AN76405

EZ-USB® FX3™/FX3S™启动选项

作者：**Rama Sai Krishna V**

相关项目：无

相关器件系列：**CYUSB30xx**

相关应用笔记：要获取完整列表，请点击此处

更多代码示例？ 我们听到了你的声音

有关 **USB SuperSpeed** 代码示例的完整列表，请访问 **http://www.cypress.com/101781**

AN76405 介绍了通过 USB、I²C、SPI 以及同步地址数据复用（ADMux）接口执行的启动选项，这些选项适用于赛普拉斯 EZ-USB® FX3™外设控制器。本应用笔记还适用于 FX3S 和 CX3 外设控制器。

目录

1 简介

EZ-USB FX3 是新一代 USB 3.0 外设控制器，它集成了高级灵活的功能，可帮助开发人员为各种应用添加 USB 3.0 功能。

FX3 支持多种启动选项，包括通过 USB、I²C、SPI、同步与异步 ADMux 以及异步 SRAM 等接口执行的启动。

注意：本应用笔记仅对 USB、I²C、SPI 和同步 ADMux 启动选项进行描述。

此外，还介绍了启动过程中 FX3 IO 的默认状态。附录 A 部分对使用 FX3 DVK 测试各启动模式提供了逐步引导。

2 相关资源

赛普拉斯的网站 www.cypress.com 上提供了大量资料，有助您正确选择器件用于设计，并帮助您能够快速和有效地将器件集成到设计中。

- 概述：USB 产品系列、USB 路线图
- USB 3.0 产品选型器：FX3、FX3S、CX3、HX3
- 应用笔记：赛普拉斯提供了大量的 USB 应用笔记，包括从基本到高级的广泛主题。下面列出的是 FX3 入门的应用笔记：
  - AN75705 — EZ-USB FX3 入门
  - AN70707 — EZ-USB FX3/FX3S 硬件设计指南和原理图检查表
  - AN65974 — 使用 EZ-USB FX3 从设备 FIFO 接口进行设计
  - AN75779 — 如何使用 EZ-USB FX3 在 USB 视频类别（UVC）框架内实现图像传感器连接
  - AN86947 — 使用 EZ-USB FX3 优化 USB 3.0 的吞吐量
  - AN84868 — 使用赛普拉斯 EZ-USB FX3 通过 USB 配置 FPGA
  - AN68829 — 用于 EZ-USB FX3 的从设备 FIFO 接口：5 位地址模式
  - AN76348 — EZ-USB FX2LP 和 EZ-USB FX3 应用的区别
  - AN89661 — 使用 EZ-USB FX3S 设计 USB RAID 1 磁盘
- 代码示例：
  - USB 高速
  - USB 全速
  - USB 超高速
- 技术参考手册（TRM）：
  - EZ-USB FX3 技术参考手册
- 开发套件：
  - CYUSB3KIT-003 · EZ-USB FX3 超高速 Explorer 套件
  - CYUSB3KIT-001 · EZ-USB FX3 开发套件
- 模型：IBIS

2.1 EZ-USB FX3 软件开发套件

赛普拉斯为 FX3 提供了完整的软件和固件堆栈，这样很容易便能够将超高速 USB 集成到嵌入式应用内。软件开发套件（SDK）带有各种工具、驱动程序和应用示例，有助于加快应用开发程序。

2.2 GPIF™ II Designer

GPIF II Designer 是一个图形软件，设计师可以通过它来配置 EZ-USB FX3 USB 3.0 器件控制器的 GPIF II 接口。

通过使用该工具，用户可以从赛普拉斯所提供的五个接口选择一个，或从头开始创建自定义的 GPIF II 接口。赛普拉斯提供了符合工业标准的接口，如异步和同步从设备 FIFO、异步和同步 SRAM。在具有上面所述接口的系统中，开发者可从一组标准参数（如总线宽度（x8、16、x32）、字节顺序、时钟设置）选择所需要的接口，然后，编译已选的接口。该工具为需要自定义接口的用户提供了一个简洁的三步骤 GPIF 接口开发程序。用户先选择引脚配置和标准参数。然后，可以使用可配置操作设计一个虚拟的状态机。最后，用户通过查看输出时序验证是否与所需时序相匹配。一旦完成这三个步骤，便可以将该接口编译并集成到 FX3。

3 FX3 启动选项

FX3 集成了一个位于屏蔽 ROM 内的 Bootloader。Bootloader 用于通过各种接口（如 USB、I²C、SPI 或 GPIF II（同步 ADMux、异步 SRAM 或异步 ADMux））下载 FX3 固件镜像。

FX3 Bootloader 使用 FX3 上的三个 PMODE 输入引脚来确定将要使用的启动选项。图 1 显示了本应用笔记所介绍的启动选项。表 1 列出了这些启动选项以及所需的 PMODE 引脚设置。

图 1. FX3 启动选项

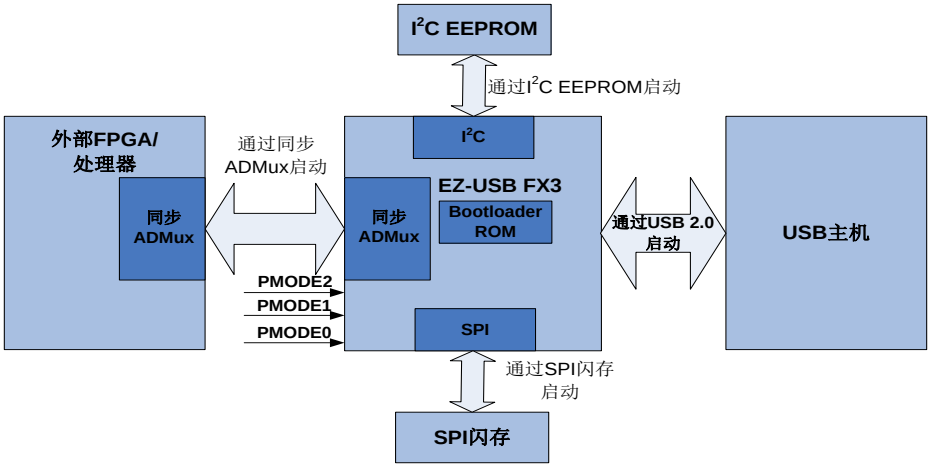


表 1. FX3 的启动选项*

PMODE[2:0]引脚			启动选项	USB 回退功能
PMODE[2]	PMODE[1]	PMODE[0]		
Z	0	0	同步 ADMux（16 位）	无
Z	1	1	USB 启动	有
1	Z	Z	I²C	无
Z	1	Z	I²C → USB	有
0	Z	1	SPI → USB	有
1	0	0	eMMC**	无
0	0	0	eMMC** → USB	有
其他组合均被保留				

注意：

***** Z = 悬空。通过执行以下操作可使 PMODE 引脚进入悬空状态：断开同其的连接；或者将其连接到某个 FPGA I/O，然后将该 I/O 配置为 FPGA 输入。

****：**eMMC 启动仅由 FX3S 支持。

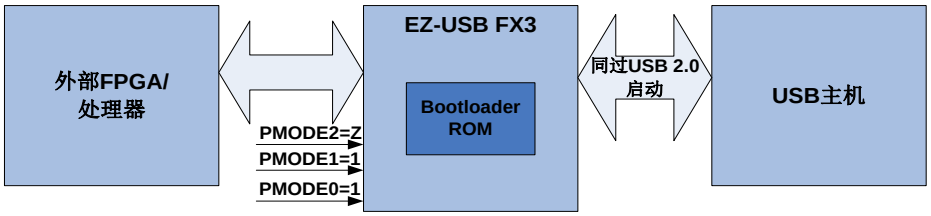
除了表 1 列出的启动选项外，FX3 还支持从异步 SRAM 和异步 ADMux 接口启动。有关的详细信息，请联系赛普拉斯应用支持。下面章节介绍了 FX3 支持的启动选项：

- USB 启动：从 USB 主机将 FX3 固件镜像下载到 FX3 系统 RAM 内。
- I²C EEPROM 启动：FX3 固件镜像被编程到一个外部 I²C EEPROM 内。发生复位时，FX3 Bootloader 将通过 I²C 下载该固件。
- SPI 启动：FX3 的固件镜像被编程到一个外部 SPI 闪存或 SPI EEPROM 内。发生复位时，FX3 Bootloader 将通过 SPI 下载该固件。
- 同步 ADMux 启动：从 FX3 GPIF II 接口上所连接的外部处理器或 FPGA 中下载 FX3 固件镜像。

4 USB 启动

图 2 显示了 FX3 通过 USB 进行启动的系统框图。

图 2. FX3 系统框图



4.1 PMODE 引脚

对于 USB 启动，PMODE[2:0]引脚的状态应该为 Z11，如表 2 所示。

表 2. USB 启动的 PMODE 引脚

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

注意：Z = 悬空

4.2 功能

在 USB 2.0 模式中，外部 USB 主机可将固件镜像下载到 FX3 内。FX3 将枚举为总线供电式 USB 供应商类型设备。

在 USB 启动模式下，FX3 的状态如下：

- USB 3.0（超高速）的信号通信功能被禁用
- USB 2.0（高速/全速）被使能
- FX3 使用供应商命令 A0h 进行下载/上传固件。该供应商命令是在 Bootloader 中实现的。（与 FX2LP™不同，A0h 供应商命令是在固件中（即在 Bootloader 代码中）实现的）

4.2.1 默认芯片 ID

默认情况下，FX3 的默认赛普拉斯半导体 VID=04B4h 和 PID=00F3h，这些信息存储在 ROM 中。除非编程了 eFUSE[1] VID/PID，否则，该 VID/PID 将使用于默认的 USB 枚举。默认的赛普拉斯 ID 值仅使用于开发目的。对于终端产品，用户必须使用他们自己的 VID/PID。通过向 USB-IF 登记，可以得到 VID。

4.2.2 Bootloader 版本

Bootloader 版本信息存储在 ROM 中的地址 FFFF_0020h 空间内，如表 3 所示。

表 3. Bootloader 版本

次要版本	FFFF_0020h
主要版本	FFFF_0021h
保留字节	FFFF_0022h、FFFF_0023h

4.2.3 ReNumeration™

FX3 支持赛普拉斯的重枚举功能，该功能由固件控制。

首次插入 USB 主机时，FX3 使用其默认的 USB 描述符自动枚举。下载固件后，FX3 再次枚举。此时，FX3 作为一个由已下载的 USB 描述符定义的设备。这个双步骤的过程被称为“重枚举”。

4.2.4 总线供电应用

Bootloader 在总线供电模式下进行枚举。FX3 可以按照 USB 2.0 规范的要求使用低于 100 mA 的电流进行枚举，从而全面支持总线供电设计。

4.2.5 USB 回退功能选项（--> USB）

在 USB 回退功能被使能时，如果通过其他选项启动，FX3 则将回退至本节中所描述的同一个 USB 启动模式。由于其他时钟源均被打开，因此工作电流可能略高于 USB 启动模式中的电流。

4.2.6 带有 VID/PID 选项的 USB

Bootloader 支持通过新的 VID/PID 进行启动，这些 VID/PID 的存储位置为：

- I²C EEPROM：请查看本应用笔记中的 I2C EEPROM 启动一节。
- SPI EEPROM：请查看本应用笔记中的 SPI 启动一节。
- eFUSE（VID/PID）：有关自定义的 eFUSE VID/PID 编程，请联系赛普拉斯销售人员。

4.2.7 USB 默认器件

FX3 Bootloader 具有单一的 USB 配置（包含一个接口（接口 0）和接口 0 的备用设置）。在该模式下，只有端点 0 被使能。其他所有端点均被关闭。

[1] eFUSE 是能够在片内重新编程某些电路的技术。有关 eFUSE 编程的详细信息，请联系您的赛普拉斯代表。

4.2.8 USB SETUP 数据包

FX3 Bootloader 对包含了表 4 中定义的 8 字节数据结构的 SETUP 数据包进行解码。

表 4. SETUP 数据包

字节	字段	说明
0	***bmRequestType***	请求类型：位 7：方向 位 6-0：接收方
1	***bRequest***	该字节为 A0h，用于固件下载/上传供应商命令
2-3	***wValue***	16 位值（低位优先格式）
4-5	***wIndex***	16 位值（低位优先格式）
6-7	***wLength***	字节数量

注意：有关字节顺序的说明，请参考 USB 2.0 规范。

4.2.9 USB 第 9 章和供应商命令

FX3 Bootloader 处理表 5 中的命令。

表 5. USB 命令

bRequest	说明
00	GetStatus：器件、端点和接口
01	ClearFeature：器件和端点
02	Reserved：返回 STALL（停止）
03	SetFeature：器件和端点
04	Reserved：返回 STALL（停止）
05	SetAddress：在 FX3 硬件中进行处理
06	GetDescriptor：ROM 中的器件描述符
07	Reserved：返回 STALL（停止）
08h	GetConfiguration：返回内部值
09h	SetConfiguration：设置内部值
0Ah	GetInterface：返回内部值
0Bh	SetInterface：设置内部值
0Ch	Reserved：返回 STALL（停止）
20h-9Fh	Reserved：返回 STALL（停止）
A0h	供应商命令：固件上传/下载，等等
A1h-FFh	Reserved：返回 STALL（停止）

4.2.10 USB 供应商命令

Bootloader 使用 A0h 供应商命令来下载/上传固件。命令的字段显示在表 6 和表 7 内。

表 6. 用于固件下载的命令字段

字节	字段	数值	说明
0	***BmRequest Type***	40h	请求类型：位 7：方向 位 6-0：接收方。
1	***bRequest***	A0h	该字节为 A0，用于固件下载/上传供应商命令。
2-3	***WValue***	AddrL（LSB）	16 位值（低位优先格式）
4-5	***WIndex***	AddrH（MSB）	16 位值（低位优先格式）
6-7	***wLength***	计数	字节数量

表 7. 用于固件上传的命令字段

字节	字段	数值	说明
0	***BmRequest Type***	C0h	请求类型：位 7：方向 位 6-0：接收方。
1	***bRequest***	A0h	该字节为 A0，用于固件下载/上传供应商命令。
2-3	***WValue***	AddrL（LSB）	16 位值（低位优先格式）
4-5	***WIndex***	AddrH（MSB）	16 位值（低位优先格式）
6-7	***wLength***	计数	字节数量

表 8. 用于跳到程序入口处的命令字段

字节	字段	数值	说明
0	***bmRequest Type***	40h	请求类型：位 7：方向 位 6-0：接收方
1	***bRequest***	A0h	该字节为 A0，用于固件下载/上传供应商命令。
2-3	***wValue***	AddrL（LSB）	32 位程序入口
4-5	***wIndex***	AddrH（MSB）	32 位程序入口>>16
6-7	***wLength***	0	该字段必须为零。

对于跳到入口命令，Bootloader 将关闭所有的中断并断开同 USB 的连接。以下内容介绍的是三个供应商命令子程序的示例。

示例 1. 8 字节 Setup 数据包的供应商命令写数据协议：

```
bmRequestType=0x40
bRequest      = 0xA0;
wValue        = (WORD)address;
wIndex        = (WORD)(address>>16);
wLength       = 1 to 4K-byte max
```

该命令将发送长度为 wLength 的 DATA OUT 数据包和长度为零的 DATA IN 数据包。

示例 2. 使用 Setup 数据包读取 Bootloader 版本

```
bmRequestType= 0xC0
bRequest      = 0xA0;
wValue        = (WORD)0x0020;
wIndex        = (WORD)0xFFFF;
wLength       = 4
```

该命令将发送各个长度为 wLength 的 DATA IN 数据包和一个长度为零的 DATA OUT 数据包。

示例 3. 使用 8 字节 Setup 数据包跳到程序入口（请参考表 8）

```
bmRequestType= 0x40

bRequest      = 0xA0;
wValue        = Program Entry       (16-bit LSB)
wIndex        = Program Entry >>16       (16-bit MSB)
wLength       = 0
```

