PCI BUS CONNECTS MIZZOURISC TO PC

A Thesis

presented to

the Faculty of the Graduate School

at the University of Missouri-Columbia

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

NAN ZHU

Dr. Harry Tyrer, Thesis Supervisor

MAY 2011

The undersigned, appointed by the dean of the Graduate School, have examined the thesis entitled

PCI BUS CONNECTS MIZZOURISC TO PC

Presented by Nan Zhu

A candidate for the degree of Master of Science,

And hereby certify that, in their opinion, it is worthy of acceptance.

Professor Harry Tyrer

Professor Guilherme DeSouza

Professor Yunxin Zhao

ACKNOWLEDGEMENTS

First and foremost, I would like to show my deepest gratitude to my supervisor, Dr. Harry Tyrer, a respectable, responsible and resourceful scholar, who has provided me with valuable guidance in every stage of the writing of this thesis. Without his enlightening instruction, impressive kindness and patience, I could not have completed my thesis. His keen and vigorous academic observation enlightens me not only in this thesis but also in my future study.

I would also like to thank all my professors who have helped me to develop the fundamental and essential academic competence. My sincere appreciation also goes to the students, who have helped me throughout my graduate study.

Last but not least, I sincerely thank my family and all my friends in China, especially Shaohua and Shuai, two understanding ladies. Their constant support and faith helped me successfully complete my graduate studies in the United States.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF EQUATIONS

ABSTRACT

This paper covers the design of a PCI transmission controller to make a 32-bit microprocessor MizzouRISC be a co-processor for a PC. The co-processor communicates with the PC through the PCI bus, and works as a target device.

We implemented the VHDL firmware consisting of PCI transmission module, memory, and an existing VHDL implementation of MizzouRISC on a PCI FPGA board. On the PC, we developed a driver for the PCI FPGA board, and an application program that acts as a controller to upload and download data from the board.

The application program averages the rgb components of each pixel in a 24-bit bmp file to get the gray-level image. It also uploads the image to the PCI board, waits for the interrupt, and reads the processed fragment from the PCI board. The VHDL memory is limited, so the image was divided into several image fragments. After using the image to test the entire system, we can find the relationship of the fragment size and the processing time.

We used VHDL to create the memory and a slow clock for MizzouRISC timing. This compromise was due to a limitation on the FPGA of the PCI board. Future design of a new board for MizzouRISC should include a faster clock for the processor, a 32-bit SDRAM hardware module, and even a PS/2 port and a VGA port on the board.

CHAPTER 1 : INTRODUCTION

MizzouRISC is a microprocessor designed by Dr. Tyrer using the RISC (Reduced Instruction Set Computing) philosophy [1]. Nash implemented MizzouRISC on an FPGA [2], and we describe here a PCI transmission module, also implemented on an FPGA, to connect MizzouRISC to a PC through the PCI bus.

FPGA (Field Programmable Gate Array) devices are particularly suitable for the implementation of MizzouRISC. FPGAs can provide processing power, to implement a microprocessor [3]. In many applications it would be convenient to have an FPGA implemented microprocessor, and this one chip solution can simplify the design of the board, and reduce the power consumption and cost.

1.1 Motivation

The MizzouRISC implemented by Nash in VHDL integrated keyboard input and VGA output for a standalone system. However, this implementation requires resource of the FPGA not easily available. Developing a PCI communication module, MizzouRISC can communicate with the PC through the PCI bus, and can also utilize the input and output devices connected to the PC.

In addition, connecting MizzouRISC on the PC through the PCI bus can make the PC use the processing power of the microprocessor MizzouRISC. In this case, MizzouRISC will be a co-processor of the PC, and lighten the burden of the CPU on the PC. Additionally, Nash’s work at multiprocessor with MizzouRISC (4 core processors) can be easily accommodated.

Finally, MizzouRISC used as an educational tool helps students understand how the processors work. Connecting the MizzouRISC on the PC can facilitate students control MizzouRISC, and more easily use a small and simple set of instructions to control the microprocessor [4].

1.2 Approach

We developed the PCI communication module and memory with VHDL, combined them with the MizzouRISC developed by Nash, and implemented them on the FPGA, which is on the RaggedStone1 board. The RaggedStone1 board maintains a PCI interface gold finger, which is connected to the FPGA through several bidirectional buffers. After we implemented the design on the FPGA and inserted it on the mother board of the PC, the board can be worked as a target PCI device. On the PC, we developed a driver for the PCI device, and a program acts as a controller to operate the system state machine. We found it convenient to put on board memory and make use of it. However, our actual implementation used memory generated by VHDL, as shown in the result (chapter 5).

1.3 Outcome

The test is to load an image on to the co-processor board, have MizzouRISC modify the image and return to the PC. We have successfully tested the hardware with the VHDL code programmed into the FPGA. We also test the PC required software, a driver for the operating system and an application program for controlling the upload and download. Our test program for MizzouRISC inverts the gray-level of the image derived from a 24-bit bmp file. The program averaged the color, uploaded to MizzouRISC, inverted the image, and returned the image to the PC. The VHDL memory is limited, so the image was divided into several image fragments. We accumulate the fragment processing time to get the image processing time.

CHAPTER 2 : BACKGROUND

2.1 MizzouRISC

MizzouRISC is a processor designed using Reduced Instruction Set Computer (RISC) philosophy [1]. MizzouRISC has 12 simple but fundamental instructions, which need a small numbers of cycles. A fast clock and pipelining provide high speed.

MizzouRISC has been implemented as a soft processor using VHDL [2]. Nash used VHDL as the hardware description language and implemented the code on a Spartan3 Starter Board manufactured by Diligent.

Figure 2.1 Interface of MizzouRISC

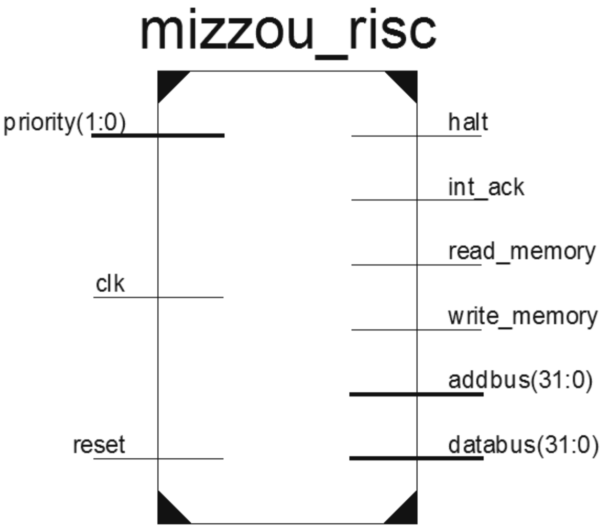


Figure 2.1 shows the pin connections of the 32-bit RISC processor following Nash's design. (We added `int_ack` as discussed later). Table 2.1 describes each of the pins, and we discuss 4 of them: `halt`, `int_ack`, `addbus` and `databus`.

Table 2.1 Pins of the MizzouRISC

Name	input/output	Function
clk	Input	Clock signal for the MizzouRISC
reset	Input	Reset signal for the MizzouRISC
priority(1:0)	Input	Interrupt signal for the MizzouRISC
halt	Output	Show halt state
int_ack	Output	Acknowledge signal for the interrupt
read_memory	Output	Memory read signal
write_memory	Output	Memory write signal
addbus(31:0)	Output	Address bus connect to the memory or I/O
databus(31:0)	Inout	Data bus connect to the memory or I/O

The `halt` signal represents the processor execute an instruction which opcode is 0, and causes the processor to stop by going into a halt state. The `int_ack` acknowledges to an outside device that the processor has received the interrupt signal, and begun to process the interrupt. The `addbus` and `databus` are 32-bits address bus and 32-bits data bus, respectively. Through these, MizzouRISC can get instructions from the ROM, read and write RAM, get data from keyboard and output data on the VGA screen.

2.2 PCI BUS

PCI stands for Peripheral Component Interconnect, created by Intel Corporation [5], is a computer bus for connecting hardware devices on a PC's mother board. There are several versions of PCI bus in normal PCs which use different clock frequencies, bus widths and voltages. In our project, the frequency of the PCI clock is 33 MHz, bus widths are 32-bits, and signals are 5-volt. We utilize 50 pins from the total 120 pins on the interface to implement the target read and target write for the

RaggedStone1 FGPA board.

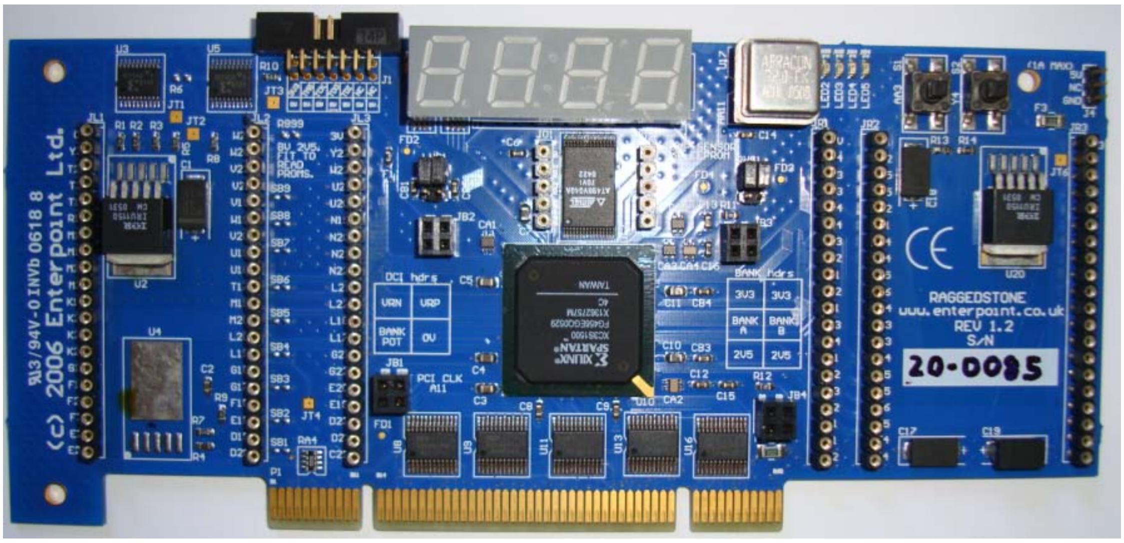
The devices connected on the PCI bus, can be divided into initiator device and target device. In our project, the PC will act as an initiator device, and the PCI board will act as a target device all the time. The initiator needs to request permission from the PCI bus arbiter on the mother board. When the initiator gets the permission of using the PCI BUS, it will send hand-shake signals to the target device. After sending the hand-shake signals, initiator needs to provide the address to the target device, and then read or write data through the PCI bus. However, the target device only needs to respond the hand-shake signals from the initiator device, and either provides or receives data as commanded by the initiator device.

PCI has three address spaces: memory space, I/O space, and configuration space [6]. The 32-bits PCI memory space, which is 4GB in size, can provide burst transactions. The I/O address is also 4GB and compatible with the Intel x86 architecture's I/O ports. PCI configuration space is divided into separate configuration address space for each functional device contained within a physical device. In our project, the PCI device (the board RaggedStone1) just maintains one PCI function, and does not use the I/O address space. We need only concern ourselves of memory space and the 64 bytes for the PCI device's configuration header region.

2.3 RaggedStone1: Spartan-3 PCI Development Board

The PCI FPGA development board we used is RaggedStone1, manufactured by Enterpoint LTD. [7]

Figure 2.2 Photo of the RaggedStone1



As the figure 2.2 shows, the element in the middle of the board is a Spartan-3 FPGA (XC3S1500) produced by Xilinx. In addition, the board integrates a 32-bit, 33MHz and 3.3/5V PCI interface, 2 flash memory, 2 push button switches, 4 LEDs and 4 7-segment LEDs. We connect the PC and the RaggedStone1 board through the parallel programming cable and JTAG, in order to write the firmware from the PC to the flash memory on the board. In addition, we used the software Xilinx ISE Design Suite 12.2 Web Pack to compile the VHDL code, Route and Place the firmware, and program the firmware to the FPGA on the RaggedStone1 board.

On the top-left corner of the RaggedStone1 board, the two chips XC3F04 and XC3F02 are the flash devices, which store the configuration VHDL firmware for the

FPGA, (e.g. MizzouRISC). The firmware produced for the FPGA XC3S1500 extends beyond the size capability of the flash device XC3F04, which requires separating the firmware into two parts and placing each into two flash devices. The advantage of saving the firmware in the flash device is that the firmware needs not be programmed to the FPGA every time, and the firmware will be loaded to the FPGA automatically once the board is powered on.

2.4 Background for Test Program

2.4.1 Inverse Gray Scale

The test program for the system is to invert the gray-level image. The pixels in the gray-level image are integers from 0 to 255 to show the color from black to white, respectively. To invert the image, get the value of the pixel from original image, subtract from 255, and plot the resulting pixel in the output image. For example, if the value of the pixel in original image is 50, the value of the corresponding pixel in the output file will be 205.

2.4.2 Counter in C Program

For PCI bus transmission, we transferred 16K byte units of data each time. We want to get the processing time of PCI bus transmission for each unit, but the transmission time is smaller than 1 millisecond in each cycle. The APIs time () or gettickcount () will generate large error, instead we need to utilize the APIs

QueryPerformanceFrequency () [8] and QueryPerformanceCounter () [9] to get a high-resolution performance counter value. QueryPerformanceFrequency () is the API which will retrieve the frequency of the high-resolution performance counter. This API needs to be supported by the installed hardware.

QueryPerformanceCounter () is the API which will retrieve the current value of the high-resolution performance counter.

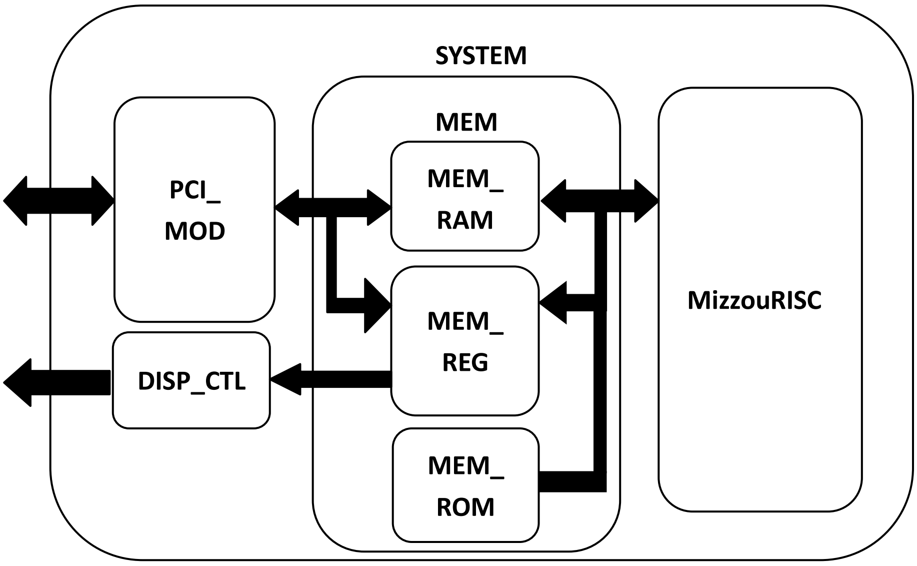
Equation 2.1 Execution Time

$$\mathrm{T_s} = \frac{(\mathrm{T_b} - \mathrm{T_e}) \times 1000}{\mathrm{F}}$$

Using the two APIs described above, C program can get the value of the counter before the execution, $\mathrm{T_b}$, and the value of the counter at the end of the execution, $\mathrm{T_e}$. In addition, the frequency F can also be obtained by the API QueryPerformanceFrequency () and the value of F will not be changed while the system is running. Using the equation described in the Equation 2.1, we can get the execution time, and result is in millisecond.

CHAPTER 3 : FIRMWARE CONNECTING PCI TO MIZZOURISC

Figure 3.1 SYSTEM Blockdiagram

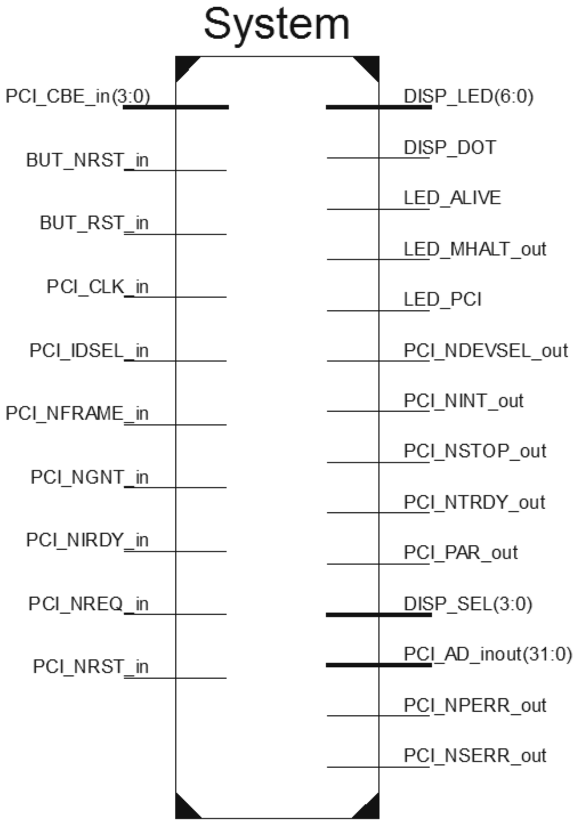


The FPGA on the RaggedStone1 board contains all the functionality components to connecting MizzouRISC to a PC through the PCI bus. The **SYSTEM** is the name of the top-level entity. Figure 3.1 shows the blockdiagram of the **SYSTEM**, which is composed of 4 submodule components: **MizzouRISC**, **MEM**, **PCI_MOD** and **DISP_CTL**. **MizzouRISC** is the soft processor developed by Nash [2], which is used to process the data according to the test program saved in the ROM. We developed the following modules as VHDL constructs. **PCI_MOD,** PCI transmission module, controls the transmission between the PC and the board through the PCI bus. It makes the PCI board act as a target device, and the functionality is to carry out target read and target write. **DISP_CTL**, controls the 7-seg LEDs, is used to display the value of the label register located at **MEM** address x3FFC$_H$. **MEM** is again a VHDL construct that contains the memory components, composed of **MEM_RAM**, **MEM_REG** and **MEM_ROM**. Both **MizzouRISC** and **PCI_MOD** can access **MEM_RAM** and **MEM_REG**. However, for now **MEM_ROM** contains the **MizzouRISC** test program, so that only **MizzouRISC** can read the data from the **MEM_ROM**. The test program can be written to the **MEM_ROM** when the FPGA is programmed by the software ISE Project Navigator.

3.1 SYSTEM: Top-Level Entity of the FPGA

SYSTEM is the name of the top module for the FPGA which is saved in the file System.vhd. Figure 3.2 shows the interface of this module.

Figure 3.2 Interface of the System



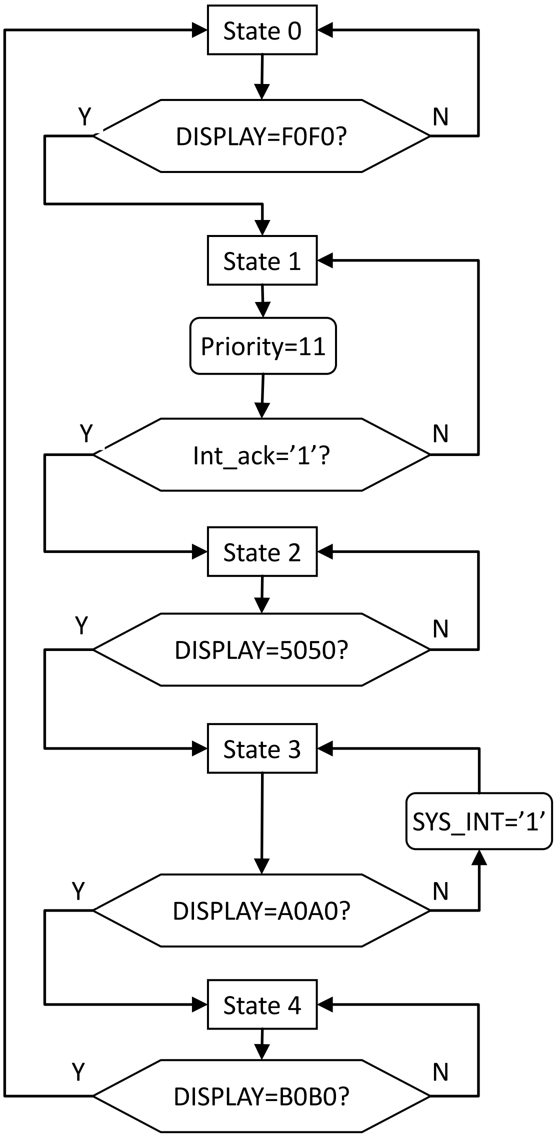
All the signals on the figure are mapped to the real pins on the FPGA according to the Implementation Constraints File System.ucf, and the signals on the left side are input, on the other hand, those on the right side are output or inout signals.

