EZ-USB™ FX2LP GPIF 入门

关于本文档

范围和目的

FX2LP 通用可编程接口 (GPIF) 提供一个独立硬件单元，用于创建数据和控制外部接口所需要的信号。通过对 GPIF 寄存器进行 CPU 读和写操作，GPIF 可以移位数据。本文档通过提供一个简单设计 (该设计对 GPIF 时钟进行 2、4 和 7 分频) 详细介绍了 GPIF 单元和它的图形设计工具 (称为 GPIF Designer)。在配置和管理该接口时，仅需要三行 C 代码。本应用笔记还提供了一个示例，用于演示了如何将 USB 连接添加到 GPIF 设计。

关联项目

有

软件版本

Keil uVision 2, GPIF Designer

相关应用笔记

要获取完整的应用笔记列表，请单击**此处**。

更多示例代码? 我们听到了。

要获取更多 USB Hi-Speed 示例代码，请访问**网站**。

目录

关于本文档 **1**
目录 **1**
1 简介 **3**
2 FX2LP 架构概述 **4**
2.1 端口模式 4
2.2 从设备 FIFO 模式 4
2.3 GPIF 模式 — 自动 5
2.4 GPIF 模式 — 手动 5
3 通用可编程接口 **6**
3.1 GPIF 概述 6
3.2 物理互连 7
3.2.1 IFCLK 7
3.2.2 GPIFADR[8:0] (仅输出) 7
3.2.3 FD[15:0] (双向) 7
3.2.4 CTL[5:0] (仅输出) 7
3.2.5 RDY[5:0] (仅输入) 7
3.2.6 GSTATE[2:0] (仅输出) 7
4 创建一个 GPIF 应用 **8**

目录

4.1 设计 GPIF 接口 8
4.2 使用固件框架 9
4.3 使用 GPIF Designer 实现 GPIF 波形 9
5 示例 1：对 GPIF 时钟进行 2 分频和 4 分频 10
6 示例 2：对 GPIF 时钟进行 7 分频 19
7 示例 3：使用单个字的读/写数据操作 22
7.1 执行 FIFO 读/写操作 22
7.2 需要时进行优化 22
7.3 将 FIFO 连接至 FX2LP GPIF 接口 24
8 USB 数据流 26
9 设计 GPIF 互联 27
9.1 单字写波形 28
9.2 单个字读波形 33
9.3 单个字操作的固件编程 36
9.4 代码段 37
9.4.1 TD_Init() 37
9.4.2 触发 GPIF 单个字写操作 39
9.4.3 GPIF 单字读操作 40
9.4.4 TD_Poll() 40
9.5 运行 GPIF 单个字操作示例 43
9.5.1 不存在外部 FIFO 43
9.5.2 存在外部 FIFO 44
9.6 单个字操作的逻辑分析仪波形 44
9.6.1 单个字写波形 44
9.6.2 单个字读波形 45
10 相关文档 46
10.1 其他 GPIF 示例 46
10.2 参考设计 46
10.3 数据手册 46
11 总结 47
文档修订记录 48

1 简介

USB 2.0 的 480 Mbps 信号速率要求控制器芯片对高速数据进行输出/输入传输。EZ-USB FX2LPGPIF 提供了一个独立的硬件单元，CPU 通过对其进行相关设置实现 USB 端点 FIFO 和外部接口之间的数据直接传输。外部接口可以是 RAM、FIFO 或第二个处理器。因此，CPU 不需要移动数据。配置时，由于数据经过了 GPIF 硬件通道，因此 CPU 仅控制各标志和中断。

通过使用 GPIF 可以实现各种协议，包括增强型 IDE (EIDE – 有时成为快速 ATA 或快速 IDE) 或 ATA 数据接口 (ATAPI) 打印机，并行接口 (IEEE P1284)，以及 Utopia。本文档对 FX2LP GPIF 的架构和实现进行了说明。介绍了应用使用模型和调试决策，并提供了各种示例用于介绍和解释 GPIF 概念。

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 的最大宽度。FX2LP 通过提供 GPIF 为外部器件提供高速并行接口，从而优化 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不同 FX2LP 模式，对 FX2LP 架构进行了简要说明。

2.1 端口模式

FX2LP 拥有 24 个接口引脚，根据模式设置，可将这些引脚用于各种目的。在“端口”模式下，它们是通用的 I/O 引脚，并且 GPIF 无效 (**Figure 1**)。

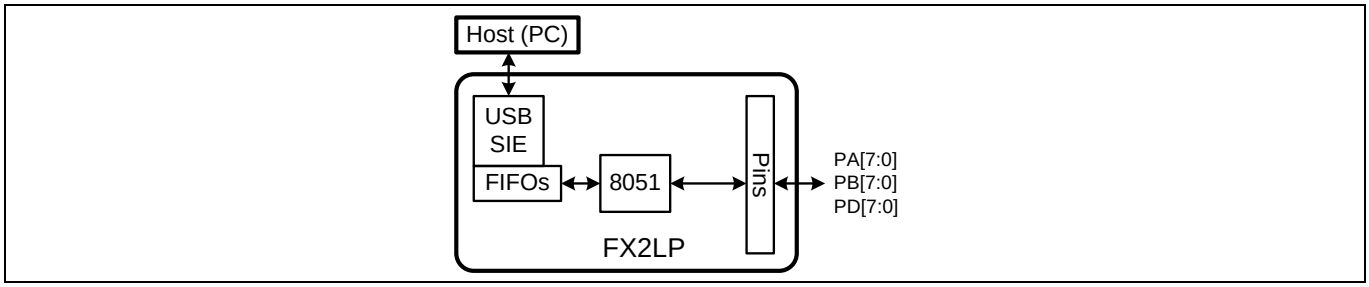


Figure 1 端口模式下的 FX2LP

2.2 从设备 FIFO 模式

在从设备 FIFO 模式中，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，该接口还提供了常用的 FIFO 信号，如 RD、WR 和 FIFO 标志。更多有关该接口的详细信息，请参考英飞凌应用笔记 **AN63787 - 使用 8 位异步接口配置 EZ-USB FX2LP GPIF 和 Slave FIFO 的示例**。

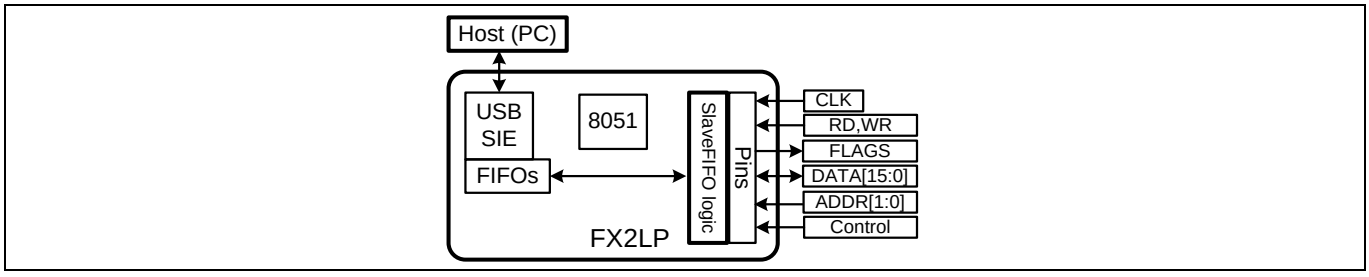


Figure 2 从设备 FIFO 模式下的 FX2LP 引脚

2.3 GPIF 模式 — 自动

当 GPIF 有效时，接口引脚作为主设备控制外部外设，如 RAM、FIFO 或外部处理器。GPIF 可在两个子模式下运行：自动模式和手动模式。在自动模式下，数据从端点 FIFO 直接输送到外部接口。8051 配置和监控该接口，但不会直接访问 FIFO 数据；请参考 **Figure 3**。

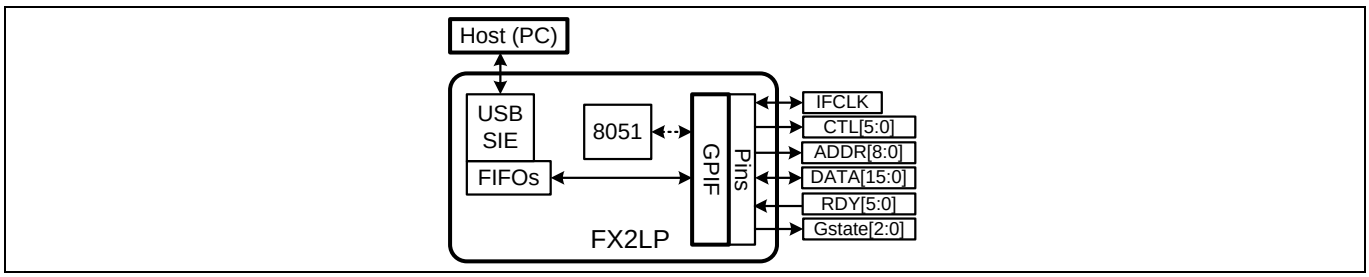


Figure 3 GPIF 自动模式下的 FX2LP

赛普拉斯应用笔记 **AN57322 - 通过 GPIF 使 SRAM 与 FX2LP 相互连接**来说明 GPIF 自动模式。该笔记介绍了如何使用 8 位异步接口和 GPIF 自动模式将赛普拉斯 CY7C1399B SRAM 连接至 FX2LP。

2.4 GPIF 模式 — 手动

在手动模式下，通过使用 GPIF 寄存器的读和写操作，8051 对该接口进行读写字节 (**Figure 4**)。在本文档的后续内容中介绍了 GPIF 手动模式的一个**示例 3：使用单个字的读/写数据操作**。

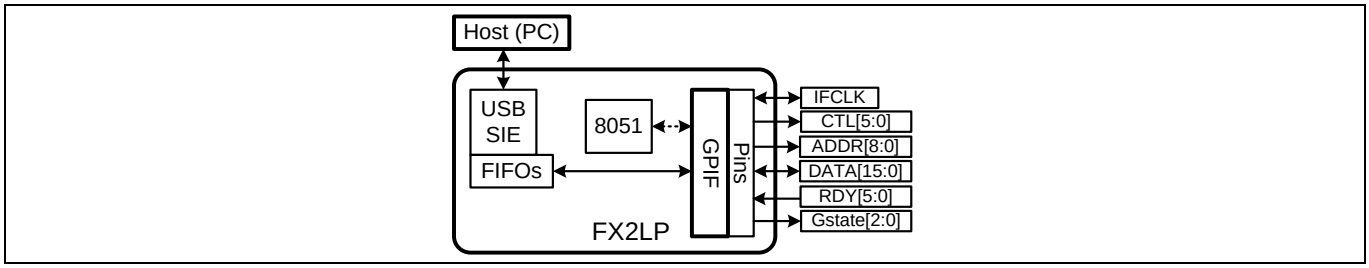


Figure 4 GPIF 手动模式下的 FX2LP

3 通用可编程接口

3.1 GPIF 概述

GPIF 的内核是一个可编程状态机，用于控制 8 位或 16 位双向数据总线，并生成多达六个控制 (CTL) 和九个地址 (GPIFADR) 输出。另外，它还接收六个外部和两个内部 READY 输入，用以确定分支条件。用户定义的四个波形描述符控制该状态机；8051 程序会选择四个波形中的某一个在给定的时间内有效。

每个 GPIF 波形描述符包含多达七种状态，名称分别为 S0-S6。预定义的 S7 用于表示闲置状态。在每个状态内，您可以将 GPIF 编程为：

- 使某些或所有 CTL 输出驱动高电平、驱动低电平或悬空
- 采样或驱动 8/16 位数据总线
- 递增 GPIF 地址总线的值
- 递增 FIFO 指针
- 向 8051 触发一个 GPIF 波形中断

每个状态中的分支决策是通过下列选项得到的两个信号的逻辑 AND、OR 或 XOR：

- 六个 READY 输入引脚
- 内部 FIFO 标志
- 内部 RDY 标志
- 内部事务处理终止标志

通过两个已选信号的逻辑结合可确定下一个状态。另外，计算好可编程延迟后，状态机会转入下一个状态。该延迟的范围为 1 到 256 个时钟周期。

采样和分支的状态被称为“决策点”。不带决策点的所有状态仅会维持一个时钟间隔，在下一个时钟间隔内，它将自动转到下一个状态。

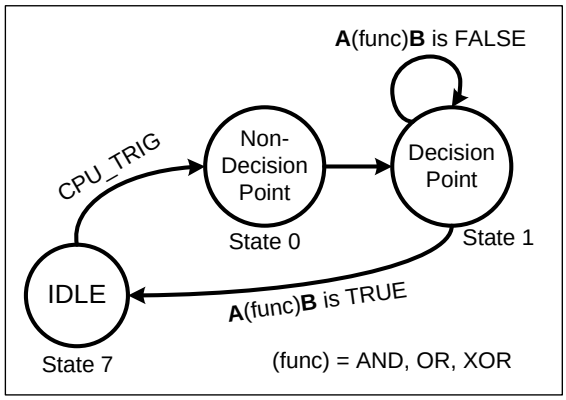


Figure 5 GPIF 状态机句法

通用可编程接口

Figure 5 中的状态机的流程如下所示：

1. 从闲置状态开始，等待 CPU_TRIG 信号激活。
2. 激活 CPU_TRIG 时，转入状态 0；在该状态中，您可以激活某些控制信号输出，移动数据或递增地址总线。该状态仅在下一个时钟前持续，因为没有任何决策，所以下面它会无条件地转入状态 1。
3. 并保持状态 1，直到两个信号 A 和 B 的逻辑组合为真为止 (即它为错误时)。例如，要想一直保持状态 1，直到 READY 信号有效或完成一个计数为止，那么需要选择 A 代表 READY 输入，并选择 B 代表一个计数器到期。如果发生两个条件其中一个 (READY 或终端计数)，状态机可通过使用逻辑运算符 OR 输出状态 1。
4. 当该条件为真时，将转入闲置状态。状态机也随即停止。

3.2 物理互连

GPIF 互连包含一个 8 位或 16 位的数据总线、地址总线、控制输出和就绪输入 (Figure 4)。另外它还包含了三个 GSTATE 输出 (用于指出 GPIF 机械的当前状态)，用于调试目的。本节详细说明了这些信号。

3.2.1 IFCLK

IFCLK (接口时钟) 是所有 GPIF 操作的参考时钟。可将其作为输入或输出信号使用；并且您可以将上升沿或下降沿选为有效沿。当作为输入信号使用时，可以使用一个频率为 5 MHz 到 48 MHz 的外部时钟驱动它。作为输出信号使用时，可以使用 FX2LP 中一个频率为 30 MHz 或 48 MHz 的内部时钟进行驱动 IFCLK。如果外部外设需要的时钟频率更低，那么可以使用 FX2LP 的内部时钟切换各 CTL 线中的某一个。在第一个示例中，GPIF 时钟被 2 分频和 4 分频，并通过使用两个 CTL 输出可以输出这些信号。如果需要进行更高的分频，请在进入下一个状态前，通过编程 GPIF 状态计算已编程的时钟数量 (1 到 256)。

3.2.2 GPIFADR[8:0] (仅输出)

GPIF 通过驱动 GPIFADR[8:0]为需要地址行的外设提供地址行。在所有 GPIF 状态下，可以保持或递增这些输出。

3.2.3 FD[15:0] (双向)

该数据总线是一个通道，通过它可以在 FX2LP 端点 FIFO 和外部外设间传输负载数据。该信号经配置后可作为 8 位或 16 位接口运行。系统需要时，可进入三态。在 16 位模式下，FD[7:0]代表端点 FIFO 的第一个字节，而 FD[15:8]则代表第二个字节。

3.2.4 CTL[5:0] (仅输出)

控制输出信号可提供外部外设所需要的各种信号，如读/写选通、使能和分频时钟。

3.2.5 RDY[5:0] (仅输入)

就绪输入信号提供了外部外设的状态信息，如 FIFO 状态标志和有效数据。GPIF 能将这些信号作为决策点限定符使用。

3.2.6 GSTATE[2:0] (仅输出)

调试输出信号代表在 GPIF 波形中执行的状态。这些信号与逻辑分析仪相连，以用于调试目的。

4 创建一个 GPIF 应用

本节介绍了创建 GPIF 应用的各步骤。

4.1 设计 GPIF 接口

要设计 GPIF 互连，您需要了解 FX2LP 和外部外设器件之间的接口。通过使用 **FX2LP 数据手册**和**技术参考手册**可以定义该接口。以下决定决定了如何配置 GPIF。

- **8 位或 16 位数据路径?**

该决定通常取决于外设提供的数据路径的大小。如果大小为 16 位，则可以使用该路径尽可能扩大物理接口的带宽。

对于 16 位的数据路径，当连接至数据总线时，需要考虑字节顺序 (字节次序) 和位编号。

- **外部还是内部接口时钟?**

决定取决于外设操作模式的灵活度。例如，如果它可以接收一个频率为 30 或 48 MHz 的外部时钟输入，那么内部 GPIF 时钟便能与外设时钟输入相连。

- **是否需要地址行?**

如果在读/写周期操作期间，外设要求对任何寄存器或存储器位置进行寻址，则可以使用 GPIFADR[8:0]。

- **控制行**

从 CTL[5:0]指定 GPIF 控制输出。在这些操作中，外设可能需要读/写信号、芯片选择和其他控制输入。确定有哪些输入，并合理分配 CTL[5:0]。通过 GPIF Designer 工具，您能够更清晰地命名这些信号；例如 CTL[0]的 WR#和 CTL[1]的 RD#。

- **状态 (RDY) 行**

确定在读/写周期期间需要监控的状态信号数量。指定包括哪些状态信号，并合适分配 RDY[5:0]。GPIF Designer 工具会再次允许您命名这些信号，以便使这些名称符合您的设计。

- **接口时序**

分配输入和输出之后，GPIF 应用的主要任务是设计时序波形，从而考虑接口时序。

Note: *并非所有 FX2LP 封装类型均提供 GPIF 接口信号的完整集。例如，100 引脚和 128 引脚的 FX2LP 封装提供了所有六个就绪输入 (RDY[5:0]) 和控制输出 (CTL[5:0])。56 引脚封装提供了两个 RDY 信号 RDY[1:0]和三个 CTL 信号 CTL[2:0]。*

4.2 使用固件框架

使用 GPIF Designer 创建接口信号和波形时，您需要使用 Keil 集成开发环境 (IDE) 对控制固件进行编写。开始操作新的 FX2LP 固件项目时，优先使用基于赛普拉斯编写的固件框架的 Keil uVision2 项目。**FX2LP Development Kit (DVK)** 所提供的固件示例都是基于框架的。您可以使用这些示例其中的任何一个进行操作，或将 Keil 项目复制到新的子目录内，以进行修改。这样，您便能够对空的固件基础进行操作。如果使用了固件框架项目，那么您可以集中到您的应用代码，因为已经写入了 USB 低电平协议代码。更多有关信息，请参考 *fw.c* 文件和开发套件文档。

应用方案拥有两个主要组件：

- 配置 GPIF 和启动 GPIF 传输的固件。该固件还会执行其他应用任务，如 USB 枚举和端点配置。
- 实现物理总线时序和数据流的 GPIF 波形描述符。

