EE 371 Lab 2

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

April 30, 2016

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

Sharyar Khalid	4/29/2016
Signature	Date
Adolfo Pineda	4/29/2016
Signature	Date
Katelyn Elizabeth Neff	4/29/2016
Signature	Date

Contents

List of Figures

List of Tables

List of Listings

1 INTRODUCTION

The rationale behind having us design an interlock system is to give us experience designing, building, and testing a more complex system than we have built in the past. In order to accomplish this, we will need to develop requirements and specifications documents to capture the high-level functionality of our system, as well as functional decomposition and block diagrams to show visually how our system will work at a high-level and begin to establish implementation requirements. The system itself will be a fairly complex finite state machine with many control signals to keep track of and edge cases to consider. Further, we will gain more practice at testing our code using iVerilog and Quartus and performing a failure mode analysis on our design. It is also improving our proficiency with behavioral Verilog and synthesizing HDL code using the Quartus environment.

The C programming part of this project will introduce us to pointers, a very important aspect and tool of the C programming language. In particular, we will practice creating and modifying integer, character, and double pointers and using them to read and modify the values they point to. We will also output these values and a simple calculation to the console when the program is ran.

2 DISCUSSION

2.1 Design Specification

The lab consisted of two separate parts. The first part relied on us designing, building and testing a simple system with several states that replicate the conditions of an interlock system between a modern day bathysphere and an experimental ocean bottom habitat. These states include a reset state, filling/pressurizing state, emptying/depressurizing state, and several intermediate states to check that it is safe to move to the next state (e.g. both ports are closed and the pressure is within 0.1atm of the outer ocean pressure). Our reset state can be activated from any state with a key press and automatically proceeds, if safe, to our safest state with both ports closed and the chamber depressurized. The design requirements also included three timers: a five-minute timer for a bathysphere signaling a request for arrival/departure; a seven minute timer for the fill/pressurize cycle; and an eight minute timer for the drain/depressurize cycle. We designed these timers as simple down counters that use a derivative of the 50Mhz DE1 clock as their clock input. We used the iVerilog environment to test the model, and then we synthesized and ported the design to the Cyclone V FPGA, ensuring that the operations are carried out correctly on the DE1-SoC board. We used the switches, keys, LEDs, and HEX display system to model the input and outputs, along with the control logic.

In the second part, we looked to work with variables and their addressing using the C language. We used pointers to carry out computations on several integers, floats, and char values, to see how calculations take place in C and gain practice using pointers to modify variables. We also output the results of the manipulations and calculations to the console. In order to provide all of this functionality, we included the C standard library, C standard I/O library, and the C type library, which gave us predefined functions to output to the console and create pointers of the types required.

2.2 Design Procedure

We figured out that the second part of the lab would be a lot easier to design and implement, so we finished that by following the instructions in the lab assignment file. Once that was complete and working, we looked to understand what was required in the first part. We realized really quickly that we would be working with several states in one large finite state machine. To figure out what states were required, we started by figuring out the flow of the program, along with the changes that were occurring. We did this by creating a flow chart of the steps taken to allow a bathysphere to enter and leave the interlock.

Once we had the flow chart, we had to figure out what would be the required states of the finite state machine. We started by figuring out the best reset state. We realized that the most important factor for us is to ensure the aquanautsâĂŹ and station crew membersâĂŹ safety. We want to be able to ensure that during reset, all doors are closed and that there is no one blocking the door, regardless of the pressure in the bathysphere. This allowed us to have a reset state that checked the doors status, close them when safe, and depressurize the interlock to the internal pressure and atmosphere of the station interior. After figuring that out, we just had to ensure we covered all states for a bathysphere arriving and departing, and also ensuring we know where the passenger is so that we do not fill or evacuate the chamber in such a way that it causes fatal results. The state changes rely on several kinds of inputs, and take into consideration all possible inputs to prevent errors.

3 TESTING

3.1 Testing Strategy

Our testing strategy involved testing our program using Icarus Verilog and Signal Tap. We used Icarus Verilog to simulate our project by developing test benches that provide inputs and outputs to our main module, and then display the results. Also from Icarus Verilog, we used the GTKWave simulator to obtain the waveform of the data. We then used Signal Tap for analyzing the behavior of our design on the DE1-SoC board.

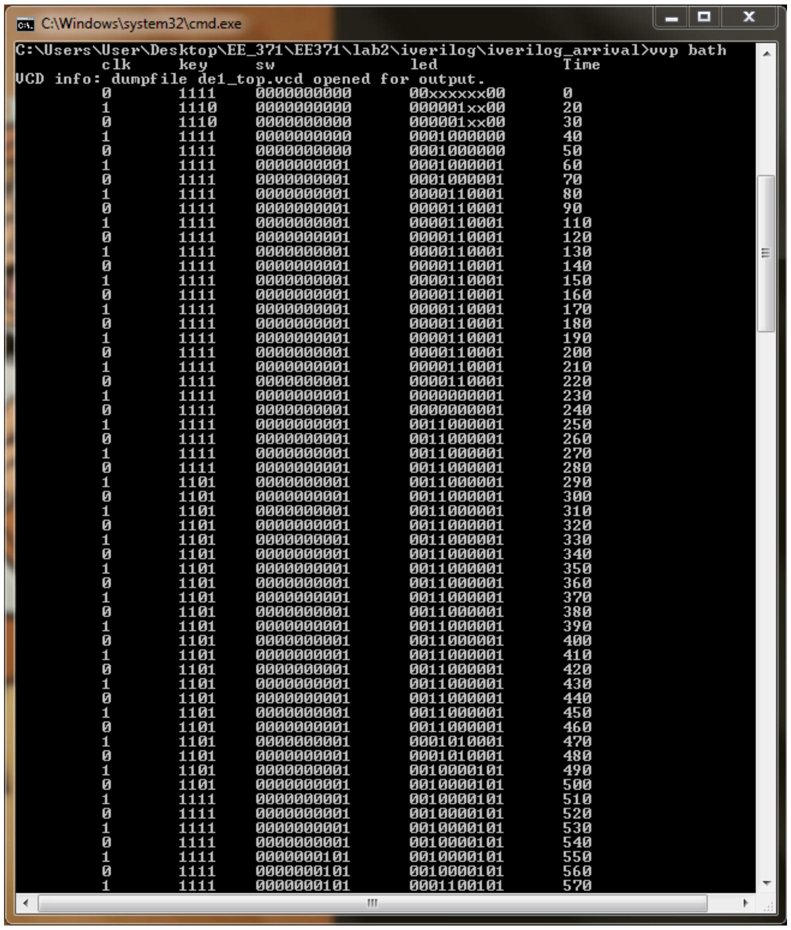
Because our design requires components to behave differently in different conditions, or more specifically when the bathysphere is arriving and when it is departing, we created two test benches for each condition. Our program uses a series of interconnected states where each state completes a specific action such as pressurizing the interlock or sending a departure signal. We used the Icarus Verilog simulators to check if the correct states were being accessed when a certain input is entered. After confirming that we were obtaining the correct data, it was then time to transfer the design onto the DE1-SoC board to observe its behavior and record it using Signal Tap.

3.2 Expected Results

We expect our program to iterate through different states with specific inputs. For example, setting the 1st bit of the key variable to 0 and then back to 1, simulating KEY 0 being pressed on the board, should put our program into the first state. To confirm this, we used the variable called led, which is also the variable used to simulate the LEDâĂŹs that are being lit up on the board, to show the state. Using the 7th, 6th, 5th, and 4th bit of the led variable, we should be able to see

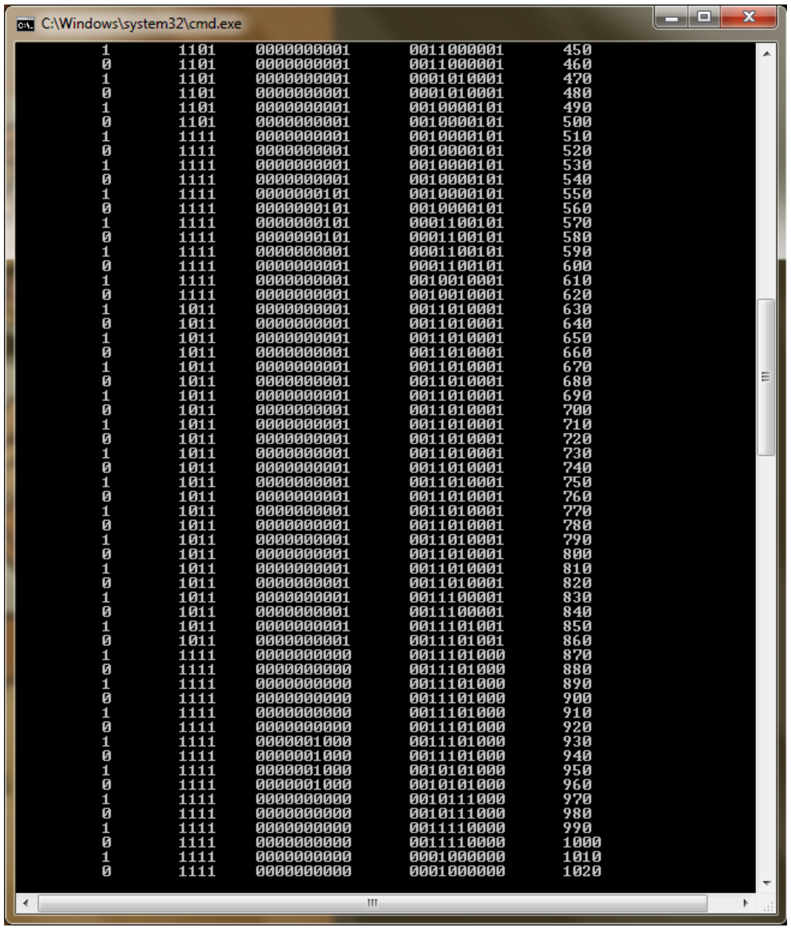
0100, the value of the first state. Continuing this example, if we were to set the 1st bit of the variable sw equal to 1, which represents flipping up SW 0 on the board, we should then enter state 0011 which activates the arrival signal and counts down from 5 minutes. After giving our program enough clock cycles to finish this countdown, it should then enter the state 1100 where it waits for the user to depressurize the interlock by setting the 2nd bit of the key variable equal to 0 and then 1 (pressing KEY 1). We should be able to continue this process of entering different states to accomplish the goal of completing one arrival cycle and arrive back to the first state 0100. From there, the departure cycle can begin and a similar process takes place.

