mruby/c の PIC24 へのポーティング手順について

しまねソフト研究開発センター

1.はじめに

mruby/c を Microchip 社製1chip マイコン PIC24 シリーズ用の Explorer 16/32 Development Board 上で動作させるための手順を記述する。

2.使用機材及び開発環境

2.1 使用機材

Explorer 16/32 Development Board (P/N DM240001)

PIC24FJ256GB210搭載 Explorer 16/32 用ドータボード (P/N MA240021)

windows 8.1

2.2 開発環境

MPLAB X Version 5.0.5 (NetBeans ベースの統合開発環境)

MPLAB XC16 Compiler Version 1.35 (Free License を使用)

MPLAB Code Configurator Version 3.65.1

3.開発環境の構築方法

3.1 MPLAB ツール群のインストール

1.ダウンロードサイト

MPLAB X

下記 URL の画面下方のダウンロードからダウンロードする。

https://www.microchip.com/mplab/mplab-x-ide

直接リンクは、下記であるが、バージョン UP 等により変更になる可能性が高いので上記 URL を推奨する。

https://www.microchip.com/mplabx-ide-windows-installer

MPLAB XC16

MPLAB X インストールの最後にインストールを促すメッセージが出るが、web サイトを立ち上げるだけなので、あらかじめダウンロードしておくと良い。

下記 URL の画面下方のダウンロードからダウンロードする。

https://www.microchip.com/mplab/compilers

直接リンクは、下記であるが、バージョン UP 等により変更になる可能性が高いので上記 URL を推奨する。

https://www.microchip.com/mplabxc16windows

2.インストール

MPLAB X

1にてダウンロードした「MPLABX-v5.05-windows-installer.exe」を実行する事により、統合開発環境がインストールされる。

MPLAB X インストールの最後に XC のインストールを促すメッセージが出るが、ダウンロードして有る場合には、そのまま終了するとよい。

XC16

1にてダウンロードした「xc16-v1.35-full-install-windows-installer.exe」を実行する事により、統合開発環境がインストールされる。

MPLAB Code Configurator

MPLAB Code Configurator（以下 MCC と略す）は、MPLAB X の Plug-in なので、インストールは、MPLAB X を立ち上げ行う。

・1.「Tools」->「Plugins」を選択する。

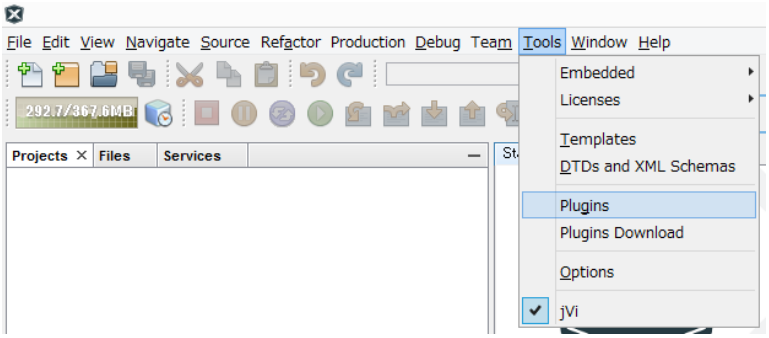

図 3-2.1.1 Plugins 設定の立ち上げ

・2.「Available Plugins」タブから「MPLAB Code Configurator」を探し、「Install」チェックボックスにチェックを入れ、「Install」ボタンを押す。

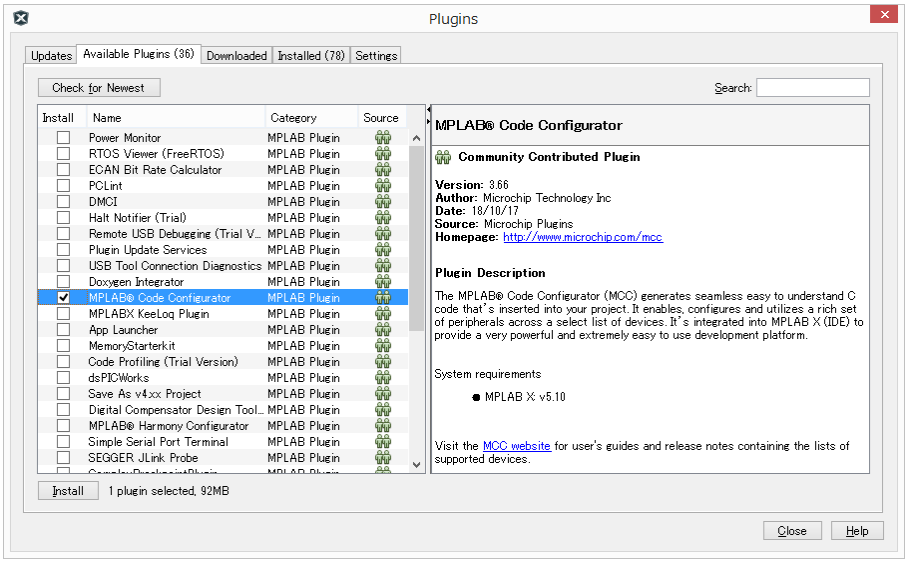

図 3-2.2.1 Plugins 選択及びインストール方法

・3.注意事項等が表示されるので、確認しインストールを行う。

・4. MPLAB X をいったん終了し、再度起動するとインストール完了する。

4.mruby/c インポート

今回は、最新版の mruby/c バージョン 1.2 を PIC24FJ256GB210 へインポートする。

ダウンロードは、下記より行う。

https://github.com/mrubyc/mrubyc/

また、mruby コンパイラは、バージョン 1.4 以降のが必須なため、合わせて下記よりダウンロードする。

http://www.s-itoc.jp/activity/research/mrubyc/mrubyc_download/

なお、mruby/c の動作を確認する方法としては、itoc サイトにて公開されているチュートリアル1～3 が動く事を確認することとする。

http://www.s-itoc.jp/activity/research/mrubyc/mrubyc_tutorial/

4.1 PIC24F 環境への mruby/c インポート

4.1 itoc チュートリアル「Chapter01「LED 点滅」」を PIC24F の環境で動かす

itoc チュートリアル「Chapter01「LED 点滅」」

http://www.s-itoc.jp/activity/research/mrubyc/mrubyc_tutorial/735

1.MPLAB X で新規プロジェクトを作成する。

「File」 -> 「New Project」を選択し、新規プロジェクトを作成する。

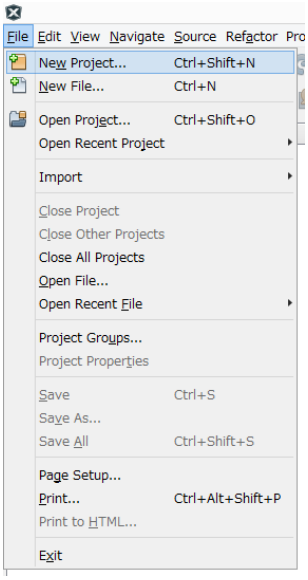

図 4.1.1-1 新規プロジェクト作成

2.「New Project」設定画面が出るので、
「Microchip Embedded」の
「Standalone Project」を選択し、
「Next」ボタンを押す。

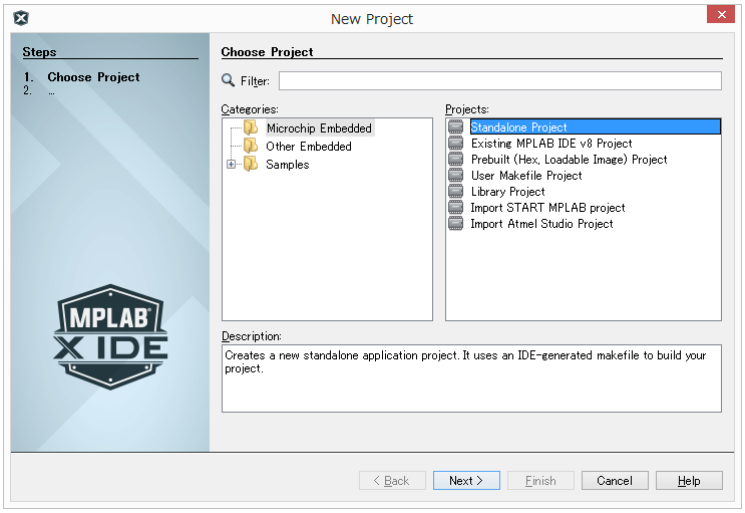

図 4.1.2-1 プロジェクト作成

3.「New Project」設定画面が出るので、
Family 欄に「16-bit MCUs（PIC24）」、
Device 欄に「PIC24FJ256GB210」
を表示させ「Next」ボタンを押す。

