サイプレスはインフィニオン テクノロジーズになりました

この表紙に続く文書には「サイプレス」と表記されていますが、これは同社が最初にこの製品を開発したからです。新規および既存のお客様いずれに対しても、引き続きインフィニオンがラインアップの一部として当該製品をご提供いたします。

文書の内容の継続性

下記製品がインフィニオンの製品ラインアップの一部として提供されたとしても、それを理由としてこの文書に変更が加わることはありません。今後も適宜改訂は行いますが、変更があった場合は文書の履歴ページでお知らせします。

注文時の部品番号の継続性

インフィニオンは既存の部品番号を引き続きサポートします。ご注文の際は、データシート記載の注文部品番号をこれまで通りご利用下さい。

AN73212

PSoC® 1 によるデバッグ

著者: Dan Sweet

関連部品ファミリ: すべての PSoC 1 ファミリ

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/go/AN73212 へアクセスしてください。

本アプリケーション ノート (AN73212) は PSoC® 1 デバッガ システムの要素をご紹介し、それらを効果的に設定および使用する方法についてご説明します。いくつかの一般的なデバッグ技術をご説明し、スタック オーバーフローやメモリ破損などよくある問題の解決をご支援します。トラブルシューティング ガイドが含まれています。

目次

1 はじめに 1
2 PSoC リソース 3
2.1 PSoC Designer 3
2.2 サンプル コード 4
2.3 テクニカル サポート 5
3 デバッグ ハードウェアおよび設定 6
3.1 オンチップ デバッガ (OCD) デバイス 6
3.2 インサーキット エミュレータ (ICE) 6
3.3 デバッグ ポッド 6
3.4 デバッグ ハードウェアの設定 8
4 デバッグ環境 – PSoC Designer IDE 14
4.1 デバッガの起動 14
4.2 デバッグのコントロール 15
4.3 トレース 17
4.4 イベント ビューアー 18
4.5 マップ ファイル (.mp) 25
4.6 リスト ファイル (.lst) 27
5 PSoC 1 デバッグのヒントおよびコツ 29
5.1 停止中にハードウェアが動作 29
5.2 フラッシュ書き込みのデバッグ 29
5.3 スリープでのデバッガの使用 30
6 代替のデバッグ オプション 31
6.1 I2C-USB ブリッジによるデバッグ 31
6.2 UART インターフェースによるデバッグ 32
6.3 ピン トグル 32
7 トラブルシューティング 33
8 まとめ 34
A 付録 A: OCD デバイスのプリント基板への追加 35
B 付録 B: スタック オーバーフロー 36
B.1 ICE なしでのスタック オーバーフローのチェック 36
B.2 スタック オーバーフローの回避方法 36
C 付録 C: レガシー ハードウェア 39
C.1 ICE-4000 39
C.2 CY3240 I2USB ブリッジ キット 39
C.3 フレックス ポッド 39
改訂履歴 40
ワールドワイドな販売と設計サポート 41
製品 41
PSoC®ソリューション 41
サイプレス開発者コミュニティ 41
テクニカルサポート 41

1 はじめに

本アプリケーション ノートは、PSoC 1 内で利用可能なハードウェアおよびソフトウェア デバッガ要素を紹介し、いくつかの一般的なデバッグ技術について説明することを目的としています。

デバッグ システムの主要なハードウェア要素は、インサーキット エミュレータ (ICE) およびオンチップ デバッガ (OCD) が有効な PSoC 1 デバイスを備えたデバッグ ポッドです。これらの要素の詳細、およびその設定と使用方法は、本アプリケーション ノートのデバッグ ハードウェアの節に記載されています。

ソフトウェア要素は PSoC Designer®を中心としています。

この統合開発環境には、ブレークポイント、ウォッチ変数、メモリ ビューアー、トレース、イベント、出力ファイル (リストおよびマップ) を含む、デバッグ用のツールが多くあります。各要素は本アプリケーション ノートの「デバッグ環境 – PSoC Designer IDE」節で説明されています。
また、以下のトピックについてもご説明します。

- PSoC 1 デバッグのヒントおよびコツ
- 代替のデバッグ オプション
- トラブルシューティング

2 PSoC リソース

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対して適切な PSoC デバイスを選択し、迅速かつ効率的にデバイスをデザインに統合する手助けをしています。本資料では、PSoC は PSoC 1 デバイス ファミリを示します。PSoC 1 の詳細については、アプリケーション ノート「AN75320 - *Getting Started with PSoC 1*」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。PSoC Designer にはデバイス選択ツールも含まれています。
- **データシート:** PSoC 1 デバイス ファミリの電気的仕様を説明します。
- **アプリケーション ノートおよびサンプル コード:** 基本レベルから上級レベルまでの幅広いトピックを提供します。多くのアプリケーション ノートはサンプル コードを含んでいます。
- **テクニカル リファレンス マニュアル (TRM):** PSoC 1 デバイスの内部アーキテクチャの詳細を説明します。
- **開発キット:**
  - CY3215A-DK In-Circuit Emulation Lite Development Kit には、インサーキット エミュレータ (ICE) が含まれています。ICE-Cube は主に PSoC 1 デバイスのデバッグに使用しますが、ISSP を使用して PSoC 1 デバイスのプログラムもできます。
  - CY3210-PSOCEVAL1 キットにより、サイプレスの PSoC 1 プログラマブル システムオンチップの設計手法とアーキテクチャの評価および実験が可能です。
  - CY8CKIT-001 は、すべての PSoC ファミリ デバイスの共通開発プラットフォームです。
- MiniProg1 および MiniProg3 デバイス: フラッシュ メモリのプログラミング用のインターフェースを提供します。

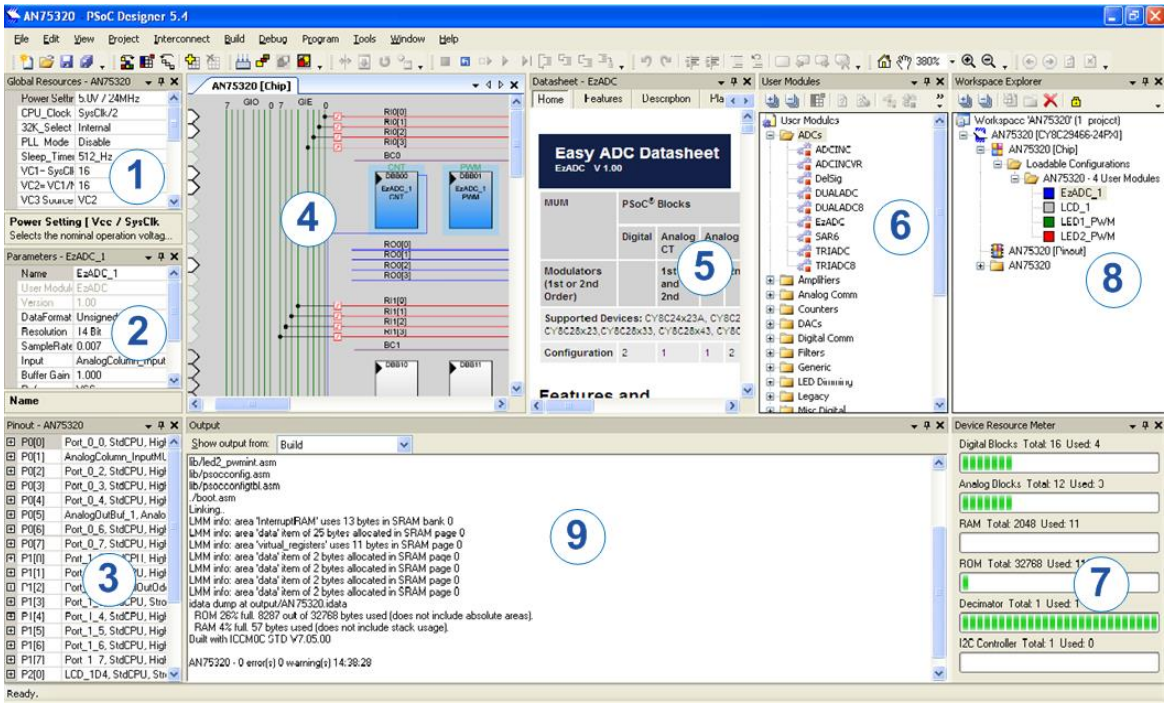
2.1 PSoC Designer

PSoC Designer は無償の 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、特性化済みのアナログとデジタル ペリフェラルのライブラリを使用してドラッグ アンド ドロップの設計環境で行われます。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 は 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 –** 選択しているユーザー 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 –** 選択しているユーザー 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 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 > 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。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/go/PSoC1Code Examples

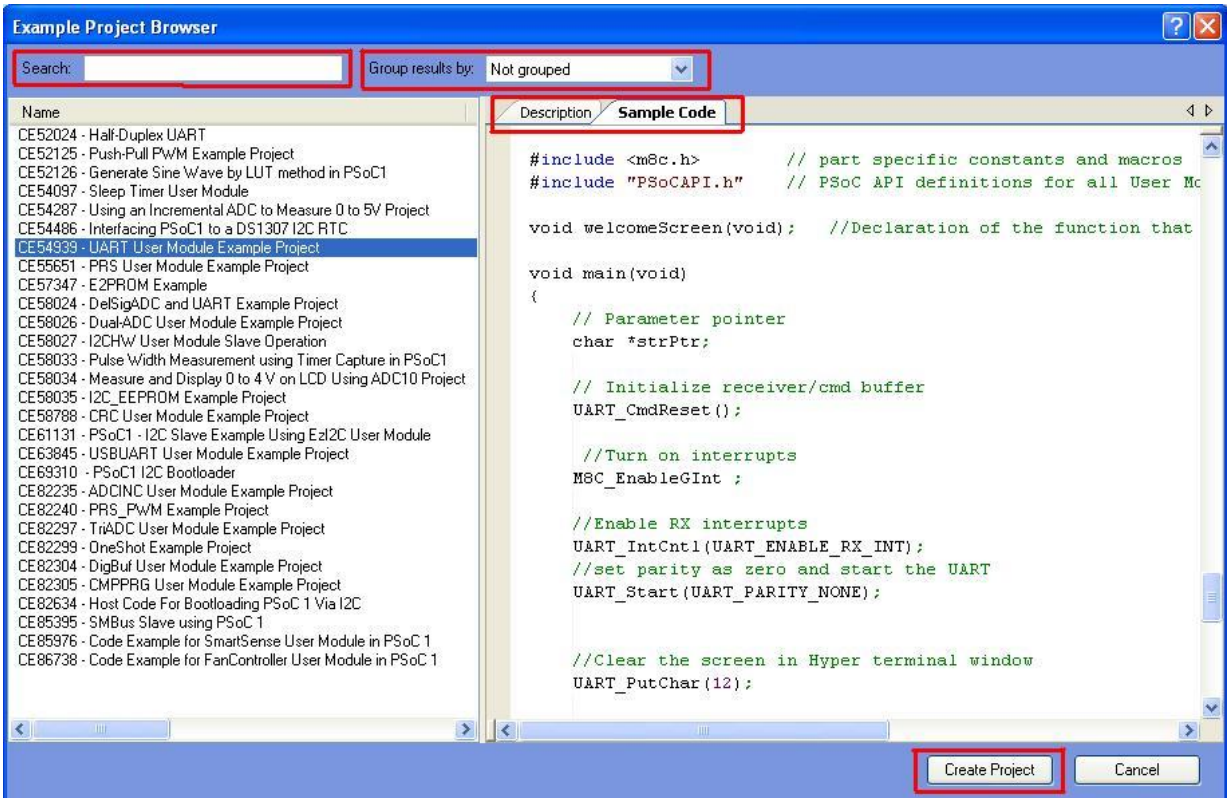
PSoC Designer に統合されているサンプル コードへアクセスするには、図 2 に示すように **Start Page > Design Catalog > Launch Example Browser** を順に選択してください。

図 2. PSoC Designer 内のサンプル 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 タブ内)
- 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コピーしプロジェクトに貼り付けてコード開発時間を短縮させることができます。
- 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から始めることで設計時間を短縮させます。その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 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。

早急な対応が求められる場合には、下記のリソースをご利用ください。

- セルフ ヘルプ
- 所在地の販売代理店

3 デバッグ ハードウェアおよび設定

本節では、PSoC 1 デバイスを使ってデバッグ システムを設定するのに必要なハードウェアについて説明します。ICE、デバッグ ポッド、適切なコネクタなどが含まれます。デバッグ ハードウェア設定の詳細については、AN73212 ウェブページに掲載されているビデオをご覧ください。以下の節では、各要素の詳細について説明します。

3.1 オンチップ デバッガ (OCD) デバイス

すべての PSoC 1 チップがデバッグをサポートするわけではありません。しかし、各 PSoC 1 デバイス ファミリには、デバッグをサポートするデバイスが少なくとも 1 バージョン用意されています。これらのデバッグ対応 PSoC デバイスは OCD デバイスと呼ばれています。特定のデバイス ファミリの OCD デバイスは、そのファミリ内のあらゆる製品番号をエミュレートできます。例えば、CY8C27002-24PVXI OCD デバイスは CY8C27xxx シリーズの任意のデバイスをエミュレートできます。

一部のデバイス ファミリには、異なるパッケージを提供する複数の OCD デバイスがあります。また別のファミリには、ファミリ全体向けに 1 つの OCD デバイスしかない場合があります。各 PSoC 1 デバイス ファミリの OCD デバイスの製品番号とピン配置については、デバイス データシートをご参照ください。

3.2 インサーキット エミュレータ (ICE)

インサーキット エミュレータ (ICE) 、すなわち ICE-Cube は、PSoC 1 デバッグ システムの主要な要素です。ICE は、PC (PSoC Designer) 上で実行中のデバッガ ソフトウェアと PSoC OCD チップ間のエミュレーション通信をすべて管理します。ICE は図 4 に示します。

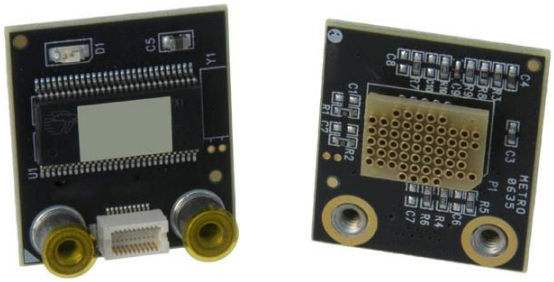
図 4. インサーキット エミュレータ (ICE)