注意：FX2LP 仅使用 16 位寻址，但 FX3 使用的是 32 位寻址。需要将地址写入到命令的 wValue 和 wIndex 字段内。

4.2.11 USB 下载示例代码

若要下载代码，应用应读取固件镜像文件，并使用供应商写命令一次写入 4 K 的数据段。该段大小不能超过 Bootloader 所使用的缓冲区大小。

注意：固件镜像的格式必须与表 14 中指定的一样。

下面示例介绍了固件下载程序的实现。

```
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;
}
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 是指向固件镜像文件的 FILE 指针。该文件的格式如表 14 所示。

4.3 校验和计算

在 USB 下载中，下载工具将处理校验和计算，如 USB 下载示例代码一节所示。

4.3.1 FX3 Bootloader 存储器分配情况

在数据紧密耦合存储器（DTCM）中，FX3 Bootloader 会分配 1280 个字节（地址范围为 0x1000_0000 到 0x1000_04FF）用于变量和堆栈。只要在固件下载过程中，该存储器空间未被初始化（未初始化的局部变量），固件应用就能使用该空间。

Bootloader 所分配的前 16 个字节（从 0x4000_0000 到 0x4000_000F）用于热启动和待机启动。固件应用不应该使用这些字节。

Bootloader 所分配的 10K 个字节（从 0x4000_23FF 地址起）用于它的内部缓冲区。固件应用可将该空间作为未初始化的局部变量/缓冲区使用。

Bootloader 不使用指令紧密耦合存储器（ITCM）。

4.3.2 寄存器/存储器访问

FX3 Bootloader 允许对 ROM、MMIO、SYSMEM、ITCM 和 DTCM 存储器进行读访问。

Bootloader 允许对 MMIO、SYSMEM、ITCM 和 DTCM 存储器（DTCM 中前 1280 个字节和系统存储器中前 10 K 的空间除外）进行写访问。对 MMIO 进行写访问时，Bootloader 的传输长度必须为 4（等于长字），并且地址要为 4 字节对齐。

4.3.3 USB eFUSE VID/PID 启动选项

通过扫描 eFuse（eFUSE_USB_ID）来确定 USB_VID 位是否被编程，FX3 Bootloader 可以使用由用户选择的 VID 和 PID 进行启动。如果这些位被编程，则 Bootloader 将使用 eFUSE VID 和 PID 值。

4.3.4 USB OTG

FX3 Bootloader 不支持 OTG 协议。它始终作为一个 USB 总线供电式器件运行。

4.3.5 Bootloader 限制

FX3 Bootloader 处理受限制的地址范围检查。如果访问实际上不存在的地址，则会导致发生不可预测的结果。

Bootloader 不会检查程序入口。一个无效的程序入口可能导致发生不可预测的结果。

Bootloader 允许对 MMIO 寄存器进行写访问。写入无效地址会导致不可预测的结果。

4.3.6 USB 看门狗定时器

FX3 USB 硬件要求 USB 内核硬件使用频率为 32 kHz 的时钟输入。如果没有 32 kHz 的外部时钟，则 Bootloader 会将看门狗定时器配置为 USB 内核的内部 32 kHz 时钟。

4.3.7 USB 暂停/恢复

如果 USB 上没有发生任何操作，则 FX3 Bootloader 将进入暂停模式。当 PC 恢复 USB 上进行的操作时，FX3 Bootloader 将恢复正常操作。

4.3.8 USB 附加 PID

根据 PMODE 引脚的设置，Bootloader 可能用 VID=0x04B4/PID=0x00BC 或 VID=0x04B4/PID=0x0053 进行启动。

4.3.9 USB 墙壁充电器检测

如果将 FX3 连接至墙壁充电器，则 Bootloader 将进入暂停模式，并将 O[60]（充电器检测输出）引脚设置为逻辑‘1’。将 FX3 连接至 USB 主机时，Bootloader 将恢复正常操作，并将 O[60]引脚设置为逻辑‘0’。

4.3.10 USB 器件描述符

下面各表显示了高速和全速模式下的 FX3 Bootloader 描述符。

注意：在全速模式下，器件限定符不可用。

表 9. 器件描述符

偏移	字段	数值	说明
0	***bLength***	12h	该描述符的长度 = 18 个字节
1	***bDescType***	01	描述符类型 = 器件
2-3	***wBCDUSB***	0200h	USB 规范版本 2.0
4	***bDevClass***	00	器件类别（没有实现任何类别特定协议）
5	***bDevSubClass***	00	器件子类（没有实现任何类别特定协议）
6	***bDevProtocol***	00	器件协议（没有实现任何类别特定协议）
7	***bMaxPktSize***	40h	端点 0 数据包大小为 64
8-9	***wVID***	04B4h	赛普拉斯半导体 VID

偏移	字段	数值	说明
10-11	***wPID***	00F3h	FX3 芯片
12-13	***wBCDID***	0100h	FX3 bcdID
14	***iManufacture***	01h	制造商索引字符串 = 01
15	***iProduct***	02h	序列号索引字符串 = 02
16	***iSerialNum***	03h	序列号索引字符串 = 03
17	***bNumConfig***	01h	单一配置

表 10. 器件限定符

偏移	字段	数值	说明
0	***bLength***	0Ah	该描述符的长度 = 10 个字节
1	***bDescType***	06	描述符类型 = 器件限定符
2-3	***wBCDUSB***	0200h	USB 规范版本 2.00
4	***bDevClass***	00	器件类别（没有实现任何类别特定协议）
5	***bDevSubClass***	00	器件子类（没有实现任何类别特定协议）
6	***bDevProtocol***	00	器件协议（没有实现任何类别特定协议）
7	***bMaxPktSize***	40h	端点 0 数据包大小为 64
8	***bNumConfig***	01h	单一配置
9	***bReserved***	00h	必须为零

表 11. 配置描述符

偏移	字段	数值	说明
0	***bLength***	09h	该描述符的长度 = 10 个字节
1	***bDescType***	02h	描述符类型 = 配置
2-3	***wTotalLength***	0012h	总长度
4	***bNumInterfaces***	01	此配置中的接口数量
5	***bConfigValue***	01	SetConfiguration 请求所使用的配置值，用于选择该接口
6	***bConfiguration***	00	描述该配置的字符串索引 = 0
7	***bAttribute***	80h	属性：总线供电，无唤醒
8	***bMaxPower***	64h	最大功率：200 mA

表 12. 接口描述符（备用设置 0）

偏移	字段	数值	说明
0	***bLength***	09h	该描述符的长度 = 10 个字节
1	***bDescType***	04h	描述符类型 = 接口
2	***bInterfaceNum***	00h	该接口的索引（从零开始） = 0
4	***bAltSetting***	00	备用的设置值 = 0

偏移	字段	数值	说明
5	***bNumEndpoints***	00	仅使用端点 0
6	***bInterfaceClass***	FFh	供应商命令类别
7	***bInterfaceSubClass***	00h	
8	***bInterfaceProtocol***	00h	
9	***iInterface***	00h	无

表 13. 字符串描述符

偏移	字段	数值	说明
0	***bLength***	04h	该描述符的长度 = 04 个字节
1	***bDescType***	03h	描述符类型 = 字符串
2-3	***wLanguage***	0409h	语言 = 英语
4	***bLength***	10h	该描述符的长度 = 16 个字节
5	***bDescType***	03h	描述符类型 = 字符串
6-21	***wStringIdx1***	–	“赛普拉斯”
22	***bLength***	18h	该描述符的长度 = 24 个字节
23	***bDescType***	03h	描述符类型 = 字符串
24-47	***wStringIdx2***	–	“WestBridge”
48	***bLength***	1Ah	该描述符的长度 = 26 个字节
49	***bDescType***	03h	描述符类型 = 字符串
50-75	***wStringIdx3***	–	“0000000004BE”

4.4 启动镜像格式

对于 USB 启动，Booloader 要求固件镜像文件格式与表 14 中显示的格式相同。EZ-USB FX3 SDK 提供了一个软件工具，通过该工具可生成一个具有 USB 启动所需的格式的固件镜像。安装 SDK 后，请参考位于 *C:\Program Files\Cypress\EZ-USB FX3 SDK\1.3\util\elf2img* 目录中的 elf2img 工具。对于 64 位的操作系统，路径中的第一个文件夹为“Program Files(x86)”。目录路径中的 *1.3* 数值是指 SDK 的版本编号，该数值可根据 FX3 SDK 的最新发布版本而变。

表 14. 启动镜像文件的格式

二进制镜像头文件	长度（16 位）	说明
wSignature	1	两个字节标签使用 ASCII 文本 “CY” 初始化 。
bImageCTL;	½	位 0 = 0：执行二进制文件；1：数据文件类型 位 3:1：进行 SPI EEPROM 启动时不使用 位 5:4（SPI 速度）： 00：10 MHz 01：20 MHz 10：30 MHz 11：保留 位 7:6：保留，并要设置为零

二进制镜像头文件	长度（16 位）	说明
bImageType;	½	bImageType = 0xB0：带校验和的普通 FW 二进制镜像 bImageType = 0xB2：使用新的 VID 和 PID 进行 I²C/SPI 启动
dLength 0	2	第一段长度，单位为长字（32 位） 当 bImageType = 0xB2 时，dLength 0 包含 PID 和 VID。Bootloader 忽略后面的所有数据。
dAddress 0	2	编程代码的第一段地址。 注意：内部 ARM 地址是字节可寻址的，因此，每一段的地址应该为 32 位对齐。
dData[dLength 0]	dLength 0*2	镜像代码/数据必须为 32 位对齐。
…		更多的段
dLength N	2	0x00000000（最后记录：终端段）
dAddress N	2	应该包含有效的程序入口（通常为启动代码，即 RESET 向量）。 **注意**：如果 bImageCTL.bit0 = 1，Bootloader 不会跳到该程序入口。 如果 bImageCTL.bit0 = 0，Bootloader 会跳到该程序入口。该地址应位于 ITCM 或 SYSTEM RAM 内。 Bootloader 不会验证程序入口。
dCheckSum	2	低位优先的 32 位无符号校验和数据占用范围是从第一段到终端段。校验和不包含 dLength、dAddress 和镜像头文件。

4.4.1 下面是以长字格式组织的启动镜像示例：

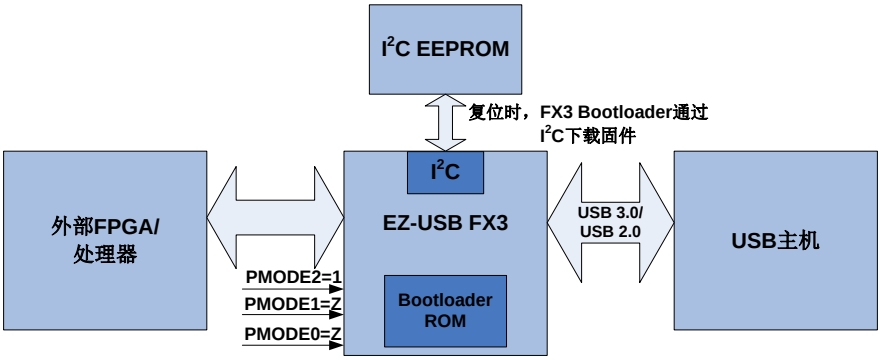
```
Location1: 0xB0 0x10 'Y' 'C'   //CY Signature, 20 MHz, 0xB0 Image
Location2: 0x00000004          //Image length of section 1 = 4
Location3: 0x40008000          //1st section stored in SYSMEM RAM
at 0x40008000
Location4: 0x12345678          //Image starts (Section1)
Location5: 0x9ABCDEF1
Location6: 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
Location 14: 0x6AF37AF2        //Checksum (0x12345678 +
                    0x9ABCDEF1 + 0x23456789 +
                          0xABCDEF12+ 0xDDCCBBAA +0x11223344)
```

附录中的 USB 启动部分显示了使用 FX3 DVK 测试 USB 启动模式的各个步骤。

5 I²C EEPROM 启动

图 3 显示了 FX3 通过 I²C 启动的系统框图。

图 3. FX3 通过 I²C 启动的系统框图



对于 I²C EEPROM 启动，PMODE[2:0]引脚的状态应该为 *1ZZ*，如表 15 所示。

表 15. I²C 启动的 PMODE 引脚

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

表 16 显示了 FX3 I²C 接口的引脚映射情况。

表 16. I²C 接口的引脚映射情况

EZ-USB FX3 引脚	I²C 接口
I2C_GPIO[58]	I2C_SCL
I2C_GPIO[59]	I2C_SDA

5.1 功能

- 通过双线 I²C 接口从 I²C EEPROM 器件启动 FX3。
- 支持下面 EEPROM[2]器件大小：
  - 32 千比特（Kb）或 4 千字节（KB）
  - 64 Kb 或 8 KB
  - 128 Kb 或 16 KB
  - 256 Kb 或 32 KB
  - 512 Kb 或 64 KB
  - 1024 Kb 或 128 KB
  - 2048 Kb 或 256 KB

[2]只支持 2 字节 I²C 地址。对于尺寸小于 32 Kb 的 I²C EEPROM，不支持单字节地址。

注意：由于释放版本中所生成的镜像文件尺寸小于调试版本中的尺寸，建议使用释放模式中所创建的固件镜像。

- 已经对 ATMEL、Microchip 和 ST Electronics 器件进行测试。
- 启动过程中，支持 100 kHz、400 kHz 和 1 MHz 的 I²C 频率。请注意，当 V_{IO5} 为 1.2 V 时，支持的最大工作频率为 100 kHz。当 V_{IO5} 为 1.8 V、2.5 V 或 3.3 V 时，支持的工作频率为 400 kHz 和 1 MHz（V_{IO5} 是 I²C 接口的 I/O 电压）。
- 支持从多个大小相同的 I²C EEPROM 器件进行启动。当 I²C EEPROM 小于固件镜像时，必须使用 I²C EEPROM 器件。Bootloader 支持从多个 I²C EEPROM 器件下载镜像。超高速 Explorer CYUSB3KIT-003 使用 ST Electronics 的 256 KB EEPROM（M24M02）。Bootloader 支持多达八个小于 128 KB 的 I²C EEPROM 器件。Bootloader 可支持多达四个大小为 128 KB 的 I²C EEPROM 器件。
- I²C EEPROM 中仅存储唯一一个固件镜像。不允许存储冗余镜像。
- Bootloader 不支持 FX3 的多主设备 I²C 特性。因此，在 FX3 I²C 启动过程中，其他 I²C 主设备不能在 I²C 总线上执行任何操作。

5.2 在 EEPROM 上存储固件镜像

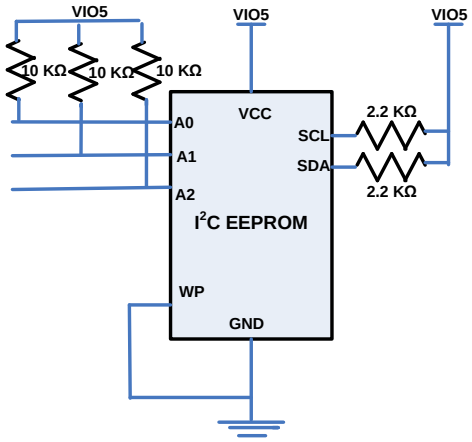
FX3 Bootloader 支持主设备 I²C 接口，用于与外部串行 I²C EEPROM 器件连接。串行 I²C EEPROM 可用于存储应用特定的代码和数据。图 4 显示了典型 I²C EEPROM 的引脚连接。

I²C EEPROM 接口包括两条有源线：串行时钟线路（SCL）和串行数据线路（SDA）。

将固件镜像写入到 EEPROM 时，需要将写保护（WP）引脚置低。

A0、A1 和 A2 引脚均是地址线。它们将从设备地址设置为 000 ~ 111。这样，可以寻址八个大小相同的 I²C EEPROM。根据所需地址，可将这些地址线置于高电平或低电平。

图 4. 典型 I²C EEPROM 的引脚连接



5.2.1 寻址 128 KB EEPROM 时应当注意的事项

各 128 KB I²C EEPROM 的寻址方式并非一样的。例如，Microchip EEPROM 将引脚 A1 和 A0 用于芯片选择，并不使用引脚 A2。但 Atmel EEPROM 会将引脚 A2 和 A1 用于芯片选择，不使用引脚 A0。这两种情况均由 Bootloader 处理。固件镜像头文件中会表明寻址方式。

表 17 显示了如何连接四个 Microchip 24LC1025 EEPROM 器件。

表 17.Microchip 24LC1025 EEPROM 器件连接

器件序号	地址范围	A2 A1 A0	大小
1	0x00000-0x1FFFF	Vcc 0 0	128 KB
2	0x20000-0x3FFFF	Vcc 0 1	128 KB
3	0x40000-0x5FFFF	Vcc 1 0	128 KB
4	0x60000-0x7FFFF	Vcc 1 1	128 KB

表 18 显示了如何连接四个 Atmel 24C1024 EEPROM 器件。

表 18. ATMEL 24C1024 EEPROM 器件连接

器件序号	地址范围	A2 A1 A0	大小
1	0x00000-0x1FFFF	0 0 NC	128 KB
2	0x20000-0x3FFFF	0 1 NC	128 KB
3	0x40000-0x5FFFF	1 0 NC	128 KB
4	0x60000-0x7FFFF	1 1 NC	128 KB

注意：NC 表示无连接状态。

例如，如果固件代码大于 128 KB，那么您必须使用两个 I²C EEPROM（具有与其相应的寻址方案），如前两个表格所示。与单一 I²C EEPROM 相同，需要在各个 EEPROM 中将固件镜像存储为一个连续的镜像。

5.3 启动镜像格式

Booloader 要求固件镜像文件的格式与表 19 中所示的格式相同。EZ-USB FX3 SDK 提供了一个软件工具，用于生成一个具有 I²C EEPROM 启动所需要的格式的固件镜像。安装 SDK 后，请参考位于 *C:\Program Files\Cypress\EZ-USB FX3 SDK\1.3\util\elf2img* 目录中的 elf2img 工具。对于 64 位的操作系统，路径中的第一个文件夹为“Program Files(x86)”。目录路径中的 *1.3* 数值是指 SDK 的版本编号，该数值可根据 FX3 SDK 的最新发布版本而变。

表 19. 固件镜像存储格式

二进制镜像头文件	长度（16 位）	说明
WSignature	1	两个字节标签使用 ASCII 文本 “CY” 初始化 。
bImageCTL;	½	位 0 = 0：执行二进制文件；1：数据文件类型 位 3:1（I²C 大小） 7：128 KB（microchip） 6：64 KB（128K ATMEL 和 256K ST Electronics） 5：32 KB 4：16 KB 3：8 KB 2：4 KB **注意：** 保留选项 1 和 0，以供将来使用。如果在这些模式下启动，会导致不可预测的结果。 位 5:4（I²C 速度）： 00：100 KHz 01：400 kHz 10：1 MHz 11：保留 **注意：** 默认情况下，Bootloader 上电速度为 100 kHz。根据要求，I²C 速度可被调整。 位 7:6：保留；应设置为零
bImageType;	½	bImageType = 0xB0：带校验和的普通 FW 二进制镜像 bImageType = 0xB2：使用新的 VID 和 PID 进行 I²C 启动
dLength 0	2	第一段长度，单位为长字（32 位） 当 bImageType = 0xB2 时，dLength 0 包含 PID 和 VID。Bootloader 忽略后面的所有数据。
dAddress 0	2	编程代码中的第一段地址，并不是 I²C 地址。 **注意：** 内部 ARM 地址是字节可寻址的，因此，每一段的地址应该为 32 位对齐。
dData[dLength 0]	dLength 0*2	所有的镜像代码/数据都必须为 32 位对齐。
…		更多的段
dLength N	2	0x00000000（最后记录：终端段）
dAddress N	2	应该包含有效的程序入口（通常为启动代码，即 RESET 向量）。 **注意：** 如果 bImageCTL.bit0 = 1，Bootloader 不会跳到该程序入口。 如果 bImageCTL.bit0 = 0，Bootloader 会跳到该程序入口。该地址应位于 ITCM 或 SYSTEM RAM 内。 Bootloader 不会验证程序入口
dCheckSum	2	低位优先的 32 位无符号校验和数据占用范围是从第一段到终端段。校验和不包含 dLength、dAddress 和镜像头文件

例如：在 I²C EEPROM 中使用下面的顺序存储二进制镜像文件：

位 0：“C”

位 1：“Y”

位 2：bImageCTL

位 3：bImageType

…..

位 N：镜像的校验和

重要注意：

- Bootloader 的默认启动速度为 100 kHz；如果要将速度从 100 kHz 调整为 1 MHz，bImageCTL<5:4>需要设置为 10。
- 使用 bImageCTL[3:1]可选择 I²C EEPROM 的大小。

  Microchip EEPROM 24LC1026 的寻址与其他 128 KB Microchip EEPROM 的寻址不同。如果使用 Microchip 24LC1026，I²C EEPROM 大小字段（例如 bImageCTL[3:1]）应该被设置为 6。

5.4 校验和计算

当下载二进制镜像 I²C EEPROM 时，Bootloader 将计算校验和。如果校验和与镜像中的值不相匹配，则 Bootloader 不会跳到程序入口内。

Bootloader 在低位优先模式下运行；因此，必须在低位优先模式下计算校验和。

低位优先的 32 位无符号校验和数据占用范围是从第一段到终端段。校验和不包含 dLength、dAddress 和镜像头文件。

5.4.1 第一个示例启动镜像

以下的镜像仅存储在 FX3 系统 RAM 中地址 0x40008000 的一个段内：

