**Lab 4: Advanced Elevator Controller**

**Authors:** C2C Lauren Humpherys and C2C Christopher Katz

**Documentation:** Capt Johnson and Col Neff helped us by answering many questions. Josh Krutz pointed out how to adjust our code so that we did not have to hold the GO button down. Josiah Hoege explained what each reset button was supposed to do and how to incorporate them. Grady Phillips Showed us that we should use hex instead of binary to make things easier, and made fun of us for using “not not” in our code instead of removing both negations.

**Purpose:** The purpose of this lab is to practice designing more complex systems that integrate sequential and combinational building blocks. A FPGA will be configured with the Moore Elevator Controller state machine. Features will then be added to the design incrementally to build up to a fully-featured elevator controller system.

**Prelab:** The first step in our design was to create the top-level design of our system, mapping all the components and connecting their inputs and outputs to one-another. This design can be found in the schematic below:

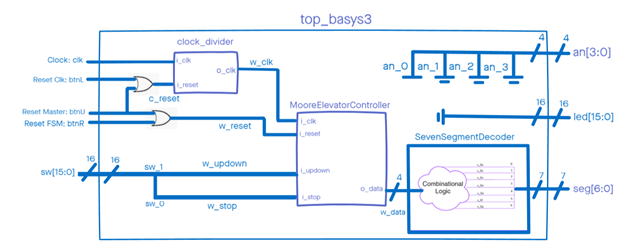


Figure 0 – Top-Level Design Schematic – Minimum Functionality

Using code from our provided top\_basys3.vhd file, we accounted for all the necessary components in our design schematic. The button btnU is the master reset button, and is thus connected to the reset inputs of both the clock divider and the Moore Elevator Controller. Meanwhile, btnL only resets the clock divider, and btnR can be pushed to reset the Moore Elevator Controller. Clk is the clock input for the clock divider, which is wired to the i\_clk input for the Moore Elevator Controller. Switch sw(1) will be used to control the direction of the elevator: setting the switch to 1 will cause the elevator to travel upward, while switching it off will send it downward. The switch sw(0) can be used to stop the elevator at any time. These inputs will be processed through the Moore Elevator Controller, and then output to the seven-segment decoder. This component will then display the appropriate output based on the input processed by the Moore Elevator Controller. For this development stage, all LED lights and anodes have been grounded.

**Design:**  In order to implement this lab our final design needed to not only match the design seen in Figure 0, but also incorporate the additional functionalities of this lab. Our first step in reaching this goal was to design and implement the minimum functionality shown in our prelab. From there, we added the functionalities for more floors, blinking lights, and finally the ability to choose a floor.

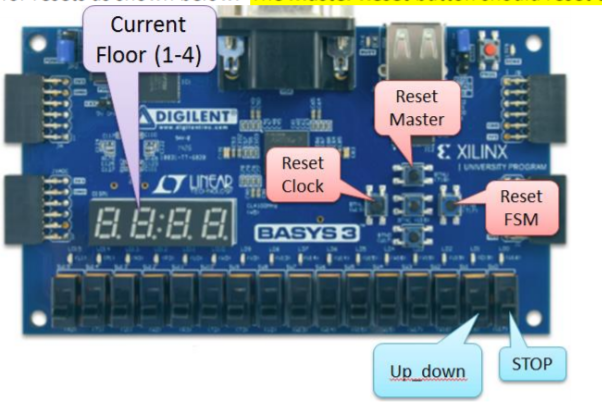


Figure 1 - Hardware Design for Minimum Functionality

**Our Approach** – Our approach was to first program the minimum functionality of the elevator, implementing the Moore Elevator Controller, Seven Segment Decoder, and a clock divider. As indicated in Figure 1, sw(1) served as a directional indicator: whenever the switch was set to ‘1’, the signal would go through the Moore Elevator Controller, which would output the next state and tell the Seven Segment Decoder to output a count one floor higher than where it is currently. The floor count would increase until it reached the fourth floor, or until sw(0) was switched to ‘1’ to initiate a stop signal. When sw(1) and sw(0) were both switched to ‘0’, the floor count would decrease until it reached the first floor, and then stay there until sw(1) is switched to ‘1’ again. The rate at which the floor would change was controlled by a clock divider operating at 2 Hz, which was connected as an input to the Moore Elevator Controller.

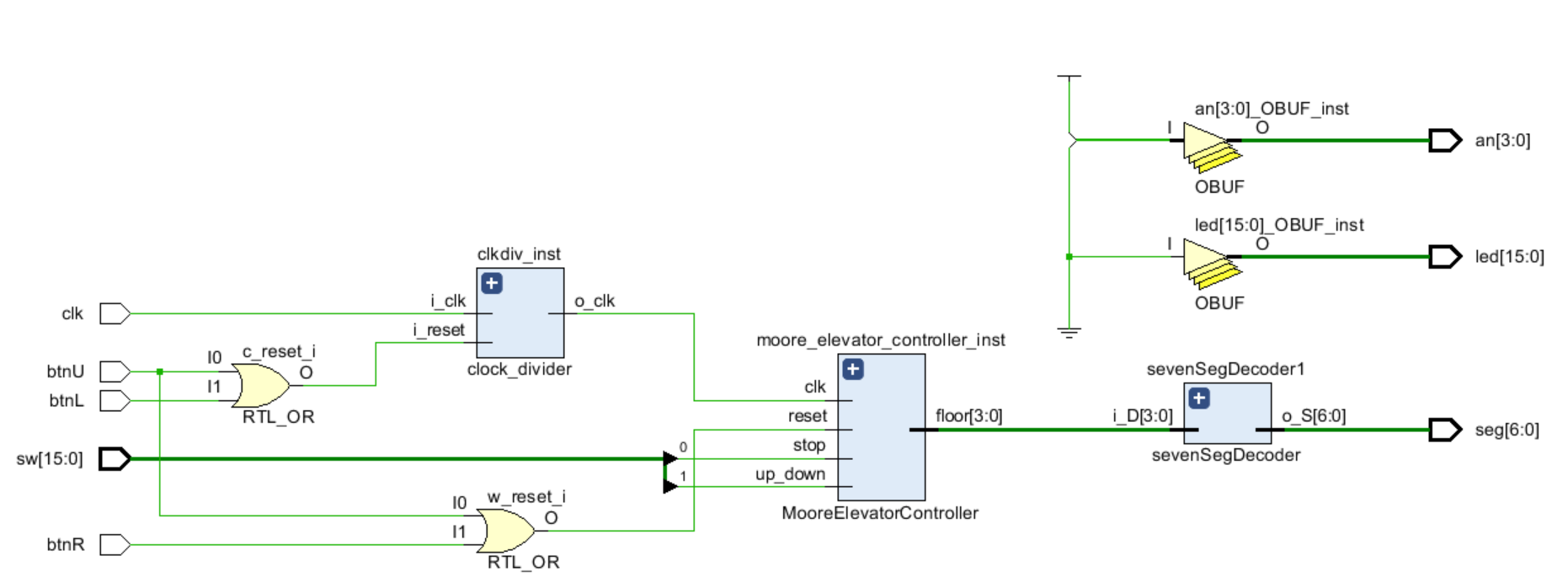


Figure 2 - High Level RTL Schematic with Moore Elevator Controller

The first of these components to be added to the design was the Moore Elevator Controller. We had already implemented this finite state machine and tested its functionality in ICE5, so all we had to do was double-check that we were getting the correct output by running another simulation. As evident by the four-bit floor vector outputting from the Moore Elevator Controller and into the Seven Segment Decoder in Figure 2, the Moore Elevator Controller was able to go from floor 1 and count all the way to floor 4, and start back at floor 4 upon FSM reset. From here, we created an instantiation of this component in our top\_basys3.vhd file.

This part of the lab taught us that we should spend more time preparing and trying to understand the desired end goal, which would mean less time executing and implementing everything. We tried jumping into things and wiring components together without first gaining a thorough understanding of what items are supposed to accomplish what outcome and why. This resulted in the minimum functionality part of the lab entailing a lot more time and confusion than it would have taken otherwise. We learned from this and were able to put this lesson into practice for the remaining stages of the lab.

With the minimum functionality complete, we moved on to the advanced functionality. To map out our target we created the schematic below:

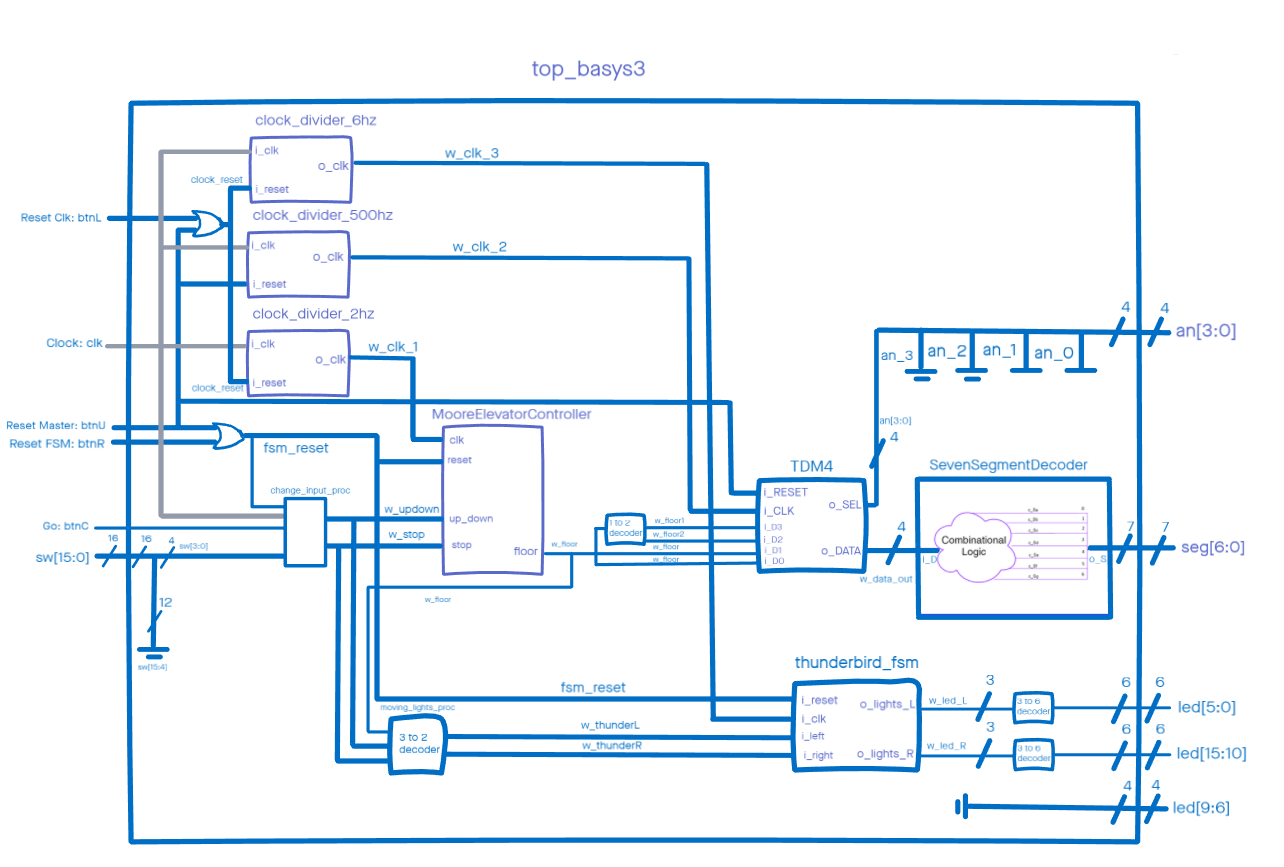


Figure 3 – Top Level Schematic – full functionality

The next phase of the lab was the addition of more floors that our elevator could travel to 15 instead of 4. Here we had to implement the use of our TDM4 finite state machine in order to send the correct number to each anode. We also had code for the TDM4 already prepared from ICE6, but we still had to build on it and add 11 more floor states to account for all 15. We also had to create an instantiation of another clock divider specifically for the TDM4, operating at 500 Hz instead of 2. Finally, after adding extra floors 5 through F, we had to implement a decoder to take the four bits coming from the elevator controller and convert it to two outputs, four bits each, going to the TDM4. This logic assigned the first output to equal 0 if the DATA was anything less than 10 and a 1 otherwise, and it set the second digit to equal DATA if below 10 and equal the 1s digit for anything higher.

