EE 371 Autumn 2016 - Lab 2

William Li, Dawn Liang, Jun Park

November 2, 2016

signatures We certify that the work in this report is our own, and that any work that is not ours is cited.

William Li	Dawn Liang	Jun Park
_____	_____	_____
Signature	*Signature*	*Signature*
_____	_____	_____
Date	*Date*	*Date*

contributions Jun worked on designing the project, drawing diagrams and building state machines, wrote iterations of the modules, and wrote the test part of the write-up. William wrote the C program and the write-up for the C program, wrote the poundOccupied module. Dawn worked on designing the project, wrote the overall module and the final iteration of the other modules, wrote the Requirements & Design Spec and Functional Decomposition, and compiled and formatted the final write-up.

Contents

1 ABSTRACT

In this lab, we were introduced to our first digital design project. We designed and built a simple time-dependent system, a poundlock control system. We wrote requirement and design specifications as well as a functional decomposition for the VHDL program, to help us organise the design process. We then wrote our modules in Verilog and tested our design in simulation using iverilog and gtkwave as well as in hardware on the Altera DE1-SoC board. Additionally, we furthered our knowledge of the C language by writing a C program using pointers. These projects helped us develop our skills in designing hardware and software programs.

2 INTRODUCTION

The first part of the lab focuses on building systems in VHDL (Verilog Hardware Description Language). We decomposed the system at a high level in terms of requirements, design, and functions, and wrote specifications for each. This helped us decide the components of our design. We built and tested each submodule in simulation using iverilog and gtkwave, and later integrated them together and simulated them again. Finally, we downloaded the system via Quartus to the DE1-SoC board and checked its behaviour in hardware. We used Signal Tap II Logic Analyser to debug our system and verify proper functionality. In the second part of the lab, we wrote a simple C program using pointer references, to introduce and familiarise ourselves with the concepts.

3 DISCUSSION

3.1 Design

3.1.1 Design Specification - VHDL

Abstract At various points along the canal system of Venice, gondolas must move between areas of significantly differing water levels. In order to do so smoothly, pound locks are in place to facilitate both transportation of gondolas and maintenance of water levels. These locks are controlled by a digital system we have designed and built.

Introduction The system for controlling locks will allow gondolas to pass between areas of the canal of uneven water levels. Upon arrival, the gondola must signal the lock to initiate the process, enter the lock, wait as it is raised/lowered, and then exit the lock to continue its journey. The opening/closing of gates and raising/lowering of water levels are controlled by a lock operator.

Inputs to the system

User Inputs

- fiveMinsTillArrival: signal from the gondola that it will be arriving in at least five minutes, to prompt the lock to adjust water levels and open the gates (1-bit trigger true/false)
  - SA0: the lock will not know when there is a boat arriving, so it will never prepare the arrival gates for entry.
  - SA1: the lock will behave as if there is always another boat in the queue. It will function as normal, except that after each boat exits, the lock will immediately prepare itself for another boat to enter.
- incrWaterLevel: control signal to increase the water level inside the lock, to bring the water levels inside and outside the lock close enough that the gates can open (1-bit trigger true/false)
  - SA0: the lock will never fill the lock up to the higher outside water level
  - SA1: the lock will be constantly filled to the higher outside water level
- decrWaterLevel: control signal to decrease the water level inside the lock, to bring the water levels inside and outside the lock close enough that the gates can open (1-bit trigger true/false)

  - SA0: the lock will never drain the lock down to the lower outside water level
  - SA1: the lock will be constantly drain to the lower outside water level
- reset: empties the lock to the lower outside water level, lets the boats out, and seals the gates (1-bit switch true/false)
  - SA0: the lock will not be able to reset, but otherwise function as normal
  - SA1: the lock will be constantly in the reset state (no boat, drained, gates sealed)

Environmental controls

- arrivOutsideLevel: the water level outside the lock, on the arrival side, determining whether the arrival gate can open (4-bit binary 0-15, each unit corresponds to 0.3125ft)
- deptOutsideLevel: the water level outside the lock, on the departure side, determining whether the departure gate can open (4-bit binary 0-15, each unit corresponds to 0.3125ft)
  - If any of the bits fault for either signal, the range of possible values is restricted to those attainable with the stuck bit

Internal signals

- poundOccupied: whether there is currently a gondola in the lock, since if the lock is occupied, the next boat cannot enter (1-bit true/false)
  - SA0: the lock will always act as if there is not a boat in the lock,
  - SA1: the lock will always act as if there is a boat in the lock, so it will not open the arrival gates to let boats enter
- insideWaterLevel: the water level inside the lock, determining whether the gates can open (4-bit binary 0-15, each unit corresponds to 0.3125ft)
  - If any of the bits fault, the range of possible values is restricted to those attainable with the stuck bit
- arrivGateOpen: whether the door on the arrival side is open (1-bit true/false)
- deptGateOpen: whether the door on the departure side is open (1-bit true/false)
  - SA0: If either gate is sealed, boats cannot enter/exit the lock
  - SA1: If either gate is open, the lock cannot adjust its inside water level

Outputs of major functions

- gateOpenClose: opens/closes the lock gates, allowing gondolas to enter/exit the lock (1-bit open/closed gate)
  - SA0: the gates are constantly closed, not allowing any boats through the lock
  - SA1: the gates are constantly open, unable to adjust the water level inside the lock and thus not allowing any boats through
- incrWaterLevel: increases the water level inside the lock, matching it to the outside higher water level so that the gate can open (4-bit binary 0-15, each unit corresponds to 0.3125ft change in the inside water level)
- decrWaterLevel: decreases the water level inside the lock, matching it to the outside higher water level so that the gate can open (4-bit binary 0-15, each unit corresponds to 0.3125ft change in the inside water level)
  - The lock can adjust its water level by up to 5ft (each bit corresponds to 0.3125 ft)
  - The lock takes up to 7 minutes to drain, 8 minutes to fill (each clock edge corresponds to 30s)
  - If any of the bits fault, the range of possible values is restricted to those attainable with the stuck bit, so the lock potentially cannot fill/drain sufficiently to open the gates

3.1.2 Design Specification - C Program

Abstract The project2.c program is a project that teaches us the fundamentals of pointers and references in the C programming language. Written in C and in the CodeBlocks IDE, our program contained hard coded values which are then processed into a simple equation. The result is then outputted to the user. Our program results matched our expectations, outputting the correct answer every time.

Introduction This project gave us a continuation to our C programming knowledge and experience working in the CodeBlocks IDE. The result of this project was a program that calculates the answer of the equation from a few hardcoded values (as described in the specifications) and outputs the value to the user.

Inputs to the system There were no inputs into this program.

Outputs of major functions The output value of the program is an integer value, which in this case is always 5.

Major functions The project2.c program calculates the solution to the expression $((A + B) \times (C - D))/E$, where A, B, C, D, and E are hardcoded to be 22, 17, 6, 4 and 9 respectively.

3.1.3 Design Procedure

VHDL Design The basic functionality of the lock was decomposed into an overall central module, opening/closing the gates, and adjusting the water level inside the lock. Each binary unit for water levels corresponds to 0.3125ft, for modeling water level heights; each clock cycle corresponds to half a minute, for modeling timing.