ICE は、デバッグを行なうために必要な全アクセサリと共に CY3215A-DK 開発キットに同梱されています。

3.3 デバッグ ポッド

デバッグ ポッドは、PSoC 1 デバッグ システムにおけるもう 1 つの重要な要素です。一般的にポッドとは、小さなプリント基板であり、オンチップ デバッガ (OCD) 対応 PSoC 1 チップ、ICE コネクタ、プリント基板を既存の基板／開発キットに接続させるコネクタを搭載しています。デバッグ ポッドには、CY3250 ポッド (図 5) および CY3210 評価用ポッド (図 6) の 2 つの主な種類があります。

図 5. CY3250 ポッドの例



CY3250 ポッドは、新規または既存のカスタム プリント基板に一時的にデバッグ機能を追加するのに最適です。CY3250 ポッドは OCD PSoC デバイス、ICE コネクタ、フットキットを通して基板に搭載できるコネクタ、および電源供給用の受動素子数個から成ります。

使用可能な CY3250 ポッドの一覧については、「PSoC® 1 Kit Selector Guide」またはご使用の PSoC 1 デバイスのデータシートに記載されているエミュレーションおよびプログラミング アクセサリ表をご参照ください。

CY3210 評価用ポッドは、評価用キットまたは 28 ピン DIP コネクタ付きのその他基板をプロトタイプ作成する場合に最適です。CY3210 評価用ポッドはほとんどの PSoC 1 デバイス ファミリに利用可能です。各評価用ポッドには、プロトタイピングのために OCD 対応 PSoC 1 デバイス、基板搭載用の 28 ピン PDIP コネクタ、ICE 接続用の RJ-45 コネクタ、5 ピン プログラミング ヘッダ、ピンへのアクセスを容易にするコネクタが同梱されています。一般的には、評価用ポッドは CY3210-PSoCEval1 開発キット、ブレッドボード、28 ピン PDIP コネクタ付きのその他基板と併用するのが最適です。

図 6. CY3210 評価用ポッドの例

表 1 に CY3210 評価用ポッドを一覧表示しています。最も一般的なファミリ向けには CY3210 評価用ポッドが用意されています。

表 1. CY3210 評価用ポッド一覧

デバイス ファミリ	評価用ポッド製品番号
CY8C21x23	CY3210-21x23
CY8C24x23	CY3210-24X23
CY8C24x94	CY3210-24x94
CY8C27xxx	CY3210-27x43
CY8C28xxx	CY3210-28xxx
CY8C29xxx	CY3210-29x66

ポッド利用の代替として、OCD デバイスと ICE コネクタを新しい PCB へと直接搭載することが可能です。直接搭載は、ポッドがプリント基板にうまくはまっていない場合、または対象の PSoC 1 デバイスに利用可能なポッドがない場合に有用です。デバッグ機能をプリント基板に直接追加する詳細については、「付録 A – OCD デバイスのプリント基板への追加」をご参照ください。

3.3.1 フット

CY3250 スタイルのポッドを基板上に正しく装着するには、フットキットおよびマスクが必要です。フットキットを使用することで、ポッドを特定のパッケージ実装面積に搭載できるため、単一のポッドをほとんどあらゆる基板に搭載することが可能となります。CY3250 ポッドと ICE を備えたフットキットにより、デバッグに対応できるように既存のプリント基板を変更することができます。適切な実装面積を有するフットキットは、既存のプリント基板にはんだ付けが可能であり、ポッドをそのフット上に搭載することができます。図 7 に、3 種類のフットの例を示します。

図 7. フットの例

図 7 に示すように、フットの下半分はプリント基板のパッケージ実装面積に適合します。この図は、28 ピン SOIC、44 ピン TQFP、28 ピン PDIP を示しますが、あらゆる PSoC 1 パッケージ サイズおよびタイプ向けのフットキットが入手可能です。

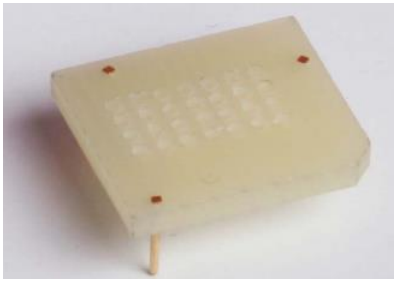
フットキットの上半分は、対応するポッドのピンに接続します。ピンの数はパッケージによって異なっており、各フットキットは OCD デバイスの必要なピンに接続します。例えば、ポッド上の OCD デバイスが 56-QFN パッケージで提供されている場合、28 ピンのみが利用可能な製品をエミュレートするには、利用可能な 56 ピンの内 28 ピンのみにフットキットを接続させます。

デバイス データシートに記載のエミュレーションおよびプログラミング アクセサリ表に、各 PSoC デバイス パッケージに適したフットキットが列挙されています。

3.3.1 マスク

マスクは、ポッドとフットキット間に装着され、フットキットからポッドの適切な穴へとピンを配列して、未使用のピンをマスクします。一部のポッドは多数の接続点があり、一部のフットには少ししかありません (例えば、8 ピン パッケージなど)。このため、マスクを使用することで、フットキットをポッド コネクタに適切に配列することができます。図 8 にマスクの例を示します。

図 8. マスクの例



あらゆるフットとポッドの組合せに、マスクが必要となるわけではありません。理由は、一部のポッドは、フットキットに一方向からしか装着できないためです。例えば、QFN ベースのポッドを QFN フットキットに接続するために、マスクは必要ありません。マスクの一般的なサイズは、28 ピン フット (DIP、SOIC、SSOP パッケージ) 対応のマスク、8 ピンまたは 20 ピン フットキットに使用できるマスクの 2 つがあります。44 ピンまたは 48 ピンのフットには、マスクは必要ありません。

必要に応じて、適切なマスクが CY3250 ポッドまたはフットキットに同梱されているため、別々に購入する必要はありません。

3.4 デバッグ ハードウェアの設定

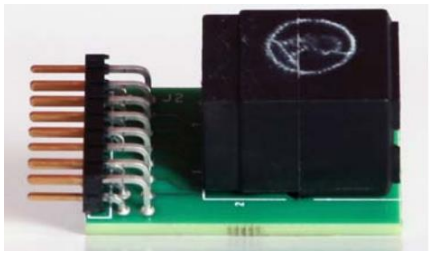
本節では、さまざまなハードウェア部材を接続し、機能的なデバッグ システムを構築する方法を説明します。使用するポッド、フットキット、マスクにより、多少設定が異なる場合があります。以下に、一般的な設定の数例を示します。

3.4.1 ICE + CY3210 評価用ポッド

ICE を CY3210 評価用ポッドに接続するには、以下の部材が必要です。

1. ICE
2. 対象となる PSoC 1 ファミリ向けの CY3210 評価用ポッド
3. 図 9 に示す RJ-45 ICE アダプタは、一部の従来のポッドでも利用できるため、「下位互換性アダプタ」とも呼ばれています。

図 9. RJ-45 ICE アダプタ



4. 長さが 12 インチ以下の短めの RJ-45／イーサネット ケーブル (CY3215A-DK キットに同梱されています)。ICE とポッド間の通信が干渉される場合があるため、長めのケーブルで代用することはできません。

図 10. RJ-45／イーサネット ケーブル

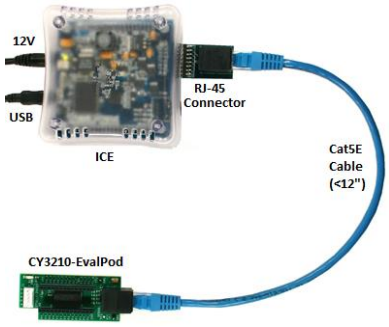
交換用のケーブルが必要な場合は、12 インチ以下の標準 Cat5E ケーブルを利用してください。

注: ICE 通信は標準のイーサネット プロトコルではありません。カスタムの通信プロトコルが使用されています。

5. USB ケーブルおよび 12V 電源 (どちらも ICE CY3215A-DK に同梱されています)。

   上記部材の適切な接続は、図 11 に示されています。オプションとして、評価用ポッドを 28 ピン DIP コネクタ付き基板へ差し込むこともできます。

図 11. ICE + CY3210 評価用ポッドの設定



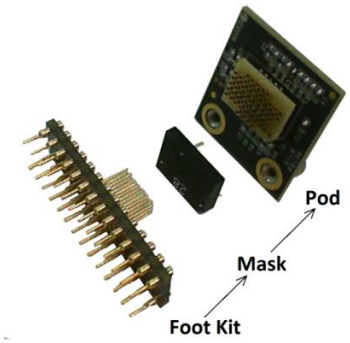
3.4.2 ICE + CY3250 ポッド

ICE を CY3250 ポッドに接続する手順は以下の通りです。

1. 対象回路の PSoC デバイスのピン配置に一致するフットを選択します。
2. 必要に応じて、所望のフットに一致するマスクを選択します。通常、マスクが必要なのは 8 ピン、20 ピン、28 ピンのフットのみです。
3. マスクをポッド下部に挿入し、マスクの面取りした角をポッド上のピン 1 の三角形に配列します。
4. フットをマスクに挿入します。マスクを使用しない場合、フットをポッドの下部に直接接続させます。

   図 12 にステップ 2～4 を示します。

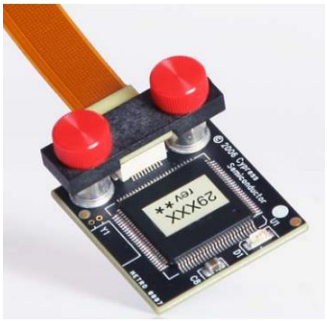
図 12. フットキット + マスク + ポッド



5. 組み立てたポッドを対象の基板に接続します (フットキットが基板に未装着である場合)。
6. ポッドを、CY3250 フレックス ケーブルの小さいほうの端に接続します。図 13 に、ポッド、ケーブルおよびコネクタを示します。図 14 に、完成した接続を示します。

図 13. CY3250 ポッド、フレックス ケーブル、コネクタ

図 14. フレックス ケーブル接続



7. 図 15 に示すように、フレックス ケーブルのもう 1 つの端を ICE に差し込みます。

図 15. CY3250 ポッドに接続された ICE

8. 最後に、ICE を USB ケーブルで PC に接続し、付属の 12V DC 電源を通して ICE に電源を供給します。

表 2 に、デバッグ プロセス中に使用するポッド コネクタのピンの名称および説明を示します。

表 2. ポッド コネクタのピン説明

ピン番号	ピン名	ピンの説明
1	POD EXTRA3	将来の使用
2	GND	グランド
3	—	—
4	POD_OCDDE	POD_OCD 偶数データ入出力
5	GND	グランド
6	POD_OCDDO	POD_OCD 奇数データ出力
7	POD EXTRA1	将来の使用

ピン番号	ピン名	ピンの説明
8	POD_XRES	リセット信号 (リセット プログラミング モードにのみ必要)
9	GND	グランド
10	POD_OCDHC	POD_OCD 高速クロック出力
11	GND	グランド
12	POD_OCDCC	POD_OCD CPU クロック出力
13	POD EXTRA4	将来の使用
14	PODVCC	電源電圧
15	—	—
16	PODVCC	電源電圧

3.4.3 ICE + 基板搭載 OCD デバイス

場合によっては、デバッグ用ポッドを利用する代わりに、OCD デバイスを対象基板に直接搭載することもできます。基板上にも搭載されているコネクタへ、適切なデバッガ ラインを引き出します。これは、対象の PSoC 1 デバイス向けに使用できるポッドがない場合、または対象のプリント基板にうまくはまっていない場合に行ないます。

OCD デバイスを基板上に直接装着すると、よりコスト効率よくデバッグを有効にできる場合があります。いくつかの PSoC 1 開発基板では、この方法を利用してデバッグを有効にしています。これらの基板は一般的に、基板搭載の RJ-45 コネクタにより識別されます。

OCD インターフェース信号: ICE は M8C マイクロコントローラーの正確な (エミュレーション) コピーを含んでいます。ICE は両方のマイクロコントローラー (M8C とそのコピー) を同時に起動し、それらを同期させながら実行させます。M8C は、M8C と ICE が同時に動作できるほど速く IO データを ICE に送信する必要があります。ICE は 8 線式インターフェース (RJ-45 コネクタ) を介してリモート M8C と通信します。表 3 に、デバッグ プロセス中に使用する OCD インターフェースの信号を示します。

表 3. OCD インターフェース信号

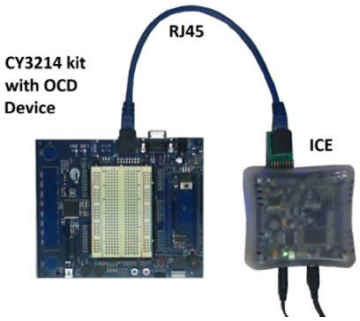
ピン番号	ICE からの方向	信号名	説明
1	入力	U_HCLK	リモートM8Cによって駆動される24／48MHzデバッグ クロック。このクロックは、ICE／M8C通信のステートマシンを駆動するために使用。このクロックは、U_CCLKクロックが24MHzで動作して48MHzに切り替わる場合を除き、常に24MHzで動作
2	出力	ICE_POD_GND	リモート M8C へのグランド信号
3	出力	ICE_POD_RST	リモート M8C へのアクティブ HIGH リセット信号
4	出力	ICE_POD_GND	リモート M8C へのグランド信号
5	入力	U_CCLK	内部 M8C CPU クロック
6	入力	U_D1_IRQ	IOデータをICEに送信するためにM8Cによって使用される2線のデータ ラインの1線。このラインは、保留中の割り込みをICEに通知するためにも使用。M8Cのみによって駆動される
7	入出力	U_D0_BRQ	IOデータをICEに送信するためにM8Cによって使用される2線のデータ ラインの1線。ICEはこのラインを使用して中止要求を送信
8	出力	ICE_POD_PWR	リモート M8C へのオプションの電源

a. RJ45 コネクタのピン番号。

OCD デバイスを基板へ直接搭載する方法の詳細については、「付録 A – OCD デバイスのプリント基板への追加」をご参照ください。図 29 に、基板搭載 OCD デバッガの回路図を示します。

ICE を OCD デバイス搭載の基板と接続することは、前述の ICE を CY3210 評価用ポッドと接続するプロセスと似ています。唯一の違いは、ケーブルを評価用ポッドではなく、対象基板の RJ-45 コネクタに接続することです。図 16 にその一例を示します。

