前回のシリアルポートで　初期化して送信出来ないトラブルについて

最初、考えにくい現象で　悩んでましたが、

それとは別に、前々回タイマー割り込みを使用したインターバルタイマ　4チャネルの実装をしました。 で それとは別に　今回単純にプログラムで　forループによる ミリセコンド単位の大雑把な　Wait処理関数を作り 追加しました。

その時、最初に単体試験で 1ミリセコンドのforループの　回数の調整を やっていました。その後、他のモジュールと連結して　テストし始めたら、forループの　Wait処理の 時間が短くなっている事に気付きました。 for文の 1ミリセコンドの　回る回数は 同じだったので そうなると　1回の回る速度が速くなった事になります。
エーッ、何で　？
また悩ましいトラブルが　発生しました。

しかし、この現象に関しては　過去に　似たような事を経験していて　原因と考えられる事が　推測出来ました。 但し、早くなる原因は分かるのですが、何故途中から早くなりだしたかが、分りません。 早くなる理由は、空ループの for文の カウンタ変数が　i とすれば その i は 通常 RAMメモリ上に　確保されます。 それが、RAMメモリではなくて、CPUのレジスタ上に 確保されて　0から、最終値まで回ると　メモリをアクセスする時間が 無くなるので その分早くなります。俗にいうレジスタ変数です。 でも、これは 人間がアセンブラで作らないなら だれがやるんだ。というとコンパイラの　最適化処理を行う オプティマイザです。ルネサスの H8マイコンのオプティマイザに関わる資料で H8マイコンの オプティマイザは レジスタ変数で最適化する機能も あるようです。

で、forループの　Wait処理の 時間が短くなった事が、コンパイラの　オプティマイザによる最適化の影響なのか確認するため、forループのカウンタ変数　i と　j に　volatile 修飾子を付けて、 i と j に　関わる最適化を除外する実験を 行いました。

その動画を　お見せします。

それと、今回の事で　コンパイラの　オプティマイザの最適化処理が　シリアル通信の障害を作り出している可能性も　あるのではないかと今、考えています。 シリアル通信の障害にも光が 見えて来た気がします。シリアル通信のSCI周辺回路の初期化処理において、言葉 悪いですけど　コンパイラの オプチィマイザに　明らかに 改ざんされそうな所が　一箇所あります。 原因となる箇所は見つけましたが、どのように　対応するかは検討中です。

詳細は、動画の後に　説明します。

ルネサス資料による volatile修飾子の説明

volatile修飾子

volatile修飾子をつけて変数宣言すると，その変数は最適化の対象から外され、
レジスタに割り付ける最適化などを行わなくなります。 volatile指定された変数に
対する操作を行うときは、必ずメモリから値を読み込み、操作後にメモリへ値を
書き込むコードに なります。
また，volatile指定された 変数のアクセス幅も 変更されません。

volatile指定されていない変数は，最適化によってレジスタに割り付けられ、
その変数を メモリからロードするコードが 削除されることがあります。 また，
volatile指定されていない変数に 同じ値を代入する場合、冗長な命令と解釈されて
最適化により 命令が削除されることもあります。
特に 周辺I/Oレジスタへ アクセスする変数や、割り込み処理で 値が変更される変数、
また、外部から 値が変更される変数に対しては、volatile指定する必要が あります。

volatile指定すべきところで 指定されていなかった場合、次の現象が起こることがあります。
① 正しい計算結果が得られない。
② ループ内で 変数を使っていた場合、ループから 抜け出せない。
③ 命令の実行順序が変わる。 ④ メモリのアクセス回数・アクセス幅が変わる

volatile修飾子 （ 前ページの続き ）

ただし、volatile指定した変数を 使用する際,
ある区間で その変数の値が 外部から変更されないことが 自明な場合、
volatile指定されていない変数に、その値を代入して
その変数を参照することにより、その変数が最適化され
実行速度が 向上する可能性が あります。

という事でした。

この、volatile の 説明は 役に立ちました。
これを 踏まえて H8の シリアル通信周辺回路 SCI の
初期化、1 byte送信処理の ソースを見てみます。

シリアル通信SCI の　初期化処理

SCI は、チャネル 0 と　チャネル1の
2つがありますが、内容が殆ど同じ
なので、チャネル 0 で 説明します。

```
//**********************************
//**  SCI ch.0 初期化処理          **
//**********************************
static void  init_sci_0( void )
{
      P9DDR = 0x03;                 // P9.1(TxD.1)と P9.0(TxD.0)を 出力にする。
//    SCI0.SCMR.BIT.SMIF = 1;       // P9.0 = TxD.0 , P9.2 = RxD.0

      SCI0.SCR.BYTE = 0;            // SCIを 停止
      SCI0.SMR.BYTE = S_tbl.cks;    // 語構成 "N81", ボーレイトクロック選択
      SCI0.BRR = S_tbl.brr;         // ボーレイト分周値設定
      SCI0.SCR.BYTE = 0x70;         // ( RIE=1、TE=1、RE=1 )
      wait_ms( 10 );                // 10[ms] 待ち

      SCI0.SSR.BYTE;                // Dummy Read
      SCI0.SSR.BYTE = 0x80;         // Clear Error Flag
      init_ring_0();                // リングバッファ初期化
}
```

