Final Elevator Project

Instructor: Soura Anabtawi

SMRTTECH 3DE3

Digital Electronics

Submitted by:

Bradley Tennant (400407276),

Ricky Solis (400460236)

Lab Section: L04

Project Description 3
Summary of Features 4
Core Hardware Features 4
Additional Implemented Features 4
Flow Chart with decision blocks 5
Schematic 6
Resources Used 7
Example Testbench Simulation 7
Pin Planner 9
RTL View 10
Pictures of Final Setup 11
Project Limitations 12
Design Selection 13
Project Achievements 14

Project Description

The goal of this project was to design, code, and wire a three story elevator system. We used a Verilog HDL file in Quartus II, digital logic circuits, and an FPGA. The elevator operates between Floors 1, 2, and 3, and responds to user inputs through three push buttons. Each push button assigned to a specific floor. A DC motor drives the elevator using a string and pulley system. All, while reed switches at each level act as position sensors that detect when the elevator has reached a floor.

The system interface includes a 7-segment display showing the current floor number and two LEDs indicating the direction of movement. The controller transitions between states based on button inputs, sensor feedback, and motor direction signals. Building the system required developing the control logic, wiring the external hardware (FPGA), and implementing the elevator behaviour in Verilog. Additional features such as blinking the display while the elevator is in motion were included to enhance usability while staying within the FPGA's I/O limitations.

Overall, the project focused on integrating combinational and sequential logic. Incorporating sensor inputs, output control signals, and a state based approach to create a functional miniature elevator that behaves clearly and predictably.

