百円R8CマイコンIOCSに I2C機能を 組み込む

過去の動画にて、R8C/M110Aと R8C/M120Aの2つに、それぞれ独立して I2C機能を 実現しました。

因みに I2C実装に関わる動画の タイトル番号は M110Aの方が、023：ハード編、024：ソフト編になってます。 M120Aの方が、036：ハード編、037：ソフト編になっています。 あと、024の方はI2Cのプロトコルというか、伝送手順の説明も、しています。 興味があれば見て下さい。

ソースを よく見比べないと何ともいえませんが多分、ハード寄りの部分で 互換性のない部分（ 使用するマイコンの足ピン ）が いくつかあると思います。、特にM110Aの方が、全部で 14ピンなので、VCC、GND、RESET端子、水晶発振子接続端子、モード設定にて、6ピン使用するので残り 8ピンです。

プログラム書き込みも含めたシリアル通信に3ピン使用するので、残り 5ピンです。 で I2Cで2ピン使用します。で、残り 3ピン。 他の空きピンを アプリで使いたい場合もあるので、そうなるとかなり厳しいです。 それと、クロック端子を 高速に動作させるために 同じポートの 他のビットと 干渉しないようにするために、ポートのビット構成に条件が伴います。 そのあたりの事を考えると、やはり足ピンは、全く同じポート構成にするのは、無理という事で、諦めました。

アプリの側からは、IOCSの I2Cのサブルーチンを使用するのは、M110Aと M120Aにて、同じ関数名、同じ引数仕様で実現できます。

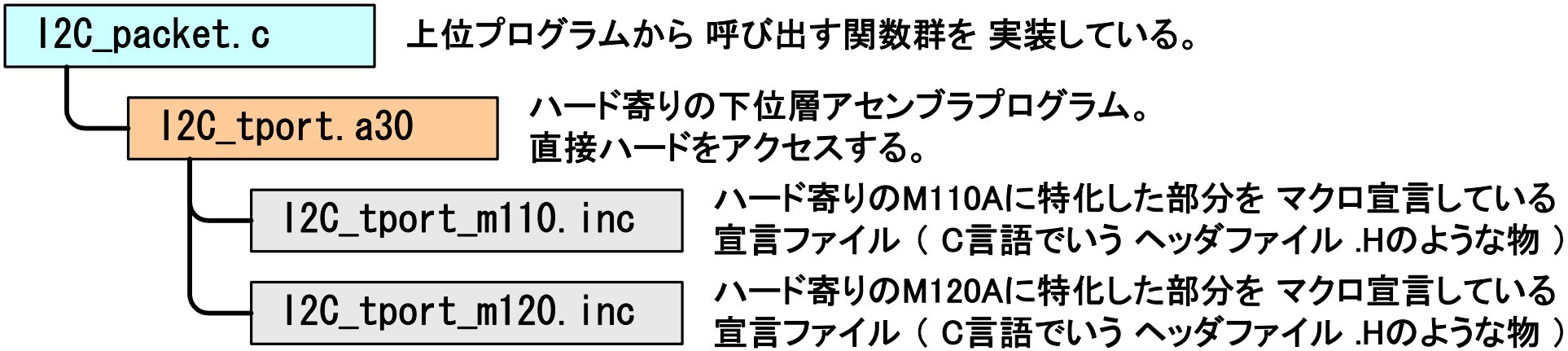
よって
足ピンの割り当てに だけ 注意して下さい。

I2Cの信号線のピンアサインは

	R8C/M110A	/	R8C/M120A
SCL =	１番 (p3_7)	/	12番 (p4_5)
SDA =	８番 (p1_7)	/	1番 (p4_2)

に、なっています。

★ I2Cのサブルーチンの ソースファイル構成: （ 以下のモジュールを IOCSに組み込みます。）



cpu_select.inc により M110A用、M120A用の どちらのプログラムを 生成するのかを
判定し 2つの .inc ファイルを スイッチして アセンブルしている。

★　I2C_packet.c 内の関数群を 紹介します。

---

①　void　init_i2c_port(BYTE spd);　// I2Cポートの初期化、使用前に 必ず１回
　　// 一番最初に呼び出す事。この関数だけ、I2C_tport.a30側に 入っている。
　　// 転送速度設定： spd = 0: 400Kbps 、spd = 1; 100kbps （ 凡その速度 ）

②　BYTE　i2C_check_slave(BYTE adr);　　// 指定アドレスのスレーブ有無確認
　　// adr = 7bit I2C アドレス、関数値 = 0 スレーブ有り、= 80h スレーブ無し

