通过 GPIF 使 SRAM 与 FX2LP 相互连接

About this document

Scope and purpose

本应用笔记说明了如何使用通用可编程接口 (GPIF) 将赛普拉斯的 SRAM CY7C1399B 连接到 FX2LP。本手册描述了使用 GPIF Designer 来创建读和写信号的方式。它还提供了用于将 FX2LP 连接到其它 SRAM 的参考。

更多代码示例？我们听到了你的声音。

要访问各种 FX2LP 代码示例，**USB High-Speed Code Examples webpage**

您在寻找 USB 3.0 外设控制器吗？

要访问 USB 3.0 产品系列，请访问我们的产品系列 **USB 3.0 Product Family webpage**

相关项目

有

相关器件系列

CY7C68013A

相关应用笔记

AN65209, **AN66806**, **AN15456**

Table of contents

About this document **1**
Table of contents **1**
1 简介 **2**
2 硬件连接 **3**
3 设计 GPIF 接口 **4**
4 GPIF 波形 **7**
4.1 写波形 7
4.2 读波形 11
4.3 导出 GPIF 波形 14
4.4 uVision2 项目和固件 14
5 使用 Control Center 进行测试 **19**
6 总结 **22**
7 参考 **23**
8 附录 A **24**
文档修订记录 **32**

1 简介

GPIF 是一个 8 位或 16 位的可编程并行接口，它通过给 EZ-USB FX2LP™和外部设备提供一个无缝接口，降低了系统成本。它是一款高度可配置且灵活的硬件，通过它您可以充分使用您的 USB 2.0 设计。GPIF 适用于需要一个外部主器件进行信息交换的应用。通过 GPIF，EZ-USB FX2LP 可以作为局部总线的主设备，并能够实施各种协议。例如，使用 EZ-USB FX2LP 的 GPIF 模块可支持 EIDE/ATAPI、打印机并行端口 (IEEE P1284)、Utopia 以及其他接口。有关 FX2LP GPIF 主题的更多信息，请参阅 **AN66806**。

GPIF Designer 是赛普拉斯提供的一种工具，用于创建 GPIF 波形描述符。根据外设的读写周期时序，将外设连接到 FX2LP。创建波形后，可以将它们导出到 C 文件中，并将该文件添加到项目工作区内。本文档对定义接口、创建波形、导出波形以及将它们添加到项目工作区内的整个过程进行了相关介绍。在设计波形过程中，熟悉 EZ-USB FX2LP 开发套件的示例和文档，**EZ-USB FX2LP 技术参考手册**中第 10 章 (GPIF) 是非常有重要的。有关应用说明的完整列表，请单击**此处**。

Note: 下一节所介绍的示例只能在 FX2LP 的 100 引脚或 128 引脚器件中演示。这是因为 GPIFADR [0：8]引脚仅在 100 引脚和 128 引脚封装提供。

2 硬件连接

本节介绍 FX2LP 和 SRAM 之间需要建立的硬件连接。根据 SRAM 数据手册，同该器件进行通信时需要使用 3 个控制信号、一个地址和一个数据总线。SRAM 的 3 个控制信号分别为：一个芯片使能 (CE/) 信号、一个输出使能 (OE/) 信号和一个写使能 (WE/) 信号。地址和数据总线宽度分别为 15 位和 8 位。要想寻址大于 512 字节的存储位置 (GPIF Designer 只提供 9 位地址总线)，需要使用额外的端口 I/O 引脚。因此，需要将 PA[7:4]连接到 PA[12:9]，并将 PA[1:0]连接到 PA[14:13]。

PA[7:4]用于控制 A[12:9]。这样，固件可以访问 16 个 512 字节块，总连续存储空间 (8K)。PA[0]和 PA[1]用于访问 4 个 8K 大小的字节组，可访问整个 32K 字节的空间。

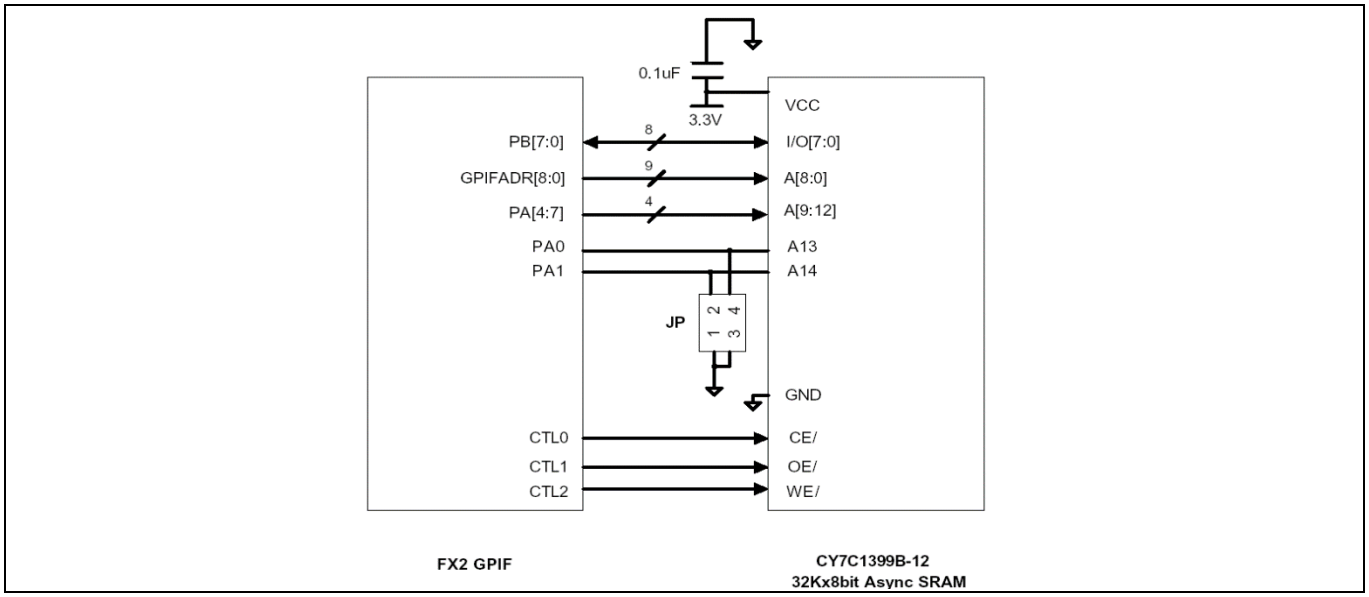
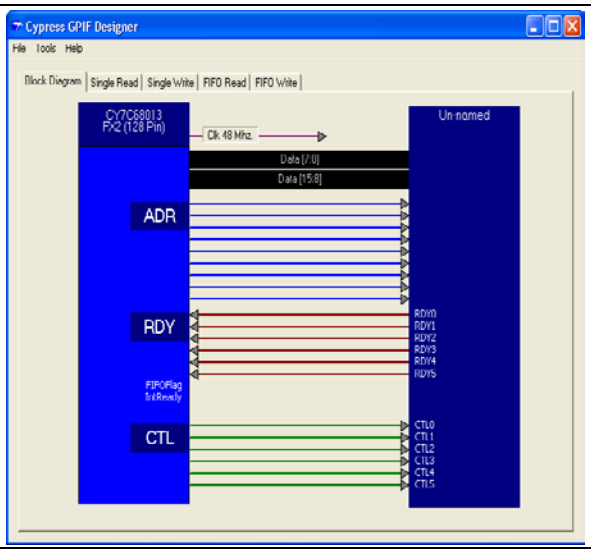


Figure 1 硬件连接框图

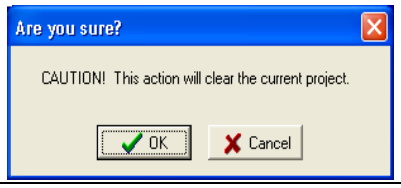
3 设计 GPIF 接口

通过 GPIF Designer 工具，可以创建波形描述符，以便对 SRAM 执行读和写操作。它与 **CY3684 EZ-USB FX2LP DVK** 设置一起提供。下面流程介绍的是如何定义接口和创建波形。

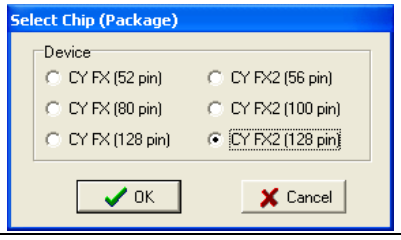
1. 启动赛普拉斯 GPIF Designer 工具。



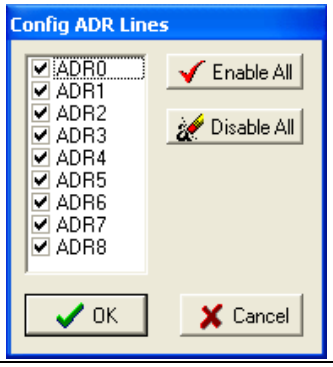
2. 依次选择 **File** > **New**。将显示以下窗口；请点击 **OK**。



3. 请在弹出的窗口内选择相应的器件，并点击 **OK**。



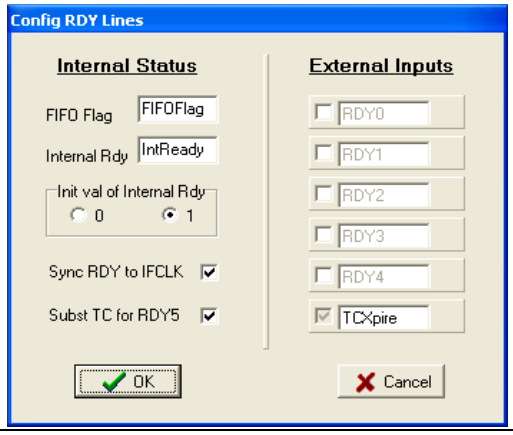
4. 右键单击 Un-named (无名) 标签，并将它重新命名为 SRAM。
5. 右键单击 Data [15:8]，然后取消选择 Use this bus (使用该总线) 项。只使用数据总线的低 8 位。
6. 右键单击 **ADR** 信号线。这时会出现 Config ADR Lines (配置 ADR 线) 对话框。GPIF 的 9 个地址线必须全部使用，保持它们为选中状态。点击 **OK**。



