AN595: ベクトル割り込みコントローラの使用およびアプリケーション

© 2009 年 11 月 AN-595-1.0

割り込みイベントを迅速に処理して、多数の割り込みを扱う能力は、多くのエンベデット・システムに重要です。ベクトル割り込みコントローラ (VIC) はこれらの要件に対処するために設計されています。VIC は Nios® II プロセッサのデフォルトの内部割り込みコントローラ (IIC) より 4 ～ 5 倍良い割り込み性能を提供することができます。また、VIC はデイジー・チェインを通して、バーチャル的に無制限の割り込み数の拡大も可能になります。

本資料では、ハードウェアとソフトウェアの観点から、ハードウェア・デザインで VIC を使用する方法について説明します。この章は、以下の項で構成されています。

前提条件

このドキュメントを完全に理解するには、以下の項目について精通していることが必要です。

- SOPC Builder でのシステムの作成
- Nios® II プロセッサの外部割り込みコントローラ (EIC) のインタフェース
- VIC コア
- Nios II エンベデッド・デザイン・スィート（EDS）によるソフトウェアの開発
- Altera® HAL 割り込みのアプリケーション・プログラミング・インタフェース（API）
- Nios II ソフトウェア・ビルド・ツールによるソフトウェアの作成とビルド

SOPC Builder について詳しくは、「Quartus II ハンドブック」の *「Volume 4: SOPC Builder」* を参照してください。Nios II プロセッサの EIC インタフェースについて詳しくは、「NiosII プロセッサ・リファレンス・ハンドブック」の *「Processor Architecture」* および *「Programming Model」* の章を参照してください。VIC コアについて詳しくは、「Quartus II ハンドブック Volume 5: エンベデッド・ペリフェラル」の *「Vectored Interrupt Controller Core」* の章を参照してください。割り込み API とソフトウェア・ビルド・ツールを含む Nios II EDS について詳しくは、 *「Nios II Software Developer's Handbook」* を参照してください。

VIC ハードウェアの概要

この項では、VIC を使用する Nios II システムの必要なハードウェア・コンポーネントについて説明します。

外部割り込みコントローラのインタフェース

VIC とは EIC のアルテラの実装です。EIC は、Nios II プロセッサ・コアから分離するコンポーネントとして、実装された割り込みコントローラです。EIC は Nios II プロセッサの IIC に高性能の代替を提供します。

EIC インタフェースは SOPC Builder 内の Nios II プロセッサに追加できる機能です。EIC インタフェースには、Avalon ストリーミング (Avalon-ST) シンクが含まれるため、EIC がプロセッサに割り込み情報を通信することを許容します。また、EIC インターフェイスの Avalon-ST シンクにより、接続された EIC はデイジー・チェイン EIC からの割り込み情報を中継することができます。

このアプリケーション・ノートは VIC についてのみ説明しますが、Nios II プロセッサの上の EIC インタフェースはカスタム EIC もサポートするように設計されています。

EIC インタフェースの一般的な情報について詳しくは、「Nios II プロセッサ・リファレンス・ハンドブック」の *「Processor Architecture」* および *「Programming Model」* の章を参照してください。

シャドー・レジスタ・セット

シャドー・レジスタ・セットを Nios II プロセッサに追加することができます。シャドー・レジスタ・セットを VIC と併用することにより、例外コンテクストとアプリケーション・コンテクストを切り換えるのに費やされた多くの時間が不要になります。EIC の HAL 割り込みサポートには少なくとも 1 つのシャドー・レジスタ・セットが必要です。

ベクトル割り込みコントローラ

VIC は高性能で、低レイテンシの割り込み処理を提供します。VIC はハードウェアでの割り込みを優先して、最高優先順位の割り込み待ちに関する情報を出力します。

VIC は Nios II プロセッサの EIC インタフェースと連携して動作します。VIC は、デイジー・チェイン・コンフィギュレーション内の EIC とのハードウェア互換性のために設計されます。ただし、Nios II HAL（Hardware Abstraction Layer）は、それらがすべて同じドライバーによってサポートされるように、デイジー・チェイン内のすべての EIC が同じクラスであることが必要です。

VIC を使用する理由

以下の理由のために、ハードウェア・デザインで VIC を使用する可能性があります。

- 平均応答時間を 1 つ以上の割り込みまで減少する必要があります。
- 割り込み性能に対してハード・リアルタイムの要件があります。
- マスク不可能割り込みを必要とします。
- 32 を超える割り込み（IIC でサポートされる最大割り込み数）を処理する必要があります。

SOPC Builder における VIC の実装

この項では、SOPC Builder システムにおける 1 つ以上の VIC の組み込む方法およびソフトウェアで VIC をサポートする方法について説明します。

VIC ハードウェアの追加

VIC を SOPC Builder システムに追加する際に、次の上位レベルのタスクを実行する必要があります。

1. EIC インタフェースを Nios II プロセッサ・コアに追加します。
2. オプションで、シャドー・レジスタ・セットを Nios II プロセッサ・コアに追加します（HAL の割り込みサポートを使用する場合に必要です）。
3. 1 つ以上の VIC コンポーネントを追加してパラメータ化します。
4. 割り込みソースを VIC コンポーネントに接続します。

EIC インタフェースのシャドー・レジスタ・セットの追加

この章は、MegaWizard™ インタフェースを通した SOPC Builder 内の Nios II プロセッサ・コアに EIC インタフェースおよびシャドー・レジスタ・セットを追加する方法について説明します。

1. SOPC Builder で、Nios II プロセッサをダブル・クリックして、MegaWizard インタフェースを開きます。
2. 図 1 に示すように、**Advanced Features** タブの **Interrupt Controller** リストで選択することによって、Nios II プロセッサ上の EIC インタフェースをイネーブルします。

   Interrupt Controller に対して、2 つのオプションを使用できます。それは **Internal** と **External** です。**Internal** を選択する場合、プロセッサは内部の割り込みコントローラで実装されます。EIC インタフェースでプロセッサを実装するには、**External** を選択します。

   ☞ EIC インタフェースを実装するとき、VIC などの EIC を接続しなければなりません。EIC を接続しない場合、SOPC Builder 誤りが発生します。

3. ご希望のシャドー・レジスタ・セットの数を選択します。**Number of shadow register sets** リストで、システム性能目標と一致するレジスタ・セットの数を選択します。
4. **Finish** をクリックして、Nios II MegaWizard インタフェースを終了します。図 2 に示すように、プロセッサが未接続 `interrupt_controller_in` Avalon-ST シンクを示すことに注意してください。