図 16. ICE + 基板搭載 OCD デバイス付きの CY3214



3.4.4 ICE を用いたデバイス プログラミング

ICE はまた、CY3215A-DK に同梱されている 5 ピン インシステム シリアル プログラミング (ISSP) ケーブルを利用して PSoC デバイスをプログラムすることも可能です。

その手順は以下の通りです。

1. ICE を RJ-45 アダプタに接続します。
2. ISSP ケーブルを RJ-45 アダプタに接続します。ケーブル (通常黄色または黒色) の一方の端に RJ-45 コネクタがあり、もう片方の端には白い 5 ピン コネクタがあります。

図 17. ICE ISSP ケーブル

3. 5 ピン メスコネクタを、対象基板上の 5 ピン プログラミング ヘッダに接続します。図 18 に、ISSP 向けに設定された対象基板と ICE を示します。

図 18. プログラミング用に設定された ICE

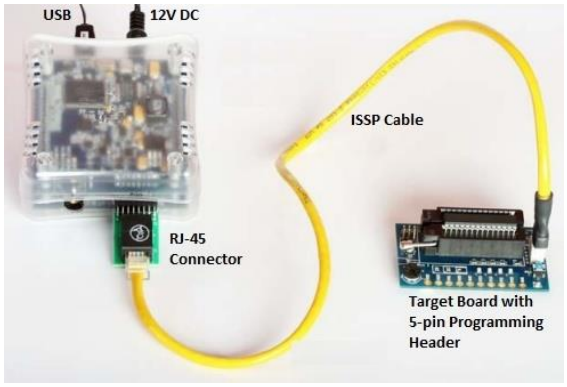
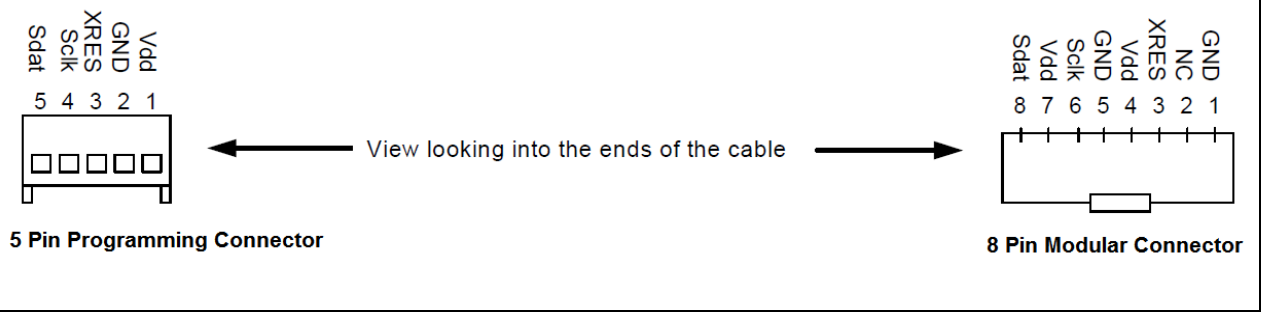


表 4. 5 ピン プログラマ コネクタと 8 ピン モジュラ コネクタ

信号	5 ピン プログラマ コネクタ	8 ピン モジュラ コネクタ
V_{dd}	ピン 1	ピン 4 とピン 7
GND	ピン 2	ピン 1 とピン 5
XRES	ピン 3	ピン 3
Sclk	ピン 4	ピン 6
Sdat	ピン 5	ピン 8



注: このコンフィギュレーションでは、デバッグは行えません。5 ピン ISSP ケーブルは、デバッグ用に使用するピンとは異なる、デバイスのプログラミング ピンに接続されています。

4 デバッグ環境 – PSoC Designer IDE

PSoC Designer には、コード開発ツールとデバッグ ツールの強力なセットが含まれています。デバッグ ツールは、本アプリケーション ノート「デバッグ ハードウェアの設定」節で説明されているハードウェア コンフィギュレーションのいずれか、および ICE と併用できます。

PSoC Designer 内の重要なデバッグ要素を、以下で簡単に説明します。各機能の追加情報については、IDE ユーザー ガイドのデバッガの節をご参照ください。IDE ガイドは、ここをクリックしてダウンロードする、または **PSoC Designer > Help > Documentation > Designer Specific Documents > IDE User Guide.pdf** にてご覧ください。

4.1 デバッガの起動

PSoC Designer を ICE に接続してデバッグを開始するには、ICE と基板に電力が供給されており、ICE が USB ケーブルで PC に接続されていることをご確認ください。対象基板は個別に、または ICE 経由で電源供給することが可能です。対象基板に電源供給する方法を制御するには、PSoC Designer 内のデバッガ設定を利用します。デバッガ設定には、図 19 に示すように、**Project -> Settings -> Debugger** ウィンドウを通してアクセスできます。

図 19. デバッガ プロジェクト設定

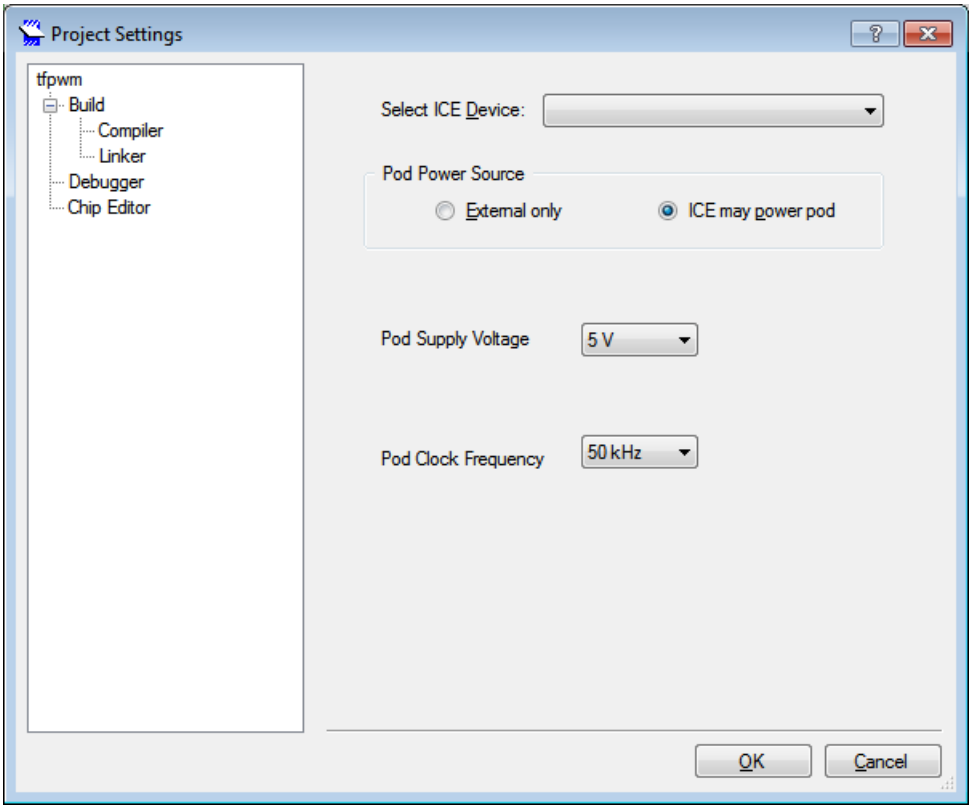


図 19 の設定により、複数の ICE デバイスが単一 PC に接続されている場合に、どの ICE に接続するかを選択することができます。また、この設定により、ポッドに電源供給するために ICE を設定することもできます。ICE からポッドに電源供給する場合、ポッドの電源電圧も指定しなければなりません。利用可能な供給電圧は 5.0V および 3.3V のみです。

ICE を PC に接続し、適切な設定を適用すると、システムのデバッグの準備は完了します。PSoC Designer ICE ツールバーは ICE に接続するためのコントロールを提供します。その機能は表 5 に記載されています。

表 5. ICE 接続のコントロール

アイコン	機能	説明
	接続	PSoC Designer を付属の ICE Cube に接続
	ダウンロード	コンパイルされたコードを ICE および OCD デバイスにダウンロード
	M8C ビューのリフレッシュ	PSoC Designer 内に表示されるメモリおよび可変ディスプレイを更新
	次のトレース データの取得	コード実行に関する情報の 64 行をトレース ウィンドウに追加。詳細は、「トレース」節をご参照ください

ICE への接続およびプログラムのダウンロードに問題がある場合は、本書後述の「トラブルシューティング」節をご参照ください。

4.2 デバッグのコントロール

PSoC Designer には、付属の ICE および OCD PSoC 上のコード実行を制御するための多数のコマンドがあります。このデバッガは、表 6 に示すように、プログラム フローを制御するいくつかの標準方法をサポートしています。

表 6. 基本的なデバッグ コントロール

アイコン	機能	説明
	起動	デバッガを起動
	カーソル位置まで実行	ソース コードにおける現時点のカーソルの位置で、一時的 (非表示) ブレークポイントを作成し、そのポイントまでアプリケーションを実行
	停止	デバッガを停止
	リセット	デバイスを PC 値「0」にリセットし、デバッガを再起動
	ステップイン	次の命令文にステップイン
	ステップアウト	現時点の関数からステップアウト
	ステップオーバー	次の命令文にステップオーバー
	ステップ ASM	アセンブリ コードの 1 行を実行。現時点のコード行が C コードである場合、リスト ファイルが開かれ、アセンブリの単一行が実行
	プログラム実行	ICE に接続し、プログラムをダウンロードしてその実行を開始

ブレークポイントは、PSoC Designer と ICE のもう 1 つのデバッグ ツールです。ブレークポイントはコード内のマーカーであり、到達したら実行を一時停止する場所を、マークされた行で示します。ブレークポイントを追加するには、コード行の脇の余白を左クリックするか、対応する行上のコード エディタを右クリックし、「Insert Breakpoint」を選択します。図 20 は、ブレークポイントが挿入されたコード行を示します。また、「ブックマーク」(行 15) も表示されています。これは、コード行の脇の余白を右クリックして追加できます。ブックマークは、コード実行には影響を及ぼしませんが、コード内の関心対象のポイントをマークするために有用です。

図 20. ブックマークとブレークポイントの例

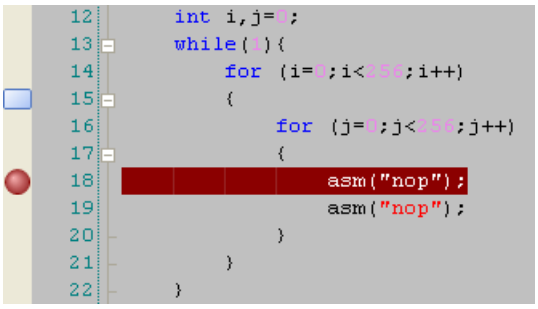
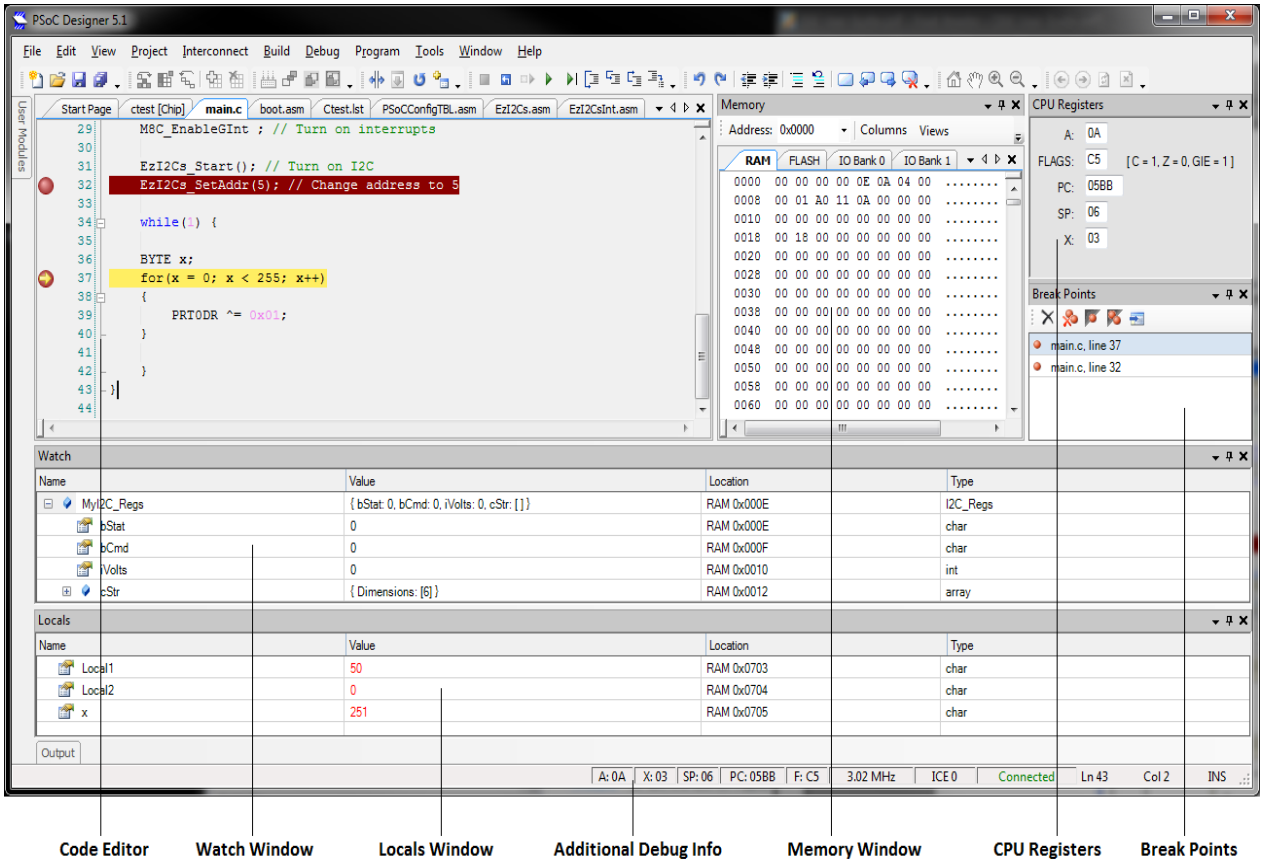