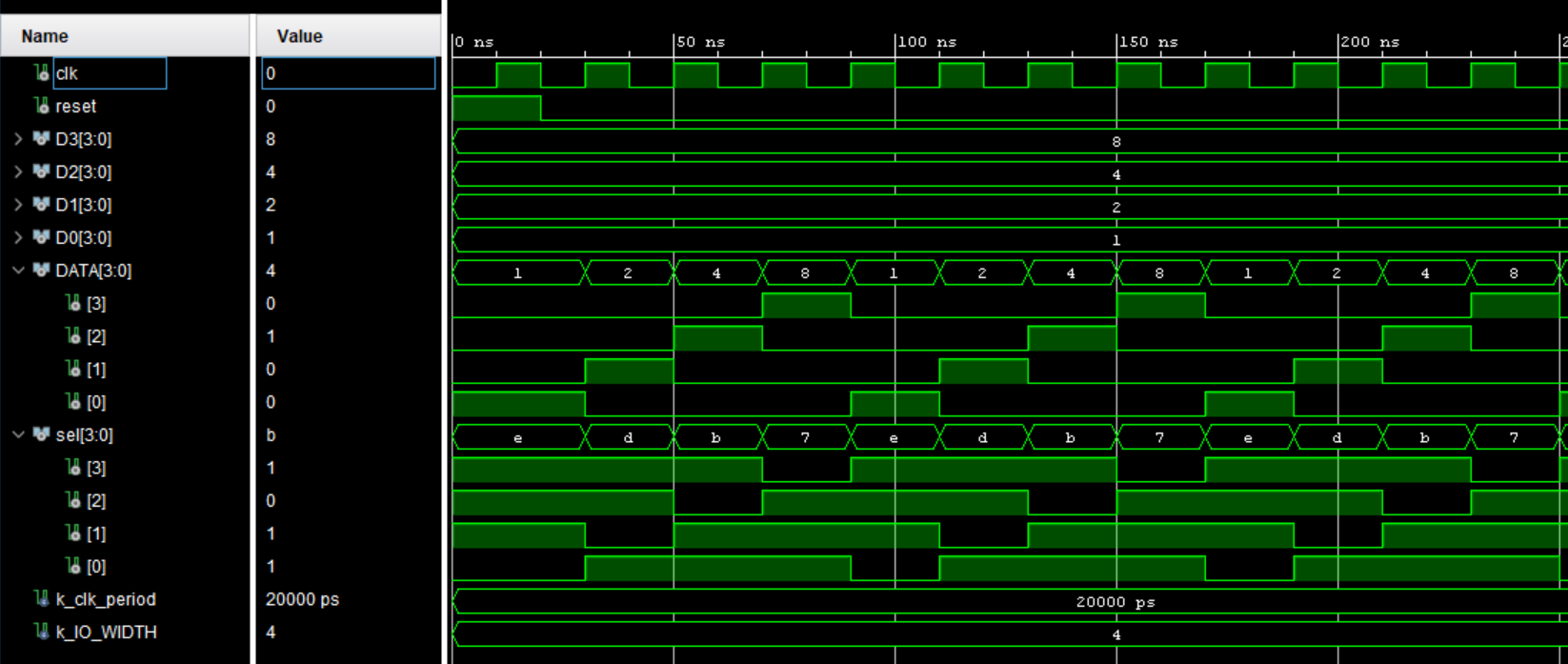


Figure 4 – TDM4 Simulation Waveform

It is important to note the two output vectors, DATA[3:0] and sel[3:0]. The sel vector, labeled as such for “select,” provided the selected data line number in the form of a 4-bit, one-cold encoding to anode 2. The DATA provided the updated floor count, which was input into the Seven Segment Decoder as the next state. The three reset buttons, the clock reset, master reset, and fsm reset, were all used in various configurations. The reset of the TDM4 component was connected directly to the master reset button. The master reset and fsm reset buttons were connected with an or gate to the more elevator controller, and the clock reset button was connected to the master reset button through an or gate and then to the 2hz clock divider. With the more floors functionality complete, we created another diagram.

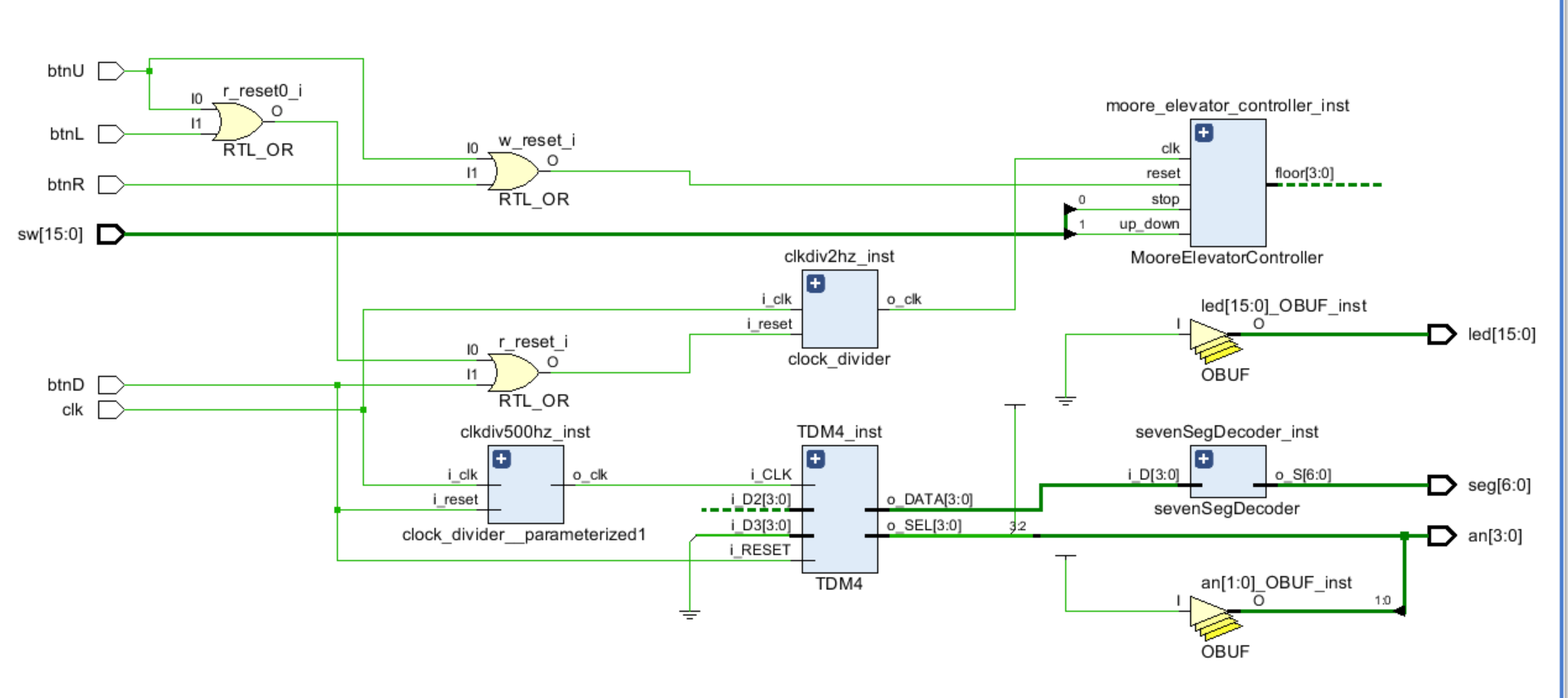


Figure 5 – Top level design with more floors functionality

The Moving Lights phase of the lab posed as a little bit more of a challenge. It was here where we began implementing the Thunderbird finite state machine. We already had a head start from our implementation in Lab 3, but now we needed to figure out how to light up 2 LED lights, then 4 and 6 in the direction the elevator travels (instead of 1, 2 and then 3 lights.) Additionally, we needed to synchronize the output with whether or not the elevator is moving - when the elevator stops moving, the LEDs shall shop flashing regardless of the state of the switches. Finally, the reset for the thunderbird fsm needs to be completely synchronized with the moore elevator controller.

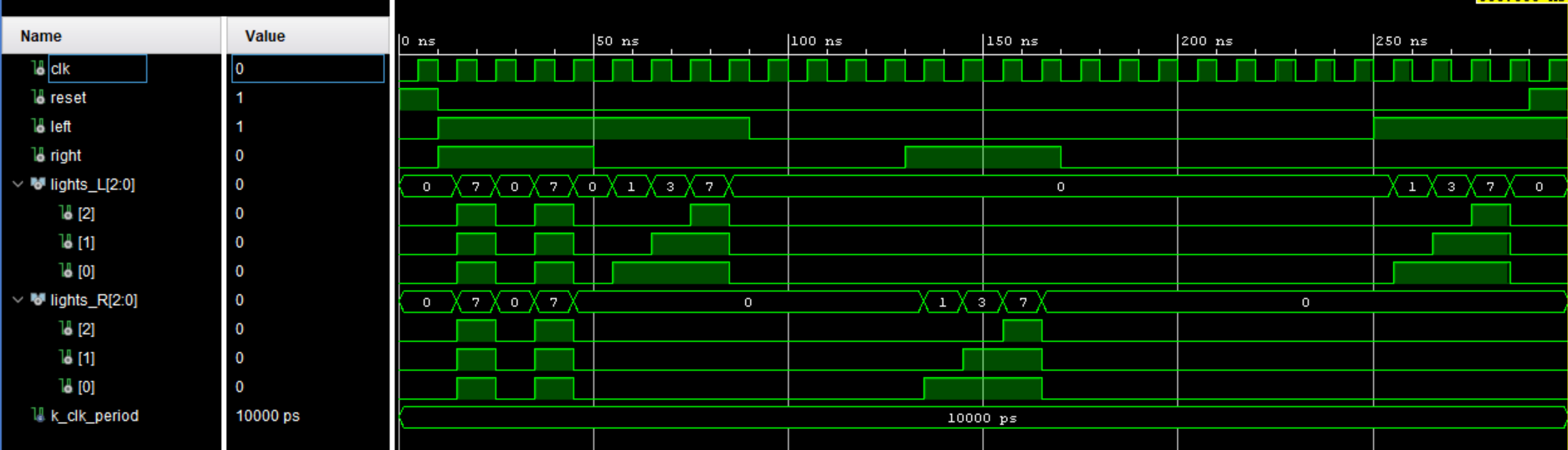


Figure 6 – Thunderbird FSM Simulation Waveform

The above simulation shows the output of the original Thunderbird FSM. With regards to lights\_L and lights\_R, our goal was to make those “pyramids” twice as large. For lights\_R, this would mean that led(0) and led(1) would light up, then additionally have led(2) and led(3), and finally these four plus led(4) and led(5). This entailed hooking up two LEDs to one signal in order to connect multiple LEDs to a single output from the FSM.

