AN82250

PSoC Creator の Verilog によるプログラマブル ロジック設計の実装方法

著者: Vijay Kumar Marrivagu/Antonio Rohit De Lima Fernandes

関連製品ファミリー: CY8C3xxx, CY8C5xxx, CY8C42xx, CY8C6xxx

Software Version: PSoC® Creator™ 4.2

関連アプリケーション ノート: AN81623、AN82156

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/AN82250 をご覧ください。

AN82250 は、PSoC® 3、PSoC 4、PSoC 5LP および PSoC 6 の PLD 部にプログラマブル デジタル ロジックの設計を実施する方法を説明します。また、PSoC の汎用デジタルブロック (UDB) およびそれらのプログラマブルロジック デバイス (PLD) のサブブロックを紹介します。サンプルプロジェクトは、PSoC Creator™内に Verilog ベースのコンポーネントを作成することにより、設計で PLD を使用する方法を説明します。

目次

1 はじめに 1
2 PSoC UDB 2
2.1 PSoC UDB 内の PLD のアーキテクチャ 3
3 PSoC Creator 5
4 サンプル プロジェクト 6
4.1 Verilog コンポーネントの作成: Counter4Bit 9
4.2 Verilog コンポーネントの作成: SeqDetector 17
5 データパス設計対 PLD ベースの設計 20
6 まとめ 20
6.1 その他の情報 20
7 関連リソース 21
A 付録 A: PSoC PLD と競合他社の CPLD とのリソース比較 22
B 付録 B: マクロセル構成図 23
C 付録 C: シーケンス検出器の Verilog コード 24
D 付録 D: ビルド後の設計の注意事項 27
D.1 プロジェクトの報告ファイル 27
D.2 静的タイミング解析 27
改訂履歴 28
ワールドワイドな販売と設計サポート 29

1 はじめに

PSoC 3、PSoC 4、PSoC 5LP および PSoC 6 (以降は PSoC と記載) は単なるマイクロコントローラ以上のものです。PSoC により、マイクロコントローラ、コンプレックス プログラマブル ロジック デバイス(CPLD)、および高性能アナログの機能を統合し、他に例を見ない柔軟性を得ることができます。これにより、費用、基板面積、消費電力、および開発時間を低減できます。

注 : このアプリケーションノートは、UDB がないデバイスには適用されません。デバイスに UDB があるかどうかは、そのデバイスのデータシートを参照してください。

このアプリケーションノートは、PSoC の汎用デジタルブロック (UDB) 内の PLD を紹介し、PSoC Creator コンポーネントを作成することにより、それらを使用する方法を説明します。それは、コンプレックス プログラマブル ロジック デバイス (CPLD) 機能を PSoC に移植するための効果的な最初のステップです。このアプリケーションノートをお読みいただいた後は、PSoC PLD に詳しくなり、PSoC Creator を使用して Verilog ベースの独自のカスタム コンポーネントを作成することができます。

PSoC のデジタル機能の詳しい活用方法については、次のステップとして「AN82156 – PSoC 3, PSoC 4 and PSoC 5LP Designing PSoC Creator Components with UDB Datapaths」をお読みください。

これは上級のアプリケーションノートであり、読者が PSoC および PSoC Creator に精通していることを前提にしています。PSoC が初めての場合は、下記をまず参照してください。

- AN54181 Getting Started with PSoC 3
- AN79953 Getting Started with PSoC 4
- AN77759 Getting Started with PSoC 5LP
- AN221774 Getting Started with PSoC 6 MCUs

PSoC Creator が初めての場合は、下記を参照してください。

- PSoC Creator home page

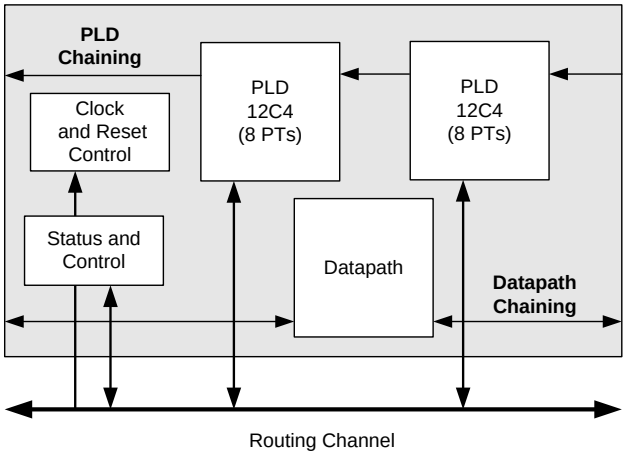
注 :

このアプリケーションノートは、デジタル設計および Verilog の基礎的な知識があることを前提にしています。これらの概念が初めてであれば、「AN81623 – PSoC 3 and PSoC 5LP Digital Design Best Practices」および「KBA86336 – Just Enough Verilog for PSoC」を参照してください。関連の PSoC デジタル設計のリソースは「参考資料」節に記述されています。

2 PSoC UDB

PSoC は、汎用デジタルブロック (UDB) と呼ばれる小型、高速、低電力デジタル ブロックのアレイによってプログラマブル ロジックを実装しています。PSoC のデバイスは最大 24 個の UDB があります。図 1 に示すように、UDB は 2 個の小型プログラマブル ロジック デバイス (PLD)、1 個のデータパス モジュール、および 1 個のステータス コントロール ロジックから成っています。

図 1. UDB のブロック図



名前が示す通り、プログラマブル ロジックは、論理素子 (AND、OR、INVERT および FLIP-FLOP) のアレイを備えているデバイス ファミリーです。一般的に、PLD は、特定のロジック機能を実行するよう設定できる回路です。

PSoC PLD は、レジスタ付きまたは組み合わせ積和ロジック、ルックアップ テーブル、マルチプレクサ、ステートマシン、およびデータパス動作の制御などを作成するのに使用されます。UDB データパスの詳細については、AN82156 を参照してください。

2.1 PSoC UDB 内の PLD のアーキテクチャ

PSoC PLD は、ほとんどの標準 PLD と同様、AND アレイの後に OR アレイが続きます。それらの両方はプログラム可能です。これは、普通、積和アーキテクチャと呼ばれています。

AND アレイ内の 8 本のプロダクト ターム (PT) に 12 本の入力を供給します。各 PT 内で、入力の真 (T) またはその補数 (C) を選択できます。PT の出力は OR アレイへの入力となります。OR ゲートの出力はマクロセル (MC) に提供されます。マクロセルは、追加の組み合わせロジックを持ったフリップ フロップです。

図 2 に示すように、各 UDB に 2 個の PLD があり、それぞれには 8 本の PT と 4 個のマクロセルがあります。PSoC には最大 48 個の PLD があり、そのため最大 192 個のマクロセルと最大 384 本の PT があります。各 PLD は独立しています。また、キャリーチェーンを介して接続するか、またはデジタル システム インターコネクト (DSI) に接続することができます。

付録 A では、PSoC PLD リソースを競合他社の同じサイズの PLD と比較します。

図 2. PSoC PLD の構造

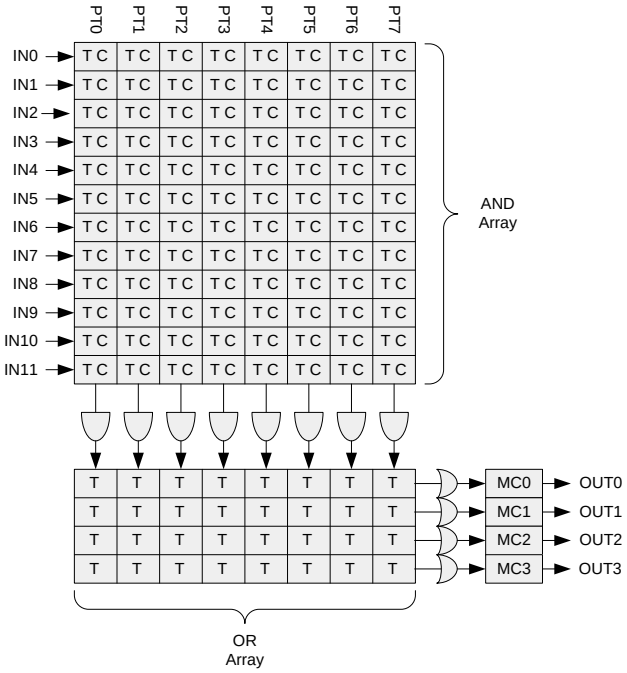
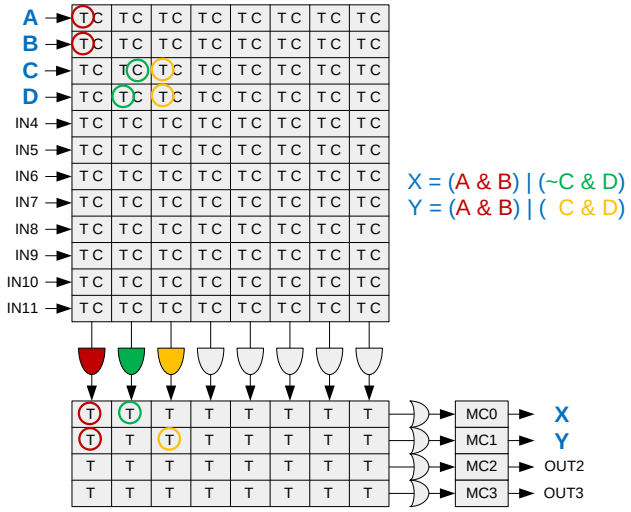


