

お客様各位

ZUD-CD-06-0038-1

1/69

2006年12月28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



V850ES/KG2ターゲット・ボード

QB-V850ESKG2-TB

クイック・スタート・ガイド

第2版

ごあいさつ

QB-V850ESKG2-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 MINICUBE2を使用して、マイコンを動作させるためのターゲット・ボードです。

クイック・スタート・ガイドでは、開発環境のご紹介と、使い方をサンプル・プログラムを用いて説明しています。本製品をご使用になる前に、ご一読ください。

本製品に関する最新情報、必要な開発ツール、およびサンプル・プログラム(順次拡充予定)は、弊社WEBページにて提供しています。

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

【本クイック・スタート・ガイドの構成について】

本ガイドは4つの章から構成されています。

はじめに

Page 2~4

ターゲット・ボードの特徴など本ガイドをお使いいただく際に必要な基本的な事柄について説明します。

準備

Page 5~10

ターゲット・ボードの仕様やシステム構成図、開発ツールのインストールについて説明します。

体験

Page 11~48

簡単なアプリケーション・プログラムの作成を通して、開発ツールの使い方について説明します。

応用

Page 49~69

ターゲット・システムの作成例を2種類ご紹介します。

付録

ターゲット・ボード回路図

ターゲット・ボードについて (その1)

V850ES/KG2ターゲット・ボード(QB-V850ESKG2-TB)の特徴

V850ES/KG2(μPD70F3732GC)搭載

システム・クロック20MHz(5MHz発振子を搭載)で高速動作可能

フラッシュメモリ:256KB、RAM:16KBを内蔵

最大で84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(S100, S000, SCK00, PCMO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

動作電圧2.7V~5.5V、電池2本で動作可能

V850ES/KG2ターゲット・ボード(QB-V850ESKG2-TB)のハードウェア仕様

CPU μPD70F3732GC	メイン・クロック 動作周波数	最大20MHz (ボード上に発振子5MHz搭載)
	サブクロック 周波数	32.768KHz (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) 50pinソケットx2(パッドのみ)	
	FP1: 16pinコネクタ(MINICUBE2接続用)	
	Power LED: 赤x1(LED3)	
	評価用LED: 黄x2(LED1はPCM3, LED2はPCM2へ接続)	
	評価用SW: SW1(INTP0へ接続)	
	Y1: メイン・クロック用5MHz発振子(X1, X2へ接続) Y2: サブクロック用32.768KHz発振子(XT1, XT2へ接続)	
動作電圧	2.7V~5.5V	



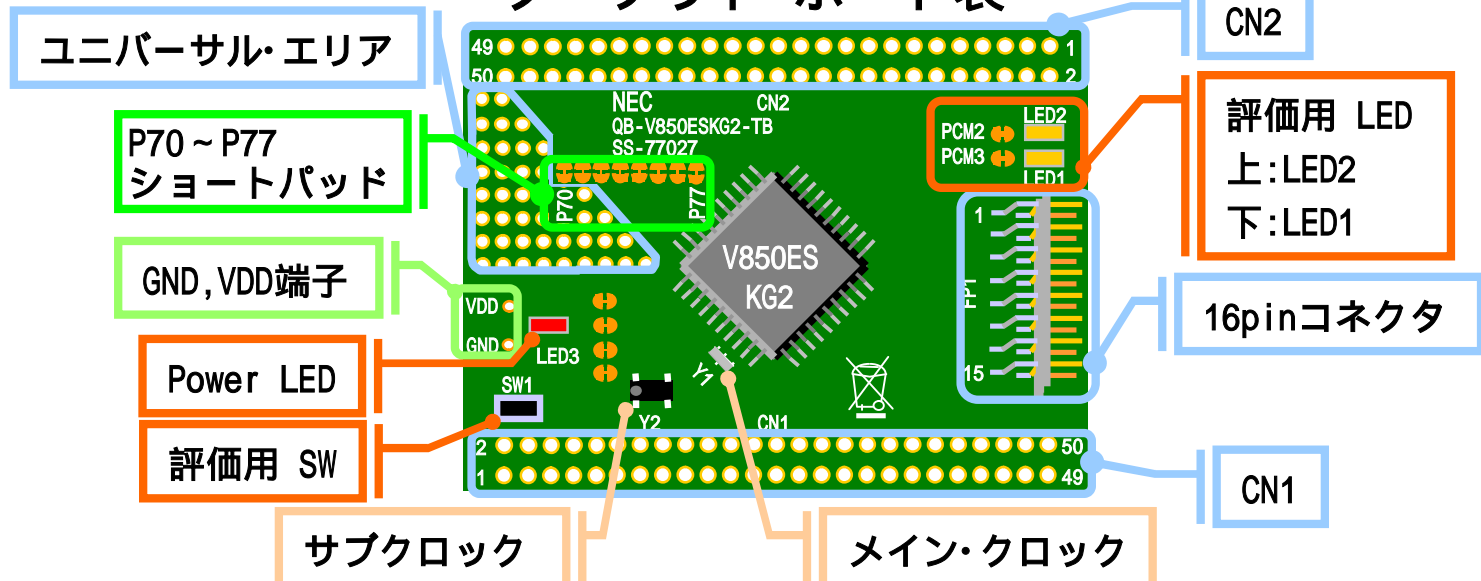
ワンポイント

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デバイスもあります。また、性能も数MIPS~数百MIPSまで揃っています。

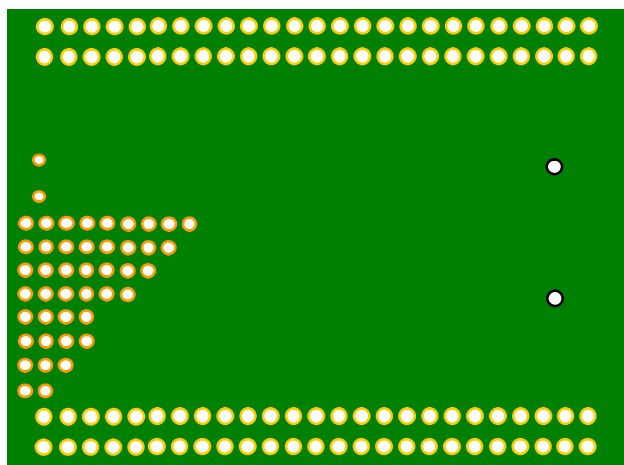
ターゲット・ボードについて (その2)

ターゲット・ボード表





- CN1/CN2: デバイス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PCM3がLOWで黄色に発光します
- 評価用LED2: ポートPCM2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
- Y1(メイン・クロック): 5MHz発振子を搭載しています
- Y2(サブ・クロック): 32.768KHz発振子を搭載しています
- ユニバーサル・エリア: ユーザが部品を載せられるエリアです
- P70~P77ショートパッド: P70~P77(AN10~AN17)はプルダウン抵抗に接続されています。必要に応じてショートパッドをパターンカットしてください。

ターゲット・ボード裏



・基板上的パターン 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。 

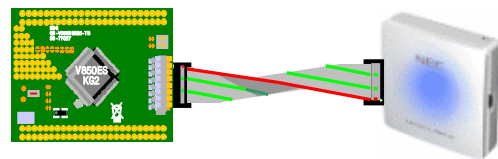
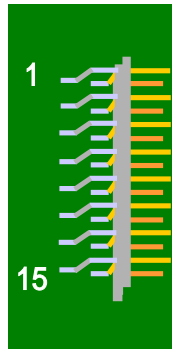
再度接続させたい場合は半田ショートしてください。 

PCM2, PCM3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。

ターゲット・ボード仕様 (その1)

16pinヘッダピンアサイン(デバッグ時に扱う信号)

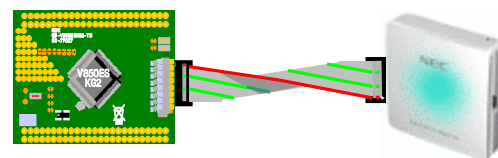
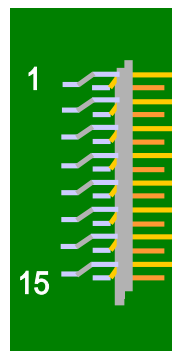
ピン番号	信号	ピン番号	信号
1	GND	2	RESET_OUT
3	S000	4	VDD
5	S100	6	R.F.U.
7	SCK00	8	PCMO
9	R.F.U.	10	R.F.U.
11	-----	12	FLMD1
13	R.F.U.	14	FLMD0
15	RESET_IN	16	R.F.U.



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	S000	4	VDD
5	S100	6	R.F.U.
7	SCK00	8	PCMO
9	R.F.U.	10	R.F.U.
11	-----	12	FLMD1
13	R.F.U.	14	FLMD0
15	R.F.U.	16	R.F.U.



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

ターゲット・ボード仕様 (その2)

CN1(詳細は付録の回路図を参照してください)

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	AVREF0		2	GND	
3	P10/AN00		4	P11/AN01	
5	AVREF1		6	P00/TOH0	
7	P01/TOH1		8	FLMD0	16pinコネクタ14へ接続
9	VDD		10	NC	
11	GND		12	NC	
13	NC		14	T_RESET	16pinコネクタ15へ接続
15	NC		16	NC	
17	P02/NMI		18	P03/INTP0	SW1にも接続
19	P04/INTP1		20	P05/INTP2	
21	P06/INTP3		22	P40/S100/RXD2	16pinコネクタ5へ接続
23	P41/S000/TXD2	16pinコネクタ3へ接続	24	P42/ $\overline{\text{SCK00}}$	16pinコネクタ7へ接続
25	P30/TXD0/T002		26	P31/INTP7/RXD0/T003	
27	P32/ASCK0/T001 /ADTRG		28	P33/TI000/T000 /TIP00/TOP00	
29	P34/TI001/T000 /TIP01/TOP01		30	P35/TI010/T001	
31	P36		32	P37	
33	GND		34	EVDD	
35	P38/SDA00		36	P39/SCL00	
37	P50/TI011/KR0/RTP00		38	P51/TI50/KR1/RTP01	
39	P52/T050/KR2/RTP02		40	P53/SIA0/KR3/RTP03	
41	P54/SOA0/KR4/RTP04		42	P55/ $\overline{\text{SCKA0}}$ /KR5/RTP05	
43	P90/TXD1/KR6/A0		44	P91/RXD1/KR7/A1	
45	P92/A2/TI020/T002		46	P93/A3/TI021	
47	P94/A4/TI030/T003		48	P95/A5/TI031	
49	P96/A6/TI51/T051		50	P97/A7/SI01	

ターゲット・ボード仕様 (その3)

CN2(詳細は付録の回路図を参照してください)

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	P98/A8/S001		2	P99/A9/ $\overline{\text{SCK01}}$	
3	P910/A10/SIA1		4	P911/A11/SOA1	
5	P912/A12/ $\overline{\text{SCKA1}}$		6	P913/A13/INTP4	
7	P914/A14/INTP5		8	P915/A15/INTP6	
9	PCS0/ $\overline{\text{CS0}}$		10	PCS1/ $\overline{\text{CS1}}$	
11	PCM0/ $\overline{\text{WAIT}}$	16pinコネクタ8へ接続	12	PCM1/CLKOUT	
13	PCM2/ $\overline{\text{HLDAK}}$	LED2へ接続	14	PCM3/ $\overline{\text{HLDRQ}}$	LED1へ接続
15	PCT0/ $\overline{\text{WRO}}$		16	PCT1/ $\overline{\text{WR1}}$	
17	PCT4/ $\overline{\text{RD}}$		18	PCT6/ASTB	
19	GND		20	BVDD	
21	PDL0/AD0		22	PDL1/AD1	
23	PDL2/AD2		24	PDL3/AD3	
25	PDL4/AD4		26	PDL5/AD5/FLMD1	16pinコネクタ12へ接続
27	PDL6/AD6		28	PDL7/AD7	
29	PDL8/AD8		30	PDL9/AD9	
31	PDL10/AD10		32	PDL11/AD11	
33	PDL12/AD12		34	PDL13/AD13	
35	PDL14/AD14		36	PDL15/AD15	
37	PDH0/A16		38	PDH1/A17	
39	PDH2/A18		40	PDH3/A19	
41	PDH4/A20		42	PDH5/A21	
43	P77/ANI7	1M でプルダウン	44	P76/ANI6	1M でプルダウン
45	P75/ANI5	1M でプルダウン	46	P74/ANI4	1M でプルダウン
47	P73/ANI3	1M でプルダウン	48	P72/ANI2	1M でプルダウン
49	P71/ANI1	1M でプルダウン	50	P70/ANI0	1M でプルダウン

システム構成図1 プログラム開発時 (MINICUBE2使用時)

プログラム開発時 (MINICUBE2使用時)

MINICUBE2とターゲット・ボードを組み合わせた構成です。赤点線で囲んだ部分は、本製品です。

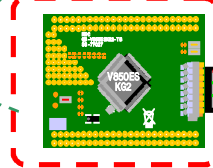
V850ES/Kx2シリーズ用開発ツール

- コンパイラ (CA850)
- PM+
- デバッガ (ID850QB)
- デバイス・ファイル (DF703734)
- デバイス・ドライバ・コンフィギュレータ (Applilet)

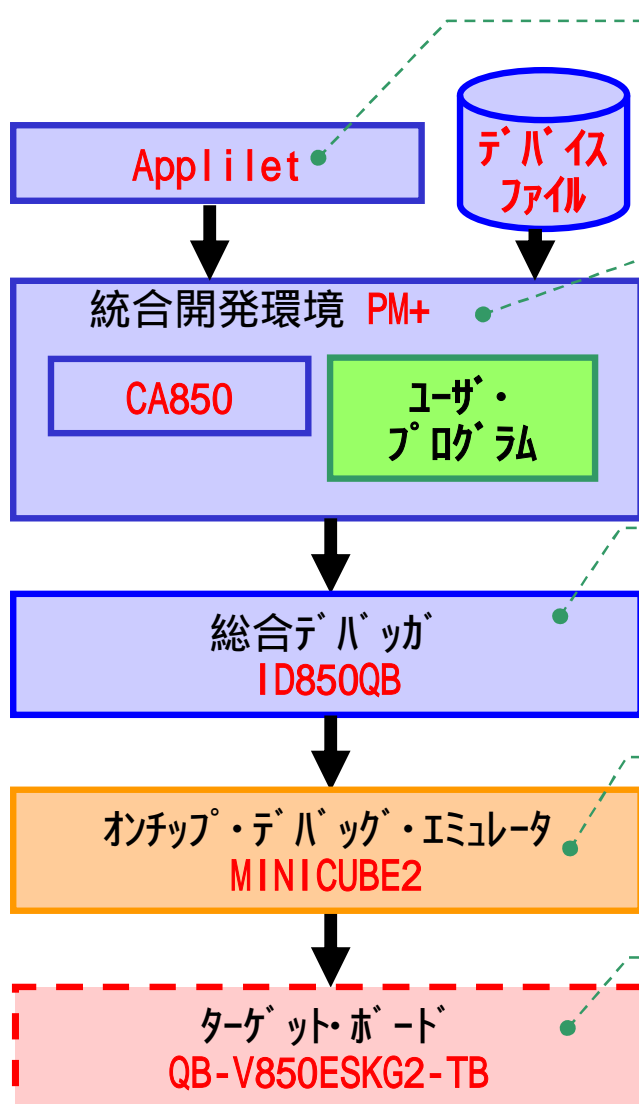
16pinターゲット・ケーブル
ターゲット・ボード：
V850ES/KG2を搭載

ホストコンピュータ

ツールのダウンロード
/インストール



MINICUBE2



Applilet : 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を自動生成します。

PM+ : ソースコードのビルド、エディタやシミュレータ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たためのツールです。

ID850QB : マイコンの動作をオンチップ・デバッグします。実機を使ってプログラム動作の確認ができます。

MINICUBE2 (QB-MINI2) : プログラミング機能付きオンチップ・デバッグ・エミュレータです。

QB-V850ESKG2-TB : 本製品です。V850ES/KG2を搭載しています。

システム構成図2 プログラム書き込み時

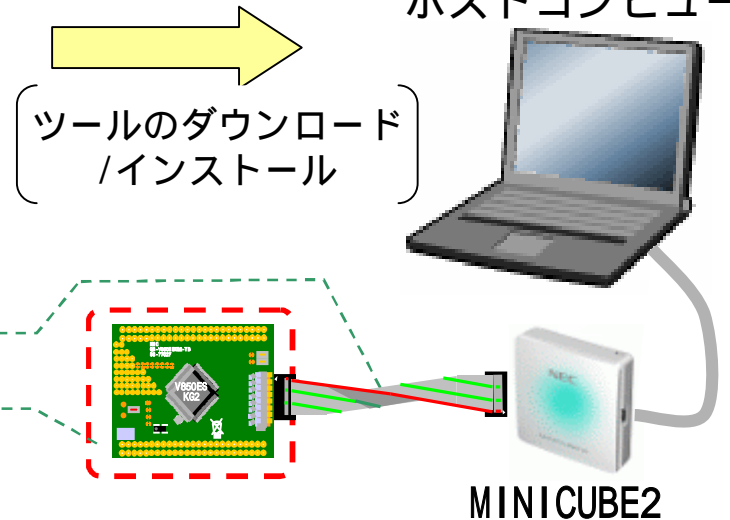
プログラム書き込み時

MINICUBE2とテストボードを組み合わせた構成です。点線で囲んだ部分は、本製品です。

V850ES/Kx2シリーズ用開発ツール

- プログラミング GUI (QB-Programmer)
- V850ES/Kx2用パラメータファイル (PRM70F3734)

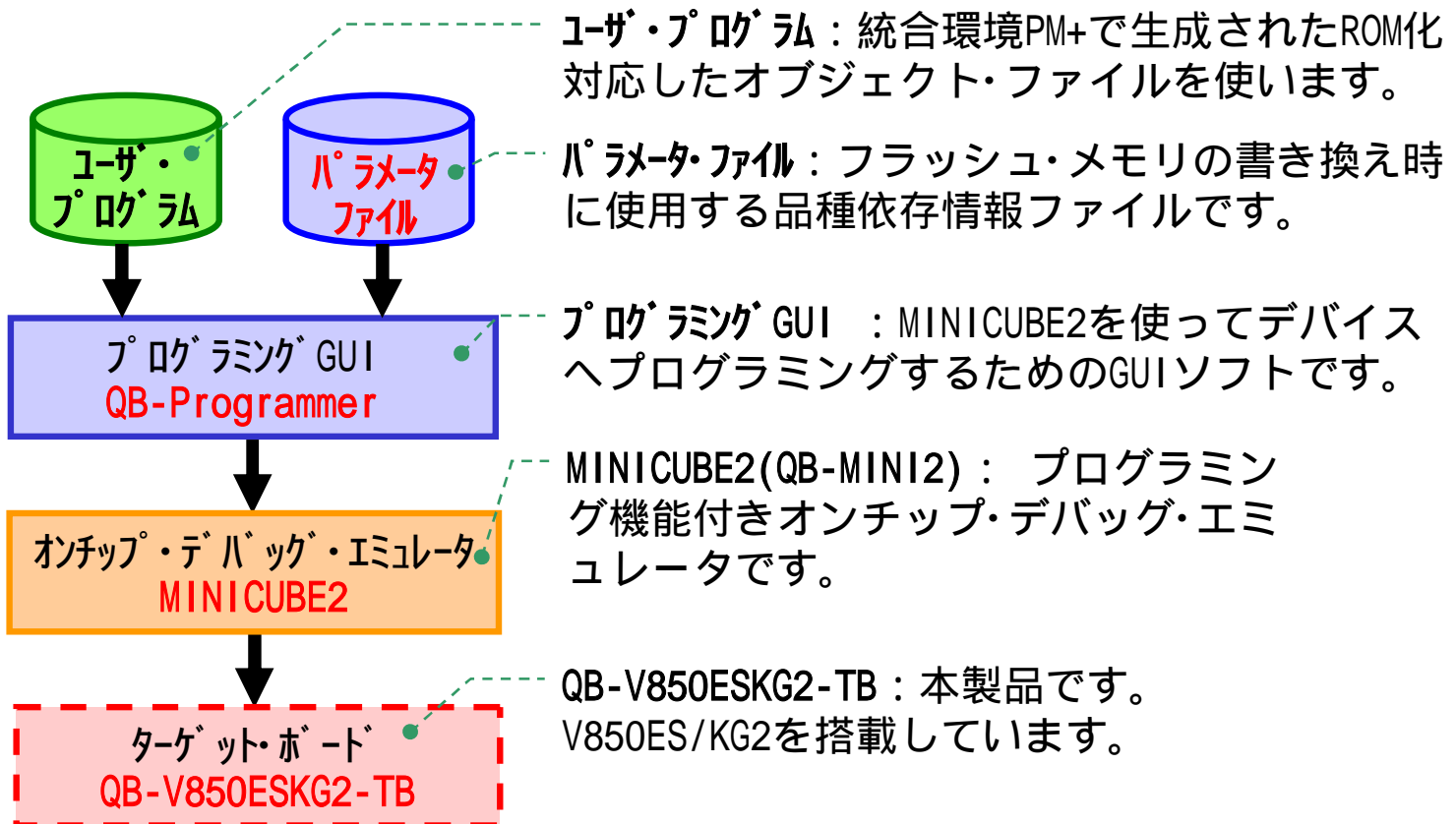
ホストコンピュータ



16pinターゲットケーブル

ターゲットボード：
V850ES/KG2を搭載

MINICUBE2



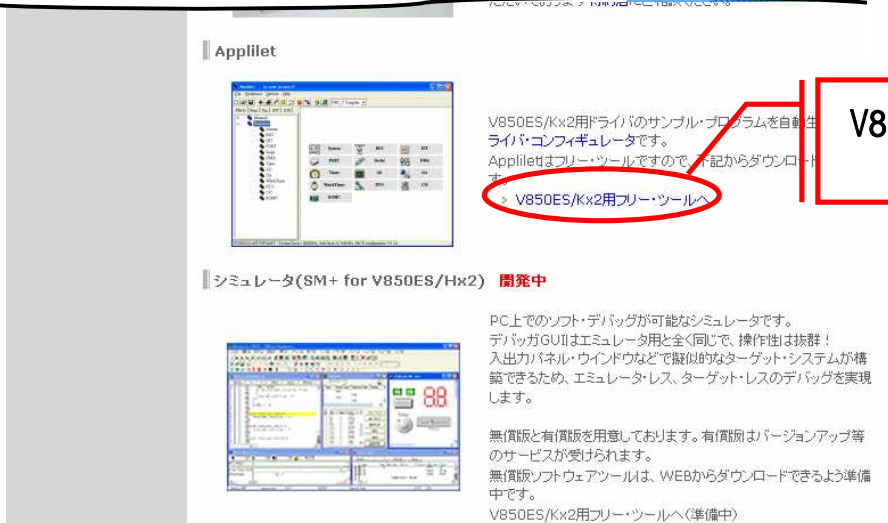
ワンポイント

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。

開発ツールのダウンロード

本製品を使うために必要な開発ツールは、弊社WEBページにて提供しています。
<http://www.necel.com/micro/ja/promotion/v850eskx2/>
<http://www.necel.com/micro/ja/development/asia/applilet/>
<http://www.necel.com/micro/ja/development/asia/minicube2/minicube2.html>



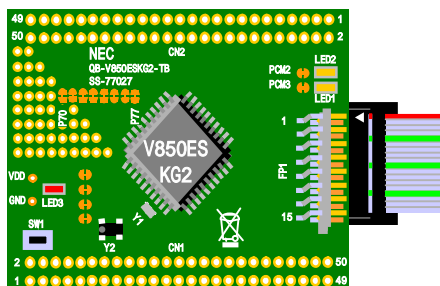
“V850ES/Kx2シリーズ用開発ツール”のインストール方法：

- **プログラミングGUI (QB-Programmer)、V850ES/Kx2用Applilet**
自己解凍形式のファイルを実行すると、フォルダが作成されます。
setup.exeを実行してインストールを行ってください。Appliletのバージョンは1.70以上を使用してください。
- **CA850**
自己解凍形式のファイルを実行すると、自動的にインストールが始まります。
画面の指示に従ってインストールを行ってください。
- **V850ES/Kx2用デバイス・ファイル**
専用のインストーラでインストールします。解凍したフォルダにあるユーザーズ・マニュアルを参照して、インストールを行ってください。
- **V850ES/Kx2用パラメータ・ファイル**
任意のフォルダに解凍してください。

マイコン開発

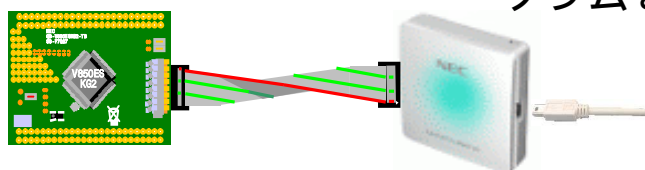
この章ではMINICUBE2、ターゲット・ボード、開発ツールを使用して、プログラムの作成から動作確認、デバッグ方法、マイコンへ書き込むまでの一連の開発手順をシステム構成別について説明します。

1. ターゲット・ボード体験編



MINICUBE2、ターゲット・ボード、無償ダウンロードツールを使用します。MINICUBE2を使って実際にマイコンを動作させます。また、統合デバッガID850QBの基本的な使い方を学びます。

