EZ-USB™ FX3/FX3S ブートオプション

About this document

Scope and purpose

本アプリケーション ノート (AN76405) は、EZ-USB™ FX3 ペリフェラル コントローラーに適用される、USB, I²C, シリアル ペリフェラル インタフェース (SPI), および同期アドレス データ多重化 (ADMux) インタフェース経由のブート オプションについて説明します。本書はまた、FX3S と CX3 ペリフェラル コントローラーにも適用可能です。

Intended audience

このアプリケーションノートは、主に EZ-USB™ FX3/ FX3S/ CX3 ユーザーを対象とします。

関連製品ファミリ

CYUSB30xx

その他のサンプル コードが必要な場合は、以下を参照してください。

USB スーパースピードのサンプル コードの総合リストについては、**https://www.cypress.com/101781** にアクセスしてください。

Table of contents

About this document **1**
Table of contents **1**
1 はじめに **4**
2 詳細情報 **5**
2.1 EZ-USB™ FX3 ソフトウェア開発キット 5
2.2 GPIF II Designer 5
3 FX3 ブート オプション **7**
4 USB ブート **9**
4.1 PMODE ピン 9
4.2 特長 9
4.2.1 デフォルト シリコン ID 9
4.2.2 ブートローダ レビジョン 10
4.2.3 ReNumeration 10
4.2.4 バスパワーのアプリケーション 10
4.2.5 USB フォールバック オプション (--> USB) 10
4.2.6 VID/PID オプションの USB 10
4.2.7 USB デフォルト デバイス 10
4.2.8 USB セットアップ パケット 11
4.2.9 USB 仕様の第 9 章およびベンダー コマンド 11
4.2.10 USB ベンダー コマンド 12
4.2.11 USB ダウンロード サンプル コード 13

Table of contents

4.3 チェックサムの計算 15
4.3.1 FX3 ブートローダ メモリ割当て 15
4.3.2 レジスタ/メモリ アクセス 15
4.3.3 USB eFUSE VID/PID ブート オプション 15
4.3.4 USB OTG 15
4.3.5 ブートローダの制限 15
4.3.6 USB ウォッチドッグ タイマー 15
4.3.7 USB 一時停止/再開 16
4.3.8 USB デバイス ディスクリプタ 16
4.4 ブート イメージ フォーマット 18
4.4.1 ロング ワードのフォーマットで構成されたブート イメージ フォーマットの例 19

5 I²C EEPROM ブート 20
5.1 特長 20
5.2 EEPROM へのファームウェア イメージの保存 21
5.2.1 128KB EEPROM のアドレス指定の注意事項 22
5.3 ブート イメージ フォーマット 23
5.4 チェックサムの計算 25
5.4.1 ブート イメージの例 1 25
5.4.2 ブート イメージの例 2 25
5.4.3 チェックサム計算のサンプル コード 26

6 USB フォールバックが有効の I²C EEPROM ブート 27
6.1 特長 27
6.2 VID と PID によるブート イメージ例 27

7 SPI ブート 28
7.1 特長 28
7.2 SPI フラッシュの選択 29
7.3 ハイブリッド セクタを持つ SPI フラッシュの使用方法 29
7.4 ファームウェア イメージの SPI フラッシュ/EEPROM への保存 29
7.5 ブート イメージ フォーマット 30
7.6 チェックサムの計算 32

8 USB フォールバックが有効の SPI ブート 34
8.1 VID と PID によるブート イメージ例 34

9 同期 ADMux ブート 35
9.1.1 インタフェース信号 36
9.1.2 同期 ADMux のタイミング 37
9.1.3 USB フォールバック (-->USB) 37
9.1.4 ウォーム ブート 37
9.1.5 ウェイクアップ/スタンバイ 37
9.1.6 GPIF II API プロトコル 40
9.1.7 ファームウェア ダウンロード例 41
9.1.8 プロセッサ ポート (P-Port) レジスタ マップ 42
9.2 ブート イメージ フォーマット 50

10 ブート時の I/O のデフォルト状態 54

11 付録 A: FX3 DVK 基板 (CYUSB3KIT-001) を用いるブート手順 57
11.1 USB ブート 58
11.2 I²C ブート 62
11.3 SPI ブート 66

12 付録 B: 同期 ADMux ブートのトラブルシューティング手順 71
12.1 初期化 71

Table of contents

12.2 レジスタ読み書きのテスト 71
12.3 FIFO 読み書きのテスト 71
12.4 ファームウェア ダウンロードのテスト 73
13 付録 C: elf2img ユーティリティを使用したファームウェアイメージの生成 74
13.1 使用方法 74
13.1.1 イメージタイプ 74
13.1.2 割込みベクタロード 74
13.1.3 EEPROM 制御 75
13.1.3.1 I2C パラメーター 75
13.1.3.2 SPI パラメーター 75
関連資料 76
改訂履歴 77

1 はじめに

EZ-USB™ FX3 は次世代の USB 3.0 ペリフェラル コントローラーであり、開発者が USB 3.0 機能を広範囲のアプリケーションに追加できるようにする高集積度かつフレキシブルな機能を提供します。FX3 はいくつかのブート オプションに対応します。そのうち、USB, I^2C, SPI, 同期 ADMux インタフェース経由のブートがあります。

ブート中の FX3 I/O のデフォルト状態についても説明します。**付録 A** は **FX3 DVK** を使用して異なるブート モードをテストする段階的な手順について記述しています。

2 詳細情報

インフィニオンは、**www.infineon.com** に大量のデータを掲載しており、ユーザーがデザインに適切なデバイスを選択し、デバイスを迅速、効果的にデザインに統合する手助けをしています。

- 概要: **USB ポートフォリオ**、**USB ロードマップ**
- USB 3.0 製品セレクタ: **FX3** , **FX3S**, **CX3**, **HX3, SX3**
- アプリケーション ノート: インフィニオンは、基本レベルから高度なレベルまでの様々なトピックに触れる大量の USB アプリケーション ノートを提供しています。以下は FX3 入門用の推奨アプリケーション ノートです。
  - **AN75705** – EZ-USB™ FX3 入門
  - **AN231295** – EZ-USB™ SX3 入門
  - **AN70707** – EZ-USB™ FX3/FX3S ハードウェア設計ガイドラインおよび回路図チェックリスト
  - **AN65974** – EZ-USB™ FX3 スレーブ FIFO インタフェースを使った設計
  - **AN75779** – How to implement an image sensor interface with EZ-USB™ FX3 in a USB Video Class (UVC) framework
  - **AN86947** – EZ-USB™ FX3 で USB 3.0 のスループットを最適化
  - **AN84868** – EZ-USB™ FX3 を使用した USB 経由での FPGA コンフィギュレーション
  - **AN68829** – Slave FIFO interface for EZ-USB™ FX3: 5-bit address mode
  - **AN76348** – Differences in implementation of EZ-USB™ FX2LP and EZ-USB™ FX3 applications
  - **AN89661** – USB RAID 1 disk design using EZ-USB™ FX3S
- サンプル コード:
  - **USB ハイスピード**
  - **USB フルスピード**
  - **USB スーパースピード**
- テクニカル リファレンス マニュアル (TRM):
  - **EZ-USB™ FX3 Technical Reference Manual**
- 開発キット:
  - **CYUSB3KIT-003, EZ-USB™ FX3 SuperSpeed Explorer Kit**
  - **CYUSB3KIT-001, EZ-USB™ FX3 Development Kit**
  - モデル: **IBIS**

2.1 EZ-USB™ FX3 ソフトウェア開発キット

インフィニオンは、FX3 がスーパースピード USB をいかなる組込みアプリケーションにも簡単に統合するための完全なソフトウェアとファームウェア スタックを提供しています。**ソフトウェア開発キット** (SDK) は、アプリケーション開発を加速させるツール、ドライバーやアプリケーション例を含みます。

2.2 GPIF II Designer

GPIF II Designer は、設計者が EZ-USB™ FX3 USB 3.0 デバイス コントローラーの GPIF II インタフェースを設定できるようにするグラフィカル ソフトウェアです。

ユーザーは、このツールでインフィニオンが提供する 5 つのインタフェースから 1 つを選択するか、あるいは独自の GPIF II インタフェースを最初から作成するか、選択可能です。インフィニオンは非同期と同期スレーブ FIFO、非同期と同期 SRAM などの業界標準インタフェースを提供します。あらかじめ定義されたこれらのインタフェースのいずれかを、既にシステム内に備えている場合、設計者はバス幅

詳細情報

(x8、x16、x32)、エンディアン、クロック設定など一連の標準パラメーターからインタフェースを選択してコンパイルすることができます。このツールはインタフェースをカスタマイズしたいユーザーに対して、能率化された 3 段階の GPIF インタフェース開発プロセスを提供します。ユーザーはまずピン コンフィギュレーションおよび標準パラメーターを選択できます。次に、コンフィギュレーション可能な操作で仮想ステート マシンを設計します。最後に、出力タイミングが期待どおりになるかを確かめられます。3 段階のプロセスを完了させると、インタフェースをコンパイルして、FX3 と統合します。

3 FX3 ブート オプション

FX3 は、マスク ROM に内蔵されたブートローダを統合しています。ブートローダの機能は、USB, I²C, SPI, GPIF II (例えば、同期 ADMux) などのさまざまなインタフェースから FX3 ファームウェア イメージをダウンロードすることです。

FX3 ブートローダは、FX3 の 3 つの PMODE 入力ピンを用いて使用するブート オプションを選択します。**Figure 1** に、本アプリケーション ノートで説明されるブート オプションを示します。**Table 1** に、これらのブート オプションおよび必要な PMODE ピン設定を一覧にします。

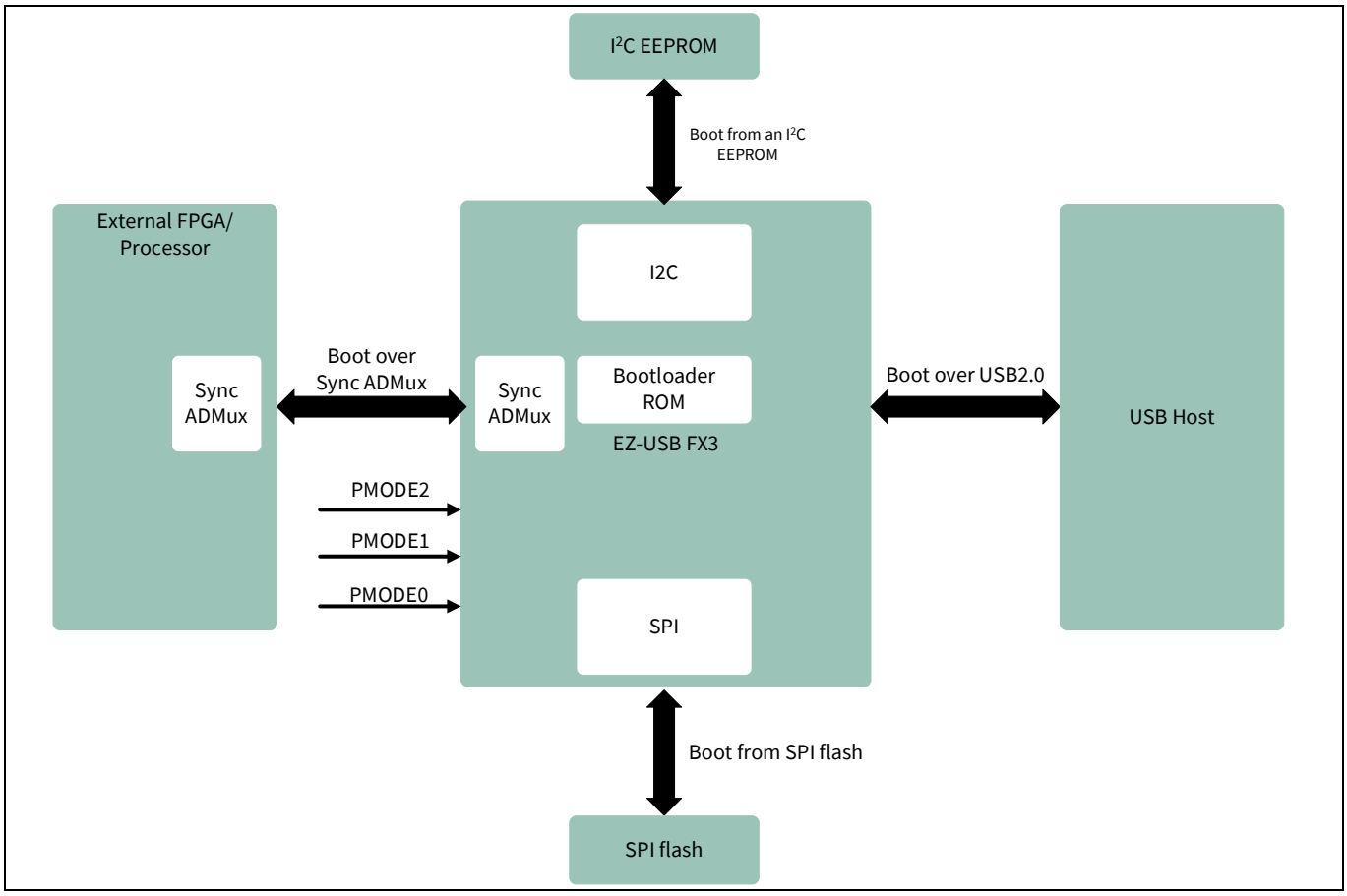

Figure 1 FX3 ブート オプション

Table 1 FX3 のブート オプション*

PMODE[2:0]ピン			ブート オプション	USB フォールバック
PMODE[2]	PMODE[1]	PMODE[0]		
Z	0	0	同期 ADMux (16 ビット)	無
Z	1	1	USB ブート	有
1	Z	Z	I²C	無
Z	1	Z	I²C → USB	有
0	Z	1	SPI → USB	有

他の組合せは予約済み

FX3 ブート オプション

Note: ** Z = 開放。PMODE ピンは、未接続のままにする、または FPGA I/O に接続してからその I/O を FPGA の入力に設定することで開放にします。*

次の節では、FX3 によりサポートされるブート オプションについて説明します。

- **USB ブート**: FX3 ファームウェア イメージは USB ホストから FX3 システム RAM にダウンロードされます。
- **I²C EEPROM ブート**: FX3 ファームウェア イメージは外部 I²C EEPROM にプログラムされ、リセットの時に、FX3 ブートローダは I²C を介してファームウェアをダウンロードします。
- **SPI ブート**: FX3 ファームウェア イメージは外部 SPI フラッシュまたは SPI EEPROM にプログラムされ、リセットの時に、FX3 ブートローダは SPI を介してファームウェアをダウンロードします。
- **同期 ADMux ブート**: FX3 ファームウェア イメージは、外部プロセッサまたは FX3 GPIF II インタフェースに接続された FPGA からダウンロードされます。

4 USB ブート

Figure 2 に、USB を介してブートする場合の FX3 システム図を示します。

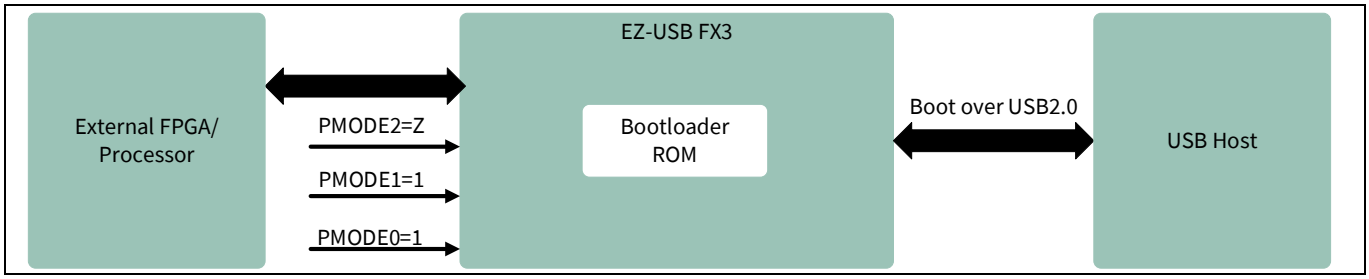

Figure 2 FX3 システム図

4.1 PMODE ピン

USB ブートでは、Table 2 に示すように PMODE[2:0]ピンの状態は Z11 です。

Table 2 USB ブートの PMODE ピン

PMODE[2]	PMODE[1]	PMODE[0]
Z	1	1

Note: Z = 開放

4.2 特長

外部 USB ホストは USB 2.0 モードで FX3 にファームウェア イメージをダウンロードできます。FX3 は、バスパワーの USB ベンダー クラス デバイスとしてエニュメレートします。

USB ブート モードでの FX3 の状態は以下のとおりです。

- USB 3.0 (スーパースピード) シグナリングは無効です。
- USB 2.0 (ハイスピード/フルスピード) は有効です。
- FX3 はファームウェア ダウンロード/アップロード用にベンダー コマンド A0h を使用します。このベンダー コマンドはブートローダで実装されます。

4.2.1 デフォルト シリコン ID

デフォルトでは、FX3 の ROM 空間にはデフォルトのサイプレス セミコンダクタ社の VID = 04B4h および PID = 00F3h が格納されています。eFUSE[1] VID/PID がプログラムされることを除いては、この VID/PID はデフォルトの USB エニュメレーションに使用されます。デフォルトのサイプレス ID 値は開発目的のためのみに使用されます。最終製品にはユーザー独自の VID/PID を使用しなければいけません。VID は USB-IF で登録することで得られます。

1 eFUSE は、チップ内の特定の回路の再プログラミングを可能にする技術です。eFUSE のプログラミングの詳細については、サイプレスの担当者にお問い合わせください。

4.2.2 ブートローダ レビジョン

Table 3 に示すように、ブートローダ レビジョンは ROM 空間内にアドレス FFFF_0020h で格納されています。

Table 3 ブートローダ レビジョン

マイナー レビジョン	FFFF_0020h
メジャー レビジョン	FFFF_0021h
予約済みバイト	FFFF_0022h、FFFF_0023h

4.2.3 ReNumeration

ReNumeration 機能は FX3 でサポートされ、ファームウェアによって制御されます。

USB ホストに初めて差し込まれると、FX3 は自分のデフォルトの USB ディスクリプタで自動的にエニュメレートします。ファームウェアがダウンロードされると、FX3 はダウンロードした USB ディスクリプタの情報で定義されるデバイスとして再びエニュメレートします。この 2 段階プロセスは「ReNumeration」と呼ばれています。

4.2.4 バスパワーのアプリケーション

ブートローダはバスパワー モードでエニュメレートします。FX3 は USB 2.0 規格で規定されているように、100mA 未満の電流でエニュメレートすることでバスパワーのデザインに完全に対応できます。

4.2.5 USB フォールバック オプション (--> USB)

USB フォールバックが有効であり、他のオプションでブートする場合、FX3 は本節で説明した同じ USB ブート モードに逆戻りします。動作中に他のクロック ソースがオンにされているため、動作電流は USB ブート モードより少し高くなります。

4.2.6 VID/PID オプションの USB

ブートローダは、以下のものに保存される新規 VID/PID でのブートをサポートします。

- I^2C EEPROM: 本アプリケーション ノートの **I2C EEPROM ブート**を参照してください。
- SPI EEPROM: 本アプリケーション ノートの **SPI ブート**を参照してください。
- eFUSE (VID/PID): カスタム eFUSE VID/PID プログラミングについては、インフィニオンの販売担当者にお問い合わせください。

4.2.7 USB デフォルト デバイス

FX3 ブートローダは、1 つのインタフェース (インタフェース 0) を含むシングル USB コンフィギュレーションおよび「0」の代替設定で構成されます。このモードでは、エンドポイント 0 のみは有効です。他のすべてのエンドポイントはオフになります。

4.2.8 USB セットアップ パケット

FX3 ブートローダは、**Table 4** で定義された 8 バイト データ構造体を含む SETUP パケットを復号します。

Table 4 セットアップ パケット

バイト	フィールド	説明
0	*bmRequestType*	要求タイプ: ビット 7: 方向 ビット 6〜0: 受信側
1	*bRequest*	このバイトはファームウェア ダウンロード/アップロード ベンダー コマンドとして A0h になる
2〜3	*wValue*	16 ビットの値 (リトルエンディアン フォーマット)
4〜5	*wIndex*	16 ビットの値 (リトルエンディアン フォーマット)
6〜7	*wLength*	バイト数

*Note: ビット単位の説明については、**USB 2.0 仕様**を参照してください。*

4.2.9 USB 仕様の第 9 章およびベンダー コマンド

FX3 ブートローダは **Table 5** でのコマンドを処理します。

Table 5 USB コマンド

bRequest	説明
00	GetStatus: デバイス、エンドポイント、およびインタフェース
01	ClearFeature: デバイス、エンドポイント
02	予約済み: STALL を返す
03	SetFeature: デバイス、エンドポイント
04	予約済み: STALL を返す
05	SetAddress: FX3 ハードウェアで処理
06	GetDescriptor: ROM 内のデバイスのディスクリプタ
07	予約済み: STALL を返す
08h	GetConfiguration: 内部値を返す
09h	SetConfiguration: 内部値を設定
0Ah	GetInterface:内部値を返す
0Bh	SetInterface: 内部値を設定
0Ch	予約済み: STALL を返す
20h〜9Fh	予約済み: STALL を返す
A0h	ベンダー コマンド: ファームウェア アップロード/ダウンロード等
A1h〜FFh	予約済み: STALL を返す

4.2.10 USB ベンダー コマンド

ブートローダは、ファームウェア ダウンロードおよびアップロード用に A0h ベンダー コマンドをサポートします。コマンドのフィールドを **Table 6** および **Table 7** に示します。

Table 6 ファームウェア ダウンロード用のコマンド フィールド

バイト	フィールド	値	説明
0	***BmRequestType***	40h	要求タイプ: ビット 7: 方向 ビット 6〜0: 受信側
1	***bRequest***	A0h	このバイトはファームウェア ダウンロード/アップロード ベンダー コマンドとして A0h になる
2〜3	***WValue***	AddrL (LSB)	16 ビットの値 (リトルエンディアン フォーマット)
4〜5	***WIndex***	AddrH (MSB)	16 ビットの値 (リトルエンディアン フォーマット)
6〜7	***wLength***	カウント	バイト数

Table 7 ファームウェア アップロード用のコマンド フィールド

バイト	フィールド	値	説明
0	***BmRequestType***	C0h	要求タイプ: ビット 7: 方向 ビット 6〜0: 受信側
1	***bRequest***	A0h	このバイトはファームウェア ダウンロード/アップロード ベンダー コマンドとして A0h になる
2〜3	***WValue***	AddrL (LSB)	16 ビットの値 (リトルエンディアン フォーマット)
4〜5	***WIndex***	AddrH (MSB)	16 ビットの値 (リトルエンディアン フォーマット)
6〜7	***wLength***	カウント	バイト数

Table 8 実行のプログラム エントリへの転送用コマンド フィールド

バイト	フィールド	値	説明
0	***bmRequestType***	40h	要求タイプ: ビット 7: 方向 ビット 6〜0: 受信側
1	***bRequest***	A0h	このバイトはファームウェア ダウンロード/アップロード ベンダー コマンドとして A0h になる
2〜3	***wValue***	AddrL (LSB)	32 ビットのプログラム エントリ
4〜5	***wIndex***	AddrH (MSB)	32 ビットのプログラム エントリ >>16
6〜7	***wLength***	0	このフィールドは「0」でなければならない

エントリへの実行転送コマンドでは、ブートローダはすべての割込みを無効にし、USB 接続を切断します。以下は 3 つのベンダー コマンド サブルーチンの例です。

例 1. 8 バイトのセットアップ パケットのベンダー コマンド書き込みデータ プロトコル

bmRequestType = 0x40

bRequest = 0xA0;

wValue = (WORD)address;

wIndex = (WORD)(address>>16);

wLength = 1 to 4K-byte max

USB ブート

このコマンドは、wLength および DATA IN 長さゼロのパケットに等しい転送長の DATA OUT パケットを送信します。

例 2. セットアップ パケットによるブートローダ レビジョン読み出し

bmRequestType = 0xC0

bRequest = 0xA0;

wValue = (WORD)0x0020;

wIndex = (WORD)0xFFFF;

wLength = 4

このコマンドは、wLength および DATA OUT 長さゼロのパケットに等しい転送長の DATA IN パケットを発行します。

例 3. 8 バイトのセットアップ パケットによるプログラム エントリへのジャンプ (**Table 8** を参照してください)

bmRequestType = 0x40

bRequest = 0xA0;

wValue = Program Entry (16-bit LSB)

wIndex = Program Entry >>16 (16-bit MSB)

wLength = 0

Note: FX3 は 32 ビット アドレッシングを使用します。アドレスは、コマンドの wValue と wIndex フィールドに書き込まれます。

4.2.11 USB ダウンロード サンプル コード

コードをダウンロードするために、アプリケーションはファームウェア イメージ ファイルを読み出し、ベンダー書き込みコマンドを使用して一度に 4K セクションに書き込む必要があります。セクションのサイズは、ブートローダ内で使用されるバッファのサイズに制限されます。

*Note: ファームウェア イメージは **Table 14** で指定されるフォーマットである必要があります。*

以下は、ファームウェア ダウンロード ルーチンを実行する方法の例です。

```
DWORD dCheckSum, dExpectedCheckSum, dAddress, i, dLen;
WORD wSignature, wLen;
DWORD dImageBuf[512*1024];
BYTE  *bBuf, rBuf[4096];

fread(&wSignature,1,2,input_file);/*fread(void *ptr, size_t size, size_t count, FILE
*stream)
                                  read signature bytes.  */
if (wSignature != 0x5943)         // check 'CY' signature byte
{
   printf("Invalid image");
   return fail;
}
```