该固件包含五个文件：*fw.c*、*periph.c* (可以重新命名该文件)、*dscr.a51*、*ezusb.lib* 和 *usbjmptb.obj*。这些文件拥有 Keil uVision 2 固件框架项目。

第二个组件是一个包含了代码的 C 源文件 (例如，*gpif.c*)，它用于定义 GPIF 波形和初始化 GPIF 单元。当您通过图形方式定义接口后，GPIF Designer 工具将创建该 C 文件。这样便不用知道 GPIF 单元中的单独寄存器和位。您可以在**此处**下载 GPIF Designer 工具。

4.3 使用 GPIF Designer 实现 GPIF 波形

GPIF Designer 生成包含了 GPIF 波形描述符的 C 代码，使用这些描述符可以实现接口的物理总线时序。GPIF Designer 可实现多个波形性能，包括四种波形类型：单写、单读、FIFO 写和 FIFO 读。CPU 通过写入 GPIFWFSELECT 寄存器控制所使用的波形。每一个描述符的长度均为 32 个字节，它位于片上存储器空间内专用的 GPIF 波形描述区域内。通过 GPIF Designer 工具，您可以将这些描述符作为“黑箱子”使用。这是因为它可以生成所有需要的 C 代码，用于实现和使用这些描述符。

将这些 GPIF 波形作为状态机使用。存在七种状态或间隔 (S0-S6)，以及一个用于终止操作的自动闲置状态 S7。

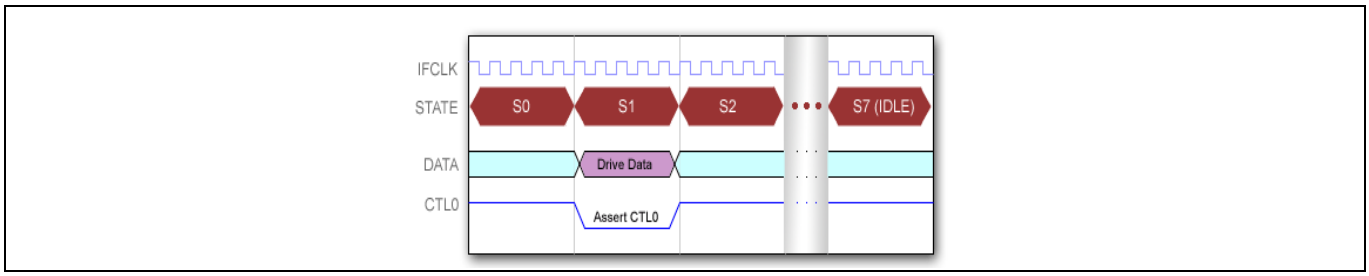


Figure 6 由各状态构成的 GPIF 波形

Figure 6 显示的是一个示例，它说明了一个简单的波形被分解为各 GPIF 状态转换。

5 示例 1：对 GPIF 时钟进行 2 分频和 4 分频

首先，需要设计一个状态机用于对 GPIF 时钟进行 2 分频和 4 分频，然后在 CTL[0]和 CTL[1]上分别输出分频后的时钟信号。这样，在介绍高级特性包括 USB 端点 FIFO 前，可以了解 GPIF Designer 工具的基本知识。由于在本示例中没有使用 USB，因此不需要固件框架。

1. 解压并存储随本应用笔记附带的 *FX2LP source code and GPIF project files.zip* 文件。转到 `FX2LP Source code and GPIF project files\Firmware\GPIF Clock Divider`。右击该文件夹并单击 **Properties** (属性) 项。如果选择了 **“Read-only...”** 项，请取消勾选它。点击 **OK**，以允许子目录采用修改内容。

   在该文件夹内，双击 **GPIF_Clock_Divider.uv2**。它包含一个基本的 GPIF 项目，此项目包含所有信息，但不包括 GPIF 信息。双击 *main.c* 文件以打开它；您将看到仅需要三行代码便能够启动 GPIF 操作：

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

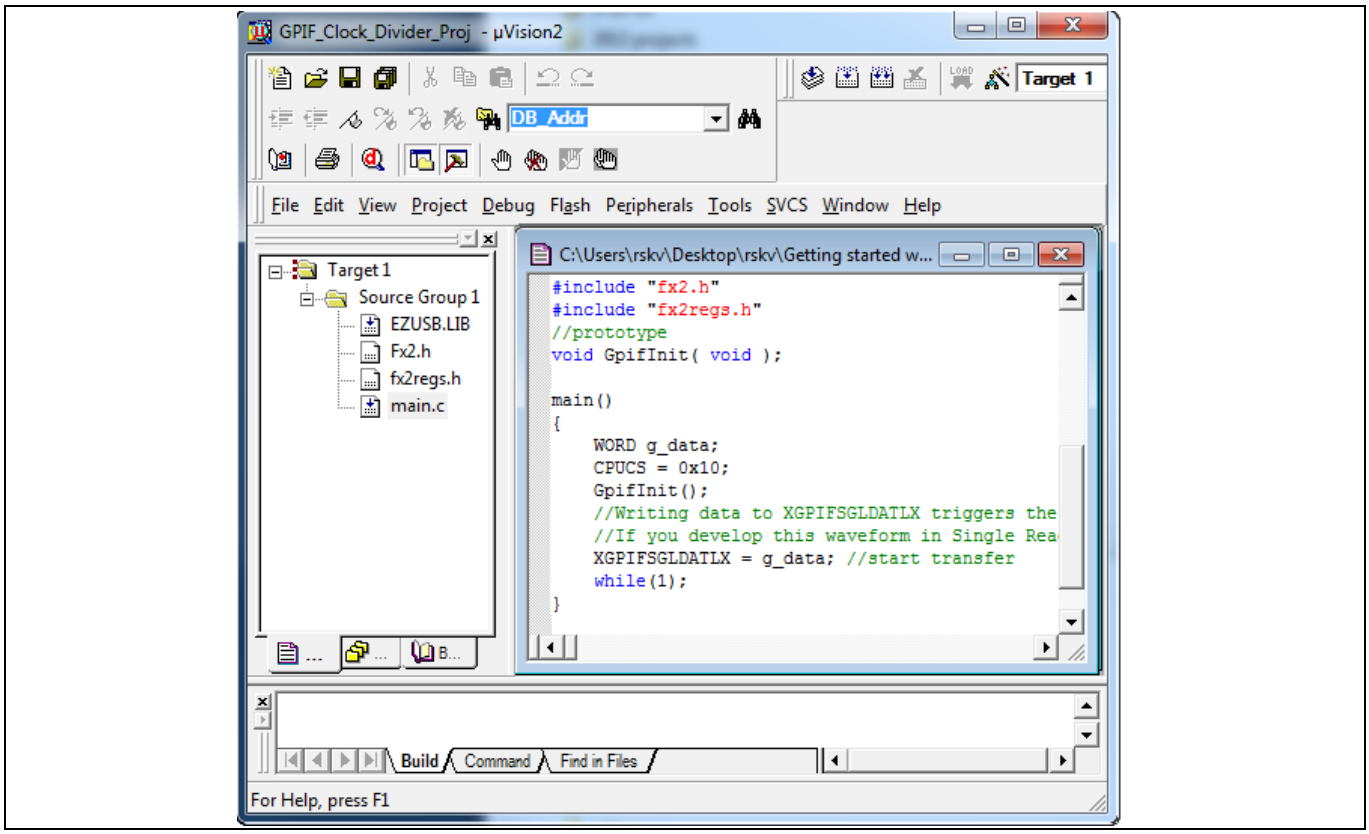


Figure 7 基本的 GPIF Keil 项目

如果您没有修改任何代码，您会看到链接器错误，这是因为 GPIF Keil 项目未包含 GPIF Designer 生成的 GPIF C 文件。该文件包含 GpifInit()函数和波形表数据。下一步骤将创建该文件。

示例 1：对 GPIF 时钟进行 2 分频和 4 分频

2. 启动 GPIF Designer，然后依次选择 **File > New** 打开下面图片显示的窗口。

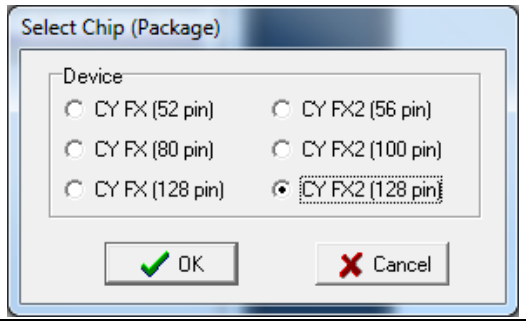


Figure 8　GPIF Designer 启动窗口

3. 要想使用 FX2LP 开发板检测该示例，请选择 **CY FX2 (128 引脚)** 并点击 **OK**。这样可打开框图窗口：

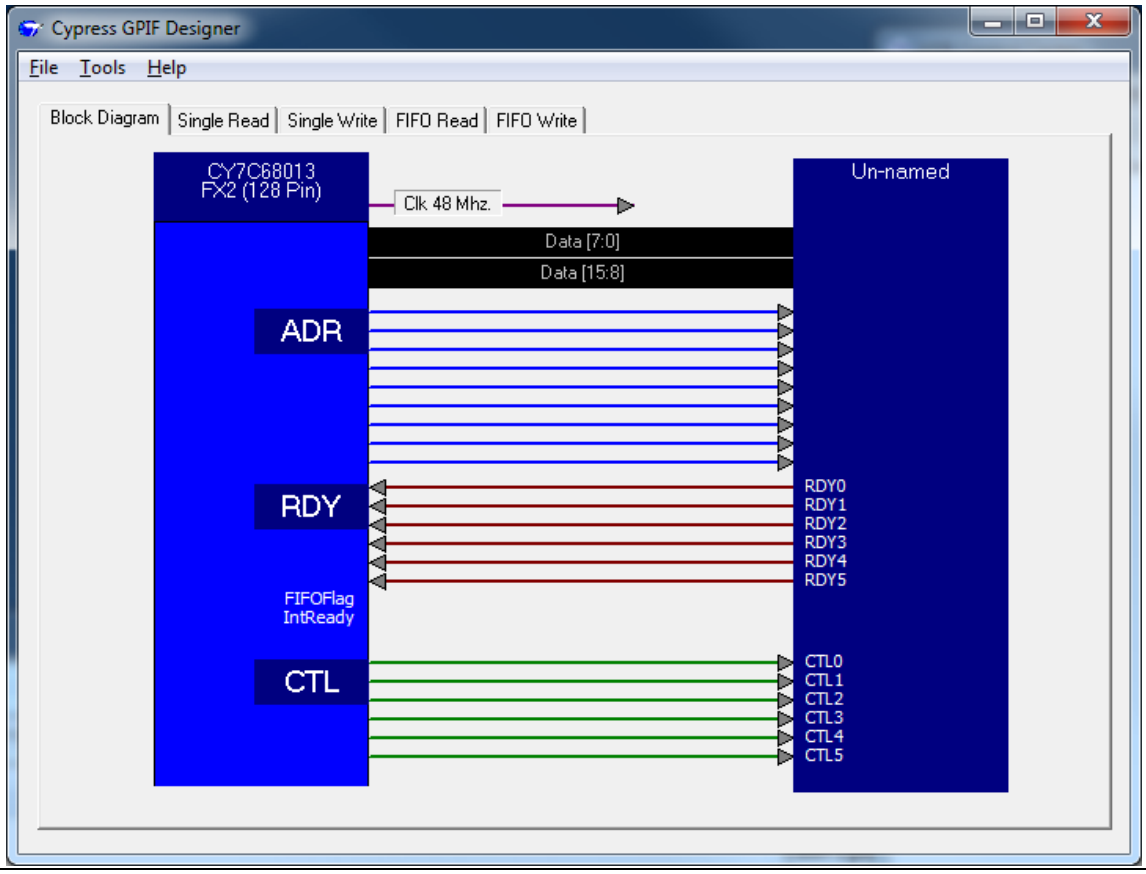


Figure 9　GPIF Designer 框图

4. 该框图顶部，右击 **Clk 48 MHz** 文本框配置 GPIF 时钟。对于该项目，则保持默认设置。

示例 1：对 GPIF 时钟进行 2 分频和 4 分频

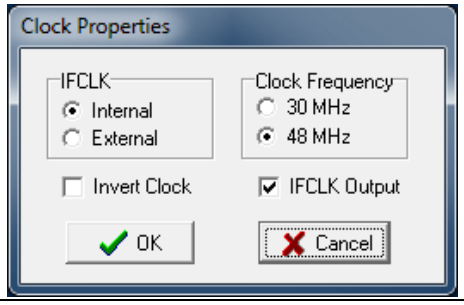


Figure 10　IFCLK 默认设置

5. 右击 **ADR** (地址) 标签。该示例没有使用地址总线，因此您可以点击 **Disable All**；这样会使框图内的各地址行变暗。

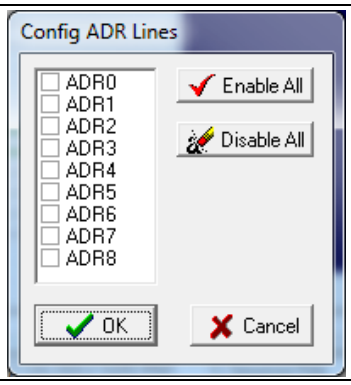


Figure 11　禁用地址行

6. 通过右击 **RDY** (就绪) 标签配置六个 RDY 输入。该示例没有使用 RDY 输入，因此未选中这些输入。要取消选择 RDY5，请先取消选择 **Subst TC for RDY5**。

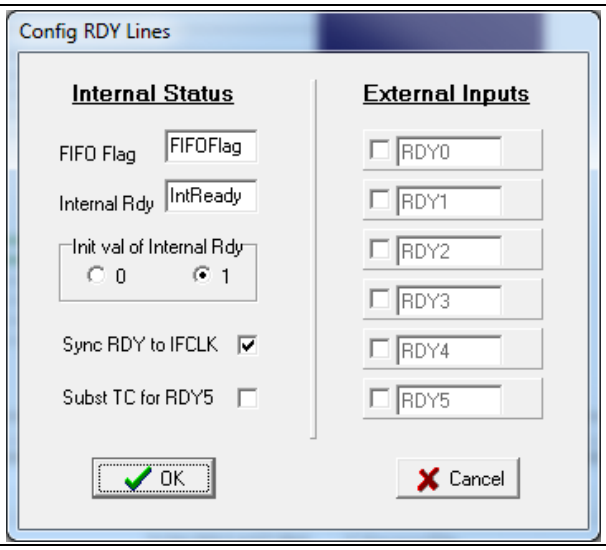


Figure 12　禁用 RDY 输入

7. 通过右击 **CTL** (控制) 标签进行配置六个 CTL 输出。按照下图设置这些输入。

示例 1：对 GPIF 时钟进行 2 分频和 4 分频

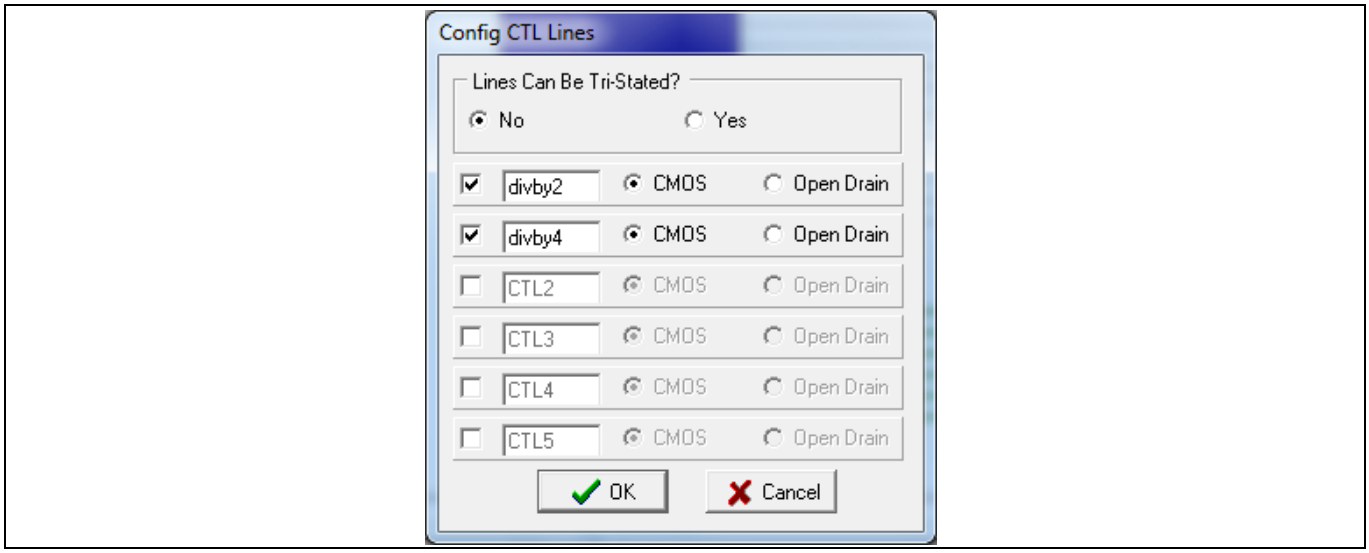


Figure 13　CTL 输出设置

将两个输出重新命名为 **divby2** 和 **divby4**，并将这两个名称输入到文本框内。推荐命名您的信号，这是因为它们会更新所有 GPIF Designer 屏幕中的标签，从而更容易识别这些信号 (例如，divby4 而不是 CTL1)。现在该框图会显得更加简单。

