

お客様各位

ZUD-CD-07-0063

1/111

2007年3月30日

NECエレクトロニクス株式会社

第四システム事業本部

汎用マイコンシステム事業部

開発ツールグループ

チームマネージャー 安藤 喜成

(担当：鈴木 康之)

CP (K), 0



# 78K0/KF2ターゲット・ボード QB-78K0KF2-TB チュートリアル・ガイド

# ごあいさつ

QB-78K0KF2-TBをお買い求めいただき、誠にありがとうございます。

本製品(QB-78K0KF2-TB)は、NECエレクトロニクス社製のプログラミング機能付きオンチップ・デバッグ・エミュレータQB-MINI2(以降MINICUBE2)を使用して、マイコンを実際に試すためのターゲット・ボードです。

チュートリアル・ガイドでは、本製品とMINICUBE2を使用した開発環境をわかりやすく説明します。サンプル・プログラム、ターゲット・システム回路例を用いて説明していますので、熟読して頂ければマイコン開発の基本が習得できます。本製品と共に活用して下さい。

初心者

中級者

上級者



みんなまとめて  
チュートリアル・ガイドにおまかせ!!

本製品に関する最新情報、必要な無償版開発ツール、およびサンプルプログラムは、弊社webサイトにて提供しています。

[MINICUBE2関連ツール]

<http://www.necel.com/micro/ja/development/asia/minicube2/minicube2.html>

[無償版開発ツール]

<http://www.necel.com/micro/freesoft/>

[サンプル・プログラム]

[http://www.necel.com/micro/ja/development/asia/minicube2/minicube2\\_opt.html](http://www.necel.com/micro/ja/development/asia/minicube2/minicube2_opt.html)

# 本書の見方(説明)



本書では、わかりやすくするため下記の構成になっています。



## [見出し]

本書の章に相当します。太い緑の線で囲まれたところは、現在の章を表します。また、薄い緑の線で囲まれたところは、他の章を表していますので本書全体の構成が、どのページでも一目でわかるようになっています。

## [表題]

章の中で伝えたいことをいくつかの項目に分けています。本書中では「xxページを参照して下さい」という表現はしません。例えば「[はじめに]の[本書の見方]を参照して下さい」という表現になります。

## [アイコンガイド]

アイコンガイドで、そのページ内容を一目で分かるようにアイコンで表現しています。後で読めばいいのか、必ず読む必要があるのか、このアイコンガイドを参考にして下さい。アイコンガイドには、下記の種類があります。



各章の学習する時間の目安を示します。単位は分です。



注意する内容です。



必ず読む必要のある内容です。



資料として使う内容です。

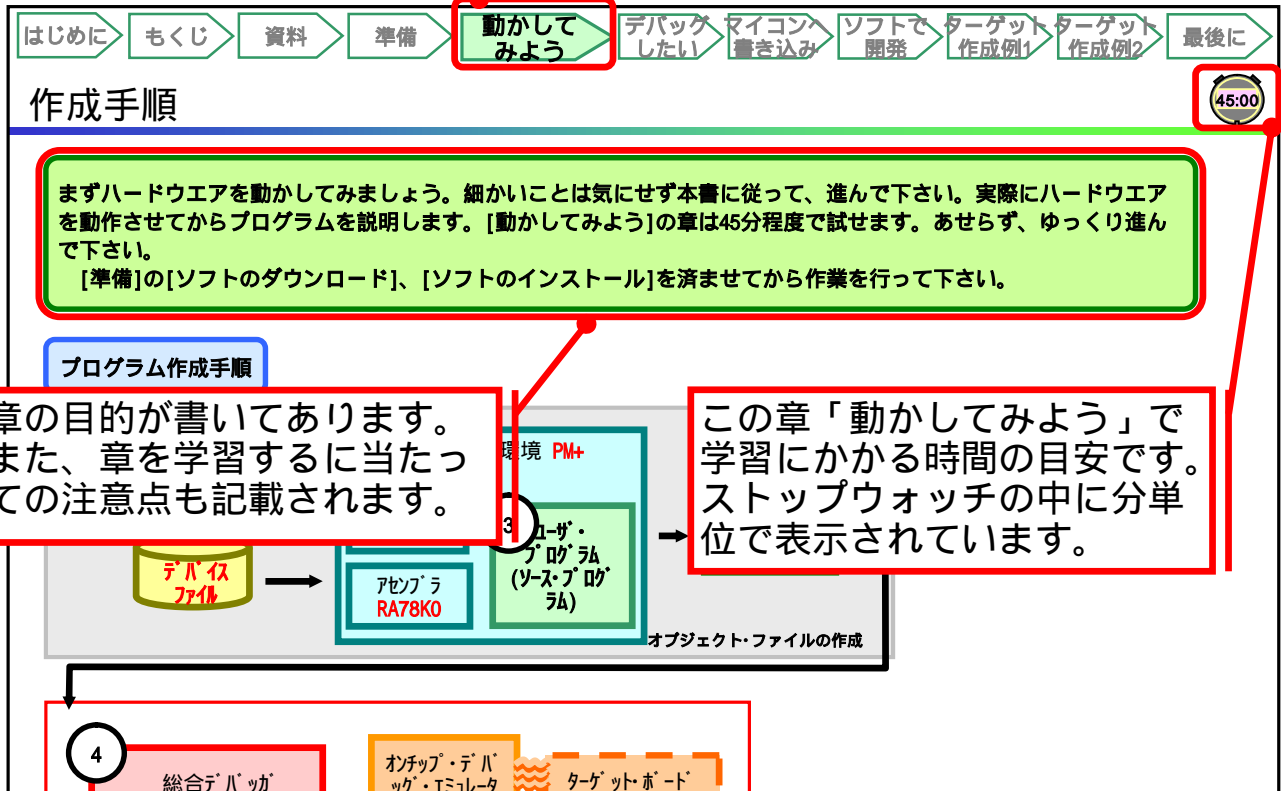


コラム・ワンポイントなど、関連する事についての内容です。

# 本書の見方(例)

前ページ[本書の見方(説明)]に沿った例を示します。

章を現しています。



目次の見方について説明します。

対象とする読者によって色分けしてあります。

- ・初心者向けゾーンでは、まず本製品QB-78K0KF2-TBを動作させることを目的としています。また、後半ではサンプル・プログラムの内容を説明しています。
- ・中級者向けゾーンでは、開発の基本を理解することを目的としています。デバッグ方法の基本と、高度なデバッグについて説明しています。
- ・上級者向けゾーンでは、応用例としてドットマトリクスLEDを使った電光掲示板を作成し、更なる開発の手がかりを示しています。



## ワンポイント

ワンポイントについて

そのページに関係する追加情報を載せています。また、深い内容についてはヒントになる語句について記しています。このワンポイントにも注目して下さい。

# 目次

はじめに	ごあいさつ - - - - -	2
	本書の見方(説明) - - - - -	3
	本書の見方(例) - - - - -	4

もくじ	目次 - - - - -	5
-----	--------------	---

資料	QB-78K0KF2-TBの特徴 - - - - -	7
	部品配置図 - - - - -	8
	コネクタ情報(CN1) - - - - -	9
	コネクタ情報(CN2) - - - - -	10
	コネクタ情報(16pinヘッダ) - - - - -	11

準備	ソフトウェアのダウンロード - - - - -	12
	ソフトウェアのインストール - - - - -	13
	開発環境について - - - - -	18
	進化したApplilet2 - - - - -	19
	システム構成図 - - - - -	20

動かしてみよう	作成手順 - - - - -	21
	Applilet2でプロジェクトを作成 - - - - -	22
	周辺機能設定[システム] - - - - -	24
	周辺機能設定[割り込み] - - - - -	25
	周辺機能設定[ポート] - - - - -	26
	周辺機能設定[タイマ] - - - - -	27
	コードの生成 - - - - -	28
	プロジェクトの保存 - - - - -	29
	PM+の起動 - - - - -	30
	プログラムの編集 - - - - -	31
	プログラムのビルド - - - - -	34
	MINICUBE2との接続 - - - - -	35
	デバッガの起動 - - - - -	36
	プログラムの実行 - - - - -	37
	プログラムの説明 - - - - -	38
	プログラムの作り方 - - - - -	43

初心者向けゾーン

デバッグしたい	デバッグの基本 - - - - -	47
	デバッガの基本画面 - - - - -	48
	ブレークポイントの設定 - - - - -	50
	ステップ実行 - - - - -	51
	変数表示 - - - - -	52
	メモリ表示 - - - - -	55
	SFR表示 - - - - -	56
	メモリマップ - - - - -	57
マイコンへ書き込み	プログラミングについて - - - - -	58
	HEXファイル作成 - - - - -	59
	QBP(QB-Programmer)の起動 - - - - -	60
	パラメータファイル読み込み - - - - -	61
	HEXファイル読み込み - - - - -	62
	マイコンへプログラミング - - - - -	63
	動作確認 - - - - -	64

中級者向けゾーン

# 目次

<b>ソフトで開発</b>	システム・シミュレータ (SM+) とは - - - - - 65	
	環境の確認 - - - - - 66	
	SM+の起動 - - - - - 67	
	SM+の基本画面 - - - - - 68	
	入出力パネルの設定 - - - - - 69	
	実行画面 - - - - - 72	
	SM+でのデバッグ - - - - - 73	
	更に高度なデバッグ機能 - - - - - 75	中級者向けゾーン
<b>ターゲット作成例1</b>	電光掲示板を作る - - - - - 81	
	回路図 - - - - - 82	
	回路説明 - - - - - 83	
	プログラム設計 - - - - - 84	
	プログラムリスト - - - - - 87	
	プログラム説明 - - - - - 95	
	動作概要 - - - - - 96	
<b>ターゲット作成例2</b>	ゲームを作る - - - - - 98	
	プログラム設計 - - - - - 99	
	プログラムリスト - - - - - 100	
	プログラム説明 - - - - - 109	
<b>最後に</b>	迷路のデータ作成方法 - - - - - 110	
	最後に - - - - - 111	上級者向けゾーン

付録 QB-78K0KF2-TB 回路図

# QB-78K0KF2-TBの特徴



## 78K0/KF2ターゲット・ボード(QB-78K0KF2-TB)の特徴

- 78K0/KF2(  $\mu$  PD78F0547DGC)搭載
- システム・クロック20MHz(発振子を搭載)で高速動作可能(4.0V ~ 5.5V供給時)
- フラッシュメモリ:128KB、高速RAM:1KB、拡張RAM:6KBを内蔵
- 最大で71本のI/Oポートを装備
- プログラミング、オンチップ・デバッグ(OCD1A, OCD1B端子使用)に両対応
- LED2個、SW1個を搭載しており簡単なテストが可能
- ユニバーサル・エリア(2.54mmピッチ)を搭載
- マイコンの端子を周辺ボード・コネクタに配置した高拡張性鉛(Pb)フリー対応品

## 78K0/KF2ターゲット・ボード(QB-78K0KF2-TB)のハードウェア仕様

CPU $\mu$ PD78F0547DGC	メイン・クロック 動作周波数	20MHz(ボード上に搭載)
搭載部品	CN1, CN2:周辺ボードコネクタ(2.54mmピッチ) 40pinソケットx2(パットのみ)	
	FP1:16pinコネクタ(MINICUBE2接続用)	
	PowerLED:LED赤x1(LED3)	
	評価用LED:LED黄x2(LED1はP76, LED2はP77へ接続)	
	評価用SW:SW1(INTPOへ接続)	
	Y1:20MHz発振子(X1, X2へ接続)	
動作電圧	4.0V ~ 5.5V(Y1:20MHz発振子使用時)	

### ワンポイント

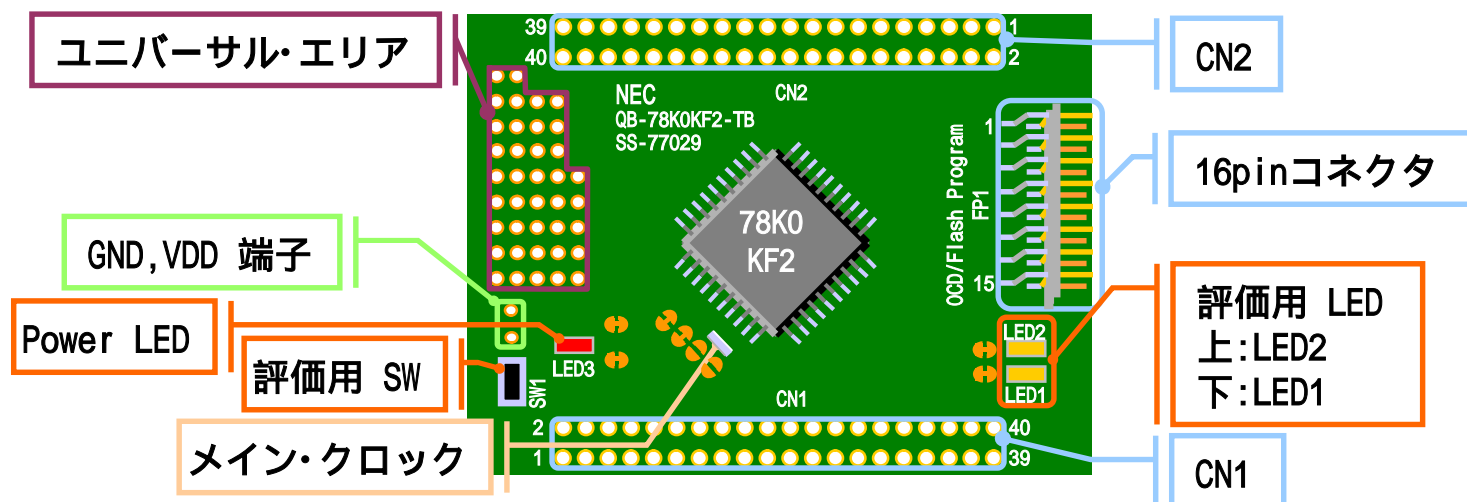
#### マイコンについて

ここで使うマイコンとはマイクロコントローラ(マイクロコンピュータ)の意味です。現在のマイコンはROM, RAM, I/OだけでなくA/D, D/A, UART, I2C, LIN, CAN, LCD制御, USB, DMAなど様々な機能をもったマイコンもあります。また、性能も数MIPS ~ 数百MIPSまで揃っています。

# 部品配置図



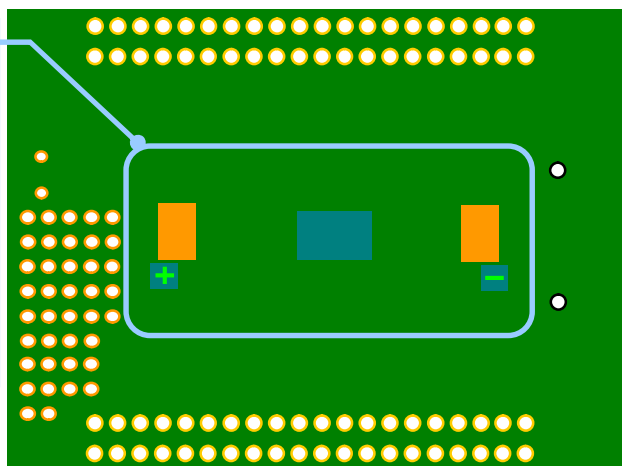
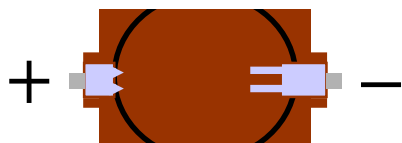
## ターゲット・ボード表





- CN1/CN2: マイコンの端子へ接続されています
- LED3(PowerLED): 電源が入った時に赤色に発光します
- 評価用LED1: ポート76(P76)がLOWで黄色に発光します
- 評価用LED2: ポート77(P77)がLOWで黄色に発光します
- 評価用SW1: INTPOに接続されています
- FP1(16pinコネクタ): オンチップ・デバッグや書き込み時に使用します  
MINICUBE2(別売)を接続します
- Y1(メインクロック): 20MHz発振子を搭載しています
- ユニバーサル・エリア: ユーザーが部品を載せられるエリアです
- GND, VDD端子: ターゲット・ボードへ電源供給する場合の端子です

## ターゲット・ボード裏

ボタン電池ホルダパッド  
ここにCR2032などの電池  
ホルダを載せます。



### ・基板上のパターン について

パターンをカットすることで、その回路はオープンとなります。   
再度ショートさせたい場合は半田ショートさせてください。   
P76, P77を使用する場合はLEDの左隣のショートパッドをパターンカットしてください。  
P121, P122を使用する場合はY1(20MHz発振子)を除去して半田ショートさせてください。



# コネクタ情報(CN1)



ピン番号	接続先 CPU端子	備考	ピン番号	接続先 CPU端子	備考
1	P120/INTPO/EXLVI	SW1にも接続	2	P47	
3	P46		4	P45	
5	P44		6	P43	
7	P42		8	P41	
9	P40		10	T_RESET	16pinコネクタ15へ接続
11	P124/XT2/EXCLKS	10K プルダウン	12	P123/XT1	10K プルダウン
13	FLMD0	16pinコネクタ14へ接続	14	P122/X2/EXCLK/OCD0B	デフォルトはマイコン未接続
15	P121/X1/OCD0A	デフォルトはマイコン未接続	16	NC	
17	GND		18	GND	
19	VDD		20	EVDD	
21	P60/SCL0		22	P61/SDA0	
23	P62/EXSCL0		24	P63	
25	P33/TI51/T051/INTP4		26	P64	
27	P65		28	P66	
29	P67		30	P77/KR7	LED2へ接続
31	P76/KR6	LED1へ接続	32	P75/KR5	
33	P74/KR4		34	P73/KR3	
35	P72/KR2		36	P71/KR1	
37	P70/KR0		38	P06/TI011/T001	
39	P05/TI001/SSI11		40	P32/INTP3/OCD1B	16pinコネクタ13へ接続

( 詳細は付録の回路図を参照して下さい )

# コネクタ情報(CN2)



ピン番号	接続先 CPU端子	備考	ピン番号	接続先 CPU端子	備考
1	P31/INTP2/OCD1A	16pinコネクタ9へ接続	2	P50	
3	P51		4	P52	
5	P53		6	P30/INTP1	
7	P17/TI50/T050		8	P16/TOH1/INTP5	
9	P15/TOH0		10	P14/RXD6	16pinコネクタ5へ接続
11	P13/TXD6	16pinコネクタ3へ接続	12	P12/S010	
13	P11/SI10/RXD0		14	P10/SCK10/TXD0	
15	P54		16	P55	
17	P56		18	P57	
19	AVREF		20	AVSS	
21	P27/ANI7		22	P26/ANI6	
23	P25/ANI5		24	P24/ANI4	
25	P23/ANI3		26	P22/ANI2	
27	P21/ANI1		28	P20/ANI0	
29	P130		30	P04/SCK11	
31	P03/SI11		32	P02/S011	
33	P01/TI010/T000		34	P00/TI000	
35	P145/STB0		36	P144/SOA0	
37	P143/SIA0		38	P142/SCKA0	
39	P141/BUZ/BUSY0/INTP7		40	P140/PCL/INTP6	

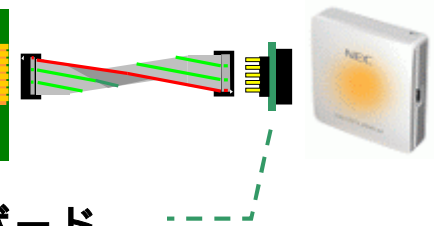
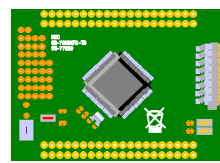
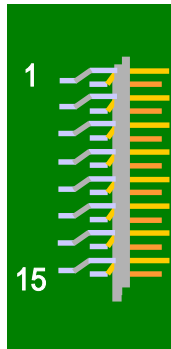
( 詳細は付録の回路図を参照して下さい )

# コネクタ情報(16pinヘッダ)



## 16pinヘッダピンアサイン(デバッグ時に扱う信号)

ピン番号	信号	ピン番号	信号
1	GND	2	RESET_OUT
3	R.F.U.	4	VDD
5	R.F.U.	6	R.F.U.
7	R.F.U.	8	R.F.U.
9	CLK	10	R.F.U.
11	R.F.U.	12	R.F.U.
13	DATA	14	FLMDO
15	RESET_IN	16	R.F.U.

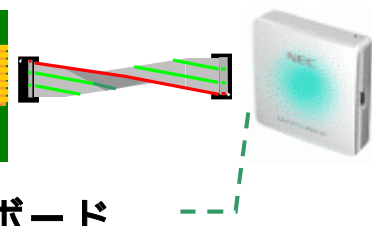
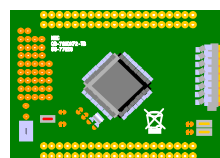
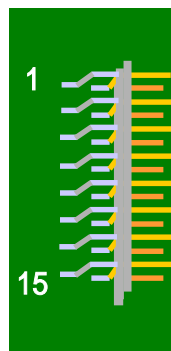


クロック・ボード  
(78K0-OCDボード)を  
接続します

R.F.U.は予約端子のためターゲット・ボード側  
でオープンになっています

## 16pinヘッダピンアサイン(プログラミング時に扱う信号)

ピン番号	信号	ピン番号	信号
1	GND	2	RESET_OUT
3	TXD	4	VDD
5	RXD	6	R.F.U.
7	R.F.U.	8	R.F.U.
9	R.F.U.	10	R.F.U.
11	-----	12	R.F.U.
13	R.F.U.	14	FLMDO
15	R.F.U.	16	R.F.U.



クロック・ボード  
(78K0-OCDボード)は  
接続しません

# ソフトのダウンロード



本書では、下記のソフトウェアをインストールする必要があります。

- ・ [MINICUBE2関連ツール]
- ・ [フリーツール]

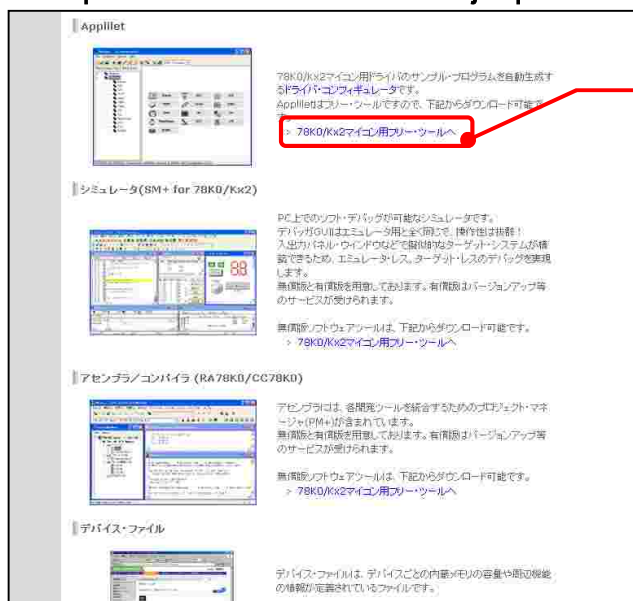
[MINICUBE2関連ツール]についてはMINICUBE2に添付している1枚マニュアルに従ってインストールして下さい。

[フリーツール]のインストールについて、これから説明します。

## ソフトウェアをダウンロードします

下記URLへアクセスして下さい。

<http://www.necel.com/micro/ja/promotion/kx2/>



「78K0/Kx2マイコン用フリー・ツールへ」のリンクをクリックします。

フリーツールをダウンロードする時にユーザー登録します(メールアドレスの登録)。登録したメールアドレスにProduct IDが送られます。このProduct IDは開発ツールソフトウェアのインストールに必要ですので、忘れないようにメモして下さい。

### ダウンロードするソフトウェア一覧

- ・ **RA78K0**  
統合開発環境PM+と78K0用アセンブラを含めたパッケージです。
- ・ **CC78K0**  
78K0用Cコンパイラです。
- ・ **SM+ for 78K0/Kx2**  
78K0用のソフトウェア・シミュレータ(SM+)です。ハードウェアを必要としない、パソコン上で78K0マイコンのデバッグ/テストができます。
- ・ **AppIilet2 for 78K0/Kx2**  
マイコンの初期プログラムを自動生成するツールです。これで作成したプロジェクトは統合開発環境PM+へ読み込み可能です。
- ・ **78K0/Kx2用デバイス・ファイル(DF780547)**  
マイコンの品種ごと、または同系列品種のグループごとに用意された、機種依存情報を持つバイナリ・ファイルです。コンパイラやアセンブラで使います。
- ・ **78K0/KF2用パラメータ・ファイル(PRM78F0547)**  
マイコンへプログラミングを行うときに必要なファイルです。

フリーツールの場合、作成可能なオブジェクトのサイズは32KBの制限があります。



# ソフトウェアのインストール

## ソフトウェアをインストールします

### RA78K0のインストール

ダウンロードしたファイル[RA78K0\_w380\_j.exe]をダブル・クリックし、インストーラを起動します。  
[インストール]を押下し、ツールのインストールを開始します。

**1** インストーラを起動します。

**2** [インストール]を押下します。

**3** [OK]を押下し、[Product ID]入力画面へ遷移します。

**4** 内容を確認して[はい]を押下します。

**5** Product IDを入力し、[次へ]を押下すると、インストールが開始されます。

**6** インストールが終了します。

# ソフトウェアのインストール



## CC78K0のインストール

ダウンロードしたファイル[CC78K0\_w370\_j.exe]をダブル・クリックし、インストーラを起動します。  
[インストール]を押下し、ツールのインストールを開始します。

**1** インストーラを起動します。

**2** [インストール]を押下します。

**3** [OK]を押下し、[Product ID] 入力画面へ遷移します。

**4** 内容を確認して[はい]を押下します。

**5** Product IDを入力し、[次へ]を押下すると、インストールが開始されます。

**6** インストールが終了します。

# ソフトウェアのインストール



## 78K0/KF2用デバイス・ファイル(DF780547) のインストール

[スタート] [プログラム(P)] [NEC Tools32] [デバイスファイル インストーラ]を起動します。  
 [インストール]を押下して、インストール情報ファイルを指定します。指定するフォルダは  
 「df780547\_v210.exe」を解凍したフォルダで、「NECSETUP.INI」を選択します。

**① インストーラを起動します。**

**② [インストール]を押下します。**

**③ [参照]を押下し、解凍したフォルダの「NECSETUP.INI」を選択します。**

**④ [同意する]を押下し、次画面へ進みます。**

**⑤ インストール先をデフォルトのフォルダとします。**

**⑥ インストールが完了します。**

**ソフトウェア使用許諾**

NECエレクトロニクス株式会社(以下「弊社」といいます。)は、以下に記載したすべての条件を承諾され、かつ遵守していただけるお客様に対し、本プログラム・プロダクト(以下「本プログラム」といいます。)の使用を許諾致します。

本プログラムを使用した場合、弊社はお客様が下記条項に同意されたものとさせていただきますので、ご使用前に十分にお読み下さい。

◇第1条(使用権)  
 お客様に設定される使用権とは、本契約とともに提供される本プログラムをご購入またはオンライン・デリバリーシステム等により正式に入手されたお客様に限り、本契約に定める条件に従って本プログラムを使用する権利です。

◇第2条(目的)

◇第3条(禁止事項)

**インストールウィザード：インストール先の指定**

共通デバイスファイルのインストール先を指定してください。

C:\NECTools32\DEV

# ソフトウェアのインストール



## SM+ for 78K0/Kx2のインストール

ダウンロードしたファイル[sm+for78k0\_kx2\_w1x0\_j.exe]をダブル・クリックし、インストーラを起動します。[インストール]を押下し、ツールのインストールを開始します。

**1** インストーラを起動します。

**2** [インストール]を押下します。

**3** [OK]を押下し、使用許諾画面へ遷移します。

**4** 使用許諾を読んでから、[はい]を押下し、[Product ID]入力画面へ遷移します。

**5** Product IDを入力し、[次へ]を押下すると、インストールが開始されます。

**6** インストールが完了します。



# ソフトウェアのインストール



## Applilet2 for 78K0/Kx2のインストール

Applilet2 for 78K0/Kx2は、マイクロソフトの「.NET Framework Version 2.0」をインストールする必要があります。インストールするソフトウェアは、「.NET Framework Version 2.0」ランタイム版で構いません。

ダウンロードしたファイルを解凍し、[setup.exe]をダブル・クリックし、セットアップする言語[日本語]を選択して、インストールを開始します。

**1** 言語を選択します。

**2** [OK]を押下します。

**3** [次へ]を押下し、使用許諾画面へ遷移します。

**4** 使用許諾を読んでから、[はい]を押下し、次画面へ進みます。

**5** [標準]にチェックし、[次へ]を押下します。

**6** プログラムフォルダに変更がなければ[次へ]を押下します。

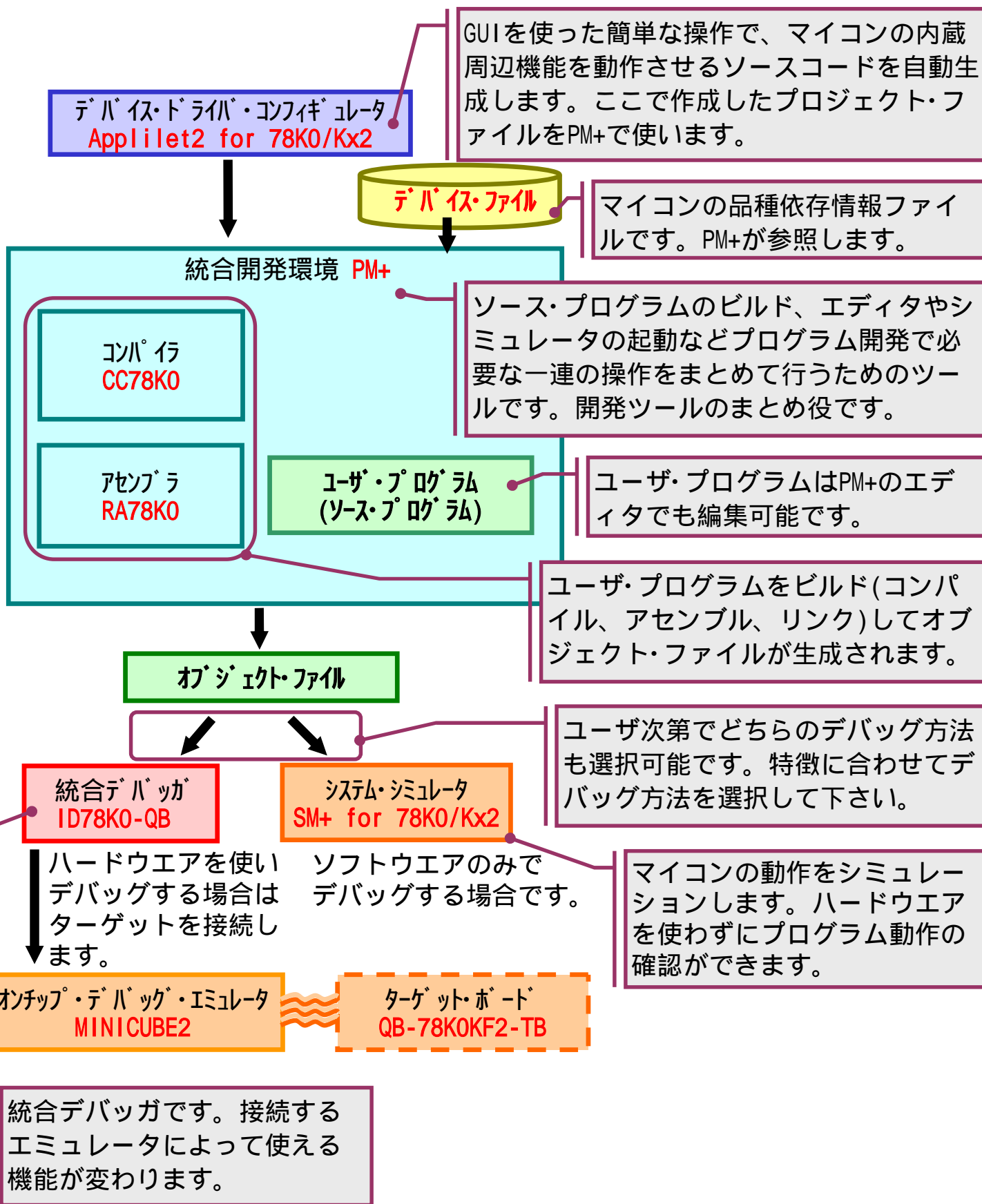
**7** 内容を確認して、[次へ]を押下して、インストールします。

**8** インストールが完了します。



# 開発環境について

NECエレクトロニクスの開発ツール環境の全体図を示します。





# 進化したApplilet2

Appliletの機能が進化して、Applilet2になりました。Appliletは下記の特徴があります。

- ・ GUIでソースを自動生成
- ・ 共通のAPIを提供
- ・ 資源の競合の自動チェック
- ・ クロックの変更による、ポーレートや周波数の自動計算

Applilet2では、下記の新機能が加わりました。

- ・ ソースコードのガード機能
- ・ API名を自由に変更可能
- ・ 設定した内容のドキュメント化

## ソースコードのガード機能

プログラムを作成する時、今までのAppliletでは [xxxx\_user.c]のファイルを修正することを前提としていました。しかし、Appliletでコード生成を行うと[xxxx\_user.c]のファイルを上書きするか、そのまま残すかありませんでした。Applilet2では、下記のコメント間にプログラムを書けば、どのファイルでも上書きされる心配はありません。例えばタイマ値、割り込みを増やしてもAPIは増えますが、今までに編集したコード部分も残ります。いわゆるマージ機能が追加されました。

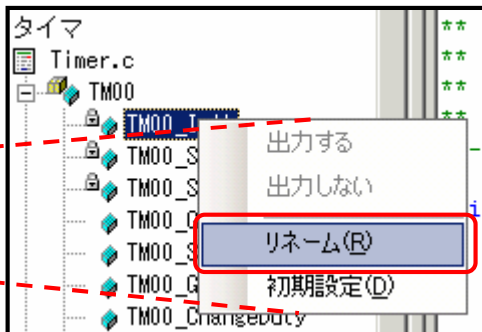
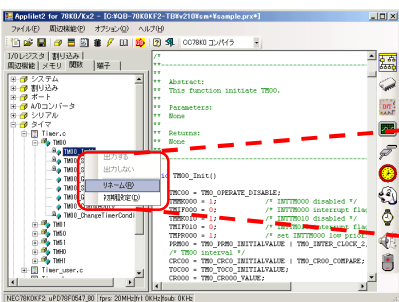
```
void function ( void )
```

```
{
/* Start user code. Do not edit comment generated here */
}

/* End user code adding. Do not edit comment generated here */
}
```

このコメントに挟まれた場所にプログラムを書けば、Applilet2のコード生成で上書きされることはありません。以前に書いたコードは、そのまま残ります。

## API名を自由に変更可能



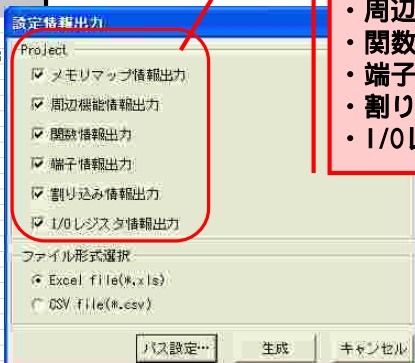
Applilet2で提供するAPIの名前を自由に変更可能です。

## 設定した内容のドキュメント化

別名	端子名	機能	ステータス	I/O	備考
3		MCU name : NEC78K0KF2			
4		Chip name : uPD78F0547_80			
5					
6	FLMDD	FLMDD	FLMDD	I	フラッシュメモリ・プログラミングモード
7	P00	P00/TI000	TI000	I	16ビット タイマ TM00 入力0
8	P01	P01/TI010/TO00	TI010	I	16ビット タイマ TM00 入力1
9	P02	P02/SO11	SO11	O	CSI11のシリアル データ出力
10	P03	P03/SH11	SH11	I	CSI11のシリアル データ入力
11	P04	P04/SCK11	SCK11	O	CSI11のクロック入力/出力
12	P05	P05/SI11/TI001	TI001	I	16ビット タイマ TM01 入力0
13	P06	P06/SI11/TI001			使用しない
14	P10	P10/SCK10/TxD0	SCK10	I/O	CSI10クロック入力/出力
15	P11	P11/SI10/RxD0	SI10	I	CSI10のシリアル データ入力
16	P12	P12/SO10	SO10	O	CSI10のシリアル データ出力
17	P13	P13/TxD6	TxD6	O	UART8のシリアル データ出力
18	P14	P14/RxD6	RxD6	I	UART8のシリアル データ入力
19	P15	P15/TOH0			使用しない
20	P16	P16/TOH1/INTP5			使用しない
21	P17	P17/TO50/TO50			使用しない
22	P20	P20/ANI0	ANI0	I	A/Dコンバータのアナログ入力
23	P21	P21/ANI1	ANI1	I	A/Dコンバータのアナログ入力
24	P22	P22/ANI2	ANI2	I	A/Dコンバータのアナログ入力
25	P23	P23/ANI3	ANI3	I	A/Dコンバータのアナログ入力
26	P30	P30/INTP1	INTP1	I	有効エッジ指定可能な外部割り込み要
27	P31	P31/INTP2	OCD1A	I	オンチップデバッグ モード引き込み用接続
28	P32	P32/INTP3	OCD1B	I	オンチップデバッグ モード引き込み用接続

下記の情報をEXCELファイルへ出力します

- ・ メモリマップ
- ・ 周辺機能情報
- ・ 関数情報
- ・ 端子情報
- ・ 割り込み情報
- ・ I/Oレジスタ情報



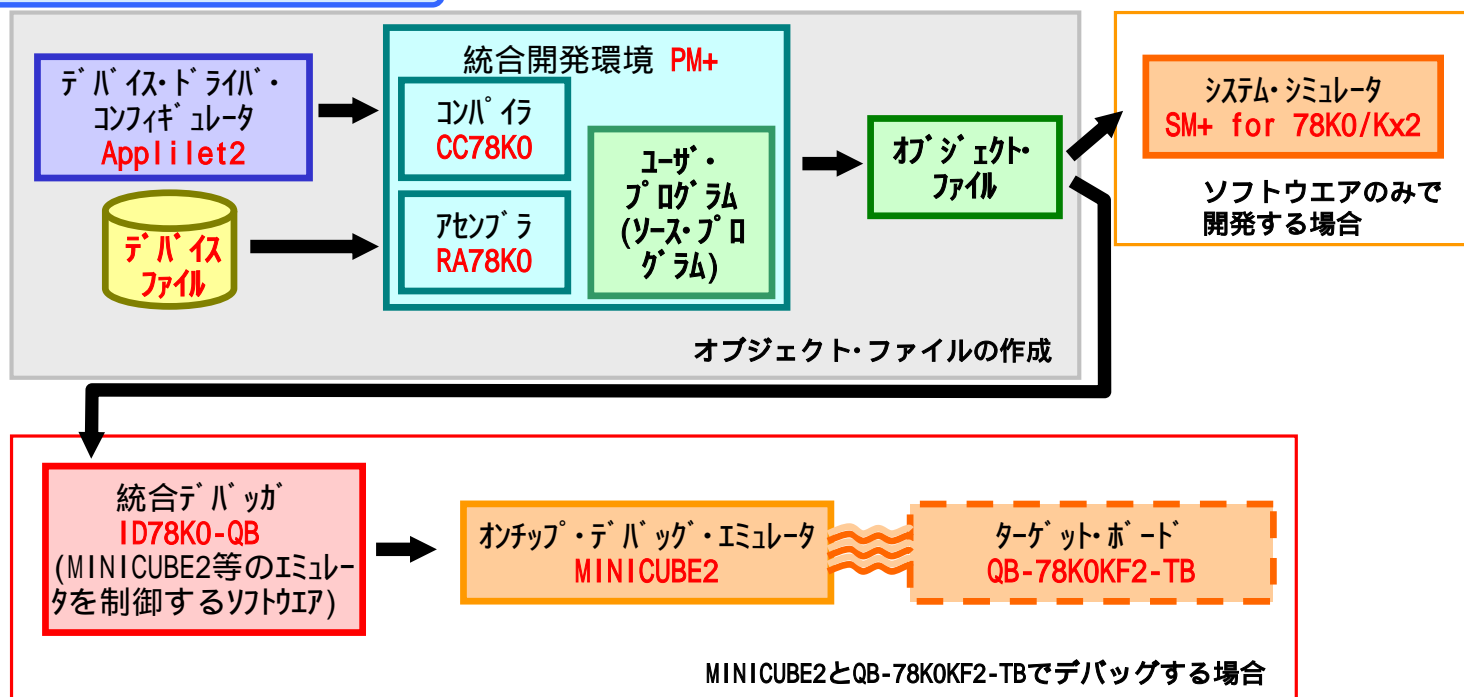
# システム構成図



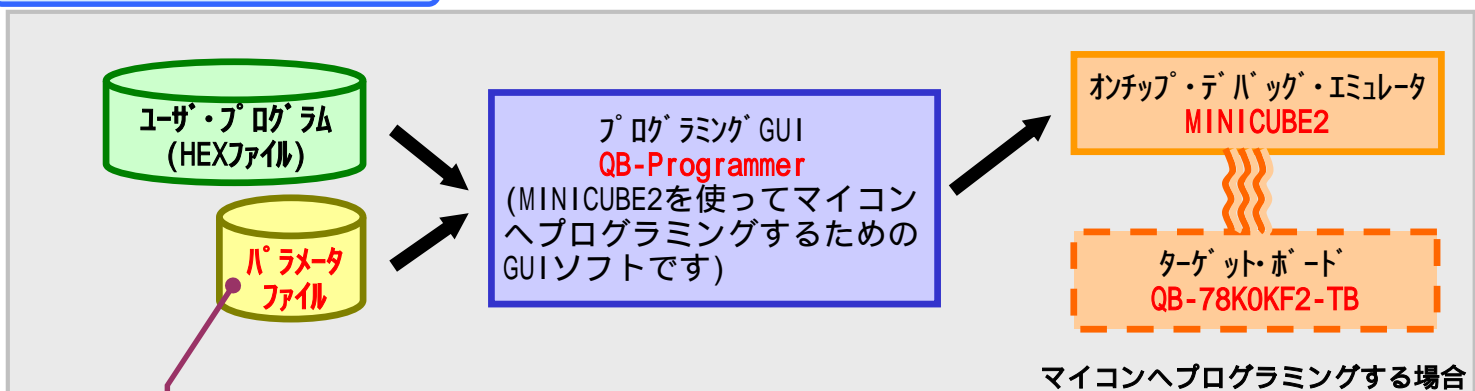
プログラムの開発方法としてソフトウェアのみで行う場合と、ハードウェアを使う場合の2通りの開発方法が選択できます。

- ・ソフトウェアのみで開発する場合(デバッグもPC上で行います)
  - ・MINICUBE2とターゲット・システムを使う場合(デバッグ時にはマイコンを使います)
- 完成したプログラムはHEXファイルにしてマイコンへ書き込みます。
- ・マイコンへプログラミングする

## プログラム開発の流れ



## プログラミングの流れ



### ワンポイント

フラッシュ・メモリの書き換え時に使用する品種依存情報ファイルです。

### プログラミングについて

通常プログラミングと言えばプログラムを作成することを示しますが、もう1つの意味があります。半導体デバイス(マイコン、各種ROMなど)へ書き込みを行う場合も「プログラミング」と呼びます。

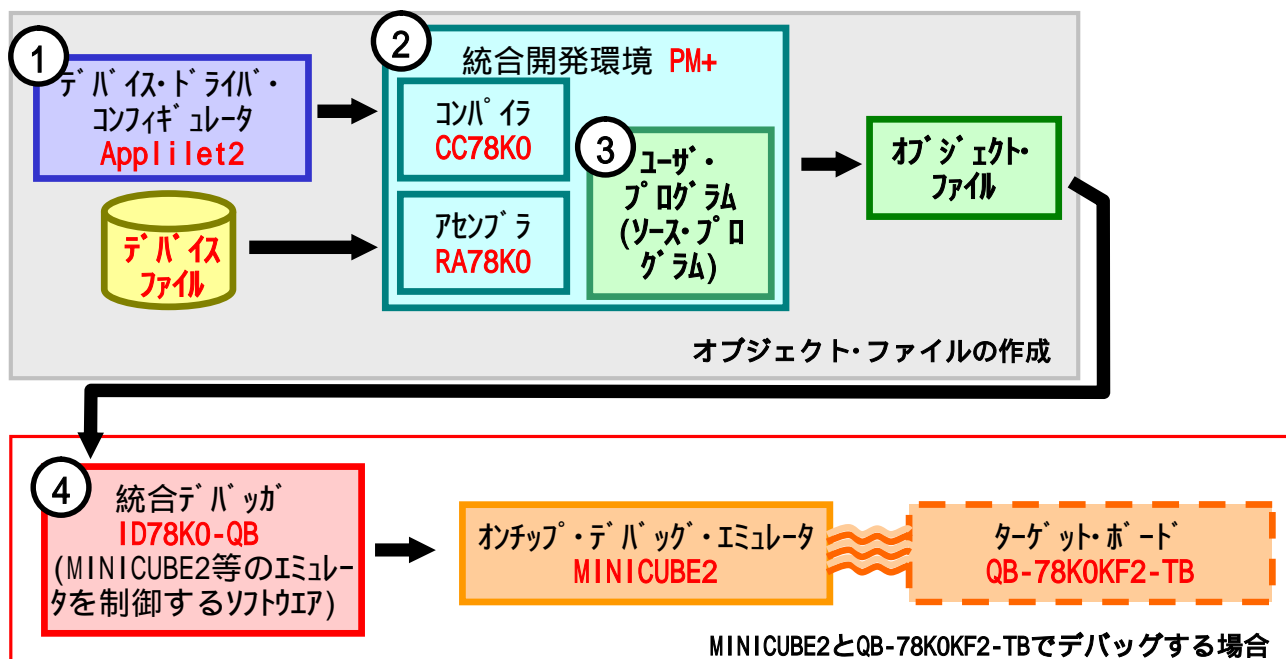
# 作成手順



まずハードウェアを動かしてみましょ。細かいことは気にせず本書に従って、進んで下さい。実際にハードウェアを動作させてからプログラムを説明します。[動かしてみよう]-プログラムの実行までは45分程度で試すことができます。あせらず、ゆっくり進んで下さい。

[準備]の[ソフトのダウンロード]、[ソフトのインストール]を済ませてから作業を行って下さい。

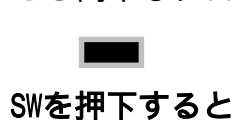
## プログラム作成手順



- ① Applilet2を使ってマイコンの設定を行います。GUIを使った簡単な操作で、マイコンの内蔵周辺機能を動作させるソースコードとプロジェクトファイルが自動生成されます。
- ② で生成したプロジェクトを読み込みます。PM+はソースコードのビルド、エディタや統合デバッガの起動などプログラム開発に必要な一連の操作をまとめて行うためのツールです。
- ③ ユーザ・プログラムを作成します。作成したプログラムをコンパイルしてオブジェクトファイルを作ります。
- ④ で作成したオブジェクト・ファイルを元にデバッグします。MINICUBE2とQB-78K0KF2-TBを使います。実際にマイコンを動作させます。

