Nios® II - Vectored Interrupt Controller の実装

Ver.14

2019 年 9 月 Rev.2

Nios® II - Vectored Interrupt Controller の実装

目次

1. はじめに

この資料は、Vectored Interrupt Controller Core（以下 VIC）を使ったハードウェアおよびソフトウェアを実装する方法を紹介しています。

通常、 Nios® II では割り込みの制御に例外レジスタ方式が採用されており、プライオリティー・エンコーダなどをソフトウェアで処理しているため、自由度の高い割り込み制御が可能になる半面、割り込み発生から割り込みサービス・ルーチンの呼び出しまでのレスポンスについて若干遅くなる傾向がありました。

しかし、 VIC を使うことによって、より早い応答時間の割り込み制御が実現できるようになりました。また、それ以外にも VIC をデイジー・チェーンで結合することで 32 を超えた割り込み制御が可能になり、プライオリティーの変更やノンマスカブル割り込みの実現など、多くの機能が強化されております。

なお、VIC を使用する場合、Nios® II Software Build Tool（以下 Nios® II SBT）を使用する必要があり、また、割り込み関連のソフトウェア記述に若干の修正が必要になりますのでご注意ください。

2. 適用条件

本資料では、以下のソフトウェア・バージョンにて動作確認を行っております。また、本資料では Nios® II SBT の基本的な操作方法やソフトウェアの一般的な記述方法については解説しておりませんので、それらは別途マニュアル等を参照ください。

■ 対応バージョン

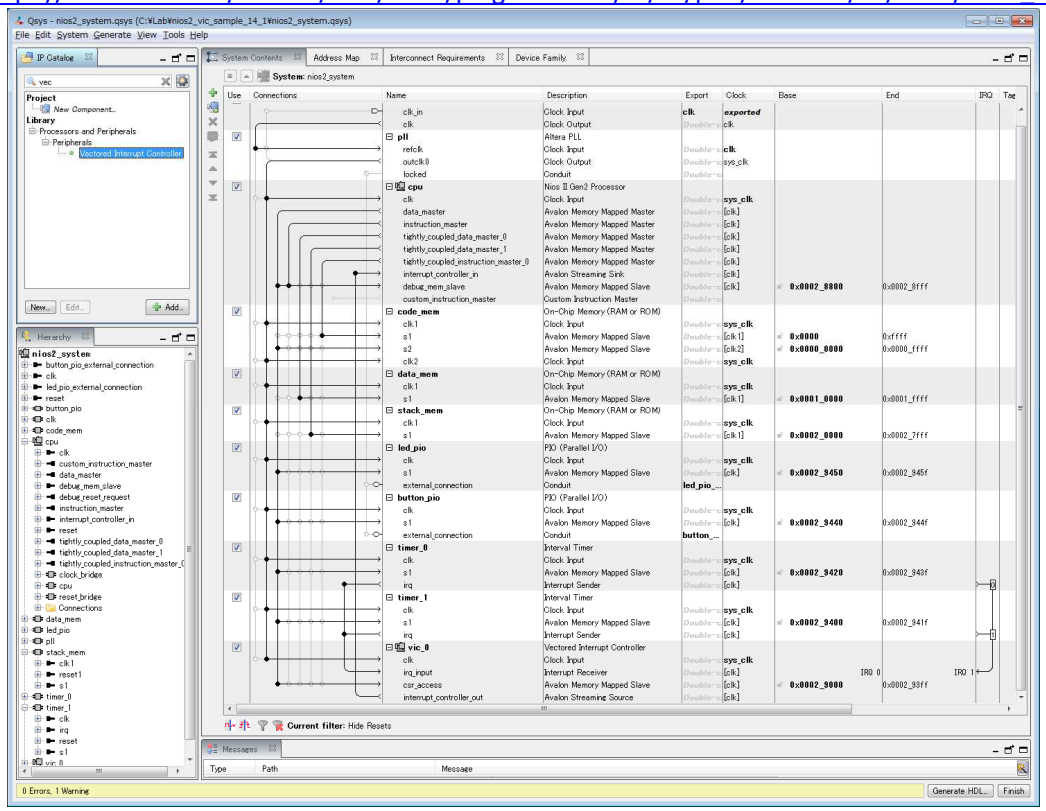
- Quartus® II 開発ソフトウェア v14.1
- Nios® II SBT v14.1
- ModelSim SE 10.1c

