大変 お待たせ いたしました。

お陰様で、通常は痛みも治まり、YouTube動画制作に 復帰出来ました。 しかし、まだ 顎を 大きく開けようとすると痛いんですよね。

食べる時と、あくびを不用意にやると、口を大きく開けようとして、アイタタタという状況です。
もう しばらくすると治るでしょう。
もう 二度と歯は 抜きたくないです。

余談はさておき、今回から 割り込み処理と アセンブラを 並行してやろうかと思います。

当初は、初心者に優しい？ と思われる C言語で、割り込み処理を 説明しようかと 思っておりました。 が、最終的に 割り込み処理は アセンブラで 作り直す事になると思うので、ちょっと技術的に難しい話が 出てくるかも 知れませんが アセンブラで 行う事にしました。

まずは、前回どこまで 話を していたのか確認するため、
最後の 1ページを 再度、説明します。

割込み信号が入って来て、割り込み処理が 実行されるまで

まずは、使用する周辺回路の初期化（ 今回は 16 bit タイマー ch.0 ）を 行っておく必要が ありま す。 そして CPUの割り込み許可フラグを 許可状態にする必要が あります。 因みに 電源 ON 直後は 割り込み禁止状態になっています。

そして、目的の割り込み信号が入ってくると、まず 現在実行中の命令が 終了したら、

① PC（ プログラムカウンタ ）の値と、CCR（ コンディションコード レジスタ)を スタックエリアに 積み上げます。

② CCRの I bit を 1 にして割り込みを 禁止します。

③ 割り込み要因に対応する割り込みベクタアドレスを 生成し その内容（ 割り込み処理の先頭アドレス)を PCに セットします。

因みに ①～③のシーケンスは ハード的に 実行されます。

④ 割り込み処理ルーチンを 実行します。

その前に 割り込み処理内で 使用するレジスタを 使用する前に スタックに 積み上げます。 そして、割り込み処理を 行います。

割り込み処理が 終わったら スタックに積み上げたレジスタのバックアップを 元のレジスタに戻します。

⑤ そして RTE命令で CCRと PCを復帰し 中断していたプログラムの実行を 再開します。

因みに C言語で 割り込み処理ルーチンを 作成する場合は、レジスタの 退避、復帰は 考える必要はありません。 ER0 ～ ER6 の全ての レジスタの退避、復帰を 行っていると 思います。

前ページにて、C言語で 割り込み処理ルーチンを 作成する場合は、レジスタの 退避、復帰は 考える必要は ありません。 ER0 ～ ER6 の全ての レジスタの 退避、復帰を 行っていると 思います。

と、書きましたが、基本的に C言語では 何番のレジスタを使用するとか 指定出来ませんし 割り込み処理内で どのレジスタを使用しているという確認も 難しい という事で、ER0 から ER6 の 全てのレジスタを スタックに 退避 復帰しているものと思います。

32bitのレジスタを 7本スタックに積み上げる訳ですが H8マイコンは データバスが 16bit幅なので ワード換算で 14個のデータを 割り込み処理の入口で スタックに積み上げます。

割り込み処理から戻る直前に 14個のワードデータを スタックから レジスタに 復帰するので レジスタの 退避 復帰処理で やや時間が かかります。 という事で 割り込み処理の内容によっては 応答速度が 遅い。
という事にも なりかねません。

速度の改善を行うには アセンブラで 割り込み処理を作成して、実際に使用するレジスタだけを 退避 復帰するようにすれば 改善できます。

次ページにて、レジスタの退避、復帰のイメージ部分だけを アセンブラにて示します。
その前に アセンブラで使用する命令語というか ごく一部の ニーモニックコードのコードの説明をします。 push命令と pop命令です。
push命令は レジスタ内の値を スタックエリアに積み上げます。 pop命令は スタックエリアに積み上げた値を レジスタに 戻します。

C言語で作成した割り込み処理ルーチン

```
Interrupt_proc:   ; 割込み処理エントリラベル
   ; レジスタ値を スタックへ退避
   push.l   er0   ; 実際は E0,R0の 2回Stackへ転送
   push.l   er1   ; 実際は E1,R1の 2回Stackへ転送
   push.l   er2   ; 実際は E2,R2の 2回Stackへ転送
   push.l   er3   ; 実際は E3,R3の 2回Stackへ転送
   push.l   er4   ; 実際は E4,R4の 2回Stackへ転送
   push.l   er5   ; 実際は E5,R5の 2回Stackへ転送
   push.l   er6   ; 実際は E6,R6の 2回Stackへ転送

   周辺回路アクセス等の 本来の割込み処理

   ; スタックからレジスタに 値を復帰
   pop.l    er6   ; 実際は R6,E6の 2回Stackから転送
   pop.l    er5   ; 実際は R5,E5の 2回Stackから転送
   pop.l    er4   ; 実際は R4,E4の 2回Stackから転送
   pop.l    er3   ; 実際は R3,E3の 2回Stackから転送
   pop.l    er2   ; 実際は R2,E2の 2回Stackから転送
   pop.l    er1   ; 実際は R1,E1の 2回Stackから転送
   pop.l    er0   ; 実際は R0,E0の 2回Stackから転送
   rte            ; 割込み処理からの リターン命令
```

H8は データバスが 16bit なので Wordレジスタ単位で スタックに ER0～ER6が 積み上がった状態。 14個

H8は データバスが 16bitなので Word レジスタを 14回スタックに積み上げる。

H8マイコンに限った話では無いのですが、最近の 組み込み用32bit CPUも 16本レジスタが 有るのが 当たり前になって

積み上がったWord レジスタの値を順に降ろして元のレジスタに戻す。

来ているので C言語で割り込み処理を実装すると 1秒間に 1万回以上の割り込み処理を 行う様な場合、問題が発生するかもしれません。

アセンブラで作成した割り込み処理ルーチン

```
Interrupt_proc:   ; 割込み処理エントリラベル
   ; レジスタ値を スタックへ退避
   push.l   er0   ; 実際は E0,R0の 2回Stackへ転送
   push.l   er1   ; 実際は E1,R1の 2回Stackへ転送

   周辺回路アクセス等の 本来の割込み処理

   ; スタックからレジスタに 値を復帰
   pop.l    er1   ; 実際は R1,E1の 2回Stackから転送
   pop.l    er0   ; 実際は R0,E0の 2回Stackから転送
   rte            ; 割込み処理からの リターン命令
```