4 RESULTS



```
C:\Users\User\Desktop\EE_371\EE371\lab2\iverilog\iverilog_arrival>vvp bath
           clk      key     sw                led               Time
VCD info: dumpfile de1_top.vcd opened for output.
           0        1111    0000000000        00xxxxxx00        0
           1        1110    0000000000        000001xx00        20
           0        1110    0000000000        000001xx00        30
           1        1111    0000000000        0001000000        40
           0        1111    0000000000        0001000000        50
           1        1111    0000000001        0001000001        60
           0        1111    0000000001        0001000001        70
           1        1111    0000000001        0000110001        80
           0        1111    0000000001        0000110001        90
           1        1111    0000000001        0000110001        110
           0        1111    0000000001        0000110001        120
           1        1111    0000000001        0000110001        130
           0        1111    0000000001        0000110001        140
           1        1111    0000000001        0000110001        150
           0        1111    0000000001        0000110001        160
           1        1111    0000000001        0000110001        170
           0        1111    0000000001        0000110001        180
           1        1111    0000000001        0000110001        190
           0        1111    0000000001        0000110001        200
           1        1111    0000000001        0000110001        210
           0        1111    0000000001        0000110001        220
           1        1111    0000000001        0000000001        230
           0        1111    0000000001        0000000001        240
           1        1111    0000000001        0011000001        250
           0        1111    0000000001        0011000001        260
           1        1111    0000000001        0011000001        270
           0        1111    0000000001        0011000001        280
           1        1101    0000000001        0011000001        290
           0        1101    0000000001        0011000001        300
           1        1101    0000000001        0011000001        310
           0        1101    0000000001        0011000001        320
           1        1101    0000000001        0011000001        330
           0        1101    0000000001        0011000001        340
           1        1101    0000000001        0011000001        350
           0        1101    0000000001        0011000001        360
           1        1101    0000000001        0011000001        370
           0        1101    0000000001        0011000001        380
           1        1101    0000000001        0011000001        390
           0        1101    0000000001        0011000001        400
           1        1101    0000000001        0011000001        410
           0        1101    0000000001        0011000001        420
           1        1101    0000000001        0011000001        430
           0        1101    0000000001        0011000001        440
           1        1101    0000000001        0011000001        450
           0        1101    0000000001        0011000001        460
           1        1101    0000000001        0001010001        470
           0        1101    0000000001        0001010001        480
           1        1101    0000000001        0010000101        490
           0        1101    0000000001        0010000101        500
           1        1111    0000000001        0010000101        510
           0        1111    0000000001        0010000101        520
           1        1111    0000000001        0010000101        530
           0        1111    0000000001        0010000101        540
           1        1111    0000000101        0010000101        550
           0        1111    0000000101        0010000101        560
           1        1111    0000000101        0001100101        570
```

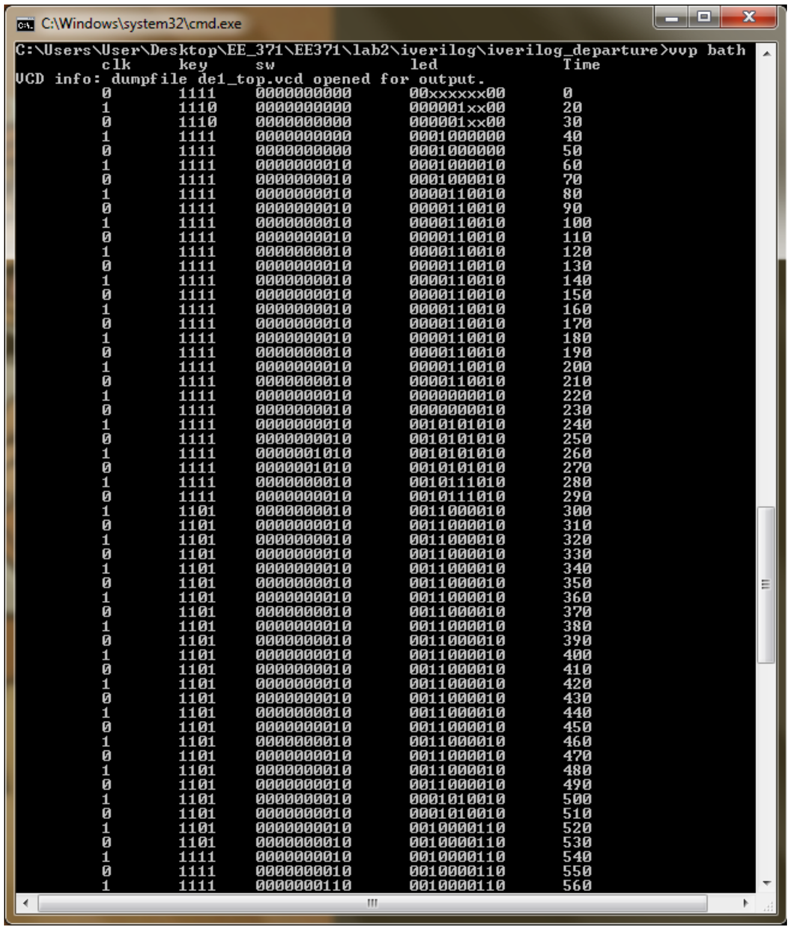
Figure 4.1: The first half of the arrival cycle using iverilog simulation



```
C:\Windows\system32\cmd.exe
        1        1101     0000000001       0011000001       450
        0        1101     0000000001       0011000001       460
        1        1101     0000000001       0001010001       470
        0        1101     0000000001       0001010001       480
        1        1101     0000000001       0010000101       490
        0        1101     0000000001       0010000101       500
        1        1111     0000000001       0010000101       510
        0        1111     0000000001       0010000101       520
        1        1111     0000000001       0010000101       530
        0        1111     0000000001       0010000101       540
        1        1111     0000000101       0010000101       550
        0        1111     0000000101       0010000101       560
        1        1111     0000000101       0001100101       570
        0        1111     0000000101       0001100101       580
        1        1111     0000000001       0001100101       590
        0        1111     0000000001       0001100101       600
        1        1111     0000000001       0010010001       610
        0        1111     0000000001       0010010001       620
        1        1011     0000000001       0011010001       630
        0        1011     0000000001       0011010001       640
        1        1011     0000000001       0011010001       650
        0        1011     0000000001       0011010001       660
        1        1011     0000000001       0011010001       670
        0        1011     0000000001       0011010001       680
        1        1011     0000000001       0011010001       690
        0        1011     0000000001       0011010001       700
        1        1011     0000000001       0011010001       710
        0        1011     0000000001       0011010001       720
        1        1011     0000000001       0011010001       730
        0        1011     0000000001       0011010001       740
        1        1011     0000000001       0011010001       750
        0        1011     0000000001       0011010001       760
        1        1011     0000000001       0011010001       770
        0        1011     0000000001       0011010001       780
        1        1011     0000000001       0011010001       790
        0        1011     0000000001       0011010001       800
        1        1011     0000000001       0011010001       810
        0        1011     0000000001       0011010001       820
        1        1011     0000000001       0011100001       830
        0        1011     0000000001       0011100001       840
        1        1011     0000000001       0011101001       850
        0        1011     0000000001       0011101001       860
        1        1111     0000000000       0011101000       870
        0        1111     0000000000       0011101000       880
        1        1111     0000000000       0011101000       890
        0        1111     0000000000       0011101000       900
        1        1111     0000000000       0011101000       910
        0        1111     0000000000       0011101000       920
        1        1111     0000001000       0011101000       930
        0        1111     0000001000       0011101000       940
        1        1111     0000001000       0010101000       950
        0        1111     0000001000       0010101000       960
        1        1111     0000000000       0010111000       970
        0        1111     0000000000       0010111000       980
        1        1111     0000000000       0011110000       990
        0        1111     0000000000       0011110000       1000
        1        1111     0000000000       0001000000       1010
        0        1111     0000000000       0001000000       1020
```

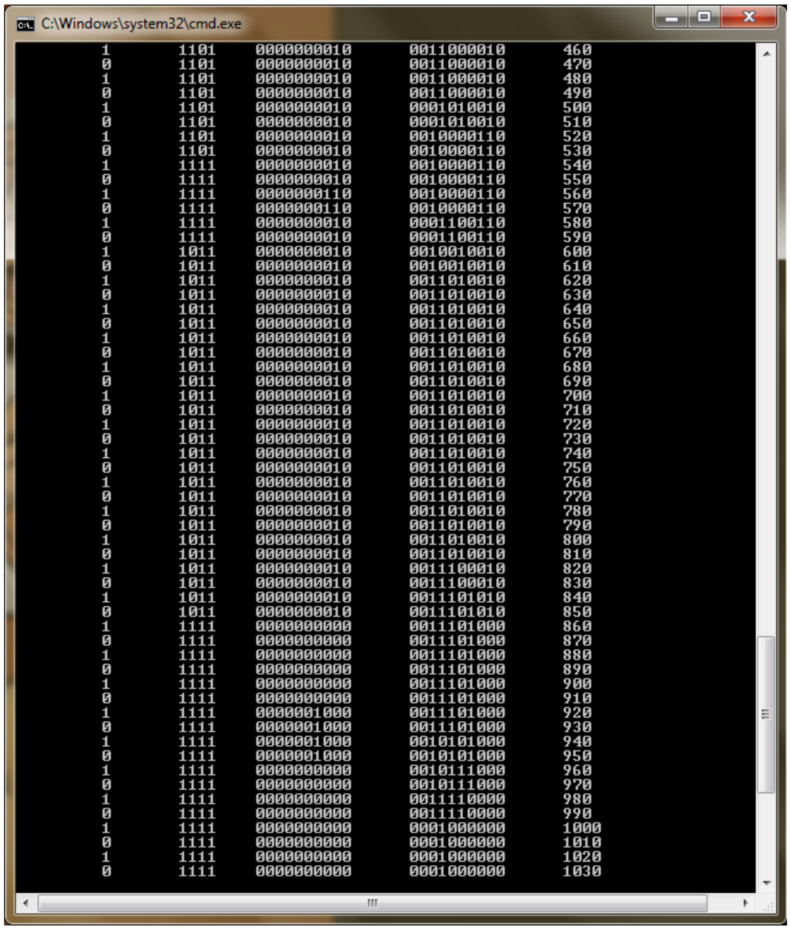
Figure 4.2: The second half of the arrival cycle using iverilog simulation