```
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 第二个示例启动镜像

以下镜像存储在 FX3 系统 RAM 中地址 0x40008000 和 0x40009000 的两个段内：

```
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)
```

同样，您可以使用一个启动镜像存储某个镜像的 N 个段。

附录中的 I²C 启动部分显示了使用 FX3 DVK 测试 USB 启动模式的各步骤。

5.4.3 校验和计算示例代码

以下内容介绍了校验和示例代码：

```
// 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;
      }
```

本节描述了 I²C 启动选项的详细信息。下一节将介绍使用 USB 回退功能时的 I²C 启动选项。

6 带有 USB 回退功能的 I²C EEPROM 启动

对于带有 USB 回退功能的 I²C EEPROM 启动，PMODE[2:0]引脚的状态应为 Z1Z，如表 20 中所示。

表 20. 带有 USB 回退功能的 I²C 启动的 PMODE 引脚

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

在所有 USB 回退功能模式中（表示为“-->USB”），如果选择了 0xB2 启动或发生错误，则 USB 将进行枚举。USB 枚举后，外部 USB 主机可使用 USB 启动模式启动 FX3。此外，还会使用带有 USB 回退功能的 I²C EEPROM 启动（I²C --> USB）来存储 USB 启动所需的供应商标识（VID）和产品标识（PID）。

I²C EEPROM 启动操作会在下面的条件中失败：

- I²C 地址周期或数据周期错误
- FX3 固件镜像中的无效标签
- 无效的镜像类型

特殊的镜像类型可用于表示 EEPROM 中的数据是 USB 启动时使用的 VID 和 PID，而不是 FX3 固件镜像。这样，可以给 USB 启动提供新的 VID 和 PID。

6.1 功能

- 对于 USB 启动，Bootloader 仅支持 USB 2.0，并不支持 USB 3.0。
- 如果指定了 0xB2 启动选项，则 USB 描述符将使用客户定义的 VID 和 PID。这些信息作为 0xB2 镜像的一部分，存储在 I²C EEPROM 中。
- 对于 USB 回退功能，如果在 I²C 启动时发生任何错误，USB 描述符将使用 VID=0x04B4 和 PID=0x00F3。
- USB 器件描述符被记录为总线供电，并消耗大约 200 mA 的电流。但在一般情况下，FX3 芯片只消耗大约 100 mA 的电流。

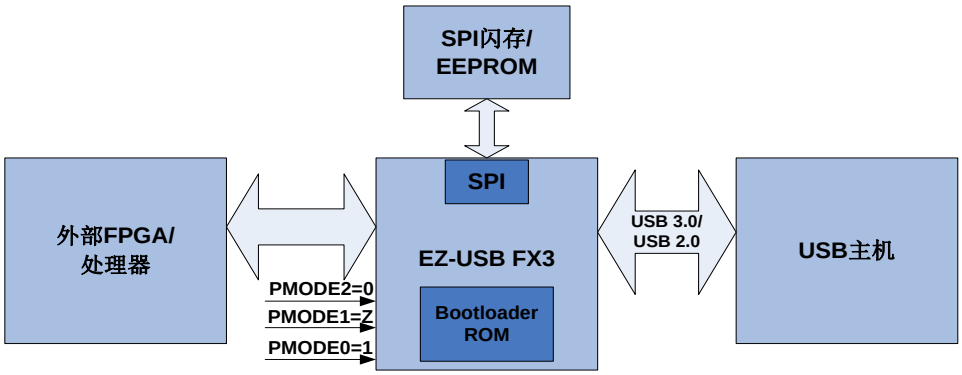
6.2 使用 VID 和 PID 启动的示例镜像

```
Location1: 0xB2 0x1A 'Y' 'C'      //CY Signature,32k EEPROM,400Khz,0xB2 Image

Location2: 0x04B40008             // VID = 0x04B4 | PID=0x0008
```

7 SPI 启动

图 5 显示了 FX3 通过 SPI 启动时的系统框图。

图 5. SPI 启动的系统框图



对于 SPI 启动，PMODE[2:0]引脚的状态应该为 0Z1，如表 21 所示。

表 21. SPI 启动的 PMODE 引脚

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

表 22 显示了 FX3 SPI 接口的引脚映射情况。

表 22. SPI 接口的引脚映射情况

EZ-USB FX3 引脚	SPI 接口
GPIO[53]	SPI_SCK
GPIO[54]	SPI_SSN
GPIO[55]	SPI_MISO
GPIO[56]	SPI_MOSI

7.1 功能

FX3 通过四线 SPI 接口从 SPI 闪存/EEPROM 器件启动。

- 支持大小为 1 Kb 到 128 Mb 的 SPI 闪存/EEPROM 器件用于启动。
- 支持的 SPI Flash 部件：
  - Cypress SPI Flash (S25FS064S (64-Mbit), S25LFL064L (64-Mbit) and S25FS128S (128-Mbit))
  - Winbond W25Q32FW (32-Mbit)
- 启动时，支持~10 MHz、~20 MHz 和~30 MHz 的 SPI 频率。
  请注意，由于对 SPI 时钟分频器和时钟输入进行了取整，因此 SPI 频率可能发生改变。
  - 如果 FX3 的晶振或时钟输入的频率为 26 MHz 或 52 MHz，那么内部 PLL 将以 416 MHz 的频率运行。PLL_CLK = 416 MHz 时，SPI 的频率可为 10.4 MHz、20.8 MHz 或 34.66 MHz。
  - 当 FX3 的晶振或时钟输入的频率为 19.2 MHz 或 38.4 MHz 时，内部 PLL 将以 384 MHz 的频率运行。PLL_CLK = 384 MHz 时，SPI 的频率可为 9.6 MHz、19.2 MHz 或 32 MHz。

- 支持的工作电压为 1.8 V、2.5 V 和 3.3 V。
- 在 SPI 闪存/EEPROM 中仅存储唯一一个固件镜像。不允许存储冗余的镜像。
- 对于 SPI 启动，Bootloader 将 CPOL 设置为 0，将 CPHA 设置为 0。（有关该 SPI 模式的时序框图，请参考 FX3 数据手册中的 SPI 时序。）
- 支持 USB 回退功能，用于存储 USB 启动所需的新 VID/PID 信息。更多信息，请参考本应用笔记中的带有 USB 回退功能的 SPI 启动一节。

7.2 SPI 闪存的选择

SPI 闪存需要支持以下命令，用以支持 FX3 启动。

- 读取数据：03h，并使用 3 字节进行寻址
- 读状态寄存器：05h
- 写入使能：06h
- 写入数据（页编程）：02h
- 扇区擦除：D8h

只要读命令匹配，可使用一个 SPI 闪存进行 FX3 启动。如果写命令存在差异，那么使用所提供的 *CyBootProgrammer.img*（位于 *C:\Program Files (x86)\Cypress\Cypress USBSuite\application\c_sharp\ controlcenter* 内）便不能成功编程 SPI 闪存；这时需要更改 FX3 SDK 的 USBFlashProg 示例项目中使用的 SPI 写命令。使用编译更改好的 USBFlashProg 项目后所创建的镜像文件替代掉所提供的 *CyBootProgrammer.img*（名称相同），以便成功编程 SPI 闪存。

7.3 在 SPI 闪存/EEPROM 中存储固件镜像

FX3 Bootloader 支持一个主设备 SPI 控制器，用于连接至外部串行 SPI 闪存/EEPROM 器件。SPI 闪存/EEPROM 用于存储应用特定的代码和数据。 图 6 显示了典型 SPI 闪存/EEPROM 的引脚布局。

SPI EEPROM 接口包括四条有源线：

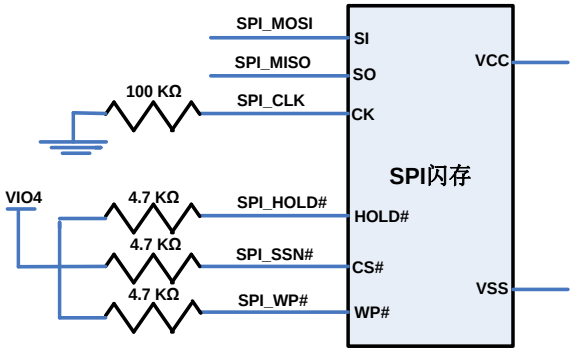
- CS#：芯片选择
- SO：串行数据输出（主入从出（MISO））
- SI：串行数据输入（主出从入（MOSI））
- SCK：串行时钟输入

从 SPI 器件启动或读取时，需要将 HOLD#信号连接至 VCC

将镜像写入到 EEPROM 时，需要将写保护（WP#）和 HOLD#信号连接至 VCC。

请注意，在 MOSI 和 MISO 信号上不应连接外部上拉，如图 6 所示。

图 6. 典型 SPI 闪存的引脚连接



7.4 启动镜像格式

对于 SPI 启动，Booloader 要求固件镜像文件与表 23 中所示的格式相同。EZ-USB FX3 SDK 提供了一个软件工具，用于生成一个具有 SPI 启动所需的格式的固件镜像。安装 SDK 后，请参考位于 *C:\Program Files\Cypress\EZ-USB FX3 SDK\1.3\util\elf2img* 目录中的 elf2img 工具。对于 64 位的操作系统，路径中的第一个文件夹为“Program Files(x86)”。目录路径中的 *1.3* 数值是指 SDK 的版本编号，该数值可根据 FX3 SDK 的最新发布版本而变。

表 23. SPI 启动选项的启动镜像格式

二进制镜像头文件	长度（16 位）	说明
wSignature	1	两个字节标签使用 ASCII 文本 “CY” 初始化 。
bImageCTL	½	位 0 = 0：执行二进制文件；1：数据文件类型 位 3:1 进行 SPI 启动时不使用 位 5:4（SPI 速度）： 00：10 MHz 01：20 MHz 10：30 MHz 11：保留 **注意**：默认情况下，Bootloader 上电时的速度为 10 MHz。根据要求， SPI 速度可被调整。FX3 SPI 硬件的运行速度不能超过 33 MHz。 位 7:6：保留。应将其设置为零。
bImageType	½	bImageType = 0xB0：带校验和的普通固件二进制镜像 bImageType = 0xB2：使用新的 VID 和 PID 进行 SPI 启动
dLength 0	2	第一段长度，单位为长字（32 位） 当 bImageType = 0xB2 时，dLength 0 包含 PID 和 VID。Bootloader 忽略后面的所有数据。
dAddress 0	2	编程代码的第一段地址。 注意：内部 ARM 地址是字节可寻址的，因此，每一段的地址应该为 32 位对齐。
dData[dLength 0]	dLength 0*2	镜像代码/数据必须为 32 位对齐。
…		更多的段
dLength N	2	0x00000000（最后记录：终端段）
dAddress N	2	应该包含有效的程序入口（通常为启动代码，即 RESET 向量）。 **注意：** 如果 bImageCTL.bit0 = 1，Bootloader 不会跳到该程序入口。 如果 bImageCTL.bit0 = 0，Bootloader 将跳到程序入口：该地址应该位于 ITCM 或 SYSTEM RAM 内。 Bootloader 不会验证程序入口。
dCheckSum	2	低位优先的 32 位无符号校验和数据占用范围是从第一段到终端段。校验和不包含 dLength、dAddress 和镜像头文件。

例如： 在 SPI EEPROM 中以下面顺序存储二进制镜像文件：

位 0：“C”

位 1：“Y”

位 2：bImageCTL

位 3：bImageType

…..

