストとターゲットの間で、コントロール・エンド・ポイントを使って行う制御用の転送方式。
デバイスの初期化や制御情報のやりとりに使用。USB の転送モードの1つ。

【 さ 】

サーバ（Server）
ネットワーク上のサービスを提供するコンピュータ（提供者）。
＜反意語＞クライアント

【 し 】

時分割多重方式（Time Division Multiplex：TDM）
時間を分割して、1本の伝送路を複数の端末に順次割り当てる方式。

ジャム信号（Jam Signal）
妨害信号。

周波数分割多重（Frequency Division Multiplex：FDM）
周波数帯をを分割して、1本の伝送路を複数の端末に割り当てる方式。
高速データを低速の狭帯域データに変換して、周波数軸上で並列伝送する。

周波数ホッピング（Frequency Hopping）
単位時間あたりの使用周波数を変化させ、実質的に広い周波数に分散させる方式。Bluetooth で利用。

シリーズAコネクタ（Series "A" Connectors）
USB のアップストリーム側のコネクタ。

シリーズBコネクタ（Series "B" Connectors）
USB のダウンストリーム側のコネクタの1種。

シリーズ・ミニBコネクタ（Series "mini-B" Connectors）
USB のダウンストリーム側のコネクタの1種。

【 す 】

ステートレス型 （Stateless）
LAN のサーバで状態保持をせず、クライアントに責任を負わせるシステム構成。
＜関連語＞NFS、TCP/IP

ストリーム（Stream）
プロセス間、ドライバ間の全二重の仮想通信路で、モジュール間の階層化を容易にする。

スプール（Simultaneous Peripheral Operation On-Line：SPOOL）

プリンタ出力やメール受信などで、データを直接送らず、一時的に別の場所に貯めて、あとでまとめて伝送すること。
その貯めておく領域（キュー）を指すこともある。

スペクトラム（スペクトル）拡散方式（Spread Spectrum）

伝送データを幅のある周波数帯に分散し、ノイズやジャムを抑える通信方式。
IEEE802.11b,Bluetooth,CDMA で利用。

【 せ 】

セッション層（Session Layer）

→ OSI 参照。

セットトップ・ボックス（Set-top Box：STB）

家庭用テレビに接続して、追加機能を提供する送受信端末装置。ケーブル・テレビのコントロール・ボックスや、通信カラオケの端末機器など、通常はテレビの上に置くため、このように呼ばれる。

セルラ（Cellular）

自動車電話や携帯電話を指す。コードレス電話と違う点は、移動しながら通話できること。
1つの基地局がカバーする通話可能エリアをセルと呼び、自動車電話や携帯電話では、基地局のカバー範囲をセル（細胞）状になるよう配置して、連続したサービス・エリアを構成している移動にともなってセルを切り換えることをハンド・オーバと呼ぶ。

セル・リレー（Cell Relay）

セル（小さな固定長データ）を使って、データの交換（スイッチ）を行う方式。
＜関連語＞ATM

セントロニクス・インタフェース（Centronics Interface）

米国 Centronics Data Computer 社が開発した、コンピュータとプリンタの8ビット・パラレル・インタフェースで、3本の制御線でハンドシェークを行う。明確な規定はない。

全二重通信方式（Full Duplex Transmission）

各通信装置内部に送信回路と受信回路を独立に持ち、個別の伝送路で同時に通信する方式。
半二重通信より大量のデータを効率良く伝送できる。代表的な応用例は電話の音声通信。
＜関連語＞半二重通信方式、単方向通信方式

【 た 】

ダイヤルアップ・ネットワーク（Dial-up Networking）

公衆電話回線などを利用して、遠隔地から LAN に接続する形態のネットワーク。

ダイバシティ受信（Diversity Reception）

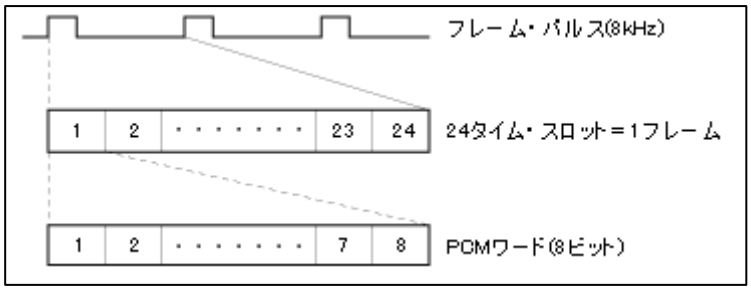
電波の受信変動を改善する方法。
移動体受信では、建物の陰で受信電波が弱くなったり、建物の反射波によるマルチパス受信を生じる。
そこで複数箇所にアンテナを設置し、受信状態のよいアンテナを選択して受信する。

＜関連語＞IMT-2000、W-CDMA、FOMA、cdmaOne

タイム・スロット(Time Slot)

PCM(パルス符号変調)で、1回のサンプリング(8kHz)で作られる1つの PCM ワード(8ビット)が占める時間領域。

μ 則時は 24 タイム・スロット(下図)。



＜関連語＞PCM、フレーム

ダウンリンク(Down-Link)

→ アップリンク(Up-Link)参照。

多元接続(Multiple Access)

→ マルチプル・アクセス参照。

単方向通信方式(Mono-directional Communication)

送信専用装置と受信専用装置で通信を行う一方向の通信方式。

＜関連語＞半二重通信方式、全二重通信方式

【 ち 】

調歩同期(Asynchronous Communication)

パソコンと通信機器間などに使用される一般的な通信プロトコル。

スタート・ビット(0)とストップ・ビット(1)でフレームを認識し、送信側と受信側の間で送受信クロックのやり取りが不要。

回線速度(ボー・レート)が比較的遅く、信頼性をあまり問わない装置に用いられる。

＜関連語＞プロトコル、ボー　＜反意語＞同期通信　＜類語＞非同期通信

直交波周波数分割多重(Orthogonal Frequency Division Multiplexing:OFDM)

TDMA、FDMA、CDMA などのデジタル多重変調方式の一つ。

FDM(周波数分割多重)に対して、さらに直交性を利用し、周波数軸上でのオーバラップを許容する。

直交変調(Orthogonal Modulation)

位相変調の一種で、主にデジタル信号によって位相変調をかける手法。

変調信号データから同相、直交(90°位相)の2つの多値信号を作り、異なる位相で各々を変調して加え合わせる。

＜類語＞位相変調

【 て 】

デイジ・チェーン(Daisy Chain)

回路の出力を次段の同種回路に入力する直列接続。

データ通信（Data Transfer）

データ伝送とも呼び、電気的な手段で情報を伝達する1つの形式。
従来の電信や電話、TV、ファクシミリと異なる点は、人間相互間の伝送ではなく、機械（データ処理装置）と機械、または適当な入出力装置を介して人間と機械の間で伝送されること。

データ・リンク層（Data Link Layer）

→ OSI 参照。

デーモン（Daemon）

UNIX 環境の裏側で、処理/管理をする常駐プログラム。

【 と 】

同期通信（Synchronous Communication）

大型コンピュータ間通信など、回線速度が速く高信頼性の用途に用いられるプロトコル。送信側と受信側の間で送受信クロックのやり取りか、装置内部でデータに同期した送受信クロックの生成が必要で、コスト高となる。
＜関連語＞プロトコル、COP　**＜反意語＞**調歩同期、非同期通信

トークン（Token）

ネットワークにデータを送信する権利を表現する特殊データで、これを所有するノードだけがネットワークへの送信権を持つ。ネットワーク上に 1 つだけ存在し、順次各ノード間で手渡される。

トークン・パッシング（Talken Passing）

端末同士をバス状に接続し、送信データの衝突を防ぐためにトークンという特殊パターンを流して、それを獲得した端末だけを送信可能とする通信プロトコル。
＜関連語＞LAN、CSMA/CD、トークン・リング

トークン・リング（Talken Ring）

端末同士をリング状に接続し、送信データの衝突を防ぐためにトークンという特殊パターンを流して、それを獲得した端末だけを送信可能とする通信プロトコル。
＜関連語＞LAN、トークン・パッシング

トポロジ（Topology）

ネットワークの物理的なシステム形態で、次のものがある。

ポイント・ツー・ポイント(1対1)	:専用線型で、相手が固定または常時接続
ポイント・ツー・マルチポイント(1対多)	:分岐型で、マルチドロップとマルチリービング（デイジ・チェーン）がある
マルチポイント・ツー・マルチポイント(多対多)	:LAN/WAN の形態で、スター型、バス型、リング型などがある

ドメイン（Domain）

サーバを中心としたネットワークを構成する管理単位。メール・アドレスの"@"のあとの部分がドメイン名。

トランザクション（Transaction）

処理プログラムなどの処理単位。フレームでは、一組の制御パケットとデータ・パケット。

トランシーバ(Transceiver)

Transmitter と Receiver の合成語。

伝送路で送信部(Transmitter)と受信部(Receiver)を切り替えて、半二重通信を行う回路。

＜関連語＞半二重通信方式

トランスペアレント・モード(Transparent Mode)

