**SAN JOSE STATE UNIVERSITY**

**Charles W. Davidson College of Engineering**

**ENGINEERING DEPARTMENT**

A black and white logo

Description automatically generated

**Elevator Controller Using Verilog on FPGA Board**

**Group Members:**

Duy Cu

Hai Long

Hung Nguyen

Tung Nguyen

**EE271**: **DIGITAL SYSTEM DESIGN & SYNTHESIS**

Professor: Binh Le

Date: May 10, 2024

**TABLE OF CONTENTS**

**I. Introduction ------------------------------------------------------------------ 2**

**II. Hardware and Software Tools ------------------------------------------- 2**

1. Hardware **------------------------------------------------------------------ 2**
2. Software **------------------------------------------------------------------- 3**

**III. Design Overview ----------------------------------------------------------- 4**

1. Objective **------------------------------------------------------------------ 4**
2. System Architecture **----------------------------------------------------- 4**
3. Finite State Machine **----------------------------------------------------- 5**
4. Block Diagram **------------------------------------------------------------ 6**
5. Challenges and solution **-------------------------------------------------- 7**

**IV. Working ----------------------------------------------------------------------- 8**

1. Overview **------------------------------------------------------------------- 8**
2. Top Module **---------------------------------------------------------------- 10**
3. Elevator Control Module **------------------------------------------------ 12**
4. Display Seven Segment Module **---------------------------------------- 17**
5. Narrator Module **----------------------------------------------------------- 21**
6. Blocking Memory Generator **-------------------------------------------- 24**
7. Simulation testbench **----------------------------------------------------- 27**

**V. Simulation --------------------------------------------------------------------- 34**

1. Simulation Link **----------------------------------------------------------- 34**
2. Vivado Link **---------------------------------------------------------------- 34**

**VI. Conclusion -------------------------------------------------------------------- 47**

**VII. Contribution ---------------------------------------------------------------- 47**

Reference **--------------------------------------------------------------------------- 49**

Annexure **---------------------------------------------------------------------------- 50**

## I. Introduction

The project aims to design an elevator control system capable of efficiently managing operations across a four-floor building. The system is tasked with facilitating smooth elevator movements, floor selection, and door operations. Leveraging the finite state machine (FSM) approach, our goal is to develop a robust elevator controller using Verilog for behavioral modeling. This design encompasses the seamless integration of elevator functions, including floor selection, door control, and emergency procedures. The elevator system is designed to interact with users, accepting floor inputs and ensuring safe and efficient transportation between floors. Additionally, the project endeavors to implement advanced features such as emergency stop functionality and status indicators. The elevator control system can be interfaced with various hardware platforms, offering flexibility in deployment and integration with existing building infrastructure. Through this project, we aim to demonstrate the feasibility and effectiveness of FPGA-based elevator control systems in enhancing building functionality and user experience.

**II. Hardware and Software Tool:**

**1. Hardware:**

The FPGA Blackboard (Xilinx XC7007S ZYNQ): The primary hardware component, crucial for elevator control operations, incorporates programmable logic capabilities and integrates various parts:

* Switches: Serve as inputs for elevator control.
* Buttons: Function as an SOS and reset button.
* Seven Segment Display: Utilized to convey elevator status; the last digit indicates the current floor, the third digit shows the direction of movement (up or down), and the first and second digits represent door states (open or closed).
* LEDs: Indicate active switches.
* Speaker Output: An external speaker is used to broadcast auditory signals.

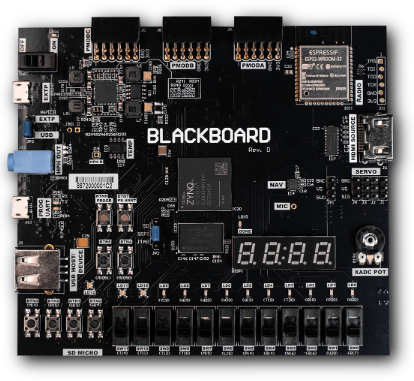


Figure 1: The FPGA Blackboard.

**2. Software:**

Xilinx Vivado: Vivado serves as the fundamental tool for FPGA development, enabling the creation, implementation, and verification of Verilog code. It is crucial for synthesizing the hardware description and generating bitstreams to program the FPGA.



Figure 2: Vivado Software Icon.

Verilog HDL (Hardware Description Language): is the programming language used to define the behavior of digital circuits on the FPGA, crucial for specifying elevator control operations.

**III. Design Overview**

**1. Objective:**

Our project aims to design a reliable elevator controller for a four-floor building, focusing on essential functions like floor selection and door operations. We prioritize optimizing processing speed and FPGA resource utilization to ensure efficient performance. By leveraging advanced techniques like FSM modeling and Verilog implementation, we aim to create a high-performance system that enhances user experience and integrates seamlessly with existing infrastructure. Through rigorous testing and validation, we ensure our controller meets safety standards and provides smooth transportation between floors.

**2. System Architecture:**

The elevator control system implemented in this project utilizes a Mealy finite state machine (FSM) to regulate its operations. This FSM manages various states, including Idle, SOS, Door Open, Door Close, Slow, Fast, and Door Jammed, each characterized by specific entry conditions and transition rules. The system's interface provides clear feedback to users by displaying crucial information such as floor position and door status on a 7-segment display. Through this architecture, the system ensures efficient, safe, and transparent elevator operations, capable of dynamically responding to both routine usage and emergency scenarios. The following figures illustrate the flowchart depicting the design of this system.

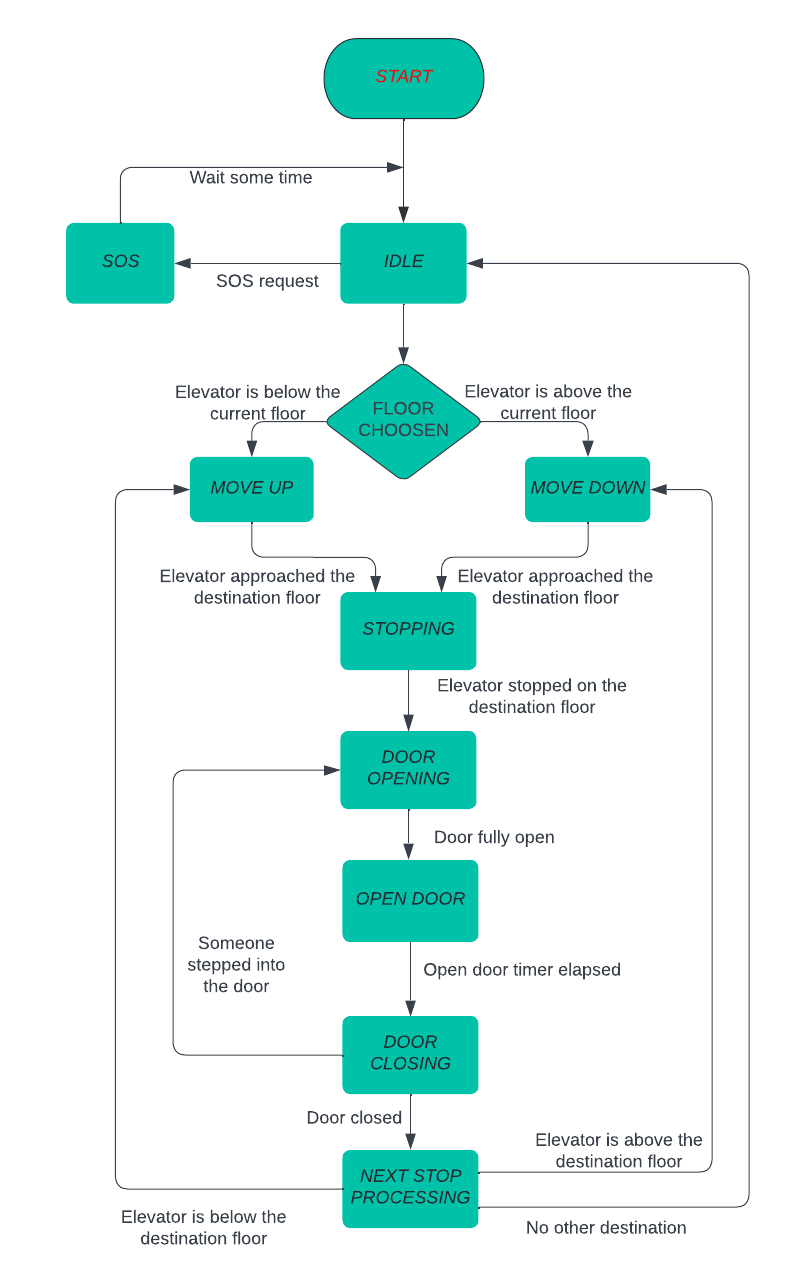


Figure 3: Flowchart of Elevator Controller.

1. **Finite State Machine:**

To begin on the project, the drafted finite state machine model is shown as in Figure 4, and Table 1 explains what is happening within each state. A Mealy finite state machine was chosen for this project since the elevator design was asynchronous and dependent on certain inputs to proceed to the next state, as opposed to Moore state machines which are sequential and only depend on a clock cycle to change outputs.

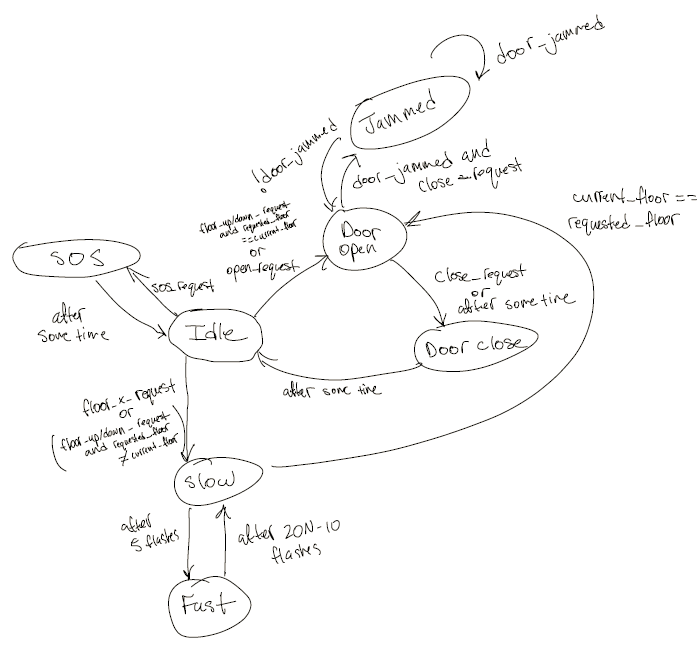
****

Figure 4: Preliminary Finite State Machine

**4. Block Diagram:**

When formulating the Verilog code, a block diagram resembling Figure 2 was sketched out. This diagram comprised essential logic blocks such as elevator control, narration, and display. Drawing from the initial finite state machine design for the elevator, connections between the elevator control logic block and both the narration and display logic blocks were established. To streamline operations, a single system clock was designated for the entire code. While certain intermediate signals remained undetermined at this stage, the block diagram served as a foundational guide for organizing the necessary Verilog modules.

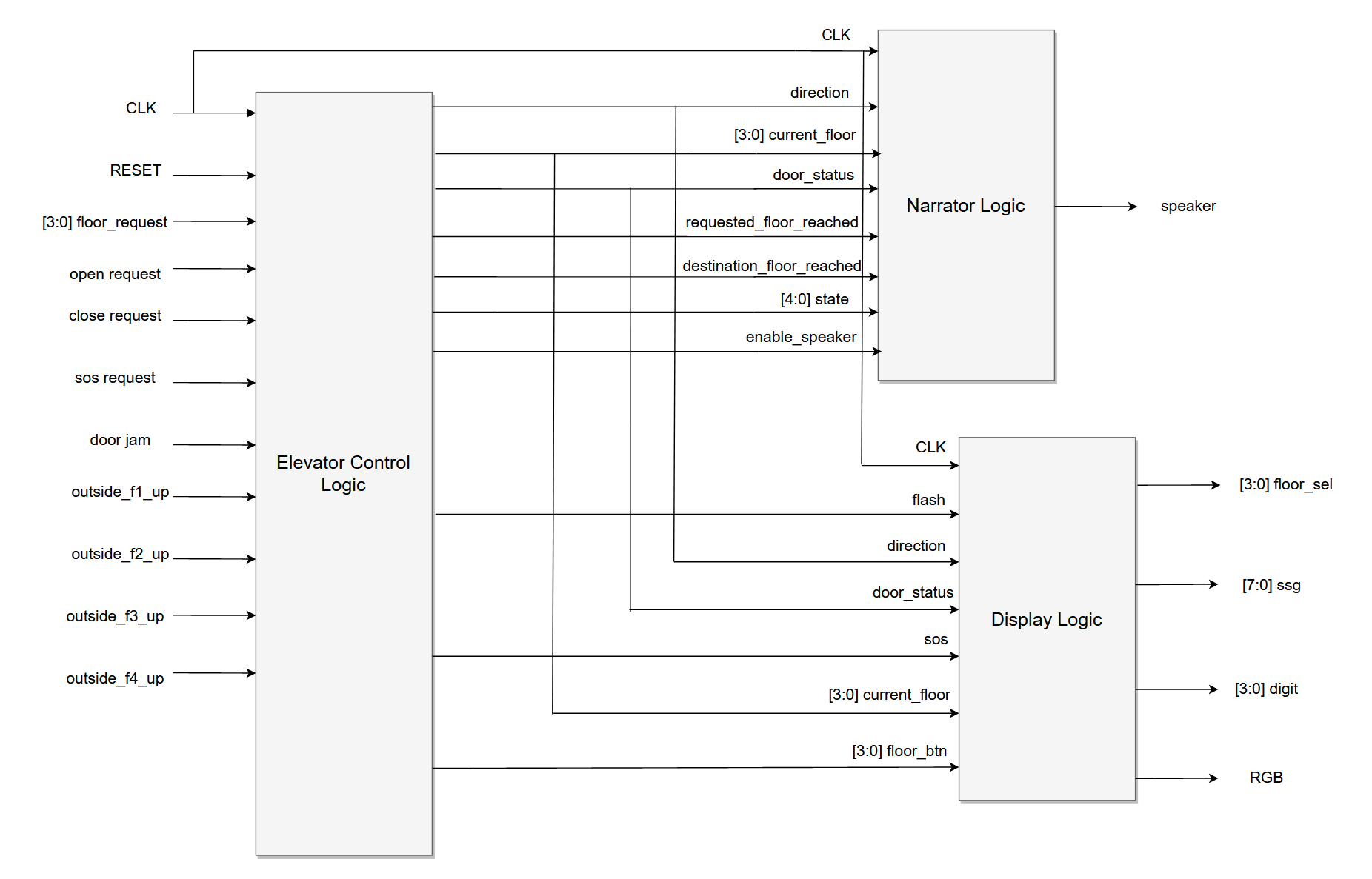


Figure 5: Block Diagram of Overall Design

Utilizing this schematic, a top-level module was created to amalgamate all individual modules within Verilog. This arrangement facilitates seamless communication between modules, aligning with the depiction in Figure 5.

**5. Challenges and Solutions:**

During the initial implementation of the Finite State Machine (FSM), the project encountered several challenges and errors. The original design assumed that elevator controls would operate independently of whether the commands or calls originated from inside or outside the elevator. This assumption proved overly complex within our allocated time frame, prompting a redesign. Consequently, the buttons on the FPGA used to control the elevator were divided into two categories: outside call buttons and inside call buttons. Additionally, it became apparent that the preliminary FSM lacked the necessary number of states to manage the elevator operations effectively, leading to numerous bugs and operational issues. To resolve these problems, the initial FSM was incorporated into the program, and new states were added as required to enhance functionality and stability.

**IV. Working**

1. **Overview**

**Idle**

The entry conditions for the elevator's default state are set to occur when the program initializes at startup, after the elevator doors have remained closed for a specified period, and when the SOS alarm is deactivated after being triggered. In this default state, the elevator is in standby mode, ready to receive user input. Essential information is continuously displayed on a 7-segment display, including the current floor, the floor the elevator is approaching or stationed at, and the door status, indicating whether it is open or closed. This state is critical as it forms the operational foundation of the elevator, ensuring that it is always prepared to respond to user commands while providing clear and immediate feedback on its status.

**SOS**

The entry conditions for the SOS state are triggered when the SOS button is pressed while the elevator is in its Idle state. Upon activation, the system emits an SOS alarm accompanied by flashing red lights and an auditory alert to address the emergency. Once the SOS signal ceases, the system seamlessly transitions back to the Idle state, ready to resume regular operations. This design ensures that emergencies are promptly addressed, while also maintaining a clear and efficient pathway for the elevator to return to normal functionality once the emergency situation has been resolved.

**Door Open**

The entry conditions for the Door Open state are met under three specific scenarios: first, when the open request button inside the elevator is pressed; second, after the elevator arrives at the requested floor; and third, if a floor request button outside the elevator is pressed and the requested floor matches the current floor. In this state, the 7-segment display shows that the doors are open, clearly indicating to passengers that they can enter or exit. The state transitions occur either when the close request button is pressed or after a set duration, at which point the system smoothly transitions to the Door Close state, readying the elevator for its next movement. This state plays a crucial role in efficient door management, ensuring both the safety and convenience of passengers as they navigate through the building.