To synchronize the timing of the two components, moore elevator controller and thunderbird fsm, we created a new clock divider running at 6hz instead of two. This means for every clock cycle of the floor controller, the lights would cycle three times.

In order to achieve the correct behavior of the reset buttons, we hooked up the thunderbird reset to the same line feeding the more elevator controller and the 6hz clock divider to the same line feeding the 2hz clock divider.

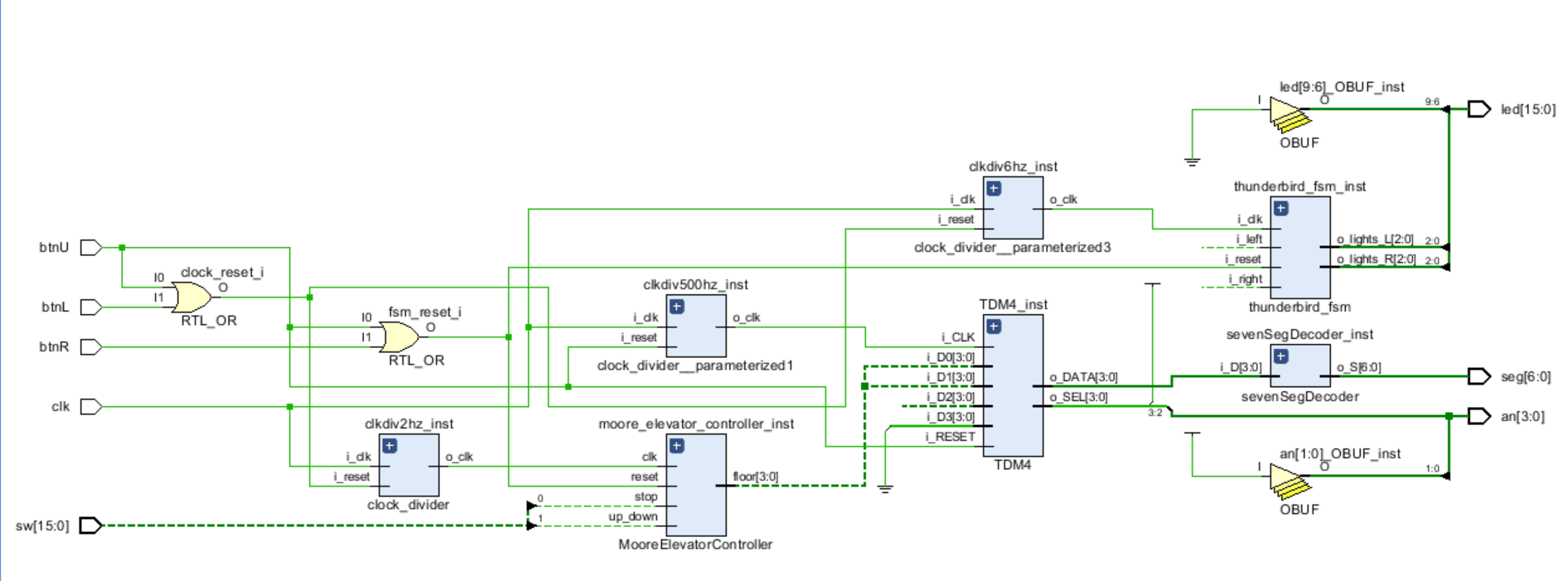
With two additional functionalities complete, we created a new diagram to represent the code, shown below:

Figure 7 – Top Level Design with moving lights functionality

Next, the final functionality we added was the ability to choose the destination floor. This functionality required using the first 4 switches to choose a floor to go to, then once the “go” button is hit the elevator moves to that floor.

To do this we did not have to implement any more components, but we did have to add a process in our top\_basys3 file. This process first needed to check if btnC, the go button was pushed, setting a signal called w\_goPress to 1 if it was. The process then had to check to see if the fsm\_reset was activated, which was the same reset as was feeding into the elevator controller and the thunderbird FSM. If the reset was activated, then the wire that originally was connected to the stop switch would be flipped to 1, stopping the elevator controller, and the w\_goPress wire would be set to 0. After those two parameters had been checked, the program would look for the rising edge of the clock. If no rising edge is found, nothing happens. Next, it looks to see if w\_goPress is activated indicating both pressure of the button and no pressure of the reset button. If w\_goPress is activated, the program then checks to see if it is above or below the desired floor. If it is below the desired floor, w\_updown, the wire originally used for switch 1, is set to 1 to raise the elevator. If it is above the desired floor, then w\_updown is set to 0 to lower the elevator. If neither case is true, that means the elevator is on the desired floor. If so, then w\_stop is switched to 1 and it starts over.

The hardest part of this portion was figuring out how to make the program not require btnC to be pressed in order for it to continue moving. The way we fixed this was to create the signal, w\_goPress, which would stay active until set inactive, regardless of whether or not btnC was being held down.

Another tricky part was figuring out exactly what each reset button was supposed to do. We had to get EI for our instructors to explain what the lab handout meant and how to incorporate that in our design. After this we were done, and the final schematic is included in the Final Results section.

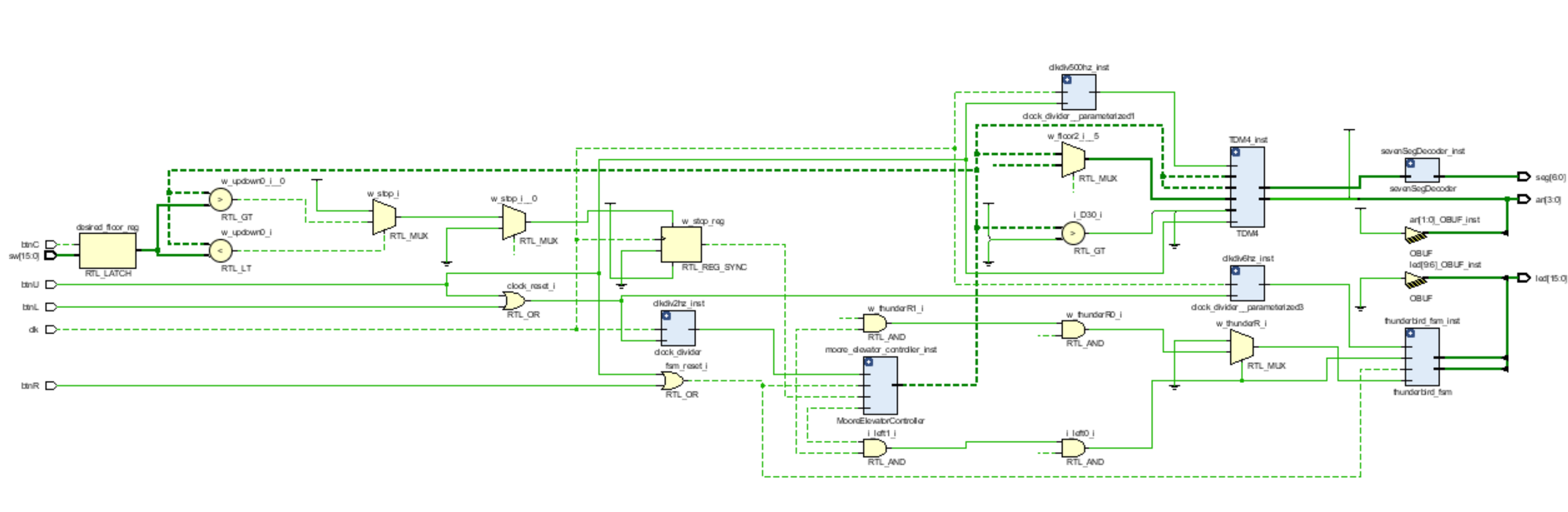
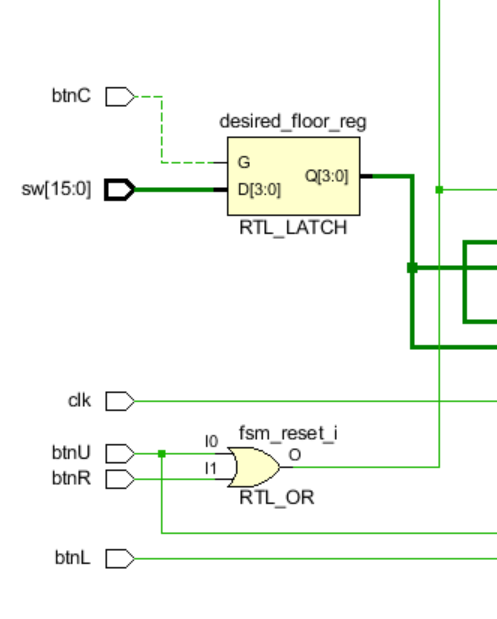
**Final Results:** Once we finished creating our design in VHDL we looked at the RTL schematic for the top\_basys3 design. We also took a look at the Clock Divider and its impact on the overall design. Figure 8 shows top\_basys3:

Figure 8 – Final Top Level Design

This schematic snapshot contains the right number and type of components we expected. It looks virtually the same as our high level design made during the early steps of the project showing our top-level entity and its internal architecture, represented in Figure 2. The only difference is that it shows more connections, muxes, and decoders than our design, but this is expected because our top level design left the details to the components and processes.