透過モード。データだけでなく制御キャラクタも含めて、すべての2進情報をそのまま転送するモード。

トランスポート層(Transport Layer)

→ OSI 参照。

トンネリング(Tunneling)

マルチキャストのパケットを IP パケットでカプセル化してルーティングすること。

【 に 】

二芯コネクタ(Two-Piece Optical Connector)

一般的には2本の信号線の接続器(コネクタ)。

光伝送リンクで、双方向通信ができるように2本の光ファイバを組み込んだコネクタ。

【 ね 】

ネゴシエーション(Negotiation)

折衝。コネクション型の通信で、接続に必要なパラメータの確認をしたり、通信エラー制御や回復処理、エラー通知などを行うこと。

＜関連語＞コネクション型

ネットワーク(Network)

複数の相手同士で相互に通信をする網の通信形態。

ネットワーク層(Network Layer)

→ OSI 参照。

【 の 】

ノード(Node)

ネットワークに接続されている端末や HUB などのネットワーク機器。

【 は 】

ハイアラーキ(Hierarchy)

トラフィックに応じて、伝送路を束ねたり分離する多重化構想。

パケット(Packet)

通信データをある一定の長さのフレームに分割したもの。

パケット交換(Packet Switching)

NTT が行っているサービスの1つで、パケットごとに番号を付与して送出する。
発信端末と着信端末間の通信経路は常に一定ではなく、網内の状態で変化する。

バースト転送(Burst Transfer)

1 回のアドレス指定で複数データをまとめて連続転送するモード。
転送ごとにアドレスを指定するサイクルがないので、データ転送速度が高い。

バーチャル LAN(Virtual LAN)

スイッチング HUB で論理的、ダイナミックに構成を変えられる仮想 LAN。
スイッチングには、セグメント型(LAN 単位)、ポート型(システム単位)、サブネット型(サブネット単位)がある。

波長多重方式(Wavelength Division Multiplexing)

1本の光ファイバで複数の信号を同時に伝送する通信方式で、マイクロ波通信における周波数分割多重方式に対応する。異なる波長の光に異なる信号を乗せ、きわめて多くの情報を送れる。

バックボーン(Backbone)

ネットワークで、システムの中核をなす基幹回線のこと。LAN でも、主要な幹線部分を指す。

ハブ(Hub)

マルチポート・トランシーバのように複数の LAN 端末を接続する装置。リピータ・ハブ(10BASE-T などの一般のハブ)、インテリジェント・ハブ(ネットワーク機能内蔵)、スイッチング・ハブ(バーチャル LAN を構成)がある。

パリティ(Parity)

調歩同期でフレーム・データの誤り有無のチェックに使用される。データ中の“1”の数が奇数または偶数になるように加える余分のビット(冗長ビット)。“1”の数が奇数なら奇数パリティ、偶数なら偶数パリティ。

バルク転送(Bulk Transfer)

別々のデータを一括転送する方式。デジタル・カメラ、MO、CD-ROM、プリンタ、スキャナなどの大量データ転送に使用され、転送速度は保証されない。USB の転送モードの1つ。

ハンドシェーク(Handshake)

複数の装置が互いに状態を確認しながら歩調を合わせて処理を行うこと。
パソコン通信ではフロー制御もこれにあたる。

半二重通信方式(Half Duplex Transmission)

1組の送受信回路を持った装置間で、1つの伝送路を切り換える通信方式。
代表的な応用例はインタホンや無線トランシーバ。
＜関連語＞全二重通信方式、単方向通信方式

【 ひ 】

ピア・ツー・ピア型 LAN(Peer-to-Peer LAN)

LAN に接続したパソコンが対等で、LAN の構築にサーバを必要としない形態の LAN。
＜関連語＞クライアント／サーバ型 LAN

光送信モジュール(Optical Transmitter Module)

光ファイバ通信に使用するE/O(電気→光)変換機能を有するモジュール。
簡単なものはLEDだけのものがあり、複雑なものではLEDレーザ駆動ICを内蔵したものや、温度補償にモニタ用のPD(フォト・ダイオード)を内蔵したものもある。

光通信(Optical Communication)

伝送信号として光を用いる通信方法。
空間に光線を発射する方法と、光通信ケーブルを使用する方法があり、電気ノイズや磁気ノイズに影響されない。

光伝送リンク(Fiber Optic Link)

光伝送モジュール(発光素子)と光受信モジュール(受光素子)、媒体としてプラスチック光ファイバを組み合わせた、通信用光結合素子。石英ファイバと半導体レーザを組み合わせた大容量長距離通信用光モジュールと比較すると、距離は50m以下、伝送容量は10Mbps程度の中容量近距離通信用。

	プラスチック光伝送リンク	石英光伝送リンク(ネオリンク)
伝送容量	～10Mbps	10Mbps～
伝送距離	～50m	1km～
使用素子	LED＋フォトICなど	半導体レーザ＋APDなど

光ファイバ(Optical Fiber)

光通信で信号を伝える媒体で、中心部のコアとそれを取り囲むクラッドから成る。おもな材質は石英ガラスで、髪の毛程度の太さの繊維にしたもの。軽量で曲げやすい、電磁誘導の影響を受けない、安価、低損失、広帯域など優れた特徴を持つ。石英ガラスの他には、多成分ガラス・ファイバやプラスチック・ファイバがある。

ビタビ・デコーダ(Viterbi Decoder)

TCMで変調された信号を復調するデコーダ。
現時点の信号状態だけでなく、過去の状態を記憶してデコードするため、耐ノイズ性が向上する。

非同期通信(Asynchronous Communication)

通信機器間で同期をとらず、ボー・レートとフレーム・パターンの統一でキャラクタを抽出する通信プロトコル。
調歩同期が一般的。
＜関連語＞プロトコル、ボー　**＜反意語＞**同期通信　**＜類語＞**調歩同期

【 ふ 】

ファイアウォール(FireWall)

ネットワーク外からの不正アクセスに対して防御するセキュリティ・システム。

ファイバ・ツー・ザ・ホーム(Fiber to The Home:FTTH)

一般家庭まで光ファイバを敷設して、 ISDNやCATVなども含めた各種の通信サービスを提供するための加入者網の光通信化構想。
基本的なシステム構成は、電話局の先にリモート・ターミナルを設置し、そこから家庭に光ファイバを引き込む。

ファイル転送(File Transfer)

LANを介して端末間でファイルを転送するサービス。

ファースト・イーサネット(Fast Ethernet)

10BASE を基に、100Mbps に高速化したイーサネット(100BASE)。

フィジカル層(Physical Layer)

→ OSI 参照。

輻輳(Congestion)

ネットワークのトラフィックが高くなり、伝送データがなくなったり速度が異常に遅くなる現象。
パケットなどの着信確認を行って、届いていなければ再送処理をするため、ネットワークがビジィで確認パケットの返信が遅くなると、誤って再送パケットを送って、さらにネットワークのトラフィックが高くなってしまう。これによって、確認パケットや再送パケット、制御パケットなどがネットワークを占有してしまい、有効な通信がほとんどできなくなる。

物理層(Physical Layer)

→ OSI 参照。

ブラウザ(Browser)

サービス内容をディスプレイに表示するソフトウエア。

プラグ&プレイ(Plug & Play)

ユーザが特に設定やインストールをすることなく、接続するだけで使用できるインタフェース。

ブリッジ(Bridge)

ネットワークパケットをデータ・リンク層で中継する装置。
ネットワークのトラフィックを軽減、異種ネットワーク間の中継、リピータ段数の延長などに使用。

プレゼンテーション層(Presentation Layer)

→ OSI 参照。

フレーム(Frame)

プロトコルに従って、伝送データを一定長に分割し、フラグやアドレスなどの制御コードを付加したビット列フィールド。
<関連語>プロトコル

フレーム・リレー(Frame Relay)

高速パケット交換サービス。従来のパケット交換技術をベースとするが、データごとのエラー訂正はしない。従来のパケット交換は 48kbps 以下、通常は 4800bps 程度だが、フレーム・リレーでは数十 Mbps の高速データ伝送が可能。

プロキシ・サーバ(Proxy Server)

インターネットとの接続窓口となり、企業セキュリティの確保のために設置するサーバ。

ブロードキャスト(Broadcast)

システム内の全ノードに同一情報を送信すること。

プロトコル(Protocol)

情報構造(フォーマット)と手順(プロシージャ)で構成される通信規約。フォーマットはブロックやフレームというデータの単位フィールドで、プロシージャには折衝(ネゴシエーション)、送受信、エラー・チェック/訂正、終結などがある。

【 へ 】

ペイロード(Payload)

ATM の物理層にマッピングされる伝送フレーム情報フィールド。

ベースバンド(Baseband)

直流のデータ信号をそのまま伝送路に送出して伝送する変調方式。FM、NRZ、NRZI、マンチェスタ方式がある。

【 ほ 】

ボー(Baud)

