



PCI-X

Development Board

User Manual



Terasic PCI-X Development Board

CONTENTS

Chapter 1 PCI Package	1
1.1 Package contents	1
1.2 Getting Help.....	1
1.3 Revision History	2
Chapter 2 Introduction.....	3
2.1 General Description	3
2.2 Layout and Components	3
2.3 Block Diagram of the PCI Board.....	5
2.4 Power-up the PCI Board	6
Chapter 3 Components & Interfaces	7
3.1 Clocking Circuitry	7
3.1.1 Clock & Programmable PLL	8
3.2 Switch	8
3.3 HSTC Expansion Connectors	9
3.4 Off-Chip Memory	17
3.4.1 DDR2 SO-DIMM Module.....	17
Chapter 4 Setup PCI Board	21
4.1 System Requirement	21
4.2 Hardware Installation: PCI Board.....	21
4.3 Software Installation: PCI Kernel Driver	22
4.4 Install License File.....	23
4.5 Diagnoses.....	23
Chapter 5 PCI System Builder	24
5.1 Introduction.....	24
5.2 Quartus Top Design	25
5.3 Built-in Logic.....	26
5.4 Save Configuration	27
5.5 Generated Code.....	27
Chapter 6 Host Software Library and Utility	30
6.1 PCI Software Stack.....	30
6.2 Data Structure in Terasic_API.h	31
6.3 API List of Terasic_API.DLL.....	31
6.4 API Description of Terasic_DLL.....	32

6.5	PCI Control Panel Utility	38
Chapter 7 Reference Design.....		41
7.1	Remote Control LED	41
7.2	Button IRQ.....	46
7.3	DDR2 Access	50
Chapter 8 Multi-Port Memory Controller		55
8.1	Principle of Read/Write Port.....	55
8.1.1	Write Port	55
8.1.2	Read Port.....	56
8.2	Port Interface.....	57
8.2.1	Simple Write Port.....	58
8.2.2	Simple Read Port	59
8.2.3	Enhanced port	60
Chapter 9 PCI Local Interface		62
9.1	PCI Local Write/Read Interface.....	62
9.2	PCI Interrupt	63
Appendix A Programming the Serial Configuration device		66

Chapter 1

PCI Package

The PCI package contains all components needed to use the PCI board in conjunction with a computer that runs the Microsoft Windows software.

1.1 Package contents

The PCI Package includes:

- Cyclone III PCI development board
- Terasic USB Blaster
- USB Cable for FPGA programming and control
- CD-ROM containing the User Manual, the Control Panel utility, the PCI System Builder and reference designs.
- THDB_HLB
- THDB_HFF
- Screw and Copper Pillar Package
- Power Cable

1.2 Getting Help

Here are the addresses where you can get help if you encounter problems:

- Altera Corporation
101 Innovation Drive
San Jose, California, 95134 USA
Email: mysupport@altera.com
- Terasic Technologies
No. 356, Sec. 1, Fusing E. Rd.
Jhubei City, HsinChu County, Taiwan, 302
Email: support@terasic.com
Web: www.terasic.com

1.3 Revision History

Date	Version	Changes
2008.12	First publication	

Chapter 2

Introduction

This chapter provides an introduction of the PCI Board features and design characteristic.

2.1 General Description

The Cyclone® III PCI development board provides a hardware platform for developing and prototyping low-power, high-performance, logic-intensive PCI-based designs. The board provides a high-density of the memory to facilitate the design and development of FPGA designs which need huge memory storage, and also includes Low-Voltage Differential Signaling (LVDS) interface of the High-Speed Terasic Connectors (HSTCs) for extra high-speed interface application.

Based on Cyclone® III FPGA and using Altera MegaCore functions, Terasic IP and the reference design, Cyclone III PCI Development Board allows users to quickly implement the design and solve design problems that require time-consuming, custom solutions.

Finally, to simplify the design process, we provide the software which calls “PCI System Builder” that provides a convenient way to build interfaces between host PC and user logic on FPGA, and also supports the interface of multi-port controller which allows shared access to a unique external memory. For more details about PCI System Builder, refer to *Chapter 4 PCI System Builder*.

2.2 Layout and Components

A photograph of the Cyclone® III PCI development board is shown in Figure 2.1 and 2.2. They depict the layout of the board and indicate the location of the connectors and key components.

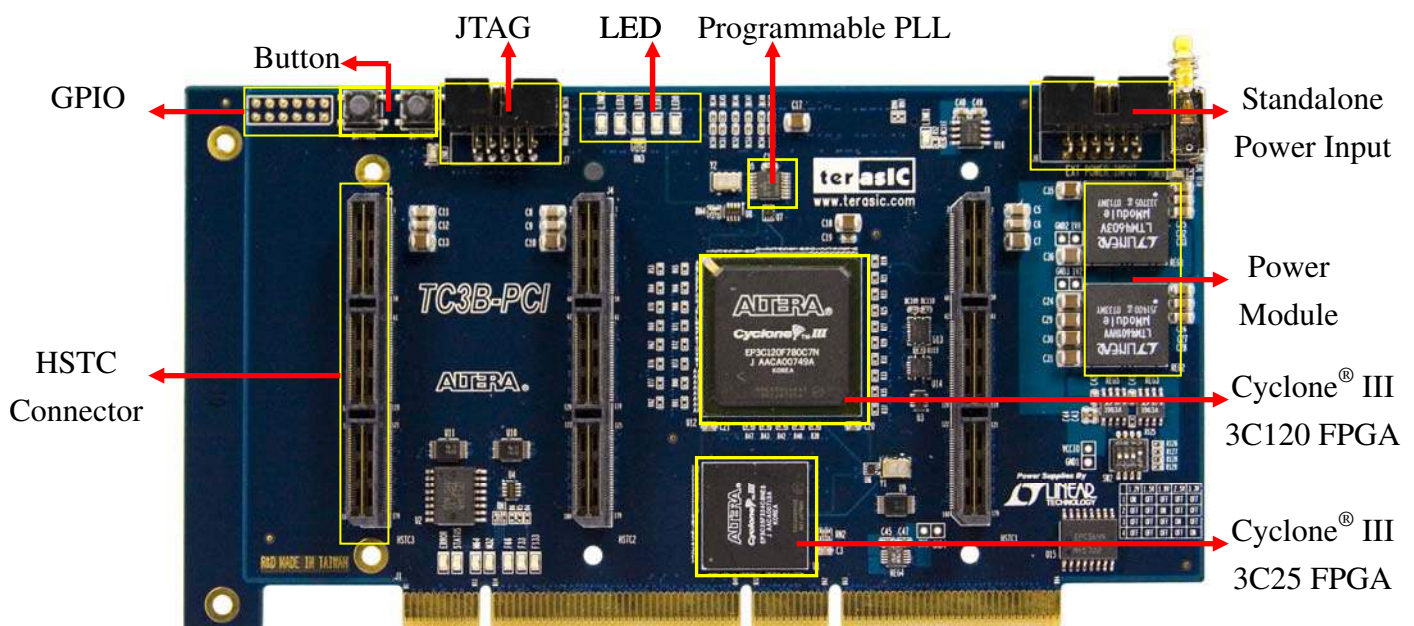


Figure 2.1 Cyclone® III PCI development board

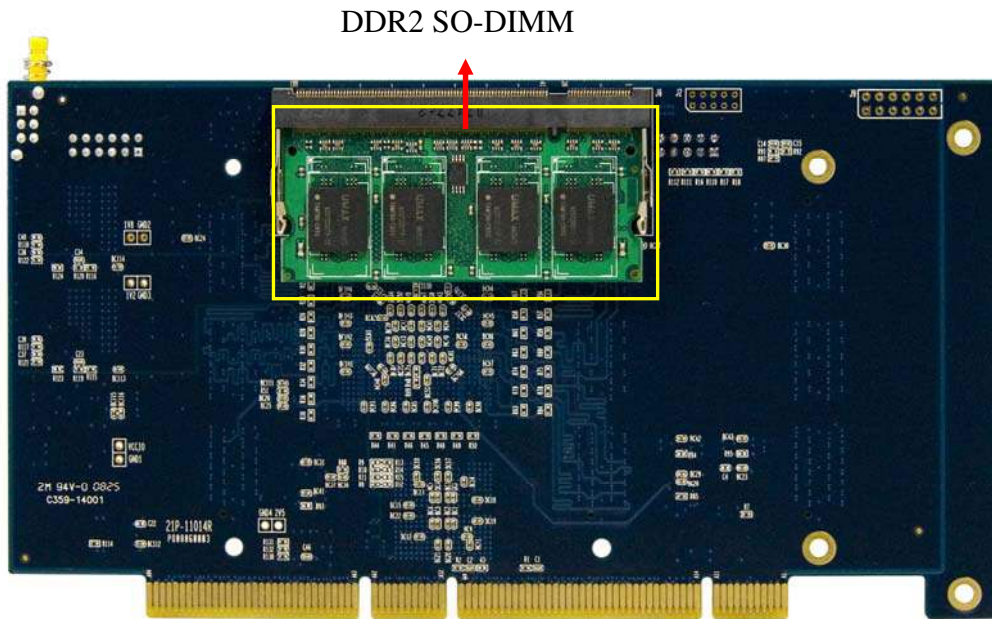


Figure 2.2 Cyclone[®] III PCI development board

The following hardware is provided on the PCI board:

- Altera Cyclone[®] III 3C120 FPGA device
 - 119,088 logic elements (LEs)
 - 3,981,312 total RAM bits
 - 288 18 x 18 multiplier blocks
- Altera Cyclone[®] III 3C25 FPGA device (PCI Bridge).
- Altera Serial Configuration device
 - EPCS64
 - EPCS16
- On-board memories
 - Up to 4GBytes DDR2 SO-DIMM
- Three HSTCs
 - 120 differential pair signals
 - 20 dedicated clock signals (8 differential pair & 4 single-end)
- PCI bus interfaces.

These features allow the user to implement the designs that need an enormous memory and high-speed data transfer. In addition to these hardware features, the PCI board has software support for PCI bus DMA, bus interrupt functions and a control panel facility to access various components.

In order to use the TC3B-PCI board, the user has to be familiar with the Quartus II software. The necessary knowledge can be acquired by reading the tutorials *Quartus II Introduction* (which exists in three versions based on the design entry method used, namely Verilog, VHDL or schematic entry).

2.3 Block Diagram of the PCI Board

Figure 2.3 gives the high-level block diagram of the PCI board. To provide maximum flexibility for the user, all connections are made through the Cyclone® III FPGA device. Thus, the user can configure the FPGA to implement any system design.

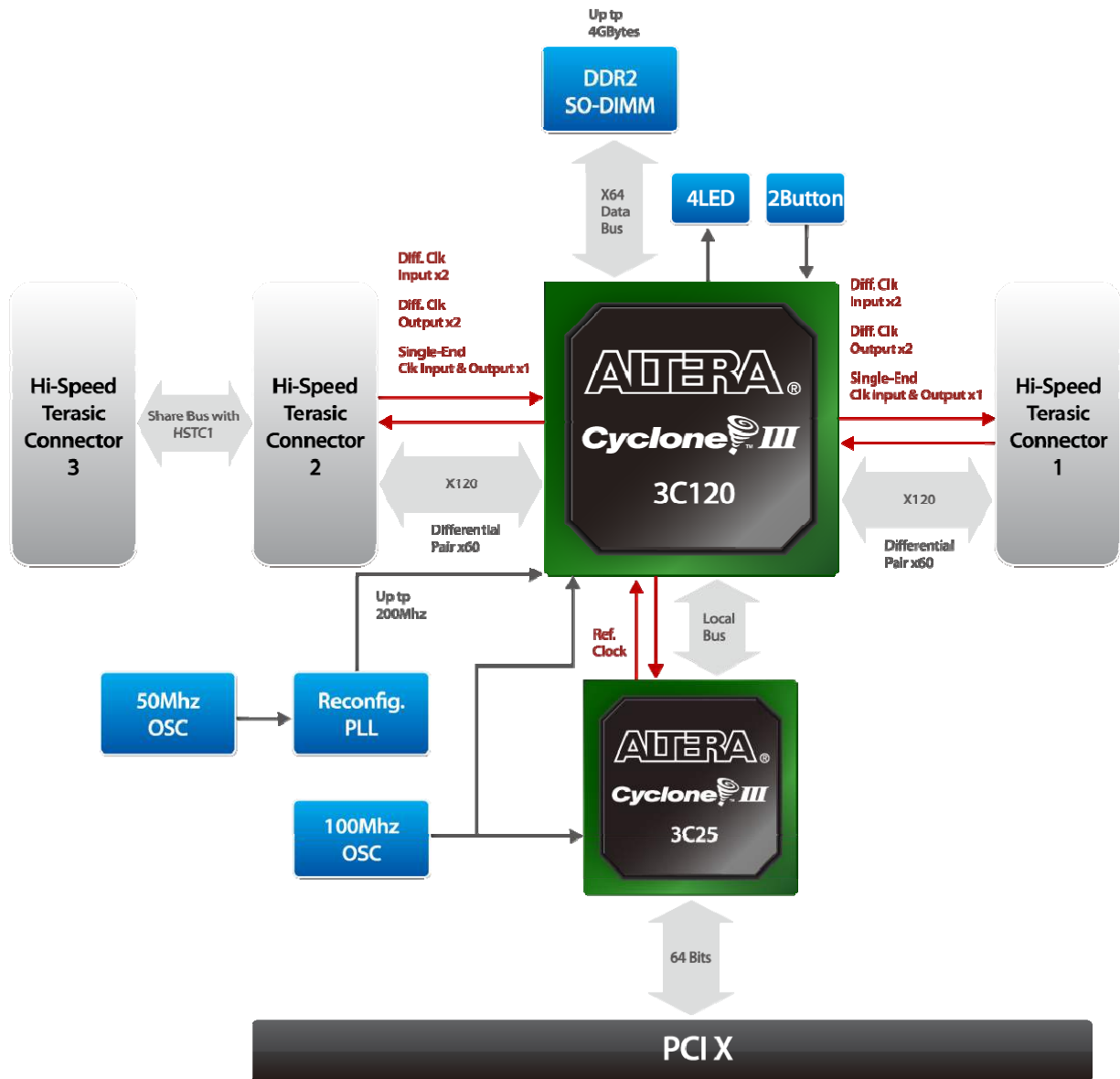


Figure 2.3 High level block diagram of the PCI board

Following is more detailed information about the blocks in Figure 2.3:

Cyclone® III 3C120 FPGA

- 119,088 LEs
- 432 M4K RAM blocks
- 3,981,312 total RAM bits
- 288 18x18 multiplier blocks
- Four phase locked loops (PLLs)

Cyclone® III 3C25 FPGA

- 24,624 LEs
- 66 M4K RAM blocks
- 608,256 total RAM bits
- 66 18x18 multiplier blocks
- Four PLLs

Serial Configuration device

- Altera's EPCS64 & EPCS16 serial configuration device
- In-system programming mode via JTAG interface ⁽¹⁾

DDR2 SDRAM

- 64-bits DDR2 SO-DIMM
- Up to 4GBytes

LED & button

- 4 user-controlled LEDs
- 2 user-controlled Buttons

Clock inputs

- Programmable PLL (80kHz ~ 200MHz)
- 100MHz oscillator

Three 180-pin HSTC expansion connectors

- 260 Cyclone[®] III I/O pins
- High-Speed connector up to GHz frequency

2.4 Power-up the PCI Board

The PCI Board contains the following ways to power-up:

1. Plug into PCI bus
2. Connect external power cable

After the PCI board powers up, the on-board configuration device which ships pre-programmed with the factory design, automatically configures the Cyclone[®] III device and the user-controlled LEDs will flash in a "Knight Rider" pattern.

Chapter 3

Components & Interfaces

This chapter describes functions of the components and interfaces on the development board, including detailed pin-out information to enable designers to create custom FPGA designs.

3.1 Clocking Circuitry

In order to achieve the design requirement which needs different frequency clock sources, the development board provides two clock sources that connect to dedicated clock input pins of Cyclone[®] III FPGA. One of the clock sources is a 100MHz oscillator and another is a programmable PLL.

For LVDS clocking, the expansion connectors (HSTCs) include the dedicated differential clock inputs and PLL output pins of Cyclone[®] III FPGA to implement high-speed clocking interface. Figure 3.1 shows the clocking diagram of the PCI board.

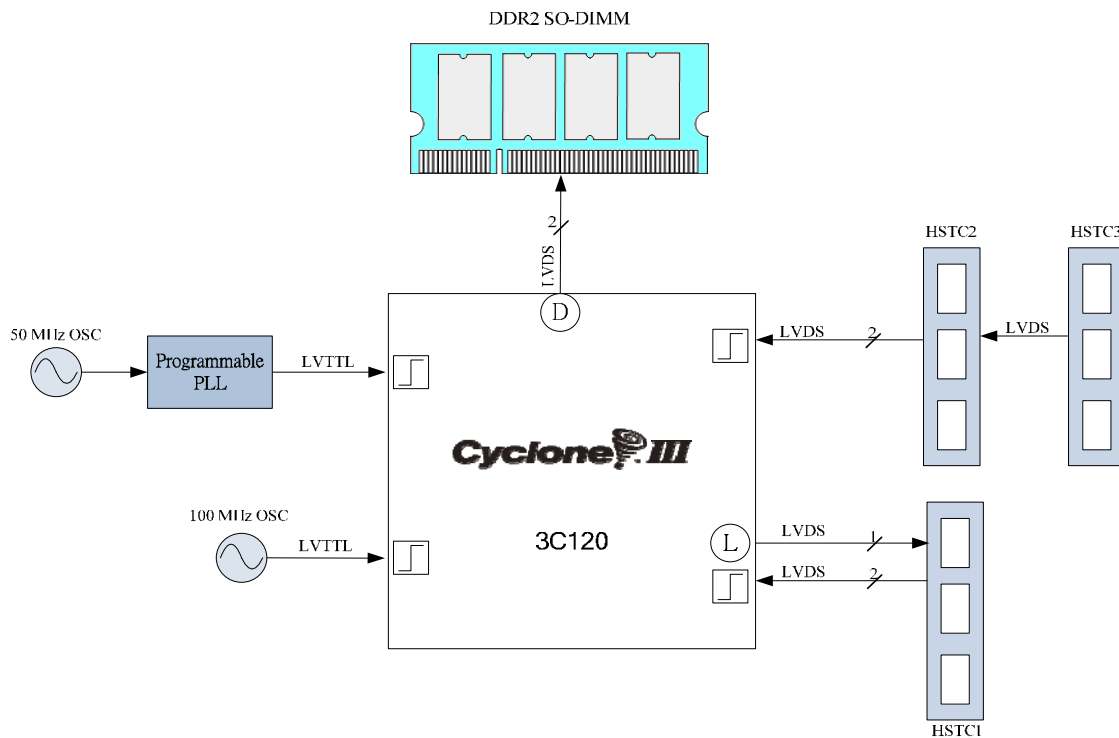


Figure 3.1 Clocking diagram of the PCI Board

Ⓛ : Dedicated PLL Output

Ⓜ : Dedicated Clock Input

ⓓ : Differential IO

3.1.1 Clock & Programmable PLL

The Cyclone[®] III PCI development board provides a programmable PLL which is driven by a 50 MHz oscillator and utilizes 2-wire serial interface SDAT and SCLK that operates up to 400 kbits/sec in read or write mode. The output frequency range of the PLL is 80 KHz to 200 MHz. A block diagram of the clock and on-board PLL showing connections to the Cyclone III FPGA is given in Figure 3.2. The associated pin assignments appear in Table 3.1

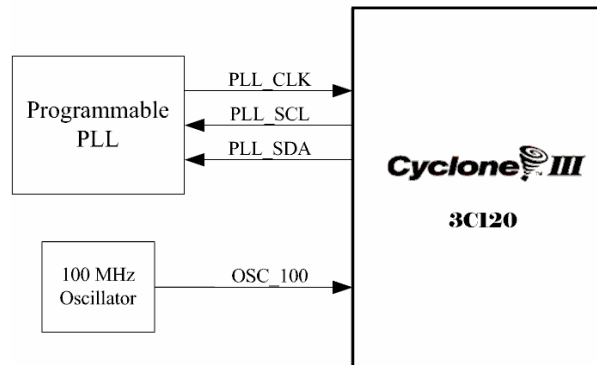


Figure 3.2 Block diagram of the clock and on-board PLL

Signal Name	FPGA Pin No.	Description
OSC_100	PIN_AG14	100 MHz Oscillator
PLL_CLK	PIN_B15	PLL Clock Output
PLL_SCL	PIN_AB24	PLL Serial Interface - Clock
PLL_SDA	PIN_AB23	PLL Serial Interface - Data

Table 3.1 Pin assignments of clock and on-board PLL

3.2 Switch

The Switch of Cyclone III PCI Board is used to select the expansion connectors IO voltage. Table 3.2 lists voltage selection by jumper.

Expansion IO Voltage	Pin number			
	1	2	3	4
1.2V	On	Off	Off	Off
1.5V	Off	Off	Off	Off
1.8V	Off	On	Off	Off
2.5V	Off	Off	On	Off
3.3V	Off	Off	Off	On

Table 3.2 Voltage selection of the expansion IO

3.3 HSTC Expansion Connectors

The Cyclone® III PCI development board contains three HSTC connectors (HSTC1, HSTC2 and HSTC3). The HSTC2 fully shares pins with HSTC3. These expansion connectors have total 240 bi-directional I/Os, 10 dedicated clock inputs and 10 PLL outputs of the Cyclone® III FPGA, and also provides DC +12V, DC +5V, DC +3.3V and GND pins. Furthermore, the voltage level of the I/O pins on the expansion connectors can be adjusted to 3.3V, 2.5V, 1.8V, 1.5V, 1.2V by using on-board switch.

