EE 371 Lab 3

Katie Neff <1168464>
Adolfo Pineda <1231310>
Sharyar Khalid <15727978>

May 15, 2016

We affirm that this report and its contents are solely the work of Sharyar Khalid, Katie Neff and Adolfo Pineda.

Sharyar Khalid — Signature

5/13/2016 — Date

Katelyn Elizabeth Neff — Signature

5/14/2016 — Date

Adolfo Pineda — Signature

5/14/2016 — Date

Contents

List of Figures

List of Tables

List of Listings

1 INTRODUCTION

This project incorporates new tools for us to use as well as the tools that we have been using up until now. The Altera Qsys system integration tool, a tool that Quartus has to offer, is one of the new tools introduced in this lab that allows us to create and program, using Verilog, a NIOS II microprocessor into the Cyclone V FPGA that is on our DE1-SoC board. After creating this microprocessor, the NIOS II SBT for Eclipse Integrated Development Environment (IDE) tool is used to allow the management of the microprocessor by programming it using the C programming language. The purpose of using these tools are for the creation and the operation of the Count Binary program, the Lights and Switches program, the SRAM program, and the Interlock Management Subsystem Integration program on the FPGA.

2 DISCUSSION

2.1 Design Specification

This project involved four tasks that incorporated the Cyclone V FPGA and the Nios II microprocessor.

2.1.1 Binary Counter

The first task was to download and create a microprocessor and download a premade template from the Eclipse IDE to the FPGA. This was an exercise in creating a NIOS microprocessor and incorporating it to the FPGA. The premade program is a binary counter.

2.1.2 Lights and Switches

The next task was to connect the switches and LEDs of the SoC board to inputs and outputs of the microprocessor. The switches will be inputs to the system while LEDS are the outputs. When switch 0 is low, LEDs 1-7 will light up according to the state of their corresponding switches. When switch 0 is high, LEDs 1- 7 will light up according to the compliment of their corresponding switches.

2.1.3 SRAM

This task involved creating a 2048 x 8 SRAM in Verilog and incorporating the microprocessor. The SRAM will be designed using behavioral Verilog and will have a bidirectional data bus that will write data to the SRAM and output data from the SRAM to be read. Figure 2.1 shows a top level schematic of the SRAM design.

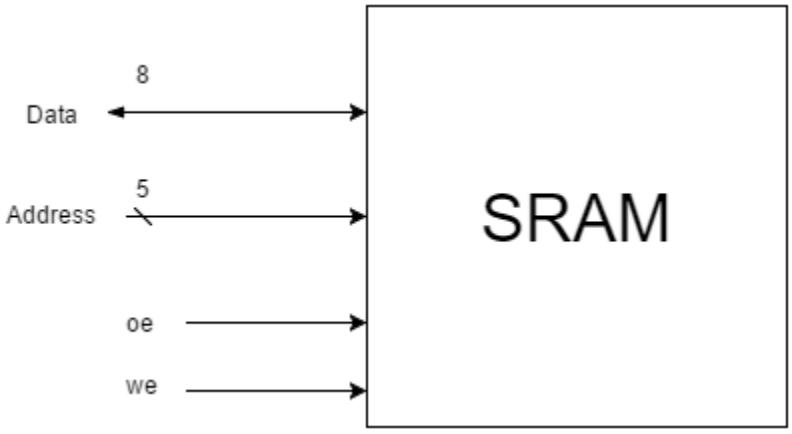

Figure 2.1: Top level schematic of the SRAM design.

To incorporate the microprocessor, all of the SRAM inputs must be connected to microprocessor outputs.

2.1.4 Interlock System

The interlock system was previously created strictly with Verilog modules and controlled through the DE1-SoC board. Now, the inputs and outputs must be ported to the microprocessor and controlled through a C program that takes user input from the console.

Previously, the inputs to the interlock system came from the switches and keys. The inputs were specified as follows:

- SW0 - Arrival Signal
- SW1 - Departure Signal
- SW2 - Inner door open
- SW3 - Inner door close
- KEY3 - Pressurize
- KEY2 - Depressurize

To integrate it with the microprocessor, all of the above inputs are received from the outputs of the microprocessor. The user types characters to the system console and the C program interprets the commands as follows.

