R8C/M110Aで 通信が出来ない障害

前回の動画にて、R8C/M110Aにて、A/Dコンバータを使用すると、シリアル通信が 出来なくなる障害の説明を行います。 要は、足ピン数が、非常に少ないため ADコンバータとして使える足ピンに、TxDも 重複して機能をアサインしてあるので、切り替えて使う事になります。右の図は、I/O ポート1の ビットマップ表です。ポート1の b4にて AN4と TxDが、重なっています。 b5も、アナログ入力とは重なっていませんが、注意が必要です。

初期化処理において、
最初に シリアル通信機能の初期化を行い
次に A/D変換機能の初期化を行っていました。

よって、原因は、ポート1の b4、b5 の 機能切り替え設定を、最初 シリアル通信で使用する設定を書き込んでいましたが、A/Dコンバータの初期化処理が 上書きして壊していました。

Port.1							
b7	b6	b5	b4	b3	b2	b1	b0
8	9	10	11	12	13	14	←足ピン
P1_7	P1_6	P1_5	P1_4	P1_3	P1_2	P1_1	
AN7			AN4	AN3	AN2	AN1	
		RxD	TxD				

処置は、IOCSルーチンの R8CM1_IOCS_UART.a30(シリアル通信処理)と R8CM1_IOCS_ADC.a30(ADC処理)との間で、open_sw という フラグ変数を設けて、シリアル通信を、オープンしていたら、ポート1の b4、b5 を シリアル通信で、使う設定に初期化するようにしました。 よってこの場合、AN4は 使えません。

110Aは、これでいいと思いますが、120Aは、足ピンが、20本あるので、シリアル通信を AD設定以外の、別のピンにアサインする事も出来ます。よって、120Aの場合は、シリアル通信の足ピン設定に応じて、ポート1の b4、b5 の 設定を変える必要が ありま す。

具体的な設定箇所

ルネサスの資料
R8C/M11Aグループ、R8C/M12Aグループ ユーザーズマニュアル ハードウェア編の 147ページに
ポート1機能マッピングレジスタ1（PMH1）の 設定に関わる説明が載っています。
すみません。 小さくて見るのが 困難ですが、右の図です。

ポート1の b4、b5 ビットを 両方 シリアル通信 TxD、RxDとして設定するには
pmh1 = 0x05; と設定します。
両方とも、ADあるいは、I/Oポートとして使用する場合は、pmh1 = 0x00; と設定します。右の図の赤枠で囲ったところですが、3bit ありますが 上位 bit bx は別のレジスタの bitで、通常 0 です。
タイマーRC で、PWM等を 使う場合は bx が 1 に なります。

12.3.7 ポート1機能マッピングレジスタ1 (PMH1)

アドレス 000C9h

ビット	b7	b6	b5	b4	b3	b2	b1	b0
シンボル	P17SEL1	P17SEL0	P16SEL1	P16SEL0	P15SEL1	P15SEL0	P14SEL1	P14SEL0
リセット後の値	0	0	0	0	0	0	0	0

ビット	シンボル	ビット名	機能	R/W
b0	P14SEL0	ポートP1_4機能選択ビット	bx b1 b0 0 0 0 : I/Oポート またはAN4入力 0 0 1 : TXD0 0 1 0 : RXD0 0 1 1 : INT0 1 0 0 : TRCIOB 上記以外 : 設定しないでください (bx : PMH1Eレジスタの P14SEL2ビット)	R/W
b1	P14SEL1			R/W
b2	P15SEL0	ポートP1_5機能選択ビット	bx b3 b2 0 0 0 : I/Oポート 0 0 1 : RXD0 0 1 0 : TRJIO 0 1 1 : INT1 1 0 0 : VCOUT1 上記以外 : 設定しないでください (bx : PMH1Eレジスタの P15SEL2ビット)	R/W
b3	P15SEL1			R/W
b4	P16SEL0	ポートP1_6機能選択ビット	b5 b4 0 0 : I/Oポート またはIVREF1入力 0 1 : CLK0 1 0 : TRJO 1 1 : TRCIOB	R/W
b5	P16SEL1			R/W
b6	P17SEL0	ポートP1_7機能選択ビット	b7 b6 0 0 : I/Oポート またはAN7入力 またはIVCMP1入力 0 1 : INT1 1 0 : TRJIO 1 1 : TRCCLK	R/W
b7	P17SEL1			R/W

