Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

AN66806

Getting Started with EZ-USB® FX2LP™ GPIF

Author Name: Rama Sai Krishna Vakkantula
Associated Project: Yes
Software Version: Keil uVision 2, GPIF Designer
For a complete list of the application notes, click here.

More code examples? We heard you.
For a consolidated list of USB Hi-Speed Code Examples, visit page.

FX2LP™General Programmable Interface(GPIF) provides an independent hardware unit that creates the data and control signals required by an external interface. The GPIF can move data using CPU reads and writes to GPIF registers. This document introduces the GPIF unit and its graphical design tool called GPIF Designer by creating a simple design that divides the GPIF clock by 2, 4, and 7. Only three lines of C code are required to configure and manage this interface. The application note also includes an example demonstrating how to incorporate a USB connection into a GPIF design.

Contents

1 Introduction

The 480 Mbps signaling rate of USB 2.0 requires the controller chip to move the high-speed dataON and OFF. The EZ-USB®FX2LPGPIFprovides an independent hardware unit that the CPU sets up to move data directly to and from USB endpoint FIFOs to an external interface. The external interface can be a RAM, FIFO, or a second processor. Therefore, the CPU does not need to move data.When configured, the CPU only monitors flags and interrupts as the data flows over the GPIF hardware channel.

A wide variety of protocols can be implemented using GPIF such asEnhanced IDE (EIDE – sometimes referred to as Fast ATA or Fast IDE)/ ATA Packet Interface (ATAPI) printer, parallel port (IEEE P1284), and Utopia. This document describes the architecture and implementation of the FX2LP GPIF. It discusses application usage models and debugging strategies, and provides examples to introduce and reinforce GPIF concepts.

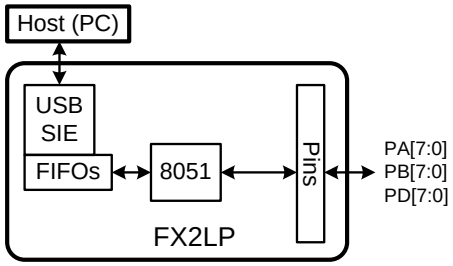
2 FX2LP Architecture Overview

EZ-USB FX2LPis a flexible USB2.0 peripheral controller, designed to handle maximum USB 2.0 bandwidth. FX2LP optimizes the USB throughput by providing the GPIF to implement a high-speed parallel interface to an external device. GPIF moves data between FX2LP endpoint FIFOsand the GPIF interface. The following sections briefly describethe FX2LP architecture by showing different possible configurable modes of FX2LP.

2.1 Ports Mode

FX2LPhas 24 interface pins, which serve different purposes depending on mode settings. In the “Ports” mode, they are general-purpose I/O pins and the GPIF is inactive (Figure 1).

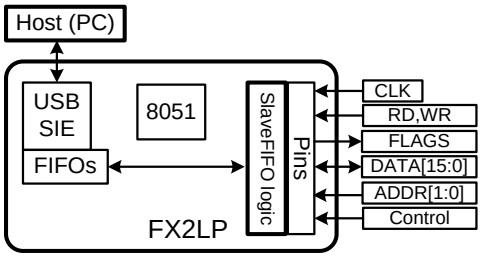
Figure 1. FX2LP in Ports Mode



2.2 Slave FIFO Mode

In Slave FIFO mode, a dedicated FX2LP logic provides control and data signals to connect the USB endpoint FIFOs to an outside FIFO controller. In addition to the data bus and FIFO select inputs, the interface provides the usual FIFO signals such as RD, WR, and FIFO flags.See AN63787 - EZ-USB FX2LP GPIF and Slave FIFO Configuration Examples Using 8-bit Asynchronous Interface.

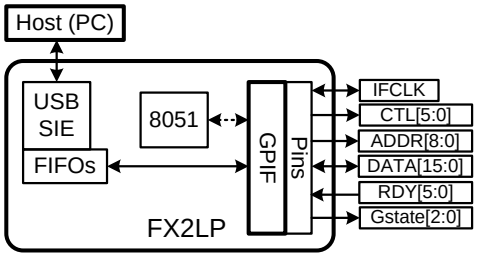
Figure 2. FX2LP Pins in Slave FIFO Mode



2.3 GPIF Mode-Auto

When the GPIF is activated, the interface pins function as a master device to control external peripherals such as a RAM, FIFO, or external processor. The GPIF operates in two sub-modes, Automatic and Manual. In Auto mode, the data flows directly from the endpoint FIFOs to the external interface. The 8051 processor configures and monitors this interface, but does not directly access the FIFO data; seeFigure 3.

Figure 3. FX2LP in GPIFAuto Mode

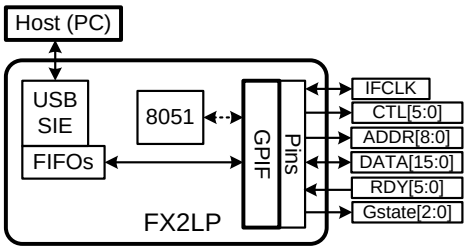


See AN57322 - Interfacing SRAM with FX2LP over GPIFto learn how to connect a Cypress CY7C1399B SRAMto FX2LP using an 8-bit asynchronous interface and GPIF Auto mode.

2.4 GPIF Mode - Manual

In Manual mode, the 8051 processor reads and writes bytes to the interface using GPIF register reads and writes (Figure 4). An example for GPIFManual mode is described in Example 3: Use Single-Word Read/Write Transactions.

Figure 4. FX2LP in GPIFManual Mode



3 General Programmable Interface

3.1 GPIF Overview

At the GPIF's core is a programmable state machine, which controls an 8-bit or 16-bit bidirectional data bus and generates up to six control (CTL) and nine address (GPIFADR) outputs. It also accepts six external and two internal READY inputs to determine branch conditions. Four user-defined waveform descriptors control the state machine; the controlling 8051 program selects one of the four waveforms to be active at any given time.

Each GPIF waveform descriptor contains up to seven states, named S0-S6. The predefined S7 serves as anIDLE state. In each state, you can program the GPIF to:

- Cause any or all of the CTL outputs drive HIGH, drive LOW, or float
- Sample or drive the 8-/16-bit data bus
- Increment the value of the GPIF address bus
- Increment the FIFO pointer
- Trigger a GPIF waveform interrupt to the 8051

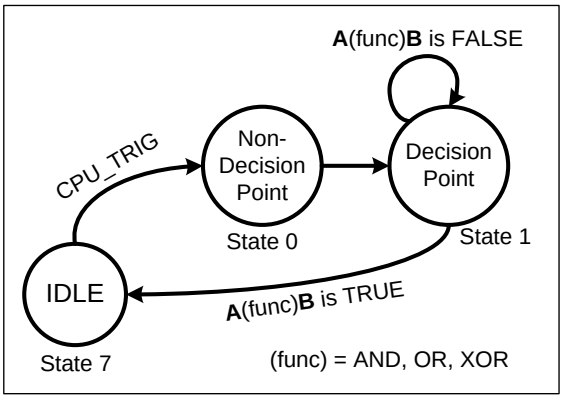
Branch decisions in any state can be constructed as the logical AND, OR, or XOR of two signals taken from the following choices:

- Six READY input pins
- An internal FIFO flag
- An internal RDYflag
- An internalTransaction-Count-Expired flag

The logical combination of the two chosen signals determines the next state. Alternatively, after a programmable delay is counted off, the state machine can move to the next state. This delay can be between 1 and 256 clocks cycles.

States that sample and branch are called "decision points".Any state without a decision point lasts for one clock interval, automatically transitioning to the next state on the next clock.

Figure 5. GPIF State Machine Syntax



The state machine in Figure 5may be expressed as follows:

1. Starting in the IDLE state, wait for the CPU_TRIG signal to assert.
2. When CPU_TRIG asserts, transition to State 0, where you can activate some control signal outputs, move data, or increment the address bus. This state lasts only until the next clock, unconditionally transitioning to State 1because no decision is involved.
3. Stay in State 1until a logical combination of the two signals Aand B isTRUE (while it is FALSE). For example, to stay in State 1 until a READY signal goes valid OR a count expires, select A to represent a READY input and B to represent expiration of a counter. Using the logical operator OR, the state machine will transition out of State 1 if *either* condition, READY or terminal count, occurs.
4. When the condition goes TRUE, transition back to the IDLE state.This stops the state machine.

3.2 Physical Interconnect

The GPIF interconnect contains an 8- or 16-bit data bus, an address bus, control outputs, and ready inputs (Figure 4). For debug purposes, it also includes three GSTATE outputs that indicate the current GPIF machine state. This section describes these signals in detail.

3.2.1 IFCLK

IFCLK (interface clock) is the reference clock for all GPIF operations. It can be an input or output signal;you can select edge, rising, or falling as the active edge. As an input signal, it can be driven using an external clock in the 5 MHz to 48 MHz range. As an output signal, IFCLK can be driven by FX2LP's internal clock at either 30 MHz or 48 MHz. If the external peripheral requires a slower clock, one of the CTL lines can be toggled using FX2LP's internal clock. In the first example, the GPIF clock is divided by 2 and 4 and the signals are output using two CTL outputs. If higher divisors are required, the GPIF state can be programmed to count a programmed number of clocks (1 to 256) before advancing to the next state.

3.2.2 GPIFADR[8:0] (output only)

The GPIF can drive GPIFADR[8:0] to provide address lines for peripherals that need them. These outputs can be held or incremented in any GPIF state.

3.2.3 FD[15:0] (bidirectional)

The data bus is the conduit for payload data transferred between FX2LP endpoint FIFOs and the external peripheral. It can be configured to operate as an 8-bit or 16-bit interface and can be tristated if the system requires it. In 16-bit mode, FD[7:0] represents the first byte in the endpoint FIFO and FD[15:8] represents the second byte.

3.2.4 CTL[5:0] (output only)

Control outputs provide signals required by the external peripheral such as read/write strobes, enables, and divided clock.

3.2.5 RDY[5:0] (input only)

Ready input signals provide status information from the external peripheral such as FIFO status flags and data available. The GPIF can use these signals as decision point qualifiers.

3.2.6 GSTATE[2:0] (output only)

Debug output signals represent the states executed in a GPIF waveform. These are connected to a logic analyzer for debug purposes.

4 Creating a GPIF Application

This section describes thestepsto create a GPIF application.

4.1 Design a GPIF Interface

To design the GPIF interconnect, you need to understand the interface between FX2LP and the external peripheral device. The FX2LP datasheet andTechnical Reference Manual are used to define the interface. The following decisions determine how to configure the GPIF.

- **8-bit or 16-bit datapath?**

This decision is often dictated by the datapath size that the peripheral offers. If it has a 16-bit datapath, use it to maximize the bandwidth over the physical interface.

For a 16-bit datapath, endian-ness (byte order)and bit numbering should also be consideredwhile connecting the data bus.

- **External or internal interface clock?**

This decision is based on how flexible the peripheral is in terms of its own operating modes. For example, if it can accept an external 30- or 48-MHzclock input, the internal GPIF clock can be connected to the peripheral clock input.

- **Address lines required?**

If the peripheral requires any registers or memory locations to be addressed during a read/write cycle operation, then GPIFADR[8:0] can be used.

- **Control lines**

Designate GPIF control outputs from CTL[5:0]. The peripheral may require read/write signals, chip selects, and other control inputs during operation. Determine what these are and allocate CTL[5:0] appropriately. The GPIF Designer tool allows you to name signals for clarity; for example, WR# for CTL[0] and RD# for CTL[1].

- **Status (RDY) lines**

Determine how many status signals need to be monitored during a read/write cycle. Identify these and allocate RDY[5:0] appropriately. Again, the GPIF Designer lets you name these signals to suit your design.

- **Interface timings**

After input and outputare allocated, the main part of a GPIF application is to design timing waveforms, considering the interface timing.

Note: Not all FX2LP package types offer the full set of GPIF interface signals. For example, the 100-pin and 128-pin FX2LP packages provide all six Ready inputs (RDY[5:0]) and Control outputs(CTL[5:0]). The 56-pin package provides two RDY signals and three CTL signals, RDY[1:0] and CTL[2:0].

4.2 Use Firmware Frameworks

When using GPIF Designer to create the interface signals and waveform, you use the Keil integrateddevelopment environment (IDE) to write the controlling firmware. When starting on a new FX2LP firmware project, it is easiest to start with a Cypress-writtenFirmware Frameworks-based Keil uVision2 project. The firmware examples provided with the FX2LP Development Kit (DVK) are Frameworks-based. You can start with one of these, or you can copy a Keil project to a new subdirectory for modification. This allows you to start with a clean firmware base. Starting with a Firmware Frameworks project also allows you to concentrate on your application code because USB low-level protocol code is already written. See the *fw.c*file and the development kit documentation for more details.

The GPIF application solution has two main components:

- The firmware that configures the GPIF and launches GPIF transfers. This firmware also performs other application tasks such as USB enumeration and endpoint configuration.

- The GPIF waveform descriptors that implement the physical bus timing and data flow

The firmware consists of five files: *fw.c*, *periph.c* (you can rename this file), *dscr.a51*, *ezusb.lib*, and *usbjmptb.obj*. These files comprisethe Keil uVision 2 Firmware Frameworks project.

The second component is a C source file (for example, *gpif.c*) that containsthe code to define GPIF waveforms and initialize the GPIF unit. The GPIF Designer tool creates this C file after you graphically define your interface. This eliminates the need to know individual registers and bits inside the GPIF unit. You can download the GPIF Designer utility here.

4.3 Implement Waveforms Using GPIF Designer

The GPIF Designer generates C code containing GPIF waveform descriptors, which implement the physical bus timing of the interface. GPIF Designer can implement multiple waveform behaviors; these can be assigned to four waveform types: Single Write, Single Read, FIFO Write, and FIFO Read. The CPU controls which waveform to use by writing a GPIFWFSELECT register. Each descriptor is 32-bytes long and resides in a special GPIF waveform descriptor area in on-chip memory space. The GPIF Designer tool allows you to regard these descriptors as “black boxes” because it generates all the C code necessary to implement and use them.

You create these GPIF waveformsas state machines. There are seven states or intervals (S0-S6) to work with plus an automatic IDLE state S7, which is used to terminate a transaction.

