# 割り込み／ポーリング／DMA/DTC（RL78 での周辺機能制御）

周辺機能を介してデータ転送を制御する方法には，大きく分けて 3 つの方法があります。その中で DMA や DTC は CPU を介することなく，高速にデータを転送することができますが，使用できるチャネル数が限られます。そのため，たとえば，CSI のスレーブでの高速通信のように限られた時間内に転送が必要な場合に使用できます。ただ，IIC のように，受信したデータで処理が異なる（たとえば，スレーブアドレスの LSB で送信／受信が切り替わるような）通信方式では使えません。

割り込みやポーリングは DMA/DTC に比べると高速性は劣りますが，CPU で処理するために，転送されたデータに応じて格納する先を変えたり，処理を変えたりするような使い方が可能です。通常，割り込みを使用しておけば，アプリケーションによってはかなりの高速通信にも対応できます。

## １．RL78 での割り込み処理（割り込み要求）

RL78 の割り込みは，周辺機能が動作完了したときに，CPU の処理または，DMA/DTC 転送を要求するためのトリガとなります。マニュアルでは INTxx で表されています。これが，全ての始まりです。これは，内部の割り込み要求元の段階でのハードウェアの信号名であり，ソフトウェアでは割り込みベクタを指定するときに使用するだけです。

```
#pragma interrupt r_tau0_channel1_interrupt(vect=INTTM01,bank=RB3,enable=true)
#pragma interrupt r_adc_interrupt(vect=INTAD)
```

その後，CPU が割り込み処理やソフトウェアで使用する割り込み要求は TMIF01 や ADIF のような割り込み要求フラグになります。ここらの関割り込み要求信号部の概要を図 1 に示します。周辺機能からの INTxx の立ち上がりエッジで割り込み要求フラグの F/F がセットされ，割り込み要求フラグ（xxIF）となります。この割り込み要求 F/F は CPU がベクタ割り込みを受け付けた時にハードウェアが自動的にクリアする以外に，命令でセットしたり，クリアしたり，読み出したりすることができます。これとは別に，xxMK 信号の反転信号で論理積が取られて CPU（割り込み制御部）に行きます。

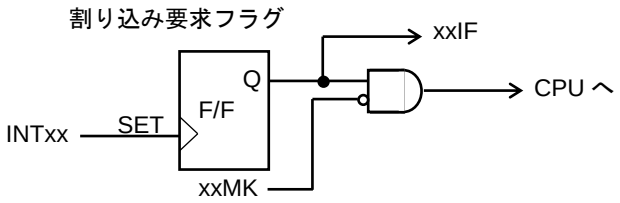


図１．　割り込み要求信号部

２．RL78 での割り込み処理（割り込み受け付け）

マスクが解除された（xxMK ビットが 0 の）割り込み要求信号は 2 つの用途で使用されます。

一つ目は，CPU のスタンバイ状態の解除です。この動作は，割り込み優先順位とは全く無関係で，マスクされていない（xxMK=0 の）割り込み要求信号が発生していると実行されます。

また，このような割り込み要求が発生している状態でスタンバイに入る命令を実行すると，一旦スタンバイに入いり，スタンバイ解除の安定時間を待ってから解除されるようです。

二つ目は，ベクタ割り込み処理です。通常，割り込みと言うと，このベクタ割り込みのことを指します。ベクタ割り込みは，以下の条件が全て満足されたときに受け付けられます。

① マスクされていない割り込み要求が発生している。
② 同時に発生している割り込み要求の中で一番優先順位が高い
③ 処理中の割り込みよりも優先順位が高い
④ CPU がベクタ割り込み受け付け可能状態
⑤ 割り込み保留命令の実行中でない

ここで，①～③が割り込みの優先度に関する条件で，④が割り込み禁止／許可の条件です。⑤については，殆ど意識されることがない条件ですが，特定のタイミングだけで割り込みを受け付けさせる目的で EI 命令と DI 命令を連続させた場合に表面化する条件です。割り込み保留命令はその命令の次の命令の実行終了まで割り込みを受け付けません。EI 命令は割り込み保留命令なので，次の命令（ここでは DI 命令）の実行終了まで割り込みは受け付けません。DI 命令も割り込み保留命令ですが，そもそも，割り込みを禁止する命令なので，EI 命令とＤＩ命令が並んでいる場合には割り込みは受け付けられないことになります。それでは，どうすればいいかと言うと，単純に２つの命令に間に NOP 命令を入れればいいだけです。これで，有効な割り込み要求があれば，NOP 命令の実行後に割り込みを受け付けます。

スタンバイ状態での割り込み受け付けは，スタンバイが解除されてからベクタ割り込みが受け付けられます。

ベクタ割り込みが受け付けられると，要求された割り込みに対応したアドレスからベクタ（処理アドレスの下位 16 ビット）を読み出して，そのアドレス（アドレス 20 ビット中の上位 4 ビットは 0）に分岐します。

