# EZ-USB™ FX2LP GPIF 入門

## About this document

### Scope and purpose

FX2LP 汎用プログラマブル インターフェース (GPIF) は、外部インターフェースが必要とするデータと制御信号を生成する独立のハードウェア ユニットを提供します。GPIF は、GPIF のレジスタへの CPU 読み出しと書き込みを使用してデータを移動します。本アプリケーション ノートは、GPIF クロックを 2、4、7 で分周する簡単なデザインを設計することで GPIF ユニットと GPIF Designer というグラフィカル設計ツールを紹介します。このインターフェースを設定と管理するには、たった 3 行の C コードだけが必要です。また本書は、USB 接続機能を GPIF 設計に実装する方法をデモする例を含んでいます。

### 関連プロジェクト

あり

### ソフトウェア バージョン

Keil uVision 2、GPIF Designer

### 関連したアプリケーション ノート

すべてのアプリケーション ノートの一覧を表示するには、ここをクリックしてください。

### 更にサンプル コードをお求めでしょうか? 以下をご参照ください。

USB Hi-Speed サンプル コードの総合リストは、このページをご覧ください。

## Table of contents

Table of contents

# 1 はじめに

USB 2.0 の 480Mbps の信号レートでは、コントローラ チップが高速データを内外に移動することを必要とします。EZ-USB™ FX2LPGPIF は独立したハードウェア ユニットを備えています。CPU はこのハードウェア ユニットをセットアップし、USB エンドポイント FIFO から／へ直接転送されたデータを外部インターフェースに移動します。外部インターフェースは RAM、FIFO、または別のプロセッサです。従って、CPU はデータを移す必要がなく、設定された後、データが GPIF ハードウェア チャネルを介して流れる時に CPU は単にフラグと割込みを監視するだけです。

拡張 IDE (EIDE - 高速 ATA または高速 IDE とも呼ばれる) ／ATA パケット インターフェース (ATAPI) プリンタ、パラレル ポート (IEEE P1284)、Utopia など、GPIF を使用してさまざまなプロトコルを実装できます。本アプリケーション ノートは FX2LP GPIF のアーキテクチャと実装について説明します。本書はアプリケーションの使用モデルとデバッグ処理方法を説明し、GPIF の概念を例を用いて紹介および補足します。

# 2 FX2LP アーキテクチャの概要

EZ-USB FX2LP は、USB2.0 の最大帯域幅に対応するように設計された柔軟な USB 2.0 ペリフェラル コントローラです。FX2LP は GPIF を提供して外部デバイスとの高速なパラレル インターフェースを実装することで USB スループットを最適化します。GPIF は FX2LP エンドポイント FIFO と GPIF インターフェースの間でデータを移動します。以下の節は、FX2LP のあり得る設定可能なモードを示すことで FX2LP アーキテクチャを簡単に説明します。

## 2.1 ポート モード

FX2LP は、モードの設定に応じて異なる役割を果たす 24 本のインターフェース ピンを備えています。「ポート」モードでは、これらは汎用 I/O ピンであり、GPIF は非アクティブになっています (**Figure 1**)。

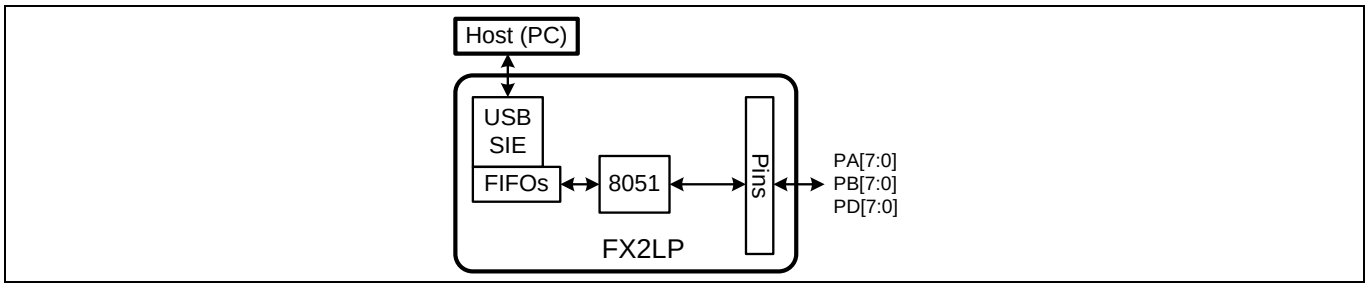


**Figure 1 ポート モードでの FX2LP**

## 2.2 スレーブ FIFO モード

スレーブ FIFO モードでは、専用 FX2LP ロジックは USB エンドポイント FIFO を外部 FIFO コントローラに接続するために制御とデータ信号を提供します。データバスおよび FIFO 選択入力に加えて、インターフェースは RD、WR および FIFO フラグなどの通常の FIFO 信号を提供します。「**AN63787 - EZ-USB FX2LP GPIF and Slave FIFO Configuration Examples Using 8-bit Asynchronous Interface**」を参照してください。

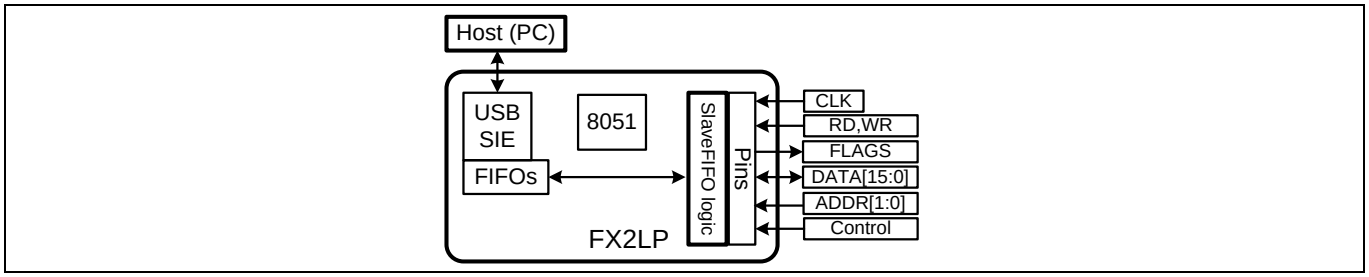


**Figure 2 スレーブ FIFO モードでの FX2LP ピン**

## 2.3 GPIF 自動モード

GPIF がアクティブになった時、インターフェースのピンはマスター デバイスとして機能し、RAM や FIFO または外部プロセッサなどの外部ペリフェラルを制御します。GPIF は自動と手動という 2 つのサブモードで動作します。自動モードでは、データはエンドポイント FIFO から外部インターフェースに直接流れます。8051 プロセッサはこのインターフェースを設定し、監視しますが、FIFO データに直接アクセスしません (**Figure 3** を参照してください)。

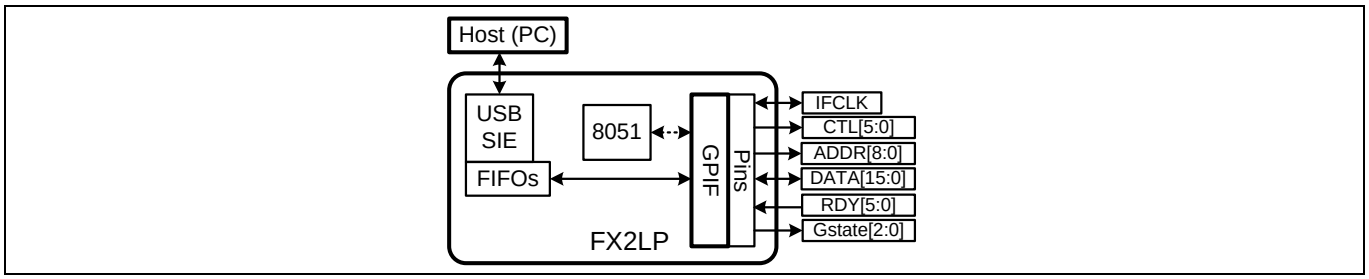


**Figure 3 FX2LP - GPIF 自動モード**

8 ビット非同期インターフェースと GPIF 自動モードを使用してサイプレス CY7C1399B SRAM を FX2LP に接続する方法は「**AN57322 - Interfacing SRAM with FX2LP over GPIF**」を参照してください。

## 2.4 GPIF モード - 手動

手動モードでは、8051 プロセッサは GPIF のレジスタの読み出しと書き込みによりインターフェースから／へのバイト読み書きを行います (**Figure 4**)。GPIF 手動モードのサンプルは**例 3: シングル ワード読み出し／書き込みトランザクションを使用**で説明します。

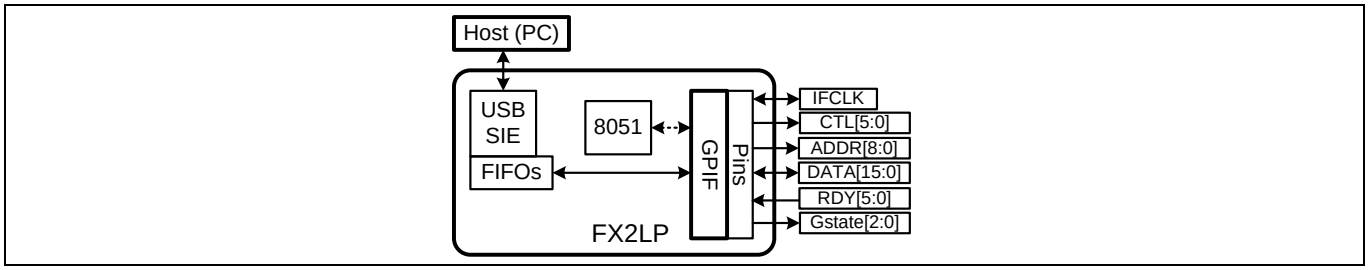


**Figure 4 FX2LP - GPIF 手動モード**

# 3 汎用プログラマブル インターフェース

## 3.1 GPIF の概要

GPIF のコアの中にはプログラム可能なステート マシンがあります。これは、8 ビットまたは 16 ビット双方向データ バスを制御し、最大 6 本の制御 (CTL) と 9 本のアドレス (GPIFADR) 出力を生成します。また、6 本の外部 READY 入力と 2 本の内部 READY 入力を受け入れて分岐の条件を判定します。4 つのユーザー定義の波形ディスクリプタはステート マシンを制御します。8051 の制御プログラムは 4 つの波形から特定の時点でアクティブにする 1 つの波形を選択します。

それぞれの GPIF 波形ディスクリプタには、S0～S6 と名付けられた最大 7 ステートがあります。事前定義の S7 は IDLE ステートとなっています。各ステートで、以下のことを実行するように GPIF をプログラムすることができます。

- CTL 出力のいずれかまたはすべてをドライブ HIGH、ドライブ LOW、またはフローティングにする
- 8 ビット／16 ビット データ バスをサンプリング／駆動する
- GPIF アドレス バスの値をインクリメントする
- FIFO ポインタをインクリメントする
- 8051 への GPIF 波形割り込みをトリガーする

どのステートの分岐決定も、以下のオプションから取られた 2 信号の論理 AND、OR、または XOR の結果と見なされます。

- 6 本の READY 入力ピン
- 1 つの FIFO フラグ
- 1 つの内部 RDY フラグ
- 1 つの内部トランザクションカウント期限切れフラグ

2 つの選択した信号の論理結合は次のステートを決定します。あるいは、プログラム可能な遅延時間が経過すると、ステート マシンが次のステートに進みます。この遅延時間は 1～256 クロック サイクルです。

サンプルとブランチのステートを「判定ポイント」と呼びます。決定点無しのステートは 1 クロック間持続した後、次のクロックで自動的に次のステートに遷移します。

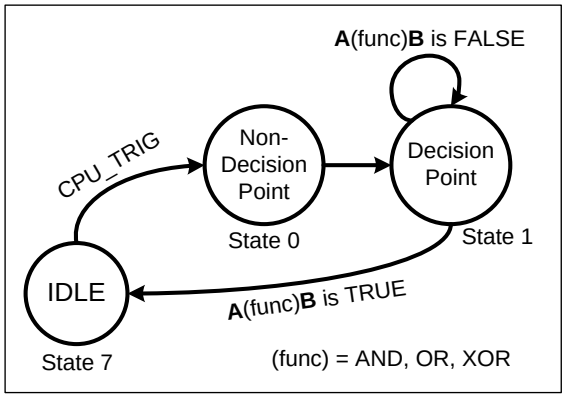


**Figure 5 GPIF ステート マシンの構文**

Figure 5 に示すステート マシンは以下のように表現できます。

1. IDLE ステートから始まり、CPU_TRIG 信号がアサートするのを待ちます。

2. CPU_TRIG がアサートすると、ステート 0 に遷移します。ステート 0 では、幾つかの制御信号出力をアクティブにしたり、データを移動したり、アドレス バスをインクリメントしたりできます。このステートは次のクロックまでしか持続しません。その後、決定がないため、無条件でステート 1 に遷移します。
3. A と B の 2 信号の論理結合が FALSE から TRUE になるまでステート 1 を持続させます。例えば、READY 信号が有効になる、またはカウントが終了するまでステート 1 を持続させるために、A を READY 入力に、B をカウンタの終了値に選択します。ステート マシンは論理演算子 OR を使用し、READY またはターミナル カウント *のいずれか*の条件が発生すると、ステート 1 から移行します。
4. 条件が TRUE になると、IDLE 状態に戻ります。これによって、ステート マシンが停止します。

## 3.2 物理的な相互接続

GPIF の相互接続は、8 または 16 ビット データ バス、1 つのアドレス バス、制御出力、レディー入力を含んでいます (**Figure 4**)。デバッグのために、現時点の GPIF ステート マシンを示す 3 本の GSTATE 出力も含んでいます。本節では、これらの信号を詳しく説明します。

### 3.2.1 IFCLK

IFCLK (インターフェース クロック) は、すべての GPIF 動作用のリファレンス クロックです。これは入力または出力信号であり、立ち上がりまたは立ち下がりエッジをアクティブ エッジに選択できます。入力信号としては、IFCLK は 5MHz～48MHz 範囲の外部クロックにより駆動できます。出力信号としては、IFCLK は FX2LP 内部クロックにより 30MHz または 48MHz のいずれかで駆動できます。外部ペリフェラルがより遅いクロックを必要とする場合、CTL ラインのいずれかは FX2LP 内部クロックを使ってトグルできます。最初の例では、GPIF クロックは 2 と 4 で分周され、信号が 2 本の CTL 出力により出力されます。より高い除数が必要とされる場合、GPIF ステートは次のステートに進む前に、プログラム済みのクロック数 (1～256) をカウントするようプログラムすることができます。

### 3.2.2 GPIFADR[8:0] (出力のみ)

GPIF は GPIFADR[8:0]を駆動してアドレス ラインを必要とするペリフェラルにアドレス ラインを提供します。これらの出力はどの GPIF ステートでも保持するかインクリメントすることができます。

### 3.2.3 FD[15:0] (双方向)

データ バスは、FX2LP のエンドポイント FIFO と外部ペリフェラル間で転送されるペイロード データ用のパイプです。これは 8 ビットまたは 16 ビット インターフェースとして動作するように設定でき、システムが必要とする場合にトライステートにすることができます。16 ビット モードでは、FD[7:0]はエンドポイント FIFO の最初のバイトを表し、FD[15:8]は 2 番目のバイトを表します。

### 3.2.4 CTL[5:0] (出力のみ)

制御出力は、読み出し／読み込みストローブ、イネーブル、分周されたクロックなど外部ペリフェラルが必要とする信号を提供します。

### 3.2.5 RDY[5:0] (入力のみ)

レディー入力信号は、FIFO 状態フラグや使用可能なデータなど外部ペリフェラルからの状態情報を提供します。GPIF はこれらの信号を使用して決定点を検証することができます。

### 3.2.6 GSTATE[2:0] (出力のみ)

デバッグ出力信号は、GPIF 波形として実行されるステートを表します。これらはデバッグのためにロジック アナライザに接続されています。

# 4 GPIF アプリケーションの作成

本節では、GPIF アプリケーションを作成する手順を説明します。

## 4.1 GPIF インターフェースの設計

GPIF の相互接続を設計するために、FX2LP と外部ペリフェラル デバイスとのインターフェースを理解する必要があります。**FX2LP データシート**と**テクニカル リファレンス マニュアル**はインターフェースを定義するために使用されます。GPIF の設定方法は以下の決定に依存します。

- **8 ビットか 16 ビット データ パス?**

この判断は、主にペリフェラルが提供するデータパス サイズに左右されます。16 ビットのデータパスを持つ場合、物理インターフェースを介して帯域幅を最大限にするために 16 ビット幅を使用します。

16 ビット データ パスでは、データ バスを接続する際にエンディアンネス (バイト オーダー) とビットの番号付けも考慮する必要があります。

- **外部か内部インターフェース クロック?**

この決定は、ペリフェラル独自の動作モードに関してはペリフェラルがどのように柔軟であるかに左右されます。例えば、外部の 30MHz または 48MHz クロック入力を受け入れられた場合、内部 GPIF クロックはペリフェラル クロック入力に接続することができます。

- **アドレス ラインが必要?**

ペリフェラルが任意のレジスタまたはメモリ位置が読み出し／書き込みサイクルの動作中にアドレス指定されることを必要とした場合、GPIFADR[8:0]を使用できます。

- **制御ライン**

CTL[5:0]からの GPIF 制御出力を指定します。ペリフェラルは動作中に読み出し／書き込み信号、チップ選択やその他の制御入力を必要とすることがあります。必要な制御信号を決定して、CTL[5:0]を適切に割り当てます。GPIF Designer ツールでは、明確化のためにこれらの信号に名前を付けることができます。例えば、WR#は CTL[0]、RD#は CTL[1]の名前です。

- **ステータス (RDY) ライン**

読み出し／書き込みサイクル中に監視する必要のあるステータス信号の数を判定します。これらを識別して、RDY[5:0]を適切に割り当てます。GPIF Designer は、設計に合わせてこれらの信号に名前を付けることができます。

- **インターフェースのタイミング**

入力と出力が割付けられた後、GPIF アプリケーションの主な部分はインターフェースのタイミングと見なされるタイミング波形を設計することです。

*Note:* *すべての FX2LP パッケージ タイプは GPIF インターフェースの完全な信号セットを提供するとは限りません。例えば、100 ピンと 128 ピン FX2LP パッケージはすべての 6 本のレディー入力 (RDY[5:0]) と制御出力 (CTL[5:0]) を提供しています。56 ピン パッケージは 2 本の RDY 信号 (RDY[1:0]) と 3 本の CTL 信号 (CTL[2:0]) を提供しています。*

## 4.2 ファームウェア フレームワークの使用

GPIF Designer を使用してインターフェース信号と波形を作成する際に、Keil 社の統合開発環境 (IDE) を使用して制御ファームウェアを書きます。新規の FX2LP ファームウェア プロジェクトを開始する際に、サイプレスが提供するファームウェア フレームワーク ベースの Keil uVision2 プロジェクトで開始するのは最も容易な方法です。**FX2LP 開発キット (DVK)** 同梱のサンプル ファームウェアはフレームワークに基づいたものです。これらのいずれかで開始する、または Keil プロジェクトを新しいサブディレクトリにコピーして修正することができます。これにより、クリーンなファームウェア ベースで開発を開始できるようになります。ファームウェア フレームワーク プロジェクトで開発を開始するもう 1 つの利点は、USB の低レベル プロトコル コードが既に書かれているため、ユーザーはアプリケーション コードに集中することができます。詳細は *fw.c* ファイルおよび開発キットの資料を参照してください。

GPIF アプリケーションには 2 つの主なコンポーネントがあります。

- GPIF を設定し、GPIF 転送を起動するファームウェア。このファームウェアは USB エニュメレーションやエンドポイント設定など他のタスクも実行します。
- 物理的なバス タイミングとデータ フローを実装する GPIF 波形ディスクリプタ。