- A - Arrival Signal
- B - Departure Signal
- 1 - Open inner door
- 2 - Close inner door
- 3 - Open outer door

- 4 - Close outer door
- p - Pressurize
- d - Depressurize

The rest of the interlock functionality remains the same. The outputs will be displayed on the SoC board HEX display. The HEX display with show a counter and the process the interlock is currently in.

3 TESTING

3.1 Binary Counter

Testing Strategy

To test the binary counter, we will simply look at the output of the console and check if the output follows as expected. This will verify that the microprocessor was correctly downloaded to the FPGA and the premade program is working correctly.

Expected Results

The microprocessor should successfully download to the FPGA, and the program should print a counter to the console.

3.2 Lights

Testing Strategy

To test the lights program, we will look at the output on the SoC board. If the output doesn't follow the expected results, the test fails.

Expected Results

When switch zero is low, the LEDs should illuminate according to the state of the corresponding switch. When switch zero is high, the LEDs should illuminate according to the opposite state of the corresponding switches.

3.3 SRAM

Testing Strategy

A specific strategy will be used to test the SRAM. Using Eclipse, a C program was written to write certain values to the SRAM and then read from those addresses. The Quartus Signal Tap logic analyzer as well as the SoC board will be used to verify that the correct results were written to the correct addresses. We will write the values 127 - 0 to the address spaces 0 - 127, then read from addresses 0 - 127. The LEDs will output the values as they are read from the addresses and the address and data buses will be tapped and displayed using Signal Tap.

Expected Results

The LEDs should appear to count down from 127 to zero. The Signal Tap outputs should show what value is being read from each address. The values written to the addresses should follow this pattern:

```
data = ~address
```

If the data at any address is incorrect, the test as failed.

3.4 Interlock

Testing Strategy

Our test plan for the Interlock system is to build and add the C program to the underlying hardware system and test it by using it through the console as an operator would. We will verify the program is functioning correctly by seeing the outputs on the DE1-SoC board and matching the results we had in the previous lab to make sure that the inputs from the C program are being interpreted by the Verilog the same way as it would when different switches were changed on the DE1-SoC board. We will also test for different cases of user input, verifying that all of the valid commands are accepted by the system and that invalid commands are handled as we have designed.

Expected Results

The output should follow the following state machines.

