IOCSとは、何なのか

過去に、私のダウンロードサイトから、HEWのProjectを ダウンロードされた方であれば、Projectのソースファイルに "IOCS"という文字列を含んだソースファイル名を、数本見た事があると思います。 IOCSは、InputOutputControlSystemの略です。 要は、各種 組み込み用マイコンの、入出力周りの アクセスを行うサブルーチンを 集めて、数本のファイルにした物です。 で、R8Cマイコン用は、私が作成しました。 過去に 似たような物を H8/300Hマイコン、SH2マイコン等で作った事があります。

まだ、この程度の物で、ライブラリと呼ぶにはおこがましいのですが、ライブラリと呼ばれる物は、通常 リンク時に連結編集される LIBファイルになっています。

パソコンの STDIOの ライブラリとかは、もう既に仕様が固まっているので、ライブラリファイルの内容を 変更、改修する事は まずないと思います。 その関係で コンパイルした オブジェクトファイルというか、リロケータブルファイルを、1本のオブジェクトファイルに まとめた物を ライブラリファイルと呼びます。 ライブラリのソースファイルは、別途管理されている場合が、多いです。 商用の開発環境であれば、ライブラリファイルのソースファイルは、公開されて無い場合が多いです。 Linuxの環境は、基本オープンソースなので、gccの ライブラリファイルの ソースをダウンロードする事も可能と思います。

で、組み込みマイコンの場合ですが、特に I/O回りは、CPUによって異なるので、なかなか標準化は、難しいと思います。よって、I/O周りのサブルーチンは、CPUのアーキテクチャを生かす形で都度作成する事になると思います。 それらのサブルーチン集を、IOCSと呼ぶ事にしています。

IOCSを構成するファイル群です。

```
cpu_select.h            // CPU選択の C言語ヘッダファイルです
cpu_select.inc          // CPU選択の アセンブラ インクルードファイルです。
IOCS_Readme.txt         // 最初に見て下さいの説明ファイルです。
IOCS_sio_print.c        // 数値 文字列変換を含む 文字列のシリアル送信 関数群です。
R8CM12_IOCS.h           // C言語で使用するための IOCSの 全関数プロトタイプ宣言です。
R8CM1_IOCS_ADC.a30      // A/Dコンバーター処理関数群です。
R8CM1_IOCS_INIT.a30     // 高速クロック切り替えを含む 基本関数群です。
R8CM1_IOCS_TIMER.a30    // 1msを ベースとするインターバルタイマ処理関数群です。
R8CM1_IOCS_UART.a30     // 調歩同期式シリアル通信 基本関数群です。
sample_main.c           // メイン関数、初期化関数の 雛型を 記述したファイルです。
sect30.inc              // RB2タイマーと シリアル受信の 割込み関数を 追加した
                        // 割込みベクトルテーブルの入ったインクルードファイルです。
string_sub.c            // 文字列編集、数値：文字列変換の 関数群のファイルです。
string_sub.h            // string_sub.cの 関数プロトタイプ宣言ファイルです。
TypeDefine.h            // 追加したデータ型のファイルです。
```

また、要求仕様によっては、その用途に合わせI/O回りのサブルーチンを 作りなおす事にも、なりかねないので、私は、ソースファイルを 公開する形で提供しています。 私の IOCSモジュールというか、ソースファイルの紹介を しておきます。

① cpu_select.h // C 用 ヘッダーファイル
② cpu_select.inc // アセンブラ用 ヘッダー
// ファイル

この場合の、cpu_selectというのは、R8C/M110A と R8C/M120Aの どちらを使うかを 宣言するファイルです。

③ IOCS_Readme.txt // コピー時の注意書きを
// 書いています。
④ sect30.inc // 元々 R8Cの Project生成時
// に 自動生成されるファイル
// です。アセンブラのインク
// ルードファイルです。

sect30.inc内には、R8Cマイコンの可変ベクトルテーブルが 含まれます。 各種周辺回路からの割り込みが発生した時に、対応する割り込み番号に対応する、割り込み処理のエントリーアドレスの ラベルを定義する事になります。