Functional Decomposition

- Monitoring overall variables for the lock, like its state & external interactions
  - locations of boats
  - interactions with boats
  - reset
- Open/close gates: the gates open/close to let boats enter/exit the lock
  - Input gate-opening conditions must be met (boat incoming/departing, comparing the water level inside the gate to the water level outside the gate, lock empty in the case of the arrival gate)
  - the gates are not FSMs, since their output does not depend on their present state; implemented using combinational logic
- Adjust water levels: the lock fills/drains to match inside and outside water levels, so that the gates can open
  - Keeping track of and updating the inside water level for draining/filling the lock
  - Filling/draining is bounded by the external water levels
  - Input fill/drain conditions must be met (gates closed, receive signal)
  - Water level matches counter behaviour very closely, so we used an enable-counter that counts up/down to min/max values based on inputs

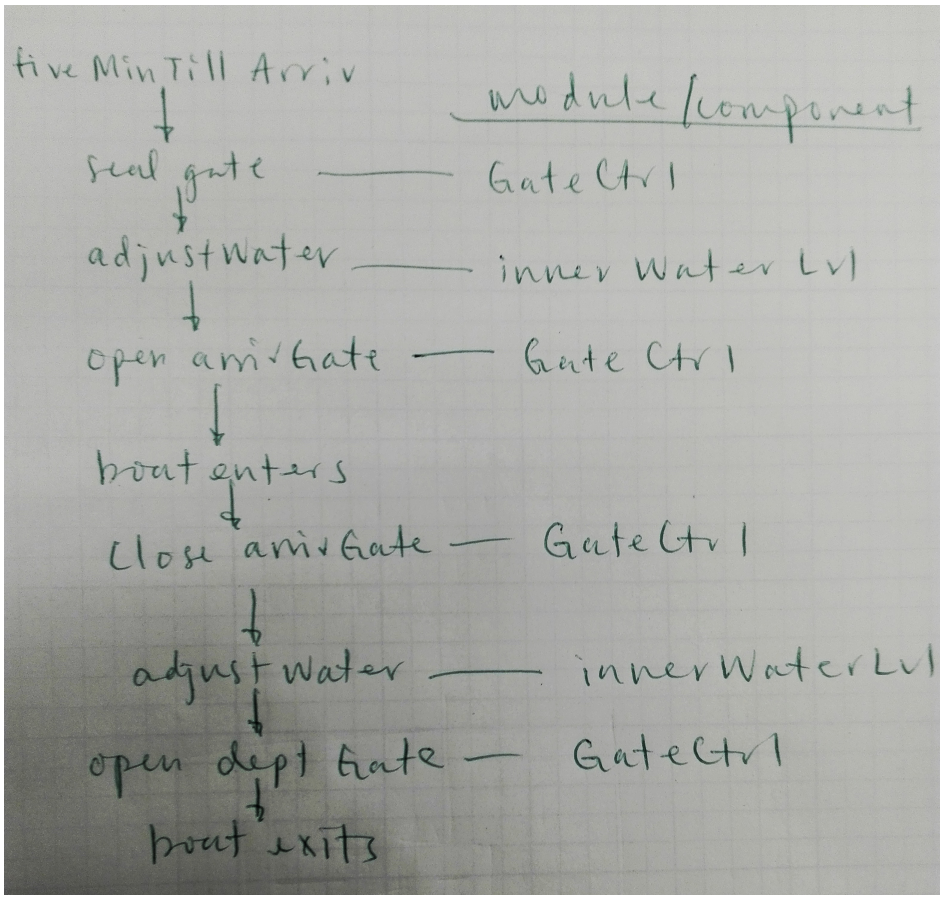

Figure 1: Functional flowchart

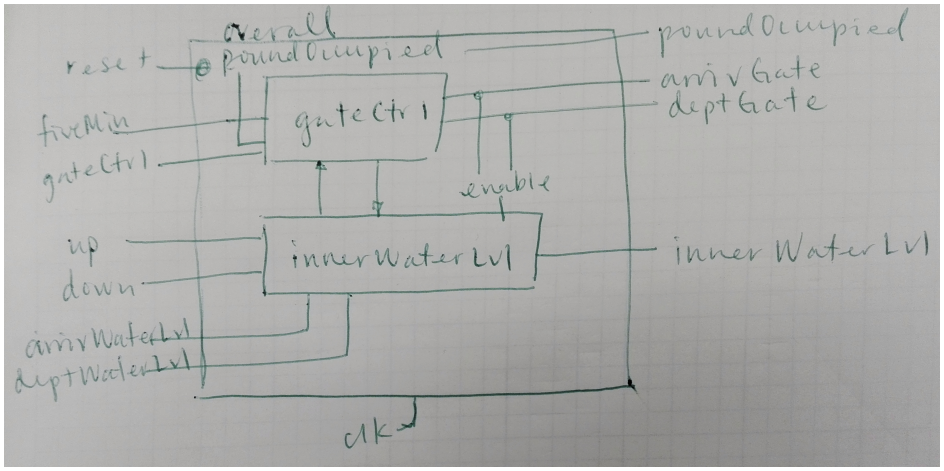

Figure 2: Functional block diagram

C Program The C program was designed as a single unit with no additional methods. Since we were required to have very specific inputs into our program, these values were hardcoded into each of the respective variables (A = 22, B = 17, C = 6, D = 4, E = 9). We then made pointers (X1 to X5), and assigned them to each of the variables respectively. From there, we used the pointers in our calculations.

3.1.4 System Description

VHDL Design Our system had three major components: an overall system management module, an innerWaterLvl module, and combinational logic for controlling the gates.

The overall module managed inputs/outputs to the system and clock division, connected submodules, and used a finite state machine to track the status of the lock.

- Inputs: fiveMinTillArrival, arrivOutsideLvl, deptOutsideLvl, incr, decr, gateCtrl, clock, reset
- Outputs: insideWaterLvl, arrivGate, deptGate, poundOccupied

The overall module handled inverting key inputs (1 for pressed, 0 for unpressed), clock division, an FSM for monitoring the state of the lock (occupied/unoccupied), and connecting up all the submodules. The overall module had to determine which water level was higher in order to connect inputs to the innerWaterLvl module. When reset, the insideWaterLvl becomes 0 and the lock becomes unoccupied.

The innerWaterLvl module modeled the water level inside the lock using a counter that counts up/down to a min/-max based on inputs, and pauses when disabled

- Inputs: enable, min, max, incr, decr, clock, reset
- Outputs: insideWaterLvl

The counter is told whether to count up/down, and counts up to a max value/down to a min value. It pauses when disabled (enable = 0), and resets to 0. The incr/decr signals must be held down in order to change the water level, and held across a clock edge. Since the water levels vary between 0 and 15bits (0 & 5ft) and the lock fills/drains at 1bit/clockedge (0.3125ft/30s), the lock takes up to 7 minutes to drain/fill.

The gates were implemented using combinatorial logic

- Inputs: fiveMinTillArrival, arrivOutsideLvl, deptOutsideLvl, insideWaterLvl, poundOccupied
- Outputs: arrivGate/deptGate (whether the gate is open/closed)

The arrival and departure gates both open only when the difference between the relevant outer and inner water levels are within 0.3ft of each other. The departure gate opens when prompted, the arrival gate must also have a fiveMinTillArrival signal and the lock must be unoccupied. The gates must be held open for at least two clock edges in order for the boat to enter/exit the lock, and if the lock is occupied, the arrival gate wil not open.