1秒間に伝送できる符号単位の変調速度。ビット単位の伝送では、ビット・レートと一致。

＜類語＞bps

ポータル・サイト(Portal Site)

インターネット・ブラウザの起動時に、最初にアクセスするウエブ・ページで、「入り口、玄関」の意味。

ホット・プラグ(Hot Plug)

活線挿抜。電源を投入したまま、抜き差し可能なインタフェース。

＜関連語＞IEEE1394、USB

ホップ・カウント(Hop Count)

TCP/IP で、相手ネットワークまでの距離のこと。ルータ(ゲートウエイ)の数のことを指す。

ホーム・ページ(Home Page)

本来は、Web ブラウザの起動時に表示する Web ページ。通常は、Web サービスの各コンテンツ・ページすべてを指す。

ポーリング(Polling)

データ伝送において中央制御局が、データ伝送路上に接続されている装置の状態を順次確認する「御用聞き」により、転送データの有無を判断する方法。

【 ま 】

マルチキャスト(Multicast)

システム内で選択した複数ノードに、同一情報を送信すること。

マルチプル・アクセス(Multiple Access)

多元接続。同時に多数の通信路を設定する／できること。その方式として、CDMA、TDMA、FDMA などがある。

＜関連語＞CDMA、FDMA、TDMA

マルチモード・ファイバ(Multi-Mode Fiber)

複数の光信号を1本で伝送する光ファイバ。ステップ・インデックス型とグレーデッド・インデックス型があり、いずれもシングルモード・ファイバよりコア径が大きい。このため、発光素子からの光を入射しやすく、光ファイバの接続も容易。光ファイバの導波モードが複数存在し、コア中を同時に伝搬するので、モード分散を生じて伝送帯域はやや狭くなる。

＜関連語＞モード分散

マンチェスタ(Manchester)

ビット・セルの前半がデータ・ビットの状態と同じで、中心で反転させるデータ変調方式。

＜関連語＞ベースバンド、NRZ、NRZI

【 む 】

無線 LAN(Wireless LAN)

配線を無線に置き換えた LAN システム(IEEE802.11)。オフィスの情報化に伴う、OA 機器の機能的な配置やレイアウト変更に対する自由度が高い。屋外 5.7GHz、屋内 5.2GHz と 5.3GHz(UNII 帯)の北米限定サービス(IEEE802.11a)、2.4GHz(ISM 帯)サービス(IEEE802.11b)がある。

【 も 】

モデム(MODEM)

Modulator Demodulator の略。通常はデータ伝送用変復調装置を指す。一般的には、2進のデジタル・データ信号を周波数変換し、アナログ回路に送出する変調装置と、 元のデジタル信号に戻す復調装置で構成される。

モード分散(Mode Dispersion)

光信号がファイバを伝搬中になまってしまう現象。伝送帯域を制限する要因の 1 つで、マルチモード・ファイバに生じる。入射端に入った矩形光パルスは、光ファイバ中を各種のモードに分かれて伝搬して、高次モードほど遅れて伝搬する結果、出射端での光パルス波形がなまる。

【 ゆ 】

ユニキャスト(Unicast)

1対1で、他のノードに情報を送信すること。

ユビキタス・コンピューティング(Uubiquitous Computing)

生活圏の随所にコンピュータを存在させ、自律的に相互連携して「いつでも、どこでも」という発想で、生活をバックアップする情報環境。コンピュータの存在は意識させず、必要に応じてネットワークに蓄積した個人情報などを参照する。1989 年に Xerox 社のパロアルト研究所が提唱した概念だが、携帯電話などの情報端末の小型化や、インターネットなどの通信技術の発展に伴って、再び注目されている。

【 り 】

リピータ(Repeater)

LAN の回線上の減衰信号を増幅、整形する装置。

【 る 】

ルータ(Router)

ネットワーク層でネットワークを相互接続する装置。ルーティング制御(経路の選択)には、スタティック・ルーティング(事前にルーティング情報を設定)とダイナミック・ルーティング(ルーティング情報を交換して最適化)がある。

ループバック機能(Loopback Function)

データが伝送路上で、ノイズ等の影響で化けていないかどうかを確認するために、受信側から送信側へ送り返すこと(折り返しテスト)。

【 れ 】

レイヤ(Layer)

通信に必要な機能を階層化したもの。OSI 基本参照モデルでは、通信機能をレイヤ1～7の7階層で分類している。

＜関連語＞OSI

【 ろ 】

ローカル・エリア・ネットワーク(Local Area Network)

→ LAN 参照。

【 A 】

ADPCM(Adaptive Differential Pulse Code Modulation)

適応差分パルス符号変調。音声データの圧縮方式の1つ(ビット・レート:20K～32Kbps)で、サンプリングごとに量子化幅 D を適応的に変化させ、さらに 2～4 ビットのデータへ符号化する。

たとえば 4 ビット ADPCM の場合、下図のように上位 1 ビットが極性(増減)を示し、下位 3 ビットで基本となる量子化幅 D の倍率を決定する。したがって一度に変化する量は、D の種類×8(3 ビット分)通り選べることになる。

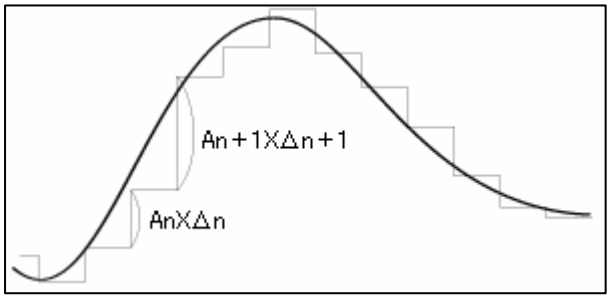
特長

・音声波形への追従性が良い。

・構成は簡単だが音質が良い。

・音声データの作成が容易。

ただしデバイス・メーカ間での互換性はない。



ADSL(Asymmetric Digital Subscriber Line)

xDSL の1つで、米ベルコア研究所が開発した高速データ通信方式。加入者電話回線(Subscriber Line)を利用して、数 Mbps～十数 Mbps の伝送をする。伝送速度は非対称(Asymmetric)で、電話局から利用者への下り方向では最大約 12Mbps、利用者から電話局への上り方向では最大約 2Mbps。4～5km までのサービスを想定。

＜類語＞xDSL、HDSL、RADSL、SDSL、VDSL

ANSI(American National Standards Institute)

米国規格協会。

API

ネットワーク上で提供するインタフェースで、2つの意味を持つ。

(1) システム上で、アプリケーション開発の環境を提供(Application Programming Interface)

(2) システム上で実行する、アプリケーションの利用環境を提供(Application Program Interface)

ASK(Amplitude Shift Keying)

振幅偏移変調。モデムの変調方式の1つで、キャリア(搬送波)の振幅変位により情報伝送する。

＜関連語＞FSK、PSK、QAM

ATA (AT Attachment)

ANSI(米国規格協会)による IDE の正式規格。IDE より伝送速度を向上させ、リムーバブル・メディアなどもサポート。

【 B 】

Bluetooth

ISM(2.4GHz)帯で約 10m の近距離を伝送する、周波数ホッピング型スペクトラム拡散方式のネットワーク。
モバイル機器への搭載を意識し、小型化、ロウ・パワー化、低コスト化が可能。
1台のマスタに最大7台のスレーブを接続。また伝送距離は約 100m まで拡張可能。

BOP(Bit Oriented Protocol)

同期通信の一方式で、イメージ・データ等の"0"、"1"データをやり取りする場合に適している。

bps(Bit Per Second)

データの伝送速度(1 秒当たりのビット数)を表す単位。

＜類語＞ボー

BPSK(Binary Phase Shift Keying)

二相位相変調。キャリア(搬送波)の位相変位により、情報伝送する変調方式。たとえば情報ビットのうち"1"に位相変化 "p"、"0"に位相変化"ゼロ"を割り当てて、搬送波の位相変化で情報を伝送する。
位相変化が大きく急であるため、変調された信号の周波数スペクトラムの広がりは大きい。

＜関連語＞PSK、QPSK

BRG(Baud Rate Generator)

調歩同期通信において、送受信クロックを生成する回路。

BS(Broadcasting Satellite)

放送用の静止衛星。これを利用した衛星放送を専用のアンテナとチューナで視聴する。地上波放送と違ってゴーストがなく、水平解像度も少し高い。国際条約で放送衛星を割り当てられている(日本は 8 チャネル)。

＜関連語＞CS

【 C 】

C バンド(C-band)

衛星通信用周波数のうち、4～8GHz 帯を指す。この帯域は大気層と電離層の影響を受けにくく、信号の減衰や雑音が最も少ないので、古くから衛星通信に広く利用されている。衛星通信でよく利用される周波数は次のとおり。

L バンド(1～2.6GHz) :携帯電話、第2世代コードレス電話
S バンド(2.6～4GHz) :衛星移動体通信、マリン・レーダ
C バンド(4～8GHz) :衛星放送
X バンド(8～12GHz) :政府間通信、軍事用通信
K バンド(12～40GHz) :衛星通信、衛星放送