ソース修正箇所

修正箇所は、赤文字の部分です。
修正箇所は、5か所に散在してます。
uart0_act_pinsel_1: で 探せば見つかります。
左のソースは、R8CM1_IOCS_UART.a30 です。

```
;       外部参照宣言
; ------------------------------------------------
        .glb    open_sw         ; シリアル通信 Open SW

 ------------------------------------------------

uart0_act_pinsel_1:
        mov.b   #1, open_sw     ; OPEN SW ON
 ------------------------------------------------
uart0_act_pinsel_2:
.IF     MPU_SEL==2      ; ★ R8C/M120A の場合のみコード生成
        mov.b   #2, open_sw     ; OPEN SW ON
 ------------------------------------------------
uart0_act_pinsel_3:
.IF     MPU_SEL==2      ; ★ R8C/M120A の場合のみコード生成
        mov.b   #3, open_sw     ; OPEN SW ON

 ------------------------------------------------

open_sw:        .blkb   1       ; シリアル通信 Open SW
```

緑の横棒は、上下の行間が、離れている事を意味します。
こっちは、p032: で 探せば見つかります。
右のソースは、R8CM1_IOCS_ADC.a30 です。

```
        .glb    open_sw         ; シリアルOPEN_SW
 ------------------------------------------------

        mov.b   open_sw, r0l    ; OPEN_SW 確認
        jz      p030            ; ROL = 0
        dec.b   r0l             ; ROL = 1
p032:
        dec.b   r0l             ; ROL = 2
        jz      p033
        jmp     p030            ; ROL = 0
p033:                           ; ROL = 2
        mov.b   #04h, pmh1      ; Port1機能マッピング設定(H)
                                        ; シリアル通信を行う時
        jmp     p031
p032:                           ; ROL = 1
        mov.b   #05h, pmh1      ; Port1機能マッピング設定(H)
                                        ; シリアル通信を行う時
        jmp     p031
p030:                           ; ROL = 0
        mov.b   #00h, pmh1      ; Port1機能マッピング設定(H)
                                ; シリアル通信を使わない時
p031:
```

A/D量子化数から温度データへ変換

前のページは、いきなり R8Cマイコンのアセンブラのソースで 申し訳ありません。
で、次も アセンブラのソースの話となります。

前回の動画で作った10bit A/Dコンバータ用の量子化数を、サーミスタ103ATの 温度値に変換するテーブルデータを 取り込むソースの説明をします。

右のソースは、10bit分解能 ADC用の 103AT サーミスタの温度読み取りテーブルです。

前も、少し説明しましたが、 ; は、それより右がコメントになります。

_at103_table_10: は ラベルと呼びます。ラベルはソース左端から記述して、最後に : を付けます。ラベルに使える文字は、だいたいC言語の変数に付けられる文字と同じです。 先頭に 数字を持って行くと、エラーになります。

C言語では、実態がある(メモリ上に存在する)もので、名前のある物は、変数名、関数名が有ります。 しかし、アセンブラのラベルは、アドレスの指標でしかありません。 その下が、データでも、プログラムでも、かまわないのです。

```
; 10bit 分解能 ADC用 103AT サーミスタ
; 温度 読み取りテーブル ( 1024 サンプル )
; ------------------------------------------------------
_at103_table_10:
    .word    -3000    ; No.   0 . -300.0 ℃ , 0.0000 V
    .word    -3000    ; No.   1 .
    .word    -3000    ; No.   2 .
    .word    -3000    ; No.   3 .
    .word    -3000    ; No.   4 .

    .word    -3000    ; No.  50 . -300.0 ℃ , 0.2441 V
    .word    -3000    ; No.  51 .
    .word    -398     ; No.  52 .  -39.8 ℃
    .word    -394     ; No.  53 .  -39.4 ℃
    .word    -390     ; No.  54 .  -39.0 ℃
```

最低限の R8Cアセンブラの説明

アセンブラのラベルは、アドレスの指標でしかない。 という事に、少々面食らった方も いるかもしれません。 元々は、遥か昔 マシン語で、プログラムコードを打ち込む時代も、あったのです。