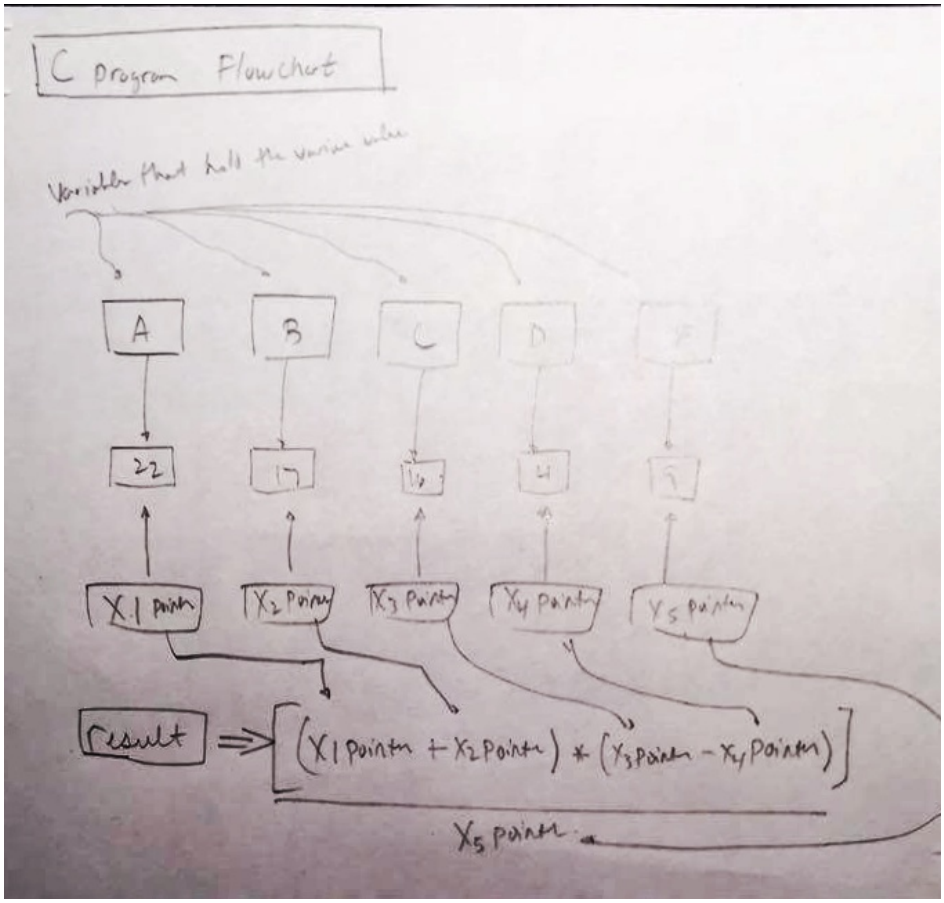

Figure 3: C program design specification

3.1.5 Software Implementation

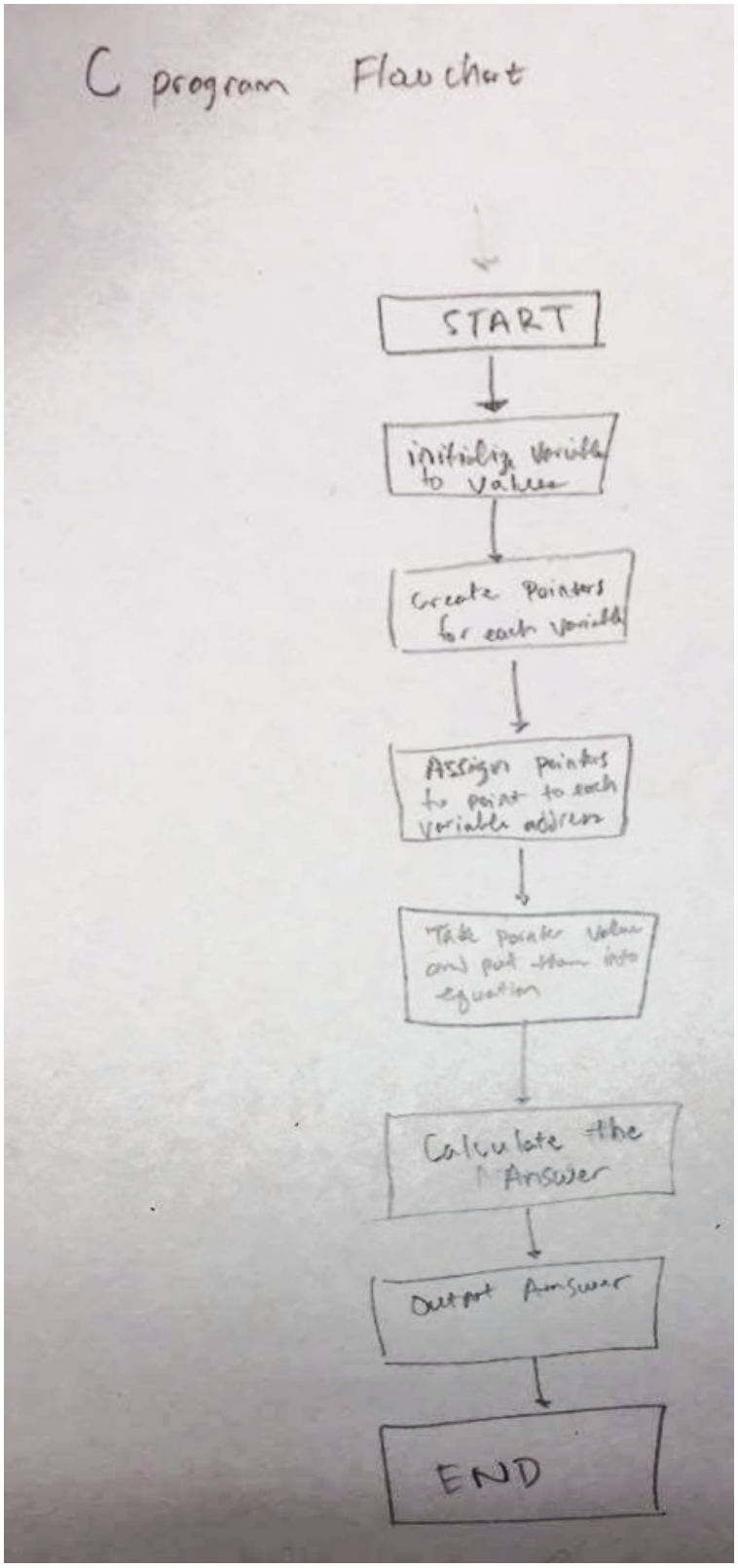

Figure 4: C program information flowchart

The project2.c program was written in C. In the program, we created only one method (the main), where we declared six variables (A, B, C, D, E, result), and initialized each variable to its respective hardcoded value (A = 22, B = 17, C = 6, D = 4, E = 9). The last variable, result, gets overwritten with the result of the calculations. We then created five pointer variables (labeled X1 to X5) and made each pointer point to the addresses of each variables (X1 $\rightarrow$ A, X2 $\rightarrow$ B, ... X5 $\rightarrow$ E). We then plug the values that are referenced by the pointers into the equation:

$$\frac{((A - B) \times (C + D))}{E}$$

The answer to the equation is then written into the result variable and outputted to the user. Since the program has no inputs, the outputted value is always 5.

3.1.6 Hardware Implementation

Our system was divided into three main functions: a top-level overall system management module, gate control, and adjusting the inner water levels. The overall system handled inputs/outputs and clock timing, kept track of the state of the lock, and connected submodules. To keep track of the state of the lock (poundOccupied signal), the overall module used a finite state machine. Information about the insideWaterLvl and state of the gates was passed between the two submodules using connections in the overall top-level module.

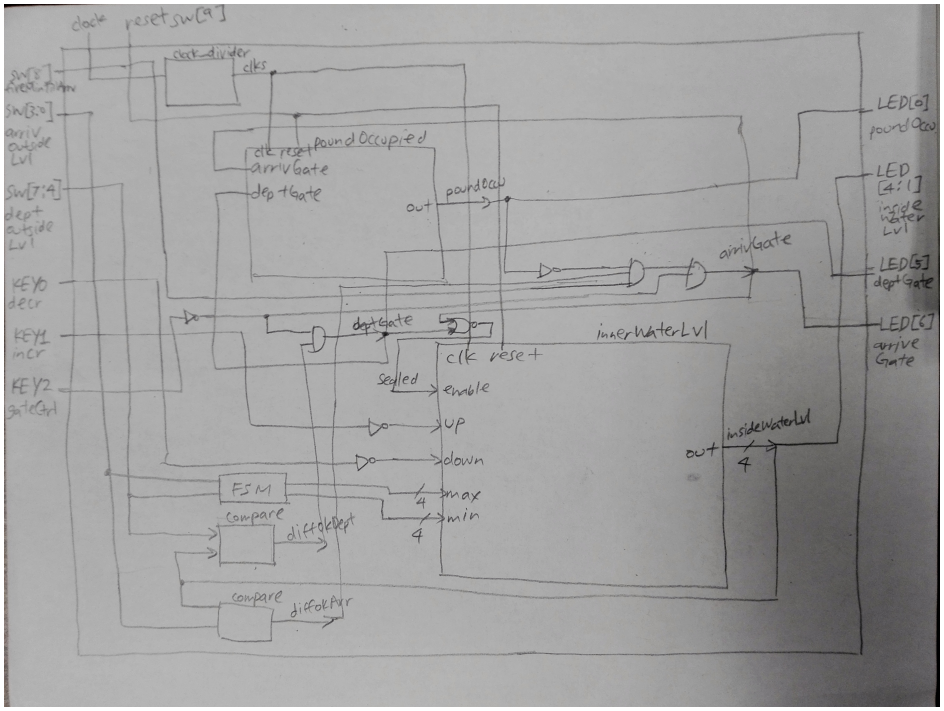

Figure 5: Diagram of overall module, including combinational logic for gate control