Table 3.1 Signals Mapped to the Pins on the FPGA

Group	Signal Name	Direction	Function
PCI	PCI_CBE_in(4)	Input	Command or Byte Enable
	PCI_CLK_in	Input	Clock from the PCI bus, about 33 MHz
	PCI_IDSEL_in	Input	Control signal for configuration cycle
	PCI_NFRAME_in	Input	Control signal for data transaction
	PCI_NGNT_in	Input	Guarantee signal for PCI bus
	PCI_NIRDY_in	Input	Master ready signal
	PCI_NREQ_in	Input	Request signal for PCI bus
	PCI_NRST_in	Input	Reset signal from PCI bus
	PCI_NDEVSEL_out	Output	Target Device selected signal
	PCI_NINT_out	Output	Interrupt signal
	PCI_NSTOP_out	Output	Target stop signal
	PCI_NTRDY_out	Output	Target ready signal
	PCI_PAR_out	Output	Parity signal
	PCI_NPERR_out	Output	Parity error report signal
	PCI_NSERR_out	Output	System error report signal
	PCI_AD_inout(32)	Inout	Time-multiplexed Address/Data bus
DISP	DISP_LED(7)	Output	Display of 7-segments LEDs
	DISP_DOT	Output	Dot on the 7-segments LEDs
	DISP_SEL(4)	Output	Selection signal of the 7-segments LEDs
BUT	BUT_NRST_in	Input	Reset# signal for PCI module
	BUT_RST_in	Input	Reset signal for MizzouRISC
LED	LED_ALIVE	Output	Lighted when system is powered on
	LED_MHALT_out	Output	Lighted when MizzouRISC in a halt state
	LED_PCI	Output	Lighted when PCI Device is worked

The pins can be divided to 4 groups: PCI, DISP, BUT and LED, they are connected to the PCI gold finger, 7-segments LEDs, buttons and LEDs on the board, respectively. We use 50 pins of the total 120 pins on the PCI gold finger, since we just want to implement 32-bit target read and target write functions. Now 7-segments LEDs, controlled by signals DISP_LED, DISP_DOT and DISP_SEL, displays the value of the label register. DISP_LED shows the hexadecimal digit from 0-9 and A-F, DISP_DOT puts out the decimal point, and DISP_SEL selects one of the four LED alternatively in each clock cycle. Buttons are used to provide two reset signals, BUT_NRST_in is for the PCI transmission module and BUT_RST_in is for the **MizzouRISC**. The LED which is connected to the LED_ALIVE will be lighted when the board is powered on; the LED shows the value of the LED_MHALT_out will be lighted when **MizzouRISC** goes into a halt state; the LED for the LED_PCI will be lighted when the board is recognized as a PCI device by PC.

Figure 3.3 State Machine on the Entity SYSTEM



The **PCI_MOD** and **MizzouRISC** cannot control the **MEM** simultaneously; the top-level entity, **SYSTEM**, needs to use a state machine showed in Figure 3.3 to guarantee that only one module can access the memory. In state 0, the PC can send data to the **PCI_MOD** through the PCI bus, and **PCI_MOD** has the right to write data to the memory. The **SYSTEM** will check the label register at the address x3FFC$_H$; once the value of the label register is set to x0000F0F0$_H$, the state will change to 1. In the state 1, **SYSTEM** sends interrupt (set `priority` to "11") to the **MizzouRISC** and waits until it receives the acknowledge signal `int_ack` before the state is changed to 2. In the state 2, **MizzouRISC** will execute the test program in the **MEM_ROM** to process the data sent from the PC before it set the label register to x00005050$_H$. The **SYSTEM** will check the label register as well, but the state will change to 3 when the value of the label register is x00005050$_H$. In the state 3, the **SYSTEM** will set `SYS_INT` to '1' in order to generate the PCI interrupt to the PC, and the state will change to 4 when PC receives the Interrupt and write x0000A0A0$_H$ to the **PCI_MOD** at label register address through the PCI bus. In the state 4, PC will read the data from the **PCI_MOD** through PCI BUS and **PCI_MOD** has the right to access the memory (entity **MEM**). The **SYSTEM** will change the state back to 0 after PC has read all the data from the memory and write x0000B0B0H to the label register.

3.2 PCI_MOD: Entity to Control the PCI Transaction

PCI_MOD, saved in the file PCI_MOD.vhd, is the top-level module of the PCI transmission module which is used to implement data transaction between the PC and the board through PCI bus [10] [11]. **PCI_MOD** can make the RaggedStone1 board work as a target PCI device, and carry out target read and target write on the board.

Table 3.2 PCI Module File Layout

File Name	Module Function
PCI_MOD.vhd	Top-level module
PCI_CTL.vhd	Control the state machine of PCI read and write
PCI_ADD.vhd	Distinguish register process and memory process
PCI_DATA.vhd	Control data flow between external memory, PCI data bus and PCI registers
PCI_REG.vhd	PCI register read, PCI register write and maintain the register data
PCI_PAR.vhd	Generate the parity check signal

The functionality of the **PCI_MOD** is to integrate all the other modules generated in the files listed in the Table 3.2 into a PCI transmission module and make it usable for the up-level entity (**SYSTEM**).

Figure 3.4 Interface of the PCI_MOD

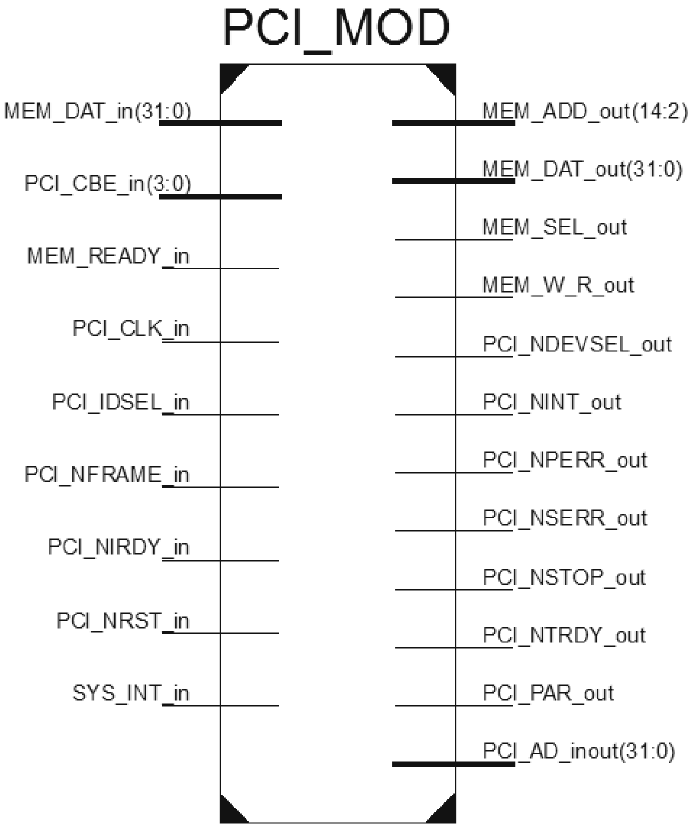


Figure 3.4 is the diagram of the VHDL entity element. The signals defined in **PCI_MOD** represent either input/output from the PCI bus or input/output from **MEM**. As Figure 3.1 shows, PCI bus pins connects to the top-level entity **SYSTEM**, and the **SYSTEM** will connects both of the signals from **MEM** and the signals from PCI bus to the **PCI_MOD**.

Table 3.3 Signals of PCI_MOD Connecting MEM

Signal Name	Direction	Function
MEM_DAT_in(32)	Input	32-bit data bus from MEM to PCI_MOD
MEM_READY_in	Input	Memory ready signal
MEM_ADD_out(13)	Output	Address bus of the memory
MEM_DAT_out(32)	Output	32-bit data bus from PCI_MOD to MEM
MEM_SEL_out	Output	Chip select signal of memory
MEM_W_R_out	Output	Write/Read signal for memory

The **PCI_MOD** module uses the signals list in the Table 3.3 to write the data to the **MEM** module, or to read the data from the **MEM** module. The address for now is 15 bits because the memory, including **MEM_RAM**, **MEM_ROM** and **MEM_REG**, is 32K byte, but the width of the address is flexible to fit for other size memory.

3.2.1 PCI_CTL: Read and Write Transfer State Machine

Figure 3.5 Interface of the PCI_CTL

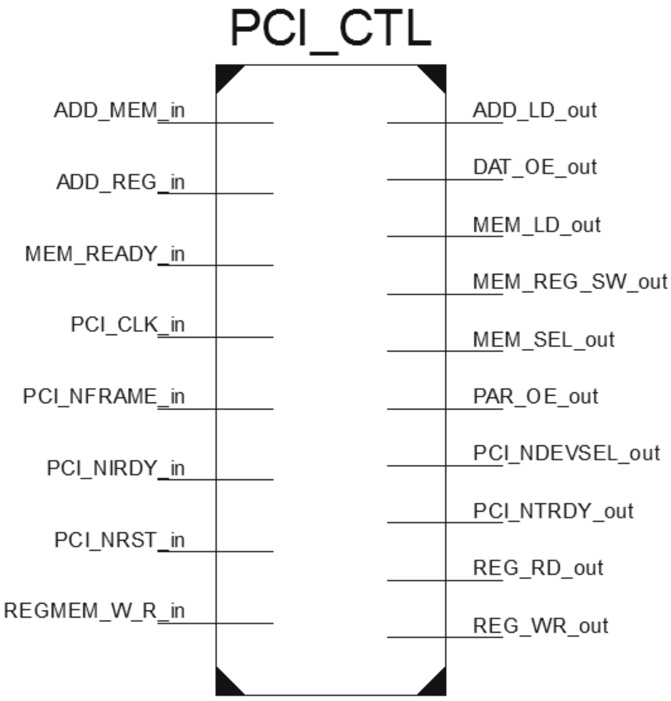


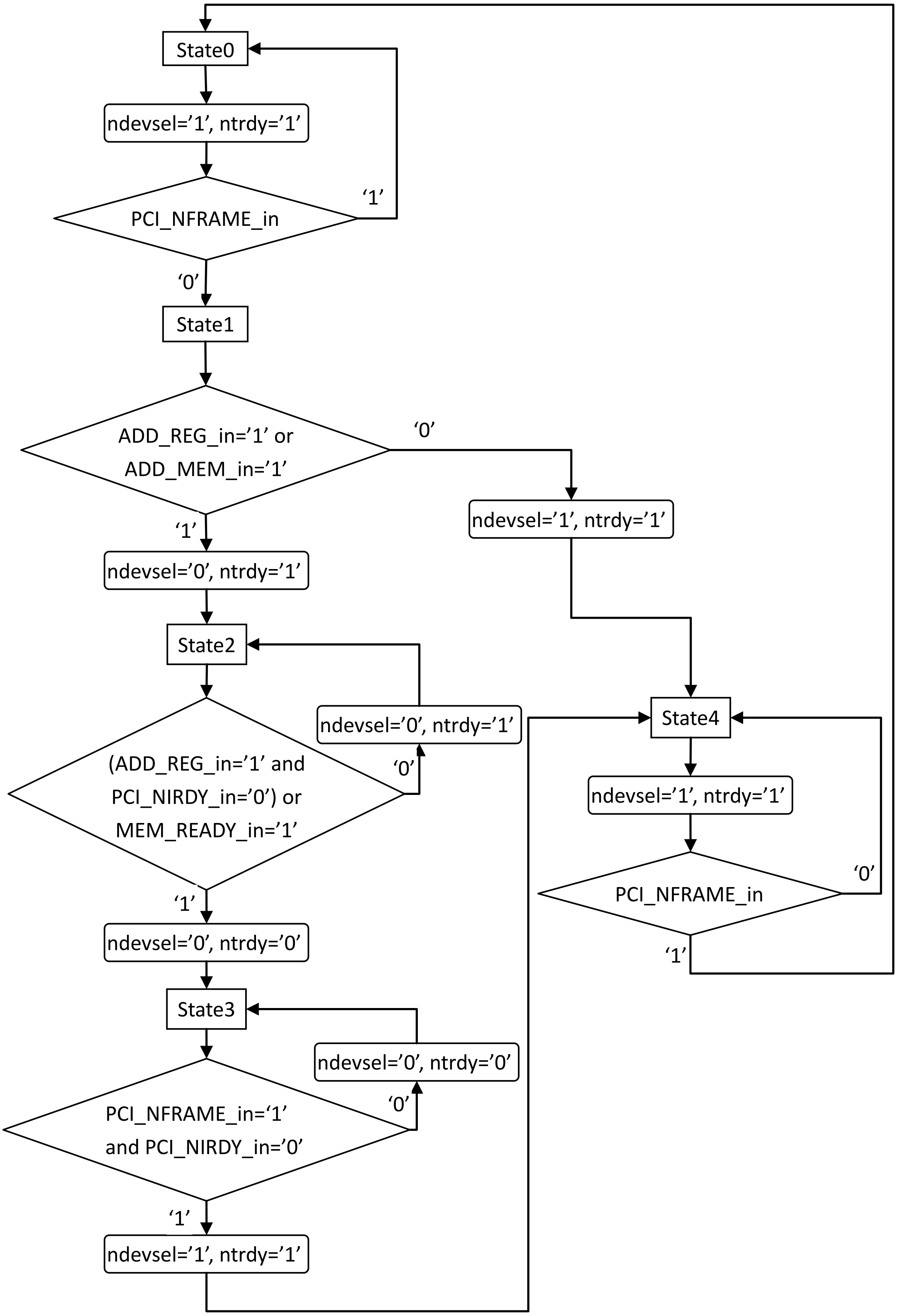
Figure 3.5 shows the diagram of the VHDL entity element. The entity **PCI_CTL**, saved in the file PCI_CTL.vhd, represents the internal module used by the **PCI_MOD**. The functionality of **PCI_CTL** is to create a state machine which satisfies the exact time sequence of PCI read and write transfer, and provides control signals to the other modules used by **PCI_MOD** (**PCI_ADD**, **PCI_DATA**, **PCI_REG** and **PCI_PAR**).

Table 3.4 Output Control Signals in the PCI_CTL

Controlled Module	Signal Name	Function
PCI_ADD	ADD_LD_out	Control PCI_ADD to load the address
PCI_DATA	DAT_OE_out	Enable PCI_DATA to output the data
	MEM_LD_out	Control PCI_DATA to load data from MEM
	MEM_REG_SW_out	Control PCI_DATA to differentiate data from MEM from data from PCI_REG
PCI_REG	REG_RD_out	Provide read signal to the PCI_REG
	REG_WR_out	Provide write signal to the PCI_REG
PCI_PAR	PAR_OE_out	Output enable signal for PCI_PAR

PCI_CTL uses the signals listed in the Table 3.4 to make the other modules used by **PCI_MOD** work. It provides appropriately timed hand-shake signals for the PCI bus. On the other hand, **PCI_CTL** will utilize the control signals from the PCI bus interface, and signals `REGMEM_W_R_in`, `ADD_MEM_in` and `ADD_REG_in,` which are from the module **PCI_ADD**, control the state machine (figure 3.6). The state machine in the **PCI_CTL** sends back hand-shake signals to PCI bus interface and makes the board recognizable as a PCI device by the PC.

Figure 3.6 State Machine Chart of PCI Controller



The state machine will wait at state 0 until the PC decides to transfer data by setting PCI_NFRAME_in to ‘0’. In state 1, the **PCI_CTL** receives the signals from the module **PCI_ADD** means that it has already obtains the address for either memory or configuration register; the state will change to state 2. Otherwise, the state will change to 4. If so, there is a transfer failure of this cycle. State 2 is used to wait for the memory to prepare for the data, once the memory is ready, the state will change to 3. In state 3, the data will be transferred through the PCI bus. The state machine may stay in the state 3 for several cycles to handle PCI bus burst transmission of the data, which occurs when the data transfer cycles appear sequentially on the PCI bus. After the data transmission, the state will change to 4, and the state machine will wait in this state until PC releases the control of the PCI bus (set PCI_NFRAME_in to ‘1’).

3.2.2 PCI_ADD: Generate address from the time-multiplexed Address/Data bus

Figure 3.7 Interface of the PCI_ADD

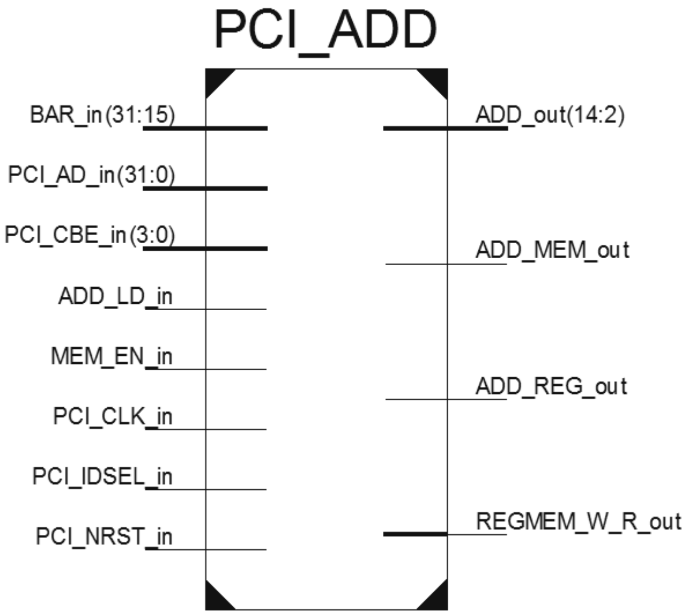


Figure 3.7 shows the diagram of the VHDL entity element. The module **PCI_ADD**, saved in the file PCI_ADD.vhd, is the internal module used by the **PCI_MOD**. The functionality of this module is to obtain the address information from the time-multiplexed Address/Data bus (PCI_AD_in), resolve the command from the PCI_CBE_in(3 downto 1) signals, and distinguish read or write from the PCI_CBE_in(0). Bar_in and MEM_EN_in are signals from the **PCI_REG** module to inform the **PCI_ADD** to check whether the address is for this PCI device. On the output side, **PCI_ADD** needs to output the address for the external memory **MEM** using the signals ADD_out and the memory write or read signal REGMEM_W_R_out.

3.2.3 PCI_DATA: Data Bus Selector

Figure 3.8 Interface of the PCI_Data

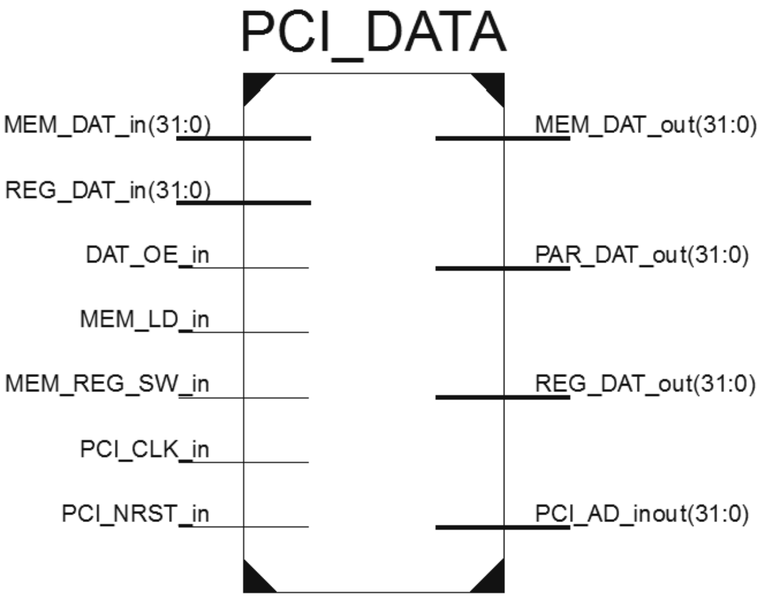


Figure 3.8 shows the diagram of the VHDL entity element. The file PCI_DATA.vhd creates the module **PCI_DATA**, which is also an internal module for the **PCI_MOD**. As Table 3.5 shows there are 5 one-way data bus (MEM_DAT_in, REG_DAT_in, MEM_DAT_out, PAR_DAT_out and REG_DAT_out) and 1 bidirectional data bus PCI_AD_inout; the functionality of the **PCI_DATA** module is to connect these data bus according to the input control signals (DAT_OE_in, MEM_LD_in and MEM_REG_SW_in).

Table 3.5 Function of Data Bus Used in the PCI_DATA

Data Bus Name	Direction	Function
MEM_DAT_in	From MEM to PCI_DATA	Read data from memory module MEM
MEM_DAT_out	From PCI_DATA to MEM	Write data to memory module MEM
REG_DAT_in	From PCI_REG to PCI_DATA	Read data from the internal configuration registers in PCI_REG
REG_DAT_out	From PCI_DATA to PCI_REG	Write data to the internal configuration registers in PCI_REG
PAR_DAT_out	From PCI_DATA to PCI_PAR	Data send to the PCI_PAR to generate the parity signal
PCI_AD_inout	Connect PCI bus interface and PCI_DATA	Receive data from the PCI bus or send data to the PCI bus

3.2.4 PCI_REG: PCI Internal Configuration Registers

Figure 3.9 Interface of the PCI_REG

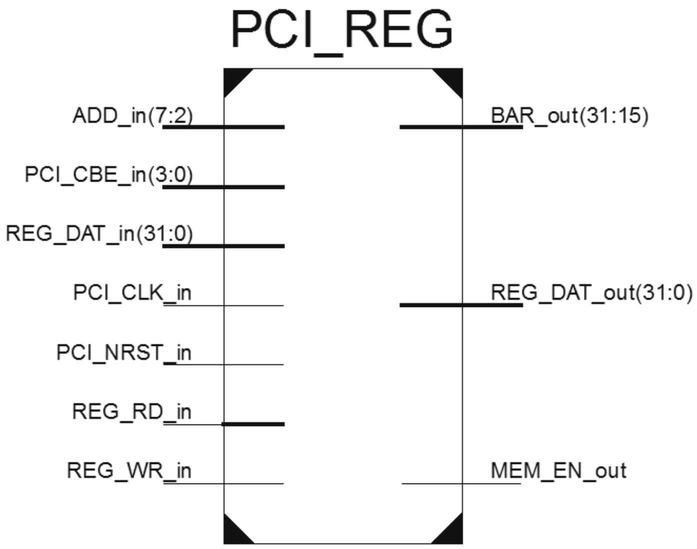


Figure 3.9 shows the diagram of the VHDL entity **PCI_REG**. The module PCI_REG, generate from the file PCI_REG.vhd, is also an internal module used by the entity **PCI_MOD**. Each functional PCI device possesses a block of 64 doublewords, showed in Table 3.4, reserved for the implementation of its configuration register. The format,

or usage, of the first 16 doublewords called device's configuration header region is predefined by the PCI specification. The functionality of this module **PCI_REG** is to implement this header region to help the PC recognize this PCI device.

Table 3.6 PCI Device's Configuration Header Region [5]

Byte 3	Byte 2	Byte 1	Byte 0	Address(Hex)
Device ID		Vendor ID		00
Status Register		Command Register		04
Class Code			Revision ID	08
BIST	Header Type	Latency Timer	Cache Line Size	0C
Base Address 0				10
Base Address 1				14
Base Address 2				18
Base Address 3				1C
Base Address 4				20
Base Address 5				24
Card Bus CIS Pointer				28
Subsystem ID		Subsystem Vendor ID		2C
Expansion ROM Base Address				30
Reserved				34
Reserved				38
Max_Lat	Min_Gnt	Interrupt Pin	Interrupt Line	3C

The Vendor ID of the device is x10EE$_H$ and the Device ID is xA100$_H$ [12], because the board uses the FPGA from the Xilinx Corporation. Our concern is only the second least significant bit in the command register, Memory Access Enable, and uses it to generate the output signal `MEM_EN_out`. The Class Code is set to x068000$_H$, which is defined in the PCI standard, because the device works as the bridge device. The PCI device needs to access the memory module **MEM**, whose space is 32KB ($32\mathrm{K} = 2^{15}$), so Base Address 0 is used, and the 17 (32-15=17) most significant bits of the Base Address 0 are fixed by PC and the 15 least significant bits are not of concern by

PCI_REG. The 17 most significant bits of Base Address 0 will be output through the signal BAR_out. The Interrupt Pin set to x"01" means the device wants to use the interrupt, and the content of Interrupt Line register comes from PC. Finally, all the other registers are set to 0.