High-speed differential I/O standards have become popular in high-speed interfaces because of their significant advantages over single-ended I/O standards. In response to the current market need, the PCI board supports LVDS channel up to 60 transmitters and 60 receivers on the expansion connectors. The channels had already achieved data rates of 600Mbps on Cyclone® III PCI development board. In summary, these features of the expansion connectors give applications the most flexibility for a variety of users. Figure 3.3 shows the schematic of HSTC expansion connector. Table 3.3 and 3.4 gives the pin assignment.

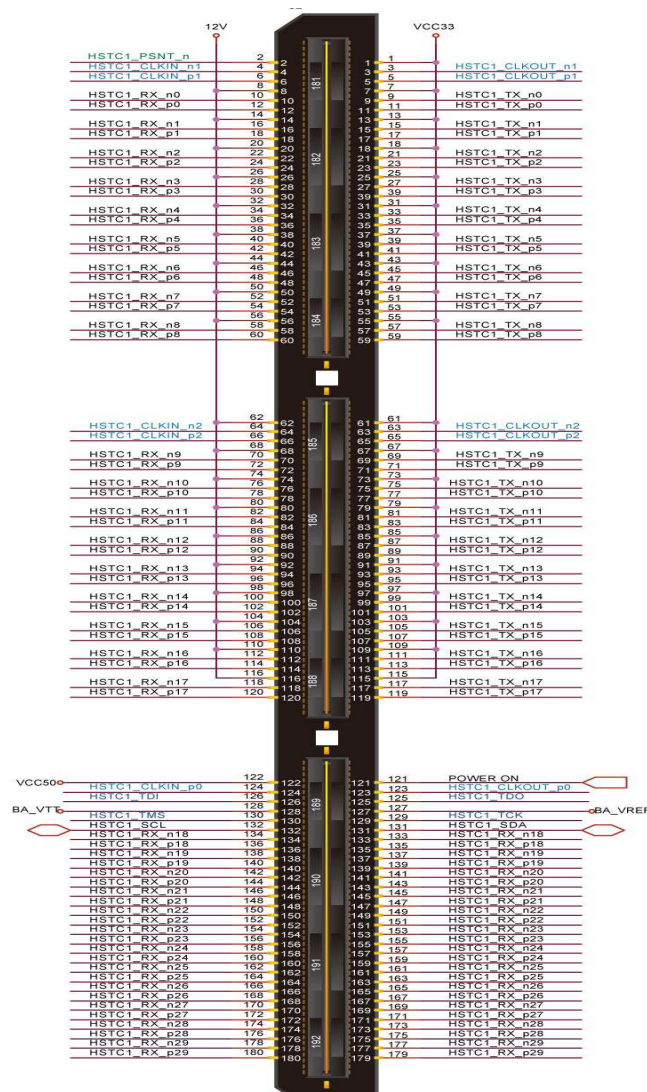


Figure 3.3 Schematic of the HSTC expansion connector

Schematic Signal Name		Connector pin no.	FPGA Pin Name
HSTC3_CLKIN_n0	HSTC2_CLKIN_n0	4	PIN_J1
HSTC3_CLKIN_p0	HSTC2_CLKIN_p0	6	PIN_J2
HSTC3_RX_n0	HSTC2_RX_n0	10	PIN_C2
HSTC3_RX_p0	HSTC2_RX_p0	12	PIN_D3
HSTC3_RX_n1	HSTC2_RX_n1	16	PIN_D1
HSTC3_RX_p1	HSTC2_RX_p1	18	PIN_D2
HSTC3_RX_n2	HSTC2_RX_n2	22	PIN_F3
HSTC3_RX_p2	HSTC2_RX_p2	24	PIN_E3
HSTC3_RX_n3	HSTC2_RX_n3	28	PIN_F1
HSTC3_RX_p3	HSTC2_RX_p3	30	PIN_F2
HSTC3_RX_n4	HSTC2_RX_n4	34	PIN_G3
HSTC3_RX_p4	HSTC2_RX_p4	36	PIN_G4
HSTC3_RX_n5	HSTC2_RX_n5	40	PIN_H3
HSTC3_RX_p5	HSTC2_RX_p5	42	PIN_H4
HSTC3_RX_n6	HSTC2_RX_n6	46	PIN_J3
HSTC3_RX_p6	HSTC2_RX_p6	48	PIN_J4
HSTC3_RX_n7	HSTC2_RX_n7	52	PIN_G5
HSTC3_RX_p7	HSTC2_RX_p7	54	PIN_G6
HSTC3_RX_n8	HSTC2_RX_n8	58	PIN_N3
HSTC3_RX_p8	HSTC2_RX_p8	60	PIN_N4
HSTC3_CLKIN_n1	HSTC2_CLKIN_n1	64	PIN_Y1
HSTC3_CLKIN_p1	HSTC2_CLKIN_p1	66	PIN_Y2
HSTC3_RX_n9	HSTC2_RX_n9	70	PIN_H24
HSTC3_RX_p9	HSTC2_RX_p9	72	PIN_H23
HSTC3_RX_n10	HSTC2_RX_n10	76	PIN_G26
HSTC3_RX_p10	HSTC2_RX_p10	78	PIN_G25
HSTC3_RX_n11	HSTC2_RX_n11	82	PIN_J24
HSTC3_RX_p11	HSTC2_RX_p11	84	PIN_J23
HSTC3_RX_n12	HSTC2_RX_n12	88	PIN_K22
HSTC3_RX_p12	HSTC2_RX_p12	90	PIN_K21
HSTC3_RX_n13	HSTC2_RX_n13	94	PIN_L22
HSTC3_RX_p13	HSTC2_RX_p13	96	PIN_L21
HSTC3_RX_n14	HSTC2_RX_n14	100	PIN_V22
HSTC3_RX_p14	HSTC2_RX_p14	102	PIN_U22
HSTC3_RX_n15	HSTC2_RX_n15	106	PIN_M1
HSTC3_RX_p15	HSTC2_RX_p15	108	PIN_M2
HSTC3_RX_n16	HSTC2_RX_n16	112	PIN_P1

HSTC3_RX_p16	HSTC2_RX_p16	114	PIN_P2
HSTC3_RX_n17	HSTC2_RX_n17	118	PIN_R1
HSTC3_RX_p17	HSTC2_RX_p17	120	PIN_R2
HSTC3_CLKIN_2	HSTC2_CLKIN_2	124	PIN_R5
HSTC3_RX_n18	HSTC2_RX_n18	134	PIN_T3
HSTC3_RX_p18	HSTC2_RX_p18	136	PIN_T4
HSTC3_RX_n19	HSTC2_RX_n19	138	PIN_R6
HSTC3_RX_p19	HSTC2_RX_p19	140	PIN_R7
HSTC3_RX_n20	HSTC2_RX_n20	142	PIN_AA3
HSTC3_RX_p20	HSTC2_RX_p20	144	PIN_AA4
HSTC3_RX_n21	HSTC2_RX_n21	146	PIN_R4
HSTC3_RX_p21	HSTC2_RX_p21	148	PIN_R3
HSTC3_RX_n22	HSTC2_RX_n22	150	PIN_V5
HSTC3_RX_p22	HSTC2_RX_p22	152	PIN_V6
HSTC3_RX_n23	HSTC2_RX_n23	154	PIN_U4
HSTC3_RX_p23	HSTC2_RX_p23	156	PIN_U3
HSTC3_RX_n24	HSTC2_RX_n24	158	PIN_AC1
HSTC3_RX_p24	HSTC2_RX_p24	160	PIN_AC2
HSTC3_RX_n25	HSTC2_RX_n25	162	PIN_V7
HSTC3_RX_p25	HSTC2_RX_p25	164	PIN_V8
HSTC3_RX_n26	HSTC2_RX_n26	166	PIN_AD1
HSTC3_RX_p26	HSTC2_RX_p26	168	PIN_AD2
HSTC3_RX_n27	HSTC2_RX_n27	170	PIN_W3
HSTC3_RX_p27	HSTC2_RX_p27	172	PIN_W4
HSTC3_RX_n28	HSTC2_RX_n28	174	PIN_AE1
HSTC3_RX_p28	HSTC2_RX_p28	176	PIN_AE2
HSTC3_RX_n29	HSTC2_RX_n29	178	PIN_AD3
HSTC3_RX_p29	HSTC2_RX_p29	180	PIN_AC3
HSTC3_CLKOUT_n0	HSTC2_CLKOUT_n0	3	PIN_G1
HSTC3_CLKOUT_p0	HSTC2_CLKOUT_p0	5	PIN_G2
HSTC3_TX_n0	HSTC2_TX_n0	9	PIN_K1
HSTC3_TX_p0	HSTC2_TX_p0	11	PIN_K2
HSTC3_TX_n1	HSTC2_TX_n1	15	PIN_K3
HSTC3_TX_p1	HSTC2_TX_p1	17	PIN_K4
HSTC3_TX_n2	HSTC2_TX_n2	21	PIN_L8
HSTC3_TX_p2	HSTC2_TX_p2	23	PIN_K8
HSTC3_TX_n3	HSTC2_TX_n3	27	PIN_K7
HSTC3_TX_p3	HSTC2_TX_p3	29	PIN_J7

HSTC3_TX_n4	HSTC2_TX_n4	33	PIN_L6
HSTC3_TX_p4	HSTC2_TX_p4	35	PIN_L7
HSTC3_TX_n5	HSTC2_TX_n5	39	PIN_J5
HSTC3_TX_p5	HSTC2_TX_p5	41	PIN_J6
HSTC3_TX_n6	HSTC2_TX_n6	45	PIN_M3
HSTC3_TX_p6	HSTC2_TX_p6	47	PIN_M4
HSTC3_TX_n7	HSTC2_TX_n7	51	PIN_L3
HSTC3_TX_p7	HSTC2_TX_p7	53	PIN_L4
HSTC3_TX_n8	HSTC2_TX_n8	57	PIN_M7
HSTC3_TX_p8	HSTC2_TX_p8	59	PIN_M8
HSTC3_CLKOUT_n1	HSTC2_CLKOUT_n1	63	PIN_L1
HSTC3_CLKOUT_p1	HSTC2_CLKOUT_p1	65	PIN_L2
HSTC3_TX_n9	HSTC2_TX_n9	69	PIN_F25
HSTC3_TX_p9	HSTC2_TX_p9	71	PIN_F24
HSTC3_TX_n10	HSTC2_TX_n10	75	PIN_G24
HSTC3_TX_p10	HSTC2_TX_p10	77	PIN_G23
HSTC3_TX_n11	HSTC2_TX_n11	81	PIN_K26
HSTC3_TX_p11	HSTC2_TX_p11	83	PIN_K25
HSTC3_TX_n12	HSTC2_TX_n12	87	PIN_U26
HSTC3_TX_p12	HSTC2_TX_p12	89	PIN_U25
HSTC3_TX_n13	HSTC2_TX_n13	93	PIN_V26
HSTC3_TX_p13	HSTC2_TX_p13	95	PIN_V25
HSTC3_TX_n14	HSTC2_TX_n14	99	PIN_V24
HSTC3_TX_p14	HSTC2_TX_p14	101	PIN_V23
HSTC3_TX_n15	HSTC2_TX_n15	105	PIN_W21
HSTC3_TX_p15	HSTC2_TX_p15	107	PIN_V21
HSTC3_TX_n16	HSTC2_TX_n16	111	PIN_Y22
HSTC3_TX_p16	HSTC2_TX_p16	113	PIN_W22
HSTC3_TX_n17	HSTC2_TX_n17	117	PIN_Y7
HSTC3_TX_p17	HSTC2_TX_p17	119	PIN_W8
HSTC3_CLKOUT_2	HSTC2_CLKOUT_2	123	PIN_AB3
HSTC3_TX_n18	HSTC2_TX_n18	133	PIN_V1
HSTC3_TX_p18	HSTC2_TX_p18	135	PIN_V2
HSTC3_TX_n19	HSTC2_TX_n19	137	PIN_U1
HSTC3_TX_p19	HSTC2_TX_p19	139	PIN_U2
HSTC3_TX_n20	HSTC2_TX_n20	141	PIN_V3
HSTC3_TX_p20	HSTC2_TX_p20	143	PIN_V4
HSTC3_TX_n21	HSTC2_TX_n21	145	PIN_U5
HSTC3_TX_p21	HSTC2_TX_p21	147	PIN_U6

HSTC3_TX_n22	HSTC2_TX_n22	149	PIN_Y5
HSTC3_TX_p22	HSTC2_TX_p22	151	PIN_Y6
HSTC3_TX_n23	HSTC2_TX_n23	153	PIN_W1
HSTC3_TX_p23	HSTC2_TX_p23	155	PIN_W2
HSTC3_TX_n24	HSTC2_TX_n24	157	PIN_AB1
HSTC3_TX_p24	HSTC2_TX_p24	159	PIN_AB2
HSTC3_TX_n25	HSTC2_TX_n25	161	PIN_Y3
HSTC3_TX_p25	HSTC2_TX_p25	163	PIN_Y4
HSTC3_TX_n26	HSTC2_TX_n26	165	PIN_AA5
HSTC3_TX_p26	HSTC2_TX_p26	167	PIN_AA6
HSTC3_TX_n27	HSTC2_TX_n27	169	PIN_AB5
HSTC3_TX_p27	HSTC2_TX_p27	171	PIN_AB6
HSTC3_TX_n28	HSTC2_TX_n28	173	PIN_AF2
HSTC3_TX_p28	HSTC2_TX_p28	175	PIN_AE3
HSTC3_TX_n29	HSTC2_TX_n29	177	PIN_AC4
HSTC3_TX_p29	HSTC2_TX_p29	179	PIN_AC5

Table 3.3 Pin assignments of the HSTC2 and HSTC3

Schematic Signal Name	Board Reference	FPGA Pin Name
HSTC1_CLKIN_n0	4	PIN_J28
HSTC1_CLKIN_p0	6	PIN_J27
HSTC1_RX_n0	10	PIN_L24
HSTC1_RX_p0	12	PIN_L23
HSTC1_RX_n1	16	PIN_R21
HSTC1_RX_p1	18	PIN_P21
HSTC1_RX_n2	22	PIN_C27
HSTC1_RX_p2	24	PIN_D26
HSTC1_RX_n3	28	PIN_R23
HSTC1_RX_p3	30	PIN_R22
HSTC1_RX_n4	34	PIN_E26
HSTC1_RX_p4	36	PIN_F26
HSTC1_RX_n5	40	PIN_H26
HSTC1_RX_p5	42	PIN_H25
HSTC1_RX_n6	46	PIN_AA13
HSTC1_RX_p6	48	PIN_Y13
HSTC1_RX_n7	52	PIN_AB14
HSTC1_RX_p7	54	PIN_AA14
HSTC1_RX_n8	58	PIN_AD11

HSTC1_RX_p8	60	PIN_AC11
HSTC1_CLKIN_n1	64	PIN_Y28
HSTC1_CLKIN_p1	66	PIN_Y27
HSTC1_RX_n9	70	PIN_AA10
HSTC1_RX_p9	72	PIN_AA8
HSTC1_RX_n10	76	PIN_AB8
HSTC1_RX_p10	78	PIN_AB9
HSTC1_RX_n11	82	PIN_AB12
HSTC1_RX_p11	84	PIN_AC12
HSTC1_RX_n12	88	PIN_AC7
HSTC1_RX_p12	90	PIN_AD8
HSTC1_RX_n13	94	PIN_AF11
HSTC1_RX_p13	96	PIN_AE11
HSTC1_RX_n14	100	PIN_AF14
HSTC1_RX_p14	102	PIN_AE14
HSTC1_RX_n15	106	PIN_AF12
HSTC1_RX_p15	108	PIN_AE12
HSTC1_RX_n16	112	PIN_AH10
HSTC1_RX_p16	114	PIN_AG10
HSTC1_RX_n17	118	PIN_AH8
HSTC1_RX_p17	120	PIN_AG8
HSTC1_CLKIN_2	124	PIN_L26
HSTC1_RX_n18	134	PIN_AC10
HSTC1_RX_p18	136	PIN_AB10
HSTC1_RX_n19	138	PIN_AC8
HSTC1_RX_p19	140	PIN_AB7
HSTC1_RX_n20	142	PIN_AH6
HSTC1_RX_p20	144	PIN_AG6
HSTC1_RX_n21	146	PIN_AH12
HSTC1_RX_p21	148	PIN_AG12
HSTC1_RX_n22	150	PIN_AF8
HSTC1_RX_p22	152	PIN_AE8
HSTC1_RX_n23	154	PIN_AF13
HSTC1_RX_p23	156	PIN_AE13
HSTC1_RX_n24	158	PIN_AH4
HSTC1_RX_p24	160	PIN_AG4
HSTC1_RX_n25	162	PIN_AH11
HSTC1_RX_p25	164	PIN_AG11
HSTC1_RX_n26	166	PIN_AH7

HSTC1_RX_p26	168	PIN_AG7
HSTC1_RX_n27	170	PIN_AF10
HSTC1_RX_p27	172	PIN_AE10
HSTC1_RX_n28	174	PIN_AA12
HSTC1_RX_p28	176	PIN_Y12
HSTC1_RX_n29	178	PIN_AF7
HSTC1_RX_p29	180	PIN_AE7
HSTC1_CLKOUT_n0	3	PIN_J26
HSTC1_CLKOUT_p0	5	PIN_J25
HSTC1_TX_n0	9	PIN_D28
HSTC1_TX_p0	11	PIN_D27
HSTC1_TX_n1	15	PIN_E28
HSTC1_TX_p1	17	PIN_E27
HSTC1_TX_n2	21	PIN_F28
HSTC1_TX_p2	23	PIN_F27
HSTC1_TX_n3	27	PIN_G28
HSTC1_TX_p3	29	PIN_G27
HSTC1_TX_n4	33	PIN_K28
HSTC1_TX_p4	35	PIN_K27
HSTC1_TX_n5	39	PIN_M28
HSTC1_TX_p5	41	PIN_M27
HSTC1_TX_n6	45	PIN_P28
HSTC1_TX_p6	47	PIN_P27
HSTC1_TX_n7	51	PIN_L28
HSTC1_TX_p7	53	PIN_L27
HSTC1_TX_n8	57	PIN_M26
HSTC1_TX_p8	59	PIN_M25
HSTC1_CLKOUT_n1	63	PIN_AF5
HSTC1_CLKOUT_p1	65	PIN_AE5
HSTC1_TX_n9	69	PIN_N26
HSTC1_TX_p9	71	PIN_N25
HSTC1_TX_n10	75	PIN_P26
HSTC1_TX_p10	77	PIN_P25
HSTC1_TX_n11	81	PIN_R28
HSTC1_TX_p11	83	PIN_R27
HSTC1_TX_n12	87	PIN_T26
HSTC1_TX_p12	89	PIN_T25
HSTC1_TX_n13	93	PIN_R26

HSTC1_TX_p13	95	PIN_R25
HSTC1_TX_n14	99	PIN_U28
HSTC1_TX_p14	101	PIN_U27
HSTC1_TX_n15	105	PIN_V28
HSTC1_TX_p15	107	PIN_V27
HSTC1_TX_n16	111	PIN_W27
HSTC1_TX_p16	113	PIN_W28
HSTC1_TX_n17	117	PIN_T22
HSTC1_TX_p17	119	PIN_T21
HSTC1_CLKOUT_2	123	PIN_H22
HSTC1_TX_n18	133	PIN_W26
HSTC1_TX_p18	135	PIN_W25
HSTC1_TX_n19	137	PIN_AC28
HSTC1_TX_p19	139	PIN_AC27
HSTC1_TX_n20	141	PIN_Y26
HSTC1_TX_p20	143	PIN_Y25
HSTC1_TX_n21	145	PIN_AA26
HSTC1_TX_p21	147	PIN_AA25
HSTC1_TX_n22	149	PIN_AB28
HSTC1_TX_p22	151	PIN_AB27
HSTC1_TX_n23	153	PIN_AB26
HSTC1_TX_p23	155	PIN_AB25
HSTC1_TX_n24	157	PIN_AD28
HSTC1_TX_p24	159	PIN_AD27
HSTC1_TX_n25	161	PIN_AD26
HSTC1_TX_p25	163	PIN_AC26
HSTC1_TX_n26	165	PIN_AF27
HSTC1_TX_p26	167	PIN_AE26
HSTC1_TX_n27	169	PIN_AE28
HSTC1_TX_p27	171	PIN_AE27
HSTC1_TX_n28	173	PIN_AC25
HSTC1_TX_p28	175	PIN_AC24
HSTC1_TX_n29	177	PIN_Y24
HSTC1_TX_p29	179	PIN_Y23

Table 3.4 Pin assignments of the HSTC1

3.4 Off-Chip Memory

The Cyclone[®] III PCI development board provides the large-capacity and high-speed memory interface.

3.4.1 DDR2 SO-DIMM Module

The board has a DDR2 SDRAM SO-DIMM memory interface with 64-bit data width. The target speed is 200 MHz DDR for a total theoretical bandwidth of nearly 25 Gb/s. Table 3.5 lists DDR2 SDRAM SO-DIMM pin-out as well as corresponding FPGA pin numbers.