2. マイコン・プログラミング体験編



1. ターゲット・ボード体験編で作成したプログラムをデバイスへ書き込みます。

💡 ワンポイント

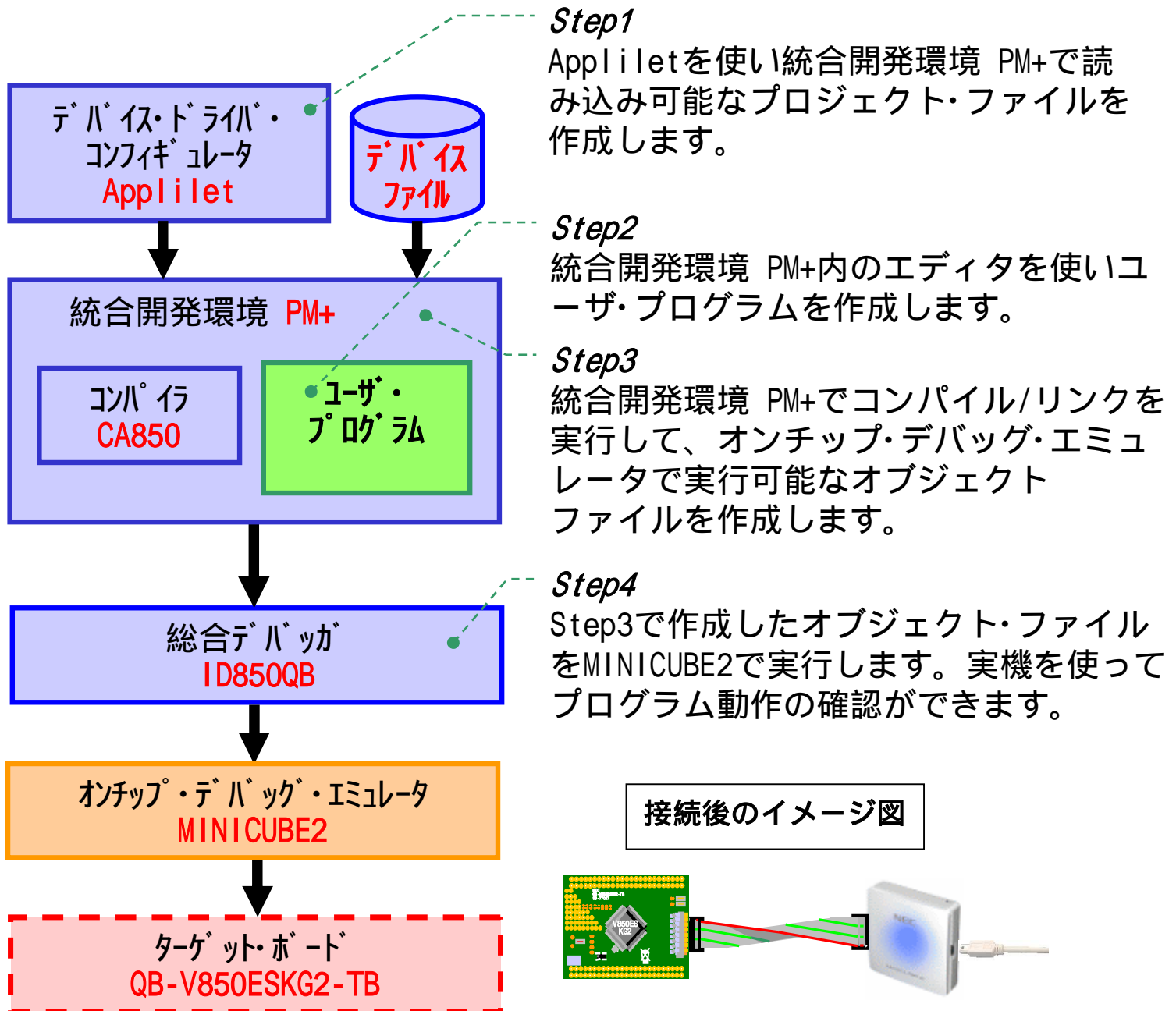
サンプル・プログラムについて

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を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。ターゲット・ボード上にはSW、LEDが搭載されているので作成したプログラムが実際に試せます。



ターゲット・ボード体験編

ターゲット・ボード体験編では実際にターゲット・ボードをMINICUBE2へ接続して使ってみるまでの手順を説明します。



次ページより Step1 ~ Step4 の詳細を説明します。

ターゲット・ボード体験編 (Step1-1)

Step1 Applilet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. Appliletを起動します。

[スタート] [プログラム(P)] [NEC Electronics Tools]
[Applilet for V850ESKX2] [Vx.xx] [Applilet for V850ESKX2 Vx.xx]

b. Applilet設定用ファイルを新規に作成します。

メニュー・バーの[ファイル(F)] [新規作成(N)...]を選択します。
『新規プロジェクト』ダイアログで、<プロジェクト名>と<チップ・シリーズ>を設定してください。



<プロジェクト名>

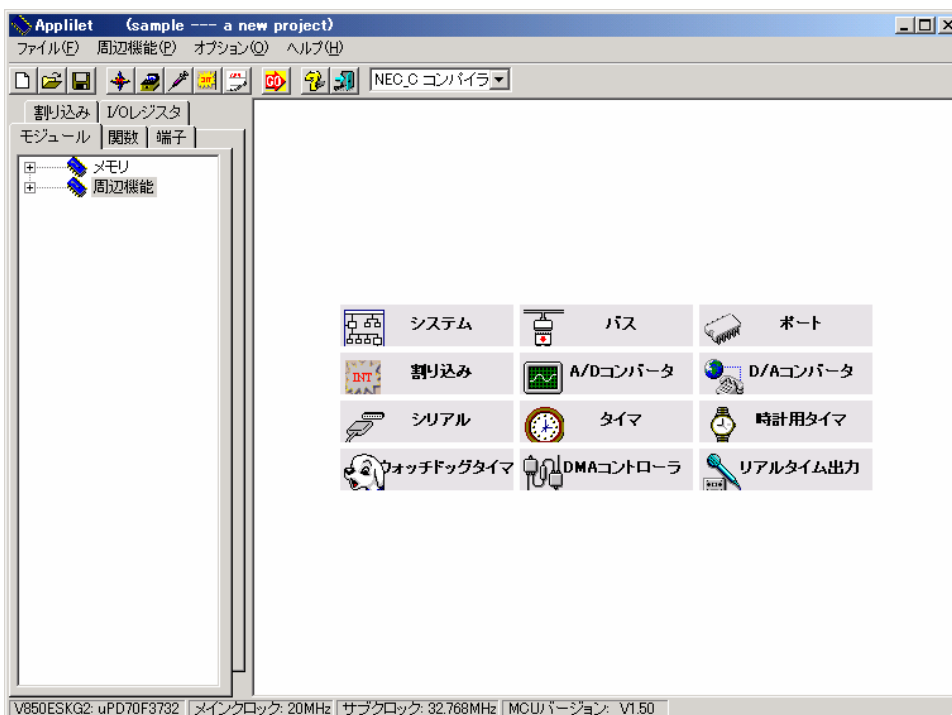
mdt sample に変更

<チップ・シリーズ>

シリーズ名： V850ESKG2 を選択

チップ名： uPD70F3732 を選択

OKをクリックすると周辺機器メニューが表示されます



ターゲット・ボード体験編 (Step1-2)

c. サンプル・プログラムで使用する周辺機器を設定します。(その1)

システム設定

[基本設定] タブ

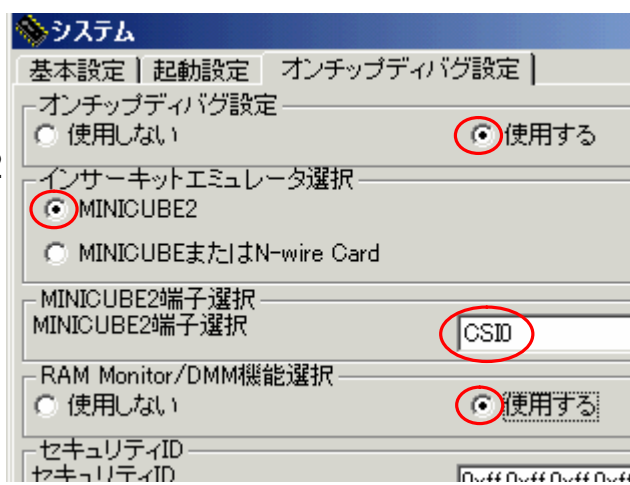
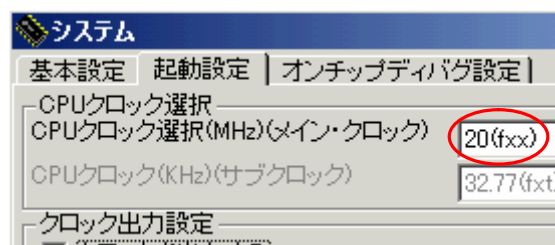
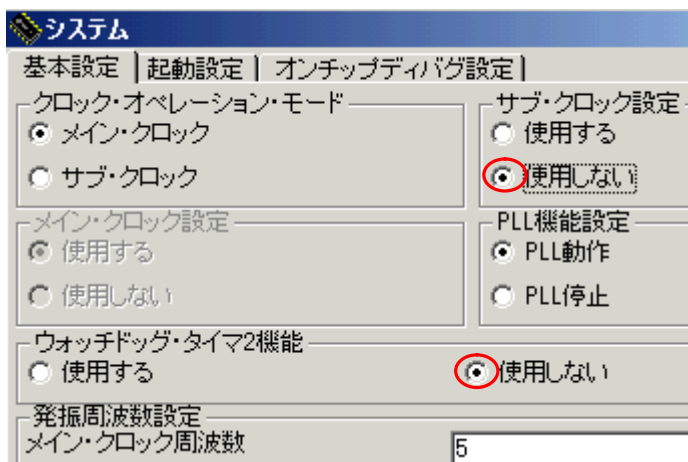
サブ・クロック設定エリアの
” 使用しない ” にチェック。
ウォッチドッグ・タイマ2機能エリアの
” 使用しない ” にチェック。

[起動設定] タブ

CPUクロック選択(MHz)メイン・クロック
” 20(fxx) ” を選択。

[オンチップディバグ設定] タブ

オンチップディバグ設定の ” 使用する ”
にチェック。インサーキットエミュレータ
選択は ” MINICUBE2 ” にチェック。MINICUBE2
端子選択は ” CS10 ”。RAM Monitor/DMM機能
選択は ” 使用する ” にチェック。



ワンポイント

システム設定

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。

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デバイス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。

ターゲット・ボード体験編 (Step1-3)

d. サンプル・プログラムで使用する周辺機器を設定します。(その2)

割り込み設定

外部割り込み設定 | 乗除算器の割り込み設定 | キー割り込み設定

NMI 設定
 NMI 有効エッジ: 立下りエッジ

INTPO 設定
 INTPO 有効エッジ: 立上りエッジ

[外部割り込み設定] タブ
 “INTPO許可” にチェック
 “有効エッジ” を立ち上がりエッジへ変更。

ポート設定

[ポート0] タブ

P03はINTPOと兼用端子です。そのため先に割り込み設定を行うとP03の「入力、出力」のチェックBOXは使用不可のアイコン になっています。Pull-upにチェックします。

[ポートCM] タブ

PCM2、PCM3の「出力、1」にチェック。LED1、LED2はそれぞれPCM3、PCM2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1に設定します。

ポート

ポート9-2 | ポートCM | ポートCS | ポートCT | ポートDH | ポートD
 ポート0 | ポート1 | ポート3-1 | ポート3-2 | ポート4 | ポート5 | ポー

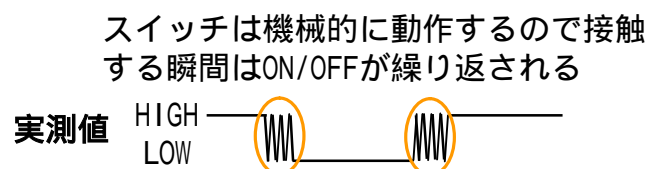
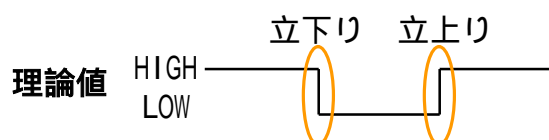
P00	<input checked="" type="radio"/> 使用しない	<input type="radio"/> 入力	<input type="radio"/> 出力	<input type="checkbox"/> Pull-up
P01	<input checked="" type="radio"/> 使用しない	<input type="radio"/> 入力	<input type="radio"/> 出力	<input type="checkbox"/> Pull-up
P02	<input checked="" type="radio"/> 使用しない	<input type="radio"/> 入力	<input type="radio"/> 出力	<input type="checkbox"/> Pull-up
P03	<input checked="" type="radio"/> 使用しない	<input type="radio"/> ? 入力	<input type="radio"/> ? 出力	<input checked="" type="checkbox"/> Pull-up

PCM2	<input type="radio"/> 使用しない	<input type="radio"/> 入力	<input checked="" type="radio"/> 出力	<input type="checkbox"/> Pull-up	<input checked="" type="checkbox"/> 1
PCM3	<input type="radio"/> 使用しない	<input type="radio"/> 入力	<input checked="" type="radio"/> 出力	<input type="checkbox"/> Pull-up	<input checked="" type="checkbox"/> 1

ワンポイント

有効エッジについて

外部割り込み設定で有効エッジを立上りエッジに変更しました。立上りエッジとは信号が0から1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を押した時が立下り、押してからSWを離れた時が立上りになります。ソフトウェア・シミュレータでは問題になりませんが実機ですとチャタリングという問題があります。解決方法についてはターゲットボード体験編を参照して下さい。



ターゲット・ボード体験編 (Step1-4)

e. サンプル・プログラムで使用する周辺機器を設定します。(その3)

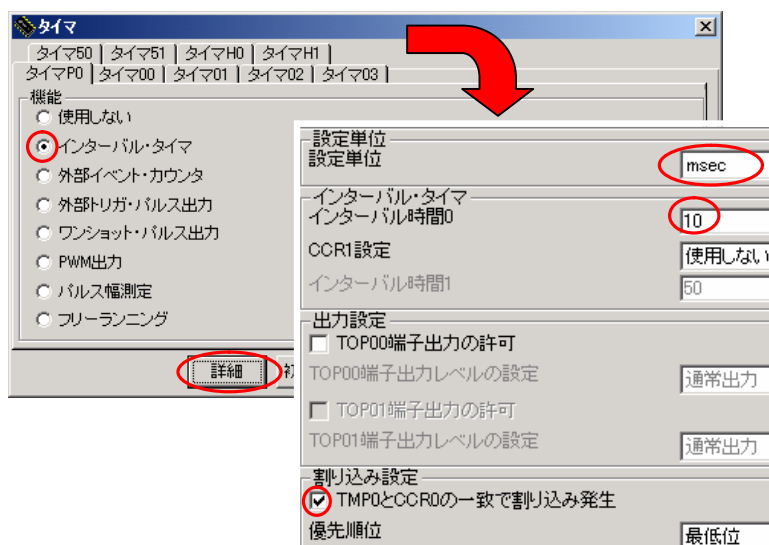


設定を行うと表示が変化します。
アイコン 黄色
タイトル 青色

タイマ設定

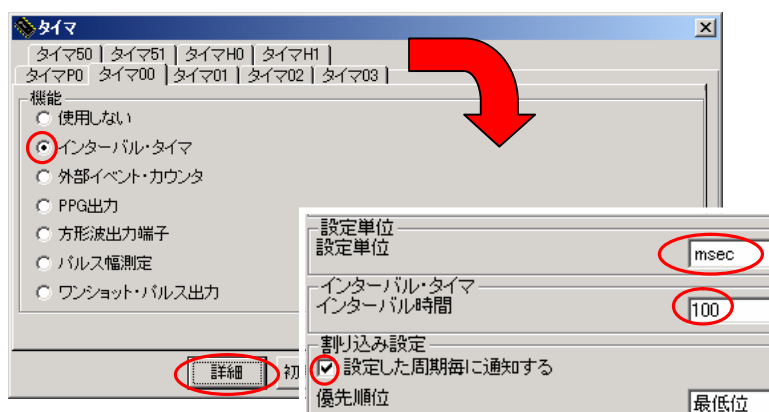
[タイマP0]タブ

タイマP0機能エリアのインターバル・タイマにチェック。詳細をクリックし次設定へ。設定単位を "msec" へ変更し、インターバル時間を "10" とします。割り込み設定で "TMP0とCCR0の一致で割り込み発生" にチェックします。



[タイマ00]タブ

タイマP0と同様にタイマ00機能エリアのインターバル・タイマにチェック。詳細をクリックし次設定へ。設定単位を "msec" へ変更し、インターバル時間を "100" とします。"設定した周期毎に通知する" にチェックします。



ワンポイント

タイマ設定

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です)、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。

ターゲット・ボード体験編 (Step1-5)

f. ソースコードを自動生成します。

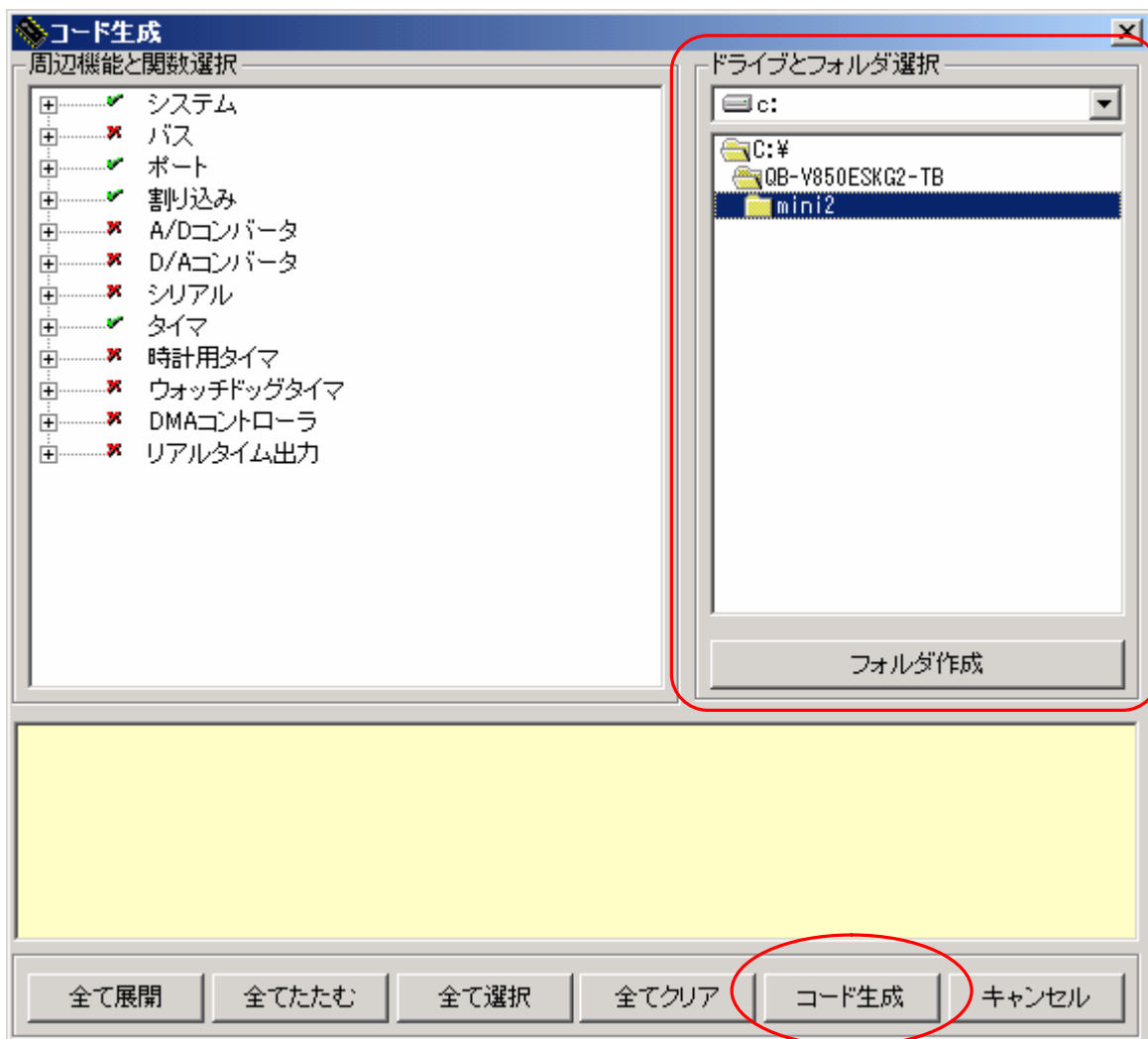
f-1) コンパイラ(CA850)のパスを設定します。

コンパイラを C:\Program Files\NEC Electronics Tools\CA850 以外の場所にインストールしている場合は、メニュー・バーの [オプション(O)] [コンパイラ選択] [パス設定...] でコンパイラのパスを設定してください。

f-2)  [GO] ボタンを押すか、メニュー・バーの [ファイル(F)] [コード生成...] を選択してください。



f-3) <ドライブとフォルダ選択> でコード生成するフォルダを確認して、[コード生成] ボタンを押してください。



ターゲット・ボード体験編 (Step1-6)

f-4) [コード生成]が完了し、ダイアログが表示されます。



Appliletで設定した値

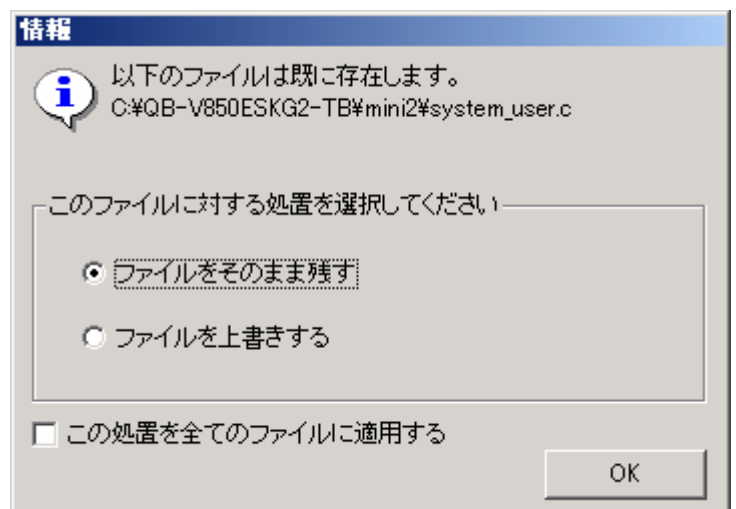
- メイン・システムクロック 5MHz
- CPUクロック選択は20MHz
- ウォッチドッグタイマ2は未使用
- サブ・クロック使用しない
- INTP0割り込み許可
- オンチップ・デバッグ・モード設定
- PMC2,PMC3を出力ポートにて、初期値を1とする
- タイマP0をインターバル・タイマに使用して、10msec毎に割り込みを行う
- タイマ00をインターバル・タイマに使用して、100msec毎に割り込みを行う

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。

ファイルをそのまま残すにチェックしても「main.c」,「xxxx_user.c」以外のファイルは必ず上書きされますので注意してください。ユーザ・プログラムを作成する場合、編集するソースファイルは「main.c」,「xxxx_user.c」を推奨しています。



ターゲット・ボード体験編 (Step2-1)

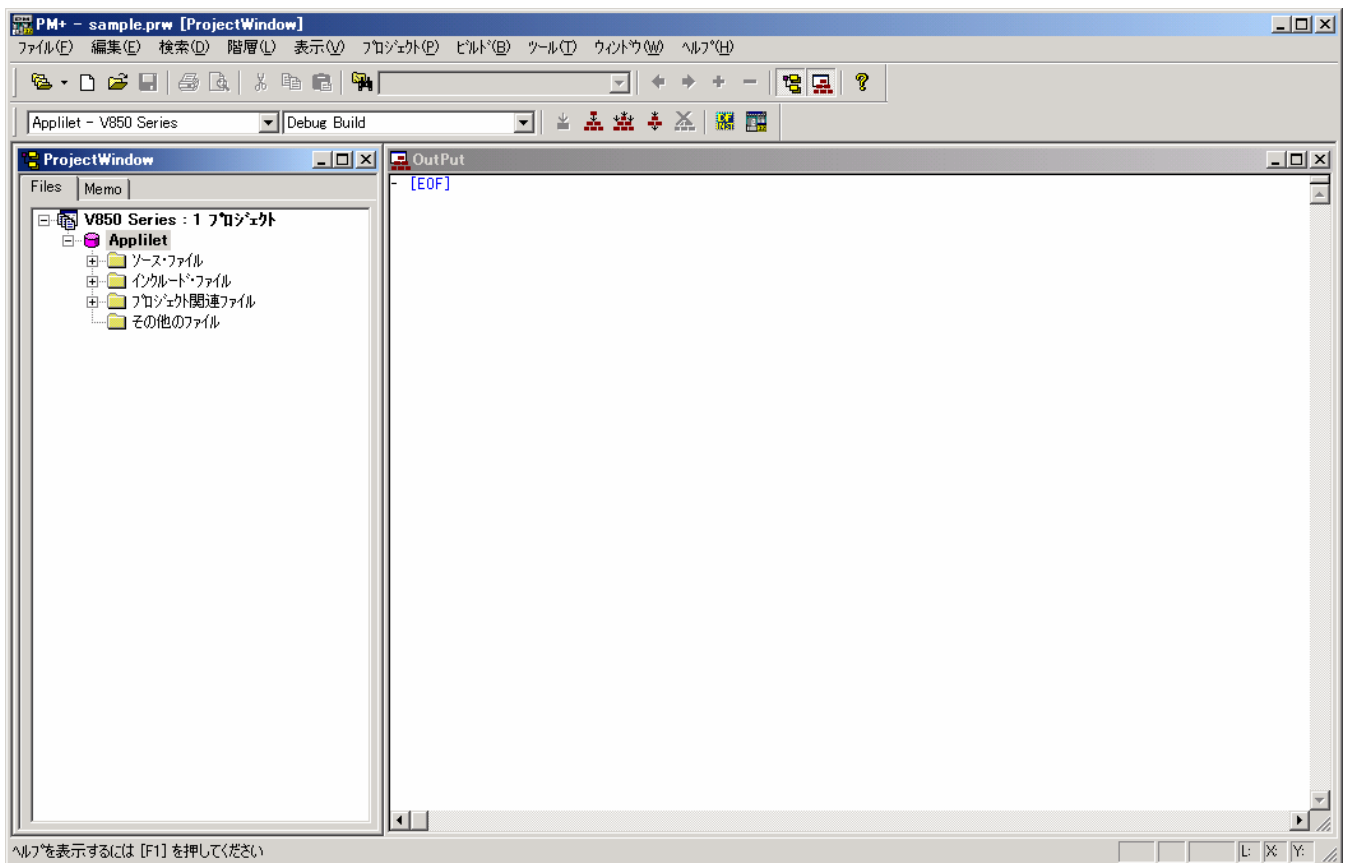
Step2 統合開発環境 PM+内のエディタを使い、サンプル・プログラムを作成します。

a. PM+ を起動します。

[スタート] [プログラム(P)] [NEC Electronics Tools] PM+ V6.xx

b. 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)...] を選択します。
ターゲット・ボード体験編 (Step1-8) のソースコードを生成したフォルダの
“sample.prw” を指定して、[開く(O)] ボタンを押してください。