図 1. 割り込みコントローラおよびシャドー・レジスタ・セットのコンフィギュレーション

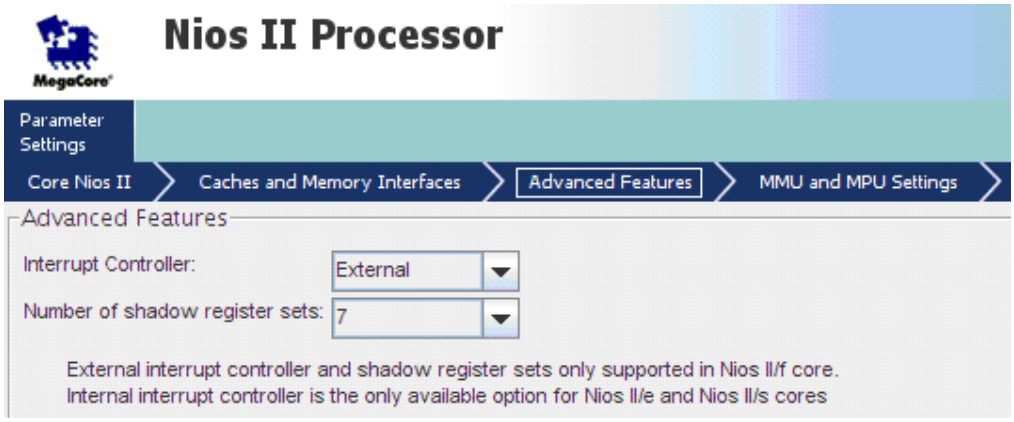


図 2. EIC インタフェース付き Nios II プロセッサ

シャドー・レジスタ・セットはレジスタの保存と復元に関連するコンテクスト・スイッチ・オーバーヘッドを低減ます。そうしないと、これは重要になる可能性があります。可能な限り、高性能を必要とする各割り込みに 1 つのシャドー・レジスタ・セットを追加します。

VIC のインスタンス化、パラメータ化、および接続

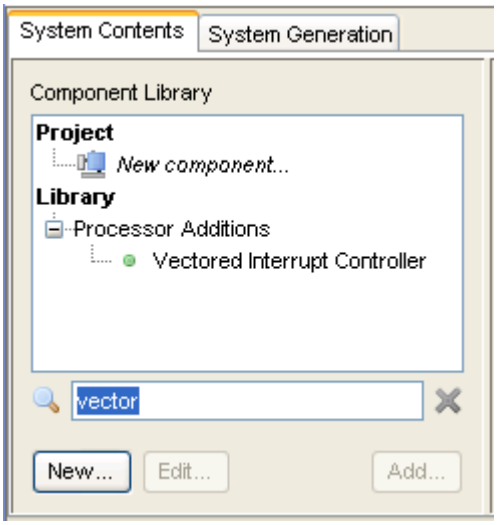
EIC インタフェースとシャドー・レジスタ・セットを Nios II プロセッサに追加した後、SOPC Builder システム内の VIC をインスタンス化してパラメータ化します。

インスタンス化

SOPC Builder システム内の VIC をインスタンス化するには、以下のステップを実行します。

1. SOPC Builder の **System Contents** タブでの **Component Library** カラムに移動します。
2. このカラムの下にある検索ボックスに `vector` を入力します。図 3 に示すように、MegaWizard インタフェースは VIC を除くすべてのコンポーネントを非表示にします。
3. このコンポーネントを SOPC Builder システムに追加するには、ベクトル割り込みコントローラのコンポーネントをダブル・クリックします。

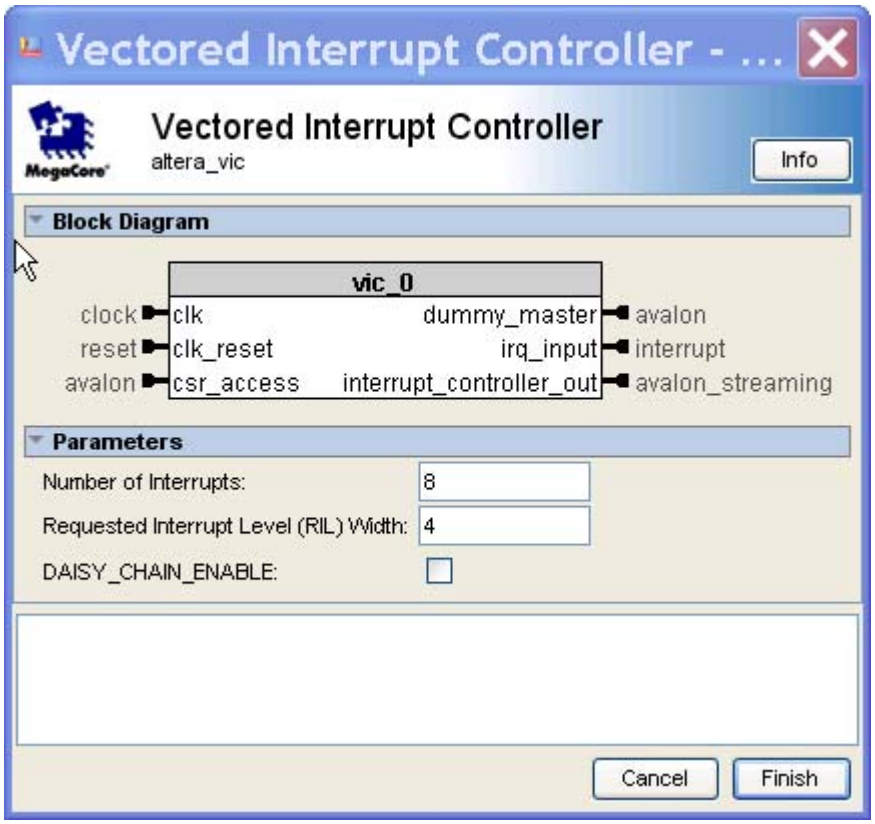
図 3. ベクトル割り込みコントローラのコンポーネント



パラメータ化

VIC をシステムに追加する際に、**Vectored Interrupt Controller** MegaWizard インタフェースは図 4 のように表示されます。

図 4. ベクトル割り込みコントローラのパラメータ化



VIC MegaWizard インタフェースにより、以下のオプションを指定できます。

- **割り込みの数** —VIC がサポートする必要がある割り込みの数。
- **要求された割り込みレベル (RIL) の幅** — ビット割り当て数は各割り込みの割り込みレベルを表します。

  デフォルトの RIL アサインメントのルールを含む RIL の完全な説明については、「QuartusII ハンドブック Volume 5: エンベデッド・ペリフェラル」の *「Vectored Interrupt Controller Core」* の章を参照してください。