3.2.5 PCI_PAR: Generating the Parity Signal

Figure 3.10 Interface of the PCI_PAR

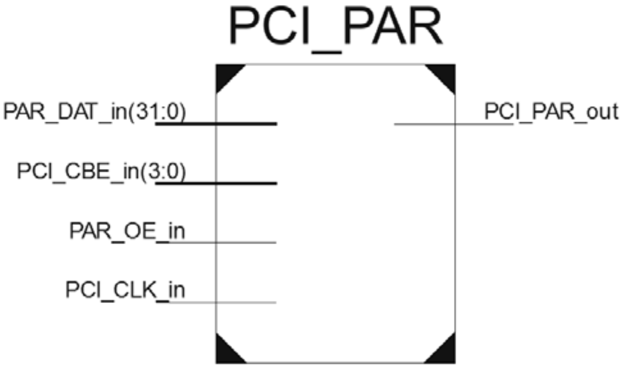


Figure 3.10 shows the diagram of the VHDL entity element PCI_PAR.The entity **PCI_PAR**, saved in the file PCI_PAR.vhd, is another internal module used by the **PCI_MOD**. The functionality of this module is to generate, and output, the parity signal which is used to ensure the correctness of the data signal PAR_DAT_in and byte enables signal PCI_CBE_in in the target read cycle.

3.3 MEM: Generating Memory Using VHDL

The entity **MEM**, saved in the file MEM.vhd, is the top-level module of the memory. The **MizzouRISC** and the module **PCI_MOD** share in using the memory, and the **MEM** is composed of three modules: **MEM_RAM**, **MEM_REG** and **MEM_ROM**;

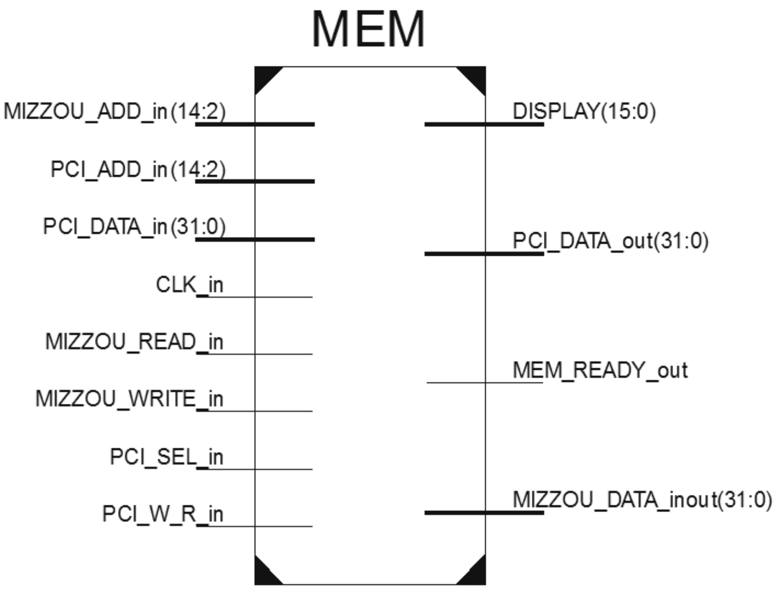
the functionality of the **MEM** is to determine which module (**MizzouRISC** or **PCI_MOD**) will access the memory, and distinguish which component (**MEM_RAM**, **MEM_REG** or **MEM_ROM**) will be accessed by the value of the address.

Table 3.7 Memory Space

Address(Hexadecimal)	Type	File	Access
x0000H - x00FFH	ROM	MEM_ROM.vhd	MizzouRISC
x3F80H - x3FFFH	REGISTERs	MEM_REG.vhd	MizzouRISC, PCI module
x4000H - x7FFFH	RAM	MEM_RAM.vhd	MizzouRISC, PCI module

Table 3.7 shows the memory space for now. The distribution and size of each module can be changed according to the need, since the FPGA can generate the memory with some flexibility.

Figure 3.11 Interface of the MEM



As figure 3.11 shows the signals of the module **MEM** can be divided to 3 groups: signals for the **MizzouRISC**, signals for the module **PCI_MOD** and signals for the **DISP_CTL**. The signals are listed in the Table 3.8.

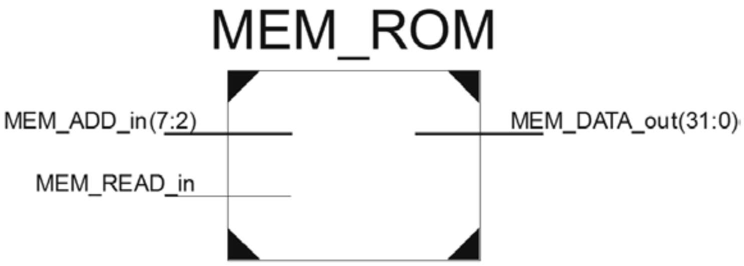
Table 3.8 Signals of the MEM

Group	Signal Name	Function
MizzouRISC	MIZZOU_ADD_in	The address bus of the MizzouRISC
	MIZZOU_READ_in	The read signal of the MizzouRISC
	MIZZOU_WRITE_in	The write signal of the MizzouRISC
	MIZZOU_DATA_inout	The data bus of the MizzouRISC
PCI_MOD	PCI_ADD_in	The address bus of the PCI_MOD
	PCI_DATA_in	The data bus from PCI_MOD to MEM
	PCI_DATA_out	The data bus from MEM to PCI_MOD
	PCI_SEL_in	MEM select signal from the PCI_MOD
	PCI_W_R_in	The write or read signal from PCI_MOD
	MEM_READY_out	Memory ready signal to the PCI_MOD
DISP_CTL	DISPLAY	Display content appears on the 7-segments LEDs

The module **MEM** uses the signals `MIZZOU_READ_in`, `MIZZOU_WRITE_in`, `PCI_SEL_in` and `PCI_W_R_in` to differentiate it is a read or write operation, and which module will access the memory.

3.3.1 MEM_ROM

Figure 3.12 Interface of the MEM_ROM



The module **MEM_ROM**, saved in the file MEM_ROM.vhd, is an internal module used by the entity **MEM**. The functionality of the **MEM_ROM** is to save the program of the **MIZZOU_RISC**. The content of this module cannot be modified by the module **PCI_MOD** or **MizzouRISC**, and only **MizzouRISC** can read the data. For now, the size of the **MEM_ROM** is 64 by 32bits, and the test program saved in it is used to do the inverse color processing.

Algorithm 3.1 Pseudocode of the test program in the MEM_ROM

```
When the interrupt is coming
Do {
    Read the fragment size
    Loop (fragment size time)
        {
            Read pixel value from the RAM
            Subtract that value from 255
            Write the new value to the RAM
        }
    }
```

The test program is used to invert the color of each pixel in the gray-level image fragment saved in the **MEM_RAM**. For the detail operation, once the interrupt is coming, **MizzouRISC** will determine the size of the fragment at first, and then, read each pixel out, subtract it from 255, and write the new value of each pixel back to the original address in the **MEM_RAM**.

3.3.2 MEM_RAM

Figure 3.13 Interface of the MEM_RAM

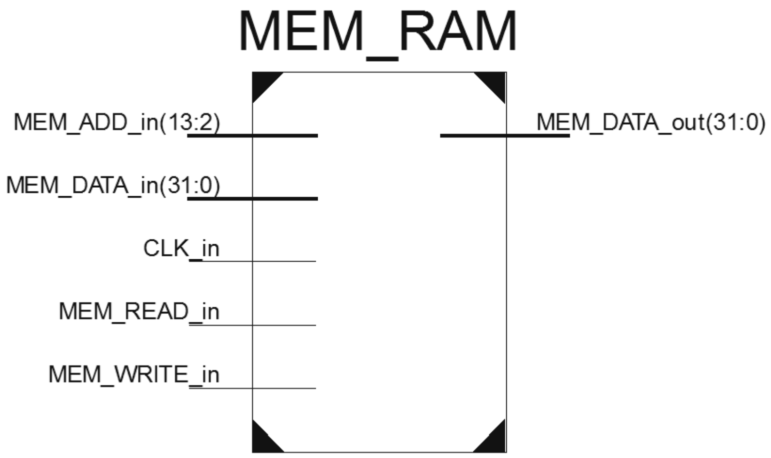


Figure 3.13 shows the diagram of the VHDL entity **MEM_RAM**. **MEM_RAM** saved in the file MEM_RAM.vhd, is also an internal module used by the entity **MEM**. The size of the **MEM_RAM** is 16KB, which is limited by the FPGA, because we generate the memory from the FPGA for now. Both of the **MizzouRISC** and **PCI_MOD** can access this module, and this module is usually to save the data blocks such as image fragment.

3.3.3 MEM_REG

Figure 3.14 Interface of the MEM_REG

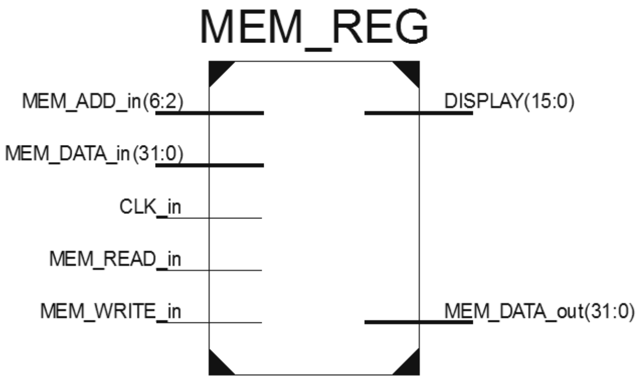
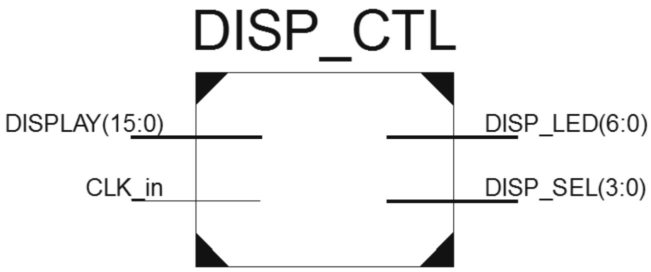


Figure 3.14 shows the signals of the VHDL entity MEM_REG. The module **MEM_REG**, saved in the file MEM_REG.vhd, is also an internal module used by entity **MEM**. The functionality of this module is quite similar with the **MEM_RAM**, but the size of the **MEM_REG** is only 32 by 32 bits. The MEM_REG is usually used to save the individual data such as the size, height or width of the image fragment. In addition, the last position of the MEM_REG, at the address x3FFCH – x3FFFH, is used to save the label register, which is an important label for the state machine of the entity **SYSTEM**. The value of the label register will be output to the signal DISPLAY, and be sent to the module **DISP_CTL**.

3.4 DISP_CTL: The Controller of the 7-Segments LEDs

Figure 3.15 Interface of the DISP_CTL



The Entity **DISP_CTL** showed in the Figure 3.15, saved in the file DISP_CTL.vhd, is the top-level file of the 7-segments LEDs controller. This entity will use another module called **DISP_DEC**, which is saved in the file DISP_DEC.vhd, and the functionality of the DISP_DEC is to translate the input hexadecimal digit to the 7-segments code. With the help of the module **DISP_DEC**, the entity **DISP_CTL** can light the 4 7-segments LEDs, and display the value of the 16 bits signal DISPLAY on the 4 7-segments LEDs in hexadecimal digit.

3.5 Summary

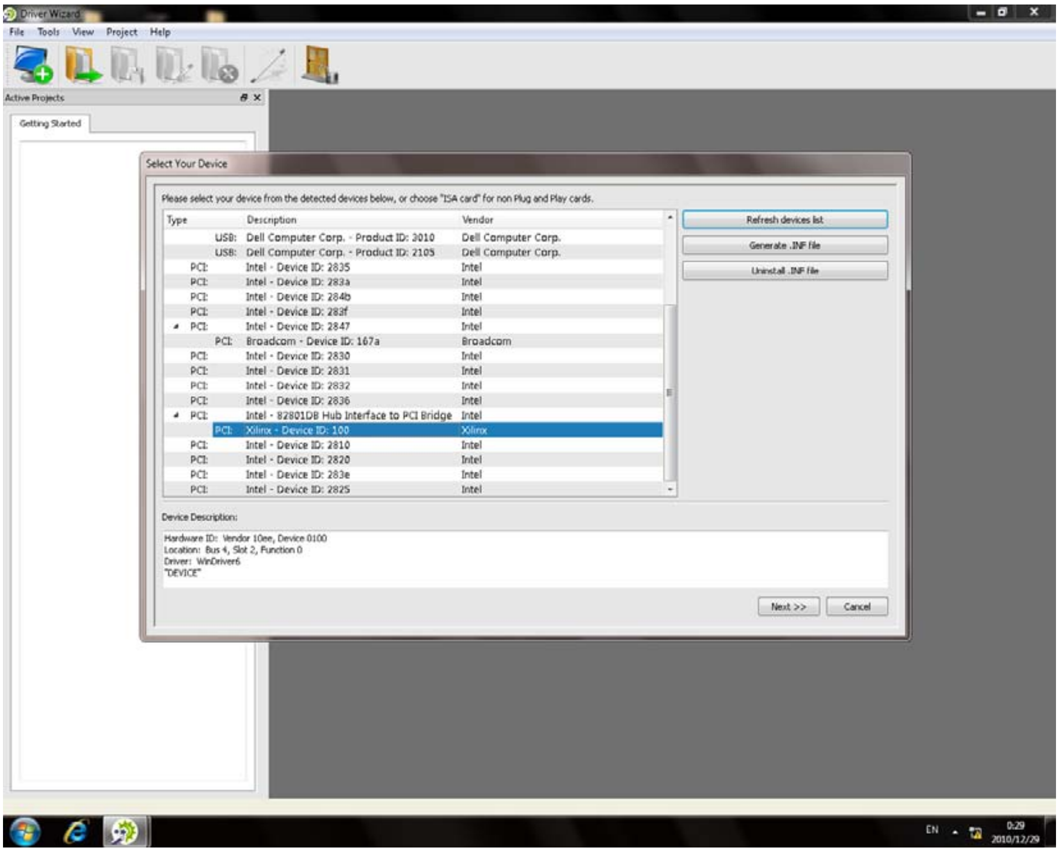
In this chapter, we introduced the VHDL entity **SYSTEM**, which is implemented on the FPGA, and the 4 submodule of the entity **SYSTEM**: **PCI_MOD**, **MEM**, **MizzouRISC** and **DISP_CTL**. The firmware can make the PC recognizes the RaggedStone1 Board as a PCI device, transfer data through the PCI bus, and make the processor MizzouRISC to process the data according to the test program.

CHAPTER 4 : DRIVER AND CONTROLLER ON PC

Chapter3 showed the VHDL implementation on the RaggedStone1 board as a PCI device when it is inserted on the motherboard. However, the PC needs a driver to recognize this hardware, also an application program to act as a controller operates the state machine. This describes the construction of the driver and the algorithm of the controller.

4.1 Driver for the RaggedStone1 Board

Figure 4.1 Selecting Hardware on the Driver Wizard



Please select your device from the detected devices below, or choose "ISA card" for non Plug and Play cards.

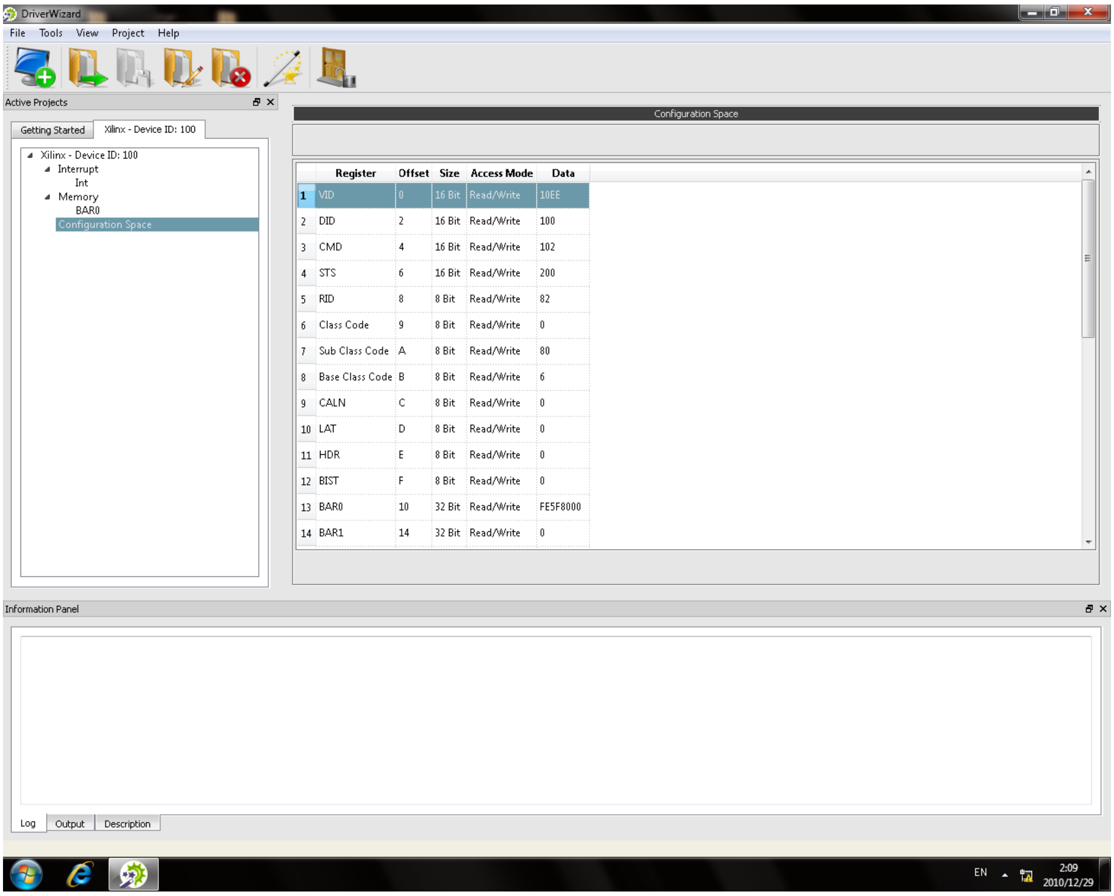
Type	Description	Vendor
USB:	Dell Computer Corp. - Product ID: 3010	Dell Computer Corp.
USB:	Dell Computer Corp. - Product ID: 2105	Dell Computer Corp.
PCI:	Intel - Device ID: 2835	Intel
PCI:	Intel - Device ID: 283a	Intel
PCI:	Intel - Device ID: 284b	Intel
PCI:	Intel - Device ID: 283f	Intel
PCI:	Intel - Device ID: 2847	Intel
PCI:	Broadcom - Device ID: 167a	Broadcom
PCI:	Intel - Device ID: 2830	Intel
PCI:	Intel - Device ID: 2831	Intel
PCI:	Intel - Device ID: 2832	Intel
PCI:	Intel - Device ID: 2836	Intel
PCI:	Intel - 82801DB Hub Interface to PCI Bridge	Intel
PCI:	Xilinx - Device ID: 100	Xilinx
PCI:	Intel - Device ID: 2810	Intel
PCI:	Intel - Device ID: 2820	Intel
PCI:	Intel - Device ID: 283e	Intel
PCI:	Intel - Device ID: 2825	Intel

Device Description:

Hardware ID: Vendor 10ee, Device 0100
Location: Bus 4, Slot 2, Function 0
Driver: WinDriver6
"DEVICE"

We use the Windows7 WinDriver PCI for Windows (Free Trial) from Jungo LTD. to develop the driver for the board [13]. Driver Wizard is a GUI tool in the WinDriver, as shown in figure4.1. The dialog box in the middle shows all the hardware installed on the PC, and we selected the RaggedStone1 board. As we introduced in the Chapter 3, the vendor ID of the hardware (FPGA) is x10EE$_H$, which means Xilinx, and the Device ID we set is x100$_H$. After we select the device, we can click the button "Generate .INF file" to get the driver of this PCI device. This inf file can make the operating system recognize this PCI hardware.