そのとき，実行していたプログラムのアドレスと PSW レジスタの値がスタックにセーブされます。その後，PSW は割り込み禁止状態になり，受け付けた割り込みの優先順位に対応したインサービス・プライオリティ・フラグに変更され，割り込み要求フラグがクリアされます。

このように割り込み処理がスタートした段階では，CPU は割り込み禁止状態になります。

割り込みは CPU の処理とは非同期に発生します。割り込み処理が通常処理の実行を妨害しないように，割り込み処理の頭では，使用する汎用レジスタをスタック領域にセーブします。こ

のためにレジスタの分だけ PUSH 命令を使用します。この PUSH 命令の実行時間がもったいないといった場合には，RL78 ではレジスタバンクが 4 バンク準備されていて，1 命令で使用する汎用レジスタを切り替えることができます。

また，割り込み処理が完了したら，PUSH 命令でセーブしていたデータを POP 命令で元に戻して，RETI 命令で割り込み処理から復帰します。このような手順をとることで，割り込み処理は割り込み前の CPU の状態に戻どします。

３．多重割り込み処理

アプリケーションによっては，複数の割り込みを使用し，その割り込みに優先度を設定して，優先度の低い割り込み処理中にも，より優先度の高い（緊急性の高い）割り込みを受け付けさせる必要が考えられます。

このための機能が多重割り込みです。RL78 では，4 レベルの割り込み優先度（優先順位）がサポートされていて，最大 4 レベルの割り込みのネスティング（多重化）が可能です。

多重割り込みの使い方は次の 2 つだけです。

①割り込み優先度の設定

②割り込み処理中での割り込み許可（EI 命令の実行）

以下の説明は参考程度として構いません。

多重割り込みを使用するには，使用する割り込みに優先度を設定します。RL78 では，各割り込みに対して，2 ビットの優先順位指定フラグ・レジスタが準備されていて，以下の 4 つのどれかに設定します。ディフォルトでは，全ての割り込みは最低優先に設定されています。

00 ： レベル 0（最優先）
01 ： レベル 1（第 2 優先）
10 ： レベル 2（第 3 優先）
11 ： レベル３（最低優先）

起動後に，割り込みを許可すると，CPU は PSW レジスタのインサービス・プライオリティ・フラグ（ISP1 と ISP0）が 11 となっており，全てのレベルの割り込みが受け付け可能になります。この状態では，早い者勝ちで受け付けられます。ここまでは，通常の割り込み処理と同じですが，多重割り込みを使用する場合には，最優先の割り込み処理以外の割り込み処理では，EI 命令（EI();）を実行して，CPU を割り込み許可状態にする必要があります。

例えば，レベル 3 の割り込みが受け付けられたとします。このとき，PSW レジスタのインサービス・プライオリティ・フラグ（ISP1 と ISP0）は 10 となります。これによりレベル 3 の割

り込みは受け付けられず，レベル 2 以上の割り込みが受け付け可能になります。これらの処理はハードウェアで実行されるので，こうなることを**知っておくだけで十分**です。

この状態で割り込みを許可して，レベル 1 の割り込みが受け付けられると，ISP1 と ISP0 は 00 となります。この状態では，割り込みを許可してもレベル 0 割り込みだけが受け付け可能になるだけで，レベル 1～3 の割り込みは受け付け禁止です。

このレベル 1 の割り込み処理を終了して，RETI 命令を実行すると，PSW は以前の値に戻るので，ISP1 と ISP0 は 10 となり，レベル 2 以上の割り込みが受け付け可能になります。

このように，割り込み優先度の設定と割り込み処理中での割り込み許可（EI 命令の実行）だけで 4 レベルの多重割り込みの制御が可能です。CPU 内部のハードウェアの動作は面倒ですが，使う方はそれらを意識しなくても多重割り込みは簡単に使うことが可能です。

あえて，注意事項を上げるとすると，割り込み優先度の設定は割り込みの関係を十分に考慮して行ってください。また，優先度を動的に変化させるのはトラブルの原因になるので，お勧めできません。

## 4. RL78 でのベクタ割り込み制御

RL78 の CPU での割り込み制御は PSW レジスタに集約されると言っても過言ではありません。以下に PSW の構成を示しますが，朱書きしたビットが割り込みに関係したビットです。

| bit7 | bit6 | bit5 | bit4 | bit3 | bit2 | bit1 | bit0 |
|---|---|---|---|---|---|---|---|
| IE | Z | RBS1 | AC | RBS0 | ISP1 | ISP0 | CY |

IE はベクタ割り込みの許可／禁止を指定するビットです。

RBS1 と RBS0 は使用する汎用レジスタのバンクを指定するビットです。

ISP1 と ISP0 は受け付け可能な割り込み優先度を指定するビットです。

このように，半分以上が割り込みの制御用です。この PSW レジスタを割り込み受け付け時にハードウェアが自動的にスタックにセーブし，RETI での割り込みからの復帰時に元に戻すことで，殆ど意識することなく，割り込みをスムースに処理できるようになっています。