- **DAISY_CHAIN_ENABLE**—VIC を別の EIC へのデイジー・チェインを可能にします。システム内の複数の VIC をサポートする場合、このオプションをオンにします。

  使用例については、本資料に付属する VIC デイジー・チェインの例を参照してください。

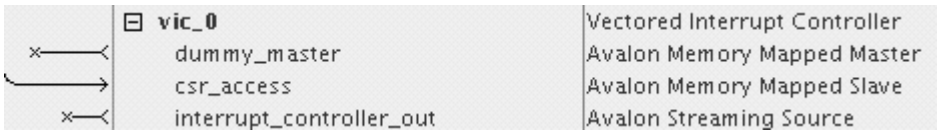
VIC のパラメータ化が完了したら、**Finish** をクリックして、SOPC Builder システム内のコンポーネントをインスタンス化します。

VIC の接続

システムに VIC を追加したら、それは図 5 に示すように SOPC Builder で表示されます。

デイジー・チェインがイネーブルされているとき、SOPC Builder は `interrupt_controller_in` と呼ばれる Avalon-ST シンクを VIC に追加します。

図 5. VIC インタフェース

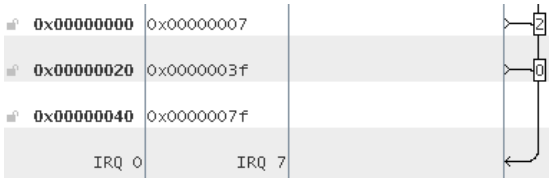


SOPC Builder システムに VIC を追加した後、VIC と EIC インタフェースをシステム・レベルでパラメータ化する必要があります。VIC を追加した直後、いくつかのエラー・メッセージが表示されます。以下の操作を順番的に実行することによって、これらのエラー・メッセージを解決します。

- デイジー・チェインのコンフィギュレーションの次の VIC または Nios II プロセッサのいずれかで、VIC の `interrupt_controller_out` Avalon-ST を `interrupt_controller_in` Avalon-ST シンクに接続します。
- VIC の `dummy_master` Avalon Memory-Mapped (Avalon-MM) ポートを `csr_access` Avalon-MM スレーブ・ポートに接続します。
- 図 6 に示すように、システムの各割り込みベースのコンポーネントに割り込み番号を割り当てます。このステップにより、各コンポーネントを VIC の上の割り込みポートに接続します。

  システムがシングルのプロセッサに接続する 1 つ以上の EIC を含んでいる場合、各コンポーネントが割り込みポートに 1 つの EIC だけに接続されることを確認する必要があります。

図 6. 割り込み番号の割り当て



HAL VIC ドライバを使用するとき、ドライバはレジスタ・セットから割り込みまでデフォルトのアサインメントを作成します。デバイスが VIC に接続される方法に基づいて、デフォルトのアサインメントは割り込み優先順位に関するいくつかの仮定をします。

デフォルト RIL および RIL 設定を含むデフォルトのアサインメントについて詳しくは、「Quartus II ハンドブック Volume 5: エンベデッド・ペリフェラル」の *「Vectored Interrupt Controller Core」* の章における「アルテラ HAL ソフトウェア・プログラミング・モデル」を参照してください。

VIC 割り込み設定のデフォルトを有効に利用するには、プロセッサに最も近い VIC で低い割り込みポート番号に最優先の割り込みを割り当てます。

ソフトウェア

VIC コンポーネントに基づいてシステムの割り込みハンドラを書き込む場合、HAL 強化割り込み API を使用してハンドラを登録し、そのランタイム環境を制御する必要があります。強化割り込み API は VIC を含む EIC と共に使用する多くの機能を提供します。この項では、強化割り込み API におけるファンクションのサブセットについて説明します。

強化割り込み API について詳しくは、「Nios II ソフトウェア開発ハンドブック」の *「Exception Handling」* の章における「割り込みサービス・ルーチン」を参照してください。

特に、このセクションでは、強化 API とレガシ API の両方をサポートするようにドライバをコードする方法を説明します。これは、強化 API の存在をテストして、条件付きに適切な機能を呼び出す必要があります。

alt_ic_isr_register() と alt_irq_register()

強化 API ファンクション `alt_ic_isr_register()` はいくつかの重要な相違点がありますが、レガシ・ファンクション `alt_irq_register()` に非常に似ています。これらの 2 つの機能の違いは、例 1 のコードを調べることによって、理解できます。この例はボード・サポート・パッケージ（BSP）に実装されるレガシ API または強化 API のいずれかにタイマー割込みを登録します。例 1 はこのドキュメントに付属するコード例から直接取得されたものです。

例 1. 両方の API による ISR のレジスタ

```
#ifdef ALT_ENHANCED_INTERRUPT_API_PRESENT
void timer_interrupt_latency_init (void* base, alt_u32 irq_controller_id, alt_u32 irq)
{
 /* Register the interrupt */
 alt_ic_isr_register(irq_controller_id, irq, timer_interrupt_latency_irq, base, NULL);
 /* Start timer */
 IOWR_ALTERA_AVALON_TIMER_CONTROL(base, ALTERA_AVALON_TIMER_CONTROL_ITO_MSK
 | ALTERA_AVALON_TIMER_CONTROL_START_MSK);
}
#else
void timer_interrupt_latency_init (void* base, alt_u32 irq)
{
 /* Register the interrupt */
 alt_irq_register(irq, base, timer_interrupt_latency_irq);
 /* Start timer */
 IOWR_ALTERA_AVALON_TIMER_CONTROL(base, ALTERA_AVALON_TIMER_CONTROL_ITO_MSK
 | ALTERA_AVALON_TIMER_CONTROL_START_MSK);
}
#endif
```

例 1 の最初の行では、BSP が強化割り込み API を実装するかどうかを検出します。強化 API が実装された場合、timer_interrupt_latency_init() ファンクションは強化ファンクションを呼び出します。そうしないと、timer_interrupt_latency_init() はレガシ割り込み API ファンクションに戻ります。

Nios II ソフトウェア・ビルド・ツールは BSP でどの API を実装するかを選択する方法の説明については、「Nios II ソフトウェア開発ハンドブック」の *「Exception Handling」* の章における「割り込みサービス・ルーチン」を参照してください。

例 2 に、強化 API に ISR を登録する alt_ic_isr_register() の関数のプロトタイプを示します。割り込みコントローラ識別子（引数 ic_id 用）および割り込みポート番号（引数 irq 用）は **system.h** で定義されます。