Summary of Features

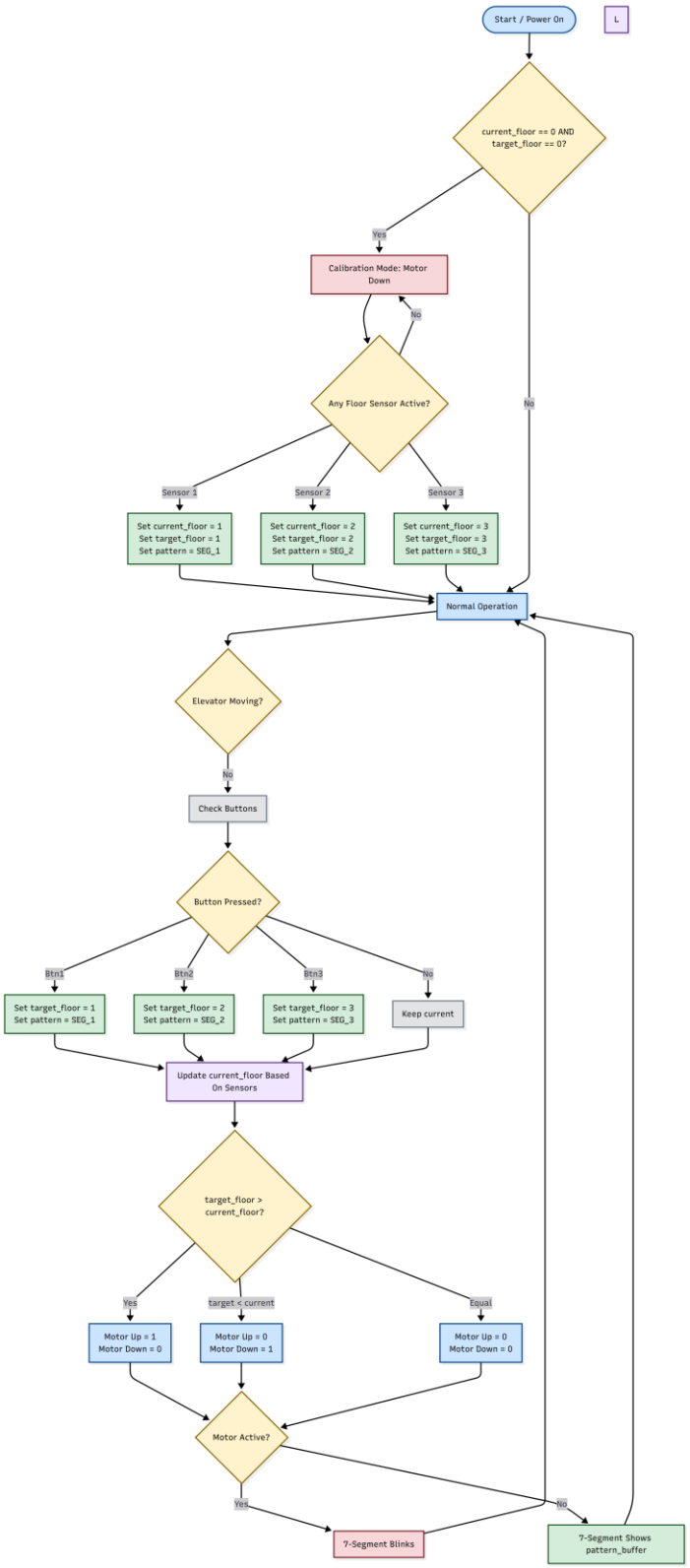
Core Hardware Features

Hardware	Trigger	Feature description
Push Buttons (3)	When a floor button is pressed.	Each button sends a request for Floor 1, 2, or 3. The elevator accepts a new request only when it is idle, preventing mid movement changes.
7 Segment Display	Floor sensors update the active floor.	Shows the current floor number. When the cabin is moving, the display briefly turns off (blink effect) using the prescaled clock.
Reed Switches	When the cabin aligns with a floor sensor.	Provide position feedback to update current_floor, stop the motor at the correct level, and complete the calibration process.