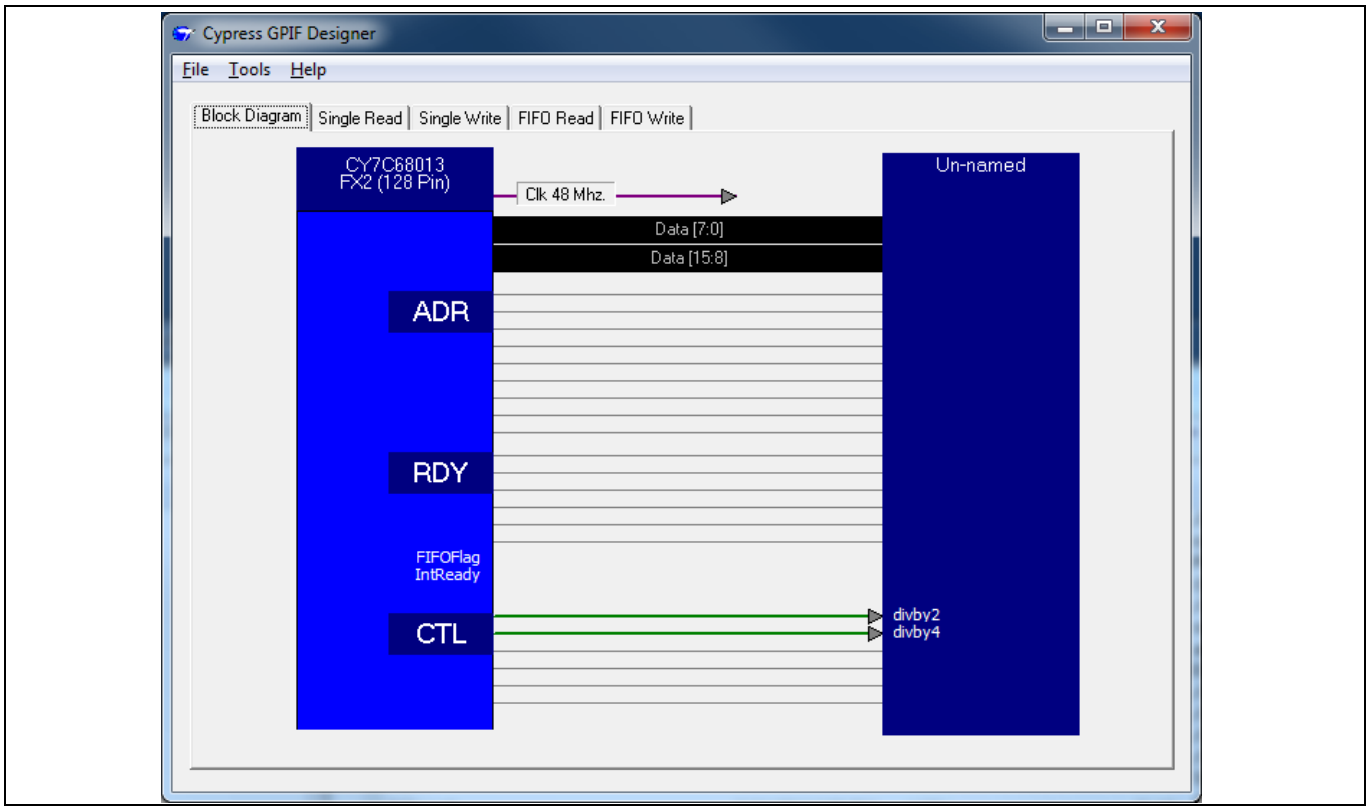


Figure 14　时钟分频器框图

8. 选择 **Single Write** 选项卡，以进行绘制两个波形。右击该选项卡的名称，选择 **Select Tab Label**，并将其重新命名为 **Divider**。您会看到一个空白波形的编辑界面，如 **Figure 15** 所示。

示例 1：对 GPIF 时钟进行 2 分频和 4 分频

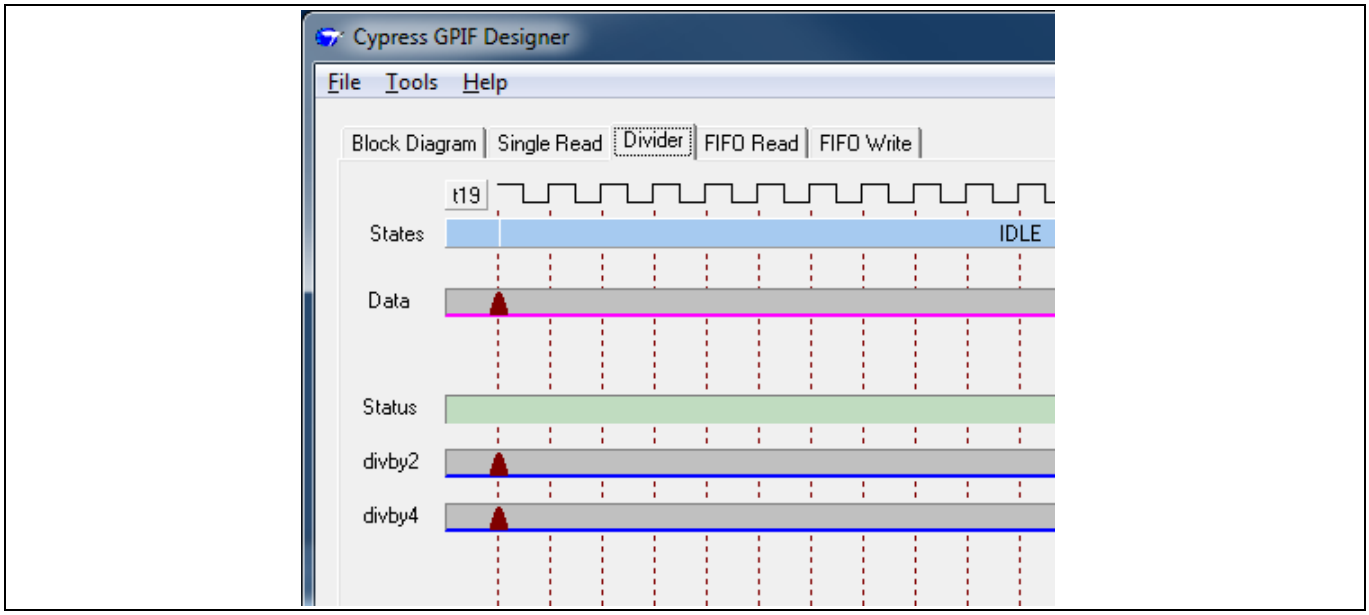


Figure 15　空白波形编辑器界面

9. 该状态机有一个名称为 IDLE (闲置) 的状态。要想添加某个状态，需要点击 CTL 带 (divby2 和 divby4) 中的某一个。另外，您还可以通过点击数据带添加各种状态，但该设计不使用数据总线。在 divby2 带中，点击第二个垂直虚线。

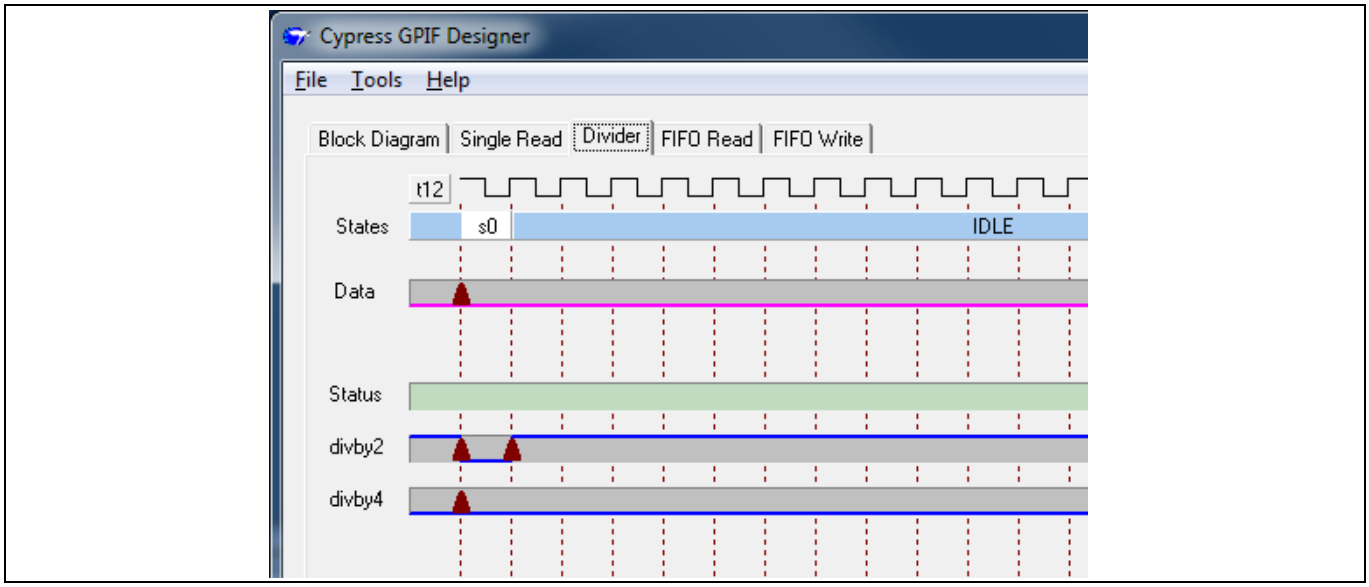


Figure 16　第一个波形切换

出现的三角形表示 GPIF Designer 已经添加了状态 s0，并在其开始和结束时切换了 divby2 输出。

10. 现在，请在后面每三个时钟切换虚线上点击 divby2 波形，这样可创建 2 分频时钟。
11. 在 divby4 波形中，点击第二个和第四个时钟线。您的屏幕将显示。

示例 1：对 GPIF 时钟进行 2 分频和 4 分频

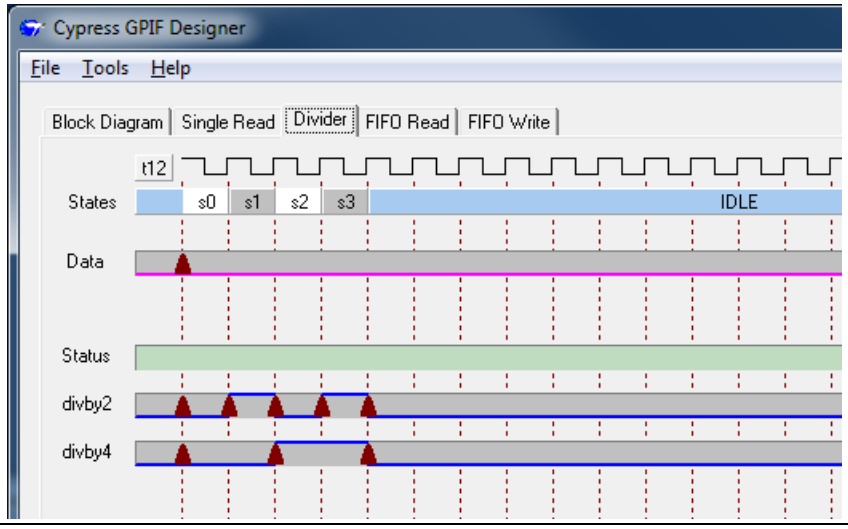


Figure 17　2 分频和 4 分频波形

Note: *可通过右击任何一个三角形来修改它的逻辑状态，从而修改信号属性。另外，通过横向拖动这些三角形，以使某个状态持续时间大于一个时钟。*

Figure 17 中显示的状态机启动时，它将在每个时钟上升沿上从 s0 状态依次转到 s1、s2 和 s3，然后以闲置状态停止。您需要重复该过程；也就是说 s3 始终要返回到 s0 状态。为此，在状态线中将添加一个‘决策点’。

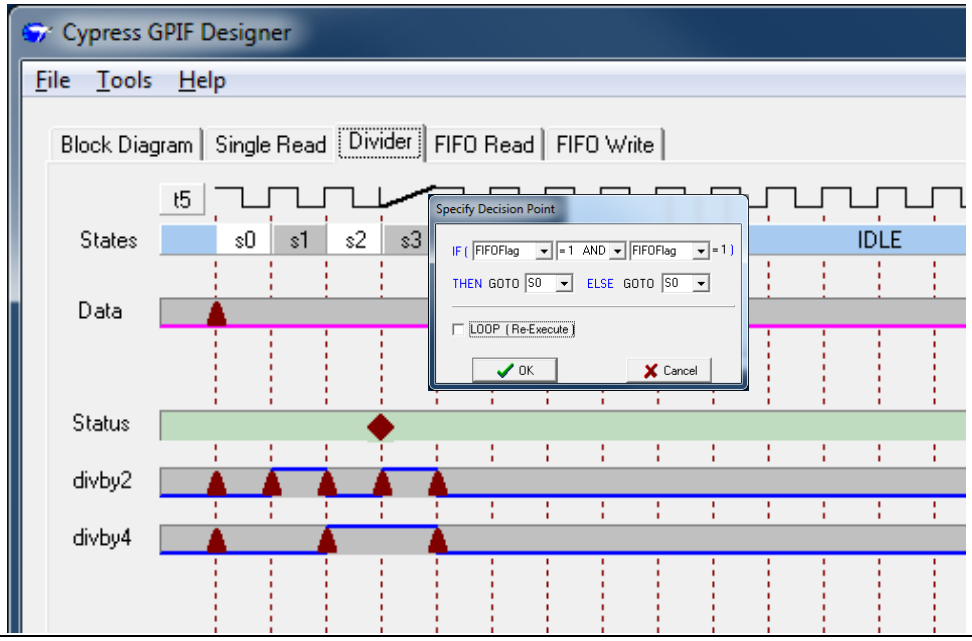


Figure 18　第一个决策点

示例 1：对 GPIF 时钟进行 2 分频和 4 分频

12. 点击 s2 和 s3 状态之间的状态区间。这样可添加一个作为决策点的菱形，并显示一个窗口用于配置该决策。要想进行一个无条件反转，您可以从下拉列表顶部选取两个逻辑信号以及它们的逻辑运算符。您需要指定 THEN 和 ELSE 条件，以切换到 S0 状态。也就是要*无条件*反转到 S0 状态。每次进入该状态，可以使用 **LOOP (Re-Execute)**复选框重新激活状态条件。s3 状态中不存在“保持该状态直到发生变化”循环，因此可以取消勾选该项。

    在状态 s3 中，时钟波形显示一个对角线以表示未知决策状态的时钟数量。这是因为时钟数量取决于满足转换条件的时间。

    追踪该设计使用的所有参数，并进行相应的选择项更新。例如，如果您添加了一个状态 s4，那么它将自动被添加到 GOTO 选项内。

    注意: 决策点位于状态的开始位置，而不是结束处 (决策点在该状态下发生)。因此，菱形位于 s3 开始的位置。

13. 要存储 GPIF Designer 项目，请依次选择 **File**>**Save As** 并将该项目存储在 Keil 项目文件夹 as*Div_2_4.gpf* 内。如果您想修改当前波形，你可以重新打开该文件。要存储 GPIF Designer 生成的 C 代码，请依次选择 **Tools**>**Export to GPIF.c file** 并将该代码存储在 Keil 项目文件夹 *GPIF_div_2_4.c* 内。*Div_2_4.gpf* 和 *GPIF_div_2_4.c* 同样也在 `FX2LP Source code and GPIF project files\GPIF Clock Divider\GPIF_div_2_4` 文件夹内提供。通过使用这些文件，您可以直接测试该项目。
14. 最后一步是将 GPIF Designer C 文件添加到 Keil 项目。右击 **Sources Group 1** 并选择 **Add files…**。找到 Keil 项目文件夹并添加 *GPIF_div_2_4.c* 文件。

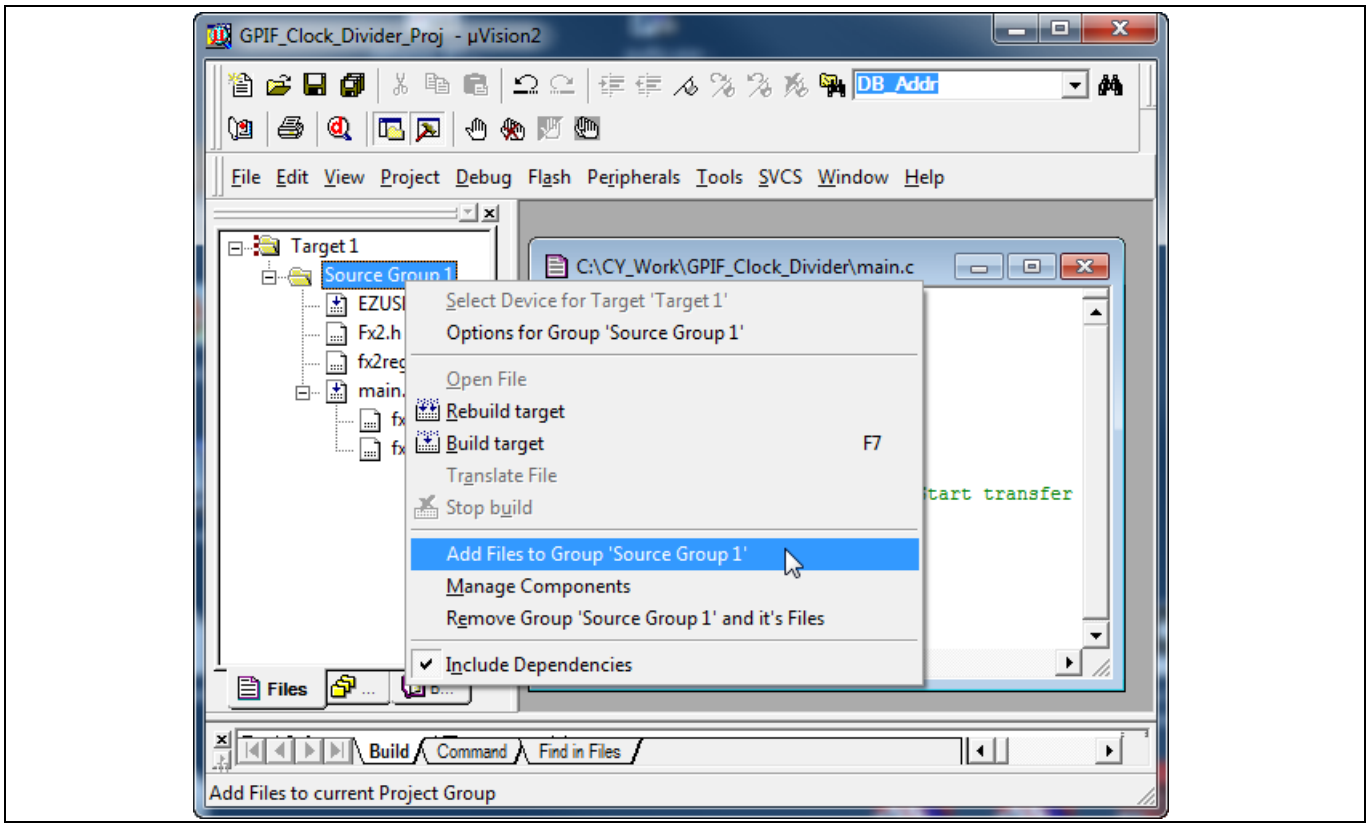


Figure 19　将 GPIF Designer C 文件添加到 Keil 项目

示例 1：对 GPIF 时钟进行 2 分频和 4 分频

15. 编译该项目。Keil IDE 编译并链接这些文件，然后通过调用 ‘hex2bix’ 工具将负载模块转换为一个与赛普拉斯 USB 控制面板加载程序相兼容的格式。

    按照下列步骤进行操作，您可以在 FX2LP 开发板上尝试执行该设计：

    1. 关闭 EEPROM SELECT 开关。以使能 FX2LP 片上的 USB 代码加载程序。
    2. 将 FX2LP 板插入到 PC USB 端口。如果是第一次操作，您会看到弹出消息，提示您安装 USB 驱动程序。请找到 `C:\Cypress\USB\CY3684_EZ-USB_FX2LP_DVK\1.0\Drivers\cyusbfx1_fx2lp`，然后选择对应 Windows OS 的文件夹。您可以通过查看 Windows Device Manager (Windows 器件管理程序) 来确认驱动程序是否安装成功：

