请注意赛普拉斯已正式并入英飞凌科技公司。
此封面页之后的文件标注有“赛普拉斯”的文件即该产品为此公司最初开发的。请注意作为英飞凌产品组合的部分，英飞凌将继续为新的及现有客户提供该产品。

文件内容的连续性
事实是英飞凌提供如下产品作为英飞凌产品组合的部分不会带来对于此文件的任何变更。未来的变更将在恰当的时候发生，且任何变更将在历史页面记录。

订购零件编号的连续性
英飞凌继续支持现有零件编号的使用。下单时请继续使用数据表中的订购零件编号。

AN63787

使用 8 位异步接口配置 EZ-USB FX2LP GPIF 和 Slave FIFO 的示例

作者：Gayathri Vasudevan

相关项目：有

关联器件系列：CY7C68013A/14A/15A/16A

软件版本：Keil µVision 2

相关应用笔记：AN65209、AN66806、AN57322

获取最新的应用笔记或相关的项目文件，请访问 http://www.cypress.com/AN63787.

需要更多的参考代码？

我们期待您的声音。如需获取各种 FX2LP 代码范例，请访问我们的 USB 高速代码范例网页。

你在寻找 USB 3.0 外围控制器？

如需获取 USB3.0 系列产品信息，请访问 USB3.0 产品系列网页.

AN63787 描述了如何在手动模式和自动模式下配置 EZ-USB® FX2LP™ 中的通用可编程接口（GPIF）和 Slave FIFO，以便执行 8 位异步并行接口。使用两个背对背连接的 FX2LP 开发套件测试本应用笔记；第一个套件以主设备模式工作，第二个套件以从设备模式工作。

目录

1 简介 1
2 FX2LP 架构概述 2
2.1 Slave FIFO 模式 2
2.2 GPIF 模式 2
2.3 硬件 3
2.4 软件 3
3 项目目录结构 3
3.1 FX2LP 背对背连接 3
3.2 文档 3
3.3 驱动器 4
3.4 固件 4
4 FX2LP 固件项目文件结构 5
5 框图 7
6 硬件连接 8
7 GPIF 波形 9
7.1 读波形 9
7.2 写波形 10
7.3 导入 gpif.c 文件 11
8 GPIF 和 Slave FIFO 配置中的 FX2LP 手动操作模式 12
8.1 固件 12
8.2 GPIF 手动模式下的配置 FX2LP 12
8.3 在 Slave FIFO 的手动模式下配置 FX2LP 16
8.4 测试项目（手动模式） 17
9 GPIF 和 Slave FIFO 配置中的 FX2LP 自动操作模式 21
9.1 固件 21
9.2 在 GPIF 主设备模式下对 FX2LP 进行初始化 21
9.3 测试项目（自动模式） 23
10 调试 LED 23
11 七段显示屏 25
12 总结 25
13 相关文档 26
文档修订记录 27
全球销售和设计支持 28

1 简介

赛普拉斯 FX2LP 是业界内最流行的可编程高速 USB 控制器之一。 通过 GPIF，EZ-USB FX2LP 可以将局部总线作为外设的主设备，并能够实施各种协议。例如，使用 FX2LP 的 GPIF 模块可以支持 EIDE/ATAPI、打印机并行端口（IEEE P1284）、Utopia 和其他接口。在该示例中，它控制着另一个 FX2LP 的 Slave FIFO 接口。这样实现了使用 GPIF Designer（赛普拉斯提供的工具，用于创建 GPIF 波形描述符）来设计应用特定的物理层。该固件以赛普拉斯 FX2LP 固件框架为基础。通过使用两个背对背的 FX2LP 电路板硬件设置可以测试本应用笔记中所附带的固件项目，一个电路板作为主设备，另一个作为从设备。

2 FX2LP 架构概述

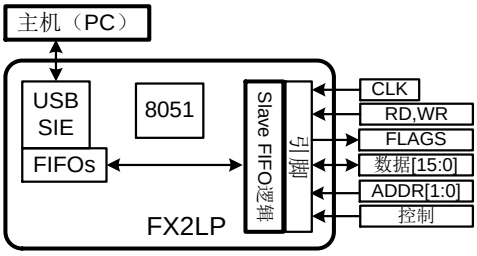
EZ-USB FX2LP 是一个灵活的 USB 2.0 外设控制器，用于处理 USB 2.0 的最大带宽。FX2LP 通过提供 GPIF 为外部器件提供了高速并行接口，用于优化 USB 的吞吐量。通过使用该 GPIF，可以在 FX2LP 端点 FIFO 和 GPIF 接口间传输数据。以下各节通过提供可配置的各种不同 FX2LP 模式，对 FX2LP 架构进行了简要说明。

2.1 Slave FIFO 模式

在 Slave FIFO 模式下，专用的 FX2LP 逻辑提供了控制和数据信号，用于将 USB 端点 FIFO 连接到外部 FIFO 控制器。除了数据总线和 FIFO 选择输入外，该接口还提供了 RD、WR 和 FIFO 标志等常用的 FIFO 控制信号，如图 1 所示。

有关 FX2LP Slave FIFO 接口的信息，请参考 EZ-USB 技术参考手册中的“Slave FIFO”一节和 AN61345 — 使用 FPGA 设计 EZ-USB FX2LP Slave FIFO 接口，该应用笔记提供了一个详细的设计示例。

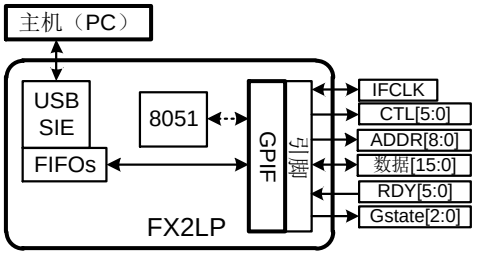
图 1. Slave FIFO 模式下的 FX2LP 引脚



2.2 GPIF 模式

GPIF 是一个可编程的 8 位或 16 位并行接口，通过在 EZ-USB FX2LP 和各种不同类型的外部设备之间提供无缝接口，从而可以降低系统成本。GPIF 有效时，接口引脚作为主设备控制 RAM、FIFO 或外部处理器等外部设备。图 2 显示的是 GPIF 接口信号。

图 2. GPIF 模式下的 FX2LP



更多有关 GPIF 的信息，请参考 EZ-USB 技术参考手册中的“GPIF”一节。AN66806 - EZ-USB® FX2LP™ GPIF 设计入门， 介绍 GPIF 单元和它的图形设计工具 -GPIF Designer，其中包含了一个范例，演示如何在一个 GPIF 设计中加入 USB 连接 。 AN57322 — 通过 GPIF 使 SRAM 与 FX2LP 相连介绍了如何使用 8 位异步接口和 GPIF 自动模式将赛普拉斯 CY7C1399B SRAM 连接至 FX2LP。

本应用笔记介绍了如何在 FX2LP 的 GPIF 和 Slave FIFO 中实现手动模式和自动模式。它描述了基于 FX2LP 至 FX2LP 背对背的链接平台的手动模式和自动模式，其中一个 FX2LP 工作于主设备（GPIF）模式，另一个工作于从设备模式：

- **手动模式：**两个 FX2LP 芯片相连，其中一个在 GPIF 手动模式下工作，另一个在 Slave FIFO 手动模式下工作。在某些应用中，数据被传送到外部器件或主机前，EZ-USB CPU 必须在数据路径上解释或修改数据。在本节中，将实现双向并行接口，从而说明主设备 CPU（GPIF 手动模式中的 FX2LP）和从设备 CPU（Slave FIFO 手动模式中的 FX2LP）如何进行数据修改。
- **自动模式：**两个 FX2LP 芯片相连，其中一个工作于 GPIF 自动模式，另一个工作于 Slave FIFO 自动模式。通常 EZ-USB CPU 被从数据路径移除，以获取最大带宽。

2.3 硬件

该示例使用两个 FX2LP 开发板（CY3684）作为开发和测试平台。附带文件中的硬件文件夹内提供了开发套件（DVK）的详细原理图。更多有关该电路板的信息在 EZ-USB 开发套件用户指南中的“高级开发板”一节提供，该文件位于：

C:\Cypress\USB\CY3684_EZ-USB_FX2LP_DVK\1.0\Documentation（DVK 安装后）。

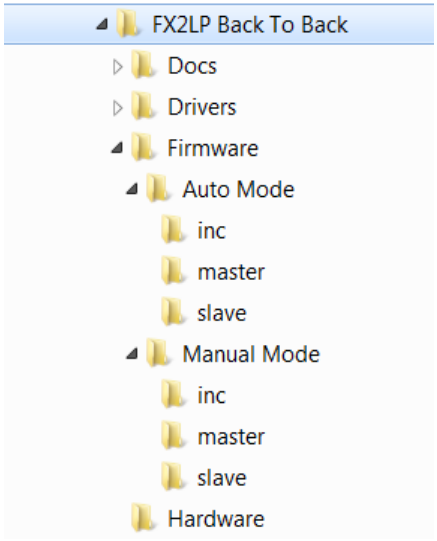
2.4 软件

- 控制中心：在 Suite USB 3.4 中提供。
- Keil µVision 2 IDE：4 KB 评估版本适用于 CY3684 DVK。如果需要完整的版本，请联系 Keil 公司。
- GPIF Designer：在这里提供。

3 项目目录结构

图 3 显示的是附带文件中的文件夹结构。

图 3. 附带文件中的文件夹



3.1 FX2LP 背对背连接

该项目文件夹是本应用笔记附带的内容，并包括了以下内容：

- 文档
- 驱动器
- 固件
- 硬件

3.2 文档

文件夹路径： *FX2LP Back To Back\Docs*

该文件夹包含 FX2LP 数据手册和技术参考手册。

3.3 驱动器

文件夹路径： *FX2LP Back To Back\Drivers*

该文件夹包含了 *CyUSB.inf* 以及用于 FX2LP 的 *CyUSB.sys* 文件；下表所示路径内的 *CyUSB.inf* 应用到您当前所使用的操作系统中。

操作系统	文件夹路径
Windows XP 32 位	wxp\x86
Windows XP 64 位	wxp\x64
Windows 7 32 位	wlh\x86
Windows 7 64 位	wlh\x64
Windows Vista 32 位	wlh\x86
Windows Vista 64 位	wlh\x64

3.4 固件

文件夹路径： *Back To Back\Firmware*

该文件夹包括以下各文件夹：

- 手动模式
- 自动模式

手动模式和自动模式文件夹包含了主设备和从设备的子文件夹。它们还包含一个 inc 文件夹，该文件夹含有项目中使用的头文件。这些头文件适用于主设备和从设备项目，因此对这些头文件进行更改会反映在主设备和从设备项目上（修改 inc 文件夹中任何头文件后，都需要重新构建主设备和从设备项目）。