H8は データバスが 16bitなので Word レジスタを 4回スタックに積み上げる。

積み上がったWord レジスタの値を順に降ろして元のレジスタに戻す。

H8は データバスが 16bit なので Wordレジスタ単位でスタックに ER0～ER1が積み上がった状態。 4個

前ページの C言語で作成した割り込み処理ルーチンに比べ、えらく短くなったな。 と 思われるかもしれません。

この アセンブラで作成した割り込み処理ルーチンでは、本来の割込み処理で使用されるレジスタは ER0と ER1の2本のレジスタがあれば足りるという判断で記述しました。

実際過去に、インターバルタイマー用、シリアル通信受信処理に使用したレジスタは ER0、ER1 の 2本が あれば用が 足りてます。 先ほど説明していませんでしたが、PUSH、POP命令の 右に .l が あります。これは Longの属性を 意味します。

ニーモニックコードの事をオペコードとも表現します。

push.l er0 この場合、push.l が オペコードで その右の er0 は オペランドと 表現します。

オペコードが 命令語で オペランドが 操作されるレジスタや メモリの値、あるいは即値の場合もあります。 オペランドは 命令によっては無い物もあります。 割り込みからの リターン命令 rte には オペランドは、根本的に ありません。

アセンブラの事に関しては 少しずつ 小出しにして行きます。

何故、割り込み＋アセンブラに したか

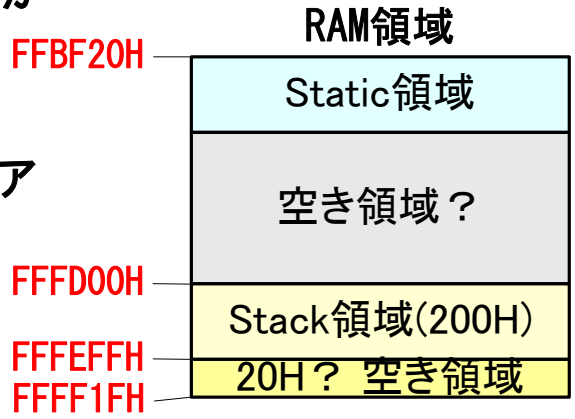
もう、前の 2ページで、私が 何故、割り込み＋アセンブラに したか 分かったと思いますが 必要最低限のレジスタのみ 退避、復帰する事で、割り込みの応答速度を 早くする事が出来る という事です。 H8のようなマイコンの場合 16bitで さほど早くないので、有効と思います。

それと、レジスタの退避、復帰に スタックという RAM領域に 積み上げましたが スタックが どういうメモリ領域か 分かりますか。？

これも C言語しかやった事のない方には、イメージしずらい物と思います。 スタックとは、データを一時的に保存したり、一時的な変数領域を確保したりするのに、使用します。

スタックを 構成するのは スタックエリアとなる RAM領域と そのスタックを 管理するスタックポインタで 構成されます。

H8の場合は ER7が スタックポインタに なります。 スタックエリアは 内蔵RAMエリア アドレス FFBF20H ～ FFFF1FHの 16Kbyte内に確保されます。RAM先頭がリンカで確保される 静的変数のエリアです。静的変数エリアのサイズは プログラムにより異なります。間違いなく言える事は Stackのサイズは 200H (512byte) です。



で、ER7のスタックポインタに設定される初期値は、基本 Stack領域の 最終番地となります。でスタックポインタは 1Word データを 積み上げる毎に -2 されます。 1Word データを降ろす場合は、+2 されます。

それと、スタックの使い方で、レジスタの退避復帰以外に　一時的な変数領域を確保したりするのに　使用します。と書きましたが、C言語の　Auto変数の領域が　もろ Stackエリアなのです。　C言語の関数が　呼び出された時は、Auto変数が必要な場合に そのメモリエリアをスタックに確保します。そして、その関数を終了してリターンする直前に　確保した Auto変数の領域を　破棄します。　あと、特に組み込みマイコンの場合　Stack全体のサイズが さほど大きくないので、Auto変数で　大きな配列を 作っては いけません。スタックオーバーフローで、システムが 破綻します。

余談ですが、パソコンは　少々の事では、スタックオーバーフローになる事は　無いようです。

というのが、組み込みマイコンでは　まず行いませんが、再帰呼び出しというプログラム手法があります。　自分の関数の中で　ある条件が成立した場合に　そこから　自分の関数を　また呼び出すという手法です。　下手するとスタックを壊しかねないプログラムになりますが、ソートプログラムで　2分岐のアルゴリズムとして使われます。
20年以上前に　Delphiで　作った事が あります。
クイックソート、ヒープソートで 使われます。
結構、高速にソート処理が 出来ます。
余談でした。

今回の　H8マイコンでは　まずタイマー割り込みを使用する事を　最初の目標にしてますので　タイマー周辺回路のブロック図を　見てみます。

H8/3069 16bitタイマー

ルネサスデータシートの概要を　紹介します。

概要:

本 LSI は、3 チャネルの 16 ビットカウンタにより構成される 16bit タイマを 内蔵しています。

特長

16 ビットタイマの特長を以下に示します。

■最大6 種類のパルス出力、または最大6 種類のパルス入力処理が可能

■各チャネル 2 本、合計 6 本のジェネラルレジスタ(G R)を持ち、各レジスタ独立に アウトプットコンペアマッチ／インプットキャプチャの機能設定が可能

■各チャネルとも　8 種類のカウンタ入力クロックを選択可能

内部クロック: φ、φ/2、φ/4、φ/8　は（ 25MHz、12.5MHz、6.25MHz、3.125MHz です。）

外部クロック: TCLKA、TCLKB、TCLKC、TCLKD

■各チャネルとも　次の動作モードを設定可能

・コンペアマッチによる波形出力:　0 出力／1 出力／トグル出力が選択可能(チャネル2 は0 出力／1 出力が可能)

・インプットキャプチャ機能:　立ち上がりエッジ／立ち下がりエッジ／両エッジ検出が選択可能

・カウンタクリア機能:　コンペアマッチ／インプットキャプチャによる カウンタクリアが 可能

・同期動作:　複数のタイマカウンタ(16TCNT)への同時書き込みが可能。 コンペアマッチ／インプットキャプチャによる同時クリアが可能。 カウンタの同期動作による　各レジスタの同期入出力が可能