USB ブート

```
fread(&i, 2, 1, input_file);     // skip 2 dummy bytes
dCheckSum = 0;
while (1)
{
   fread(&dLength,4,1,input_file);  // read dLength
   fread(&dAddress,4,1,input_file); // read dAddress
   if (dLength==0) break;           // done
   // read sections
   fread(dImageBuf, 4, dLength,  input_file);
   for (i=0; i<dLength; i++) dCheckSum += dImageBuf[i];
   dLength <<= 2;  // convert to Byte length
   bBuf = (BYTE*)dImageBuf;
   while (dLength > 0)
   {
       dLen = 4096;  // 4K max
       if (dLen > dLength) dLen = dLength;
       VendorCmd(0x40, 0xa0, dAddress, dLen, bBuf); // Write data
       VendorCmd(0xc0, 0xa0, dAddress, dLen, rBuf); // Read data
       // Verify data: rBuf with bBuf
       for (i=0; i<dLen; i++)
       {
          if (rBuf[i] != bBuf) { printf("Fail to verify image"); return fail; }
      }
      dLength -= dLen;
      bBuf += dLen;
      dAddress += dLen;
   }
}
// read pre-computed checksum data
fread(&dExpectedChecksum, 4, 1, input_file);
if  (dCheckSum != dExpectedCheckSum)
{
   printf("Fail to boot due to checksum error\n");
  return fail;
}
// transfer execution to Program Entry
VendorCmd(0x40, 0xa0, dAddress, 0, NULL);
```

input_file は、**Table 14** で指定されるフォーマットのファームウェア イメージ ファイルを指す FILE ポインタです。

Note: *実装例については、EZ-USB FX3 SDK に付属する USB Control Center アプリケーションのソースコードプロジェクトを参照してください。*

4.3 チェックサムの計算

USB ダウンロードでは、**USB ダウンロード サンプル コード**に示すように、ダウンロード ツールはチェックサムの計算を処理するように予想されます。

4.3.1 FX3 ブートローダ メモリ割当て

FX3 ブートローダはその変数とスタックのために 0x1000_0000～0x1000_04FF のデータ密結合メモリ (DTCM) の 1280 バイトを割り当てます。ファームウェア ダウンロード中に、このメモリ領域が初期化されない限り、すなわち未初期化のローカル変数である限り、ファームウェア アプリケーションはこの領域を使用できます。

ブートローダは 0x4000_0000～0x4000_000F の最初の 16 バイトをウォーム ブートとスタンバイ ブート用に割り当てます。これらのバイトはファームウェア アプリケーションで使用してはいけません。

ブートローダは 0x4000_23FF から約 10K バイトをその内部バッファ用に割り当てます。ファームウェア アプリケーションはこの領域を未初期化のローカル変数/バッファとして使用できます。

ブートローダは命令密結合メモリ (ITCM) を使用しません。

4.3.2 レジスタ/メモリ アクセス

FX3 ブートローダは、ROM、MMIO、SYSMEM、ITCM、および DTCM メモリ空間からの読み出しアクセスが可能です。DTCM の最初の 1280 バイトおよびシステム メモリの最初の 10K バイトを除いて、MMIO、SYSMEM、ITCM、および DTCM メモリ空間への書き込みアクセスが可能です。MMIO 空間に書き込む時、ブートローダの予想される転送長は 4 (ロング ワードと同等) であり、アドレスは 4 バイト アラインである必要があります。

4.3.3 USB eFUSE VID/PID ブート オプション

FX3 ブートローダは、eFUSE (eFUSE_USB_ID) をスキャンして USB_VID ビットがプログラムされたかどうかを確認することで、選択した VID と PID でブートできます。ビットがプログラムされた場合、ブートローダは eFUSE 値を VID と PID に使用します。

4.3.4 USB OTG

FX3 ブートローダは USB On-The-Go (OTG) プロトコルに対応せず、USB バスパワーのデバイスとして動作します。

4.3.5 ブートローダの制限

FX3 ブートローダは制限されたアドレス範囲のチェックを処理します。存在しないアドレスへのアクセスは、予期できない結果となる可能性があります。

ブートローダはプログラム エントリをチェックしません。無効なプログラム エントリは予期できない結果となる可能性があります。ブートローダは、MMIO レジスタ空間への書き込みアクセスが可能です。無効なアドレスへの書き込みアクセスは予期できない結果となる可能性があります。

4.3.6 USB ウォッチドッグ タイマー

FX3 USB ハードウェアは USB コア ハードウェアへの 32 kHz クロック入力を必要とします。外部 32kHz クロックがない場合、ブートローダはウォッチドッグ タイマーを USB コア用の内部 32 kHz クロック入力に設定します。

4.3.7 USB 一時停止/再開

USB では動作がない場合、FX3 ブートローダは一時停止モードに入ります。PC が USB 作動を再開すると、ブートローダは再開します。

4.3.8 USB デバイス ディスクリプタ

次の表は、ハイスピードおよびフルスピードの FX3 ブートローダ ディスクリプタを一覧にします。

Note: デバイス修飾子はフルスピード モードで使用できません。

Table 9 デバイス ディスクリプタ

オフセット	フィールド	値	説明
0	***bLength***	12h	ディスクリプタの長さ = 18 バイト
1	***bDescType***	01	ディスクリプタのタイプ = デバイス
2〜3	***wBCDUSB***	0200h	USB 仕様バージョン 2.0
4	***bDevClass***	00	デバイス クラス (クラス固有のプロトコルが実装されない)
5	***bDevSubClass***	00	デバイス サブクラス (クラス固有のプロトコルが実装されない)
6	***bDevProtocol***	00	デバイス プロトコル (クラス固有のプロトコルが実装されない)
7	***bMaxPktSize***	40h	Endpoint0 のパケット サイズは 64
8〜9	***wVID***	04B4h	サイプレス セミコンダクタ社の VID
10〜11	***wPID***	00F3h	FX3 シリコン チップ
12〜13	***wBCDID***	0100h	FX3 bcdID
14	***iManufacture***	01h	製造者インデックス文字列 = 01
15	***iProduct***	02h	製品インデックス文字列 = 02
16	***iSerialNum***	03h	シリアル番号インデックス文字列 = 03
17	***bNumConfig***	01h	1 つのコンフィギュレーション

Table 10 デバイス修飾子

オフセット	フィールド	値	説明
0	***bLength***	0Ah	ディスクリプタの長さ = 10 バイト
1	***bDescType***	06	ディスクリプタのタイプ = デバイス修飾子
2〜3	***wBCDUSB***	0200h	USB 仕様バージョン 2.00
4	***bDevClass***	00	デバイス クラス (クラス固有のプロトコルは実装されない)
5	***bDevSubClass***	00	デバイス サブクラス (クラス固有のプロトコルは実装されない)
6	***bDevProtocol***	00	デバイス プロトコル (クラス固有のプロトコルは実装されない)
7	***bMaxPktSize***	40h	エンドポイント 0 のパケット サイズは 64
8	***bNumConfig***	01h	1 つのコンフィギュレーション
9	***bReserved***	00h	「0」でなければならない

Table 11　コンフィギュレーション ディスクリプタ

オフセット	フィールド	値	説明
0	***bLength***	09h	ディスクリプタの長さ = 10 バイト
1	***bDescType***	02h	ディスクリプタのタイプ = コンフィギュレーション
2〜3	***wTotalLength***	0012h	全長
4	***bNumInterfaces***	01	本コンフィギュレーションでのインタフェース数
5	***bConfigValue***	01	このインタフェースを選択するために SetConfiguration 要求によって使用されたコンフィギュレーション値
6	***bConfiguration***	00	本コンフィギュレーションを説明する文字列のインデックス = 0
7	***bAttribute***	80h	属性: バスパワー、ウェイクアップなし
8	***bMaxPower***	64h	最大電流: 200mA

Table 12　インタフェース ディスクリプタ (代替設定 0)

オフセット	フィールド	値	説明
0	***bLength***	09h	ディスクリプタの長さ = 10 バイト
1	***bDescType***	04h	ディスクリプタのタイプ = インタフェース
2	***bInterfaceNum***	00h	このインタフェースのゼロベース インデックス = 0
4	***bAltSetting***	00	代替の設定値 = 0
5	***bNumEndpoints***	00	エンドポイント 0 のみ
6	***bInterfaceClass***	FFh	ベンダー コマンド クラス
7	***bInterfaceSubClass***	00h	
8	***bInterfaceProtocol***	00h	
9	***iInterface***	00h	なし

Table 13　文字列ディスクリプタ

オフセット	フィールド	値	説明
0	***bLength***	04h	このディスクリプタの長さ = 04 バイト
1	***bDescType***	03h	ディスクリプタのタイプ = 文字列
2〜3	***wLanguage***	0409h	言語 = 英語
4	***bLength***	10h	ディスクリプタの長さ = 16 バイト
5	***bDescType***	03h	ディスクリプタのタイプ = 文字列
6〜21	***wStringIdx1***	–	「Cypress」
22	***bLength***	18h	ディスクリプタの長さ = 24 バイト
23	***bDescType***	03h	ディスクリプタのタイプ = 文字列
24〜47	***wStringIdx2***	–	「WestBridge」
48	***bLength***	1Ah	ディスクリプタの長さ = 26 バイト
49	***bDescType***	03h	ディスクリプタのタイプ = 文字列
50〜75	***wStringIdx3***	–	「0000000004BE」

4.4 ブート イメージ フォーマット

USB ブートでは、ブートローダはファームウェア イメージ ファイルが **Table 14** に示すフォーマットであると想定しています。**EZ-USB™ FX3 SDK** は、USB ブートに必要なフォーマットでファームウェア イメージを生成するためのソフトウェア ユーティリティを提供します。SDK をインストールした後、*C:\Program Files\Cypress\EZ-USB FX3 SDK\1.3\util\elf2img* ディレクトリにある *elf2img* ユーティリティを参照してください。64 ビット システムでは、パスにある最初のフォルダは Program Files (x86) です。ディレクトリ パスにある *1.3* 番号は SDK のバージョン番号ですが、FX3 SDK の最新リリースによって異なります。elf2img ユーティリティの使用の詳細については、**付録 C** および**付録 A** の **Figure 15** を参照してください。

Table 14 ブート イメージ フォーマット

2 進数イメージ ヘッダ	長さ (16 ビット)	説明
wSignature	1	ASCII 表記の「CY」の初期値を持つ 2 バイトのシグネチャ
bImageCTL;	½	ビット 0 = 0: 2 進数実行ファイル；1:データ ファイル タイプ ビット 3〜1:SPI EEPROM でブートする場合は未使用 ビット 5〜4 (SPI 速度): 00: 10MHz 01: 20MHz 10: 30MHz 11: 予約済み ビット 7〜6: 予約済み、「0」にセットする必要がある
bImageType;	½	bImageType = 0xB0: チェックサムを含む普通の 2 進数ファームウェア イメージ bImageType = 0xB2: 新規 VID と PID による I^2C/SPI ブート
dLength 0	2	ロング ワード (32 ビット) での最初のセクションの長さ bImageType = 0xB2 の場合、dLength 0 は PID と VID を含む。ブートローダは残りの後続データを無視
dAddress 0	2	プログラム コードの最初のセクション アドレス *Note: 内部 ARM アドレスはバイト アドレス指定が可能で、各セクションのアドレスは 32 ビット アラインである必要がある*
dData[dLength 0]	dLength 0*2	イメージ コード/データは 32 ビット アラインである必要がある
…		他のセクション
dLength N	2	0x00000000 (最後の記録:終端セクション)
dAddress N	2	有効なプログラム エントリ (通常はスタートアップ コード、すなわち RESET ベクタ) を含む必要がある *Note: bImageCTL.bit0 = 1 の場合、ブートローダはこのプログラム エントリに実行を転送しない* *bImageCTL.bit0 = 0 の場合、ブートローダはこのプログラム エントリに実行を転送。このアドレスは ITCM 空間または SYSTEM RAM 空間にある必要がある。* *ブートローダはプログラム エントリを確認しない*

2 進数イメージ ヘッダ	長さ (16 ビット)	説明
dCheckSum	2	32 ビット符号なしのリトルエンディアン チェックサム データは最初のセクションから終端セクションまで。チェックサムは dLength、dAddress、およびイメージ ヘッダを含まない

4.4.1 ロング ワードのフォーマットで構成されたブート イメージ フォーマットの例

```
Location1: 0xB0 0x10 'Y' 'C' //CY Signature, 20 MHz, 0xB0 Image
Location2: 0x00000004        //Image length of section 1 = 4
Location3: 0x40008000        //1st section stored in SYSMEM RAM at 0x40008000
Location4: 0x12345678        //Image starts (Section1)
Location5: 0x9ABCDEF1
Location6: 0x23456789
Location7: 0xABCDEF12        //Section 1 ends
Location8: 0x00000002        //Image length of section 2 = 2
Location9: 0x40009000        //2nd section stored in SYSMEM RAM at 0x40009000
Location10: 0xDDCCBBAA       //Section 2 starts
Location11: 0x11223344
Location12: 0x00000000       //Termination of Image
Location13: 0x40008000       //Jump to 0x40008000 on FX3 System RAM
Location14: 0x6AF37AF2       //Checksum (0x12345678 + 0x9ABCDEF1 + 0x23456789 +
              0xABCDEF12+ 0xDDCCBBAA +0x11223344)
```

FX3 DVK を使用して USB ブート モードをテストする段階的な手順は、**付録 A** の **USB ブート**で記述します。

5 I²C EEPROM ブート

Figure 3 に、I²C を介してブートする場合の FX3 システム図を示します。

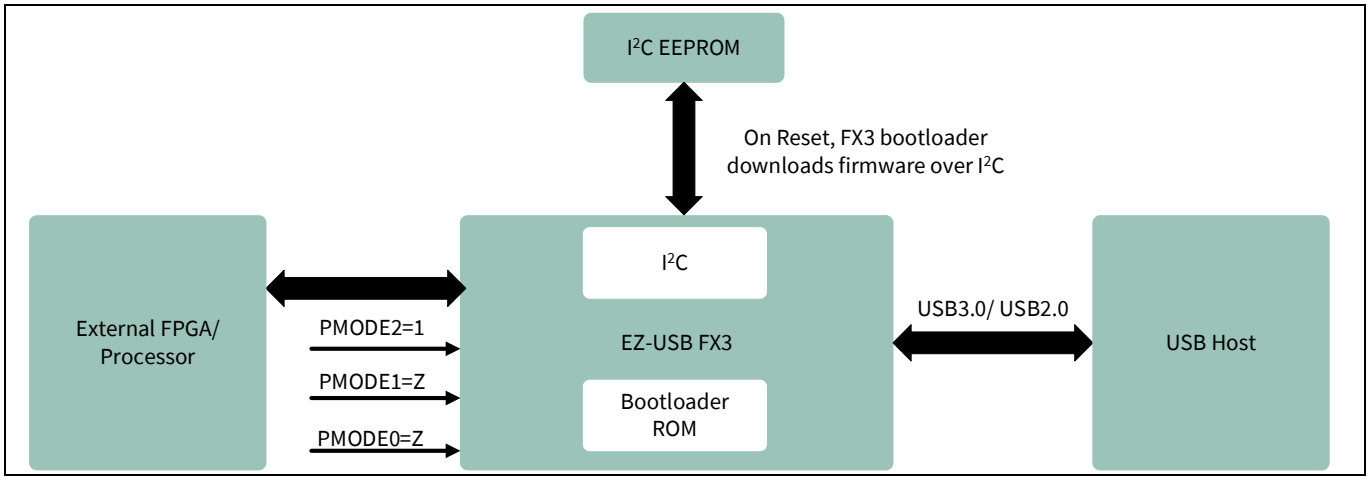

Figure 3 I²C ブートの FX3 システム図

I²C EEPROM ブートでは、**Table 15** に示すように PMODE[2:0]ピンの状態は *1ZZ* です。

Table 15 I²C ブートの PMODE ピン

PMODE[2]	PMODE[1]	PMODE[0]
1	Z	Z

FX3 I²C インタフェースのピン マッピングを **Table 16** に示します。

Table 16 I²C インタフェースのピン マッピング

EZ-USB™ FX3 のピン	I²C インタフェース
I2C_GPIO[58]	I2C_SCL
I2C_GPIO[59]	I2C_SDA

5.1 特長

- FX3 は 2 線式 I²C インタフェースを介して I²C EEPROM デバイスからブートします。
- サポートされる EEPROM[1]デバイス サイズは以下のようになります。
  - 32 キロビット (Kb)または 4 キロバイト (KB)
  - 64 Kb または 8 KB
  - 128 Kb または 16 KB
  - 256 Kb または 32 KB
  - 512 Kb または 64 KB
  - 1024 Kb または 128 KB
  - 2048 Kb または 256 KB

1 2 バイトの I²C アドレスのみはサポートされます。1 バイト アドレスは 32Kb 以下の I²C EEPROM でサポートされません。

I2C EEPROM ブート

Note: リリース バージョンで生成されたイメージ ファイルのサイズがデバッグ バージョンより小さいため、リリース モードでビルドされたファームウェア イメージを使用することを推奨します。

- ATMEL 社、Microchip 社および ST Electronics 社のデバイスはテストされました (例: M24M02, AT24C1024, など)。
- 100kHz、400kHz および 1MHz の I^2C 周波数はブート処理中にサポートされます。V_{O5} が 1.2V の場合、対応する最大動作周波数は 100kHz であることに注意してください。V_{IO5} が 1.8V、2.5V、または 3.3V の場合、対応する動作周波数は 400kHz と 1MHz です。(V_{IO5} は I^2C インタフェース用の I/O 電圧です)。
- FX3 ブートローダは、スレーブアドレス 0xA0 の I^2C EEPROM のみをサポートしています
- 同じサイズを持つ複数の I^2C EEPROM デバイスからのブートはサポートされます。I^2C EEPROM がファームウェア イメージより小さい場合、複数の I^2C EEPROM デバイスを使用する必要があります。ブートローダは、複数の I^2C EEPROM デバイスにわたりイメージをロードすることをサポートします。SuperSpeed Explorer CYUSB3KIT-003 は ST Electronics 社の 256KB EEPROM (M24M02) を使用します。 ブートローダは、128KB より小さい I^2C EEPROM デバイスを最大 8 個までサポートでき、128KB の I^2C EEPROM デバイスを最大 4 個までサポートできます。
- I^2C EEPROM に保存できるファームウェア イメージは 1 つのみです。重複イメージは許可されません。
- ブートローダは、FX3 のマルチマスターI^2C 機能をサポートしません。したがって、FX3 I^2C ブート プロセス中、他の I^2C マスターは I^2C バス上で作動してはいけません。

5.2 EEPROM へのファームウェア イメージの保存

FX3 ブートローダは、外部シリアル I^2C EEPROM デバイスのためにマスターI^2C インタフェースを提供します。シリアル I^2C EEPROM はアプリケーション固有のコードとデータを格納するために使用できます。**Figure 4** に標準 I^2C EEPROM のピン接続を示します。

I^2C EEPROM インタフェースは、シリアル クロック ライン (SCL) とシリアル データ ライン (SDA) という 2 本のアクティブ ラインを含みます。

EEPROM にファームウェア イメージを書き込む中に、書き込み保護 (WP) ピンは LOW にプルする必要があります。

A0、A1、および A2 ピンはアドレス ラインです。これらはスレーブ デバイス アドレスを 000 から 111 まで設定します。これにより、8 個の同じサイズの I^2C EEPROM をアドレス指定することが可能です。これらのラインは、要求されるアドレスによって HIGH または LOW にプルされます。

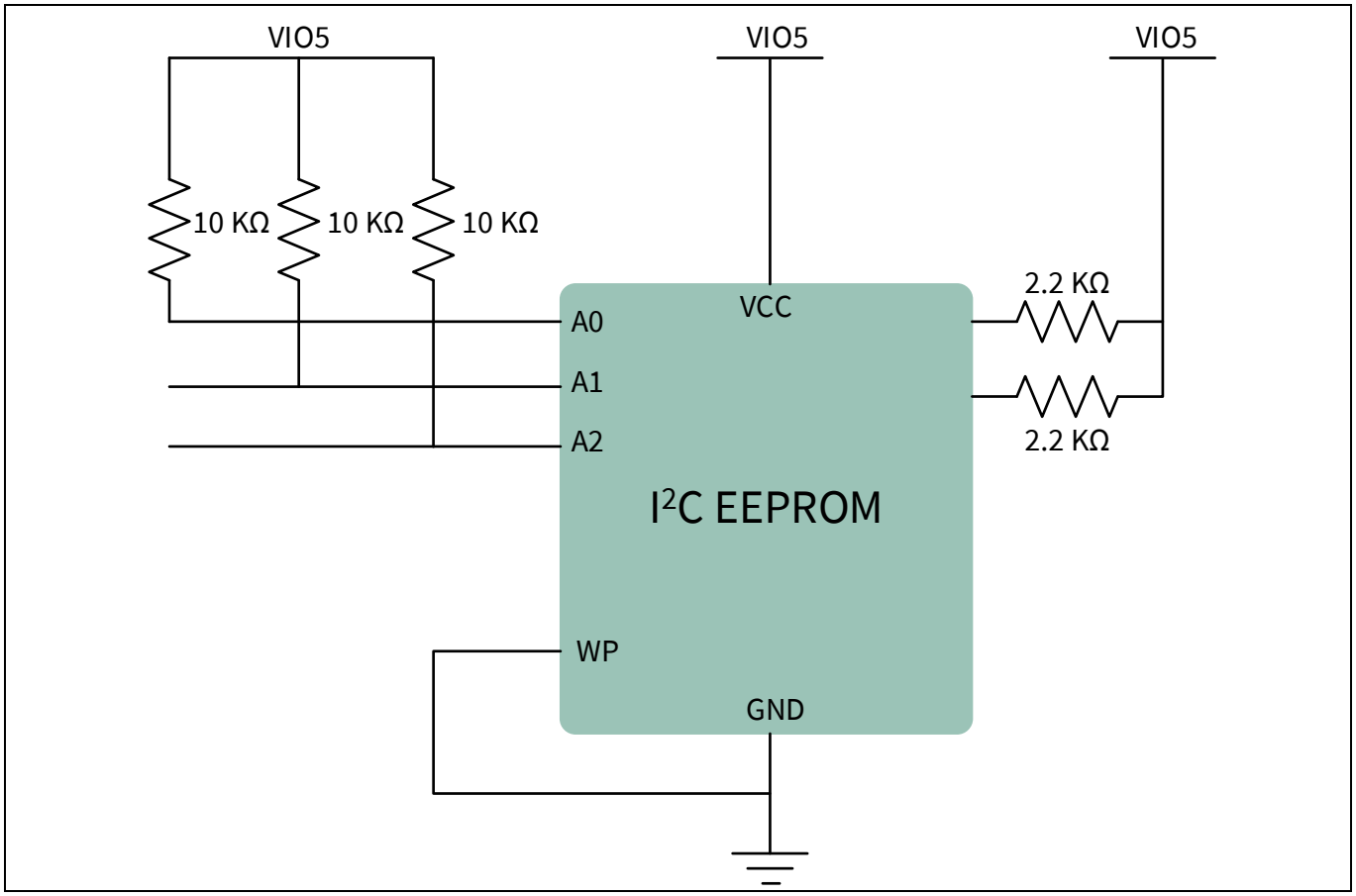

Figure 4　　標準 I²C EEPROM のピン接続

5.2.1　128KB EEPROM のアドレス指定の注意事項

128KB の I²C EEPROM の場合、このアドレス指定方式は全ての EEPROM の標準ではありません。例えば、Microchip 社の EEPROM は A1 と A0 ピンをチップ選択用に使用し、A2 ピンは未使用です。一方、Atmel 社の EEPROM はチップ選択用に A2 と A1 ピンを使用し、A0 ピンは未使用です。これらの両方のケースはブートローダによって対応します。アドレス指定方式はファームウェア イメージ ヘッダに示されます。

Table 17 に、4 つの Microchip 24LC1024 EEPROM デバイスの接続方法を示します。

Table 17　　Microchip 24LC1024 EEPROM デバイスの接続

デバイス番号	アドレス範囲	A2 A1 A0	サイズ
1	0x00000～0x1FFFF	Vcc 0 0	128KB
2	0x20000～0x3FFFF	Vcc 0 1	128KB
3	0x40000～0x5FFFF	Vcc 1 0	128KB
4	0x60000～0x7FFFF	Vcc 1 1	128KB

Table 18 に、4 つの Atmel 24C1024 EEPROM デバイスの接続方法を示します。

I2C EEPROM ブート

Table 18　ATMEL 24C1024 EEPROM デバイスの接続

デバイス番号	アドレス範囲	A2 A1 A0	サイズ
1	0x00000～0x1FFFF	0 0 NC	128KB
2	0x20000～0x3FFFF	0 1 NC	128KB
3	0x40000～0x5FFFF	1 0 NC	128KB
4	0x60000～0x7FFFF	1 1 NC	128KB

Note:　NC とは未接続を意味します。

例えば、ファームウェア コード サイズが 128KB より大きい場合、上記の 2 つの表に示すアドレス指定方式で 2 つの I²C EEPROM を使用する必要があります。ファームウェア イメージは、シングル I²C EEPROM 内の連続したイメージとして各 EEPROM に保存する必要があります。

5.3　ブート イメージ フォーマット

ブートローダはファームウェア イメージ ファイルが **Table 19** に示すフォーマットであると予想します。**EZ-USB™ FX3 SDK** は、I²C EEPROM ブートに必要なフォーマットでファームウェア イメージを生成するためのソフトウェア ユーティリティを提供します。SDK をインストールした後、*C:\Program Files\Cypress\EZ-USB FX3 SDK\1.3\util\elf2img* ディレクトリにある *elf2img* ユーティリティを参照してください。64 ビット システムでは、パスにある最初のフォルダは Program Files (x86) です。ディレクトリ パスにある *1.3* 番号は SDK のバージョン番号ですが、FX3 SDK の最新リリースによって異なります。elf2img ユーティリティの使用の詳細については、**付録 C** および**付録 A** の **Figure 21** を参照してください。

Table 19　ファームウェア イメージ保存フォーマット

2 進数イメージ ヘッダ	長さ (16 ビット)	説明
WSignature	1	ASCII 表記の「CY」の初期値を持つ 2 バイトのシグネチャ
bImageCTL;	½	ビット 0 = 0:2 進数実行ファイル；1:データ ファイル タイプ ビット 3～1 (I²C サイズ): 7: 128 KB (Microchip) 6: 64 KB (128K ATMEL および 256K ST Electronics) 5: 32 KB 4: 16 KB 3: 8 KB 2: 4 KB *Note:　オプション 1 と 0 は将来使用のために予約済み。これらのモードでブートすると、予測できない結果が発生* *ビット 5～4 (I²C 速度):* *00: 100 kHz* *01: 400 kHz* *10: 1 MHz* *11: 予約済み*

I2C EEPROM ブート

2 進数イメージ ヘッダ	長さ (16 ビット)	説明
		Note: ブートローダの電源投入時にデフォルト値は 100kHz であり、必要に応じて I²C 速度を調整 *ビット 7～6: 予約済み；「0」にセットする必要がある*
bImageType;	½	bImageType = 0xB0: チェックサムを含む普通の 2 進数ファームウェア イメージ bImageType = 0xB2: 新規 VID と PID による I²C ブート
dLength 0	2	ロング ワード (32 ビット) での最初のセクションの長さ bImageType = 0xB2 の場合、dLength 0 は PID と VID を含む。ブートローダは残りの後続データを無視
dAddress 0	2	I²C アドレスではなく、プログラム コードの最初のセクション アドレス *Note: 内部 ARM アドレスはバイト アドレス指定が可能で、各セクションのアドレスは 32 ビット アラインである必要がある*
dData[dLength 0]	dLength 0*2	すべてのイメージ コード/データも 32 ビット アラインである必要がある
…		他のセクション
dLength N	2	0x00000000 (最後の記録:終端セクション)
dAddress N	2	有効なプログラム エントリ (通常はスタートアップ コード、すなわち RESET ベクタ) を含む必要がある *Note: bImageCTL.bit0 = 1 の場合、ブートローダはこのプログラム エントリに実行を転送しない* *bImageCTL.bit0 = 0 の場合、ブートローダはこのプログラム エントリに実行を転送。このアドレスは ITCM 空間または SYSTEM RAM 空間にある必要がある。* *ブートローダはプログラム エントリを確認しない*
dCheckSum	2	32 ビット符号なしのリトルエンディアン チェックサム データは最初のセクションから終端セクションまで。チェックサムは dLength、dAddress、およびイメージ ヘッダを含まない

例: 2 進数イメージ ファイルは I²C EEPROM に以下の順序で保存されます。

バイト 0: 「C」

バイト 1: 「Y」

バイト 2:bImageCTL

バイト 3:bImageType

…..