We were able to successfully complete one arrival cycle using Icarus Verilog. At time 40, we entered state 0100 which allowed us to set the first bit of the sw variable, which represents the SW 0 on the board, to 1. The led column shows the rest of the states that our design enters, which can be mapped to our Verilog code that denotes the value of the state and its functionality. The following results show one full departure cycle which were obtained using a similar process and the GTKWaves for both arrival and departure.



```
C:\Users\User\Desktop\EE_371\EE371\lab2\iverilog\iverilog_departure>vvp bath
           clk       key      sw                 led                Time
VCD info: dumpfile de1_top.vcd opened for output.
           0         1111     0000000000         00xxxxxx00         0
           1         1110     0000000000         000001xx00         20
           0         1110     0000000000         000001xx00         30
           1         1111     0000000000         0001000000         40
           0         1111     0000000000         0001000000         50
           1         1111     0000000010         0001000010         60
           0         1111     0000000010         0001000010         70
           1         1111     0000000010         0000110010         80
           0         1111     0000000010         0000110010         90
           1         1111     0000000010         0000110010         100
           0         1111     0000000010         0000110010         110
           1         1111     0000000010         0000110010         120
           0         1111     0000000010         0000110010         130
           1         1111     0000000010         0000110010         140
           0         1111     0000000010         0000110010         150
           1         1111     0000000010         0000110010         160
           0         1111     0000000010         0000110010         170
           1         1111     0000000010         0000110010         180
           0         1111     0000000010         0000110010         190
           1         1111     0000000010         0000110010         200
           0         1111     0000000010         0000110010         210
           1         1111     0000000010         0000000010         220
           0         1111     0000000010         0000000010         230
           1         1111     0000000010         0010101010         240
           0         1111     0000000010         0010101010         250
           1         1111     0000001010         0010101010         260
           0         1111     0000001010         0010101010         270
           1         1111     0000000010         0010111010         280
           0         1111     0000000010         0010111010         290
           1         1101     0000000010         0011000010         300
           0         1101     0000000010         0011000010         310
           1         1101     0000000010         0011000010         320
           0         1101     0000000010         0011000010         330
           1         1101     0000000010         0011000010         340
           0         1101     0000000010         0011000010         350
           1         1101     0000000010         0011000010         360
           0         1101     0000000010         0011000010         370
           1         1101     0000000010         0011000010         380
           0         1101     0000000010         0011000010         390
           1         1101     0000000010         0011000010         400
           0         1101     0000000010         0011000010         410
           1         1101     0000000010         0011000010         420
           0         1101     0000000010         0011000010         430
           1         1101     0000000010         0011000010         440
           0         1101     0000000010         0011000010         450
           1         1101     0000000010         0011000010         460
           0         1101     0000000010         0011000010         470
           1         1101     0000000010         0011000010         480
           0         1101     0000000010         0011000010         490
           1         1101     0000000010         0001010010         500
           0         1101     0000000010         0001010010         510
           1         1101     0000000010         0010000110         520
           0         1101     0000000010         0010000110         530
           1         1111     0000000010         0010000110         540
           0         1111     0000000010         0010000110         550
           1         1111     0000000110         0010000110         560
```

Figure 4.3: The first half of the departure cycle using iverilog simulation



```
        1        1101     0000000010        0011000010      460
        0        1101     0000000010        0011000010      470
        1        1101     0000000010        0011000010      480
        0        1101     0000000010        0011000010      490
        1        1101     0000000010        0001010010      500
        0        1101     0000000010        0001010010      510
        1        1101     0000000010        0010000110      520
        0        1101     0000000010        0010000110      530
        1        1111     0000000010        0010000110      540
        0        1111     0000000010        0010000110      550
        1        1111     0000000110        0010000110      560
        0        1111     0000000110        0010000110      570
        1        1111     0000000010        0001100110      580
        0        1111     0000000010        0001100110      590
        1        1011     0000000010        0010010010      600
        0        1011     0000000010        0010010010      610
        1        1011     0000000010        0011010010      620
        0        1011     0000000010        0011010010      630
        1        1011     0000000010        0011010010      640
        0        1011     0000000010        0011010010      650
        1        1011     0000000010        0011010010      660
        0        1011     0000000010        0011010010      670
        1        1011     0000000010        0011010010      680
        0        1011     0000000010        0011010010      690
        1        1011     0000000010        0011010010      700
        0        1011     0000000010        0011010010      710
        1        1011     0000000010        0011010010      720
        0        1011     0000000010        0011010010      730
        1        1011     0000000010        0011010010      740
        0        1011     0000000010        0011010010      750
        1        1011     0000000010        0011010010      760
        0        1011     0000000010        0011010010      770
        1        1011     0000000010        0011010010      780
        0        1011     0000000010        0011010010      790
        1        1011     0000000010        0011010010      800
        0        1011     0000000010        0011010010      810
        1        1011     0000000010        0011100010      820
        0        1011     0000000010        0011100010      830
        1        1011     0000000010        0011101010      840
        0        1011     0000000010        0011101010      850
        1        1111     0000000000        0011101000      860
        0        1111     0000000000        0011101000      870
        1        1111     0000000000        0011101000      880
        0        1111     0000000000        0011101000      890
        1        1111     0000000000        0011101000      900
        0        1111     0000000000        0011101000      910
        1        1111     0000001000        0011101000      920
        0        1111     0000001000        0011101000      930
        1        1111     0000001000        0010101000      940
        0        1111     0000001000        0010101000      950
        1        1111     0000000000        0010111000      960
        0        1111     0000000000        0010111000      970
        1        1111     0000000000        0011110000      980
        0        1111     0000000000        0011110000      990
        1        1111     0000000000        0001000000      1000
        0        1111     0000000000        0001000000      1010
        1        1111     0000000000        0001000000      1020
        0        1111     0000000000        0001000000      1030
```

Figure 4.4: The second half of the departure cycle using iverilog simulation

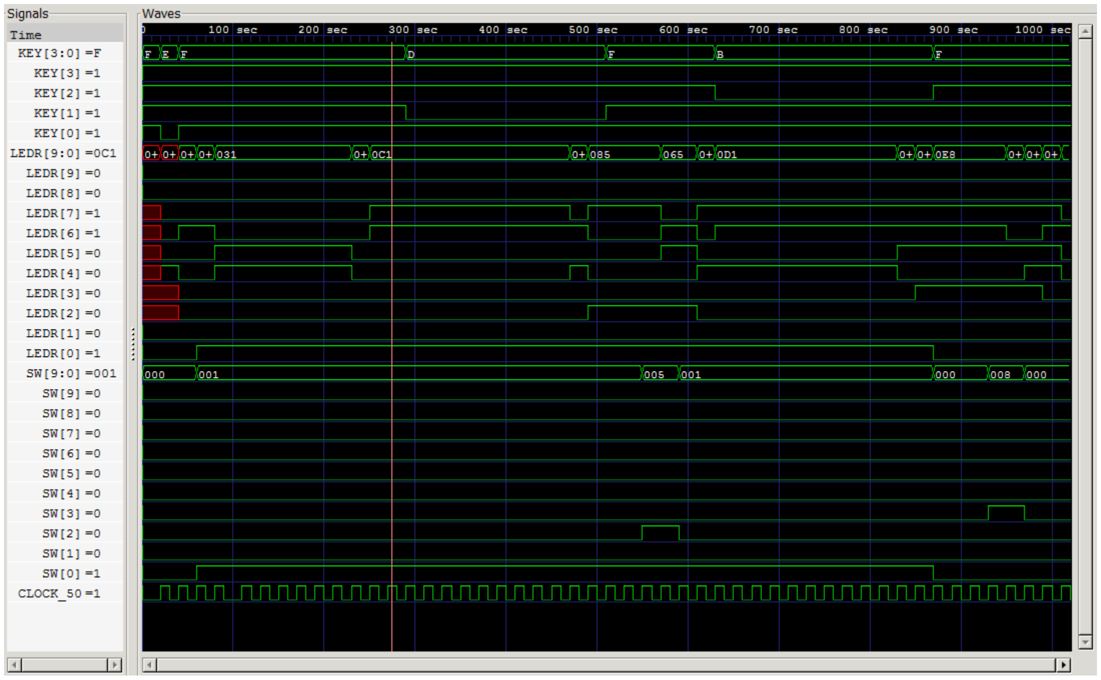


Figure 4.5: GTKWave of the arrival cycle

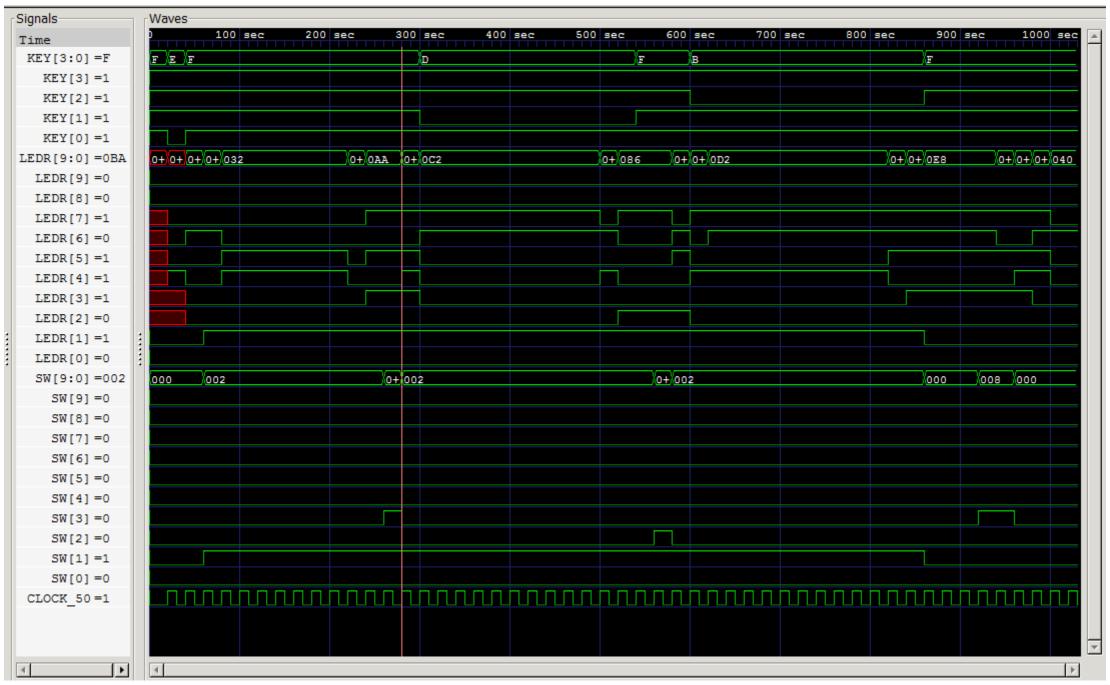


Figure 4.6: GTKWave of the departure cycle

And finally, after acquiring the desired results, we then transferred our design onto the DE1-SoC board and used Signal Tap to record the results. We were able to observe that the DE1-SoC board presented the desired results, and so we used Signal Tap to record this and obtained the following waveforms:

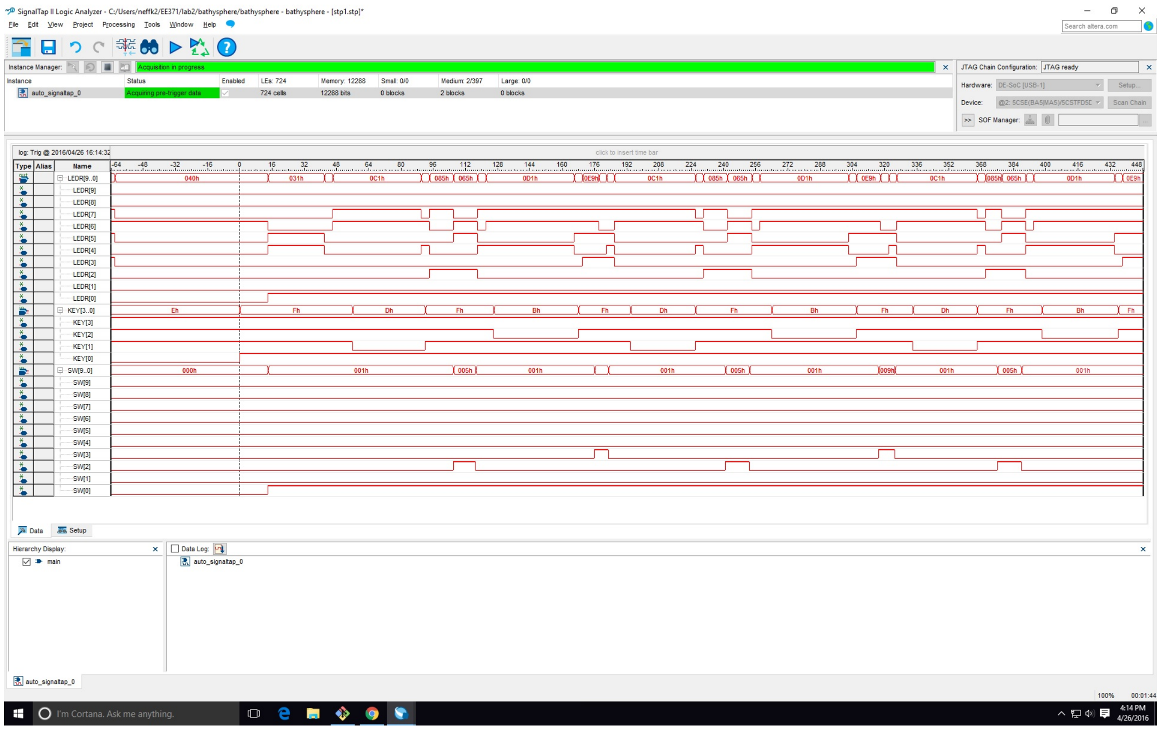

SignalTap II Logic Analyzer - C:/Users/neffk2/EE371/lab2/bathysphere/bathysphere - bathysphere - [stp1.stp]*
File Edit View Project Processing Tools Window Help
Instance Manager:
Acquisition in progress
JTAG Chain Configuration: JTAG ready
Instance Status Enabled LEs: 724 Memory: 12288 Small: 0/0 Medium: 2/397 Large: 0/0
auto_signaltap_0 Acquiring pre-trigger data 724 cells 12288 bits 0 blocks 2 blocks 0 blocks
Hardware: DE-SoC [USB-1]
Device:
SOF Manager:
log: Trig @ 2016/04/26 16:14:32
Type Alias Name
LEDR[9..0]
KEY[3..0]
SW[9..0]
Data
Setup
Hierarchy Display:
main
Data Log:
auto_signaltap_0
I'm Cortana. Ask me anything.
4:14 PM
4/26/2016

Figure 4.7: Signal Tap results of the arrival cycle

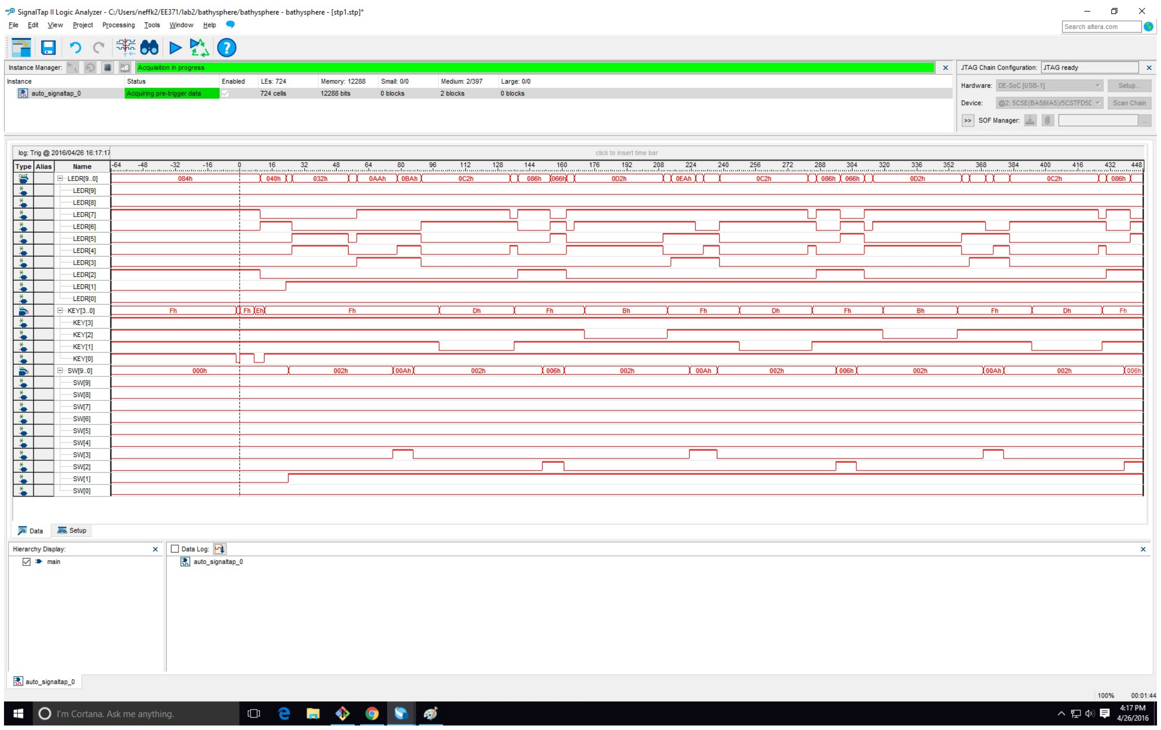

SignalTap II Logic Analyzer - C:/Users/neffk2/EE371/lab2/bathysphere/bathysphere - bathysphere - [stp1.stp]*
Instance Manager:
Acquisition in progress
auto_signaltap_0
Acquiring pre-trigger data
JTAG Chain Configuration: JTAG ready
Hardware: DE-SoC [USB-1]
Device: @2: 5CSE(BA5|MA5)/5CSTFD5C
log: Trig @ 2016/04/26 16:17:17
LEDR[9..0]
KEY[3..0]
SW[9..0]
Data
Setup
Hierarchy Display:
Data Log:

Figure 4.8: Signal Tap results of the departure cycle

5 CONCLUSION

The bathysphere system was a way to implement a complicated finite state machine. Through development of this project, it was important to plan the design before implementing. The states had to be clearly specified and the inputs that change each state has to be decided. With careful planning, the complexity of the project could be tackled. Without planning and clear naming of states, it would be impossible to debug if something when wrong. When dealing with this level of complexity, it is important to keep these things in mind.

6 APPENDICES

6.1 Interlock System Requirements Document

6.1.1 Abstract

In this document, we will describe the specified requirements for the interlock system. We will describe the inputs, outputs and major functions.

6.1.2 Introduction

The interlock system will take input from the user and output various things according to current state. Based on the description of the system, we have decided that the interlock should have the inputs and outputs described in this report. These inputs and outputs will be ported to the DE1-SoC port.

6.1.3 Inputs to the System

The system must have the inputs specified below.

- Bathysphere arriving signal: A signal that announces that the bathysphere is arriving to the interlock and intending to port.
- Bathysphere departing signal: A signal that announces the bathysphere wants to depart the interlock.
- Outer door open/close signal: This will open/close the interlock door to the ocean.
- Inner door open/close signal: This will open/close the door to the underwater habitat.
- Person inside interlock signal: This will tell the system that a person has entered the interlock.
- Pressurize chamber: This will cause the interlock chamber to fill with water.
- Depressurize chamber: This will empty the interlock chamber with water.
- Reset: This resets the system to an original state.
- Clock: The clock that the system runs on.

6.1.4 Outputs from the System

The interlock system must produce the following outputs. Some output signals may seem similar to the input signals.

- Bathysphere arriving: This will show the user that the bathysphere is on its way to the interlock to be ported.
- Bathysphere departing: This will show the user that the bathysphere is getting ready to depart.
- Timer: This will be used to display the minute counter as it counts down from five seven or eight minutes during the waiting phases of arrival or departure.
- Inner/outer door open: two signals that correspond to the inner and outer doors being opened. If the signal is high, the corresponding door is opened.
- State indicator: this signal indicated which current state the interlock system.

6.1.5 Major Functions

The interlock state machine can be divided into two components: the arrival and departure sequence. Each sequence requires different input from the user and displays different output behaviors of the interlock system. A diagram of the states of the arrival and departure processes can be found in Figures 6.1 and 6.2 respectively.

6.2 Interlock System Design Specification

6.2.1 Abstract

This document will outline the specific usages of the interlock system. We will discuss inputs, outputs and major functions.

6.2.2 Introduction

The interlock system was designed to be a preparation chamber between a bathysphere used to explore the ocean and an underwater habitat where humans live. The interlock system takes input from the operator and must follow a sequence of instructions to work correctly. The system prepares for arrival and departure of a bathysphere, and has numerous safety checks to ensure the inside habitat is not flooded with water and that no one is stuck in the chamber while it is pressurizing.

6.2.3 Inputs to the System

The inputs form the operator to the system are as follows.

- Bathysphere arriving signal: A signal that announces that the bathysphere is arriving to the interlock and intending to port. This signal is low (0) by default and high (1) when the bathysphere is intending to port to the interlock. This signal should only be high for the five minutes before the interlock prepares for the arrival of the bathysphere. Other times it should be kept low.
- Bathysphere departing signal: A signal that announces the bathysphere wants to depart the interlock. This signal is low (0) by default, when there is no bathysphere leaving, and high (1) when the bathysphere signals for departure. This signal must be kept high only during the moment the bathysphere signals its departure and the five minute waiting period between the original signal and the preparation stage.
- Outer door open/close signal: This will open/close the interlock door to the ocean. This signal is low (0) when the door is closed and high (1) when the door is opened. This system will only allow the signal to be high when the chamber and ocean have a less than .1 atm of pressure difference.
- Inner door open/close signal: This will open/close the door to the underwater habitat. This signal is low (0) when the door is closed and high (1) when the door is open. The system will only allow the signal to be high when there is a less than .1 atm pressure difference between the interlock and the underwater habitat.
- Person inside interlock signal: This will tell the system that a person has entered the interlock. This signal is low (1) when there is not a person in the interlock and high (1) when there is. This signal is controlled by the operator. When a person enters, it must be pulled high. When they exit, it must be pulled low. This signal will stop the chamber from pressurizing if its pulled high to avoid filling the chamber with water when someone is inside.

- Pressurize chamber: This will cause the interlock chamber to fill with water. When the signal is high (1), the chamber will pressurize. However, it will only pressurize if other conditions are met. Both the inner and outer door must be closed and sealed and there must not be a person inside the chamber. In addition, the interlock system must have previously received an arrival or departure signal. Pressurizing takes seven minutes.
- Depressurize chamber: This will empty the interlock chamber of water. When the signal is high (1), the chamber will depressurize. However, it will only depressurize if certain conditions are met. Both the inner door and outer door must be closed and the system must have previously received an arrival or departure signal. Depressurizing takes eight minutes.
- Reset: This resets the system to an original state, i. e. the neutral beginning state before the system receives any arrival or departure signals. If the signal is high (1) the system is reset. This signal is active high.
- Clock: The clock that the system runs on. The frequency of the clock affects the speed of the minute counter.

