

お客様各位

ZUD-CD-06-0039-1

1/71

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

78K0S/KB1+ターゲット・ボード QB-78K0SKB1-TB クイック・スタート・ガイド

第2版

ごあいさつ

QB-78K0SKB1-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。

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 5~11

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 12~51

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 52~71

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)

78K0S/KB1+ターゲット・ボード(QB-78K0SKB1-TB)の特徴

78K0S/KB1+(μ PD78F9234MC)搭載

LED2個、SW1個を搭載しており簡単なテストが可能

メイン・クロック8MHzで高速動作可能(4.0V~5.5V供給時)

ユニバーサル・エリア(2.54mmピッチ)を搭載

フラッシュ・メモリ・プログラミング、オンチップ・デバッグに両対応

(X1, X2, INTP3, RESET端子使用)

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

QB-78K0SKX1MINI接続用端子を搭載

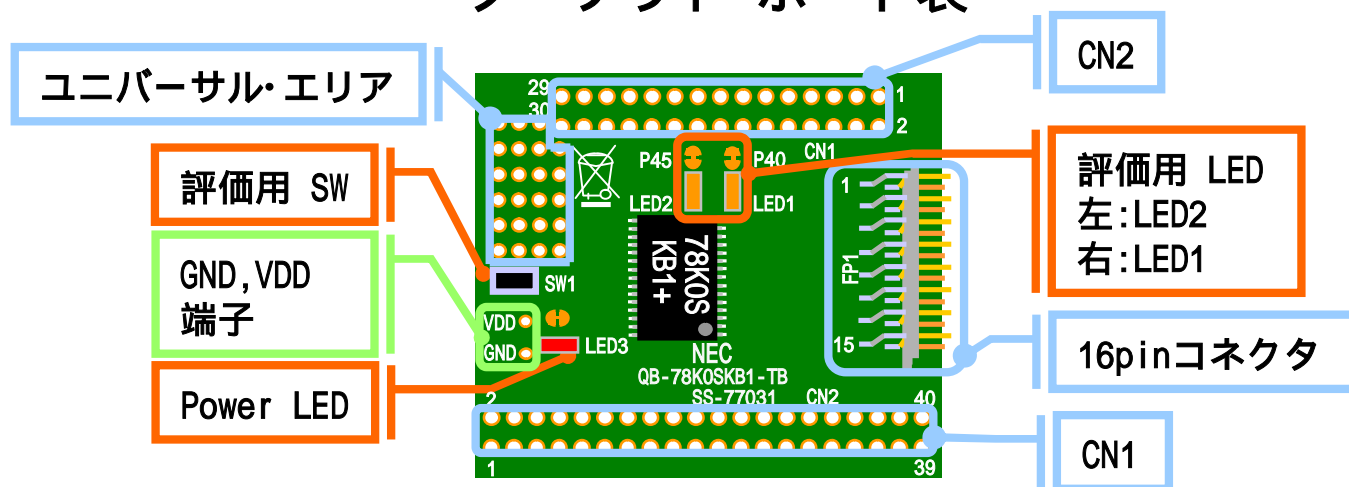
鉛(Pb)フリー対応品

78K0S/KB1+ターゲット・ボード(QB-78K0SKB1-TB)のハードウェア仕様

CPU μ PD78F9234MC	メイン・クロック 動作周波数	8MHz(高速内蔵発振器)
搭載部品	CN1: 周辺ボードコネクタ(2.54mmピッチ) 30pinソケット(パッドのみ)	
	CN2: 周辺ボードコネクタ(2.54mmピッチ) 40pinソケット(パッドのみ) インサーキット・エミュレータ接続用端子搭載	
	FP1: 16pinコネクタ(MINICUBE2接続用)	
	PowerLED: LED赤x1(LED3)	
	評価用LED: LED黄x2(LED1はP40, LED2はP45へ接続)	
	評価用SW: SW1(INTP0へ接続)	
動作電圧	4.0V~5.5V	

ターゲット・ボードについて (その2)

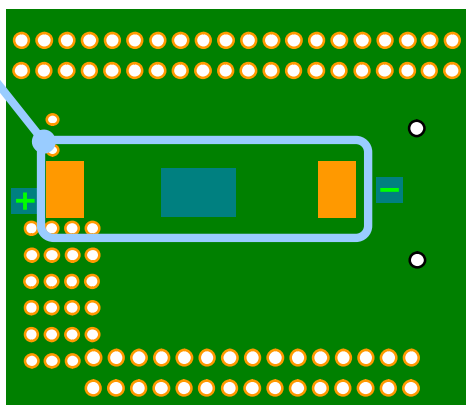
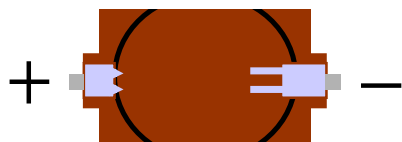
ターゲット・ボード表



- CN1/CN2: 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40(P40)がLOWで黄色に発光します
- 評価用LED2: ポート45(P45)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
- 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。 

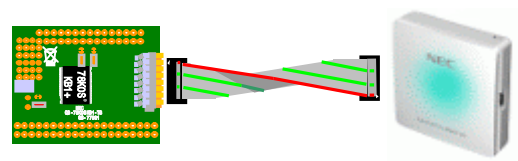
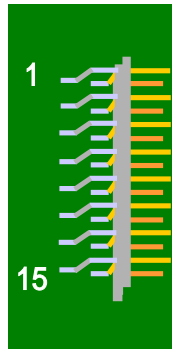
再度ショートさせたい場合は半田ショートさせてください。 

P40, P45を使用する場合はLEDの上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)

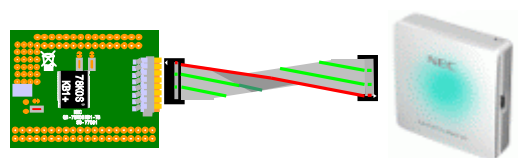
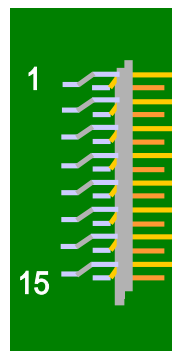
ピン番号	信号	ピン番号	信号
1	GND	2	RESET_OUT
3	R.F.U.	4	VDD
5	X2	6	R.F.U.
7	R.F.U.	8	INTP3(H/S)
9	X1	10	R.F.U.
11	-----	12	INTP3
13	R.F.U.	14	R.F.U.
15	RESET_IN	16	R.F.U.



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	R.F.U.	4	VDD
5	X2	6	R.F.U.
7	R.F.U.	8	R.F.U.
9	X1	10	R.F.U.
11	-----	12	R.F.U.
13	R.F.U.	14	R.F.U.
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	P03		2	P02	
3	P01		4	P00	
5	P123		6	GND	
7	VDD		8	NC	
9	NC		10	T_RESET	16pinコネクタ15へ接続
11	P33		12	P32	
13	P31/TI010/T000/ INTP2		14	P30/TI000/INTP0	SW1にも接続
15	P40	LED1へ接続	16	P41/INTP3	16pinコネクタ12へ接続
17	P42/TOH1		18	P43/TxD6/INTP1	
19	P44/RxD6		20	P45	LED2へ接続
21	P46		22	P47	
23	P130		24	P23/ANI3	
25	P22/ANI2		26	P21/ANI1	
27	P20/ANI0		28	VDD	
29	GND		30	P120	

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	VDD		2	GND	
3	P20/ANI0		4	GND	
5	P21/ANI1		6	GND	
7	P22/ANI2		8	GND	
9	P23/ANI3		10	GND	
11	VDD		12	GND	
13	P120		14	GND	
15	NC		16	GND	
17	NC		18	GND	
19	P123		20	NC	
21	P00		22	P40	LED1へ接続
23	P01		24	P41/INTP3	16pinコネクタ12へ接続
25	P02		26	P42/TOH1	
27	P03		28	P43/TxD6/INTP1	
29	P130		30	P44/RxD6	
31	P30/TI000/INTP0	SW1にも接続	32	P45	LED2へ接続
33	P31/TI010/T000/ INTP2		34	P46	
35	P32		36	P47	
37	P33		38	GND	
39	T_RESET	16pinコネクタ15へ接続	40	GND	

システム構成図1 プログラム開発時（シミュレータ使用時）

プログラム開発時（シミュレータ使用時）

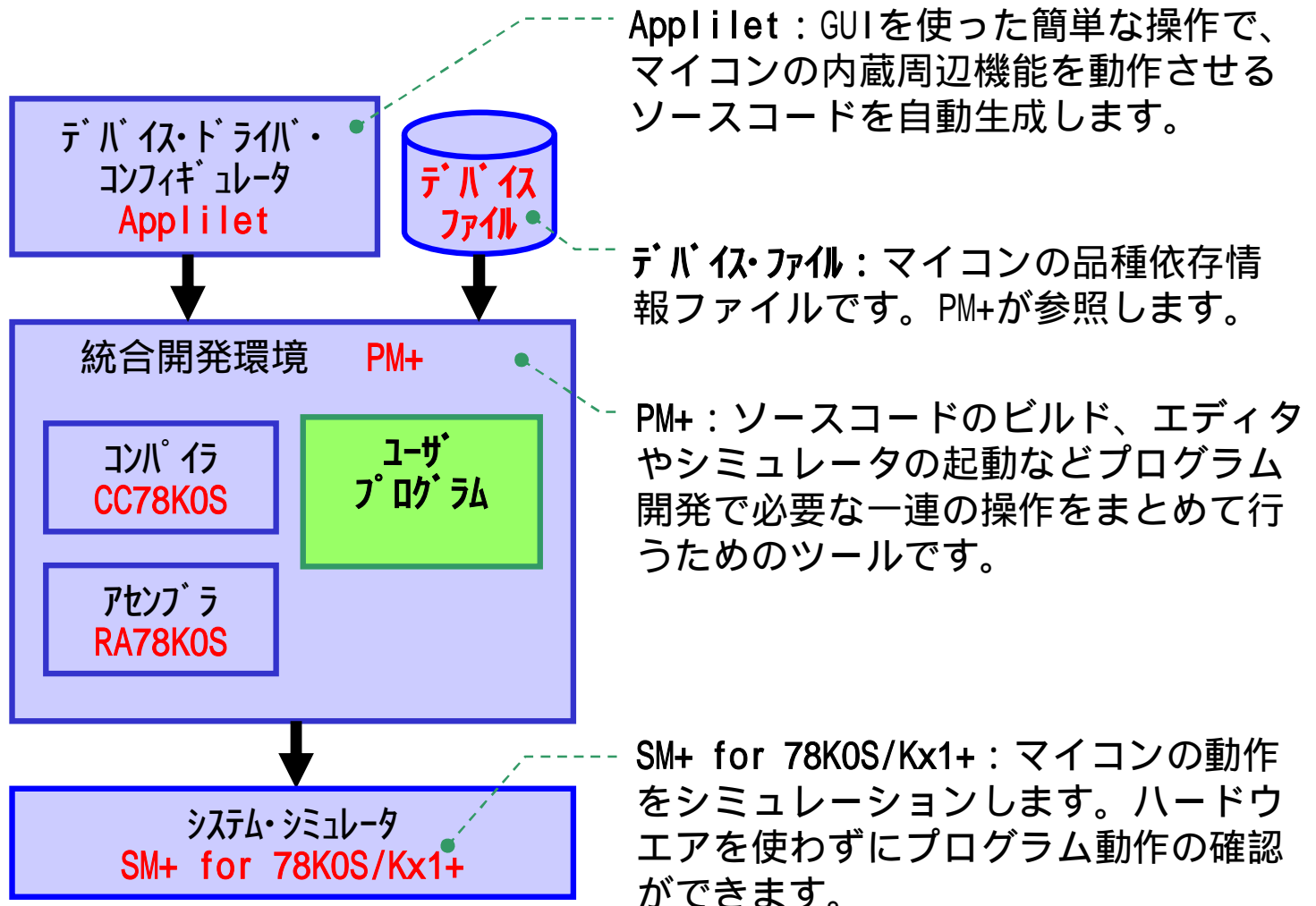
ハードウェアを必要としない構成です

ホストコンピュータ

78K0S/Kx1+シリーズ用開発ツール[注1]

- コンパイラ (CC78K0S)
- PM+ (RA78K0S,アセンブラを含む)
- システム・シミュレータ (SM+ for 78K0S/Kx1+)
- デバイス・ファイル (DF789234)
- デバイス・ドライバ・コンフィギュレータ(Applilet)

ダウンロード
/インストール



[注1]プログラム開発に必要な開発ツールは、弊社WEBページから無償でダウンロードできます。
<http://www.necel.com/micro/jpn/product/sc/lowpin/lowpin-freesoft.html>

システム構成図2 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。

78K0S/Kx1+シリーズ用開発ツール

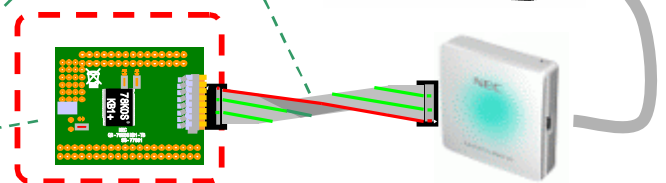
- コンパイラ (CC78K0S)
- PM+ (RA78K0S, アセンブラを含む)
- デバッガ (ID78K0S-QB)
- デバイス・ファイル (DF789234)
- デバイス・ドライバ・コンフィギュレータ (Applilet)

16pinターゲット・ケーブル

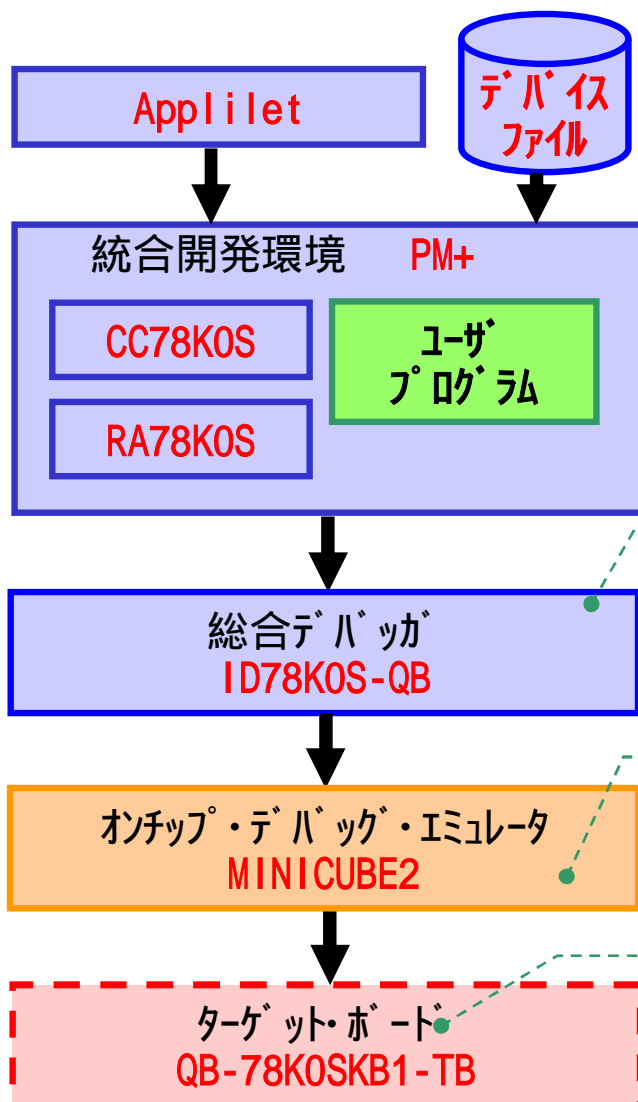
ターゲット・ボード：
78K0S/KB1+を搭載

ホストコンピュータ

ダウンロード
インストール



MINICUBE2



ID78K0S-QB：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-78K0SKB1-TB：本製品です。78K0S/KB1+を搭載しています。

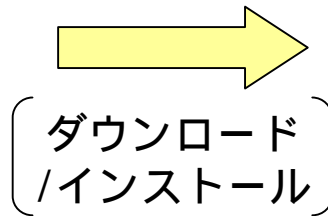
システム構成図3 プログラム書き込み時

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。

78K0S/Kx1+シリーズ用開発ツール

- プログラミング GUI (QB-Programmer)
- 78K0S/Kx1+用パラメータファイル (PRM78F9234)

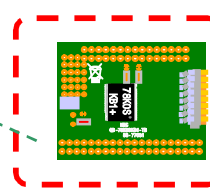


ホストコンピュータ



16pinターゲットケーブル

ターゲットボード：
78K0S/KB1+を搭載



MINICUBE2

ユーザ・プログラム：統合環境PM+で生成されたROM化対応したオブジェクト・ファイルを使います。

パラメータファイル：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 GUI：MINICUBE2を使ってマイコンへプログラミングするためのGUIソフトです。

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-78K0SKB1-TB：本製品です。78K0S/KB1+を搭載しています。



プログラミング GUI
QB-Programmer

オンチップ・デバッグ・エミュレータ
MINICUBE2

ターゲットボード
QB-78K0SKB1-TB



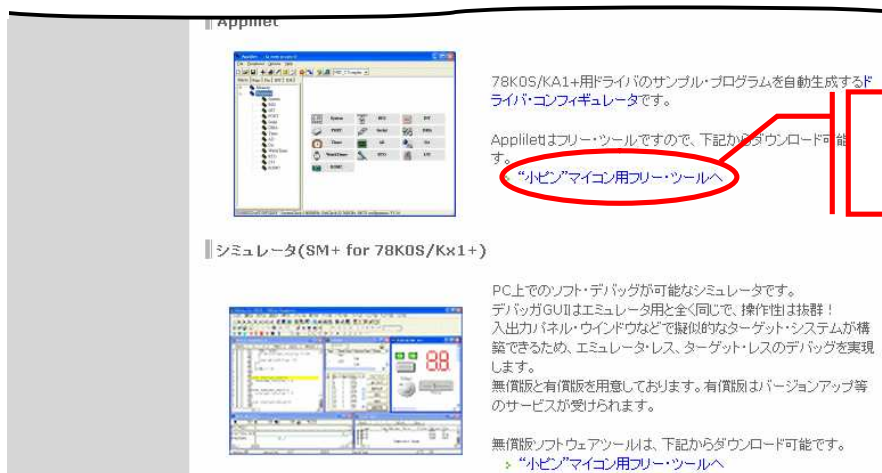
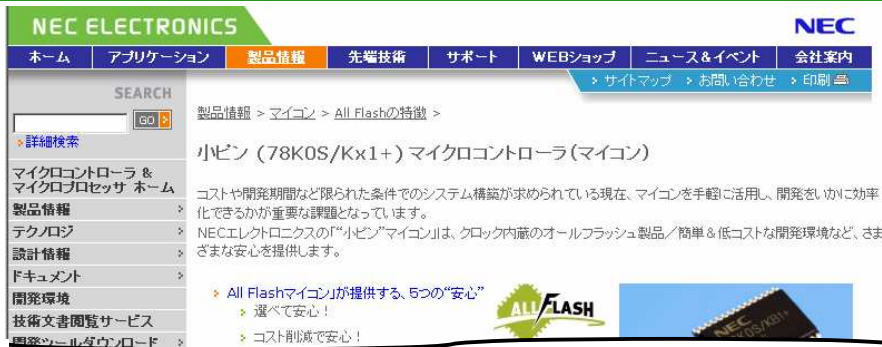
ワンポイント

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



“小ピン”マイコン用フリー・ツールへ をクリック!!