## ワンポイント

ここで作成するサンプル・プログラムはSWを1つ、LEDを2つ使います。交互に点滅を繰り返すLEDをSWの押下によって点滅の速度を変化させる簡単なプログラムです。

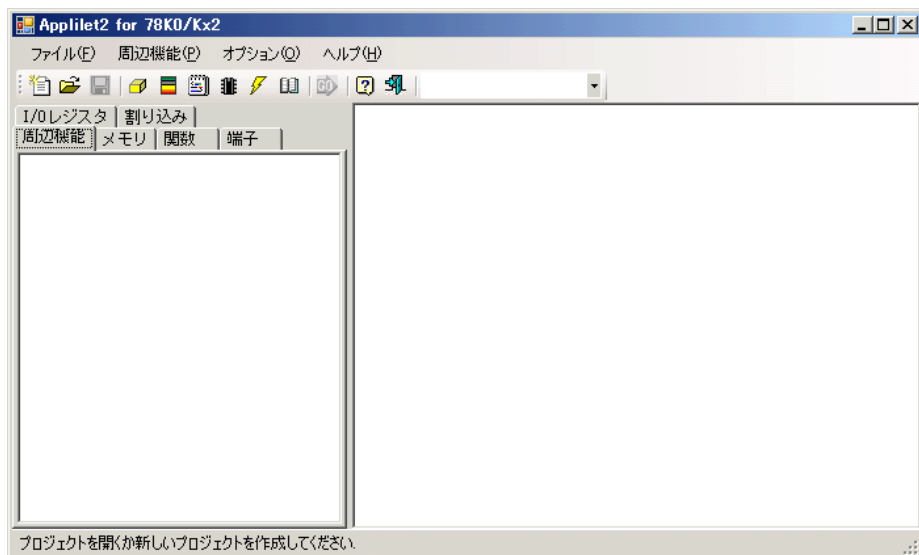


# Applilet2でプロジェクト作成

Applilet2を使い統合開発環境 PM+で読み込み可能なプロジェクトファイルを作成します。

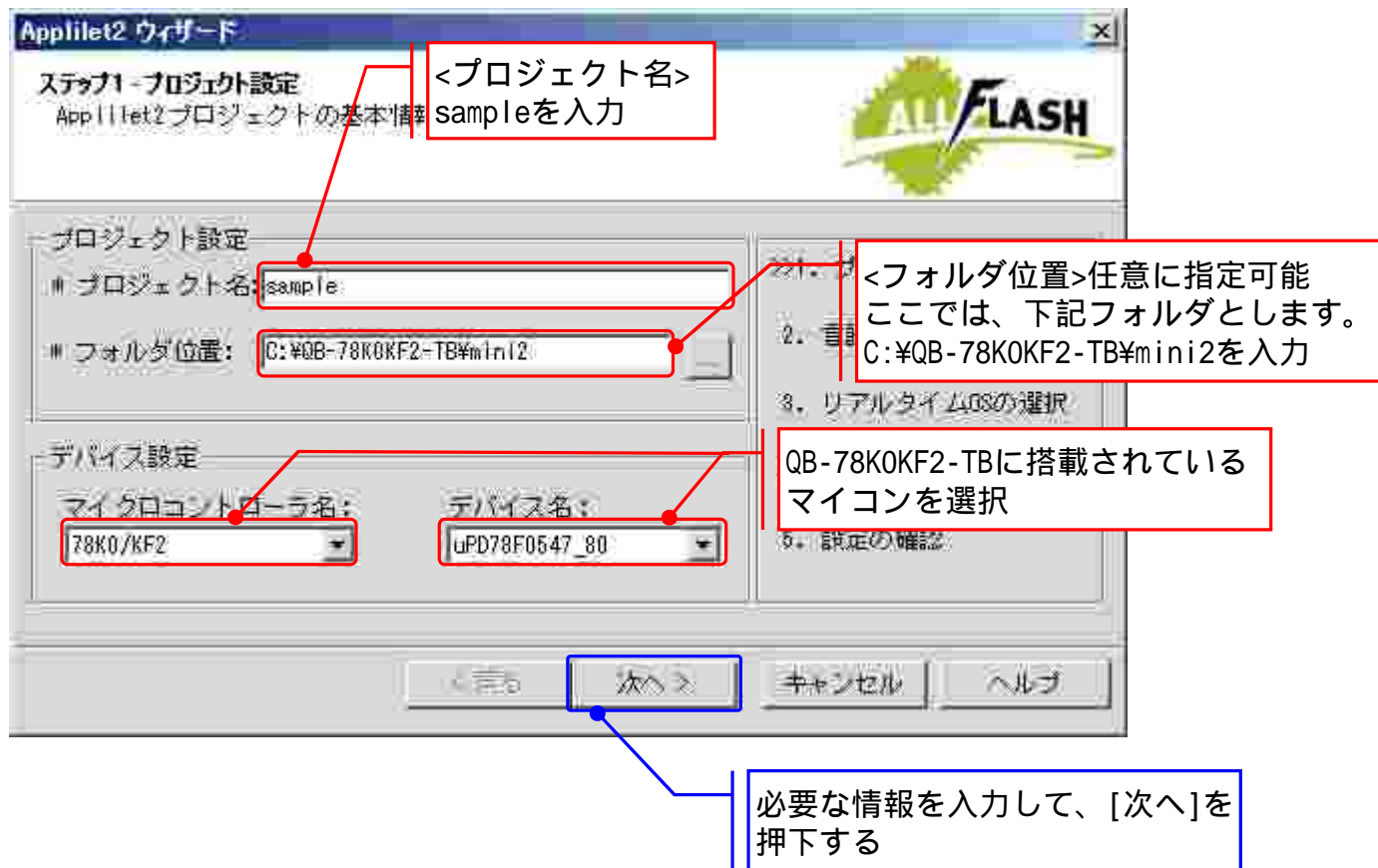
## a. Applilet2を起動します。

[スタート] [プログラム(P)] [NEC Electronics Tools]  
 [Applilet2 for 78K0KX2] [Vx.xx] [Applilet2 for 78K0KX2 Vx.xx]



## b. Applilet2のプロジェクトファイルを新規に作成します。

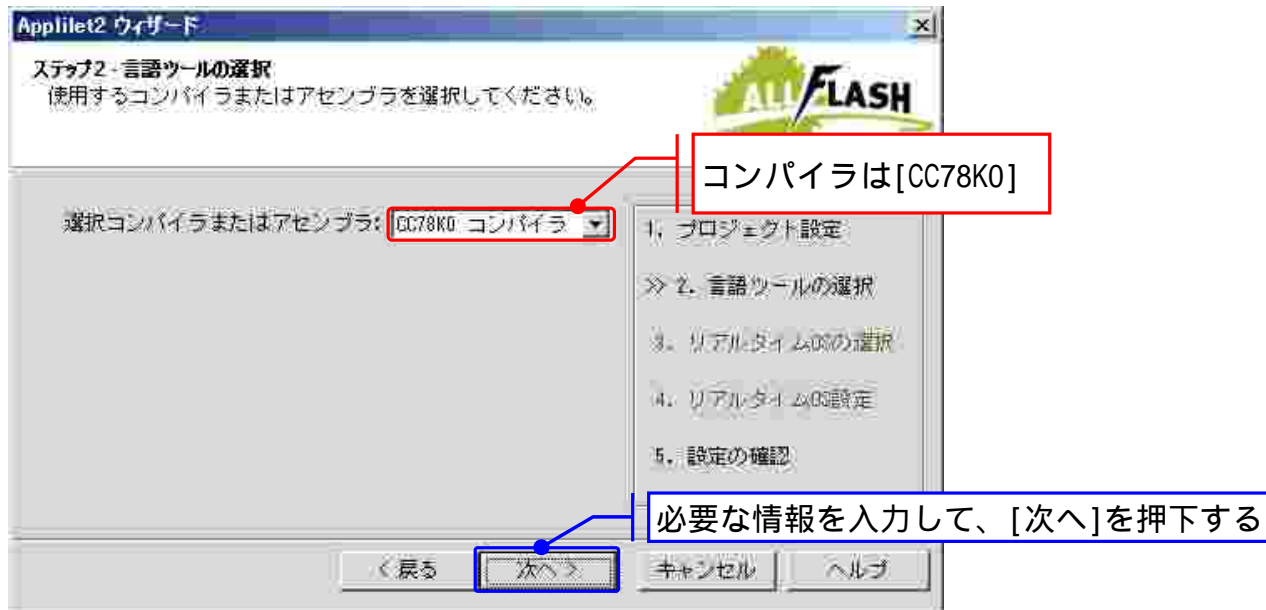
メニュー・バーの[ファイル(E)] [新規作成(N)...]を選択します。  
 「Applilet2ウィザード」ダイアログで、必要な情報を設定して下さい。



# Applilet2でプロジェクト作成

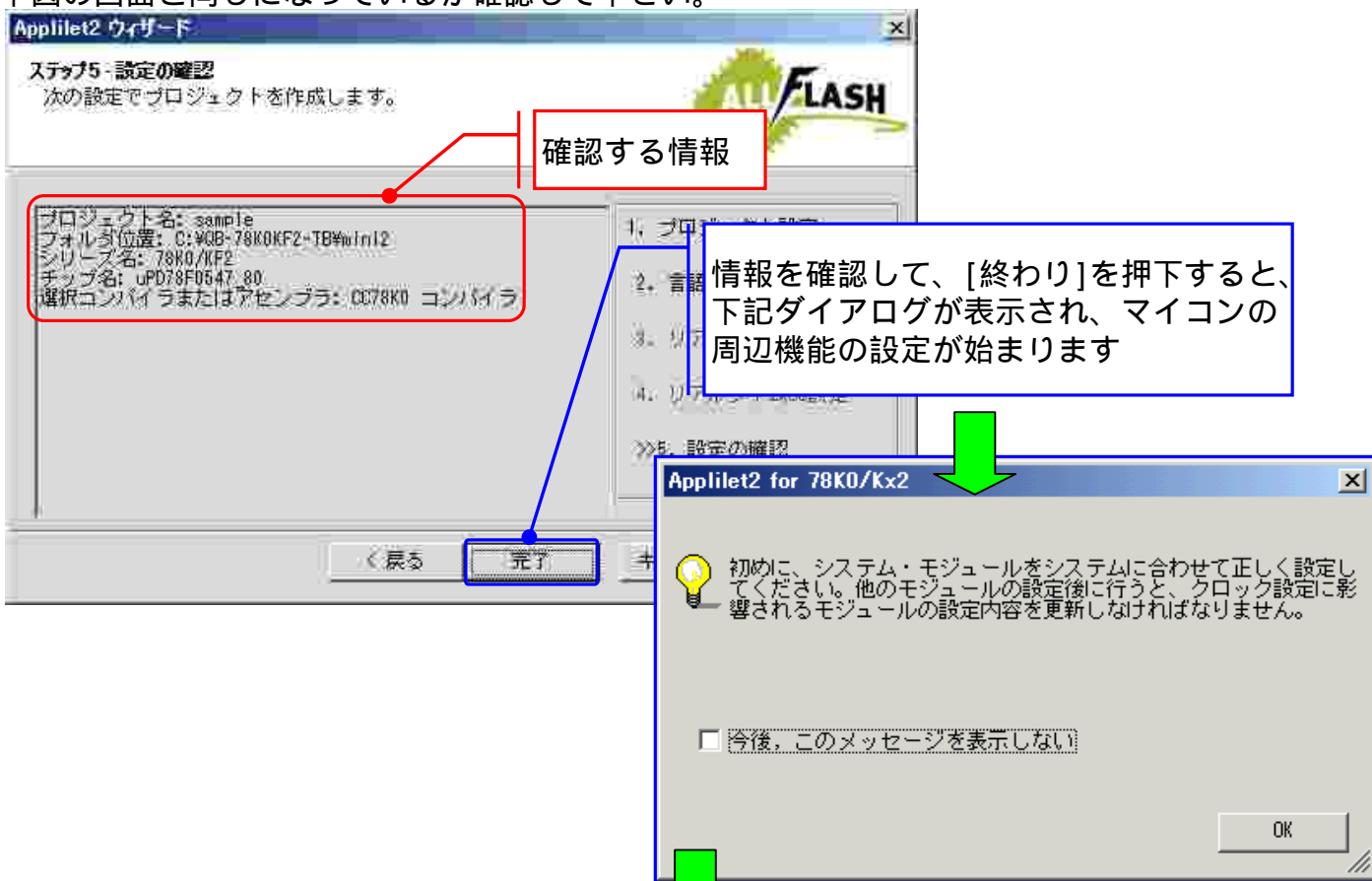
## c. 言語ツールを選択します。

[CC78K0]が表示されていることを確認して[次へ]を押下します。



## d. Applilet2の設定を確認します。

下図の画面と同じになっているか確認して下さい。



次よりマイコンの周辺機能の設定を行います

# 周辺機能設定[システム]

## e. [システム]を設定します。

[システム]ダイアログは、クロック等マイコンの基本を設定します。

情報を確認して、[オンチップ・デバッグ設定]タブを選択

高速内蔵発振器クロック設定  
 動作  
 周波数(MHz) **チェックを外す**

高速システム・クロック  
 **チェックする**

サブクロックの動作モード  
 使用しない  
 サブクロック(KHz) **20MHzを選択する**

CPUと周辺クロック設定  
 CPUクロック(KHz) 20000(fxp)  
 周辺クロック(MHz) 20(fxh) **20を入力する**

低速内蔵発振器  
 ソフトウェアによる停止可能(オプション・バイト) **チェックする**  
 停止  
 低速内蔵発振器(KHz) 240

初期設定 情報 **OK** Cancel

システム  
 クロック設定 | **オンチップ・デバッグ設定** | パワーオン・クリア設定

オンチップ・デバッグ設定  
 使用しない  
 **使用する**

端子選択  
 OCD0A, 0B  
 **チェックする**  
 OCD1A, OCD1B

RAM モニタ機能選択  
 使用しない  
 **使用する** **チェックする**

セキュリティID認証失敗時の設定  
 フラッシュ・メモリのデータを消去しない  
 フラッシュ・メモリのデータを消去する

セキュリティID設定  
 セキュリティIDを使用する  
 セキュリティID 0xffffffffffffffff

初期設定 情報 **OK** Cancel ヘルプ

情報を確認して、[OK]を押下すると確認ダイアログが表示されます

Applet2 for 78K0/Kx2

システムクロックが変更されました。クロック設定に影響されるモジュールの設定内容を更新する必要があります。

OK

ウォッチドッグ・タイマ

動作設定  
 使用する  
 **使用しない**

動作クロック  
 監視時 **チェックする**

ウィンドウ・オープン  
 ウィンドウ・オープン期間(%) 100%

初期設定 情報 **OK** Cancel ヘルプ

情報を確認して、[OK]を押下

### 💡 ワンポイント

**ウォッチドッグ・タイマ**について  
 ウォッチドッグタイマとはプログラムの暴走を検出するための機構です。プログラム暴走と検出された場合は内部リセット信号が発生されマイコンはリセットされます。より信頼性の高いプログラムにするためにはウォッチドッグタイマを使用します。ここでは、プログラムが複雑になってしまうので、ウォッチドッグ・タイマは使用しません。

### 💡 ワンポイント

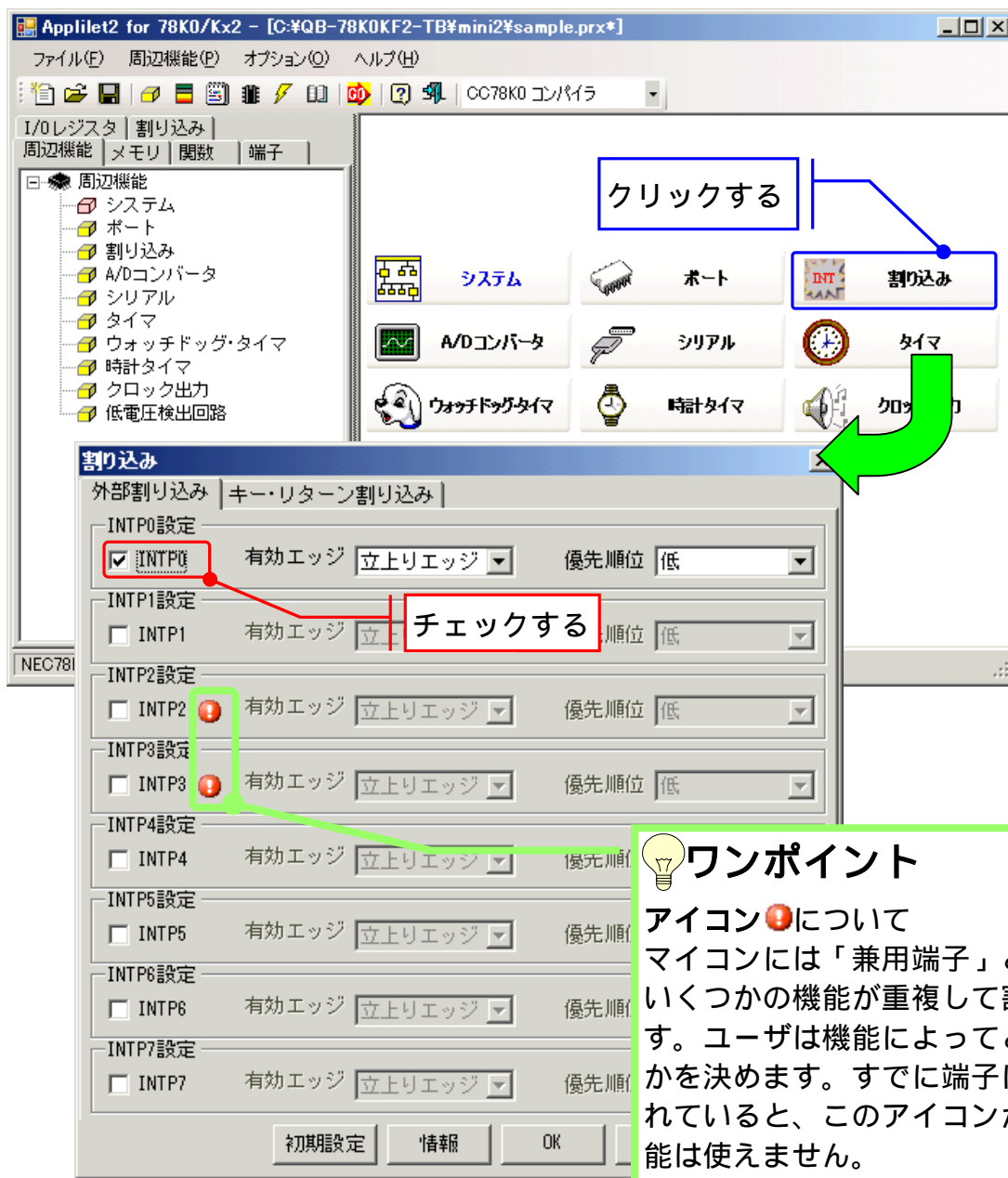
#### システムについて

メイン・システム・クロック、サブクロックの設定を行います。ここで指定したクロック値は、タイマ・モジュールのコンペア・レジスタ値の計算やシリアル・モジュールのボーレートに影響を与えます。後から動作クロックを変更した場合には、シリアルのボーレートの値等を確認して下さい。



# 周辺機能設定[割り込み]

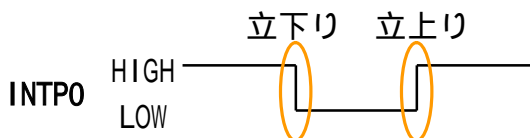
- f. [割り込み]を設定します。  
 [割り込み]ダイアログは、外部割り込みを設定します。



## ワンポイント

### 外部割り込みについて

外部端子割り込み、キー割り込みの設定を行います。サンプルプログラムではINTP0の外部端子にSWが接続されています。有効エッジの立下がりエッジとは、信号が1から0へ変化するとき有効とする設定です。ターゲット・ボードのSWを押した時が立ち下がり、押してからSWを離れた時が立ち上がりになります。「立下がり」の逆が「立上がり」です。



# 周辺機能設定[ポート]

## g. [ポート]を設定します。

[ポート]ダイアログは、ポートの入出力を設定します。

クリックする

[ポート7]タブの設定

P76、P77の「出力、1」にチェックします。LED1、LED2はそれぞれP76、P77のポートに接続されています。LEDはLOW出力によって点灯するので初期値をHIGH(1)にします。

[ポート12]タブの設定

P120のPull-upにチェックします。P120はINTP0と兼用端子です。INTP0の外部割り込み端子がLOWの時に割り込みが発生するように設定するので、ここではPull-up設定します。「入力、出力」の前にあるアイコン(赤い丸)は、P120の入出力ポートが使えないことを意味しています。またP121、P122にはX1発振のシステムクロックが接続されているので使えません。

## 💡ワンポイント

### ポートについて

各ポートの設定を行います。各ポートは入力/出力、内蔵プルアップ抵抗(Pull-up)、初期値の設定が可能です。ポートは他の周辺I/Oと兼用端子になっている場合が殆どです。

# 周辺機能設定[タイマ]

h. [タイマ]を設定します。

[タイマ]ダイアログで、TM00、TM01のタイマ割り込みを設定します。

**TM00の設定**

クリックする

[インターバル・タイマ]にチェックし、[詳細]を押下し、次画面へ進みます。

タイマ00のインターバル時間を10msecへ変更します。

**TM01の設定**

[インターバル・タイマ]にチェックし、[詳細]を押下し、次画面へ進みます。

タイマ01のインターバル時間を100msecへ変更します。

## 💡ワンポイント

タイマについて

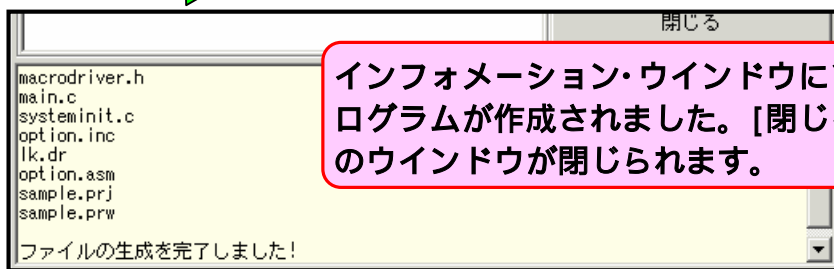
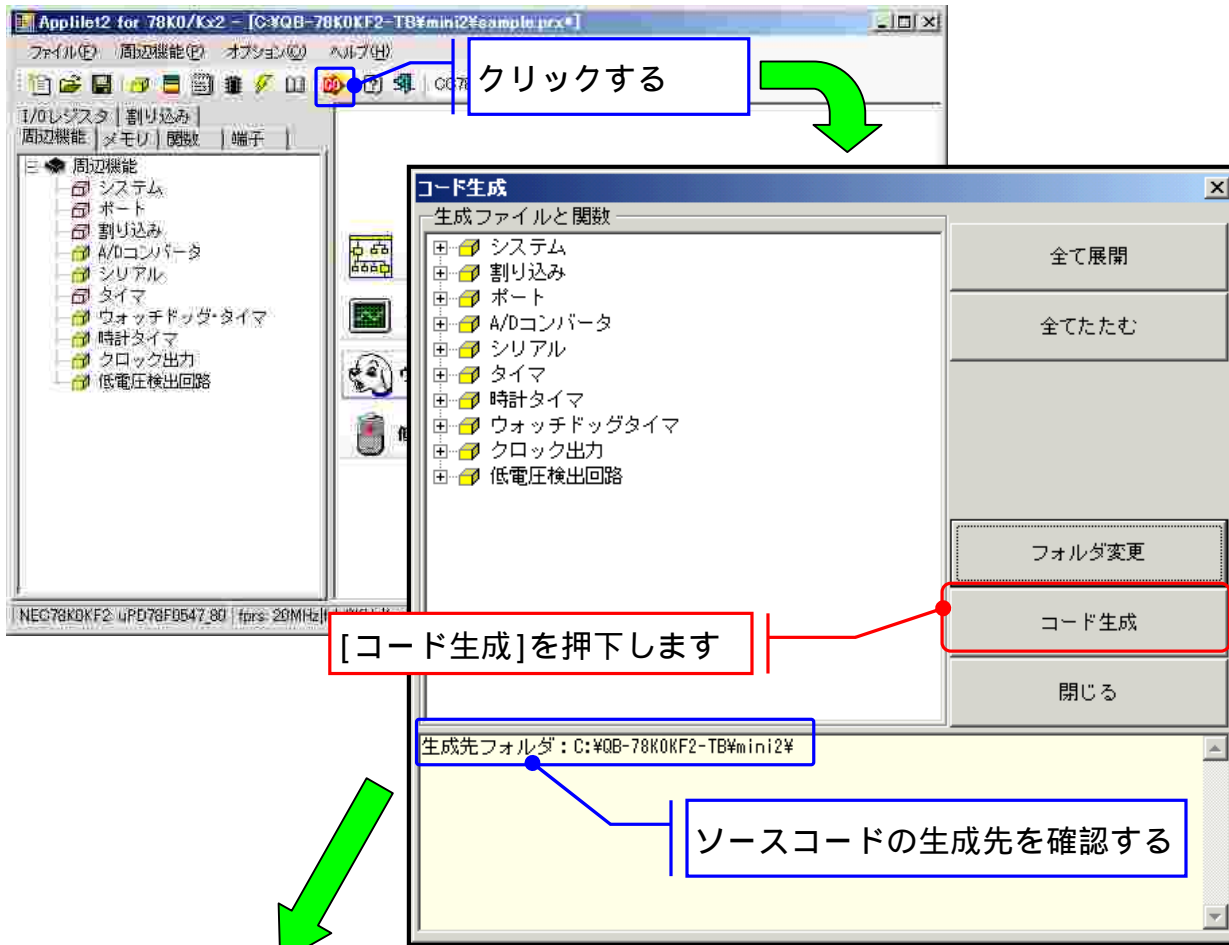
タイマにはインターバル・タイマの他にも様々な機能があります。

外部イベント・カウンタ（外部から入力される信号のパルス数を測定できます）、方形波出力（任意の周波数の方形波出力が可能）、PPG出力（周波数と出力パルス幅を任意に設定できる矩形波を出力できます）、ワンショット・パルス出力（出力パルス幅を任意に設定できるワンショット・パルスを出力できます）、パルス幅測定（外部から入力される信号のパルス幅を測定できます）、PWM出力などがあります。

# コード生成

## i. コード生成します。

コード生成アイコン  を押下してプログラムを生成します。



## 💡 ワンポイント

### コード生成について

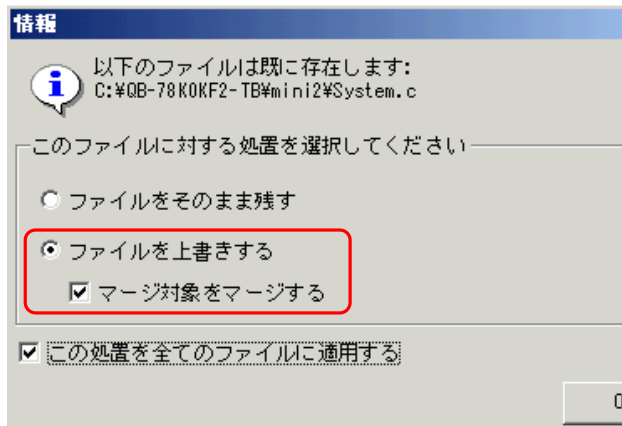
すでにソースファイルが存在していた場合は、右図のダイアログが表示されます。「ファイルを上書きする」、「マージ対象をマージする」にチェックすれば、下記のコメントに、はさまれたソースコードは上書きされずにマージされます。

```
/* Start user code. Do not edit comment generated here */
```

ユーザのソースプログラム

```
/* End user code adding. Do not edit comment generated here */
```

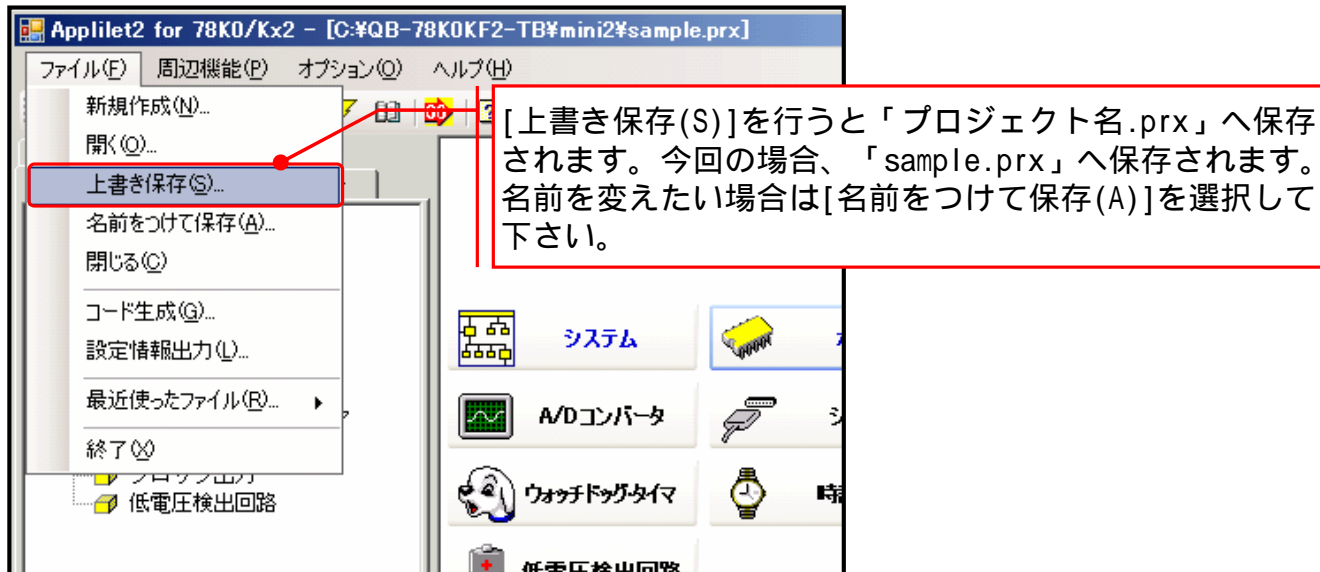
これはApplilet2の新機能「ソースコードのガード機能」です。



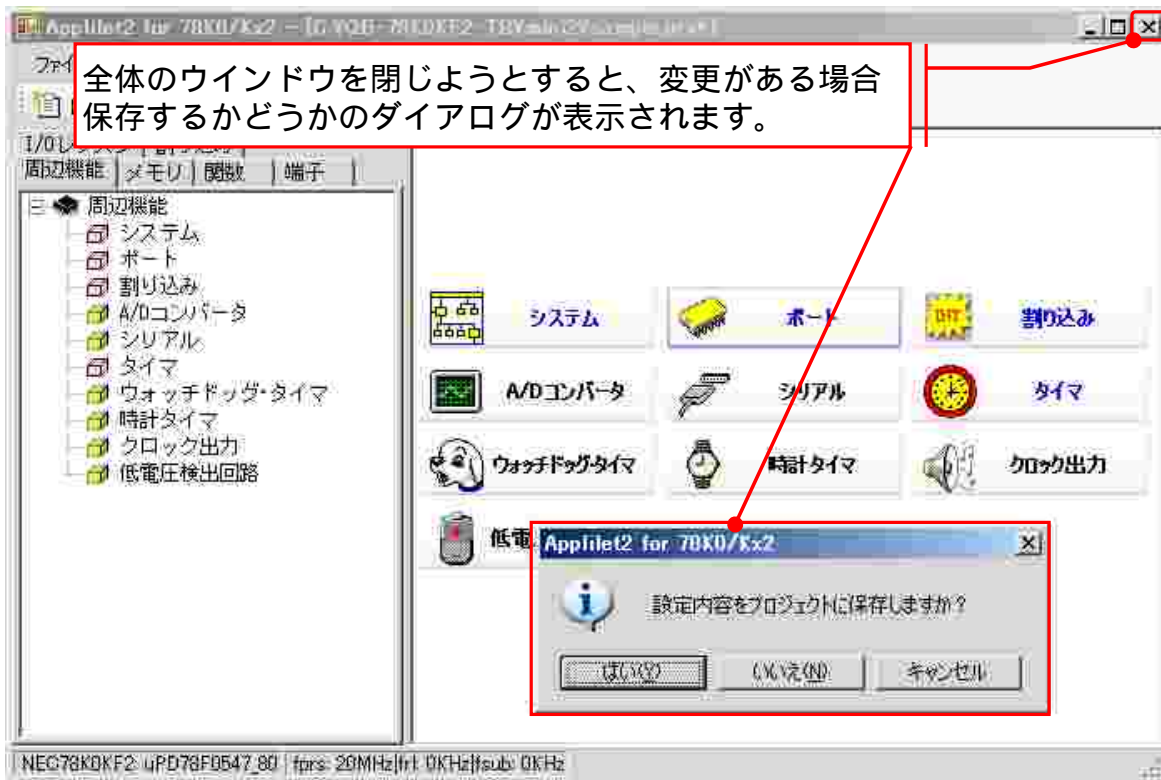
# プロジェクトの保存

## j. プロジェクトの保存

メニューより ファイル(F) -> 上書き保存(S)...でプロジェクトを保存します。



また、AppIilet2のウィンドウ自体を閉じようとする時、設定に変更がある場合プロジェクトを保存するかダイアログが表示されます。



# PM+を起動する

統合開発環境 PM+でプロジェクトファイルを読み込み環境の設定を行います。

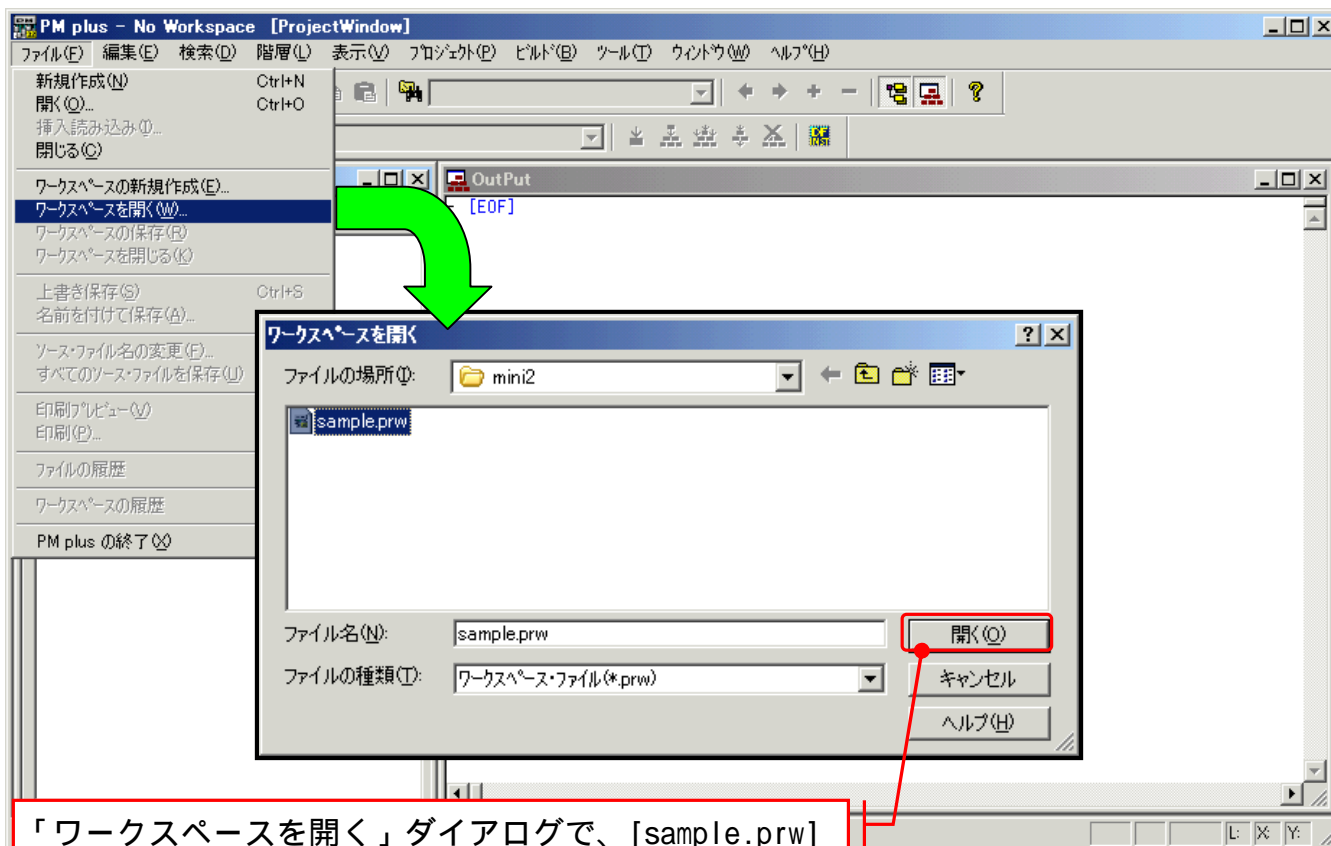
## k. PM+を起動します。

[スタート] [プログラム(P)] [NEC Tools32] [PM plus]

## l. ワークスペース(sample.prw)を開きます。

メニューの[ファイル(F)] [ワークスペースを開く(W)...] を選択し、

"c:¥QB-78K0KF2-TB¥mini2¥sample.prw" を指定して、[開く(O)] ボタンを押して下さい。



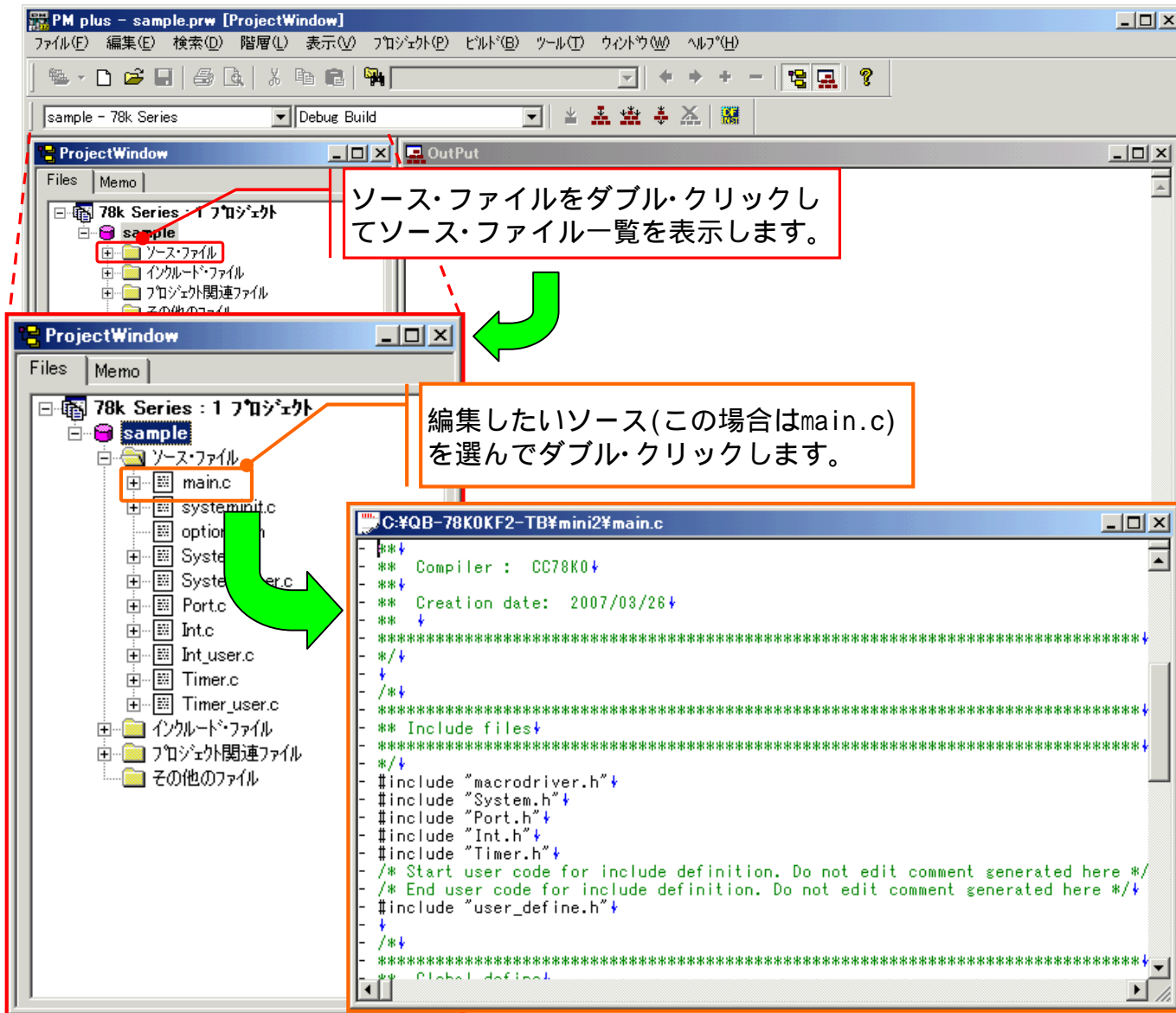
「ワークスペースを開く」ダイアログで、[sample.prw] を選び、[開く]を押下してワークスペースを開きます。

# プログラムの編集

ユーザ・プログラムを作成します。

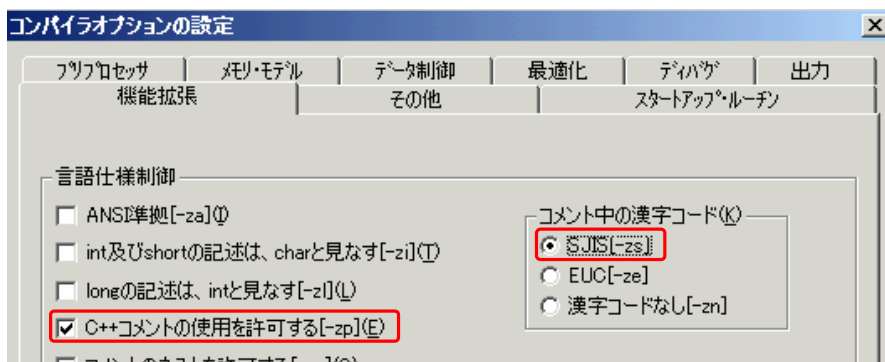
## m. 編集するソースファイルを開きます。

[ProjectWindow] の ソース・ファイルをダブル・クリック 編集したいソースをダブル・クリック  
 します。次ページの内容に従って、編集して下さい。



## ワンポイント

ソースの記述について  
 ソースファイル中にコメントを漢字で記述することができます。また、コメントの記述として // が使用可能です。メニューの[ツール(T)] [コンパイラオプションの設定(C)] を選択します。機能拡張タブを選択し、右図の通りに設定して下さい。



# プログラムの編集

## n. 各ソースを編集します。

### main.c

```
void main( void )
{
    /* Start user code. Do not edit comment generated here */
    g_interval_sw = 0;
    g_interval_count = 1;

    TM00_Start();
    TM01_Start();

    while (1) {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

コードを追加して下さい

### Int\_user.c

```
/*
*****
** Global define
*****
*/
/* Start user code for global definition. Do not edit comment generated here */

UCHAR g_interval_sw = 0;
UCHAR g_onetime_sw = 0;

/* End user code for global definition. Do not edit comment generated here */

/*
** -----
** Abstract:
** This function is INTPO interrupt service routine.
** Parameters:
** None
** Returns:
** None
** -----
*/
__interrupt void MD_INTPO( void )
{
    /* Start user code. Do not edit comment generated here */

    if ( g_onetime_sw == 0 )
    {
        g_onetime_sw = 2;
        g_interval_sw++;
        g_interval_sw = g_interval_sw & 7;
    }

    /* End user code. Do not edit comment generated here */
}
```

コードを追加してください

コードを追加して下さい



# プログラムの編集

## Timer\_user.c

```

/*****
** Global define
*****/
*/
/* Start user code for global definition. Do not edit comment generated here */
UINT g_interval_count;
UCHAR g_counter_data[ 8 ] = { 2, 4, 7, 15, 31, 47, 63, 127 };

/* End user code for global definition. Do not edit comment generated here */

/*-----
** Abstract: This function is INTTM00 interrupt service routine.
**-----
*/
__interrupt void MD_INTTM00( void )
{
    /* Start user code. Do not edit comment generated here */
    UCHAR inreg, outreg;
    if ( g_interval_count == 0 )
    {
        g_interval_count = g_counter_data[ g_interval_sw & 7 ];
        inreg = P7.6;
        outreg = inreg ^ 1;
        P7.6 = outreg;
        P7.7 = inreg;
    }
    g_interval_count--;
    /* End user code. Do not edit comment generated here */
}

/*-----
** Abstract: This function is INTTM01 interrupt service routine.
**-----
*/
__interrupt void MD_INTTM01( void )
{
    /* Start user code. Do not edit comment generated here */
    if ( g_onetime_sw != 0 )
    {
        g_onetime_sw--;
    }
    /* End user code. Do not edit comment generated here */
}

```

コードを追加して下さい

コードを追加して下さい

コードを追加して下さい

## user\_define.h

```

/*****
** Macro define
*****/
*/
/* Start user code for definition. Do not edit comment generated here */
extern UCHAR g_interval_sw;
extern UCHAR g_onetime_sw;
extern UINT g_interval_count;
/* End user code for definition. Do not edit comment generated here */