- 主设备

  文件夹路径： *FX2LP Back To Back\Firmware\Manual Mode\master* and *FX2LP Back To Back\Firmware\Auto Mode\master*

  该文件夹包含主设备 FX2LP 所需的固件； *master.uv2* 为项目文件。该文件夹还包含 *master.hex* 和 *master.iic* 文件。您可以通过加载 *master.hex* 文件来编程 FX2LP 的 RAM，或者通过加载 *master.iic* 文件来编程大容量的 EEPROM。

- 从设备

  文件夹路径：*FX2LP Back To Back\Firmware\Manual Mode\slave* and *FX2LP Back To Back\Firmware\Auto Mode\slave*

  该文件夹包含从设备 FX2LP 所需的固件；*slave.uv2* 是项目文件。它还包含了 *slave.hex* 和 *slave.iic* 文件。您可以通过加载 *slave.hex* 文件来编程 FX2LP 的 RAM，或者通过加载 *slave.iic* 文件来编程大容量的 EEPROM。

- inc

  文件夹路径：*FX2LP Back To Back\Firmware\inc*

  该文件夹包含下面各头文件：*fx2.h*、*fx2regs.h*、*fx2sdly.h* 和 *syncdly.h*。

4 FX2LP 固件项目文件结构

当点击打开 *master.uv2* 或 *slave.uv2* 文件时，Keil µVision 2 会打开相应的 FX2LP 项目。窗口左侧将出现一个文件树，它属于该项目的一部分（请见图 4）。

表 1 列出了这些文件，并解释它们的相应函数。

图 4. 使用 Keil IDE 打开 FX2LP 固件项目

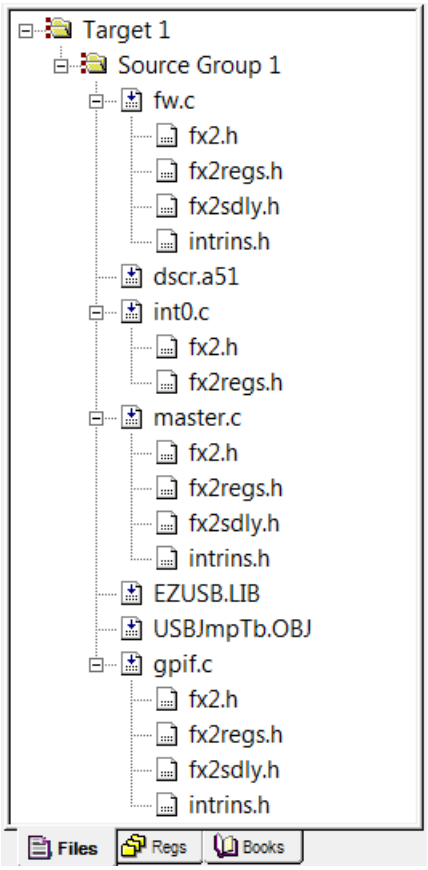


表 1. FX2LP 固件项目文件和说明内容

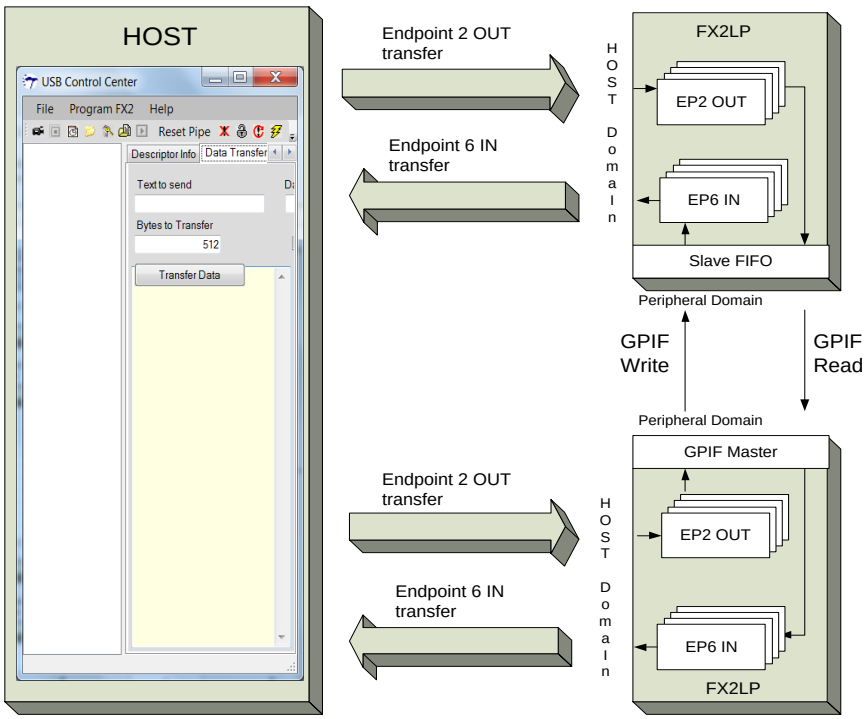
文件	说明
fw.c	该文件是赛普拉斯编写的固件框架的一部分，用于处理低级的 USB 细节。该文件无需修改。Fw.c 包含项目的主函数，每次启动它都将调用 `TD_Init` 函数，并在操作过程中重复调用 `TD_Poll` 函数。该无限循环也会处理 CONTROL 端点（EP0）SETUP 数据包。对于 `GET_DESCRIPTOR` 请求，它们使用了 *dscr.a51* 中提供的描述符数据。对于其它主机请求，将调用您的应用来处理各种操作，如更改某个接口的备用设置。
dscr.a51	它是一个包含了用户特定 USB 器件描述符数据的 8051 汇编语言模块。该文件包括.db（定义字节）声明，用于列出描述符表格数据。
int0.c	该文件包括 INT0#引脚的中断服务子程序（ISR），外设通过它来发出长度为零的数据包（ZLP）。
master.c/slave.c	这是用户应用。该模块借助于 USB Framework（USB 框架）进行编写本应用代码： 用户编写 `TD_Init` 和 `TD_Poll` 函数，从而使它们符合自定义应用。 *Fw.c* 调用代码来具体命名代码中的函数，以适应各种不同设备通过 Endpoint 0 的请求。赛普拉斯代码模板（*peripheral.c*）通过提供包含了所有函数存根的代码帧架节省了创建函数的工作。 用户提供的 ISR 会处理应用程序所使用的中断。大部分这类程序是简单的应答事项；如果不需要执行其他行动，不用修改它们。
EZUSB.LIB	该文件包括 EZ-USB 库函数的二进制。更多相关信息，请参考 EZ-USB 开发套件用户指南中的“EZ-USB 库”一节，该文件位于：*C:\Cypress\USB\CY3684_EZ-USB_FX2LP_DVK\1.0\Documentation*。
USBJmpTb.OBJ	此文件包括了 USB 中断的中断向量和跳转表。如果该目标文件被链接到您的项目，那么在使能中断前需要使能自动向量。由跳转表调用的函数名称位于 *USBJMPTB.A51* 源文件内。
gpif.c	该文件由 GPIF Designer 生成，用于配置 FX2LP 的 GPIF。文本段中标记为“DO NOT EDIT ...”的部分不能进行修改。在这种情况下，该文件仅存在于 FX2LP 主设备项目中，因为主设备使用了 GPIF 接口信号来控制从设备。
fx2.h、*fx2regs.h*、*fx2sdly.h*、*intrins.h*	两个项目（主设备和从设备）都包含了这些头文件。在每个文件内将会详细介绍它们各自的功能。

更多有关 FX2LP 固件框架的信息，请参考 EZ-USB 开发套件用户指南中的“EZ-USB 固件框架”一节，该文件位于：*C:\Cypress\USB\CY3684_EZ-USB_FX2LP_DVK\1.0\Documentation*。

5 框图

图 5 显示的是用于演示该应用功能的系统级框图。

图 5. 系统级框图



两个 FX2LP 芯片相连：一个工作于 GPIF 模式，另一个工作于 Slave FIFO 模式。配置这两个芯片，从而使它们具有以下端点：

- EP2 — BULK OUT、512 个字节、四缓冲
- EP6 — BULK IN、512 个字节，四缓冲

从主机发送到主设备的 EP2 OUT 端点的数据被传输给从设备的 EP6 IN 端点。同样，从主机传输到从设备的 EP2 OUT 的数据被传输给主设备的 EP6 IN。因此，自从设备（主设备）的 EP6 IN 端读取的数据就是传输到主设备（从设备）上 EP2 OUT 的数据。

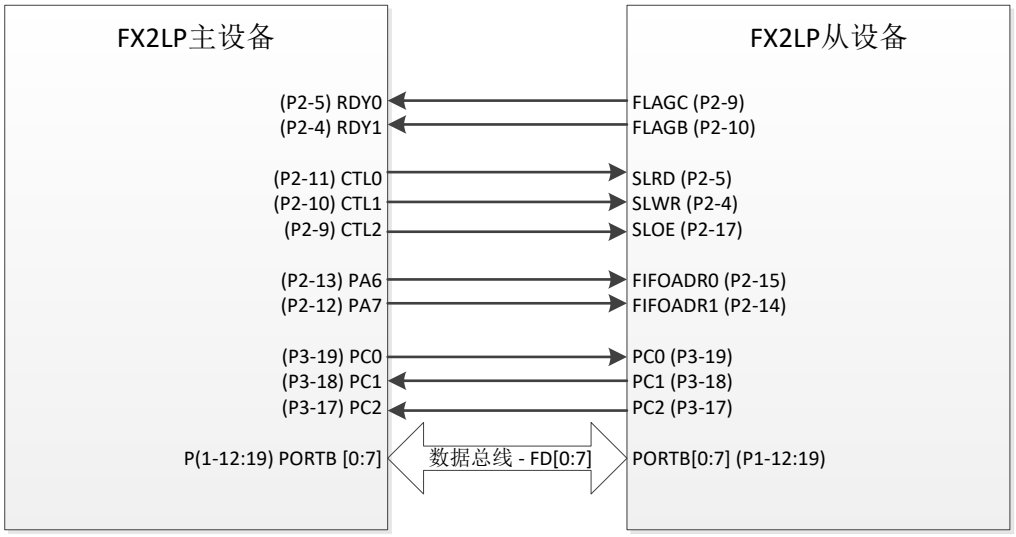
为了演示自动模式和手动模式间的区别，需要在手动模式下修改主设备 FX2LP 和从设备 FX2LP 中的数据。