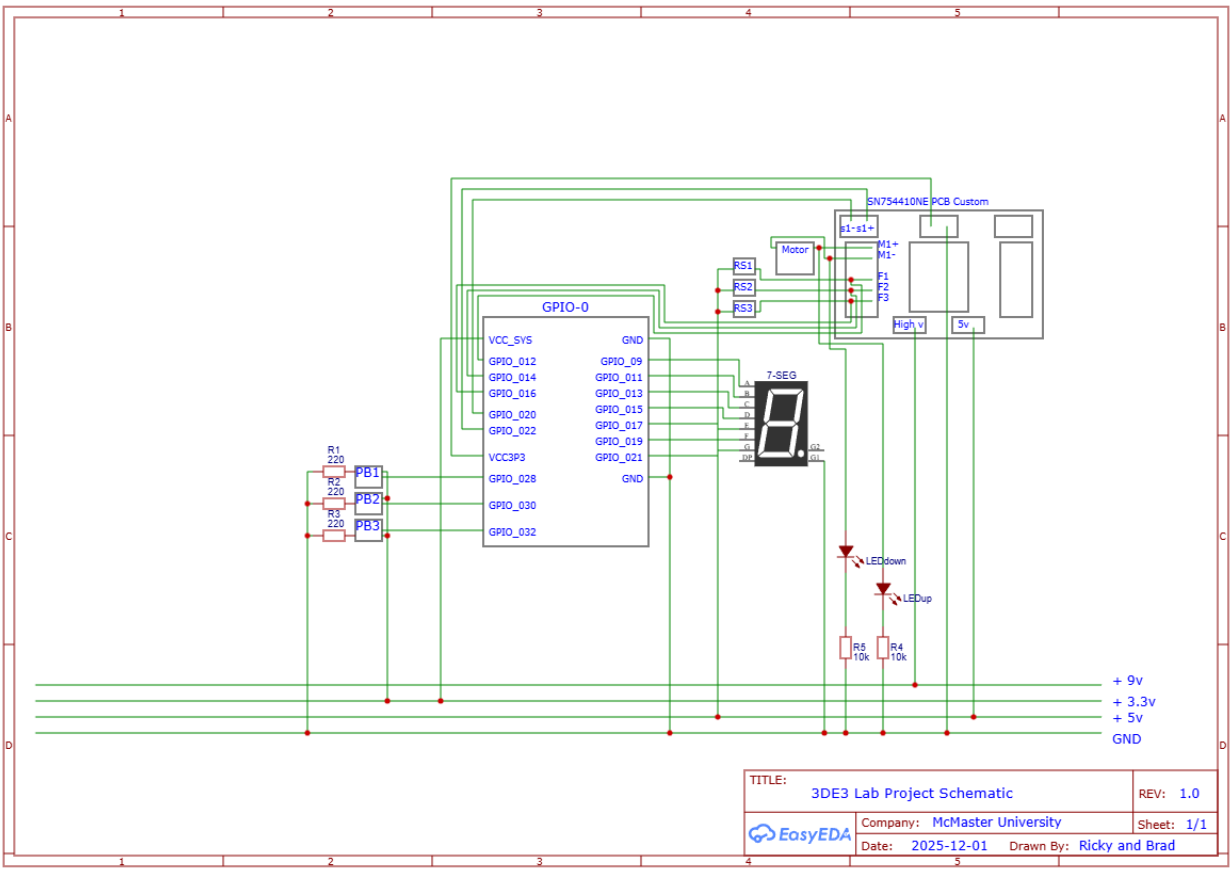
Additional Implemented Features

Hardware	Trigger	Feature description
LED Pair (Up/Down) **(Direction Indicators)**	Elevator begins travelling up or down.	One LED shows upward motion and the other shows downward motion, giving simple visual feedback on motor direction.
Push Buttons (3) **(Motion Lock)**	The button is pressed while the elevator is moving.	Button presses are ignored during movement to prevent conflicting requests. The system only accepts a new input after reaching its target floor.
Push Buttons + Reed Switches **(Auto-Calibration)**	System powers on or enters reset state (current_floor = 0).	The elevator automatically moves downward until a floor sensor is activated. Once aligned, the system locks onto the correct starting floor.
7-Segment Display **(Travel Blink)**	The elevator starts travelling to a selected floor.	Instead of showing a constant number while moving, the display blinks using the divided clock signal until the cabin reaches the target.

Flow Chart with decision blocks



Schematic

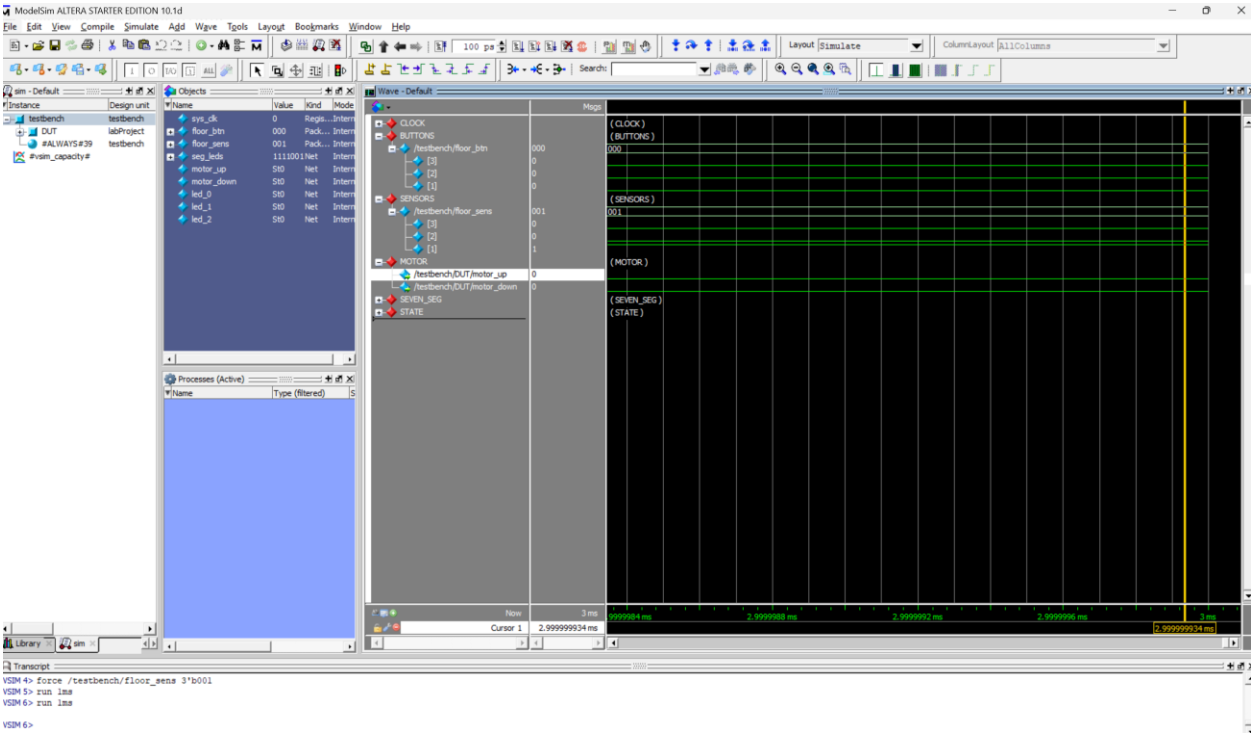


This schematic shows how the elevator system's hardware connects to the FPGA and PCB. Three push buttons (PB1–PB3) act as floor request inputs, each wired into a dedicated GPIO pin. The reed switches (RS1–RS3) provide floor position feedback by triggering when the elevator reaches each level. Two indicator LEDs are driven by separate GPIO pins to show the current direction of travel. A motor driver is connected to the FPGA's motor_up and motor_down outputs to control the DC motor's direction. The 7-segment display gets its pattern directly from the FPGA through seven GPIO pins that output the correct segment pattern for each floor. All components share common power and ground rails, allowing the FPGA to read inputs and drive outputs reliably. This schematic shows how each sensor, button, LED, and output device connect back to the controller to form the elevator control system.