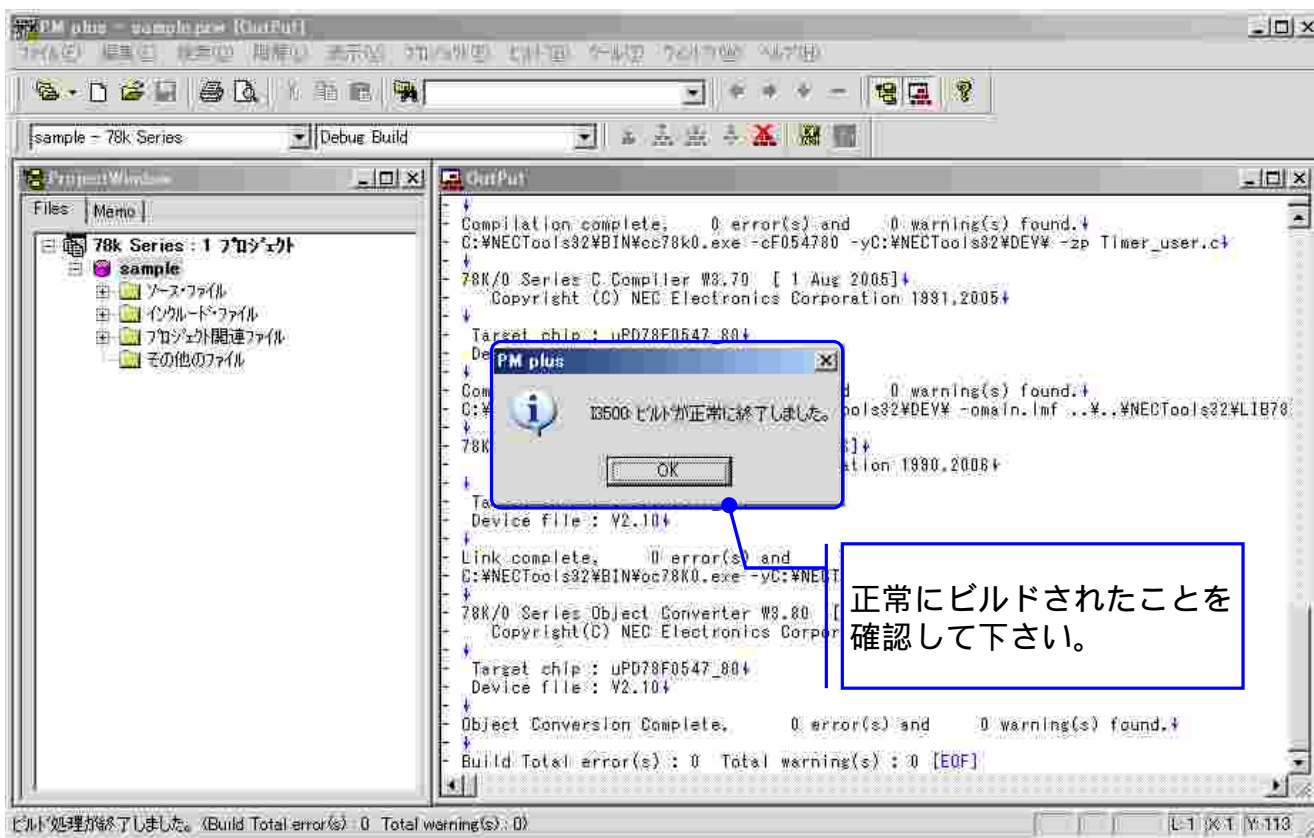
```

コードを追加して下さい

# プログラムのビルド

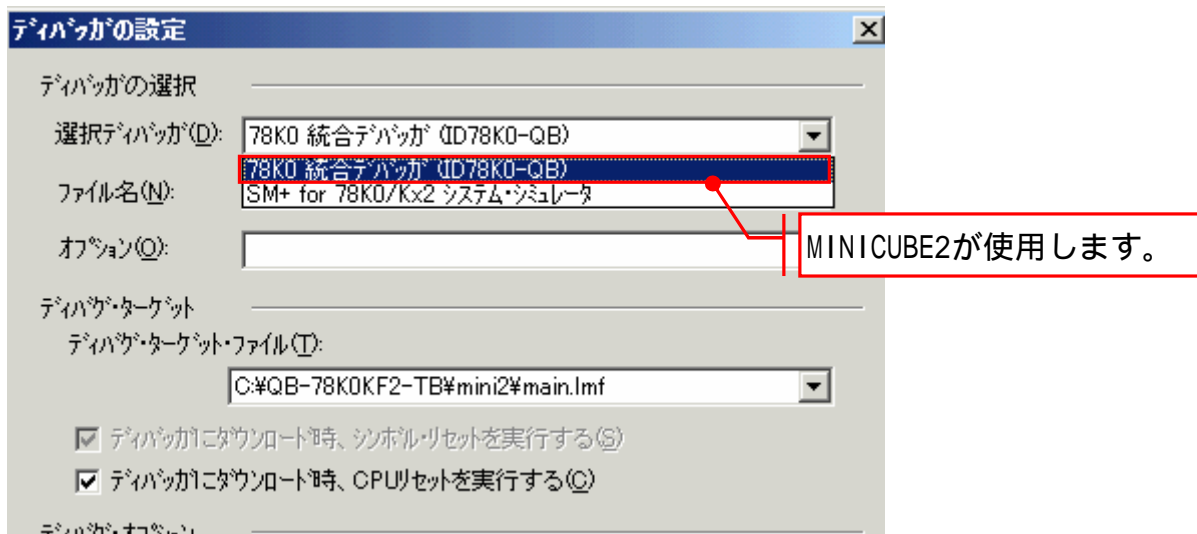
## o. プログラムをビルドします。

メニューの[ビルド(B)] [ビルド(B)] を選択するか、「F7」キーを押下して下さい。  
プログラムがコンパイルされオブジェクト・ファイルが作成されます。



## p. デバッガを設定します。

メニューの[ツール(T)] [デバッガの設定(D)...] で「デバッガの設定」ダイアログを開き「ID78K0-QB」を選択して下さい。



# MINICUBE2の接続

実際にQB-78K0KF2-TB上でプログラムを実行します。

q. PC本体、MINICUBE2、QB-78K0KF2-TBを接続します。  
接続する順番に注意して接続します。

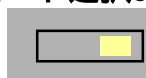
1. MINICUBE2のスイッチを設定します。



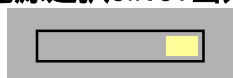
スイッチを設定します

モード選択SW:M2

電源選択SW:5V出力



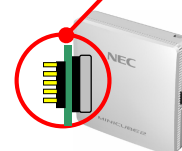
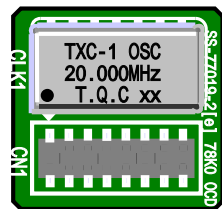
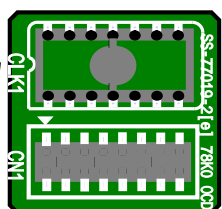
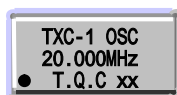
M1 M2



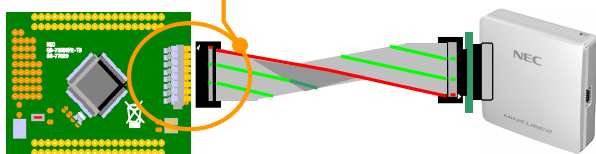
3 T 5

2. 78K0OCDボードとMINICUBE2を接続します。

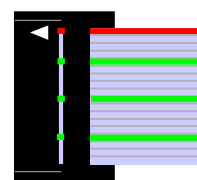
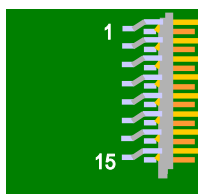
クロック・ボード(78K0-OCDボード)に発振器が載っていることを確認します



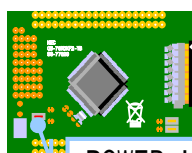
3. QB-78K0KF2-TBと接続します。



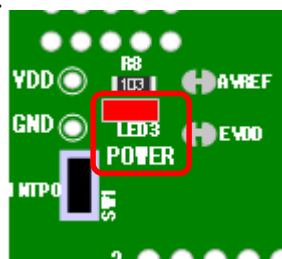
コネクタの1pinを合わせて接続します



4. MINICUBE2とPC本体をUSBケーブルで接続します。



POWER LED(LED3) が点灯します

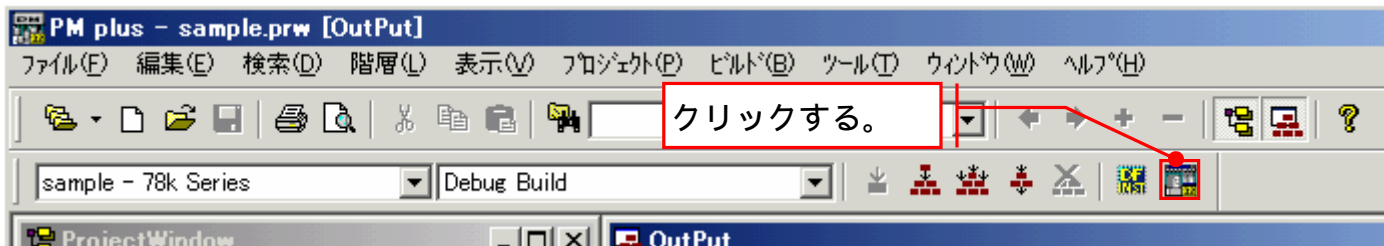


中央のLEDが点灯します。

# デバッグの起動

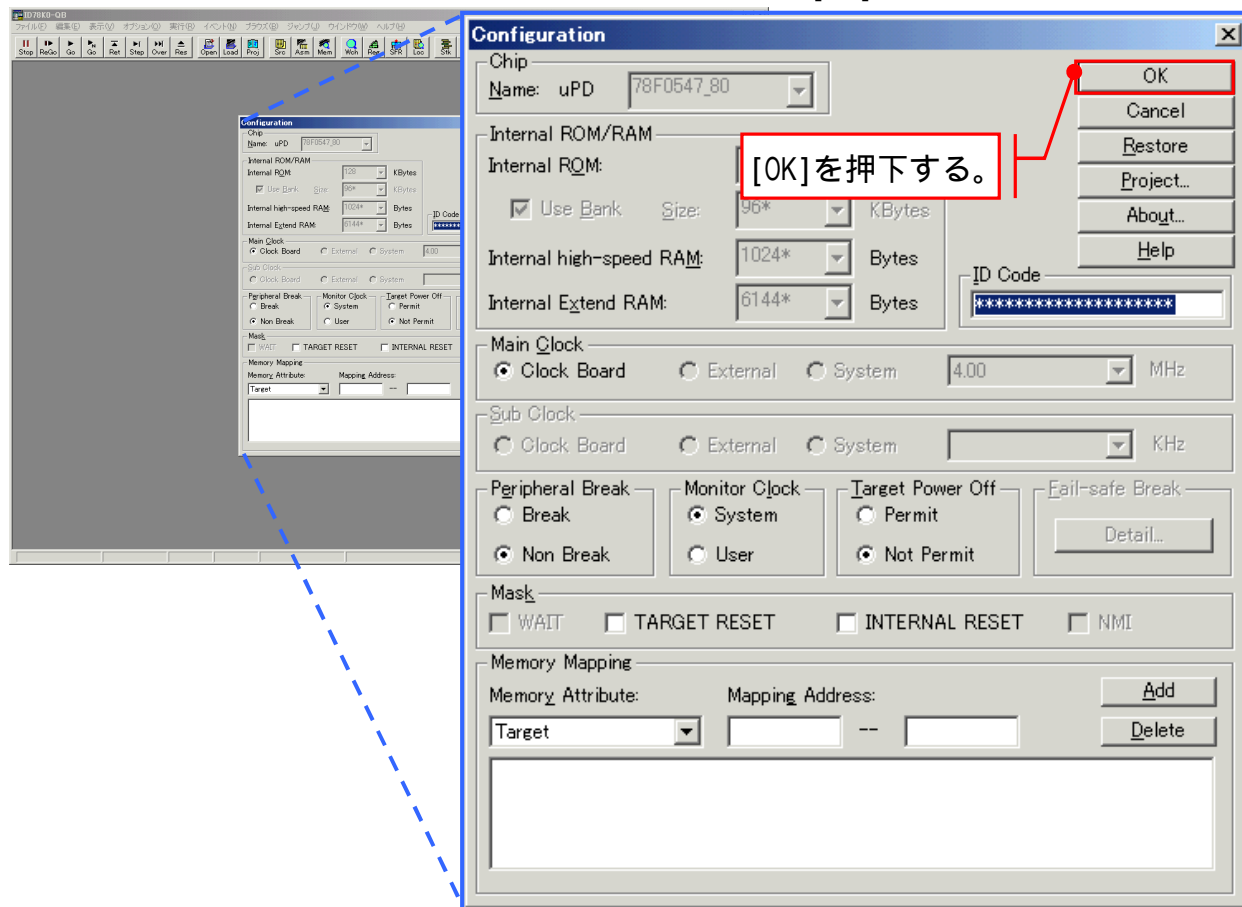
## r. デバッガを起動します。

PM+のデバッグ・アイコンを押下して[ID78K0-QB]を起動します。



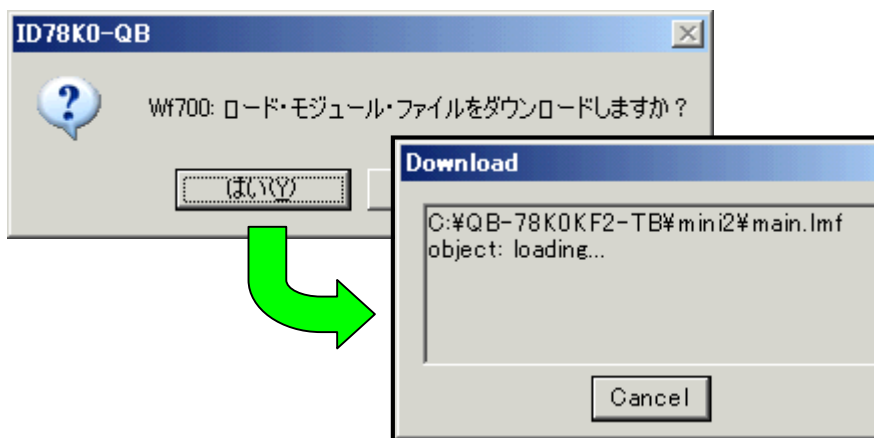
## s. デバッガの設定を行います。

デバッガが起動するのでターゲットとの接続を設定し、[OK]を押下します。



## t. プログラムをダウンロードします。

下記ダイアログが表示されるので[はい(Y)]を押下して、プログラムをダウンロードします。



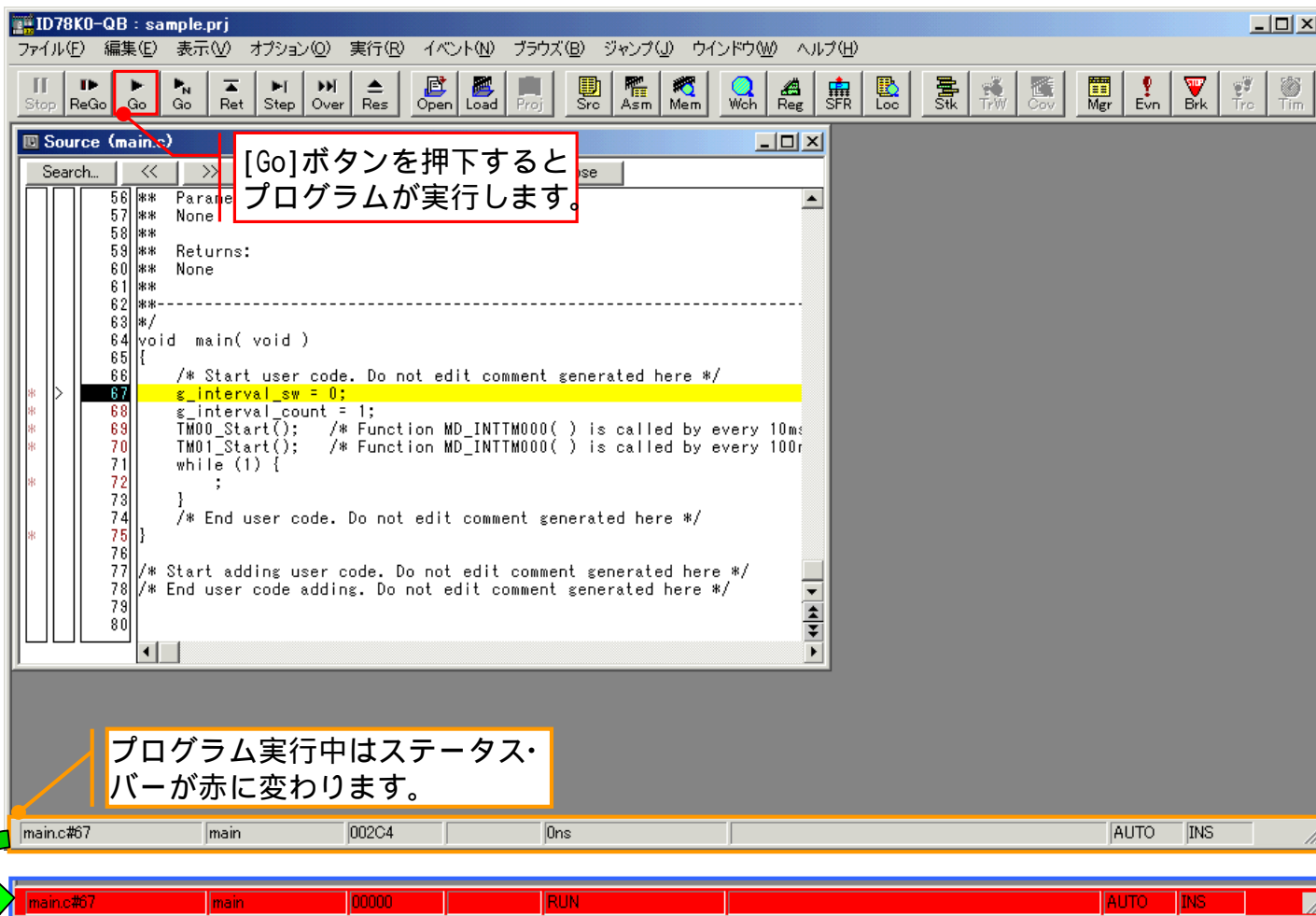
**ワンポイント**

ダウンロードの時間は2~3秒ほどです。また、ダウンロード中はMINICUBE2のLEDが1秒間隔で点滅します。

# プログラムの実行

## u. プログラムを実行します。

プログラムがロードされ、ソースファイルが表示されたら、プログラムを実行します。



プログラム実行中はステータス・バーが赤に変わります。

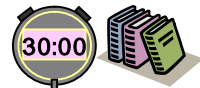
SWを押下するとLEDの点滅速度が変化します。

LEDが点滅します。最初は高速に点滅するため両方点灯しているように見えますがSWを押すと点滅がゆっくりになります。

ターゲット・ボード上の動作

プログラムは動いたでしょうか？

思ったより簡単にマイコンを動かすことができましたと思います。次ページよりサンプル・プログラムについて説明します。プログラムの動作を理解して、自由にプログラムを改造して下さい。更にマイコンへの理解が深まるはずです。



# プログラムの説明

マイコンについて基本的な知識は弊社のWEBページ「マイコンe-Learning」で学習できます。下記URLよりアクセスして下さい。

<http://www.necel.com/micro/ja/eLearning/>

これよりサンプル・プログラムについて説明します。学習予定時間は30分です。

## 1. サンプル・プログラムの構成

大きく分けて[メイン]、[タイマ割り込み]、[外部割り込み]の3つに分類できます。

### メイン処理:

マイコンがリセットされてから動作する処理です。変数初期化、タイマの開始を行った後は無限ループになります。

### タイマ割り込み処理:

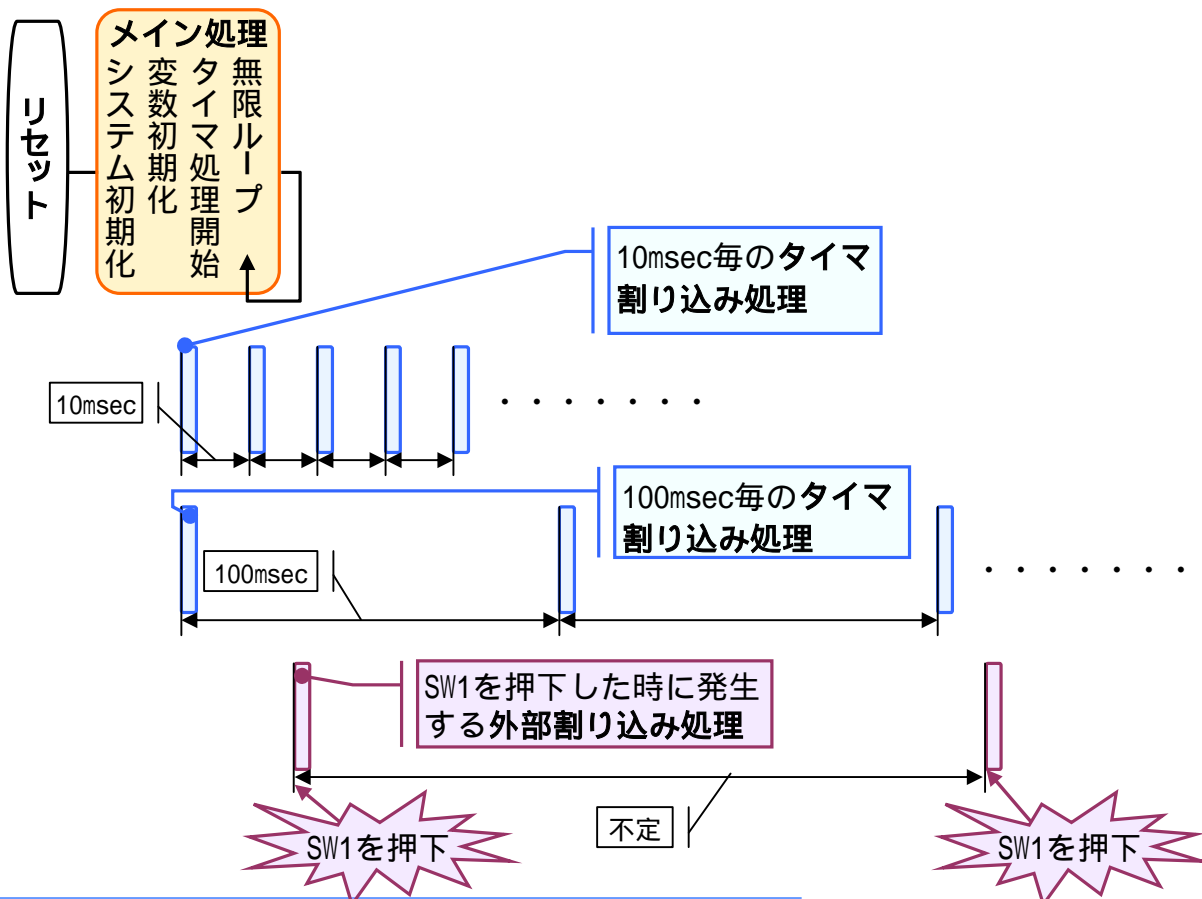
10msec毎と100msec毎に呼ばれる2つの処理があります。10msecの割り込み処理はLEDの点灯処理、100msecの割り込み処理はチャタリング防止処理です。

### 外部割り込み処理:

QB-78K0KF2-TB上のSW1が押された時に呼ばれる割り込み処理です。回路にチャタリング防止が施されていないのでソフトウェアでチャタリング防止を行います。

## 2. プログラムの時間的な流れ

[メイン]は初期化とタイマ処理開始だけの処理です。プログラムは[タイマ割り込み]、[外部割り込み]で動作します。

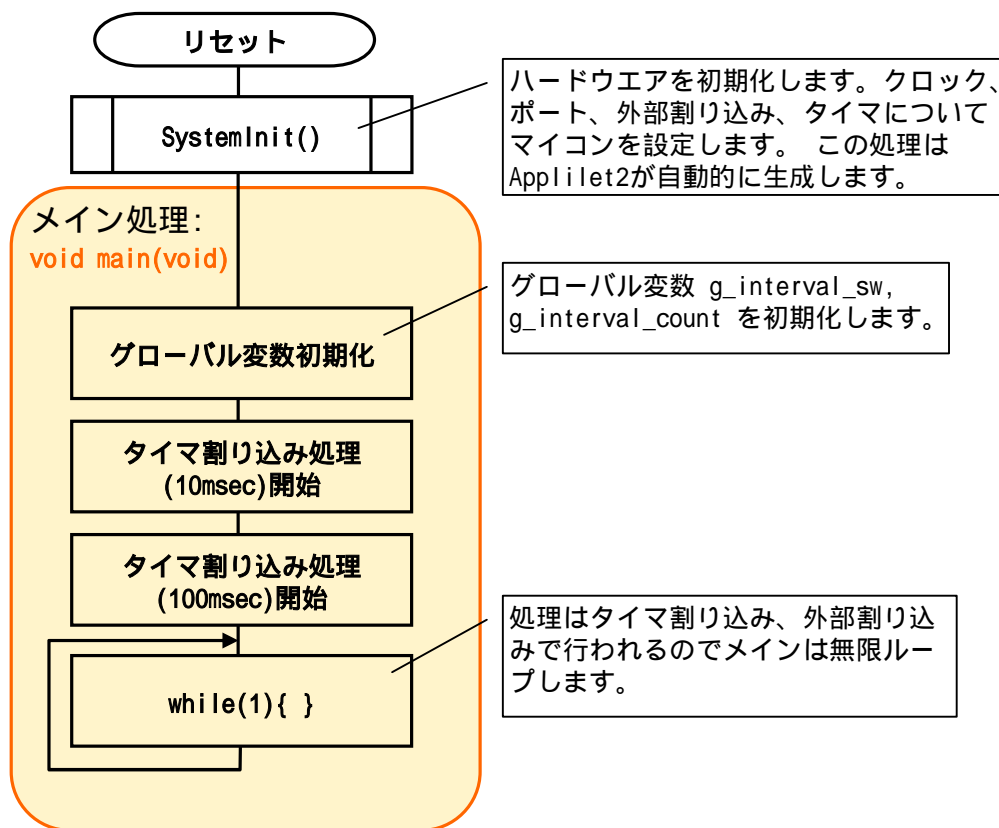




# プログラムの説明

## 3. 処理の説明

[メイン]、[タイマ割り込み]、[外部割り込み]、それぞれの処理について説明します。



### 全体的な処理の説明

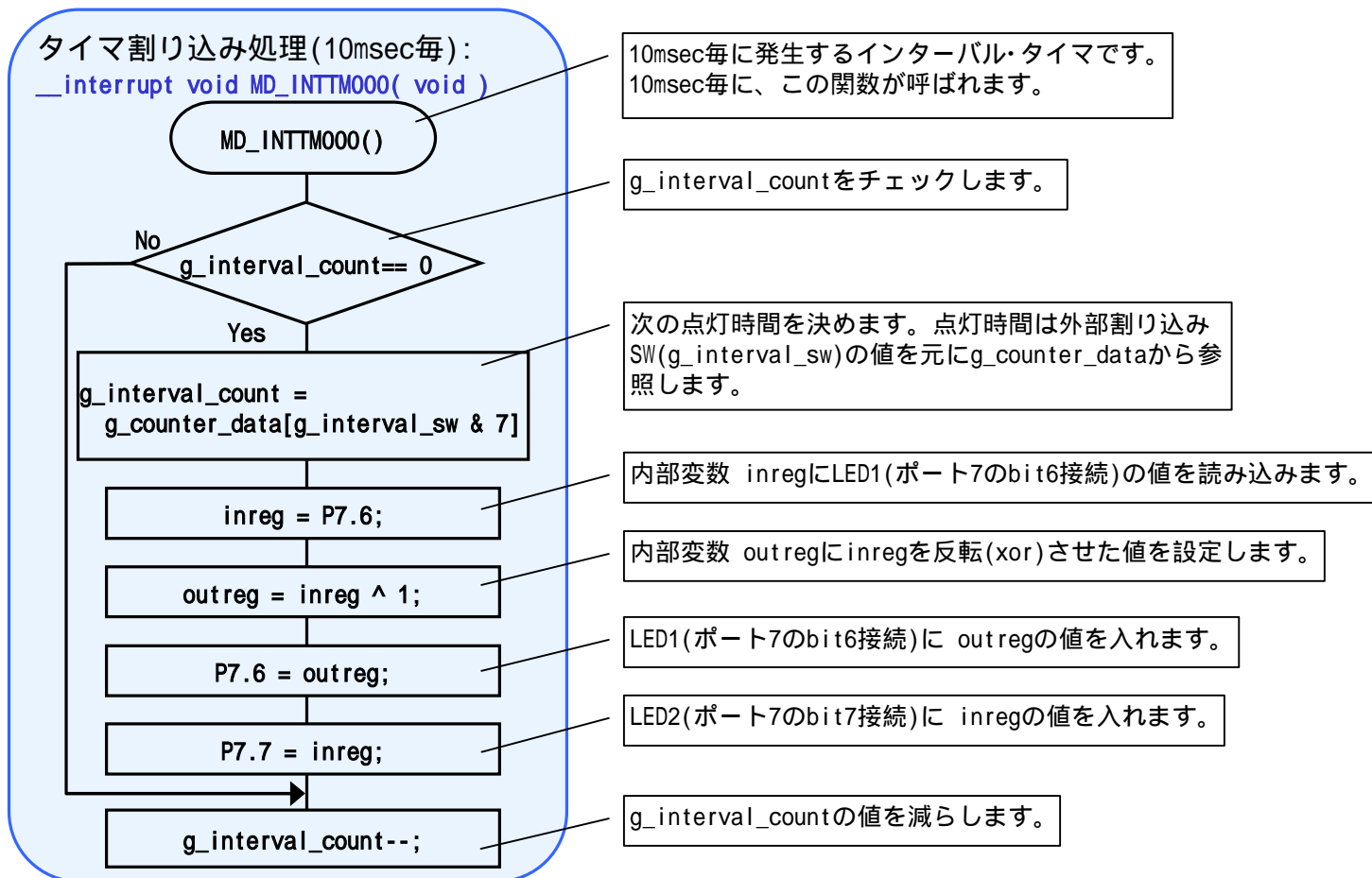
10msec毎に呼ばれる関数内でg\_interval\_countを-1して、それが0になった場合にLEDを点滅させる処理をしています。最初はg\_interval\_countが1なので2回に1回LEDが点滅します。するとP76,P77に接続されたLEDが20msec毎に交互に点滅します。はじめは20msec毎ですが、外部SWを押下することによって点滅速度が遅くなります。点滅速度は 20msec, 40msec, 80msec, 160msec, 320msec, 480msec, 640msec, 1280msecを繰り返します。

### グローバル変数の説明

- g\_interval\_sw: 外部SWを押下すると変化する。0~7の値になる。
- g\_onetime\_sw: この値が0の時に外部割り込み処理を有効とします。
- g\_interval\_count: 10msec毎に -1されるカウンタ。0の時にLED点灯を変更します。
- g\_counter\_data[8]: 点灯速度を決めるデータ。8段階のスピードを設定します。



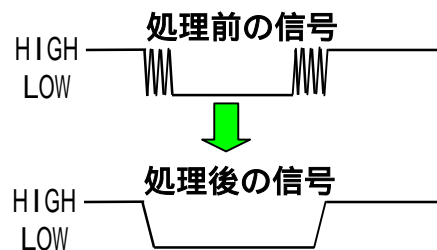
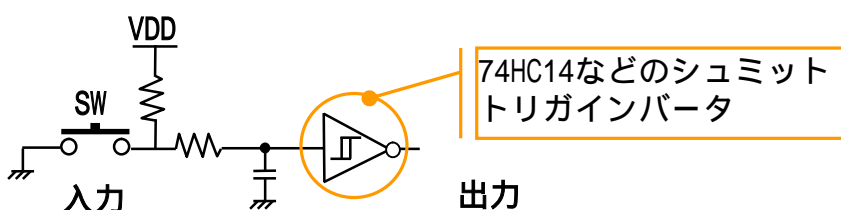
# プログラムの説明



## 💡ワンポイント

### チャタリングについて

SWを押した時、実際には機械的な接点では瞬間的にON->OFFを繰り返します。これを「チャタリング」と言います。QB-78K0KF2-TBでは、ハードウェア回路にはチャタリング防止を行っていないのでチャタリング防止をソフトウェアで行う必要があります。ハードウェアで、チャタリング防止する方法としてシュミットトリガインバータ(74HC14など)を使うのが一般的です。この方法はノイズ除去にも使われます。

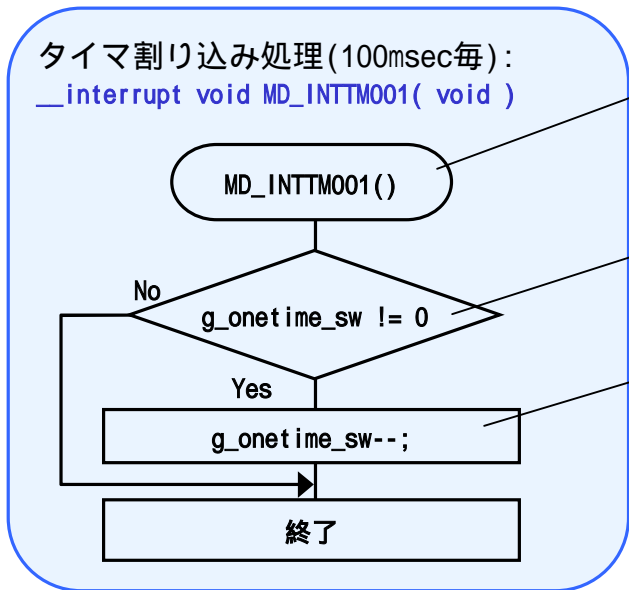






# プログラムの説明

タイマ割り込み処理 (100msec毎):  
`__interrupt void MD_INTTM001( void )`

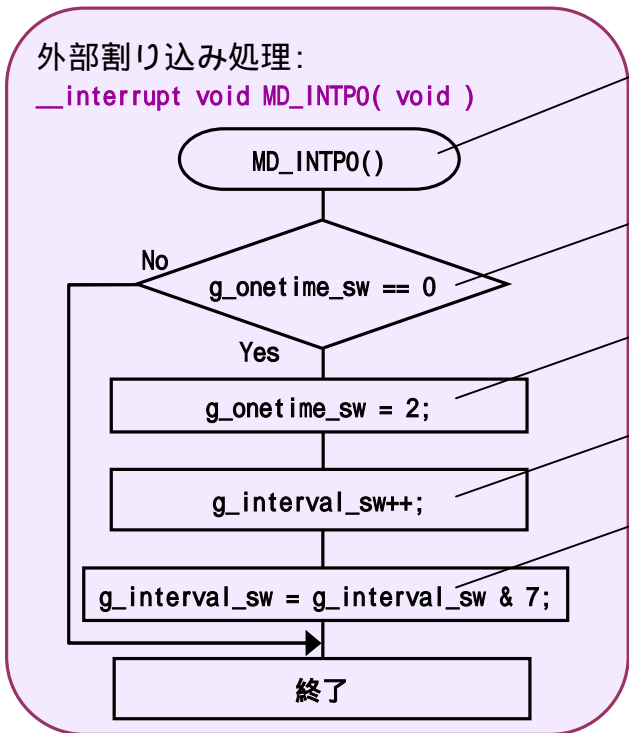


100msec毎に発生するインターバル・タイマです。一度押したSWを200msec以内は無視するための処理です。

g\_onetime\_swをチェックします。

g\_onetime\_swの値を減らします。

外部割り込み処理:  
`__interrupt void MD_INTPO( void )`



SWが押下された時に呼ばれます。

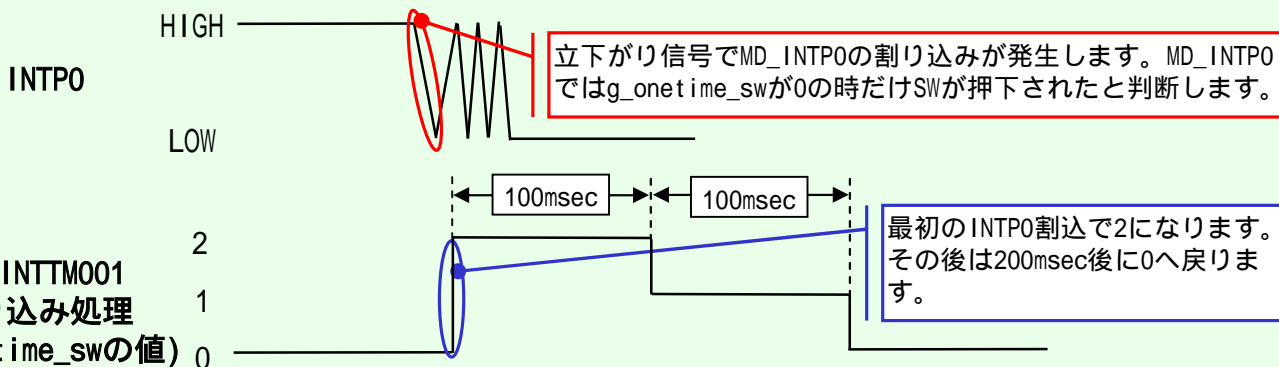
g\_onetime\_swをチェックします。

g\_onetime\_swに値をセットします。

g\_interval\_swを増やします。LEDの点灯時間を参照する時に使用します。

LED点灯時間のデータは8個なので参照する値を調整します。

外部SWが押下されると g\_onetime\_sw に2が代入されます。g\_onetime\_swは100msec毎に-1されます。1度外部SWが押下されるとg\_onetime\_swが0になるまで200msecかかるわけです。MD\_INTPO()の処理は、このg\_onetime\_swが0でない限りLED点滅のタイミングを変更しません。ゆえにチャタリング発生し短い時間に何度MD\_INTPO()が呼ばれても200msec以上の時間を空けない限りLED点滅タイミングは変更されません。



# プログラムの説明



プログラムの説明は以上です。追加するコードの小さい事に驚いたのではないのでしょうか？デバイスドライバ・コンフィギュレータ「Applilet2」を使えば、誰でも手軽にマイコンの開発ができる事でしょう。Appliletの情報については、下記URLよりアクセスして下さい。

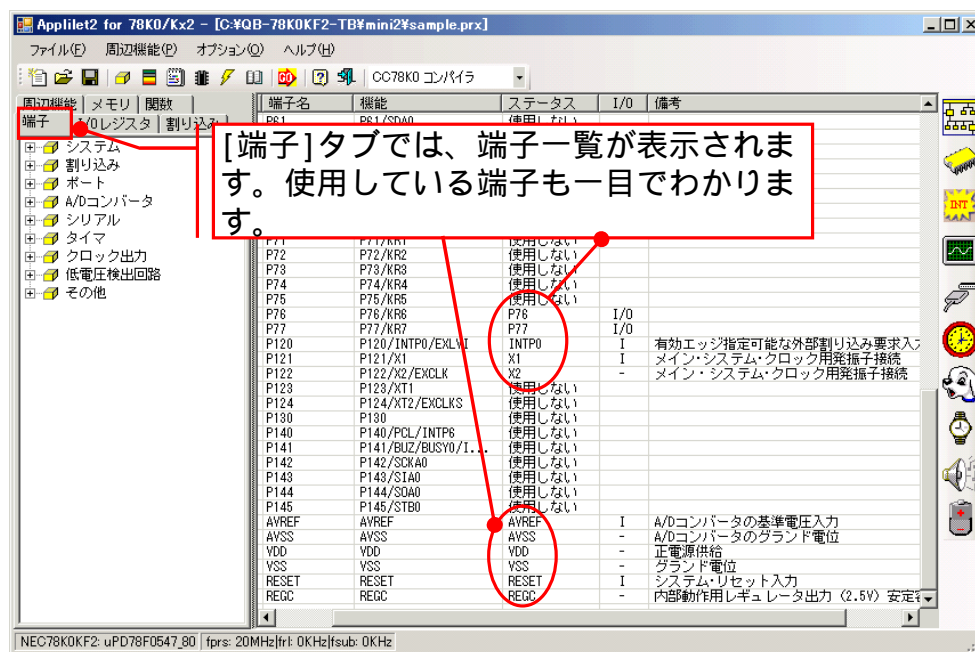
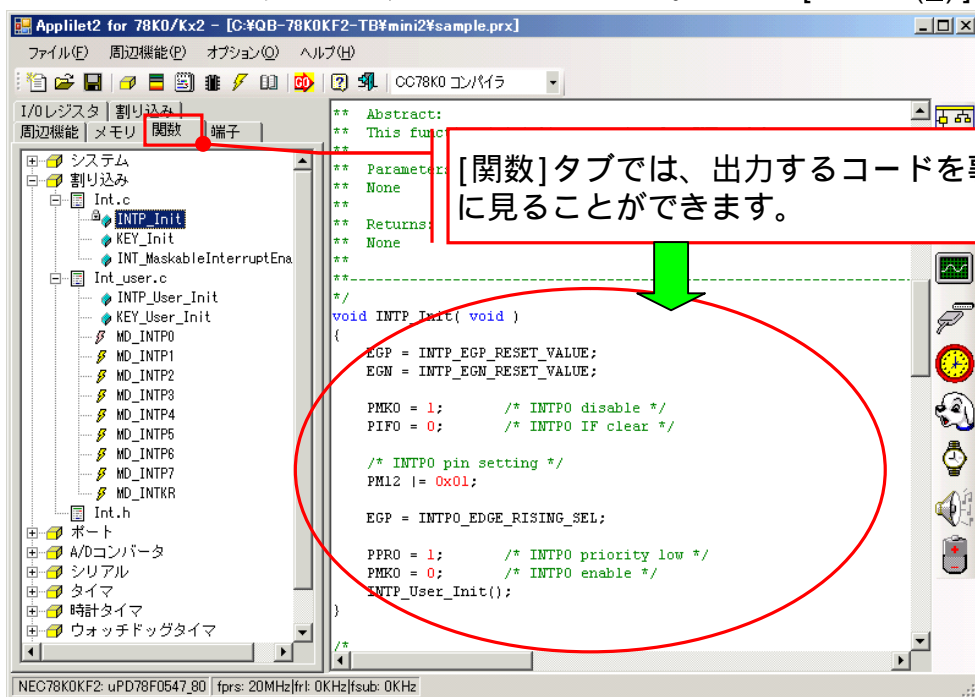
<http://www.necel.com/micro/ja/development/asia/applilet/>

## ワンポイント

### Applilet2の機能ついて

タブ切り替えを行うことで、機能が変わります。詳細は[ヘルプ(H)]

[ユーザーズ・ガイド(G)...]を参照して下さい。





# プログラムの作り方

ここでは、プログラムを作成する時の諸注意を挙げます。ここにあげる内容が全てではありませんが、プログラムを作る上での参考にして下さい。また、有名なコーディング規約として、自動車関連ソフトウェアの業界団体(MISRA)が策定した組込みC言語用ガイドライン「MISRA-C」があります。参考にして下さい。

## 設計時に考えること

### [マイコン資源の見積]

入出力ポートの使い方、タイマ割り込み、などハードウェア資源とソフトウェアの資源について充分に見積って下さい。例えばA/D変換を行うタイミングとして100us毎で処理するのか10ms毎でも充分なのか、処理方法を考えて下さい。

### [ソース・ファイルの構成]

Applilet2が必ず生成するソース一覧

- option.asm / inc オプション・バイトの設定などの定義
- systeminit.c マイコン初期化
- System.c / h マイコン初期化
- System\_user.c ユーザー追加用初期化関数
- macrodriver.h Applilet2用定数定義
- user\_define.h ユーザー追加用定義ファイル
- main.c メイン関数

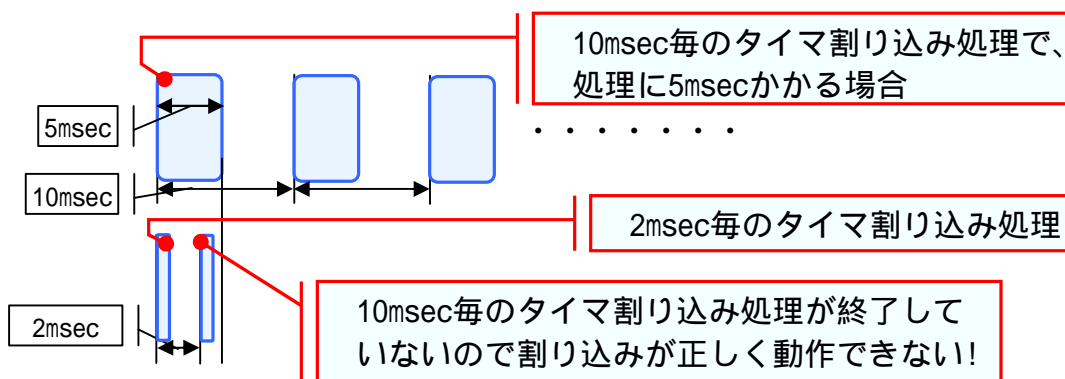
その他にマイコン周辺機能を使う場合(ポート、外部割り込み、タイマを追加した場合)

- Port.c / h 入出力ポート定義
- Int.c / h 外部割り込み
- Int\_user.c ユーザー追加用外部割り込みハンドラ
- Timer.c / h タイマ処理
- Timer\_user.c ユーザー追加用タイマ割り込みハンドラ

Applilet2が必要なソースを生成しますので、基本的には xxxx\_user.c にプログラムを追加すれば大丈夫です。

### [割り込みハンドラ内では処理速度を優先に考える]

割り込みハンドラ処理中は、さらに優先度の高い割り込み処理が入らない限り、他の処理は停止してしまいます。割り込みハンドラは、処理速度を優先させて下さい。そのため処理としては、割り込み要因を解除し、割り込み処理自体はmainから制御します。mainへ処理を渡すために割り込みハンドラ内ではグローバル変数などのフラグを立てて、それをmainで処理するのが望ましいです。割り込み処理を考える時は、**割り込みの優先度**にも注意して設計して下さい。





# プログラムの作り方

## コーディング時に考えること

### [名前の付け方を考える]

define、マクロ、変数などの命名をきちんと定義すべきです。

例えば

- define、マクロ名は全て大文字にする。
  - グローバル変数には[g\_]、グローバルポインタ変数には[gp\_]を接頭語としてつける。
- また enum、構造体の定義も解るように命名定義しましょう。

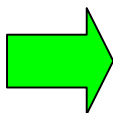
**命名も重要なコーディング**と考えて下さい。

### [関数の作り方で注意したい事]

- ・ 入力と出力は1箇所にする。

なぜ、入力と出力は1箇所にするのか。例えば、こんな関数があったとします。

```
void function( UCHAR para )
{
    size_t *size;
    size = malloc(100);
    if ( size == NULL )
        return;
    if ( para == 0 )
        return;
    :
    :
    free( size );
}
```



```
void function( UCHAR para )
{
    size_t *size;
    size = malloc(100);
    if ( size != NULL )
    {
        if ( para != 0 )
        {
            :
            :
        }
        free( size );
    }
}
```

上の行のreturnは、正しく動作するがここでのreturnは領域確保した\*sizeを開放していないのでメモリリークが発生する。

階層は深くなりますが、メモリリークを防ぐ意識の書き方になります。

他の利点としてデバッグ時は、入口と出口だけチェックだけでよいのでデバッグ効率があがります。

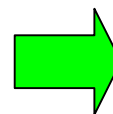
- ・ 多くても1つの関数を100行程度にする。

例えば、車の構成を考えて下さい。車は小さい部品で構成されているとは思いませんか？小さい部品一つ一つがテストされ組み上がっています。プログラムも同様です。全体で1万行のプログラムに1000行を超えるような関数があるとすれば、かなりアンバランスです。関数一つにしても可読性、移植性を考えるべきです。

- ・ ハードウェアのアクセスは最小限にする。

下記の例は、一見問題無いように思われますが、潜在的なバグがあります。

```
void function()
{
    UCHAR a = 0;
    if ( P4.0 == 0 )
    {
        a = 1;
    }
    else if ( P4.0 == 1 )
    {
        a = 2;
    }
}
```