Schematic Signal Name	Connector pin no.	FPGA Pin Name
DDR2_A0	102	PIN_G11
DDR2_A1	101	PIN_D15
DDR2_A2	100	PIN_E10
DDR2_A3	99	PIN_H15
DDR2_A4	98	PIN_A10
DDR2_A5	97	PIN_J15
DDR2_A6	94	PIN_F8
DDR2_A7	92	PIN_D7
DDR2_A8	93	PIN_F14
DDR2_CLK_P0	30	PIN_D8
DDR2_CLK_P1	164	PIN_J19
DDR2_CLK_N0	32	PIN_C8
DDR2_CLK_N1	166	PIN_H19
DDR2_A9	91	PIN_J13
DDR2_A10	105	PIN_F15
DDR2_A11	90	PIN_C7
DDR2_A12	89	PIN_B12
DDR2_A13	116	PIN_D24
DDR2_A14	86	PIN_A6
DDR2_A15	84	PIN_C6
DDR2_DQ0	5	PIN_C10
DDR2_DQ1	7	PIN_E11
DDR2_DQ2	17	PIN_C11
DDR2_DQ3	19	PIN_H13
DDR2_DQ4	4	PIN_B7
DDR2_DQ5	6	PIN_B6
DDR2_DQ6	14	PIN_A7
DDR2_DQ7	16	PIN_D10

DDR2_DQ8	23	PIN_D13
DDR2_DQ9	25	PIN_C13
DDR2_DQ10	35	PIN_E14
DDR2_DQ11	37	PIN_C14
DDR2_DQ12	20	PIN_C12
DDR2_DQ13	22	PIN_A12
DDR2_DQ14	36	PIN_B11
DDR2_DQ15	38	PIN_A11
DDR2_DQ16	43	PIN_C17
DDR2_DQ17	45	PIN_B18
DDR2_DQ18	55	PIN_A19
DDR2_DQ19	57	PIN_D20
DDR2_DQ20	44	PIN_C16
DDR2_DQ21	46	PIN_E17
DDR2_DQ22	56	PIN_C19
DDR2_DQ23	58	PIN_B19
DDR2_DQ24	61	PIN_C22
DDR2_DQ25	63	PIN_C21
DDR2_DQ26	73	PIN_A22
DDR2_DQ27	75	PIN_C24
DDR2_DQ28	62	PIN_E18
DDR2_DQ29	64	PIN_D21
DDR2_DQ30	74	PIN_B21
DDR2_DQ31	76	PIN_A21
DDR2_DQ32	123	PIN_A23
DDR2_DQ33	125	PIN_D22
DDR2_DQ34	135	PIN_E22
DDR2_DQ35	137	PIN_F21
DDR2_DQ36	124	PIN_B25
DDR2_DQ37	126	PIN_C25
DDR2_DQ38	134	PIN_A26
DDR2_DQ39	136	PIN_B26
DDR2_DQ40	141	PIN_AG17
DDR2_DQ41	143	PIN_AG18
DDR2_DQ42	151	PIN_AF15
DDR2_DQ43	153	PIN_AF16
DDR2_DQ44	140	PIN_AH17
DDR2_DQ45	142	PIN_AH18
DDR2_DQ46	152	PIN_AB16

DDR2_DQ47	154	PIN_AE17
DDR2_DQ48	157	PIN_AD17
DDR2_DQ49	159	PIN_AE19
DDR2_DQ50	173	PIN_AG22
DDR2_DQ51	175	PIN_AF24
DDR2_DQ52	158	PIN_AG21
DDR2_DQ53	160	PIN_AH21
DDR2_DQ54	174	PIN_AH22
DDR2_DQ55	176	PIN_AH23
DDR2_DQ56	179	PIN_AD18
DDR2_DQ57	181	PIN_AF20
DDR2_DQ58	189	PIN_AE21
DDR2_DQ59	191	PIN_AF22
DDR2_DQ60	180	PIN_AE24
DDR2_DQ61	182	PIN_AE25
DDR2_DQ62	192	PIN_AG26
DDR2_DQ63	194	PIN_AH25
DDR2_DQS0	13	PIN_E12
DDR2_DQS1	31	PIN_D12
DDR2_DQS2	51	PIN_B17
DDR2_DQS3	70	PIN_D17
DDR2_DQS4	131	PIN_A25
DDR2_DQS5	148	PIN_AF17
DDR2_DQS6	169	PIN_AE18
DDR2_DQS7	188	PIN_AF26
DDR2_DM0	10	PIN_A8
DDR2_DM1	26	PIN_B10
DDR2_DM2	52	PIN_E15
DDR2_DM3	67	PIN_C20
DDR2_DM4	130	PIN_B23
DDR2_DM5	147	PIN_AC15
DDR2_DM6	170	PIN_AH19
DDR2_DM7	185	PIN_AF25
DDR2_CS_N0	110	PIN_G18
DDR2_CS_N1	115	PIN_D25
DDR2_CKE0	79	PIN_H8
DDR2_CKE1	80	PIN_E8
DDR2_BA0	107	PIN_D16

DDR2_BA1	106	PIN_A17
DDR2_BA2	85	PIN_H12
DDR2_RAS_N	108	PIN_J16
DDR2_CAS_N	113	PIN_D19
DDR2_WE_N	109	PIN_H16
DDR2_ODT0	114	PIN_E21
DDR2_ODT1	119	PIN_C26
DDR2_SCL	197	PIN_J17
DDR2_SDA	195	PIN_C23

Table 3.5 Pin assignments of the DDR2 SO-DIMM

Chapter 4

Setup PCI Board

This chapter describes how to setup the PCI board and driver on users' PC.

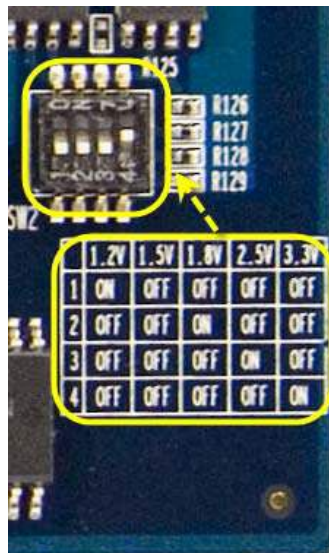
4.1 System Requirement

- Windows, 32-bits
- One 32 or 64 PCI slot
- Quartus Installed. Quartus 8.0 or 8.1 is recommended.
- USB-Blaster and USB Cable

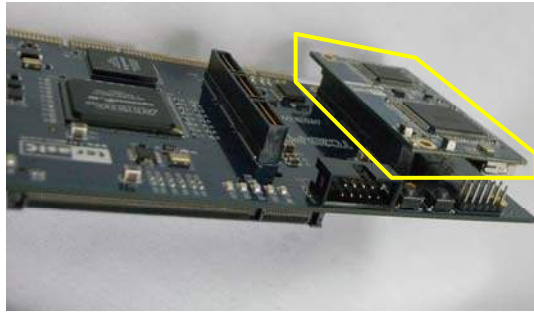
4.2 Hardware Installation: PCI Board

Follow these steps to install your PCI board into your computer:

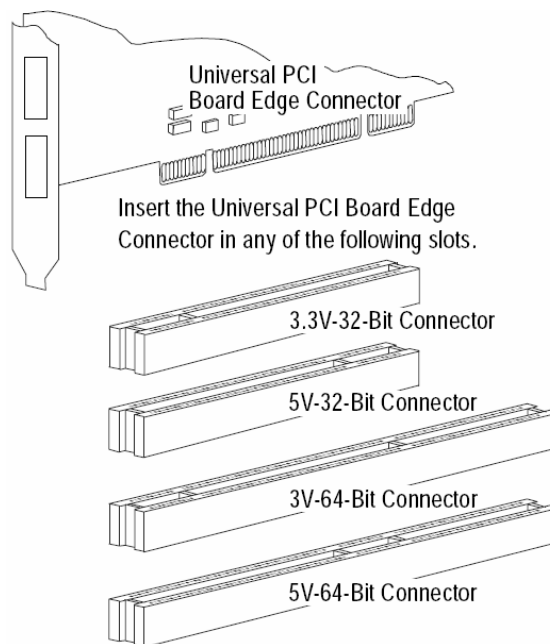
1. Switch SW2 to select the IO voltage level of HSTC on PCI board.



2. Make the connection between the daughter board and PCI board if your design needs it.



3. Switch off the computer and disconnect from the power socket.
4. Remove the cover of the PC.
5. Choose any open slot and insert PCI board.
 - * The Cyclone[®] III PCI development board has a Universal PCI Board edge connector. It can be inserted into any of the PCI slots.



6. Insert bracket screw and ensure that the board sits firmly in the PCI socket.
7. Replace the cover of the PC.
8. Reconnect all power cables and switch the power on.
9. The hardware installation is now complete.

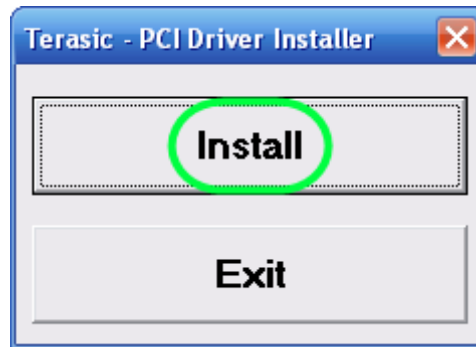
4.3 Software Installation: PCI Kernel Driver

Before users can use Terasic's PCI library to communicate the PCI board, PCI kernel mode driver should be installed in users' PC first.

The kernel driver is located in the "Install PCI Driver" folder of PCI CD-ROM. Please follow below procedures to install the kernel driver:

10. Copy the folder "Install PCI Driver" to your hard-disk.
11. Double click "PCI_DriverInstall.exe" to launch the installation program.

12. Click “Install” to start installing process.



13. It takes several second to install the driver. When installation is completed, as information dialog will popup.
14. Click “Exit” to close the installation program.

4.4 Install License File

To compile the project created by PCI system builder, users need to add a specified license to Quartus. The license file is located in the “license” folder of the PCI CD-ROM.

4.5 Diagnoses

Below shows the procedure to perform the diagnosis:

1. Make sure PCI board is installed on your PC.
2. Make sure PCI driver is installed on your PC.
3. Make sure Quartus is installed on your PC.
4. Copy the “Diagnose” folder in PCI CD-ROM to your hard-disk.
5. Download PCI_TEST.sof to PCI board.
6. Double click “PCI_TEST.exe” to start diagnosis process.
7. The diagnosis will check DDR2 and LED. When diagnosis is completed, the result will display on the console windows.

Chapter 5

PCI System Builder

This chapter describes how to quickly create a PCI project framework based on the software utility - PCI System builder.

5.1 Introduction

PCI System Builder is a Windows-Based utility. It can help users quickly and accurately to create a QUARTUS project. Figure 5.1 shows the graphical user interface of the utility.

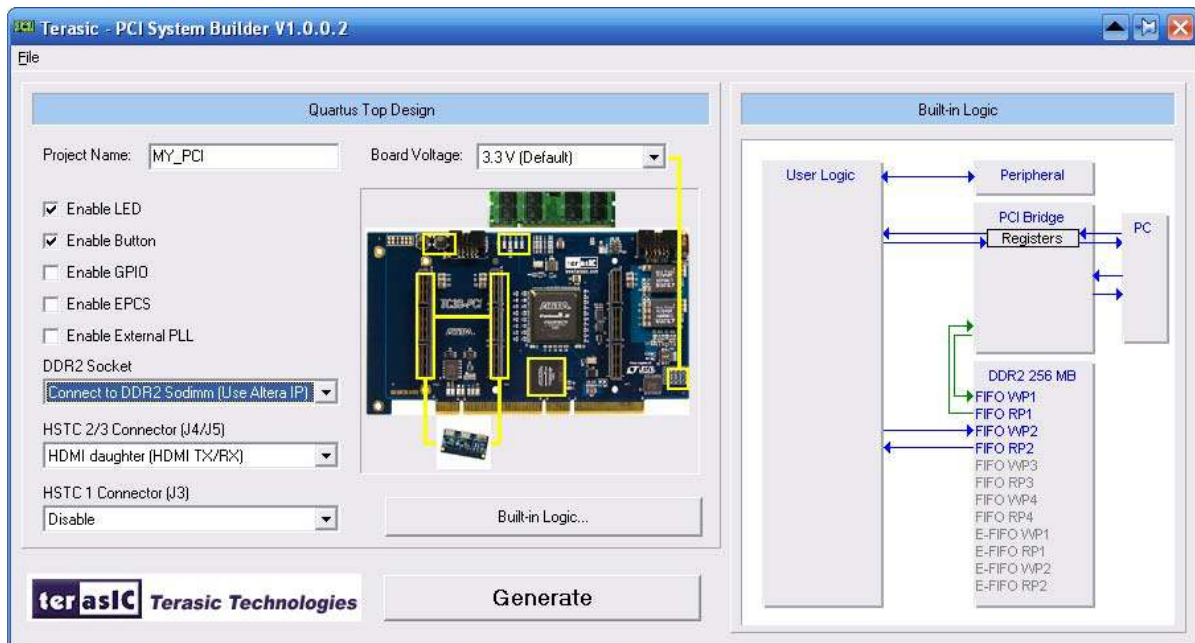


Figure 5.1 User interface of PCI System Builder

The utility consists of two major functions:

1. Quartus Top Design
2. Built-in Logic

For Quartus Top Design, the utility creates Quartus project and pin assignment according to users' selected peripherals and daughter boards. For Built-in Logic, the utility generates verilog code according to users' configuration for PCI Bridge, DDR2 multi-ports, and custom registers. If PCI Bridge is to be included, the driver and library for the PC side will also be created.

5.2 Quartus Top Design

Figure 5.2 shows the user interface of Quartus Top Design. User can select desired peripherals and daughter boards on the users interface.

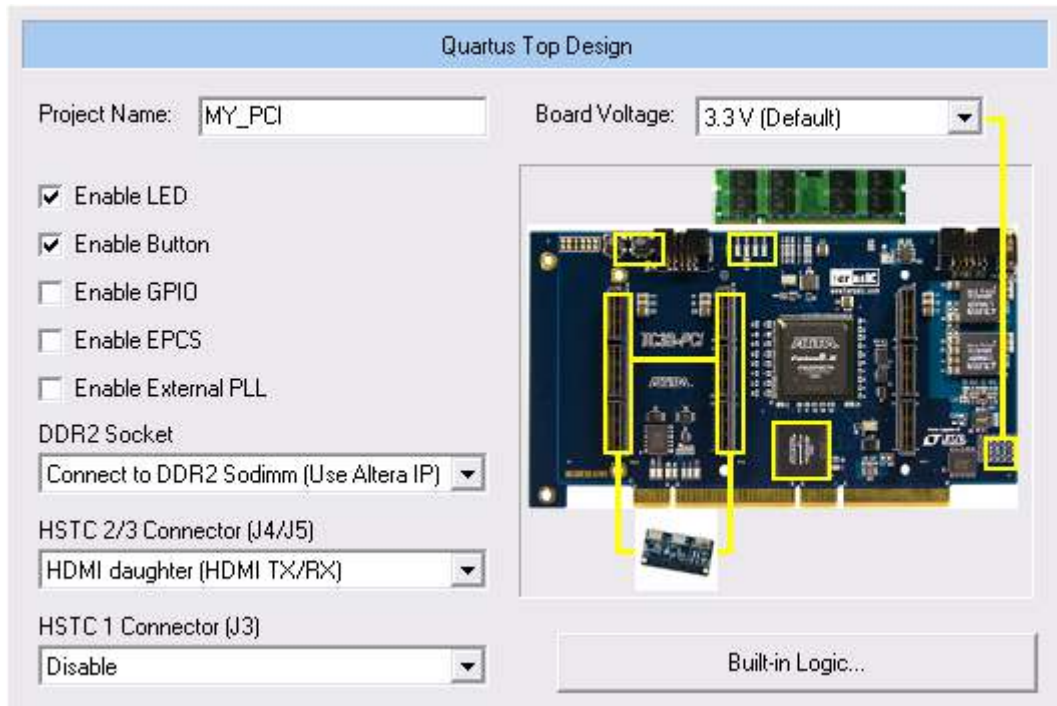


Figure 5.2 User Interface Quartus Top Design

In the Project Name field, users can input a desired name. It will be used as the name of Quartus project, top-design file, and the folder to store the Quartus project. For Board Voltage pull-down list, users can select the IO-Standard voltage of the HSTC connectors on PCI board. The voltage must be consistent with the daughter boards attached to the PCI board. Please select 3.3V for Terasic daughter boards. As shown in Figure 5.3, users must select the correct board voltage carefully, or the hardware could be damaged.

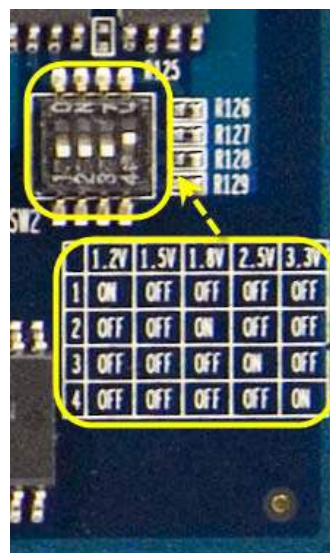


Figure 5.3 Board Voltage Selections

For peripheral selections, users can directly check the desired peripherals. The associated component will be highlighted with yellow rectangle. For daughter boards, users can select desired daughter board from the associated pull-down list. The photo of selected daughter boards will be displayed.

5.3 Built-in Logic

If users wish to include Built-in Logic in the Quartus project, click “Built-in Logic...” button and a Logic Configuration dialog will pop up, as shown in Figure 5.4.

Note. All digitals in the dialog are interpreted as a hexadecimal value.

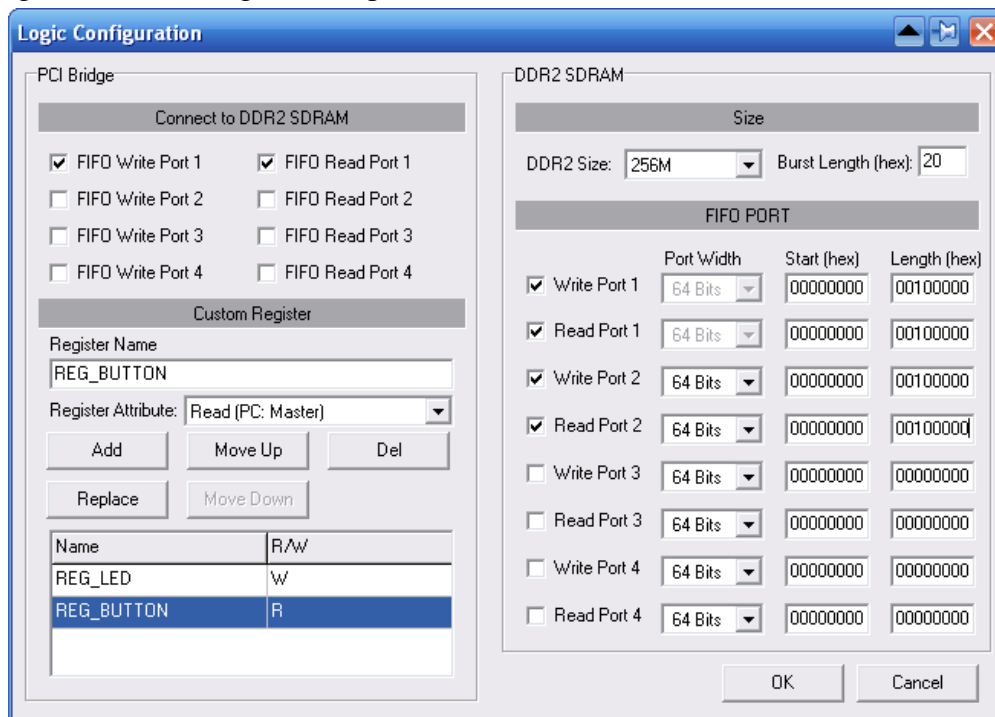


Figure 5.4 Built-in Logic Configurations

For DDR2 SDRAM, users must select SDRAM Size and Burst length. The burst length must be larger than or equal to 0x10. For FIFO port, the built-in logic can offer up to 4 FIFO-write ports and 4 FIFO-read ports. For each FIFO port, users need to specify its bus width (unit in bit), start address (byte address), and FIFO length (unit in byte).

There are some constraints for the value of start address and FIFO length:

- The value of start address must be multiple of 32.
- The FIFO length must be multiple of 32 and larger than or equal to 32 x (burst length)

For ENHANCED Port, the built-in logic can offer 2 enhanced-write ports and 2 enhanced-read ports at maximal. For each enhanced port, users need to specify its bus width, unit in byte. The

enhanced port size is as assumed as same as DDR2 SDRAM size.

If users wish to access the DDR2 from PC, they can tick the associated DDR2 ports in “Connect DDR2 SDRAM” group. For enhanced ports, only one enhanced-write port and one enhanced-read port can be connected to PCI Bridge at the same time. Moreover, if a DDR2 port is connected, its bus width will be fixed to be 64-bits.

If users wish to perform remote control from PC, custom registers can be added. The attribute of each register is neither read-only nor write-only. The size of each register is fixed to 32-bits. To add a register, users need to specify register name and attribute first and click “Add”; To delete an existed register, users need to select the existed register and click “Del”; To modify the name or attribute of an existed register, users need to select the register first, then modify the name or attribute, finally click “Replace”. Users can also change the register sequence by clicking “Move Up” and “Move Down”.

5.4 Save Configuration

Once users finish the configuration for top-level design and built-in logic, they can save the configuration into a file by selecting “File → Save Project As...”, as shown in Figure 5.5. Users can reload the configuration afterwards by selecting “File → Open Project...”.

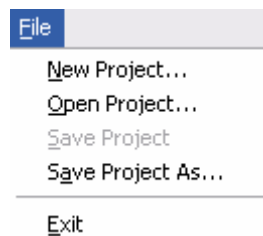


Figure 5.5 Configuration Save and Load

5.5 Generated Code

After user finish Quartus top design and built-in logic configuration, just click “Generate” to generate desired codes. Some of the generated file are naming based on the project name. Assume the project name is called as “MY_PCI”, the generated files will include:

- QUARTUS Project, contains:
 - QUARTUS Project (MY_PCI.QPF)
 - QUARTUS Top-Design File (MY_PCI.V)
 - QUARTUS Pin Assignment File (MY_PCI.QSF)
 - QUARTUS timing constrain file (MY_PCI.SDC)

- HTML Design Document (**MY_PCI.HTM**)
- PCI System-Builder Configuration File (**MY_PCI.PSC**)

- User Logics, contains:
 - PCI Bridge Logic: Top design file is PCI_Interface.v
 - DDR2 Multi-Port Logic: Top design files is Multi_Port_Controller.v
 - Custom Register logic: Top design files is User_Logic.v

- Windows Driver, contains:
 - PCI Library and Header files:
 - ✓ Terasic_API.dll
 - ✓ Terasic_FPGA.dll
 - ✓ Wdapi921.dll
 - ✓ FPGA_BOARD.cpp
 - ✓ FPGA_BOARD.h
 - ✓ Terasic_API.h
 - System header file: pci_system.h
 - Control Panel Software Utility: PCI_ControlPanel.exe
 - PCI Control Interface File (**MY_PCI.PCI**)

The generated Quartus Project and User Logic are located at the sub-directory under the folder where the PCI system builder is executed. The sub-directory name is as same as the name specified in the Project Name. The Windows Driver is located at the folder “**PC_CODE**” under the sub-directory.