ファームウェアは次の 5 ファイルから構成されています: *fw.c*、*periph.c* (このファイル名が変更可能)、*dscr.a51*、*ezusb.lib*、*usbjmptb.obj*。これらのファイルは Keil uVision 2 ファームウェア フレームワーク プロジェクトを格納しています。

2 番目のコンポーネントは、GPIF 波形を定義し GPIF ユニットを初期化するためのコードを格納している C ソース ファイル (*gpif.c* など) です。インターフェースをグラフで定義した後、GPIF Designer ツールはこの C ファイルを生成します。これにより、GPIF ユニット内の各々のレジスタとビットを知ることが不要になります。**こちら**から GPIF Designer ユーティリティをダウンロードできます。

## 4.3 GPIF Designer を使用して波形を改善

GPIF Designer は、インターフェースの物理的なバス タイミングを実装する GPIF 波形ディスクリプタを含む C コードを生成します。GPIF Designer は複数の波形を実装できます。それらは 4 種類に分けられています: シングル書き込み、シングル読み出し、FIFO 書き込み、FIFO 読み出し。CPU は GPIFWFSELECT レジスタに書き込むことでどの波形を使用するかを制御します。各ディスクリプタは 32 バイトの長さで、内蔵メモリ空間内の特殊な GPIF 波形ディスクリプタ領域に保存されています。GPIF Designer ツールは、これらのディスクリプタを実装して使用するのにすべての必要な C コードを生成するため、これらのディスクリプタを「ブラック ボックス」 (black box) として見なすことができます。

GPIF 波形をステート マシンとして作成することができます。動作用の 7 ステートまたは間隔 (S0～S6) およびトランザクションを終了するための自動 IDLE ステート S7 があります。

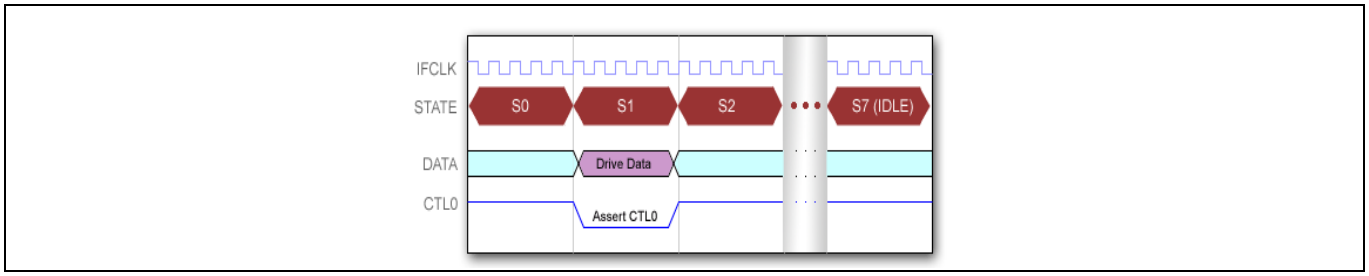


**Figure 6 ステートから構築された GPIF 波形**

**Figure 6** には、GPIF ステート遷移に分けられた簡単な波形の 1 例を示します。

# 5 例 1: GPIF クロックを 2 または 4 で分周

まず、GPIF クロックを 2 または 4 で分周して、分周したクロック信号を CTL[0]と CTL[1]に出力するためのステート マシンを設計して見ましょう。これで、USB エンドポイント FIFO に関する高度な機能を紹介する前に、GPIF Designer ツールの基礎知識を提供します。この例では USB は使用しないため、ファームウェア フレームワークは必要ありません。

1. 本アプリケーション ノートに添付された *FX2LP source code and GPIF project files.zip* ファイルを解凍して保存します。FX2LP Source code and GPIF project files\Firmware\GPIF Clock Divider に移動します。このフォルダを右クリックし、**Properties** をクリックします。「**Read-only...**」オプションが選択されている場合、そのチェックボックスをオフにします。サブフォルダに変更を適用するために **OK** をクリックします。

   このフォルダから、**GPIF_Clock_Divider.uv2** をダブルクリックして Keil uVision2 IDE を起動します。これには、GPIF 情報だけを持っているスケルトン GPIF プロジェクトがあります。*main.c* をダブルクリックして開きます。すると、3 行のコードだけで GPIF 動作を開始できることが分かります。

```
WORD g_data;
GpifInit();
XGPIFSGLDATLX=g_data;// Start transfer
```

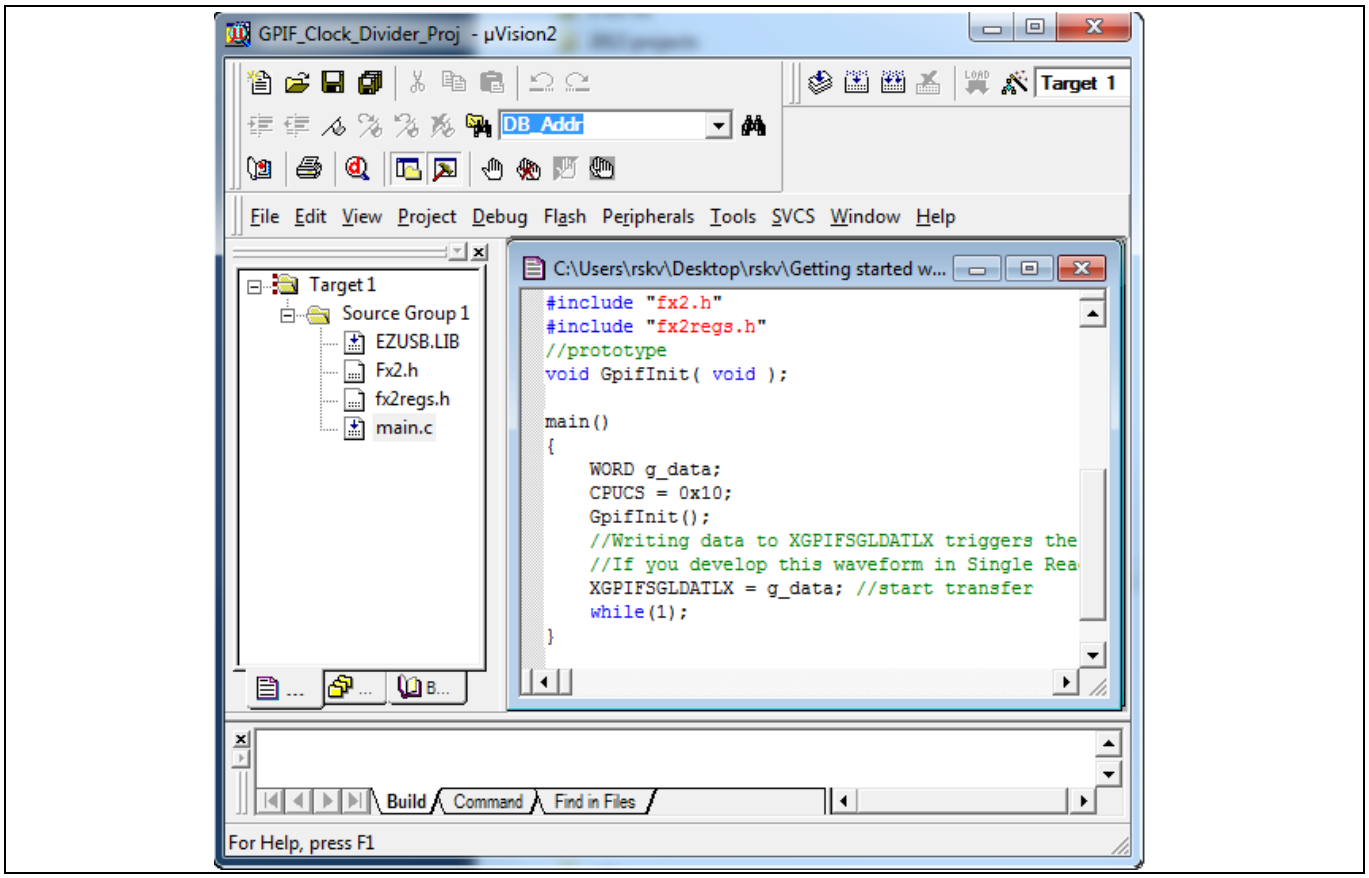


**Figure 7 スケルトン GPIF Keil プロジェクト**

このコードをそのままコンパイルすると、GPIF Designer によって生成された GPIF C が含まれていないため、リンカー エラーが表示されます。このファイルは GpifInit()関数および波形テーブル データを格納しています。次に、このファイルを作成して見ましょう。

### 例 1: GPIF クロックを 2 または 4 で分周

2. GPIF Designer を起動し、**File** > **New** を選択すると、下図のようなウィンドウが表示されます。

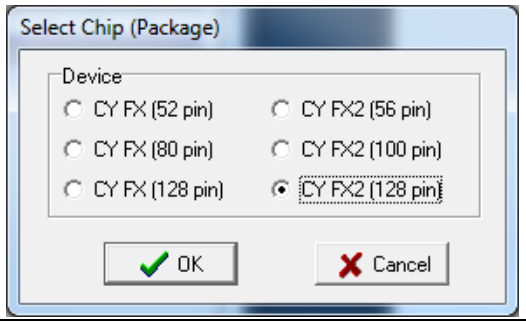


**Figure 8 GPIF Designer のスタートアップ ウィンドウ**

3. FX2LP 開発キット基板を使用してサンプルをテストするために、**CY FX2 (128pin)** を選択して **OK** をクリックします。これにより、ブロック図ウィンドウが表示されます。

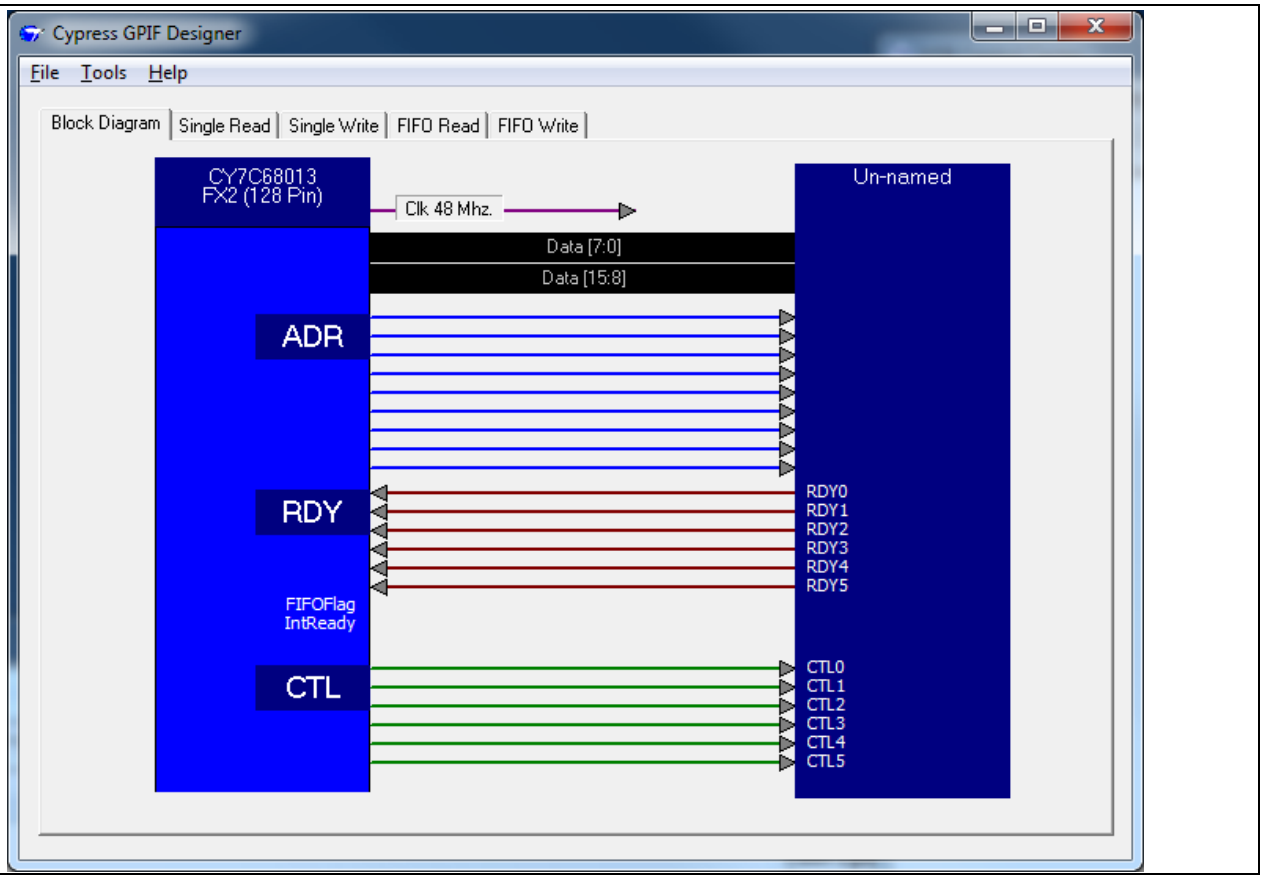


**Figure 9 GPIF Designer のブロック図**

4. ブロック図の最上部で、**Clk 48 MHz** テキスト ボックスを右クリックして GPIF クロックを設定します。このプロジェクトでは、デフォルト設定のままにします。

### 例 1: GPIF クロックを 2 または 4 で分周

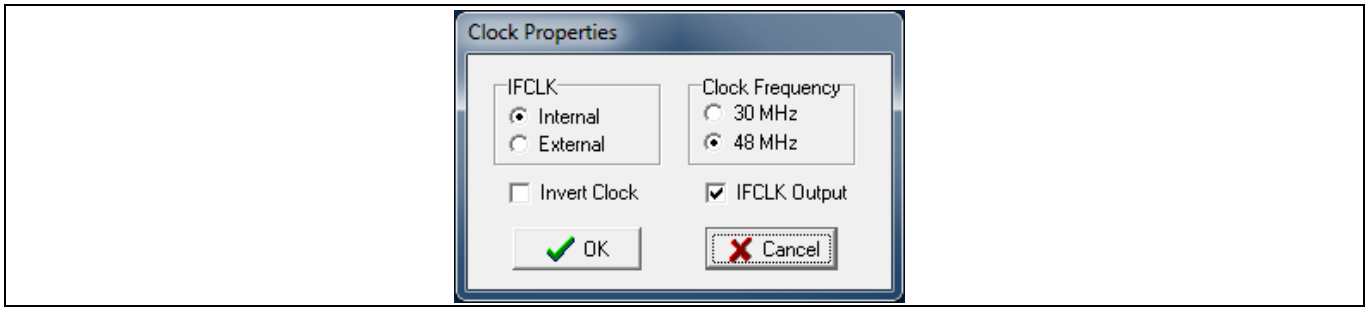


**Figure 10　IFCLK のデフォルト設定**

5. **ADR** (アドレス) ラベルを右クリックします。この例ではアドレス バスを使用しないため、**Disable All** をクリックできます。これにより、ブロック図内のアドレス ワイヤは暗くなります。

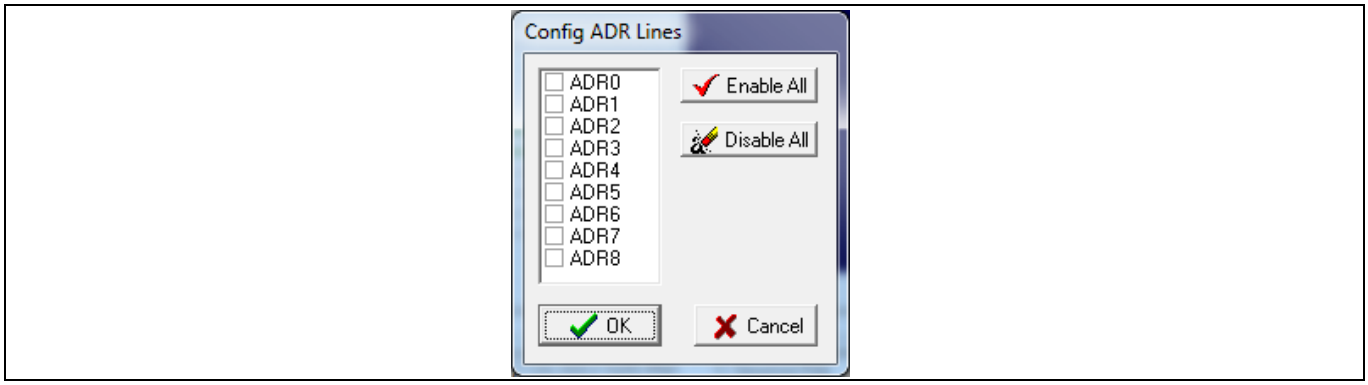


**Figure 11　アドレス ラインの無効**

6. **RDY** (レディー) ラベルを右クリックして 6 本の RDY 入力を設定します。この例では RDY 入力は未使用のため、すべてを無効にすることができます。RDY5 を選択解除するために、まず **Subst TC for RDY5** を選択解除します。

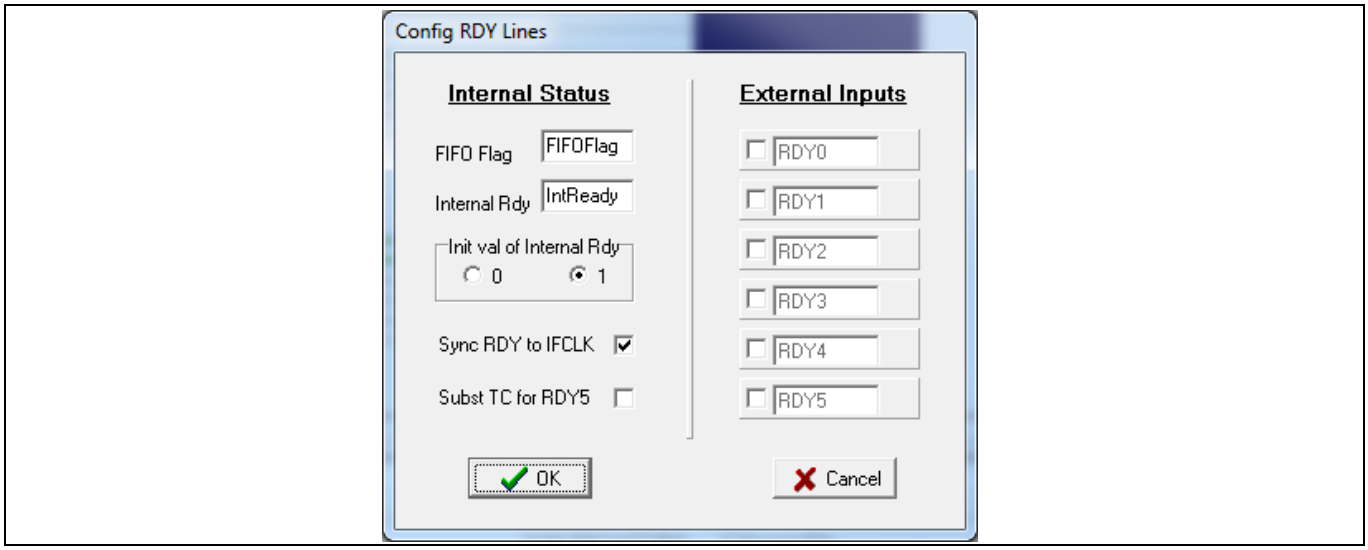


**Figure 12　RDY 入力の無効**

### 例 1: GPIF クロックを 2 または 4 で分周

7. **CTL** (制御) ラベルを右クリックして 6 本の CTL 出力を設定します。これらを下図のように設定します。

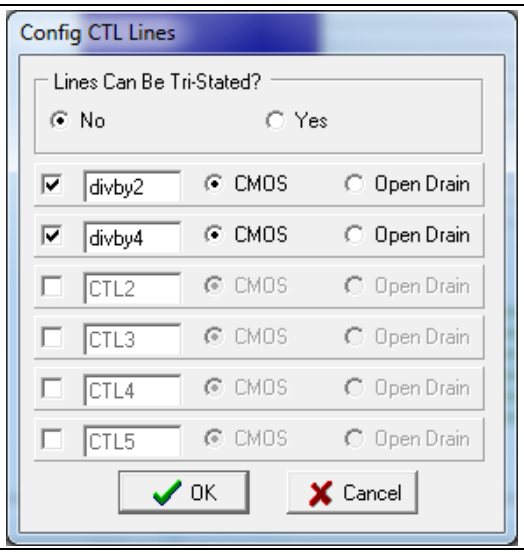


**Figure 13　CTL 出力の設定**

テキスト ボックスに入力して 2 つの出力名を **divby2** と **divby4** に変更します。信号を名付けると、すべての GPIF Designer 画面でラベルが更新され、信号が識別しやすくなるため (例えば、CTL1 でなく divby4)、これは良い実践方法です。これで、ブロック図がもっと簡単になります。

