XPS_FX2 custom IP core block

Description

XPS_FX2 is a communication core to interface Xilinx Microblaze soft processor and a popular USB High Speed microcontroller Cypress CY7C68013A (also known as EzUSB FX2).

The XPS_FX2 supports 8bit Slave FIFO mode of operation on FX2 side. The FX2 has 4 endpoints with 2kB buffers (EP2, EP4, EP6, EP8).

The FX2 firmware was designed to support 4 high speed slave FIFOs: 3x TX (to PC, EP2-4-6), 1x RX (from PC, EP8).

The EP2-4-6 are FPGA outputs and EP8 is FPGA input.

This asymmetry is a consequence of the core being developed for data streaming to the PC (DAQ boards,cameras…).

The XPS_FX2 core side is driven by 48MHz IFCLK which is provided by FX2. The other side is driven by PLB clock and two separate clock signals for the data FIFOs.

The FIFOs can be accessed with PLB bus and/or through direct high speed FIFO ports.

The PLB bus FIFO access is only possible if the PLB clock matches FIFO clock.

Maximal supported bandwidth is normally limited by

- FX2 microcontroller's USB connection with host computer (see Data throughput limit of Bulk transfer type, C# API: Data Transfer Throughput Optimization and C++ API: Data Transfer Throughput Optimization);
- FX2 microcontroller's FIFO interface with FPGA (this interface can transfer up to 48 MB/s burst rate).

Pin Name	FPGA Direction	FX2 direction	Description	In the reference design case (1)
FD[7:0]	Bidirectional	Bidirectional	FD[0:7] are used for byte data transfer between FPGA and USB FX2 C.	If the custom IP blocks • XPS_FX2 (FX2 microcontroller FPGA) and • XPS_NPI_DMA (FPGA DRAM memory) and MicroBlaze API Commands (MB Commands)

- FX22MB_REG0_START_RX command (or derived work)

- FX22MB_REG0_START_TX command (or derived work)

				• FX22MB_REG0_STOP command (or derived work) are used, data bytes array could be transfered between • TE USB FX2 module's DRAM (MicroBlaze's software) and • host computer's memory (host computer's software)

FD[7:0] bus pins: FPGA <-> USB FX2 microcontroller connection;

(1) Reference design case: Logic Architecture Layer = Reference Architecture Layer and reference USB FX2 C firmware used

? Unknown Attachment

System integration block scheme

The XPS_FX2 has 4 interfaces:

- USB FX2 8-bit wide slave FIFO interface synchronous to IFCLK offering 48MB/s peak bandwidth.
- Xilinx PLBv4.6 created with IPIF wizard for access to 9 32-bit registers. These registers control the whole peripheral.
- Proprietary synchronous 32-bits wide FIFO_OUT bus used for data streaming directly from receive FIFO. The port is synchronous to RX_FIFO_Clk.
- Proprietary synchronous 32-bits wide FIFO_IN bus used for data streaming directly to transmit FIFO. The port is synchronous to TX_FIFO_Clk.

? Unknown Attachment

Peripheral internal structure block scheme

XPS_FX2 Core Design Parameters

Feature/Description	Parameter Name	Allowable Values	Default Value	VHDL Type
System Parameters				
Target FPGA family	C_FAMILY	spartan3, spartan3e, spartan3a, spartan3adsp, spartan3an, virtex2p, virtex4, qvirtex4, qrvirtex4, virtex5	virtex5	string
PLB Parameters				
PLB base address	C_BASEADDR	Valid Address	None	std_logic_vector
PLB high address	C_HIGHADDR	Valid Address	None	std_logic_vector
PLB least significant address bus width	C_SPLB_AWIDTH	32	32	integer
PLB data width	C_SPLB_DWIDTH	32, 64, 128	32	integer
Shared bus topology	C_SPLB_P2P	0 = Shared bus topology	0	integer
PLB master ID bus Width	C_SPLB_MID_WIDTH	log2(C_SPLB_NUM_MASTERS) with a minimum value of 1	1	integer
Number of PLB masters	C_SPLB_NUM_MASTERS	1 - 16	1	integer
Width of the slave data bus	C_SPLB_NATIVE_DWIDTH	32	32	integer
Burst support	C_SPLB_SUPPORT_BURSTS	0 = No burst support	0	integer
XPS_FX2 Parameters				
Size of Transmit FIFO	C_TX_FIFO_KBYTE	2, 4, 8,16,32	2	integer

Size of Receive FIFO	C_RX_FIFO_KBYTE	0, 2	2	integer
Using Address FIFO (1)	C_USE_ADDR_FIFO	0 (not working)	0	integer
Shift transmit FIFO clock by 180 degrees (2)	C_TX_FIFO_CLK_180	0, 1	0	integer

XPS_FX2 Core Design Parameters

⚠ (1) Address FIFO was designed to avoid FIFO draining before the EP ADDRESS changing. Will be fixed in further releases.