図 21 は、PSoC Designer でのデバッグ セッション中に利用可能な多数のデバッグ ウィンドウを示します。示された各ウィンドウはオプションであり、好みに応じて移動やサイズ変更が可能です。以下は各ウィンドウの簡単な説明です。また、IDE ユーザーガイドには、各ウィンドウについての追加説明が記載されています。IDE ガイドは、ここをクリックしてダウンロードする、または **PSoC Designer > Help > Documentation > Designer Specific Documents > IDE User Guide.pdf** にてご覧ください。

図 21. PSoC Designer デバッグ ウィンドウ



コード エディタ ウィンドウ: コード エディタ ウィンドウはデバッグ セッション中のコードを表示します。このウィンドウはまた、ブレークポイントの位置する場所、およびコード実行が停止した場所 (プログラムが実行中でない場合) を表示します。デバッグ中にコードを編集することはできません。デバッガ稼動中にコードを編集した場合、PSoC Designer はデバッグ モードを終了し、その変更がコードに反映されるようにします。

Watch ウィンドウ: Watch ウィンドウは、デバッグ セッション中にユーザーが追加したウォッチ変数を表示します。Watch ウィンドウに変数を追加するには、デバッグ セッション中にコード エディタ ウィンドウを右クリックし、**Add Watch** をクリックします。

Locals ウィンドウ: Locals ウィンドウは、プログラムが停止した時に範囲内にあるローカル変数を表示します。現時点で範囲内にあるローカル変数は自動的に追加されます。

追加のデバッグ情報: PSoC Designer 画面の下部には、常に有用なデバッグ情報が表示されています。以下のようなものが挙げられます。

- アキュムレータ (A) の現時点の値
- X レジスタ (X)
- スタック ポインター (SP)
- プログラム カウンター (PC)
- フラグ レジスタ (F)

ICE 接続速度と接続ステータスについての情報も表示されます。

Memory ウィンドウ: Memory ウィンドウによりユーザーは、停止中のチップ上のさまざまなメモリ領域を検証できます。各領域 (RAM、フラッシュ、IO バンク 0、IO バンク 1) について、Memory デバッグ ウィンドウ内に独自のタブが用意されています。RAM、IO バンク 0、IO バンク 1 は、アクティブなデバッグ セッション中に編集することが可能です。

CPU Registers ウィンドウ: CPU Registers デバッグ ウィンドウは、PSoC デバイスのさまざまな内部 CPU レジスタを表示します。表示されるレジスタは、アキュムレータ (A)、フラグ レジスタ、プログラム カウンター (PC)、スタック ポインター (SP) および X レジスタです。CPU Registers ウィンドウ内に表示される値は、デバッグセッションがアクティブな間に編集可能です。

Breakpoints ウィンドウ: Breakpoints ウィンドウは、プロジェクト内のすべての配置済みブレークポイントを、それぞれのファイルおよび行番号を含めて表示します。個別またはすべてのブレークポイントを、このウィンドウから削除、無効化、または有効化できます。

4.3 トレース

PSoC Designer および ICE デバッガ内のトレース ウィンドウでは、命令ごとにデバイス アクティビティを追跡／ログ記録できます。このウィンドウは、デバイスの内部動作の継続的かつ設定可能なリストを表示します。プログラム実行が開始されるたびに、トレース バッファはクリアされます。バッファが満杯になった場合、継続して動作し、古いデータを上書きします。

利用可能なトレース モードは 3 つあり、以下で説明します。

トレース モード 1 - PC のみ – PSoC によって実行された各命令のプログラム カウンター (PC) とアセンブリ命令を一覧表示します。

図 22. トレース モード 1 - PC のみ

Trace.txt

```
1   PC / SYMBOL          INSTRUCTION
2   0456                 NOP
3   0455                 NOP
4   0454                 NOP
5   0453                 NOP
6   039C                 RETI
7   0390                 JNC 039Ch
8   038E                 INC [0006h]
9   038A                 JZ 038Eh
10  0387                 TST [0001h], FFh
11  0383                 JZ 0387h
12  0380                 TST [0002h], FFh
```

トレース モード 2 - PC／レジスタ – PC、命令、データ、アキュムレータ (A)、X レジスタ、スタック ポインター (SP)、フラグ レジスタ、SRAM ページを一覧表示します。

図 23. トレース モード 2 - PC／レジスタ

Trace.txt

	PC / SYMBOL	INSTRUCTION	DATA	A	X	SP	FLAGS	SRAM PAGE
2	0456	NOP	18	18	5B	02	C5	00
3	0455	NOP	18	18	5B	02	C5	00
4	0454	NOP	18	18	5B	02	C5	00
5	0453	NOP	18	18	5B	02	C5	00
6	039C	RETI	18	18	5B	02	C5	07
7	0390	JNC 039Ch	18	18	5B	05	00	00
8	038E	INC [0006h]	03	18	5B	05	00	00
9	038A	JZ 038Eh	18	18	5B	05	02	00
10	0387	TST [0001h], FFh	18	18	5B	05	02	00
11	0383	JZ 0387h	18	18	5B	05	02	00
12	0380	TST [0002h], FFh	18	18	5B	05	02	00

トレース モード 3 - PC／タイムスタンプ – PC、命令、アキュムレータ (A)、SRAM ページ、タイムスタンプを一覧表示します。プログラムで実行された CPU サイクルの集計を続けるタイムスタンプは、コードのセクションにより実行が必要な CPU サイクルをカウントする効率的な方法です。

図 24. トレース モード 3 - PC／タイムスタンプ

Trace.txt

	PC / SYMBOL	INSTRUCTION	A	SRAM PAGE	TIMESTAMP
2	0456	NOP	18	00	262
3	0455	NOP	18	00	258
4	0454	NOP	18	00	254
5	0453	NOP	18	00	250
6	039C	RETI	18	07	246
7	0390	JNC 039Ch	18	00	236
8	038E	INC [0006h]	18	00	231
9	038A	JZ 038Eh	18	00	224
10	0387	TST [0001h], FFh	18	00	219
11	0383	JZ 0387h	18	00	211
12	0380	TST [0002h], FFh	18	00	206

トレース バッファ サイズは 256kB です。これにより、128k トレース命令はトレース モード 1 で追跡可能となり、32k トレース命令はトレース モード 2 と 3 で追跡可能となります。

トレース ウィンドウは、**Debug->Windows->Trace** メニューで開きます。トレース モードは、**Debug->Trace Mode** メニューを使用して変更可能です。一度トレース ウィンドウを開くと、プログラムの停止時に、最新の 64 のトレース エントリがトレース ウィンドウにロードされます。64 の追加エントリをロードするには、「Get Next Trace Data」ツールバーまたはメニュー オプションを使用してください。または、「Get all Trace Data」メニュー オプションを使用してすべての使用可能なトレース データをロードできます。

4.4 イベント ビューアー

イベント ビューアーを使用し、満たされた場合にブレークポイントまたはトレースをトリガーする条件か条件のシーケンスを定義できます。その結果、標準のデバッグ ツールでは対処できない、より複雑なコード シーケンスを検証およびデバッグできます。

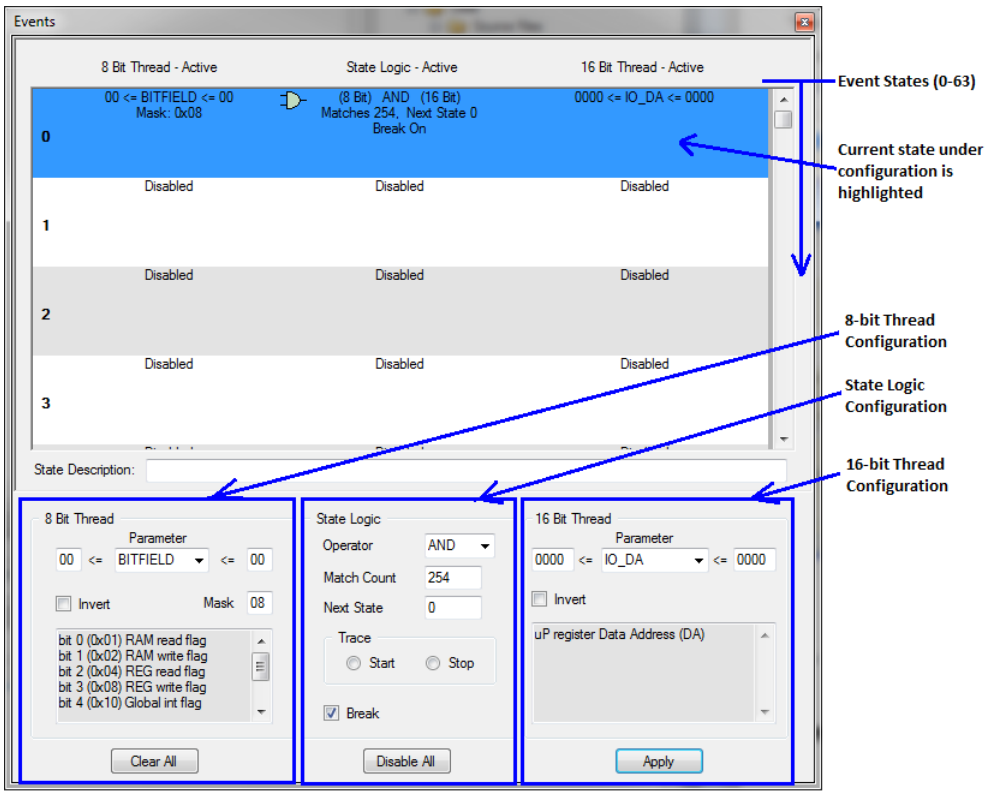
多くのデバッグ状況で役立つイベントを設定することができます。イベントの設定は、標準の if-then 文と比較が可能です。まず、比較を行ないます。その比較が真と評価される場合、措置を取ることができます。この措置は、ブレークポイントの設定、トレース バッファの開始または停止、別の連鎖された if-then 文への移動である場合があります。最大 64 の文を定義可能です。

文の「if」の部分は、8 ビットまたは 16 ビットのスレッドで作られます。各スレッドにより、RAM アドレス、PC 値、スタック値などいくつかの使用可能な 8 ビットまたは 16 ビットのソースで比較が可能となります。

「if」文に柔軟性を持たせるために、8 ビットおよび 16 ビットの比較を個別に使用することができます。または、論理関数 (OR、AND、NOR、NAND) 経由で組み合わせることが可能です。さらに、8 ビットまたは 16 ビット スレッドそれぞれの出力論理を反転し、アクティブ LOW またはクリアされたデータの確認をより良くサポートすることも可能です。

図 25 に示すイベント デバッグ ウィンドウは、PSoC Designer 内で **Debug > Windows > Events** メニューからアクセスします。このウィンドウには、利用可能な各 64 イベント状態を選択および表示するためのセクションがあります。特定のイベント状態を設定するには、単にウィンドウ上部からそれを選択してください。下部では状態セットの設定パラメーターを設定します。利用可能な各設定の詳細は、以下の節で説明します。

図 25. イベント デバッグ ウィンドウ



以下で、イベントの使用可能な 8 ビット スレッド、16 ビット スレッド、論理比較および出力の各オプションについてご説明します。

4.4.1 8 ビット スレッド オプション

8 ビット スレッドでは、PSoC 内の 8 ビット幅パラメーターを評価する方法をいくつか使用できます。比較は、特定の値または 0x00～0xFF の値の範囲に対して行えます。

Invert: このオプションでは、8 ビット パラメーター比較の結果を反転されることができます。

Mask: マスク フィールドは、入力データに対してパラメーター比較を行う前に、特定のビットをマスクすることができます。このオプションは、パラメーター選択値の特定のビットのみに関心がある場合に有用です。

Parameters: パラメーター ドロップダウン メニューでは、8 ビット比較で使用するフィールドを選択できます。利用可能な 8 ビット パラメータ比較オプションは以下の通りです。

NONE: 8 ビット スレッド比較を無効にします。

A: アキュムレータ比較。アキュムレータ内のデータを評価します。

IO_DA: レジスタ アドレス比較。レジスタ メモリ空間内のデータ アドレスを評価します。

IO_DB: レジスタ データ比較。レジスタ メモリ空間内のデータ バス上のデータを評価します。

IR: 命令レジスタ比較。命令レジスタ内の命令オペコードを評価します。

MEM_DA: RAM アドレス比較。RAM メモリ空間内のアドレスを評価します。

MEM_DB: RAM データ比較。RAM メモリ空間内のデータ バス上のデータを評価します。

SP: スタック ポインター比較。スタック ポインターの現時点のアドレス (SP が指しているデータではない) を評価します。

X: X レジスタ比較。X レジスタ内の現時点の値を評価します。

BITFIELD: 書き込みまたは読み出しを実行する場合や、ゼロまたはキャリー フラグがセットされている場合など、チップの内部動作について特定フラグ チェックを可能にします。使用可能な BITFIELD チェックは以下の通りです。複数のフラグを同時にチェック可能です。

ビット 0 (0x01): RAM 読み出しフラグ

ビット 1 (0x02): RAM 書き込みフラグ

ビット 2 (0x04): レジスタ読み出しフラグ

ビット 3 (0x08): レジスタ書き込みフラグ

ビット 4 (0x10): グローバル割り込みフラグ

ビット 5 (0x20): ゼロ フラグ

ビット 6 (0x40): キャリー フラグ

ビット 7 (0x80): 拡張 IO フラグ

BITFIELD 比較オプションは通常、追加の比較 (または比較のシーケンス) と組合わせます。例えば、ビット 1 (RAM 書き込みフラグ) は、特定の値が RAM ロケーションにいつ書き込まれるかを確認するために RAM アドレスおよび／または RAM データ比較 (16 ビット スレッドで) と共によく使用されます。RAM アドレスおよび RAM データ比較はこれらのタスクを独立して実行しますが、最初の書き込みが行われた後でも真として評価し続けます (デバッグ状況によっては好ましくない場合があります)。

BITFIELD 内のビットはアクティブ LOW です。BITFIELD 内の特定のビットをチェックするには:

- マスク フィールドを使用し、関心対象のビットをチェックします。例えば、ゼロ フラグをチェックするにはマスク値を 0x20 に設定します。これにより、不要なビットがマスクされ、パラメーター比較により評価されなくなります。
- 高および低パラメーター比較値を 0 に設定します。マスクは、すでに関心対象のビットに対してスクリーンされているため、パラメータ比較はもはや必要ありません。高および低比較の値 00 を使用し、マスクを通るあらゆるビットを当該比較により評価することができます (BITFIELD のビットがアクティブ LOW であるため)。

