CEUSB2 GENERAL PURPOSE FIRMWARE INTERFACE SPECIFICATIONS

Ver2.1 – Sep. 16, 2003

Firmware Interface Specifications Document for Software Programmers

CONTENTS

CONTENTS 1
1 INTRODUCTION 2
1.1 About This Documentation 2
1.2 Prerequisites 2
1.3 Distribution 2
1.4 Firmware Interface Overview 3
2 USING GENERIC FIRMWARE INTERFACE 3
3 GENERIC FIRMWARE INTERFACE SOFTWARE REFERENCE 6
3.1 Generic Firmware Interface Vendor Commands 6
3.1.1 CEUSB2_CMD_READ_WRITE_INTRAM 6
3.1.2 CEUSB2_CMD_GET_FIRMWARE_INFO 6
3.1.3 CEUSB2_CMD_IS_HIGH_SPEED 6
3.1.4 CEUSB2_CMD_FPGA_INIT 7
3.1.5 CEUSB2_CMD_FPGA_GET_STATUS 7
3.1.6 CEUSB2_CMD_FPGA_RESET 8
3.1.7 CEUSB2_CMD_FPGA_SEND_CONFIG_DATA 8
3.1.8 CEUSB2_CMD_FW_REINIT 8
3.1.9 CEUSB2_CMD_SOFT_RESET 9
3.1.10 CEUSB2_CMD_CPU_SETTING 9
3.1.11 CEUSB2_CMD_GPIF_SETTING 9
3.1.12 CEUSB2_CMD_GPIF_GET_STATUS 10
3.1.13 CEUSB2_CMD_GPIF_ABORT 10
3.1.14 CEUSB2_CMD_GPIF_READ_WRITE_INIT_DATA 11
3.1.15 CEUSB2_CMD_GPIF_READ_WRITE_WAVEFORM 11
3.1.16 CEUSB2_CMD_GPIF_SINGLE_READ_WRITE 11
3.1.17 CEUSB2_CMD_GPIF_SINGLE_READ_WRITE_WORDWIDE 12
3.1.18 CEUSB2_CMD_READ_WRITE_EXTRAM 12
3.1.19 CEUSB2_CMD_READ_WRITE_SERIAL 12
3.1.20 CEUSB2_CMD_READ_WRITE_EEPROM 13
3.2 Generic Firmware Interface Structures 13
3.2.1 CEUSB2_FIRMWARE_INFO 13
4 GENERIC FIRMWARE GPIF INTERFACE 14
4.1 GPIF Single Write 14
4.2 GPIF Single Read 14
4.3 GPIF FIFO Write 15

4.4 GPIF FIFO Read 16
4.5 GPIF Interface Timing 16
5 GPIF INTERFACE DESIGN EXAMPLE WITH VHDL 18

1 INTRODUCTION

1.1 About This Documentation

This documentation is the specifications for the generic firmware interface for CeUsb2 type of devices. This interface is generic for all other firmwares supplied by Cesys GmbH.

1.2 Prerequisites

An intermediate or upper knowledge of C/C++ programming is required to be able to follow the example codes given in this documentation.

CeUsb2 boards are USB2.0 devices, so it is necessary to understand basics of the USB protocol in order to understand the USB communication and data transfer operations.

Some CeUsb2 type of devices has an on-board FPGA chip. This document references some sample hardware designs for the FPGA written in VHDL hardware description language. Therefore, an intermediate or upper level of VHDL language is also required.

1.3 Distribution

After you install CeUsb2 software, you can find the header file for generic firmware interface, CeFirmware.h, in CeUsb2\inc directory. It includes all vendor commands and structures necessary for programming.

1.4 Firmware Interface Overview

CeUsb2 devices are able to run different firmwares with different features such as USB configurations, interfaces, endpoints; data transfer types (bulk, isochronous), internal data transfer logic (e.g., using GPIF or not), interfaces to external components (FPGA, CPLD, DSP etc.). All the firmware applications developed by Cesys use the generic firmware interface which defines the minimum required functionality for the firmwares, except large data transfers performed with bulk or isochronous pipes. Supported USB interfaces with their alternate settings, USB pipes and their transfer logic are explained in a separate documentation for each firmware executable. Firmwares which needs other vendor or class requests extends this generic firmware interface in a separate header file.

2 USING GENERIC FIRMWARE INTERFACE

Communication between the firmware and user mode APIs and applications with vendor or class requests are performed through the CeUsb2 drivers. IOCTL_CEUSB2_VENDOR_OR_CLASS_REQUEST and IOCTL_CEUSB2_VENDOR_REQUEST I/O codes can be used to perform vendor or class requests. For more information about CeUsb2 IOCTL interface and vendor or class requests check the CeUsb2 driver specification documentation.

IOCTL_CEUSB2_VENDOR_OR_CLASS_REQUEST code together with the structure VENDOR_OR_CLASS_REQUEST_CONTROL is used to perform a vendor or class specific request which is directed to a device, interface, endpoint or another device-defined target.

IOCTL_CEUSB2_VENDOR_REQUEST code together with the structure CEUSB2_VENDOR_REQUEST_CONTROL is used to perform vendor requests which are directed to the device only. The entire request codes defined in the generic firmware is vendor requests and the recipient is the device itself. Therefore, vendor request I/O is enough to use all the functionalities of this interface.

