AN90799

PSoC® 4 割り込み

作成者: Rajiv Badiger

関連プロジェクト: サンプル コード

関連パーツファミリ: すべての PSoC 4 パーツ

ソフトウェアバージョン: PSoC Creator™4.2 以降

このアプリケーションノートの最新バージョンを入手するには、http://www.cypress.com/AN90799 にアクセスしてください。

さらに多様なサンプル コードが必要な場合は、以下を参照してください。

増え続けている何百もの PSoC サンプルコードについては、サンプルコードの Web ページにアクセスしてください。ここで PSoC 4 ビデオライブラリを探索することもできます。

AN90799 は、PSoC 4 の割り込みアーキテクチャと PSoC Creator™での設定について説明しています。このドキュメントは、割り込みベースのプロジェクトを開発する際のガイドとして役立ちます。遅延、ベクター選択、割り込みコードの最適化、およびデバッグ手法などの高度な割り込みの概念についても解説しています。

目次

1 はじめに

割り込みはどのような組込みアプリケーションにとっても重要な要素です。これにより、CPU は常に特定のイベントをポーリングする必要から解放されます。CPU は、そのイベントが発生したときにのみ通知を受け取ります。PSoC などのシステムオンチップ (SoC) アーキテクチャでは、オンチップ周辺機器のステータスを CPU に伝えるために割り込みが頻繁に使用されます。

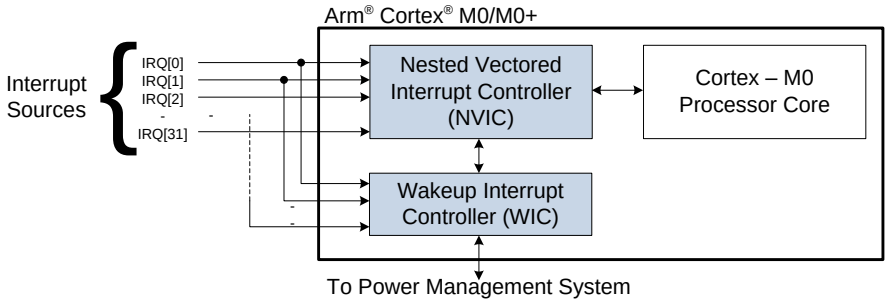
このドキュメントは、PSoC 4 割り込みアーキテクチャの説明から始まります。PSoC Creator IDE の割り込みサポートについて知りたい場合は、PSoC Creator での割り込みサポートにスキップしてください。サンプルコードの例については、サンプル コードを参照してください。割り込みプロジェクトをデバッグする場合は、割り込みの問題を見つけて解決するためのいくつかのヒントを紹介しているデバッグ ヒントセクションに移動してください。割り込みに関する高度なトピックのセクションでは、割り込みに関する高度なトピックについて説明します。

本アプリケーション ノートは PSoC および PSoC Creator IDE の経験者を対象としています。PSoC を初めて使用する場合は、*PSoC 4 入門* アプリケーションノート AN79953 で導入の確認や、および PSoC Creator のホームページの利用も活用してください。

2 PSoC 4 割り込みアーキテクチャ

図 1 は、PSoC 4 の割り込みアーキテクチャの簡略ブロックダイヤグラムを示します。

図 1. PSoC 4 の割り込みアーキテクチャ



IRQ[0]～IRQ[31]の最大 32 の割り込みラインがあり、それぞれが 0～3 の 4 段階の優先順位を持ちます。各割り込みラインには、割り込みベクターアドレスが割り当てられています。CPU は割り込み要求を受け取った後、このアドレスに分岐します。そこでは、割り込みサービスルーチン (ISR) と呼ばれる特別な関数が実行されます。

割り込み信号はネスト型ベクター割り込みコントローラー (NVIC) によって受信されます。割り込み信号がアクティブになると、NVIC は割り込み要求信号とともに割り込みベクターアドレスをプロセッサコアに送信します。それに対して、プロセッサコアは ISR に出入りするときに確認応答を送信します。NVIC はユーザー設定に基づいて、割り込みの有効化/無効化を担当します。また、複数の割り込み要求が同時に発生した場合には優先順位の整理を行い、重複した割り込みの中から優先度の低いものを保留して優先度の高いものを処理します。

ウェイクアップ割り込みコントローラー (WIC) ブロックにより、デバイスは割り込みを使用して低電力モード (スリープ、ディープスリープ、休止状態) からウェイクアップできます。NVIC、プロセッサコア、およびその他のデバイス周辺機器が停止している間も WIC ブロックはアクティブのままです。割り込みがトリガーされると、WIC は電源管理システムをアクティブにして NVIC とプロセッサコアを他の周辺機器とともに再起動します。NVIC が後を引き継ぎ、ベクターアドレスをプロセッサコアに送信して ISR を実行します。PSoC 4 デバイスにはウェイクアップに使用できるソースがいくつかあります。たとえば、図 1 は、NVIC とともに WIC にもルーティングされた IRQ[0]および IRQ[1]を示しています。これらは GPIO からの割り込みラインです。

PSoC 4 は割り込みに関する以下の特長があります。

- **構成可能な割り込みベクターアドレス:** 割り込みが発生すると、CPU 実行を任意の ISR コードに直接分岐できるため、レイテンシが短縮されます。
- **柔軟な割り込みソース:** 従来のマイクロコントローラーでは、割り込みソースは各割り込みラインに固定されていました。PSoC では各割り込みラインの割り込みソースを選択できる柔軟性があります。この柔軟なアーキテクチャによりデジタル信号はどれも割り込みソースに設定できます。

2.1 割り込みソース

PSoC 4 割り込みソースには 2 つのタイプがあります:

固定機能割り込みソース: これらはオンチップペリフェラルと対応する事前定義された割り込みソースのセットです。

ユニバーサルデジタルブロック (UDB) 割り込みソース (PSoC 4200、PSoC 42x7_BLE、PSoC 4200M、および PSoC 4200L パーツで利用可能) : UDB はタイマー、PWM、UART、SPI などのさまざまなデジタル機能の基本的な構成ブロックです。これは、プログラマブルロジック (PLD) 、データパス、および柔軟なルーティングで構成されています。固定機能割り込みソースとは対照的に、UDB で生成されたデジタル信号は割り込みをトリガーできます。信号は デジタルシステムインターコネクト (DSI) と呼ばれるルーティングファブリックを介して割り込みコントローラーにルーティングされます。詳細については、PSoC 4 テクニカルリファレンスマニュアルを参照してください。

表 1 に割り込みソースを示します。表に記載されている割り込みソースは、特に明記されていない限り、すべての PSoC 4 デバイスで利用できます。各割り込みソースの詳細については、表 1 にリストされている PSoC Creator コンポーネントデータシートを参照してください。付録 A は、デバイスに応じた割り込みソースの完全なリストです。

表 1. PSoC 4 割り込みソース

割り込みソース	詳細	
GPIO	各ポートは 8 つのピンで構成されています。各ピンは割り込みを生成できますが、ベクターアドレスはポートのすべてのピンに共通です。ファームウェアは、割り込みを引き起こしたピンを特定する必要があります。 PSoC 4 は、GPIO 信号の立ち上がりエッジ、立ち下がりエッジ、または両方のエッジで割り込みトリガーを有効にします。この割り込みにより、デバイスをスリープ、ディープスリープ、および休止状態モードからウェイクアップさせることができます。	
低電力コンパレータ (LPCOMP)	GPIO と同様に、コンパレータ出力信号の立ち上がりエッジ、立ち下がりエッジ、または両方のエッジで割り込みをトリガーできます。LPCOMP は、デバイスをスリープ、ディープスリープ、および休止状態モードからウェイクアップさせることもできます。LPCOMP は PSoC 4000 では使用できません。	
WDT	ウォッチドッグタイマー (WDT) は、デバイスをリセットしたり、割り込みを生成したりできるタイマーです。PSoC 4000、4000S、4100S、4100S Plus、および 4100PS デバイスには 16 ビットフリーランニング WDT がありますが、他の PSoC 4 パーツには 2 つの 16 ビット WDT と 1 つの 32 ビット WDT があります。WDT は、デバイスをスリープモードおよびディープスリープモードからウェイクアップできます。	
SCB	PSoC 4 には I²C、SPI、または UART として構成できるシリアル通信ブロック (SCB) が最大 5 つあります。SCB ブロックの正確な数は、デバイスファミリによって異なります。	
	I²C	割り込み生成イベント: アービトレーションロスト、スレーブアドレス一致、スタート/ストップ検出、バスエラー、バイト/ワード転送完了、TX FIFO がフルではない、TX/RX FIFO が空、RX FIFO が空ではない、RX FIFO オーバーラン、および RX FIFO がフル。スレーブアドレス一致イベントにより、デバイスをスリープモードおよびディープスリープモードからウェイクアップさせることができます。
	SPI	割り込み生成イベント: 転送完了、アイドル、TX FIFO がフルではない、TX/RX FIFO が空、バイト/ワード転送完了、RX FIFO が空ではない、フルの RX FIFO への書き込み、および RX FIFO フル。
	UART	割り込み生成イベント: 送信完了、スマートカードモードで UART TX が NACK を受信、LIN またはスマートカードモードでの UART アービトレーションロスト、フレームエラー、パリティエラー、LIN ボーレート検出完了、LIN 成功ブレーク検出。また、低電力モードからデバイスをウェイクアップすることもできます[1]。
システムパフォーマンスコントローラー (SPC)	SPC ブロックはフラッシュ書き込み操作を制御します。フラッシュへの書き込み操作完了時に割り込みがトリガーされます。	
SysTick	SysTick は、Arm® Cortex®-M0/Cortex M0+プロセッサに組み込まれた 24 ビットタイマーです。これは、通常、リアルタイムオペレーティングシステム (RTOS) によってティックタイマーとして使用されます。ただし、汎用タイマーとしても使用できます。詳細については、SysTick のセクションを参照してください。	
パワーマネージャー[2]	このブロックは、デバイスの電源電圧がしきい値を下回ると、低電圧検出 (LVD) 割り込みを生成します。	
SAR ADC	逐次比較型アナログデジタルコンバータ (SAR ADC) は、変換の終了、データオーバーフロー、スキャン衝突、データ飽和、およびデータオーバーレンジイベントで割り込みを生成できます。	
CapSense (CSD)	タッチアプリケーションに使用される CSD は、センサーのスキャン完了時に割り込みを生成します。	
タイマー、カウンター、パルス幅変調器 (TCPWM)	TCPWM ブロックは、16 ビットのタイマー、カウンター、または PWM として機能するように構成できます。最終カウント、入力キャプチャ信号、または「比較条件の成立」イベントで割り込みを生成できます。	
コントローラーエリアネットワーク(CAN)	PSoC 4200M および PSoC 4200L デバイスには 2 つの CAN ブロックがあります。PSoC 4100S Plus デバイスには 1 つの CAN ブロックがあります。CAN ブロックは、メッセージ受信、メッセージ送信、さまざまなエラーなどのイベントで割り込みを生成できます。詳細については、テクニカルリファレンスマニュアルの CAN の章を参照してください。	