3. システムの構成

3-1. Qsys の設定

Qsys システム統合ツール（以下、Qsys）では、割り込み制御の検証のために、2 つのタイマーを実装し IRQ 番号を 0 と 1 で割り当てています。また、レスポンス計測時のキャッシュの影響を除くため、Nios® II のコード・メモリやデータ・メモリをタイトリー・カップルド・メモリーで実装しています。タイトリー・カップルド・メモリーに関しては、下記をご参照ください。

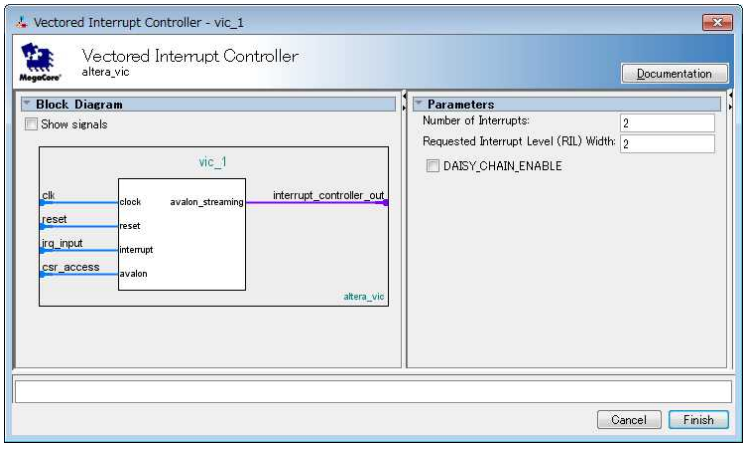
https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/nios2/n2sw_nii5v2gen2.pdf



3-2. ペリフェラルの設定

3-2-1. VIC の設定

- Number of Interrupts : “2” ・・・ IRQ 番号の上限
- Requested Interrupt Level (RIL) Width : “2bit” ・・・ 割り込みレベルのビット幅（0～3）