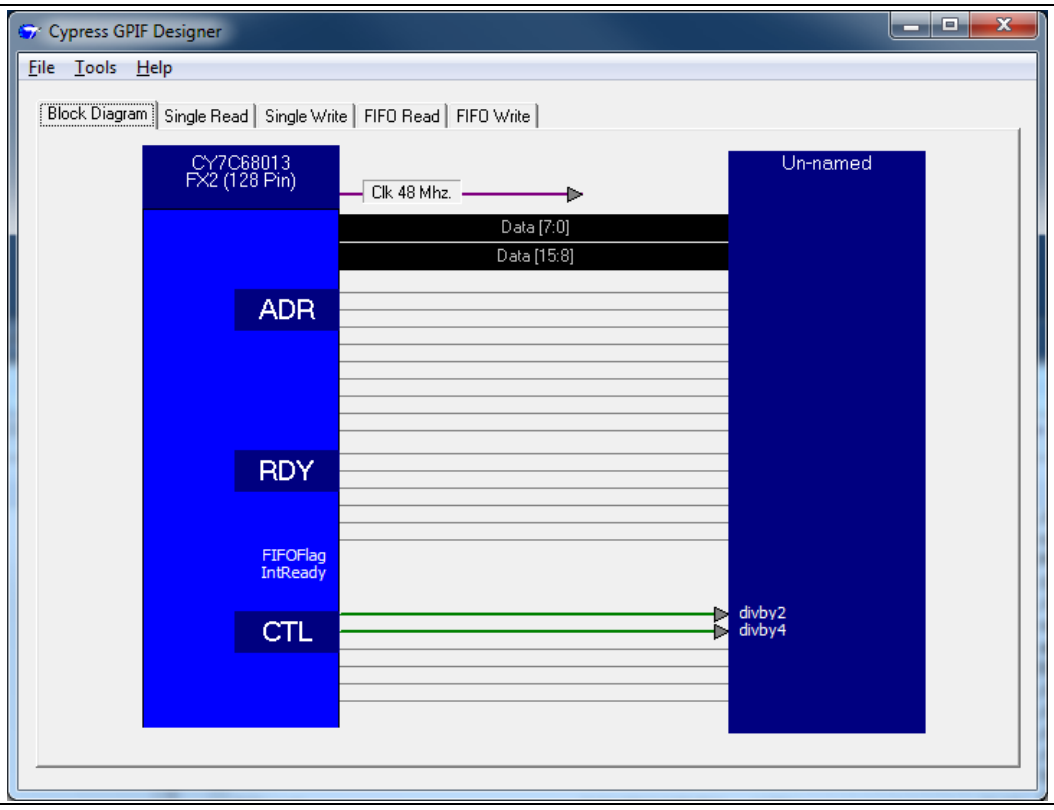


**Figure 14　クロック分周器のブロック図**

### 例 1: GPIF クロックを 2 または 4 で分周

8. 2 本の波形を描くための **Single Write** タブを選択します。タブ名を右クリックし、**Select Tab Label** を選択し、名前を **Divider** に変更します。**Figure 15** のように空の波形編集画面が表示されます。

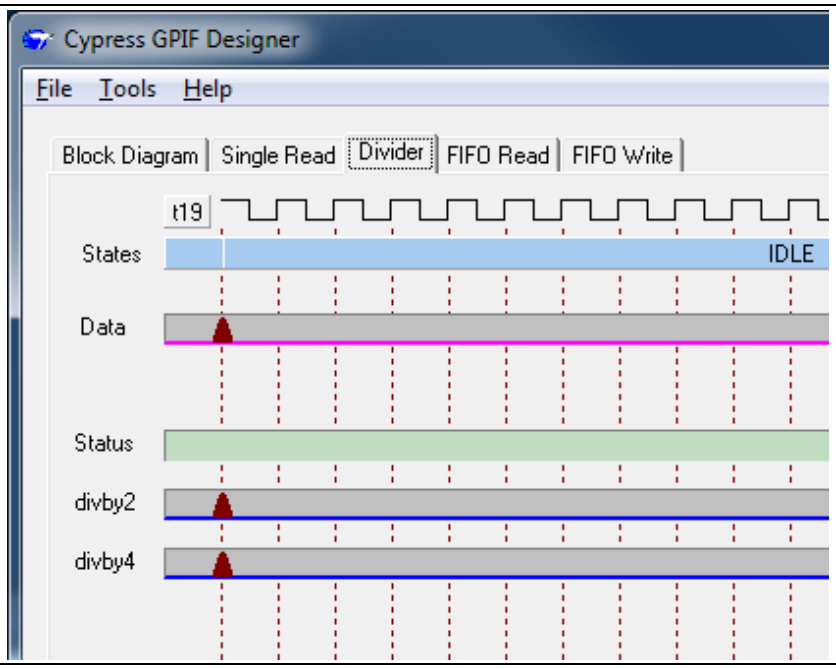


**Figure 15　空の波形編集画面**

9. ステート マシンには IDLE というステートがあります。ステートを追加するために、divby2 および divby4 と名付けられた CTL バンドのどちらかをクリックします。Data バンドをクリックしてステートを追加することもできますが、この設計ではデータ バスを使用していません。divby2 バンドでは、2 番目の垂直の点線をクリックします。

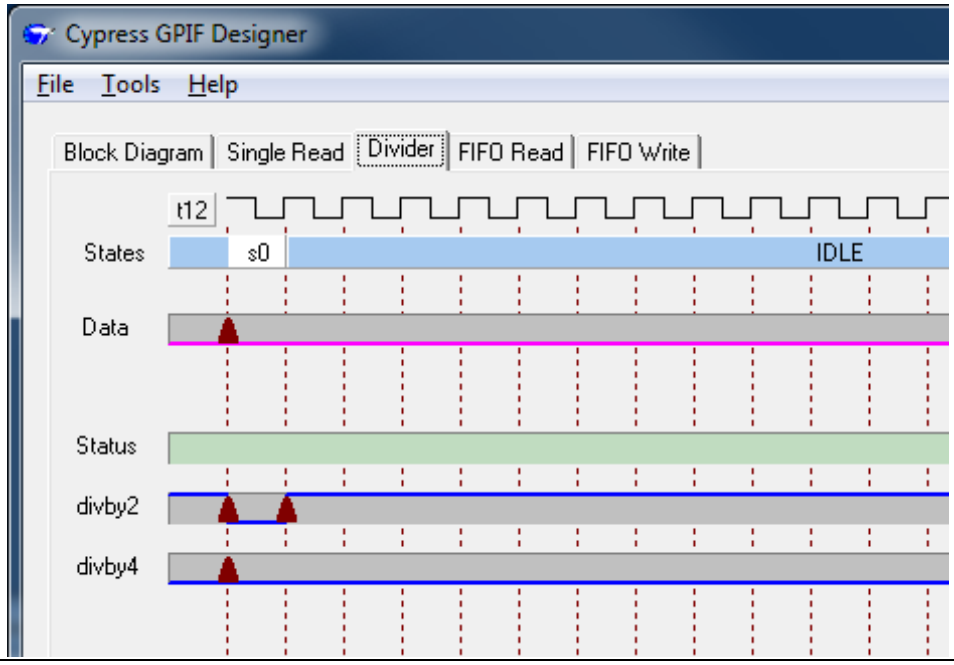


**Figure 16　最初の波形遷移**

### 例 1: GPIF クロックを 2 または 4 で分周

三角形が表示されて、GPIF Designer がステート s0 を追加し、s0 の始まりと終わりで divby2 出力をトグルしたことを示します。

10. これで、次の 3 本のクロック遷移点線ごとに divby2 波形をクリックします。これにより、クロックの divide-by-2 波形が作成されます。
11. divby4 波形では、2 番目と 4 番目の線をクリックします。画面が **Figure 17** のようになります。


**Figure 17 Div by 2 と Div by 4 波形**

*Note: どの三角形も右クリックして論理ステートを変更することで信号極性を修正することができます。三角形を水平にドラッグしてステートを 1 クロックより長くすることもできます。*

**Figure 17** のステート マシンは開始した時、各立ち上がりエッジで s0、s1、s2、s3 ステートの順番で遷移して IDLE ステートで停止します。そのように繰り返す必要があります。つまり、s3 は常に s0 に戻って分岐しなければなりません。このために、Status ラインで「決定点」を追加します。

### 例 1: GPIF クロックを 2 または 4 で分周

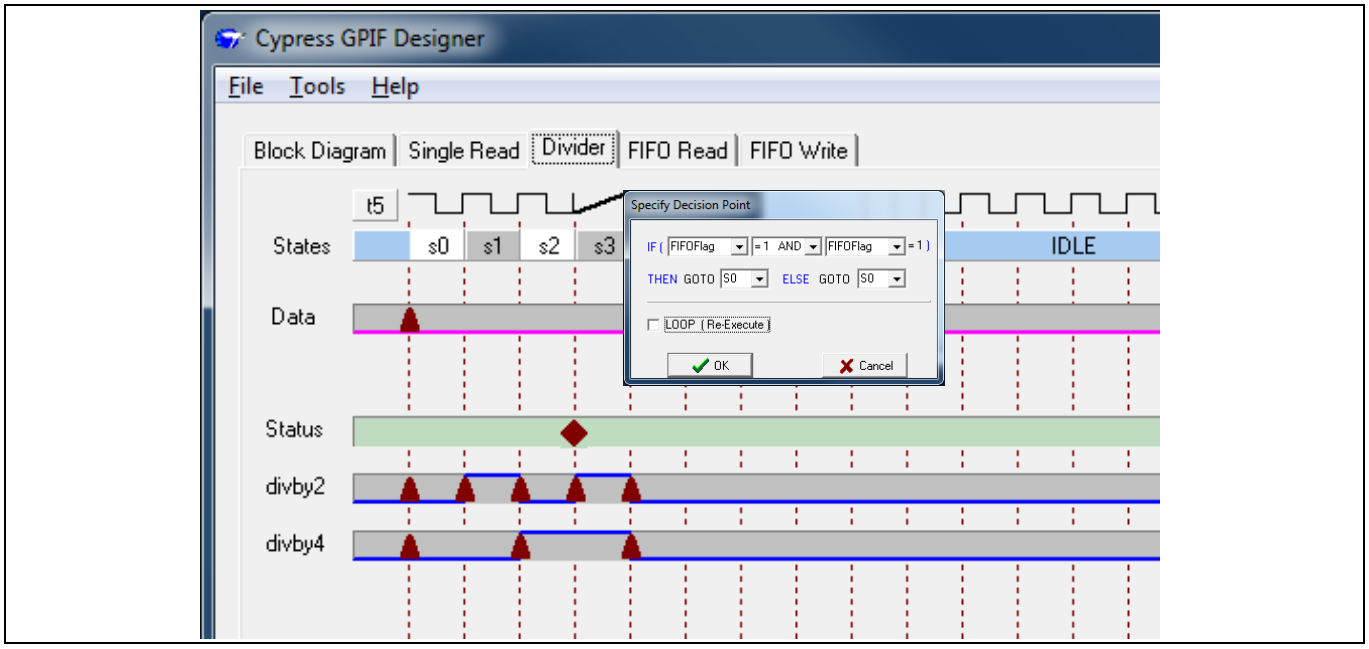


**Figure 18　最初の決定点**

12. s2 と s3 ステート間の Status バンドをクリックします。すると、決定点としてダイヤモンド形が追加され、決定を設定するためのウィンドウが表示されます。無条件分岐を行うために、最上部のドロップダウン リストから任意の 2 本の論理信号およびそれらに任意の論理演算子を選択できます。S0 への遷移のために THEN と ELSE 両方の条件を指定する必要があります。即ち、*無条件で* S0 に分岐します。**LOOP (Re-Execute)**チェックボックスは、ステートに遷移するたびにステート条件を再アサートするのに使用されます。s3 には「変更されるまでこのステートを持続させる」ループがないため、チェックを入れないことができます。

    クロック波形は s3 に斜線を表示して、決定ステート (decision state) のクロック数が未定であることを示します。 理由は、クロック数が分岐条件が満たされる時点に因るためです。

    GPIF Designer は設計で使用するすべてのパラメータを追跡してそれに応じて選択項目を更新します。例えば、ステート s4 を追加すると、これが自動的に GOTO 選択項目に追加されます。

    **重要な注意:** 決定点はそれが発生するステートの始まり (終わりではない) に置きます。この理由で、ダイヤモンド形は s3 の始まりに置いてあります。

13. GPIF Designer プロジェクトを保存するために、**File** > **Save As** を選択し、Keil プロジェクト フォルダに *Div_2_4.gpf* として保存します。波形を修正するために後でファイルを再び開くことができます。GPIF Designer が生成した C コードを保存するために、**Tools**>**Export to GPIF.c file** を選択し、Keil プロジェクト フォルダに *GPIF_div_2_4.c* として保存します。*Div_2_4.gpf* と *GPIF_div_2_4.c* ファイルは FX2LP Source code and GPIF project files\GPIF Clock Divider\GPIF_div_2_4 フォルダに用意されています。それらを直接使用してプロジェクトをテストすることができます。
14. 最後の手順は GPIF Designer の C ファイルを Keil プロジェクトに含めることです。**Sources Group 1** を右クリックして **Add files...**を選択します。Keil プロジェクト フォルダに移動してそれに *GPIF_div_2_4.c* ファイルを入れます。

### 例 1: GPIF クロックを 2 または 4 で分周

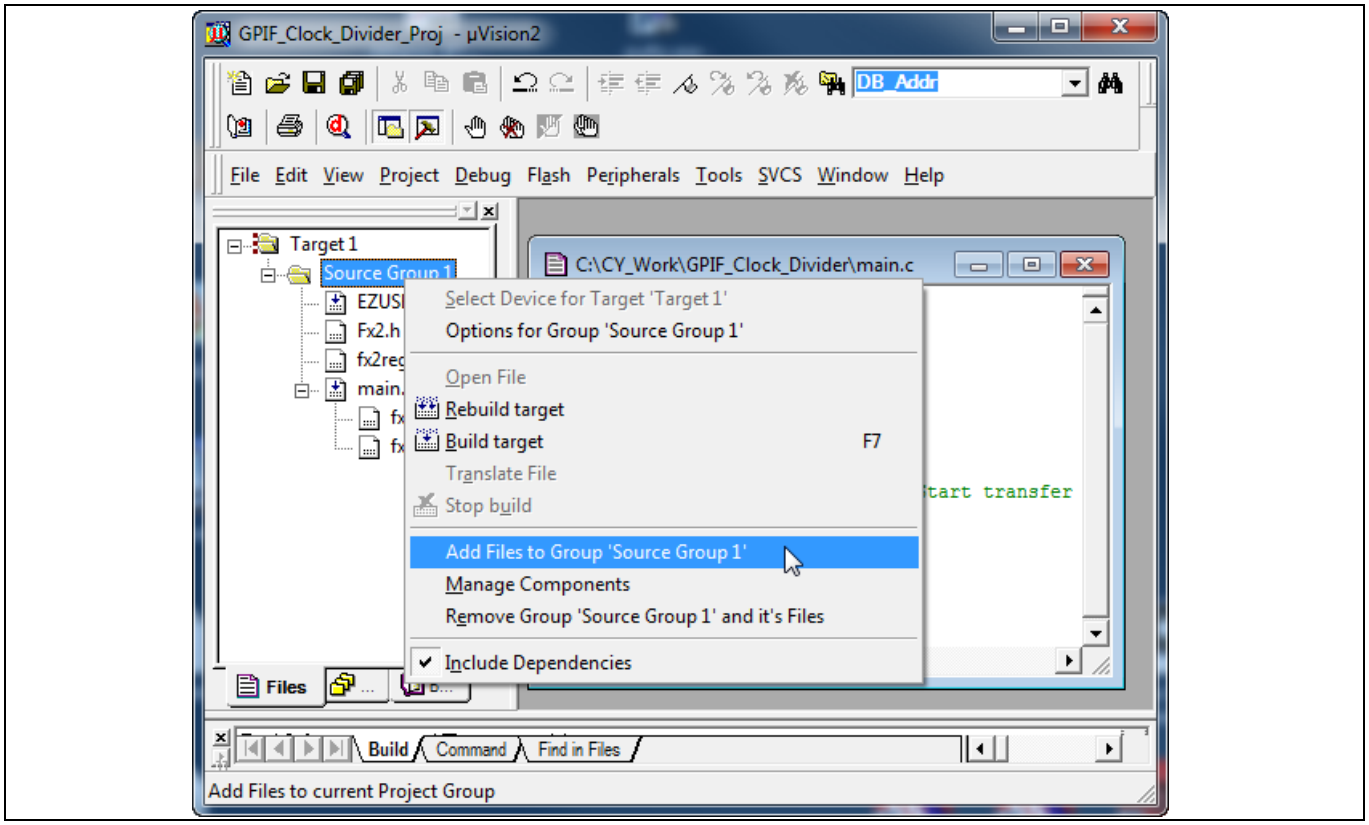


**Figure 19 GPIF Designer の C ファイルを Keil プロジェクトに追加**

15. プロジェクトをコンパイルします。Keil IDE はファイルをコンパイルしてリンクさせた後、ロード モジュールをサイプレス USB コントロール パネル ローダと互換性があるフォーマットに変換するために「hex2bix」ユーティリティを呼び出します。

    以下のように設計を FX2LP 開発基板上に試すことができます。

    a) EEPROM SELECT スイッチをオフの位置 (下) にします。すると、FX2LP の搭載 USB コード ローダが有効になります。

    b) FX2LP 基板を PC USB ポートに接続します。これが初めての場合、USB ドライバをインストールするようポップアップ メッセージが表示されます。C:\Cypress\USB\CY3684_EZ-USB_FX2LP_DVK\1.0\Drivers\cyusbfx1_fx2lp に移行し、Windows OS に応じたフォルダを選択します。Windows Device Manager にてドライバのインストールを確認します。