バイト N: イメージのチェックサム

I2C EEPROM ブート

Attention:

- ブートローダのデフォルト ブート速度 = 100kHz。その速度を 100kHz から 1MHz に変更するために、bImageCTL<5:4>を「10」にセットします。
- I^2C EEPROM サイズを選択するために、bImageCTL[3:1]を使用します。
- Microchip EEPROM 24LC1026 のアドレス指定は、他の 128KB の Microchip EEPROM のアドレス指定とは違います。Microchip EEPROM 24LC1026 を使用する場合、I^2C EEPROM サイズ フィールド (例えば、bImageCTL[3:1]) を「6」にセットする必要があります。
- FX3 SDK で生成された.img ファイルの bImageCTL バイトのデフォルト値は 0x1C (サイズ 64KB、スピード 400KHz) です。 異なる I^2C 構成のイメージファイルを生成する方法の詳細については、**Generate bootable image file for FX3 with I^2C EEPROM - KBA218344** を参照してください。

5.4 チェックサムの計算

ブートローダは、I^2C EEPROM 内の 2 進数イメージをロードする時にチェックサムを計算します。この値はイメージのチェックサム値と一致しない場合、ブートローダはプログラム エントリに実行を転送しません。

ブートローダがリトルエンディアン モードで作動するため、チェックサムはリトルエンディアン モードで計算する必要があります。

32 ビット符号なしのリトルエンディアン チェックサム データは最初のセクションから終端セクションまでです。チェックサムは dLength、dAddress、およびイメージ ヘッダを含みません。

5.4.1 ブート イメージの例 1

次のイメージは、FX3 のシステム RAM 上の 1 つのセクション (0x40008000 の位置) にのみ保存されます。

```
Location1: 0xB0 0x1A 'Y' 'C' //CY Signature, 32KB EEPROM,400Khz,0xB0 Image
Location2:  0x00000004       //Image length =4
Location3:  0x40008000       // 1st section stored in FX3 System RAM at 0x40008000
Location4: 0x12345678        //Image starts
Location5: 0x9ABCDEF1
Location6: 0x23456789
Location7: 0xABCDEF12
Location8: 0x00000000        //Termination of Image
Location9: 0x40008000        //Jump to 0x40008000 in FX3 System RAM
Location 10: 0x7C048C04      //Check sum (0x12345678 + 0x9ABCDEF1 + 0x23456789 +
                               0xABCDEF12)
```

5.4.2 ブート イメージの例 2

次のイメージは、FX3 のシステム RAM 上の 2 つのセクション (0x40008000 と 0x40009000 の位置) に保存されます。

```
Location1: 0xB0 0x1A 'Y' 'C' //CY Signature, 32KB EEPROM,400Khz,0xB0 Image
Location2:  0x00000004       //Image length of section 1 =4
Location3:  0x40008000       //1st section stored in FX3 System RAM at 0x40008000
Location4: 0x12345678        //Image starts (Section1)
Location5: 0x9ABCDEF1
Location6: 0x23456789
Location7: 0xABCDEF12        //Section 1 ends
Location8:  0x00000002       //Image length of section 2 =2
```

```
Location9: 0x40009000        //2nd section stored in FX3 System RAM at 0x40009000
Location10: 0xDDCCBBAA       //Section 2 starts
Location11: 0x11223344
Location12: 0x00000000       //Termination of Image
Location13 0x40008000        //Jump to 0x40008000 in FX3 System RAM
Location 14: 0x6AF37AF2      //Check sum (0x12345678 + 0x9ABCDEF1 + 0x23456789 +
                               0xABCDEF12 + 0xDDCCBBAA + 0x11223344)
```

同様に、1 つのブート イメージを N セクションに保存できます。

FX3 DVK を使用して USB ブート モードをテストする段階的な手順は、**付録 A** の **I2C ブート** で記述します。

5.4.3 チェックサム計算のサンプル コード

以下はチェックサムのサンプル コードです。

```
// Checksum sample code
DWORD dCheckSum, dExpectedCheckSum;
WORD wSignature, wLen;
DWORD dAddress, i;
DWORD dImageBuf[512*1024];

fread(&wSignature,1,2,input_file); // read signature bytes
if (wSignature != 0x5943)          // check 'CY' signature byte
{
   printf("Invalid image");
   return fail;
}
fread(&i, 2, 1, input_file);       // skip 2 dummy bytes
dCheckSum = 0;
while (1)
{
   fread(&dLength,4,1,imput_file);  // read dLength
   fread(&dAddress,4,1,input_file); // read dAddress
   if (dLength==0) break;           // done
   // read sections
   fread(dImageBuf, 4, dLength,  input_file);
   for (i=0; i<dLength; i++) dCheckSum += dImageBuf[i];
}
// read pre-computed checksum data
fread(&dExpectedChecksum, 4, 1, input_file);
if  (dCheckSum != dExpectedCheckSum)
{
   printf("Fail to boot due to checksum error\n");
  return fail;

}
```

ここでは、I²C ブート オプションの詳細について説明しました。次節では、USB フォールバックが有効の I²C ブート オプションを説明します。

6 USB フォールバックが有効の I²C EEPROM ブート

USB フォールバックが有効の I²C EEPROM ブートでは、**Table 20** に示すように PMODE[2:0]ピンの状態は Z1Z です。

Table 20 USB フォールバックが有効の I²C ブートの PMODE ピン

PMODE[2]	PMODE[1]	PMODE[0]
Z	1	Z

すべての USB フォールバック モード (「--> USB」と表記) では、0xB2 ブートが選択されるか、またはエラーが発生する場合、USB エニュメレーションが行われます。USB エニュメレーション後、外部 USB ホストは USB ブートで FX3 をブートすることができます。また、USB フォールバックが有効の I²C EEPROM ブート (I²C --> USB) は、USB ブート用のベンダー識別子 (VID) と製品識別子 (PID) のみ保存する場合もあります。

I²C EEPROM ブートは以下の条件で失敗になります。

- I²C アドレス サイクルまたはデータ サイクル エラー
- FX3 ファームウェア イメージでの無効なシグネチャ
- 無効なイメージ タイプ

FX3 ファームウェア イメージの代わりに EEPROM 上のデータが USB ブート用の VID と PID であることを示すために、特別なイメージ タイプは使用されます。これにより、USB ブート用の新規 VID と PID を得られるようになります。

6.1 特長

- USB ブートの場合、ブートローダは USB 3.0 をサポートせず、USB 2.0 のみをサポートします。
- 0xB2 ブート オプションが指定される場合、USB ディスクリプタは、I²C EEPROM の 0xB2 イメージの一部として保存される顧客定義 VID と PID を使用します。
- USB フォールバックでは、I²C ブート中にエラーが発生すると、USB ディスクリプタは VID = 0x04B4 および PID = 0x00F3 を使用します。
- USB デバイス ディスクリプタは約 200mA を消費するバスパワーとして報告されます。しかし、FX3 チップは通常、約 100mA の電流を消費することが観察されます。

6.2 VID と PID によるブート イメージ例

```
Location1: 0xB2 0x1A 'Y' 'C' //CY Signature,32k EEPROM,400Khz,0xB2 Image
Location2: 0x04B40008        //VID = 0x04B4 | PID=0x0008
```

7 SPI ブート

Figure 5 に、SPI を介してブートする場合の FX3 システム図を示します。

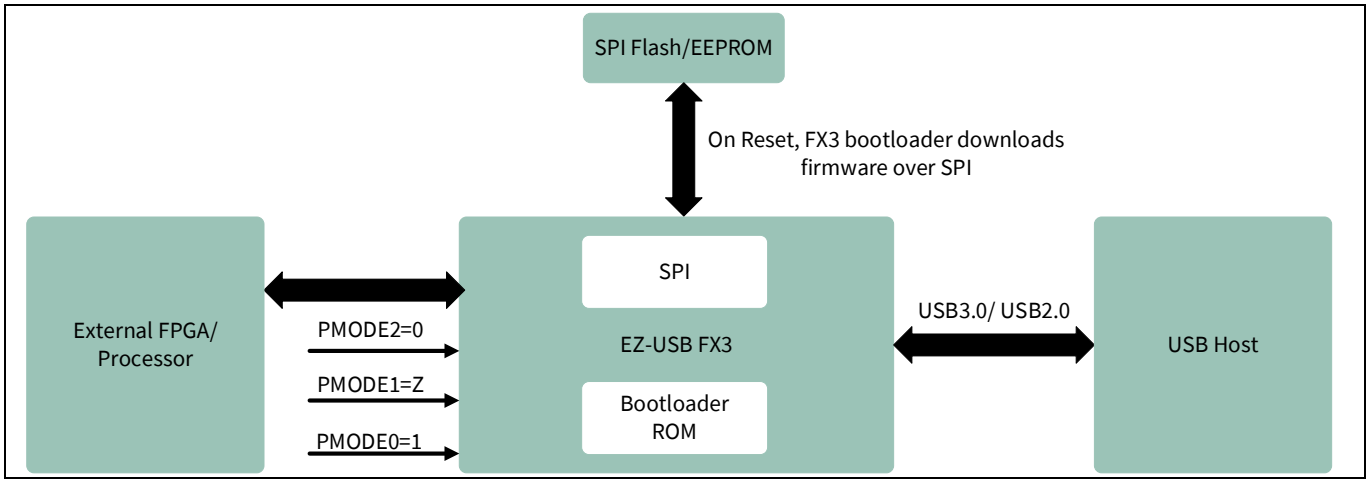

Figure 5 SPI ブートのシステム図

SPI ブートでは、Table 21 に示すように PMODE[2:0]ピンの状態は 0Z1 です。

Table 21 SPI ブートの MODE ピン

PMODE[2]	PMODE[1]	PMODE[0]
0	Z	1

FX3 SPI インタフェースのピン マッピングを Table 22 に示します。

Table 22 SPI インタフェースのピン マッピング

EZ-USB™ FX3 のピン	SPI インタフェース
GPIO[53]	SPI_SCK
GPIO[54]	SPI_SSN
GPIO[55]	SPI_MISO
GPIO[56]	SPI_MOSI

7.1 特長

FX3 は、SPI フラッシュ/EEPROM デバイスから 4-線式 SPI インタフェースを介してブートします。

- サイズが 1 Kb～128 Mb の SPI フラッシュ/EEPROM デバイスはブートに対応します。

  サポートされる SPI フラッシュ部品：

  - SPI フラッシュ (S25FS064S (64 Mbit), S25FL064L (64 Mbit) および S25FS128S (128 Mbit))
  - Winbond W25Q32FW (32 Mbit)
- ブートの際にサポートされる SPI 周波数は 10 MHz, 20 MHz, および 30 MHz です。
- SPI 周波数が SPI クロック分周器とクロック入力の丸めによって異なることに注意してください。
  - FX3 への水晶またはクロック入力が 26 MHz または 52 MHz の場合、内部 PLL は 416 MHz で動作します。PLL_CLK = 416 MHz での SPI 周波数は 10.4 MHz, 20.8 MHz, または 34.66 MHz となります。

SPI ブート

- FX3 への水晶またはクロック入力が 19.2 MHz または 38.4 MHz の場合、内部 PLL は 384 MHz で動作します。PLL_CLK = 384MHz での SPI 周波数は 9.6 MHz, 19.2 MHz, および 32 MHz となります。

- 対応動作電圧は 1.8 V, 2.5 V, および 3.3 V です。
- SPI フラッシュ/EEPROM に保存できるファームウェア イメージは 1 つのみです。重複イメージは許可されません。
- SPI ブートでは、ブートローダは CPOL = 0 と CPHA = 0 にセットします。(この SPI モードのタイミング図については、**FX3 データシート**の SPI タイミングの節を参照してください。)
- USB フォールバックはサポートされ、USB ブート用の新規 VID/PID 情報を保存するために使用されます。詳細については、本アプリケーション ノートの **USB フォールバックが有効の SPI ブート**を参照してください。

7.2 SPI フラッシュの選択

SPI フラッシュは FX3 のブートに対応するために、以下のコマンドをサポートします。

- データ読み出し: 03h、3 バイトのアドレス指定
- ステータス レジスタ読み出し: 05h
- 書き込みイネーブル: 06h
- データ書き込み (ページ プログラム): 02h
- セクタ消去: D8h

読み出しコマンドが一致する限り、SPI フラッシュは FX3 のブートに使用できます。書き込みコマンドに違いがある場合、SPI フラッシュのプログラミングは、提供された *CyBootProgrammer.img* (*C:\Program Files (x86)\Cypress\EZ-USB FX3 SDK\1.3\bin* に配置) で成功しません。FX3 SDK の USBFlashProg サンプル プロジェクトで使用される SPI 書き込みコマンドを変更する必要があります。変更した USBFlashProg プロジェクトをビルドした後に作成したイメージ ファイルは、SPI フラッシュのプログラミングが正常に終了するために、デフォルトの *CyBootProgrammer.img* を (同じ名前で) 置き換える必要があります。

7.3 ハイブリッド セクタを持つ SPI フラッシュの使用方法

一部の SPI フラッシュ・デバイス (S25FS128S/S25FS256S など) は、ハイブリッド セクタをサポートする場合があります。例えば、64KB の第 1 セクタを 9 セクタ (4KB の 8 セクタと 32KB の 1 セクタ) に分割できます。残りのセクタはすべて一律 (各 64KB) です。FX3 SDK の USB Control Center アプリケーションを使用して SPI フラッシュをプログラミングする場合、内部プログラミングユーティリティ (CyBootProgrammer.img) はフラッシュデバイスのセクタ サイズが均一であると仮定しており、ハイブリッドセクタである最初のセクタでデータが破損してしまうことがあります。これを防ぐためには、適切なコンフィギュレーション・レジスタを変更し、各 64KB のユニフォーム セクタを使用するようにフラッシュ・デバイスを設定してください。

FX3 でのハイブリッド SPI フラッシュ デバイスの使用については、**KBA231163** を参照してください。

7.4 ファームウェア イメージの SPI フラッシュ/EEPROM への保存

FX3 ブートローダは、外部シリアル SPI フラッシュ/EEPROM デバイスと相互接続するためにマスターSPI コントローラーをサポートします。SPI フラッシュ/EEPROM はアプリケーション固有のコードとデータを格納するために使用できます。**Figure 6** に標準 SPI フラッシュ/EEPROM のピン配置を示します。

SPI EEPROM インタフェースは 4 つのアクティブ線を含みます。

- CS#: チップ セレクト
- SO: シリアル データ出力 (マスター入力、スレーブ出力 (MISO))
- SI: シリアル データ入力 (マスター出力、スレーブ入力 (MOSI))

SPI ブート

- SCK: シリアル クロック入力

HOLD#信号は、SPI デバイスからブートする、または読み出す中に VCC に接続する必要があります。

書き込み保護 (WP#) と HOLD#信号は、EEPROM にイメージを書き込む中に VCC に接続する必要があります。

Note: ***Figure 6*** *に示すように、外部プルアップ抵抗は MOSI と MISO 信号に接続してはいけません。*

Note: *MISO ラインにプルダウン抵抗 (2K) を含めることを推奨します。*

Note: *SPI ピンの接続に関する推奨事項については、****AN70707 EZ-USB FX3/FX3S ハードウェア設計ガイドラインおよび回路図チェックリスト****の「SPI および UART」セクションを参照してください。*

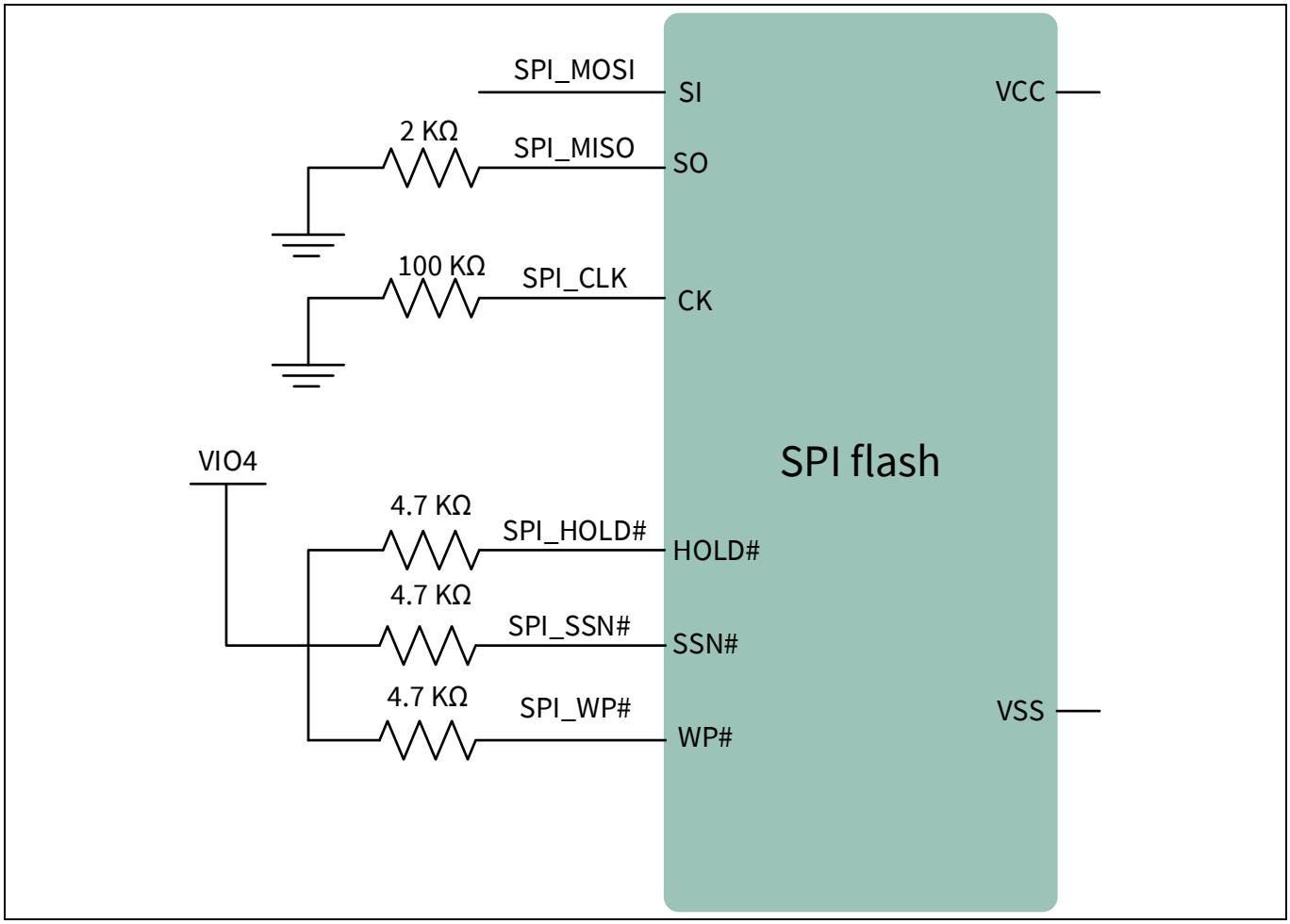

Figure 6 M25P40-VMN6TPB SPI フラッシュへの接続端子例

7.5 ブート イメージ フォーマット

SPI ブートでは、ブートローダはファームウェア イメージ ファイルが **Table 23** に示すフォーマットであると予想しています。**EZ-USB™ FX3 SDK** は、SPI ブートに必要なフォーマットでファームウェア イメージを生成するためのソフトウェア ユーティリティを提供します。SDK をインストールした後、*C:\Program Files\Cypress\EZ-USB FX3 SDK\1.3\util\elf2img* ディレクトリにある *elf2img* ユーティリティを参照してください。64 ビット システムでは、パスにある最初のフォルダは Program Files (x86) です。ディ

SPI ブート

レクトリ パスにある *1.3* 番号は SDK のバージョン番号ですが、FX3 SDK の最新リリースによって異なります。elf2img ユーティリティの使用の詳細については、**付録 C** および**付録 A** の **Figure 29** を参照してください。

Table 23　SPI ブート オプション用のブート イメージ フォーマット

2 進数イメージ ヘッダ	長さ (16 ビット)	説明
wSignature	1	ASCII 表記の「CY」の初期値を持つ 2 バイトのシグネチャ
bImageCTL	½	ビット 0 = 0:2 進数実行ファイル；1:データ ファイル タイプ ビット 3〜1:SPI からブートする場合は未使用 ビット 5〜4 (SPI 速度): 00: 10 MHz 01: 20 MHz 10: 30 MHz 11: 予約済み *Note:　ブートローダの電源投入時にデフォルト値は 10 MHz であり、必要に応じて SPI 速度を調整。FX3 SPI ハードウェアの最大動作速度は 33 MHz* *ビット 7〜6: 予約済み。「0」にセットする必要がある*
bImageType	½	bImageType = 0xB0: チェックサムを含む普通の 2 進数ファームウェア イメージ bImageType = 0xB2: 新規 VID および PID による SPI ブート
dLength 0	2	ロング ワード (32 ビット) での最初のセクションの長さ bImageType = 0xB2 の場合、dLength 0 は PID と VID を含む。ブートローダは残りの後続データを無視
dAddress 0	2	プログラム コードの最初のセクション アドレス *Note:　内部 ARM アドレスはバイト アドレス指定が可能で、各セクションのアドレスは 32 ビット アラインである必要がある*
dData[dLength 0]	dLength 0*2	イメージ コード/データは 32 ビット アラインである必要がある
…		他のセクション
dLength N	2	0x00000000 (最後の記録:終端セクション)
dAddress N	2	有効なプログラム エントリ (通常はスタートアップ コード、すなわち RESET ベクタ) を含む必要がある *Note:　bImageCTL.bit0 = 1 の場合、ブートローダはこのプログラム エントリに実行を転送しない* *bImageCTL.bit0 = 0 の場合、ブートローダはこのプログラム エントリに実行を転送。このアドレスは ITCM 空間または SYSTEM RAM 空間にある必要がある* *ブートローダはプログラム エントリを確認しない*

2 進数イメージ ヘッダ	長さ (16 ビット)	説明
dCheckSum	2	32 ビット符号なしのリトルエンディアン チェックサム データは最初のセクションから終端セクションまで。チェックサムは dLength、dAddress、およびイメージ ヘッダを含まない

例: 2 進数イメージ ファイルは SPI EEPROM に以下の順序で保存されます。

バイト 0: 「C」

バイト 1: 「Y」

バイト 2:bImageCTL

バイト 3:bImageType

…..