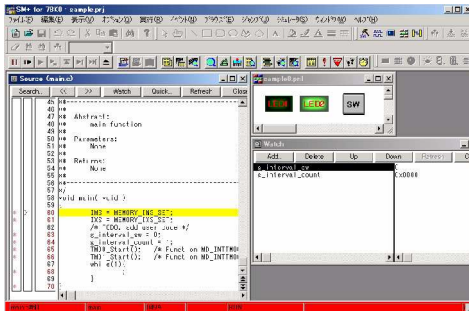
“78K0S/Kx1+用開発ツール”のインストール方法：

- **プログラミングGUI (QB-Programmer)、78K0S/Kx1+用Appilet**
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。Appiletのバージョンは1.70以上を使用して下さい。
- **SM+ for 78K0S/Kx1+、RA78K0S、CC78K0S**
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。
- **78K0S/Kx1+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。
- **78K0S/Kx1+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

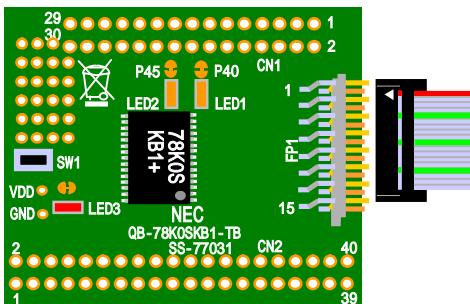
この章ではMINICUBE2、ターゲット・ボード、無償版開発ツールを使用して、プログラムの作成から動作確認、マイコンへ書き込むまでの一連の操作を、実際のプログラムを用いて説明します。システム構成によって開発手順が異なるので、それぞれについて説明します。

1. システム・シミュレータ (SM+) 体験編



ハードウェアを必要としない、無償ダウンロードツールだけで動作します。サンプルとしてタイマ割り込みを使用して2個のLEDを点灯させるプログラム作成します。30分程の時間で一通り体験できます

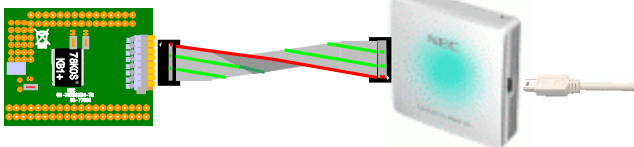
2. ターゲット・ボード体験編



MINICUBE2、ターゲット・ボード、フリー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ID78K0S-QBの基本的な使い方を学びます。

3. マイコン・プログラミング体験編

2. ターゲット・ボード体験編で作成したプログラムをマイコンへ書き込みます。



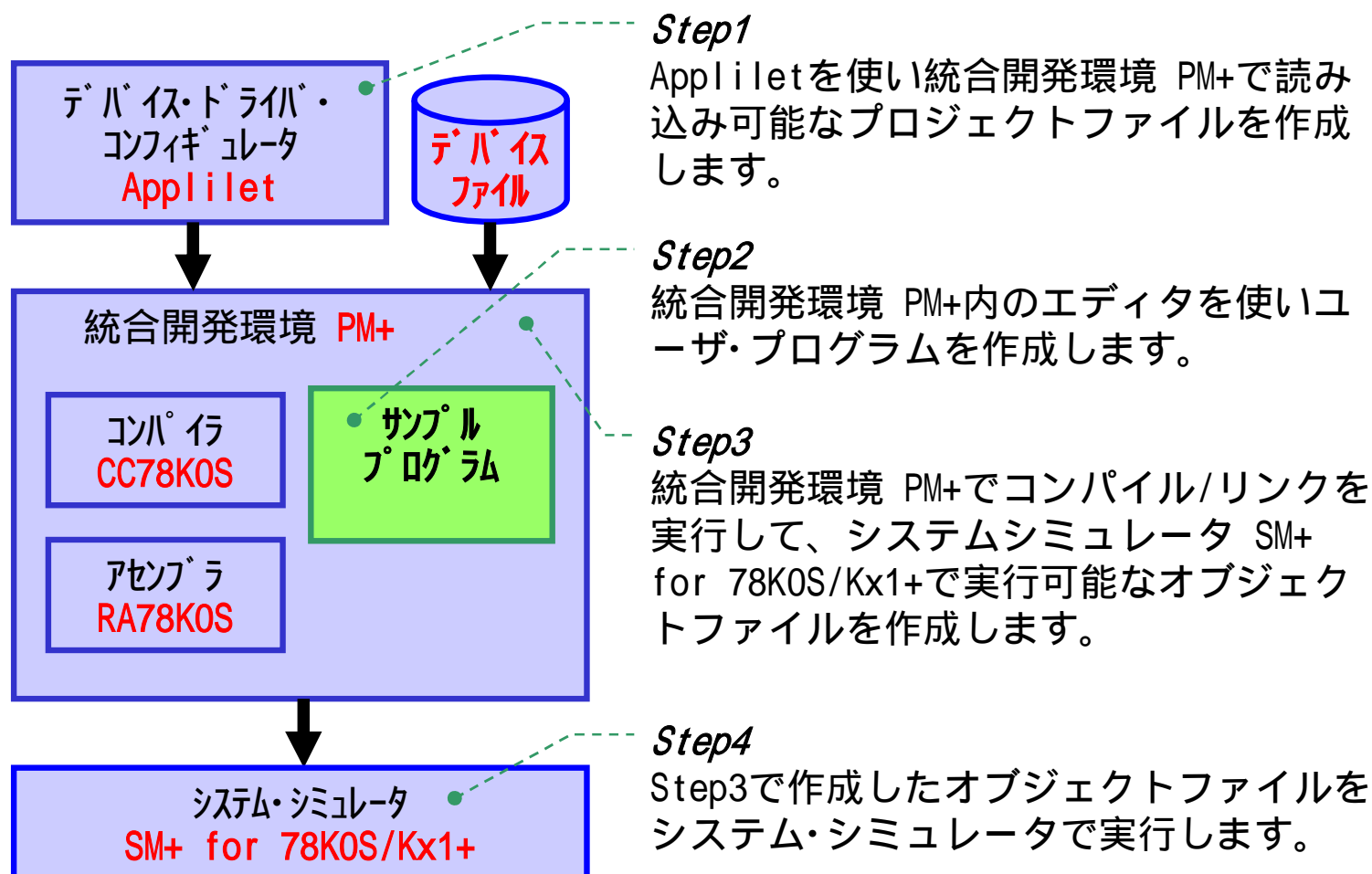
ワンポイント

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。

システム・シミュレータ (SM+) 体験編

SM+編ではフリー・ツールを使用したソフトウェア構成でのプログラムの作成手順を説明します。



次ページより *Step1* ~ *Step4* の詳細を説明します。

システム・シミュレータ (SM+) 体験編 (Step1-1)

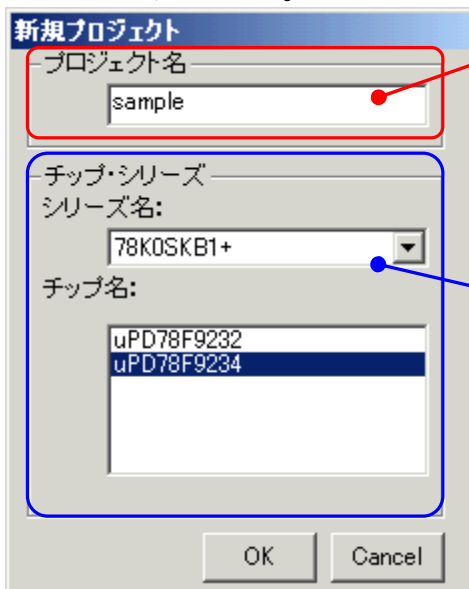
Step1 Applilet を使い統合開発環境 PM+ で読み込み可能なプロジェクトファイルを作成します。

a. Applilet を起動します。

[スタート] [プログラム(P)] [NEC Electronics Tools]
[Applilet for 78K0SKX1+] [Vx.xx] [Applilet for 78K0SKX1+ 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>

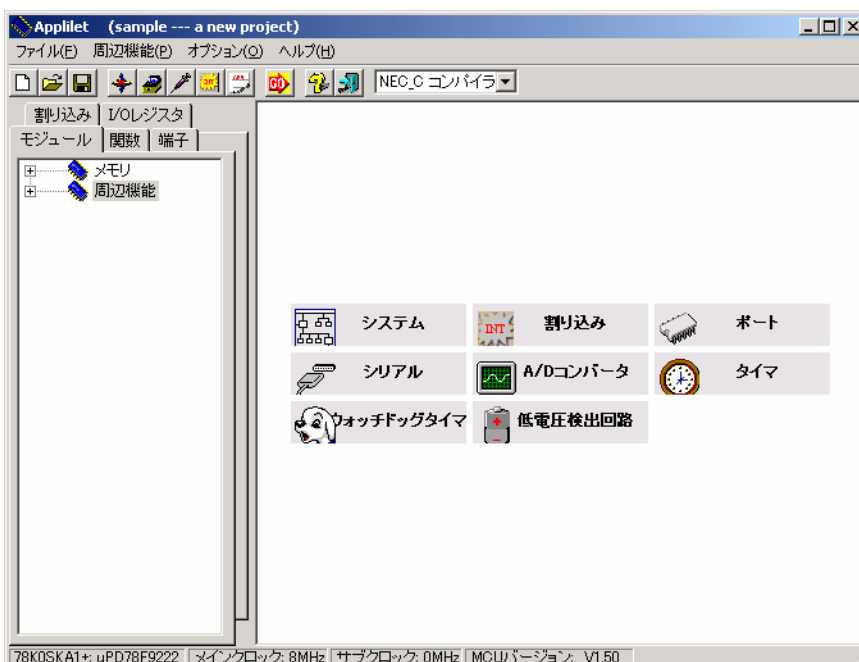
mdt sample に変更

<チップ・シリーズ>

シリーズ名: 78K0SKB1+ を選択

チップ名: uPD78F9234 を選択

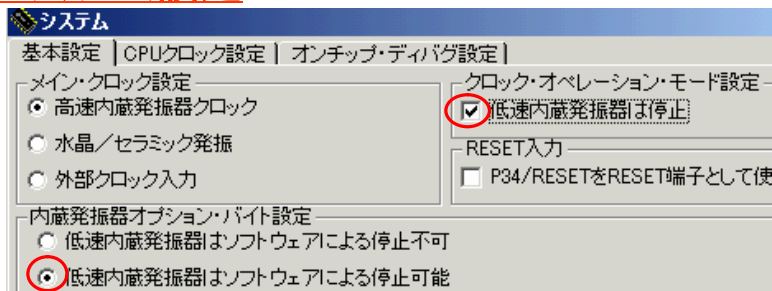
OK をクリックすると周辺機器メニューが表示されます



システム・シミュレータ (SM+) 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



低速内蔵発振器はソフトウェアによる停止可能にチェック。低速内蔵発振器は停止にチェック

割り込み設定

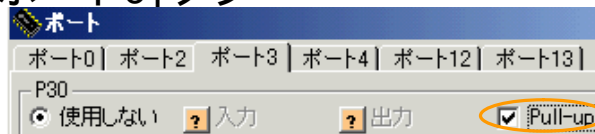


[外部割り込み設定]タブ
“INTP0許可”に
チェック



ポート設定

[ポート3]タブ



P30のPull-up(プルアップ)にチェック

P30はINTP0と兼用端子です。そのため先に割り込み設定を行うとP30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。ここではINTP0のPull-upの設定をします

[ポート4]タブ



P40、P45の「出力、1」にチェック

LED1、LED2はそれぞれP40、P45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にします。

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。

割り込み設定

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。

ポート設定

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。P30/INTP0が兼用端子のためINTP0を設定した後ではP30の設定はPull-upのみチェック可能となっています。

システム・シミュレータ (SM+) 体験編 (Step1-3)

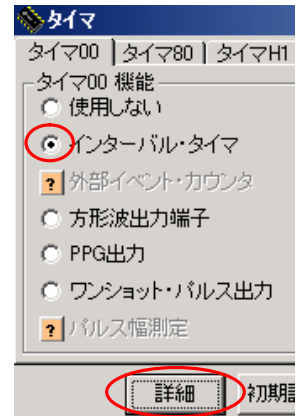
d. サンプル・プログラムで使用する周辺機器を設定します。その2



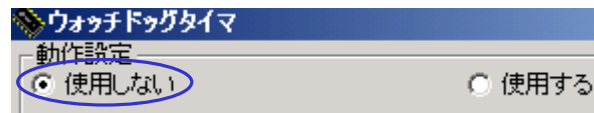
設定を行うと表示が変化します。
アイコン 黄色
タイトル 青色

タイマ設定

[タイマ00]タブ
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。



ウォッチドッグタイマ設定



使用しないにチェック。

ワンポイント

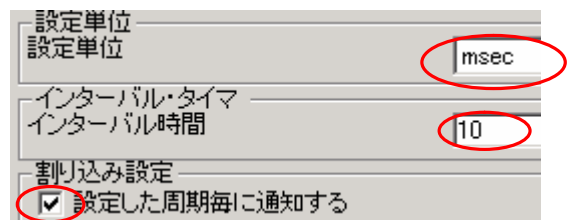
ウォッチドッグタイマ設定

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ア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。

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。



設定単位をmsecへ変更し、インターバル時間を10とします。割り込み設定で設定した周期毎に通知するにチェックします。この設定でタイマ00は10msec毎に割り込みルーチンが呼ばれる設定になります。

設定できるインターバル時間は以下の通りです。（メイン・クロック8MHz動作時）


- ・ msec: 1 ~ 2097
- ・ usec: 1 ~ 2097119
- ・ nsec: 1 ~ 2097119999

システム・シミュレータ (SM+) 体験編 (Step1-4)

e. ソースコードを自動生成します。

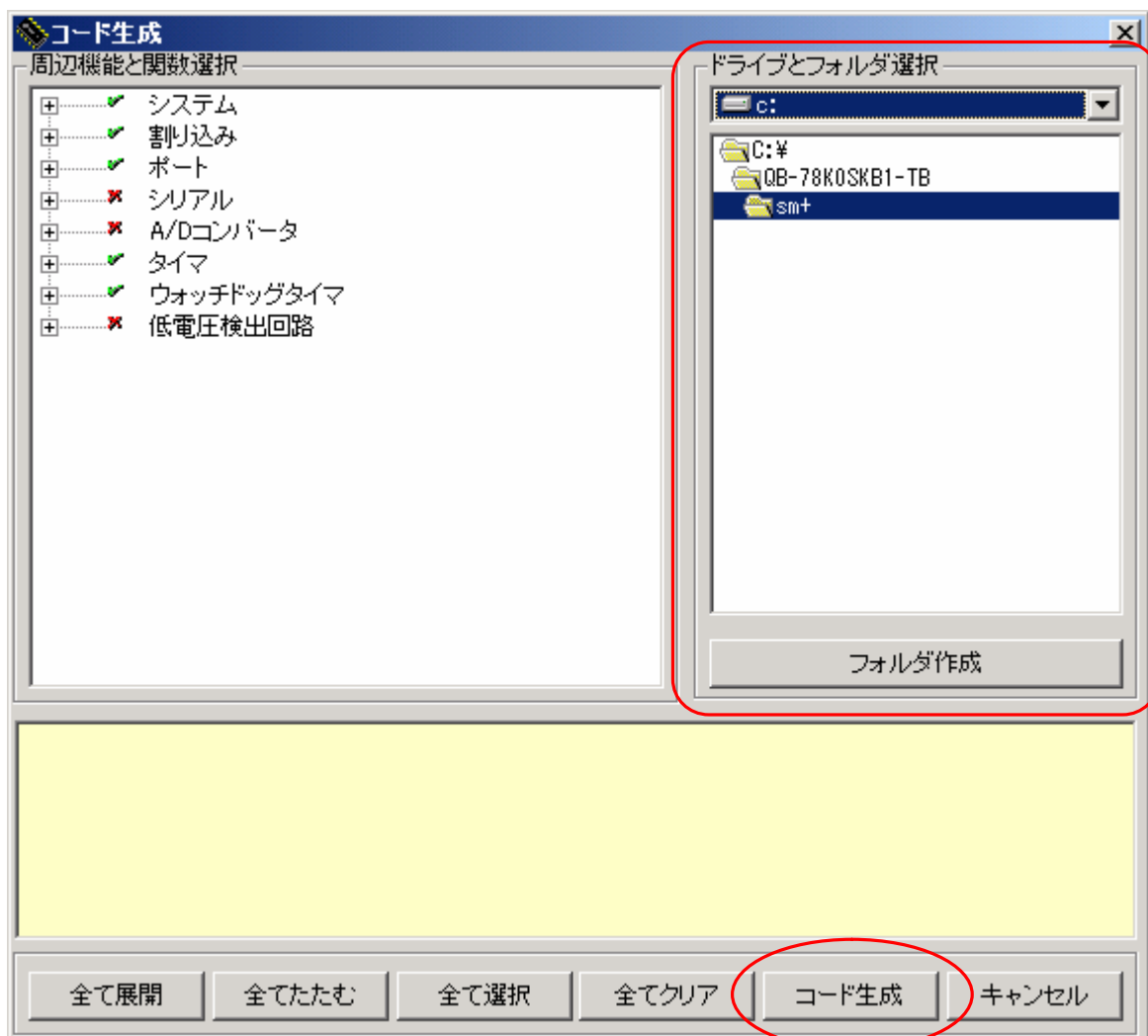
e-1) コンパイラ (CC78K0S) のパスを設定します。

コンパイラを C:\NECTools32 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。

e-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。



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



[注] コード生成するフォルダおよびパスに、空白文字や漢字などの2バイト文字は使用しないでください。

システム・シミュレータ (SM+) 体験編 (Step1-5)

e-4) [コード生成]が完了し、ダイアログが表示されます。



e-5) Appliletを終了します。

Appliletで設定した値

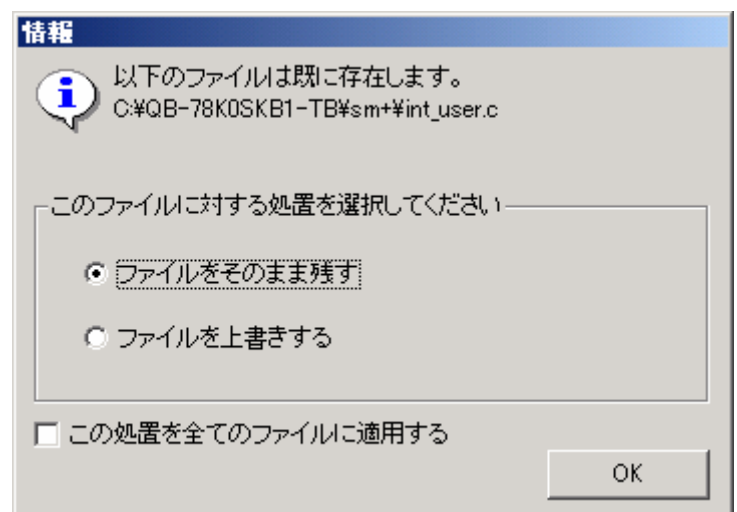
- メイン・システムクロック 8MHz
- 低速内蔵発振器停止
- INTP0割り込み許可
- P40, P45を出力ポートにて、初期値を1とする
- タイマ00をインターバル・タイマに使用して、10msec毎に割り込みを行う
- ウォッチドッグタイマ未使用

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。ユーザー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。



システム・シミュレータ (SM+) 体験編 (Step2-1)

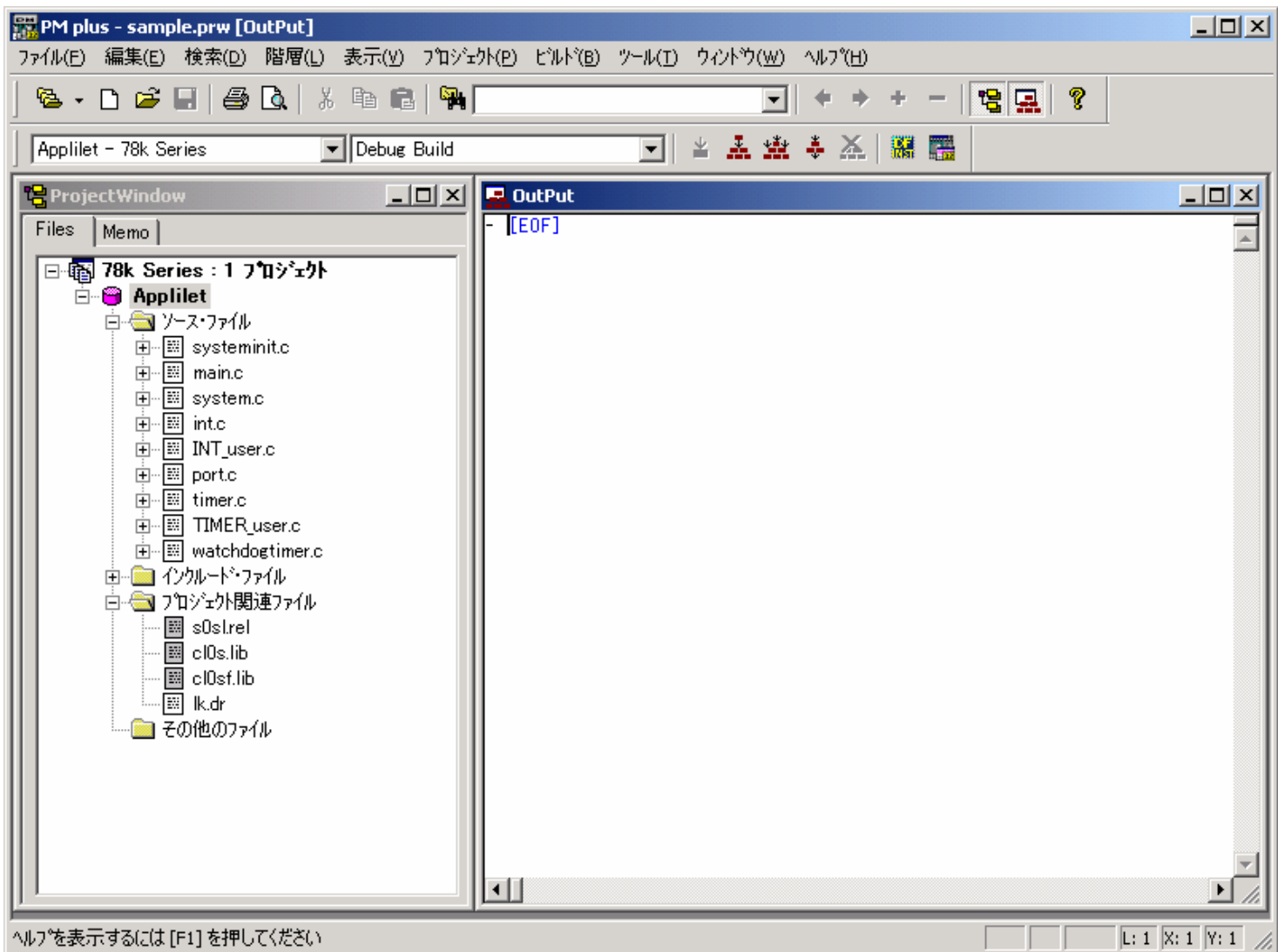
Step2 統合開発環境 PM+内のエディタを使い、サンプル・プログラムを作成します。

a. PM+を起動します。

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

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。
- 『ワークスペースを開く』ダイアログで、システム・シミュレータ(SM+)体験編 Step1-4で指定したフォルダの“sample.prw”を指定して、[開く(O)] ボタンを押してください。