Project生成直後は、この sect30.inc には、割り込み処理のエントリーアドレスは 何も登録されていません。 要は、割り込み処理ルーチンが何もない訳です。 で、私の方で、最低限の割り込み処理という事で、

```
.lword    irq_uart0_recv  ; vector 18 / UART受信
.lword    irq_timer_rb2   ; vector 24 / タイマー処理
```

を 宣言しています。
ベクター18の UART受信は、シリアル通信の受信処理です。irq_uart0_recv が、受信割り込み処理エントリアドレスです。 ベクター 24の タイマー処理は、インターバルタイマの時間の基準となる定周期割り込みです。 irq_timer_rb2 が、タイマー割り込み処理 エントリアドレスです。

シリアル通信では、送信割り込みの機能も使えますが、使っていません。 理由は RAM容量が 小さいからです。通信で割り込みを使う際は、リングバッファも、組みにして使う事になります。 受信時のリングバッファは、仕方ないので、デフォルト 64byte 宣言しています。

送信時のリングバッファを 宣言したくなかったので、送信割り込みを使用していません。
送信割り込みを使わない事による弊害は、僅かですが、送信時に SIO周辺回路の送信バッファレジスタが、1文字送信し終わって、空きになるまで、待たされることです。 最大の待ち時間は、38400bpsで、260μsです。 その待ち時間の間に、タイマー割り込み処理も 受け付ける事も 可能なので、全体のパフォーマンスはそう落ちないと思います。

どこから、シリアル通信の送信割り込みの話になったかというと、sect30.incの 割り込みテーブルの話からでしたね。
では、IOCSのモジュールの話に戻ります。

⑤ R8CM1_IOCS_INIT.a30 ; R8Cのアセンブラ
クロック発振器の変更その他基本的な機能が、入ってます。R8Cマイコンは、起動直後は125KHzの遅いクロックで動いてます。それを外付けの 20MHzの水晶発振子に変える事が出来ます。 周波数精度は落ちますが、CPU内蔵の高速RC発振器で 20MHzで動かす事も可能です。

⑥ R8CM1_IOCS_TIMER.a30 ; R8Cのアセンブラ
先ほどの irq_timer_rb2 の割り込み処理関数は、このモジュール内に宣言されています。
1ms周期のタイマー割り込みですが、ソフト的には、1ms単位のタイマ処理2本と 1/10分周した周期で 10ms単位のタイマ処理2本の計4本使えます。

⑦ R8CM1_IOCS_UART.a30 ; R8Cのアセンブラ
先ほどの irq_uart0_recv の 割り込み処理関数は、このモジュール内に宣言されています。 外から受信する データは 割り込み処理内で、最大 64byteまで 貯め込む事が出来ます。 リングバッファ内から受信文字を取り出す関数が あります。送信処理は 1byte 直接 SIO周辺回路のレジスタに書き込みます。 あと、文字列等を送信するラッパー関数を C言語のモジュールで 作成しています。

⑧ R8CM1_IOCS_ADC.a30 ; R8Cのアセンブラ
A/Dコンバータの初期化と、シンプルな A/D変換を 実行する関数を用意しています。
A/D変換で使用できるピンと シリアル通信で使用するピンが、一部 重なっています。
シリアル通信を使わない場合は、A/D入力として使える全てのピンを、使用できます。

シリアル通信を 使用する場合は シリアル通信で使用するピンを避けて、A/D入力の 初期化を行って下さい。

⑨ string_sub.c // 文字列編集用のモジュール
文字列の編集以外に、数値と文字列の変換処理も入れています。 下の IOCS_sio_print.c 内から、呼び出される関数も あります。
このモジュールの関数を宣言する string_sub.h も あります。

⑩ IOCS_sio_print.c // シリアル通信で 文字列
// を出力するモジュール
Byte、Word、LongWordの整数を その値を表す文字列に変換する関数 16進のダンプ処理を 実装しています。 ここまでで、説明した string_subの関数以外は、R8CM12_IOCS.h にプロトタイプ宣言が、行われています。
その他、追加したデータ型を宣言した Type_Define.h が、あります。