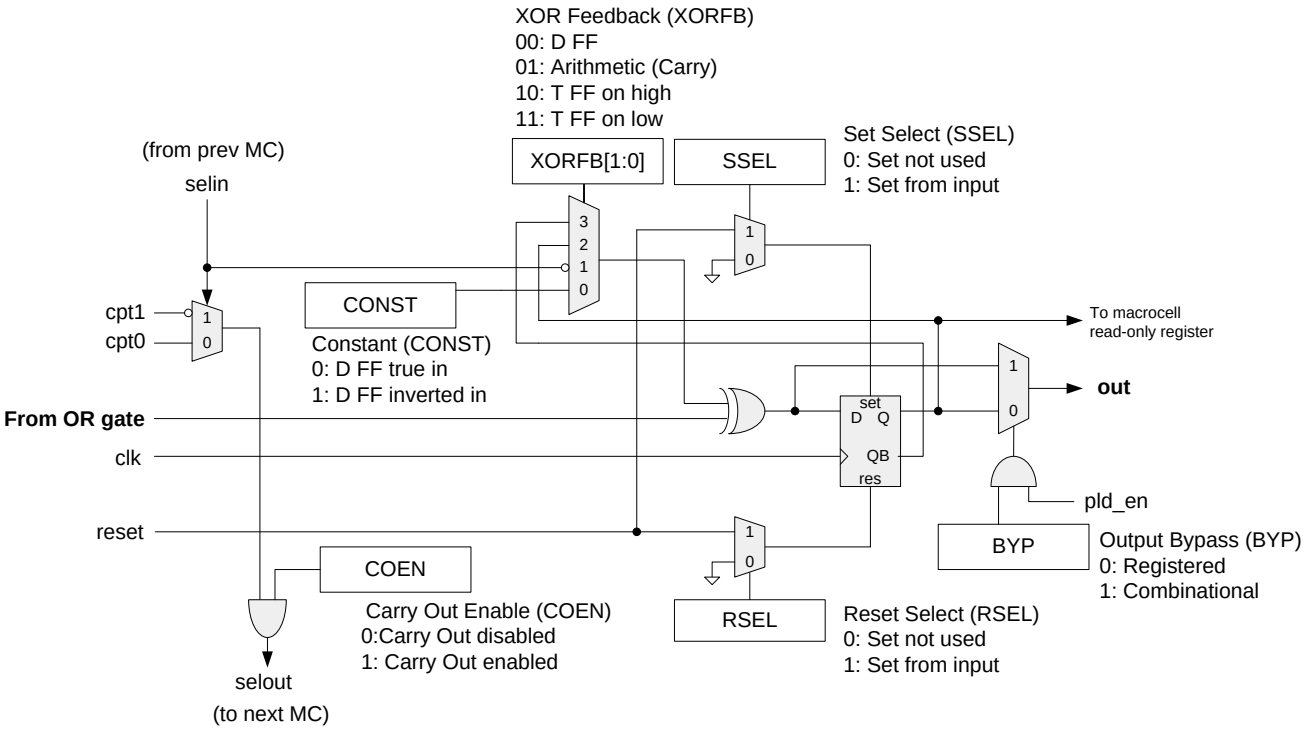
図 3 には、PSoC PLD にマップされた論理式の例を示します。

図 3. PLD にマップされた論理式



マクロセルのアーキテクチャは図 4 に示されます。マクロセル出力はレジスタ付き出力、または組み合わせ出力のいずれかです。付録 B では、2 つの例を使ってマクロセルを介したデータフローを説明します。詳細情報については、使用しているデバイスのテクニカル リファレンス マニュアルの UDB 章のマクロセルの節を参照してください。

図 4. PSoC PLD マクロセルのアーキテクチャ



3 PSoC Creator

PSoC Creator は、ハードウェア開発に対応した回路図ベースの環境を提供します。それは、次の 2 つの大まかな方法で UDB PLD 内にロジック機能とステート マシンを実装することができます。

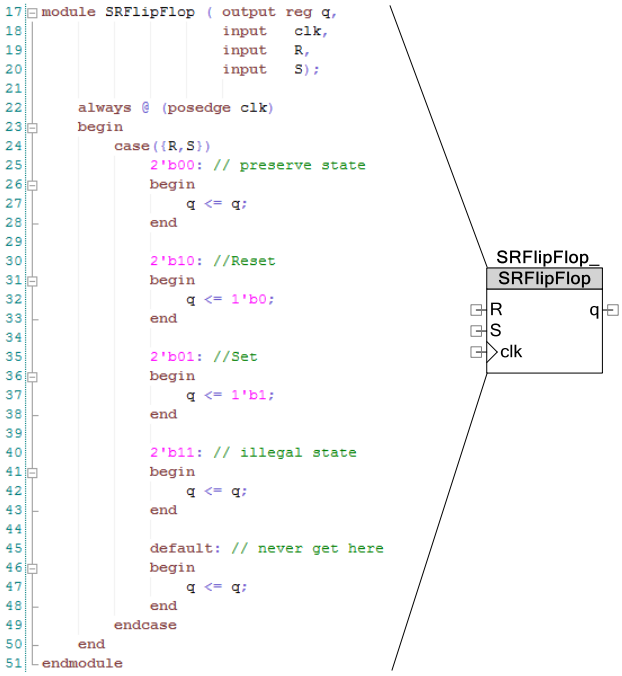
1. Verilog: PSoC Creator は、ハードウェア記述言語 (HDL) である Verilog に対応しています。Verilog を使用して、PSoC UDB にマップされるデジタル機能を実装できます。このプロセスでは、PSoC Creator に備えられた Verilog コンパイラである Warp™合成ツールを使用します。

   このアプリケーションノートは、Verilog ベースのコンポーネントの作成方法を説明します (図 5 を参照してください)。

   Verilog の詳細については、「KBA86336 – Just Enough Verilog for PSoC」を参照してください。

   注：Warp の詳細については、PSoC Creator の Warp Verilog リファレンス ガイド (**Help > Documentation**) を参照してください。

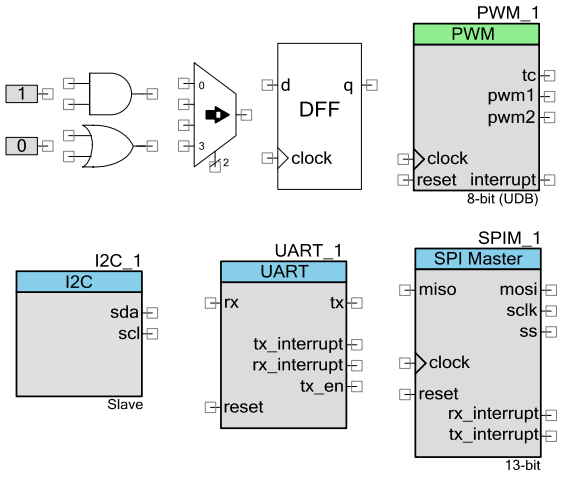
図 5. Verilog ベースのコンポーネント



2. 回路図: このプロセスは、必要な機能を実行するために個別のゲート (AND、OR、XOR、NOT)、DFF、および他のデジタル論理ブロックを配線することを必要とします。PSoC Creator は、すべての論理演算だけでなく、マルチプレクサ、ルックアップ テーブル (LUT)、および他の簡単な PLD ベースの機能にもゲート記号を提供します。

   また、PSoC Creator は作成済み、かつテスト済みの標準ペリフェラル コンポーネントも提供します。これらのコンポーネントは、PLD とデータパスの両方を含む UDB アレイにマップされます。その一部を図 6 に示します。これらのコンポーネントを使用するのは、Verilog を使わずに PSoC の PLD 機能を使用する際の最も簡単な方法です。

図 6. PSoC Creator 内のデジタル コンポーネント



4 サンプル プロジェクト

PSoC について学ぶための最良の方法の 1 つは、それを使用することです。このサンプル プロジェクトでは、簡単な PLD ベースの Verilog コンポーネントを作成する手順について説明します。

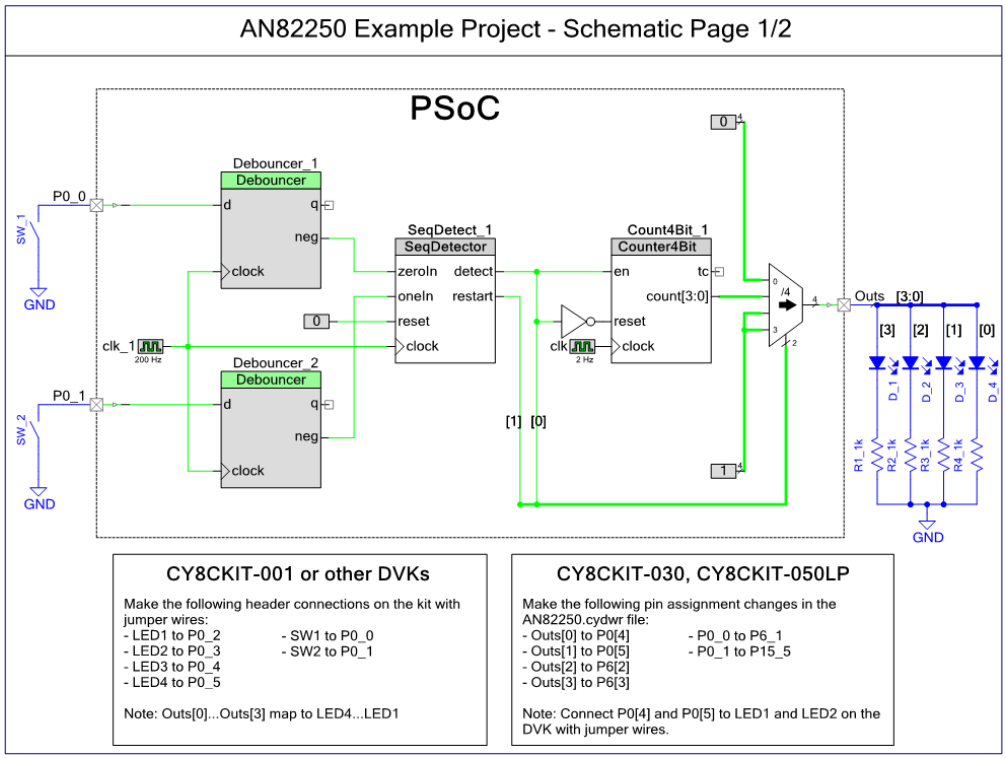
まず、アプリケーションノートのランディング ページで AN82250.zip ファイルをダウンロードします。プロジェクトを見るために、そのファイルをフォルダに解凍して、PSoC Creator で AN82250.cywrk ファイルを開きます。このプロジェクトは、回路図面の指示に従って CY8CKIT-001 PSoC 開発キット (DVK) の PSoC 5LP で作業するよう設計しました。少し変更すると、他の開発プラットフォームで実行できます。このプロジェクトをビルドして PSoC DVK にプログラムします。