位 N：镜像的校验和

重要的注意事项：

Bootloader 的默认启动速度为 10 MHz；想要将速度从 10 MHz 改为 20 MHz，需要将 bImageCTL[5:4]设置为 01。

7.5 校验和计算

当通过 SPI 下载二进制镜像时，Bootloader 将计算校验和。如果校验和与镜像中的值不匹配，Bootloader 不会跳到程序入口。

Bootloader 在低位优先模式下运行；因此，必须在低位优先模式下计算校验和。

低位优先的 32 位无符号校验和数据占用范围是从第一段到终端段。校验和不包含 dLength、dAddress 和镜像头文件。有关计算校验和的示例代码的信息，请参考**校验和计算示例代码**一节。

示例 1. 下面示例显示了固件镜像仅存储在 FX3 系统 RAM 中地址 0x40008000 的一个段内。

```
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)
```

示例 2. 下面的示例显示了一个固件镜像被存储在 FX3 系统 RAM 中地址 0x40008000 和 0x40009000 的两个段内。

```
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)
```

同样，您可以使用一个启动镜像存储某个镜像的 N 个段。

附录中的 SPI 启动部分显示了使用 FX3 DVK 测试 USB 启动模式的各步骤。

8 带有 USB 回退功能的 SPI 启动

在所有的 USB 回退功能（“-->USB”）模式中，如果选择 0xB2 启动或发生错误，USB 将进行枚举。在 USB 枚举后，外部 USB 主机可使用 USB 启动模式启动 FX3。此外，还能使用带 USB 回退功能的 SPI 启动（SPI --> USB）来存储 USB 启动所需的 VID 和 PID。

SPI 启动在下面条件下操作失败：

- SPI 地址周期或数据周期错误
- FX3 固件中的无效标签。无效的镜像类型

特殊的镜像类型可用于表示 SPI 闪存/EEPROM 中的数据是 USB 启动时使用的 VID 和 PID，而不是 FX3 固件镜像。这样，可以给 USB 启动提供新的 VID 和 PID。

- 对于 USB 启动，Bootloader 仅支持 USB 2.0，并不支持 USB 3.0。
- 如果指定了 0xB2 启动选项，USB 描述符将使用客户定义的 VID 和 PID。这些信息作为 0xB2 镜像的一部分，存储在 SPI 闪存/EEPROM 中。
- 对于 USB 回退功能，如果在 I²C 启动时发生任何错误，USB 描述符将使用 VID=0x04B4 和 PID=0x00F3。
- USB 器件描述符被记录为总线供电，消耗大约 200 mA 的电流。但在一般情况下，FX3 芯片只消耗大约 100 mA 的电流。

8.1 使用 VID 和 PID 启动的示例镜像

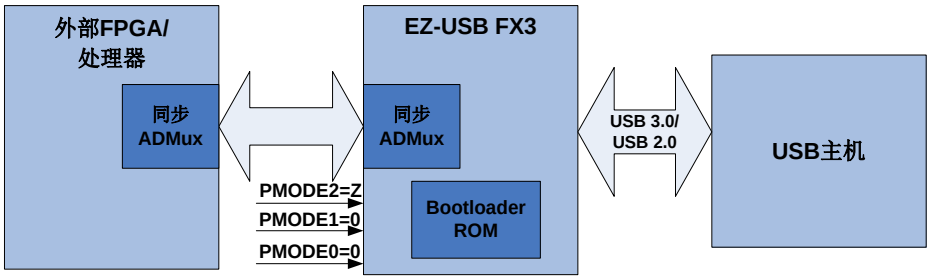
```
Location1: 0xB2 0x10 'Y' 'C'      //CY Signature, 20 MHz, 0xB2 Image

Location2: 0x04B40008             //VID = 0x04B4 | PID = 0x0008
```

下一节将介绍同步 ADMux 接口以及通过该接口启动的详细信息。

9 同步 ADMux 启动

图 7 显示了通过同步 ADMux 接口启动的 FX3 系统框图。

图 7. 同步 ADMux 启动的系统框图



想要通过同步 ADMux 接口启动，那么应将 PMODE[2:0]引脚的状态设置为 Z00，如表 24 所示。

表 24. 同步 ADMux 启动的 PMODE 引脚

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

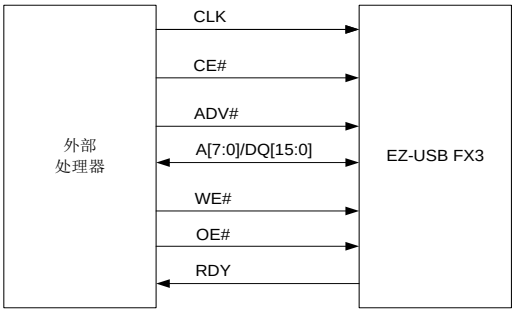
FX3 GPIF II 接口支持一个同步 ADMux 接口，该接口用于从外部处理器或 FPGA 下载固件镜像。由 Bootloader 配置的同步 ADMux 接口包括以下各信号：

- PCLK：该信号必须作为 FX3 的时钟输入使用。支持的时钟输入频率最大为 100 MHz
- DQ[15:0]：16 位数据总线
- A[7:0]：8 位地址总线
- CE#：低电平有效的芯片使能
- ADV#：低电平有效的地址有效
- WE#：低电平有效的写使能
- OE#：低电平有效的输出使能
- RDY：高电平有效的就绪信号

9.1.1 接口信号

图 8 中显示了由 Bootloader 配置并连接至外部处理器的同步 ADMux 接口的典型互联框图。

图 8. 同步 ADMUX 接口



进行读操作时，必须激活 CE#和 OE#。

进行写操作时，则要激活 CE#和 WE#。

在读/写操作的地址阶段内，必须将 ADV#置为低电平。在读/写操作的数据阶段内，则必须将 ADV#置于高电平。

FX3 器件的 RDY 输出信号表示数据有效，并能够对其进行读取。

表 25 显示了 FX3 同步 ADMux 接口的引脚映射情况。

表 25. 同步 ADMux 接口的引脚映射情况

EZ-USB FX3 引脚	同步 ADMUX 接口
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 同步 ADMux 时序

有关同步 ADMux 时序图（同步 ADMux 接口 — 读周期时序和写周期时序）以及时序参数的详细信息，请参考图 9、图 10 和表 26。

同步 ADMUX 模式上电延迟

上电时或在 RESET#线上发生硬复位时，Bootloader 需要花费一段时间来配置实现同步 ADMux 接口的 GPIF II。大概需要几百微秒的时间。仅在配置好 Bootloader 后，才能对 FX3 进行读/写访问。否则，将损坏数据。为了防止发生这种情况，请使用以下方案中的一个：

- 取消激活 RESET#后再等待 1 ms。
- 持续轮询 PP_IDENTIFY 寄存器，直到回读 0x81 值为止。
- 等待 INT#信号被激活，然后读取 RD_MAILBOX 寄存器，并验证回读的值是否等于 0x42575943（即‘CYWB’）。

9.1.3 USB 回退功能（--> USB）

在同步 ADMUX 启动过程中，即使命令发生了错误，USB 回退功能也不会有效。

9.1.4 热启动

当检测到热启动时，Bootloader 会跳到上次存储的“程序入口”（可能就是用户的 RESET 向量）处。在这种情况下，GPIF II 配置被保留。

9.1.5 唤醒/待机

从待机模式唤醒后，应用固件将配置并恢复硬件寄存器、GPIF II 配置、ITCM 或 DTCM。

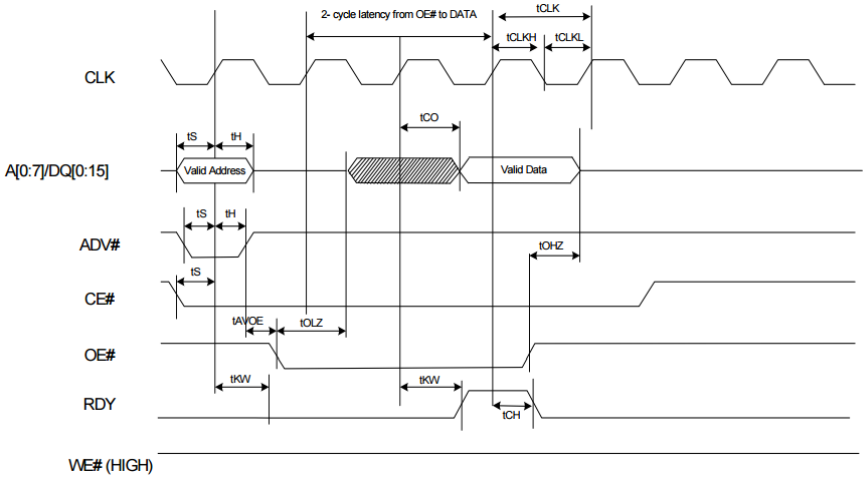
从待机模式唤醒后，Bootloader 将检查已被使能的 ITCM 和 DTCM。

注意：Bootloader 从待机模式唤醒或从热启动过程唤醒时，它会跳到复位中断服务子程序，并执行以下操作：

- 使 DCACHE 和 ICACHE 无效
- 打开 ICACHE
- 禁用 MMU
- 打开 DTCM 和 ITCM
- 使用 DTC 设置堆栈

Bootloader 分配了 0x500 字节（范围为 0x1000_0000 到 0x1000_04FF），因此，0x1000_0500 到 0x1000_1FFF 的地址空间可用于下载固件。当下载应用运行时，可将存储器中 0x1000_0000 到 0x1000_04FF 的地址空间用于其他目的。

图 9. 同步 ADMux 接口 — 读周期时序



注意：

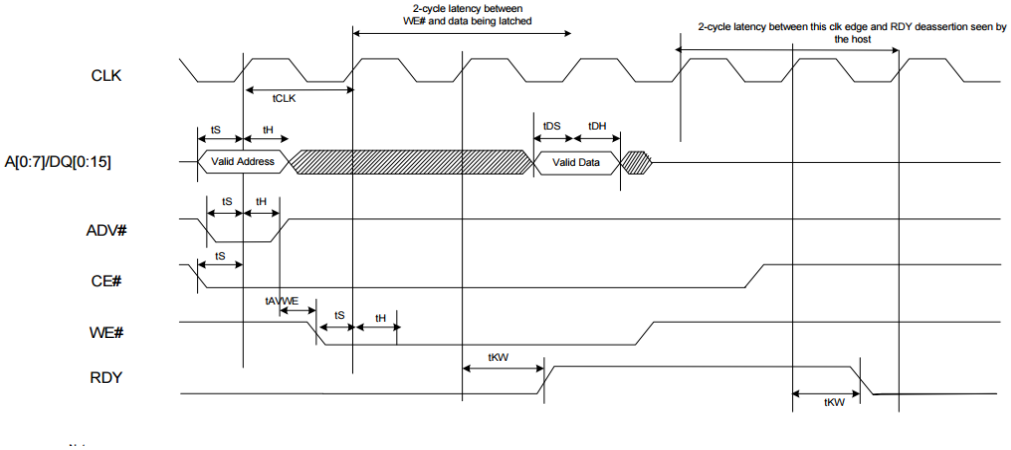
1) 外部 P-Port 处理器和 FX3S 在同一个时钟沿上运行。

2) 外部处理器看到在 OE#激活两个周期后，RDY 激活，并看到数据输出一个周期后，RDY 取消激活。

3) OE#激活后，有效的输出数据存在两个周期。数据一直保持到 OE#取消激活为止。

4) 对于 0-100 MHz 的操作，延迟等于两个周期。对于小于 50 MHz 的操作，延迟可减少为一个周期。（Bootloader 不支持一周期的延迟）

图 10. 同步 ADMux 接口 — 写周期时序



注意：

1) 外部 P-Port 处理器和 FX3S 在同一个时钟沿上运行。

2) 外部处理器看到在 WE#激活两个周期后，RDY 激活，并看到在采样数据三个周期后，RDY 取消激活。

3) 对于 0-100 MHz 的操作，延迟等于两个周期。对于小于 50 MHz 的操作，延迟可减少为一个周期。（Bootloader 不支持一周期的延迟）

表 26. 同步 ADMux 时序参数

参数	说明	最小值	最大值	单位
FREQ	接口时钟频率	–	100	MHz
tCLK	时钟周期	10	–	ns
tCLKH	时钟高电平时间	4	–	ns
tCLKL	时钟低电平时间	4	–	ns
tS	CE#/WE#/DQ 的建立时间	2	–	ns
tH	CE#/WE#/DQ 的保持时间	0.5	–	ns
tCH	时钟上升沿后数据输出保持的时间	0	–	ns
tDS	数据输入的建立时间	2	–	ns
tDH	时钟上升沿后数据输入保持的时间	0.5	–	ns
tAVDOE	从 ADV#为高电平到 OE#为低电平的时间	0	–	ns
tAVDWE	从 ADV#为高电平到 WE#为低电平的时间	0	–	ns
tHZ	从 CE#为高电平到数据为高阻态的时间	–	8	ns
tOHZ	从 OE#为高电平到数据为高阻态的时间	–	8	ns
tOLZ	从 OE#为低电平到数据为高阻态的时间	0	–	ns
tKW	时钟上升沿后到 RDY 有效的时间	–	8	ns

9.1.6 GPIF II API 协议

该协议仅在 GPIF II 启动模式下使用。复位后，外部应用处理器（AP）会使用表 27 中定义的命令协议与 Bootloader 进行通信。

表 27. GPIF II API 协议

字段	说明
bSignature[2]	**2 个字节** 发送方使用“CY”初始化 Bootloader 使用“WB”响应
bCommand	发送方：**1 个字节**的命令 0x00：NOP 0x01：WRITE_DATA_CMD：写数据命令 0x02：进入启动模式 0x03：READ_DATA_CMD：读数据命令 Bootloader 忽略其他所有命令，并返回 bLenStatus 中的错误代码
bLenStatus	**输入：（1 个字节）** 00：bLenStatus = 0（如果 bCommand 为 WRITE_DATA_CMD，Bootloader 将跳到 dAddr 中的 addr，并忽略其他所有命令） 01：表示长度，单位为长字（最大值为（512-8）/4） 02：512 字节大小的块的数量（最大值为 16） 03：表示长度，单位为长字（最大值为（512-8）/4） Bootloader 在 PIB_RD_MAILBOX1 寄存器中响应下述数据： 0x00：成功

字段	说明
	0x30：命令过程中发生错误，导致操作失败 0x31：读取操作发生错误，导致操作失败 0x32：中止检测 0x33：Bootloader 向应用发送的 PP_CONFIG.BURSTSIZE 邮件箱提示。PIB_RD_MAILBOX0 包含 GPIF_DATA_COUNT_LIMIT 寄存器。 0x34：Bootloader 检测 DLL _LOST_LOCK。PIB_RD_MAILBOX0 包含 PIB_DLL_CTRL 寄存器。 0x35：Bootloader 检测 PIB_PIB_ERR 位。PIB_RD_MAILBOX0 包含 PIB_PIB_ERROR 寄存器。 0x36：Bootloader 检测 PIB_GPIF_ERR 位。PIB_RD_MAILBOX0 包含 PIB_PIB_ERROR 寄存器。
dAddr	**4 字节** 发送方：由命令 1 和 3 使用的地址
dData[bLenStatus]	由 bLenStatus 确定的数据长度 发送方：由发送方写入的数据

注意：错误代码 bLenStatus 将在 GPIF II 的邮件箱中报告。

当使用同步 ADMUX 将固件下载到 FX3 时，外部 AP 需要进行以下操作：

- 命令块长度必须等于 512 个字节。
- 响应块长度必须等于 512 个字节。
- 需要将 Bootloader 二进制镜像转换成一个被分成多个 512 字节的数据流。
- Bootloader 镜像的数据块大小不能超过 8 K。例如，对于命令 0x02，bLenStatus 不能占用超过 16 个块（8 K 个字节）。
- 主机不应该发送大小超过镜像总空间的数据量。

Bootloader 不支持排队命令。因此，每次发送某个命令时，主机必须读取其响应。

在下载过程中，您应该防止破坏该 API 结构。

主机不应该将固件下载到 Bootloader 的保留系统地址（0x4000_0000 到 0x4000_23FF）内。如果固件应用尝试使用该地址空间，那么将返回错误结果。

不应该使用 DTCM 中前 1280 个字节（0x1000_0000 – 0x1000_04FF）。

对于 WRITE_DATA_CMD：当 bLenStatus = 0 时，Bootloader 将跳到 dAddr 的程序入口。

9.1.7 固件下载示例

本节介绍了通过 16 位同步 ADMux 接口，将固件从主机处理器下载到 FX3 的一个简单方法。

主机处理器与 FX3 Bootloader 进行通信，以执行固件下载。进行通信时，主机处理器需要对 FX3 寄存器和数据套接字进行读和写操作。

注意：有关 FX3 中套接字概念的详细信息，请参考 EZ-USB FX3 入门应用笔记中的“FX3 术语”部分。

主机处理器使用可用的 GPIF II 套接字将数据块传入/传出 FX3。FX3 Bootloader 使用三个数据套接字来处理固件下载协议，这三个数据套接字分别用于命令、响应和固件数据。

```
#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
```

主机处理器通过这些数据套接字与 FX3 Bootloader 进行通信，以进行固件下载。命令和响应是用于固件下载协议的数据结构。其大小都为 512 个字节。在这些数据结构中对各个位字段进行了定义，用以执行 FX3 Bootloader 的各个功能。

在本文档所提供的简单示例实现中，实际上仅使用了命令和响应的头 4 个字节。命令和响应中剩余的数据字节无需关注。

对于高级别的 FX3 固件，下载过程要求主机处理器执行以下套接字访问序列：

1. 一个命令套接字写操作，命令块被初始化为：

```
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. 另一个响应套接字读操作，所需的响应块数据为：

```
   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. 一个数据套接字写操作，它将结构为字节数组的整个固件镜像传输到 FX3 内。

请注意，固件镜像传输完成后，FX3 Bootloader 将自动跳转到刚下载固件的入口处，并开始执行。主机处理器与已经下载的固件进行通信前，建议等待一段时间（取决于固件的实现），以确保固件已被完全初始化。另一个更好的方法是，初始化完成后，通过邮件箱寄存器在固件中实现状态更新。在这种情况下，每当固件准备好时，主机处理器会得到通知。

9.1.8 处理器端口（P-Port）寄存器映射情况

表 28 中列出的寄存器表示 PP_xxx 寄存器在外部 P-Port 地址空间内的映射情况。该空间内的地址是一个字，并非一个字节。同步 ADMux 接口提供了八条地址线，用于访问这些寄存器。