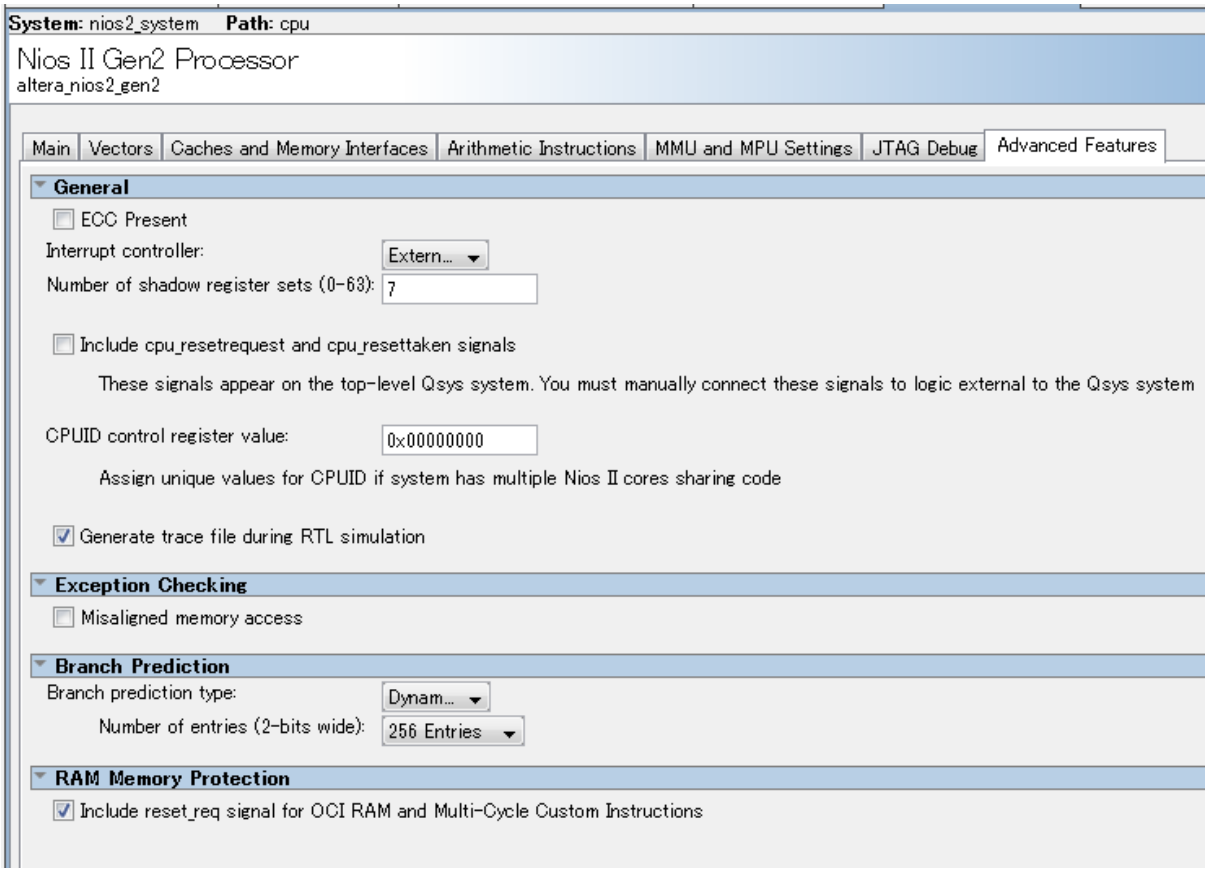


3-2-2. Nios® II の設定

- Interrupt Controller : “External” ・・・ VIC を使用する場合は “External” を選択

  ※これまでの内部の割り込みコントローラを使用する場合は ”Internal” を選択

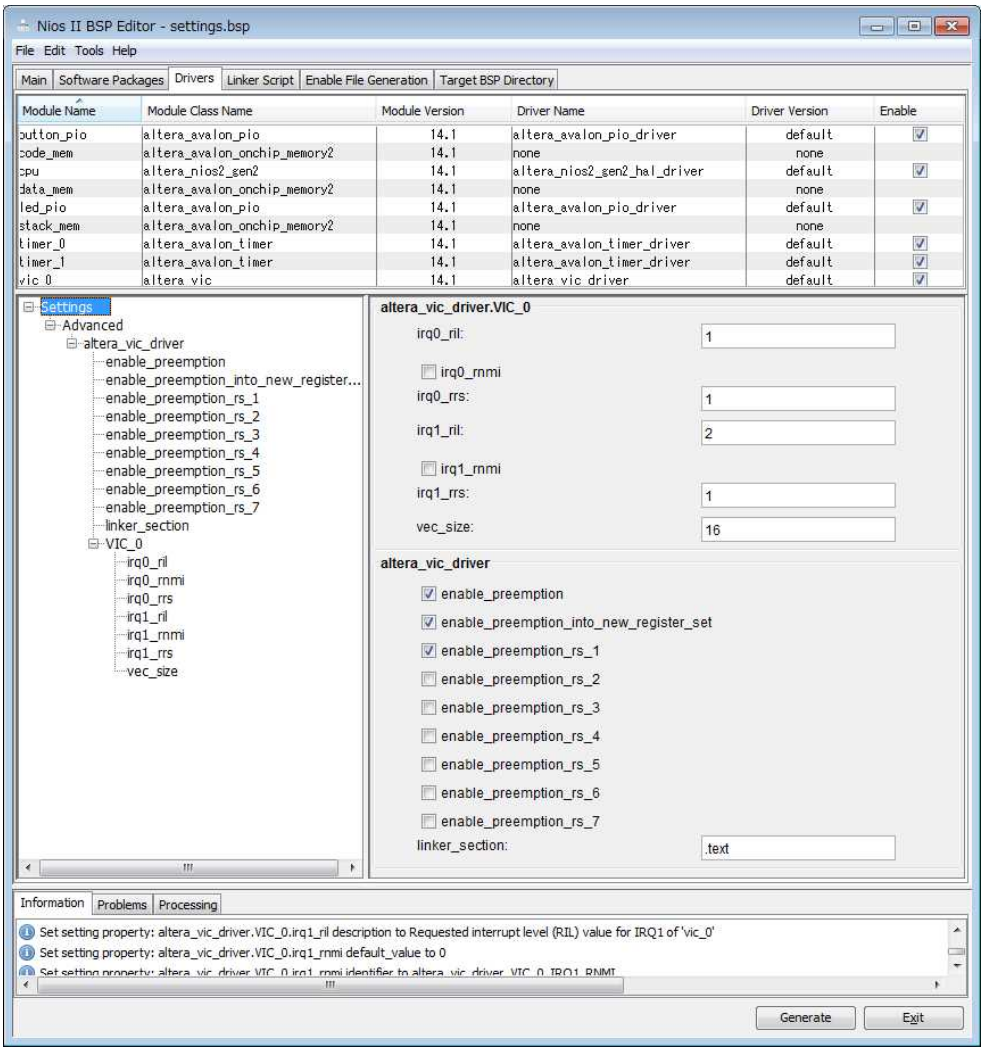
- Number of shadow register sets : 7 ・・・ シャドウ・レジスタの数（標準は 7）