In the Quartus Project, users can add their logic in the verilog file User_Logic.v. All of desired peripherals, daughter boards, and control pins are included in this module. The PCI System-Bulider Configuration File (.PSC) contains the project configuration in PCI system builder. Users can select the menu “file→open project...” in PCI system builder to open this file.

For Windows Driver, the kernel PCI driver is not includes in the “**PC_CODE**” folder. The kernel PCI driver should be installed before calling the PCI library API. For detail installation, per refer to the section **Installation of PCI kernel driver** in the next chapter.

The PCI Library includes Terasic_API.DLL , Terasic_FPGA.DLL, and WDAPI921.DLL. Uses can call the exported API in the Terasic_API.DLL to communicate with the PCI board. The System Header File pci_system.h defined the address of custom registers defined in built-in logic. Users’ application software can use these constants to specified desired custom register. PCI_ControlPanle.exe is a software utility for users to remove control the PCI board. Before access the PCI board, this utility inquires users to input the PCI Control Interface File (.PCI) that contain

the control interface specified in the built-in logic configuration dialog.

Chapter 6

Host Software Library and Utility

The PCI Kits provide necessary PCI driver/library and PCI utility on host site, so users can easily control the PCI board. Users must to install PCI kernel driver before PCI library and utility can work well.

Note. The PCI driver only supports 32-bits MS Windows.

6.1 PCI Software Stack

Figure 6.1 shows the PCI software stack. To communicate with the PCI board, Users Application should dynamically load the Terasic_API.dll and call the exported API. Also, users need to include Terasic_API.h into their C/C++ project.

If users' project is C++ project, they can refer to FPGA_BOARD.cpp and FPGA_BOARD.h which implement the DYNAMIC DLL LOADING procedure. The implemented class name is TFPGA_BOARD. FPGA_BOARD.h includes Terasic_API.h, and pci_system.h.

The low-level PCI driver is called WinDriver which is developed by Jungle Company. It includes wdapi921.dll and windrvr6.sys. In this kit, the PCI driver only supports 32-bits Windows. Also, users are not allowed to call wdapi921.dll directly due to license limitation. For 64-bits Windows and other OS platform, users need to develop the driver by their self or purchase development kits from Jungle Company.

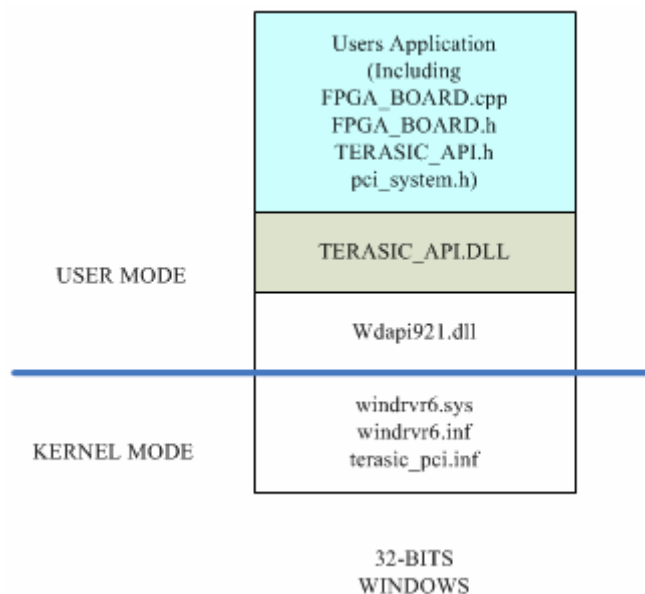


Figure 6.1 PCI Software Stack

6.2 Data Structure in TERASIC_API.h

The data structure is shown below. APP_DDR2_PORT_ID enumerates the ID for DDR2 FIFO port. The calling convention is defined as “FAR PASCAL”. The handle of FPGA board is defined as a pointer. The address of register is defined as 32-bits unsigned integer, the value of register is defined as 32-bits unsigned integer, and id of DDR2 FIFO PORT is defined as 32-bits unsigned integer. The interrupt service routine prototype is also defined.

```
#define TERASIC_API FAR PASCAL
typedef void *FPGA_BOARD;
typedef DWORD FPGA_REG_ADDRESS;
typedef DWORD FPGA_DDR2_PORT_ID;
typedef DWORD FPGA_REG_TYPE;
typedef void (TERASIC_API *FPGA_ISR)(void);
```

6.3 API List of TERASIC_API.DLL

Below table shows the exported API of TERASIC_API.DLL

API Name	API Description
System Function	
SYS_BoardNum	Return the number of FPGB available on your system.
SYS_GetDLLVersion	Retrieve the version of the software kits

FPGA Control Function	
FPGA_Connect	Connect to a specified FPAG board.
FPGA_Disconnect	Disconnect the connected FPAG board.
Information	
FPGA_IsReady	Check whether the FPGA is configured.
FPGA_GetFPGAVersion	Retrieve the version of build-in logic
FPGA_GetTickCount	Read the tick count, unit in ms, from FPGA counter logic.
FPGA Custom Register Access Function	
FPGA_RegWrite	Write data to a specified register.
FPGA_RegRead	Read data from a specified register.
FPGA DDR2 FIFO Port Access Function	
FPGA_FifotDmaWrite	Write a block of data to a memory port in DMA mode
FPGA_FifoDmaRead	Read a block of data from a memory port in DMA mode
FPGA_PortReset	Reset DDR2 port
FPGA_PortFlush	Flush DDR2 read port
Interrupt Function	
FPGA_RegisterISR	Register interrupt callback function
Bridge	
FPGA_GetBridgeVersion	Retrieve the version of the pci bridge hardware
FPGA_BridgeReset	Reset bridge circuit.

6.4 API Description of TERASIC_DLL

This section will explain the PCI library API in details.

Function Prototype	Function Description
<pre> BOOL TERASIC_API SYS_BoardNum(WORD wVendorID, WORD wDeviceID, WORD *pwBoardNum); </pre>	<p>Function: Query the number of PCI boards installed on the host.</p> <p>Parameters: wVendorID: Specifies the vendor ID of the target PCI board.</p> <p>wDeviceID: Specifies the device ID of the target PCI board.</p> <p>pwBoardNum: Points to the buffer to retrieve the number of PCO boards installed on the host.</p> <p>Return Value:</p>

	<p>If the function succeeds, the return value is true. Otherwise, the return value is false.</p>
<pre> BOOL TERASIC_API SYS_GetDLLVersion(DWORD *pdwVersion); </pre>	<p>Function: Query the software version of TERAISC_API.DLL.</p> <p>Parameters: pdwVersion: Points to the buffer to retrieve the version information.</p> <p>Return Value: If the function succeeds, the return value is true. Otherwise, the return value is false.</p>
<pre> BOOL TERASIC_API FPGA_Connect(FPGA_BOARD *phFPGA, WORD wVendorID, WORD wDeviceID, WORD wBoardIndex); </pre>	<p>Function: Connect to a specified PCI board.</p> <p>Parameters: phFPGA: Points to the buffer to retrieve the driver handle of the target PCI board.</p> <p>wVendorID: Specifies the vendor ID of the target PCI board.</p> <p>wDeviceID: Specifies the device ID of the target PCI board.</p> <p>wBoardIndex: Specifies the board index of the target PCI board. The index of first board is zero.</p> <p>Return Value: If the function succeeds, the return value is true. Otherwise, the return value is false.</p>
<pre> BOOL TERASIC_API FPGA_Disconnect(FPGA_BOARD hFPGA); </pre>	<p>Function: Disconnect the specified PCI board.</p> <p>Parameters: hFPGA: A handle to specify the target PCI board. The handle is returned by calling FPAG_Connect API.</p> <p>Return Value: If the function succeeds, the return value is true.</p>

	Otherwise, the return value is false.
<pre> BOOL TERASIC_API FPGA_IsReady (FPGA_BOARD hFPGA,); </pre>	<p>Function: Check whether the FPGA is configured. The FPGA circuit framework is assumed to be generated by the PCI system builder.</p> <p>Parameters: hFPGA: A handle to specify the target PCI board. The handle is returned by calling FPAG_Connect API.</p> <p>Return Value: If the FPGA is configured, the return value is true. Otherwise, the return value is false.</p>
<pre> BOOL TERASIC_API FPGA_GetFPGAVersion(FPGA_BOARD hFPGA, DWORD *pdwVersion); </pre>	<p>Function: Query the version of the PCI Framework RTL code embedded in Clylone III 3C125. The framework is automatically generated by the PCI system builder utility.</p> <p>Parameters: hFPGA: A handle to specify the target PCI board. The handle is returned by calling FPAG_Connect API.</p> <p>pdwVersion: Points to the buffer to retrieve the version information.</p> <p>Return Value: If the function succeeds, the return value is true. Otherwise, the return value is false.</p>
<pre> BOOL TERASIC_API FPGA_GetTickCount(FPGA_BOARD hFPGA, DWORD *pdwTickCount); </pre>	<p>Function: Query the tick-count, unit in ms, of the PCI Framework RTL code embedded in Clylone III 3C125. The tick-count logic is automatically generated by PCI system builder utility. When FPGA is reconfigured, the counter is reset to zero.</p> <p>Parameters: hFPGA: A handle to specify the target PCI board. The handle is returned by calling FPAG_Connect API.</p>

	<p>pdwTickCount: Points to the buffer to retrieve the tick-count value. The unit of the tick-count value is 1/1000 second.</p> <p>Return Value: If the function succeeds, the return value is true. Otherwise, the return value is false.</p>
<pre> BOOL TERASIC_API FPGA_RegRead (FPGA_BOARD hFPGA, FPGA_REG_ADDRESS RegAddr, FPGA_REG_TYPE *pRegValue); </pre>	<p>Function: Read data from a specified register.</p> <p>Parameters: hFPGA: A handle to specify the target PCI board. The handle is returned by calling FPAG_Connect API.</p> <p>RegAddr: Specifies the address of the target register. The address is defined in pci_system.h.</p> <p>pRegValue: Points to the buffer to retrieve the data value of the specified register.</p> <p>Return Value: If the function succeeds, the return value is true. Otherwise, the return value is false.</p>
<pre> BOOL TERASIC_API FPGA_RegWrite (FPGA_BOARD hFPGA, FPGA_REG_ADDRESS RegAddr, FPGA_REG_TYPE RegValue); </pre>	<p>Function: Write data to a specified register.</p> <p>Parameters: hFPGA: A handle to specify the target PCI board. The handle is returned by calling FPAG_Connect API.</p> <p>RegAddr: Specifies the address of the target register. The address is defined in pci_system.h.</p> <p>RegValue: Specifies the data value written to the specified register.</p> <p>Return Value: If the function succeeds, the return value is true. Otherwise, the return value is false.</p>

<pre> BOOL TERASIC_API FPGA_FifoDmaRead(FPGA_BOARD hFPGA, FPGA_DDR2_PORT_ID DDR2PortID, void *pBuffer, DWORD dwBufSize); </pre>	<p>Function: Read data from a specified DDR2 FIFO Port.</p> <p>Parameters: hFPGA: A handle to specify the target PCI board. The handle is returned by calling FPAG_Connect API.</p> <p>DDR2PortID: Specifies the DDR2 FIFO Port for reading. The PORT ID is defined in pci_system.h.</p> <p>pBuffer: Points to the buffer to retrieve the data reading from the specified DDR2 FIFO PORT.</p> <p>dwBufSize: Specifies the size, in bytes, of the buffer specified by the pBuffer parameter.</p> <p>Return Value: If the function succeeds, the return value is true. Otherwise, the return value is false.</p>
<pre> BOOL TERASIC_API FPGA_FifoDmaWrite(FPGA_BOARD hFPGA, FPGA_DDR2_PORT_ID DDR2PortID, void *pData, DWORD dwDataSize); </pre>	<p>Function: Write data to a specified DDR2 FIFO Port.</p> <p>Parameters: hFPGA: A handle to specify the target PCI board. The handle is returned by calling FPAG_Connect API.</p> <p>DDR2PortID: Specifies the DDR2 FIFO Port for writing. The PORT ID is defined in pci_system.h.</p> <p>pData: Points to the buffer containing the data to be written to the specified DDR2 FIFO PORT.</p> <p>dwDataSize: Specifies the number of bytes to write to the specified DDR2 FIFO PORT.</p> <p>Return Value: If the function succeeds, the return value is true. Otherwise, the return value is false.</p>
<pre> BOOL TERASIC_API </pre>	<p>Function: Reset a specified DDR2 FIFO Port. When a fifo</p>

<pre> FPGA_PortReset(FPGA_BOARD hFPGA, FPGA_DDR2_PORT_ID DDR2PortID); </pre>	<p>port is reset, the fifo pointer is reset to the beginning of the fifo port.</p> <p>Parameters: hFPGA: A handle to specify the target PCI board. The handle is returned by calling FPAG_Connect API.</p> <p>DDR2PortID: Specifies the DDR2 FIFO Port for resetting. The PORT ID is defined in pci_system.h.</p> <p>Return Value: If the function succeeds, the return value is true. Otherwise, the return value is false.</p>
<pre> BOOL TERASIC_API FPGA_PortFlush(FPGA_BOARD hFPGA, FPGA_DDR2_PORT_ID DDR2PortID); </pre>	<p>Function: Flush a specified DDR2 FIFO Port. When a fifo port is flushed, the data in fifo are written to DDR2 immediately.</p> <p>Parameters: hFPGA: A handle to specify the target PCI board. The handle is returned by calling FPAG_Connect API.</p> <p>DDR2PortID: Specifies the DDR2 FIFO Port for resetting. The PORT ID is defined in pci_system.h.</p> <p>Return Value: If the function succeeds, the return value is true. Otherwise, the return value is false.</p>
<pre> BOOL TERASIC_API FPGA_RegisterISR(FPGA_BOARD hFPGA, FPGA_ISR ISR_Function); </pre>	<p>Function: Register/Unregister an interrupt service routine for the PCI interrupt event.</p> <p>Parameters: hFPGA: A handle to specify the target PCI board. The handle is returned by calling FPAG_Connect API.</p> <p>ISR_Funciton: Specifies the location of interrupt service routine. If the value is NULL, the unregister interrupt service routine.</p>

	<p>Return Value: If the function succeeds, the return value is true. Otherwise, the return value is false.</p>
<pre> BOOL TERASIC_API FPGA_GetBridgeVersion(FPGA_BOARD hFPGA, DWORD *pdwVersion); </pre>	<p>Function: Query the version of PCI Bridge RTL code embedded in Cyclone III 3C25.</p> <p>Parameters: hFPGA: A handle to specify the target PCI board. The handle is returned by calling FPAG_Connect API.</p> <p>pdwVersion: Points to the buffer to retrieve the version information.</p> <p>Return Value: If the function succeeds, the return value is true. Otherwise, the return value is false.</p>
<pre> BOOL TERASIC_API FPGA_BridgeReset (FPGA_BOARD hFPGA); </pre>	<p>Function: Reset bridge circuit located at Cyclone III 3C25.</p> <p>Parameters: hFPGA: A handle to specify the target PCI board. The handle is returned by calling FPAG_Connect API.</p> <p>Return Value: If the function succeeds, the return value is true. Otherwise, the return value is false.</p>

6.5 PCI Control Panel Utility

Except for calling PCI Library, users also can use the PCI Control Panel Utility to communicate with the PCI board. The execution file name of this utility is named as, PCI_ControlPanel.exe. It is automatically generated by PCI System builder. It is located in the “PCI_CODE” folder. Figure 6.2 shows the user interface of the PCI control panel utility.

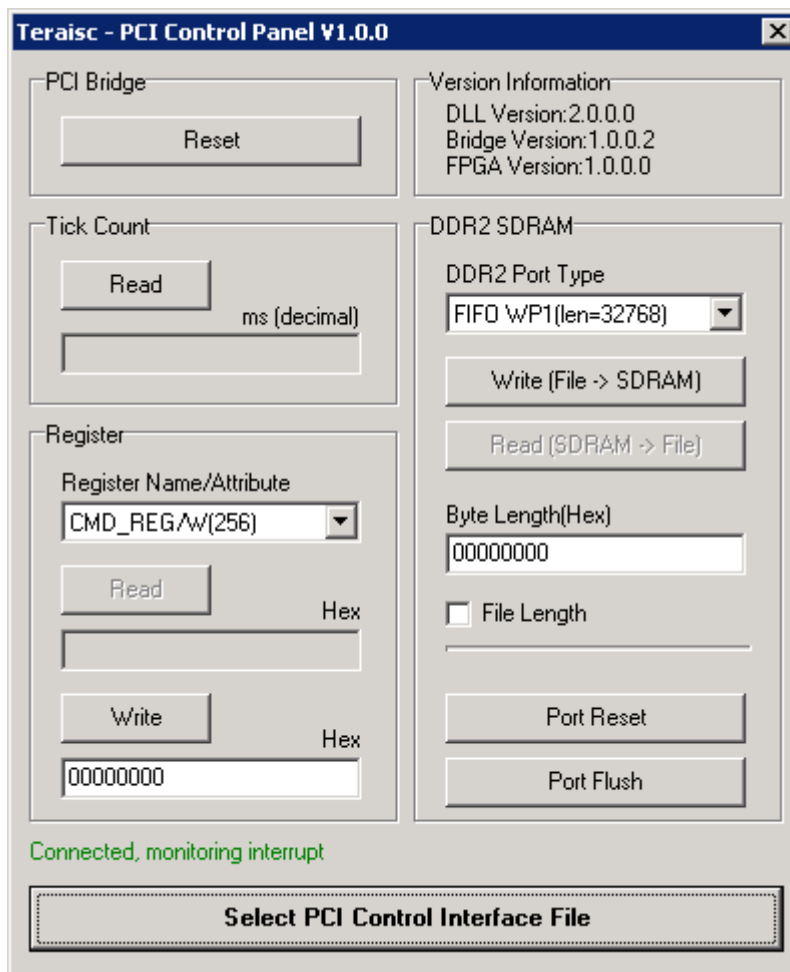


Figure 6.2 User Interface of PCI Control Panel

Before launch this utility, users need to install the PCI board and driver first, then download .SOF to the PCI Board. After launch this utility, a “Connected, monitoring interrupt” message will appears if it connects the PCI board successfully. In additional, the version information also is displayed in the version information group. Then, users need to select “PCI Control Interface File” first. This file is automatically generated by the PCI System Builder. It is located at the “PC_CODE” folder and its extension name is “.PCI”. Users can click “Select PCI Control Interface File” button to select the .PCI file.

If users implement interrupt in user logic, users can use this utility to verify. When hardware interrupt happen, a message dialog will popup to inform users that interrupt occurs.

If users implement register access in user logic, users also can use this utility to verify. First, users need to select the desired register in the pull down menu Register Name/Attribute. Then, click “Read” and “Write” button to access the register.

If users enable the DDR2 FIFO, users can use this utility to verify, too. First, users need to select the

desired DDR2 FIFO PORT in the pull down menu DDR2 Port Type. Then, click “Read” and “Write” button to access the data of DDR2 FIFO. In “Read” function, the data from fifo port will be written to a specified file. In “Write” function, the file content will be written to fifo port with a given length, unit in bytes. In additional, click “Port Reset” and “Port Flush” to reset and flush the DDR2 FIFO, individually.

The PCI System Builder automatically generated a tick-count circuit. To test this function, users can click “Read” in the Tick Count Group.

Chapter 7

Reference Design

This chapter illustrates some example showing how to develop Quartus project based on PCI system builder. All of these reference designs are developed by Quartus 8.1.

7.1 Remote Control LED

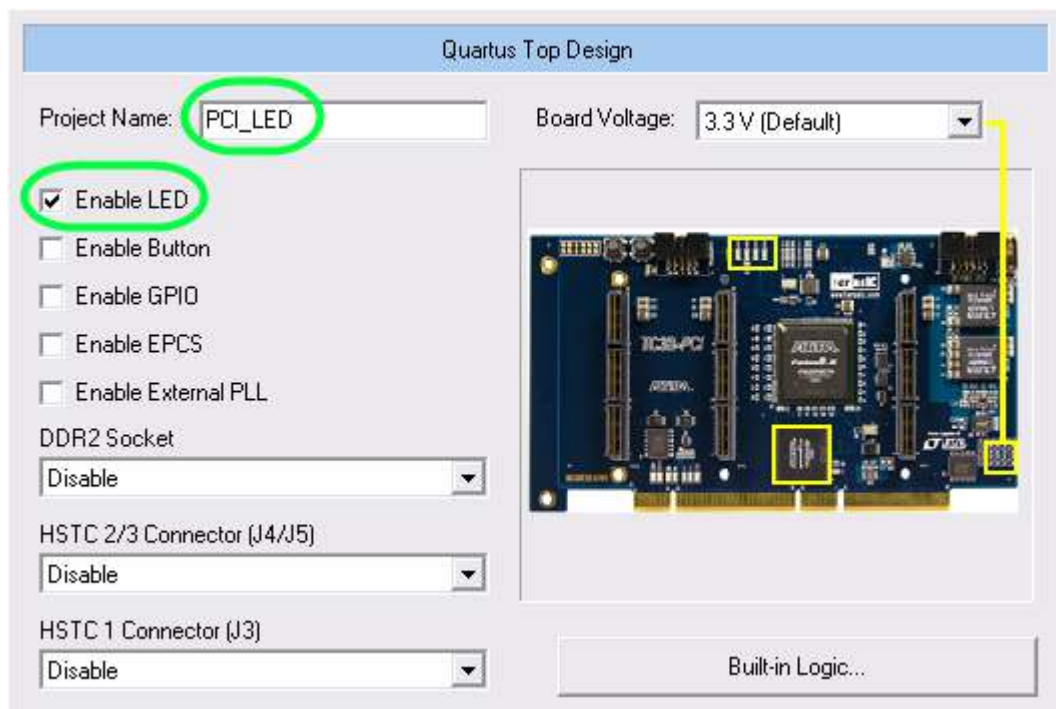
■ Function Description

This design shows how to implement remote control the LED in the PCI board. In host site, application call register access API to control the LED on the PCI board.