バイト N: イメージのチェックサム

重要な注意点:

ブートローダのデフォルト ブート速度 = 10 MHz です。その速度を 10 MHz から 20 MHz に変更するために、bImageCTL[5:4]を「01」にセットします。

7.6 チェックサムの計算

ブートローダは、SPI を介して 2 進数イメージをロードする時にチェックサムを計算します。この値はイメージのチェックサム値と一致しない場合、ブートローダはプログラム エントリに実行を転送しません。

ブートローダがリトルエンディアン モードで作動するため、チェックサムはリトルエンディアン モードで計算する必要があります。

32 ビット符号なしのリトルエンディアン チェックサム データは最初のセクションから終端セクションまでです。チェックサムは dLength、dAddress、およびイメージ ヘッダを含みません。チェックサムを計算するサンプル コードについては、**チェックサム計算のサンプル コード**を参照してください。

例 1: 次の例は、FX3 のシステム RAM 上の 1 つのセクション (0x40008000 の位置) にのみ保存されるファームウェア イメージです。

```
Location1: 0xB0 0x10 'Y' 'C' //CY Signature, 20 MHz, 0xB0 Image
Location2: 0x00000004        //Image length = 4
Location3: 0x40008000        //1st section stored in FX3 System RAM at 0x40008000
Location4: 0x12345678        //Image starts
Location5: 0x9ABCDEF1
Location6: 0x23456789
Location7: 0xABCDEF12
Location8: 0x00000000        //Termination of Image
Location9: 0x40008000        //Jump to 0x40008000 in FX3 System RAM
Location 10: 0x7C048C04      //Checksum (0x12345678 + 0x9ABCDEF1 + 0x23456789 +
                               0xABCDEF12)
```

SPI ブート

例 2: 次の例は、FX3 のシステム RAM 上の 2 つのセクション (0x40008000 と 0x40009000 の位置) に保存されるファームウェア イメージです。

```
Location1: 0xB0 0x10 'Y' 'C' //CY Signature, 20MHz, 0xB0 Image
Location2: 0x00000004        //Image length of section 1 = 4
Location3: 0x40008000        //1st section stored in FX3 System RAM at 0x40008000
Location4: 0x12345678        //Image starts (Section1)
Location5: 0x9ABCDEF1
Location6: 0x23456789
Location7: 0xABCDEF12        //Section 1 ends
Location8: 0x00000002        //Image length of section 2 = 2
Location9: 0x40009000        //2nd section stored in FX3 System RAM at 0x40009000
Location10: 0xDDCCBBAA       //Section 2 starts
Location11: 0x11223344
Location12: 0x00000000       //Termination of Image
Location13: 0x40008000       //Jump to 0x40008000 in FX3 System RAM
Location 14: 0x6AF37AF2      //Checksum (0x12345678 + 0x9ABCDEF1 + 0x23456789 +
                               0xABCDEF12 + 0xDDCCBBAA + 0x11223344)
```

同様に、1 つのブート イメージを N セクションに保存できます。

FX3 DVK を使用して USB ブート モードをテストする段階的な手順は、**付録 A** の **SPI ブート**で記述します。

8 USB フォールバックが有効の SPI ブート

すべての USB フォールバック (「-->USB」) モードでは、0xB2 ブートが選択されるかまたはエラーが発生する場合、USB エニュメレーションが行われます。USB エニュメレーション後、外部 USB ホストは USB ブートで FX3 をブートできます。また、USB フォールバックが有効の SPI ブート (SPI --> USB) は、USB ブート用の VID と PID を保存するためにも使用されます。

SPI ブートは、以下の条件で失敗になります。

- SPI アドレス サイクルまたはデータ サイクル エラー
- FX3 ファームウェアでの無効なシグネチャ無効なイメージ タイプ

FX3 ファームウェア イメージの代わりに SPI フラッシュ/EEPROM 上のデータが USB ブート用の VID と PID であることを示すために、特別なイメージ タイプが使用されます。これにより、USB ブート用の新規 VID と PID を得られるようになります。

- USB ブートの場合、ブートローダは USB 3.0 をサポートせず、USB 2.0 のみをサポートします。
- 0xB2 ブート オプションが指定される場合、USB ディスクリプタは、SPI フラッシュ/EEPROM の 0xB2 イメージの一部として保存される顧客定義 VID と PID を使用します。
- USB フォールバックでは、I^2C ブート中にエラーが発生すると、USB ディスクリプタは VID = 0x04B4 および PID = 0x00F3 を使用します。
- USB デバイス ディスクリプタは約 200 mA を消費するバスパワーとして報告されます。しかし、FX3 チップは通常、約 100mA の電流を消費することが観察されます。

8.1 VID と PID によるブート イメージ例

```
Location1: 0xB2 0x10 'Y' 'C' //CY Signature, 20 MHz, 0xB2 Image
Location2: 0x04B40008        //VID = 0x04B4 | PID = 0x0008
```

次の節では、同期 ADMux インタフェースおよびそれによるブートの詳細について説明します。

9 同期 ADMux ブート

Figure 7 に、同期 ADMux インタフェースを介してブートする場合の FX3 システム図を示します。

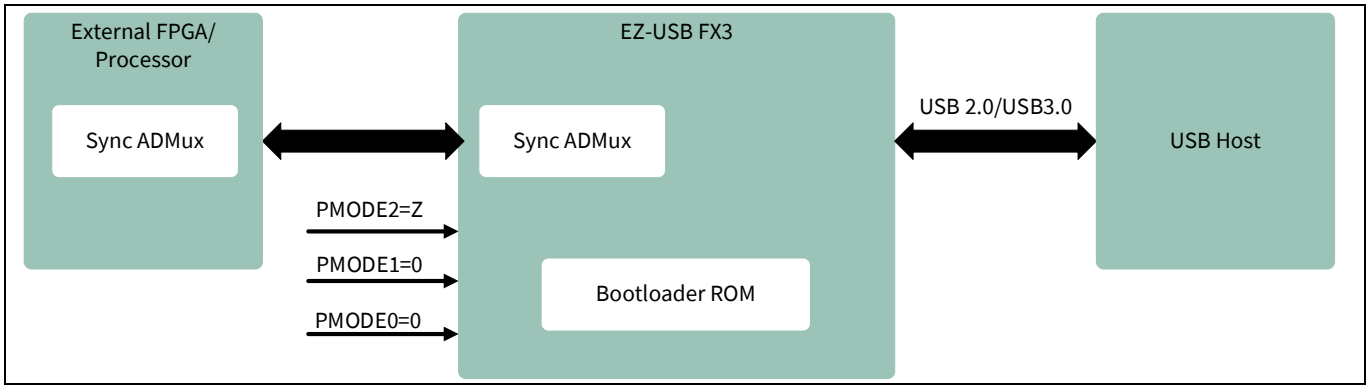

Figure 7 同期 ADMux ブートのシステム図

同期 ADMux インタフェースを介してブートする場合、Table 24 に示すように PMODE[2:0]ピンの状態は Z00 です。

Table 24 同期 ADMux ブートの PMODE ピン

PMODE[2]	PMODE[1]	PMODE[0]
Z	0	0

FX3 GPIF II インタフェースは、外部プロセッサまたは FPGA からファームウェア イメージをダウンロードするために使用される同期 ADMux インタフェースをサポートします。ブートローダによって設定された同期 ADMux インタフェースは次の信号を含みます。

- PCLK: FX3 へのクロック入力である必要があります。クロック入力のサポートされる最大周波数は 100 MHz です。
- DQ[15:0]: 16 ビット データ バス
- A[7:0]: 8 ビット アドレス バス
- CE#: アクティブ LOW チップ イネーブル
- ADV#: アクティブ LOW アドレス有効
- WE#: アクティブ LOW 書き込みイネーブル
- OE#: アクティブ LOW 出力イネーブル
- RDY: アクティブ HIGH レディー信号

9.1.1 インタフェース信号

Figure 8 に、ブートローダによって設定され、外部プロセッサに接続する同期 ADMux インタフェースの標準的な相互接続図を示します。

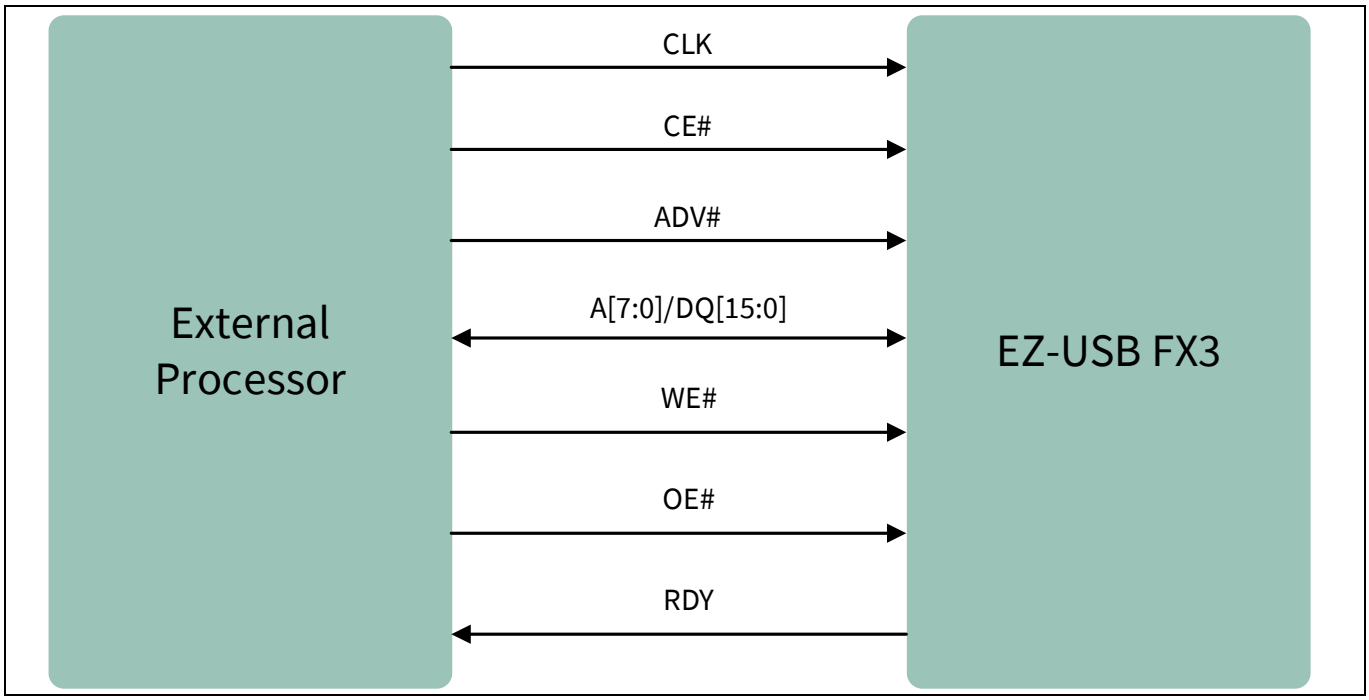

Figure 8 同期 ADMUX インタフェース

読み出し動作では、CE#と OE#両方はアサートされます。

書き込み動作では、CE#と WE#両方はアサートされます。

ADV#は、読み書き動作のアドレス フェーズ中には LOW であり、読み書き動作のデータ フェーズ中には HIGH である必要があります。

FX3 デバイスからの RDY 出力信号は、データが読み出し転送用に有効であることを示します。

FX3 同期 ADMux インタフェースのピン マッピングを Table 25 に示します。

Table 25 同期 ADMux インタフェースのピン マッピング

EZ-USB™ FX3 のピン	同期 ADMux インタフェース
GPIO[7:0]/GPIO[15:0]	A[7:0]/DQ[15:0]
GPIO[16]	CLK
GPIO[17]	CE#
GPIO[18]	WE#
GPIO[19]	OE#
GPIO[25]	RDY
GPIO[27]	ADV#

9.1.2 同期 ADMux のタイミング

同期 ADMux タイミング図 (同期 ADMux インタフェースの読み出しサイクル タイミングと書き込みサイクル タイミング) およびタイミング パラメーターの詳細については、**Figure 9**, **Figure 10**, および **Table 26** を参照してください。

同期 ADMUX モードの電源投入遅延

電源投入時または RESET#ライン上のハード リセットの時、ブートローダは同期 ADMux インタフェース用に GPIF II を設定するために若干の時間を要します。このプロセスは数百マイクロ秒かかります。FX3 への読み出し/書き込みアクセスは、ブートローダが設定を完了した後にのみ実行する必要があります。そうしないと、データ破損が発生する可能性があります。これを防ぐために、以下の方式のいずれか 1 つ使用してください。

- RESET#デアサート後、1 ms 待ちます。
- 0x81 の値が返されるまで PP_IDENTIFY レジスタをポーリングし続けます。
- INT#信号がアサートするまで待ち、RD_MAILBOX レジスタを読み出して、返される値が 0x42575943 (すなわち、「CYWB」) に等しいかを確認します。

9.1.3 USB フォールバック (-->USB)

同期 ADMUX ブート中にコマンドでエラーが発生しても、USB フォールバックは有効にされません。

9.1.4 ウォーム ブート

ウォーム ブートが検出される時、ブートローダは、以前保存した「プログラム エントリ」 (ユーザーの RESET ベクタでもあり得る) に実行を転送します。この場合には、GPIF II コンフィギュレーションは保存されます。

9.1.5 ウェイクアップ/スタンバイ

スタンバイ モードからのウェイクアップ後、アプリケーション ファームウェアはハードウェア レジスタ、GPIF II コンフィギュレーション、ITCM または DTCM を設定し、復元します。

スタンバイ モードからのウェイクアップ後、ブートローダは ITCM と DTCM の両方が有効にされたかを確認します。

Note: スタンバイ モードまたはウォーム ブート プロセスからウェイクアップする場合、ブートローダはリセット割込みサービス サブルーチンにジャンプして次のことを実行します。

- DCACHE と ICACHE の両方を無効にします。
- ICACHE をオンにします。
- MMU を無効にします。
- DTCM と ITCM をオンにします。
- DTC を使ってスタックを設定します。

ブートローダは 0x1000_0000～0x1000_04FF の 0x500 バイトを割り当てるため、0x1000_0500～0x1000_1FFF はファームウェアのダウンロードに利用できます。ダウンロード アプリケーションが引き継ぐと、0x1000_0000～0x1000_04FF のメモリは他の目的に使用できます。

同期 ADMux ブート

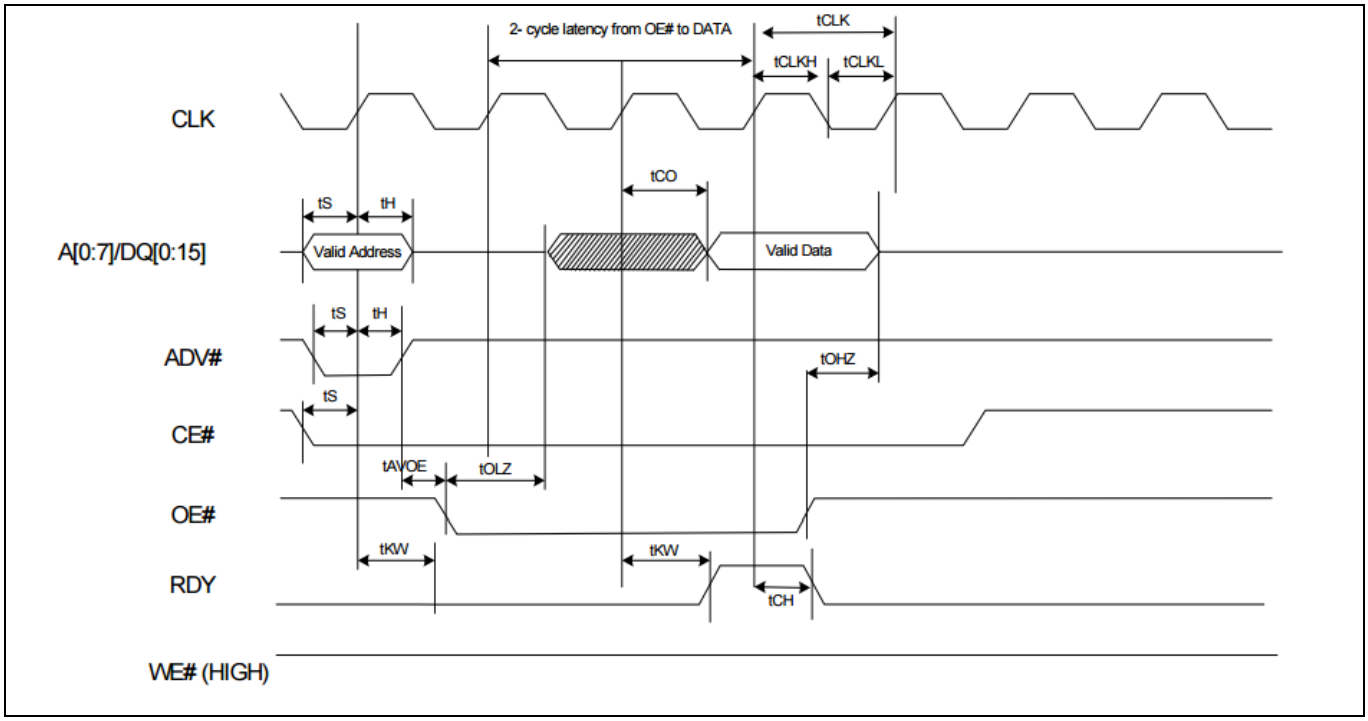

Figure 9　　同期 ADMux インタフェース – 読み出しサイクル タイミング

Note:

1. *外部 P-Port プロセッサと FX3S は同じクロック エッジで動作します。*
2. *外部プロセッサは、OE#がアサートしてから 2 サイクル後に RDY のアサートを検出し、データが出力に現れてから 1 サイクル後に RDY のアサート解除を検出します。*
3. *有効な出力データは OE#がアサートしてから 2 サイクル後に現れます。このデータは OE#がアサート解除するまで保持されます。*
4. *0～100MHz 動作では 2 サイクル遅延が適用されます。50MHz 以下の動作では遅延を 1 サイクルに減少できます。(この 1 サイクル遅延はブートローダによってサポートされません。)*

同期 ADMux ブート

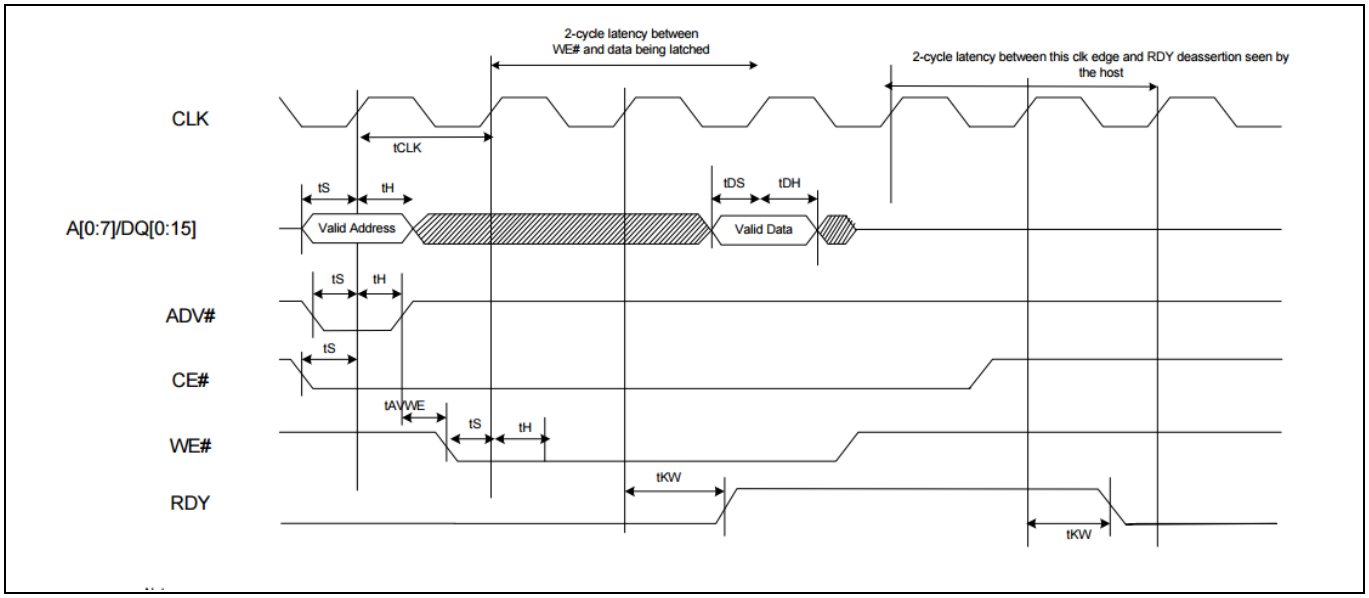

Figure 10　　同期 ADMux インタフェース – 書き込みサイクル タイミング

Note:

1. *外部 P-Port プロセッサと FX3S は同じクロック エッジで動作します。*
2. *外部プロセッサは、WE#がアサートしてから 2 サイクル後に RDY のアサートを検出し、データがサンプリングされるエッジから 3 サイクル後に RDY のアサート解除を検出します。*
3. *0～100 MHz 動作では 2 サイクル遅延が適用されます。50 MHz 以下の動作では遅延を 1 サイクルに減少できます。(この 1 サイクル遅延はブートローダによってサポートされません。)*

Table 26　　同期 ADMux タイミングのパラメーター

パラメーター	説明	Min	Max	単位
FREQ	インタフェース クロック周波数	-	100	MHz
tCLK	クロック周期	10	-	ns
tCLKH	クロック HIGH 時間	4	-	ns
tCLKL	クロック LOW 時間	4	-	ns
tS	CE#/WE#/DQ セットアップ時間	2	-	ns
tH	CE#/WE#/DQ ホールド時間	0.5	-	ns
tCH	クロックからデータ出力ホールドまでの時間	0	-	ns
tDS	データ入力セットアップ時間	2	-	ns
tDH	クロックからデータ入力ホールドまでの時間	0.5	-	ns
tAVDOE	ADV# HIGH から OE# LOW までの時間	0	-	ns
tAVDWE	ADV# HIGH から WE# LOW までの時間	0	-	ns
tHZ	CE# HIGH からデータ H-Z までの時間	-	8	ns
tOHZ	OE# HIGH からデータ HI-Z までの時間	-	8	ns

同期 ADMux ブート

パラメーター	説明	Min	Max	単位
tOLZ	OE# LOW からデータ HI-Z までの時間	0	-	ns
tKW	クロックから RDY 有効までの時間	-	8	ns

9.1.6 GPIF II API プロトコル

このプロトコルは GPIF II ブート モードでのみ使用されます。リセット後、外部アプリケーション プロセッサ (AP) は、**Table 27** に定義されたコマンド プロトコルを使用してブートローダと通信します。

Table 27 GPIF II API プロトコル

フィールド	説明
bSignature[2]	**2 バイト** 「CY」の初期値を持つ送信元 ブートローダは「WB」で応答
bCommand	送信元: 1 バイトのコマンド 0x00: NOP 0x01: WRITE_DATA_CMD: データ書き込みコマンド 0x02: ブート モードに入る 0x03: READ_DATA_CMD: データ読み出しコマンド ブートローダは、他の全ての値を動作なしと取り扱い、bLenStatus でエラー コードを返す
bLenStatus	**入力: (1 バイト)** bCommand 00:bLenStatus = 0 (bCommand が WRITE_DATA_CMD である場合、ブートローダは dAddr 内のアドレスにジャンプし、他の全てのコマンドの値を無視) bCommand 01: ロング ワードでの長さ (Max = (512-8)/4) bCommand 02: 512 バイト ブロックの数 (Max = 16) bCommand 03: ロング ワードでの長さ (Max = (512-8)/4) ブートローダは PIB_RD_MAILBOX1 レジスタ内の次のデータで応答: 0x00: 成功 0x30: コマンド プロセス エラーで失敗 0x31: 読み出しプロセス エラーで失敗 0x32: 検出中止 0x33: ブートローダからアプリケーションへの PP_CONFIG.BURSTSIZE メールボックス通知。PIB_RD_MAILBOX0 は GPIF_DATA_COUNT_LIMIT レジスタを含む 0x34: ブートローダは DLL _LOST_LOCK を検出。PIB_RD_MAILBOX0 は PIB_DLL_CTRL レジスタを含む 0x35: ブートローダは PIB_PIB_ERR ビットを検出。PIB_RD_MAILBOX0 は PIB_PIB_ERROR レジスタを含む 0x36: ブートローダは PIB_GPIF_ERR ビットを検出。PIB_RD_MAILBOX0 は PIB_PIB_ERROR レジスタを含む
dAddr	**4 バイト** 送信元: コマンド 1 とコマンド 3 で使用されるアドレス
dData[bLenStatus]	bLenStatus で判定されるデータ長 送信元: 送信元によって書き込まれるデータ

同期 ADMux ブート

Note:

1. *エラー コード bLenStatus は GPIF II メールボックスに報告されます。*
2. *同期 ADMUX を使用して FX3 にファームウェアをダウンロードする時、外部 AP は以下のことを保証します。*
   a) コマンド ブロック長は正確に 512 バイトです。
   b) 応答ブロック長は正確に 512 バイトです。
   c) ブートローダの 2 進数イメージは、512 バイトの倍数のセグメントに分割されるデータ ストリームに変換されます。
   d) ブートローダ イメージのデータ チャンクは 8K のサイズを超えません。例えば、コマンド 0x02 で、bLenStatus は 16 ブロック (8K バイト) より大きくしてはいけません。この制限は最大 DMA バッファサイズによるものです。ブートローダがサポートする最大 DMA_SIZE は 8K です。このため、AP は転送ごとに 8K データ (最大) までしか送信できません。ファームウェアのファイルサイズが 8K より大きい場合は、ファームウェアを FX3 へ完全にダウンロードするために、複数回の転送が必要です。すなわち、完全なファームウェアの転送が終わるまで、各転送について **12.4** のステップ 4 を繰り返さなければいけません。
   e) ホストは合計イメージ サイズ以上を送信しません。
3. *ブートローダは待ち行列コマンドをサポートしません。したがって、コマンドが送信されるたびに、ホストは応答を読み出さなければいけません。*
4. *ダウンロード プロセス中における、この API 構造体の破損を防止するために、ホストは、予約したブートローダ システム アドレス (0x4000_0000～0x4000_23FF) にファームウェアをダウンロードしてはいけません。ファームウェア アプリケーションがこの空間を使用しようとすると、エラーが返されます。 DTCM の最初の 1280 バイト (0x1000_0000～0x1000_04FF) も使用してはいけません。*
5. *On the WRITE_DATA_CMD: WRITE_DATA_CMD では、bLenStatus = 0 の場合、ブートローダは dAddr のプログラム エントリにジャンプします。*

9.1.7 ファームウェア ダウンロード例

本節では、16 ビット同期 ADMux インタフェースを介してホスト プロセッサから FX3 にファームウェア ダウンロードを実行する簡単な方法を説明します。

ホスト プロセッサは、ファームウェア ダウンロードを実行するために FX3 ブートローダと通信します。この通信には、ホスト プロセッサが FX3 レジスタとデータ ソケットを読み書きすることが必要です。

*Note: FX3 のソケットの概念については、**EZ-USB™FX3 入門**アプリケーション ノートの FX3 用語の節を参照してください。*

ホスト プロセッサは、利用可能な GPIF II ソケットを使用して FX3 からデータ ブロックを転送します。FX3 ブートローダはファームウェア ダウンロード プロトコルに対応するために 3 つのデータ ソケット (コマンド、応答、およびファームウェア データ用) を保持します。

```
#define CY_WB_DOWNLOAD_CMD_SOCKET     (0x00)    // command block write only socket
#define CY_WB_DOWNLOAD_DATA_SOCKET    (0x01)    // data block read/write socket
```

```
#define CY_WB_DOWNLOAD_RESP_SOCKET    (0x02)    // response read only socket
```