③　void　i2c_write_7b(BYTE adr, BYTE* buf, WORD len); // 7bitアドレス
　　// データ書き込み
　　// adr: 7bit I2C アドレス、buf: データバッファ、len: 送信するデータ長

④　void　i2c_read_7b(BYTE adr, BYTE* buf, int len); // 7bitアドレス
　　// データ読み出し
　　//　adr: 7bit I2C アドレス、buf: データバッファ、len: 受信データ長

　次に出てくる２つの関数は、③の i2c_write_7b で 代用出来るのですが、I2Cデバイス側で、複数のレジスタを持っている場合に、最初に レジスタアドレス1バイトを 出力し、次に、データを、1バイト書き込む、あるいは複数バイトの データ書き込みを行う場合に、視認性を良くする目的で 用意しました。

⑤ void i2c_write_7bfix(BYTE adr, BYTE fdt, BYTE ptn, WORD len); // 7bitアドレ
// ス データ書き込み(2) adr: 7bit I2C アドレス fdt: 先頭固定バイト(デバイス
// 内のレジスタアドレス等) ptn: パターンデータ、len: 書き込み個数
// この関数は 最初 fdt 1バイトを出力して パターンデータを 1個、または 同じパ
// ターンデータを複数個 書き込みます。

⑥ void i2c_write_7bfv(BYTE adr, BYTE fdt, BYTE *ptr, WORD len); // 7bitアドレ
// ス データ書き込み(3) adr: 7bit I2C アドレス fdt: 先頭固定バイト(デバイス
// 内のレジスタアドレス等) ptr: データバッファのポインタ、len: 書き込み個数
// この関数は 最初 fdt 1バイトを出力して バッファ内データを 先頭から順次読み
// 出して 複数個 書き込みます。

I2Cの 伝送制御手順には、変則的な物が有り、通常の１本の電文は スタートコンディションに始まり、ストップコンディションで終わります。 スタートコンディションの次の先頭バイトが、I2Cのコントロールバイトで、１バイトの b7～b1 が スレーブアドレス 7bitで、最下位bitが、Read/Writeを 表しており、1=Read、0=Write です。 変則的な物というのは、リピートスタートコンディションというものです。 これは、最初 Writeモードでスタートし、その電文の途中で、リピートスタートコンディションで、Readモードに 変更するという機能です。 信号の出し方の詳細は、024の動画にて説明しています。

⑦ void i2c_write_rep_7b(BYTE adr, BYTE* buf, WORD len); // 7bit アドレス
// リピートスタート前半／データの書き込み
// adr: 7bit I2C アドレス 、buf: データ格納バッファ 、len: 書き込むデータ長
// リピートスタートコンディションを出す直前までの 電文 書き込み処理です。

⑧ void i2c_rep_read_7b(BYTE adr, BYTE* buf, int len); // 7bit アドレス
// リピートスタート後半／データの読み出し
// adr: 7bit I2C アドレス 、buf: データ格納バッファ 、len: 読み出すデータ長
// リピートスタートコンディションを 出した後の 電文 読み出し処理です。

i2c_packet.c の 関数の説明は、以上です。

尚、10ビットアドレスの I2C通信は、一度も使用した事が 無いので省略します。

I2Cの 通信仕様を説明した書籍に、10bitのアドレス規格がある事を書いてあったので
最初は、用意していましたが、7bitアドレスでも アドレスレンジは 127あるので、
I2Cの２本の信号線に、100台以上のスレーブデバイスを接続するというのは、ファンアウト
（ 信号線のドライブ能力 ）の問題もあり、現実的ではないと 私は考えます。
じゃ、どれくらい接続出来るのと聞かれると、接続するデバイスにも影響を受けるので
ハッキリとは いえませんが、条件が良くて 10台以下のデバイスと考えます。

そして、今回のI2Cの テストを兼ねた実験ですが、I2Cデバイスで、
データが 分かりやすい物で試すとすると、RTC　リアルタイムクロックが
データが 時刻なので　分かりやすいと判断しました。
それと、RTCへの 時刻の設定、時刻の読み出しは、I2Cの 書き込み、読み出しの
両方のテストになるので好都合と考えました。

I2Cの IOCSへの組み込み 及び、RTCのアクセス処理の プログラム作成等は
地味な作業なので、実験のソフトが、出来た状態で、確認動作の状態を
動画にて紹介する事にします。