割り込みソース	詳細
ダイレクトメモリアクセス (DMA)	PSoC 4100M/4200M、PSoC 4200L、PSoC 4100S Plus、および PSoC 4100PS デバイスは、周辺機器間でデータを転送する DMA を備えています。データ転送完了時に割り込みを生成できます。
ユニバーサル デジタルブロック(UDB)	タイマー、PWM、カウンター、UART などの UDB 実装は、固定機能の同等ブロックと同様に、さまざまなイベントで割り込みを生成できます。UDB は、PSoC 4200、PSoC 42xx_BLE、PSoC 4200M、および PSoC 4200L で利用できます。
USB	PSoC 4200L は、フレーム開始割り込みおよびデータエンドポイントを介した通信完了割り込みをサポートする USB を備えています。
電圧 DAC (VDAC)	これは、プログラム可能なアナログサブシステムの一部です。VDAC はコンパレータトリガーなどのイベント時、または VDAC 結果の準備完了時に割り込みを生成します。これは PSoC 4100PS でのみ利用可能です。
CTB/CTBm	連続時間アナログ機能を提供します。コンパレータ トリガーなどのイベントで割り込みを生成 します。
WCO WDT/WCO	PSoC 41000S および PSoC 4100S Plus は、WCO によってクロックされるタイマーを備えています。これらのタイマーは割り込みを生成できます。

(1) ピンの制限があります。すべてのポートに専用の割り込みがあるわけではありません。選択された UART ピンのポートに専用の割り込みがない場合、デバイスをウェイクアップできません。専用の割り込みを持つポートについては、デバイスアーキテクチャテクニカルリファレンスマニュアル (TRM) の「割り込み」の章を参照してください。
(2) PSoC 4000 / PSoC 4000S / PSoC 4100S/4100S Plus パーツでは利用できません

2.2 レベルトリガーおよびエッジトリガー割り込み

PSoC 4 は、割り込みのレベルおよびエッジトリガーをサポートします。図 2 は、トリガータイプを選択するロジックを示しています。このロジックは、NVIC がサポートする各割り込みラインに存在します。固定機能割り込みはレベルトリガーにのみ設定できますが、UDB を含む DSI ソースの場合、割り込みは立ち上がりエッジトリガーとレベルトリガーのどちらにでも設定できます。立ち上がりエッジ検出ブロックは、DSI 割り込み信号の立ち上がりエッジごとにパルスを生成します。NVIC がレベル設定およびエッジ設定の割り込みにどのように応答するかについては、タイミング図 (図 3 および図 4) を参照してください。

図 2. レベルトリガーとエッジトリガー

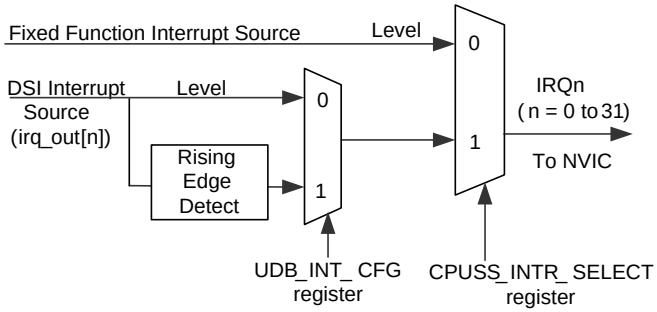


図 3. レベルトリガー割り込み

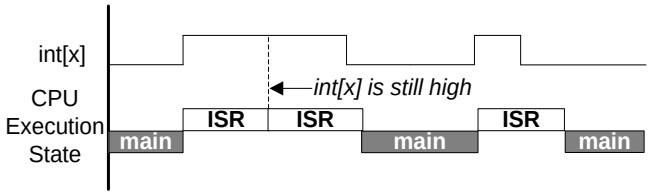
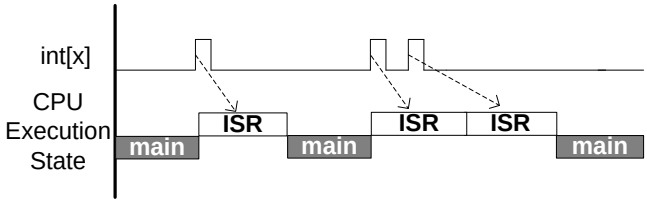


図 4. エッジトリガー割り込み



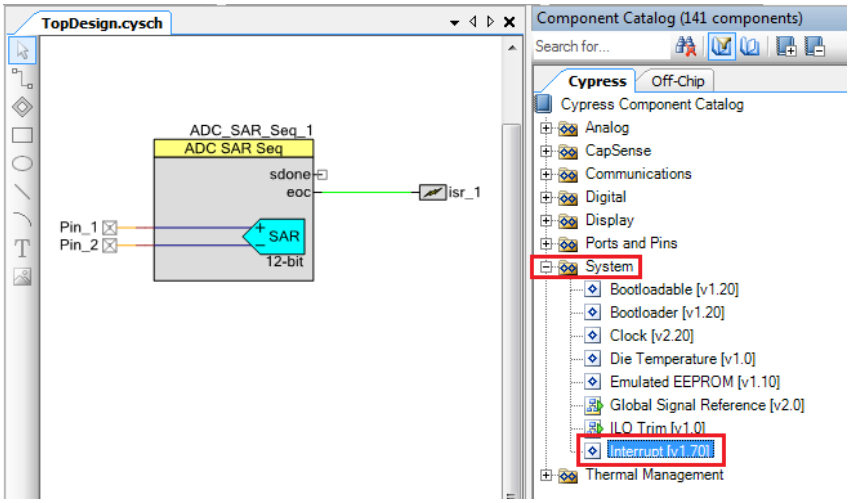
注: GPIO 割り込みロジックには、立ち上がりエッジ、立ち下がりエッジ、および両方のエッジでの割り込みをサポートする回路が追加されています。詳細については、PSoC 4 テクニカルリファレンスマニュアルを参照してください。

3 PSoC Creator での割り込みサポート

前のセクションでは、レベルまたはエッジトリガー、ベクターアドレス、割り込み優先度など、割り込みのいくつかのプロパティを設定する必要があることを示しました。PSoC Creator Interrupt コンポーネントによってこの設定を容易にします。このコンポーネントは、図 5 に示すように、Component Catalog ウィンドウの System タブにあります。

Interrupt コンポーネントの各インスタンスは、NVIC に送られる 32 ラインの割り込みラインのうち 1 つを使用します。図 5 に示す例では、SAR ADC からの変換終了 (eoc) 信号が Interrupt コンポーネント「isr_1」に接続されています。SAR ADC には、NVIC ベクターラインの一つが割り当てられています (付録 A を参照) 。たとえば、PSoC 4200 では、IRQ14 が SAR ADC 割り込みに割り当てられています。したがって、Interrupt コンポーネント「isr_1」は、図 2 に示すマルチプレクサロジックを介して eoc 信号を IRQ14 ラインに配線します。

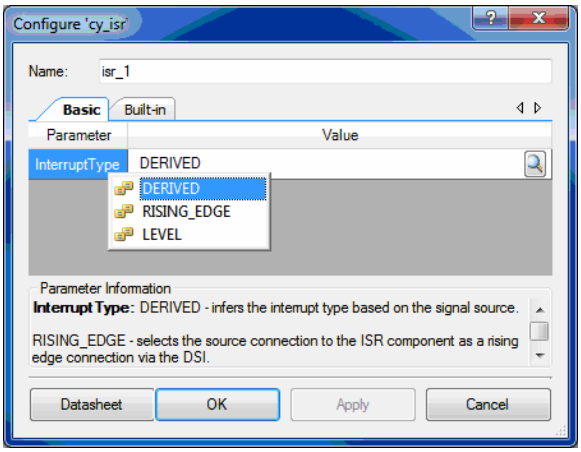
図 5. PSoC Creator Interrupt コンポーネント



3.1 Interrupt コンポーネントの設定

図 6 は、Interrupt コンポーネント設定ダイアログを示しています。コンポーネントには、DERIVED、RISING_EDGE、LEVEL の 3 つのオプションがあります。

