AN2094

PSoC® 1 - GPIO 入門

著者: Meenakshi Sundaram R

関連製品ファミリ: CY8C24x23A、CY8C24x94、CY8C21x34、CY8C20x34、CY8C21x23、CY8C21x45、CY8C22x45、CY8C27x43、CY8C28xxx、CY8C29x66、CY7C64215、CYWUSB6953

関連アプリケーション ノート: なし

本アプリケーション ノートの最新版または関連プロジェクト ファイルについては、http://www.cypress.com/AN2094 へアクセスしてください。

AN2094 は PSoC® 1 GPIO をお使いいただけるよう、GPIO 駆動モード、シャドー レジスタ、GPIO 割り込み等の汎用入出力 (GPIO) 関連のトピックについて説明します。またこの文書には PSoC 1 GPIO に関連付けられる他のリソースに関するヒントと簡単な説明も記載されています。

目次

1 概要

汎用入出力 (GPIO) は外部と MCU をつなぐ橋渡しとして機能し、マイクロコントローラー ユニット (MCU) の重要部分です。この橋渡しの種類と性質は最終アプリケーションによって異なります。PSoC は、従来の MCU と比べてより多くの機能を提供する、強力且つフレキシブルな汎用 I/O (GPIO) ピンを利用しています。

例えば ADC では GPIO がアナログ入力である必要があるのに対し、I^2C または SPI デジタル コミュニケーション ブロックでは同じ GPIO がデジタルである必要があります。この外部へのブリッジを正しくセットアップするには最終アプリケーションだけでなく使用される MCU の GPIO システムも知っておく必要があります。他のマイクロコントローラーのように PSoC には独自の GPIO システムがあります。

このアプリケーション ノートでは GPIO システムのアプリケーション固有のパラメータについて説明します。システムの詳細なテクニカル概要はデバイス個々のテクニカル リファレンス マニュアル (TRM) 内の PSoC コア セクションの General-Purpose I/O の章に掲載されています。

このアプリケーション ノートで説明する内容は以下のとおりです。

- **GPIO 駆動モード:** PSoC1 で利用可能な駆動モードのタイプ、各駆動モードの用法、ファームウェアでの駆動モードの動的再構成手順について説明します。
- **シャドー レジスタ:** GPIO シャドー レジスタの目的および用法を説明します。
- **GPIO 割り込み:** 割り込みを使用した単純な LED 点滅動作を例にとり、PSoC 1 での GPIO 割り込みを説明します。

この文書は読者が PSoC Designer™ IDE に精通していることを前提としています。

2 はじめに

サイプレスは、www.cypress.com に大量のデータを掲載しており、ユーザーがデザインに適切な PSoC デバイスを選択し、デザインに迅速かつ効率的に統合する手助けをしています。リソースの総合リストについては、知識ベース記事「How to Design with PSoC® 1, PowerPSoC®, and PLC – KBA88292」をご参照ください。以下は PSoC 1 のリソースの要約です。

- 概要: PSoC ポートフォリオ、PSoC ロードマップ
- 製品セレクタ: PSoC 1、PSoC 3、PSoC 4、PSoC 5LP
- さらに、PSoC Designer にはデバイス選択ツールが含まれています。
- アプリケーション ノート: サイプレスは、基本レベルから上級レベルまでの様々なトピックに触れる大量の PSoC アプリケーション ノートを提供しています。以下は PSoC 1 入門用の推奨アプリケーション ノートです。
  - AN75320 - PSoC® 1 入門
  - AN2094 - PSoC® 1 - GPIO 入門
  - AN74170 - PSoC® 1 のアナログ構造および PSoC Designer™を用いたコンフィギュレーション
  - AN2041 - PSoC 1® スイッチト キャパシタ アナログ ブロックについて
  - AN2219 - PSoC® 1 アナログ グランドおよびリファレンスの選択

注: CY8C29X66 デバイスに関するアプリケーション ノートについては、ここをクリックしてください。

- 開発キット:
  - CY3210-PSoCEval1 は CY8C25/26xxx デバイスを除いて、車載用デバイスを含むすべての PSoC 1 混在信号アレイ ファミリに対応しています。このキットには、LCD モジュールやポテンショメーター、LED、ブレッドボードが含まれています。
  - CY3214-PSoCEvalUSB は CY8C24x94 PSoC デバイス向けの開発用基板を備えています。この基板の特殊な特長には、USB および CapSense 開発とデバッグ サポートが含まれています。

注: CY8C29X66 デバイスに関する開発キットについては、ここをクリックしてください。

MiniProg1 および MiniProg3 デバイスは、フラッシュ メモリのプログラミングとデバッグ用のインターフェースを提供します。

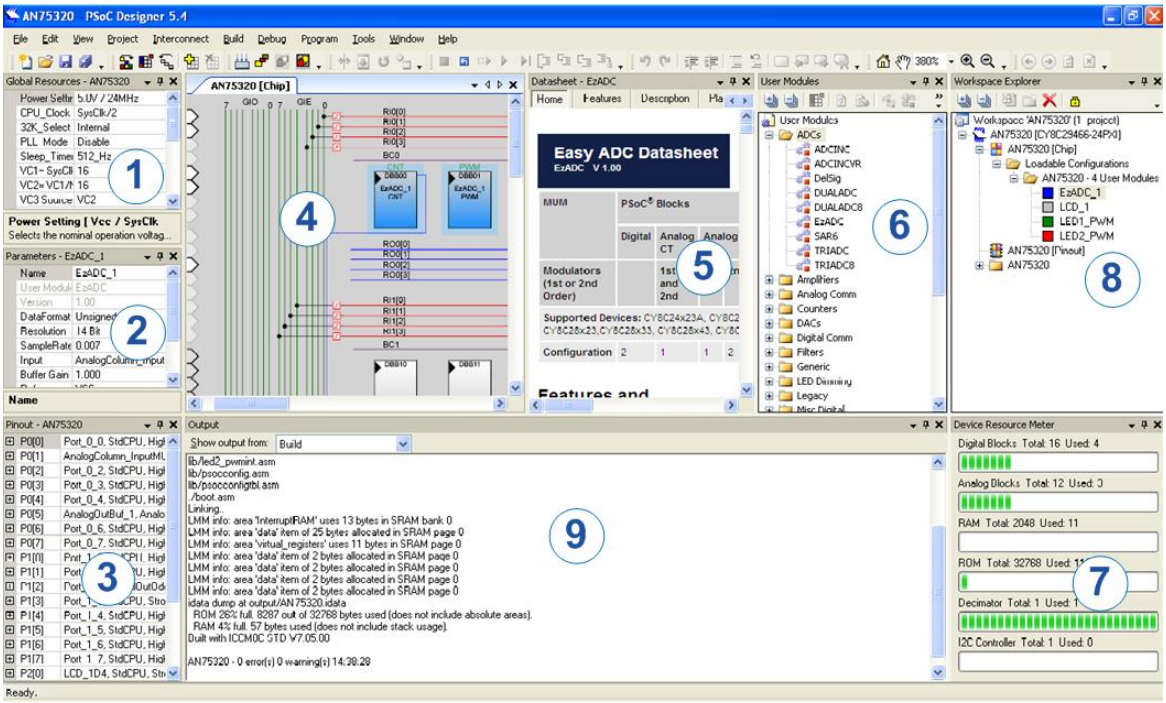
2.1 PSoC Designer

PSoC Designer は無料の Windows ベースの統合設計環境 (IDE) です。アプリケーション開発は、ドラッグ & ドロップの設計環境で特性化済みのアナログおよびデジタル ペリフェラルのライブラリを使用して行われます。また、API ライブラリ上の動的生成が行えるコードを活用して、設計をカスタマイズすることも可能です。図 1 に、PSoC Designer ウィンドウを示します。
注: これはデフォルト画面ではありません。

1. グローバル リソース – すべてのデバイス ハードウェアの設定。
2. パラメーター – 選択したユーザー モジュールのパラメーター。
3. ピン配置 – デバイスのピンに関する情報。
4. チップ レベル エディター – 選択したチップで使用可能なリソースの図。
5. データシート – 選択したユーザー モジュールのデータシート。
6. ユーザー モジュール – 選択したデバイスのすべての使用可能なユーザー モジュール。
7. デバイス リソース メーター – 現時点のプロジェクトのコンフィギュレーション用のデバイス リソースの使用率。
8. ワークスペース – プロジェクトに関するファイルを表示するツリー レベル図。
9. 出力 – プロジェクト ビルドおよびデバッグ処理からの出力。

注: PSoC Designer の詳細情報については、PSoC Designer IDE > **Help** > **Documentation** > Designer Specific Documents > IDE User Guide を順に選択して情報をご参照ください。

図 1. PSoC Designer のレイアウト



2.2 サンプル コード

以下のウェブページには PSoC Designer ベースのサンプル コードがリストアップされています。これらのサンプル コードは、空のページの代わりに完了した設計で始まり設計時間を短縮させることができ、PSoC Designer ユーザー モジュールが様々な用途にどのように利用できるかを示します。

http://www.cypress.com/documentation/code-examples/psoc-1-code-examples

PSoC Designer に統合されているサンプル コードへアクセスするには、図 2 に示すように **Start Page > Design Catalog > Launch Example** *Browser* を順に選択してください。