  BITFIELD 比較パラメーターを示す具体例については、「例 1: 特定アドレスの書き込みのブレーク」および「例 2: 特定値とアドレスの書き込みのブレーク 」をご参照ください。

4.4.2 16 ビット スレッド オプション

16 ビット スレッドでは、PSoC 内の 16 ビット幅パラメーターを評価する方法をいくつか使用できます。比較は、特定の値または 0x0000～0xFFFF の値の範囲に対して行えます。16 ビット スレッドで使用可能な比較オプションの一部は 8 ビット幅のみであり、0x00～0xFF の値を評価するため、ご注意ください。

Invert: このオプションでは、16 ビット パラメーター比較の結果を反転されることができます。

Parameter: パラメーター ドロップダウンでは、8 ビット スレッドで多数の比較を使用できますが、いくつかの追加の 16 ビット幅パラメーターに対して比較を行なう能力もあります。一部の 16 ビット比較を使用し、2 つの 8 ビット パラメーターを単一の比較で同時にチェック可能です。使用可能な 16 ビット比較オプションは以下の通りです。

NONE: このパラメーターを選択すると、16 ビット スレッド比較が無効になります。

A[1]*:* アキュムレータ比較は、8 ビット スレッド バージョンと同一の機能を備えています。

IO_DA[1]*:* レジスタ アドレス比較は、8 ビット スレッド バージョンと同一の機能を備えています。

IO_DA_DB: 複合レジスタ アドレスとデータ比較。入力された 8 ビット MSB はレジスタ アドレス比較用に使用され (IO_DA と同様)、入力された 8 ビット LSB はレジスタ データ比較用に使用されます (IO_DB と同様)。

IO_DB[1]*:* レジスタ データ比較は、8 ビット スレッド バージョンと同一の機能を備えています。

IR[1]*:* 命令レジスタ比較は、8 ビット スレッド バージョンと同一の機能を備えています。

IR_SP: 複合命令レジスタとスタック ポインター比較。8 ビット MSB は命令レジスタ比較を実行し、8 ビット LSB はスタック ポインター比較を実行します。

MEM_DA[1]*:* RAM アドレス比較は、8 ビット スレッド バージョンと同一の機能を備えています。

MEM_DA_DB: 複合 RAM アドレスと RAM データ比較。8 ビット MSB は RAM アドレス比較を実行し (MEM_DA と同様)、8 ビット LSB は RAM データ比較を実行します (MEM_DB と同様)。

MEM_DB[1]*:* RAM データ比較は、8 ビット スレッド バージョンと同一の機能を備えています。

PC16: プログラム カウンター比較。このパラメーターを使用し、プログラム カウンター レジスタ内の現時点の値を評価できます。

SP[1]*:* スタック ポインター比較は、8 ビット スレッド バージョンと同一の機能を備えています。

X[1]*:* X レジスタ比較は、8 ビット スレッド バージョンと同一の機能を備えています。

X_A: 複合 X レジスタとアキュムレータ比較。8 ビット MSB は X レジスタに対して比較を実行し、8 ビット LSB はアキュムレータに対して比較を実行します。

注 1: 16 ビット幅比較入力フィールドに入力した 8 ビット LSB のみは、A 比較、IO_DA 比較、IO_DB 比較、IR 比較、MEM_DA 比較、MEM_DB 比較、SP 比較、X 比較のために評価されます。ページ ポインター レジスタは、RAM に対する 8 ビット以上の比較またはレジスタ アドレス比較のために自動的にチェックされることはありません。

4.4.3 ステート ロジック

Operator: 演算子フィールドをでは、8 ビット スレッドおよび 16 ビット スレッドを、両方が有効の場合、論理的に複合できます。使用可能な論理演算子は AND、OR、NAND、NOR です。8 ビット スレッドまたは 16 ビット スレッドのいずれかが無効の場合、演算子は無視されます。

Match Count: マッチ カウントでは、トレースまたはブレークポイントを有効にする前、あるいは別のイベント ステートに移動させる前に、当該イベントを何回発生させる必要があるかを特定できます。設定可能なマッチ カウントは 1～32,767 です。32,767 を超えたマッチが必要な場合、複数のイベント ステートを合わせてチェーン化することが可能です。

Next State: 次のステートのパラメーターでは、複数のイベント ステートを合わせてチェーン化し、より複雑で柔軟なイベント シーケンスを作成できます。指定したマッチ数を含む現時点のイベント ステートが満たされると、次のステートが実行されます。ステートのチェーン化を望まない場合、次のステートを現時点のステート番号にセットする必要があります。

すべてのステートが同時に評価されるわけではありません。ステート 0 が常に最初となります。ステート 0 が無効の場合、またはその評価が偽である場合、それ以降のステートは評価されません。そのため、イベントまたはイベントのチェーンはすべて、チェーン化を望まない場合でも、ステート 0 で開始しなければなりません。

Trace: トレース バッファは、成功したイベント ステート マッチから起こり得る結果に応じて開始または停止されます。これは、コードの特定セクションのみをトレースしようとする場合に有用です。

Break: ブレーク チェックボックスでは、イベント条件が満たされた時、このイベントによりブレークポイントを発生させるかどうかを指定できます。

4.4.4 イベントについての注意

- 成功したイベント条件に対してブレークを行なった後、PSoC Designer はイベント発生後に 2 つの命令を停止させます。停止の理由は、イベント パラメーターの ICE 評価が実際のポッド実行を 2 命令サイクル遅滞させるためです。この動作は時折、各 C コード行のためにアセンブリの複数行が必要となるため、C デバッグにおいては目立ちません。
- イベント ウィンドウ内の変更は、実行中のデバイスには適用できません。イベントまたは一連のイベントを変更する場合、デバイスが停止している時に「適用」ボタンをクリックし、ICE イベントが更新されたイベントを認識できるようにしなければなりません。
- 8 ビット スレッドの BITFIELD 比較パラメーターを使用する場合、次の 2 つを行なう必要があります: 高と低比較値を 0x00 に設定し、Mask フィールドを使用して適切な BITFIELD パラメーターをチェックします。 BITFIELD パラメーターの設定および使用の例については、「例 1: 特定アドレスの書き込みのブレーク」をご参照ください。
- イベント設定の評価が真であり、ブレークが発生させられた場合、もはや評価が真であるイベントを発生させないステートにチップのステートが変化するまで、その設定の評価は真であり続けます (また、ブレークが誘導され続けます)。例えば、アキュムレータに特定値がロードされてもその条件の後に続く命令がアキュムレータを変更しない場合、ブレークが起こるようイベントを設定したら、アキュムレータが変更されるまでブレークが発生し続けます (1 度に 1 命令)。

4.4.5 イベントの一般的用法

イベントは多くのデバッグ状況で使用可能ですが、一般的に可能な使用法は以下の通りです:

- レジスタまたは RAM アドレスがいつ読み出される／書き込まれるかを特定します (後述のイベント例 1 と 2 をご参照ください)。
- スタック オーバーフローを特定します (「例 3: スタック オーバーフロー」をご参照ください)。
- 関数のジャンプまたはコールアウトを検出します。
- 特定範囲のコードをトレースします。
- 割り込みレイテンシを測定します。
- コード実行の「n」回目行をブレークします (Match Count、後述の例 4 をご参照ください)。
- キャリー フラグ ステータスをブレークします。
- 特定数の命令を待機します。
- アキュムレータの特定データをブレークします。

いくつかの一般的な使用例における、イベント ツールの設定法および使用法の詳細については、以下の通りです。

4.4.6 例 1: 特定アドレスの書き込みのブレーク

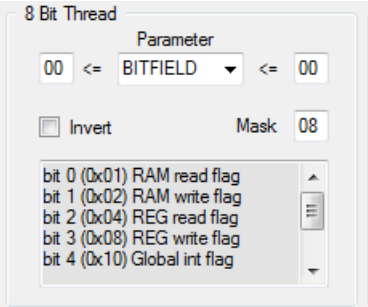
状況によっては、特定アドレスにいつ書き込みが行なわれるかを把握することが重要である場合があります。

例えば、I2C スレーブに関する断続的な問題を抱えていて、マスターが定期的にそのスレーブから不正確な値を読み出しているとします。ユーザーは、この不正確な値が I2C データ レジスタにロードされていることに気付かないかもしれません。イベントを使用し、I2C 出力データ レジスタが書き込まれる時にブレークすることが可能です。

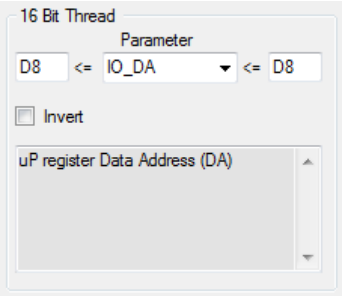
これを実現するには、8 ビット スレッドを使用してレジスタ アドレスへの書き込みがいつ行なわれるかをチェックします。このチェックは、I2C データ レジスタ アドレスをチェックする 16 ビット スレッドと複合させることができます。当該アドレスに対して書き込みが行なわれると、ブレークポイントが発生します。

特定アドレスへの書き込みをブレークするには、以下のようにパラメーターを設定してください。

1. 8 ビット スレッドのパラメーターを設定します。



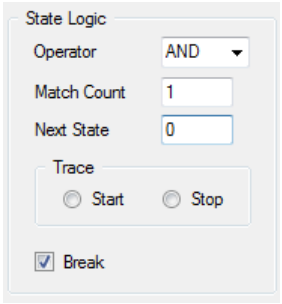
   a. Parameter = BITFIELD。この設定により、この例で使用した「register write」チェックを含む、さまざまなチェックが可能となります。
   b. パラメーターの高および低比較値を「00」に設定します。BITFIELD はアクティブ LOW であるため、すべてのビットをチェックすることができます。Mask 値は、関心対象の特定ビットをチェックするために使用します。
   c. Mask 値を 0x08 に設定すると、レジスタ書き込みが行なわれる時はいつでも BITFIELD 比較の評価が真となります。
2. 16 ビット スレッドのパラメーターを設定します。



a. Parameter = IO_DA。この設定により、特定レジスタ アドレスがバス上にある時にイベントを発生させることができます。

b. 高および低比較値を「D8」に設定します。これは、I2C ブロックのデータ レジスタです。このレジスタ、またはその他ブロックのレジスタは、ユーザー モジュール データシートまたはテクニカル リファレンス マニュアルに記載されています。

3. ステート ロジックのフィールドを設定します。



a. Operator フィールドを「AND」に設定します。これにより、イベントの評価が真となる前に、8 ビット スレッドと 16 ビット スレッドで定義された両方の条件が満たされるようにイベント ステートが設定されます。この場合、書き込みが I2C データ レジスタに対して行われる場合のみ、イベントの評価が真となります (読み出しは無視されます)。

b. Match Count を「1」に設定します。このイベントは、ブレークポイントがトリガーされる前に 1 度のみ生じる必要があります。

c. Next State を「0」に設定します。この場合、所望の条件を評価するためにイベント ステートが 1 つのみ必要となるため、ステートのチェーン化は必要ありません。

d. トレースの「Start」および「Stop」オプションは未設定のままにしてください。この例ではトレースを使用しません。

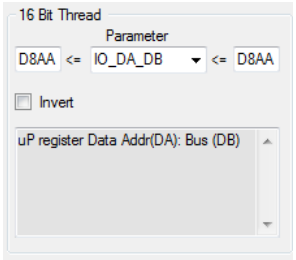
e. 「Break」チェックボックスを選択します。これにより、イベント ステート全体のために定義された条件の評価が真である時はいつでも、ブレークポイントが発生するようになります。

4.4.7 例 2: 特定値とアドレスの書き込みのブレーク

イベント例 1 を拡張するには、設計者は、特定値がレジスタにいつ書き込まれるかについても知っておく必要があります。例 1 で示されているイベントは、I2C データ レジスタに値が書き込まれるたびにブレークしますが、レジスタに特定の不正確な値が書き込まれる時のみにブレークするほうが有益である場合があります。

この機能を追加するには、16 ビット スレッドを多少変更する必要があります。イベント ステートのその他の部分は、例 1 で指定したものと同じようにしておく必要があります。

1. 16 ビット スレッドのパラメーターを設定します。



a. Parameter = IO_DA_DB。この設定により、バス上の特定レジスタ アドレスに特定値が書き込まれる時にイベントを発生させることができます。

高および低比較値を「D8AA」に設定します。これにより、0xAA が I2C データ レジスタ (0xD8) に書き込まれた時はいつでも、ブレークが発生するようになります。値の範囲は、0xAA だけではなく、高および低比較値を変化させることによりチェック可能となることにご注意ください。

4.4.8 例 3: スタック オーバーフロー

イベント ビューアーのもう 1 つの有用なシナリオとして、スタック オーバーフローをチェックすることが挙げられます。スタック オーバーフローは、予期しない動作を引き起こし、時にはデバッグすることが困難となる場合があります。幸いなことに、イベント ビューアーを使用すれば、スタック オーバーフローの検出およびスタック使用の監視が容易となります。

イベントを伴うスタック オーバーフローについてプログラムを監視するには、8 ビット スレッドを設定してスタック ポインターを監視しなければなりません。スタック オーバーフローの発生間近にブレークを行なうには、以下のようにパラメーターを設定してください。

1. 8 ビット スレッドのパラメーターを設定します。
   a. Parameters = SP。この設定により、スタックポインター値の監視が可能となります。
   b. 低比較値を FD に、高比較を FF に設定します。これにより、スタックポインターが FD に達した時はいつでも、ブレークが発生させられるようになります。
   c. Mask 値を FF に設定します。この設定でマスクされるビットはありません。
2. 16 ビット パラメーター値を「NONE」に設定し、16 ビット スレッドを無効にします。
3. ステート ロジックのフィールドを設定します。
   a. 16 ビット スレッドが無効にされたため、Operator の選択は重要ではありません。16 ビット スレッドが無効の場合、Operator の選択は無視されます。
   b. Match Count を 1 に設定します。このイベントは、ブレークポイントがトリガーされる前に 1 度だけ行なう必要があります。
   c. Next State を 0 に設定します。この場合、所望の条件を評価するために 1 つのイベント ステートのみが必要なので、ステートのチェーン化の必要はなくなります。
   d. トレースの「Start」および「Stop」オプションは未設定のままにしてください。この例ではトレースを使用しませんが、スタック オーバーフローの原因を判定する際にはトレースが有用である場合があります。
   e. 「Break」チェックボックスを選択します。これにより、イベント ステート全体のために定義された条件の評価が真である時はいつでも、ブレークポイントが発生するようになります。