**Door Close**

The entry conditions for this state occur when the close request button is pressed while the elevator is in the Door Open state or after a predetermined duration has passed in the Door Open state. In this state, the 7-segment display indicates that the door is in the process of closing, providing a clear visual to passengers. Following a set period, the system seamlessly transitions back to the Idle state, signaling the completion of the door closing process and the elevator's readiness for further commands. This state ensures efficient operation by managing the transition from the Door Open state back to the Door Close state and default state, maintaining smooth functionality and passenger safety throughout the process.

.

**Slow**

This state is activated under two conditions: firstly, when a specific floor button is pressed inside the elevator, and secondly, when a floor request button is pressed outside the elevator, provided the requested floor differs from the current floor. In this state, the 7-segment display's tenth digit flashes slowly, conveying whether the elevator is moving up or down. The number of flashes correlates with the distance to the requested floor, providing passengers with visual feedback on the elevator's movement. Transitioning from this state occurs in two ways: firstly, after a specific count of slow flashes, the system transitions to the Fast state; secondly, upon reaching the requested floor, the elevator moves to the Door Open state, facilitating entry or exit. This state optimizes passenger experience by providing clear indications of elevator movement and ensuring prompt transitions between states based on floor requests.

**Fast**

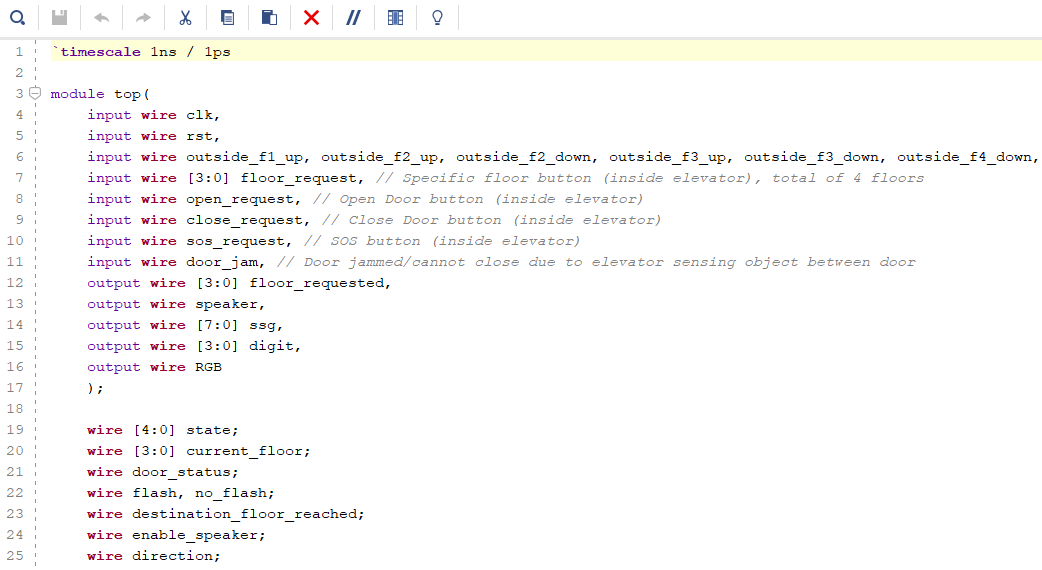
The entry condition for this state occurs when the Slow state finishes its predetermined number of flashes. In this state, the 7-segment display flashes rapidly, with the frequency determined by the distance between floors. As the elevator moves, the displayed floor number updates accordingly. Transitioning out of this state happens after the fast flashes complete, where the system returns to the Slow state to continue its operation, or if the destination floor is reached, it transitions to the Door Open state, allowing passengers to enter or exit the elevator. This state ensures efficient and dynamic floor tracking while facilitating smooth transitions between states based on elevator movement and floor destinations.

**Door Jammed**

When the entry conditions are met, specifically if the door is jammed and the close request button is pressed, the elevator enters the Door Jammed state. In this state, the 7-segment display indicates that the door is open as a clear signal of the jam. The elevator remains in this state until the jam is resolved, ensuring safety and preventing further operation until the issue is addressed. Once the jam is resolved, the system smoothly transitions to the Door Open state, allowing normal operation to resume. This state prioritizes safety by halting operation when a door jam occurs, ensuring passengers are not put at risk and facilitating the prompt resolution of the issue before returning to regular functionality.

**2. Top Module**

The top module serves as the central integration point for an elevator control system, designed to handle a variety of inputs and outputs essential for elevator operation. It manages inputs such as a clock signal (clk), a reset signal (rst), and numerous user-interface elements like inside and outside elevator floor request buttons, door control buttons, an emergency SOS button, and a sensor input for door jams. The key outputs of this module include signals to control a speaker, a seven-segment display, and an RGB output for additional visual feedback.



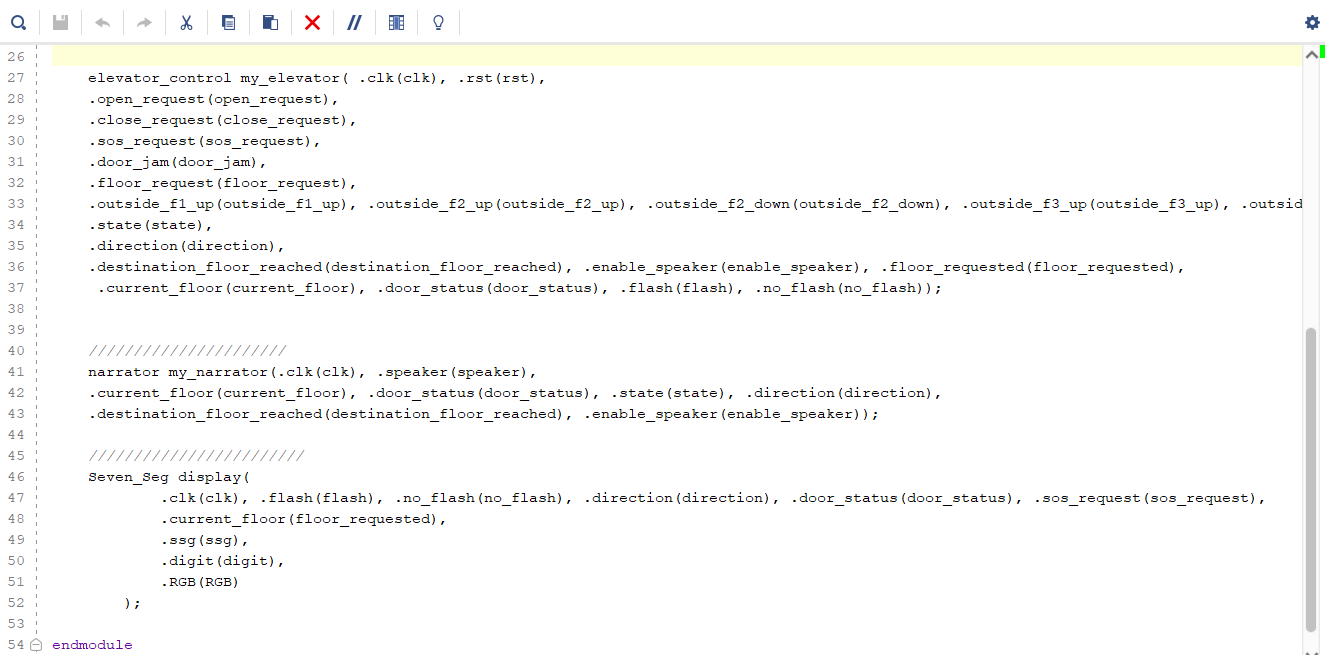


Figure 6 : Top module

Within the module, several submodules play critical roles in ensuring the smooth operation of the elevator. The elevator\_control submodule processes all critical functions such as floor requests, door operations, and emergency responses. It determines the current state of the elevator, manages door statuses, and directs the elevator’s movement and floor positioning. Additionally, the narrator submodule is responsible for audible notifications such as floor announcements and emergency alerts, enhancing user interaction with clear audio cues based on the elevator's current state and activities.

The visual feedback for the elevator system is managed by the Seven\_Seg submodule, which controls the seven-segment display to indicate the current floor, door status, and directions or emergency states through interactive visual signals like flashing. This comprehensive management of display and audio feedback ensures a user-friendly interface that keeps passengers informed and safe. Overall, the top module orchestrates these complex components to deliver a cohesive, efficient, and safe elevator operation, demonstrating sophisticated integration of control logic, user communication, and system responsiveness.

**3. Elevator Control module** The elevator\_control module implemented in Verilog is a comprehensive digital system designed to manage the operation of an elevator in a building with four floors. This module controls the elevator's movements and door operations based on various inputs and maintains an internal state machine to navigate between these states effectively.

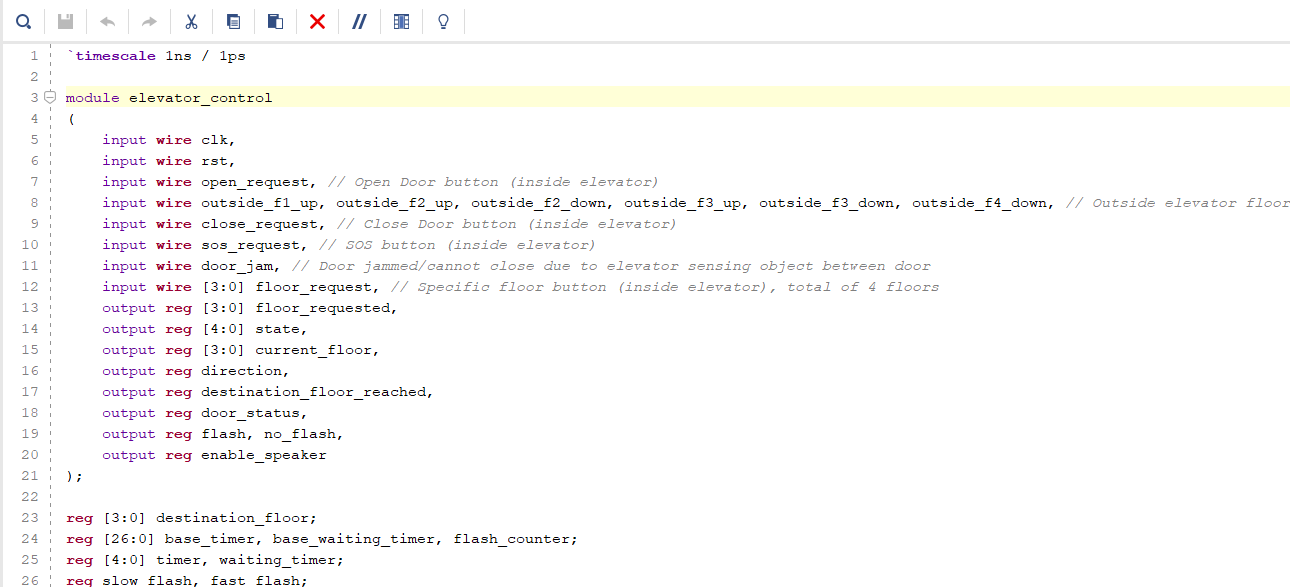


Figure 7: elevator\_control module with input and output

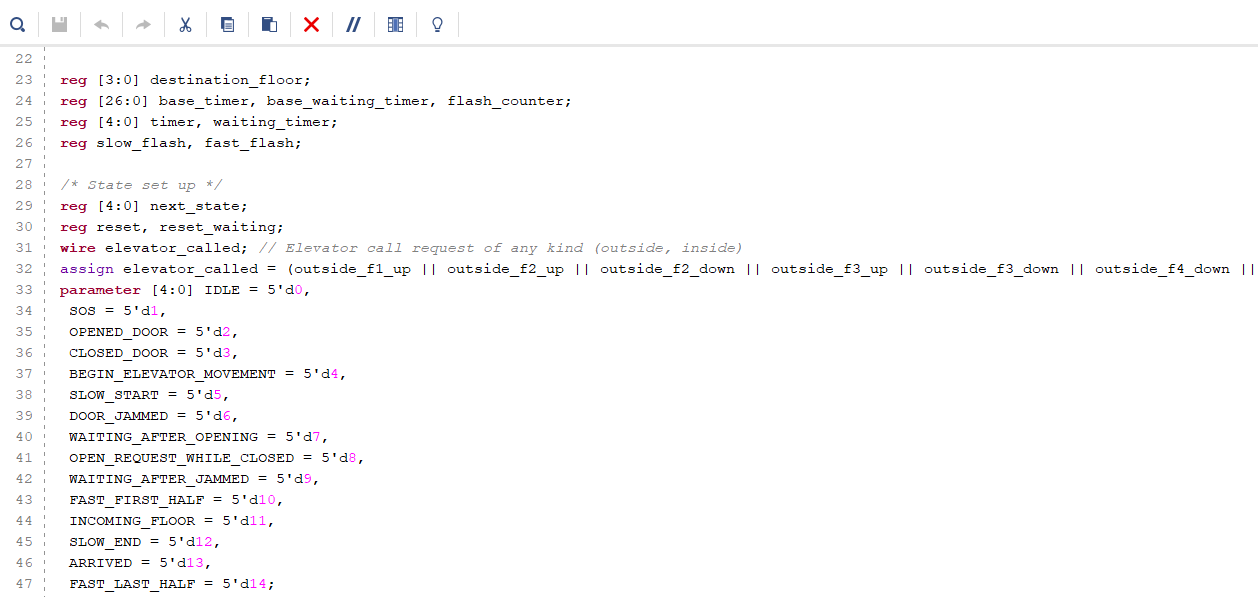
The module's inputs include signals from floor request buttons both inside the elevator (floor\_request) and on each floor (outside\_f1\_up, outside\_f2\_up, outside\_f2\_down, etc.), door operation requests (open\_request and close\_request), an emergency button (sos\_request), and a door jam detection signal (door\_jam). It also receives a clock (clk) and reset (rst) signal, essential for its time-based operations and initializations.

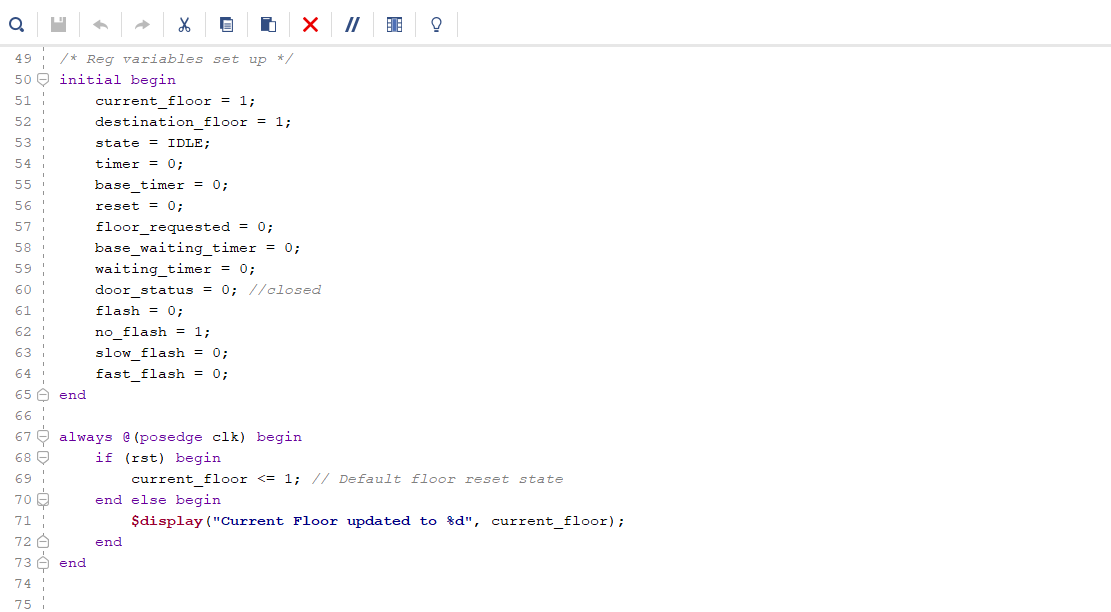
The outputs of the module include the requested floor (floor\_requested), the current state of the elevator (state), the current floor (current\_floor), the direction of movement (direction), door status (door\_status), and signals for managing visual or audio alerts (flash, no\_flash, enable\_speaker).

At the heart of the module lies a state machine with states such as IDLE, OPENED\_DOOR, CLOSED\_DOOR, BEGIN\_ELEVATOR\_MOVEMENT, and emergency states like SOS and DOOR\_JAMMED, among others. The state transitions are based on the current state, input conditions, and timers that manage durations for actions like door open durations or movement phases.

For instance, the elevator can move from an idle state to responding to a floor request, initiating movement, managing slow and fast travel phases (SLOW\_START, FAST\_FIRST\_HALF, etc.), and finally reaching the destination and opening the door (ARRIVED). Special conditions like door jams or emergency signals trigger states that handle these exceptions, ensuring safety and responsiveness.

Timers are crucial in this module, managing delays for actions such as the response time after a door opens or during the slow start and slow end phases of the elevator's movement. The flash and no\_flash outputs manage signaling, such as blinking lights to indicate specific states or actions, enhancing the interface's intuitiveness and safety features.