### 例 1: GPIF クロックを 2 または 4 で分周

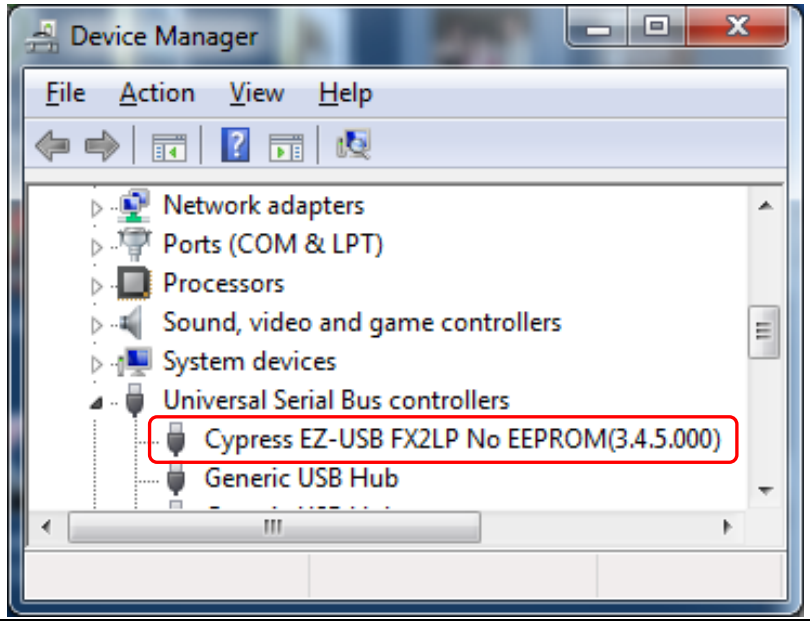


**Figure 20　サイプレス USB ローダ ドライバのインストールが完了**

16. C:\Cypress\Cypress Suite USB 3.4.7\CyUSB.NET\bin\CyControl.exe から USB コントロール パネルを起動します。
17. FX2LP 基板および他の接続された USB デバイスが左パネルに表示されます。FX2LP 基板だけを表示するために、右パネルの **Device Class Selection** タブをクリックし、**Devices served by the CyUSB.sys driver (or a derivative)**アイテムを除いてすべてのチェックボックスをオフにします。左パネルが **Figure 21** のようになります。

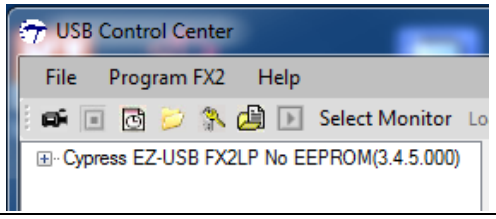


**Figure 21　USB コントロール センターに表示された FX2LP 基板**

18. これで、Keil のコンパイルされた HEX ファイルを基板にロードできるようになります。EZ-USB エントリーをハイライトし、**Program FX2** > **RAM** を選択します。

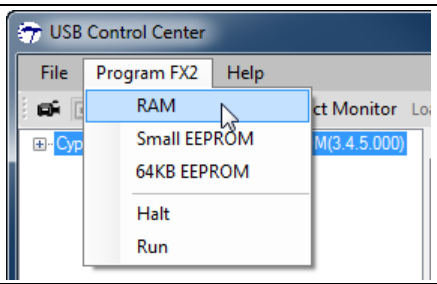


**Figure 22　新しいデバイス コードを RAM にロード**

### 例 1: GPIF クロックを 2 または 4 で分周

19. FX2LP 開発基板上の **RESET** ボタンを押します。すると、USB ローダが有効になります。
20. Keil プロジェクト フォルダに移動して Keil コンパイラの結果である *GPIF_Clock_Divider_Proj.hex* ファイルを選択します。
21. P2 ピン 11 (CTL0=divby2) と P2 ピン 10 (CTL1=divby4) をプローブ (probe) します。波形が **Figure 23** のようになります。

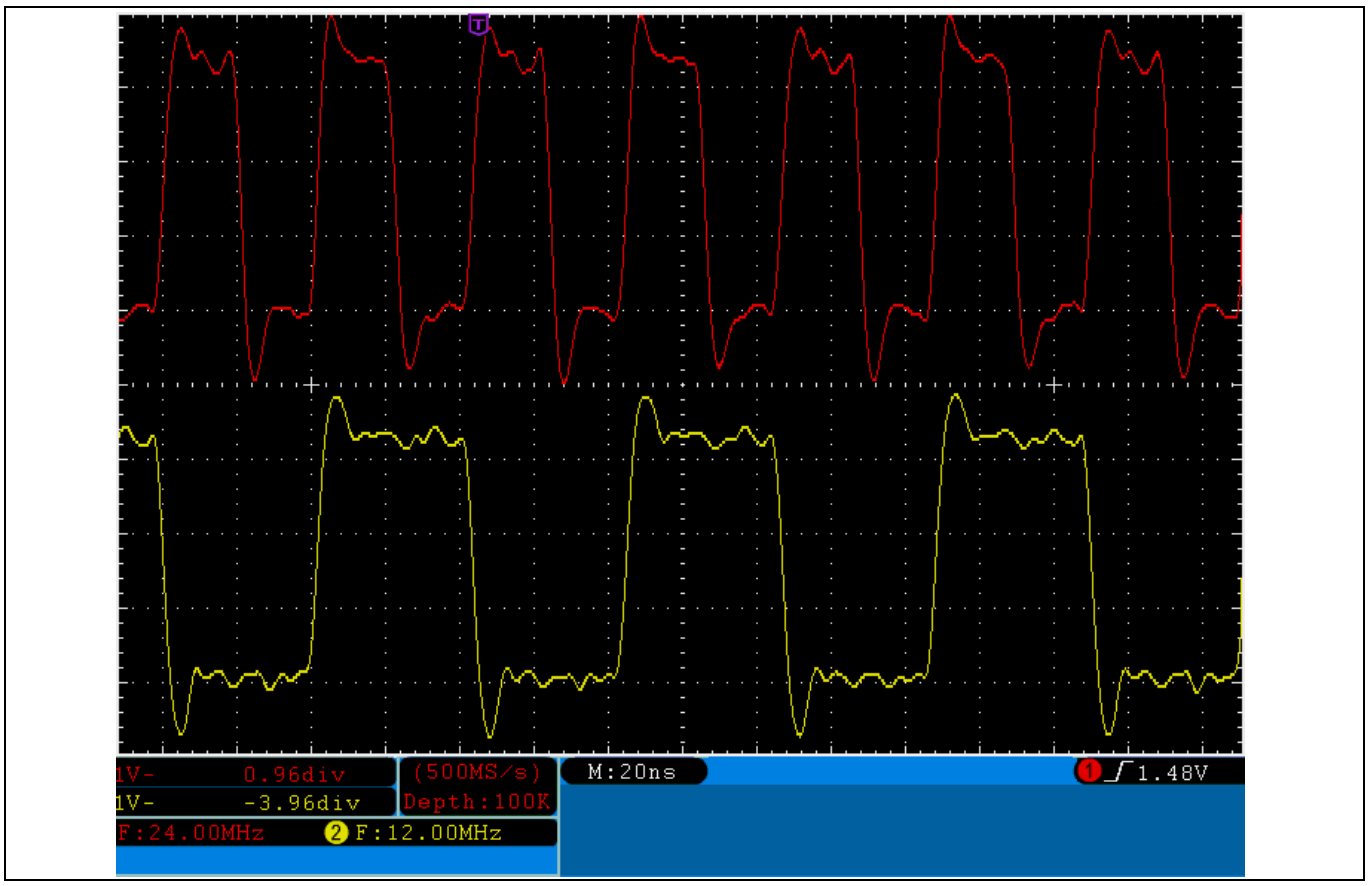


**Figure 23　2 と 4 で分周された GPIF 48MHz クロック**

# 6 例 2: GPIF クロックを 7 で分周

1. 例 1 を修正することで任意のクロック分周器を選択でき、奇数の除数を使用しても構いません。
2. GPIF Designer を起動し、**File** > **New** を選択して新規の GPIF プロジェクトを作成します。**CY FX2(128 pin)** デバイスを選択します。
3. 上記のように ADR と RDY 信号を除去し、2 本の CTL 信号の名前 CTL0 と CTL1 をそれぞれに divby7 と stb に変更します。残りの 4 本の信号を無効にします。

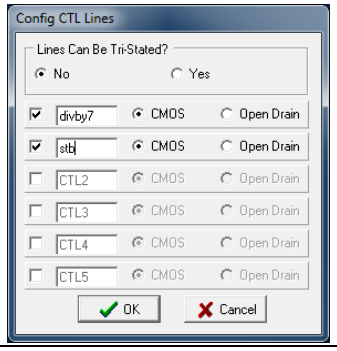


**Figure 24 2 本の出力「Divide by 7」と「Strobe」**

4. **Single Write** タブを選択します。divby7 バンドでは、最初の三角形から 3 クロック後にステート遷移の三角形を置き、それから 4 クロック後にもう 1 つの三角形を置きます。stb バンドでは、その始まりから 1 クロック後に三角形を置きます。最初の三角形を右クリックして **HI** にセットし、2 番目の三角形を **LO** にセットします。すると、s0 で負極性パルスが正極性パルスに変換されます。波形が **Figure 25** のようになります。

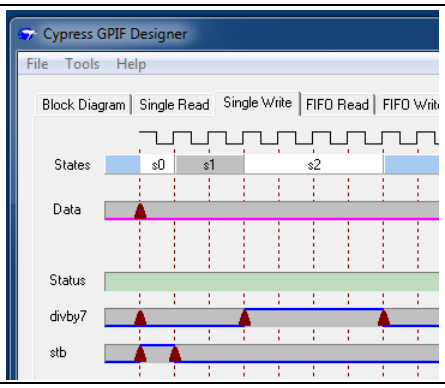


**Figure 25 「Divide by 7」と「Strobe」の波形**

最後に、s2 の終わりの 1 クロック前に対応する位置で Status バンドのクロック ラインをクリックします。すると、決定点と新しいステート s3 が作成されます。決定点を S0 に無条件で分岐するよう設定する必要があります。THEN GOTO と ELSE GOTO の両ステートが S0 と選択されたため、IF 条件は任意となり、ドロップダウン リストからどれも選択できます。

### 例 2: GPIF クロックを 7 で分周

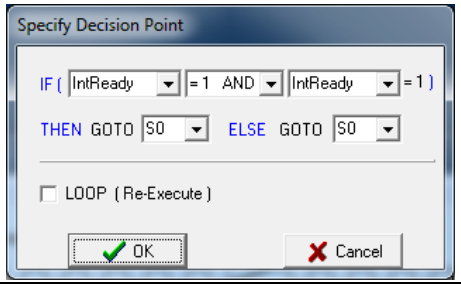


**Figure 26　S0 に無条件に分岐**

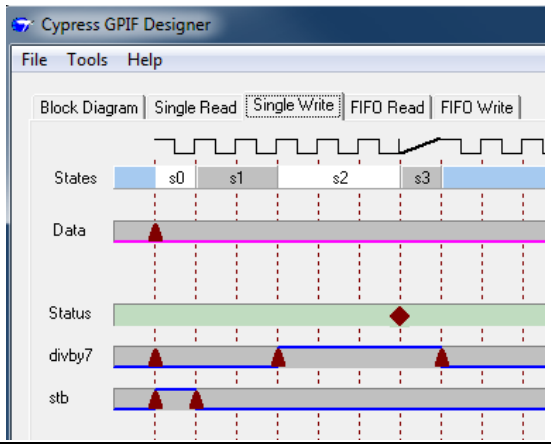


**Figure 27　最終の「Divide-by-7」波形**

5. このファイルを Keil プロジェクト フォルダに *GPIF_div_7.c* としてエクスポートします (**Tools**>**Export to GPIF.c**)。プロジェクトファイルを *Div_by_7.gpf* として保存します (**File**>**Save As**)。*Div_by_4.gpf* と *GPIF_div_7.c* ファイルは `FX2LP Source code and GPIF project files\GPIF Clock Divider\GPIF_div_7` フォルダにも用意されています。それらを直接使用してプロジェクトをテストすることができます。
6. Keil Files タブで、**Source Group 1** を右クリックし、**Add Files...**をクリックして *GPIF_div_7.c* をプロジェクトに追加します。これで、2 つの GPIF 波形ファイルがあるため、Keil コンパイラはどのファイルを使用するかを知る必要があります。ビルドから何らかのソース ファイルを無効にするために、それを右クリックして **Options for file...**を選択し、**Include in Target Build** チェックボックスをオフにします。プロジェクトに 2 つ以上の *GPIF.c* ファイルがある場合、チェックボックスがオンになっているものが 1 つだけであることを確認します。
7. **Project** > **Rebuild All Target Files** を選択して再びコンパイルします。
8. FX2LP 開発基板上の **RESET** ボタンを押します。すると、USB コード ローダが元に戻ります。
9. 上記のように USB コントロール センターを使用して新しくコンパイルされた *GPIF_Clock_Divider_Proj.hex* ファイルをロードします。
10. P2 ピン 11 (CTL0=divby7) をプローブします。画面が **Figure 28** のようになります。

### 例 2: GPIF クロックを 7 で分周

**Figure 28　最上部:「Divide by 7」；最下部:「stb」；信号は 1 クロック幅 (参照のため)**

div-by-87 などのようなより高い除数のためには、何をする必要があるのでしょうか? GPIF Designer では、ステートの期間を 1〜256 クロックに設定できます。States バンドで任意のステートをクリックします。**Figure 29** のようなウィンドウが表示されます。**Set State Duration** をクリックし、クロックの数を設定します (**Figure 30**)。ステートのクロック数を手動でセットすると、タイミング図のクロック ラインはもはやステートのクロック数を示さなくなります。

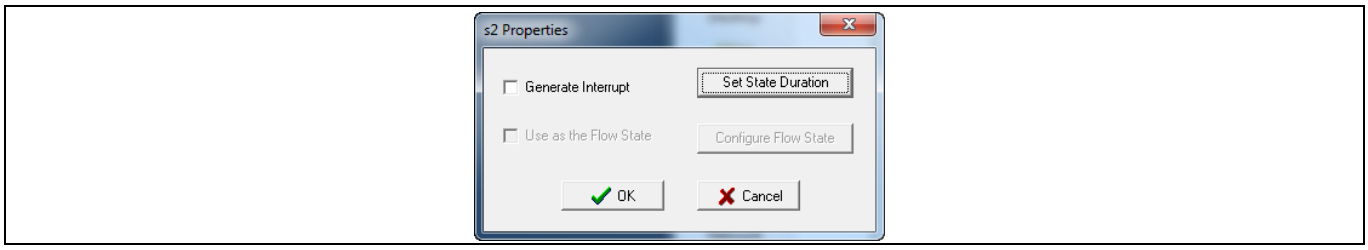


**Figure 29　s2 のプロパティ**

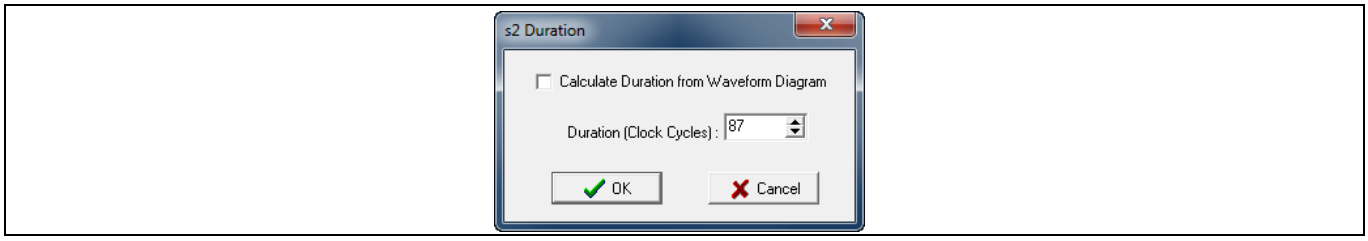


**Figure 30　1〜256 クロック サイクルを選択**

# 7 例 3: シングル ワード読み出し／書き込みトランザクションを使用

この例では、サイプレス CY7C4265-15AXC の外部 FIFO を介して FX2LP データをループします。GPIF 手動モードを使用し、16 ビット データ バスを介してシングルの読み出し／書き込み GPIF トランザクションを実装します。FIFO 書き込み動作はスコープまたはロジック アナライザで監視できますが、読み出しのテストの場合は、外部 FIFO チップを FX2LP 開発基板に接続する必要があります。

この例の場合、開発キット同梱の試作用基板を使用して外部 FIFO を FX2LP 開発基板に搭載します。外部 FIFO の完全なハードウェア仕様については、**CY7C4265 データシート**をダウンロードして参照してください。FX2LP 開発基板と試作基板を接続する際のピン配置リストおよび外部 FIFO 試作用基板の完全な回路図は、本アプリケーション ノートに添付された「Hardware」フォルダにあります。

シングル読み出し／書き込みトランザクションは FX2LP と外部ペリフェラルの間で 1 データ バイトまたは 1 データ ワードを転送します。これらのトランザクションは FIFO 読み出し／書き込みトランザクションより実装しやすく、最初に実装することが推奨されています。GPIF 開発サイクルの最初にこの段階を実装することで、より複雑な設計に進む前にシステムのすべてのエリア (ハードウェア、ファームウェア、ソフトウェア) を検証することができます。物理的な相互接続と基本的なデータ移動を検証した後、完全な設計に進むことが容易になります。

## 7.1 FIFO 読み出し／書き込みトランザクションを実装

シングル トランザクションを実装した後、次の手順としては、より高い帯域幅を達成するために GPIF FIFO 読み出し／書き込みトランザクションを実装します。最初に FIFO 書き込み波形を実装することで、完全なループバック ソリューションをテストする際に発生し得る問題を回避できます。これを実現できないなら、書き込み、読み出し、または両方から問題を切り分けることが困難になります。

## 7.2 必要に応じて最適化

GPIF 波形を生成する際、初期に物理的バスの HIGH 時間を設定することがあります。設定した場合、ペリフェラルが必要としたタイミング パラメータを満たしながら各 GPIF トランザクションのサイクル時間を削減することができます。この段階では、設計を改善してファームウェア コード効率と全体的なファームウェア コード フローを改善させることができます。