このサンプル プロジェクトでは、ファームウェアを必要とせず、5 ビットのシーケンス検出器をハードウェアに完全に実装します。回路図面の詳細については、図 7 を参照してください。このプロジェクトの重要な特長は、回路図面に表示されている全てのコンポーネントが、クロックとピンを除き、UDB PLD に実装されていることです。

このプロジェクトは 2 進数のパターンを入力とします。パターンは、PSoC DVK での 2 個の押しボタン式スイッチにより生成されます。「SW_1」スイッチのボタンを押すと論理 0、また「SW_2」スイッチのボタンを押すと論理 1 と解釈されます。検出器の状態を示すために、4 個の出力で、DVK 上の LED を駆動します。

PSoC をリセットすると、LED が灯り、PSoC が入力受け入れ準備 (即ち、スイッチを押すこと) ができたことを示します。PSoC は図 8 に示した状態図に従います。不完全ではあるが、正しいシーケンスを入力すると、LED は消えて、部分的なシーケンスが入力されたことを示します。間違えてスイッチを押すと、4 個の LED が灯ります。完全な 5 ビット シーケンスを正しく入力すると、LED は 2 進数でカウントし始めます。

図 7. サンプル プロジェクト用の TopDesign 回路図面



入力から出力への信号の流れは以下の通りです。

- 2 個の押しボタン スイッチの入力は、デバウンサ コンポーネントを使って跳ね返りが抑えられ、エッジが検出されます。このコンポーネントをアクセスするためには、PSoC Creator をコンポーネント パック 4 かそれ以降のものに更新してください。
- 入力ピンは、抵抗プルアップとして設定されます。そのため、押しボタンの入力は、スイッチを押すとハイからローに変化します。従って、デバウンサの「ネガティブ エッジでの検出」出力は、スイッチが押されたことが有効であることを示すために使用されます。
- その後、これらの信号は、10110 のシーケンスを検出するよう設定された SeqDetector コンポーネントに行きます。このパターンは、コンポーネントのカスタマイザで 0 (00000) ～ 31 (11111) の間の値を入力することで変更できます (図 9 を参照してください)。
- シーケンス全体を正しく入力すると、「検出」出力がアサートされます。入力をたった一度だけ間違えても、「再起動」出力がアサートされます。
- 「検出」信号がアサートされると、4 ビット カウンターはカウントし始めます。それまでは、リセット状態が維持されます。カウンター周期は、コンポーネントのカスタマイザ内に 1～15 の間の望ましい 4 ビット周期値を入力することで調整できます (図 10 を参照してください)。
- 「再起動」および「検出」信号は出力のマルチプレクサを制御して、図 8 の状態図に基づき 4 個の LED を駆動します。

図 8. サンプル プロジェクトの状態図

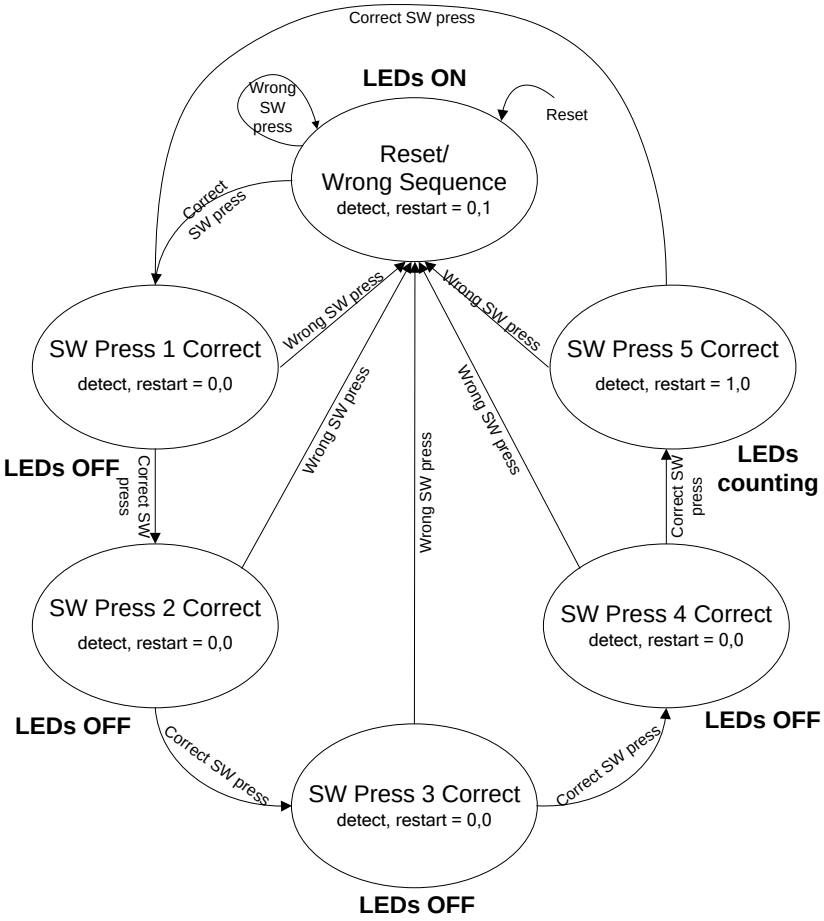


図 9. SeqDetector コンポーネントのカスタマイザ

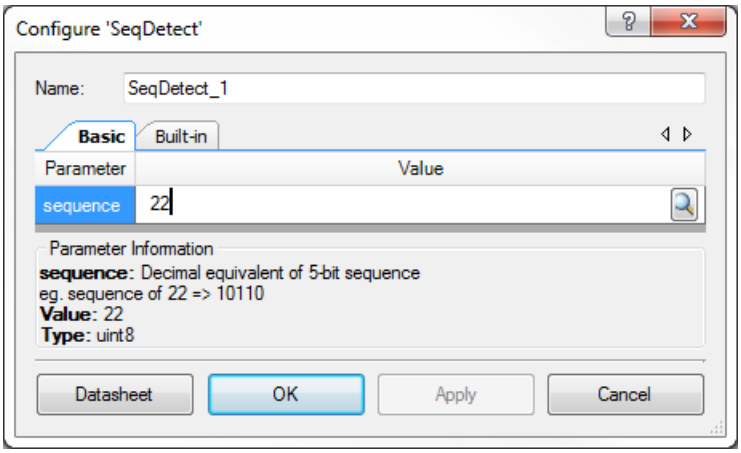
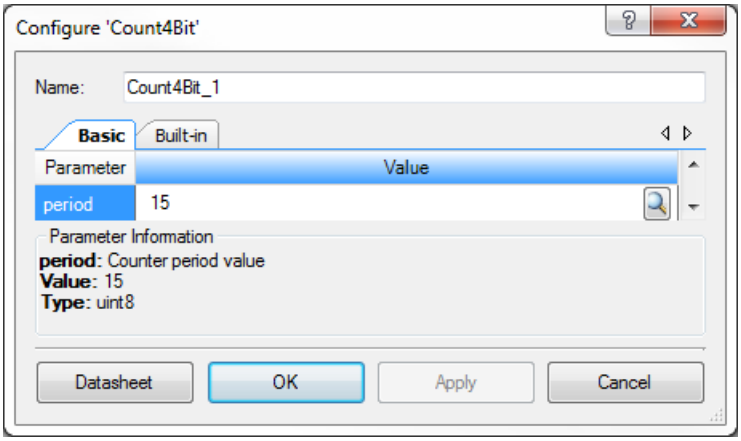


図 10. カウンター コンポーネントのカスタマイザ



注: 4 ビット カウンター用の Verilog ファイルとシーケンス検出器コンポーネントを表示するために、**Workspace Explorer** の **Components** タブに移動してください。

PSoC PLD を効果的に使用するための重要な点は、PSoC Creator で Verilog ベースのコンポーネントを作成することです。「KBA86338 – Creating a Verilog-based Component」では、Verilog ベースのコンポーネントの作成プロセスをまとめます。事例として SeqDetector と Counter4Bit コンポーネントを使用すると、このプロセスについて理解できるようになります。

4.1 Verilog コンポーネントの作成: Counter4Bit

最も簡単なカスタム Verilog ベースのコンポーネントの 1 つは、同期リセットとイネーブル機能を備えた 4 ビットのアップカウンターです。

4.1.1 4 ビット カウンター コンポーネントの作成手順

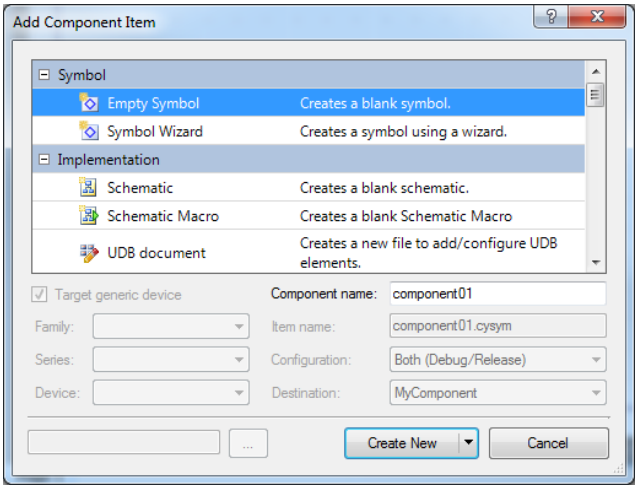
既存のプロジェクトに新しいコンポーネントを追加することができますが、この事例では、既存のプロジェクトが無い状態から始めます。