CAI（Common Air Interface）

CT-2 方式のコードレス・ホン・システムにおいて、一つの携帯機で異なる事業者すべての基地局との接続が可能な共通無線インタフェース。

＜関連語＞CT-2

CAN（Control Area Network）

システム内でインテリジェント・デバイス（センサやアクチュエータ等）をネットワーク化するネットワーク・システム（ISO 11898）。伝送速度が高く、セキュリティ、信頼性、リアルタイム性に優れる。自動車産業で生まれ、温度・湿度範囲が広く、干渉、振動、電磁放射に強い。現在では、医療用機器など多業種で利用される。

CCIR（Comite Consultatif International des Radio Communications）

国際無線通信諮問委員会。

CCITT（Comite Consultatif International Telegraphique et Telephonique）

国際電信電話諮問委員会。現在は ITU-T。

CDMA（Code Division Multiple Access）

符号分割多元接続。IEEE802.11 無線 LAN の物理層プロトコル。

多数の通信路を確保する手段として、通信情報を拡散符号という暗号状の符号でスペクトル拡散（SS）し、信号帯域を数百～数千倍に広げて、それらを端末ごとにずらしながら送受信する方式。スペクトル拡散で通信するため、妨害に強く劣悪な通信路での使用が可能だが、変復調のための処理が複雑になる。

＜同義語＞SSMA（Spread Spectrum Multiple Access）

cdmaOne

IMT-2000 方式による au の第 3 世代携帯電話サービス。

＜関連語＞IMT-2000、W-CDMA、FOMA、ダイバシティ受信

CEPT（Conference Europeenne des Postes et des Telecommunications）

欧州郵便電気通信主官庁会議。

CIRC（Cross Interleave Reed-Solomon Code）

「サーク」と読む誤り訂正方式のひとつ。ランダム・エラー訂正能力の高い誤り訂正符号リードソロモンと、バースト・エラーをインタリーブなランダム・エラーに変換する手段を組み合わせたもの。

CL（Connectionless）

→ コネクションレス型参照。

CO（Connection Oriented）

→ コネクション型参照。

CODEC（Coder Decoder）

アナログ信号とデジタル信号の間の変換を行う部分で、符号化/複合化することで、「コーデック」と呼ばれる。

COP（Character Oriented Protocol）

同期通信の一方式で、送信側はデータの先頭に 16H または 32H を付加して、受信側にデータの開始を明示する。
データ形式としては，JIS コードや ASCII コードを用いる。キャラクタ・データをやり取りする場合に適している。

CRC（Cyclic Redundancy Check）

HDLC の FCS など、誤り検出に用いられる方式。通信局間で、この CRC によって生成されたデータをやりとりすることにより、通信障害等の際に発生する通信誤りを検出する。

CS（Communications Satellite）

通信用の静止衛星。CS テレビ放送には、JCSAT3 衛星を利用して音楽、映画、スポーツなどの情報をデジタル配信している「スカイパーフェク TV!」などがある。
CS と BS では、送信周波数帯とチャンネル割り当て周波数が異なるため、BS チューナでの受信はできない。
＜関連語＞BS

CSMA/ACK（Carrier Sence Multiple Access Acknowledge）

「シーエスエムエー・アック」と読む。米国オムニネット社が開発した LAN 用プロトコル。
データを送信する前に回線をセンス（監視）して、他に送信している端末がなければ送信を開始する。また送信中に他の端末が送信してデータが壊れた場合に、 ACK パケットと呼ばれるパケットを用いて、データの送達を確認する。

CSMA/CA（Carrier Sence Multiple Access/Collision Avoidance）

無線 LAN(IEEE802.11)で使用されている通信方式。MAC 層で回線上を監視して、データが衝突しないよう待ち合わせ制御をする。

CSMA/CD（Carrier Sence Multiple Access/Collision Detect）

イーサネット LAN(IEEE802.3)で使用されている通信方式。
回線上のデータの有無にかかわらずデータを送信して回線上を監視し、データの衝突を検出したら再送する。

CSU（Channel Service Unit）

通信サービス・ユニット。

CT-1（Cordless Telephone-1）

欧州のアナログ・コードレス電話システム。

CT-2（Cordless Telephone-2）

英国のデジタル・コードレス電話システムで発信専用。テレポイントとも呼ばれる。多重方式は FDMA/TDD を採用。
＜関連語＞CAI

CT-3（Cordless Telephone-3）

1990 年にエリクソン社が CT-2 をベースに開発した、発着信できるデジタル・コードレス電話システム。
多重方式は TDMA/TDD を採用。

CTIA（Cellular Telecommunication Industory Association）

米国セルラ電話工業会。

【 D 】

DCE(Data Circuit Terminating Equipment)

データ回線終端装置。

DDX 網(Digital Data Exchange)

NTT がデータ通信専門のサービスとして提供している通信網。
回線交換でデータ通信を行う DDX-C 網と、パケットでデータ通信を行う DDX-P 網がある。

DECT(Digital European Cordless Telephone)

欧州統一のデジタル・コードレス電話システムで、多重方式は TDMA/TDD を採用。

DFS(Distributed File System)

分散コンピューティング環境での分散ファイル・システム。

DNA(Digital Network Architecture)

DEC 社のネットワーク・アーキテクチャ。DECnet は、その商品名。

DNS(Domain Name System)

インターネット(IP ネットワーク)上で、ホスト(コンピュータ)にドメイン名を付けて、 IP アドレスに変換するプロトコル。

DPLL(Digital Phase-Locked Loop)

受信データの歪み(回線遅延等によって発生する歪み)に追従した受信クロックを生成する回路。
受信側に PLL 機能があれば、送受信のためのクロックを端末間でやり取りすることが不要になる。
DPLL はアナログ PLL ほどの精度ではないが、デジタル回路で一様な特性を容易に実現できる。

DSLAM(Digital Subscriber Line Access Multiplexer)

複数の xDSL 回線を束ね、ルータなどの通信機器と接続して、高速で大容量な基幹回線(バックボーン)への橋渡しをする集線装置。電話局内に設置。

DSRC(Dedicated Short Range Communication)

狭域通信。800MHz 帯の電波を使用した符号分割多元接続方式の携帯無線通信システム。
ITS(Intelligent Transport System)の路車間通信方式として、ETC や駐車場、ガソリン・スタンドなどの料金徴収システム、走行支援道路システム(AHS:Advanced Cruise-Assist Highway System)などへ応用されている。
<関連語>ETC

DSU(Digital Service Unit)

NTT が提供するデータ網(DDX-P、INS ネット 64 等)の終端装置。
端末はこの装置に接続することにより、網を介して相手端末と通信が可能となる。

【 E 】

ECC(Error Checking and Correction)

受信側で、転送途中に発生した誤りがデータ中にあることを検出し、元の正しいデータに訂正できるように、一定規則に基づいた冗長ビットを含ませた誤り訂正方式。

EHCI(Enhanced Host Controller Interface)

USB をサポートするパソコンのインタフェースの1種(インテルが規定)。

<関連語>OHCI、UHCI

EIA(Electronic Industries Association)

米国電子工業会。物理的、電気的インタフェースの標準化機関。

RS(Recommended Standard)として、RS-232C,422,423 などを規定している。

EISA(Extended ISA)バス

ISA と互換性を維持しつつ機能・性能を向上させたバス。アドレス/データ・バスを 32 ビットに拡張、バス・クロックは 8.33MHz のまま新たなバス・サイクルを追加、データ転送速度が向上(33M バイト/s ピーク)。DMA も 32 ビット化し、バス・マスタ機能のサポートも ISA より大幅に改善。

ETC(Electronic Toll Collection)

自動料金収受システム。車載器と有料道路の路側無線装置の間で DSRC 通信を行い、停車せずに通行料金を預貯金口座から自動収受するシステム。

<関連語>DSRC

ETSI(European Telecommunications Standards Institute)

欧州電気通信標準化機構(民間)。

【 F 】

FDDI(Fiber Distributed Data Interface)

WAN の一例で、光ファイバを用いて 100Mbps 以上のデータ通信を可能にするネットワーク。

<関連語>WAN、LAN、MAN、IVDLAN

FDM(Frequency Division Multiplex)

→ 周波数分割多重参照。

FDMA(Frequency Division Multiple Access)

周波数分割多元接続。FDM による多重通信アクセス。

→ 周波数分割多重参照。

FFT(Fast Fourier Transform)

周波数分析を高速に行うための計算方法。

周波数分析とは、いろいろな周期や強さの正弦波を合成した信号を分析して、元の信号の周期や強さを見つけること。この計算には乗算を多数含むので、高速化のため DSP(デジタル・シグナル・プロセッサ)が用いられる。

FIFO(First-In First-Out)

「ファイフォ」と読む。先に格納したデータを先に取り出す複数段のバッファ回路。エンプティやフルであることを示す信号で、ホストに状態を通知する。これにより、通信回線(シリアル・データ)とは非同期に、送信バッファには送信すべきパラレル・データをホストが書き込み、受信バッファからは受信してパラレル変換したデータをホストが読み出せる。