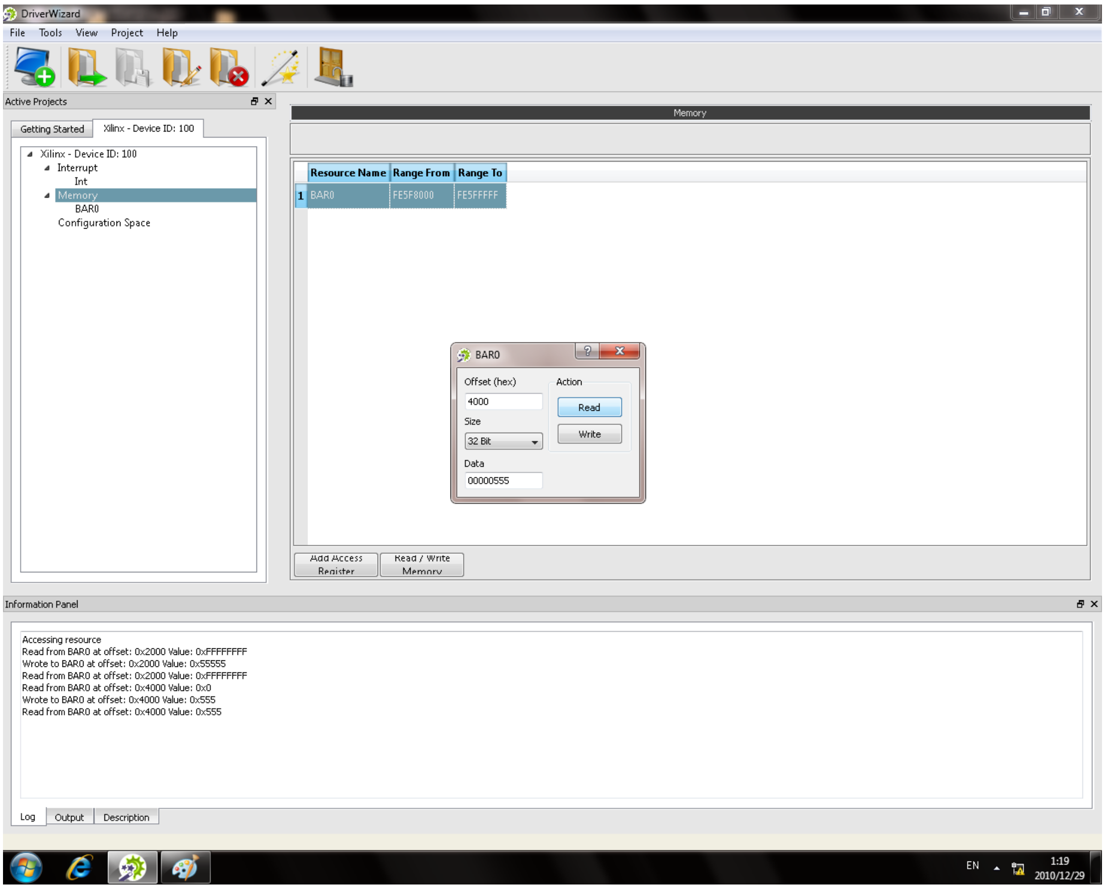
Figure 4.2 Specific of the PCI Device on Driver Wizard



	Register	Offset	Size	Access Mode	Data
1	VID	0	16 Bit	Read/Write	10EE
2	DID	2	16 Bit	Read/Write	100
3	CMD	4	16 Bit	Read/Write	102
4	STS	6	16 Bit	Read/Write	200
5	RID	8	8 Bit	Read/Write	82
6	Class Code	9	8 Bit	Read/Write	0
7	Sub Class Code	A	8 Bit	Read/Write	80
8	Base Class Code	B	8 Bit	Read/Write	6
9	CALN	C	8 Bit	Read/Write	0
10	LAT	D	8 Bit	Read/Write	0
11	HDR	E	8 Bit	Read/Write	0
12	BIST	F	8 Bit	Read/Write	0
13	BAR0	10	32 Bit	Read/Write	FE5F8000
14	BAR1	14	32 Bit	Read/Write	0

After we install the RaggedStone1 driver, we can use the software Driver Wizard to see the detail of the PCI device. As Figure 4.2 shows, the software displays the values of the configuration registers which is set by the VHDL entity **PCI_REG** described in Chapter 3. In addition, the left column of this figure represents that the device has 1 memory space and can use the interrupt, which is set by the configuration space of this PCI device.

Figure 4.3 Reading and Writing Memory on Driver Wizard



	Resource Name	Range From	Range To
1	BAR0	FE5F8000	FE5FFFFF

Offset (hex): 4000

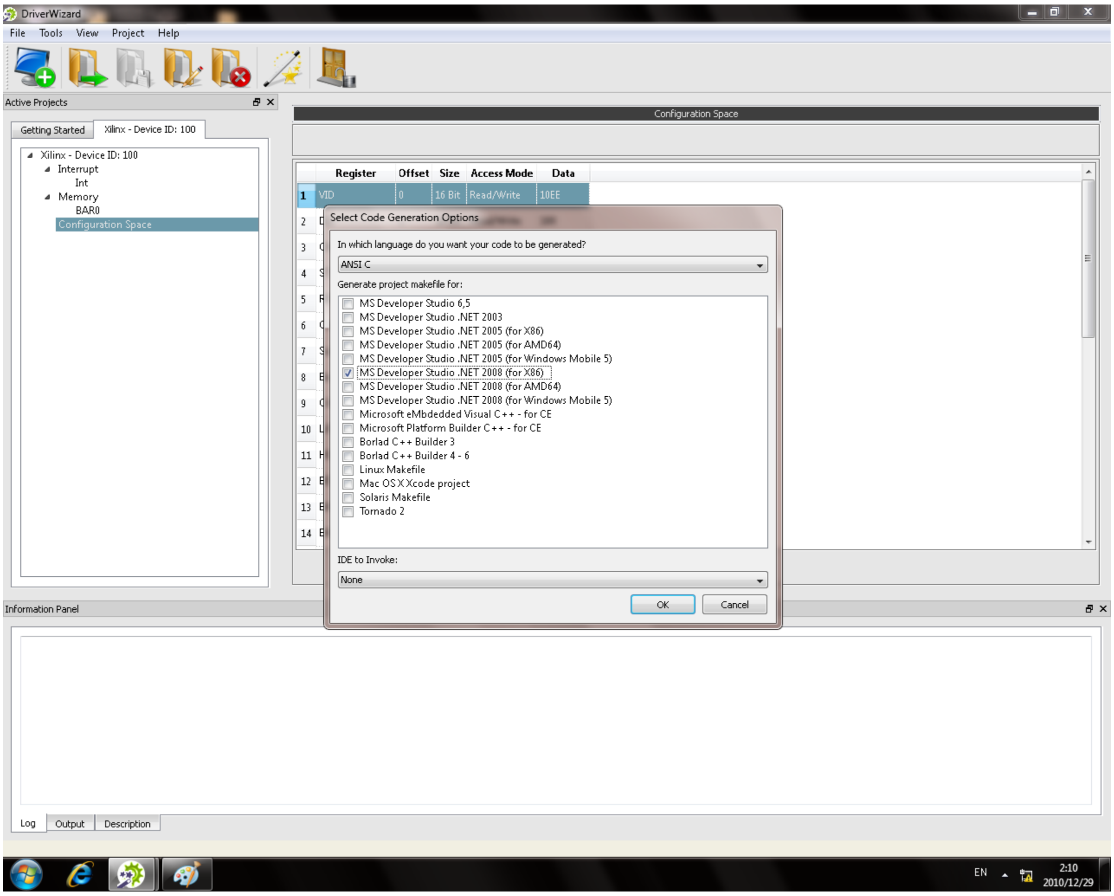
Size: 32 Bit

Data: 00000555

```
Accessing resource
Read from BAR0 at offset: 0x2000 Value: 0xFFFFFFFF
Wrote to BAR0 at offset: 0x2000 Value: 0x55555
Read from BAR0 at offset: 0x2000 Value: 0xFFFFFFFF
Read from BAR0 at offset: 0x4000 Value: 0x0
Wrote to BAR0 at offset: 0x4000 Value: 0x555
Read from BAR0 at offset: 0x4000 Value: 0x555
```

The software Driver Wizard provides some simple functionality to test the hardware such as Read/Write Memory. As Figure 4.3 shows, we can select the "Memory" on the left column, and click the button "Read/Write Memory", and then, the dialog box will appear in the middle of the screen. To input the address of the memory use the field Offset (Hex), to select the width of the data bus (8 bits, 16 bits, and 32bits) use the size field, to input single data value written to memory use the data field, finally click the button “Write”. In that case, the data will be written to the memory at the location specified. After that, we can click the button “Read” to test whether the PCI device, the RaggedStone1 board, works correctly. As expect, the result we read from the memory is same as the data we write, the PCI device work correctly demonstrated that we can read and write data through the PCI bus to the RaggedStone1 board.

Figure 4.4 Generating Code Using the Software Driver Wizard



Configuration Space

	Register	Offset	Size	Access Mode	Data
1	VID	0	16 Bit	Read/Write	10EE

Select Code Generation Options

In which language do you want your code to be generated?

ANSI C

Generate project makefile for:

- □ MS Developer Studio 6,5
- □ MS Developer Studio .NET 2003
- □ MS Developer Studio .NET 2005 (for X86)
- □ MS Developer Studio .NET 2005 (for AMD64)
- □ MS Developer Studio .NET 2005 (for Windows Mobile 5)
- ☑ MS Developer Studio .NET 2008 (for X86)
- □ MS Developer Studio .NET 2008 (for AMD64)
- □ MS Developer Studio .NET 2008 (for Windows Mobile 5)
- □ Microsoft eMbdedded Visual C++ - for CE
- □ Microsoft Platform Builder C++ - for CE
- □ Borlad C++ Builder 3
- □ Borlad C++ Builder 4 - 6
- □ Linux Makefile
- □ Mac OS X Xcode project
- □ Solaris Makefile
- □ Tornado 2

IDE to Invoke:

None

OK Cancel

We can acquire some API functions for the PCI device RaggedStone1 board by clicking the button “Generating Code” on the top of the screen. As Figure 4.4 shows, the dialog box will appear on the screen after we click the button “Generating Code”, and then we can select the IDE we used for the development of the controller. In our project, we use the IDE Microsoft Visual Studio.net 2008 as one of our development tools.

4.2 Controller for the RaggedStone1 Board

We developed the application program, which works as a controller, with the C programming language, using the IDE Visual Studio 2008. We also generated APIs from the software Driver Wizard, which can help us write data, read data, and listen to the interrupt from the PCI Device RaggedStone1 board. The controller needs to read the image file, generate useful information from the header of the image file, and write the header of the output file. In addition, the controller also needs to write all the fragments, wait for the interrupt, and read the processed fragments from the PCI device RaggedStone1 board. Finally, the controller has to write the processed data (fragments) to the output file.

Algorithm 4.1 Reading File

```
Open 24-bit bmp file
Read the header of the file and get the length, width and size of the image
Write the header of the output file
For i=0 to length
    For j=0 to width
        Read 3 characters from the file saved in pixel.red, pixel.green and pixel.blue
        Mypdata[i*width+j]=pixel.red*0.33 + pixel.green * 0.33 + pixel.blue * 0.33
```

In our project, the controller is set to handle the 24-bit bmp file. As Algorithm 4.1 shows, after the controller opens the file, gets information from the header such as length, width and size, and write the header to the output file [14]. The loop reads a column then all rows in that column and generates gray-level data (8 bits) by calculating the average value of the red, green and blue components of each pixel. This is because the test program, saved in the VHDL entity **MEM_ROM** described in Chapter 3, is to invert the gray scale of the image.

Algorithm 4.2 PCI Write and PCI Read

```
Write fragment size to PCI bus at address x3F80H
Do(
    Write fragment to PCI bus from address 4000H to 7FFFH
    Write 0000F0F0 to PCI bus at label register’s address 0x3FFCH
    Interrupt Enable
    While(1)
        If mylabel=1 then break
    Interrupt disable and set mylabel to 0
    Write 0000A0A0 to PCI bus at label register’s address 0x3FFCH
    Read data from address 4000H to 7FFFH
    Write 0000B0B0 to PCI bus at label register’s address 0x3FFCH)
Until all the data has been processed
```

The controller needs to interact with the state machine in the FPGA, introduced in Chapter3. The Algorithm 4.2 shows the pseudocode of the PCI transaction from the PC point of view. The **MizzouRISC** needs to know the size of the fragment to determine how many cycles it will use to handle the whole fragment; so PC will write the number of the fragment size to the VHDL entity **MEM_REG** located at x3F80$_H$. And then, PC can write the fragments of the image to the VHDL entity MEM_RAM, and write x0000F0F0$_H$ to the label register at the VHDL entity **MEM_REG** in order to declare that the writing process is done.

After writing the fragment, the PC will wait for the interrupt; and it will write x0000A0A0$_H$ to the label register in order to stop the interrupt, until the PC gets the interrupt from the PCI device. Afterward, the PC can read the processed data through the PCI bus, and set the label register to x0000B0B0$_H$ to inform the hardware that reading process is done.

The size of the **MEM_RAM** is 16K byte. If the gray-level image is larger than 16KB, the PC needs to divide the image to several 16KB fragments and repeat the writing and reading procedure to process the whole image. This is the reason we use a loop in the Algorithm 4.2.

Algorithm 4.3 Writing File and Close File

```
Writing processed data to the output file
Free the buffer
Close files
```

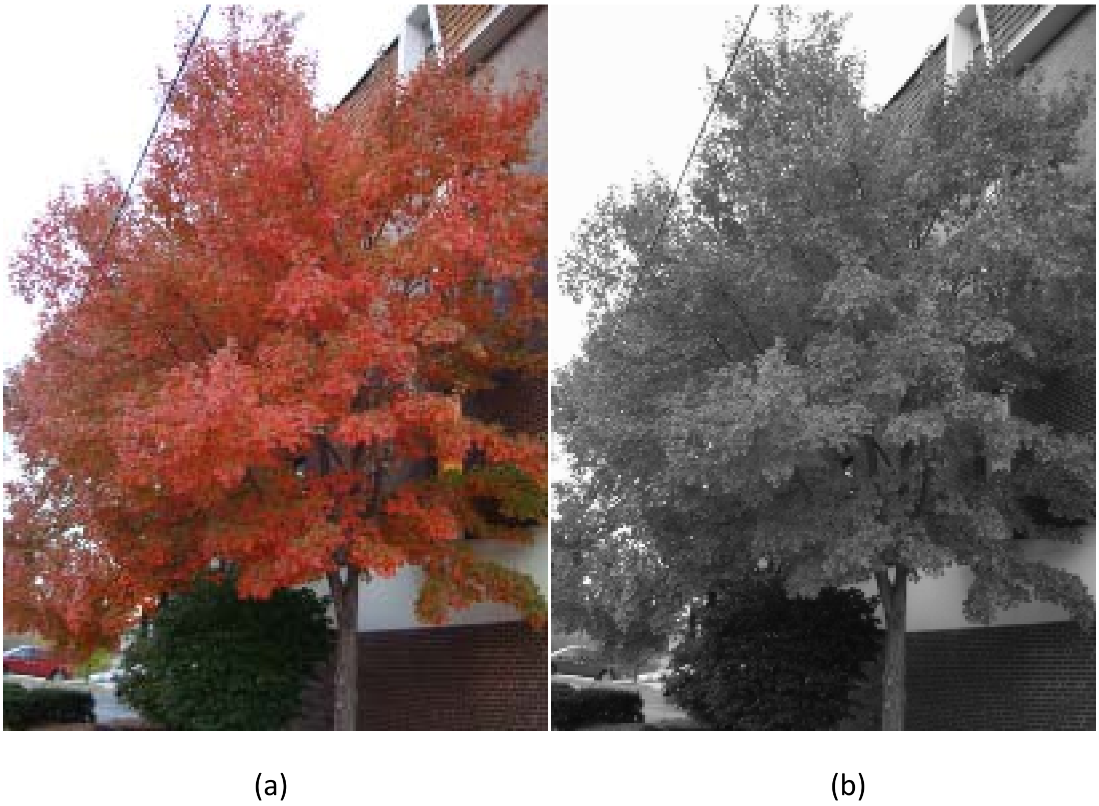
All the processed fragments will be saved in a large buffer, and the controller will write the data in the buffer to the output file as Algorithm 4.3 described. After that, the controller can free the buffer, and close the original file and new output file.

CHAPTER 5 : EXPERIMENT RESULT

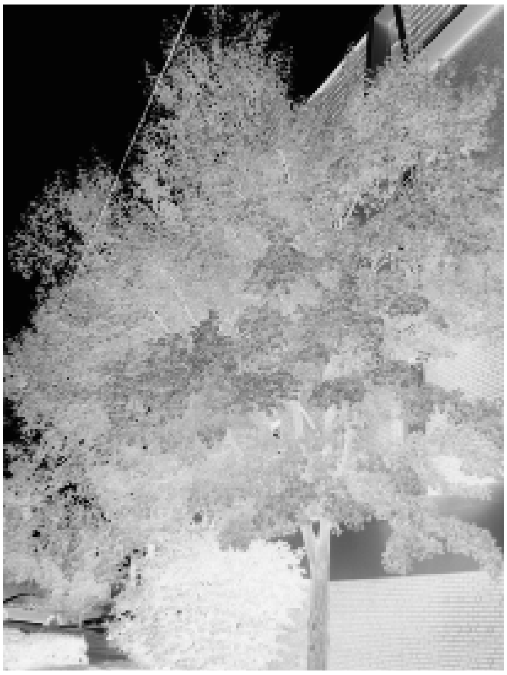
We carried out two experiments on the Entire System. The first one is to input an image to the board, read it back, and display the result on the PC to see whether the PCI transmission module and MizzouRISC work. The second experiment is to send several images in different size to the board, and clock the processing time to evaluate the relationship between the time and the image size.

5.1 Figure Result

Figure 5.1 Test Figure and Result



(a) (b)



(c)

Figure 5.1 (a) is the input figure. The size of the figure is 800X600 pixels and the figure is saved in a 24-bits bmp file. The program on the PC will open the file and read the information from the figure including the figure size, height, width and color map. And then, it will read the red, green and blue colors of each pixel, add them together, and divide by 3 to get the gray-level figure. Figure 5.1 (b) shows the gray-level figure after the processing with the C program.

The C program then will write the value of the pixels in Figure 5.1 (b) to the **MEM_RAM** on the board through the PCI bus. After waiting for the **MizzouRISC** to process the data, the PC will read the data back from the memory. In the experiment, because the size of the **MEM_RAM** is only 16KB, the processing cycle will be repeated 30 times. Finally, the PC can save the data after processing in another bmp file; and the result shows in Figure 5.1 (c).

5.2 Figure Size and Processing Time

Our experiment was to access the performance of the PCI connection to MizzouRISC. It is convenient to divide into 5 operative steps. The first operative is read file from the disk, which includes reading the header and generating the gray-level figure. Second, write 16KB fragment data from the gray-level figure to the board through the PCI bus. Third, wait until MizzouRISC has finishes processing this fragment. Fourth, read the data back from the board through the PCI bus. The three steps PCI writing, MizzouRISC Processing and PCI reading will, of course, be repeated several times. The last step is writing the data to a new file.

Figure 5.2 Flow Chart of the PC

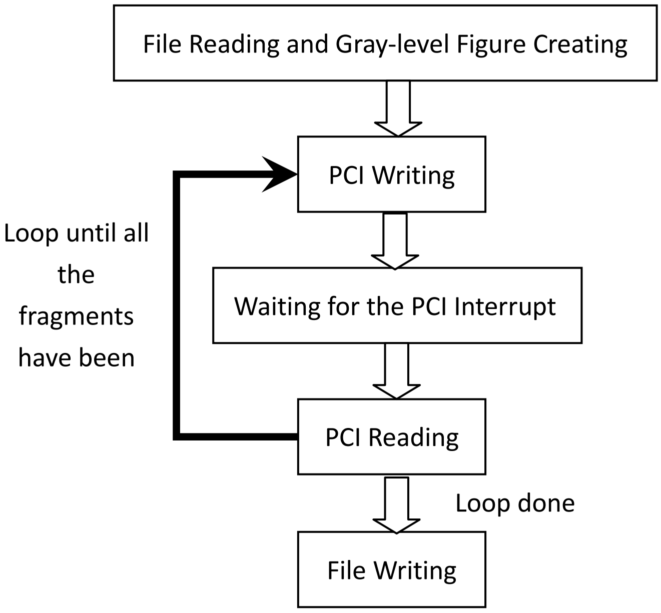

File Reading and Gray-level Figure Creating
PCI Writing
Loop until all the fragments have been
Waiting for the PCI Interrupt
PCI Reading
Loop done
File Writing

Figure 5.3 Figure Size and Execute Time (1)

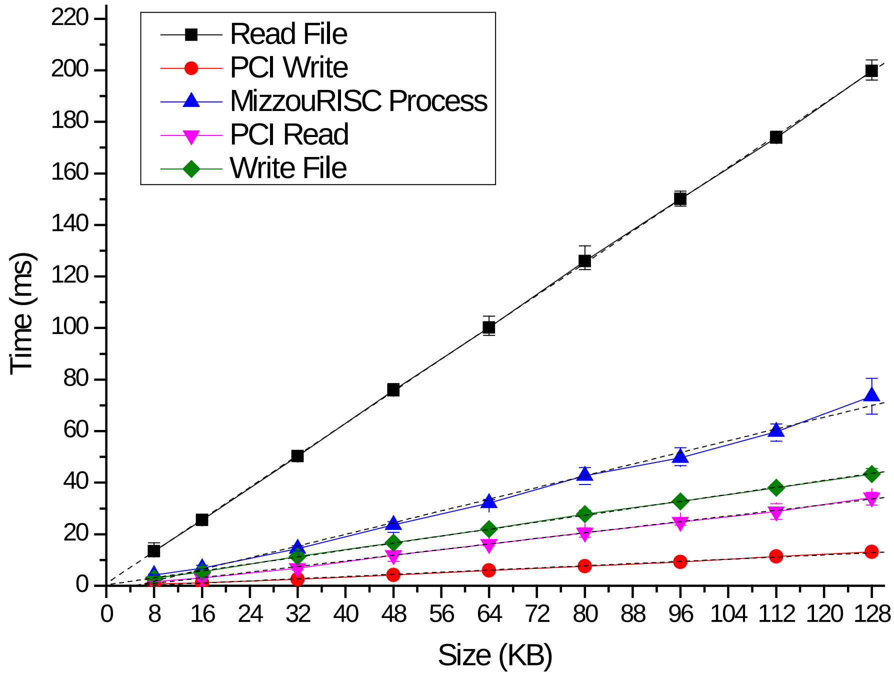


Figure 5.3 shows the relationship between the figure size and the time spent on each step. The horizontal axis of the figure shows the size of the test figuret, 8KB, 16KB, 32KB, 48KB, 64KB, 80KB, 96KB, 112KB and 128KB. The vertical axis shows the time spent on each step of processing. In the figure, each experiment for the same size of figure is replicated 10 times; each point plotted is the average of the 10 values and the bars show the range of errors.

Each line shows the relationship between the time required for the activity and the figure size. The black line with square shows the time to read the file. The file reading includes getting information from the header of the original bmp file, writing the header of the output file, and generating the gray-level figure. For the reading file, the time spent on processing the header is negligible compared with the time to read pixels from the file and generate the gray-level image. So the time spent on file reading has a strong relationship with the figure size and the line for file reading is linear. However, the file writing time records the time of writing the value of each pixel to the file. So the line of writing is also linear. It is obvious that file reading requires much more time than file writing.