表 28. 处理器端口寄存器映射情况

寄存器名称	地址	宽度（位）	说明
PP_ID	0x80	16	P-Port 器件 ID 寄存器。提供器件 ID 信息。
PP_INIT	0x81	16	P-Port 复位和电源控制。该寄存器用于复位和电源控制，并确定 P-Port 的字节顺序方向。
PP_CONFIG	0x82	16	P-Port 配置寄存器。
PP_IDENTIFY	0x83	16	P-Port 标识寄存器。该寄存器的低 8 位是只读的，默认情况下为 0x81。
PP_INTR_MASK	0x88	16	P-Port 中断屏蔽寄存器。该寄存器的布局与 PP_EVENT 相同，并且屏蔽导致中断激活的事件。
PP_DRQR5_MASK	0x89	16	P-Port DRQ/R5 屏蔽寄存器。该寄存器的布局与 PP_EVENT 相同，并分别屏蔽导致中断或 DRQ/R5 激活的事件。
PP_ERROR	0x8C	16	P-Port 错误指示寄存器。
PP_DMA_XFER	0x8E	16	P-Port DMA 传输寄存器。该寄存器用于设置和控制一个 DMA 传输。
PP_DMA_SIZE	0x8F	16	P-Port DMA 传输大小寄存器。该寄存器表示传输的（剩余）大小。
PP_WR_MAILBOX	0x90	64	P-Port 写邮件箱寄存器。这些寄存器包含了从应用处理器到 FX3 固件高达 8 字节的信息。

寄存器名称	地址	宽度（位）	说明
PP_MMIO_ADDR	0x94	32	P-Port MMIO 地址寄存器。这些寄存器一起构成了一个 32 位的地址，用于访问 FX3 内部 MMIO 空间。
PP_MMIO_DATA	0x96	32	P-Port MMIO 数据寄存器。这些寄存器一起形成了一个 32 位的数据，用于访问 FX3 内部 MMIO 空间。
PP_MMIO	0x98	16	P-Port MMIO 控制寄存器。该寄存器控制着对 FX3 MMIO 空间的访问。
PP_EVENT	0x99	16	P-Port 事件寄存器。该寄存器会显示导致中断或 DRQ 的所有事件类型。
PP_RD_MAILBOX	0x9A	64	P-Port 读邮件箱寄存器。这些寄存器包含从 FX3 固件到应用处理器的高达 8 字节信息。
PP_SOCK_STAT	0x9E	32	P-Port 套接字状态寄存器。这些寄存器的一位表示 P-port 上 32 个套接字中的每一个的缓冲区相应状态。

有关这些寄存器的位字段定义的信息，请参考 EZ-USB FX3 技术参考手册中的“寄存器”章节。

要想进一步了解 FX3 固件下载的详细信息，请注意本文档中示例实现通常使用的下面各函数，它们是与平台相关的。有关在特定平台上实现这些函数的更多信息，请联系赛普拉斯支持。

```
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
              //   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
```

注意：执行寄存器访问时，8 位地址中的最高有效位需为 1，以通知 FX3 这是寄存器访问操作。相同的，执行套接字访问时，最高有效位需为 0。

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

下面介绍了 fx3_firmware_download()函数的示例实现。该函数将指向固件数据数组的指针和固件大小作为参数。

```
/* 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)) {
                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);
                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;
}
```

下面内容介绍了套接字写和套接字读函数的示例实现。除了数据方向外，套接字写和套接字读函数的实现还取决于 PP_*寄存器接口的以下各命令、配置和状态位：

- PP_SOCK_STAT.SOCK_STAT[N]。该状态位表示某个套接字的缓冲区可以进行数据交换（缓冲区的数据或空间可用）。
- PP_DMA_XFER.DMA_READY。该状态位表示 GPIF II 是否准备好对套接字进行读/写操作（通过 PP_DMA_XFER 寄存器选择有效套接字）。经过几个周期后，PP_EVENT.DMA_READY_EV 才会映射 PP_DMA_XFER.DMA_READY 的值。
- PP_EVENT.DMA_WMARK_EV。该状态位与 DMA_READY 相同，但在当前缓冲区完成交换前，它将取消激活一个可编程的数据数量。在信号通信接口中，可以使用该位创建带有偏移延迟的流控制信号。
- PP_DMA_XFER.LONG_TRANSFER。该配置位表示长（多缓冲区）传输是否被使能。在传输启动过程中，应用处理器将设置该位。
- PP_CONFIG.BURSTSIZE 和 PP_CONFIG.DRQMODE。可使用这些配置位来定义并使能 DMA 突发的大小。每当成功更新 PP_CONFIG 寄存器时，FX3 Bootloader 将在 PP_RD_MAILBOX 寄存器中返回 0x33 值。
- PP_DMA_XFER.DMA_ENABLE。该命令和状态表示 DMA 传输已被使能。在传输启动过程中，主机处理器将设置该位。对于短传输，在传输完成后，FX3 硬件将清除该位；对于长传输，应用处理器将清除该位。

```
/* 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
   #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;
        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;
         /* 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;
}
```

9.2 启动镜像格式

对于同步 ADMux 启动，Booloader 要求固件镜像与表 29 中所显示的格式相同。EZ-USB FX3 SDK 提供了一个软件工具，用于生成一个具有同步 ADMux 启动所需要的格式的固件镜像。安装 SDK 后，请参考位于 *C:\Program Files\Cypress\EZ-USB FX3 SDK\1.3\util\elf2img* 目录中的 elf2img 工具。对于 64 位的操作系统，路径中的第一个文件夹为“Program Files(x86)”。目录路径中的 *1.3* 数值是指 SDK 的版本编号，该数值可根据 FX3 SDK 的最新发布版本而变。

请注意，elf2img 编译后的命令将生成一个.*img* 文件。需要将该文件转换成一个能够用于上述下载示例的数据组。图 11 显示了如何发送 elf2img 编译后的命令，该图下方是按照 ASCII 格式将.*img* 文件的内容复制到数组中的示例。

表 29. 同步 ADMux 启动选项的启动镜像格式

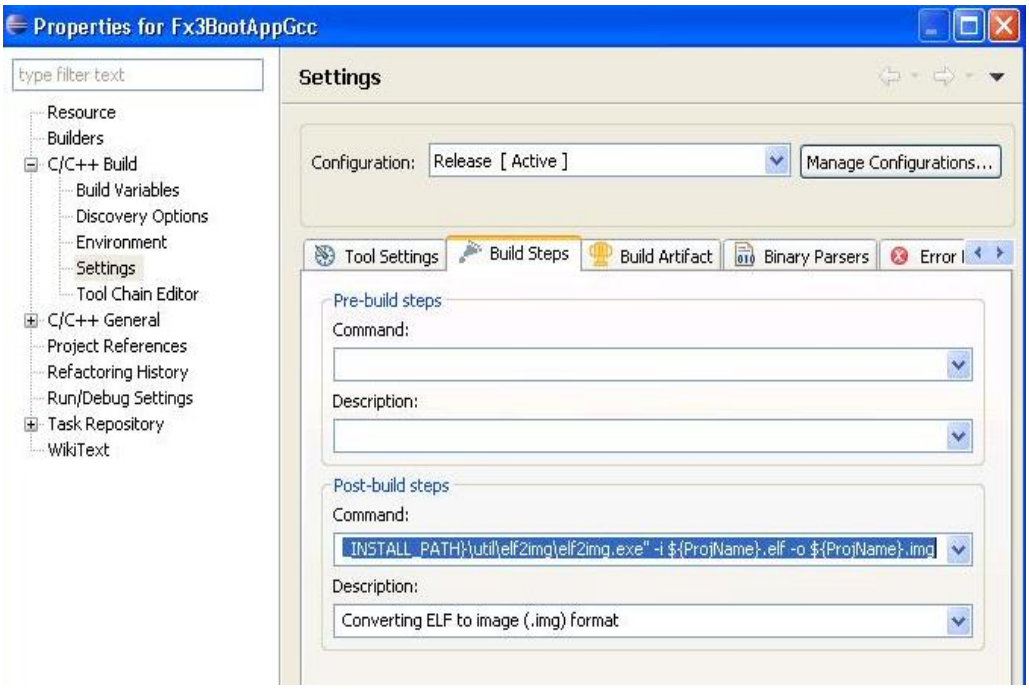
二进制镜像头文件	长度（16 位）	说明
wSignature	1	两个字节标签使用 ASCII 文本 “CY” 初始化 。
bImageCTL;	½	位 0 = 0：执行二进制文件；1：数据文件类型 位 3:1 在 SPI EEPROM 启动时不使用 位 5:4（SPI 速度）： 00：10 MHz 01：20 MHz 10：30 MHz 11：保留 位 7:6：保留，并要设置为零
bImageType;	½	bImageType = 0xB0：带校验和的普通 FW 二进制镜像 bImageType = 0xB2：使用新的 VID 和 PID 进行 SPI 启动
dLength 0	2	第一段长度，单位为长字（32 位） 当 bImageType = 0xB2 时，dLength 0 包含 PID 和 VID。Bootloader 忽略后面的所有数据。

二进制镜像头文件	长度（16 位）	说明
dAddress 0	2	编程代码的第一段地址。 **注意：**内部 ARM 地址是字节可寻址的，因此，每一段的地址应该为 32 位对齐。
dData[dLength 0]	dLength 0*2	镜像代码/数据必须为 32 位对齐。
…		更多的段
dLength N	2	0x00000000（最后记录：终端段）
dAddress N	2	应该包含有效的程序入口（通常为启动代码，如 RESET 向量）。 **注意：** 如果 bImageCTL.bit0 = 1，Bootloader 不会跳到该程序入口。 如果 bImageCTL.bit0 = 0，Bootloader 会跳到该程序入口。该地址应位于 ITCM 或 SYSTEM RAM 内。Bootloader 不会验证程序入口。
dCheckSum	2	低位优先的 32 位无符号校验和数据占用范围是从第一段到终端段。校验和不包含 dLength、dAddress 和镜像头文件。

下面是以长字格式组织的启动镜像示例：

```
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
Location 14: 0x6AF37AF2           //Checksum (0x12345678 + 0x9ABCDEF1 + 0x23456789 +
                                    0xABCDEF12+ 0xDDCCBBAA +0x11223344)
```

图 11. Eclipse IDE 中的编译后命令



下面显示了按照 ASCII 格式将*.img* 文件的内容复制到数组内的示例代码：

```
#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");
```

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

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

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

10 eMMC 启动

FX3S 外设控制器支持从 eMMC 器件启动。将 eMMC 器件连接到 FX3S 的 S0 存储端口，固件启动可从该端口执行。FX3S 还支持带有 USB 回退功能的 eMMC 启动。如果在 eMMC 中不存在有效的固件，则 FX3S 会返回到 USB 启动模式。有关使能 eMMC 启动所需的 PMODE 引脚设置，请参考表 30。

表 30. 实现 eMMC 启动的 PMODE 设置

PMODE[2]	PMODE[1]	PMODE[0]	启动选项
1	0	0	eMMC
0	0	0	eMMC -> USB

下载 FX3 SDK 后，请参考 *cyfwstorprog_usage.txt*，了解实现 eMMC 启动的详细指导。该文件位于：

```
<FX3 SDK installation path>\Cypress\EZ-USB FX3 SDK\1.x\util\cyfwstorprog\
```

注意：eMMC 启动仅由 FX3S 外设支持。

11 启动时 I/O 的默认状态

表 31 显示了 Bootloader 运行（应用固件下载前）时，FX3 IO 在不同启动模式下的默认状态。

注意：在 FX3 复位时和在 Bootloader 完成配置后， GPIO 的默认状态无需同样。

表 31. 启动时 I/O 的默认状态

GPIO	SPI 启动的默认状态	USB 启动的默认状态	I²C 启动的默认状态	同步 ADMux 启动的默认状态
GPIO[0]	三态	三态	三态	三态
GPIO[1]	三态	三态	三态	三态
GPIO[2]	三态	三态	三态	三态
GPIO[3]	三态	三态	三态	三态
GPIO[4]	三态	三态	三态	三态
GPIO[5]	三态	三态	三态	三态
GPIO[6]	三态	三态	三态	三态
GPIO[7]	三态	三态	三态	三态
GPIO[8]	三态	三态	三态	三态

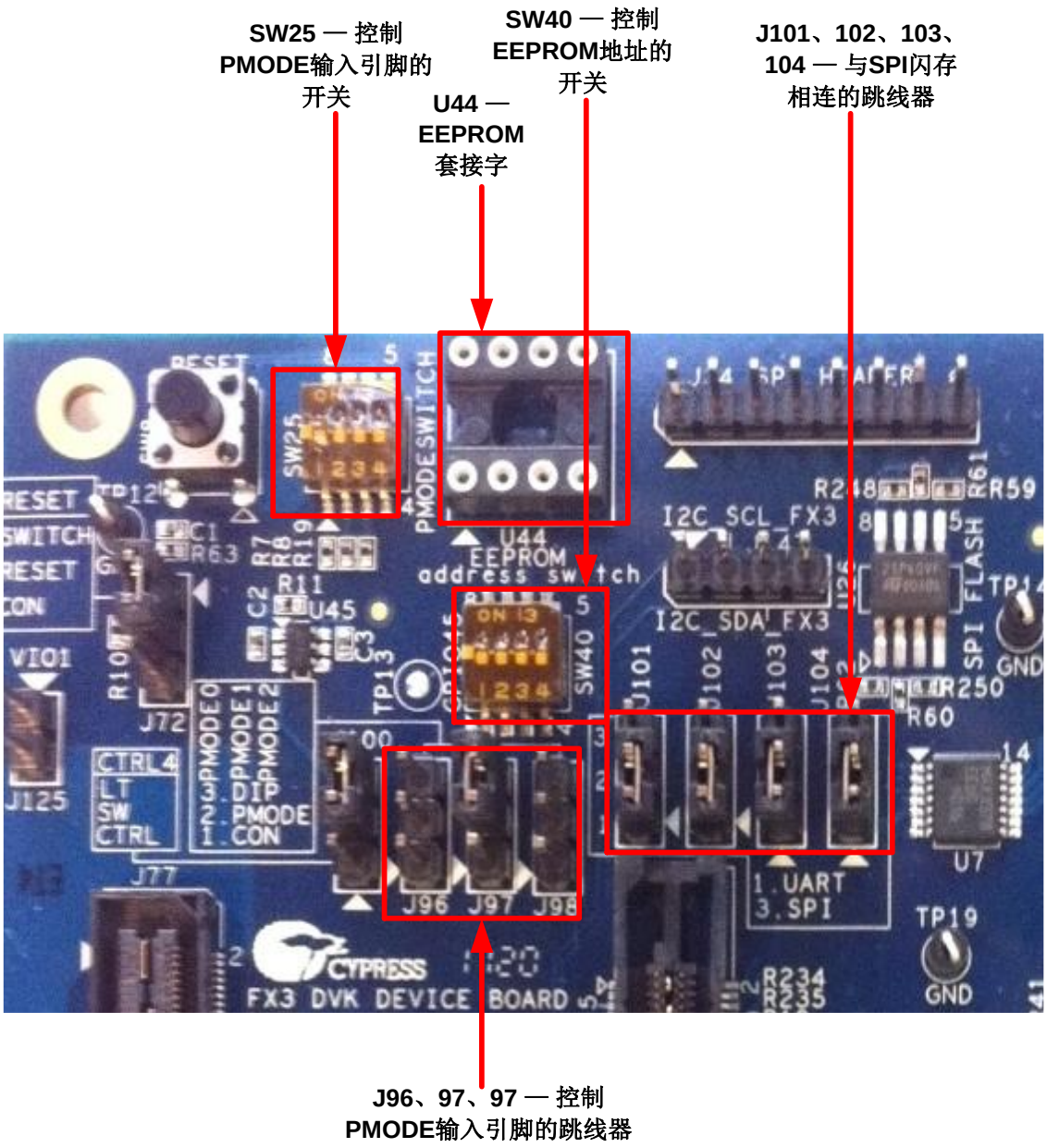
GPIO	SPI 启动的默认状态	USB 启动的默认状态	I²C 启动的默认状态	同步 ADMux 启动的默认状态
GPIO[9]	三态	三态	三态	三态
GPIO[10]	三态	三态	三态	三态
GPIO[11]	三态	三态	三态	三态
GPIO[12]	三态	三态	三态	三态
GPIO[13]	三态	三态	三态	三态
GPIO[14]	三态	三态	三态	三态
GPIO[15]	三态	三态	三态	三态
GPIO[16]	三态	三态	三态	CLK 输入
GPIO[17]	三态	三态	三态	输入
GPIO[18]	三态	三态	三态	输入
GPIO[19]	三态	三态	三态	输入
GPIO[20]	三态	三态	三态	输入
GPIO[21]	三态	三态	三态	输出
GPIO[22]	三态	三态	三态	三态
GPIO[23]	三态	三态	三态	输入
GPIO[24]	三态	三态	三态	三态
GPIO[25]	三态	三态	三态	三态
GPIO[26]	三态	三态	三态	三态
GPIO[27]	三态	三态	三态	输入
GPIO[28]	三态	三态	三态	三态
GPIO[29]	三态	三态	三态	三态
GPIO[30]	PMODE[0] I/P 至 FX3	PMODE[0] I/P 至 FX3	PMODE[0] I/P 至 FX3	PMODE[0] I/P 至 FX3
GPIO[31]	PMODE[1] I/P 至 FX3	PMODE[1] I/P 至 FX3	PMODE[1] I/P 至 FX3	PMODE[1] I/P 至 FX3
GPIO[32]	PMODE[2] I/P 至 FX3	PMODE[2] I/P 至 FX3	PMODE[2] I/P 至 FX3	PMODE[2] I/P 至 FX3
GPIO[33]	三态	三态	三态	三态
GPIO[34]	三态	三态	三态	三态
GPIO[35]	三态	三态	三态	三态
GPIO[36]	三态	三态	三态	三态
GPIO[37]	三态	三态	三态	三态
GPIO[38]	三态	三态	三态	三态
GPIO[39]	三态	三态	三态	三态
GPIO[40]	三态	三态	三态	三态
GPIO[41]	三态	三态	三态	三态
GPIO[42]	低电平	低电平	低电平	低电平
GPIO[43]	三态	三态	三态	三态