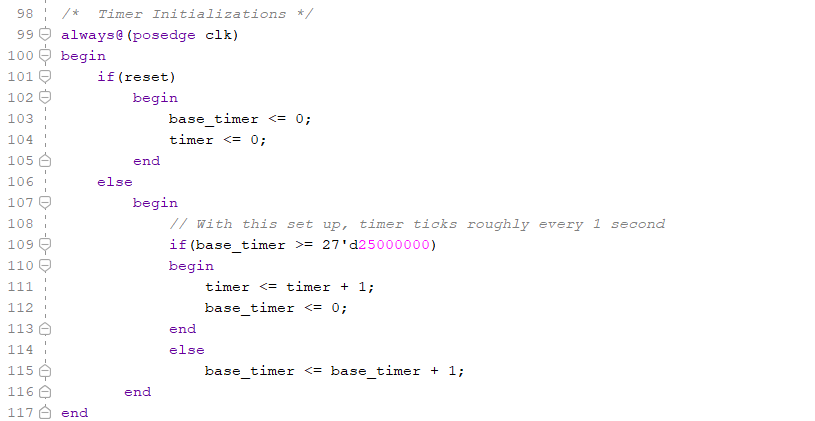
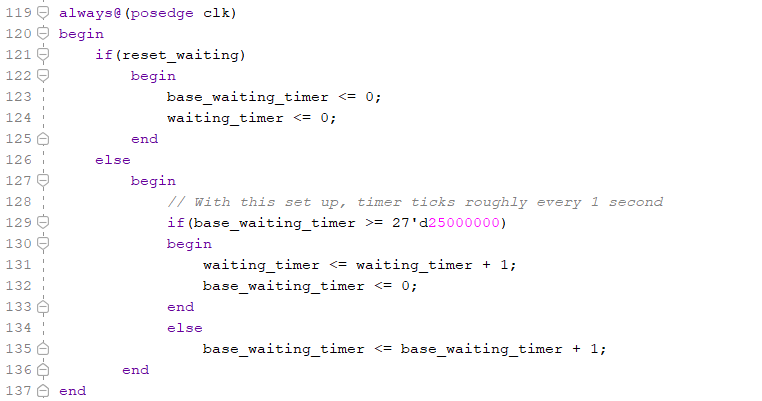
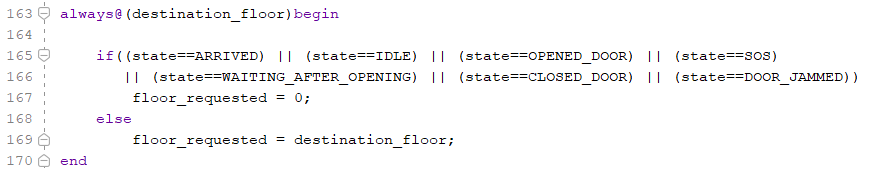
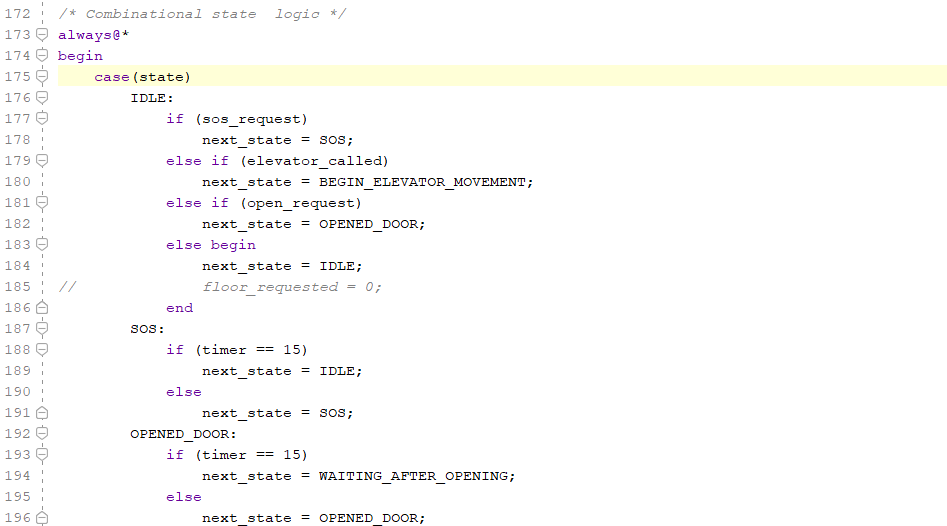


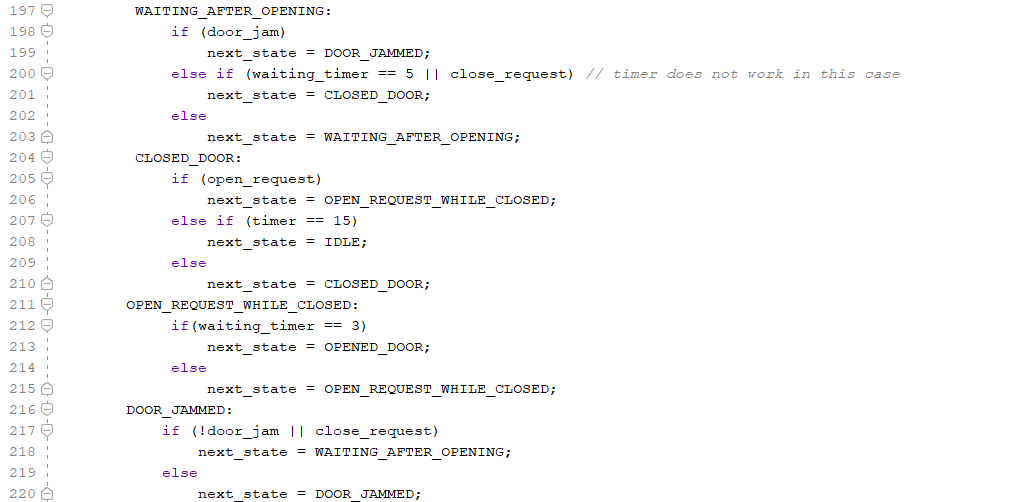
Figure 8: States set up and Reg variables et up in elevator\_control module

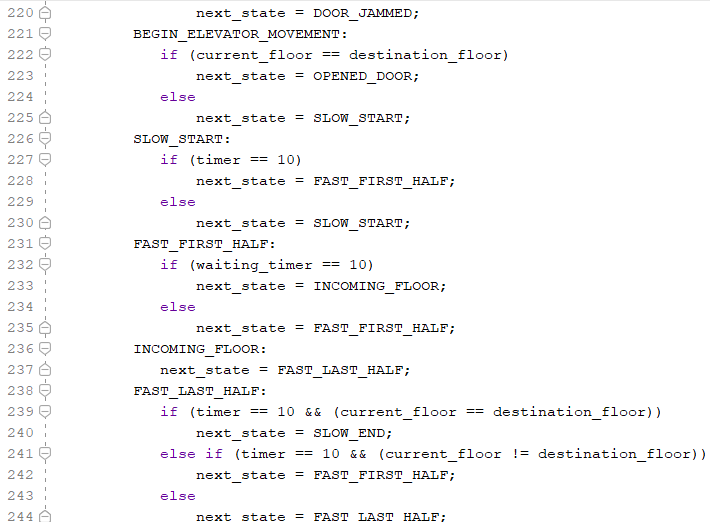
The Verilog code snippet outlines the combinational logic of an elevator control system using a state machine. It dynamically evaluates transitions based on the current state and input conditions. For instance, from the IDLE state, the elevator can move to SOS upon an SOS request, start moving if called, or open its doors if the open request is activated. Additional states manage the elevator's operations, such as OPENED\_DOOR, which transitions to WAITING\_AFTER\_OPENING after a set time, or to DOOR\_JAMMED if a jam is detected. The elevator's movement through the building is controlled by sequences like SLOW\_START and FAST\_FIRST\_HALF, adjusting the speed as necessary. This logic ensures the elevator operates efficiently and safely, responding to both standard and emergency conditions effectively.











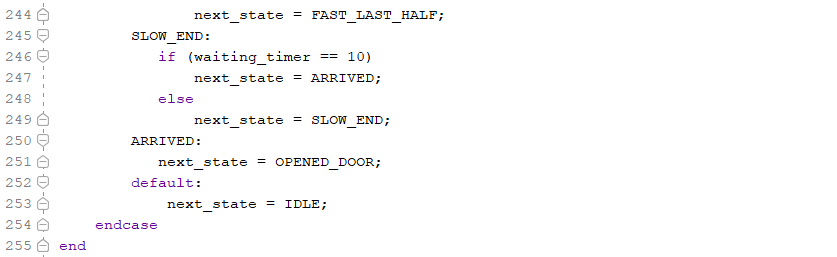
****

Figure 9: Combinational State Logic in elevator\_control module

Overall, the elevator\_control module is a robust design tailored for an elevator system, emphasizing safety, efficiency, and user responsiveness through well-structured states and transitions governed by internal logic and timed operations.

**4. Display Seven Segment**

The Seven\_Seg module in Verilog is designed to control the display aspects of an elevator system on a seven-segment display (SSG), incorporating various status indicators including floor number, elevator direction, and door status. It receives multiple inputs such as the current floor, elevator direction, door status, flash control signals, and an SOS request, and outputs the relevant display data through segment and digit controls, along with an RGB LED output. This module uses a clock signal (clk) to drive its timing-sensitive operations, ensuring that display updates are smooth and timely.

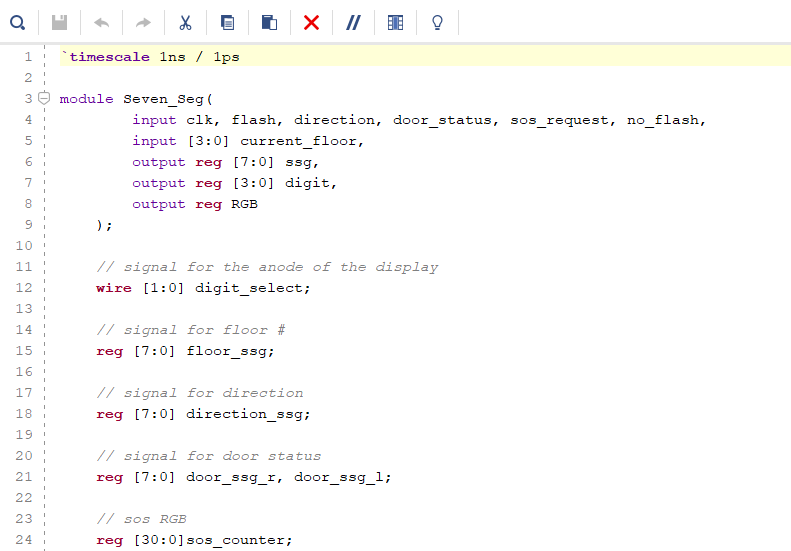
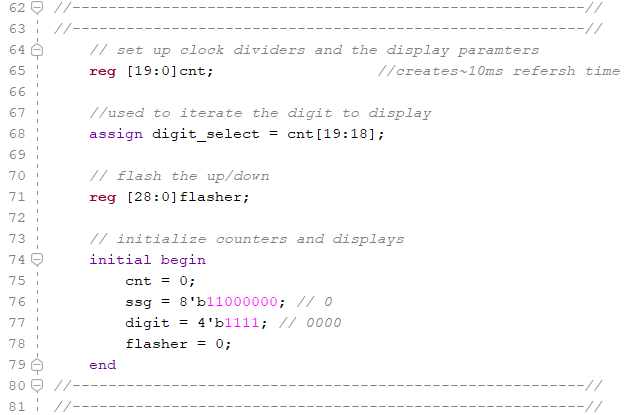
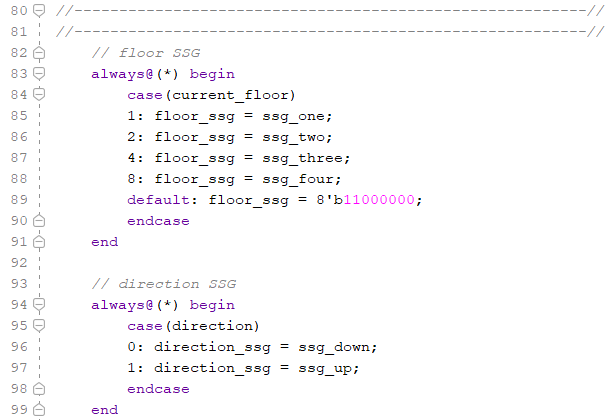
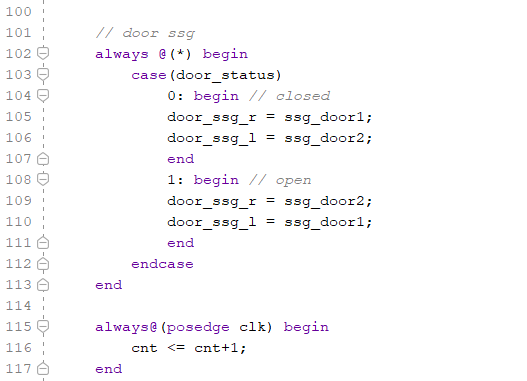


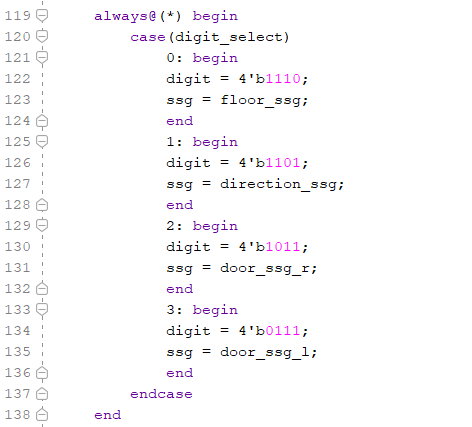
Figure 10: Seven\_Seg module

The module's internal logic includes multiple registers and parameters to manage and represent the display information. For floor numbers, predefined seven-segment codes are assigned to each floor, switching based on the current\_floor input. Directional status is indicated by either an up or down arrow, which flashes based on the flash signal to attract attention when the elevator is moving. Door status is shown by distinct segment patterns for open and closed states, toggling between these based on the door\_status input. Additionally, an SOS mode is managed through a counter that toggles the RGB output, creating a flashing effect when an SOS request is active.









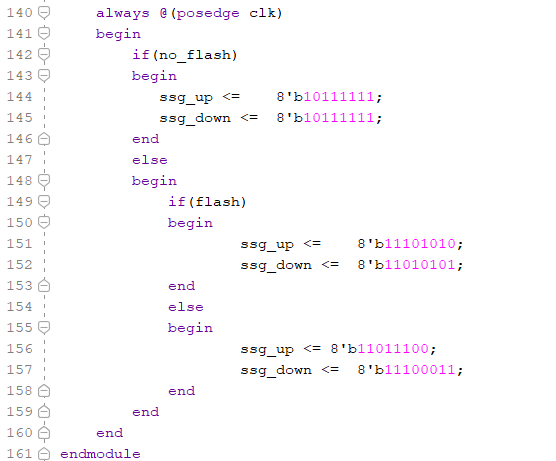


Figure 11: Set up for 4 Seven Segment Display in Seven\_Seg module

In terms of operation, the module utilizes several counters: cnt for managing the refresh timing of the display, ensuring each segment is updated in a round-robin fashion through the digit\_select signal, and flasher to control the flashing of directional arrows based on the elevator's movement status. The logic to select and display the appropriate seven-segment code for each digit is encapsulated in an always@(\*) block, which updates the output registers ssg and digit according to the digit currently being driven by digit\_select. This ensures that the elevator's status is clearly communicated to passengers through both visual and interactive elements, enhancing the overall user interface of the system.

**5. Narrator module**

The narrator module in Verilog is a sophisticated audio management system designed to handle audible notifications within an elevator control system. It operates based on a series of inputs such as clk for the clock signal, enable\_speaker to activate or deactivate the speaker, state indicating the current state of the elevator, current\_floor, requested\_floor\_reached, direction, door\_status, and destination\_floor\_reached. The output from this module is primarily the speaker signal, which is used to drive audible alerts corresponding to various states and events within the elevator.