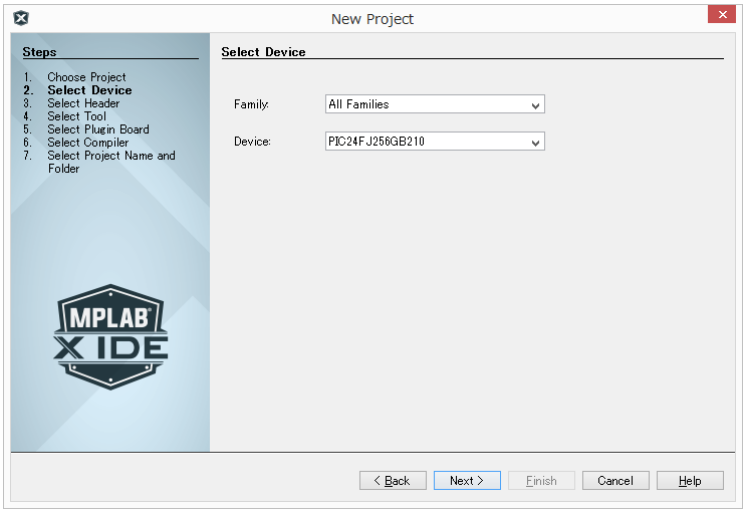

図 4.1.3-1 プロジェクト作成

4. 書き込みツールの設定画面になるので、
「Microchip Starter Kits」
を選択し「Next」ボタンを押す。

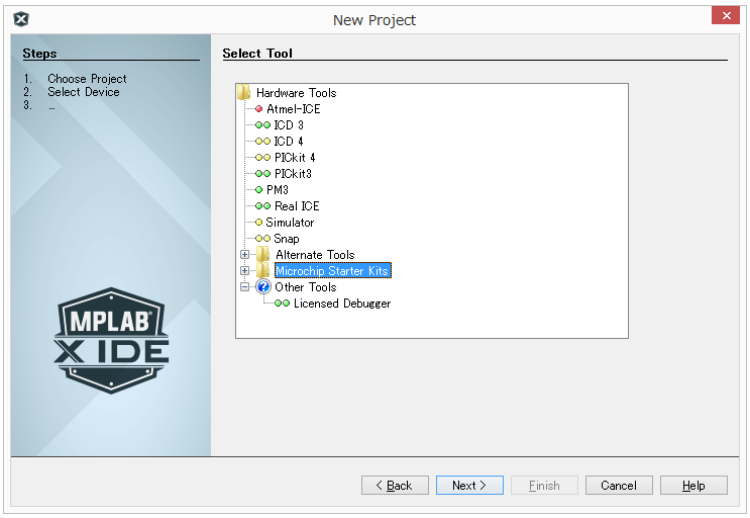

図 4.1.4-1 書き込みツール選択

5. コンパイラの設定画面になるので、
「XC16（V1.35）」
を選択し「Next」ボタンを押す。

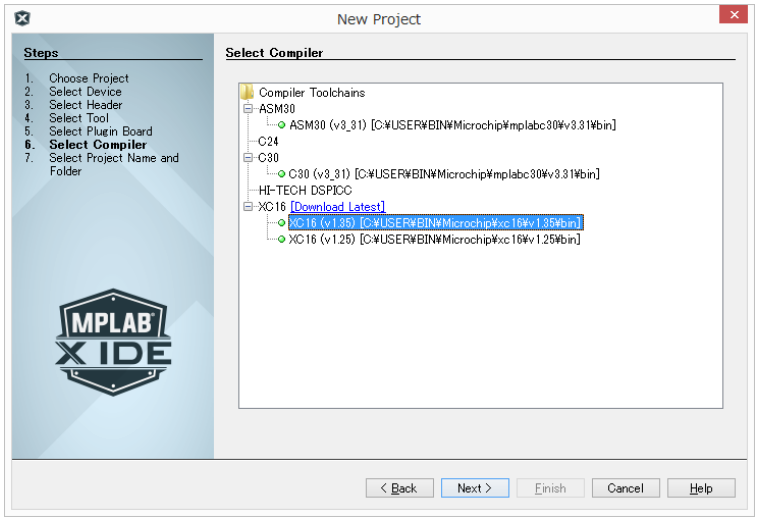

図 4.1.5-1 コンパイラ選択

6.プロジェクト名の設定画面になるので、
任意のプロジェクト名（ここでは、「mrubyc_led1」）
を入力し「Finish」ボタンを押す。

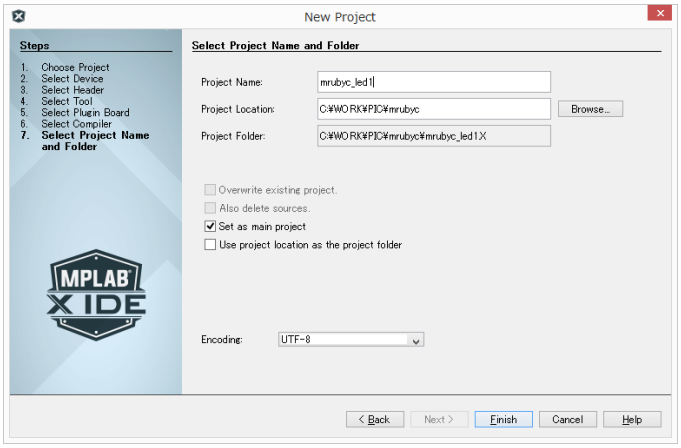

図 4.1.6-1 プロジェクト名入力

7.MCC を使いピンアサインを決定する

MCC でソースコードの自動生成を行うので、マージ作業を避けるため、ソースコードの無い状態で MCC を使いピンアサインなどを決定しておくと効率的である。

ここでは、EXPLORER 16/32 の
LEDsと Buttons のアサインを決定する。

まず、「MCC」ボタンを押し、MCC を
起動する

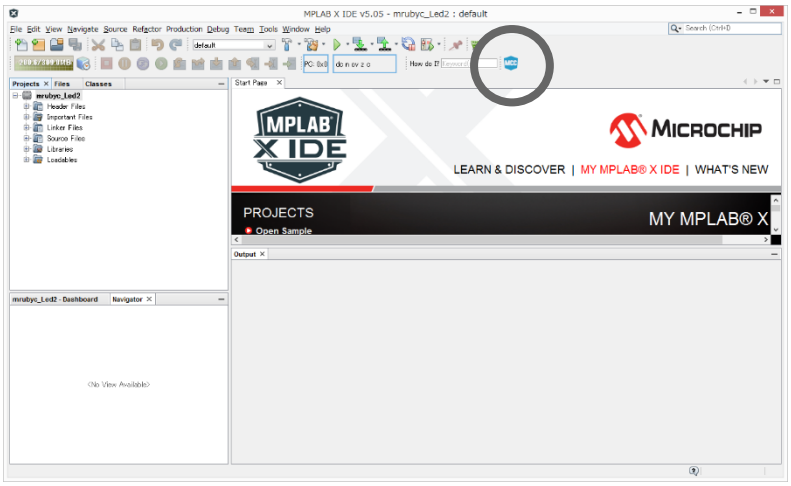

図 4.1.7-1 プロジェクト画面

MCC のコンフィグレーションファイル名の
設定画面になるので、任意の名前を入力し
「保存」ボタンを押す。
問題なければ、初期値のままで良い。

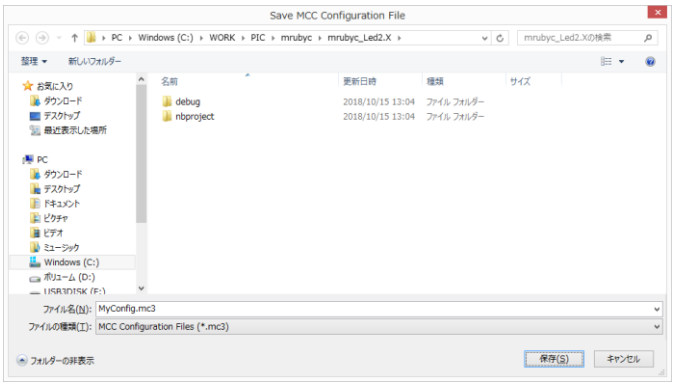

図 4.1.7-2 ファイル名入力

保存すると、MCC のメイン画面が開く。
Device Resources の一覧枠、設定項目選択の枠と、設定のメイン枠、ピンアサインの枠が表示される。

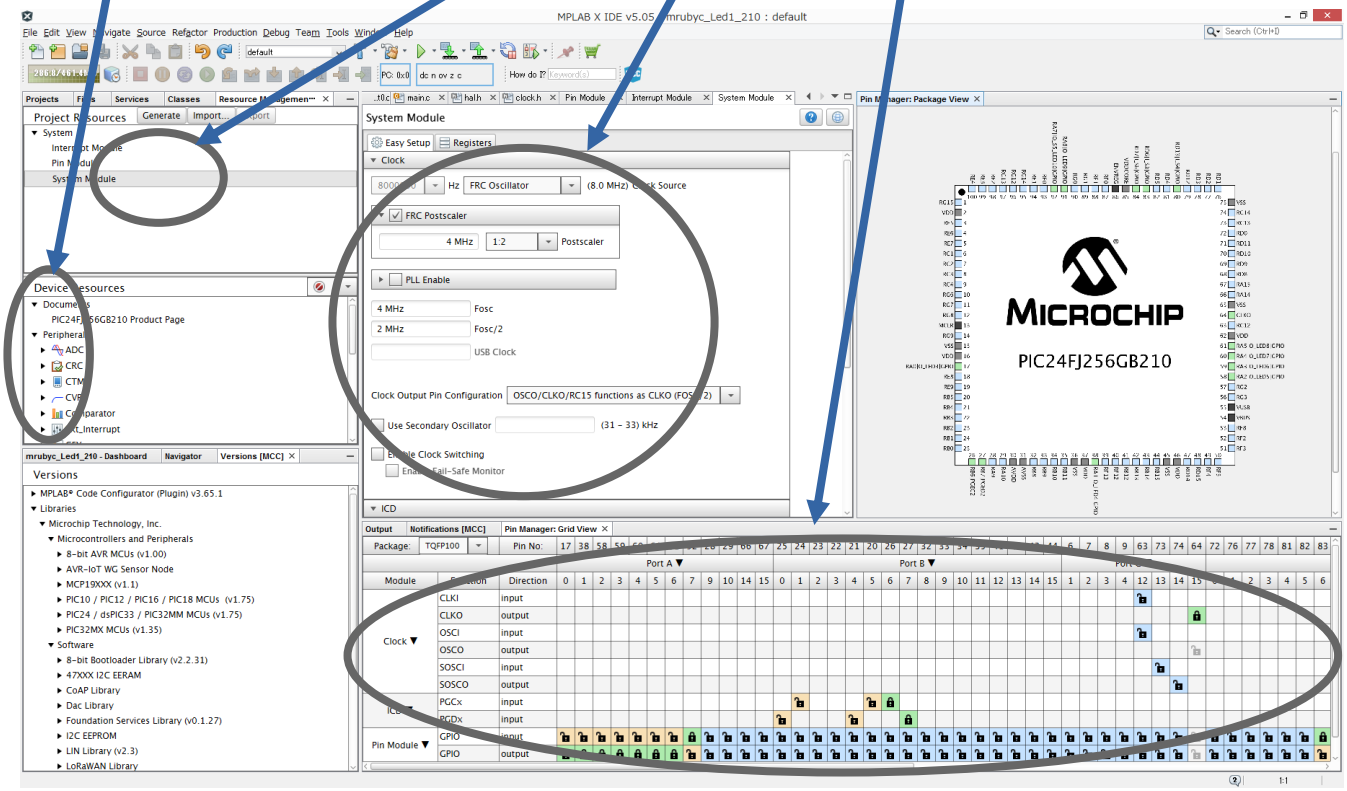

図 4.1.7-3 MCC メイン画面

右上のチップの絵については、ピンの状況の表示を行っている。

8.MCC のコンフィグレーションを開始する。