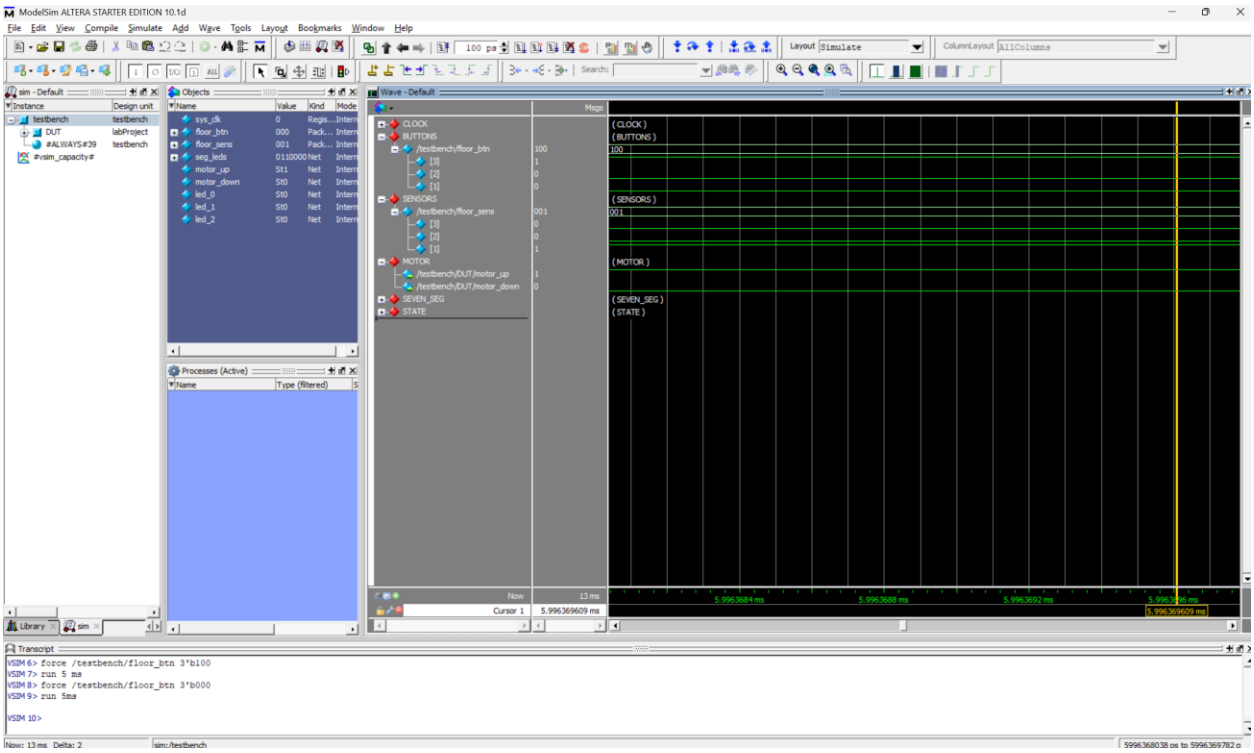
Resources Used

In order to troubleshoot and progress through this project from home, we used ModelSim-Altera to make an interactive simulation of our elevator system. This simulation displayed the waveforms of each input and output individually, to help us visualize how our variables were interacting with each other. In order to simulate each input, we used a “force” command on each variable to trigger a positive signal. We also used a “run” command to progress the simulation through time so that our inputs would register with the clock posedge and update the rest of the system periodically. Using this simulation, we verified that our code was working correctly through testing each push button arriving at the correct floor and verifying that all other possibilities were acceptable.

