Final Report

[Elevator Controller by FPGA]

Student ID 20191337

Name 강성준

1. Project Overview

Project Overview

This project aims to design and implement an **elevator controller** using **Verilog HDL** on an **FPGA board**. The system leverages the FPGA's built-in components, such as **7-segment displays**, **LED indicators**, and **switch functionalities**, to control and simulate real-world elevator operations.

The controller ensures that elevator tasks—such as handling multiple floor requests, prioritizing the nearest floor, monitoring weight limits, and managing emergency stops—are executed efficiently and accurately. The end goal is to demonstrate a functional and reliable elevator control system that can be further expanded for practical applications.

Background and Necessity

As of **2023**, South Korea has approximately **840,000 elevators** installed nationwide, reflecting an exceptionally high demand for elevator systems. This demand is primarily driven by the prevalence of apartment-style residential buildings in urban areas, where elevators are an essential part of daily life.

However, with increasing population density and aging infrastructure, there is a growing need for **reliable, efficient, and cost-effective elevator controllers**. Existing systems often face challenges such as:

- High maintenance costs
- Inefficient floor selection algorithms
- Limited integration of modern control logic

This project addresses these challenges by developing a programmable elevator controller on an FPGA, capable of:

- Implementing optimized control logic to improve responsiveness and efficiency
- Simulating real-world elevator features like **weight limit management**, **emergency handling**, and **multi-floor prioritization**

Such systems have significant potential to revolutionize the elevator market, enabling new opportunities for innovation and cost reduction.

Objective

The primary goals of this project are as follows:

1. **Develop a functional elevator control system** on an FPGA board using Verilog HDL.
2. Implement essential elevator functionalities, including:

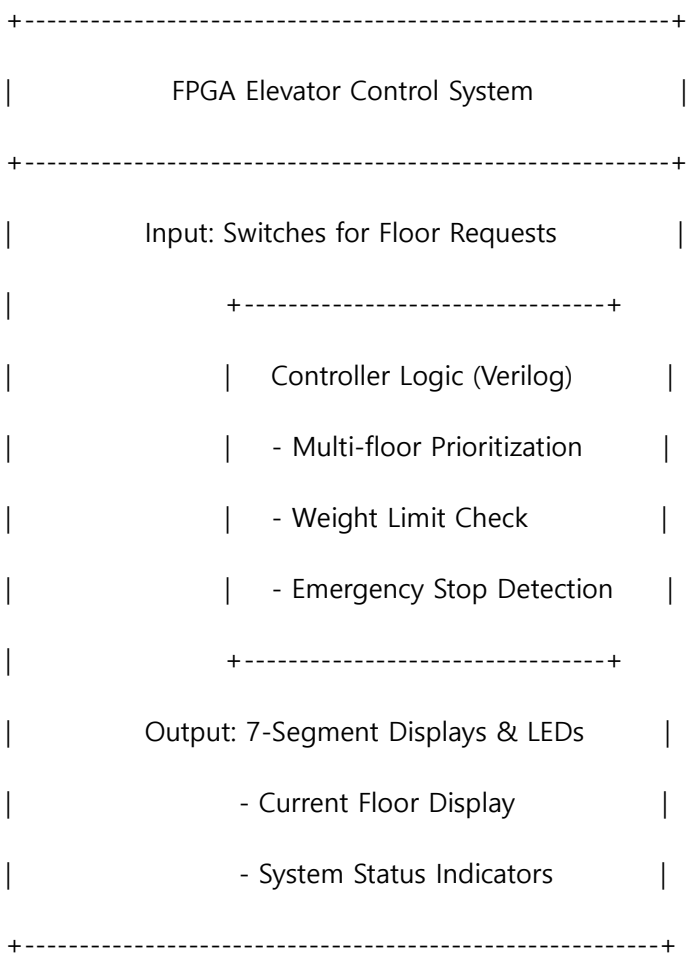
- Multi-floor requests with **priority for the nearest floor**
- **Weight limit detection** to ensure safe operation
- **Emergency stop handling** to manage critical situations

3. Utilize FPGA features such as **7-segment displays**, **LEDs**, and **switch inputs** to simulate and display elevator operations in real-time.

4. Create a system that can be further refined and potentially **integrated into commercial elevator controllers**, contributing to market innovation.