GPIO	SPI 启动的默认状态	USB 启动的默认状态	I²C 启动的默认状态	同步 ADMux 启动的默认状态
GPIO[44]	三态	三态	三态	三态
GPIO[45]	三态（如果 SPI 启动失败，则为高电平）	高电平	高电平	高电平
GPIO[46]	高电平	三态	三态	三态
GPIO[47]	三态	三态	三态	三态
GPIO[48]	高电平	三态	三态	三态
GPIO[49]	三态	三态	三态	三态
GPIO[50]	三态（如果 SPI 启动失败，则为低电平）	三态	三态	三态
GPIO[51]	低电平	低电平	低电平	低电平
GPIO[52]	高电平	三态	三态	三态
GPIO[53]	低电平（在 SPI 数据传输过程中进行切换）	高电平	高电平	高电平
GPIO[54]	高电平	三态	三态	三态
GPIO[55]	三态	高电平	高电平	高电平
GPIO[56]	低电平	三态	三态	三态
GPIO[57]	低电平	三态	三态	三态
GPIO[58] I2C_SCL	三态	三态	三态（在数据传输过程中进行切换，然后转换成三态）	三态
GPIO[59] I2C_SDA	三态	三态	三态	三态

A 附录：使用 FX3 DVK 电路板进行启动的步骤

本附录介绍了使用 FX3 DVK 电路板进行 USB 启动、I²C 启动和 SPI 启动的各步骤。图 12 显示了 FX3 DVK 电路板的一部分，其中包括开关和跳线器。需要根据每个启动选项正确配置这些开关和跳线器。另外，还介绍了开关和跳线器所需的设置。

图 12. FX3 DVK 电路板：用于启动的关键开关和跳线器配置



USB 启动

1. 在 Eclipse IDE 中编译固件镜像，如图 13、图 14 和图 15。

图 13. 右击 Eclipse IDE 中的项目

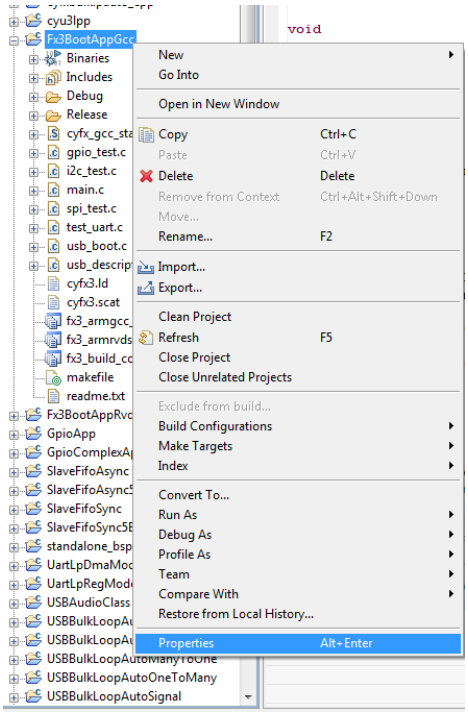


图 14.选择设置

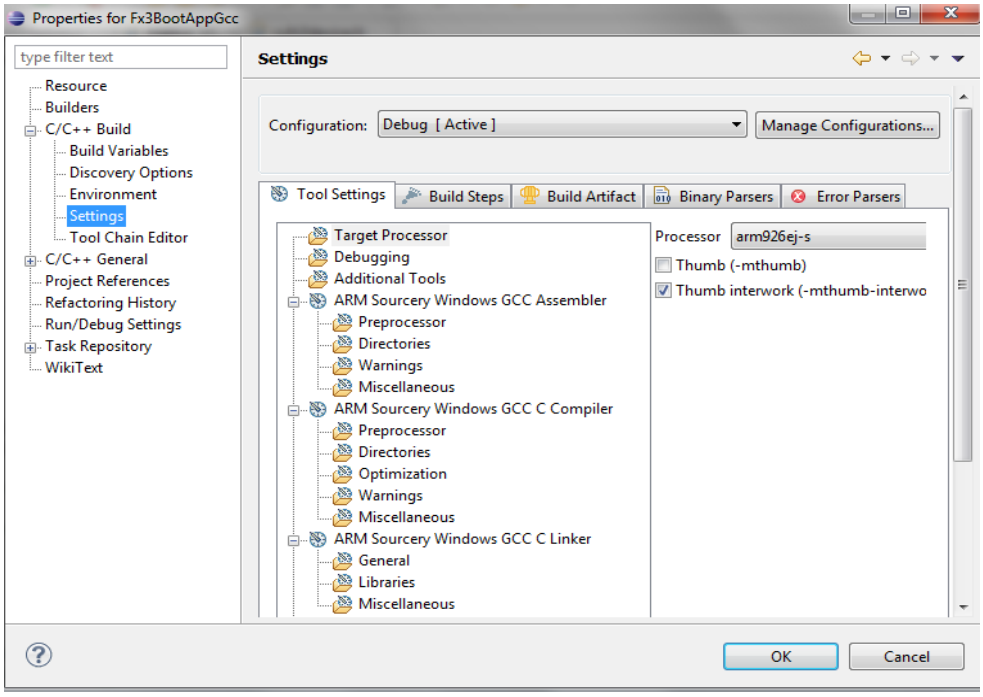
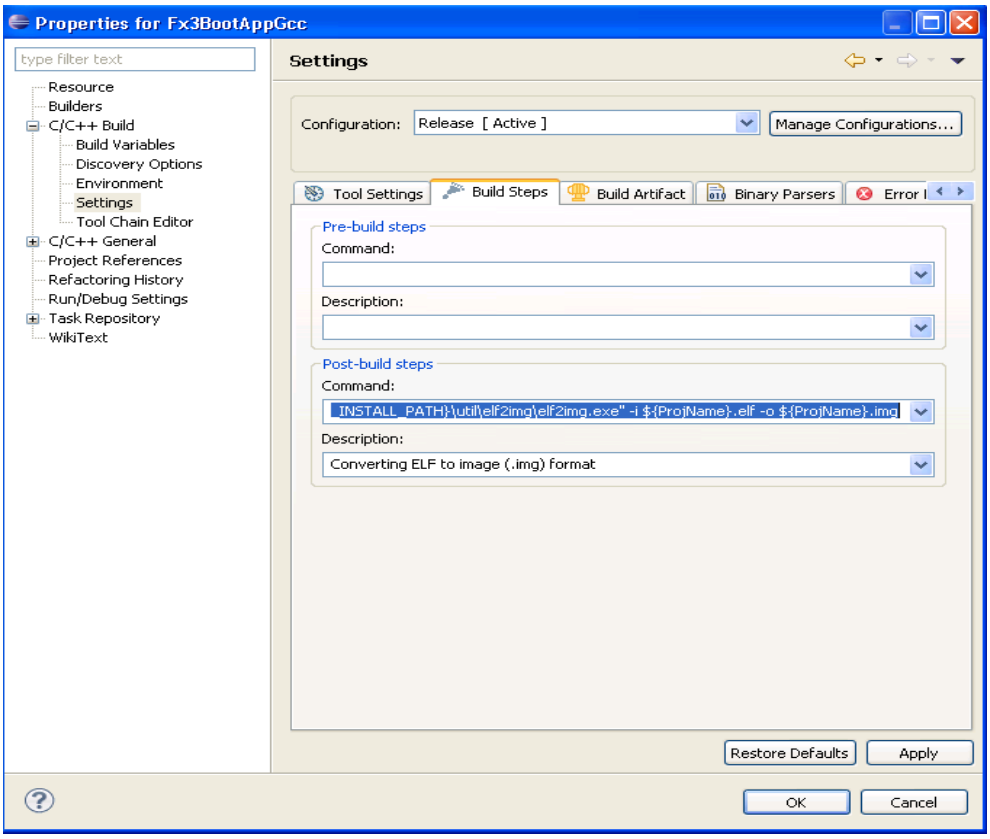


图 15. “Post-Build Steps”中用于 USB 启动镜像的 elf2img 命令配置



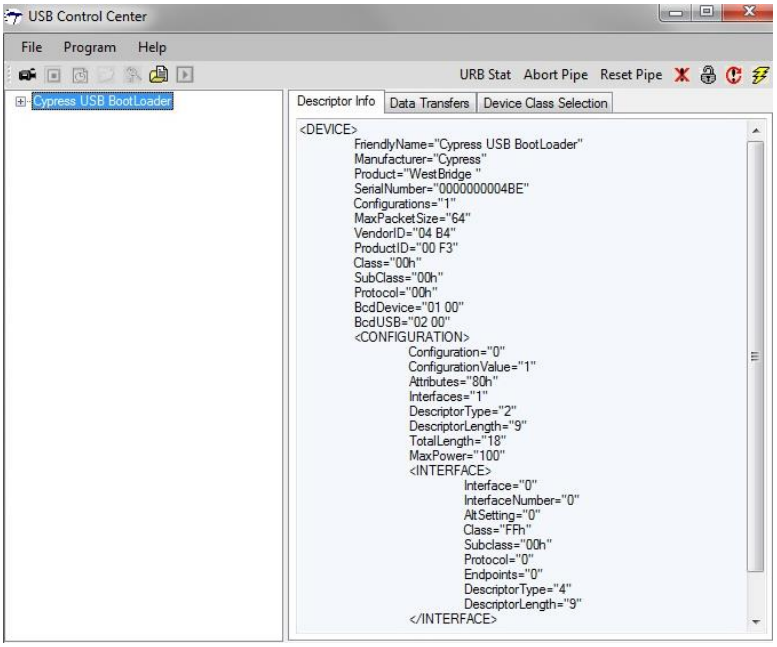
2. 将 PMODE[2:0]引脚设置为 Z11 来使能 USB 启动。在 DVK 电路板上，通过配置跳线器和开关，可以实现该操作，如表 32 所示。

表 32. USB 启动的跳线器配置

跳线器/开关	位置	相应 PMODE 引脚的状态
J96（PMODE0）	2-3 短接	PMODE0 由 SW25 控制
J97（PMODE1）	2-3 短接	PMODE1 由 SW25 控制
J98（PMODE2）	开路	PMODE2 悬空
SW25.1-8（PMODE0）	开路（OFF 位置）	PMODE0 = 1
SW25.2-7（PMODE1）	开路（OFF 位置）	PMODE1 = 1
SW25.3-6（PMODE2）	无需关注	PMODE2 悬空

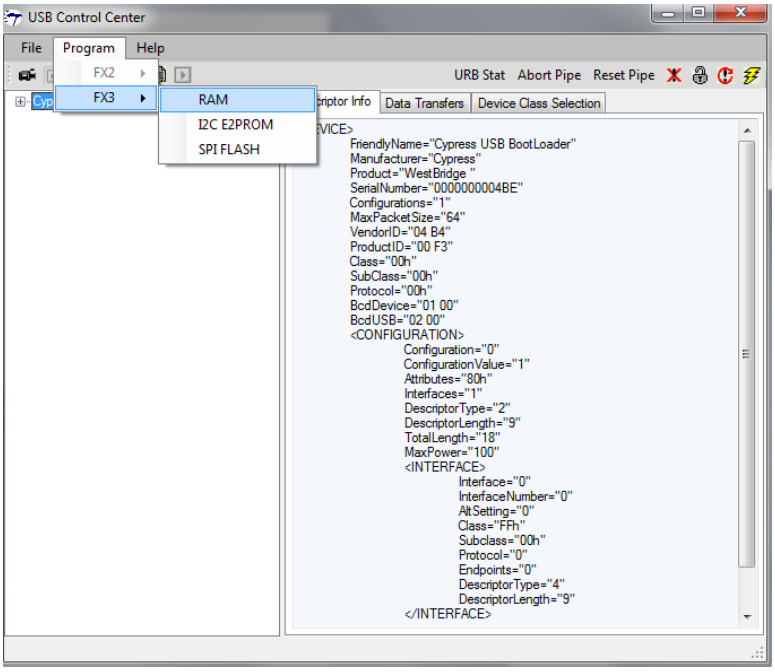
3. 当连接到 USB 主机时，FX3 器件在 USB Control Center 中作为“Cypress USB Bootloader”进行枚举，如图 16 中所示。

图 16.赛普拉斯 USB BootLoader 在 Control Center 中的枚举



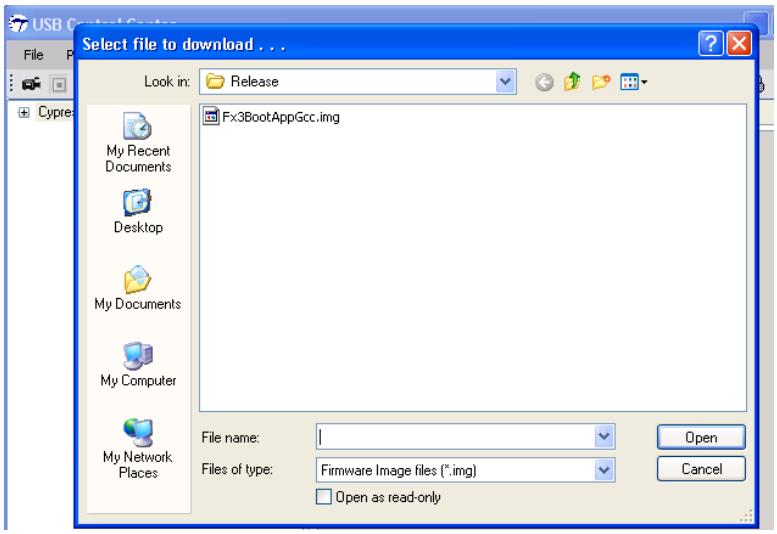
4. 在 Control Center 中，依次选择 **Program > FX3 > RAM** 来选中 FX3 器件，如图 17 所示。

图 17.从 Control Center 中选择器件



5. 然后，导航到需要编程到 FX3 RAM 的相应*.img* 文件。双击*.img* 文件，如图 18 中所示。

图 18.选择.img 文件



6. “Programming Succeeded”信息会显示在 Control Center 的左下方，并且 FX3 器件将使用已编程的固件重新进行枚举。

I²C 启动

1. 在 Eclipse IDE 中编译固件镜像，如图 19、图 20 和图 21 所示。

图 19.右击 Eclipse IDE 中的项目

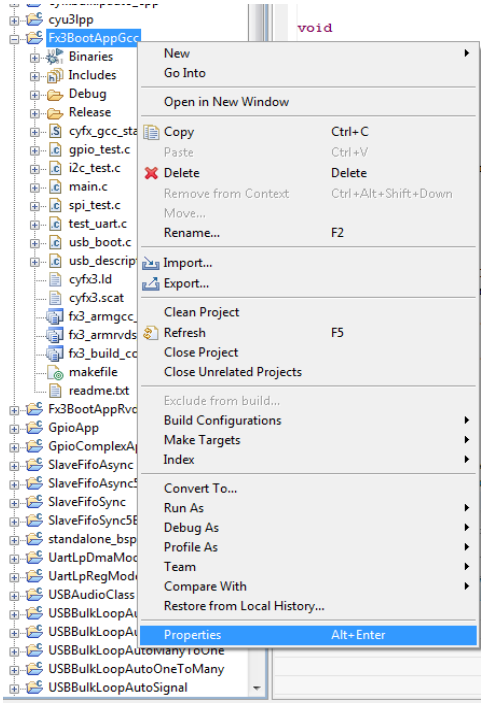


图 20. Fx3BootAppGcc 的属性

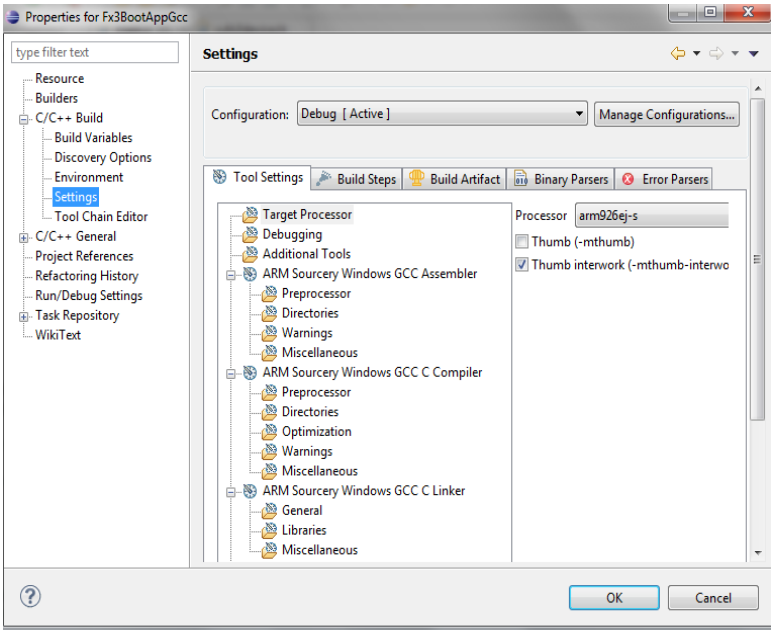
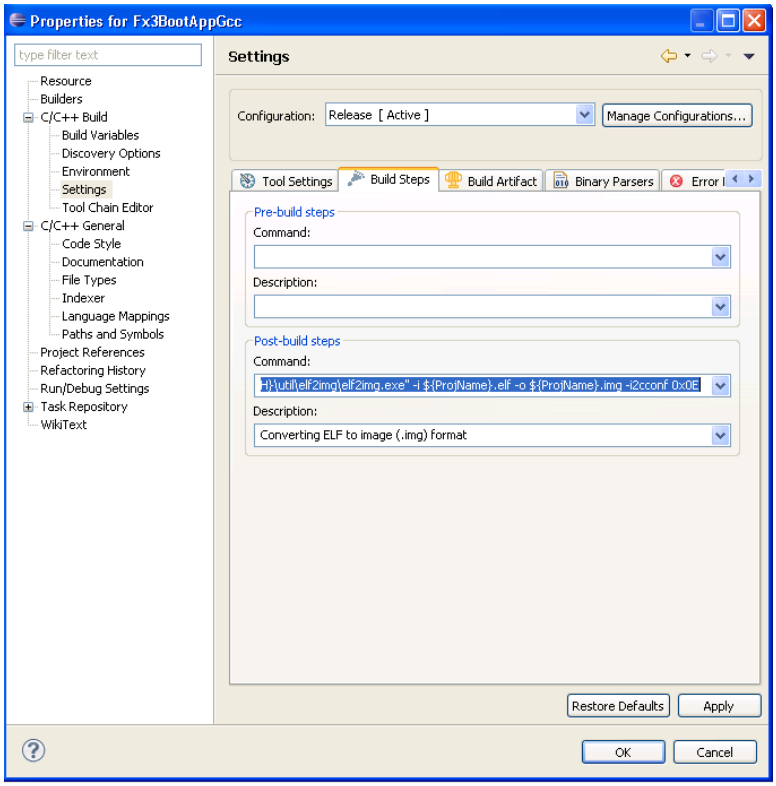