**Figure 31** は GPIF 設計フロー図で本節の手順をまとめています。

例 3: シングル ワード読み出し／書き込みトランザクションを使用

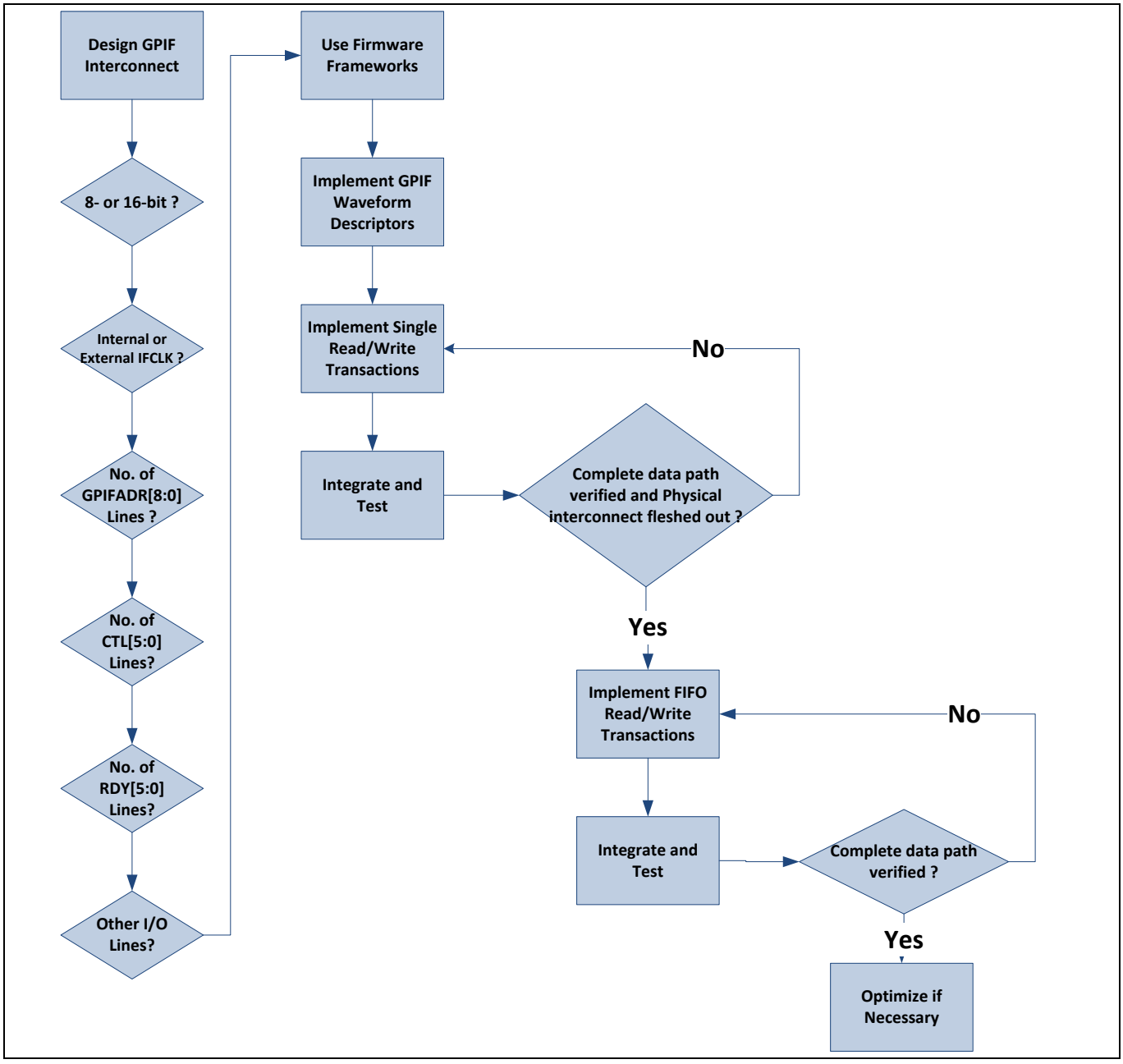


**Figure 31　GPIF 設計フロー図**

## 7.3　FIFO を FX2LP GPIF インターフェース

**Figure 32** は FX2LP と外部 FIFO の接続を示します。FX2LP は双方向バス FD[15:0]を使用して外部 FIFO から／へデータを読み書きします。FIFO データ バスは、その出力データ バス Q[15:0]と入力データ バス D[15:0]を接続することで双方向にします。この例では、USB BULK 転送を使用して FIFO データを読み書きします。これらの転送は、**Suite USB 3.4 - Visual Studio 向けの USB 開発ツール**同梱の USB コントロール センター ユーティリティを使用して実行できます。

**Table 1** は GPIF の相互接続を詳しく説明します。CTL と RDY ピンは最小の FX2LP パッケージ (56 ピン) と互換性があるように割り当てられています。

### 例 3: シングル ワード読み出し／書き込みトランザクションを使用

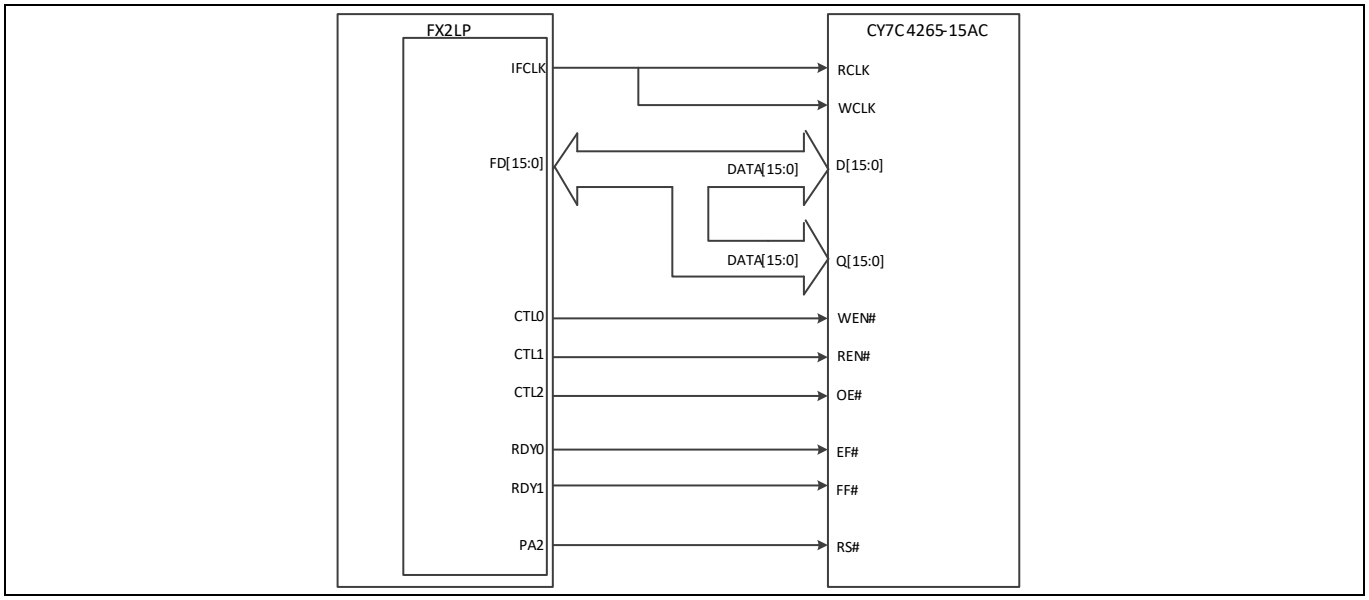


**Figure 32　GPIF の外部同期 FIFO との接続**

**Table 1　　FX2LP GPIF 信号の CY7C4265-15AC 信号への割り当て**

| FX2LP の GPIF 信号 | CY7C4265-15AXC の信号 | 説明 |
|---|---|---|
| IFCLK | WCLK、RCLK | IFCLK は、外部 FIFO の書き込みと読み出しクロック入力 (WCLK、RCLK) に接続。WEN#がアサートされた時、データは WCLK の各立ち上がりエッジで外部 FIFO にクロック入力。同様に、REN#と OE#がアサートされた時、FIFO は RCLK の各立ち上がりエッジで Q[15:0]にデータを出力。外部 FIFO は最大 66.7MHz までの入力クロック周波数を受け入れられるため、30MHz または 48MHz の入力 IFCLK 周波数に対応可能。注: ハッシュ マーク (例えば、OE#) はアクティブ LOW を意味 |
| FD[15:0] | D[15:0]、Q[15:0] | ワード幅の動作のために、GPIF データ バス FD[15:0]は外部 FIFO の入力データ バス D[15:0]に接続。FX2LP が FIFO データ内容を読み戻せるために、外部 FIFO の出力データ バス Q[15:0]も GPIF データ バスに接続。2 本の単方向 FIFO データ バスが互いに接続されているため、GPIF はバス競合が発生しないように OE#信号を制御する必要。特に、FX2LP がデータ バスを駆動している時、OE#は LOW になってはならない。その場合、FX2LP と FIFO がデータ バスを同時に駆動するためバス競合が発生する |
| CTL0 | WEN# | CTL0 は外部 FIFO の書き込みイネーブル ピン WEN#に接続。GPIF が WEN#を LOW に維持している間、データは WCLK の各立ち上がりエッジで外部 FIFO に書き込まれる |
| CTL1 | REN# | CTL1 は外部 FIFO の読み出しイネーブル ピン REN#に接続。GPIF が REN#と OE#を LOW に維持している間、FIFO は RCLK の各立ち上がりエッジで新しいデータを Q[15:0]に駆動 |
| CTL2 | OE# | CTL2 は外部 FIFO の出力イネーブル ピン OE#に接続。REN#と OE#が LOW に維持されている間、FIFO は RCLK の各立ち上がりエッジで新しいデータを Q[15:0]に駆動 |

### 例 3: シングル ワード読み出し／書き込みトランザクションを使用

| FX2LP の GPIF 信号 | CY7C4265-15AXC の信号 | 説明 |
|---|---|---|
| RDY0 | EF# | RDY0 は外部 FIFO の EMPTY フラグ EF#に接続。FIFO は空の時、このフラグをアサート (LOW にする)。READY 信号が各 GPIF 分岐ステートでテストできるため、GPIF は外部 FIFO から読み出している間にこの信号を使用してデータ転送を調整することが可能 |
| RDY1 | FF# | RDY1 は外部 FIFO の FULL フラグ FF#に接続。FIFO は満杯の時、このフラグをアサート (LOW にする)。GPIF は外部 FIFO に書き込まれている間にこの信号を使用してデータ転送を調整することが可能 |
| PA2 | RS# | PA2 は外部 FIFO の RESET ピンに接続。PA2 は FX2LP の GPIO ピンであり、GPIF ロジックの一部ではない。8051 コードは PA2 を使用して、GPIF データ転送が開始される前の既知のステートに外部 FIFO をリセット |

# 8 USB データ フロー

USB のエンドポイント 2 OUT (EP2OUT) は外部 FIFO への GPIF 書き込みのソース エンドポイントとして使用され、エンドポイント 6 IN (EP6IN) は外部 FIFO からの GPIF 読み出しのシンク エンドポイントとして使用されます。USB の IN と OUT の方向はホストを中心にしています:

- EP2OUT は、USB ホスト (PC) によって送信されて FX2LP によって受信されたデータ パケットを格納します。
- EP6IN は、FX2LP によって送信されて PC によって受信されたデータ パケットを格納します。

# 9 GPIF の相互接続を設計

GPIF の相互接続は最初の例で述べられた手順に従ってセットアップし、本節では簡単に説明します。

1. サイプレス GPIF Designer ツールを起動します。
2. FX2LP 開発基板で使用されたように、**File** > **New** に移動して **CY FX2 (128 pin)**を選択します。
3. 外部デバイス ブロックで **Un-named** ラベルを右クリックしてその名前を **CY7C4265-15AC** に変更します。
4. 初期設定ではデータ バスは 16 ビットです。この例は 16 ビット データ バスを使用します。
5. **ADR** ラベルを右クリックします。この設計ではアドレス ラインを使用しないため、ブロック図を簡略化するためにそれらを無効にします。
6. **RDY** ラベルを右クリックし、**Figure 33** のように RDY 信号を設定します。

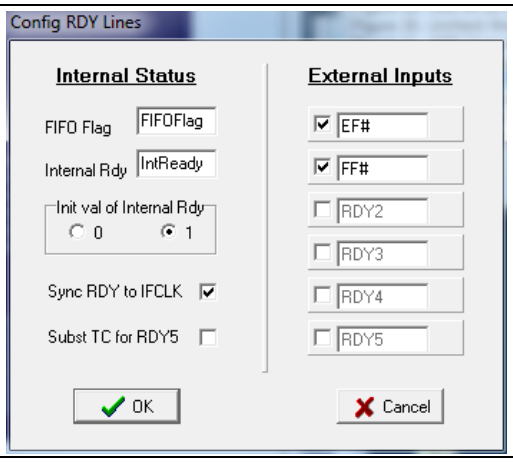


**Figure 33 RDY ピンのコンフィギュレーション**

7. **CTL** ラベルを右クリックして、**Figure 34** のように CTL 信号を設定します。

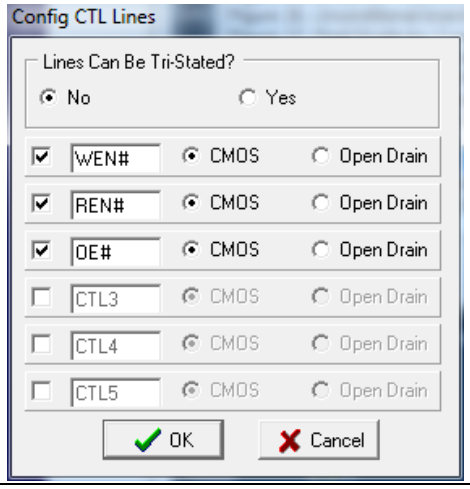


**Figure 34 CTL ピンのコンフィギュレーション**

8. **48 MHz CLK** テキスト ボックスを右クリックして、**Figure 35** のようにクロック特性を設定します。**Invert Clock** チェックボックスをオンにします。GPIF は立ち上がりエッジで信号を変更およびサンプ

### GPIF の相互接続を設計

リングし、FIFO は立ち下がりエッジで信号を変更およびサンプリングします。この半クロックのオフセットにより、インターフェースは十分なセットアップとホールド時間を有します。

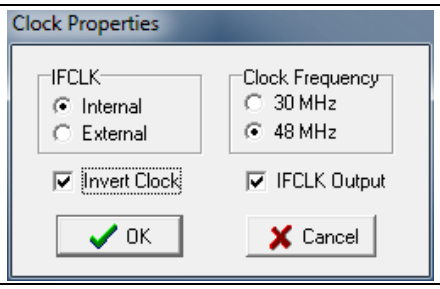


**Figure 35　IFCLK を設定**

ブロック図は **Figure 36** のようになります。これで、クロックは「nClk 48 Mhz」 (「n」がクロック反転を示す) とラベル付けられています。

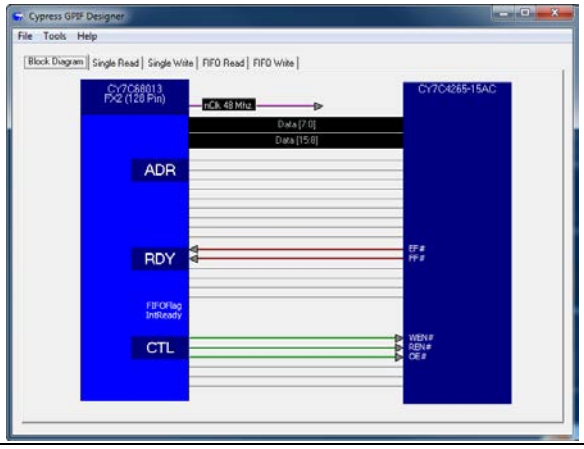


**Figure 36　FIFO インターフェースを設定**

## 9.1　シングル ワード書き込み波形

書き込み波形は FX2LP OUT エンドポイント FIFO から外部 FIFO にデータを移動するように設計されます。「FIFO Read」タブを右クリックしてその名前を「Unused」に変更します。「FIFO Write」タブに対しても同様です。顔面が **Figure 37** のようになります。

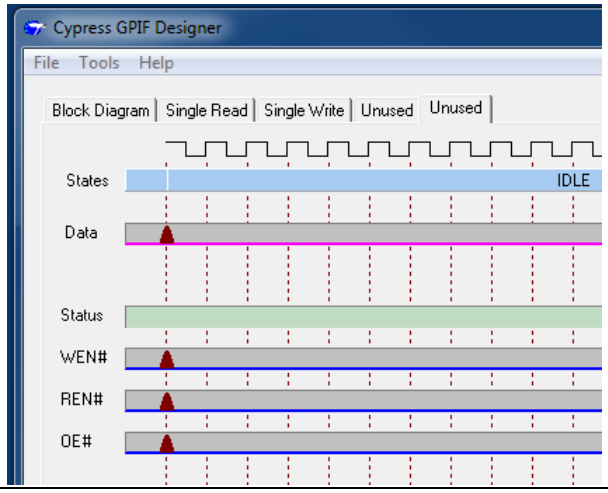


**Figure 37　波形の画面**

「**Tools** > **Map Waveforms** to WFSELECT」を選択します (**Figure 38**)。

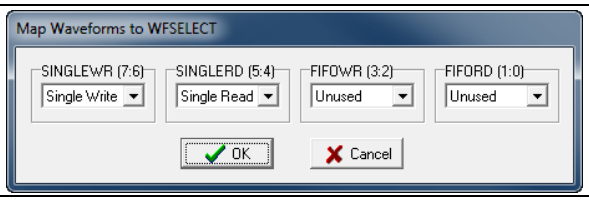


**Figure 38　波形のマッピング ダイアログ**

「Single Write」波形が「SINGLEWR」に、「Single Read」波形が「SINGLERD」にマッピングするようにしてください。これにより、波形を開始する 8051 CPU レジスタと GPIF で名付けられた波形がペアになります。

*Note:*　*波形のマッピングにより、同じ転送タイプに設定された 1 つ以上の波形組を定義することができます。例えば、Single Write に 2 つの異なる波形タイプを使用できます。*

書き込み波形は FX2LP OUT エンドポイントから外部 FIFO にデータを移動するバイト転送を管理します。FIFO 書き込み波形を構成するために、まず **CY7C4265 データシート**に記載されている CY7C4265 の FIFO 書き込みサイクル タイミングを確認してください。**Figure 39** は、最下部に GPIF ステートを追加した書き込みサイクルのタイミング図を示します。