ホスト プロセッサはファームウェア ダウンロードを実行するために、これらのデータ ソケットを介して FX3 ブートローダと通信します。コマンドと応答は、ファームウェア ダウンロード プロトコルに使用されるデータ構造体です。両方のサイズは 512 バイトです。ビット フィールドは、FX3 ブートローダによってさまざまな機能を実行するためにこれらのデータ構造で定義されます。本書で提供される簡単な実装例では、コマンドと応答両方の最初の 4 バイトのみは実際に使用されます。コマンドと応答の残りのデータ バイトはドント ケアです。

高レベルの FX3 ファームウェアの視点から、ダウンロードにはホスト プロセッサが以下のソケット アクセス シーケンスを実行する必要があります。

1. 以下のように初期化されたコマンド ブロックでのコマンド ソケット書き込みます。

```
command[0] = 'C';
command[1] = 'Y';        /* first two bytes are signature bytes with
                            constant value of "CY" */
command[2] = 0x02;       /* 0x2 is value for boot mode command. */
command[3] = 0x01;       /* 1 data block */
```

2. 以下のように応答ブロック データを予想する応答ソケット読み出します。

```
response[0] = 'W';
response[1] = 'B';        /* first two bytes are signature bytes with
                             constant value of "WB" */
//response[2] = 0x0;      /* this byte is don't care. */
response[3] = 0x0;        /* indicate command is accepted */
```

3. バイト アレイに関連するファームウェア イメージ全体を FX3 に転送するデータ ソケット書き込みます。

ファームウェア イメージが完全に転送されると、FX3 ブートローダは新たにダウンロードされたファームウェアのエントリ ポイントに自動的にジャンプし、実行を開始することに注意してください。ホスト プロセスがダウンロードされたファームウェアと通信できる前に、ファームウェアが完全に初期化されるまで一定の時間 (ファームウェアの実装による) 待つことを推奨します。それより良いオプションは、初期化後にメールボックス レジスタを介してステータス更新をファームウェアに実装することです。この場合には、ファームウェア側の準備ができるとホスト プロセッサは直ちに通知されます。

9.1.8 プロセッサ ポート (P-Port) レジスタ マップ

Table 28 に示すレジスタ リストは、PP_xxx レジスタが外部 P-Port アドレス空間でどのようにマッピングされるかを示します。この空間でのアドレスは、バイトではなく、ワード アドレスを示します。同期 ADMux インタフェースは、これらのレジスタにアクセスするために 8 つのアドレス ラインを提供します。

Table 28 プロセッサ ポート レジスタ マップ

レジスタ名	アドレス	幅 (ビット)	説明
PP_ID	0x80	16	P-Port デバイス ID レジスタ: デバイス ID 情報を提供
PP_INIT	0x81	16	P-Port リセットおよび電源制御: このレジスタはリセットおよび電源制御に使用され、P-Port のエンディアン タイプを指定
PP_CONFIG	0x82	16	P-Port コンフィギュレーション レジスタ

同期 ADMux ブート

レジスタ名	アドレス	幅 (ビット)	説明
PP_IDENTIFY	0x83	16	P-Port 識別レジスタ: このレジスタの下位 8 ビットは読み出し専用で、初期設定値は 0x81
PP_INTR_MASK	0x88	16	P-Port 割込みマスク レジスタ: このレジスタは、PP_EVENT との同じレイアウトを持ち、割込みのアサートを引き起こすイベントをマスキング
PP_DRQR5_MASK	0x89	16	P-Port DRQ/R5 マスク レジスタ: このレジスタは、PP_EVENT との同じレイアウトを持ち、割込みまたは DRQ/R5 のアサートを引き起こすイベントをマスキング
PP_ERROR	0x8C	16	P-Port エラー表示レジスタ
PP_DMA_XFER	0x8E	16	P-Port DMA 転送レジスタ: このレジスタは DMA 転送を設定し制御するために使用
PP_DMA_SIZE	0x8F	16	P-Port DMA 転送サイズ レジスタ: このレジスタは転送の (残りの) サイズを示す
PP_WR_MAILBOX	0x90	64	P-Port 書き込みメールボックス レジスタ: これらのレジスタは、AP から FX3 ファームウェアへの最大 8 バイトのメッセージを含む
PP_MMIO_ADDR	0x94	32	P-Port MMIO アドレス レジスタ: これらのレジスタは一緒に、FX3 内部 MMIO 空間にアクセスするための 32 ビット アドレスを形成
PP_MMIO_DATA	0x96	32	P-Port MMIO データ レジスタ: これらのレジスタは一緒に、FX3 内部 MMIO 空間にアクセスするための 32 ビット データを形成
PP_MMIO	0x98	16	P-Port MMIO 制御レジスタ: このレジスタは FX3 MMIO 空間へのアクセスを制御
PP_EVENT	0x99	16	P-Port イベント レジスタ: このレジスタは割込みまたは DRQ のアサートを引き起こすあらゆるイベントの種類を示す
PP_RD_MAILBOX	0x9A	64	P-Port 読み出しメールボックス レジスタ: これらのレジスタは FX3 ファームウェアから AP への最大 8 バイトのメッセージを含む
PP_SOCK_STAT	0x9E	32	P-Port ソケット ステータス レジスタ: これらのレジスタでは、P-port の 32 個のソケットごとに 1 ビットが用意される。このビットは各ソケットのバッファの可用性を示す

これらのレジスタのビット フィールドの定義については、**EZ-USB™ FX3 TRM** のレジスタの章を参照してください。

FX3 ファームウェア ダウンロードの詳しく調べる前に、次の関数が、本書の実装例によく使用され、プラットフォームに依存します。これらの関数を特定のプラットフォームで実装する方法については、**サイプレス サポート** までお問い合わせください。

```
IORD_REG16(); // 16-bit read from GPIF II
IOWR_REG16(); // 16-bit write to GPIF II
IORD_SCK16(); // 16-bit read from active socket set in PP_DMA_XFER. The address
driven on
              //   on the Sync ADMux bus during the address phase is treated as a
```

同期 ADMux ブート

```
                //   don't-care
IOWR_SCK16(); // 16-bit write to active socket set in PP_DMA_XFER. The address driven
on
                //   on the Sync ADMux bus during the address phase is treated as a
                //   don't-care
```

Note: レジスタ アクセスの実行中、8 ビット アドレスの最上位ビットは、レジスタ アクセス動作である旨を FX3 に通知するために 1 にセットしなければなりません。同様に、ソケット アクセスの実行には、最上位ビットを 0 にセットする必要があります。

```
mdelay();      // millisecond delay
udelay();      // microsecond delay
```

次は、ファームウェア データ アレイへのポインタとファームウェアのサイズをパラメーターとする fx3_firmware_download() 関数の実装例です。

```
/* Register addresses and the constants used in the code shown below. */
#define CY_WB_DOWNLOAD_CMD_SOCKET   0x00      // command block write only socket
#define CY_WB_DOWNLOAD_DATA_SOCKET  0x01      // data block read/write socket
#define CY_WB_DOWNLOAD_RESP_SOCKET  0x02      // response read only socket

// All register addresses defined with bit 7 set to indicate Register access (not
Socket)
#define PP_CONFIG             0x82
   #define CFGMODE            0x0040



int fx3_firmware_download(const u8 *fw, u16 sz)
{
        u8  *command=0, *response=0;
        u16 val;
        u32 blkcnt;
        u16 *p = (u16 *)fw;
        int i=0;

        printf("FX3 Firmware Download with size = 0x%x\n", sz);

        /* Check PP_CONFIG register and make sure FX3 device is configured */
        /*   When FX3 bootloader is up with correct PMODE, bootloader configures */
        /*   the GPIF II into proper interface and sets the CFGMODE bit on PP_CONFIG
*/
        val = IORD_REG16(PP_CONFIG);
        if ((val & CFGMODE)== 0) {
                printf("ERROR: WB Device CFGMODE not set !!! PP_CONFIG=0x%x\n", val);
                return FAIL;
        }

        /* A good practice to check for size of image */
        if (sz > (512*1024)) {
```

同期 ADMux ブート

```
                printf("ERROR: FW size larger than 512kB !!!\n");
                return FAIL;
        }

        /* Allocate memory for command and response */
        /* Host processor may use DMA sequence to transfer the command and response
*/
        /* In that case make sure system is allocating contiguous physical memory
area */
        command  = (u8 *) malloc(512);
        response = (u8 *) malloc(512);
        memset(command, 0, 512);
        memset(response, 0, 512);

        if (command==0 || response==0) {
                printf("ERROR: Out of memory !!!\n");
                return FAIL;
        }

        /* Initialize the command block */
        command[0] = 'C';
        command[1] = 'Y';
        command[2] = 0x02;      /* Enter boot mode command. */
        command[3] = 0x01;      /* 1 data block */

        /* Print the command block if you like to see it */
        for (i=0; i<512; i++) {
           if (!(i%16))
              printf("\n%.3x: ", i);
           printf("%.2x ",command[i]);
        }
        printf("\n");

     /* write boot command into command socket */
        sck_bootloader_write(CY_WB_DOWNLOAD_CMD_SOCKET, 512, (u16 *)command);

        /* read the response from response socket */
    sck_bootloader_read(CY_WB_DOWNLOAD_RESP_SOCKET, 512, (u16 *)response);

    /* Check if correct response */
        if ( response[3]!=0 || response[0]!='W' || response[1]!='B' ) {
                printf("ERROR: Incorrect bootloader response =
0x%x !!!\n",response[3]);
                for (i=0; i<512; i++) {
                        if (!(i%16))
                                printf("\n%.3x: ", i);
                        printf("%.2x ",response[i]);
                }
               printf("\n");
                kfree(command);
                kfree(response);
```

同期 ADMux ブート

```
                return FAIL;
        }

        /* Firmware image transfer must be multiple of 512 byte */
        /* Here it rounds up the firmware image size */
        /* and write the array to data socket */
        blkcnt = (sz+511)/512;
        sck_bootloader_write(CY_WB_DOWNLOAD_DATA_SOCKET, blkcnt*512, p);

        /* Once the transfer is completed, bootloader automatically jumps to */
        /* entry point of the new firmware image and start executing */

        kfree(command);
        kfree(response);
        mdelay(2);        /* let the new image come up */
        return PASS;
}
```

次は、ソケット書き込みとソケット読み出し関数の実装例です。データ方向の他に、ソケット書き込みおよびソケット読み出しの関数実装は、以下の PP_*レジスタ インタフェース上のコマンド, コンフィギュレーション, およびステータス ビットに基づきます。

- PP_SOCK_STAT.SOCK_STAT[N]: このステータス ビットはソケットごとに、データを交換するためにソケットが利用可能なバッファがある (利用可能なデータまたは空間を持つ) ことを示します。
- PP_DMA_XFER.DMA_READY: このステータス ビットは、GPIF II が、アクティブなソケットとの読み書きを実行できるかを示します (アクティブなソケットは PP_DMA_XFER レジスタで選択されます)。PP_EVENT.DMA_READY_EV は、僅か数サイクル遅延されて PP_DMA_XFER.DMA_READY を反映します。
- PP_EVENT.DMA_WMARK_EV: このステータス ビットは DMA_READY と同様ですが、現時点のバッファが完全に交換される前に、プログラム可能なワード数をアサート解除します。このビットは信号伝達インタフェースでオフセット レイテンシのあるフロー制御信号を作成するために使用できます。
- PP_DMA_XFER.LONG_TRANSFER: この設定ビットは、長い転送 (マルチバッファ) が有効にされたかを示します。このビットは、転送開始の一部としてアプリケーション プロセッサによって設定されます。
- PP_CONFIG.BURSTSIZE と PP_CONFIG.DRQMODE: これらの設定ビットは DMA バーストのサイズを定義し有効にします。PP_CONFIG レジスタの更新が成功すると、FX3 ブートローダは PP_RD_MAILBOX レジスタで 0x33 の値で応答します。
- PP_DMA_XFER.DMA_ENABLE: このコマンドとステータスは DMA 転送が有効であることを示します。このビットは、ホスト プロセッサによって転送開始の一部としてセットされ、転送終了時に、FX3 ハードウェア (短い転送の場合) またはアプリケーション プロセッサ (長い転送の場合) によってクリアされます。

```
/* Register addresses and the constants used in the code shown below. */
#define PP_CONFIG             0x82
   #define CFGMODE            0x0040

#define PP_DRQR5_MASK         0x89
   #define DMA_WMARK_EV       0x0800

#define PP_DMA_XFER           0x8E
   #define LONG_TRANSFER      0x0400
   #define DMA_DIRECTION      0x0200
```

同期 ADMux ブート

```
   #define DMA_ENABLE          0x0100
#define PP_EVENT               0x99
   #define DMA_READY_EV        0x1000

#define PP_RD_MAILBOX0         0x9A     // 64 Bit register accessed as 4 x 16 bit
registers
#define PP_RD_MAILBOX1         0x9B
#define PP_RD_MAILBOX2         0x9C
#define PP_RD_MAILBOX3         0x9D

#define PP_SOCK_STAT_L         0x9E     // LSB 16 bits of 32 bit register
#define PP_SOCK_STAT_H         0x9F     // MSB 16 bits of 32 bit register

static u32 sck_bootloader_write(u8 sck, u32 sz, u16 *p)
{

   u32 count;
   u16 val, buf_sz;
   int i;

        buf_sz = 512;
        /* Poll for PP_SOCK_STAT_L and make sure socket status is ready */
        do {
                val = IORD_REG16(PP_SOCK_STAT_L);
                udelay(10);
        } while(!(val&(0x1<<sck)));

        /* write to pp_dma_xfer to configure transfer
               socket number, rd/wr operation, and long/short xfer modes */
        val = (DMA_ENABLE | DMA_DIRECTION | LONG_TRANSFER | sck);
        IOWR_REG16(PP_DMA_XFER, val);

        /* Poll for DMA_READY_EV */
        count = 10000;
        do {
                val = IORD_REG16(PP_EVENT);
                udelay(10);
                count--;
        } while ((!(val & DMA_READY_EV)) && (count != 0));

        if (count == 0) {
                printf("%s: Fail timeout; Count = 0\n", __func__);
                return FAIL;
        }

        /* enable DRQ WMARK_EV for DRQ assert */
        IOWR_REG16(PP_DRQR5_MASK, DMA_WMARK_EV);

        /* Change FX3 FW to single cycle mode */
        val = IORD_REG16(PP_CONFIG);
        val = (val&0xFFF0)|CFGMODE;
```

同期 ADMux ブート

```
        IOWR_REG16(PP_CONFIG, val);

        /* Poll for FX3 FW config init ready */
        count = 10000;
        do {
                val = IORD_REG16 (PP_RD_MAILBOX2);
                udelay(10);
              count --;
        } while ((!(val & 0x33)) || count==0);  /* CFGMODE bit is cleared by FW */

        if (count == 0) {
                printk("%s: Fail timeout; Count = 0\n", __func__);
                return FAIL;
        }

       count=0;
       do {
              for (i = 0; i < (buf_sz / 2); i++)
                     IOWR_SCK16(*p++); /* Write 512 bytes of data continuously to
data socket 16 bits at a time ( Sync ADMux has 16 data lines) */
              count += (buf_sz / 2);

              if (count < (sz/2))
                     do {
                            udelay(10);
                            val = IORD_REG16 (PP_SOCK_STAT_L); /* After writing 512
bytes to data socket of the device, P-Port Socket Status Register is read to check if
the Socket is available for reading or writing next set of 512 bytes data */
                     } while(!(val&(0x1<<sck)));/* You remain in this Do-while loop
till PP_SOCK_STAT_L register makes the bit corresponding to the socket as 1
indicating socket is now available for next read/write */

       } while (count < (sz/2)); /* sz is the total size of data to be written. In
case of firmware_download, sz will be total size of the firmware */

        /* disable dma */
        val = IORD_REG16(PP_DMA_XFER);
        val &= (~DMA_ENABLE);
        IOWR_REG16(PP_DMA_XFER, val);

        printf("DMA write completed .....\n");
       return PASS;
}

static u32 sck_bootloader_read(u8 sck, u32 sz, u16 *p)
{

   u32 count;
   u16 val, buf_sz;
   int i;
        buf_sz = 512;
```

同期 ADMux ブート

```
        /* Poll for PP_SOCK_STAT_L and make sure socket status is ready */
        do {
                val = IORD_REG16(PP_SOCK_STAT_L);
                udelay(10);
        } while(!(val&(0x1<<sck)));

        /* write to PP_DMA_XFER to configure transfer
socket number, rd/wr operation, and long/short xfer modes */
        val = (DMA_ENABLE | LONG_TRANSFER | sck);
        IOWR_REG16(PP_DMA_XFER, val);

        /* Poll for DMA_READY_EV */
        count = 10000;
        do {
                val = IORD_REG16 (PP_EVENT);
                udelay(10);
                count--;
        } while ((!(val & DMA_READY_EV)) && (count != 0));

        if (count == 0) {
                printk("%s: Fail timeout; Count = 0\n", __func__);
                return FAIL;
        }

        /* enable DRQ WMARK_EV for DRQ assert */
        IOWR_REG16(PP_DRQR5_MASK, DMA_WMARK_EV);

        /* Change FX3 FW to single cycle mode */
        val = IORD_REG16(PP_CONFIG);
        val = (val&0xFFF0)|CFGMODE;
        IOWR_REG16(PP_CONFIG, val);

        /* Poll for FX3 FW config init ready */
        count = 10000;
        do {
                val = IORD_REG16 (PP_RD_MAILBOX2);
                udelay(10);
               count --;
        } while ((!(val & 0x33)) || count==0);  /* CFGMODE bit is cleared by FW */

        if (count == 0) {
                printk("%s: Fail timeout; Count = 0\n", __func__);
                return -1;
        }

       count=0;
       do {
              for (i = 0; i < (buf_sz / 2); i++) {
                     p[count+i] = IORD_SCK16();
              }
              count += (buf_sz / 2);  /* count in words */
```

```
            if (count < (sz/2))
            do {
                    udelay(10);
                    val = IORD_REG16 (PP_SOCK_STAT_L);
            } while(!(val&(0x1<<sck)));

        } while (count < (sz/2));

         /* disable dma */
         val = IORD_REG16(PP_DMA_XFER);
         val &= (~DMA_ENABLE);
         IOWR_REG16(PP_DMA_XFER, val);

         printf("DMA read completed .....\n");
        return PASS;
}
```

同期 ADMux ブートのトラブルシューティングについては、**付録 B: 同期 ADMux ブートのトラブルシューティング手順**を参照してください。

9.2 ブート イメージ フォーマット

同期 ADMux ブートでは、ブートローダはファームウェア イメージが **Table 29** に示すフォーマットであると予想しています。**EZ-USB™ FX3 SDK** は、同期 ADMux ブートに必要なフォーマットでファームウェア イメージを生成するためのソフトウェア ユーティリティを提供します。SDK をインストールした後、*C:\Program Files\Cypress\EZ-USB FX3SDK\1.3\util\elf2img* ディレクトリにある elf2img ユーティリティを参照してください。64 ビット システムでは、パスにある最初のフォルダは Program Files (x86) です。ディレクトリ パスにある *1.3* 番号は SDK のバージョン番号ですが、FX3 SDK の最新リリースによって異なります。

elf2img ビルド後のコマンドが*.img* ファイルを作成することに注意してください。その後、このファイルは、以前に示したダウンロード例に使用できるアレイに変換する必要があります。**Figure 11** は、elf2img ビルド後のコマンドの発行方法を示し、*.img* ファイルの内容を ASCII フォーマットでのアレイに変換する例も提供します。

Table 29 同期 ADMux ブート オプションのブート イメージ フォーマット

2 進数イメージ ヘッダ	長さ (16 ビット)	説明
wSignature	1	ASCII 表記の「CY」の初期値を持つ 2 バイトのシグネチャ
bImageCTL;	½	ビット 0 = 0:2 進数実行ファイル；1:データ ファイル タイプ ビット 3〜1:SPI EEPROM でブートする場合は使用禁止 ビット 5〜4 (SPI 速度): 00: 10 MHz 01: 20 MHz 10: 30 MHz 11: 予約済み ビット 7〜6: 予約済み、「0」にセットする必要がある

同期 ADMux ブート

2 進数イメージ ヘッダ	長さ (16 ビット)	説明
bImageType;	½	bImageType = 0xB0: チェックサムを含む普通の 2 進数ファームウェア イメージ bImageType = 0xB2: 新規 VID および PID による SPI ブート
dLength 0	2	ロング ワード (32 ビット) での最初のセクションの長さ bImageType = 0xB2 の場合、dLength 0 は PID と VID を含む。ブートローダは残りの後続データを無視
dAddress 0	2	プログラム コードの最初のセクション アドレス *Note: 内部 ARM アドレスはバイト アドレス指定が可能で、各セクションのアドレスは 32 ビット アラインである必要がある*
dData[dLength 0]	dLength 0*2	イメージ コード/データは 32 ビット アラインである必要がある
…		他のセクション
dLength N	2	0x00000000 (最後の記録:終端セクション)
dAddress N	2	有効なプログラム エントリ (通常はスタートアップ コード、例えば RESET ベクタ) を含む必要がある *Note: bImageCTL.bit0 = 1 の場合、ブートローダはこのプログラム エントリに実行を転送しない* *bImageCTL.bit0 = 0 の場合、ブートローダはこのプログラム エントリに実行を転送。このアドレスは ITCM 空間または SYSTEM RAM 空間にある必要がある。ブートローダはプログラム エントリを確認しない*
dCheckSum	2	32 ビット符号なしのリトルエンディアン チェックサム データは最初のセクションから終端セクションまで。チェックサムは dLength、dAddress、およびイメージ ヘッダを含まない

次は、ロング ワード フォーマットで構成されるブート イメージ フォーマット例です。

```
Location1: 0xB0 0x10 'Y' 'C' //CY Signature, 20 MHz, 0xB0 Image
Location2: 0x00000004        //Image length of section 1 = 4
Location3: 0x40008000        //1st section stored in SYSMEM RAM at 0x40008000
Location4: 0x12345678        //Image starts (Section1)
Location5: 0x9ABCDEF1
Location6: 0x23456789
Location7: 0xABCDEF12        //Section 1 ends
Location8: 0x00000002        //Image length of section 2 = 2
Location9: 0x40009000        //2nd section stored in SYSMEM RAM at 0x40009000
Location10: 0xDDCCBBAA       //Section 2 starts
Location11: 0x11223344
Location12: 0x00000000       //Termination of Image
Location13: 0x40008000       //Jump to 0x40008000 on FX3 System RAM
Location 14: 0x6AF37AF2      //Checksum (0x12345678 + 0x9ABCDEF1 + 0x23456789 +
                               0xABCDEF12 + 0xDDCCBBAA + 0x11223344)
```

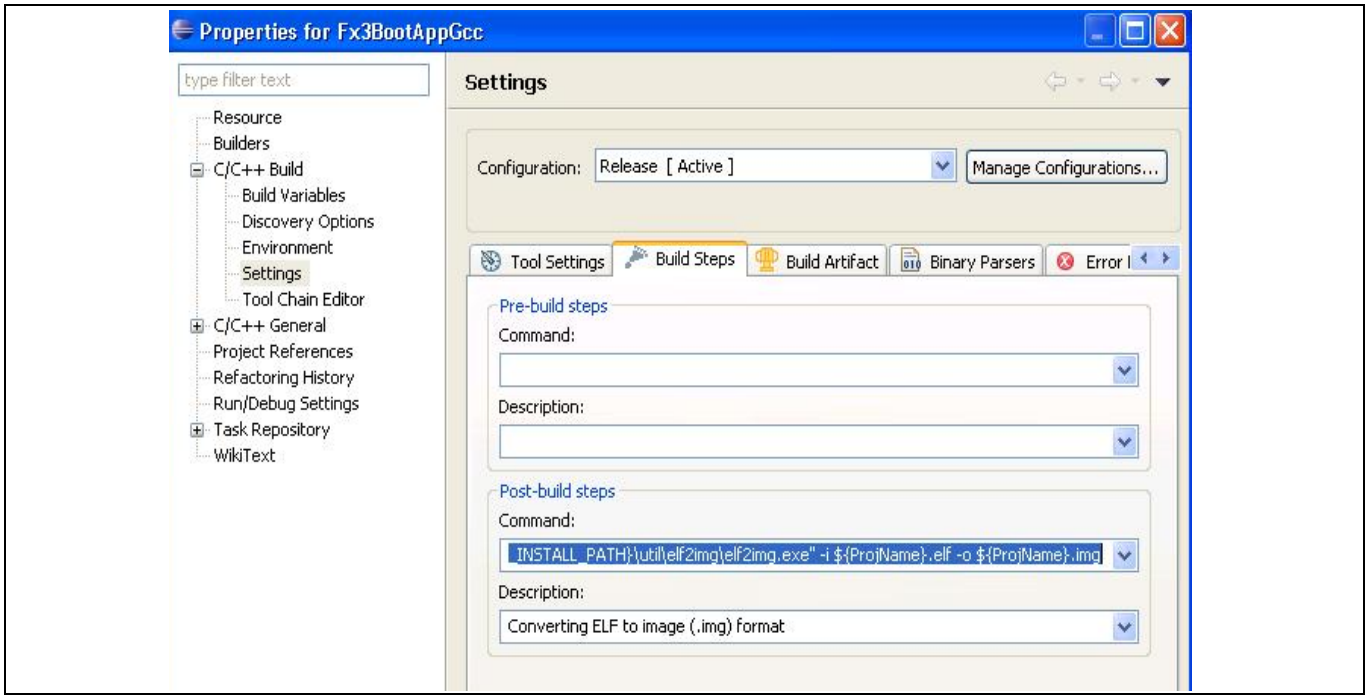

Figure 11　　Eclipse IDE におけるビルド後のコマンド

次は、*.img* ファイルの内容を ASCII フォーマットでのアレイに変換するサンプル コードです。

```
#include <stdio.h>
#include <stdint.h>
int main (int argc, char *argv[])
{
        char *filename = "firmware.img";
FILE *fp;
        int i = 0;
        uint32_t k;

        if (argc > 1)
                filename = argv[1];

        fprintf (stderr, "Opening file %s\n", filename);
        fp =  fopen (filename, "r");
        printf ("const uint8_t fw_data[] = {\n\t");

        while (!feof(fp))
        {
                fread (&k, sizeof (uint32_t), 1, fp);
                printf ("0x%02x, 0x%02x, 0x%02x, 0x%02x,",
                                ((uint8_t *)&k)[0], ((uint8_t *)&k)[1],
                                ((uint8_t *)&k)[2], ((uint8_t *)&k)[3]);
                i++;
                if (i == 4)
                {
                        i = 0;
                        printf ("\n\t");
```