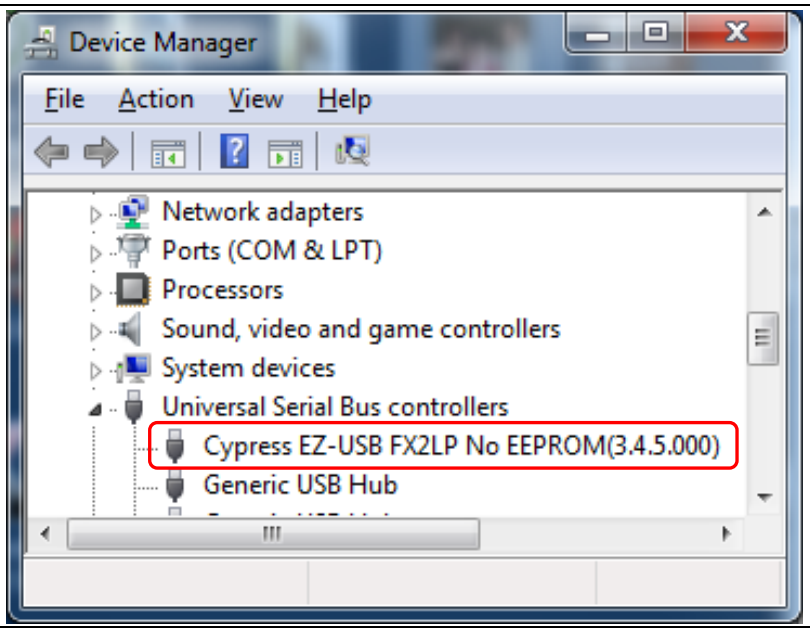


Figure 20　英飞凌 USB 加载驱动程序已安装

16. 在 `C:\Cypress\Cypress Suite USB 3.4.7\CyUSB.NET\bin\CyControl.exe` 中启动 USB 控制面板。
17. 在左侧面板上，您可看到 FX2LP 板以及与其相连的其他 USB 器件。如果仅需查看 FX2LP 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。左侧面板如 **Figure 21** 所示。

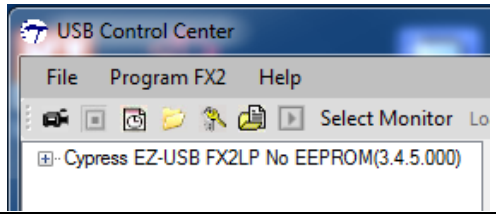


Figure 21　USB 控制中心内的 FX2LP 板

18. 现在您可以将 Keil 编译的十六进制文件加载到该板上。加亮显示 EZ-USB 输入并选择 **Program FX2 >RAM**。

示例 1：对 GPIF 时钟进行 2 分频和 4 分频

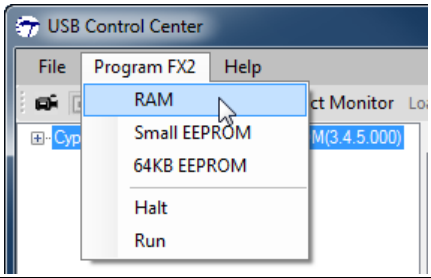


Figure 22　将新器件代码加载到 RAM

19. 按下 FX2LP 开发板上的 **RESET** 按键，以使能 USB 加载程序。
20. 找到 Keil 项目文件夹并选择 Keil 编译器的结果 *GPIF_Clock_Divider_Proj.hex*。
21. 用探针检测 P2 引脚 11 (CTL0=divby2) 和 P2 引脚 10 (CTL1=divby4)。此时，您可以看到如 **Figure 23** 所示的波形。

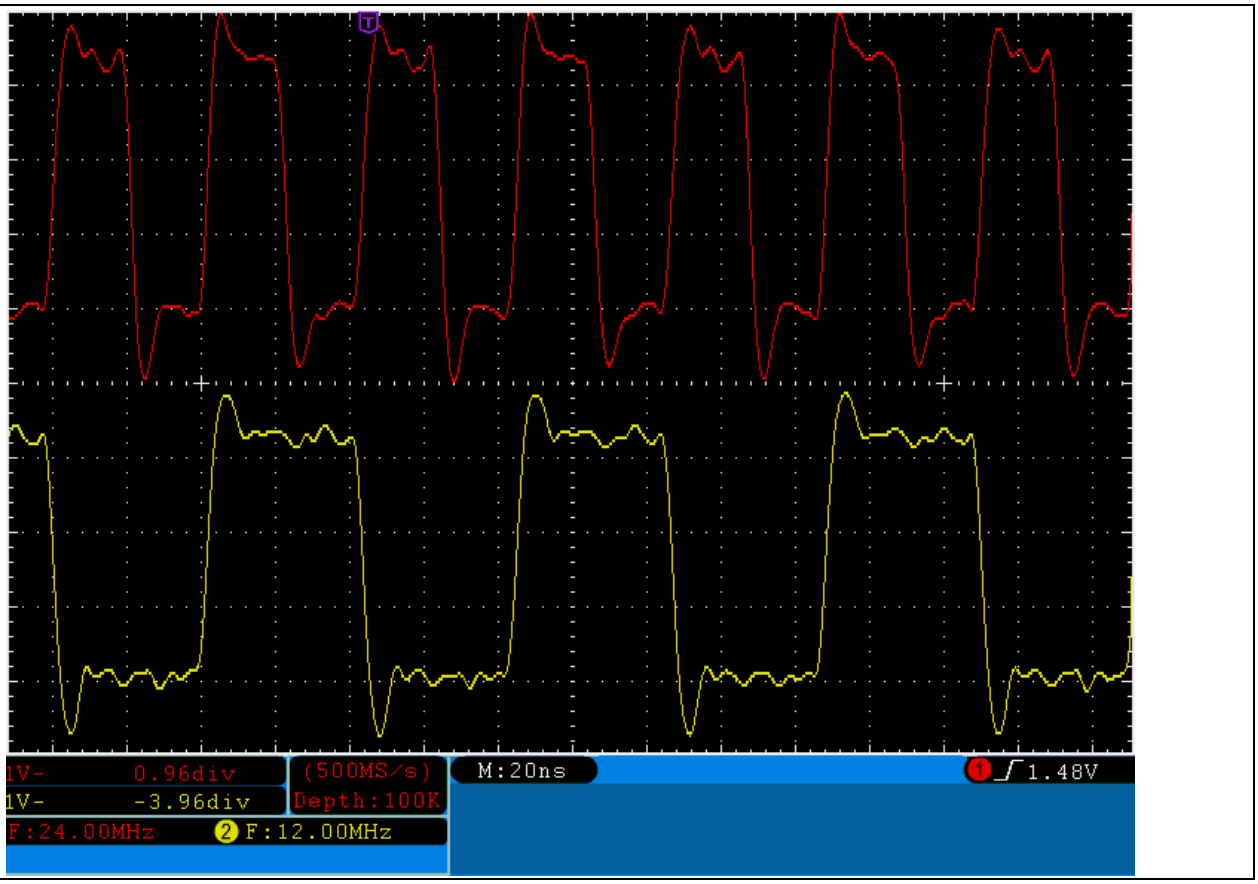


Figure 23　进行 2 分频和 4 分频后的 GPIF 48 MHz 时钟

6 示例 2：对 GPIF 时钟进行 7 分频

1. 通过修改示例 1，您可以选择任何时钟分频器，即使使用的是奇数除数的时钟。
2. 启动 GPIF Designer; 依次选择 **File**>**New** 来创建 GPIF 项目。选择 **CY FX2(128 pin)** 器件。
3. 清除先前描述的 ADR 和 RDY 信号，并将 CTL0 重新命名为 divby7，然后将 CTL1 重新命名为 stb。禁用其他四个信号。

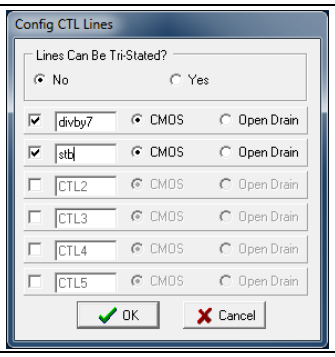


Figure 24 两个输出：7 分频和探针

4. 选择 **Single Write** 选项卡。在 divby7 带中，在第一个三角形后面的三个时钟位置上放置一个状态转换三角形，并在其后面四个时钟所在的位置上放置另一个三角形。在离 stb 带开始后一个时钟的位置上放置一个三角形；右击第一个三角形，并将它设置为 **HI**；将第二个三角形设置为 **LO**。这样会将负极性脉冲转换为状态 s0 中的正极性脉冲。这时，波形如 **Figure 25** 所示。

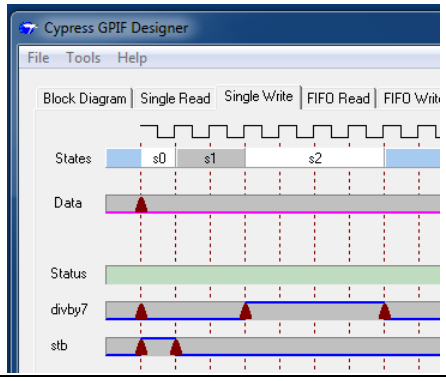


Figure 25 7 分频和探针波形

最后，点击时钟线上的状态带，该线同状态 s2 结束前面一个时钟的位置相对应。这样可以创建决策点和新状态 s3。应该将该决策点配置为无条件转换到状态 S0。由于 THEN 和 ELSE GOTO 的状态都是 S0，因此 IF 条件不重要——您可以从下拉列表中选择任何 IF 条件。

示例 2：对 GPIF 时钟进行 7 分频

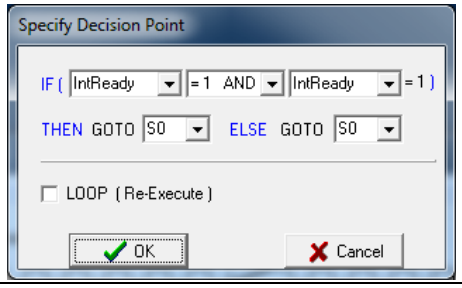


Figure 26　无条件转换到 S0

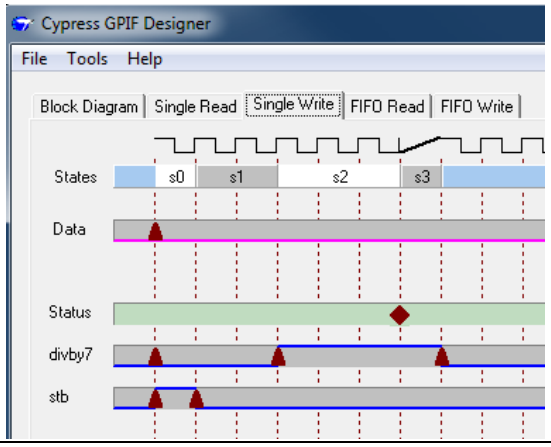


Figure 27　进行 7 分频的最后波形

5. 将该文件 (**Tools**>**Export to GPIF.c**) 导出到 Keil 项目文件夹内，并将它命名为 *GPIF_div_7.c*。将该项目文件 (**File**>**Save As**) 保存为 *Div_by_7.gpf*。*Div_by_4.gpf* 和 *GPIF_div_7.c* 文件也在 `FX2LP Source code and GPIF project files\GPIF Clock Divider\GPIF_div_7` 文件件中提供。您可以直接使用这些文件，进行该项目的测试。
6. 在 Keil Files 选项卡中，右击 **Source Group 1**，点击 **Add Files…**并将 *GPIF_div_7.c* 添加到该项目内。现在您拥有两个 GPIF 波形文件，因此 Keil 编译器需要确定将要使用的文件。要想禁用编译中的任何源文件，需要右击它，选择 **Options for file…**，并取消勾选 **Include in Target Build** 复选框。如果您的项目中拥有多个 *GPIF.c* 文件，请确保仅勾选其中一项。
7. 选择 **Project**>**Rebuild All Target Files**，进行重新编译。
8. 按下 FX2LP 开发板上的 **RESET**，这样可以恢复 USB 代码加载程序的状态。
9. 使用 USB 控制中心面板加载新编译的 *GPIF_Clock_Divider_Proj.hex* 文件。
10. 探针 P2 引脚 11 (CTL0=divby7)；您的屏幕将如 **Figure 28** 所示。

示例 2：对 GPIF 时钟进行 7 分频



Figure 28　顶部：7 分频；底部：stb 信号的宽度为一个时钟，以用于参照目的

如果您需要更长的时钟分频 (如 87 分频)，将如何进行？GPIF Designer 允许您设置一个状态的持续时间为 1 到 256 个时钟。点击状态带中的任何状态；将出现如 **Figure 29** 中所示的窗口。点击 **Set State Duration**，然后设置时钟数量 (**Figure 30**)。当您对某个状态的时钟数量进行手动设置时，时序框图的时钟线将不再指出该状态的时钟数。

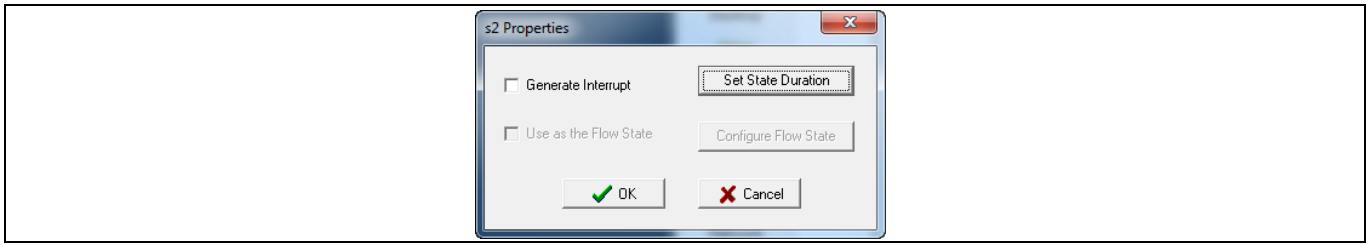


Figure 29　s2 状态的属性

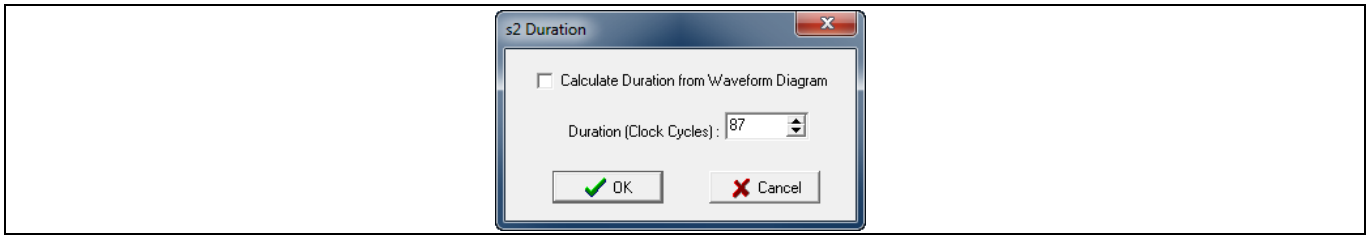


Figure 30　选择 1 到 256 个时钟周期

7 示例 3：使用单个字的读/写数据操作

在该示例中，FX2LP 数据通过赛普拉斯 CY7C4265-15AXC 外部 FIFO 进行循环。该示例使用 GPIF 手动模式并通过一个 16 位数据总线进行单次的读/写 GPIF 数据操作。您可以使用示波器或逻辑分析仪监控 FIFO 写操作，但在检测读操作时，您需要将外部 FIFO 芯片连接至 FX2LP 开发板。

例如，使用开发套件所提供的原型电路板将外部 FIFO 安装到 FX2LP 开发板上。如需了解外部 FIFO 的全部硬件规范，请下载 CY7C4265 数据手册。原型板连接到 FX2LP 开发板的引脚分配列表和外部 FIFO 原型板的完整原理图可在 Hardware 文件夹中找到，该文件夹是本应用笔记附件的一部分。

通过单次读/写操作可在 FX2LP 和外设之间传输一个字节/字的数据。与 FIFO 读/写操作相比，这些操作更简单，因此应该优先执行这些操作。如果在 GPIF 开发周期中您先执行该步骤，那么在处理更复杂的设计前，您需要验证系统的所有部分 (硬件、固件、软件)。在验证完物理互联和基本数据传输后，应该转到完整的设计。

7.1 执行 FIFO 读/写操作

执行完单次操作后，需要执行 GPIF FIFO 读/写操作，以增大带宽。先实现 FIFO 写波形可避免检测整个环回方案时遇到问题；如果它不工作，那么很难区分该问题是读操作、写操作还是两个操作。

7.2 需要时进行优化

生成 GPIF 波形时，要先设置物理总线为高电平的时间。这样可以缩短每个 GPIF 操作的循环时间，同时仍可满足外设所要求的时序参数。您还可以对该设计进行修正，以提高固件代码的效率，并改进本阶段的整体固件代码流。

Figure 31 通过使用 GPIF 设计流程图对本章进行了总结。

示例 3：使用单个字的读/写数据操作

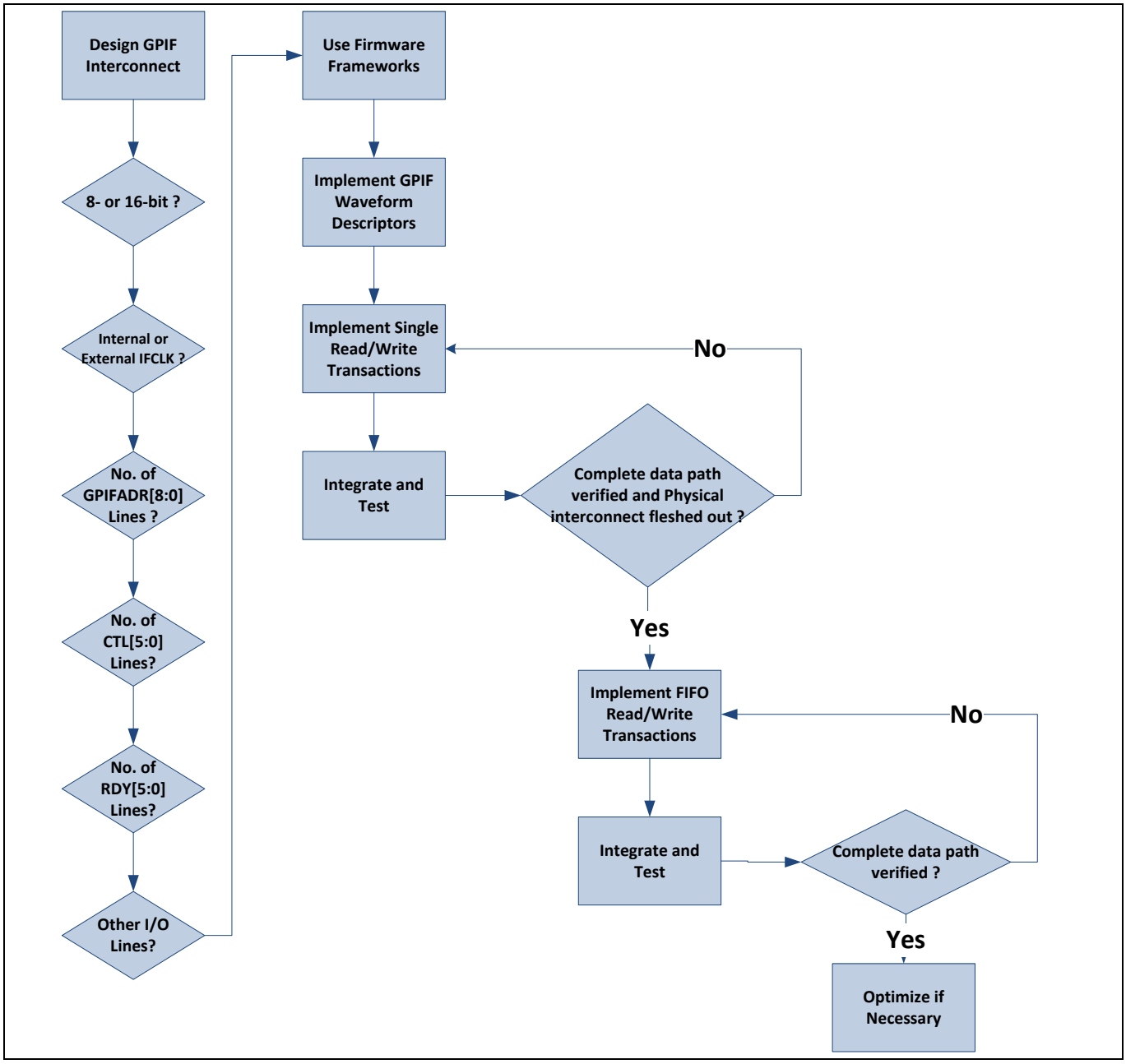


Figure 31　　IF 设计流程图

7.3 将 FIFO 连接至 FX2LP GPIF 接口

Figure 32 显示的是 FX2LP 与外部 FIFO 之间的连接。FX2LP 使用其双向总线 FD[15:0] 对外部 FIFO 中的数据进行读写。通过将 FIFO 的输出数据总线 Q[15:0]和输入数据总线 D[15:0] 连接在一起，可将 FIFO 数据总线设置为双向。该示例通过 USB BULK 传输对 FIFO 数据进行读写。使用 **SuiteUSB 3.4 - USB Development tools for Visual Studio** 所提供的 USB 控制中心，可以执行这些传输。