The line with the inverted triangle (pink) shows reading the PCI time and the line with the dot (red) shows the time of PCI writing. Because **MEM_RAM** is limited to 16KB, figures larger than 16KB require PCI reading and PCI writing to repeat several times. In this case, the program accumulates the time of PCI reading and PCI writing from the first cycle to the last cycle to get the final result. For PCI writing, the PC acting as initiator device controls the PCI's address/data (shared) bus in both address cycle and data cycle. In that case, the address and data cycle are adjacent. However, for PCI reading, PC (master) controls the address/data bus during the address cycle, but the board (target) provides the data. It is possible to insert one or several turn-around cycle by the PC to wait until the board retrieved the data, hence PCI read requires more time than PCI write on Figure 5.3. Finally, for both of the PCI read and PCI write cycles, there will be 16KB data transacted on the PCI bus in each cycle. So the time of the PCI read and PCI write relates to the file size and then both lines are linear.

The line with the triangle (blue) shows the MizzouRISC processing time. It is linear for the same reason as the PCI read and PCI write; time is related to the number of the cycles. In addition, PC uses interrupt listening to get the end flag of MizzouRISC processing, and for the PCI bus the interrupt line is connected to the interrupt controller in PC. Since other PC devices connect to the interrupt controller, the program cannot guarantee that the PC can respond to the interrupt as soon as the board sends the interrupt. This is the reason why the points have more errors and are not exactly on the line.

Typically, the time to transmit the data through the PCI bus is smaller than the time to access the hard disk. MizzouRISC uses the clock from the PCI bus and it will use several instructions to handle one pixel. In that case, it will use more time than that required for PCI read and PCI write.

Figure 5.4 Figure Size and Execute Time (2)

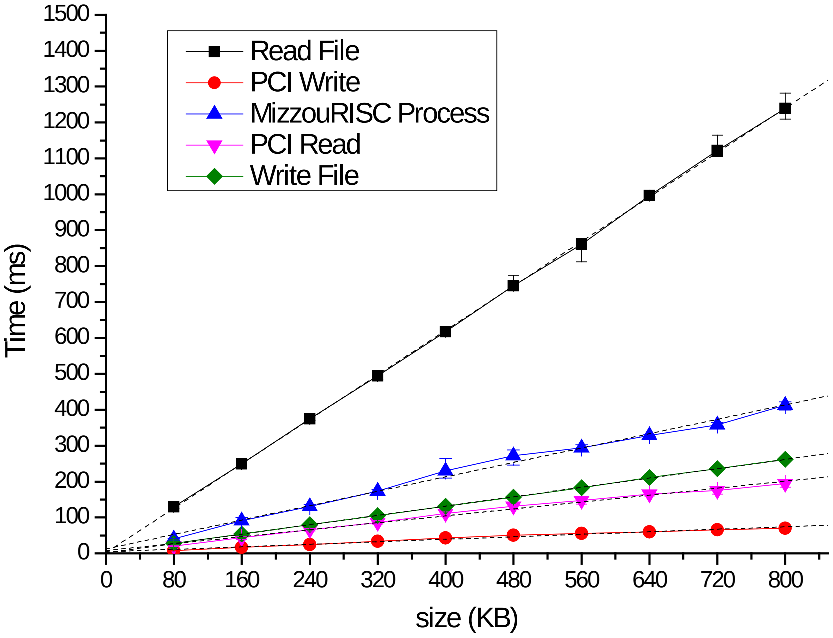


Figure 5.4 is quite similar with the figure 5.3. However, the fragments in figure 5.4 rise from 80KB to 800KB, 80KB steps. The data shows that the speeds of data transfer rates are dependent on file size.

CHAPTER 6 : DISCUSSION AND CONCLUSION

After the design of the PCI transmission module (**PCI_MOD**) and the memory (**MEM**) in VHDL, we have combined it with the **MizzouRISC**, created the firmware **SYSTEM**, and implemented it on the RaggedStone1 board with Xilinx XC3S1500 FPGA. On the other hand, we use Windriver PCI for Windows Free Trial and Microsoft Visual Studio 2008 to build the driver for the Windows7 operating system, and the application program (controller) on the PC. Figure 5.1 demonstrated that the success of the driver and test program on PC, and firmware on the hardware performed the image inverse; also we showed in Figure 5.2 and Figure 5.3 that data transfer rates depend on figure size and surprisingly the limited RAM for the system will not affect the result. This works are by dividing the test to 5 operations, reading the clock to evaluate the relationship of the test fragment size and the time.

The size of the memory is the main problem for the design. It makes the software on the PC complicated, and also makes the evaluation imprecise. In addition, the Xilinx ISE Design Suit 12.2 Web pack edition needs long time to route and place the firmware, because **MEM_RAM** utilizes a lot of hardware resources; also, debugging the interrupt for the PCI in software is difficult.

Future Work

As mentioned above the **MEM_RAM** created by the FPGA limited the performance of the project; hence a SDRAM hardware module will reduce much of this burden. In addition, a faster clock can make the MizzouRISC perform better, because the processor need not be limited by the speed of the PCI bus. Further design of a new board for MizzouRISC should include a faster clock for processor and SDRAM hardware module, use a separate PCI clock for the PCI module, and even integrate PS/2 port and VGA port on the board to provide MizzouRISC with its own input and output device.

In software, Windriver PCI for Windows Free Trial is very good software to test the hardware, but the Free Trial has limited execution time. Developing a driver and controller based on DDK is a solution to make MizzouRISC commercial.

REFERENCES

[1] Tyrer, Harry, "Design And Simulation of a Reduced Instruction Set Computer" in Progress in Computer-Aided VLSI Design, volume 3, George Zobrig, Ed. Norwood, NJ: Abblex, 1990, pg 325-351.

[2] Nash, Sean, “MizzouSMP”, Electrical and Computer Engineering electronic theses and dissertations (MU) 2009.

[3] Soft CPU Cores for FPGA. http://www.1-core.com/library/digital/soft-cpu-cores/ (accessed January 3, 2011).

[4] Charles H. Roth, Jr., and Lizy Kurian John, “Digital Systems Design Using VHDL”, Second Edition, Thomson Learning, 2008, Chapter 9.

[5] MindShare, Inc., “PCI System Architecture”, Third Edition, Addison Wesley, 1995.

[6] Conventional PCI. http://en.wikipedia.org/wiki/Conventional_PCI/ (accessed January 3, 2011).

[7] Low Cost Spartan-3 PCI Development Board. http://www.enterpoint.co.uk/moelbryn/raggedstone1.html, (accessed January 3, 2011).

[8] QueryPerformanceFrequency Function. http://msdn.microsoft.com/en-us/library/ms644905(v=vs.85).aspx, (accessed January 3, 2011).

[9] QueryPerformanceCounter Function.

http://msdn.microsoft.com/en-us/library/ms644904(v=vs.85).aspx

(accessed January 3, 2011).

[10] Benjamin Krill, “High Performance Reconfigurable Computing Enviroment”, MSc in Distributed Computing Systems Engineering, Brunel University, 2007.

[11] “Xilinx PCI core enables low-cost FPGA solution”, Electronic News, Volume 45, Iss. 2257, pg 30, 1pgs, Feb 15, 1999.

[12] PCI Vendor and Device Lists. http://www.pcidatabase.com/,

(accessed January 3, 2011).

[13] PCI Software: WinDriver PCI for windows.

http://www.jungo.com/st/windriver_windows.html, (accessed January 3, 2011).

[14] BMP File Format. http://en.wikipedia.org/wiki/BMP_file_format/,

(accessed January 3, 2011).

APPENDIX

Appendix A – Measurement Result

Raw data for the for the Figure 5.4 and Figure 5.5

A.1 Relationship of the Processing Time and Fragment Size (Fragments from 8KB to 128 KB)

Table A.1.1 Read File (Fragments from 8KB to 128 KB)

size	Repeat 10 times(ms)									
8KB	13.179	12.988	16.73	13.059	12.912	12.866	13.084	12.833	12.899	13.185
16KB	24.977	25.234	25.593	26.497	25.371	25.797	25.394	25.925	25.327	25.517
32KB	49.323	50.182	50.379	50.15	50.434	49.858	52.431	50.138	50.272	49.973
48KB	78.513	78.039	74.826	74.511	74.274	76.471	77.903	74.925	74.878	74.283
64KB	101.682	99.901	99.564	97.409	99.034	97.088	104.595	98.356	104.599	98.999
80KB	126.071	122.687	123.393	127.264	127.891	124.864	127.734	131.913	124.297	123.647
96KB	151.703	149.634	148.088	149.856	147.935	150.188	153.171	153.08	147.364	150.227
112KB	175.613	173.705	176.36	171.866	172.128	172.794	172.886	175.767	173.428	173.957
128KB	204.033	200.488	198.232	196.222	200.652	197.845	200.349	199.882	202.993	196.476

Table A.1.2 PCI Write (Fragments from 8KB to 128 KB)

size	Repeat 10 times(ms)									
8KB	0.774	0.732	0.715	0.714	0.712	0.709	0.721	0.714	0.722	0.726
16KB	1.09	1.101	1.102	1.091	1.087	1.099	1.087	1.087	1.085	1.093
32KB	2.062	2.316	2.853	2.171	2.814	2.326	2.081	2.841	2.809	2.804
48KB	4.465	3.788	3.802	3.832	4.182	3.821	4.286	5.08	3.811	5.047
64KB	5.564	6.08	5.48	6.024	6.051	5.935	6.03	6.7	5.368	5.567
80KB	7.661	7.221	7.798	7.147	8.101	7.32	9.064	7.172	7.65	7.236
96KB	9.543	10.394	8.104	8.115	8.294	10.549	10.051	8.298	8.065	10.296
112KB	12.465	10.487	11.031	12.705	10.571	10.91	12.056	11.127	11.475	10.763
128KB	12.02	13.515	13.311	13.278	12.723	13.794	13.819	14.199	12.935	11.577

Table A.1.3 MizzouRISC Process (Fragments from 8KB to 128 KB)

size	Repeat 10 times(ms)									
8KB	4.203	4.603	4.043	4.016	4.068	4.051	4.048	4.057	4.081	4.188
16KB	7.18	6.644	6.669	7.907	6.77	6.656	6.656	6.674	6.612	6.73
32KB	13.561	14.935	14.97	15.436	14.911	13.707	13.478	15.573	13.637	13.727
48KB	23.718	23.774	24.633	20.747	23.87	24.842	23.983	24.401	23.529	23.311
64KB	32.808	31.645	32.123	32.01	31.722	31.451	33.433	32.005	30.606	32.569
80KB	41.839	44.724	42.303	39.272	41.106	45.815	43.645	43.51	39.745	45.179
96KB	51.817	50.66	47.717	48.53	50.442	53.525	48.818	51.818	47.128	46.591
112KB	56.117	62.094	58.095	58.526	61.591	61.24	62.816	58.814	59.632	58.258
128KB	78.75	80.085	66.516	66.874	77.85	69.751	68.005	66.706	80.478	79.711

Table A.1.4 PCI Read (Fragments from 8KB to 128 KB)

size	Repeat 10 times(ms)									
8KB	1.662	1.683	1.65	1.63	1.649	1.636	1.657	1.649	1.654	1.632
16KB	3.01	3.011	3.034	2.989	3.014	2.997	2.986	3.003	3.032	3.028
32KB	6.185	7.482	7.474	7.584	6.822	6.169	6.141	7.52	6.117	6.095
48KB	13.242	12.607	11.36	9.486	12.144	12.268	12.229	12.568	11.069	12.107
64KB	16.433	16.841	15.975	16.351	16.514	15.447	16.99	15.504	15.571	15.986
80KB	20.804	19.679	20.348	18.782	20.262	21.085	22.531	20.631	21.203	20.483
96KB	25.591	24.193	23.804	24.507	24.858	25.636	25.469	24.857	24.778	23.474
112KB	28.449	30.061	31.885	30.346	28.821	26.923	31.487	27.342	27.381	25.833
128KB	36.098	34.804	32.158	33.834	31.317	36.271	34.6	33.738	35.892	33.403

Table A.1.5 Write File (Fragments from 8KB to 128 KB)

size	Repeat 10 times(ms)									
8KB	3.04	2.816	2.861	2.87	2.915	3.02	2.93	3.007	2.813	2.853
16KB	5.558	5.479	5.453	5.437	5.562	5.427	5.515	5.478	5.402	5.595
32KB	10.688	11.558	12.015	11.905	11.52	10.951	10.75	11.808	11.131	10.815
48KB	16.34	16.947	16.668	16.298	16.477	16.814	17.054	16.739	16.596	16.88
64KB	21.934	21.474	21.757	22.467	21.995	22.587	22.038	22.066	22.121	21.575
80KB	27.308	27.384	27.468	28.852	27.295	27.483	26.649	29.253	27.362	29.364
96KB	32.629	32.614	33.229	32.847	32.706	32.623	32.474	32.741	32.714	32.407
112KB	36.7	39.035	38.256	37.286	37.315	38.441	38.524	39.254	37.611	38.206
128KB	43.408	42.498	43.126	42.407	45.488	42.656	44.354	43.557	42.516	43.2

A.2 Relationship of the Processing Time and Fragment Size (Fragments from 80KB to 800 KB)

Table A.2.1 Read File (Fragments from 80KB to 800 KB)

size	Repeat 10 times(ms)									
80KB	126.798	130.894	130.691	135.18	135	130.87	125.05	128.629	128.688	127.683
160KB	245.976	249.293	251.326	244.815	250.186	249.396	245.115	249.381	249.04	257.503
240KB	372.557	379.31	384.558	372.195	372.261	376.566	369.351	372.524	373.513	370.941
320KB	498.936	490.663	493.229	496.034	508.397	496.738	483.753	491.017	497.118	486.525
400KB	617.968	618.757	612.543	622.778	624.345	616.559	608.096	616.771	611.338	621.368
480KB	739.797	773.407	744.583	737.429	747.09	738.428	737.416	737.453	745.925	753.733
560KB	865.535	811.612	853.468	860.214	873	859.97	867.3	875.533	877.422	867.344
640KB	1006.484	985.975	1011.132	984.9363	984.065	1005.088	1003.256	986.035	995.029	1002.018
720KB	1164.633	1111.036	1132.207	1130.802	1124.93	1120.024	1108.817	1105.143	1115.044	1107.963
800KB	1282.097	1255.263	1227.744	1209.819	1227.07	1222.908	1243.141	1226.684	1244.759	1252.203

Table A.2.2 PCI Write (Fragments from 80KB to 800 KB)

size	Repeat 10 times(ms)									
80KB	6.484	7.447	8.323	6.413	7.409	6.597	6.448	7.734	7.591	7.211
160KB	16.464	17.906	16.996	16.637	15.356	18.761	17.378	18.119	16.628	15.074
240KB	25.282	24.023	23.921	24.741	26.097	26.613	25.47	24.131	24.105	23.436
320KB	30.863	34.698	34.882	33.934	31.939	35.147	32.021	36.523	37.502	33.588
400KB	40.967	44.387	40.383	44.171	41.502	45.314	41.213	44.188	41.381	42.453
480KB	48.66	48.431	49.629	51.573	48.183	51.799	53.89	55.211	51.66	50.349
560KB	55.396	55.474	54.944	54.513	54.323	54.053	55.208	58.618	56.7	53.251
640KB	59.679	62.236	62.203	59.108	58.437	59.849	60.258	57.848	58.913	58.732
720KB	68.542	65.113	65.671	65.125	68.445	64.208	62.297	67.815	63.695	63.929
800KB	70.085	72.758	68.958	72.006	69.337	73.829	69.54	68.643	69.134	68.367

Table A.2.3 MizzouRISC Process (Fragments from 80KB to 800 KB)

size	Repeat 10 times(ms)									
80KB	38.239	42.06	42.339	39.158	42.043	41.057	36.986	40.848	39.95	41.039
160KB	98.972	86.319	87.369	92.655	88.66	84.643	96.862	87.484	92.132	95.596
240KB	131.504	134.57	127.187	131.163	129.477	129.524	128.217	130.067	136.005	128.537
320KB	175.023	178.79	173.754	173.184	171.011	171.384	169.814	172.861	173.969	172.954
400KB	243.261	264.849	218.219	227.971	217.751	219.899	209.931	216.017	220.368	257.932
480KB	267.493	267.265	279.374	278.222	245.681	268.135	284.354	276.108	287.92	268.624
560KB	296.477	295.033	291.615	293.751	300.383	302.445	288.477	288.427	289.639	294.054
640KB	329.069	326.615	326.09	326.796	331.68	331.599	330.109	325.345	329.42	329.433
720KB	355.71	358.34	364.344	355.607	362.707	360.637	357.907	355.363	354.342	358.222
800KB	418.258	410.607	398.867	410.224	402.899	421.721	416.413	413.036	418.336	415.171

Table A.2.4 PCI Read (Fragments from 80KB to 800 KB)

size	Repeat 10 times(ms)									
80KB	18.974	20.313	22.734	19.452	20.811	20.414	19.532	22.332	22.393	19.089
160KB	41.683	44.28	42.049	46.53	44.183	44.345	38.324	45.34	44.927	43.538
240KB	68.181	64.736	62.64	64.412	66.242	69.669	63.04	66.725	66.275	66.433
320KB	89.286	83.505	86.05	91.208	87.425	82.595	84.188	90.654	82.049	84.161
400KB	114.029	101.657	113.458	112.876	115.796	114.272	109.427	103.991	111.69	115.714
480KB	122.619	129.743	135.551	135.999	128.112	135.535	132.74	134.035	128.058	133.373
560KB	149.255	149.264	148.81	150.349	149.64	150.085	148.303	141.018	137.561	147.059
640KB	165.374	161.428	160.243	165.771	167.479	166.329	166.997	166.893	164.454	164.9
720KB	166.557	166.97	181.629	169.059	184.628	180.436	179.52	165.284	176.983	182.202
800KB	187.557	184.356	197.807	193.49	198.423	185.72	200.963	198.097	198.844	196.648

Table A.2.5 Write File (Fragments from 80KB to 800 KB)

size	Repeat 10 times(ms)									
80KB	29.806	27.501	27.544	28.156	29.054	28.098	27.591	27.713	27.123	27.322
160KB	53.975	53.261	53.56	53.409	53.974	52.989	53.422	53.361	53.894	53.737
240KB	80.357	79.784	80.01	79.693	80.179	78.601	79.679	80.824	81.495	78.661
320KB	105.818	106.301	104.827	106.529	104.145	105.427	104.373	109.116	104.9	104.588
400KB	131.605	130.048	130.166	133.751	130.491	133.075	129.889	132.616	130.076	131.611
480KB	155.387	156.757	157.368	157.418	158.731	155.812	156.569	155.734	155.107	157.953
560KB	181.789	182.425	182.664	182.511	183.715	185.78	181.443	184.014	183.003	179.666
640KB	210.487	211.935	208.984	208.579	221.008	209.015	213.148	209.478	210.037	207.781
720KB	234.359	235.281	237.914	237.16	236.886	234.01	238.335	236.119	234.203	235.107
800KB	263.269	261.866	261.36	261.702	265.708	261.821	261.936	263.369	260.776	258.489

Appendix B – VHDL Source Code

B.1 System

B.1.1 System.vhd

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY System IS
PORT(
            PCI_CLK_in          : in std_logic;
            PCI_NRST_in         : in std_logic;
            PCI_AD_inout        : inout std_logic_vector(31 downto 0);
            PCI_CBE_in          : in std_logic_vector(3 downto 0);
            PCI_PAR_out         : out std_logic;
            PCI_NFRAME_in       : in std_logic;
            PCI_NIRDY_in        : in std_logic;
            PCI_NTRDY_out       : out std_logic;
            PCI_NDEVSEL_out     : out std_logic;
            PCI_NSTOP_out       : out std_logic;
            PCI_IDSEL_in        : in std_logic;
            PCI_NPERR_out       : inout std_logic;
            PCI_NSERR_out       : inout std_logic;
            PCI_NINT_out        : out std_logic;
            PCI_NREQ_in         : in std_logic;
            PCI_NGNT_in         : in std_logic;
            BUT_RST_in          : in std_logic;
            BUT_NRST_in         : in std_logic;
            DISP_SEL            : inout std_logic_vector(3 downto 0);
            DISP_LED            : out std_logic_vector(6 downto 0);
            DISP_DOT            : out std_logic;
            LED_MHALT_out       : out std_logic;
            LED_PCI             : out std_logic;
            LED_ALIVE           : out std_logic
);
END System;

ARCHITECTURE behave OF System IS
      COMPONENT mizzou_risc IS
      PORT(
            clk, reset : in std_logic;
            addbus : inout std_logic_vector(31 downto 0) := "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
            databus : inout std_logic_vector(31 downto 0) := "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
```

```
        priority : in std_logic_vector(1 downto 0);
        int_ack : out std_logic := '0';
        read_memory : out std_logic := '0';
        write_memory : out std_logic := '0';
        halt : out std_logic := '0'
);
END COMPONENT;

COMPONENT MEM IS
PORT(
        CLK_in                  : in std_logic;
        MIZZOU_ADD_in           : in std_logic_vector(14 downto 2);
        MIZZOU_DATA_inout       : inout std_logic_vector(31 downto 0);
        MIZZOU_READ_in          : in std_logic;
        MIZZOU_WRITE_in         : in std_logic;
        PCI_W_R_in              : in std_logic;
        PCI_SEL_in              : in std_logic;
        PCI_ADD_in              : in std_logic_vector(14 downto 2);
        PCI_DATA_in             : in std_logic_vector(31 downto 0);
        PCI_DATA_out            : out std_logic_vector(31 downto 0);
        DISPLAY                 : out std_logic_vector(15 downto 0);
        MEM_READY_out           : out std_logic
);
END COMPONENT;

COMPONENT PCI_MOD IS
PORT(
        PCI_CLK_in          : in std_logic;
        PCI_NRST_in         : in std_logic;
        PCI_AD_inout        : inout std_logic_vector(31 downto 0);
        PCI_CBE_in          : in std_logic_vector(3 downto 0);
        PCI_PAR_out         : out std_logic;
        PCI_NFRAME_in       : in std_logic;
        PCI_NIRDY_in        : in std_logic;
        PCI_NTRDY_out       : out std_logic;
        PCI_NDEVSEL_out     : out std_logic;
        PCI_IDSEL_in        : in std_logic;
        PCI_NINT_out        : out std_logic;
        PCI_NSTOP_out       : out std_logic;
        PCI_NPERR_out       : out std_logic;
        PCI_NSERR_out       : out std_logic;
        SYS_INT_in          : in std_logic;
        MEM_READY_in      : in std_logic;
        MEM_SEL_out       : out std_logic;
        MEM_W_R_out       : out std_logic;
        MEM_ADD_out       : out std_logic_vector(14 downto 2);
        MEM_DAT_in        : in std_logic_vector(31 downto 0);
        MEM_DAT_out       : out std_logic_vector(31 downto 0)
    );
    END COMPONENT;