6 硬件连接

本节介绍了两个 FX2LP 开发套件间需要进行的硬件连接。

图 6 显示的是在手动操作模式下，两个 FX2LP DVK 间的硬件连接。使用跳线实现该连接，因为该项目没有专用的互连电路板。

图 6. 手动模式下的硬件连接（主设备和从设备都属于 128 引脚封装）



注意：在自动操作模式下，不需要 PC0 和 PC1 线。所有其他硬件连接都要保持它们在手动模式下的状态，如图 1 显示。

在图 6 中，括号内的标记表示 FX2LP DVK 中各头文件的名称。例如，P2-5 表示 FX2LP DVK 中端口 2 的引脚 5。数据总线宽度为 8 位，并且是异步接口。

从设备 FX2LP 的硬件连接以及它们相应的用途

- 主设备引脚：
  - CTL[5:0]是可编程的控制输出，可将它们作为选通，读/写线或其他输出使用。该应用程序使用的控制信号（CTL0、CTL1 和 CTL2）连接到从设备的 SLRD、SLWR 和 SLOE。
  - RDY[5:0]为“就绪”输入，可对它们进行采样，并允许数据传输等待（插入等待状态）、继续或重复，直到该信号处于合适的电平为止。该过程使用了 RDY0 和 RDY1 来控制数据流。RDY0 连接着从设备的 FLAGC（EP2 空标志），而 RDY1 连接着从设备的 FLAGB（EP6 满标志）。
  - 主设备 FX2LP 使用端口 A 引脚[6, 7]来驱动由主设备访问的 Slave FIFO 地址。
  - PC2（`SLAVEREADY`）是从设备到主设备的输入，表示目前从设备已经复位，其固件准备就绪。这是为了防止主设备在从设备复位前或在复位过程中读取其 EP6 FIFO 中任何无效值。
  - 对于手动操作模式，从设备和主设备间进行的交换需要其他两条线来实现主出从入的传输。PC0（`Txn_Over`）和 PC1（`Pkt_Committed`）用于实现此目的。后面各章节说明了更多有关这些引脚使用情况的信息。
- 从设备引脚：
  - FLAGB 和 FLAGC 用于报告 Slave FIFO 的状态。
  - FLAGB — EP6FF（端点 6 满标志）表示 EP6 FIFO 的“满”状态。
  - FLAGC — EP6FF（端点 2 空标志）表示 EP2 FIFO 的“空”状态。
  - 所使用的 Slave FIFO 控制引脚分别是 SLOE（从设备输出使能）、SLRD（从设备读取）、SLWR（从设备写入）和 FIFOADR[1:0]（FIFO 选择）。
  - FIFOADR[1:0]引脚用于选择与数据总线相连接的四个 FIFO（由外部主设备控制）。
  - PC2 是从设备的输出，在从设备复位后它被设置为高电平，因此准备好了它的固件。

FD[0:7]

这是被配置为 8 位数据总线的端口 B。如果设置任意 EPxFIFOCG 寄存器的 WORDWIDE（EPxFIFOCG.0）位，那么端口 D 将被配置为 FD[8:15]。所有四个 WORLDWIDE 位的或操作，会导致 PORTD 成为 PORTD 或者 FD[15：8]。每一个单独的 WORLDWIDE 位指示每个单独的端点数据如何传递。这种传递使用 8-bit 接口如何该过程使用了一个 8 位的接口。

7 GPIF 波形

通过 GPIF Designer 工具，可以创建波形描述符，以便对从设备 FX2LP 进行读和写操作。首先，您必须定义接口，然后使用该工具来创建波形。配置该接口后，请使用该接口上进行的通信来创建读和写波形。

有关使用赛普拉斯的 GPIF Designer 工具的指南，请参考 AN66806 — EZ-USB FX2LP GPIF 入门中介绍的内容。

GPIF 读和写波形遵循 N 迭代（S1 - S2 -...- 空闲）逻辑，其中 N 是数据传输计数值。通过将所需数据传输量加载到寄存器 GPIFTCB3:0 来指定该计数值。

更多有关寄存器的信息，请参考 EZ-USB 技术参考手册第 15 章中的内容。

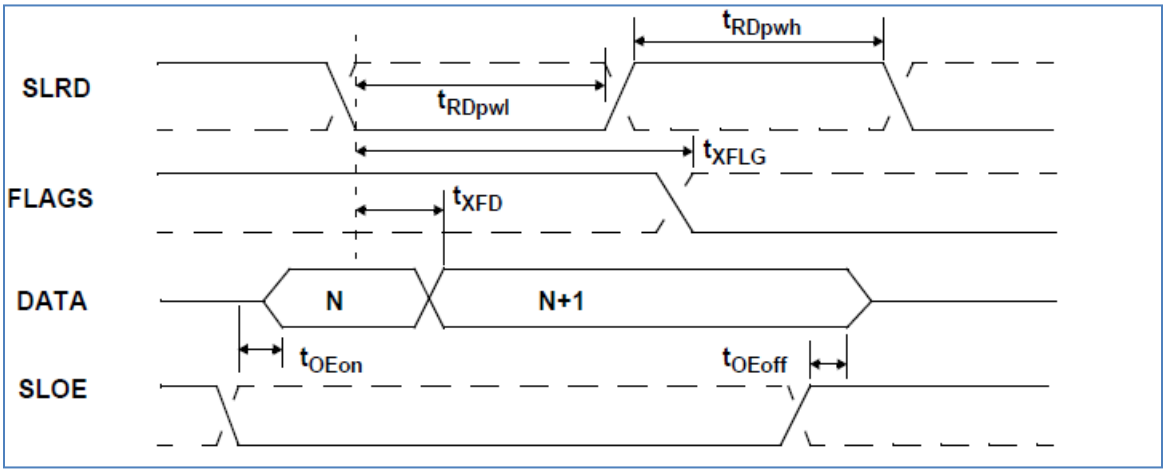
该应用笔记使用了两个波形，分别表示对从设备 FX2LP 进行读和写操作，如以下页面中的截图。

7.1 读波形

读波形将读取从设备 FX2LP 的 OUT 端点（EP2）中的数据，然后将该数据写入到主设备 FX2LP 的 IN 端点（EP6）内。它们必须满足参与 FX2LP 的读周期中各种信号的时序要求。

图 7 显示的是 FIFO 读取的时序图，图 8 显示的是波形。

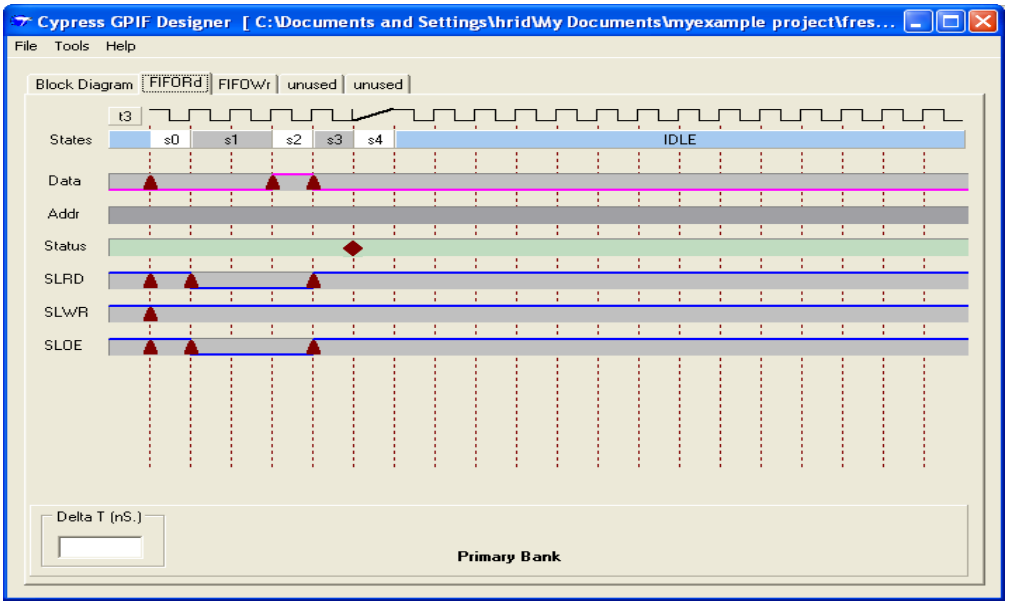
图 7. 带有时序参数的 Slave FIFO 读波形



必须满足以下各时序参数：

参数	说明	最小值（ns）	最大值（ns）
t_{RDpwl}	SLRD 低电平脉冲宽度	50	–
t_{RDpwh}	SLRD 高电平脉冲宽度	50	–
t_{XFLG}	从 SLRD 到 FLAGS 输出的传输时间延迟	–	70
t_{XFD}	从 SLRD 到 FIFO 数据输出传输延迟	–	15
t_{OEon}	从 SLOE 打开到 FIFO 数据生效	–	10.5
t_{OEoff}	从 SLOE 关闭到 FIFO 数据保持	–	10.5

图 8. GPIF Designer 工具中的 FIFO 读波形

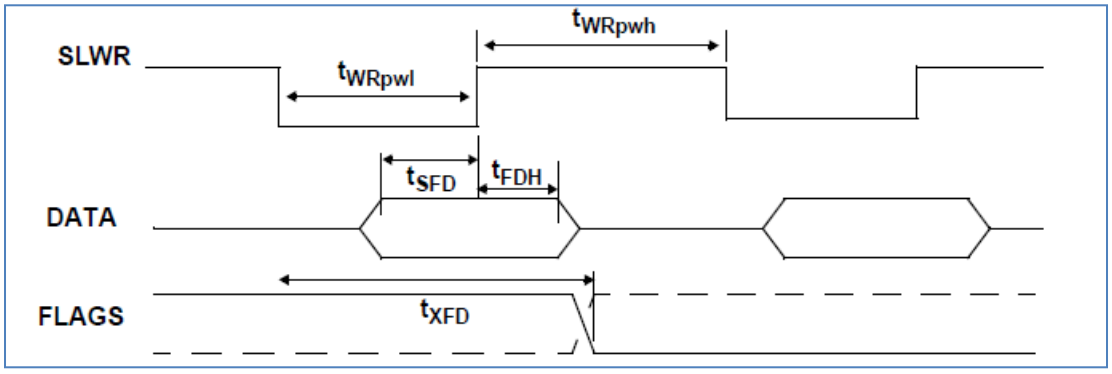


7.2 写波形

写波形将主设备 FX2LP 的 OUT 端点（EP2）上的数据写入到从设备 FX2LP 的 IN 端点（EP6）内。它们必须满足参与 FX2LP 的写周期中各种信号的时序要求。