IOCTL_CEUSB2_VENDOR_REQUEST is called with DeviceIoControl function of Win32 SDK with a pointer to CEUSB2_VENDOR_REQUEST structure as input buffer. DeviceIoControl function is defined as follows.

```
BOOL DeviceIoControl(
        HANDLE hDevice,             // handle to device
        DWORD dwIoControlCode,      // operation
        LPVOID lpInBuffer,          // input data buffer
```

```
        DWORD nInBufferSize,        // size of input data buffer
        LPVOID lpOutBuffer,         // output data buffer
        DWORD nOutBufferSize,       // size of output data buffer
        LPDWORD lpBytesReturned,    // byte count
        LPOVERLAPPED lpOverlapped   // overlapped information
        );
```

For vendor requests dwIoControlCode is always IOCTL_CEUSB2_VENDOR_REQUEST and lpOverlapped is NULL since CeUsb2 doesn't support asynchronous device I/O for vendor requests.

hDevice is a 32 bit handle of previously opened CeUsb2 device.

lpInBuffer is a pointer to a CEUSB2_VENDOR_REQUEST structure which holds information about a single vendor request and nInBufferSize is the size of this structure.

lpOutBuffer is the buffer for the data which will be transferred either from the host to the CeUsb2 or from CeUsb2 to host. This buffer can be NULL if no data is transferred. nOutBufferSize is the amount of data transferred by this vendor request.

lpBytesReturned is the actual amount of data transferred after the DeviceIoControl function returns.

DeviceIoControl function returns a nonzero value (TRUE) if it succeeds; otherwise it returns 0 (FALSE). Like most other Windows functions, the reason for the error can be retrieved with GetLastError function.

Code example 2.1 shows how to use the vendor request CEUSB2_CMD_GET_FIRMWARE_INFO to get versioning information from the firmware and CEUSB2_CMD_IS_HIGH_SPEED to determine if the device is a high speed or full speed CeUsb2 device. OpenDevice() and CloseDevice() functions simply open and close a handle to the device. For more information about opening and closing a CeUsb2 devices check CeUsb2 driver specifications documentation.

```
// Code example 2.1
// CeUsb2 vendor command test

// 32 bit handle of the device
// set by OpenDevice(), and cleared by CloseDevice()
HANDLE ghDevice;

DWORD FirmwareTest(void)
{
        DWORD BytesTransfered = 0; // actual number of bytes transferd
        CEUSB2_VENDOR_REQUEST_CONTROL VenRequest; // vendor request control structure
```

```
	CEUSB2_FIRMWARE_INFO FwInfo; // firmware information structure
	UCHAR IsHighSpeed; // device speed infirmation

	DWORD dwStatus = OpenDevice(); // open the device
	if(!CE_SUCCESS(dwStatus))
		return dwStatus; // error opening the device

	// set up the firmware information structure to retrieve firmware versioning information
	VenRequest.Request = CEUSB2_CMD_GET_FIRMWARE_INFO;
	VenRequest.Direction = 1; //input
	VenRequest.Value = 0; // this request doesn't use Value and Index fields
	VenRequest.Index = 0;

	// call the driver
	if(!DeviceIoControl(
		ghDevice,		// handle to device
		IOCTL_CEUSB2_VENDOR_REQUEST,	// operation
		&VenRequest,		// input data buffer
		sizeof(CEUSB2_VENDOR_REQUEST_CONTROL),	// size of input data buffer
		& FwInfo,		// output data buffer
		sizeof(CEUSB2_FIRMWARE_INFO),	// size of output data buffer
		&BytesTransfered,	// actual amount of data transferred
		NULL	// overlapped information
		))return GetLastError();

	// display the version of this firmware
	printf("Firmware version : %u.%u\n", FwInfo.MajorVersion, FwInfo.MinorVersion);

	VenRequest.Request = CEUSB2_CMD_IS_HIGH_SPEED;
	VenRequest.Direction = 1; //input
	VenRequest.Value = 0; // this request doesn't use Value and Index fields
	VenRequest.Index = 0;

	// call driver
	if(!DeviceIoControl(
		ghDevice,		// handle to device
		IOCTL_CEUSB2_VENDOR_REQUEST,	// operation
		&VenRequest,		// input data buffer
		sizeof(CEUSB2_VENDOR_REQUEST_CONTROL),	// size of input data buffer
		& IsHighSpeed,		// output data buffer
		sizeof(UCHAR),		// size of output data buffer
		&BytesTransfered,	// actual amount of data transferred
		NULL	// overlapped information
		))return GetLastError();

	// display device speed information
	printf("Device is %s \n", (IsHighSpeed == 0) ? "full speed" : "high speed");

	//close the device
	CloseDevice();
}
```

3 GENERIC FIRMWARE INTERFACE SOFTWARE REFERENCE

3.1 Generic Firmware Interface Vendor Commands

3.1.1 CEUSB2_CMD_READ_WRITE_INTRAM

Direction IN/OUT
Buffer Executable data extracted from a hex file or buffer which will hold the internal RAM data received from the firmware.
Buffer Length Number of data bytes.
Value Starting address (offset) to read or write.
Index 0
Bytes returned Actual number of bytes downloaded to or read from the internal RAM.
Description Reads or writes internal RAM of the USB controller chip.
This vendor request is used for downloading executable firmware code on the 8051 CPU of FX2.
Notes This request code is from Cypress Semiconductors.

3.1.2 CEUSB2_CMD_GET_FIRMWARE_INFO

Direction IN
Buffer CEUSB2_FIRMWARE_INFO structure.
Buffer Length 3 (size of CEUSB2_FIRMWARE_INFO structure)
Value 0
Index 0
Bytes returned 3 (size of CEUSB2_FIRMWARE_INFO structure)
Description Gets firmware versioning information with major and minor version.
Notes This vendor command must be implemented in all firmwares.