Figure 6. GPIF Waveforms Built from States

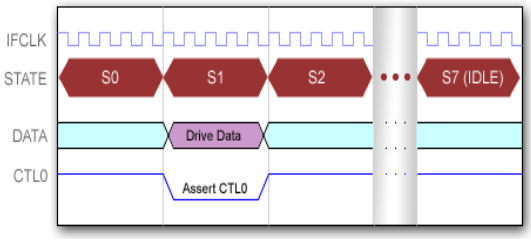


Figure 6 shows an example of a simple waveform broken down into GPIF state transitions.

5 Example 1: Divide GPIF Clock by 2 and 4

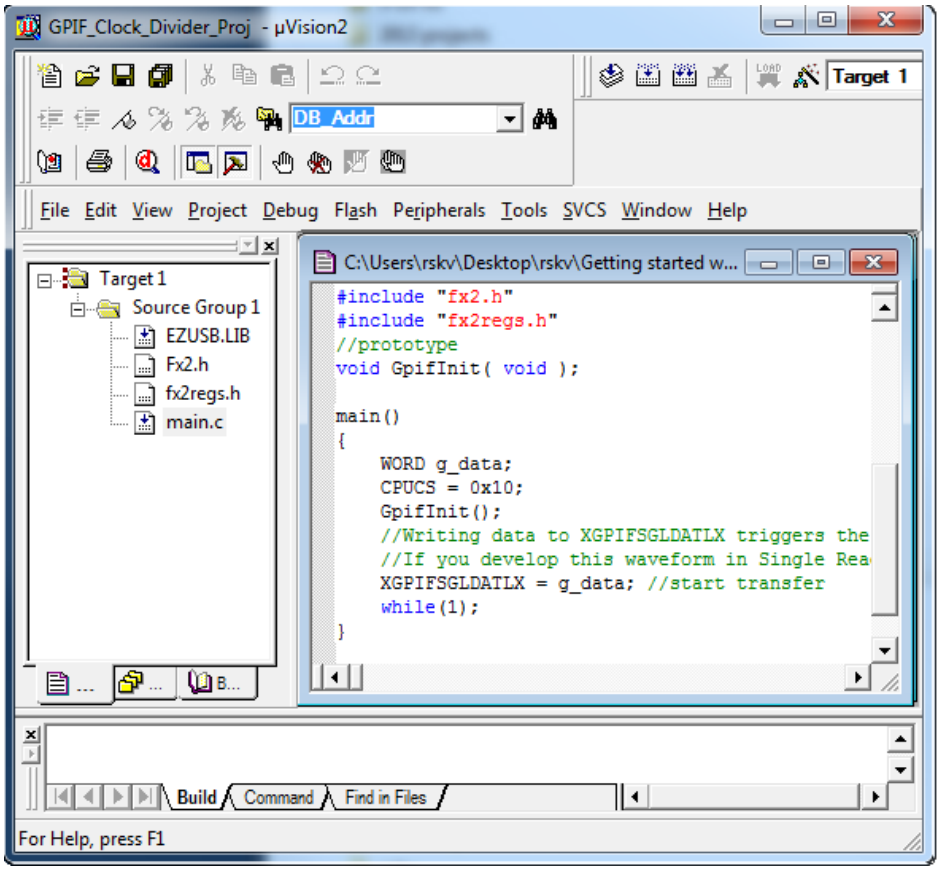
To start, let us design a state machine to divide the GPIF clock by 2 and 4, and output the divided clock signals on CTL[0] and CTL[1]. This will give a basic understanding of the GPIF Designer tool before introducing advanced features involving USB endpoint FIFOs.Because USB is not used in this example, the firmware frameworks are not required.

1. Unzip and save the *FX2LP source code and GPIF project files.zip*file attached to this application note. Go to `FX2LP Source code and GPIF project files\Firmware\GPIF Clock Divider`.Right-click on the folder and then click**Properties**. If the **“Read-only...”**option is selected, uncheck it. Click **OK** to approve applying changes to subfolders.

   From this folder, start the Keil uVision2 IDE by double-clicking **GPIF_Clock_Divider.uv2**.This contains a skeleton GPIF project with everything but the GPIF information. Double-click *main.c*to open it; you will see that a GPIF operation can be started with only three lines of code:

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

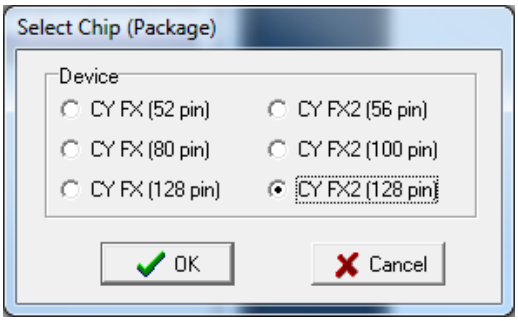
Figure 7. Skeleton GPIF Keil Project



If you compile this code asis, you will see linker errors becauseit does not include the GPIF C file generated by GPIF Designer. This file contains the GpifInit() function and the waveform table data. The next step is to create this file.

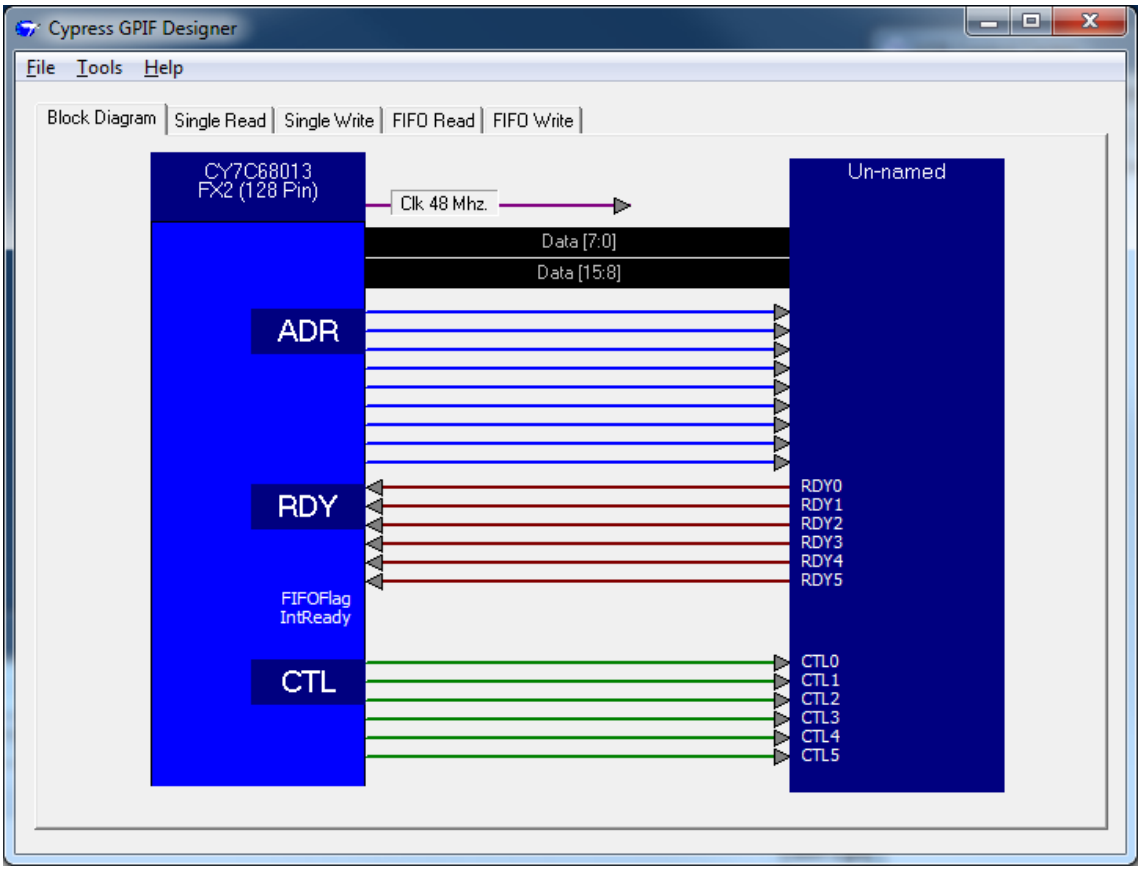
2. Start GPIF Designer and select **File**>**New**to bring up the window in the following figure.

Figure 8. GPIF Designer Startup Window



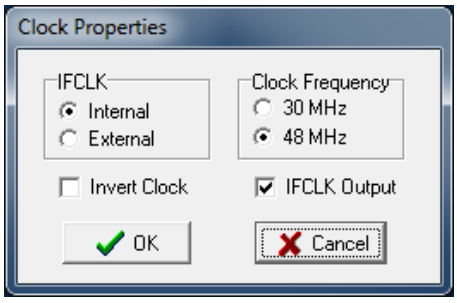
3. To use the FX2LP development kit board to test the example, select **CY FX2 (128pin)**and click **OK**. This brings up ablock diagram window:

Figure 9. GPIF Designer Block Diagram



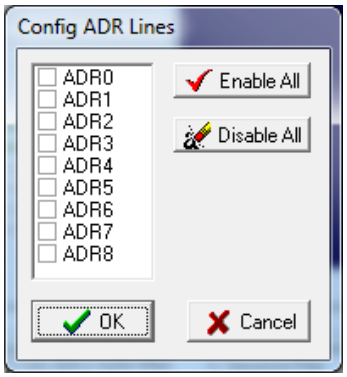
4. At the top of the diagram, right-click in the **Clk 48 MHz** text box to configure the GPIF clock. For this project, leave the default settingsunchanged.

Figure 10. IFCLK Default Settings



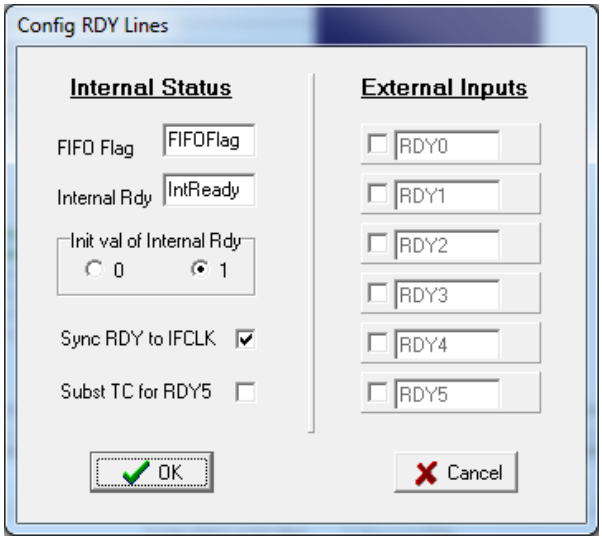
5. Right-click on the **ADR** (address) label. This example does not use the address bus, so you can click **Disable All**; this dims the address wires in the block diagram.

Figure 11. Disable Address Lines



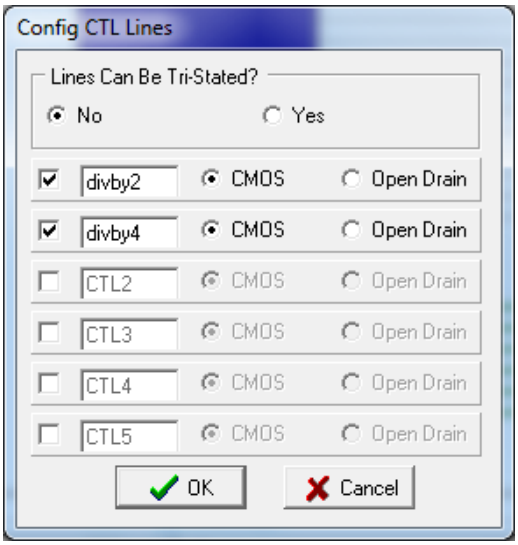
6. Right-click on the **RDY**(ready) label to configure the six RDY inputs. This example does not use RDY inputs, so they can all be deselected. To deselect RDY5, first deselect **Subst TC for RDY5**.

Figure 12. Disable RDY Inputs



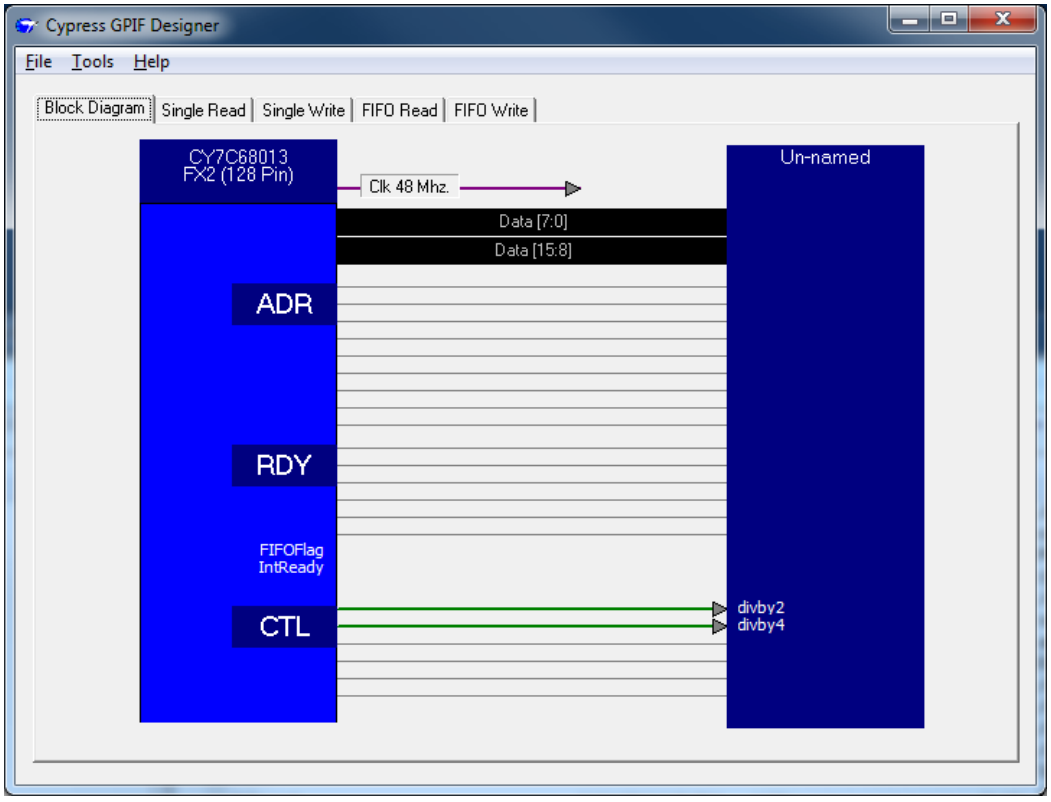
7. Right-click on the **CTL**(control) label to configure the six CTL outputs. Set them as shown in the following figure.