例 2. 強化関数 alt_ic_isr_register()

```
extern int alt_ic_isr_register(alt_u32 ic_id,
 alt_u32 irq,
 alt_isr_func isr,
 void *isr_context,
 void *flags);
```

比較的に、例 3 はレガシ API に ISR を登録する alt_irq_register() の関数のプロトタイプを示します。

例 3. レガシ関数 alt_irq_register()

```
extern int alt_irq_register (alt_u32 id,
 void* context,
 alt_isr_func handler);
```

alt_ic_isr_register() に渡された引数は、alt_irq_register() に渡されたものとわずかに異なっています。表 1 は 2 つの関数の引数を比較します。

表 1. alt_ic_isr_register() と alt_irq_register() の引数

alt_ic_isr_register() の引数	用途	alt_irq_register() の引数
`alt_u32 ic_id`	**system.h** で定義される固有の割り込みコントローラ ID。	—
`alt_u32 irq`	**system.h** で定義される割り込み要求 (IRQ) の番号。	`alt_u32 id`
`alt_isr_func isr`	割り込みサービス・ルーチン (ISR) の関数ポインタ。	`handler`
`void* isr_context`	コンポーネント特定のデータ構造体へのオプションのポインタ。	`context`
`void* flags`	予約されている。他の EIC の実装は、この引数を使用する可能性があります。	なし

レガシ・割り込み API と強化割り込み API の間には、他の著しい違いがあります。これらの違いのいくつかは、ISR ボディに影響します。特に、2 つの API が完全に異なった割り込み先取りモデルを使用します。このアプリケーション・ノートに付属するコード例では、多くの違いを示します。

HAL 割り込み API における他の関数について詳しくは、「Nios II ソフトウェア開発ハンドブック」の *「Exception Handling」* および *「HAL API Reference」* の章、かつ「Quartus II ハンドブック Volume 5: エンベデッド・ペリフェラル」の *「Vectored Interrupt Controller Core」* の章を参照してください。

デザイン例

このセクションでは VIC の使用法を示すために、このアプリケーション・ノートと共に提供されるデザイン例について簡単な説明を提供します。更に、このセクションはソフトウェア例を Nios II エンベデッド評価キット (NEEK) ハードウェアに実行するための指示を提供します。

例記述

デザイン例は **AN595_VIC_collateral.zip** と呼ばれるファイルに提供されます。

AN595_VIC_collateral.zip はアルテラ・ウェブサイトの Literature: Nios II Processor ページにあります。ファイルのリンクは *「AN595: Vectored Interrupt Controller Usage and Applications」*（本資料）の隣に表示されます。

表 2 に、ファイルで見つけられた各デザイン例について説明します。

表 2. AN595_VIC_collateral.zip におけるデザイン例

例の名称	フォルダの名称	説明
VIC Basic	**VIC_Example**	シングルの VIC
VIC Daisy-Chain	**VIC_DaisyChain_Example**	2 つのデイジー・チェーン VIC
VIC Table-Resident	**VIC_ISRnVectorTable_Example**	ベクトル・テーブルに配置している ISR を有する VIC
IIC	**VIC_noVIC_Example**	VIC 例と比較する用の IIC 例

AN595_VIC_collateral と呼ばれる **AN595_VIC_collateral.zip** 内のトップレベル・フォルダは、次のファイルが含まれます。

- **run_sw.sh**—1 つ、数個またはすべての例を実行するためのシェル・スクリプト
- **README.txt**—.zip ファイルの内容を説明

各例のハードウェアは図 7 と図 8 のデザインに基づいています。図 7 に、SOPC Builder 内の VIC Basic システム例を示します。

図 7. VIC Basic の例

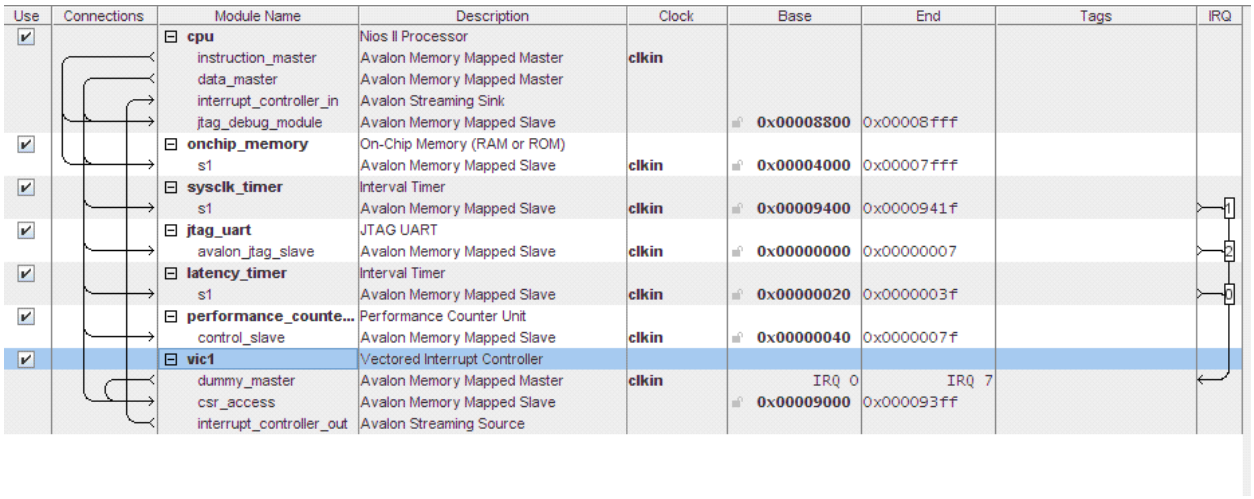


図 8 に、SOPC Builder 内のデイジー・チェーンのシステム例を示します。

図 8. VIC デイジー・チェーンの例

VIC と EIC インタフェースを IIC に置き換えられて、IIC デザインは VIC Basic デザインと同じです。VIC Table-Resident デザインは VIC Basic デザインと同じです。

各例では、ソフトウェアは割り込み性能を測定するために性能カウンタに関連するタイマーを使用します。各例のソフトウェアは性能を計算して、結果を stdout に送信します。

AN595_VIC_collateral.zip は **run_sw.sh** のスクリプトを含み、1 つ、数個またはすべての例を実行します。**run_sw.sh** は各例のために、SRAM Object File (**.sof**) および Executable and Linkable Format File (**.elf**) をダウンロードします。そして、コマンド・ラインで指定する例に対して、NEEK ハードウェアの上のコードを実行します。