設定項目選択の枠で、「System Module」を選択する。

「System Module」は、クロック等の設定を行う画面となる。

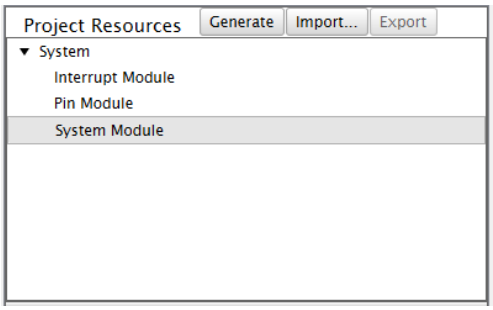

図 4.1.8-1 設定選択画面

System Module については、ICD(In Curcuit Debugger)の設定を変更する以外は初期値のままで問題ない。

EXPLORER 16/32 と MA240021 の組み合わせでは、ICD は、「PGEC2/PGED2」に接続されるので、そのように設定する。

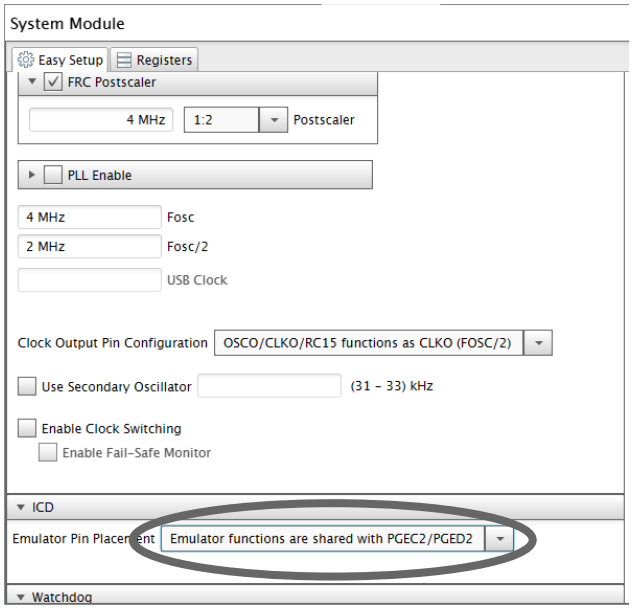

図 4.1.8-2 System Module 設定選択画面

9. 設定項目選択の枠で、「Pin Module」を選択する。

「Pin Module」の設定は、GPIO 等のピンアサイン設定を行う画面となる。

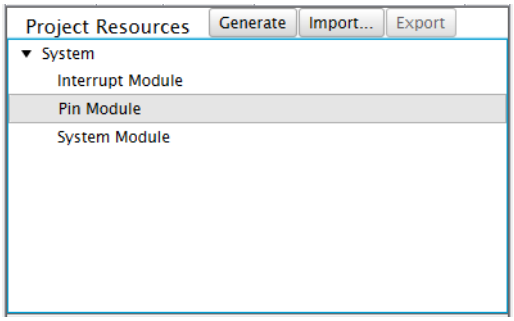

図 4.1.9-1 設定選択画面

ピンアサイン設定画面で機能と使用するピンを設定し、「Pin Module」設定画面で詳細を設定する。
プログラム中で分かり易いように「Custom Name」を設定しておくと良い。

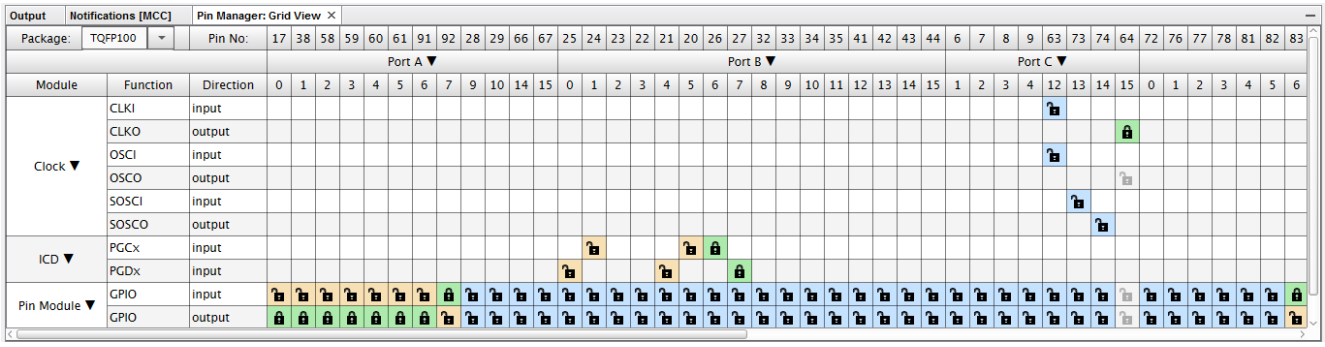

図 4.1.9-2 ピンアサイン設定画面

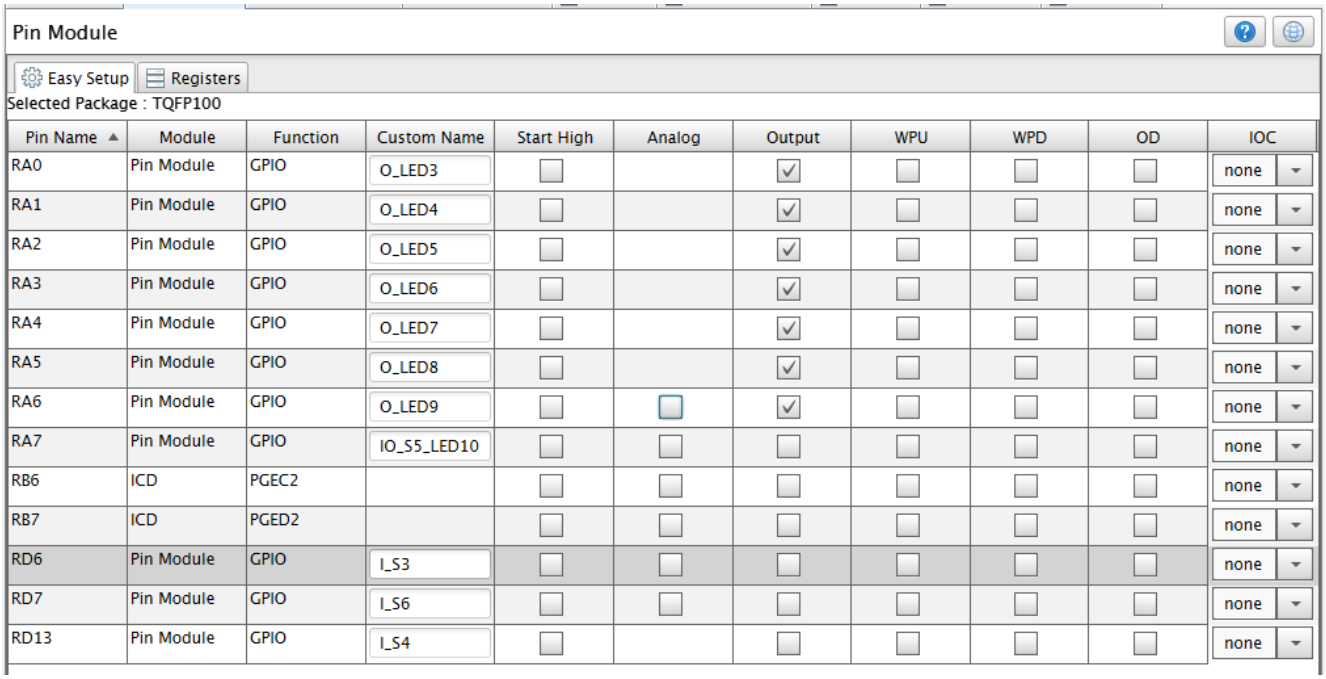

図 4.1.9-3Pin Module 設定画面

10.MCC の設定をソースに展開する。
「Generate」ボタンを押し、MCC の設定をソースに展開する。
MCC で変更を行った際には、この操作を行わないと、ソースに反映されないので注意が必要である。

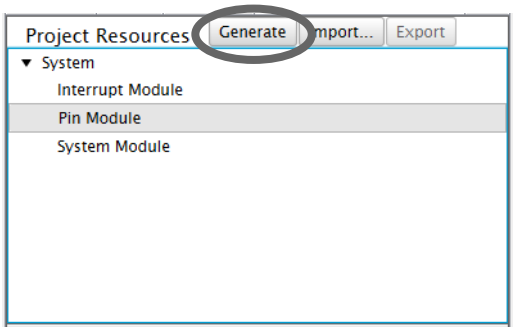

図 4.1.10-1 設定選択画面

11.mruby/c VM ソースコードを、プロジェクトへ追加

・mruby/c ソースコードを展開し、src フォルダー内の全ファイルを src フォルダーごとプロジェクトフォルダー（項目 6 で保存したフォルダ内）にコピーする。