Figure 13.CTL Output Settings



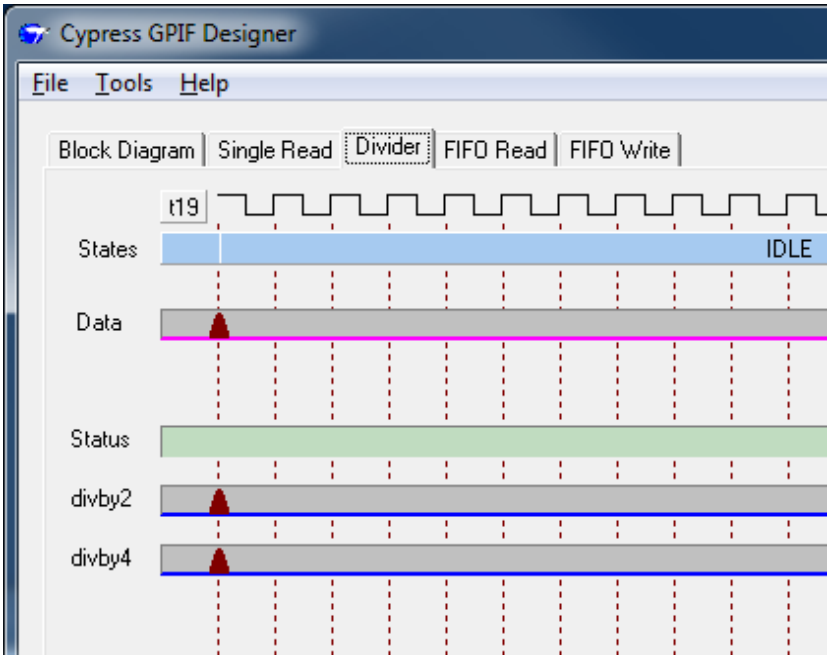
Rename the two outputs as **divby2** and **divby4** by typing in the text boxes. It is good practice to name your signals because they update labels in all the GPIF Designer screens, making the signals easier to identify (for example, divby4 instead of CTL1). Now the block diagram looks much simpler.

Figure 14. Block Diagram for Clock Divider



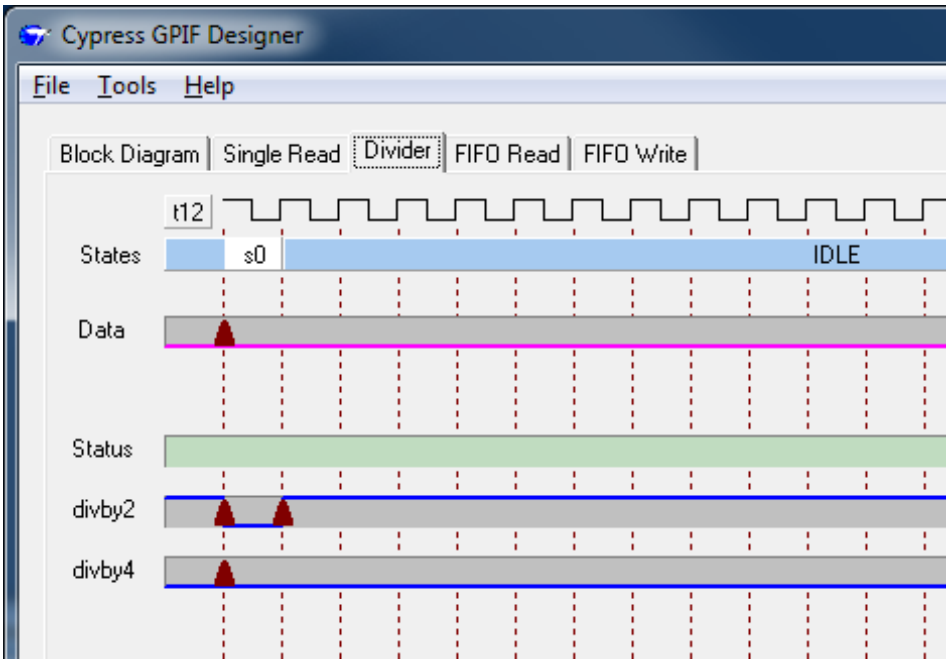
8. Select the **Single Write** tab, which is used to draw the two waveforms. Right-click on the tab name, select **Select Tab Label** and rename it as**Divider**. You will see a blank waveform editing screen, as shown in Figure 15.

Figure 15. Blank Waveform Editor Screen



9. The state machine has one state called IDLE. To add states, click in one of the CTL bandslabeled divby2 and divby4. You can also add states by clicking in the Data band, but this design does not use the data bus. In the divby2 band, click on the second dotted vertical line.

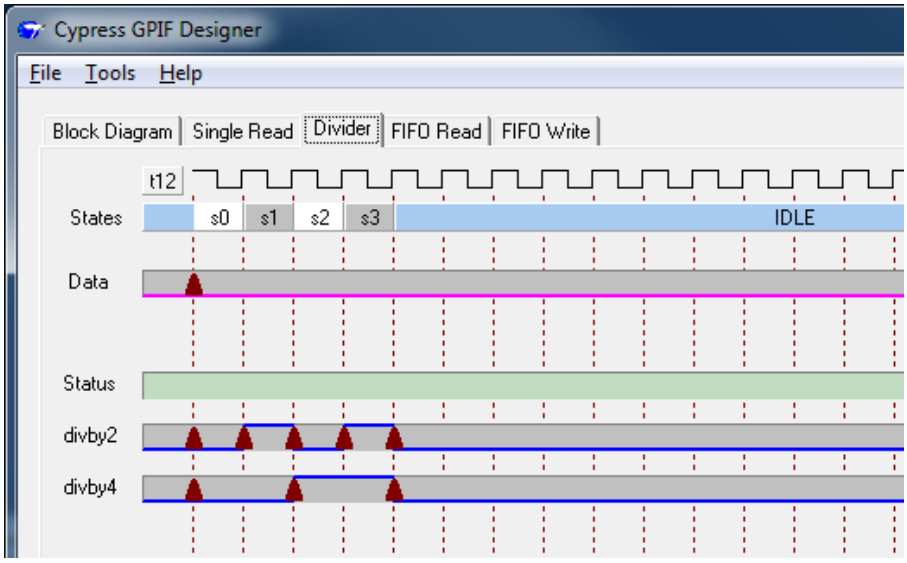
Figure 16. First Waveform Transition



The appearance of the triangle signifies that GPIF Designer has added a state s0 and toggled the divby2 output at the beginning and end of s0.

10. Now, click in the divby2 waveform at each of the next three clock transition dotted lines. This creates a divide-by-2 of the clock.
11. In the divby4 waveform, click on the second and fourth clock lines. Your screen should appearas shown in Figure 17.

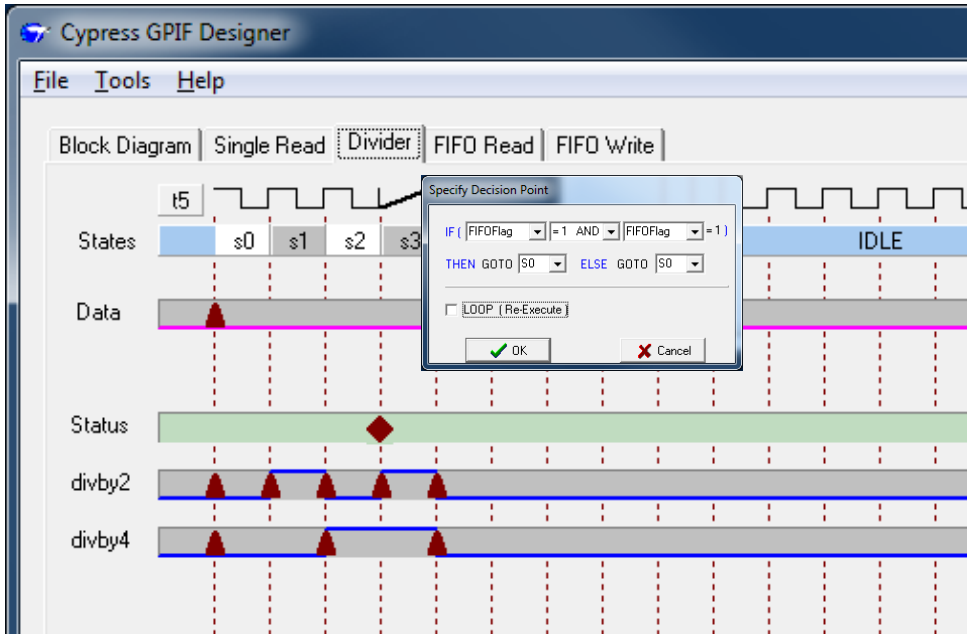
Figure 17. Div by 2 and Div by 4 Waveforms



Note: You can right-click any triangle to change its logic state to modify signal polarities. You can also drag triangles horizontally to make states last longer than one clock.

When the Figure 17 state machine starts, it will move from state s0 to s1 to s2 to s3 on each rising clock edge, and then stop at IDLE. You need a way to make it repeat; in other words, s3 should always branch back to s0. To do this, add a 'decision point' in the Status line.

Figure 18. First Decision Point



12. Click in the Status band between the s2 and s3 states. This adds a diamond as a decision point and brings up a window to configure the decision. To make an unconditional branch, you can pick any two logic signals from the top drop-down lists and any logical operator for the two signals. You need to specify both the THEN and ELSE conditions to transition to S0. In other words, *unconditionally* branch to S0. The **LOOP (Re-Execute)** check box is used to reassert the state conditions every time the state is entered. There is no “stay in this state until something changes” loop in s3, so this can be unchecked.

    The clock waveform displays a diagonal line in s3 to indicate that the number of clocks for a decision state is unknown, because the number of clocks depends on when the branch condition is satisfied.

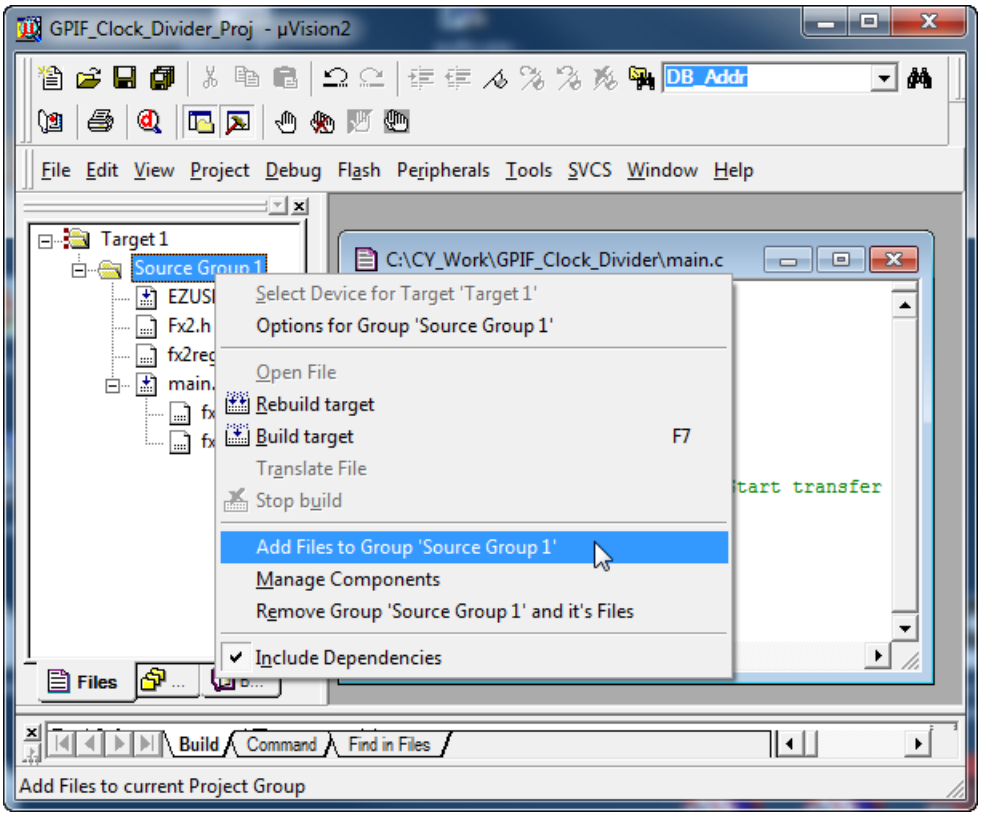
    GPIF Designer tracks all parameters used by the design and updates the selection choices accordingly. For example, if you add a state s4, it is automatically added to the GOTO choices.

    Important: Decision points are placed at the beginning of the state in which they occur and not the end. This is why the diamond is placed at the beginning of s3.

13. To save the GPIF Designer project, select **File**>**Save As** and save it to the Keil project folder as*Div_2_4.gpf*. You can reopen this file later if you wantto modify your waveform. To save the C code generated by GPIF Designer, select **Tools**>**Export to GPIF.c file** and save it in theKeil project folder as *GPIF_div_2_4.c.Div_2_4.gpf* and *GPIF_div_2_4.c* are also provided in the `FX2LP Source code and GPIF project files\GPIF Clock Divider\GPIF_div_2_4` folder. You can directly use them to test this project.

14. The last step is to include the GPIF Designer C file in the Keil project. Right-click **Sources Group 1** and select **Add files…**. Navigate to the Keil project folder and include the *GPIF_div_2_4.c* file.