図 6. Interrupt コンポーネントの設定

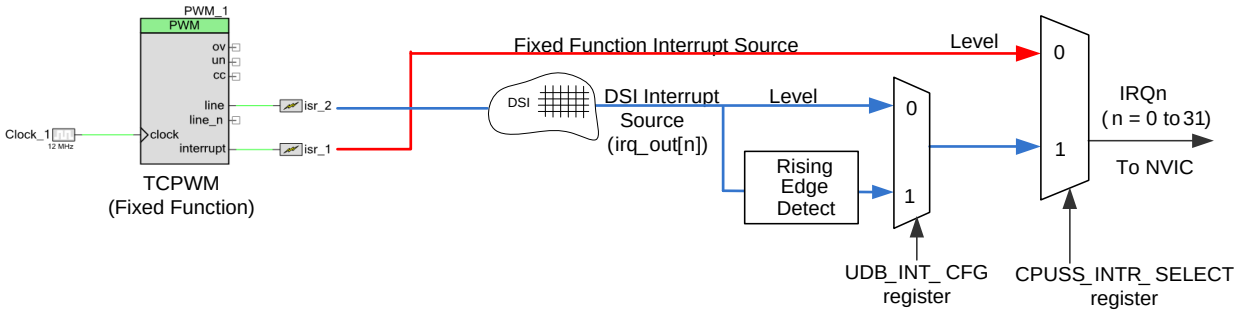


この設定は、図 2 に示すマルチプレクサを設定します。オプションの選択肢は、割り込みソース (固定機能または UDB/DSI) およびアプリケーション要件によって異なります。

固定機能ブロック: 固定機能ブロックからの割り込みラインは、図 7 の赤い線で示すように、常に「専用ルート」を介してルーティングされます。このパスに設定されている場合、割り込みはレベル トリガーとなり、ベクター番号は使用されているハードウェアブロックに基づいて決定されます。割り込みラインに接続された Interrupt コンポーネント (isr_1) は、レベル トリガーとしてのみ設定できます。RISING_EDGE トリガーに設定すると、ビルドエラーが発生します。DERIVED に設定されている場合、ツールは Level 割り込みのみを選択します。

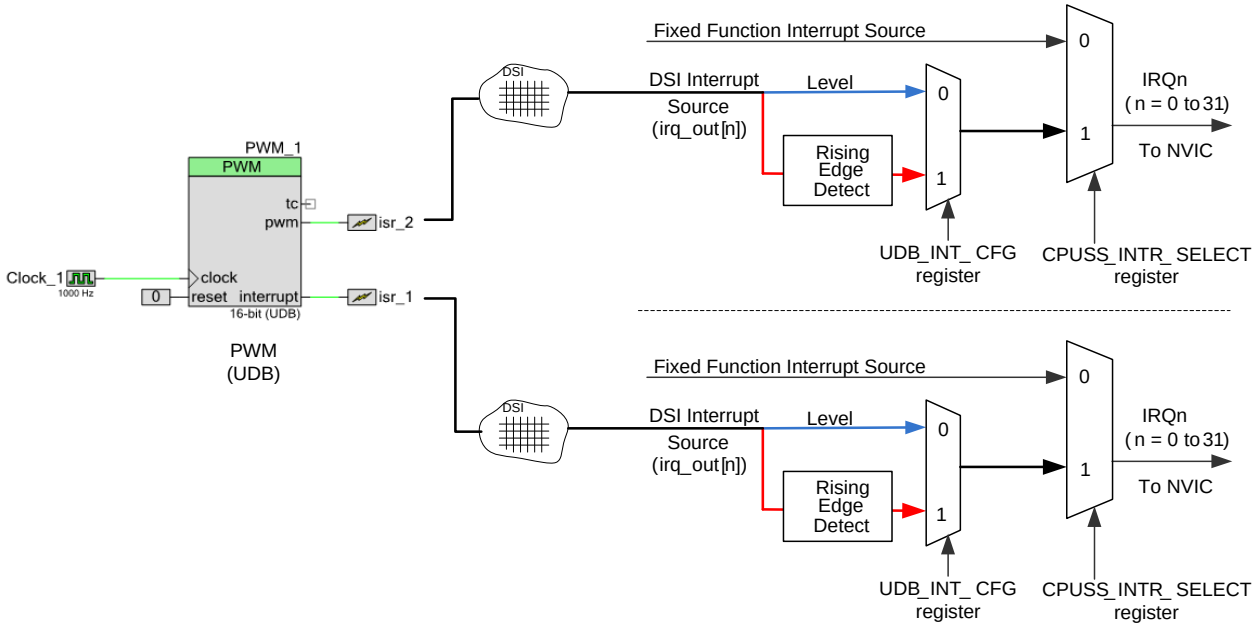
ただし、DSI を備えた PSoC パーツでは、固定機能ブロックからの他の出力信号を割り込みにルーティングできます。これにより、図 7 の青い線で示すように、PWM コンポーネントの line 出力に対して RISING_EDGE オプションを使用できます。PWM の出力に接続された Interrupt コンポーネント (isr_2) は、Level または RISING_EDGE に設定できます。DERIVED オプションが選択されている場合、ツールはレベル トリガー設定を選択します。このような場合のレベル トリガーは、信号が HIGH である限り ISR が繰り返し実行されるため、通常は役に立ちません。したがってほとんどの場合、RISING_EDGE が使用されます。

図 7. 固定機能ブロックの割り込みルーティング



UDB: UDB の場合、図 8 に示すように、DSI を使用して (UDB コンポーネントの割り込みラインまたは任意の出力から) 信号をマルチプレクサロジックにルーティングされます。したがって UDB からのすべての信号に LEVEL オプションと RISING_EDGE オプションの両方が使用できます。Interrupt コンポーネント (isr_1 または isr_2) で DERIVED オプションが選択されている場合、RISING_EDGE オプションが設定されます。これは、固定機能ブロック出力の DSI 信号ルーティングの場合とは対照的です。

図 8. UDB の割り込みルーティング



注: PSoC 4 BLE、PSoC 4200M、および PSoC 4200L パーツには 8 つの DSI チャネルがあり、各チャネルは **4 ライン**にデマルチプレクスされ、Arm Cortex-M0 プロセッサの 32 **(8x4)**の割り込みラインに割り当てられます。したがって、DSI 割り込みの最大数は、1 つのデザインにおいては **8** に制限されています。

表 2 は、Interrupt コンポーネントの InterruptType パラメーターを設定するためのガイドラインを示しています。

表 2. Interrupt コンポーネントの設定

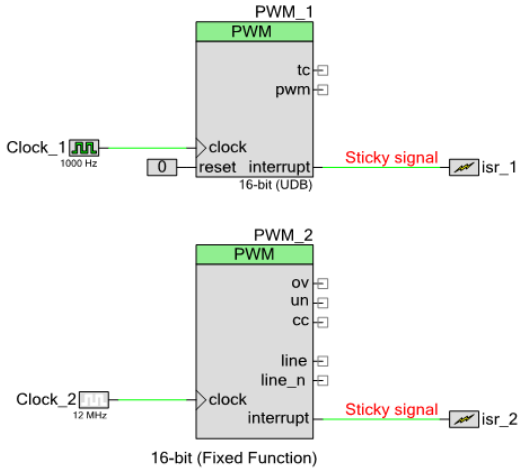
割り込みソース	信号	Interrupt コンポーネントの設定
固定機能	割り込み	LEVEL または DERIVED を選択します。RISING_EDGE は許可されません。
	ブロック出力	RISING_EDGE を選択します。それ以外の場合、信号が論理 HIGH 状態の間、割り込みが繰り返しトリガーされます。
UDB 関数	割り込み	RISING_EDGE または DERIVED を選択します。
	ブロック出力	RISING_EDGE を選択します。LEVEL を選択すると、信号が論理 HIGH 状態の間、割り込みが繰り返しトリガーされます。

3.1.1 スティッキービット

割り込み信号は「スティッキー」である場合があります。これは、割り込みラインが読み取られるかクリアされるまで、割り込みラインがアクティブ (HIGH) のままであることを意味します。ここで、Interrupt コンポーネントが RISING_EDGE に設定されている場合、ISR は 1 回実行されます。Interrupt コンポーネントが LEVEL に設定されている場合、ISR は繰り返し実行されます。これを処理するには、コンポーネントによって提供される API を使用して、割り込みソースをクリアします。割り込みソースのコンポーネントデータシートを参照してください。タイマー割り込みの使用例を示す、割り込みサービスルーチン (ISR) の作成のセクションを参照することもできます。

固定機能ブロックまたは UDB の出力ライン (たとえば、図 9 に示す PWM コンポーネントの「pwm」ライン) が割り込みラインの代わりに Interrupt コンポーネントに接続されている場合、割り込みをクリアします。ただし、Interrupt コンポーネントが LEVEL に設定されている場合、信号が HIGH である限り、ISR は繰り返し実行されます。

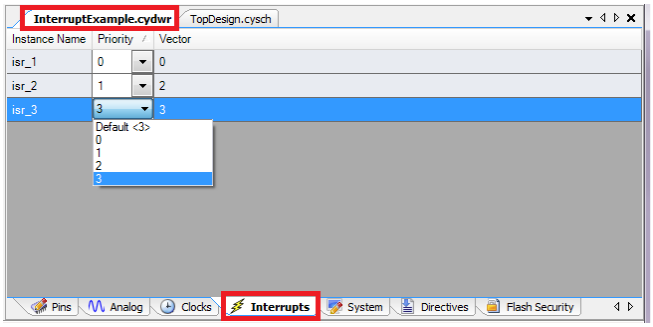
図 9. スティッキー信号



3.2 割り込み優先順位の設定

PSoC Creator プロジェクトの設計全体リソース ウィンドウ (*project_name.cydwr*) には Interrupts タブがあり、図 10 に示すように、Interrupt コンポーネントのインスタンス名、その優先順位、およびベクター番号が表示されます。isr_1、isr_2、isr_3 は、デザインで使用される Interrupt コンポーネントです。

図 10. *cydwr* ウィンドウの Interrupts タブ



割り込みの優先度を変更するためには、*cydwr* ウィンドウを使用してください。0 が最高の優先順位であることを覚えておいてください。そして最も低い優先順位が 3 です。Cortex-M0/Cortex M0+ CPU は割り込みネスティングをサポートしています。詳細については、ネストされた割り込みを参照してください。