    COMPONENT DISP_CTL IS
    PORT(
        CLK_in            : in std_logic;
        DISPLAY           : in std_logic_vector(15 downto 0);
        DISP_SEL          : inout std_logic_vector(3 downto 0);
        DISP_LED          : out std_logic_vector(6 downto 0)
    );
    END COMPONENT;

    signal rst           : std_logic;
    signal nrst          : std_logic;
    signal state         : std_logic_vector(2 downto 0):="000";
    signal nextstate     : std_logic_vector(2 downto 0):="000";
    signal DISPLAY       : std_logic_vector(15 downto 0);
    signal int_ack       : std_logic;
    signal SYS_INT       : std_logic;
    signal priority      : std_logic_vector(1 downto 0):="00";
    signal mizzouadd     : std_logic_vector(31 downto 0):="00000000000000000000000000000000";
    signal mizzoudata    : std_logic_vector(31 downto 0):="00000000000000000000000000000000";
    signal mizzouread    : std_logic;
    signal mizzouwrite   : std_logic;
    signal MEM_READY     : std_logic;
    signal PCI_SEL       : std_logic;
    signal PCI_W_R       : std_logic;
    signal PCI_ADD       : std_logic_vector(14 downto 2);
    signal PCI_DAT_A     : std_logic_vector(31 downto 0);
    signal PCI_DAT_B     : std_logic_vector(31 downto 0);

BEGIN
    rst <= not BUT_RST_in;
    nrst <= BUT_NRST_in and PCI_NRST_in;
    LED_PCI <= (not PCI_NREQ_in) and (not PCI_NGNT_in);
    LED_ALIVE <= '1';
```

```
DISP_DOT<='1';

PROCESS(PCI_CLK_in)
BEGIN
        IF (PCI_CLK_in = '1' and PCI_CLK_in'event) THEN
                state <= nextstate;
        END IF;
END PROCESS;

PROCESS(state, DISPLAY, int_ack)
BEGIN
        SYS_INT <= '0';
        priority <= "00";
        CASE state IS
                WHEN "000" =>
                        IF DISPLAY = x"F0F0" THEN
                                nextstate <= "001";
                        ELSE
                                nextstate <= "000";
                        END IF;
                WHEN "001" =>
                        priority <= "11";
                        IF int_ack = '1' THEN
                                nextstate <= "010";
                        ELSE
                                nextstate <= "001";
                        END IF;
                WHEN "010" =>
                        IF DISPLAY = x"5050" THEN
                                nextstate <= "011";
                        ELSE
                                nextstate <= "010";
                        END IF;
                WHEN "011" =>
                        IF DISPLAY = x"A0A0" THEN
                                nextstate <= "100";
                        ELSE
                                SYS_INT <= '1';
                                nextstate <= "011";
                        END IF;
                WHEN "100" =>
                        IF DISPLAY = x"B0B0" THEN
```

```
                    nextstate <= "000";
                ELSE
                    nextstate <= "100";
                END IF;
            WHEN others=>
                    nextstate <= "000";
        END CASE;
    END PROCESS;

E1: COMPONENT PCI_MOD port map (PCI_CLK_in, nrst, PCI_AD_inout, PCI_CBE_in, PCI_PAR_out,
PCI_NFRAME_in, PCI_NIRDY_in, PCI_NTRDY_out, PCI_NDEVSEL_out, PCI_IDSEL_in, PCI_NINT_out,
PCI_NSTOP_out, PCI_NPERR_out, PCI_NSERR_out, SYS_INT, MEM_READY, PCI_SEL, PCI_W_R, PCI_ADD,
PCI_DAT_A, PCI_DAT_B);
E2: COMPONENT mizzou_risc port map (PCI_CLK_in, rst, mizzouadd, mizzoudata, priority, int_ack,
mizzouread, mizzouwrite, LED_MHALT_out);
E3: COMPONENT MEM port map (PCI_CLK_in, mizzouadd(14 downto 2), mizzoudata, mizzouread,
mizzouwrite, PCI_W_R, PCI_SEL, PCI_ADD, PCI_DAT_B, PCI_DAT_A, DISPLAY, MEM_READY);
E4: COMPONENT DISP_CTL port map (PCI_CLK_in, DISPLAY, DISP_SEL, DISP_LED);
END behave;
```

B.1.2 System.ucf

```
NET "BUT_NRST_in"     LOC = "AA3"    |IOSTANDARD = LVTTL |PULLUP ;
NET "BUT_RST_in"      LOC = "Y4"     |IOSTANDARD = LVTTL |PULLUP ;
NET "LED_MHALT_out"   LOC = "AB5"    |IOSTANDARD = LVTTL |DRIVE = 24 ;
NET "LED_PCI"         LOC = "AA4"    |IOSTANDARD = LVTTL |DRIVE = 24 ;
NET "LED_ALIVE"       LOC = "AB4"    |IOSTANDARD = LVTTL |DRIVE = 24 ;
NET "DISP_LED<6>"     LOC = "W18"    |IOSTANDARD = LVTTL |DRIVE = 24 ;
NET "DISP_LED<5>"     LOC = "AA18"   |IOSTANDARD = LVTTL |DRIVE = 24 ;
NET "DISP_LED<4>"     LOC = "AB18"   |IOSTANDARD = LVTTL |DRIVE = 24 ;
NET "DISP_LED<3>"     LOC = "Y17"    |IOSTANDARD = LVTTL |DRIVE = 24 ;
NET "DISP_LED<2>"     LOC = "V18"    |IOSTANDARD = LVTTL |DRIVE = 24 ;
NET "DISP_LED<1>"     LOC = "AA20"   |IOSTANDARD = LVTTL |DRIVE = 24 ;
NET "DISP_LED<0>"     LOC = "AB20"   |IOSTANDARD = LVTTL |DRIVE = 24 ;
NET "DISP_DOT"        LOC = "W17"    |IOSTANDARD = LVTTL |DRIVE = 24 ;
NET "DISP_SEL<3>"     LOC = "U14"    |IOSTANDARD = LVTTL |DRIVE = 24 ;
NET "DISP_SEL<2>"     LOC = "U16"    |IOSTANDARD = LVTTL |DRIVE = 24 ;
NET "DISP_SEL<1>"     LOC = "U17"    |IOSTANDARD = LVTTL |DRIVE = 24 ;
NET "DISP_SEL<0>"     LOC = "AA17"   |IOSTANDARD = LVTTL |DRIVE = 24 ;
NET "PCI_CLK_in"      LOC = "A11"    |IOSTANDARD = LVTTL ;
NET "PCI_NRST_in"     LOC = "A19"    |IOSTANDARD = LVTTL ;
NET "PCI_PAR_out"     LOC = "A9"     |IOSTANDARD = LVTTL ;
NET "PCI_NFRAME_in"    LOC = "C13"    |IOSTANDARD = LVTTL ;
NET "PCI_NIRDY_in"     LOC = "A13"    |IOSTANDARD = LVTTL ;
NET "PCI_NTRDY_out"    LOC = "B13"    |IOSTANDARD = LVTTL ;
NET "PCI_NDEVSEL_out"  LOC = "E12"    |IOSTANDARD = LVTTL ;
NET "PCI_NSTOP_out"    LOC = "A12"    |IOSTANDARD = LVTTL ;
NET "PCI_IDSEL_in"     LOC = "D14"    |IOSTANDARD = LVTTL ;
NET "PCI_NPERR_out"    LOC = "D12"    |IOSTANDARD = LVTTL ;
NET "PCI_NSERR_out"    LOC = "B12"    |IOSTANDARD = LVTTL ;
NET "PCI_NINT_out"     LOC = "B19"    |IOSTANDARD = LVTTL ;
NET "PCI_NREQ_in"      LOC = "C18"    |IOSTANDARD = LVTTL ;
NET "PCI_NGNT_in"      LOC = "D18"    |IOSTANDARD = LVTTL ;
NET "PCI_CBE_in<0>"    LOC = "F9"     |IOSTANDARD = LVTTL ;
```

B.2 PCI_MOD

B.2.1 PCI_MOD.vhd

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY PCI_MOD IS
PORT(
        PCI_CLK_in          : in std_logic;
        PCI_NRST_in         : in std_logic;
        PCI_AD_inout        : inout std_logic_vector(31 downto 0);
        PCI_CBE_in          : in std_logic_vector(3 downto 0);
        PCI_PAR_out         : out std_logic;
        PCI_NFRAME_in       : in std_logic;
        PCI_NIRDY_in        : in std_logic;
        PCI_NTRDY_out       : out std_logic;
        PCI_NDEVSEL_out     : out std_logic;
        PCI_IDSEL_in        : in std_logic;
        PCI_NINT_out        : out std_logic;
        PCI_NSTOP_out       : out std_logic;
        PCI_NPERR_out       : out std_logic;
        PCI_NSERR_out       : out std_logic;
        SYS_INT_in          : in std_logic;
        MEM_READY_in        : in std_logic;
        MEM_SEL_out         : out std_logic;
        MEM_W_R_out         : out std_logic;
        MEM_ADD_out         : out std_logic_vector(14 downto 2);
        MEM_DAT_in          : in std_logic_vector(31 downto 0);
        MEM_DAT_out        : out std_logic_vector(31 downto 0)
);
END PCI_MOD;

ARCHITECTURE behave OF PCI_MOD IS

    COMPONENT PCI_CTL
    PORT(
        PCI_CLK_in         : in std_logic;
        PCI_NRST_in        : in std_logic;
        PCI_NFRAME_in      : in std_logic;
        PCI_NIRDY_in       : in std_logic;
        PCI_NDEVSEL_out    : out std_logic;
        PCI_NTRDY_out      : out std_logic;
        REGMEM_W_R_in      : in std_logic;
        ADD_REG_in         : in std_logic;
        ADD_MEM_in         : in std_logic;
        ADD_LD_out         : out std_logic;
        DAT_OE_out         : out std_logic;
        PAR_OE_out         : out std_logic;
        MEM_LD_out         : out std_logic;
        MEM_REG_SW_out     : out std_logic;
        REG_WR_out         : out std_logic;
        REG_RD_out         : out std_logic;
        MEM_SEL_out        : out std_logic;
        MEM_READY_in       : in std_logic
    );
    END COMPONENT;

    COMPONENT PCI_ADD
    PORT(
        PCI_CLK_in         : in std_logic;
        PCI_NRST_in        : in std_logic;
        PCI_AD_in          : in std_logic_vector(31 downto 0);
        PCI_CBE_in         : in std_logic_vector(3 downto 0);
        PCI_IDSEL_in       : in std_logic;
        BAR_in             : in std_logic_vector(31 downto 15);
        MEM_EN_in          : in std_logic;
        ADD_LD_in          : in std_logic;
        ADD_REG_out        : out std_logic;
        ADD_MEM_out        : out std_logic;
        ADD_out            : out std_logic_vector(14 downto 2);
        REGMEM_W_R_out : out std_logic
);
END COMPONENT;

COMPONENT PCI_DATA
PORT(
        PCI_CLK_in          : in std_logic;
        PCI_NRST_in         : in std_logic;
        PCI_AD_inout        : inout std_logic_vector(31 downto 0);
        DAT_OE_in           : in std_logic;
        MEM_LD_in           : in std_logic;
        MEM_REG_SW_in       : in std_logic;
        PAR_DAT_out         : out std_logic_vector(31 downto 0);
        MEM_DAT_in          : in std_logic_vector(31 downto 0);
        MEM_DAT_out         : out std_logic_vector(31 downto 0);
        REG_DAT_in          : in std_logic_vector(31 downto 0);
        REG_DAT_out         : out std_logic_vector(31 downto 0)
);
END COMPONENT;

COMPONENT PCI_PAR
PORT(
        PCI_CLK_in          : in std_logic;
        PAR_DAT_in          : in std_logic_vector(31 downto 0);
        PCI_CBE_in          : in std_logic_vector(3 downto 0);
        PAR_OE_in           : in std_logic;
        PCI_PAR_out         : out std_logic
);
END COMPONENT;

COMPONENT PCI_REG
PORT(
        PCI_CLK_in          : in std_logic;
        PCI_NRST_in         : in std_logic;
        PCI_CBE_in          : in std_logic_vector(3 downto 0);
        ADD_in              : in std_logic_vector(7 downto 2);
        REG_DAT_in          : in std_logic_vector(31 downto 0);
        REG_DAT_out         : out std_logic_vector(31 downto 0);
        REG_WR_in           : in std_logic;
        REG_RD_in           : in std_logic;
        BAR_out             : out std_logic_vector(31 downto 15);
        MEM_EN_out          : out std_logic
        );
        END COMPONENT;

        signal REGMEM_W_R       : std_logic;
        signal ADD_REG          : std_logic;
        signal ADD_MEM          : std_logic;
        signal ADD_LD           : std_logic;
        signal DAT_OE           : std_logic;
        signal PAR_OE           : std_logic;
        signal MEM_LD           : std_logic;
        signal MEM_REG_SW       : std_logic;
        signal MEM_EN           : std_logic;
        signal REG_WR           : std_logic;
        signal REG_RD           : std_logic;
        signal BAR              : std_logic_vector(31 downto 15);
        signal ADD              : std_logic_vector(14 downto 2);
        signal PAR_DAT          : std_logic_vector(31 downto 0);
        signal REG_DAT_A        : std_logic_vector(31 downto 0);
        signal REG_DAT_B        : std_logic_vector(31 downto 0);

BEGIN
        MEM_W_R_out     <= REGMEM_W_R;
        MEM_ADD_out     <= ADD;
        PCI_NSERR_out   <= 'Z';
        PCI_NSTOP_out   <= 'Z';
        PCI_NPERR_out   <= 'Z';
        PCI_NINT_out    <= '0' WHEN SYS_INT_in = '1' ELSE '1';
P1: COMPONENT PCI_CTL port map (PCI_CLK_in, PCI_NRST_in, PCI_NFRAME_in, PCI_NIRDY_in,
PCI_NDEVSEL_out, PCI_NTRDY_out, REGMEM_W_R, ADD_REG, ADD_MEM, ADD_LD, DAT_OE,
PAR_OE, MEM_LD, MEM_REG_SW, REG_WR, REG_RD, MEM_SEL_out, MEM_READY_in);
P2: COMPONENT PCI_ADD port map (PCI_CLK_in, PCI_NRST_in, PCI_AD_inout, PCI_CBE_in,
PCI_IDSEL_in, BAR, MEM_EN, ADD_LD, ADD_REG, ADD_MEM, ADD, REGMEM_W_R);
P3: COMPONENT PCI_DATA port map(PCI_CLK_in, PCI_NRST_in, PCI_AD_inout, DAT_OE, MEM_LD,
MEM_REG_SW, PAR_DAT, MEM_DAT_in, MEM_DAT_out, REG_DAT_A, REG_DAT_B);
P4: COMPONENT PCI_PAR port map (PCI_CLK_in, PAR_DAT, PCI_CBE_in, PAR_OE, PCI_PAR_out);
P5: COMPONENT PCI_REG port map (PCI_CLK_in, PCI_NRST_in, PCI_CBE_in, ADD(7 downto 2),
REG_DAT_B, REG_DAT_A, REG_WR, REG_RD, BAR, MEM_EN);
END behave;
```

B.2.2 PCI_ADD.vhd

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY PCI_ADD IS
PORT(
        PCI_CLK_in          : in std_logic;
        PCI_NRST_in         : in std_logic;
        PCI_AD_in           : in std_logic_vector(31 downto 0);
        PCI_CBE_in          : in std_logic_vector(3 downto 0);
        PCI_IDSEL_in        : in std_logic;
        BAR_in              : in std_logic_vector(31 downto 15);
        MEM_EN_in           : in std_logic;
        ADD_LD_in           : in std_logic;
        ADD_REG_out         : out std_logic;
        ADD_MEM_out         : out std_logic;
        ADD_out             : out std_logic_vector(14 downto 2);
        REGMEM_W_R_out : out std_logic
);
END PCI_ADD;

ARCHITECTURE behave OF PCI_ADD IS
        signal add          : std_logic_vector(31 downto 0);
        signal cmd          : std_logic_vector(3 downto 0);
        signal idsel        : std_logic;

BEGIN
    PROCESS(PCI_NRST_in, PCI_CLK_in, PCI_AD_in, PCI_CBE_in, PCI_IDSEL_in)
    BEGIN
        IF(PCI_NRST_in = '0') THEN
            add <= ( others => '1' );
            cmd <= ( others => '1' );
            idsel <= '0';
        ELSIF(PCI_CLK_in = '1' and PCI_CLK_in'event) THEN
            IF(ADD_LD_in = '1') then
                add <= PCI_AD_in;
                cmd <= PCI_CBE_in;
                idsel <= PCI_IDSEL_in;
            END IF;
        END IF;
    END PROCESS;
    ADD_REG_out <= '1' WHEN ((idsel = '1') and (add(1 downto 0) = "00") and (cmd(3 downto 1) =
    "101")) ELSE '0';
    ADD_MEM_out <= '1' WHEN ((MEM_EN_in = '1') and (add(31 downto 15) = BAR_in) and (add(1
    downto 0) = "00") and (cmd(3 downto 1) = "011"))   ELSE '0';
    ADD_out <= add(14 downto 2);
    REGMEM_W_R_out <= cmd(0);
END behave;
```

B.2.3 PCI_CTL.vhd

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY PCI_CTL IS
PORT(
        PCI_CLK_in          : in std_logic;
        PCI_NRST_in         : in std_logic;
        PCI_NFRAME_in       : in std_logic;
        PCI_NIRDY_in        : in std_logic;
        PCI_NDEVSEL_out     : out std_logic;
        PCI_NTRDY_out       : out std_logic;
        REGMEM_W_R_in       : in std_logic;
        ADD_REG_in          : in std_logic;
        ADD_MEM_in          : in std_logic;
        ADD_LD_out          : out std_logic;
        DAT_OE_out          : out std_logic;
        PAR_OE_out          : out std_logic;
        MEM_LD_out          : out std_logic;
        MEM_REG_SW_out      : out std_logic;
        REG_WR_out          : out std_logic;
        REG_RD_out          : out std_logic;
        MEM_SEL_out         : out std_logic;
        MEM_READY_in        : in std_logic
);
END PCI_CTL;

ARCHITECTURE behave OF PCI_CTL IS
    signal state            : integer range 0 to 4;
    signal nextstate        : integer range 0 to 4;
    signal state0           : std_logic;
    signal state2           : std_logic;
    signal state3           : std_logic;
```

```
        signal nextstate0       : std_logic;
        signal nextstate2       : std_logic;
        signal nextstate3       : std_logic;
        signal nextstate4       : std_logic;
        signal targetOE         : std_logic;
        signal ndevsel          : std_logic;
        signal devsel           : std_logic;
        signal trdy             : std_logic;
        signal ntrdy            : std_logic;
        signal REG_RD           : std_logic;

BEGIN
        state0                  <= '1' WHEN (state = 0) ELSE '0';
        state2                  <= '1' WHEN (state = 2) ELSE '0';
        state3                  <= '1' WHEN (state = 3) ELSE '0';
        nextstate0              <= '1' WHEN (nextstate = 0) ELSE '0';
        nextstate2              <= '1' WHEN (nextstate = 2) ELSE '0';
        nextstate3              <= '1' WHEN (nextstate = 3) ELSE '0';
        nextstate4              <= '1' WHEN (nextstate = 4) ELSE '0';
        PCI_NDEVSEL_out         <= devsel WHEN (TargetOE = '1') else 'Z';
        PCI_NTRDY_out           <= trdy WHEN (TargetOE = '1') else 'Z';
        MEM_SEL_out             <= '1' WHEN (ADD_MEM_in = '1' and state2 = '1') ELSE '0';
        REG_WR_out              <= '1' WHEN (ADD_REG_in = '1' and REGMEM_W_R_in = '1' and state3 = '1')
                                ELSE '0';
        REG_RD                  <= '1' WHEN (ADD_REG_in = '1' and REGMEM_W_R_in = '0' and (state2 = '1'
                                or state3 = '1')) ELSE '0';
        ADD_LD_out              <= '1' WHEN (PCI_NFRAME_in = '0' and state0 = '1') ELSE '0';
        MEM_LD_out              <= MEM_READY_in;
        MEM_REG_SW_out <= not REG_RD;
        REG_RD_out              <= REG_RD;

        PROCESS(PCI_NRST_in, PCI_CLK_in, nextstate)
        BEGIN
                IF(PCI_NRST_in = '0' ) THEN
                        state <= 0;
                ELSIF(PCI_CLK_in = '1' and PCI_CLK_in'event) THEN
                        state <= nextstate;
                END IF;
        END PROCESS;

        PROCESS(state, PCI_NFRAME_in, PCI_NIRDY_in, ADD_REG_in, ADD_MEM_in, MEM_READY_in)
        BEGIN
        ndevsel    <= '1';
        ntrdy <= '1';
        CASE state IS
            WHEN 0 =>
                IF(PCI_NFRAME_in = '0') THEN
                    nextstate <= 1;
                ELSE
                    nextstate <= 0;
                END IF;
            WHEN 1 =>
                IF(ADD_REG_in = '0' and ADD_MEM_in = '0') THEN
                    nextstate <= 4;
                ELSE
                    nextstate <= 2;
                    ndevsel <= '0';
                END IF;
            WHEN 2 =>
                ndevsel <= '0';
                IF(MEM_READY_in = '1' or (ADD_REG_in = '1' and PCI_NIRDY_in = '0')) THEN
                    nextstate <= 3;
                    ntrdy <= '0';
                ELSE
                    nextstate <= 2;
                END IF;
            WHEN 3 =>
                IF(PCI_NFRAME_in = '1' and PCI_NIRDY_in = '0') THEN
                    nextstate <= 4;
                ELSE
                    nextstate <= 3;
                    ndevsel    <= '0';
                    ntrdy <= '0';
                END IF;
            WHEN 4 =>
                IF(PCI_NFRAME_in = '1') THEN
                    nextstate <= 0;
                ELSE
                    nextstate <= 4;
                end if;
        END CASE;
END PROCESS;

PROCESS(PCI_NRST_in, PCI_CLK_in, REGMEM_W_R_in, nextstate2, nextstate4)
```

```
BEGIN
        IF(PCI_NRST_in = '0') THEN
                DAT_OE_out <= '0';
        ELSIF(PCI_CLK_in = '1' and PCI_CLK_in'event) THEN
                IF(nextstate2 = '1' and REGMEM_W_R_in = '0') then
                        DAT_OE_out <= '1';
                ELSIF(nextstate4 = '1') THEN
                        DAT_OE_out <= '0';
                END IF;
        END IF;
END PROCESS;

PROCESS(PCI_NRST_in, PCI_CLK_in, REGMEM_W_R_in, nextstate3, nextstate4)
BEGIN
        IF(PCI_NRST_in = '0') THEN
                PAR_OE_out <= '0';
        ELSIF(PCI_CLK_in = '1' and PCI_CLK_in'event) THEN
                IF((nextstate3 = '1' or nextstate4 = '1' ) and REGMEM_W_R_in = '0') THEN
                        PAR_OE_out <= '1';
                ELSE
                        PAR_OE_out <= '0';
                END IF;
        END IF;
END PROCESS;

PROCESS(PCI_NRST_in, PCI_CLK_in, nextstate2, nextstate0)
BEGIN
        IF(PCI_NRST_in = '0') THEN
                targetOE <= '0';
        ELSIF(PCI_CLK_in = '1' and PCI_CLK_in'event) THEN
                IF(nextstate2 = '1') THEN
                        targetOE <= '1';
                ELSIF(nextstate0 = '1') THEN
                        targetOE <= '0';
                END IF;
        END IF;
END PROCESS;

PROCESS(PCI_NRST_in, PCI_CLK_in, ndevsel, ntrdy)
BEGIN
        IF(PCI_NRST_in = '0') THEN
                devsel <= '1';
```

```
                        trdy <= '1';
                ELSIF(PCI_CLK_in ='1' and PCI_CLK_in'event) THEN
                        devsel <= ndevsel;
                        trdy <= ntrdy;
                END IF;
        END PROCESS;
END behave;
```