・src フォルダー内の hal_psoc5lp フォルダーを hal にリネームする。

・src フォルダー内の hal_posix、hal_esp32 フォルダーを消去する。

・「mrubyc_Led1」の統合環境プロジェクト内にソースを登録する。

下記手順で、ソースを登録する。

プロジェクトツリーを表示し、「Header Files」の項目で、右クリックし、コンテキストメニューで、「Add Existing Items From Folders」を選ぶ。

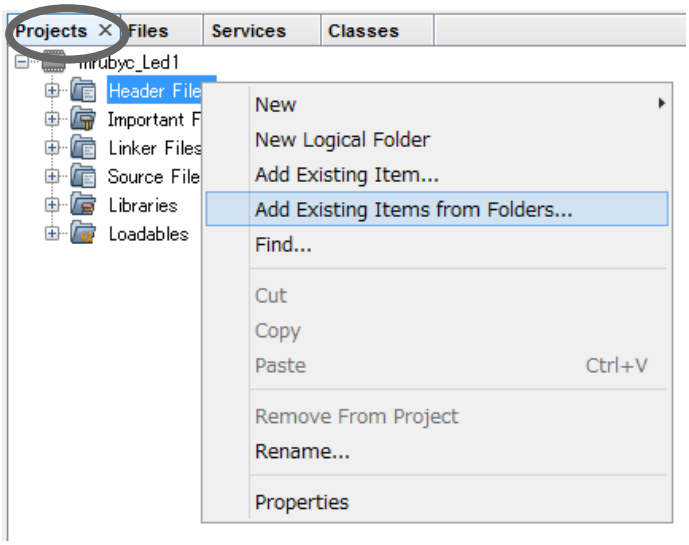

図 4.1.11-1 mruby/cヘッダファイル登録

下記の「Add Files」のダイアログが出るので、「Add Folder」ボタンを押す。

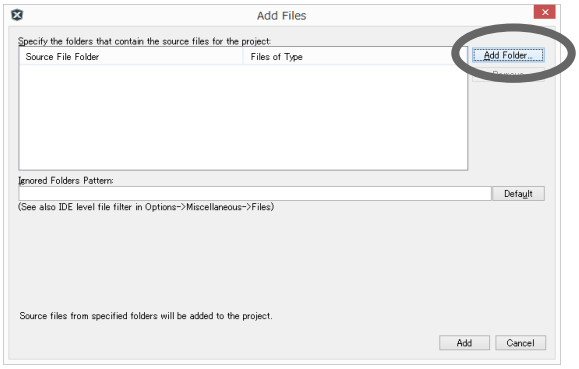

図 4.1.11-2 mruby/cヘッダファイル登録

下記の「Select Source File Folder」のダイアログが出るので、ファイルのタイプを「Header Files」にした後に mruby/c の src フォルダーを選択し、「Select」ボタンを押す。

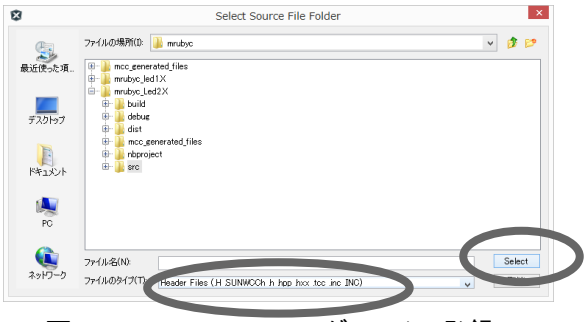

図 4.1.11-3 mruby/cヘッダファイル登録

下記の「Add Files」のダイアログに戻るので、「Add」ボタンを押しヘッダファイルを登録する。

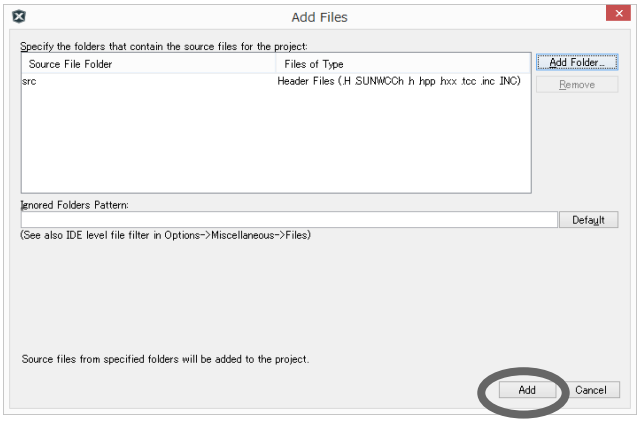

図 4.1.11-4 mruby/cヘッダファイル登録

次に、ソースファイルを登録する。ヘッダファイルの時と同様にプロジェクトツリーを表示し、「Source Files」の項目で、右クリックし、コンテキストメニューで、「Add Existing Items From Folders」を選ぶ。

「Add Files」のダイアログが出るので、「Add Folder」ボタンを押す。

下記の「Select Source File Folder」のダイアログが出るので、ファイルのタイプを「C Source Files」にした後に mruby/c の src フォルダーを選択し、「Select」ボタンを押す。

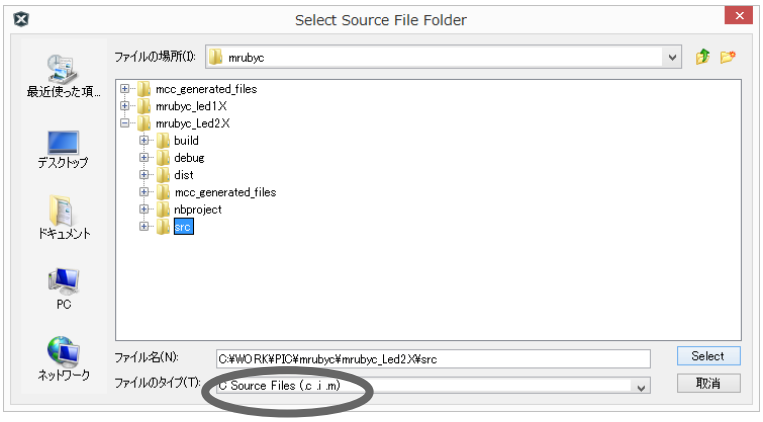

図 4.1.11-5 mruby/cソースファイル登録

下記の「Add Files」のダイアログに戻るので、「Add」ボタンを押しヘッダファイルを登録する。

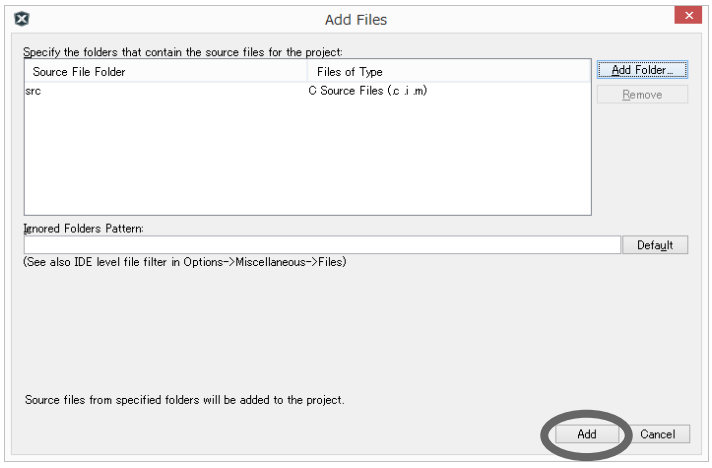

図 4.1.11-6 mruby/cソースファイル登録

12.プロジェクトプロパティの設定

プロジェクトプロパティの設定を設定する。設定内容一覧は下記である。

表 4.1.12-1 プロジェクトプロパティの設定

設定場所 1	設定場所2	設定項目	内容
XC16	Most Useful Options	C include dirs	src,src¥hal
XC16	Most Useful Options	Define common macros	MRBC_NO_TIMER
xc16-ld	General	Heap size	1024
xc16-ld	General	Min stack size	1024

プロジェクトツリーのルート、プロジェクト名の所で右クリックし、プロジェクトのプロパティを選択する。

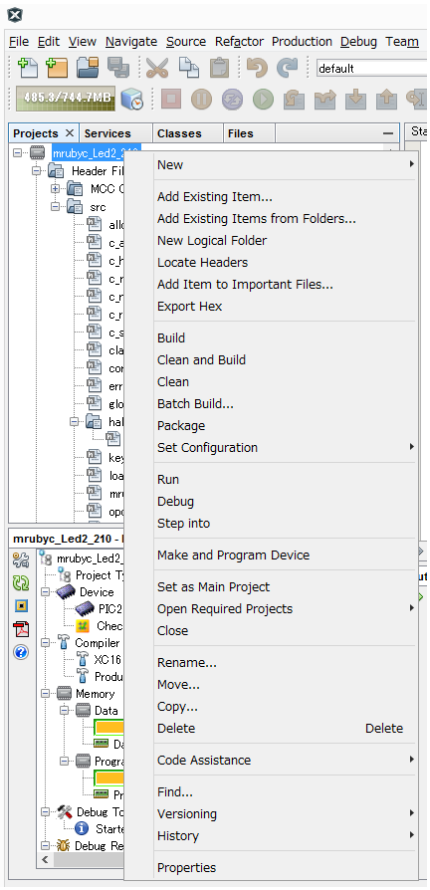

図 4.1.12-1 プロジェクトプロパティの設定

「Project Propeties」ダイアログが開くので、左ペインで「設定場所1」を選択（ここでは「XC16」）し、右ペインの「Option categories」で「設定場所2」を選択（ここでは、「Most Useful Options」）を選び、「設定項目」の右側の欄をクリックすると数字入力や、パラメータ入力が出来る。