4. Nios® II SBT での実装

4-1. BSP の設定

BSP Editor を開き、Drivers タブの Settings を選択します。

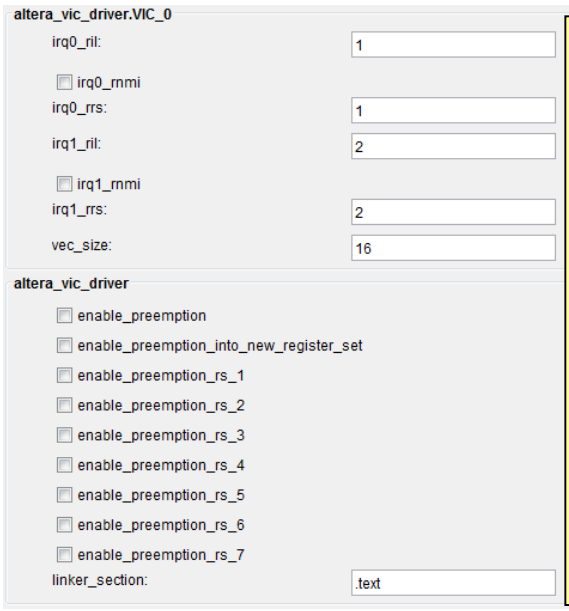


- enable_preemption : 割り込みのネストを許可します。
- enable_preemption_into_new_register_set :プライオリティーの低い割り込みの実行時に、異なるレジスタ・セットを使用するプライオリティーの高い割り込みが発生した場合にはネストを許しますが、同じレジスタ・セットを使用する割り込みは制限します。
- enable_preemption_rs_1 ～ 7 : プライオリティーの低い割り込みの実行時に、同じレジスタ・セットを使用するプライオリティーの高い割り込みが発生した場合、実行途中のプライオリティーの低い割り込みは、レジスタの値をシャドウ・レジスタでは無くスタックに積んでプライオリティーの高い割り込みに遷移します。ただし、この動作はスタックに退避するオーバー・ヘッドを含むため、割り込みの応答時間に余裕のある処理で使用してください。
- irqN_ril : 割り込みレベルを設定します。
- irqN_rnml : ノンマスカブル割り込みを選択します。
- irqN_rrs : 使用するシャドウ・レジスタを設定します。