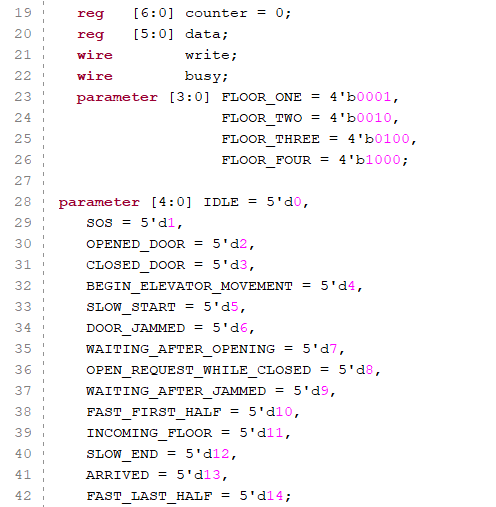
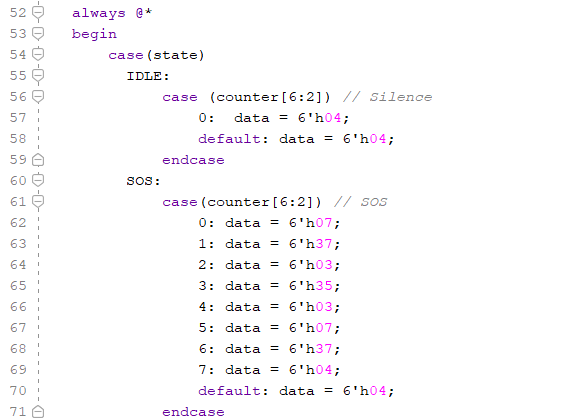
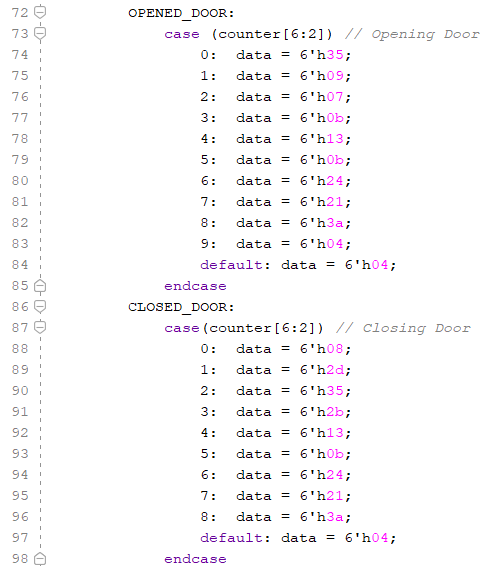
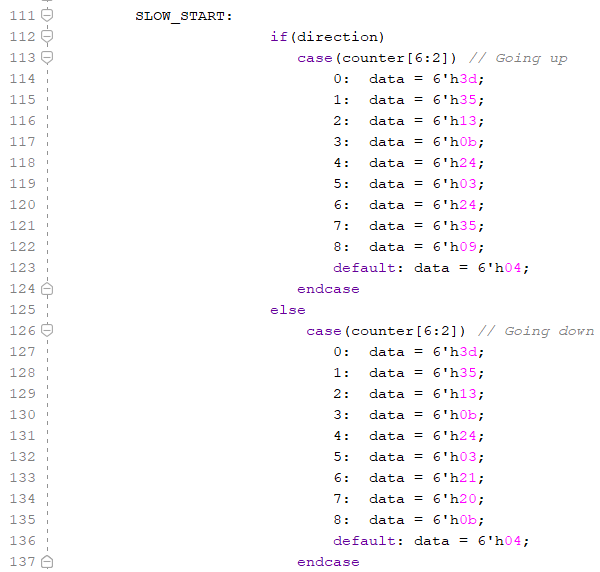


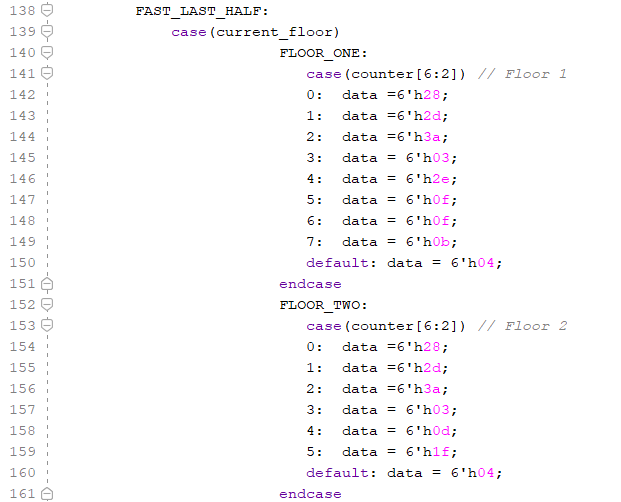
Figure 12: narrator module

Internally, the narrator utilizes a counter (counter) to sequence through different sound data (data) values that are predefined for different states and conditions such as idle, door opening, door closing, elevator movement, and emergencies like door jams. This approach allows for a diverse range of sounds that provide clear and distinct notifications for different elevator scenarios. For example, it produces a series of beeps or tones to indicate the opening or closing of doors, a unique set of tones for emergencies, and different sounds based on whether the elevator is ascending or descending.









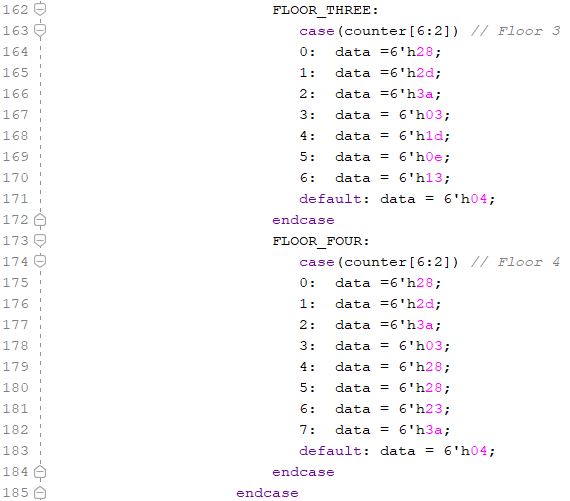


Figure 13: Scenario of States in narrator module

The output logic is controlled through an always @(posedge clk) block which increments the counter in each clock cycle, looping through the counter to sequence sound data based on the current state and conditions. An always @\* block is used to dynamically assign the data based on the current state and floor number, which is then written to a chatter module instantiated within the narrator. The chatter module effectively manages these data sequences to produce the actual sound signals emitted by the speaker, coordinating closely with the narrator to ensure timely and appropriate audio feedback corresponding to the elevator's operational state. This setup ensures that passengers are well-informed of the elevator’s status through auditory cues, enhancing user experience and safety.

**6. Block Memory Generator**

The my\_phonemes component within the design hierarchy is an instantiation of a Block Memory Generator IP core, specifically configured for use as a Single Port ROM as seen in the first image. This setup ensures that the memory is read-only, which aligns perfectly with its intended purpose—to store phonemes or audio samples that do not require modification during runtime. With a 32-bit address interface, this component is capable of managing a significant amount of data, facilitating efficient access to a variety of stored phonetic content essential for the system’s audio output needs.

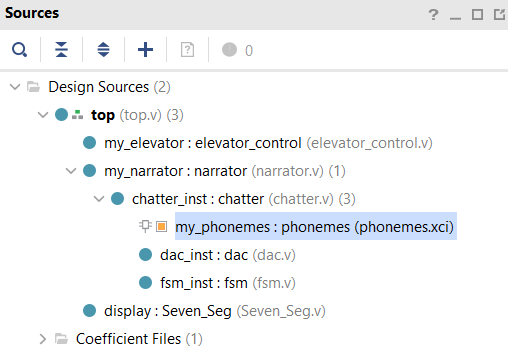
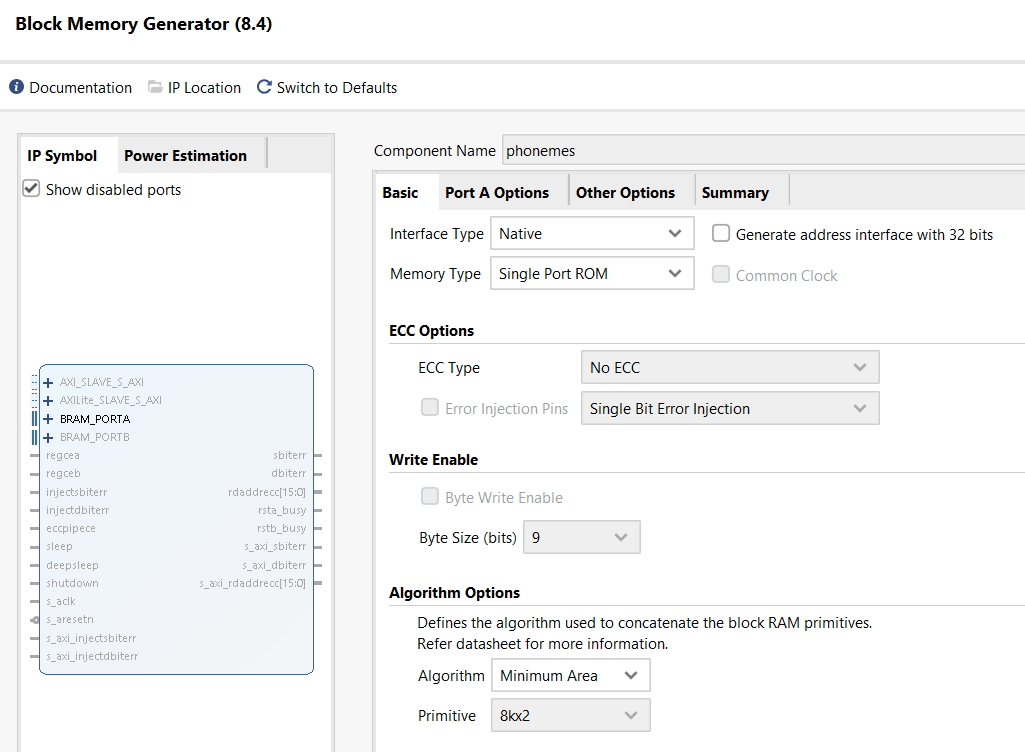


Figure 14: my\_phonemes component within the design hierarchy

The my\_phonemes component is configured as a Single Port ROM, indicating its read-only memory status which is ideal for storing phonemes that do not require modification during system operation but are consistently accessed for audio output. Its memory interface features a 32-bit address interface, enabling the handling of a substantial volume of data and allowing for efficient access to a diverse array of stored phonemes or audio samples. This configuration ensures that the system can quickly retrieve and process sound data, facilitating effective auditory communication within the elevator control system.

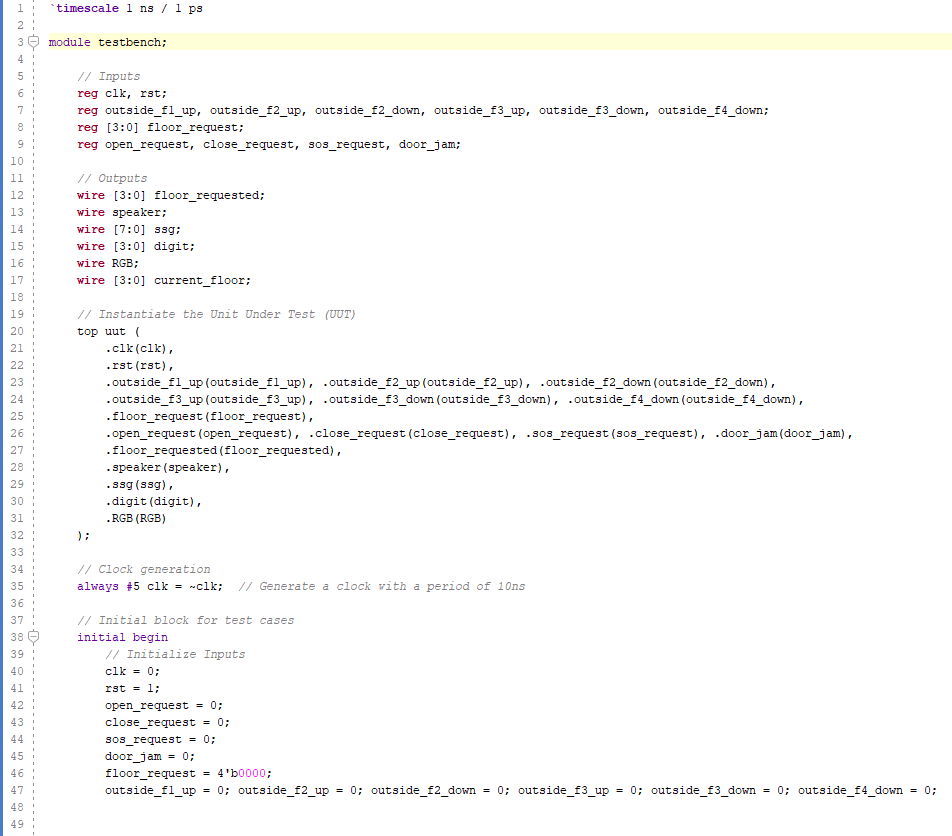
****Figure 15: Block Memory Generator for my\_phonemes component

Within the elevator control system, the my\_phonemes plays a crucial role in enhancing user interactions by providing clear auditory feedback. It stores a library of phonemes or pre-recorded messages which the narrator module accesses to produce audible notifications or instructions through the speaker system. This functionality is especially important for accessibility and user guidance, delivering real-time audio cues in response to user actions or system status changes, such as announcing floors, directions, or safety alerts. The integration of this phoneme storage within the FPGA ensures that these audio samples are readily accessible with minimal delay, essential for maintaining an effective and responsive user interface in the elevator environment.

**7. Simulation Testbench**

1. **Top module testbench**

The testbench module meticulously evaluates the elevator control system's performance and responsiveness via simulated user inputs and scenarios. It employs a global reset for consistency and synchronizes operations with a 5-nanosecond clock signal. Through simulated internal and external requests, including abnormal conditions like door jams and SOS signals, the testbench monitors output signals to validate expected system responses. This comprehensive assessment spans 50,000 nanoseconds, ensuring thorough validation of critical functions from normal navigation to emergency protocols.



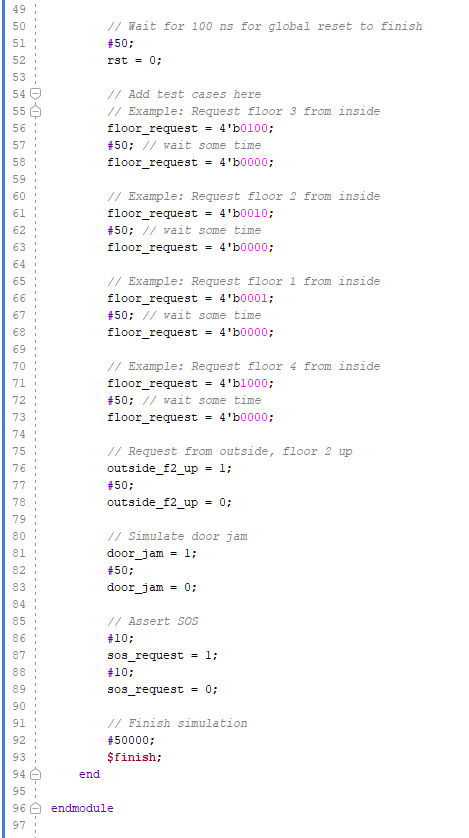
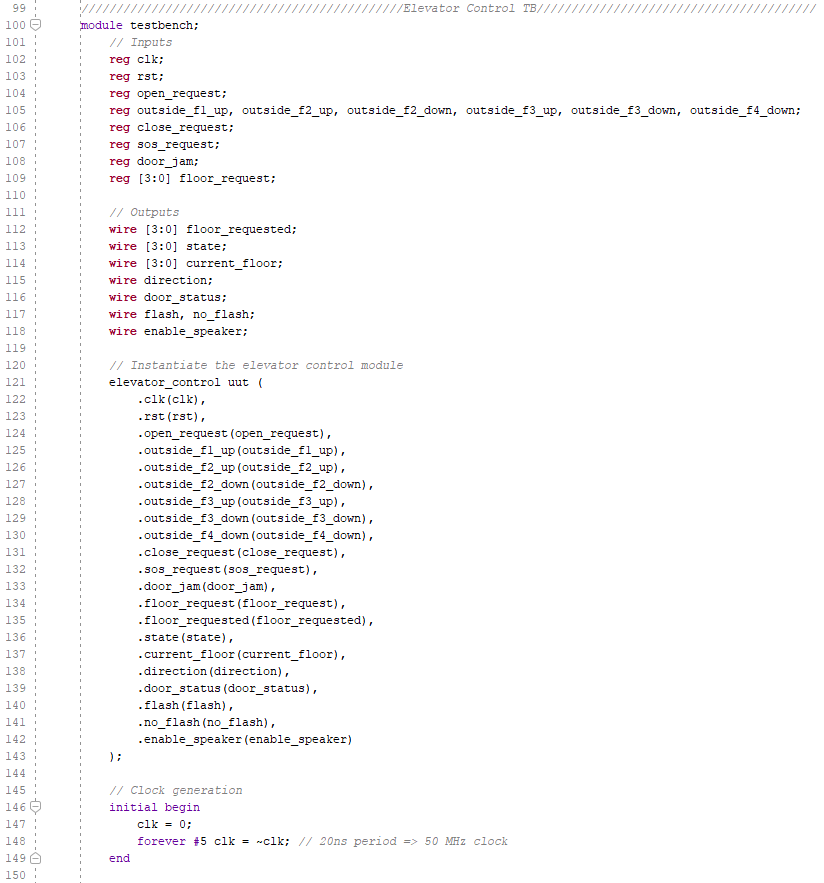


Figure 14: Top Module Testbench

1. **Elevator Control Testbench**

The testbench for the `elevator\_control` module simulates various elevator scenarios by generating a 50 MHz clock signal and initializing control signals. Following an initial reset, it tests two cases: an elevator request to floor 2 and another to floor 3 with a simulated door jam. The `elevator\_control` module instance is connected to relevant inputs and outputs, including floor requests and status indicators, to verify the system's functionality.