(2) Set to 1 only if the user experience 8-bit data shifting after a received packet of data. Normally set to 0.

XPS_FX2 Core I/O Signals

Name	Interface	I/O	Initial State	Description
ChipScope[0:31]	-	O	-	Debug port
USB_IFCLK	-	I	-	USB 48MHz clock
USB_SLRD	-	O	0	USB Data Read strobe
USB_SLWR	-	O	0	USB Data Write strobe
USB_FLAGA	-	I	-	USB programmable status flag (not used)
USB_FLAGB	-	I	-	USB TX FIFO full flag
USB_FLAGC	-	I	-	USB TX FIFO empty flag
USB_FLAGD	-	I	-	USB RX FIFO empty flag
USB_SLOE	-	O	0	Toggles FX2 IO buffer (1=read from USB)
USB_PKTEND	-	O	0	Commences current packet (if smaller than 512 bytes)
USB_FIFOADR[1:0]	-	O	0	Selects USB endpoint: • "00"=EP2, • "01"=EP4, • "10"=EP6, • "11"=EP8
USB_FD[7:0]	-	I/O	0	USB tristate data Bus
TX_FIFO_Clk	-	I	-	TX FIFO clock
RX_FIFO_Clk	-	I	-	RX FIFO clock
TX_FIFO_DIN[0:31]	FIFO_IN	I	-	TX FIFO input data
TX_FIFO_VLD	FIFO_IN	I	-	TX FIFO data valid strobe
TX_FIFO_RDY	FIFO_IN	O	0	TX FIFO is ready flag
RX_FIFO_DOUT[0:31]	FIFO_OUT	O	zeros	RX FIFO output data
RX_FIFO_VLD	FIFO_OUT	O	0	RX FIFO data valid strobe
RX_FIFO_RDY	FIFO_OUT	I	-	RX FIFO is ready flag
IP2INTC_Irpt	-	O	0	Processor interrupt
OTHERS ARE PLBv4.6 SIGNALS	PLBv4.6	-	-	-

XPS_FX2 I/O Signal Descriptions

Writing and reading to/from FIFO_IN and FIFO_OUT ports

The point to point unidirectional buses use simple handshaking protocol.

When (slave) “ready” signal is high the port is open for writing.

A write is performed when “valid” signal goes high. The “data” should be valid when valid signal is high. If “valid” signal goes high and the ready is low then the data are discarded.

The signals are updated on rising edge of clock.

The “valid” signal can be also continuous.

FIFO_IN port is ready when it is not under reset or being full.

If FIFO_OUT port is not ready then it will not send data ("valid" stays low and data stays in FIFO).

FIFO_OUT port latency to act upon "ready" goes low is 1 cycle.

? Unknown Attachment

FIFO high speed communication ports principle of operation

XPS_FX2 Core Registers

XPS_FX2 has a full access of a microprocessor to the core functionality through a 5 user 32-bit and 7 IPIF Interrupt registers attached to PLBv4.6 bus.

Base Address + Offset (hex)	Register Name	Access Type	Default Value (hex)	Description
XPS FX2 IP Core Grouping				
C_BASEADDR + 00	CR	R/W	0x00000000	Control Register
C_BASEADDR + 04	FTR	R/W	0x00000000	FIFOs Threshold Register
C_BASEADDR + 08	SR	Read	0x00000000	Status Register
C_BASEADDR + 0C	FWR	R/W	0x00000000	FIFO Write Register
C_BASEADDR + 10	FRR	Read	0x00000000	FIFO Read Register
IPIF Interrupt Controller Core Grouping				
C_BASEADDR + 200	INTR_DISR	Read	0x00000000	interrupt status register
C_BASEADDR + 204	INTR_DIPR	Read	0x00000000	interrupt pending register
C_BASEADDR + 208	INTR_DIER	Write	0x00000000	interrupt enable register
C_BASEADDR + 218	INTR_DIIR	Write	0x00000000	interrupt id (priority encoder) register
C_BASEADDR + 21C	INTR_DGIER	Write	0x00000000	global interrupt enable register
C_BASEADDR + 220	INTR_IPISR	Read	0x00000000	ip (user logic) interrupt status register
C_BASEADDR + 228	INTR_IPIER	Write	0x00000000	ip (user logic) interrupt enable register

XPS_FX2 Core Registers

⚠ The First (LSB) interrupt from user_logic is masked on the left!!

Details of XPS_FX2 Core Registers

The parts of the registers (or the whole registers) with a non-capital designation (e.g. wr_fifo_rst) are usually the names of the HDL signals connected to the described register.

Control Register (CR)

The Control Register is used to control basic peripheral functions. All the bit flags are assembled here.