3.1.3 CEUSB2_CMD_IS_HIGH_SPEED

Direction IN
Buffer 1 byte unsigned character (USB speed information).
Buffer Length 1
Value 0

Index 0
Bytes returned 1
Description Checks if the USB device is a high speed or a full speed device. If the returned unsigned character value is 0 then it is a full speed device; otherwise it is a high speed device.
Notes This vendor command must be implemented in all firmwares.

3.1.4 CEUSB2_CMD_FPGA_INIT

Direction IN
Buffer 1 byte unsigned character (FPGA status).
Buffer Length 1
Value 0
Index 0
Bytes returned 1
Description Initializes the FPGA device for configuration. Returned buffer (unsigned character) is the status of the FPGA after initialization process is finished. Possible status values are:

0x00 - FPGA initialization is successful
0xAB - FPGA initialization timed out
others - Unknown FPGA status.

This request actually clears the configuration of the FPGA so that it can accept new configurations.
Notes This vendor command might be implemented in firmwares which have an FPGA, CPLD or ASIC interface

3.1.5 CEUSB2_CMD_FPGA_GET_STATUS

Direction IN
Buffer 1 byte unsigned character (FPGA status).
Buffer Length 1
Value 0
Index 0
Bytes returned 1
Description This request must be called after the initialization of the FPGA is completed successfully and all configuration data is sent to the FPGA. After all these processes are done, it checks whether the FPGA is configured correctly or not. Returned buffer (unsigned character) is the status of the FPGA. Possible status values are:

0x00 - FPGA configuration is successful.
0xAB - FPGA configuration is unsuccessful (timed out).
others - Unknown FPGA status.

Notes This vendor command might be implemented in firmwares which have an FPGA, CPLD or ASIC interface.

3.1.6 CEUSB2_CMD_FPGA_RESET

Direction None (Use OUT).
Buffer None
Buffer Length 0
Value 0
Index 0
Bytes returned 0
Description Resets the FPGA design. Actually sends a reset signal to the FPGA. Its effects are FPGA design specific.
Notes This vendor command might be implemented in firmwares which have an FPGA, CPLD or ASIC interface and runs a design which needs an external reset signal.

3.1.7 CEUSB2_CMD_FPGA_SEND_CONFIG_DATA

Direction OUT
Buffer FPGA configuration data buffer.
Buffer Length FPGA configuration data buffer length.
Value 0
Index 0
Bytes returned Actual number of configuration bytes sent to the FPGA.
Description Most of the CeUsb2 devices use GPIF interface with Bulk or Isochronous pipes for FPGA configuration. Some however, use Endpoint 0 (control pipe) to transfer configuration data and implements this vendor command. For more information about FPGA configuration process, check general CeUsb2 design guide, CeUsb2dg.pdf.
Notes This vendor command might be implemented in firmwares which have an FPGA, CPLD or ASIC interface and do not use Bulk or Isochronous pipes for FPGA configuration.

3.1.8 CEUSB2_CMD_FW_REINIT

Direction None (Use OUT)
Buffer None
Buffer Length 0
Value 0
Index 0
Bytes returned 0

Description Reinitializes the firmware. It directs the code execution of the firmware to its init function.
Notes This command is optional.

3.1.9 CEUSB2_CMD_SOFT_RESET

Direction None (Use OUT)
Buffer None
Buffer Length 0
Value 0
Index 0
Bytes returned 0
Description Performs a soft reset in the firmware (vector to org 00h).
Notes This command is optional.

3.1.10 CEUSB2_CMD_CPU_SETTING

Direction IN/OUT
Buffer 1 byte unsigned character (CPU register).
Buffer Length 1
Value 0
Index 0
Bytes returned 0 when OUT, else 1.
Description Sets\Gets the FX2 CPU clock parameters. CPU register bits are:
4-3 = clock speed selection (00 = 12 MHz, 01 = 24 MHz, 10 = 48 MHz, 11 = invalid)
2 = CPU clock invert (if 1, CPU clock is inverted)
1 = CPU clock output enable (0 = disable, 1 = enable)
Bits 0, 5, 6 and 7 are not used.
If it is used to set CPU parameters, changes will not take effect until the CEUSB2_CMD_FW_REINIT command is called.
Notes This command is optional.

3.1.11 CEUSB2_CMD_GPIF_SETTING

Direction IN/OUT
Buffer 1 byte unsigned character (GPIF register).
Buffer Length 1
Value 0
Index 0

Bytes returned 0 when OUT, else 1.
Description Sets\Gets the GPIF parameters. GPIF register bits are:

7 = selects interface clock source (0 = external, 1 = internal)
6 = interface clock speed (0 = 30, 1 = 48 MHz)
5 = interface clock output enable (0 = disable, 1 = enable)
4 = if 1, polarity of the interface clock is inverted
1 = if 1, GPIF auto mode enabled
0 = GPIF interface word wide bit (0 = 8 bit data, 1 = wordwide 16 bit data)
Bits 2 and 3 are not used.

If it is used to set GPIF parameters, changes will not take effect until the CEUSB2_CMD_FW_REINIT command is called.
Notes This command may be implemented in the firmwares which use the GPIF.

3.1.12 CEUSB2_CMD_GPIF_GET_STATUS

Direction IN
Buffer 1 byte unsigned character (GPIF status).
Buffer Length 1
Value 0
Index 0
Bytes returned 1
Description Retrieves the current status of the GPIF. GPIF Status register's bits are:

7 = GPIF done bit (0 = GPIF busy, 1 = GPIF idle)
5-0 = Instantaneous states of the RDY pins. The current state of the RDY[5:0] pins, sampled at each rising edge of the GPIF clock.
Bit 6 is not used.

Notes This command may be implemented in the firmwares which use the GPIF.