图 9 显示的是 FIFO 编写的时序图，而图 10 显示的是波形。

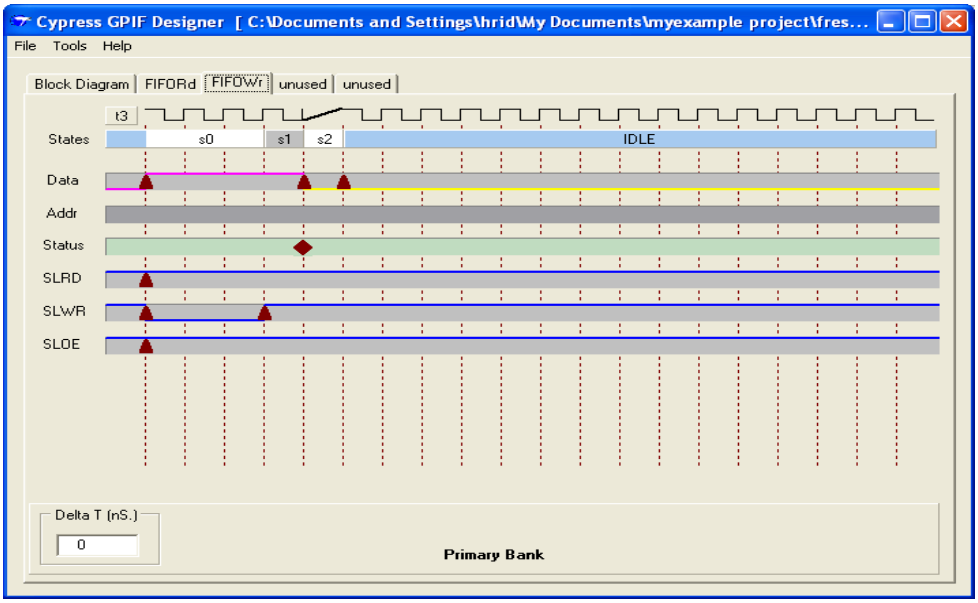
图 9. 带有时序参数的 Slave FIFO 写波形



必须满足以下各时序参数：

参数	说明	最小值（ns）	最大值（ns）
t_{WRpwl}	SLWR 低电平脉冲宽度	50	–
t_{WRpwh}	SLWR 高电平脉冲宽度	70	–
t_{SFD}	从 SLWR 到 FIFO 的数据设置时间	10	–
t_{FDH}	从 FIFO 数据到 SLWR 的保持时间	10	–
t_{XFD}	从 SLWR 到 FLAGS 输出的传输时间延迟	–	70

图 10. FIFO 写波形

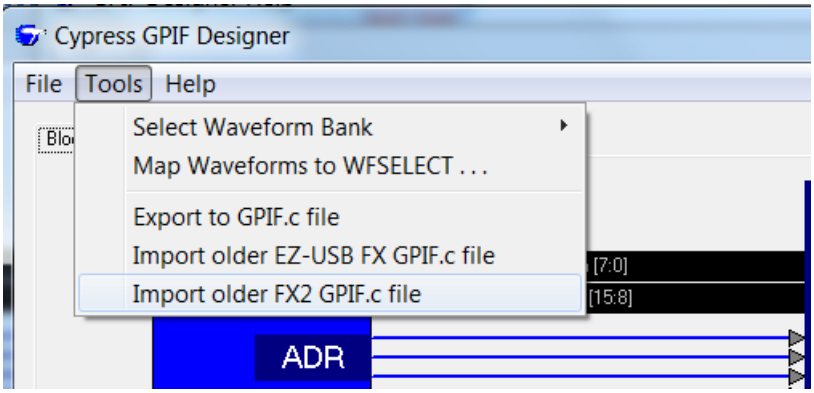


7.3 导入 gpif.c 文件

要想查看图 8 和图 10 中所示的波形，需要导入固件项目中包含的 *gpif.c* 文件。

1. 打开 GPIF Designer。
2. 点击 **Tools**（工具）选项卡。选择 **Import older FX2 GPIF.c file** 并找到 *FX2LP Back To Back\Firmware\Manual Mode\master* 中的 *gpif.c* 文件（如图 11 所示）。现在，您可以在 **FIFORd** 选项卡中查看 Slave FIFO 读波形，并能够在 **FIFOWr** 选项卡中查看 Slave FIFO 写波形。

图 11. 导入 *gpif.c* 文件



8 GPIF 和 Slave FIFO 配置中的 FX2LP 手动操作模式

8.1 固件

要想查看主设备的代码，请打开 *FX2LP Back To Back\Firmware\Manual Mode\master*\路径中的 *master.uv2*。要想查看从设备的代码，请打开 *FX2LP Back To Back\Firmware\Manual Mode\slave*\路径中的 *slave.uv2*。

8.2 GPIF 手动模式下的配置 FX2LP

- 更多有关固件所使用的寄存器的信息，请参考 EZ-USB 技术参考手册中的“第 15 章：寄存器”。
- 将 REVCTL.0 位设置为‘1’。这样可以对 IN 和 OUT 数据包进行编辑、传输、跳过和发送等操作。
- 将 REVCTL.1 位设置为‘1’；如果它从 AUTOUT = 0 转换至 AUTOOUT = 1，将禁用 OUT 端点的自动激活。

```
REVCTL = 0x03;      // CPU can source and edit both IN and OUT packets
SYNCDELAY;
```

- 将 EP2FIFOCFG 和 EP6FIFOCFG 设置为‘0’，这样会使各端点进入 8 位手动模式。

```
EP2FIFOCFG = 0x00; // manual out mode, 8 bit data bus
SYNCDELAY;
EP6FIFOCFG = 0x00; // manual in mode, 8 bit data bus
SYNCDELAY;
```

- 分别将 PC0（Txn_Over）、PC1（Pkt_Committed）和 PC2（SLAVEREADY）配置为输出引脚、输入引脚和输入引脚。

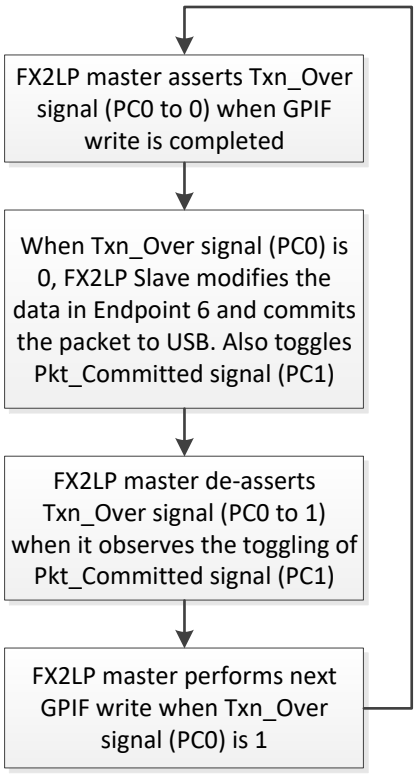
```
/* #define statements before TD_Init()*/
#define Txn_Over PC0
#define Pkt_Committed PC1
#define SLAVEREADY PC2

/* code snippets from TD_Init() */
PORTCCFG = 0x00; //configure port C as an I/O port
OEC= 0xF9;       /*Txn_Over configured as output, Pkt_Committed configured as input,
                  SLAVEREADY configured as input*/
PC0=1;
```

- 为使 Slave FIFO 处于“手动”模式，需要在从设备和主设备之间连接两条线以实现交换。对两条线 Txn_Over 和 Pkt_Committed 进行命名。当 GPIF 数据传输完成后，主设备会确认 Txn_Over 和 Pkt_Committed。当它发送一个数据包时，从设备将切换 Pkt_Committed 线。
- Txn_Over，取消确认状态：Txn_Over = 1 表示主设备可以开始进行下一个 GPIF 写数据传输（将主设备的 EP2 OUT 中的数据写入到从设备的 EP6 IN 内）。
- Txn_Over，确认状态：Txn_Over = 0 表示主设备已经完成了 GPIF 写数据传输，并且从设备可以开始读取其 IN 端点（EP6）中的数据。
- Pkt_Committed：该信号每次发生切换都意味着从设备已经处理并发送了 GPIF 主设备之前所发送的数据包，现在它可以接收其他数据包。
- SLAVEREADY 输入通知主设备：从设备已被复位，并且它的固件已经启动并正在运行。

图 12 流程图显示的是在 FX2LP 主设备写入到从设备 FX2LP 期间，交换信号 Txn_Over 和 Pkt_Committed 的状态。

图 12. 在主设备写入和从设备读取期间，交换信号 Txn_Over 和 Pkt_Committed 的状态。



8.2.1 GPIF 手动输出模式

- 连续检查主设备中的 TD_Poll 函数，从而检查是否切换了 Pkt_Committed 引脚。

```
if(Pkt_Committed == ~b)
 {
  b = Pkt_Committed; /* store the current state of Pkt_Committed in
                        variable b so that the next toggle can be detected */
  Txn_Over = 1;
 }
```

- 检查到发生切换时（即从设备的 IN 端点中刚接收到的数据包被发送到它的 USB 域时），将取消确认 Txn_Over，以表示主设备可以开始进行下一个 GPIF 数据传输。

```
if( (!( EP2468STAT & 0x01 )) )    /*if EP2 not empty, modify packet and commit it to
                                    peripheral domain */
    {   SYNCDELAY;
        EP2FIFOBUF[0] = 0x01;     // editing the packet
        SYNCDELAY;
        EP2FIFOBUF[1] = 0x02;
        SYNCDELAY;
        EP2FIFOBUF[2] = 0x03;
        SYNCDELAY;
        EP2FIFOBUF[3] = 0x04;
        SYNCDELAY;
        EP2FIFOBUF[4] = 0x05;
        SYNCDELAY;
        EP2BCH = 0x02;
        SYNCDELAY;
        EP2BCL = 0x00;                      // commit edited pkt. to interface fifo
        SYNCDELAY;
    }
  if ( ! (EP24FIFOFLGS & 0x02) )
```

```
{
    if((SLAVENOTFULL) && (Txn_Over == 1))
    {
       if( GPIFTRIG & 0x80 )              // if GPIF interface IDLE
           {
            PERIPH_FIFOADR0 = 0;          // FIFOADR[1:0]=10 - point to peripheral EP6
            PERIPH_FIFOADR1 = 1;
            SYNCDELAY;
            if(enum_high_speed)
            {

               SYNCDELAY;
               GPIFTCB1 =   0x02;         // setup transaction count 512
                SYNCDELAY;
               GPIFTCB0 =   0x00;
               SYNCDELAY;
             }
             else
             {
                 SYNCDELAY;
                 GPIFTCB1 = 0x00;         // setup transaction count 64
                 SYNCDELAY;
                 GPIFTCB0 = 0x40;
                 SYNCDELAY;
             }
             SYNCDELAY;
             GPIFTRIG = GPIFTRIGWR | GPIF_EP2; /* launch GPIF FIFO WRITE Transaction
                                                  from EP2 */
             SYNCDELAY;
             while( !( GPIFTRIG & 0x80 ) ) // poll GPIFTRIG.7 GPIF Done bit
                   {
                        ;
             }
             SYNCDELAY;
             Txn_Over = 0;    /*assert Txn_Over signal to indicate that packet has been
                              transmitted */
       }
    }
 }
```