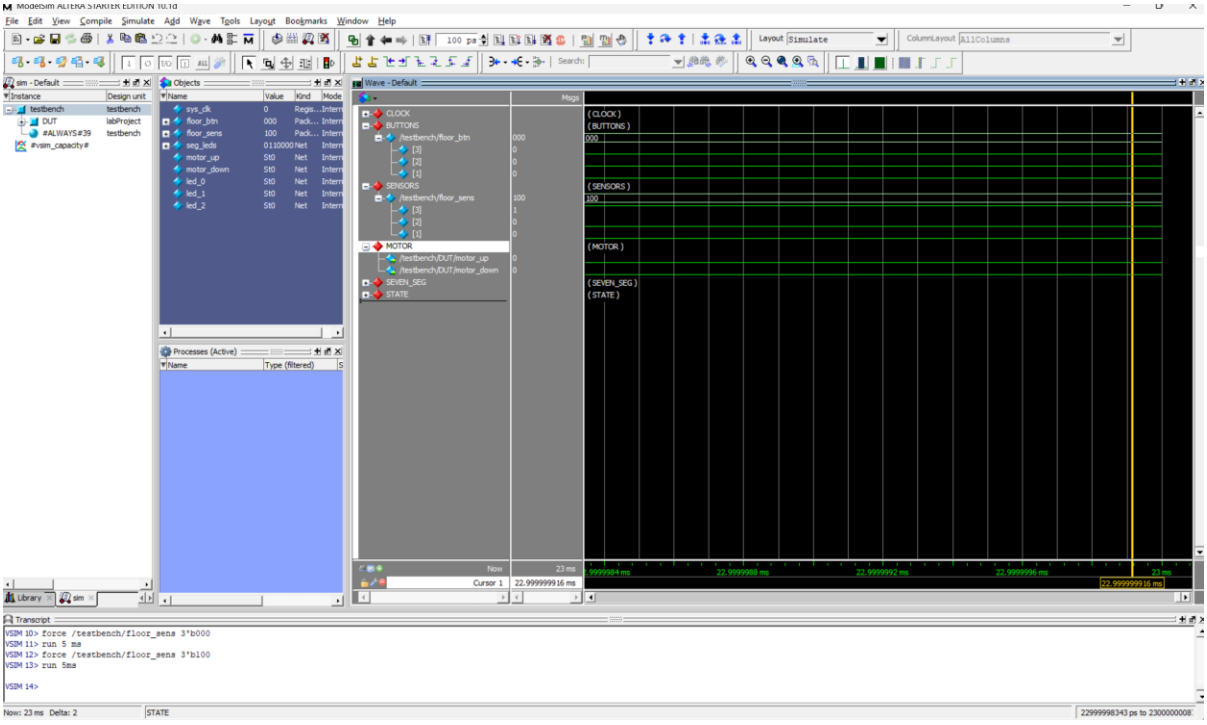
Example Testbench Simulation



Here, we are currently positioned on floor 1, the program has just entered our normal operation mode after calibrating. No buttons pressed. Motor up and down are 0 because current floor is equal to target floor. The seven-segment display shows a solid 1.