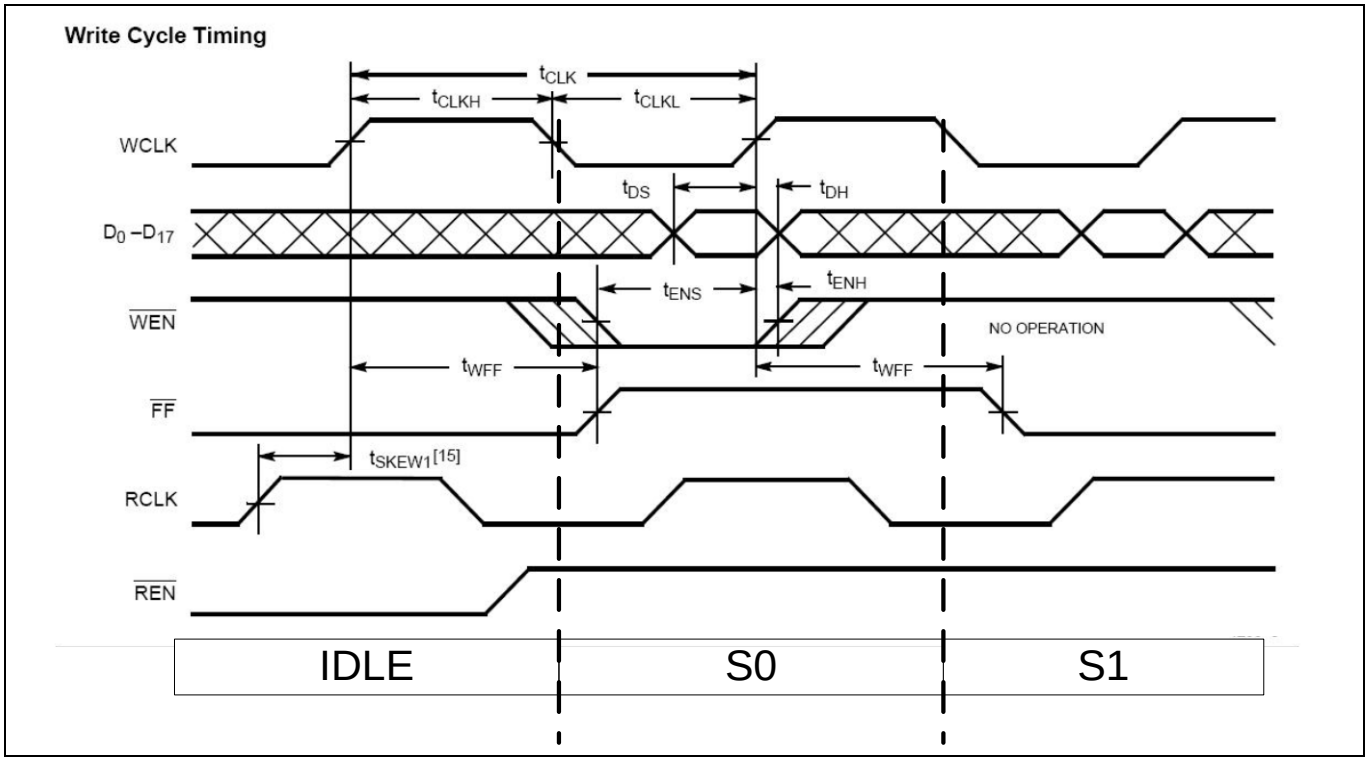


**Figure 39　書き込みサイクルのタイミング図**

書き込みサイクルのタイミング図から **Figure 40** のステート図を構成します。

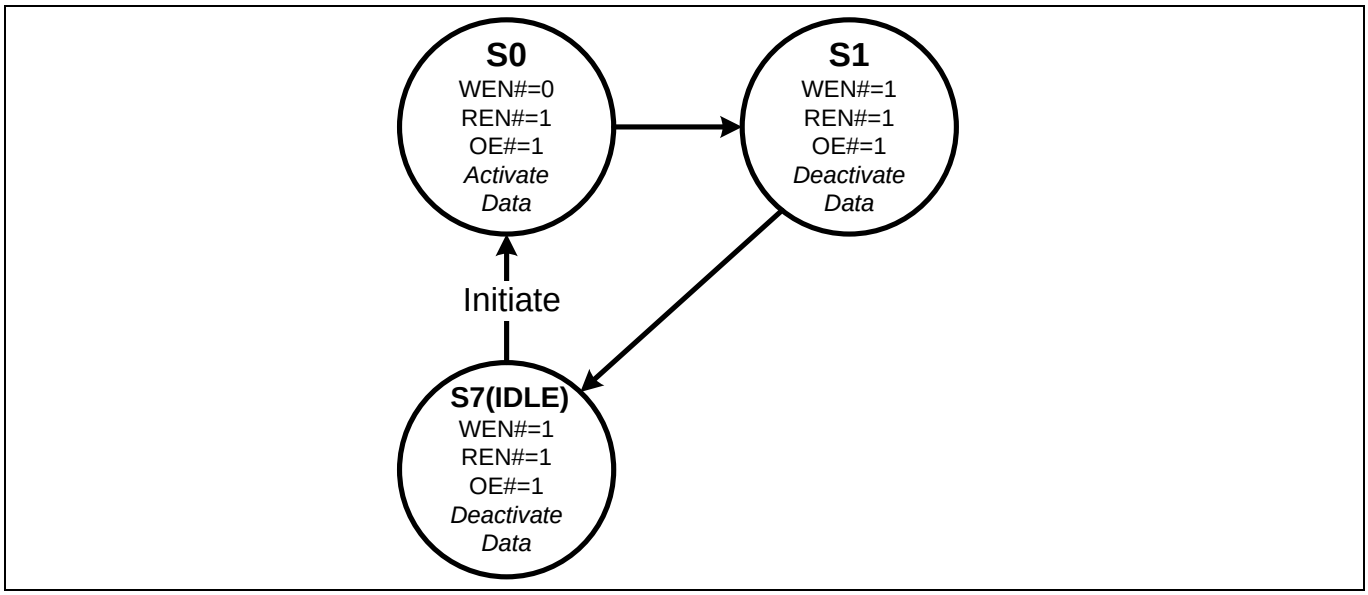


**Figure 40　シングル ワード FIFO 書き込みのステート図**

- シングル書き込み波形では、WEN#を論理 LOW にし、データ バスを駆動することでデータを外部 FIFO に書き込みます。
- S1 では、WEN#は非アクティブ (HIGH) になり、GPIF データ バスは駆動を停止 (フロート) します。S1 は、IDLE に無条件分岐を強制的に行って波形を終了する決定点のステートです。IDLE ステートでは、動作が発生しません。

### GPIF の相互接続を設計

- 書き込み波形が開始されるたびに、GPIF エンジンは S0、S1 の順番で巡回して S7 (IDLE) で停止します。

シングル書き込み波形を完了するために、以下の手順を行います。

1. **Single Write** 波形のタブをクリックします。
2. 左の境界線から 1 クロック サイクル後に WEN#トレースをクリックします。すると、アクション ポイントが置かれ、WEN#波形が作成されます。ステート 0 (s0) は自動的に生成され、1 IFCLK サイクル (20.83ns) 間持続します。
3. GPIF が外部 FIFO に書き込んでいるため、REN#と OE#は、外部 FIFO がそのデータ バスを駆動しないように波形の間ずっと HIGH にアサートする必要があります。これを保証するために、OE#と REN#トレース上のアクション ポイントを右クリックして **High (1)**を選択します。波形が **Figure 41** のようになります。

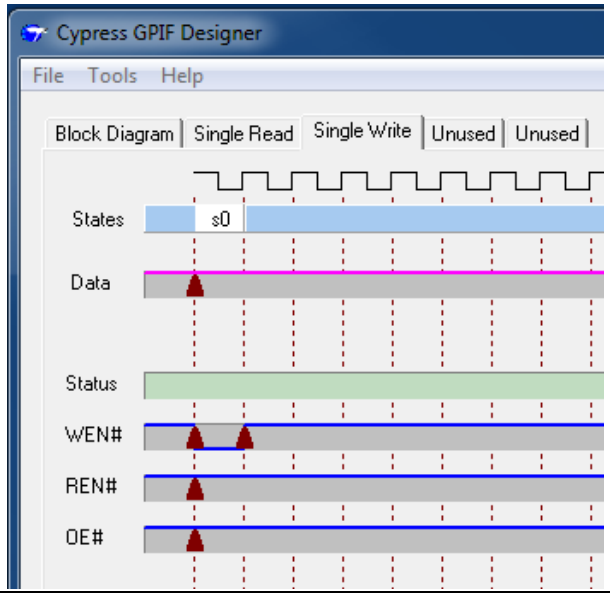


**Figure 41　1 クロックの間 WEN#パルスが LOW**

4. データ バスは s0 で駆動されます。このためには、データ アクション ポイントを右クリックして **Activate Data** を選択します。

   データ バスは 1 クロック サイクルの間だけ駆動する必要があります。1 クロック サイクル後にデータの駆動を停止するために、Data バンドで最初のアクション ポイントから 1 クロック後に別のアクション ポイントを置きます。これで Data バンドが s0 の間だけ HIGH になることに注意してください。波形が **Figure 42** のようになります。

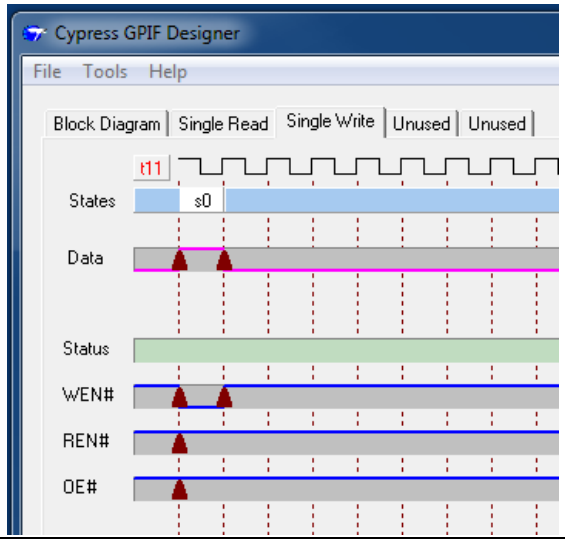


**Figure 42　S0 中に 1 クロックの間データを駆動**

5. **Figure 43**IDLE ステートに戻る分岐を実装するために、決定点のステート S1 を作成します。S0 の右の境界線で Status バンドをクリックして、図 43 のように無条件分岐を設定します。

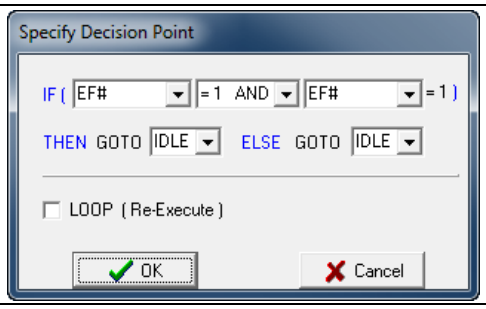


**Figure 43　IDLE への無条件分岐**

これで、シングル ワード書き込み波形が **Figure 44** のようになります。

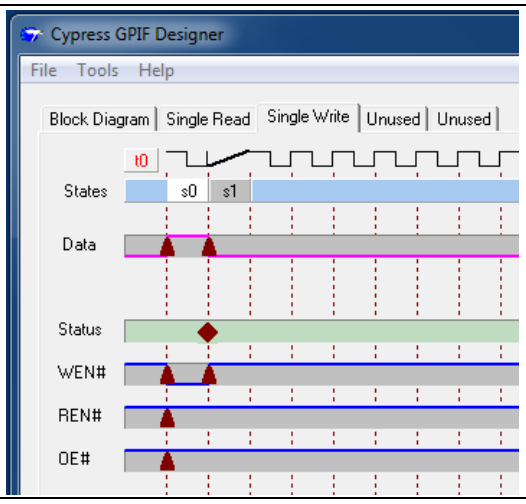


**Figure 44　シングル ワード書き込み波形**

## 9.2 シングル ワード書き込み波形

読み出し波形は外部 FIFO から FX2LP IN エンドポイント FIFO へのバイト転送を管理します。書き込み波形と同様に、GPIF 波形は外部 FIFO のタイミング要件を満たす必要があります。まず、外部 FIFO の読み出しサイクル タイミングを確認して、シングル ワード読み出しのステート マシンを作成します。**Figure 45** は、最下部に GPIF ステートを追加した CY7C4265 の読み出しタイミング図を示します。

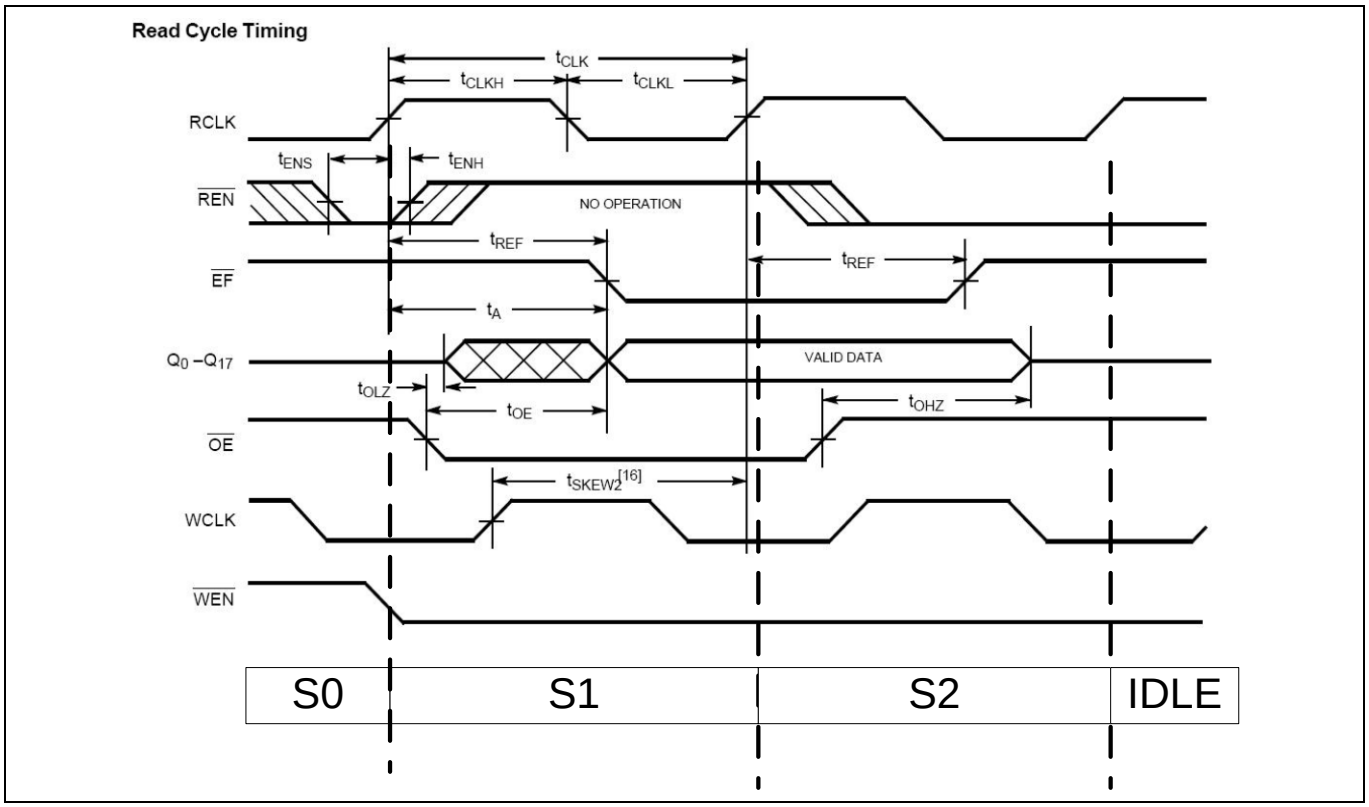


**Figure 45 読み出しサイクルのタイミング図**

タイミング情報から **Figure 46** のバイト読み出しトランザクション用のステート マシンを構成できます。

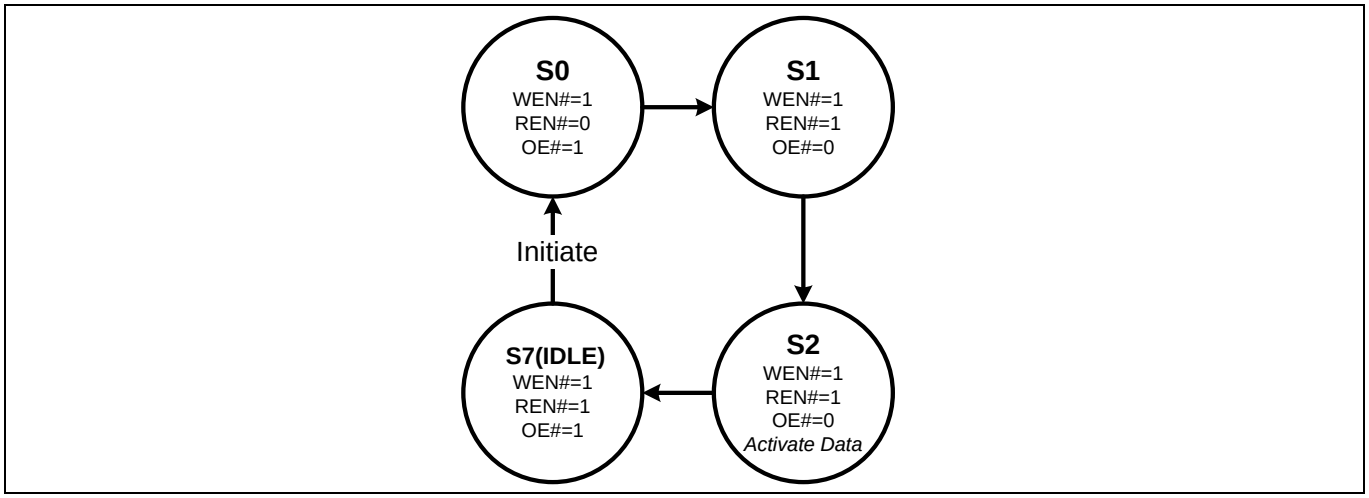


**Figure 46 バイト読み出しのステート図**

## GPIF の相互接続を設計

- シングル読み出し波形では、REN#は S0 で 1 IFCLK サイクルの間論理 LOW になります。これは、データシートに記載されている OE#アサート前の外部 FIFO の $t_{ENS}$ セットアップ時間の理由です。それから 1 クロック後に REN#をデアサートすることで、FIFO が読み出し動作用にもう 1 ワード増加しないようにします。
- その後、S1 は OE#をアサートして S2 に遷移します。
- 分岐する必要があるため、S2 は決定点のステートとなります。S1 の始まりではデータがまだ外部 FIFO に存在していないため、GPIF は S2 の始まりでデータをサンプリングします。
- シングル読み出し波形が開始されるたびに、GPIF エンジンは S0、S1、S2、S7 の順番で巡回します。

シングル読み出し波形を作成するために、以下の手順を行います。

1. **Single Read** 波形のタブをクリックします。
2. REN#トレース上の最初のクロックを右クリックして **Low (0)**を選択します。左の境界線から 1 クロック サイクル後に REN#トレースをクリックします。すると、アクション点が置かれ、REN#パルスが作成されます。ステート 0 (s0) は自動的に生成され、1 IFCLK サイクル (20.83ns) 間持続します。従って、REN#は 20.83ns 間アサートされます (**Figure 47**)。

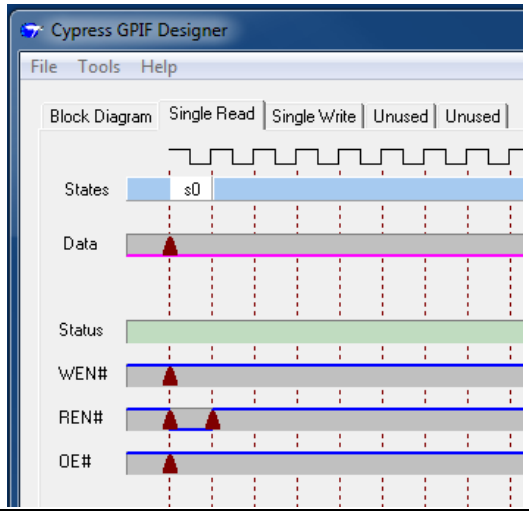


**Figure 47　REN#が 1 クロックの間アサート**

3. OE#バンドで、S0 の右の境界線にアクション点を置き、それから 1 クロック後にもう 1 つのアクション点を置きます。すると、OE#が S0 の後にデアサートされ、1 IFCLK サイクル (20.83ns) 間持続するステート S1 が作成されます。

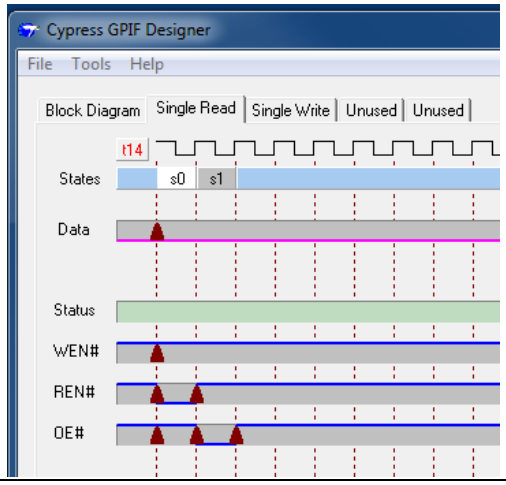


**Figure 48　OE#が 1 クロックの間アサート**

4. Status バンドで s1 の終わりに対応したクロックをクリックして決定点 (DP) のステートを追加します。すると、S2 が作成され、「Specify Decision Point」ダイアログ ボックスが表示されます。決定点を **Figure 49** のように設定して IDLE ステートに無条件に分岐します。波形が **Figure 50** のようになります。