図 2. PSoC Designer のサンプル コード

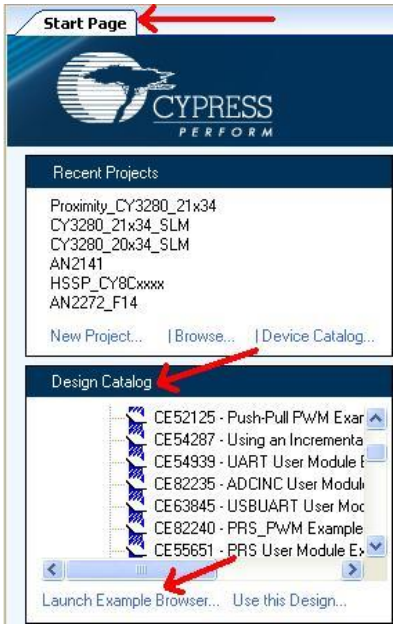
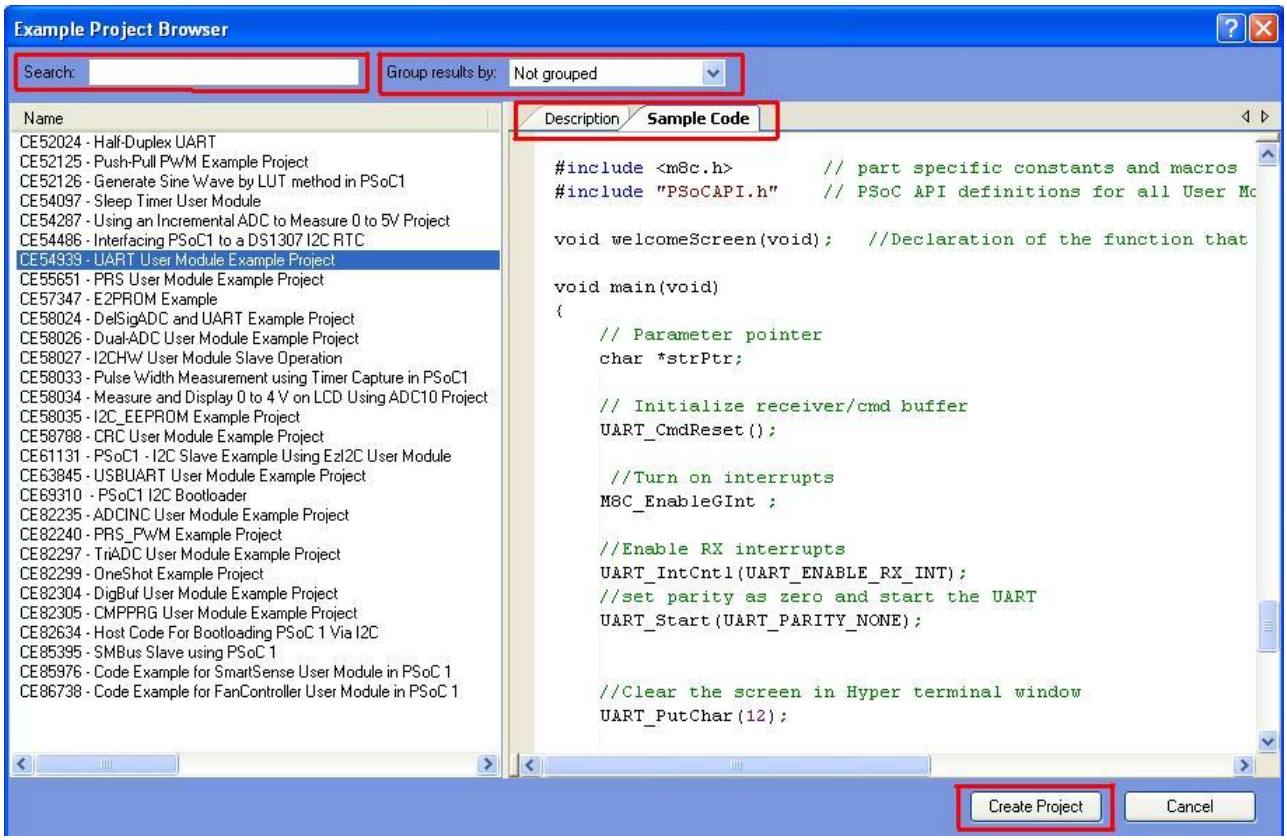


図 3 に示す Example Projects Browser の場合、以下のオプションがあります。

- プロジェクトをフィルタリングするためのキーワード検索。
- カテゴリ ベースのプロジェクト リスト。
- 選択したプロジェクトのデータシートのレビュー機能 (Description タブで)。
- 選択したプロジェクトのサンプル コードのレビュー機能。このウィンドウからコードをプロジェクトにコピー アンド ペーストして、コード開発時間を短縮させることができます。または、
- 選択に応じた新規プロジェクト (また、必要な場合は新規ワークスペース) の作成機能。完成した基本的な設計で開始して設計時間を短縮させます。そのようにして完成した設計をアプリケーションに適用できます。

図 3. サンプル プロジェクトおよびサンプル コード



2.3 PSoC Designer のヘルプ

PSoC Designer ホーム ページへアクセスして PSoC Designer の最新版をダウンロードしてください。次に、PSoC Designer を起動して次の項目に移動します。

- **IDE ユーザー ガイド**: **Help > Documentation > Designer Specific Documents > IDE User Guide.pdf** を選択します。このガイドは PSoC Creator プロジェクトを開発するための基礎知識を提供します。
- **シンプルなユーザー モジュールのサンプル コード**: **Start Page > Design Catalog > Launch Example Browser** を選択します。これらのサンプル コードは PSoC Designer ユーザー モジュールのコンフィギュレーションおよび使用方法を示します。
- **テクニカル リファレンス マニュアル**: **Help > Documentation > Technical Reference Manuals** を選択します。このガイドは PSoC デバイスのシステム機能を一覧にして説明します。
- **ユーザー モジュール データシート**: ユーザー モジュールを右クリックして「Datasheet」を選択します。このデータシートは選択されたユーザー モジュールのパラメーターおよび API を説明します。
- **デバイス データシート**: **Help > Documentation > Device Datasheets** を選択して特定の PSoC デバイスのデータシートへアクセスします。
- **Imagecraft コンパイラ ガイド**: **Help > Documentation > Compiler and Programming Documents > C Language Compiler User Guide.pdf** を選択します。このガイドは Imagecraft コンパイラ固有の命令と関数について説明します。

2.4 テクニカル サポート

ご質問は、弊社のテクニカル サポート チームが対応いたします。サイプレスのテクニカル サポート ページにアクセスし、お問い合わせ内容をケースとして作成し送信してください。

米国のお客様は、弊社フリーダイヤル (+1-800-541-4736) までお電話いただければ、弊社のテクニカル サポート チームより通話でご対応いたします。オプション「8」を選択してください。

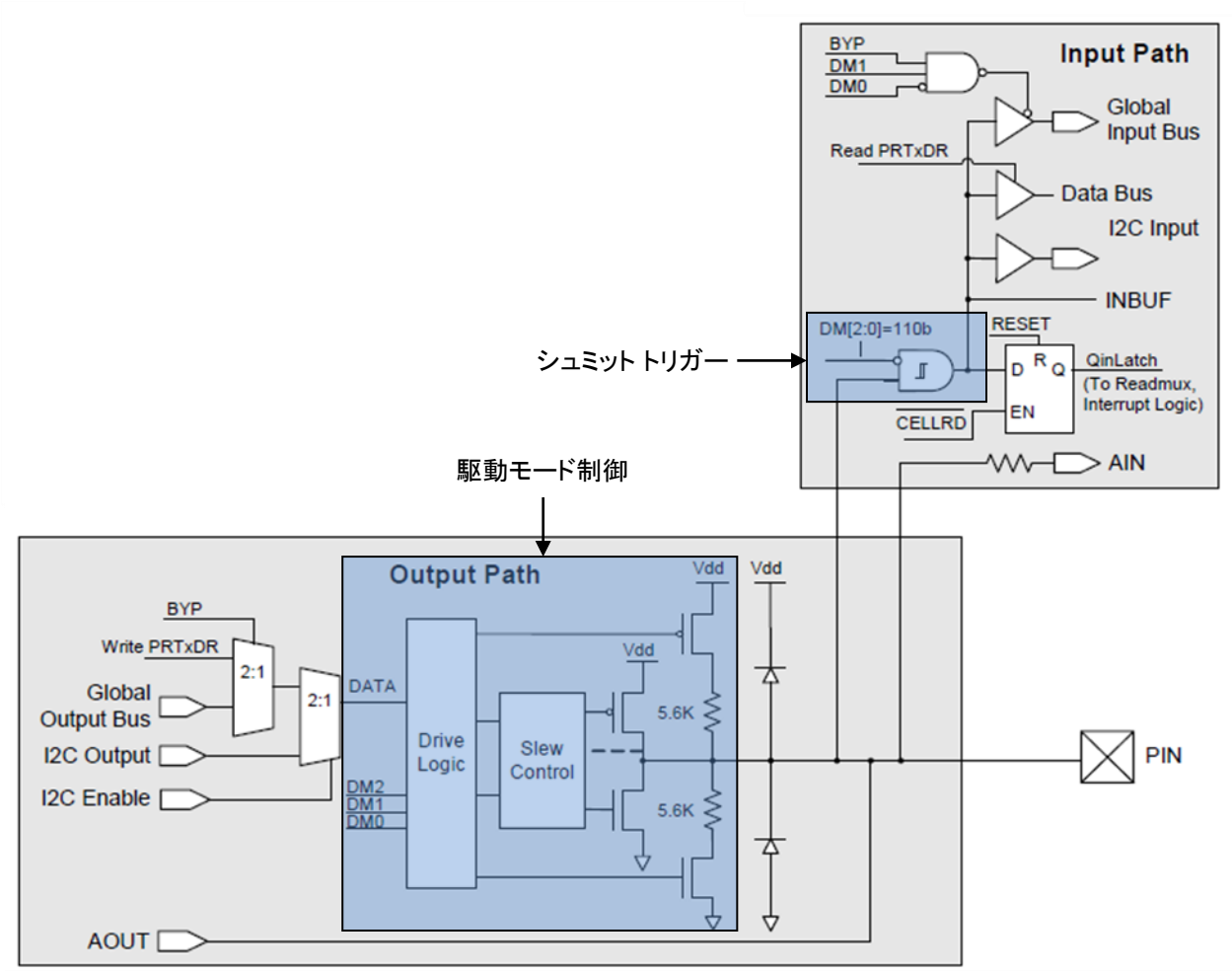
早急な対応が求められる場合には、下記の方法をご利用ください

- Self-help (セルフ ヘルプ)
- Local Sales Office Locations (販売事務所所在地)

3 GPO アーキテクチャ

PSoC1 の GPIO アーキテクチャを図 4 に示します。

図 4. PSoC 1 の GPIO セル アーキテクチャ



GPIO セルには入力パスと出力パスという 2 つの主な部分があります。

入力パスはデジタル信号用のピンと MCU 間のインターフェースとして動作するシュミット トリガー回路を備えています。シュミット トリガーの出力は MCU のデータ バス、グローバル入力バス、I²C 入力、割り込みコントローラー等に接続されます。またピンは PSoC デバイス内のアナログ ブロックにインターフェースで接続する内部アナログ バスに直接接続します。

出力パスはピンを 8 つの駆動モードのいずれかに構成できる駆動モード選択ロジックを持っています。駆動モード選択ロジックへの入力はデータ レジスタ、I²C バスおよびグローバル出力バス (デジタル ブロックの出力に接続される) を含んでいる内部データ バスから来ます。さらにアナログ出力バッファに接続する内部アナログ バスへの接続 (選択されるピン上) もあります。

4 GPIO 駆動モード

表 1 に示すように PSoC 1 は GPIO ピンを構成するための 8 つの駆動モードを提供しています。図 5 には駆動モードのそれぞれの GPIO セル コンフィギュレーションが示されています。