7. 右键单击 **RDY** 信号线。这时会出现 Config RDY Lines (配置 RDY 线) 对话框。SRAM 没有任何状态指示信号。因此，不需要定义任何 RDY 线。取消选择“External Inputs” (外部输入) 下的各个复选框。请勾选以下复选框：

- Sync RDY to IFCLK
- Subst TC for RDY5

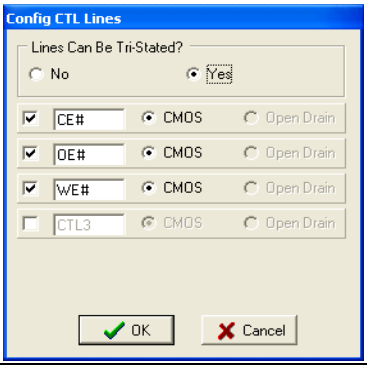
窗口显示如下。点击 **OK**。



8. 右键单击 **CTL** 信号线。这时将显示 Config CTL Lines (配置 CTL 线) 窗口。在该窗口中，

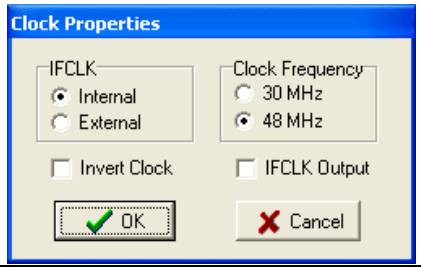
- 选择‘Lines can be tristated?’ (信号线能否被三态化?) 下方的 **Yes**。当 CTL 线配置为三态时，CTL [5：4]不可使用。
- 重新将 CTL [0-2]各行分别命名为 **CE#**、**OE#**和 **WE#**。
- 取消选择“unused” (未使用) 标签 (CTL3)。

窗口显示如下。点击 **OK**。

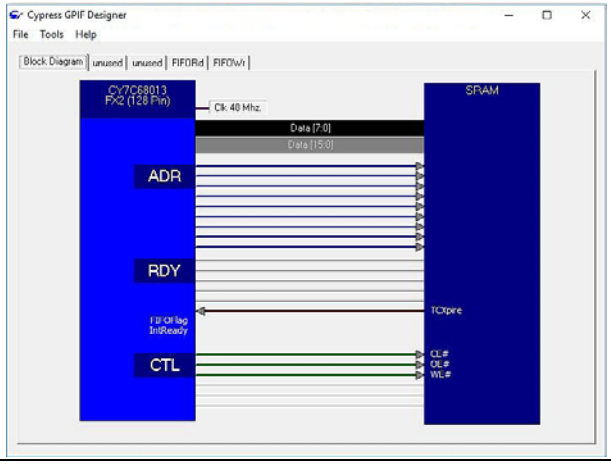


设计 GPIF 接口

9. 右键单击 **48 MHz CLK**。取消选择 **IFCLK Output** 项。将出现 ‘Clock Properties’ (时钟属性) 窗口。这是一个异步接口，并且 GPIF 使用了 48 MHz 的内部时钟。



这些步骤定义并配置了 SRAM 的 GPIF 接口。下图显示了界面在配置后的外观。



下一步骤为设计读和写波形。

4 GPIF 波形

配置接口后，创建该接口上进行通信的读和写波形。

4.1 写波形

通过写波形，将端点 FIFO 的数据写入到 SRAM 中。它们必须满足对 SRAM 写周期涉及的各种信号的时序要求。

1. 在 GPIF Designer 窗口中，请单击 **Single Read** (单次读取) 选项卡来选择它。右键单击并选择 **Set Tab Label** (设置选项卡标签)，然后它重新命名为“Unused” (未使用)。对于“Single Write” (单次写入) 选项卡，请重复该操作，同样将其重名为“Unused” (未使用)。

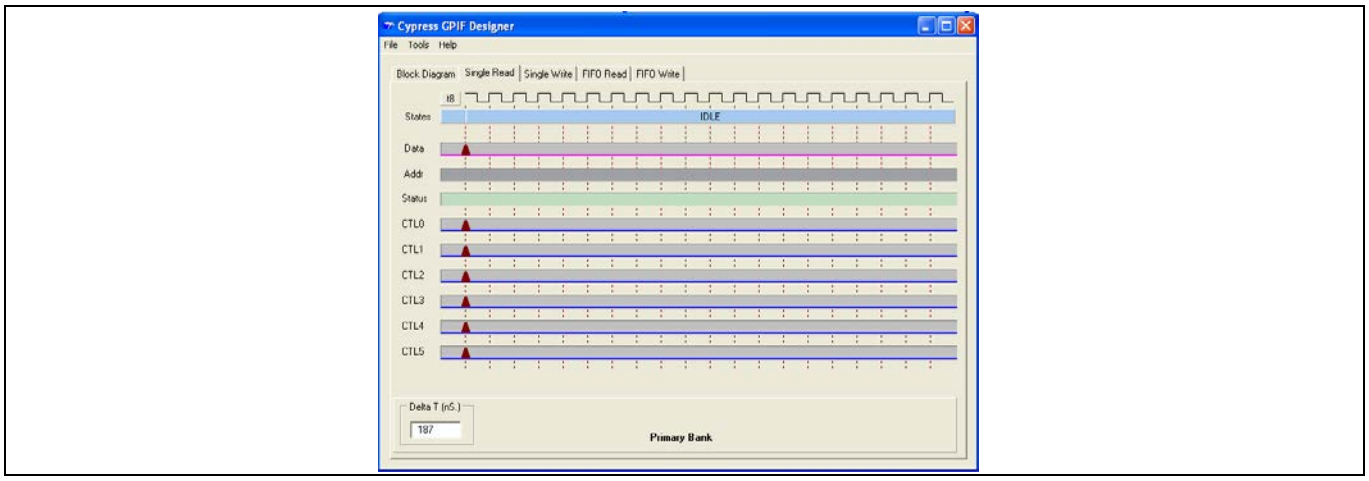


Figure 2 单次读取选项卡

2. 依次选择 Tools > Map Waveforms to WFSELECT。

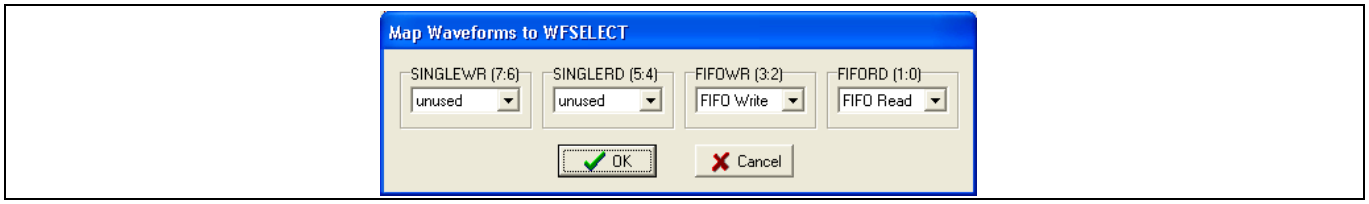


Figure 3 映射波形

请确保 FIFO 写波形被映射到 FIFOWR，而 FIFO 读波形被映射到 FIFORD。这样可以保证：启动 GPIF FIFO 写操作时可执行 FIFO 写波形，启动 GPIF FIFO 读操作时可执行 FIFO 读波形。这些位字段的映射与 GPIFWSELECT 寄存器中的位字段完全相同。各波形均被准确映射；然后点击 **OK**。

要创建 FIFO 写波形，首先要查看 SRAM 的写周期时序(**Figure 4**)及其时序参数(**Table 1**)。

GPIF 波形

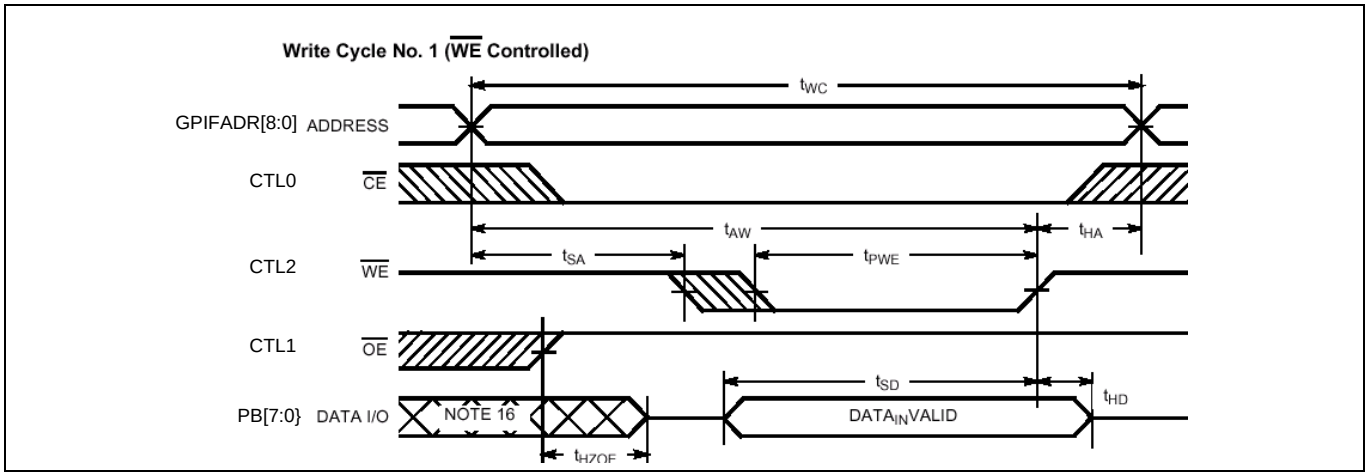


Figure 4 SRAM 的写周期时序

*Note: FX2LP GPIF 的 PB [7：0]引脚连接到 SRAM 的数据 I / O [7：0]引脚。 如 **Figure 1** 所示。*

Table 1 写周期时序参数

参数	时间 (ns)	注意
tWC (写周期时间) (最小值)	12	当 IFCLK = 48 MHz 时，每个 GPIF 周期为 20.83 ns。因此，只需要一个周期便能写入一个字节。
tPWE (WE/脉冲宽度) (最小值)	8	当 IFCLK = 48 MHz 时，每个 GPIF 周期为 20.83 ns。因此，只需要一个周期便能将 WE/置为低电平。
tSD (从数据建立到写周期结束的时间) (最小值)	7	在将 WE/置为低电平的同时驱动数据会轻松满足建立时间的要求。
tHA (写周期结束后的地址保持时间) (最小值)	0	WE/变为高电平后，无需地址值继续保持。
tSA (从地址设置到写周期开始的时间) (最小值)	0	WE/变为低电平时，不需要地址建立时间。因此可以同时设置地址和 WE/。
tAW (从地址设置到写周期结束的时间) (最小值)	8	由于地址仅被设置一个 GPIF 周期，并且 WE/在下一个周期变为高电平，因此很容易满足这种建立时间的要求。
tHD (写周期结束后数据保持时间) (最小值)	0	WE/变为高电平后无需驱动数据。