4-2. 割り込みネストを許可しない場合の割り込み

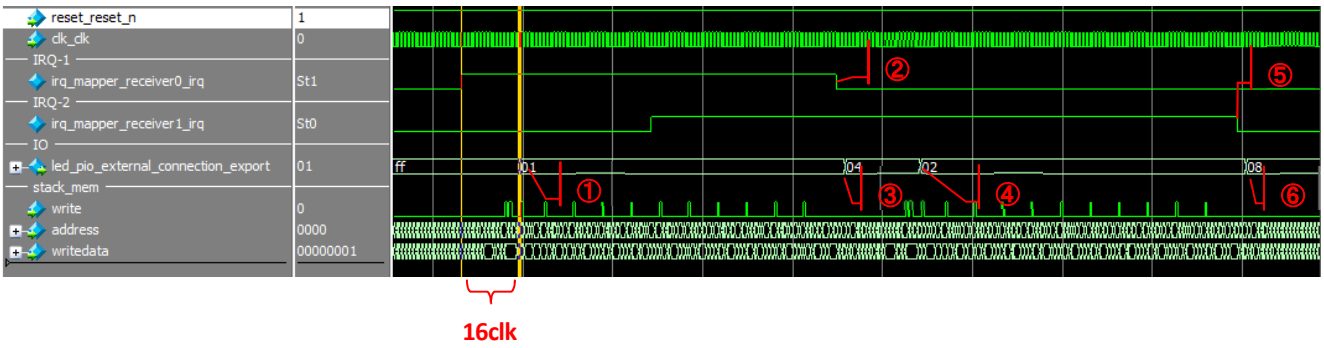
2 つの割り込みを用意し、1つの割り込み処理の途中で別の割り込みを発生させ,波形シミュレーションを行っています。なお、割り込みサービス・ルーチンの動作状況を判別するため、関数の先頭と最後で I/O に任意の値を書き込む処理を追加しています。

【BSP の設定】




```
void isr_timer0(void *context)
{
        volatile int i;
        // 処理マーカー
        IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, 0x01); -①
        for(i = 0;i < 10;i ++);   // Wait
        // 割り込みのクリア
        IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER_0_BASE,0); -②
        // 処理マーカー
        IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, 0x04); -③
}

void isr_timer1(void *context)
{
        volatile int i;
        // 処理マーカー
      IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, 0x02); -④
        for(i = 0;i < 10;i ++);   // Wait
        // 割り込みのクリア
        IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER_1_BASE,0);   -⑤
        // 処理マーカー
        IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, 0x08);   -⑥
}
```



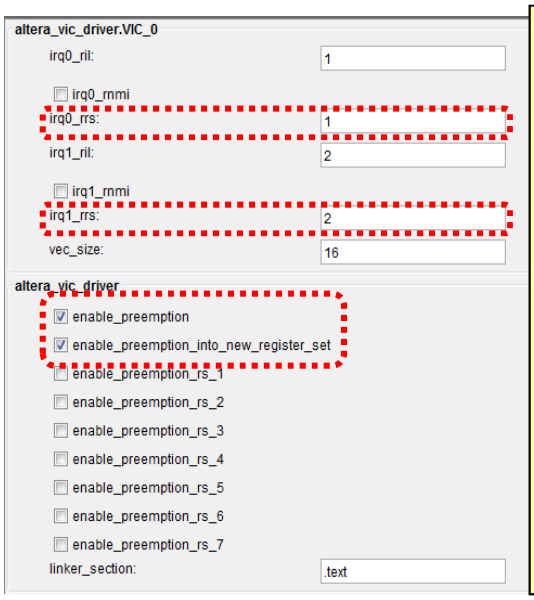
IRQ-1 の処理の途中で IRQ-2 が発生しています。IRQ-1 は IRQ-2 よりも割り込みレベル (RIL) が低いのですが、ネストを許可していないため IRQ-1 が終了するまで IRQ-2 の実行は待たされます。