同期 ADMux ブート

```
                }
                else
                        printf (" ");
        }

        printf ("\n};\n");

        fclose (fp);
        return 0;
        }
```

10 ブート時の I/O のデフォルト状態

Table 30 に、ブートローダが実行する中に (アプリケーション ファームウェア ダウンロード前に) 異なるブート モードで FX3 I/O のデフォルト状態を示します。

Note: GPIO のデフォルト状態は、FX3 リセット時およびブートローダの設定後の状態と異なる必要はありません。

Table 30 ブート時の I/O のデフォルト状態

GPIO	SPI ブート時のデフォルト状態	USB ブート時のデフォルト状態	I^2C ブート時のデフォルト状態	同期 ADMux ブート時のデフォルト状態
GPIO[0]	トライステート	トライステート	トライステート	トライステート
GPIO[1]	トライステート	トライステート	トライステート	トライステート
GPIO[2]	トライステート	トライステート	トライステート	トライステート
GPIO[3]	トライステート	トライステート	トライステート	トライステート
GPIO[4]	トライステート	トライステート	トライステート	トライステート
GPIO[5]	トライステート	トライステート	トライステート	トライステート
GPIO[6]	トライステート	トライステート	トライステート	トライステート
GPIO[7]	トライステート	トライステート	トライステート	トライステート
GPIO[8]	トライステート	トライステート	トライステート	トライステート
GPIO[9]	トライステート	トライステート	トライステート	トライステート
GPIO[10]	トライステート	トライステート	トライステート	トライステート
GPIO[11]	トライステート	トライステート	トライステート	トライステート
GPIO[12]	トライステート	トライステート	トライステート	トライステート
GPIO[13]	トライステート	トライステート	トライステート	トライステート
GPIO[14]	トライステート	トライステート	トライステート	トライステート
GPIO[15]	トライステート	トライステート	トライステート	トライステート
GPIO[16]	トライステート	トライステート	トライステート	CLK 入力
GPIO[17]	トライステート	トライステート	トライステート	入力
GPIO[18]	トライステート	トライステート	トライステート	入力
GPIO[19]	トライステート	トライステート	トライステート	入力
GPIO[20]	トライステート	トライステート	トライステート	入力
GPIO[21]	トライステート	トライステート	トライステート	出力
GPIO[22]	トライステート	トライステート	トライステート	トライステート
GPIO[23]	トライステート	トライステート	トライステート	入力
GPIO[24]	トライステート	トライステート	トライステート	トライステート
GPIO[25]	トライステート	トライステート	トライステート	トライステート
GPIO[26]	トライステート	トライステート	トライステート	トライステート
GPIO[27]	トライステート	トライステート	トライステート	入力
GPIO[28]	トライステート	トライステート	トライステート	トライステート
GPIO[29]	トライステート	トライステート	トライステート	トライステート

ブート時の I/O のデフォルト状態

GPIO	SPI ブート時のデフォルト状態	USB ブート時のデフォルト状態	I²C ブート時のデフォルト状態	同期 ADMux ブート時のデフォルト状態
GPIO[30]	FX3 の PMODE[0]に接続	FX3 の PMODE[0]に接続	FX3 の PMODE[0]に接続	FX3 の PMODE[0]に接続
GPIO[31]	FX3 の PMODE[1]に接続	FX3 の PMODE[1]に接続	FX3 の PMODE[1]に接続	FX3 の PMODE[1]に接続
GPIO[32]	FX3 の PMODE[2]に接続	FX3 の PMODE[2]に接続	FX3 の PMODE[2]に接続	FX3 の PMODE[2]に接続
GPIO[33]	トライステート	トライステート	トライステート	トライステート
GPIO[34]	トライステート	トライステート	トライステート	トライステート
GPIO[35]	トライステート	トライステート	トライステート	トライステート
GPIO[36]	トライステート	トライステート	トライステート	トライステート
GPIO[37]	トライステート	トライステート	トライステート	トライステート
GPIO[38]	トライステート	トライステート	トライステート	トライステート
GPIO[39]	トライステート	トライステート	トライステート	トライステート
GPIO[40]	トライステート	トライステート	トライステート	トライステート
GPIO[41]	トライステート	トライステート	トライステート	トライステート
GPIO[42]	LOW	LOW	LOW	LOW
GPIO[43]	トライステート	トライステート	トライステート	トライステート
GPIO[44]	トライステート	トライステート	トライステート	トライステート
GPIO[45]	トライステート (SPI ブートができない場合は HIGH)	HIGH	トライステート	HIGH
GPIO[46]	HIGH	トライステート	トライステート	トライステート
GPIO[47]	トライステート	トライステート	トライステート	トライステート
GPIO[48]	HIGH	トライステート	トライステート	トライステート
GPIO[49]	トライステート	トライステート	トライステート	トライステート
GPIO[50]	トライステート (SPI ブートができない場合は LOW)	トライステート	トライステート	トライステート
GPIO[51]	LOW	LOW	LOW	LOW
GPIO[52]	HIGH	トライステート	トライステート	トライステート
GPIO[53]	LOW (SPI トランザクション中にトグル)	HIGH	トライステート	HIGH
GPIO[54]	HIGH	トライステート	トライステート	トライステート
GPIO[55]	トライステート	HIGH	HIGH	HIGH
GPIO[56]	LOW	トライステート	トライステート	トライステート
GPIO[57]	LOW	トライステート	トライステート	トライステート

ブート時の I/O のデフォルト状態

GPIO	SPI ブート時のデフォルト状態	USB ブート時のデフォルト状態	I^2C ブート時のデフォルト状態	同期 ADMux ブート時のデフォルト状態
GPIO[58] I2C_SCL	トライステート	トライステート	トライステート (SPI トランザクション中にトグル、その後はトライステート)	トライステート
GPIO[59] I2C_SDA	トライステート	トライステート	トライステート	トライステート

11 付録 A: FX3 DVK 基板 (CYUSB3KIT-001) を用いるブート手順

ここでは、FX3 DVK 基板を使用して USB ブート, I^2C ブート, および SPI ブートを実行する段階的な手順を説明します。**Figure 12** に、それぞれのブート オプションに適切に設定する必要があるスイッチとジャンパを含む FX3 DVK 基板の一部を示します。それらのスイッチとジャンパに必要な設定も説明します。

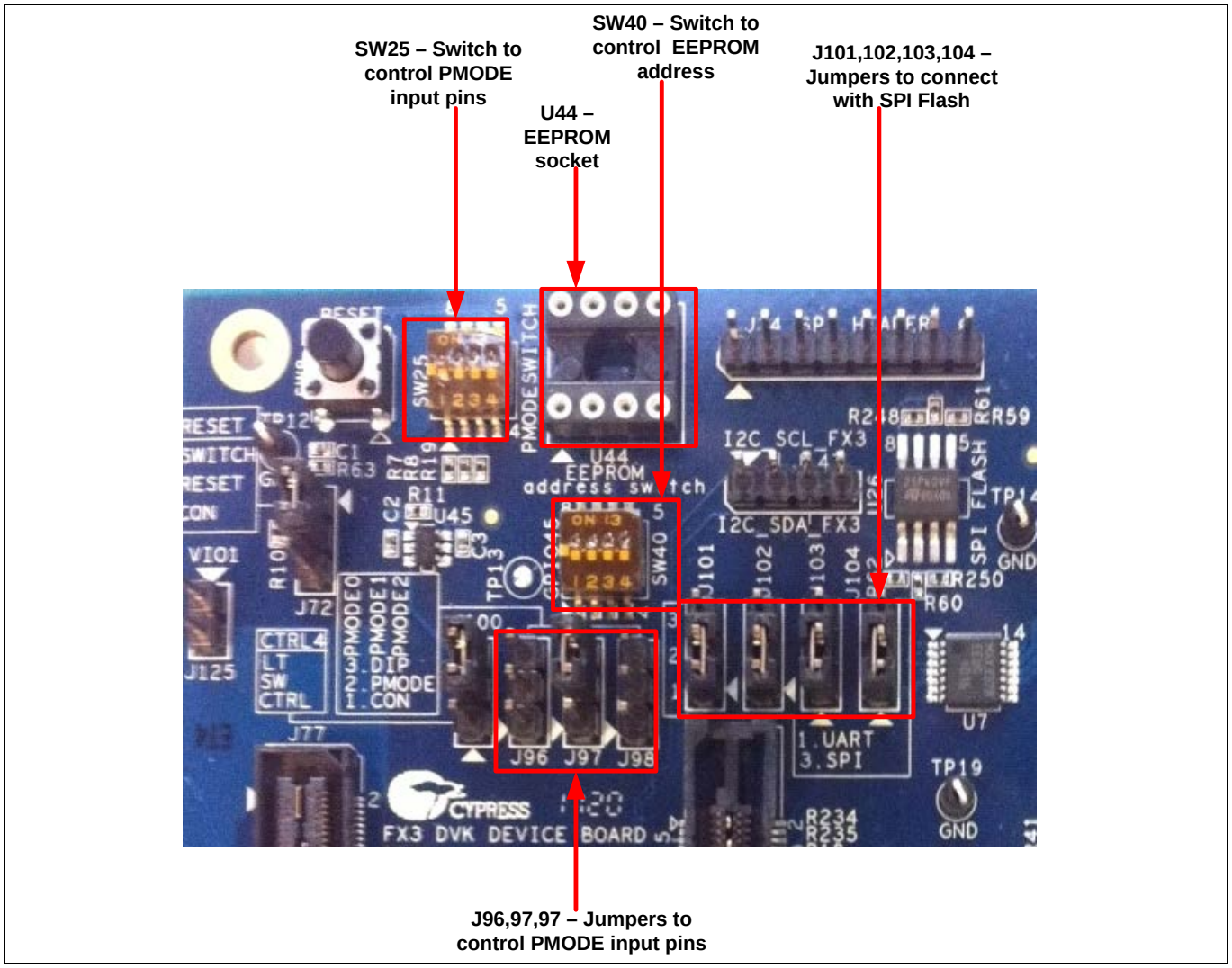

Figure 12 FX3 DVK 基板: ブート用に設定する重要なスイッチとジャンパ

11.1 USB ブート

1. **Figure 13**, **Figure 14**, および **Figure 15** に示すように、Eclipse IDE でファームウェア イメージをビルドしてください。

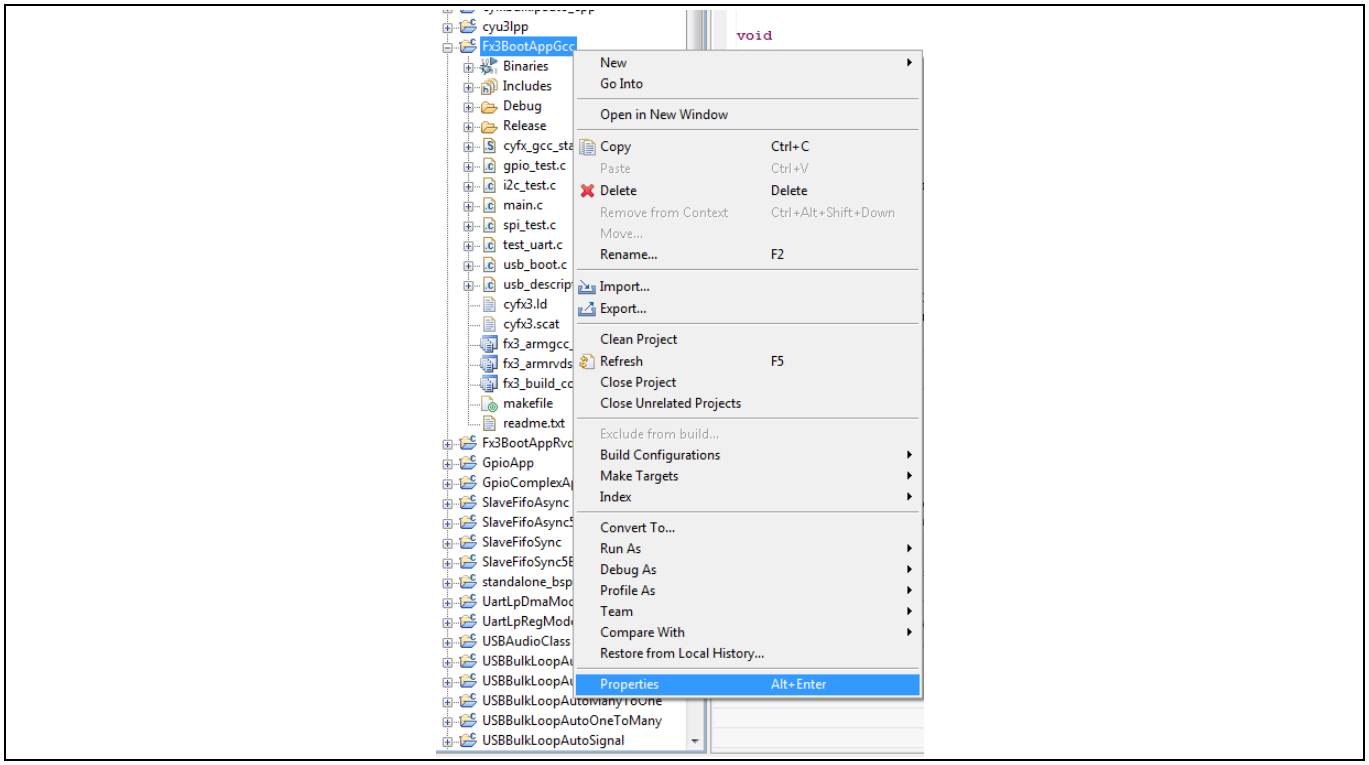

Figure 13 Eclipse IDE でプロジェクトを右クリック

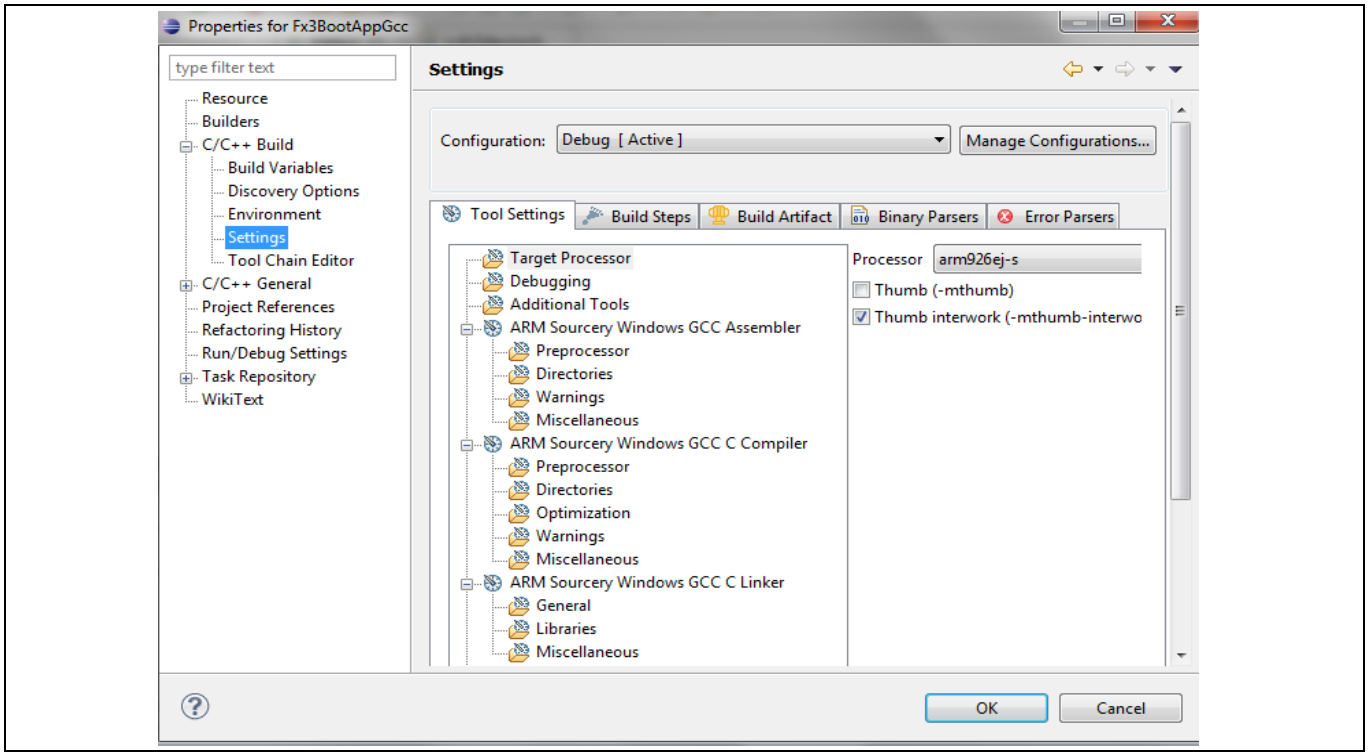

Figure 14 「Settings」を選択

付録 A: FX3 DVK 基板 (CYUSB3KIT-001) を用いるブート手順

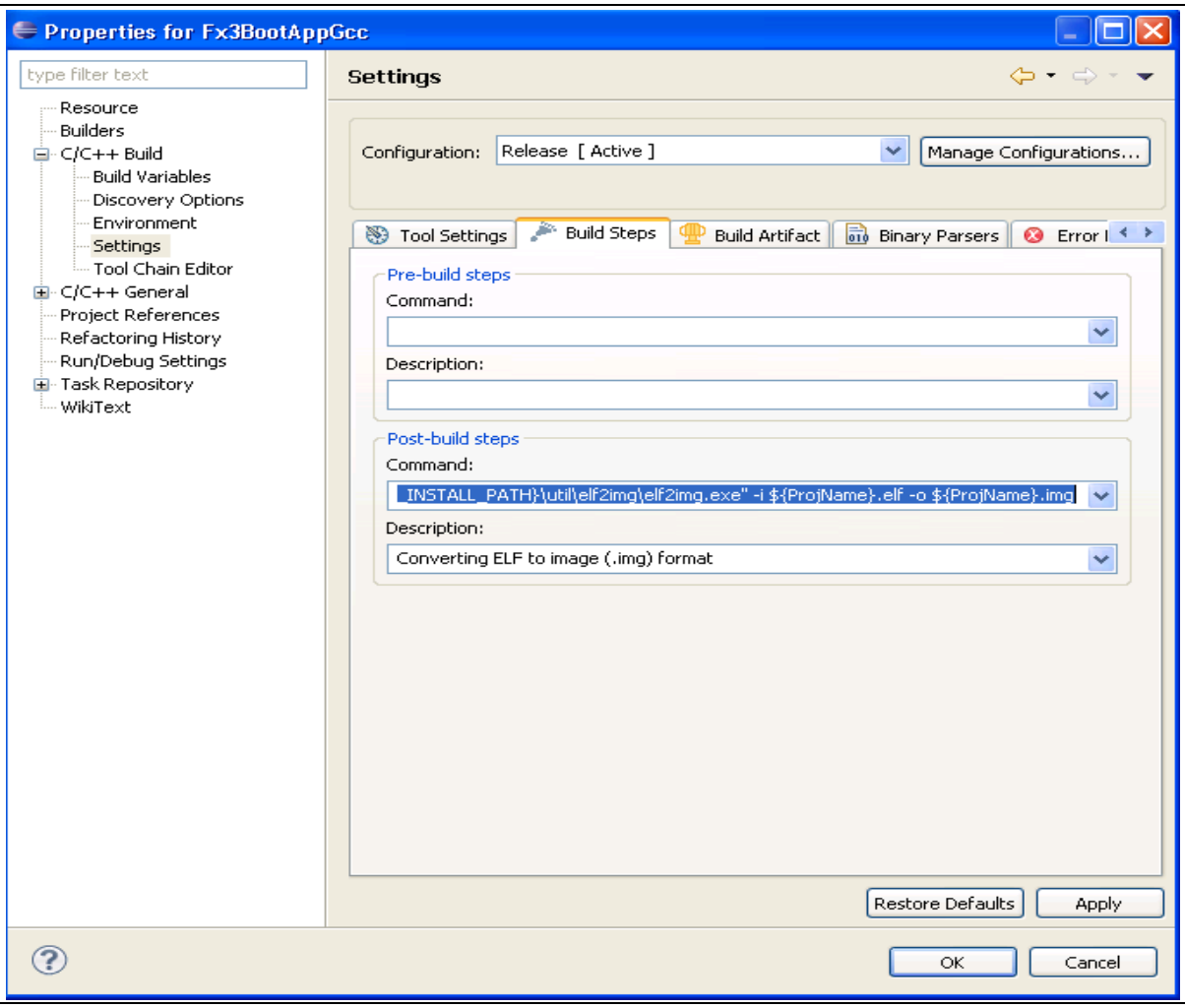

Figure 15 USB ブート イメージ用の elf2img コマンド コンフィギュレーション (ビルド後ステップで)

2. PMODE[2:0]ピンを Z11 に設定することで USB ブートを有効にしてください。DVK 基板では、これは **Table 31** に示すようにジャンパとスイッチを設定することで実現されます。

Table 31 USB ブートのジャンパ設定

ジャンパ/スイッチ	位置	該当する PMODE ピンの状態
J96 (PMODE0)	2-3 投入	PMODE0:SW25 で制御
J97 (PMODE1)	2-3 投入	PMODE1:SW25 で制御
J98 (PMODE2)	開放	PMODE2:未接続
SW25.1-8 (PMODE0)	開放 (OFF 位置)	PMODE0 = 1
SW25.2-7 (PMODE1)	開放 (OFF 位置)	PMODE1 = 1
SW25.3-6 (PMODE2)	ドント ケア	PMODE2:未接続

付録 A: FX3 DVK 基板 (CYUSB3KIT-001) を用いるブート手順

3. USB ホストに接続した時、FX3 デバイスは **Figure 16** に示すように、コントロール センター内で「Cypress USB BootLoader」 (サイプレス USB ブートローダ) としてエニュメレートします。

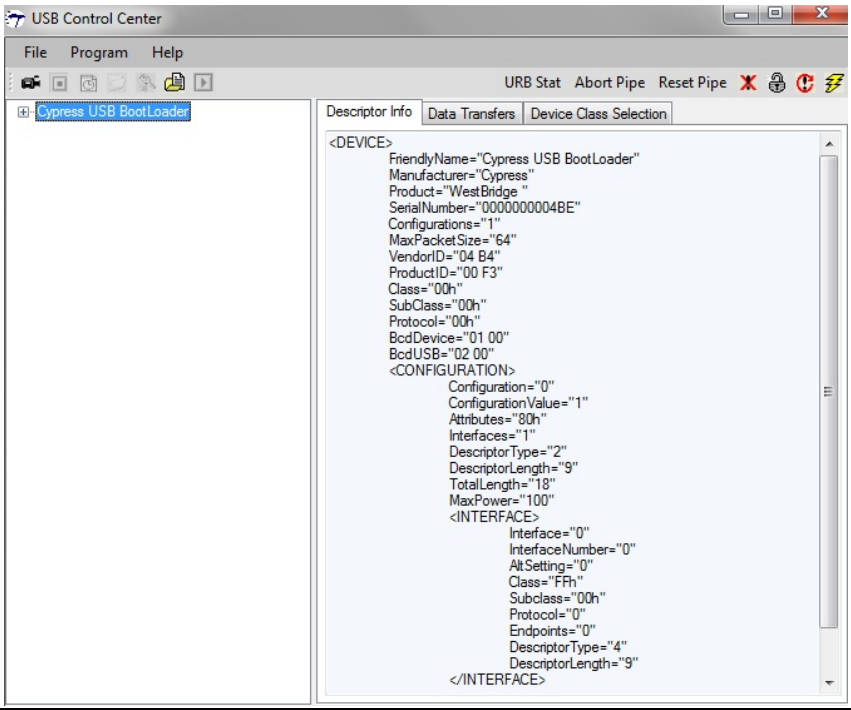

Figure 16　　コントロール センターでのサイプレス USB ブートローダ エニュメレーション

4. コントロール センター内で、**Figure 17** に示すように **Program > FX3 > RAM** を選択することで FX3 デバイスを選択してください。

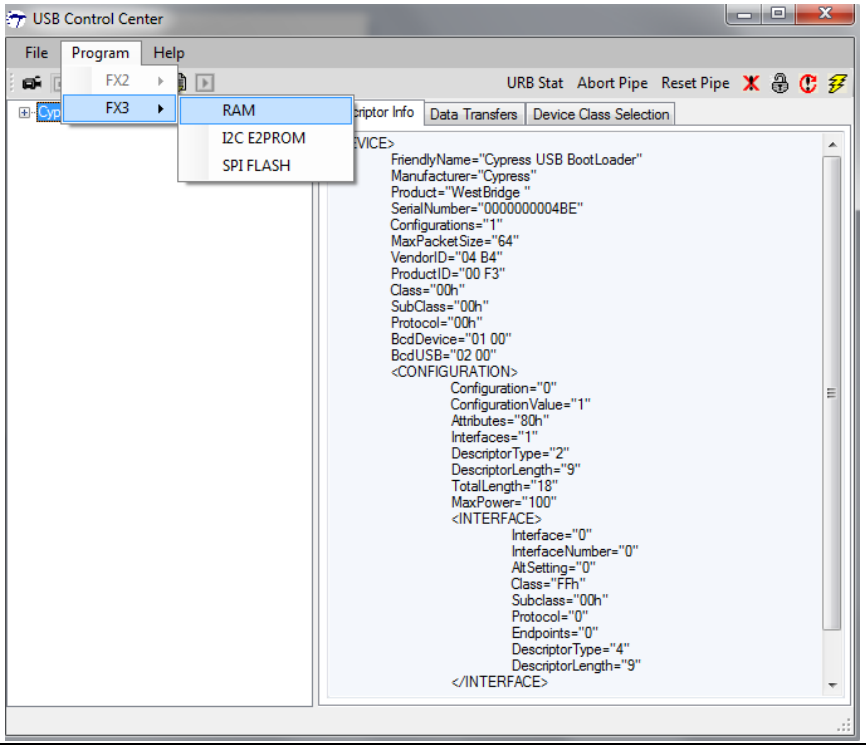

Figure 17　　コントロール センターからデバイスを選択

付録 A: FX3 DVK 基板 (CYUSB3KIT-001) を用いるブート手順

5. 次に、FX3 RAM にプログラムされる*.img* ファイルをブラウズしてください。**Figure 18** に示すように*.img* ファイルをダブルクリックしてください。