现在已经定义了各有关时序参数，因此可以使用 GPIF Designer 设计写波形。必须完成以下状态流程图：

GPIF 波形

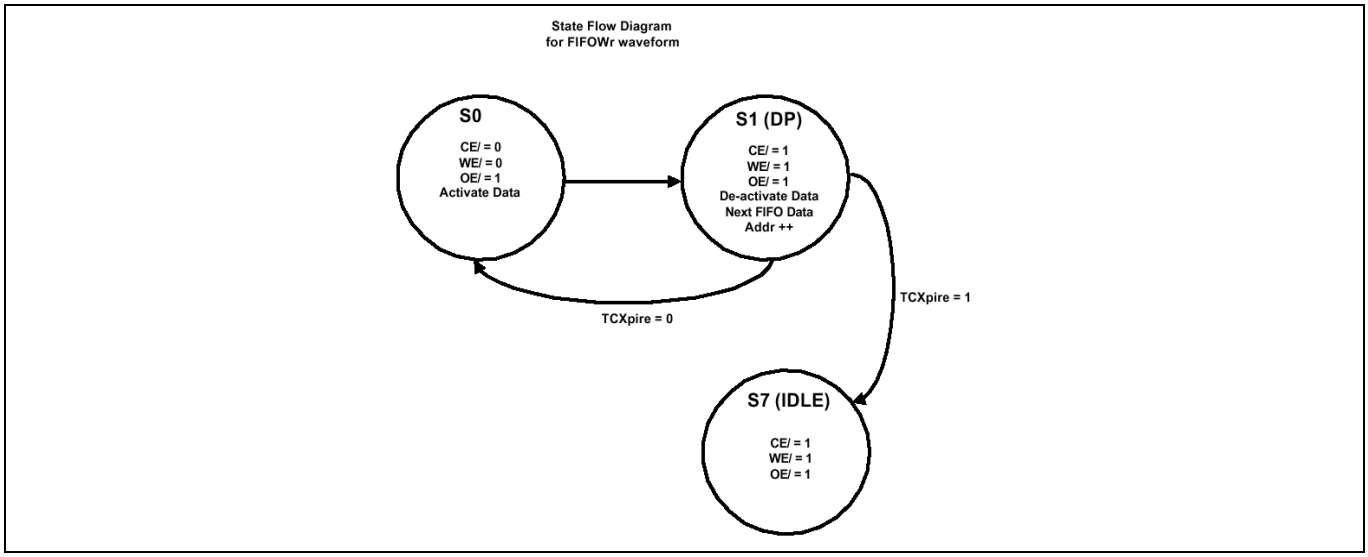
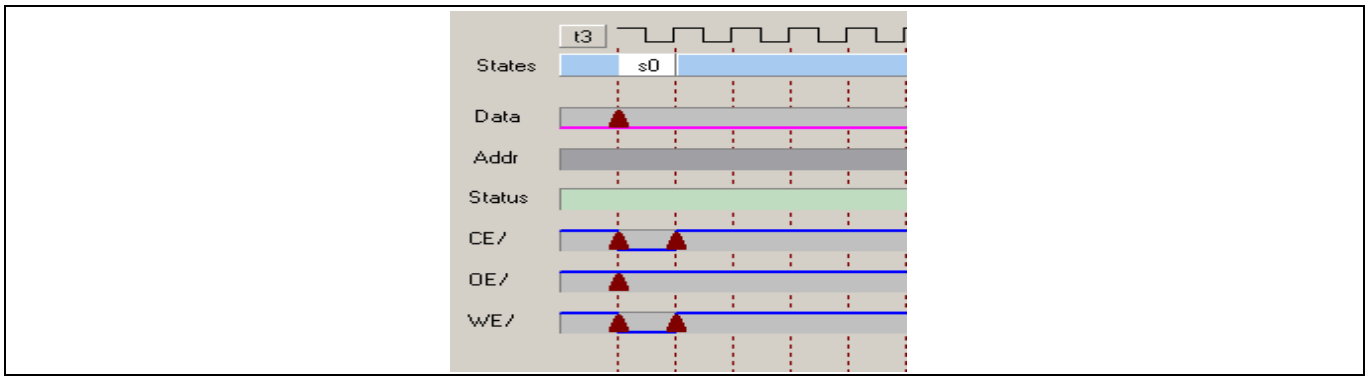


Figure 5　状态框图

*Note:　在 **Figure 5** 中，DP 代表决策点。*

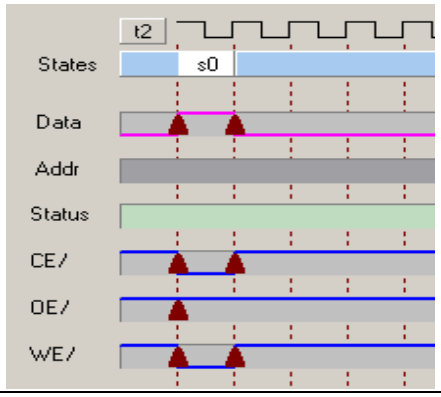
请按照下列各步骤完成 FIFO 写波形：

3. 点击 **FIFO Write** (FIFO 写) 波形选项卡。
4. 点击 **WE/**信号线，使它从左边界起持续一个时钟周期。这样便设置了一个操作点并创建 WE/波形。状态 0 (s0) 自动生成，并将持续一个 IFCLK 周期 (20.83 ns)。WE/将持续低电平 20.83 ns。这样很容易便能满足 tPWE 的要求。
5. 将 CE/和 WE/设置为一样的电平。点击 CE/信号线，使它从左边界持续低电平一个时钟周期即可实现该操作。
6. 在整个波形中，OE/必须被设置为高电平。为此，需要右键单击 **OE/**信号线上的操作点，并选择 **High (1)**。这样考虑到了 CTL 线的活动，并且波形显示如下。



7. 在 s0 状态下，数据线也需要被驱动。因此，请右击数据操作点，并选择 **Activate Data** (有效数据)。
8. 数据总线应被驱动一个时钟周期的时长。经过一个时钟周期后，为了停止驱动数据，请在数据线上放置另一个操作点。请注意，现在数据走线只在 s0 期间保持为高电平。波形显示如下。

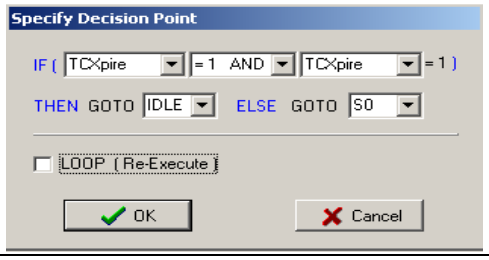
GPIF 波形



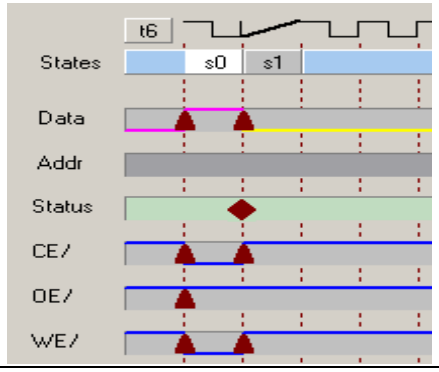
9. 下一步要添加一个策略点 (DP) 状态，以保持该波形，直到 GPIF 数据操作计数 (GPIFTC) 计满为止。要实现该操作，请在 DP 状态中测试内部 TCXpire 标志，并且只有数据操作计数计满时才使它转移到 IDLE (空闲) 状态。在 DP 状态下，GPIFADR 线也会增加。
10. 必须在 s0 状态后添加一个策略点。因此，需要通过点击 s0 的右边界来在状态线上设置一个操作点。



11. 将出现一个对话框，提示 DP 的分支条件。请按照下列对话框选择条件。

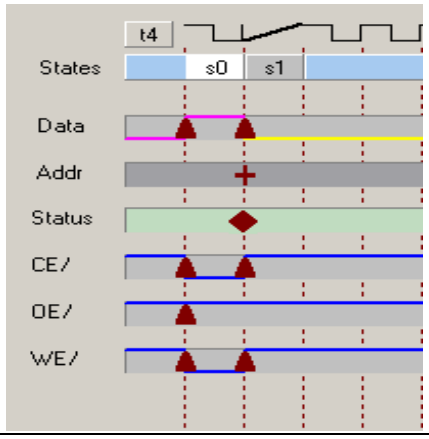


12. 这样会使 GPIF 仅看到一个信号，即 TCXpire 标志。数据操作计数计满时，GPIF 会将 TCXpire 标志设为 1。设置分支条件，转移到 IDLE (空闲) 状态并终止波形。否则，返回 S0 并继续执行波形。每次进行 “Next FIFO Data” (下个 FIFO 数据) 操作，都会递减数据操作计数。
13. 内部 FIFO 指针必须在波形中递增。在 s0 结束时，请右键单击数据线上的操作点并选择 **Next FIFO Data** (下个 FIFO 数据)。该信号线使用黄色突出显示。波形如下。