IOCSに関わるヘッダーファイルは、R8CM12_IOCS.h を 1本呼び出せばいいです。R8CM12_IOCS.h 内で、cpu_select.h と Type_Define.h を 呼びだしています。

IOCSファイルを HEWの 新規プロジェクトにコピーした直後、すぐに 行う必要がある設定があります。cpu_select.inc にて、使用するマイコンが、M110A か M120A かを 選択する必要が、あります。

以下は、cpu_select.inc の 内容です。

```
    .include    sfr_r8m11a.inc              ; R8C/....
;   .include    sfr_r8m12a.inc              ; R8C/....

MPU_SEL    .EQU  1          ; R8C/M110A を選択
;MPU_SEL   .EQU  2          ; R8C/M120A を選択

DEBUG_LED  .EQU  1  ; デバッグ用 LED 使用する
 ; DEBUG_LED  .EQU  0  ; デバッグ用 LED 使用しない
```

行の左端に、; が、付いていると、それより右は、コメント(無効な行)に なります。

左下の cpu_select.inc の 設定では 左端の ; を 赤で 示しています。 で、緑の文字が 無効な行になっています。 よって、最初の2行は インクルードの sfr_r8m11a.inc が 有効になっています。sfr_r8m11a.inc は R8C/M110A CPU用 I/Oポートや、周辺回路の 名前の宣言を しています。 Project生成時、CPUに応じて HEWにて自動的に付加されます。 内容は、かなり大量の宣言ファイルで 長いです。で、このファイルは、R8C/M110Aと R8C/M120Aでは、若干内容が異なります。 異なる部分は、主に M120Aには、あるが M110A には無い物(足ピンと それに接続される周辺回路)の 宣言です。 HEWにて Project生成時、指定した CPUの sfrファイルが生成されます。 もう片側の sfr ファイルは生成されませんので、使用する CPUに合わせ sfr ファイルの 指定を 行って下さい。

以下は、R8C/M110A 用です。

```
    .include    sfr_r8m11a.inc              ; R8C/....
;   .include    sfr_r8m12a.inc              ; R8C/....

MPU_SEL    .EQU  1          ; R8C/M110A を選択
;MPU_SEL   .EQU  2          ; R8C/M120A を選択

DEBUG_LED  .EQU  1  ; デバッグ用 LED 使用する
;DEBUG_LED  .EQU  0  ; デバッグ用 LED 使用しない
```

以下は、R8C/M120A 用です。

```
;   .include    sfr_r8m11a.inc              ; R8C/....
    .include    sfr_r8m12a.inc              ; R8C/....

;MPU_SEL    .EQU  1          ; R8C/M110A を選択
MPU_SEL    .EQU  2          ; R8C/M120A を選択

DEBUG_LED  .EQU  1  ; デバッグ用 LED 使用する
;DEBUG_LED .EQU  0  ; デバッグ用 LED 使用しない
```

中段の MPU_SELは、CPUの選択です。
ここも、必要でない側を ; で コメント化して下さい。

DEBUG_LED は p3_7 のポートに 基板上のLEDが Lowで 点灯する状態で 接続されている事を想定して IOCSの タイマー割り込み処理にて 割り込み処理を行っている間だけ点灯します。 タイマー割り込み処理が正常に行われている場合、1ms周期で 10～20μs 点灯します。
見た目は、LEDが 薄く点灯します。

p3_7を 別の用途で使用したい場合は、
DEBUG_LED .EQU 0 を 有効にして下さい。
タイマー割り込み処理内では、アクセスされなくなります。

あと、似たようなファイルで、cpu_select.h というファイルが あります。 設定する内容は、同じですが、C言語用なので、コメントは ; では無くて // で 行って下さい。

sample_main.c ソースに関して

main 関数の 先頭部分と、初期化処理関数は、毎回　ほぼ同じ作りになります。で あれば雛型を　１本作っておく方が、手っ取り早いかなと、思った次第です。　中身を見てみましょう。