Figures 8.1 through 8.3 show the low level designs of the TDM4, SevenSegmentDecoder, and thunderbird\_fsm

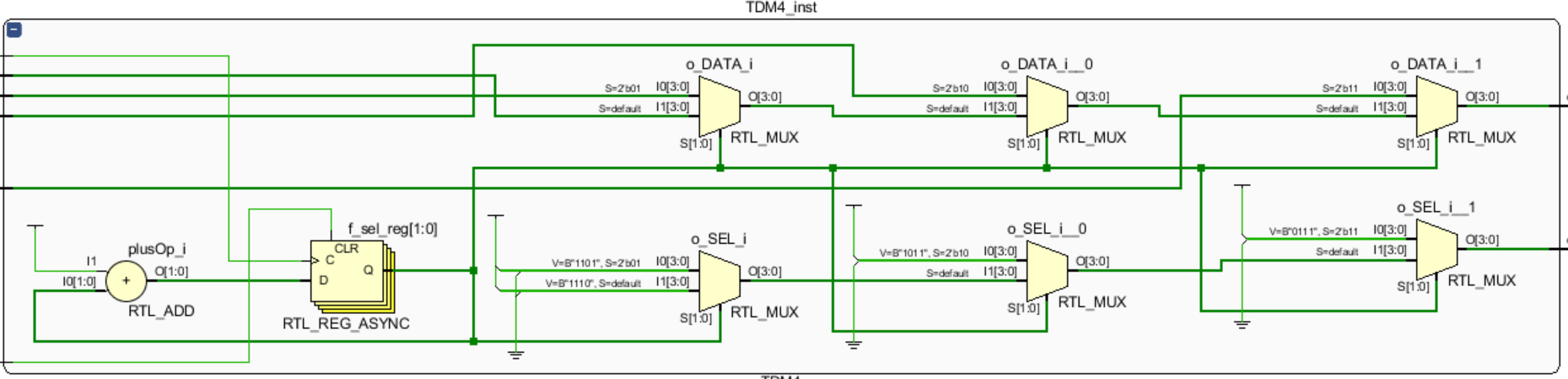


Figure 8.1 – LowLevel TDM4 Design

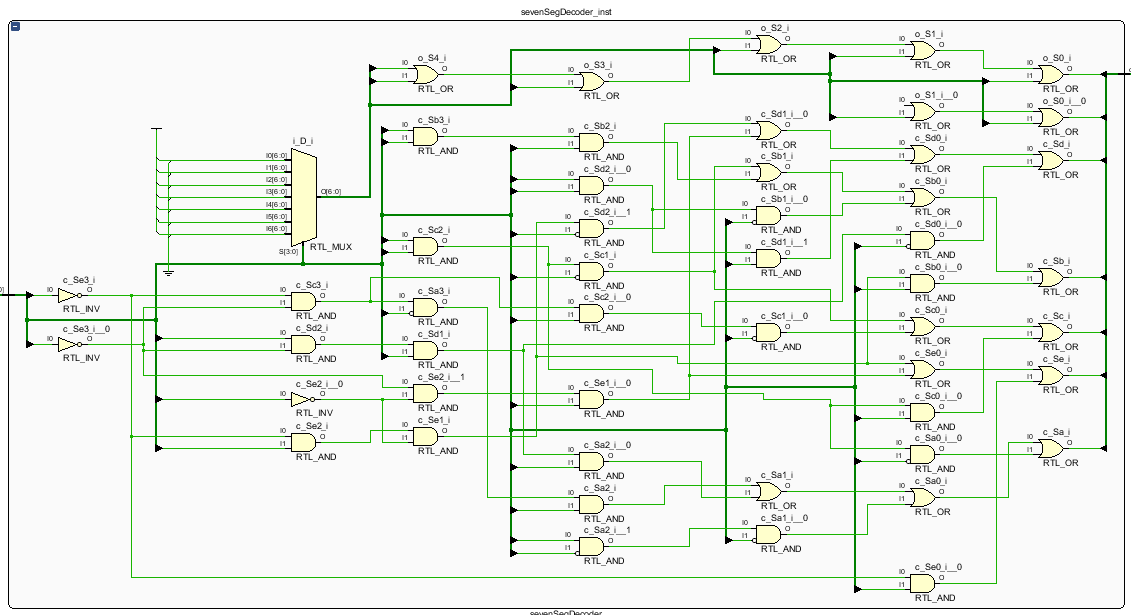


Figure 8.2 – LowLevel sevenSegDecoder Design

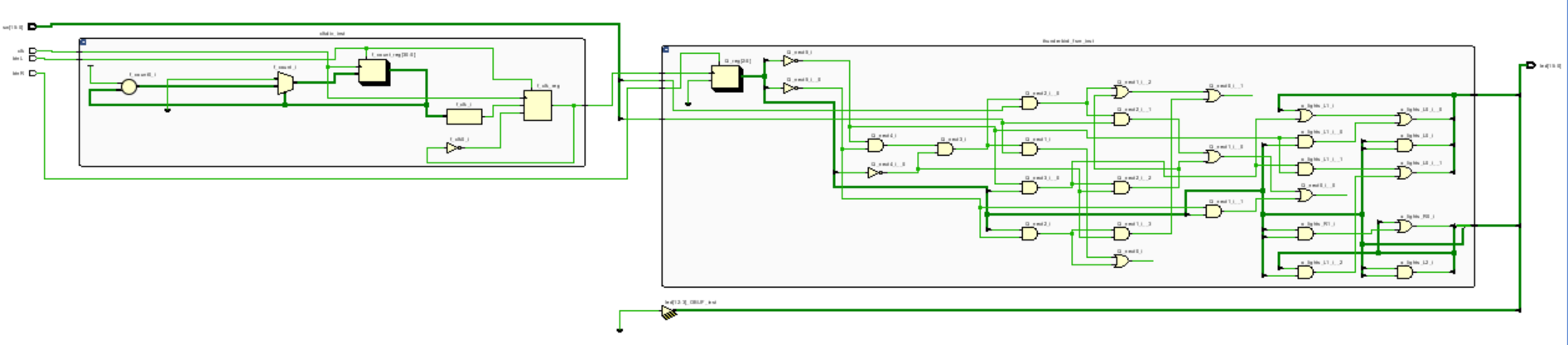


Figure 8.3 – LowLevel Thunderbird FSM Design

All of these designs are consistent not only with our expected outcomes, but also our previous ICEs and labs.

**Conclusions:** In this lab, we learned the hard way how important it is to thoroughly plan our approach, especially when working on something as complex as programming an elevator with several inputs. We were so anxious to begin diving into the actual project, and did not take as much time as we should have to make sure we completely understand the desired end state of our project. As a result, countless hours were spent just trying to clear up our confusion. We learned that if the planning is done correctly, the execution should be fairly straightforward and timely, even if roadblocks come up along the way.

**Reflection:**

* **Number of hours spent on Lab4 (Combined):** 100
* What portion of the lab was the most difficult for you? How did you overcome it?
  + The most difficult portion of this lab was the addition of each additional component of the advanced functionality. Although we had the individual vhd files pretty much complete from previous labs, there was a lot of trial and error involved in trying to figure out how
* What lessons, previous assignments, or activities did you find helpful is completing this lab?
  + This lab was a culmination of what we had learned before in previous labs and In-Class Exercises - specifically Lab 2, Lab 3, ICE5 and ICE6.
* What suggestions do you have for improving Lab4 in future years? Be specific. Ex: “The instructions were confusing” does not help. What parts of the instructions were confusing
  + For a lab of this complexity, it would have been better to have each portion of the lab due every other lesson instead of every lesson. This allows more time to ask questions and schedule EI. Between the lab submissions, the Zybooks reading, daily homeworks, and completing asynchronous video lectures all while still meeting for class, we found ourselves spending more time on ECE than in all other classes combined. This makes it much more difficult to learn the material we are supposed to learn. We have already spent 5 hours on this project today trying to write this report, and that is what most of our days look like.
  + Also, there was minimal instruction concerning the final two functionalities (the last of which we never completed). It would have been helpful to have more information and instruction about how to complete them, instead of going to our friends who were experienced programmers.

Appendix 1: top\_basys3.vhd

--+----------------------------------------------------------------------------

--|

--| COPYRIGHT 2018 United States Air Force Academy All rights reserved.

--|

--| United States Air Force Academy \_\_ \_\_\_\_\_\_\_ \_\_\_ \_\_\_\_\_\_\_\_\_

--| Dept of Electrical & / / / / \_\_\_// | / \_\_\_\_/ |

--| Computer Engineering / / / /\\_\_ \/ /| | / /\_ / /| |

--| 2354 Fairchild Drive Ste 2F6 / /\_/ /\_\_\_/ / \_\_\_ |/ \_\_/ / \_\_\_ |

--| USAF Academy, CO 80840 \\_\_\_\_//\_\_\_\_/\_/ |\_/\_/ /\_/ |\_|

--|

--| ---------------------------------------------------------------------------

--|

--| FILENAME : top\_basys3.vhd

--| AUTHOR(S) : Capt Phillip Warner (modified by C3C Lauren Humpherys and C3C Christopher Katz)

--| CREATED : 3/9/2018 Modified by Capt Dan Johnson (3/30/2020), again on 4/27/2020

--| DESCRIPTION : This file implements the top level module for a BASYS 3 to

--| drive the Lab 4 Design Project (Advanced Elevator Controller).

--|

--| Inputs: clk --> 100 MHz clock from FPGA

--| btnL --> Rst Clk

--| btnR --> Rst FSM

--| btnU --> Rst Master

--| btnC --> GO (request floor)

--| sw(15:12) --> Passenger location (floor select bits)

--| sw(3:0) --> Desired location (floor select bits)

--| - Minumum FUNCTIONALITY ONLY: sw(1) --> up\_down, sw(0) --> stop

--|

--| Outputs: led --> indicates elevator movement with sweeping pattern (additional functionality)

--| - led(10) --> led(15) = MOVING UP

--| - led(5) --> led(0) = MOVING DOWN

--| - ALL OFF = NOT MOVING

--| an(3:0) --> seven-segment display anode active-low enable (AN3 ... AN0)

--| seg(6:0) --> seven-segment display cathodes (CG ... CA. DP unused)

--|

--| DOCUMENTATION : Col Neff pointed out that not all names of our variables in MooreElevatorController

--| did not match the ones topBasys3.vhd

--|

--+----------------------------------------------------------------------------

--|

--| REQUIRED FILES :

--|

--| Libraries : ieee

--| Packages : std\_logic\_1164, numeric\_std

--| Files : MooreElevatorController.vhd, clock\_divider.vhd, sevenSegDecoder.vhd

--| thunderbird\_fsm.vhd, sevenSegDecoder, TDM4.vhd

--|

--+----------------------------------------------------------------------------

--|

--| NAMING CONVENSIONS :

--|

--| xb\_<port name> = off-chip bidirectional port ( \_pads file )

--| xi\_<port name> = off-chip input port ( \_pads file )

--| xo\_<port name> = off-chip output port ( \_pads file )

--| b\_<port name> = on-chip bidirectional port

--| i\_<port name> = on-chip input port

--| o\_<port name> = on-chip output port

--| c\_<signal name> = combinatorial signal

--| f\_<signal name> = synchronous signal

--| ff\_<signal name> = pipeline stage (ff\_, fff\_, etc.)