システム・シミュレータ (SM+) 体験編 (Step2-2)

c. LED2個を一定間隔で点灯させ、外部SWによってその間隔を制御するプログラムを作成します。

c-1) PM plus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
main関数にタイマをスタートさせる『TM00_Start();』の呼び出しを追加します。
また、グローバル変数の初期化も追加します。

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 )  
{  
    /* TODO. add user code */  
    g_interval_sw = 0;  
    g_interval_count = 1;  
    TM00_Start(); /*Function MD_INTTM000( ) is called by every 10msec*/  
    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。
外部SWが押された場合に呼ばれる割り込み関数MD_INTPO()へ処理を追加します。

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

システム・シミュレータ (SM+) 体験編 (Step2-3)

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。
10msec毎に呼ばれる関数MD_INTTM000()へ処理を追加します。

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

__interrupt void MD_INTTM000( )
{
    /*TODO*/
    UCHAR inreg, outreg;
    if ( g_interval_count == 0 )
    {
        g_interval_count = g_counter_data[ g_interval_sw ];
        inreg = P4.0;
        outreg = inreg ^ 1;
        P4.0 = outreg;
        P4.5 = inreg;
    }
    g_interval_count--;
}
```

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P40,P45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。

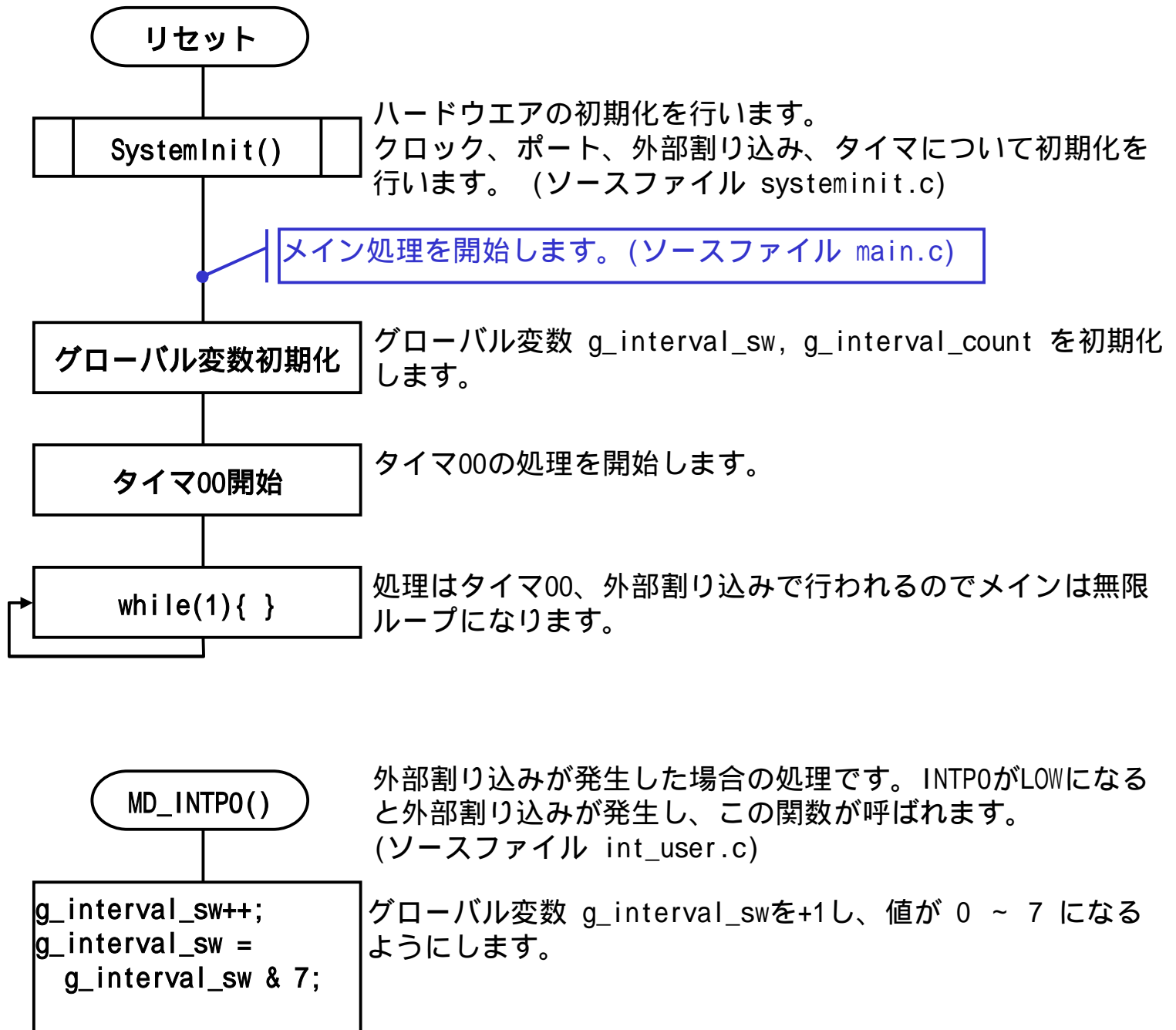
g_interval_swはSWを押下すると外部割り込み関数 MD_INTP0()が呼ばれ +1されます。g_interval_swは0~7の値になります。g_interval_countは10msec毎に呼ばれる関数MD_INTTM000()で -1されます。g_interval_countが0の場合にP40,P45に接続されたLEDを交互に点灯させます。

グローバル変数の説明

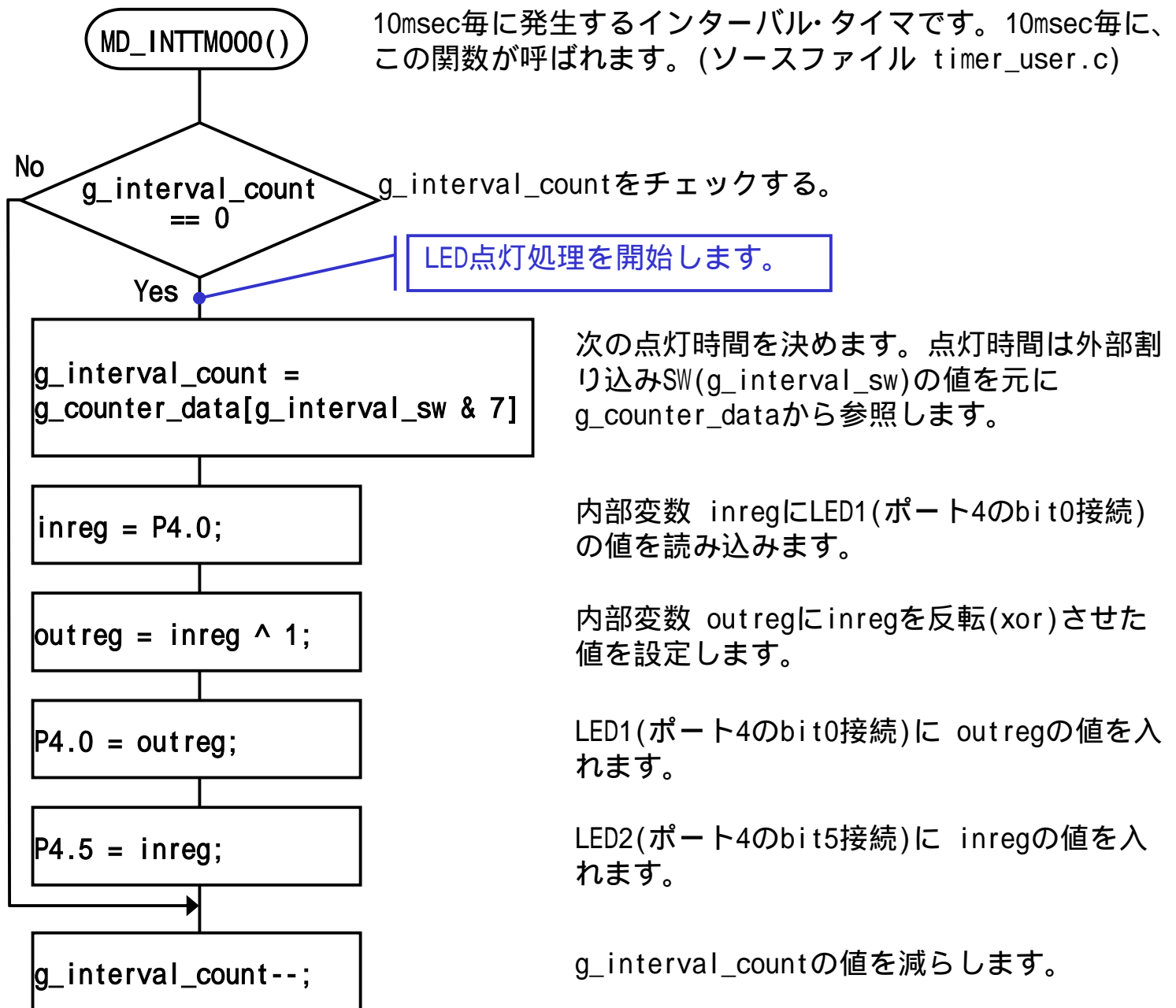
g_interval_sw: 外部SWを押下すると変化する。0~7の値になる。
g_interval_count: 10msec毎に -1されるカウンタ。0の時にLED点灯を変更します。
g_counter_data[8]: 点灯速度を決めるデータ。8段階のスピードを設定します。

システム・シミュレータ (SM+) 体験編 (Step2-4)

d. サンプル・プログラムの流れ図を示します。



システム・シミュレータ (SM+) 体験編 (Step2-5)



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。

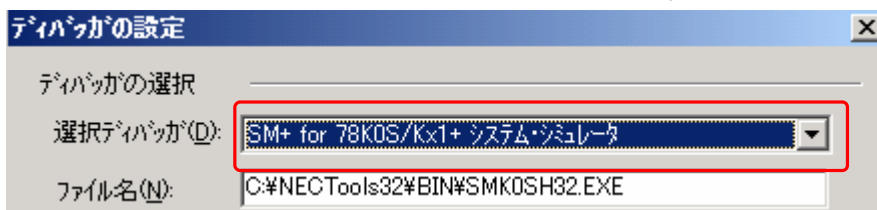
システム・シミュレータ (SM+) 体験編 (Step3)

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、ビルドメニューよりビルドを選択してください。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+ から連携起動するシミュレータを設定します。
メニュー・バーの[ツール(T)] [デバッグの設定(D)...]を選択します。『デバッグの設定』ダイアログのプルダウンメニューで、選択デバッグに“SM+ for 78K0S/Kx1+ システム・シミュレータ”を設定します。

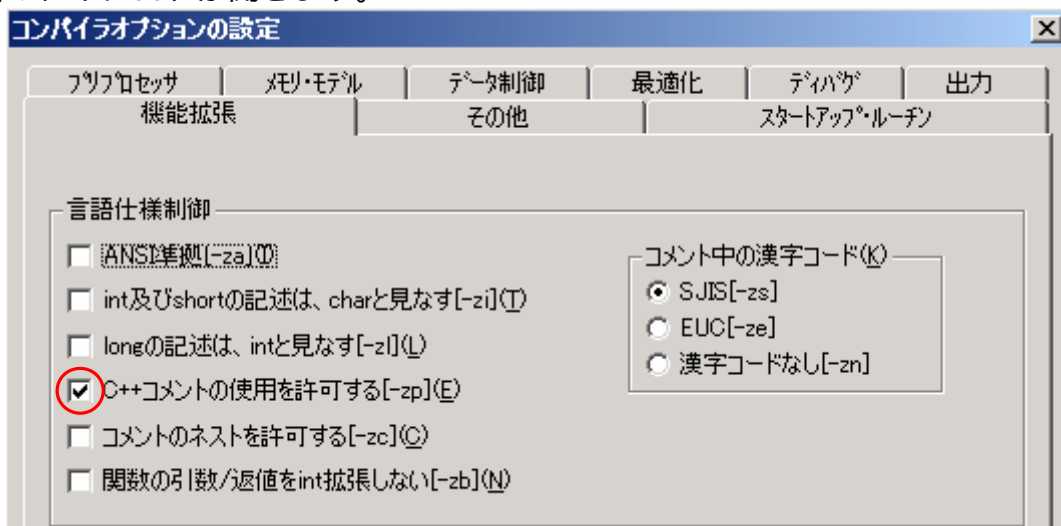


💡ワンポイント

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。機能拡張タブを選択すると下図のダイアログが開きます。




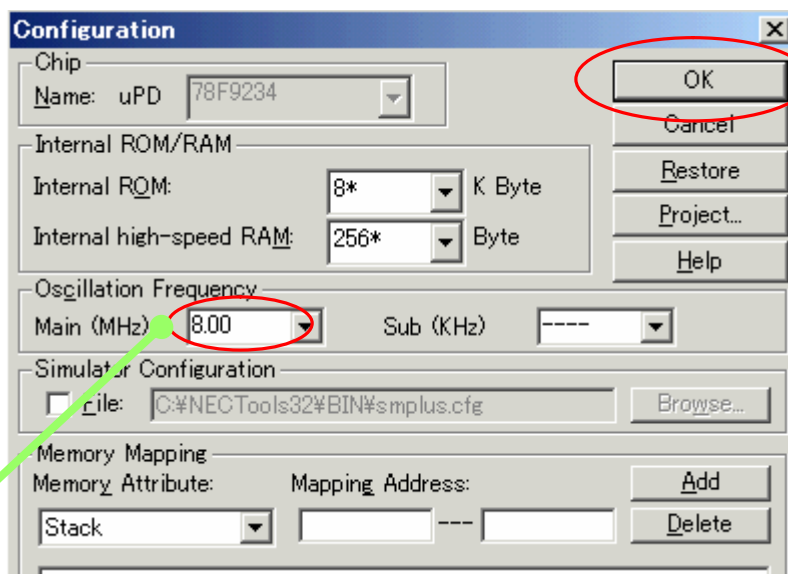
C++コメントの使用を許可する[-zp](E)、コメント中の漢字コード(K)エリアのSJIS[-zs]にチェックしてください。

システム・シミュレータ (SM+) 体験編 (Step4-1)

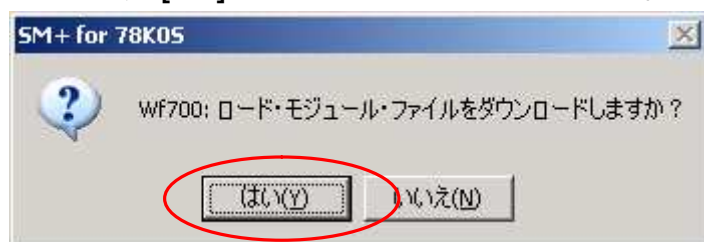
Step4 シミュレータ (SM+ for 78K0S/Kx1+) でプログラムの動作を確認します。

a. PM+ から SM+ for 78K0S/Kx1+ を起動します。

[デバッグ] ボタン  を押して、SM+ for 78K0S/Kx1+ を起動します。起動すると『Configuration』ダイアログが表示されるので、Oscillation Frequency の値を 8.00 MHz に設定して、[OK] ボタンを押してください。



ダウンロード確認のダイアログが表示されますので、[OK] ボタンを押してください。




ワンポイント

Configurationについて


Oscillation Frequency の値はAppliletの[基本設定]タブのメイン・クロック設定が「水晶/セラミック発振」「外部クロック」選択時に有効となります。その際は周波数設定と同じ値を設定してください。この値はシリアル・ポート、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ください。

システム・シミュレータ (SM+) 体験編 (Step4-2)


b. LEDが正しく点灯しているかを、シミュレータ(SM+ for 78K0S/Kx1+)の入出力パネルに擬似ターゲット・システム(LED,SW)を構築して確認します。

b-1) [入出力パネル]ボタンを押すか [シミュレータ(S)] [入出力パネル(P)]を選択して『入出力パネル1』を表示します。



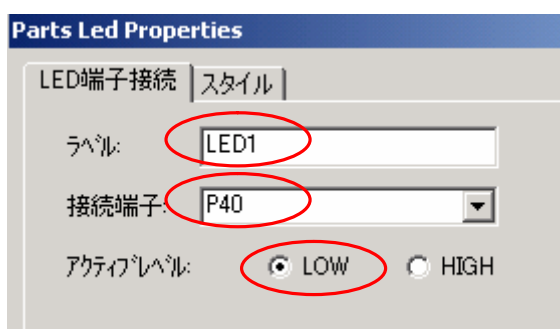
b-2) 『入出力パネル1』をアクティブにした状態で[LED作成]ボタンを押すか [部品(P)] [LED(E)]を選択して、『入出力パネル1』に任意の大きさのLED2個を貼り付けます。



b-3) 同様に[ボタン作成]ボタンを押すか [部品(P)] [ボタン(B)]を選択して『入出力パネル1』に任意の大きさのボタンを貼り付けます。



b-4) 左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(Button) Properties』ダイアログで、端子等を設定します。



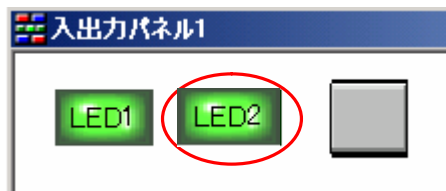
ラベルをLED1、接続端子をP40に設定し、アクティブレベルLOWへチェックします。
これで1つのLEDが接続できました。



システム・シミュレータ (SM+) 体験編 (Step4-3)

- b-5) 同様に右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(Button) Properties』ダイアログで、端子等を設定します。

ラベルをLED2、接続端子をP45に設定し、アクティブレベルLOWへチェックします。これでLEDが2個設定できました。



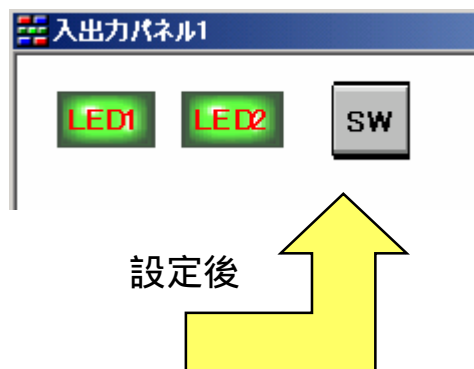
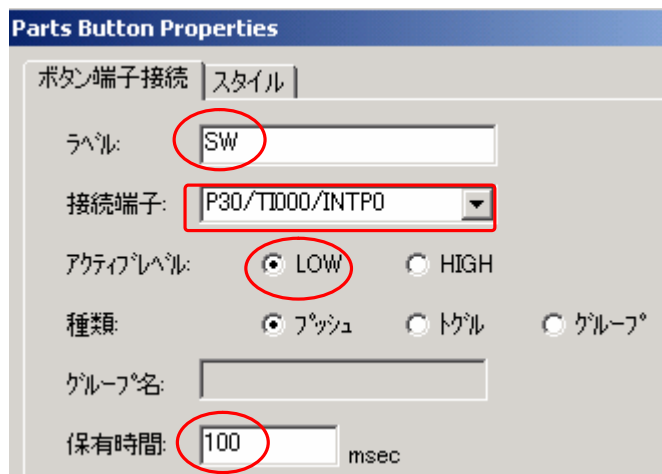
ラベルのフォントを変更する場合

[図形(E)] [フォントの指定(Q)...] を選択します。フォント名、色、スタイル、サイズが変更可能です。右図は色を「赤」、スタイル「太字」に変更した場合です。



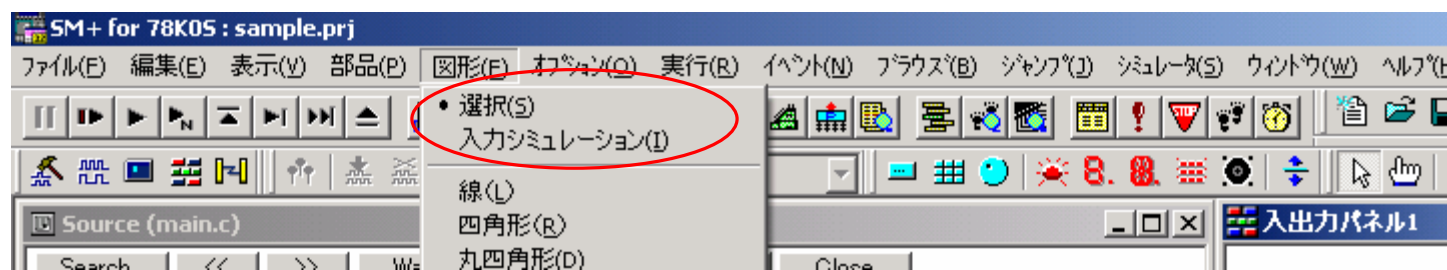
- b-6) SW部品の上で右クリック [プロパティ(R)] を選択して表示される『Parts Led(Button) Properties』ダイアログで、端子等を設定します。

ラベルをSW、接続端子をP30に設定し、アクティブレベルLOWへチェックします。また保有時間を100msecに設定します。フォントも「太字」へ変更します。



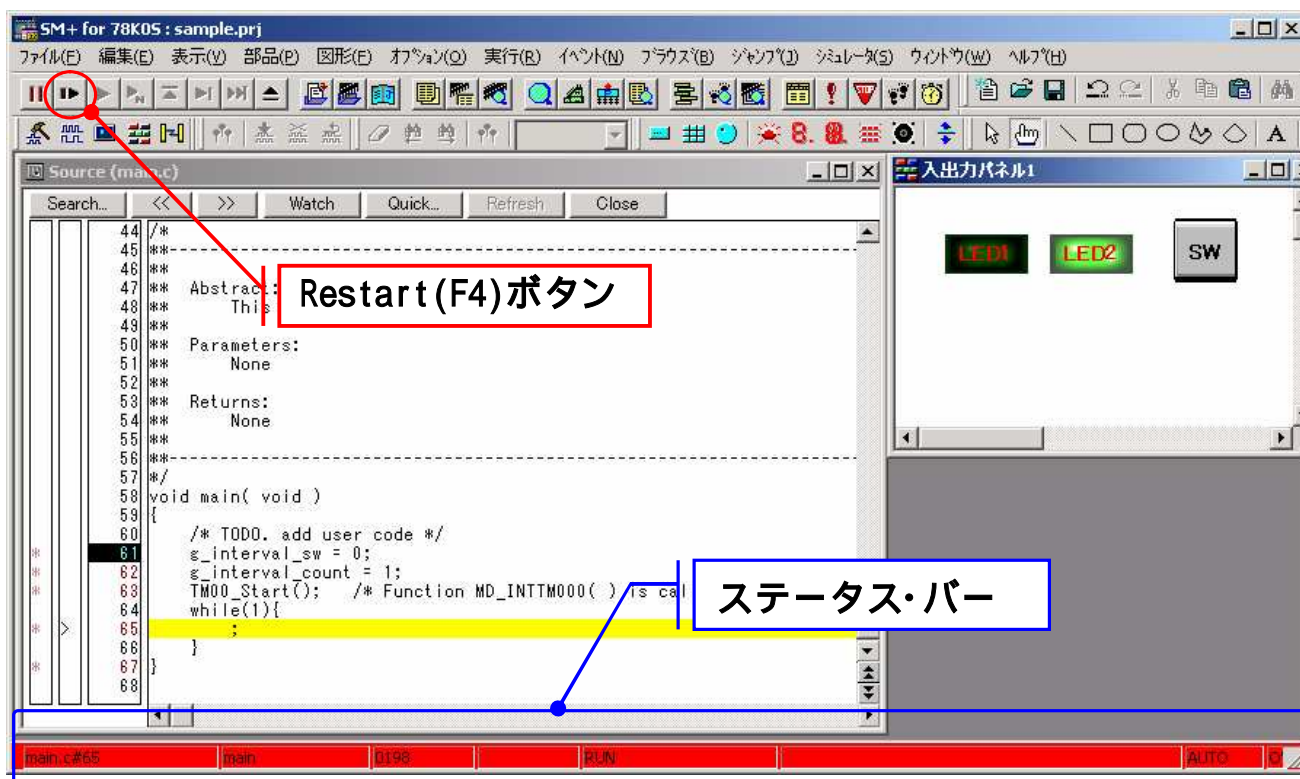
💡ワンポイント

部品の編集を行うときは[選択(S)]、シミュレーションを実行するときは[入力シミュレーション(I)]をチェックしてください。



システム・シミュレータ (SM+) 体験編 (Step4-4)

- c. [Restart]ボタンを押して、シミュレーションを実行します。
シミュレーション中は、ステータス・バーが赤く表示されます。



- c-1) LEDの点灯を確認します。



交互に高速(シミュレータ上20msec毎)に点滅します。



- c-2) SWを押下します。



SWを押下すると、一瞬ボタンが押されLEDの点滅速度が遅くなります。押下する毎に点滅速度が低下します。8回押下すると高速点滅に戻ります。

💡ワンポイント

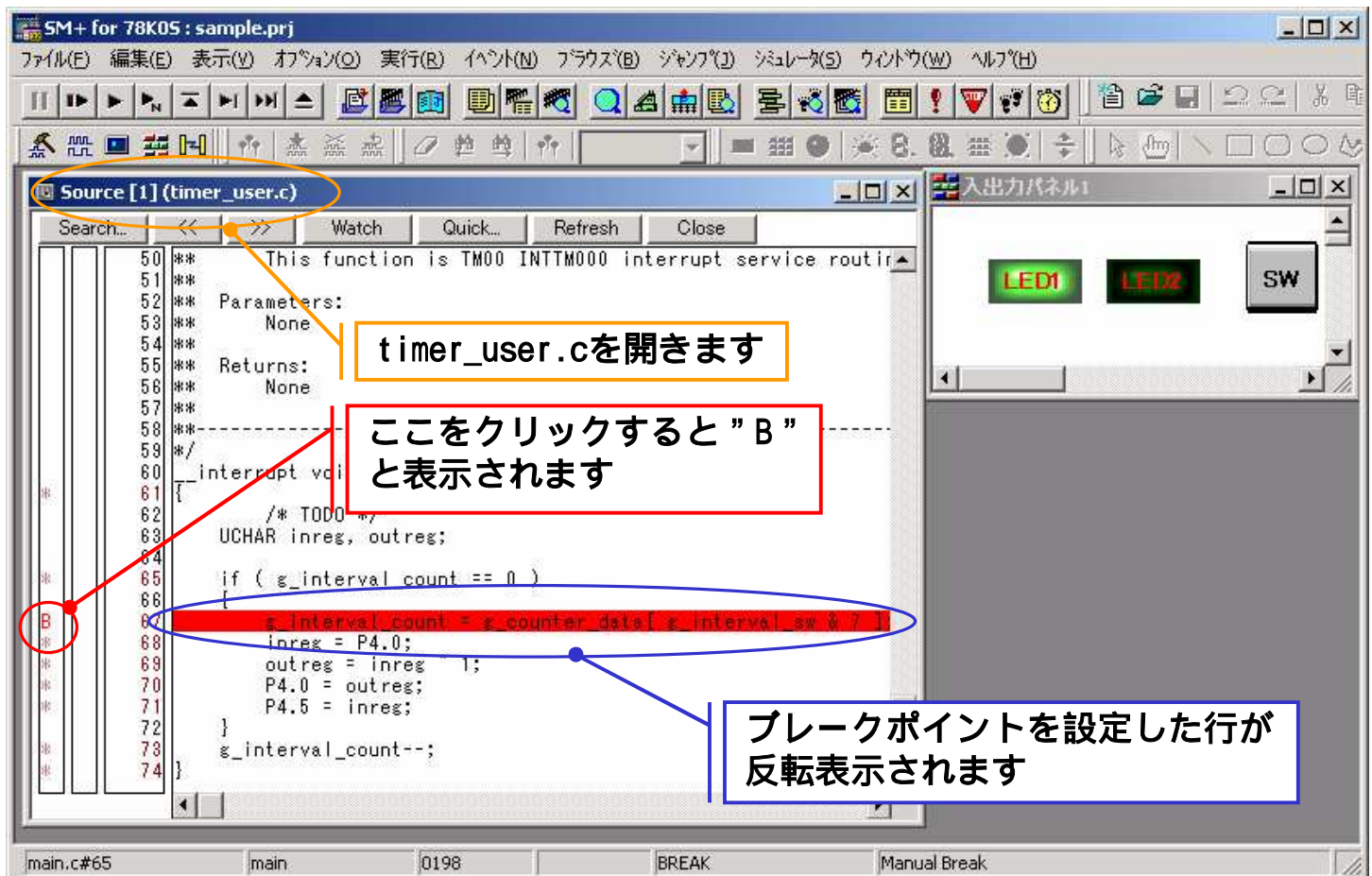
SM+ の設定保存について

プロジェクト・ファイルの保存確認ダイアログが開くので、通常は[はい]を選択してください。ブレーク・ポイントの設定や、入出力パネルの設定内容等が保存されます。作成した「入出力パネル1」はSM+ の情報保存時に名前が変更され「samplep0」になります。保存した内容は次回PM+ からSM+ を起動した際に自動的に読み込まれます。

システム・シミュレータ (SM+) 体験編 (Step4-5)

d. SM+ の基本的な使い方について(ブレークポイントの設定)

- d-1) ブレークポイントの設定し、任意の場所でプログラムの実行を停止させます。
[ファイル(F)] [開く(O)...]を選択し「timer_user.c」を開きます。
- d-2) 「timer_user.c」を開いたら67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。




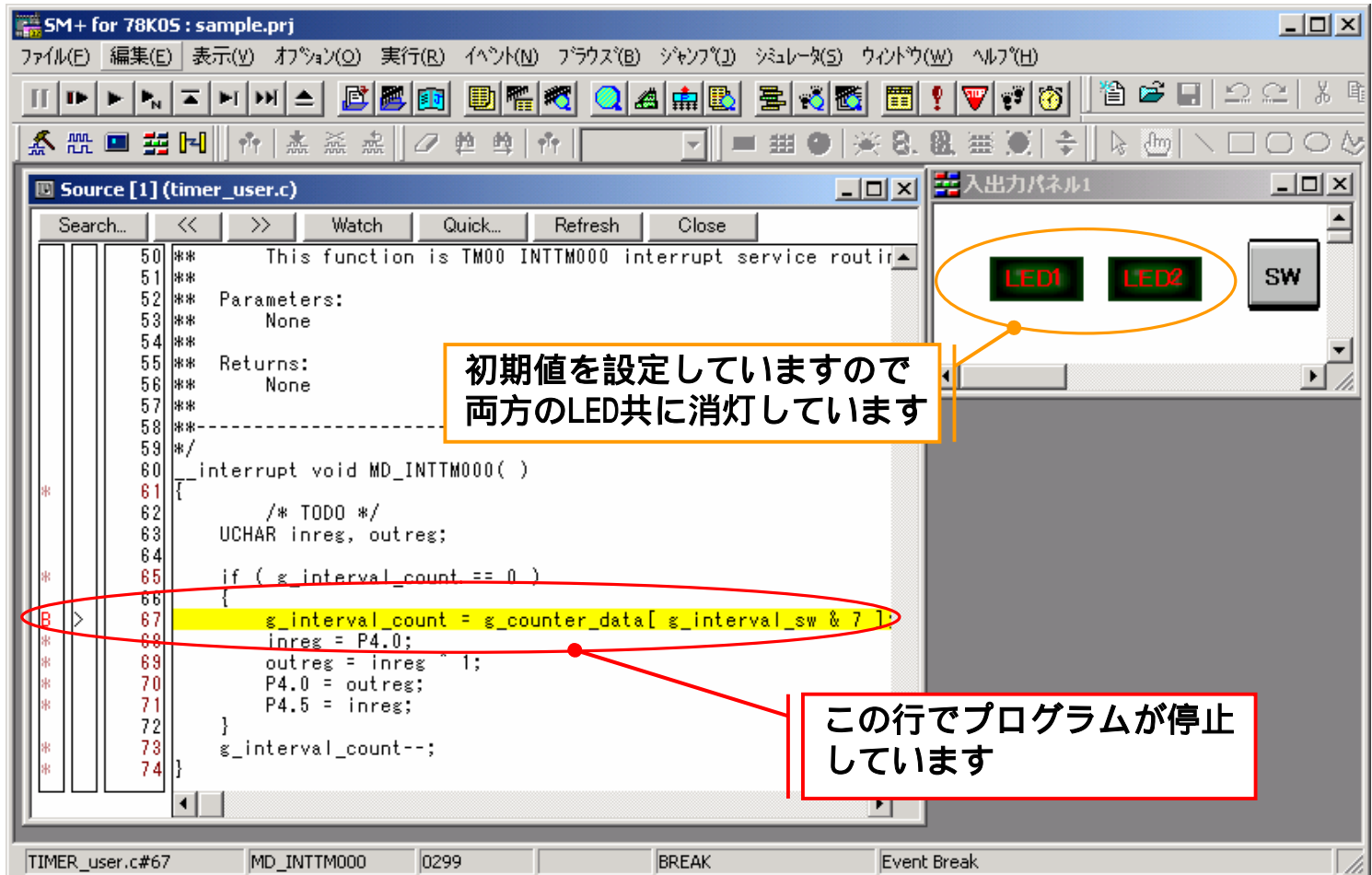
💡ワンポイント

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。SM+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。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。

システム・シミュレータ (SM+) 体験編 (Step4-6)

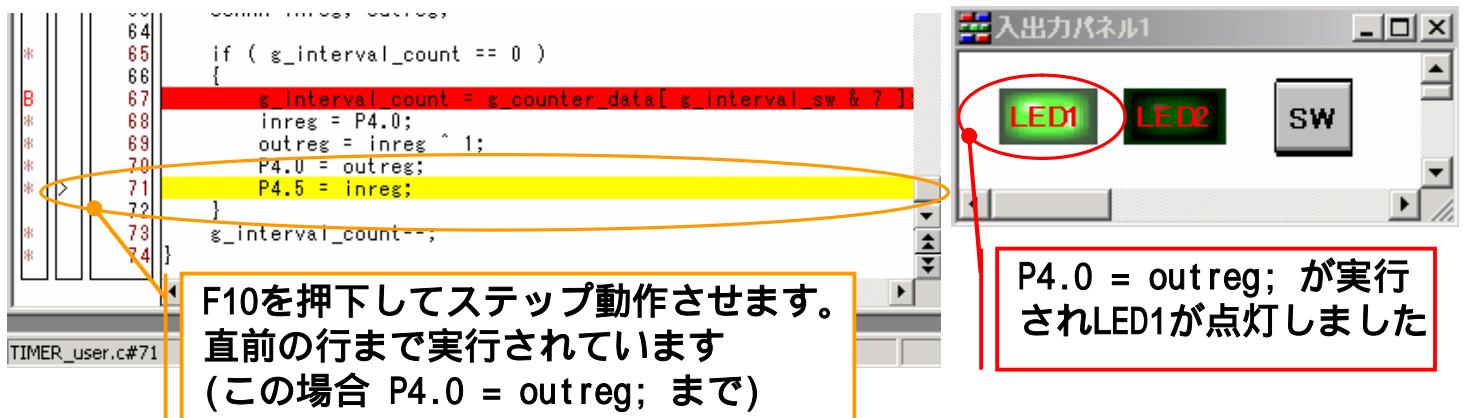
d-3) [Restart] ボタン  を押して、シミュレーションを実行します。
すぐにプログラムはブレーク・ポイントで停止します。



初期値を設定していますので
両方のLED共に消灯しています

この行でプログラムが停止
しています

d-4) この状態でF10を押下してステップ動作させます。71行までステップ動作させると P4.0 = outreg; が実行されLED1が点灯します



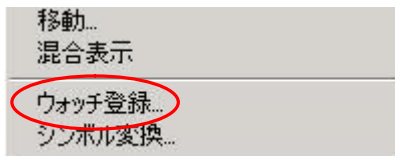
F10を押下してステップ動作させます。
直前の行まで実行されています
(この場合 P4.0 = outreg; まで)

P4.0 = outreg; が実行
されLED1が点灯しました

システム・シミュレータ (SM+) 体験編 (Step4-7)

e. SM+ の基本的な使い方について(ウォッチの登録)

e-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ウインドウへデフォルトの表示方法で追加されます。

e-2) ウォッチ登録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

10進表記に設定します

マウスでドラッグします

