Design Project Overview:

This design project is worth 2 Lab grades due to the quantity of work required. The primary purpose of this lab is to provide you practice in designing more complex systems that integrate sequential and combinatorial building blocks. In this lab, you will test and debug the basic elevator controller built during the in-class exercise by synthesizing and configuring a FPGA with your Moore state machine. You will then incrementally add features to your design that build up to a fully featured elevator controller system. To accomplish this, you will need to integrate modules from previous in class exercises and labs.

Authorized Resources: For this lab, you may work in teams of up to two (including the prelab). You may seek help from any cadet or instructor, not limited to this course, and reference any publication in its completion. Online resources are acceptable. However, no resources containing solutions to the course homework, labs, exams, quizzes, or in class/out of class exercises are allowed. Your work (and your code) must **always** be your own. Normal documentation of all resources utilized is required.

Due Dates: Prelab – 1700 T27 via Gradescope

Minimum Functionality – 1700 T30 in person or via video link

Moving Lights – 1700 T32 in person or via video link

More Floors – 1700 T33 in person or via video link

Change Inputs – 1700 T34 in person or via video link

Passenger Pickup – 1700 T34 in person or via video link

Report – 1700 T35 via Gradescope

Minimum Functionality (1700 T30 in person or via video link)

The initial phase of your design will be to implement MINIMUM functionality. The minimum functionality is what is required to consider the lab accomplished for passing the course. However, the minimum functionality will only get you a 60% of the total points assuming you receive all of the points for the final deliverables (report, code, etc). Figure 1 below shows the BASYS3 board interface that you will use for MINIMUM functionality. As can be seen in the figure, your elevator's floor should be shown on seven-segment display 2 (Display 0 is the far right display). This should be updated according to the output of your basic elevator controller at a clock rate of **2 Hz**. Your elevator must not teleport! Three push buttons are to be used for resets as shown below. The **Master Reset** button should reset the **FSM** AND the **clock**.

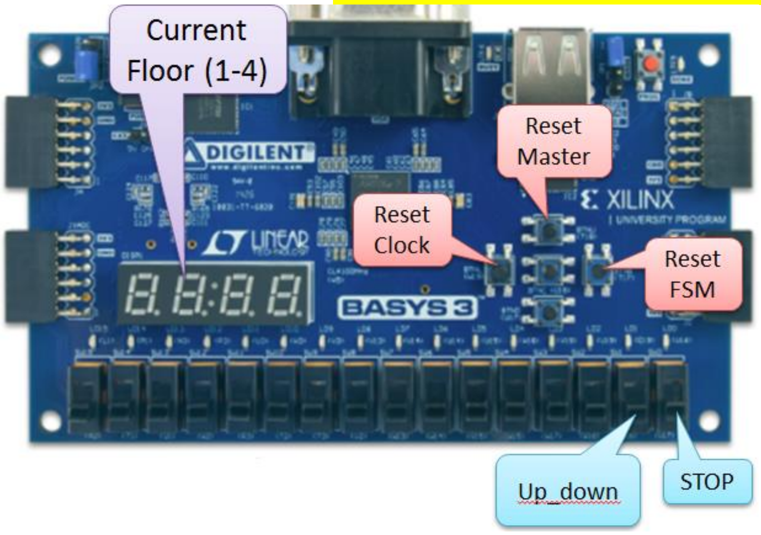

Figure 1. MINIMUM functionality interface

Additional Functionalities for Desired End State

To receive full credit for this lab you must implement additional functionalities. These additional functionalities include:

1) Moving lights to indicate elevator travel direction
2) More floors (up to 15)
3) Change inputs (Selecting desired floor by number using 4 switches)
4) Passenger Pickup

Figure 2 below shows the BASYS3 board interface that you will use for these additional functionalities. Note, that you do not need to use all of the hardware interfaces unless you implement all functionalities. Notice that now there are 15 floors that you must display using two seven-segment displays. Additionally, four switches are used to indicate the floor a passenger is on, and another four are used to indicate what floor they would like an elevator to transport them to. Upon pushing the GO button, the *nearest* elevator will travel to them and subsequently take them to the desired floor. Finally, a sweeping pattern of lights (similar to Thunderbird tail lights) indicate which direction the selected elevator is moving.