- 如果 EP2 OUT 端点的 USB 域中存在一个数据包，那么数据包中前 5 个字节会被修改，然后被发送。
- 这时，如果 FX2LP 从设备的 EP6 IN 端点中存在空间，并且 `Txn_Over` 未被激活，那么将触发 GPIF 写数据传输。
- GPIF 写操作：从主设备 FX2LP 的 EP2 OUT 中写入从设备 FX2LP 的 EP6 IN。
- 数据传输结束后（通过轮询 GPIFTRIG 寄存器中的“完成”位来确定），将确认 `Txn_Over`，以通知从设备该数据传输已经结束，并且它可以开始读取该数据包。

8.2.2 GPIF 手动输入模式

在手动模式下实现 GPIF 读取时需要的固件如下：

```
if(SLAVEREADY)                  //checking if slave firmware is ready i.e if PC2=1
{
if ( GPIFTRIG & 0x80 )          // if GPIF interface IDLE - triggering gpif IN transfers
{
  PERIPH_FIFOADR0 = 0;
  PERIPH_FIFOADR1 = 0;          // FIFOADR[1:0]=00 - point to peripheral EP2
  SYNCDELAY;
  if ( SLAVENOTEMPTY )                  // if slave is not empty
  {
    if ( !( EP68FIFOFLGS & EP6FULL ) )// if EP6 FIFO is not full
    {
      if(enum_high_speed)
      {
    SYNCDELAY;
```

```
        GPIFTCB1 = 0x02;              // setup transaction count 512
        SYNCDELAY;
        GPIFTCB0 = 0x00;
        SYNCDELAY;
  }
else
  {
   SYNCDELAY;
   GPIFTCB1 = 0x00;              // setup transaction count 64
        SYNCDELAY;
   GPIFTCB0 = 0x40;
   SYNCDELAY;
      }

   GPIFTRIG = GPIFTRIGRD | GPIF_EP6; // launch GPIF FIFO READ Transaction to EP6 FIFO
   SYNCDELAY;
```

- 在 GPIF 手动输入模式中，检查从设备的“空”标志，从而验证从设备中是否存在需要读取的数据包。如果取消确认空标志，那么 FIFO 读数据传输将开始读取从设备。
- GPIF 读操作：对从设备 FX2LP 的 EP2 OUT 引脚到主设备 FX2LP 的 EP6 IN 引脚执行读取操作。
- 连续轮询 GPIFTRIG.7 位，直到 GPIF 数据传输完为止。

```
      while( !( GPIFTRIG & 0x80 ) );      // poll GPIFTRIG.7 GPIF Done bit

      EP6FIFOBUF[ 4 ] = 0x05; //edit the last five packets before committing
      EP6FIFOBUF[ 3 ] = 0x04;
      EP6FIFOBUF[ 2 ] = 0x03;
      EP6FIFOBUF[ 1 ] = 0x02;
      EP6FIFOBUF[ 0 ] = 0x01;
      SYNCDELAY;
      SYNCDELAY;
      EP6BCH = 0x02;              //committing the packet
      SYNCDELAY;
      EP6BCL = 0x00;
      SYNCDELAY;
     }
  }
```

- 数据传输结束时，数据包中的前 5 个字节会被修改，然后将这 5 个字节发送给主机域（通过将这些字节数量写入到 EP6BCH/BCL 寄存器实现该操作）。
- 清除从设备复位过程中主机可在其 EP6 FIFO 中读取的所有无用数值，EP6 被复位，直到从设备不再维持 PC2 信号处于高水平状态为止。

```
// This will keep resetting Master EP6FIFO untill slave firmware starts to run
else {
        FIFORESET = 0x80;   // set NAKALL bit to NAK all transfers from host
        SYNCDELAY;
        FIFORESET = 0x06;   // reset EP2 FIFO
        SYNCDELAY;
        FIFORESET = 0x00; // reset EP6 FIFO
        SYNCDELAY;
    }
```

8.3 在 Slave FIFO 的手动模式下配置 FX2LP

该 TD_Init 函数分别将端点 2 和端点 6 配置为 OUT 端点和 IN 端点，这两个端点均工作于 8 位手动模式。

8.3.1 Slave FIFO 手动输入模式

下面这段代码可用于修改 Slave FIFO 所收到的数据，然后将其发送到 USB 的 FX2LP 从设备固件中（手动模式）：

```
PC2=1;          //making it 1 to show that slave firmware has started running
if (PC0 == 0 &&(!(EP68FIFOFLGS & 0x02))) /*if (PC0/Txn_Over = Zero) meaning master is
                                        done writing to Slave and EP6 FIFO is not empty*/
{
      EP6FIFOBUF[ 507 ] = 0x05; //edit the last five packets before committing
      EP6FIFOBUF[ 508] = 0x04;
      EP6FIFOBUF[ 509 ] = 0x03;
      EP6FIFOBUF[ 510 ] = 0x02;
      EP6FIFOBUF[ 511 ] = 0x01;
      SYNCDELAY;
      SYNCDELAY;
      EP6BCH = 0x02;              //committing the packet
      SYNCDELAY;
      EP6BCL = 0x00;
      SYNCDELAY;
      PC1 = ~PC1;              //toggle PC0 to indicate that the buffer has been passed
      while(  PC0 != 1);          //wait for PC0 to become high again.
}
```

- PC2 处于高电平状态，以便通知主设备从设备已复位，并且其固件已经开始运行。
- 在 Td_Poll 函数中，连续检查 PC0（与主设备 FX2LP 的 Txn_Over 相连），从而验证 Txn_Over 被确认，并指出主设备进行的 FIFO 写数据传输已经结束。该检查非常必要；否则，在整个数据包从主设备传输到 USB 域前，从设备 FX2LP 的 IN FIFO 中的数据可以先被发送到 USB 域内。发生这种情况时，主机将发现数据被划分为很多更小的数据包。
- 数据包被发送时，从设备将切换 PC1（与主设备 FX2LP 中的 Pkt_Committed 相连），以通知主设备该数据包已被发送，并且现在可以开始传输下一个数据包。当主设备发现 Pkt_Committed 已被切换，它将取消确认 Txn_Over 线。
- 当 PC0（Txn_Over）被确认，并且 EP6 IN 不为空时，将在从设备 FX2LP 中修改数据包最后 5 个字节，然后将其发送到主机域内。

8.3.2 Slave FIFO 的手动输出模式

处于 Slave FIFO 手动输出模式下的固件会检查 OUT 端点的空标志是否被取消确认。如果它被取消确认，那么从设备 FX2LP EP2 OUT 端点中数据包的最后 5 个字节将被编辑，然后被发送到外设域。

```
if( !( EP2468STAT & 0x01 ) )//if EP2 not empty, modify packet and commit it to peripheral side
       {    SYNCDELAY;
            EP2FIFOBUF[511] = 0x01; // editing the packet
            SYNCDELAY;
            EP2FIFOBUF[510] = 0x02;
            SYNCDELAY;
            EP2FIFOBUF[509] = 0x03;
            SYNCDELAY;
            EP2FIFOBUF[508] = 0x04;
            SYNCDELAY;
            EP2FIFOBUF[507] = 0x05;
            SYNCDELAY;
            EP2BCH = 0x02;
            SYNCDELAY;
            EP2BCL = 0x00;       // commit edited pkt. to interface fifo
            SYNCDELAY;
        }
```

8.4 测试项目（手动模式）

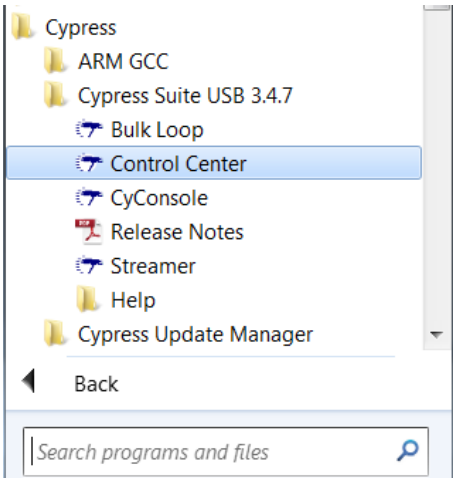
1. 下载并安装 Cypress SuiteUSB 3.4。该操作会安装一个被命名为控制中心的工具。
2. DVK 的开关 SW1 和 SW2 的位置：按照表 2 所提供的信息来保持这些开关。

表 2. DVK 开关位置

状态	正在执行的操作	SW1	SW2
1	编程 RAM	无需关注	无 EEPROM
2	编程大容量的 EEPROM	大容量的 EEPROM	EEPROM

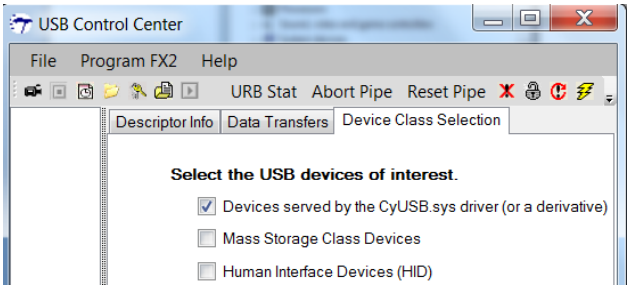
3. 按图 6 连接两个 FX2LP 开发板。将这两个开发板上的 SW1 和 SW2 设置为状态 1，如表 2 所示，并通过 USB 线缆将每个开发板连接到主机 PC 上。它们使用默认的内部描述符进行枚举。使用驱动器文件夹（AN63787\FX2LP Back To Back\FX2LP Back To Back\Drivers\）中的 CyUSB.inf 文件来绑定器件。有关绑定驱动器的详细信息，请参考驱动器文件夹中的“MatchingDriverToUSBDevice.htm”。
4. 依次选择 Start（）> All Programs > Cypress > Cypress Suite USB 3.4.7 > Control Center，实现启动 USB Control Center（请参见图 13）。