****

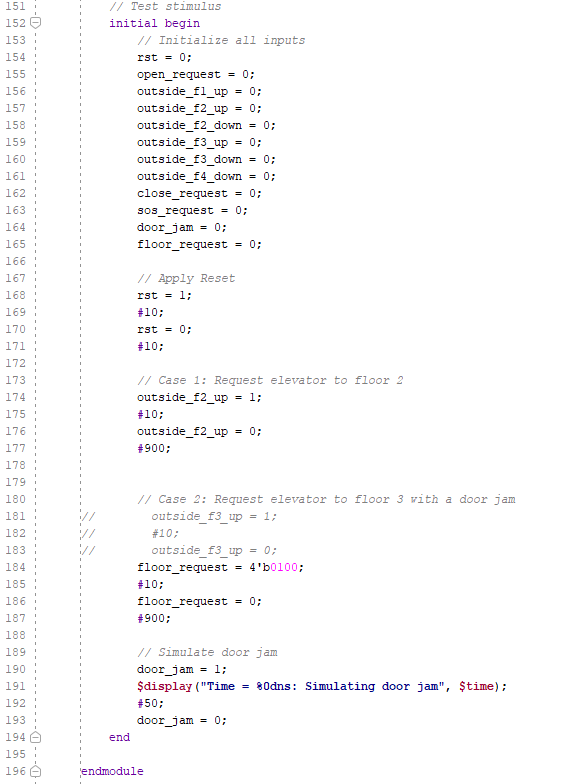
****

Figure 15: Elevator Control Testbench

1. **Seven Segment Display Testbench**

The testbench for the `Seven\_Seg` module verifies various input conditions by generating a 100 MHz clock signal and applying a range of stimuli to control signals. The test sequence initializes inputs such as `flash`, `direction`, `door\_status`, `sos\_request`, `no\_flash`, and `current\_floor`, and then simulates various scenarios, including different floor displays, direction changes, flashing modes, door states, and SOS requests. An instance of the `Seven\_Seg` module connects to the relevant inputs and outputs, including seven-segment displays (`ssg`), digit select (`digit`), and RGB outputs, to verify proper operation across all simulated conditions.

****

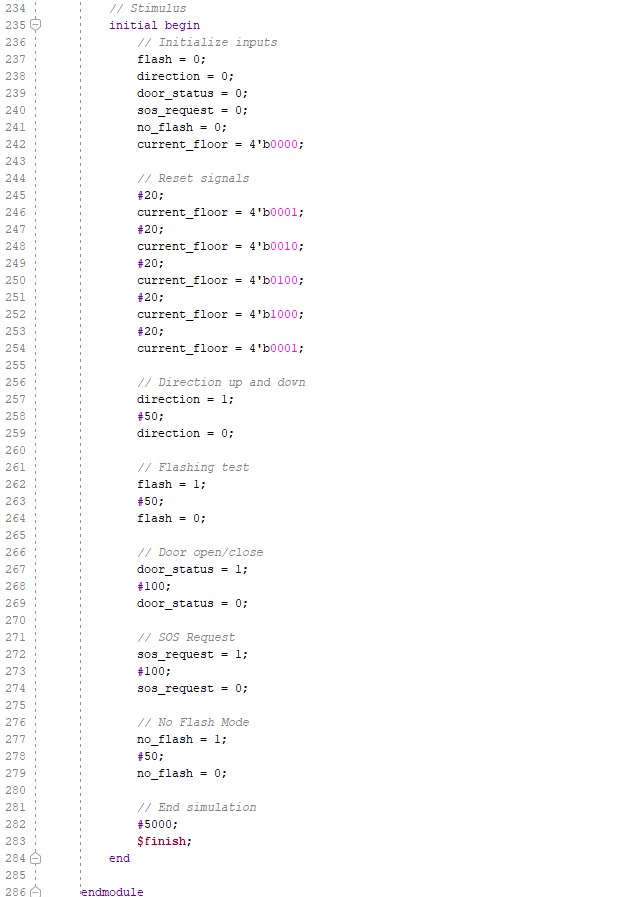
****

Figure 16: Seven Segment Display Testbench

1. **Narrator Testbench**

The testbench for the narrator module generates a 100 MHz clock signal by toggling clk every 5 time units. A wire speaker captures the output of the narrator module. An initial block provides user instructions for running the simulation, pauses for an extended period, and then terminates. The narrator module is instantiated and connected to the clk and speaker signals to verify synchronization of the IP clock with the design.

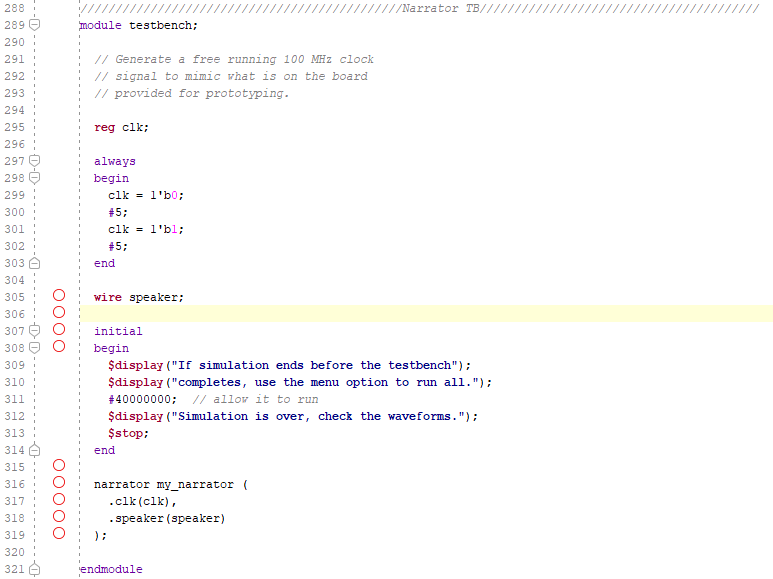
****

Figure 17: Narrator Testbench

**V. Simulation Results**

1. **Simulation Link:** [**Link Video of Simulation**](https://youtu.be/hYH2JSp1T8g)
2. **Vivado Link:** [**Link Vivado**](https://drive.google.com/file/d/1TuOh9WtyL8rKeT_7ehMYeHdS3Ht20vyh/view?usp=drive_link)
3. **Top Module Simulation**

In Test Case 1, the request signal for floor 3 (floor\_request = 4'b0100) is asserted to simulate pressing the button inside the elevator for the third floor. After holding the signal for 50 nanoseconds, the request is cleared, representing the release of the button. This brief assertion aims to verify the system's ability to capture the request promptly and initiate movement to the desired floor, ensuring proper recognition and response to internal elevator commands.



Figure 18: waveform in test case 1

When the floor\_request signal is set to '4' (binary 0100), indicating a call to the third floor, it is effectively acknowledged by the system as shown by the floor\_requested output, which mirrors this input by also displaying '4'. This output updates almost concurrently with the input signal, signaling prompt responsiveness of the system. The floor\_requested output remains set even after the input is reset, indicating that the system retains the request until it is processed or fulfilled. Additionally, the ssg (seven-segment display) output shifts from c0 ((binary 1100 0000) to b0 (binary 1011 0000) around the same time the request is recognized and later reverts to b0, likely reflecting a visual cue or status indicator related to the floor request. This behavior confirms the system's quick response and ongoing acknowledgment of the floor request through both internal status and visual indicators.

In Test Case 2 of the Verilog testbench for an elevator control system, the scenario involves simulating a request to move the elevator to the second floor from inside. This is achieved by setting the floor\_request signal to 4'b0010, which corresponds to the second floor, and maintaining this signal for 50 nanoseconds to ensure the system registers the request. After this duration, the signal is cleared (floor\_request = 4'b0000) to mimic a user releasing the button, effectively testing the system’s responsiveness to internal commands and its ability to reset for subsequent operations.



Figure 19: waveform in test case 2

In the waveform results for Test Case 2 from the testbench, the floor\_requested and ssg (seven-segment display) signals demonstrate effective system responses to an internal elevator request for the second floor. After the floor\_request input is set to 2 (binary 4'b0010), there is a corresponding and timely response in the floor\_requested output, which confirms the system's acknowledgment of the requested floor. Subsequently, the ssg value changes from b0 ( 8’b1011 0000) to a4 (8’b10100100) indicating dynamic updates on the elevator's display panel, in this scenario is number 2. These updates likely reflect the elevator's operational status such as current floor during movement or other status indicators.

In Test Case 3, the request signal for floor 1 (floor\_request = 4'b0001) is asserted to simulate pressing the button inside the elevator for the first floor. After holding the signal for 50 nanoseconds, the request is cleared, representing the release of the button. This brief assertion aims to verify the system's ability to capture the request promptly and initiate movement to the desired floor, ensuring proper recognition and response to internal elevator commands.



Figure 20: waveform in test case 3

When the floor\_request signal is set to '1' (binary 0001), indicating a call to the first floor, the system promptly acknowledges it, as evidenced by the floor\_requested output displaying '1', mirroring the input. This synchronicity underscores the system's prompt responsiveness. The floor\_requested output persists even after the input signal is reset, indicating that the system retains the request until it's processed or fulfilled. As a result, the ssg (seven-segment display) output transitions from a4 (binary 10100100) to f9 (binary 1111 1001) around the moment the request is recognized, and later reverts to b0. The status indicator related to the floor request, in this case number 1 is displayed in the seven segment affirming the system's swift response and ongoing acknowledgment through internal and visual cues.

In Test Case 4, the request signal for floor 4 (floor\_request = 4'b1000) is asserted to simulate pressing the button inside the elevator for the fourth floor. After holding the signal for 50 nanoseconds, the request is cleared, representing the release of the button. This brief assertion aims to verify the system's ability to capture the request promptly and initiate movement to the desired floor, ensuring proper recognition and response to internal elevator commands.

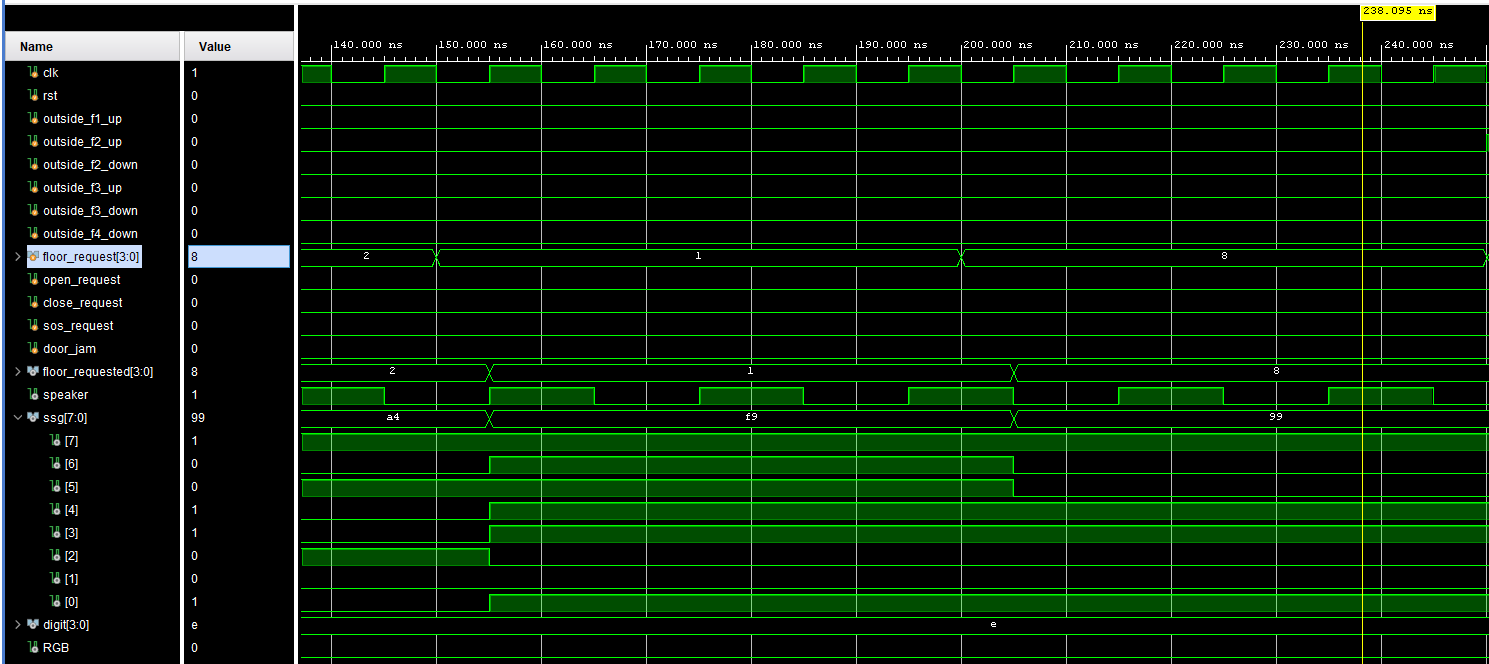


Figure 21: waveform in test case 4

The waveform analysis for Test Case 4 in the testbench reveals the system's efficient handling of an internal elevator request to the 4th floor. Upon setting the floor\_request input to (binary 4'b1000), the system promptly responds with an update in the floor\_requested output, confirming acknowledgment of the requested floor. Subsequently, the ssg value transitions from f9 (8 '11111001) to 99 (8' 10011001), indicating dynamic changes on the elevator's display panel, likely representing the numeral 4. These alterations likely signify the elevator's operational status, such as its current floor during movement or other relevant status indicators.

In Test Case 5, the floor 2 request signal (floor\_request = 4'b0010) is asserted to simulate pressing the button outside the elevator for the second floor. After a 50-nanosecond hold, the request is cleared to simulate button release. This validates the system's promptness in capturing and responding to external elevator commands, ensuring accurate recognition and timely response to floor\_ request.

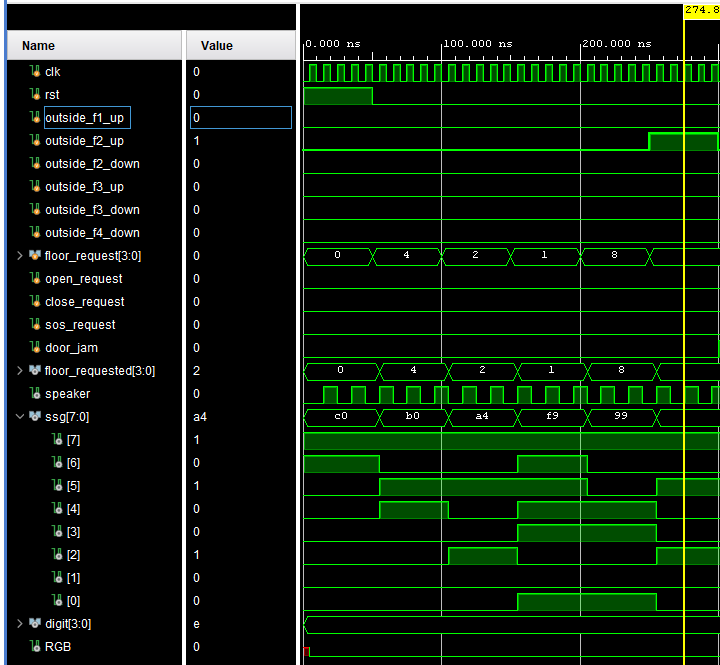


Figure 22: waveform in test case 5

When the outside\_f2\_up signal is activated (set to '1'), indicating a call to the 2nd floor, the system promptly acknowledges it by displaying '2' on the floor\_requested output, mirroring the input. This synchronicity highlights the system's prompt responsiveness. Notably, the floor\_requested output remains active even after the input signal is reset, indicating the system's retention of the request until processed or fulfilled. Additionally, the ssg (seven-segment display) output transitions from 99 (8’b10011001) to a4 (10100100), indicating dynamic updates on the elevator's display panel, likely reflecting its operational status, such as the current floor during movement or other status indicators.

In Test Case 6, the request signal for door\_jam.

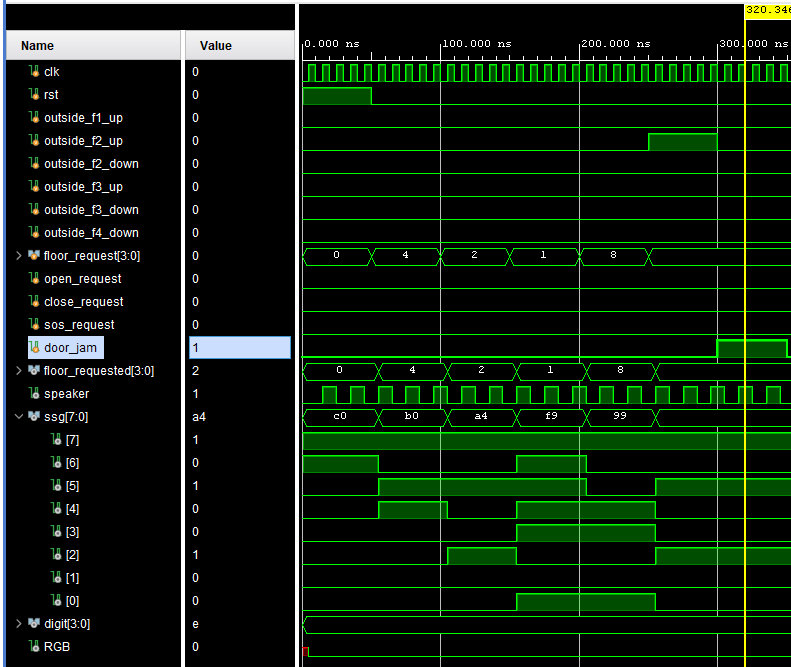


Figure 23: waveform in test case 6

The floor will remain the previous current floor due to the door jam signal. Therefore, the number 2 will be displayed in a seven segment display.