FM(Frequency Modulation)

周波数変調。搬送波の周波数を信号の振幅に応じて変化させる変調方式。

FOMA(Freedom Of Mobile Multimedia Access)

「フォーマ」と読む。IMT-2000 方式による NTT ドコモの第 3 世代携帯電話サービス。

＜関連語＞IMT-2000、W-CDMA、cdmaOne、ダイバシティ受信

FSK(Frequency Shift Keying)

周波数偏移変調。モデムの変調方式の1つで、キャリア(搬送波)の周波数変位により情報伝送する。たとえば情報ビットのうち、"1"に周波数 f1、"0"に周波数 f2 を割り当て、その周波数変化により情報を伝送する。

＜関連語＞ASK、PSK、QAM

FTP(File Transfer Protocol)

リモート・ネットワーク上のシステム間で、TCP によるファイル転送を行うプロトコル。

【 G 】

GGP(Gateway to Gateway Protocol)

TCP/IP で IP データグラムを相手局に届けるための通信経路情報(ゲートウエイ数とネットワークのアドレス・グループ)を交換するプロトコル。

GPS(Global Positioning System)

米国海軍が使用を始めた、衛星からの電波を受信して位置を知る装置。衛星が発射する電波を受信し、電波の到達時間から自分と衛星との距離を計算し、位置を検出する。

衛星からは、軍事用の電波のほか、民間用の電波も発射されている。自動車に搭載して、位置を地図上に表示するカー・ナビゲーションに使われて、一般に知られるようになった。

民間用の電波を使ったナビゲーションの精度は 30m～100m 程度。測量用に使う場合の精度は 1～2cm 程度。

【 H 】

H シリーズ勧告(H Series Recommendation)

ITU-T の国際標準のうち、マルチメディア情報通信システムの基本的プロトコル標準(H.120～H.331)。

H.310 :ATM(AAL1/5)で MPEG2(H.262)画像を送信。

H.320 :既存の狭帯域 ISDN で H.261 画像を送信。

H.323 :TCP/IP プロトコルで H.261/263 画像を送信。

H.324 :電話網(V.43)で H.261/263 画像を送信。

HTTP(Hyper Text Transfer Protocol)

WWW として発展したインターネットのアプリケーション・レベルのプロトコル。

分散環境のハイパーメディア情報システムに適し、容易にマルチメディアを実現する。

【 I 】

I シリーズ勧告(I Series Recommendation)

ITU-T の国際標準のうち、ISDN に関してまとめた勧告(I.110～I.751)。

I.430 :ISDN 端末と DSU/デジタル PBX 間の物理的特性、電気的特性、論理的特性。

I.440,441 :ISDN 端末と DSU/デジタル PBX 間の D チャネル転送プロトコル。

I.450,451 :発信端末と着信端末間を接続するための中継網の選択や、通信経路を設定するプロトコル。

I.460 :既存端末を ISDN に接続するため、その通信速度を 64kbps に変換する速度変換とプロトコル変換。

I.461 :I.460 をうけ、X.21 端末を ISDN に接続するのに必要な速度変換則とプロトコル変換。

I.462 :I.460 をうけ、X.25 端末を ISDN に接続するのに必要な速度変換則とプロトコル変換。

I.463 :I.460 をうけ、V シリーズ端末を ISDN に接続するのに必要な速度変換則とプロトコル変換。

I^2C バス(I^2C Bus)

「アイ・スクェア・シー・バス」または「アイ・トゥー・シー・バス」と読む。フィリップス社が開発した装置内シリアル・バス。バイト転送を自律制御し、シリアル転送の状態を監視する。

IDE(Integrated Drive Electronics)

パソコンと内蔵用ハードディスクを接続する標準規格で、CD-ROM ドライブや MO ドライブなども接続できる。
1本のケーブルでマスタとスレーブの2台を接続でき、パソコンのプライマリとセカンダリの2つコネクタと、合計4台の装置を接続できる。現在は IDE を拡張したエンハンスト IDE やウルトラ ATA、ウルトラ ATA/66 が主流。

＜関連語＞ATA

IEBus(Inter Eguipment Bus)

NEC の半導体部門(現ルネサス エレクトロニクス社に統合)が開発した、装置間の小規模のデジタル・データ伝送システム用のバス。半二重非同期通信方式で、CSMA/CD によるアクセス制御を行う。

IEEE(Institute of Electrical snd Electronics Engineers)

「アイ・トリプル・イー」と読む。米国電気・電子技術者協会。通信関連のおもな規定は次のとおり。

<table>
<tr><td>IEEE802</td><td>LAN/MAN<table>
<tr><td>IEEE802.3</td><td>1/10/100/1000Mbps イーサネット</td></tr>
<tr><td>IEEE802.4</td><td>1/5/10Mbps ブロードバンド(トークン・パッシング・バス)</td></tr>
<tr><td>IEEE802.5</td><td>4/16Mbps トークン・パッシング・リング</td></tr>
<tr><td>IEEE802.9</td><td>ISDN と整合性を持つ 4/20Mbps マルチメディア LAN、16Mbps アイソクロナス・イーサネット(IS LAN)</td></tr>
<tr><td>IEEE802.11</td><td>無線 LAN
802.11a:5.7GHz(54Mbps)
802.11b:2.4GHz(11/22Mbps)
802.11g:2.4GHz(54Mbps)
802.11n:2.4/5GHz(300Mbps)</td></tr>
</table></td></tr>
<tr><td>IEEE1394</td><td>コンピュータと周辺装置、民生機器のインタフェース(100/200/400Mbps、ホット・プラグ(作動中の着脱)採用)。周辺装置を最大 63 台まで接続可能。</td></tr>
<tr><td>IEEE1520</td><td>ネットワーク・アプリケーションのプログラミング・インタフェース</td></tr>
</table>

IETF(Internet Engineering Task Force)

インターネット上の新技術を標準化するために設立されたコンソシアム。
ドキュメント RFC(Requests For Comment)を発行している。

IMAP(Internet Message Access Protocol)

「アイマップ」と読む。インターネットでメールを読み出し／送信するためのプロトコルで、RFC1730～1733 で定義。

IMT-2000(International Mobile Telecommunications-2000)

ITU 標準の第 3 世代の移動体通信システム。「2000」は、周波数帯に 2GHz(2000MHz)使用と 2000 年サービスを開始目標に由来する。実現方式は、 NTT ドコモとエリクソン(スウェーデン)などが共同開発した「W-CDMA(日欧方式)」(max.2Mbps)と、クアルコムが開発した「cdma2000(米国方式)」(max.153kbps／384kbps)、クアルコムがデータ通信に特化した「HDR」(max.2.4Mbps)が有名。
＜関連語＞W-CDMA、FOMA、cdmaOne、ダイバシティ受信

IP(Internet Protocol)

TCP/IP ネットワークのネットワーク層プロトコル。

IPv6(Internet Protocol version 6)

インターネットのユーザ増加に対応するため、IETF の IPNG ワーク・グループが、IPv4 の後継として推進しているプロトコル。IP アドレスの 128 ビット化(IPv4 は 32 ビット)、パケット・ヘッダの簡素化、セキュリティ機能など盛り込んでいる。

IrDA(Infrared Data Association)

赤外線によるパーソナル・データ通信の仕様をまとめた業界団体。

ISA(Industry Standard Architecture)バス

「アイサ・バス」と読む。IBM PC/AT 互換機で一般的な拡張バス。
最高 8.3MHz という制約はあるが、複雑な制御が不要で I/O、メモリ空間が開放されていて、レジスタ制御が容易。

IS LAN(Integrated Service LAN)

音声／データや LAN/ISDN のサービスを統合したマルチメディア・フロア LAN。

ISM 帯(Industrial Scientific and Medical Frequency Band)

スペクトラム拡散方式の無線 LAN で使用される 2.4GHz の周波数帯(2400～2500MHz)。電波雑音の多いバンドだが、ほぼ世界共通で、米国では無許可で使用可能(日本では 10mW 以下に規制)。防犯用無線、電子レンジ、医療メスなど産業科学医療用機器に開放。

ISP(Internet Service Provider)

インターネット・サービス・プロバイダ。
＜関連語＞NSP

ITU(International Telecommunication Union)

国際電子通信連合。

ITU-T 勧告(ITU-T Recommendation)

ITU の電気通信標準化部門 ITU-T(ITU-Telecommunication Standardization Sector:旧 CCITT)が定める勧告。

D シリーズ :料金(D.000～D.306)

F シリーズ :網の運用、サービス(F.1～F.901)

G シリーズ :デジタル伝送網(G.100～G.981)

H シリーズ :非電話番号系伝送回線(H.120～H.331)

I シリーズ :ISDN(I.110～I.751)

Q シリーズ :デジタル交換、共通線信号(Q.50～Q.2971)

T シリーズ :テレマティーク・サービス(T.0～T.661)