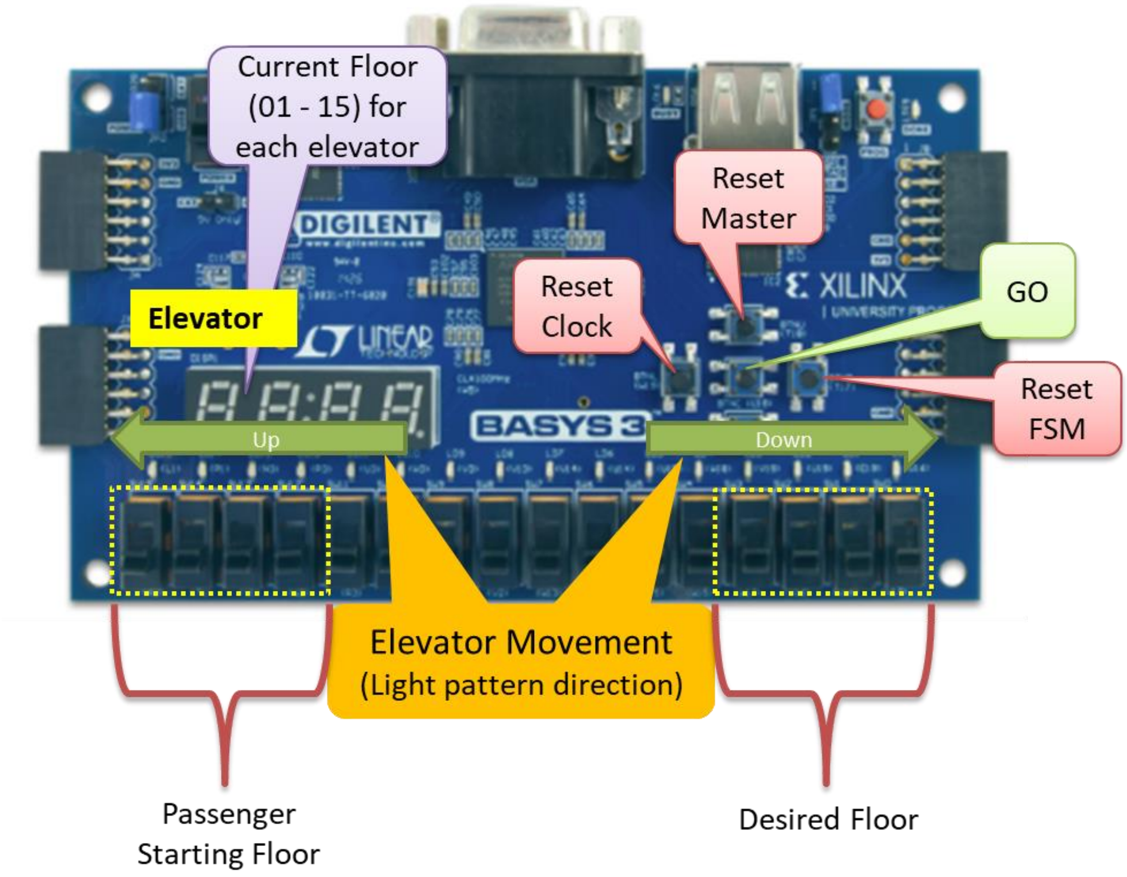

Figure 2. Additional functionality interface (including additional functionality)

Prelab tasks (1700 T27 via Gradescope)

1) Provide a top-level schematic showing the architecture of how you will implement MINIMUM functionality (draw.io, PowerPoint, etc – do not hand draw). The expectation is you will update this schematic at the end of this lab to reflect your final design.

   a. Note, you will need to include several components in this schematic. Please use the top-level entity interface found in the **top_basys3.vhd** file included with the Lab 4 files (notice there are signals you will not use for minimum functionality, leave them in the file as they are included for the additional functionality, and include them in your prelab schematic). You need to include a clock divider module, and the remaining modules you need will come from your previous in-class exercises and labs (MooreElevatorController and sevenSegDecoder).