・PWM モード:　任意デューティの　PWM 出力が可能 同期動作と組み合わせることにより、最大 3 相の PWM 出力が 可能。

■チャネル2 は　位相計数モードを設定可能
　2 相エンコーダの　カウント数の自動計測が可能

■内部16 ビットバスによる高速アクセス
　16TCNT、GR の 16 ビットレジスタに対して、16 bit バスによる 高速アクセスが 可能

■タイマ出力初期値を　任意に 設定可能

■9 種類の　割り込み要因
　各チャネルとも　コンペアマッチ／インプットキャプチャ兼用　割り込み×2 要因、オーバフロー割り込み×1 要因があり、それぞれ独立に要求可能

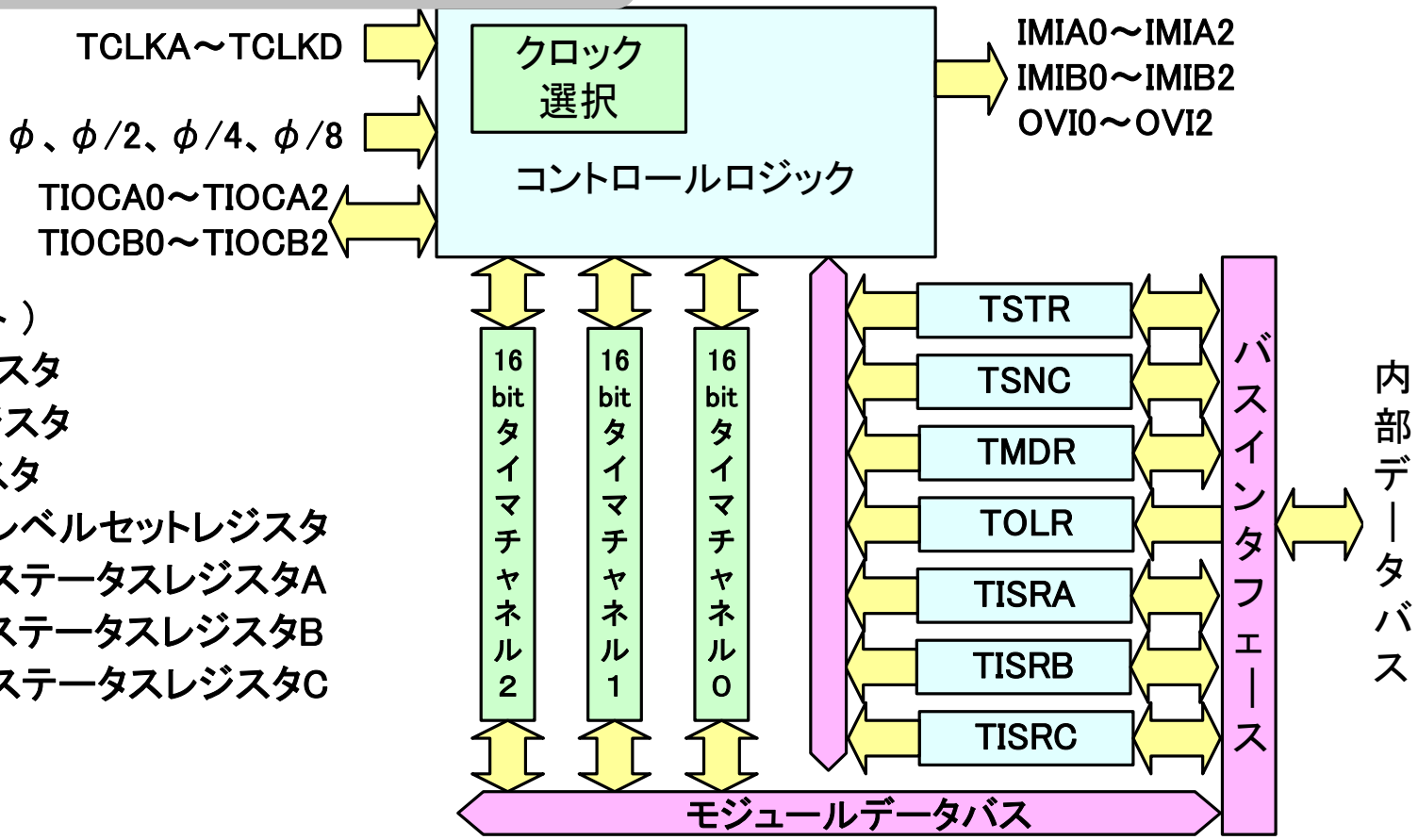
■プログラマブル パターンコントローラ（TPC）の
　出力トリガが 生成可能。 チャネル 0～2 の コンペアマッチ／インプットキャプチャ信号を TPC の 出力トリガとして 使用可能

以上　概要として書いてありましたが、訳 分からない用語が　多々出てきて何のこっちゃ という感じだったと思います。 これらの専門用語は、全て理解する必要はありません。 さしあたり、自分が使うタイマー機能に関係する部分だけ理解出来れば OKです。