14. 然后递增 GPIFADR 线。请点击 s1 的左边界处的 **Addr** 线。最终波形如下所示：

GPIF 波形



依次选择 **File** > **Save** 来保存您的波形。

4.2 读波形

读波形用于读取 SRAM 的数据，然后写入到端点 FIFO 上。它们必须满足对 SRAM 读周期涉及的各种信号的时序要求。

该流程和设计写波形相似。首先，请查看的 SRAM 读周期时序(**Figure 6**)及其时序参数(**Table 2**)。

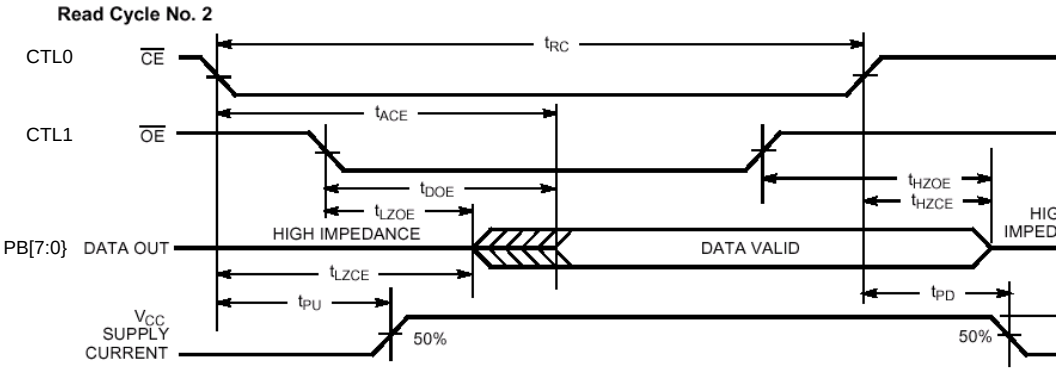


Figure 6 SRAM 的读周期时序

Note: 在整个读取周期内，WE /信号必须保持高电平。

Table 2 读周期时序参数

参数	时间 (ns)	注意
tRC — 读周期时间 (最小值)	12	当 IFCLK = 48 MHz 时，每个 GPIF 周期均为 20.83 ns。
tACE — 从 CE/为低电平到数据有效的时间 (最大值)	12	当 IFCLK = 48 MHz 时，每个 GPIF 周期均为 20.83 ns。因此，在 s0 状态必须将 CE/驱动为低电平一个周期；并在下一个周期内对数据进行采样。在整个状态过程中，数据在 IFCLK 的上升沿上被采样。
tDOE — 从 OE/为低电平到数据有效的时间 (最大值)	5	最差情况下，当 OE/被设置低电平后，数据的有效时间为 5 ns。数据在下一个 GPIF 周期内有效；在 s1 中采样数据。
tHZOE — 从 OE/为高电平到高阻态的时间 (最大值)	5	由于数据已经被采样，因此可以在 s1 中将 OE/设置为高电平。数据保持时间不用考虑。

GPIF 波形

现在已经定义了各有关时序参数，因此可以使用 GPIF Designer 设计读波形。必须完成以下状态流程图：

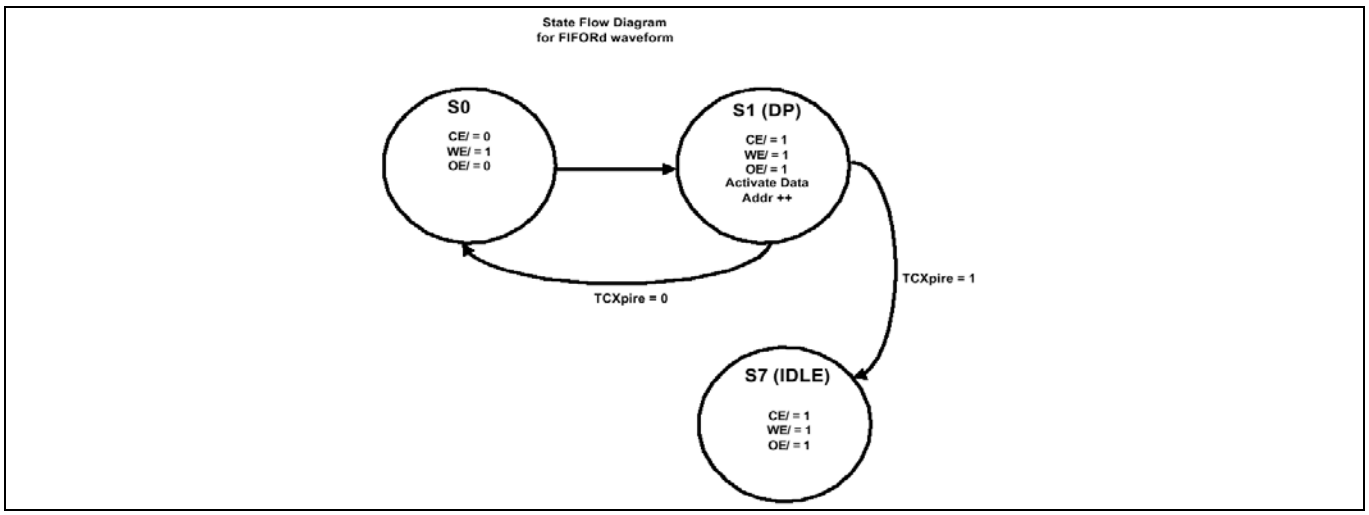
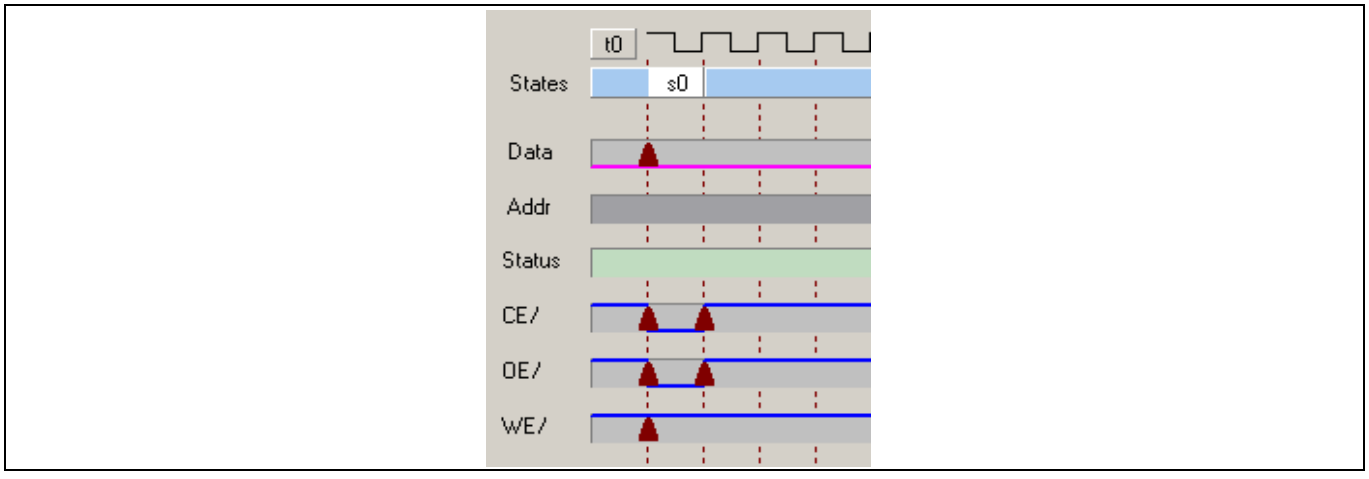


Figure 7　状态框图

请按照下列各步骤完成 FIFO 读波形：

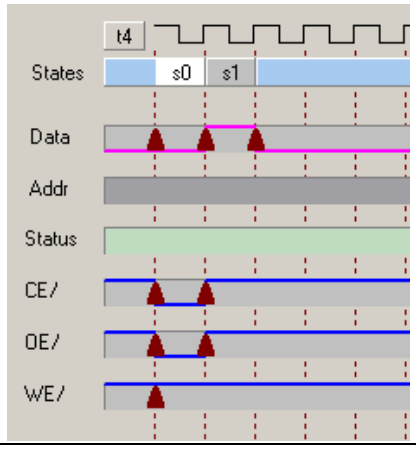
10. 点击 **FIFO Read** (FIFO 读) 选项卡。
11. 右键单击 **OE/**信号线的左边界，并选择 Low (0)。点击 **OE/**信号线使它从左边界起持续一个时钟。这样便设置一个操作点，并创建 OE/波形。自动生成状态 0 (s0)，并持续一个 IFCLK 周期 (20.83 ns)。因此，OE/将保持低电平 20.83 ns。
12. 将 CE/和 OE/设置为一样的波形。可通过右键单击 **CE/**信号线并选择 Low (0)来实现该操作。现在请点击 **CE/**信号线，使之从左边界起持续一个时钟周期。
13. 在整个波形中，WE/必须被设置为高电平。从波形上可以看到，默认情况下 WE/为高电平。
14. 在数据线上，会观察到黄色线。这是因为 GPIF Designer 已经强制全部 4 个波形均处于空闲状态。右键单击数据线上的操作点，然后点击 **Same Data**。此时，即使在写波形中也不会出现黄色线。完成读波形后，请修改它。
15. 波形显示如下：



16. 必须在 CE/为低电平后的一个时钟周期内对数据总线进行采样，这样可确保数据被采样前保持有效状态 (tACE)。请在 s0 的右边界点击 **Data** 信号线。这样会使数据信号线切换为高电平 (即设置一个“有效数据”事件)。

GPIF 波形

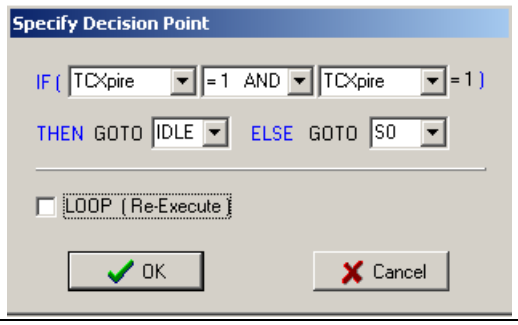
17. 只应该在一个时钟周期内对数据总线进行采样。要想在下一周期停止采样数据，请在下一周期结束处在数据信号线上放置另一个操作点。请注意，现在数据信号线只在 s1 时间内保持为高电平。波形显示如下：



18. 下一步需要添加一个策略点 (DP) 状态，以保持该波形，直到 GPIF 数据操作计数 (GPIFTC) 计满为止。要实现该操作，请在 DP 状态中测试内部 TCXpire 标志，并且只有数据操作计数计满时才将它转换到 IDLE (空闲) 状态。在 DP 状态中，递增 GPIFADR 线。
19. 必须在 s1 开始的位置设置一个 DP。因此，需要在 s1 的左边界点击状态信号线上设置一个操作点。