Table 1 对 GPIF 互联进行了详细说明。指定 CTL 和 RDY 引脚与最小 FX2LP 封装 (包含 56 个引脚) 相兼容。

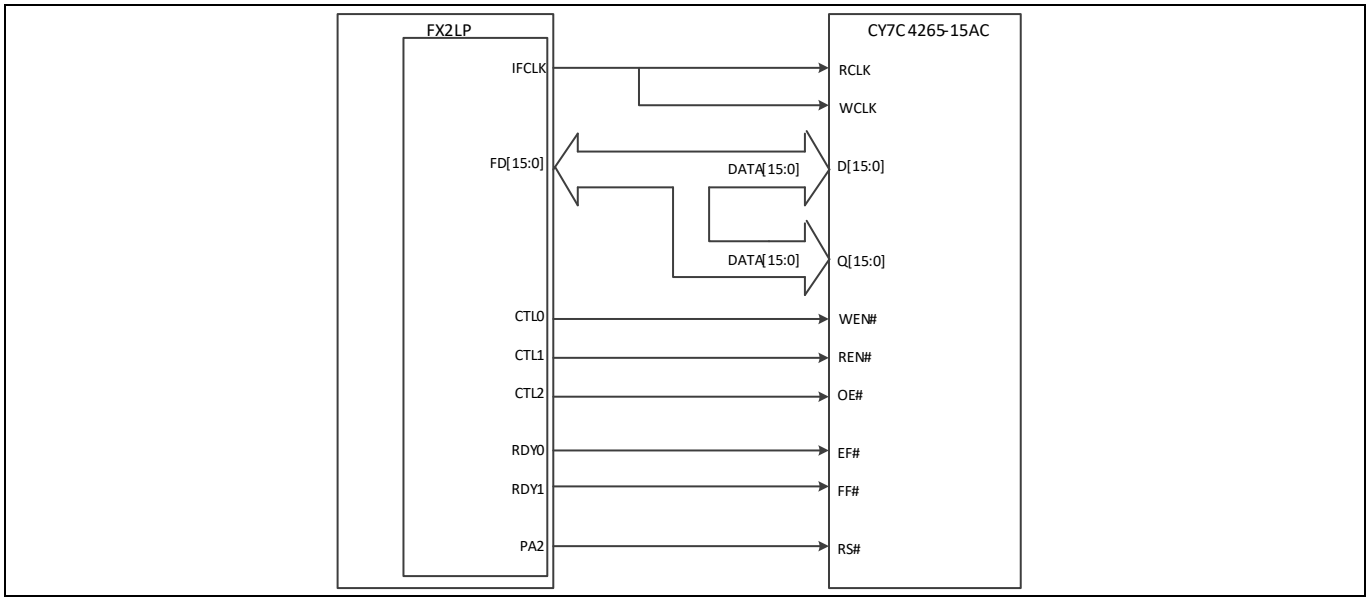


Figure 32 GPIF 与外部同步 FIFO 间的连接

Table 1 分配至 CY7C4265-15AC 信号的 FX2LP GPIF 信号

FX2LP GPIF 信号	CY7C4265-15AXC 信号	说明
IFCLK	WCLK, RCLK	IFCLK 连接到外部 FIFO 的读写时钟输入 (WCLK、RCLK)。确认 WEN#时，数据在每个 WCLK 上升沿上为外部 FIFO 提供时钟脉冲。同样，确认 REN#和 OE#时，FIFO 在 RCLK 每个上升沿上的 Q[15:0] 显示数据。外部 FIFO 可接受最大为 66.7 MHz 的输入时钟频率，因此，能够处理的输入 IFCLK 频率为 30 MHz 或 48 MHz。请注意井字符 (例如 OE#) 表示低电平有效。
FD[15:0]	D[15:0], Q[15:0]	GPIF 数据总线 (FD[15:0]) 连接至外部 FIFO 的输入数据总线 (D[15:0])，以进行字操作。外部 FIFO 的输出数据总线 Q[15:0]也被连接至 GPIF 数据总线，以便 FX2LP 可以读回数据内容。由于两个单向 FIFO 数据总线彼此连接，因此 GPIF 必须控制 OE#信号，以避免发生总线冲突。特别是当 FX2LP 驱动数据总线时，不能将 OE# 置于低电平状态 — 这是因为 FX2LP 和 FIFO 同时驱动数据总线时，会导致总线冲突。
CTL0	WEN#	CTL0 连接至外部 FIFO 写使能引脚 WEN#。GPIF 将 WEN#置于低电平时，在每个 WCLK 上升沿上将数据写入到外部 FIFO 内。
CTL1	REN#	CTL1 连接至外部 FIFO 读使能引脚 REN#。当 GPIF 将 REN#和 OE# 置于低电平时，FIFO 在 RCLK 的每个上升沿上驱动 Q[15:0]中的数据。

示例 3：使用单个字的读/写数据操作

FX2LP GPIF 信号	CY7C4265-15AXC 信号	说明
CTL2	OE#	CTL2 连接至外部 FIFO 输出使能引脚 OE#。将 REN#和 OE#置于低电平时，FIFO 在 RCLK 的每个上升沿上驱动 Q[15:0]上的数据。
RDY0	EF#	RDY0 连接至外部 FIFO EMPTY 标志 EF#，如果 FIFO 为空，它将置于低电平。由于能够在 GPIF 转换状态中检测 READY 信号，因此从外部 FIFO 读取数据时，GPIF 可以使用该信号进行控制数据传输。
RDY1	FF#	RDY1 连接至外部 FIFO FULL 标志 FF#，如果 FIFO 已满，它将置于低电平。当写入外部 FIFO 时，GPIF 可使用该信号控制数据传输。
PA2	RS#	PA2 连接到外部 FIFO RESET 引脚。PA2 是一个 FX2LP GPIO 引脚，并且不属于 GPIF 逻辑。在启动 GPIF 数据传输前，8051 代码会使用 PA2 将外部 FIFO 复位到一个已知状态。

8 USB 数据流

USB 端点 2 OUT (EP2OUT)作为 GPIF 写入外部 FIFO 的发送端点使用，端点 6 IN (EP6IN) 作为 GPIT 从外部 FIFO 读取的接收端点使用。从主机端的角度来看，USB IN 和 OUT 方向如下：

- EP2OUT 包含了 USB 主机 (PC) 发送和 FX2LP 接收的数据包。
- EP6IN 包含了 FX2LP 发送和 PC 接收的数据包。

9 设计 GPIF 互联

通过使用第一个示例中所描述的 GPIF 各步骤可以设置 GPIF 互联，具体如下。

1. 启动赛普拉斯 GPIF Designer 工具。
2. 请依次选择 **File** > **New**，然后选择 FX2LP 开发板使用的 **CY FX2 (128 pin)**。
3. 右击外部器件模块中的 **Un-named** 标签并将其重新命名为 **CY7C4265-15AC**。
4. 默认情况下，数据总线被配置为 16 位。该示例使用了 16 位数据总线。
5. 右击 **ADR** 标签。该设计没有使用地址行，因此可禁用它们以简化框图。
6. 右击 **RDY** 标签，并对 RDY 信号进行配置，如 **Figure 33** 中所示。

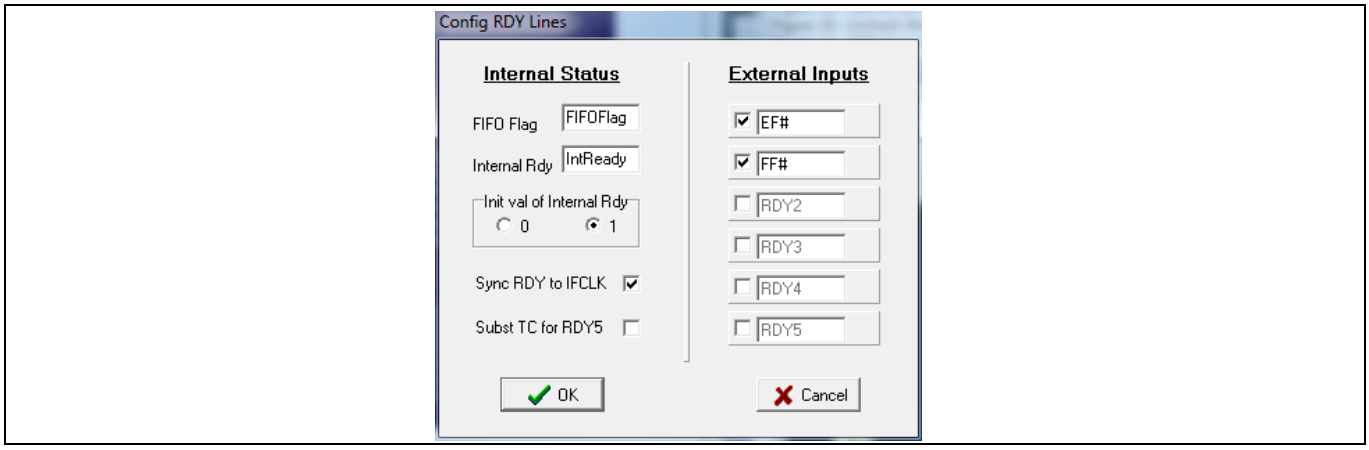


Figure 33 RDY 引脚配置

7. 右击 CTL 标签，并对 CTL 信号进行配置，如 **Figure 34** 所示。

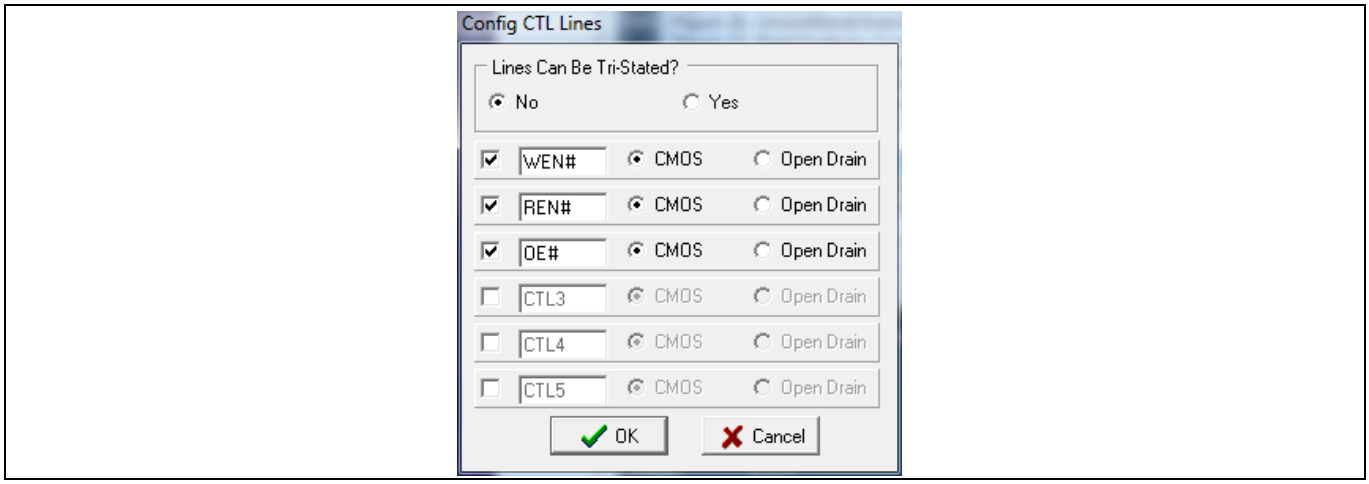


Figure 34 CTL 引脚配置

8. 右击 **48 MHz CLK** 文本框，并按照 **Figure 35** 的内容配置时钟属性。选择 **Invert Clock** 复选框。GPIF 更改并采样时钟上升沿上的信号；FIFO 将更改并采样时钟下降沿上的信号。这个半时钟偏移提供了各种接口时序设置和保持时间。

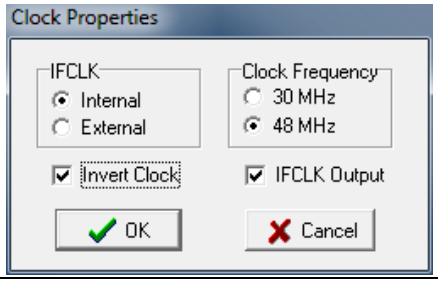


Figure 35　配置 IFCLK

该框图将如 **Figure 36** 所示。现在，该时钟被标签为“nClk 48 Mhz”，其中“n”表示时钟反转。

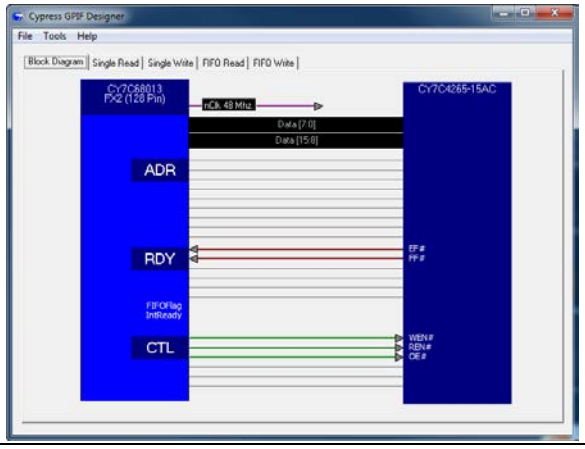


Figure 36　FIFO 的配置接口

9.1　单字写波形

通过写波形，可以将数据从 FX2LP OUT 端点 FIFO 传输到外部 FIFO。右击 FIFO Read 选项卡并将其重新命名为“Unused”。对 FIFO Write 选项卡进行类似操作。您的屏幕将如 **Figure 37** 所示。

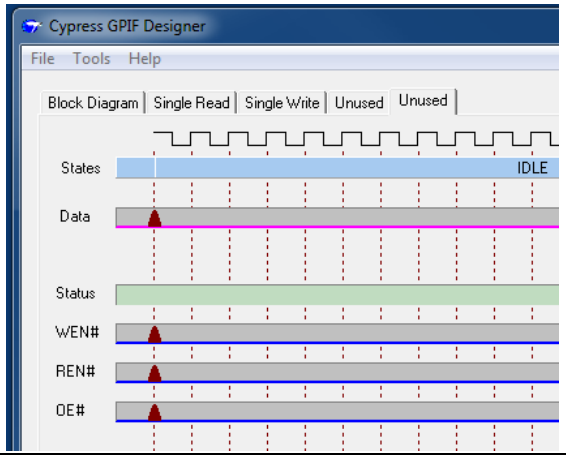


Figure 37　波形屏幕

请依次点击 **Tools** > **Map Waveforms** 进行 WFSELECT (选择波形) (**Figure 38**)。

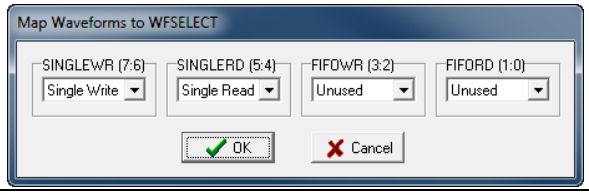


Figure 38　波形映射对话框

请确保单写波形被映射到 SINGLEWR，而单读波形被映射到 SINGLERD。这样可对 8051 寄存器进行配对 (这些寄存器将这些波形设置为 GPIF 命名的波形)。

Note:　*波形映射允许您为同一类传输定义多个波形组。例如，您可以将两种不同的波形类型使用在单写操作中。*

通过写波形可以管理各字节传输，这些字节传输将 FX2LP OUT 端点的数据传输到外部 FIFO。要想创建 FIFO 写波形，请先检查 CY7C4265 数据手册中提供的 CY7C4265 FIFO 的写周期时序。**Figure 39** 显示的是写周期时序，其中 GPIF 状态被添加到该框图的底部。