6.2.4 Outputs from the System

The outputs from the system are described as follows.

- Bathysphere arriving: This will show the user that the bathysphere is on its way to the interlock to be ported. If the signal is high (1) the bathysphere is on its way to the interlock.
- Bathysphere departing: This will show the user that the bathysphere is getting ready to depart. If the signal is high, the bathysphere has signaled its departure and is in the five minute waiting stage.
- Timer: This will be used to display the minute counter as it counts down from five seven or eight minutes during the waiting phases of arrival or departure. This is specifically designed to be ported to a HEX display. The speed at which the timer counts down is dependent on the clock.
- Inner/outer door open: two signals that correspond to the inner and outer doors being opened. If the signal is high, the corresponding door is opened. If low, the door is closed.
- State indicator: this signal indicated which current state the interlock system. Each state has a four bit number, and this signal outputs the four bit binary representation of which state the system is in.

6.2.5 Major Functions

The interlock system has two major functions: docking the bathysphere and launching the bathysphere. These functions can be referred to as the "arrival phase" and the "departure phase". The arrival phase follows a certain procedure to dock the bathysphere: it detects the bathysphere arriving, fills the chamber with water, allows the bathysphere to enter the chamber, empties the chamber, and opens the inner door to let the aquanauts out. The departure phase follows a similar procedure: the aquanauts are let in the chamber, the chamber is filled with water, the outer door is opened to let the bathysphere leave, the door is closed and the chamber is emptied of water again. These functions require input from the operator to proceed in most cases.

6.3 Interlock System Functional Decomposition

The two top level functions of the interlock systems can be divided into two sections: the arrival and the departure functions. A state diagram of both of these functions can be found in Figures 6.1 and 6.2 respectively. Each function requires different sequences of input from the user to function correctly. Below describes how each function works in more detail.

Arrival

The arrival process starts at a neutral beginning state. The bathysphere could be inside the interlock or outside. Once the arrival signal is received, the interlock system assumes that the bathysphere is outside and prepares for the arrival.

After the arrival, a timer displays and counts down from five minutes. This is to meet the requirement that the interlock system receives the signal five minutes before the arrival of the bathysphere. After five minutes, the interlock checks if any of its doors are open before pressurizing. If both the doors are closed, the system moves on to the next state and prepares for pressurization of the chamber.

To prepare for pressurization, the system checks to see if there is a person in the chamber. If there is a person, the chamber will not pressurize until the inner door is opened and the person is let out. After the chamber is cleared for pressurization, the operator is instructed to pressurize by pressing a key and holding it down until the timer counts down from seven minutes.

Next, the operator is instructed to open the outer door and let the bathysphere inside. Once the bathysphere is inside, the operator must close the outer door. Once the outer door is closed, the operator is instructed to depressurize the chamber by pressing a key and holding until the timer counts down from eight minutes.

After the chamber is depressurized, the inner door opens and the aquanauts are let out. The inner door is closed and the system returns to its neutral beginning state, where it can receive a bathysphere arrival or departure signal.

Departure

The departure functions starts in an initial beginning state. After receiving the departure signal from the bathysphere, a timer is displayed and counts down from five minutes. After the five minute buffer period, the interlock can prepare for departure.

The operator first must open the inner door to let the aquanauts in the chamber and into the bathysphere. Next, the door must be closed and sealed to prepare for pressurization. Before pressurization, the chamber is checked to see if there's a person inside (who's not an aquanaut in the bathysphere). If there is confirmation that a person is inside, the inner door must be opened to let them out. After they are let out, the chamber is allowed to pressurize.

To pressurize the chamber, the operator must press a key and hold it for seven minutes. A timer is displayed while it counts down from seven minutes. After the chamber is fully pressurized, the outer door is opened and the bathysphere leaves the chamber. Next, the outer door is closed and the operator must depressurize the chamber. The operator presses a button for eight minutes to depressurize the chamber, and the interlock returns to its neutral beginning state.

6.4 Interlock System Functional Failure Mode Analysis

Here we will do a failure mode analysis for each of the signals of the interlock system. We will describe errors when the signal is stuck at zero and when the signal is stuck at one.

6.4.1 Stuck At Zero

Failure mode analysis of each signal if it is "stuck at zero" is described below.

Clock

All of our behavior is dependent on the positive edge of the clock. If the clock is stuck at zero, we would never enter our behavior loop (`always @(posedge clk)`) and the behavior code would never execute. Thus, the clock signal being stuck at zero would be a fatal error to the system. No outputs would be produced and no other inputs could be read. This error is unlikely if we assert that some clock signal is being ported to the system.

Reset

Our reset is active low. If the reset signal is stuck at zero, then the system would be stuck in its reset state. The outputs would never change from their default reset settings. This would be a fatal error to the system.

Bathysphere Arriving

The bathysphere signals its arrival five minutes before porting. This is the first signal that the interlock must recieve before it can begin preparing for the arrival of the bathysphere (i. e. the arrival signal must be pulled high). If this signal is stuck at zero, then the arrival of the bathysphere is never signaled and the interlock does not enter into the preparation process. This would be a fatal error to the system.

Bathysphere Departing

Similar to the signal described above, the bathysphere must signal its departure five minutes before the system begins to prepare the chamber for departure. The system must recieve this signal before it can prepare for departure (i. e. the departure signal must be pulled high). If this signal is stuck at zero, the system never recieves a departure signal and cannot enter into the process of preparing the chamber for departure. This would be a fatal error to the system.

Person Inside

This signal tells the system that there is a person inside the interlock chamber. This is a safety measure to ensure the chamber doesn’t pressurize or open the outter door while the person is there. If the signal is low, this indicates that there is not a person in the chamber, and the interlock can proceed with any pressurizing or door locking process. If the signal is high, this indicates there is a person in the chamber and the inner door must be open to let them out. If this signal is stuck at zero, the system would never recieve a signal that a person is inside, and pressurizing/door opening proceedures would never be suspended. This would be a major safety hazard and would require that the error is fixed immediately. If not, the chamber could fill while there is a person inside.

Inner Door Open

This signal opens the door connecting the interlock and the underwater habitat. If this signal is stuck at zero, the door would never open, and the people/aquanauts inside the interlock would never be able to enter the habitat. Additionally, aquanats would never be able to enter the interlock from the inside of the habitat. This would also interrupt many processes of the state machine. For example, if a person is detected inside the interlock, the door must be opened and they must be let out before the system can continue with the pressurizing stage. If this signal is stuck at zero, the person would never be let out and the system coudl never advance to the pressurizing stage. This would be a fatal error.

Outer Door Open

This signal opens the outer door between the interlock and the ocean. The signal must be pulled high to open the outer door and allow the bathysphere to enter or exit the interlock. If this signal is stuck at zero, the bathysphere would never be able to leave or enter. This would not be a seriously dangeous scenario, but it would need to be addressed for the system to work as intended. Additionally, the system must recieve the door open signal before it can begin the process of depressurizing the chamber again.

Pressurize

This signal fills the interlock chamber with water until it's within .1 atm of the outside ocean. If this signal is stuck at zero, the chamber would never be able to pressurize and the outer door would never be able to open. This error would cause the system to not work as specified and would need to be addressed.

Depressurize

This signal empties the water in the chamber and depressurizes the chamber until it is within .1 atm of the inside habitat. This must happen before the inner door can be opened. If this signal is stuck at zero, the chamber would be depressurized and the inner door would never be able to be opened. This error would cause the system to not work as specified and would need to be addressed.

6.4.2 Stuck at One

The failure mode analysis of each signal if it is "stuck at one" is described below.

Clock

All of our behavior is dependent on the positive edge of the clock. If the clock is stuck at one, we would never enter our behavior loop (`always @(posedge clk)`) and the behavior code would never execute. Thus, the clock signal being stuck at zero would be a fatal error to the system.

Reset

The reset signal is active low. If the reset signal is stuck at one, the system would never be able to reset, which could cause some undefined behavior. This would not be a fatal error to the system, but it would need to be addressed to avoid undefined behavior.

Bathysphere Arriving

If pulled high, this signals that the bathysphere is arriving and the interlock is prepared for the arrival. If this signal is stuck at one, the system would always recieve the signal that the interlock is arriving. This would not cause a problem in the arrival stage, but when the bathysphere wants to depart, the arrival signal pulled high would disrupt the departure process. This would be a fatal error that needs to be addressed.

Bathysphere Departing

This signal tells the interlock system that the bathysphere wants to depart the chamber. If this signal is stuck at one, the system will recieve the signal that the bathysphere wants to leave and will behave accordingly. However, in the arrival phase, this signal pulled high would cause problems. Thus, this error should be addressed for the system to work properly.

Person Inside

This signal tells the interlock system that there is a person inside the chamber. If the signal is high, the system will not be able to pressurize the chamber until the signal is pulled low again. If the signal is stuck at one, then the system would never pressurize the chamber.

Inner Door Open

This signal tells the inner door to open. If this signal is stuck at one, then the door will never close and the interlock will never be able to pressurize. This would not allow the system to behave as specified and would need to be addressed.

Outer Door Open

This signal tells the outer door to the ocean to open. If this signal is stuck at one, then the door will never close, and the chamber will never be able to depressurize. This would not allow the system to behave as specified and would need to be addressed.

Pressurize

The pressurize signal tells the interlock to fill with water. This can only happen when both the doors are closed. If the pressurize signal is stuck at one, the chamber would fill with water every time both doors are closed. This would be a saftey hazard, because the chamber could pressurize at the wrong time, e. g. when people are inside the chamber. This would also cause the chamber to fill with water directly after being depressurized, which would not allow time for the aquenauts to vacat the chamber. This error would have to be addressed.

Depressurize

This signal tells the interlock to empty the water. If this signal is stuck at one, then the water would be drained right after it filled again. This would never allow the outer door to open and the bathysphere would never be able to leave. This would be an error that would need to be addressed for the system to work correctly.