B.2.4 PCI_DATA.vhd

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY PCI_DATA IS
PORT(
                PCI_CLK_in              : in std_logic;
                PCI_NRST_in             : in std_logic;
                PCI_AD_inout            : inout std_logic_vector(31 downto 0);
                DAT_OE_in               : in std_logic;
                MEM_LD_in               : in std_logic;
                MEM_REG_SW_in           : in std_logic;
                PAR_DAT_out             : out std_logic_vector(31 downto 0);
                MEM_DAT_in              : in std_logic_vector(31 downto 0);
                MEM_DAT_out             : out std_logic_vector(31 downto 0);
                REG_DAT_in              : in std_logic_vector(31 downto 0);
                REG_DAT_out             : out std_logic_vector(31 downto 0)
);
END PCI_DATA;

ARCHITECTURE behave OF PCI_DATA IS
        signal MEM_DAT          : std_logic_vector(31 downto 0);
        signal PCI_DAT          : std_logic_vector(31 downto 0);
BEGIN
        PROCESS(PCI_NRST_in, PCI_CLK_in, MEM_LD_in, MEM_DAT_in)
        BEGIN
                IF(PCI_NRST_in = '0') THEN
                        MEM_DAT <= (others => '1');
                ELSIF(PCI_CLK_in'event and PCI_CLK_in = '1') THEN
                        IF(MEM_LD_in = '1') THEN
                                MEM_DAT <= MEM_DAT_in;
                        END IF;
                END IF;
```

```
    END PROCESS;
    PCI_DAT        <= MEM_DAT WHEN (MEM_REG_SW_in = '1') ELSE REG_DAT_in;
    PCI_AD_inout   <= PCI_DAT WHEN (DAT_OE_in = '1') ELSE "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
    PAR_DAT_out    <= PCI_DAT;
    MEM_DAT_out    <= PCI_AD_inout;
    REG_DAT_out    <= PCI_AD_inout(31 downto 0);
END behave;
```

B.2.5 PCI_PAR.vhd

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY PCI_PAR IS
PORT(
        PCI_CLK_in          : in std_logic;
        PAR_DAT_in          : in std_logic_vector(31 downto 0);
        PCI_CBE_in          : in std_logic_vector(3 downto 0);
        PAR_OE_in           : in std_logic;
        PCI_PAR_out         : out std_logic
);
END PCI_PAR;

ARCHITECTURE behave OF PCI_PAR IS
        signal par          : std_logic;
BEGIN
    PROCESS(PCI_CLK_in)
    BEGIN
        IF (PCI_CLK_in = '1' and PCI_CLK_in'event) THEN
            par <= PAR_DAT_in(0) xor PAR_DAT_in(1) xor PAR_DAT_in(2) xor PAR_DAT_in(3) xor
                PAR_DAT_in(4) xor PAR_DAT_in(5) xor PAR_DAT_in(6) xor PAR_DAT_in(7) xor
                PAR_DAT_in(8) xor PAR_DAT_in(9) xor PAR_DAT_in(10) xor PAR_DAT_in(11) xor
                PAR_DAT_in(12) xor PAR_DAT_in(13) xor PAR_DAT_in(14) xor PAR_DAT_in(15) xor
                PAR_DAT_in(16) xor PAR_DAT_in(17) xor PAR_DAT_in(18) xor PAR_DAT_in(19) xor
                PAR_DAT_in(20) xor PAR_DAT_in(21) xor PAR_DAT_in(22) xor PAR_DAT_in(23) xor
                PAR_DAT_in(24) xor PAR_DAT_in(25) xor PAR_DAT_in(26) xor PAR_DAT_in(27) xor
                PAR_DAT_in(28) xor PAR_DAT_in(29) xor PAR_DAT_in(30) xor PAR_DAT_in(31) xor
                PCI_CBE_in(3) xor PCI_CBE_in(2) xor PCI_CBE_in(1) xor PCI_CBE_in(0);
        END IF;
    END PROCESS;
    PCI_PAR_out    <= par WHEN (PAR_OE_in = '1' ) ELSE 'Z';
END behave;
```

B.2.6 PCI_REG.vhd

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY PCI_REG IS
PORT(
        PCI_CLK_in          : in std_logic;
        PCI_NRST_in         : in std_logic;
        PCI_CBE_in          : in std_logic_vector(3 downto 0);
        ADD_in              : in std_logic_vector(7 downto 2);
        REG_DAT_in          : in std_logic_vector(31 downto 0);
        REG_DAT_out         : out std_logic_vector(31 downto 0);
        REG_WR_in           : in std_logic;
        REG_RD_in           : in std_logic;
        BAR_out             : out std_logic_vector(31 downto 15);
        MEM_EN_out          : out std_logic
);
END PCI_REG;

ARCHITECTURE behave OF PCI_REG IS
    CONSTANT VendorID               : std_logic_vector(15 downto 0)     := x"10EE";
    CONSTANT DeviceID               : std_logic_vector(15 downto 0) := x"A123";
    CONSTANT RevisionID             : std_logic_vector(7 downto 0) := x"02";
    CONSTANT ClassCode              : std_logic_vector(23 downto 0) := x"068000";
    CONSTANT SubsystemVendorID      : std_logic_vector(15 downto 0) := x"1AB0";
    CONSTANT SubsystemID            : std_logic_vector(15 downto 0) := x"0001";
    CONSTANT InterruptPin           : std_logic_vector(7 downto 0) := x"01";
    CONSTANT Min_Gnt                : std_logic_vector(7 downto 0) := x"00";
    CONSTANT Max_Lat                : std_logic_vector(7 downto 0) := x"00";

    signal CMD0_WE                  : std_logic;
    signal BAR3_WE                  : std_logic;
    signal BAR2_WE                  : std_logic;
    signal BAR1_WE                  : std_logic;
    signal INTREG0_WE               : std_logic;
    signal MEM_EN                   : std_logic;
    signal InterruptLine            : std_logic_vector(7 downto 0) := "00000000";
    signal BAR                      : std_logic_vector(31 downto 15) := "00000000000000000";

BEGIN
    CMD0_WE     <= '1' when (ADD_in(7 downto 2) = "000001" ) and REG_WR_in = '1' and
```

```
                PCI_CBE_in(0) = '0' else '0';
BAR3_WE         <= '1' when (ADD_in(7 downto 2) = "000100" ) and REG_WR_in = '1' and
                PCI_CBE_in(3) = '0' else '0';
BAR2_WE         <= '1' when (ADD_in(7 downto 2) = "000100" ) and REG_WR_in = '1' and
                PCI_CBE_in(2) = '0' else '0';
BAR1_WE         <= '1' when (ADD_in(7 downto 2) = "000100" ) and REG_WR_in = '1' and
                PCI_CBE_in(1) = '0' else '0';
INTREG0_WE      <= '1' when (ADD_in(7 downto 2) = "001111" ) and REG_WR_in = '1' and
                PCI_CBE_in(0) = '0' else '0';
MEM_EN_out      <= MEM_EN;
BAR_out         <= BAR;

PROCESS(PCI_CLK_in, PCI_NRST_in, CMD0_WE, REG_DAT_in)
BEGIN
    IF(PCI_NRST_in = '0') THEN
        MEM_EN <= '0';
    ELSIF(PCI_CLK_in = '1' and PCI_CLK_in'event) THEN
        IF(CMD0_WE = '1') THEN
            MEM_EN <= REG_DAT_in(1);
        END IF;
    END IF;
END PROCESS;

PROCESS(PCI_CLK_in, PCI_NRST_in, INTREG0_WE, REG_DAT_in)
BEGIN
    IF(PCI_NRST_in = '0') THEN
        InterruptLine <= (others => '0');
    ELSIF(PCI_CLK_in = '1' and PCI_CLK_in'event) THEN
        IF(INTREG0_WE = '1') THEN
            InterruptLine <= REG_DAT_in(7 downto 0);
        END IF;
    END IF;
END PROCESS;

PROCESS(PCI_CLK_in, PCI_NRST_in, BAR3_WE, BAR2_WE, BAR1_WE, REG_DAT_in)
BEGIN
    IF(PCI_NRST_in = '0') THEN
        BAR <= ( others => '1' );
    ELSIF(PCI_CLK_in = '1' and PCI_CLK_in'event) THEN
        IF(BAR3_WE = '1') THEN
            BAR(31 downto 24) <= REG_DAT_in(31 downto 24);
        END IF;
```

```
            IF(BAR2_WE = '1') THEN
                BAR(23 downto 16) <= REG_DAT_in(23 downto 16);
            END IF;
            IF(BAR1_WE = '1') THEN
                BAR(15) <= REG_DAT_in(15);
            END IF;
        END IF;
    END PROCESS;

    PROCESS(ADD_in, MEM_EN, BAR, InterruptLine, REG_RD_in)
    BEGIN
        IF (REG_RD_in = '1') THEN
            CASE ADD_in IS
                WHEN "000000" => REG_DAT_out <= DeviceID & VendorID;
                WHEN "000001" => REG_DAT_out <= "0000000000000000" &
                                 "00000000000000" & MEM_EN & '0';
                WHEN "000010" => REG_DAT_out <= ClassCode & RevisionID;
                WHEN "000100" => REG_DAT_out <= BAR & b"000000000000000";
                WHEN "001011" => REG_DAT_out <= SubsystemID & SubsystemVendorID;
                WHEN "001111" => REG_DAT_out <= Max_Lat & Min_Gnt & InterruptPin &
                                 InterruptLIne;
                WHEN others   => REG_DAT_out <= (others => '0');
            END CASE;

        ELSE
            REG_DAT_out <= (others => '0');
        END IF;
    END PROCESS;
END behave;
```

B.3 Memory

B.3.1 MEM.vhd

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY MEM IS
PORT(
		CLK_in				: in std_logic;
		MIZZOU_ADD_in		: in std_logic_vector(14 downto 2);
		MIZZOU_DATA_inout	: inout std_logic_vector(31 downto 0);
		MIZZOU_READ_in		: in std_logic;
		MIZZOU_WRITE_in		: in std_logic;
		PCI_W_R_in			: in std_logic;
		PCI_SEL_in			: in std_logic;
		PCI_ADD_in			: in std_logic_vector(14 downto 2);
		PCI_DATA_in			: in std_logic_vector(31 downto 0);
		PCI_DATA_out		: out std_logic_vector(31 downto 0);
		DISPLAY				: out std_logic_vector(15 downto 0);
		MEM_READY_out		: out std_logic
);
END ENTITY;

ARCHITECTURE behave OF MEM IS
COMPONENT MEM_RAM IS
PORT(
	CLK_in			: in std_logic;
	MEM_WRITE_in	: in std_logic;
	MEM_READ_in		: in std_logic;
	MEM_ADD_in		: in std_logic_vector(13 downto 2);
	MEM_DATA_in		: in std_logic_vector(31 downto 0);
	MEM_DATA_out	: out std_logic_vector(31 downto 0)
);
END COMPONENT;

COMPONENT MEM_ROM IS
PORT(
	MEM_READ_in		: in std_logic;
	MEM_ADD_in		: in std_logic_vector(7 downto 2);
	MEM_DATA_out	: out std_logic_vector(31 downto 0)
);
```

```
END COMPONENT;

COMPONENT MEM_REG IS
PORT(
        CLK_in                  : in std_logic;
        MEM_WRITE_in            : in std_logic;
        MEM_READ_in             : in std_logic;
        MEM_ADD_in              : in std_logic_vector(6 downto 2);
        MEM_DATA_in             : in std_logic_vector(31 downto 0);
        MEM_DATA_out            : out std_logic_vector(31 downto 0);
        DISPLAY                 : out std_logic_vector(15 downto 0)
);
END COMPONENT;

        signal mizzouramread            : std_logic;
        signal mizzouramwrite           : std_logic;
        signal mizzouregread            : std_logic;
        signal mizzouregwrite           : std_logic;
        signal pciramread               : std_logic;
        signal pciramwrite              : std_logic;
        signal pciregread               : std_logic;
        signal pciregwrite              : std_logic;
        signal romread                  : std_logic;
        signal regread                  : std_logic;
        signal regwrite                 : std_logic;
        signal ramread                  : std_logic;
        signal ramwrite                 : std_logic;
        signal ramadd                   : std_logic_vector(13 downto 2);
        signal regadd                   : std_logic_vector(6 downto 2);
        signal romdata                  : std_logic_vector(31 downto 0);
        signal ramdata_in               : std_logic_vector(31 downto 0);
        signal ramdata_out              : std_logic_vector(31 downto 0);
        signal regdata_in               : std_logic_vector(31 downto 0);
        signal regdata_out              : std_logic_vector(31 downto 0);

BEGIN
        Mizzouramread           <= '1' WHEN MIZZOU_ADD_in(14) = '1' and MIZZOU_READ_in = '1' ELSE '0';
        mizzouramwrite          <= '1' WHEN MIZZOU_ADD_in(14) = '1' and MIZZOU_WRITE_in = '1' ELSE '0';
        romread                 <= '1' WHEN MIZZOU_ADD_in(14 downto 8) = "0000000" and
                                MIZZOU_READ_in = '1' ELSE '0';
        mizzouregread           <= '1' WHEN MIZZOU_ADD_in(14 downto 7) = "01111111" and
                                MIZZOU_READ_in = '1' ELSE '0';
        mizzouregwrite      <= '1' WHEN MIZZOU_ADD_in(14 downto 7) = "01111111" and
                            MIZZOU_WRITE_in = '1' ELSE '0';
        pciramread          <= '1' WHEN PCI_ADD_in(14) = '1' and PCI_W_R_in = '0' and PCI_SEL_in = '1'
                            ELSE '0';
        pciramwrite         <= '1' WHEN PCI_ADD_in(14) = '1' and PCI_W_R_in = '1' and PCI_SEL_in = '1'
                            ELSE '0';
        pciregread          <= '1' WHEN PCI_ADD_in(14 downto 7) = "01111111" and PCI_W_R_in = '0'
                            and PCI_SEL_in = '1' ELSE '0';
        pciregwrite         <= '1' WHEN PCI_ADD_in(14 downto 7) = "01111111" and PCI_W_R_in = '1'
                            and PCI_SEL_in = '1' ELSE '0';
        ramread             <= mizzouramread or pciramread;
        ramwrite            <= mizzouramwrite or pciramwrite;
        regread             <= mizzouregread or pciregread;
        regwrite            <= mizzouregwrite or pciregwrite;
        ramadd    <= PCI_ADD_in(13 downto 2) WHEN (pciramread = '1' or pciramwrite = '1')
            ELSE MIZZOU_ADD_in(13 downto 2) WHEN (mizzouramread = '1' or mizzouramwrite = '1')
            ELSE "ZZZZZZZZZZZZ";
        regadd    <= PCI_ADD_in(6 downto 2) WHEN (pciregread = '1' or pciregwrite = '1')
            ELSE MIZZOU_ADD_in(6 downto 2) WHEN (mizzouregread = '1' or mizzouregwrite = '1')
            ELSE "ZZZZZ";
        ramdata_in          <= MIZZOU_DATA_inout WHEN mizzouramwrite = '1' ELSE
                            PCI_DATA_in WHEN pciramwrite = '1' ELSE
                            "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
        regdata_in          <= MIZZOU_DATA_inout WHEN mizzouregwrite = '1' ELSE
                            PCI_DATA_in WHEN pciregwrite = '1' ELSE
                            "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
        MIZZOU_DATA_inout       <= romdata WHEN romread = '1' ELSE
                                ramdata_out WHEN mizzouramread = '1' ELSE
                                regdata_out WHEN mizzouregread = '1' ELSE
                                "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
        PCI_DATA_out            <= ramdata_out WHEN pciramread = '1' ELSE
                                regdata_out WHEN pciregread = '1' ELSE
                                "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
        MEM_READY_out           <= '1' WHEN PCI_SEL_in = '1' or MIZZOU_READ_in = '1' or
                                MIZZOU_WRITE_in = '1' ELSE '0';
M1: COMPONENT MEM_RAM PORT MAP (CLK_in, ramwrite, ramread, ramadd, ramdata_in,
ramdata_out);
M2: COMPONENT MEM_ROM PORT MAP (romread, MIZZOU_ADD_in(7 downto 2), romdata);
M3: COMPONENT MEM_REG PORT MAP (CLK_in, regwrite, regread, regadd, regdata_in, regdata_out,
DISPLAY);
END behave;
```

B.3.2 MEM_RAM.vhd

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY MEM_RAM IS
PORT(
        CLK_in              : in std_logic;
        MEM_WRITE_in        : in std_logic;
        MEM_READ_in         : in std_logic;
        MEM_ADD_in          : in std_logic_vector(13 downto 2);
        MEM_DATA_in         : in std_logic_vector(31 downto 0);
        MEM_DATA_out        : out std_logic_vector(31 downto 0)
);
END MEM_RAM;

ARCHITECTURE behave OF MEM_RAM IS
    TYPE ramtype IS array(0 to 4095) OF std_logic_vector(31 downto 0);
    signal RAM : ramtype:=(others => x"00000000");
BEGIN
    MEM_DATA_out <= RAM(conv_integer(MEM_ADD_in(13 downto 2))) WHEN MEM_READ_in='1'
            ELSE "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
    PROCESS(CLK_in, MEM_WRITE_in)
    BEGIN
        IF CLK_in'event and CLK_in='1' THEN
            IF MEM_WRITE_in='1' THEN
                RAM(conv_integer(MEM_ADD_in(13 downto 2))) <= MEM_DATA_in;
            END IF;
        END IF;
    END PROCESS;
END behave;
```

B.3.3 MEM_REG.vhd

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY MEM_REG IS
```

```
PORT(
            CLK_in              : in std_logic;
            MEM_WRITE_in        : in std_logic;
            MEM_READ_in         : in std_logic;
            MEM_ADD_in          : in std_logic_vector(6 downto 2);
            MEM_DATA_in         : in std_logic_vector(31 downto 0);
            MEM_DATA_out        : out std_logic_vector(31 downto 0);
            DISPLAY             : out std_logic_vector(15 downto 0)
);
END MEM_REG;

ARCHITECTURE behave OF MEM_REG IS
      TYPE regtype IS array(0 to 31) OF std_logic_vector(31 downto 0);
      signal REG : regtype := (others => x"00000000");
BEGIN
      DISPLAY <= REG(31)(15 downto 0);
      MEM_DATA_out <= REG(conv_integer(MEM_ADD_in(6 downto 2))) WHEN MEM_READ_in = '1'
                  ELSE "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
      PROCESS(CLK_in, MEM_WRITE_in)
      BEGIN
            IF CLK_in'event and CLK_in = '1' THEN
                  IF    MEM_WRITE_in = '1' THEN
                        REG(conv_integer(MEM_ADD_in(6 downto 2))) <= MEM_DATA_in;
                  END IF;
            END IF;
      END PROCESS;
END behave;
```

B.3.4 MEM_ROM.vhd

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY MEM_ROM IS
PORT(
            MEM_READ_in               : in std_logic;
            MEM_ADD_in                : in std_logic_vector(7 downto 2);
            MEM_DATA_out              : out std_logic_vector(31 downto 0)
);
END MEM_ROM;
```