V シリーズ :電話網でのデータ通信(V.1～V.80)

X シリーズ :データ通信ネットワーク(X.1～X.903)

IVDLAN(Integrated Voice & Data Local Area Network)

イーサネットLAN等のネットワークに音声、画像データ等のサービスを行うISDNを相互に接続可能としたネットワーク。

＜関連語＞WAN、MAN、LAN

【 J 】

J-TACS(Japan TACS)

米国の AMPS 方式を基本とした、アナログ方式の自動車・携帯電話システム。

日本ではモトローラ方式とも呼び、関西セルラー電話などが採用した。

＜関連語＞AMPS、TACS

【 L 】

LAN(Local Area Network)

「ラン」と読む。オフィスやビルなど狭い区域で、パソコン等をケーブルで結び、データをやり取りするネットワーク。

＜関連語＞WAN、MAN、FDDI

LIN(Local Interconnect Network)

「リン」と読む。車載のモータ制御バスで、インテリジェント・センサやアクチュエータへの適用を考慮した最大 20kbps のネットワーク。UART をベースとした、マスタ・スレーブのアクセス方式。

物理層のドライバに、ISO9141 のトランシーバを拡張して採用。

【 M 】

MAC(Media Accesss Control)

OSI データ・リンク層の下部分で、LAN 媒体(ケーブルや空間)へのデータ送出を制御。

MAN(Metropolitan Area Network)

「マン」と読む。WAN の一種で、IEEE 委員会で IEEE802.6 として規定された通信プロトコル。

＜関連語＞LAN、FDDI、WAN、IVDLAN

MBONE（Multicast Backbone）

「エムボーン」と読む。トンネリングで仮想的にマルチキャストを実現し、インターネットを放送形態で利用する世界的なネットワーク。

MCA（Multi Channel Access）

業務用無線サービスの俗称。通信網は、加入者の設置する指令局（基地局）と移動局、サービス事業者の設置する制御局から成る。マルチプル・アクセスと同じ意味に使われる場合もある。

MIME（Multipurpose Internet Mail Extensions）

US-ASCII 以外の文字、イメージ、音声、動画などを電子メールで扱うための、RFC1521,1522 による規定。

MSK（Minimum Shift Keying）

キャリア（搬送波）の位相の変位により、情報伝送する変調方式の一種。
BPSK では、情報ビットのうち"1"に位相変化"π"、"0"に位相変化"ゼロ"を割り当て、位相変化が大きく急となる。
MSK は、重み付けでこの位相変化を滑らかにしたもので、QPSK よりスペクトルの広がりが少ない。

【 N 】

NFS (Network File System)

RPC によって UDP で動作する LAN 上の、コネクションレス型分散ファイル通信システム。
機種や OS から独立して、リモート・アクセスとローカル・アクセスも同様に行う。構成はステートレス型。
＜関連語＞ステートレス型、RPC、UDP

NMT（Nordic Mobile Telephone）

北欧のアナログ方式の自動車・携帯電話システム。
900MHz 帯の NMT-900 と 450MHz 帯の NMT-450 等のシステムがあり、ノルディック 4 ヶ国間で使用できる。

NRZ（Non-Return To Zero）

通信回線の信号状態について、正レベルで"1"、負レベルで"0"を表すデータの変調方式。
＜関連語＞ベースバンド、NRZI、マンチェスタ

NRZI（Non-Return To Zero Inverse）

データが"0"で通信回線状態のレベルを反転させ、"1"で変化させないデータ変調方式。
＜関連語＞ベースバンド、NRZ、マンチェスタ

NSP（Network Service Provider）

商用インターネット・サービスの提供者。ユーザとの接続形態には、次の3タイプがある。

専用線接続	: 双方の IP ルータを専用線で LAN 間接続
ダイヤルアップ IP 接続	: 電話や ISDN などの公衆交換網で IP 接続
UUCP 接続	: 公衆交換網で UUCP プロトコルを使用して接続

N-TACS（Narrow TACS）

狭帯域 TACS。チャネル間隔を半分に狭帯域化したアナログ方式の自動車・携帯電話システム。
＜関連語＞TACS

【 O 】

OFDM(Orthogonal Frequency Division Multiplexing)

→ 直交波周波数分割多重参照。

OHCI(Open Host Controller Interface)

USB や IEEE1394 をサポートするパソコンのインタフェースの1種。USB では「OpenHCD」と表示される。

＜関連語＞EHCI、UHCI

OSF(Open Software Foundation)

オープン・ソフトウエア財団。HP、IBM、DEC など各社を中心に設立したオープン・システムの製品技術の研究/開発を目的とする非営利団体。サン・マイクロシステムズ社と AT&T 社の提携に対抗して結成。

OSI(Open System Interconnection)

ISO(国際標準化機構)が、通信プロトコルの標準化を目的とした規格。通信機メーカやコンピュータ・メーカが独自の通信プロトコルを使用していたので、異機種間の通信が不可能だったため、IBM、DEC などが通信プロトコルの標準化を図り、機能別に次 7 つの機能グループを分割定義した(OSI 基本参照モデル)。

最上位 ↕ 最下位

レイヤ 7<アプリケーション層(応用層)>	応用プロセス間で通信するための全サービスを提供
レイヤ 6<プレゼンテーション層>	データ構文を共通のプレゼンテーション・イメージ構文に変換
レイヤ 5<セッション層>	優先付けや秩序正しいデータ交換を提供
レイヤ 4<トランスポート層>	順序制御、フロー制御など、透過データの流れを制御
レイヤ 3<ネットワーク層>	透過的で信頼性の高いデータ伝送を行うための制御を提供
レイヤ 2<データ・リンク層>	通信回線の設定、維持、解放などの伝送制御を提供
レイヤ 1<フィジカル層(物理層)>	端末を網に接続する機械的、電気的、機能的な手段を提供

対象分野 :レイヤ 1～4＝データ転送、レイヤ 5～7＝情報処理

機能分類 :レイヤ 1～3＝伝達機能、 レイヤ 4～7＝プロセス間通信

【 P 】

PBX(Private Branch Exchange)

企業などの私設通信網の回線交換を行い、外部の公衆網と接続する装置。

PCI(Peripheral Component Interconnect)バス

パソコンの標準的 32 ビット高速バス。64 ビット CPU のために、64 ビット・バスの規定も含む。

バス・クロックは最大 33MHz で、転送レートは 132M バイト/s(32 ビット・バスのピーク値)。

電源電圧と入出力信号電圧は、5V と 3.3V を規定し、3.3V の多用で消電力化に対応。

またバス・インタフェースを 1 チップ ASIC で実現し、低コスト化にも対応。

PCI バスと他のバスを橋渡しするバス・ブリッジが利用できる(CPU-PCI、ISA／EISA-PCI、PCI-PCI)。

ATM や USB の外部インタフェースにも採用される。

PCM(Pulse Code Modulation)

電話音声信号をアナログ-デジタル変換、デジタル-アナログ変換する場合の、パルス符号変調方式の一種。

項　　目	PCM 方式	
	μ 則（μ-law）	A 則（A-law）
符号器/複合器の近似方式(非線形)	15 折線近似	13 折線近似
PCM フレーム構成	24 タイム・スロット	32 タイム・スロット
PCM データ・クロック周波数	1.544MHz ((8 ビット×24＋1 ビット)×8kHz)	2.048MHz ((8 ビット×32)×8kHz)
準拠規定	ATT(Bell)が提案 ITU-T 標準	CEPT が提案 ITU-T 標準
おもな採用国	米国、 カナダ、 日本、台湾，韓国、香港など	ヨーロッパ、南米など

PCN（Personal Communication Network）

英国が提唱したデジタル携帯電話システム。

半径数百～数キロ・メートルのマイクロセル、準マイクロ波帯（1.8GHz）を使用し、双方向通信可能で GSM 規格に準拠。

＜関連語＞セルラ、GSM

PCS（Personal Communication System）

北米のマイクロセル・タイプの双方向デジタル携帯電話システム。PCN の北米版。

＜関連語＞セルラ、PCN

PDC（Personal Digital Cellular）

800MHz/1.5GHz 帯のデジタル携帯電話方式で、アナログに次ぐ第 2 世代の移動体通信システム。

データ伝送速度は 42kbps、変調方式は QPSK、音声符号化は VSELP（11.2kbps）を採用。

＜関連語＞QPSK、VSELP

PHP（Personal Handyphone）

郵政省電気通信技術審議会で標準化した簡易型携帯電話システム。基本的にデジタル・コードレス電話で、共通化した親機（公衆基地局）を事務所内や屋外に設置し、同一端末でアンテナを設置してある場所ならどこでも利用可能。いわばコードレス電話の公衆サービス版。

PHS（Personal Handyphone System）

パーソナル・ハンディーホン・システム。PHP を利用した移動体通信システム。

PHY（Physical Layer Protocol）

OSI 物理層の上部分で、ステーション間の伝送データのコード化/フレーム処理を規定。

PLMN（Public Land Mobile Network）

陸上移動通信サービスの提供を目的とした、無線回線を含む移動通信専用ネットワーク。

PMD（Physical Layer Medium Dependent）

物理媒体依存サブレイヤ。OSI 物理層の下部分で、FDDI ステーションの媒体要件を規定。

＜関連語＞ATM、UTOPIA

POP(Post Office Protocol)