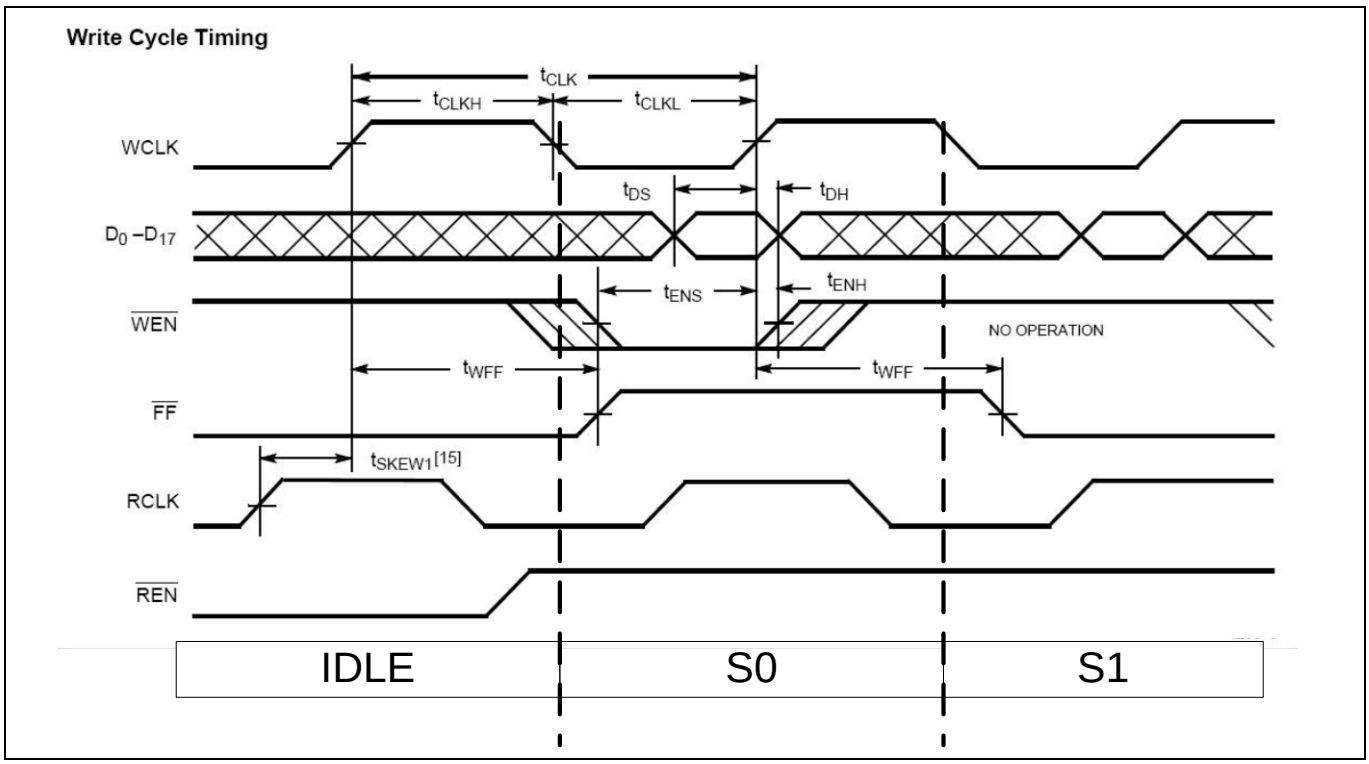


Figure 39　写周期时序框图

设计 GPIF 互联

在写周期时序框图中，您可以创建如 **Figure 40**。

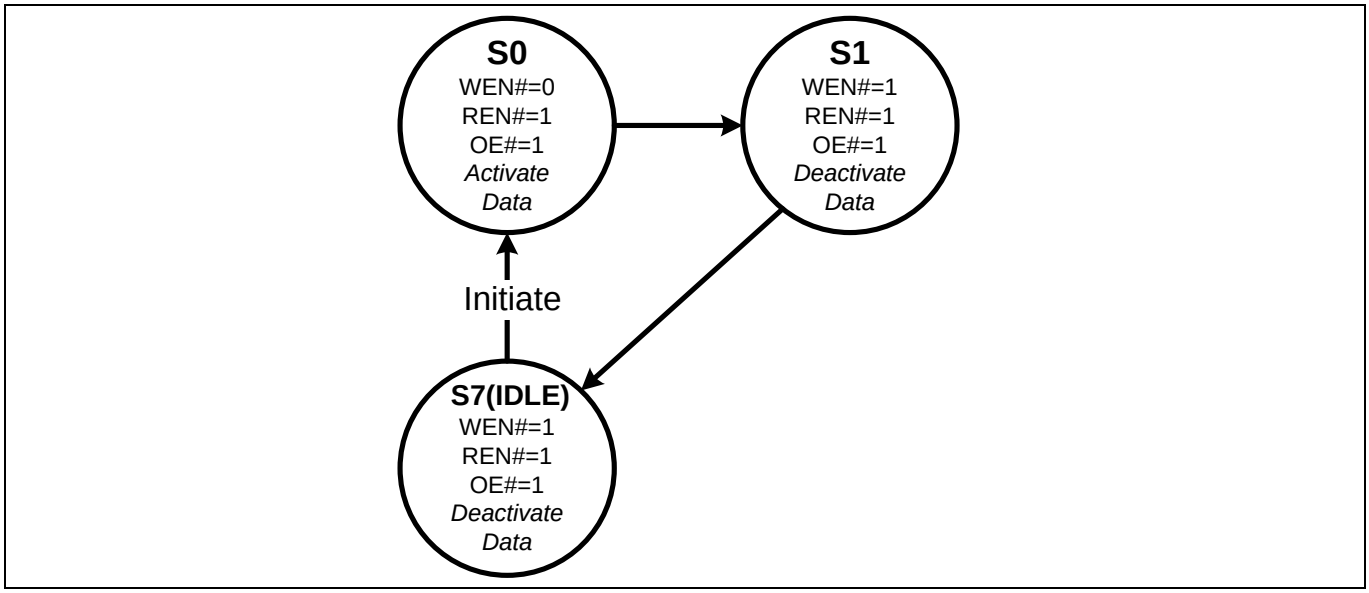


Figure 40　单字 FIFO 写状态框图

- 对于单写波形，通过将 WEN#逻辑置于低电平和驱动数据总线，可以在状态 S0 期间将数据写入到外部 FIFO 内。
- 在状态 S1 内，WEN#被禁用 (为高电平)，并且 GPIF 数据总线停止驱动 (悬空)。S1 是一个决策点状态，它会强制无条件转换到闲置状态，从而停止波形。闲置状态下并没发生任何活动
- 每次触发单写波形时，GPIF 引擎都会经过 S0、S1，然后停止于 S7 (空闲)。

请按照下列各步骤完成单写波形。

1. 点击 **Single Write** 波形选项卡。
2. 点击离左边界一个时钟周期的 WEN#跟踪，在此处放置一个操作点并创建 WEN#波形。自动生成状态 0 (s0)，并会持续一个 IFCLK 周期 (20.83 ns)。
3. 由于 GPIF 被写入到外部 FIFO，因此在波形期间，必须将 REN#和 OE#置于高电平，为防止外部 FIFO 驱动它的数据总线。为此，需要右击 OE#和 REN#跟踪上的操作点，并选择 **High (1)**。各波形如 **Figure 41** 所示。

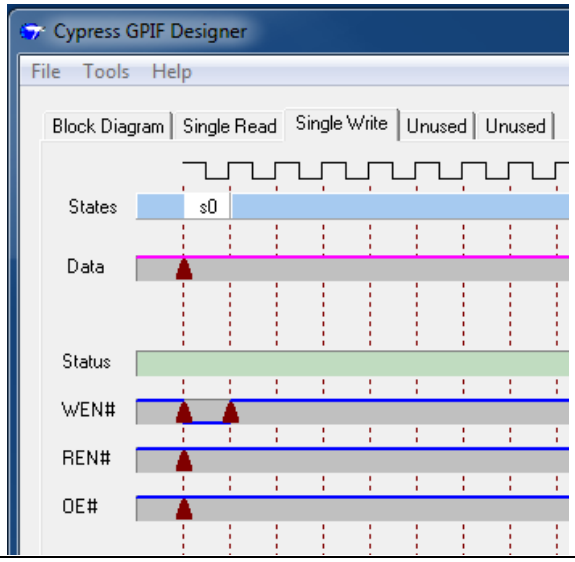


Figure 41　WEN#在一个时钟内保持低脉冲

4. 必须将数据总线驱动到 s0 状态。因此，需要右击数据操作点，并选择 **Activate Data**。

   应该只在一个时钟周期内驱动数据总线。经过一个时钟周期后，要想停止驱动数据，需要在数据带中距离第一个时钟一个时钟的位置上放置另一个操作点。请注意，现在数据带只在 s0 期间保持为高电平。各波形如 **Figure 42** 中所示。

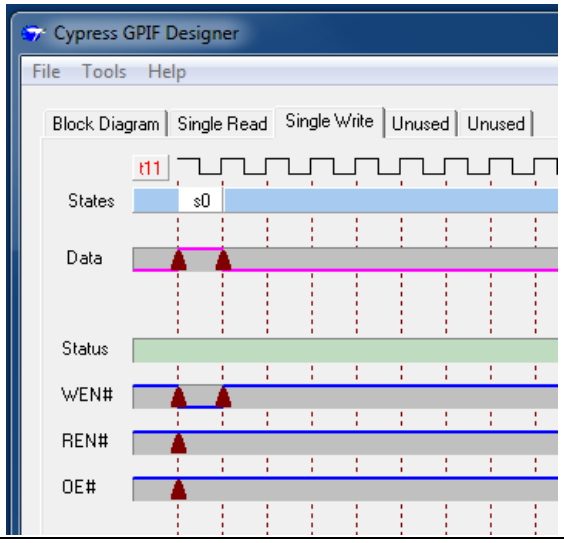


Figure 42　S0 期间，会在一个时钟周期内驱动数据

5. 通过创建一个决策点状态 S1 可以返回闲置状态。在 S0 状态的右边界上点击状态带，然后按照 **Figure 43** 中显示的内容设置无条件转换。

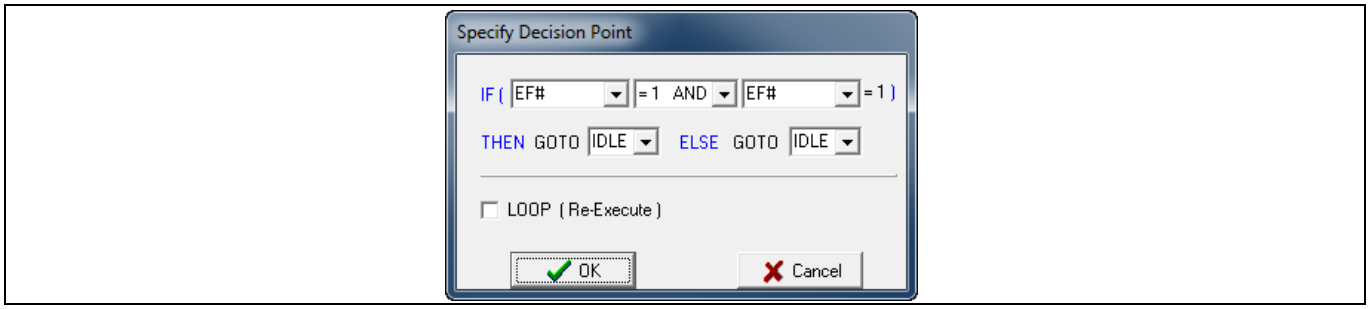


Figure 43　无条件转换到闲置状态

单个字写波形如 **Figure 44** 所示。

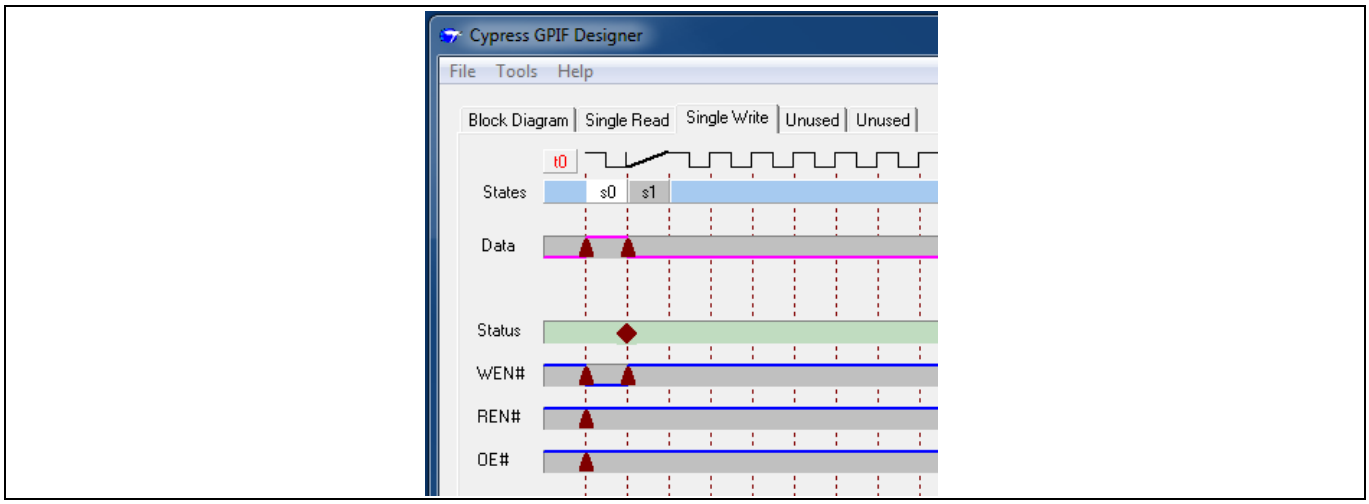


Figure 44　单个字写波形

9.2 单个字读波形

通过读波形可以管理从外部 FIFO 到 FX2LP IN 端点将 FIFO 的字节传输。类似于写周期，GPIF 波形必须满足外部 FIFO 时序的要求。首先，需要检查外部 FIFO 的读周期时序，然后创建单读状态机。**Figure 45** 显示的是 CY7C4265 的读周期时序，其中 GPIF 状态被添加到该框图的底部。

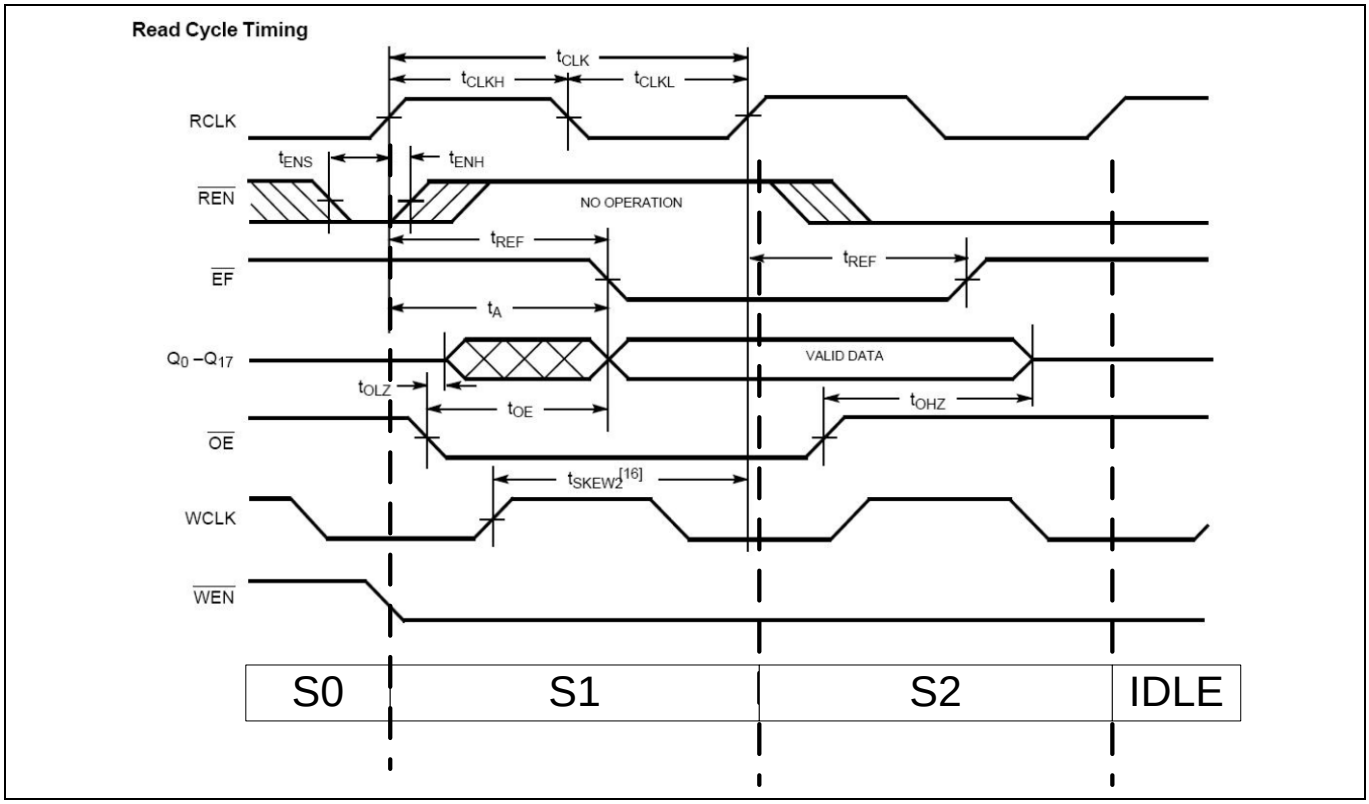


Figure 45 读周期时序框图

通过时序信息您可以创建如 **Figure 46** 所示的状态机，以用于字节读操作。

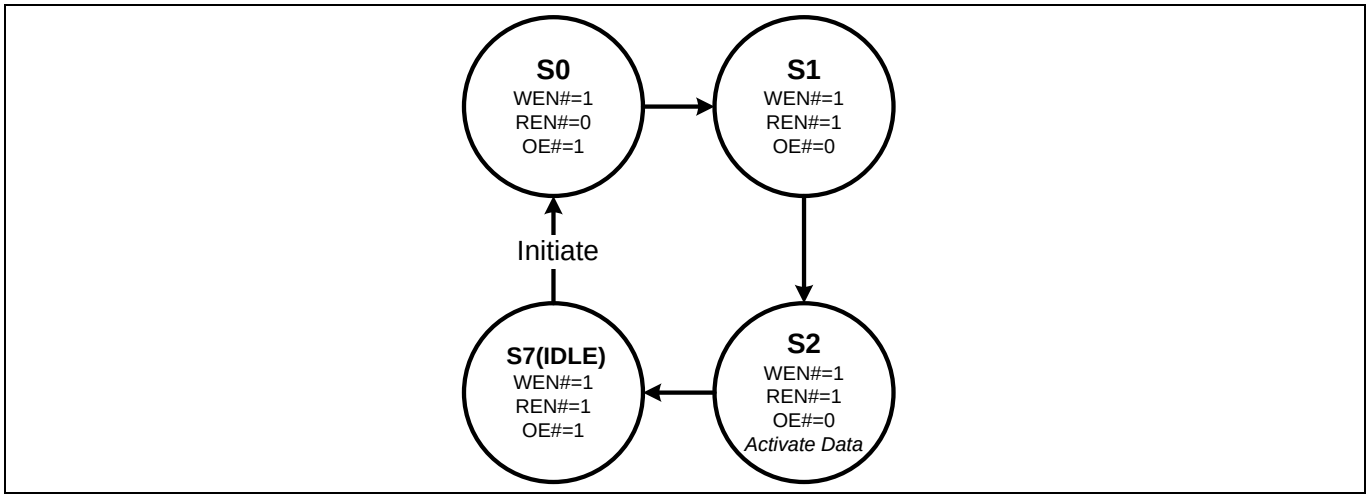


Figure 46 字节读状态框图

设计 GPIF 互联

- 对于单读波形，在 S0 状态期间，REN#在一个 IFCLK 周期内保持逻辑低电平状态。这样会在确认 OE# 前，外部 FIFO 需要记录在数据手册中的 t_{ENS} 设置时间。经过一个时钟后取消确认 REN#可确保 FIFO 未递增一个字，以执行读操作。
- 然后 S1 将确认 OE#并转到 S2 状态。
- 由于它需要进行反转，因此 S2 作为决策点状态。S1 状态开始阶段，仍无法从外部 FIFO 获得数据，因此，GPIF 只能在 S2 开始时“采样”数据。
- 每次触发单读波形时，GPIF 引擎都会按 S0、S1、S2 和 S7 的顺序循环。