多機能なのが有りがたく思えるのは、三相PWMとか複雑な出力を行う時です。 複雑な出力を行う時はかゆい所に手が届く的な 便利さが あります。

次ページに　16bitタイマーのブロック図（全体図）をお見せします。

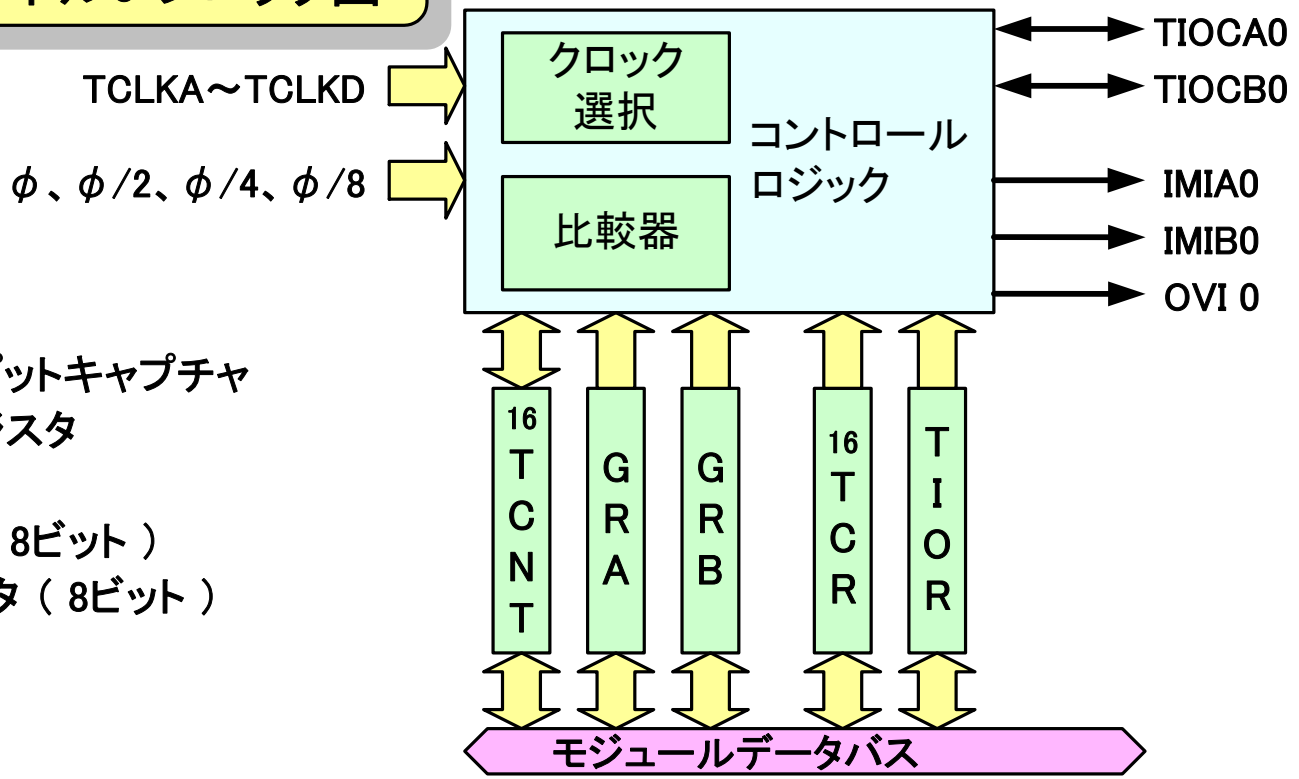
H8/3069F　16 ビットタイマのブロック図（全体図）



【記号説明】（ 全て 8ビット ）

TSTR:　タイマスタートレジスタ
TSNC:　タイマシンクロレジスタ
TMDR:　タイマモードレジスタ
TOLR:　タイマアウトプットレベルセットレジスタ
TISRA:　タイマインタラプトステータスレジスタA
TISRB:　タイマインタラプトステータスレジスタB
TISRC:　タイマインタラプトステータスレジスタC

H8/3069F 16 ビットタイマのチャネル0 ブロック図



【記号説明】

16TCNT0： タイマカウンタ（ 16ビット）

GRA0、 ジェネラルレジスタA、B（インプットキャプチャ

GRB0： ／アウトプットコンペア兼用レジスタ

（ 16ビット × 2 ））

16TCR0： タイマコントロールレジスタ（ 8ビット ）

TIOR0： タイマI/Oコントロールレジスタ（ 8ビット ）

実際に　インターバルタイマー処理に使用するレジスタ

16bit タイマー全体のブロック図と、チャネル 0 の　ブロック図を紹介しましたが、多数の　設定用、状態確認用のレジスタが ありました。

ここでは、単機能のインターバルタイマー実現に 的を絞り　最低限の設定で動かす事を 実現します。 4または 5種類のレジスタを用いれば実現出来ます。 では、どのレジスタを 使うかを紹介します。

① 16TCR0:　タイマコントロールレジスタ (8bit)
② GRA0H:　ジェネラルレジスタA (16bit)
③ TISRA:　タイマ割込み、設定 確認 (8bit)
④ TSTR:　タイマスタートレジスタ (8bit)
⑤ TCNT0:　タイマ 0 カウンタ (16bit)

順次、上記 レジスタの 各ビットの意味を説明します。

① 16TCR0:　タイマコントロールレジスタ (8bit)

16TCRは 8 ビットのレジスタです。16 ビットタイマには、各チャネル 1本、計 3 本の 16TCR が あります。因みに初期値は b7以外は　All 0 です。

b7	b6	b5	b4	b3	b2	b1	b0
－	CCLR1	CCLR0	CKEG1	CKEG0	TPSC2	TPSC1	TPSC0
リザーブ	カウンタ クリア b6、b5 カウンタ クリア要因を選択するビットです。		クロックエッジ b4、b3 クロックの検出エッジを選択するビットです。		タイマ プリスケーラ b2～b0 16TCNTの カウントクロックを選択するビットです。		

タイマ プリスケーラ設定

TPSC2	TPSC1	TPSC0	設定内容
0	0	0	内部clock: φ 25MHz(初期値)
0	0	1	内部clock: φ/2　12.5MHz
0	1	0	内部clock: φ/4　6.25MHz
0	1	1	内部clock: φ/8　3.125MHz

タイマ プリスケーラ設定　続き

TPSC2	TPSC1	TPSC0	設定内容
1	0	0	外部クロックA：TCLKA 端子入力
1	0	1	外部クロックB：TCLKB 端子入力
1	1	0	外部クロックC：TCLKC 端子入力
1	1	1	外部クロックD：TCLKD 端子入力

クロック検出エッジを選択

CKEG1	CKEG0	設定内容
0	0	立ち上がりエッジでカウント（初期値）
0	1	立ち下がりエッジでカウント
1	0	立ち上がり/立ち下がりエッジの 両エッジでカウント
1	1	

16TCNT のカウンタクリア要因を選択します。

CKEG1	CKEG0	設定内容
0	0	16TCNT のクリア禁止　（初期値）
0	1	GRA のコンペアマッチで　クリア
1	0	GRB のコンペアマッチで　クリア
1	1	他のタイマのカウンタクリアに 同期

② GRA0H：　ジェネラルレジスタA（16bit ）

16bitタイマー　1チャネルに付き、16bitの　ジェネラルレジスタは、GRAと　GRBの　2本ありますが今回は、GRAしか使いません。 GRA後ろの 0 はチャネル 0 の意味です。 その後に Hが　付いてますが、これは、16bit GRAの 上位バイトを　アクセスする用途で用意されたと思います。 H8はビッグエンディアンなので、GRA0Hは　GRAを　ワード アクセスする際の　先頭アドレスにも なります。 そのすぐ後ろのアドレスに　GRA0Lも ありますが、GRA0Lは　Byteアクセスしか出来ません。　今回は　ワードで　アクセスします。