```
void function()
{
    UCHAR a = 0, port = P4.0;
    if ( port == 0 )
    {
        a = 1;
    }
    else
    {
        a = 2;
    }
}
```

このif命令を実行している時点ではP4.0=1の値で次のelses if命令を実行する時にはP4.0=0の値の場合、どのケースにも該当しないでa=0の場合がある。

ハードウェアのアクセスを最小にし、更にロジック的にありえないelse if命令を消去しました。



# プログラムの作り方

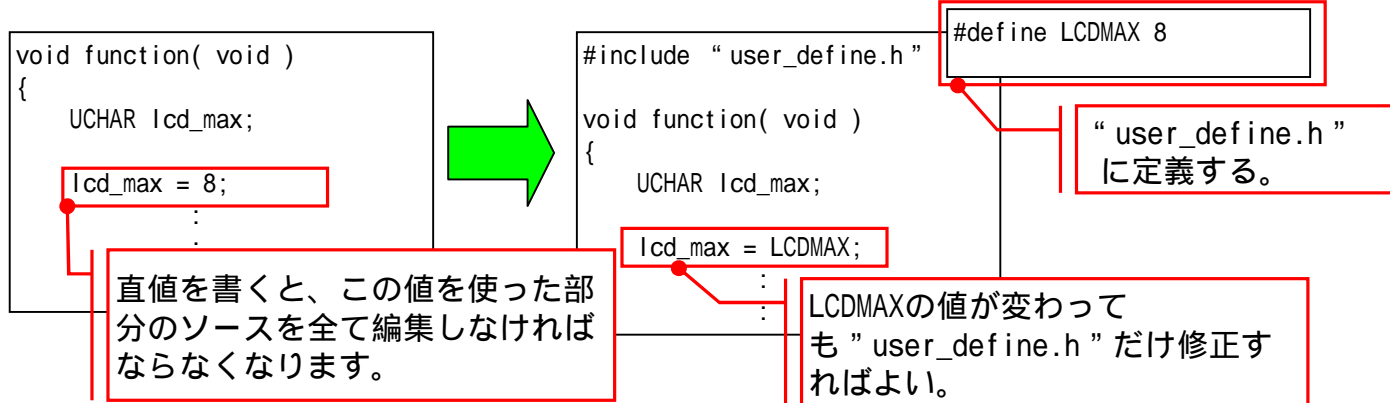
## コーディング時に考えること

### [ソースを見やすくするために]

- ・直値を書かない。

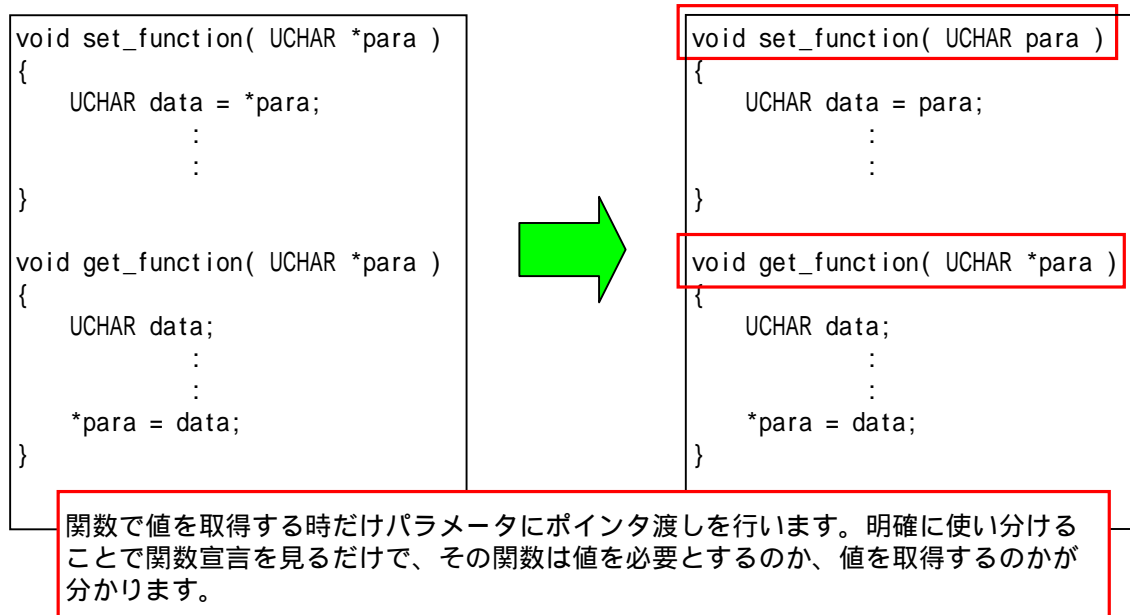
数値をソース中に書く場合に、直接的な値を書かないようにします。

値を直接書くと修正する箇所が複数発生するため、修正ミスが発生する確率が高くなります。



- ・関数の引数宣言について

通常、関数引数を扱う場合、引数で値を返す場合はポインタ渡しを行います。関数に値を渡すだけの場合には、ポインタ渡しを使用をしないようにします。このように、明確に書き方を分けておけば、関数の宣言だけで引数の意味の判断が可能です。



- ・1ファイルの行数を長くしない。

メンテナンスなどを考えると多くても3000行程度が妥当です。それ以上になるときは、機能を分割し、ファイルも分割すべきです。昨今の開発は、多人数で行うのが当然で、ソースファイルそれぞれに、バージョン管理ソフト(RCS/CVS/VSSなど)を使うことを考えれば、当然ファイルも分割しなければなりません。



# プログラムの作り方

## マイコン開発特有の注意点

### [マイコン資源関係]

- 固定データにはconstを書く。  
単に定義したデータはRAMに確保される場合があります。マイコンのRAM領域はROMに比べて少ないので変更しない参照するだけのデータには、必ずconstをつけるようにします。

```
const UCHAR g_font_numeric[] =
{ /* font data 0 - 9 */
  0x00, 0x0E, /* 0 */
  0x00, 0x04, /* 1 */
  0x00, 0x0E, /* 2 */
  0x00, 0x0E, /* 3 */
  0x00, 0x02, 0x06, 0x0A, 0x12, 0x1F, 0x02, 0x02 /* 4 */
  , 0x00, 0x1F, 0x10, 0x10, 0x1E, 0x01, 0x01, 0x1E /* 5 */
  , 0x00, 0x0F, 0x10, 0x10, 0x1E, 0x11, 0x11, 0x0E /* 6 */
  , 0x00, 0x1F, 0x01, 0x02, 0x04, 0x04, 0x04, 0x04 /* 7 */
  , 0x00, 0x0E, 0x11, 0x11, 0x1E, 0x11, 0x11, 0x0E /* 8 */
  , 0x00, 0x0E, 0x11, 0x11, 0x0F, 0x01, 0x01, 0x0E /* 9 */
};
```

この様なフォントデータは、変更しないのでconst宣言してROM領域に置く。

- レジスタから値を取得する際はvolatileをつける。  
volatile(ボラタイル)とは最適化しないという宣言です。  
例えば、レジスタなどハードウェアからの情報を取得してグローバル変数などに代入する場合にvolatileをつけます。

Applilet2でシリアルを使用した時のソースコード「Serial.c」

```
/*
*****
** Include files
*****
*/
#include "macrodriver.h"
#include "user_define.h"
#include "Serial.h"
/* Start user code for include definition. Do not edit comment generated here */
/* End user code for include definition. Do not edit comment generated here */

/*
*****
** Global define
*****
*/
volatile UCHAR*      gUartOTxAddress; /* uart0 transmit buffer address */
volatile USHORT      gUartOTxCnt;    /* uart0 transmit data number */
volatile UCHAR*      gUartORxAddress; /* uart0 receive buffer address */
volatile USHORT      gUartORxCnt;    /* uart0 receive data number */
volatile USHORT      gUartORxLen;    /* uart0 receive data length */
/* Start user code for global definition. Do not edit comment generated here */
/* End user code for global definition. Do not edit comment generated here */
```

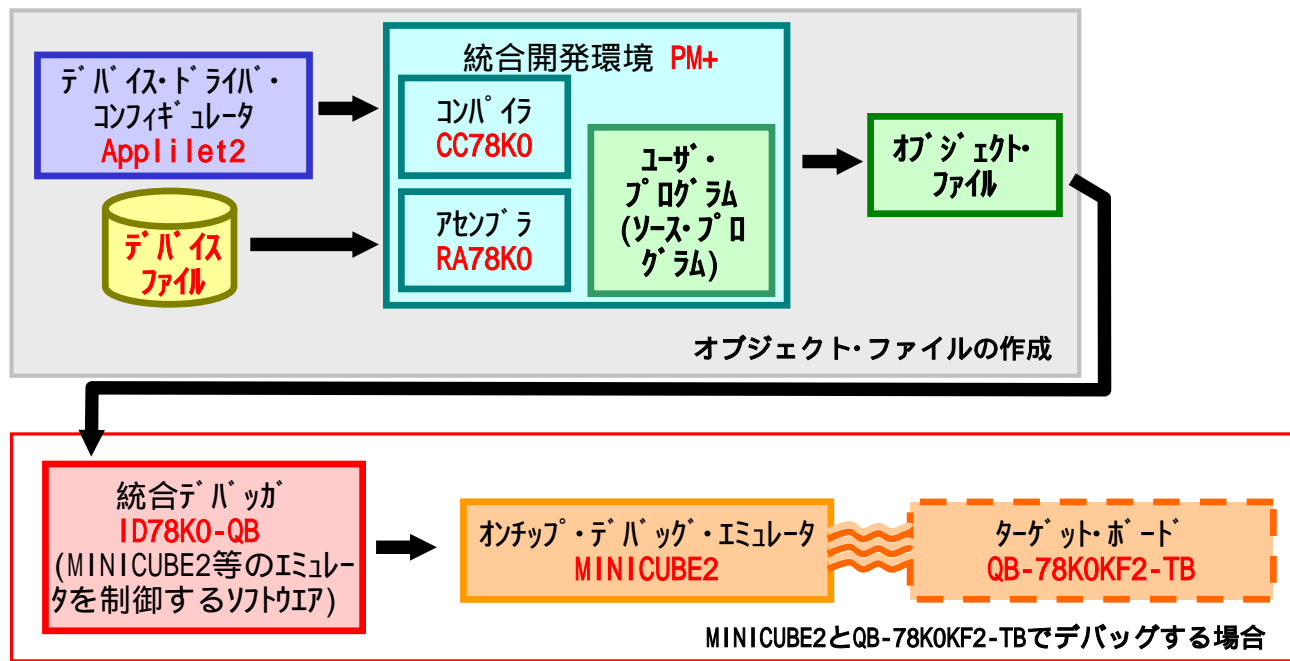
基本的なプログラムの作り方については以上です。弊社WEBページにはサンプルプログラムも掲載しています。下記URLも参考にしてください。

<http://www.necel.com/micro/ja/designsupports/sampleprogram/>

# デバッグの基本

[デバッグしたい]の章では、ハードウェアを使ってデバッグする時に必要な基本項目を学習します。  
[動かしてみよう]の章で使用したシステム構成(MINICUBE2 + QB-78K0KF2-TB)で統合デバッガ(ID78K0-QB)の使い方を学習します。学習時間は30分です。

## この章でのデバッグ時のシステム構成



## 次ページより下記の項目順に説明します

- ・ デバッガの基本画面
- ・ ブ레이크ポイントの設定
- ・ ステップ実行
- ・ 変数表示
- ・ メモリ表示
- ・ レジスタ表示

## 💡 ワンポイント

### 統合デバッガ(ID78K0-QB)について

ここで説明するデバッガの機能は、オンチップ・デバッグ・エミュレータ(MINICUBE2)を使った場合です。もっと高機能なハードウェア・エミュレータ「IECUBE」を使えば豊富なデバッグ機能(トレース機能、イベント間時間測定、カバレッジ機能、疑似エミュレーション機能)があります。

「IECUBE」詳細については下記URLにアクセスして下さい。

<http://www.necel.com/micro/ja/development/asia/78k0kx2/iecube.html>

# デバッグの基本画面

## a. デバッガを起動します。

[動かしてみよう] の章 [デバッグの起動] を参考にしてください。2回目以降に起動する場合、プロジェクトが保存されていれば、自動的にオブジェクト・ファイルがダウンロードされます。

## b. ファイラー一覧を表示します。

メニューの[ブラウズ(B)] [その他(E)] [2 LIST]を選択して下さい。プロジェクトに登録されているファイル一覧が表示されます。

The screenshot shows the debugger interface for 'ID78K0-QB : sample.prj'. The 'Source (main.c)' window is open, showing C code. A red box highlights the menu path: 'ブラウズ(B)' > 'その他(E)' > '2 List'. A red callout box points to this path with the text: 'メニューより[List]を選択する。' (Select [List] from the menu).

Below, the 'List' window is shown, displaying a tree view of the project files:

- main.lmf
  - Int.c
  - Int\_user.c
  - main.c
  - option.asm
  - Port.c
  - System.c
  - System\_user.c
  - systeminit.c
  - Timer.c
  - Timer\_user.c

A red box highlights the 'List' window with the text: '[List]ウィンドウが表示され、プロジェクトに登録されているCソースが表示されます。なお、初めに表示される時は[List]ウィンドウと[Source]ウィンドウが重なって表示されるので適宜、ウィンドウを移動して見やすく表示して下さい。' (The [List] window is displayed, and C sources registered in the project are shown. Note that when first displayed, the [List] window and [Source] window overlap, so please move the window appropriately to make it easier to see.)

## 💡ワンポイント

### デバッガ(ID78K0-QB)の環境ファイルについて

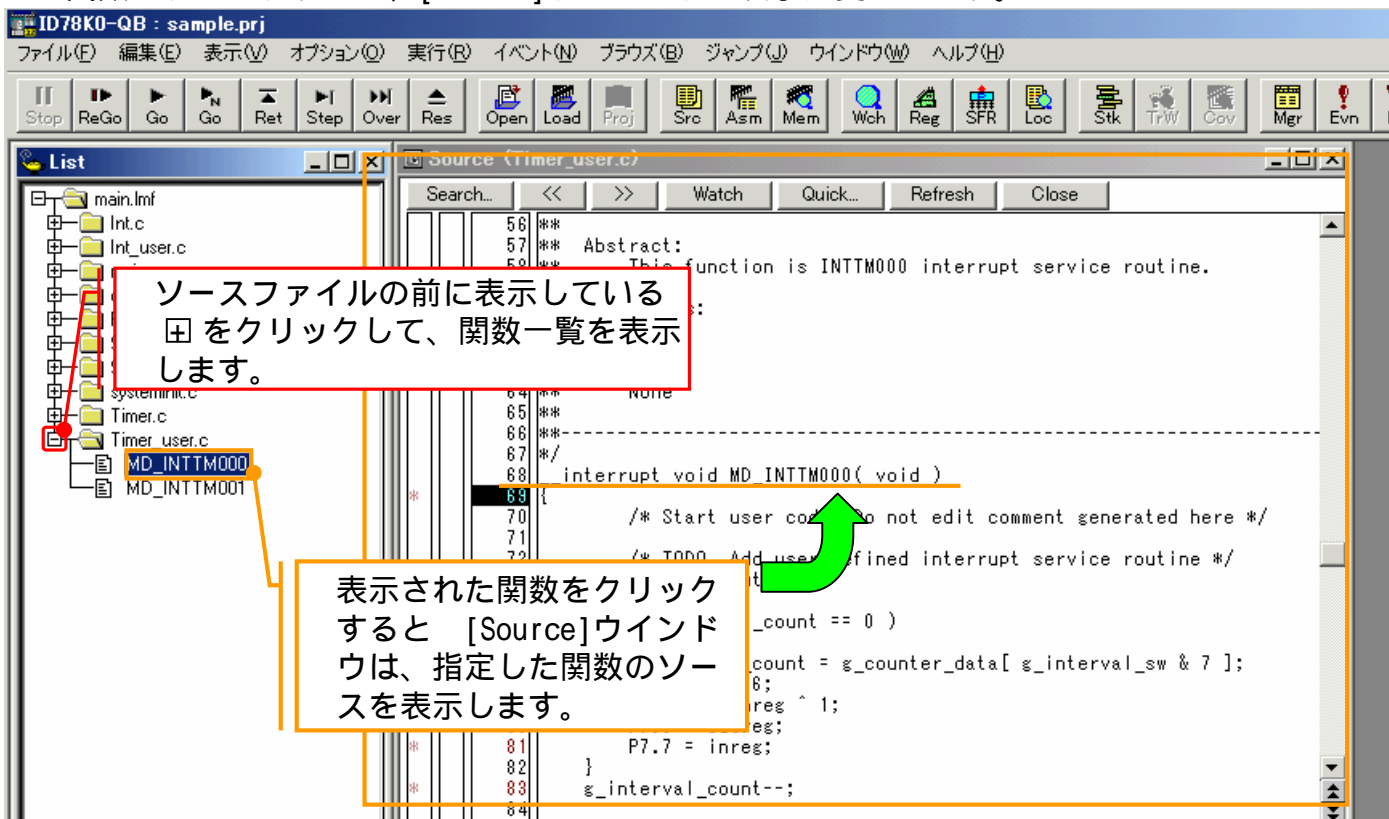
設定したデバッガの環境は、「プロジェクト名.pri」ファイルに保存されます。サンプル・プログラムのプロジェクト名は[sample]なので、「sample.pri」へ保存されます。デバッガの環境を複数持ちたい時などは、この「プロジェクト名.pri」を名前を変更して保存して下さい。また、2回目以降のデバッグの起動は「.pri」ファイルを使用するので、プログラムのダウンロードも自動的に行われます。



# デバッガの基本画面

## c. ソース上の、ある関数に移動したい時。

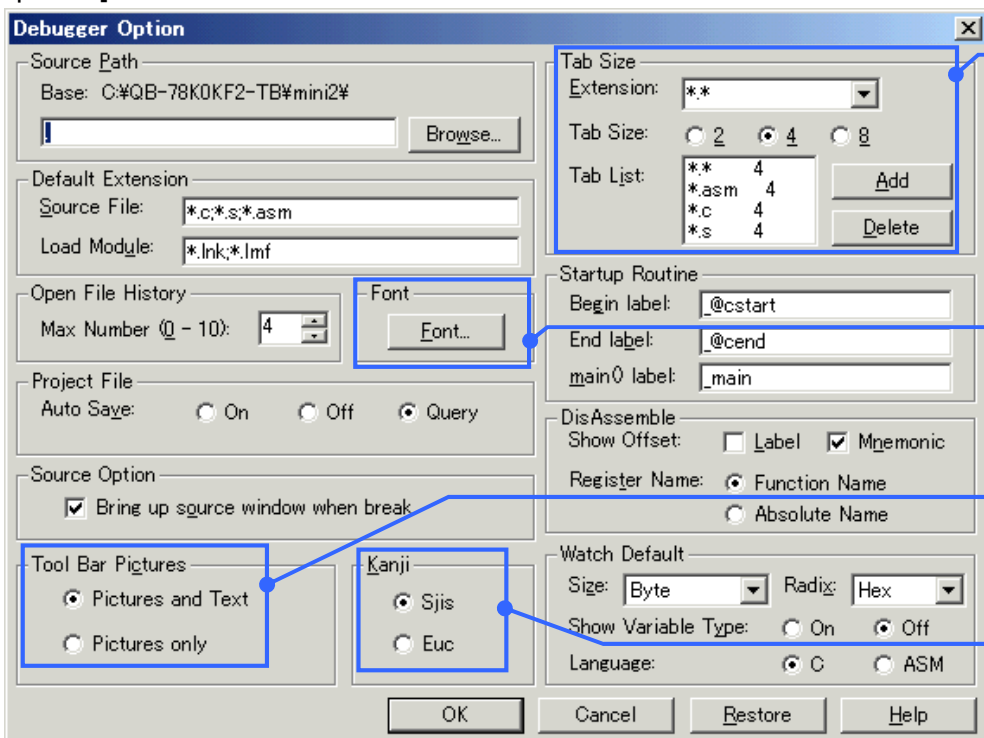
[List]ウインドウのソースファイルをクリックして関数一覧を表示します。関数が表示されたらその関数をクリックすると、[Source]ウインドウの表示が変わります。



## 💡ワンポイント

### デバッガ(ID78K0-QB)のオプションについて

メニューの[オプション(O)] [デバッガ・オプション(G)...] で表示の変更ができます。[Debugger Option]ダイアログが表示されますので、各種設定して下さい。



ソース表示するタブのインデントサイズを設定。

表示する際のフォントを設定。

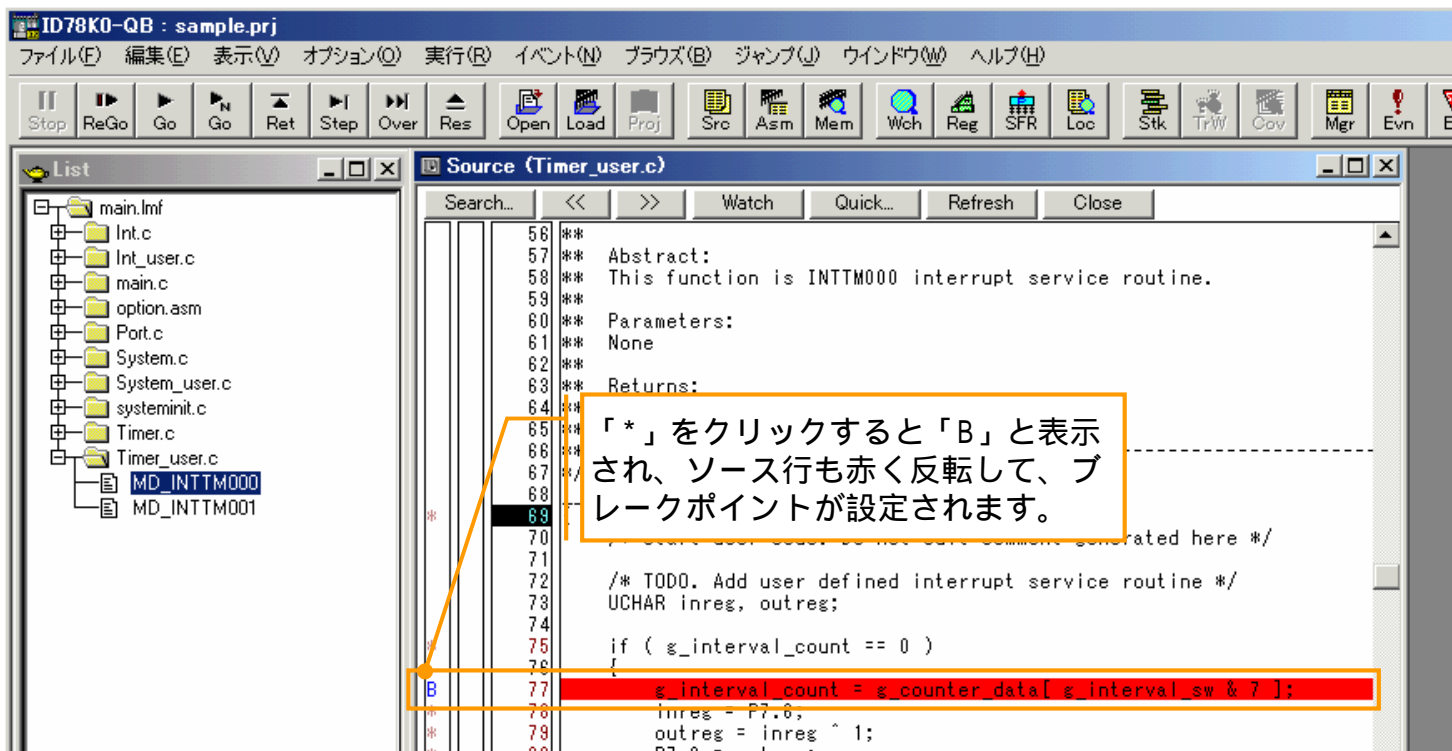
メニューのアイコンに文字の説明をつける時の設定。

漢字コードの設定。

# ブレークポイントの設定

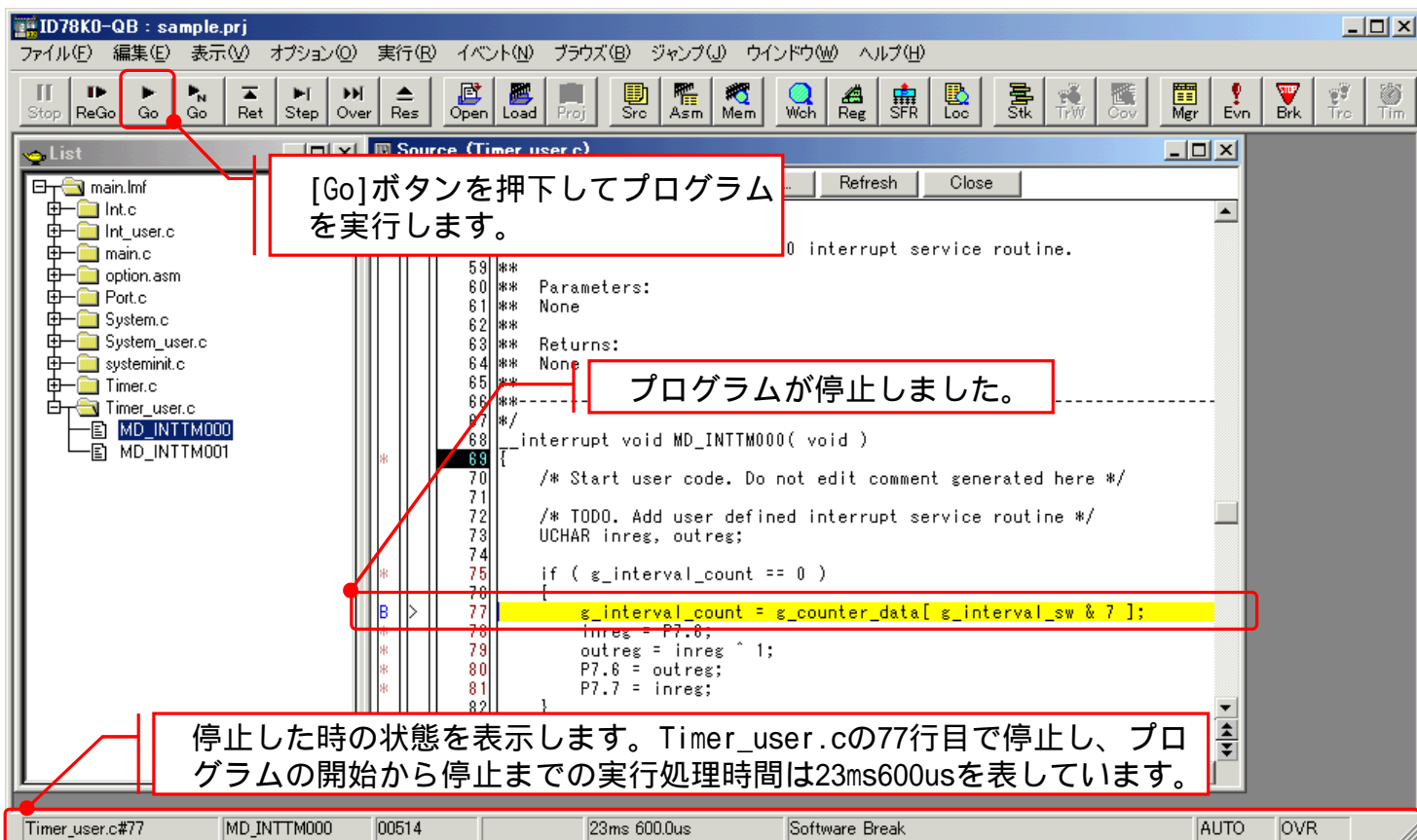
## d. ブレークポイントを設定したい画面を表示します。

ソースを表示させ、行番号前の「\*」をクリックします。すると、クリックした行が赤に反転表示され、ブレークポイントが設定されます。



## e. 実行してブレークするか確認します。

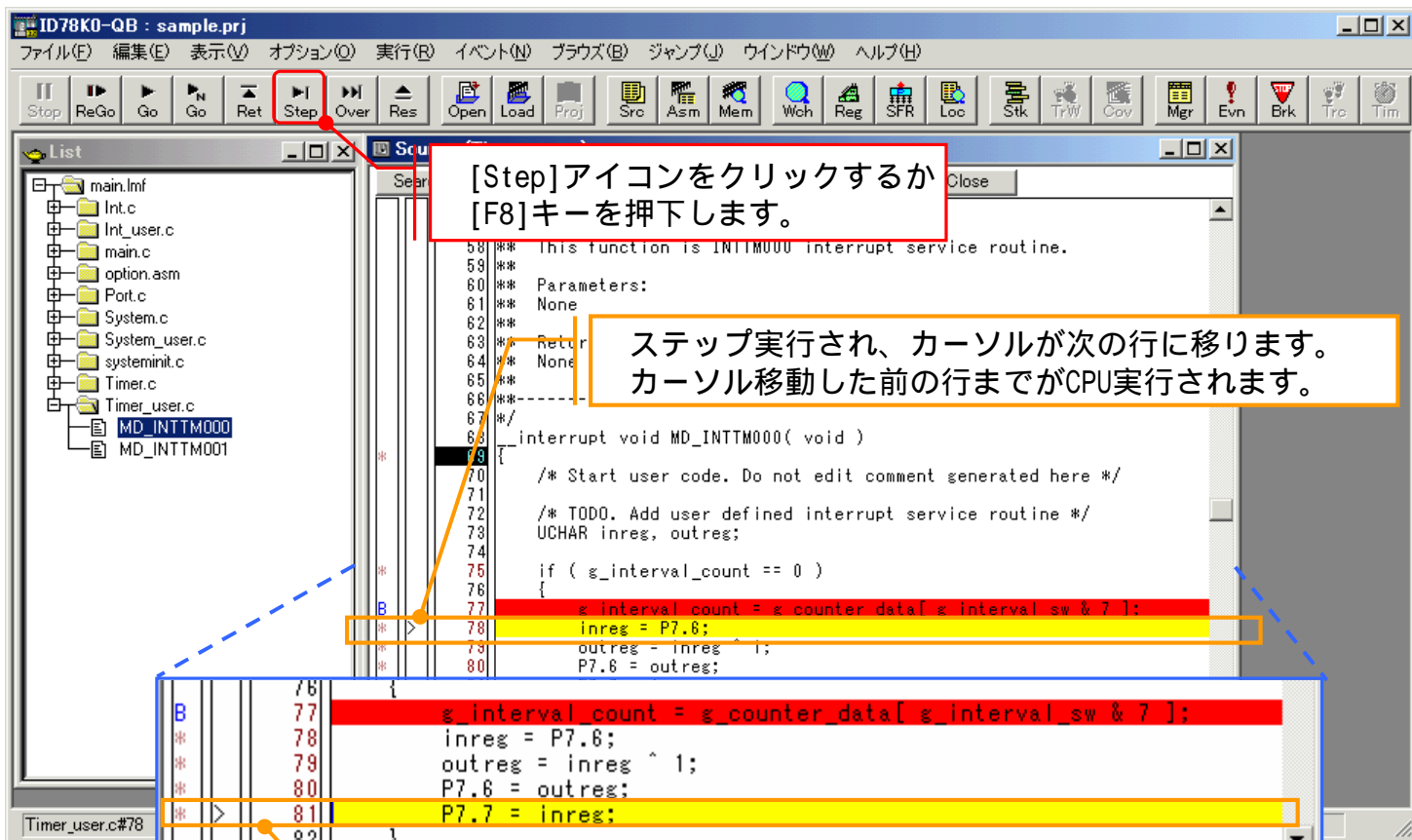
[Go]ボタンを押下して実行し、プログラムが停止することを確認します。



# ステップ実行

## f. ブレークした場所からステップ実行します。

停止している時に、[F8]キーを押下または、[Step]アイコンをクリックしてステップ実行します。ステップ実行して 81行目までカーソルを移動して下さい。



ステップ実行を繰り返し、81行目まで移動します。この場合は80行目の「P7.6=outreg;」までが実行されるのでP7.6に接続されているLED1が点灯します。左図のようにLED1が点灯することを確認します。



## 💡ワンポイント

### ステップ実行の種類について

ステップ実行には色々な種類があります。[ステップ・イン(F8)]、[ネクスト・オーバー(F10)]、[リターン・アウト(F7)]など他にも[カーソル位置まで実行]、[カーソル位置から実行]もあります。

- ・ [ステップ・イン(F8)]

関数呼び出しがあった場合、その呼び出し関数内もステップ実行します。全てのシーケンスを確認したい時に使用します。

- ・ [ネクスト・オーバー(F10)]

関数呼び出しがあった場合、その呼び出し関数は一度に処理されます。ブレークしたソース内のみをデバッグしたい場合に便利です。

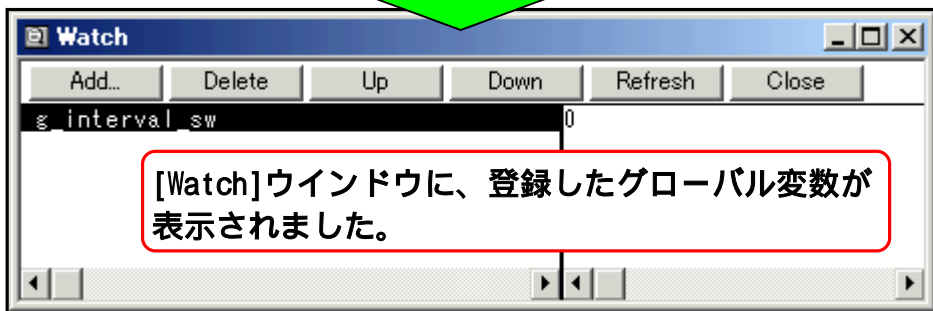
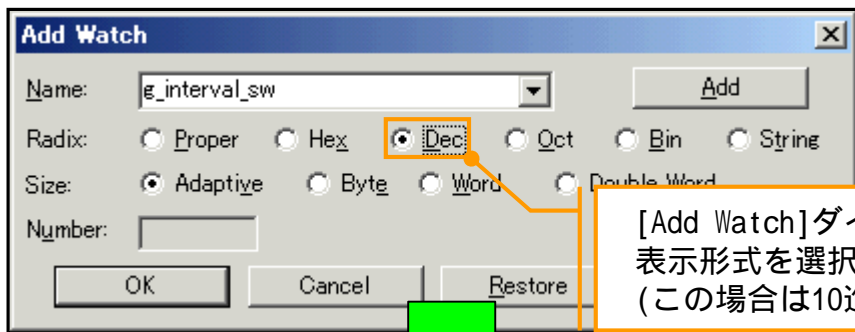
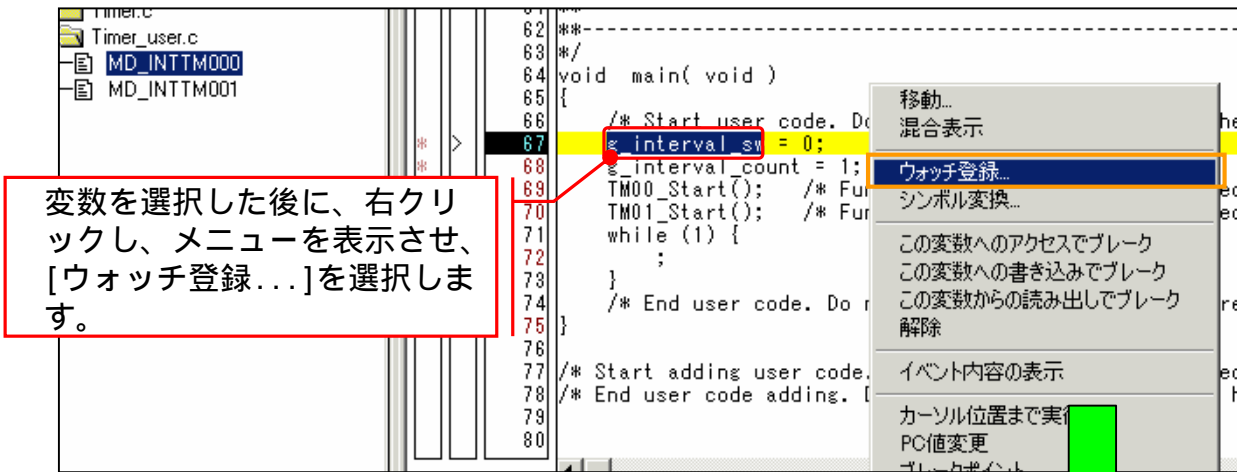
- ・ [リターン・アウト(F7)]

ブレークした関数内をリターンするまで一度に処理します。


# 変数表示

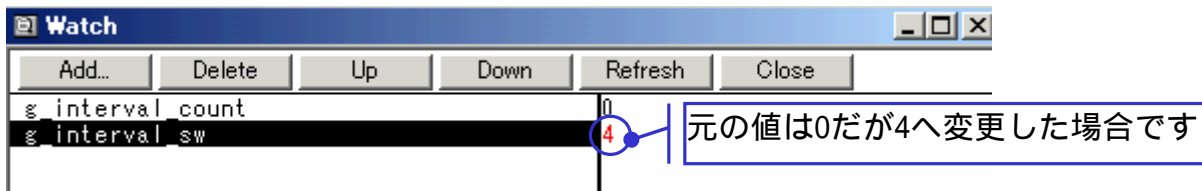
## g. グローバル変数の表示。

グローバル変数を表示したい場合は、[Source]ウインドウより表示させたい変数を選択して、ウォッチ登録します。[Add Watch]ダイアログより表示形式を選択すればOKです。



## 💡 ワンポイント

Watchウインドウに表示している変数の書き換えについて  
 プログラム実行中にメニューの[実行(R)] [ストップ(S)]またはStopボタン  でプログラムを停止させた時Watchウインドウに表示されている変数の値を変更することができます。



# 変数表示

## h. プログラム実行中のグローバル変数の表示方法。

プログラム実行中にグローバル変数を表示したい場合、メニューの[オプション(O)] [拡張オプション(X)...] を選び、[Extended Option]ダイアログを表示します。 [Extended Option]ダイアログで設定します。

[オプション(O)] [拡張オプション(X)...] を選びます。

チェックします。

変数表示の表示はデフォルトで500msec毎に更新されます。表示を更新する時間を100msec単位指定で100 ~ 65500まで指定できます。

[Extended Option]ダイアログを設定した後に、[Watch]ウインドウを開き「RRM Setting」ウインドウを開きます。変数のアドレスが表示されるので[Set]ボタンで設定した後、[Close]して下さい。

右クリックして、[RRM設定...]を選び「RRM Setting」ウインドウを開きます。

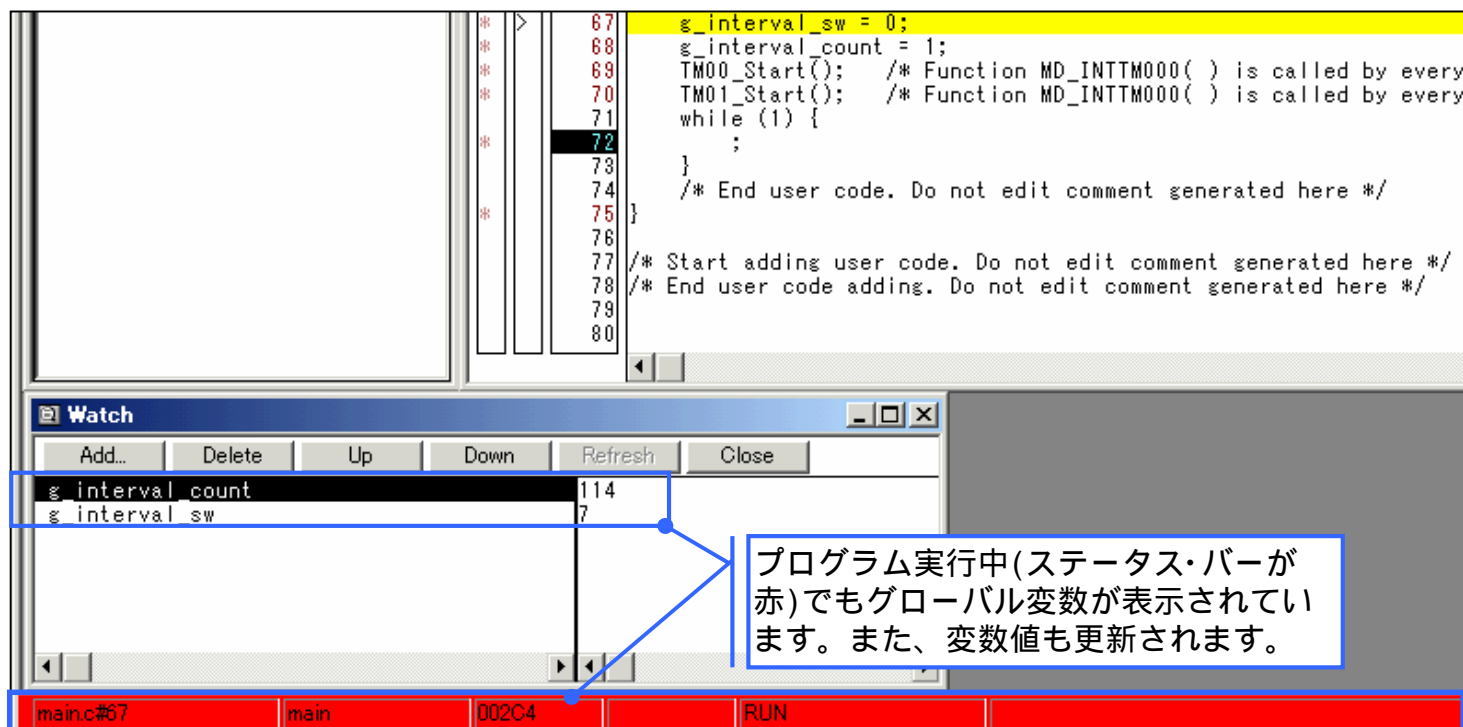
Address	Size	Symbol
0xfb8a	1	_g_interval_sw

**g\_interval\_sw**のアドレスが表示されます。変数に応じて必要なサイズを入力して下さい(この場合はchar型なので1Byte)

[Set]ボタンを押下すると設定されます。[Close]ボタンでウインドウを閉じます。

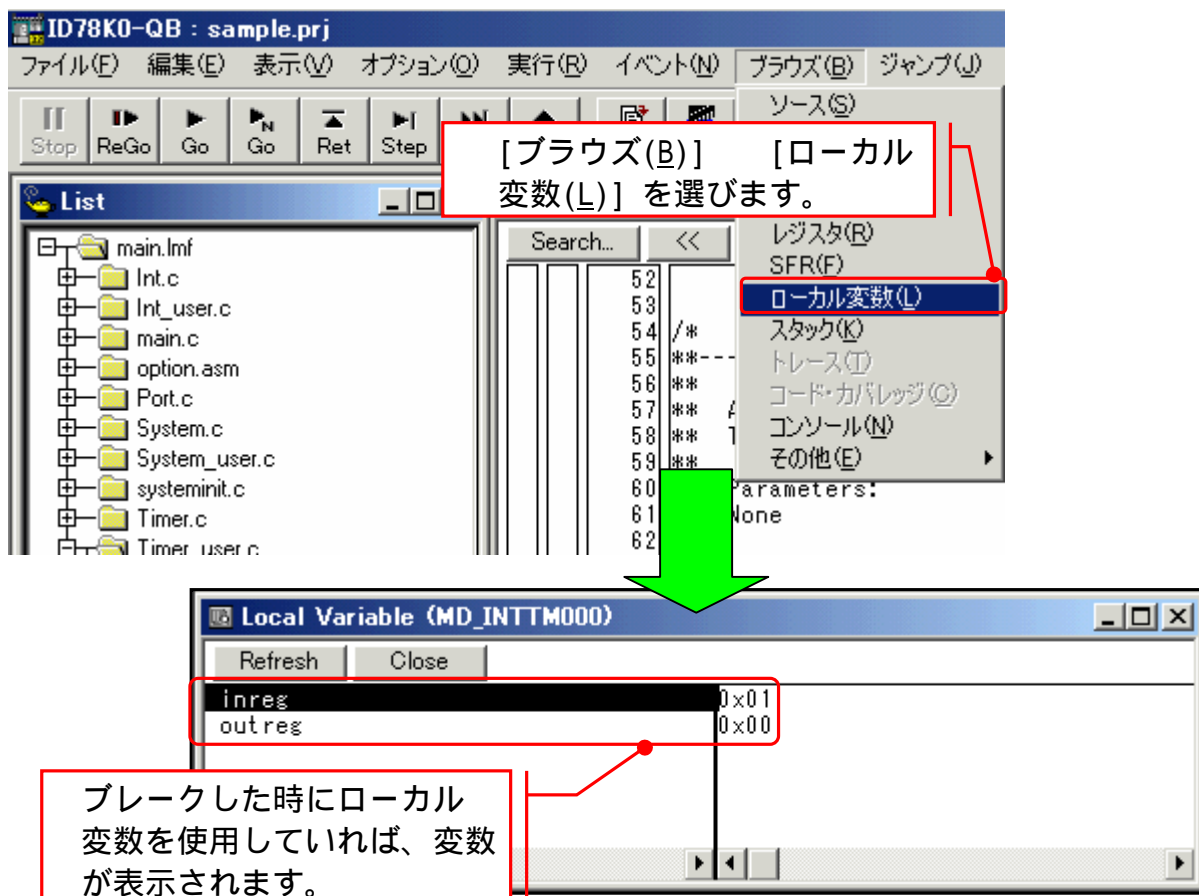
# 変数表示

## i. プログラム実行中のグローバル変数表示画面。



## j. ローカル変数の表示方法。

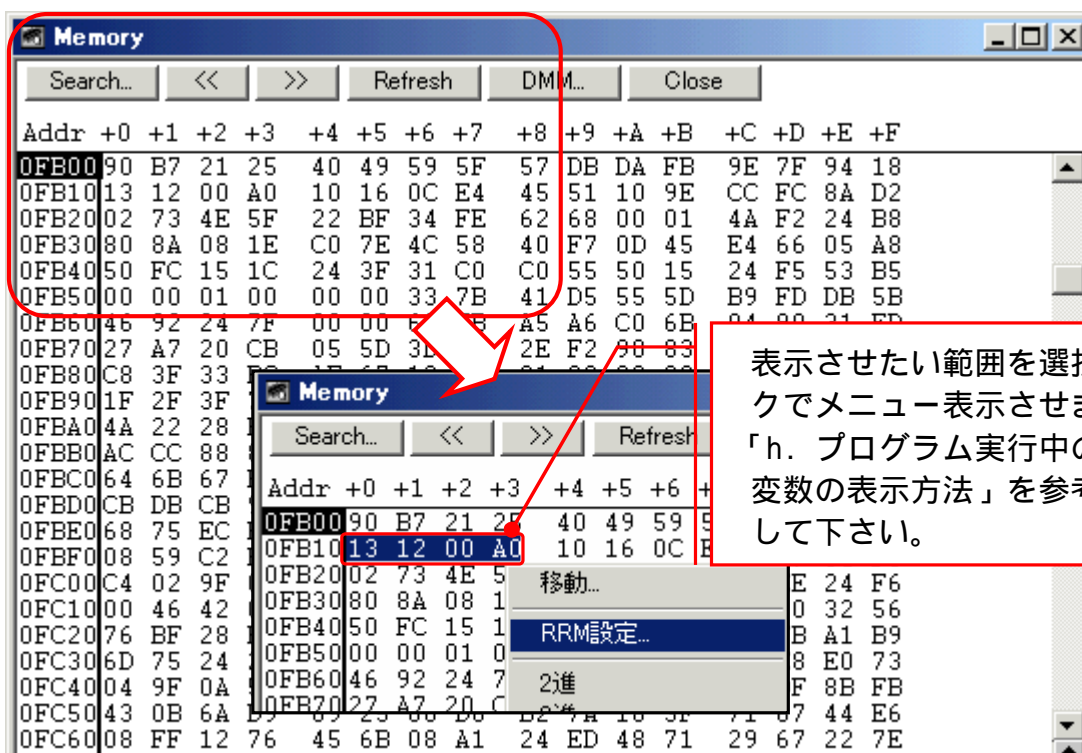
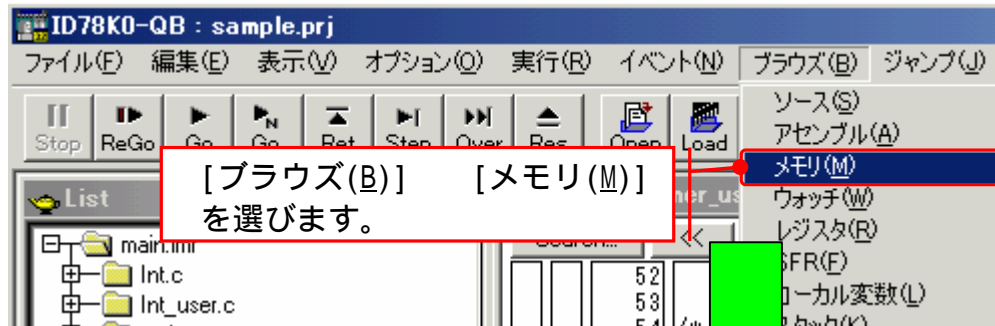
メニューの[ブラウズ(B)] [ローカル変数(L)] を選ぶと、[Local Variable]ウィンドウが表示されます。ローカル変数は実行中には表示されません。ブレークした時のみ表示します。



# メモリ表示

## k. メモリ表示方法。

メニューの[ブラウズ(B)] [メモリ(M)] を選ぶと、[Memory]ウィンドウが表示されます。プログラム実行中は通常、メモリ内容を表示できません。しかし、一部であれば[RRM設定...]を行うことにより、プログラム実行中でもメモリ内容の一部を表示できます。



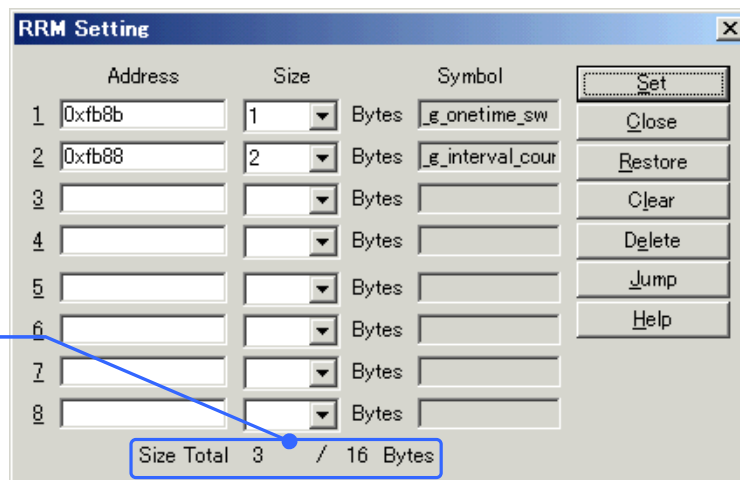
表示させたい範囲を選択、右クリックでメニュー表示させます。  
「h. プログラム実行中のグローバル変数の表示方法」を参考にRRM設定して下さい。

## 💡ワンポイント

### 実行中の表示制限について

実行中に表示するグローバル変数、メモリには表示制限があります。全ての表示を合わせて16バイト以内しか表示できません。表示する際には必要なものだけを設定するようにして下さい。

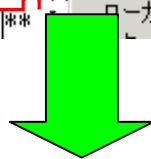
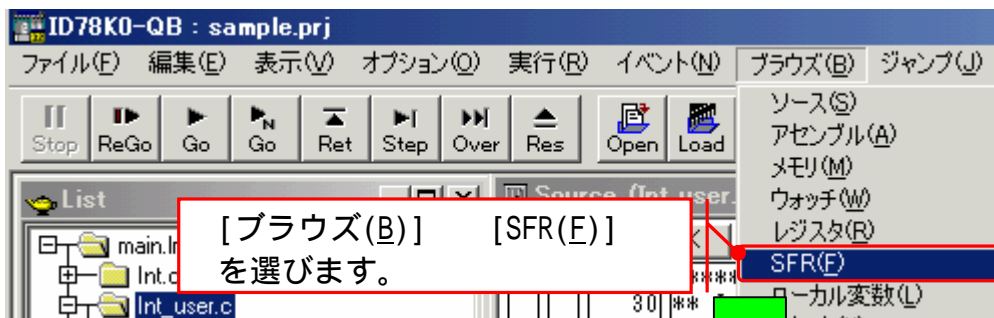
この設定では制限の16バイトの内、3バイトを使用している事を表しています。



# SFR表示

## 1. SFR表示方法。

メニューの[ブラウズ(B)] [SFR(E)] を選ぶと、[SFR]ウィンドウが表示されます。



Name	Attribute	Value
E0	R/W 1,8	FF00
P1	R/W 1,8	FF01
P2	R/W 1,8	FF02
P3	R/W 1,8	FF03
P4	R/W 1,8	FF04
P5	R/W 1,8	FF05
P6	R/W 1,8	FF06
P7	R/W 1,8	FF07
ADCR	R 16	FF08
ADCRH	R 8	FF09
RXB6	R* 8	FF0A
TXB6	R/W 8	FF0B
P12	R/W 1,8	FF0C
P13	R/W 1,8	FF0D
P14	R/W 1,8	FF0E
SIO10	R* 8	FF0F
TM00	R 16	FF10
CR000	R/W 16	FF12
CR010	R/W 16	FF14
TM50	R 8	FF16
CR50	R/W 1,8	FF17
CMP00	R/W 8	FF18
CMP10	R/W 8	FF19
CMP01	R/W 8	FF1A
CMP11	R/W 8	FF1B
TM51	R 8	FF1E

## 💡ワンポイント

### SFRについて

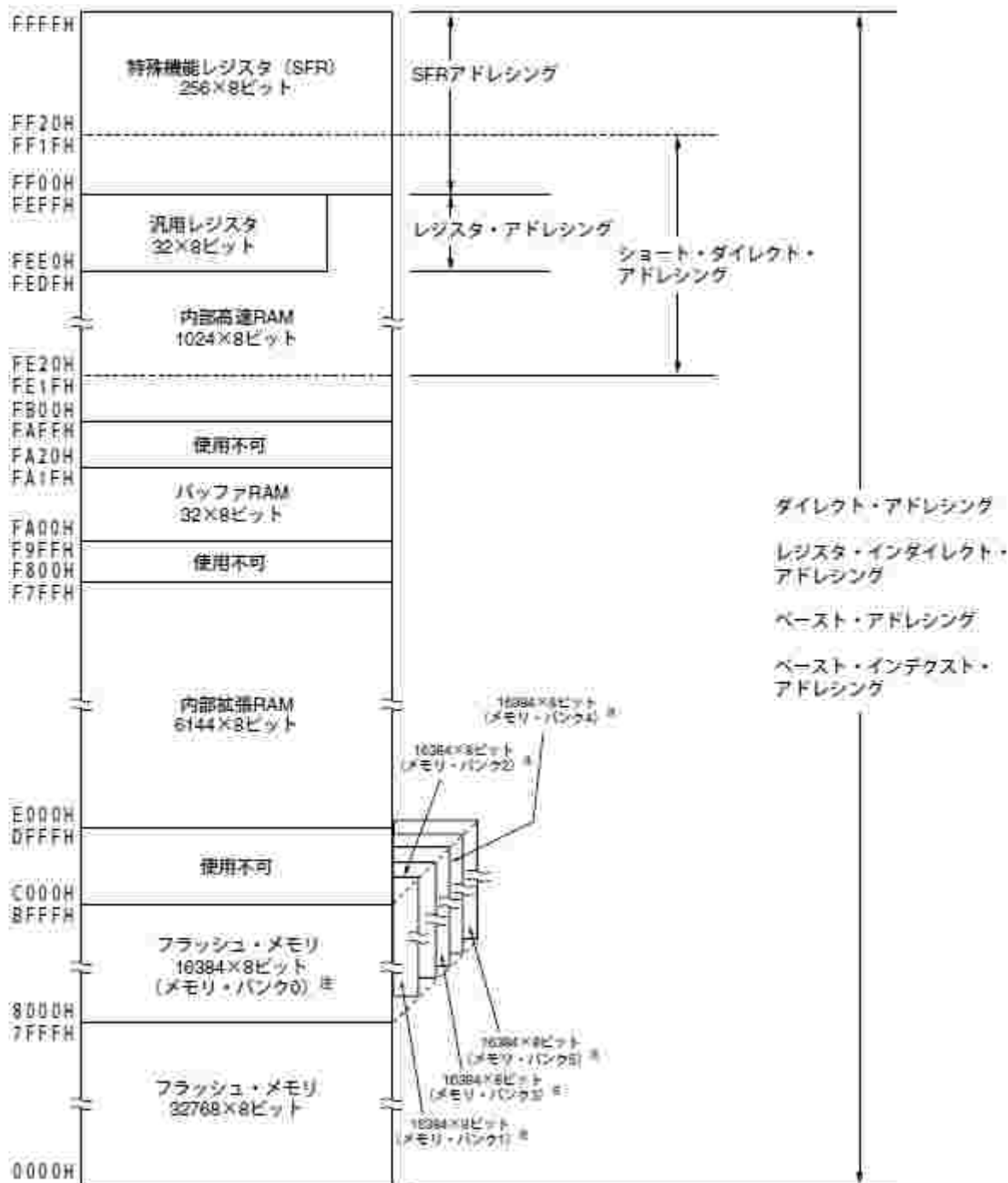
特殊機能レジスタの意味です。レジスタ類はメモリ空間に割り当てられています。78K0/Kx2のマイコンは、FF00h ~ FFFFhの256バイトが特殊機能レジスタ(SFR)と呼ばれます。



# メモリマップ



m. QB-78K0KF2-TBに搭載されているマイコン(μPD78F0547D)のメモリマップです。



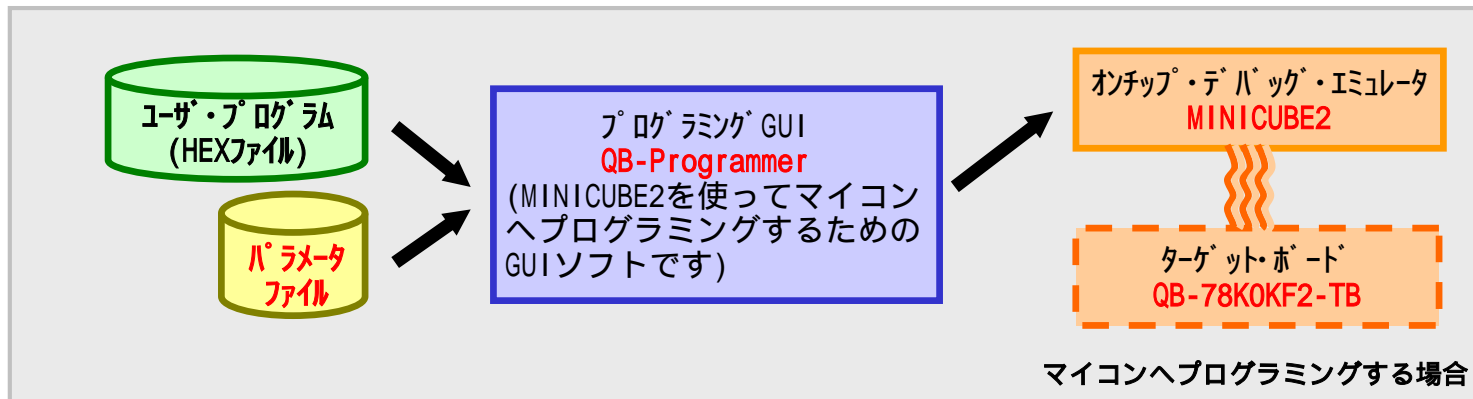
注 メモリ・バンク選択レジスタ (BANK) で設定していないメモリ・バンクに分岐およびアドレッシングする場合は、BANKでメモリ・バンクの設定を切り替えてください。

「デバッグしたい」の章は以上です。ここに記したデバッグ手法は基本的に過ぎません。プログラムが複雑になるほど多くの問題が発生します。多くの例では、スタックの問題が挙げられます。マイコンはRAMの容量が限られています。配列などを多く使うと知らない内にプログラムが使うスタック領域が足りなくなる問題が発生します。プログラムを作る際はマイコンのメモリマップも意識する必要があります。次章より「マイコンへ書き込み」を説明します。マイコンへプログラミング(書き込み)する場合は、次章を読んで下さい。

# プログラミングについて

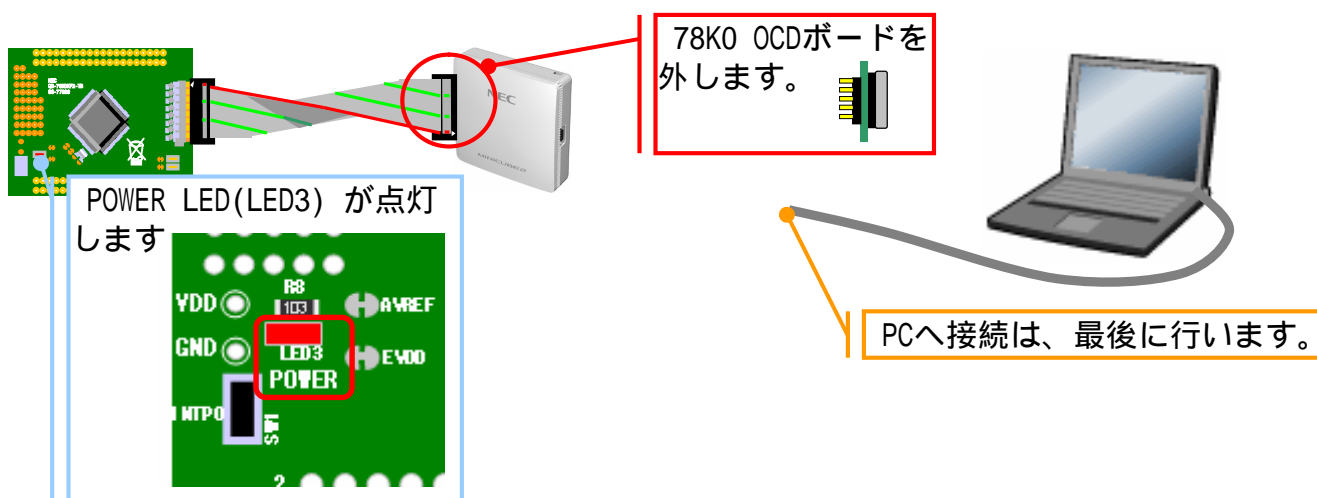
[マイコンへ書き込み]の章では、作成したプログラムをマイコンへプログラミング(書き込み)します。一度プログラミングすれば電源を供給するだけでプログラムが実行されます。学習時間は15分です。

## プログラミング方法



## MINICUBE2との接続

78K0 OCDボードを取り外して、MINICUBE2と接続します。



## 💡 ワンポイント

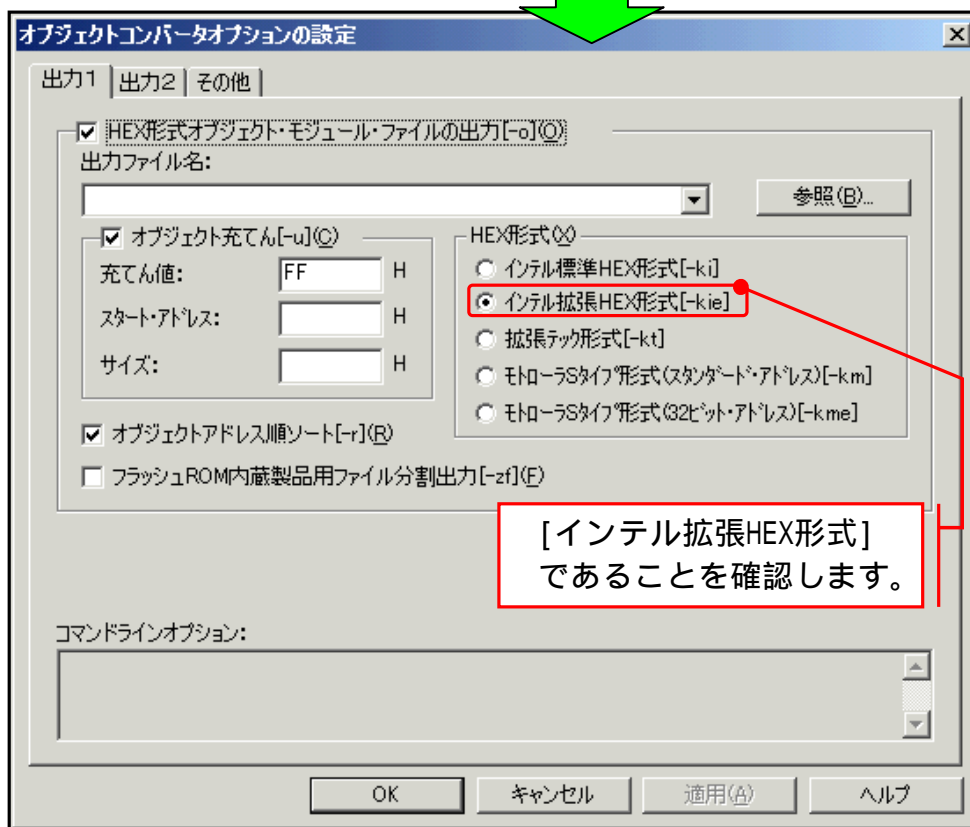
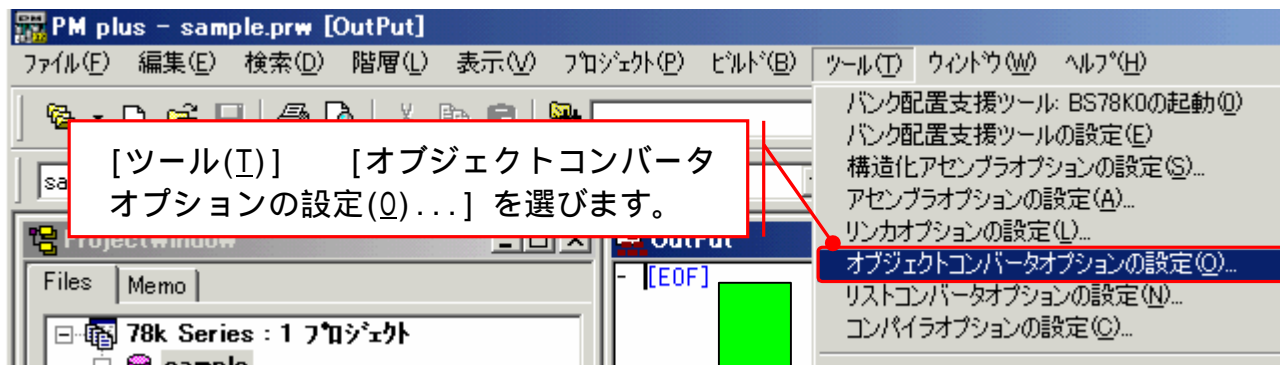
### 電源選択SWについて

ターゲット用電源を切り替えます。切り替えは5V供給、3V供給、ターゲット電源使用の3種類です。78K0/KF2は20MHzの動作電圧が4.0~5.5Vなので5V供給で動作します。小さい回路ならMINICUBE2からの電源供給で大丈夫ですが、100mAを超えるような(モータを使うなど)回路は必ずターゲット・システム側で電源を用意して下さい。

# HEXファイル作成

## a. HEXファイル作成する設定を確認します。

PM+のメニューより[ツール(T)] [オブジェクトコンバータオプションの設定(O)...] を選んで下さい。[オブジェクトコンバータオプションの設定]ウィンドウが開きます。



## b. ビルドを行い、HEXファイルを作成します。

PM+のメニューより[ビルド(B)] [ビルド(B)...] を選んでビルドを行って下さい。詳細は[動かしてみよう]の章[プログラムのビルド]を参考にして下さい。

## 💡 ワンポイント

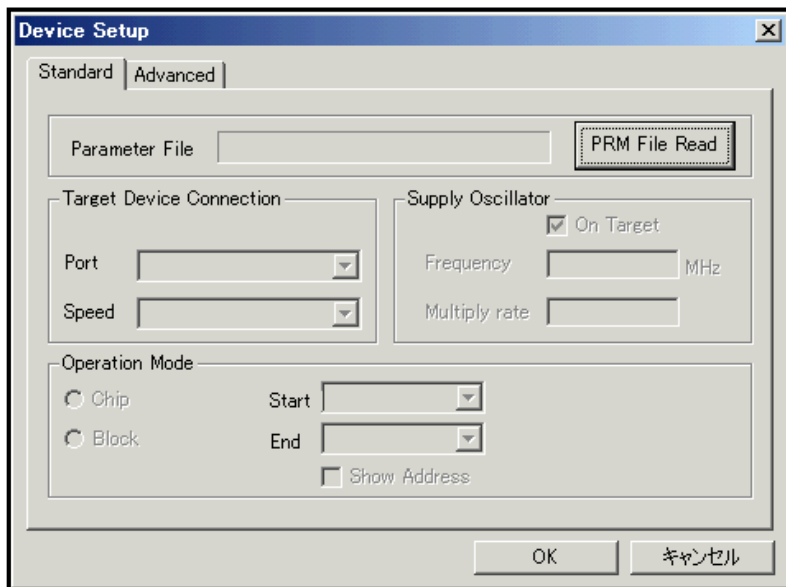
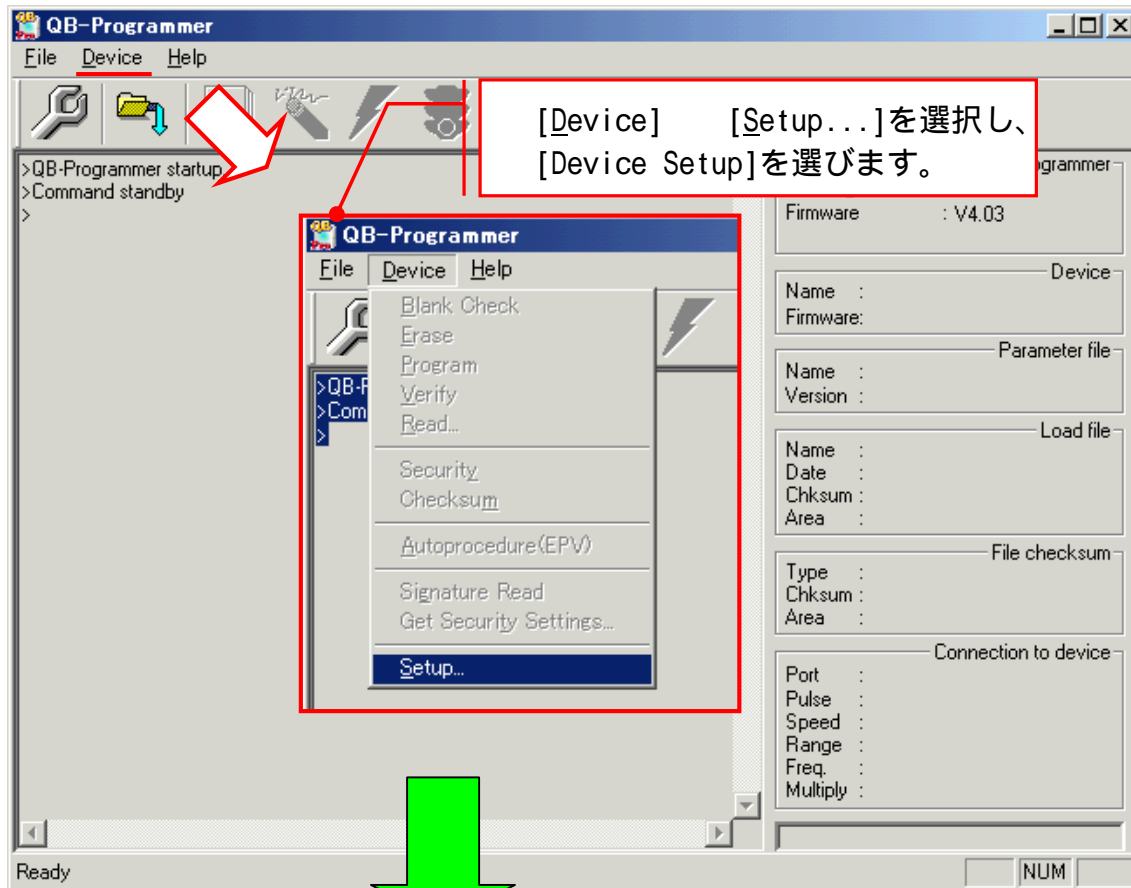
### HEX形式について

「インテル拡張HEX形式」ではアドレス0xFFFFFまでをサポートしています。「インテル標準HEX形式」では64KByteのアドレス0xFFFFまでサポートしているので、64KBのアドレス空間を持つ78K0マイコンでは充分ですが、ここでは「インテル拡張HEX形式」を使います。MINICUBE2以外でプログラミングする場合には「モトローラSタイプ形式」を使う場合があります。

# QBP(QB-Programmer)の起動

c. マイコンへプログラミングするソフトウェア(QBP)を起動します。

[スタート] [プログラム(P)] [NEC Electronics Tools] [QBP] [Vx.xx]  
 [QBP Vx.xx QB-Programmer]でQBPを起動します。起動後に[Device] [Setup...]を選択し、  
 [Device Setup]ウインドウを開きます。



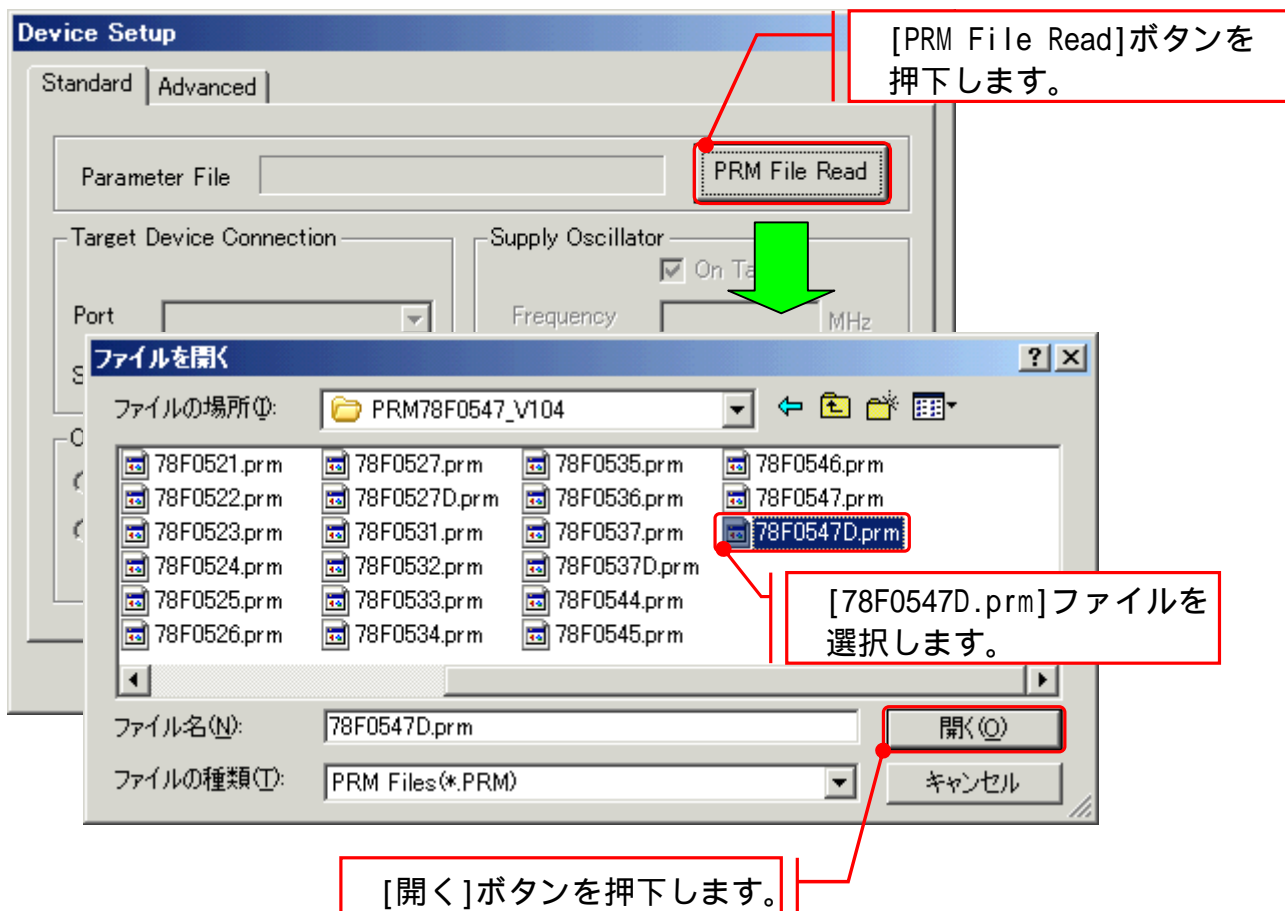
**ワンポイント**

QBP起動時について  
 QBPを起動すると、MINICUBE2の中央LEDが緑に変わります。これはプログラムモードに入った事を示します。

# パラメータファイルの読み込み

## d. パラメータファイルを読み込みます。

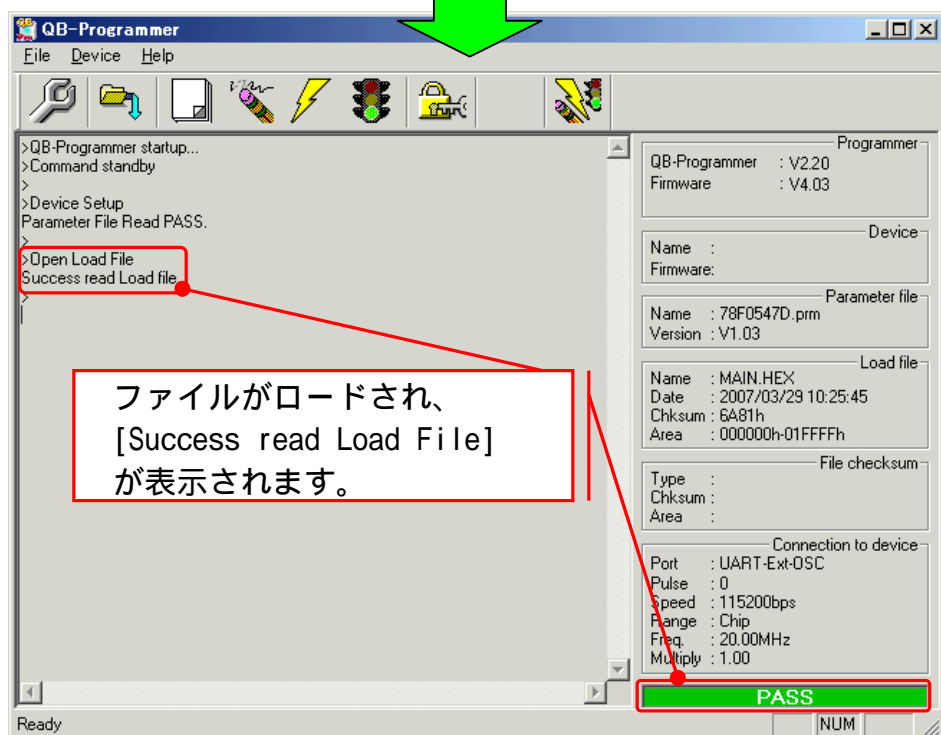
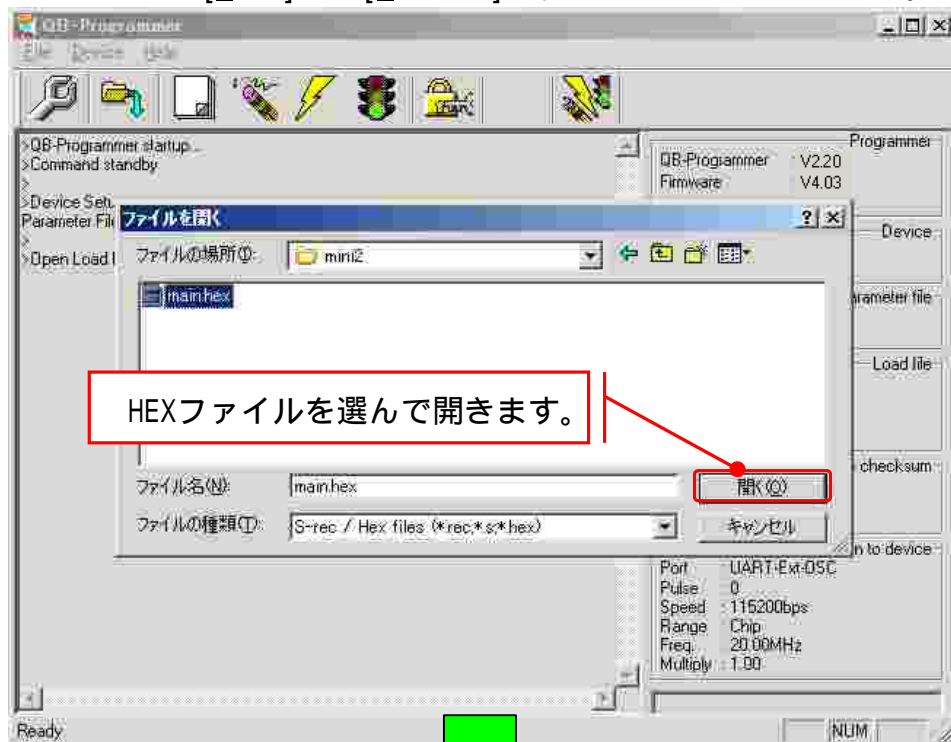
[Device Setup]ウインドウより[PRM File Read]ボタンを押下してパラメータファイルを読んで下さい。



# HEXファイルの読み込み

e. HEXファイルを読み込みます。

メニューより[File] [Load...]で、HEXファイルを選びます。



## 💡ワンポイント

QBP(QB-Programmer)の起動について

2回目以降にQBPを起動した場合は、以前の設定が反映されます。そのためパラメータファイルの設定、プログラムのダウンロードなど再設定する必要がありません。

# マイコンへプログラミング

## e. マイコンへプログラミングします。

メニューより [File] [Autoprocedure(EPV)] を選ぶとプログラミングされます。

The screenshot shows the QB-Programmer interface. The 'File' menu is open, and 'Autoprocedure(EPV)' is highlighted. A red box around the menu item is annotated with the text: "[Autoprocedure(EPV)] を選ぶとプログラミングが開始されます。" (Selecting [Autoprocedure(EPV)] starts programming.)

The main window shows the progress of the programming. The status bar at the bottom left displays 'Ready'. The main text area shows:
 