1. **PSoC Creator** を起動して、新規プロジェクトを作成します。「MyComponents」をプロジェクト名とします。
2. **Workspace Explorer** の **Source** タブで、**MyComponents** ワークスペースを右クリックして、**Add** > **New Project** をクリックします。
3. この新規プロジェクトをコンポーネント ライブラリに設定するために、**Create Project** ダイアログ ボックスで **Design project** の下にある **PSoC Library** を選択し、**Next** をクリックします。次の画面で、このライブラリを作るための CPU コアを選択します。その後、ロケーションをデフォルトのままにして、**Project name** を「MyLibrary」に変更します。

   作成したライブラリに対し、新しいコンポーネントを以下の通りに追加します。
4. **Components** タブで、「MyLibrary」プロジェクトを右クリックして、コンテキスト メニューから **Add Component Item** をクリックします(図 11 を参照してください)。

   バージョン番号をコンポーネント名に含めるのは良いやり方です。コンポーネント名に「_vX_Y」タグを加えます (その内、「X」はメジャーバージョンであり、「Y」はマイナー バージョンです)。PSoC Creator はバージョン管理機能を持っていて、コンポーネントの複数のバージョンを追跡できます。
5. Symbol Wizard コンポーネントのテンプレートを選択して、コンポーネントに「Count4Bit_v1_00」と名付けます。

図 11. カスタム コンポーネントを作成する



空のシンボルから始めることができますが、この事例では、時間をかけないためにウィザードを使用します。詳細については、**Help > Documentation** にあるコンポーネント作成者ガイドを参照してください。

6. Create New ボタンをクリックして、コンポーネントのシンボル ウィザードを起動します。

   このウィザードでは、入力と出力を定義するよう要求して、この情報を使ってコンポーネント シンボルを作成します。

7. 図 12 に示すように、回路図シンボル用に 3 個の入力端子と 2 個の出力端子を定義します。

図 12. Count4Bit 用のシンボル作成ウィザード

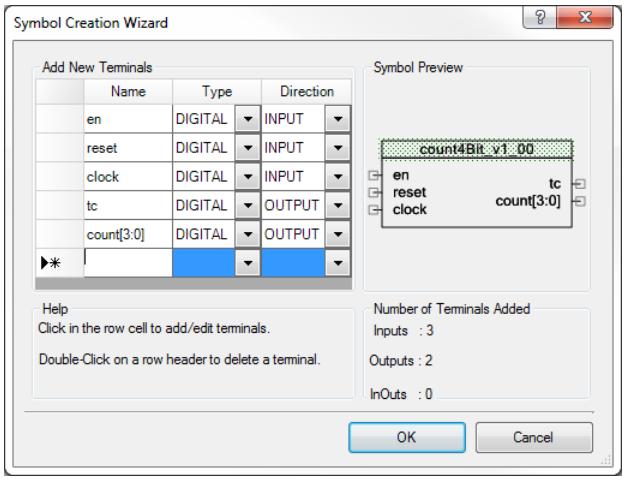


図 13 に示すように、**OK** をクリックして、シンボル回路図でシンボルを生成します。

図 13. 4 ビット カウンター用の最初のシンボル

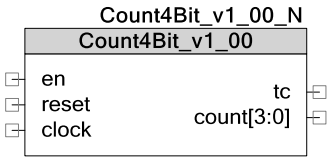
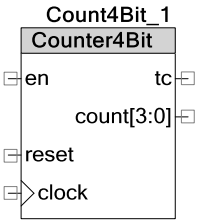


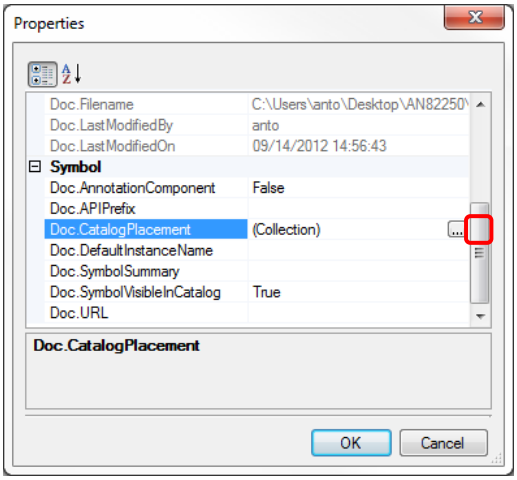
図 14 に示すように、コンポーネントのサイズを変更でき、コンポーネントの外観も変更できます。

図 14. 4 ビット カウンターの最終シンボル



8. シンボル回路図で空きスペースを右クリックして、Properties をクリックします。図 15 に示すように、プロパティ フィールドの Symbol セクションで、Doc.CatalogPlacement の省略記号 (…) をクリックします。

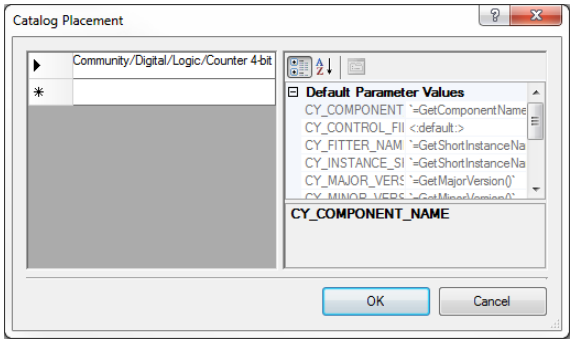
図 15. シンボル属性のダイアログボックス



9. 図 16 に示すように、Catalog Placement ダイアログで、「Community/Digital/Logic/Counter 4-bit」を入力します。

   これにより、カウンターは、**Component Catalog** ウィンドウの **Community** タブで、「Digital」フォルダの「Logic」サブフォルダの中に配置されます。そのカタログ名は「Counter 4-bit」です。

図 16. カタログ配置を設定する

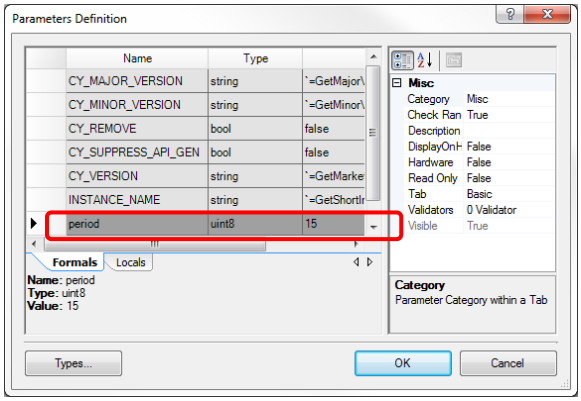


   カウンター用に設定可能な周期値を持たせるために、コンポーネントのパラメーターを追加する必要があります。

10. シンボル回路図で空きスペースを右クリックして、***Symbol Parameters*** をクリックします (図 17 を参照してください)。

    パラメーターの名、タイプ、およびデフォルトの*値*を、それぞれ「period」、「uint8」、および「15」に指定します。このパラメーターにより、ユーザーはカスタマイザでカウンターの周期値を指定できます (図 10 を参照してください)。

図 17. Count4Bit のシンボル パラメーター

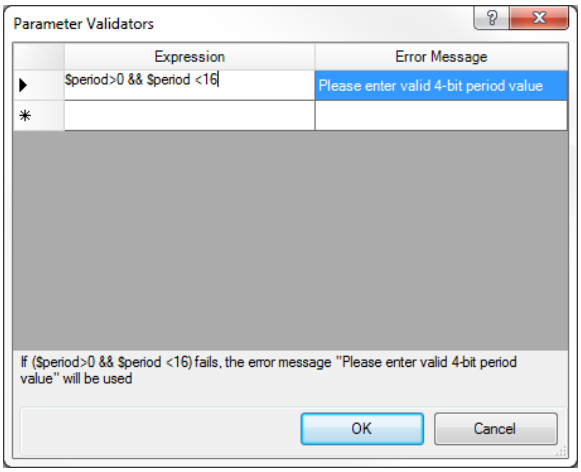


11. **Parameter Definition** ダイアログ ボックスの **Misc** セクションで **Description** フィールドをクリックして、このパラメーター用に記述します。

    Validators フィールドをクリックして、パラメーターの検証ツールを設定します。検証ツールは、パラメーターが許容入力範囲以内にあるかをチェックします。

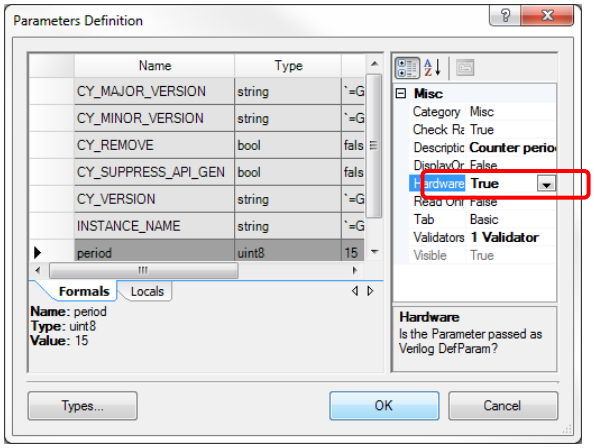
    図 18 のように、周期値が 1～15 の間にあることを保証するよう検証ツールを設定します。**OK** をクリックして、変更を適用します。

図 18. Count4Bit の検証ツールを追加する



12. 図 19 に示すように、**Parameter Definition** ダイアログ ボックスで、**Hardware** フィールドを **True** にセットします。これは、パラメーターを Verilog ファイルに渡すために必要です。