```


46 /**
47 /** Abstract:
48 /** This function implements
49 /**
50 /** Parameters:
51 /** None
52 /**
53 /** Returns:
54 /** None
55 /**
56 /**-----
57 /**/
58 void main( void )
59 {
60 /** IODD. add user code */
61 g_interval_sw = 0;
62 g_interval_count = 1;
63 TMO0_Start(); /* Function MD_INTTMO0( ) is called by ev
64 while(1){
65 ;
66 }
67 }
68

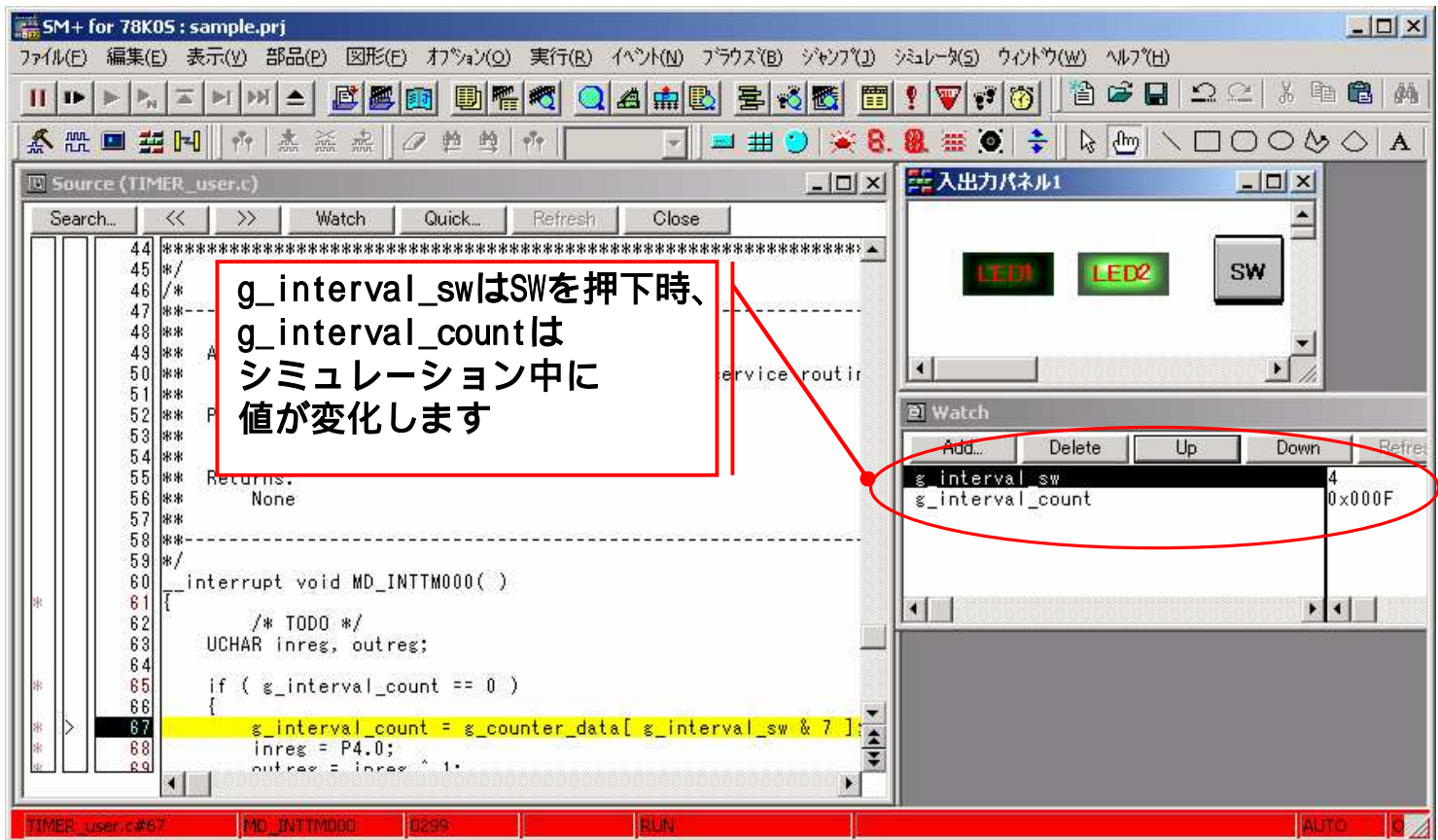
```

e-3) Watchウインドウが開き、変数が表示されます。

g_interval_swは10進表示、
g_interval_countは16進表示
に設定した例です

システム・シミュレータ (SM+) 体験編 (Step4-8)

e-4) [Restart] ボタン  を押して、シミュレーションを実行します。



The screenshot shows the SM+ simulator interface. The main window displays the source code for 'TIMER_user.c'. A red box highlights the following code snippet:

```

44 *****
45 /*
46 /*
47 ***
48 ***
49 ***
50 ***
51 ***
52 ***
53 ***
54 ***
55 *** Returns.
56 *** None
57 ***
58 ***
59 */
60 _interrupt void MD_INTTM000( )
61 {
62     /* TODO */
63     UCHAR inreg, outreg;
64
65     if ( g_interval_count == 0 )
66     {
67         g_interval_count = g_counter_data[ g_interval_sw & 7 ];
68         inreg = P4.0;
69         outreg = inreg + 1;

```

A red arrow points from this code to the Watch window, which shows the following values:

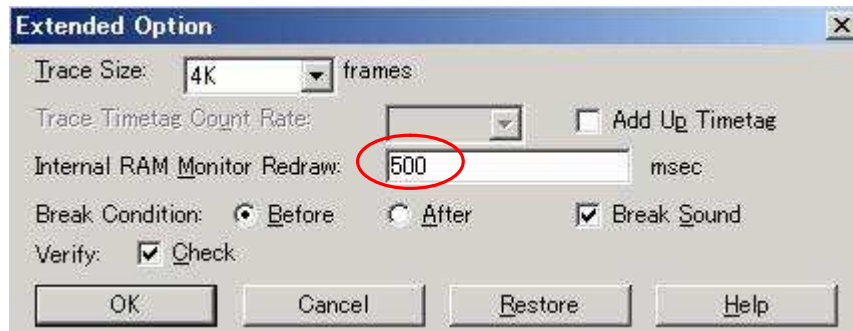
Variable	Value
g_interval_sw	4
g_interval_count	0x000F

The Watch window also shows a diagram of the hardware components: LED1, LED2, and SW.

💡ワンポイント

シミュレーション中の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。



Internal RAM Monitor Redraw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 100 ~ 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。しかしシミュレータ内部の動作としては正しい時間で動作しています。

システム・シミュレータ (SM+) 体験編 (おわり)

システム・シミュレータ (SM+) 体験編は以上です。まだまだシステム・シミュレータ (SM+) には様々な機能があります。

- ・ シリアル信号入出力の確認ができる「シリアル」
 - ・ ポートの波形を表示する「タイミングチャート」
 - ・ ポートの値を変更する「信号データエディタ」
 - ・ デバッグ時の表示に便利「標準入出力」
 - ・ 「入出力パネル」はLEDマトリックス、レベルゲージなどの多彩な部品を用意
- デバッグにかかせない下記機能も搭載されています。
- ・ トレース
 - ・ カバレッジ
 - ・ 条件ブレーク
 - ・ Tcl言語の実行

是非システム・シミュレータ (SM+) の便利な機能を体験してください。

次ページよりターゲット・ボードを使った「ターゲット・ボード体験編」が始まります。

A/D入力、LED、7セグメントLED、ボタン、部品が揃っています

シリアル信号の入出力もエミュレーション可能

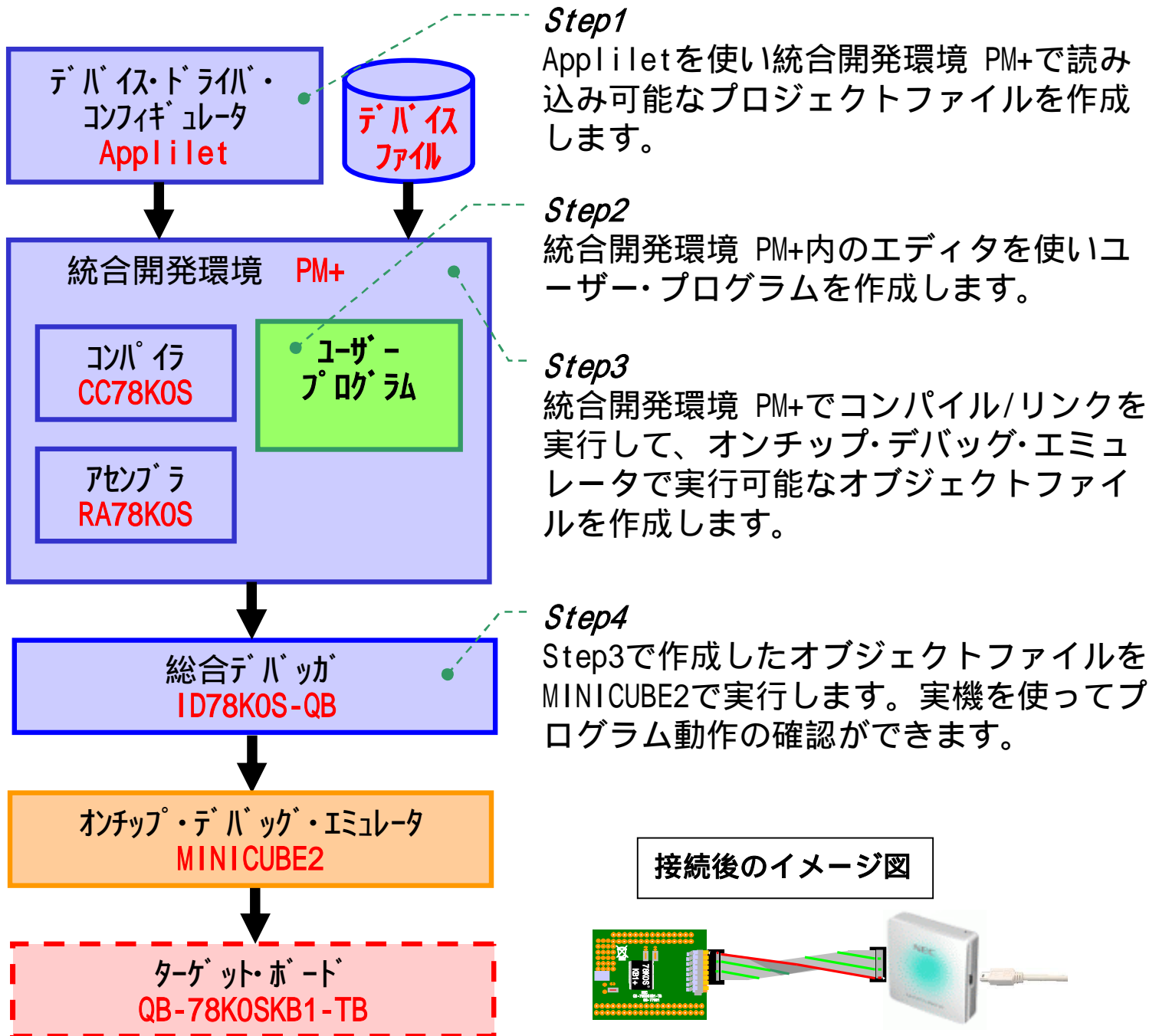
シミュレーションがTcl言語で自動テストできます。テスト結果もファイルに残せます

端子の信号を波形観測できます。また信号の時間も測れます

システム・シミュレータ (SM+) の様々な機能を使用した例

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)

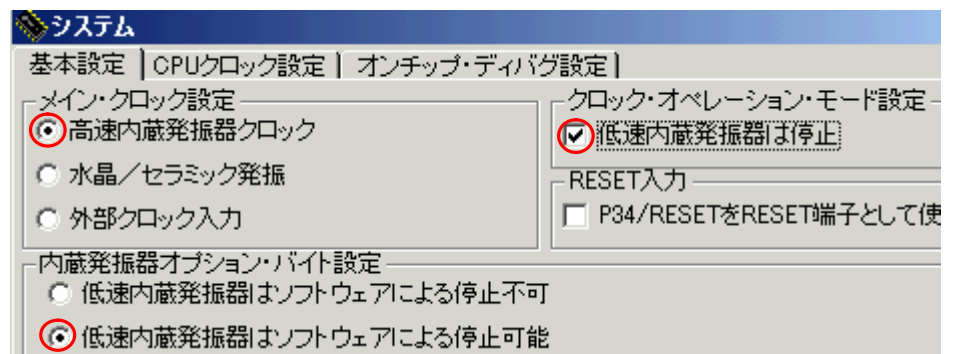
Step1 Appliletを使い統合開発環境 PM+で読み込み可能なプロジェクトファイルを作成し、統合デバッガID78K0S-QBの使い方を学びます。Appliletの使い方については、システム・シミュレータ(SM+)体験編(Step1-1) ~ (Step1-5)も参照してください。

a. サンプル・プログラムで使用する周辺機器を設定します。

システム設定

[基本設定]タブ

“高速内蔵発振器クロック”にチェック。”低速内蔵発振器はソフトウェアによる停止が可能”にチェック。クロック・オペレーション・モード設定の”低速内蔵発振器は停止”にチェック。

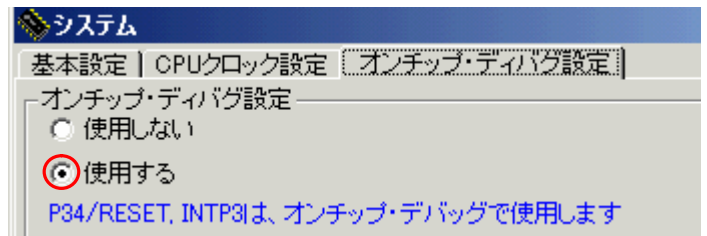


[CPUクロック設定]タブ

CPUクロック選択(MHz)は「8」、周辺クロック(MHz)も「8」で変更ありません。

[オンチップ・ディバグ設定]タブ

“使用する”にチェック。



ターゲット・ボード体験編 (Step1-2)

割り込み設定

[外部割り込み設定]タブ

“INTP0許可”にチェック。有効エッジを”立上がりエッジ”へ変更。



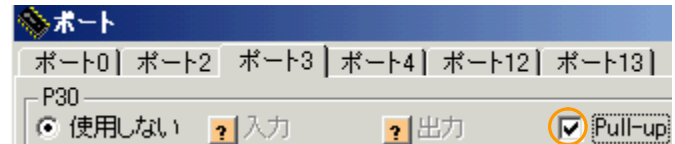
ポート設定

[ポート3]タブ

P30のPull-up(プルアップ)にチェック
P30はINTP0と兼用端子です。INTP0の外部割込の発生をアクティブLOWで設定するため、ここではINTP0のPull-upの設定をします

[ポート4]タブ

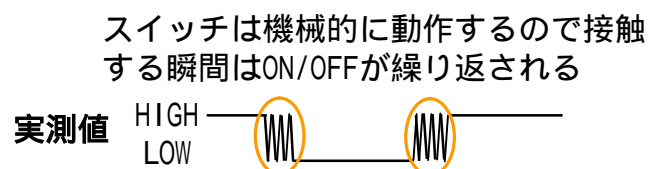
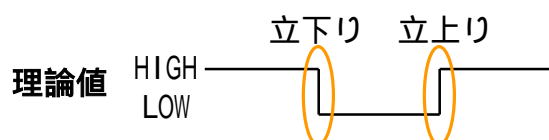
P40、P45の「出力、1」にチェック。
LED1、LED2はそれぞれP40、P45のポートに接続されています。LEDはLOW出力によって点灯するので初期値をHIGHにします。



ワンポイント

有効エッジについて

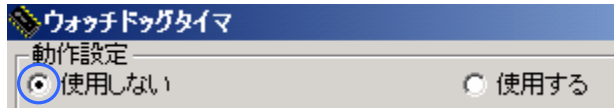
外部割り込み設定で有効エッジを立上りエッジに変更しました。立上りエッジとは信号が0から1へ変化するとき有効とする設定です。ターゲット・ボードのSWを押した時が立下り、押してからSWを離れた時が立上りになります。ソフトウェア・シミュレータでは問題になりませんが実機ですとチャタリングという問題があります。解決方法についてはターゲットボード体験編(Step2-3)を参照して下さい。



ターゲット・ボード体験編 (Step1-3)

ウォッチドッグタイマ設定

使用しないにチェックし、OKを押下して設定します。

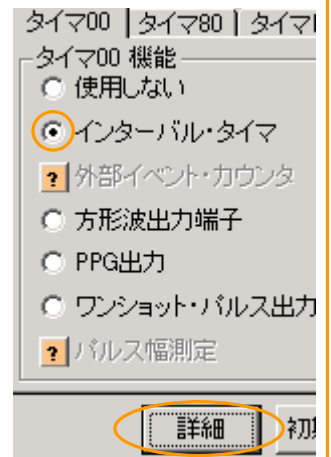
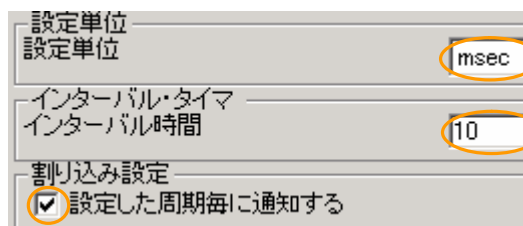


タイマ設定

[タイマ00]タブ

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。

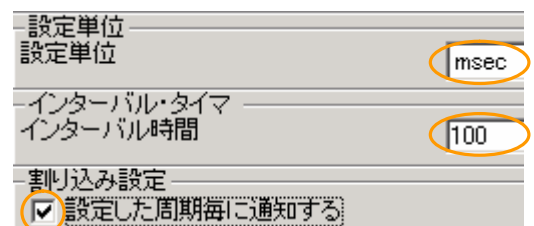
設定単位をmsecへ変更し、インターバル時間を10とします。また、"設定した周期毎に通知する"にチェックします。



[タイマ80]タブ

タイマ00の設定と同様にタイマ80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。また、"設定した周期毎に通知する"にチェックします。



b. ソースコードを自動生成します。

システム・シミュレータ (SM+) 体験編 (Step1-4)

「e. ソースコードを自動生成します」を参照してください。

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. Appliletでソースコードを自動生成した後にPM+で読み込みます。
システム・シミュレータ(SM+)体験編(Step2-1)を参照してください。

b. プログラムを作成します

b-1) main.c

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 )  
{  
    /* TODO. add user code */  
    g_interval_sw = 0;  
    g_interval_count = 1;  
    TM00_Start(); /*Function MD_INTTM000( ) is called by every 10msec*/  
    TM80_Start(); /*Function MD_INTTM80( ) is called by every 100msec*/  
    while(1){  
        ;  
    }  
}
```

b-2) int_user.c

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_INTP0( 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 (Step2-2)

b-3) timer_user.c

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 };

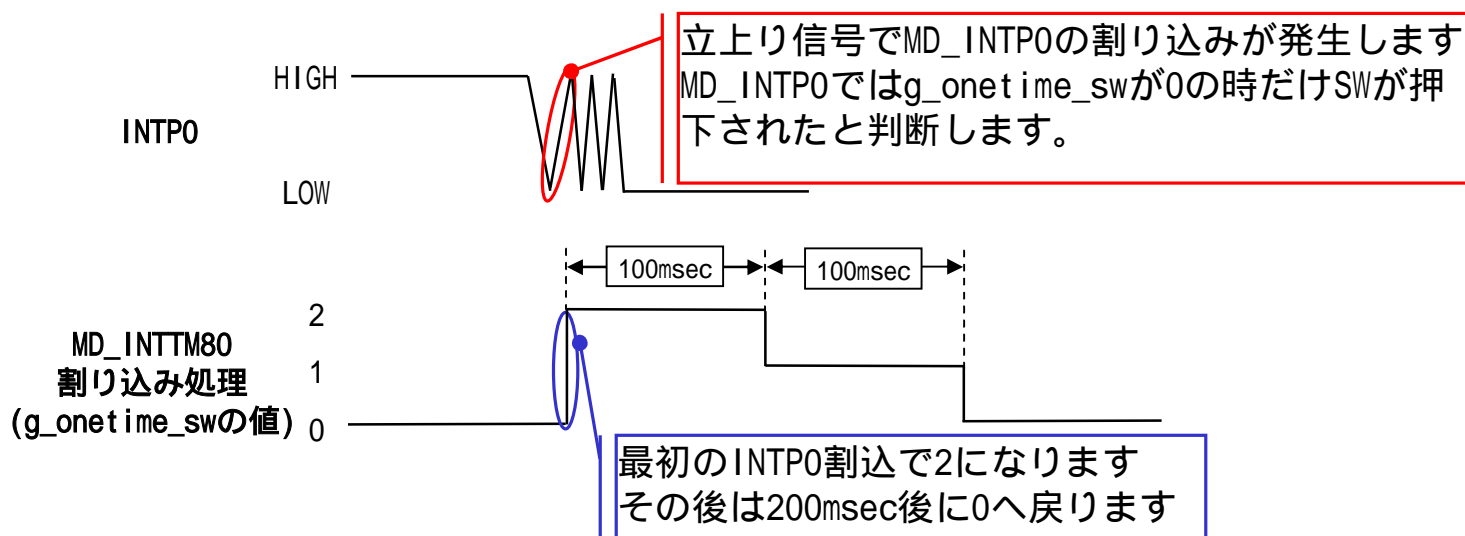
__interrupt void MD_INTTM000( )
{
    /*TODO*/
    UCHAR inreg, outreg;
    if ( g_interval_count == 0 )
    {
        g_interval_count = g_counter_data[ g_interval_sw ];
        inreg = P4.0;
        outreg = inreg ^ 1;
        P4.0 = outreg;
        P4.5 = inreg;
    }
    g_interval_count--;
}