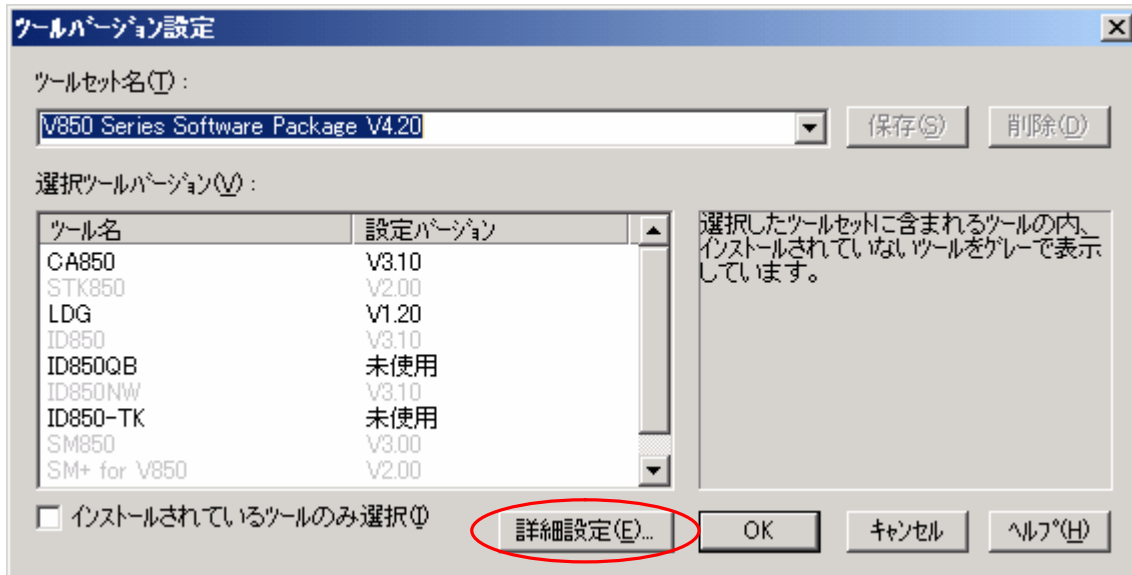


b-1) お使いのツールのバージョンによって、ワークスペースを開いた時に際に下図のメッセージが表示される事があります。

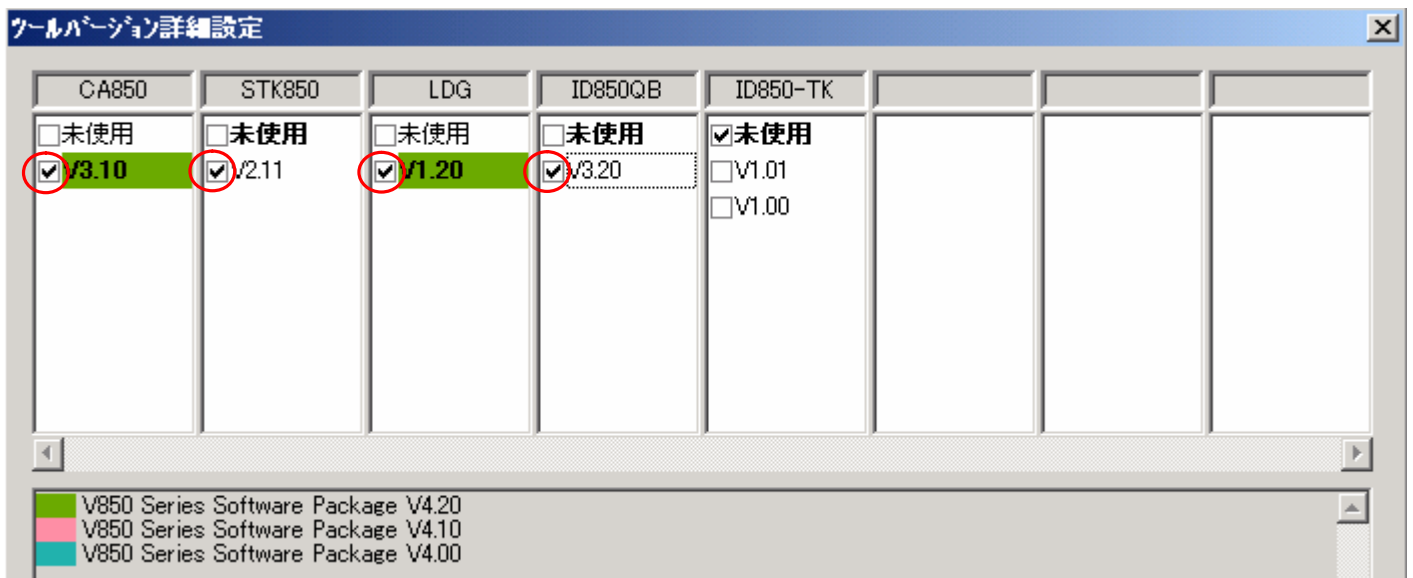


ターゲット・ボード体験編 (Step2-2)

b-2) ツールを選択し直してください。が表示された「OK」を押下し、ツールバージョン設定を行います。「詳細設定」を押下してください。



b-3) 未使用にチェックされている項目のツールを選択し直してください。CA850, ID850QBを必ずチェックしてください。OKを押下してPM+に戻ります。



ワンポイント

ツールのバージョンについて

ここで表示しているツールのバージョンは開発中の場合がありますので実際のバージョン表示と異なります。最新バージョンにつきましてはMINICUBE2のWEBページ、もしくは対象デバイスのWEBページを参照してください。

ターゲット・ボード体験編 (Step2-3)

c. プログラムを作成します

c-1) PM+ の ProjectWindow で main.c をダブルクリックしてエディタを起動します。

下記に示す青色の付いた部分のコードを追加してください。

```
extern UCHAR g_interval_sw;  
extern UINT g_interval_count;  
  
void main( void )  
{  
    g_interval_sw = 0;  
    g_interval_count = 1;  
    TMPO_Start();  
    TMOO_Start();  
    __EI();  
    while(1){  
        ;  
    }  
}
```

c-2) 同様に int_user.c をダブルクリックしてエディタを起動します。

下記に示す青色の付いた部分のコードを追加してください。

```
UCHAR g_interval_sw;  
UCHAR g_onetime_sw;  
  
__interrupt void MD_INTPO( void )  
{  
    /* TODO. Add user defined interrupt service routine */  
    if ( g_onetime_sw == 0 )  
    {  
        g_onetime_sw = 2;  
        g_interval_sw++;  
        g_interval_sw = g_interval_sw & 7;  
    }  
}
```

ターゲット・ボード体験編 (Step2-4)

c-3) 同様にtimer_user.cをダブルクリックしてエディタを起動します。

下記に示す青色の付いた部分のコードを追加してください。

```
extern UCHAR g_interval_sw;
extern UCHAR g_onetime_sw;
UINT g_interval_count;
UCHAR g_counter_data[ 8 ] = { 1, 3, 7, 15, 31, 47, 63, 127 };

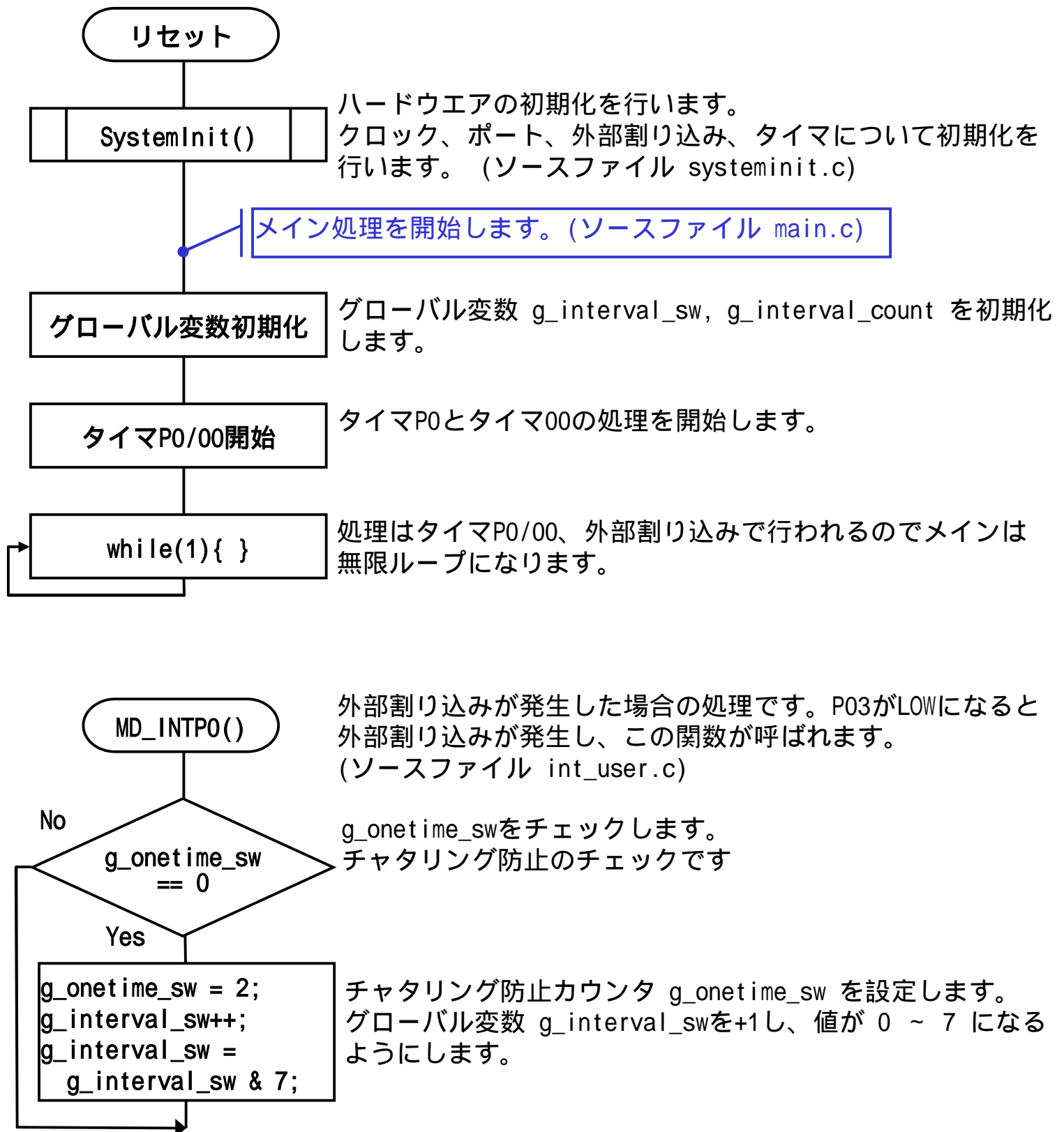
/*
** -----
** Abstract:
**       This function is TMO INTTP0CC0 interrupt service routine.
** Parameters:
**       None
** Returns:
**       None
** -----
*/
__interrupt void MD_INTTP0CC0( )
{
    /* TODO. Add user defined interrupt service routine */
    UCHAR inreg, outreg;
    if ( g_interval_count == 0 )
    {
        g_interval_count = g_counter_data[ g_interval_sw ];
        inreg = PCM.2;
        outreg = inreg ^ 1;
        PCM.2 = outreg;
        PCM.3 = inreg;
    }
    g_interval_count--;
}

        省略

__interrupt void MD_INTTM000( void )
{
    /* TODO. user code */
    if ( g_onetime_sw != 0 )
    {
        g_onetime_sw--;
    }
}
```


ターゲット・ボード体験編 (Step2-5)

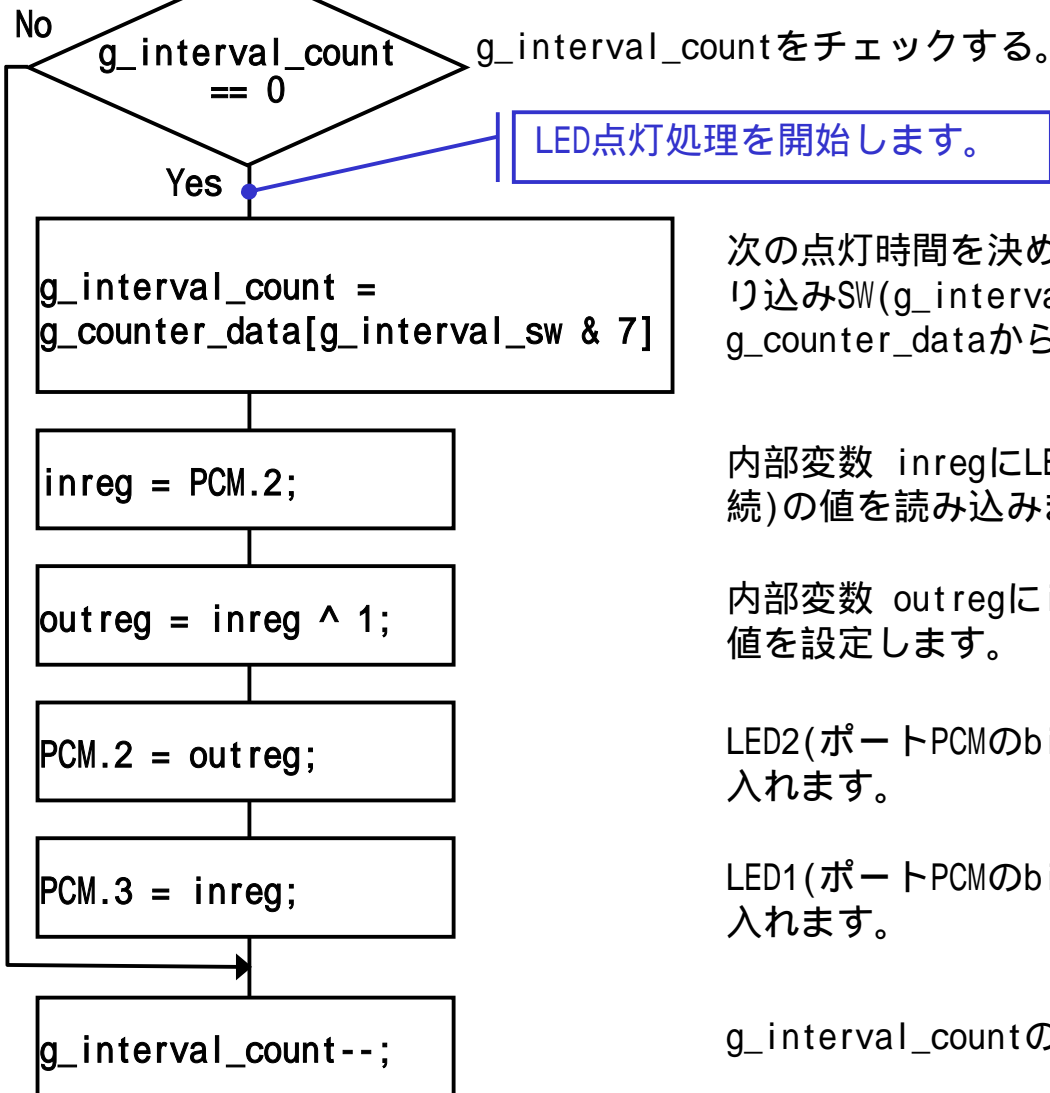
d. サンプル・プログラムの流れ図を示します。



ターゲット・ボード体験編 (Step2-6)

MD_INTTPOCCO()

10msec毎に発生するインターバル・タイマです。10msec毎に、この関数が呼ばれます。(ソースファイル timer_user.c)



LED点灯処理を開始します。

次の点灯時間を決めます。点灯時間は外部割り込みSW(g_interval_sw)の値を元にg_counter_dataから参照します。

内部変数 inregにLED2(ポートPCMのbit2接続)の値を読み込みます。

内部変数 outregにinregを反転(xor)させた値を設定します。

LED2(ポートPCMのbit2接続)に outregの値を入れます。

LED1(ポートPCMのbit3接続)に inregの値を入れます。

g_interval_countの値を減らします。

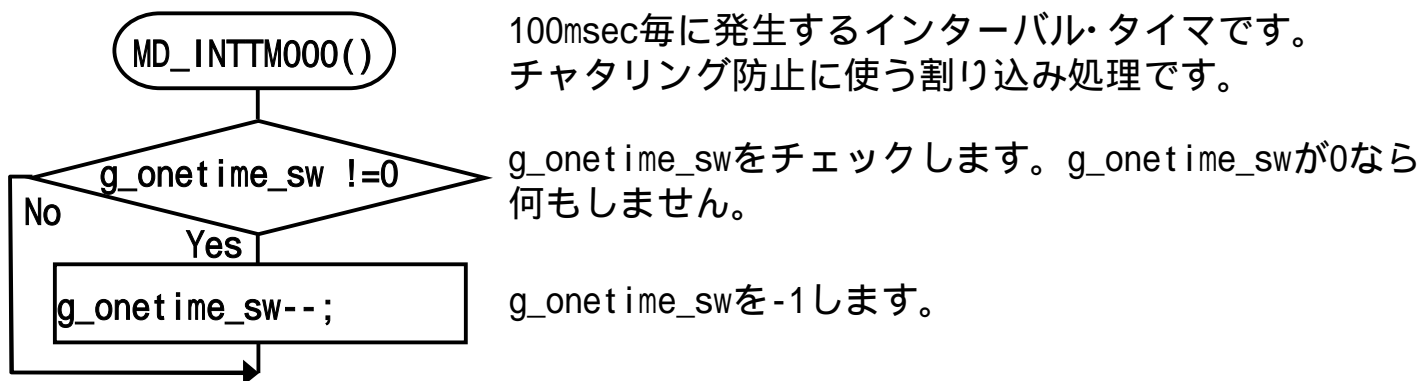


ワンポイント

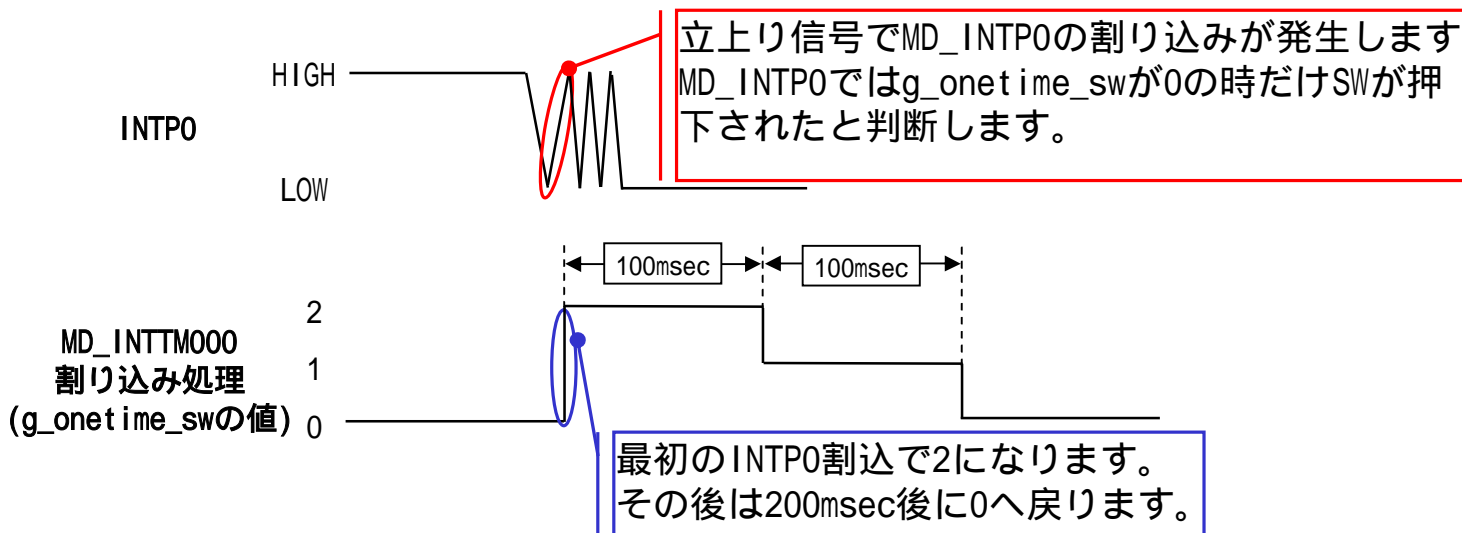
ポートの設定について

ポートの設定はbit単位で行うことができます。例えばポート5のbit1を1にするには P5.1 = 1; のように記述します。注意しなければならないのは、bit単位で指定するときの値は必ず1か0です。ポート5のbit7を1にするのは P5.7 = 0x80; ではなく P5.7 = 1; になります。

ターゲット・ボード体験編 (Step2-7)



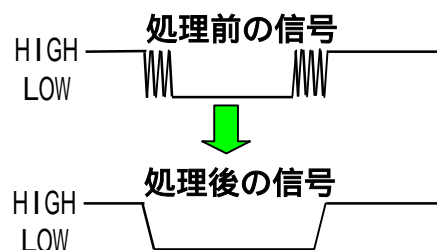
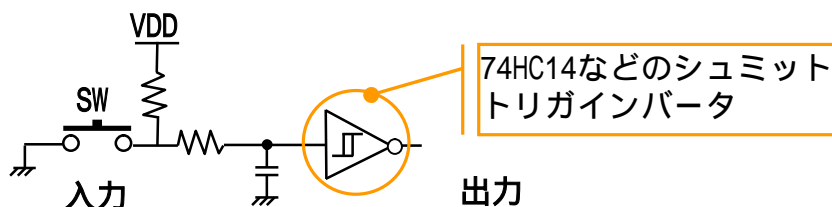
外部SWが押下されると g_onetime_sw に2が代入されます。g_onetime_swは100msec毎に-1されます。1度外部SWが押下されるとg_onetime_swが0になるまで200msecかかるわけです。MD_INTPO()の処理は、このg_onetime_swが0でないとLED点滅のタイミングを変更しません。ゆえにチャタリング発生し短い時間に何度MD_INTPO()が呼ばれても200msec以上の時間を空けないとLED点滅タイミングは変更されません。



💡 ワンポイント

チャタリングについて

ターゲット・ボード体験編ではチャタリング防止をソフトウェアで行いました。他にハードウェア(ターゲットシステム作成例参照)で行う場合など、様々な方法があります。今回ソフトウェアでチャタリングを防止しましたが、チャタリングが発生する期間を無視する時間を200msにしています。この処理では高速な応答を必要とするシステムには使えないので、この時間を短くするなどの工夫が必要になります。ハードウェアでは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。



ターゲット・ボード体験編 (Step2-8)

e. 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PCM2,PCM3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。g_interval_swはSWを押下すると外部割り込み関数 MD_INTP0() が呼ばれ +1されます。g_interval_swは0~7の値になります。g_interval_countは10msec毎に呼ばれる関数 MD_INTTPOCC0() で -1されます。g_interval_countが0の場合にPCM2,PCM3に接続されたLEDを交互に点灯させます。

グローバル変数の説明

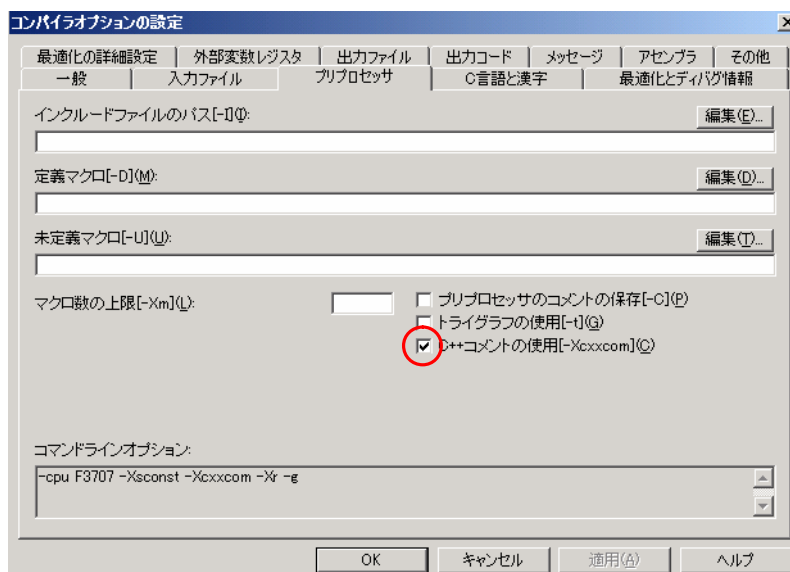
g_interval_sw: 外部SWを押下すると変化する。0~7の値になる。
g_onetime_sw: この値が0の時に外部割込が有効になります
g_interval_count: 10msec毎に -1されるカウンタ。0の時にLED点灯を変更します。
g_counter_data[8]: 点灯速度を決めるデータ。8段階のスピードを設定します。