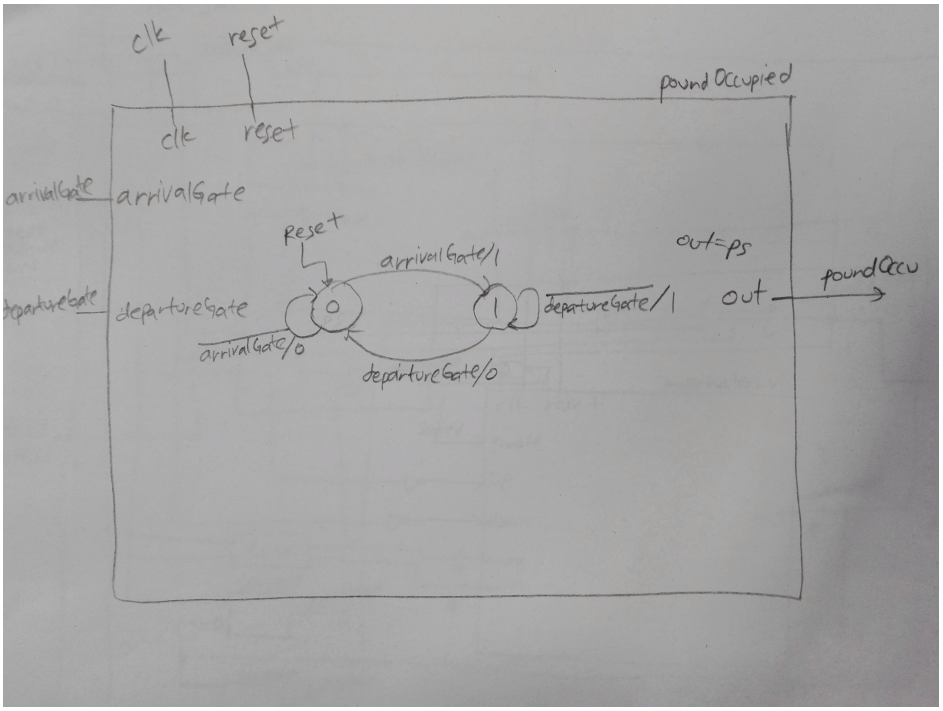

Figure 6: Diagram of poundOccupied FSM module

The innerWaterLvl module controlled the water level inside the lock. It was a generic counter that took min/max values and three control signals (enable, up, down) as input, and gave the counter value as output. The overall module handled the control logic, namely:

- `Enable = ~(arrivGate || deptGate)` (if a gate is open, dont change the water level)
- min/max: determining which outside water level was the higher water level, regardless of direction

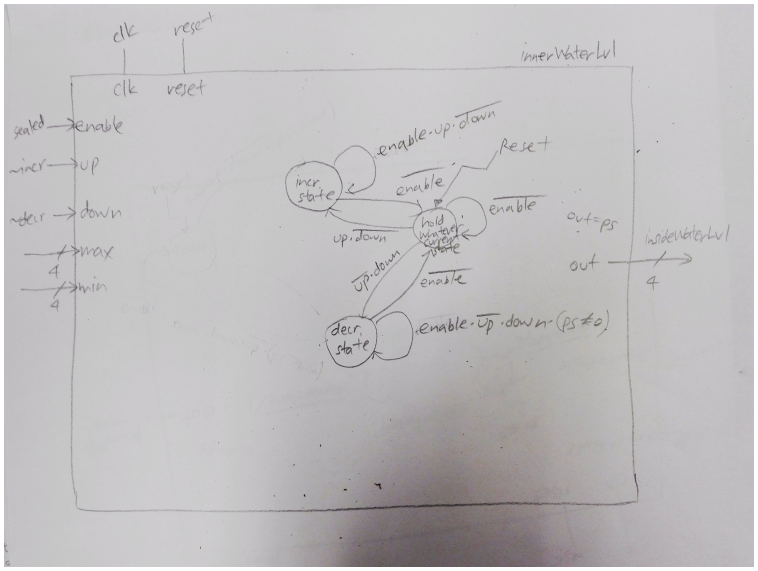

Figure 7: Diagram of innerWaterLvl counter module

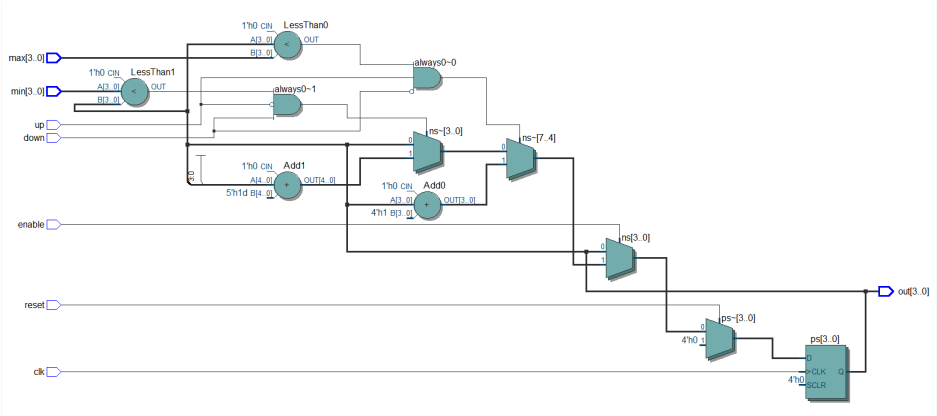

Figure 8: RTL view of innerWaterLvl module

The gate control was implemented using combinational logic. Each gate took the relevant outside water level, the inside water level, the gateCtrl signal, and (in the case of the arrival gate) the fiveMinTillArriv and poundOccupied signal as input, and gave the state of the gate as output (1 = open/0 = closed). The logic equations are as follows:

- `arrivGate = gateCtrl \&\& diffOkArriv \&\& ~poundOccupied \&\& fiveMinTillArriv`
- `deptGate = gateCtrl \&\& diffOkDept`

Finally, the user/environmental inputs and system outputs were connected to the input/outputs of the DE1-SoC board for visual monitoring of the signals.

- `SW[9]` reset
- `SW[8]` fiveMinTillArriv
- `SW[7:4]` deptOutsideWaterLvl, `SW[3:0]` arrivOutsideWaterLvl
- `KEY[0]` decr, `KEY[1]` incr
- `KEY[2]` gateCtrl
- `LED[0]` poundOccupied
- `LED[4:1]` innerWaterLvl
- `LED[5]` deptGate, `LED[6]` arrivGate

3.2 Test

3.2.1 Test Plan

VHDL Design Our lock system was tested under normal operating conditions. This is essentially the process of a boat passing through the lock. When the gondola arrives, the lock receives the five minutes signal. It should then seal the gates and adjust the water level to the arrivals side, then open the arrival gate. The boat enters the lock and it becomes occupied. On departure, the lock should seal the gates and adjust the water level to the departures side, then open the departure gate. The boat exits the lock and the lock becomes unoccupied.

Additionally, we tested the lock system under some unexpected failure conditions. In these cases, the lock system should fail. For example, when the lock is already occupied, another boats arrival signal should not open the gates; if the water levels are not close enough, the gates should not open; if the gates are not sealed, the water levels should not change; if the outside water levels (inputs to the system) are changed during operation, the system should account for it; if the system receives conflicting inputs, it should hold state to avoid faults. Finally, the reset should put the system in the expected reset state.

C Program The C Program was tested to see if the output value matched the value we computed by hand (5).

3.2.2 Test Specification

VHDL Design The primary functions of the lock system were tested under various conditions: the gates, the water level adjustment, and the state of the lock (occupied/unoccupied). This was done both in simulation with iverilog and gtkwave, and on hardware with the DE1-SoC board and Signal Tap II.

The gates were tested to ensure that they open under the correct conditions. For the departure gate, it is that the difference in water levels is less than 0.3ft (1 bit binary difference, i.e. equal values). For the arrival gate, the water levels must be close enough, and it must receive the fiveMinTillArriv signal and the lock must be unoccupied in order for the gates to open. Otherwise, the gates should not open. We isolated the combinational logic for the gates into a separate module for testing purposes, and wrote a separate test bench for it.