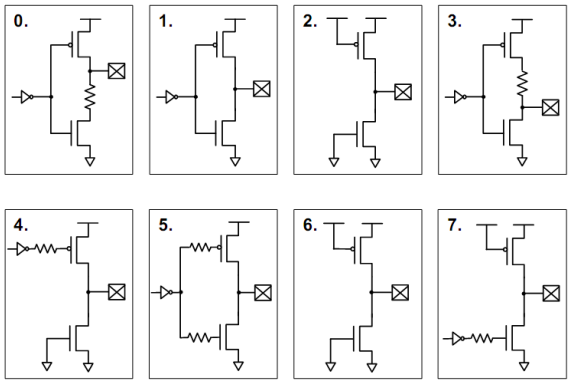
表 1. 駆動モードの詳細

Sl. No.	駆動モード	説明	応用
1	High-Z	高インピーダンスのデジタル入力モード。このモードではピンはデジタル入力として機能する。このモードではピンに「1 」か「0」を書き込んでも何の影響もない	ストロング駆動、プルアップまたはプルダウン抵抗、外部プルアップ抵抗に接続されるオープンコレクタ、または外部プルダウン抵抗に接続されるオープン エミッタを持つ信号ソースにインターフェースするデジタル入力ピンに使用
2	High-Z アナログ	高インピーダンスのアナログ モード。このモードではピンはアナログピンとして機能。このモードで GPIO セル内のデジタル入力回路 (図 1 のシュミット トリガー) が無効になることを除き、デジタル High-Z モードと同様。デジタル入力としてピンを使用している場合「HighZ アナログ」駆動モードではなく「HighZ」駆動モードを選択したことを確かめてください	アナログ入力ピン、およびアナログ入力および出力としての選択したピンに使用
3	オープン ドレイン HIGH (ODH)	このモードでは「1」を書き込むとピンが V_{DD} に駆動され、「0」を書き込むとハイインピーダンス状態となる。これはオープンエミッタコンフィギュレーションと同様	外部プルダウン抵抗に接続されるオープン エミッタを持つインターフェースを提供するために使用。この駆動モードは「Wired-OR」接続を実現
4	オープン ドレイン LOW (ODL)	このモードでは「0」を書き込むとピンが GND に駆動され、「1」を書き込むとハイインピーダンス状態となる。これはオープンコレクタコンフィギュレーションと同様	外部プルアップ抵抗に接続されるオープン コレクタを持つインターフェースを提供するために使用。このモードは「Wired-AND」接続を実現。例 – I^2C ピン
5	ストロング	このモードでは「1」を書き込むとピンが V_{DD} に駆動され、「0」を書き込むとそれが GND に駆動される	デジタル出力ピンに使用
6	プルダウン	このモードでは「1」を書き込むとピンが V_{DD} に駆動され、「0」を書き込むとそれが抵抗 (約 5.6kΩ) を介して GND に駆動される	オープンエミッタ出力を持つ信号へのインターフェース、または V_{DD} に接続されたスイッチへのインターフェースとして使用。電流吸い込み (シンク) モードでのインターフェース LED への出力として使用することができる
7	プルアップ	このモードでは「1」を書き込むとピンが抵抗 (約 5.6kΩ) を介して V_{DD} に駆動され、「0」を書き込むとそれが GND に駆動される	オープン コレクタ出力を持つ信号 (モーターからの回転速度計信号等) へのインターフェースまたは GND に接続されたスイッチへのインターフェースとして使用。電流吐き出し (ソース) モードでのインターフェース LED への出力として使用することができる
8	ストロング(低速)	このモードはストロング モードに似ているが、出力のスロープがわずかに制御される。よって出力切り替え時に高調波は現れない	不要輻射が低減されたデジタル出力として使用

図 5. 駆動モード コンフィギュレーションの詳細

Drive Modes

Drive Mode	Diagram Number	Data = 0	Data = 1
Resistive Pull Down	0	Resistive	Strong
Strong Drive	1	Strong	Strong
High Impedance	2	High Z	High Z
Resistive Pull Up	3	Strong	Resistive
Open Drain, Drives High	4	High Z	Strong (Slow)
Slow Strong Drive	5	Strong (Slow)	Strong (Slow)
High Impedance Analog	6	High Z	High Z
Open Drain, Drives Low	7	Strong (Slow)	High Z



GPIO は 2 つの方法で構成できます。1 つ目の方法は PSoC Designer のデバイス エディタで初期化の一部としてコンフィギュレーションを定義します。ピン コンフィギュレーションが常に固定されている場合にこの方法は便利です。2 つ目の方法はファームウェアでピンを構成する方法です。この方法では実行時に GPIO を柔軟に構成できます。

4.1 デバイス エディタ コンフィギュレーション

I/O ピンはデバイス エディタの Pinout ビュー モードを使用して構成することができます。Pinout ビュー モードでは PSoC Designer インターフェースの左下隅に表が表示されます (このウィンドウの位置はウィンドウの配置によって異なります) 。図 6 に表が示されています。

図 6. PSoC Designer の「Pinout」ウィンドウ (「駆動」リスト)

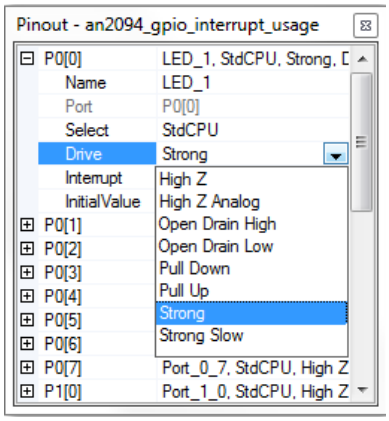


図 6 に示されている様々なフィールドの説明は以下のとおりです。

1. 「**Name (名前)** 」フィールドにはピン名が表示されます。ユーザーは目的をより明確にするためにピン名を変更することができます。ピン名を変更すると *PSoCGPIOINT.inc* および *PSoCGPIOINT.h* にピン データ レジスタ、ピン マスク、駆動モード レジスタ等のピンのマクロが生成されます。これは「ピン名のアサイン」で詳細に説明されています。
2. 「**Port (ポート)** 」フィールドにはピンの物理的なマッピングが示されます。このフィールドは編集できません。

3. 「**Select (選択)** 」フィールドは以下のようにピンの特殊な動作の一部を構成します。
   a. **AnalogInput:** ポート 0 とポート 2 のみに追加のアナログ入力およびアナログ出力オプションがあります。AnalogInput は外部からアナログ信号を取得し、アナログ カラム入力 MUX または PSoC ブロックに直接接続します。例えば ADC を使用する場合外部からアナログ信号を得るためには AnalogInput として少なくとも 1 つのピンを構成する必要があります。
   b. **AnalogOutputBuf:** デバイス ファミリに応じてポート 0 のいくつかのピンが内部アナログ出力バッファに接続されます。
   c. **Default:** グローバル バスが接続されておらず、駆動能力が High-Z アナログです。
   d. **StdCPU:** ポートを介した標準 I/O。これは PRTxDR データ レジスタを介して CPU[1]によって制御されます。
   e. **Global_IN、Global_OUT:** グローバル入出力はクロックおよびデータ信号を PSoC のデジタル ブロックへ送る機能を提供します。Global_IN (入力) または Global_OUT (出力) としてピンを設定するとピンはデジタルブロックとやり取りすることができます。例えば Global_IN が選択された場合はこの選択により Global_INPUT バスにその特定のピンが接続されます。このバスは PSoC のデジタル ブロックへの入力として使用されます。
   f. **AnalogMuxInput**[2]: これにより PSoC デバイス内の様々なアナログ ブロックにルーティングすることができるアナログ MUX バスへのピンの接続が可能になります。

      前述したピンの種類とは別に特別機能を持つピンがあり、それらは一覧化されています。例えば P1[0]および P1[1] は XtalOut と XtalIn を、P1[4]は ExtSysClk を、P1[5]および P1[7]は I2C_SDA と I2C_SCL を持っているというようになっています。
   g. 表 1 および図 5 で説明されているように 「**Drive (駆動)** 」 フィールドはピンの駆動モードを設定します。
   h. 「Pinout」ウィンドウ内の「**Interrupt (割り込み)** 」フィールドはピンの割り込みの種類を設定します。ピンには立ち上がりエッジ、立ち下がりエッジ、または両方の割り込みがある場合があります。「GPIO 割り込み」をご参照ください。
   i. 「Pinout」ウィンドウ内の「**Initial Value (初期値)** 」フィールドは起動時のピンの出力初期値を設定します。この値は自動的に生成されたブート コードの実行中にピンのデータ レジスタの移入によって課されます。
   j. **AnalogMUXBus**2[2]: チップ エディタにおいてピンの AMUX バスへの接続を有効／無効にします。ファームウェアで同じことを行う方法については「アナログ MUX (AMUX) バス制御レジスタ」をご参照ください)。

4.2 コード レベルの構成

I/O ピンを設定するもう 1 つの方法はアセンブリ言語または C 言語を使用してファームウェア[3]で関連するレジスタを直接変更する方法です。この方法ではプログラムの実行中に動的に I/O ポートを構成することができます。以下のレジスタは GPIO を制御します:

1. **PRTxDR レジスタ**: これはピンの出力状態を制御するデータ レジスタです。各ポートには関連するデータ レジスタが 1 個あり、そのレジスタの各ビットは 1 つのピンを表します。例えば PRT0DR はポート 0 を制御し、PRT0DR のビット#0 は P0[0]を制御します。GPIO ピンの状態は PRTxDR レジスタを使用して読み出すこともできます。「 ポートの読み出しおよび書き込み」をご参照ください。
2. **PRTxDMx レジスタ**: 各ポートには駆動モードを制御する 3 個のレジスタがあります。これらのレジスタに書き込むことで実行中にピンの駆動モードを動的に変更することができます。「GPIO ピンの駆動モードの変更」をご参照ください。
3. **PRTxICx レジスタ**: 各ポートにはピンの割り込みの種類 (立ち上がりエッジ、立ち下がりエッジ、状態の変化および割り込み無し) を制御する 2 つのレジスタがあります。これらのレジスタに書き込むことで実行中にピンの割り込みの種類を動的に変更することができます。「GPIO 割り込み」をご参照ください。

---

[1] このモードにおいてのみ CPU はピンの出力状態を制御することができます。つまりポート データ レジスタへのレジスタ書き込みがピン上で有効になります。

[2] CY8C21x34、21x45、22x45、24x94、28xxx デバイス ファミリのみで使用可能です。AnalogMuxBus フィールド (前述の各デバイスで利用可能) と連動して使用します。

[3] ピン コンフィギュレーションが固定されている場合はピン コンフィギュレーションにユーザー作成コードは必要ありません。PSoC Designer は自動的に起動コードを生成し、デバイス エディタの設定に従ってピンを設定します。

4. **PRTxGS レジスタ:** このレジスタは GPIO ピンのグローバル入力またはグローバル出力バスへの切断または接続を行うために使用されます。「GPIO グローバル選択レジスタ」をご参照ください。
5. **MUX_CRx レジスタ:** アナログ MUX バスを備えるデバイスではこのレジスタはピンのアナログ MUX バスへの切断または接続を行うために使用されます。各ポートは 1 個のレジスタで表され、レジスタ内の各ビットは対応するポート ピンを制御します。例えば MUX_CR0 レジスタのビット#0 は P0[0]を制御します。

4.3 ポートの読み出しおよび書き込み

PRTxGS レジスタ内の対応するビットをクリアする (または GPIO コンフィギュレーション ウィンドウでピンを StdCPU として構成する) ことでポート ピンがグローバル バスから切断され、ピンの駆動モードが *HighZ* または *HighZ* アナログでない場合ピンの状態は PRTxDR レジスタに書き込むことで制御できます。

ポート 0 データ レジスタ (PRT0DR、アドレス = バンク 0、00h)

ポート 1 データ レジスタ (PRT1DR、アドレス = バンク 0、04h)

ポート 2 データ レジスタ (PRT2DR、アドレス = バンク 0、08h)

ポート 3 データ レジスタ (PRT3DR、アドレス = バンク 0、0Ch)

ポート 4 データ レジスタ (PRT4DR、アドレス = バンク 0、10h)

ポート 5 データ レジスタ (PRT5DR、アドレス = バンク 0、14h)

ポート 6 データ レジスタ (PRT6DR、アドレス = バンク 0、18h)

ポート 7 データ レジスタ (PRT7DR、アドレス = バンク 0、1Ch)

特定のポート ピンに書き込むには対応するマスクおよびビット単位の「AND」または「OR」演算を使用します。例えば P0[0]のセットとクリアを行うには:

アセンブリ言語を使う場合:

```
or  reg[PRT0DR],  0x01 ; Set P0[0]
and reg[PRT0DR], ~0x01 ; Clear P0[0]
```

C 言語を使う場合:

```
PRT0DR |= 0x01; // Set P0[0]
PRT0DR &= ~0x01; // Clear P0[0]
```

ポート ピンから読み出すには PRTxDR レジスタを読み出して対応するビット マスクを使用します。例えば P0[1]の状態をチェックして P0[0]上の LED を更新するには:

アセンブリ言語を使う場合:

```
   mov A, reg[PRT0DR]
      and A, 0x02
      jnz PinHigh

      ; Code to process Pin cleared state
      and reg[PRT0DR], ~0x01 ; Set P0[0]
      jmp Exit

PinHigh:
      ; Code to process Pin set state
      or reg[PRT0DR], 0x01 ; Clear P0[0]

Exit:
```

C 言語を使う場合:

```
if (PRT0DR & 0x02)
{
   // Code to process Pin Set state
   PRT0DR |= 0x01; // Set P0[0]
}
else
{
   // Code to process Pin cleared state
   PRT0DR &= ~0x01; // Clear P0[0]
}
```

ポート ピンが GPIO コンフィギュレーション ウィンドウで意味のある名前を付けられた場合、PSoC Designer によって生成されたピン マクロを使用してデータ レジスタおよびピン マスクにアクセスすることもできます。「ピン名のアサイン」をご参照ください。

4.4 GPIO ピンの駆動モードの変更

すべてのポート ピンの駆動モードを設定する各ポートには 3 つのレジスタがあります。これらは PRTxDM0、PRTxDM1、PRTxDM2 レジスタで「x」はポート番号を表しています。これら 3 つのレジスタからの 1 ビットはそれぞれ一緒に特定のピンを構成します。例えば PRT0DM0、PRT0DM1、PRT0DM2 のビット 0 は P0[0]駆動モードを制御します。表 2 にコンフィギュレーションの詳細を示します。

表 2. 駆動モード レジスタ値

PRTxDM2[n]	PRTxDM1[n]	PRTxDM0[n]	駆動モード
0	0	0	抵抗プルダウン
0	0	1	ストロング駆動
0	1	0	高インピーダンス - デジタル
0	1	1	抵抗プルアップ
1	0	0	オープンドレイン - HIGH
1	0	1	低速ストロング駆動
1	1	0	高インピーダンス - アナログ
1	1	1	オープン ドレイン - LOW

表 2 では、「x」はポート番号に対応し、「n」は駆動モード レジスタのビットおよび構成されるポート ピンに対応しています。例えばポート 0 のピン 1 を抵抗プルダウンとして構成するには PRT0DM0、PRT0DM1、PRT0DM2 レジスタのビット「1」をクリアします。詳細についてはデバイスの TRM をご参照ください。

注意すべき重要点はすべての PRTxDM0 および PRTxDM1 レジスタはレジスタ バンク 1 (レジスタ バンクの詳細については TRM を参照) にあるのに対し、すべての PRTxDM2 レジスタはレジスタ バンク 0 にあるということです。この知識はアセンブリ言語で駆動モード レジスタを使用するために必要となります。ここではユーザーはレジスタにアクセスする前にレジスタ バンクを選択する必要があります。C ではコンパイラは使用するレジスタに基づいてバンクの割り当てを処理します。

アセンブリ言語を使う場合:

```
M8C_SetBank1
or   reg[PRT2DM0], 0x20
and  reg[PRT2DM1], ~0x20
M8C_SetBank0
and  reg[PRT2DM2], ~0x20
```

アセンブリ言語の例では最初の行が「1」にレジスタバンクを切り替える M8C_SetBank1 マクロへの呼び出しとなっています。PRT2DM0 および PRT2DM1 がレジスタ バンク 1 にあるためこの処理が行われます。「OR」命令および 0x20 のマスクを使

用して PRT2DM0 レジスタのビット 5 がセットされます。次に「AND」命令および 0x20 の逆数のマスクを使用して PRT2DM1 レジスタのビット 5 がクリアされます。

M8C_SetBank0 を使用してそれがレジスタ バンク「0」に再度切り替えられ、「AND」命令および 0x20 の逆数のマスクを使用してビット 5 または PRT2DM2 レジスタがクリアされます。「OR」および「AND」命令は読み出し、変更、または書き込み命令です。レジスタのコンテンツが最初に読み出され、「OR」または「AND」演算が値で行われ、その結果が同じレジスタに書き戻されます。この方法では他に影響を与えずに特定のビットを変更できます。

C 言語を使う場合:

```
PRT2DM0 |= 0x20;
PRT2DM1 &= ~0x20
PRT2DM2 &= ~0x20;
```

C ではバンクの切り替えが C コンパイラによって処理されるためコードははるかにシンプルとなります。「ビット単位の AND」(&=) または「ビット単位の OR」(|=) はレジスタの対応するマスクと併用される必要があります。

ポート ピンが GPIO コンフィギュレーション ウィンドウで改名された場合、PSoC Designer によって生成されたピン マクロを使用して駆動モード レジスタにアクセスすることもできます。「ピン名のアサイン」をご参照ください。

GPIO 駆動モードの変更はアプリケーション ノードのウェブ ページからダウンロードできる例 1 で示されています。このプロジェクト例の詳細については「サンプル プロジェクト」をご参照ください。

5. シャドー レジスタ

5.1 シャドー レジスタの使用

多くの設計では同じポートには入力ピン (例えば、プルアップ／プルダウン入力を持つスイッチ) だけでなく、出力ピン (例えば、LED) も入れることができます。このような設計では出力ピンを更新するために使用される命令は入力ピンを「1」または「0」に永久的にラッチすることができます。

例えば次のようなシナリオを考えてみましょう。P0_1 上のスイッチ入力がプルダウン モードで構成されます (スイッチは V_{DD} とピンの間に接続されます)。P0_0 上の LED 出力が P0_1 のスイッチ状態に従い、ピンはストロング駆動 (出力ピン) として構成されます。以下のコードはそれを実行します。

```
if (PRT0DR & 0x02)
{
    // Switch pressed. Turn on the LED
    PRT0DR |= 0x01; // Set P0[0]
}
else
{
    // Switch released. Turn off the LED
    PRT0DR &= ~0x01; // Clear P0[0]
}
```

では、スイッチが押されたと仮定すると LED をオンにする必要があります。「PRT0DR |= 0x01」コードは以下を実行する読み出し-変更-書き込み命令です。

1. 読み出し – PRT0DR = x x x x x x 1 0 (ビット 0 = 0 ➔ LED がオフになります ; ビット 1 = 「1」➔ スイッチが押される)
2. 変更 – (PRT0DR | 00000001) = x x x x x x 1 1
3. 書き込み – PRT0DR = x x x x x x 1 1 (ビット 0 = 1 ➔ LED がオンになります ; ビットを「1」にセットするとピンが内部で V_{DD} に接続されます。その理由はプルダウン駆動モードでピンに「1」を書き込むとそのピンが V_{DD} に接続されるからです)
4. ステップ 3 では「1」が P0[1]に書き込まれます。よってこのピンの状態は永久的に「1」に変更されます。これからスイッチが離されてもデータ レジスタはピンに「1」を駆動し続けてスイッチ入力イベント (スイッチが離された場合) の時にコードは常に「1」を読み出します。

このシナリオを克服するにはシャドー レジスタと呼ばれる変数がこのようなポート (入力と出力の組み合わせがあるポート) に使用されます。

シャドー レジスタを使用する場合はすべてのピンへの書き込みはこの変数を介して行われ、この変数は入力ピンの正確な状態で初期化される必要があります。このレジスタではプルアップまたはオープン ドレイン LOW 入力ピンの値が「1」に設定され、プルダウンまたはオープン ドレイン HIGH 入力が「0」に設定される必要があります。出力ピンの状態を変更しなければならない場合読み出し-変更-書き込み命令をシャドー レジスタで実行してからシャドー レジスタをポート データ レジスタにコピーする必要があります。すべての読み出しはポート データ レジスタ上で直接行われます。以下のコードはシャドー レジスタを実装します。

```
// Use 'extern' when using ShadowRegs UM
extern BYTE Port_0_Data_SHADE;

// Using shadow variables in code
        if(PRT0DR & 0x02)
{
        Port_0_Data_SHADE |= 0x01;
        PRT0DR = Port_0_Data_SHADE;
}

else
{
        Port_0_Data_SHADE &= ~0x01;
        PRT0DR = Port_0_Data_SHADE;
        }
```

ここで読み出し-変更-書き込み動作は以下のようになります。

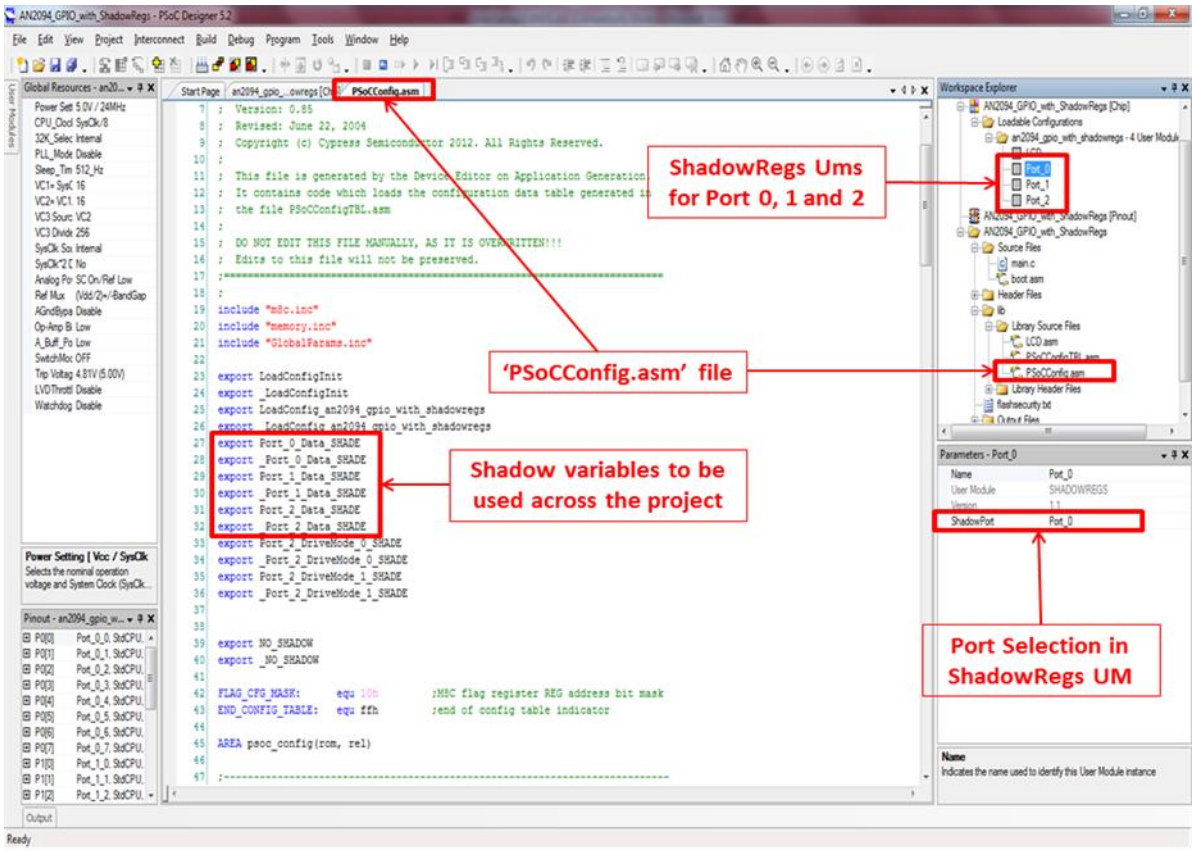
1. 読み出し – Port_0_Data_Shade = x x x x x x 0 0 (ビット 0 = 0 → LED オフ; 初期化されるとビット 1 = 「0」、これはポートからの直接のデータではありません)
2. 変更 – (Port_0_Data_Shade | 00000001) = x x x x x x 0 1
3. 書き込み – Port_0_Data_Shade = x x x x x x 0 1
4. ポートへの書き込み – PRT0DR = Port_0_Data_Shade