6.5 Figures

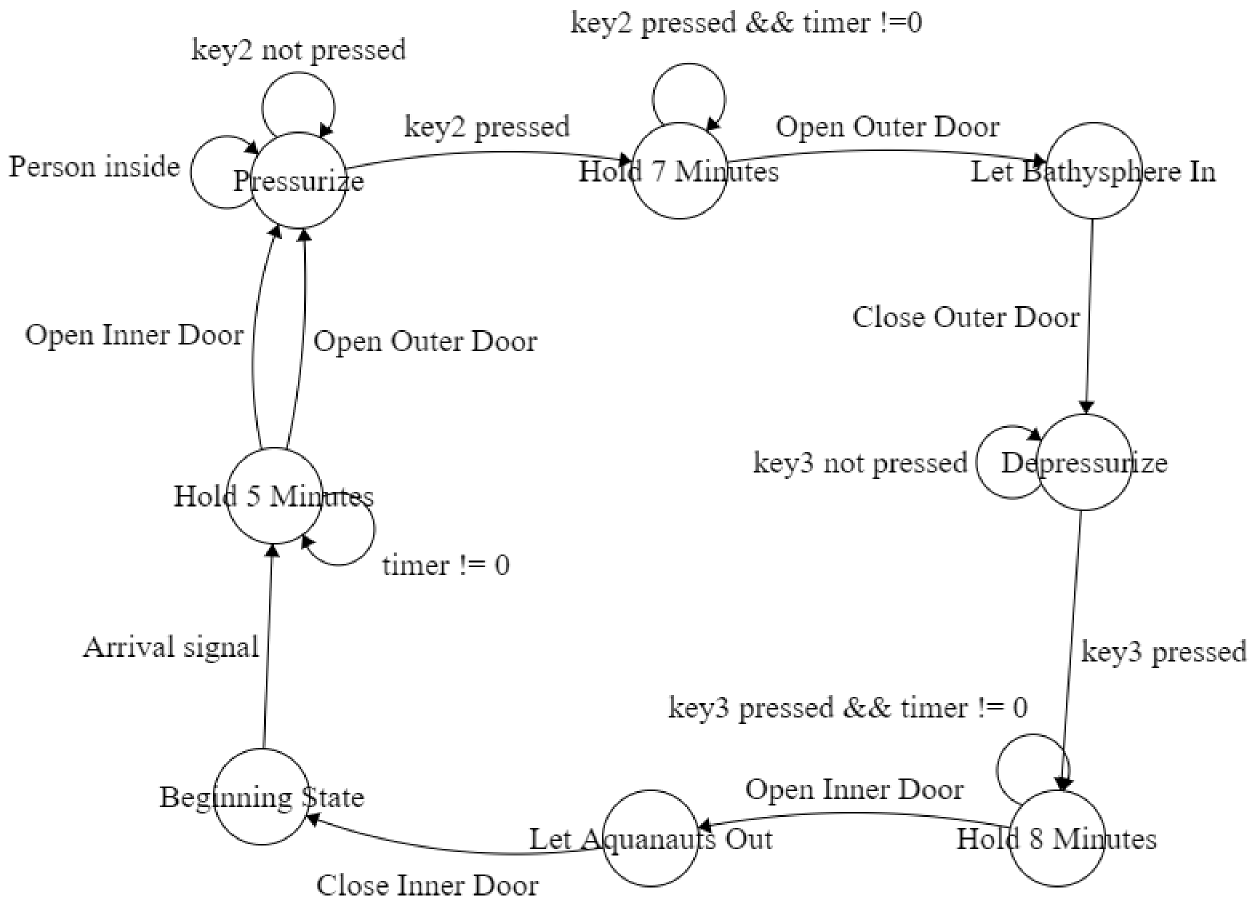


Figure 6.1: Top level diagram of the arrival process of the interlock system

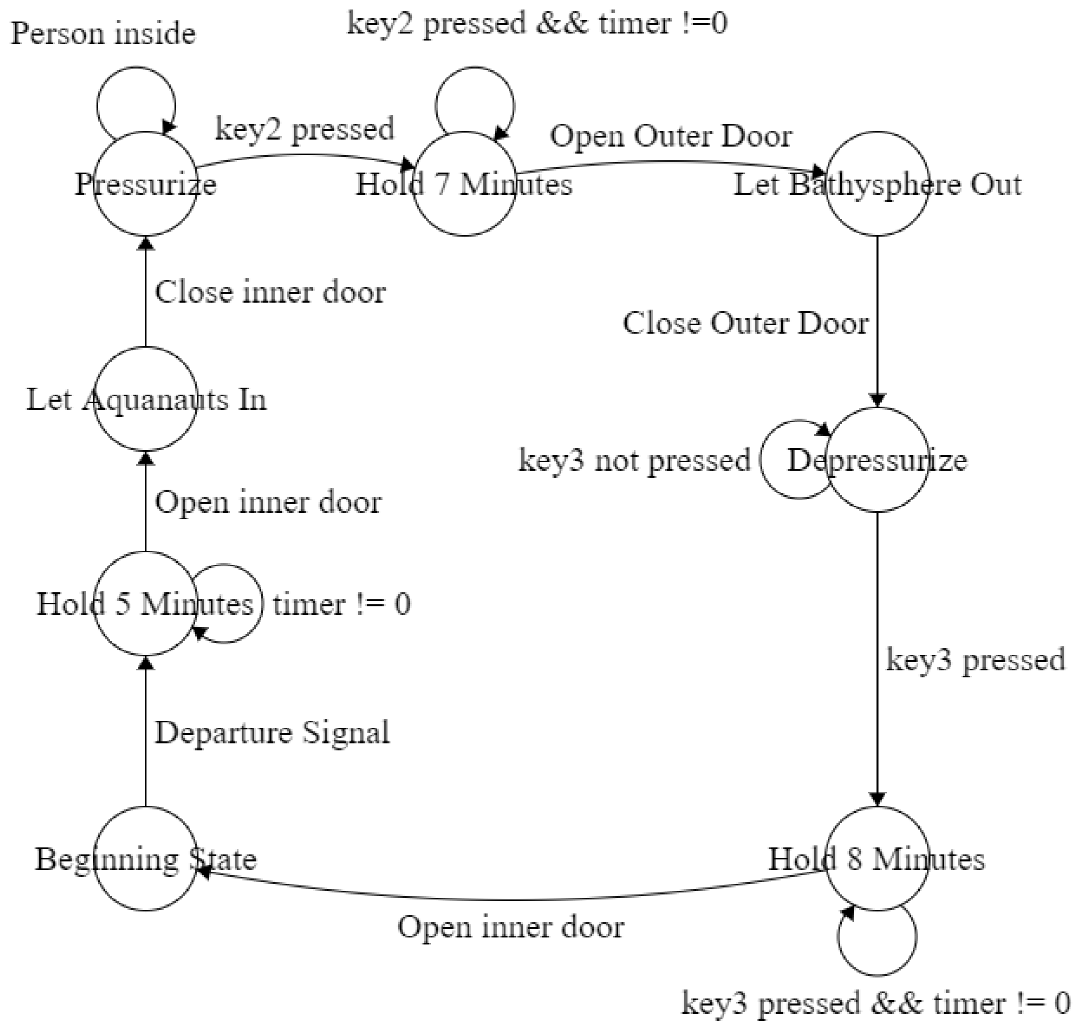


Figure 6.2: Top level diagram of the departure process of the interlock system

6.6 Code

Listing 1: Top module for incorporating the interlock system onto the DE1-SoC board

```
/*Authors : Adolfo Pineda
        Katelyn Elizabeth Neff
        Sharyar Khalid

  This is the main module which calls all other module
  this module calls the interlock module that does all the
  main features of this lab, then it calls the timer module and
  sends it the approriate inputs depending on which state the
  interlock module is. It also calls the display module that outputs
  on the HEX display allowing the user to see what he/she is currently
  doing*/

module main(CLOCK_50, LEDR, SW, HEX0, HEX1, KEY, HEX5, HEX4);

  input [3:0] KEY;
  input [9:0] SW;
  input CLOCK_50;

  output [6:0] HEX0 , HEX1, HEX5, HEX4;
  output [9:0] LEDR;

  wire filling , draining, fillFinished, drainFinished, waiting, waitFinished;
  wire [3:0] drainVal, fillVal, waitVal;
  wire [6:0] letter, number;

  assign HEX1 = letter;
  assign HEX0 = number;

  assign LEDR[9] = SW[9];
  assign LEDR[8] = SW[8];
  assign LEDR[1:0] = SW[1:0];

  reg [25:0] tBase;
  always@(posedge CLOCK_50) begin
    tBase <= tBase + 1'b1;
  end

  // The following module does most of the main functions of this lab
  interlock inter(
              .draining(draining),
              .filling(filling),
              .innerDoor(LEDR[3]),
              .outerDoor(LEDR[2]),
              .resetLeds(LEDR[7:4]),
              .bathLeaving(SW[1]),
              .bathArriving(SW[0]),
              .personCheck(SW[8]),
              .pressureCheck(SW[9]),
              .innerDoorSwitch(SW[3]),
              .outerDoorSwitch(SW[2]),
              .clk(tBase[24]),
              .reset(KEY[0]),
              .drain(KEY[2]),
              .fill(KEY[1]),
              .fillFinished(fillFinished),
              .drainFinished(drainFinished),
              .waiting(waiting),
              .waitFinished(waitFinished)
            );
  // The following displays the output of either pressurizing, waiting or
     depressurizing
  // onto HEX0 and HEX 1
  display disp(
            .letter(letter),
            .num(number),
            .clk(CLOCK_50),
            .drainVal(drainVal),
            .fillVal(fillVal),
            .draining(draining),
            .filling(filling),
            .waiting(waiting),
              .waitVal(waitVal)
            );
    /* The following module initiates when the interlock module
       is in the pressurization state and allows the user to press and
      hold key 1 to immitate the pressurization of the interlock chamber*/
    timer pressurize(
               .val(fillVal),
               .finished(fillFinished),
               .clk(tBase[24]),
               .startingVal(4'b0111),
               .change(!KEY[1]),
               .start(filling)
             );

    /*This module initiates when the interlock module is in the depressurize
      state and allows the user to press and hold key 2 to immitate the
         depressurization
      of the interlock chamber*/
    timer depressurize(
                 .val(drainVal),
                 .finished(drainFinished),
                 .clk(tBase[24]),
                 .startingVal(4'b1000),
                 .change(!KEY[2]),
                 .start(draining)
               );
    /*This module initiate when the bathysphere signals either arriving or
       departing and the
      interlock module goes into the waiting state then a signal to this module
         activates it
      and a countdown goes from 5 to 0 immitating the 5 minute waiting times for
          both departure
      and arrival of the bathysphere */
    timer waiting5Sec(
                 .val(waitVal),
                 .finished(waitFinished),
                 .clk(tBase[24]),
                 .startingVal(4'b0101),
                 .change(1'b1),
                 .start(waiting)
               );

  endmodule
```