「ポップ」と読む。電子メールをスプールしているシステムから、TCP/IP でスプール内容を読み出すためのプロトコル。

POS(Point Of Sales)

「ポス」と読む。流通ネットワーク・システムの1つで、流通データをセンタの大型コンピュータで処理するネットワーク。

POTS(Plain Old Telephone Service)

従来のアナログ電話回線サービス、またはこれらのアナログ通信で利用していた 0～4kHz 程度の低い周波数帯域。

PPP(Point to Point Protocol)

公衆回線で IP 接続するためのプロトコル。

PSK(Phase Shift Keying)

位相偏移変調。モデムの変調方式の1つで、キャリア(搬送波)の位相変化によってデータを送る。
おもに、位相を4種類に変化させる 4 相 PSK と、8 種類に変化させる 8 相 PSK が使われる。
＜関連語＞ASK、FSK、QAM

PTT(Ministere des Poste et Telecommunicationes et de la Telediffusion)

郵便電気通信省。日本の総務省郵政事業庁に相当。
特定の国の省庁を示す略語ではなく、フランス、スイスなど欧州で使われる。

【 Q 】

QAM(Quadrature Amplitude Modulation)

直交振幅変調。モデムの変調方式の1つで、キャリア(搬送波)の位相と振幅の変化によってデータを送る。
おもに、位相と振幅を計 16 種類に変化させる 16 相 QAM が使われる。
＜関連語＞ASK、FSK、PSK

QPSK(Quatenary Phase Shift Keying)

四相位相変調。キャリア(搬送波)の位相変化により、情報伝送する変調方式の一種。たとえば情報ビットを 2 ビットずつまとめて"00,01,10,11"のグループ(シンボル)とし、それぞれに割り当てた位相変化で情報伝送する。
＜関連語＞PSK、BPSK

【 R 】

RADSL(Rate Adaptive Asymmetric Digital Subscriber Line)

xDSL の1つで、通信路の状態に応じてデータ伝送速度を動的に変える。
ADSL では距離やノイズなどの影響で伝送速度が変化するが、これを動的に制御して、つねに最高速の伝送をする。
＜類語＞xDSL、ADSL、HDSL、SDSL、VDSL

RAS(Remote Access Service)

Windows NT でのダイヤルアップ・ネットワークの呼称。
＜関連語＞ダイヤルアップ・ネットワーク

RCR(Research & Development Center for Radio Systems)

(財)電波システム開発センター。

RFC(Request for Comments)

インターネット標準化のためのドキュメント。

RPC (Remote Procedure Call)

UDP で通信をするための遠隔手続き呼び出し(関数呼び出し)。フォーマットは XDR。

<関連語>UDP、XDR

RPE-LTP(Regular Pulse Excitation with Long Term Prediction)

音声符号化方式の一種。振幅と位置が異なる複数個の音声信号のパルスを一定間隔で区切り、パルス列(フレーム)をつくる。これにより、パルス位置の検索と、これに関する情報の伝送が不要となり、それだけパルスを多く伝送できる。パルスを APCM で符号化し、これにピッチ予測を加えて音質を向上させた符号化方式。

汎欧州デジタル携帯電話(GSM、PCN)の音声コーデックに使用。

<関連語>コーデック

RS-232C

ITU-T(旧 CCITT)の勧告に基づき、米国 EIA が定めたデータ通信システム用の規格。

直列 2 進データ伝送に関するデータ端末装置とデータ通信装置の間の接続条件を電気的特性、機械的特性、機能について規定した国際的な標準規約。他に RS-422,423 などの規格がある。

RSSI(Received Signal/Strength Indicator)

受信電界強度。

【 S 】

SAR(Segmentation and Reassembly)

AAL でセルを分割/組み立てるサブレーヤ。

SCSI(Small Computer System Interface)

「スカジィ」と読む。ANSI が規格化した、パソコンとパソコン周辺装置間の 8 ビット・パラレル・インタフェース。

SDH(Synchronous Digital Hierarchy)

デジタル伝送路の効率化を図る同期/多重方式のハイアラーキ。送信側と受信側を分離し、異メーカ装置の対向が可能。一般に欧州では SDH、北米では SONET と呼ばれるため、SONET/SDH と表記する。

SDLC(Synchronous Data Link Control)

2 つ以上の装置間でデータ伝送を行う場合の、装置間の論理的な結合の確立と、その解除や転送データの完全性を保証する高水準なデータ・リンク制御手順の1つ。IBM が発表したもので、 HDLC の UNC(Unbalanced Operation Normal Response mode Class:不平衡型正規応答モード・クラス)の基本機能に付加機能を追加したもの。

<類語>HDLC(High Level Data Link Control)

SDSL(Symmetric Digital Subscriber Line)

xDSL の 1 つで、伝送速度を抑えて対称なデータ転送をする。HDSL と類似だが、加入者宅と電話局間での使用を想定。POTS スプリッタにより、従来のアナログ電話回線サービス(POTS)も利用できる。

<類語>xDSL、ADSL、HDSL、RADSL、VDSL

SERDES（Serialize Deserialize）

「サーデス」と読む。ATM などでの「シリアル-パラレル」間の高速データ変換。
クロック・リカバリ機能など、周辺機能も含めて呼ぶこともある。

SMTP（Simple Mail Transfer Protocol）

電子メールを送信するためのプロトコルで、RFC821 などで定義。

SNA（Systems Network Architecture）

SDLC を採用した、IBM 社のネットワーク・アーキテクチャ。

SONET（Synchronous Optical Network）

同期光通信網。SDH を採用した米国の高速専用線サービス。

STS-1	:52Mbps
STS-3	:156Mbps
STS-12	:622Mbps
STS-24	:1244Mbps
STS-48	:2488Mbps （STS：Synchronous Transport Signal）

SSMA（Spread Spectrum Multiple Access）

スペクトラム拡散多元接続。
＜同義語＞CDMA（Code Division Multiple Access）

STB（Set-top Box）

→ セットトップ・ボックス参照。

STM（synchronous transfer mode）

同期転送モード。連続情報の伝送に適する多重伝送方式。
伝送路を一定長のフレームに分割し、各フレームを伝送速度に応じてさらにタイム・スロットに分割する。
たとえば、1.5Mbps の伝送路で情報速度が 64kbps の場合、1 フレームを 24 のタイム・スロットで構成する。
一般に電話やファックスなどのデジタル回線交換に用いられるため、回線型転送モードとも呼ばれる。
＜関連語＞ATM

【 T 】

TA（Terminal Adaptor）

ISDN や xDSL のインタフェースを内蔵していない端末（TE2）を網終端装置 DSU（NT1）やデジタル PBX（NT2）に接続するための端末アダプタ装置。

TACS（Total Access Communication System）

英国のアナログ方式の自動車・携帯電話システム。米国の AMPS 方式の変形版。

TC（Transmission Convergence）

伝送コンバージェンス・サブレイヤ。OSI 物理層の上部分で、LAN 個別の伝送媒体の特性を吸収する。
＜関連語＞ATM、UTOPIA

TCM（Trellis Coded Modulation）

モデムの変調方式の1つで、送信データに冗長ビットを加えて、耐ノイズ性を向上。キャリア信号の位相と振幅を変化させる点は QAM と同様。おもに、32 点 TCM、64 点 TCM、128 点 TCM が使われる。

TCP（Transmission Control Protocol）

IP 層の上で、信頼性のある伝送をするコネクション型のトランスポート層プロトコル

TCP/IP（Transmission Control Protocol/Internet Protocol）

RFC でまとめた LAN のレイヤ 3、4 のプロトコルの一例で、インターネット標準。
米国の各種ワークステーションなどでも採用されている。

TDD（Time Division Duplex）

時分割双方向。同一通信路（有線は電線、無線は特定周波数の電波）を一定時間で区切り、交互に送受信する通信方式。ピンポン伝送方式とも言う。1通信路で実現できるが、区切り（フレーム）を検出して同期をとる必要がある。

Duplex　：同時に送受信
Simplex　：交互に送受信（Press To Talk）

TDM（Time Division Multiplex）

→ 時分割多重方式参照。

TDMA（Time Division Multiple Access）

時分割多元接続。TDM による多重通信アクセス。

TDSW（Time Division Switch）

「ティー・ディー・スイッチ」と読む。デジタル回線網における時分割タイム・スイッチ。たとえば、PCM のような時分割多重化された回線間で、個々のタイム・スロット（チャネル）ごとの切り換え接続を行うスイッチ回路網。