--| <signal name>\_n = active low signal

--| w\_<signal name> = top level wiring signal

--| g\_<generic name> = generic

--| k\_<constant name> = constant

--| v\_<variable name> = variable

--| sm\_<state machine type> = state machine type definition

--| s\_<signal name> = state name

--|

--+----------------------------------------------------------------------------

**library** ieee**;**

**use** ieee**.**std\_logic\_1164**.all;**

**use** ieee**.**numeric\_std**.all;**

**entity** top\_basys3 **is**

**port(**

clk **:** **in** std\_logic**;** -- native 100MHz FPGA clock

-- Switches (16 total)

sw **:** **in** std\_logic\_vector**(**15 **downto** 0**);**

-- Buttons (5 total)

btnC **:** **in** std\_logic**;** -- GO

btnU **:** **in** std\_logic**;** -- master\_reset

btnL **:** **in** std\_logic**;** -- clk\_reset

btnR **:** **in** std\_logic**;** -- fsm\_reset

--btnD : in std\_logic;

-- LEDs (16 total)

led **:** **out** std\_logic\_vector**(**15 **downto** 0**);**

-- 7-segment display segments (active-low cathodes)

seg **:** **out** std\_logic\_vector**(**6 **downto** 0**);**

-- 7-segment display active-low enables (anodes)

an **:** **out** std\_logic\_vector**(**3 **downto** 0**)**

**);**

**end** top\_basys3**;**

**architecture** top\_basys3\_arch **of** top\_basys3 **is**

-- declare components and signals

**component** MooreElevatorController **is**

**Port** **(** clk **:** **in** STD\_LOGIC**;**

reset **:** **in** STD\_LOGIC**;**

stop **:** **in** STD\_LOGIC**;**

up\_down **:** **in** STD\_LOGIC**;**

floor **:** **out** STD\_LOGIC\_VECTOR**(**3 **DOWNTO** 0**)**

**);**

**end** **component** MooreElevatorController**;**

**component** clock\_divider **is**

**generic** **(** **constant** k\_DIV **:** natural **:=** 2 **);**

**port** **(** i\_clk **:** **in** std\_logic**;** -- basys3 clk

i\_reset **:** **in** std\_logic**;** -- asynchronous

o\_clk **:** **out** std\_logic -- divided (slow) clock

**);**

**end** **component** clock\_divider**;**

**component** sevenSegDecoder **is**

**port** **(**

i\_D **:** **in** std\_logic\_vector **(**3 **downto** 0**);**

o\_S **:** **out** std\_logic\_vector **(**6 **downto** 0**)**

**);**

**end** **component** sevenSegDecoder**;**

**component** TDM4 **is**

**generic** **(** **constant** k\_WIDTH **:** natural **:=** 4**);** -- bits in input and output

**Port** **(** i\_CLK **:** **in** STD\_LOGIC**;**

i\_RESET **:** **in** STD\_LOGIC**;** -- asynchronous

i\_D3 **:** **in** STD\_LOGIC\_VECTOR **(**k\_WIDTH **-** 1 **downto** 0**);**

i\_D2 **:** **in** STD\_LOGIC\_VECTOR **(**k\_WIDTH **-** 1 **downto** 0**);**

i\_D1 **:** **in** STD\_LOGIC\_VECTOR **(**k\_WIDTH **-** 1 **downto** 0**);**

i\_D0 **:** **in** STD\_LOGIC\_VECTOR **(**k\_WIDTH **-** 1 **downto** 0**);**

o\_DATA **:** **out** STD\_LOGIC\_VECTOR **(**k\_WIDTH **-** 1 **downto** 0**);** --7seg decoder

o\_SEL **:** **out** STD\_LOGIC\_VECTOR **(**3 **downto** 0**)** -- selected data line (one-cold) anode

**);**

**end** **component** TDM4**;**

**component** thunderbird\_fsm **is**

**port(**

i\_clk**,** i\_reset **:** **in** std\_logic**;**

i\_left**,** i\_right **:** **in** std\_logic**;**

o\_lights\_L **:** **out** std\_logic\_vector**(**2 **downto** 0**);**

o\_lights\_R **:** **out** std\_logic\_vector**(**2 **downto** 0**)**

**);**

**end** **component** thunderbird\_fsm**;**

-- Reset signals

**signal** fsm\_reset **:** std\_logic**;**

**signal** clock\_reset **:** std\_logic**;**

-- MooreElevatorController signals

**signal** w\_clk\_1 **:** std\_logic**;**

**signal** w\_updown **:** std\_logic**;**

**signal** w\_stop **:** std\_logic**;**

**signal** w\_floor **:** std\_logic\_vector**(**3 **downto** 0**);**

-- TDM4 signals

**signal** w\_clk\_2 **:** std\_logic**;**

**signal** w\_floor1 **:** std\_logic\_vector**(**3 **downto** 0**);**

**signal** w\_floor2 **:** std\_logic\_vector**(**3 **downto** 0**);**

**signal** w\_data\_out **:** std\_logic\_vector**(**3 **downto** 0**);**

-- Thunderbird signals

**signal** w\_clk\_3 **:** std\_logic**;**

**signal** w\_thunderL **:** std\_logic**;**

**signal** w\_thunderR **:** std\_logic**;**

**signal** w\_led\_R **:** std\_logic\_vector**(**2 **downto** 0**);**

**signal** w\_led\_L **:** std\_logic\_vector**(**2 **downto** 0**);**

-- Change Input Signals

**signal** desired\_floor **:** std\_logic\_vector**(**3 **downto** 0**);**

**signal** current\_floor **:** std\_logic\_vector**(**3 **downto** 0**);**

**signal** w\_goPress **:** std\_logic**;**

**signal** w\_sel **:** std\_logic**;**

**begin**

-- PORT MAPS ----------------------------------------

moore\_elevator\_controller\_inst **:** MooreElevatorController

**port** **map(**

clk **=>** w\_clk\_1**,**

reset **=>** fsm\_reset**,**

up\_down **=>** w\_updown**,**

stop **=>** w\_stop**,**

floor **=>** w\_floor

**);**

clkdiv2hz\_inst **:** clock\_divider --instantiation of clock\_divider to take

**generic** **map** **(** k\_DIV **=>** 25000000 **)** -- 2 Hz clock from 100 MHz

**port** **map** **(** -- MooreElevatorController

i\_clk **=>** clk**,**

i\_reset **=>** clock\_reset**,**

o\_clk **=>** w\_clk\_1

**);**

clkdiv500hz\_inst **:** clock\_divider --instantiation of clock\_divider to take

**generic** **map** **(** k\_DIV **=>** 50000 **)** -- 500 Hz clock from 100 MHz

**port** **map** **(** -- TDM4

i\_clk **=>** clk**,**

i\_reset **=>** btnU**,**

o\_clk **=>** w\_clk\_2

**);**

clkdiv6hz\_inst **:** clock\_divider --instantiation of clock\_divider to take

**generic** **map** **(** k\_DIV **=>** 6250000 **)** -- 6 Hz clock from 100 MHz

**port** **map** **(** -- Thunderbird

i\_clk **=>** clk**,**

i\_reset **=>** clock\_reset**,**

o\_clk **=>** w\_clk\_3

**);**

sevenSegDecoder\_inst**:** sevenSegDecoder

**port** **map(**

i\_D **=>** w\_data\_out**,**

o\_S**(**6**)** **=>** seg**(**6**),**

o\_S**(**5**)** **=>** seg**(**5**),**

o\_S**(**4**)** **=>** seg**(**4**),**

o\_S**(**3**)** **=>** seg**(**3**),**

o\_S**(**2**)** **=>** seg**(**2**),**

o\_S**(**1**)** **=>** seg**(**1**),**

o\_S**(**0**)** **=>** seg**(**0**)**

**);**

TDM4\_inst **:** TDM4

**port** **map(**

i\_CLK **=>** w\_clk\_2**,**

i\_RESET **=>** btnU**,**

i\_D3 **=>** w\_floor1**,**

i\_D2 **=>** w\_floor2**,**

i\_D1 **=>** w\_floor**,**

i\_D0 **=>** w\_floor**,**

o\_DATA **=>** w\_data\_out**,**

o\_SEL**(**3**)** **=>** an**(**3**),**

o\_SEL**(**2**)** **=>** an**(**2**),**

o\_SEL**(**1**)** **=>** an**(**1**),**

o\_SEL**(**0**)** **=>** an**(**0**)**

**);**

thunderbird\_fsm\_inst **:** thunderbird\_fsm

**port** **map** **(**

i\_reset **=>** fsm\_reset**,**

i\_clk **=>** w\_clk\_3**,**

i\_left **=>** w\_thunderL**,**

i\_right **=>** w\_thunderR**,**

o\_lights\_R**(**0**)** **=>** w\_led\_R**(**2**),**

o\_lights\_R**(**1**)** **=>** w\_led\_R**(**1**),**

o\_lights\_R**(**2**)** **=>** w\_led\_R**(**0**),**

o\_lights\_L**(**0**)** **=>** w\_led\_L**(**0**),**

o\_lights\_L**(**1**)** **=>** w\_led\_L**(**1**),**

o\_lights\_L**(**2**)** **=>** w\_led\_L**(**2**)**

**);**

-- CONCURRENT STATEMENTS ----------------------------

an**(**1**)** **<=** '1'**;** -- Ground unused anodes

an**(**0**)** **<=** '1'**;**

fsm\_reset **<=** btnU or btnR**;** -- btnU or btnR may be pressed to reset all FSMs (except TDM4 btnU only)

clock\_reset **<=** btnU or btnL**;** -- Resets all clocks except the 500 Hz clock that controls TDM4

led**(**9 **downto** 6**)** **<=** "0000"**;** -- Ground unused LEDs

w\_floor1 **<=** "0001" **when** w\_floor **>** "1001" **else** -- Enables the use of two anodes for double-digit floors

"0000"**;**

w\_floor2 **<=** w\_floor **when** w\_floor **<** "1010" **else**

"0000" **when** w\_floor **=** "1010" **else**

"0001" **when** w\_floor **=** "1011" **else**

"0010" **when** w\_floor **=** "1100" **else**

"0011" **when** w\_floor **=** "1101" **else**

"0100" **when** w\_floor **=** "1110" **else**

"0101" **when** w\_floor **=** "1111" **else**

"0001"**;**

moving\_lights\_proc **:** **process** **(**w\_updown**,** w\_stop**,** w\_floor**)** -- Illuminates left or right LEDs, depending on elevator direction

**begin**

**if** w\_updown **=** '1' and w\_stop **=** '0' and w\_floor **<** "1111" **then**

w\_thunderL **<=** '1'**;**

w\_thunderR **<=** '0'**;**

**elsif** w\_updown **=** '0' and w\_stop **=** '0' and w\_floor **>** "0001" **then**

w\_thunderL **<=** '0'**;**

w\_thunderR **<=** '1'**;**

**else**

w\_thunderL **<=** '0'**;**

w\_thunderR **<=** '0'**;**

**end** **if;**

**end** **process** moving\_lights\_proc**;**

change\_input\_proc **:** **process** **(**clk**)** -- Allows user to send elevator to predetermined desired floor