图 21. “Post-Build Steps”中用于 I²C 启动镜像的 elf2img 命令配置



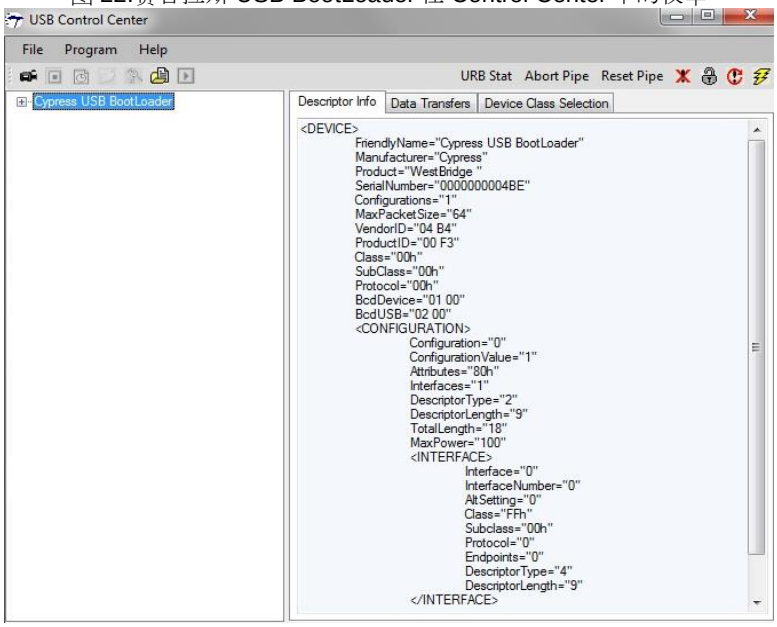
2. 将 PMODE[2:0]引脚设置为 Z11 来使能 USB 启动。在 DVK 电路板上，通过配置跳线器和开关，可以实现该操作，如表 33 所示。

表 33 USB 启动的跳线器配置

跳线器/开关	位置	相应 PMODE 引脚的状态
J96（PMODE0）	2-3 短接	PMODE0 由 SW25 控制
J97（PMODE1）	2-3 短接	PMODE1 由 SW25 控制
J98（PMODE2）	开路	PMODE2 悬空
SW25.1-8（PMODE0）	开路（OFF）	PMODE0 = 1
SW25.2-7（PMODE1）	开路（OFF）	PMODE1 = 1
SW25.3-6（PMODE2）	无需关注	PMODE2 悬空

3. 当连接至 USB 主机时，FX3 器件在 USB Control Center 中作为“Cypress USB Bootloader”进行枚举，如图 22. 中所示。

图 22.赛普拉斯 USB BootLoader 在 Control Center 中的枚举



4. 尝试编程 EEPROM 前，请确保已经使用开关 SW40 正确配置 EEPROM 的地址信号（对于 Microchip 器件 24AA1025，1-8 打开、2-7 打开、3-6 关闭）。此外，在 DVK 电路板上，需要短接 I²C 时钟（SCL）和数据线（SDA）跳线器 J42 和 J45 引脚 1-2。在 Control Center 窗口中，选择 FX3 器件。然后，依次选择 Program > FX3 > I2C E2PROM，如图 23 所示。这样，特殊的 I²C 启动固件将被编程到 FX3 器件内，从而能够对连接至 FX3 的 I²C 器件进行编程。FX3 将作为“Cypress USB BootProgrammer”重新进行枚举，如图 24 所示。

图 23.依次选择 Program > FX3 > I2C E2PROM

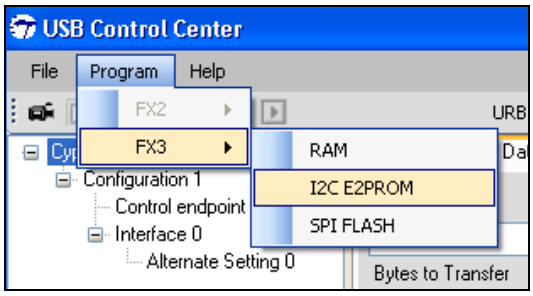
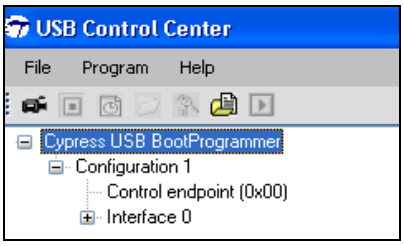
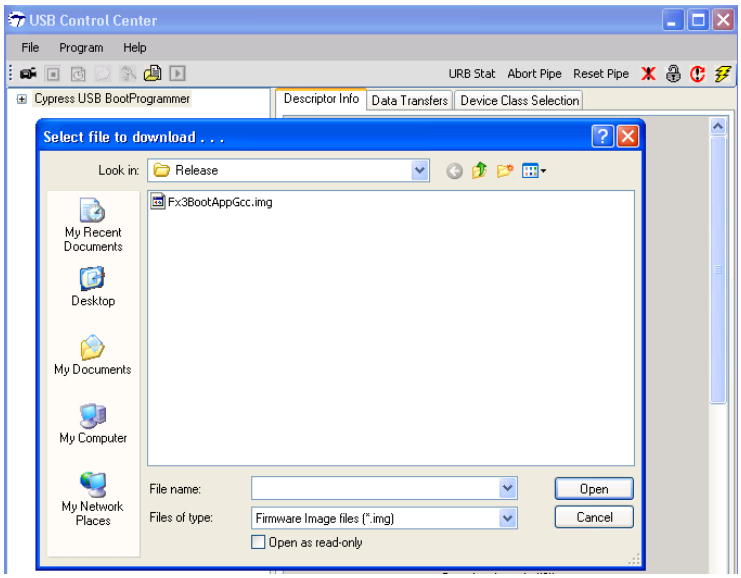


图 24. FX3 作为“Cypress USB BootProgrammer”重新进行枚举



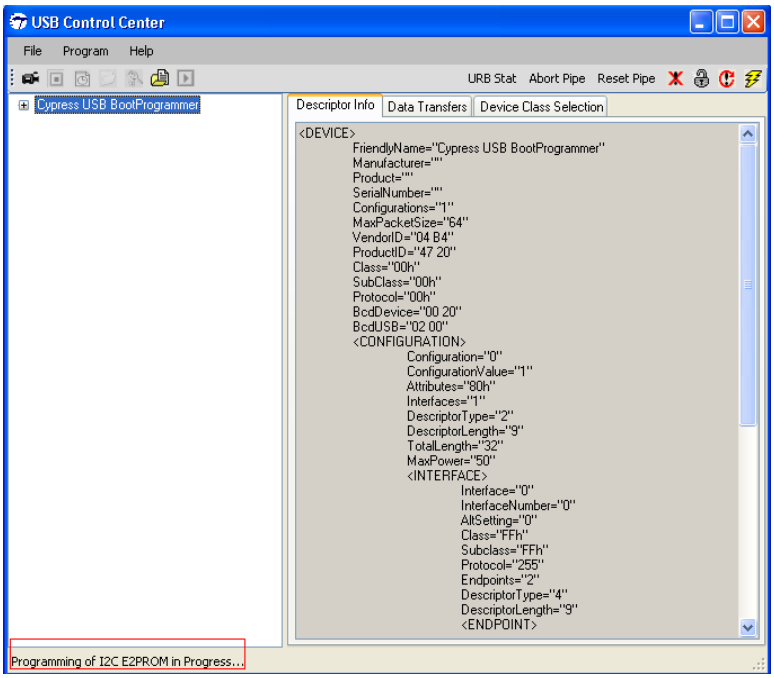
5. FX3 DVK 电路板作为‘Cypress USB BootProgrammer’重新进行枚举后，Control Center 应用将提示用户选择需要下载的固件二进制文件。导航到被编程到 I²C EEPROM 内的*.img* 文件，如 图 25 所示。

图 25.选择需要下载的固件镜像



编程完成后，该窗口的左下角将显示“Programming of I2C E2PROM Succeeded”信息，如图 26 所示。

图 26.I²C EEPROM 在 Control Center 中的编程更新



6. 将 DVK 电路板上的 PMODE 引脚改为 Z1Z，以使能 I²C 启动。在该电路板上，通过配置跳线器和开关，可以实现该操作，如表 34 所示。

表 34.I²C 启动的跳线器配置

跳线器/开关	位置	相应 PMODE 引脚的状态
J96（PMODE0）	开路	PMODE0 悬空
J97（PMODE1）	2-3 短接	PMODE1 由 SW25 控制
J98（PMODE2）	开路	PMODE2 悬空
SW25.1-8（PMODE0）	无需关注	PMODE0 悬空
SW25.2-7（PMODE1）	开路（OFF 位置）	PMODE1 = 1
SW25.3-6（PMODE2）	无需关注	PMODE2 悬空

7. 复位 DVK。现在，FX3 器件从 I²C EEPROM 启动。

SPI 启动

1. 在 Eclipse IDE 中编译固件镜像，如图 27、图 28 和图 29 所示。

图 27.右击 Eclipse IDE 中的项目

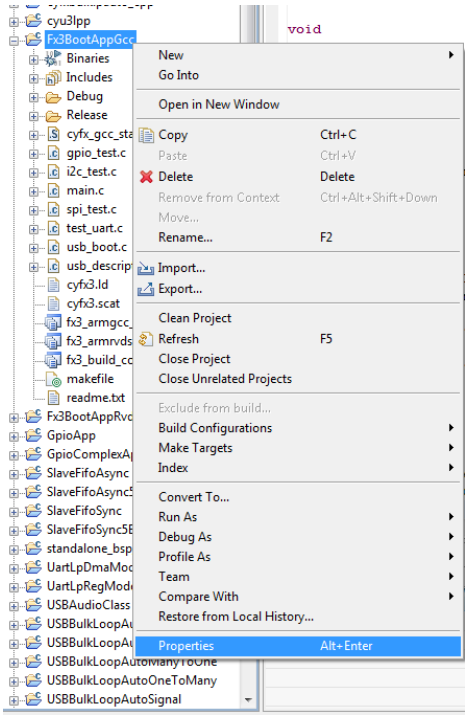


图 28. 选择“Settings”

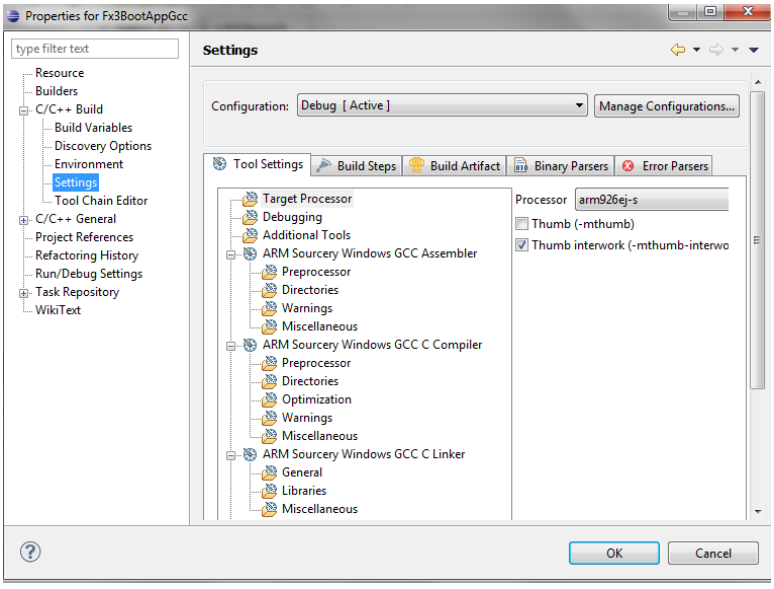
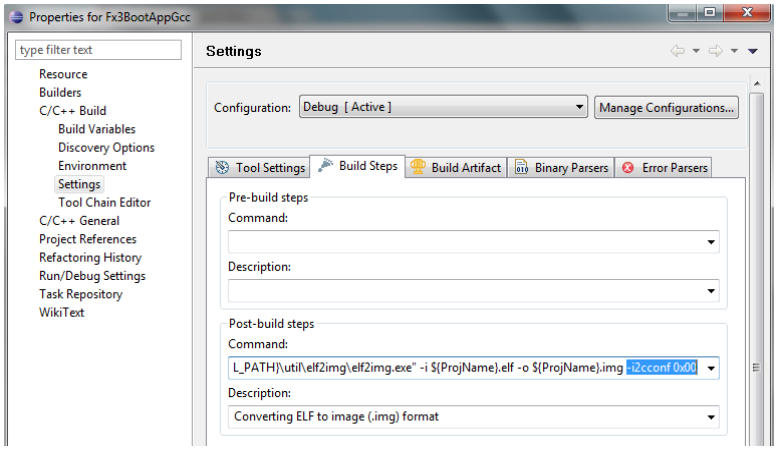


图 29.“Post-build steps”中用于 SPI 启动镜像的 elf2img 命令配置



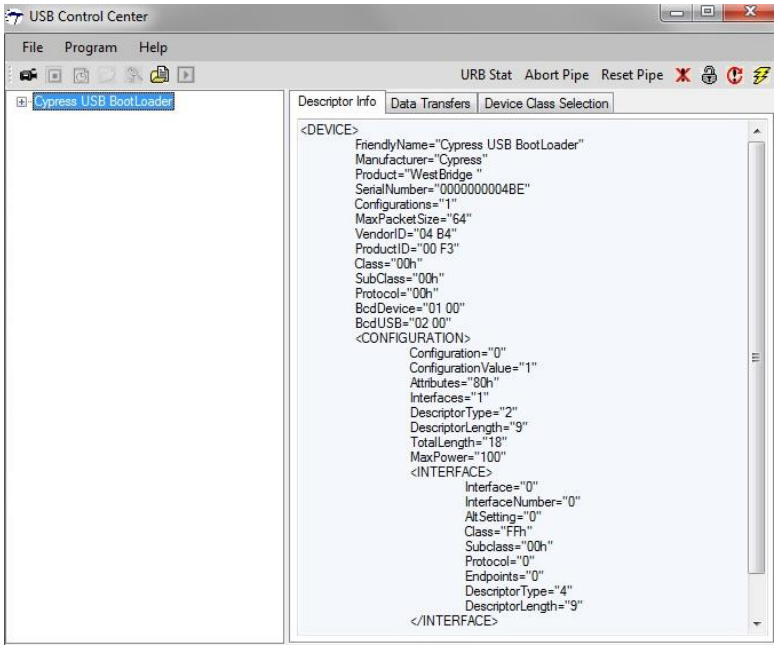
2. 将 PMODE[2:0]引脚设置为 Z11 来使能 USB 启动。在 DVK 电路板上，通过配置跳线器和开关，可以实现该操作，如表 35 所示。