The innerWaterLvl counter was tested to ensure that it operated within bounds. It receives input signals enable, up, down, max, and min, and outputs a 4-bit binary value (the inner water level). It should count only when enabled and given an up or down input, and only count to the max/min value (outside water levels). It should hold state when it is disabled, is given no up/down or conflicting up/down input, and it should be able to handle a changing max/min value.

The poundOccupied signal should track the state of the lock - specifically, it should only change state when the arrival gate or departure gate is opened, depending on whether there is currently a boat in the lock.

C Program Since there were no inputs to the program, it was only necessary to check that the output matched the expected value. The program outptus an integer, so we truncated our hand calculated value to an integer as well (5). We also checked to make sure the output was consistent every time we ran it.

3.2.3 Test Cases

VHDL Design The lock system's three components were tested individually.

For gate control, we checked the pass case and a few fail cases.

- pass case
  - inputs: fiveMinTillArriv = 1, poundOccupied = 0, diffOkArriv = 1 or diffOkDept = 1 (matching water levels)
  - measure: whether the gate is open
  - pass condition: the gate should open under these conditions (departure gate should open regardless of fiveMinTillArriv and poundOccupied)
- fail case: water levels do not match
  - inputs: diffOkArriv = 0 or diffOkDept = 0; everything else as in pass case
  - measure: whether the gate is open
  - pass condition: the gate should not open under these conditions
- fail case: no fiveMinTillArriv signal (arrival gate only)
  - input: fiveMinTillArriv = 0; everything else as in pass case
  - measure: whether the gate is open
  - pass condition: the gate should not open under these conditions
- fail case: already occupied (arrival gate only)

  - input: ponudOccupied = 1; everything else as in pass case
  - measure: whether the gate is open
  - pass condition: the gate should not open under these conditions

For the inner water level counter, we checked the effects of changing the control signals on the behaviour

- enable: the counter should not be able to count when enable = 0, regardless of other inputs
- up/down: when enable = 1, the counter should count up when up is pressed and down when down is pressed. When neither/both are presesd, the counter should not count.
- max/min: the counter should count only to the max/min value and then hold. When the max/min is changed while counting, the counter should not fault and instead count to the new value. If the counter's current state becomes out of bounds by a changing max/min, the counter should not be able to count any further out of bounds, but it should be able to count back into the bounds.

For the poundOccupied signal, we tested change/hold state conditions.

- state change cases
  - input: arrivGate = 1 & poundOccupied = 0 or deptGate = 1 & poundOccupied = 1
  - measure: whether poundOccupied changes state
  - pass condition: poundOccupied changes state
- hold state case: gates do not open
  - input: arrivGate = 0 & deptGate = 0; poundOccupied = x
  - measure: whether poundOccupied changes state
  - pass condition: poundOccupied should not change state
- hold state case: gate opens but no boat in lock (departure gate only)
  - input: deptGate = 1 & poundOccupied = 0
  - measure: whether poundOccupied changes state
  - pass condition: poundOccupied should not change state

C Program There were no inputs to the system, so the only case that needed to be tested was regular operation. In this case, the program should output the value 5, matching our hand-computed value (after integer casting truncation).

4 RESULTS

VHDL design The lock system VHDL design worked as expected under normal operating parameters, and under edge cases.

When the lock receives a fiveMinTillArriv signal, the water level is adjusted, the arrival gate is opened and a boat enters, the gate is sealed and the water level adjusted again, and then the departure gate is opened and the boat exits the lock. Environmental inputs like outside water levels are input using the switches, and user inputs like the control signals for the gate and water level adjustment are through the keys. These signals behaved as expected both in simulation with iverilog/gtkwave, and visually on the outputs of the DE1-SoC board.

On reset, the lock reset to the expected reset state, namely the innerWaterLvl is set to 0 (lock drained) and the lock becomes unoccupied. Under unexpected conditions, the lock system handled the error as it should. Namely, if it was a failure condition, the property of the lock failed (gates did not open, water levels did not change). Therefore, the design successfully handled those errors.

C program The C program worked as expected. It outputs the computed value of 5 every time the program is run, and since there are no inputs, there is no chance of abnormal operation.

4.1 Analysis of Errors

VHDL design Ultimately, our project worked and successfully demonstrated the expected behaviour. The issues we ran into were due mostly to preliminary design faults and clock timing. At many points we found that we had failed to consider something in our original design, and this led to (at best) a minor tweak/fix, or (at worst) a major overhaul of our design and a new iteration. These problems ranged from failing to consider a variable in a logic equation to underlying misunderstandings when figuring out the required states and edge conditions in our FSMs. Another common error involved the clock edge timings for FSMs and sequential logic - sometimes we had idiosyncratic errors with a signal needing to be held for a certain number of clock cycles in order to affect the system. We solved these by checking that all our sequential logic ran on the same clock, and within the same sequential logic block. It is important to note that the modularisation of the design was very helpful in isolating the errors and minimising error propagation.

C Program Our C program also worked as expected. We had initially computed the resulting value by hand, which matched what our program's output. We ran into a few issues when trying to output the correct answer; for instance, we would try to use the address in memory in our calculations, since we had forgotten the & for the pointer to go to the specific location in memory, but these were easily resolved.

5 SUMMARY & CONCLUSION

Summary This project designed a canal lock control system that allows gondolas to pass from one side to another by opening/closing gates and adjusting water levels. The lock is operated by a user, who controls it by pressing buttons to open/close the gate and increase/decrease the water level, and receives a trigger signal from the gondola to open the arrival gate. These features and behaviours were tested both in simulation with iverilog and gtkwave as well as on hardware with the DE1-SoC board. As a second component, the project involves writing a C program, using pointers rather than variables.

Conclusion Overall, this project was a good start in building simple VHDL designs. The lock system had a very simple workflow, with enough variables to make it testable for failure conditions. We were able to practice working with iverilog/gtkwave and Quartus/Signal Tap more, as well as learn about new concepts in the C language.