Listing 2: The main interlock module

```
/*This is the interlock module where most of the functions of this lab happens
  This module takes care of arriving, departing, checking pressure and
     checking
  if there is a person in the interlock. This module also takes care of the
  pressurization and depressurization of the interlock chamber*/

module interlock(
            filling ,
            draining,
          innerDoor,
          outerDoor,
          resetLeds,
          bathLeaving,
          bathArriving,
          personCheck,
          pressureCheck,
          drain,
          fill,
          innerDoorSwitch,
          outerDoorSwitch,
          clk,
          reset,
          drainFinished,
          fillFinished,
          waitFinished,
          waiting
        );
// The following inputs come from the main module
input bathLeaving,
    bathArriving,
    personCheck,
    pressureCheck,
    drain,
    fill,
    innerDoorSwitch,
    outerDoorSwitch,
    clk,
    reset,
    waitFinished,
    drainFinished,
    fillFinished;
// The following outputs are displayed onto LEDR 3 and 2
output reg innerDoor,
        outerDoor;

/*The following output displays the state in which the interlock
  is currently in, it displays these states on LEDRs 7 to 4 */
output reg [3:0] resetLeds;

/* These following outputs go to the timer module. They act as a
  signal to activate either pressurization, depressurization or
  the waiting timer. These signals activate depending on which
  state the interlock module is in*/
output reg filling,
        draining,
        waiting;



reg [3:0] ps;

wire nReset;
not resetNot (nReset , reset);

// Follow are the states that this module goes through
```

```
parameter [3:0] everythingClosedLow = 4’b0100,
           everythingClosedHigh = 4’b0101;
parameter [3:0] openOuterDoor = 4’b0110,
           openInnerDoor = 4’b0111;
parameter [3:0] outerDoorOpen = 4’b1000,
           outerDoorClose = 4’b1001;
parameter [3:0] innerDoorOpen = 4’b1010,
           innerDoorClose = 4’b1011;
parameter [3:0] pressurizationState = 4’b1100,
           depressurizationState = 4’b1101;
parameter [3:0] checkingPressureAndPerson = 4’b0000,
           wait5Minutes = 4’b0011;

parameter [3:0] innerDooClosedLow = 4’b1110;

parameter [3:0] bufferState = 4’b1111;




always@(posedge clk) begin
// Following if statement checks to see if the user has
// pressed the reset key, which is key 0.
  if (nReset) begin
      ps <= everythingClosedLow;
      resetLeds [3:0] <= everythingClosedLow;
      innerDoor <= 1’b0;
      outerDoor <= 1’b0;
      draining <= 0;
      filling <= 0;
  end else begin
    case (ps)
       // The following state makes sure if the interlock is empty or not
      // It checks if the pressure is already high in the chamber
      // or if there is already a person in there
      checkingPressureAndPerson: begin
        if (!personCheck && !pressureCheck && !bathLeaving) begin
          ps <= pressurizationState;
          innerDoor <= 1’b0;
          outerDoor <= 1’b0;
          resetLeds <= pressurizationState;
        end
        else if (bathLeaving && !pressureCheck && !personCheck)
        begin
              ps <=innerDoorOpen ;
             resetLeds <= innerDoorOpen;
             innerDoor <= 1’b1;
        end
        else if (pressureCheck && bathLeaving)
        begin
             ps <= depressurizationState ;
            resetLeds <= depressurizationState;
            draining <= 1 ;

        end
        else if(!personCheck && pressureCheck)
        begin
             ps <= outerDoorOpen;
            resetLeds <= outerDoorOpen;
            outerDoor <= 1'b1;
         end
        else if ( personCheck && !pressureCheck)
        begin
              ps <= innerDoorOpen;
             resetLeds <= innerDoorOpen;
             innerDoor <= 1'b1;
        end
      end

      // The following state is the initial state that the interlock module
      // goes into and then wait for either the arrival signal or departure
         signal
      // when either of those are activated the interlock module goes into
         the
      // wait5 module where the 5 minute countdown begins.
      everythingClosedLow: begin
        if (bathArriving) begin
            ps <= wait5Minutes;
            waiting <= 1'b1;
            resetLeds <= wait5Minutes;
        end else
        if(bathLeaving) begin
            ps <= wait5Minutes;
            innerDoor <= 1'b0;
            outerDoor <=1'b0;
            waiting <= 1'b1;
            resetLeds <= wait5Minutes;
        end else
        if (innerDoorSwitch) begin
          ps <= everythingClosedLow;
          innerDoor <= 1'b1;
          resetLeds <= everythingClosedLow;
        end
      end

      // This is the state when the interlock is depressurizzed and the
         inner
      // door is closed
      innerDooClosedLow: begin
        if (!innerDoorSwitch) begin
          innerDoor <=1'b1;
          ps <= innerDooClosedLow;
          resetLeds <= innerDooClosedLow;
        end else
        if (innerDoorSwitch) begin
          innerDoor <= 1'b1;
          ps <= innerDoorOpen;
          resetLeds <= innerDoorOpen;
        end
        end
```

```
// The interlock module arrives in this state when the chamber is
   pressurized
// and the outer door is closed, as well as the inner door.
everythingClosedHigh: begin
  if (bathArriving && !outerDoor) begin
    ps <= outerDoorOpen;
    outerDoor <= 1'b1;
    resetLeds <= outerDoorOpen;
  end
  else if (bathLeaving) begin
    ps <= outerDoorOpen;
    outerDoor <= 1'b1;
    resetLeds <= outerDoorOpen;
  end
end

// The onterlock module arrives in this state when the interlock is
// pressurized and the outer door is open
openOuterDoor: begin
  if (outerDoorSwitch) begin
    ps <= openOuterDoor;
    outerDoor <= 1'b1;
    resetLeds <= openOuterDoor;
  end else if (!outerDoorSwitch) begin
    outerDoor <= 1'b0;
    ps <= outerDoorClose;
    resetLeds <= outerDoorClose;
  end
end

// The inter lock comes to this module if the innerDoor is opened
openInnerDoor: begin
  if (!bathLeaving) begin
    ps <= innerDoorClose;
    innerDoor <= 1'b0;
    resetLeds <= innerDoorClose;
  end
end

//The interlock goes to this state if the bathysphere is arriving and
// the outer door is not open or if the bathysphere is about to depart
   .
outerDoorOpen: begin
  if (outerDoorSwitch && !personCheck) begin
    ps <= openOuterDoor;
    resetLeds <= openOuterDoor;
  end
end

// The interlock arrives to this module after the outer door is open
   and
// stays in here until the outer door is closed then it goes to
   depressurization
// state
outerDoorClose: begin
  if (!outerDoorSwitch && !personCheck) begin
```

```
    ps <= depressurizationState;
    draining <= 1'b1;
    resetLeds <= depressurizationState;
  end
end

// The interlock arrives in this state after the inner door is open
// and stays in here until the inner door is closed.
innerDoorOpen: begin
  if (!innerDoorSwitch) begin
    ps <= innerDoorOpen;
    innerDoor = 1'b1;
    resetLeds = innerDoorOpen;
  end
  else if (innerDoorSwitch)
  begin
       ps<= innerDoorClose;
      innerDoor <= 1'b1;
      resetLeds <= innerDoorClose;
  end
end

// Arrives in this state after the inner door is open and then closed
innerDoorClose: begin
        if (!innerDoorSwitch && bathLeaving && !bathArriving)
      begin
            ps <= pressurizationState;
          resetLeds <= pressurizationState;
          innerDoor <= 1'b0;
      end
      else if (!innerDoorSwitch && bathArriving && !bathLeaving)
      begin
          ps <= pressurizationState;
          resetLeds <= pressurizationState;
           innerDoor <= 1'b0;
      end
  else if (!innerDoorSwitch && !personCheck) begin
    ps <= bufferState;
    innerDoor <= 1'b0;
    resetLeds <= bufferState;
  end
end
// this is the state where depressurization is activated
depressurizationState: begin
  if (drainFinished) begin
    ps <= innerDooClosedLow;
    draining <= 1'b0;
    resetLeds <= innerDooClosedLow;
  end
end

// This is the state where pressurization is activated
pressurizationState: begin
  filling<=1'b1;
  if (fillFinished) begin
    ps <= everythingClosedHigh;
```

```
            filling <= 1'b0;
            resetLeds <= everythingClosedHigh;
          end
        end

        // This is the state where the timer for waiting 5 mminutes is
           activated
        wait5Minutes: begin
          if (waitFinished) begin
             if (bathArriving || bathLeaving)
            begin
                  ps <= checkingPressureAndPerson;
                resetLeds<= checkingPressureAndPerson;
            end
            waiting <= 1'b0;
          end
        end

        bufferState: begin
          if (bathArriving) begin
            ps <= bufferState;
            resetLeds <= bufferState;
          end else
          if (bathLeaving) begin
            ps <= bufferState;
            resetLeds <= bufferState;
          end else begin
            ps <= everythingClosedLow;
            resetLeds <= everythingClosedLow;
          end
        end

        default: resetLeds [3:0] <= 4'b0001;

      endcase
    end
  end

endmodule
```

Listing 3: The module that handles displaying the timers on the HEX boards

```
/* This is the display module where all the outputs on HEX 0 and HEX 1 are
   produced
   this takes values from the main which acts as a signal for this module to
      know which
   case to activate. */

module display (letter , num , clk , drainVal , fillVal , draining , filling ,
     waiting , waitVal);

  output reg [6:0] letter , num;
  input clk , draining , filling , waiting;
  input [3:0] drainVal , fillVal , waitVal;
```

```
always @(posedge clk)
//The following case keeps hex off if the interlock module
//is neither in the waiting, pressurization nor depressurization state
begin
  if(!draining && !filling && !waiting)
    begin
      letter <= ~7’b0000000;
      num <= ~7’b0000000;
    end
  else
  begin

    /* The following if statement activates if the interlock module is in
       the
      depressurization state and it sends the draining signal as true
      it starts to display from number 8. These numbers are send by the
         timer module
      and they decrease by 1 every clock cycle as long as the user is
         holding down
      key 2 */
    if(draining)
      begin
      letter <= ~7’b1011110;
      case(drainVal)

        // Light: 6543210
        0: num <= ~7’b0111111;
        1: num <= ~7’b0000110;
        2: num <= ~7’b1011011;
        3: num <= ~7’b1001111;
        4: num <= ~7’b1100110;
        5: num <= ~7’b1101101;
        6: num <= ~7’b1111101;
        7: num <= ~7’b0000111;
        8: num <= ~7’b1111111;
        default: num <= 7’bX;
      endcase
      end

    /* The following else if statement activates if the interlock module is
       in the
      pressurization state and it sends the filling signal as true
      it starts to display from number 7. These numbers are send by the
         timer module
      and they decrease by 1 every clock cycle as long as the user is
         holding down
      key 1 */
    else if(filling)
      begin
      letter <= ~7’b1110011;
      case(fillVal)
        // Light: 6543210
        0: num <= ~7’b0111111;
        1: num <= ~7’b0000110;
        2: num <= ~7’b1011011;
```

```
          3: num <= ~7'b1001111;
          4: num <= ~7'b1100110;
          5: num <= ~7'b1101101;
          6: num <= ~7'b1111101;
          7: num <= ~7'b0000111;
          8: num <= ~7'b1111111;
          default: num <= 7'bX;
        endcase
        end
      else
        begin
        letter <= ~7'b1000000;

      /* The following case activates if the interlock module is in the
         waiting state which is activated by the arrival or departure signal
            it
         stays in this case statement until it reaches the 0 number from 5 and
             sends
         a signal back to timer to stop and this case also ends .*/
        case(waitVal)
          // Light: 6543210
          0: num <= ~7'b0111111;
          1: num <= ~7'b0000110;
          2: num <= ~7'b1011011;
          3: num <= ~7'b1001111;
          4: num <= ~7'b1100110;
          5: num <= ~7'b1101101;
          6: num <= ~7'b1111101;
          7: num <= ~7'b0000111;
          8: num <= ~7'b1111111;
          default: num <= 7'bX;
        endcase
        end
      end
    end


endmodule
```