各 Interrupt コンポーネントの割り込みベクター番号は、プロジェクトのビルド時に PSoC Creator によって自動的に割り当てられますが、手動で変更できます。詳細については、割り込みベクター番号のを参照してください。また、ベクター番号は*.cydwr* ウィンドウにオフセットとともに表示されることに注意してください。0 のベクター番号は、Cortex-M0/Cortex M0+ の例外番号 16 に対応します。Cortex-M0/Cortex M0+ 例外の概要については、例外を参照してください。

3.3 割り込み API 関数

PSoC Creator は、プロジェクトのコンポーネントごとに API (*.c* および*.h* ファイル) を生成します。これらの API には、各コンポーネントを構成して使用するための関数が含まれています。次の API 関数は、Interrupt コンポーネントに関連付けられています。

- `<instance_name> Start()` および `<instance_name>_Stop()`

  `Start()` は割り込みを有効にし、そのベクターをデフォルトの ISR に設定し、割り込み優先度を設定します。

  `Stop()` は割り込みを無効にします。

- `<instance_name>_StartEx()`

  `Start();` と同様 唯一の違いは、この関数はベクターアドレスを入力として受け取るため、コンポーネントによって生成されたデフォルトの ISR を使用するのではなく、カスタム ISR を記述できることです。

- `<instance_name> Enable()` および `<instance_name>_Disable()`

  これらの関数は、割り込みを有効または無効にするために Start()および Stop()によって内部的に呼び出されます。これらの関数を呼び出して、割り込みを動的に有効または無効にすることができます。

- `<instance_name> SetVector()` および `<instance_name> SetPriority()`

  これらの関数は `Start()` および `Stop()` によって内部的に呼び出され、割り込みベクターアドレスと割り込み優先度を設定します。これらの関数を呼び出して、ベクターと優先順位を動的に設定することもできます。これらの関数を呼び出す前に、割り込みが無効になっていることを確認してください。

- `<instance_name> SetPending()`

  割り込み要求なしで、つまりファームウェアで割り込みを発生します。

- `<instance_name> ClearPending()`

  保留状態の割り込みをクリアします。この関数は、割り込みソース信号には影響を与えませんのでソースの割込みがクリアされていない場合には引き続き割込みが発生します。

API の詳細な説明については、Interrupt Component Datasheet を参照してください。

3.3.1 クリティカルセクション制御関数

PSoC Creator は、*CyLib.h* および *CyLib.c* ファイルで一連の汎用割り込み関数も提供します。これらのファイルは、プロジェクトのビルド時に生成されます。重要なものは、`CyEnterCriticalSection` と `CyExitCriticalSection` です。これら 2 つの関数は、ファームウェア変数とハードウェアレジスタの内容破損を回避するために使用されます。`CyEnterCriticalSection` は割り込みを無効にし、割り込み状態値を返します。`CyExitCriticalSection` は、割り込み状態を復元します。

これがどのように機能するかを確認するために、タイマー制御レジスタへの書き込みの例を考えてみます。

```
TCPWM_BLOCK_CONTROL_REG |= TCPWM_MASK;
```

上記のステートメントの実行中に、次の一連の操作が発生します。

1. CPU は、TCPWM の制御レジスタを読み取り、一時レジスタに格納します。
2. CPU は、一時レジスタとそのマスク値の論理 OR 演算を実行します。
3. CPU は、論理 OR 演算の結果を制御レジスタにロードします。

手順 1 と 2 の間で割り込みが発生し、その ISR が新しい値を同じ制御レジスタにロードすることがあります。ISR の実行後、CPU が手順 2 の実行を再開すると、CPU は一時レジスタにあった古い制御レジスタ値を使用します。これにより、データが破損します。

この問題を回避するには、次のコードを追加します。

```
InterruptState = CyEnterCriticalSection();
TCPWM_BLOCK_CONTROL_REG |= TCPWM_MASK;
CyExitCriticalSection(InterruptState);
```

`CyEnterCriticalSection` 関数と `CyExitCriticalSection` 関数は、コントロールレジスタへの書き込み中の割り込みを無効にすることで問題を解決します。これらの関数は、共有されている変数またはレジスタを書き込むときに使用してください。

これらの機能の詳細については、システムリファレンスガイドを参照してください (PSoC Creator メニューの **Help** > **Documentation** からも利用できます) 。

4 割り込みサービスルーチン (ISR) の作成

ISR の記述方法を理解するために、例としてタイマー割り込みを考えます。図 11 に示すように、Interrupt コンポーネント「isr_1」は Timer_1 の interrupt 端子に接続されています。

プロジェクトをビルドした後、PSoC Creator は、図 12 のようにすべてのコンポーネントに関連付けられたファイルを生成します。 *isr_1.c* および *isr_1.h* は、Interrupt コンポーネント isr_1 用に生成されたファイルです。これらのファイルは、ISR を含むコンポーネントを設定および使用するための API を提供します。

図 11. タイマー割り込みの例

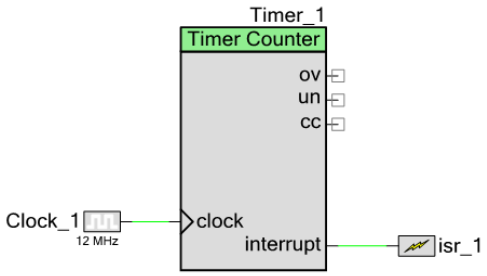
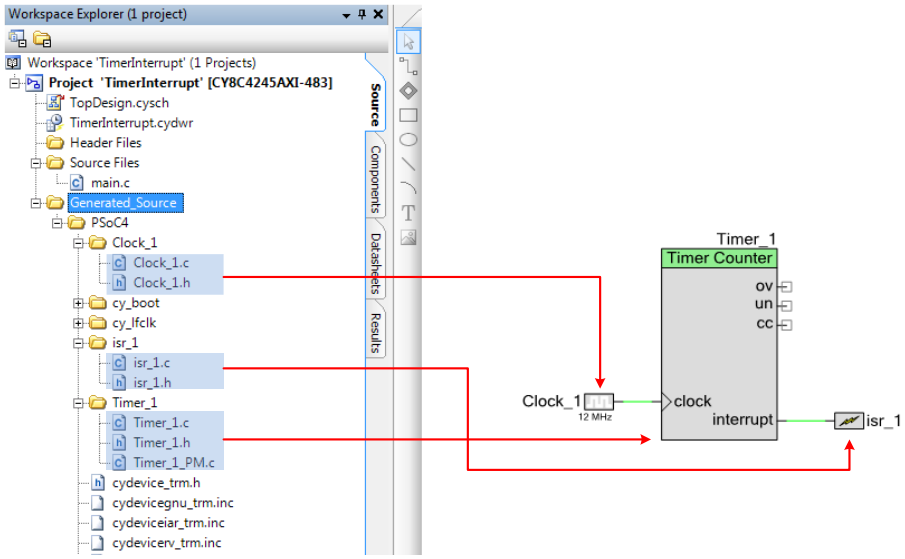


図 12. Interrupt コンポーネント用に生成されたファイル



ISR を作成する方法は 2 つあります。PSoC Creator で自動生成された ISR を使用する方法と、カスタム ISR 関数を作成する方法です。

4.1 自動生成された ISR の使用

以下は、*isr_1.c* にデフォルトで生成される ISR です。ISR 関数名は `CY_ISR(<isr_name>_interrupt)` の形式です。

```
CY_ISR(isr_1_Interrupt)
{
       #ifdef isr_1_INTERRUPT_INTERRUPT_CALLBACK
              isr_1_Interrupt_InterruptCallback();
        #endif /* isr_1_INTERRUPT_INTERRUPT_CALLBACK */

       /*  Place your Interrupt code here. */
       /* `#START isr_1_Interrupt` */
```

```
        /* `#END` */
}
```

この関数には 2 つの部分があります。1 つはコールバック関数を呼び出す部分で、もう 1 つはハンドラコードのプレースホルダーです。コールバック関数については、次のセクションで説明します。この自動生成された ISR の #START マーカーと#END マーカーの間にハンドラコードを記述できます。これらのマーカーの外側に記述されたコードは、プロジェクトファイルが再生成されるときに削除されることに注意してください。

割り込みを有効にするには、isr コンポーネントを起動します。以下は、割り込みソース、つまりタイマーと Interrupt コンポーネントを開始する *main.c* コードです。

```
int main()
{
       /* Start the timer component */
       Timer_1_Start();

       /* Start the interrupt component */
       isr_1_Start();

      /* Enable global interrupt */
      CyGlobalIntEnable;

       for(;;)
       {
          /* Place your application code here. */
       }
}
```

Interrupt コンポーネントを有効にすることに加えて、`CyGlobalIntEnable` マクロを使用してグローバル割り込みを有効にする必要があることに注意してください。ISR 内で、スティッキービットで説明されているように割り込みをクリアします。この例では、次の API 関数を使用してタイマー割り込みがクリアされます。

```
void Timer_1_ClearInterrupt(uint32 interruptMask)
```

`interruptMask` パラメーターは、Timer コンポーネントのターミナルカウント割り込みマスクまたは比較/キャプチャカウント割り込みマスクにできます。Timer コンポーネントデータシートまたは *timer_1.h* ファイルを参照してください。特定のコンポーネントからの割り込みをクリアする API および割り込みマスクについては、各コンポーネントのデータシートを参照してください。

4.1.1 extern キーワードの使用

多くの場合、自動生成された ISR では、変数にアクセスし、ユーザーソースファイルで定義されている関数を呼び出す必要があります。ただし、自動生成された ISR で変数と関数呼び出しを使用するには、isr_1 ファイルで宣言する必要があります。「extern」キーワードを変数宣言に使用します。

ファイルの冒頭で次のコードを探します。

```
/*******************************************************************************
*  Place your includes, defines and code here
********************************************************************************/
/* `#START isr_1_intc` */

/* `#END` */
```