右のソースが、sample_main.c の前半です。最初にインクルードファイル　R8CM12_IOCS.h を 取り込んでいます。

関数プロトタイプ宣言で　void　main(void);　と　void　init_proc(void); の ２本を宣言しています。　init_proc 関数は　次のページでお見せします。

制御系で使う　メイン関数は　殆どの場合、最初に　各 I/O の　初期化処理を 行い、次にメイン処理の エンドレスループに 入ります。

```
//   インクルードファイル
// ------------------------------------------
#include "R8CM12_IOCS.h"   // IOCS サブルーチン宣言

//   関数プロトタイプ宣言
// ------------------------------------------
void  main( void );       // メイン関数
void  init_proc( void );   // 初期化処理メイン

//************************
//**  メイン処理        **
//************************
void main(void)
{
     init_proc();             // 初期化処理

     while( 1 )
     {
     }
}
```

init_proc 関数は、下の関数名と 始まり中カッコから、右側のソースに続きます。

起動直後は 125kHzの低速クロックなので、20MHzの高速クロックに切り替えます。

ちなみに、20MHzで動かしても マイコン単体では、10mAも 消費しません。 かなり低消費電力です。 切り替えは、setup_ext_osc関数で、外付けの水晶発振子を使うか、setup_in_osc20関数で、CPU内部の RC発振器を使う事になります。 クロック周波数の精度、安定性は、水晶発振子が、優れています。

RC発振器を使う場合は、発振周波数のブレが、±1%以内で 問題ない場合で、水晶発振子の足ピンを 他の用途に使いたい
(要は、足ピンが足りない) 場合です。

```
void  init_proc( void )
{
//     高速クロック切り替え設定
// ------------------------------------------------
       setup_ext_osc();  // 外部発信子使用
//     setup_in_osc20(); // 内部 20MHz RC OSC使用
//  R8C/M120Aの I/O ポート初期化
// ------------------------------------------------
       p1   = 0x00;          // P1 出力値 仮初期化
       p3   = 0x80;          // P3 出力値 仮初期化
       p4   = 0x00;          // P4 出力値 仮初期化

       pd1   = 0x44;         // ポート１ 入出力方向設定
       pd3   = 0x80;         // ポート３ 入出力方向設定
       pd4   = 0xC0;         // ポート４ 入出力方向設定
//     周辺回路 初期化
// ------------------------------------------------
//     adc_init( 0x010F );      // 内蔵 ADC 初期化
                  //( ch.0 ～ ch.3 使用 ) M120A用
//     adc_init( 0x0102 );      // 内蔵 ADC 初期化
              // ( ch.1 のみ使用 ) M110Aで使用した
       sio_init( 0, 0 );     // シリアル通信  初期化
       // ( 38400bps, data=8, Parity=Non, stop=1 )
       timer_init();  // インターバルタイマー初期化
       asm( " fset I" );         // 割り込み処理 許可
}
```

R8C/Mマイコンにおいて、実装されている I/O ポート初期化は、p1と p3と p4 3つです。M120Aは、p1 が 8bit揃っていますが、M110Aは、p1 は 7bitです。 他のポートも M110Aの方が ピン数が少ないです。 ポート表は 次のページに示します。 で、各ポートに出力する初期値は、各 bitが 0 であれば、Lowを 出力します。1で あれば Hi を 出力します。 よって正論理です。 pd1 ～ pd4 は ポートの入出力方向の設定です。 各 bit に 1 を 書き込めば、出力ポートになります。 0 を 書き込めば 入力になります。 用途に応じて、ポートの 入出力の方向や 初期値を 決めて下さい。