In Test Case 7, the request signal is sos\_request.

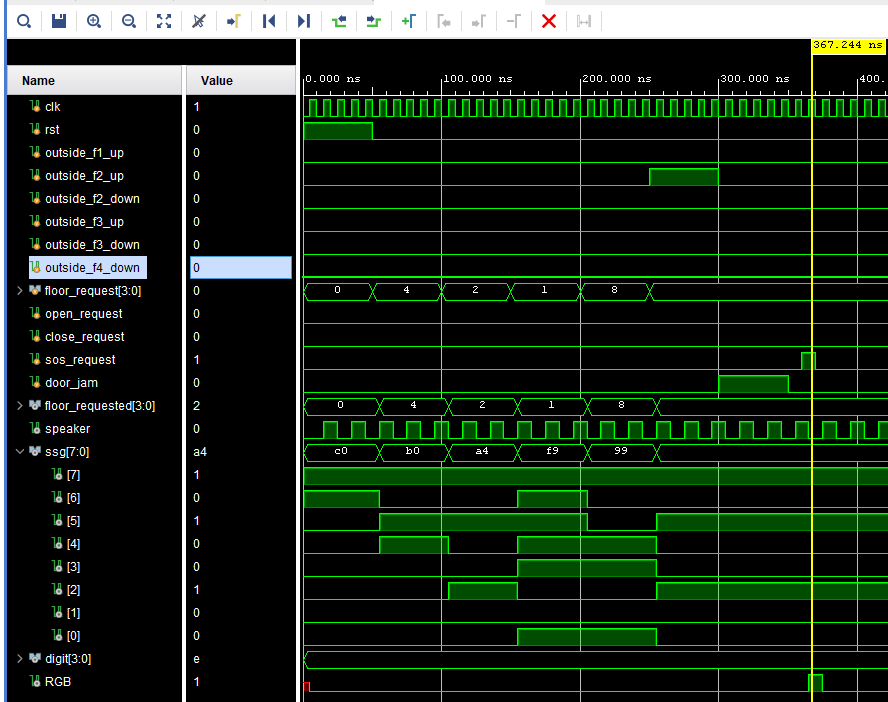


Figure 24: waveform in test case 7

Test Case 7 focuses on simulating an emergency situation within the elevator control system. Here, the trigger for this scenario is the activation of the sos\_request signal, indicating a call for assistance or emergency response. Upon activation of this signal, the system responds by turning on the RGB indicator, likely signaling an emergency state with a distinct visual cue. Additionally, the activation of the sos\_request signal also triggers the activation of flash LEDs on the board, providing further visual indication of the emergency status. The ssg (seven-segment display) output will remain the same value from the previous test case scenario.

**b. Elevator Control Simulation**

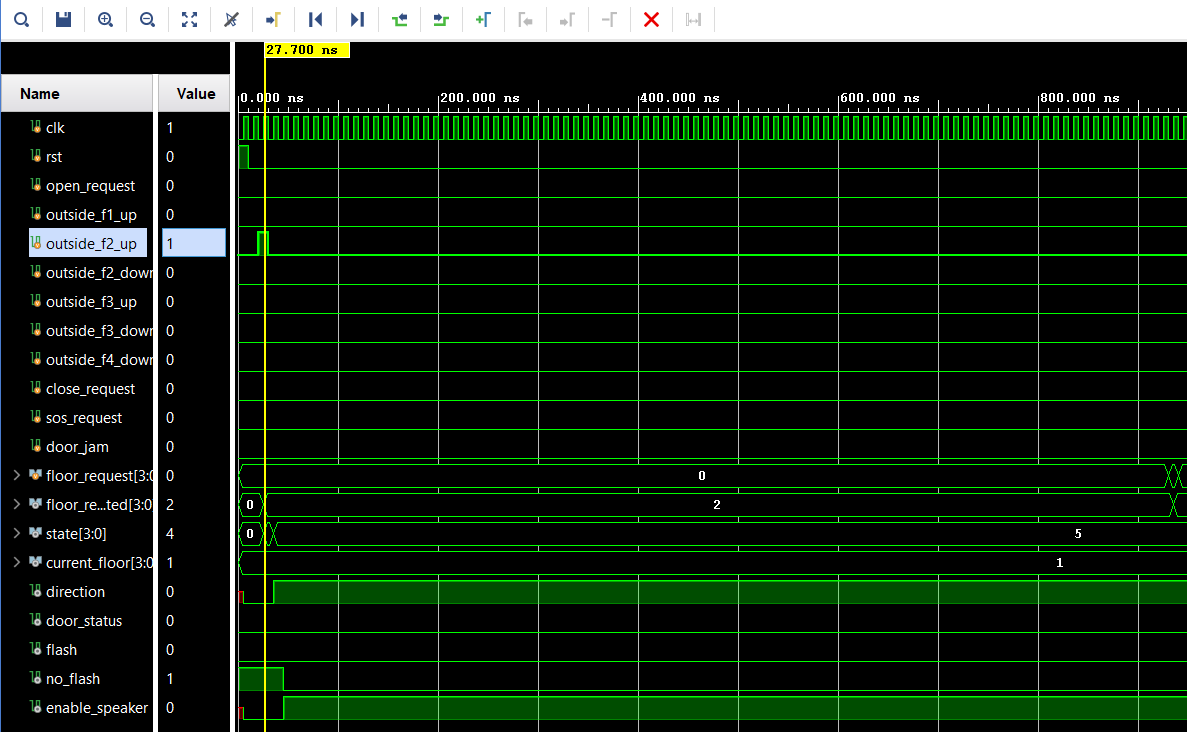
****

Figure 25: Simulation waveform of outside\_f2\_up request

The simulation waveform demonstrates the behavior of the `elevator\_control` module in response to the `outside\_f2\_up` request. The clock signal (`clk`) operates at 50 MHz, and the reset signal (`rst`) initializes the system at the start. After the reset, the `outside\_f2\_up` signal is briefly asserted, setting the `floor\_request` to floor 2 (`4'b0010`). The state machine (`state`) transitions appropriately, and the `current\_floor` progresses to floor 2. The `direction` signal indicates upward movement, while the `door\_status` becomes active, signaling that the elevator doors are open. Visual (`flash`) and audio (`enable\_speaker`) alerts are also triggered as expected. Overall, the waveform confirms the proper handling of the `outside\_f2\_up` request by the `elevator\_control` module.

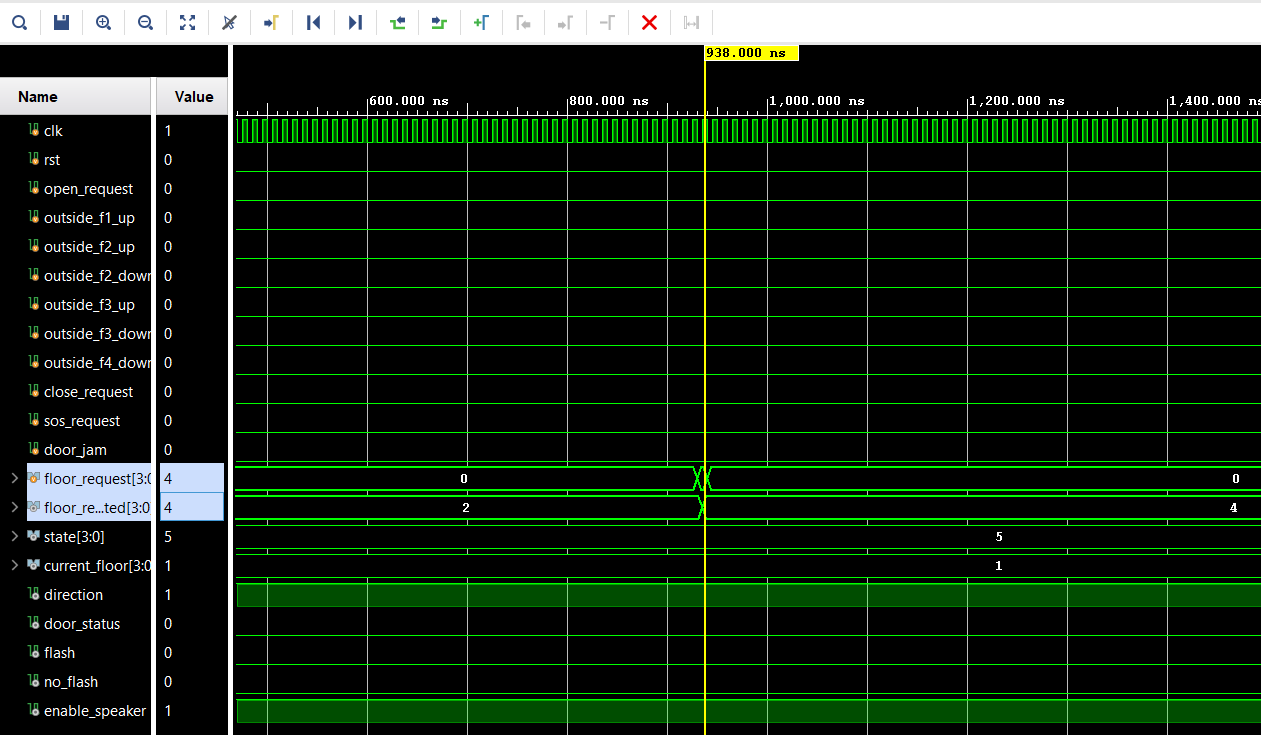
****

Figure 26: Simulation waveform of floor\_request[3:0] = 4’b0100 reflect second floor request

The simulation waveform illustrates the response of the `elevator\_control` module to a request for the second floor. The clock (`clk`) operates at 50 MHz, while the reset signal (`rst`) initializes the system at the start. The `floor\_request[3:0]` signal is set to `4'b0100`, corresponding to a request for the second floor. The state machine (`state`) transitions to state 5, and `current\_floor` updates to reflect floor 2 (`4'b0010`). The `direction` signal indicates upward movement, and `door\_status` becomes active, indicating the elevator doors are open. Alerts (`enable\_speaker` and `flash`) are triggered as expected, confirming the correct handling of the request by the `elevator\_control` module.

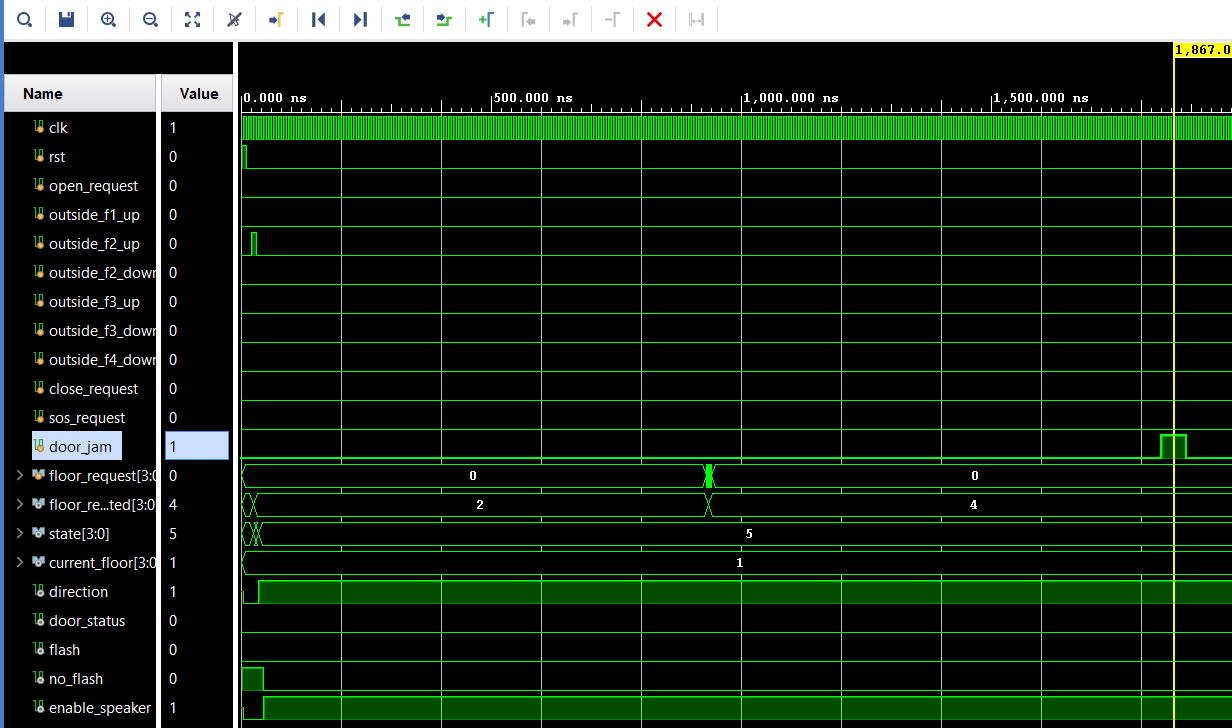
****

Figure 27:Simulation waveform of door\_jam case

The simulation waveform depicts the response of the `elevator\_control` module to a door jam. The 50 MHz clock (`clk`) operates steadily, while the reset signal (`rst`) initializes the system. A floor request is made to floor 2 (`floor\_request[3:0] = 4'b0100`), prompting the state machine (`state`) to transition to state 5 and update `current\_floor` to floor 2 (`4'b0010`). The `door\_status` signal becomes active, indicating that the doors are open.

When the `door\_jam` signal is asserted, the state machine remains in state 5 with the doors open, preventing further movement. Alerts, such as `enable\_speaker` and `flash`, are triggered to signal the jam. Once the `door\_jam` signal is de-asserted, normal operation resumes. This waveform confirms that the `elevator\_control` module handles door jams correctly by maintaining the open-door state and activating the necessary alerts. The system resumes normal operation, confirming that the elevator\_control module is effectively programmed to manage and signal abnormalities like floor\_request, out\_side\_floor up, or door jams. These tests validate the comprehensive functionality and error management within the elevator control system.

**c. Seven Segment Display Simulation**

In the waveform from the testbench, the ssg[7:0] signal, which likely controls a seven-segment display, reflects changes corresponding to the current\_floor[3:0] signal, indicating an interaction designed to display the elevator's floor number visually. In the following figure, all cases of ssg[7:0] signal reflect the values of 8 bits of seven segments according to current\_floor from 1 to 4.

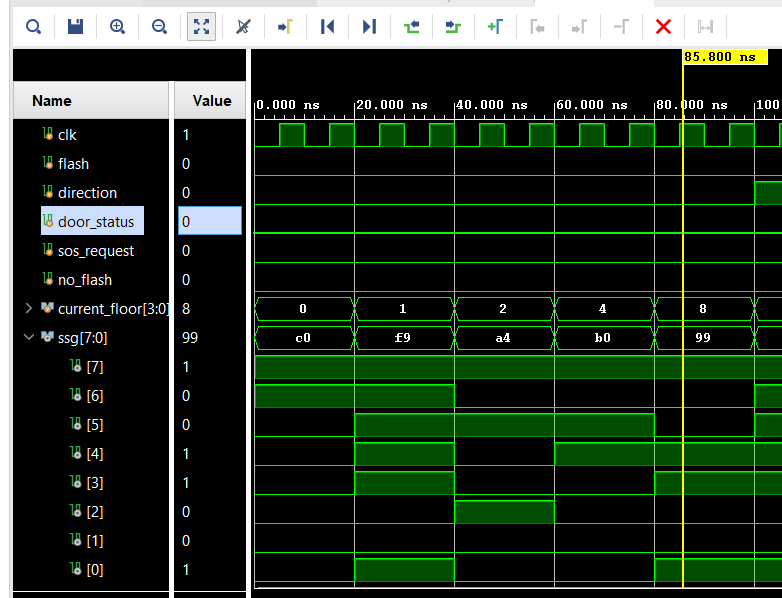


Figure 28: simulation waveform of Seven\_Seg with 4 cases of current\_floor[3:0]

The test bench waveform image illustrates the interactions between the current\_floor and ssg signals in a simulated elevator control system. Initially, the current\_floor signal starts at 0, denoting the ground floor, and transitions through the sequence of floors. The elevator ascends virtually from the ground floor upwards, with the current\_floor shifting from 0 (reset status) to 1 (first floor with hot code 4'b0001), then to 2 (second floor with hot code 4'b0010), progressing to 4 (third floor with hot code 4'b0100), and finally reaching 8 (fourth floor with hot code 4'b1000), occurring at distinct nanosecond timestamps.

For the ssg[7:0], at the beginning, the pattern bit is 8’b10000000 which represents all seven segments displayed (0 is on and 1 is off respectively, and the MSB is the dot segment bit). Then the current\_floor[3:0] is changed to 1, so the ssg[7:0] reflects the number 1 with pattern bit 8’b1111001. The next case of current\_floor[3:0] is second floor, and the pattern bit (8’b10100100) of ssg[7:0] represents the number 2 or second floor. In the next two cases, the third floor and fourth floor is represented by ssg[7:0] by the pattern bit 8’b10110000 and 8’b10011001, respectively. The values shown in the waveform do match the expected numerical representations for floors 1 through 4 on a standard seven-segment display.