☞ **run_sw.sh** は、1 つだけの JTAG ダウンロード・ケーブルがホスト・コンピュータに接続されることを仮定します。複数の JTAG ケーブルがある場合、NEEK ハードウェアに接続されたケーブルを指定するように **run_sw.sh** を変更する必要があります。

使用例

最初に、ご使用のハードウェアに対してデザイン例の性能を観察するために、アルテラは各デザイン例をそのまま実行することを勧めます。その後、いずれかの例を変更することで、VIC の性能オプションを調べたり、あるいはアプリケーションのコードをカスタマイズすることができます。

各デザイン例を実行するには、次のステップを実行します。

1. NEEK ハードウェアをパワー・アップします。
2. USB ケーブルを接続します。
3. **AN595_VIC_collateral.zip** ファイルを作業ディレクトリに解凍して、フォルダ名を展開します。

   ☞ 作業ディレクトリへのパス名にスペースを含めないように注意してください。

4. Nios II コマンド・シェルで、トップレベル・ディレクトリ、**AN595_VIC_collateral** に移動します。
5. コマンド・プロンプトで次のコマンドを入力します。

   ```
   ./run_sw.sh↵
   ```

   スクリプトはオプションのリストを示しています。
6. 実行する例を指定するか、またはすべての例を実行するために、コマンドラインのオプションを使用して **run_sw.sh** を再実行します。例 4 に、サンプル・セッションを示します。

   run_sw.sh スクリプトは、以下のステップを実行します。

   a. コマンド・ライン引数を解析し、実行する例を決定します。
   b. 選択した例の **.sof** をダウンロードします。
   c. 選択した例の **.elf** をダウンロードします。
   d. **nios2-terminal** を起動して、ソフトウェアの出力をキャプチャします。

ソフトウェアの説明

様々なデザイン例のソフトウェアは非常に似ています。例えば、VIC Basic 例のソフトウェアと IIC 例のソフトウェアの違いは、端末に出力を生成する `printf()` 呼び出しです。

すべてのソフトウェアは、以下のステップを実行します。

1. 測定目的に使用されるタイマーをコンフィギュレーションします。
2. 割り込みサービス・ルーチン (ISR) を登録します。
3. グローバル変数を `0xfeedface` に設定します。
4. 割り込み時間を測定するために、性能カウンタを開始ます。
5. ISR が `0xfacefeed` にグローバル変数を設定することを待ちます。
6. 性能カウンタを停止して、割り込み時間を計算します。

10 ページの 図 8 に示すように、VIC Daisy-Chain の例はデイジー・チェインで接続された両方の VIC の測定を実行します。

VIC Table-Resident コード例の動作方法について詳しくは、「ベクタ・テーブルにおける ISR の位置」を参照してください。ソフトウェア例における性能カウンタの利用について詳しくは、19 ページの「性能カウンタによるレイテンシの測定」を参照してください。

例 4.

```
[NiosII EDS]$ ./run_sw.sh --VIC_Example

Running software...

Running for VIC_Example
/cygdrive/c/Data/workdir/AN595_VIC_collateral/VIC_Example/software_examples/app/
vic_test /cygdrive/c/Data/workdir/AN595_VIC_collateral
Searching for SOF file:
in ../../../
  VIC_Example.sof

Info: ******************************************************************
Info: Running Quartus II Programmer
Info: Command: quartus_pgm --no_banner --mode=jtag -o p;c:/Data/workdir/
AN595_VIC_collateral/VIC_Example/
VIC_Example.sof
Info: Using programming cable "USB-Blaster [USB-0]"
Info: Started Programmer operation at Mon Nov 2 13:00:33 2009
Info: Configuring device index 1
Info: Device 1 contains JTAG ID code 0x020F30DD
Info: Configuration succeeded -- 1 device(s) configured
Info: Successfully performed operation(s)
Info: Ended Programmer operation at Mon Nov 2 13:00:35 2009
Info: Quartus II Programmer was successful. 0 errors, 0 warnings
    Info: Peak virtual memory: 61 megabytes
    Info: Processing ended: Mon Nov 2 13:00:35 2009
    Info: Elapsed time: 00:00:02
    Info: Total CPU time (on all processors): 00:00:00
Using cable "USB-Blaster [USB-0]", device 1, instance 0x00
Pausing target processor: OK
Initializing CPU cache (if present)
OK
Downloaded 4KB in 0.0s
Verified OK
Starting processor at address 0x00004020
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "USB-Blaster [USB-0]", device 1, instance 0
nios2-terminal: (Use the IDE stop button or Ctrl-C to terminate)

Starting VIC Example roundtrip performance test.

Interrupt Time:       48 clocks.

Sending EOT to force an exit.

nios2-terminal: exiting due to ^D on remote

Done...
```

高度なトピック

このセクションでは高度な割込み処理に役立ついくつかのトピックを提示します。

ベクタ・テーブルにおける ISR の位置

小型の重要な ISR を持つ場合、ISR コードを直接ベクトルテーブルに配置することによって、最高性能を達成できます。これにより、HAL 漏斗を通してベクトル・テーブルから ISR に分岐のオーバーヘッドを削除します。

この項では、VIC Table-Resident の例を作成するために、VIC Basic ソフトウェア例を変更する方法を説明します。ステップを理解していることを保証するために、この例を使用してください。そして、カスタム・コードにおける同等の変更を作ることができます。

ベクトル・テーブルに ISR を配置することは、高度で、エラーが発生しやすい手法であり、HAL によって直接サポートされていません。ISR コードがベクトル・テーブル項目に適合することを保証するために、十分気を付ける必要があります。ISR はベクトル・テーブル項目をオーバーフローする場合、ベクトル・テーブルにおける他の項目および割り込み処理システム全体を破壊します。

ISR をベクトル・テーブルに配置するとき、それを登録する必要がありません。ベクトル・テーブルの内容を上書きするので、`alt_ic_isr_register()` を呼び出しません。

ISR がベクトル・テーブルにあるとき、HAL は漏斗コードを提供しません。したがって、ISR コードは通常、漏斗によって処理されるコンテキスト・スイッチ動作も実行する必要があります。漏斗コンテキスト・スイッチは以下の動作のいくつか、またはすべてを含むことができます。

- レジスタの保存と復元
- 先取りの管理
- スタック・ポインタの管理

可能な限り高速な ISR を作成するには、以下のガイドラインに従うことによって、ISR が実行する必要があるコンテキスト・スイッチング動作を最小化するか、または削除します。

- ISR をアセンブリ言語で記述します。
- ISR の使用のために、シャドー・レジスタの設定を割り当てます。
- 別の ISR が同じレジスタ・セットを使用することで、ISR を先取りできないことを確認します。デフォルトで、レジスタ・セット内の先取りは Nios II プロセッサでディセーブルされます。レジスタ・セットに ISR の排他的なアクセスを与えることによって、この状態を確認することができます。