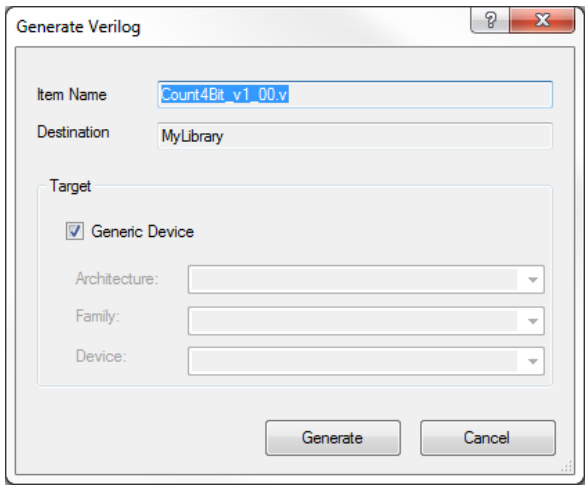
図 19. パラメーターのハードウェアへの引き渡し



次のステップとして、回路図シンボルを Verilog ファイルにリンクさせます。PSoC Creator は、コンポーネント シンボルに基づいて Verilog シェルを生成します。

13. このためには、シンボル回路図で空きスペースを右クリックして、**Generate Verilog** をクリックします。図 20 のように、**Generate Verilog** ダイアログ ボックスでデフォルト設定のままにして、**Generate** をクリックします。

図 20. シンボル用の Verilog ファイルの生成



Target の値は、特定のデバイスの設定を制限するのに使用できますが、この事例ではデフォルト設定を使用します。

作成したシンボルの Verilog ファイルが出来上がります。

注: Verilog ファイルには、3 組の`#start header` - `#end` があります。このファイルを編集する時、全てのコードをこれらのセクション内に配置してください。これらのセクション外での Verilog ファイルへの変更は、Verilog ファイルを再生成すると無くなります。

この時点で、Verilog でカウンターを記述できるようになります。参照のために、完全なコードはコード 1 に示します。

4.1.2 Verilog 設計: 4 ビット カウンター

まず、出力をレジスタ付き出力にします。Count4Bit_v1_00 モジュールの I/O リストを次のように変更します。

```
output reg [3:0] count,
output reg tc,
```

注: Verilog ファイルを再生成する場合、これらの変更を再び行う必要があります。また、これらの定義はファイル内のどこか他の所で行うことはできません。

次に、これが同期設計であるため、Verilog ファイルで`#start body` と`#end` コメントの間に *always* ブロック (およびクロック エッジの情報) を加えます。

```
always @ (posedge clock)
      begin
      . . .
end
```

注: タイミングと同期化の失敗の可能性を低減するために、PSoC 設計でポジティブ エッジ クロックを使用したほうが良いです。

カウンターには同期リセット信号があります。この信号がアサートされると、「tc」と「count」の両方をクリアします。

```
if(reset)
      begin
   count <= 4'b0000;
   tc <= 1'b0;
end
```

注: 非同期リセット／プリセット信号もサポートされます。非同期フリップ フロップ合成の詳細については、「*Warp Verilog Reference Guide*」の 3.3.2 節を参照してください。

`en` 入力は、ハードウェア イネーブルです。この入力が LOW の場合、出力はアクティブのままであるが、コンポーネントの状態は変わりません。

```
if(en)    /* start counting */
begin
. . .
end
else      /* preserve state */
begin
  count <= count;
  tc <= tc;
end
```

`count` が周期値に達すると、`count` が `period` に等しいである限り、端末のカウント出力 `tc` は論理 1 となる必要があります。

```
if(count == period)
 begin
   tc <= 1'b1;
   count <= 4'b0000;
```

```
  end
```

そうでないと、4 ビット カウンターは 0 から period までカウントし、ポジティブ エッジごとに count 出力をインクリメントする必要があります。

```
      else
 begin
   count <= count + 1;
   tc <= 1'b0;
 end
```

Verilog ファイルへの変更が終わったら、保存します。ここで、4 ビット アップ カウンターの Verilog 記述が完了します。完全なコードはコード 1 に示します。

コード 1. 4 ビット カウンターの完全な Verilog 設計

```
module Count4Bit_v1_00 (
  output reg [3:0] count,
  output reg tc,
  input   clock,
  input   en,
  input   reset
);

  parameter period = 0;

//`#start body` -- edit after this line, do not edit this line

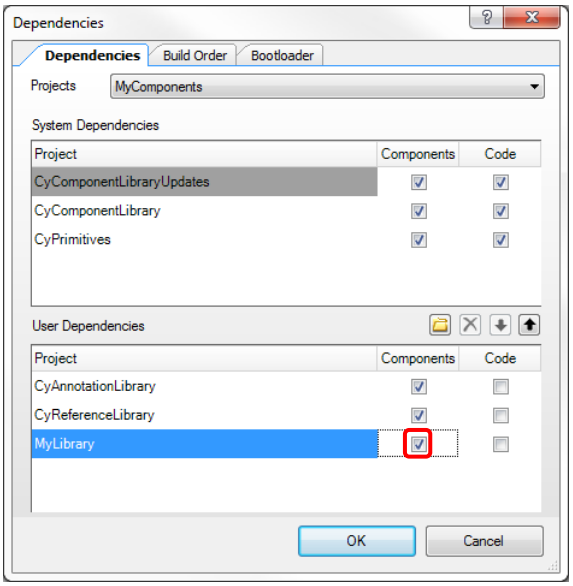
  always @ (posedge clock)
  begin
    if(reset)
    begin
      count <= 4'b0000;
      tc <= 1'b0;
    end
    else
    begin
      if(en)
      begin
        if(count == period)
        begin
          tc <= 1'b1;
          count <= 4'b0000;
        end
        else
        begin
          count <= count + 1;
          tc <= 1'b0;
        end
      end
      else
      begin
        count <= count;
        tc <= tc;
      end
    end
  end

//`#end` -- edit above this line, do not edit this line
      endmodule
```

「MyComponents」プロジェクトでこのコンポーネントを使用するには、「MyLibrary」を依存関係として設定する必要があります。

そうするためには、**Source** タブで **MyComponents** を右クリックして、**Dependencies** を選択します。図 21 に示すように、**User Dependencies** の下にある「MyLibrary」のチェックボックスがチェックされていることを確認してください。

図 21. コンポーネントのライブラリ依存関係を追加



設計で使用するためには、**Component Catalog** プロジェクトの *TopDesign.cysch* ファイルを開いて、Component Catalog に移動します。4 ビット カウンターは **Community** タブに配置されています。それを回路図にドラッグしドロップして、必要な接続を行います。

注：ライブラリ プロジェクトの作成と使用方法については、PSoC Creator のヘルプ記事「Library Component Project」と「Basic Hierarchical Design Tutorial」を参照してください。

次は、Verilog でシーケンス検出器コンポーネントを作成します。

4.2 Verilog コンポーネントの作成: SeqDetector

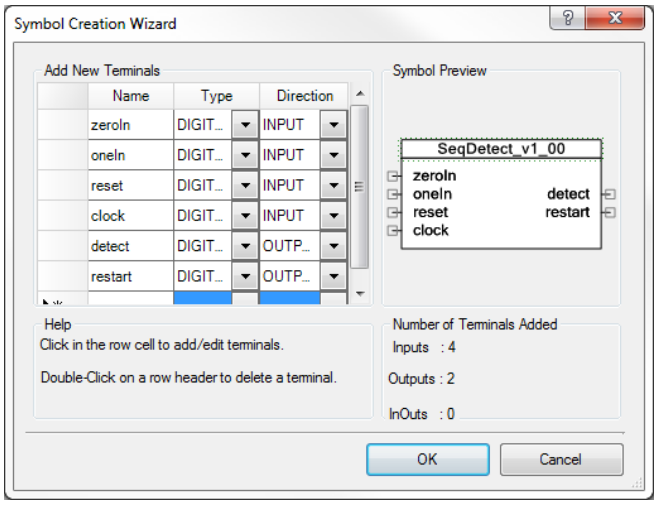
SeqDetector コンポーネントはこのサンプルプロジェクトの中心です。それは、PSoC PLD に実装された設定可能な 5 ビットの 2 進数のシーケンス検出器です。

4.2.1 SeqDetector コンポーネントの作成手順

SeqDetector の作成手順はカウンターと同様です。

1. **Workplace Explorer** の **Components** タブを選択します。**MyLibrary** プロジェクトを右クリックして、**Add Component Item** をクリックします。
2. **Symbol Wizard** コンポーネントのテンプレートを選択して、コンポーネントに **SeqDetect_v1_00** と名付けます。
3. **Create New** ボタンをクリックして、コンポーネント シンボル ウィザードを起動します。
4. 図 22 のように、回路図シンボル用に 4 個の入力端末と 2 個の出力端末を定義します。

図 22. SeqDetector 用のシンボル作成ウィザード



5. 図 23 に示すように、**OK** をクリックして、シンボル回路図でシンボルを生成します。

図 23. シーケンス検出器の最初のシンボル

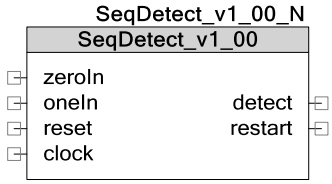
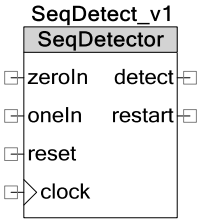


図 24 に示すように、コンポーネントのサイズを変更できます。

図 24. シーケンス検出器の最終のシンボル



6. シンボル回路図で空きスペースを右クリックして、**Properties** をクリックします。

プロパティ フィールドの Symbol セクションで、Catalog Placement の省略記号 (…) をクリックします。CatalogPlacement で Community/Digital/Logic/Sequence Detector 5-bit を入力します。