#START マーカーと#END マーカーの間で変数と関数を直接宣言するか、宣言を含むヘッダーファイルを含めることができます。変数と関数宣言を使用した例を示します。

```
/*******************************************************************************
*  Place your includes, defines and code here
********************************************************************************/
```

```
/* `#START isr_1_intc` */

extern uint8 userVariable;
void userFunction(void);
/* `#END` */
```

4.2 コールバック関数の使用

自動生成された ISR でハンドラコードを記述する代わりに、ISR から独自の関数を呼び出すことができます。これにより、ユーザーコードと生成されたコードを分離することができます。

自動生成された ISR には、コールバック関数を呼び出すための、マクロによって制御される条件付きコンパイルのコードがあります。

```
CY_ISR(isr_1_Interrupt)
{
    #ifdef isr_1_INTERRUPT_INTERRUPT_CALLBACK
        isr_1_Interrupt_InterruptCallback();
    #endif /* isr_1_INTERRUPT_INTERRUPT_CALLBACK */

    /*  Place your Interrupt code here. */
    /* `#START isr_1_Interrupt` */

    /* `#END` */
}
```

デフォルトでは、`isr_1_INTERRUPT_INTERRUPT_CALLBACK` マクロは定義されていないため、`isr_1_Interrupt_InterruptCallback()` の呼び出しは無効になります。これはユーザーがソースファイルに記述する必要があるコールバック関数です。

1. コールバック関数を有効にします。これを行うには、プロジェクトの「Header Files」の下にある *cyapicallbacks.h* ファイルで `isr_1_INTERRUPT_INTERRUPT_CALLBACK` を定義します。また、次のように同じファイルでコールバック関数を宣言します。

```
#ifndef CYAPICALLBACKS_H
#define CYAPICALLBACKS_H


    /*Define your macro callbacks here */
    /*For more information, refer to the Writing Code topic in the PSoC
Creator Help.*/

    #define isr_1_INTERRUPT_INTERRUPT_CALLBACK
    void isr_1_Interrupt_InterruptCallback(void);

#endif /* CYAPICALLBACKS_H */
```

   自動生成された ISR で関数呼び出しが有効になっていることに注意してください。

2. 他の関数と同様に、コールバック関数をユーザーのソースファイルに記述します。

4.3 カスタム ISR の作成

ISR は、自動生成されたコードを変更する代わりに、完全に独自のソースファイルに記述することもできます。この方法には、コールバック関数の場合に発生する関数呼び出しの時間を節約できるという利点があります。独自の関数、例えば MyCustomISR を Interrupt コンポーネント *isr_1* の ISR にするには次のようにします。

1. CY_ISR_PROTO マクロを使用してカスタム関数を宣言します。
   CY_ISR_PROTO(MyCustomISR);

CY_ISR マクロを使用してカスタム関数を定義します。

```
CY_ISR(MyCustomISR)
{
  /* ISR code goes here */
}
```

main.c ファイルの起動コードで、isr_1_Start()の代わりに isr_1_StartEx()API 関数の呼び出しを追加します。isr_1_StartEx() API 関数は isr_1_Start()に似ていますが、isr_1_StartEx()には ISR 関数を引数にします：isr_1_StartEx(MyCustomISR);

```
CY_ISR_PROTO(MyCustomISR);
/*******************************************************************
* Function Name: MyCustomISR
*******************************************************************/
CY_ISR(MyCustomISR)
{
       /* Add code here */
}

int main()
{
       /* Start the timer component */
       Timer_1_Start();

       /* Set the custom ISR */
       isr_1_StartEx(MyCustomISR);

      /* Enable global interrupt */
      CyGlobalIntEnable;

      for(;;)
      {
           /* Place your application code here. */
      }
}
```

4.3.1 キーワード CY_ISR の重要性:

割り込みソースファイルは、CY_ISR マクロを使用して ISR 関数を定義します。このマクロは、自動生成された *cytypes.h* ファイルで定義されています。これは、PSoC 3 や PSoC 5LP などの他の PSoC デバイスファミリとの互換性とコードのポーティングを容易にするために使用されます。

同様に、CY_ISR_PROTO マクロは ISR 関数プロトタイプを宣言します。宣言は、Interrupt コンポーネントのヘッダーファイルにあります。たとえば、*isr_1* Interrupt コンポーネントのヘッダーファイル *isr_1.h* には、次の関数プロトタイプ宣言があります。

```
CY_ISR_PROTO(isr_1_Interrupt);
```

5 サンプル コード

表 3 に、割り込み機能を使用するサンプルコードのリストを示します。

表 3. 割り込みコードの例

サンプルコード	割り込みソース
CE210557 – PSoC 4 Timer Interrupt	タイマー
CE210558 – PSoC 4 GPIO Interrupt	GPIO
CE95915 – Implementing an RTC with PSoC® 4100/PSoC 4200 Devices	TCPWM
CE95333 – Low Power Comparator with PSoC 4	LPCOMP
CE95321 – Hibernate and Stop Power Modes with PSoC 4	LPCOMP, GPIO
CE95400 – Watchdog Timer Reset and Interrupt for PSoC 41xx/42xx Devices	WDT
CE95275 – Sequencing SAR ADC and Die temperature sensor with PSoC 4	SAR ADC
CE95298 – Switch Debouncer with PSoC 3/4/5LP	GPIO、デバウンサー
CE97089 – PSoC 4 ADC to Memory Buffer DMA Transfer	DMA
CE210741 – UART Full Duplex and printf Support with PSoC	UART

6 デバッグ ヒント

このセクションでは、割り込みプロジェクトのデバッグに関するヒントを提供します。以下は、頻繁に発生するケースの一部です。

a. 割り込みがトリガーされない

- 割り込みソースとグローバル割り込みが有効になっていることを確認してください。
- ベクターが正しい ISR に設定されているかどうかを確認してください。割り込みソースのハンドラを記述して割り当てる方法の詳細については、割り込みサービスルーチン (ISR) の作成を参照してください。
- 繰り返しトリガーされ、CPU 帯域幅全体を消費している他の割り込みソースがないかを確認します。
- 割り込みが 1 回だけトリガーされているかを確認してください。これは、Interrupt コンポーネントが立ち上がりエッジに設定されていて、割り込みソースがクリアされていない場合に発生します。

b. 割り込みが繰り返しトリガーされる

これは複数のケースで発生する可能性があります。

- ソースコンポーネントからの割り込みラインがレベルタイプに設定された Interrupt コンポーネントに接続されている場合。この動作を解決するには、割り込みソースをクリアします。
- コンポーネントからの割り込みラインではないデジタル出力がレベルタイプに設定された Interrupt コンポーネントに接続されている場合。Interrupt コンポーネントを立ち上がりエッジに設定して、立ち上がりエッジごとに 1 つの割り込みを取得します。

詳細については、スティッキービットを参照してください。

c. 割り込みが一度だけトリガーされる

ソースコンポーネントからの割り込みラインが立ち上がりエッジタイプに設定された Interrupt コンポーネントに接続されている場合。割り込みソースをクリアして、すべての立ち上がりエッジで割り込みがトリガーされるようにします。詳細については、スティッキービットを参照してください。

d. 割り込みサービスルーチン (ISR) の実行に予想よりも長い時間がかかる

これは、ISR の実行中に他の優先度の高い割り込みがトリガーされている場合に発生する可能性があります。他の割り込みソースと比較して、割り込みの優先度を上げます。

PSoC 4 デバイスには、シリアルワイヤーデバッグ (SWD) インターフェースを使用したオンチップデバッグ機能があります。ブレークポイントの追加、変数の評価と編集、CPU レジスタの表示、アセンブラー命令の監視、メモリの読み取りと書き込みを行うことができます。

デバッグモードは、以下のように割り込みをチェックするのに役立ちます。

- 割り込みが実行されているかどうかを確認するには、ISR のいずれかの命令にブレークポイントを追加します。
- デバッガーの Call Stack ウィンドウを使用して、割り込みがいつ実行されるかを確認します。また、これを使用して、優先度の低い ISR の実行中に優先度の高い割り込みが発生したかどうかを確認することもできます。

Breakpoint Hit Count を使用して、割り込みがトリガーされている回数を検出します。これは特に割り込み信号にグリッチがあり割り込みが複数回トリガーされていないかを確認するのに役立ちます。

デバッガーの使用方法の詳細については、PSoC Creator ヘルプの「Using the Debugger」セクションを参照してください。ドキュメントにアクセスするには、**F1** キーを押すか、PSoC Creator の **Help** > **Topics** を使用します。

デバッガーの代わりに、以下を行うことでピンをビットバンすることもできます。

- CPU が ISR に入っているかどうかを確認してください。
- ISR の実行時間を測定してください。これは、ISR の最初にピンをセットして、最後にピンをリセットすることでできます。

7 割り込みに関する高度なトピック

7.1 例外

例外はプロセッサに現在実行中のコードを一時停止させハンドラに分岐させるイベントです。割り込みは例外のサブセットです。表 4 に示すように、割り込みの他に、オペレーティングシステムアプリケーション用と障害処理用の例外があります。

表 4. Arm Cortex M0 の例外

例外	例外番号	割り込み優先度	変更内容
リセット	1	-3 (最高)	パワーオンリセットまたは外部リセットでトリガーされます。
ハードフォールト	3	–1	未定義のオペコードの検出などの障害状態で生成されます。
SVCall (スーパーバイザーコール)	11	プログラム可能	監視呼び出し (SVC 命令の実行) でトリガーされます。通常、オペレーティングシステムアプリケーションで使用されます。
PendSV (Pendable サービス呼び出し)	14	プログラム可能	SVCall に似ていますが、ハンドラへの分岐は、すべての優先度の高いタスクが完了した後でのみ行われます。
SysTick	15	プログラム可能	SysTick は、Cortex M0 に存在する 24 ビットのダウンカウントタイマーです。オペレーティングシステムアプリケーションで使用する定期的な割り込みを生成します。
IRQ0〜IRQ31	16–47	プログラム可能	外部 (ピン) または内部ペリフェラル割り込み。
リセット	1	-3 (最高)	パワーオンリセットまたは外部リセットでトリガーされます。