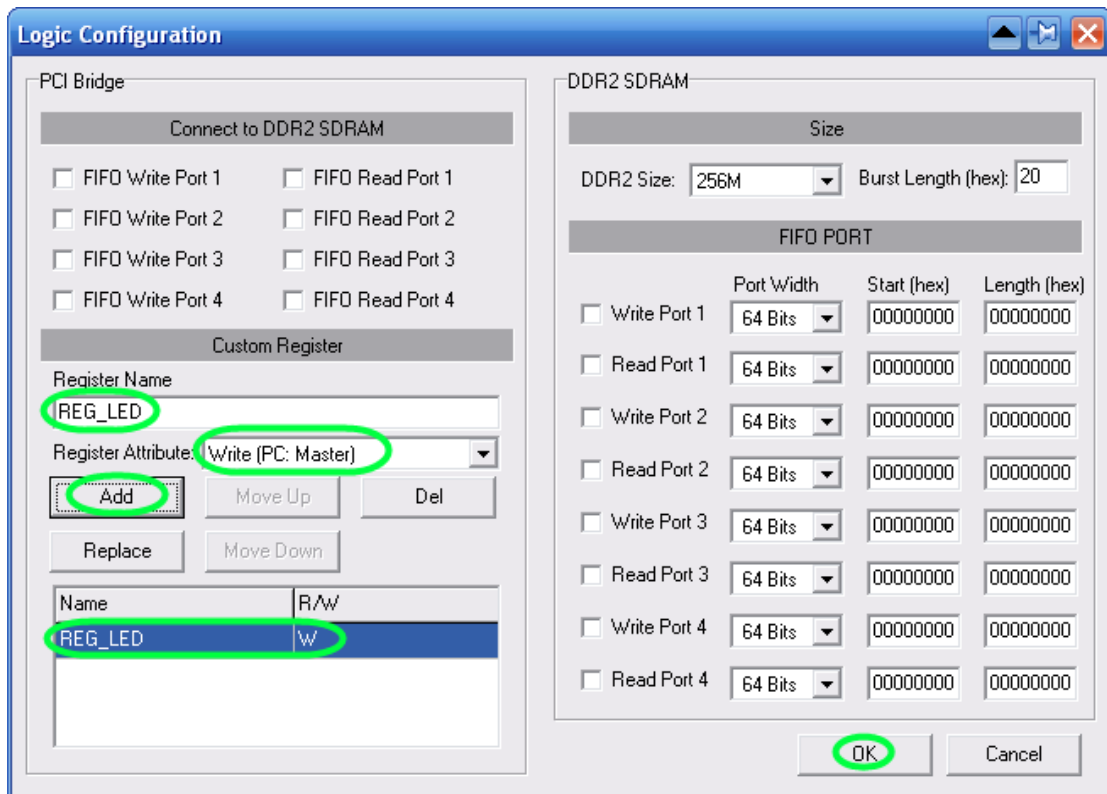
■ Build Project by PCI System Builder

Below shows the procedure to create the project framework by using PCI System builder utility.

1. Launch PCI_SystemBuilder.exe and specify project name and enable LED.



2. Click “Built-in Logic...” and Logic Configuration Dialog will pop up. Add “REG_LED” register with WRITE attribute and click “OK”

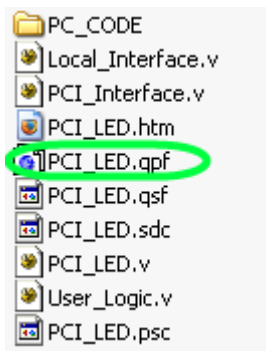


3. Click “Generate” to generate codes.

■ Add User Logic in Quartus Project

Below show the procedures to add user logic in the generated Quartus Project.

1. The generate codes are shown as below. Open the generated Quartus project by double clicking “PCI_LED.qpf”.



2. Open User_Logic.v and add “assign LED = iREG_LED;” statement.

```

39
40 //-----
41 //  Add your design here.
42 //-----
43
44 assign LED = iREG_LED;
45
46 endmodule
47

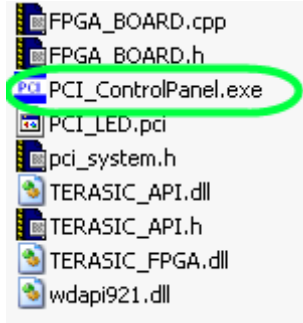
```

3. Compile the project and download the generated file PCI_LED.sof to the PCI board.

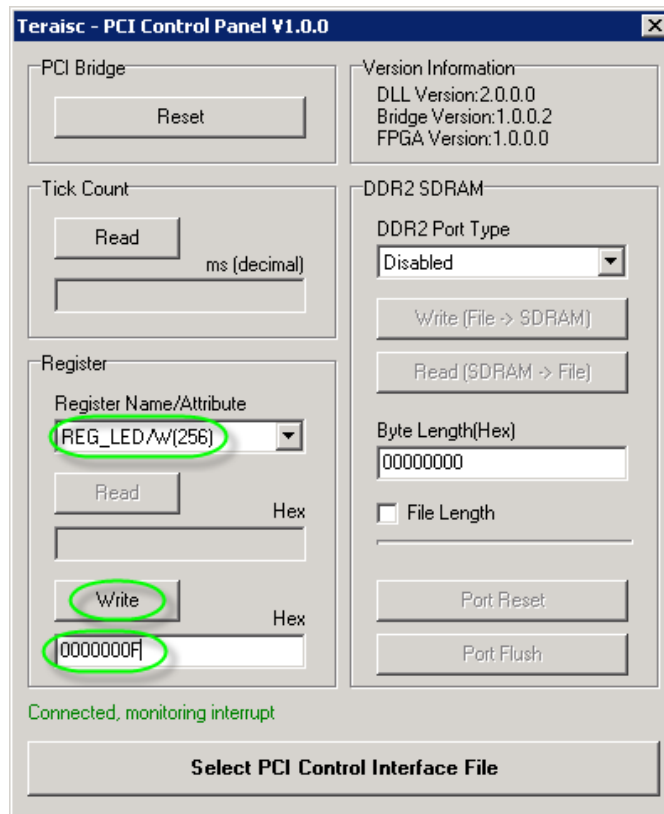
■ Remote Control by PCI_ControlPanel.exe

Below show the procedures to remote control the LED by PCI_ControlPanel.exe.

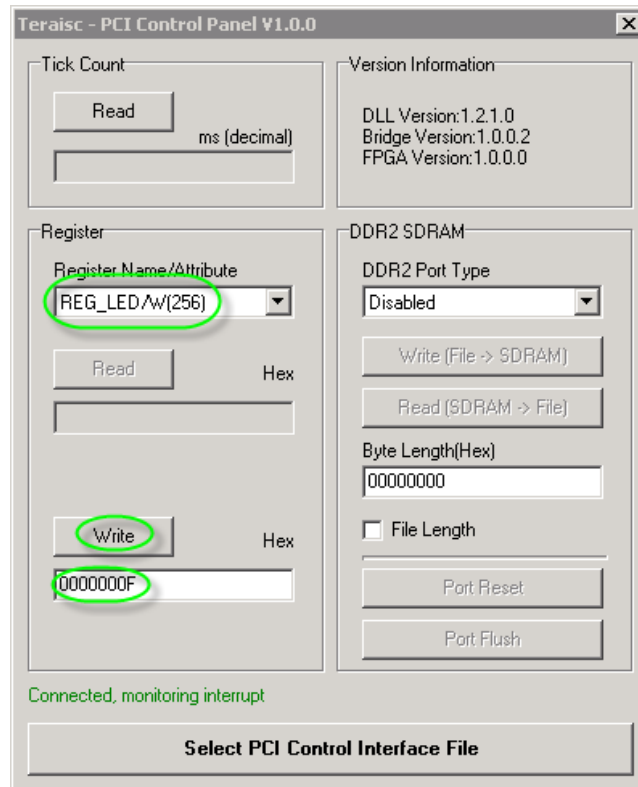
1. Launch PCI_ControlPanel.exe under the PC_CODE folder



2. Click “Select Configure File” to select PCI_LED.pci



3. Type “0000000F” in the write edit box of the Register group. Click “Write” and the LED will be turned off immediately.



■ Remote Control by Your C++ Program

Below show the procedures to remote control the LED by creating a C++ program.

1. Create a C++ project.
2. Copy FPGA_BOARD.cpp, FPGA_BOARD.h, TERASIC_API.h, and pci_system.h under the PC_CODE folder to the source code folder of your C++ project.
3. Copy TERASIC_API.DLL and wdapi921.dll under the PC_CODE folder to the execution file folder of your C++ project.
4. Include FPGA_BOARD.cpp into your C++ project.
5. Modify your main procedure as:

```
#include "FPGA_BOARD.h"

int main(int argc, char* argv[])
{
    TFPGA_BOARD Board;
    FPGA_REG_TYPE RegValue = 0x00;
    BOOL bSuccess = TRUE;

    printf("==== LED Demo =====\n");

    // check whether the PCI driver is available
    if (!Board.IsDriverAvailable()){
        printf("Failed to load the PCI driver.\n");
        getchar();
        return 0;
    }
}
```

```

// connect the PCI board
if (!Board.Connect()){
    printf("Failed to connect the PCI board.\n");
    getchar();
    return 0;
}

// make sure FPGA is configured
if (!Board.IsReady()){
    printf("FPGA is not configured. Please make sure .sof is downloaded.\n");
    getchar();
    return 0;
}

// start to control the LED
printf("LED blinking...\n");
while(bSuccess){
    bSuccess = Board.RegWrite(REGW_REG_LED, RegValue);
    if (!bSuccess){
        printf("Failed to set register.\n");
    }else{
        RegValue ^= 0x0F;
        Sleep(500);
    }
}

//
printf("Program is terminated.\n");
Board.Disconnect();
getchar();
return 0;
}

```

6. Compile and execute the code. (**Note.** If an error “fatal error C1010: unexpected end of file while looking for precompiled header directive” occurs while compiling, please disable the **Precompiled Headers** function in the VC++ project.)
7. Now, you are expected to see the LED on the PCI board is blinking.

Note. For first time to use the PCI board on your compiler, you should install the PCI kernel driver first.

■ Source Code:

The Quartus Project:

- ◆ Source code : PCI DC-ROM\reference_design\PCI_LED
- ◆ Development Tool: Quartus 8.1

The C++ Project:

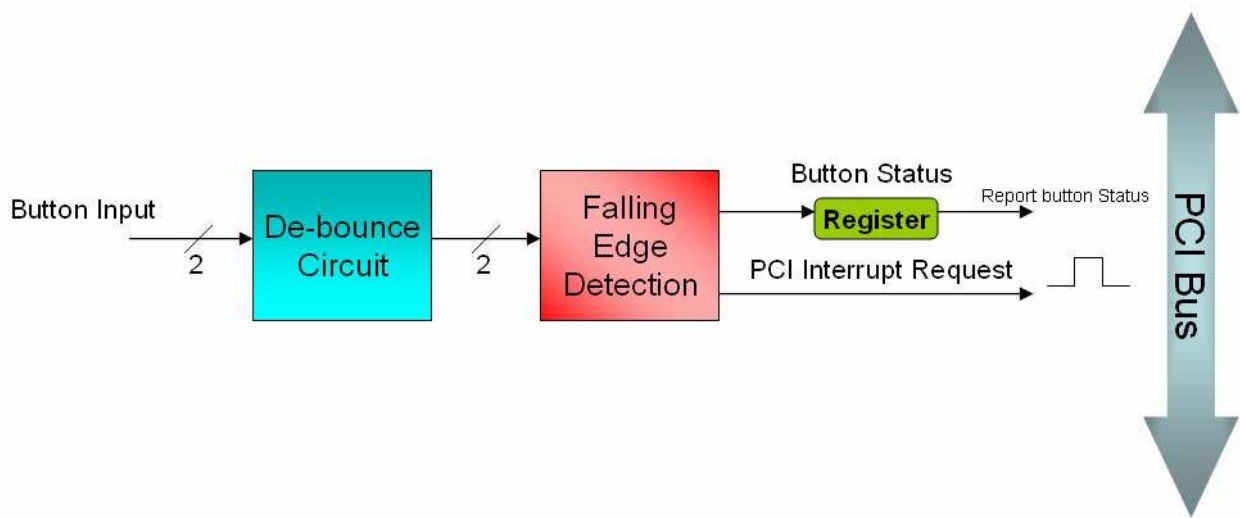
- ◆ Source code : PCI DC-ROM\reference_design\PCI_LED\PC_CODE\vb_led
- ◆ Development Tools: Visual C++ 6.0

7.2 Button IRQ

■ Function Description

This design shows how to implement an interrupt function. In the design, the interrupt is triggered by the BUTTON on the PCI board. In host site, the application should register an interrupt service routine first. When interrupt happen (users press button on the PCI board), the service routine is called. Then, the application reads the button status and shows the status on host's console window.

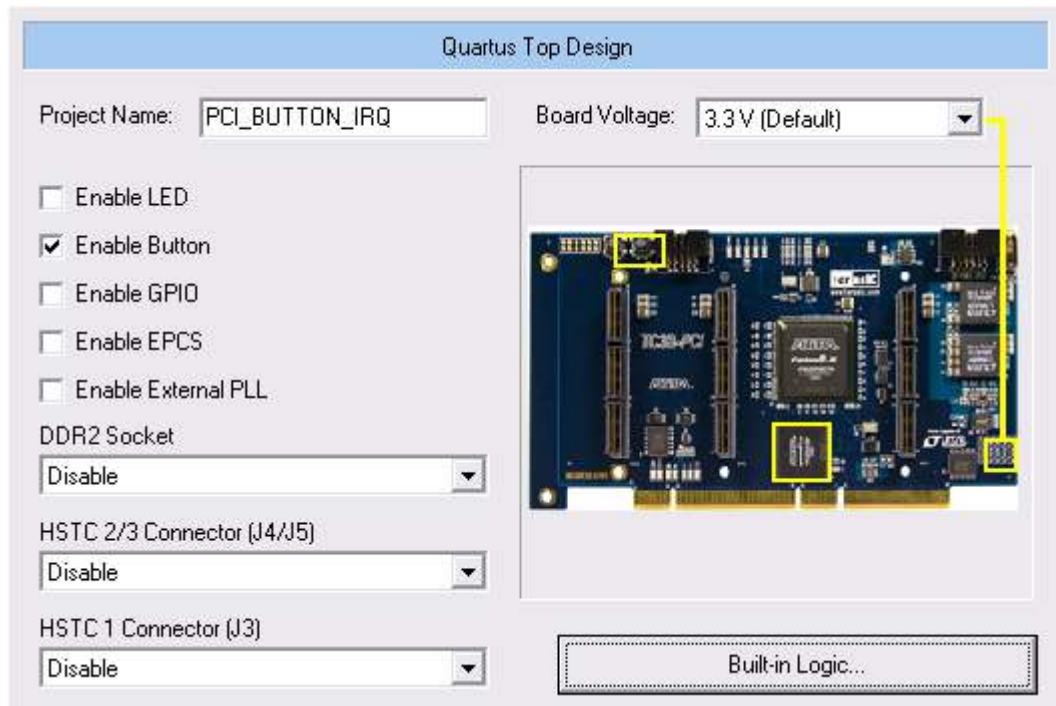
The following figure is the block diagram of Button IRQ reference design. Before detecting the button press, the input signals need to be processed by de-bounce circuit. Once the button is pressed, the interrupt request signal will active one clock cycle to trigger PCI interrupt, and the button status register will record it(which button had been pressed).



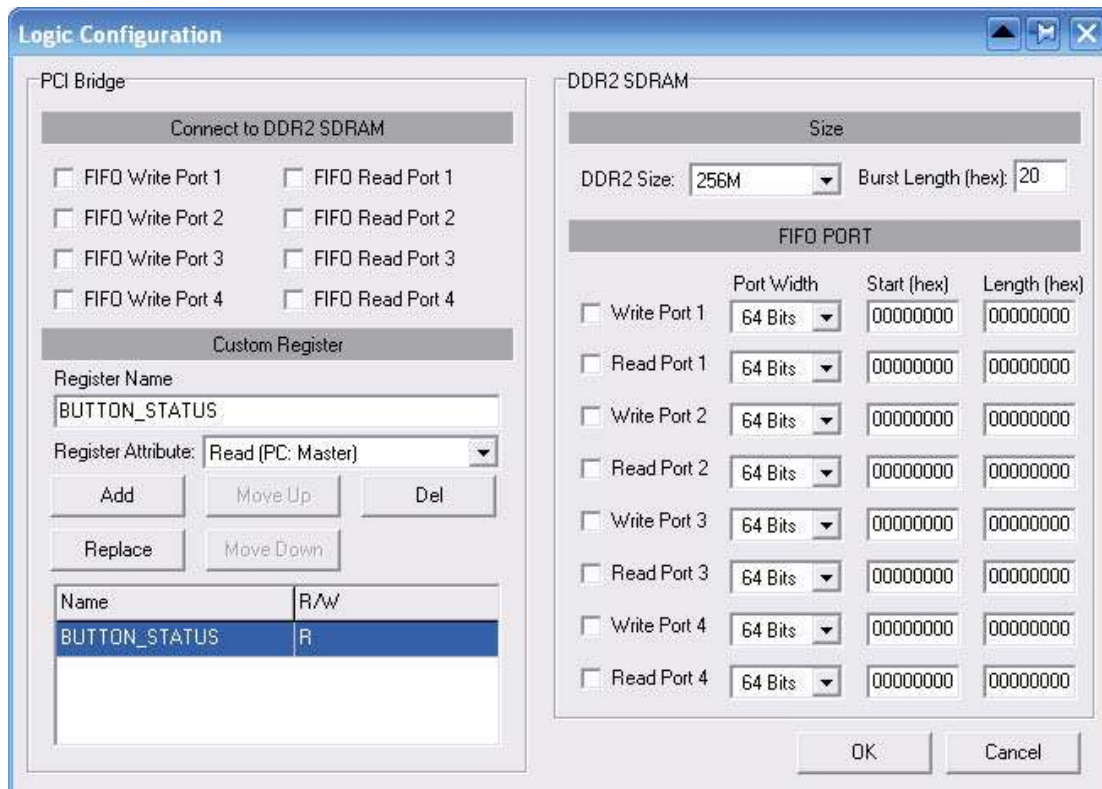
■ Build Project by PCI System Builder

Below shows the procedure to create the project framework by using PCI System builder utility.

1. Launch PCI_SystemBuilder.exe and specify project name and enable BUTTON.



2. Click “Built-in Logic...” and Logic Configuration Dialog will pop up. Add “REG_LED” register with WRITE attribute and click “OK”



3. Click “Generate” to generate codes.

■ Add User Logic in Quartus Project

1. Open the generated Quartus project by double clicking "PCI_BUTTON_IRQ.qpf".
2. Copy User_Logic.v to your current project folder from PCI_BUTTON_IRQ folder of example project in the CD-ROM.
3. Compile the project and download the generated file PCI_BUTTON_IRQ.sof

■ Implement Users' C++ Program

Below show the procedures to implement C++ program for interrupt handling

1. Create a C++ project.
2. Copy FPGA_BOARD.cpp, FPGA_BOARD.h and Terasic_API.h, and pci_system.h under the PC_CODE folder to the source code folder of your C++ project.
3. Copy Terasic_API.DLL and wdapi921.dll under the PC_CODE folder to the execution file folder of your C++ project.
4. Include FPGA_BOARD.cpp into your C++ project.
5. Modify your main procedure as:

```
#include "FPGA_BOARD.h"

static BOOL bCheckButton = FALSE;
void Terasic_API BUTTON_ISR(void){
    bCheckButton = TRUE;
}

int main(int argc, char* argv[])
{
    TFPGA_BOARD Board;
    FPGA_REG_TYPE RegValue;
    BOOL bSuccess = TRUE;

    printf("==== Button IRQ Demo ====\n");

    // check whether the PCI driver is available
    if (!Board.IsDriverAvailable()){
        printf("Failed to load the PCI driver.\n");
        getchar();
        return 0;
    }

    // connect the PCI board
    if (!Board.Connect()){
        printf("Failed to connect the PCI board.\n");
        getchar();
        return 0;
    }

    // make sure FPGA is configured
    if (!Board.IsReady()){
        printf("FPGA is not configured. Please make sure .sof is downloaded.\n");
        getchar();
        return 0;
    }
}
```

```

// register interrupt service routine
if (!Board.RegisterISR(BUTTON_ISR)){
    printf("Failed to register interrupt service routine.\n");
    getchar();
    return 0;
}

// start to control the LED
printf("Button monitoring...\n");

while(bSuccess){
    if (bCheckButton){
        bCheckButton = FALSE;
        bSuccess = Board.RegRead(REGR_BUTTON_STATUS, &RegValue);
        if (!bSuccess){
            printf("failed to read button's status.\n");
        }else{
            // change to high active
            RegValue ^= 0x03;
            //
            if ((RegValue & 0x03) == 0x03)
                printf("BUTTON 0 and 1 are pressed.\n");
            else if (RegValue & 0x01)
                printf("BUTTON 0 is pressed.\n");
            else if (RegValue & 0x02)
                printf("BUTTON 1 is pressed.\n");
        } // if
    } // if
} // while

//
printf("Pogram is terminated.\n");
getchar();

return 0;
}

```

6. Compile and execute the code. (**Note.** If an error “fatal error C1010: unexpected end of file while looking for precompiled header directive” occurs while compiling, please disable the **Precompiled Headers** function in the VC++ project.)
7. Now, click the button on PCI board. The relative information will be displayed in the console window.