Diagram

Below is a conceptual system overview diagram illustrating the project components and interactions:



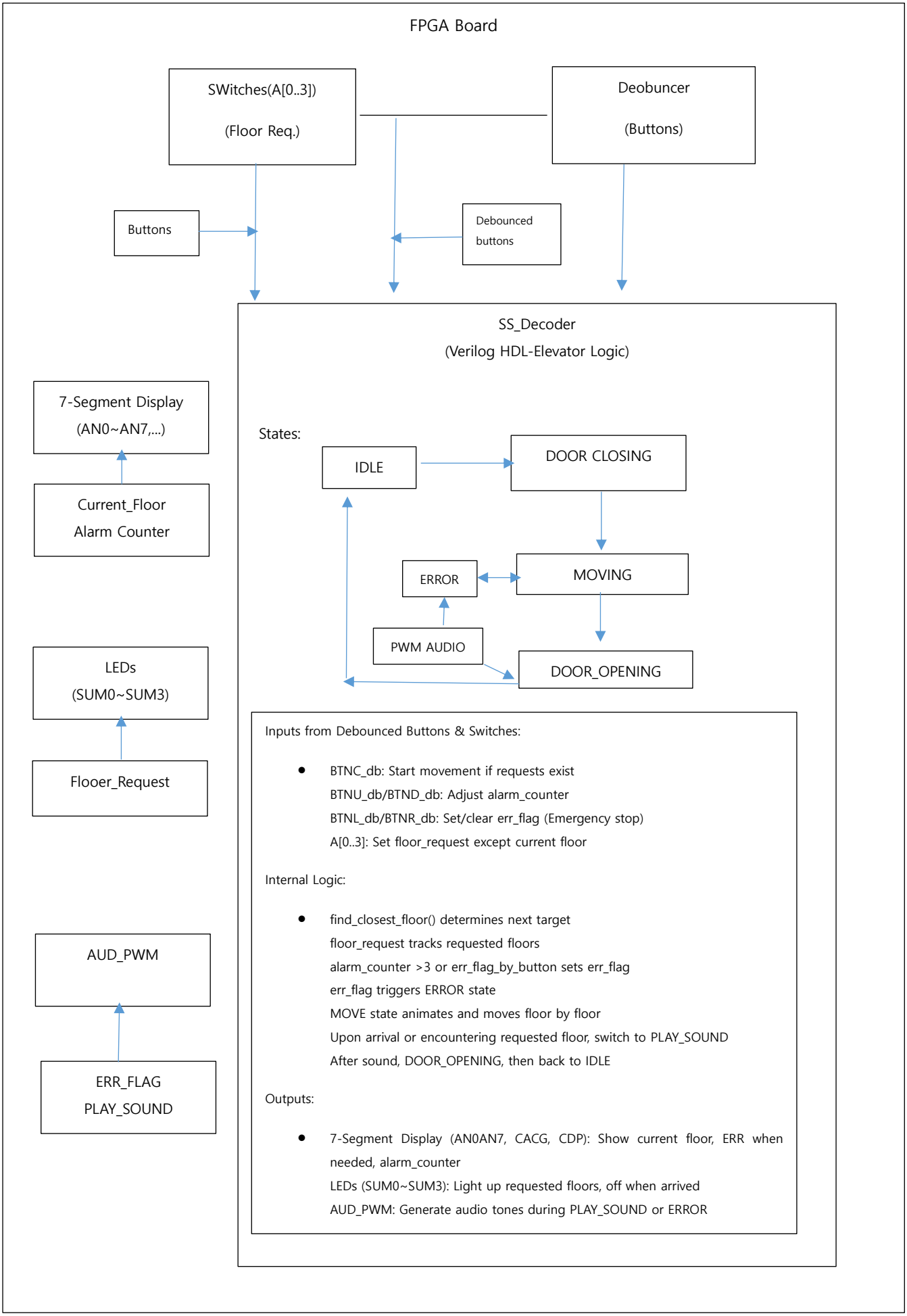
GitHub Repository

For further details and access to the source code, please visit:
https://github.com/jun03091105/2024_2_verilog_20191337.git

---

2. Technology and Design

System Architecture Diagram



In this diagram, the FPGA reads input signals from switches and buttons, processes them through the SS_Decoder module (implemented in Verilog HDL), and controls the outputs such as the 7-segment display, LEDs, and audio PWM signal.