マシン語でというと、2進数表記 1 と 0 の長い表記では、あまりに効率が悪いので、16進数、または8進数で、マシン語は、表記します。

で、16bit マシンであれば、16個並んだスイッチを、パチパチと切り替えて、マシンコードの 書き込みを 行いました。 その効率の悪さを改善する目的で、アセンブラは 作られました。 アセンブラは、基本 マシン語と 1対1に 対応する言語です。 で、例えば 呼び出したいサブルーチンがあったとして、そのサブルーチンの 先頭アドレスが、呼び出しアドレスとなります。汎用的なアセンブラ表現では CALL 200H とか、表す事になります。

しかし、 CALL 200H では、それよりも前のコードに、追加が、発生した場合、サブルーチンの 先頭アドレスが、後ろにずれてしまいます。

何番地後ろに、ずれたかを確認して CALL命令の オペランドを 200H から 210H に 変更することになります。 これは、面倒な事で、バグの原因にも、なります。 よって、サブルーチンの頭にアドレスの指標となる ラベルを、付ける事になったのです。 ラベルは、アドレスの指標だけで、メモリは 消費しないので、ラベルを使う事で、アドレスが、ずれる事は ありません。よって、サブルーチン先頭に SUB_A とか ラベルを 付けます。 そして呼び出す時は、CALL SUB_A とすればいい事になります。 SUB_Aの アドレスがズレても、ラベルを使えば、呼び出し側は 影響を受けません。 因みに R8Cマイコンでは、サブルーチン呼び出しは、CALL ではなくて JSR と記述します。

次に

```
.word    -398    ; No.52  -39.8 ℃
```

とか、温度変換テーブルの一部を持ってきましたが、.word は、初期値を持った 2byte整数を メモリに確保します。 ,word の後に 書き込む値を 記述します。 ; 以降は、コメントです。
1byte単位の場合は .byte に なります。
今回の場合は、10bit A/Dコンバータの量子化数に対応した温度テーブルなので、.word の 宣言が 1024 個 あります。 このテーブルのファイル名は、Ther_ADC_TBL_10.inc です。

このテーブルファイルを読み出す アセンブラファイルの名前は ADv_ondo.a30 です。
R8Cマイコンの場合アセンブラファイルの 拡張子は、.a30 です。
つぎは、ADv_ondo.a30 のアセンブラソースの説明を、行います。

ADv_ondo.a30

```
    .include    cpu_select.inc
    .section    program, CODE,ALIGN

;     量子化数 --> 温度 ( 0.1℃単位 )
;     変換テーブル
; ------------------------------------
    .include   Ther_ADC_TBL_10.inc
```

上は、アセンブラソース ADv_ondo.a30 の 先頭部分です。 .include cpu_select.inc は、CPUが、M110Aか、M120A かを指定する役目と、メーカーの周辺回路レジスタ宣言ファイルを取り込む役目を 行います。

.section program, CODE,ALIGN は、
セクション情報という リンカに渡される、メモリ上の 配置情報です。
単純に このままで 使って下さい。

.include　Ther_ADC_TBL_10.inc が、今回のA/D量子化数を、温度値に変換するテーブルファイルの　読み込み箇所です。 単純に、この場所にTher_ADC_TBL_10.inc の 内容が　展開されます。

ソースの次ですが、8行の ＊ で囲ったコメントは、このアセンブラ関数の説明です。
.glb　は、外部宣言、外部参照を意味します。
後ろの $get_adv_ondo は、リンカに渡される名前です。この関数はC言語から呼び出す関数です。
$get_adv_ondo: はアセンブラ内では、単に ラベルです。
C言語側で、呼び出す場合は、頭の $は必要ありません。
ondo = get_adv_ondo(ad);
になります。
今回は、アセンブラ側で　読み出す処理を用意しました。

```
; ********************************************************************
; **  ADC量子化数 ---> 0.1℃単位の温度値を 返す                      **
; ** ---------------------------------------------------------------- **
; **  引数 :   R1  ADC量子化数 ( 0 ～ 1023 )                         **
; ** ---------------------------------------------------------------- **
; **  関数値 :   R0 = 0.1℃単位の 温度値 ( -395 ～ 1095 )             **
; **                                  -39.5 ～ 109.5 ℃              **
; ********************************************************************
      .glb  $get_adv_ondo
$get_adv_ondo:
      push.w      a0                  ; A0 保存
      shl.w #1, r1                    ; R1 の 量子化数を 2倍にする
      mov.w r1, a0                    ; R1 を A0 に 入れる
      add.w #_at103_table_10, a0      ; 変換テーブル先頭アドレスを A0 に 加算
      mov.w [a0], r0                  ; A0が 示すテーブル内容を R0 に 入れる
      pop.w a0                        ; A0 元に戻す
      rts
```