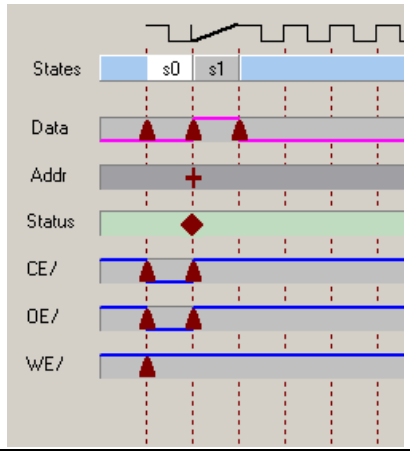


20. 将出现一个对话框，提示 DP 的分支条件。请按照下列对话框选择条件：



21. 这样会使 GPIF 仅看到一个信号，即 TCXpire 标志。数据操作计数计满时，GPIF 会将 TCXpire 标志设为 1，设置分支条件，转移到 IDLE (空闲) 状态并终止波形。否则，返回 S0 并继续执行波形。对于 FIFO 读操作，每次进行“Activate Data” (有效数据) 操作，都会递减数据操作计数。请注意，与 FIFO 写操作则完全不同，它要求使用一个“Next Data” (下个数据) 事件来递减数据操作计数。
22. 然后递增 GPIFADR 线。请在 s1 左边界处点击 **Addr** 信号线。最终波形显示如下：

GPIF 波形



23. 通过依次选择 **File** > **Save** 来保存您的波形。

这样便完成了读波形。创建读波形过程中会修改写波形。要想将它修改回来，请点击 **FIFO Write** 选项卡，并在状态 s1 结束时在数据信号线上设置一个操作点。这样会使数据信号线切换为高电平。请在 s1 结束处右键单击数据信号线上的操作点，并选择 **De-activate Data** (无效数据)。右击 s1 开始位置上的操作点，然后选择 **Next FIFO** (下个 FIFO) 数据。这样做是因为位于空闲状态的左边沿上的数据操作点不能提供任何形式的“Next Data” (下一个数据)。

这样便已经完成了设计 GPIF 波形。现在可以将该波形导出到 *gpif.c* 文件内，并将其添加到项目中。

4.3 导出 GPIF 波形

要将波形导出到 C 文件内，并将其添加到固件项目中，请进行下列操作：

1. 依次选择 **Tools** > **Export to gpif.c** File。
2. 将该文件以 *gpif.c* 的文件形式保存在临时的存储位置上。

4.4 uVision2 项目和固件

下一步是创建一个包含 GPIF 波形文件 (gpif.c) 的项目，并将固件写入接口 SRAM 到 FX2LP 器件。

Note: 您可以在赛普拉斯网页上直接使用本应用笔记提供的 uVision 项目'FX2_SRAM_GPIF'，也可以直接创建一个新项目，如下所示。

以下步骤说明如何使用 **FX2LP DVK** 提供的示例项目 (此处使用 Bulkloop 示例项目作为参考) 创建新的 uVision 项目。

1. 创建一个新文件夹，命名为 SRAM_GPIF。

   如果您已经安装了 EZ-USB FX2LP 开发工具，那么请在 `C:\Cypress\USB\Examples\FX2LP` 路径下创建一个新的文件夹，并将其命名为“SRAM_GPIF”。

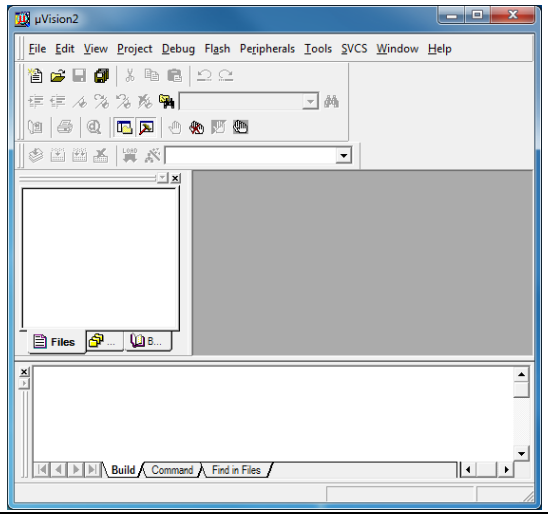
2. 将 `C:\Cypress\USB\Examples\FX2LP\bulkloop` 中以下各个项目文件复制到刚创建的文件夹内。

- dscr.a51
- fw.c
- bulkloop.c

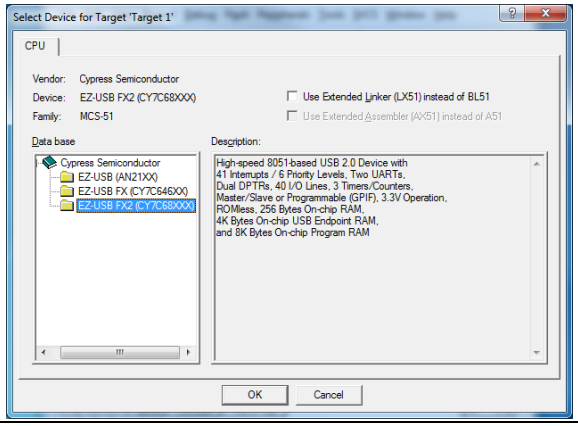
3. 将 bulkloop.c 重名为 FX2_SRAM_GPIF.c。
4. 将导出 GPIF 波形时所保存的 gpif.c 文件转移到该目录中。

GPIF 波形

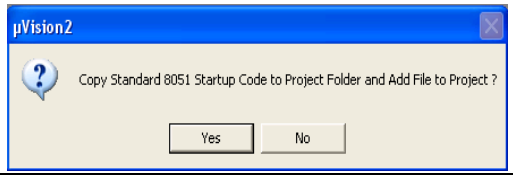
5. 启动 Microvision (uv2)。项目窗口显示如下。



6. 依次选择 **Project** > **New Project**。该工具会提示您命名项目并保存。浏览到新创建的 SRAM_GPIF 目录，并将该项目保存为 FX2_SRAM_GPIF。
7. 将出现下面的窗口，提示选择器件类型。请从“Cypress Semiconductor” (赛普拉斯半导体) 下方的名单中选择 **EZ-USB FX2 (CY7C68013)**，然后点击 **OK**。

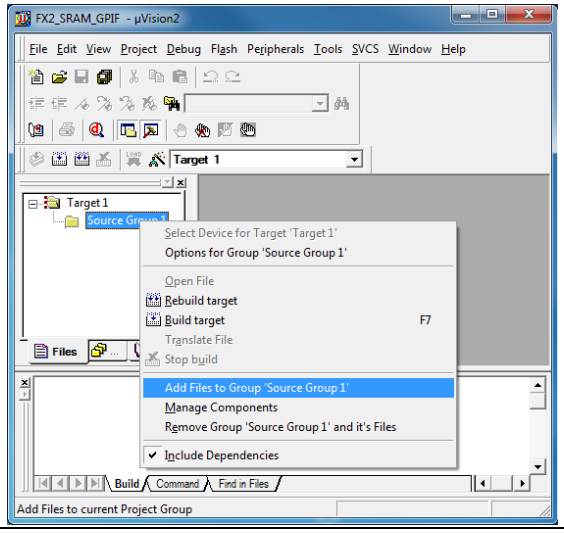


8. 提示您将出现以下对话框。请点击 **No**。



9. 这时会出现以下窗口：将所有相关的源文件添加到 FX2_SRAM_GPIF 项目中。要添加这些文件，请右击 **Source Group1** 目录，然后选择 **Add Files to “Source Group1”**。

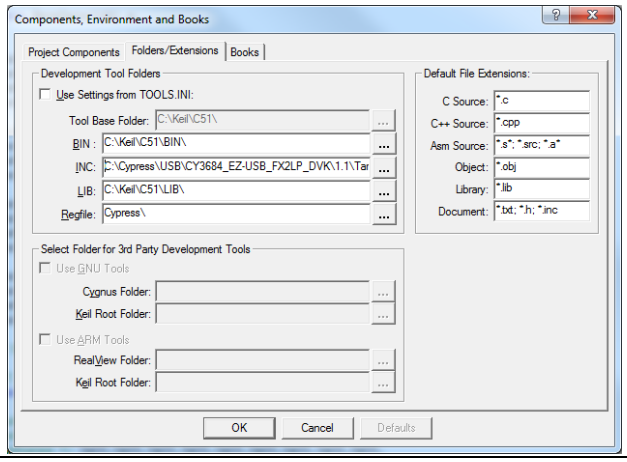
GPIF 波形



10. 下列路径中的各个文件需要添加。

- C:\Cypress\USB\Examples\FX2LP\ SRAM_GPIF
  - gpif.c
  - fw.c
  - FX2_SRAM_GPIF.c
  - dscr.a51
- C:\Cypress\USB\Target\Lib\LP
  - USBJmpTb.OBJ
  - Ezusb.lib

11. 要想设置 Microvision 的设置环境，请依次选择 **Project** > **File Extensions, Books and Environment**。

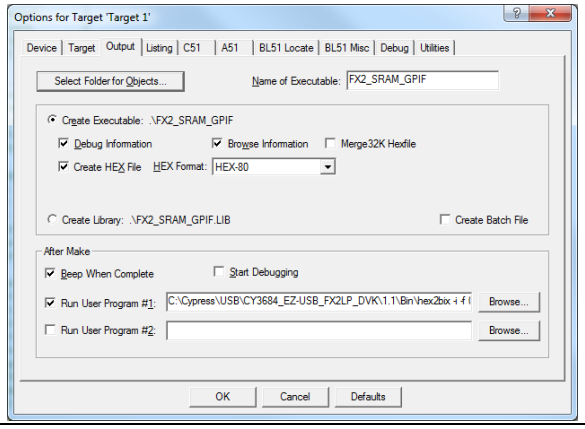


12. 在 Environment Setup (环境设置) 选项卡中，需要编辑如下字段：

- BIN Folder: C:\Keil\C51\BIN\
- INC Folder: C:\CYPRESS\USB\Target\Inc\;C:\Keil\C51\INC\
- LIB Folder: C:\Keil\C51\LIB\

13. 依次选择 **Project** > **Options for Target 'Target1'**。选择 Output (输出) 选项卡，并勾选 **Create Hex File** 框。