💡 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 // を使用が可能です。


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。[プリプロセッサ]タブを選択すると下図のダイアログが開きます。



C++コメントの使用 [-Xcxcocom] (C) にチェックしてください。

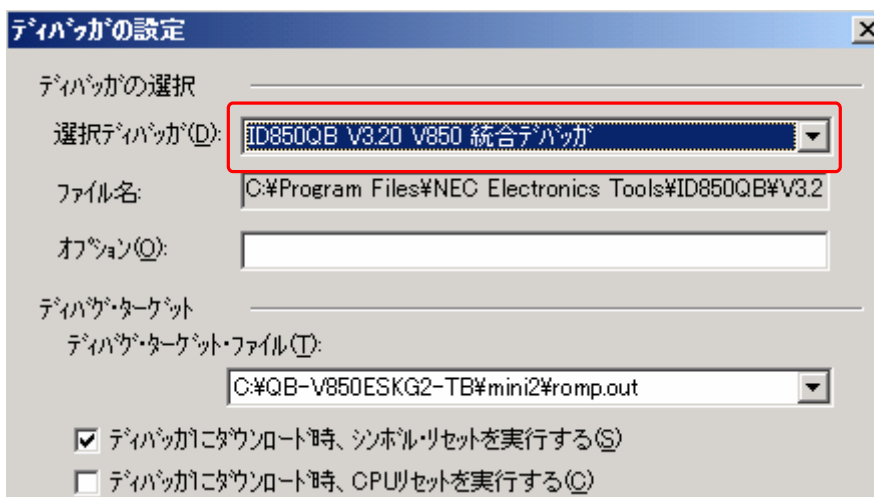
ターゲット・ボード体験編 (Step3)

Step3 PM+でサンプル・プログラムのビルド(コンパイル/リンク)を行います。

- a. ビルド・ボタン  を押すか、メニュー・バーの[ビルド (B)] [ビルド (B)]を選択します。記述したソースに誤りが無ければ下図のダイアログが表示されます。



- b. PM+ から連携起動するデバッガを設定します。
メニュー・バーの[ツール(T)] [デバッガの設定(D)...]を選択します。『デバッガの設定』ダイアログのプルダウンメニューで、[選択デバッガ (D)]に “ ID850QB Vx.xx 総合デバッガ ” を設定します。



💡 ワンポイント

ビルドについて

ビルドメニューの[ビルド -> デバッグ (A) F5]または[ビルド -> デバッグ (U)]を選択する場合

ビルド->デバッグ(A)	F5
ビルド->デバッグ(U)	

必ず連携起動するデバッガを指定してください。

デバッガの指定により下記のデバッグ環境が実現されます。

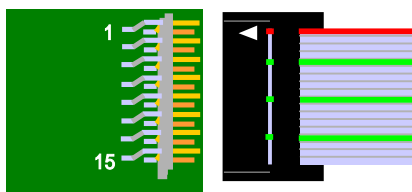
- ・MINICUBE2/IECUBE用デバッガ: ID850QB V3.20 V850 総合デバッガ
- ・TK用デバッガ : ID850-TK V1.01
- TK-V850KJ2 (株)アプリケーション製StarterKit

ターゲット・ボード体験編 (Step4-1)

Step4 Step3で作成したオブジェクトファイルをMINICUBE2でオンチップ・デバッグし、統合デバッガID850QBの使い方を学びます

a. MINICUBE2のターゲット・ボードへの接続方法

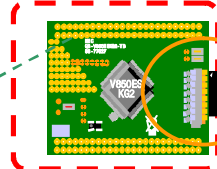
コネクタの1pinを合わせて接続します



ターゲット・ボードとMINICUBE2の設定が終わるまでPCと接続しないでください。

16pinターゲット・ケーブル

ターゲット・ボード



MINICUBE2

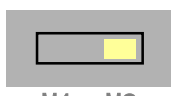
スイッチを設定します

モード選択SW

電源選択SW

M2

5V出力



M1 M2

3 T 5

b. 接続の順番について

下記の順番で接続を行ってください。順番を間違えるとターゲット・システムを壊す原因となります。


1. MINICUBE2のSWを設定する。
2. ターゲット・ボードへ16pinターゲット・ケーブルを接続する。
3. MINICUBE2とPC本体を接続する。

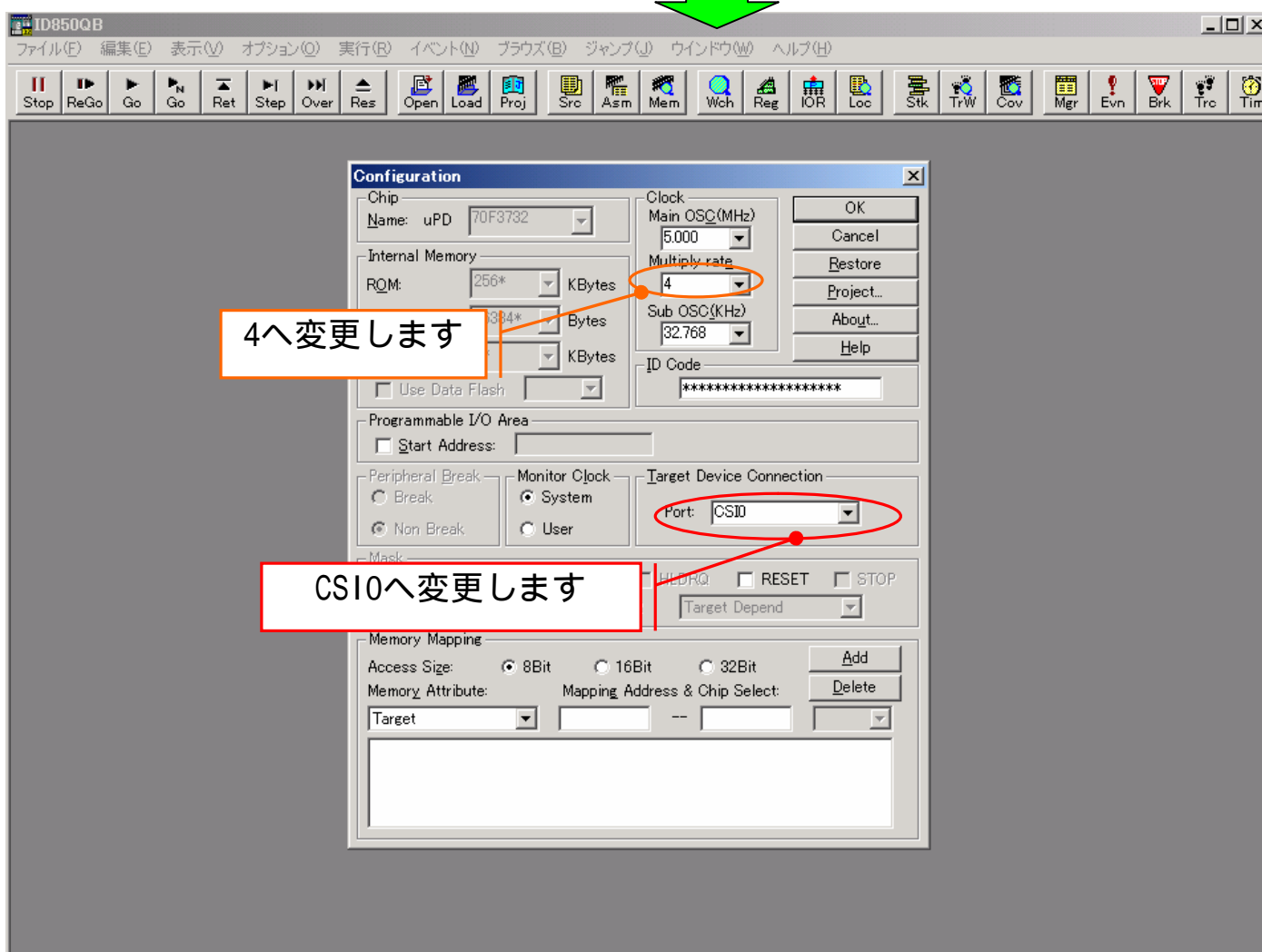
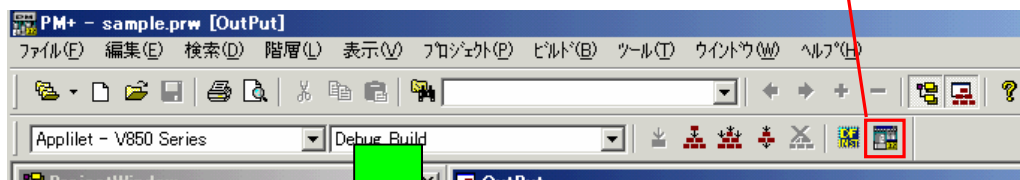
💡ワンポイント

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。5V、3Vの供給はデバイスによって異なります。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ください。

ターゲット・ボード体験編 (Step4-2)

- c. ID850QBを起動します。PM+のメニューよりデバッグ・ボタンを押すか、メニュー・バーの[ビルド (B)] [デバッグ (D)]を選択します。Target Device ConnectionのPortをCSI0へ変更します。また、Multiply rateを4にします。



ワンポイント

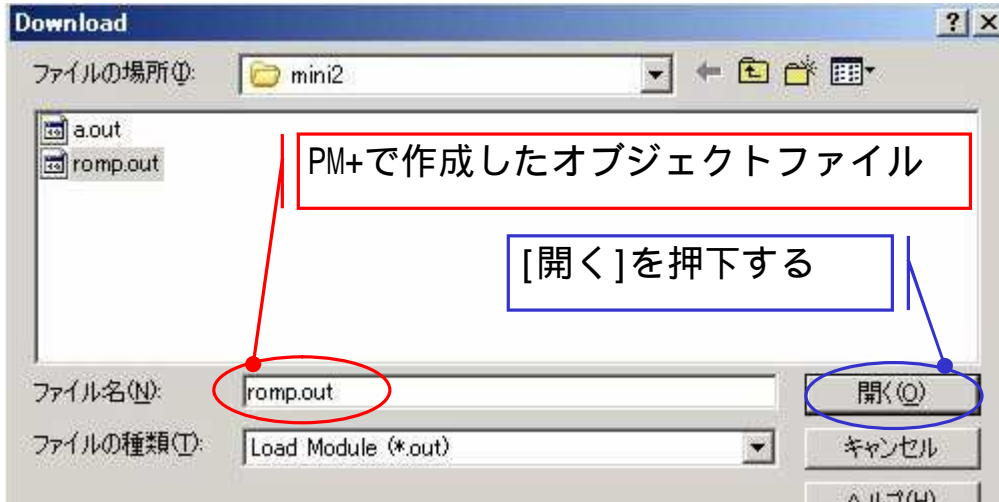
Multiply rate、Target Device Connectionについて

Multiply rateはメイン・クロックの逡倍を指定します。本ボードに搭載のV850ES/KG2は4逡倍(入力クロックが5MHzに対して動作クロックは20MHz)なので4を選択します。

Target Device Connectionはオンチップ・デバッグを行う際の通信方法を指定します。本ボードはS100, S000, SCK00, PCM0の各端子で3線式シリアル・インタフェースを利用してオンチップ・デバッグを行っています。

ターゲット・ボード体験編 (Step4-3)

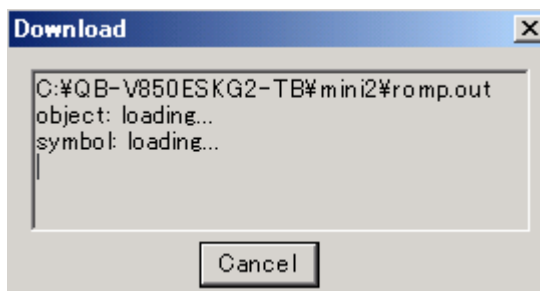
- d. OKを押下し、[ファイル(F)] [ダウンロード(D)...]を選択し「romp.out」を開きます。2回目以降の起動では自動的にダウンロードもできます。



ワンポイント
MINICUBE2とターゲット・ボードの接続が成功するとMINICUBE2のLEDが青に点灯します。

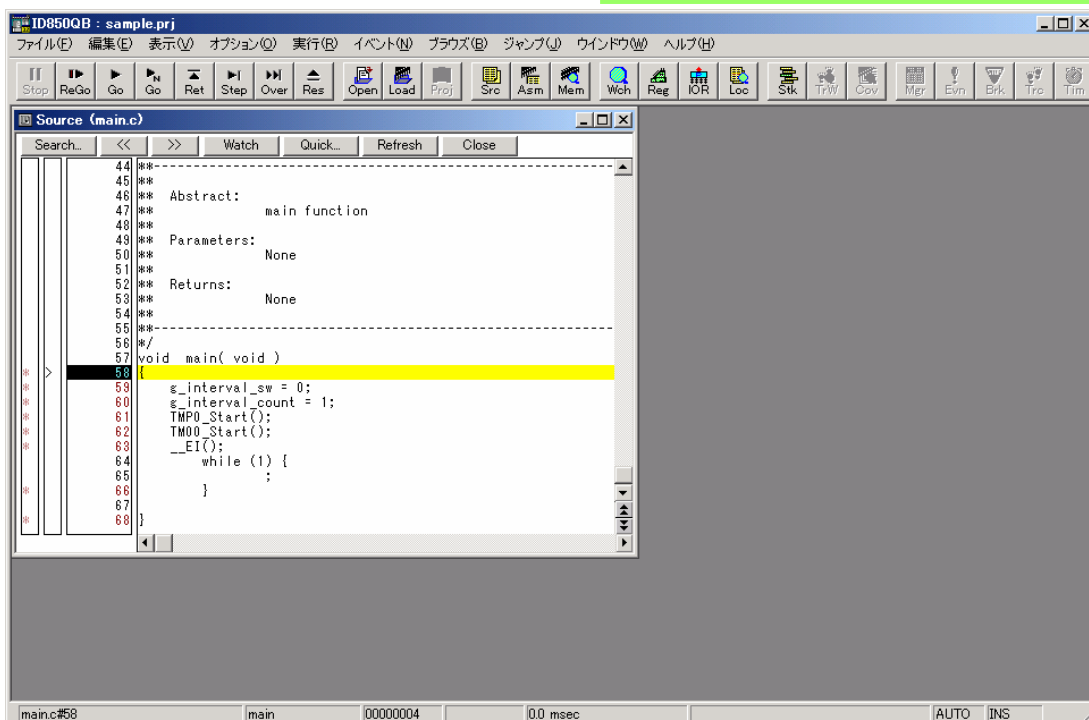
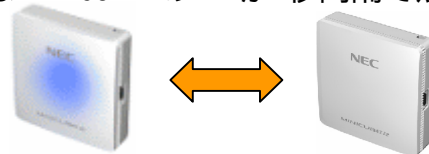


- e. ファイルがダウンロードされ、main.cが開きます



ワンポイント

ダウンロードの時間はプログラムのサイズやターゲット・システムとMINICUBE2との接続方法によって変化しますが、このサンプル・プログラムでは8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。

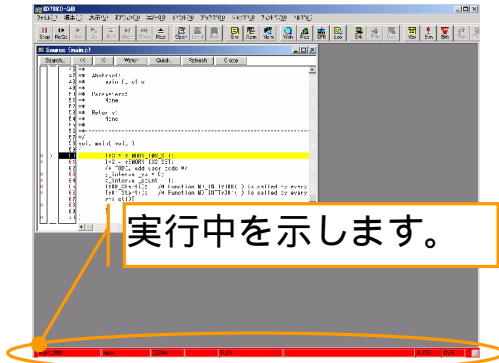


ターゲット・ボード体験編 (Step4-4)

f. Goボタン を押下してプログラムを実行します

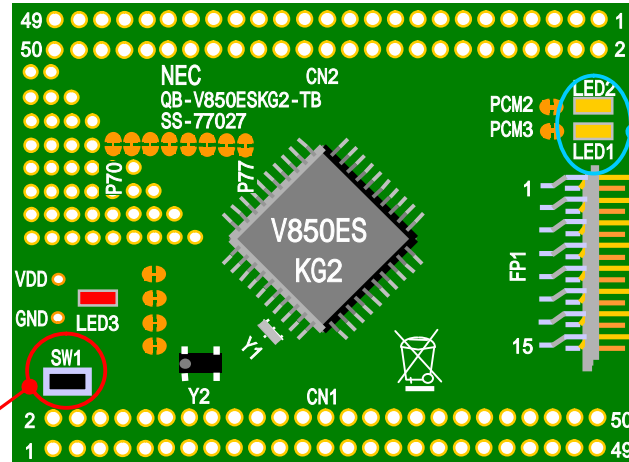
PC画面の表示

ターゲット・ボード上の動作



実行中を示します。

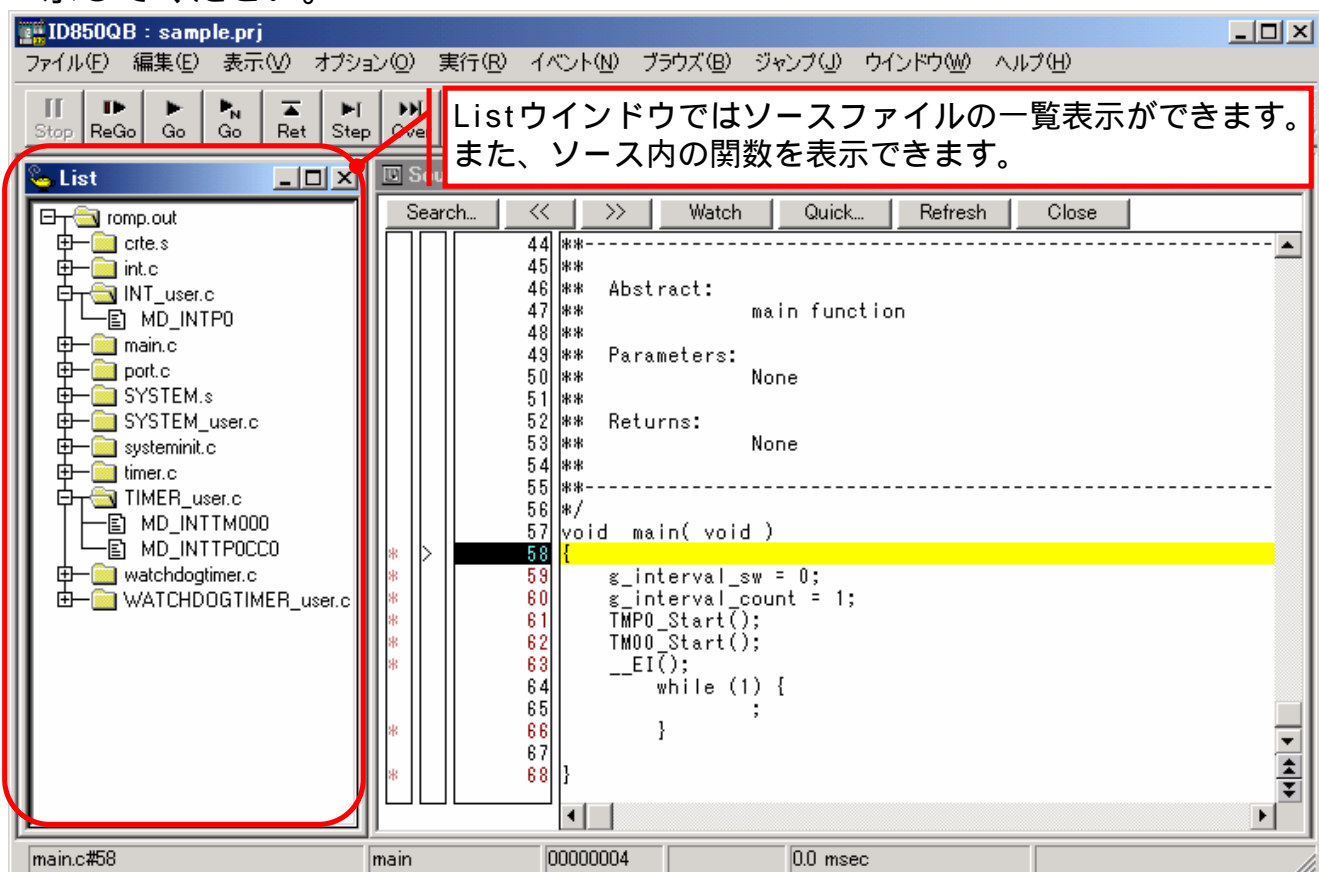
SWを押下するとLEDの点滅速度が変化します。



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を押すと点滅がだんだんゆっくりになります。

g. Listウィンドウの表示方法

メニュー・バーの [ブラウズ(B)] [その他(E)] [2 List]でListウィンドウを表示してください。



ターゲット・ボード体験編 (Step4-5)

h. ブレーク・ポイントの設定方法

h-1) ブレーク・ポイントを設定し、任意の場所でプログラムの実行を停止させます。
Listウインドウより[MD_INTTPOCC0]関数をダブルクリックし、「timer_user.c」を開きます。

h-2) 「timer_user.c」を開いたら65行の左をクリックしてブレーク・ポイントを設定します。設定した行は赤い反転表示になります。

The screenshot shows the IDE interface with the following annotations:


- 1** MD_INTTPOCC0関数をダブルクリックします。 (Double-click the MD_INTTPOCC0 function in the List window.)
- 2** timer_user.cを開きます。 (timer_user.c is opened.)
- 3** ここをクリックすると「B」と表示されます。 (Clicking here displays 'B').
- 4** ブレーク・ポイントを設定した行が反転表示されます。 (The line where the breakpoint is set is displayed in reverse.)

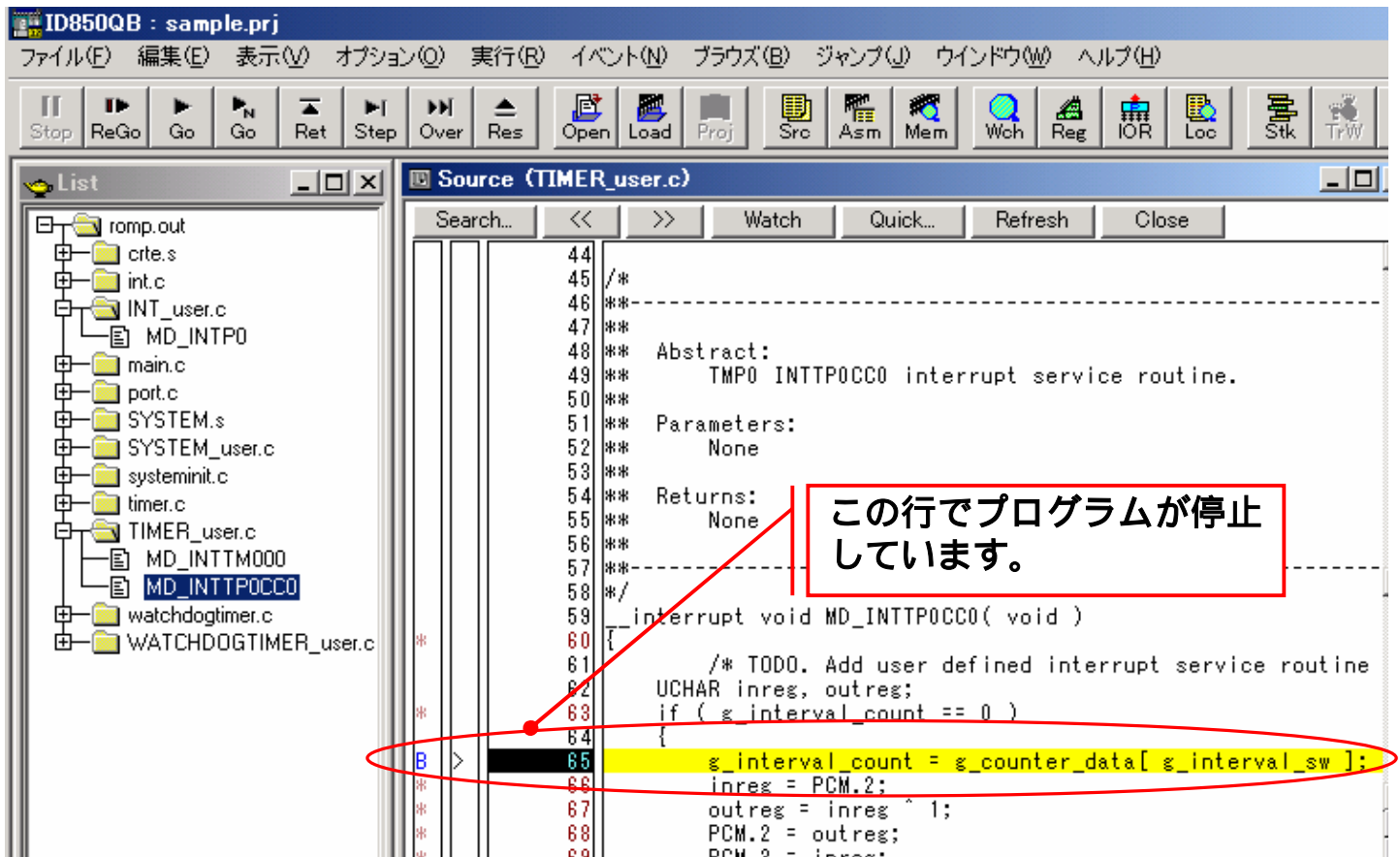
💡ワンポイント

イベントについて

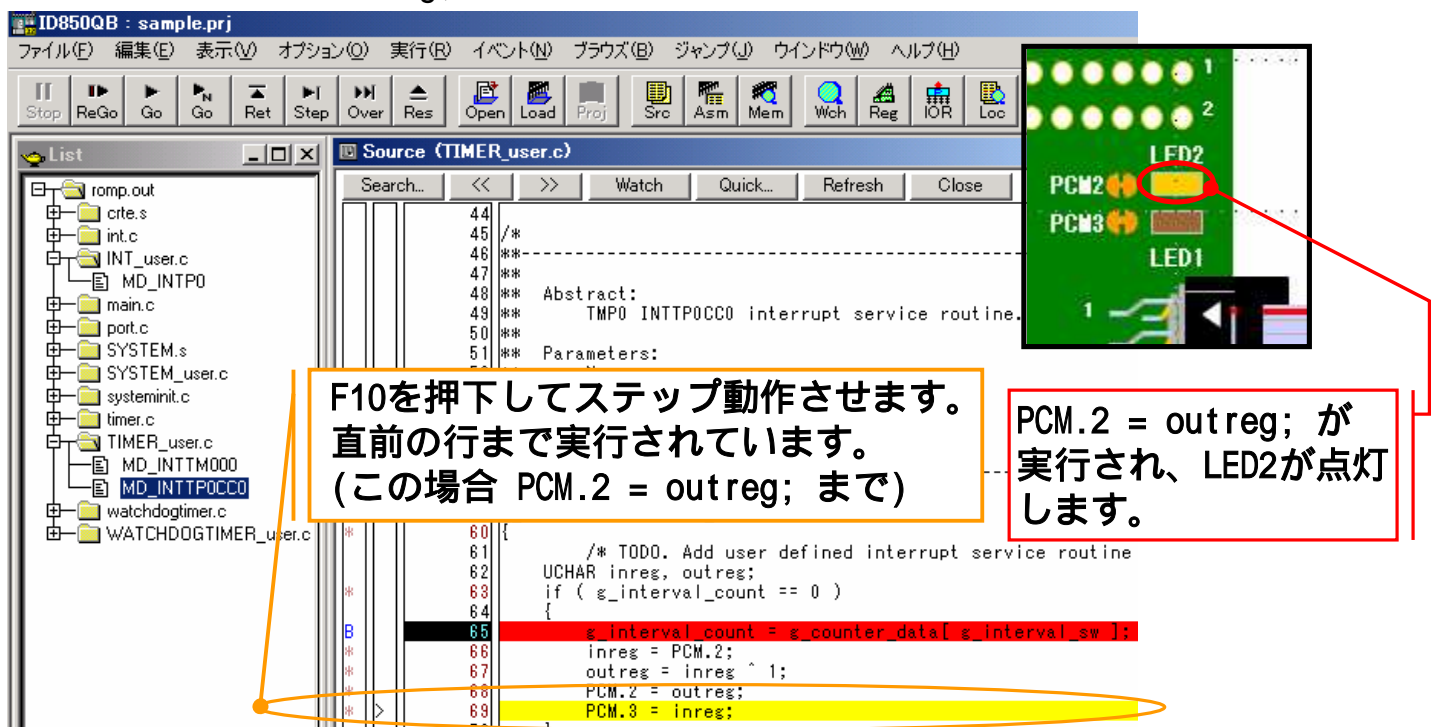
イベントとは、「アドレス0x1000番地をフェッチした」、「アドレス0x2000番地にデータを書き込んだ」などのデバッグにおけるターゲット・システムの特定の状態を指しています。このようなイベントをブレーク、トレース等の各デバッグ機能のアクション・トリガとして利用しています。ここで設定した「B」はブレーク・イベントです。イベントの種類は他に「トレース・イベント」、「タイマ・イベント」、「スタブ・イベント」、「スナップショット・イベント」があります。