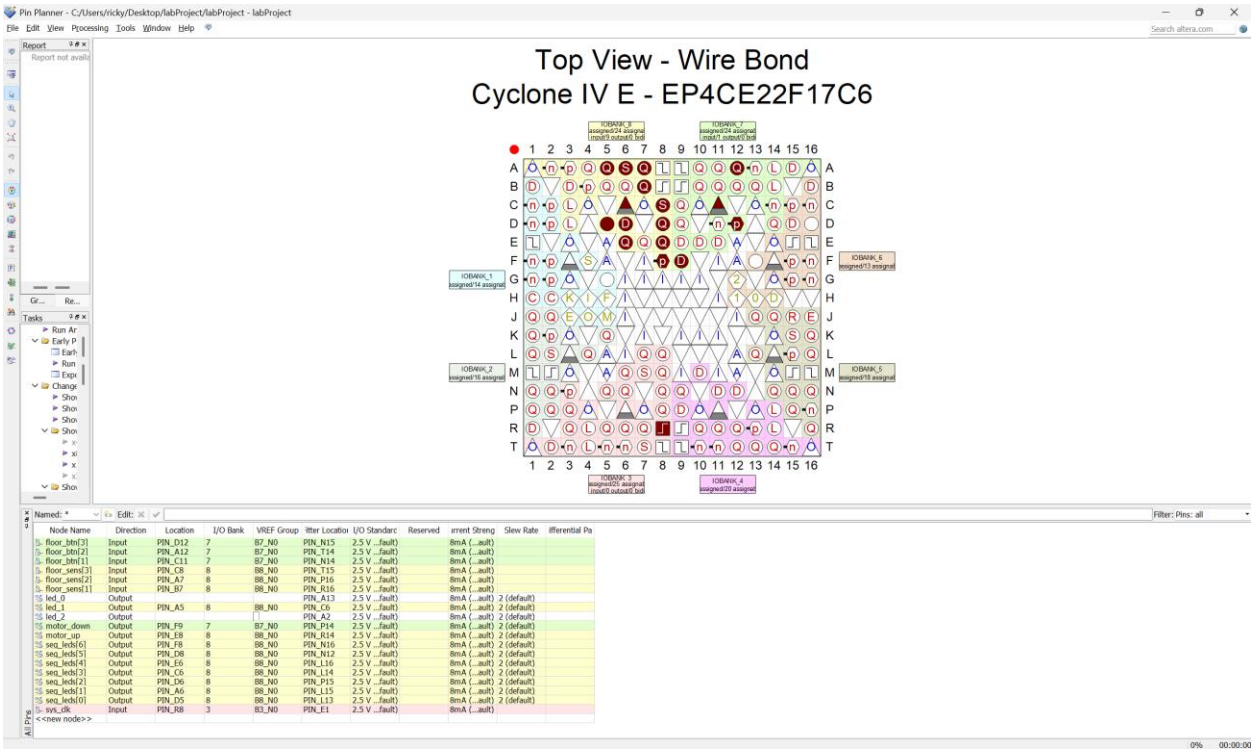


Now Floor Button 3 has been pressed and released, activating motor up. The seven-segment display now blinks 3, until that floor 3 is reached.



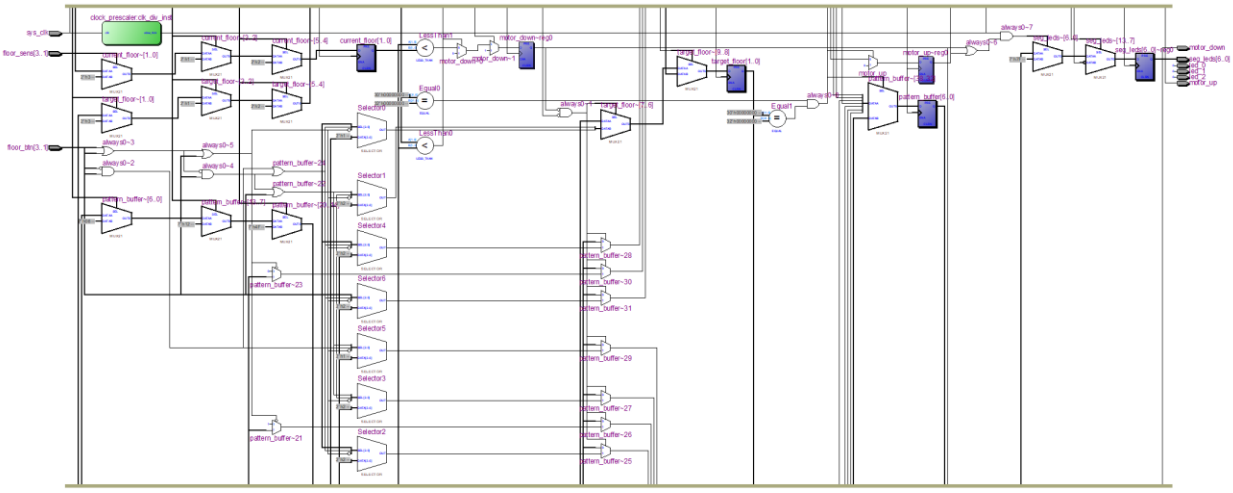
Now Floor Sensor 3 has activated, signaling that the elevator has reached floor 3, which disables motor up, and stops blinking the seven-segment display, showing a solid 3.

Pin Planner



The Pin Planner was used to assign all inputs and outputs of the elevator controller to the physical pins on the FPGA. Each external signal in our design, such as the floor buttons, reed switches, motor control outputs, LEDs, and the 7-segment display lines are associated with the programmable I/O GPIO-0 pins shown.