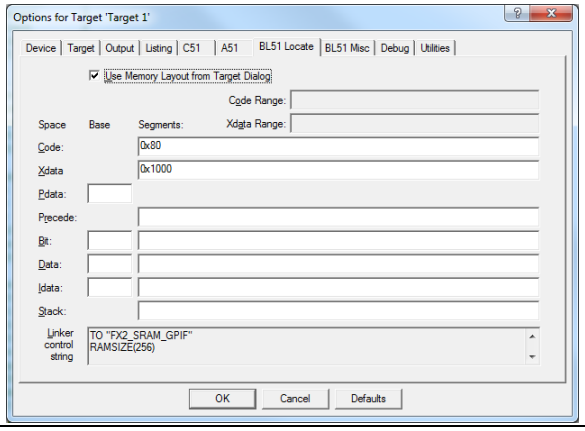
GPIF 波形



“Run User Program #1”字段应填写以下行：

C:\Cypress\USB\CY3684_EZ-USB_FX2LP_DVK\1.1\Bin\hex2bix -i -f 0xC2 -o FX2_SRAM_GPIF.iic FX2_SRAM_GPIF.hex

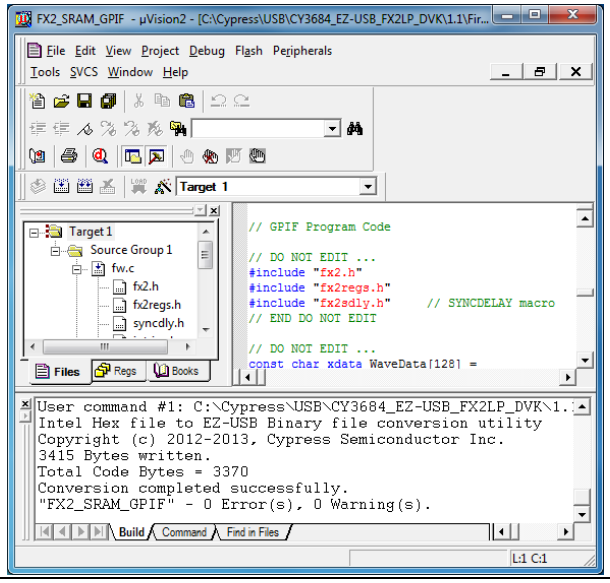
14. 点击 **BL51 Locate** 选项卡，然后设置代码和 xdata 区，如下所示。



FX2LP 的中断向量位于 0x00 至 0x7C 之间。 因此，建议将代码位置设置为 0x80，将 Xdata 位置设置为 0x1000。

15. 通过点击 **Rebuild all target files** (重新构建所有目标文件) 按键  来构建项目。请确保所编译的代码准确无误。警告信息请点击 **OK**。
16. 如果成功完成构建，在输出窗口内会显示以下的内容。固件已经构建好，您需要添加 GPIF 传输操作部分的固件。

GPIF 波形



17. 现在，以下固件部分已添加到 FX2_SRAM_GPIF.c 文件中，以创建用于将 SRAM 连接到 FX2LP 设备的最终固件：

- TD_Init()子程序中的初始化代码
- TD_Poll()中的程序打开/关闭 LED 从而表示固件正在运行。
- 供应商特定指令会进行下列操作：
  - 触发 GPIF FIFO 写操作，以将数据写入到 SRAM 内 (处理单次写入和块写入操作)
  - 触发 GPIF FIFO 读操作，从而读取 SRAM 中的数据 (处理单次读取和块读取操作)

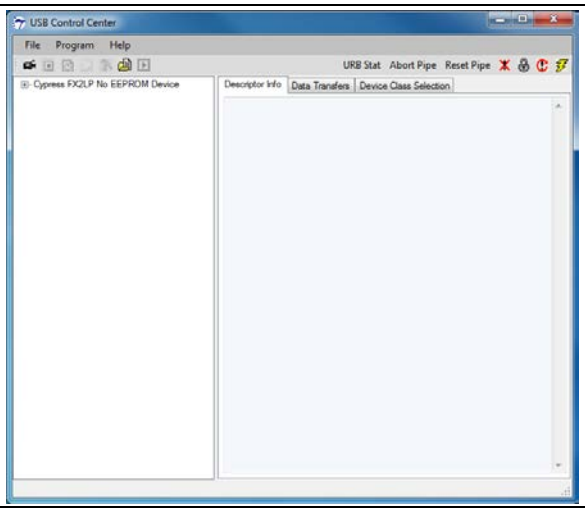
附录 A 中提供了这些操作的代码。将代码复制到 *FX2_SRAM_GPIF.c* 文件中相应位置。例如，使用本文档中的 TD_poll()代替该现有函数。同样复制所有宏定义和有关 LED 函数。

18. 重新编译项目。如果您正在使用 4K Keil 评估工具，可能会担心 4K 的代码大小限制。在这种情况下，注释掉供应商指令 (即 bulkloop 示例的剩余部分)，然后重新编译项目。

5 使用 Control Center 进行测试

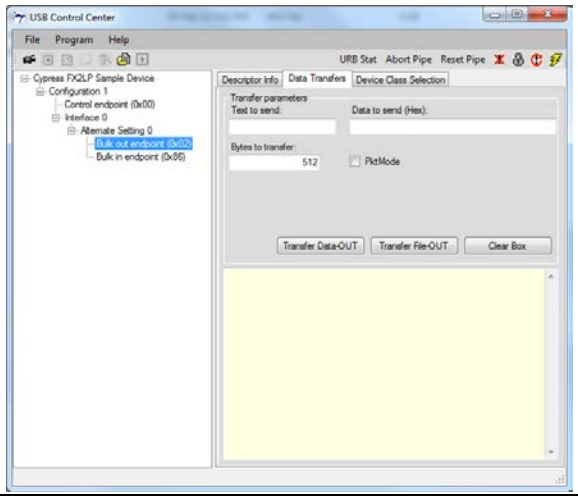
通过使用 CY3684 开发板和 Control Center 主机应用，可以验证所构建项目的功能。

1. 将开发板上的 EEPROM 使能开关设为 **OFF** (关闭)，并将该板 USB 接口插入到主机上
2. 它会枚举为“Cypress EZ-USB FX2LP – EEPROM Missing”，并且能在器件管理器中看到它。
3. 下载并安装 **Cypress Suite USB 3.4**。 这将安装 Control Center 实用程序，如下所示：



4. 在左窗格中选择设备，然后单击 **Program> FX2> RAM** 并导航到新创建的 FX2_to_SRAM.hex 文件。 等待固件重新枚举并重新连接为 Cypress FX2LP 样本设备。
5. 该器件重新枚举成功时，通过供应商的 IN 指令 0xBB 和 0xBC，可以将它连接到 SRAM。

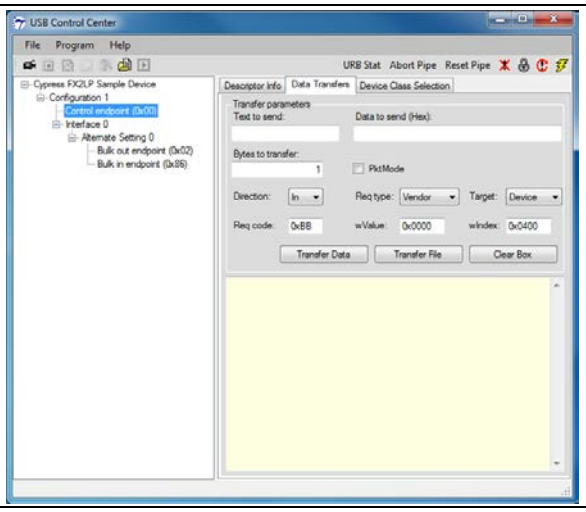
将数据传输给 EP2OUT 端点。可以通过以下两种方法实现该操作：使用批量传输栏指定端点，从而选择数据值和请求长度；或者使用 **Transfer File-OUT** (文件传输) 按键来传输已知的数据文件。请确保在下拉框中选中 **Bulk out endpoint (0x02)**。单击右窗格中的 **Data Transfers** 选项卡，然后单击 **Transfer File-OUT** 按钮并选择位于 `C:\Cypress\USB\CY3684_EZ-USB_FX2LP_DVK\1.1\Target\File_Transfer` 路径中的 ***1024_count.hex***。



6. 在左窗格中选择 **Control endpoint (0x00)**，然后单击右窗格中的 **Data Transfers** 选项卡。 选择 **Req type** 作为 **Vendor**。 **Req code** 代码字段表示请求的类型。 在此字段中输入 0xBB 或 0xBC，具体取决于您是要分别启动写入还是读取。。**wValue** 字段指定 SRAM 地址。 **wIndex** 字段指定 HEX 中的传输长

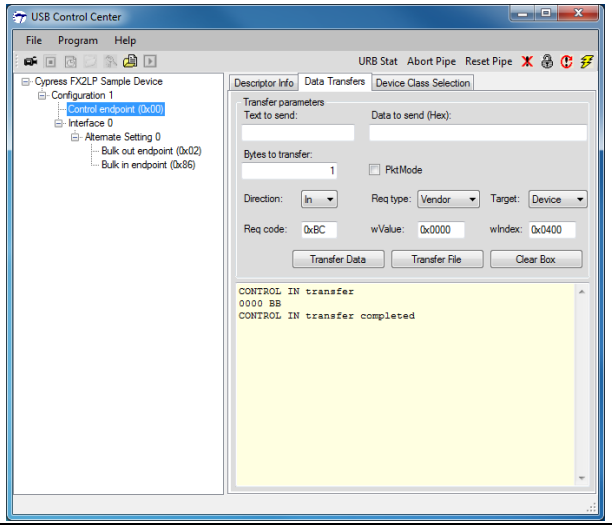
使用 Control Center 进行测试

度。 长度字段必须是步骤 7 中指定的传输长度的等效 HEX 值。例如，下图显示了如何在 Vend Req 传输栏的 SRAM 中从地址 0x0000 开始写入 1K (0x0400) 字节。



点击 **Transfer Data** 按键，以触发对 SRAM 进行的写操作。请求得到处理后，输出窗口中会显示供应商请求 (0xBB) 的值。

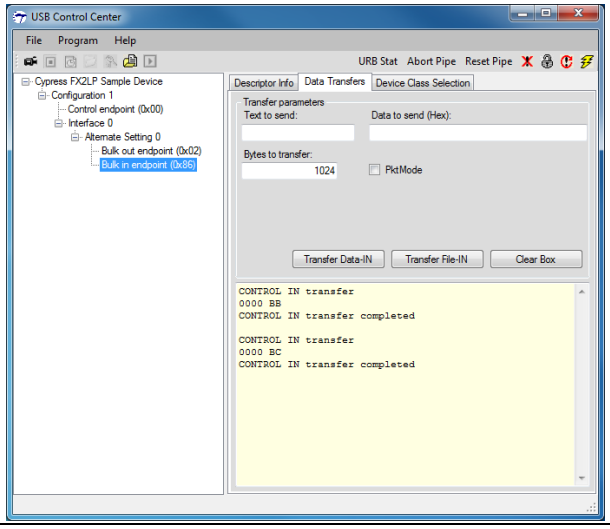
7. 要想读取 SRAM 中的 1K (即 0x0400) 字节 (从地址 0x0000 开始)，需要在 Vend Req 传输栏中指定下面各值。