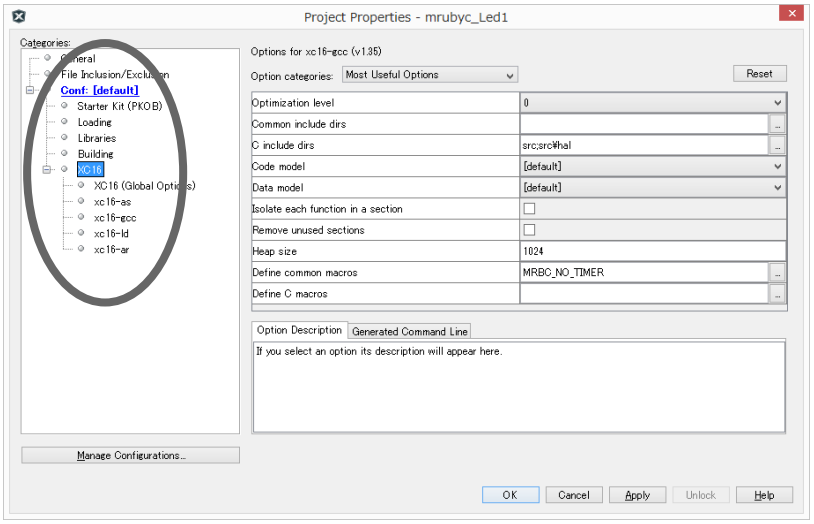

図 4.1.12-2 プロジェクトプロパティの設定

複数の項目を入力する場所では、下記のダイアログが開くので、それぞれ入力していく。

（下記は、「C include dirs」をクリックした際のダイアログなので、それぞれ「src」ディレクトリと「hal」ディレクトリを登録する）

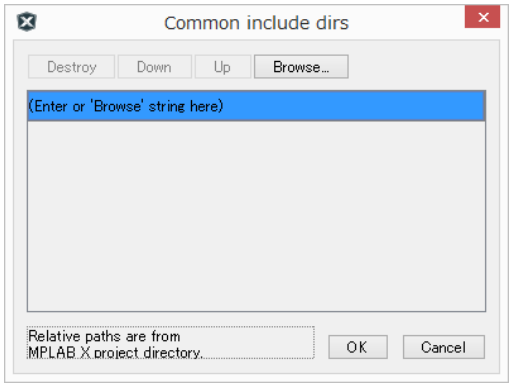

図 4.1.12-3 プロジェクトプロパティの設定

「Define common macros」の所では、下記のダイアログが開くので、それぞれ入力していく。

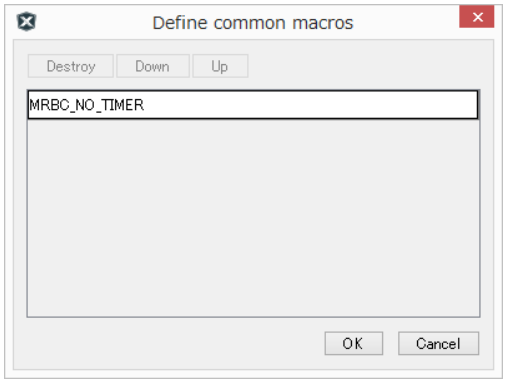

図 4.1.12-4 プロジェクトプロパティの設定

13.mruby/c プログラムの作成及びコンパイル

mruby/c プログラムの作成及びコンパイルを行う。

以下のプログラムを「sample1.rb」として保存する。

```
led = 0

while true
  if led == 1
    led = 0
  else
    led = 1
  end

  if sw1_read() == 0
    led = 1
  end

  led1_write( led )
  sleep 1
end
```

このプログラムを下記の通りコンパイルする。出来た「sample1.c」を、「mrubyc_Led1」のプロジェクトフォルダーに配置する。

```
mrbc.exe -E -Bsample1 sample1.rb
```

次に、ソースファイルを登録する。ヘッダファイルの時と同様にプロジェクトツリーを表示し、「Source Files」の項目で、右クリックし、コンテキストメニューで、「Add Existing Item」を選ぶ。

「Select Item」のダイアログが出るので、「sample1.c」を選び、「Select」ボタンを押し登録する。

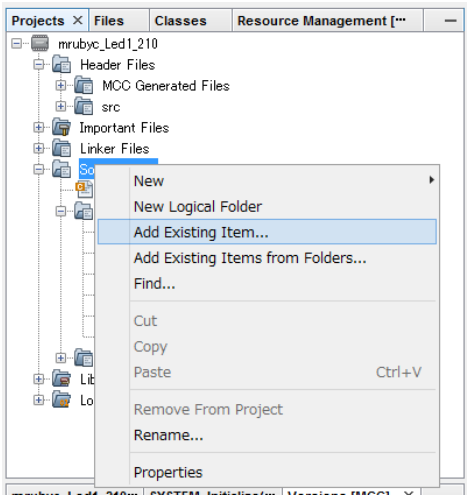

図 4.1.13-1 ソースファイルの追加

14.「main.c」プログラムの変更

MCC によって「main.c」が自動生成されているので、ボード初期化ルーチンと mruby/c 起動ルーチンを組み込む。

・main.c

```
#include "mcc_generated_files/system.h"
#include "mcc_generated_files/clock.h"
#include "mcc_generated_files/pin_manager.h"

#define  FCY    _XTAL_FREQ/2

#include "mrubyc.h"

// extern char sample1[];
extern const uint8_t sample1[];

#define MEMORY_SIZE (1024*20)
static uint8_t memory_pool[MEMORY_SIZE];

//================================================================

/*! オンボード SW 現在状態の読み込み
 */
static void c_sw1_read(mrb_vm *vm, mrb_value *v, int argc) {
    int sw1 = I_S4_GetValue();
    SET_INT_RETURN(sw1);
}

//================================================================

/*! オンボード LED ON/OFF
 */
static void c_led1_write(mrb_vm *vm, mrb_value *v, int argc) {
    int set_value = GET_INT_ARG(1);
    if (set_value == 0)
        O_LED3_SetLow();
    else
        O_LED3_SetHigh();
}


//================================================================

/*! HAL
*/
int hal_write(int fd, const void *buf, int nbytes) {
    return 0;
}

int hal_flush(int fd) {
    return 0;
}
```

```
/*
                         Main application
 */
int main(void)
{
    // initialize the device
    SYSTEM_Initialize();

    mrbc_init(memory_pool, MEMORY_SIZE);

    mrbc_define_method(0, mrbc_class_object, "sw1_read", c_sw1_read);
    mrbc_define_method(0, mrbc_class_object, "led1_write", c_led1_write);

    mrbc_create_task(sample1, 0);
    mrbc_run();

    return 1;
}
```

15.「hal.h」の修正

ハードウエアレイヤーのルーチンである「hal.h」を修正する。

```
#ifdef __cplusplus
extern "C" {
#endif

#include "../mcc_generated_files/clock.h"

#ifndef FCY
#define  FCY    _XTAL_FREQ/2
#endif

/***** Feature test switches ************************************************/
/***** System headers *******************************************************/
#include <xc.h>
#include <libpic30.h>

/***** Local headers ********************************************************/
/***** Constant values ******************************************************/
/***** Macros ***************************************************************/

#ifndef MRBC_NO_TIMER
# define hal_init()        ((void)0)
# define hal_enable_irq()  ((void)0)
# define hal_disable_irq() ((void)0)
# define hal_idle_cpu()    ((void)0)

#else // MRBC_NO_TIMER
# define hal_init()        ((void)0)
# define hal_enable_irq()  ((void)0)
#define hal_disable_irq() ((void)0)
#define hal_idle_cpu()    ((__delay_ms(1)), mrbc_tick())

#endif
```

16.「vm_comfig.h」の変更

PIC は、little endian の16bit alignment の CPU なので、対応出来る様に「vm_comfig.h」を変更する。

```
/* Hardware dependent flags */

/* Endian
   Define either MRBC_BIG_ENDIAN or MRBC_LITTLE_ENDIAN.
*/
#if !defined(MRBC_BIG_ENDIAN) && !defined(MRBC_LITTLE_ENDIAN)
# define MRBC_LITTLE_ENDIAN
#endif

/* 32bit alignment
   If 32-bit alignment is required, enable the following line.
 */
#define MRBC_REQUIRE_32BIT_ALIGNMENT
```

17.「alloc.c」の変更

「alloc.c」の「mrbc_raw_alloc()」関数で、メモリ確保の下限値丸め込みルーチンを有効化する。

これは、mruby/c で利用しているメモリアロケータの最小メモリアロケーションサイズが16byteであり、32bit 以上のマシンでは、下限値を下回る事が無い為、無駄なコードとなるのでコメントアウトされている。

```
  // check minimum alloc size.  if need.
#if 1   // <- 元は「0」無効であった
  if( alloc_size < (1 << MRBC_ALLOC_IGNORE_LSBS) ) {
    alloc_size = (1 << MRBC_ALLOC_IGNORE_LSBS);
  }
#else
  assert( alloc_size >= (1 << MRBC_ALLOC_IGNORE_LSBS) );
#endif
```