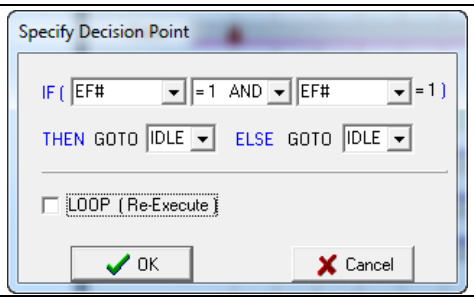


**Figure 49　IDLE への無条件分岐**

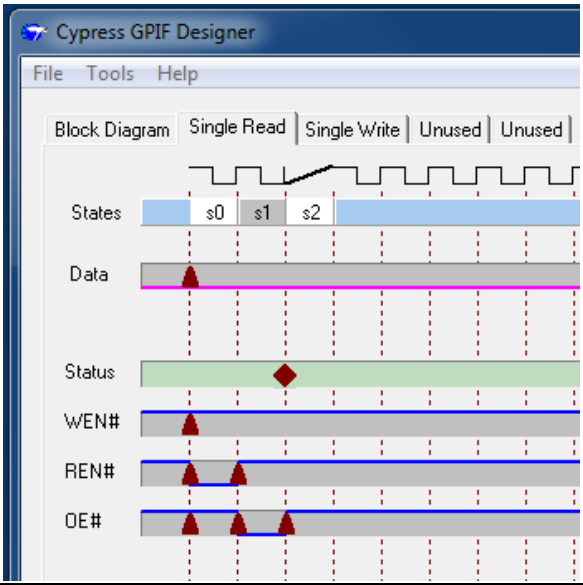

**Figure 50　ステート s2 を追加**

5. 最後に、もう 1 つの調整を行わなければなりません。データは S2 の間にサンプリングする (読み出す) 必要があります。データをサンプリングするために、Data バンドで S2 の始まりと終わりにアクション点を置きます。これが「Single Read」タブであるため、HIGH レベルが Data バンドの制御よりはむしろサンプリングに対応します。最終のシングル ワードの読み出し波形は **Figure 51** のようになります。
6. 通常、GPIF Designer プロジェクト ファイル (*.gpf) および生成された C ファイルを Keil プロジェクト フォルダに保存する必要があります。この場合、それらは本アプリケーション ノートのコードの一部として Keil プロジェクト フォルダに置いてあります。

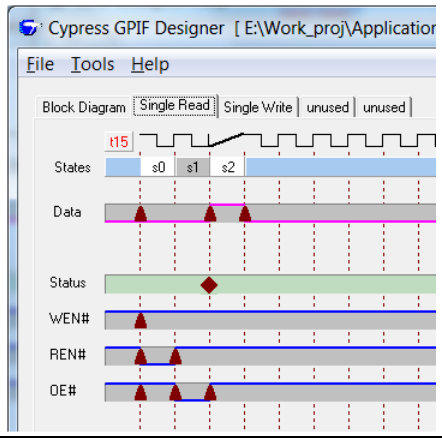


**Figure 51　最終のシングル ワードの読み出し波形**

## 9.3　GPIF シングル トランザクション用のファームウェア プログラミング

シングル ワードのトランザクション波形を GPIF Designer に実装した後、次の手順としては、USB ファームウェアを GPIF Designer の出力に組み込みます。これは、外部 FIFO への書き込みと外部 FIFO からの読み出しを USB を介して実行することを可能にします。既存のファームウェア フレームワーク プロジェクトを開始し、GPIF 管理コードを追加します。Keil プロジェクト ファイル *FX2_to_extsyncFIFO.uv2* をダブルクリックします。ファイルが開くと、「Files」パネルが **Figure 52** のように表示されます。**Table 2** はこれらのファイルを説明します。

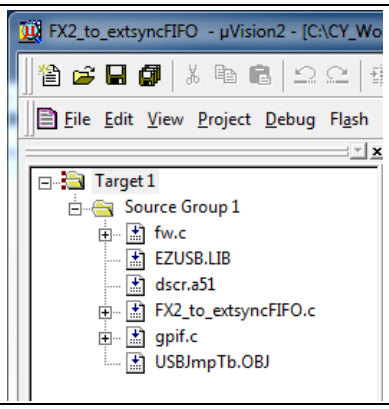


**Figure 52　「FX2_to_extsyncFIFO」プロジェクト ファイル**

**GPIF の相互接続を設計**

ファイル名 *periph.c* は *FX2_to_extsyncFIFO.c* に変更されました。このファイルでは、USB エンドポイントと GPIF 初期化コードを TD_Init()関数に追加し、GPIF 管理コードを TD_Poll()関数に追加します。

**Table 2　　プロジェクト ファイルの説明**

| ファイル | 説明 |
|---|---|
| *fw.c* | USB 要求を処理し、タスク ディスパッチャ TD_Poll() を呼び出すファームウェア フレームワーク |
| *Ezusb.lib* | サスペンド、再開、$I^2C$ 動作などを処理する関数の集合 |
| *USBJmpTb.OBJ* | USB (INT2) と GPIF/スレーブ FIFO (INT4) 割り込みソースの割り込みベクタ ジャンプ テーブル |
| *Dscr.a51* | EP2OUT と EP6IN を FX2LP デバイスの使用可能なエンドポイントとして報告する、FIFO 実施例のデバイス ディスクリプタ テーブル |
| *FX2_to_extsyncFIFO.c* (periph.c から改名) | TD_Poll()と TD_Init()を含んでいる主なユーザー アプリケーション コード。*fw.c* を更新せず、このファイルを修正する必要 |
| *Gpif.c* | シングル/FIFO GPIF トランザクション波形動作を実装する GPIF 波形ディスクリプタ テーブルを含んでいるファイル。これは GPIF Designer ツールからエクスポートされた C ファイル |

## 9.4　コード スニペット

### 9.4.1　TD_Init()

TD_INIT()は以下のことを行いします。

- CPU クロック周波数を 48MHz に切り替える (電源投入時のデフォルトクロックは 12MHz)
- EP2 をバルク OUT エンドポイント (サイズ 512 の 4 重バッファリング) に設定する
- EP6 をバルク IN エンドポイント (サイズ 512 の 4 重バッファリング) に設定する
- FIFO を手動モード、ワード幅動作に設定する
- FIFORESET レジスタを使用してエンドポイントをリセットし、USB ホスト (PC) からデータを受け入れられるように EP2 OUT エンドポイントを有効にする

```
void TD_Init(void)             // Called once at startup
{
   // set the CPU clock to 48MHz
      CPUCS = ((CPUCS & ~bmCLKSPD) | bmCLKSPD1);
      SYNCDELAY;

      EP2CFG = 0xA0;      // EP2OUT, bulk, size 512, 4x buffered
      SYNCDELAY;
      EP4CFG = 0x00;      // EP4 not valid
      SYNCDELAY;
      EP6CFG = 0xE0;      // EP6IN, bulk, size 512, 4x buffered
      SYNCDELAY;
      EP8CFG = 0x00;      // EP8 not valid
      SYNCDELAY;

      EP2FIFOCFG = 0x01; // manual mode, disable PKTEND zero length send,
word ops
      SYNCDELAY;
```

### GPIF の相互接続を設計

```
      EP6FIFOCFG = 0x01; // manual mode, disable PKTEND zero length send,
word ops
      SYNCDELAY;

      FIFORESET = 0x80;  // set NAKALL bit to NAK all transfers from host
      SYNCDELAY;
      FIFORESET = 0x02;  // reset EP2 FIFO
      SYNCDELAY;
      FIFORESET = 0x06;  // reset EP6 FIFO
      SYNCDELAY;
      FIFORESET = 0x00;  // clear NAKALL bit to resume normal operation
      SYNCDELAY;

   // out endpoints do not come up armed
   //because EP2OUT is quad buffered, write dummy byte counts four times

      EP2BCL = 0x80;     // arm EP2OUT by writing byte count w/skip.
      SYNCDELAY;
      EP2BCL = 0x80;
      SYNCDELAY;
      EP2BCL = 0x80;
      SYNCDELAY;
      EP2BCL = 0x80;
      SYNCDELAY;

  GpifInit (); // initialize GPIF registers
```

- 次に、TD_Init は *gpif.c* に存在する関数 GPIFInit() を呼び出します。
- GPIFInit()は GPIF 波形ディスクリプタ テーブルを内蔵メモリにロードし、他の GPIF レジスタを設定します。
- どの時点においても、4 つの波形だけをロードすることができます。4 本以上の波形が物理インターフェースの動作を記述するのに必要となる場合、別の 4 本の波形の組を手動でロードしなければなりません。
- IFCONFIG レジスタは物理インターフェースを定義する GpifInit()内で設定されます。
- TD_Init()は以下のコード スニペットに示すように PA2 (RS#) にパルスを発生させることで外部 FIFO をリセットします。これにより、データ操作を開始する前に外部 FIFO が初期化されたことを保証します。

```
// reset the external FIFO

  OEA |= 0x04;     // turn on PA2 as output pin
  IOA |= 0x04;     // pull PA2 high initially
  IOA &= 0xFB;     // bring PA2 low
  EZUSB_Delay (1); // keep PA2 low for ~1ms, more than enough time
  IOA |= 0x04;     // bring PA2 high
```

- USB ベンダー コマンド 0xB2 は以下のコードで定義します。これにより、ホスト PC がベンダー コマンドを発行することで外部 FIFO をリセットすることが可能になります。

```
BOOL DR_VendorCmnd(void)
{
switch (SETUPDAT[1])
```

```
  {
case VX_B2:
    {
// reset the external FIFO

      OEA |= 0x04;     // turn on PA2 as output pin
      IOA |= 0x04;     // pull PA2 high initially
      IOA &= 0xFB;     // bring PA2 low
      EZUSB_Delay (1); // keep PA2 low for ~1ms, more than enough time
      IOA |= 0x04;     // bring PA2 high

      *EP0BUF = VX_B2;
         EP0BCH = 0;
         EP0BCL = 1;          // Arm endpoint with # bytes to transfer
         EP0CS |= bmHSNAK;    // Acknowledge handshake phase of device request
break;
    }
```

### 9.4.2 GPIF シングル ワードの書き込みトランザクションをトリガー

- 8051 は XGPIFSGLDATH、XGPIFSGLDATLX、XGPIFSGLDATLNOX レジスタにアクセスすることで、シングル ワード読み出し／シングル ワード書き込みの GPIF 波形をトリガーします。これはデータ転送を開始します。
- GPIF シングル ワードの書き込みトランザクションをトリガーするために、以下のように XGPIFSGLDATH と XGPIFSGLDATLX に書き込みます:

```
XGPIFSGLDATH = <word_value>>> 8;
XGPIFSGLDATLX = <word_value> ;  // trigger GPIF Single Word Write transaction
```

- これは転送されるワード値の MSB と LSB を設定し、XGPIFSGLDATLX レジスタへの書き込みはシングル ワードの書き込みトランザクションをトリガーします。
- この例では、これは GPIF_SingleWordWrite() 関数内で行われます。この関数はワード値を入力引数として受け入れ、GPIF シングル ワードの書き込みトランザクションをトリガーします。

```
void GPIF_SingleWordWrite( WORD gdata )
{
while( !( GPIFTRIG & 0x80 ) )  // poll GPIFTRIG.7 Done bit
  {
    ;
  }

// using registers in XDATA space
  XGPIFSGLDATH = gdata;
  XGPIFSGLDATLX = gdata >> 8;         // trigger GPIF Single Word Write
transaction
}
```

- この関数は GPIFTRIG.7 ビットをポーリングすることで、トランザクションを起動する前に GPIF が IDLE ステートにあるかどうかを確認します (GPIF ステート マシンが IDLE ステートにある場合、このビットがセットされます)。どの GPIF トランザクションも起動する前に GPIF は IDLE ステートであることが必要です。

**GPIF の相互接続を設計**

- エンドポイント バッファが FIFO として構成されてからの、シングル トランザクション レジスタへのアクセス手順に注意してください。この手順により、エンドポイント バッファの最初のバイトが FD[7:0]に書き出され、2 番目のバイトが FD[15:8] (リトル エンディアン形式) に書き出されることを保証します。

### 9.4.3 GPIF シングル読み出しトランザクションをトリガー

- GPIF シングル ワード読み出しトランザクションをトリガーするために、XGPIFSGLDATX レジスタからのダミー読み出しを行います。この読み出しでは、データが転送されず、GPIF 波形のみが開始されます。GPIF の DONE ビットをテストした後、8051 は XGPIFSGLDATH と XGPIFSGLDATLNOX レジスタ内のワード値を読み出します。
- この例では、これは GPIF_SingleWordRead() 関数内で行われます。この関数はデスティネーション変数用のワード ポインタを引数として受け入れ、GPIF シングル ワードの読み出しトランザクションを行います。

```
void GPIF_SingleWordRead( WORD xdata *gdata )
{
static BYTE g_data = 0x00;      // dummy variable

while( !( GPIFTRIG & 0x80 ) )  // poll GPIFTRIG.7 Done bit
  {
     ;
  }

// using register in XDATA space
  g_data = XGPIFSGLDATLX;         // dummy read to trigger GPIF
// Single Word Read transaction

while( !( GPIFTRIG & 0x80 ) )  // poll GPIFTRIG.7 Done bit
  {
     ;
  }

// using register(s) in XDATA space, retrieve word just read from ext. FIFO
  *gdata = ( ( WORD )XGPIFSGLDATLNOX << 8 ) | ( WORD )XGPIFSGLDATH;
}
```

- この関数はまず、XGPIFSGLDATLX からダミー読み出しを行って GPIF シングル ワードの読み出しトランザクションをトリガーする前に GPIF が IDLE ステートであることを確認します。その後、GPIF がトランザクションを完了するのを待ってから、ワード値を格納しているレジスタを読み出します。

### 9.4.4 TD_Poll()

- 主なアプリケーション コードは、デバイスの動作中に繰り返して呼び出される TD_Poll() 関数内にあります。
- この関数により、GPIF_SingleWordWrite() と GPIF_SingleWordRead() 関数が呼び出されます。
- GPIF_SingleWordWrite() はデータを EP2OUT から外部 FIFO に送信し、GPIF_SingleWordRead() はデータを外部 FIFO から EP6IN に読み出します。
- USB OUT 転送を処理するコードは以下のようです。

```
if(!(EP2468STAT & bmEP2EMPTY) && (EXTFIFONOTFULL))
  {
```

### GPIF の相互接続を設計

```
// if host sent data to EP2OUT AND external FIFO is not full,

    Tcount = (EP2BCH << 8) + EP2BCL; // load transaction count with EP2
byte count
    Tcount /= 2;                     // divide by 2 for word wide
transaction
    Source = (WORD *)(&EP2FIFOBUF);
for( i = 0x0000; i < Tcount; i++ )
    {
// transfer data from EP2OUT buffer to external FIFO
      GPIF_SingleWordWrite (*Source);
      Source++;
    }
    EP2BCL = 0x80; // re-arm EP2OUT
  }
```

- EP2 空フラグは EP2468STAT レジスタで確認され、EP2OUT に USB ホストからのデータがあるかどうかを判断します。
- また、外部 FIFO からの FF#フラグは GPIFREADYSTAT レジスタにアクセスして確認されます。これにより、外部 FIFO には EP2OUT から外部 FIFO へ転送されるデータ用の空き容量があることを保証します。8051 は、GPIFREADYSTAT レジスタにアクセスして GPIF RDY 信号ステートを確認します。EXTFIFONOTFULL は、bmBIT1 と AND された GPIFREADYSTAT のマクロです。
- EP2OUT エンドポイントにデータがあり、外部 FIFO が満杯になっていない場合、ワード変数 Tcount はカウント値で初期化されます。ホストから EP2OUT に転送されるバイト数は EP2BCH/L レジスタにアクセスして確認します。各 GPIF シングル ワード書き込みトランザクションが外部 FIFO に 1 ワードを送信するため、トランザクション数は実際にエンドポイント バッファに格納されているバイト数の半分となります。
- 次に、「for」ループ GPIF_SingleWordWrite 関数を「Tcount」回呼び出し、エンドポイント バッファ EP2 の値を 1 つずつ読み出して外部 FIFO に 1 度に 1 データ ワードを送信します。各ループは 1 つの GPIF シングル書き込みトランザクションをトリガーして、外部 FIFO に 1 回に 1 データ ワードを送信します。
- 「Tcount」のデータ ワードを送信した後、ホストからの USB データ パケットが受け入れられるように EP2 エンドポイントを再び有効にします。FX2LP は新しいパケットを受け入れられるまで自動的にすべての OUT パケットを NAK します。このため、ホストは OUT 転送を実行しようとし続けます。
- USB IN 転送を処理するコードは以下のようです。

```
if(in_enable) // if IN transfers are enabled,
  {
if(!(EP2468STAT & bmEP6FULL) && (EXTFIFONOTEMPTY))
    {
// if EP6IN is not full AND there is data in the external FIFO,

      Destination = (WORD *)(&EP6FIFOBUF);
for( i = 0x0000; i < Tcount; i++ )
      {
// transfer data from external FIFO to EP6IN buffer
        GPIF_SingleWordRead (Destination);
        Destination++;
      }
        Tcount *= 2;          // multiply by 2 to obtain byte count value
      EP6BCH = MSB(Tcount);
```

### GPIF の相互接続を設計

```
      SYNCDELAY;
      EP6BCL = LSB(Tcount); // arm EP6IN to send data to the host
      SYNCDELAY;
    }
  }
```

- in_enable フラグが TRUE の場合、EP6IN エンドポイント バッファが満杯になっていない、かつ外部 FIFO が空でないことを保証するために確認します (EXTFIFONOTEMPTY は GPIFREADYSTAT と bmBIT0 のマクロです)。
- EP6IN エンドポイント バッファが満杯になっていなく、外部 FIFO が空でない場合、「for」ループは GPIF_SingleWordRead() 関数を呼び出します。これは、1 つの GPIF ワード読み出しをトリガーし、EP6IN FIFO 用に設定されたデスティネーション アドレスに結果を格納します。ループが繰り返されるたびに、デスティネーション アドレスがインクリメントされ、EP6IN FIFO が外部 FIFO から読み出されたワードで満杯になります。
- EP6IN FIFO が外部 FIFO データで満杯になった後、8051 はデータをホスト PC に転送するために EP6IN エンドポイントを有効にします。有効にされる前に、EP6 へのすべてのホスト IN 要求は自動的に FX2LP USB ロジックによって NAK されます。8051 は、ホスト IN 転送で転送するバイト数を示すバイト カウントを書き込むことで IN エンドポイントを準備します。各 GPIF シングル ワード読み出しトランザクションは外部 FIFO から完全な 1 ワード (2 バイト) を受信するため、ホストに送信されるバイト数は GPIF トランザクション数の 2 倍となります。
- IN 転送は、「in_enable」フラグに適切な値を割り当てることで有効または無効にします。以下のコードに示すように、この例で 2 つのベンダー コマンドを定義します: IN 転送を有効にするものと IN 転送を無効にするもの。

```
case VX_B3: // enable IN transfers
      {
        in_enable = TRUE;
      *EP0BUF = VX_B3;
        EP0BCH = 0;
        EP0BCL = 1;
        EP0CS |= bmHSNAK;
      break;
    }
      case VX_B4: // disable IN transfers
      {
        in_enable = FALSE;
      *EP0BUF = VX_B4;
        EP0BCH = 0;
        EP0BCL = 1;
        EP0CS |= bmHSNAK;
      break;
    }
```

- IN ベンダー コマンド 0xB3 は、in_enable を TRUE にセットすることで IN 転送を有効にするように定義されています。
- IN ベンダー コマンド 0xB4 は、in_enable を FALSE にセットすることで IN 転送を無効にするように定義されています。
- in_enable の初期設定値は FALSE です。
- in_enable フラグを使うことで、読み取りと書き込みを独立してテストすることができます。常に有効になっている場合、IN 転送コードは OUT 転送コードの直後に処理されます。コードを 1 行ずつ進

めることで、デバッグ処理用のロジック アナライザを使用して各読み出し／書き込み動作を取り込むことができます。

## 9.5 サンプル GPIF シングル トランザクションの実行

### 9.5.1 外部 FIFO なし

外部 FIFO を内蔵した FX2LP 開発基板アドオンをビルドしない場合でも、オシロスコープで GPIF の*書き込み*転送を観察することができます。これは、データを PC から FX2LP チップにパイプを介して転送する USB コードおよび外部 FIFO にデータを出力する GPIF 波形を検証します。