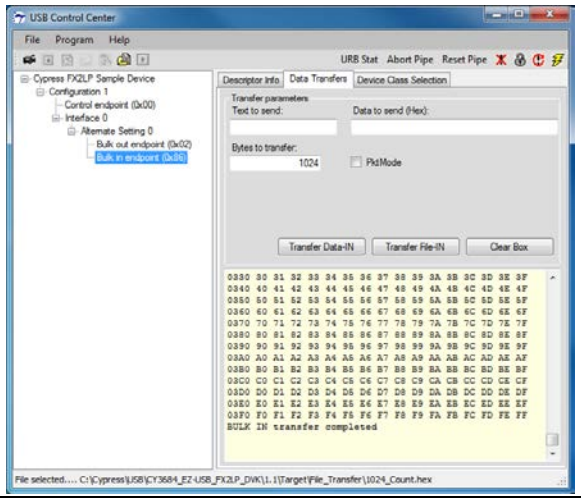
请求得到处理后，输出窗口中会显示供应商请求 (0xBC) 的值。

8. 要想通过 **Bulk in endpoint (0x86)** 端点将从 SRAM 中读取的数据传送给主机，在左窗格中选择端点，然后使用 **Data Transfers** 选项卡指定请求长度，并单击“**Transfer Data-IN**。

使用 Control Center 进行测试



9. 批量传输结束后，主机接收到的数据会如下显示在控制面板中。



10. 现在，您可以对所有 SRAM 地址 (从 0x0000 到 0x1FFF，即 8K 字节范围) 和传输长度 (从 0x0001 到 0x2000，即 1 到 8192 字节) 重复执行该操作。

6 总结

本应用笔记介绍了一个使用 GPIF Designer 来连接和创建波形的简便流程。然后介绍如何使用这些波形文件创建固件项目，以将 SRAM 连接到 FX2LP 开发套件板。

7 参考

有关 GPIF 的更详细的信息，请参考我们网站中提供的下列文档：

- **FX2LP GPIF 主-从设备 FIFO 背靠背连接设置**
- **GPIF 和从设备 FIFO 配置中的 EZ-USB FX2LP™手动操作模式**
- **EZ-USB FX2 GPIF 入门**
- **AN63787 - 使用 8 位异步接口的 EZ-USB®FX2LP™GPIF 和从设备 FIFO 配置示例**

有关 GPIF II 的信息，请参阅以下应用文件：

- **AN87216 - 设计 GPIF II 主接口**
- **AN65974 - 使用 EZ-USB®FX3™从设备 FIFO 接口进行设计**
- **AN68829 - EZ-USB®FX3™的从设备 FIFO 接口：5 位地址模式**
- **AN75779 - 如何在 USB 视频类 (UVC) 框架中使用 EZ-USB®FX3™实现图像传感器接口**

8 附录 A

Code Listing 1

```
/* GPIF write */
#define VX_BB 0xBB
/* GPIF read */
#define VX_BC 0xBC

#define GPIFTRIGRD 4
#define GPIF_EP2 0
#define GPIF_EP4 1
#define GPIF_EP6 2
#define GPIF_EP8 3

/* flag to let firmware know FX2 enumerated at high speed */
BOOL enum_high_speed = FALSE;
/* variable that contains EP6FIFOBCH/L value */
static WORD xFIFOBC_IN = 0x0000;
static WORD xdata LED_Count = 0;
static BYTE xdata LED_Status = 0;

WORD addr, len, Tcount;
/* ...debug LEDs: accessed via movx reads only ( through CPLD ) */
/* it may be worth noting here that the default monitor loads at 0xC000 */
xdata volatile const BYTE LED0_ON  _at_ 0x8000;
xdata volatile const BYTE LED0_OFF _at_ 0x8100;
xdata volatile const BYTE LED1_ON  _at_ 0x9000;
xdata volatile const BYTE LED1_OFF _at_ 0x9100;
xdata volatile const BYTE LED2_ON  _at_ 0xA000;
xdata volatile const BYTE LED2_OFF _at_ 0xA100;
xdata volatile const BYTE LED3_ON  _at_ 0xB000;
xdata volatile const BYTE LED3_OFF _at_ 0xB100;
/* use this global variable when (de)asserting debug LEDs...*/

BYTE xdata ledX_rdvar = 0x00;
BYTE xdata LED_State = 0;

void LED_Off (BYTE LED_Mask);
void LED_On (BYTE LED_Mask);
void GpifInit ();


void LED_Off (BYTE LED_Mask)
{
      if (LED_Mask & bmBIT0)
      {
            ledX_rdvar = LED0_OFF;
            LED_State &= ~bmBIT0;
      }
      if (LED_Mask & bmBIT1)
      {
            ledX_rdvar = LED1_OFF;
            LED_State &= ~bmBIT1;
      }
      if (LED_Mask & bmBIT2)
      {
            ledX_rdvar = LED2_OFF;
            LED_State &= ~bmBIT2;
      }
      if (LED_Mask & bmBIT3)
      {
            ledX_rdvar = LED3_OFF;
            LED_State &= ~bmBIT3;
```

附录 A

Code Listing 1

```
      }
}

void LED_On (BYTE LED_Mask)
{
      if (LED_Mask & bmBIT0)
      {
            ledX_rdvar = LED0_ON;
            LED_State |= bmBIT0;
      }
      if (LED_Mask & bmBIT1)
      {
            ledX_rdvar = LED1_ON;
            LED_State |= bmBIT1;
      }
      if (LED_Mask & bmBIT2)
      {
            ledX_rdvar = LED2_ON;
            LED_State |= bmBIT2;
      }
      if (LED_Mask & bmBIT3)
      {
            ledX_rdvar = LED3_ON;
            LED_State |= bmBIT3;
      }
}



//-----------------------------------------------------------------------------
// Task Dispatcher hooks
//   The following hooks are called by the task dispatcher.
//-----------------------------------------------------------------------------

/* Called once at startup */
void TD_Init(void)
{
      /* set the CPU clock to 48MHz*/
      CPUCS = ((CPUCS & ~bmCLKSPD) | bmCLKSPD1) ;
      SYNCDELAY;
      /* set the slave FIFO interface to 48MHz*/
      IFCONFIG |= 0x40;

      /*change EP configuration */
      EP2CFG = 0xA0;
      SYNCDELAY;
      EP4CFG = 0x00;
      SYNCDELAY;
      EP6CFG = 0xE0;
      SYNCDELAY;
      EP8CFG = 0x00;

      /* out endpoints do not come up armed */
      /* set NAKALL bit to NAK all transfers from host */
      FIFORESET = 0x80;
      SYNCDELAY;
      /* reset EP2 FIFO */
      FIFORESET = 0x02;
      SYNCDELAY;
      /* reset EP6 FIFO */
      FIFORESET = 0x06;
      SYNCDELAY;
      /* clear NAKALL bit to resume normal operation */
```

附录 A

Code Listing 1

```
      FIFORESET = 0x00;
      SYNCDELAY;
      /* allow core to see zero to one transition of auto out bit*/
      EP2FIFOCFG = 0x00;
      SYNCDELAY;
      /* auto out mode, disable PKTEND zero length send, byte ops*/
      EP2FIFOCFG = 0x10;
      SYNCDELAY;
      /* auto in mode, disable PKTEND zero length send, byte ops */
      EP6FIFOCFG = 0x08;
      SYNCDELAY;

      /* nable dual autopointer feature */
      AUTOPTRSETUP |= 0x01;

      /* initialize GPIF registers */
      GpifInit ();
      PORTACFG = 0x00;
      OEA |= 0xF3;
      IOA &= 0xFC;

      /* Enable remote-wakeup */
      Rwuen = TRUE;
}

/* Called repeatedly while the device is idle */
void TD_Poll(void)
{
  /* blink LED0 to indicate firmware is running */
  if (++LED_Count == 10000)
  {
    if (LED_Status)
    {
      LED_Off (bmBIT0);
      LED_Status = 0;
    }
    else
    {
      LED_On (bmBIT0);
      LED_Status = 1;
    }
    LED_Count = 0;
  }

}

/* Called before the device goes into suspend mode */
BOOL TD_Suspend(void)
{
  return(TRUE);
}

/* Called after the device resumes */
BOOL TD_Resume(void)
{
  return(TRUE);
}

//-----------------------------------------------------
// Device Request hooks
//   The following hooks are called by the end point 0
//device request parser.
//-----------------------------------------------------
```

附录 A

Code Listing 1

```
BOOL DR_GetDescriptor(void)
{
  return(TRUE);
}

/* Called when a Set Configuration command is received */
BOOL DR_SetConfiguration(void)
{
   if( EZUSB_HIGHSPEED( ) )
  {
            /* FX2 enumerated at high speed */
    SYNCDELAY;
            /* set AUTOIN commit length to 512 bytes */
    EP6AUTOINLENH = 0x02;
    SYNCDELAY;
    EP6AUTOINLENL = 0x00;
    SYNCDELAY;
    enum_high_speed = TRUE;
  }

  else
  {
      /* FX2 enumerated at full speed */
    SYNCDELAY;
      /* set AUTOIN commit length to 64 bytes */
    EP6AUTOINLENH = 0x00;
    SYNCDELAY;
    EP6AUTOINLENL = 0x40;
    SYNCDELAY;
    enum_high_speed = FALSE;
  }


   Configuration = SETUPDAT[2];
   return(TRUE);
}

/* Called when a Get Configuration command is received */
BOOL DR_GetConfiguration(void)
{
   EP0BUF[0] = Configuration;
   EP0BCH = 0;
   EP0BCL = 1;
   return(TRUE);
}

/* Called when a Set Interface command is received */
BOOL DR_SetInterface(void)
{
   AlternateSetting = SETUPDAT[2];
   return(TRUE);
}

/* Called when a Set Interface command is received */
BOOL DR_GetInterface(void)
{
   EP0BUF[0] = AlternateSetting;
   EP0BCH = 0;
   EP0BCL = 1;
   return(TRUE);
}
```