__interrupt void MD_INTTM80( )
{
    /*TODO*/
    if ( g_onetime_sw != 0 )
    {
        g_onetime_sw--;
    }
}
```

ターゲット・ボード体験編 (Step2-3)

d. 処理の説明

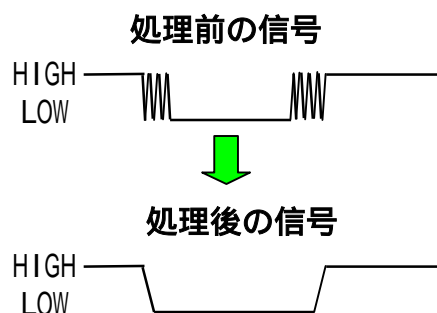
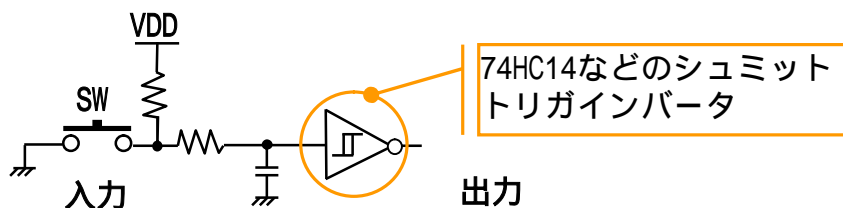
システム・シミュレータ (SM+) 体験編 (Step2-4) の処理説明と同じですが、1点だけ処理の追加があります。100msec毎に呼ばれる関数 MD_INTTM80() のチャタリング防止処理です。外部SWが押下されると g_onetime_sw に2が代入されます。g_onetime_swは100msec毎に-1されます。1度外部SWが押下されるとg_onetime_swが0になるまで200msecかかるわけです。MD_INTPO()の処理は、このg_onetime_swが0でないときLED点滅のタイミングを変更しません。ゆえにチャタリングのために短い時間に何度MD_INTPO()が呼ばれても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。他にハードウェアで行う方法があります。今回ソフトウェアでチャタリングを防止しましたが、ソフトウェアではチャタリングが発生する期間を無視するという処理上から高速な応答を必要とするシステムには使えません。ハードウェアではシュmitt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・システム作成例AではSWにコンデンサを接続し、ハードウェア的に急激なON/OFFをしない簡易な方法でチャタリングを防止しています。



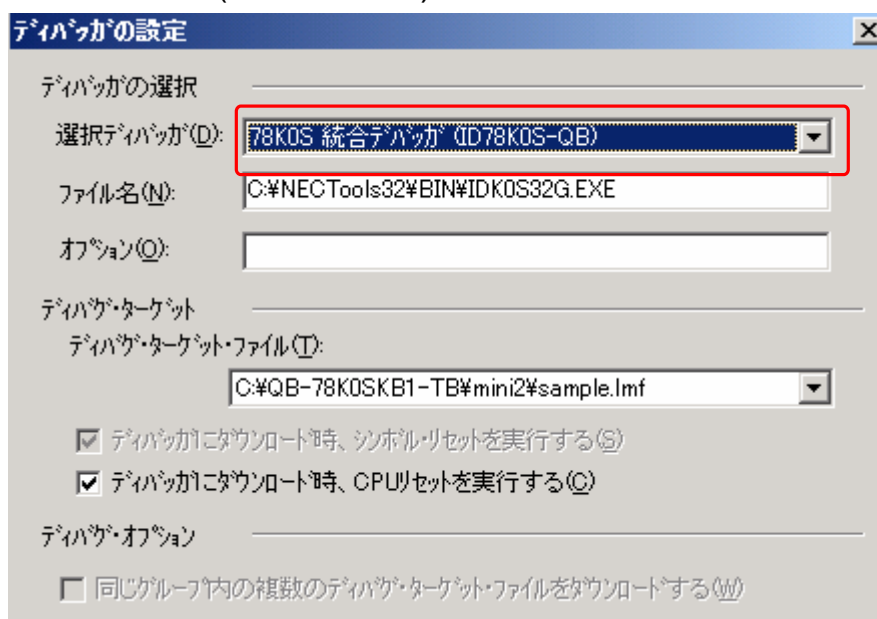
ターゲット・ボード体験編 (Step3)

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“78K0S 総合デバッガ (ID78K0S-QB)”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 -> デバグ (A) F5]または[ビルド -> デバグ (U)]を選択する場合

ビルド->デバグ(A)	F5
ビルド->デバグ(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。

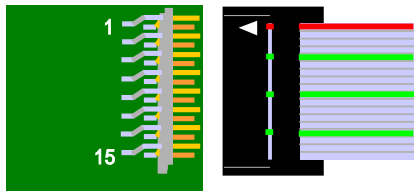
- ・システム・シミュレータ : SM+ for 78K0S/Kx1 システム・シミュレータ
- ・MINICUBE2/IECUBE用デバッガ: 78K0S総合デバッガ (ID78K0S-QB)

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でデバッグします。

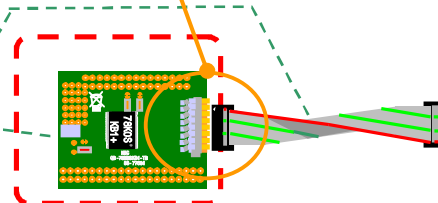
a. MINICUBE2のターゲット・ボードへの接続方法

コネクタの1pinを合わせて接続します



16pinターゲット・ケーブル

ターゲット・ボード



MINICUBE2の設定
が終わるまで接続
しないで下さい



MINICUBE2

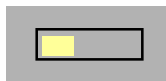
スイッチを設定します

モード選択SW

M1

電源選択SW

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




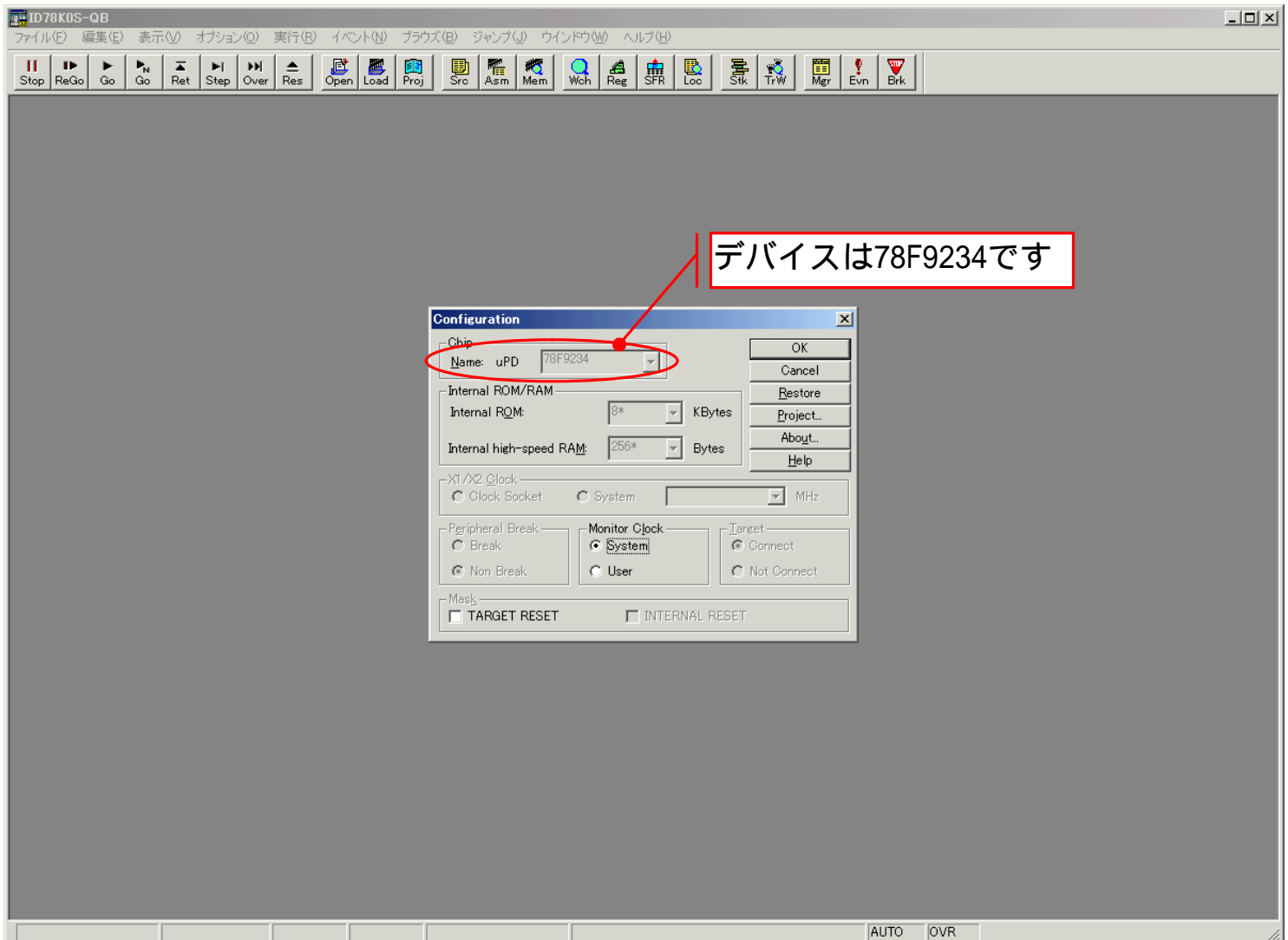
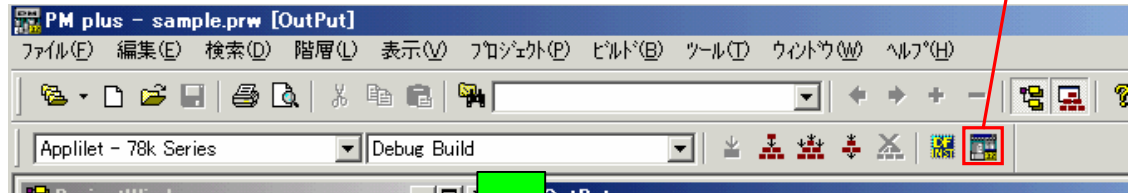
ワンポイント

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の供給はマイコン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. ID78K0S-QB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)] [デバグ (D)]を選択します。



デバイスは78F9234です

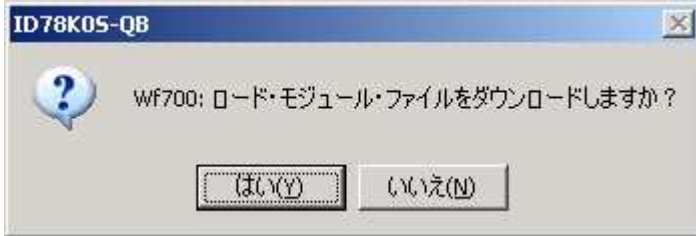
💡ワンポイント

2回目以降の統合デバッガ(ID78K0S-QB)の起動について

総合デバッガ(ID78K0S-QB)で設定した各種の表示(Watchウインドウ、ブレーク・ポイントの設定など)の環境を保存することができます。「*.pri」というファイルにデバッガの設定が保存され、2回目以降のデバッガ起動では自動的に環境を復元します。

ターゲット・ボード体験編 (Step4-3)

- d. OKを押下すると下記ダイアログが表示されます。はい(Y)ボタンの押下でダウンロードが開始されます。



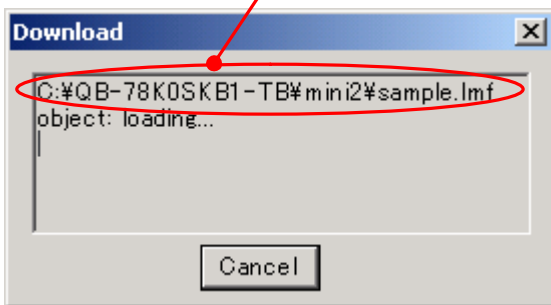
ワンポイント

MINICUE2とターゲット・ボードの接続が成功するとMINICUBE2のLEDが緑に点灯します。



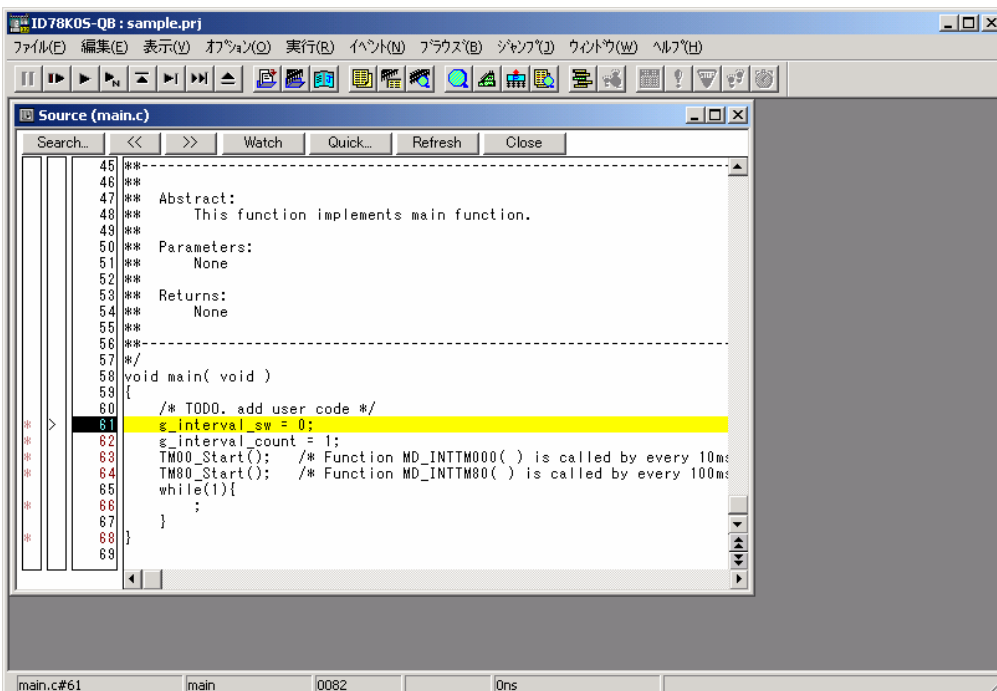
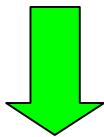
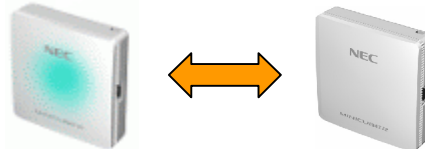
- e. ファイルがダウンロードされ、main.cが開きます

PM+で作成したロードモジュールファイル



ワンポイント

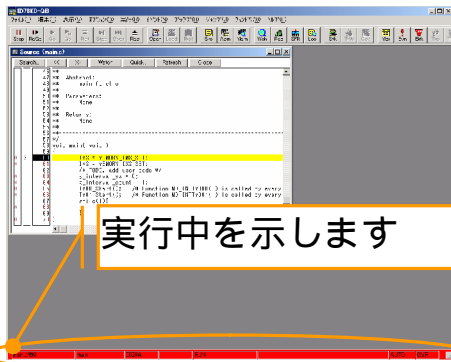
ダウンロードの時間はプログラムのサイズやターゲット・システムとMINICUBE2との接続方法によって変化しますが、このサンプル・プログラムでは10秒ほどです。また、ダウンロード中はMINICUBE2のLEDが1秒間隔で点滅します。



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

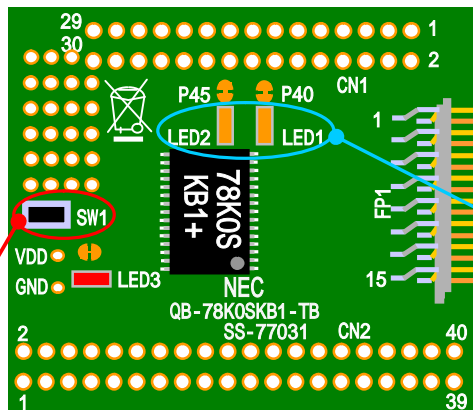
PC画面の表示



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

ターゲット・ボード上の動作



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。

ワンポイント

sample.lmf(ロードモジュールファイル)がダウンロードできない場合について
QB-78K0SKB1-TBではデバッグ機能を実現するためにマイコンに通信/デバッグ用領域を必要とします。プログラムの暴走などによりこの領域にデータが書き込まれるとsample.lmf(ロードモジュールファイル)がダウンロードできない場合があります。今までダウンロード可能だったのが、ダウンロードできなくなった場合は下記の処理を行ってください。

- ・ ID78K0S-QB, PM+などのプログラムを終了します
- ・ [スタート] [プログラム(P)] [NEC Tools32] [OCD Checker]を起動します
- ・ 下記ダイアログを確認し、[Test]ボタンを押下します



名前が正しく表示されていたらTestボタンを押下してください



- ・ “OKダイアログ”が表示されたら完了です。OCD Checkerを終了してください
- ・ ID78K0S-QBでダウンロード可能になっているはずですが

ターゲット・ボード体験編 (おわり)

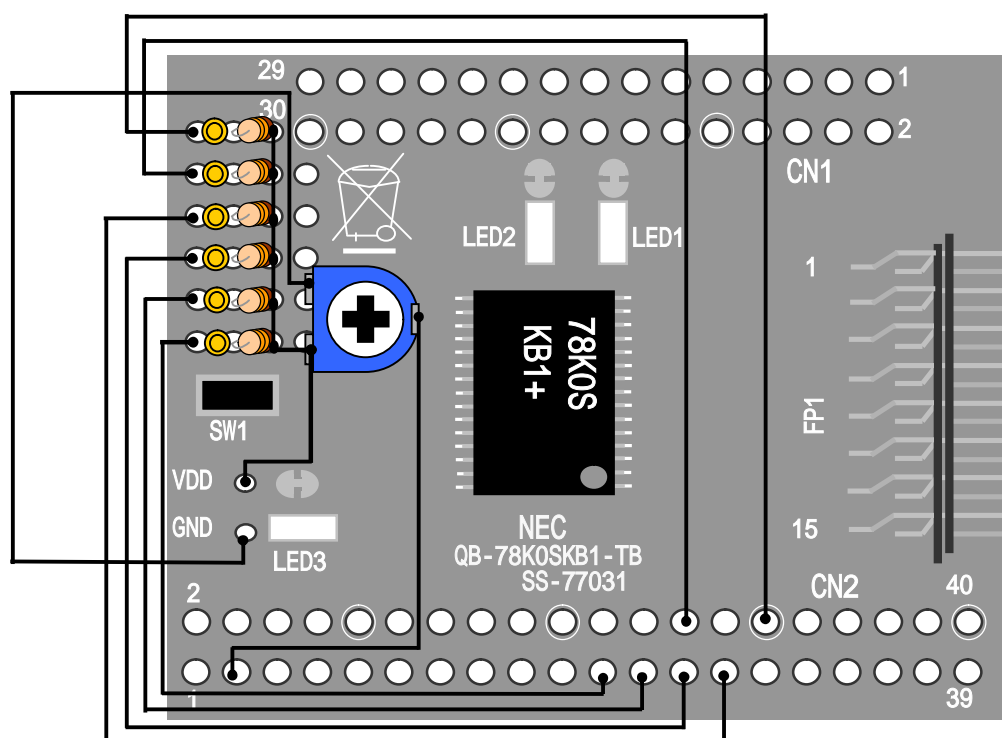
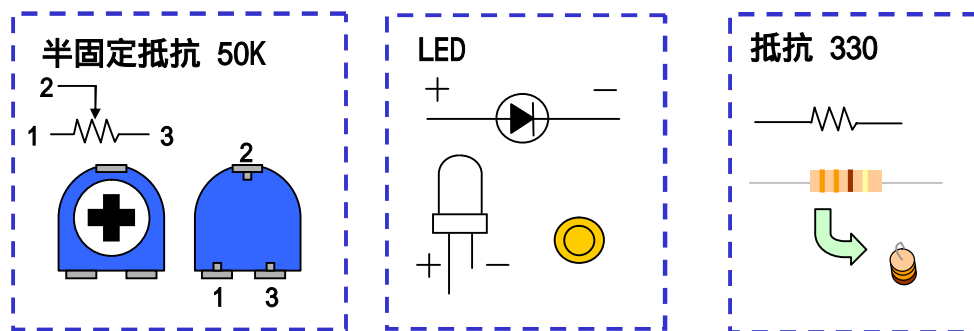
ターゲット・ボード体験編は以上です。MINICUBE2ではハードウェア・ブレイクの数などに制限があります。ブレイク・ポイントを多数設定したい場合はソフトウェア・ブレイクを使用してください。MINICUBE2では実行中の変数表示や、RAMモニタ機能は使えません。その場合はIECUBE, MINICUBE+をお使いください。IECUBEは更にトレース機能を実現した低価格で高機能なエミュレータです。

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、ソフトウェアの工夫次第によっていろいろ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。

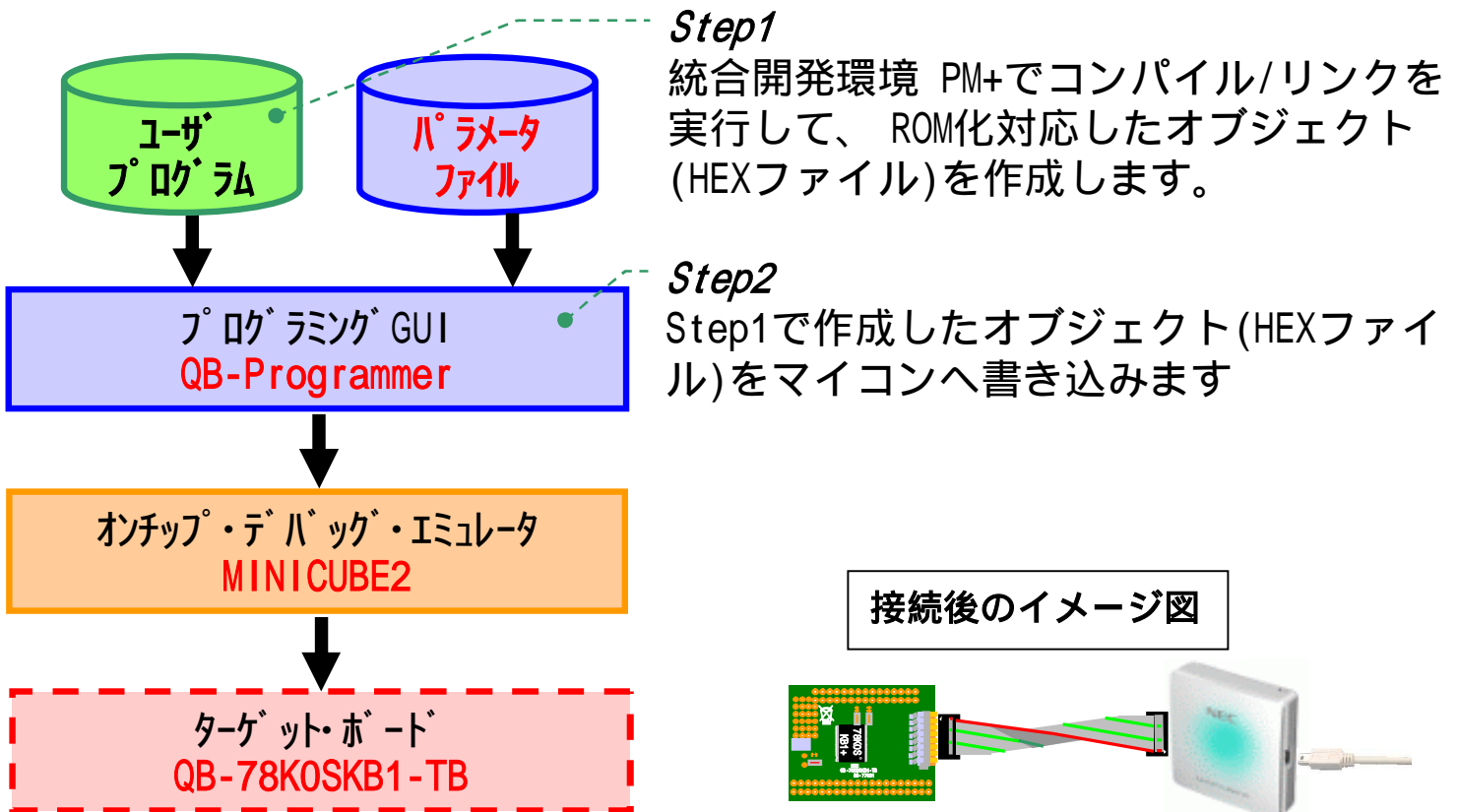
下図はターゲット・ボードに半固定抵抗とLEDを追加した例です。

- ・半固定抵抗をAN10へ接続
- ・LEDをP00 ~ P03, P42, P44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・ボードにプログラミングするまでの手順を説明します。



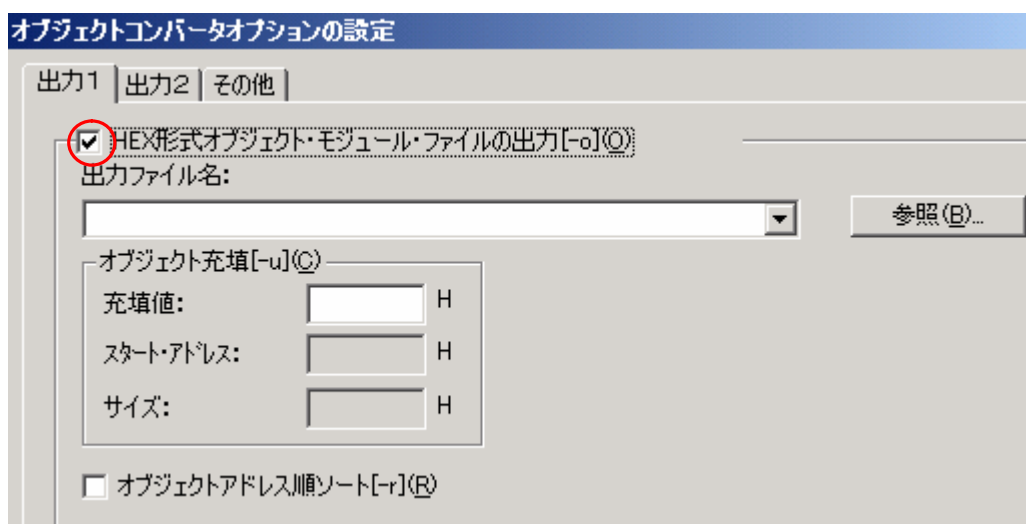
次ページより *Step1*, *Step2*でプログラミング手順を説明します

マイコン・プログラミング体験編 (Step1)

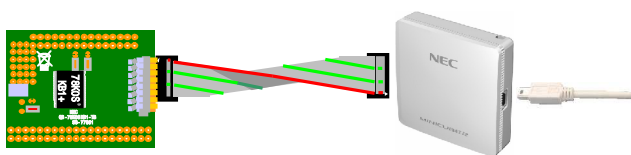
Step1 統合開発環境 PM+でコンパイル/リンクを実行して、ROM化対応したオブジェクト(HEXファイル)を作成します。

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)] [オブジェクトコンバージョンの設定(O)...]を選択します。HEX形式オブジェクト・モジュール・ファイルの出力[-o](O)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。空欄にするとプロジェクト名に「.hex」の拡張子をつけたファイル名が作成されます。



b. MINICUBE2とターゲット・ボードを接続します



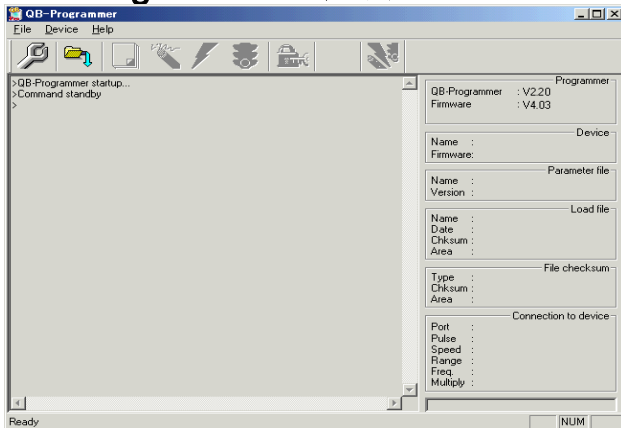
c. パラメータファイルをダウンロードしてください。

本ボード(QB-78K0SKB1-TB)に搭載のマイコン(μ PD78F9234MC)に対応したパラメータファイルを弊社WEBページ「バージョンアップサービス」Lよりダウンロードして、ファイルを解凍してください。

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)をマイコンへ書き込みます。

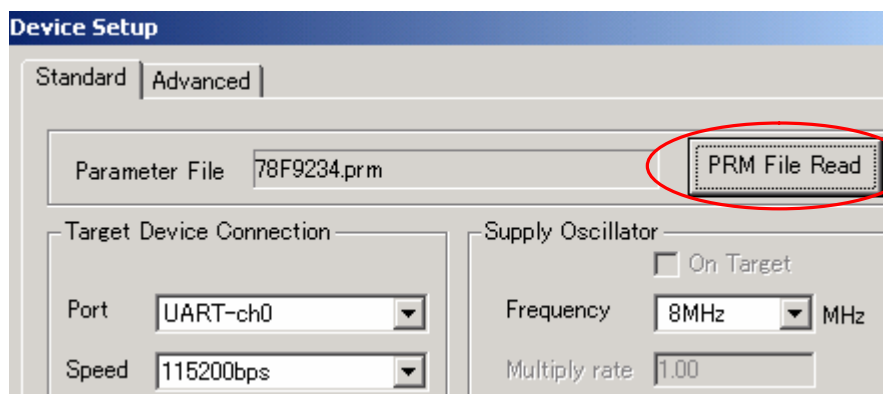
a. QB-Programmerを起動します



MINICUBE2のLEDはアイドルモードより緑点灯へ変わります

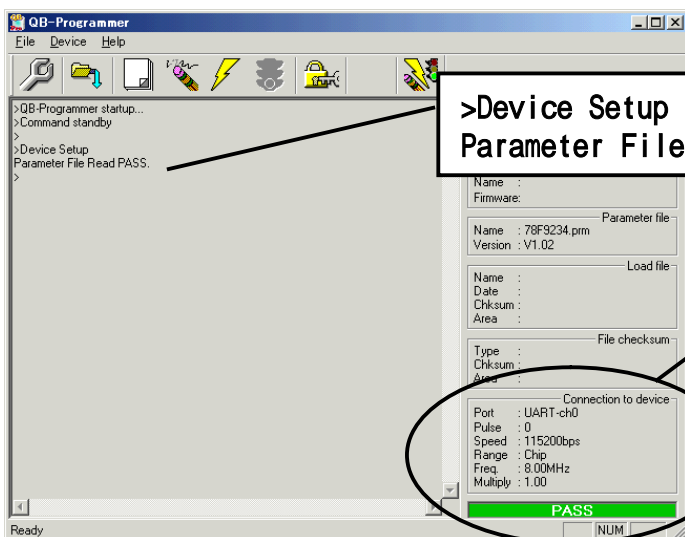
b. パラメータ・ファイルをロードします

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78F9234.prmです。



c. QB-Programmerの画面を確認します

パラメータ・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。



>Device Setup
Parameter File Read PASS.

パラメータ・ファイルの読み込みが成功し各種データが表示されています

マイコン・プログラミング体験編 (Step2-2)

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。ファイルは先ほどPM+で作成した「sample.hex」を指定します。

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 HEX file.
```