最後に、アセンブラソースの場合、最後には
.end を記述して下さい。
アセンブラ ADv_ondo.a30 ファイルは、先ほど
２つに分けた部分と 最後の .end だけです。

今回もダウンロード出来るようにしますので
自分のプロジェクトに 組み込まれる場合は、
ADv_ondo.a30 と Ther_ADC_TBL_10.inc をコピー
して、ADv_ondo.a30 を プロジェクトに加えれば
使えるように なります。
中には、
プロジェクトに加えるのは、どうするの。？
という人もいるかも 知れませんので、これについ
ては、動画で、HEWの操作を 説明します。

有線LANルータが
ハングする別の可能性

有線LANルータの、放熱対策に関わる動画は今回で 終わりにします。
しかし 有線LANルータが、ハングする原因と思われる事として今回、放熱対策を行いましたがハングする理由は、別に あるかもしれません。

メーカーさんに 失礼な事を書くかもしれませんがあくまで、その可能性も考えられるという事でファームウェアの潜在的なバグという事も考えられます。 Windos環境では、GetTickCount関数というAPI関数が、あります。

GetTickCount関数
パラメータ無し。
戻り値： 符号なし 32bit 整数
システムを起動した後の 経過時間を
ミリ秒単位で 返す。

比較的 短い時間で、イベント発生を 時間監視するような用途 あるいは、短い時間待ちを作成する用途で使用されます。

通常のクライアントマシンのように、朝 電源を入れて、帰るとき、電源を切るような使い方の場合は、問題は、発生しません。
しかし、サーバーマシンのように、電源入れっぱなしの場合は、忘れた頃に問題が発生する場合が あります。 内部で 継続的に 32bit 整数を、インクリメントしているので、いずれかは オーバーフローして 0 に 戻ります。
そのオーバーフロー直前に、タイマー監視 開始時間として GetTickCountの値を読み取り、TickCount が 0 に 折り返した後に、経過時間確認に 2回目以降の読み取りを行うと、いつまで経っても、後で読んだ値が、最初に読んだ値を越える事は、ありません。 因みに $2^{32}-1$は、4,294,967,295 です。 これを、1日分の ミリ秒値 1000x60x60x24 で割ると 約 49.7日になります。

という事で、今回の有線LANルータの　1ヶ月以上経過すると ハングするという現象に　電源ON後、何日経過してダウンしたかとか 計った事が無いので、ハッキリした事は言えませんが、可能性は 否定出来ないと 思います。
　であれば、49日経過する前に、ルータを再起動するといい。 という事になります。

IT企業は、別ですが、個人の家で、電気を消して寝ている時は、パソコンも電源を落としているので、ルータとかも、一括電源を落としていいと思います。 例えば、夜中の 12時に ルータの電源を OFF して、朝　7時に 電源を ONする。
それを自動で、運用できると いいですね。
　そのような タイマーも、今 検討中です。

今は、ギガビットイーサネットになっているのでイーサネットポートも 処理するCPUも かなり高速なものを使用しているでしょうから、夜中の電源を切る事は、省エネにつながるのではと思います。

余談、A/Dコンバータのデータについて

　ページの右半分が、空いていたのでちょっと余談を、書いておきます。
　A/Dコンバータは、アナログ信号の電圧を読み取って、電圧値に対応する デジタルデータ量子化数に、変換します。 多分、この動画を視聴されている方は、この事はご存じと思います。
　ただ、実際に使ってみると　下位 2bit 程度が、パラパラとノイズが乗って動きます。 で、A/Dコンバータは、下位 2bit 程度は、ノイズで、パラパラと動いて当たり前と考えて下さい。 で、このノイズの影響を少なくするには 今回は、連続5回の単純平均処理を行いました。 場合によっては　移動平均を取る場合も あります。
　それと、今回A/D入力の前段に、5Vレール to レールのオペアンプを入れました。 このタイプのオペアンプは スルーレイトが遅いので、それが いい意味で、ローパスフィルターになっているという事も あります。