Technical Specifications

- **Verilog HDL**:
  Verilog HDL is used as the hardware description language to implement digital logic circuits at a high level of abstraction. By writing code instead of manually wiring logic gates, the design becomes more flexible, scalable, and easier to modify. Verilog code is synthesized into the FPGA's programmable logic resources, enabling custom hardware functionality.
- **FPGA Internal Features**:
  - **7-Segment Display**:
    The FPGA drives a common anode 7-segment display using a scanning method. Although only one digit is lit at a time, the FPGA switches rapidly between them, creating the illusion of all digits being simultaneously displayed. The segments (CACG, CDP) are activated by driving them low, and the respective digit (AN0AN7) is enabled by setting its corresponding line low.

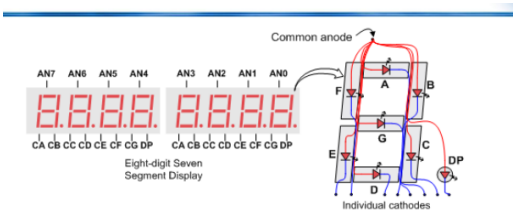
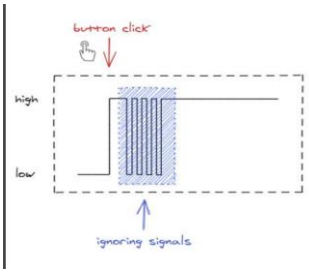


Figure 18. Common anode circuit node.

  - **Switches**:
    On-board switches are used to provide input signals that control LED patterns, determine destination floors, or adjust alarm settings. The FPGA reads these switch inputs directly.
  - **Audio (Mono) via PWM**:
    By using PWM signals, the FPGA generates simple tones or audio signals. In this project, it is used to provide auditory feedback such as an elevator arrival chime or an error tone.
  - **Buttons**:
    The FPGA utilizes on-board push buttons to start elevator movement, trigger emergency stops, or adjust counters. To ensure stable and reliable button input signals, a debouncer module is employed, filtering out mechanical switch noise and delivering a clean input signal to the logic.



- **LED Indicators**:
  LEDs are used to indicate requested floors. When a floor is requested, the corresponding LED is lit. Upon arrival at that floor, the LED is turned off, providing real-time feedback of the elevator state to the user.

- **Development Tools and Platform**:
  - The project is implemented on a standard FPGA development board that includes built-in switches, buttons, LEDs, 7-segment displays, and an audio output pin.
  - Synthesis and implementation are performed using standard FPGA toolchains (e.g., Vivado, Quartus) that convert Verilog HDL into a bitstream for FPGA configuration.
  - The .xdc (Xilinx Design Constraints) file maps logical signals to physical FPGA pins.

Development Process

- **Design Methodology**:
  A top-down approach was used, starting with the high-level functional requirements of the elevator system. Verilog modules were then created for individual tasks (e.g., debouncing, display control, request handling).

- **Incremental Testing**:
  Each module (e.g., debouncer) was tested independently before integration. Once the modules were verified, the entire system was combined and tested on the FPGA board.

- **Iterative Refinement**:
  Adjustments to timing parameters (e.g., debounce delays, display scanning rate) and logic conditions (e.g., error states, requests) were made incrementally until the desired behavior was achieved.

Module Description

---

1. Overall System Operation

The FPGA-based elevator control system is designed using Verilog HDL and implemented on an FPGA development board. It simulates real elevator operations such as handling floor requests, moving between floors, managing door control, and responding to emergency or overload conditions. The system interacts with switches, buttons, 7-segment displays, LEDs, and audio outputs to deliver a realistic simulation.

The system is divided into three primary components:

1. **SS_Decoder Module**: Implements the main elevator control logic.
2. **Debouncer Module**: Ensures clean and stable button inputs.
3. **XDC File**: Maps the logical Verilog signals to the physical FPGA board pins.

2. Module Descriptions and Code Analysis

2.1 SS_Decoder Module

Purpose:
The SS_Decoder module acts as the core of the system, managing all elevator functionalities, including state transitions, movement control, error detection, and display updates.

Key Functionalities and Code Segments:

1. **Floor Request Handling**
   When the elevator is idle, it checks for floor requests from switches (A[0..3]) and ignores the current floor. The floor_request register keeps track of requested floors.

Code Example:

```
if (A[0] && current_floor != 4'd1) floor_request[0] <= 1'b1;

if (A[1] && current_floor != 4'd2) floor_request[1] <= 1'b1;

if (A[2] && current_floor != 4'd3) floor_request[2] <= 1'b1;

if (A[3] && current_floor != 4'd4) floor_request[3] <= 1'b1;
```

Explanation:

- If a switch is activated, the corresponding floor is marked as requested.
- The floor_request array ensures that the current floor is not redundantly requested.