デバッガなしでスタック オーバーフローをチェックする方法およびスタック オーバーフローを回避する技術を含む、スタック オーバーフローの追加情報については、「付録 B – スタック オーバーフロー」をご参照ください。

4.4.9 例 4: イベントの X 発生後のブレーク

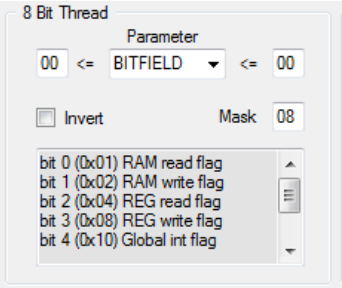
この例では、イベント条件の評価が複数回にわたって真となった後の、ブレークポイントの設定方法について説明します。この措置は、プログラムが実行を停止する場合に、開発者がより微細に制御できるよう支援します。

この例では、以下のコードにおける for ループを通じて、254 回目にブレークするようイベントビューアーを設定します。

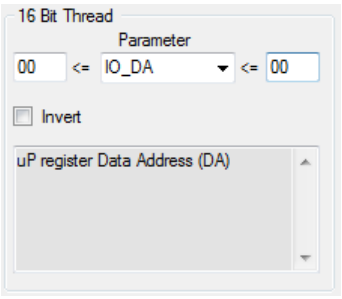
```
BYTE x;
for(x = 0; x < 255; x++)
{
   PRT0DR ^= 0x01;
}
```

このイベント ステートの設定は、例 1 で示したステートと類似しています。BITFIELD パラメーターはレジスタ書き込みを検出するために使用されます。これと、PRT0DR レジスタに対する 16 ビット スレッド チェックと組合わせることで、イベント カウントがループを通して毎度増加するようにします。このイベント ステートを設定するには、当該イベントに以下の変更を行なってください。

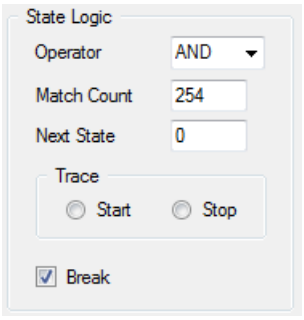
1. 8 ビット スレッドのパラメーターを設定します。これらは例 1 で使用したものと同一の設定です。各設定の詳細については、例 1 をご参照ください。



2. 16 ビット スレッドのパラメーターを設定します。これらの設定は、例 1 で使用したものと類似しています。ただし、アドレス比較は 00 (PRT0DR アドレス) に変更されています。



3. ステート ロジックのフィールドを設定します。例 1 との唯一の違いは、Match Count が「254」に変更されていることです。これにより、イベントの評価が真となり、ループを 254 回通り抜けた後にブレークポイントがトリガーされます。



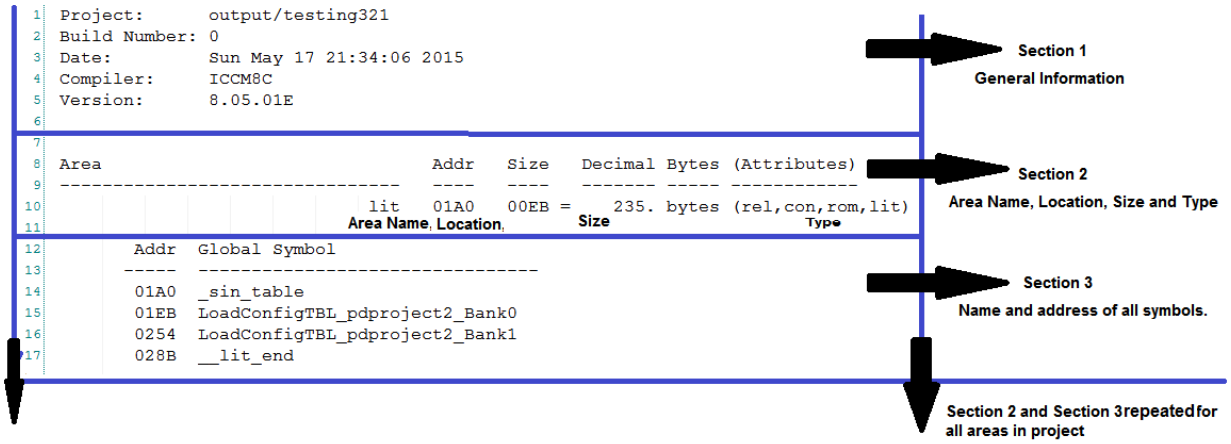
4.5 マップ ファイル (.mp)

マップ ファイルはビルド プロセス中に作成され、グローバル シンボル アドレスおよびメモリ内で定義された領域についての情報が記載されています。マップ ファイルは、メモリのどこに変数やコードの一部があるのか判定する場合、またはコード部分がどれだけのメモリ量を占めているかを判定する場合に、有用なデバッグ ツールです。

グローバルでない変数や関数、ラベルはマップ ファイルには表示されません。

マップ ファイルの各セクションは図 26 に示されています。レイアウトと説明は、ImageCraft C コンパイラおよびリンカーV7.0.5 に特有であり、コンパイラのバージョンによって多少異なる場合があります。マップ ファイルは、PSoC Designer の Workspace Explorer 内の「Output Files」フォルダで使用可能です。

図 26. コメント付きマップ ファイル



セクション 1: 一般情報

マップ ファイルの冒頭 (行 1～5) には、プロジェクト名、日付、最新ビルドのタイムスタンプ、コンパイラ バージョンを含む、プロジェクトについての一般情報が一覧表示されます。

セクション 2: 領域名、ロケーション、サイズ、タイプ

次のセクション (行 8～10) は、メモリの次の領域のヘッダです。このセクションには、領域名、ロケーション (メモリ アドレス)、サイズ、領域のタイプが表示されています。

領域名: これは領域が宣言される時に割り当てられます。 領域とは、コンパイラがコードやデータを配置するメモリのセクションです。ユーザーは独自のメモリ領域を指定できますが、ImageCraft コンパイラではいくつかの事前定義された領域を使用します。

top: 割り込みベクターおよび boot.asm コードを含みます。この領域は、ROM 内のアドレス 0 に配置されます。

lit: 定数データを含んでおり、ROM 内に配置されます。

idata: グローバル データの初期化値を格納します。

text: ユーザー コードを含んでいます。

psoc_config: デバイスを設定するのに使用する関数 (およびデバイスを動的に再設定するための関数) を含みます。

usermodules: ユーザー モジュール API ルーチンを含みます。

領域ロケーション: 領域の開始アドレスです。これは、ROM 内で定義された領域の ROM アドレス、RAM 内で定義された領域の RAM アドレスです。

領域サイズ: これは領域の総サイズであり、16 進数および 10 進数で示されます。領域の終点は、領域ロケーションに領域サイズを加えたものです。

領域タイプ: 特定の領域の定義された属性です。これらは、領域内に配置されるデータのタイプおよび、リンカーが領域を配置する方法を定義します。以下は、領域のタイプを定義するのに使用できる有効なキーワードの全リストです。

REL – 再配置可能領域。このキーワードにより、リンカーは選択したアドレスに領域を再配置します。再配置可能領域には、事前定義されたロケーションがなく、ビルドのリンク段階の途中でアドレスが判定されます。

ABS – 絶対領域、すなわち再配置不可能領域。このキーワードは、領域アドレスがコードで定義されており、リンカーによる移動ができないことを意味します。

領域は再配置可能または絶対のどちらかであり、両方であることはあり得ません。

CON – 連結領域。このキーワードは、メモリ内で領域が互いに連続して続くよう指定します。

OVR – オーバーレイ領域。このキーワードにより、複数のオーバーレイ領域が同一の開始アドレスを持ち、同一のメモリ空間を占めるよう指定します。これは RAM 領域にのみ適用できます。

領域は、連結またはオーバーレイのどちらかであり、両方であることはあり得ません。

RAM – データを RAM 内に格納し、変数ストレージにのみ使用するよう指定します。

ROM – データをフラッシュ内に格納するよう指定します。コードと定数データの両方を ROM 領域に格納できます。

領域は RAM または ROM のどちらかであり、両方であることはあり得ません。

この例では、表示されている領域は「lit」、すなわち、リテラル データ領域です。この領域は、プロジェクトで使用されたあらゆる定数データを格納します。総サイズは 235 バイトであり、領域タイプは「rel、con、rom」です。これは、コード (ROM) メモリ空間内の再配置可能な連結領域を意味します。

セクション 3: 全シンボルの名およびアドレス。

マップ ファイルのこのセクションには、セクション 2 で説明および定義された領域の内容が記載されます。当該領域に配置された各シンボルの名およびアドレスが順に一覧表示されます。各シンボルが連続して配置されているため、各要素のサイズは、特定のシンボルのアドレスと次のシンボルのアドレスを比較することで判定できます。

一覧表示されたアドレスは、特定の領域のメモリ タイプに対応します (RAM 領域は RAM アドレス、ROM 領域は ROM アドレスとなります)。

領域内のすべてのシンボルが、必ずしもメモリを占めるわけではありません。グローバル シンボル ラベルもまた、リスト内に表示されます。例えば、図 26 の「__lit_end」シンボルは、lit 領域の終わる場所を示しているだけで (アドレス 0x028B)、メモリを物理的に占めていません。

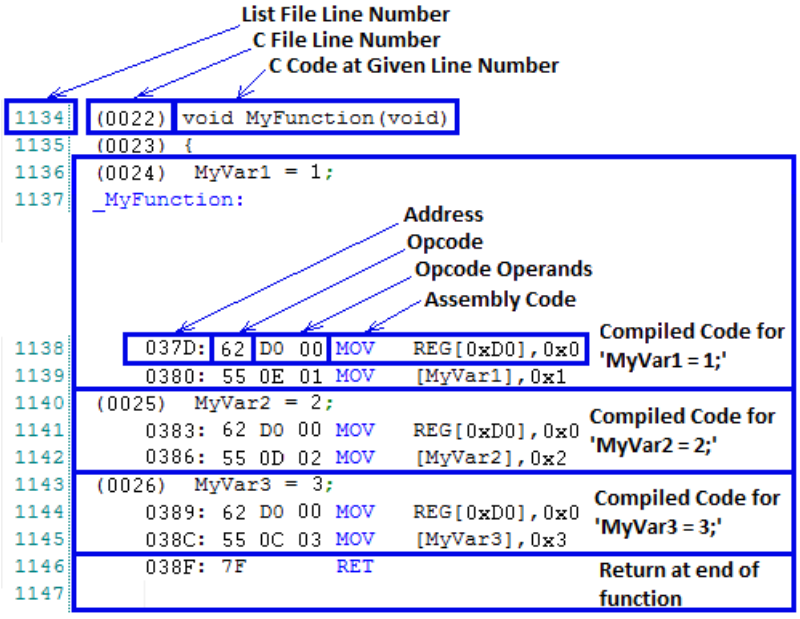
4.6 リスト ファイル (.lst)

PSoC Designer が生成するもう 1 つの有用なファイルは、リスト (.lst) ファイルです。これには、C コンパイラにより生成されたアセンブリ コードが記載されています。期待通りに動作しない機能をデバッグする場合、または C コードの部分とアセンブリ コードの部分をインターフェース接続する場合には、リスト ファイルをご参照ください。また、デバッガの「Step ASM」機能でも、リスト ファイルを見ることができます。

図 27 は、リスト ファイルからの抜粋を示したものであり、重要情報がハイライトされています。C ソース ファイルの各行について、リスト ファイルは C コード行を一覧表示しており、該当行に必要なすべてのコンパイル アセンブリを連続的に一覧表示しています。リスト ファイルはフラッシュのアドレス 0x00 で始まるよう編成されており、メモリの終わりまですべてのコンパイル コードを連続的に一覧表示します。そのため、大規模プロジェクトではファイルが長くなる場合があります。検索機能は、大規模なリスト ファイル内で特定の関数またはコードの部分を見つけるための簡単な方法です。

またアセンブリの各行は、命令用のコード アドレスおよびアセンブリ命令用の特定オペコードとオペコード オペランドも一覧表示します。アセンブリ オペコードの詳細情報については、アセンブリ言語ユーザー ガイド (**PSoC Designer > Documentation > Compiler and Programming Documents > Assembly Language User Guide**) をご参照ください。

図 27. コメント付きリスト ファイル



このリスト ファイルの抜粋は、定数値を 3 つのグローバル変数 (MyVar1、MyVar2、MyVar3) に割り当てる「MyFunction」を示したものです。図に示すように、各 C 割り当てはアセンブリ コードの 2 行 (ページ レジスタを設定する行および値を変数に割り当てる行) にコンパイルされます。

5 PSoC 1 デバッグのヒントおよびコツ

本節には、PSoC 1 デバイスをデバッグする場合に覚えておくとよい固有の側面が記載されています。

5.1 停止中にハードウェアが動作

ICE は PSoC デバイス内のコード実行の状態をエミュレートおよび制御できますが、ICE はデバイスのデジタルおよびアナログ部分については明白な制御能力を持ちません。そのため、デバッガが停止する時はいつでも、その他のハードウェア要素が動作し続けます。ブレークポイントでデバッガが報告するハードウェア値は、停止した時間におけるこれらのレジスタの値のスナップショットです。

この動作が最も顕著になるのは、カウンターおよびタイマーと併用する場合に、または一部の ADC ユーザー モジュールなどカウンターおよびタイマーを使用するユーザー モジュールと併用する場合です。デバッガが CPU の実行を停止した後、カウンターは動作し続けます。このため、プログラムが再起動されると、タイマー／カウンター／PWM (およびカウンターを使用するあらゆる ADC) により報告される最初の値は、不正確である場合があります。初期に不正確な値を報告した後、タイマー／カウンター／PWM (およびタイマーを使用するあらゆる ADC) は再同期化し、再び正しい値を報告し始めます。

