1 STATEMENT OF OBJECTIVES

- Learn how to upload the NIOS II processor onto the FPGA
- Learn how to program the CPU with assembly code
- Learn how to edit the registers and memory in Altera Monitor Program
- Learn how to use breakpoints in Altera Monitor Program
- Learn how to generate the Hex representation of an instruction

2 THEORETICAL BACKGROUND

The FPGA allows us to upload a custom logic circuit to the board, such as a CPU. Altera provides the NIOS II processor for exactly this purpose. In this lab, only the basic processor was used without any peripherals, but future labs can add on to the system.

3 EXPERIMENTAL PROCEDURE FOR PART I

3.1 Equipment Used

- Dell Inspiron 7559 (Laptop)
- Altera Monitor Program 15.1
- Cyclone V GX Starter Kit

3.2 Procedure for Part I

- Connect your PC to the USB Blaster on the FPGA
- Open Altera Monitor Program and create a new project
- Download the NIOS II system onto the FPGA
- Compile the code in Appendix A and upload it to the board
- Set the Program Counter to 0x0
- Place a break-point at address 0x28
- Run the program and record the value of r7 each time the break-point is reached
- Set the PC to 0x8 and r4 to 0x504
- Run the program and record the results

4 ANALYSIS FOR PART I

4.1 Experimental Results

Altera Monitor Program successfully compiled the program and uploaded it to the FPGA. When the program reached the break-point, r7 contained 0x7 the first time and 0x9 the second time. Then the program ended with the registers as shown in Figure **??** and memory as shown in Figure **??**. After changing the value of r4 to 0x504 and rerunning the program, the result was still 0x9, but it was stored at address 0x504 instead of 0x500 as shown in Figures **??** and **??**, respectively.

4.2 Data Analysis

The code would start by loading the number of elements in the array from 0x504. Then starting from 0x508, the program would load values from the array and store the largest value in r7. At the end of the array, it would store the largest value into 0x500. Since r4 held the address to the number of elements in the array, increasing it's value by four caused the entire memory access to be off by four bytes including writing back to 0x504 instead of 0x500.

5 EXPERIMENTAL PROCEDURE FOR PART II

5.1 Equipment Used

- Dell Inspiron 7559 (Laptop)
- Altera Monitor Program 15.1
- Cyclone V GX Starter Kit

5.2 Procedure for Part II

- Connect your PC to the USB Blaster on the FPGA
- Open Altera Monitor Program and create a new project
- Download the NIOS II system onto the FPGA
- Compile the code in Appendix A and upload it to the board
- Derive the Hex value for the instruction `blt r7, r8, LOOP` and place it into 0x24
- Run the program and explain what happened

6 ANALYSIS FOR PART II

6.1 Experimental Results

Altera Monitor Program successfully compiled the program and uploaded it to the FPGA. The instruction `blt r7, r8, LOOP` was found to be `0x3A3FFB16` and caused the program to store 0x0 into r7 and 0x500 instead of 0x9 as shown in Figures **??** and **??**, respectively.

6.2 Data Analysis

By changing the instruction `bge r7, r8, LOOP` into `blt r7, r8, LOOP`, the program kept track of the smallest value of the array instead of the largest value. Nothing else changed from Part I.

7 EXPERIMENTAL PROCEDURE FOR PART III

7.1 Equipment Used

- Dell Inspiron 7559 (Laptop)
- Altera Monitor Program 15.1
- Cyclone V GX Starter Kit

7.2 Procedure for Part III

- Connect your PC to the USB Blaster on the FPGA
- Open Altera Monitor Program and create a new project
- Download the NIOS II system onto the FPGA
- Modify the code in Appendix A to use an array of bytes instead of an array of words
- Compile the above program and upload it to the board
- Run the program and explain what happened

8 ANALYSIS FOR PART III

8.1 Experimental Results

Altera Monitor Program successfully compiled the program and uploaded it to the FPGA. Since all data was less than 256, it all fit into one byte. As such the result was still 0x9 as shown in Figures **??** and **??**, respectively.

8.2 Data Analysis

Since the only thing that changed was the data bit width, the data was stored at different addresses in memory. For example, `COUNT` was stored at 0x501 instead of 0x504 and the data started at 0x502 instead of 0x508. Nothing else changed from Part I. Please refer to Appendix B for the source code.

9 CONCLUSION

FPGA's are very versatile and can implement many peripherals into the same chip as the CPU to allow for more versatile systems and hardware accelerated applications in addition to traditional assembly or C code. In later we will be able to take advantage of this.