6 APPENDIX

6.1 Lock Control System

6.1.1 Verilog code

```
// EE371 Lab2 Autumn 2016
// authors: William Li, Dawn Liang, Jun Park
// date: 02 November 2016
//
// top-level module for lock control system
// manages I/O's, handles control signals & connects submodules
//
// DE1-SOC BOARD I/O's
//  LED[0] poundOccupied
//  LED[4:1] insideWaterLvl
//  LED[5] deptGate
//  LED[6] arrivGate
//  SW[3:0] arrivOutsideLvl
```

```
//  SW[7:4] deptOutsideLvl
//  SW[8] fiveMinTillArrival
//  SW[9] reset
//  KEY0 decr
//  KEY1 incr
//  KEY2 gateCtrl
module overall(arrivGate, deptGate, insideWaterLvl, poundOccupied, incr, decr,
    arrivOutsideLvl, deptOutsideLvl, fiveMinTillArrival, gateCtrl, clock, reset);
  output [3:0] insideWaterLvl; // LED[4:1]
  output arrivGate, deptGate; // LED[6], LED[5]
  output poundOccupied; // LED[0]

  input [3:0] arrivOutsideLvl, deptOutsideLvl; // SW[3:0], SW[7:4]
  input incr, decr, gateCtrl; // KEY[1], KEY[0], KEY[2]
  input fiveMinTillArrival; // SW[8]
  input reset; // SW[9]

  input clock; // internal 50MHz clock

  // CLOCK DIVIDER
  wire [31:0] clks;
  clock_divider cDiv (clock, clks);
  wire clk;
  assign clk = clks[22];

  // LOCK STATE (OCCUPIED/UNOCCUPIED)
  poundOccupied occup(.out(poundOccupied), .arrivalGate(arrivGate),
      .departureGate(deptGate), .clk(clk), .reset(reset));

  // CONTROLLING GATES
  wire diffOkArr, diffOkDept;
  assign diffOkArr = (insideWaterLvl == arrivOutsideLvl);
  assign diffOkDept = (insideWaterLvl == deptOutsideLvl);
  assign arrivGate = ~gateCtrl && (fiveMinTillArrival && ~poundOccupied && diffOkArr);
  assign deptGate = ~gateCtrl && diffOkDept;

  wire sealed;
  assign sealed = ~(arrivGate || deptGate);

  // CONTROLLING LOCK WATER LEVELS

  // determine bounds for innerWaterLvl counter
  reg [3:0] max, min;
  always@(*) begin
    if (arrivOutsideLvl > deptOutsideLvl) begin
      max = arrivOutsideLvl;
      min = deptOutsideLvl;
    end else begin
      max = deptOutsideLvl;
      min = arrivOutsideLvl;
    end
  end

  innerWaterLvl inner (.out(insideWaterLvl), .enable(sealed), .up(~incr),
      .down(~decr), .max(max), .min(min), .clk(clk), .reset(reset));
endmodule
```

```
// EE371 Lab2 Autumn 2016
```

```
// authors: William Li, Dawn Liang, Jun Park
// date: 02 November 2016
//
// testbench for overall module: tests normal operating conditions
// failure test cases: no fiveMinTillArrival signal, incr & decr pressed
`include "overall.v"
module overall_testbench();
  reg clk, reset;
  reg incr, decr, fiveMinTillArrival, gateCtrl;
  reg [3:0] arrivOutsideLvl, deptOutsideLvl;

  wire arrivGate, deptGate, poundOccupied;
  wire [3:0] insideWaterLvl;

  // connect I/Os
  overall dut (.arrivGate(arrivGate), .deptGate(deptGate),
    .insideWaterLvl(insideWaterLvl), .poundOccupied(poundOccupied), .incr(incr),
    .decr(decr), .arrivOutsideLvl(arrivOutsideLvl), .deptOutsideLvl(deptOutsideLvl),
    .fiveMinTillArrival(fiveMinTillArrival), .gateCtrl(gateCtrl), .clock(clk),
        .reset(reset));

  // set up clock
  parameter CLOCK_PERIOD = 5;
  initial clk = 1;
  always begin
    #(CLOCK_PERIOD/2);
    clk = ~clk;
  end

  // start simulation
  initial begin
    reset <= 1;                                    @(posedge clk);
                                                  @(posedge clk);
    reset <= 0; arrivOutsideLvl <= 4'b0100; deptOutsideLvl <= 4'b0000;
          incr <= 0; decr <= 0; fiveMinTillArrival <= 0; gateCtrl <= 0; @(posedge
              clk);
                     gateCtrl <= 1;           @(posedge clk);
                                              @(posedge clk);
                     gateCtrl <= 0;           @(posedge clk);
                     fiveMinTillArrival <= 1;    @(posedge clk);
                                              @(posedge clk);
          incr <= 1;                          @(posedge clk);
                                              @(posedge clk);
                                              @(posedge clk);
                                              @(posedge clk);
                                              @(posedge clk);
                                              @(posedge clk);
                                              @(posedge clk);
                                              @(posedge clk);
                     fiveMinTillArrival <= 1;    @(posedge clk);
                     gateCtrl <= 1;           @(posedge clk);
                                              @(posedge clk);
                     gateCtrl <= 0;           @(posedge clk);
                                              @(posedge clk);
          decr <= 1;                          @(posedge clk);
          incr <= 0;                          @(posedge clk);
                                              @(posedge clk);
                                              @(posedge clk);
```

```
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
                            gateCtrl <= 1;        @(posedge clk);
                            gateCtrl <= 0;        @(posedge clk);
                                                @(posedge clk);
                                                @(posedge clk);
    $finish;
  end

  // gtkwave filedump
  initial begin
    $dumpfile("overall.vcd");
    $dumpvars;
  end
endmodule
```

```
// EE371 Lab2 Autumn 2016
// authors: William Li, Dawn Liang, Jun Park
// date: 02 November 2016
//
// innerWaterLvl module: enable-counter that counts up/down based on inputs,
//  to max/min values
module innerWaterLvl(out, enable, up, down, max, min, clk, reset);
  input enable, up, down, clk, reset;
  input [3:0] max, min;
  output [3:0] out;

  // combinational logic
  // if enabled & not at max/min, count up/down based on input
  // otherwise don't count
  reg [3:0] ps, ns;
  always@(*) begin
    if (enable) begin
      if (up && ~down && (ps < max)) begin
        ns = ps + 1'b1;
      end else if (down && ~up && (ps > min)) begin
        ns = ps - 1'b1;
      end else begin
        ns = ps;
      end
    end else begin
      ns = ps;
    end
  end

  // output logic
  assign out = ps;

  // sequential logic
  always@(posedge clk) begin
```

```
    if (reset) begin
      ps <= 4'b0;
    end else begin
      ps <= ns;
    end
  end
endmodule
```

```
// EE371 Lab2 Autumn 2016
// authors: William Li, Dawn Liang, Jun Park
// date: 02 November 2016
//
// testbench for innerWaterLvl module: tests normal operating conditions,
//  failure test cases include pressing up & down, disabled counter
`include "innerWaterLvl.v"
module innerWaterLvl_testbench();
  reg clk, reset, up, down, enable;
  reg [3:0] max, min;
  wire [3:0] out;

  // set up clock
  parameter CLOCK_PERIOD = 5;
  initial clk = 1;
  always begin
    #(CLOCK_PERIOD/2);
    clk = ~clk;
  end

  // connect I/Os
  innerWaterLvl dut (.out(out), .enable(enable), .up(up), .down(down), .max(max),
      .min(min), .clk(clk), .reset(reset));

  // run simulation
  initial begin
    reset <= 1;                                          @(posedge clk);
    reset <= 0; enable <= 0; up <= 0; down <= 0; max <= 4'b0100; min <= 4'b0001;
        @(posedge clk);
                  up <= 1;                          @(posedge clk);
           enable <= 1;                               @(posedge clk);
                                                     @(posedge clk);
                                                     @(posedge clk);
                                                     @(posedge clk);
                                                     @(posedge clk);
                                                     @(posedge clk);
                  up <= 0;    down <= 1;               @(posedge clk);
                                                     @(posedge clk);
                  up <= 1;                           @(posedge clk);