RTL View



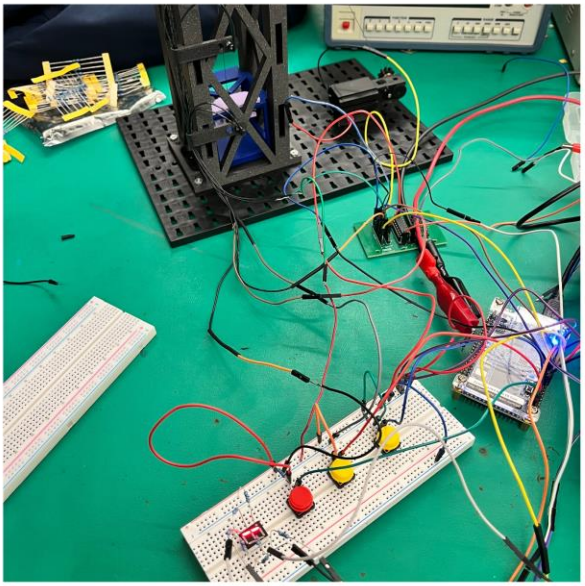
The RTL view shows the structure of our elevator controller after the Verilog code has been compiled. Instead of displaying the behavioral code, this view breaks the design into combinational logic blocks, multiplexers, comparators, and registers that the FPGA uses to implement the system.

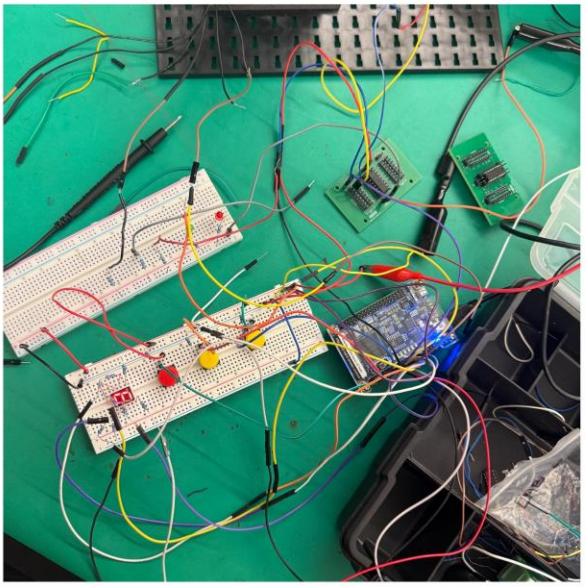
At the left side of the diagram, the inputs (sys_clk, floor_btn, and floor_sens) feed into a network of multiplexers that update the current_floor, target_floor, and pattern_buffer registers. These registers represent the main state of the controller and are driven by the clock through flip-flops. The clock prescaler module appears at the top left and produces the blink_pulse signal used for displaying blinking and calibration.

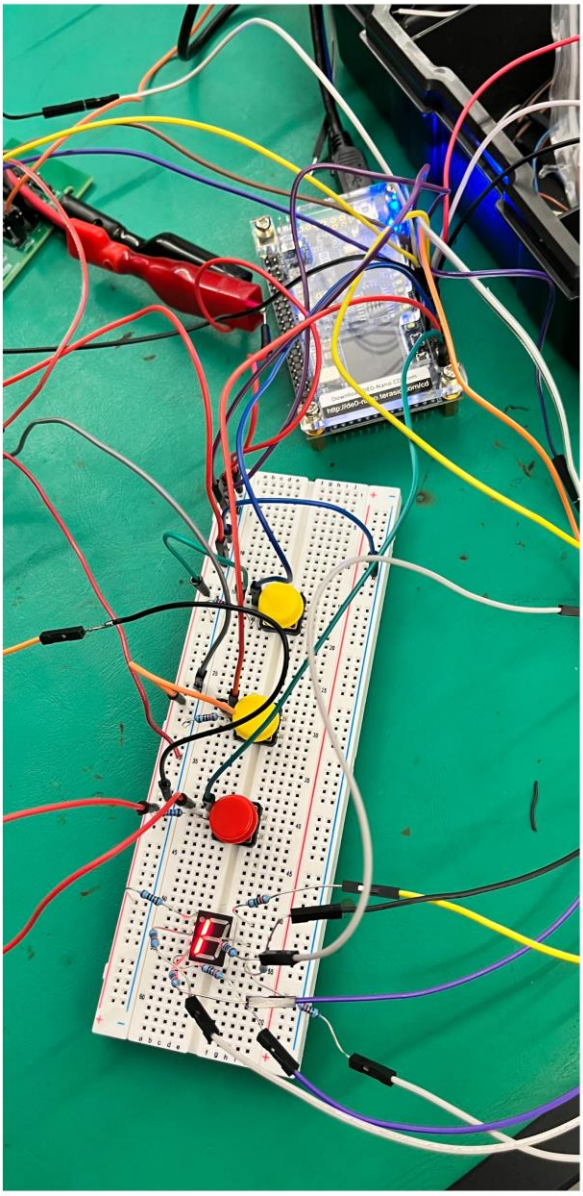
Across the center of the RTL view, comparators such as less than, equal, and greater than blocks determine whether the elevator should move up, move down, or stop. These comparator outputs feed into another set of multiplexers that update the motor_up and motor_down control signals.