ターゲット・ボード体験編 (Step4-6)

- h-3) [ReGo]ボタン  を押して、プログラムを実行します。
 すぐにプログラムはブレーク・ポイントで停止し、LEDは両方消灯しています。



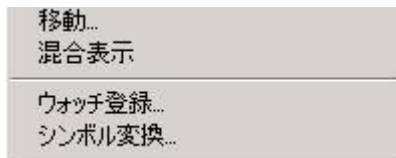
- h-4) この状態でF10を押下してステップ動作させます。69行までステップ動作させると PCM.2 = outreg; が実行されLED2が点灯します。



ターゲット・ボード体験編 (Step4-7)

i. ウォッチの登録

- i-1) 表示させたい変数をマウスでドラッグして選択し、右クリックで「ウォッチ登録」を選択します。



[表示(V)] [ウォッチ登録(W)...]でも同様です。[表示(V)] [ウォッチ追加(I)]の場合はすぐにWatchウィンドウヘデフォルトの表示方法で追加されます。

- i-2) Watchウィンドウでは表示方法を設定します。10進表示を行うときはDecをチェックします。

The screenshot shows the IDE interface. The source code editor displays the following code:

```

42 /*
43 **
44 **
45 **
46 ** Abstra
47 **
48 **
49 ** Parame
50 **
51 **
52 ** Return
53 **
54 **
55 **
56 */
57 void main( void )
58 {
59     g_interval_sw = 0;
60     g_interval_count = 1;
61     TMPO_Start();
62     TMOO_Start();
63     while
64
65

```

The 'Add Watch' dialog box is open, showing the following settings:

- Name: g_interval_sw
- Radix: Proper Hex Dec Oct Bin Strine
- Size: Adaptive Byte Half Word Word
- Number: []

Annotations in the image:

- ①: Points to the code line `g_interval_sw = 0;` in the source code editor.
- ②: Points to the 'Dec' radio button in the 'Add Watch' dialog box.

Text boxes provide instructions:

- 「マウスでドラッグし、右クリックします。メニューより「ウォッチ登録...」を選択します。」 (Drag with mouse, right-click. Select 'Add Watch...' from the menu.)
- 「10進表記に設定します」 (Set to decimal notation.)

- i-3) Watchウィンドウが開き、変数が表示されます。

The Watch window is open, showing the following variables and their values:

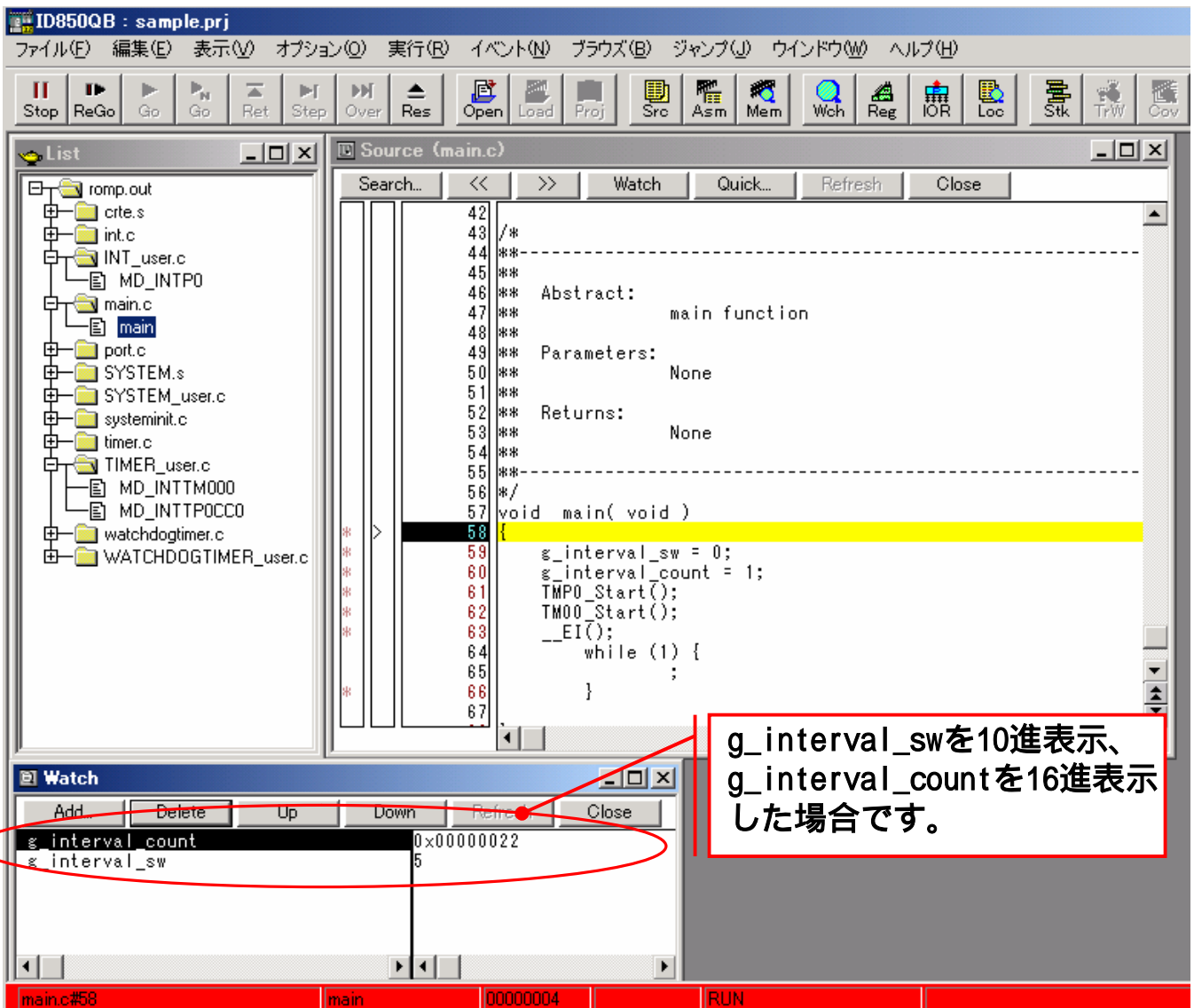
Variable	Value
g_interval_sw	0
g_interval_count	0x00000000

Annotations in the image:

- A red box highlights the variable names and their values.
- A red box contains the text: 「g_interval_swは10進表示、g_interval_countは16進表示に設定した例です」 (Example where g_interval_sw is set to decimal display and g_interval_count is set to hexadecimal display.)

ターゲット・ボード体験編 (Step4-8)

i-4) [ReGo]ボタン  を押して、プログラムを実行します。



The screenshot shows the IDE interface for ID850QB: sample.prj. The source code window displays the main function in main.c, with line 58 highlighted. The Watch window shows the following variables:

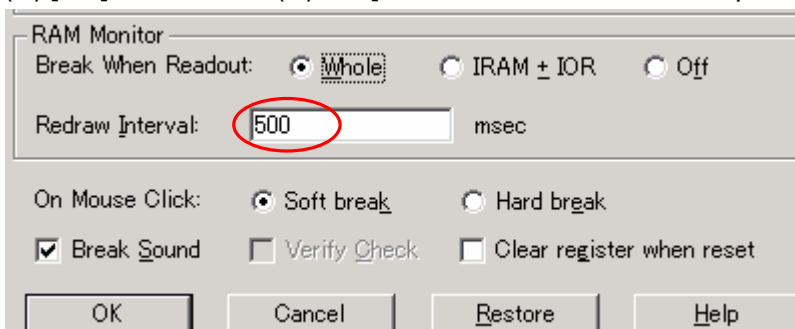
Variable	Value
g_interval_count	0x00000022
g_interval_sw	5

A red box highlights the Watch window, and a callout points to it with the text: "g_interval_swを10進表示、g_interval_countを16進表示した場合です。"

ワンポイント

実行中のWatchウィンドウ更新について

Watchウィンドウ内の表示はデフォルトで500msec毎に更新されます。この値を変えるには[オプション(O)] [拡張オプション(X)...]を選択し、「Extended Option」ダイアログを表示します。



The screenshot shows the 'Extended Option' dialog box. The 'Redraw Interval' field is set to 500 msec and is circled in red. Other options include 'Break When Readout' (Whole), 'On Mouse Click' (Soft break), and 'Break Sound' (checked).

Redraw Internalの値で表示を更新する時間が指定できます。時間は100msec単位で0 ~ 65500まで指定できます。0、または空欄を指定した場合はリアルタイム表示を行いません。

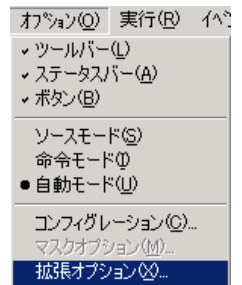
ここで設定した値で必ず表示更新されるとは限りません。ホストマシンの速度や実行中のプログラムに影響されるので実動作は遅くなる場合があります。

ターゲット・ボード体験編 (Step4-9)

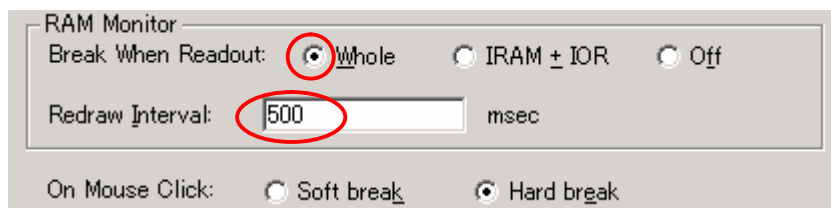
j. RAMモニタ機能を使う(実行中の変数表示を行います)

j-1) ID850QBの設定を変更します

[オプション(O)] [拡張オプションの設定(X)...]を選択します。



j-2) RAM Monitorエリアの [Break When Readout]項目の“Whole”にチェックします。また、“Redraw Interval:”の値で表示更新時間を変更できます。



j-3) 任意の変数をウォッチ登録します。



j-4) プログラムを実行すると実行中でも変数がリアルタイムに表示されます。プログラムが割り込みを受け付け中のみ表示が更新されます。

プログラムが実行中を示します。

プログラムが実行中だが、変数の値がリアルタイムに変化しています。

```

57 void main( void )
58 {
59     g_interval_sw = 0;
60     g_interval_count = 1;
61     TM00_Start();
62     TM00_Start();
63     __EI();
64     while (1) {

```

Watch window content:

```

g_interval_count 0
g_interval_sw 0

```

IDE status bar content:

```


main.c#58 | main | 00000004 | RUN

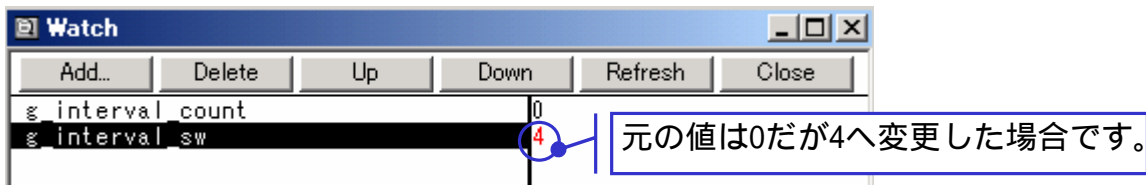
```



ターゲット・ボード体験編 (Step4-10)

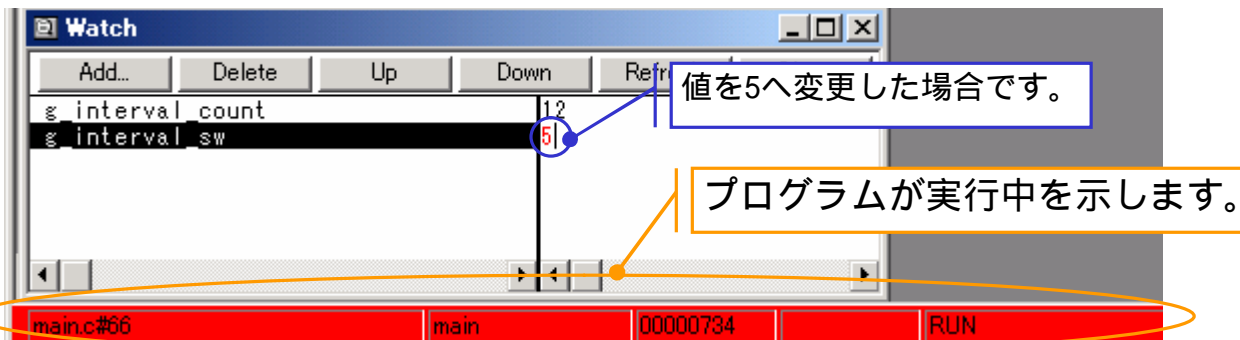
💡 ワンポイント

ブレーク時の変数書き換えについて


プログラム実行中に[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ウィ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。

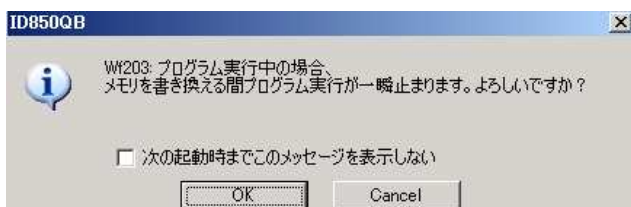
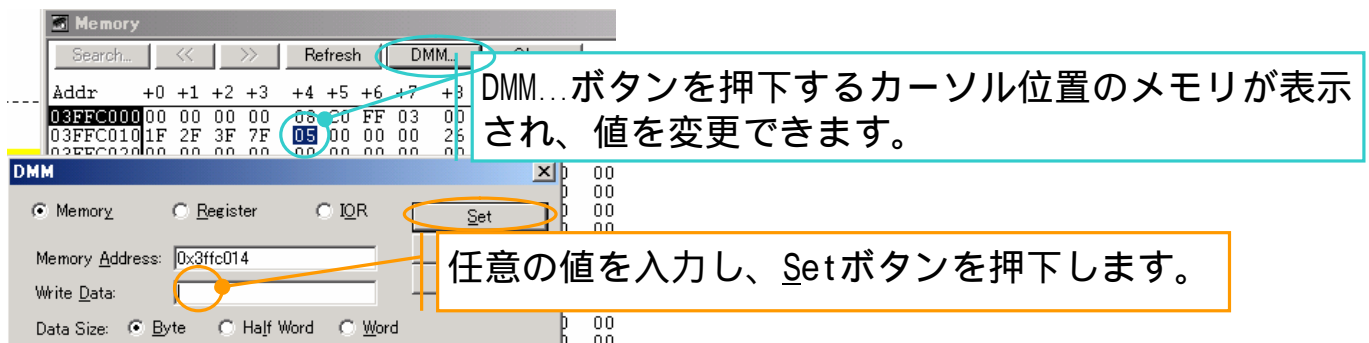


変数の値を変更後、[実行(R)] [継続して実行(G)]またはGoボタン  で実行を再開できます。上記の例(g_interval_swを4へ変更した場合)ですとLED点滅の速度がゆっくりになります。また、プログラム実行中に値を変更することも可能です。



実行中にMemory/Register/IQRの内容を変更する方法としてDMM機能があります。

[編集(E)] [DMM(D)...]で設定可能です。メモリの内容を変更するには[プラス(B)] [メモリ(M)] またはMemボタン  でMemoryウィンドウを開き、Memoryウィンドウ内のDMM...ボタンを押下すると表示されているMemoryAddressが自動で挿入されるので容易に設定可能です。



値を入力すると左記のダイアログが表示されます。「OK」ボタンを押下すると、入力した値が反映されます。

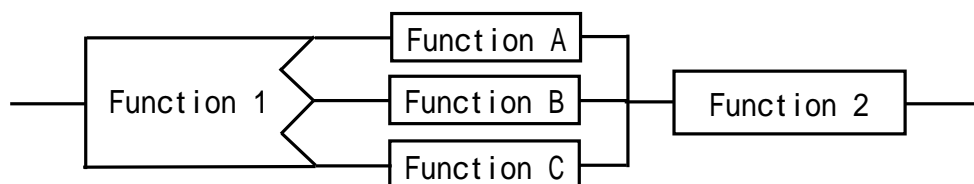
ターゲット・ボード体験編 (Step4-11)

💡 ワンポイント

高度なブレーク設定(イベントリンク)について(その1)

ID850QBには高度なブレーク・ポイントの設定があります。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。

ID850QBを起動し“ romp.out ”をダウンロードしてください。そして「int_user.c」を開いてください。このファイルはターゲット・ボードのSWが押下された時に呼ばれる割り込み関数です。

メニューより[イベント(N)] [イベント(E)...]を選択し、Eventウィンドウを開きます。[Event Status]を“Before Execution”にします。[Address]は設定したい行番号を指定します。指定方法は“ファイル名#行番号”です。設定後に[Set]ボタン押下でイベントが作成されます。

[Set]ボタン押下でイベントが作成されました。

```

42 /*
43 **
44 **
45 ** Abstract:
46 ** This function is INTPO Interrupt se
47 **
48 ** Parameters:
49 ** None
50 **
51 ** Returns:
52 ** None
53 **
54 **
55 */
56
57 _interrupt void MD_INTPO( void )
58 {
59     /* TODO: Add user defined interrupt
60     if ( g_onetime_sw == 0 )
61     {
62         g_onetime_sw = 2;
63         g_interval_sw++;
64         g_interval_sw = g_interval_sw &
65     }
66 }
  
```

Event Manager:

[E.] Evt00001

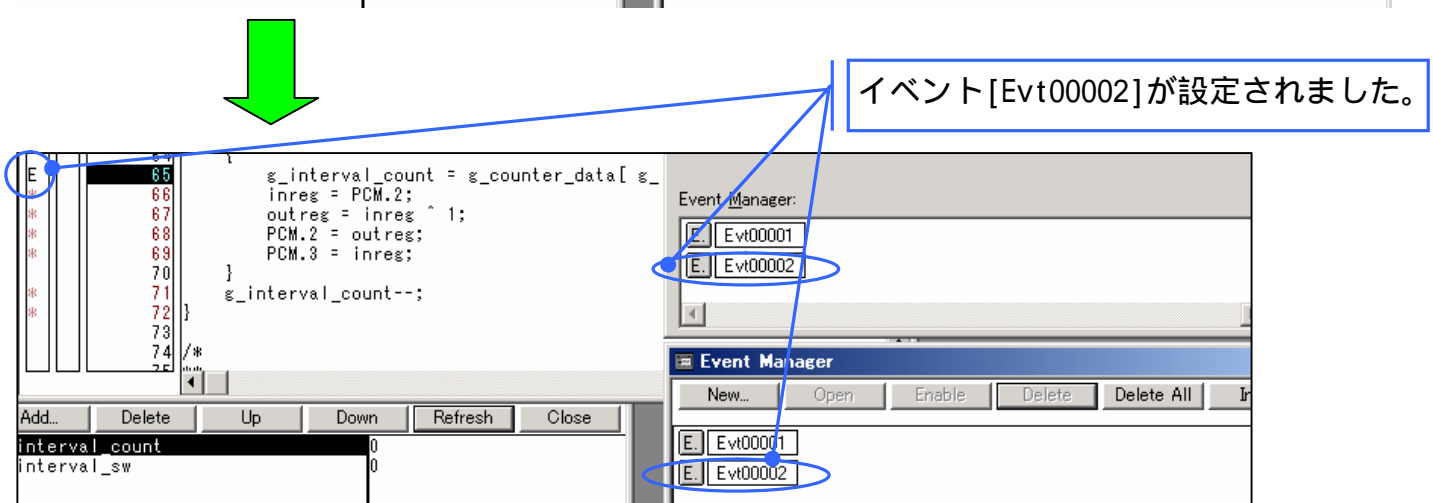
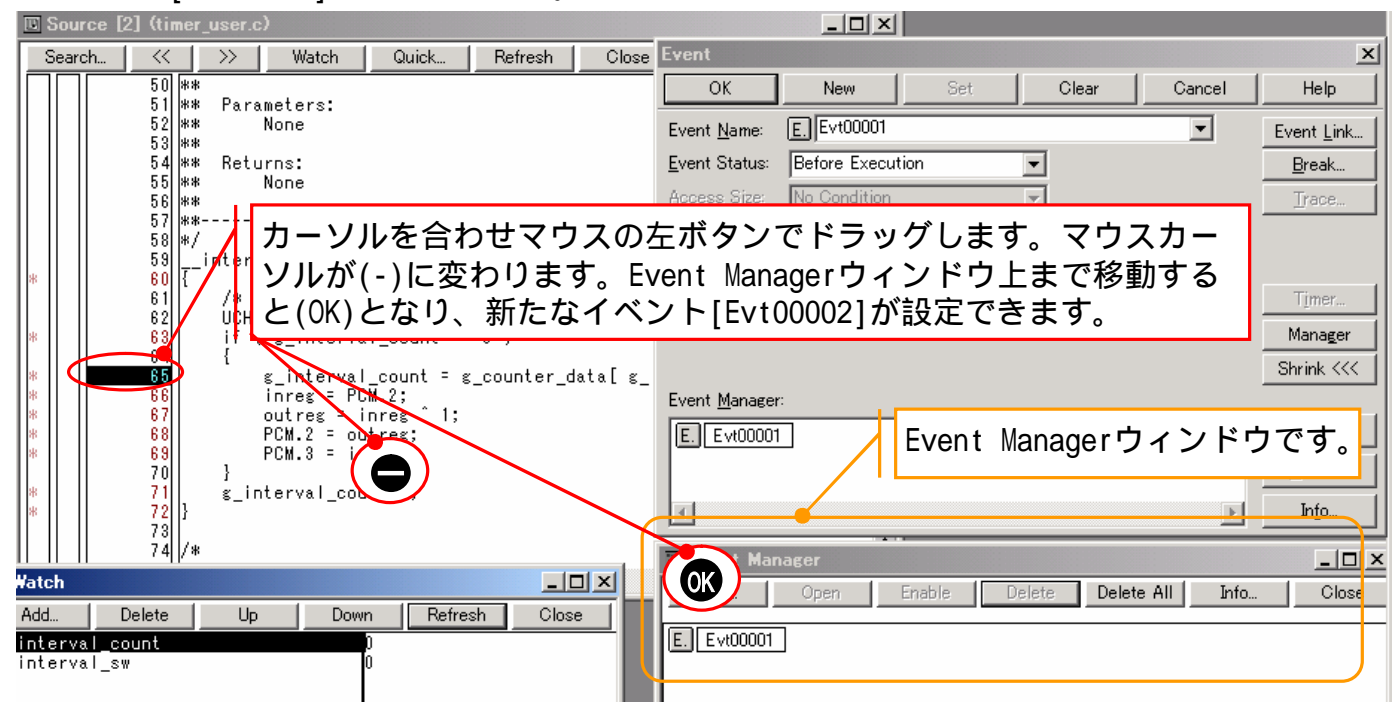
ターゲット・ボード体験編 (Step4-12)