例外番号は Arm によって定義されていることに注意してください。PSoC Creator プロジェクトの *.cydwr* ウィンドウの **Interrupt** タブに表示される割り込みベクター番号には、例外オフセットが含まれています。たとえば、割り込みベクター 0 は例外番号 16 (IRQ0) です。

リセットは、デバイスで最も優先度の高い例外で、ハードフォールトがそれに続きます。これらには固定の優先順位がありますが、他の例外は優先順位をプログラム可能です。PSoC Creator は、すべての例外にデフォルトハンドラを提供します。リセットの場合、デフォルトのハンドラは *Cm0Start.c* ファイルの `Reset()` です。この関数は、起動時に最初に実行されます。他のすべての例外では、*Cm0Start.c* ファイルで提供される `IntDefaultHandler()` 関数がデフォルトのハンドラです。ただし (PSoC Creator コンポーネントまたはユーザーによって定義された) 割り込みを含む、使用される例外のベクターアドレスはプログラムの実行中にベクターテーブルにロードされます。未使用の例外は引き続きデフォルトのハンドラを使用します。

現在処理されている例外を特定するには、割り込みプログラムステータスレジスタ (IPSR) を読み取ります。これはデフォルトハンドラが実行中の場合に特に役立ちます。

例外の詳細については、http://infocenter.arm.com/help/index.jsp を参照してください。

7.2 割り込みレイテンシ

割り込みレイテンシは、割り込みのアサートから ISR 内の最初の命令の実行までの時間遅延として定義されます。PSoC 4 デバイスの Arm Cortex-M0 または Arm Cortex-M0+ プロセッサのレイテンシは、それぞれ 16 および 15 CPU クロックサイクル (ワーストケース) で、さらにペリフェラルと Cortex-M0/Cortex M0+ 割り込みライン間の同期による CPU サイクルが追加されます。表 5 は、DSI および固定機能ソース割り込みのさまざまな PSoC 4 ファミリにおける同期 CPU クロックサイクル遅延の一覧です。

表 5. DSI および固定機能ソース割り込みのシンクロナイザクロックサイクル遅延

デバイス	DSI 割り込み	固定機能割り込み
PSoC 4000	なし	周辺機器に依存: • SCB-I2C、GPIO、WDT: 3 CPU サイクル • SPC、CSD、TCPWM: 0 CPU サイクル
PSoC 4200 / PSoC 4100	0 CPU サイクル	3 CPU サイクル

デバイス	DSI 割り込み	固定機能割り込み
PSoC 42x7 BLE / PSoC 41x7 BLE	3 CPU サイクル	周辺機器に依存: • SCB-I2C、GPIO、WDT、CTBm、LPCOMP、BLE、LVD: 3 CPU サイクル • SPC、CSD、TCPWM、SAR: 0 CPU サイクル
PSoC 4200M / PSoC 4100M / PSoC 4200L	3 CPU サイクル	周辺機器に依存: • SCB-I2C、GPIO、WDT、CTBm、LPCOMP、LVD: 3 CPU サイクル • SPC、CSD、TCPWM、SAR、DMA、CAN、USB (PSoC 4200L でのみ利用可能) : 0 CPU サイクル
PSoC 4000S / PSoC 4100S / PSoC 4100PS	なし	周辺機器に依存: • SCB、GPIO、WDT、CTBm/CTB、LPCOMP: 2 CPU サイクル • CSD、TCPWM、SAR: 0 CPU サイクル
PSoC 4100S Plus	なし	周辺機器に依存: • SCB、GPIO、WDT、CTBm/CTB、LPCOMP: 2 CPU サイクル • CSD、暗号、CAN、SAR: 0 CPU サイクル

Cortex M0 の 16 サイクルのレイテンシまたは Cortex M0+ の 15 サイクルのレイテンシの期間に、以下の処理が行われます。

1. プロセッサは、現在のプログラムカウンター (PC) 、リンクレジスタ (LR) 、プログラムステータスレジスタ (PSR) 、およびいくつかの汎用レジスタをスタックにプッシュします。
2. プロセッサは NVIC からベクターアドレスを読み取り、PC に更新します。
3. プロセッサは NVIC レジスタを更新します。

したがって、ISR が実行中または開始間際の場合、レイテンシは Cortex M0 の 16 サイクルおよび Cortex M0+の 15 サイクルとは異なります。プロセスを効率的にするために、Cortex-M0/Cortex M0+ プロセッサは次の 2 つの機構を実装しています。

1. *Tail Chaining*: プロセッサが別の割り込みハンドラを実行しているときに割り込みが保留状態にある場合、最初の割り込みの実行が終了するとスタックの復帰が省略され、保留中の割り込みのハンドラがすぐに実行されます。これにより、レジスタをスタックから復帰してそれを再びスタックにプッシュする時間を節約できます。これは、優先度の低い割り込みの待ち時間短縮に有効です。
2. *Late Arrival*: 優先度の低い割り込みのスタック処理中に優先度の高い割り込みが発生した場合、プロセッサは優先度の低い割り込みハンドラではなく優先度の高い割り込みハンドラにジャンプします。プロセッサは、スタックプッシュ処理の最後に、優先度の高い割り込みのベクターアドレスを読み取ります。優先順位の高い割り込みハンドラの実行が完了すると、保留中の優先順位の低い割り込みハンドラのベクターアドレスがフェッチされて実行されます。これにより優先度の低い ISR に入ることと、レジスタ値をスタックにプッシュすることによって引き起こされる遅延が省略されて、優先度の高い割り込みのレイテンシが短縮されます。

割り込みがトリガーされたときに現在実行中の命令により、ISR の実行に遅延が追加されることに注意してください。デバイスが割り込みからウェイクアップする場合、電源投入シーケンス後の電圧安定化により、追加の遅延が発生します。仕様については、デバイスのデータシートを参照してください。

7.3 割り込みコードの最適化

割り込みベースのアプリケーションでは、ISR コード実行時間が重要な性能要件の 1 つです。アプリケーションによっては、割り込み要求を受信してから特定の時間内に ISR の重要なコードを実行する必要があります。また、割り込みの実行に時間をかけすぎて、メインコードの実行やその他の割り込みを停止させないようにする必要があります。これらの要件を満たすには、次のガイドラインを使用します。

- **ISR で長い関数を呼び出さないでください。**文字 LCD 表示ルーチンなどの機能は実行に時間がかかるため、他の優先度の低い割り込みの実行をブロックしてしまいます。

  推奨される方法は、緊急ではない関数呼び出しはメインコードに移動し、ISR ではフラグ変数のセットだけを行います。メインコードは定期的にフラグをチェックし、セットされている場合はそれをクリアして関数を呼び出します。
- **割り込みに適切な優先度を割り当ててください。**複数の割り込みがあるアプリケーションでは、より緊急性の高い割り込みに高い優先度を割り当てます。

7.4 割り込み内蔵コンポーネント

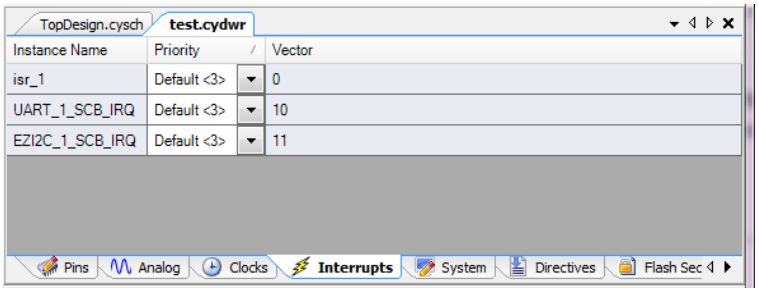
多くのPSoC Creatorコンポーネントには、実装の一部としてInterruptコンポーネントを内蔵しています。例としては、CapSense®、SAR ADC、EZI2C、セグメントLCDなどがあります。

Interruptコンポーネントと同様に、これらのコンポーネントの内部ISRは、ユーザーコードを記述するためのプレースホルダー領域を提供します。これらのコンポーネントでの割り込みの使用方法は PSoC Creator で提供される各コンポーネントのデータシートと関連するサンプルコードを参照してください。

7.5 割り込みベクター番号の制御

PSoC Creator は、プロジェクト内の Interrupt コンポーネントのベクター番号を自動的に割り当てます。プロジェクトをビルドした後、図 13 に示すように、*.cydwr* ウィンドウの Interrupts タブで割り当てられたベクター番号を確認できます。割り込み信号が DSI を経由する場合、割り込み信号に特定のベクター番号を選択することもできます。このセクションでは、これを行うための段階的な手順を示します。

図 13. *cydwr* ウィンドウの割り込みベクター番号



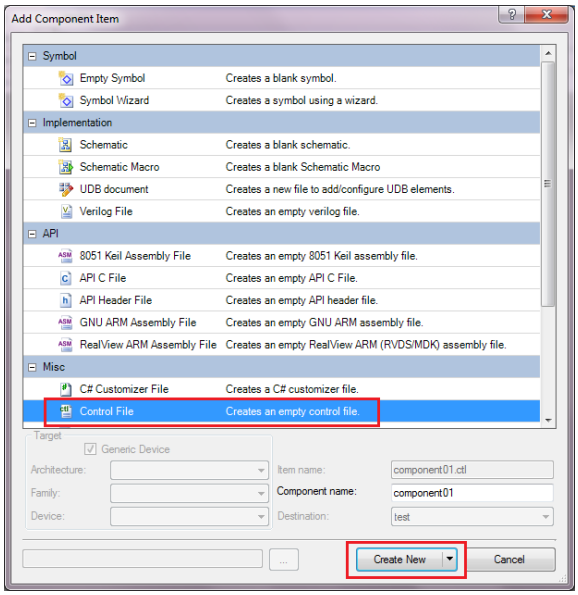
PSoC Creator によって割り当てられたベクター番号を上書きして手動でベクター番号を割り当てるには、*Control File* が使用されます。以下の手順に従ってください。