```
>AutoProcedure(Epv)
Blank check Chip: Not blank, Erase need.
Erasing...
Erase Chip : PASS
Program Chip.
10%
20%
```

 A blue box highlights the progress bar and the percentage values (10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, 100%, PASS). A red box points to this area with the text: "書き込みの進行に応じて、数値とプログレスバーが変化します。" (As the progress of writing changes, the numerical value and progress bar change.)

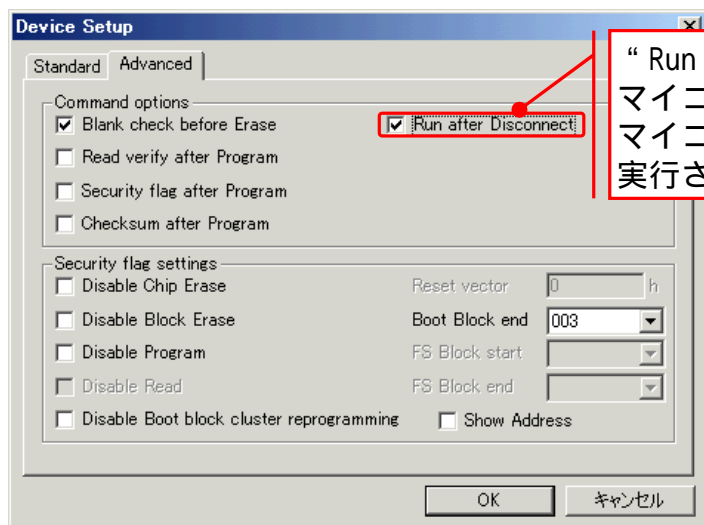
At the bottom right, a green box highlights the 'PASS' status, with a red box pointing to it from the text: "書き込み終了のメッセージが表示されます。" (The message indicating the end of writing is displayed.)

# 動作確認



## f. マイコンの動作確認します。

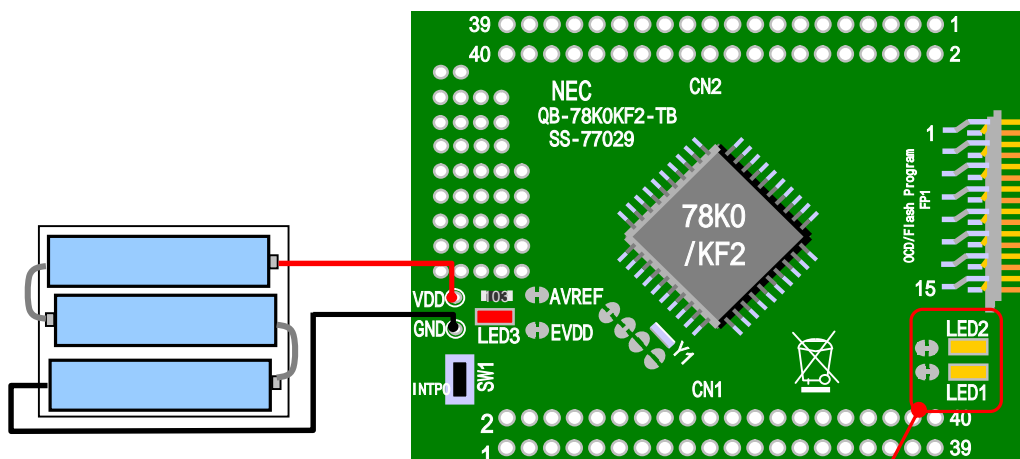
[Device] [Setup...]を選択し、[Device Setup]ウインドウを開きます。[Device Setup]ウインドウより[Advanced]タブを選択し、「Run after Disconnect」へチェックして下さい。



“Run after Disconnect”へチェックすると、「e. マイコンへプログラミングします。」の処理を行い、マイコンの書き込みが終了した後に、プログラムが実行されます。

また、下記の方法でも動作確認できます。

MINICUBE2とUSBケーブルを外し、16pinケーブルも外してQB-78K0KF2-TBだけにします。VDD, GNDに乾電池3本を接続し、動作することを確認します。



電池を接続するとLEDが点滅開始します。SW1の押下でLEDの点滅タイミングも変化します。

「マイコンへ書き込み」の章は以上です。現在までの章「動かしてみよう」、「デバッグしたい」とあわせて、マイコンの開発一通りを学びました。次章ではプログラムの実行、デバッグをソフトウェアのみで実現する「システム・シミュレータ」について説明します。



# システム・シミュレータ (SM+) とは

[ソフトで開発]の章では、作成したプログラムの実行、デバッグをソフトウェアで実現する「システム・シミュレータ(以下SM+)」について説明します。学習時間は30分です。

デバッグ可能なハードウェア環境があるのに、なぜSM+を使う必要があるのか？と思われるでしょう。SM+には、以下の特徴があります。

- ・ **ターゲットを必要としない評価の実現**

ハードウェアを必要としないので、実際のデバッグ環境を構築せずに先行してソフトウェアのみでデバッグが可能です。

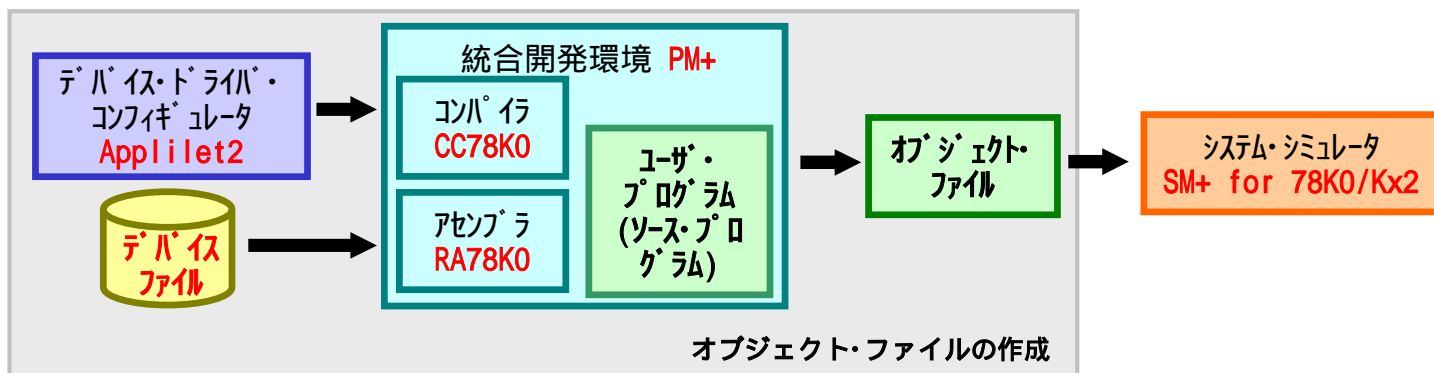
- ・ **さまざまなシミュレーション機能の実現**

スイッチ、LEDなどの入出力パネルをソフトウェアで構築可能です。また、マイコンの入出力波形をタイミング・チャート・ウインドウ(ロジック・アナライザのイメージ)で計測可能です。

- ・ **制限のないデバッグ**

ブレークポイントの数やカバレッジ機能などオンチップ・デバッグでは実現できない豊富なデバッグ手法が使えます。

## プログラム開発環境



# 環境の確認

## a. PM+で使用するデバッガを設定します。

メニューの[ツール(T)] [デバッガの設定(D)...] で「デバッガの設定」ダイアログを開き「SM+ for 78K0/Kx2 システム・シミュレータ」を選択して下さい。

デバッガの設定

デバッガの選択

選択デバッガ(D): SM+ for 78K0/Kx2 システム・シミュレータ

78K0 総合デバッガ (ID78K0-QB)

ファイル名(N): SM+ for 78K0/Kx2 システム・シミュレータ

オプション(O):

デバッガ・ターゲット

デバッガ・ターゲット・ファイル(F): C:\QB-78K0KF2-TB\mini2\main.lmf

デバッガにダウンロード時、シンボル・リセットを実行する(S)

デバッガにダウンロード時、CPUリセットを実行する(C)

デバッガ・オプション

同じグループ内の複数のデバッガ・ターゲット・ファイルをダウンロードする(W)

ダウンロード・ファイル一覧(L):

OK キャンセル ヘルプ(H)

システム・シミュレータを選択します。

## 💡ワンポイント

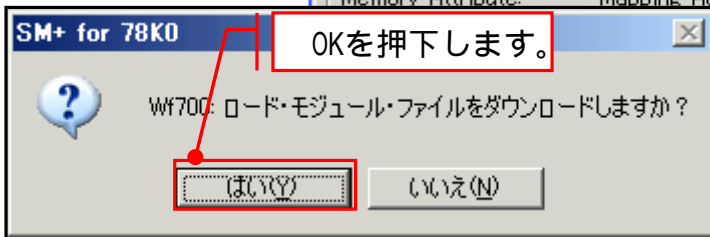
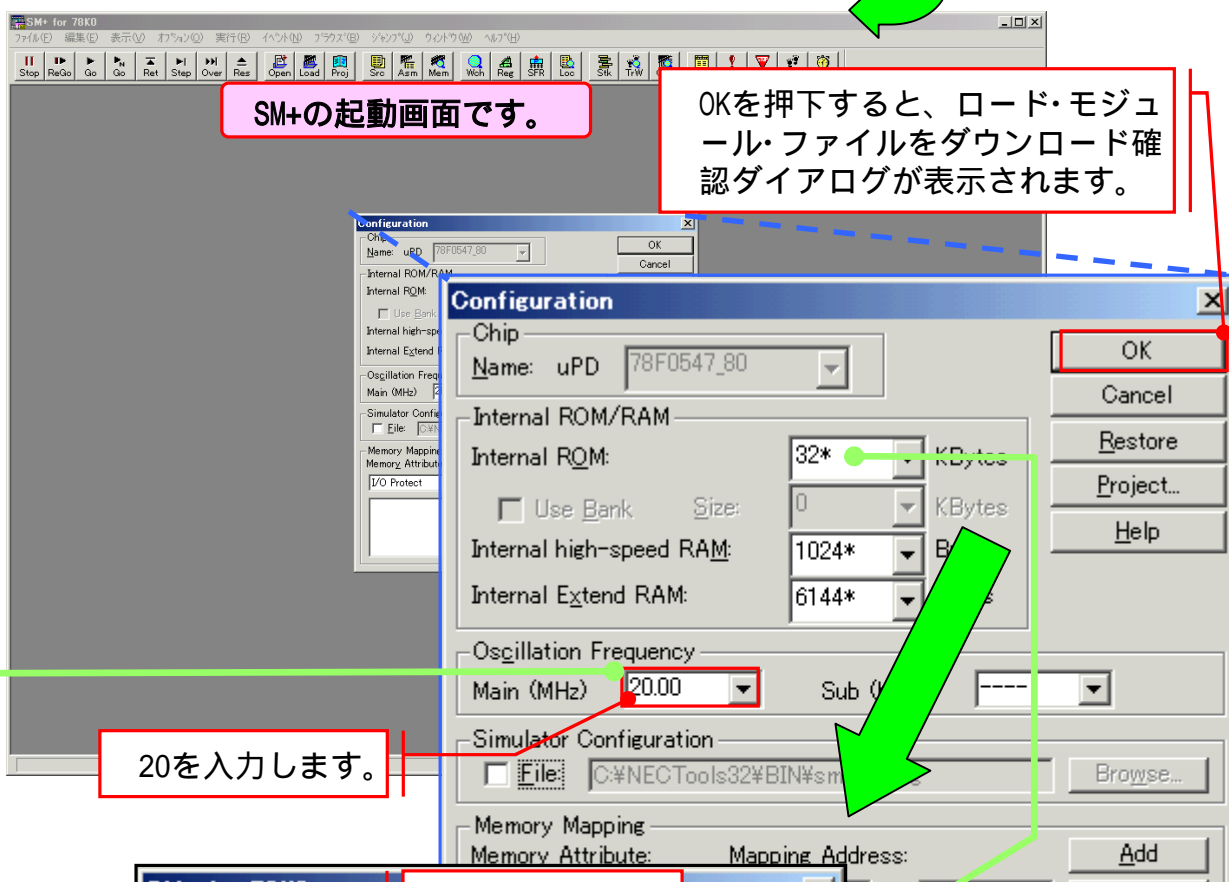
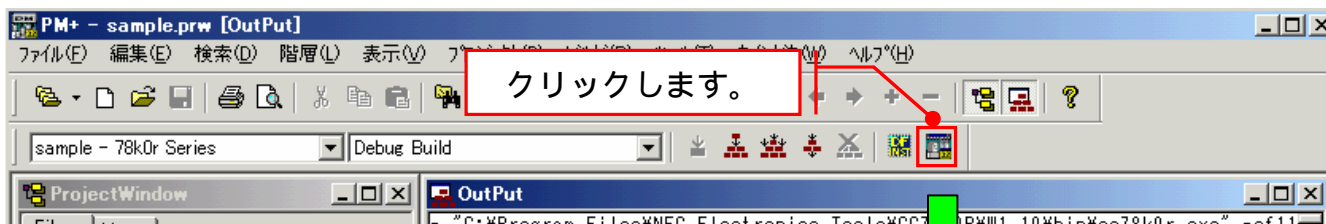
### 実行速度について

システム・シミュレータの実行速度は、PCの環境に依存します。実デバイスと異なり総じて実行時間は遅くなります。なるべく早いPC環境での動作をお勧めします。なお、内部的に発生するタイミングについては、正しく計算されておりPC上での動作時間には影響されません。

# SM+の起動

## b. SM+を起動します。

PM+のデバッグ・アイコンを押下して[SM+]を起動します。プログラムのダウンロードまで行って下さい。



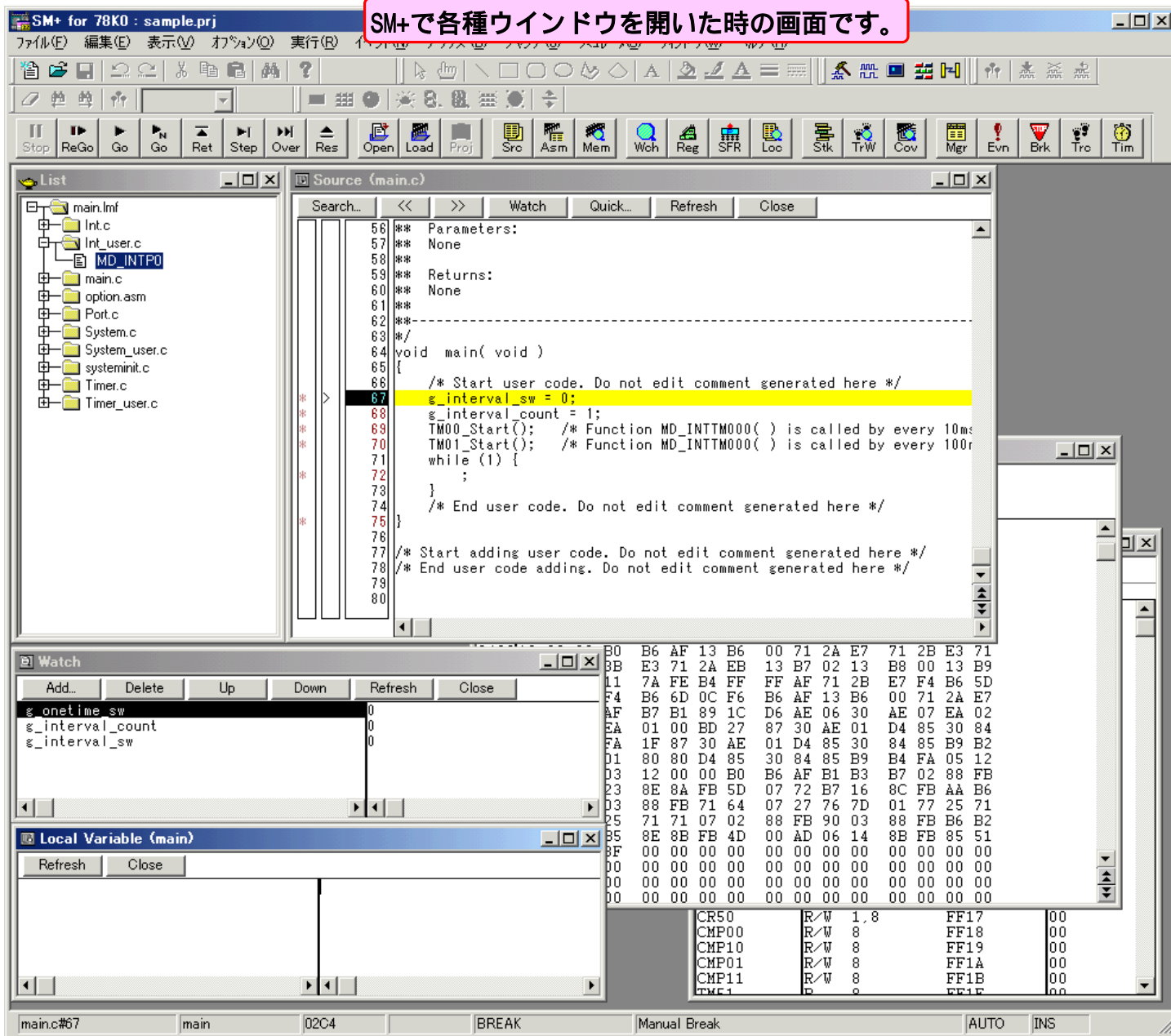
**💡ワンポイント**  
**Internal ROMについて**  
 フリー・ツール版のSM+ for 78K0/Kx2では、プログラムサイズ32KBまでの制限があります。その他のデバッグ機能については制限ありません。

**💡ワンポイント**  
**Oscillation Frequencyについて**  
 Oscillation Frequencyの値はApplilet2の高速システム・クロック設定の周波数(MHz)と同じ値を設定して下さい。この値はシリアル・ボーレート、タイマの動作などに影響しますので正しい値を設定して下さい。

# SM+の基本画面

## c. SM+の基本画面を設定します。

[デバッグしたい] の章 [デバッガの基本画面] を参考に[List], [Watch], [Local Variable]のウィンドウを開いて下さい。ただし、[デバッグしたい] の章[変数表示]で行っている [オプション(O)] [拡張オプション(X)...]の[Use MINICUBE Extended Function]にチェックする必要はありません。



## 💡ワンポイント

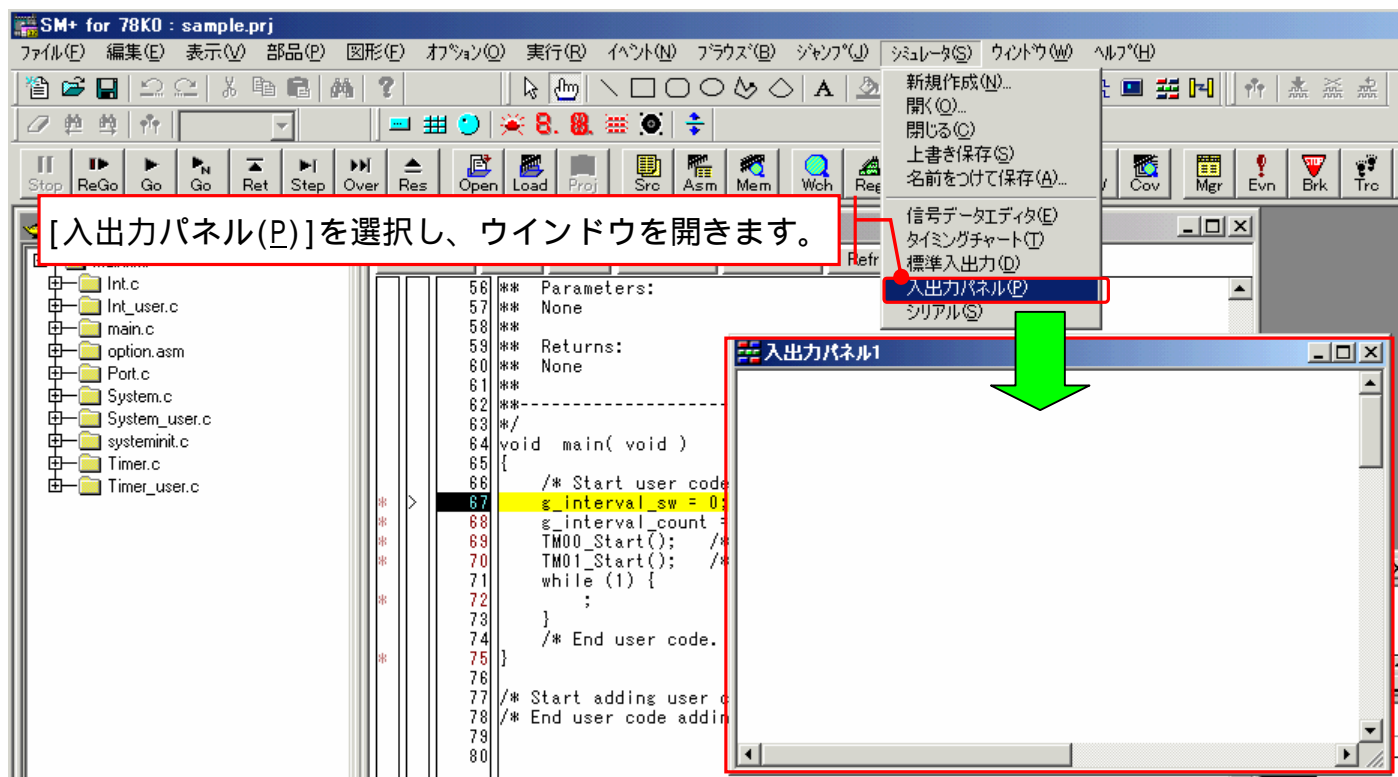
### 基本的な使い方について

システム・シミュレータ(SM+)も統合デバッガ(ID78K0-QB)も使い方としては同じです。ブレークポイントの設定、Watchウィンドウの設定方法も同じです。しかし、SM+にはさまざまなシミュレーション機能があります。次ページよりシミュレーション機能を説明します。


# 入出力パネルの設定

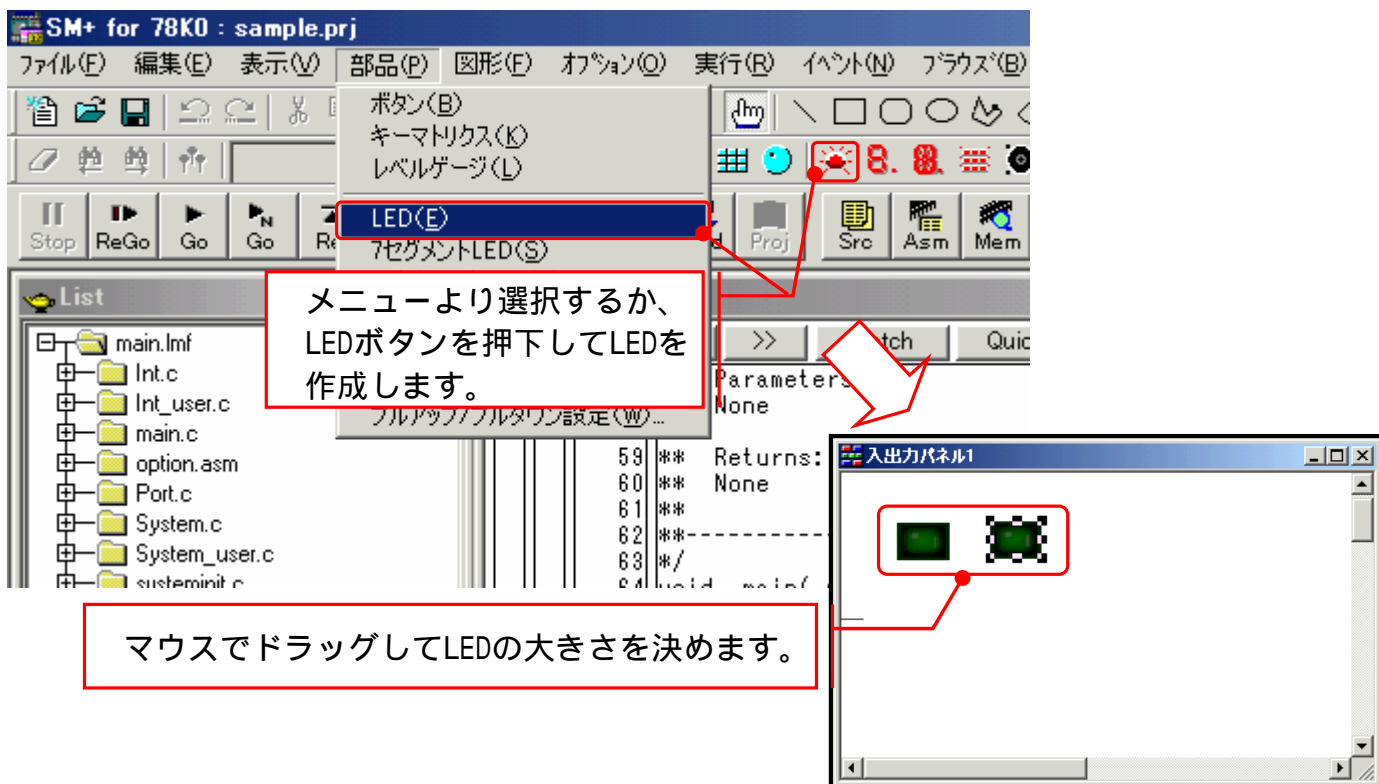
## d. SWやLEDを作る為のウィンドウを開きます。

SM+のメニュー [シミュレータ(S)] [入出力パネル(P)]を選び[入出力パネル]ウィンドウを開きます。




## e. LEDを作成する。

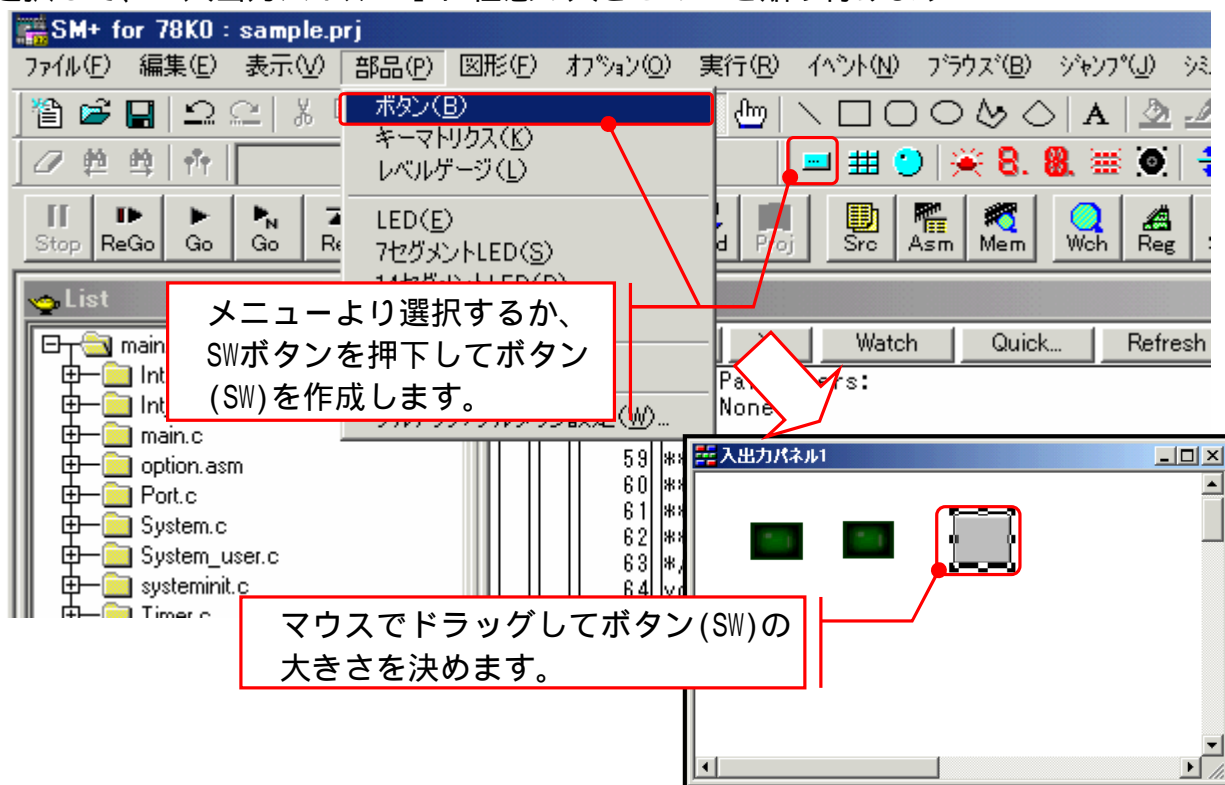
「入出力パネル1」をアクティブにした状態でLED作成アイコン  を押すか[部品(P)] [LED(E)]を選択して、「入出力パネル1」に任意の大きさのLEDを2個、貼り付けます



# 入出力パネルの設定

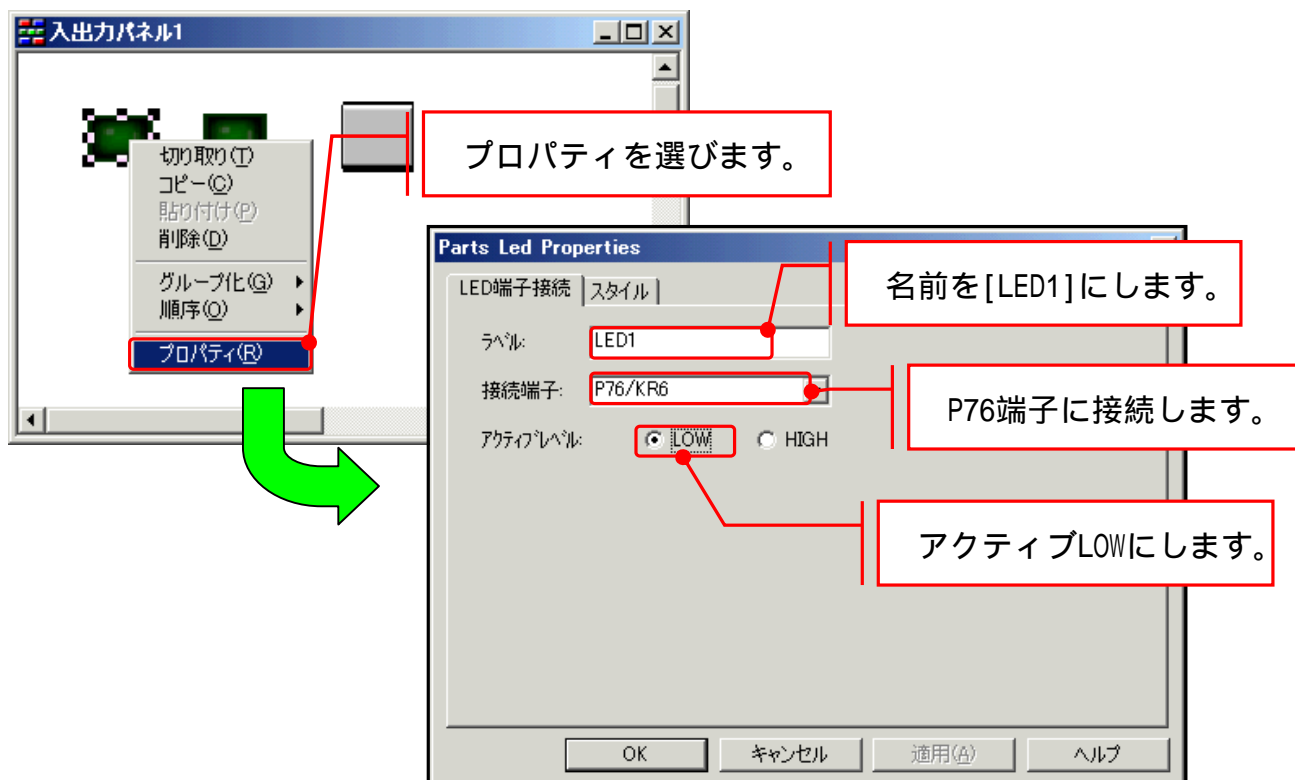
## f. SW(ボタン)を作成します。

LEDを作成するのと同様にボタン作成アイコンを押下するか[部品(P)] [ボタン(B)]を選択して、「入出力パネル1」に任意の大きさのSWを貼り付けます



## g. 作成したLEDに端子を割り当てます。

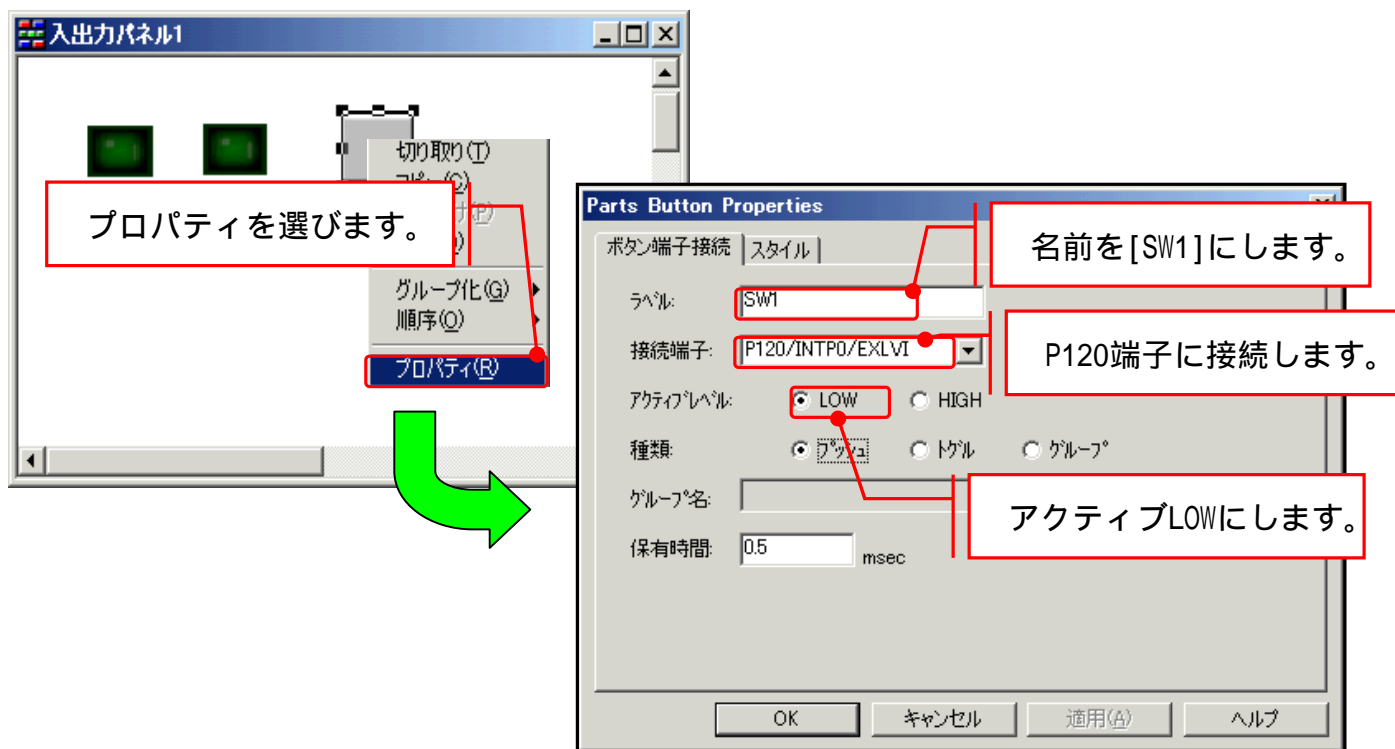
LED部品の上で右クリック [プロパティ(R)]を選択して表示される[Parts Led Properties]ダイアログで、接続する端子を設定します。同様に2つ目のLEDはラベル[LED2]、接続端子[P77]、アクティブレベル[LOW]に設定します



# 入出力パネルの設定

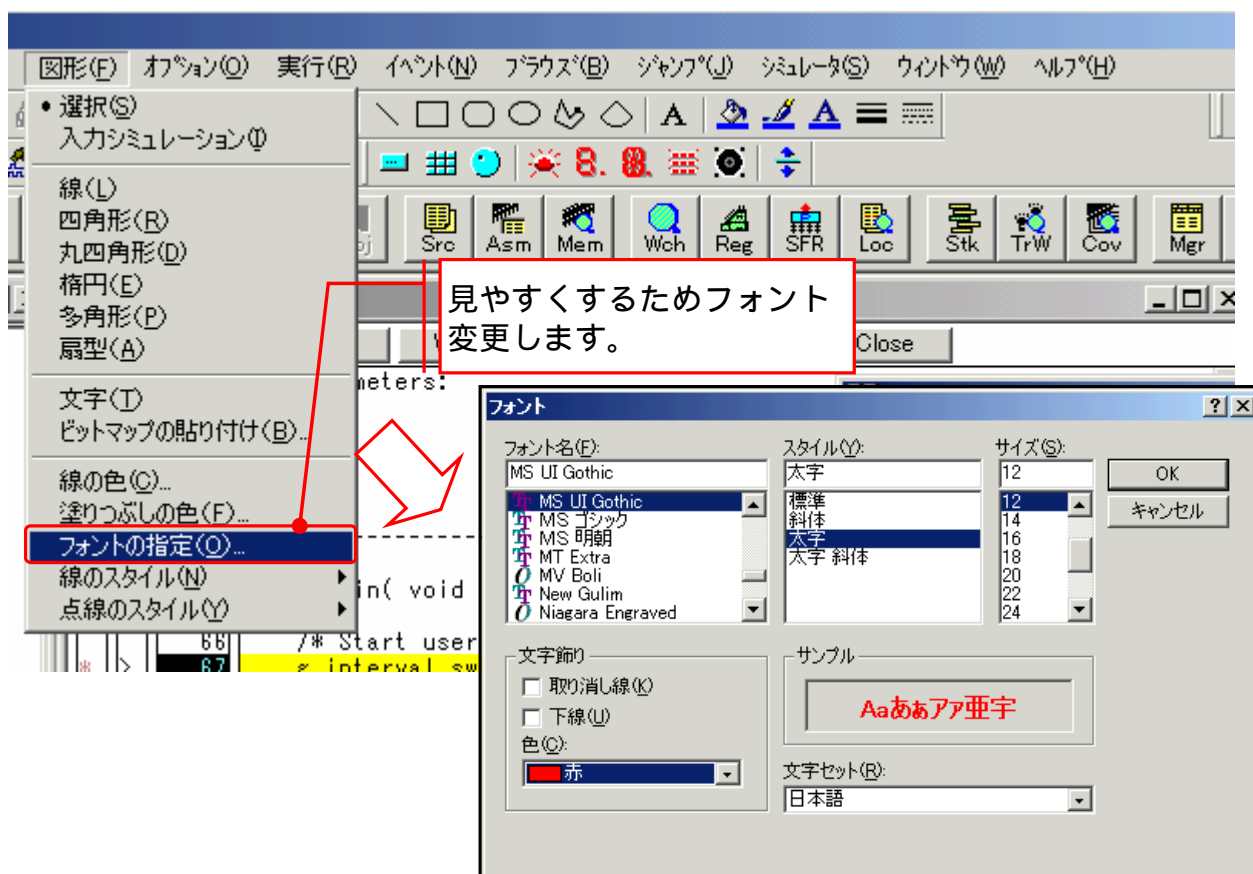
## h. 作成したSWに端子を割り当てます。

LED部品の上で右クリック [プロパティ(R)] を選択して表示される [Parts Button Properties] ダイアログで、接続する端子を設定します。



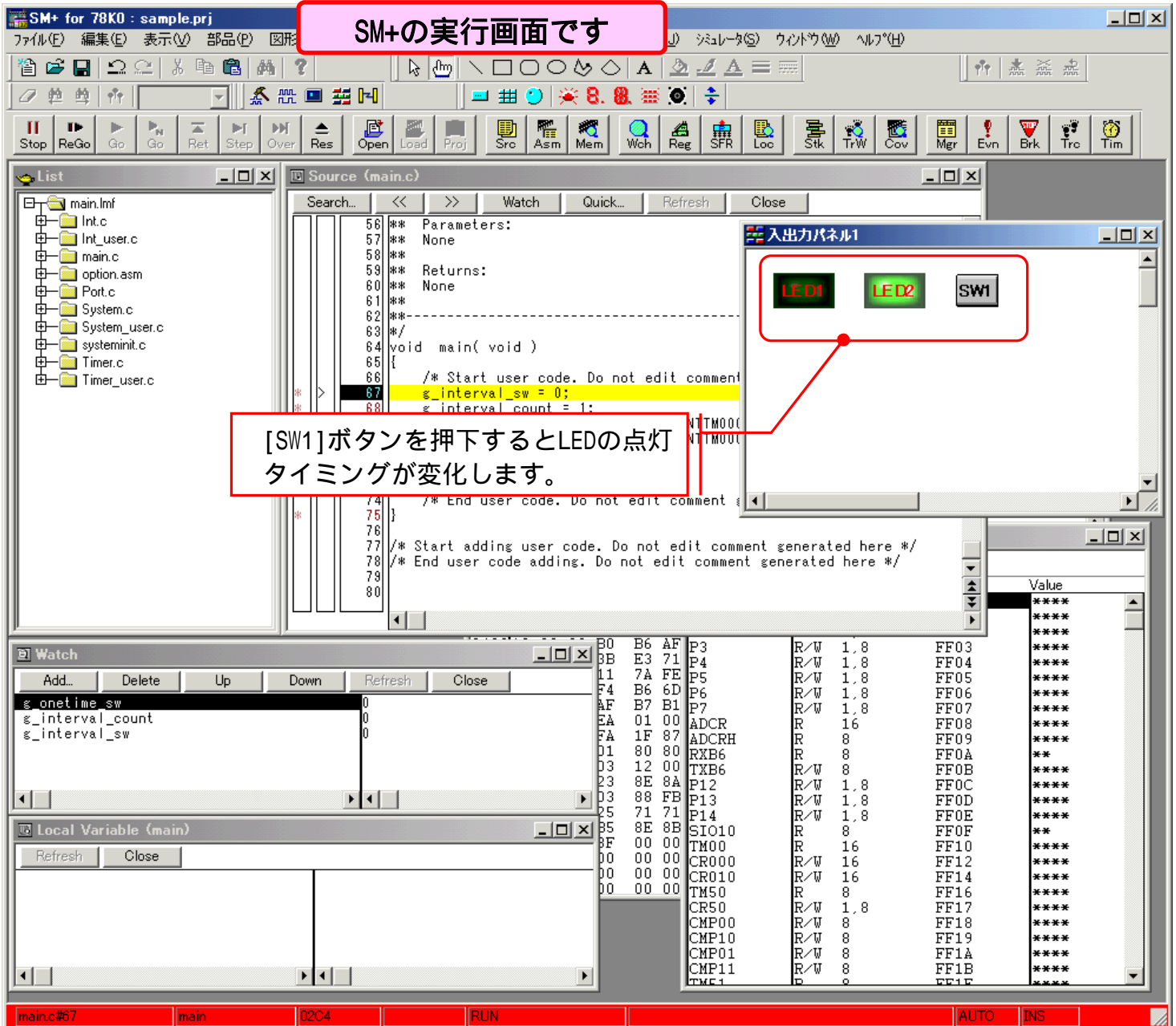
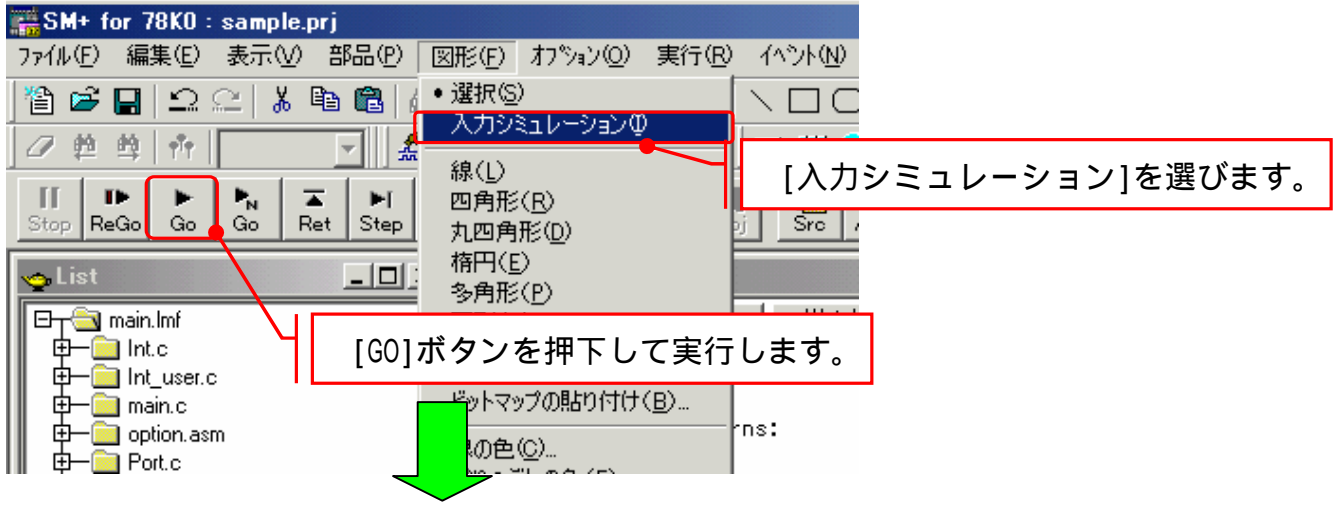
## i. フォントを変更します。

SM+のメニュー [図形(E)] [フォントの指定(O)...] でフォント変更が可能です。



# 実行画面

- j. 入出力パネルを[入力シミュレーション]へ設定して実行します。  
 入出力パネル編集時は[選択]、プログラムを実行するときは[入力シミュレーション]にして下さい。

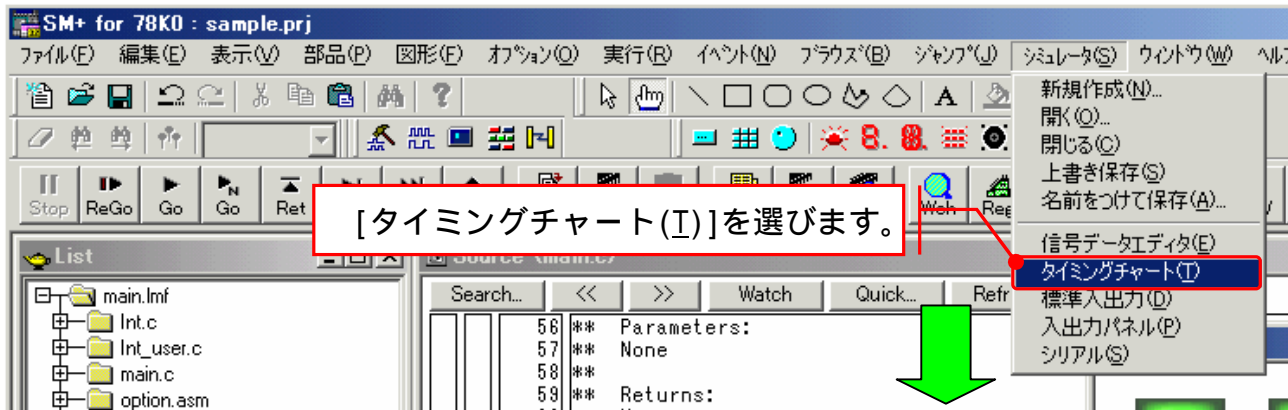




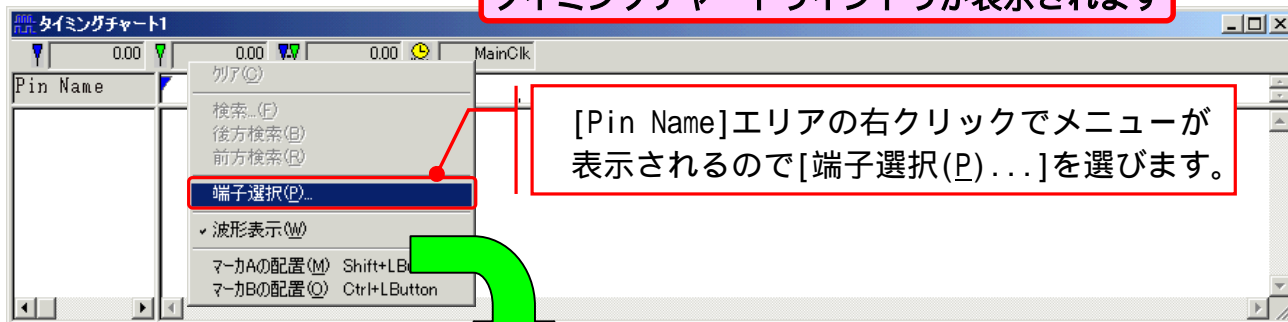
# SM+でのデバッグ

## k. タイミングチャートを設定します。

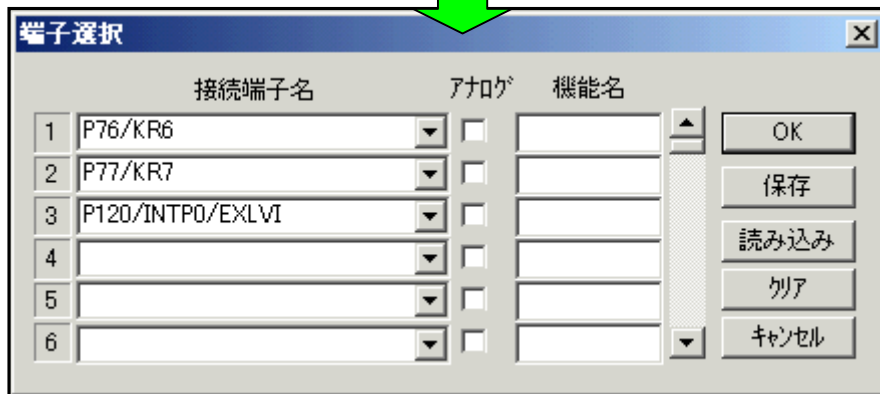
SM+では信号のタイミングチャートを表示する機能を持っています。



タイミングチャートウィンドウが表示されます

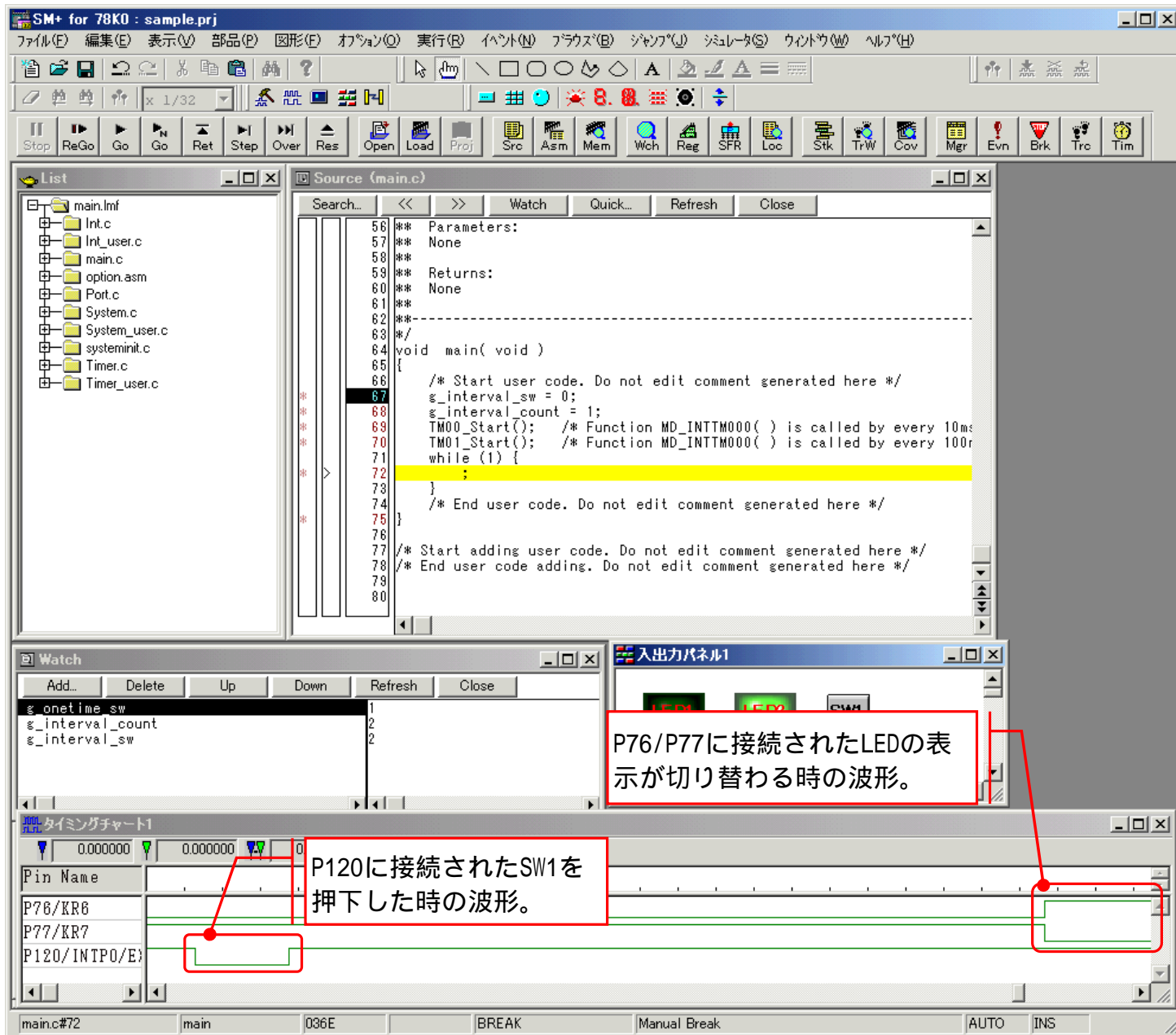


[端子選択]ウィンドウが表示されたら、接続端子を入力します。[1]へP76、[2]へP77、[3]へINTP0を割り当てます。



# SM+でのデバッグ

## 1. タイミングチャートを表示した実行画面。



システム・シミュレータ (SM+) の基本的な使い方については、以上です。次章では「システム・シミュレータ」のもう少し高度なデバッグ方法について説明します。



# 更に高度なデバッグ機能

SM+では更に高度なデバッグ機能が色々あり、全てを伝えるのは難しいのですが、是非知って欲しい機能があります。それは「イベント・リンク」です。

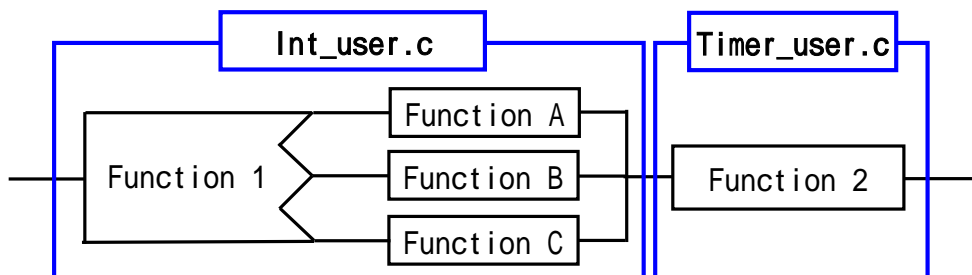
「イベント」とは、「アドレス0x1000番地をフェッチした」、「アドレス0x2000番地にデータを書き込んだ」などのデバッグにおけるターゲット・システムの特定の状態を指しています。デバッガ(SM+, ID78K0-QB)では、このようなイベントをブレーク、トレース等の各デバッグ機能のアクション・トリガとして利用しています。ブレークポイントで設定した「B」はブレーク・イベントなのです。イベントの種類は他に「トレース・イベント」、「タイマ・イベント」、「スタブ・イベント」、「スナップショット・イベント」があります。

ここで説明するのは、デバッグ用にいくつかのイベントを定義し、ある条件下によってブレークする場合を説明します。

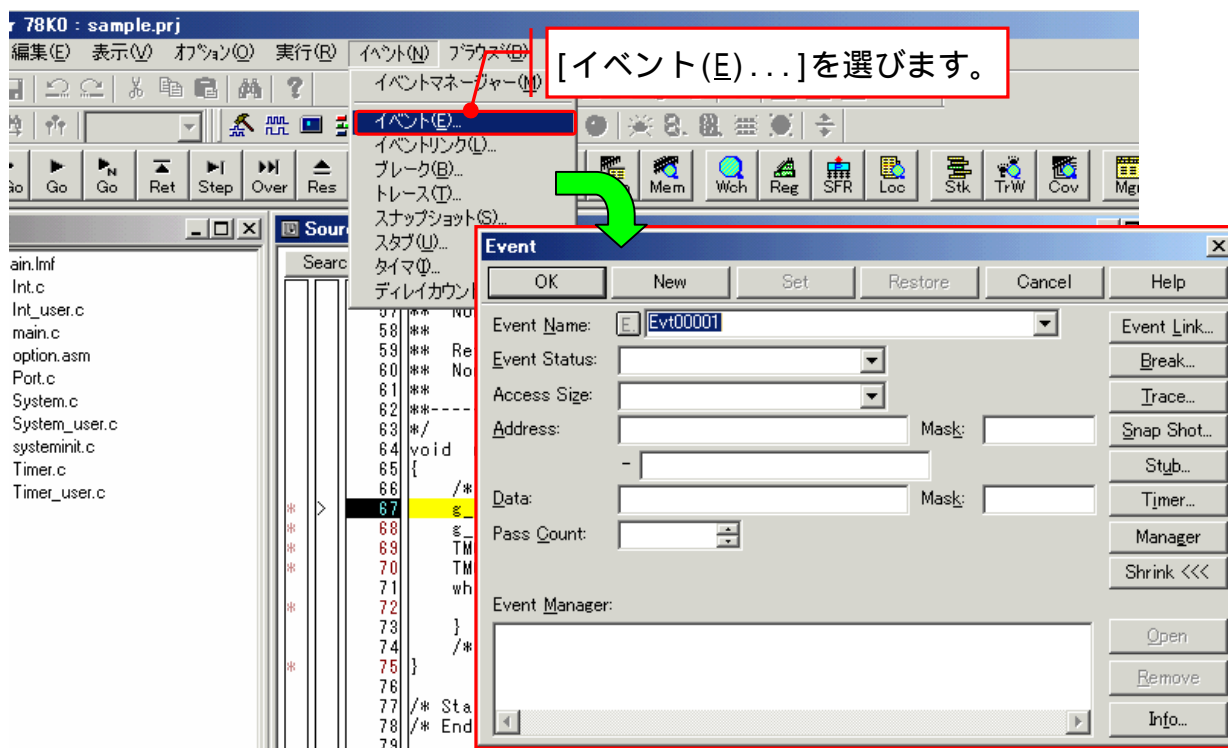
## m. [Event]ダイアログを開く。

「イベント・リンク」で任意のイベントを通過した時のみブレークすることができます。この機能はプログラム・シーケンスの確認にとっても有効です。

下記の例を説明します。プログラムは左から流れてきています。Function 1で、ある値の判断を行い処理を分岐してFunction A~Cを実行します。最後にFunction 2を行うフローです。



Function 2でプログラムを止める場合、Function A Function 2を通った場合だけプログラムを停止させることが可能です。実際にサンプル・プログラムで試してみます。まず、SM+上で[イベント]ダイアログを表示させます。

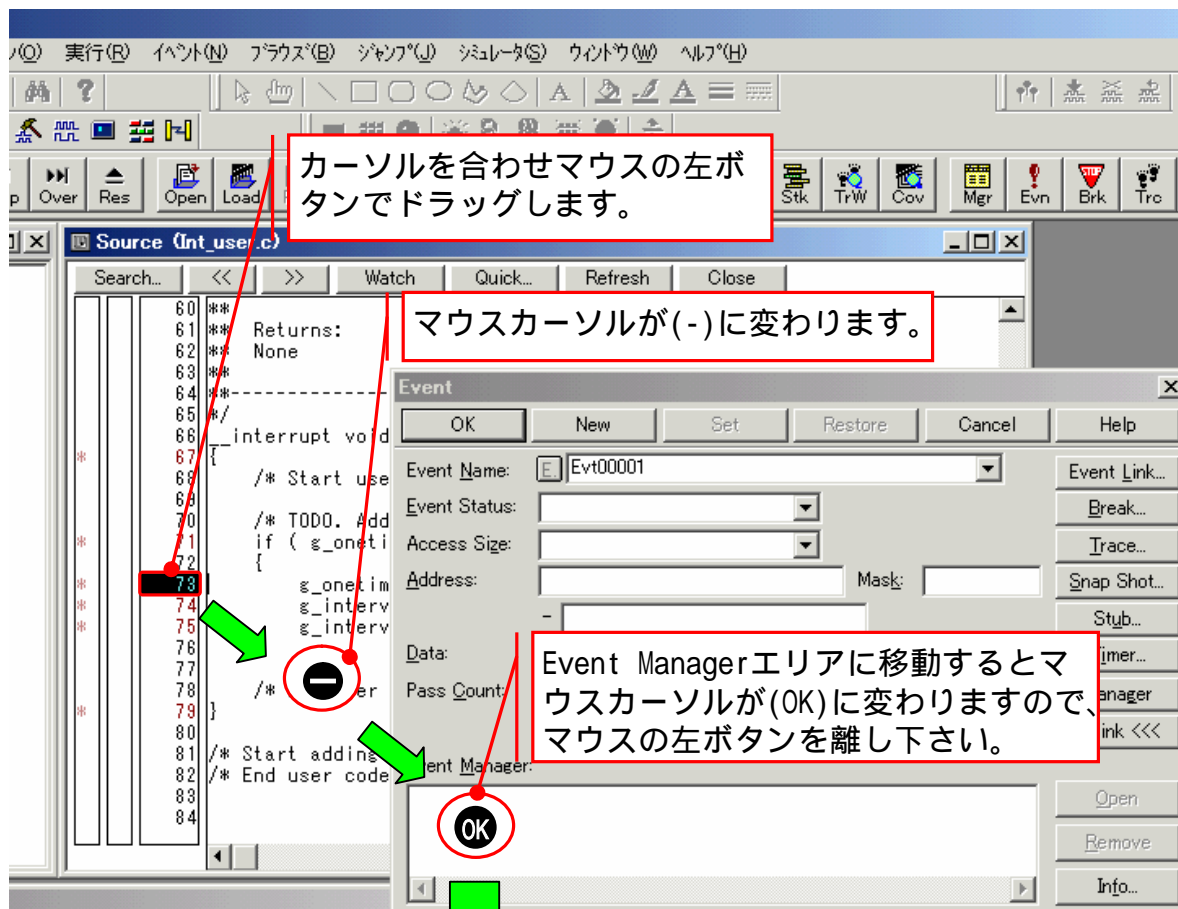




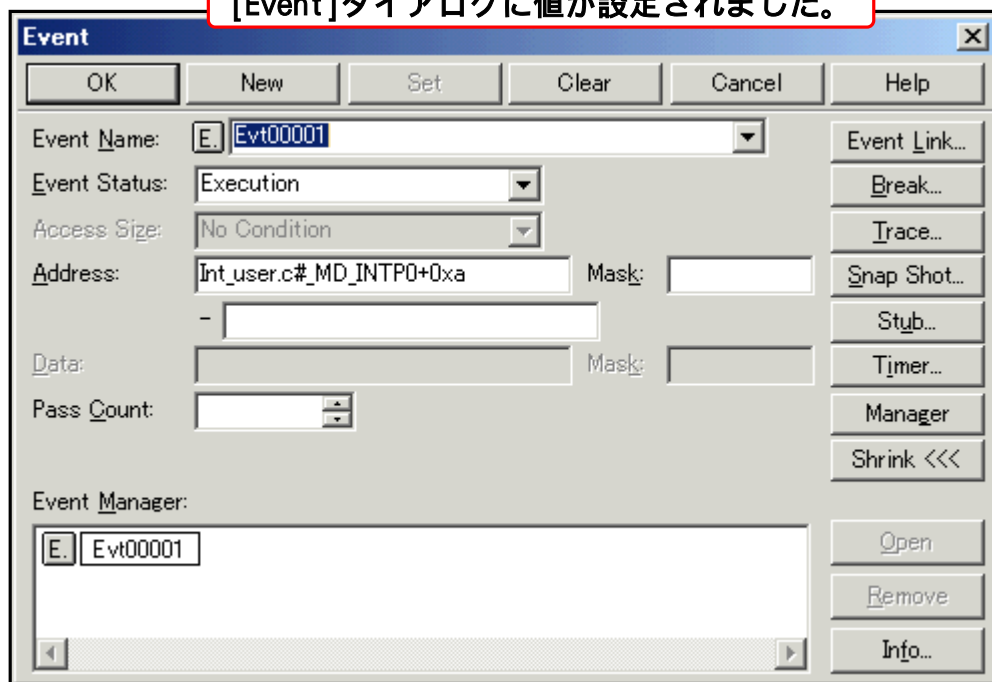