1. FX2LP 開発基板の P2～11 をプローブして CTL0=WEN#を観察し、P1～19 をプローブして FIFO データバス D0 を観察します。
2. 上記のように USB コントロール センターを起動します。FX2LP 開発基板を PC ポートに差し込みます。すると、基板が **Figure 21** のように USB ローダ デバイスとしてエニュメレートします。そうでばい場合は、**Figure 21** になるために手順に従います。
3. **Program FX2** > **RAM** を選択し、*FX2_to_extsyncFIFO.hex* ファイルに移動してそれをロードします。
4. Bulkloop デバイスを拡張して **Bulk out endpoint (0x02)** を選択します。
5. **Figure 53** に示すように数字を **Data to send (Hex)**テキスト ボックスに入力します。
6. **Transfer Data-OUT** ボタンをクリックします。

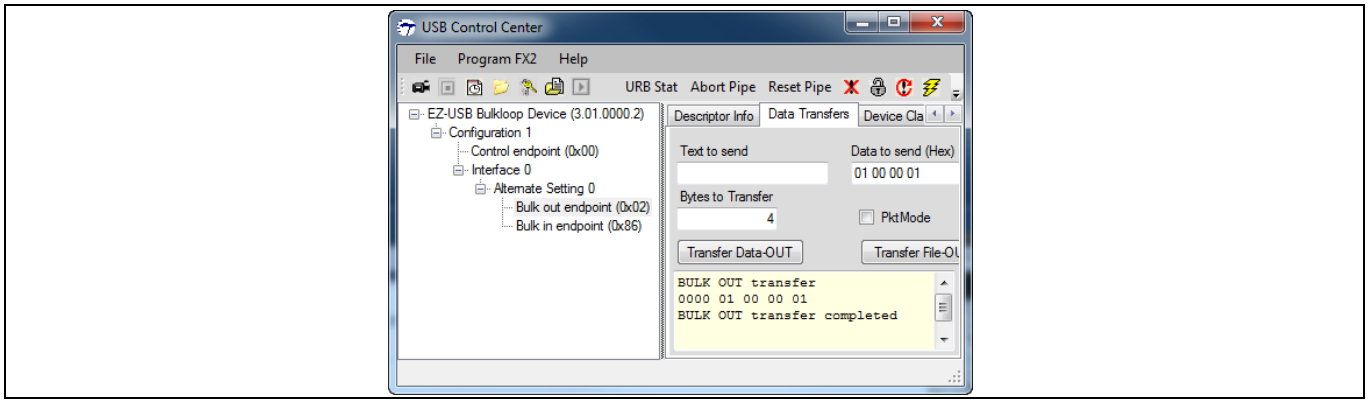


**Figure 53 USB コントロール センターはデータを転送**

7. 右下隅のテキスト ウィンドウは転送を確認し、スコープは WEN#パルスのネガティブ エッジでトリガーします (**Figure 54**)。

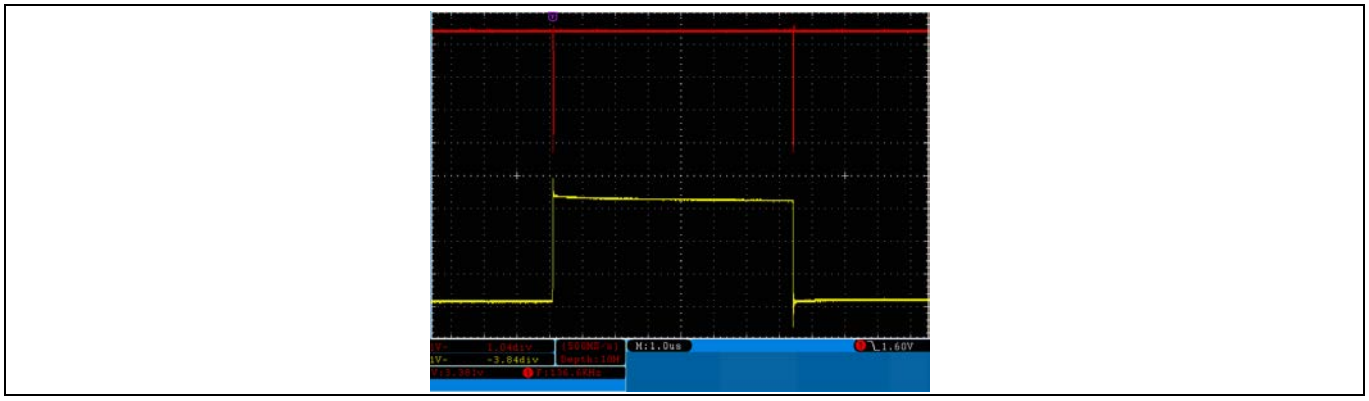

**Figure 54 最上部: WEN#、最下部: FD[0]**

**GPIF の相互接続を設計**

最初の FIFO 書き込み転送は FD[0]が HIGH であり、2 番目のものは FD[0]が LOW であることに注意してください。これにより、16 ビット ワードが L-H の順番で出力されることを保証します: 最初のワードは 0001、2 番目のワードは 0100 です。

### 9.5.2 外部 FIFO あり

外部 FIFO 基板のファームウェアをビルドして基板を差し込んだ後、ループバックをテストするために、まず前節で説明したようにデータに対し USBOUT 転送を行います。次に、同じバイト数で Data-IN の転送を行います。このテストでは、512 データ バイトを持っている *512_count.hex* を使用します。

1. ツリー図で **Bulk out endpoint (0x02)**を選択します。**Data Transfers** タブをクリックします。**Transfer File-OUT** ボタンを押し、Keil プロジェクト フォルダ (FX2LP Source code and GPIF project files\Firmware\FX2_to_extsync FIFO GPIF Single Transactions) で *512_count.hex* を選択します。**Open** をクリックします。すると、512 バイトが外部 FIFO に送信されます。
2. IN 転送を使用して 512 バイトを外部 FIFO から読み戻すために、USB ベンダー要求を使って *in_enable* フラグを TRUE にセットする必要があります。ツリー図で **Control endpoint (0x00)**を選択し、**Req code** フィールドに「0xB3」を入力します。**Req Type** を「Vendor」、**Direction** を「In」、**Bytes to Transfer** を「1」にセットします。**Transfer Data** をクリックします。
3. ツリー図で **Bulk in endpoint (0x86)**を選択します。**Bytes to Transfer** 内のバイト数が 512 であることを確認します。**Transfer Data-IN** ボタンを押します。これで、同じ 512 データ バイトが FIFO が読み戻されるはずです。

## 9.6 シングル トランザクション用のロジック アナライザ波形

本節は、GPIF Designer で定義された、波形用の GPIF エンジンによって生成されたタイミングを示します。

### 9.6.1 シングル ワード書き込み波形

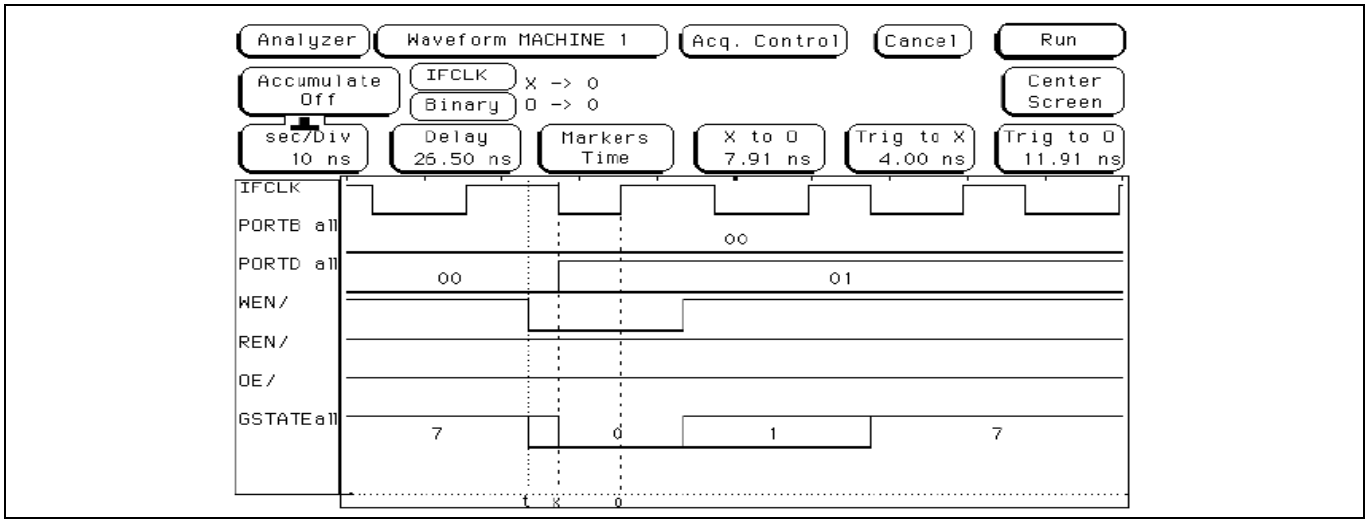


**Figure 55 シングル ワード書き込み波形**

Figure 55 は、GPIF Designer で定義された、シングル書き込み波形用の GPIF エンジンによって生成されたタイミング信号を示します。GSTATE [2:0]を含むすべての信号はここに表示されています。GSTATE [2:0]は、シングル書き込みトランザクションを実行する時に経る GPIF エンジンのステートを示します。

**GPIF の相互接続を設計**

**デバッグのヒント:**

- ロジック アナライザ ヘッダに GSTATE 信号を引き出すことで、GPIF Designer 波形と物理インターフェースで生成された信号がマッチするかを確認することができます。これは、ステート遷移が正しいかどうかを検証できるため、デバッグ処理プロセスも支援します。
- S0 はバスにデータを配置し (PORTB は FD[7:0]、PORTD は FD[15:8])、(外部 FIFO の WEN#ラインに接続された) CTL0 をアサートします。すると、16 ビットのデータ値が外部 FIFO に書き込まれます。
- 外部 FIFO の最小データ セットアップ時間は 4ns (CY7C4265 データシートを参照) であるため、IFCLK の立ち上がりエッジまでのデータ セットアップ時間が十分に提供されていることに注意してください。
- S1 は、無条件に IDLE ステートへ分岐してトランザクションを終了する決定点のステートです。
- 無条件分岐を使わないと、GPIF エンジンは IDLE ステート (S7) に達するまで、残りのステート (S2～S6) を順番に遷移してきます。
- バルク OUT 転送で書き出すすべてのワードで、S0、S1、S7 を巡回する GPIF エンジン サイクルが表示されます。
- 波形を捕捉するためには、CTL0 の立ち下がりエッジでロジック アナライザをトリガーします。
- 4ns のサンプリング レートで、**Figure 55** の波形に示したものと同じ分解能が得られます。

## 9.6.2 シングル ワード読み出し波形

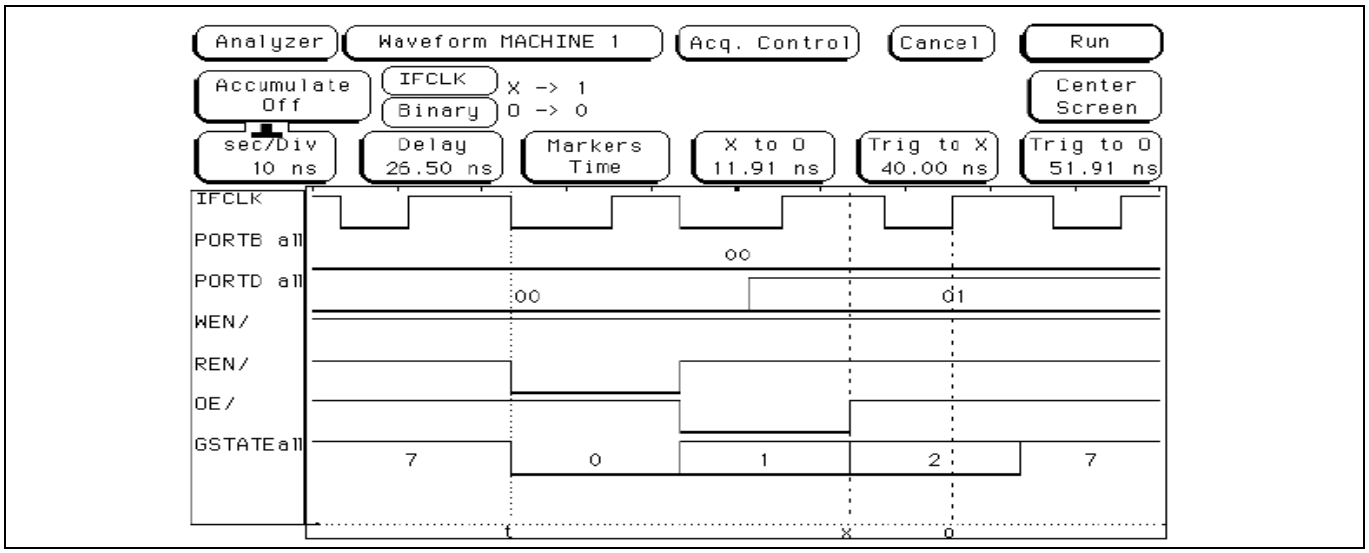


**Figure 56 シングル ワード読み出し波形**

**Figure 56** は、GPIF Designer で定義された、シングル ワード読み出し波形用の GPIF エンジンによって生成されたタイミングを示します。GSTATE [2:0]を含むすべての信号はここに表示されています。GSTATE [2:0]は、シングル読み出しトランザクションを実行する時に経る GPIF エンジンのステートを示します。

**デバッグのヒント:**

- S0 は (外部 FIFO の REN#ラインに接続された) CTL1 をアサートし、S1 は (外部 FIFO の OE#ラインに接続された) CTL2 をアサートし、S2 はデータ バス (PORTB は FD [7:0]、PORTD は FD[15:8]) をサンプリングします。すると、16 ビットのデータ値が外部 FIFO から読み出されます。
- GPIF の最小データ セットアップ時間が 9.2ns (FX2LP データシートを参照) であるため、IFCLK の立ち上がりエッジまでのデータ セットアップ時間が十分に提供されていることに注意してください。
- S2 は、無条件に IDLE ステートへ分岐してトランザクションを終了する決定点のステートです。

### GPIF の相互接続を設計

- 無条件分岐を使わないと、GPIF エンジンは IDLE ステート (S7) に達するまで、残りのステート (S3～S6) を順番に遷移してきます。
- バルク IN 転送で外部 FIFO から読み出すすべてのワードで、S0、S1、S2、S7 を巡回する GPIF エンジン サイクルが表示されます。
- 波形を捕捉するためには、CTL1 の立ち下がりエッジでロジック アナライザをトリガーします。
- 4ns のサンプリング レートで、**Figure 56** の波形に示したものと同じ分解能が得られます。

# 10 関連文書:

- 「**Getting Started with FX2LP**」: この文書はユーザーが FX2LP に詳しくなる手助けになります。
- 「**FX2LP Technical Reference Manual**」: この文書は FX2LP のテクニカル ガイドとなります。「General Programmable Interface (GPIF)」章は GPIF を詳しく説明しています。
- 「GPIF Designer Utility User Guide」: この文書にアクセスするためには、**GPIF Designer** からユーティリティをダウンロードします。インストール後、**Help** > **This Tool** を選択します。
- 「EZ-USB FX1-EZ-USB FX2LP Development Kit Quick Start Guide.pdf」と「EZ-USB Development Kit User Guide.pdf」: これらの文書は CY3684 キットの使用方法を説明しており、(DVK のインストール後に) C:\Cypress\USB\CY3684_EZ-USB_FX2LP_DVK\1.0\Documentation にあります。
- 「**Endpoint FIFO Architecture of EZ-USB FX1/FX2LP**」: このアプリケーション ノートは FX1/FX2LP 内のデータ フローを理解する手助けになります。

## 10.1 他の GPIF 例

他の GPIF の例は以下のサイプレスのアプリケーション ノートにあります。

- 「**AN57322: Interfacing SRAM with FX2LP over GPIF**」: このアプリケーション ノートは GPIF 8 ビット非同期 (クロックなし) インターフェースを使用して CY7C1399B SRAM を FX2LP に接続する方法を説明します。また、この文書は FX2LP を他の SRAM に接続するガイドとなります。
- 「**AN63787: EZ-USB FX2LP GPIF and Slave FIFO Configuration Examples Using 8-bit Asynchronous Interface**: このアプリケーション ノートは、8 ビット非同期パラレル インターフェースを実装するように EZ-USB FX2LP の GPIF とスレーブ FIFO を手動モードと自動モードの両方で設定する方法を説明します。デザインは、2 個の相互接続された FX2LP 開発基板 (1 個が GPIF マスターとして、もう 1 個が GPIF スレーブとして動作) を使用してテストします。
- 「**AN4051: FX2LP GPIF Flow State Feature for UDMA**: このアプリケーション ノートは GPIF の「フロー ステート」機能を紹介します。この機能により、GPIF 機能は ATAPI UDMA を処理できるように拡張されます。

## 10.2 リファレンス デザイン

- 「**CY4611B - USB 2.0 USB to ATA Reference Design**」: 一般的な FX2LP の応用の 1 つは USB 大容量記憶装置です。FX2LP GPIF により、接続されたドライブとの切れ目のない接続を実現できます。サイプレスは、FX2LP を使った大容量記憶装置の参考設計を提供しています。

## 10.3 データシート

- 「**EZ-USB FX2LP USB Microcontroller High-Speed USB Peripheral Controller**」
- **CY7C4265 データシート**

# 11 まとめ

本アプリケーション ノートは EZ-USB FX2LP GPIF への紹介資料となります。本書は、汎用プログラマブル インターフェースを作成する手順を説明し、GPIF Designer の主な特長を示す例を提供します。

# 改訂履歴

| Document version | Date of release | Description of changes |
|---|---|---|
| ** | 2012-01-20 | Japanese translation of 001-66806 |
| *A | 2015-03-11 | No Change; Completing sunset review |
| *B | 2018-06-11 | これは英語版 001-66806 Rev. *G を翻訳した日本語版 Rev. *B です。 |
| *C | 2021-06-02 | テンプレートの変更を実施。<br>これは英語版 001-66806 Rev. *H を翻訳した日本語版 Rev. *C です。 |

**Trademarks**
All referenced product or service names and trademarks are the property of their respective owners.

**Edition 2021-06-02**
**Published by**
**Infineon Technologies AG**
**81726 Munich, Germany**



**Do you have a question about this document?**
**Go to www.cypress.com/support**

**Document reference**
**001-75653 Rev. *C**

**重要事項**
本文書に記載された情報は、いかなる場合も、条件または特性の保証とみなされるものではありません（「品質の保証」）。本文に記された一切の事例、手引き、もしくは一般的価値、および／または本製品の用途に関する一切の情報に関し、インフィニオンテクノロジーズ（以下、「インフィニオン」）はここに、第三者の知的所有権の不侵害の保証を含むがこれに限らず、あらゆる種類の一切の保証および責任を否定いたします。

さらに、本文書に記載された一切の情報は、お客様の用途におけるお客様の製品およびインフィニオン製品の一切の使用に関し、本文書に記載された義務ならびに一切の関連する法的要件、規範、および基準をお客様が遵守することを条件としています。

本文書に含まれるデータは、技術的訓練を受けた従業員のみを対象としています。本製品の対象用途への適合性、およびこれら用途に関連して本文書に記載された製品情報の完全性についての評価は、お客様の技術部門の責任にて実施してください。

本製品、技術、納品条件、および価格についての詳しい情報は、インフィニオンの最寄りの営業所までお問い合わせください（**www.infineon.com**）。

**警告事項**
技術的要件に伴い、製品には危険物質が含まれる可能性があります。当該種別の詳細については、インフィニオンの最寄りの営業所までお問い合わせください。

インフィニオンの正式代表者が署名した書面を通じ、インフィニオンによる明示の承認が存在する場合を除き、インフィニオンの製品は、当該製品の障害またはその使用に関する一切の結果が、合理的に人的傷害を招く恐れのある一切の用途に使用することはできないこと予めご了承ください。