VIC Table-Resident 例は BSP 生成したファイル、**altera_vic1_vector_tbl.S** の変更を必要とします。これらの変更をした後に BSP を再生成する場合、Nios II ソフトウェア・ビルド・ツールは **altera_vic1_vector_tbl.S** を再生成して、変更が上書きされます。

ベクトル・テーブルのエントリ・サイズの増加

ISR をベクトル・テーブルに挿入するとき、ISR 全体がベクトル・テーブルのエントリに適合するように、ベクトル・エントリのサイズを増加させる必要があります。`altera_vic_driver.<vic_instance>.vec_size` BSP 設定を使用して、ベクトル・テーブルのエントリ・サイズを調整します。Nios II ソフトウェア・ビルド・ツールのコマンド・ラインでは、`--set` コマンド・ライン・オプションによるこの設定を操作することができます。Nios II BSP エディタで、この設定を変更できます。

VIC Table-Resident 例では、*<vic_instance>* が `VIC1` であり、*<size>* は 256 バイトに設定されます。

ISR を登録しない

ベクトル・テーブルに配置する割り込みのための alt_ic_isr_register() に呼び出しを除去します。それを alt_ic_irq_enable() 呼び出しに置き換えます。ベクトル・テーブルの内容を上書いて、ISR のボディを破壊するので、alt_ic_isr_register() を呼び出してはいけません。

ベクトル・テーブルに ISR の挿入

このドキュメントに含まれる VIC Table-Resident 例では、ISR コードが BSP フォルダ内の **vector.h** と呼ばれるファイルにあります。

このコードをベクトル・テーブルに挿入するには、以下のステップを実行します。

1. **create-this-bsp** スクリプトを実行することによって、BSP を生成します。
2. 例 4 に示すように、**altera_vic1_vector_tbl.S** を変更します。

例 5. altera_vic1_vector_tbl.S の変更

```
#include "altera_vic_funnel.h"
#include "vector.h"            /* ADD THIS LINE MANUALLY */
    .section .text
    .align 2
    .globl VIC1_VECTOR_TABLE
VIC1_VECTOR_TABLE:
    MY_ISR 256                 /* THIS LINE REPLACES THE FIRST VECTOR TABLE ENTRY */
    ALT_SHADOW_NON_PREEMPTIVE_INTERRUPT 256
    ALT_SHADOW_NON_PREEMPTIVE_INTERRUPT 256
    ALT_SHADOW_NON_PREEMPTIVE_INTERRUPT 256
    ALT_SHADOW_NON_PREEMPTIVE_INTERRUPT 256
```

これらのステップの完了後に、ソフトウェアをビルドして、実行します。そして、レポートされた割り込み時間を観察します。この例は変更されていない VIC Basic 例より、約 20 クロック・サイクル速いです。

いくつかの変化は、「リアルタイム・レイテンシの問題」で説明した可能な理由です。

詳細については、コード例を参照してください。

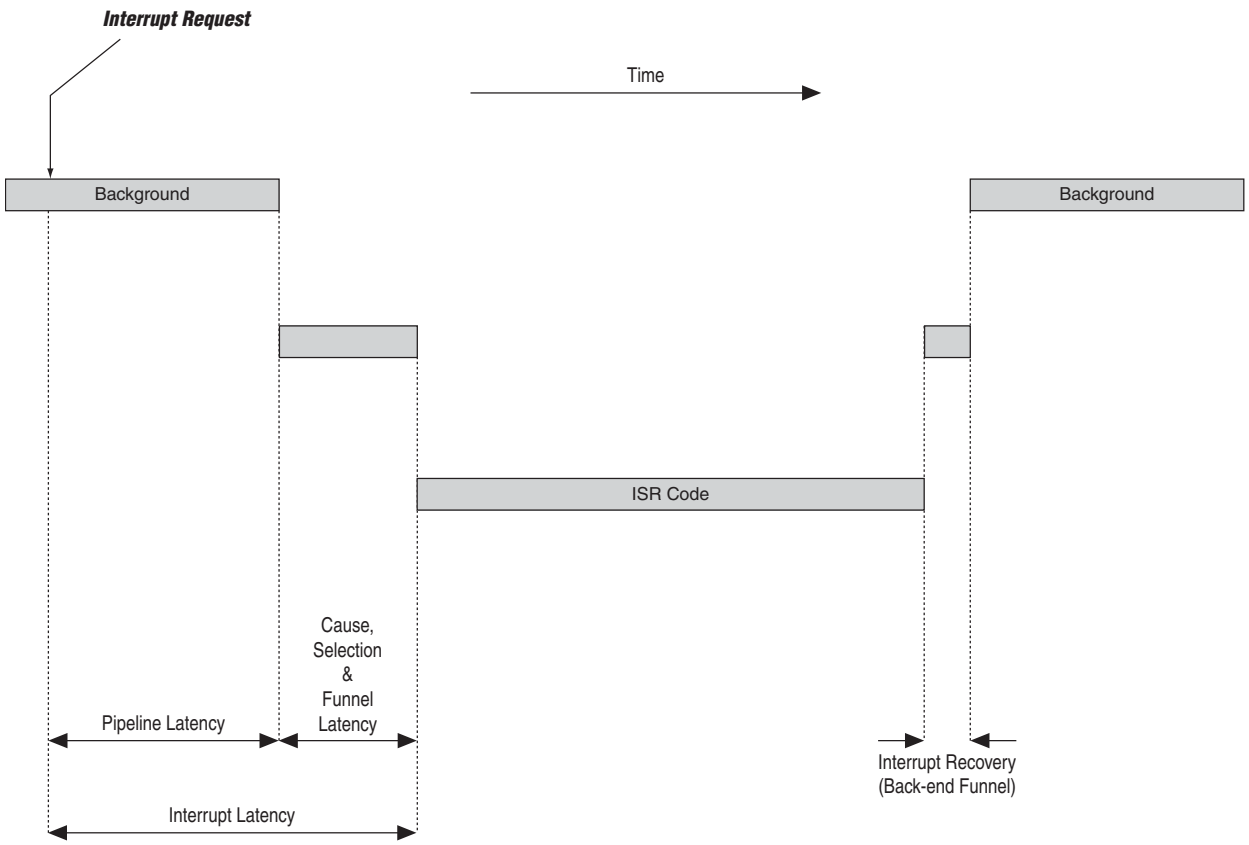
リアルタイム・レイテンシの問題

この項では、割り込みレイテンシの概観、割り込みレイテンシを決定するのに結合される要素、および測定する方法を提示します。割り込みレイテンシには、以下の要素を含みます。