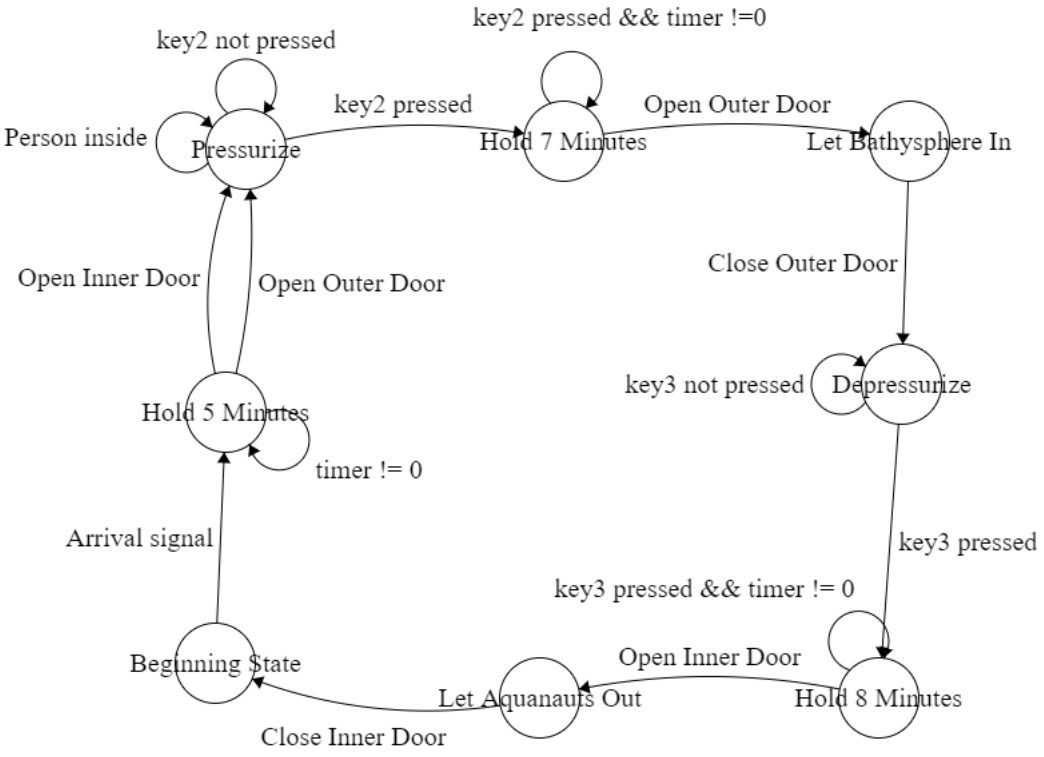

Figure 3.1: State machine and expected outputs for the arrival phase

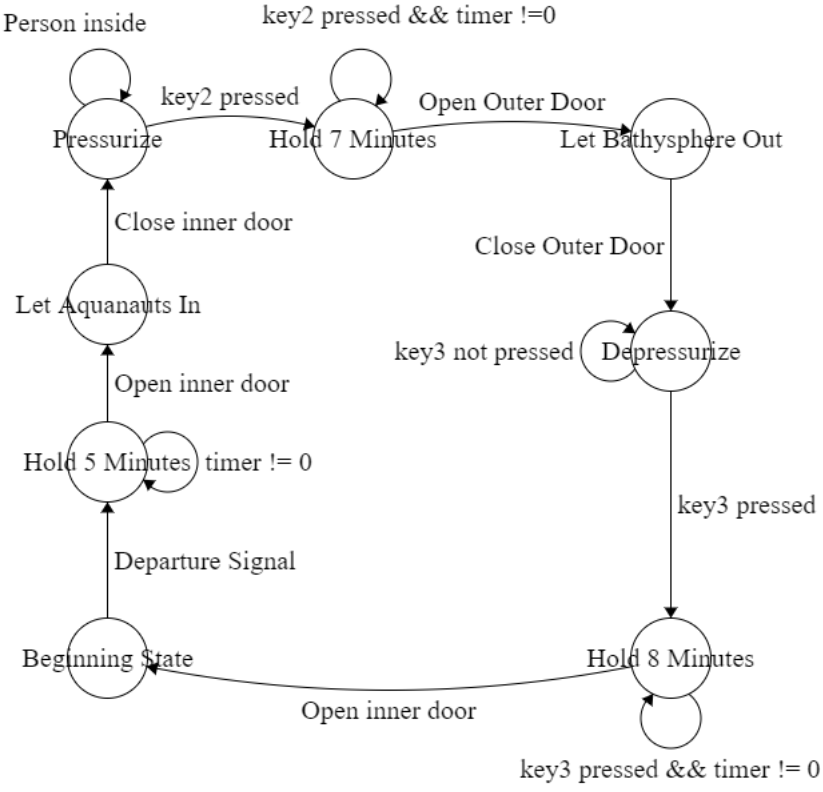

Figure 3.2: State machine and expected outputs for the departure phase

4 RESULTS

The following section describes the results we obtained from our programs through analysis.

Count Binary Results

Our count binary microprocessor was successfully downloaded onto the FPGA and we were able to see it counting, observing that the LEDs turned on in accordance to the binary value the board was reading.

Lights and Switches Results

Our results were just as we expected them to be. SW 0 on the DE1-SoC board was the control switch, and the program behaved according to the current state of that switch. If SW 0 was off on the DE1-SoC board, then the LEDs turned off and on according to the state of their corresponding switch. For example, if SW 1 was on the corresponding LED 1 would be on, if SW 2 was on the corresponding LED 2 would be on, and so on. However, if switch 0 was off the program behaved the opposite, so when we turned on SW 0, the LEDs corresponded the opposite of what the state of their corresponding switch was. If SW 1 was on LED 1 would turn off and if SW1 was off LED 1 would turn on. It was the same for all the other LEDs.

SRAM Results

Our program returned the results we expected it to return, and we were able to verify this using the Signal Tap Logic Analyzer and observing the behavior of our board. The Signal Tap results for the SRAM are shown in Figure 7.1, proving that the required data is being written into the appropriate address. Using a select amount of values for Signal Tap analyzation, Figure 7.1 shows that the value 36, which is the decimal value 54 in hex, is stored in address 4A, which is the decimal value 73 in hex, and this is the result we were expecting this address to have. The subsequent values are also correct since we can observe that the address values increment while the data values decrement. We also observed this behavior on the board noting that, during the read mode operation, the LEDs showed the data in the addresses was decrementing as the address incremented.