次は、周辺回路の初期化ですが、ここで呼んでいるのは、IOCS内の初期化関数です。

```
//  R8C/M120Aの I/O ポート初期化
// ------------------------------------------------
      p1  = 0x00;        // P1 出力値 仮初期化
      p3  = 0x80;        // P3 出力値 仮初期化
      p4  = 0x00;        // P4 出力値 仮初期化

      pd1  = 0x44;       // ポート１ 入出力方向設定
      pd3  = 0x80;       // ポート３ 入出力方向設定
      pd4  = 0xC0;       // ポート４ 入出力方向設定
//    周辺回路 初期化
// ------------------------------------------------
//    adc_init( 0x010F );     // 内蔵 ADC 初期化
                  //( ch.0 ～ ch.3 使用 ) M120A用
//    adc_init( 0x0102 );     // 内蔵 ADC 初期化
            // ( ch.1 のみ使用 ) M110Aで使用した
      sio_init( 0, 0 );    // シリアル通信  初期化
      // ( 38400bps, data=8, Parity=Non, stop=1 )
      timer_init();  // インターバルタイマー初期化
      asm( " fset I" );       // 割り込み処理 許可
}
```

R8C/M110AN

Port. 1							
b7	b6	b5	b4	b3	b2	b1	b0
P1_7	P1_6	P1_5	P1_4	P1_3	P1_2	P1_1	
8	9	10	11	12	13	14	

Port. 3							
b7	b6	b5	b4	b3	b2	b1	b0
P3_7							
1							

Port. 4							
b7	b6	b5	b4	b3	b2	b1	b0
P4_7	P4_6	P4、P4_7は 水晶で使用					
3	5						

Port. A							
b7	b6	b5	b4	b3	b2	b1	b0
PA_0は /RESET で 使用							PA_0
							2

P1_4 ～ P1_6は 通信 PGM書き込みで使用

R8C/M120AN

Port. 1							
b7	b6	b5	b4	b3	b2	b1	b0
P1_7	P1_6	P1_5	P1_4	P1_3	P1_2	P1_1	P1_0
13	14	15	16	17	18	19	20

Port. 3							
b7	b6	b5	b4	b3	b2	b1	b0
P3_7		P3_5	P3_4	P3_3			
2		9	10	11			

Port. 4							
b7	b6	b5	b4	b3	b2	b1	b0
P4_7	P4_6	P4_5			P4_2		
4	6	12			1		

Port. A							
b7	b6	b5	b4	b3	b2	b1	b0
							PA_0
							3

最初の adc_init 関数は、A/Dコンバータの 初期化処理です。 引数の 下位 8bitの値に より使用する A/Dコンバータの足ピンを 指定できます。 詳細は R8CM1_IOCS_ADC.a30 を 参照して下さい。 尚、A/D変換に使用できる足ピンの一部が、シリアル通信にも使われているので多重初期化に ならないように注意して下さい。

重なっているポートは p1_4 です。
(AN4であると共に、TxD でも あります。)
因みに adc_init(0x010F); と adc_init(0x0102) の2つの初期化ルーチンを 書いてますが、引数 0x010Fは M120Aにて A/D入力を 4チャネル設定出来ます。 M110Aは、AN0の 端子が、無いので、ch.1 ～ ch.3 の 3チャネルとなります。 A/D入力が 1チャネルで良ければ、引数 0x0102 で使用して下さい。 AN1が 入力チャネルになります。

sio_init 関数は、シリアル通信の初期化です。
今のところ、ボーレイトの分周比の関係で 38400bps が 安定した最高値になっています。
38400bps、データ長 8bit、Nonパリティ、1 Stop bit で使って下さい。 引数は、0、0 です。

timer_init関数は、呼び出すだけで引数は、ありません。
最後の asm("fset I"); は、割り込み許可のインラインアセンブラ命令です。

```
//    周辺回路 初期化
// ------------------------------------------
//    adc_init( 0x010F );     // 内蔵 ADC 初期化
                  //( ch.0 ～ ch.3 使用 ) M120A用
//    adc_init( 0x0102 );     // 内蔵 ADC 初期化
            // ( ch.1 のみ使用 ) M110Aで使用した
      sio_init( 0, 0 );    // シリアル通信  初期化
      // ( 38400bps, data=8, Parity=Non, stop=1 )
      timer_init();  // インターバルタイマー初期化
      asm( " fset I" );       // 割り込み処理 許可
}
```

IOCSのコピー後、やる事は、まだ ありますが
殆どが HEW上で 行う事に なりますので、動画
にて お見せします。