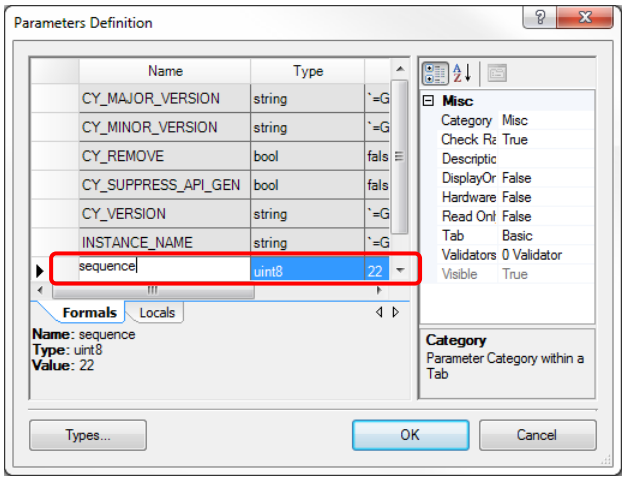
これにより、SeqDetector は、Component Catalog の **Community** タブで、「Digital」フォルダの「Logic」サブフォルダの中に配置されます。そのカタログ名は「Sequence Detector 5-bit」です。

7. SeqDetector のシーケンス値を設定可能にするために、コンポーネントのパラメーターを追加します。

   シンボル回路図で空きスペースを右クリックして、**Symbol Parameters** をクリックします。

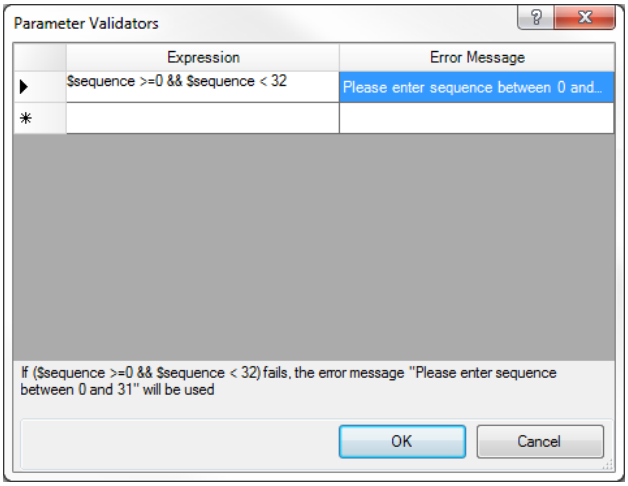
   図 25 のように、パラメーターの名称、タイプ、および初期値をそれぞれ「sequence」、「uint8」、および「22」に指定します。

図 25. SeqDetect 用にパラメーターを定義する



8. **Parameter Definition** ダイアログボックスの **Misc** セクションで **Description** フィールドをクリックして、このパラメーター用に記述します。
9. **Validators** フィールドをクリックして、シーケンスの検証ツールを設定します。図 26 のように、シーケンス値が 0～31 の間にあることを保証するために検証ツールを設定します。**OK** をクリックして、変更を適用します。

図 26. SeqDetect 用の検証ツール



10. **Parameter Definition** ダイアログ ボックスに戻った時、**Hardware** フィールドを **True** にセットします。
11. このためには、シンボル回路図で空きスペースを右クリックして、**Generate Verilog** をクリックします。

    次のステップとして、回路図シンボルを Verilog ファイルにリンクさせます。

12. **Generate Verilog** ダイアログ ボックスですべての設定を初期値のままにして、**Generate** をクリックします。

シーケンス検出器モジュール用の完全な Verilog コードは、付録 C に記載されています。次の節では、コードの主なパーツを説明します。

4.2.2 Verilog 設計: SeqDetector

カウンターと同様、最初の段階として、SeqDetect モジュールの端末リストに出力を登録します。

```
output reg detect,
output reg restart,
```

シーケンス検出器の重要な要素は、合計 6 状態あるステートマシンです (図 8)。それぞれの状態に対応したローカル パラメーターを作成するために、下記のコードを`#start body` コメントの直後に加えます。

```
localparam START   = 3'd0;
localparam STATE_1 = 3'd1;
localparam STATE_2 = 3'd2;
localparam STATE_3 = 3'd3;
localparam STATE_4 = 3'd4;
localparam DETECT  = 3'd5;
```

状態定義の定数が `localparam` キーワードを使って宣言されることに注意してください。これにより、他のモジュールで同じ名前の定数と衝突することを防ぎます。

状態変数を「register」タイプとして、パターン変数を「wire」タイプとして宣言します。

```
reg [2:0] state_curr, state_next;
wire [4:0] pattern = sequence;
```

シーケンス検出器モジュールには、順次ブロックと組み合わせブロックという 2 個の `always` ブロックがあります。

`always` 順次ブロックはポジティブ エッジでトリガーされたフリップ フロップと同じように動作します。

`always` 組み合わせブロックは、以下の通りに定義されるセンシティビティ リストがあります。

```
always @ (oneIn or zeroIn or state_curr or pattern)
```

注: PSoC Creator 用に Verilog、`always` の文言はセンシティビティ リストを持たなければなりません。

`always` 組み合わせブロックは、入力をデコードし、入力と現時点での状態に基づいて次の状態を割り当てます。1 か 0 の入力のいずれかが検出されると、その入力がチェックされ、あるいは現時点での状態が維持されます。

例えば、最初の状態は以下のようです。

```
if((oneIn & pattern[4]) ||
(zeroIn & !pattern[4]))
begin
  state_next <= STATE_1;
end
else
begin
  state_next <= START;
end
```

注: `pattern[4]`はシーケンスの最初の正しい値を保存します。

他の状態は同様に、コンポーネント入力を `pattern[3]`, ... , `pattern[0]`と比較して、次の状態をデコードします。

Verilog ファイルへの変更が終わったら、保存します。この時点で、シーケンス検出器の設計は完了しました。それを設計で使用するには、カウンターの節で説明した手順に従います。

5 データパス設計対 PLD ベースの設計

通信、タイミング、および制御アプリケーションは、各機能を支える論理構成に関する要求が異なります。

経験則として、UDB リソースを利用する最適な方法は以下の通りです。

- PLD (ランダム ロジック): 制御機能、CPLD 統合、グルーロジック。
- データパス (ストラクチャード ロジック): 通信、タイミング、演算。

例えば、以下の、データパス対 PLD で実行された 8 ビット演算と論理演算を考えてみます。

機能	PLD のみにおけるリソース消費		データパスのみにおけるリソース消費	
	PLD	使用量 (%)	データパス	使用量 (%)
ADD8	5	10.4%	1	4.2%
SUB8	5	10.4%	1	4.2%
CMP8	3	6.3%	1	4.2%
SHIFT8	3	6.3%	1	4.2%

注:率は 24 個の UDB を備えているデバイスで計算されます。

PSoC PLD で複合機能を実行できますが、データパス モジュールを活用しないと、リソース不足になりがちです。

6 まとめ

このアプリケーション ノートでは、UDB PLD を紹介し、PSoC Creator で Verilog ベースのコンポーネントを作成するための設計プロセスについて説明します。このアプリケーション ノートをお読みいただいた後、PLD アーキテクチャに詳しくなり、Verilog ベースの独自のカスタム コンポーネントを作成することができます。

PSoC UDB は、デジタル設計に柔軟かつ効果的なアーキテクチャを提供しています。簡単なものからある程度複雑なものまで広範囲のロジック設計を PSoC の PLD に移植することができます。高度に複雑な設計は、PLD とデータパスを組み合わせて使用することで最もよく設計できます。UDB データパスの詳細については、AN82156 を参照してください。

6.1 その他の情報

付録 A は、リソース カウントに関する、PSoC PLD と競合他社の CPLD との比較表を記載します。

付録 B では、2 つの例を使ってマクロセルを介したデータフローを説明します。

付録 C には、シーケンス検出器モジュール用の完全な Verilog コードを記載します。

付録 D では、プロジェクトの報告ファイルと静的タイミング解析を簡単に説明します。

7 関連リソース

アプリケーション ノート

AN82156 - PSoC 3, PSoC 4 and PSoC 5LP Designing PSoC Creator Components with UDB Datapaths

AN81623 – PSoC 3 and PSoC 5LP Digital Design Best Practices

AN62510 – Implementing State Machines with PSoC 3 and PSoC 5LP

AN61290 – PSoC 3 and PSoC 5LP Hardware Design Considerations

AN72382 – Using PSoC 3 and PSoC 5LP GPIO Pins

AN60580 – SIO Tips and Tricks in PSoC 3 and PSoC 5LP

AN54181 – Getting started with PSoC 3

AN79953 – Getting Started with PSoC 4

AN77759 – Getting started with PSoC 5LP

AN221774 Getting Started with PSoC 6 MCUs

KB 記事

KBA86336 – Just Enough Verilog for PSoC

KBA86338 – Creating a Verilog-based Component

KBA81772 – Adding Component Primitives / Verilog Components to a Project

Basics of Verilog and Datapath Configuration Tool for Component Creation

ビデオ

以下のビデオは、PSoC Creator および Verilog コンポーネントの作成プロセスについて説明します。

基本

Creating a New Project

Using the Start Page

コンポーネントの作成

PSoC Creator 113: PLD Based Verilog Components

Creating a New Component Symbol

Creating a Verilog Implementation

Creating a Schematic Implementation

著者について

氏名:	Vijay Kumar Marrivagu
役職:	システム エンジニアのプリンシパル
経歴:	デジタル設計と検証で数年の経験。
氏名:	Antonio Rohit De Lima Fernandes
役職:	アプリケーション エンジニア
経歴:	電気工学における B.E (India、Rajasthan、Pilani、BITS)。

A 付録 A: PSoC PLD と競合他社の CPLD とのリソース比較

表 1 では、PSoC PLD リソースを競合他社の同じサイズの CPLD と比較しています。この表では、UDB データパスでのプログラマブル ロジックを考慮に入れていないことに注意してください。PSoC PLD とデータパスの両方を使用すれば、PSoC はより大きなサイズの CPLD と競争できるようになります。