1. Workspace Explorer ウィンドウの **Components** タブをクリックしてください。

TopDesign コンポーネントを右クリックし、**Add Component Item…**を選択します。Add Component Item ダイアログが開きます。

図 14 に示すように、**Misc** グループまでスクロールして **Control File** を選択し、**Create New** をクリックしてください。

図 14. Control File 追加



TopDesign.ctl ファイルが作成され、Workspace Explorer ウィンドウに追加されます。

TopDesign.ctl ファイルをダブルクリックして編集用に開きます。*attribute* キーワードは、各 Interrupt コンポーネントの割り込みベクター番号を指定するために Control File で使用されます。割り込みベクター番号を指定する方法は、サンプル回路図に Interrupt コンポーネントを配置したか、回路図の PSoC Creator コンポーネントで内部的に使用されているかによって異なります。その 2 つの方法は以下のとおりです。

a) 回路図に配置した Interrupt コンポーネントの場合、構文は次のとおりです。

```
attribute placement_force of instance_name : label is "Intr(0, DesiredVectorNumber)";
```

ここで、instance_name は回路図の Interrupt コンポーネントの名前を示し、DesiredVectorNumber はベクター番号 (0～31) です。たとえば、ベクター17 を Interrupt コンポーネント *isr_1* に割り当てるには:

```
attribute placement_force of isr_1 : label is "Intr(0, 17)";
```

b) EZI2C など、内蔵の割り込みを使用するコンポーネントの構文は次のとおりです。

```
attribute placement_force of \top_instance_name : InternalInterruptName\: label is "Intr(0,
DesiredVectorNumber)";
```

ここで、top_instance_name は、内部割り込みを使用するコンポーネントの名前を指します。InternalInterruptName は、コンポーネントの内部割り込みに割り当てられた名前を参照します。これは、*cydwr* ウィンドウの Interrupts タブで確認できます。ここでは、割り込み名が上部のコンポーネントインスタンス名に追加されます。図 13 では、*SCB_IRQ* は EZI2C コンポーネントと UART コンポーネントの内部割り込み名です。次のステートメントは、EZI2C コンポーネントのベクターを 11 に割り当てます。

```
attribute placement_force of \EZI2C_1:SCB_IRQ\ : label is "Intr(0,11)"
```

割り込みベクター番号を割り当てた後、**Save** をクリックして、制御ファイルに加えた変更を保存します。

新しい割り込みベクターの割り当てを有効にするためのサンプルを **Clean and Build** します。*cydwr* ウィンドウの Interrupts タブに、変更された割り込みベクター番号の割り当てが表示されます。

7.6 SysTick タイマー

SysTick は、24 ビットのダウンカウントタイマーです。その割り込みは、通常、リアルタイムシステムでのタスク切り替えに使用されます。カウントには Cortex-M0/Cortex M0+ 内部クロックを使用します。SysTick は、以下の API を使用して設定されます。

1. 割り込みハンドラの設定

```
CyIntSetSysVector(SYSTICK_VECTOR_NUMBER, SysTick_ISR);
```

`SYSTICK_VECTOR_NUMBER` は、SysTick 割り込みの例外番号で、Cortex–M0 の場合は 15 です。`SysTick_ISR` は割り込みハンドラです。

割り込み周期の設定

```
(void)SysTick_Config(CLOCK_FREQ / INTERRUPT_FREQ);
```

`CLOCK_FREQ` は CPU クロック周波数です。`INTERRUPT_FREQ` は、SysTick から派生した割り込み率です。

以下は、SysTick タイマー使用のプログラムコードです。

```
#define SYSTICK_INTERRUPT_VECTOR_NUMBER 15u /* Cortex-M0/M0+ hard vector */

/* clock and interrupt rates, in Hz */
#define CLOCK_FREQ     24000000u
#define INTERRUPT_FREQ 2u

CY_ISR(SysTick_ISR)
{
    /* Interrupt Handler */
}

int main()
{
    /* Point the Systick vector to the ISR */
    CyIntSetSysVector(SYSTICK_INTERRUPT_VECTOR_NUMBER, SysTick_ISR);

      /* Set the number of ticks between interrupts */
    (void)SysTick_Config(CLOCK_FREQ / INTERRUPT_FREQ);

      /* Enable Global Interrupt */
    CyGlobalIntEnable;

    for(;;)
    {
    }
}
```

7.7 ネストされた割り込み

NVIC は、ソフトウェアのオーバーヘッドなしでネストされた割り込みを自動的に処理します。優先度の低い割り込みハンドラの実行中に優先度の高い割り込みがアサートされると、汎用レジスタの一部がスタックにプッシュされ、プロセッサーコアが NVIC からベクターアドレスを読み取って、優先度の高い割り込みハンドラにジャンプします。その実行が完了すると、プロセッサはレジスタ値を復帰して、優先度の低い割り込みの実行が再開されます。

7.8 GlobalSignal コンポーネントの使用

GlobalSignal コンポーネントは、システム内のさまざまなリソースからの割り込み信号へのアクセスを助けます。アクセスされる割り込み信号には、ウォッチドッグタイマー割り込み、統合ポート割り込み、統合低電力コンパレータ割り込み、システムパフォーマンスコントローラーインターフェイス (SPCIF) タイマー(フラッシュ書き込み操作の場合に使用) などがあります。使用可能な割り込み信号の詳細については、コンポーネントのデータシートを参照してください。統合ポート割り込みについては、以下で説明します。

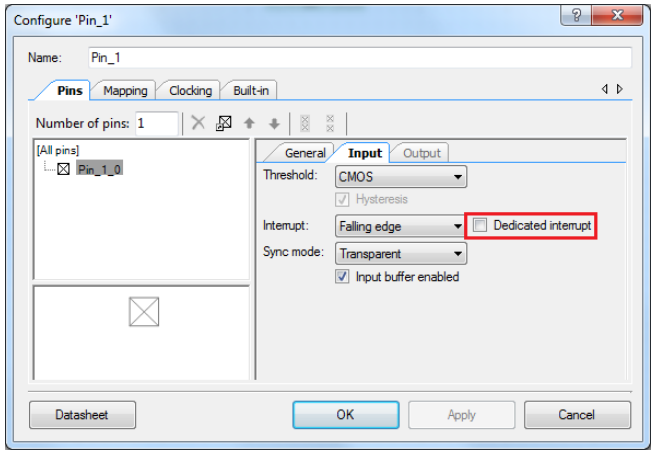
7.8.1 統合ポート割り込み

ほとんどの PSoC 4 デバイスでは、すべてのポートに専用の割り込みベクターがあるわけではありません (表 6 を参照) 。このような場合は、GlobalSignal コンポーネントのポート割り込み機能を組み合わせて使用することをお勧めします。統合ポート割り込みは、OR ロジックを使用してポート割り込み信号を統合します。

たとえば、PSoC 4100PS デバイスで、専用の割り込みがないピン P5[0]の信号から割り込みを生成する場合は、以下を実行します。

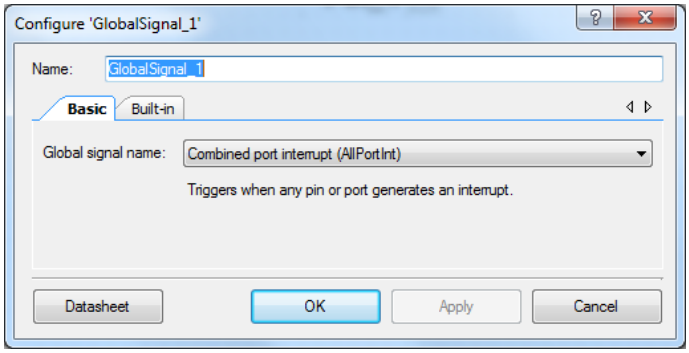
1. デジタル入力ピンコンポーネントを配置し、P5[0]に設定してください。
2. ピン割り込みを設定してください。専用割り込みがチェックされていないことを確認してください。

図 15. 統合割り込みのピンプロパティ



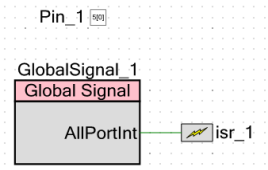
3. GlobalSignal コンポーネントを配置し、「Combined Port Interrupt (AllPortInt) 」に設定してください。

Figure 16. GlobalSignal コンポーネントの配置



4. Interrupt コンポーネントを GlobalSignal コンポーネントに接続してください。

図 17. Interrupt コンポーネントの GlobalSignal コンポーネントへの接続



これで isr_1 コンポーネント用に ISR を書き込むことができます。複数のポートピンが割り込みで有効になっている場合は、GPIO_PRTx_INTR レジスタを確認して、割り込みをトリガーしたポートピンを特定する必要があります。レジスタの詳細については、デバイスの *レジスタテクニカルリファレンス マニュアル (TRM)* を参照してください。

7.9 グローバル変数の揮発性の使用

割り込みでは一般的なケースとして、変数をフラグとして使用し割り込みハンドラで設定し、メインループでポーリングします。このような場合、コンパイラーはフラグを更新するコードがプログラムフローに存在しないと想定して、変数を最適化します。これにより、実行時にエラーが発生します。この問題を回避するには ISR とメインループの両方でアクセスされるグローバル変数を常に揮発性として宣言します。

8 まとめ

割り込みは組込みアプリケーションで一般的に使用されているものです。PSoC 4 などのシステムオンチップアーキテクチャの場合、割り込みはオンチップペリフェラルのステータスを CPU に伝達するという重要な役割を果たします。このアプリケーションノートでは、利用可能なインフラストラクチャを理解し、割り込みベースのプロジェクトを作成するために必要な情報を提供しました。