Note. For first time to use the PCI board on your compiler, you should install the PCI kernel driver first.

■ Source Code:

The Quartus Project:

- ◆ Source code : \reference_design\PCI_BUTTON_IRQ
- ◆ Development Tool: Quartus 8.1

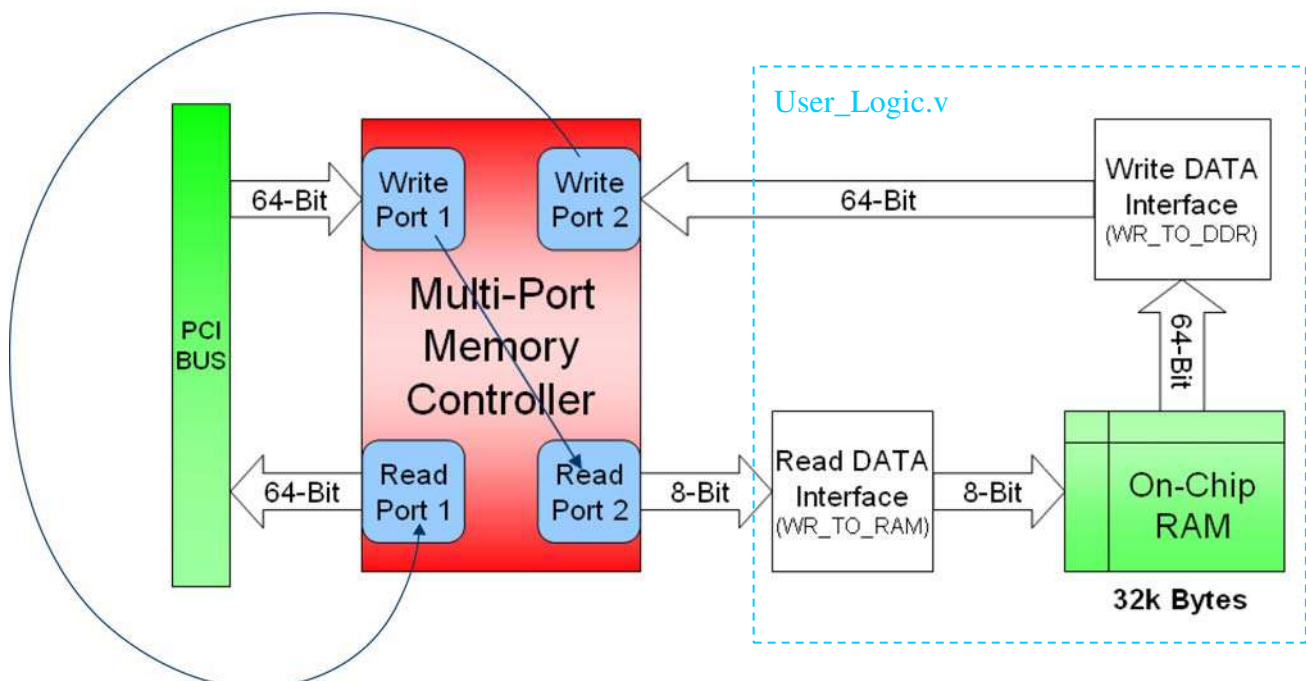
The C++ Project:

- ◆ Source code : \reference_design\PCI_BUTTON_IRQ\PC_CODE\vb_button_irq
- ◆ Development Tools: Visual C++ 6.0

7.3 DDR2 Access

■ Function Description

This section illustrates an example of how to access DDR2 from PC site and FPGA local site. This reference design provides a sample interface that connects PCI Bus to internal RAM through Multi-Port Memory Controller (MPMC). The following figure is the high level block diagram of the reference design. The Read DATA Interface and Write DATA Interface receive the command individually from host PC and execute them; The Read DATA Interface controls read port of MPMC and write the data to on-chip memory. Write DATA Interface reads the data from on-chip memory and writes the data into write port of MPMC.

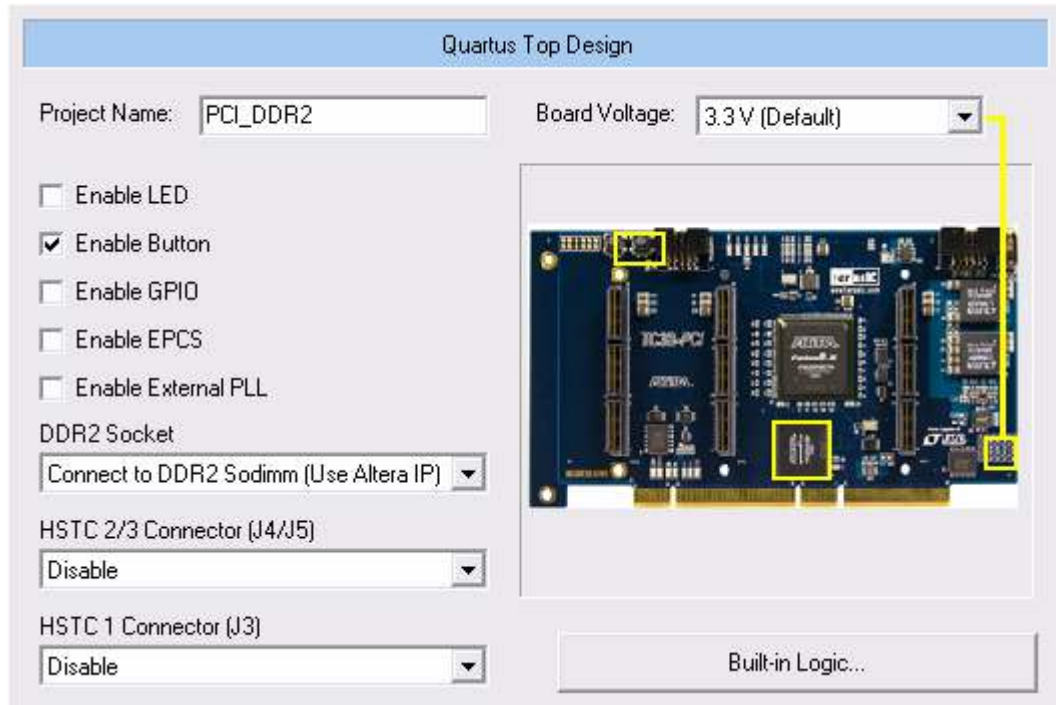


- ✓ Write port 1 and read port 2 are the same starting address and depth.
- ✓ Write port 2 and read port 1 are the same starting address and depth.

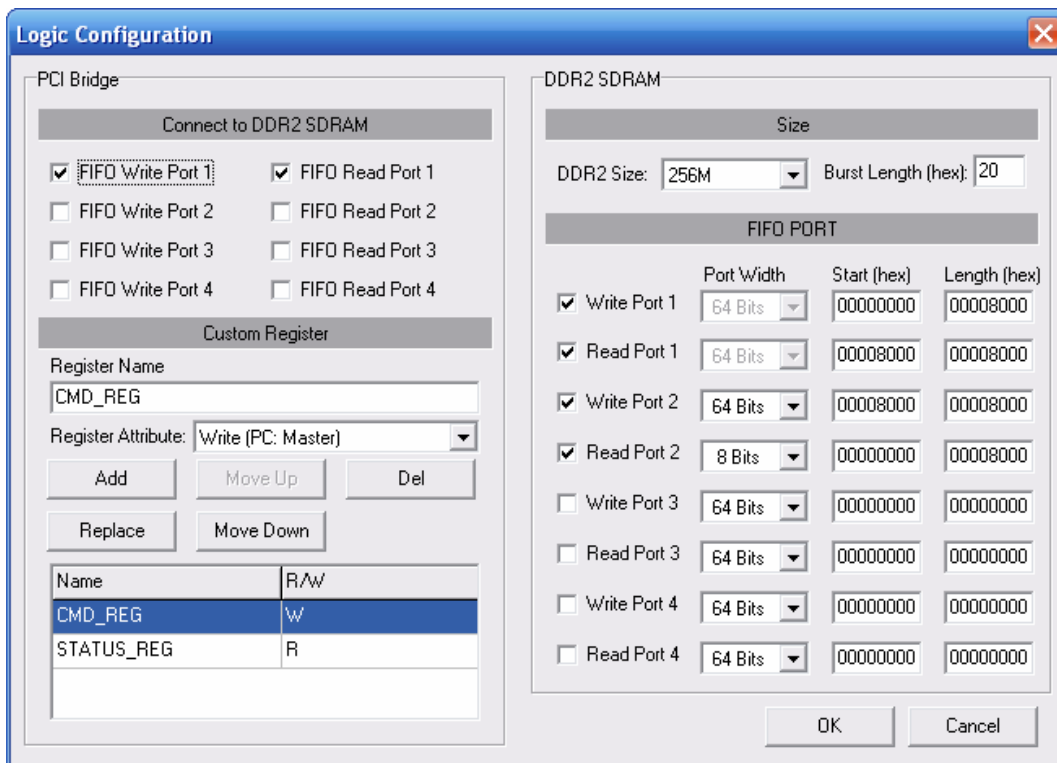
■ Build Project by PCI System Builder

The procedures are listed below:

8. Launch PCI_SystemBuilder.exe, then specify project name and select “Connect to DDR2 SO-DIMM (Use Altera IP).



9. Click “Built-in Logic...” and Logic Configuration Dialog will pop up. Add “CMD_REG” register with WRITE attribute and “STATUS_REG” register with READ attributes. Enable FIFO port Write Port 1, Read Port 1, Write Port 2, and Read Port 2. Specify the Start and Length of each fifo port as below. Finally, click “OK”



10. Click “Generate” to generate codes.

■ Implement Users’ C++ Program

Below show the procedures to implement C++ program for interrupt handling

1. Create a C++ project.
2. Copy FPGA_BOARD.cpp, FPGA_BOARD.h and TERASIC_API.h, and pci_system.h under the PC_CODE folder to the source code folder of your C++ project.
3. Copy TERASIC_API.DLL, TERASIC_FPGA.DLL, and wdapi921.dll under the PC_CODE folder to the execution file folder of your C++ project.
4. Include FPGA_BOARD.cpp into your C++ project.
5. Modify your main procedure as:

```
#include "FPGA_BOARD.h"

int main(int argc, char* argv[])
{
    const int nTestSize = 32*1024; // 32K-bytes
    int i;
    BYTE *pWrite, *pRead;
    DWORD dwValue;
    TFGA_BOARD Board;

    printf("==== DDR2 FIFO Write/Read Test =====\n");
```

```

if (!Board.IsDriverAvailable()){
    printf("Failed to load the PCI driver.\n");
    getchar();
    return 0;
}

if (!Board.Connect()){
    printf("Failed to connect the PCI board.\n");
    getchar();
    return 0;
}

// reset fifo port
Board.PortReset(APP_DDR2_FIFO_WP1);
Board.PortReset(APP_DDR2_FIFO_RP1);

pWrite = (BYTE *)::GlobalAlloc(GPTR, nTestSize);
pRead = (BYTE *)::GlobalAlloc(GPTR, nTestSize);
for(i=0;i<nTestSize;i++){
    *(pWrite+i) = i;
}

// write data to fifo port
printf("PC --> DDR2 FIFO WP1\n");
Board.FifoDmaWrite(APP_DDR2_FIFO_WP1, pWrite, nTestSize);
Board.PortFlush(APP_DDR2_FIFO_WP1);

// copy fifo data to local memroy
printf("DDR2 FIFO RP2 --> Local Memory\n");
Board.RegWrite(REGW_CMD_REG, 0x01); // send trigger message
Board.RegWrite(REGW_CMD_REG, 0x00);
dwValue = 0;
while((dwValue & 0x01) == 0){ // wait task done
    Board.RegRead(REGR_STATUS_REG, &dwValue);
}

// copy fifo data to local memroy
printf("DDR2 FIFO WP2 <-- Local Memory\n");
Board.RegWrite(REGW_CMD_REG, 0x02); // send trigger message
Board.RegWrite(REGW_CMD_REG, 0x00);
dwValue = 0;
while((dwValue & 0x02) == 0){ // wait taks done
    Board.RegRead(REGR_STATUS_REG, &dwValue);
}

// read data from fifo port
printf("PC <-- DDR2 FIFO RP1\n");
Board.PortReset(APP_DDR2_FIFO_RP1);
Board.FifoDmaRead(APP_DDR2_FIFO_RP1, pRead, nTestSize);

// compare
bool bSame = true;
for(i=0;i<nTestSize && bSame;i++){
    if (*(pWrite+i) != *(pRead+i)){
        bSame = false;
        printf("Test NG, pWrite[%d]=%d, pRead[%d] = %d\n",
            i, *(pWrite+i), i, *(pRead+i) );
    }
}

```



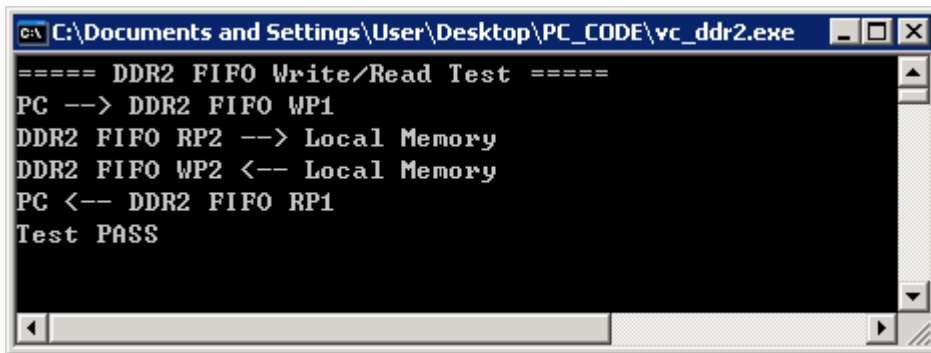
```
}
if (bSame)
    printf("Test PASS\n");

//
getchar();

//
::GlobalFree(pWrite);
::GlobalFree(pRead);

return 0;
}
```

6. Compile and execute the code. (**Note.** If an error “fatal error C1010: unexpected end of file while looking for precompiled header directive” occurs while compiling, please disable the **Precompiled Headers** function in the VC++ project.)
7. The test result will be displayed on the console window, as shown figure below.



Note. For first time to use the PCI board on your compiler, you should install the PCI kernel driver first.

■ Source Code:

The Quartus Project:

- ◆ Source code : \reference_design\PCI_DDR2
- ◆ Development Tool: Quartus 8.1

The C++ Project:

- ◆ Source code : \reference_design\PCI_DDR2\PC_CODE\vb_dds2
- ◆ Development Tools: Visual C++ 6.0

Chapter 8

Multi-Port Memory Controller

The Terasic Multi-Port Memory Controller provides a simple and high-performance solution to interface with external memory device. This controller supports up to 12 local ports which have two kinds of interface, one is simple port and another is enhanced port. The starting address and port depth can be only modified right away on enhanced port. Figure 8.1 gives the high-level block diagram of Terasic Multi-Port Memory Controller.

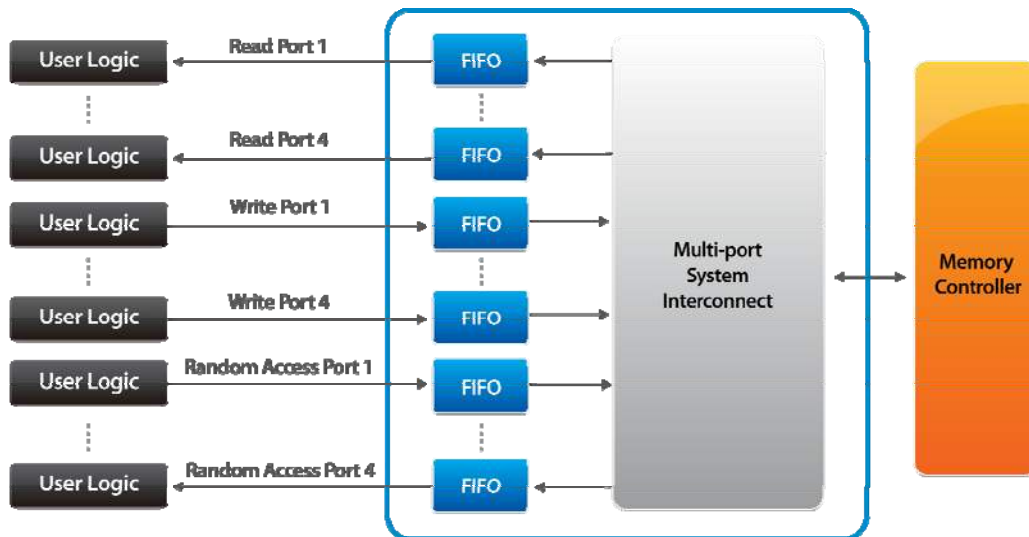


Figure 8.1 High level block diagram of the Multi-Port Memory Controller

8.1 Principle of Read/Write Port

This section will introduce how read/write port accesses the memory, and cautions users need to be aware of.

8.1.1 Write Port

When users start writing data to the write port, 64 words (default setting) of data will be written to the physical memory at once only if the internal buffer of the write port reaches 64 words. In another words, the write port will stop writing any data to the physical memory if the data remained in the buffer is less than 64 words. Hence a flush command is required to write the data remained in the internal buffer of the write port to the physical memory, unless the rest is not important. Figure 8.2 shows initial workflow of the write port.

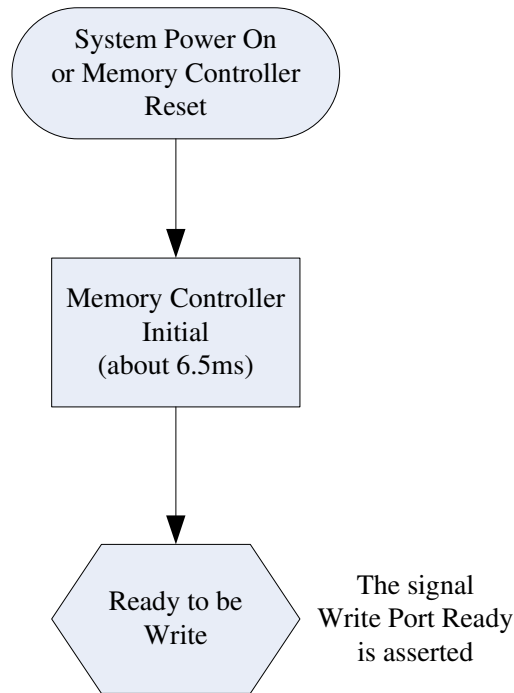


Figure 8.2 Initial workflow of the write port

8.1.2 Read Port

When the memory controller is ready to operate, the read port will start accessing 64 words (default setting) of data immediately from the physical memory to the internal buffer of its own. The signal port ready will be driven by the read port, which indicates users can start reading the data. However, such process will be triggered whenever the memory controller is initialized and the read port is not in the reset state. Hence undesired data will be captured to the internal buffer of the read port at the time.

To prevent the situation from happening, users must reset the read port to make sure data retrieved is valid. When the read port is reset, the current address will also be reset to pre-defined starting address of the read port, instead of the beginning address of current internal buffer. The initial workflow of the read port is shown in figure 8.3. If the read port is not in reset state, it will read the data into the buffer immediately from physical memory after memory controller completes initial stage. This may cause that undesired data will be get to the internal buffer of the port, because the data of physical memory have not been written.

There are two methods that we recommend to solve this problem:

1. Keep the read port reset signal low until start reading data.
2. To trigger the reset of read port before the first time to read data.

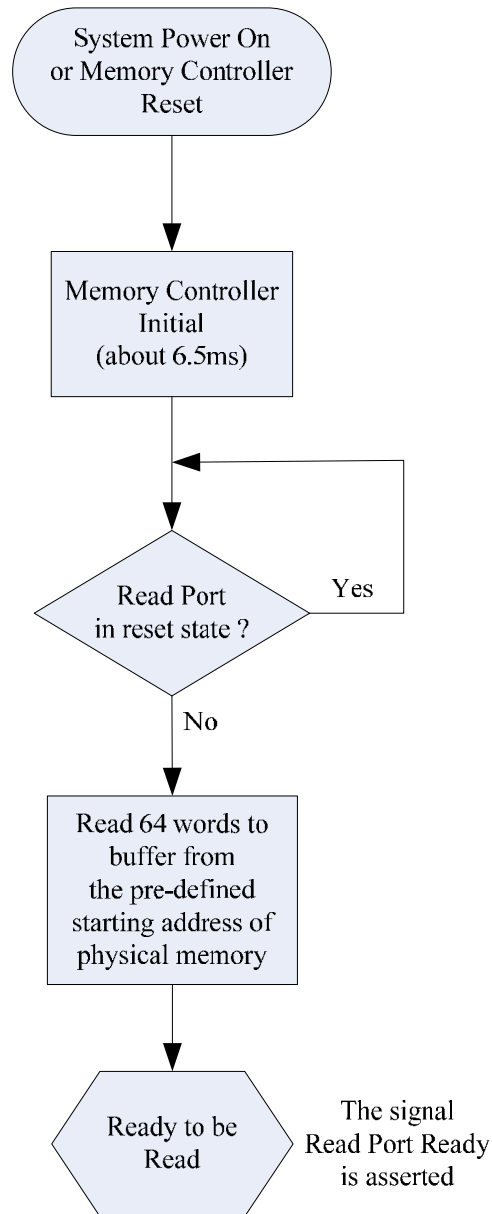


Figure 8.3 Initial workflow of the read port

8.2 Port Interface

One physical memory device could be replaced several memory blocks by using Terasic Multi-Port Memory Controller. Each memory block may have its own write and read port. These ports are similar to synchronous FIFO and must define the starting address and depth to configure a memory block that works on sequential mode. Figure 8.4 shows the memory arrangements.