请按照下列各步骤创建单读波形。

1. 点击 **Single Read** 波形选项卡。
2. 右击 REN#跟踪中的第一个时钟，并选择 **Low (0)**。点击离左边界一个时钟周期的 REN#跟踪，在此处放置一个操作点并创建 REN#脉冲。自动生成状态 0 (s0)，并会持续一个 IFCLK 周期 (20.83 ns)。因此，REN#在 20.83ns 时间内有效 (**Figure 47**)。

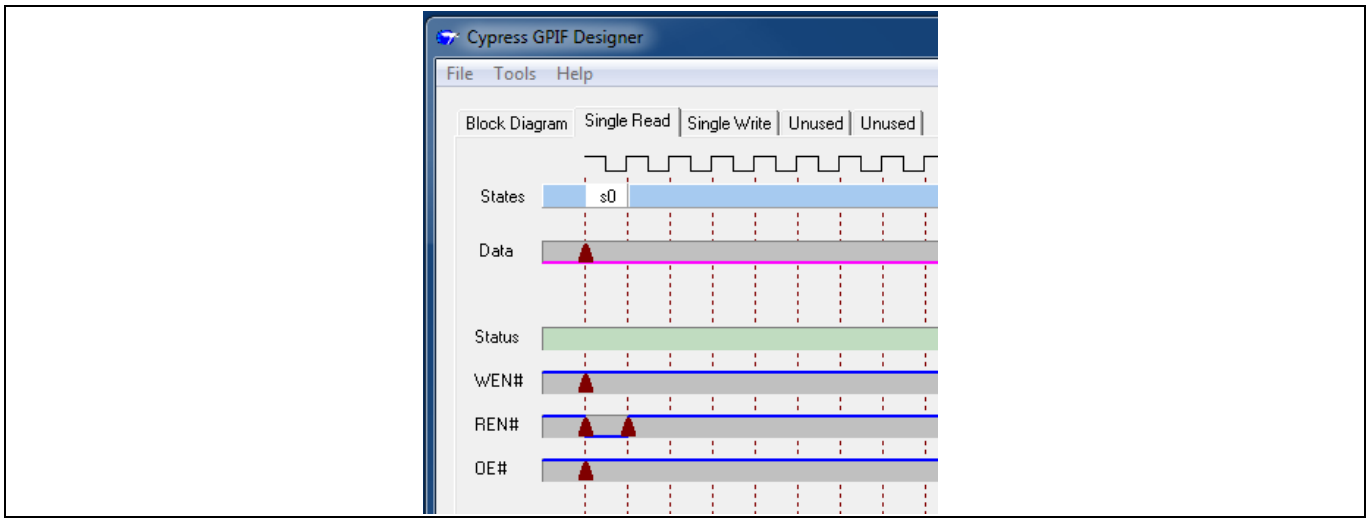


Figure 47　REN#在一个时钟内有效

3. 在 OE#带内，将一个操作点放置在 S0 的右边界上，并在离它后面一个时钟的位置上放置另一个操作点。这样，在完成状态 S0 后，可自动取消确认 OE#并创建持续一个 IFCLK 周期的 S1 (20.83 ns) 状态。

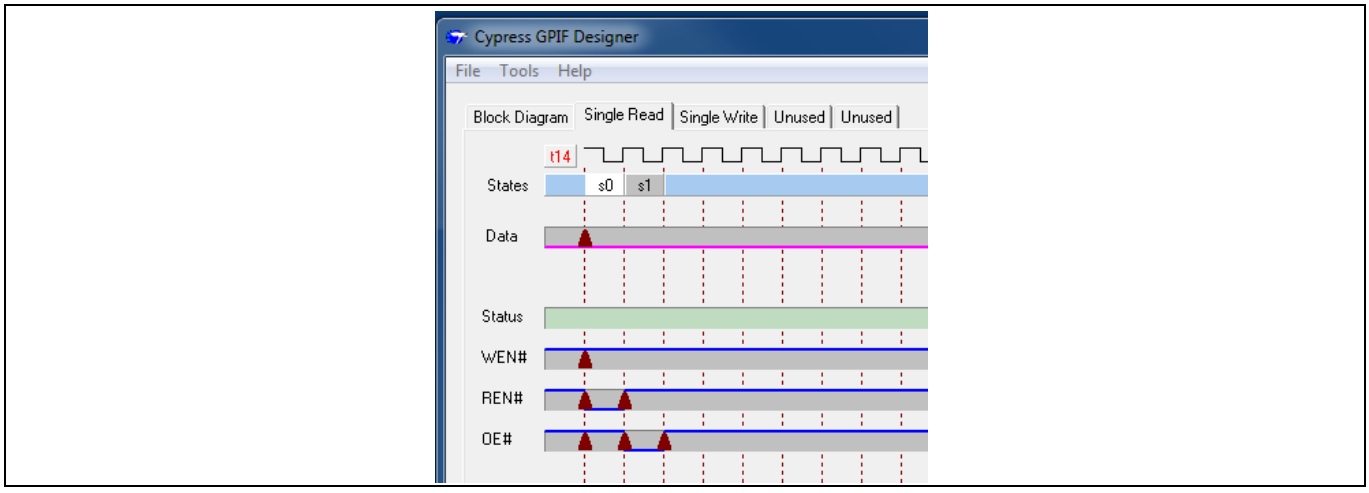


Figure 48　OE#在一个时钟内有效

4. 通过点击时钟终端 s1 上的状态带来添加一个决策点 (DP) 状态。从而创建状态 S2，并会弹出“Specify Decision Point”对话框。按照 **Figure 49** 配置该决策点，为了无条件转换到闲置状态。这时，波形如 **Figure 50** 所示。

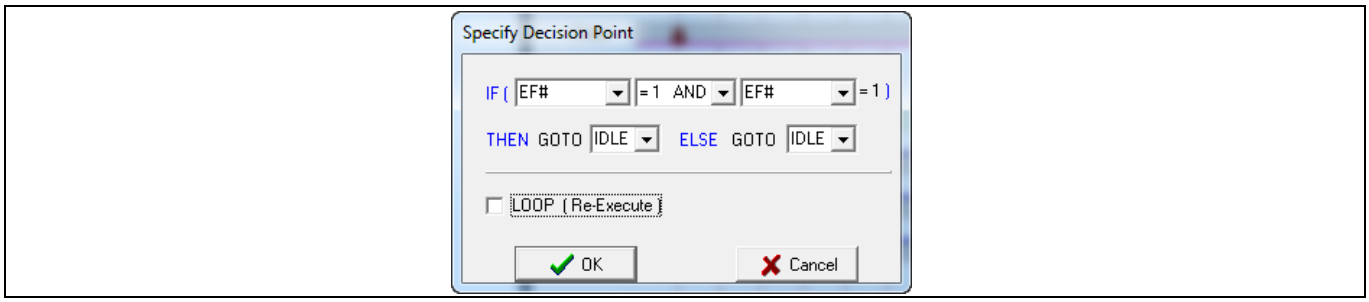


Figure 49　无条件转换到闲置状态

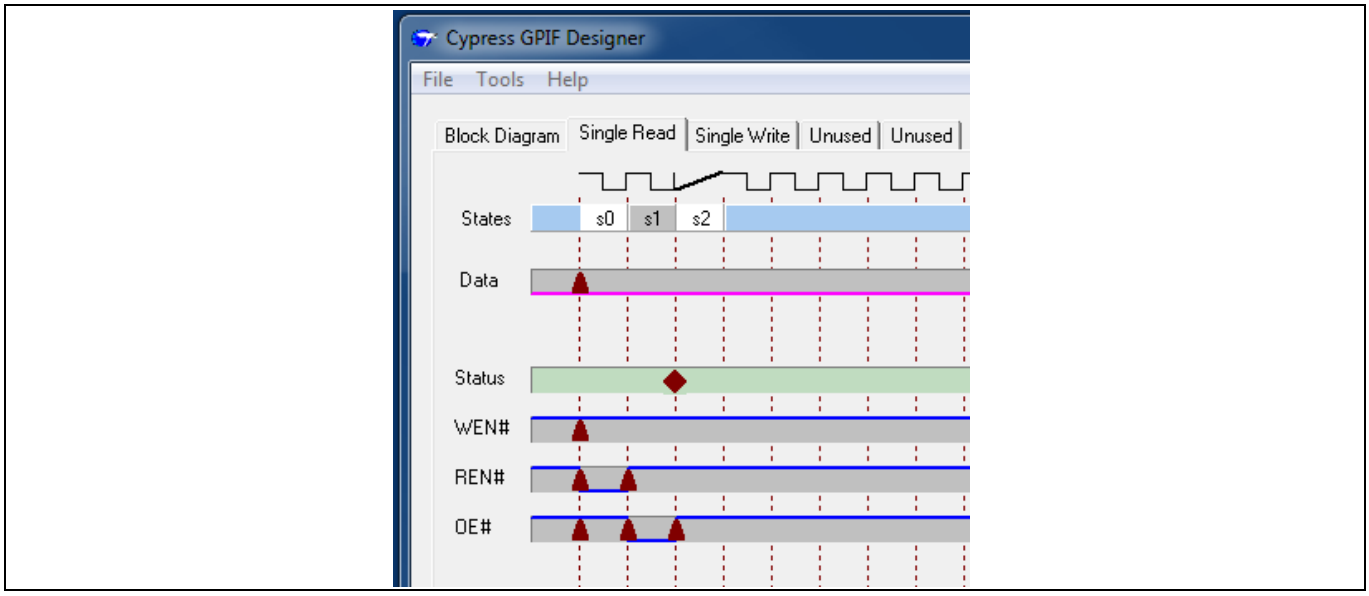


Figure 50　添加了状态 s2

5. 最后需要进行的调整。必须在 S2 期间采样 (读) 数据。要想采样该数据，需要将操作点放置在 S2 状态的开始和结束的数据带上。请注意，由于当前是在 Single Read (单读) 选项卡中进行操作，因此高电平对应于采样，而不是驱动数据带。最后，单个字读波形如 **Figure 51** 所示。
6. 通常，您应该将 GPIF Designer 项目文件 (*.gpf) 和它生成的 C 文件存储在您的 Keil 项目文件夹内。在这个范例中，它们已经作为本应用笔记的代码显示在该文件夹内。

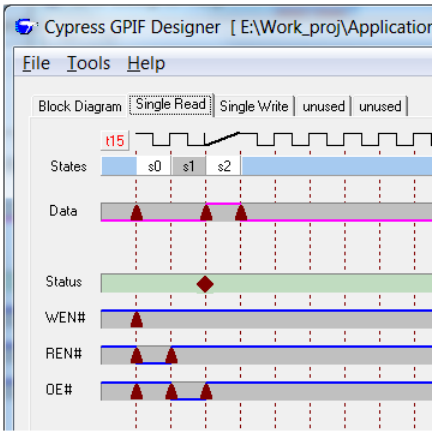


Figure 51 最后单个字读波形

9.3 单个字操作的固件编程

在 GPIF Designer 中实现单字操作波形后，需要将 USB 固件与 GPIF Designer 输出集成在一起。这样可以通过 USB 对外部 FIFO 进行读写操作。开始对现有的固件框架项目进行操作并且添加 GPIF 管理代码。双击 Keil 项目文件 *FX2_to_extsyncFIFO.uv2*。该文件打开时，将出现 Files (文件) 面板，如 **Figure 52**。**Table 2** 介绍了这些文件。

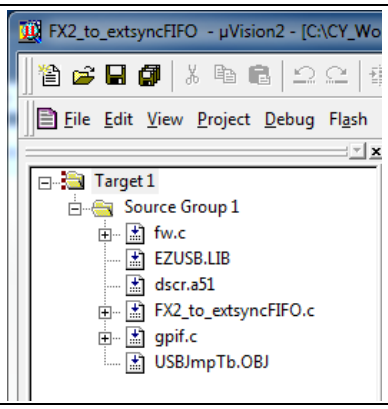


Figure 52 “FX2_to_extsyncFIFO”项目文件

已经将 *periph.c* 重命名为 *FX2_to_extsyncFIFO.c*。在该文件内，将 USB 端点和 GPIF 初始化代码添加到 TD_Init()函数内，并将 GPIF 管理代码添加到 TD_Poll()函数内。

设计 GPIF 互联

Table 2 项目文件说明

文件	说明
fw.c	固件框架用于处理 USB 请求，并调用任务调度程序 TD_Poll()。
Ezusb.lib	处理暂停、恢复、I²C 等操作的函数集。
USBJmpTb.OBJ	USB (INT2)和 GPIF/从设备 FIFO (INT4)中断源的中断矢量跳转表。
Dscr.a51	将 EP2OUT 和 EP6IN 作为 FX2LP 器件可用端点报告的 FIFO 示例器件描述符表。
FX2_to_extsyncFIFO.c (由 periph.c 重命名得到)	可在这里找到 TD_Poll()和 TD_Init()的主用户应用代码。您需要修改该文件，并不更新 *fw.c*。
Gpif.c	包含了执行单/FIFO GPIF 操作波形特性的 GPIF 波形描述符表的文件。它是从 GPIF Designer 工具导出的 C 文件。

9.4 代码段

9.4.1 TD_Init()

TD_INIT()进行下列操作：

- 将 CPU 时钟速度切换到 48 MHz (加电状态时默认速度为 12 MHz)
- 将 EP2 配置为批量 OUT 端点，大小为 512 缓存的 4 倍
- 将 EP6 配置为批量 IN 端点，大小同样为 512 缓存的 4 倍
- 将 FIFO 配置为手动模式的字操作
- 使用 FIFORESET 寄存器复位这些端点，并使 EP2 OUT 端点确保可以接收 USB 主机 (PC) 中的数据

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

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

    EP2FIFOCFG = 0x01; // manual mode, disable PKTEND zero length send,
  word ops
    SYNCDELAY;
    EP6FIFOCFG = 0x01; // manual mode, disable PKTEND zero length send,
  word ops
    SYNCDELAY;

    FIFORESET = 0x80;  // set NAKALL bit to NAK all transfers from host
    SYNCDELAY;
```

设计 GPIF 互联

```
    FIFORESET = 0x02;  // reset EP2 FIFO
    SYNCDELAY;
    FIFORESET = 0x06;  // reset EP6 FIFO
    SYNCDELAY;
    FIFORESET = 0x00;  // clear NAKALL bit to resume normal operation
    SYNCDELAY;

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

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

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

- 然后，TD_Init 将调用位于 *gpif.c* 中的函数 GPIFInit()。
- GPIFInit()将 GPIF 波形描述符表加载到片上存储器内，并对其他 GPIF 寄存器进行配置。
- 每次只能加载四种波形。如果描述物理接口操作所需的波形超过了四种，则必须手动加载另外一套四种波形。
- IFCONFIG 寄存器通过定义物理接口的 GpifInit()函数被配置。
- TD_Init()通过脉冲 PA2 (RS#)复位外部 FIFO，如下面代码段中所示。这样可确保在启动数据操作前已经初始化外部 FIFO。

```
// reset the external FIFO

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

- 在下面的代码中定义了 USB 供应商指令 0xB2。它允许主 PC 通过发出供应商指令复位外部 FIFO。

```
BOOL DR_VendorCmnd(void)
{
switch (SETUPDAT[1])
  {
case VX_B2:
    {
// reset the external FIFO

      OEA |= 0x04;     // turn on PA2 as output pin
```

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

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

9.4.2 触发 GPIF 单个字写操作

- 8051 代码通过访问 XGPIFSGLDATH、XGPIFSGLDATLX 和 XGPIFSGLDATLNOX 寄存器触发单个字读/单个字写 GPIF 波形。从而启动数据传输。
- 为了触发 GPIF 单个字写操作，您需通过以下方式写入 XGPIFSGLDATH 和 XGPIFSGLDATLX：

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

- 这样可以将单字值 MSB 和 LSB 设置为可传输的，同时，对 XGPIFSGLDATLX 寄存器进行的写操作可触发单个字写操作。
- 在本示例中，该操作是在 GPIF_SingleWordWrite()函数内执行的。该函数把一个字的值作为输入参数使用，并触发 GPIF 单字写操作。

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

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

- 该函数在启动操作前通过轮询位 GPIFTRIG.7 检查 GPIF 是否处于 IDLE (闲置) 状态 (如果 GPIF 状态机为闲置状态，将设置该位)。在启动任何一个 GPIF 操作前，都必须始终确保 GPIF 处于闲置状态。
- 请注意单个操作寄存器的访问顺序，因为端点缓冲区是以 FIFO 形式进行组织的。这样的顺序可确保端点缓冲区的第一个字节在 FD[7:0]被写出，第二个字节在 FD[15:8] (低位优先格式) 被写出。

9.4.3 GPIF 单字读操作

- 通过从 XGPIFSGLDATX 寄存器执行虚拟读，可以触发 GPIF 单个字读操作。该读操作并没有进行任何数据传输，它仅启动了 GPIF 波形。检测 GPIF DONE 位后，8051 将读取 XGPIFSGLDATH 和 XGPIFSGLDATLNOX 寄存器中的字量。
- 在本示例中，此操作是在 GPIF_SingleWordRead()函数内执行的。该函数把目标变量的字指针作为参数使用，并执行 GPIF 单字读操作：

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

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

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

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

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

- 该函数会先确保 GPIF 处于闲置状态，然后才从 XGPIFSGLDATLX 执行虚拟读，以触发 GPIF 单字读操作。然后，在读取包含了字值的寄存器前，它会等待完成 GPIF 的执行。

9.4.4 TD_Poll()

- 位于函数 TD_Poll() 内的主应用代码会在器件操作期间被连续调用。
- 在此函数中，要调用 GPIF_SingleWordWrite()和 GPIF_SingleWordRead() 函数。
- GPIF_SingleWordWrite() 将 EP2OUT 中的数据发送到外部 FIFO，而 GPIF_SingleWordRead()会读取外部 FIFO 中的数据，并将结果写入到 EP6IN 内。
- 处理 USB OUT 传输的代码如下：

```
if(!(EP2468STAT & bmEP2EMPTY) && (EXTFIFONOTFULL))
  {
// if host sent data to EP2OUT AND external FIFO is not full,

    Tcount = (EP2BCH << 8) + EP2BCL; // load transaction count with EP2
byte count
    Tcount /= 2;                     // divide by 2 for word wide
transaction
    Source = (WORD *)(&EP2FIFOBUF);
for( i = 0x0000; i < Tcount; i++ )
    {
```

设计 GPIF 互联