3.1.13 CEUSB2_CMD_GPIF_ABORT

Direction None (Use OUT)
Buffer None
Buffer Length 0
Value 0
Index 0
Bytes returned 0
Description Aborts the current GPIF waveform transition.
Notes This command may be implemented in the firmwares which use the GPIF.

3.1.14 CEUSB2_CMD_GPIF_READ_WRITE_INIT_DATA

Direction IN/OUT
Buffer 7 bytes unsigned character array (GPIF Init data).
Buffer Length 7
Value 0
Index 0
Bytes returned 0 when OUT, else 7.
Description Sets\Gets the 7-byte GPIF init data array which used for waveform programming. If it is used to set GPIF init data, changes will not take effect until the CEUSB2_CMD_FW_REINIT command is called.
Notes This command may be implemented in the firmwares which use the GPIF.

3.1.15 CEUSB2_CMD_GPIF_READ_WRITE_WAVEFORM

Direction IN/OUT
Buffer 1-128 bytes unsigned character array (GPIF waveform data).
Buffer Length 1-128
Value Waveform data offset.
Index 0
Bytes returned 0 when OUT, else 1 to 128.
Description Sets\Gets the 128-byte GPIF waveform data (or a portion of it) used for waveform programming. If it is used to set GPIF waveform data, changes will not take effect until the CEUSB2_CMD_FW_REINIT command is called.
Notes This command may be implemented in the firmwares which use the GPIF.

3.1.16 CEUSB2_CMD_GPIF_SINGLE_READ_WRITE

Direction IN/OUT
Buffer Data buffer.
Buffer Length Data buffer length.
Value Offset (address) to read or write.
Index Mode variable.
Bytes returned Actual number of bytes read or written.
Description Reads and writes specified amount of data with GPIF single byte transitions (8 bits data). Mode variable determines addressing logic. If mode is 0 address is incremented after each access. Otherwise, read/write operation is performed with the same address value.
Notes This command is optional.

3.1.17 CEUSB2_CMD_GPIF_SINGLE_READ_WRITE_WORDWIDE

Direction IN/OUT
Buffer Data buffer
Buffer Length Buffer length
Value Offset (address) to read or write
Index Mode variable
Bytes returned Actual number of bytes read or written.
Description Reads and writes specified amount of data with GPIF single word transitions (16 bits data). Mode variable determines addressing logic. If mode is 0 address is incremented after each access (once for one word). Otherwise, read/write operation is performed with the same address value.
Notes This command is optional.

3.1.18 CEUSB2_CMD_READ_WRITE_EXTRAM

Direction IN/OUT
Buffer External RAM data buffer
Buffer Length Buffer length
Value Address (offset) to read or write
Index 0
Bytes returned Actual number of bytes read or written.
Description Reads and writes specified amount of data to or from the external RAM of the USB controller chip. It is mostly use to write executable code data to the external RAM is the CeUsb2 device has one. Currently this vendor command is implemented only in the loader firmware (CeU2Ld.hex).
Notes This optional vendor command may be implemented in firmwares which need external RAM data access.

3.1.19 CEUSB2_CMD_READ_WRITE_SERIAL

Direction IN/OUT
Buffer Serial I/O data buffer.
Buffer Length Data buffer length.
Value When writing, this value is the "mode" parameter. Otherwise 0,
Index 0
Bytes returned Actual amount of data read or written over serial I/O.
Description Reads and writes specified amount of data over serial (RS232) interface. When writing, if mode is 0, then the characters are sent over serial line as they are without any change. If mode is other than 0, characters are sent in hexadecimal format with a space character between two of them. This feature is used for dumping buffers.

Notes This optional vendor command may be implemented in firmwares which implements RS232 interface.

3.1.20 CEUSB2_CMD_READ_WRITE_EEPROM

Direction IN/OUT
Buffer EEPROM data buffer
Buffer Length Buffer length
Value Address (offset) to read or write
Index 0
Bytes returned Actual number of bytes read or written.
Description Reads and writes specified amount of data to or from the EEPROM found on CeUsb2 boards. Normally EEPROM includes initial configuration data for the board, and should be changed by the loader driver (CeUsLd.sys) only. Currently this vendor command is implemented only in the loader firmware (CeU2Ld.hex).
Notes This optional vendor command may be implemented in firmwares which needs EEPROM data access.

3.2 Generic Firmware Interface Structures

3.2.1 CEUSB2_FIRMWARE_INFO

```
typedef struct _CEUSB2_FIRMWARE_INFO
{
        unsigned char       MajorVersion;
        unsigned char       MinorVersion;
        unsigned char       Reserved;
}CEUSB2_FIRMWARE_INFO,*PCEUSB2_FIRMWARE_INFO;
```

Description: This structure holds versioning information for a CeUsb2 firmwawre and used with CEUSB2_CMD_GET_FIRMWARE_INFO vendor request. It is shared between the firmware application and the user mode applications and APIs.
Members:
MajorVersion - 8051 firmware executable major version.
MinorVersion - 8051 firmware executable minor version.
Reserved - Reserved for future use.

4 GENERIC FIRMWARE GPIF INTERFACE

Generic firmware handles GPIF single read and write transactions over control pipe with USB vendor commands and GPIF FIFO read and write transactions over bulk pipes with USB bulk transfers.

Following sections explains the interface timing between the GPIF and an external peripheral for all four different data transfer types. Last section (4.5) shows the exact timing requirements of the interface.

4.1 GPIF Single Write

Generic firmware doesn't use any ready signals for GPIF Single Write transactions. This command is issued in two steps as follows:

i -) GPIF activates single write control pin (CTL3) for one clock cycle and applies data and address on the bus.