💡ワンポイント

高度なブレーク設定(イベントリンク)について(その2)

前ページで「int_user.c」の61行目にイベントを作成しました。同様に「timer_user.c」の65行目にイベントを作成します。この関数「MD_INTTPOCC0」は10msec毎に呼ばれます。ここにブレーク・ポイントを設定すると10msec毎にプログラムが止まってしまうのですが、前に設定したイベントとリンクさせると条件に合致した場合のみプログラムを停止することができます。

下記の例を説明します。「timer_user.c」の65行目にイベントを作成します。前ページとは異なるもう1つの方法で設定します。まず、「timer_user.c」を開きます。メニューより[イベント(N)] [イベント・マネージャ(M)]を選択し、Event Managerウィンドウを開きます。「timer_user.c」の65行目にカーソルを移動し、マウスの左ボタンでドラッグしたままEvent Managerウィンドウへ移動するとイベント[Evt00002]が設定できます。



ターゲット・ボード体験編 (Step4-13)

💡 ワンポイント

高度なブレイク設定(イベントリンク)について(その3)

前ページまででイベントを2つ作成できました。次に設定するのは「Event Link」です。イベントとイベントを連携する設定です。この関数「MD_INTTP0CC0」は10msec毎に呼ばれます。ここにブレイク・ポイントを設定すると10msec毎にプログラムが止まってしまうのですが、前に設定したイベントとリンクさせると条件に合致した場合のみプログラムを停止することができます。

Eventウィンドウ内の[Event Link...]ボタンを押下し、Event Linkウィンドウを開きます。同じウィンドウ内の[Evt00001]、[Evt00002]をそれぞれ[Phase 1:]、[Phase 2:]へマウスの左ボタンをドラッグしてください。[OK]ボタン押下で[Lnk00001]が設定されます。

[Event Link...]ボタンを押下して Event Linkウィンドウを開きます。

[Lnk00001]が設定されました。

```

50  /**
51  /**  Parameters:
52  /**      None
53  /**
54  /**  Returns:
55  /**      None
56  /**
57  /**
58  /**
59  _interrupt void MD_INTTP0CC0( void
60  {
61  /* TODO. Add user defined inter
62  UCHAR inreg, outre;
63  if ( §_interval_count == 0
64  {
65  §_interval_count = §_counte
66  inreg = PCM.2;
67  outre = inreg ^ 1;
68  PCM.2 = outre;
69  PCM.3 = inreg;
70  }
71  §_interval_count--;
72  }
73  }
74  }
75  }

```

interval_count 0
interval_sw 0

Event Link dialog fields:
Link Name: Lnk00001
Phase 1: E. Evt00001
Phase 2: E. Evt00002
Event Manager list:
E. Evt00001
E. Evt00002
L. Lnk00001

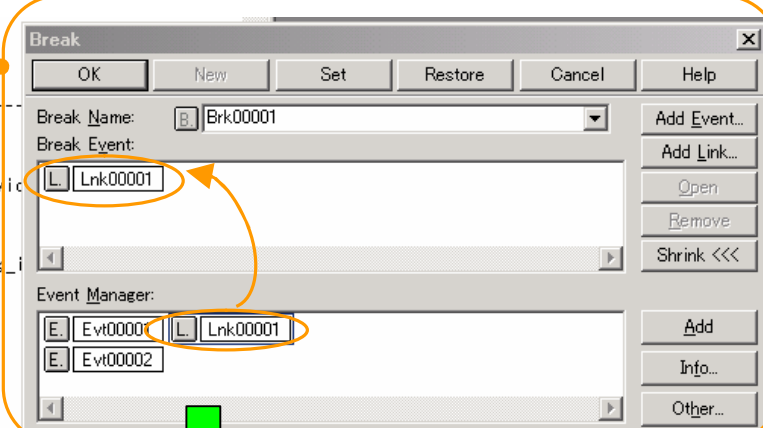
ターゲット・ボード体験編 (Step4-14)





ワンポイント

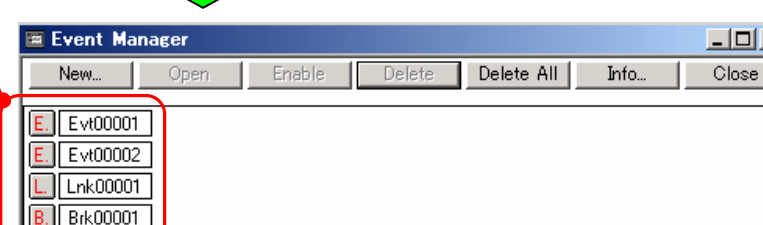
高度なブレイク設定(イベントリンク)について(その4)

最後に設定するのは「ブレイク・ポイント」です。Event Linkを元に「ブレイク・ポイント」を設定します。メニューより[イベント(N)] [ブレイク(B)...]を選択し、Breakウィンドウを表示します。同じウィンドウ内の「Event Manager:」より[Lnk00001]をマウスの左ボタンでドラッグし、「Break Event」へ設定します。[OK]ボタンを押下するとブレイク・ポイントが設定されます。

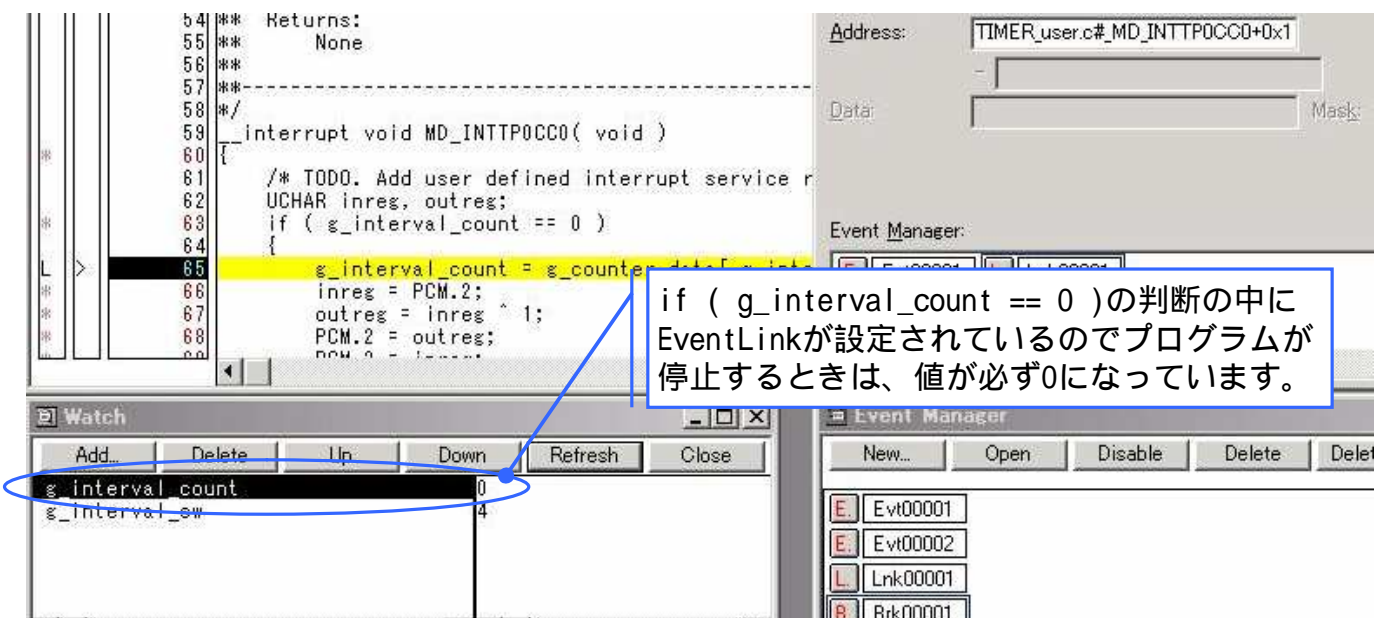
Breakウィンドウを開きます。



[Brk00001]が設定され、それぞれのイベントが有効になり、アイコンが   から   へ変わりました。



プログラムを実行してください。SWを押した時にg_interval_countが0だった場合にのみプログラムが停止します。少し停止しにくいですが、何度かSWを押下してください。



if (g_interval_count == 0)の判断の中に EventLinkが設定されているのでプログラムが停止するときは、値が必ず0になっています。

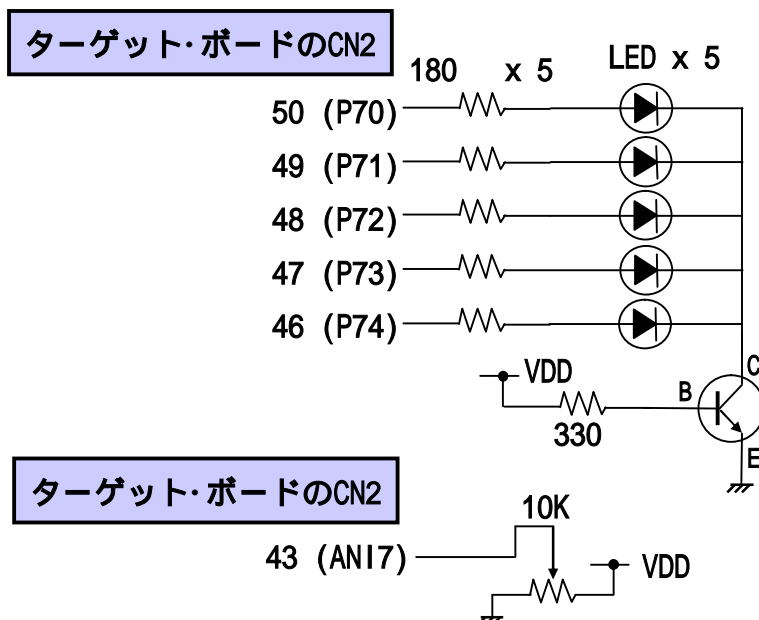
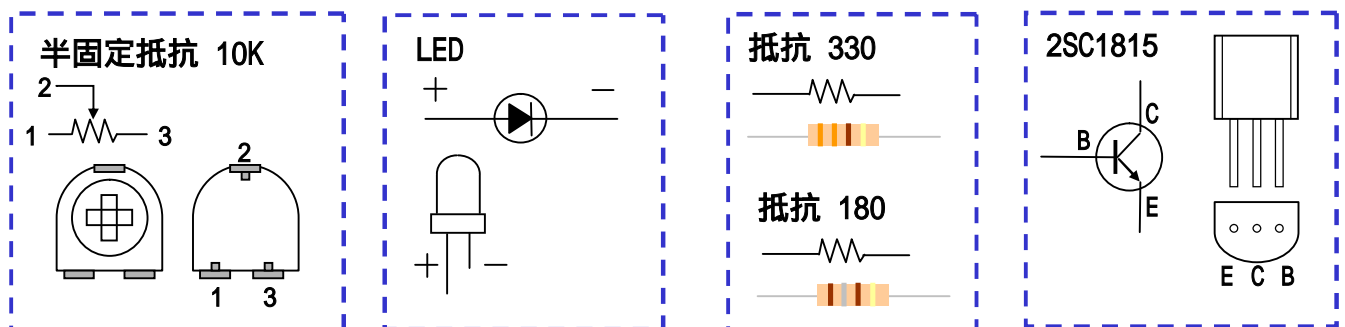
ターゲット・ボード体験編 (おわり)

ターゲット・ボード体験編は以上です。ターゲット・ボードにはSWとLEDが2つのシンプルな構成になっていますが、ソフトウェアを工夫して表示を変えてみてください。またターゲット・ボード上にはユニバーサル・エリアがありますので、ここに半固定抵抗、LED、トランジスタを搭載しても充分マイコンのポートで制御可能です。

次ページよりターゲット・ボードを使った「マイコン・プログラミング体験編」が始まります。マイコンにプログラミングを行うとターゲット・ボードへ電源を供給するだけで動作します。電池ボックスをつければ電池駆動も可能になります。

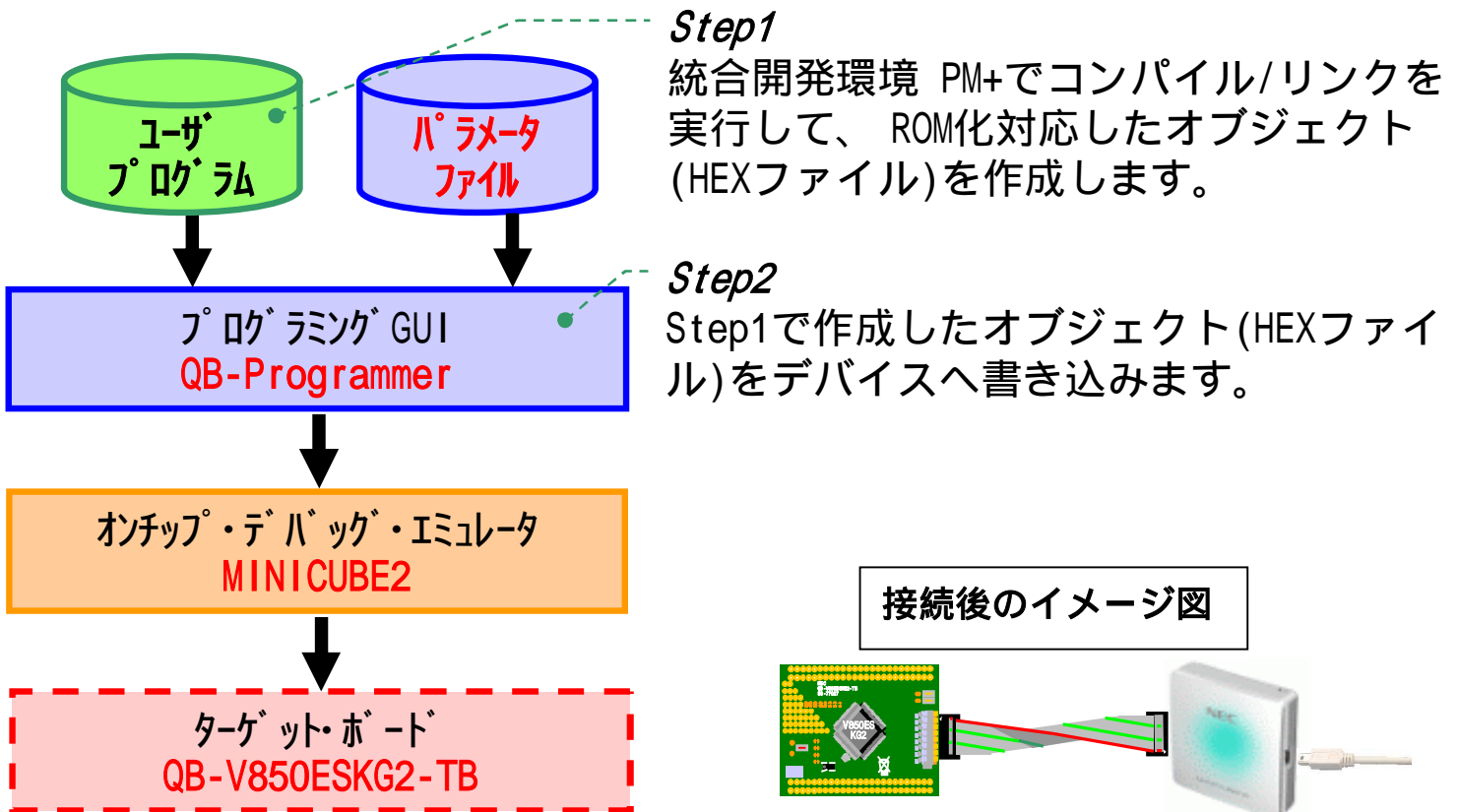
下図はターゲット・ボードに半固定抵抗とLED、トランジスタを追加した例です。

- ・半固定抵抗をAN17へ接続
- ・LEDをP70～P74へ接続



マイコン・プログラミング体験編

マイコン・プログラミング体験編ではターゲット・ボード(マイコン)にプログラミングするまでの手順を紹介します。



次ページより *Step1, Step2* でプログラミング手順を説明します

マイコン・プログラミング体験編 (Step1)

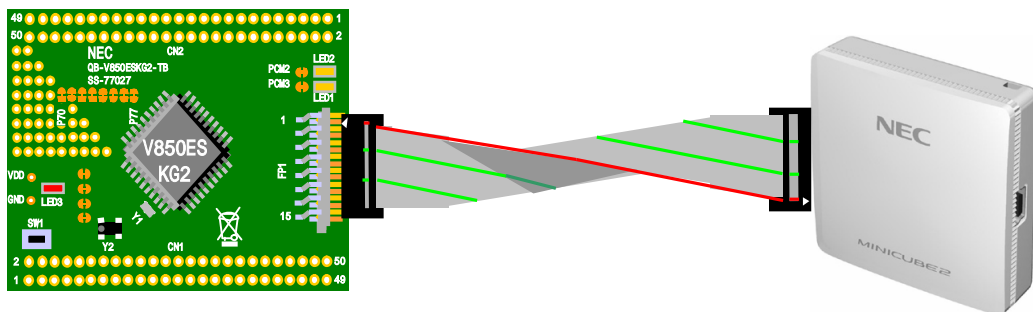
Step1 統合開発環境 PM+でコンパイル/リンクを実行して、ROM化対応したオブジェクト(HEXファイル)を作成します。

a. HEX形式のファイルを作成します

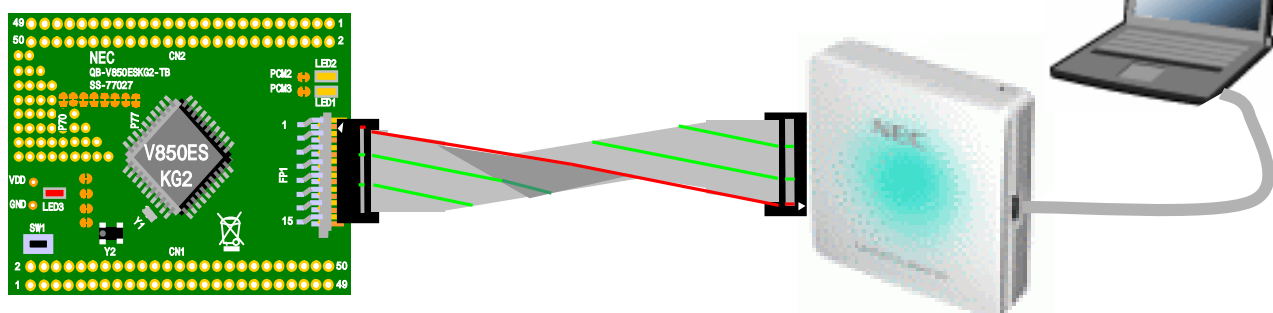
PM+のメニュー・バーの[ツール(T)] [ヘキサコンバ -タオプションの設定(O)...]を選択します。ヘキサコンバ -タオプションの設定」のファイルタブで”使用する(U)”にチェックされていることを確認してください。出力ファイル名は空欄でOKです。空欄にすると”romp.hex”が作成されます。



b. MINICUBE2とターゲット・ボードを接続します



c. MINICUBE2をPCへ接続します

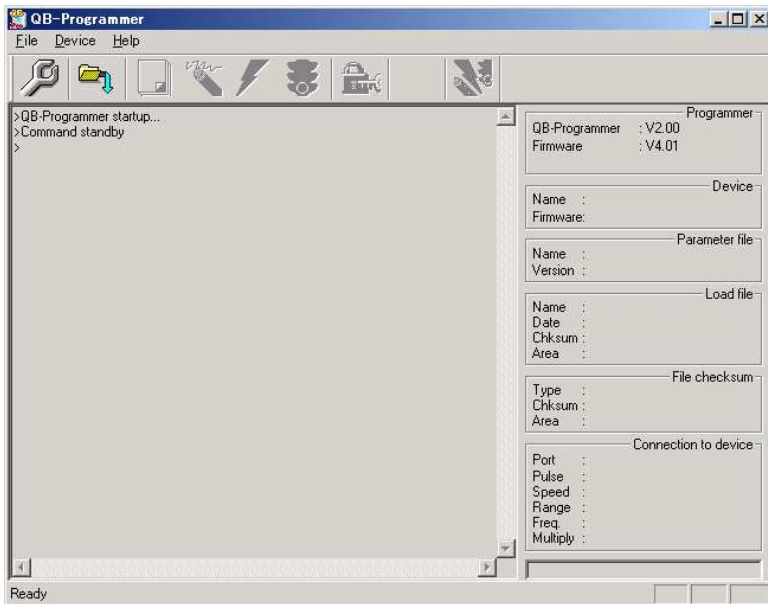


マイコン・プログラミング体験編 (Step2-1)

Step2 Step1で作成したオブジェクト(HEXファイル)をデバイスへ書き込みます。(画面はQB-Programmer V2.00の場合です)

a. QB-Programmerを起動します。

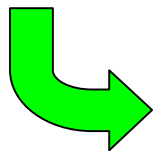
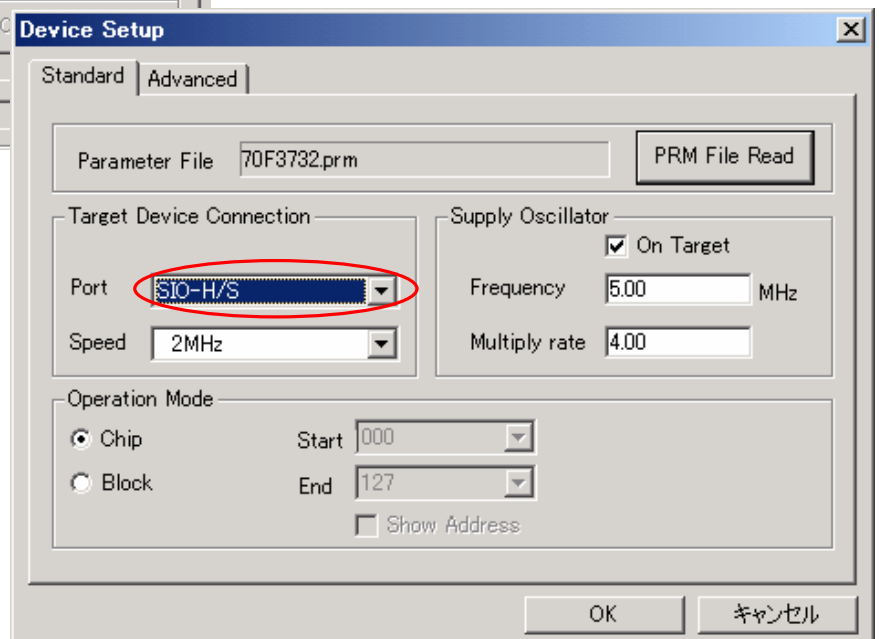
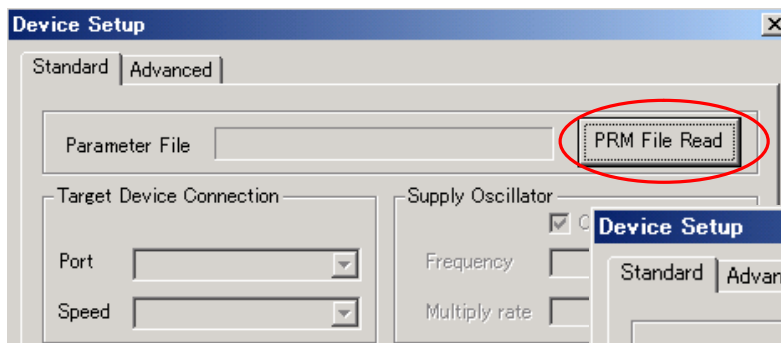
ID850QBを起動している場合は終了してください



MINICUBE2のLEDはアイドルモードより緑点灯へ変わります

b. パラメータ・ファイルをロードします

メニュー・バー[Device] [Setup...] [PRM File Read]ボタンを押下します。ロードするファイルは70F3732.prmです。



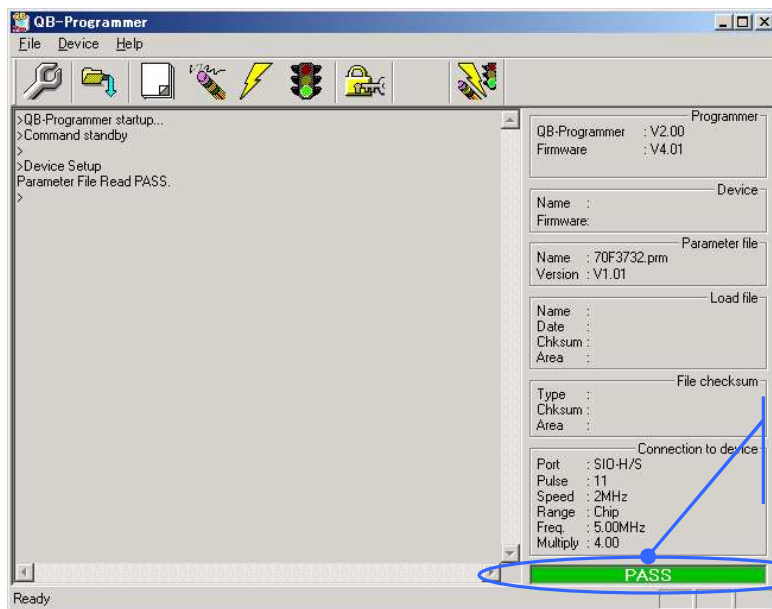
パラメータ・ファイルをロードすると各種の設定が行われます。「Port」のインターフェースを“SIO-H/S”へ変更します。

マイコン・プログラミング体験編 (Step2-2)

c. QB-Programmerの画面を確認します

Device Setupに成功すると下記のメッセージが表示されます。

```
>Device Setup  
Parameter File Read PASS.
```