著者について

名前: Rajiv Badiger

役職: スタッフアプリケーションエンジニア

背景 電子通信工学、工学士

Appendix A. PSoC 4 の割り込みソースとベクター番号

表 6 は、PSoC 4 の 32 の割り込みベクターの割り込みソースの一覧です。

表 6. PSoC 4 割り込みソース (「–」は機能が利用できないことを示します)

固定機能割り込みソース	DSI 割り込みソース (not for PSoC 4000/4000S/ 4100S/4100S Plus/ PSoC 4100PS)	割り込みベクター								
		PSoC 4000	PSoC 4100/ 4200	PSoC 4 BLE	PSoC 4 M	PSoC 4 L	PSoC 4000S	PSoC 4100S	PSoC 4100S Plus	PSoC 4100PS
GPIO 割り込み–ポート 0	DSI	IRQ0	IRQ0	IRQ0	IRQ0	IRQ0	IRQ0	IRQ0	IRQ0	IRQ0
GPIO 割り込み–ポート 1	DSI	IRQ1	IRQ1	IRQ1	IRQ1	IRQ1	IRQ1	IRQ1	IRQ1	IRQ1
GPIO 割り込み–ポート 2	DSI	IRQ2	IRQ2	IRQ2	IRQ2	IRQ2	IRQ2	IRQ2	IRQ2	IRQ2
GPIO 割り込み–ポート 3	DSI	IRQ3	IRQ3	IRQ3	IRQ3	IRQ3	IRQ3	IRQ3	IRQ3	IRQ3
GPIO 割り込み–ポート 4	DSI	–	IRQ4	IRQ4	IRQ4	IRQ4	–	–	–	–
GPIO 割り込み–ポート 5	DSI	–	–	IRQ5	–	–	–	–	–	–
GPIO 割り込み–ポート 13 (USB ウェイクアップ)	DSI	–	–	–	–	IRQ5	–	–	–	–
GPIO 割り込み–すべてのポート*	DSI	–	–	IRQ6	IRQ5	IRQ6	IRQ4	IRQ4	IRQ4	IRQ4
–	DSI	–	IRQ5	–	–	–	–	–	–	–
–	DSI	–	IRQ6	–	–	–	–	–	–	–
–	DSI	–	IRQ7	–	–	–	–	–	–	–
LPCOMP (低電力コンパレータ)	DSI	–	IRQ8	IRQ7	IRQ6	IRQ7	IRQ5	IRQ5	IRQ5	IRQ5
WDT (ウォッチドッグ タイマー)	DSI	IRQ4	IRQ9	IRQ8	IRQ7	IRQ8	IRQ6	IRQ6	IRQ6	IRQ7
SCB0 (シリアル通信ブロック 0)	DSI	IRQ5	IRQ10	IRQ9	IRQ8	IRQ9	IRQ7	IRQ7	IRQ7	IRQ8
SCB1 (シリアル通信ブロック 1)	DSI	–	IRQ11	IRQ10	IRQ9	IRQ10	IRQ8	IRQ8	IRQ8	IRQ9
SCB2 (シリアル通信ブロック 2)	DSI	–	–	–	IRQ10	IRQ11	–	IRQ9	IRQ9	IRQ10
SCB3 (シリアル通信ブロック 3)	DSI	–	–	–	IRQ11	IRQ12	–	–	IRQ10	–
SCB3 (シリアル通信ブロック 4)	DSI	–	–	–	–	–	–	–	IRQ11	–
CTBm 割り込み (すべての CTBms)	DSI	–	–	IRQ11	IRQ12	IRQ13	–	IRQ10	IRQ12	–
CTB 割り込み	–	–	–	–	–	–	–	–	–	IRQ6

固定機能割り込みソース	DSI 割り込みソース (not for PSoC 4000/4000S/ 4100S/4100S Plus/ PSoC 4100PS)	割り込みベクター								
		PSoC 4000	PSoC 4100/ 4200	PSoC 4 BLE	PSoC 4 M	PSoC 4 L	PSoC 4000S	PSoC 4100S	PSoC 4100S Plus	PSoC 4100PS
BLE サブシステム割り込み	DSI	–	–	IRQ12	–	–	–	–	–	–
DMA 割り込み	DSI	–	–	–	IRQ13	IRQ14	–	–	IRQ14	IRQ12
SPCIF 割り込み	DSI	IRQ6	IRQ12	IRQ13	IRQ14	IRQ15	IRQ9	IRQ10	IRQ15	IRQ13
SRSS LVD 割り込み	DSI	–	IRQ13	IRQ14	IRQ15	IRQ16	–	–	–	–
SAR (逐次比較型 ADC)	DSI	–	IRQ14	IRQ15	IRQ16	IRQ17	–	IRQ19	IRQ25	IRQ15
CSD0 (CapSense)	DSI	IRQ7	IRQ15	IRQ16	IRQ17	IRQ18	IRQ10	IRQ13	IRQ16	IRQ14
CSD1 (CapSense)	DSI	–	–	–	IRQ18	IRQ19	–	–	–	–
VDAC 割り込み (両方の VDAC)	–	–	–	–	–	–	–	–	–	IRQ16
TCPWM0 (タイマー/カウンター/PWM0)	DSI	IRQ8	IRQ16	IRQ17	IRQ19	IRQ20	IRQ11	IRQ14	IRQ17	IRQ17
TCPWM1 (タイマー/カウンター/PWM1)	DSI	–	IRQ17	IRQ18	IRQ20	IRQ21	IRQ12	IRQ15	IRQ18	IRQ18
TCPWM2 (タイマー/カウンター/PWM2)	DSI	–	IRQ18	IRQ19	IRQ21	IRQ22	IRQ13	IRQ16	IRQ19	IRQ19
TCPWM3 (タイマー/カウンター/PWM3)	DSI	–	IRQ19	IRQ20	IRQ22	IRQ23	IRQ14	IRQ17	IRQ20	IRQ20
TCPWM4 (タイマー/カウンター/PWM 4)	DSI	–	–	–	IRQ23	IRQ24	IRQ15	IRQ18	IRQ21	IRQ21
TCPWM5 (タイマー/カウンター/PWM 5)	DSI	–	–	–	IRQ24	IRQ25	–	–	IRQ22	IRQ22
TCPWM6 (タイマー/カウンター/PWM 6)	DSI	–	–	–	IRQ25	IRQ26	–	–	IRQ23	IRQ23
TCPWM7 (タイマー/カウンター/PWM 7)	DSI	–	–	–	IRQ26	IRQ27	–	–	IRQ24	IRQ24
CAN0 割り込み	DSI	–	–	–	IRQ27	IRQ28	–	–	IRQ26	–
CAN1 割り込み	DSI	–	–	–	IRQ28	IRQ29	–	–	–	–
フレームの USB 開始	DSI	–	–	–	–	IRQ30	–	–	–	–
USB EP1-EP8 データ	DSI	–	–	–	–	IRQ31	–	–	–	–
暗号割り込み	–	–	–	–		–	–	–	IRQ27	–
WCO/WDT 割り込み	–	–	–	–	–	–	–	–	IRQ13	IRQ11
–	DSI	–	IRQ20	IRQ21	IRQ29	–	–	–	–	–
–	DSI	–	IRQ21	IRQ22	IRQ30	–	–	–	–	–

固定機能割り込みソース	DSI 割り込みソース (not for PSoC 4000/4000S/ 4100S/4100S Plus/ PSoC 4100PS)	割り込みベクター								
		PSoC 4000	PSoC 4100/ 4200	PSoC 4 BLE	PSoC 4 M	PSoC 4 L	PSoC 4000S	PSoC 4100S	PSoC 4100S Plus	PSoC 4100PS
–	DSI	–	IRQ22	IRQ23	IRQ31	–	–	–	–	–
–	DSI	–	IRQ23	IRQ24	–	–	–	–	–	–
–	DSI	–	IRQ24	IRQ25	–	–	–	–	–	–
–	DSI	–	IRQ25	IRQ26	–	–	–	–	–	–
–	DSI	–	IRQ26	IRQ27	–	–	–	–	–	–
–	DSI	–	IRQ27	IRQ28	–	–	–	–	–	–
–	DSI	–	IRQ28	IRQ29	–	–	–	–	–	–
–	DSI	–	IRQ29	IRQ30	–	–	–	–	–	–
–	DSI	–	IRQ30	IRQ31	–	–	–	–	–	–
–	DSI	–	IRQ31	–	–	–	–	–	–	–

改訂履歴

文書名: AN90799 –PSoC® 4 割り込み

文書番号: 001-90799

版	ECN	発行日	変更内容
**			本版は英語版 001-90799 Rev. *Eについて、CYPRESS DEVELOPER COMMUNITYの参画者によって日本語に翻訳されたドキュメントです。

セールス, ソリューションおよび法律情報

ワールドワイドな販売と設計サポート

サイプレスは、事業所、ソリューションセンター、メーカー代理店、および販売代理店の世界的なネットワークを保持しています。お客様の最寄りのオフィスについては、サイプレスのロケーションページをご覧ください。

製品

Arm® Cortex® Microcontrollers	cypress.com/arm
車載用	cypress.com/automotive
クロック&バッファ	cypress.com/clocks
インターフェース	cypress.com/interface
IoT (モノのインターネット)	cypress.com/iot
メモリ	cypress.com/memory
マイクロコントローラ	cypress.com/mcu
PSoC	cypress.com/psoc
電源用 IC	cypress.com/pmic
タッチセンシング	cypress.com/touch
USB コントローラー	cypress.com/usb
ワイヤレス	cypress.com/wireless

PSoC®ソリューション

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6 MCU

サイプレス開発者コミュニティ

コミュニティ | サンプルコード | Projects | ビデオ | ブログ | トレーニング | Components

テクニカルサポート

cypress.com/support

本書で言及するその他すべての商標または登録商標は、それぞれの所有者に帰属します。

Cypress Semiconductor
An Infineon Technologies Company
198 Champion Court
San Jose, CA 95134-1709