图 13. 打开 USB Control Center 应用



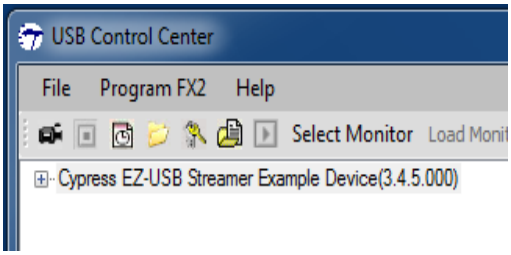
5. 在左侧面板上，您可看到 FX2LP 电路板以及与其相连的其他 USB 器件。如果只需查看 FX2LP 电路板，则在右侧面板上点击 Device Class Selection 选项卡，并取消勾选除 Devices served by the CyUSB.sys driver (or a derivative)外的所有选项，如图 14 所示。

图 14. 选择使用 *CyUSB.sys* 的器件的选项。



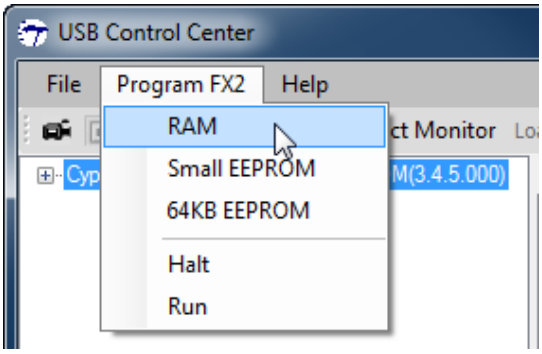
左侧面板将如图 15 所示。

图 15. USB 控制中心查找 FX2LP 电路板。



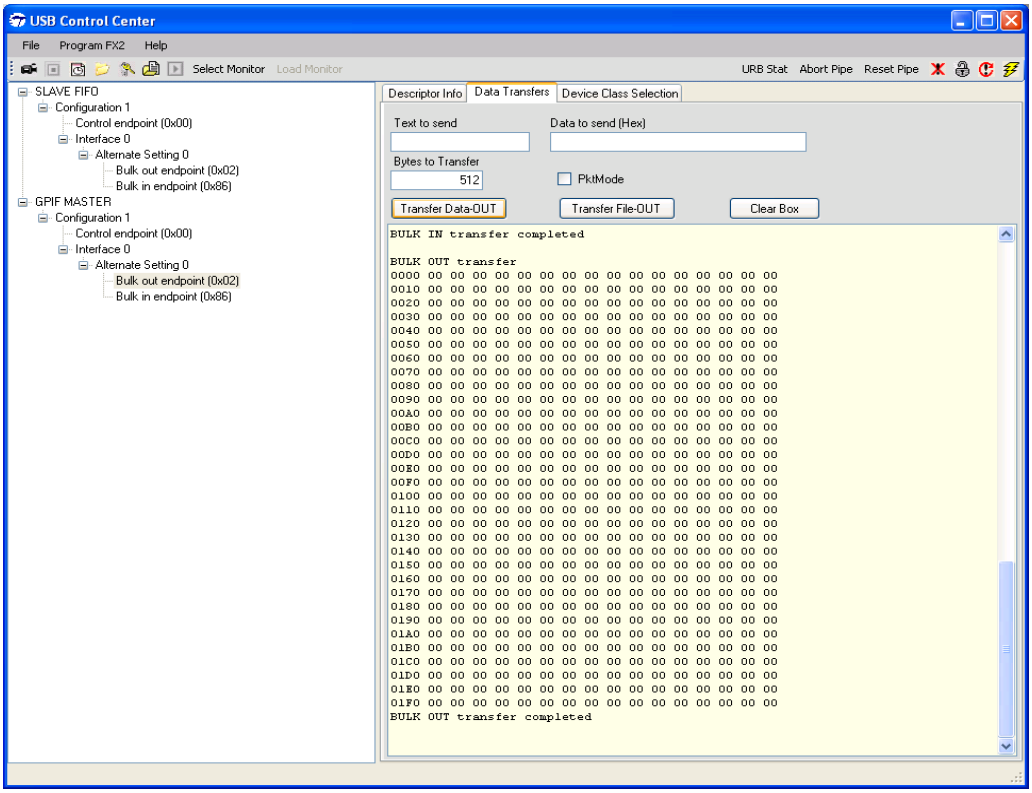
6. 现在，您可以将 FX2LP Back To Back\Firmware\Manual Mode\slave 中提供的 Keil-compiled slave.hex（用于 RAM）或 slave.iic（用于 EEPROM）文件加载到该 FX2LP DVK 电路板上。通过加亮显示 EZ-USB 输入并依次选择 Program FX2 > RAM 可将代码下载到 FX2LP RAM 内，如图 16 所示。编程成功时，一个表示编程成功的信息将出现在 Control Center 窗口的左下角。

图 16. 编程 FX2LP DVK RAM



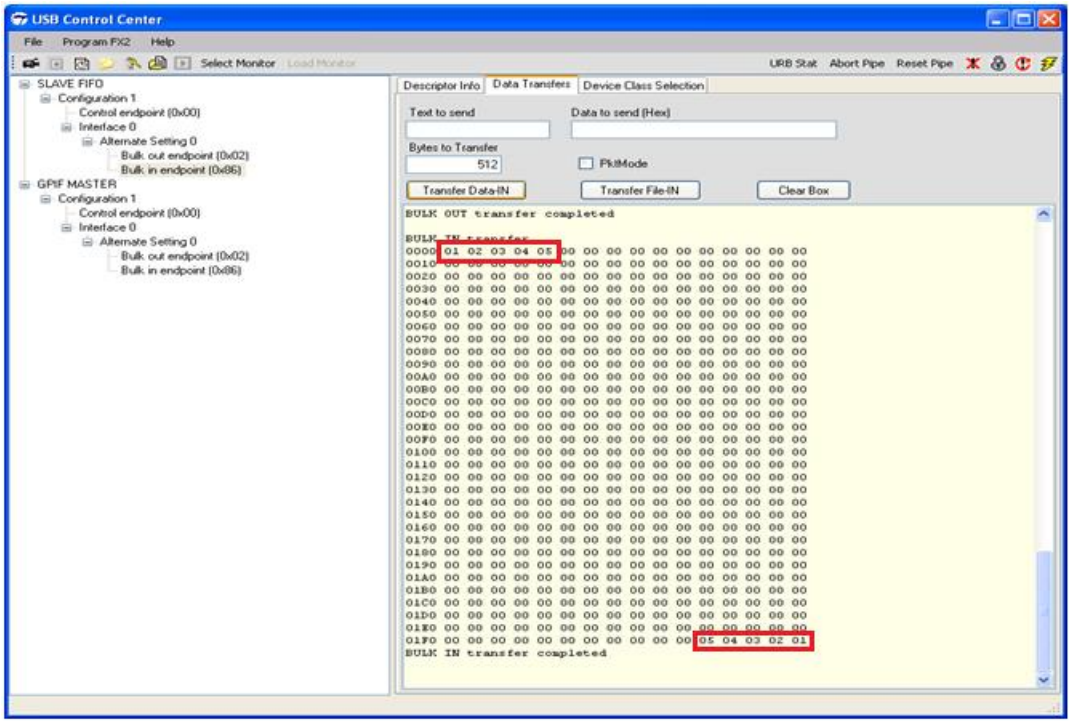
7. 弹出窗口出现，要求绑定驱动器。根据所使用的 OS，使用驱动器文件夹中正确的 CyUSB.inf 文件进行绑定。
8. 现在，通过使用 Control Center 工具，将附带文件中的 master.hex（在 FX2LP Back To Back\Firmware\Manual Mode\master 中提供）下载到主设备 FX2LP 中的 RAM 内。在左侧面板上选择主设备。依次选择 Program > RAM，对 RAM 进行编程。
9. 弹出窗口出现，要求绑定驱动器。使用驱动器文件夹中相同的 CyUSB.inf 文件来绑定。
10. 展开左侧面板中的文件树，直到打开完所有端点为止。
11. 使用 Control Center 传送主设备 FX2LP 中 EP2 的 512 个字节。点击右侧面板中的 Data Transfers（数据传输）选项卡。在左侧面板上，选择主设备的 Bulk out endpoint （0x02）。请检查 Bytes to Transfer 字段是否等于 512。点击 Transfer Data-OUT。请观察如图 17 中所示的数据传输。

图 17. GPIF 主设备输出数据传输



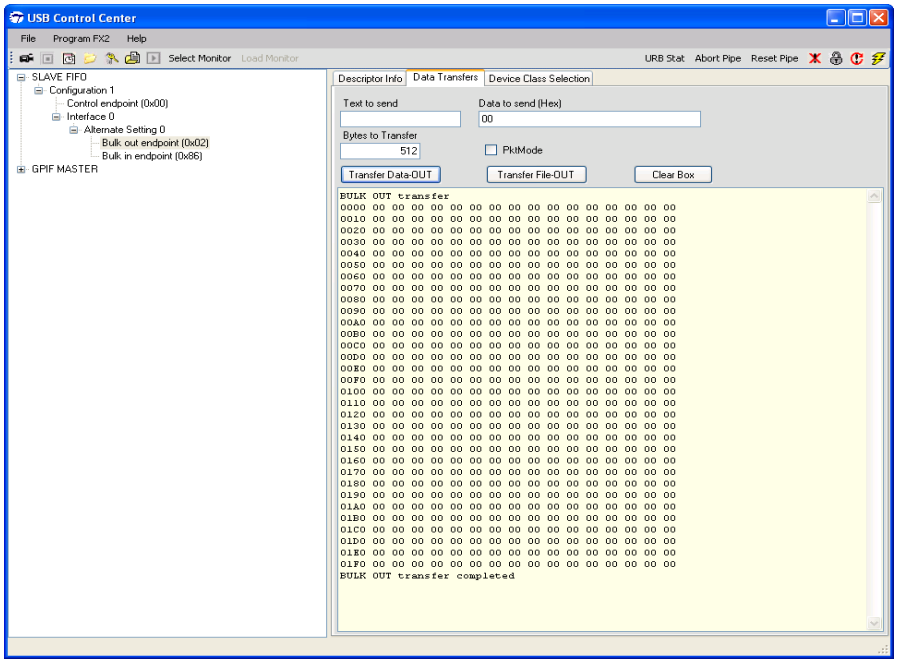
12. 从从设备 FX2LP 的 EP6 进行传输 512 字节的 BULK IN。在左侧面板上，选择从设备的 Bulk in Endpoint（0x86）。请检查 Bytes to Transfer 字段值是否等于 512。点击 Transfer Data-IN。所收到的数据应同传送到主设备 FX2LP 的 EP2 OUT 的数据一样，被修改的前 5 个和最后 5 个字节除外。主设备会修改前 5 个字节，而从设备则修改最后 5 个字节。请观察图 18 中所示的数据传输。