e. マイコンを消去します

メニュー・バー[Device] [Erase...]でマイコンを消去します。

マイコンの消去に成功すると下記のメッセージが表示されます。

```
>Erase
Blank check Chip: PASS. Erase skipped.
Erase PASS
```

または

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

f. マイコンにプログラミング(書き込み)します

メニュー・バー[Device] [Program...]でマイコン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
Program PASS
```

プログラム
中は1秒ごと
にLEDが黄色
に点滅します



書き込みに成功すると
LEDが緑点灯します



Programmer	
QB-Programmer	: V2.20
Firmware	: V4.03
Device	
Name	: UPD78F9234
Firmware	: 0.00
Parameter file	
Name	: 78F9234.prm
Version	: V1.02
Load file	
Name	: EXTEND.HEX
Date	: 2006/06/14 16:02:06
Chksum	: 6DA1h
Area	: 000000h-01FFFFh
Connection to device	
Port	: UART-ch0
Pulse	: 0
Speed	: 115200bps
Range	: Chip
Freq.	: 8.00MHz
Multiply	: 1.00

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

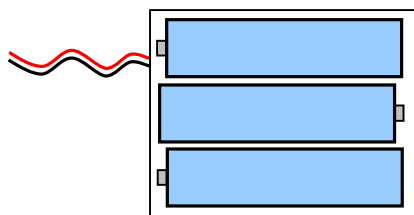
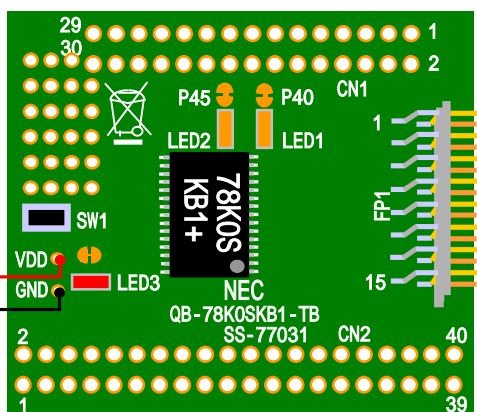
プログラミングについて

マイコンの消去/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。

マイコン・プログラミング体験編 (おわり)

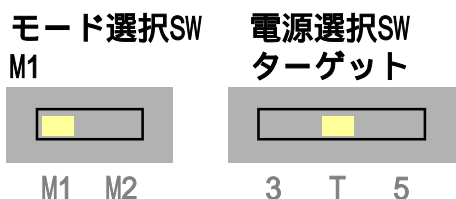
マイコン・プログラミング体験編は以上です。プログラミングしたターゲット・ボードは電源を供給すれば動作します。USBポートより電源のみを供給するケーブルを自作するか、または乾電池を3本直列で供給してもよいでしょう。

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5.5Vまでなので乾電池なら3本の使用します。もし乾電池4本使うのなら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、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の作成します。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。

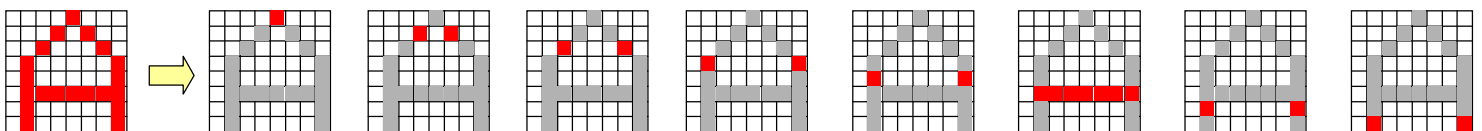
ターゲット・ボードを使用した回路を作成します。ここでは8x8のドットマトリクスLEDをダイナミック点灯させます。ダイナミック点灯とは表示を分割して(今回は8個のLED)行う方法です。表示を高速に順次繰り返す事によって人の目には全てが表示されているように見えます。

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

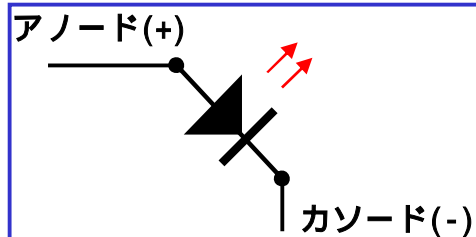
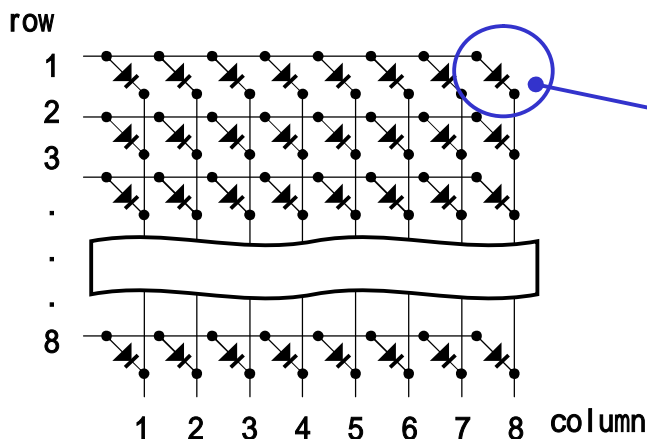
高速に表示を繰り返す



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。

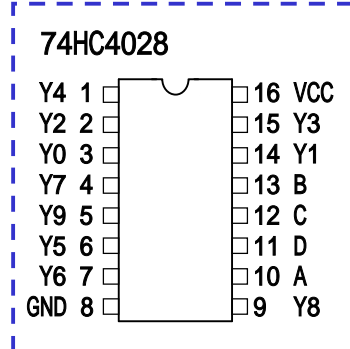
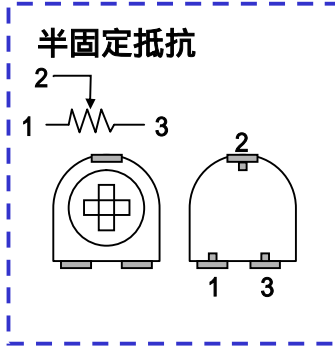
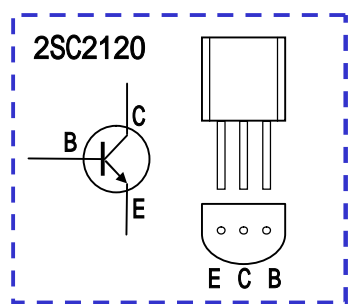
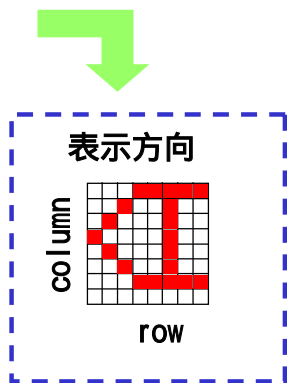
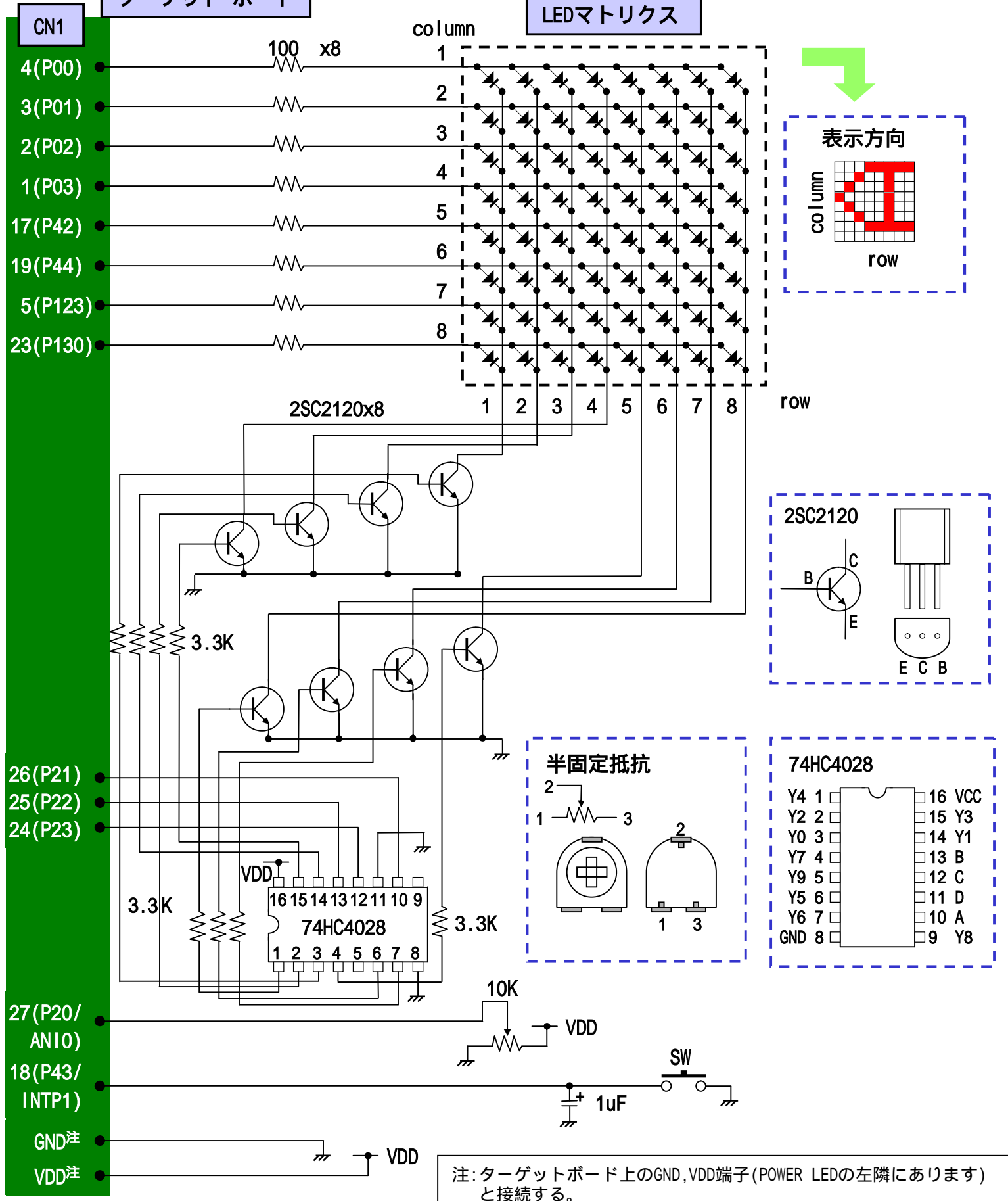


例えば四角内のLEDを点灯させるにはrow1に+、column8に-を接続すれば点灯します。LEDを1つ点灯させるには通常10mA ~ 20mA必要です。これを+5Vで供給するには抵抗150 ~ 300 を接続します。マトリクスLEDは1列に8個接続されていますので80mA ~ 160mAが必要です。これを+5Vで供給するには抵抗37.5 ~ 75 を接続します。しかし、マイコンの1ポートに80mA ~ 160mAを流せませんので何らかの回路(トランジスタ制御など)が必要になります。

ターゲット・システム作成例A (その2)

ターゲット・ボード

LEDマトリクス



ターゲット・システム作成例A (その3)

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P21～P23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(AN10)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

78K0S/KB1+のポートはソース電流で10mA(端子合計44mA)、シンク電流で20mA(端子合計44mA)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})

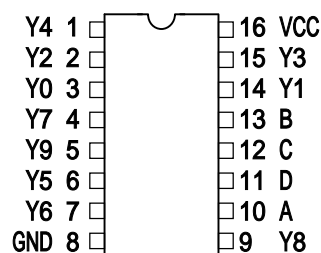


74HC4028について

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)

a. プログラムを作成します。まず、Appliletで設定します。

システム設定

[基本設定]タブ

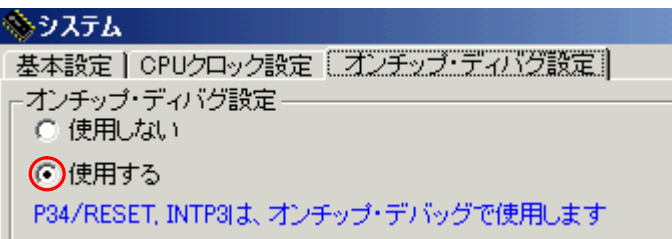
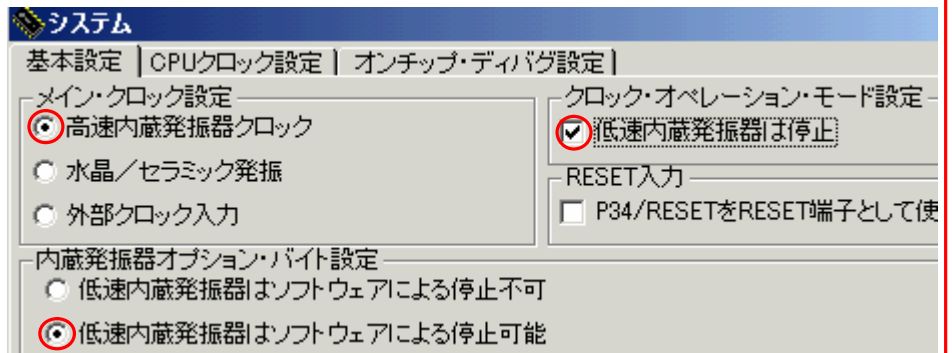
“高速内蔵発振器クロック”にチェック。“低速内蔵発振器はソフトウェアによる停止が可能”にチェック。クロック・オペレーション・モード設定の“低速内蔵発振器は停止”にチェック。

[CPUクロック設定]タブ

CPUクロック選択(MHz)は「8」、周辺クロック(MHz)も「8」で変更ありません。

[オンチップ・デバッグ設定]タブ

“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

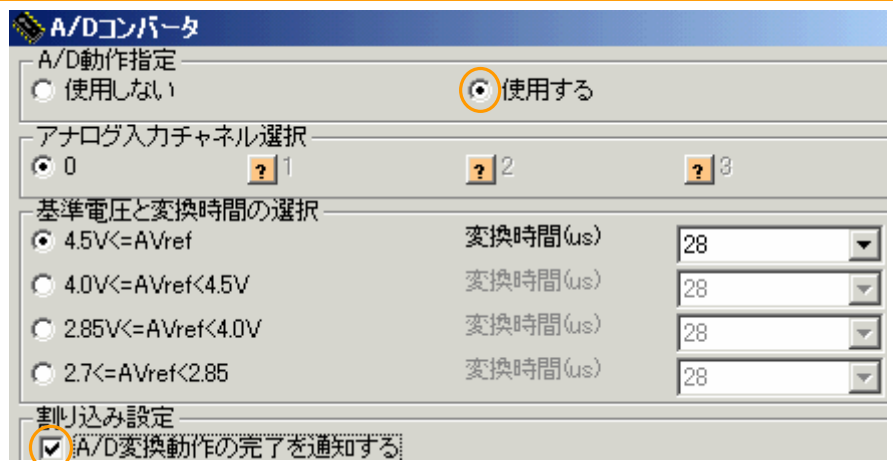
A/Dコンバータ設定

[A/D動作指定]

“使用する”にチェック。

[割り込み設定]

“A/D変換動作の完了を通知する”にチェック。



ワンポイント

Applilet起動について

システム・シミュレータ(SM+)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「78K0KB1+」、チップ名「μPD78F9234」で作成しています。

ターゲット・システム作成例A (その5)

タイマ設定

[タイマ00]タブ

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1とします。また、"設定した周期毎に通知する"にチェックします。

[タイマ80]タブ

タイマ00と同様にインターバル・タイマにチェックします。

設定単位をmsecへ変更し、インターバル時間を1とします。また、"設定した周期毎に通知する"にチェックします。



ウォッチドッグタイマ設定

"使用しない"にチェック。

ポート設定

[ポート2]タブ

P21 ~ P23の"出力"にチェック。

P21	<input type="radio"/> 使用しない	<input type="radio"/> 入力	<input checked="" type="radio"/> 出力
P22	<input type="radio"/> 使用しない	<input type="radio"/> 入力	<input checked="" type="radio"/> 出力
P23	<input type="radio"/> 使用しない	<input type="radio"/> 入力	<input checked="" type="radio"/> 出力

[ポート13]タブ

P130の"出力"にチェック。

P130	<input type="radio"/> 使用しない	<input checked="" type="radio"/> 出力	<input type="checkbox"/> 1
------	-----------------------------	-------------------------------------	----------------------------

[ポート2]タブ

P123の"出力"にチェック。

P121	<input checked="" type="radio"/> 使用しない	<input type="radio"/> 入力	<input type="radio"/> 出力
P122	<input checked="" type="radio"/> 使用しない	<input type="radio"/> 入力	<input type="radio"/> 出力
P123	<input type="radio"/> 使用しない	<input type="radio"/> 入力	<input checked="" type="radio"/> 出力

[ポート4]タブ

P40, P42 ~ P45の"出力"にチェック。P43のPull-upにチェック。

P40	<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 type="checkbox"/> 1
P41	<input checked="" type="radio"/> 使用しない	<input type="radio"/> 入力	<input type="radio"/> 出力	<input type="checkbox"/> Pull-up	<input type="checkbox"/> 1
P42	<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 type="checkbox"/> 1
P43	<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	<input type="checkbox"/> 1
P44	<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 type="checkbox"/> 1
P45	<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 type="checkbox"/> 1

[ポート0]タブ

P00 ~ P03の"出力"にチェック。

P00	<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 type="checkbox"/> 1
P01	<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 type="checkbox"/> 1
P02	<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 type="checkbox"/> 1
P03	<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 type="checkbox"/> 1

ターゲット・システム作成例A (その6)

b. 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