**begin**

**if** **(**btnC **=** '1'**)** **then**

desired\_floor **<=** sw**(**3 **downto** 0**);**

w\_goPress **<=** '1'**;**

**end** **if;**

**if** **(**fsm\_reset **=** '1'**)** **then**

w\_stop **<=** '1'**;**

w\_goPress **<=** '0'**;**

**end** **if;**

**if** **(rising\_edge(**clk**))** **then**

**if** **(**w\_goPress **=** '1'**)** **then**

**if** **(**w\_floor **<** desired\_floor**)** **then**

w\_updown **<=** '1'**;**

w\_stop **<=** '0'**;**

**elsif** **(**w\_floor **>** desired\_floor**)** **then**

w\_updown **<=** '0'**;**

w\_stop **<=** '0'**;**

**else**

w\_stop **<=** '1'**;**

w\_goPress **<=** '0'**;**

**end** **if;**

**else**

w\_stop **<=** '1'**;**

w\_updown **<=** '0'**;**

**end** **if;**

**end** **if;**

**end** **process** change\_input\_proc**;**

-- Need to wire LEDs that light up simultaneously

led**(**15**)** **<=** w\_led\_L**(**2**);**

led**(**14**)** **<=** w\_led\_L**(**2**);**

led**(**13**)** **<=** w\_led\_L**(**1**);**

led**(**12**)** **<=** w\_led\_L**(**1**);**

led**(**11**)** **<=** w\_led\_L**(**0**);**

led**(**10**)** **<=** w\_led\_L**(**0**);**

led**(**5**)** **<=** w\_led\_R**(**2**);**

led**(**4**)** **<=** w\_led\_R**(**2**);**

led**(**3**)** **<=** w\_led\_R**(**1**);**

led**(**2**)** **<=** w\_led\_R**(**1**);**

led**(**1**)** **<=** w\_led\_R**(**0**);**

led**(**0**)** **<=** w\_led\_R**(**0**);**

**end** top\_basys3\_arch**;**

Appendix 2: MooreElevatorController.vhd

--+----------------------------------------------------------------------------

--|

--| COPYRIGHT 2018 United States Air Force Academy All rights reserved.

--|

--| United States Air Force Academy \_\_ \_\_\_\_\_\_\_ \_\_\_ \_\_\_\_\_\_\_\_\_

--| Dept of Electrical & / / / / \_\_\_// | / \_\_\_\_/ |

--| Computer Engineering / / / /\\_\_ \/ /| | / /\_ / /| |

--| 2354 Fairchild Drive Ste 2F6 / /\_/ /\_\_\_/ / \_\_\_ |/ \_\_/ / \_\_\_ |

--| USAF Academy, CO 80840 \\_\_\_\_//\_\_\_\_/\_/ |\_/\_/ /\_/ |\_|

--|

--| ---------------------------------------------------------------------------

--|

--| FILENAME : MooreElevatorController.vhd

--| AUTHOR(S) : Capt Phillip Warner, Lauren Humpherys, Christopher Katz

--| CREATED : 03/2018 Modified by Capt Johnson on 18 March 2020, again on 27 April 2020

--| DESCRIPTION : This file implements the ICE5 Basic elevator controller (Moore Machine)

--|

--| The system is specified as follows:

--| - The elevator controller will traverse four floors (numbered 1 to 4).

--| - It has two external inputs, Up\_Down and Stop.

--| - When Up\_Down is active and Stop is inactive, the elevator will move up

--| until it reaches the top floor (one floor per clock, of course).

--| - When Up\_Down is inactive and Stop is inactive, the elevator will move down

--| until it reaches the bottom floor (one floor per clock).

--| - When Stop is active, the system stops at the current floor.

--| - When the elevator is at the top floor, it will stay there until Up\_Down

--| goes inactive while Stop is inactive. Likewise, it will remain at the bottom

--| until told to go up and Stop is inactive.

--| - The system should output the floor it is on (1 – 4) as a four-bit binary number.

--|

--| DOCUMENTATION : Only help recieved in EI from Capt Johnson

--|

--+----------------------------------------------------------------------------

--|

--| REQUIRED FILES :

--|

--| Libraries : ieee

--| Packages : std\_logic\_1164, numeric\_std, unisim

--| Files : None

--|

--+----------------------------------------------------------------------------

--|

--| NAMING CONVENSIONS :

--|

--| xb\_<port name> = off-chip bidirectional port ( \_pads file )

--| xi\_<port name> = off-chip input port ( \_pads file )

--| xo\_<port name> = off-chip output port ( \_pads file )

--| b\_<port name> = on-chip bidirectional port

--| i\_<port name> = on-chip input port

--| o\_<port name> = on-chip output port

--| c\_<signal name> = combinatorial signal

--| f\_<signal name> = synchronous signal

--| ff\_<signal name> = pipeline stage (ff\_, fff\_, etc.)

--| <signal name>\_n = active low signal

--| w\_<signal name> = top level wiring signal

--| g\_<generic name> = generic

--| k\_<constant name> = constant

--| v\_<variable name> = variable

--| sm\_<state machine type> = state machine type definition

--| s\_<signal name> = state name

--|

--+----------------------------------------------------------------------------

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**entity** MooreElevatorController **is**

**Port** **(** clk **:** **in** STD\_LOGIC**;**

reset **:** **in** STD\_LOGIC**;**

stop **:** **in** STD\_LOGIC**;**

up\_down **:** **in** STD\_LOGIC**;**

floor **:** **out** STD\_LOGIC\_VECTOR **(**3 **downto** 0**)**

**);**

**end** MooreElevatorController**;**

**architecture** Behavioral **of** MooreElevatorController **is**

-- Below you create a new variable type! You also define what values that

-- variable type can take on. Now you can assign a signal as

-- "sm\_floor" the same way you'd assign a signal as std\_logic

-- how would you modify this to go up to 15 floors?

-- ANSWER: To go up to 15 floors you would add varriables s\_floor5 thorugh s\_floor15

**type** sm\_floor **is** **(**s\_floor1**,** s\_floor2**,** s\_floor3**,** s\_floor4**,** s\_floor5**,** s\_floor6**,** s\_floor7**,** s\_floor8**,** s\_floor9**,** s\_floor10**,** s\_floor11**,** s\_floor12**,** s\_floor13**,** s\_floor14**,** s\_floor15**);**

-- Here you create variables that can take on the values

-- defined above.

**signal** current\_state**,** next\_state **:** sm\_floor**;**

**begin**

-- Next state logic ---------------------------------

next\_state **<=** s\_floor1 **when** **(**current\_state **=** s\_floor1 and up\_down **=** '0' and stop **=** '0'**)** or **(**current\_state **=** s\_floor1 and stop **=** '1'**)** or **(**current\_state **=** s\_floor2 and up\_down **=** '0' and stop **=** '0'**)** **else**

s\_floor2 **when** **(**current\_state **=** s\_floor1 and up\_down **=** '1' and stop **=** '0'**)** or **(**current\_state **=** s\_floor2 and stop **=** '1'**)** or **(**current\_state **=** s\_floor3 and up\_down **=** '0' and stop **=** '0'**)** **else**

s\_floor3 **when** **(**current\_state **=** s\_floor2 and up\_down **=** '1' and stop **=** '0'**)** or **(**current\_state **=** s\_floor3 and stop **=** '1'**)** or **(**current\_state **=** s\_floor4 and up\_down **=** '0' and stop **=** '0'**)** **else**

s\_floor4 **when** **(**current\_state **=** s\_floor3 and up\_down **=** '1' and stop **=** '0'**)** or **(**current\_state **=** s\_floor4 and stop **=** '1'**)** or **(**current\_state **=** s\_floor5 and up\_down **=** '0' and stop **=** '0'**)** **else**

s\_floor5 **when** **(**current\_state **=** s\_floor4 and up\_down **=** '1' and stop **=** '0'**)** or **(**current\_state **=** s\_floor5 and stop **=** '1'**)** or **(**current\_state **=** s\_floor6 and up\_down **=** '0' and stop **=** '0'**)** **else**

s\_floor6 **when** **(**current\_state **=** s\_floor5 and up\_down **=** '1' and stop **=** '0'**)** or **(**current\_state **=** s\_floor6 and stop **=** '1'**)** or **(**current\_state **=** s\_floor7 and up\_down **=** '0' and stop **=** '0'**)** **else**

s\_floor7 **when** **(**current\_state **=** s\_floor6 and up\_down **=** '1' and stop **=** '0'**)** or **(**current\_state **=** s\_floor7 and stop **=** '1'**)** or **(**current\_state **=** s\_floor8 and up\_down **=** '0' and stop **=** '0'**)** **else**

s\_floor8 **when** **(**current\_state **=** s\_floor7 and up\_down **=** '1' and stop **=** '0'**)** or **(**current\_state **=** s\_floor8 and stop **=** '1'**)** or **(**current\_state **=** s\_floor9 and up\_down **=** '0' and stop **=** '0'**)** **else**

s\_floor9 **when** **(**current\_state **=** s\_floor8 and up\_down **=** '1' and stop **=** '0'**)** or **(**current\_state **=** s\_floor9 and stop **=** '1'**)** or **(**current\_state **=** s\_floor10 and up\_down **=** '0' and stop **=** '0'**)** **else**

s\_floor10 **when** **(**current\_state **=** s\_floor9 and up\_down **=** '1' and stop **=** '0'**)** or **(**current\_state **=** s\_floor10 and stop **=** '1'**)** or **(**current\_state **=** s\_floor11 and up\_down **=** '0' and stop **=** '0'**)** **else**

s\_floor11 **when** **(**current\_state **=** s\_floor10 and up\_down **=** '1' and stop **=** '0'**)** or **(**current\_state **=** s\_floor11 and stop **=** '1'**)** or **(**current\_state **=** s\_floor12 and up\_down **=** '0' and stop **=** '0'**)** **else**

s\_floor12 **when** **(**current\_state **=** s\_floor11 and up\_down **=** '1' and stop **=** '0'**)** or **(**current\_state **=** s\_floor12 and stop **=** '1'**)** or **(**current\_state **=** s\_floor13 and up\_down **=** '0' and stop **=** '0'**)** **else**

s\_floor13 **when** **(**current\_state **=** s\_floor12 and up\_down **=** '1' and stop **=** '0'**)** or **(**current\_state **=** s\_floor13 and stop **=** '1'**)** or **(**current\_state **=** s\_floor14 and up\_down **=** '0' and stop **=** '0'**)** **else**

s\_floor14 **when** **(**current\_state **=** s\_floor13 and up\_down **=** '1' and stop **=** '0'**)** or **(**current\_state **=** s\_floor14 and stop **=** '1'**)** or **(**current\_state **=** s\_floor15 and up\_down **=** '0' and stop **=** '0'**)** **else**

s\_floor15 **when** **(**current\_state **=** s\_floor14 and up\_down **=** '1' and stop **=** '0'**)** or **(**current\_state **=** s\_floor15 and stop **=** '1'**)** or **(**current\_state **=** s\_floor15 and up\_down **=** '1' and stop **=** '0'**)** **else** s\_floor1**;**

-- State memory ------------

register\_proc **:** **process** **(**clk**,** reset**)**

**begin**

**if** reset **=** '1' **then**

current\_state **<=** s\_floor1**;**

**elsif** **(rising\_edge(**clk**))** **then**

current\_state **<=** next\_state**;**

**end** **if;**

**end** **process** register\_proc**;**

-- reset is active high and will return elevator to floor1

-- Output logic ---------------------------------

-- default is floor1

floor **<=** X"1" **when** current\_state **=** s\_floor1 **else**

X"2" **when** current\_state **=** s\_floor2 **else**

X"3" **when** current\_state **=** s\_floor3 **else**

X"4" **when** current\_state **=** s\_floor4 **else**

X"5" **when** current\_state **=** s\_floor5 **else**

X"6" **when** current\_state **=** s\_floor6 **else**

X"7" **when** current\_state **=** s\_floor7 **else**

X"8" **when** current\_state **=** s\_floor8 **else**

X"9" **when** current\_state **=** s\_floor9 **else**

X"A" **when** current\_state **=** s\_floor10 **else**

X"B" **when** current\_state **=** s\_floor11 **else**

X"C" **when** current\_state **=** s\_floor12 **else**

X"D" **when** current\_state **=** s\_floor13 **else**

X"E" **when** current\_state **=** s\_floor14 **else**

X"F" **when** current\_state **=** s\_floor15 **else** X"1"**;**

**end** Behavioral**;**

Appendix 3: sevenSegmentDecoder.vhd

--+----------------------------------------------------------------------------

--|

--| COPYRIGHT 2017 United States Air Force Academy All rights reserved.