图 18. Slave FIFO 输入的传输



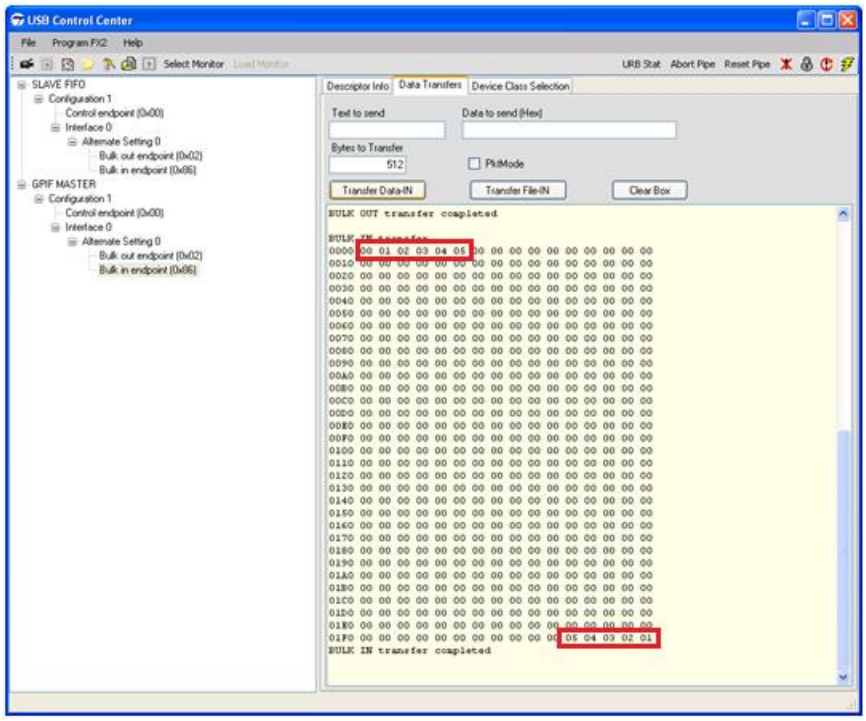
13. 通过使用跟第 12 步骤一样的程序从 Slave FIFO 的 EP2 传送容量为 512 个字节的数据。请观察图 19 中所示的数据传输。

图 19. Slave FIFO 输出传输



14. 通过使用跟第 13 步骤一样的程序从主设备 FX2LP 的 EP6 传输容量为 512 个字节的 Bulk IN。在读数据中，前 5 个和最后 5 个字节被修改。主设备会修改前 5 个字节，而从设备则修改最后 5 个字节。请观察图 20 中所示的数据传输。

图 20. GPIF 主设备输入传输



9 GPIF 和 Slave FIFO 配置中的 FX2LP 自动操作模式

两个 FX2LP 芯片相连，如图 6 所示，除 PC0 和 PC1 的连接外，在自动模式中不需要这两个芯片的互连。其中的一个芯片工作于主设备 GPIF 模式，另外一个工作于 Slave FIFO 模式。

9.1 固件

要想查看主设备的代码，请打开 *FX2LP Back To Back\Firmware\Auto Mode\master* 路径中的 *master.uv2*。要想查看从设备 FX2LP 的代码，请打开 *FX2LP Back To Back\Firmware\ Auto Mode\slave* 路径中的 *slave.uv2*。

9.2 在 GPIF 主设备模式下对 FX2LP 进行初始化

该 `TD_Init` 函数采用以下步骤实现整个初始化过程。

1. 将 EP2 和 EP6 配置为四缓冲端点，该缓冲区大小等于 512。
2. 复位 EP2 和 EP6 两个端点的 FIFO。
3. 将 EP2 和 EP6 两个端点的 FIFO 配置为 8 位自动模式。

9.2.1 GPIF 自动输出模式

GPIF 主设备的 TD_Poll 函数实现从 GPIF 主设备的 EP2 OUT 到 Slave FIFO 的 EP6 IN 的全部数据回送。通过使用 GPIF 的 **FIFOWr** 波形，将数据从主设备 FX2LP 上传输给从设备。

```
if( GPIFTRIG & 0x80 )                // if GPIF interface IDLE
  {
      if ( ! ( EP24FIFOFLGS & EP2EMPTY ) )  // if there's a packet in the peripheral domain
                                            //for EP2
      {
        PERIPH_FIFOADR0 = 0;      // FIFOADR[1:0]=10 - point to peripheral EP6 of Slave FX2LP
         PERIPH_FIFOADR1 = 1;
         SYNCDELAY;               // used here as "delay", refer to Synchronization
         if ( SLAVENOTFULL )            // if the slave is not full
        {
            if(enum_high_speed)
            {
               SYNCDELAY;
               GPIFTCB1 = 0x02;             // setup transaction count 512
                SYNCDELAY;
              GPIFTCB0 = 0x00;
               SYNCDELAY;
            }
            else
            {
               SYNCDELAY;
               GPIFTCB1 = 0x00;             // setup transaction count 64
                SYNCDELAY;
                GPIFTCB0 = 0x40;
               SYNCDELAY;
            }
              SYNCDELAY;
              GPIFTRIG = GPIFTRIGWR | GPIF_EP2;  //launch GPIF WRITE Transaction from EP2
             SYNCDELAY;
             while( !( GPIFTRIG & 0x80 ) );       // poll GPIFTRIG.7 GPIF Done bit

             SYNCDELAY;
        }
     }
  }
```

数据包一旦进入主设备 FX2LP 的 EP2 OUT，它将自动发送至外设域。因此，在 TD_Poll 函数中，每次在主设备 FX2LP 的 EP2 OUT 端点上存在数据时，您只需要触发 GPIF 写数据传输便可。

9.2.2 GPIF 自动输入模式

```
if (SLAVEREADY)
{
if ( GPIFTRIG & 0x80 )                  // if GPIF interface IDLE
  {
      PERIPH_FIFOADR0 = 0;
      PERIPH_FIFOADR1 = 0;              // FIFOADR[1:0]=00 - point to peripheral EP2
      SYNCDELAY;
      if ( SLAVENOTEMPTY )                // if slave is not empty
      {
         if ( !( EP68FIFOFLGS & EP6FULL ) )     // if EP6 FIFO is not full
         {
            if(enum_high_speed)
           {
              SYNCDELAY;
               GPIFTCB1 = 0x02;                // setup transaction count 512
               SYNCDELAY;
```

```
                GPIFTCB0 = 0x00;
                SYNCDELAY;
            }
            else
            {
               SYNCDELAY;
               GPIFTCB1 = 0x00;                // setup transaction count 64
                SYNCDELAY;
               GPIFTCB0 = 0x40;
               SYNCDELAY;
            }
             GPIFTRIG = GPIFTRIGRD | GPIF_EP6;  // launch GPIF READ Transaction to EP6

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

两个 FX2LP 的 EP6 IN 端点都处于自动模式，并且 AUTO IN 数据包的长度被设置为 512。因此，如果所传送的数据是 512（字节）的整数倍，它将自动从外设被发送到主机侧，并在主机上可用。只有它是 512（字节）的整数倍时，该固件才可以传输数据。

9.2.3 从设备自动输出和输入模式

由于各数据包被自动传输，因此除了初始化以外不需要 Slave FIFO 中的任何代码。在自动模式下，从设备固件负责控制 LED 并通过 PC2（SLAVEREADY）标志通知主设备其固件已经启动并正在运行。因此，从设备 FX2LP 的 TD_Poll 函数中没有任何数据处理代码。

9.3 测试项目（自动模式）

自动模式下的测试项目跟手动模式下的测试项目相同。有关连接框图，请参见图 6。在自动模式下，不需要 PC0 和 PC1 之间其他两个连接。在手动模式配置中才需要这种连接来进行信号交换。

被下载到主设备 FX2LP 和从设备 FX2LP 的代码如下：

- 主设备 FX2LP：*master.hex* 被包括在附带文件内，并提供在 *FX2LP Back To Back\Firmware\ Auto Mode\master* 路径中。
- 从设备 FX2LP：*slave.hex* 被包括在附带文件内，并提供在 *FX2LP Back To Back\Firmware\Auto Mode\slave* 路径中。
- 自动模式与手动模式之间唯一的区别是从从设备（主设备）FX2LP 的 EP6 IN 端点接收的数据与传送到主设备（从设备）FX2LP 的 EP2 OUT 的数据相同。

10 调试 LED

FX2LP DVK 具有四个 LED（D2、D3、D4 和 D5）可用于调试目的。该应用使用了三个 LED：D2、D4 和 D5，而 D3 始终被点亮。自动和手动模式下的主设备和从设备的固件包含 LED_Control 函数，该函数会控制 LED（D2、D4 和 D5）的行为。

在主设备和从设备中，由这些 LED 表示的器件特殊状态保持相同，无论操作模式如何（自动或手动）。下面是 LED 状态的说明内容：

- D2：只要固件运行于器件上，该 LED 会连续闪烁。
- D3：该 LED 不被设置为任何特殊状态，而是始终被点亮。

- D4：该 LED 表示器件中端点 2 为“非空”状态。 这便表示该器件的端点 2 中存在数据，并且可以将该数据传输到与其相连的其他器件内；例如，LED D4 在主设备（从设备）FX2LP 上被点亮，表示其 EP2 具有可被传输到从（主）设备上的数据。当该器件的 EP2 为空时，D4 会关闭。从算术角度来看，当 EP2 至少具有一个数据包时，该 LED 会被点亮。
- D5：该 LED 表示器件中端点 6 为“非满”状态。这便表示该器件的 EP6 没有完全被充满，它可以接收其他连接器件中的更多数据包；例如，LED D5 在从设备（主设备）上被点亮，表示其端点 6 没有完全被充满，并且可以接收主（从）设备中更多的数据包。当 EP6 被完全充满时，D5 将关闭。只有主机先输入器件中的数据后，才能将数据传输给该器件。从算术角度来看，当 EP6 中的数据包小于 4 时，该 LED 会点亮（谨记要将四缓冲用于 FIFO）。

主设备中 LED_Control()函数的代码

通过在同一个 *fx2.h* 文件（*FX2LP Back To Back\Firmware\Auto(Manual)\inc*）中注释#define LED_Enable 宏，可以关闭 LED D2-D5 功能。