18.クリーンビルド

「Production」->「Crean and Build Project」を行う。

エラーが出ずにコンパイル出来る事を確認する。

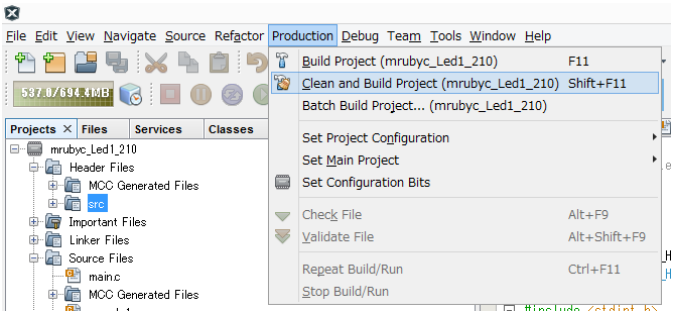

図 4.1.19-1 ビルドメニュー

19.本体への書き込み及び実行

EXPLORER16/32 を PC と付属の USB ケーブル（USB-A MicroB）で接続する。

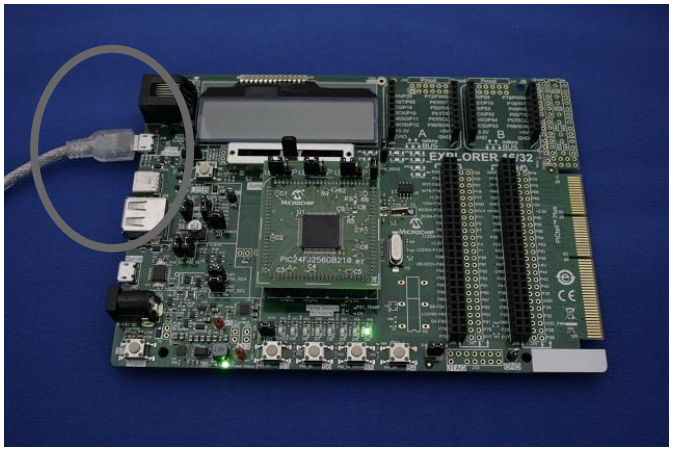

写真 4.1.20-1 EXPLORER16/32 と PC との接続

「Debug」->「Debug Project」を選択する。

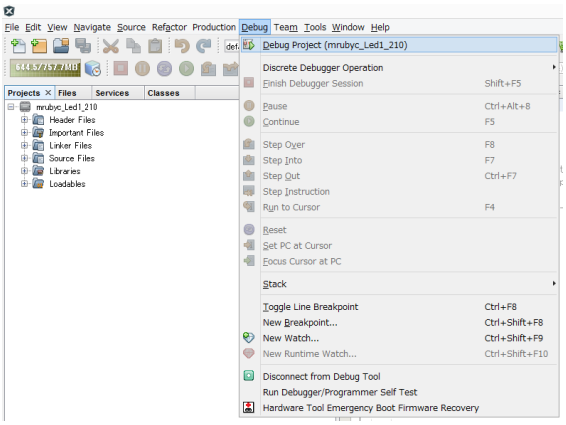

図 4.1.20-1 EXPLORER16/32 書き込み

書き込みが終わるとデバッガ画面となるので、「Continue」ボタンを押すと、ボード上でプログラムが実行され、LED D3 が緑色に 1 秒おきに点滅する。

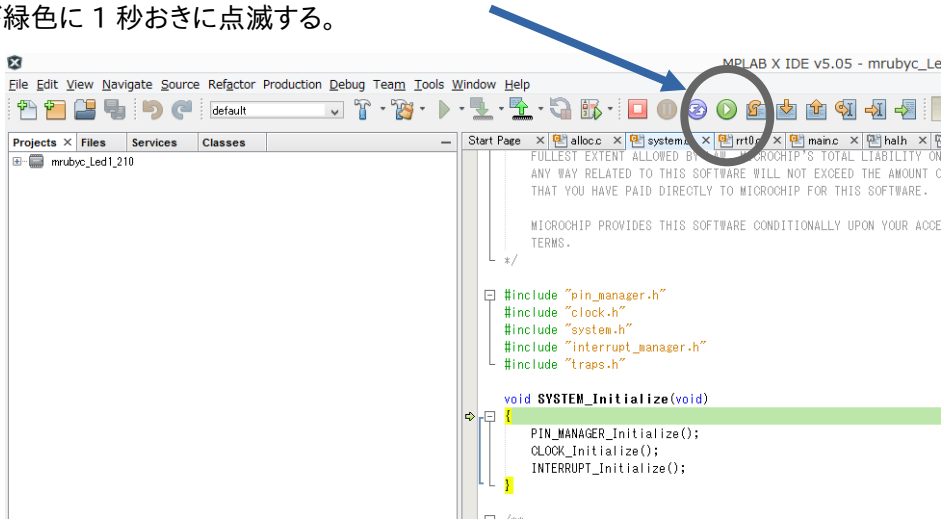

図 4.1.20-2 デバッガ画面

LED D3 が緑色に 1 秒おきに点滅する。

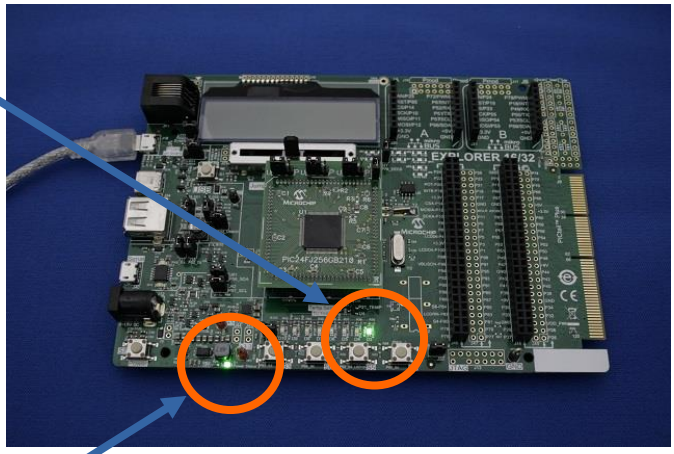

写真 4.1.20-2 LED 点滅

なお、左方の LED D1は、電源ランプであり緑色に点灯していると電源が入っている事を示す。

また、スイッチ S4 を押すと押している間 LED D3 が点灯したままになる事を確認する。

4.2 Chapter02「LED 点滅の速さを変える」を PIC24F の環境で動かす

4.1 で作成したプロジェクトをベースにする。

1.mruby スクリプトの自動コンパイル処理の追加

・コンパイル用バッチファイル mrbc.bat を mruby ソースコードをおいたディレクトリへコピーする。

バッチファイルは、ITOC 様のページの mruby コンパイラをダウンロードする。

https://www.s-itoc.jp/activity/research/mrubyc/mrubyc_download/

・mrbc.bat 中の set MRBC 行を mrbc.exe の絶対パスに書き替える。

・プロジェクトツリーのルート、プロジェクト名の所で右クリックし、プロジェクトのプロパティを選択する。

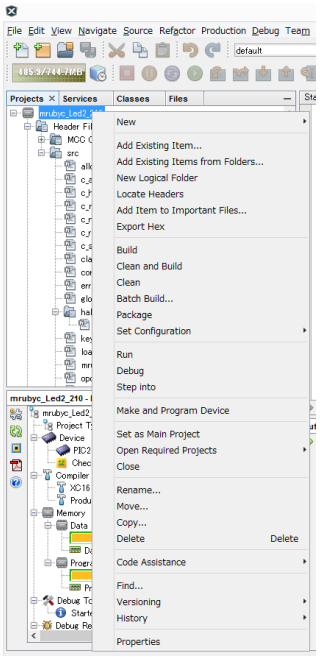

図 4.2.1-1 自動コンパイルスクリプト追加

・「Building」を選び、「Execute This line before build」にチェックをいれ、「mrbc.bat」を登録する。

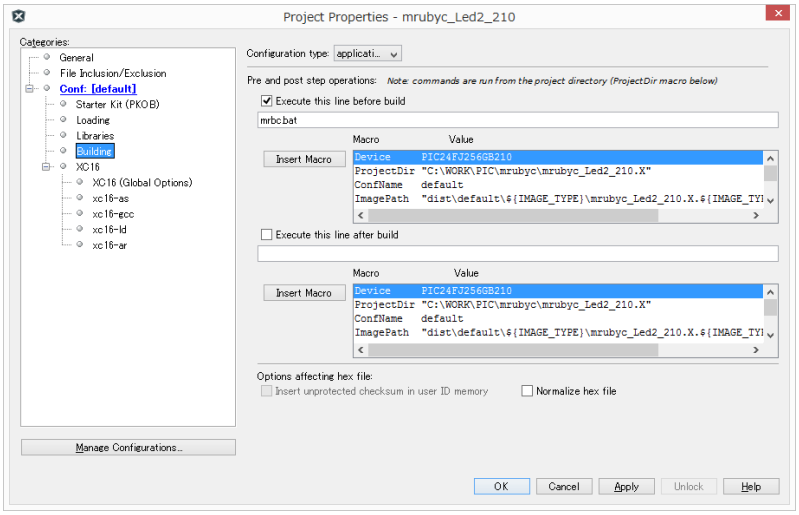

図 4.2.1-2 自動コンパイルスクリプト追加

2.浮動小数点の利用

1秒おきのループを、以下の通り 0.2 秒おきに点滅するように変更する

```
led = 0

while true
  if led == 1
    led = 0
  else
    led = 1
  end

  if sw1_read() == 0
    led = 1
  end

  led1_write( led )
  sleep 0.2
end
```

4.1 の 18 以下の手順通りにコンパイルし実行する。

すると、LED D3 が 0.2 秒間隔で点滅する。(写真 4.1.20-2 LED 点滅 参照)