--|

--| United States Air Force Academy \_\_ \_\_\_\_\_\_\_ \_\_\_ \_\_\_\_\_\_\_\_\_

--| Dept of Electrical & / / / / \_\_\_// | / \_\_\_\_/ |

--| Computer Engineering / / / /\\_\_ \/ /| | / /\_ / /| |

--| 2354 Fairchild Drive Ste 2F6 / /\_/ /\_\_\_/ / \_\_\_ |/ \_\_/ / \_\_\_ |

--| USAF Academy, CO 80840 \\_\_\_\_//\_\_\_\_/\_/ |\_/\_/ /\_/ |\_|

--|

--| ---------------------------------------------------------------------------

--|

--| FILENAME : sevenSegDecoderder.vhd

--| AUTHOR(S) : C3C Lauren Humpherys and C3C Christopher Katz

--| CREATED : 02/15/2020, Modified 04/2020

--| DESCRIPTION : Program decoder to generate appropriate output for input of

--| 7 segments of the second annode (1s column).

--| DOCUMENTATION : See top\_basys3.vhd

--|

--+----------------------------------------------------------------------------

--|

--| REQUIRED FILES :

--|

--| Libraries : ieee

--| Packages : std\_logic\_1164, numeric\_std, unisim

--| Files : None.

--|

--+----------------------------------------------------------------------------

--|

--| NAMING CONVENSIONS :

--|

--| xb\_<port name> = off-chip bidirectional port ( \_pads file )

--| xi\_<port name> = off-chip input port ( \_pads file )

--| xo\_<port name> = off-chip output port ( \_pads file )

--| b\_<port name> = on-chip bidirectional port

--| i\_<port name> = on-chip input port

--| o\_<port name> = on-chip output port

--| c\_<signal name> = combinatorial signal

--| f\_<signal name> = synchronous signal

--| ff\_<signal name> = pipeline stage (ff\_, fff\_, etc.)

--| <signal name>\_n = active low signal

--| w\_<signal name> = top level wiring signal

--| g\_<generic name> = generic

--| k\_<constant name> = constant

--| v\_<variable name> = variable

--| sm\_<state machine type> = state machine type definition

--| s\_<signal name> = state name

--|

--+----------------------------------------------------------------------------

**library** ieee**;**

**use** ieee**.**std\_logic\_1164**.all;**

**use** ieee**.**numeric\_std**.all;**

**library** unisim**;**

**use** UNISIM**.**Vcomponents**.ALL;**

**entity** sevenSegDecoder **is**

**port(**

i\_D **:** **in** std\_logic\_vector**(**3 **downto** 0**);**

o\_S **:** **out** std\_logic\_vector**(**6 **downto** 0**)**

**);**

**end** sevenSegDecoder**;**

**architecture** sevenSegDecoder\_arch **of** sevenSegDecoder **is**

**signal** c\_Sa **:** std\_logic**;**

**signal** c\_Sb **:** std\_logic**;**

**signal** c\_Sc **:** std\_logic**;**

**signal** c\_Sd **:** std\_logic**;**

**signal** c\_Se **:** std\_logic**;**

**signal** c\_Sf **:** std\_logic**;**

**signal** c\_Sg **:** std\_logic**;**

**begin**

-- CONCURRENT STATEMENTS "MODULES" ------------------

o\_S**(**0**)** **<=** c\_Sa**;**

o\_S**(**1**)** **<=** c\_Sb**;**

o\_S**(**2**)** **<=** c\_Sc**;**

o\_S**(**3**)** **<=** c\_Sd**;**

o\_S**(**4**)** **<=** c\_Se**;**

o\_S**(**5**)** **<=** c\_Sf**;**

o\_S**(**6**)** **<=** c\_Sg**;**

--Sa = BC'D' + ABC' + A'B'C'D + AB'CD

c\_Sa **<=** **(**not i\_D**(**3**)** and not i\_D**(**2**)** and not i\_D**(**1**)** and i\_D**(**0**)** **)**

or **(** i\_D**(**3**)** and not i\_D**(**2**)** and i\_D**(**1**)** and i\_D**(**0**)** **)**

or **(** i\_D**(**2**)** and not i\_D**(**1**)** and not i\_D**(**0**)** **)**

or **(** i\_D**(**3**)** and i\_D**(**2**)** and not i\_D**(**1**)** **);**

--Sb = ABD' + A'BC'D + ACD + BCD'

c\_Sb **<=** **(**i\_D**(**3**)** and i\_D**(**2**)** and not i\_D**(**0**))**

or **(**i\_D**(**3**)** and i\_D**(**1**)** and i\_D**(**0**))**

or **(**i\_D**(**2**)** and i\_D**(**1**)** and not i\_D**(**0**))**

or **(**not i\_D**(**3**)** and i\_D**(**2**)** and not i\_D**(**1**)** and i\_D**(**0**));**

--Sc = ABD' + ABC + A'B'CD'

c\_Sc **<=** **(**i\_D**(**3**)** and i\_D**(**2**)** and not i\_D**(**0**))**

or **(**not i\_D**(**3**)** and not i\_D**(**2**)** and i\_D**(**1**)** and not i\_D**(**0**))**

or **(**i\_D**(**3**)** and i\_D**(**2**)** and i\_D**(**1**));**

--Sd = A'BC'D' + B'C'D + BCD + AB'CD'

c\_Sd **<=** **(**not i\_D**(**3**)** and i\_D**(**2**)** and not i\_D**(**1**)** and not i\_D**(**0**))**

or **(**not i\_D**(**2**)** and not i\_D**(**1**)** and i\_D**(**0**))**

or **(**i\_D**(**2**)** and i\_D**(**1**)** and i\_D**(**0**))**

or **(**i\_D**(**3**)** and not i\_D**(**2**)** and i\_D**(**1**)** and not i\_D**(**0**));**

--Se = A'BC' + A'D + C'D

c\_Se **<=** **(**not i\_D**(**3**)** and i\_D**(**2**)** and not i\_D**(**1**))**

or **(**not i\_D**(**2**)** and not i\_D**(**1**)** and i\_D**(**0**))**

or **(**not i\_D**(**3**)** and i\_D**(**0**));**

--Sg = A'B'C' + A'BCD

c\_Sg **<=** '1' **when** **(** **(**i\_D **=** x"0"**)** or

**(**i\_D **=** x"1"**)** or

**(**i\_D **=** x"7"**)** **)** **else** '0'**;**

--Sf = ABC' + A'B'D + A'B'C + A'CD

c\_Sf **<=** '1' **when** **(** **(**i\_D **=** x"1"**)** or

**(**i\_D **=** x"2"**)** or

**(**i\_D **=** x"3"**)** or

**(**i\_D **=** x"7"**)** or

**(**i\_D **=** x"C"**)** or

**(**i\_D **=** x"D"**)** **)** **else** '0'**;**

**end** sevenSegDecoder\_arch**;**

Appendix 4: thunderbird\_fsm.vhd

--+----------------------------------------------------------------------------

--|

--| COPYRIGHT 2017 United States Air Force Academy All rights reserved.

--|

--| United States Air Force Academy \_\_ \_\_\_\_\_\_\_ \_\_\_ \_\_\_\_\_\_\_\_\_

--| Dept of Electrical & / / / / \_\_\_// | / \_\_\_\_/ |

--| Computer Engineering / / / /\\_\_ \/ /| | / /\_ / /| |

--| 2354 Fairchild Drive Ste 2F6 / /\_/ /\_\_\_/ / \_\_\_ |/ \_\_/ / \_\_\_ |

--| USAF Academy, CO 80840 \\_\_\_\_//\_\_\_\_/\_/ |\_/\_/ /\_/ |\_|

--|

--| ---------------------------------------------------------------------------

--|

--| FILENAME : thunderbird\_fsm.vhd

--| AUTHOR(S) : C3C Lauren Humpherys and C3C Christopher Katz

--| CREATED : 03/30/2020

--| DESCRIPTION : Implementation of the thunderbird FSM module, which is used to

--| show the direction of elevator movement through the LED lights

--|

--| DOCUMENTATION : Captain Johnson helped us differentiate between state and

--| output logic. He also helped us correct major errors by telling us to uncomment

--| all switches and LED lights in the constraints file, and ground the LEDs not being

--| used for this lab. C3C Felix Zheng verified that our RTL Schematic was correct.

--|

--+----------------------------------------------------------------------------

--|

--| REQUIRED FILES :

--|

--| Libraries : ieee

--| Packages : std\_logic\_1164, numeric\_std, unisim

--| Files : None.

--|

--+----------------------------------------------------------------------------

--|

--| NAMING CONVENSIONS :

--|

--| xb\_<port name> = off-chip bidirectional port ( \_pads file )

--| xi\_<port name> = off-chip input port ( \_pads file )

--| xo\_<port name> = off-chip output port ( \_pads file )

--| b\_<port name> = on-chip bidirectional port

--| i\_<port name> = on-chip input port

--| o\_<port name> = on-chip output port

--| c\_<signal name> = combinatorial signal

--| f\_<signal name> = synchronous signal

--| ff\_<signal name> = pipeline stage (ff\_, fff\_, etc.)