表 1. PSoC PLD と競合他社の PLD とのマクロセル比較

デバイス	マクロセル (MC)	ブロック数	ブロックごとの マクロセル (MC) 数	ブロックごとの入力数	プロダクト ターム (PT) 数	ブロックごとのプロダクト ターム (PT) 数
サイプレス PSoC						
スーパセット PSoC 3、PSoC 5LP	192	48	4	12	384	8
CY8C42	32	8	4	12	64	8
Altera MAX-II						
EPM240	128～240*	24	10	36	*	*
Lattice ispMACH						
4032ZE	32	2	16	36	160	80
4064ZE	64	4	16	36	320	80
40128ZE	128	8	16	36	640	80
Xilinx Coolrunner-II						
XC2C32A	32	2	16	56	112	56
XC2C64A	64	4	16	56	224	56
XC2C128	128	8	16	56	448	56

* Altera MAX-II は従来のプロダクト タームのアーキテクチャではありません。

B 付録 B: マクロセル構成図

図 27 および図 28 には、それぞれ D フリップ フロップ (D-FF) と T フリップ フロップ (T-FF) 機能のマクロセルを介したデータ フローを示します。

図 27. D-FF 機能が有効なマクロセル

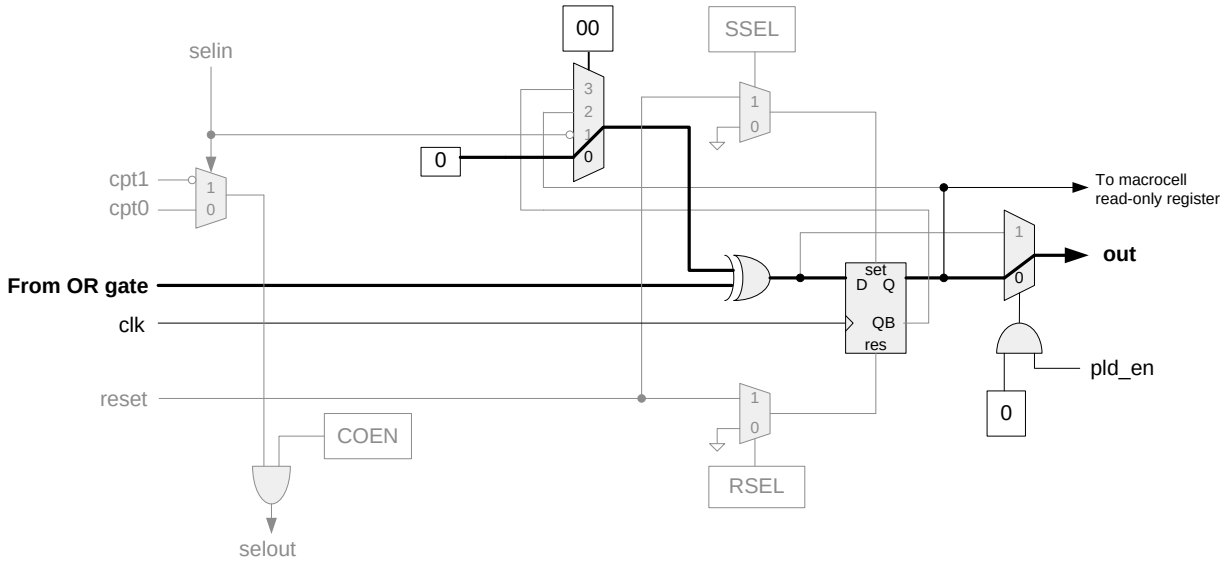
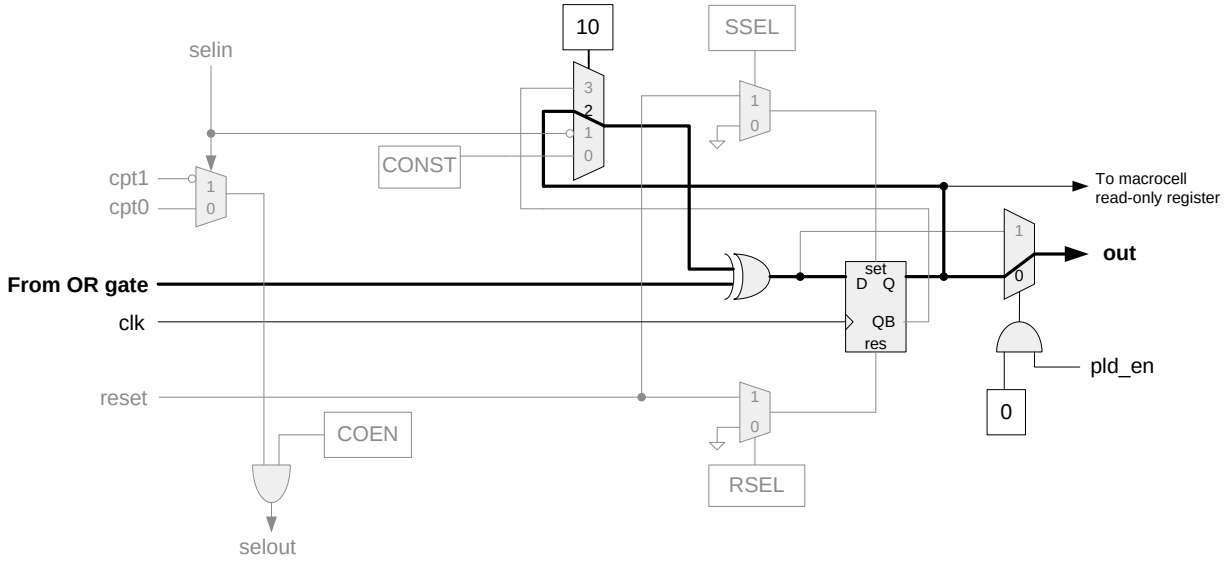


図 28. T-FF 機能が有効なマクロセル



C 付録 C: シーケンス検出器の Verilog コード

```
module SeqDetect_v1_20 (
 output reg detect,
 output reg restart,
 input   clock,
 input   oneIn,
 input   reset,
 input   zeroIn
);
 /* Note that the value assigned to the parameter in this line
  * has no effect. The actual parameter value is taken from
  * the component customizer.
  */
 parameter sequence = 0;

//`#start body` -- edit after this line, do not edit this line
 /* Six states are required.
  * The states follow START -> STATE_1 -> ... -> DETECT if the
  * correct inputs are entered. As soon as a wrong input is entered
  * the design jumps to the START state. The states are defined as
  * localparams (instead of `defines) to limit their scope to this
  * module only.
  */
 localparam START   = 3'd0;   /* detect, restart = 0, 1 */
 localparam STATE_1 = 3'd1;   /* detect, restart = 0, 0 */
 localparam STATE_2 = 3'd2;   /* detect, restart = 0, 0 */
 localparam STATE_3 = 3'd3;   /* detect, restart = 0, 0 */
 localparam STATE_4 = 3'd4;   /* detect, restart = 0, 0 */
 localparam DETECT  = 3'd5;   /* detect, restart = 1, 0 */

      /* registered value to hold 3-bit state */
      reg [2:0] state_curr, state_next;

 /* pattern[4:0] holds the user-supplied sequence value
  * suppose sequence = 22 then pattern[4:0] = 5'b10110
  * Note that pattern[4] is the first-entered user input
  */
 wire [4:0] pattern = sequence;

  /* Sequential block of the state machine - outputs are assigned here */
 always @ (posedge clock)
 begin
    /* reset causes the component to enter the START state */
    if(reset)
    begin
       state_curr <= START;
       /* Immediately assign detect and restart values */
       detect <= 1'b0;
       restart <= 1'b1;
    end
    else /* reset is not asserted - go through states */
    begin
       state_curr <= state_next;

       /* Assign 'detect' value - 1 only in DETECT state, 0 otherwise */
       if (state_next == DETECT)
       begin
          detect <= 1'b1;
       end
       else
```

```
        begin
           detect <= 1'b0;
        end

        /* Assign 'restart' value - 1 only in RESTART state, 0 otherwise*/
        if (state_next == START)
        begin
           restart <= 1'b1;
        end
        else
        begin
           restart <= 1'b0;
        end
     end
end

/* Finite State Machine combinatorial block - contains most of the
 * combinatorial logic.
 */
always @ (oneIn or zeroIn or state_curr or pattern)
begin
   /* If either a one or zero has been entered, take action */
   if(oneIn | zeroIn)
   begin
      case(state_curr)
         START:     /* Initial state */
         begin
            /* check whether the first bit entered is correct */
            if((oneIn & pattern[4]) || (zeroIn & !pattern[4]))
            begin
            state_next <= STATE_1;/* advance to the next state */
            end
            else       /* revert to the initial state */
            begin
               state_next <= START;
            end
         end

         STATE_1:   /* First input is correct */
         begin
            if((oneIn & pattern[3]) || (zeroIn & !pattern[3]))
            begin
               state_next <= STATE_2;
            end
            else
            begin
               state_next <= START;
            end
         end

         STATE_2:   /* Two inputs are correct */
         begin
            if((oneIn & pattern[2]) || (zeroIn & !pattern[2]))
            begin
               state_next <= STATE_3;
            end
            else
            begin
               state_next <= START;
            end
         end
```

```
            STATE_3:   /* Three inputs are correct */
            begin
               if((oneIn & pattern[1]) || (zeroIn & !pattern[1]))
               begin
                  state_next <= STATE_4;
               end
               else
               begin
                  state_next <= START;
               end
            end

            STATE_4:    /* Four inputs are correct */
            begin
               if((oneIn & pattern[0]) || (zeroIn & !pattern[0]))
               begin
                  state_next <= DETECT;
               end
               else
               begin
                  state_next <= START;
               end
            end

            DETECT:    /* All five inputs are correct! */
            begin
/* When in the detect state, if an input is given, show same behavior as START */
/* check whether the bit entered is the correct beginning to a new sequence*/
               if((oneIn & pattern[4]) || (zeroIn & !pattern[4]))
               begin
                  state_next <= STATE_1;
               end
               else /* revert to the initial state */
               begin
                  state_next <= START;
               end
            end

            default:/* we should never get here - reset the component*/
            begin
               state_next <= START;
            end
         endcase
      end
      else /* if neither 1 or 0 have been entered, stay in same state */
      begin
         state_next <= state_curr;
      end
   end

//`#end` -- edit above this line, do not edit this line
endmodule
```