```
// transfer data from EP2OUT buffer to external FIFO
      GPIF_SingleWordWrite (*Source);
      Source++;
    }
    EP2BCL = 0x80; // re-arm EP2OUT
  }
```

- 检查 EP2468STAT 寄存器中的 EP2 空标志，以确定 EP2OUT 端点中的 USB 主机是否存在数据。
- 另外，通过访问 GPIFREADYSTAT 寄存器，可以检查外部 FIFO 中的 FF#标志。这样可以确保外部 FIFO 中有足够的空间用于存储从 EP2OUT 传输来的数据。8051 通过访问 GPIFREADYSTAT 寄存器检查 GPIF RDY 信号的状态。EXTFIFONOTFULL 是 GPIFREADYSTAT 的宏，它同 bmBIT1 进行 AND 运算。
- 如果 EP2OUT 端点中有数据，并且外部 FIFO 未满，那么字变量 Tcount 将使用计数值初始化。通过访问 EP2BCH/L 寄存器，可以查看从主机传输到 EP2OUT 的字节数量。每个 GPIF 单字写操作都会将一个字发送至外部 FIFO，因此，操作数始终是端点缓冲区实际包含字节数的一半。
- 然后，“For loop”语句以“Tcount”的次数调用 GPIF_SingleWordWrite 函数，并索引端点缓冲区 EP2 值，这样是为每次能以一字的方式向外部 FIFO 发送数据。每个循环都会触发一个 GPIF 单写操作，因此每次将一个字的数据发送到外部 FIFO。
- 传输“Tcount”字的数据后，将重新激活 EP2 端点，以便可以接受主机中的下个 USB 数据包。FX2LP 会自动否认所有 OUT 数据包，直到可以接收新的数据包为止。因此，主机需要连续重试 OUT 传输。
- 处理 USB IN 传输的代码如下：

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

      Destination = (WORD *)(&EP6FIFOBUF);
for( i = 0x0000; i < Tcount; i++ )
      {
// transfer data from external FIFO to EP6IN buffer
        GPIF_SingleWordRead (Destination);
        Destination++;
      }
        Tcount *= 2;          // multiply by 2 to obtain byte count value
      EP6BCH = MSB(Tcount);
      SYNCDELAY;
      EP6BCL = LSB(Tcount); // arm EP6IN to send data to the host
      SYNCDELAY;
    }
  }
```

- 如果 in_enable 标志为真，则需要确保 EP6IN 端点缓冲区未满，以及外部 FIFO 非空 (EXTFIFONOTEMPTY 为 GPIFREADYSTAT 和 bmBIT0 的宏)。
- 如果 EP6IN 端点缓冲区未满，并且外部 FIFO 非空，则“for loop”将调用 GPIF_SingleWordRead()函数。从而触发一个 GPIF 字读取，并将该结果存储在目标地址中 (该地址被设置为 EP6IN FIFO)。每个重复次数都会递增目标地址，从而将在外部 FIFO 中检索到的字节填充到 EP6IN FIFO 内。
- 将外部 FIFO 数据填充到 EP6IN FIFO 后，8051 代码会通过使能 EP6IN 端点将数据传输到主机 PC。使能前，FX2LP USB 逻辑会自动否认 EP6 的所有主机 IN 请求。8051 代码通过写入一个字节数 (表示主机 IN

传输的字节数量) 使能 IN 端点。由于每个 GPIF 单字读操作接收外部 FIFO 中包含的两字节的整个字，因此，发送至主机的字节数始终是 GPIF 操作数的两倍。

- 通过将相应的值分配给“in_enable”标志，可以使能或禁用 IN 传输。本示例定义了两条供应商指令，一个使能 IN 传输，另一个禁用 IN 传输，如下面代码所示：

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

- IN 供应商指令 0xB3 通过将 in_enable 设置为 TRUE (真) 来使能 IN 传输。
- IN 供应商指令 0xB4 通过将 in_enable 设置为 FALSE (假) 来禁用 IN 传输。
- in_enable 的默认值为 FALSE。
- 通过 in_enable 标志您可以对每一次读写操作单独进行测试。如果它始终被使能，那么将立即在 OUT 传输代码后处理 IN 传输代码。通过单步处理该代码，您可轻松捕获使用逻辑分析仪的每个读/写操作 (适用于调试目的)。

9.5 运行 GPIF 单个字操作示例

9.5.1 不存在外部 FIFO

如果您尚未构建包含外部 FIFO 的 FX2LP 开发板，您同样能够使用示波器来观察 GPIF *写*传输。这样可以验证将数据从 PC 传输到 FX2LP 芯片的 USB 代码，以及将数据输出到外部 FIFO 的 GPIF 波形。

1. 通过探针 FX2LP 开发板上的 P2-11 观察 CTL0=WEN#，并且通过探针 P1-19 观察 FIFO 数据总线 D0。
2. 启动 USB 控制中心。将 FX2LP 开发板插入到主 PC 端口内，该端口应该作为 USB 加载程序器件 (如 **Figure 21** 所示) 进行枚举。否则，按照进入 **Figure 21** 的各步骤进行操作。
3. 请依次选择 **Program FX2**>**RAM** 并找到 *FX2_to_extsyncFIFO.hex* 文件以加载它。
4. 展开 Bulk loop 器件并选择 **Bulk out endpoint (0x02)**。
5. 在 **Data to send (Hex)** 文本框内输入 **Figure 53** 中显示的数字。
6. 点击 **Transfer Data-OUT** 按键。

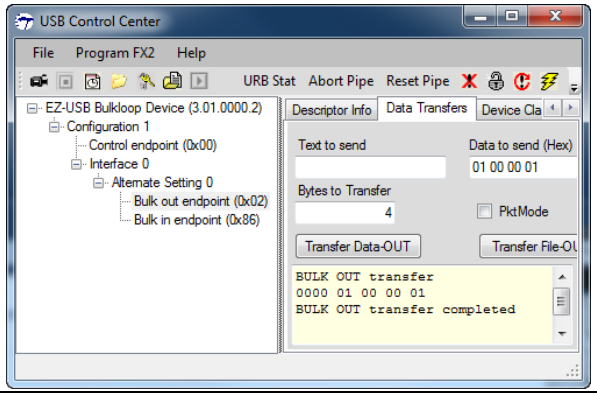


Figure 53 USB 控制中心传输数据

7. 右下角中的文本窗口可确认该传输，并且应该在 WEN#脉冲的下降沿上触发该示波器(**Figure 54**)。

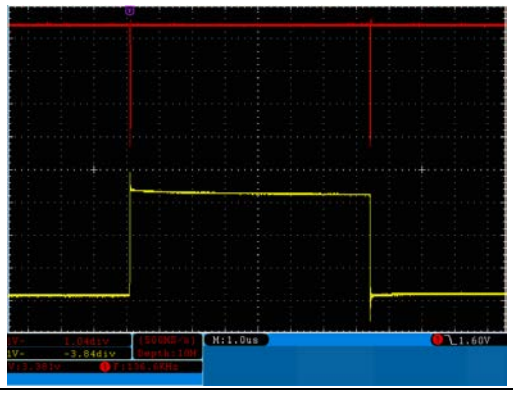


Figure 54 顶部：WEN#，底部：FD[0]

请注意，第一个 FIFO 写传输会将 FD[0]置于高电平，第二个写传输会将 FD[0]置于低电平。这样可以确保 16 位字按从低到高的顺序输出；第一字为 0001，第二字为 0100。

9.5.2 存在外部 FIFO

如果您已经编译好并将外部 FIFO 添加到开发板上，请先通过传输 USB 数据 OUT 测试该环回，如前面所述。然后，使用同一个字节数进行传输数据 IN 操作。本测试使用的是包含 512 个数据字节的 *512_count.hex* 文件。

1. 在树状图中选择 **Bulk out endpoint (0x02)** (批量输出端点 (0x02))。单击 **Data Transfers** tab (数据传输) 选项卡。按下 **Transfer File-OUT** 按键，并选择 Keil 项目文件夹：`FX2LP Source code and GPIF project files\Firmware\FX2_to_extsyncFIFO GPIF Single Transactions` 中的 *512_count.hex* 文件。点击 **Open**，将 512 个字节发送到外部 FIFO。
2. 要想使用 IN 传输从外部 FIFO 中读回 512 个字节，必须使用一个 USB 供应商请求将 *in_enable* 标志设置为真。在树状图中选择 **Control endpoint (0x00)**；在 **Req code** 字段中输入 ‘0xB3’ 。将 **Req Type** 设置为 ‘Vendor’ ，将 **Direction** 设置为 ‘In’ ，并将 **Bytes to Transfer** 设置为 ‘1’ 。点击 **Transfer Data**。
3. 在树状图中选择 **Bulk in endpoint (0x86)** (批量输入端点 (0x86))。请确保 **Bytes to Transfer** 中的字节数量为 512。按下 **Transfer Data-IN** 按键。现在您可以看到从设备 FIFO 回读的 512 个字节。

9.6 单个字操作的逻辑分析仪波形

本节显示了 GPIF 引擎为 GPIF Designer 定义的波形生成的时序。

9.6.1 单个字写波形

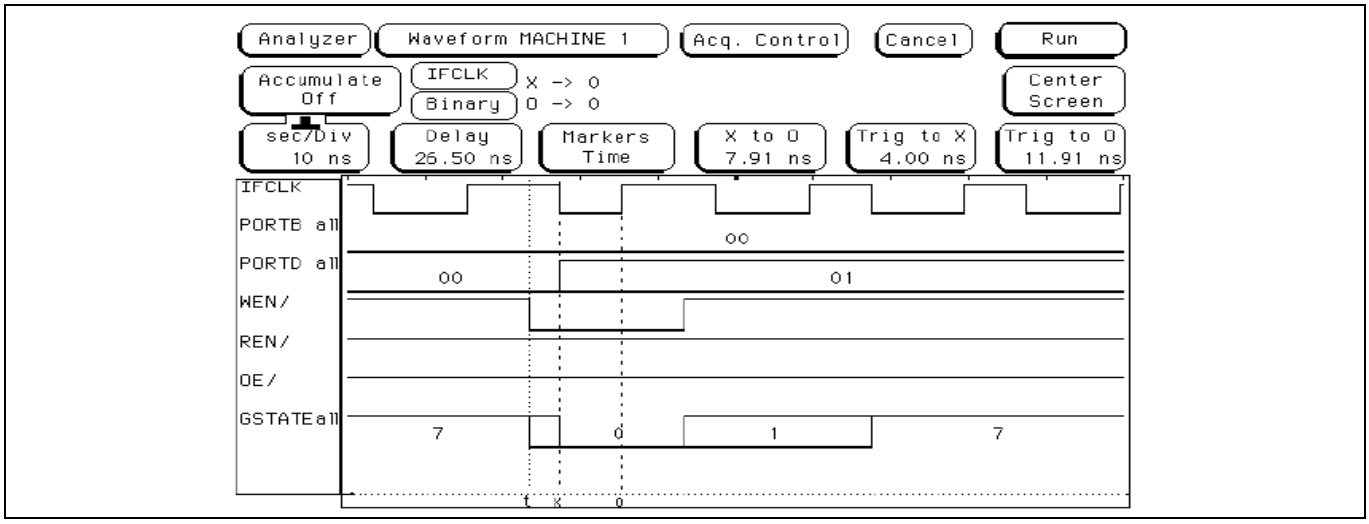


Figure 55 单个字写波形

Figure 55 显示了 GPIF 引擎为 GPIF Designer 定义的单写波形生成的时序信号。本文档介绍了所有信号，包括 GSTATE [2:0]，该信号显示了 GPIF 引擎循环在执行单写操作时循环所经过的状态。

调试提示：

- 将 GSTATE 信号突出显示为逻辑分析仪标题时，您可以通过使用物理接口上生成的信号来确认 GPIF Designer 波形。这样还能够为调试流程提供辅助，因为您可以验证状态切换的正确性。
- S0 将数据放置在总线 (PORTB 为 FD[7:0]，PORTD 为 FD[15:8]) 上，并确认 CTL0 (连接至外部 FIFO 的 WEN#线)。该操作可将 16 位数据值写入到外部 FIFO 内。
- 请注意，需要为 IFCLK 上升沿提供足够的数据设置时间，因为外部 FIFO 的最小设置时间为 4 ns (请参见 CY7C4265 数据手册)。

设计 GPIF 互联

- S1 是一个决策点状态，自动转至闲置状态，以终止该操作。
- 如果不存在无条件转移，那么在达到闲置状态 (S7) 之前，GPIF 引擎将依次经过其余的所有状态 (S2-S6)。
- 对于在批量 OUT 传输中写出的每个字，您应看到 GPIF 引擎按 S0、S1 和 S7 的顺序循环。
- 为捕获波形，请触发位于 CTL0 下降沿的逻辑分析仪。
- 4 ns 的采样频率将为您提供与 **Figure 55** 所示波形的分辨率。

9.6.2 单个字读波形

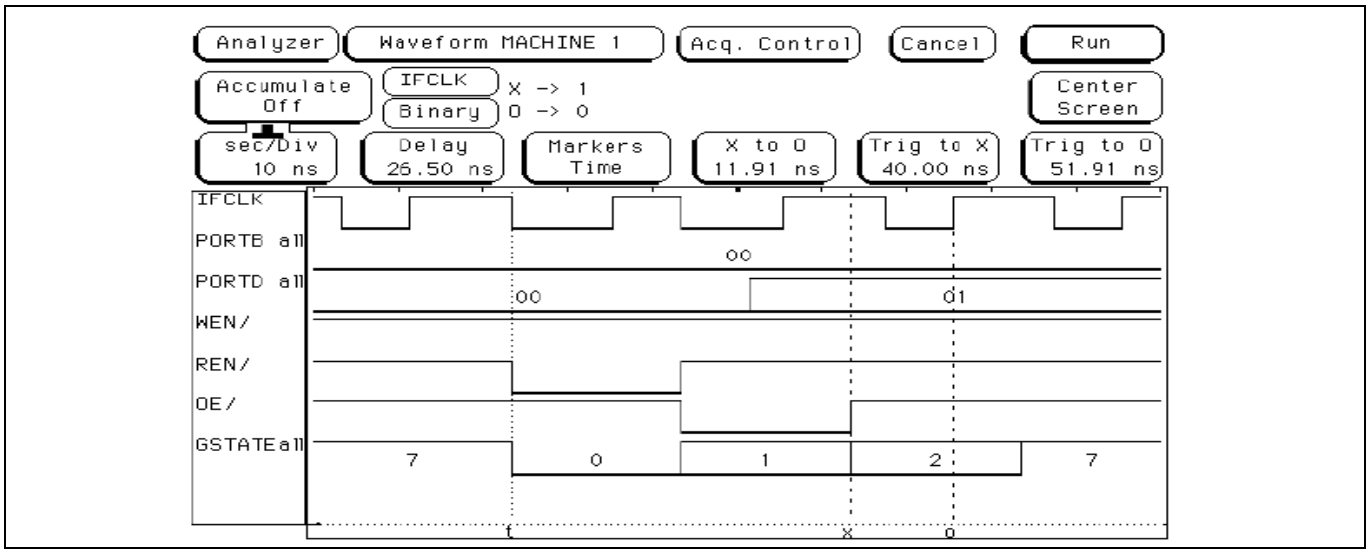


Figure 56 单个字读波形

Figure 56 显示的是 GPIF 引擎为 GPIF Designer 定义的单个字读波形生成的时序。本文档介绍了所有信号，其中包括了 GSTATE [2:0]，该信号显示了 GPIF 引擎循环在执行单读操作时循环所经过的状态。

调试提示：

- S0 确认 CTL1 (连接至外部 FIFO 的 REN#线)，S1 确认 CTL2 (连接至外部 FIFO 的 OE#线)，S2 对数据总线进行采样 (PORTB 为 FD [7:0]，PORTD 为 FD[15:8])。该操作可将 16 位数据值写入到外部 FIFO 内。
- 请注意，需要为 IFCLK 上升沿提供足够的数据设置时间，因为 GPIF 的最小设置时间为 9.2 ns (请参见 FX2LP 数据手册)。
- S2 是一个决策点状态，自动转至闲置状态，以终止该操作。
- 如果没有无条件转移，达到闲置状态 (S7) 之前，GPIF 引擎将依次经过其余所有状态 (S3-S6)。
- 对于从批量 IN 传输中的外部 FIFO 读取的每个字，您都应看到 GPIF 引擎按 S0、S1、S2 和 S7 的顺序循环。
- 为捕获波形，请触发位于 CTL1 下降沿的逻辑分析仪。
- 4 ns 的采样频率将为您提供与 **Figure 56** 中的波形显示一样的分辨率。

10 相关文档

- **FX2LP 入门**：该文档可帮助新用户熟悉 FX2LP。
- **FX2LP 技术参考手册**: 本文档作为 FX2LP 的技术指南使用。“通用可编程接口 (GPIF)”章节对 GPIF 进行了详细说明。
- GPIF Designer 工具用户指南：为打开该文件，需要下载 **GPIF Designer** 中的工具。安装后，请依次选择 **Help** > **This Tool**。
- EZ-USB FX1-EZ-USB FX2LP Development Kit Quick Start Guide.pdf (EZ-USB FX1-EZ-USB FX2LP 开发套件快速入门指南) 和 EZ-USB Development Kit User Guide.pdf (EZ-USB 开发套件用户指南)：这两个文档介绍的都是如何使用 CY3684 套件， (安装 DVK 后) `C:\Cypress\USB\CY3684_EZ-USB_FX2LP_DVK\1.0\Documentation` 中找到。
- **EZ-USB FX1/FX2LP 的端点架构**：通过本应用笔记可以了解 FX1/FX2LP 中的数据流。

10.1 其他 GPIF 示例

可以在下面各赛普拉斯应用笔记中查找更多 GPIF 示例：

- **AN57322: 通过 GPIF 使 SRAM 与 FX2LP 相互连接**: 该应用笔记介绍的是如何使用一个 GPIF 8 位异步 (无时钟) 接口将 CY7C1399B SRAM 连接至 FX2LP。它还介绍了将 FX2LP 连接至其它 SRAM 的指导信息。
- **AN63787: 使用 8 位异步接口配置 EZ-USB FX2LP GPIF 和 Slave FIFO 的示例**: 本应用笔记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) 和从设备 FIFO，以执行 8 位异步并行接口。通过使用两个互联的 FX2LP 开发板 (一个作为 GPIF 主设备使用，另一个作为 GPIF 从设备使用) 可以测试该设计。
- **AN4051: UDMA 的 FX2LP GPIF 流状态特性**: 该应用笔记介绍了 GPIF 的“流状态”特性。该特性扩展了 GPIF 的性能，以处理 ATAPI UDMA。

10.2 参考设计

- CY4611B — USB 2.0 USB 到 ATA 参考设计：流行的 FX2LP 应用就是 USB 批量存储器。通过 FX2LP GPIF 可以轻松连接至附加驱动。赛普拉斯提供了一个使用 FX2LP 的完整批量存储器参考设计。

10.3 数据手册

- **EZ-USB FX2LP USB 微控制器高速 USB 外设控制器**
- CY7C4265 数据手册

11 总结

本应用笔记是用于介绍 EZ-USB FX2LP GPIF 的文档。它说明了创建一个通用的可编程接口的操作步骤，并提供了用于描述 GPIF Designer 主要特性的各种示例。

文档修订记录

版本	提交日期	变更说明
**	2014-08-29	本文档版本号为 Rev**，译自英文版 001-66806 Rev*D。
*A	2017-09-12	本文档版本号为 Rev*A，译自英文版 001-66806 Rev*G。
*B	2022-07-07	更新至英飞凌模板 翻译自: 001-66806 Rev. *H

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

Edition 2022-07-07
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
001-92462 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。