- パイプライン・レイテンシ —「パイプライン・レイテンシ」
- 原因レイテンシ —「原因レイテンシ」に記載される
- 選択レイテンシ —「選択レイテンシ」に記載される
- 漏斗レイテンシ —「漏斗レイテンシ」に記載される
- コンパイラ関連のレイテンシ —19 ページの「コンパイラ関連のレイテンシ」に記載される

図 9 の割り込みレイテンシ図は、これらの要素を示します。

図 9. 割り込みレイテンシの要素



このセクションは、レイテンシの各要素をまとめて、レイテンシを測定する方法を説明します。付属のデザイン例は、タイミング測定のすべてをキャプチャするのに性能カウンタのコアを使用します。性能カウンタ・コアの使用については 19 ページの「性能カウンタによるレイテンシの測定」 で説明されています。

性能カウンタ・コアについて詳しくは、「Quartus II ハンドブック Volume 5: エンベデッド・ペリフェラル」 *「Performance Counter Core」* の章を参照してください。

パイプライン・レイテンシ

パイプライン・レイテンシはアサートされる割り込み信号と例外ベクトルで最初の命令の実行の間のクロック・サイクル数として定義されます。プロセッサが実行しているメモリのタイプとハードウェアの他のマスターポートの影響に応じて、大きく異なる場合があります。理論的に、不正な動作をするマスター・ポートは、プロセッサをメモリにアクセスすることを妨げて、そしてプロセッサを停止させる場合、この時間は無限になる可能性があります。

原因レイテンシ

原因レイテンシは、プロセッサが例外をハードウェア割り込みとして識別するために必要な時間です。VIC などの EIC で、各ハードウェア割り込みにはソフトウェア例外ベクトル・アドレスから分離する専用の割り込みベクトル・アドレスがあるので、原因レイテンシはゼロになります。

選択レイテンシ

選択レイテンシは、トリガされる割り込みに応じて、システムが正しい割り込みベクトルにコントロールを転送するために必要な時間です。VIC コンポーネントを有する選択レイテンシは修理する割り込みの数に依存します。表 3 は、割り込み数の関数としてシングル VIC 上の選択レイテンシの概要を示します。

表 3. VIC レイテンシのコンポーネント

割り込みの総数	割り込み要求クロック遅延（クロック）	優先順位の処理クロック遅延（クロック）	ベクタ生成クロック遅延（クロック）	全割り込みレイテンシ（クロック）
1	2	0	1	3
2—4	2	1	1	4
5—16	2	2	1	5
17—32	2	3	1	6

漏斗レイテンシ

漏斗レイテンシは、割り込み漏斗がコンテキストを切り換えるのに必要な時間です。漏斗レイテンシは、レジスタの保存と復元、先取りの管理、およびスタック・ポインタの管理を含むことができます。漏斗レイテンシは以下の要因によって異なります。

- 個別の割り込みスタックが使用される
- 命令のロードと格納に必要なクロック・サイクルの数
- 割り込みが異なるレジスタ・セットに切り換える
- 割り込みが同じレジスタ・セット内で別の割り込みを先取りする
- レジスタ・セット内の先取りが許される

レジスタ・セット内の先取りは特別な注意が必要です。割り込みが同じレジスタ・セットに割り当てられた別の割り込みを先取りできる場合、HAL VIC ドライバは特別な漏斗コードを提供します。この場合、漏斗はレジスタの内容を保存と復元するために、追加のオーバーヘッドを発生します。BSP を作成する際に、VIC ドライバの `altera_vic_driver_enable_preemption_rs_<n>` 設定を使用することで、レジスタ・セット内の先取りを制御できます。

☞ `altera_vic_driver_enable_preemption_rs_<n>` 設定について詳しくは、「Quartus II ハンドブック Volume 5: エンベデッド・ペリフェラル」の *「Vectored Interrupt Controller Core」* の章における「アルテラ HAL のソフトウェア・プログラミング・モデル」を参照してください。

表 4 および表 5 に、様々なコンフィギュレーションに対する漏斗レイテンシを示します。

表 4. シングルのスタック HAL レイテンシ

漏斗タイプ	ロードまたは格納に必要なクロック・サイクル *(1)*	
	1	2
レジスタ・セット内の先取りがディセーブルされたシャドー・レジスタ・セット	10	13
レジスタ・セット内の先取りがイネーブルされたシャドー・レジスタ・セット	42 同じレジスタ・セット (sstatus.SRS=0)	64 同じレジスタ・セット (sstatus.SRS=0)
	26 異なるレジスタ・セット (sstatus.SRS=1)	32 異なるレジスタ・セット (sstatus.SRS=1)

表 4 の注:

(1) 密結合メモリで、Nios II プロセッサは 1 サイクル・サイクルにおけるロードを実行するか、または命令をストアすることができます。密結合なしのオンチップ・メモリで、プロセッサは 2 クロック・サイクルが必要です。

表 5. 個別の割り込みスタック HAL レイテンシ

漏斗タイプ	ロードまたは格納に必要なクロック・サイクル *(1)*	
	1	2
レジスタ・セット内の先取りがディセーブルされたシャドー・レジスタ・セット	11 別の割り込みを先取りしない (sstatus.IH=0)	14 別の割り込みを先取りしない (sstatus.IH=0)
	12 別の割り込みを先取りする (sstatus.IH=1)	15 別の割り込みを先取りする (sstatus.IH=1)
レジスタ・セット内の先取りがイネーブルされたシャドー・レジスタ・セット	42 同じレジスタ・セット (sstatus.SRS=0)	64 同じレジスタ・セット (sstatus.SRS=0)
	27 ■ 異なるレジスタ・セット (sstatus.SRS=1) ■ 別の割り込みを先取りしない (sstatus.IH=0)	33 ■ 異なるレジスタ・セット (sstatus.SRS=1) ■ 別の割り込みを先取りしない (sstatus.IH=0)
	28 ■ 異なるレジスタ・セット (sstatus.SRS=1) ■ 別の割り込みを先取りする (sstatus.IH=1)	34 ■ 異なるレジスタ・セット (sstatus.SRS=1) ■ 別の割り込みを先取りする (sstatus.IH=1)

表 5 の注:

(1) 密結合メモリで、Nios II プロセッサは 1 サイクル・サイクルにおけるロードを実行するか、または命令をストアすることができます。密結合なしのオンチップ・メモリで、プロセッサは 2 クロック・サイクルが必要です。