   b. Label all wires not connected to the top-level entity ports. They will be additional signals you will create in your VHDL code.

   c. Only turn on seven-segment display 2 (display 0 is the far right). Force the others OFF by disabling their anodes.

   d. Turn OFF (ground) all LEDs, since you do not need them for MINIMUM functionality, but they are in the entity for later functionality.

   e. Just like in the last lab, the clock divider uses *generics* so that you can redefine characteristics at instantiation. For instance, you can redefine the number of times the clock divider divides a clock for each instance of the module. Be sure to specify the value for your clock divider to obtain 2 Hz.

   f. All component names should match the labels given to them (ex: MooreElevatorController should be labeled MooreElevatorController)

   g. All components should have their internal ports labeled (So the reader knows what signals are being connected)

Lab Tasks

Part 1: Minimum Functionality

1) Create a Vivado project called **Lab4**.

2) Create a **code** folder and add copies of all of the code files you will need for your design
   a. Do not forget to add the files to your project

3) Using your Prelab drawing, complete the top-level module
   a. Remember this basically involves:
      i. Copying in component definitions
      ii. Creating additional wires
      iii. Instantiating your component instances and connecting all wires

   b. For the clock divider, what value for `k_DIV` do you need to produce a 2 Hz clock?
      i. If desired, you can test it with the provided testbench.

4) Create your XDC file from the XDC template, and uncomment of the hardware support you will need per your top_basys3.

5) When you synthesize and later implement your design you may receive several warnings about unconnected wires or ports. Ignore these.

6) Test that your design works as expected.

   a. Once you have verified all functionality, demo your MINIMUM functionality to an instructor.
   b. While not required, feel free to create a test bench for your top_basys3 if it doesn't appear to be working correctly. You will need to create an artificial clock (which you have done to test FSMs before)
   c. **Does your master reset work as intended? Think about how your basic elevator controller implements a reset. If you had to fix something, explain what you did in your report.**

7) If you have not already done so, create a **bitFiles** folder inside the same folder where your **code** folder resides. The bitFiles folder will store all of your .bit files.

8) Change the name of your .bit file from top_basys3.bit to **LAB4_MINIMUM_functionality.bit** and move it to your **bitFiles** folder.
   a. This will ensure your bit file doesn't get overwritten!

9) Commit and push your work!

10) Demo your functioning design to an instructor.

Part 2: Additional Functionality

Before you start any of these, it is highly recommended that you **create a well thought-out plan**. This should include as necessary word descriptions of functionality or algorithms, equations, truth tables, hardware schematics, and FSM state transition diagrams. Having a good plan will reduce the problems you will run into. Finally, do not forget to commit at least after each feature milestone is achieved. I have had students mess up their code on the last piece of functionality, and because they didn't commit earlier, were forced to restart because they could not figure out what they messed up.

1. Moving lights to indicate elevator direction (1700 T32 in person or via video link)

As your elevator moves up or down, 6 LEDs should create a sweeping pattern as indicated in Figure 2. With each clock cycle, a pair of adjacent LEDs will light up in a similar pattern as the Lab 2 Thunderbird lights (e.g., led[1:0], then led[3:0], then led[5:0], then off). **You do not need to modify your Thunderbird FSM**!! When you are stopped, your LEDs should be off. Make sure that all 6 LEDs come on at least once per floor change, but do not let them change so fast that you cannot tell the direction the elevator is moving. **Hint:** Think about how fast your elevator controller cycles and base the rate off of that.

2. More floors (1700 T33 in person or via video link)

Redesign the elevator controller to handle 15 floors, and display the appropriate floor as the elevator moves. Displaying floors 10-15 in hexadecimal is fine for partial credit (10 pts). For full credit (20 pts), you must display the floor you are on using a two digit **decimal** number (no hex!). Use the left **two** seven-segment displays to accomplish this. For instance, if you are on floor one, display "01" (some people get unsettled by floor "00"). If you are on floor 15, display "15".