Figure 19. Add GPIF Designer C File to Keil Project



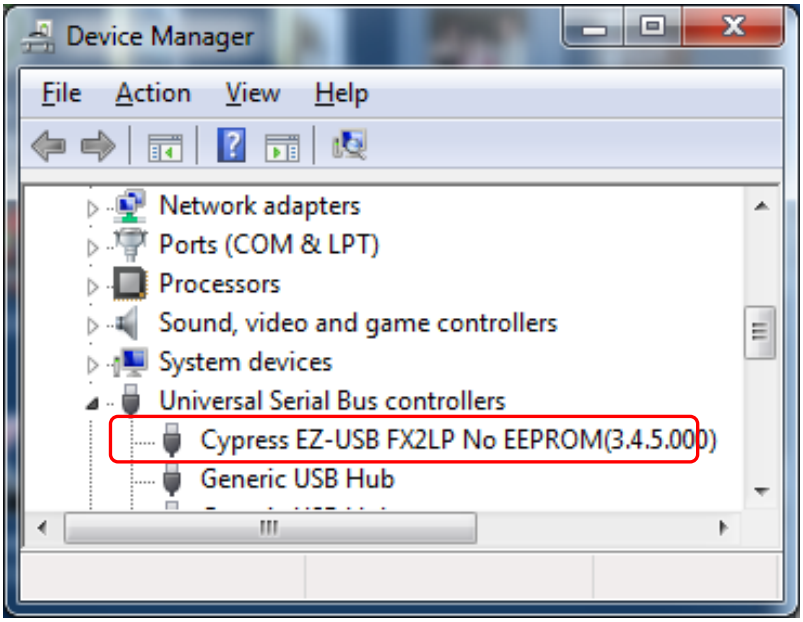
15. Compile the project. The Keil IDE compiles and links the files, then calls the ‘hex2bix’ utility to convert the load module to a format compatible with the Cypress USB Control Panel loader.

    You can try out the design on an FX2LP development board as follows:

    1. Move the EEPROM SELECT switch to the off (down) position. This enables the FX2LP on-chip USB code loader.

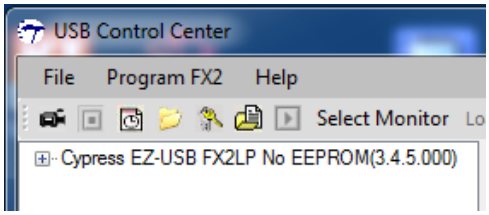
2. Plug the FX2LP board into a PC USB port. If this is the first time, you should see pop-up messages to install a USB driver. Navigate to `C:\Cypress\USB\CY3684_EZ-USB_FX2LP_DVK\1.0\Drivers\cyusbfx1_fx2lp` and select the folder corresponding to your Windows OS.You can confirm a successful driver installation by viewing the Windows Device Manager:

Figure 20.Cypress USB Loader DriverInstalled



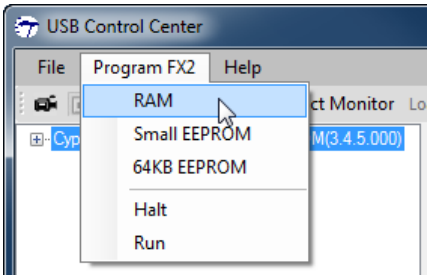
16. Launch the USB Control Panel at `C:\Cypress\Cypress Suite USB 3.4.7\CyUSB.NET\bin\CyControl.exe`.
17. You should see the FX2LP board along with other connected USB devices listed in the left panel. To view only the FX2LP board, click the **Device Class Selection** tab in the right panel and uncheck everything but **Devices served by the CyUSB.sys driver (or a derivative).** The left panel should appear similar to Figure 21.

Figure 21. FX2LP Board in USB Control Center



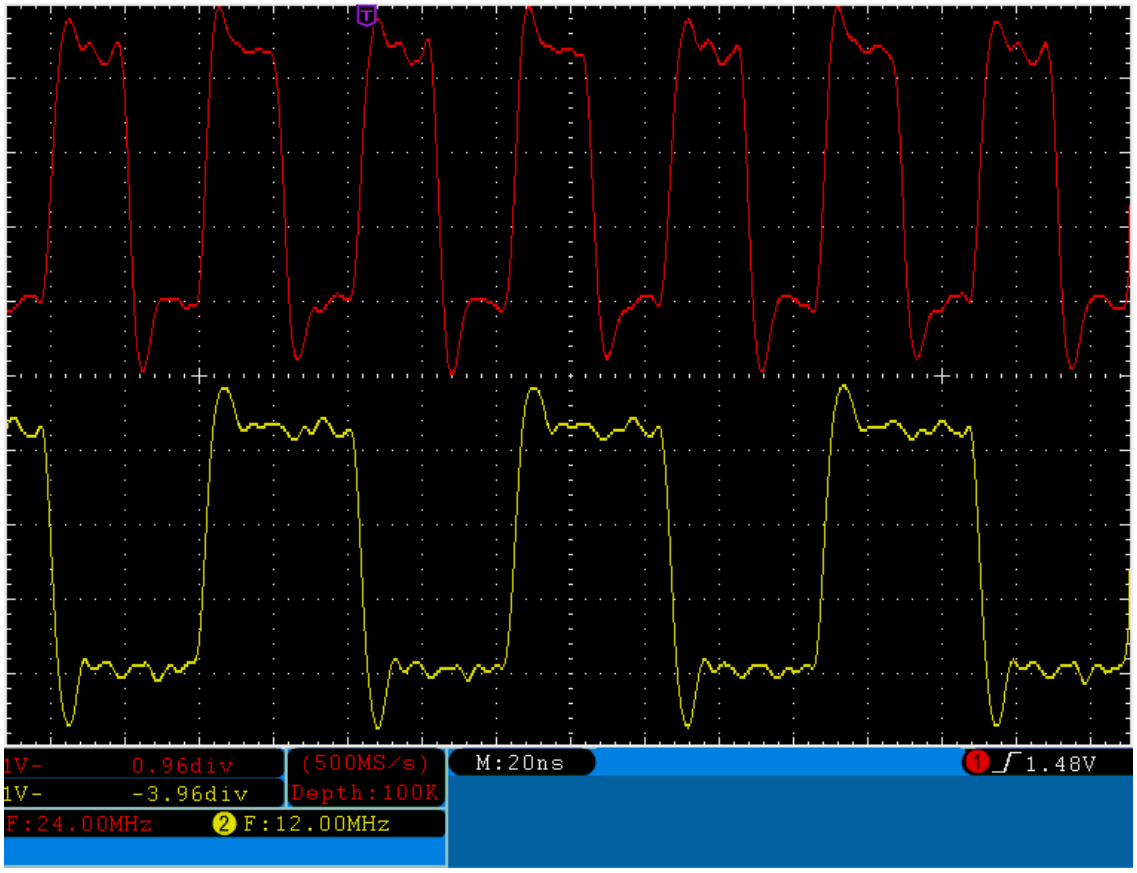
18. Now,you can load the Keil-compiled hex file into the board. Highlight the EZ-USB entry andselect **Program FX2** >**RAM**.

Figure 22. Load New Device Code into RAM



19. Press the **RESET** button on the FX2LP development board. This enables the USB loader.
20. Navigate to the Keil project folder and select the result of the Keil compiler, *GPIF_Clock_Divider_Proj.hex*.
21. Probe P2 pin 11 (CTL0=divby2) and P2 pin 10 (CTL1=divby4). You should see waveforms similar to Figure 23.

Figure 23. GPIF 48-MHz Clock Divided by 2 and 4

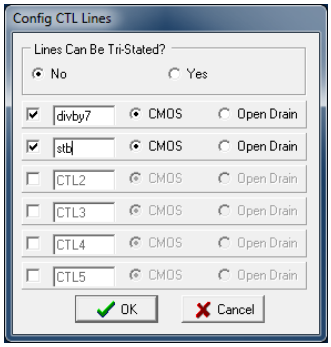


6 Example 2: Divide GPIF Clock by 7

By modifying Example 1, you can choose any clock divider, even using odd divisors.

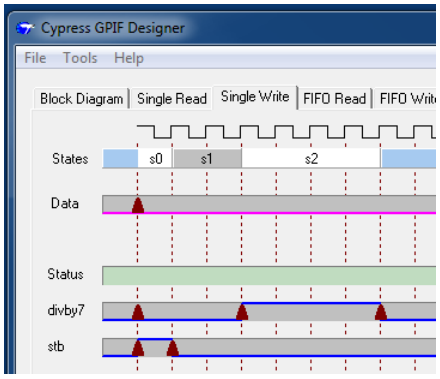
3. Start GPIF Designer; select **File**>**New** to create aGPIF project. Select the **CY FX2(128 pin)** device.
4. Eliminate the ADR and RDY signals as described earlier and rename the two CTL signals: CTL0 as divby7 and CTL1 as stb. Disable the other four signals.

Figure 24. Two Outputs, Divide by 7 and Strobe



5. Select the **Single Write** tab. In the divby7 band, place a state transition triangle three clocks after the first triangle and another triangle four clocks after that. In the stb band, place a triangle one clock after the beginning; right-click on the first triangle and set it **HI**; set the second triangle to **LO**. This converts the negative polarity pulse to a positive polarity pulse in s0. The waveforms should look similar to Figure 25.

Figure 25. Divide by 7 and Strobe Waveforms



6. Finally, click in the Status band at the clock line corresponding to one clock before the end of s2. This creates a decision point and a new state s3. The decision point should be configured to unconditionally branch to S0. Because the THEN and ELSE GOTO states are both S0, the IF conditions are irrelevant—you can pick any of them in the drop-down lists.

Figure 26. Unconditional Branch to S0

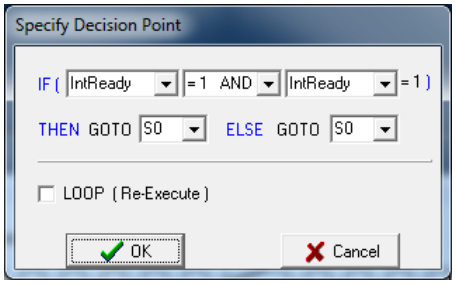
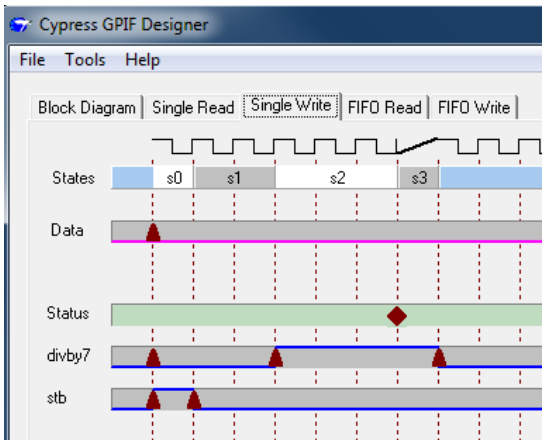
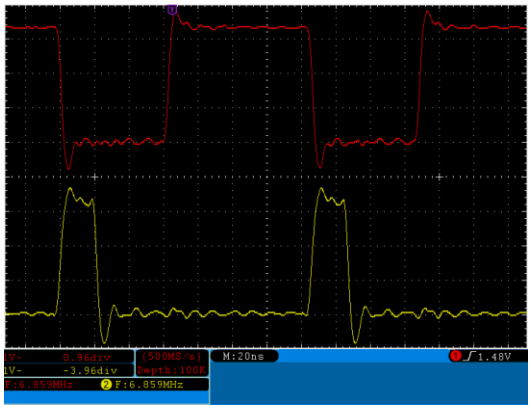


Figure 27. Final Divide-by-7 Waveforms



7. Export this file (**Tools**>**Export to GPIF.c**) to the Keil project folder as *GPIF_div_7.c*.Save the project file (**File**>**Save As**) as *Div_by_7.gpf*. The *Div_by_4.gpf* and *GPIF_div_7.c* files are also provided in the`FX2LP Source code and GPIF project files\GPIF Clock Divider\GPIF_div_7` folder. You can directly use them for testing this project.
8. In the Keil Files tab, right-click **Source Group 1**, click **Add Files…** and add *GPIF_div_7.c* to the project. You now have two GPIF waveform files, so the Keil compiler needs to know which one to use. To disable any source filefrom the build, right-click it, select**Options for file…**, and uncheck the **Include in Target Build** checkbox. If you have more than one *GPIF.c* file in your project, make sure that only one of them has this box checked.
9. Select **Project**>**Rebuild All Target Files** to recompile.
10. Press the **RESET** button on the FX2LP development board. This reinstates the USB code loader.
11. Use the USB Control Center panelas before to load the newly compiled *GPIF_Clock_Divider_Proj.hex* file.
12. Probe P2 pin 11 (CTL0=divby7); the screen appears as shown in Figure 28.

Figure 28. Top: Divide by 7; Bottom: stb Signal is One Clock Wide for Reference



What if you need longer clock divisions, such as div-by-87? GPIF Designer allows you to set the duration of astate from 1 to 256 clocks. Click any state in the States band; a window appears as shown in Figure 29. Click **Set State Duration** and set the number of clocks (Figure 30).When you manually set the number of clocks in a state, the timing diagram clock line is no longer indicative of the clocks in that state.

Figure 29. s2 Properties

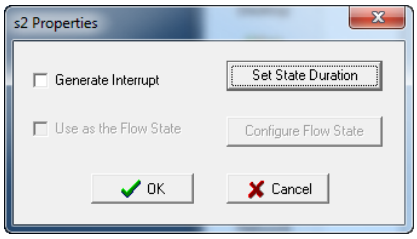
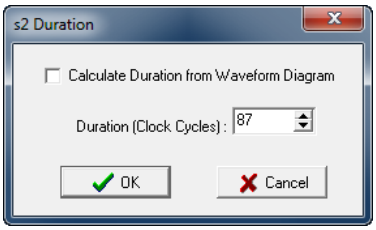


Figure 30. Select 1-256 Clock Cycles



7 Example 3: Use Single-Word Read/Write Transactions

This example loops FX2LP data through the Cypress CY7C4265-15AXC external FIFO. The example uses GPIFManual mode and exercises single-wordread/write GPIF transactions over a 16-bit data bus. You can monitor the FIFO write operations with a scope or logic analyzer, but to test the reads you need to connect an external FIFO chip to the FX2LP development board.

For this example, mount the external FIFO onto an FX2LP development board using the prototype board supplied with the development kit. For full hardware specifications of the external FIFO, download theCY7C4265 datasheet. The pinout list for the prototype board connection to the FX2LP development board and a full schematic for the external FIFO prototype board is available in theHardware folder, which is part of the attachment to this application note.

Single-word read/write transactionstransfer one byte or word of data between the FX2LP and the peripheral. These transactions are simpler to implement than FIFO read/write transactions and it is a good idea to implement them first. Performing this stage first in a GPIF development cycle allows you to verify all areas of the system (hardware, firmware, software) before proceeding to the more complex design. After validating the physical interconnect and basic data movement, it is easier to move on to the full design.