PSoC Designer は ShadowRegs と呼ばれるユーザー モジュール (UM) を提供します。このユーザー モジュールは「Misc Digital」ユーザー モジュール カテゴリにあります。プロジェクトにこの UM を配置してポートを割り当てることによって「*Library source files* (ライブラリソースファイル) 」ディレクトリにある *PSoCConfig.asm* ファイルに Port_x_Data_SHADE と呼ばれる変数が作成されます (図 7 を参照)。その内「x」はポート番号です。これで「extern BYTE Port_x_Data_SHADE」を使用してこの変数をインポートすることによって、ファイル全体でそれを使用できるようになります。

GPIO ピンを操作する TX8SW 等のユーザー モジュールもシャドー レジスタを使用するため、入力ピンが他のユーザー モジュールによって使用される出力ピンと混合されることは問題を引き起こしません。

ユーザーがピン名を変更してシャドー レジスタをポートに配置した時、シャドー レジスタ用のピン名を持つエイリアスも *psocgpioint.h* および *psocgpioint.inc* ファイルに作成されます。「ピン名のアサイン」をご参照ください。

図 7. PSoC Designer のシャドー変数の場所



シャドー レジスタの使用はアプリケーション ノードのウェブ ページからダウンロードできる例 2 で示されています。このサンプル プロジェクトの詳細についてはサンプル プロジェクトをご参照ください。

6 GPIO 割り込み

優先順位があるデジタル信号を処理する必要がある場合は特に割り込みは GPIO システムのもう 1 つの重要部分です。PSoC での割り込みシステムは広範なトピックです。PSoC で使用できるすべての割り込みおよびその優先順位についてはそれぞれのデバイスの TRM をご参照ください。本節では GPIO 割り込みについて説明します。

PSoC の各 GPIO ピンは、立ち上がりエッジ、立ち下がりエッジ、前回の読み出し内容との差のイベント発生時に割り込みを生成するように構成できます。GPIO 割り込みをトリガするイベントを構成する方法は 2 つあります。1 つは GPIO コンフィギュレーション ウィンドウで割り込みを構成する方法 (「デバイス エディタ コンフィギュレーションデバイス エディタ コンフィギュレーション」を参照) でもう 1 つはファームウェアで構成する方法です。GPIO 割り込みの構成および有効化に関連付けられるレジスタは表 3 に示されています。

表 3. GPIO 割り込みコンフィギュレーション関連のレジスタ

関連レジスタ	説明	値
PRTxIE	各ポート「x」の割り込みイネーブル レジスタ。このレジスタのビットを設定またはクリアすることでその特定のピンでの割り込みを有効／無効にする	1 – イネーブル 0 – 割り込みをディスエーブル
PRTxIC0 および PRTxIC1	割り込み制御レジスタ - 割り込みをトリガするイベントの種類 (立ち上がりエッジ、立ち下がりエッジ、読み出し内容との差) を設定するために使用。	表 4
INT_MSK0	割り込みイネーブル マスク レジスタ 0	レジスタのビット 5 はグローバル GPIO 割り込みのイネーブル／ディスエーブル ビットです。INT_MSK0_GPIO マクロはレジスタの 5 ビット目を有効／無効にするマスクとして使用することが可能
INT_CLR0	発行された割り込みの読み出しおよびクリア用のレジスタ 0	レジスタのビット 5 は発行された GPIO 割り込みビット。ビットに 0 を書き込むとすべての発行された GPIO 割り込みがクリアされる。ビット読み出しが「1」の場合は発行された GPIO 割り込みがあり、または「1」を書き込んでソフトウェアを介して GPIO 割り込みを発行[4]

表 4. 割り込み制御レジスタの設定

PRTxIC1 [n]	PRTxIC0 [n]	割り込みの種類	説明
0	0	無効	割り込みが無効化
0	1	立ち下がりエッジ	入力信号の 1 から 0 への遷移時に発生する割り込み
1	0	立ち上がりエッジ	入力信号の 0 から 1 への遷移時に発生する割り込み
1	1	読み出し内容との差	PRTxDR[n]の最後の読み出し値とピンの状態が異なる時に発生する割り込み

表 4 では「x」はポート番号を表し、「n」は構成されるポートのレジスタ／ピンのビットを表しています。

GPIO 割り込みを実行するには次のようにしてください。

1. GPIO コンフィギュレーション ウィンドウではピン用の割り込みの種類を設定します。この設定は PRTxICx レジスタに書き込んでコードで行うこともできます。
2. メイン アプリケーション コードで INT_MSK0 レジスタのビット 5 をセットして GPIO 割り込みを有効にします。これは M8C_EnableIntMask マクロを使用して行うことができます。

[4] 「0」を書き込んでかつENSWINT = 0の場合発行された割り込みが存在すればクリアします。
「1」を書き込んでかつ ENSWINT = 0の場合何の影響もありません。
「0」を書き込んでかつENSWINT = 1の場合何の影響もありません。
「1」を書き込んでかつ ENSWINT = 1 の場合汎用入力と出力 (ピン) の割り込みを発行します。

ENSWINT ビットは INT_MSK3 レジスタのビット 7 です。

3. PRTxIE レジスタに書き込むことで対応するピンの割り込みを有効にします。割り込みの種類が GPIO コンフィギュレーション ウィンドウで選択された場合 PSoC Designer は起動処理中に PRTxIE レジスタ ビットを自動的にセットします。
4. 割り込みの処理用に C またはアセンブリ言語の ISR を書きます。ISR が C 言語で書かれた場合関数が ISR であることをコンパイラに通知するために「#pragma interrupt_handler」ディレクティブを使用して ISR を宣言します。
5. コードを配置して割り込みを ISR 関数へリダイレクトします。これは 2 つの方法で実行できます。
   a. 割り込みが GPIO コンフィギュレーション ウィンドウで有効にされた時 PSoC Designer は *psocgpioint.asm* と呼ばれるライブラリ ファイルを生成します。このファイルには PSoC_GPIO_ISR というプレースホルダ関数があります。割り込みがこの関数へリダイレクトされる可能性があります。

      例えば MyGpioIsr という C 関数がある場合「ljmp _MyGpioIsr」コードを配置します (アセンブリ言語から C 関数を読み出す時に下線を関数名に追加する必要があります) 。
   b. リダイレクト命令を起動コードに直接追加します。それを行うにはプロジェクト フォルダにある *boot.tpl* ファイルを開きます。これは PSoC Designer が *boot.asm* ファイルを生成するために使用するテンプレート ファイルです。*boot.tpl* ファイルの GPIO 割り込みベクタでは「@INTERRUPT_7」コードをコメントアウトし、リダイレクト命令をユーザーの GPIO ISR に追加します。*boot.tpl* ファイルを保存してアプリケーションを生成します。ここでは *boot.asm* ファイルに GPIO ISR へのリダイレクト命令があります。

      GPIO 割り込みに関連する割り込みベクタが 1 つだけある点に注意してください。割り込みが複数のピンで有効にされた場合割り込みを発生させるピンを検出して、それに応じて処理するのはアプリケーション コードの責任です。

      以下のコード例では P0_0 および P0_1 での割り込みが有効にされます。P0_0 割り込みは立ち下がりエッジ割り込みとして構成されている一方で、P0_1 割り込みは「読み出し内容との差」として構成されています。GPIO 割り込みは「M8C_EnableIntMask」マクロを使用して有効にされ、最終的にグローバル割り込みが有効にされます。GPIO_ISR ではどのピンが ISR のこのインスタンスをもたらし、それに応じてデータが処理されるかをチェックする「if」制御構造が配置されます。

GPIO 割り込みの設定および有効化

```
/* P0_0 configured as falling edge interrupt */
PRT0IC0 |= 0x01;
PRT0IC1 &= ~0x01;

/* P0_1 configured as Change from Read interrupt */
PRT0IC0 |= 0x02;
PRT0IC1 |= 0x02;

/* Enable P0_1 and P0_0 interrupts */
PRT0IE |= 0x03;

/* Enable GPIO interrupts */
M8C_EnableIntMask(INT_MSK0, INT_MSK0_GPIO);

/* Enable Global interrupts */
M8C_EnableGInt;
```

GPIO ISR の書き込み

```
/* Function prototype for GPIO_ISR*/
#pragma interrupt_handler GPIO_ISR

/* GPIO ISR in C where GPIO interrupts are processed */
void GPIO_ISR(void)
{
/* variable to have a copy of prev P0_1 value for change from read comparison
*/
static BYTE port0_prevValue;

/* Check if interrupt because of P0_0 falling edge:
First condition checks for P0_0 to be '0'
Second condition checks if it was '1' in the last ISR */
if(((PRT0DR & 0x01) == 0) && ((PRT0DR & 0x01) == 0x01))
{
/* Process P0_0 interrupt */
}

/* Check if interrupt because of P0_1 change from read */
if ((PRT0DR ^ port0_prevValue)==0x02)
{
/* Process P0_1 interrupt */
}

/* Store values of P0_0 and P0_1 for next ISR */
port0_prevValue = PRT0DR & 0x03;
}
```

6.1.1 ISR へのリダイレクトの設定

psocgpioint.asm ファイル内で:

```
;-----------------------------------------
;  FUNCTION NAME: PSoC_GPIO_ISR
;
;  DESCRIPTION: Unless modified, this implements only a null handler stub.
;
;----------------------------------------- PSoC_GPIO_ISR:

;@PSoC_UserCode_BODY@ (Do not change this line.)
;---------------------------------------
; Insert your custom code below this banner
;---------------------------------------
   ljmp _GPIO_ISR
;---------------------------------------
; Insert your custom code above this banner
;---------------------------------------
;@PSoC_UserCode_END@ (Do not change this line.)
   reti
```

同様に以下のコードはアセンブリ言語での同じ実装を示します。

boot.tpl ファイル内で:

```
org   1Ch     ;GPIO Interrupt Vector
;`@INTERRUPT_7`
ljmp _GPIO_ISR
reti
```