PASSが表示され、エラーのないことを確認してください

d. HEXファイルをロードします

メニュー・バー[File] [Load...]でロードするファイルを指定します。“ romp.hex ”があるフォルダを指定してロードしてください。HEXファイルのロードに成功すると下記のメッセージが表示されます。

```
>Open Load File....  
Success read Load file.
```

e. デバイスを消去します

メニュー・バー[Device] [Erase...]でデバイスを消去します。デバイスの消去に成功すると下記のメッセージが表示されます。(ブロック0に書き込まれていた場合)

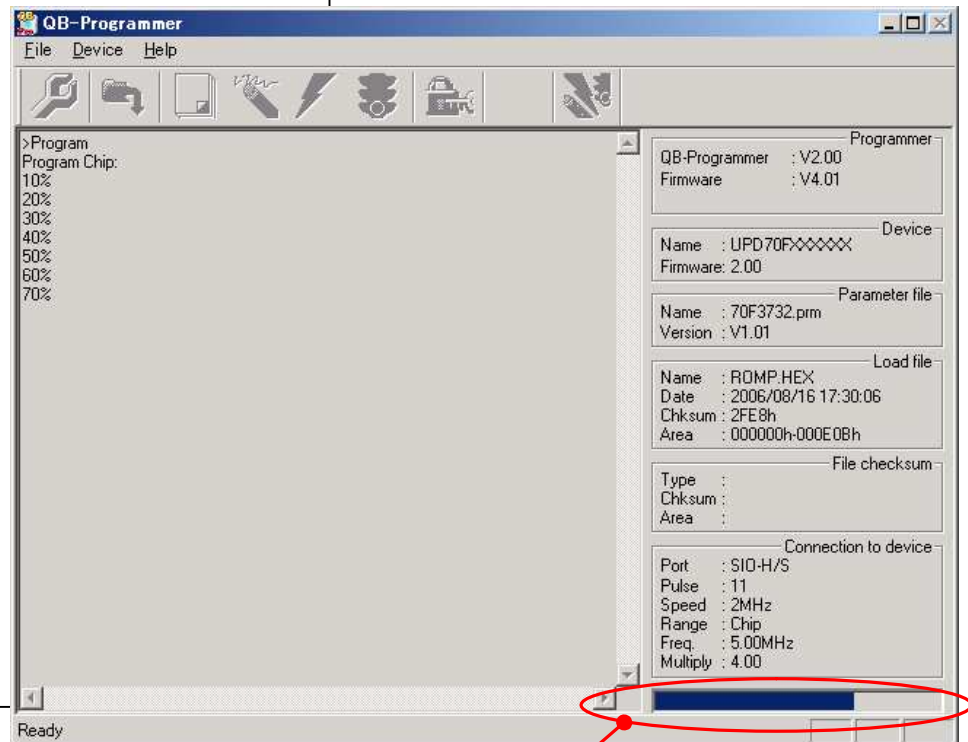
```
>Erase  
Blank check Chip: Not blank, Erase need.  
Erasing...  
Erasing Chip:PASS  
Erase PASS  
>
```

マイコン・プログラミング体験編 (Step2-3)

f. デバイスにプログラミング(書き込み)します

メニュー・バー [Device] [Program...] でデバイスにプログラミングします。プログラミング(書き込み)に成功すると下記のメッセージが表示されます。時間は10秒ほどで終了します。

```
>Program
Program Chip:
10%
20%
30%
40%
50%
60%
70%
80%
90%
100%
PASS
Set Security Flags
Program PASS
>
```

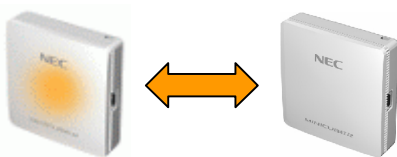


書き込み中にプログレスバーが表示されます。

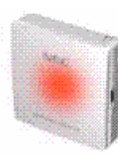
💡ワンポイント

AutoプログラミングとMINICUBE2のLEDについて

デバイスの消去/書き込み/ベリファイ連の動作を1つのコマンドで行う事もできます。メニューより [Device] [Autoprocedure (EPV)] を選択してください。時間は全体で20秒ほどで終了します。また、デバイスの消去/書き込み/ベリファイの処理中はMINICUBE2のLEDが黄色点滅します。各処理にエラーがあった場合MINICUBE2のLEDは赤が点灯します。



各種コマンド実行中は黄色点滅



各種コマンドでエラーが発生した場合は赤点灯

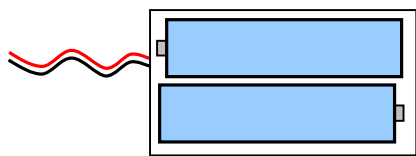
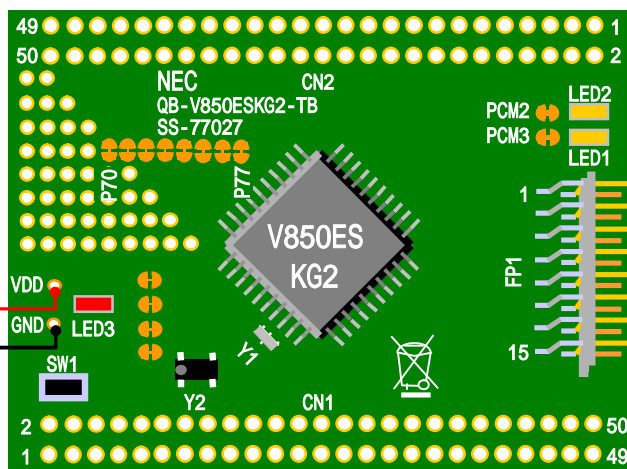


各種コマンドが正しく終了した場合は緑点灯

マイコン・プログラミング体験編 (おわり)

マイコン・プログラミング体験編は以上です。プログラミングしたターゲット・ボードは電源を供給すれば動作します。USBポートより電源のみを供給するケーブルを自作するか、または乾電池を3本直列で供給してもよいでしょう。

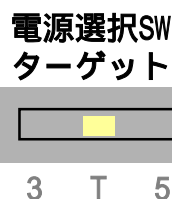
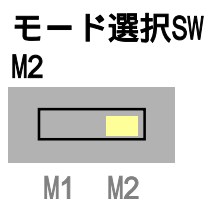
次ページよりターゲット・ボードを使った「ターゲット・システム作成例」が始まります。ドットマトリクスLEDを使用した回路例を紹介しています。



動作電圧が2.7V～5.5Vまでなので乾電池なら2～3本の使用します。それ以上の乾電池を使うのなら3端子レギュレータなどを使用して電圧を降下させる必要があります。

また、外部電源を利用してMINICUBE2でオンチップデバッグする際はSW設定に注意してください。

MINICUBE2の電源選択SWは必ず“T”に設定します



MINICUBE2

ターゲット・システム作成例A (その1)

この章ではターゲット・システム作成例を紹介します。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Bでは電光掲示板プログラムを流用した簡単なゲームを作成しています。

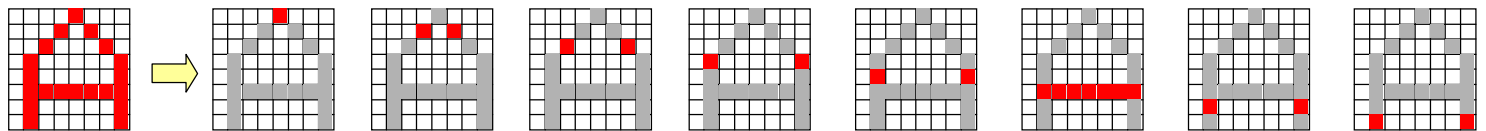
ターゲット・ボードを使用した回路を作成します。ここでは8x8のドットマトリクスLEDをダイナミック点灯させます。ダイナミック点灯とは表示を分割して(今回は8個のLED)行う方法です。表示を高速に順次繰り返す事によって人の目には全てが表示されているように見えます。

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: スタティック点灯に比べて表示が暗い。

人の目に見える表示

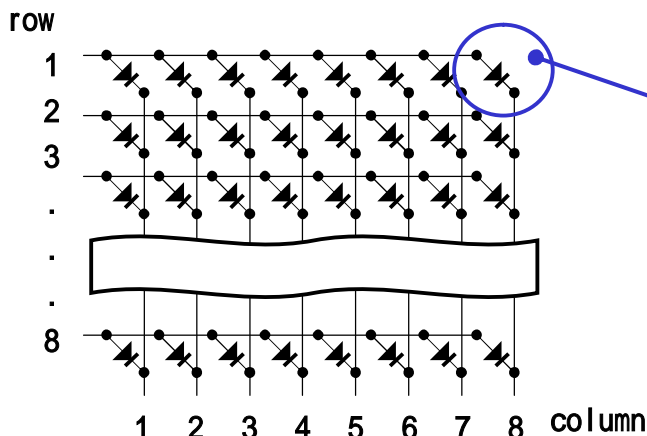
高速に表示を繰り返しています



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。

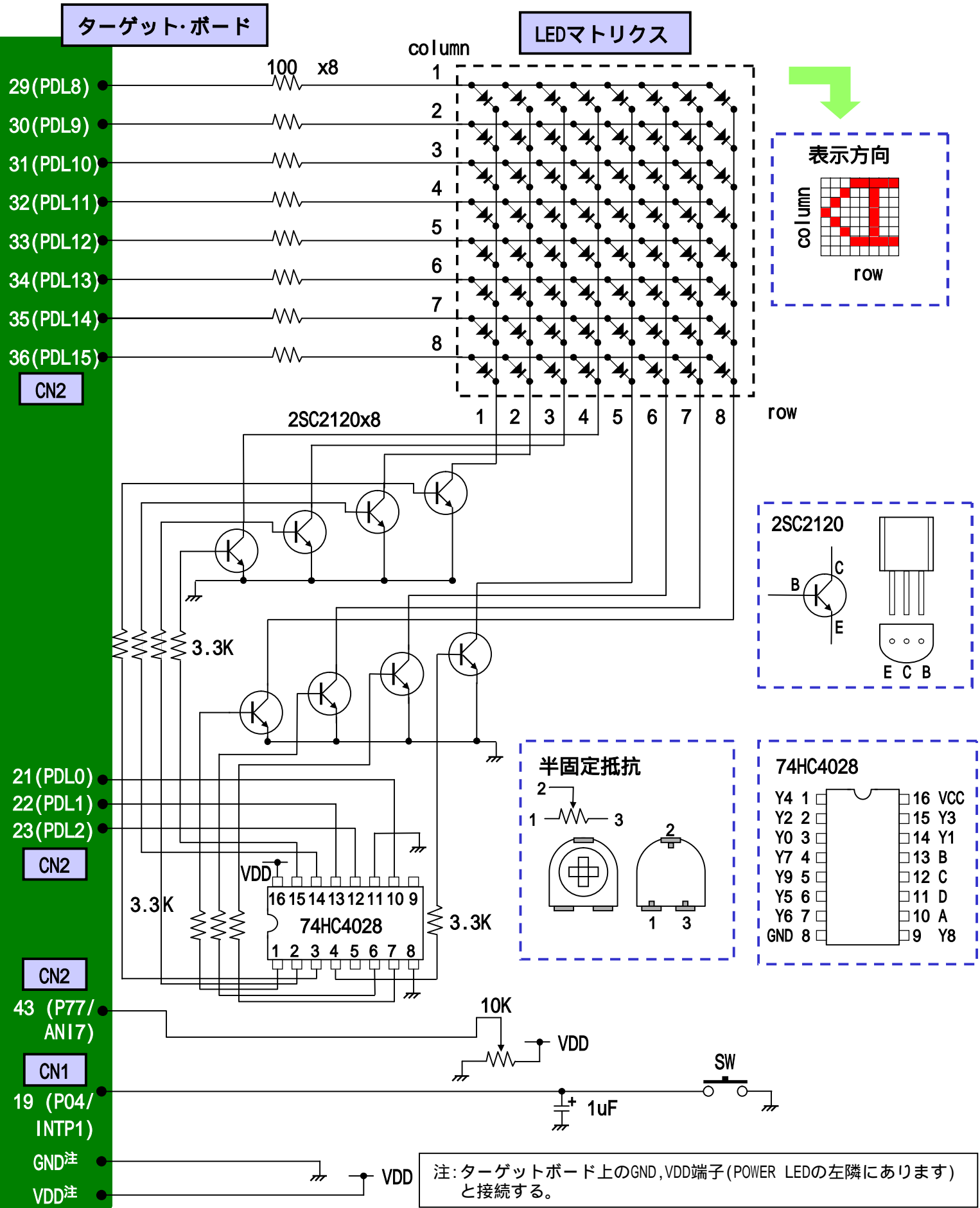


アノード(+)

カソード(-)

例えば四角内のLEDを点灯させるにはrow1に+, column8に-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。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。

ターゲット・システム作成例A (その2)



ターゲット・システム作成例A (その3)

a. 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でドライブします。

74HC4028はBCDコードを10進化します。ポートPDL0～PDL2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(ANI7)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用しますのでコンデンサのみ使用します)

電源はMINICUBE2より供給します。ターゲットボードのGND,VDDと接続してください。

ワンポイント

シンク電流、ソース電流について

V850ES/KG2のポートはソース電流で5mA(端子合計30mA)、シンク電流で10mA(端子合計30mA)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})

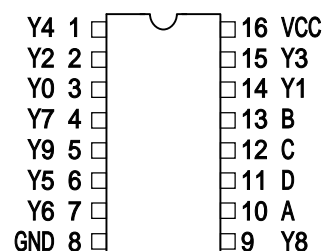


74HC4028について

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



ターゲット・システム作成例A (その4)

b. プログラム作成

システム設定 [ターゲットボード体験編(Step1-2)と同じ設定をします]

[基本設定]タブ

サブ・クロック設定エリアの ” 使用しない ” にチェック。ウォッチドッグ・タイマ2機能エリアの ” 使用しない ” にチェック。

[起動設定]タブ

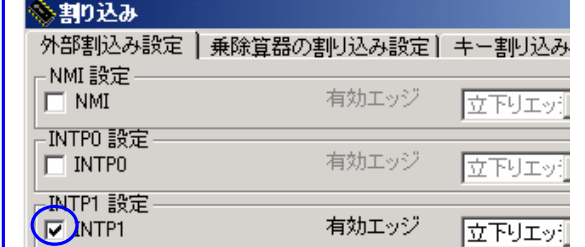
CPUクロック選択(MHz)メイン・クロック “ 20(fxx) ” を選択。

[オンチップディバグ設定]タブ

オンチップディバグ設定の “ 使用する ” にチェック。インサーキットエミュレータ選択は ” MINICUBE2 ” にチェック。MINICUBE2端子選択は ” CS10 ” 。RAM Monitor/DMM機能選択は ” 使用する ” にチェック。



割り込み設定



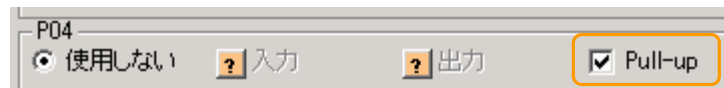
[外部割り込み設定]タブ

“ INTP1許可 ” にチェック

ポート設定

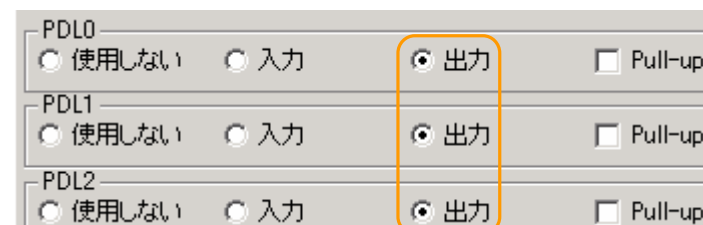
[ポート0]タブ

P04の ” Pull-up ” にチェック



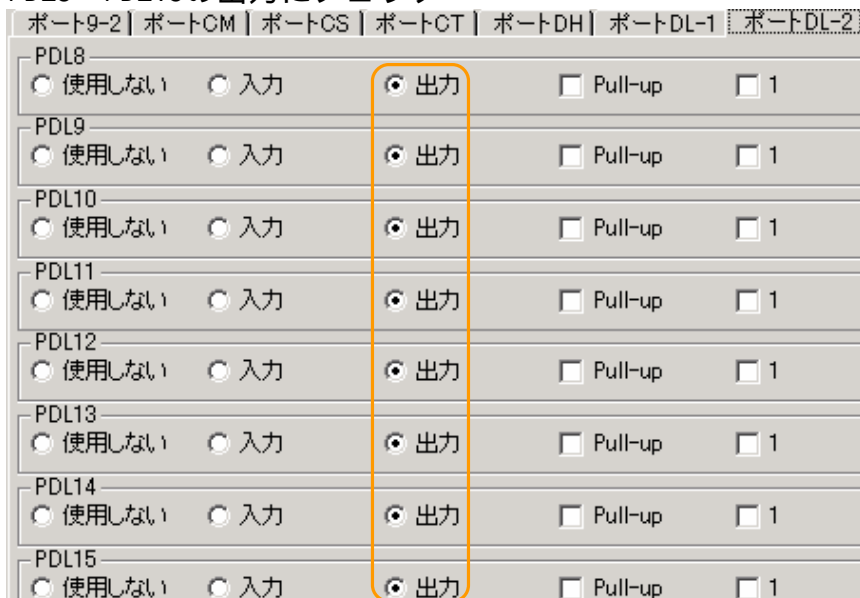
[ポートDL-1]タブ

PDL0 ~ PDL2の ” 出力 ” にチェック



[ポートDL-2]タブ

PDL8 ~ PDL15の出力にチェック



ワンポイント

Applilet起動について

ターゲットボード体験編(Step1-1)「a. Appliletを起動します」を参照してプロジェクトを作成してください。ここではプロジェクト名を「extend」、チップシリーズ名「V850ESKG2」、チップ名「μPD70F3732」で作成しています。

ターゲット・システム作成例A (その5)

A/Dコンバータ設定

「モード設定」

“使用する” にチェック。

「トリガ・モードを指定」

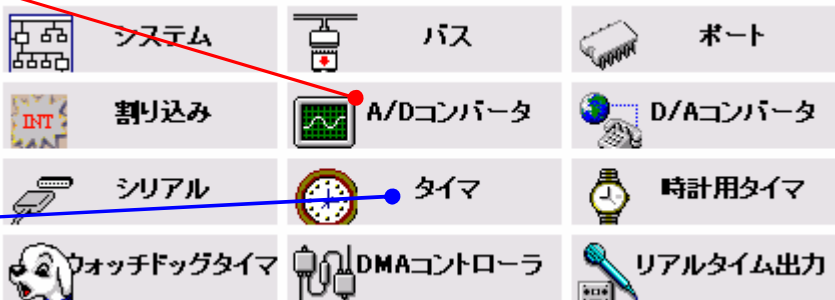
“ソフトウェアトリガ・モード” にチェック。

「動作モードを指定」

“セレクト・モード” にチェック
“アナログ入力チャンネル選択” を ANI7 にする。

「割り込み設定」

“A/Dの割り込み許可” にチェック。
“変換終了ごとに割り込み” にチェック。



タイマ設定

[タイマP0]タブ

“インターバル・タイマ” にチェックして “詳細” を押下します。[TMP0インターバル・タイマ]ダイアログが表示されます。設定単位を “msec”、インターバル時間を “1” にします。また、「割り込み設定」エリアで “TMP0とCCR0の一致で割り込み発生” にチェックします。

[タイマ00]タブ

タイマP0と同様に “インターバル・タイマ” にチェックします。

[TM00インターバル・タイマ]ダイアログでは、設定単位を “msec”、インターバル時間を “1” にします。「割り込み設定」エリアで “設定した周期毎に通知する” にチェックします。

ターゲット・システム作成例A (その6)

プログラム作成

下記に示す青字のコードを追加してください

main.c(リスト1)

リスト省略

```
#include "macrodriver.h"
#include "ad.h"
#include "int.h"
#include "port.h"
#include "timer.h"
/*
*****
** MacroDefine
*****
*/

/* ----- */
#pragma section sconst begin

extern const UCHAR g_font_numeric[] =
{ /* font data 0 - 9 */
  0x00, 0x0E, 0x11, 0x13, 0x15, 0x19, 0x11, 0x0E
, 0x00, 0x04, 0x0C, 0x04, 0x04, 0x04, 0x04, 0x0E
, 0x00, 0x0E, 0x11, 0x01, 0x0E, 0x10, 0x10, 0x1F
, 0x00, 0x0E, 0x11, 0x01, 0x0E, 0x01, 0x11, 0x0E
, 0x00, 0x02, 0x06, 0x0A, 0x12, 0x1F, 0x02, 0x02
, 0x00, 0x1F, 0x10, 0x10, 0x1E, 0x01, 0x01, 0x1E
, 0x00, 0x0F, 0x10, 0x10, 0x1E, 0x11, 0x11, 0x0E
, 0x00, 0x1F, 0x01, 0x02, 0x04, 0x04, 0x04, 0x04
, 0x00, 0x0E, 0x11, 0x11, 0x1E, 0x11, 0x11, 0x0E
, 0x00, 0x0E, 0x11, 0x11, 0x0F, 0x01, 0x01, 0x0E
};

extern const UCHAR g_font_alphabet[] =
{ /* font data A - Z */
  0x00, 0x04, 0x0A, 0x0A, 0x11, 0x1F, 0x11, 0x11
, 0x00, 0x1E, 0x11, 0x11, 0x1E, 0x11, 0x11, 0x1E
, 0x00, 0x0E, 0x11, 0x10, 0x10, 0x10, 0x11, 0x0E
, 0x00, 0x1C, 0x12, 0x11, 0x11, 0x11, 0x12, 0x1C
, 0x00, 0x1F, 0x10, 0x10, 0x1E, 0x10, 0x10, 0x1F
, 0x00, 0x1F, 0x10, 0x10, 0x1E, 0x10, 0x10, 0x10
, 0x00, 0x0E, 0x11, 0x10, 0x17, 0x11, 0x11, 0x0E
, 0x00, 0x11, 0x11, 0x11, 0x1F, 0x11, 0x11, 0x11
, 0x00, 0x0E, 0x04, 0x04, 0x04, 0x04, 0x04, 0x0E
, 0x00, 0x02, 0x02, 0x02, 0x02, 0x02, 0x12, 0x0C
, 0x00, 0x11, 0x12, 0x14, 0x18, 0x14, 0x12, 0x11
, 0x00, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x1F
, 0x00, 0x11, 0x11, 0x1B, 0x15, 0x11, 0x11, 0x11
, 0x00, 0x11, 0x11, 0x19, 0x15, 0x13, 0x11, 0x11
, 0x00, 0x0E, 0x11, 0x11, 0x11, 0x11, 0x11, 0x0E
, 0x00, 0x1E, 0x11, 0x11, 0x1E, 0x10, 0x10, 0x10
, 0x00, 0x0E, 0x11, 0x11, 0x11, 0x15, 0x13, 0x0F
, 0x00, 0x1E, 0x11, 0x11, 0x1E, 0x14, 0x12, 0x11
, 0x00, 0x0E, 0x11, 0x10, 0x0E, 0x01, 0x11, 0x0E
, 0x00, 0x1F, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04
, 0x00, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x0E
, 0x00, 0x11, 0x11, 0x11, 0x11, 0x0A, 0x0A, 0x04
, 0x00, 0x11, 0x15, 0x15, 0x15, 0x15, 0x15, 0x0A
, 0x00, 0x11, 0x11, 0x0A, 0x04, 0x0A, 0x11, 0x11
, 0x00, 0x11, 0x11, 0x0A, 0x04, 0x04, 0x04, 0x04
, 0x00, 0x1F, 0x01, 0x02, 0x04, 0x08, 0x10, 0x1F
};
```

main.c(リスト2)

```
extern const UCHAR g_textdata[][ D_MAXTEXT ] =
{ /* text data */
  "THIS IS V850ESKG2 TARGET BOARD  "
, "DISPLAY CHANGES IF SW IS PUSHED  "
, "THIS IS PROGRAM SAMPLE OF NECEL  "
, "abcdefghijklmnopqrstuvwxyz  "
, "0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6"
, 0
};
#pragma section sconst end

/* ----- */
/* ---- prototype ---- */
void initialize_text( void );
/* font_ is displayed to virtual vram. */
void disp_putfont( UCHAR font_ );

/* ----- */
/* ---- global value ---- */
/* Interval timer counter */
UINT g_timer0_counter = 0;
UINT g_timer1_counter = 0;

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

/* Matrix LED vram */
UCHAR g_matrix_ram[ D_MLED_CNT ][ D_MLED_ROW ];

/* ----- */
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 );
}
```

ターゲット・システム作成例A (その7)

プログラム作成

下記に示す青字のコードを追加してください

main.c(リスト3)

```

/* ----- */
/* font_ is displayed to virtual vram. */

void disp_putfont( UCHAR font_ )
{
    int i;
    UCHAR cnvf, fnt;

    cnvf = (UCHAR)(toupper( font_ ));
    if ( cnvf>='A' && cnvf<='Z' )
    { /* Alphabet font */
        cnvf -= 'A';
        for ( i=0; i<D_MLED_ROW; i++ )
        {
            fnt = g_font_alphabet[ cnvf*8 + i ];
            g_matrix_ram[ 1 ][ i ] = (fnt << 2);
        }
    }
    else if ( cnvf>='0' && cnvf<='9' )
    { /* Numeric font */
        cnvf -= '0';
        for ( i=0; i<D_MLED_ROW; i++ )
        {
            fnt = g_font_numeric[ cnvf*8 + i ];
            g_matrix_ram[ 1 ][ i ] = (fnt << 2);
        }
    }
    else
    { /* Null font */
        for ( i=0; i<D_MLED_ROW; i++ )
        {
            g_matrix_ram[ 1 ][ i ] = 0;
        }
    }
}

/* ----- */
/* 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;
    }
}