**d. Narrator Module Simulation**

The waveform displayed in Figure shows a digital simulation specifically focusing on a clock (clk) signal and a speaker control signal (speaker). The clk signal exhibits a typical digital clock waveform with a regular, high-frequency square wave pattern, alternating between high (1) and low (0) states. This clock is essential for timing the operations within the digital system it is simulating. In parallel, the speaker signal also shows a square wave pattern, but at a significantly lower frequency compared to the clk. This suggests that the speaker signal might be controlling an audio output device, such as a buzzer or speaker, intended to generate sound at a frequency defined by the period of the waveform.

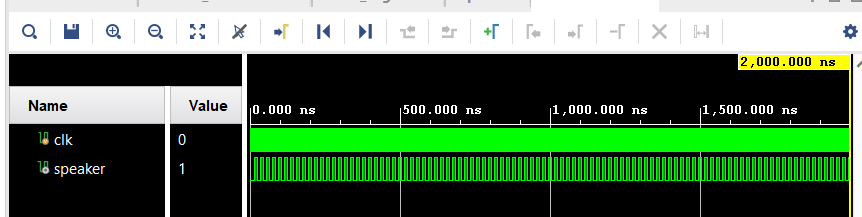
****

Figure 29: Waveform Showing Clock and Speaker Signals for Block Memory Generator

The presence of both signals in the testbench implies that the system is likely designed to output audio signals in synchronization with other system operations, governed by the clock. This setup is commonly used in systems where auditory feedback is required, indicating the successful integration and function of timing and sound generation components within a simulated environment.

**IV. Conclusion**

Our elevator controller was successfully devised and executed on the Black Board by amalgamating components from preceding labs conducted throughout the semester. Key elements such as the 7-segment display, sound module, finite state machine (FSM), blinking LED display, and reset/pause functionalities were drawn from prior lab exercises. These preparatory labs provided the foundation for formulating the elevator design, enabling us to showcase the diverse functionalities of these individual modules. The implementation details, including flowcharts, FSM diagrams, and transition tables, elucidate our approach to integrating these modules. Furthermore, organizing our code with a Top module orchestrating other modules enhanced its clarity and organization. Input and output designations were deliberately chosen for simplicity and clarity. Looking ahead, any future iterations of this project would necessitate modifications to ensure compliance with all specifications. Addressing previous challenges, such as implementing the flashing arrow feature without disrupting program flow, would require careful refinement to achieve seamless operation.

**VII. Contribution**

**Hai Long - Elevator FSM and Control Path Implementation**

Hai Long is primarily responsible for designing and implementing the finite state machine (FSM) for the elevator control system. His duties encompass defining a comprehensive array of system states, including idle, upward and downward movement, door operations, and emergency protocols. Moreover, he develops the control logic that facilitates transitions between different states based on user inputs, sensor data, and system conditions. He meticulously implements the control path to optimize elevator movements, encompassing floor selection, door operations, and emergency procedures. Furthermore, Hai Long collaborates closely with team members to seamlessly integrate the FSM into the overarching Verilog implementation, ensuring strict adherence to project specifications and requirements.

**Tung Nguyen - Signal Definitions and Mapping**

Tung Nguyen undertakes the critical responsibility of defining and mapping the signals essential for communication within the elevator control system. This multifaceted task involves identifying the diverse signals required to facilitate seamless communication among various components of the elevator system, including buttons, sensors, actuators, and the central control unit. Tung meticulously defines signal protocols and interfaces to guarantee compatibility and smooth interaction between hardware components. Additionally, he strategically maps these signals to specific pins or channels on the FPGA board, establishing robust connectivity and efficient data transfer within the system. Furthermore, Tung meticulously verifies signal integrity and reliability through rigorous testing and validation processes, ensuring error-free communication throughout elevator operations.

**Hung Nguyen - Elevator Speaker Output Development**

Hung Nguyen assumes a pivotal role in developing the speaker output functionality for the elevator control system. His responsibilities include designing and implementing the audio output system for essential auditory feedback, such as voice prompts, chimes, and alarms. Hung integrates the speaker output with the finite state machine (FSM) and control logic, ensuring seamless synchronization with elevator movements and events. Thorough testing guarantees clear sound quality and precise timing, synchronized with other system operations. Additionally, he incorporates emergency announcements and system warnings to enhance user safety and communication within the elevator.

**Duy Cu - Audio Mapping and System Integration**

Duy Cu leads the mapping and integration of audio signals into the elevator control system. He maps signals from various sources, such as the speaker output and emergency communication system, onto FPGA interfaces. Cu develops intricate audio mapping logic to efficiently route signals based on system events and user inputs. He ensures seamless integration with vital components like the FSM and conducts thorough testing to verify performance under various conditions, ensuring optimal functionality within the elevator environment.

**Reference**

[1] P. Dutta, B. Mondal, S. Mukherjee, and A. Goswami, "Design and Implementation of an Elevator Controller using Finite State Machine (FSM) on FPGA," International Journal of Computer Applications, vol. 97, no. 10, pp. 1-5, Jul. 2014.

[2] S. J. Raut and M. S. Bhide, "Design of Elevator Controller Using FPGA," International Journal of Engineering Research & Technology (IJERT), vol. 2, no. 2, pp. 1-4, Feb. 2013.

[3] F. Reale and M. Onofri, "Optimization of an elevator control system via reinforcement learning," in Proc. IEEE 2020 15th International Workshop on Semantic and Social Media Adaptation and Personalization (SMAP), Zakynthos, Greece, 2020, pp. 1-6.

**Annexure**

**TOP MODULE CODE**

/////////////////////////////// TOP MODULE /////////////////////////////////////////

`timescale 1ns / 1ps

module top(

input wire clk,

input wire rst,

input wire outside\_f1\_up, outside\_f2\_up, outside\_f2\_down, outside\_f3\_up, outside\_f3\_down, outside\_f4\_down, // Outside elevator floor call buttons

input wire [3:0] floor\_request, // Specific floor button (inside elevator), total of 4 floors

input wire open\_request, // Open Door button (inside elevator)

input wire close\_request, // Close Door button (inside elevator)

input wire sos\_request, // SOS button (inside elevator)

input wire door\_jam, // Door jammed/cannot close due to elevator sensing object between door

output wire [3:0] floor\_requested,

output wire speaker,

output wire [7:0] ssg,

output wire [3:0] digit,

output wire RGB

);

wire [4:0] state;

wire [3:0] current\_floor;

wire door\_status;

wire flash, no\_flash;

wire destination\_floor\_reached;

wire enable\_speaker;

wire direction;

elevator\_control my\_elevator( .clk(clk), .rst(rst),

.open\_request(open\_request),

.close\_request(close\_request),

.sos\_request(sos\_request),

.door\_jam(door\_jam),

.floor\_request(floor\_request),

.outside\_f1\_up(outside\_f1\_up), .outside\_f2\_up(outside\_f2\_up), .outside\_f2\_down(outside\_f2\_down), .outside\_f3\_up(outside\_f3\_up), .outside\_f3\_down(outside\_f3\_down), .outside\_f4\_down(outside\_f4\_down),

.state(state),

.direction(direction),

.destination\_floor\_reached(destination\_floor\_reached), .enable\_speaker(enable\_speaker), .floor\_requested(floor\_requested),

.current\_floor(current\_floor), .door\_status(door\_status), .flash(flash), .no\_flash(no\_flash));

//////////////////////

narrator my\_narrator(.clk(clk), .speaker(speaker),

.current\_floor(current\_floor), .door\_status(door\_status), .state(state), .direction(direction),

.destination\_floor\_reached(destination\_floor\_reached), .enable\_speaker(enable\_speaker));

////////////////////////

Seven\_Seg display(

.clk(clk), .flash(flash), .no\_flash(no\_flash), .direction(direction), .door\_status(door\_status), .sos\_request(sos\_request),

.current\_floor(floor\_requested),

.ssg(ssg),

.digit(digit),

.RGB(RGB)

);

endmodule

/////////////////////////// ELEVATOR\_CONTROL MODULE /////////////////////////////////

`timescale 1ns / 1ps

module elevator\_control

(

input wire clk,

input wire rst,

input wire open\_request, // Open Door button (inside elevator)

input wire outside\_f1\_up, outside\_f2\_up, outside\_f2\_down, outside\_f3\_up, outside\_f3\_down, outside\_f4\_down, // Outside elevator floor call buttons

input wire close\_request, // Close Door button (inside elevator)

input wire sos\_request, // SOS button (inside elevator)

input wire door\_jam, // Door jammed/cannot close due to elevator sensing object between door

input wire [3:0] floor\_request, // Specific floor button (inside elevator), total of 4 floors

output reg [3:0] floor\_requested,

output reg [4:0] state,

output reg [3:0] current\_floor,

output reg direction,

output reg destination\_floor\_reached,

output reg door\_status,

output reg flash, no\_flash,

output reg enable\_speaker

);

reg [3:0] destination\_floor;

reg [26:0] base\_timer, base\_waiting\_timer, flash\_counter;

reg [4:0] timer, waiting\_timer;

reg slow\_flash, fast\_flash;

/\* State set up \*/

reg [4:0] next\_state;

reg reset, reset\_waiting;

wire elevator\_called; // Elevator call request of any kind (outside, inside)

assign elevator\_called = (outside\_f1\_up || outside\_f2\_up || outside\_f2\_down || outside\_f3\_up || outside\_f3\_down || outside\_f4\_down || floor\_request);

parameter [4:0] IDLE = 5'd0,

SOS = 5'd1,

OPENED\_DOOR = 5'd2,

CLOSED\_DOOR = 5'd3,

BEGIN\_ELEVATOR\_MOVEMENT = 5'd4,

SLOW\_START = 5'd5,

DOOR\_JAMMED = 5'd6,

WAITING\_AFTER\_OPENING = 5'd7,

OPEN\_REQUEST\_WHILE\_CLOSED = 5'd8,

WAITING\_AFTER\_JAMMED = 5'd9,

FAST\_FIRST\_HALF = 5'd10,

INCOMING\_FLOOR = 5'd11,

SLOW\_END = 5'd12,

ARRIVED = 5'd13,

FAST\_LAST\_HALF = 5'd14;

/\* Reg variables set up \*/

initial begin

current\_floor = 1;

destination\_floor = 1;

state = IDLE;

timer = 0;

base\_timer = 0;

reset = 0;

floor\_requested = 0;

base\_waiting\_timer = 0;

waiting\_timer = 0;

door\_status = 0; //closed

flash = 0;

no\_flash = 1;

slow\_flash = 0;

fast\_flash = 0;

end

/\* Determination of destination floor \*/

always@(posedge clk)

begin

if((floor\_request == 4'b0001) || outside\_f1\_up)

destination\_floor <= 4'b0001;

else if((floor\_request == 4'b0010) || outside\_f2\_up || outside\_f2\_down)

destination\_floor <= 4'b0010;

else if((floor\_request == 4'b0100) || outside\_f3\_up || outside\_f3\_down)

destination\_floor <= 4'b0100;

else if((floor\_request == 4'b1000) || outside\_f4\_down)

destination\_floor <= 4'b1000;

end

/\* Sequential next state logic \*/

always @(posedge clk or posedge rst)

begin

if(rst)

state <= IDLE;

else

state <= next\_state;

end

/\* Timer Initializations \*/

always@(posedge clk)

begin

if(reset)

begin

base\_timer <= 0;

timer <= 0;

end

else

begin

// With this set up, timer ticks roughly every 1 second

if(base\_timer >= 27'd25000000)

begin

timer <= timer + 1;

base\_timer <= 0;

end

else

base\_timer <= base\_timer + 1;

end

end

always@(posedge clk)

begin

if(reset\_waiting)

begin

base\_waiting\_timer <= 0;

waiting\_timer <= 0;

end

else

begin

// With this set up, timer ticks roughly every 1 second

if(base\_waiting\_timer >= 27'd25000000)

begin

waiting\_timer <= waiting\_timer + 1;

base\_waiting\_timer <= 0;

end

else

base\_waiting\_timer <= base\_waiting\_timer + 1;

end

end

always@(posedge clk)

begin

if(slow\_flash)

begin

flash\_counter <= flash\_counter + 1;

if (flash\_counter == 27'd80000000)

begin

flash <= ~flash;

flash\_counter <= 0;

end

end

else if(fast\_flash)

begin

flash\_counter <= flash\_counter + 1;

if (flash\_counter == 27'd33333333)

begin

flash <= ~flash;

flash\_counter <= 0;

end

end

else

flash\_counter <= 0;

end

always@(destination\_floor)begin

if((state==ARRIVED) || (state==IDLE) || (state==OPENED\_DOOR) || (state==SOS)

|| (state==WAITING\_AFTER\_OPENING) || (state==CLOSED\_DOOR) || (state==DOOR\_JAMMED))

floor\_requested = 0;

else

floor\_requested = destination\_floor;

end

/\* Combinational state logic \*/

always@\*

begin

case(state)

IDLE:

if (sos\_request)

next\_state = SOS;

else if (elevator\_called)

next\_state = BEGIN\_ELEVATOR\_MOVEMENT;

else if (open\_request)

next\_state = OPENED\_DOOR;

else begin

next\_state = IDLE;

// floor\_requested = 0;

end

SOS:

if (timer == 15)

next\_state = IDLE;

else

next\_state = SOS;

OPENED\_DOOR:

if (timer == 15)

next\_state = WAITING\_AFTER\_OPENING;

else

next\_state = OPENED\_DOOR;

WAITING\_AFTER\_OPENING:

if (door\_jam)

next\_state = DOOR\_JAMMED;

else if (waiting\_timer == 5 || close\_request) // timer does not work in this case

next\_state = CLOSED\_DOOR;

else

next\_state = WAITING\_AFTER\_OPENING;

CLOSED\_DOOR:

if (open\_request)

next\_state = OPEN\_REQUEST\_WHILE\_CLOSED;

else if (timer == 15)

next\_state = IDLE;

else

next\_state = CLOSED\_DOOR;

OPEN\_REQUEST\_WHILE\_CLOSED:

if(waiting\_timer == 3)

next\_state = OPENED\_DOOR;

else

next\_state = OPEN\_REQUEST\_WHILE\_CLOSED;

DOOR\_JAMMED:

if (!door\_jam || close\_request)

next\_state = WAITING\_AFTER\_OPENING;

else

next\_state = DOOR\_JAMMED;

BEGIN\_ELEVATOR\_MOVEMENT:

if (current\_floor == destination\_floor)

next\_state = OPENED\_DOOR;

else

next\_state = SLOW\_START;

SLOW\_START:

if (timer == 10)

next\_state = FAST\_FIRST\_HALF;

else

next\_state = SLOW\_START;

FAST\_FIRST\_HALF:

if (waiting\_timer == 10)

next\_state = INCOMING\_FLOOR;

else

next\_state = FAST\_FIRST\_HALF;

INCOMING\_FLOOR:

next\_state = FAST\_LAST\_HALF;

FAST\_LAST\_HALF:

if (timer == 10 && (current\_floor == destination\_floor))

next\_state = SLOW\_END;

else if (timer == 10 && (current\_floor != destination\_floor))

next\_state = FAST\_FIRST\_HALF;

else

next\_state = FAST\_LAST\_HALF;

SLOW\_END:

if (waiting\_timer == 10)

next\_state = ARRIVED;

else

next\_state = SLOW\_END;

ARRIVED:

next\_state = OPENED\_DOOR;

default:

next\_state = IDLE;

endcase

end

/\* Outputs \*/

always@(posedge clk)

begin

case(state)

IDLE:

begin

reset <= 1;

reset\_waiting <= 1;

enable\_speaker <= 0;

direction <= 0;

door\_status <= 0;

no\_flash <= 1;

end

SOS:

begin

reset <= 0;

reset\_waiting <= 1;

enable\_speaker <= 1;

end

OPENED\_DOOR:

begin

reset <= 0;

reset\_waiting <= 1;

enable\_speaker <= 1;

door\_status <= 1;

end

WAITING\_AFTER\_OPENING:

begin

reset <= 1;

reset\_waiting <= 0;

enable\_speaker <= 0;

door\_status <= 1;

end

CLOSED\_DOOR:

begin

reset <= 0;

reset\_waiting <= 1;

enable\_speaker <= 1;

door\_status <= 0;

end

OPEN\_REQUEST\_WHILE\_CLOSED:

begin

reset <= 1;

reset\_waiting <= 0;

enable\_speaker <= 0;

door\_status <= 0;

end

DOOR\_JAMMED:

begin

reset <= 0;

reset\_waiting <= 1;

enable\_speaker <= 1;

door\_status <= 1;

end

BEGIN\_ELEVATOR\_MOVEMENT:

begin

if(current\_floor < destination\_floor)

direction <= 1;

else

direction <= 0;

reset <= 1;

reset\_waiting <= 1;

enable\_speaker <= 0;

end

SLOW\_START:

begin

reset <= 0;

reset\_waiting <= 1;

enable\_speaker <= 1;

no\_flash <= 0;

slow\_flash <= 1;

end

FAST\_FIRST\_HALF:

begin

reset <= 1;

reset\_waiting <= 0;

enable\_speaker <= 0;

no\_flash <= 0;

slow\_flash <= 0;

fast\_flash <= 1;

end

INCOMING\_FLOOR:

begin

if (direction)

current\_floor <= (current\_floor << 1);

else

current\_floor <= (current\_floor >> 1);

reset <= 1;

reset\_waiting <= 1;

enable\_speaker <= 0;

end

FAST\_LAST\_HALF:

begin

reset <= 0;

reset\_waiting <= 1;

enable\_speaker <= 1;

no\_flash <= 0;

fast\_flash <= 1;

end

SLOW\_END:

begin

reset <= 1;

reset\_waiting <= 0;

enable\_speaker <= 0;

no\_flash <= 0;

slow\_flash <= 1;

end

ARRIVED:

begin

reset <= 1;

reset\_waiting <= 1;

enable\_speaker <= 0;

no\_flash <= 1;

slow\_flash <= 0;

fast\_flash <= 0;

end

default:

begin

reset <= 1;

reset\_waiting <= 1;

direction <= 0;

enable\_speaker <= 0;

no\_flash <= 1;

door\_status <= 0;

end

endcase

end

endmodule

////////////////////// SEVEN SEGMENT MODULE /////////////////////////////////////////

`timescale 1ns / 1ps

module Seven\_Seg(

input clk, flash, direction, door\_status, sos\_request, no\_flash,

input [3:0] current\_floor, floor\_btn,

output reg [7:0] ssg,

output reg [3:0] digit,

output reg RGB

);