表 4 および表 5 に、最も低いレイテンシは以下の条件で発生することに注意してください。

- 異なるレジスタ・セット — シャドー・レジスタ・セット・スイッチ ; ISR は中断されるタスクと異なるレジスタ・セットで実行して、レジスタを保存と復元する必要はありません。
- レジスタ・セット内の先取り（ネスティング）がディセーブルされます。

逆に、最も高いレイテンシは以下の条件で発生します。

- 同じレジスタ・セット — シャドー・レジスタ・セットスイッチなし ; ISR は中断されるタスクと同じレジスタ・セットで実行して、漏斗コードはレジスタを保存と復元する必要があります。
- レジスタ・セット内の先取りがイネーブルされます。

これらの 2 つの重要な要素では、先取りはレイテンシに最大の影響を与えます。ディセーブルされた先取りで、他の要因に関係なく、非常に低いレイテンシが発生します。

コンパイラ関連のレイテンシ

GNU C コンパイラは ISR を含む多くの C 機能のためのプロローグとエピローグを作成します。プロローグとエピローグは C ランタイム環境のコンテキストを保存と復元するなどのハウスキーピング・タスクを処理するコードのシーケンスです。プロローグとエピローグの必要な時間はコンパイラ関連のレイテンシと呼ばれます。

C コンパイラは必要に応じてプロローグとエピローグを発生します。コンパイラの最適化がイネーブルされ、ルーチンがコンパクトである場合、プロローグとエピローグは通常、いくつかのローカル変数で省略されます。プロローグとエピローグが機能のアセンブリ・コードを検査することによって発生するかどうかを判断します。

通常、コンパイラ・レイテンシは割り込みサービス性能全体にマイナーな影響しか与えません。コンパイラ・レイテンシに関してを懸念すとき、2 つのオプションがあります。

- コンパイラによる最適化をイネーブルして、ISR を簡素化することによって、ローカル変数最小化します。
- ISR をアセンブリ言語に記述します。

性能カウンタによるレイテンシの測定

Altera Complete Design Suite は、迅速かつ正確に性能を測定することを可能にするツールを提供します。このドキュメントで含まれるすべての例は、性能カウンタのコンポーネントを使用して、割り込みレイテンシを測定します。

割り込みをサービスするために、使用した合計の時間の測定は、この例は以下のステップを実行します。

1. グローバル変数（`interrupt_watch_value`）を既知の値（`0xfeedface`）に初期化します。
2. タイマー割込みをセットアップするには、`0xfacefeed` に設定する `interrupt_watch_value` ISR を登録します。
3. タイマーを起動します。
4. `interrupt_watch_value` が `0xfacefeed` になるまで `while()` ループで待ちます。

5. while() ループを終了した直後に、パフォーマンス・カウンタを停止します。そして、クロック・サイクルを計算して、stdout に計算された値を表示します。

システム内のリアル・タイムの割り込みレイテンシを決定するために、同様な方法を使用できます。

ソフトウェア割り込みの使用

ソフトウェアは適切な VIC コントロールおよびステータス・レジスタ (CSR) に書き込むことで任意の VIC 割り込みをトリガできます。ソフトウェアはハードウェア割り込みソースに接続される割り込みだけではなく、ハードウェアに接続されない割り込み（ソフトウェアのみの割り込み）もトリガできます。

ソフトウェアからの割り込みはトリガデバッグに役立ちます。ソフトウェアは、ちょうど割り込みがトリガされたときに制御して、システムの割り込み応答を測定できます。

割り込みの優先するものを考え直すために、ソフトウェアのみの割込みを使用できます。最優先ハードウェア割込みに応じる ISR は、ハードウェアによって必要な最小の処理を実行できます。そして、優先順位の低いレベルでソフトウェアのみの割込みをトリガして、割込み処理を終了します。

次の機能は、ソフトウェア割り込みを管理するのに利用可能です。

- alt_vic_sw_interrupt_set()
- alt_vic_sw_interrupt_clear()
- alt_vic_sw_interrupt_status()

BSP を生成した後、これらの関数の実装は **bsp/hal/drivers/src/altera_vic_sw_intr.c** にあります。関数の詳しい説明は、「Quartus II ハンドブック Volume 5: エンベデッド・ペリフェラル」の *「Vectored Interrupt Controller Core」* の章を参照してください。

例 6 に、ソフトウェア割り込みを登録する方法を示します。

☞ SOFT_IRQ の割り込み番号の値を定義する必要があります。

例 6. ソフトウェア割り込みのレジスタ

```
alt_ic_isr_register(
    VIC1_INTERRUPT_CONTROLLER_ID,
    SOFT_IRQ,
    soft_interrupt_latency_irq,
    NULL, NULL)
```

比較のために、例 7 にタイマー割り込みのレジスタを示します。

例 7. タイマー割り込みのレジス（比較用）

```
alt_ic_isr_register(
    LATENCY_TIMER_IRQ_INTERRUPT_CONTROLLER_ID,
    LATENCY_TIMER_IRQ,
    timer_interrupt_latency_irq,
    LATENCY_TIMER_BASE,
    NULL);
```

以下のコードはソフトウェア割り込みを生成します。

```
alt_vic_sw_interrupt_set(VIC1_INTERRUPT_CONTROLLER_ID, SOFT_IRQ);
```

まとめ

本書では、VIC および Nios II EIC インタフェースの利点および使用について説明します。ハードウェアおよびソフトウェアの説明や使用例は、システムで VIC の使用を開始するのに十分な情報を提供します。高度なトピック・カバレッジは VIC で達成できる性能向上の増加方法を提供しています。

参考資料

このアプリケーション・ノートでは、以下のドキュメントを参照しています。

- Nios II プロセッサ・リファレンス・ハンドブックの *「Processor Architecture」* の章
- Nios II プロセッサ・リファレンス・ハンドブックの *「Programming Model」* の章
- Quartus II ハンドブック Volume 5: エンベデッド・ペリフェラルの *「Performance Counter Core」* の章
- Quartus II ハンドブックの *「Volume 4: SOPC Builder」* の章
- Quartus II ハンドブック Volume 5: エンベデッド・ペリフェラルの *「Vectored Interrupt Controller Core」* の章
- Nios II ソフトウェア開発ハンドブックの *「Exception Handling」* の章
- Nios II ソフトウェア開発ハンドブック *「HAL API Reference」* の章
- Nios Community Wiki (www.nioswiki.com)

改訂履歴

表 6 に、本資料の改訂履歴を示します。

表 6. 改訂履歴

日付およびドキュメント・バージョン	変更内容	概要
2009 年 11 月 v1.0	初版	—