で、ジェネラルレジスタとは　何なのかというと、用途により色々な機能を持たせる事が出来る16bitの 多目的レジスタといえます。

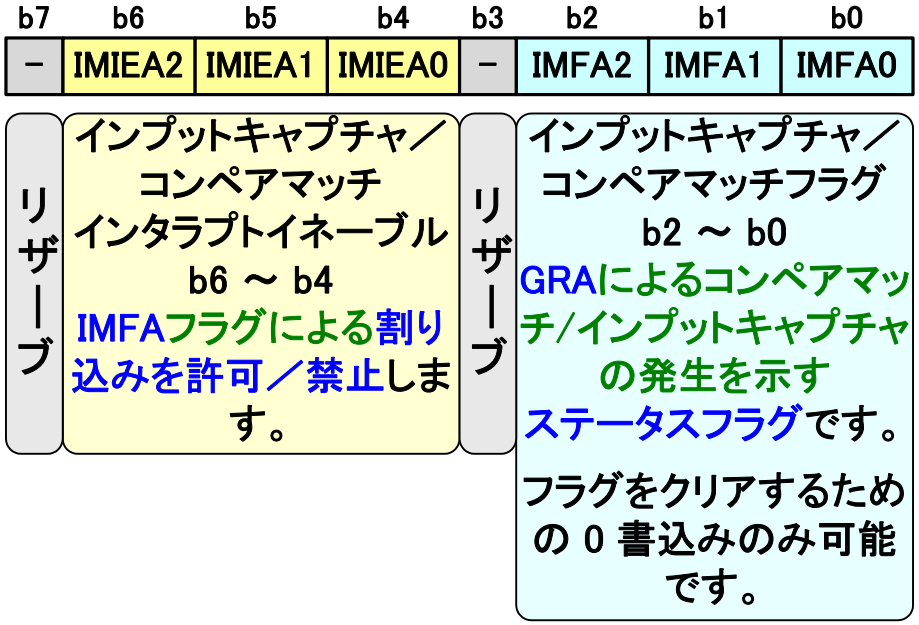
で、今回は 1ms周期の インターバルタイマーとして使用するので、φ/8の 3.125MHz を入力して更に分周する事により 1KHzの周期を作り出します。 そのためには 分周値が必要なので φ/8 を 1KHzで 割ります。
分周値＝ 3,125,000 / 1,000 で 3,125 と なります。

この 3,125 を GRA0に初期値として 書き込みます。で、TCNT0 と GRA0 とで、コンペアマッチ（ TCNT0 と GRA0 が イコールになる事 ）で TCNT0を ゼロクリアして 割り込み信号を 出すようにします。 16bit カウンタ TCNT0は 初期値 0 で、カウント開始すると 入力クロックに従いインクリメント動作を 行います。そして TCNT0 = GRA0 (3,125)になった瞬間、TCNT0 = 0 になり タイマー 0 から割り込み信号が 出ます。

③ TISRA: タイマインタラプトステータスレジスタA (8bit) は 8 ビットのリード／ライト可能なレジスタで、GRAの コンペアマッチ／インプットキャプチャの発生を示し、GRAの コンペアマッチ／インプ ットキャプチャ割り込み要求の許可／禁止を制御します。

b7	b6	b5	b4	b3	b2	b1	b0
－	IMIEA2	IMIEA1	IMIEA0	－	IMFA2	IMFA1	IMFA0

リザーブ	インプットキャプチャ／コンペアマッチインタラプトイネーブル b6 ～ b4 IMFAフラグによる割り込みを許可／禁止します。	リザーブ	インプットキャプチャ／コンペアマッチフラグ b2 ～ b0 GRAによるコンペアマッチ/インプットキャプチャの発生を示すステータスフラグです。 フラグをクリアするための 0 書込みのみ可能です。



左のレジスタの図を見ると、割込み許可/禁止のフラグ b6 ～ b4と、インプットキャプチャ/コンペアマッチ フラグ b2 ～ b0 の　各 3個ありますが、これは　16bitタイマーの チャネル番号と 対応しているので、今回は チャネル0 16bitタイマーの GRAを　使用しているので　フラグ名の最後の 2 文字が　A0 の物だけに 着目すればいいです。

　よって　今回使用するのは　b4の IMIEA0 と　b0 の　IMFA0　2つだけです。

④　TSTR:　タイマスタートレジスタは　8 ビットのリード／ライト可能なレジスタで、チャネル 0 ～ 2 の　16TCNT（ 16bit カウンタ ）の動作／停止を選択します。

TSTR:　タイマスタート　レジスタ

b7	b6	b5	b4	b3	b2	b1	b0
–	–	–	–	–	STR2	STR1	STR0

カウンタスタート 2～0
16bit TCNT2 ～
16 bit TCNT0
の動作／停止を
選択するビットです。

1 = カウント動作
0 = 停止

今回は　b0 の　STR0 しか使用しません。
カウンタ 0 を　スタートさせる時 1 に　します。
カウンタ 0 を　ストップさせる時 0 に　します。

⑤　TCNT0:　タイマ 0 カウンタ (16bit)
これは、2ページ前でも説明しましたが、GRAと組みにして使用する　16bit の カウンタ です。

これは、電源 ON 時は　0 に　初期化されています。　一旦、インターバルタイマーを起動したら電源 OFF まで　動かし続ける場合は　TCNT は何もアクセスする必要は　ありません。

一つ　気にしていたのは　一旦タイマーを起動した後に　STR0 = 0 にして、タイマーを　停止させた場合に、TCNT0 カウンタ内に 中途半端な カウント値が　残っていると思われます。

この状態で　STR0 = 1 で　カウンターを　走らせると　最初の1回目だけ　1msに 満たない中途半端な　タイムインターバルで　割り込みが　かかる恐れが　あります。