これらの変更を行った後 boot.tpl ファイルを保存してアプリケーションを生成します。

6.1.2 GPIO 割り込みの設定および有効化

```
; P0_1 configured as change from read
; Interrupt

M8C_SetBank1
or reg[PRT0IC0], 0x02
or reg[PRT0IC1], 0x02

; Enable P0_1 interrupt
M8C_SetBank0
or reg[PRT0IE], 0x02

;Enable GPIO interrupts
M8C_EnableIntMask INT_MSK0, INT_MSK0_GPIO

;Enable Global interrupts
M8C_EnableGInt
```

コード 1: GPIO ISR

```
export MyGpioIsr

area text
;GPIO ISR in ASM where all GPIO ISRs ;are processed
GPIO_ISR:

;Preserve CUR_PP, X and A
push A
push X
mov A, reg[CUR_PP]
push A

;Change CUR_PP to port0PrevValue's Ram ;page (port0PrevValue can be
defined ;in any '.c' file included in the ;project as a global BYTE variable)
RAM_SETPAGE_CUR >_port0PrevValue

;Read PRT0DR into A
mov A, reg[PRT0DR]

;take a copy of PRT0DR into X for ;storing in port0PrevValue at the end
mov X, A

;XOR PRT0DR value in A and PRT0DR's ;prev value
xor A, [<_port0PrevValue]

;AND with 0x02 to read P0_1's state
and A, 0x02

;If zero flag is set, no change in ;state, so not P0_1 ISR
jz .NoP0_1_ISR
```

```
;
;Comes here if the XOR operation ;resulted a non zero result
;
;Process code for P0_1 ISR
;
.NoP0_1_ISR:
;
;Process code for other GPIO ISRs
;
RAM_SETPAGE_CUR >_port0PrevValue
mov [<_port0PrevValue], X

;Restore CUR_PP, X and A in order they ;were pushed
pop A
mov reg[CUR_PP], A
pop X
pop A
reti
```

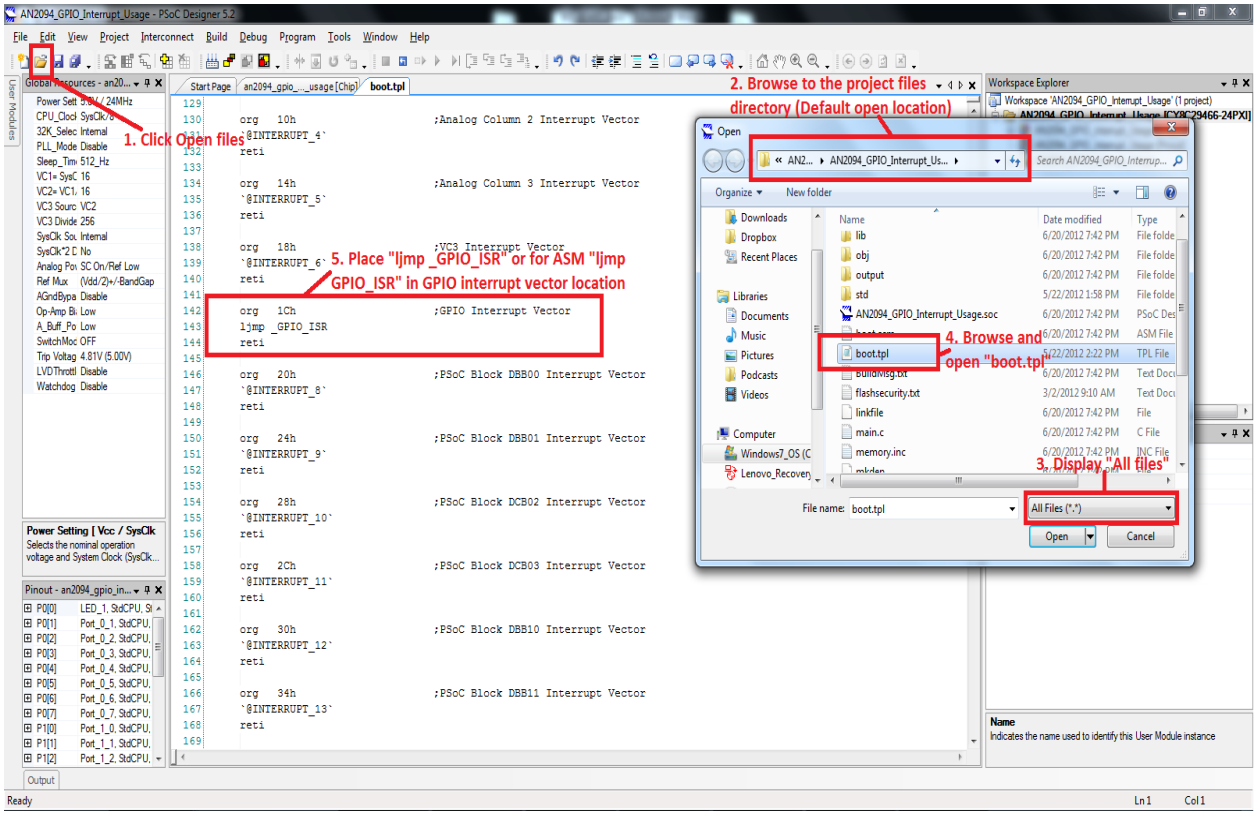
ISR へリダイレクトする方法は C コード例の中で使用したものと同じです。*psocgpioint.asm* ファイルまたは *boot.tpl* ファイル内に配置した ljmp 命令では MyGpioIsr 前の下線は不要です。

例 3 は GPIO 割り込みの実装を示します。アプリケーション ノートのウェブ ページからこの例をダウンロードできます。このプロジェクト例の詳細については「サンプル プロジェクト」を参照してください。

6.2 割り込みの注意事項

- グローバル割り込みイネーブル ビットは「M8C_EnableGInt」マクロを使用して設定する必要があります。
- GPIO の駆動モードは High-Z アナログに設定するべきではありません。そうでなければその特定のピンに割り込みが発生しません。
- GPIO 割り込みが「読み出し内容との差」として構成された場合ピンの値は次の割り込みの発生のために PRTxDR レジスタを使用して読み出される必要があります。PRTxDR の最後の読み出し値とピンの状態が異なる時にのみこの割り込みはトリガされます。
- ISR が正しく実行され制御を返すためには C 言語の ISR 関数は「#pragma interrupt_handler」ディレクティブを使用して定義される必要があります。
- *asm* ファイルの中で定義された ISR 関数は使用したレジスタを保存し、終了する前にこれらのレジスタを復元する必要があります。復帰のために通常の RET 命令ではなく RETI 命令を使用しなければなりません。
- C または ASM で定義した ISR 関数へのリダイレクトは *psocgpioint.asm* ファイルまたは *boot.tpl* ファイルに配置されます。*boot.tpl* ファイルおよび GPIO ISR の場所は図 8 に示されています。

図 8.*boot.tpl* ファイル内の GPIO ISR の場所



7 その他の GPIO リソースおよびヒント

7.1 GPIO グローバル選択レジスタ

PRTxGS レジスタは GPIO ピンが CPU の制御下にあるか、またはグローバル入力バスかグローバル出力バスに接続されているかを決定します。PRTxGS レジスタのピンに対応するビットがクリアされると、PRTxDR レジスタに書き込むことによって CPU がピンを制御することができます。PRTxGS レジスタのビットがセットされると、ピンはグローバル バス (グローバル入力およびグローバル出力) に接続され、デジタル ブロックの入力または出力に直接接続することができます。

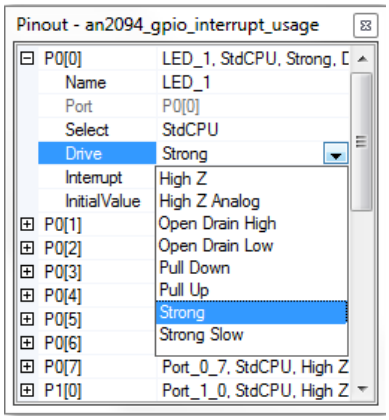
7.2 アナログ MUX (AMUX) バス制御レジスタ

アナログ MUX (AMUX) バス制御レジスタ (MUX_CRx)[5] は内部アナログ MUX バスへの GPIO ピンの接続を有効／無効にします。そしてこのアナログ MUX バスは PSoC デバイス内の様々なアナログ ブロックへの入力として使用可能となります。例えば MUX_CR1 のビット「0」をセットすると、P1_0 が AMuX バスに接続されます。AMUX 設定およびルーティングの詳細については各デバイスの TRM をご参照ください。

7.3 ピン名のアサイン

図 9 に示すように GPIO コンフィギュレーション ウィンドウでは各ピンに意味のある固有の名前を付けられます。

図 9. ピン名のアサイン



ピンに名前を与えた時、PSoC Designer はピンに関連付けられているすべてのレジスタのマクロを、*PSoCGPIOINT.h* ファイル (C ファイルで使用) および *PSoCGPIOINT.asm* ファイル (ASM ファイルで使用) に生成してアクセスを容易にします。マクロ一覧には以下のためのマクロが含まれています。

- ポート データ レジスタ (PRTxDR)
- ポート駆動モード レジスタ (PRTxDMx)
- ポート割り込みイネーブル レジスタ (PRTxIE)
- ポート割り込み設定レジスタ (PRTxICx)
- ポート グローバル選択レジスタ (PRTxGS)
- ピン マスク
- シャドー レジスタ

[5] CY8C21x34、21x45、22x45、24x94、28xxx デバイス ファミリのみで使用可能です。

表 5 には *PSoCGPIOINT.h* ファイル (C ファイルで使用) および *PSoCGPIOINT.inc* ファイル (ASM ファイルで使用) に生成されるマクロの概要が説明されています。これらのマクロはそのピン関連の設定および情報にアクセスするためにすべての関数で直接使用することができます。

表 5. 名前を与えたピンに関連付けられるマクロ

ピン名	LED_1
ポート データ レジスタ	LED_1_Data_ADDR
ポート駆動モード 0 レジスタ	LED_1_DriveMode_0_ADDR
ポート駆動モード 1 レジスタ	LED_1_DriveMode_1_ADDR
ポート駆動モード 2 レジスタ	LED_1_DriveMode_2_ADDR
ポート グローバル選択レジスタ	LED_1_GlobalSelect_ADDR
ポート割り込みイネーブル レジスタ	LED_1_IntEn_ADDR
ポート割り込み制御 0 レジスタ	LED_1_IntCtrl_0_ADDR
ポート割り込み制御 1 レジスタ	LED_1_IntCtrl_1_ADDR
ピン マスク	LED_1_MASK
シャドー レジスタ	LED_1_DataShadow

ピン名およびピン マクロを使用する利点はピンが異なったポートに移動されれば、PSoC Designer はピン マクロに関連付けられたレジスタを自動的に更新し、アプリケーション コードでの変更が不要になることです。

以下のコード スニペットはこれらのマクロの使用例です。

データ レジスタを使用して LED_1 ピンに直接書き込む

```
/* Write 1 to LED_1 pin */
LED_1_Data_ADDR |= LED_1_MASK;

/* Write 0 to LED_1 pin */
LED_1_Data_ADDR &= ~LED_1_MASK;
```

シャドー レジスタを使用して LED_1 ピンに書き込む

```
/* Write 1 to LED_1 */
LED_1_DataShadow |= LED_1_MASK;
LED_1_Data_ADDR = LED_1_DataShadow;

/* Write 0 to LED_1 */
LED_1_DataShadow &= ~LED_1_MASK;
LED_1_Data_ADDR = LED_1_DataShadow;
```