? Unknown Attachment

Bits	Name	Description Reset Value	Description Reset Value	Valid Value
31	tx_fifo_rst	Transmit FIFO reset	0	0,1
30	rx_fifo_rst	Receive FIFO reset	0	0,1
29	tx_fifo_disable	When '1' TX_FIFO_RDY='0'	0	0,1
26-27	usb_fifoadr	USB endpoint selection (1)	0x0	Valid Endpoints: • 0 = EP2 • 1 = EP4 • 2 = EP6
0-15	pktend_timeout	Packet end timeout (2)	0x0	0-32767

Control Register bits

Packet end timount and USB_PKTEND

(1) Endpoint EP8 is **read only** and is switched automatically when data arrives. To achieve maximal throughput use only one endpoint and *prev ent TX FIFO draining* (**TX FIFO empty should not occur**).

(2) Packet end timeout timer automatically asserts USB_PKTEND signal when **TX_FIFO is empty** for a programmed number of cycles **and current USB EndPoint FIFO is not empty**.

Cycle timer is also reset when switched to EP8 – incoming data. The USB_PKTEND send current packet and enables the PC to receive packet smaller than 512 bytes. If user setup the timer properly then the packets are automatically send when there is no more data available in the TX_FIFO.

FIFOs Threshold Register (FTR)

This register is used to setup thresholds for interrupt triggering when FIFO occupancy reaches set number of words. For RX FIFO the prog_full flag goes high when number of words in a FIFO is higher than threshold. For TX FIFO the prog_empty flag goes high when number of words in a FIFO is lower than threshold.

The tx_fifo_threshold can have 9, 10 or 11 bits according to size of the TX_FIFO. This register is usually accessed using 16-bit writes.

? Unknown Attachment

FIFOs Threshold Register (FTR) bits

Status Register (SR)

In the status register the peripheral reports of the current status. The tx_fifo_count can have 9-13 bits according to size of the TX_FIFO. This register is usually accessed using 16-bit reads

? Unknown Attachment

Bits	Name	Description	Reset Value
19-31	tx_fifo_count	Transmit FIFO occupancy in words	0
18	tx_fifo_overflow	Transmit FIFO overflow flag	0
17	tx_fifo_full	Transmit FIFO full flag	0
16	tx_fifo_empty	Transmit FIFO empty flag	1
7-15	rx_fifo_count	Receive FIFO occupancy in words	0

3	rx_fifo_prog_full	Receive FIFO programmable full flag	0
2	rx_fifo_underflow	Receive FIFO underflow flag	0
1	rx_fifo_full	Receive FIFO full flag	0
0	rx_fifo_empty	Receive FIFO empty flag	1

Status Register (SR) bits

FIFO Write Register (FWR)

Single beat write to this register puts a single word (4 bytes) to TX FIFO. For proper operation PLB clock frequency should be less or equal to TX_FIFO_Clk.

FIFO Read Register (FRR)

Single beat read from this register pops one word (4 bytes) from RX FIFO. For proper operation PLB clock frequency should be less or equal to RX_FIFO_Clk.

Interrupt enable/pending registers

With INTR_IPIER register the user can enable/disable peripheral interrupt sources. With INTR_IPISR the user can identify interrupt source. Writing a value to INTR_IPISR also clears interrupt.

> ⚠ **"Ghost" interrupts**
>
> **The user must make sure that triggered interrupts will be cleared in a consinstent way (single owner)**; the user (host computer's software) **must only clear triggered interrupts**. Otherwise the user will trigger "ghost" interrupts which were not triggered by peripheral, but the interrupt controller itself.

Writing 0x7 to INTR_DIER will enable IP interrupt sources and writing 0x80000000 to INTR_DGIER will enable global interrupt.

The image below presents a conection of user logic interrupt to INTR_IPIER and INTR_IPISR.

? Unknown Attachment

Interrupt enable/pending registers

Programming model

> ⚠ By setting control register (CR) make sure that user **not override the previously set bits**.

Example 1

Resetting the TX_FIFO:

1. Write 0x00000001 to CR

2. Write 0x00000000 to CR

Example 2

Resetting the RX_FIFO:

1. Write 0x00000002 to CR

2. Write 0x00000000 to CR

Example 3

Setting the endpoint address to EP4

1. Write 0x00000010 to CR

Example 4 (if Reference Design is used): test XPS_NPI_DMA and XPS_FX2 using MB Commands

XPS_NPI_DMA and XPS_FX2 custom IP blocks are both necessary to connect (throgh USB connection) host computer's software and TE USB FX2 module's DRAM.

The MB Commands FX22MB_REG0_START_RX, FX22MB_REG0_START_TX, FX22MB_REG0_STOP are used for data throughput and integrity test.

MB Commands require the XPS_I2C_SLAVE custom IP block and a proper FX2 interrupt handler (i2c_slave_int_handler() function in interrupt.c running on MicroBlaze); the FX2 interrupt handler is called to handle the signal interrupt xps_i2c_slave_0_IP2INTC_Irpt. The i2c_slave_int_handler() function actually execute the I2C delivered MB Command.

⚠ Write test should be executed before read test; otherwise the read test will fail.