Listing 4: The module that handles the timers that count minutes

```
/*This module is activated when either the departure or the arrival signal is
   activated
  in which case it will count down from 5 to 0. In the case of
     depressurization it will countdown
  from 8 to 0, and in the case of pressurization it will count down from 7 to
     0 */
module timer(val , finished , clk , startingVal , change , start);

  output reg [3:0] val;
  output reg finished;
  input clk , change , start;
  input [3:0] startingVal;
  always@(posedge clk)
  begin
    if(!start)
    begin
      val <= startingVal;
      finished <= 0;
    end
    /* when the module has counted down to 0 it sends a true finish signal
       back
       which allows the interlock module to exit either the wait5, timer fill
           or
      timer drain module */
    else if(val == 4'b0000)
      finished <= 1;

    /*The following module keeps subtracting 1 from the value that was sent by
        the main module
      in the case of wait its 5 in the case of pressurize its 7 and in the
          case of depressurize its 8*/
    else if(change)
      val <= val - 4'b0001;
  end
endmodule
```

Listing 5: Testing module for arrival

```
`include "main.v"
`include "timer.v"
`include "display.v"
`include "interlock.v"

module DE1_Top;

  wire [9:0] SW , LEDR;
  wire [3:0] KEY;
  wire [6:0] HEX0 , HEX1, HEX4, HEX5;
  wire CLOCK_50, wait_var, finished_var;

  main DE1board (.CLOCK_50(CLOCK_50),
           .LEDR(LEDR),
           .SW(SW),
           .KEY(KEY),
           .HEX0(HEX0),
           .HEX1(HEX1),
           .HEX4(HEX4),
           .HEX5(HEX5),
           .wait_var(wait_var),
           .finished_var(finished_var)
           );

  tester DE1Test (.clk(CLOCK_50),
          .key(KEY),
          .sw(SW),
          .led(LEDR),
          .wait_var(wait_var),
        .finished_var(finished_var)
        );

initial begin
  $dumpfile("de1_top.vcd");
  $dumpvars(1, DE1board);
end
endmodule




module tester(clk, key, sw, led, wait_var, finished_var);

  output reg clk;
  output reg [3:0] key;
  output reg [9:0] sw;
  input [9:0] led;
  //input [6:0] HEX0;
  input wait_var, finished_var;

    parameter stimDelay = 10;

  initial // Response
  begin
    $display("\t clk \t key \t sw \t\t led \t\t Time");
    $monitor("\t %b \t %b \t %b \t %b \t %g" , clk, key, sw, led, $time);

  end

  initial //Stimulus

  begin
    //#stimDelay;
    //#stimDelay;
    sw = 9'b0; key[3:0] = 4'b1111; clk = 1'b0;
    #stimDelay;
    #stimDelay clk = 1'b1; key[0] = 0;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1; key[0] = 1;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1; sw[0] = 1;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1;
    #stimDelay clk = 1'b0;
    #stimDelay;
    #stimDelay clk = 1'b1;
    //#stimDelay key[0] = 1'b0 ; sw[0] = 0;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1;
    //#stimDelay key[0] = 1'b1;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1;
    #stimDelay clk = 1'b0;
```

```
#stimDelay clk = 1'b1;
//#stimDelay sw[1] = 1'b1;
#stimDelay clk = 1'b0;
#stimDelay clk = 1'b1;
#stimDelay clk = 1'b0;
#stimDelay clk = 1'b1;
#stimDelay clk = 1'b0;
#stimDelay clk = 1'b1;
#stimDelay clk = 1'b0;
#stimDelay clk = 1'b1;
#stimDelay clk = 1'b0;
#stimDelay clk = 1'b1; key[1] = 0;
#stimDelay clk = 1'b0;
#stimDelay clk = 1'b1;
#stimDelay clk = 1'b0;
#stimDelay clk = 1'b1;
#stimDelay clk = 1'b0;
#stimDelay clk = 1'b1;
#stimDelay clk = 1'b0;
#stimDelay clk = 1'b1;
#stimDelay clk = 1'b0;
#stimDelay clk = 1'b1;
#stimDelay clk = 1'b0;
#stimDelay clk = 1'b1;
#stimDelay clk = 1'b0;
#stimDelay clk = 1'b1;
#stimDelay clk = 1'b0;
#stimDelay clk = 1'b1;
#stimDelay clk = 1'b0;
#stimDelay clk = 1'b1;
#stimDelay clk = 1'b0;
#stimDelay clk = 1'b1;
#stimDelay clk = 1'b0;
#stimDelay clk = 1'b1; key[1] = 1;
#stimDelay clk = 1'b0;
#stimDelay clk = 1'b1;
#stimDelay clk = 1'b0;
#stimDelay clk = 1'b1; sw[2] = 1;
#stimDelay clk = 1'b0;
#stimDelay clk = 1'b1;
#stimDelay clk = 1'b0;
#stimDelay clk = 1'b1; sw[2] = 0;
#stimDelay clk = 1'b0;
#stimDelay clk = 1'b1;
#stimDelay clk = 1'b0;
#stimDelay clk = 1'b1; key[2] = 0;
#stimDelay clk = 1'b0;
#stimDelay clk = 1'b1;
#stimDelay clk = 1'b0;
#stimDelay clk = 1'b1;
#stimDelay clk = 1'b0;
#stimDelay clk = 1'b1;
#stimDelay clk = 1'b0;
#stimDelay clk = 1'b1;
#stimDelay clk = 1'b0;
#stimDelay clk = 1'b1;
```

```
    #stimDelay clk = 1’b0;
    #stimDelay clk = 1’b1;
    #stimDelay clk = 1’b0;
    #stimDelay clk = 1’b1;
    #stimDelay clk = 1’b0;
    #stimDelay clk = 1’b1;
    #stimDelay clk = 1’b0;
    #stimDelay clk = 1’b1;
    #stimDelay clk = 1’b0;
    #stimDelay clk = 1’b1;
    #stimDelay clk = 1’b0;
    #stimDelay clk = 1’b1;
    #stimDelay clk = 1’b0;

  $finish;

  end


endmodule
```

Listing 6: Testing module for departure

```
`include "main.v"
`include "timer.v"
`include "display.v"
`include "interlock.v"

module DE1_Top;

  wire [9:0] SW , LEDR;
  wire [3:0] KEY;
  wire [6:0] HEX0 , HEX1, HEX4, HEX5;
  wire CLOCK_50, wait_var, finished_var;

  main DE1board (.CLOCK_50(CLOCK_50),
           .LEDR(LEDR),
           .SW(SW),
           .KEY(KEY),
           .HEX0(HEX0),
           .HEX1(HEX1),
           .HEX4(HEX4),
           .HEX5(HEX5),
           .wait_var(wait_var),
           .finished_var(finished_var)
           );

  tester DE1Test (.clk(CLOCK_50),
          .key(KEY),
          .sw(SW),
          .led(LEDR),
          .wait_var(wait_var),
          .finished_var(finished_var)
          );
  initial begin
    $dumpfile("de1_top.vcd");
    $dumpvars(1, DE1board);
  end
  endmodule




module tester(clk, key, sw, led, wait_var, finished_var);

  output reg clk;
  output reg [3:0] key;
  output reg [9:0] sw;
  input [9:0] led;
  //input [6:0] HEX0;
  input wait_var, finished_var;

    parameter stimDelay = 10;

  initial // Response
  begin
    $display("\t clk \t key \t sw \t\t led \t\t Time");
    $monitor("\t %b \t %b \t %b \t %b \t %g" , clk, key, sw, led, $time);

  end

  initial //Stimulus

  begin
    //#stimDelay;
    //#stimDelay;
    sw = 9'b0; key[3:0] = 4'b1111; clk = 1'b0;
    #stimDelay;
    #stimDelay clk = 1'b1; key[0] = 0;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1; key[0] = 1;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1; sw[1] = 1;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1; key[1] = 0;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1; key[1] = 1;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1; //sw[3] = 1;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1; //sw[3] = 0;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1; //key[1] = 0;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1; //key[2] = 0;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1;
    #stimDelay clk = 1'b0;
    #stimDelay clk = 1'b1;
    #stimDelay clk = 1'b0;

  $finish;

  end
```

```
endmodule
```

Listing 7: The C program about pointers

```
#include <stdio.h>
#include <stdlib.h>

int main ()
{
  /* Part 1 */
    /*Declare several variables of the following types:
    two variables of type int, two
    variables of type float, and two variables of type char.*/
    int a = 5;
    int b = 17;
    float c = 3.14;
    float d = -5128823.0;
    char e ='e';
    char f = 'f';

  /*Declare the following pointer type variables:
    one pointer to an int, one pointer to
      a float, and one pointer to a char.*/
    int* xPtr;
    float* yPtr;
    char* zPtr;

  /*Assign the address of one of the integer variables
    to the pointer to int. Print out
    the value of that integer.*/
    xPtr = &a;
    printf("The value of a = %d\n", *xPtr);
    printf("The address of a = %d\n\n", xPtr);

    //Repeat with the second integer.
    xPtr = &b;
    printf("The value of b = %d\n", *xPtr);
    printf("The address of b = %d\n\n", xPtr);

    //Repeat steps 3 and 4 with the two floats and then the two chars.
    yPtr = &c;
    printf("The value of c = %f\n", *yPtr);
    printf("The address of c = %d\n\n", yPtr);

      yPtr = &d;
      printf("The value of d = %f\n", *yPtr);
    printf("The address of d = %d\n\n", yPtr);

      zPtr = &e;
      printf("The value of e = %c\n", *zPtr);
    printf("The address of e = %d\n\n", zPtr);

      zPtr = &f;
      printf("The value of f = %c\n", *zPtr);
    printf("The address of f = %d\n\n", zPtr);
    //Part 2 âĂŞ Working with Pointer Variables
    // Declare and define variables
      int A = 22;
      int B = 17;
      int C = 6;
      int D = 4;
      int E = 9;
      int result = 0;

  /*declare and define five variables of type pointer to
    integer and let each refer to one
      of the variables.*/
      int* aPtr = &A;
      int* bPtr = &B;
      int* cPtr = &C;
      int* dPtr = &D;
      int* ePtr = &E;

  /*perform the computation:result = ((A âĂŞ B)*(C+D))/E
      only instead of using the variables directly, refer
    to each through its pointer*/
    result = ((*aPtr - *bPtr) * (*cPtr + *dPtr)) / (*ePtr);
    printf("result = %d\n\n", result);

  return 0;



}
```

6.7 Division of Work

Table 6.1: Division of work among group members

Name	Tasks	Hours
Katie Neff	Testing and debugging Verilog and C program, Signal Tap, conducting error analysis, writing report	20
Adolfo Pineda	Testing and debugging Verilog and C program, developing testing strategy and analyzing results, writing report	20
Sharyar Khalid	Writing Verilog/C program, lab report, testing and debugging, adding extra features, writing report	20