よって、STR0 = 1 に する直前で　TCNT0 に　0 を　書き込んでから、STR0 = 1 で　TCNT0 を RUNさせようと 思います。

今回の メイン関数を含むソース 1/3

atest_1.src

```
    .include "H8_define.inc"    ; Assemble Comon File

; **  メイン処理
; -------------------------
        .export  _main
_main:
        mov.b   #H'E0, r0l              ; b7:青LED、b6:黄LED、b5:赤LED 出力に設定
        mov.b   r0l, @h8_PBddr          ; PortB  入出力設定

        bsr     _init_1ms_timer         ; タイマー処理初期化 呼び出し
        bsr     _enable_interrupt       ; 割り込み処理 許可  呼び出し
loop:                                   ; 空ループ
        bra     loop
        rts                             ; リターン ツゥ サブルーチ ン
```

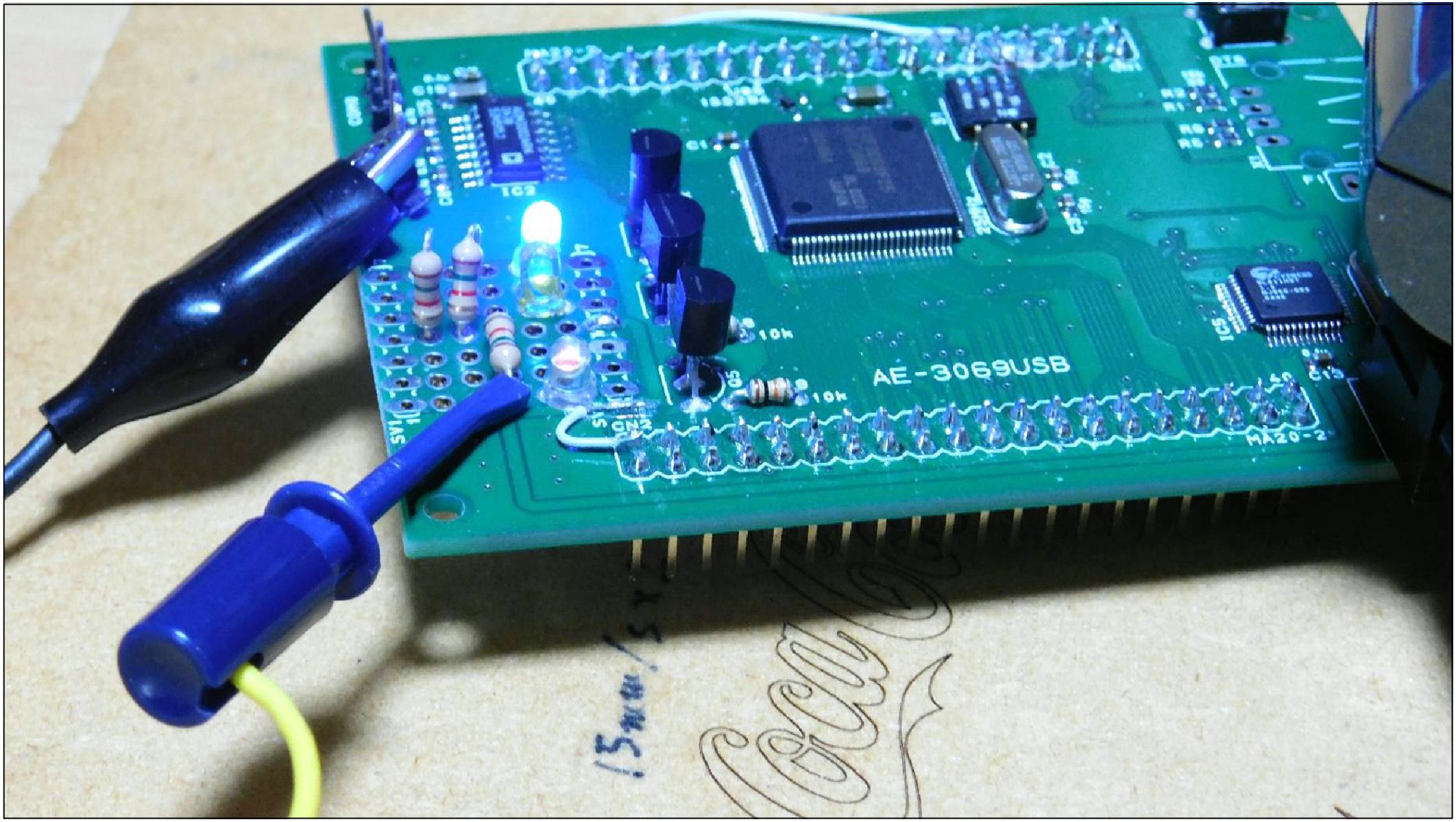
今回の メイン関数を含むソース 2/3

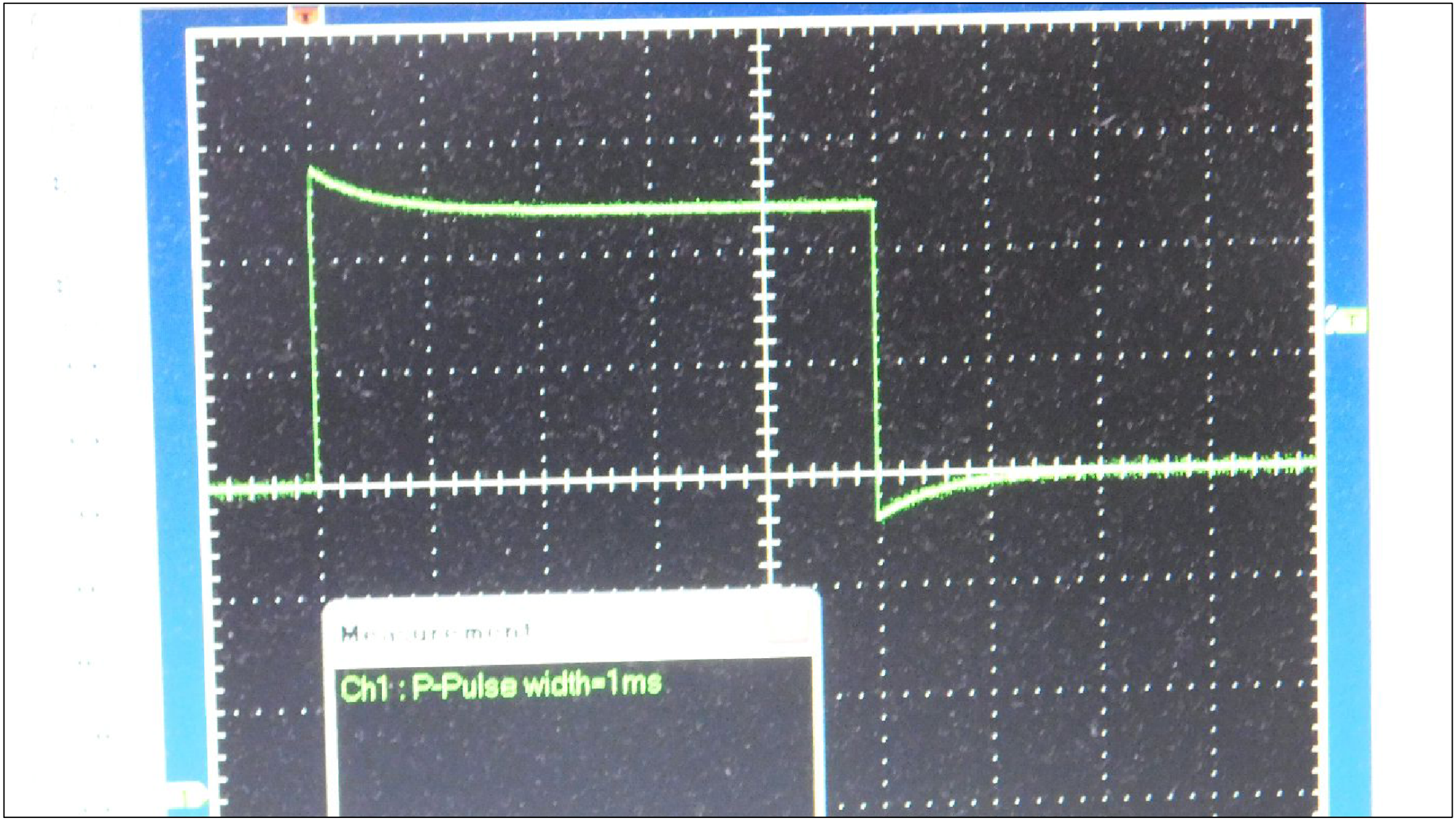
atest_1.src

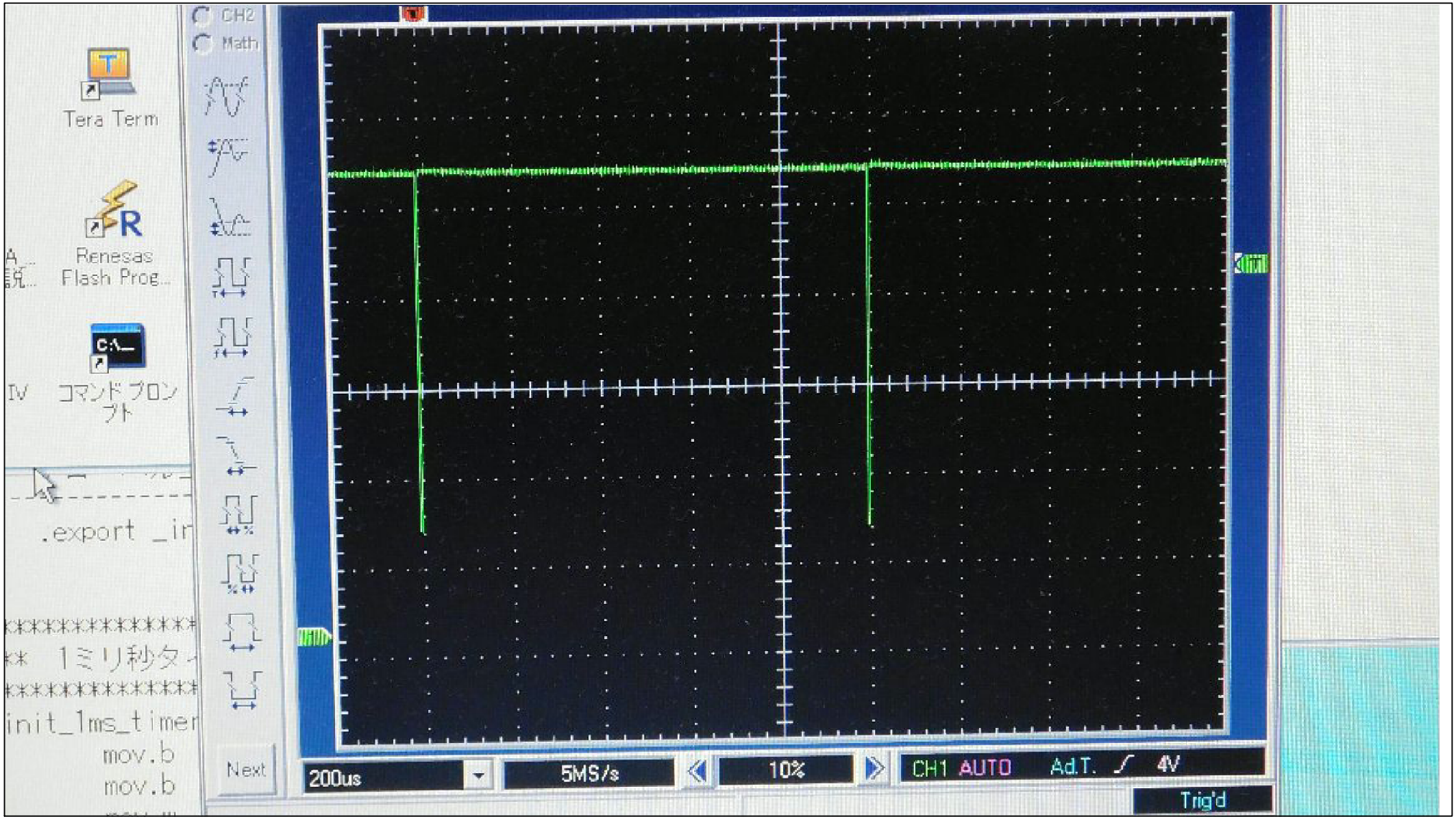
```
; **  全割り込みの 許可
; --------------------------
_enable_interrupt:
        _sti                ; 割り込み許可マクロ
        rts                 ; リターン ツゥ サブルーチ ン

;**  1ミリ秒タイマー初期化処理
;-----------------------------------
init_1ms_timer:
        bclr    #0, @h8_t16_tstr      ; TSTR = 0  カウンタを 一旦停止させる
        mov.w   #0, r0
        mov.w   r0, @h8_t16_cnt0h     ; TCNT0 = 0  カウンタを ゼロクリア
        mov.b   #H'23, r0l            ; tcr_0 = 0010 0011
        mov.b   r0l, @h8_t16_tcr0     ; GRAコンペアマッチ、P_Edge、φ/8
        mov.w   #3125, r0
        mov.w   r0, @h8_t16_gra0h     ; GRA0 = 3125   1ms 分周値設定
        bset    #4, @h8_t16_tisra     ; IMIEA0 = 1  GRA0によるコンペアマッチで割り込みを出す
        bset    #0, @h8_t16_tstr      ; TSTR = 1  カウンタを 走らせる
        rts                           ; リターン ツゥ サブルーチ ン
```