＜関連語＞PCM、タイム・スロット

TE1（Terminal Equipment 1）

ISDN インタフェースを内蔵した ISDN 電話機、G4 FAX などの端末装置。

TE2（Terminal Equipment 2）

ISDN インタフェースを内蔵しない既存端末。

TIA（Telecommunications Industry Association）

米国電気通信工業会。

TVRO（Television Receiver Only）

米国で運用されているテレビジョン受信オンリー（通信衛星を使った TV 放送）。DBS との違いは、放送専用の衛星を使えないこと。BS の出力は TVRO の約5倍（100W）もある。

【 U 】

UDP (User Datagram Protocol)

NFS などで、データを転送するだけのコネクションレス型プロトコル。

＜関連語＞NFS

UHCI(Universal Host Controller Interface)

USB をサポートするパソコンの、インタフェースの1種。

＜関連語＞EHCI、OHCI

UNII(Unlicensed National Information Infrastructure)帯

米国で無免許利用が認可されている 5GHz の周波数帯。5.15～5.25GHz(200mW)、5.25～5.35GHz(1W)、5.725～5.825GHz(4W)の 3 バンド(計 300MHz)。日本では気象レーダなどの帯域と重複し、無免許の特定小電力無線で利用できるのは 5.15～5.25GHz(100MHz)。空中線電力 10mW 以下で屋内使用に限定。

URL(Uniform Resource Locator)

HTTP でリソースを示す参照規約 URI(Uniform Resource Identifier)で「名前」URN(Uniform Resource Name)とともに定義される「場所」のこと。

USART(Universal Synchronous Asynchronous Receiver Transmitter)

アシンクロナス(調歩同期)方式とシンクロナス(同期)方式のどちらの通信プロトコルも可能とする通信制御回路。
UART はアシンクロナス方式だけ。

USB(Universal Serial Bus)

当初はパソコンと周辺機器のマルチメディアを目的としたインタフェースで、パソコン以外の装置にも広まった。
プラグ＆プレイで機器を認識するホット・プラグ(作動中の着脱)に対応。周辺装置を最大 127 台まで接続可能。

ロウ・スピード　:1.5Mbps
フル・スピード　:12Mbps
ハイ・スピード　:480Mbps(USB2.0 だけ)

USB-IF(USB Implementers Forum)

USB 仕様の標準化委員会。ルネサス エレクトロニクス、Intel、Microsoft、Compaq などがコア・メンバ。

UTOPIA(Universal Test and Operation PHY Interface for ATM)

「ユートピア」と読む。ATM フォーラムが標準化している ATM レイヤ-物理レイヤ間のインタフェース。
物理レイヤを伝送コンバージェンス(TC)サブレイヤと物理媒体依存(PMD)サブレイヤに分け、 PMD で個別伝送媒体に対応し、TC で個別の特性を吸収する。これにより LAN の特性差が上位 ATM レイヤには見えず、物理レイヤに接続される LAN が ATM レイヤに影響を及ぼさないシームレスな通信が可能。

UUCP(Unix to Unix Copy)

電話回線などのシリアル回線を使用した、バッチ・ファイル転送プロトコル。

【 V 】

V シリーズ勧告(V Series Recommendation)

電話回線を使ったデータ通信について、ITU-T 国際標準の勧告(V.1～V.80)。モデムの変調方式は V シリーズ勧告。

V.110 :V シリーズ端末を ISDN に接続するのに必要な速度変換則とプロトコル変換。モデム置換に適合し、システム設計が容易。

V.120 :V シリーズ端末を ISDN に接続するのに必要な速度変換則とプロトコル変換(統計的多重法)。無手順端末の接続で、伝送高信頼性を得る。

VAN(Value-Added Network)

NTT などの第一種電気通信事業者から一括に通信回線を借り、コンピュータと接続して付加価値をつけ、利用者に通信サービスを行う業務。

VDSL(Very High Speed Digital Subscriber Line)

xDSL の1つで、接続距離を限定して最大伝送速度を約 50Mbps に高速化。大容量回線を約 1.5km までの距離に使用。

＜類語＞xDSL、ADSL、HDSL、RADSL、SDSL

VSAT(Very Small Aperture Terminal)

超小型衛星局。一般の衛星通信が 20～40m の巨大なパラボラアンテナを使用するのに比べ、 VSAT は 1.2～1.8m の小さなアンテナで衛星通信が可能。このため運用コストが少なく、企業内通信や放送局の中継用として普及した。

VSELP(Vector Sum Exited Liner Predictive)

「ブイ・セルプ」と読む。音声符号化方式の一種。音源コードブックの探索にベクトルを導入し、CELP 方式の欠点だった膨大な演算量を低減する。モトローラ社がこの方式の特許を所有。

日本と米国のデジタル携帯電話機の音声コーデックに採用されている。

＜関連語＞CELP、コーデック

【 W 】

WAN(Wide Area Network)

「ワン」と読む。LAN よりさらに広大なエリア内の端末間のデータ伝送をサポートする通信網。

＜関連語＞LAN、MAN、FDDI、IVDLAN

W-CDMA(Wide-band CDMA)

広帯域・符号分割多元接続方式。スペクトラム拡散方式に直接拡散(Direct Sequence:DS)を採用し、CDMA 方式の特長に加え、電力制御やパス・ダイバシティ受信、干渉除去、誤り制御、音声符号化の信頼性向上を行う。

＜関連語＞IMT-2000、FOMA、cdmaOne、ダイバシティ受信

Web

World Wide Web(WWW)の略称。

Web サイト(Website)

Web でサービスを提供するコンピュータや、それを運営する団体。

Web ページ

Web ブラウザで表示する各コンテンツ。

Web マスタ(Webmaster)

Web サーバの管理者。

WWW(World Wide Web)

世界的な情報網(グローバル・ネットワーク)。

【 X 】

XDR (External Data Representation)

外部データ表現のための、CPU アーキテクチャに依存しないライブラリ・ルーチン。

＜関連語＞RPC

xDSL(Digital Subscriber Line)

既存の電話回線を利用した、高速デジタル・データ通信技術の総称。

＜類語＞ADSL、HDSL、RADSL、SDSL、VDSL

X シリーズ勧告(X Series Recommendation)

ITU-T の国際標準のうち、データ通信網に関してまとめた勧告(X.1～X.903)。おもな勧告は次のとおり。

X.3 :公衆データ網での、パケット組み立て/分解

X.21 :公衆データ網での、同期式の DTE-DCE 間インタフェース

X.25 :公衆データ網に専用線で接続したパケット・モード端末の、DTE-DCE 間インタフェース

X.36 :公衆データ網に専用線で接続したフレーム・リレー・データ伝送の、DTE-DCE 間インタフェース

X.140 :公衆データ網を介す通信のサービス品質

XNS(Xerox Network System)

ゼロックス社の分散ネットワーク・アークテクチャ。

X/Open

UNIX 関連仕様の共通化を進める民間団体。

【 数字 】

0 挿入／0 除去(Zero Insertion and Zero Deletion)

HDLC コントローラが、フラグ(01111110)と通常データを区別するため、送信データに対して"0"を挿入、受信データに対して"0"を除去する動作。

0 挿入では、データ中に 5 ビットの"1"が連続すると、その次に1ビットの"0"を強制的に挿入する。

0 除去ではその逆で、5 ビットの"1"が連続し、その次が"0"なら、それを1ビット除去する。

＜関連語＞HDLC、SDLC

10BASE2/5/-T

10 はイーサネットがサポートする転送速度 10Mbps し、BASE はデータを変調せずに送るベース・バンド方式を表す。また、2/5/-T はケーブルで、T はツイスト・ペア、2 は最大ケーブル長 200m、5 は最大ケーブル長 500m を表す。

100BASE-T

100Mbps のイーサネット(Fast Ethernet)。

T はツイスト・ペアで、ケーブルの種類によって、100BASE-TX、100BASE-T4、100BASE-FX の 3 種類がある。

★ 1000BASE-T

1Gbps のイーサネット(Giga-bit Ethernet)。

T はツイスト・ペア。

FAQ（よくあるお問い合わせ） インフォメーション
ルネサス エレクトロニクスの半導体製品

発行年月日　2006 年 11 月 1 日 Rev.1.00
2010 年 12 月 22 日 Rev.4.00

発行　ルネサス エレクトロニクス株式会社
〒211-8668 神奈川県川崎市中原区下沼部 1753

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   標準水準： コンピュータ、OA機器、通信機器、計測機器、AV機器、家電、工作機械、パーソナル機器、産業用ロボット

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせないようお客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制するRoHS指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

注1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注2. 本資料において使用されている「当社製品」とは、注1において定義された当社の開発、製造製品をいいます。

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口 http://www.renesas.com

※営業お問合せ窓口の住所・電話番号は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

ルネサス エレクトロニクス販売株式会社　〒100-0004　千代田区大手町2-6-2（日本ビル） (03)5201-5307

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：http://japan.renesas.com/inquiry

© 2010 Renesas Electronics Corporation. All rights reserved.

Colophon 1.0

FAQ（よくあるお問い合わせ）

ルネサス エレクトロニクス株式会社

R00ZZ0001JJ0400
(旧番号：X18437JJ3V0IF00)