Interlock Results

Our Interlock system worked perfectly as designed. It was able to accept commands from the user and pass them to the hardware system, and it was able to display the status messages on the HEX display of the DE1-SoC board. It also successfully handled any invalid commands that were entered. Further, the Interlock program did not interfere with any of our designed operation of the Interlock subsystem, allowing it to function as smoothly and safely as it had before the addition of the C Interlock interface. The C program was able to send signals through commands entered by the user on the console. These signals were then interpreted by the NIOS II processor and

converted to inputs to the Verilog which represented switch simulations from the board. The user would enter commands for example, "open outer door" or "close inner door", these were previously simulated by the switches on the DE1-SoC board, but now we were able to simulate these from the C program. These commands were successfully interpreted by the Verilog code which resulted in the bathysphere program to run successfully.

5 SUMMARY

With the help of the Altera Qsys system integration tool and the NIOS II SBT for Eclipse Integrated Development Environment (IDE) tool, we were able to implement programs using both the Verilog hardware description language and the C programming language on the FPGA. For this project, we were able to incorporate a microprocessor into our programs and use it to implement a binary counter using a template from the NIOS II SBT for Eclipse, a lights and switches C program that was given to us and modify it to use complements, an SRAM that read data from the 128 addresses that had data written in them, and integrate the interlock management subsystem we designed for lab 2 and manage it using said microprocessor.

6 CONCLUSION

This lab helped expand our knowledge of building microprocessors, as well as taught us how to use new Quartus tools such as Qsys. It was very useful in teaching us how to debug and run C applications, as well as how the SRAM works. We learnt a lot about how to follow instructions from datasheets and tutorials, and what to do when things do not work as expected. It further strengthened our knowledge in Signal Tap and also C programming.

In the first part, we learnt how to design a microprocessor. Following that, we learnt how to use that to design a working SRAM, and that lead to us implementing a working microprocessor for our habitat system, in which an operator had full control of the interlock through a console on their computer, without the need of physically changing switches or pressing keys.

7 APPENDICES

7.1 Figures

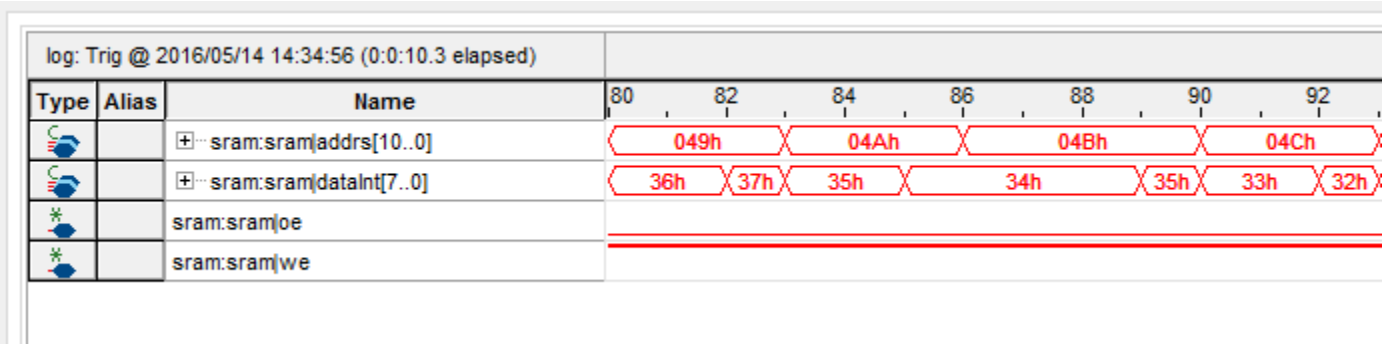

Figure 7.1: Signal Tap of the SRAM

7.2 Division of Work

Table 7.1: Division of work among group members

Name	Tasks	Hours
Katie Neff	Interlock, SRAM, debugging, report	20
Adolfo Pineda	Binary counter, lights, debugging, report	20
Sharyar Khalid	Interlock, SRAM, debugging, report	20