LED_1 の駆動モードをストロング モードに変更する

```
/* Set LED_1 drive mode to strong */
LED_1_DriveMode_0_ADDR |= LED_1_MASK;
LED_1_DriveMode_1_ADDR &= ~LED_1_MASK;
LED_1_DriveMode_2_ADDR &= ~LED_1_MASK;
```

グローバル バスへの LED_1 の接続または切断を行う

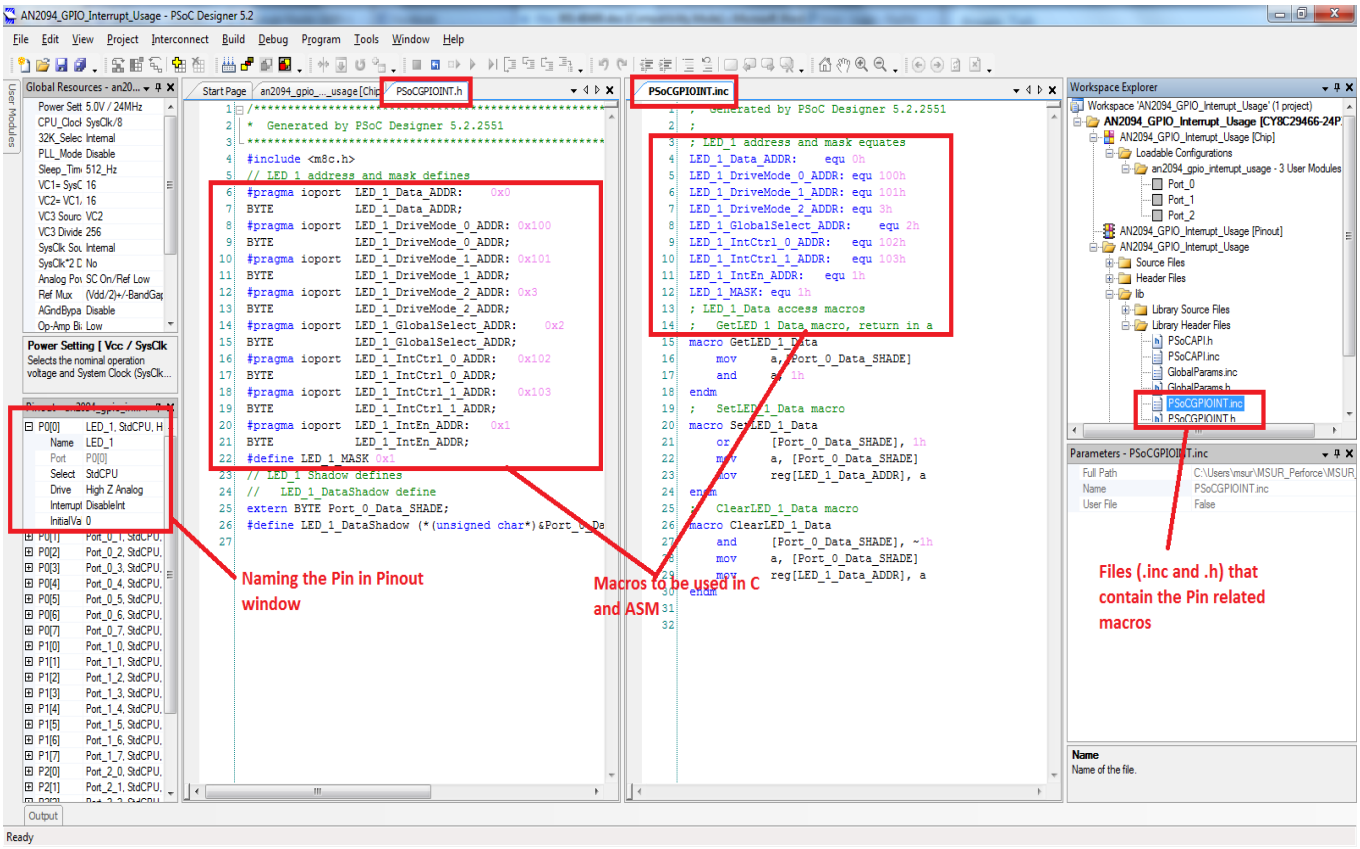
```
/* Connect LED_1 to global bus */
LED_1_GlobalSelect_ADDR |= LED_1_MASK;

/* Disconnect LED_1 from global bus */
LED_1_GlobalSelect_ADDR &= ~LED_1_MASK;
```

SW という名前のピンから読み出し、LED_1 という名前のピンに書き込む

```
if (SW_Data_ADDR & SW_MASK)
{
  /* Write 1 to LED_1 */
  LED_1_DataShadow |= LED_1_MASK;
  LED_1_Data_ADDR = LED_1_DataShadow;
}
else
{
  /* Write 0 to LED_1 */
  LED_1_DataShadow &= ~LED_1_MASK;
  LED_1_Data_ADDR = LED_1_DataShadow;
}
```

図 10 は *psocgpiont.h* および *psocgpioint.inc* ファイル内のピン マクロです。

図 10. ピン関連のマクロ



7.4 レジスタとそれに関連するレジスタ バンク

PSoC 1 のレジスタ マップでは使用可能な 2 つのバンクがあります。各ポート コンフィギュレーション レジスタはこれらのレジスタ バンクの 1 つに属しています。アセンブリ言語でレジスタにアクセスするには特定のレジスタがどのバンクに属しているかを理解する必要があります。C コードではコンパイラはバンクの切り替えを自動的に行います。

ASM でレジスタ バンクをバンク 0 に変更するには M8C_SetBank0 マクロを使用することができます。同様にバンク 1 には M8C_SetBank1 マクロを使用することができます。表 6 には本アプリケーションノートで説明されている GPIO レジスタのそれぞれのレジスタ バンクの詳細が記載されています。

表 6. GPIO 関連レジスタおよびそのレジスタ バンク

レジスタ	レジスタ バンク
PRTxDR	0
PRTxDM0	1
PRTxDM1	1
PRTxDM2	0
PRTxIE	0
PRTxGS	0
PRTxIC0	1
PRTxIC1	1
INT_MSK0	0
INT_CLR0	0
MUX_CRx	1

8 サンプル プロジェクト

8.1 プロジェクト 1: LED 駆動モードの検出

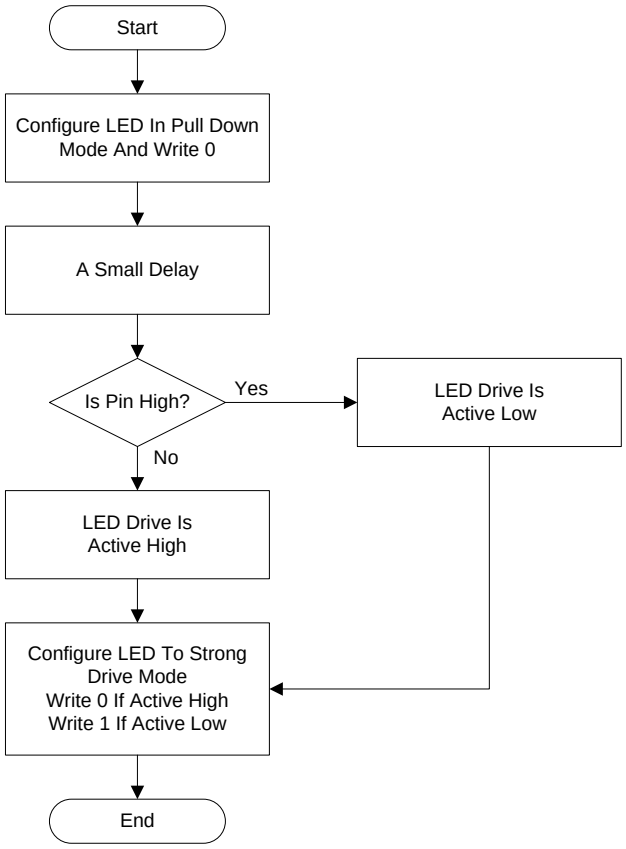
このサンプル プロジェクトはピンの駆動モードが PRTxDMx レジスタを使用して動作中にどのように再構成されるかを示します。

殆どのシステムでは LED をオンするために、通常 GPIO を吸いこみ設定にします。特定のシステムではシンクする代わりに GPIO が LED へ電流を供給してそれらをオンにします。システム設計では LED の実装が知られていますが、ファームウェアを変更することなく設計をアップグレード／更新したい場合があります。

例えばソース GPIO LED デザインを含めた後 GPIO が電流を充分にソースすることができないことが分かりました。またはより高い電流定格で LED に移して LED シンク モードに設計を変更したが、LED が GPIO に接続されているモードに応じてデバイスがそれ自体を調整するようにしたい場合です。この例では LED が GPIO ピンに接続されておりそれに応じて LED がオン／オフされるモードを見つけられる場合、設計にその機能を追加することができます。

図 11 に示すように、フローチャートは L ED 駆動モードがどのように内部で検出されるかを説明しています。

図 11. LED 駆動モード アルゴリズムの検出



サンプル プロジェクトを実装するには、以下のハードウェアが必要です。

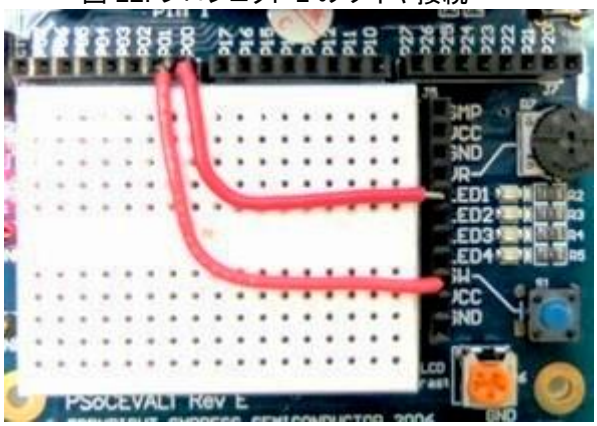
- CY3210 28 ピン CY8C29466-24PXI PDIP PSoC 1 付きの PSoCEval1
- LED シンク モードをチェックするための予備 LED および 1kΩ 抵抗
- CY3217 MiniProg1 または CY8CKIT-002 MiniProg3
- 接続用ワイヤ

サンプル プロジェクトをテストするために、以下の手順に従ってください。

1. CY3210 基板に設けられている 28 ピン PDIP ソケットに CY8C29466-24PXI28 ピンデバイスを挿入します。

2. MiniProg1 または MiniProg3 を CY3210 基板のプログラミング ヘッダー (J11) に接続します。
3. PSoC programmer 3.23.1 を開きます (またはその後) 。MiniProg1 または MiniProg3 を接続します。
4. 本アプリケーションノートに添付されている AN2094_GPIO_DM_Reconfig プロジェクトのルートディレクトリにある AN2094_GPIO_DM_Reconfig.hex ファイルを閲覧します。例えば AN2094 > AN2094_GPIO_DM_Reconfig > AN2094_GPIO_DM_Reconfig.hex の手順に従ってください。
5. PSoC programmer のプログラム ボタンを使用して、選択したファイルでデバイスをプログラムします。プログラミング オプションの詳細については、「AN2015 - PSoC 1 Reading and Writing Flash & E2PROM」 をご参照ください。
6. 図 12 に示すように、P0_1 を SW に接続し、P0_0 を L ED1 に接続します。