```

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

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
};

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, 0x04, 0x11, 0x11
, 0x00, 0x11, 0x11, 0x0A, 0x04, 0x04, 0x04, 0x04
, 0x00, 0x1F, 0x01, 0x02, 0x04, 0x08, 0x10, 0x1F
};

const UCHAR g_textdata[][ D_MAXTEXT ] =
{ /* text data */
  "THIS IS 78KOSKB1 TARGET BOARD "
, "DISPLAY CHANGES IF SW IS PUSHED "
, "THIS IS PROGRAM SAMPLE OF NECEL "
, 0
};

```

main.c(リスト2)

```

/* ---- 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)

c. 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 )
{
    /* Target-Board LED off */
    P4.0 = 1;
    P4.5 = 1;

    initialize_value();

    /* initialize for text scroll */
    initialize_text();

    /* start timer */
    TM00_Start();
    TM80_Start();

    while( 1 )
    {
        ;
    }
}

```

macrodrive.h(リスト1)

リスト省略

```

#define SYSTEMCLOCK      8000000
#define SUBCLOCK         0
#define MAINCLOCK       8000000
#define FXPCLOCK        8000000
#define FRCLOCK         240000

#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 */

#endif

```

ターゲット・システム作成例A (その8)

d. 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"

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 ];

/* ---- MATRIX LED display column ---- */
void disp_line( UCHAR _data )
{
    P0 = _data & 0xf;
    P4.2 = (_data & 0x10) >> 4;
    P4.4 = (_data & 0x20) >> 5;
    P12.3 = (_data & 0x40) >> 6;
    P13.0 = (_data & 0x80) >> 7;
}

/*
** -----
** Abstract:
** This function is TM00 INTTM000 interrupt
service
** Parameters:
** None
**
** Returns:
** None
** -----
*/
__interrupt void MD_INTTM000( )
{
    UCHAR row, line;

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

    /* display column */
    disp_line( 0 );
    line = g_matrix_ram[ 0 ][ row ];
    disp_line( line );

    /* display row */
    P2.1 = ( row & 1 );
    P2.2 = (( row & 2 ) >> 1);
    P2.3 = (( row & 4 ) >> 2);
}
```

timer_user.c(リスト2)

```
g_timer0_counter++;
}

/*
** -----
** Abstract:
** This function is TM80 INTTM80 interrupt
service routine.
** Parameters:
** None
**
** Returns:
** None
** -----
*/
__interrupt void MD_INTTM80( void )
{
    UCHAR font;

    AD_Start( ADChannel0 );

    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)

e. 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)

```
#pragma interrupt INTAD MD_INTAD

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

extern USHORT g_text_speed;

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

```
#pragma interrupt INTP1 MD_INTP1

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

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 )
{
    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)

f. 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

`__interrupt void MD_INTTM000()`
 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_INTTM80()`
 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。

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

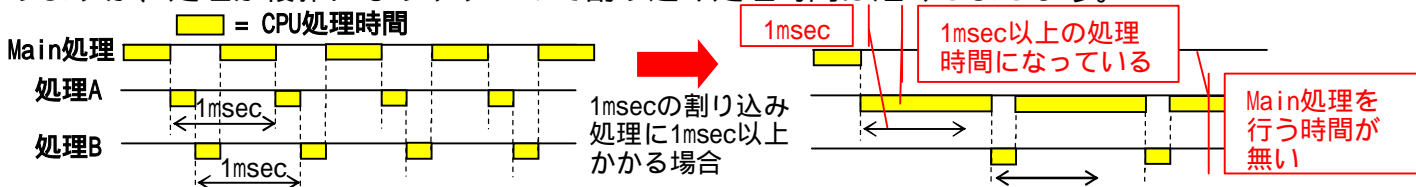
int_user.c

`__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。

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)

g. 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

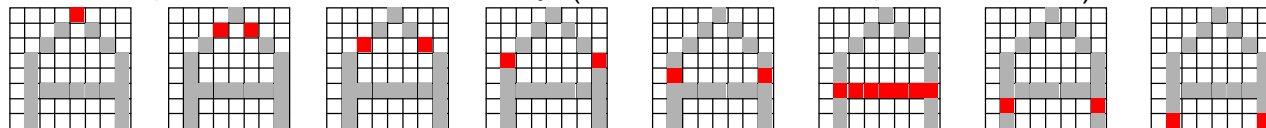
1. メイン処理
2. タイマ00(インターバル・タイマとして使用)
3. タイマ80(インターバル・タイマとして使用)
4. A/Dコンバータ完了割り込み
5. INTP1外部割り込み

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。
処理としては電源ON後に1度しか動きません。後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80 `__interrupt void MD_INTTM80()`

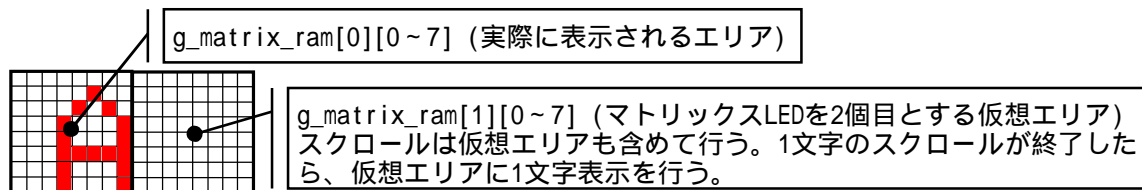
1msec毎に起動します。

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。

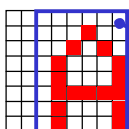
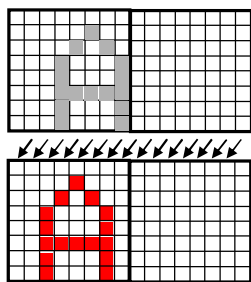
A/Dコンバートの結果は `g_text_speed` へ反映されるので、それを元にスクロールスピードを決めます
タイマ01は呼ばれる毎に `g_timer1_counter` を+1していますので、`g_text_speed`と`g_timer1_counter`
の値を比較してスクロールを行うか行わないかを判断しています。

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

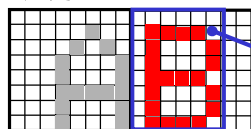


a. マトリックスLEDへ表示されているデータを1ドット左へ移動する



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01で開始されたA/D変換が完了したときに呼ばれます。

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。

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。

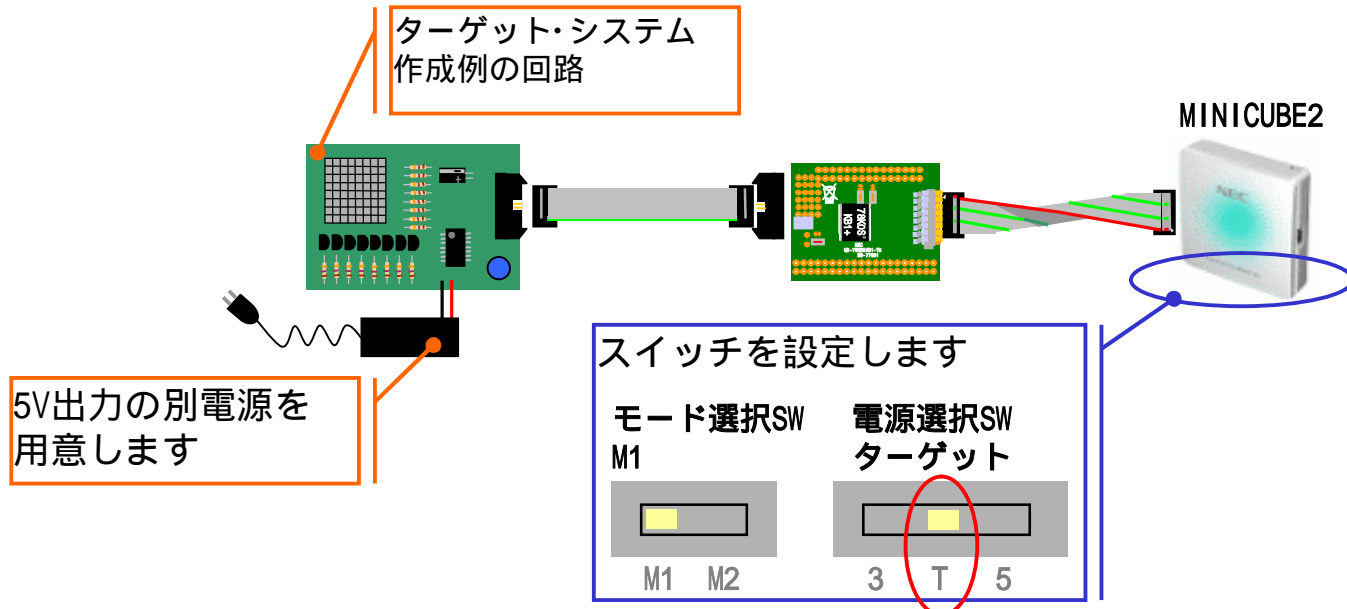
スクロールするテキストを変えます。表示させるテキストは1文が32文字で構成されています。

`g_textdata[][D_MAXTEXT]` の内容を書き換えればマトリックスLEDへ表示させる変える事ができますが、表示させるデータは必ず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。



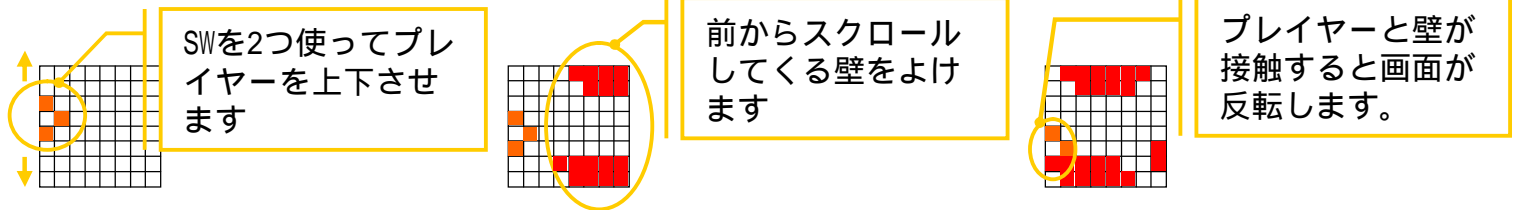
下記の順番で接続を行ってください。順番を間違えるとターゲット・システムを壊す原因となります

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

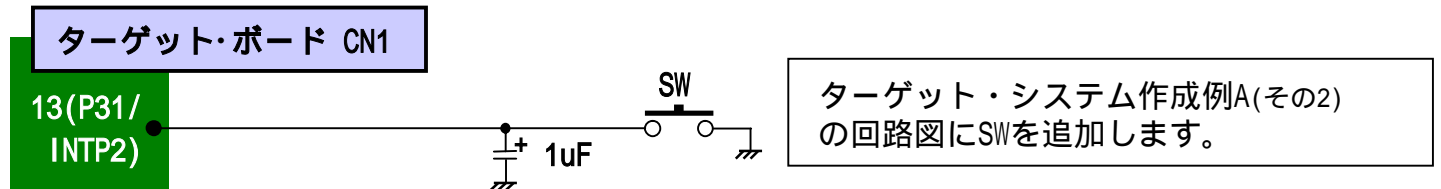
ターゲット・システム作成例B (その1)

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説明します。

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。



a. Appliletの設定を追加します

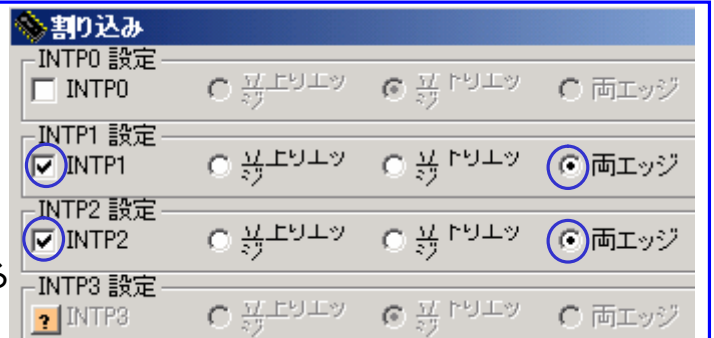
Appliletの設定は、ターゲット・システム作成例A(その4)、(その5)と同様の設定を行います。更に下記の設定を追加します。



割り込み設定

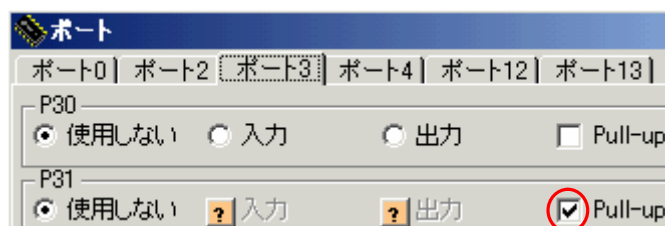
[外部割り込み設定] タブ

“INTP1許可”, “INTP2許可” にチェックし、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を離したときに割り込みが発生します。
 また、[INTP3]はオンチップ・デバッグで使用するポートなので、設定できません。



ポートの設定

[ポート3] タブ



P31のPull-upにチェック。P31はINTP2の外部割り込みと兼用端子ですので、Pull-upの設定は“ポート”項目で行います。

ターゲット・システム作成例B (その2)

b. 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

```

/* MacroDefine */
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,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,
    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,

    0x00,0x00,0x08,0x0e,0x06,
    0x3e,0x30,0x83,0xe0,0x1e,0x1f,0x80,0x70,
    0x0e,0x00,0x1f,0xff,0xff,0xff,0xff,
    0xff,0xff,0xff,0xf8,0x00,0x00,0x00,
    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,
    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,
    0x00,0x40,0x00,0x00,0x00,0x00,0x00,
    0x20,0x00,0x00,0x00,0x40,0x00,0x00,0x41,

    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,
    0xe0,0x00,0x40,0x00,0x00,0x00,0x40,0x04,
    0x00,0x00,0x04,0x00,0x00,0x10,0x08,0x00,
    0x04,0x10,0x02,0x04,0x00,0x88,0x00,0x00,

```

main.c(リスト2)

```

    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,
    0x01,0x80,0x00,0x01,0x80,0x00,0x00,0x60,
    0x0c,0x03,0x00,0x03,0x00,0x00,0x03,0xe0,
    0x03,0xc0,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x3f,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,
    0x00,0x00,0x00,0x01,0xff,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,
    0x00,0x00,0xe0,0x00,0x00,0x00,0x00,0x7f,
    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,
    0x00,0x00,0x00,0x01,0x00,0x02,0x00,0x00,
    0x00,0x02,0x00,0x04,0x00,0x01,0x00,0x00,
    0x40,0x00,0x08,0x00,0x00,0x00,0x40,0x00,

    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,
    0x00,0x00,0x00,0x30,0x00,0x18,0xc0,0x03,
    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,

```


ターゲット・システム作成例B (その3)

c. 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)

```

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,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
};

/* 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 = 220;

/* INTP1, INTP2 status */
UCHAR g_intp1_sw;
UCHAR g_intp2_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;

```

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 = MD_FALSE;
    g_bump_counter = 3000;
}

/* maze 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;
    }
}

void main( void )
{
    /* Target-Board LED off */
    P4.0 = 1;
    P4.5 = 1;

    initialize_value();

    /* start timer */
    TMO0_Start();
    TM80_Start();

    while( 1 )
    {
        ;
    }
}

```

ターゲット・システム作成例B (その4)

d. 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

macrodrive.h(リスト1)

```

                                リスト省略
#define SYSTEMCLOCK             8000000
#define SUBCLOCK                 0
#define MAINCLOCK                8000000
#define FXPCLOCK                 8000000
#define FRCLOCK                  240000

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

#endif

```

timer_user.c(リスト1)

```

                                リスト省略
#include "macrodriver.h"
#include "timer.h"
#include "ad.h"

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_realam[ D_MLED_ROW ];
extern UCHAR g_intp1_sw;
extern UCHAR g_intp2_sw;
extern UCHAR g_plane_location;
extern BOOL g_plane_bump;

/* ---- MATRIX LED display column ---- */
void disp_line( UCHAR _data )
{
    P0 = _data & 0xf;
    P4.2 = (_data & 0x10) >> 4;
    P4.4 = (_data & 0x20) >> 5;
    P12.3 = (_data & 0x40) >> 6;
    P13.0 = (_data & 0x80) >> 7;
}

/*
** -----
** Abstract:
** This function is TMOO INTTM000 interrupt
service routine.
** Parameters:
** None
** Returns:
** None
*/

```

timer_user.c(リスト2)

```

__interrupt void MD_INTTM000( )
{
    UCHAR row, line, pdat;

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

    /* display column */
    disp_line( 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;
        }

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

    /* display row */
    P2.1 = ( row & 1 );
    P2.2 = (( row & 2 ) >> 1);
    P2.3 = (( row & 4 ) >> 2);
    g_timer0_counter++;
}

/*
** -----
** Abstract:
** This function is TM80 INTTM80 interrupt
service routine.
** Parameters:
** None
** Returns:
** None
** -----
*/

```

ターゲット・システム作成例B (その5)

e. 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3)

```

__interrupt void MD_INTTM80( )
{
  UCHAR dat;
  UINT i;

  AD_Start( ADChannel0 );

  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--;
      }
      g_matrix_realram[ i ] = dat;
    }
    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 position */
      if ( g_intp2_sw )
      {
        if ( g_plane_location > 0 )
        {
          g_plane_location--;
        }
      }
    }
  }
}

```

timer_user.c(リスト4)

```

    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];
    }
}

```

ad_user.c(リスト1)

```

#pragma interrupt INTAD MD_INTAD

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

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6)

f. 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

int_user.c(リスト1)

```
#pragma interrupt INTP1 MD_INTP1
#pragma interrupt INTP2 MD_INTP2

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

extern UCHAR g_intp1_sw;
extern UCHAR g_intp2_sw;

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

/*
** -----
** Abstract:
** INTP1 Interrupt service routine.
** Parameters:
** None
** Returns:
** None
** -----
*/
__interrupt void MD_INTP1( void )
{
    g_intp1_sw = P4.3 ^ 1;
}

/*
** -----
** Abstract:
** INTP2 Interrupt service routine.
** Parameters:
** None
** Returns:
** None
** -----
*/
__interrupt void MD_INTP2( void )
{
    g_intp2_sw = P3.1 ^ 1;
}
```

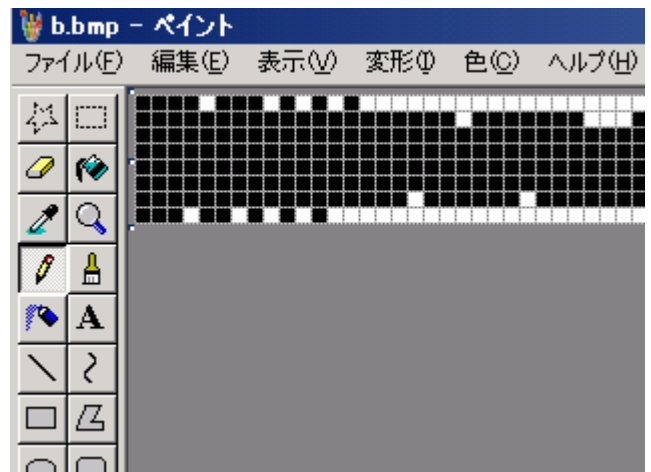


ワンポイント

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ます。まず画像を作成します

- ・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

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。

} x 8

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7)

g. 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;
```

衝突した時にブザーを鳴らす時間です

```
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_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ックスLEDの非表示領域へ出力します

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

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

```
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_INTP2( void )
```

外部割り込みINTP2に呼ばれる関数です。

SWの値を g_intp2_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_INTTM80()
```

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

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。

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初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。

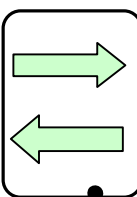
💡 ワンポイント

プログラムの共通化について

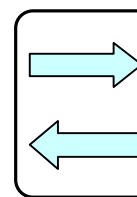
Appliletを使用するとプログラムを共通化するのが楽になります。ここでいう共通化とはCPUを変えた場合を指します。例えば78K0S/KB1+(8bitマイコン)からV850ES/HG2(32bitマイコン)へ変更してもプログラムはそれほど変化なく作成できます。QB-V850ESHG2-TBのクイック・スタート・ガイドと比べてください。電光掲示板プログラムで変わる点は `_interrupt void MD_INTTM000(void)` などのポート制御部分です。またポート制御も `define` 定義すれば、もっと共通化できます。

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。