```
ARCHITECTURE behave OF MEM_ROM IS
	TYPE romtype IS array (0 to 63) OF std_logic_vector(31 downto 0);
	CONSTANT ROM	: romtype:=
	(
		x"CF000004",
		x"CF000004",
		x"CFFFFFFC",
		x"8F103F80",
		x"81004000",
		x"82000004",
		x"5EF10000",
		x"8300FFFF",
		x"8420FFFF",
		x"55340000",
		x"601E0000",
		x"CC000018",
		x"80510000",
		x"63050000",
		x"93510000",
		x"53120000",
		x"81430000",
		x"CF3FFFE0",
		x"81003FFC",
		x"83005050",
		x"93510000",
		x"FF000000",
		others=>x"00000000");
BEGIN
	MEM_DATA_out  <=  ROM(conv_integer(MEM_ADD_in(7  downto  2)))  WHEN  MEM_READ_in='1'
ELSE "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
END behave;
```

B.4 7-segment LEDs

B.4.1 DISP_CTL.vhd

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
ENTITY DISP_CTL IS
PORT(
        CLK_in              : in std_logic;
        DISPLAY             : in std_logic_vector(15 downto 0);
        DISP_SEL            : inout std_logic_vector(3 downto 0);
        DISP_LED            : out std_logic_vector(6 downto 0)
);
END DISP_CTL;

ARCHITECTURE behave OF DISP_CTL IS
COMPONENT DISP_DEC IS
PORT(
    DISP_DEC_in             : in std_logic_vector(3 downto 0);
    DISP_DEC_out            : out std_logic_vector(6 downto 0)
);
END COMPONENT;

    signal DISP_CNT             : std_logic_vector(6 downto 0):="0000000";
    signal DISP_DATA            : std_logic_vector(3 downto 0):="0000";
    signal DISP_DATA_LED        : std_logic_vector(6 downto 0);
    signal DISP_POS             : std_logic_vector(3 downto 0):="0001";
    CONSTANT DISP_CNT_MAX       : std_logic_vector(6 downto 0):= "1111111";

BEGIN
    PROCESS(CLK_in)
    BEGIN
        IF CLK_in'event and CLK_in = '1' THEN
            DISP_CNT <= DISP_CNT + 1;
        END IF;
    END PROCESS;

    PROCESS(CLK_in)
    BEGIN
        IF CLK_in'event and CLK_in = '1' THEN
            IF DISP_CNT = DISP_CNT_MAX THEN
                DISP_POS <= DISP_POS(2 downto 0) & DISP_POS(3);
                DISP_SEL <= DISP_POS;
            END IF;
        END IF;
    END PROCESS;

    PROCESS(CLK_in)
    BEGIN
        IF CLK_in'event and CLK_in = '1' THEN
            CASE DISP_SEL IS
                WHEN "1000" =>
                    DISP_DATA <= DISPLAY(3 downto 0);
                WHEN "0100" =>
                    DISP_DATA <= DISPLAY(7 downto 4);
                WHEN "0010" =>
                    DISP_DATA <= DISPLAY(11 downto 8);
                WHEN "0001" =>
                    DISP_DATA <= DISPLAY(15 downto 12);
                WHEN OTHERS =>
                    DISP_DATA <= (others => '0');
            END CASE;
        END IF;
    END PROCESS;

    u1: COMPONENT DISP_DEC PORT MAP (DISP_DATA, DISP_DATA_LED);

    PROCESS(CLK_in)
    BEGIN
        IF CLK_in'event and CLK_in = '1' then
            DISP_LED <= DISP_DATA_LED;
        END IF;
    END PROCESS;
END behave;
```

B.4.2 DISP_DEC.vhd

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY DISP_DEC IS
PORT(
        DISP_DEC_in             : in std_logic_vector(3 downto 0);
        DISP_DEC_out            : out std_logic_vector(6 downto 0)
);
END DISP_DEC;

ARCHITECTURE behave OF DISP_DEC IS
```

```
BEGIN
    PROCESS(DISP_DEC_in)
    BEGIN
        CASE DISP_DEC_in IS
            WHEN "0000" =>
                DISP_DEC_out <= "1000000";
            WHEN "0001" =>
                DISP_DEC_out <= "1111001";
            WHEN "0010" =>
                DISP_DEC_out <= "0100100";
            WHEN "0011" =>
                DISP_DEC_out <= "0110000";
            WHEN "0100" =>
                DISP_DEC_out <= "0011001";
            WHEN "0101" =>
                DISP_DEC_out <= "0010010";
            WHEN "0110" =>
                DISP_DEC_out <= "0000010";
            WHEN "0111" =>
                DISP_DEC_out <= "1111000";
            WHEN "1000" =>
                DISP_DEC_out <= "0000000";
            WHEN "1001" =>
                DISP_DEC_out <= "0010000";
            WHEN "1010" =>
                DISP_DEC_out <= "0001000";
            WHEN "1011" =>
                DISP_DEC_out <= "0000011";
            WHEN "1100" =>
                DISP_DEC_out <= "1000110";
            WHEN "1101" =>
                DISP_DEC_out <= "0100001";
            WHEN "1110" =>
                DISP_DEC_out <= "0000110";
            WHEN "1111" =>
                DISP_DEC_out <= "0001110";
            WHEN others =>
                DISP_DEC_out <= "1111111";
        END CASE;
    END PROCESS;
END behave;
```

Appendix C – C Program Controller

C.1 Function for the C Program Controller

```
static void MyWrite(WDC_DEVICE_HANDLE hDev)
{
    DWORD mydwAddrSpace = ACTIVE_ADDR_SPACE_NEEDS_INIT;
    WDC_ADDR_MODE mymode = WDC_MODE_32;
    BOOL myfBlock = TRUE;
    WDC_DIRECTION direction =    WDC_WRITE;
    DWORD mydwOffset = 16384;
    DWORD mydatanum = 16384;
    DWORD cycletime = 0;
    DWORD regOffset = 16380;
    DWORD sddwOffset =16256;
    DWORD mydwSize = 256;
    DWORD u32Data;
    PVOID pBuf = NULL;
    PVOID myresultpBuf = NULL;
    PVOID tempmypBuf = NULL;
    PVOID tempmyresultpBuf=NULL;
    unsigned char c;
    long int length ,width,counts=0;
    long int filelength=0;
    int i,j,k;
    long int filelengthtemp,figuretemp;
    DWORD figuresize;
    unsigned char gfl1,gfl2,gfl3,gfl4,bfl1,bfl2,bfl3,bfl4,gff1,gff2,gff3,gff4,bff1,bff2,bff3,bff4;
    int r=0;
    int g=0;
    int b=0;
    int gray=0;
    int blackw=0;
    struct node{
                unsigned char r;
                unsigned char g;
                unsigned char b;
            }data[1600][900];
    char gnode[1600][900];
    int blacklength;
    int HH,VV,LL,RR;
    int CC=2;
    int thres=100;
    PVOID mypBuf = NULL;
```

```
PBYTE mypData;
PBYTE myresultpData;
DWORD mydwStatus;
WDC_ADDR_MODE mode = WDC_MODE_32;
WDC_ADDR_RW_OPTIONS options = WDC_ADDR_RW_DEFAULT;

FILE *fin=fopen("C:\\Users\\eldercare\\Desktop\\deliver\\test128k.bmp","rb");
FILE *fout1=fopen("C:\\Users\\eldercare\\Desktop\\deliver\\test8kgray.bmp","wb");
FILE *fout2=fopen("C:\\Users\\eldercare\\Desktop\\deliver\\test8kbw.bmp","wb");

for(i=0;i<0x20;i++)
{
        fscanf(fin,"%c",&c);
        if(i==0x2)filelength=(int)c;
        if(i==0x3)filelength+=(int)c*256;
        if(i==0x4)filelength+=(int)c*256*256;
        if(i==0x5)filelength+=(int)c*256*256*256;
        if(i==0x12)width=(int)c;
        if(i==0x13)width+=(int)c*256;
        if(i==0x14)width+=(int)c*256*256;
        if(i==0x15)width+=(int)c*256*256*256;
        if(i==0x16)length=(int)c;
        if(i==0x17)length+=(int)c*256;
        if(i==0x18)length+=(int)c*256*256;
        if(i==0x19)length+=(int)c*256*256*256;
}
        rewind(fin);
        filelengthtemp=(filelength-54)/3+1024+54;
        gfl1=(char)(filelengthtemp%256);
        filelengthtemp=filelengthtemp/256;
        gfl2=(char)(filelengthtemp%256);
        filelengthtemp=filelengthtemp/256;
        gfl3=(char)(filelengthtemp%256);
        filelengthtemp=filelengthtemp/256;
        gfl4=(char)(filelengthtemp%256);
        filelengthtemp=(filelength-54)/24+54+8;
        bfl1=(char)(filelengthtemp%256);
        filelengthtemp=filelengthtemp/256;
        bfl2=(char)(filelengthtemp%256);
        filelengthtemp=filelengthtemp/256;
        bfl3=(char)(filelengthtemp%256);
        filelengthtemp=filelengthtemp/256;
```

```
    bfl4=(char)(filelengthtemp%256);
    figuretemp=length*width;
    gff1=(char)(figuretemp%256);
    figuretemp=figuretemp/256;
    gff2=(char)(figuretemp%256);
    figuretemp=figuretemp/256;
    gff3=(char)(figuretemp%256);
    figuretemp=figuretemp/256;
    gff4=(char)(figuretemp%256);
    figuretemp=length*width/8;
    bff1=(char)(figuretemp%256);
    figuretemp=figuretemp/256;
    bff2=(char)(figuretemp%256);
    figuretemp=figuretemp/256;
    bff3=(char)(figuretemp%256);
    figuretemp=figuretemp/256;
    bff4=(char)(figuretemp%256);
for(i=0;i<0x36;i++)
{
    fscanf(fin,"%c",&c);
    switch(i){
    case 0x2:
        fprintf(fout1,"%c",gfl1);
        fprintf(fout2,"%c",bfl1);
    break;
    case 0x3:
        fprintf(fout1,"%c",gfl2);
        fprintf(fout2,"%c",bfl2);
    break;
    case 0x4:
        fprintf(fout1,"%c",gfl3);
        fprintf(fout2,"%c",bfl3);
    break;
    case 0x5:
        fprintf(fout1,"%c",gfl4);
        fprintf(fout2,"%c",bfl4);
    break;
    case 0xA:
        fprintf(fout1,"%c",c);
        c=(char)0x3E;
        fprintf(fout2,"%c",c);
    break;
    }
        case 0xB:
            fprintf(fout2,"%c",c);
            c=(char)0x4;
            fprintf(fout1,"%c",c);
        break;
        case 0x1C:
            c=(char)0x8;
            fprintf(fout1,"%c",c);
            c=(char)0x1;
            fprintf(fout2,"%c",c);
        break;
        case 0x22:
            fprintf(fout1,"%c",gff1);
            fprintf(fout2,"%c",bff1);
        break;
        case 0x23:
            fprintf(fout1,"%c",gff2);
            fprintf(fout2,"%c",bff2);
        break;
        case 0x24:
            fprintf(fout1,"%c",gff3);
            fprintf(fout2,"%c",bff3);
        break;
        case 0x25:
            fprintf(fout1,"%c",gff4);
            fprintf(fout2,"%c",bff4);
        break;
        case 0x2E:
            fprintf(fout1,"%c",c);
            c=(char)0x2;
            fprintf(fout2,"%c",c);
        break;
        case 0x2F:
            fprintf(fout2,"%c",c);
            c=(char)0x1;
           fprintf(fout1,"%c",c);
        break;
        default:
            {fprintf(fout1,"%c",c);
            fprintf(fout2,"%c",c);}
        }
    }
```

```
for(i=0; i<256; i++)
{
    fprintf(fout1,"%c",(char)(i));
    fprintf(fout1,"%c",(char)(i));
    fprintf(fout1,"%c",(char)(i));
    fprintf(fout1,"%c",(char)(0));
    fprintf(fout2,"%c",(char)(i));
    fprintf(fout2,"%c",(char)(i));
    fprintf(fout2,"%c",(char)(i));
    fprintf(fout2,"%c",(char)(0));
}
    fprintf(fout2,"%c",(char)(0));
    fprintf(fout2,"%c",(char)(0));
    fprintf(fout2,"%c",(char)(0));
    fprintf(fout2,"%c",(char)(0));
    fprintf(fout2,"%c",(char)(255));
    fprintf(fout2,"%c",(char)(255));
    fprintf(fout2,"%c",(char)(255));
    fprintf(fout2,"%c",(char)(0));*/

if (width*length%8>0){
    blacklength=1+width*length/8;}
else{
    blacklength=width*length/8;}

figuresize=width*length;
for(i=0;i<length;i++)
{
    for(j=0;j<width;j++){
        fscanf(fin,"%c",&data[j][i].r);
        fscanf(fin,"%c",&data[j][i].g);
        fscanf(fin,"%c",&data[j][i].b);
    }
}

mypBuf = malloc(figuresize);
myresultpBuf = malloc(figuresize);
memset(mypBuf, 0, figuresize);
memset(myresultpBuf, 0, figuresize);
mypData = (PBYTE)mypBuf;
myresultpData = (PBYTE)myresultpBuf;
```

```
for(i=0;i<length;i++)
{
        for(j=0;j<width;j++)
        {
        mypData[i*width+j]=(BYTE)((int)data[j][i].r*0.33+(int)data[j][i].g*0.33+(int)data[j][i].b*0.33);
         fprintf(fout1,"%c",mypData[i*width+j]);
         fprintf(fout2,"%c",mypData[i*width+j]);
        }
}

if (ACTIVE_ADDR_SPACE_NEEDS_INIT == mydwAddrSpace)
{
         DWORD mydwNumAddrSpaces = MIZZOU_GetNumAddrSpaces(hDev);

        /* Find the first active address space */
         for (mydwAddrSpace = 0; mydwAddrSpace < mydwNumAddrSpaces; mydwAddrSpace++)
        {
                 if (WDC_AddrSpaceIsActive(hDev, mydwAddrSpace))
                         break;
        }

        /* Sanity check */
        if (mydwAddrSpace == mydwNumAddrSpaces)
        {
                MIZZOU_ERR("MenuReadWriteAddr: Error - no active address spaces found\n");
                mydwAddrSpace = ACTIVE_ADDR_SPACE_NEEDS_INIT;
                return;
        }
}
if (mydatanum>figuresize)        mydatanum=figuresize;

mydwStatus = WDC_WriteAddr32(hDev, mydwAddrSpace, sddwOffset, mydatanum);

if (WD_STATUS_SUCCESS == mydwStatus)
         printf("MizzouRisc will process %d byte data every time\n" , mydatanum);

if (figuresize%mydatanum==0)
         cycletime=figuresize/mydatanum;
else
         cycletime=figuresize/mydatanum+1;

for (k=0;k<cycletime;k++)
```

```
{
        tempmypBuf = &mypData[mydatanum*k];
        tempmyresultpBuf = &myresultpData[mydatanum*k];
        if (figuresize%mydatanum>0 && k==cycletime-1)
        {
                mydatanum=figuresize%mydatanum;
                mydwStatus = WDC_WriteAddr32(hDev, mydwAddrSpace, sddwOffset, mydatanum);
                if (WD_STATUS_SUCCESS == mydwStatus)
                        printf("MizzouRisc will process %d byte data every time\n" , mydatanum);
        }
        printf("Part %d data are writing to the MizzouRisc\n" , k);
        mydwStatus = WDC_WriteAddrBlock(hDev, mydwAddrSpace, mydwOffset, mydatanum,
        tempmypBuf, mode, options);

        if (WD_STATUS_SUCCESS == mydwStatus)
                printf("Part %d Data write is done\n" , k);

        mydwStatus = MIZZOU_IntEnable(hDev, DiagIntHandler);

        if (WD_STATUS_SUCCESS == mydwStatus)
                printf("Listing to the Interrupt\n");

        u32Data=61680;
        mydwStatus = WDC_WriteAddr32(hDev, mydwAddrSpace, regOffset, u32Data);

        if (WD_STATUS_SUCCESS == mydwStatus)
                printf("Write F0F0 to the register\n");

        while(1)
        {
                if (mylabel==1)
                        break;
        }
        mylabel=0;

        if (WD_STATUS_SUCCESS == MIZZOU_IntDisable(hDev))
                printf("Interrupts disabled\n");

        printf("Part %d data are reading from the MizzouRisc\n" , k);
        mydwStatus = WDC_ReadAddrBlock(hDev, mydwAddrSpace, mydwOffset, mydatanum,
        tempmyresultpBuf, mode, options);
        if (WD_STATUS_SUCCESS == mydwStatus)
            printf("Part %d Data reading is done\n" , k);

        u32Data=45232;
        mydwStatus = WDC_WriteAddr32(hDev, mydwAddrSpace, regOffset, u32Data);

        if (WD_STATUS_SUCCESS == mydwStatus)
            printf("Write B0B0 to the register\n");
    }
    printf("Output the figure!\n");


    for(i=0;i<figuresize;i++)
    {
        fprintf(fout2,"%c",myresultpData[i]);
    }

    fclose(fout1);
    fclose(fout2);
    fclose(fin);

    free(mypBuf);
    free(myresultpBuf);
}
```

Appendix D – Signal Relationship for the Firmware

D.1 Signal Relationship in the Entity System

	System		mizzou_risc		MEM		PCI_MOD		DISP_CTL	
	input	output	input	output	input	output	input	output	input	output
PCI_CLK	x		x		x		x		x	
PCI_NRST	x									
PCI_AD	x BUS	x BUS					x BUS	x BUS		
PCI_CBE	x	x					x			
PCI_PAR		x						x		
PCI_NFRAME	x						x			
PCI_NIRDY	x						x			
PCI_NTRDY		x						x		
PCI_NDEVSEL		x						x		
PCI_NSTOP		x						x		
PCI_IDSEL	x						x			
PCI_NPERR	x	x						x		
PCI_NSERR	x	x						x		
PCI_NINT		x						x		
PCI_NREQ	x									
PCI_NGNT	x									
BUT_RST	x									
BUT_NRST	x									
DISP_SEL	x	x								x
DISP_LED		x								x
DISP_DOT		x								
LED_MHALT		x		x						
LED_PCI		x								
LED_ALIVE		x								
rst			x							
mizzouadd			x BUS	x BUS	x BUS					
mizzoudata			x BUS	x BUS	x BUS	x BUS				
priority			x		x BUS	x BUS				
int_ack				x						
mizzouread				x	x					
mizzouwrite				x	x					
PCI_W_R					x			x		
PCI_SEL					x			x		
PCI_ADD					x BUS			x BUS		
PCI_DAT_B					x BUS			x BUS		
PCI_DAT_A						x BUS	x BUS			
DISPLAY						x			x	
MEM_READY					x			x		

	System		mizzou_risc		MEM		PCI_MOD		DISP_CTL	
	input	output	input	output	input		input	output	input	output
nrst							x			
SYS_INT							x			

D.2 Signal Relationship in Entity PCI_MOD

	PCI_MOD		PCI_CTL		PCI_ADD		PCI_DATA		PCI_PAR		PCI_REG	
	Input	Output	Input	Output	Input	Output	Input	Output	Input	Output	Input	Output
PCI_CLK	x		x		x		x		x		x	
PCI_NRST	x		x		x		x				x	
PCI_AD	x BUS	x BUS			x BUS		x BUS	x BUS				
PCI_CBE	x				x				x		x	
PCI_PAR		x								x		
PCI_NFRAME	x		x									
PCI_NIRDY	x		x									
PCI_NTRDY		x		x								
PCI_NDEVSEL		x		x								
PCI_IDSEL	x				x							
PCI_NINT		x										
PCI_NSTOP		x										
PCI_NPERR		x										
PCI_NSERR		x										
SYS_INT	x											
MEM_READY	x		x									
MEM_SEL		x		x								
MEM_W_R		x										
MEM_ADD		x BUS				x BUS					x BUS	
MEM_DAT_in	x BUS						x BUS					
MEM_DAT_out		x BUS						x BUS				
REGMEM_W_R			x			x						
ADD_REG			x			x						
ADD_MEM			x			x						
ADD_LD				x	x							
DAT_OE				x			x					
PAR_OE				x					x			
MEM_LD				x			x					
MEM_REG_SW				x			x					
REG_WR				x							x	
REG_RD				x							x	
BAR					x BUS							x BUS
MEM_EN					x							x
PAR_DAT								x BUS	x BUS			
REG_DAT_A							x BUS					x BUS
REG_DAT_B								x BUS			x BUS	

D.3 Signal Relationship in Entity MEM

	MEM		MEM_RAM		MEM_ROM		MEM_REG	
	Input	Output	Input	Output	Input	Output	Input	Output
CLK	x		x				x	
MIZZOU_ADD	x BUS							
MIZZOU_DATA	x BUS	x BUS						
MIZZOU_READ	x							
MIZZOU_WRITE	x							
PCI_W_R	x							
PCI_SEL	x							
PCI_ADD	x BUS							
PCI_DATA_in	x BUS							
PCI_DATA_out		x BUS						
DISPLAY		x						x
MEM_READY		x						
MEM_WRITE			x				x	
MEM_READ			x		x		x	
MEM_ADD			x BUS		x BUS		x BUS	
MEM_DATA_in			x BUS				x BUS	
MEM_DATA_out				x BUS		x BUS		x BUS

D.4 Signal Relationship in Entity DISP_CTL

	DISP_CTL		DISP_DEC	
	Input	Output	Input	Output
CLK	x			
DISPLAY	x			
DISP_SEL		x		
DISP_LED		x		
DISP_DATA			x	
DISP_DATA_LED				x

Appendix E – Assembly Program for the MizzouRISC

E.1 Assembly Program for the MizzouRISC

```
Address  Instructions  Label   Instruction              Comments
0000     CF000004              BRN (0)  WAIT            ; Wait for the interrupt
0004     CF000004              BRN (0)  BEGIN           ; Begin program
0008     CFFFFFFC      WAIT    BRN (3)  WAIT            ; Wait for the interrupt
000C     8F103F80      BEGIN   LOAD     (L#3F80), R15   ; Save SIZE in R15
0010     81004000              LOAD     L#4000, R1      ; Start address of data
0014     82000004              LOAD     L#0004, R2      ; Set R2 to “4”
0018     5EF10000              ADD      R1, R15, R14    ; End address of data
001C     8300FFFF              LOAD     L#FFFC, R3      ; Set R3 to “0000FFFF”
0020     8420FFFF              LOAD     H#FFFF, R4      ; Set R4 to “FFFF0000”
0024     55340000              ADD      R4, R3, R5      ; All 1 (“FFFFFFFF”)
0028     601E0000      LOOP    SUB      R14, R1, R0     ; Change loop counter
002C     CC000018              BRN (EQ) CRG             ; End LOOP
0030     80510000              LOAD     (#R1), R0       ; Load data
0034     63050000              SUB      R5, R0, R3      ; Invert color
0038     93510000              STO      R3, (#R1)       ; Save data
003C     53120000              ADD      R2, R1, R3      ; Address of next data
0040     81430000              LOAD     R3, R1          ; Use R1 save address
0044     CF3FFFE0              BRN (0)  LOOP            ; Loop forever
0048     81003FFC      CRG     LOAD     L#3FFC, R1      ; Add of label register
0050     83005050              LOAD     L#5050, R3      ; Flag of process end
0054     93510000              STO      R3, (#R2)       ; set register
0058     FF000000              RTN (0)                  ; Return to wait
```