表 35. USB 启动的跳线器配置

跳线器/开关	位置	相应 PMODE 引脚的状态
J96（PMODE0）	2-3 短接	PMODE0 由 SW25 控制
J97（PMODE1）	2-3 短接	PMODE1 由 SW25 控制
J98（PMODE2）	开路	PMODE2 悬空
SW25.1-8（PMODE0）	开路（OFF 位置）	PMODE0 = 1
SW25.2-7（PMODE1）	开路（OFF 位置）	PMODE1 = 1
SW25.3-6（PMODE2）	无需关注	PMODE2 悬空

3. 当连接至 USB 主机时，FX3 器件在 USB Control Center 中作为“Cypress USB Bootloader”进行枚举，如图 30 中所示。

图 30. 赛普拉斯 USB BootLoader 在 Control Center 中的枚举



4. 在 Control Center 窗口中，选择 FX3 器件，然后依次选择 Program > FX3 > SPI FLASH，如图 31 所示。浏览到被编程到 SPI 闪存内的.*img* 文件，如图 32 所示。

图 31.在 Control Center 中依次选择 Program > FX3 > SPI FLASH

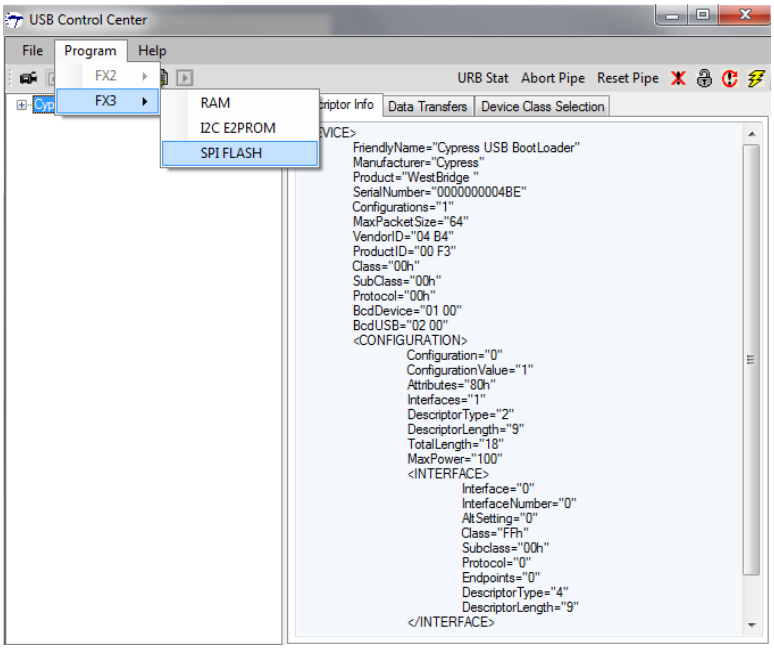
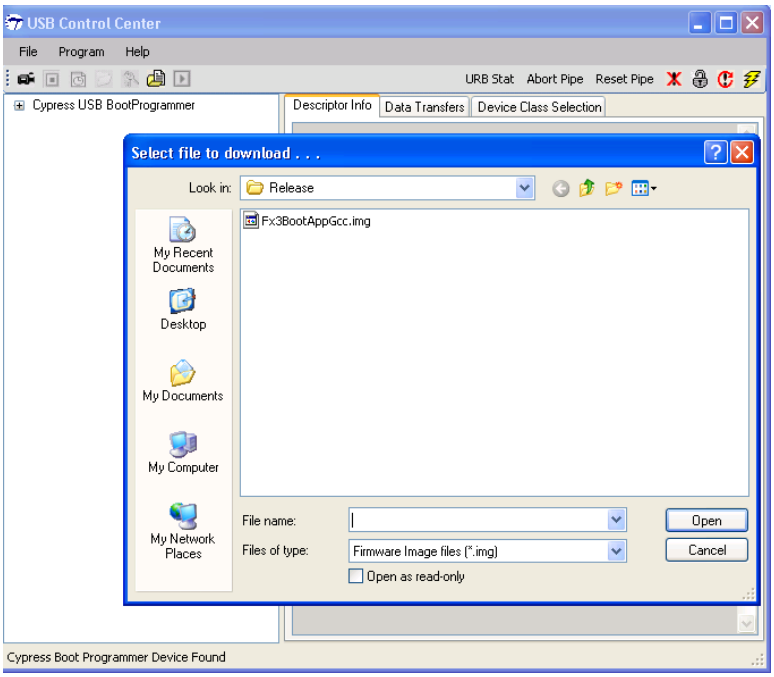
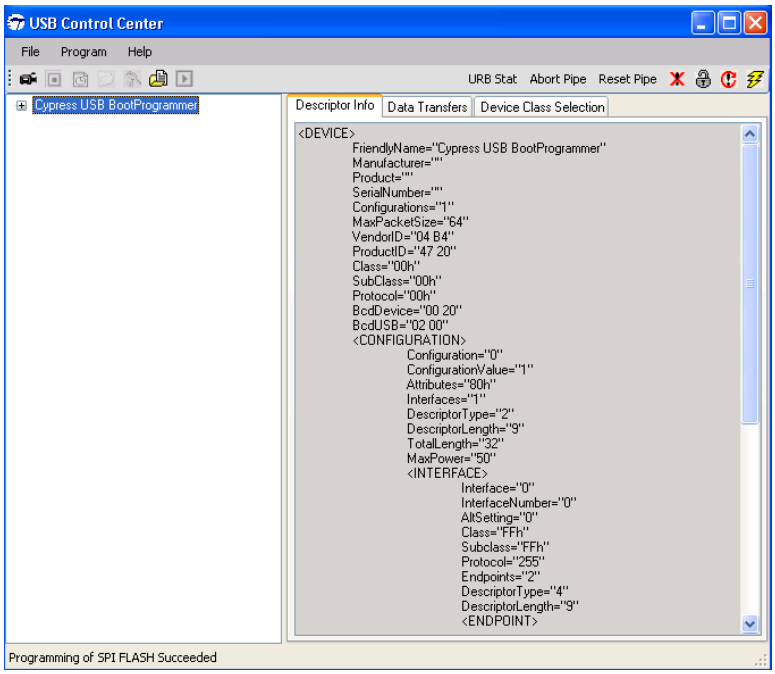


图 32. 双击需要编程到 SPI 闪存内的.img 文件



5. 编程完成后，该窗口的左下角将显示“Programming of SPI FLASH Succeeded”信息，如图 33.所示。

图 33. SPI 闪存成功编程在 Control Center 窗口的左下角的显示



6. 将 DVK 电路板上的 PMODE[2:0]引脚改为 0Z1，以使能 SPI 启动。在该电路板上，通过配置跳线器和开关，可以实现该操作，如表 36 所示。

表 36.SPI 启动的跳线器配置

跳线器/开关	位置	相应 PMODE 引脚的状态
J96（PMODE0）	2-3 已被关闭	PMODE0 由 SW25 控制
J97（PMODE1）	打开	PMODE1 悬空
J98（PMODE2）	2-3 已被关闭	PMODE2 由 SW25 控制
SW25.1-8（PMODE0）	打开（OFF 位置）	PMODE0 = 1
SW25.2-7（PMODE1）	无需关注	PMODE1 悬空
SW25.3-6（PMODE2）	已被关闭（ON 位置）	PMODE2 = 0

7. 复位 DVK。现在，FX3 将从 SPI 闪存启动。

B 附录 B：同步 ADMux 引导的故障排除步骤

本附录描述了有关如何测试和调试同步 ADmux 引导的各种序列的逐步说明。

B.1 初始化

1. 在主机侧配置存储器接口以满足 FX3 的时序要求。
2. 如果需要（在 Linux 的情况下），将 FX3 接口对应的地址区域内存映射到虚拟内存区域。
3. 将 0x0000 写入地址的 PP_INIT 寄存器（FX3 地址= 0x81）。
4. 将 0x0040 写入地址的 PP_CONFIG 寄存器（FX3 地址= 0x82）。

B.2 测试寄存器读/写

1. 使用各种值写入 PP_SOCK_MASK_H（0x8B）和 PP_SOCK_MASK_L（0x8A）寄存器并将其读回以进行测试和验证。
2. 使用各种值写入 PP_INTR_MASK（0x88）寄存器并将其读回以进行验证。 请注意，该寄存器具有多个保留位：值读取=写入的值和 0xF8FF。
3. 将以下值写入这些寄存器以设置 FIFO 访问测试：

   PP_SOCK_MASK_H = 0x0000

   PP_SOCK_MASK_L = 0x0007

   PP_INTR_MASK = 0x2001

B.3 测试 FIFO 读/写

FX3 引导加载程序提供的内存写入和读取调试命令用于测试 FIFO 访问，验证接口是否正常工作，以及是否可用于固件下载。

1. 将数据模式写入存储器地址 0x40003000：
   a. 等到 PP_SOCK_STAT_L 寄存器（0x9E）的位 0（Socket 0 Available）置位。
   b. 将 0x0300 写入 PP_DMA_XFER 寄存器（0x8E）
   c. 等到 PP_DMA_XFER 寄存器的位 12 和位 15 置 1。
   d. 使用以下格式将 256 字节数据写入 FX3 器件地址 0（SOCKET 0）：

      Byte 0 = 0x43
      Byte 1 = 0x59
      Byte 2 = 0x01 (写命令)
      Byte 3 = 0x7E
      Byte 4 = 0x00 (LSB 的地址)
      Byte 5 = 0x30
      Byte 6 = 0x00
      Byte 7 = 0x40 (MSB 的地址)
      Bytes 8 to 255 可以包含随机数据

2. 回读写操作的状态：
   a. 等到 PP_SOCK_STAT_L 寄存器（0x9E）的第 2 位（Socket 2 Available）置位。
   b. 将 0x0102 写入 PP_DMA_XFER 寄存器（0x8E）。
   c. 等到 PP_DMA_XFER 寄存器的第 12 位置 1。
   d. 读取 PP_DMA_SIZE 寄存器（0x8F）并验证该值是否为 0x0100。
   e. 从 FX3 地址 0x02 读取 256 字节的数据（128 个周期）。
   f. 验证前四个字节是否包含模式 0x53,0x42,0x01 和 0x00。
3. 启动 FIFO 读取命令以从地址 0x40003000 读取数据：
   a. 等到 PP_SOCK_STAT_L 寄存器（0x9E）的第 0 位置位。
   b. 将 0x0300 写入 PP_DMA_XFER 寄存器（0x8E）。
   c. 等到 PP_DMA_XFER 寄存器的位 12 和位 15 置 1。
   d. 使用以下格式将 256 字节数据写入 FX3 器件地址 0（SOCKET 0）：

      Byte 0 = 0x43

      Byte 1 = 0x59

      Byte 2 = 0x03 (读命令)

      Byte 3 = 0x7E

      Byte 4 = 0x00 (LSB的地址)

      Byte 5 = 0x30

      Byte 6 = 0x00

      Byte 7 = 0x40 (MSB的地址)

      Bytes 8 to 255 可忽略

4. 从插槽 2 读回内存数据：
   a. 等到 PP_SOCK_STAT_L 寄存器（0x9E）的第 2 位置位。
   b. 将 0x0102 写入 PP_DMA_XFER 寄存器（0x8E）。
   c. 等到 PP_DMA_XFER 寄存器的第 12 位置 1。
   d. 读取 PP_DMA_SIZE 寄存器（0x8F）并验证该值是否为 0x0100。

e. 从 FX3 地址 0x02 读取 256 字节数据（128 个周期）。

f. 验证前 4 个字节是否包含模式 0x53,0x42,0x03,0x00。

g. 验证字节 8 到 255 是否与上面步骤 1 中写入的随机数据匹配。

5. 对其他内存地址和数据模式重复步骤 1 到 4。

B.4 测试固件下载

如果上述所有检查均正常，则可以继续进行固件下载测试。 以下序列用于固件下载：

1. 使用目标固件将 img 文件的内容读入内存缓冲区。 根据需要将数据填充为 256 字节的倍数。

2. 按照 B.3 节中的步骤 1 将以下数据写入套接字 0：0x43,0x59,0x02,0x01，...（其余 252 字节无关紧要）。

3. 按照 B.3 节中的步骤 2 从插槽 2 读取固件下载命令状态。验证字节 3（状态）的值是否为 0x00。

4. 现在，将完整的固件内容写入套接字 1，一次写入 256 个字节。 按照下面给出的步骤将每个 256 字节写入套接字 1。

   a. 等到 PP_SOCK_STAT_L 寄存器（0x9E）的第 1 位置 1。

   b. 将 0x0301 写入 PP_DMA_XFER 寄存器（0x8E）。

   c. 等到 PP_DMA_XFER 寄存器的位 12 和位 15 置 1。

   d. 将 256 字节的数据写入 FX3 器件地址 1。

C 附录 C：使用 elf2img 程序生成固件映像 Image

本附录介绍如何使用 elf2img 程序（在 SDK 安装路径的 util \ elf2img 文件夹中）生成本应用笔记中提到的引导选项的固件映像。

C.1 用法

该实用程序是一个控制台应用程序，需要使用以下选项调用：

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

<elf filename>: 输入带路径的 ELF 文件名

<image filename>: 输出带路径的文件名

<eeprom control>: I2C / SPI EEPROM 控制字以十六进制形式

<image type>: 十六进制形式的镜像类型字节

-v: 在转换过程中启用详细日志

-h: 打印帮助信息

C.1.1 镜像类型

所有固件应用程序的<image type>应为 0xB0。 其他值保留。

C.1.2 中断向量加载

FX3 器件上的 ARM926EJ-S 内核的复位和中断向量存储在存储器的前 256 个字节中（地址范围为 0x00-0x100）。 不建议将任何代码直接加载到此地址范围内，因为它可能会干扰引导加载程序或活动固件操作。 FX3 固件库和默认链接器设置确保没有有效代码直接加载到此地址范围。 固件开始运行后，中断向量会安全地复制到此区域。

默认模式下的 elf2img 程序会在生成引导映像时删除 0x00-0x100 地址范围内的所有数据。 这是安全的，因为推荐的链接器设置确保在此地址范围内没有放置有效的代码/数据。 可以使用-vectorload 命令行选项覆盖此行为。

<vecload>值是 yes / no 字符串，当设置为“yes”时，工具会在引导映像中保留此地址范围内的任何数据。 此参数的默认值为“no”。

C.1.3 EEPROM 控制

该参数仅适用于从 I2C EEPROM 或 SPI FLASH 引导的情况。 如果 FX3 通过 USB 或 GPIF 端口启动，则不使用该字段，生成 img 文件时可省略该字段

在 I2C 引导的情况下，<eeprom control>字节指定所使用的 EEPROM 的类型和速度。

在 SPI 引导的情况下，<eeprom control>字节指定应该从 EEPROM 引导 SPI 的速度。 当使用任何其他引导模式时，此字节无关紧要。

C.1.3.1 I2C 参数

I2C 启动时的编码如下：

Bit 0	必须为零
Bits 3 - 1	EEPROM 大小[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	必须为零

例如，值 0x1C 对应于在 400kHz 频率下使用 64KB EEPROM。

C.1.3.2 SPI 参数

SPI 启动时的编码如下：

Bit 0	必须为零
Bits 3 - 1	可忽略
Bits 5 - 4	SPI 工作频率 [0 = 10 MHz, 1 = 20 MHz, 2 = 30 MHz]
Bits 7 - 6	必须为零

例如，值为 0x1C 将为 SPI 工作频率 20 MHz 生成.img。

文档修订记录

文档标题：AN76405 — EZ-USB® FX3™/FX3S™启动选项

文档编号：001-98024

版本	ECN	变更者	提交日期	变更说明
**	4802504	WEIZ	07/29/2015	本文档版本号为 Rev**，译自英文版 001-76405 Rev*D。
*A	4929871	LIP	09/29/2015	本文档版本号为 Rev*A，译自英文版 001-76405 Rev*E。
*B	5688204	AESATMP8	04/19/2017	更新徽标和版权。
*C	6388930	LIP	11/20/2018	本文档版本号为 Rev*C，译自英文版 001-76405 Rev*I。

销售、解决方案以及法律信息

全球销售和设计支持

赛普拉斯公司具有一个由办事处、解决方案中心、厂商代表和经销商组成的全球性网络。要想查找离您最近的办事处，请访问赛普拉斯所在地。

产品

Arm® Cortex® 微控制器	cypress.com/arm
汽车级产品	cypress.com/automotive
时钟与缓冲器	cypress.com/clocks
接口	cypress.com/interface
物联网	cypress.com/iot
存储器	cypress.com/memory
微控制器	cypress.com/mcu
PSoC	cypress.com/psoc
电源管理 IC	cypress.com/pmic
触摸感应	cypress.com/touch
USB 控制器	cypress.com/usb
无线连接	cypress.com/wireless

PSoC®解决方案

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6 MCU

赛普拉斯开发者社区

社区 | 项目 | 视频 | 博客 | 培训 | 组件

技术支持

cypress.com/support

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