// signal for the anode of the display

wire [1:0]digit\_select;

// signal for floor #

reg [7:0] floor\_ssg;

// signal for direction

reg [7:0] direction\_ssg;

// signal for door status

reg [7:0] door\_ssg\_r, door\_ssg\_l;

// sos RGB

reg [30:0]sos\_counter;

always @(posedge clk) begin

sos\_counter <= sos\_counter + 1;

if(sos\_request)

RGB <= sos\_counter[5];

else

RGB <= 0;

end

//---------------------------------------------------------//

//---------------------------------------------------------//

//floor numbers digit[0](right digit)

parameter [7:0] ssg\_one = 8'b11111001,

ssg\_two = 8'b10100100,

ssg\_three = 8'b10110000,

ssg\_four = 8'b10011001;

//up/down digit[1]

reg [7:0] ssg\_up = 8'b11101010,

ssg\_down = 8'b11010101;

//door left = digit[3], right = digit[2]

// DOOR 1 DOOR2

// ---- ----

// | |

// | |

// ---- ----

parameter [7:0] ssg\_door1 = 8'b11000110,

ssg\_door2 = 8'b11110000;

//---------------------------------------------------------//

//---------------------------------------------------------//

// set up clock dividers and the display paramters

reg [19:0]cnt; //creates~10ms refersh time

//used to iterate the digit to display

assign digit\_select = cnt[19:18];

// flash the up/down

reg [28:0]flasher;

// initialize counters and displays

initial begin

cnt <= 0;

ssg <= 8'b11000000;

digit <= 4'b1111;

flasher <= 0;

end

//---------------------------------------------------------//

//---------------------------------------------------------//

// floor SSG

always@(\*) begin

case(current\_floor)

1: floor\_ssg <= ssg\_one;

2: floor\_ssg <= ssg\_two;

4: floor\_ssg <= ssg\_three;

8: floor\_ssg <= ssg\_four;

default: floor\_ssg <= 8'b11000000;

endcase

end

// direction SSG

always@(\*) begin

case(direction)

0: direction\_ssg <= ssg\_down;

1: direction\_ssg <= ssg\_up;

endcase

end

// door ssg

always @(\*) begin

case(door\_status)

0: begin // closed

door\_ssg\_r <= ssg\_door1;

door\_ssg\_l <= ssg\_door2;

end

1: begin // open

door\_ssg\_r <= ssg\_door2;

door\_ssg\_l <= ssg\_door1;

end

endcase

end

always@(posedge clk) begin

cnt <= cnt+1;

end

always@(\*) begin

case(digit\_select)

0: begin

digit <= 4'b1110;

ssg <= floor\_ssg;

end

1: begin

digit <= 4'b1101;

ssg <= direction\_ssg;

end

2: begin

digit <= 4'b1011;

ssg <= door\_ssg\_r;

end

3: begin

digit <= 4'b0111;

ssg <= door\_ssg\_l;

end

endcase

end

always @(posedge clk)

begin

if(no\_flash)

begin

ssg\_up <= 8'b10111111;

ssg\_down <= 8'b10111111;

end

else

begin

if(flash)

begin

ssg\_up <= 8'b11101010;

ssg\_down <= 8'b11010101;

end

else

begin

ssg\_up <= 8'b11011100;

ssg\_down <= 8'b11100011;

end

end

end

endmodule

/////////////////////////// NARRATOR MODULE /////////////////////////////////////////

`timescale 1 ns / 1 ps

module narrator (

input wire clk,

input wire enable\_speaker,

input wire [4:0] state,

input [3:0] current\_floor,

input requested\_floor\_reached,

input direction,

input door\_status,

input destination\_floor\_reached,

output wire speaker

);

// Create a test circuit to exercise the chatter

// module, rather than using switches and a

// button.

reg [6:0] counter = 0;

reg [5:0] data;

wire write;

wire busy;

parameter [3:0] FLOOR\_ONE = 4'b0001,

FLOOR\_TWO = 4'b0010,

FLOOR\_THREE = 4'b0100,

FLOOR\_FOUR = 4'b1000;

parameter [4:0] IDLE = 5'd0,

SOS = 5'd1,

OPENED\_DOOR = 5'd2,

CLOSED\_DOOR = 5'd3,

BEGIN\_ELEVATOR\_MOVEMENT = 5'd4,

SLOW\_START = 5'd5,

DOOR\_JAMMED = 5'd6,

WAITING\_AFTER\_OPENING = 5'd7,

OPEN\_REQUEST\_WHILE\_CLOSED = 5'd8,

WAITING\_AFTER\_JAMMED = 5'd9,

FAST\_FIRST\_HALF = 5'd10,

INCOMING\_FLOOR = 5'd11,

SLOW\_END = 5'd12,

ARRIVED = 5'd13,

FAST\_LAST\_HALF = 5'd14;

always @(posedge clk)

begin

if (!busy)

counter <= counter + 1;

if (!enable\_speaker)

counter <= 0;

end

always @\*

begin

case(state)

IDLE:

case (counter[6:2]) // Silence

0: data = 6'h04;

default: data = 6'h04;

endcase

SOS:

case(counter[6:2]) // SOS

0: data = 6'h07;

1: data = 6'h37;

2: data = 6'h03;

3: data = 6'h35;

4: data = 6'h03;

5: data = 6'h07;

6: data = 6'h37;

7: data = 6'h04;

default: data = 6'h04;

endcase

OPENED\_DOOR:

case (counter[6:2]) // Opening Door

0: data = 6'h35;

1: data = 6'h09;

2: data = 6'h07;

3: data = 6'h0b;

4: data = 6'h13;

5: data = 6'h0b;

6: data = 6'h24;

7: data = 6'h21;

8: data = 6'h3a;

9: data = 6'h04;

default: data = 6'h04;

endcase

CLOSED\_DOOR:

case(counter[6:2]) // Closing Door

0: data = 6'h08;

1: data = 6'h2d;

2: data = 6'h35;

3: data = 6'h2b;

4: data = 6'h13;

5: data = 6'h0b;

6: data = 6'h24;

7: data = 6'h21;

8: data = 6'h3a;

default: data = 6'h04;

endcase

DOOR\_JAMMED:

case(counter[6:2]) //Door jammed

0: data = 6'h21;

1: data = 6'h3a;

2: data = 6'h04;

3: data = 6'h0a;

4: data = 6'h18;

5: data = 6'h11;

6: data = 6'h15;

7: data = 6'h15;

default: data = 6'h04;

endcase

SLOW\_START:

if(direction)

case(counter[6:2]) // Going up

0: data = 6'h3d;

1: data = 6'h35;

2: data = 6'h13;

3: data = 6'h0b;

4: data = 6'h24;

5: data = 6'h03;

6: data = 6'h24;

7: data = 6'h35;

8: data = 6'h09;

default: data = 6'h04;

endcase

else

case(counter[6:2]) // Going down

0: data = 6'h3d;

1: data = 6'h35;

2: data = 6'h13;

3: data = 6'h0b;

4: data = 6'h24;

5: data = 6'h03;

6: data = 6'h21;

7: data = 6'h20;

8: data = 6'h0b;

default: data = 6'h04;

endcase

FAST\_LAST\_HALF:

case(current\_floor)

FLOOR\_ONE:

case(counter[6:2]) // Floor 1

0: data =6'h28;

1: data =6'h2d;

2: data =6'h3a;

3: data = 6'h03;

4: data = 6'h2e;

5: data = 6'h0f;

6: data = 6'h0f;

7: data = 6'h0b;

default: data = 6'h04;

endcase

FLOOR\_TWO:

case(counter[6:2]) // Floor 2

0: data =6'h28;

1: data =6'h2d;

2: data =6'h3a;

3: data = 6'h03;

4: data = 6'h0d;

5: data = 6'h1f;

default: data = 6'h04;

endcase

FLOOR\_THREE:

case(counter[6:2]) // Floor 3

0: data =6'h28;

1: data =6'h2d;

2: data =6'h3a;

3: data = 6'h03;

4: data = 6'h1d;

5: data = 6'h0e;

6: data = 6'h13;

default: data = 6'h04;

endcase

FLOOR\_FOUR:

case(counter[6:2]) // Floor 4

0: data =6'h28;

1: data =6'h2d;

2: data =6'h3a;

3: data = 6'h03;

4: data = 6'h28;

5: data = 6'h28;

6: data = 6'h23;

7: data = 6'h3a;

default: data = 6'h04;

endcase

endcase

default:

case (counter[6:2]) // Silence

0: data = 6'h04;

default: data = 6'h04;

endcase

endcase

end

assign write = (counter[1:0] == 2'b00);

// Instantiate the chatter module, which is

// driven by the test circuit.

chatter chatter\_inst (

.data(data),

.write(write),

.busy(busy),

.clk(clk),

.speaker(speaker)

);

endmodule

/////////////////////////////////////////////////////////////////////////////////////

/////////////////////////////TOP MODULE TESTBENCH////////////////////////////////////

`timescale 1 ns / 1 ps

module testbench;

// Inputs

reg clk, rst;

reg outside\_f1\_up, outside\_f2\_up, outside\_f2\_down, outside\_f3\_up, outside\_f3\_down, outside\_f4\_down;

reg [3:0] floor\_request;

reg open\_request, close\_request, sos\_request, door\_jam;

// Outputs

wire [3:0] floor\_requested;

wire speaker;

wire [7:0] ssg;

wire [3:0] digit;

wire RGB;

wire [3:0] current\_floor;

// Instantiate the Unit Under Test (UUT)

top uut (

.clk(clk),

.rst(rst),

.outside\_f1\_up(outside\_f1\_up), .outside\_f2\_up(outside\_f2\_up), .outside\_f2\_down(outside\_f2\_down),

.outside\_f3\_up(outside\_f3\_up), .outside\_f3\_down(outside\_f3\_down), .outside\_f4\_down(outside\_f4\_down),

.floor\_request(floor\_request),

.open\_request(open\_request), .close\_request(close\_request), .sos\_request(sos\_request), .door\_jam(door\_jam),

.floor\_requested(floor\_requested),

.speaker(speaker),

.ssg(ssg),

.digit(digit),

.RGB(RGB)

);

// Clock generation

always #5 clk = ~clk; // Generate a clock with a period of 10ns

// Initial block for test cases

initial begin

// Initialize Inputs

clk = 0;

rst = 1;

open\_request = 0;

close\_request = 0;

sos\_request = 0;

door\_jam = 0;

floor\_request = 4'b0000;

outside\_f1\_up = 0; outside\_f2\_up = 0; outside\_f2\_down = 0; outside\_f3\_up = 0; outside\_f3\_down = 0; outside\_f4\_down = 0;

// Wait for 100 ns for global reset to finish

#50;

rst = 0;

// Add test cases here

// Example: Request floor 3 from inside

floor\_request = 4'b0100;

#50; // wait some time

floor\_request = 4'b0000;

// Example: Request floor 2 from inside

floor\_request = 4'b0010;

#50; // wait some time

floor\_request = 4'b0000;

// Example: Request floor 1 from inside

floor\_request = 4'b0001;

#50; // wait some time

floor\_request = 4'b0000;

// Example: Request floor 4 from inside

floor\_request = 4'b1000;

#50; // wait some time

floor\_request = 4'b0000;

// Request from outside, floor 2 up

outside\_f2\_up = 1;

#50;

outside\_f2\_up = 0;

// Simulate door jam

door\_jam = 1;

#50;

door\_jam = 0;

// Assert SOS

#10;

sos\_request = 1;

#10;

sos\_request = 0;

// Finish simulation

#50000;

$finish;

end

endmodule

/////////////////////////ELEVATOR CONTROL TESTBENCH//////////////////////////////

module testbench;

// Inputs

reg clk;

reg rst;

reg open\_request;

reg outside\_f1\_up, outside\_f2\_up, outside\_f2\_down, outside\_f3\_up, outside\_f3\_down, outside\_f4\_down;

reg close\_request;

reg sos\_request;

reg door\_jam;

reg [3:0] floor\_request;

// Outputs

wire [3:0] floor\_requested;

wire [3:0] state;

wire [3:0] current\_floor;

wire direction;

wire door\_status;

wire flash, no\_flash;

wire enable\_speaker;

// Instantiate the elevator control module

elevator\_control uut (

.clk(clk),

.rst(rst),

.open\_request(open\_request),

.outside\_f1\_up(outside\_f1\_up),

.outside\_f2\_up(outside\_f2\_up),

.outside\_f2\_down(outside\_f2\_down),

.outside\_f3\_up(outside\_f3\_up),

.outside\_f3\_down(outside\_f3\_down),

.outside\_f4\_down(outside\_f4\_down),

.close\_request(close\_request),

.sos\_request(sos\_request),

.door\_jam(door\_jam),

.floor\_request(floor\_request),

.floor\_requested(floor\_requested),

.state(state),

.current\_floor(current\_floor),

.direction(direction),

.door\_status(door\_status),

.flash(flash),

.no\_flash(no\_flash),

.enable\_speaker(enable\_speaker)

);

// Clock generation

initial begin

clk = 0;

forever #5 clk = ~clk; // 20ns period => 50 MHz clock

end

// Test stimulus

initial begin

// Initialize all inputs

rst = 0;

open\_request = 0;

outside\_f1\_up = 0;

outside\_f2\_up = 0;

outside\_f2\_down = 0;

outside\_f3\_up = 0;

outside\_f3\_down = 0;

outside\_f4\_down = 0;

close\_request = 0;

sos\_request = 0;

door\_jam = 0;

floor\_request = 0;

// Apply Reset

rst = 1;

#10;

rst = 0;

#10;

// Case 1: Request elevator to floor 2

outside\_f2\_up = 1;

#10;

outside\_f2\_up = 0;

#900;

// Case 2: Request elevator to floor 3 with a door jam

// outside\_f3\_up = 1;

// #10;

// outside\_f3\_up = 0;

floor\_request = 4'b0100;

#10;

floor\_request = 0;

#900;

// Simulate door jam

door\_jam = 1;

$display("Time = %0dns: Simulating door jam", $time);

#50;

door\_jam = 0;

end

endmodule

/////////////////////////SEVEN SEGMENT DISPLAY TESTBENCH/////////////////////////////

module testbench;

reg clk;

reg flash;

reg direction;

reg door\_status;

reg sos\_request;

reg no\_flash;

reg [3:0] current\_floor;

//reg [3:0] floor\_requested;

wire [7:0] ssg;

wire [3:0] digit;

wire RGB;

// Instantiate the Design Under Test (DUT)

Seven\_Seg dut (

.clk(clk),

.flash(flash),

.direction(direction),

.door\_status(door\_status),

.sos\_request(sos\_request),

.no\_flash(no\_flash),

//.current\_floor(floor\_requested),

.current\_floor(current\_floor),

.ssg(ssg),

.digit(digit),

.RGB(RGB)

);

// Clock Generation

initial begin

clk = 0;

end

always #5 clk = ~clk; // 100 MHz clock, clock period 10ns

// Stimulus

initial begin

// Initialize inputs

flash = 0;

direction = 0;

door\_status = 0;

sos\_request = 0;

no\_flash = 0;

current\_floor = 4'b0000;

// Reset signals

#20;

current\_floor = 4'b0001;

#20;

current\_floor = 4'b0010;

#20;

current\_floor = 4'b0100;

#20;

current\_floor = 4'b1000;

#20;

current\_floor = 4'b0001;

// Direction up and down

direction = 1;

#50;

direction = 0;

// Flashing test

flash = 1;

#50;

flash = 0;

// Door open/close

door\_status = 1;

#100;

door\_status = 0;

// SOS Request

sos\_request = 1;

#100;

sos\_request = 0;

// No Flash Mode

no\_flash = 1;

#50;

no\_flash = 0;

// End simulation

#5000;

$finish;

end

endmodule

////////////////////////////// NARRATOR TESTBENCH //////////////////////////////////

module testbench;

// Generate a free running 100 MHz clock

// signal to mimic what is on the board

// provided for prototyping.

reg clk;

always

begin

clk = 1'b0;

#5;

clk = 1'b1;

#5;

end

wire speaker;

initial

begin

$display("If simulation ends before the testbench");

$display("completes, use the menu option to run all.");

#40000000; // allow it to run

$display("Simulation is over, check the waveforms.");

$stop;

end

narrator my\_narrator (

.clk(clk),

.speaker(speaker)

);

endmodule