7.1.1 Implement FIFO Read/Write Transactions

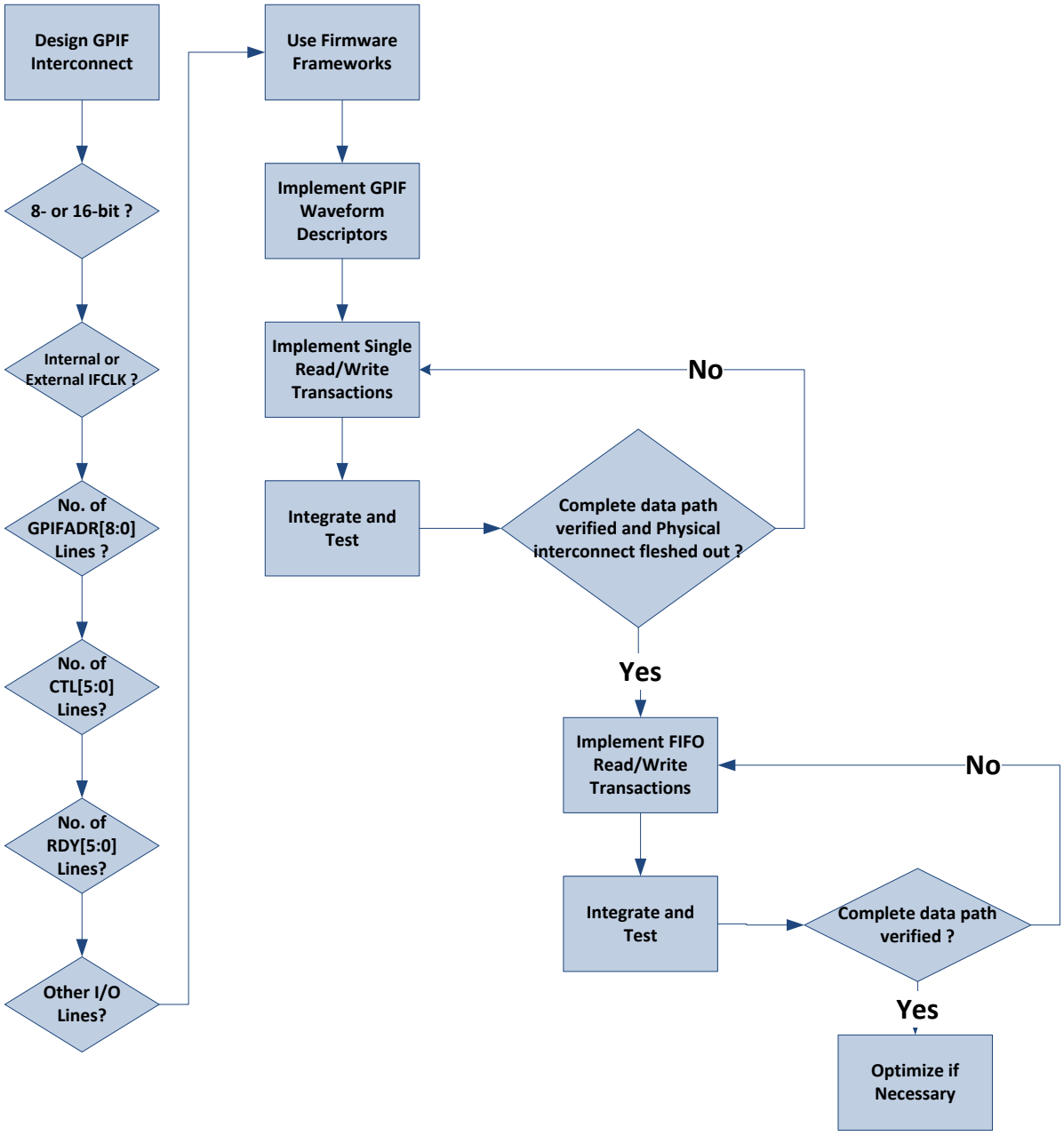
After you implement the single-word transaction, the next step is to implement GPIF FIFO read/write transactions to achieve higher bandwidth. Implementingthe FIFO write waveforms firstavoids the problem of testing a full loopback solution; if it does not work, it is difficult to isolate the problem to the write, read, or both sections.

7.1.2 Optimize if Necessary

In generating the GPIF waveforms, you may initially set a high physical bus time. If you have done this, it is possible to cut down the cycle time for each GPIF transaction and still meet the timing parameters required by the peripheral. You can also revise the design to improve firmware code efficiency and overall firmware code flow at this stage.

Figure 31summarizes the steps in this section in the form of a GPIF design flow diagram.

Figure 31. GPIF Design Flow Diagram



7.1.3 Connect FIFO to FX2LP GPIF Interface

Figure 32shows the FX2LP to external FIFO connections. The FX2LP uses its bidirectional bus FD[15:0] to write and read data from the external FIFO. The FIFO data bus is made bidirectional by connecting its output data bus Q[15:0] and input data bus D[15:0] together. The example uses USB BULK transfers to write and read FIFO data. These transfers can be exercised by using the USB Control Center utility supplied with the SuiteUSB 3.4 - USB Development tools for Visual Studio.

Table 1describes the GPIF interconnect in detail. CTL and RDY pins are assigned to be compatible with the smallest FX2LP package, having 56 pins.

Figure 32. GPIF Connection to External Synchronous FIFO

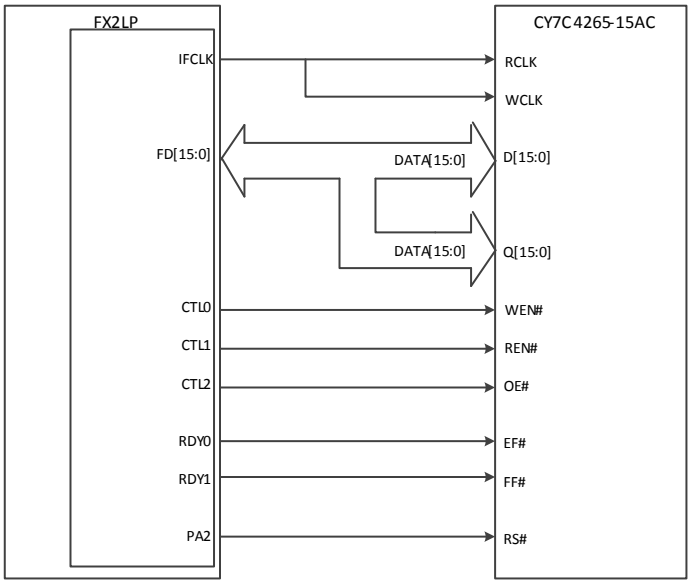


Table 1. Assignment of FX2LP GPIF Signals to CY7C4265-15AC Signals

FX2LP GPIF Signals	CY7C4265-15AXC Signals	Description
IFCLK	WCLK, RCLK	IFCLK is connected to the write and read clock inputs (WCLK, RCLK) of the external FIFO. Data is clocked into the external FIFO on every rising edge of WCLK while WEN# is asserted. Similarly, the FIFO presents data on Q[15:0] on every rising edge of RCLK while REN# and OE# are asserted. The external FIFO can accept an input clock frequency of up to 66.7 MHz, so it can handle the incoming IFCLK frequency of either 30 MHz or 48 MHz. Note that the hash symbols (for example OE#) mean active-low.
FD[15:0]	D[15:0], Q[15:0]	The GPIF data bus (FD[15:0]) is connected to the external FIFO's input data bus D[15:0] for word-wide operation. The external FIFO's output data bus Q[15:0] is also connected to the GPIF data bus to allow the FX2LP to read back the FIFO data contents. Because the two uni-directional FIFO data buses are connected together, the GPIF must control the OE# signal to avoid bus contention. Specifically, OE# must never be low while FX2LP is driving the data bus—this will create bus contention due to both FX2LP and the FIFO driving the data bus at the same time.
CTL0	WEN#	CTL0 connects to the external FIFO write enable pin WEN#. While the GPIF holds WEN# LOW, data is written into the external FIFO on every rising edge of WCLK.
CTL1	REN#	CTL1 connects to the external FIFO read enable pin REN#. While the GPIF holds REN# and OE# LOW, the FIFO drives new data on Q[15:0] on every rising edge of RCLK.
CTL2	OE#	CTL2 connects to the external FIFO output enable pin OE#. While REN# and OE# are held LOW, the FIFO drives new data on Q[15:0] on every rising edge of RCLK.
RDY0	EF#	RDY0 connects to the external FIFO EMPTY flag EF#, which the FIFO asserts (LOW) if it is empty. Because READY signals can be tested in GPIF branch states, the GPIF can use this signal to regulate data transfers while reading from the external FIFO.
RDY1	FF#	RDY1 connects to external FIFO the FULL flag FF#, which the FIFO asserts (LOW) if it is full. The GPIF can use this to regulate data transfers when writing to the external FIFO.
PA2	RS#	PA2 connects to the external FIFO RESET pin. PA2 is an FX2LP GPIO pin and is not part of the GPIF logic. 8051 code uses PA2 to reset the external FIFO to a known state before GPIF data transfers start.

8 USB Data Flow

USB Endpoint 2 OUT (EP2OUT) is used as the source endpoint for GPIF writes to the external FIFO; Endpoint 6 IN (EP6IN) is used as the sink endpoint for GPIF reads from the external FIFO. USB IN and OUT directions are from the host point of view:

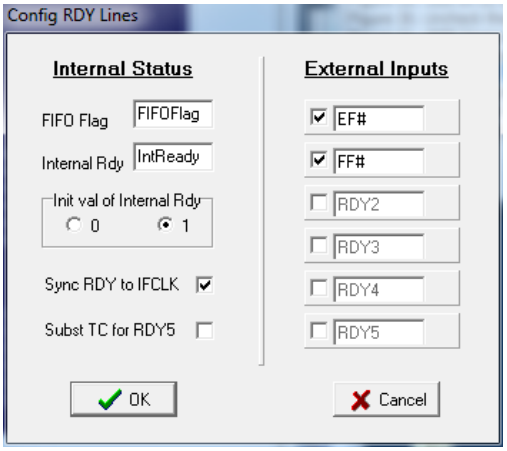
- EP2OUT contains data packets sent by the USB host (a PC) and received by FX2LP.
- EP6IN contains the data packets sent by FX2LP and received by the PC.

9 Design the GPIF Interconnect

The GPIF interconnect is set up using the same steps as outlined in the first example and are described briefly here.

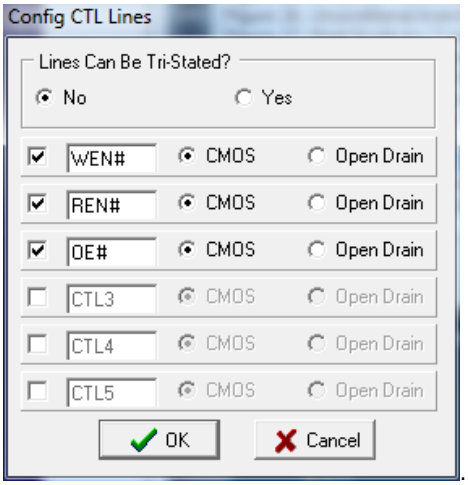
1. Start the Cypress GPIF Designer tool.
2. Go to **File** > **New** and select the **CY FX2 (128 pin)** as used on the FX2LP development board.
3. Right-click the **Un-named** label in the external device block and rename it **CY7C4265-15AC**.
4. By default, the data bus is configured to 16-bit. In this example, a 16-bit data bus is used.
5. Right-click the **ADR** label. This design does not use the address lines, so disable them to simplify the block diagram.
6. Right-click the **RDY**label and configure the RDY signals, as shown in Figure 33.

Figure 33. RDY PinConfiguration



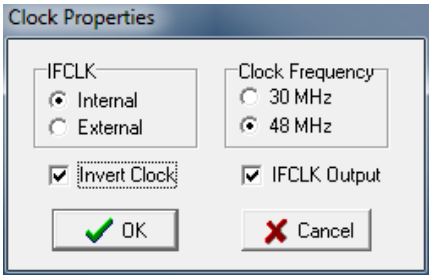
7. Right-click the**CTL**label and configure it, as shown in Figure 34.

Figure 34. CTL Pin Configuration



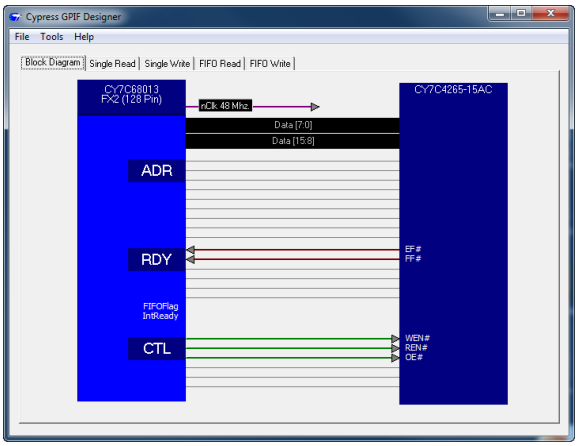
8. Right-clickthe **48 MHz CLK** text box and configure the clock properties as shown inFigure 35. Select the **Invert Clock** checkbox. The GPIF changes and samples signals on the rising clock edge; the FIFO changes and samples signals on the falling clock edge. This half-clock offset gives the interface timing ample setup and hold times.

Figure 35. Configure IFCLK



The Block Diagram should appear similar to Figure 36. The clock is now labeled “nClk 48 Mhz”, where the “n” indicates clock inversion.

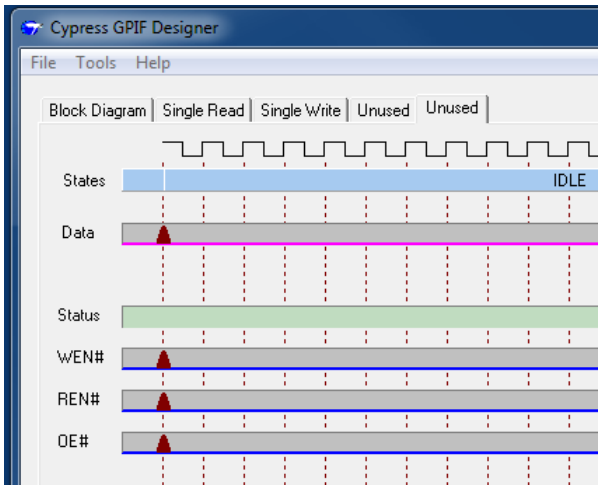
Figure 36. Configured FIFO Interface



9.1 Single-Word Write Waveforms

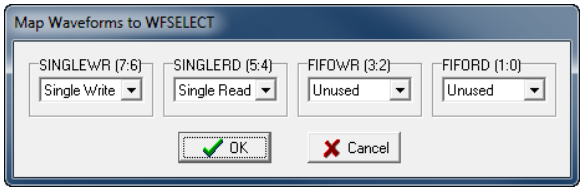
The write waveform is designed to move data from an FX2LP OUT Endpoint FIFO to the External FIFO. Right-click the FIFO Read tab and rename it “Unused”. Do the same for the FIFO Write tab. Your screen should appear similar to Figure 37.