# 更に高度なデバッグ機能

## n. イベントを設定します。

Int\_user.cの73行目にカーソルを合わせ、左ボタンを押したままマウスを移動します。[Event]ダイアログのEvent Managerのエリアにカーソルを合わせて左ボタンを離して下さい。



[Event]ダイアログに値が設定されました。





# 更に高度なデバッグ機能

## o. 次のイベントを設定します。

同様にTimer\_user.cの77行目にカーソルを合わせて、イベントを作成します。

このスクリーンショットは、IDEのソースコード表示画面で、Timer\_user.cの77行目を指している様子を示しています。以下の注釈が追加されています：

- カーソルを合わせマウスの左ボタンでドラッグします。
- マウスカーソルが(-)に変わります。
- Event Managerエリアに移動するとマウスカーソルが(OK)に変わりますので、マウスの左ボタンを離して下さい。

このスクリーンショットは、「Event」ダイアログボックスの表示画面です。Event Managerリストに「E. Evt00002」が追加されていることが確認できます。

[Event]ダイアログにEvt00002が追加されました。



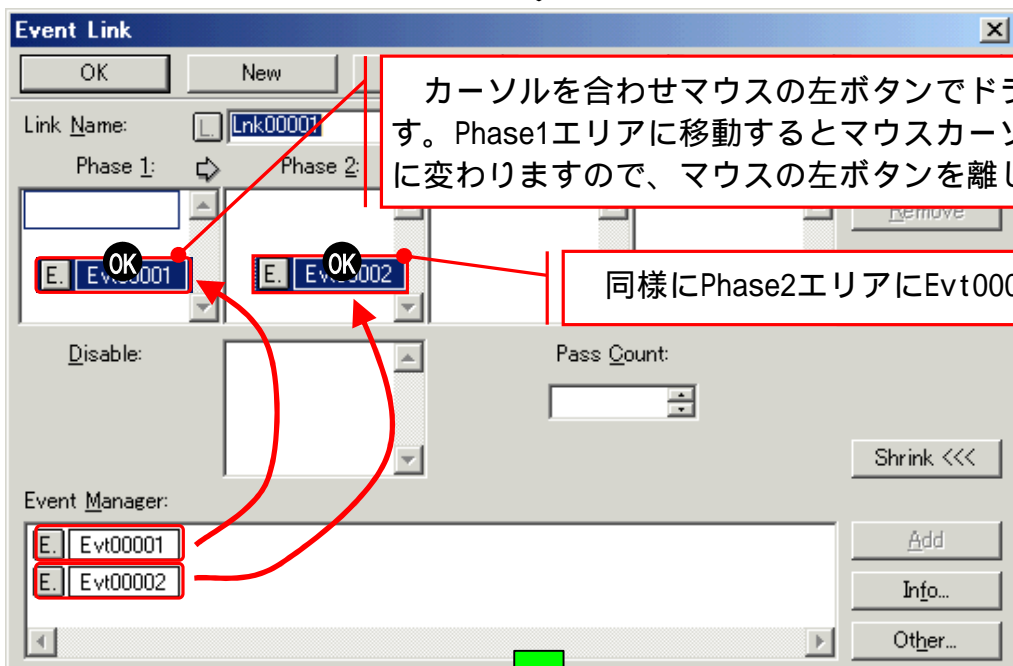
# 更に高度なデバッグ機能

p. イベント・リンクを設定します。

メニュー[イベント(N)] [イベントリンク(L)...]で、[Event Link]ダイアログを表示して下さい。

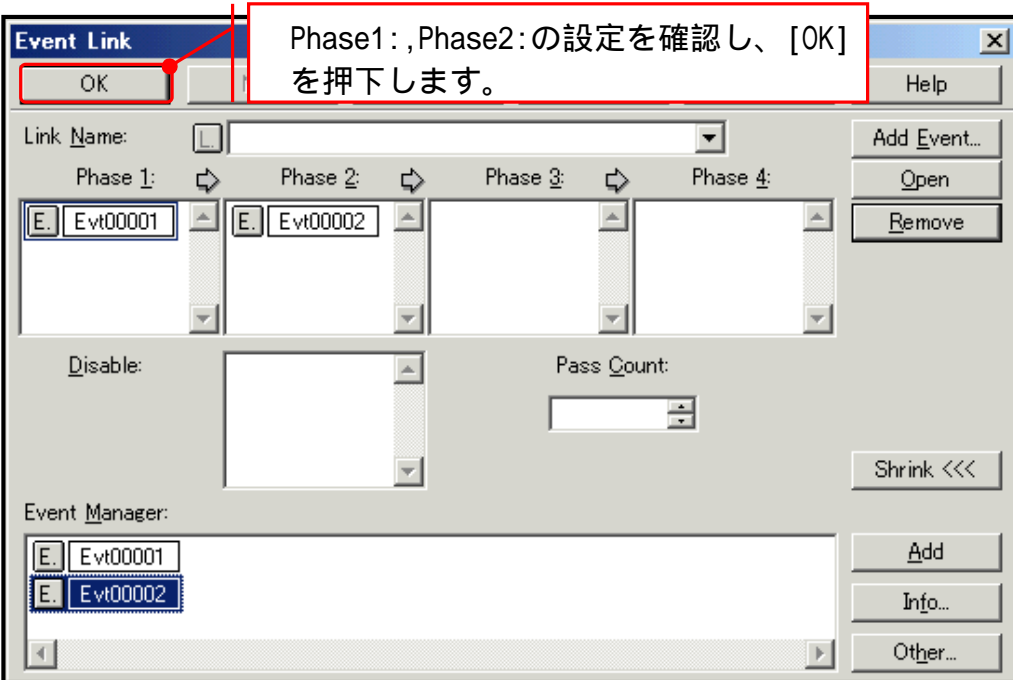


[Event Link]ダイアログを開きます。



カーソルを合わせマウスの左ボタンでドラッグします。Phase1エリアに移動するとマウスカーソルが(OK)に変わりますので、マウスの左ボタンを離して下さい。

同様にPhase2エリアにEvt00002を設定します。



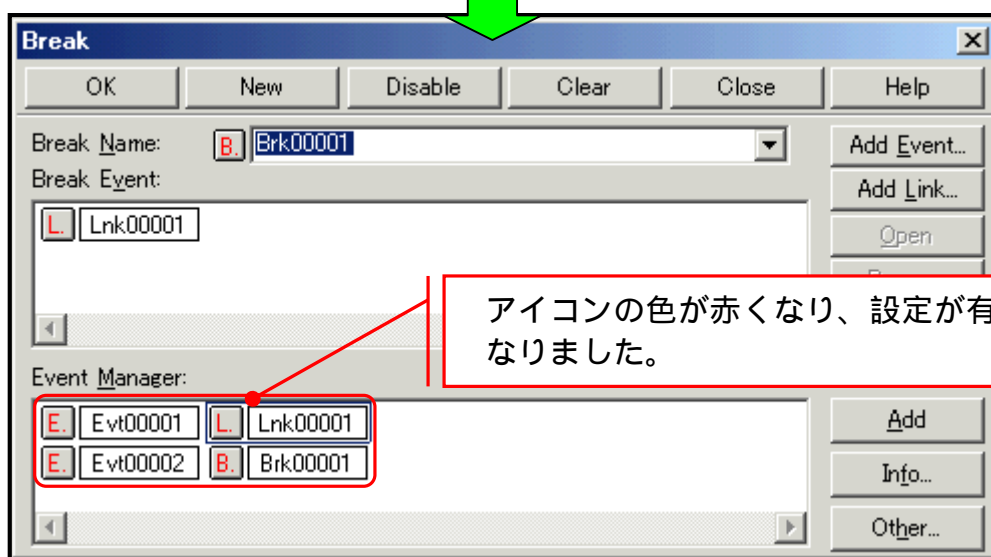
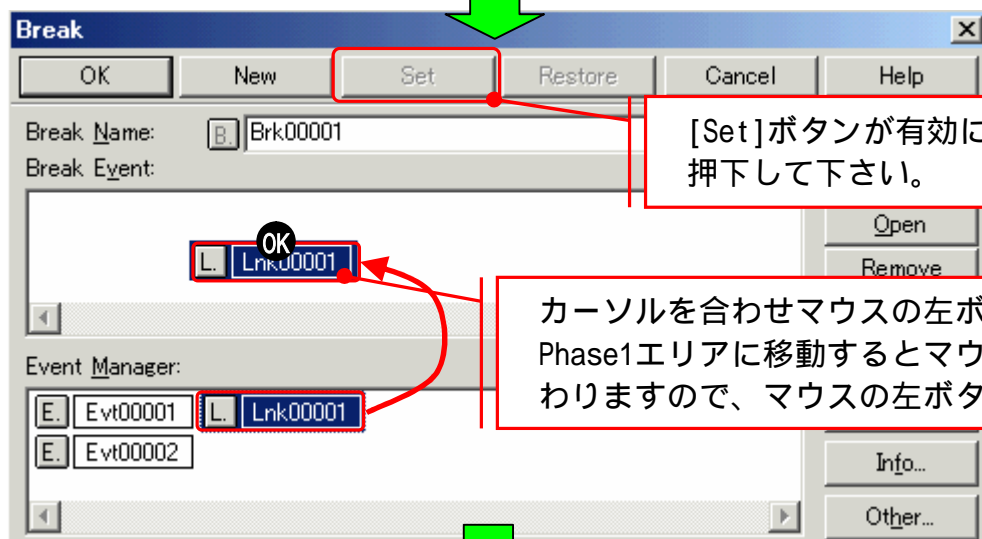
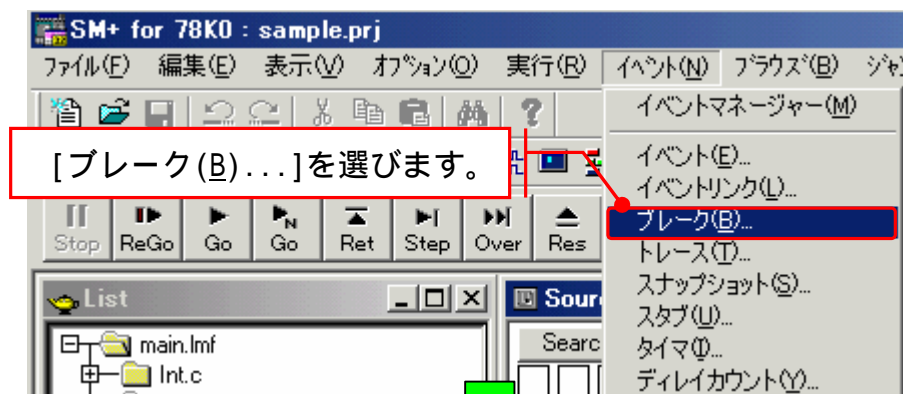
Phase1: ,Phase2: の設定を確認し、[OK]を押下します。



# 更に高度なデバッグ機能

q. ブレークポイントを設定します。

メニュー[イベント(N)] [ブレーク(B)...]で、[Break]ダイアログを表示して下さい。

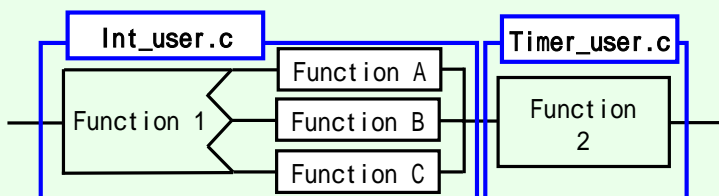




# 更に高度なデバッグ機能

- r. 実行して入出力パネルのSW1を押下します。  
 プログラムを実行してSW1を押下するとTimer\_user.cの77行目で停止します。

以上のように、Int\_user.c(Function1)を通り、Timer\_user.c(Function2)で指定したイベントリンクで停止しました。単にTimer\_user.c(Function2) にブレーク・ポイントを設定すると、毎回200ms毎に停止してしまうのですが、 イベントリンクを使うことによってSW1を押した時だけ停止させることができました。





# 電光掲示板を作る

この章ではターゲット・システム作成例1を紹介します。ドットマトリクスLEDを使用した電光掲示板を作成します。マイコンに表示機能とSWがあればゲーム作成などにも応用可能です。ターゲット・システム作成例2では、電光掲示板を応用した簡単なゲームを作成しています。

ターゲット・ボードを使用した回路を作成します。ここでは8x8のドットマトリクスLEDをダイナミック点灯させます。ダイナミック点灯とは表示を分割して(今回は8個のLED)行う方法です。表示を高速に順次繰り返す事によって人の目には全てが表示されているように見えます。

ダイナミック点灯には下記の特徴があります。

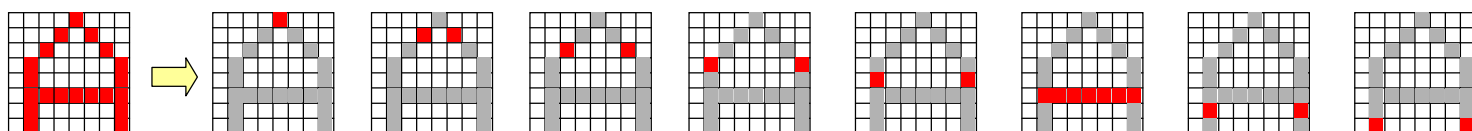
利点: 常時点灯していないので消費電力が少ない。ポート制御が少なく済む。

欠点: スタティック(常時)点灯に比べて表示が暗い。

## ダイナミック点灯の原理

人の目に見える表示

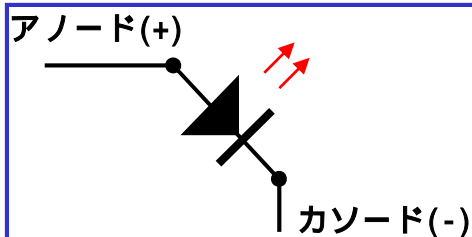
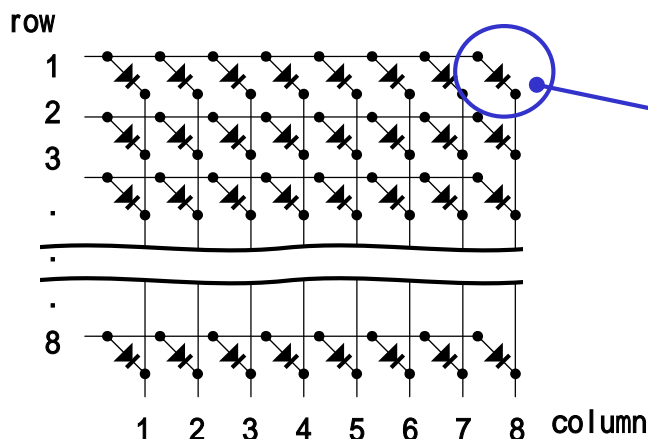
高速に表示を繰り返す



## ワンポイント

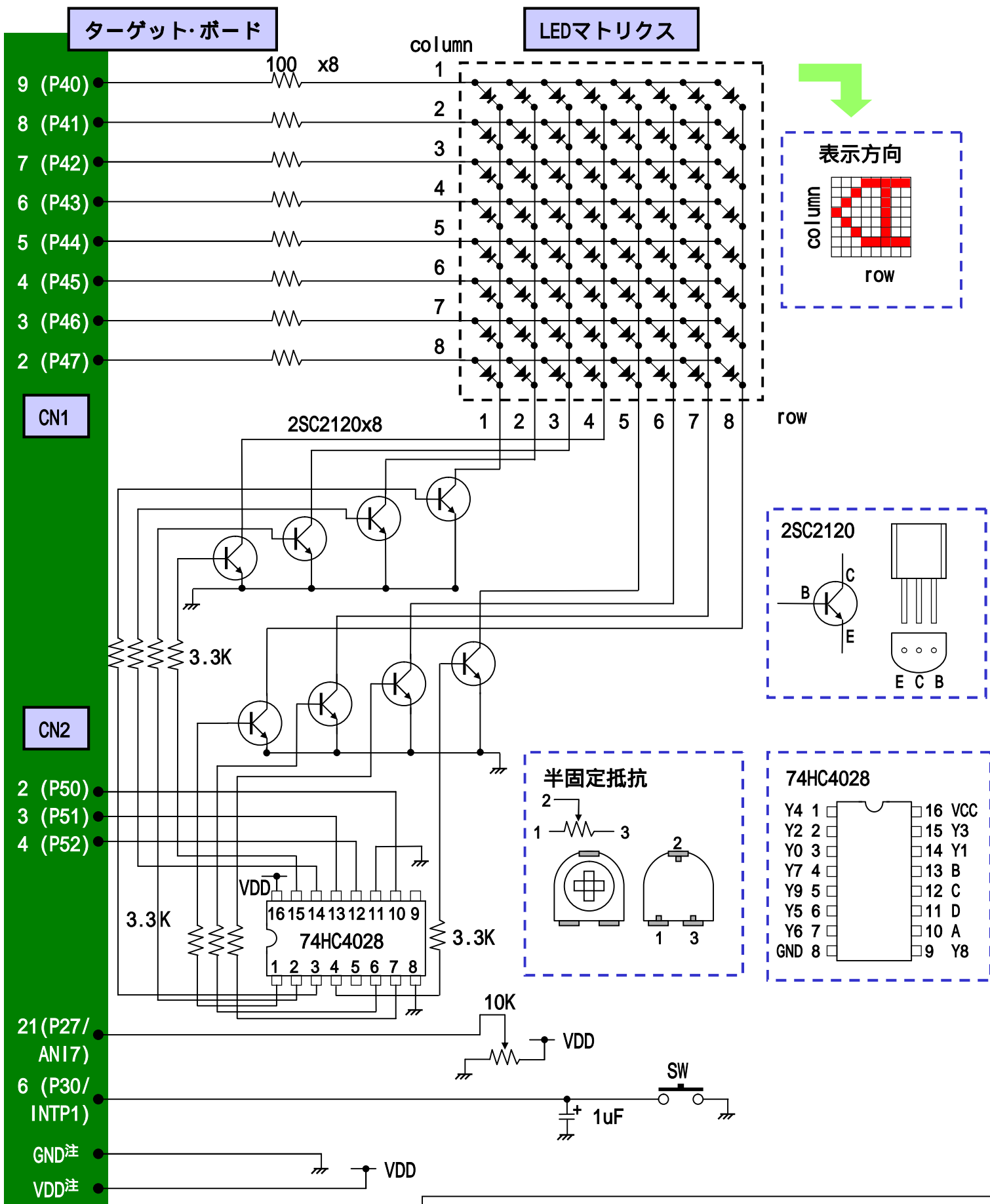
ドットマトリクスLEDについて

ドットマトリクスLEDとはLEDがマトリクス(matrix)状に並んだ表示器です。8x8ならLEDが行(column) 8個、列(row) 8個が格子状に64個並んでいます。



例えば、上記の青丸内のLEDを点灯させるにはrow1に+(プラス)、column8に-(GND)を接続すれば点灯します。LEDを1つ点灯させるには通常10mA ~ 20mA必要です(LEDの色、LEDの特性によって異なります)。これを+5Vで供給するには抵抗150 ~ 300 を接続します。マトリクスLEDは1列に8個接続されていますので80mA ~ 160mAが必要です。これを+5Vで供給するには抵抗37.5 ~ 75 を接続します。しかし、マイコンの1ポートに80mA ~ 160mAを流せませんので何らかの回路(トランジスタ制御など)が必要になります。

# 回路図



注: ターゲットボード上のGND, VDD端子 (POWER LEDの左隣にあります) と接続する。

# 回路説明

ドットマトリクスLEDは8個のLEDを点灯させるため、マイコンで直接ドライブできませんのでトランジスタを使用します。

74HC4028はBCDコードを10進化します。ポート50～52からの出力を74HC4028のABCで受け、その結果をY0～Y7へ変換します。ドットマトリクスLEDのrowは0～7の値ですのでY8, Y9は不要です。そのため74HC4028のD入力はGNDに接します。

更にA/Dコンバータ(AN17)を使用します。半固定抵抗の抵抗値によって電光掲示板の流れる速度を調整できるようにします。

INTP1に接続しているSWは表示切り替え用です。SWにはチャタリング防止回路を入れてあります。(内蔵のプルアップ抵抗を使い、コンデンサのみを使用した簡易なチャタリング防止です。)

電源はMINICUBE2より供給します。ただし供給可能な電流は100mAなので、使用する部品によって100mAを超える場合は「ターゲット作成例1」の[動作概要]のコラムを参照して外部電源を利用して下さい。

## 💡ワンポイント

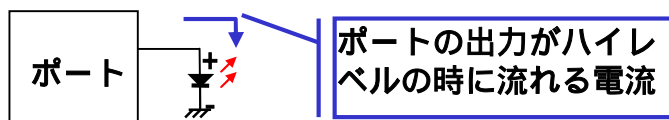
### シンク電流、ソース電流について

78K0/KF2のポートはソース電流で10mA～55mA(端子合計80mA)、シンク電流で30mA～140mA(端子合計200mA)の容量があります(ポート番号によって異なります)。

シンク電流( $I_{OL}$ )



ソース電流( $I_{OH}$ )

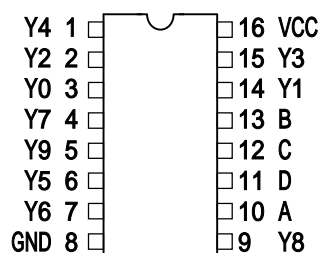


### 74HC4028について

BCD TO DECIMAL DECODERです。下図のABCDの入力に応じてY0～Y9がH(ハイレベル)になります

BCD入力				DECIMAL出力									
A	B	C	D	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7	Y8	Y9
L	L	L	L	H	L	L	L	L	L	L	L	L	L
H	L	L	L	L	H	L	L	L	L	L	L	L	L
L	H	L	L	L	L	H	L	L	L	L	L	L	L
H	H	L	L	L	L	L	H	L	L	L	L	L	L
L	L	H	L	L	L	L	L	H	L	L	L	L	L
H	L	H	L	L	L	L	L	L	H	L	L	L	L
L	H	H	L	L	L	L	L	L	L	H	L	L	L
H	H	H	L	L	L	L	L	L	L	L	H	L	L
L	L	L	H	L	L	L	L	L	L	L	L	H	L
H	L	L	H	L	L	L	L	L	L	L	L	L	H

74HC4028



# プログラム設計

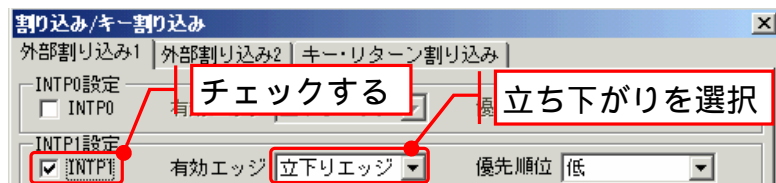
## a. Applilet2を使いプログラムを設計します。

[動かしてみよう]の[Applilet2でプロジェクト作成]を参照して、プロジェクトを作成して下さい。ここではプロジェクト名を[extend]とします。[動かしてみよう]の[周辺機能設定(システム)]と同様に設定して下さい。下記の設定になります。

- ・高速システム・クロックX1発振20MHzを使用
- ・オンチップ・デバッグを使用し、RAMモニタ機能も使用
- ・ウォッチドッグ・タイマは使用しない

## b. [割り込み]を設定します。

[割り込み]ダイアログは、外部割り込みを設定します。

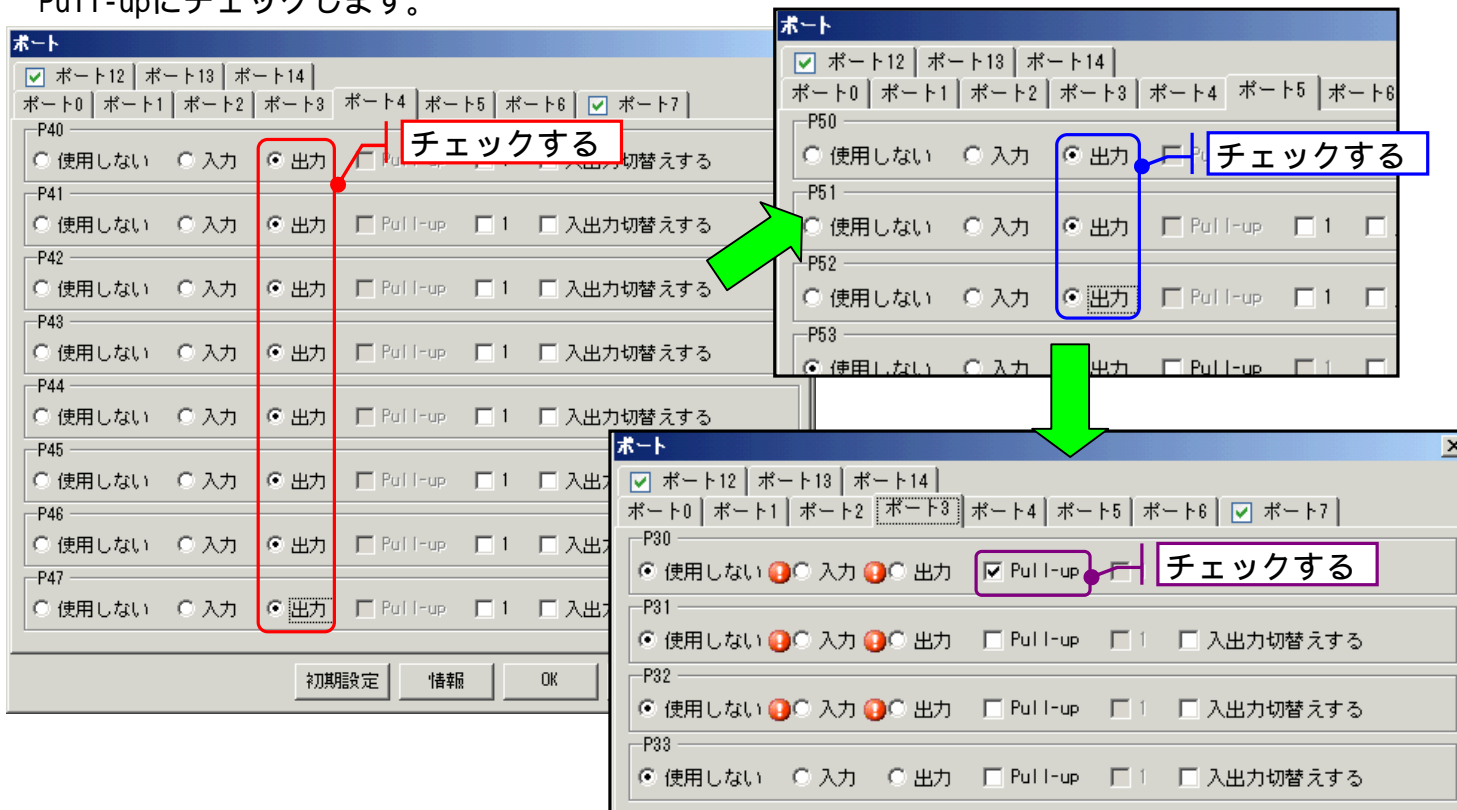


### 💡ワンポイント

INTP1兼用端子のP46の設定も必要です。INTP1を立ち下りエッジで使用するので、信号をHigh、P30をPull-upしておく必要があります。抵抗を端子に接続してもよいのですが、マイコンの機能を利用して回路を省略しています。

## c. [ポート]を設定します。

ポート4[P40~P47]の出力にチェック、ポート5[P50~P52]の出力にチェック、ポート3[P30]のPull-upにチェックします。



# プログラム設計

## d. [タイマ]を設定します。

タイマTM00、TM01へインターバル・タイマ1msecを設定します。他はデフォルトの設定です。

タイマ

機能

- 使用しない
- インターバル・タイマ
- PPG出力
- パルス幅測定設定
- 外部イベント・カウンタ
- 方形波出力
- ワンショット・パルス出力
- フリー・ラン

詳細 初期設定 情報 OK

タイマ00 インターバル・タイマ

クロック・モード選択

- 内部クロック
  - 自動
  - fprs
  - fprs/4
  - fprs/256
- 外部クロック
  - TI000 立下りエッジ
  - TI000 立上りエッジ
  - TI000 両

インターバル時間設定

インターバル時間: 1 msec

(実際の値: 1)

割り込み設定

- 設定した周期毎に通知する

優先順位: 低

初期設定 情報 OK Cancel ヘルプ

TM00のインターバル・タイマにチェックする

[詳細設定]を押下する

「1」を入力

「msec」へ変更

タイマTM01も同様に設定します。

タイマ

機能

- 使用しない
- インターバル・タイマ
- PPG出力
- パルス幅測定設定
- 外部イベント・カウンタ
- 方形波出力
- ワンショット・パルス出力
- フリー・ラン

詳細 初期設定 情報 OK Cancel

タイマ01 インターバル・タイマ

クロック・モード選択

- 内部クロック
  - 自動
  - fprs
  - fprs/16
  - fprs/64
- 外部クロック
  - TI001 立下りエッジ
  - TI001 立上りエッジ
  - TI001 両

インターバル時間設定

インターバル時間: 1 msec

(実際の値: 1)

割り込み設定

- 設定した周期毎に通知する

優先順位: 低

初期設定 情報 OK Cancel ヘルプ

TM01のインターバル・タイマにチェックする

[詳細設定]を押下する

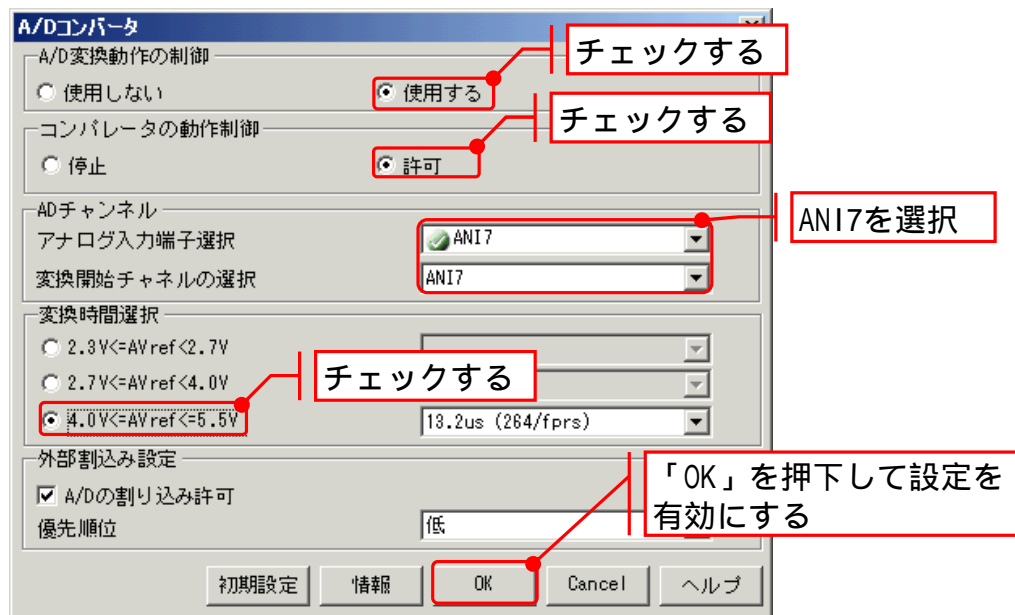
「1」を入力

「msec」へ変更

# プログラム設計

## e. [A/Dコンバータ]を設定します。

A/Dを使用する、コンパレータの動作制御を許可、アナログ入力端子選択ANI7、変換開始チャンネルの選択をANI7、変換時間選択を  $4.0V \leq AVref \leq 5.5V$  にチェックします。他はデフォルトの設定です。



## f. コード生成します。

[動かしてみよう]の[コード生成]を参考にコードを生成を行い、プログラムの編集をします。プログラムの編集については、[動かしてみよう]の[プログラムの編集]を参考にしてください。

編集するファイルはApplilet2生成プログラム中の下記に示すファイルです。

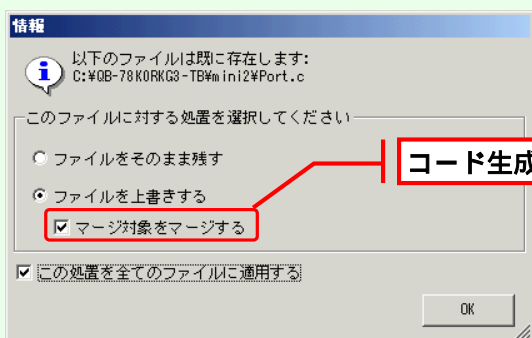
- user\_define.h ユーザが定義する定数などを書く
- main.c ユーザのメイン関数を含む
- Int\_user.c ユーザが書く割り込み処理
- Timer\_user.c ユーザが書くタイマ割り込み処理
- Ad\_user.c ユーザが書くA/D変換終了の割り込み処理

プログラムを作成する時は、下記コメント間にプログラムを書いて下さい。

