Michael Chan, Siddhanta Shrestha

ECE 332: Lab Report 1

**Introduction**

The goal of this lab was to give us the opportunity to explore the FPGA in addition to familiarizing ourselves with the environments necessary to program the board. The lab was split up into two parts; The first was a guided exploration with the objective to understand how the components of the board interact through their connections. The second was to expand upon this knowledge by applying it towards a more complex project. Together this lab served as an introduction to the FPGA and to illustrate the stark contrast from microcontrollers and microprocessors.

The objective of the first part of the lab was to configure the board to display “Hello, World” on the console everytime a switch transitioned from 0 to 1. This allowed us to learn about how a switch, which embodies an or gate, functions and how it serves as an output to the digitally connected console. Additionally, we learned how to program an external off-chip memory so that it can be used for both instruction and data memory. This is necessary so that the program can be stored in the instruction memory while the data memory allows variables to be stored. With these components the board is able to properly run the “Hello, World” script.

The objective of the second part of the lab was to configure the FPGA inorder to implement a counter displayed in both their corresponding decimal and binary values. These values would then be displayed on the seven segment display and LED display respectively (in addition to the console). Similar to the previous part, we established a communication between the switches and displays in order to ensure that the proper method of count was used based on the logic of the switches. While the console continuously displayed the counter, the seven seg and LED would only display the count value depending on which of the three switches had a logic high. As a result, we programmed the board to run a more complex project where multiple parts were communicating with each other.

**Detailed Procedure**

To begin the first part of the lab we created a new project in our *Lab1* directory. We then configured the project for the FPGA board. We then launched QSYS software so that we could program specific chips on the board. In QSYS we added necessary system components required for the “Hello, World small” project. Clock source was added so that the system could regulate the internal components by ensuring that all components are synchronized. The NIOS 2 processor, the board's CPU, is required so that it can provide instructions to components and processing power required for the work. The JTAG UART is necessary for us to display the message on the console. Finally, an on chip memory was added so that the system can handle the program. With all the components added, we then wired the connections so that they could communicate with each other. For example, the clocks of the components are all connected together, as are the resets. We then assigned the base addresses for the components before generating the HDL (QIP file written in the board’s prefered language Verilog). In Quartus we added the QIP file to the project which we then synthesized. To ensure that everything would work, we assigned the clock and reset pins with the board's 50 MHz oscillator (*PIN\_AF14*) and switch (*SW[x] = PIN\_AB12*). Upon completion, we then compiled the design so that we could send it to the board. Before doing so, we first needed to connect the board through the programmer. We changed the default file to the SOF file of the *5CSEMA5* device. Now the final thing left was to write the script so that the board would do what we intended it to do. This was simple because a template code was used from the NIOS II IDE upon creating the application. We simply had to make sure that we added the file path for the SOPC file so that the program would properly run using our design. We then made sure that switch 1 on the board was in the HIGH state before running the script. Now, everytime the switch transitioned from 0 to 1 the board would print “"Hello from Nios II!” on the console.

Starting the second part of the lab was very similar to the previous part. Upon creating a new project in our *Lab1* folder named *count\_binary* we added the necessary internal components in QSYS. A couple of changes from the previous that had to be made were the PIN assignments. For the PIN Assignments, we used the provided DE\_CoE.qsf file that was provided and uploaded it to Quartus. This allowed the corresponding pins to communicate with the FPGA board. A count\_binary\_top.v file was also added to instantiate the needed components. This verilog file was made as the top-level module.

**Hardware Changes**

In the first part of the lab there weren't many hardware changes necessary. For this lab we used the resource-optimized version of the NIOS II Processor. Additionally, for the On-Chip Memory we used a data width of 32 bits and a memory size of 4096 bytes. Finally, we selected the On-Chip memory for the Reset and Exception vectors under the vectors tab of the NIOS II processor.

In the second part of the lab we made similar changes to the hardware. For this lab we used a timer which we gave a period of 125 ms. For the SDRAM controller we needed 64 MB of memory, we adjusted the width to have 13 rows and 10 columns. PIN assignments were assigned based on the DE\_CoE.qsf file. The button\_pio was set to 3 bits wide to allow for operating sw-0, 1, 2. The led\_pio was set to 8 bits wide to allow the system to count up to 255 in binary. The seven\_seg\_pio was set to 16 bits wide to allow for the HEX4,5 to display decimal numbers. Connects were made in Qsys: CLK, RESET, S1, IRQ. Each input/output PIO has its respective external\_connection export assigned.