Creating a testbench is recommended but not required. However, you must **include in your REPORT a drawing of the hardware required** to split the 4-bit floor number into two 4-bit binary coded decimal numbers (one for each seven-segment display). Note, in order to actually use multiple seven-segment displays you must make use of a time division multiplexer (TDM) module.

Understanding the TDM and why you need it

You have multiple anodes to select which seven segment display you want to use, but you only have one set of cathodes (there is ony one seg signal on the board controlling 4 displays). How do you display different things on your four seven segment displays then? The answer is that you must change which display you use at a regular rate. This rate must be fast enough that it looks as if they are all on at the same time. This is where a TDM comes in. It will output each of the input data lines an equal amount of time.

The 4 to 1 TDM included with this lab will allow you to use up to four displays. If you only need two displays, then you can give each of the displays two "turns" from the TDM. The clock frequency for the TDM needs to be at least 4 times as fast as the incoming data changes, and it needs to be fast enough that the human eye will not notice the displays flickering. However, the clock frequency does not need to be excessively fast. Try something around **500 Hz** and adjust if needed. The displays will likely look brighter when using higher rates. **Warning:** If you drive the TDM too high, it will not work.

You can adjust the bit width of the TDM using the `k_WIDTH` generic. This allows you to use a seven-segment decoder(s) on either side (input or output) of the TDM by specifying either 4 or 7 bits. Be sure you read the code and understand what it is doing. You already tested the TDM in ICE5 and should be familiar with how it functions.

As can be seen in the figure below, the primary parts of the TDM are two 4:1 multiplexers and a 2-bit counter. The counter cycles through four values (one per clock cycle) which then feed into the select lines of the multiplexers. Based on the line selected, the DATA multiplexer outputs the respective data to the output, while the SEL multiplexer outputs a four bit one-cold value indicating which data line was selected. The DATA output simply needs to be connected to a seven-segment decoder or display, while the SEL output should be connected to the active-low display anodes. Remember that if you only want to use a subset of the inputs you can provide the same input to two of the data inputs.

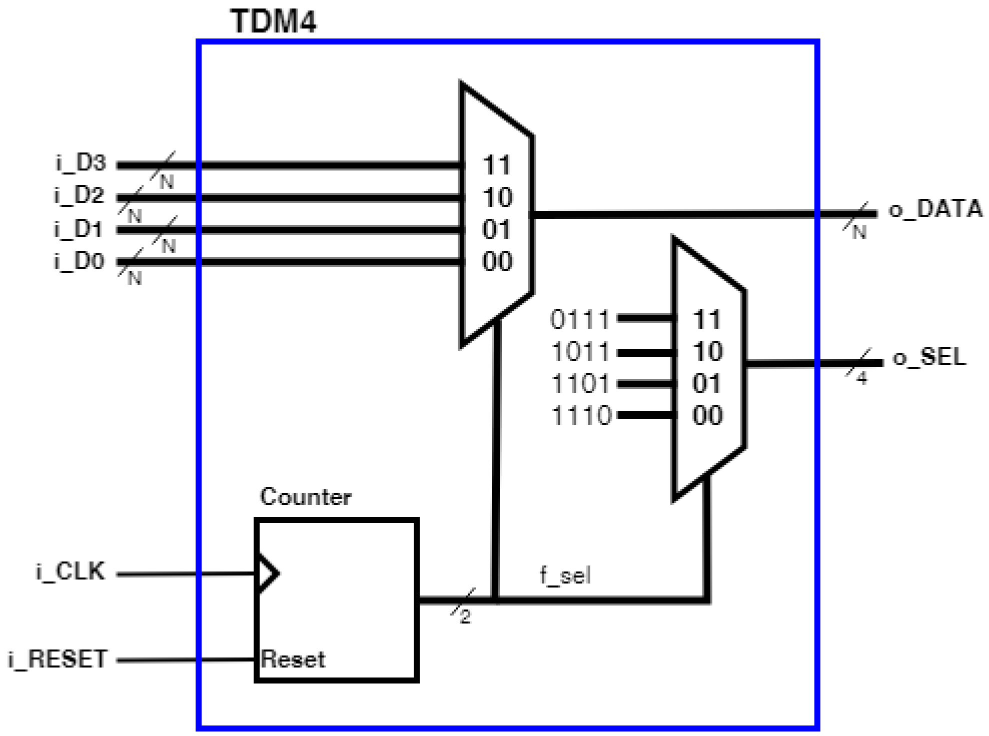

Figure 3. Time division multiplexer schematic

Finally, the reset input for both the TDM *and* the clock divider that generates its clock should be directly connected to the top-level master reset signal. You still want to be able to view data on multiple displays when pausing the system, so the TDM should not be affected directly by the FSM or clock reset signals.

