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:

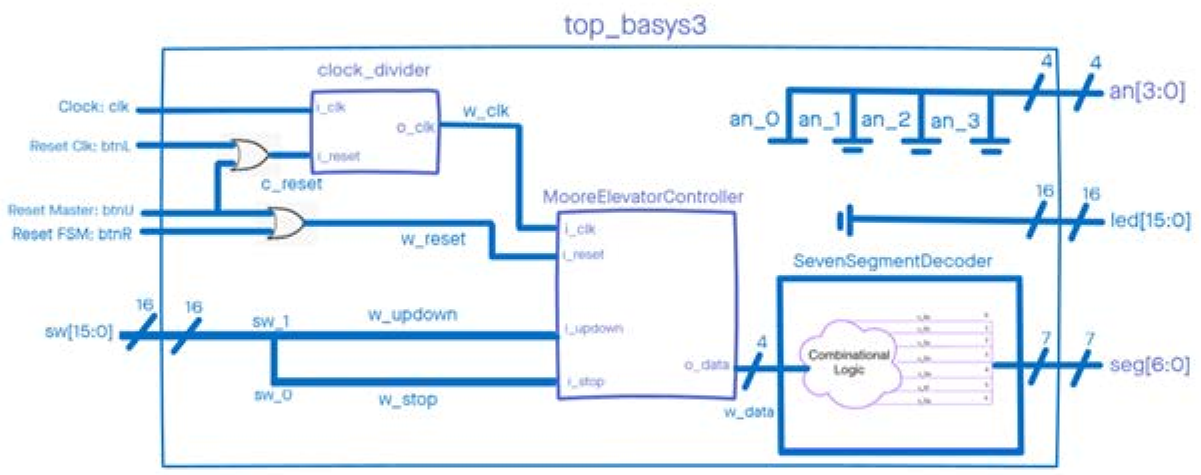

Figure 0 – Top-Level Design Schematic – Minimum Functionality

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.