なお、VIC を使った割り込みでは、割り込み発生からサービス・ルーチンの実行までの応答時間は 16clk となっています。

4-3. 割り込みネストを許可する場合の割り込み（異なるレジスタ・セット）

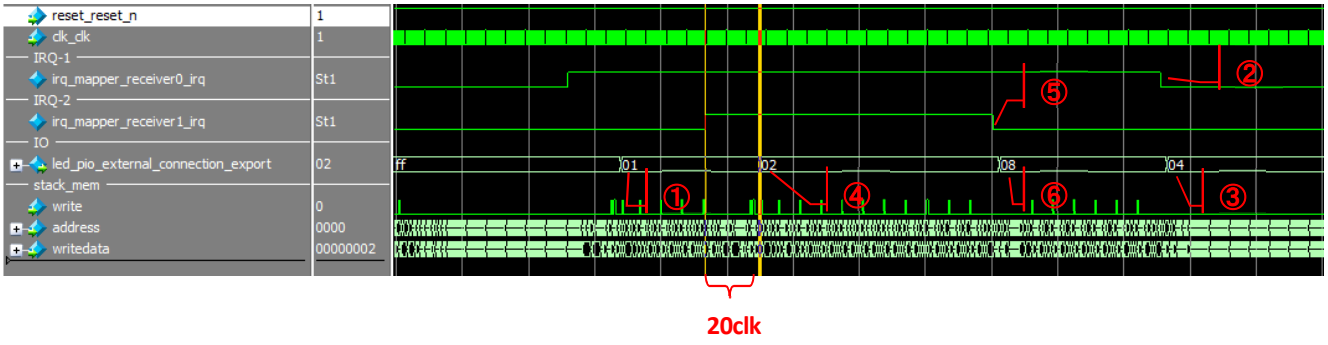
前述の 4-2 と同じ条件で、BSP の enable_preemption と enable_preemption_into_new_register_set に チェックを入れてシミュレーションを行っています。

【BSP の設定】




```
void isr_timer0(void *context)
{
        volatile int i;
        // 処理マーカー
        IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, 0x01); -①
        for(i = 0;i < 10;i ++);   // Wait
        // 割り込みのクリア
        IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER_0_BASE,0); -②
        // 処理マーカー
        IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, 0x04); -③
}

void isr_timer1(void *context)
{
        volatile int i;
        // 処理マーカー
      IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, 0x02); -④
        for(i = 0;i < 10;i ++);   // Wait
        // 割り込みのクリア
        IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER_1_BASE,0);   -⑤
        // 処理マーカー
        IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, 0x08);   -⑥
}
```



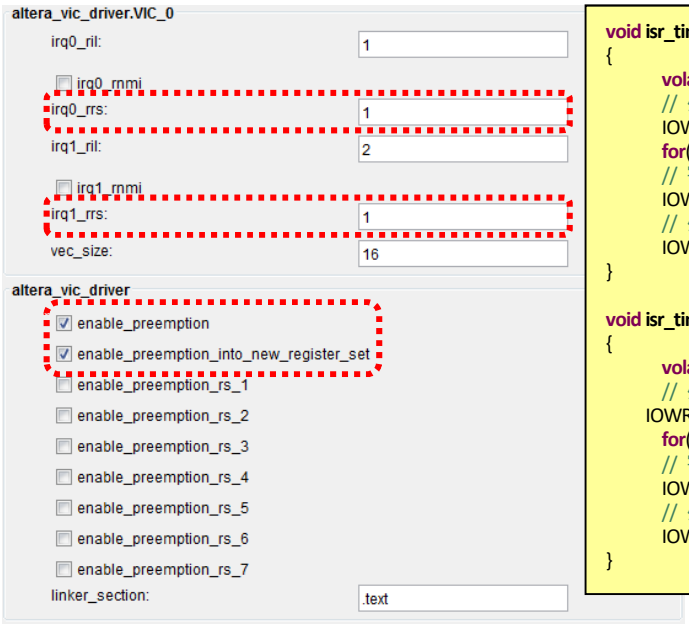
IRQ-1 の処理の途中で IRQ-2 が発生していますが、レジスタ・セットが異なる割り込みのため IRQ-1 の中で IRQ-2 に制御が移されています。

なお、ネスト割り込みでは、割り込み発生からサービス・ルーチンの実行までの応答時間は 20clk となっています。

4-4. 割り込みネストを許可する場合の割り込み（同一レジスタ・セット）

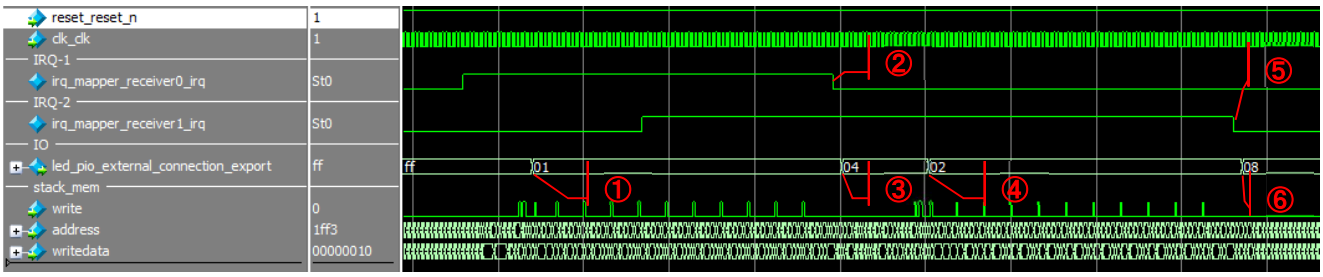
前述の 4-3 と同じ条件で、BSP のレジスタ・セット (Requested Register Set : RRS) を同一にし、シミュレーションを行っています。