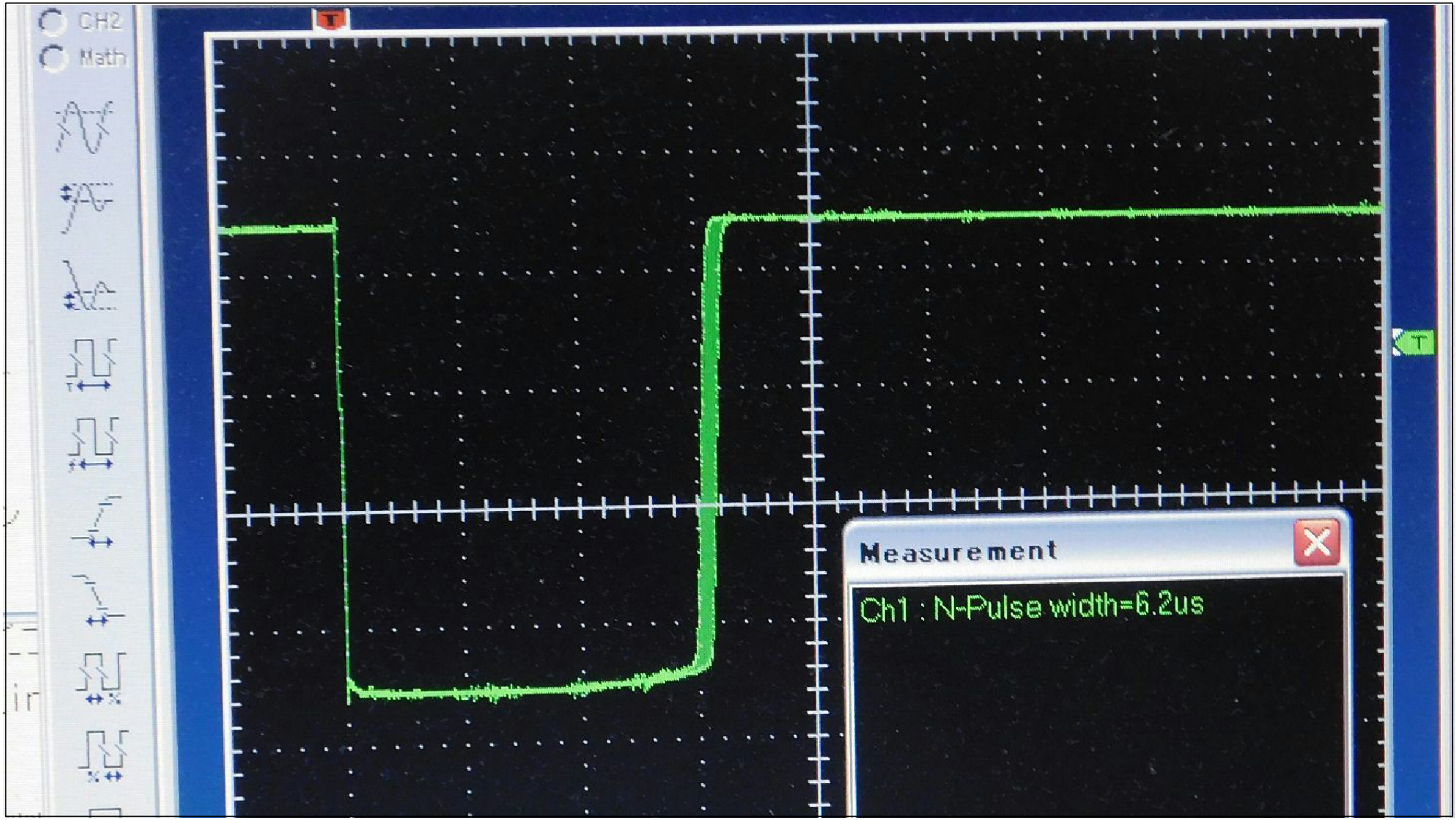
今回の メイン関数を含むソース 3/3

```
;>>  16bitタイマ0 GRA 割込み処理     <<
;*************************************
     .global    _INT_IMI_A0       ; ★  名前を デフォルトから ちょっと 変えている
_INT_IMI_A0:                    ; ラベル ( エントリアドレス )
           bset #5, @h8_PBdr          ; PB.b7(赤LED)に 1 を 入れる (赤点灯)
           push.l     er0                        ; ER0,ER1 を スタックへ退避
           push.l     er1

;;         btst #0, @h8_t16_tisra     ; IMFA0 = 1 確認
;;         beq        p002                ; IMFA0 が 1ではない時 p002へ飛ぶ

           bclr #0, @h8_t16_tisra     ; IMFA0 = 0 GRAフラグクリア
           btst #7, @h8_PBdr          ; PB.b7(青LED)の 確認
           beq        p001
           bclr #7, @h8_PBdr          ; PB.b7(青LED)に 0 を入れる      (青消灯)
           bra        p002
p001:      bset #7, @h8_PBdr          ; PB.b7(青LED)に 1 を入れる      (青点灯)

p002:      pop.l er1                          ; ER1,ER0 を スタックから復帰
           pop.l er0
           bclr #5, @h8_PBdr          ; PB.b5(赤LED)に 0 を入れる       (赤消灯)
           rte
```


AE-3069USB
10k
10k
HA20-2


Ch1 : P-Pulse width=1ms


CH2
Math
Tera Term
Renesas Flash Prog...
コマンド プロンプト
.export _in
1ミリ秒タ
init_1ms_timer
mov.b
mov.b
Next
200us
5MS/s
10%
CH1 AUTO
Ad.T.
4V
Trig'd


CH2
Math
Measurement
Ch1 : N-Pulse width=6.2us


Measurement
Ch1 : N-Pulse width=10.2us