Figure 37. Waveform Screen



Select **Tools** > **Map Waveforms** to WFSELECT (Figure 38).

Figure 38. Waveform Mapping Dialog

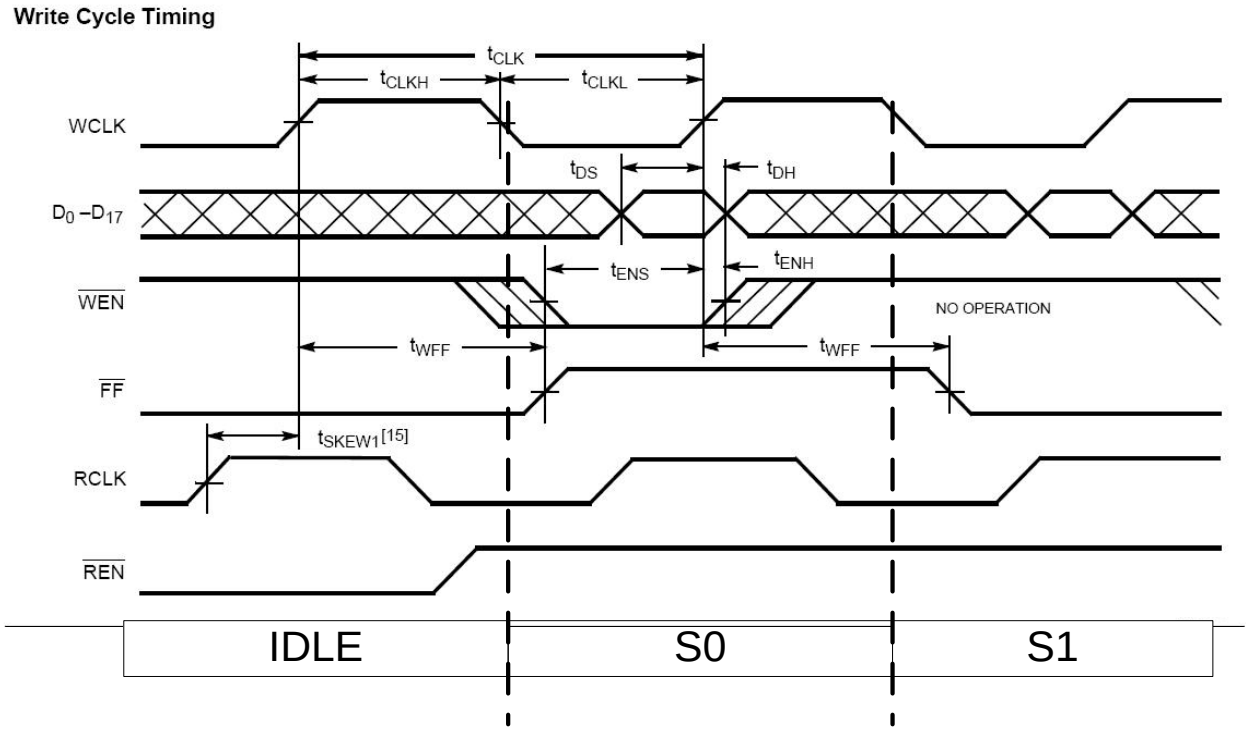


Make sure the Single Write waveform is mapped to SINGLEWR and the Single Read waveform is mapped to SINGLERD. This pairs the 8051 CPU registers that launch the waveforms to the GPIF-named waveforms.

Note: Waveform mapping allows you to define more than one waveform set to the same type of transfer. For example, you can use two different waveform types for Single Write.

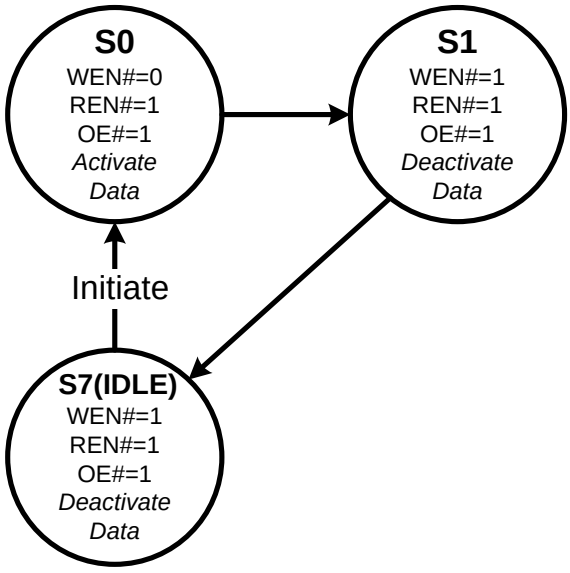
Write waveforms manage byte transfers that transfer data from an FX2LP OUT endpoint to the external FIFO.To construct the FIFO Write waveform, first review the write cycle timing for the CY7C4265 FIFO given in the CY7C4265 datasheet. Figure 39shows the write cycle timing with GPIF states added to the bottom of the diagram.

Figure 39. Write Cycle Timing Diagram



From the write cycle timing diagram, you can construct the state diagram in Figure 40.

Figure 40. Single-Word FIFO WriteState Diagram



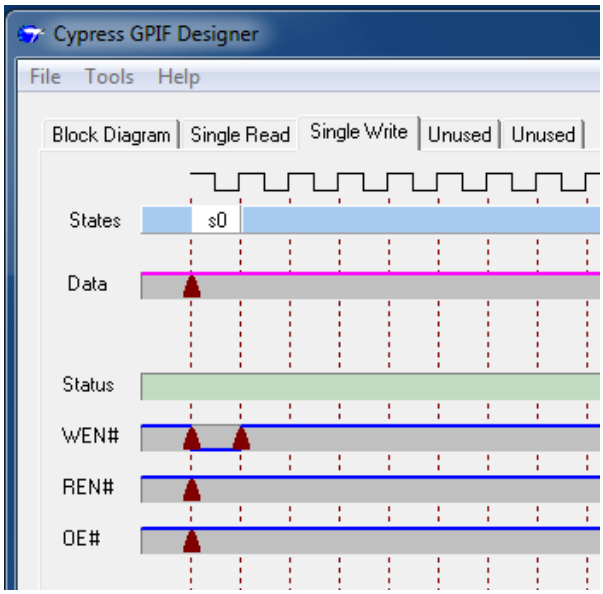
- For the single-word write waveform, data is written to the external FIFO during S0 by making WEN# logic LOW and driving the data bus.

- In S1, WEN# deactivates (goes HIGH) and the GPIF data bus stops driving (floats). S1 is a decision-point state that forces an unconditional branch to the IDLE state, which terminates the waveform. No activity occurs in the IDLE state.
- Every time a single-word write waveform is initiated, the GPIF engine cycles through S0, S1, and then stops in S7 (IDLE).

Follow these steps to complete the single-word write waveform.

1. Click the **Single Write** waveform tab.
2. Click on the WEN# trace one clock cycle from the left boundary. This places an action point and creates the WEN# waveform. State 0 (s0) is generated automatically and lasts for one IFCLK cycle (20.83ns).
3. Because the GPIF is writing to the external FIFO, REN# and OE# must be asserted high throughout the waveform to keep the external FIFO from driving its data bus. To ensure this, right-click on the action point on the OE#andREN# trace and select **High (1)**. The waveformsappear as shown in Figure 41.

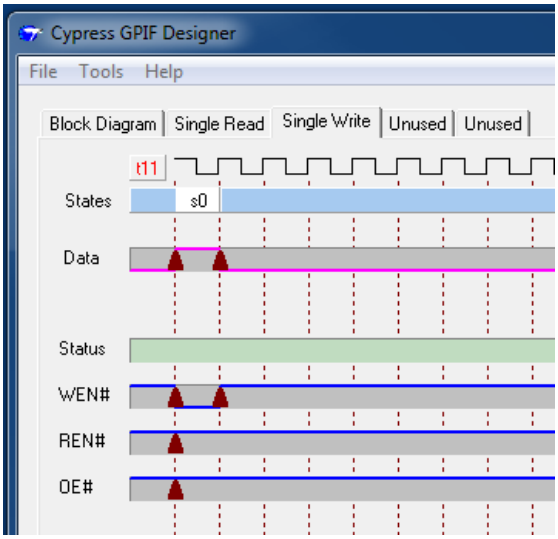
Figure 41. WEN# Pulses Low for One Clock



4. The data bus is to be driven in s0. To do this, right-click on the data action point, and select **Activate Data**.

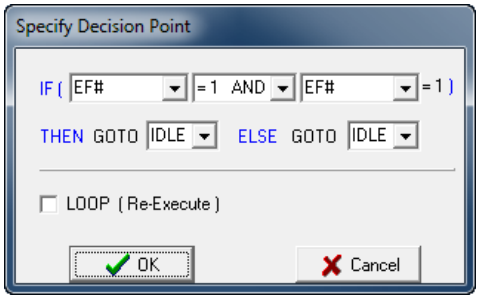
   The data bus should only be drivenfor one clock cycle..To stop driving the data after one clock cycle, place another action point in the Databandone clock after the first. Notice that now the databandis high just for the duration of s0. The waveforms should appear as in Figure 42.

Figure 42. Drive Data for One Clock during S0



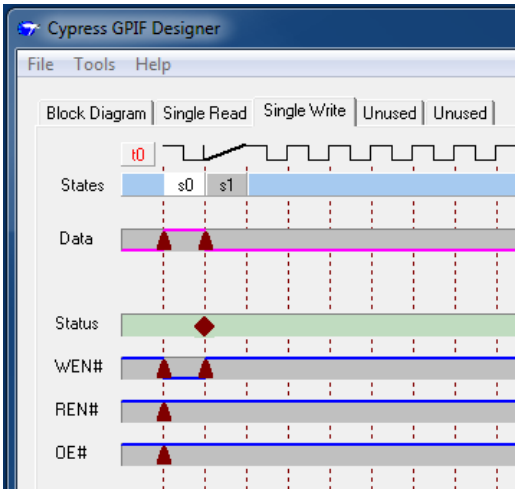
5. Create a decision point state S1 to implement the branch back to the IDLE state. Click in the Status band at the right boundary of S0; then, set an unconditional branch as in Figure 43.

Figure 43. Unconditional Branch to IDLE



The single-word write waveform should now appear as shown inFigure 44.

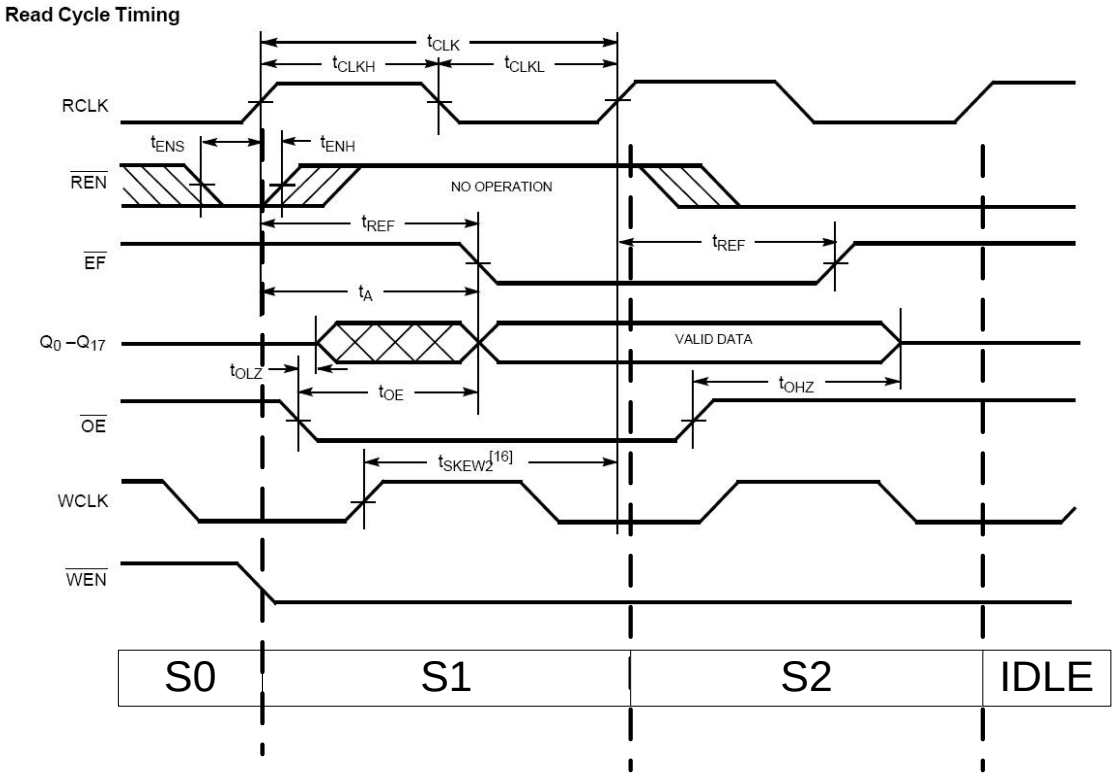
Figure 44. Single-Word Write Waveforms



9.2 Single-WordRead Waveforms

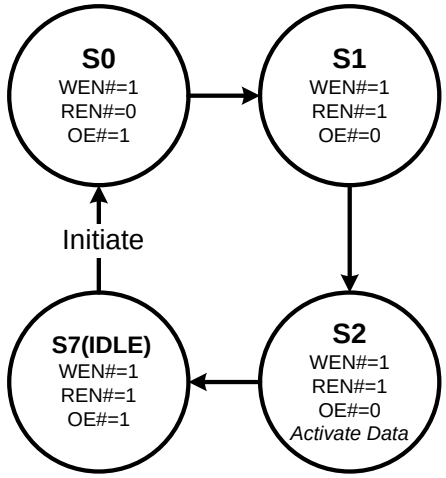
Read waveforms managebyte transfers from the external FIFOinto anFX2LP IN endpoint FIFO. As with the write cycle the GPIF waveforms must satisfy the external FIFO timing requirements. First, review the read cycle timing for the external FIFO, then,create the single-wordread state machine. Figure 45shows the CY7C4265 read timing with GPIF states added to the bottom of the diagram.