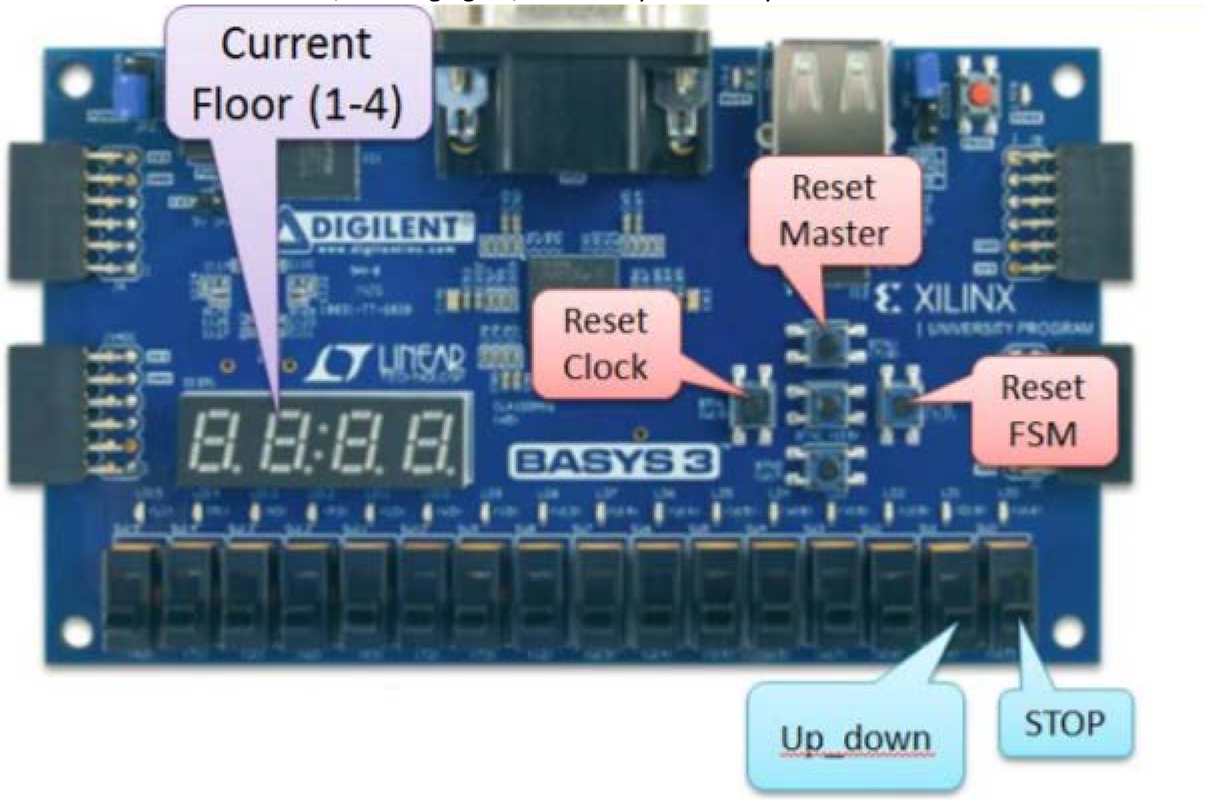

Figure 1 - Hardware Design for Minimum Functionality

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.

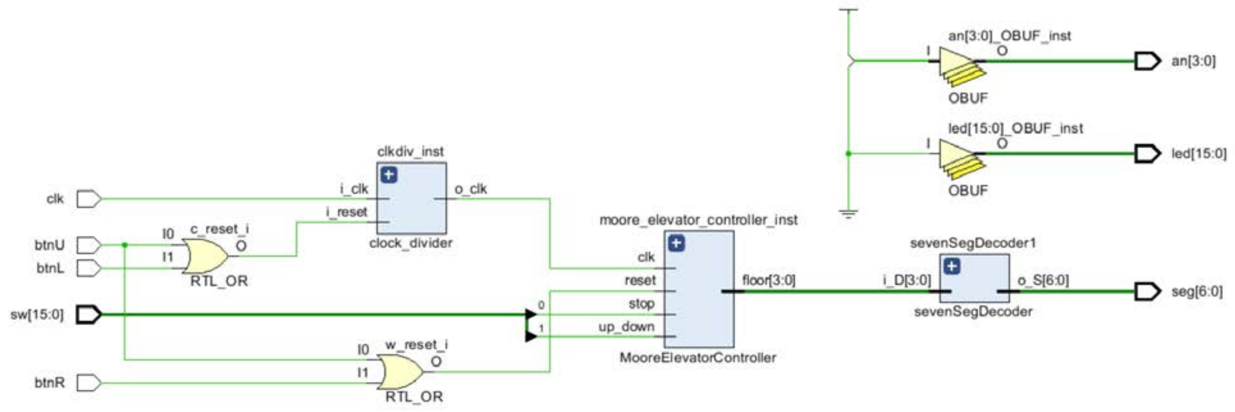

Figure 2 - High Level RTL Schematic with Moore Elevator Controller

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:

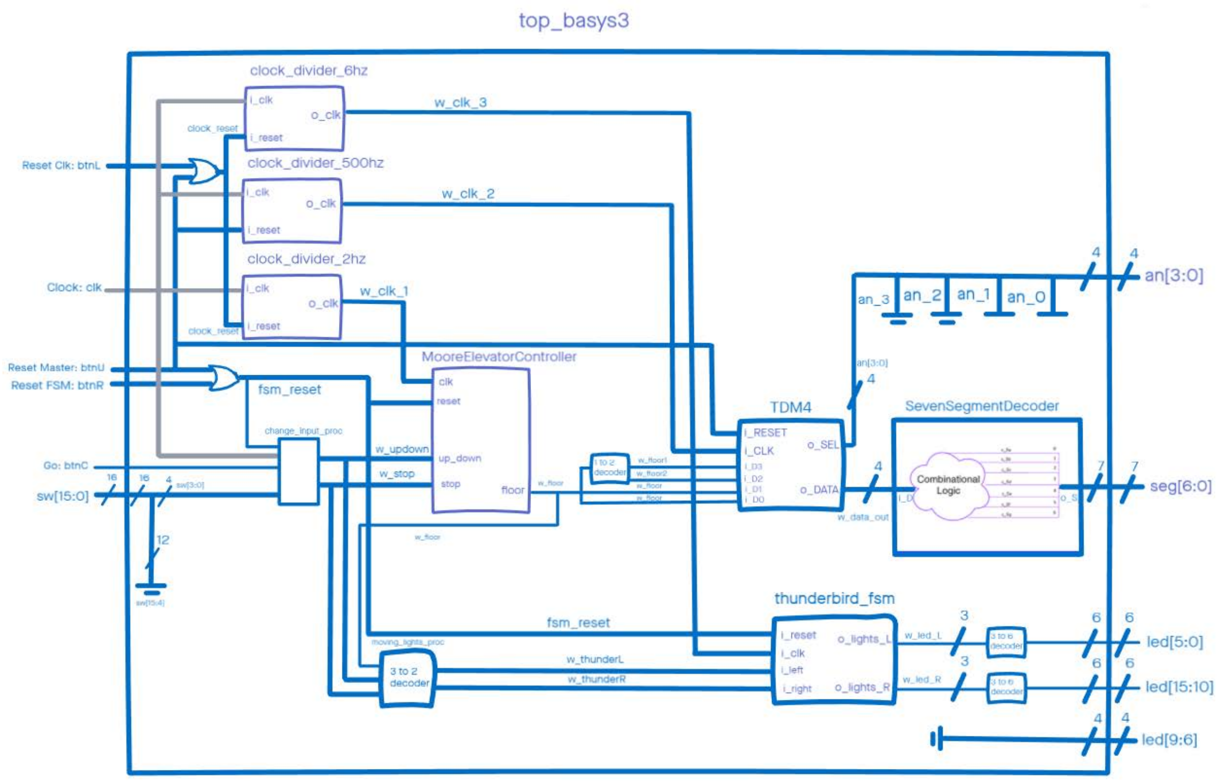

Figure 3 – Top Level Schematic – full functionality

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.

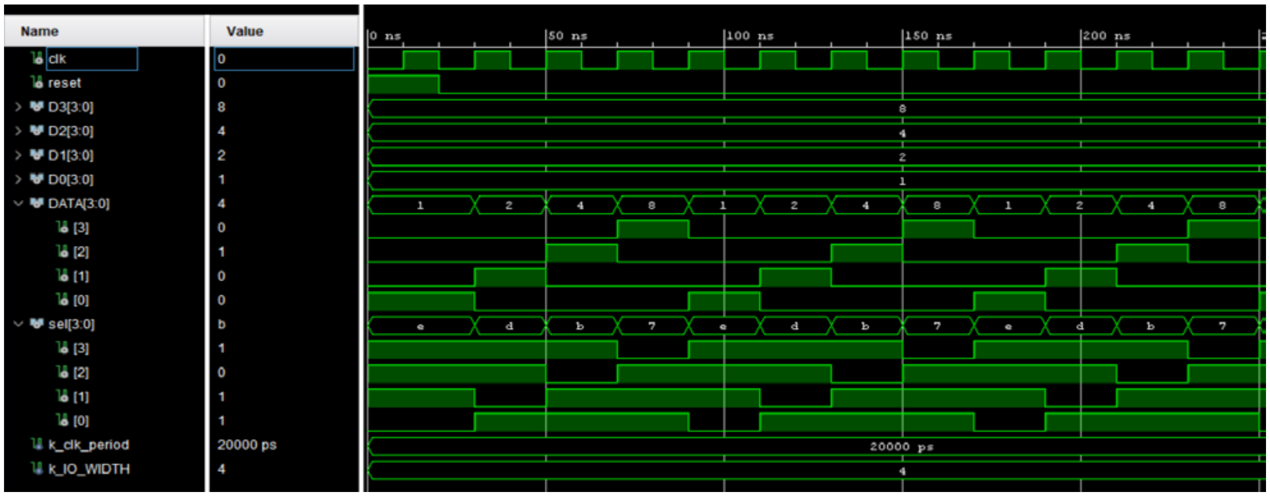

Figure 4 – TDM4 Simulation Waveform

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.