The right side of the diagram contains the logic for the 7-segment display control. The pattern_buffer value passes through multiple multiplexers and eventually drives the seg_leds output. Additional logic gates use the blink_pulse signal to override the display when the elevator is moving, which produces the blinking effect.

Pictures of Final Setup







Project Limitations

One of the main limitations in our design was related to state storage and our original plan to implement the controller using two discrete D flip-flops. While the state diagram appeared straightforward on paper, mapping the logic directly onto individual D-FFs on the FPGA led to a mix of synthesis and hardware issues. The flip-flops did not always power up in a known state, and the behaviour we observed on the board often differed from our simulation results. Because of this, we switched to the current approach where the system state is stored using the current_floor and target_floor registers inside the elevator_core module. This resolved the functional issues in simulation but prevented us from fully correcting the original D-FF design.

Another limitation came from relying on internal reset behaviour. We initialized current_floor and target_floor to zero and treated this condition as a calibration state. While this worked reliably in simulation, the FPGA does not guarantee those register values at power-up. As a result, the system could become stuck in calibration or fail to transition into normal operation. A dedicated hardware reset input, and a clearly defined calibration phase would make the design more stable.

Although the full system worked as expected in simulation, we were unable to demonstrate all features on the physical hardware. During the demo, the blinking 7-segment display, clear floor indication via the reed switches, and the direction LEDs driven by the motor control logic all functioned correctly. However, the motor itself did not physically move, even though the expected control signals were generated in simulation. This exposed a gap between digital design theory and real world hardware behaviour. Overall, emphasizing the importance of proper signal conditioning, output drive capability, and thorough hardware verification.

Despite these challenges, the project still allowed us to build a functioning elevator controller in simulation and significantly deepened our understanding of sequential logic, initialization behaviour, and hardware debugging. The experience also pointed out several improvements we would make in future iterations, including a more reliable reset strategy, better motor driver integration, and stronger overall state management.

Design Selection

For our design, we chose to implement the elevator controller using a combination of sequential registers (current_floor, target_floor, and pattern_buffer) and simple combinational decision logic instead of relying strictly on D flip-flops. This approach allowed us to maintain clear state control, which was easier to understand and visualize, while avoiding the instability and hardware issues we encountered with manual flip-flop mapping. Using registers inside the always @(posedge clk) block gave us much more reliable behavior on the FPGA.

We selected a priority based input scheme for the push buttons so the elevator only accepts a new request when it is not already moving. This helped keep the logic simple and prevent conflicting commands, which were breaking our system. For positioning, we used reed switches directly as the feedback mechanism to update the current floor.

To manage the display behavior, we used a clock divider (prescaler) to generate a slow blinking signal from the 50 MHz system clock, making it visual for the human eye. This allowed us to create a visual indicator on the 7-segment display whenever the elevator was in motion. Blinking the display rather than updating numbers continuously helped keep the design simple while still meeting the requirement for visual feedback.

These design choices focused on building a stable, easy to understand controller that behaved consistently. The combination of state registers, comparator motor logic, and reed switch sensing provided a solid balance between functionality and simplicity.

Project Achievements

Despite the challenges we encountered throughout the project, we were able to successfully code a three level elevator system that met all the requirements in the manual, as well as a few bonus inclusions. However, full physical functionality with the motor hardware was incomplete. The system consistently responded to Floors 1, 2, and 3 after a press of each push button then, showed up on the seven-segment display, and blinked until a reed switch gave an input corresponding to the target floor.

One of our key achievements, that solved many of our early troubles, was making the system run on a clean and predictable state-based control approach using registers rather than our initial idea of manually controlled D flip-flops. This resulted in smoother operation and simpler code which eliminated the inconsistent behavior that was giving us errors during testing. The auto-calibration feature, which makes the elevator move down until a floor is found on startup, was also verified in our simulation. However, it was not working on the physical model as the reed switches on demo day were inconsistent with our setup made. We also achieved the blinking 7-segment display during movement, clear floor indication, and simple direction feedback with LEDs through the motor control outputs.

In conclusion, despite some inconsistencies between simulation and physical hardware, the project demonstrated our ability to combine digital logic design, hardware testing, troubleshooting, and how to create a functioning elevator controller. The final system behaves consistently in simulation, responds accurately to user inputs, and showcases a practical application of sequential logic on an FPGA.