Ii -) GPIF deactivates single write control pin.

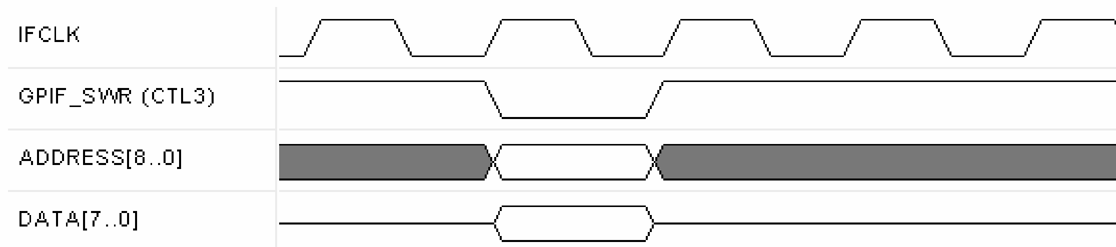

Figure 4.1 - GPIF Single Write Timing diagram

4.2 GPIF Single Read

Generic firmware doesn't use any ready signals for GPIF Single Read transactions. This command is issued in three steps as follows:

i -) GPIF activates single read control pin (CTL4) for one clock cycle and applies address on the bus.

ii -) GPIF deactivates single write control pin.

iii-) GPIF reads the data from the data bus.

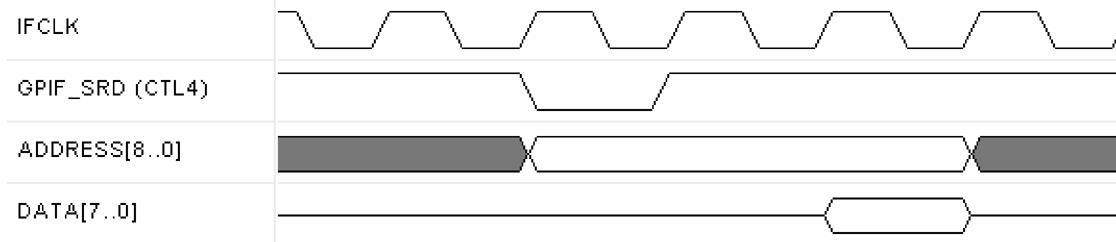

Figure 4.2 - GPIF Single Read Timing diagram

4.3 GPIF FIFO Write

Generic firmware doesn't use address bus for GPIF FIFO Write transactions. This command is issued in four steps as follows:

i -) User sends data over USB bulk pipes. GPIF checks the associated ready signal for writing (RDY1 for generic firmware). If it is high it starts writing.

ii -) GPIF activates FIFO write control pin (CTL2) for one clock cycle and applies the next data on the bus.

iii -) GPIF deactivates single write control pin.

iv -) It completes if the requested amount of data by the user is sent to the peripheral, otherwise jumps to step ii after one clock idle time.

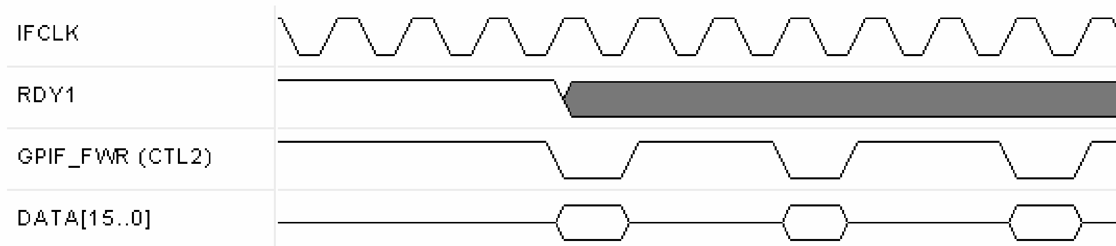

Figure 4.3 - GPIF FIFO Write Timing diagram

4.4 GPIF FIFO Read

Generic firmware doesn't use address bus for GPIF FIFO Read transactions. This command is issued in four steps as follows:

i -) GPIF checks the associated ready signal for reading (RDY0 for generic firmware). If it is high it starts the process.

ii -) GPIF activates FIFO read control pin (CTL1) .

iii -) GPIF stays idle.

iv -) GPIF deactivates single read control pin. At the same time it reads the data on the bus and puts it into the GPIF FIFOs, from which the user can read over USB bulk pipes.

v -) It completes if the GPIF FIFOs are full, otherwise jumps to step ii after one clock idle time.