                  up <= 0;                           @(posedge clk);
                                                     @(posedge clk);
                                                     @(posedge clk);
                                                     @(posedge clk);
    $finish;
  end

  // gtkwave filedump
  initial begin
    $dumpfile("innerWaterLvl.vcd");
    $dumpvars;
```

```
  end
endmodule
```

```
// EE371 Lab2 Autumn 2016
// authors: William Li, Dawn Liang, Jun Park
// date: 02 November 2016
//
// FSM to keep track of the state of the lock; whether it is currently occupied
module poundOccupied(out, arrivalGate, departureGate, clk, reset);
  output out;
  input arrivalGate, departureGate;
  input clk, reset;

  // comb logic
  // 1 = occupied, 0 = unoccupied
  reg ps, ns;
  always@(*) begin
    case (ps)
      0: if (arrivalGate)
           ns = 1;
         else
           ns = 0;

      1: if (departureGate)
           ns = 0;
         else
           ns = 1;
    endcase
  end

  // output logic
  assign out = ps;

  // sequential logic
  always @(posedge clk) begin
    if (reset) begin
      ps <= 0;
    end else begin
      ps <= ns;
    end
  end
endmodule
```

```
// EE371 Lab2 Autumn 2016
// authors: William Li, Dawn Liang, Jun Park
// date: 02 November 2016
//
// testbench for poundOccupied module
// tests all possible input conditions for FSM
`include "poundOccupied.v"
module poundOccupied_testbench;
  wire occupied;
  reg arrivalGate, departureGate, clk, reset;

  poundOccupied dut (.out(occupied), .arrivalGate(arrivalGate),
      .departureGate(departureGate), .clk(clk), .reset(reset));
```

```
  parameter CLOCK_PERIOD = 5;
  initial clk = 1;

  always begin
    #(CLOCK_PERIOD/2) clk = ~clk;
  end


  initial begin
    $display("reset, clk, arrivalGate, departureGate, canalState");
    $monitor("%b\t %b\t %b\t %b\t %b\t", reset, clk, arrivalGate, departureGate,
        occupied);


      reset <= 1;  arrivalGate <= 0; departureGate <= 0; @(posedge clk);
      reset <= 0;                          @(posedge clk);
                          departureGate <= 1; @(posedge clk);
                arrivalGate <= 1; departureGate <= 0; @(posedge clk);
                arrivalGate <= 0;           @(posedge clk);
                          departureGate <= 1; @(posedge clk);
                                      @(posedge clk);
    $finish;
  end

  initial begin
    $dumpfile("poundOccupied.vcd");
    $dumpvars;
  end
endmodule
```

```
// EE371 Lab2 Autumn 2016
// authors: William Li, Dawn Liang, Jun Park
// date: 02 November 2016
//
// clock divider module: outputs a 32-bit counter bus
//   choose the clock speed by choosing the bit
module clock_divider (clk, divided_clocks);
  input clk;
  output [31:0] divided_clocks;

  reg [31:0] clocks;
  initial clocks = 0;

  always@(posedge clk) begin
    clocks <= clocks + 1'b1;
  end

  assign divided_clocks = clocks;
endmodule
```

```
// EE371 Lab2 Autumn 2016
// authors: William Li, Dawn Liang, Jun Park
// date: 02 November 2016
//
// testbench for clock_divider module
module clock_divider_testbench();
  reg clk;
```

```
  wire divided_clock;

  // set up clock
  parameter CLOCK_PERIOD = 5;
  initial clk = 1;
  always begin
    #(CLOCK_PERIOD/2);
    clk = ~clk;
  end

  // connect I/O's
  clock_divider #(2) dut (.clk(clk), .divided_clock(divided_clock));

  // start simulation
  initial begin
    @(posedge clk);
    @(posedge clk);
    @(posedge clk);
    @(posedge clk);
    @(posedge clk);
    @(posedge clk);
    @(posedge clk);
    $finish;
  end

  // gtkwave filedump
  initial begin
    $dumpfile("clock_divider.vcd");
    $dumpvars;
  end
endmodule
```

```
// EE371 Lab2 Autumn 2016
// authors: William Li, Dawn Liang, Jun Park
// date: 02 November 2016
//
// temporary separate gate module for testing combinational logic
module gate(doorOpen, fiveMinTillArriv, diffOk, whatGate, poundOccu, ctrl, reset);
  output doorOpen;
  input fiveMinTillArriv, diffOk, whatGate, poundOccu, ctrl, reset;

  assign doorOpen = ctrl && ((~whatGate && diffOk) ||    // departure gate logic
    (whatGate && diffOk && ~poundOccu && fiveMinTillArriv)); // arrival gate logic
endmodule
```

```
// EE371 Lab2 Autumn 2016
// authors: William Li, Dawn Liang, Jun Park
// date: 02 November 2016
//
// testbench for temporary gate module: checks every possible input condition
`include "gate.v"
module gate_testbench;
  wire doorOpen;
  reg fiveMinTillArriv, diffOk, whatGate, poundOccu, ctrl, reset;

  gate dut(.doorOpen(doorOpen), .fiveMinTillArriv(fiveMinTillArriv), .diffOk(diffOk),
      .whatGate(whatGate), .poundOccu(poundOccu), .ctrl(ctrl), .reset(reset));
```

```
  parameter delay = 10;
  initial begin
    reset = 0; #delay;
    reset = 1; #delay;

    $display("whatGate ctrl poundOccu 5minTA diffOk ");
    $monitor("%b\t %b\t %b\t %b\t %b\t", whatGate, ctrl, poundOccu, fiveMinTillArriv,
        diffOk, doorOpen);

    whatGate = 0; ctrl = 0; poundOccu = 0; fiveMinTillArriv = 0; diffOk = 0; #delay;
    whatGate = 0; ctrl = 0; poundOccu = 0; fiveMinTillArriv = 0; diffOk = 1; #delay;
    whatGate = 0; ctrl = 0; poundOccu = 0; fiveMinTillArriv = 1; diffOk = 0; #delay;
    whatGate = 0; ctrl = 0; poundOccu = 0; fiveMinTillArriv = 1; diffOk = 1; #delay;
    whatGate = 0; ctrl = 0; poundOccu = 1; fiveMinTillArriv = 0; diffOk = 0; #delay;
    whatGate = 0; ctrl = 0; poundOccu = 1; fiveMinTillArriv = 0; diffOk = 1; #delay;
    whatGate = 0; ctrl = 0; poundOccu = 1; fiveMinTillArriv = 1; diffOk = 0; #delay;
    whatGate = 0; ctrl = 0; poundOccu = 1; fiveMinTillArriv = 1; diffOk = 1; #delay;
    whatGate = 0; ctrl = 1; poundOccu = 0; fiveMinTillArriv = 0; diffOk = 0; #delay;
    whatGate = 0; ctrl = 1; poundOccu = 0; fiveMinTillArriv = 0; diffOk = 1; #delay;
    whatGate = 0; ctrl = 1; poundOccu = 0; fiveMinTillArriv = 1; diffOk = 0; #delay;
    whatGate = 0; ctrl = 1; poundOccu = 0; fiveMinTillArriv = 1; diffOk = 1; #delay;
    whatGate = 0; ctrl = 1; poundOccu = 1; fiveMinTillArriv = 0; diffOk = 0; #delay;
    whatGate = 0; ctrl = 1; poundOccu = 1; fiveMinTillArriv = 0; diffOk = 1; #delay;
    whatGate = 0; ctrl = 1; poundOccu = 1; fiveMinTillArriv = 1; diffOk = 0; #delay;
    whatGate = 0; ctrl = 1; poundOccu = 1; fiveMinTillArriv = 1; diffOk = 1; #delay;
    whatGate = 1; ctrl = 0; poundOccu = 0; fiveMinTillArriv = 0; diffOk = 0; #delay;
    whatGate = 1; ctrl = 0; poundOccu = 0; fiveMinTillArriv = 0; diffOk = 1; #delay;
    whatGate = 1; ctrl = 0; poundOccu = 0; fiveMinTillArriv = 1; diffOk = 0; #delay;
    whatGate = 1; ctrl = 0; poundOccu = 0; fiveMinTillArriv = 1; diffOk = 1; #delay;
    whatGate = 1; ctrl = 0; poundOccu = 1; fiveMinTillArriv = 0; diffOk = 0; #delay;
    whatGate = 1; ctrl = 0; poundOccu = 1; fiveMinTillArriv = 0; diffOk = 1; #delay;
    whatGate = 1; ctrl = 0; poundOccu = 1; fiveMinTillArriv = 1; diffOk = 0; #delay;
    whatGate = 1; ctrl = 0; poundOccu = 1; fiveMinTillArriv = 1; diffOk = 1; #delay;
    whatGate = 1; ctrl = 1; poundOccu = 0; fiveMinTillArriv = 0; diffOk = 0; #delay;
    whatGate = 1; ctrl = 1; poundOccu = 0; fiveMinTillArriv = 0; diffOk = 1; #delay;
    whatGate = 1; ctrl = 1; poundOccu = 0; fiveMinTillArriv = 1; diffOk = 0; #delay;
    whatGate = 1; ctrl = 1; poundOccu = 0; fiveMinTillArriv = 1; diffOk = 1; #delay;
    whatGate = 1; ctrl = 1; poundOccu = 1; fiveMinTillArriv = 0; diffOk = 0; #delay;
    whatGate = 1; ctrl = 1; poundOccu = 1; fiveMinTillArriv = 0; diffOk = 1; #delay;
    whatGate = 1; ctrl = 1; poundOccu = 1; fiveMinTillArriv = 1; diffOk = 0; #delay;
    whatGate = 1; ctrl = 1; poundOccu = 1; fiveMinTillArriv = 1; diffOk = 1; #delay;

    $finish; // finish simulation
  end

  // file for gtkwave
  initial begin
    $dumpfile("gate.vcd");
    $dumpvars;
  end
endmodule
```