また、スイッチ S4 を押すと押している間 LED D3 が点灯したままになる事を確認する。

3. タイマーの利用

次にハードウェアタイマーを利用するように設定を変更する。

・MCC にてハードウェアタイマーの TMR1 を設定する。

MCC を立ち上げ、Device Resources 欄から「Timer」->「TMR1」をダブルクリックし、「Project Resources」に登録する。

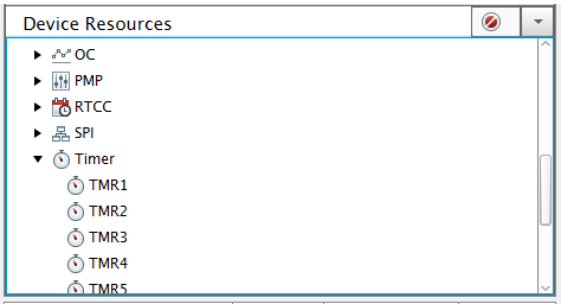

図 4.2.3-1 Device Resources の Timer 項目

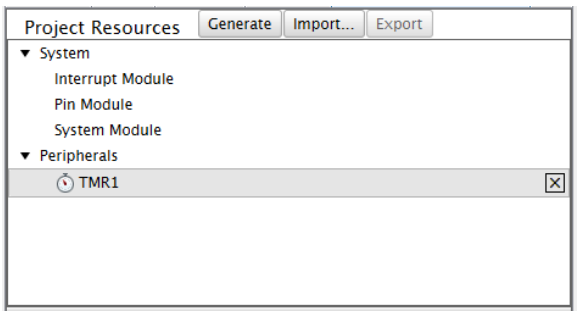

図 4.2.3-2 Project Resouces に TMR1 を登録

TMR1の設定を行う。

TMR1 を1ms毎に実行するので、「Tmer Period」欄に「1ms」を入力し、「Enable Timer interrupt」を有効化し、「Generate」する。

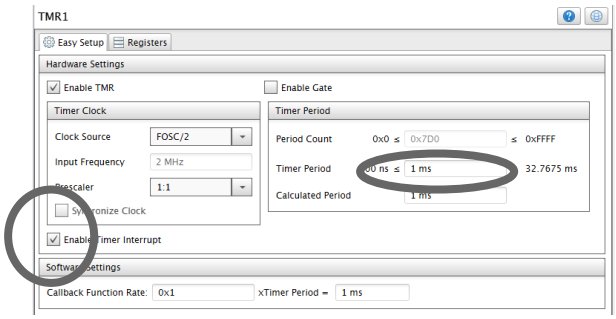

図 4.2.3-2 TMR1 の設定

4.「hal.h」の修正

ハードウエアレイヤーのルーチンである「hal.h」を修正する。

割り込み許可、禁止、cpu 休止命令を追加する。

```
#ifndef MRBC_NO_TIMER
#define hal_init()         ((void)0)
#define hal_enable_irq()   __builtin_disi(0x0000)
#define hal_disable_irq()  __builtin_disi(0x3FFF)
#define hal_idle_cpu()     Idle()
```

5.「main.c」に、タイマー割り込みハンドラとスタートを設定する。

```
/*
                         Main application
 */
int main(void)
{
    // initialize the device
    SYSTEM_Initialize();

    mrbc_init(memory_pool, MEMORY_SIZE);

    mrbc_define_method(0, mrbc_class_object, "sw1_read", c_sw1_read);
    mrbc_define_method(0, mrbc_class_object, "led1_write", c_led1_write);

    mrbc_create_task(sample1, 0);

    TMR1_Start();       // <- added

    mrbc_run();

    return 1;
}

/*
 * TIMER CallBack
 */
void TMR1_CallBack(void) {      // <- added
    mrbc_tick();                // <- added
}                               // <- added
```

6.プロジェクトプロパティの設定

プロジェクトプロパティの設定を設定する。設定内容一覧は下記である。

4.1 との違いは、ハードウェアタイマーを使う為、「MRBC_NO_TIMER」宣言をしないことである。

表 4.2.6-1 プロジェクトプロパティの設定

設定場所 1	設定場所2	設定項目	内容
XC16	Most Useful Options	C include dirs	src,src¥hal
xc16-ld	General	Heap size	1024
xc16-ld	General	Min stack size	1024

プロジェクトツリーのルート、プロジェクト名の所で右クリックし、プロジェクトのプロパティを選択する。

以前のプロジェクトを引き継いでいる場合は、設定が記述済みである。

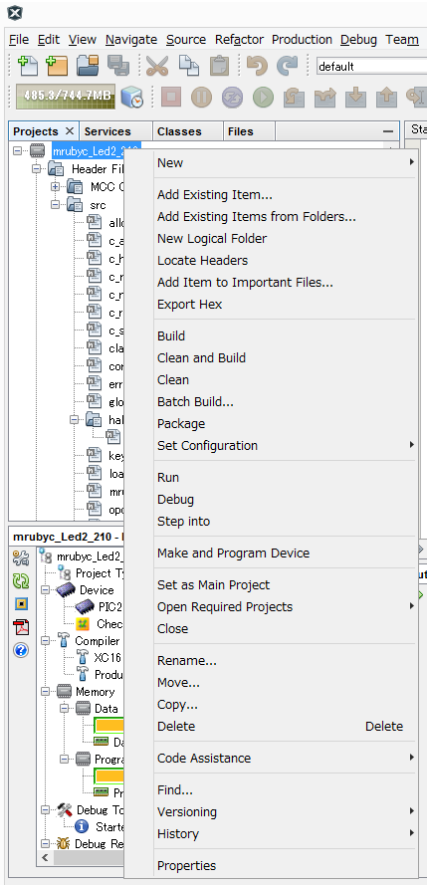

図 4.2.6-1 プロジェクトプロパティの設定

「Project Propeties」ダイアログが開くので、左ペインで「設定場所1」を選択（ここでは「XC16」）し、右ペインの「Option categories」で「設定場所2」を選択（ここでは、「Most Useful Options」）を選び、「設定項目」の右側の欄をクリックすると数字入力や、パラメータ入力が出来る。

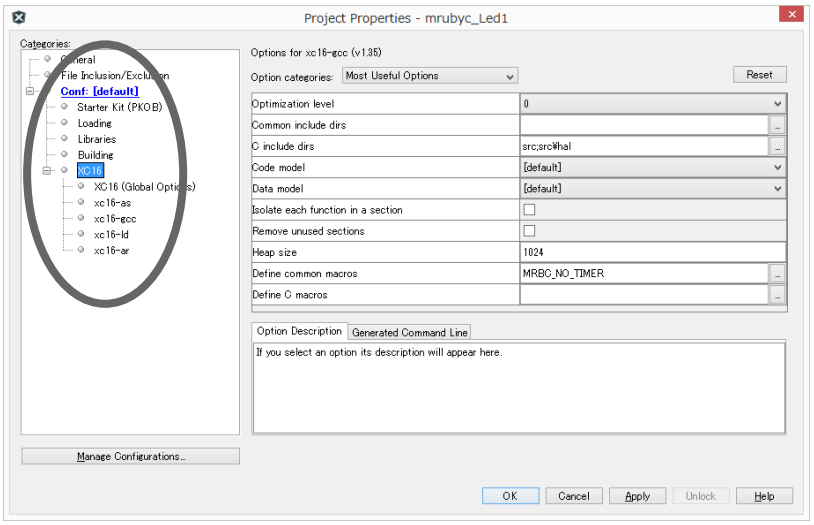

図 4.2.6-2 プロジェクトプロパティの設定

複数の項目を入力する場所では、下記のダイアログが開くので、それぞれ入力していく。

（下記は、「C include dirs」をクリックした際のダイアログなので、それぞれ「src」ディレクトリと「hal」ディレクトリを登録する）

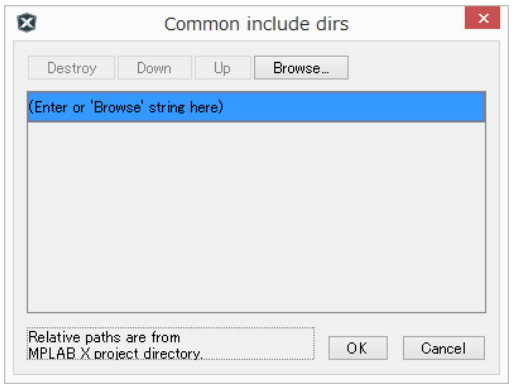

図 4.2.6-3 プロジェクトプロパティの設定

「Define common macros」の所では、以前のプロジェクトを引き継いでいる場合には、下記のように「MRBC_NO_TIMER」が入力済みであるので、削除する。

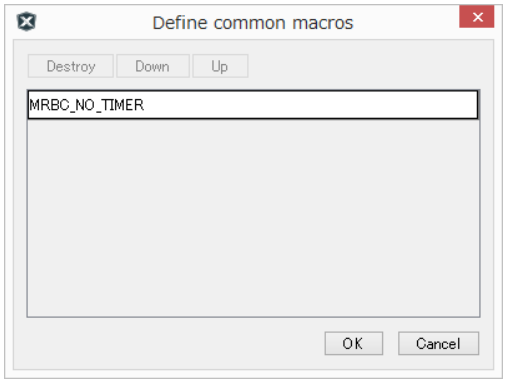

図 4.2.6-4 プロジェクトプロパティの設定

7.クリーンビルトと実行

4.1 の 18 以下の手順通りにコンパイルし実行する。

すると、LED D3 が 0.2 秒間隔で点滅する。(写真 4.1.20-2 LED 点滅 参照)

また、スイッチ S4 を押すと押している間 LED D3 が点灯したままになる事を確認する

4.3 Chapter03「複数の mruby プログラムを同時に動かす」を PIC24F の環境で動かす

複数の mruby プログラムを同時に動かします。

1.mruby プログラムの用意とコンパイル

下記の mruby プログラム「sample1.rb」「sample2.rb」を用意する。

コンパイルは、4.2 で準備した自動コンパイル処理にて行われる。

「sample1.rb」

```
led = 0

while true
  if $sw1 == 1
    led = 0
  else
    led = 1
  end

  if sw1_read() == 0
    led = 1
  end

  led1_write( led )
  sleep 0.2
end
```