**Software Changes**

There weren't any software changes necessary for the first part of the lab because we utilized the “Hello, World small” template.

For the second part of the lab, we used the provided “count\_binary” template, however we did have to make some adjustments to the code. In the conditional group *SEVEN\_SEG\_PIO\_BASE* we made the following changes:

| static alt\_u8 segments[16] = { 0x81, 0xCF, 0x92, 0x86, 0xCC, 0xA4, 0xA0, 0x8F, 0x80, 0x84, /\* 0-9 \*/ 0x88, 0xE0, 0xF2, 0xC2, 0xB0, 0xB8 }; /\* a-f \*/ unsigned int data = segments[hex & 15] | (segments[(hex >> 4) & 15] << 8); | static alt\_u8 segments[10] = { 0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7C, 0x07, 0x7F, 0x67, /\* 0-9 \*/ unsigned int data = segments[hex % 10] | (segments[(hex / 10) % 10 ] << 8); |
| --- | --- |

The code on the left prints out the counter values in hexadecimal to both the console and the seven segment display. To print out the decimal values we needed to delete the hex values for a-f and adjust the size to 10 because there were only 10 indexes. After that we adjusted the data integer variable so that the appropriate count values would print out on the seven seg. The segments[hex % 10] allows for continuous counting to show as a value between 0-9. This represents the first integer value on the HEX4 seven\_seg display. The segments[(hex/ 10) % 10] << 8 shifts the bits to the left by 8 to allow for continuous counting on the HEX5 seven\_seg display. These values are counted in decimal.

[#20753946 = 36] [#31820438 = 29] [65 % 100 = 65 count]

In the main section, we adjusted the if statement to if( count == 65 ) because the sum of our SPIRE ID values summed together mod 100 was equal to 65. These were the necessary changes we needed to make for the lab. Additional changes we made was a timer which we defined in the count\_binary.c file and initialized in the *main* function. We did this prior to the while loop (post initial message function call) using start = time(NULL); and in the if statement when we reset the counter. We then initialized the end time within the if statement using end = time(NULL); We added this time function to help us determine how long it took to count to 65.

The code in the static void handle\_button\_press(alt\_u8 type, FILE \*lcd) was altered to remove the use of the LCD display as the DE1-SoC does not have one. This change allowed for SW0, SW1, and SW2 to control LED, HEX, and both respectively.

The usleep(100000) was changed to 270000 to allow for counting up to 65 in approximately 60 seconds.

**Problems Encountered and Solutions**

We encountered many problems during this lab. Few of the problems come from misunderstanding the lab procedure steps, which the TA explained was updated from previous years labs.

One of the issues we kept getting was the “ELF process failed”. This error could come from many issues and it took us many multiple attempts to determine the cause. One cause could be from faulty/incorrect wiring in Qsys. Another we had was due to us selecting the wrong board causing the PIN assignments to be incorrect. This occurred after we uploaded the DE1\_SoC.qsf file, which reset the board selection in Quartis. Sometimes the issue was corrected by refreshing the connection.

Another issue we had was with the Verilog file. We had errors when we tried to perform the Analysis & Synthesis. This was due to the mismatched Module name with the call function. We changed both to be count\_binary\_top. This verilog file was also set as the top-level entity.

Sometimes the board would stop counting on any of the displays after a switch was flipped. This also caused our program to crash. We determined that the interruption was programmed incorrectly. To correct this, we made sure that the interrupt was set on any “edge” trigger.

Understanding that cycling powering to the FPGA will require the FPGA to be reprogrammed was one of the lessons learned during this lab.

**Results and Conclusions**

With the timer function in our code, we were able to accurately determine the length of time it took for the program to initially count to 65. Using the initial usleep time of 100,000, we found that it took 22 seconds. Experimenting further, we determined that that 200,000 yielded a time of 44 seconds. This was how we calculated that the needed value of usleep was approximately 270,000 to reach our goal of counting to 65 in approximately 60 seconds. Our project successfully counts to 65 in binary on the LEDs and in decimal on the seven segment display. The switches successfully alternates between the use of LEDs, HEX, or Both.