D 付録 D: ビルド後の設計の注意事項

D.1 プロジェクトの報告ファイル

Workspace Explorer ウィンドウの **Results** タブからプロジェクトのビルド レポート (<project_name>.rpt) をアクセスします。正常に完了したビルド後に作成されます。以下は、報告ファイルの主な部分です。

- 技術的なマッピングのまとめ: マクロセル、プロダクト ターム、データパス、ピン、クロック分周器などが図 29 に示されます。

図 29. PSoC Creator プロジェクトのビルド レポート ファイル

TopDesign.cysch | Design09.cydwr | Mycounter.v | **Design09.rpt**

```
660 ------------------------------------------------------------
661 Technology mapping summary
662 ------------------------------------------------------------
663
664 Resource Type                 : Used : Free :  Max :  % Used
665 ============================================================
666 Digital domain clock dividers :    1 :    7 :    8 :  12.50%
667 Analog domain clock dividers  :    0 :    4 :    4 :   0.00%
668 Pins                         :    9 :   63 :   72 :  12.50%
669 Macrocells                   :    5 :  187 :  192 :   2.60%
670 Unique Pterms                :    5 :  379 :  384 :   1.30%
671 Total Pterms                 :    6 :      :      :
672 Datapath Cells               :    0 :   24 :   24 :   0.00%
673 Status Cells                 :    0 :   24 :   24 :   0.00%
674 Control Cells                :    0 :   24 :   24 :   0.00%
```

- 合成結果: Verilog コンパイル、構文解析、高レベルの合成、最適化など合成の各段階で生成されたエラーと警告を一覧表示します。同節は、シンセサイザによって最適化して除去したロジックの詳細情報を含みます。これは、デバッギングおよびトラブルシューティングに役立ちます。
- デジタル部の配置: PLD パッキングのまとめ。図 30 は PLD の利用例を示します。

図 30. PLD パッキングのまとめレポート

```
------------------------------------------------------------
PLD Packing Summary
------------------------------------------------------------
        Resource Type : Used : Free :  Max :  % Used
      ================================================
                PLDs :    2 :   46 :   48 :   4.17%
```

- デジタル部の配置: PLD パッキングのまとめ: PLD の統計値: 図 31 は、ロジック アレイ ブロック (LAB) ごとの PLD PT とマクロセルの平均使用量の例を示します。

図 31. PLD の使用量のレポート例

```
    PLD Resource Type :     Average/LAB
  ========================================
              Inputs :            2.00
              Pterms :            2.50
           Macrocells :           2.50
```

- 最終配置のまとめ: コンポーネントの詳細を提示します。同節は、UDB の使用量、占有率、統計値、および配置 (座標) の詳細を示します。

D.2 静的タイミング解析

デジタル設計のデバッグ処理の重要な一部は、静的タイミング解析 (STA) です。STA はデジタル設計を評価し、信号の入出力間の遅延を計算します。それらの遅延に基づいて、設計で使用される各クロックの最大許容周波数を算出します。

プロジェクトをビルドすると、PSoC Creator は静的タイミング解析レポートを自動的に作成します。このレポートは、各クロックの周波数を制限する、設計内の重要な部分を提示します。算出された最大周波数が要求されたクロック周波数より小さい場合、設計内にタイミング違反が存在することを示す警告を表示します。

タイミング違反の回避方法、および PSoC Creator STA 警告の処理の詳細については、「AN81623 – PSoC 3 and PSoC 5LP Digital Design Best Practices」を参照してください。

改訂履歴

文書名: AN82250 - PSoC Creator の Verilog によるプログラマブル ロジック設計の実装方法

文書番号: 001-89367

版	ECN	発行日	変更内容
**	4136657	09/26/2013	これは英語版 001-82250 Rev. *D を翻訳した日本語版 001-89367 Rev. **です。
*A	4722751	04/27/2015	これは英語版 001-82250 Rev. *F を翻訳した日本語版 001-89367 Rev. *A です。
*B	5184202	03/22/2016	これは英語版 001-82250 Rev. *H を翻訳した日本語版 001-89367 Rev. *B です。
*C	5473611	10/13/2016	新しいテンプレートに更新しました。 サンセットレビューを完了する。
*D	5817630	07/19/2017	ロゴと著作権を更新しました。
*E	6504619	08/27/2019	これは英語版 001-82250 Rev. *J を翻訳した日本語版 001-89367 Rev. *E です。

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

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
198 Champion Court
San Jose, CA 95134-1709

© Cypress Semiconductor Corporation, 2012-2019. 本書面は, Cypress Semiconductor Corporation 及び Spansion LLC を含むその子会社 (以下「Cypress」という。) に帰属する財産である。本書面 (本書面に含まれ又は言及されているあらゆるソフトウェア若しくはファームウェア (以下「本ソフトウェア」という。) を含む) は, アメリカ合衆国及び世界のその他の国における知的財産法令及び条約に基づき Cypress が所有する。Cypress はこれらの法令及び条約に基づく全ての権利を留保し, 本段落で特に記載されているものを除き, その特許権, 著作権, 商標権又はその他の知的財産権のライセンスを一切許諾しない。本ソフトウェアにライセンス契約書が伴っておらず, かつ Cypress との間で別途本ソフトウェアの使用方法を定める書面による合意がない場合, Cypress は, (1) 本ソフトウェアの著作権に基づき, (a) ソースコード形式で提供されている本ソフトウェアについて, Cypress ハードウェア製品と共に用いるためにのみ, かつ組織内部でのみ, 本ソフトウェアの修正及び複製を行うこと, 並びに (b) Cypress のハードウェア製品ユニットに用いるためにのみ, (直接又は再販売者及び販売代理店を介して間接のいずれかで) 本ソフトウェアをバイナリーコード形式で外部エンドユーザーに配布すること, 並びに (2) 本ソフトウェア (Cypress により提供され, 修正がなされていないもの) が抵触する Cypress の特許権のクレームに基づき, Cypress ハードウェア製品と共に用いるためにのみ, 本ソフトウェアの作成, 利用, 配布及び輸入を行うことについての非独占的で譲渡不能な一身専属的ライセンス (サブライセンスの権利を除く) を付与する。本ソフトウェアのその他の使用, 複製, 修正, 変換又はコンパイルを禁止する。

適用される法律により許される範囲内で, Cypress は, 本書面又はいかなる本ソフトウェア若しくはこれに伴うハードウェアに関しても, 明示又は黙示をとわず, いかなる保証 (商品性及び特定の目的への適合性の黙示の保証を含むがこれらに限られない) も行わない。いかなるコンピューティングデバイスも絶対に安全ということはない。従って, Cypress のハードウェアまたはソフトウェア製品に講じられたセキュリティ対策にもかかわらず, Cypress は, Cypress 製品への権限のないアクセスまたは使用といったセキュリティ違反から生じる一切の責任を負わない。加えて, 本書面に記載された製品には, エラッタと呼ばれる設計上の欠陥またはエラーが含まれている可能性があり, 公表された仕様とは異なる動作をする場合がある。適用される法律により許される範囲内で, Cypress は, 別途通知することなく, 本書面を変更する権利を留保する。Cypress は, 本書面に記載のある, いかなる製品若しくは回路の適用又は使用から生じる一切の責任を負わない。本書面で提供されたあらゆる情報 (あらゆるサンプルデザイン情報又はプログラムコードを含む) は, 参照目的のためのみに提供されたものである。この情報で構成するあらゆるアプリケーション及びその結果としてのあらゆる製品の機能性及び安全性を適切に設計, プログラム, かつテストすることは, 本書面のユーザーの責任において行われるものとする。Cypress 製品は, 兵器, 兵器システム, 原子力施設, 生命維持装置若しくは生命維持システム, 蘇生用の設備及び外科的移植を含むその他の医療機器若しくは医療システム, 汚染管理若しくは有害物質管理の運用のために設計され若しくは意図されたシステムの重要な構成部分としての使用, 又は装置若しくはシステムの不具合が人身傷害, 死亡若しくは物的損害を生じさせるようなその他の使用 (以下「本目的外使用」という。) のためには設計, 意図又は承認されていない。重要な構成部分とは, それの不具合が装置若しくはシステムの不具合を生じさせるか又はその安全性若しくは実効性に影響すると合理的に予想できるような装置若しくはシステムのあらゆる構成部分をいう。Cypress 製品のあらゆる本目的外使用から生じ, 若しくは本目的外使用に関連するいかなる請求, 損害又はその他の責任についても, Cypress はその全部又は一部をとわず一切の責任を負わず, かつ Cypress はそれら一切から本書により免除される。Cypress は Cypress 製品の本目的外使用から生じ又は本目的外使用に関連するあらゆる請求, 費用, 損害及びその他の責任 (人身傷害又は死亡に基づく請求を含む) から免責補償される。

Cypress, Cypress のロゴ, Spansion, Spansion のロゴ及びこれらの組み合わせ, WICED, PSoC, CapsSense, EZ-USB, F-RAM, 及び Traveo は, 米国及びその他の国における Cypress の商標又は登録商標である。Cypress のより完全な商標のリストは, cypress.com を参照すること。その他の名称及びブランドは, それぞれの権利者の財産として権利主張がなされている可能性がある。