```

main.c(リスト4)

```

void main( void )
{
    /* initialize */
    initialize_value();
    initialize_text();

    /* TB-board LED off */
    PCM.2 = 1;
    PCM.3 = 1;

    /* start timer */
    TMPO_Start();
    TMOO_Start();
    __EI();

    while(1 ){
        ;
    }
}

```

macrodrive.h(リスト1)

リスト省略

```

#define SYSTEMCLOCK 20000000
#define SUBCLOCK 32768
#define MAINCLOCK 5000000
#define FxInuse 0

#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 */

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

#endif

```



ワンポイント

systeminit.cの_rcopy(&S_romp, -1)について

Appliletが生成するソースファイル

「systeminit.c」で初期化を行う関数

void SystemInit(void)にあるコード

_rcopy(&S_romp, -1) ですが

プログラムをROM化する際に必要なコードです。

ROM化での設定方法については、CA850のユーザーズ・マニュアル操作編「6.4 ROM 化用オブジェクトの作成」を参照してください。

ターゲット・システム作成例A (その8)

プログラム作成

下記に示す青字のコードを追加してください

timer_user.c(リスト1)

リスト省略

```
#include "macrodriver.h"
#include "timer.h"
#include "ad.h"

#pragma interrupt INTTPOCCO MD_INTTPOCCO
#pragma interrupt INTTM000 MD_INTTM000

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

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 ];

/*
** -----
** Abstract:
** This function is TMO0 INTTPOCCO interrupt service
routine.
**
** Parameters:
** None
**
** Returns:
** None
** -----
*/
__interrupt void MD_INTTPOCCO( void )
{
    /*TODO.Add user defined interrupt service routine*/
    UCHAR row, line;

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

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

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

    g_timer0_counter++;
}

/*
** -----

```

timer_user.c(リスト2)

```
** Abstract:
** TMP1 INTTM000 interrupt service routine.
**
** Parameters:
** None
**
** Returns:
** None
** -----
*/
__interrupt void MD_INTTM000( void )
{
    /*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;
    }
}

```


ターゲット・システム作成例A (その9)

プログラム作成

下記に示す青字のコードを追加してください

ad_user.c(リスト1)

```

/*
*****
** Include files
*****
*/
#include "macrodriver.h"
#include "ad.h"
#pragma interrupt INTAD MD_INTAD

extern USHORT g_text_speed;

/*
-----
**
** Abstract:
** INTAD Interrupt service routine.
**
** Parameters:
** None
**
** Returns:
** None
**
-----
*/
__interrupt void MD_INTAD( void )
{
/*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;
  }
}

```

int_user.c(リスト1)

```

/*
*****
** Include files
*****
*/
#include "macrodriver.h"
#include "int.h"
#pragma interrupt INTP1 MD_INTP1

extern void initialize_text( void );

extern UCHAR g_text_sw;
extern UCHAR g_matrix_ram[D_MLED_CNT][D_MLED_ROW];
extern const UCHAR g_textdata[][ D_MAXTEXT ];

/*
*****
** MacroDefine
*****
*/

/*
-----
**
** Abstract:
** INTP1 Interrupt service routine.
**
** Parameters:
** None
**
** Returns:
** None
**
-----
*/
__interrupt void MD_INTP1( void )
{
/*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();
}

```

ターゲット・システム作成例A (その10)

プログラム説明

プログラムリストを元に説明します

main.c

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

マトリクス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_font_alphabet[] /* font data A - Z */`

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

タイマ割り込み時に+1されるカウンタです

`UINT g_timer0_counter = 0;`
`UINT g_timer1_counter = 0;`

表示するテキストを選択します

`UCHAR g_text_sw = 0;`

表示されているテキストが何文字目かを示します

`UCHAR g_text_cnt = 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

`UINT g_text_scrollcnt = 0;`

スクロールスピードを示します

`USHORT g_text_speed = 0;`

マトリクス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_matrix_ram[D_MLED_CNT][D_MLED_ROW];`

変数初期化を行う関数です

`void initialize_value(void)`
`void initialize_text(void)`

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

`void disp_putfont(UCHAR font_)`

マトリクスLEDの表示を1ドット移動(スクロール)します

`void disp_move1dot(void)`

timer_user.c

1msec毎に呼ばれる関数です。マトリクス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_INTTPOCCO()`

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。

`__interrupt void MD_INTTM000()`

ad_user.c

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

`__interrupt void MD_INTAD(void)`

int_user.c

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。

`__interrupt void MD_INTP1(void)`

macrodriver.h

`#define D_MLED_CNT 2 /* Matrix LED number */`
 マトリクスLEDの数2つあるのは1つはバッファ領域として使用しているため。先頭が実際の表示領域。

`#define D_MLED_ROW 8 /* Matrix LED row */`
 マトリクスLEDの列の数。

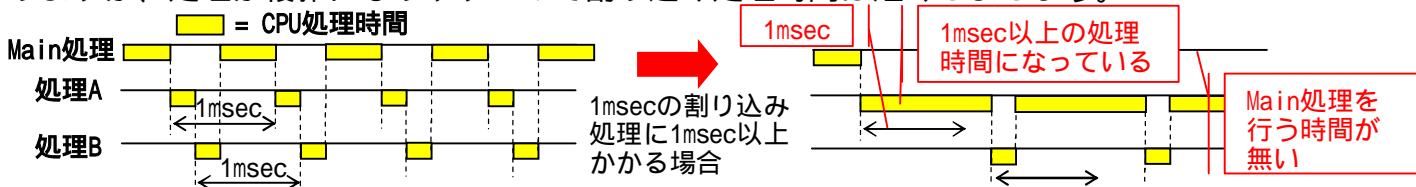
`#define D_FONT_WIDTH 6 /* font width */`
 表示するフォントの幅(ドット数)。

`#define D_MAXTEXT 34 /* Max display of text */`
 スクロールして表示するテキストの文字数。

ワンポイント

割り込み処理について

割り込みの処理時間は短くするのが基本です。1msecの割り込みがA,Bとあった場合Aの処理に1msec以上かかるとBの処理が待たされてしまいます。それを避けるために多重割り込みという方法もありますが、処理が複雑になりやすいので割り込み処理時間は短くしましょう。



ターゲット・システム作成例A (その11)

動作概要

プログラムの動作として、下記の5つに分けられます

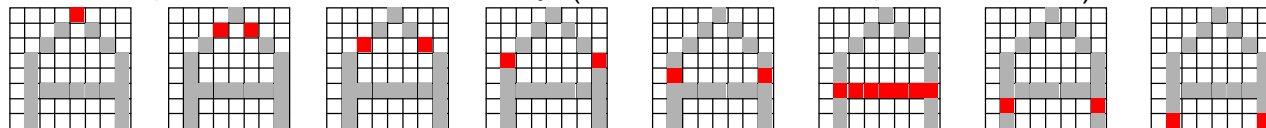
1. メイン処理
2. タイマP0(インターバル・タイマとして使用)
3. タイマ00(インターバル・タイマとして使用)
4. A/Dコンバータ完了割り込み
5. INTP1外部割り込み

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P0/P1を開始します。
処理としては電源ON後に1度しか動きません。
後はwhileで無限ループとなります。

2. タイマP0 `__interrupt void MD_INTTPOCC0()`

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00 `__interrupt void MD_INTTM000()`

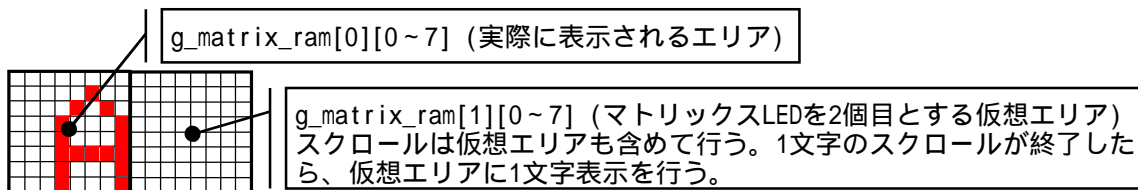
1msec毎に起動します。

A/Dコンバートの開始を行います。A/Dコンバートが完了すると割り込みが発生します。

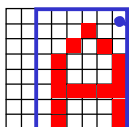
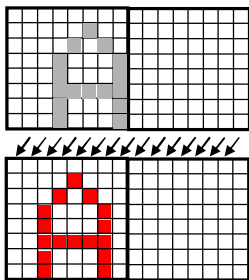
A/Dコンバートの結果は `g_text_speed` へ反映されるので、それを元にスクロールスピードを決めます
タイマ01は呼ばれる毎に `g_timer1_counter` を+1していますので、`g_text_speed`と`g_timer1_counter`
の値を比較してスクロールを行うか行わないかを判断しています。

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

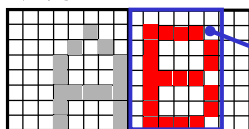


a. マトリックスLEDへ表示されているデータを1ドット左へ移動する



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。

ターゲット・システム作成例A (その12)

4. A/Dコンバータ完了割り込み `__interrupt void MD_INTAD(void)`

タイマP1で開始されたA/D変換が完了した場合に呼ばれます。

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。

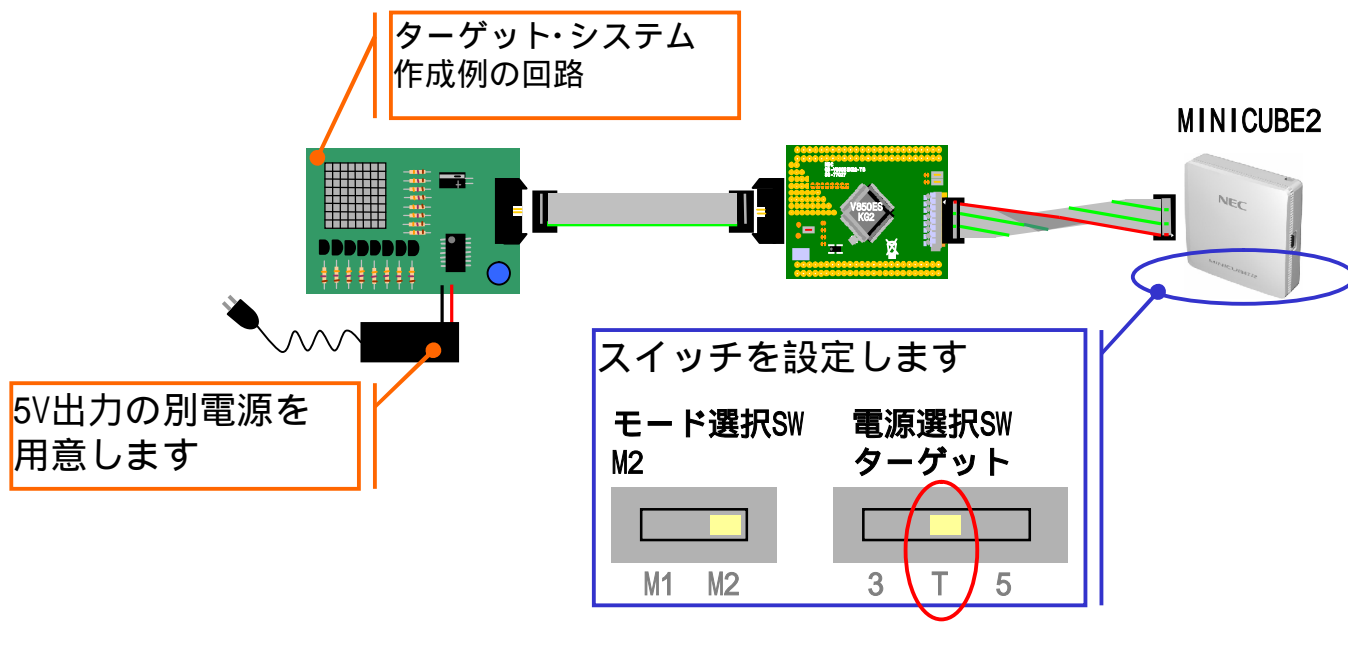
スクロールするテキストを変えます。表示させるテキストは1文が32文字で構成されています。

`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で構成してください。

💡 ワンポイント

電源供給について

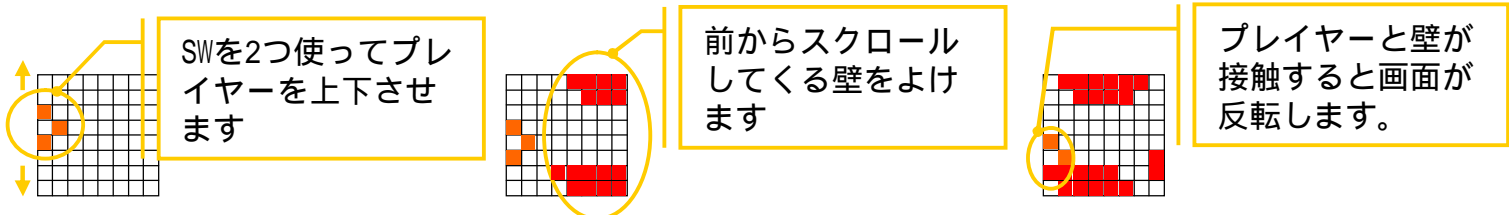
MINICUBE2より電源供給する場合5V供給で最大定格100mAです。ターゲット・システム作成例の回路で使用する部品によっては100mAを超える場合があります。その場合は5V電源を別に用意してください。ターゲット・システム側の電源を使用する場合はMINICUBE2の「電源選択SW」を「T」に設定してください。



ターゲット・システム作成例B (その1)

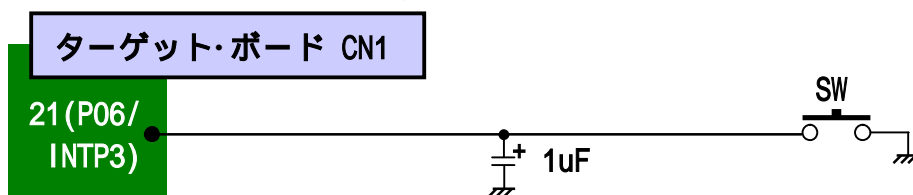
ターゲット・システム作成例BではドットマトリクスLEDを使用した電光掲示板の回路を応用した簡単なゲームの作成例を紹介します。

電光掲示板の回路にSWを1つ追加して簡単なスクロールゲームを作成します。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。



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にし、ゲームの難易度を調整できるようにします。プレイヤーと壁は同じ色で表示しますが、プレイヤーは点滅するので壁と判別可能です。

電光掲示板回路にスイッチを追加します



a. Appliletの設定を追加します

Appliletの設定ですが、ターゲット・システム作成例A(その4)、(その5)と同様の設定を行います。更に下記の設定を追加します

システム

バス

割り込み

ポート

シリアル

A/Dコンバータ

割り込み設定
[外部割り込み設定]タブ
“INTP1許可”, “INTP3許可” にチェック
有効エッジを“両エッジ”へ変更します
“両エッジ”へ設定すると、SWを押下した時とSW離したときに割り込みが発生するようになります。

-INTP1 設定		有効エッジ	両エッジ
<input checked="" type="checkbox"/> INTP1			
-INTP2 設定		有効エッジ	立下りエッジ
<input type="checkbox"/> INTP2			
-INTP3 設定		有効エッジ	両エッジ
<input checked="" type="checkbox"/> INTP3			

ポート設定

[ポート0]タブ
P04, P06の”Pull-up”にチェック
INTP1, INTP3の割り込み端子をプルアップします。

P04	<input checked="" type="radio"/> 使用しない	<input type="radio"/> 入力	<input type="radio"/> 出力	<input checked="" type="checkbox"/> Pull-up
P05	<input checked="" type="radio"/> 使用しない	<input type="radio"/> 入力	<input type="radio"/> 出力	<input type="checkbox"/> Pull-up
P06	<input checked="" type="radio"/> 使用しない	<input type="radio"/> 入力	<input type="radio"/> 出力	<input checked="" type="checkbox"/> Pull-up

ターゲット・システム作成例B (その2)

b. ソースコードの修正を行います

下記に示す青字のコードを追加してください

main.c(リスト1)

リスト省略

```
#include "macrodriver.h"
#include "ad.h"
#include "int.h"
#include "port.h"
#include "timer.h"

/*
*****
** MacroDefine
*****
*/

/* ----- */
#pragma section sconst begin

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,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x03,0xc0,0x00,0x00,0x00,0x07,0xf8,0x00,
    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,
    0x20,0x00,0x00,0x00,0x40,0x00,0x00,0x41,
```

main.c(リスト2)

```
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,
0x00,0x00,0xc0,0x01,0x86,0x00,0xc0,0x60,
0x0c,0x03,0x00,0x60,0x0c,0x00,0x03,0xe0,
0x07,0xe0,0x00,0xc0,0x07,0xe0,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x20,
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,
0xe0,0x00,0x40,0x00,0x00,0x00,0x40,0x04,
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,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x3f,
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,
0x00,0x00,0x00,0x00,0x00,0x06,0x00,0x00,
0x00,0x00,0x00,0x80,0x00,0x40,0x00,0x80,
0x00,0x00,0x00,0x00,0x02,0x00,0x00,0x11,
0x01,0x01,0x00,0x00,0x10,0x00,0x04,0x04,

0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x06,
0x00,0x00,0x00,0x01,0xff,0xff,0xff,0xff,
0xf8,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x01,0x80,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x06,0x03,0x00,0x07,0x80,0x00,
0x00,0x00,0xe0,0x00,0x00,0x00,0x00,0x7f,
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,
0x00,0x00,0x00,0x01,0x00,0x02,0x00,0x00,
0x00,0x02,0x00,0x04,0x00,0x01,0x00,0x00,
0x40,0x00,0x08,0x00,0x00,0x00,0x40,0x00,
```


ターゲット・システム作成例B (その3)

c. ソースコードの修正を行います

下記に示す青字のコードを追加してください

main.c(リスト3)

```

0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x08,0x02,
0x00,0x04,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x1f,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0x80,0x00,0x00,0x00,
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,
0x00,0xc2,0x00,0x00,0x00,0x00,0x00,0x00,
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,
0x00,0xff,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,0xc0,0x00,0x00,0x00,0x3f,
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,

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,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,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x02,
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,
0x00,0x00,0x00,0x00,0x08,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x04,0x02
};

#pragma section sconst end
/* ----- */

```

main.c(リスト4)

```

/* 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 g_intp3_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_realram[ D_MLED_ROW ];
UCHAR g_plane_location;
BOOL g_plane_bump;

/* ----- */
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 = MD_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;
    }
}

```

ターゲット・システム作成例B (その4)

d. ソースコードの修正を行います

下記に示す青字のコードを追加してください

main.c(リスト5)

```

/* ----- */
/* 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;
    }
}

/*
**-----
**
** Abstract:
**     main function
**
** Parameters:
**     None
**
** Returns:
**     None
**-----
*/
void main( void )
{
    /* initialize */
    initialize_value();

    /* TB-board LED off */
    PCM.2 = 1;
    PCM.3 = 1;

    /* start timer */
    TMPO_Start();
    TMOO_Start();
    __EI();

    while (1)
    {
        ;
    }
}

```

macrodrive.h(リスト1)

```

                                リスト省略
#define SYSTEMCLOCK 2000000
#define SUBCLOCK 32768
#define MAINCLOCK 500000
#define FxInuse 0

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

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

#endif

```

ad_user.c(リスト1)

```

/*
*****
** Include files
*****
*/
#include "macrodriver.h"
#include "ad.h"

#pragma interrupt INTAD MD_INTAD

extern USHORT g_maze_speed;

/*
**-----
**
** Abstract:
**     INTAD Interrupt service routine.
**
** Parameters:
**     None
**
** Returns:
**     None
**-----
*/
__interrupt void MD_INTAD( void )
{
    USHORT adval;

    AD_Stop();

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

```

ターゲット・システム作成例B (その5)

e. ソースコードの修正を行います

下記に示す青字のコードを追加してください

timer_user.c(リスト1)

リスト省略

```
#include "macrodriver.h"
#include "timer.h"
#include "ad.h"

#pragma interrupt INTTPOCCO MD_INTTPOCCO
#pragma interrupt INTTMO00 MD_INTTMO00

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

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_intp3_sw;
extern UCHAR g_plane_location;
extern BOOL g_plane_bump;

/*
** -----
** Abstract:
** TMO INTTPOCCO interrupt service routine.
**
** Parameters:
** None
**
** Returns:
** None
** -----
*/
__interrupt void MD_INTTPOCCO( void )
{
    UCHAR row, line, pdat;

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

    /* display column */
    PDLH = 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 ) )

```

timer_user.c(リスト2)

```
{
    pdat = 0x80;
}
else
{
    pdat = 0;
}

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

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

g_timer0_counter++;
}

/*
** -----
** Abstract:
** TMP1 INTTMO00 interrupt service routine.
**
** Parameters:
** None
**
** Returns:
** None
** -----
*/
__interrupt void MD_INTTMO00( void )
{
    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;
                g_bump_counter--;
            }
            g_matrix_realram[ i ] = dat;
        }
    }
}
```

ターゲット・システム作成例B (その6)

f. ソースコードの修正を行います

下記に示す青字のコードを追加してください

timer_user.c(リスト3)

```

if ( g_bump_counter <= 0 )
{
    initialize_value();
}
}
else
{
    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 postion */
        if ( g_intp3_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];
}
}

```

int_user.c(リスト1)

リスト省略

```

#include "macrodriver.h"
#include "int.h"

#pragma interrupt INTP1 MD_INTP1
#pragma interrupt INTP3 MD_INTP3

/*
*****
** MacroDefine
*****
*/

extern UCHAR g_intp1_sw;
extern UCHAR g_intp3_sw;

/*
** -----
** Abstract:
** This function is INTP1 Interrupt service
routine.
**
** Parameters:
** None
**
** Returns:
** None
** -----
*/
__interrupt void MD_INTP1( void )
{
    g_intp1_sw = P0.4 ^ 1;
}

/*
** -----
** Abstract:
** This function is INTP3 Interrupt service
routine.
**
** Parameters:
** None
**
** Returns:
** None
** -----
*/
__interrupt void MD_INTP3( void )
{
    g_intp3_sw = P0.6 ^ 1;
}

```

ターゲット・システム作成例B (その7)

g. 迷路データ作成のヒント

`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、画像を作成します

- ・大きさを1000x8ドット
- ・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バイトを変換すれば実データになります。

ラインデータは逆にになってい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ください。

ターゲット・システム作成例B (その8)

h. プログラム説明

プログラムリストを元に説明します

main.c

タイマ割り込み時に+1されるカウンタです

```
UINT g_timer0_counter = 0;
UINT g_timer1_counter = 0;
```

衝突した時にLEDを点滅する時間です

```
UINT g_bump_counter = 0;
```

表示するテキストを選択します

```
UCHAR g_text_sw = 0;
```

表示されている壁が何ドット移動したかを示します

```
UINT g_maze_scrollcnt = 0;
```

壁のスクロール速度を示します

```
USHORT g_maze_speed = 0;
```

マトリクス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_matrix_ram[ D_MLED_CNT ][ D_MLED_ROW ];
UCHAR g_matrix_realram[ D_MLED_ROW ];
```

プレイヤーの位置を示します

```
UCHAR g_plane_location;
```

プレイヤーと壁が衝突したかを示します

```
BOOL g_plane_bump;
```

変数初期化を行う関数です

```
void initialize_value( void )
```

パラメータ cnt_ で指定された壁をマトリクスLEDの非表示領域へ出力します

```
void disp_putmaze( USHORT cnt_ )
```

マトリクスLEDの表示を1ドット移動(スクロール)します

```
void disp_move1dot( void )
```

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_INTP3( void )
```

外部割り込みINTP3に呼ばれる関数です。

SWの値を g_intp3_sw へ反映します

timer_user.c

```
__interrupt void MD_INTTPOCCO()
```

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_INTTM000()
```

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

macrodriver.h

```
#define D_MLED_CNT 2 /* Matrix LED number */
マトリクスLEDの数2つあるのは1つはバッファ領域として使用しているため。先頭が実際の表示領域。
```

```
#define D_MLED_ROW 8 /* Matrix LED row */
マトリクスLEDの列の数。
```

```
#define D_FONT_WIDTH 8 /* font width */
表示するフォントの幅(ドット数)。実際に表示するのはフォントではなく迷路データになる。
```


ターゲット・システム作成例 (おわり)

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。このゲームには音がないので、タイマをPWM出力させたポートへ圧電スピーカを接続しても面白いと思います。圧電スピーカは小型で非常に薄く、高能率、低消費電力ですのでアンプを通さず、そのままポートに接続しても充分音がでます。

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。このクイック・スタート・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ください。

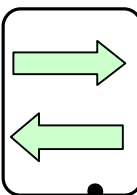
💡 ワンポイント

プログラムの共通化について

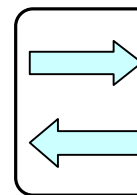
Appliletを使用するとプログラムを共通化するのが楽になります。ここでいう共通化とはCPUを変えた場合を指します。例えば78K0/KF2(8bitマイコン)からV850ES/KG2(32bitマイコン)へ変更してもプログラムはそれほど変化なく作成できます。QB-78K0KF2-TBのクイック・スタート・ガイドと比べてください。電光掲示板プログラムで変わる点は_interrupt void MD_INTTPOCC0(void)のポート制御部分です。またポート制御もdefine定義すれば、もっと共通化できます。

78K0/V850の8bit/
32bitマイコンで
ユーザプログラム
が共通化可能です。

ユーザ
プログラム



Applilet
出力コード



マイコン

AppliletのAPIを使うのでマイコンが変わっても使用する周辺I/Oが同じならプログラムの修正は最小限で済みます。

Appliletの設定はGUIで行うので直感的に周辺I/Oなど設定可能です。複雑なマイコンのマニュアルは読まなくても大丈夫です。