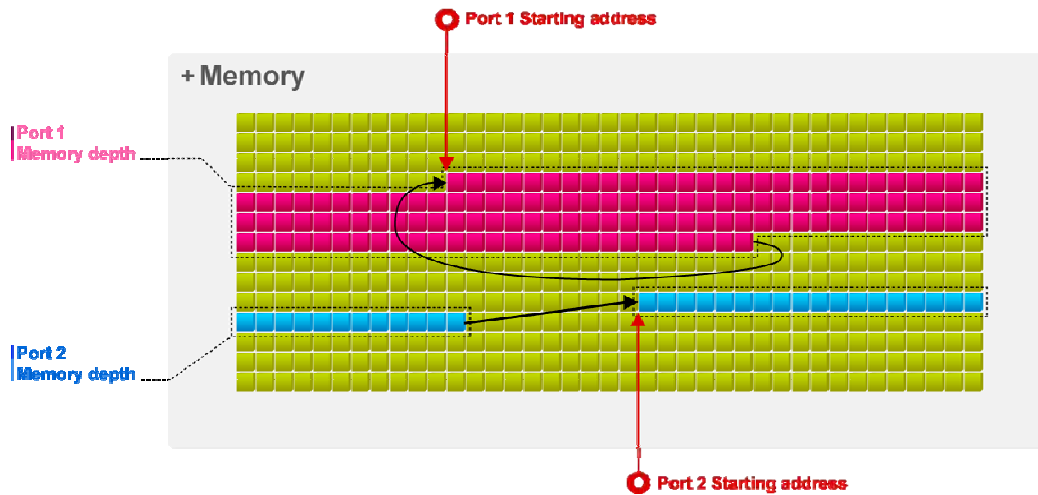


Figure 8.4 Memory arrangements

8.2.1 Simple Write Port

Figure 8.5 shows the writing waveform of simple write port. Each port has its own clock domain. It synchronizes data write transactions to FIFO of the port. When the number of data in the FIFO reaches a certain value, the write port will start writing the data from FIFO to the external memory which location is related to the writing address pointer. Figure 8.6 shows how to force a flush of the write port by asserting `iFLUSH_REQ`. During clock cycle 5, the signal `oFLUSH_BSY` is asserted to inform the local side that it is flushing the data which remain in the FIFO. When `oFULL` or `oFLUSH_BSY` is asserted or `oWRITE_PORT_READY` is inactive, the circuits of `iWRITE` are disabled.

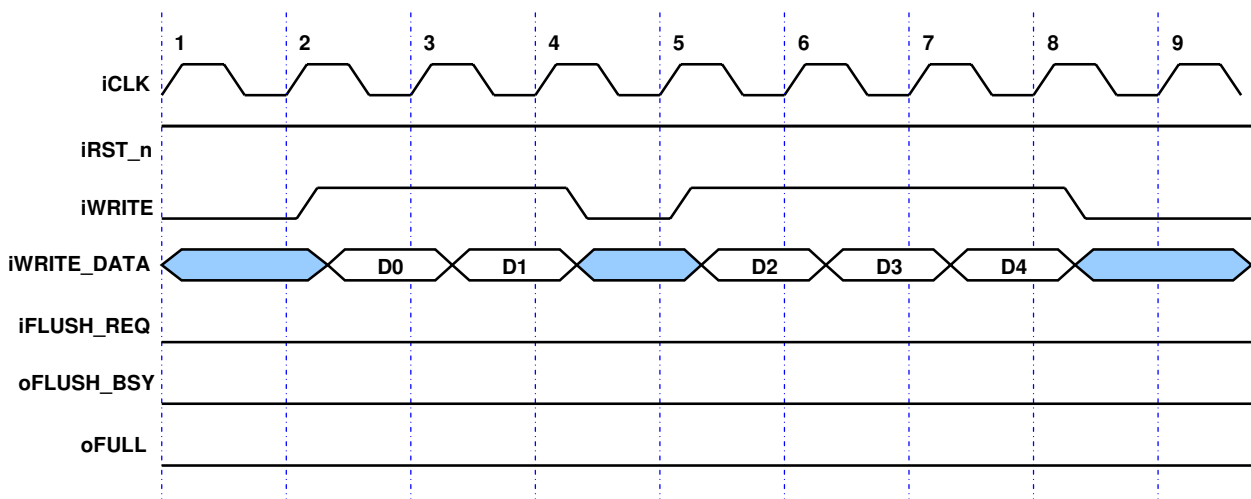


Figure 8.5 Write waveform of the simple write port

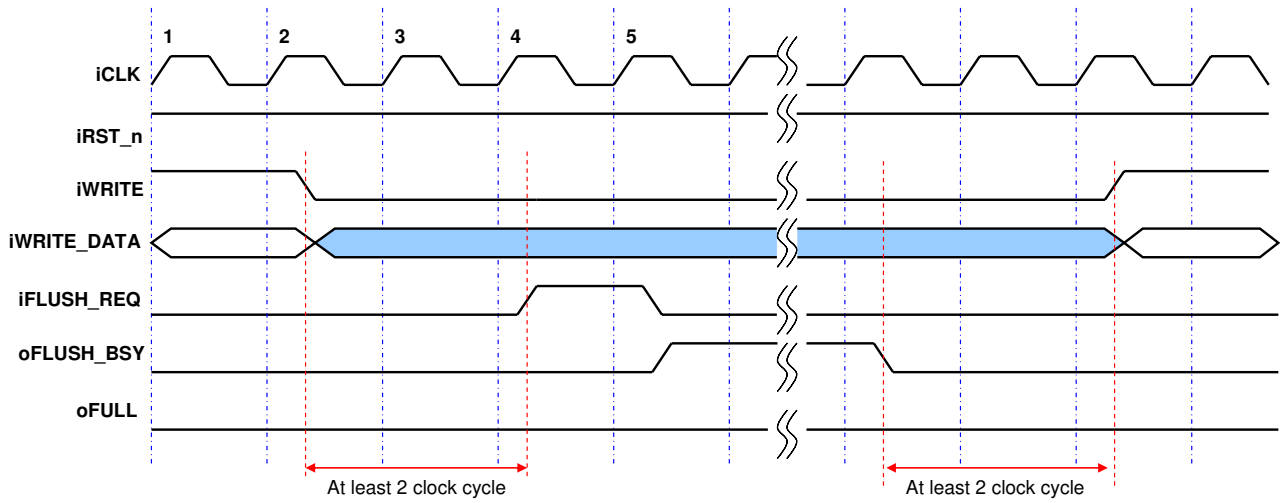


Figure 8.6 Flush waveform of the simple write port

8.2.2 Simple Read Port

Figure 8.7 shows read transactions of simple read port. The read signal operates as a read-acknowledge signal. Thus, the data bus outputs the first data word regardless of whether a read operation occurs. Figure 8.8 shows the port reset. When a reset operation occurs, the starting address of the port will be reloaded and oPORT_READY signal will be de-asserted to indicate that it is not enough data word to be read. While oPORT_READY is asserted, data of the read port can be read.

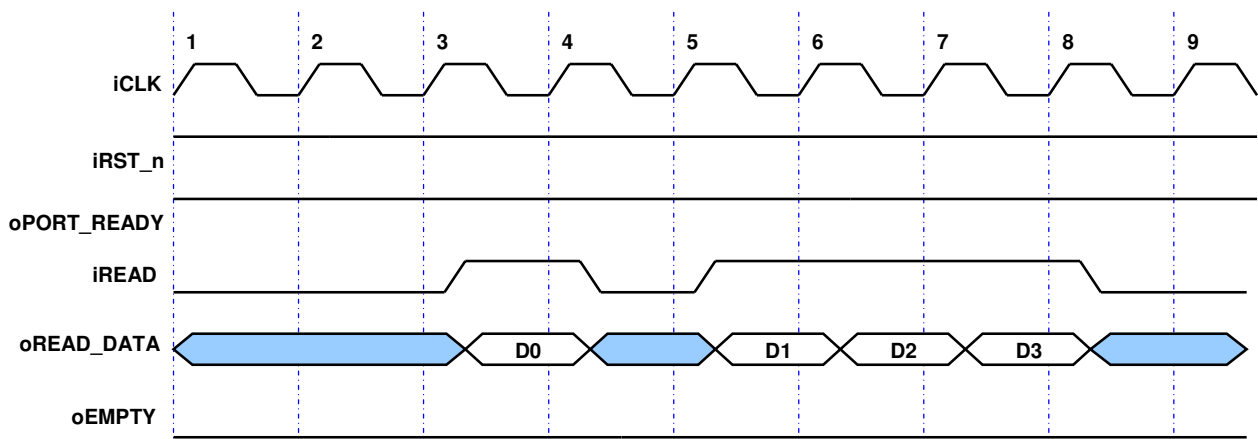


Figure 8.7 Read transfer of simple read port

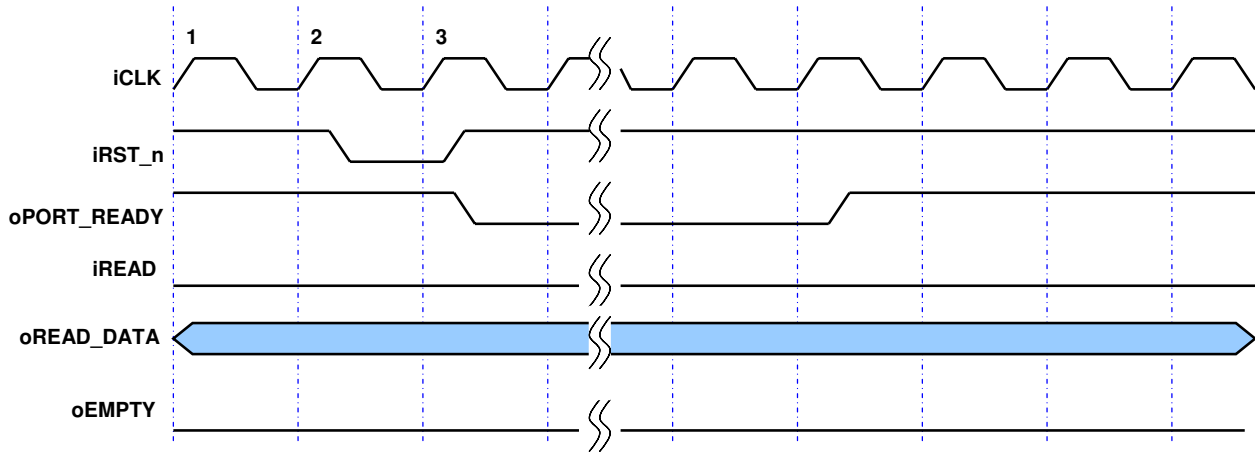


Figure 8.8 Reset Operation of simple read port

8.2.3 Enhanced port

Figure 8.9 shows write transfer of enhanced write port. The differences between the enhanced and simple port are that starting address and port depth of the enhanced port could be modified immediately. Figure 8.10 shows how to reload the starting address and port depth of the enhanced port. When the signal `iRST_n` is asserted, the data in the FIFO will be clear. Furthermore, the starting address and port depth signals will be reloaded into the registers of this port. Figure 8.11 shows the parameters reloading waveform of enhanced read port.

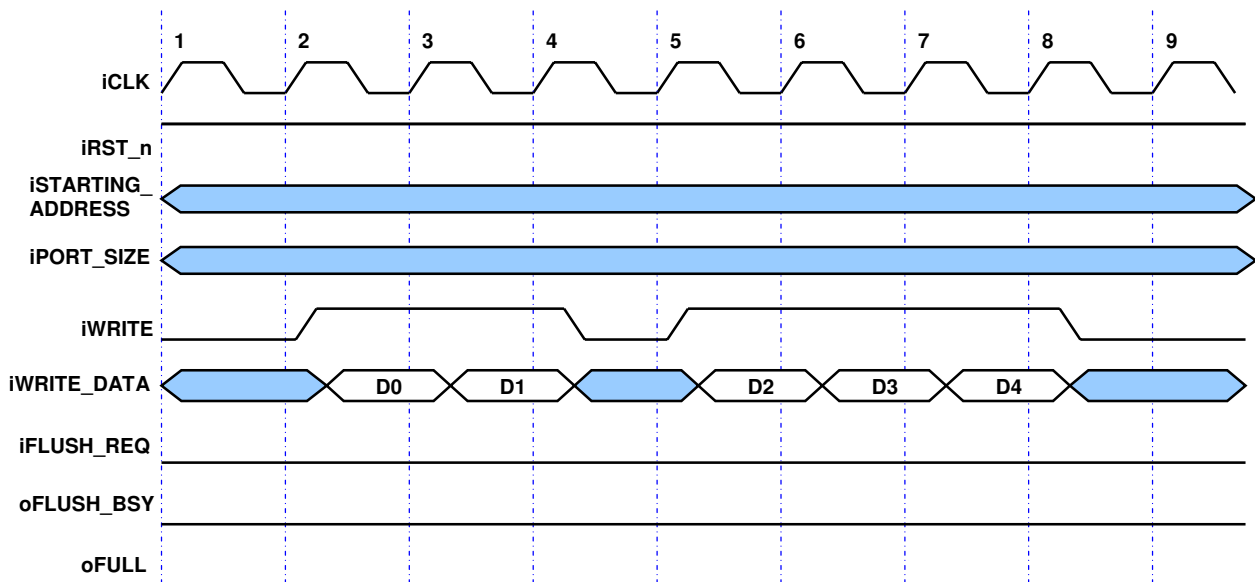


Figure 8.9 Write transfer of enhanced write port

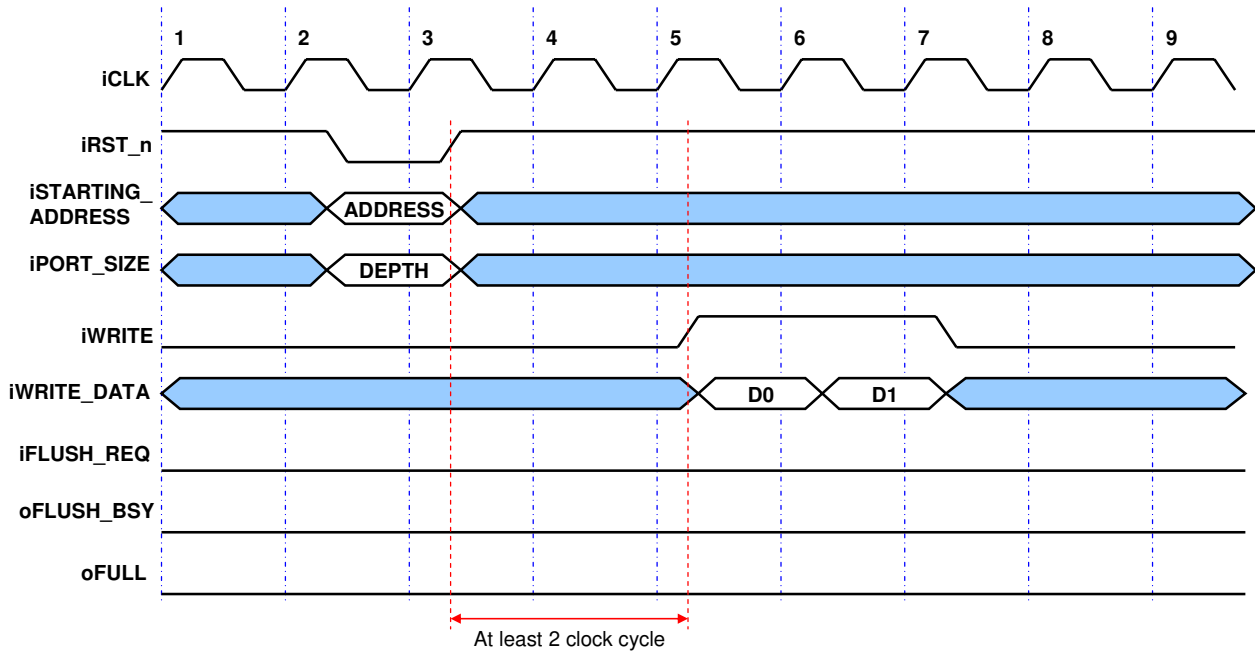


Figure 8.10 Parameter reloading of enhanced write port

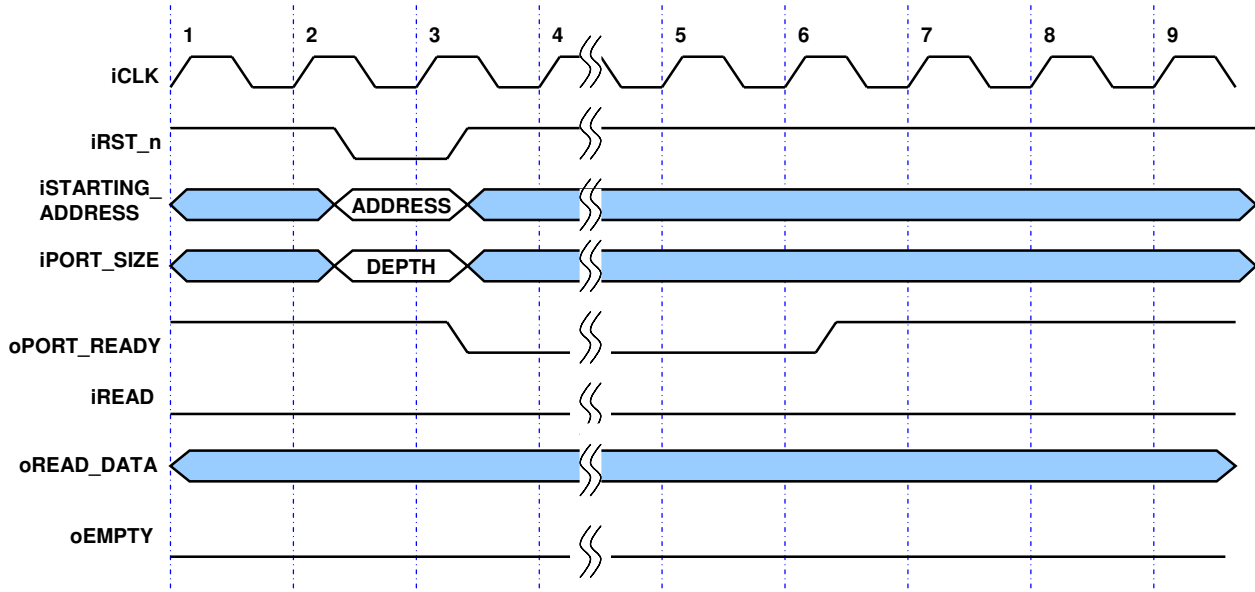


Figure 8.11 Parameter reloading of enhanced read port

Chapter 9

PCI Local Interface

This section describes how to directly communicate with PCI Bus and trigger a PCI interrupt.

9.1 PCI Local Write/Read Interface

The PCI local interface could be distributed several read and write local interface by PCI System Builder, and . The write interface is familiar memory-like write interface which supports wait-state insertion. The read interface has a burst count signal that is used to indicate the number of transfers in each read, and read interface is not support wait-state. Figure 9.1 shows a 64-bit write transfer waveform of PCI local interface. The wait-state of each write transfer do not exceed 16 clock cycles because the time of wait-state affects data transmission performance of PCI interface. The 64-bit read transfer waveform of PCI local interface is shown on figure 9.2, and wait-state mode is invalid on read transfer. While the iM_SEL signal of the local interface isn't asserted high, read or write transmission of the local interface has to ignore.