```
void function ( void )
{
/* Start user code. Do not edit comment generated here */
}
/* End user code adding. Do not edit comment generated here */
}
```

この場所にプログラムを書きます。Applilet2のコード生成で上書きされることはありません。

Applilet2でコード生成する際に、既存のコードに「マージ」することができます。上記以外の場所にプログラムを書くと、「ファイルを上書きする」にチェックしてもマージされず、コードが消去されます。



# プログラムリスト

## g. プログラムを編集して下さい。

下記に示す青字のコードを追加して下さい

### user\_define.h

#### リスト省略

```

#ifndef _MD_USER_DEF_
#define _MD_USER_DEF_
/*
*****
** Macro define
*****
*/
/* Start user code for macro definition. Do not edit comment generated here */

#define D_MLED_CNT          2 /* Matrix LED number */
#define D_MLED_ROW         8 /* Matrix LED row */
#define D_FONT_WIDTH       6 /* font width */
#define D_MAXTEXT          34 /* Max display of text */

extern UINT    g_timer0_counter;
extern UINT    g_timer1_counter;
extern UCHAR   g_text_sw;
extern UCHAR   g_text_cnt;
extern UINT    g_text_scrollcnt;
extern USHORT  g_text_speed;
extern UCHAR   g_matrix_ram[ D_MLED_CNT ][ D_MLED_ROW ];
extern const UCHAR g_textdata[][ D_MAXTEXT ];

/* End user code for macro definition. Do not edit comment generated here */
#endif

```

### main.c(リスト1)

#### リスト省略

```

/*
*****
** Include files
*****
*/
#include "macrodriver.h"
#include "System.h"
#include "Port.h"
#include "Int.h"
#include "Ad.h"
#include "Timer.h"
/* Start user code for include definition. Do not edit comment generated here */

#include "string.h"
#include "ctype.h"

/* End user code for include definition. Do not edit comment generated here */
#include "user_define.h"

/*
*****
** Global define
*****
*/
/* Start user code for global definition. Do not edit comment generated here */

void initialize_value( void );
void initialize_text( void );
void disp_putfont( UCHAR font_ );
void disp_move1dot(void );

```

# プログラムリスト

## main.c(リスト2)

```

const UCHAR g_font_numeric[] =
{ /* font data 0 - 9 */
    0x00, 0x0E, 0x11, 0x13, 0x15, 0x19, 0x11, 0x0E /* 0 */
, 0x00, 0x04, 0x0C, 0x04, 0x04, 0x04, 0x04, 0x0E /* 1 */
, 0x00, 0x0E, 0x11, 0x01, 0x0E, 0x10, 0x10, 0x1F /* 2 */
, 0x00, 0x0E, 0x11, 0x01, 0x0E, 0x01, 0x11, 0x0E /* 3 */
, 0x00, 0x02, 0x06, 0x0A, 0x12, 0x1F, 0x02, 0x02 /* 4 */
, 0x00, 0x1F, 0x10, 0x10, 0x1E, 0x01, 0x01, 0x1E /* 5 */
, 0x00, 0x0F, 0x10, 0x10, 0x1E, 0x11, 0x11, 0x0E /* 6 */
, 0x00, 0x1F, 0x01, 0x02, 0x04, 0x04, 0x04, 0x04 /* 7 */
, 0x00, 0x0E, 0x11, 0x11, 0x1E, 0x11, 0x11, 0x0E /* 8 */
, 0x00, 0x0E, 0x11, 0x11, 0x0F, 0x01, 0x01, 0x0E /* 9 */
};

const UCHAR g_font_alphabet[] =
{ /* font data A - Z */
    0x00, 0x04, 0x0A, 0x0A, 0x11, 0x1F, 0x11, 0x11 /* A */
, 0x00, 0x1E, 0x11, 0x11, 0x1E, 0x11, 0x11, 0x1E /* B */
, 0x00, 0x0E, 0x11, 0x10, 0x10, 0x10, 0x11, 0x0E /* C */
, 0x00, 0x1C, 0x12, 0x11, 0x11, 0x11, 0x12, 0x1C /* D */
, 0x00, 0x1F, 0x10, 0x10, 0x1E, 0x10, 0x10, 0x1F /* E */
, 0x00, 0x1F, 0x10, 0x10, 0x1E, 0x10, 0x10, 0x10 /* F */
, 0x00, 0x0E, 0x11, 0x10, 0x17, 0x11, 0x11, 0x0E /* G */
, 0x00, 0x11, 0x11, 0x11, 0x1F, 0x11, 0x11, 0x11 /* H */
, 0x00, 0x0E, 0x04, 0x04, 0x04, 0x04, 0x04, 0x0E /* I */
, 0x00, 0x02, 0x02, 0x02, 0x02, 0x02, 0x12, 0x0C /* J */
, 0x00, 0x11, 0x12, 0x14, 0x18, 0x14, 0x12, 0x11 /* K */
, 0x00, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x1F /* L */
, 0x00, 0x11, 0x11, 0x1B, 0x15, 0x11, 0x11, 0x11 /* M */
, 0x00, 0x11, 0x11, 0x19, 0x15, 0x13, 0x11, 0x11 /* N */
, 0x00, 0x0E, 0x11, 0x11, 0x11, 0x11, 0x11, 0x0E /* O */
, 0x00, 0x1E, 0x11, 0x11, 0x1E, 0x10, 0x10, 0x10 /* P */
, 0x00, 0x0E, 0x11, 0x11, 0x11, 0x15, 0x13, 0x0F /* Q */
, 0x00, 0x1E, 0x11, 0x11, 0x1E, 0x14, 0x12, 0x11 /* R */
, 0x00, 0x0E, 0x11, 0x10, 0x0E, 0x01, 0x11, 0x0E /* S */
, 0x00, 0x1F, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04 /* T */
, 0x00, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x0E /* U */
, 0x00, 0x11, 0x11, 0x11, 0x11, 0x11, 0x0A, 0x0A /* V */
, 0x00, 0x11, 0x15, 0x15, 0x15, 0x15, 0x15, 0x0A /* W */
, 0x00, 0x11, 0x11, 0x0A, 0x04, 0x0A, 0x11, 0x11 /* X */
, 0x00, 0x11, 0x11, 0x0A, 0x04, 0x04, 0x04, 0x04 /* Y */
, 0x00, 0x1F, 0x01, 0x02, 0x04, 0x08, 0x10, 0x1F /* Z */
};

const UCHAR g_textdata[][ D_MAXTEXT ] =
{ /* text data */
    "THIS IS 78K0KF2 TARGET BOARD "
, "DISPLAY CHANGES IF SW IS PUSHED "
, "THIS IS PROGRAM SAMPLE OF NECEL "
, "abcdefghijklmnopqrstuvwxy "
, "0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6"
, 0
};

/* Interval timer counter */
UINT g_timer0_counter = 0;
UINT g_timer1_counter = 0;

/* control for text */
USHORT g_text_speed = 0;
UCHAR g_text_sw = 0;
UCHAR g_text_cnt = 0;
UINT g_text_scrollcnt = 0;

```



# プログラムリスト

## main.c(リスト3)

```
/* g_matrix_ram[0] is real vram. [1] is buffer vram. */
UCHAR g_matrix_ram[ D_MLED_CNT ][ D_MLED_ROW ];

/* End user code for global definition. Do not edit comment generated here */

/*
** -----
** Abstract:
**          This function implements main function.
** Parameters:
**          None
** Returns:
**          None
** -----
*/
void main( void )
{
    /* Start user code. Do not edit comment generated here */

    /* Target-Board LED off */
    P7.6 = 1;
    P7.7 = 1;

    initialize_value();

    initialize_text();

    /* start timer */
    TMO0_Start();
    TMO1_Start();
    while (1) {
        ;
    }

    /* End user code. Do not edit comment generated here */
}

/* Start adding user code. Do not edit comment generated here */

/* ----- */
void initialize_value( void )
{
    memset( g_matrix_ram, 0x00, sizeof( g_matrix_ram ) );
    g_timer0_counter = 0;
    g_timer1_counter = 0;
    g_text_sw = 0;
}

/* ----- */
void initialize_text( void )
{
    UCHAR font;

    memset( g_matrix_ram, 0x00, sizeof( g_matrix_ram ) );
    g_text_scrollcnt = 0;
    g_text_cnt = 0;

    font = g_textdata[ g_text_sw ][ 0 ];
    disp_putfont( font );
}
```

# プログラムリスト

## main.c(リスト4)

```
/* ----- */
/* 1 dot is scrolled. */

void disp_move1dot(void )
{
    int i;
    UCHAR vtmp, vtmp2;

    for ( i=0; i<D_MLED_ROW; i++ )
    {
        vtmp = ( g_matrix_ram[ 0 ][ i ] & 0x7f) << 1;
        vtmp2 = ( g_matrix_ram[ 1 ][ i ] & 0x80) >> 7;
        g_matrix_ram[ 0 ][ i ] = vtmp + vtmp2;
        vtmp2 = ( g_matrix_ram[ 1 ][ i ] & 0x7f) << 1;
        g_matrix_ram[ 1 ][ i ] = vtmp2;
    }
}

/* End user code adding. Do not edit comment generated here */
```

# プログラムリスト

## Int\_user.c

### リスト省略

```
#pragma interrupt INTP1 MD_INTP1
/*
*****
** Include files
*****
*/
#include "macrodriver.h"
#include "Int.h"
/* Start user code for include definition. Do not edit comment generated here */

#include "string.h"
#include "ctype.h"

/* End user code for include definition. Do not edit comment generated here */
#include "user_define.h"

/*
*****
** Global define
*****
*/
/* Start user code for global definition. Do not edit comment generated here */

extern void initialize_text( void );

/* End user code for global definition. Do not edit comment generated here */

/*
** -----
** Abstract:
**          This function is INTP1 interrupt service routine.
** Parameters:
**          None
** Returns:
**          None
** -----
*/
__interrupt void MD_INTP1( void )
{
    /* Start user code. Do not edit comment generated here */
    /* TODO. Add user defined interrupt service routine */

    g_text_sw++;
    if ( g_textdata[ g_text_sw ][ 0 ] == 0 )
    {
        g_text_sw = 0;
    }

    memset( g_matrix_ram, 0x00, sizeof( g_matrix_ram ) );
    initialize_text();

    /* End user code. Do not edit comment generated here */
}

/* Start adding user code. Do not edit comment generated here */
/* End user code adding. Do not edit comment generated here */
```

# プログラムリスト

## Timer\_user.c(リスト1)

### リスト省略