3. Change Inputs (Selecting Desired Floor by number) (1700 T34 in person or via video link)

Design a FSM to use a 4-bit binary destination (desired) floor as an input instead of up_down and stop. The elevator will not start moving to the desired floor until after a GO signal is asserted (via a button push). Upon pressing GO, store the value of the desired floor. The elevator will gradually move to the stored floor. Do not change the stored floor until the next press of GO. The elevator must only move on the rising edge of a clock. Display the floor you are on as you move to the floor denoted by the binary input. The elevator must be stable upon a reset. If a reset is pressed, the elevator should go back to floor 1 and it should not try to go to the stored floor.

- **HINT 1:** Instead of redesigning the primary elevator controller FSM, you can simply design circuitry that generates the up_down and stop signals based on the stored floor (SF) and the current floor (CF). **If** you choose to create a FSM to do this it will need to react *quickly* to changes in the CF, so consider what type of FSM would be required. You may find that combinatorial logic will suffice instead of using a FSM.

- **HINT 2**: You can save the desired floor as the stored floor simply with a register that uses GO as a clock. To do this you will need to add the following line to your .xdc file assuming you are using btnC for GO
    - `set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets {btnC_IBUF}]`

Include in your REPORT a sketch of any new hardware modules (or state transition diagrams) and explain how they work as needed for understanding. Again, a testbench is useful, but not required.

4. Passenger Pick Up (1700 T34 in person or via video link)

It is highly recommended that you do not attempt this unless you have completed the previous functionalities successfully. The basic idea is that you will enhance your design to pick a passenger up and take them to their desired floor.

Refer back to Figure 2. You have two four-bit inputs: one specifies the floor the request was made from, and the other specifies the desired floor to travel to. After the GO button is pushed, the elevator should then react to the request, first going to the floor to pick up the passenger, pausing one clock cycle, and then taking the passenger to the desired floor. Remember, elevators cannot teleport! The levator's current floor should be displayed on a seven-segment display per Figure 2. Your elevator shall move between all available floors as necessary. For example, the elevator is on floor 2. A request is made from floor 1 to move to floor 7. Elevator 1 moves (one floor each clock cycle) first to floor 1 to pick up the passenger, pauses one clock cycle, and then moves to floor 7 to deliver the passenger.

Lab 4 Deliverables

An overview of the point distribution for this lab is shown in Table 1 below. This is supposed to be a fun, hands-on lab where you get to integrate a lot of the techniques you have learned thus far. To that end, half of the points are for functionality. You will receive points for as many functionalities as you can complete. However, you must complete at least minimum functionality. Additional notes on expectations for selected items listed Table 1 are discussed next.

Table 1: Overview of Lab 4 Point Distribution

Item	Points	Out of
Prelab/Schematic		20
Functionality – Minimum		50
Functionality – More floors		30
Functionality – Moving Lights		30
Functionality – Change Inputs		20
Functionality – Pick up passenger		10
VDHL files, XDC and BIT files		10
Report		30
Total		**200**

(20 pts) Prelab/Schematic

- Top level schematic
  - Reset Clock, FSM, and Master all connected correctly
  - Floor output sent to seven seg display module
  - Up_down switch connected correctly
  - Stop switch connected correctly
  - Seven Seg Display Decoder included and wired correctly
  - Basic Elevator Controller FSM included and wired correctly
- Label all wires and ports
- Only turn on Seven Segment Display 2
- Turn off all LEDs