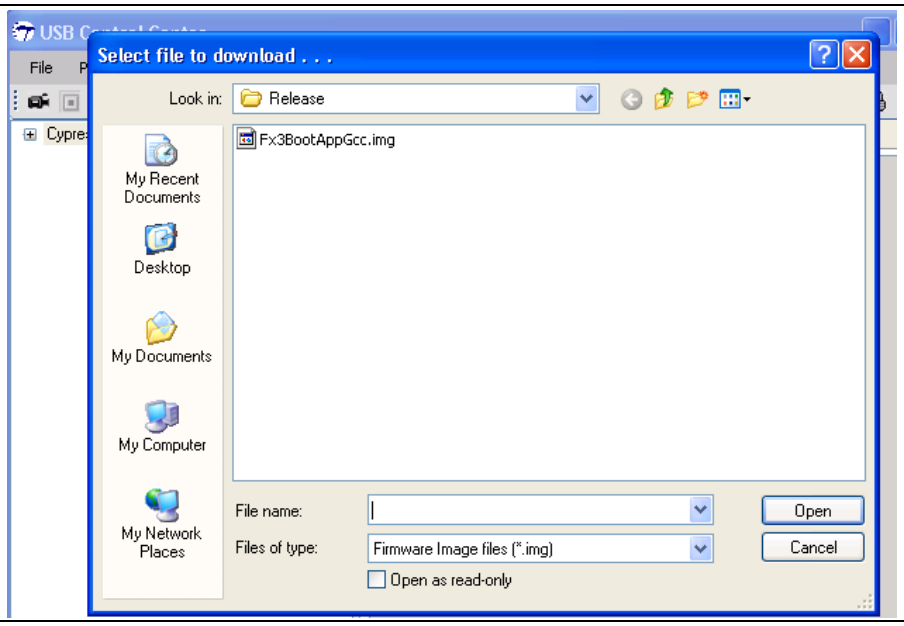

Figure 18 .img ファイルを選択

6. 「Programming Succeeded」 (プログラムが成功) メッセージはコントロール センターの左下に表示され、FX3 はプログラムされたファームウェアで再エニュメレートします。

11.2 I²C ブート

1. **Figure 19**, **Figure 20**, および **Figure 21** に示すように Eclipse IDE でファームウェア イメージをビルドしてください。

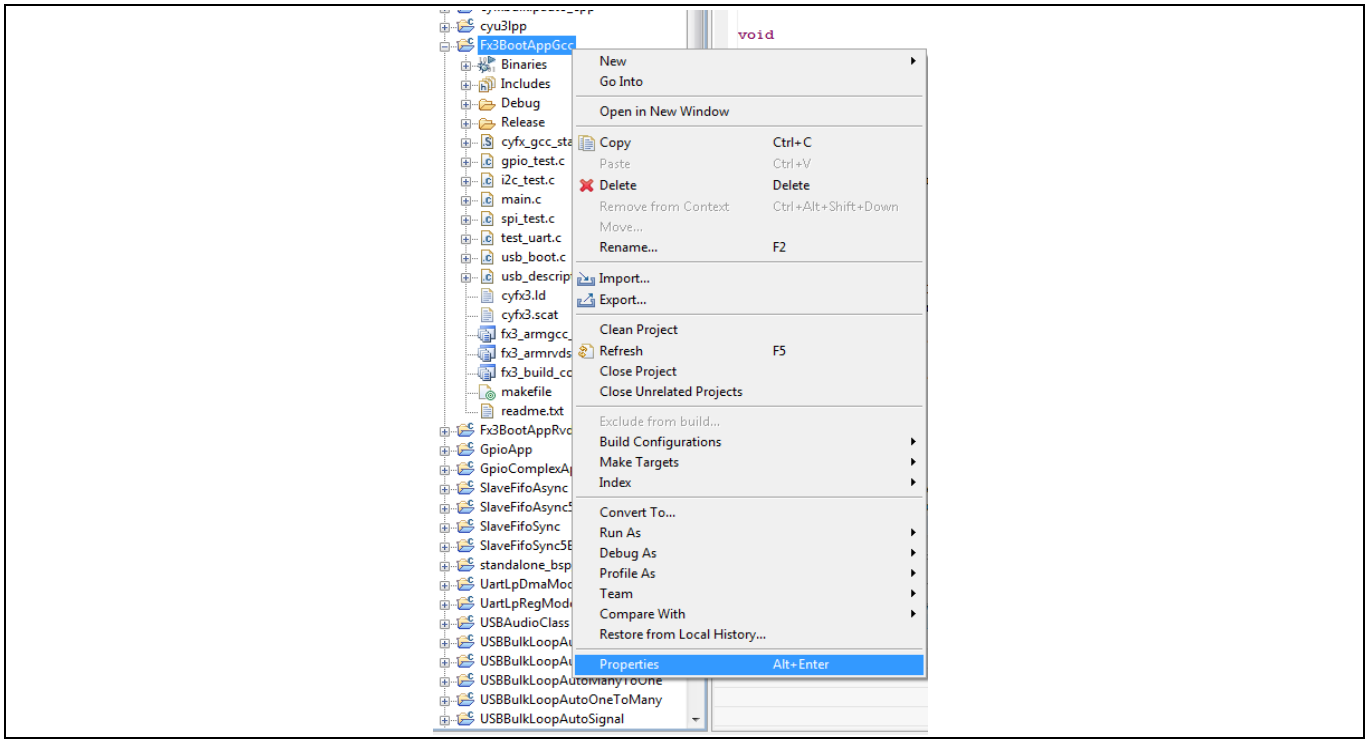

Figure 19 Eclipse IDE でプロジェクトを右クリック

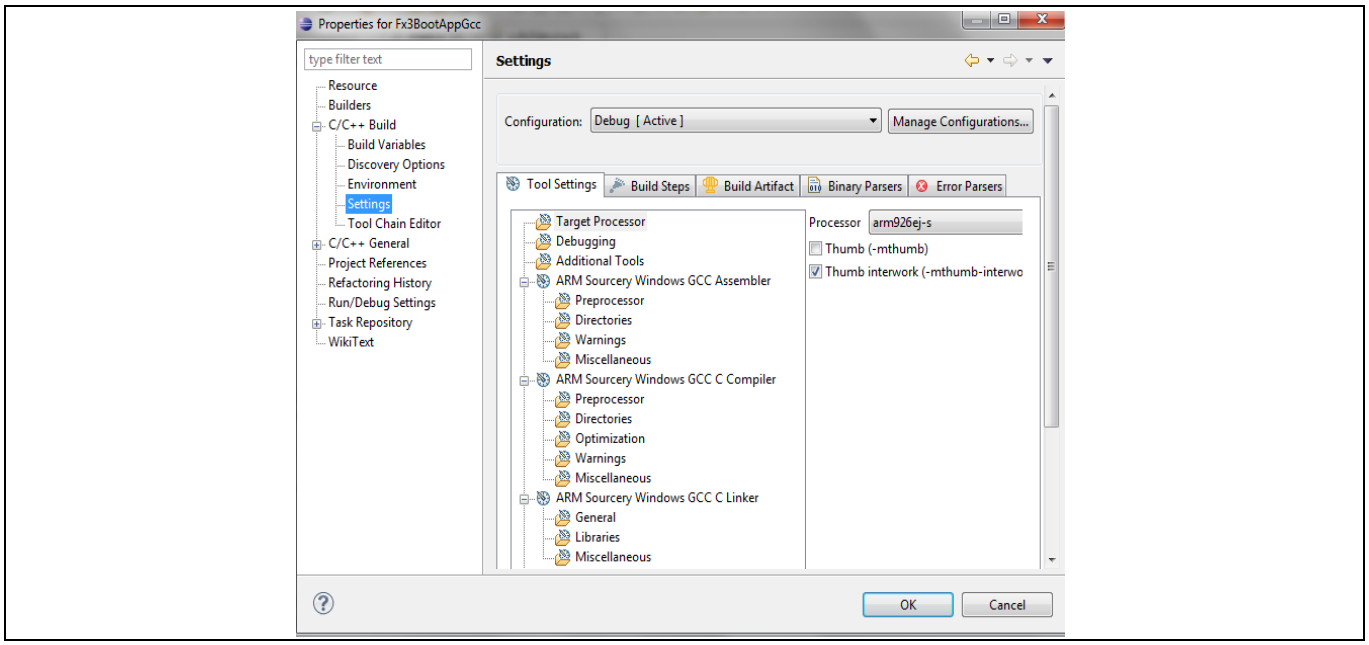

Figure 20 Fx3 BootAppGcc のプロパティ

付録 A: FX3 DVK 基板 (CYUSB3KIT-001) を用いるブート手順

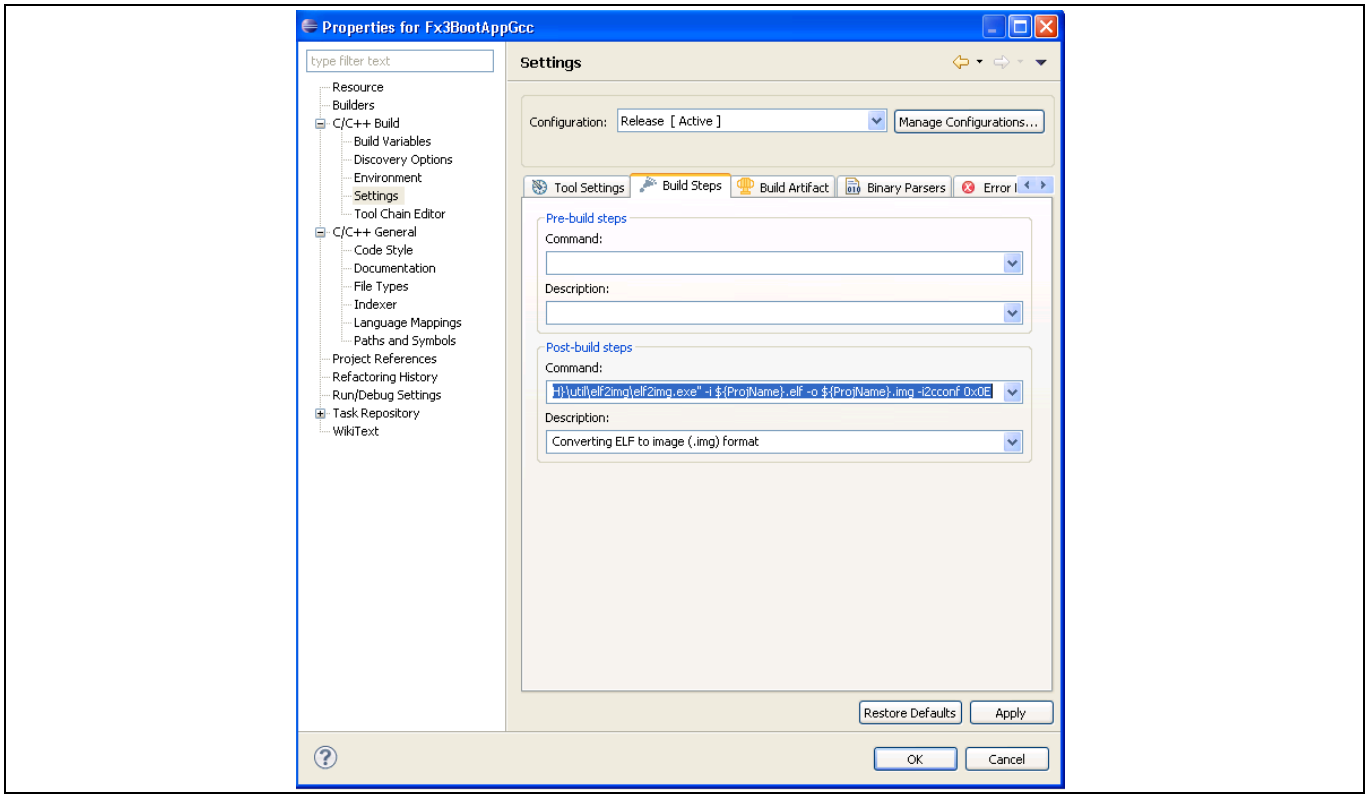

Figure 21 **I²C ブート イメージの elf2img コマンド コンフィギュレーション (ビルド後のステップで)**

2. PMODE[2:0]ピンを Z11 に設定することで USB ブートを有効にしてください。DVK 基板では、これは **Table 32** に示すようにジャンパとスイッチを設定することで実現されます。

Table 32 **USB ブートのジャンパ設定**

ジャンパ/スイッチ	位置	該当する PMODE ピンの状態
J96 (PMODE0)	2-3 投入	PMODE0:SW25 で制御
J97 (PMODE1)	2-3 投入	PMODE1:SW25 で制御
J98 (PMODE2)	開放	PMODE2:未接続
SW25.1-8 (PMODE0)	開放 (OFF)	PMODE0 = 1
SW25.2-7 (PMODE1)	開放 (OFF)	PMODE1 = 1
SW25.3-6 (PMODE2)	ドント ケア	PMODE2:未接続

3. USB ホストに接続した時、FX3 デバイスは **Figure 22** に示すように、コントロール センター内で「Cypress USB BootLoader」 (サイプレス USB ブートローダ) としてエニュメレートします。

付録 A: FX3 DVK 基板 (CYUSB3KIT-001) を用いるブート手順

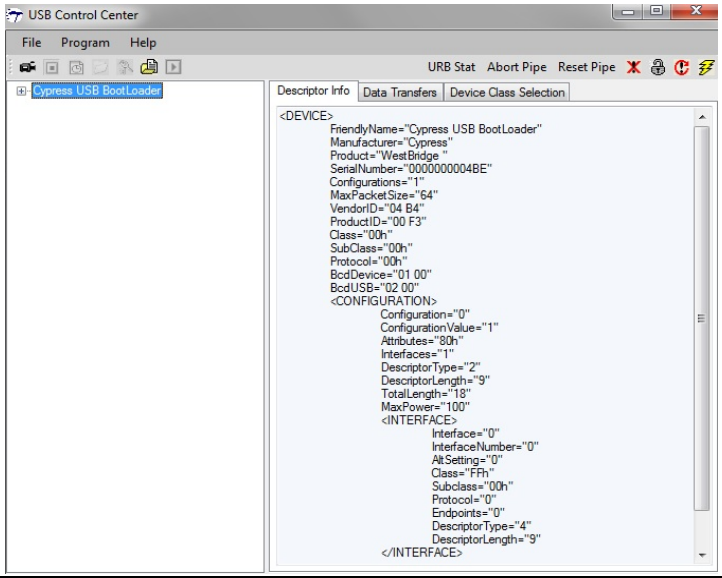

Figure 22　　コントロール センターでのサイプレス USB ブートローダ エニュメレーション

4. EEPROM をプログラムする前に、EEPROM のアドレス信号がスイッチ SW40 を使って正しく設定されたこと (マイクロチップの 24AA1025 製品では、1-8 ON, 2-7 ON, 3-6 OFF) を確認してください。また、DVK 基板では、I²C クロック (SCL) とデータ ライン (SDA) のジャンパ J42 および J45 のピン 1 と 2 を短絡させる必要があります。コントロール センター内で FX3 デバイスを選択します。次に、**Figure 23** に示すように Program > FX3 > I2C E2PROM を選択してください。これにより、特別な I²C ブート ファームウェアが FX3 デバイスにプログラムされ、その後 FX3 に接続された I²C デバイスのプログラミングが可能になります。ここで、**Figure 24** のように FX3 デバイスは「Cypress USB BootProgrammer」として再エニュメレートします。

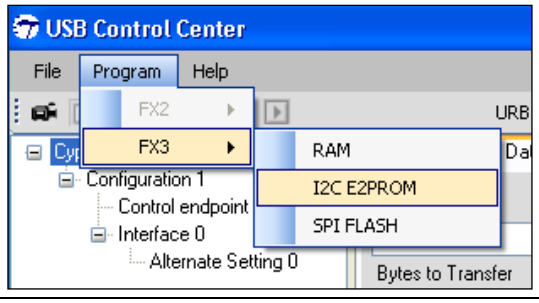

Figure 23　　Program > FX3 > I2C E2PROM を選択

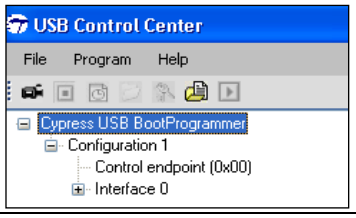

Figure 24　　「Cypress USB BootProgrammer」として FX3 の再エニュメレート

5. FX3 DVK 基板が「Cypress USB BootProgrammer」としてエニュメレートした後、コントロール センター アプリケーションはダウンロードする 2 進数ファームウェア イメージを選択するようユーザーに指示します。に示すように、I²C EEPROM にプログラムされる*.img* ファイルをブラウズしてください。

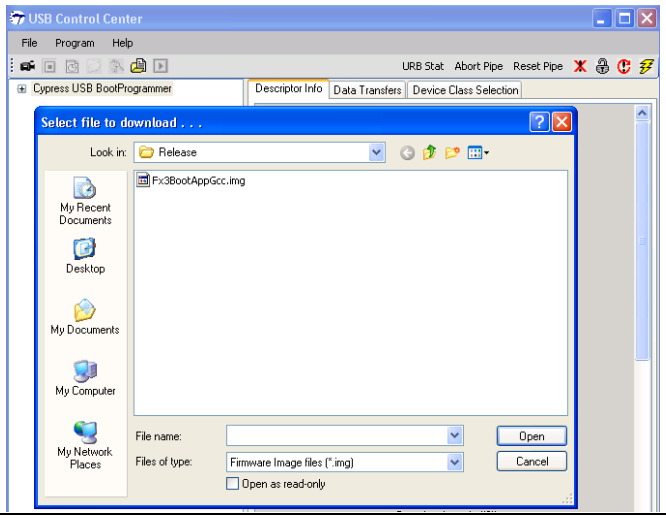

Figure 25　　ダウンロードするファームウェア イメージを選択

プログラミングが完了した後、**Figure 26** に示すように画面の左下隅に「Programming of I2C EEPROM Succeeded」 (I2C EEPROM のプログラミングが成功) が表示されます。

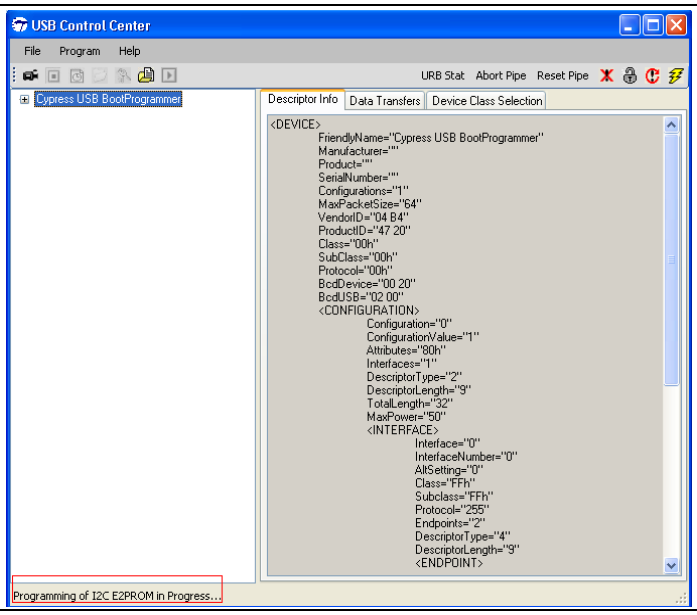

Figure 26　　コントローラー センター内の I²C EEPROM のプログラミング更新

6. DVK 基板の PMODE ピンを Z1Z に変更して I²C ブートを有効にしてください。DVK 基板では、これは **Table 33** に示すようにジャンパとスイッチを設定することで実現されます。

Table 33　　I²C ブートのジャンパ設定

ジャンパ/スイッチ	位置	該当する PMODE ピンの状態
J96 (PMODE0)	開放	PMODE0:未接続
J97 (PMODE1)	2-3 投入	PMODE1:SW25 で制御
J98 (PMODE2)	開放	PMODE2:未接続
SW25.1-8 (PMODE0)	ドント ケア	PMODE0:未接続
SW25.2-7 (PMODE1)	開放 (OFF 位置)	PMODE1 = 1
SW25.3-6 (PMODE2)	ドント ケア	PMODE2:未接続

7. DVK をリセットします。ここで、FX3 デバイスは I²C EEPROM からブートします。

11.3　SPI ブート

1. **Table 27**, **Table 28**, および **Figure 29** に示すように Eclipse IDE でファームウェア イメージをビルドしてください。

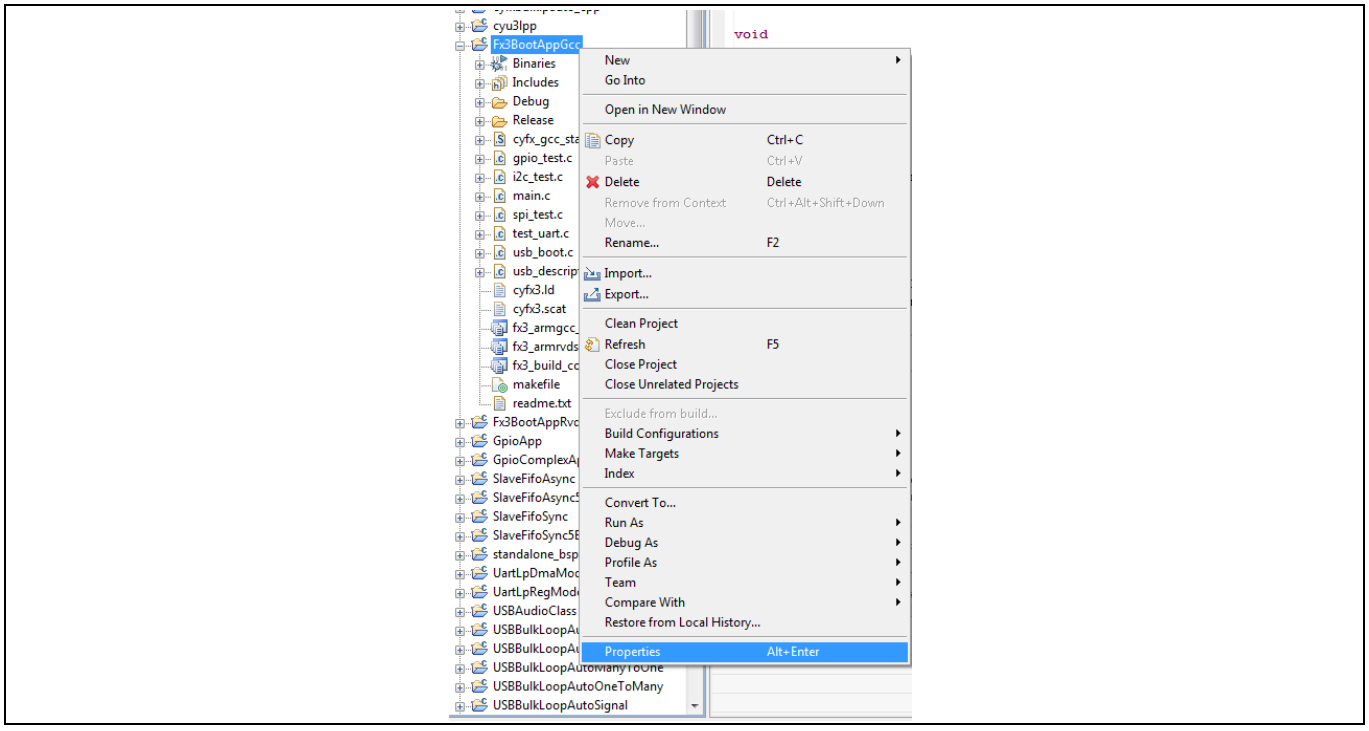

Figure 27　　Eclipse IDE でプロジェクトを右クリック

付録 A: FX3 DVK 基板 (CYUSB3KIT-001) を用いるブート手順

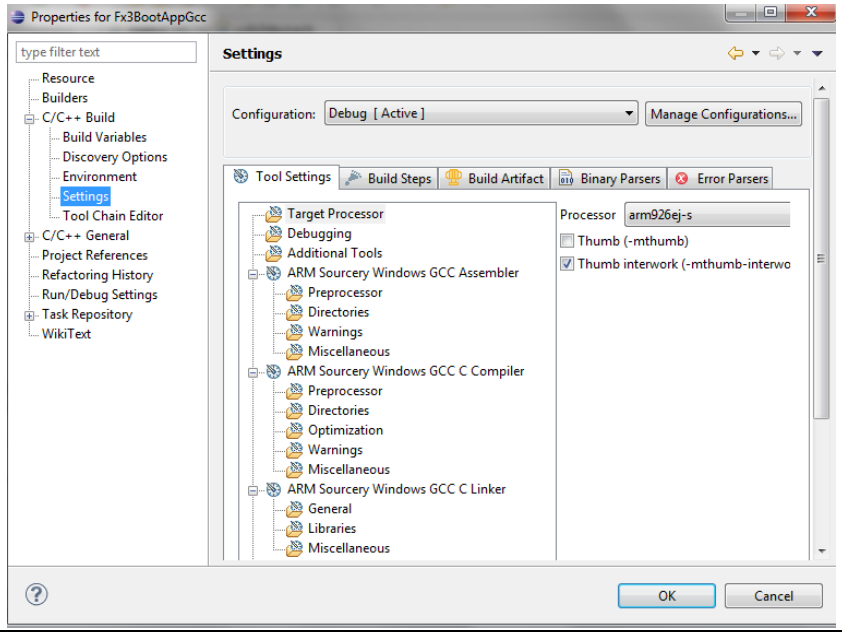

Figure 28　「Settings」を選択

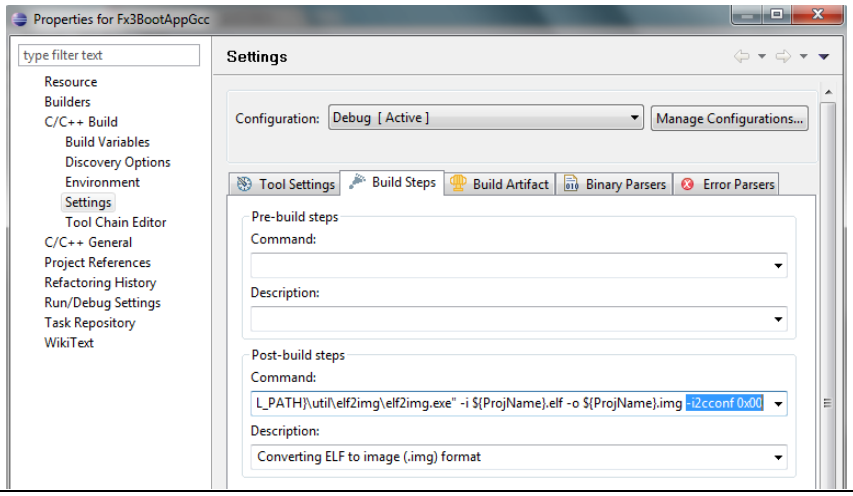

Figure 29　SPI ブート イメージ用の elf2img コマンド コンフィギュレーション (ビルド後のステップで)