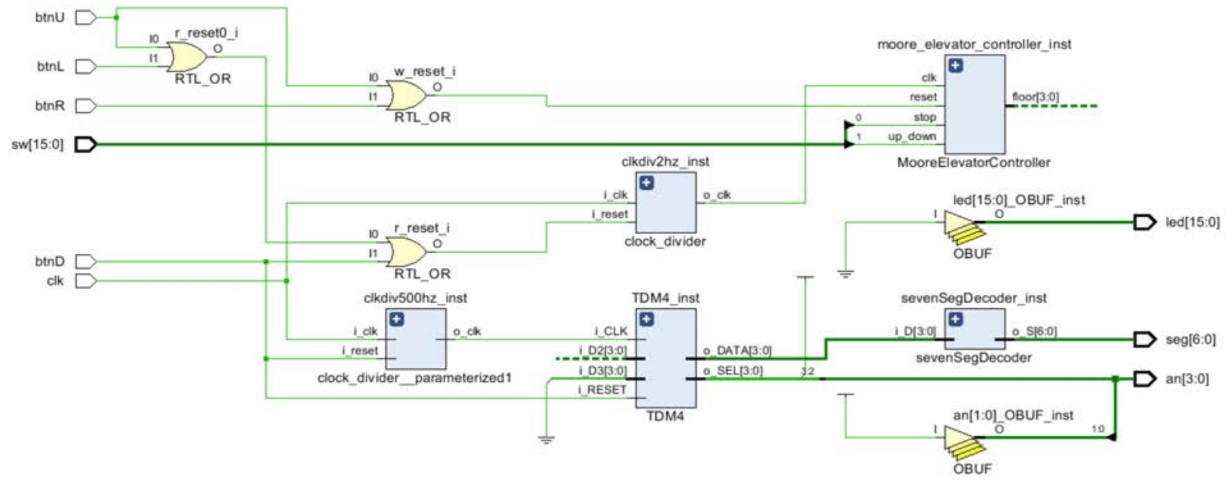

Figure 5 – Top level design with more floors functionality

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.

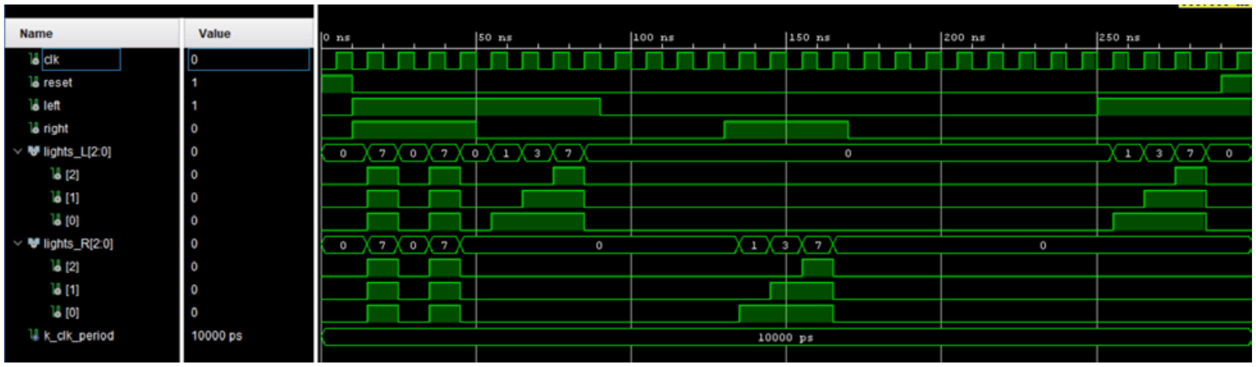

Figure 6 – Thunderbird FSM Simulation Waveform

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:

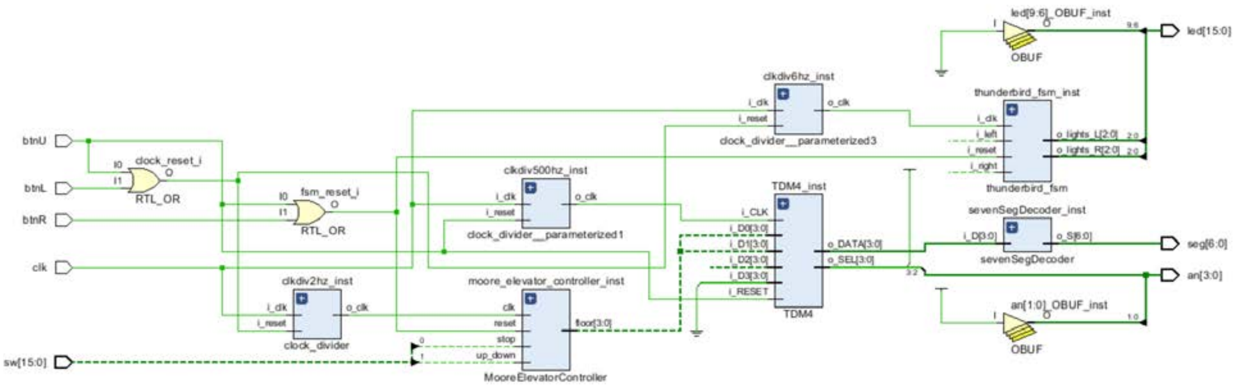

Figure 7 – Top Level Design with moving lights functionality

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:

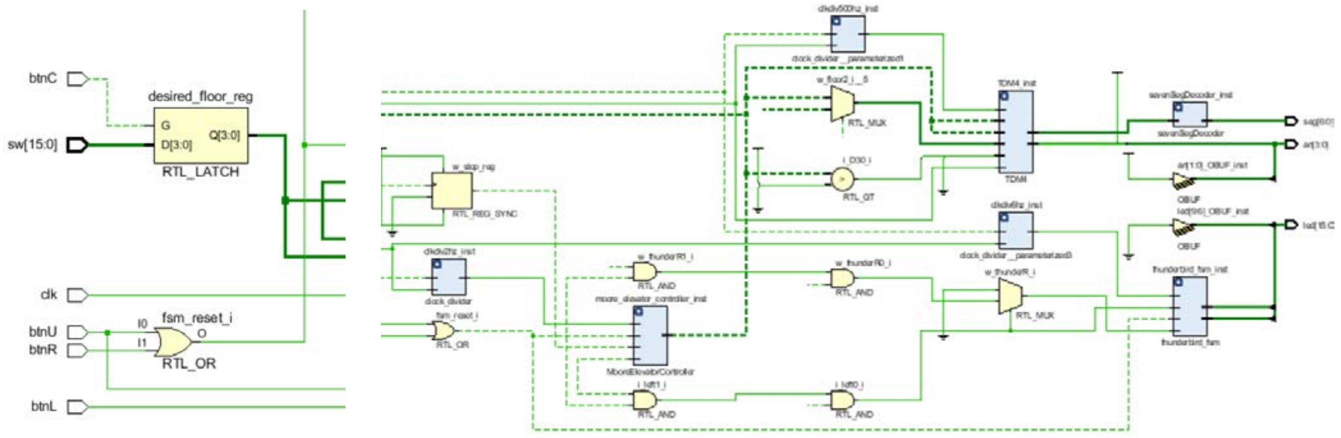

Figure 8 – Final Top Level Design

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

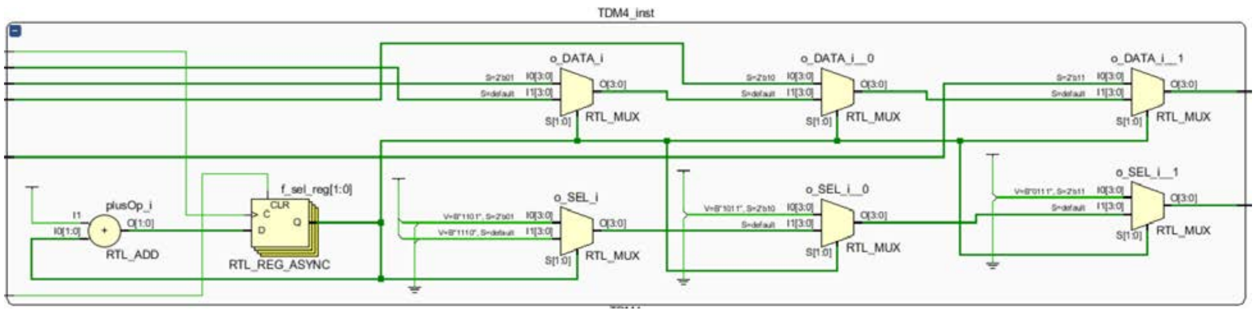

Figure 8.1 – LowLevel TDM4 Design

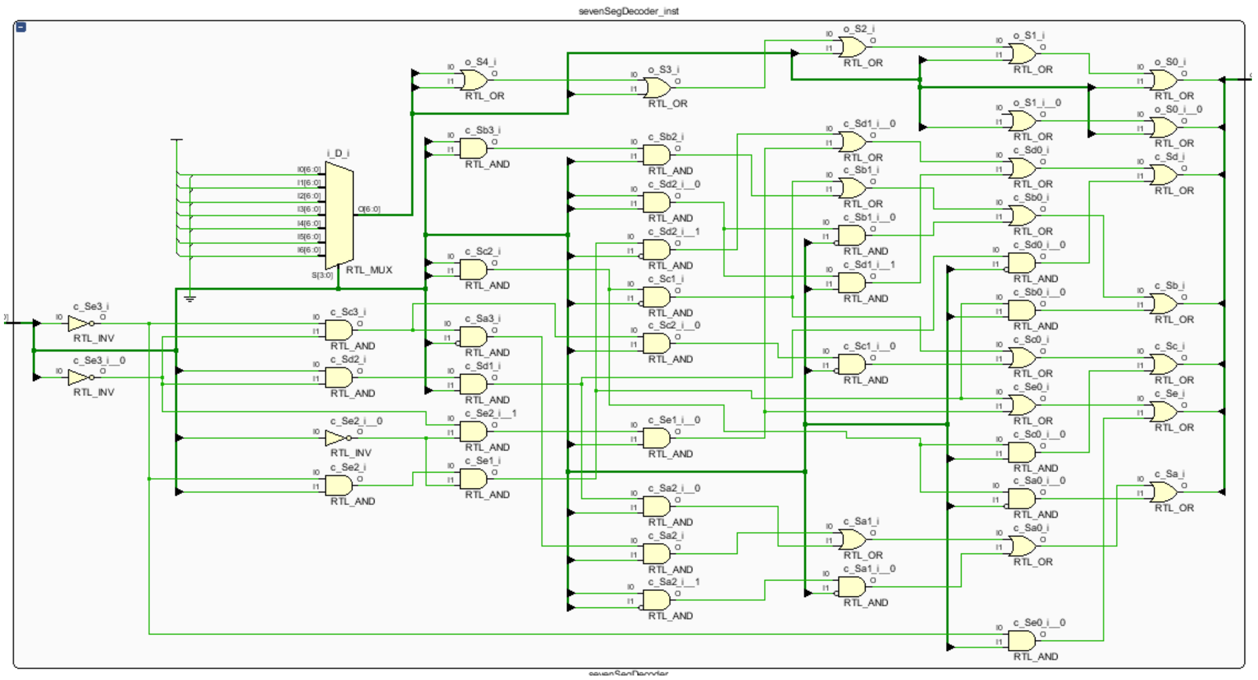

Figure 8.2 – LowLevel sevenSegDecoder Design

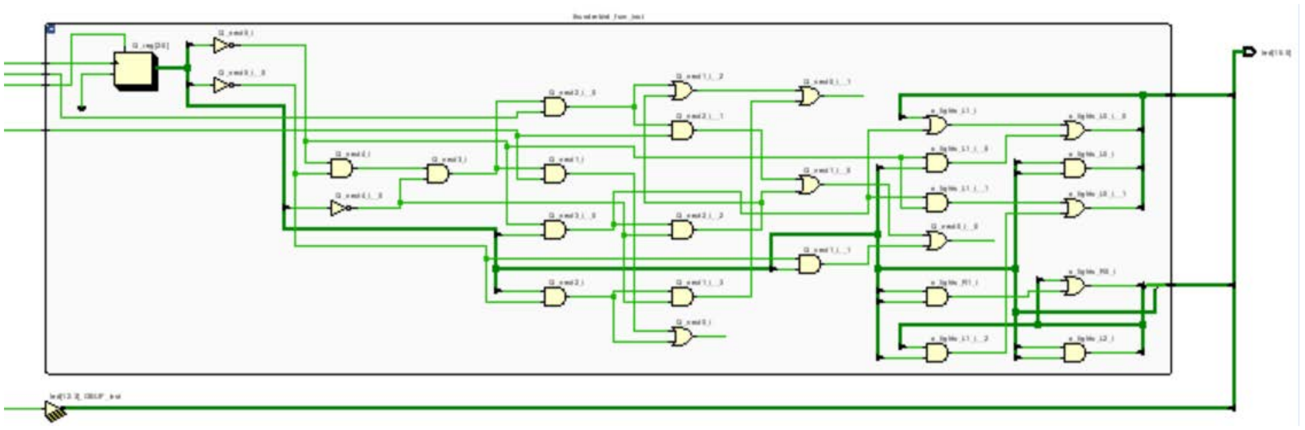

Figure 8.3 – LowLevel Thunderbird FSM Design

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
--|
--+----------------------------------------------------------------------------
--
--|
```

```vhdl
--| 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
--|
--+--------------------------------------------------------------------------
--
```

```vhdl
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;
```