Figure 45. Read Cycle Timing Diagram



From the timing information,you can construct the state machine in Figure 46for a byte read transaction.

Figure 46. Byte Read State Diagram



- For the single-word read waveform, REN#is logic LOW in S0 for one IFCLK cycle. This accounts for the datasheet t_{ENS} setup time for the external FIFO before OE# asserts. Deasserting REN# after one clock ensures that the FIFO does not advance more than one word for the read operation.
- S1 then asserts OE# and moves to S2.
- Because it needs to branch, S2 is a decision point state. At the beginning of S1 the data is not yet available from the external FIFO; therefore, the GPIF samplesthe data at the beginning of S2.
- Every time a single-word read waveform is initiated, the GPIF engine cycles through S0, S1, S2, and then S7.

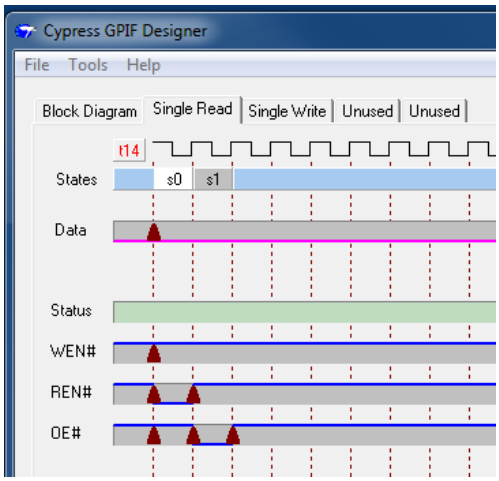
Follow these steps to create the single-word read waveform.

1. Click the **Single Read** waveform tab.
2. Right-click on the first clock in the REN# trace and select **Low(0)**. Click on the REN# trace one clock cycle from the left boundary. This places an action point and creates the REN#pulse. State 0 (s0) is generated automatically and lasts for one IFCLK cycle (20.83ns). Thus, REN#is asserted for 20.83ns (Figure 47).

Figure 47. REN# Asserts for One Clock

3. In the OE# band, place an action point on the right boundary of S0 and another action point one clock later. This automatically deasserts the OE# after S0 and createsstate S1 that lasts for one IFCLK cycle (20.83ns).

Figure 48. OE# Asserts for One Clock



4. Add a decision point (DP) state by clicking the Status band on the clock ending s1. This creates the state S2 and pops up the “Specify Decision Point” dialog box. Configure this decision point as shown in Figure 49 to unconditionally branch to the IDLE state. The waveforms should look similar to Figure 50.

Figure 49. Unconditional Branch to IDLE

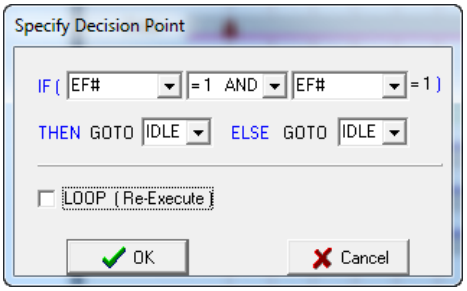
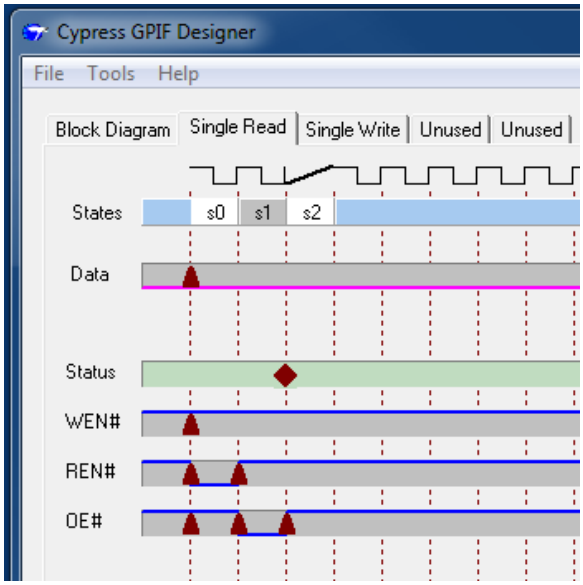
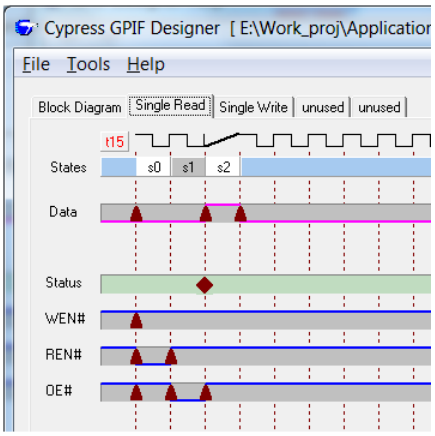


Figure 50. State s2 Added



5. One final adjustment needs to be made. The data must be sampled (read) during S2. To sample the data, place action points in the Data band at the beginning and end of S2. Note that the high level corresponds to sampling, rather than driving the Data band because this is in the Single Read tab. The final single-word read waveformsshould appear as shown Figure 51.
6. Normally, you would save the GPIF Designer project file (*.gpf) and the C file it generates in your Keil project folder. In this case, they are already there as part of this application note code.

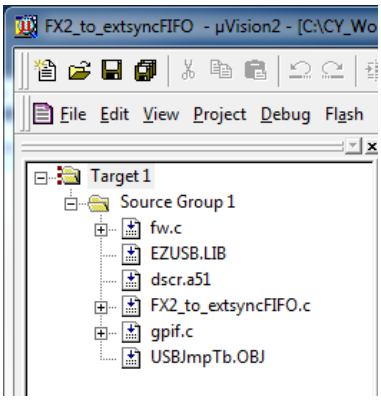
Figure 51. Final Single-Word Read Waveforms



9.3 Firmware Programming for GPIF Single-Word Transactions

After implementing the single-word transaction waveforms in GPIF Designer, the next step is to integrate the USB firmware with the GPIF Designer output. This allows write and read operations to and from the external FIFO over USB. Start with an existing Firmware Frameworks project and add the GPIF management code. Double-click the Keil project file,*FX2_to_extsyncFIFO.uv2*. When opened, the Files panel appears, as shown in Figure 52. Table 2 describes these files.

Figure 52. “FX2_to_extsyncFIFO”Project Files



The *periph.c*has been renamed*FX2_to_extsyncFIFO.c*. In this file, add the USB endpoint and GPIF initialization code to the TD_Init() function, and GPIF management code to the TD_Poll() function.

Table 2. Project File Descriptions

Files	Description
fw.c	Firmware Frameworks, which handle USB requests and calls the task dispatcher TD_Poll().
Ezusb.lib	Collection of functions that handle suspend, resume, I^2C operations, and so on.
USBJmpTb.OBJ	Interrupt vector jump table for USB (INT2) and GPIF/Slave FIFO (INT4) interrupt sources.
Dscr.a51	Device descriptor tables for the FIFO example, which report EP2OUT and EP6IN as the available endpoints for the FX2LP device.
FX2_to_extsyncFIFO.c (renamed from periph.c)	Main user application code where TD_Poll() and TD_Init() are located. You need to modify this file and not update *fw.c*.
Gpif.c	File that contains the GPIF waveform descriptor tables, which implement the Single/FIFO GPIF transaction waveform behavior. This is the C file exported from the GPIF Designer tool.

9.4 Code Snippets

9.4.1 TD_Init()

TD_INIT() does the following:

- Switches the CPU clock speed to 48 MHz (power-on default clock is 12 MHz)
- Configures EP2 as a Bulk OUT endpoint, 4x buffered of size 512
- Configures EP6 as a Bulk IN endpoint, 4x buffered of size 512.
- Configures the FIFOs for manual mode, word-wide operation
- Resets the endpoints using FIFORESET register and arms EP2 OUT endpoint to ensure that it is ready to accept data from USB host (PC).

```
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;
     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
```

- TD_Init then calls the function GPIFInit() which resides in *gpif.c*.
- GPIFInit() loads the GPIF waveform descriptor table into on-chip memory and configures other GPIF registers.
- At any one time,four waveforms can be loaded. If more than four waveforms are required to describe the operation of the physical interface, you will have to manually load another set of four waveforms.
- IFCONFIG register is configured inside GpifInit() that defines the physical interface.
- TD_Init() resets the external FIFO by pulsing PA2 (RS#) as shown in the following code snippet. This ensures that the external FIFO is initialized before commencing data operations.

```
// 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
```

- A USB vendor command 0xB2 is defined in the following code. This allows the host PC to reset the external FIFO by issuing the vendor command.

```
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 Triggering GPIF Single-Word Write Transactions

- The 8051 triggerssingle-word read/single-word write GPIF waveforms by accessing the registers XGPIFSGLDATH, XGPIFSGLDATLX, and XGPIFSGLDATLNOX. This initiates the data transfers.
- To trigger a GPIF single-word write transaction, write to the XGPIFSGLDATH and XGPIFSGLDATLX as follows:

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

- This sets up the MSB and LSB of the word value to be transferred, and writing to the XGPIFSGLDATLX register triggers the single-word write transaction.
- In this example, this is done inside the function GPIF_SingleWordWrite(). It accepts a word value as an input argument and triggers the GPIF single-word write transaction.

```
        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
        }
```

- This function checks to see if the GPIF is in the IDLE state before it launches the transaction by polling the bit GPIFTRIG.7, which is set if the GPIF state machine is in IDLE state. The GPIF should be in the IDLE state before launching any GPIF transaction.
- Note the access sequence to the single-word transaction registers since the endpoint buffer is organized as a FIFO. This sequence ensures that the first byte in the endpoint buffer is written out to FD[7:0], and the second byte is written out to FD[15:8](little endian format).

9.4.3 Triggering GPIF Single-WordReadTransactions

- A GPIF single-word read transaction is triggered by performing a dummy read from the XGPIFSGLDATX register. No data is transferred with this read; it only kicks off the GPIF waveform. After testing the GPIF DONE bit, the 8051 reads the word values in the registers XGPIFSGLDATH and XGPIFSGLDATLNOX.
- In this example, this is done inside the function GPIF_SingleWordRead(). It accepts a word pointer for the destination variable as anargument and performs the GPIF single-word read transaction:

```
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;
}
```

- This function first ensures that the GPIF is in the IDLE state before performing a dummy read from XGPIFSGLDATLX to trigger the GPIFsingle-word read transaction. Then it waits again for the GPIF to be done before reading the registers that contain the word value.

9.4.4 TD_Poll()

- The main application code resides in the function TD_Poll(), which is called repeatedly during device operation.
- Within this function, GPIF_SingleWordWrite()and GPIF_SingleWordRead() functions are called
- GPIF_SingleWordWrite() sends data from EP2OUT to the external FIFO and GPIF_SingleWordRead() reads data from the external FIFO into EP6IN.
- Code that handles USB OUT transfers is as follows:

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

- The EP2 Empty flag is checked in the EP2468STAT register to determine if there is data from the USB host in the EP2OUT endpoint.
- In addition, the FF# flag from External FIFO is checked by accessing the GPIFREADYSTAT register. This ensures that the external FIFO has room for the data to be transferred from EP2OUT to the external FIFO. The 8051 checks the GPIF RDY signal states by accessing the GPIFREADYSTAT register. EXTFIFONOTFULL is a macro for GPIFREADYSTAT AND'ed with bmBIT1.
- If there is data in EP2OUT endpointand the external FIFO is not full, the word variable Tcount is initialized with the count value. The number of bytes transferred from host to EP2OUT is found by accessing the EP2BCH/L registers. Since each GPIF single-word write transaction sends a word to the external FIFO, the number of transactions is half the number of bytes actually contained within the endpoint buffer.
- The "For loop" then calls the GPIF_SingleWordWrite function "Tcount" times and indexes through the endpoint buffer EP2 values, sending data out to the external FIFO one word at a time. Every loop triggers one GPIF single-word write transaction, sending one word of data out to the external FIFO at a time.

- After transferring “Tcount” words of data, the EP2 endpoint is re-armed so that the next USB data packet from the host can be accepted. The FX2LP automatically NAK’s any OUT packets until ready to accept new ones, which causes the host to keep re-trying the OUT transfers.
- Code that handles USB IN Transfers follows:

```
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;
    }
  }
```

- If the in_enable flag is TRUE, it checks to ensure that the EP6IN endpoint buffer is not full and the external FIFO is not empty (EXTFIFONOTEMPTY is a macro for GPIFREADYSTAT and bmBIT0).
- If the EP6IN endpoint buffer is not full and if the external FIFO is not empty, the “for loop” calls the GPIF_SingleWordRead() function. This triggers one GPIF Word Read and stores the result in the Destination address, which is set to the EP6IN FIFO. Each loop iteration increments the destination address, filling the EP6IN FIFO with words retrieved from the external FIFO.
- After the EP6IN FIFO is filled with external FIFO data, the 8051 arms the EP6IN endpoint for transfer to the host PC. Until armed, any host IN requests to EP6 are automatically NAK’ed by the FX2LP USB logic. The 8051 arms an IN endpoint by writing a byte count indicating the number of bytes to transfer in the host IN transfer. Because each GPIF single-word read transaction receives a entire word—two bytes—from the external FIFO, the number of bytes to send to the host is twice the number of GPIF transactions.
- The IN transfers can be enabled or disabled by assigning appropriate values to the “in_enable” flag. Two vendor commands are defined in this example, one to enable and the other to disable IN transfers, as shown in the following code:

```
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;
    }
```

- The IN vendor command, 0xB3, is defined to enable the IN transfers by setting in_enable as TRUE.
- The IN vendor command, 0xB4, is defined to disable the IN transfers by setting in_enable as FALSE.
- Default value of in_enable is FALSE.
- The in_enable flag makes it possible for you to test each read and write operation independently. If it is always enabled, the IN transfercode is processed immediately after the OUT transfer code. By single-stepping the code, you can capture each read/write operation using a logic analyzer for debugging purposes.

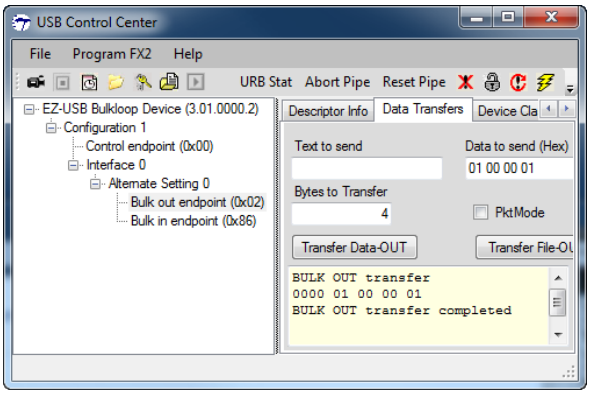
9.5 Running GPIF Single-Word Transaction Example

9.5.1 Without External FIFO

If you do not build the FX2LP development board add-on that contains the external FIFO, you can still observe the GPIF *write* transfers with an oscilloscope. This verifies the USB code that pipes data from the PC into the FX2LP chip and the GPIF waveforms that output the data to the external FIFO.

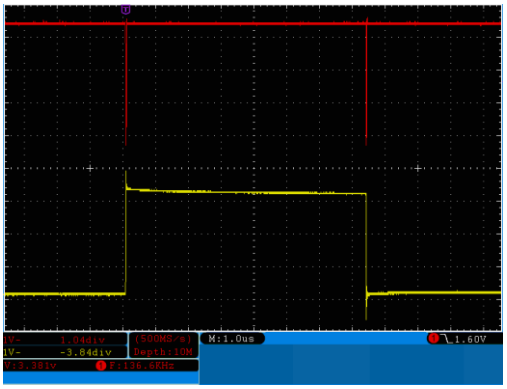
1. Probe the FX2LP development board P2-11 to observe CTL0=WEN# and P1-19 to observe FIFO data bus D0.
2. Launch the USB Control Center as before. Plug the FX2LP development board into a host PC port, and the board should enumerate as the USB loader device as in Figure 21. Otherwise, follow the steps that lead to Figure 21.
3. Select **Program FX2**>**RAM** and navigate to the *FX2_to_extsyncFIFO.hex* file to load it.
4. Expand the Bulkloop device and select **Bulk out endpoint (0x02)**.
5. In the **Data to send (Hex)** textbox enter the number, as shown in Figure 53.
6. Click the **Transfer Data-OUT** button.

Figure 53. USB Control Center Transfers Data



7. The text window in the lower right corner confirms the transfer, and the scope should trigger on the negative edge of the WEN# pulse (Figure 54).

Figure 54. Top: WEN#, Bot: FD[0]



Note that the first FIFO write transfer has FD[0] high and the second has FD[0] low. This confirms that the 16-bit words come out in L-H order; the first word is 0001 and the second word is 0100.

9.5.2 With External FIFO

If you have built and attached the external FIFO board, test the loopback by first transferring USB data OUT, as described in the previous section. Then, perform a Transfer Data-IN operation with the same number of bytes. For this test,*512_count.hex*is used, which contains 512 data bytes.

1. Select **Bulk out endpoint (0x02)** in the tree. Click the **Data Transfers**tab. Press the **Transfer File-OUT** button and select *512_count.hex* in the Keil project folder:`FX2LP Source code and GPIF project files\Firmware\FX2_to_extsyncFIFO GPIF Single Transactions`. Click **Open**; this action sends 512 bytes to the external FIFO.
2. To use IN transfers to read back 512 bytes from the external FIFO, the *in_enable* flag must be set to TRUE by using a USB Vendor Request. Select **Control endpoint (0x00)** in the tree; enter '0xB3' in the **Req code** field. Set **Req Type** as 'Vendor**'**, **Direction** as 'In' and **Bytes to Transfer** as '1'. Click **Transfer Data**.
3. Select **Bulk in endpoint (0x86)** in the tree. Ensure the number of bytes in **Bytes to Transfer** is 512. Press the **Transfer Data-IN** button. You should now see the same 512 bytes of data read back from the FIFO.

9.6 Logic Analyzer Waveforms for Single-Word Transactions

This section shows the timing generated by the GPIF engine for the waveforms as defined by the GPIF Designer.

9.6.1 Single-Word Write Waveform

Figure 55. Single-Word Write Waveforms

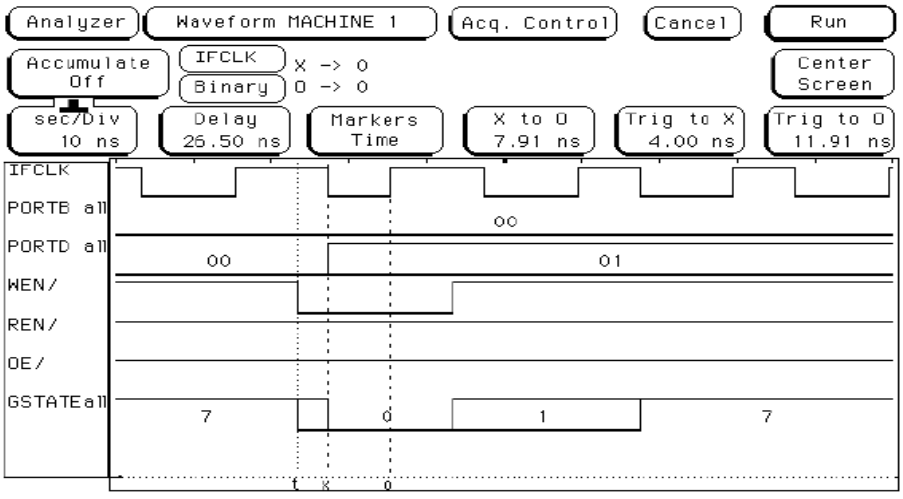


Figure 55shows the timing signals generated by the GPIF engine for the single-wordwrite waveform as defined byGPIF Designer. All the signals are presented here, including GSTATE [2:0], which displays the states the GPIF engine cycles through as it performs the single-wordwrite transaction.

Debug Tip:

- Bringing out the GSTATE signals to the logic analyzer headers allows you to confirm GPIF Designer waveforms with signals generated on the physical interface. This also aids the debugging process because you can verify that the state transitions are correct.
- S0 places the data on the bus (PORTB is FD[7:0] and PORTD is FD[15:8]) and asserts CTL0 (connected to the external FIFO's WEN# line). This writes the 16-bit data value into the external FIFO.
- Note that enough data setup time to the rising edge of IFCLK is provided, because the minimum data setup time for the external FIFO is 4 ns (see the CY7C4265 datasheet).
- S1 is a decision point state that unconditionally branches to the IDLE state to terminate the transaction.
- Without the unconditional branch, the GPIF engine will sequentially move through the remaining states S2-S6before reaching the IDLE state (S7).
- For every word written out in a bulk-OUT transfer, you should see the GPIF engine cycle through S0, S1, and S7.
- To capture the waveform, trigger the logic analyzer on the falling edge of CTL0.
- A sampling rate of 4 ns gives you the resolution shown in the waveform in Figure 55.

9.6.2 Single-Word Read Waveform

Figure 56. Single-Word Read Waveforms

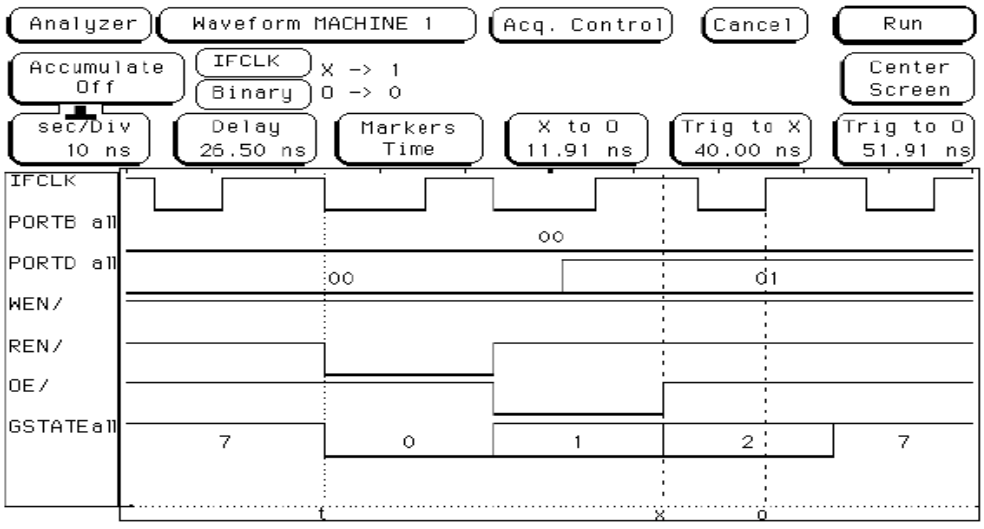


Figure 56 shows the timing generated by the GPIF engine for the single-word read waveforms as defined by GPIF Designer. All the signals are presented here, including GSTATE [2:0], which displays the states the GPIF engine cycles through as it performs the single-wordread transaction.

Debug Tip:

- S0 asserts CTL1 (connected to the external FIFO's REN# line), S1 asserts CTL2 (connected to the external FIFO's OE# line), and S2 samples the data bus (PORTB is FD [7:0] and PORTD is FD[15:8]). This reads the 16-bit data value from the external FIFO.
- Note that enough data setup time to the rising edge of IFCLK is provided, because the minimum data setup time for the GPIF is 9.2 ns (see the FX2LP datasheet).
- S2 is a decision point state that unconditionally branches to the IDLE state to terminate the transaction.

- Without the unconditional branch, the GPIF engine will sequentially move through the remaining states S3-S6 before reaching the IDLE state (S7).
- For every word read out from the external FIFO in a bulk-IN transfer, you should see the GPIF engine cycle through S0, S1, S2, and S7.
- To capture the waveform, trigger the logic analyzer on the falling edge of CTL1.
- A sampling rate of 4 ns will give you the same resolution shown in the waveform in Figure 56.

10 Related Documents

- Getting Started with FX2LP™: This document helps new users to become familiar with FX2LP.
- FX2LP Technical Reference Manual:This document serves as a technical guide for FX2LP. The “General Programmable Interface (GPIF)” chapter explains details of the GPIF.
- The GPIF Designer Utility User Guide:To access this document, downloadthe utility from GPIF Designer. After installation, go to **Help** > **This Tool**.
- EZ-USB FX1-EZ-USB FX2LP Development Kit Quick Start Guide.pdfand EZ-USB Development Kit User Guide.pdf:These documents describe how to use the CY3684 kit and are available (after DVK install)at `C:\Cypress\USB\CY3684_EZ-USB_FX2LP_DVK\1.0\Documentation`.
- Endpoint FIFO Architecture of EZ-USB FX1/FX2LP:This application note helps to understand the data flow inside FX1/FX2LP.

10.1 Other GPIF Examples

More GPIF examples are available in the following Cypress applicationnotes:

- AN57322: Interfacing SRAM with FX2LP over GPIF:This application note describeshow to connectthe CY7C1399BSRAM to FX2LP using a GPIF8-bit asynchronous (no clock) interface. This note is also useful as a guide for connecting FX2LP to other SRAMs.
- AN63787: EZ-USB FX2LP GPIF and Slave FIFO Configuration Examples Using 8-bit Asynchronous Interface: This application note discusses how to configure the GPIF and slave FIFOs of EZ-USB FX2LP in both manual mode and auto mode, to implement an 8-bit asynchronous parallel interface. The design is tested using two interconnected FX2LP development boards, one operating as a GPIF master and the other as a GPIF slave.
- AN4051: FX2LP GPIF Flow State Feature for UDMA:This application noteintroduces the "flow state" feature of the GPIF. This feature extends the GPIF to handle ATAPI UDMA.

10.2 Reference Designs

- CY4611B - USB 2.0 USB to ATA Reference Design: A popular FX2LP application is USB mass storage. The FX2LP GPIF allows glue-less connection to an attached drive. Cypress provides a complete mass storage reference design using FX2LP.

10.3 Datasheets

- EZ-USB FX2LP USB Microcontroller High-Speed USB Peripheral Controller
- CY7C4265 Datasheet

11 Summary

This application note serves as an introduction to EZ-USB FX2LP GPIF. It describes the steps to create a general programmable interface and provides examples that illustrate the main features of GPIF Designer.

Document History

Document Title: AN66806 - Getting Started with EZ-USB® FX2LP™ GPIF

Document Number: 001-66806

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	3152007	CPPK	01/24/2011	Obtain spec. # for note to be added to spec. system. This note had no technical updates.
*A	3174061	CPPK	02/15/2011	Updated Document Title to read “EZ-USB® FX2LP™ GPIF Design Guide”. Replaced screenshots of Ez-USB Control Panel with USB Control Center tool. Removed Unwanted abbreviations like GADFM etc. Included Timing diagrams of the interface between FX2LP and FIFO CY7C4265-15AXC Replaced all references to FX2 with FX2LP. Updated Design Examples: Updated 16-bit Interface to an External Synchronous FIFO CY7C4625-15AC: Updated Running the Example for GPIF Single Transactions: Removed “Exercise bulk loopback function”. Updated Interfacing to a TI 5416 DSP via the Host Port Interface (HPI): Updated Running the DSP Example: Removed “Bootloading the DSP code”.
*B	3772662	GAYA	10/10/2012	Added references to other ANs with examples. Restructured and reorganized various sections.
*C	4121396	RSKV	09/12/2013	Updated FX2LP Architecture Overview. Updated description and figures. Updated General Programmable Interface: Updated GPIF Overview: Updated description. Removed “GPIF Features”. Updated “Design Examples”: Removed “Example 1: Interfacing External Synchronous FIFO with FX2LP over GPIF”. Removed “Example 2: Interfacing SRAM with FX2LP over GPIF”. Added “Example 1: Divide GPIF Clock by 2 and 4”. Added “Example 2: Divide GPIF Clock by 7”. Added “Example 3: Use Single Word Read/Write Transactions”. Updated attached associated project: Changed the attachment file name from “Project” to “FX2LP Source code and GPIF project files”. Added “GPIF Clock Divider” folderunder “Firmware” folder in the attachment.
*D	4235014	GAYA	01/02/2014	No technical updates. Completing Sunset Review.
*E	5141487	GAYA	06/15/2016	Added reference to code examples above Abstract. Updated attached associated project: Removed “Hardware” folder. Updated to new template.
*F	5624132	GAYA	02/08/2017	Updated to new template. Completing Sunset Review.
*G	5687831	AESATMP7	04/19/2017	Updated Cypress Logo and Copyright.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

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

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

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