「sample2.rb」

```
while true
  $sw1 == sw1_read()
end
```

2.ビルド及び実行を行う。

4.1 の 18 以下の手順通りにコンパイルし実行する。

すると、LED D3 が 0.2 秒間隔で点滅する。(写真 4.1.20-2 LED 点滅 参照)

また、スイッチ S4 を押すと押している間 LED D3 が点灯したままになる事を確認する。

3.コンソールを有効にする

UART を使ったコンソールを有効化する。

・MCC にてハードウェア UART の UART1 を設定する。

MCC を立ち上げ、Device Resources 欄から「UART」->「UART1[Foundation Services Library by Microchip Technology,Inc]」をダブルクリックし、「Project Resources」に登録する。

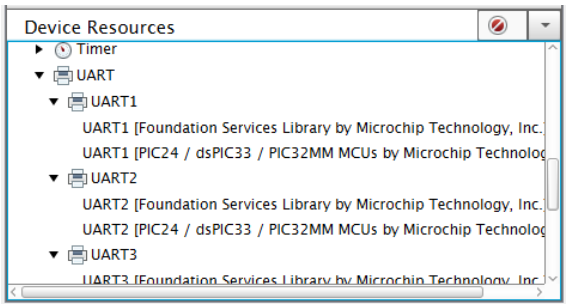

図 4.3.3-1 Device Resources の UART 項目

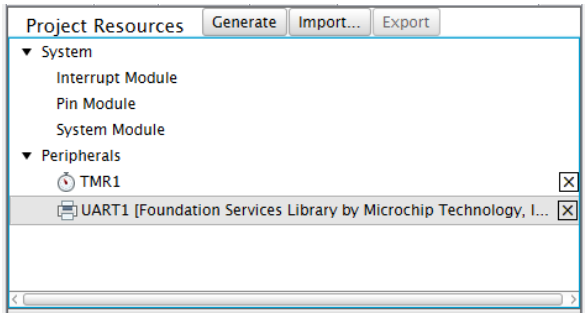

図 4.3.3-2 Project Resouces に UART1 を登録

UART1の設定を行う。

UART1 を Hardware Settings は初期値で、ピンアサインは、「U1RX:49」「U1TX:50」に設定し、「Generate」する。

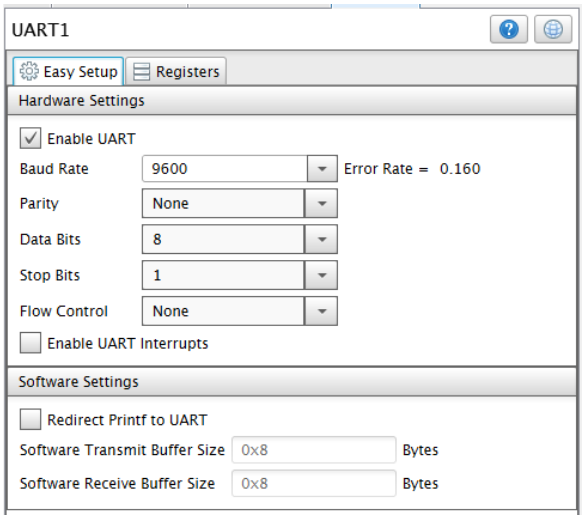

図 4.3.3-3 UART1 設定

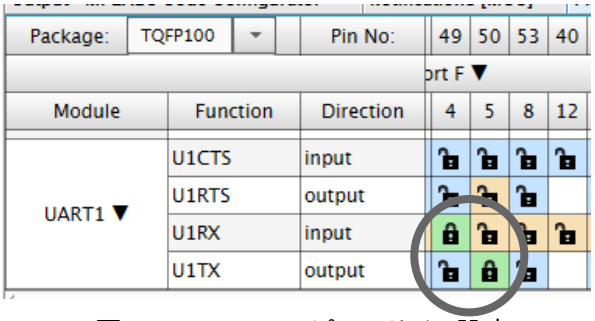

図 4.3.3-4 UART1 ピンアサイン設定

4.「main.c]プログラムの変更

下記の通り、hal_write()を実装する。

```
//============================================================

/*! HAL （別途説明します）
 */
int hal_write(int fd, const void *buf, int nbytes) {
    int i;
    while (U1STAbits.TRMT == 0);
    for (i = nbytes; i; --i) {
        while (U1STAbits.TRMT == 0);
        U1TXREG = *(char*) buf++;
    }
    return (nbytes);
}
```

5.mruby/c プログラムの修正

「sample1.rb」を下記の通り修正する

```
led = 0

while true
  if led == 1
    led = 0
  else
    led = 1
  end

  if sw1_read() == 0
    led = 1
  end

  led1_write( led )
  puts led     # 追加
  sleep 0.2
end
```

6. ビルド及び実行を行う。

4.1 の 19 の手順通りにコンパイルし実行する。

すると、LED D3 が 0.2 秒間隔で点滅する。(写真 4.1.20-2 LED 点滅 参照)

さらにシリアルコンソールに LED D3 点灯時1消灯時 0 が出力される。

なお、文字区切りコードは、LF なので、注意が必要である。

また、確認の際には、EXPLORER16/32 に搭載されているシリアル・USB 変換の ICMicrochip 社製の MCP2221A を利用する。

ドライバーは、下記からダウンロードできる。

https://www.microchip.com/wwwproducts/en/MCP2221A

から、下方の Tab の「Doccuments」内「Software」より、適宜ダウンロードする。

windows ドライバー直接ダウンロード(URL が変わる可能性が有ります)

http://ww1.microchip.com/downloads/en/DeviceDoc/MCP2221%20Windows%20Driver%202014-10-09.zip

図 4.3.6-1 ターミナル画面

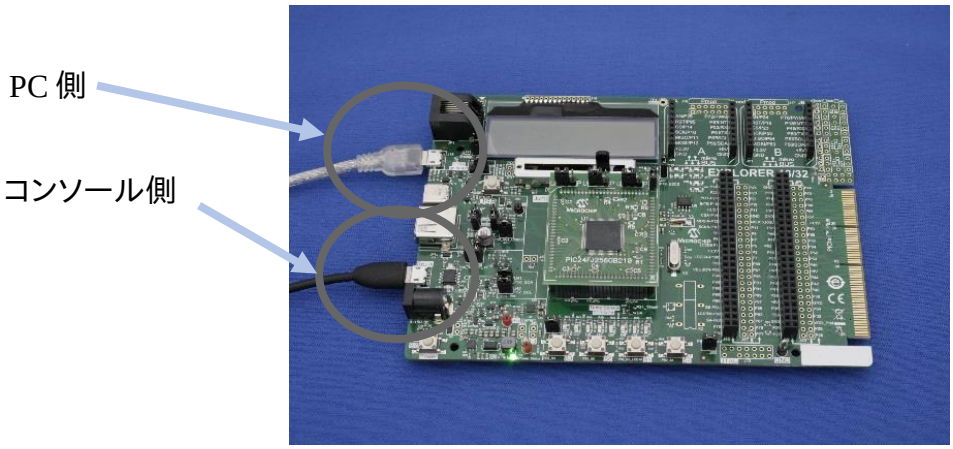

写真 4.3.6-1 EXPLORER16/32 と PC との接続

また、スイッチ S4 を押すと押している間 LED D3 が点灯したままになる事を確認する。

その際には、シリアルコンソールには、LED D3 点灯中なので1が出力される。

5.、メモリ消費量について

mruby/c は、メモリ必要量が少ない事が特徴の 1 つであるが、CPU のビット数によって変化するかを検証した。

本レポートで使用している PIC 16bit 環境、比較対象として、intel 64bit 環境、OPENRISC 32bit 環境を比較する事とし、メモリ消費量としては、mrbc_init()で確保した memory_pool の RAM 使用量で比較する。

ruby のコードは、mruby/c の「sample1.rb」を利用した。

表 5-1 CPU ビット数と RAM 消費量

CPU ビット数	Intel x64 64bit OS FreeBSD	OPENRISC 32bit OS なし	PIC24F 16bit OS なし
RAM 消費量	12,952	8,416	5,304
16bit の消費量を 1 とした比	2.4	1.6	1

計測の結果、CPU の bit 数が小さくなるほど消費するメモリ量が削減される事が分かった。

理由としては、ポインタのサイズが、それぞれ CPU のビット数で異なる事が原因と思われる。

また、この事は、同じ RAM の量でも16bitCPUの方が 32bitCPU より 1.6 倍 RAM が使える事を示しており、より大きなプログラムを動かせる可能性を示すといえる。

また、PIC24Fシリーズ（dsPIC33 含む）には、427 品種あるが、RAM が 8kByte を超えるのは324品種あるので、PIC24F の半数以上の品種で mruby/c が動作可能といえそうである。

ただし、8Kbyre の品種では、利用可能な RAM は 2kByte を切るため、どの程度のコードが動作可能かは十分検証する必要が有る。

実用的には16kByte以上のメモリが必要となると思われるので、199 品種に減る事となる。

また、PIC24Fのデータメモリエリアは 32kByte 毎のバンク切り替えとなっている為、32kByte 以上のエリアをアクセスする配列を作るには、「__eds__」型修飾子を付ける必要が有るので注意する必要が有る。

なお、現状の mruby/c コードでは、vm 立ち上げ時に一時的にではあるが、スタックに多量のメモリを確保するのでその点を改良するとさらに使用できるメモリを増やす事が出来ると思われる。

6.まとめ

Microchip 社製の PIC24F の16bitの環境にて mruby/c のチュートリアルの動作が確認出来た。

CPUの bit 数が少ないほどメモリ消費量が少なくて済む傾向がわかった。

mruby/cのPIC24へのポーティング調査報告書作成

調査報告書作成　株式会社バイタルリード　片岡 晃
調査報告書監修　しまねソフト研究開発センター　東 裕人