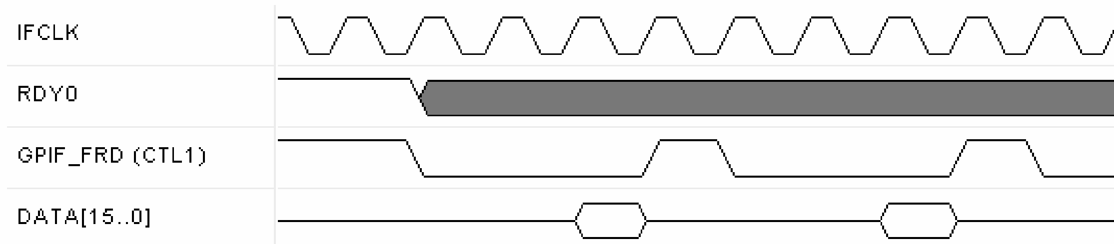

Figure 4.4 - GPIF FIFO Read Timing diagram

4.5 GPIF Interface Timing

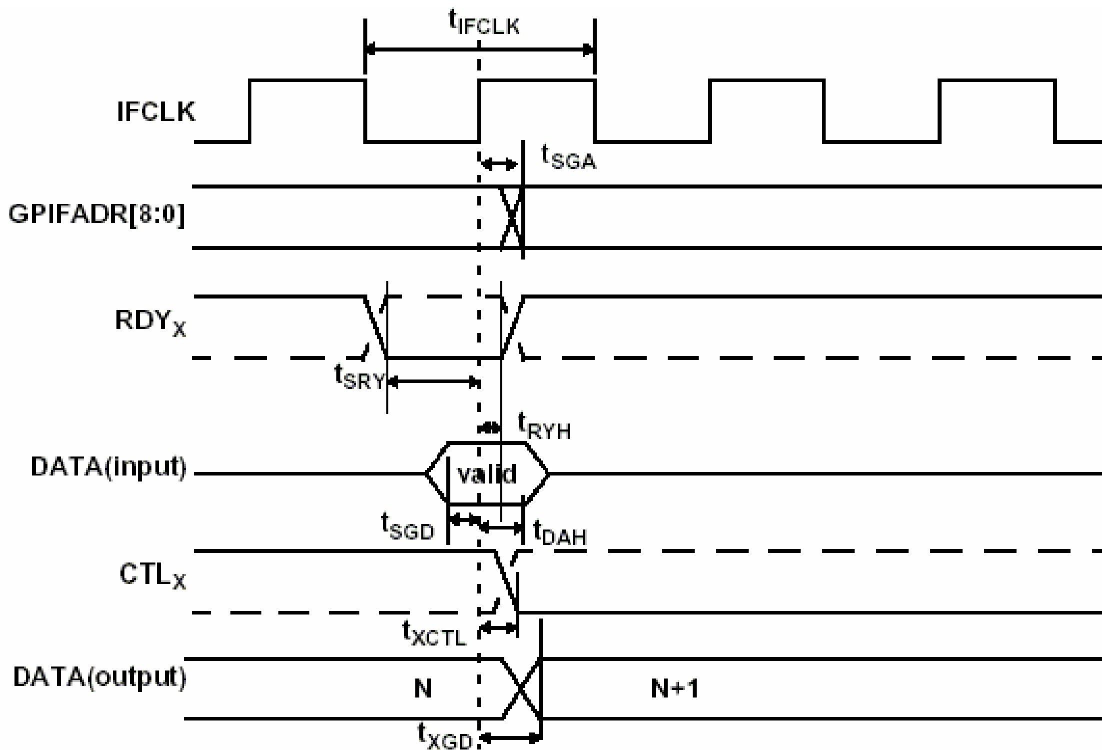

Figure 4.5 - GPIF Synchronous Signals Timing Diagram

Parameter	Description	Min. (ns)	Max. (ns)
tIFCLK	IFCLK Period	20.83	
tSRY	RDYX to Clock Set-up Time	8.9	
tRYH	Clock to RDYX	0	
tSGD	GPIF Data to Clock Set-up time	9.2	
tDAH	GPIF Data Hold Time	0	
tSGA	Clock to GPIF Address Propagation Delay		7.5
tXGD	Clock to GPIF Data Output Propagation Delay		11
tXCTL	Clock to CTLX Output Propagation Delay		6.7

Table 4.1 - GPIF Synchronous Signals Parameters with Internally Sourced IFCLK

Parameter	Description	Min. (ns)	Max. (ns)
tIFCLK	IFCLK Period	20.83	200
tSRY	RDYX to Clock Set-up Time	2.9	
tRYH	Clock to RDYX	3.7	

tSGD	GPIF Data to Clock Set-up time	3.2	
tDAH	GPIF Data Hold Time	4.5	
tSGA	Clock to GPIF Address Propagation Delay		11.5
tXGD	Clock to GPIF Data Output Propagation Delay		15
tXCTL	Clock to CTLX Output Propagation Delay		10.7

Table 4.2 - GPIF Synchronous Signals Parameters with Internally Sourced IFCLK

5 GPIF INTERFACE DESIGN EXAMPLE WITH VHDL

This section gives a generic VHDL application, to show how GPIF is connected to an FPGA chip. This design meets all functional and timing requirements of the generic firmware interface.
Code example 5.1 is the VHDL design itself. You can find the sources also in the distribution CD under ..\fpga\gpif_test directory. Also a graphical test program GpifTest is available in the distribution with the whole C++ source codes.

Code example 5.2 is a part of the constraints file (gpif_test.ucf) written for this design. It only contains the most important parts of the file which requires more attention. Notice that it is very important to write a constraints file with respect to the timing requirements stated in section 4.5.

Figure 5.1 shows the pin connections between FX2 and FPGA for this design. IFCLK (GPIF interface clock, named as SCLK in VHDL sample) can be either 38.4 or 48 MHz. It is configurable through the firmware. Data bus can be configured as 8-bit or 16-bit (WORDWIDE), through a register defined in the design (Address 0x0100).