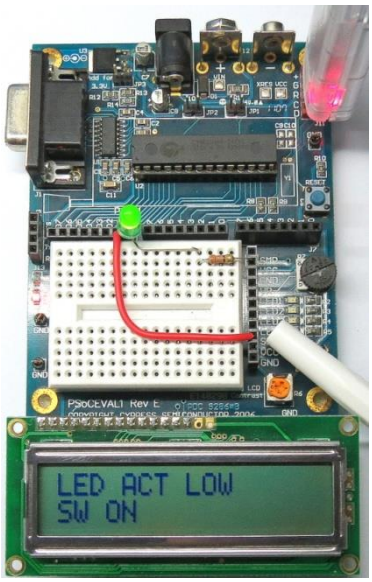
図 12. プロジェクト 1 のワイヤ接続



7. CY3210 基板の J9 ヘッダーにキャラクタ LCD (CY3210 PSoCEval1 kit で提供された) を接続します。
8. 5V 動作のために CY3210 キットの JP3 を取り外します。
9. MiniProg1 または MiniProg3 または 5V の DC アダプタを使ってボードに電力を供給します。
10. 図 13 に示されているように LCD の行 0 に「LED ACT HI」が表示され、SW を押すと LED1 が点灯し、LCD の行 1 に「SW ON」が表示されます。
11. 同様に図 14 に示すように 1kΩ 抵抗を介してシンク モードで予備の LED をワイヤでつなぎ (一方の端は LED に接続し、残りの端は Vcc に接続します)、それを P0_0 に接続すると LCD に「LED ACT LOW」が表示され、LED ON/OFF が SW ON/OFF に続きます。

図 13. 例 1 の出力 - LED アクティブ HIGH

図 14. 例 1 の出力 - LED アクティブ LOW



8.2 プロジェクト 2: シャドー レジスタの使用

シャドーレジスタの使用方法と重要性を説明するために図 15 に示すように CY3210-PSoCEval1 基板を使用した簡単な構成があります。

この例では電源オン時にハードウェアはシャドー レジスタ機能を有効／無効にする前提があります。電源がオンになっている間、P0_2 が V_{DD} に接続されている場合はシャドー レジスタは無効であり、それが GND に接続されている場合はシャドーレ ジスタが有効になります。この例では「シャドー レジスタの使用」で説明されているシナリオを実行します。同じポートに入力スイッチと出力 LED があるとします。

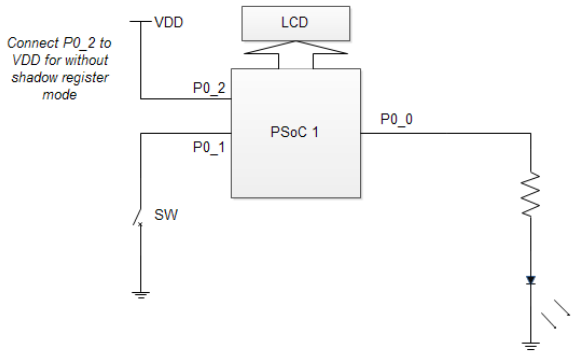
8.2.1 必要なハードウェア

- CY3210 28 ピン CY8C29466-24PXI PDIP PSoC 1 付きの PSoCEval1
- CY3217 MiniProg1
- 接続用ワイヤ

8.2.2 テスト手順

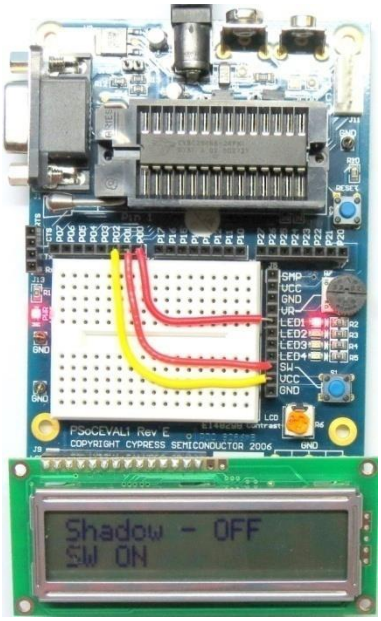
1. CY3210 基板に設けられている 28 ピン PDIP ソケットに CY8C29466-24PXI デバイスを挿入します。
2. MiniProg1 または MiniProg3 を CY3210 基板のプログラミング ヘッダー (J11) に接続します。
3. PSoC programmer 3.23.1 を開きます (またはその後) 。MiniProg1 または MiniProg3 を接続します。
4. 本アプリケーションノートに添付されている「AN2094_GPIO_with_ShadowRegs」プロジェクトのルート ディレクトリにある *AN2094_GPIO_with_ShadowRegs.hex* ファイルを閲覧します。例えば、AN2094>AN2094_GPIO_with_ShadowRegs> AN2094_GPIO_with_ShadowRegs.hex の手順に従ってください。
5. PSoC programmer のプログラム ボタンを使用して、選択したファイルでデバイスをプログラムします。プログラミング オプションの詳細については、「AN2015 - PSoC 1 Reading and Writing Flash & E2PROM」 をご参照ください。
6. CY3210 基板の J9 ヘッダーにキャラクタ LCD (CY3210 PSoCEval1 kit で提供されました) を接続します。
7. シャドー変数を使用せずにプロジェクトをテストするには図 15 に示すように P0_0 を LED1 に、P0_1 を SW に、P0_2 を V_{DD} に接続します。

図 15. 例 2 のシャドー レジスタなしのピン接続



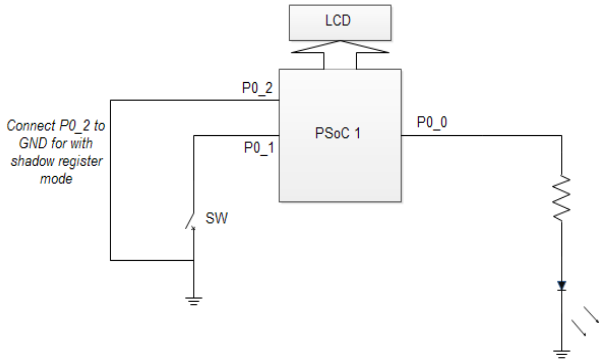
8. 5V 動作のために CY3210 キットの JP3 を取り外します。
9. MiniProg1 または MiniProg3 または 5V の DC アダプタを使って基板に電力を供給します。
10. SW を一度押して放すと LCD の行 1 に常に「ON SW」が表示され、LED1 がオンになります。これは使用されているシャドー変数がないためです。

図 16. プロジェクト 2 のシャドー レジスタなしの出力



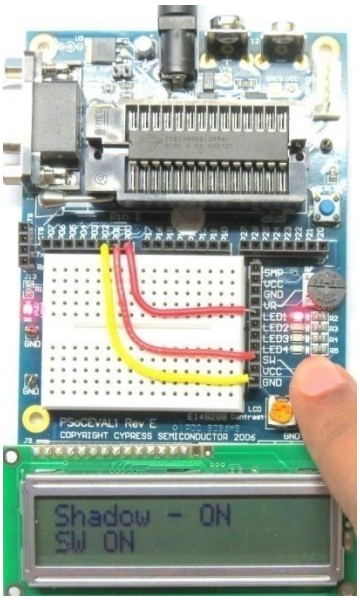
11. 基板の電源を切って図 17 に示されているように P0_2 を GND に接続します。

図 17. 例 2 のシャドー レジスタありのピン接続



12. 基板の電源をオンにします。
13. 図 18 に示すように、LCD の行 0 に「Shadow - ON」が表示されます。スイッチを押すと「SW ON」が表示され LED が点灯します。スイッチを放すと「SW OFF」が表示され LED が消灯します。

図 18. 例 2 のシャドー レジスタありの出力



8.3 プロジェクト 3: 割り込みを使用した LED 点滅動作

GPIO 割り込みの見本として、単純な LED 点滅アルゴリズムを実装した例を挙げます。立ち上がりエッジ割り込みはスイッチに接続されたピン上で有効になります。ISR ではスイッチ上の立ち上がりエッジを示すためにフラグがセットされます。LED は単純な 2ms のデバウンスの後、while ループでトグルされます。LED の ON/OFF 状態が LCD に表示されます。

8.3.1 必要なハードウェア

- CY3210 28 ピン CY8C29466-24PXI PDIP PSoC 1 付きの PSoCEval1
- CY3217 MiniProg1
- 接続用ワイヤ

8.3.2 テスト手順

1. CY3210 基板に設けられている 28 ピン PDIP ソケットに CY8C29466-24PXI 28 ピンデバイスを挿入します。
2. MiniProg1 または MiniProg3 を CY3210 基板のプログラミング ヘッダー (J11) に接続します。

3. PSoC programmer 3.23.1 を開きます (またはその後) 。MiniProg1 または MiniProg3 を接続します。
4. 本アプリケーション ノートに添付されている「AN2094_GPIO_Interrupt_Usage」プロジェクトのルート ディレクトリにある *AN2094_GPIO_Interrupt_Usage.hex* ファイルを閲覧します。例えば、AN2094>AN2094_GPIO_Interrupt_Usage> AN2094_GPIO_Interrupt_Usage.hex の手順に従ってください。
5. PSoC programmer のプログラム ボタンを使用して、選択したファイルでデバイスをプログラムします。プログラミング オプションの詳細については、「AN2015 - PSoC 1 Reading and Writing Flash & E2PROM」 をご参照ください。
6. CY3210 基板の J9 ヘッダーにキャラクタ LCD (CY3210 PSoCEval1 kit で提供されました) を接続します。
7. 図 19 に示すように、プロジェクトをテストするには P0_0 を LE D1 に、P0_1 を SW に接続します。
8. 5V 動作のために CY3210 キットの JP3 を取り外します。
9. MiniProg1 または 5V の DC アダプタを使って基板に電力を供給します。
10. SW を押す度に LED1 がトグルするか確認します。
11. 図 19 と図 20 に示すように LCD 上の行 0 に「GPIO ISR Proj」が表示され、行 1 に LED の ON/OFF 状態が表示されます。

図 19. 例 3 の LED が ON の出力

図 20. 例 3 の LED が OFF の出力

8.4 その他のサンプルコード

PSoC 1 デバイスを使用したより多くのサンプルコードが PSoC Designer 5.4 SP1 またはそれ以降に用意されています。それにアクセスするには **Start Page > Design Catalog > Launch Example Browser** をクリックしてください。

改訂履歴

文書名: AN2094 – PSoC® 1 - GPIO 入門

文書番号: 001-85730

版	ECN 番号	変更者	発行日	変更内容
**	3874055	TMKS	01/20/2013	これは英語版 001-40480 Rev. *E を翻訳し た日本語版 001-85730 Rev. **です。
*A	4669797	HZEN	03/27/2015	これは英語版 001-40480 Rev. *I を翻訳し た日本語版 001-85730 Rev. *A です。
*B	5067584	HZEN	12/24/2015	これは英語版 001-40480 Rev. *J を翻訳し た日本語版 001-85730 Rev. *B です。
*C	6071920	SSAS	02/15/2018	これは英語版 001-40480 Rev. *K を翻訳し た日本語版 001-85730 Rev. *C 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 | 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

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709