```

#pragma interrupt INTTM000 MD_INTTM000
#pragma interrupt INTTM001 MD_INTTM001
/*
*****
** Include files
*****
*/
#include "macrodriver.h"
#include "Timer.h"
/* Start user code for include definition. Do not edit comment generated here */

#include "Ad.h"

/* End user code for include definition. Do not edit comment generated here */
#include "user_define.h"

/*
*****
** Global define
*****
*/
/* Start user code for global definition. Do not edit comment generated here */

extern void disp_putfont( UCHAR font_ );
extern void disp_move1dot( void );

/* End user code for global definition. Do not edit comment generated here */

/*
** -----
** Abstract:
**           This function is INTTM000 interrupt service routine.
** Parameters:
**           None
** Returns:
**           None
** -----
*/
__interrupt void MD_INTTM000( void )
{
    /* Start user code. Do not edit comment generated here */
    /* TODO. Add user defined interrupt service routine */

    UCHAR row, line;

    /* Matrix LED line disp */
    row = (UCHAR)(g_timer0_counter) & 7;

    /* display column */
    P4 = 0;
    line = g_matrix_ram[ 0 ][ row ];
    P4 = line;

    /* display row */
    P5.0 = ( row & 1 );
    P5.1 = (( row & 2 ) >> 1);
    P5.2 = (( row & 4 ) >> 2);

    g_timer0_counter++;

    /* End user code. Do not edit comment generated here */
}

```

# プログラムリスト

## Timer\_user.c(リスト2)

```
/*
**-----
** Abstract:
**          This function is INTTM001 interrupt service routine.
** Parameters:
**          None
** Returns:
**          None
**-----
*/
__interrupt void MD_INTTM001( void )
{
    /* Start user code. Do not edit comment generated here */
    /* TODO. Add user defined interrupt service routine */

    UCHAR font;

    AD_Start();

    g_timer1_counter++;
    if ( g_timer1_counter > g_text_speed )
    {
        if ( ( g_text_scrollcnt % D_FONT_WIDTH ) == 0 )
        {
            /* next font */
            font = g_textdata[ g_text_sw ][ g_text_cnt ];
            disp_putfont( font );

            g_text_cnt++;
            if ( g_text_cnt >= D_MAXTEXT )
                g_text_cnt = 0;
        }

        /* scroll */
        disp_move1dot();

        g_text_scrollcnt++;
        g_timer1_counter = 0;
    }

    /* End user code. Do not edit comment generated here */
}

/* Start adding user code. Do not edit comment generated here */
/* End user code adding. Do not edit comment generated here */
```

# プログラムリスト

## Ad\_user.c

### リスト省略

```
#pragma interrupt INTAD MD_INTAD
/*
*****
** Include files
*****
*/
#include "macrodriver.h"
#include "Ad.h"
/* Start user code for include definition. Do not edit comment generated here */
/* End user code for include definition. Do not edit comment generated here */
#include "user_define.h"

/*
*****
** Global define
*****
*/
/* Start user code for global definition. Do not edit comment generated here */
/* End user code for global definition. Do not edit comment generated here */

/*
** -----
** Abstract:
**          This function is INTAD interrupt service routine.
** Parameters:
**          None
** Returns:
**          None
** -----
*/
__interrupt void MD_INTAD( void )
{
    /* Start user code. Do not edit comment generated here */
    /* TODO. Add user defined interrupt service routine */

    USHORT adval;

    AD_Stop();

    AD_Read( &adval );
    g_text_speed = adval/2 + 1;
    if ( g_text_speed > 500 )
    {
        g_text_speed = adval;
    }

    /* End user code. Do not edit comment generated here */
}

/* Start adding user code. Do not edit comment generated here */
/* End user code adding. Do not edit comment generated here */
```

# プログラム説明

## h. プログラムで使用している関数について説明します。

### main.c

```
const UCHAR g_font_numeric[] /* font data 0 - 9 */
マトリックスLEDに表示する数字フォントデータです

const UCHAR g_font_alphabet[] /* font data A - Z */
マトリックスLEDに表示する英字(大文字のみ)フォント
データです

const UCHAR g_textdata[][D_MAXTEXT] /* text data */
マトリックスLEDに表示するテキストデータです

UINT g_timer0_counter = 0;
UINT g_timer1_counter = 0;
タイマ割り込み時に+1されるカウンタです

UCHAR g_text_sw = 0;
表示するテキストを選択します

UCHAR g_text_cnt = 0;
表示されているテキストが何文字目かを示します

UINT g_text_scrollcnt = 0;
表示されている文字が何ドット移動したかを示します

USHORT g_text_speed = 0;
スクロールするスピードを示します

UCHAR g_matrix_ram[ D_MLED_CNT ][ D_MLED_ROW ];
マトリックスLEDに表示するデータです。8x16ドット分
ありますが実際に表示されるのは8x8ドットです

void initialize_value( void )
void initialize_text( void )
変数初期化を行う関数です

void disp_putfont( UCHAR font_ )
パラメータ font_(asciiコード)で指定された文字を
マトリックスLEDの非表示領域へ出力します

void disp_move1dot(void )
マトリックスLEDの表示を1ドット移動(スクロール)します
```

### Timer\_user.c

```
__interrupt void MD_INTTM000()
1msec毎に呼ばれる関数です。
マトリックスLEDにg_matrix_ram よりデータを読み込み
1ライン表示します

__interrupt void MD_INTTM001()
1msec毎に呼ばれる関数です。下記の処理を行います。
・A/D変換を開始します
・移動のチェックを行い、1文字分のスクロールが終了
したら disp_putfont を呼び出し、指定された文字を
マトリックスLEDの非表示領域へ出力します。
そうでなければ disp_move1dot を呼び出し、1ドット
分スクロールします。
```

### Ad\_user.c

```
__interrupt void MD_INTAD( void )
A/D変換終了時に呼ばれる関数です。
A/D値を読み込み g_text_speed へ反映させます
```

### Int\_user.c

```
__interrupt void MD_INTP1( void )
外部割り込みINTP1に呼ばれる関数です。
SWが押下されたら g_text_sw を+1し、表示するテキス
トを変更します。
```

### user\_define.h

```
#define D_MLED_CNT 2 /* Matrix LED number */
マトリックスLEDの数を定義します。実際は1つですが仮
想的に2つ持っていることにしています。

#define D_MLED_ROW 8 /* Matrix LED row */
マトリックスLEDのROWの数

#define D_FONT_WIDTH 6 /* font width */
1文字のフォントの幅

#define D_MAXTEXT 34 /* Max display of text */
表示するメッセージの長さ
```

# 動作概要

## g. 全体の処理

プログラムの動作として、下記の5つに分けられます

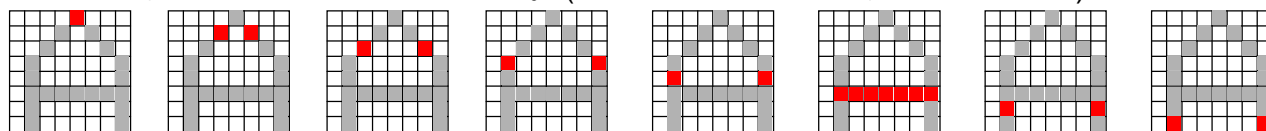
1. メイン処理
2. タイマ00(インターバル・タイマとして使用)
3. タイマ01(インターバル・タイマとして使用)
4. A/Dコンバータ完了割り込み
5. INTP1外部割り込み

### 1. メイン処理 `void main( void )`

変数/マトリックスLEDの初期化、タイマ00/01を開始します。  
処理としては無限ループとなります。

### 2. タイマ00 `__interrupt void MD_INTTM000( )`

1msec毎に起動します。 `g_matrix_ram[0][0~7]`の1byteをマトリックスLEDの1ラインへ出力します。呼ばれる毎に1ライン表示を行います。(下図の点灯が1msec毎に行われます)



### 3. タイマ01 `__interrupt void MD_INTTM001( )`

1msec毎に起動します。

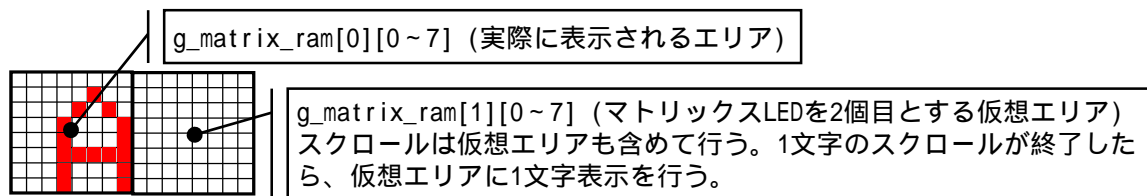
A/D変換の開始を行います。A/D変換が完了するとA/D変換終了割り込みが発生します。

A/D変換の結果は `g_text_speed` へ反映されるので、それを元にスクロールスピードを決めます

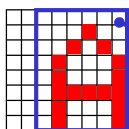
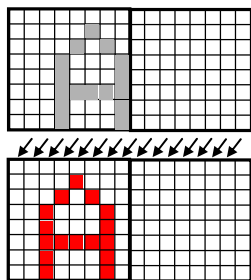
タイマ01は呼ばれる毎に `g_timer1_counter` を+1していますので、`g_text_speed`と`g_timer1_counter`の値を比較してスクロールを行うか行わないかを判断しています。

スクロールを行う場合は2通りの処理があります。

- a. マトリックスLEDへ表示されているデータを1ドット左へ移動する
- b. 1文字のスクロールが終了したので新たな文字を表示する



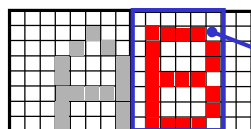
#### a. マトリックスLEDへ表示されているデータを1ドット左へ移動する



1フォントは縦8x横6ドットで構成されています。a.の処理は1文字のスクロール処理(6ドット左へ移動)のうち5ドット左移動までの分を行います。

1文字のスクロール処理開始時は b.1文字のスクロールが終了したので新たな文字を表示する の処理を行います。

#### b. 1文字のスクロールが終了したので新たな文字を表示する



1フォントは縦8x横6ドットで構成されています。1文字のスクロール開始時にはマトリックスLEDを2個目とする仮想エリアへ1文字出力します。



# 動作概要

## 4. A/D変換終了割り込み `__interrupt void MD_INTAD( void )`

タイマ01で開始されたA/D変換が終了時に呼ばれます。

A/D変換の中止を行い、変換結果を読み込みます。

結果は `g_text_speed` へ代入されます。`g_text_speed` の値は 0 以外を想定していますので、念の為変換結果に +1 を行います。表示するスピードが調整しやすいよう変換結果を変えています。

## 5. INTP1外部割り込み `__interrupt void MD_INTP1( void )`

INTP1端子がLOWになった場合(SWが押下された場合)に呼ばれます。

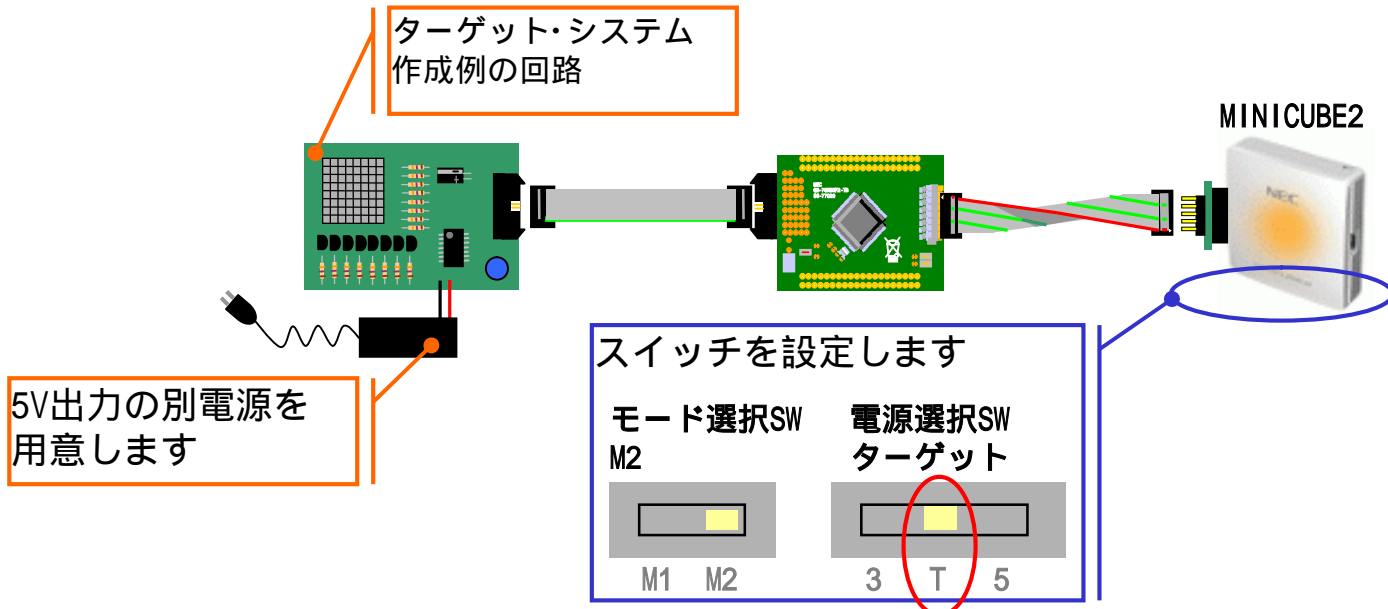
スクロールするテキストを変えます。表示させるテキストは1文が32文字で構成されています。

`g_textdata[][D_MAXTEXT]`の内容を書き換えればマトリックスLEDへ表示させる事ができますが、表示するデータは必ず32byteで構成して下さい。

## 💡 ワンポイント

### 電源供給について

MINICUBE2より電源供給する場合5V供給で最大定格100mAです。ターゲット・システム作成例の回路で使用する部品によっては100mAを超える場合があります。その場合は5V電源を別に用意して下さい。ターゲット・システム側の電源を使用する場合はMINICUBE2の「電源選択SW」を「T」に設定して下さい。



下記の順番で接続を行ってください。

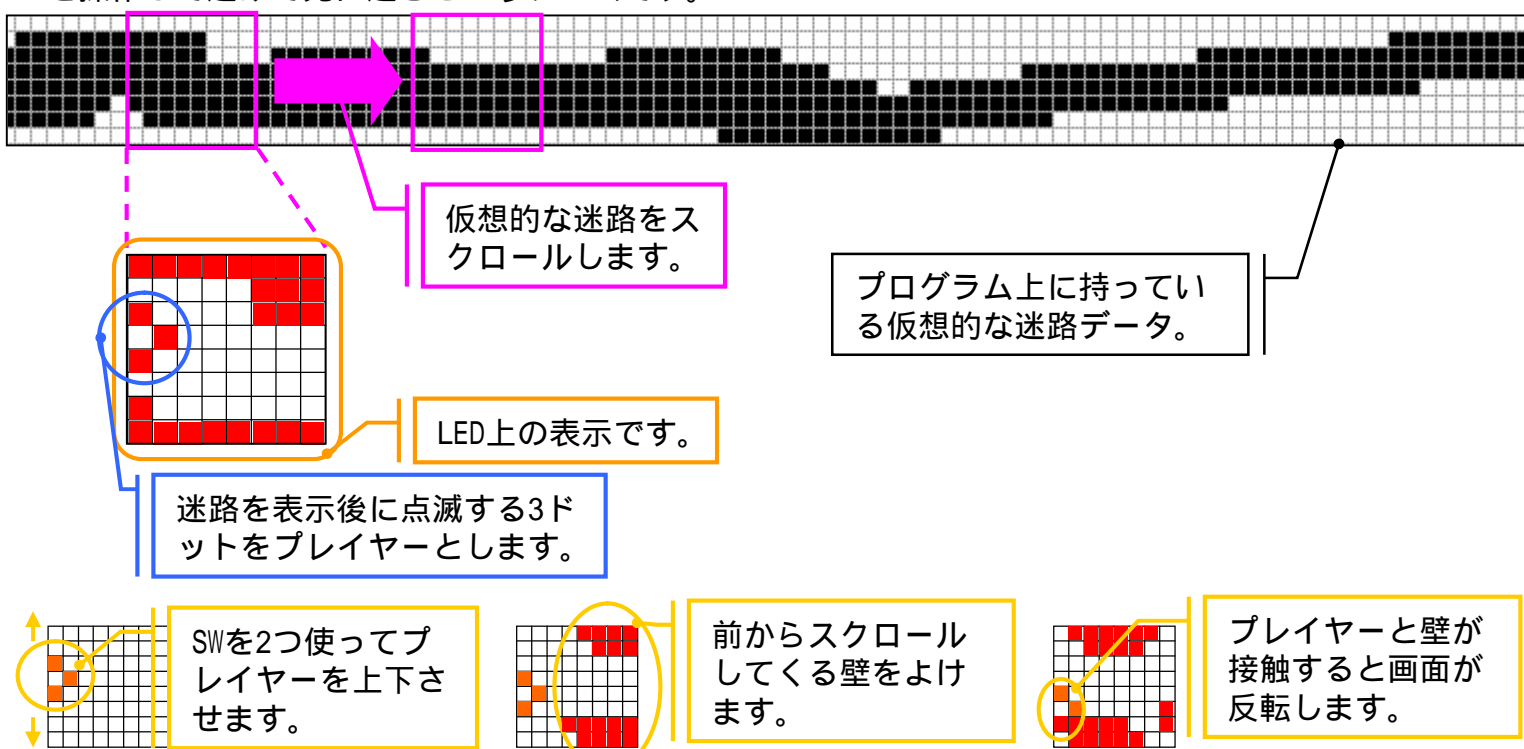
1. MINICUBE2へクロック・ボードを接続し、MINICUBE2のSWを設定する
3. 16pinターゲット・ケーブルをクロック・ボードへ接続し、ターゲット・ボードへ接続する
4. ターゲット・システム回路とターゲット・ボードを接続する
5. MINICUBE2とPC本体を接続する(MINICUBE2のLEDが白点滅します)
6. ターゲット・システム回路へ電源を供給する(MINICUBE2のLEDが白点灯します)

# ゲームを作る

この章ではターゲット・システム作成例2を紹介します。ターゲット作成例1(電光掲示板)の回路を流用し、SWを1つ追加しました。電光掲示板では文字をスクロールさせましたが、今度は文字ではなく迷路をスクロールさせます。ただそれだけではゲーム性に乏しいのでプレイヤーを操作して障害物をよけるゲームにします。ハードが同じでもソフトウェアによって、いろいろ応用が広がることを体験して下さい。

## ゲーム概要

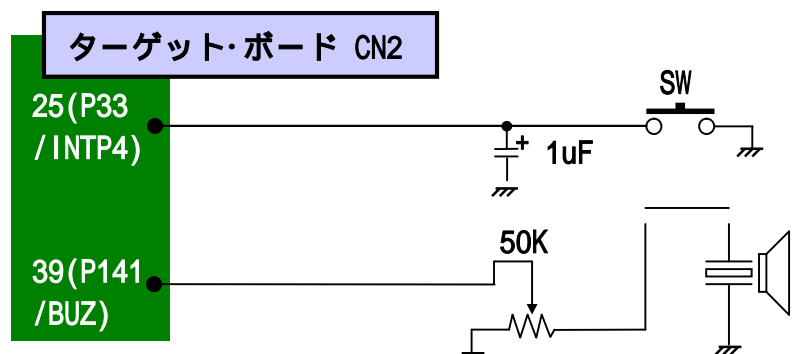
プログラム中に仮想的な迷路データを作ります。白い部分を壁としてドットマトリクスLEDへ表示します。プレイヤーをLEDの左側に3ドットで表現しています。このスクロールする迷路をプレイヤーを操作して避けて先に進むというゲームです。



壁のスクロール速度はA/Dに接続した半固定抵抗で調整可能です。これで、ゲームの難易度を調整します。プレイヤーと壁は同じ色(LED1色のみ)で表示しますが、プレイヤーは点滅するので壁と判別可能です。

## 追加回路

ターゲット作成例1の回路にSW/圧電スピーカを追加します。



## ワンポイント

**圧電スピーカについて**  
 圧電スピーカはヘッドホンでも代用できます。駆動電圧が少なくアンプ回路を必要としないスピーカを接続します。50K の可変抵抗がボリューム代わりです。

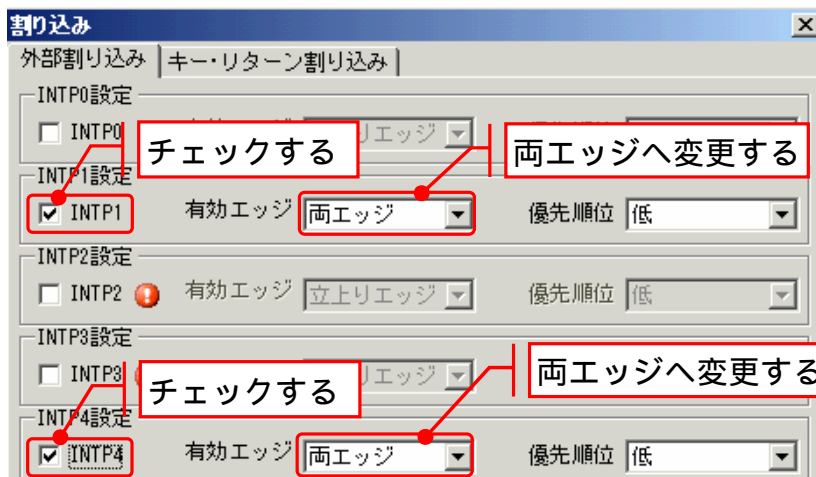
# プログラム設計

## a. Applilet2を使いプログラムを設計します。

[ターゲット作成例1]の[プログラム設計]を参照して同じ設定をして下さい。ここでは、異なる設定部分のみを説明します。

## b. [割り込み]を設定します。

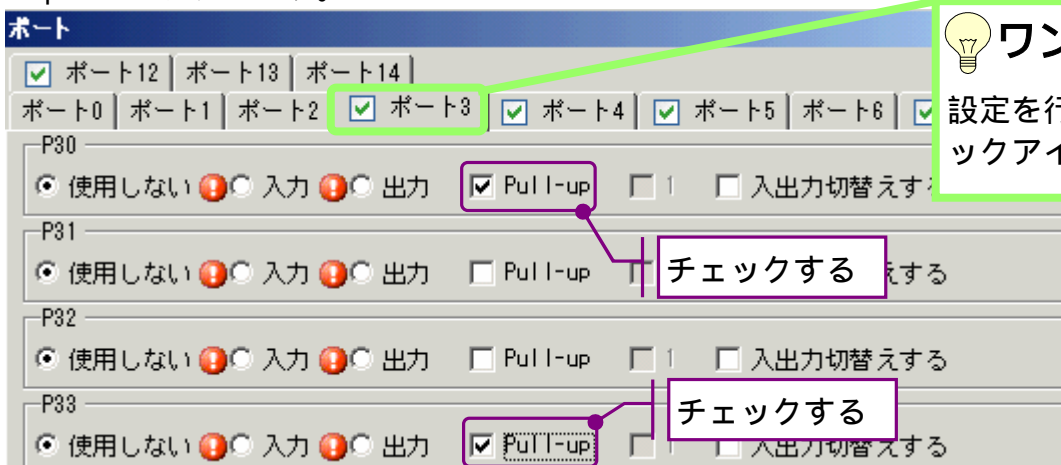
INTP1、INTP4の設定をします。



**ワンポイント**  
INTP1兼用端子のP30の設定、INTP4兼用端子のP33の設定も必要です。

## c. [ポート]を設定します。

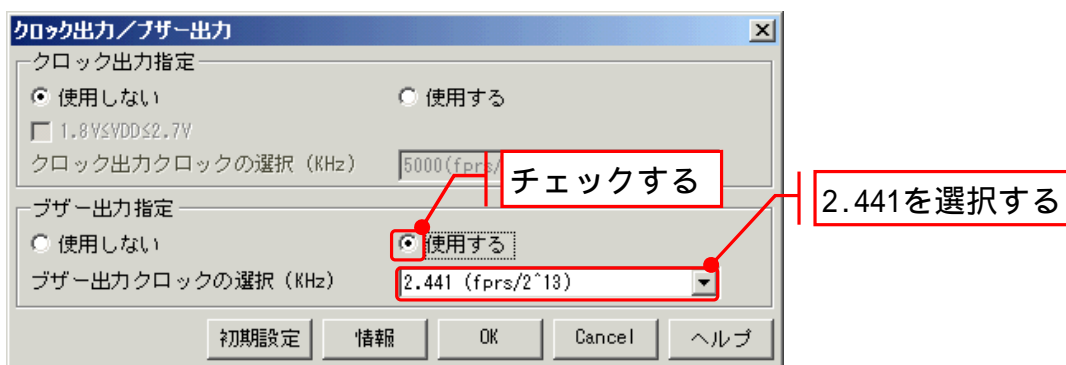
ポート4[P40~P47]、ポート5[P50~P52]以外に設定の追加があります。ポート3[P30、P33]のPull-upをチェックします。



**ワンポイント**  
設定を行っているポートにはチェックアイコン  が表示されます。

## d. [クロック出力/ブザー出力]を設定します。

ブザー出力指定で[使用する]にチェックします。



# プログラムリスト

## e. プログラムを編集して下さい。

下記に示す青字のコードを追加して下さい

### user\_define.h

#### リスト省略

```
#ifndef _MD_USER_DEF_
#define _MD_USER_DEF_
/*
*****
** Macro define
*****
*/
/* Start user code for definition. Do not edit comment generated here */

#define D_MLED_CNT 2 /* Matrix LED number */
#define D_MLED_ROW 8 /* Matrix LED row */
#define D_FONT_WIDTH 8 /* font width */

extern UINT g_timer0_counter;
extern UINT g_timer1_counter;
extern UINT g_bump_counter;
extern USHORT g_maze_scrollcnt;
extern USHORT g_maze_speed;
extern UCHAR g_matrix_ram[ D_MLED_CNT ][ D_MLED_ROW ];
extern UCHAR g_matrix_realram[ D_MLED_ROW ];
extern UCHAR g_intp1_sw;
extern UCHAR g_intp4_sw;
extern UCHAR g_plane_location;
extern BOOL g_plane_bump;

/* End user code for definition. Do not edit comment generated here */
#endif
```

### main.c(リスト1)

#### リスト省略

```
/*
*****
** Include files
*****
*/
#include "macrodriver.h"
#include "System.h"
#include "Port.h"
#include "Clkout.h"
#include "Int.h"
#include "Ad.h"
#include "Timer.h"
/* Start user code for include definition. Do not edit comment generated here */

#include "string.h"
#include "ctype.h"

void initialize_value( void );
void disp_putmaze( USHORT cnt_ );
void disp_move1dot(void );

/* End user code for include definition. Do not edit comment generated here */
#include "user_define.h"
/*
*****
** Global define
*****
*/
```

# プログラムリスト

## main.c(リスト2)

```

/* Start user code for global definition. Do not edit comment generated here */
const UCHAR g_maze_data[] =
{
    0x08,0xab,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
    0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xe0,0x00,0x00,
    0x00,0xff,0xff,0xff,0xff,0xff,0xff,0xf0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x03,0xc0,0x00,0x00,0x00,0x07,0xf8,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x20,0x20,0x00,0x00,0x02,0x02,0x00,0xff,
    0x00,0x03,0xff,0xf8,0x00,0xff,0xfc,0x00,0x0f,0xfc,0x0f,0xf0,0x00,0x00,0x1f,0xff,
    0xf8,0x00,0x7f,0xc0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,

    0x00,0x00,0x08,0x0e,0x06,0x3e,0x30,0x83,0xe0,0x1e,0x1f,0x80,0x70,
    0x0e,0x00,0x1f,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xf8,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x3f,0xff,0xfe,0x00,0x00,0x00,0xc0,0x00,0x06,0x00,0xc0,0x00,
    0x00,0x00,0x00,0x60,0x0c,0x00,0x03,0xe0,0x03,0xe0,0x00,0xe0,0x00,0x00,0x00,0x00,
    0x07,0xe0,0x00,0x00,0x3c,0x00,0xc0,0x00,0x20,0x20,0x10,0x00,0x02,0x02,0x00,0xff,
    0x00,0x03,0xff,0xf8,0x00,0x7f,0xfc,0x00,0x0f,0xfc,0x03,0xe0,0x03,0xff,0xff,0xfe,
    0x00,0x00,0x07,0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x40,0x00,0x00,0x00,0x00,0x00,0x00,0x20,0x00,0x00,0x40,0x00,0x00,0x41,

    0x00,0x00,0x00,0x00,0x00,0x0c,0x00,0x00,0x00,0x04,0x0f,0x00,0x20,
    0x04,0x00,0x1e,0x00,0x7f,0xf0,0x01,0xff,0xff,0xff,0x80,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xc0,0x01,0x86,0x00,0xc0,0x60,
    0x0c,0x03,0x00,0x60,0x0c,0x00,0x03,0xe0,0x07,0xe0,0x00,0xc0,0x07,0xe0,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x20,0x20,0x20,0x10,0x02,0x02,0x02,0x00,0x30,
    0x00,0x00,0xff,0xf0,0x00,0x1f,0xf8,0x00,0x07,0xfc,0x00,0x00,0xff,0xff,0xff,0xe0,
    0x00,0x00,0x00,0x00,0x00,0x0f,0x80,0x00,0xe0,0x00,0x40,0x00,0x00,0x40,0x04,
    0x00,0x00,0x04,0x00,0x00,0x10,0x08,0x00,0x04,0x10,0x02,0x04,0x00,0x88,0x00,0x00,

    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x06,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x3f,0xfc,0x00,0x00,0x00,0x00,0x03,0xff,0xff,
    0xf0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x80,0x00,0x01,0x80,0x00,0x00,0x60,
    0x0c,0x03,0x00,0x03,0x00,0x00,0x03,0xe0,0x03,0xc0,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x3f,0x80,0x00,0x00,0x20,0x22,0x21,0x11,0x02,0x00,0x02,0x00,0x00,
    0x00,0x00,0x1f,0xe1,0xfe,0x07,0xf8,0x00,0x03,0xf8,0x00,0x00,0x1f,0xf0,0x00,0x00,
    0x00,0x00,0x00,0x00,0x06,0x00,0x00,0x00,0x00,0x00,0x80,0x00,0x40,0x00,0x80,
    0x00,0x00,0x00,0x02,0x00,0x00,0x11,0x01,0x01,0x00,0x00,0x10,0x00,0x04,0x04,

    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x06,0x00,0x00,0x00,0x01,0xff,0xff,0xff,0xff,
    0xf8,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x80,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x06,0x03,0x00,0x07,0x80,0x00,0x00,0x00,0xe0,0x00,0x00,0x00,0x7f,
    0xe0,0x00,0x00,0x00,0x00,0x00,0x02,0x02,0x21,0x01,0x00,0x20,0x00,0x00,0x00,
    0x00,0x00,0x03,0xc1,0xff,0x01,0xf0,0x00,0x03,0xf0,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x00,0x02,0x00,0x00,
    0x00,0x02,0x00,0x04,0x00,0x01,0x00,0x00,0x40,0x00,0x08,0x00,0x00,0x40,0x00,

    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x08,0x02,
    0x00,0x04,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x1f,0xff,0xff,0xff,0xff,
    0xff,0xff,0xff,0xff,0x80,0x00,0x00,0x00,0x00,0x00,0x30,0x00,0x18,0x0c,0x03,
    0x00,0x30,0x06,0x03,0x00,0x07,0x80,0x00,0xc0,0x00,0xe0,0x01,0x00,0x07,0xe0,0x00,
    0x00,0x01,0xfc,0x00,0x00,0xfc,0x04,0x02,0x02,0x01,0x01,0x10,0x20,0x40,0x18,0x00,
    0x0f,0x80,0x00,0x01,0xff,0x00,0x00,0x1f,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0xf8,0x00,0x03,0xf0,0x00,0xc2,0x00,0x00,0x00,0x00,0x00,
    0x84,0x00,0x00,0x80,0x00,0x00,0x00,0x40,0x00,0x10,0x00,0x01,0x00,0x00,0x00,0x20,

    0x00,0x00,0x40,0x80,0xc0,0x61,0x8e,0x38,0x0f,0x00,0x00,0x1c,0x07,
    0x00,0x0e,0x00,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0xff,0xff,0xff,0xff,
    0xff,0xff,0xff,0xff,0xf0,0x1f,0x80,0x00,0x00,0x00,0x30,0x18,0x18,0x0c,0x03,
    0x00,0x30,0x60,0x00,0x03,0x07,0x80,0x00,0xe0,0x00,0xe0,0x01,0xc0,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x04,0x02,0x02,0x01,0x00,0x10,0x20,0x40,0x7f,0x00,
    0x0f,0xe0,0x00,0x03,0xff,0x00,0x00,0x7f,0x00,0x00,0x0f,0xc0,0x00,0x00,0x3f,
    0xff,0xfc,0x00,0x00,0xff,0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x08,0x00,
    0x00,0x00,0x10,0x00,0x10,0x00,0x00,0x01,0x02,0x00,0x40,0x40,0x02,0x08,0x00,0x00,

```

# プログラムリスト

## main.c(リスト3)

```

        0x12,0xaf,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xe0,0x00,0x7f,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xfc,0x00,0x00,0x18,0x00,0x00,0x03,
0x00,0x00,0x60,0x00,0x03,0x00,0x00,0x00,0xe0,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x02,0x02,0x01,0x00,0x00,0x00,0x40,0xff,0x80,
0x1f,0xe0,0x00,0x03,0xff,0x00,0x00,0xff,0x00,0x00,0x1f,0xe0,0x00,0x00,0xff,
0xff,0xff,0xfe,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x08,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x04,0x02
};

/* Interval timer counter */
UINT g_timer0_counter = 0;
UINT g_timer1_counter = 0;
UINT g_bump_counter;

/* control for maze */
USHORT g_maze_scrollcnt = 0;
USHORT g_maze_speed = 0;

/* INTP1, INTP4 status */
UCHAR g_intp1_sw;
UCHAR dummy;
UCHAR g_intp4_sw;

/* g_matrix_ram[0] is real vram. [1] is buffer vram. */
UCHAR g_matrix_ram[ D_MLED_CNT ][ D_MLED_ROW ];
UCHAR g_matrix_realam[ D_MLED_ROW ];
UCHAR g_plane_location;
BOOL g_plane_bump;

/* End user code for global definition. Do not edit comment generated here */

/*
** -----
** Abstract:
**          This function implements main function.
** Parameters:
**          None
** Returns:
**          None
** -----
*/
void main( void )
{
    /* Start user code. Do not edit comment generated here */

    /* TB-board LED off */
    P7.6 = 1;
    P7.7 = 1;

    initialize_value();

    /* start timer */
    TMO0_Start();
    TMO1_Start();

    BUZ_Stop();

    while (1) {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
/* Start adding user code. Do not edit comment generated here */

```

# プログラムリスト

## main.c(リスト4)

```
/* ----- */
void initialize_value( void )
{
    memset( g_matrix_ram, 0x00, sizeof( g_matrix_ram ) );
    g_timer0_counter = 0;
    g_timer1_counter = 0;
    g_plane_location = 2;
    g_plane_bump = FALSE;
    g_bump_counter = 3000;
}

/* ----- */
/* font_ is displayed to virtual vram. */
void disp_putmaze( USHORT cnt_ )
{
    int i;
    UCHAR fnt;

    for ( i=0; i<D_MLED_ROW; i++ )
    {
        fnt = g_maze_data[ i*125 + (cnt_/8) ];
        g_matrix_ram[ 1 ][ i ] = fnt;
    }
}

/* ----- */
/* 1 dot is scrolled. */
void disp_move1dot(void )
{
    int i;
    UCHAR vtmp, vtmp2;

    for ( i=0; i<D_MLED_ROW; i++ )
    {
        vtmp = ( g_matrix_ram[ 0 ][ i ] & 0x7f) << 1;
        vtmp2 = ( g_matrix_ram[ 1 ][ i ] & 0x80) >> 7;
        g_matrix_ram[ 0 ][ i ] = vtmp + vtmp2;
        vtmp2 = ( g_matrix_ram[ 1 ][ i ] & 0x7f) << 1;
        g_matrix_ram[ 1 ][ i ] = vtmp2;
    }
}

/* End user code adding. Do not edit comment generated here */
```

# プログラムリスト

## Ad\_user.c

### リスト省略

```
#pragma interrupt INTAD MD_INTAD
/*
*****
** Include files
*****
*/
#include "macrodriver.h"
#include "Ad.h"
/* Start user code for include definition. Do not edit comment generated here */
/* End user code for include definition. Do not edit comment generated here */
#include "user_define.h"

/*
*****
** Global define
*****
*/
/* Start user code for global definition. Do not edit comment generated here */
/* End user code for global definition. Do not edit comment generated here */

/*
** -----
** Abstract:
**          This function is INTAD interrupt service routine.
** Parameters:
**          None
** Returns:
**          None
** -----
*/
__interrupt void MD_INTAD( void )
{
    /* Start user code. Do not edit comment generated here */

    USHORT adval;

    AD_Stop();

    AD_Read( &adval );
    g_maze_speed = adval/2 + 1;
    if ( g_maze_speed > 500 )
    {
        g_maze_speed = adval;
    }

    /* End user code. Do not edit comment generated here */
}

/* Start adding user code. Do not edit comment generated here */
/* End user code adding. Do not edit comment generated here */
```



# プログラムリスト

## Int\_user.c

### リスト省略

```
#pragma interrupt INTP1 MD_INTP1
#pragma interrupt INTP4 MD_INTP4
/*
*****
** Include files
*****
*/
#include "macrodriver.h"
#include "Int.h"
/* Start user code for include definition. Do not edit comment generated here */
/* End user code for include definition. Do not edit comment generated here */
#include "user_define.h"

/*
*****
** Global define
*****
*/
/* Start user code for global definition. Do not edit comment generated here */
/* End user code for global definition. Do not edit comment generated here */

/*
** -----
** Abstract:
**          This function is INTP1 interrupt service routine.
** Parameters:
**          None
** Returns:
**          None
** -----
*/
__interrupt void MD_INTP1( void )
{
    /* Start user code. Do not edit comment generated here */

    g_intp1_sw = P3.0 ^ 1;

    /* End user code. Do not edit comment generated here */
}

/*
** -----
** Abstract:
**          This function is INTP4 interrupt service routine.
** Parameters:
**          None
** Returns:
**          None
** -----
*/
__interrupt void MD_INTP4( void )
{
    /* Start user code. Do not edit comment generated here */

    g_intp4_sw = P3.3 ^ 1;

    /* End user code. Do not edit comment generated here */
}

/* Start adding user code. Do not edit comment generated here */
/* End user code adding. Do not edit comment generated here */
```

# プログラムリスト

## Timer\_user.c(リスト1)

### リスト省略

```
#pragma interrupt INTTM000 MD_INTTM000
#pragma interrupt INTTM001 MD_INTTM001
/*
*****
** Include files
*****
*/
#include "macrodriver.h"
#include "Timer.h"
/* Start user code for include definition. Do not edit comment generated here */
#include "Ad.h"
#include "Clkout.h"
/* End user code for include definition. Do not edit comment generated here */
#include "user_define.h"

/*
*****
** Global define
*****
*/
/* Start user code for global definition. Do not edit comment generated here */
extern void initialize_value( void );
extern void disp_putmaze( USHORT cnt_ );
extern void disp_move1dot( void );
/* End user code for global definition. Do not edit comment generated here */

/*
** -----
** Abstract:      This function is INTTM000 interrupt service routine.
** Parameters:   None
** Returns:      None
** -----
*/
__interrupt void MD_INTTM000( void )
{
    /* Start user code. Do not edit comment generated here */
    UCHAR row, line, pdat;

    /* Matrix LED line disp */
    row = (UCHAR)(g_timer0_counter) & 7;

    /* display column */
    P4 = 0;
    if ( g_timer0_counter & 0x40 )
    {
        if ( row == g_plane_location )
        {
            pdat = 0x80;
        }
        else if ( row == ( g_plane_location + 1 ) )
        {
            pdat = 0x40;
        }
        else if ( row == ( g_plane_location + 2 ) )
        {
            pdat = 0x80;
        }
        else
        {
            pdat = 0;
        }
    }
}
```

# プログラムリスト

## Timer\_user.c(リスト2)

```
if ( g_matrix_realram[ row ] & pdat )
{ /* bump judgment */
  g_plane_bump = TRUE;
}
else
{
  g_matrix_realram[ row ] |= pdat;
}
}
line = g_matrix_realram[ row ];
P4 = line;

/* display row */
P5.0 = ( row & 1 );
P5.1 = (( row & 2 ) >> 1);
P5.2 = (( row & 4 ) >> 2);
g_timer0_counter++;

/* End user code. Do not edit comment generated here */
}

/*
**-----
** Abstract:
**           This function is INTTM001 interrupt service routine.
** Parameters:
**           None
** Returns:
**           None
**-----
*/
__interrupt void MD_INTTM001( void )
{
  /* Start user code. Do not edit comment generated here */
  UCHAR dat;
  UINT i;

  AD_Start();

  g_timer1_counter++;
  if ( g_plane_bump )
  {
    for ( i=0; i<D_MLED_ROW; i++ )
    {
      dat = g_matrix_ram[ 0 ][ i ];
      if ( g_timer1_counter & 0x40 )
      {
        dat = dat ^ 0xff;
        BUZ_Start();
        g_bump_counter--;
      }
      else
      {
        BUZ_Stop();
      }
      g_matrix_realram[ i ] = dat;
    }
    if ( g_bump_counter <= 0 )
    {
      BUZ_Stop();
      initialize_value();
    }
  }
  else
```

# プログラムリスト

## Timer\_user.c(リスト3)

```
{
  if ( g_timer1_counter > g_maze_speed )
  {
    if ( ( g_maze_scrollcnt % D_FONT_WIDTH ) == 0 )
    { /* next maze */
      disp_putmaze( g_maze_scrollcnt );
    }

    /* scroll */
    disp_move1dot();

    g_maze_scrollcnt++;
    if ( g_maze_scrollcnt >=1000 )
    {
      g_maze_scrollcnt = 0;
    }
    g_timer1_counter = 0;

    /* plane position */
    if ( g_intp4_sw )
    {
      if ( g_plane_location > 0 )
      {
        g_plane_location--;
      }
    }
    if ( g_intp1_sw )
    {
      if ( g_plane_location < 5 )
      {
        g_plane_location++;
      }
    }
  }

  for ( i=0; i<D_MLED_ROW; i++ )
  {
    g_matrix_realram[ i ] = g_matrix_ram[ 0 ][ i ];
  }
}

/* End user code. Do not edit comment generated here */

/* Start adding user code. Do not edit comment generated here */
/* End user code adding. Do not edit comment generated here */
```

# プログラム説明

f. プログラムで使用している関数について説明します。

## main.c

```
UINT g_timer0_counter = 0;
UINT g_timer1_counter = 0;
タイマ割り込み時に+1されるカウンタです

UINT g_bump_counter = 0;
衝突した時に点滅する時間です

UCHAR g_text_sw = 0;
表示するテキストを選択します

UINT g_maze_scrollcnt = 0;
表示されている壁が何ドット移動したかを示します

USHORT g_maze_speed = 0;
壁のスクロール速度を示します

UCHAR g_matrix_ram[ D_MLED_CNT ][ D_MLED_ROW ];
UCHAR g_matrix_realram[ D_MLED_ROW ];
マトリックスLEDに表示するデータです。8x16ドット分
ありますが実際に表示されるのは8x8ドットです

UCHAR g_plane_location;
プレイヤーの位置を示します

BOOL g_plane_bump;
プレイヤーと壁が衝突したかを示します

void initialize_value( void )
変数初期化を行う関数です

void disp_putmaze( USHORT cnt_ )
パラメータ cnt_ で指定された壁をマトリックスLEDの
非表示領域へ出力します

void disp_move1dot(void )
マトリックスLEDの表示を1ドット移動(スクロール)します
```

## Ad\_user.c

```
__interrupt void MD_INTAD( void )
A/D変換終了時に呼ばれる関数です。
A/D値を読み込み g_maze_speed へ反映させます
```

## Int\_user.c

```
__interrupt void MD_INTP1( void )
外部割り込みINTP1に呼ばれる関数です。
SWの値を g_intp1_sw へ反映します

__interrupt void MD_INTP4( void )
外部割り込みINTP4に呼ばれる関数です。
SWの値を g_intp4_sw へ反映します
```

## Timer\_user.c

```
__interrupt void MD_INTTM000()
1msec毎に呼ばれる関数です。下記の処理を行います
・プレイヤーの表示
・プレイヤーと壁の衝突判定
・マトリックスLEDにg_matrix_realram よりデータを読み込み1ライン表示

__interrupt void MD_INTTM001()
1msec毎に呼ばれる関数です。下記の処理を行います。
・A/D変換を開始します
・プレイヤーと壁が衝突していたら衝突処理します
・スクロールのチェックを行い、1画面分のスクロール
が終了したら disp_putmaze を呼び出し、指定された
壁をマトリックスLEDの非表示領域へ出力します。
そうでなければ disp_move1dot を呼び出し、1ドット
分スクロールします。
・プレイヤーの移動処理します
・マトリックスLEDの非表示領域より実際の表示領域
g_matrix_realram へデータ転送します
```

## user\_define.h

```
#define D_MLED_CNT 2 /* Matrix LED number */
マトリックスLEDの数を定義します。実際は1つですが仮
想的に2つ持っていることにしています。

#define D_MLED_ROW 8 /* Matrix LED row */
マトリックスLEDのROWの数

#define D_FONT_WIDTH 8 /* font width */
1文字のフォントの幅。これは壁データになります
```

# 迷路のデータ作成方法

g. プログラム内で定義している迷路データについて説明します。

## const UCHAR g\_maze\_data[] データ作成方法

main.cのconst UCHAR g\_maze\_data[] ですが簡単に作成できる方法があります。Windowsのペイントツールを使います。まず画像を新規作成します。

- ・大きさを1000x8
- ・色は黒
- ・ファイルの種類はモノクロビットマップ

次にメニューより[表示(V)] [拡大(Z)] [拡大率の指定(U)]で800%を指定します。また、[表示(V)] [拡大(Z)] [グリッドを表示(G)]を指定します。画面は下図のようになります



白を描画すると、それが壁のデータになります。後は保存したbmpファイル(バイナリファイル)をソースへ変換して下さい。bmpファイルのフォーマット説明します。

最初の59バイトはbmpヘッダ	} x 8
3バイト(ヘッダ)+125バイト(1000ドット分)が	
1ライン分のデータです。	

1ライン分データの125バイト(1000ドット)を変換すれば実データになります。ラインデータは逆にあります画面最下ラインから上に向かってのデータになっていますので、ソースへ変換後順序を入れ替えて下さい。

# 最後に

ターゲット・システム作成例は以上です。最後の作成例はゲーム性を持たせホビーの要素を取り入れました。

昨今の電機製品の殆どにマイコンが使われています。特に子供向けのゲームなど1000円前後で買えるものは1チップマイコンにスピーカー、LEDをつけただけです。それが音声出力、各種制御まで行っています。これだけでもマイコンの可能性がわかるのではないのでしょうか。このチュートリアル・ガイドがマイコンを始めるきっかけになれば幸いです。

この作成例を元に電光掲示板の大きさを大きくして表示可能なドット数を増やしたり、赤外線送受信を加えて対戦方式のゲームを作成など、これを応用して是非チャレンジして下さい。