--| <signal name>\_n = active low signal

--| w\_<signal name> = top level wiring signal

--| g\_<generic name> = generic

--| k\_<constant name> = constant

--| v\_<variable name> = variable

--| sm\_<state machine type> = state machine type definition

--| s\_<signal name> = state name

--|

--+----------------------------------------------------------------------------

**library** ieee**;**

**use** ieee**.**std\_logic\_1164**.all;**

**use** ieee**.**numeric\_std**.all;**

**library** unisim**;**

**use** UNISIM**.**Vcomponents**.ALL;**

**entity** thunderbird\_fsm **is**

**port(**

i\_clk**,** i\_reset **:** **in** std\_logic**;**

i\_left**,** i\_right **:** **in** std\_logic**;**

o\_lights\_L **:** **out** std\_logic\_vector**(**2 **downto** 0**);**

o\_lights\_R **:** **out** std\_logic\_vector**(**2 **downto** 0**)**

**);**

**end** thunderbird\_fsm**;**

**architecture** thunderbird\_FSM\_arch **of** thunderbird\_FSM **is**

**signal** Q**,** Q\_next**:**std\_logic\_vector**(**2 **downto** 0**)** **:=** "000"**;**

**begin**

-- State Transition Logic ------------------------------

-- Q(0)\* = Q(0)'Q(1)'Q(2)'R(sw) + Q(0)Q(1)'

Q\_next**(**0**)** **<=** **(**not Q**(**0**)** and not Q**(**1**)** and not Q**(**2**)** and i\_right**)** or **(**Q**(**0**)** and not Q**(**1**));**

-- Q(1)\* = Q(0)'Q(1)'Q(2)'L(sw)R(sw) + Q(0)'Q(1)Q(2)' + Q(1)'Q(2)

Q\_next**(**1**)** **<=** **(**not Q**(**0**)** and not Q**(**1**)** and not Q**(**2**)** and i\_left and i\_right**)** or **(**not Q**(**0**)** and Q**(**1**)** and not Q**(**2**))** or **(**not Q**(**1**)** and Q**(**2**));**

-- Q(2)\* = Q(0)'Q(1)'Q(2)'L(sw) + Q(0)'Q(1)Q(2)' + Q(0)Q(1)'Q(2)'

Q\_next**(**2**)** **<=** **(**not Q**(**0**)** and not Q**(**1**)** and not Q**(**2**)** and i\_left**)** or **(**not Q**(**0**)** and Q**(**1**)** and not Q**(**2**))** or **(**Q**(**0**)** and not Q**(**1**)** and not Q**(**2**));**

-- Output Logic --------------------------------------

-- One left light

o\_lights\_L**(**0**)** **<=** **(**Q**(**1**)** and Q**(**2**))** or **(**not Q**(**0**)** and Q**(**1**))** or **(**not Q**(**0**)** and Q**(**2**));**

-- Two left lights

o\_lights\_L**(**1**)** **<=** **(**not Q**(**0**)** and Q**(**1**))** or **(**Q**(**0**)** and Q**(**1**)** and Q**(**2**));**

-- Three left lights

o\_lights\_L**(**2**)** **<=** Q**(**1**)** and Q**(**2**);**

-- One right light

o\_lights\_R**(**0**)** **<=** Q**(**0**);**

-- Two right lights

o\_lights\_R**(**1**)** **<=** **(**Q**(**0**)** and Q**(**1**))** or **(**Q**(**0**)** and Q**(**2**));**

-- Three right lights

o\_lights\_R**(**2**)** **<=** Q**(**0**)** and Q**(**1**);**

-- State Memory with Asynchronous Reset-------------------

register\_proc **:** **process** **(**i\_clk**,** i\_reset**)**

**begin**

**if** i\_reset **=** '1' **then**

Q **<=** "000"**;** --Reset state is off

**elsif** **(rising\_edge(**i\_clk**))** **then**

Q **<=** Q\_next**;** --Next state becomes current state

**end** **if;**

**end** **process** register\_proc**;**

**end** thunderbird\_FSM\_arch**;**

Appendix 5: MooreElevatorController\_tb

--+----------------------------------------------------------------------------

--|

--| COPYRIGHT 2017 United States Air Force Academy All rights reserved.

--|

--| United States Air Force Academy \_\_ \_\_\_\_\_\_\_ \_\_\_ \_\_\_\_\_\_\_\_\_

--| Dept of Electrical & / / / / \_\_\_// | / \_\_\_\_/ |

--| Computer Engineering / / / /\\_\_ \/ /| | / /\_ / /| |

--| 2354 Fairchild Drive Ste 2F6 / /\_/ /\_\_\_/ / \_\_\_ |/ \_\_/ / \_\_\_ |

--| USAF Academy, CO 80840 \\_\_\_\_//\_\_\_\_/\_/ |\_/\_/ /\_/ |\_|

--|

--| ---------------------------------------------------------------------------

--|

--| FILENAME : MooreElevatorController\_tb.vhd (TEST BENCH)

--| AUTHOR(S) : Capt Phillip Warner, Lauren Humpherys, Christopher Katz

--| CREATED : 03/2017

--| DESCRIPTION : This file tests the Moore elevator controller module

--|

--| DOCUMENTATION : Only help recieved in EI from Capt Johnson

--|

--+----------------------------------------------------------------------------

--|

--| REQUIRED FILES :

--|

--| Libraries : ieee

--| Packages : std\_logic\_1164, numeric\_std, unisim

--| Files : MooreElevatorController.vhd

--|

--+----------------------------------------------------------------------------

--|

--| NAMING CONVENSIONS :

--|

--| xb\_<port name> = off-chip bidirectional port ( \_pads file )

--| xi\_<port name> = off-chip input port ( \_pads file )

--| xo\_<port name> = off-chip output port ( \_pads file )

--| b\_<port name> = on-chip bidirectional port

--| i\_<port name> = on-chip input port

--| o\_<port name> = on-chip output port

--| c\_<signal name> = combinatorial signal

--| f\_<signal name> = synchronous signal

--| ff\_<signal name> = pipeline stage (ff\_, fff\_, etc.)

--| <signal name>\_n = active low signal

--| w\_<signal name> = top level wiring signal

--| g\_<generic name> = generic

--| k\_<constant name> = constant

--| v\_<variable name> = variable

--| sm\_<state machine type> = state machine type definition

--| s\_<signal name> = state name

--|

--+----------------------------------------------------------------------------

**library** ieee**;**

**use** ieee**.**std\_logic\_1164**.all;**

**use** ieee**.**numeric\_std**.all;**

**entity** MooreElevatorController\_tb **is**

**end** MooreElevatorController\_tb**;**

**architecture** test\_bench **of** MooreElevatorController\_tb **is**

**component** MooreElevatorController **is**

**Port** **(** clk **:** **in** STD\_LOGIC**;**

reset **:** **in** STD\_LOGIC**;** -- synchronous

stop **:** **in** STD\_LOGIC**;**

up\_down **:** **in** STD\_LOGIC**;**

floor **:** **out** STD\_LOGIC\_VECTOR **(**3 **downto** 0**));**

**end** **component** MooreElevatorController**;**

-- test signals

**signal** clk**,** reset**,** stop**,** up\_down **:** std\_logic **:=** '0'**;**

**signal** floor **:** std\_logic\_vector**(**3 **downto** 0**)** **:=** **(others** **=>** '0'**);**

**constant** k\_clk\_period **:** time **:=** 20 ns**;**

**begin**

-- PORT MAPS ----------------------------------------

uut\_inst **:** MooreElevatorController **port** **map** **(**

clk **=>** clk**,**

reset **=>** reset**,**

stop **=>** stop**,**

up\_down **=>** up\_down**,**

floor **=>** floor

**);**

-- PROCESSES ----------------------------------------

-- Clock Process ------------------------------------

clk\_process **:** **process**

**begin**

clk **<=** '0'**;**

**wait** **for** k\_clk\_period**/**2**;**

clk **<=** '1'**;**

**wait** **for** k\_clk\_period**/**2**;**

**end** **process** clk\_process**;**

-- Test Plan Process --------------------------------

test\_process **:** **process**

**begin**

-- reset into initial state (floor 1)

reset **<=** '1'**;** **wait** **for** k\_clk\_period**;** reset **<=** '0'**;**

-- active UP signal

up\_down **<=** '1'**;**

-- stay on each floor for 2 cycles and then move up to the next floor

stop **<=** '1'**;** **wait** **for** k\_clk\_period **\*** 2**;** -- wait two cycles

stop **<=** '0'**;** **wait** **for** k\_clk\_period**;**

stop **<=** '1'**;** **wait** **for** k\_clk\_period **\*** 2**;** -- wait two cycles

stop **<=** '0'**;** **wait** **for** k\_clk\_period**;**

stop **<=** '1'**;** **wait** **for** k\_clk\_period **\*** 2**;** -- wait two cycles

stop **<=** '0'**;** **wait** **for** k\_clk\_period**;**

stop **<=** '1'**;** **wait** **for** k\_clk\_period **\*** 2**;** -- wait two cycles

stop **<=** '0'**;** **wait** **for** k\_clk\_period**;**

stop **<=** '1'**;** **wait** **for** k\_clk\_period **\*** 2**;** -- wait two cycles

stop **<=** '0'**;** **wait** **for** k\_clk\_period**;**

stop **<=** '1'**;** **wait** **for** k\_clk\_period **\*** 2**;** -- wait two cycles

stop **<=** '0'**;** **wait** **for** k\_clk\_period**;**

stop **<=** '1'**;** **wait** **for** k\_clk\_period **\*** 2**;** -- wait two cycles

stop **<=** '0'**;** **wait** **for** k\_clk\_period**;**

stop **<=** '1'**;** **wait** **for** k\_clk\_period **\*** 2**;** -- wait two cycles

stop **<=** '0'**;** **wait** **for** k\_clk\_period**;**

stop **<=** '1'**;** **wait** **for** k\_clk\_period **\*** 2**;** -- wait two cycles

stop **<=** '0'**;** **wait** **for** k\_clk\_period**;**

stop **<=** '1'**;** **wait** **for** k\_clk\_period **\*** 2**;** -- wait two cycles

stop **<=** '0'**;** **wait** **for** k\_clk\_period**;**

stop **<=** '1'**;** **wait** **for** k\_clk\_period **\*** 2**;** -- wait two cycles

stop **<=** '0'**;** **wait** **for** k\_clk\_period**;**

stop **<=** '1'**;** **wait** **for** k\_clk\_period **\*** 2**;** -- wait two cycles

stop **<=** '0'**;** **wait** **for** k\_clk\_period**;**

stop **<=** '1'**;** **wait** **for** k\_clk\_period **\*** 2**;** -- wait two cycles

stop **<=** '0'**;** **wait** **for** k\_clk\_period**;**

stop **<=** '1'**;** **wait** **for** k\_clk\_period **\*** 2**;** -- wait two cycles

stop **<=** '0'**;** **wait** **for** k\_clk\_period**;**

**wait** **for** k\_clk\_period **\*** 2**;** -- wait on floor 4 (stop should NOT matter)

-- from top floor, return to first floor without stopping

up\_down **<=** '0'**;**

**wait** **for** k\_clk\_period **\*** 4**;**

-- wait one more clk period just to prove that you will stay at first floor

**wait** **for** k\_clk\_period**;**

**wait;** -- wait forever

**end** **process;**

-----------------------------------------------------

**end** test\_bench**;**