6.1.2 iverilog & gtkwave waveforms

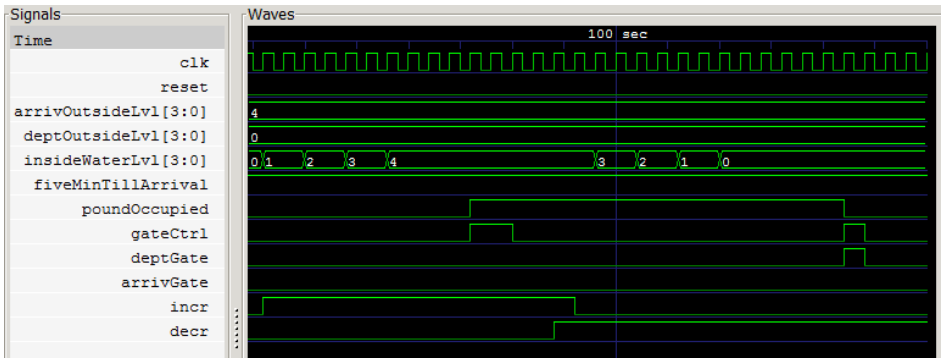

Figure 9: waveform for overall module

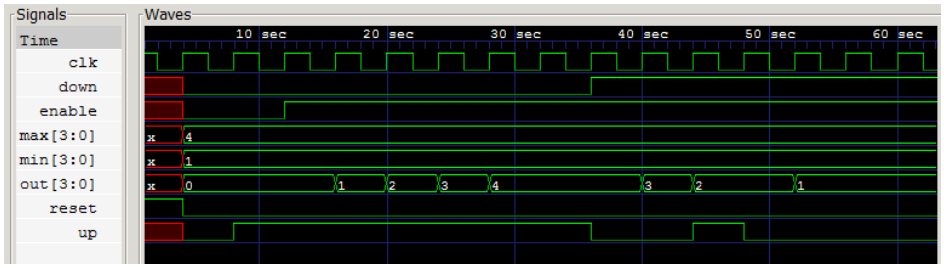

Figure 10: waveform for innerWaterLvl module

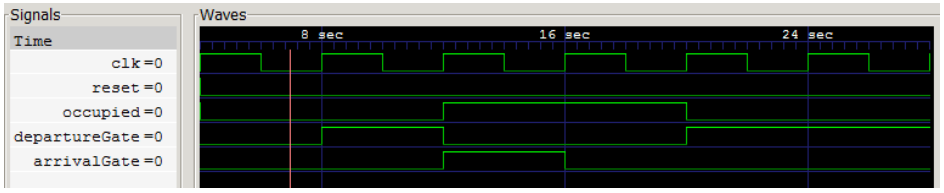

Figure 11: waveform for poundOccupied module

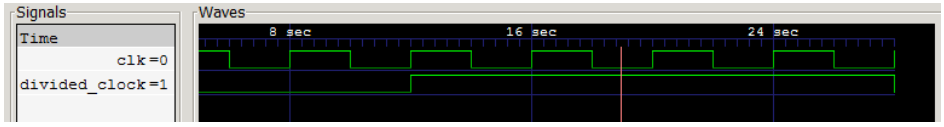

Figure 12: waveform for clock divider

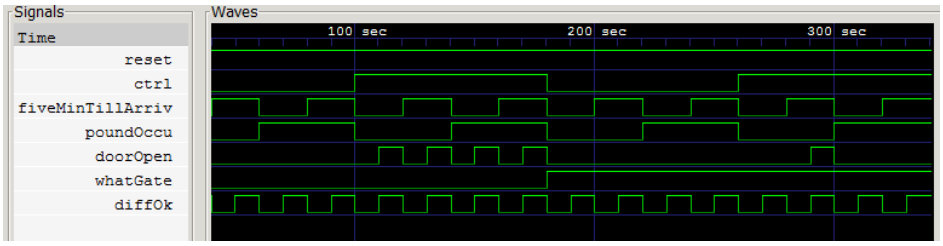

Figure 13: waveform for gate module

```
VCD info: dumpfile gate.vcd opened for output.
whatGate ctrl poundOccu 5minTA diffOk
0        0        0       0      0      0
0        0        0       0      1      0
0        0        0       1      0      0
0        0        0       1      1      0
0        0        1       0      0      0
0        0        1       0      1      0
0        0        1       1      0      0
0        0        1       1      1      0
0        1        0       0      0      0
0        1        0       0      1      1
0        1        0       1      0      0
0        1        0       1      1      1
0        1        1       0      0      0
0        1        1       0      1      1
0        1        1       1      0      0
0        1        1       1      1      1
1        0        0       0      0      0
1        0        0       0      1      0
1        0        0       1      0      0
1        0        0       1      1      0
1        0        1       0      0      0
1        0        1       0      1      0
1        0        1       1      0      0
1        0        1       1      1      0
1        1        0       0      0      0
1        1        0       0      1      0
1        1        0       1      0      0
1        1        0       1      1      1
1        1        1       0      0      0
1        1        1       0      1      0
1        1        1       1      0      0
1        1        1       1      1      0
```

Figure 14: truth table for gate module

6.1.3 Signal Tap II data

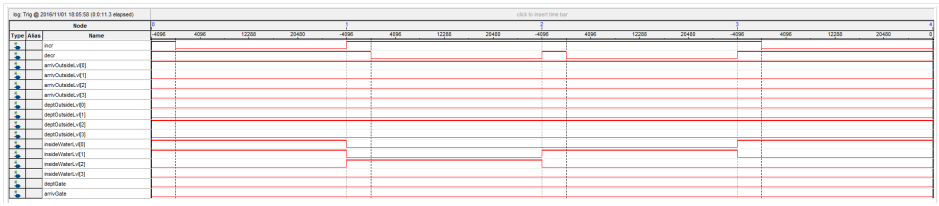

Figure 15: Signal Tap data checking behaviour of the inner water level after pressing control signal

6.1.4 Requirements Documentation

Abstract At various points along the canal system of Venice, gondolas must move between areas of significantly differing water levels. In order to do so smoothly, pound locks are in place to facilitate both transportation of gondolas and maintenance of water levels. These locks are controlled by a digital system we have designed and built.

Introduction The system for controlling locks will allow gondolas to pass between areas of the canal of uneven water levels. Upon arrival, the gondola must signal the lock to initiate the process, enter the lock, wait as it is raised/lowered, and then exit the lock to continue its journey. The opening/closing of gates and raising/lowering of water levels are controlled by a lock operator.

Inputs to the system

User inputs

- fiveMinsTillArrival: signal from the gondola that it will be arriving in at least five minutes, to prompt the lock to adjust water levels and open the gates
- incrWaterLevel: control signal to increase the water level inside the lock, to bring the water levels inside and outside the lock close enough that the gates can open
- decrWaterLevel: control signal to decrease the water level inside the lock, to bring the water levels inside and outside the lock close enough that the gates can open
- reset: empties the lock to the lower outside water level, lets the boats out, and seals the gates

Environmental controls

- arrivOutsideLevel: the water level outside the lock, on the arrival side, determining whether the arrival gate can open
- deptOutsideLevel: the water level outside the lock, on the departure side, determining whether the departure gate can open

Internal signals

- poundOccupied: whether there is currently a gondola in the lock, since if the lock is occupied, the next boat cannot enter
- insideWaterLevel: the water level inside the lock, determining whether the gates can open
- arrivGateOpen: whether the door on the arrival side is open
- deptGateOpen: whether the door on the departure side is open

Outputs of major functions

Gate control

- gateOpenClose: opens/closes the lock gates, allowing gondolas to enter/exit the lock

Adjusting lock water level

- incrWaterLevel: increases the water level inside the lock, matching it to the outside higher water level so that the gate can open
- decrWaterLevel: decreases the water level inside the lock, matching it to the outside lower water level so that the gate can open

6.2 C Program

```
/* EE 371 Lab 2
   Dawn Liang, Jun Park, William Li
   C Pointer Program

   This C program teaches us the fundamentals of
   pointers and references. The output of this
   program is the based on the equation:
   ((A - B) * (C + D)) / E, where the following
   variables are integer values that have been
   referenced from memory using pointers.
*/

#include <stdio.h>

int main() {
    int A = 22;
    int B = 17;
    int C = 6;
    int D = 4;
    int E = 9;
    int result;

    int* x1;
    int* x2;
    int* x3;
    int* x4;
    int* x5;

    x1 = &A;
    x2 = &B;
    x3 = &C;
    x4 = &D;
    x5 = &E;

    result = ((*x1 - *x2) * (*x3 + *x4)) / *x5;
    printf("%d\n", result);

    return 0;
}
```

```
Last login: Wed Nov  2 16:42:08 on ttys000
D-69-91-164-136:~ dawnliang$ /Users/dawnliang/Documents/school/ee371/lab2/c/proj
ect2
5
D-69-91-164-136:~ dawnliang$
```

Figure 16: Output of C program