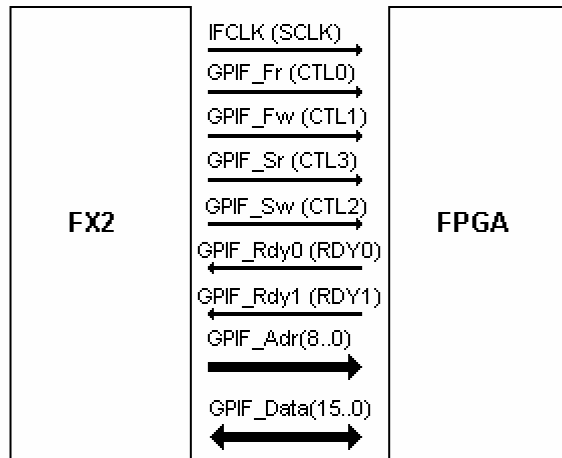

Figure 5.1 – "gpif_test " sample design's pin connections

```
-- Code example 5.1, gpif_test.vhd
-- VHDL design code for GPIF interface

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- synopsys translate_off
Library XilinxCoreLib;
-- synopsys translate_on

entity GPIF_test is

Port (
SCLK : in std_logic; -- system clock, connected to GPIF interface clock
RESET : in std_logic; -- reset signal

-- gpif control signals
GPIF_Sr : in std_logic; -- single read strobe from GPIF
GPIF_Sw : in std_logic; -- single write strobe from GPIF
GPIF_Fw : in std_logic; -- fifo write strobe from GPIF
GPIF_Fr : in std_logic; -- fifo read strobe from GPIF

-- gpif ready signals
GPIF_Rdy0 : out std_logic; -- gpif can read data
GPIF_Rdy1 : out std_logic; -- gpif can write data

-- gpif address and data busses
GPIF_Adr : in std_logic_vector(8 downto 0); -- 8 bit gpif address bus
GPIF_DataL : inout std_logic_vector(7 downto 0); -- 8 bit bidirectional gpif data bus (lower 8 bits)
GPIF_DataH : inout std_logic_vector(7 downto 0); -- 8 bit bidirectional gpif data bus (higher 8 bits), active
                                                  -- only for gpif wordwide mode

Test : out std_logic_vector(1 downto 0); -- test signals
```

```
 Leds : out std_logic_vector(1 downto 0) -- test leds on the board

 );
end GPIF_test;

architecture Bhv of GPIF_test is

-- internal fifo for data loop-back, generated with Core Generator
component fifo8x2047
		port (
		din: IN std_logic_VECTOR(7 downto 0); -- input data
		wr_en: IN std_logic; -- write enable signal
		wr_clk: IN std_logic; -- write clock
		rd_en: IN std_logic; -- read enable signal
		rd_clk: IN std_logic; -- read clock
		ainit: IN std_logic; -- reset (init) signal
		dout: OUT std_logic_VECTOR(7 downto 0); -- output data
		full: OUT std_logic; -- fifo full signal
		empty: OUT std_logic; -- fifo empty signal
		rd_count: OUT std_logic_VECTOR(1 downto 0) -- data count signal
		);
end component;

-- synopsys translate_off
-- here comes some codes for simulation only, look at the sources
-- synopsys translate_on

-- versioning constants of this design
constant MajorVersion : std_logic_vector(7 downto 0) := "00000010";
constant MinorVersion : std_logic_vector(7 downto 0) := "00000000";

-- signals
signal FifoFull : std_logic; -- internal fifo full signal
signal FifoEmpty : std_logic; -- internal fifo empty signal
signal FifoWr : std_logic; -- internal fifo write signal
signal FifoRd : std_logic;  -- internal fifo read signal
signal FifoOutData : std_logic_vector(7 downto 0); -- internal fifo ouput data
signal FifoOutData2 : std_logic_vector(7 downto 0); -- internal fifo ouput data
signal FifoRdCount : std_logic_vector (1 downto 0); -- internal fifo count signal

signal CounterValue : std_logic_vector(15 downto 0); -- 16 bit counter value

-- counter command register
-- bit 0 : start/stop counter ('1' - start, '0' - stop)
-- bit 1 : counter direction  ('1' - up, '0' - down)
signal CounterCommand : std_logic_vector(1 downto 0);

-- Setting this bit enables readback test
-- Clearing this bit enables dummy read write test
signal TestMode : std_logic;


type ST_TYPE is (sIDLE,sREAD_FIFO,sDRIVE_DATA);
signal State : ST_TYPE;

begin
```

```
Test(1) <= FifoRd;
Test(0) <= FifoWr;

-- state machine process for GPIF single and fifo read operations
read_sm_proc: process (SCLK,RESET)
variable latched : boolean;
begin
	if RESET = '1' then
		GPIF_DataL <= (others => 'Z');
		GPIF_DataH <= (others => 'Z');

		GPIF_Rdy0 <= '0';
		latched := false;
		FifoRd <= '0';

		State <= sIDLE;

	elsif rising_edge(SCLK) then

		GPIF_Rdy0 <= FifoRdCount(1) or (not TestMode);

		case State is
		when sIDLE =>
			if GPIF_Sr = '0' then
				GPIF_DataH <= (others => '0');

				case GPIF_Adr is
				when "000000000" =>
				GPIF_DataL <= MajorVersion; -- reading major version

				when "000000001" =>
				GPIF_DataL <= MinorVersion; -- reading minor version

				when "000000010" =>
				GPIF_DataL <= CounterValue(7 downto 0); -- lower 8 bits of counter

				when "000000011" =>
				GPIF_DataL <= CounterValue(15 downto 8); -- upper 8 bits of counter

				when "000000100" => -- read internal fifo status
				GPIF_DataL <= "0000" & FifoRdCount & FifoFull & FifoEmpty;

				-- add other address encoding code for reading

				when others =>
					GPIF_DataL <= (others => '0');
				end case;
				State <= sDRIVE_DATA;

			-- pipelining process
			elsif latched = false and FifoEmpty = '0' and TestMode = '1' then
			FifoRd <= '1';
				latched := true;
				GPIF_DataL <= (others => 'Z');
				GPIF_DataH <= (others => 'Z');
```

```
                    State <= sREAD_FIFO;
                elsif GPIF_Fr = '0' then