(50 pts) Minimum Functionality

- Only hex display 2 shows
- The elevator should change floors at a rate of 2 Hz
- Elevator counts from 1 to 4 when up
- Elevator counts from 4 to 1 when down
- Elevator will not travel when stop is switched
- Reset Clock
- Reset FSM
- Reset Master should reset the elevator controller FSM in addition to the clocks

(30 pts) More Floors

- Floor counts from 1-15 utilizing two displays
- Not all displays are shown when clock is paused. Per the directions, the reset input for both the TDM *and* the clock divider that generates its clock should be directly connected to the top-level master reset signal.
- Half credit if using only one display and showing floor number in hexadecimal

(30 pts) Moving Lights

- The lights should stop when you reach the bottom or top floor
- The lights should create the desired pattern
- The lights change too slowly. All six LEDs must light up per every floor change.
- The clock reset should pause and not reset the lights

(20 pts) Changing Inputs

- The design does should correctly handle a desired floor input of “0000”
- The elevator should not move until GO is pressed
- Elevator successful travels to the desired floor

(10 pts) Passenger Pickup

- The design does should correctly handle a desired floor input of “0000”
- Elevator travels from current floor to passenger floor
- Elevator travels from passenger floor to desired floor

(10 pts) VHDL files

- Provide any VHDL files you modified or created in **code** folder
    - top_basys3
    - MooreElevatorController
    - sevenSegDecoder
    - thunderbird_fsm
    - TDM4
    - Module for splitting display into two parts
    - Module for changing inputs
    - Module for Passenger pick up FSM
- **ALL MEMBERS** of the group will submit **YOUR OWN** code. This does not mean you can't collaborate, but make sure you have done the code yourself to learn the material.
- Constraints (XDC) file
- REQUIRED functionality bit file
- FINAL functionality bit file
- It is recommended that you create a bit file for each demo you make in case adding functionality messes up a previous demo (could be 6 bit files)
- Use generics instead of modifying or creating multiple module files for the clock divider
- Aside from deductions due to missing files, the code files will be graded according to the following rubric

Rating	Points	Quality
Excellent	9-10	All required files are included in the "Code" folder of the repository. Each file includes an updated header with accurate filename, author, creation date, description, and documentation. Each entity has a name that matches the file name. Entity inputs and outputs match provided/designed schematics. Variables have descriptive names that meet the naming conventions. Code is commented throughout providing concise descriptions of what is occurring (not every line needs to be commented, but generally blocks of code should be). There are no extraneous comments or code. Spacing/indentation is clear and organized.
Good	7-9	Each file includes an updated header with author. Each entity has a name, inputs, and outputs that match provided/designed schematics. Variables meet the naming conventions. There are no extraneous comments or code.
Acceptable	3-6	Each entity has a name that matches the provided/designed schematics. Variables meet the naming conventions. Portions of headers were updated, some code was cleaned up.
Unacceptable	0-3	Files were not included. Header was not updated. Extraneous comments exist. Variables do not meet naming conventions. No helpful comments.

(30 pts) Report (1700 T35 via Gradescope)

Utilize the lab 2 template to provide **ONLY** the following sections for the Lab 4 report:

- Front matter
    - Title, Authors, and Documentation Statement.
- Design
    - Schematics for all subcomponents used (you can grab the RTL schematic from your previous labs/exercises), but drawings/FSMs are expected for any new components you generate. Schematics should not be dumped in the report en masse. They should be part of your discussion of the development of your design. Examples include:
        - Component for blinking lights (thunderbird, RTL okay, include FSM)
        - Component for controlling floor (Basic Elevator Controller, RTL okay, include FSM)
        - Component for interfacing with the seven segment display (SevenSegDecoder, include schematic from Lab)
        - Component to split a hex number into two digits
        - Component to store floor destination
        - Component to store passenger location
        - Component to control elevator based on passenger floor/destination
- Results
    - Final top-level design (computer generated with Visio, draw.io, powerpoint, etc) (10 points)
        - Include all of your final functionality
        - This final top-level schematic should be neatly computer-generated by you (not just using the RTL Schematic).
            - Powerpoint, visio, and draw.io are all good tools to use. Hand drawn schematics will be by exception only (show it to your instructor first)