デバッグが ADC 読み出し値の正確性に依存している場合、ADC 用のデバッガが報告する値を確実に正しいものとするために使用できる迂回策が 2 つあります。

1. フラッシュ書き込みのデバッグ: プログラムを実行しながら、ADC 結果を読み出す命令の後に、コード ウィンドウの左の余白 (秒番号の左側) を左クリックしてブレークポイントを動的に設定します。プログラムが停止すると、ADC からの読み出し値が正確となります。別の読み出し値を取得するには、ブレークポイントをクリアし、プログラムを実行してもう一度ブレークポイントを動的に配置します。
2. 選択した ADC が単一のサンプル モード (インクリメンタル ADC など) をサポートする場合、別のオプションを利用可能です。連続サンプル モードで ADC を実行する代わりに、単一サンプル モードで実行します。単一サンプル モードでは、各変換後にカウンターおよびタイマーが停止します。これにより、ADC から返される値の正確性が保たれます (変換開始時と結果返答時との間にブレークポイントがない限り)。この方法はデルタ シグマ ベースの ADC には適用されないことにご注意ください。

5.2 フラッシュ書き込みのデバッグ

内部フラッシュ書き込みを含むプロジェクトを開発する場合、ICE およびデバッガを使用することができます。例として、ブートローダや、フラッシュにデータを保存するその他のアプリケーションが挙げられます。フラッシュ書き込みに ICE デバッガを使用する場合、以下を考慮してください。

- ICE がフラッシュ書き込みをエミュレートできるようにするには、書き込むフラッシュ ブロックの保護を解除しなければなりません。保護されていないフラッシュは、外部デバイスのフラッシュへの書き込みを可能とする唯一のセキュリティ設定です。ICE が内部フラッシュ書き込みをエミュレートしている場合でも、ICE は技術的に PSoC デバイス内のフラッシュの更新を要求する外部デバイスです。
- ICE を使用してエミュレートされたフラッシュ書き込みは、スタンドアロン PSoC デバイスでのフラッシュ書き込みと比べて、時間が非常に長いです。フラッシュ書き込み中の ICE と PSoC デバイス間で必要とされる通信量に時間がかかります。ICE による書き込み時間はそれぞれ異なりますが、通常 PSoC デバイス データシートで定義される標準的な書き込み時間よりも 10～20 倍長くなります。
- m8ssc.inc の `SSC_Action (OpCode)` マクロ (すべての PSoC Designer プロジェクトに含まれる) を使用せずにシステム監視コール (SSC) 命令を実行しないでください。フラッシュ書き込みにつながる命令、またはその他の SSC 命令は、ICE が SSC 命令を検出してエミュレートできるように必ず特定シーケンスに従う必要があります。`SSC_Action` マクロは命令の適切なシーケンスを実行し、SSC 動作中のエミュレーションを可能にします。ユーザー モジュールおよびサイプレス提供のライブラリ (Flashblock 書き込みライブラリなど) は正しい SSC マクロを使用しています。
- E2PROM ユーザー モジュールと Flashblock API ライブラリ フラッシュ書き込みルーチンは、事前コンパイルされたライブラリとして利用できます。このソースは、デバッグ中にデバッガが「ステップイン」するために提供されておらず、利用できません。しかし、この関数は「ステップオーバー」される場合でも正確に実行されます。

5.3 スリープでのデバッガの使用

スリープを使用するプロジェクトをデバッグする場合、以下を考慮してください。

- ブレークポイントは、M8C_Sleep 命令の後に続く行に配置できません。スリープは、スリープする前に移行する前に次の命令のプリフェッチを実行します。結果として、ICE は PSoC との同期がずれるようになり、スリープ命令の直後にブレークポイントが発生する場合は接続が切断されます。
- デバッガは、スリープ命令に単一のステップインまたはステップオーバーを許可しません。そうすれば、ICE はデバイスから切断されます。これを回避するには、スリープ命令から 2 行後にブレークポイントを設定し、スリープ命令に明示的にステップオーバーするのではなく、デバッガが自由にブレークポイントまで動作できるようにします。
- 接続された PSoC 1 デバイスがスリープしている間にデバッガを手動で停止させる場合、ICE は切断されます。これを回避するには、スリープを使用するプロジェクトで、停止コマンドを直接クリックするのではなく、常にブレークポイントまたはイベントを使用してプログラムを停止させる必要があります。

6 代替のデバッグ オプション

本節では、代替のデバッグ ツールおよび技術について説明します。これらの代替オプションには標準デバッグ設定の機能セットが欠けていますが、標準 ICE デバッグ設定が利用できないか実用的でない場合にはコスト効率に優れたオプションです。またこれらの代替オプションは、デバイスの動作を物理的に停止せずにデバッグできるため、リアルタイムのデバッグ サポートが必要な状況に適しています。

6.1 I2C-USB ブリッジによるデバッグ

I2C-USB ブリッジを使用し、USB ポートを PC から PSoC 内の I2C ユーザー モジュールへブリッジすることにより、PsoC アプリケーションのハードウェアおよびソフトウェアをテスト、調整、およびデバッグすることができます。ブリッジは I2C ホストとして機能し、USB インターフェースからバス上の I2C スレーブにコマンドを発行します。

サイプレスは CY8CKIT-002 PSoC MiniProg3 プログラムおよびデバッグ キットで I2C-USB ブリッジを提供しています。MiniProg3 には組込み I2C-USB ブリッジ機能が搭載されており、すべての PSoC 1、3、5 デバイスをプログラム可能です。MiniProg3 は図 28 に示されています。

図 28. MiniProg3 I2C-USB ブリッジ

I2C-USB ブリッジは、リアルタイム アプリケーションのデバッグ用として、または ICE や OCD 設定を利用できない状況で有用です。この方法でのデバッグには ICE も OCD 対応の PSoC も必要ありません。

一般的な使用例には以下の要素が含まれます。

- I2C-USB ブリッジ。
- P1[5]と P1[7]または P1[0]と P1[1]のいずれかに接続されている、実行中の EZI2C スレーブ ユーザー モジュールを備えた対象 PSoC デバイス。
- ブリッジへコマンドを発行するため実行中の、ブリッジ コントロール パネル ソフトウェア搭載の PC。

その後、I2C-USB ブリッジはホスト PC と I2C スレーブとして機能する対象 PSoC デバイスとの間で接続されます。ブリッジ コントロール パネルは、情報を取得するために定期的に PSoC デバイスをクエリするために使用されます。ポーリング レートおよび転送データ量は完全に設定可能であり、実行中の PSoC デバイスからのリアルタイム デバッグ情報を閲覧するために役立ちます。

一般的な I2C-USB ブリッジ デバッグの使用事例:

- リアルタイムの CapSense ボタン チューニング データのストリーミング: ブリッジ コントロール パネルは、受信 I2C データのリアルタイム グラフ作成もサポートします。この使用事例では、PSoC プログラムはボタン情報を I2C アレイに通常通りロードし、I2C ブリッジはそのデータを定期的にクエリします。
- 事前定義された「ウォッチ」変数のセットの監視: この使用事例では、PSoC プログラムは関心対象の変数を I2C アレイに通常通りコピーし、I2C ブリッジはこのデータを定期的にクエリし、監視します。
- I2C スレーブ機能のテスト: I2C-USB ブリッジは I2C マスターとして機能し、バス上の I2C スレーブの機能をテストできます。
- I2C ベースのブートローダの機能のテスト: I2C-USB ブリッジの使用は、I2C ベースのブートローダと通信するための一般的な方法です。PSoC Designer は、ブリッジ コントロール パネルが I2C ブートロードの動作のために直接利用できるダウンロード ファイルの生成をサポートしています。

ブリッジ キットの資料およびそれらの使用法の関連資料については、CY8CKIT-002 PSoC MiniProg3 プログラムおよびデバッグをご参照ください。

6.2 UART インターフェースによるデバッグ

UART インターフェースによるデバッグは、組込みプロセッサでよく用いられる方策です。幸いなことに、UART インターフェースを PSOC 1 デバイスに追加することは容易であり、多くの PSoC 1 開発キットは RS-232 トランシーバを備えています。

また、ほとんどあらゆる PC 上で、UART デバイスにより送信された情報を閲覧できます。シリアル ポートは一般的ではなくなっていますが、USB-UART ブリッジ デバイスは広く使用されています。Hyperterminal などデータを閲覧するための PC ソフトウェアも、広く利用されています。

UART デバッグの一般的な使用事例は、前述した I2C-USB ブリッジの事例と類似しています。UART ユーザー モジュールに設けられている API により、デバッグ情報を PSoC デバイスとの間で渡すことが容易になります。API のオプションには、PutString、PutChar、GetString、GetChar などがあります。

PSoC 1 上での UART の設定および使用の詳細については、PSoC Designer 内に用意されている UART ユーザー モジュール データシートをご参照ください。

6.3 ピン トグル

ピン トグルは基本ですが、特にリアルタイムのデバッグが必要な状況において便利なデバッグ ツールです。ピン トグルは以下の状況において頻繁に使用されます。

- 関心対象のセクションを 2 つのピン トグルで収納し、コードのセクションの長さを計る: この手法は、割り込みサービス ルーチン (ISR) の実行時間を計るためによく使用されます。
- コード セクションに到達したかどうかをチェックする: このタスクはブレークポイントを利用して行なうこともできますが、コード実行にへの干渉なしにこれをチェックする際に役立つ場合があります。

7 トラブルシューティング

本節では、PSoC 1 デバッガの使用中に発生する一般的な問題をリストアップします。また、考えられる解決法についても説明します。

問題	可能な解決方法
ICE がポッドを検出したり接続したりすることができない、あるいはポッドと互換性がない	ポッドがプロジェクト用に選択したデバイスと一致していない。各ポッドはファミリ内のさまざまなデバイスをサポート。PSoC デバイス データシートを参照し、使用するポッドまたは OCD デバイスがプロジェクト内で選択した特定の製品番号と互換性があることをご確認ください
	PSoC デバイスが適切に電源供給されていることを確認してください。PSoC は外部から、または ICE 経由で電源供給が可能。Project > Settings > Debugger 設定 (図 19 に示す) は、選択した電源供給方式に一致していなければならない
	ICE が電源供給されていない。ICE は、12V／1A DC 電源を使用して電源供給する必要がある。これより低い電圧源を使用すると、ICE は点灯するが正常に動作しない
	ポッドが適切に接続されていない可能性がある。一般的なデバッグ ハードウェア設定については、「デバッグ ハードウェアの設定」節をご参照ください
	PSoC Designer／PSoC Programmer の単一インスタンスのみが開かれ、対象デバイスに接続されていることを確認してください
	旧バージョンのポッドが ICE に接続されている可能性がある。現行レビジョンのポッドを使用していることを確認してください
デバッグ セッション中に ICE が切断される	ICE が適切に電源供給されていない可能性がある。ICE が 12V／1A DC 電源から電源供給されていることを確認してください。異なる電源によりデバイスの接続は可能ですが、デバッグ開始後に不具合が生じる場合がある
	コード内にフラッシュ書き込みやその他の SSC 命令が発生していないかをチェックしてください。フラッシュ書き込みが行なわれた場合、フラッシュ書き込みによる ICE デバッガの使用方法に関する情報については、「フラッシュ書き込みのデバッグ」節をご参照ください
	コード内でスリープ動作が発生していないかをチェックしてください。スリープ動作が発生していた場合、スリープを使用するプロジェクトに対する ICE の使用方法に関する情報については、「フラッシュ書き込みのデバッグ」節をご参照ください
	デバッグ中は位相同期回路 (PLL) の使用を避けてください。この問題を回避するには、PLL をオフにし、すべての外部水晶ハードウェアを PSoC から切断してデバッグしてください
	デバッグ中は外部水晶発振器 (ECO) の使用を避けてください。この問題を回避するには、ECO をオフにし、内部主発振器 (IMO) をオンにしてデバッグしてください。
	PC、ICE、PSoC が正しくしっかりと接続されていることを確認してください。一般的なデバッグ ハードウェア設定については、「デバッグ ハードウェアの設定」節をご参照ください
デバッグ セッション中に不当メモリ参照が発生	不当メモリ参照 (IMR) は、デバッグ セッション中に ICE の切断を引き起こすのと同じ原因で発生。の問題を解決する方法の提案については、「フラッシュ書き込みのデバッグ」節をご参照ください
選択した ICE ポートが見つからない	USB ケーブルが、ICE または PC のいずれかから外れている可能性がある。USB ケーブルが適切に接続されていることを確認し、数秒待ってから接続を再試行してください
	ICE が適切に電源供給されていない可能性がある。12V／1A DC 電源から電源供給されていることを確認してください
	PSoC Designer／PSoC Programmer の単一インスタンスのみが開かれ、対象デバイスに接続されていることを確認してください
プログラム実行が予期しないロケーションで停止	boot.asm のアドレス 0x04 (LVD 割り込みベクター) において、または LVD 割り込みサービス ルーチン (定義されている場合) においてプログラムを停止させる措置である、低電圧検出 (LVD) 割り込みがトリガーされている可能性がある。 この場合、 - PSoC 電源をチェックしてください。PSoC への電圧が、グローバル パラメーター設定で指定した LVD トリップ電圧を下回って変動する場合、LVD 割り込みが発生 - ICE が適切に電源供給されていること確認してください。ICE は 12V／1A DC 電源から電源供給されなければならない。これを下回ると予期しない動作が起こる場合がありる - ICE が対象デバイスまたは基板に電源供給している場合、対象が必要とする電流が 250mA 以下であることを確認してください

問題	可能な解決方法
	スタック オーバーフローが発生していないことを確認してください。関数の呼び出し中にスタックにロードされた PC 値が上書きされると、スタック オーバーフローによりプログラム カウンター (PC) が破損する可能性がある。これに起因してプログラム実行がコード内の予期しないロケーションに到達する場合がある。スタック オーバーフローの検出および防止のヒントについては、本アプリケーション ノートの「付録 B – スタック オーバーフロー」をご参照ください
USB ハブ電力を超過している	基板が ICE 経由で電源供給を受けている場合、その基板は、ICE (および USB ハブ) が供給するよりも多くの電流を必要としている可能性がある。(ICE で電源供給する代わりに) 直接外部電源を使用するか、PC USB ポートに近い USB ポートを使用してください
イベントがまったく検出されない	イベントが適切に適用されていることを確認してください。イベント ステートを ICE で認識するためにデバッガが停止している場合は、イベント ウィンドウ内の「Apply」ボタンをクリックする必要がある。デバッガの実行中または切断中にイベント ステートを適用すると、イベント情報は ICE に適用されない
	関心対象のイベントの「Break」チェックボックスが選択されていることを確認してください
どの解決法でもうまくいかない場合	可能であれば、別のハードウェアを利用してください: - まず、ICE を別の (できれば新しい) ICE と取り替えて、問題が解決されるかどうか確認してください。これで解決された場合、当該 ICE ユニットに問題があったことになる - PSoC ユニット搭載のポッドまたは基板を、新しいまたは別のハードウェアに交換してください。これで問題が解決された場合、ポッド、プリント基板、または ICE コネクタに問題があったことになる
	PSoC プログラマと PSoC Designer をアンインストールし、各ツールの最新バージョンを再インストールしてください。ドライバーまたはソフトウェア内蔵のその他のデバッグ関連ツールが適切にインストールされていない場合、接続の問題が発生する可能性がある。再インストールすることで、必要なツールが最新のバグ修正と一緒に正しくインストールされるようになる