なお，これらのことは知らなくても割り込みを使いこなすことは十分可能です。

## 5. RL78 でのベクタ割り込み応用

RL78 の割り込み要求フラグはハードウェアでセットされることを前提にして話してきましたが，ソフトウェアでトリガすることも可能です。

この方法を使う最もいい例が，割り込みによるデータ送信です。

CSI や UART による送信を行う場合に，割り込みは送信完了か送信バッファが空になったときに発生します。つまり，最初のデータに対しては割り込みが発生しないので，2 番目以降のデータと異なる処理を行わないといけないことになります。

この処理を真面目に実行しているのがコード生成の R_UART0_Send 関数です。その該当する部分の抜粋を以下に示します。

```
gp_uart0_tx_address = tx_buf;
g_uart0_tx_count = tx_num;
STMK0 = 1U;    /* disable INTST0 interrupt */
TXD0 = *gp_uart0_tx_address;
gp_uart0_tx_address++;
g_uart0_tx_count--;
STMK0 = 0U;    /* enable INTST0 interrupt */
```

1,2 行目で送信データの格納されたバッファのアドレスとデータ数を作業用の変数にコピーした後で割り込みをマスクして処理を行い，割り込みマスクを解除しています。

これは，UART0 が送信完了割り込みでなく，バッファ空き割り込みだった場合に排他制御を行うための処理です。

これは，制御方法を工夫すれば，不要な処理となります。見直したものを以下に示します。

```
gp_uart0_tx_address = tx_buf;
g_uart0_tx_count = tx_num;
STIF0 = 1U;    /* set INTST0 interrupt request flag */
```

この方法は，最初の送信データの書き込みから割り込み処理で行わせるものです。ポインタやカウンタの更新も全て割り込み処理で既に行っているものを流用したものです。

このように，割り込み要求フラグを利用すると，処理がすっきりします。

なお，この方法は DMA には使えません。DMA の場合には DMA 転送開始ソフトウェアトリガを使用することになります。

## 6. RL78 での割り込みのポーリング制御

RL78 の割り込み要求は，これまで説明してきた様に，DMA/DTC のトリガ，スタンバイの解除及びベクタ割り込みで使用する以外に，ポーリングすることが可能です。

初心者には，割り込みに対して拒否反応を示す人が見受けられます。これは，割り込みが非同期の事象であり，プログラムの流れが途切れることへの恐れがあるのかもしれません。

そこで，割り込みを使用しない制御として挙げられるのが，割り込み要求フラグ（xxIF）のポーリングです。

割り込み要求が発生すると，割り込み要求フラグがセットされるので，それをソフトウェアで確認するのが割り込みのポーリング制御です。

割り込み要求フラグは原因となる事象が発生（A/D 変換が完了した，通信が完了した等）することでセットされます。

割り込み要求フラグは単なる 1 ビットのフラグで，割り込み要求があるか無いかだけを示すことしかできません。いつ，どのようにして割り込み要求フラグをクリアするかが問題です。ベクタ割り込みでは，ハードウェアでセットされ，ハードウェアでクリアされるので，気にする必要はありませんが，ポーリングでは自動的にはクリアされないので，注意が必要です。

一番簡単なのは，割り込み要求フラグがセットされたことを確認したらクリアすることです。しかし，これだけでは不十分で，フラグをポーリングする前にクリアしておくことが必要な場合があることです。

簡単な例を以下に示します。

```
while ( SRIF0 == 0 )
{
        NOP();
}
SRIF0 = 0;
```

ここで，while ループ中に NOP();を入れてあるのは，デバッグを意識したものです。

これだけでうまくいきそうですが，注意する必要があることが一つあります。それは，INTSR0 割り込みをベクタ割り込み禁止にしておく必要があるということです。SRIF0 ビットはベクタ割り込みが受け付けられるとクリアされます。割り込みを使わないからと，割り込み処理部に何も処理を記述してなかったとしても，割り込みの受付と RETI は処理する可能性があります。そうすると，割り込み受け付けで SRIF0 フラグがクリアされてしまい，上記のプログラムでは SRIF0 がいつまでたってもセットされないとか，while ループから抜けたり抜けなかったりすると言った不安定な動作が発生することがあります。

そのため，上記ポーリングを行う前に INTSR0 割り込みをマスクしたり DI にしてベクタ割り込みを禁止したりすることが必要です。

```
SRMK0 = 1;
while ( SRIF0 == 0 )
{
        NOP();
}
SRIF0 = 0;
```

また，割り込みそのものは許可しておき，割り込み処理でグローバル変数に準備したフラグをセットし，そのフラグをポーリングする方法もあります。