```
//This function controls the state of D4 and D5 LEDs on the Master FX2LP DVK based upon the
state of EP2 and EP6 FIFOs. Also it blinks LED D2 while the firmware on the device is running
void LED_Control()
{
//For LED D4 and D5
if (!( EP24FIFOFLGS & EP2EMPTY ))        //LED D4 turns on whenever EP2 has got data to
                                         //transfer to Slave i.e. EP2 is not Empty
LED_On(bmBIT2);
else
LED_Off(bmBIT2);

if (!( EP68FIFOFLGS & EP6FULL ))         //LED D5 turns on whenever EP6 can accept data from
                                         //Slave i.e. EP6 is not Full
LED_On(bmBIT3);
else
LED_Off(bmBIT3);
//For LED D2, LED D2 blinks to indicate that firmware is running.
if (++LED_Count == Blink_Rate)            //Blink_rate=10000 for Seven_segment enabled and
                                          //30000 otherwise
  {
    if (LED_Status)
    {
      LED_Off (bmBIT0);
      LED_Status = 0;
    }
    else
    {
      LED_On (bmBIT0);
      LED_Status = 1;
    }
    LED_Count = 0;
  }}
```

11 七段显示屏

该固件通过使用 FX2LP DVK 中的七段显示屏来表示器件上 EP6 FIFO 缓冲区中数据包的数量。由于该应用将四缓冲用于 EP6，因此七段显示屏上的数值可以从 0（表示 EP6 为空）上升到 4（表示 EP6 为满）。

下面各步骤说明了七段显示屏的工作流程：

1. 将一个数据包从主机传输到主设备 EP2，并且从设备 EP6 FIFO 未满时，该数据包会被传输给从设备。这样从设备上七段显示屏的计数将增加 1。
2. 如果主机输入从设备 EP6 FIFO 中的数据，从设备显示屏上的计数值会减 1。这意味着主设备 EP2 中没有任何数据（该数据将被传输到从设备 EP6（如果这里有空间）），从而表示 EP6 未满。
3. 类似的逻辑方法用于在主机>从设备>主设备>主机间发生的传输。

请注意，如果您不想使用七段功能，那么您可以通过注释 *fx2.h* 文件（*FX2LP Back To Back\Firmware\Auto(Manual)\inc*）中的 #define Seven_segment 宏来完成这操作。您也可以直接在 Keil IDE 内部访问该文件。如果您注释了该宏，那么需要重新为主设备和从设备构建该项目。

通过使用以下代码，您可以控制七段显示屏：

```
// 7-segment readout
#define LED_ADDR           0x21
BYTE xdata Digit[] = { 0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x98, 0x88, 0x83,
0xc6, 0xa1, 0x86, 0x8e };

  #ifdef Seven_segment
  EZUSB_INITI2C();                      // initialize I2C for 7-seg readout
  #endif

// update 7-seg readout with number of IN packets in EP6 waiting for transfer to the host
#ifdef Seven_segment
  waiting_inpkts = (EP6CS & 0xF0)>>4;                //right shift by 4 bits
  EZUSB_WriteI2C(LED_ADDR, 0x01, &(Digit[waiting_inpkts]));
  EZUSB_WaitForEEPROMWrite(LED_ADDR);
#endif
```

12 总结

本应用笔记介绍了设置 GPIF 的信息，从而可以通过 8 位异步接口将数据传输到其他 EZ-USB FX2LP 的 Slave FIFO 内。它包含了硬件设置、GPIF 波形的创建以及 8051 代码的编写，它反复进行 USB 输入和输出。

该文档重点介绍了使用两个 EZ-USB FX2LP 电路板特殊的背对背电路板的设置内容。然而，本应用笔记所使用的很多概念和见解均作为主流应用的基本框架。

13 相关文档

下列文档提供了解 FX2LP 芯片，以及运行本应用笔记所附固件项目的附加信息。

- EZ-USB Development Kit User Guide: 本文档包含第一次如何使用 CY3684 套件的相关信息，该文件位于 *C:\Cypress\USB\CY3684_EZ-USB_FX2LP_DVK\1.0\Documentation* (在 FX2LP DVK 安装后).
- AN65209 - Getting Started with FX2LP: 该文档适用于新用户熟悉 FX2LP 的起点。
- AN66806 - Getting Started with EZ-USB® FX2LP™ GPIF: 此文档介绍 GPIF 单元及图形化设计工具 GPIF Designer，且包含如何将 USB connection 合并进 GPIF 设计的范例。
- AN57322 - Interfacing SRAM with FX2LP over GPIF: 此文档讨论如何通过通用可编程接 GPIF 连接 Cypress SRAM CY7C1399B 到 FX2LP。描述了如何使用 GPIF Designer 创建读写波形。此应用笔记作为参考用于 FX2LP 与其他 SRAMS 连接也有益处。
- AN70983 - Designing a Bulk Transfer Host Application for EZ-USB® FX2LP™/FX3™: 此文档介绍.NET 类库并演示如何使用“bulkloop”固件范例，创建 Windows 例程实现数据收发，并运行于 FX2LP or FX3 开发套件 (DVK).
- AN73609 - EZ-USB® FX2LP™/FX3™ Developing Bulk-Loop Example on Linux: 此文档描述如何利用 libusb， 开发针对于 Cypress EZ-USB FX2LP/FX3 产品，基于 Linux 操作系统的 USB 主机应用。赛普拉斯也提供 Linux SDK，用户可在 FX3 SDK web page 找到。
- AN74505 - EZ-USB® FX2LP™ - Developing USB Application on MAC OS X using LIBUSB: 此文档描述 libusb-1.0 如何被用于在 Cypress EZUSB FX2LP 产品上，开发基于 MAC OS X 10.6/10.7 的 USB 主机应用(Cocoa Application) 。此应用笔记仅供参考。赛普拉斯也提供 MAC SDK，用户可在 FX3 SDK web page 找到。
- 完整 USB Hi-Speed Code 范例列表, 请访问网页
- EZ-USB Technical Reference Manual: 此文档涵盖 the slave FIFOs and GPIF 的具体信息，包括寄存器级描述。

关于作者

姓名： Gayathri Vasudevan

职务： 应用工程师

联系邮箱： gaya@cypress.com

文档修订记录

文档标题：AN63787 — 使用 8 位异步接口配置 EZ-USB FX2LP GPIF 和 Slave FIFO 的示例

文档编号：001-98021

版本	ECN	提交日期	变更说明
**	4802499	07/14/2015	本文档版本号为 Rev**，译自英文版 001-63787 Rev*F。
*A	5792627	06/30/2017	更新徽标和版权。
*B	5930052	10/16/2017	Sunset review.
*C	7025221	11/17/2020	更新自英文版 001-63787 Rev*I， 更新模板 更新 FX2LP DVK 安装路径 格式微小调整

全球销售和设计支持

赛普拉斯公司拥有一个由办事处、解决方案中心、厂商代表和经销商组成的全球性网络。要找到离您最近的办事处，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 | 代码示例 | 项目 | 视频 | 博客 | 培训 | 组件

技术支持

cypress.com/support

此处引用的所有其他商标或注册商标归其各自所有者所有。

Cypress Semiconductor
An Infineon Technologies Company
198 Champion Court
San Jose, CA 95134-1709

赛普拉斯半导体公司，2010-2020 年。本文件是赛普拉斯半导体公司及其子公司，包括 Spansion LLC（“赛普拉斯”）的财产。本文件，包括其包含或引用的任何软件或固件（“软件”），根据全球范围内的知识产权法律以及美国与其他国家签署条约由赛普拉斯所有。除非在本款中另有明确规定，赛普拉斯保留在该等法律和条约下的所有权利，且未就其专利、版权、商标或其他知识产权授予任何许可。如果软件并不附随有一份许可协议且贵方未以其他方式与赛普拉斯签署关于使用软件的书面协议，赛普拉斯特此授予贵方属人性质的、非独家且不可转让的如下许可（无再许可权）（1）在赛普拉斯特软件著作权项下的下列许可权（一）对以源代码形式提供的软件，仅出于在赛普拉斯硬件产品上使用之目的且仅在贵方集团内部修改和复制软件，和（二）仅限于在有关赛普拉斯硬件产品上使用之目的将软件以二进制代码形式的向外部最终用户提供（无论直接提供或通过经销商和分销商间接提供），和（2）在被软件（由赛普拉斯公司提供，且未经修改）侵犯的赛普拉斯专利的权利主张项下，仅出于在赛普拉斯硬件产品上使用之目的制造、使用、提供和进口软件的许可。禁止对软件的任何其他使用、复制、修改、翻译或汇编。
在适用法律允许的限度内，赛普拉斯未对本文件或任何软件作出任何明示或暗示的担保，包括但不限于关于适销性和特定用途的默示保证。没有任何电子设备是绝对安全的。因此，尽管赛普拉斯在其硬件和软件产品中采取了必要的安全措施，但是赛普拉斯并不承担任何由于使用赛普拉斯产品而引起的安全问题及安全漏洞的责任，例如未经授权的访问或使用赛普拉斯产品。此外，本材料中所介绍的赛普拉斯产品有可能存在设计缺陷或设计错误，从而导致产品的性能与公布的规 格不一致。（如果发现此类问题，赛普拉斯会提供勘误表）赛普拉斯保留更改本文件的权利，届时将不另行通知。在适用法律允许的限度内，赛普拉斯不对因应用或使用本文件所述任何产品或电路引起的任何后果负责。本文件，包括任何样本设计信息或程序代码信息，仅为供参考之目的提供。文件使用人应负责正确设计、计划和测试信息应用和由此生产的任何产品的功能和安全性。赛普拉斯产品不应被设计为、设定为或授权用作武器操作、武器系统、核设施、生命支持设备或系统、其他医疗设备或系统（包括急救设备和手术植入物）、污染控制或有害物质管理系统中的关键部件，或产品植入之设备或系统故障可能导致人身伤害、死亡或财产损失其他用途（“非预期用途”）。关键部件指，若该部件发生故障，经合理预期会导致设备或系统故障或会影响设备或系统安全性和有效性的部件。针对由赛普拉斯产品非预期用途产生或相关的任何主张、费用、损失和其他责任，赛普拉斯不承担全部或部分责任且贵方不应追究赛普拉斯之责任。贵方应赔偿赛普拉斯因赛普拉斯产品任何非预期用途产生或相关的所有索赔、费用、损失和其他责任，包括因人身伤害或死亡引起的主张，并使之免受损失。
赛普拉斯、赛普拉斯徽标、Spansion、Spansion 徽标，及上述项目的组合，WICED，及 PSoC、CapSense、EZ-USB、F-RAM 和 Traveo 应视为赛普拉斯在美国和其他国家的商标或注册商标。请访问 cypress.com 获取赛普拉斯商标的完整列表。其他名称和品牌可能由其各自所有者主张为该方财产。