【BSP の設定】




```
void isr_timer0(void *context)
{
        volatile int i;
        // 処理マーカー
        IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, 0x01); -①
        for(i = 0;i < 10;i ++);   // Wait
        // 割り込みのクリア
        IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER_0_BASE,0); -②
        // 処理マーカー
        IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, 0x04); -③
}

void isr_timer1(void *context)
{
        volatile int i;
        // 処理マーカー
      IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, 0x02); -④
        for(i = 0;i < 10;i ++);   // Wait
        // 割り込みのクリア
        IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER_1_BASE,0);   -⑤
        // 処理マーカー
        IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, 0x08);   -⑥
}
```

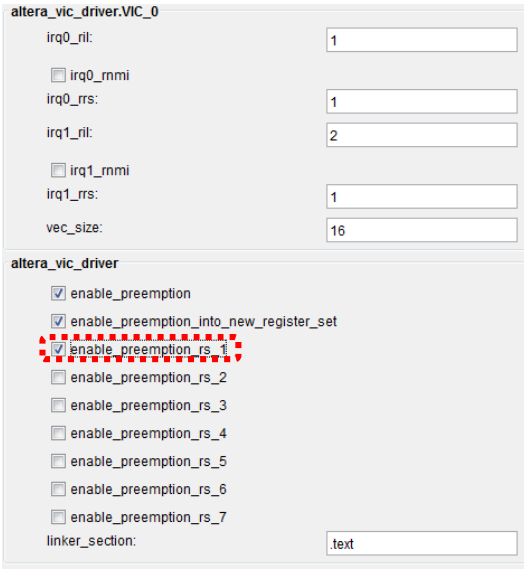


IRQ-1 の処理の途中で IRQ-2 が発生しています。IRQ-1 は IRQ-2 よりも割り込みレベル (RIL) が低いのですが、同一のレジスタ・セットが指定されているため IRQ-1 が終了するまで IRQ-2 の実行は待たされます。

4-5. 同一レジスタ・セットでも割り込みネストを許可する場合の割り込みの設定

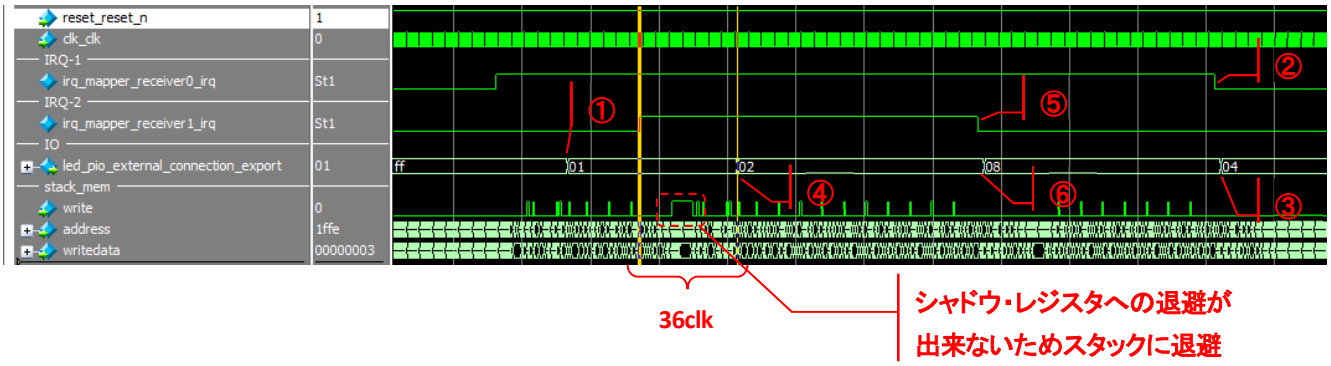
前述の 4-4 と同じ条件で、BSP の enable_preemption_rs_1 にチェックを入れてシミュレーションを行っています。