附录 A

Code Listing 1

```
BOOL DR_GetStatus(void)
{
   return(TRUE);
}

BOOL DR_ClearFeature(void)
{
   return(TRUE);
}

BOOL DR_SetFeature(void)
{
   return(TRUE);
}

BOOL DR_VendorCmnd(void)
{
  BYTE tmp;

  switch (SETUPDAT[1])
  {
            case VR_NAKALL_ON:
                  tmp = FIFORESET;
                  tmp |= bmNAKALL;
                  SYNCDELAY;
                  FIFORESET = tmp;
                  break;
            case VR_NAKALL_OFF:
                  tmp = FIFORESET;
                  tmp &= ~bmNAKALL;
                  SYNCDELAY;
                  FIFORESET = tmp;
                  break;
            /* actuate write to SRAM */
            case VX_BB:
            {
                  EP0BUF[0] = VX_BB;
                  /* select bank of 16x512 (bit shift MSB of wValue by 3*/
                  /* and OR it with PA[7:4] */
                  IOA = (IOA & 0x0F) + (SETUPDAT[3] << 3);
                  /* set GPIFADR[8:0] to address passed down in wValue */
                  GPIFADRH = SETUPDAT[3];
                  GPIFADRL = SETUPDAT[2];

                  /* get transfer length from wIndex field */
                  len = ( (SETUPDAT[5] << 8) + SETUPDAT[4] );
                  /* while the transfer length is non-zero */
                  while (len)
                  {
                        /* if GPIF interface IDLE */
                        if( GPIFTRIG & 0x80 )
                        {
                              /* if there's a packet in the peripheral domain for EP2 */
                              if ( ! ( EP24FIFOFLGS & 0x02 ) )
                              {
                                    /* if the FX2 enumerated at high-speed */
                                    if(enum_high_speed)
                                    {
                                          /* if the transfer length is greater than 512 bytes */
                                          if ( len > 0x0200 )
                                          {
                                                /* set GPIF transaction count to 512, since*/
                                                /* GPIFADR can only access 512 locations at a time */
```

附录 A

Code Listing 1

```
                                        GPIFTCB1 = 0x02;
                                        SYNCDELAY;
                                        GPIFTCB0 = 0x00;
                                        SYNCDELAY;
                                        Tcount = 0x0200;
                                  }
                                  else
                                  {
                                        /* else set GPIF transaction count to EP2 */
                                        GPIFTCB1 = EP2FIFOBCH;
                                        SYNCDELAY;
                                        GPIFTCB0 = EP2FIFOBCL;
                                        SYNCDELAY;
                                        Tcount = len;
                                  }
                            }
                            /* if the FX2 enumerated at full-speed */
                            else
                            {
                                  /* if the transfer length is greater than 64 bytes, */
                                  if ( len > 0x040 )
                                  {
                                        /* set GPIF transaction count to 64 */
                                        GPIFTCB1 = 0x00;
                                        SYNCDELAY;
                                        GPIFTCB0 = 0x40;
                                        SYNCDELAY;
                                        Tcount = 0x0040;
                                  }
                                  else
                                  {
                                        GPIFTCB1 = EP2FIFOBCH;
                                        SYNCDELAY;
                                        GPIFTCB0 = EP2FIFOBCL;
                                        SYNCDELAY;
                                        Tcount = len;
                                  }
                            }
                            /* launch GPIF FIFO WRITE Transaction from EP2 FIFO*/
                            GPIFTRIG = GPIF_EP2;
                            SYNCDELAY;
                            /* poll GPIFTRIG.7 GPIF Done bit */
                            while( !( GPIFTRIG & 0x80 ) )
                            {
                            ;
                            }
                            SYNCDELAY;
                            /* decrement transfer length by Tcount*/
                            len = len - Tcount;
                            /* if the transfer length is not a modulus of 512, no need to */
                            /* reset GPIFADR[8:0] to access next bank of 512 bytes,*/
                            /* handles full-speed case and high-speed case */
                            /* reset GPIFADR[8:0] to access the next bank at offset 0*/
                            if (!(len % 0x0200))
                            {
                                GPIFADRH = 0x00;
                                GPIFADRL = 0x00;
                                /* increment the bank address by 1 to access next bank of 512*/
                                  IOA = ( ( ( IOA >> 4 ) + 1 ) << 4 );
                            }
                      }
                }
          }
```

附录 A

Code Listing 1

```
        EP0BCH = 0;
        EP0BCL = 1;
        EP0CS |= bmHSNAK;
        break;
    }
    /* actuate read from SRAM */
    case VX_BC:
        {
        EP0BUF[0] = VX_BC;

        /* select bank of 16x512 (bit shift MSB of wValue by 3
           and OR it with PA[7:4]*/
        IOA = (IOA & 0x0F) + (SETUPDAT[3] << 3);
        /* set GPIFADR[8:0] to address passed down in wValue */
        GPIFADRH = SETUPDAT[3];
        GPIFADRL = SETUPDAT[2];
        /* get transfer length from wIndex field */
        len = ( (SETUPDAT[5] << 8) + SETUPDAT[4] );
        /* while the transfer length is non-zero */
        while (len)
        {
            /* if GPIF interface IDLE */
            if( GPIFTRIG & 0x80 )
            {
                /* if EP6 FIFO is not full */
                if( !( EP68FIFOFLGS & 0x01 ) )
                {
                    /* if the FX2 enumerated at high-speed */
                    if(enum_high_speed)
                    {
                        /* if the transfer length is greater than 512 bytes, */
                        if ( len > 0x0200 )
                        {
                            /*set GPIF transaction count to 512, since
                             GPIFADR can only access 512*/
                            GPIFTCB1 = 0x02;
                            SYNCDELAY;
                            GPIFTCB0 = 0x00;
                            SYNCDELAY;
                            Tcount = 0x0200;
                        }
                        else
                        {
                            GPIFTCB1 = MSB(len);
                            SYNCDELAY;
                            GPIFTCB0 = LSB(len);
                            SYNCDELAY;
                            Tcount = len;
                        }
                    }
                    /* if the FX2 enumerated at full-speed */
                    else
                    {
                        /* if the transfer length is greater than 64 bytes*/
                        if ( len > 0x0040 )
                        {
                            GPIFTCB1 = 0x00;
                            SYNCDELAY;
                            GPIFTCB0 = 0x40;
                            SYNCDELAY;
                            Tcount = 0x0040;
                        }
```

附录 A

Code Listing 1

```
                                        else
                                        {
                                              GPIFTCB1 = MSB(len);
                                              SYNCDELAY;
                                              GPIFTCB0 = LSB(len);
                                              SYNCDELAY;
                                              Tcount = len;
                                        }
                                  }

                                  /* launch GPIF FIFO READ Transaction to EP6IN */
                                  GPIFTRIG = GPIFTRIGRD | GPIF_EP6;
                                  SYNCDELAY;
                                  /* poll GPIFTRIG.7 GPIF Done bit */
                                  while( !( GPIFTRIG & 0x80 ) )
                                  {
                                  ;
                                  }
                                  SYNCDELAY;
                                  /* get EP6FIFOBCH/L value */
                                  xFIFOBC_IN = ( ( EP6FIFOBCH << 8 ) + EP6FIFOBCL );
                                  /* if pkt is short*/
                                  if( ( xFIFOBC_IN > 0) && ( xFIFOBC_IN < 0x0200 ) )
                                  {
                                        INPKTEND = 0x06;
                                  }
                                  /* decrement transfer length by Tcount */
                                  len = len - Tcount;
                                  /* if the transfer length is not a modulus of 512, no need to
                                        reset GPIFADR[8:0] to access next bank of 512 bytes, */
                                  if(!(len % 0x0200))
                                  {
                                        /* handles full-speed case and high-speed case */
                                        /* reset GPIFADR[8:0] to access the next bank at offset 0 */
                                        GPIFADRH = 0x00;
                                        GPIFADRL = 0x00;
                                        /* increment the bank address by 1 to access */
                                        /* next bank of 512*/
                                        IOA = ( ( ( IOA >> 4 ) + 1 ) << 4 );

                                  }
                            }
                      }
                }

                EP0BCH = 0;
                EP0BCL = 1;
                EP0CS |= bmHSNAK;
                break;
          }
          default:
          return(TRUE);
     }
     return(FALSE);
}
```

文档修订记录

Document version	Date of release	Description of changes
**	2015-07-08	本文档版本号为 Rev**，译自英文版 001-57322 Rev*C。
*A	2018-12-26	本文档版本号为 Rev*A，译自英文版 001-57322 Rev*F。
*B	2022-02-28	更新至英飞凌模板

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

Edition 2022-02-28
Published by
Infineon Technologies AG
81726 Munich, Germany

© 2022 Infineon Technologies AG.
All Rights Reserved.

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

Document reference
001-98023 Rev. *B

重要提示

本文档所提供的任何信息**绝不应当**被视为针对任何条件或者品质而做出的保证（质量保证）。英飞凌对于本文档中所提及的任何事例、提示或者任何特定数值及/或任何关于产品应用方面的信息均在此明确声明其不承担任何保证或者责任，包括但不限于其不侵犯任何第三方知识产权的保证均在此排除。

此外，本文档所提供的任何信息均取决于客户履行本文档所载明的义务和客户遵守适用于客户产品以及与客户对于英飞凌产品的应用所相关的任何法律要求、规范和标准。
本文档所含的数据仅供经过专业技术培训的人员使用。客户自身的技术部门有义务对于产品是否适宜于其预期的应用和针对该等应用而言本文档中所提供的信息是否充分自行予以评估。

如需产品、技术、交付条款和条件以及价格等进一步信息，请向离您最近的英飞凌科技办公室接洽(**www.infineon.com**)。

警告事项

由于技术所需产品可能含有危险物质。如需了解该等物质的类型，请向离您最近的英飞凌科技办公室接洽。

除非由经英飞凌科技授权代表签署的书面文件中做出另行明确批准的情况外，英飞凌科技的产品不应当被用于任何一项一旦产品失效或者产品使用的后果可被合理地预料到可能导致人身伤害的任何应用领域。