8 まとめ

サイプレスは多数のハードウェアおよびソフトウェア ツールを提供しており、PSoC 1 プロジェクトのデバッグをご支援しています。本アプリケーション ノートにより、プロジェクトや問題をデバッグするためにアクセスすべきのハードウェア、ソフトウェア、デバッグ技術はどれであるか、自信を持って把握できるようになるはずです。

著者について

氏名: Dan Sweet

役職: 上級アプリケーション エンジニア

A 付録 A: OCD デバイスのプリント基板への追加

オンチップ デバッガ (OCD) 対応 PSoC 1 デバイスのプリント基板レイアウトへの直接追加は、フット キットやポッドが対象のプリント基板にうまくはまっていない場合、または特定の製品番号に使用可能なポッドがない場合に実行可能なデバッグ ソリューションです。

OCD デバイスをプリント基板に直接追加すると、標準のデバッグ ポッドで以下のメリットが得られます。

- フット キットおよびフル ポッド アセンブリが不要であるため、デバイスのコストが低下します。
- デバッグが必要な各基板用に個別のポッドを購入するよりも、デバッグ機能を持つ基板を複数作成するほうが容易です。
- フット キットが不要であるため、PSoC チップ用の PCB 空間が少なくて済む。コネクタ用の基板空間が必要ですが、OCD デバイスから 4 インチ程度です。

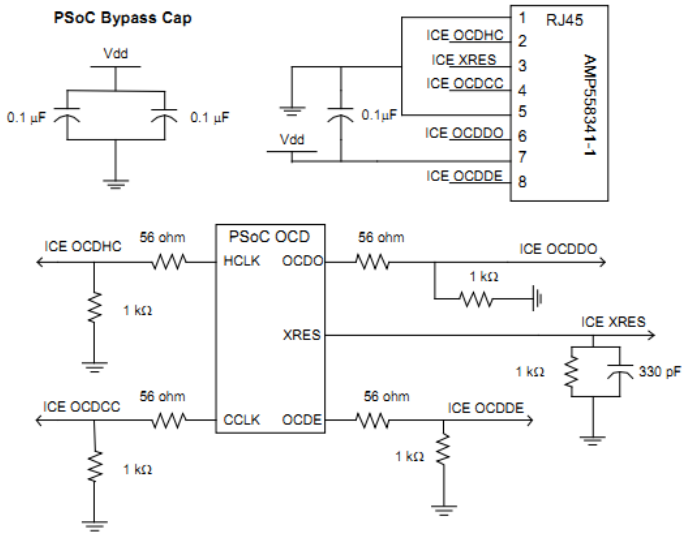
OCD デバイスをプリント基板に追加するために必要なコンポーネントは、表 7 に列挙されています。

表 7. 基板搭載 OCD デバッガに必要なコンポーネント

説明	数量	注
RJ-45 コネクタ	1	AMP/Tyco Electronics 製品番号 5557785-1 DigiKey 製品番号 A31457-ND
PSoC OCD デバイス	1	「オンチップ デバッガ (OCD) デバイス」をご参照ください
56Ω 抵抗	4	1/16W、5%
1kΩ 抵抗	4	1/16W、1%
330pF コンデンサ	1	5V セラミック NPO
0.1µF コンデンサ	3	5V セラミック Y5V

基板搭載 OCD デバッガの回路図は、図 29 に示しています。

図 29. 基板搭載 OCD デバッガ回路図



すべての直列抵抗は、信号ライン上のインピーダンス整合のために終端として使用されています。

バイパス コンデンサが含まれているのは、回路内の重要な要素から AC ノイズをフィルターで除去するためです。

OCDHC、OCDCC、OCDDO、および OCDDE の配線長は、4 インチ未満に制限してください。

B 付録 B: スタック オーバーフロー

B.1 ICE なしでのスタック オーバーフローのチェック

本アプリケーション ノートで、ICE およびイベントを使用してスタック オーバーフローを検出する方法についてすでに説明しました (例 3: スタック オーバーフロー)。この方法はスタック オーバーフローを捕捉し、ブレークするのに最適ですが、デバッグ能力のないデバイスでは機能しません。

幸いなことに、スタック オーバーフローはファームウェア経由でも検出可能です。このような方法の 1 つは、既知の値でスタック ページの終わりを事前充填することが挙げられます。スタックがスタック ページの終わりに達して事前充填値を新しい値で上書きする場合、スタック オーバーフローの問題が発生する可能性があります。

スタック ページで事前充填値を設定するには、以下の C コードを使用してください。

```
char* StackPtr = (char*) 0x7FF;

*StackPtr = 0xAA;
```

特定のコードにより、スタック ページ (この場合では 7 ページ) の最終 RAM ロケーションへのポインターが作成され、その後ロケーションに事前充填値 0xAA が与えられます。RAM の単一ページを持つ PSoC 1 デバイスでは、事前充填アドレス 0x0FF を使用してください。3 桁目は常に、プロジェクトに使用するスタック ページに一致させてください。

その後、以下のコードを使用し、コード実行中の任意ポイントでこのロケーションにスタックが充填されているかどうかをチェックします。

```
if (*StackPtr != 0xAA)
{
       //Stack overflow
}
```

スタックが大きくなり過ぎる前に、コード初期化シーケンスの早い段階で事前充填値を当該スタックに適用してください。事前充填値は、その後どのポイントでもチェックが可能です。メイン ループなど定期的に実行されるコードのセクションにこれを配置すると、最もよく動作します。プログラムの実行中の任意ポイントでスタックがページの終わりに達した場合、チェックがそれを検出します。

充填値 (この例では「0xAA」) が、事前充填ロケーションでスタックにプッシュされる値とたまたま同じである場合、オーバーフローは検出されないので、ご注意ください。この懸念がある場合、異なる事前充填値で複数回テストを実行可能です。または、2 つ以上のスタック ロケーションを事前充填してチェックすることができます。また StackPtr ロケーションは、任意のスタック ページやスタック ページ内の任意のロケーションに変更可能であることにもご注意ください。これにより、特定のプロジェクト向けにスタックがどれだけ充填されるかを判定できます。例えば、StackPtr ロケーションは 0x780 または 0x7C0 に配置し、7 ページのスタックが 50%または 75%充填されているかどうかを確認できます。

B.2 スタック オーバーフローの回避方法

以下の節では、コードを書くための別の方法、およびスタック オーバーフローを回避／修正するための変更について詳細に説明します。すべての方法があらゆるプロジェクトに該当するわけではありません。このため、それぞれのニーズに合った方法をリストから選んで使用してください。

- コンパイラ制限のため、CASE 文の使用は避けてください。
- 可能であれば、複雑な IF 文は避けてください。
- 渡す変数の数とサイズを制限してください。
- ローカル変数の数を制限してください。

B.2.1 CASE 文の回避

CASE 文は最も効率的なコーディングではありません。CASE 文は、それぞれ可能な SWITCH 状態の中間結果を格納するためのスタック空間の使用量を増加させます。サンプル コードはコード 1 に示されており、コード 2 に示されているコードで置き換えられます。

コード 1. スタック空間使用量を増加させるサンプル コード

```
switch ( bSwitchVariable )

  {
     case (0x01):    // do something
        break;
     case (0x02):    // do something else
        break;
     default:        // do third thing
        break;
  }
```

コード 2. スタック空間の使用量増加を回避するためのサンプル コード

```
  If ( bSwitchVariable == 0x01 )
  {
     // do something
  }
  else if (bSwitchVariable == 0x02 )
  {
     // do something else
  }
  else
  {
     // do third thing
  }
```

B.2.2 複雑な IF 文の回避

スタック上の追加バイトを生じさせるもう 1 つの原因として、複雑な IF 文が挙げられます。前節で説明した CASE 文による問題と同様に、複数の判定ポイントを持つ IF 文が原因で、各比較のスタックにバイトがプッシュされます。可能である場合は、必ずこの種の文を最もシンプルな形式に置き換えてください。サンプル コードはコード 3 に示されており、コード 4 に示されているコードで置き換えられます。

コード 3. 複雑な IF 文のサンプル コード

```
if (!(( bVariable == 0x00 )
    || ( bVariable == 0x10 )
    || ( bVariable == 0x20 )
    || ( bVariable == 0x30 )
    || ( bVariable == 0x40 )
    || ( bVariable == 0x50 )
    || ( bVariable == 0x60 )
    || ( bVariable == 0x70 )))
{
    // do something

}
```

コード 4. シンプルな IF 文のサンプル コード

```
if (!(( bVariable & 0x8F ) == 0x00))
{
    // do something
}
```

B.2.3 渡す変数の数とサイズの制限

渡す各変数は、C における関数の呼び出し中に、格納のためにスタックにプッシュされます。そのため、スタック使用量を減らすために、渡す変数の数とサイズを制限します。解決法の 1 つとして、グローバル変数が挙げられます。グローバル変数は一度固定 RAM ロケーションに格納され、あらゆる関数によりアクセス可能であり、スタックには何も追加しません。変数を渡すか、グローバルを宣言するよりむしろ、ポインターをローカル変数に渡すことができます。しかし PSoC 1 デバイスでは、ポインターが 2 バイトであり、スタックへの影響は 2 倍となります。ポインターは、バッファまたはアレイの最初のアドレスを渡すためにのみ使用してください。これにより、アレイの全内容ではなく、2 バイトのアドレスのみがスタックに配置されます。

B.2.4 ローカル変数の数の制限

関数で使用する各ローカル変数には、関数に入れられる時にスタック上のバイトが割り当てられます。そのため、関数により使用されるローカル変数の数を制限するようにしてください。サンプル コードはコード 5 に示されており、コード 6 に示されているコードで置き換えられます。

コード 5. 多数のローカル変数のサンプル コード

```
void SampleFunction ( * abMessageBuffer, bMessageLength )
{
   // local variables
   BYTE bMessageAddress;
   BYTE bMessageType;
   BMessageAddr = abMessageBuffer[0];

   if ( bMessageAddr == 0x01 )
   // do something

   bMessageType = abMessageBuffer[1];
   if ( bMessageType == 0x01 )
   // do something else

   return;

}
```

コード 6. 制限されたローカル変数のサンプル コード

```
void SampleFunction ( * abMessageBuffer, bMessageLength )
{
   if ( abMessageBuffer[0] == 0x01 )
   // do something

   if ( abMessageBuffer[1] == 0x01 )
   // do something else

   return;
}
```

C 付録 C: レガシー ハードウェア

本節では、サイプレスがもはや積極的にサポートを行なっていないレガシー デバッグ ハードウェアの一部を列記します。サイプレスは、レガシー ハードウェアを、本アプリケーション ノートに記載した最新バージョンのハードウェアにアップグレードするよう推奨しています。

C.1 ICE-4000

ICE-Cube の先行品である ICE-4000 は、PSoC Designer 4.4 および PSoC Programmer 2.3 以降ではサポートされていません。

ICE-4000 はパラレル ポート経由で PC に接続し、主にレガシーCY8C25xxx および CY26xxx デバイスをデバッグするために使用されました (新設計には推奨しません)。

図 30. ICE-4000

C.2 CY3240 I2USB ブリッジ キット

CY3240 キットは I2C-USB ブリッジをサポートしますが、プログラミング機能は備えていません。本キットは、サイプレス ソフトウェア ツールによりサポートされていますが、新規ユーザーは代わりに CY8CKIT-002 MiniProg3 キットの購入を推奨します。MiniProg3 は I2C-USB ブリッジをサポートし、CY3240 機能のスーパーセットを備えています。

図 31. CY3240 I2USB

C.3 フレックス ポッド

フレックス ポッドは、また別の広く使用されているレガシー デバッグ用ハードウェアです。フレックス ケーブルの一端に標準 ICE コネクタがあり、OCD デバイスがフレックス ケーブル自体に搭載されており、ケーブルのもう一方の端には (プロトタイピングに使用される) PDIP コネクタがあります。

このハードウェアは、ICE および PSoC Designer ではもはやサポートされていません。フレックス ポッドは、最新バージョンの CY3250 ポッドまたは CY3210 評価用ポッドにアップグレードしてください。

図 32. レガシー フレックスポッド

改訂履歴

文書名: AN73212 - PSoC® 1 によるデバッグ

文書番号: 001-79360

版	ECN	変更者	発行日	変更内容
**	3620161	HZEN	05/16/2012	これは英語版 001-73212 Rev. **を翻訳した日本語版 001-79360 Rev. **です。
*A	3887918	HZEN	01/28/2012	これは英語版 001-73212 Rev. **を翻訳した日本語版 001-79360 Rev. **です。 AN number error corrected.
*B	4773026	HZEN	05/27/2015	変化なし日没 ECN ありません
*C	5045451	HZEN	12/17/2015	これは英語版 001-73212 Rev. *B を翻訳した日本語版 001-79360 Rev. *C 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

車載用	cypress.com/go/automotive
クロック & バッファ	cypress.com/go/clocks
インターフェース	cypress.com/go/interface
照明 & 電源管理	cypress.com/go/powerpsoc
メモリ	cypress.com/go/memory
PSoC	cypress.com/go/psoc
タッチ センシング	cypress.com/go/touch
USB コントローラー	cypress.com/go/usb
ワイヤレス／RF	cypress.com/go/wireless

PSoC®ソリューション

psoc.cypress.com/solutions

PSoC 1 PSoC 3 |PSoC 4 PSoC 5LP

サイプレス開発者コミュニティ

コミュニティ | フォーラム | ブログ | ビデオ | トレーニング

テクニカルサポート

cypress.com/go/support

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

Phone : 408-943-2600
Fax : 408-943-4730
Website : www.cypress.com