                    GPIF_DataL <= FifoOutData;
                    GPIF_DataH <= FifoOutData2;
                    latched := false;
                    State <= sDRIVE_DATA;
                else
                    GPIF_DataL <= (others => 'Z');
                    GPIF_DataH <= (others => 'Z');
                end if;

            when sREAD_FIFO =>
                FifoRd <= '0';
                State <= SIDLE;

            when sDRIVE_DATA =>
                GPIF_DataL <= (others => 'Z');
                GPIF_DataH <= (others => 'Z');
                State <= sIDLE;

            end case;

        end if;
end process;

-- gpif single write process
RegWriteTransaction: process (SCLK, RESET)
begin
        if (RESET = '1') then
            TestMode <= '0';
            CounterCommand <= (others => '0');
            GPIF_Rdy1 <= '0';
            Leds <= "11";
        elsif rising_edge(SCLK)then

            -- give ready signal to GPIF so it can start writing data
            GPIF_Rdy1 <= (not FifoRdCount(1)) or (not TestMode);

            if(GPIF_Sw = '0') then -- gpif single byte write
                case GPIF_Adr is

                when "000000000" =>
                CounterCommand <= GPIF_DataL(1 downto 0); -- set counter command

                when "000000001" =>
                Leds <= GPIF_DataL(1 downto 0); -- redirect input to leds

                when "100000000" => -- change test operation mode
                TestMode <= GPIF_DataL(0);

                -- add other address encoding code for GPIF single writing

                when others => null;

                end case;
            end if;
```

```
		end if;
end process;

-- counter process
CounterProc: process (SCLK,RESET)
begin
		if (RESET = '1') then
				CounterValue <= (others => '0'); -- counter is 0 initially
		elsif rising_edge(SCLK) then -- count on rising edge of the clock
				if(CounterCommand(0) = '1') then   -- if counter is enabled
						if(CounterCommand(1) = '1') then
								CounterValue <= CounterValue + 1; -- count up
						else
								CounterValue <= CounterValue - 1; -- count down
						end if;
				end if;
		end if;
end process;

FifoWr <= (not GPIF_Fw) and TestMode;

-- loop back fifo 1
fifo0 : fifo8x2047
		port map(
		din => GPIF_DataL, -- input data comes directly from gpif
		wr_en => FifoWr, -- write enable signal
		wr_clk => SCLK, -- write clock is system clock
		rd_en => FifoRd, -- read enable signal
		rd_clk => SCLK, -- read clock is system clock
		ainit => RESET, -- fifo reset
		dout => FifoOutData, -- fifo output data (loop-back data)
		full => FifoFull,   -- fifo full signal
		empty => FifoEmpty, -- fifo empty signal
		rd_count => FifoRdCount -- fifo count signal
		);

-- loop back fifo 2, meaningful only when 16-bit wordwide data bus is enabled
fifo1 : fifo8x2047
		port map(
		din => GPIF_DataH, -- input data comes directly from gpif
		wr_en => FifoWr, -- write enable signal
		wr_clk => SCLK, -- write clock is system clock
		rd_en => FifoRd, -- read enable signal
		rd_clk => SCLK, -- read clock is system clock
		ainit => RESET, -- fifo reset
		dout => FifoOutData2, -- fifo output data (loop-back data)
		full => open,		-- fifo full signal
		empty => open, -- fifo empty signal
		rd_count => open -- fifo count signal
		);

end Bhv;
```

```
-- Code example 5.2, gpif_test.ucf
-- gpif_test design constraints file

#
# Timing constraints
#
NET "sclk" TNM_NET = "sclk";
TIMESPEC "TS_sclk" = PERIOD "sclk" 48 MHz HIGH 50 %;
NET "gpif_datal<*>" OFFSET = IN 9830 ps BEFORE "sclk";
NET "gpif_datah<*>" OFFSET = IN 9830 ps BEFORE "sclk";
NET "gpif_adr<*>" OFFSET = IN 13330 ps BEFORE "sclk";
NET "gpif_sr" OFFSET = IN 14130 ps BEFORE "sclk";
NET "gpif_sw" OFFSET = IN 14130 ps BEFORE "sclk";
NET "gpif_fw" OFFSET = IN 14130 ps BEFORE "sclk";
NET "gpif_fr" OFFSET = IN 14130 ps BEFORE "sclk";
NET "gpif_datal<*>" OFFSET = OUT 11630 ps AFTER "sclk";
NET "gpif_datah<*>" OFFSET = OUT 11630 ps AFTER "sclk";
NET "gpif_rdy0" OFFSET = OUT 11930 ps AFTER "sclk";
NET "gpif_rdy1" OFFSET = OUT 11930 ps AFTER "sclk";

#
# Pin connections
#
NET "sclk" LOC = "p77";
NET "reset" LOC = "p133";
# other pin connections come here

#
# Configuration options
#
NET "gpif_datal<*>" SLOW;
NET "gpif_datah<*>" SLOW;
NET "gpif_rdy0" SLOW;
NET "gpif_rdy1" SLOW;
NET "leds<0>" SLOW;
NET "leds<1>" SLOW;
NET "test<0>" SLOW;
NET "test<1>" SLOW;
NET "gpif_adr<*>" IOBDELAY = BOTH;
NET "gpif_datal<*>" IOBDELAY = BOTH;
NET "gpif_datah<*>" IOBDELAY = BOTH;
NET "gpif_fr" IOBDELAY = BOTH;
NET "gpif_fw" IOBDELAY = BOTH;
NET "gpif_sr" IOBDELAY = BOTH;
NET "gpif_sw" IOBDELAY = BOTH;
NET "reset" IOBDELAY = BOTH;
NET "reset" PULLUP;
NET "gpif_datal<*>" PULLUP;
NET "gpif_datah<*>" PULLUP;
```