2. PMODE[2:0]ピンを Z11 に設定することで USB ブートを有効にしてください。DVK 基板では、これは Table 34 に示すようにジャンパとスイッチを設定することで実現されます。

Table 34　USB ブートのジャンパ設定

ジャンパ/スイッチ	位置	該当する PMODE ピンの状態
J96 (PMODE0)	2-3 投入	PMODE0:SW25 で制御
J97 (PMODE1)	2-3 投入	PMODE1:SW25 で制御
J98 (PMODE2)	開放	PMODE2:未接続
SW25.1-8 (PMODE0)	開放 (OFF 位置)	PMODE0 = 1
SW25.2-7 (PMODE1)	開放 (OFF 位置)	PMODE1 = 1

付録 A: FX3 DVK 基板 (CYUSB3KIT-001) を用いるブート手順

ジャンパ/スイッチ	位置	該当する PMODE ピンの状態
SW25.3-6 (PMODE2)	ドント ケア	PMODE2:未接続

3. USB ホストに接続した時、FX3 デバイスは **Figure 30** に示すように、コントロール センター内で「Cypress USB BootLoader」 (サイプレス USB ブートローダ) としてエニュメレートします。

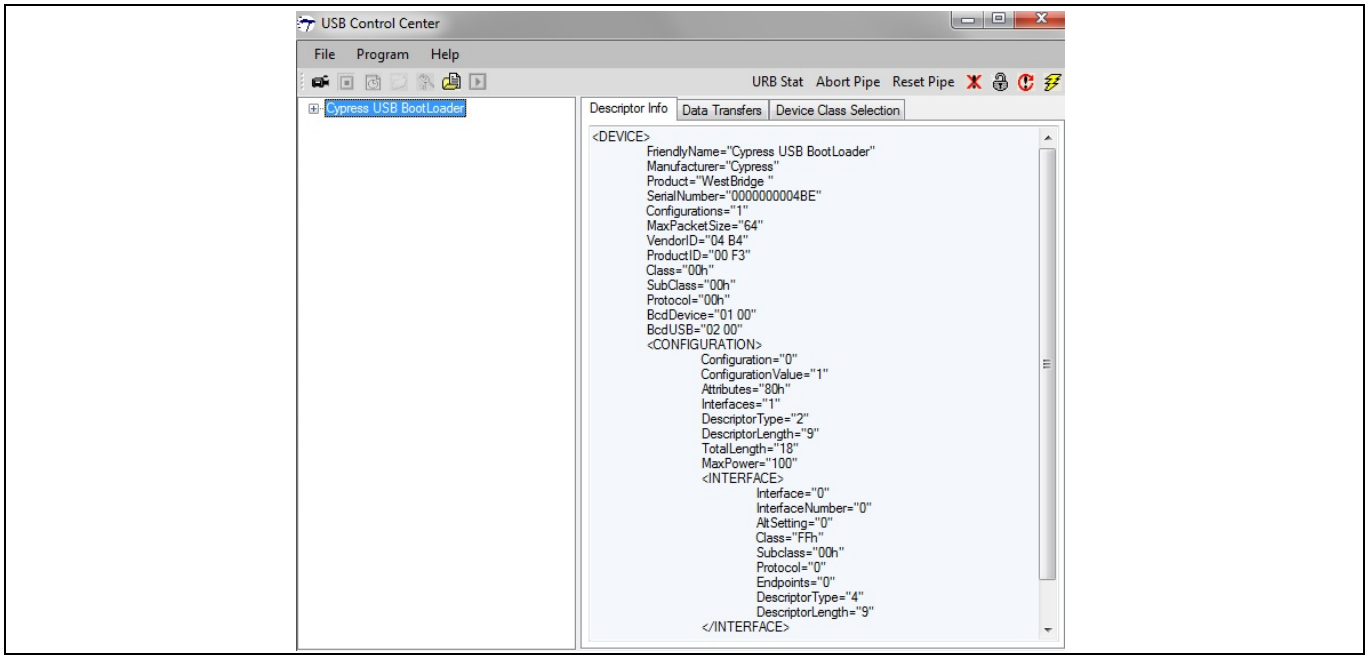

Figure 30　　コントロール センターでのサイプレス USB ブートローダ エニュメレーション

4. コントロール センター内で、**Figure 31** に示すように **Program** > **FX3** > **SPI FLASH** を選択することで FX3 デバイスを選択してください。に示すように、SPI フラッシュにプログラムされる*.img* ファイルをブラウズしてください。

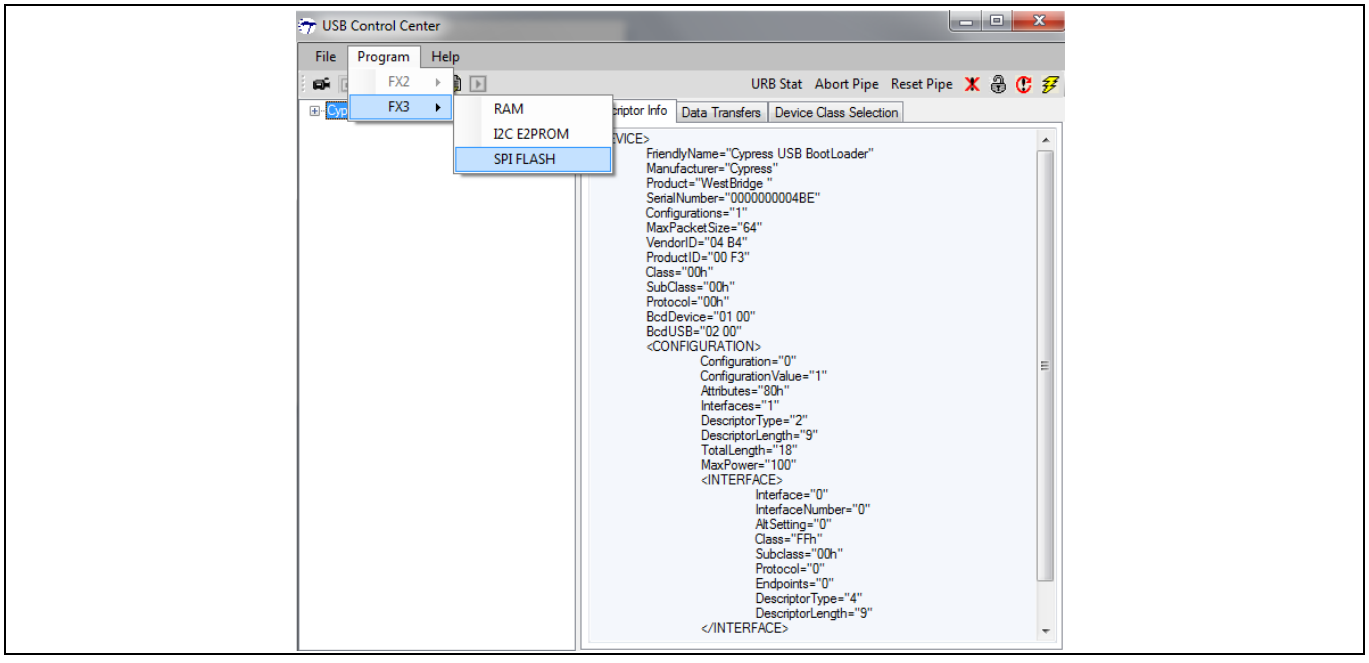

Figure 31　　コントロール センターで Program > FX3 > SPI FLASH を選択

付録 A: FX3 DVK 基板 (CYUSB3KIT-001) を用いるブート手順

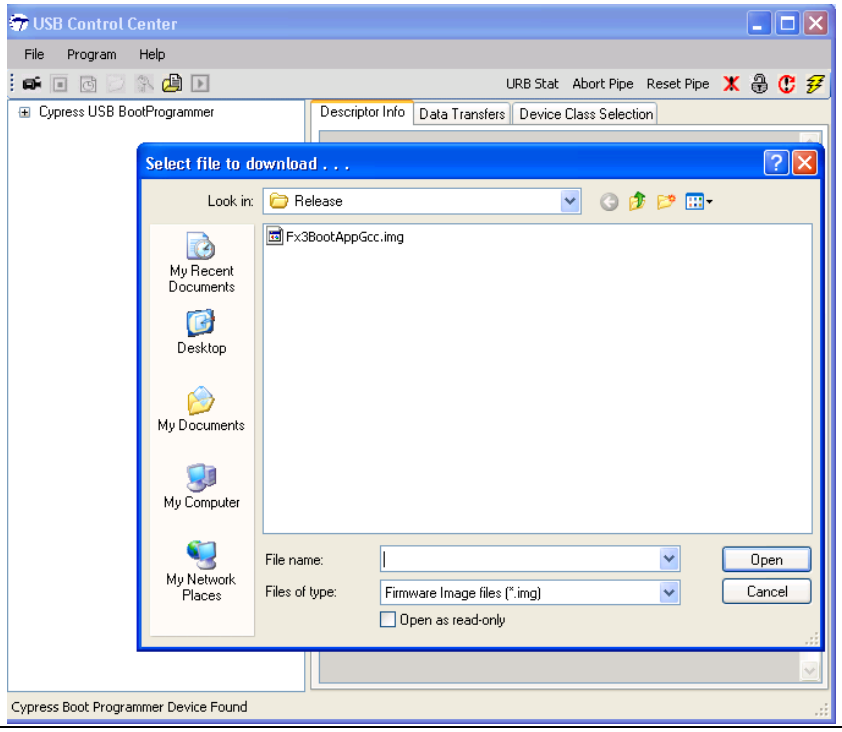

Figure 32　　SPI フラッシュにプログラムされる.img ファイルをダブルクリック

5. プログラミングが完了した後、**Figure 33** に示すように画面の左下隅に「Programming of SPI FLASH Succeeded」 (SPI FLASH のプログラミングが成功) が表示されます。

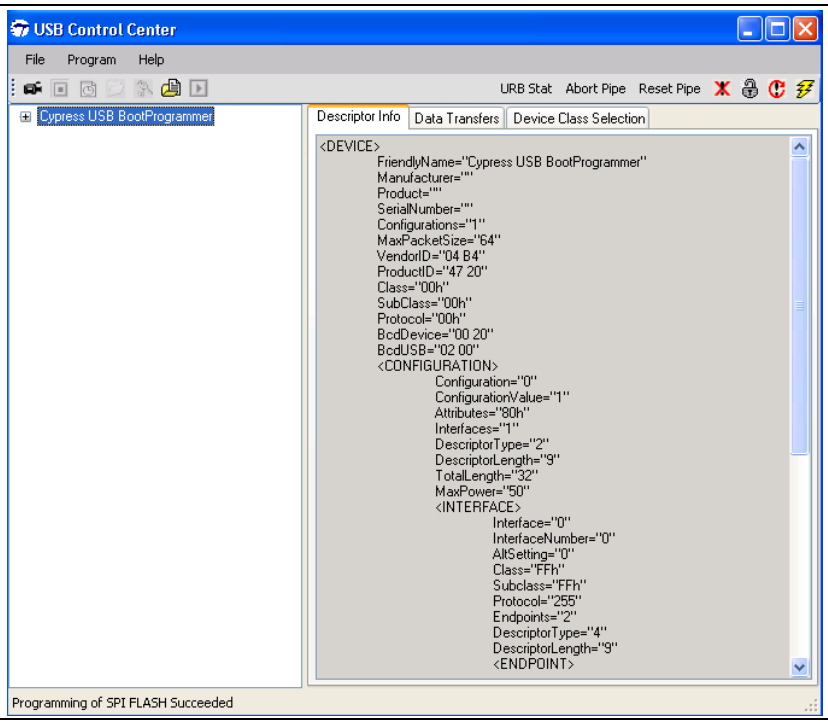

Figure 33　　SPI フラッシュの成功したプログラミングはコントロール センターの左下に表示

付録 A: FX3 DVK 基板 (CYUSB3KIT-001) を用いるブート手順

6. DVK 基板の PMODE[2:0]ピンを 0Z1 に変更して SPI ブートを有効にしてください。DVK 基板では、これは **Table 35** に示すようにジャンパとスイッチを設定することで実現されます。

Table 35 SPI ブートのジャンパ設定

ジャンパ/スイッチ	位置	該当する PMODE ピンの状態
J96 (PMODE0)	2-3 投入	PMODE0:SW25 で制御
J97 (PMODE1)	開放	PMODE1 開放
J98 (PMODE2)	2-3 投入	PMODE2:SW25 で制御
SW25.1-8 (PMODE0)	開放 (OFF 位置)	PMODE0 = 1
SW25.2-7 (PMODE1)	ドント ケア	PMODE1:未接続
SW25.3-6 (PMODE2)	投入 (ON 位置)	PMODE2 = 0

DVK をリセットしてください。ここで、FX3 は SPI フラッシュからブートします。

12 付録 B: 同期 ADMux ブートのトラブルシューティング手順

ここでは、同期 ADmux ブートのさまざまなシーケンスをテストおよびデバッグする段階的な手順を説明します。

12.1 初期化

1. マスター側のメモリ インタフェースを FX3 のタイミング要件を満たすように設定してください。
2. 必要な場合 (Linux などのような場合)、FX3 インタフェースに対応するアドレス領域を仮想メモリの領域にマッピングしてください。
3. アドレス (FX3 アドレス = 0x81) で PP_INIT レジスタに 0x0000 を書き込んでください。
4. アドレス (FX3 アドレス = 0x82) で PP_CONFIG レジスタに 0x0040 を書き込んでください。

12.2 レジスタ読み書きのテスト

1. PP_SOCK_MASK_H (0x8B) および PP_SOCK_MASK_L (0x8A) のレジスタにさまざまな値を書き込んで、テストおよび検証のために読み出してください。
2. PP_INTR_MASK (0x88) レジスタにさまざまな値を書き込んで、検証のために読み出してください。このレジスタには予約済みビットがあることに注意してください (読み出し値 = 書き込み値および 0xF8FF) 。
3. FIFO アクセス テストのセットアップのために下記の値をこれらのレジスタに書き込んでください。

   PP_SOCK_MASK_H = 0x0000

   PP_SOCK_MASK_L = 0x0007

   PP_INTR_MASK = 0x2001

12.3 FIFO 読み書きのテスト

FX3 ブートローダが提供したメモリ読み書きデバッグ コマンドは FIFO アクセスのテスト、インタフェースが正常に動作するかの検証、またファームウェアのダウンロードに使用されます。

1. メモリ アドレス 0x40003000 にデータ パターンを書き込んでください。
   a) PP_SOCK_STAT_L レジスタ (0x9E) のビット 0 (ソケット 0 有効) がセットされるまで待ってください。
   b) PP_DMA_XFER レジスタ (0x8E) に 0x0300 を書き込んでください。
   c) PP_DMA_XFER レジスタのビット 12 とビット 15 がセットされるまで待ってください。
   d) 以下のフォーマット[1]で 512 バイトのデータを FX3 デバイス アドレス 0 (ソケット 0) に書き込んでください。

      バイト 0 = 0x43

      バイト 1 = 0x59

      バイト 2 = 0x01 (書き込みコマンド)

      バイト 3 = 0x7E

      バイト 4 = 0x00 (アドレスの LSB)

      バイト 5 = 0x30

1 パケット構造の詳細については、**USB 2.0 Specification** を参照してください。

付録 B: 同期 ADMux ブートのトラブルシューティング手順

バイト 6 = 0x00

バイト 7 = 0x40 (アドレスの MSB)

バイト 8～511 はランダムなデータを含みます。

2. 書き込みの状態を読み出してください。
   a) PP_SOCK_STAT_L レジスタ (0x9E) のビット 2 がセットされるまで待ってください。
   b) PP_DMA_XFER レジスタ (0x8E) に 0x0102 を書き込んでください。
   c) PP_DMA_XFER レジスタのビット 12 がセットされるまで待ってください。
   d) PP_DMA_SIZE レジスタ (0x8F) を読み出して、値が 0x0200 であることを確認してください。
   e) FX3 アドレス 0x02 から 512 [1]バイトのデータ (256 サイクル) を読み出してください。
   f) 最初の 4 バイトが 0x57、0x42、0x01 (ドントケア) および 0x00 パターンを含むことを確認してください。
3. アドレス 0x40003000 からデータを読み出すための読み出しコマンドを発行してください。
   a) PP_SOCK_STAT_L レジスタ (0x9E) のビット 0 がセットされるまで待ってください。
   b) PP_DMA_XFER レジスタ (0x8E) に 0x0300 を書き込んでください。
   c) PP_DMA_XFER レジスタのビット 12 とビット 15 がセットされるまで待ってください。
   d) 以下のフォーマットで 512 バイトのデータを FX3 デバイス アドレス 0 (ソケット 0) に書き込んでください。

      バイト 0 = 0x43

      バイト 1 = 0x59

      バイト 2 = 0x03 (読み出しコマンド)

      バイト 3 = 0x7E

      バイト 4 = 0x00 (アドレスの LSB)

      バイト 5 = 0x30

      バイト 6 = 0x00

      バイト 7 = 0x40 (アドレスの MSB)

      バイト 8～511 はドント ケアです。

4. ソケット 2 からメモリ データを読み出してください。
   a) PP_SOCK_STAT_L レジスタ (0x9E) のビット 2 がセットされるまで待ってください。
   b) PP_DMA_XFER レジスタ (0x8E) に 0x0102 を書き込んでください。
   c) PP_DMA_XFER レジスタのビット 12 がセットされるまで待ってください。
   d) PP_DMA_SIZE レジスタ (0x8F) を読み出して、値が 0x0200 であることを確認してください。
   e) FX3 アドレス 0x02 から 512 バイトのデータ (256 サイクル) を読み出してください。
   f) 最初の 4 バイトが 0x57, 0x42, 0x03, および 0x00 パターンを含むことを確認してください。
   g) バイト 8～511 が上記のステップ 1 で書き込んだランダムなデータと一致することを確認してください。
5. 他のメモリ アドレスおよびデータ パターンに対してステップ 1～4 を繰り返してください。

---

1 書き込み動作の状態を読み出している間に、正しい応答が受信されない場合は、512 バイトの代わりに FX3 応答ソケット (アドレス 0x02) から 256 バイトのデータ(128 サイクル)を読み出してください。

12.4 ファームウェア ダウンロードのテスト

上記のすべてのチェックが合格であれば、ファームウェア ダウンロードのテストに進んでください。ファームウェア ダウンロードのためには、以下の手順に従います [1]。

1. ターゲット ファームウェアで.img ファイルの内容を読み出して、メモリ バッファに転送してください。必要に応じてデータを 512 バイトの倍数に分割してください。これは、ブートローダがフルパケットのみをサポートするように設計されているためです。フルパケットのサイズは bLenStatus フィールドで指定されます。
2. **12.3** のステップ 1 に従い、ソケット 0 に 0x43、0x59、0x02、0x01、... のデータを書き込んでください (残りの 508 バイトはドント ケアです)。
3. **12.3** のステップ 2 に従い、ソケット 2 からファームウェア ダウンロード コマンドを読み出してください。バイト 3 (ステータス) が 0x00 の値を持つことを確認してください。
4. ここで、完全なファームウェア内容を (512 バイトずつ) ソケット 1 に書き込んでください。下記の手順に従い、512 バイトずつソケット 1 の書き込んでください。
   a) PP_SOCK_STAT_L レジスタ (0x9E) のビット 1 がセットされるまで待ってください。
   b) PP_DMA_XFER レジスタ (0x8E) に 0x0301 を書き込んでください。
   c) PP_DMA_XFER レジスタのビット 12 とビット 15 がセットされるまで待ってください。
   d) FX3 デバイス アドレス 1 に 512 バイトのデータを書き込んでください。

1 ここで説明している手順は bLenStatus = 1 (単一の 512 バイトブロック) に基づきます。bLenStatus が 1 より大きい場合は、手順 1～4 で説明した転送ごとのデータチャンクのサイズをそれに応じて変更する必要があります。512 バイト < 1 転送あたりのデータチャンクのサイズ < 8KB であることに注意してください。

13 付録 C: elf2img ユーティリティを使用したファームウェアイメージの生成

ここでは、このアプリケーションノートに記載されている起動オプションのファームウェアイメージを生成するために、elf2img ユーティリティ (SDK インストールパスの *util\elf2img* フォルダ内) を使用する方法について説明します。

13.1 使用方法

このユーティリティは、次のオプションを指定して呼び出す必要があるコンソールアプリケーションです。

```
elf2img.exe -i <elf filename> -o <image filename> [-i2cconf <eeprom control>]
[-vectorload <vecload>] [-imgtype <image type>] [-v] [-h]
```

ここで、

`<elf filename>`: 入力する ELF ファイルについて、パス付でのファイル名

`<image filename>`: 出力するファイルについて、パス付でのファイル名

`<eeprom control>`: 16 進数形式の I2C / SPI EEPROM 制御ワード

`<image type>`: 16 進形式のイメージタイプバイト

-v: 変換処理中に詳細ログを有効にする

-h: ヘルプ情報を表示する

13.1.1 イメージタイプ

`<image type>`は、すべてのファームウェアアプリケーションで 0xB0 でなければいけません。その他の値は予約されています。

13.1.2 割込みベクタロード

FX3 デバイスの ARM926EJ-S コアは、メモリの最初の 256 バイト (アドレス範囲 0x00-0x100) に格納されたリセットおよび割込みベクタを持ちます。ブートローダまたはアクティブなファームウェアの動作を妨げる可能性があるため、このアドレス範囲に直接コードをロードすることは推奨しません。FX3 ファームウェアライブラリとデフォルトのリンカ設定は、有効なコードがこのアドレス範囲に直接ロードされないようにします。ファームウェアの実行が開始されると、割込みベクタはこの領域に安全にコピーされます。

デフォルトモードの elf2img ユーティリティは、ブートイメージの生成中に 0x00-0x100 のアドレス範囲のデータをすべて削除します。推奨されるリンカ設定では、このアドレス範囲に有効なコード/データは配置されていないため、削除に問題はありません。この動作は、`-vectorload` コマンドラインオプションを使用して無効にできます。

`<vecload>`の値は yes/no 文字列です。これを「yes」に設定すると、ツールはブートイメージのこのアドレス範囲のデータを保持します。このパラメーターのデフォルト値は「no」です。

13.1.3 EEPROM 制御

このパラメーターは、I2C EEPROM または SPI FLASH からブートする場合にのみ適用されます。 FX3 が USB または GPIF ポート経由で起動されている場合、img ファイルの生成中にこのフィールドは使用されず、省略できます。

I2C ブートの場合、<eeprom control>バイトは使用される EEPROM のタイプと速度を指定します。

SPI ブートの場合、<eeprom control>バイトは、EEPROM から SPI ブートを実行する速度を指定します。このバイトは、他のブートモードが使用されている場合でも影響はありません。

13.1.3.1 I2C パラメーター

I2C ブートの場合のエンコーディングは次のとおりです。

Bit 0	ゼロでなければならない
Bits 3 – 1	EEPROM サイズ [7 = 128 KB, 6 = 64 KB, 5 = 32 KB, 4 = 16 KB, 3 = 8 KB, 2 = 4 KB]
Bits 5 – 4	EEPROM 速度 [0 = 100 KHz, 1 = 400 KHz, 2 = 1 MHz]
Bits 7 - 6	ゼロでなければならない

例えば、0x1C の値は、400 kHz の周波数で 64 KB の EEPROM を使用することを意味します。

13.1.3.2 SPI パラメーター

SPI ブートの場合のエンコーディングは次のとおりです。

Bit 0	ゼロでなければならない
Bits 3 – 1	ドント ケア
Bits 5 – 4	SPI 動作周波数 [0 = 10 MHz, 1 = 20 MHz, 2 = 30 MHz]
Bits 7 - 6	ゼロでなければならない

例えば、0x1C の値は、20 MHz の SPI 動作周波数に対して.img を生成します。

関連資料

[1] **AN75705** - EZ-USB™ FX3 入門

[2] **EZ-USB™ FX3 TRM**

改訂履歴

Document version	Date of release	Description of changes
**	2013-04-22	これは英語版 001-76405 Rev. *B を翻訳した日本語版 Rev. **です。
*A	2015-10-01	これは英語版 001-76405 Rev. *E を翻訳した日本語版 Rev. *A です。
*B	2017-06-13	これは英語版 001-76405 Rev. *G を翻訳した日本語版 Rev. *B です。
*C	2018-11-28	これは英語版 001-76405 Rev. *I を翻訳した日本語版 Rev. *C です。
*D	2019-07-03	これは英語版 001-76405 Rev. *J を翻訳した日本語版 Rev. *D です。
*E	2021-07-21	これは英語版 001-76405 Rev. *K を翻訳した日本語版 Rev. *E です。
*F	2022-04-20	これは英語版 001-76405 Rev. *L を翻訳した日本語版 Rev. *F です。

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

Edition 2022-04-20
Published by
Infineon Technologies AG
81726 Munich, Germany

© 2022 Infineon Technologies AG.
All Rights Reserved.

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

Document reference
001-87273 Rev. *F

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。