【BSP の設定】




```
void isr_timer0(void *context)
{
        volatile int i;
        // 処理マーカー
        IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, 0x01); -①
        for(i = 0;i < 10;i ++);   // Wait
        // 割り込みのクリア
        IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER_0_BASE,0); -②
        // 処理マーカー
        IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, 0x04); -③
}

void isr_timer1(void *context)
{
        volatile int i;
        // 処理マーカー
      IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, 0x02); -④
        for(i = 0;i < 10;i ++);   // Wait
        // 割り込みのクリア
        IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER_1_BASE,0);   -⑤
        // 処理マーカー
        IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, 0x08);   -⑥
}
```



IRQ-1 の処理の途中で IRQ-2 が発生しています。IRQ-1 と IRQ-2 は同じレジスタ・セットを使用しますが、ネストを許可しているため、IRQ-2 はシャドウ・レジスタでは無くスタックにレジスタの値を退避します。

この条件での割り込みでは、スタックへの退避をソフトウェアで実現しているため、通常より長い応答時間が必要となりますので、ご注意ください。

5. ソフトウェアの変更

5-1. 記述の違い

Nios® II 内部の割り込みコントローラを使用する HAL の記述と、新しい VIC を使用した HAL の記述を対比しています。

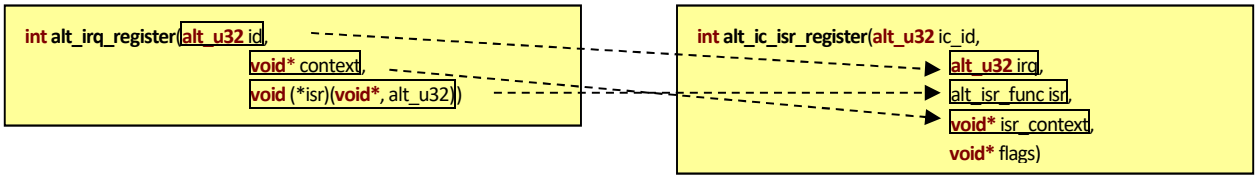
	内部割込みコントローラ使用時	VIC 使用時
HAL API	alt_irq_register() alt_irq_disable() alt_irq_enable() alt_irq_interruptible() alt_irq_non_interruptible() alt_irq_disable_all( ) alt_irq_enable_all( ) alt_irq_enabled()	alt_ic_irq_enable() alt_ic_irq_disable() alt_ic_irq_enabled() alt_ic_isr_register()

5-2. 記述の変更

割り込みサービス・ルーチンの登録と、割り込みサービス・ルーチン自身の記述の違いを下記に示します。

割り込みサービス・ルーチンの登録 (alt_irq_register) を使用している場合、新たな API を使用する場合大きな変更は必要ありません。唯一、VIC のデイジー・チェーン時に必要となる ic_id を system.h から取得して設定することで変更は終了します。

また、割り込みサービス・ルーチン自身については、これまで在った id が削除されておりますので、割り込みサービス・ルーチンを複数の割り込みで共用している場合などは、isr_context で渡す情報の中に割り込みを識別できるような情報を埋め込んで頂く等の対応が必要となりますのでご注意ください。



【新たに追加された引数】

alt_u32 ic_id : デイジー・チェーンの VIC を使用する際の VIC ID を指定します。この値は system.h に記述されています。例. VIC_0_INTERRUPT_CONTROLLER_ID

```
void (*alt_isr_func)(void* isr_context, alt_u32 id);
```

```
void (*alt_isr_func) (void* isr_context);
```

上記以外にも割り込みを Enable/Disable する関数などがありますが、それらは BSP での設定等で対応可能です。

改版履歴

Revision	年月	概要
1	2015 年 4 月	初版
2	2019 年 9 月	内容をそのままにフォーマットを更新（リンク先 URL など微修正）