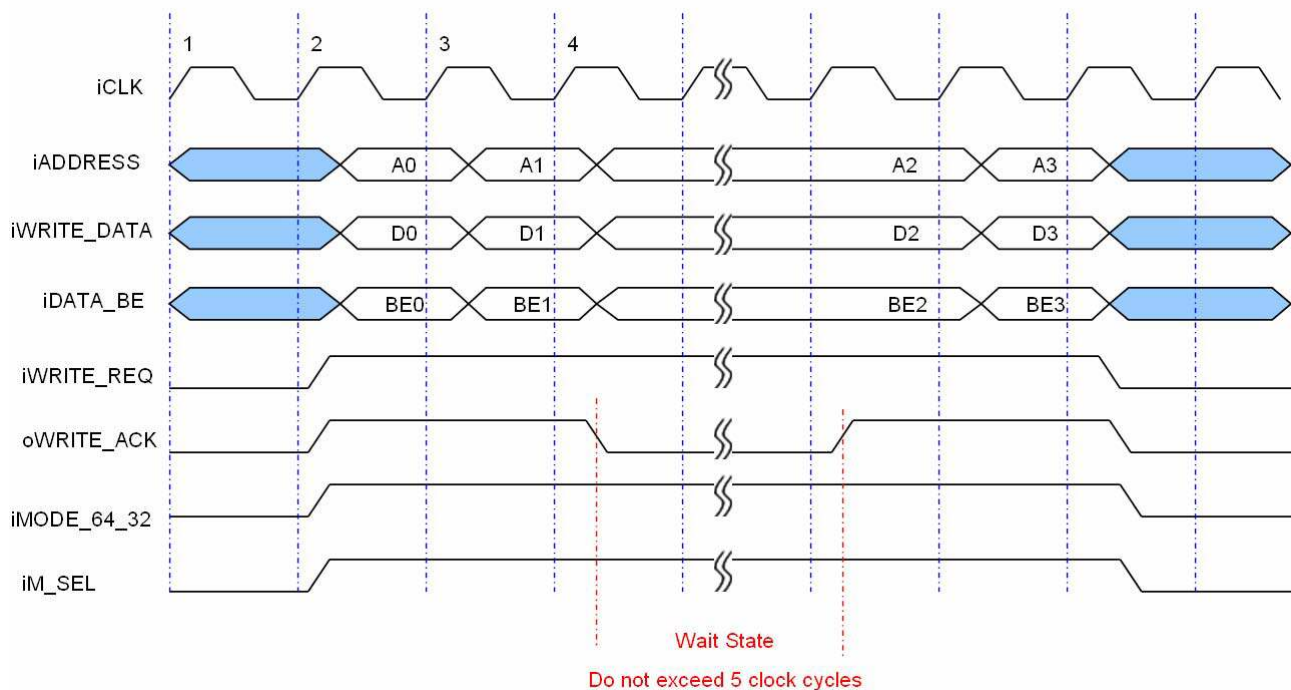


Figure 9.1 64-bit Write transfer waveform of PCI local interface

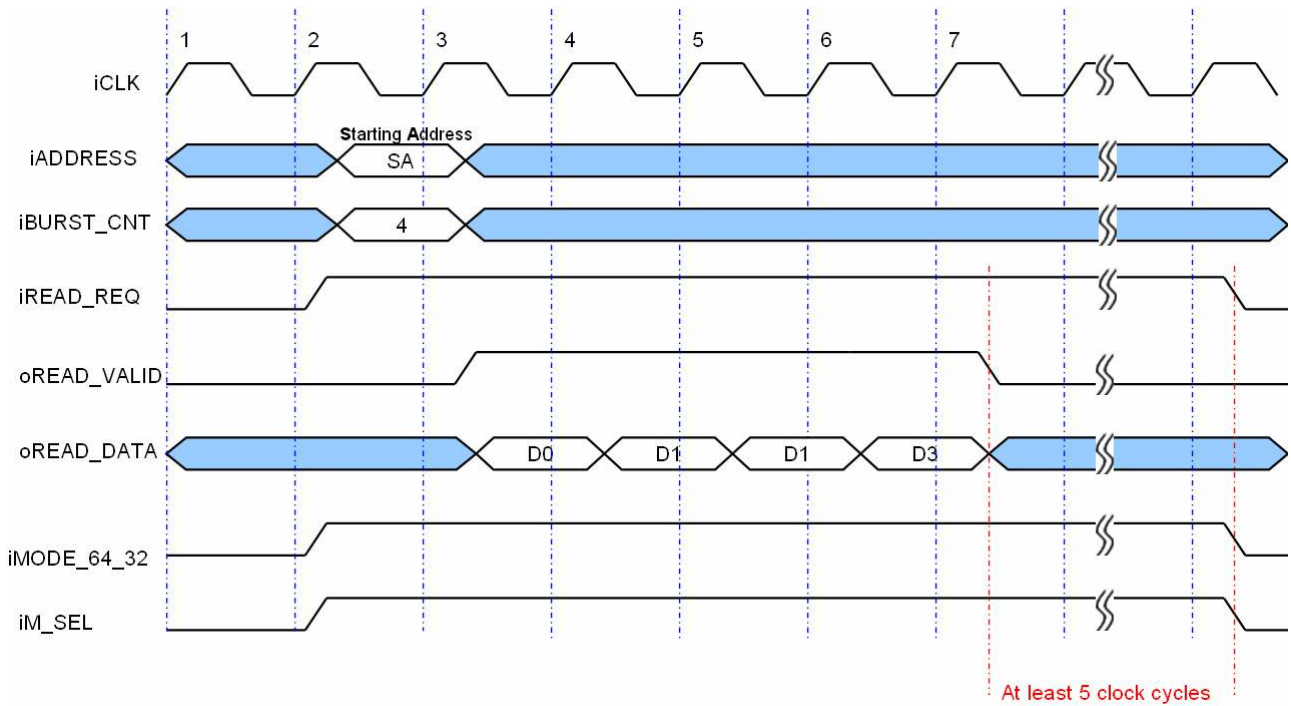


Figure 9.2 64-bit Read transfer waveform of PCI local interface

9.2 PCI Interrupt

For PCI interrupt, we provide a simple interface that allows user logic of local side to trigger the event. Figure 9.3 shows how to control the interrupt of PCI local interface. When the oCTRL_INT_REQ is active one clock cycle, the PCI interrupt will be triggered. Once PCI interrupt occurs, the software on PC side will clear the interrupt flag of PCI Bridge (it will assert iCTRL_INT_ACK one clock cycle to acknowledge) and execute the interrupt function.

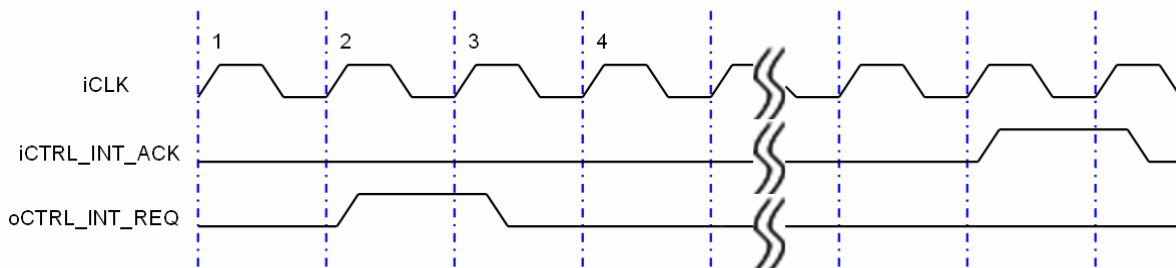


Figure 9.3 Interrupt trigger of PCI local interface

Table 9-1 PCI Local Interface's User Signals (PCI_Interface:Local_Interface)			
Name	Type	Polarity	Description
CLK	Input	-	Clock. The reference clock of PCI local interface.
ADDRESS[31..0]	Input	-	Address bus. The ADDRESS[31..0] is a byte-unit address bus.
WRITE_DATA[63..0]	Input	-	Write data bus. The width of WRITE_DATA bus is depending on PCI Bus data width (oMODE_64_32).
DATA_BE[7..0]	Input	-	Data bytes enable. If the bit on DATA_BE bus is asserted high, the byte data of WRITE_DATA is enabled.
WRITE_REQ	Input	High	Write request. The WRITE_REQ signal is an output from PCI bridge that indicates the beginning and duration of a write operation.
WRITE_ACK	Output	High	Write acknowledge. The WRITE_ACK is a user logic output, indicates the user logic of local side is accepting data.
READ_REQ	Input	High	Write request. The READ_REQ signal is an output from PCI bridge that indicates the beginning and duration of a read operation.
READ_VALID	Input	High	Read data valid.
READ_DATA[63..0]	Output	-	Read data bus. The width of READ_DATA bus is depending on PCI Bus data width (oMODE_64_32).
BURST_CNT[31..0]	Input	-	Burst count. Only during read transfer, the BURST_CNT bus is valid. It indicates the number of data will be transferred.
MODE_64_32	Input	-	Data width mode. The MODE_64_32 signal indicates the width of data bus on PCI local interface.
M_SEL	Input	High	Interface Select. When the M_SEL signal is asserted, the PCI bus transmission was decoded to transfer with this user local interface this time.
MEM_REG_SEL	Input	-	Memory/Register mapping. When the signal MEM_REG_SEL is asserted high, the mapping of the transfer is memory. When it is assert low, the mapping of the transfer is register.
ACCESS_MODE	Input	-	Port/Memory access mode. When the signal ACCESS_MODE is asserted high, the mode of the transfer is port access. When it is assert low, the mode of the transfer is memory access.

Table 9-2 PCI Local Interrupt Signals (on User_Logic)			
Name	Type	Polarity	Description
iCLK	Input	-	Clock. The reference clock output of PCI local interface.
oCTRL_INT_REQ	Output	High	PCI interrupt request. When the signal oCTRL_INT_REQ is active one clock cycle by user of PCI local side, the PCI interrupt will be trigger.
iCTRL_INT_ACK	Input	High	PCI interrupt acknowledge. The iCTRL_INT_ACK is an output from PCI bridge that indicates the interrupt acknowledge of host PC.

Appendix A

Programming the Serial Configuration Device

This appendix describes how to program the serial configuration device with Serial Flash Loader (SFL) function via the JTAG interface. User can program serial configuration devices with a JTAG indirect configuration (.jic) file. To generate JIC programming files with the Quartus II software, users need to generate a user-specified SRAM object file (.sof), which is the input file first. Next, users need to convert the SOF to a JIC file. To convert a SOF to a JIC file in Quartus II software, follow these steps:

■ Convert SOF to JIC

1. Choose “**Convert Programming Files...**” under Quartus’s File menu.
2. In the **Programming file type** pull-down menu, select the item “**JTAG Indirect Configuration File (.jic)**”.
3. In the pull-down menu “**Configuration device**”, select the targeted serial configuration device (Select EPCS64).
4. In the **File name** edit box, browse to the target directory and specify an output file name.
5. Select the “**SOF Data**” in the Input files to convert section, as showing in [Figure 0.1](#).
6. Click “**Add File...**” button. In the “**Select Input File**” dialog, select the SOF that you want to convert to a JIC file, and then click “**Open**”.
7. Select the “**Add Device**” in the Input files to convert section, as showing in [Figure 0.2](#).
8. Click **OK**. The Select Devices page displays.

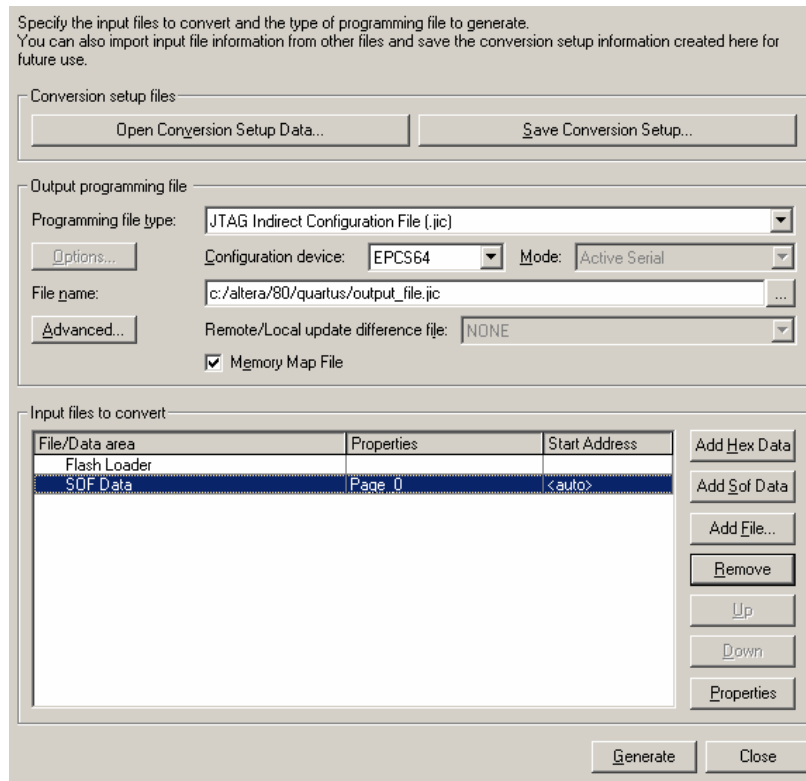


Figure 0.1. Convert Programming Files Dialog Box

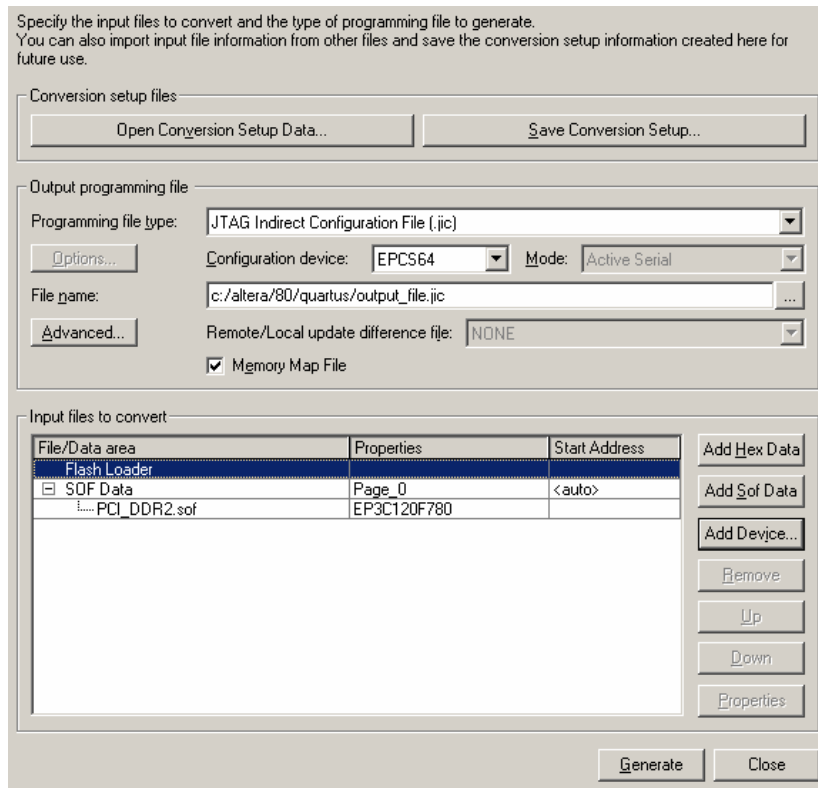


Figure 0.2. Highlight Flash Loader

9. Select the targeted FPGA that you are using to program the serial configuration device. See [Figure 0.3](#).

10. Click OK. The **Convert Programming Files** page displays. See [Figure 0.4](#).
11. Click **Generate**.

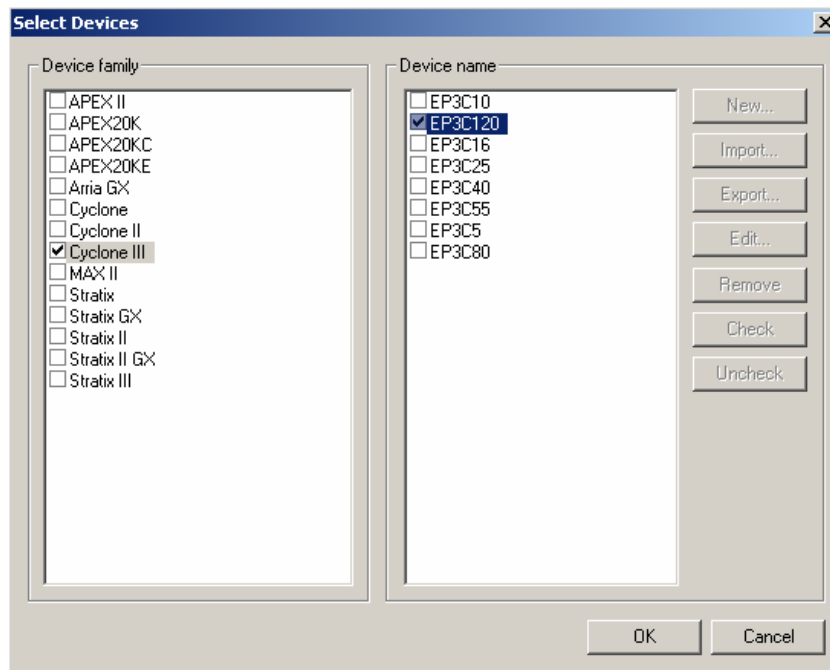


Figure 0.3. Select Devices Page

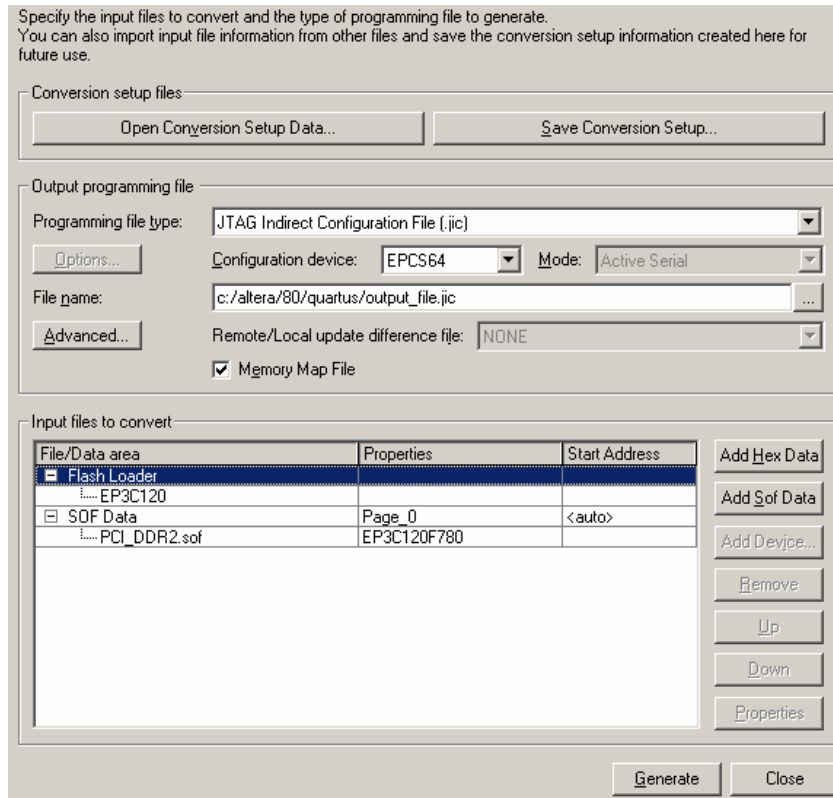


Figure 0.4. Convert Programming Files Page

■ Write JIC File into Serial Configuration Device

To program the serial configuration device with the JIC file that you just created, add the file to the Quartus II Programmer window and follow the steps:

1. When the SOF-to-JIC file conversion is complete, add the JIC file to the Quartus II Programmer window:
 - i. Choose **Programmer** (Tools menu). The **Chain1.cdf** window displays.
 - ii. Click **Add File**. From the **Select Programming File** page, browse to the JIC file.
 - iii. Click **Open**.
2. Program the serial configuration device by checking the corresponding **Program/Configure** box, a Factory default SFL image will be load (See [Figure 0.5](#)).

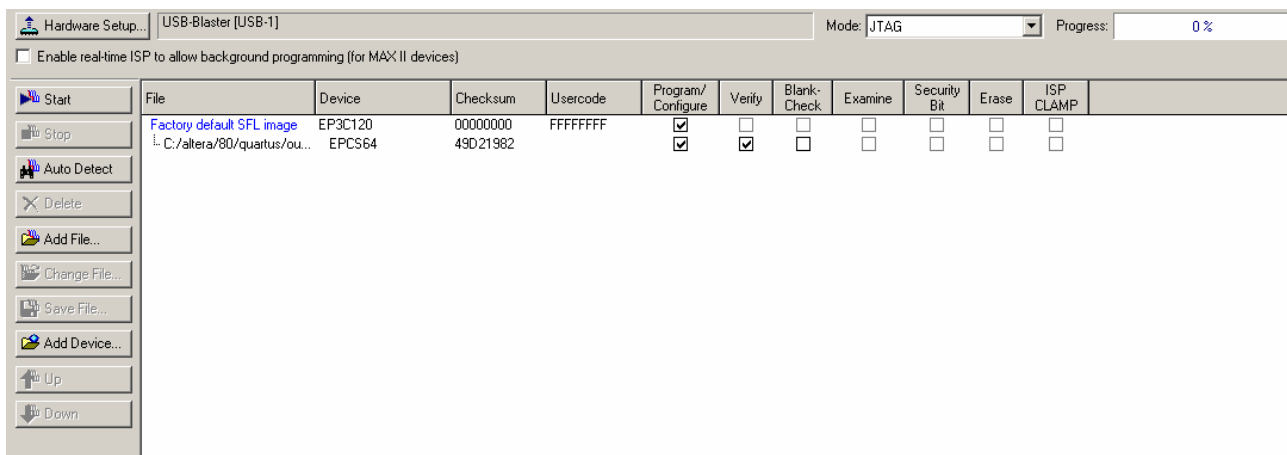


Figure 0.5. Quartus II programmer window with one JIC file

3. Click **Start** to program serial configuration device.

■ Erase the Serial Configuration Device

To erase the existed file in the serial configuration device, follow the steps listed below:

1. Choose **Programmer** (Tools menu). The **Chain1.cdf** window displays.
2. Click **Add File**. From the Select Programming File page, browse to a JIC file.
3. Click **Open**.
4. Erase the serial configuration device by checking the corresponding **Erase** box, a Factory default SFL image will be load (See [Figure 0.6](#)).

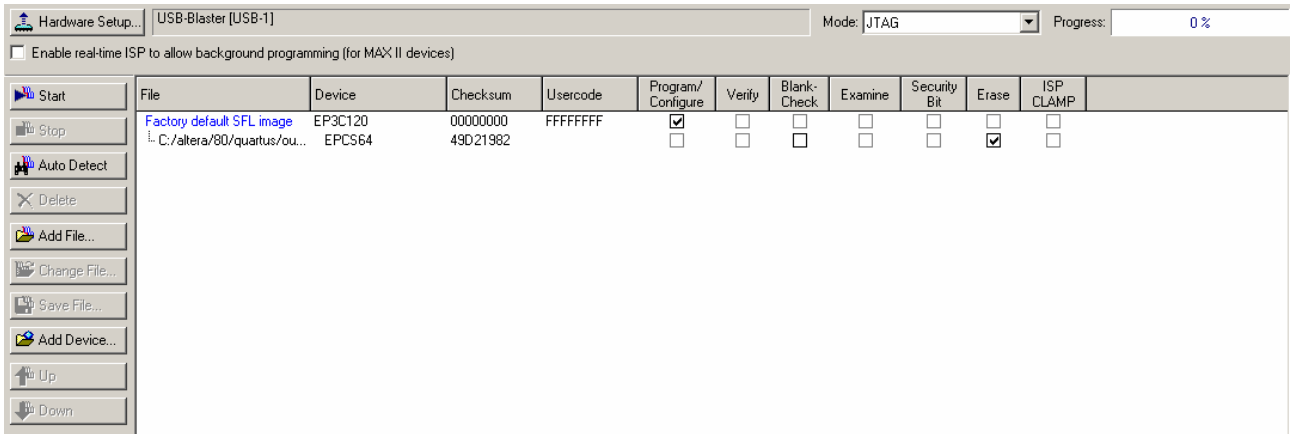


Figure 0.6 Erasing setting in Quartus II programmer window

5. Click **Start** to erase the serial configuration device.