コンパイラのオプティマイザに
削除される危険性が 高い。
この行のコーディングは、SCI
周辺回路の都合で 1回内容を
読み出さないと、うまく初期化
されないものと 思われます。

シリアル通信SCI の　1byte送信処理

```
//*********************************
//**  SCI ch.0  1byte 送信       **
//*********************************
void send_sci_0( char dt )
{
PBDR.BIT.B7 = 1;              // 青LED 点灯
    while( SCI0.SSR.BIT.TDRE == 0 );     // 送信レディ待ち
      SCI0.TDR = dt;          // 1byte 送信
PBDR.BIT.B7 = 0;              // 青LED 消灯
}
```

ここも、コンパイラのオプティマイザに
改ざんされる 恐れがあります。
この行のコーディングは、1byte送信時に、前の
データをシフトレジスタに渡し　送信レジスタが、
空になった事を　確認する処理です。

　これらの　オプティマイザの改ざんによる誤動作を防ぐために前のページでは　SCI0.SSR.BYTE、このページでは　SCI0.SSR.BIT.TDRE　どちらも、SSR ですね。 SSRは　シリアル通信のステータスレジスタです。 これらに　volatile を　付ければいいんじゃないか。 と思われる方もいると思います。 しかし、SCI0.SSR には　volatile を　付けられないのです。 何故かというと、
ルネサス資料による　volatile修飾子の説明にて

　volatile修飾子をつけて変数宣言する と書いてあります。

しかし、SCI0.SSR.BYTE 及び SCI0.SSR.BIT.TDRE は、変数では　ありません。 iodefine.h 内で　#define で　宣言されている　構造体、共用体の オフセットアドレスを計算できる周辺回路レジスタの アドレス値。
これは、メモリ上に実態のない一種のマクロですね。 よって変数では無いので、volatile を付ける事は　出来ません。

では、どうするかというと　今のC言語ソースで、どうすれば 問題が解決出来るかは、今のところ思いつきません。

その前に　アドレス情報を持つマクロが、最適化されるのか どうかも 分かりません。

一応、念のため最適化で改ざんされる恐れのある部分を　局所的に　アセンブラのサブルーチンに 置き変える事にしました。

この処置を行った後に、シリアル通信の初期化と、1byte 送信のプログラムの動きを確認してみました。

残念ながら、結果は　1byte 送信してくれませんでした。

ガクッと　きましたね。

この事からして、1byte送信が 出来ない原因は他にある。 と 考えないと いけません。

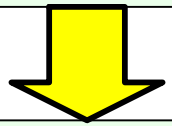
今までは、初期化処理に　問題があるのではないかと、ずっと考えていましたが、1byte送信処理も 確認してみました。 凡そ 20年前に作成した、正常に動作するアセンブラの関数を C言語に 移植した訳ですが、再度 見なおすと　1byte送信処理にて、C言語側で　1行抜けている部分を見つけました。

もしかして、と思って　その1行を追加しました。　そして、ビルドして　マイコンに書き込み動作確認を　行いました。

さて　どうなるか。？

1文字送信 （ 変更前 ）

```
void send_sci_0( char dt )
{
  PBDR.BIT.B7 = 1;          // 青LED 点灯
     while( SCI0.SSR.BIT.TDRE == 0 );
                            // 送信レディ待ち
     SCI0.TDR = dt;         // 1byte 送信
  PBDR.BIT.B7 = 0;          // 青LED 消灯
}
```



1文字送信 （ 変更後 ）

```
void send_sci_0( char dt )
{
  PBDR.BIT.B7 = 1;          // 青LED 点灯
     send_sci0_check();     // ★ 送信レデイ待ち

     SCI0.TDR = dt;         // 1byte 送信
     SCI0.SSR.BYTE &= ~0x80;  // 送信開始
  PBDR.BIT.B7 = 0;          // 青LED 消灯
}
```

左に SCI0の 1文字送信の send_sci_0 関数です。 内部を お見せします。 上が 変更前で、下が 変更後です。

下は、変更箇所が2ヶ所ありますが、上の赤枠で囲った部分は 変更前の while ループと その中を アセンブラの関数で 置き換えた物です。 この、while ループの置き換えは、効果が ありませんでした。

緑の枠で囲った部分が アセンブラからの移植で 抜けていた行です。 今回この行を追加したお陰で、正常に 文字及び 文字列を 送信できる事を確認しました。 あと シリアル通信では 受信処理も必要ですが、受信は 受信割り込みを用い 255byteの リングバッファを 間に挟む構成で作成しました。 受信の方は トラブル無く すんなり出来ました。 SCI0、SCI1 共に 連続した送受信に 成功しました。