---

2. **State Machine Control**
   The system transitions through states such as IDLE, DOOR_CLOSING, MOVING, DOOR_OPENING, PLAY_SOUND, and ERROR.

State Diagram:

IDLE → DOOR_CLOSING → MOVING → PLAY_SOUND → DOOR_OPENING → IDLE

↓

ERROR

Code Example for State Transitions:

```
if (BTNC_db && floor_request != 4'b0000 && goal_floor != current_floor)

    elevator_state <= STATE_DOOR_CLOSING;
```

Explanation:

- The start button (BTNC_db) triggers the elevator to close the door and move if there are pending

requests.

- The state machine ensures smooth transitions between door operations, movement, and stopping.

---

3. **Movement Logic**
   The elevator moves floor by floor, checking if it has reached the requested floor. If so, it stops and plays an arrival sound.

Code Example:

```
if (move_counter >= 200_000_000) begin
    move_counter <= 0;
    if (current_floor < goal_floor)
        current_floor <= current_floor + 1;
    else
        current_floor <= current_floor - 1;

    if (floor_request[current_floor-1]) begin
        floor_request[current_floor-1] <= 0;
        elevator_state <= STATE_PLAY_SOUND;
    end
end
```

Explanation:

- The elevator moves every **2 seconds** based on move_counter.
- Upon reaching a requested floor, it stops, clears the floor request, and transitions to the PLAY_SOUND state.

---

4. **Error Handling**
   The elevator enters an ERROR state if the alarm counter exceeds a threshold or an emergency button is pressed.

Code Example:

```
if (err_flag_by_button || (alarm_counter > 4'd3))
    err_flag <= 1;
```

```
if (err_flag) begin
    prev_elevator_state <= elevator_state;
    elevator_state <= STATE_ERROR;
end
```

Explanation:

- An emergency condition sets err_flag and forces the system into STATE_ERROR.
- In this state, all movement stops, and "Err" is displayed on the 7-segment display.

---

5. **Display and LEDs**
   The 7-segment display shows the current floor, alarm counter, or error messages, while LEDs indicate requested floors.

Code Example for Display Control:

```
case (current_floor)
    4'd1: display_data = 8'b10011111; // Display '1'
    4'd2: display_data = 8'b00100101; // Display '2'
    ...
endcase
```

Explanation:

- The display data changes dynamically based on the current floor or error state.
- LEDs light up for requested floors and turn off upon servicing.

2.2 Debouncer Module

Purpose:
The debouncer module stabilizes noisy button inputs by ensuring that a signal remains stable for a set time before registering a change.

Key Code Snippet:

```
always @(posedge clk) begin
    if (I == Iv) begin
        if (cnt == 20'd999_999)
```

```
            O <= I;     // Latch output after ~10ms stable input
        else
            cnt <= cnt + 1;
    end else begin
        cnt <= 0;
        Iv <= I;
    end
end
```

Explanation:

- If an input (I) remains stable for ~10ms, it is considered valid, and output (O) is updated.
- This prevents false triggers caused by mechanical switch bouncing.

2.3 XDC File

Purpose:
The XDC file maps logical signals in the Verilog code to the physical pins on the FPGA board.

Sample XDC Entries:

```
## Clock Signal
set_property -dict { PACKAGE_PIN E3 IOSTANDARD LVCMOS33 } [get_ports { clock }];
create_clock -name clk_pin -period 10.00 [get_ports { clock }];

## Switches for Floor Requests
set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [get_ports { A[0] }];
set_property -dict { PACKAGE_PIN L16 IOSTANDARD LVCMOS33 } [get_ports { A[1] }];

## 7-Segment Display Pins
set_property -dict { PACKAGE_PIN H17 IOSTANDARD LVCMOS33 } [get_ports { AN0 }];
set_property -dict { PACKAGE_PIN K15 IOSTANDARD LVCMOS33 } [get_ports { AN1 }];
```

Explanation:

- The XDC file ensures correct connections between logical signals (A[0], clock, etc.) and the FPGA hardware

pins.

- It also defines voltage standards such as LVCMOS33 for compatibility.

---

3. Overall Summary

The FPGA-based elevator system integrates three main modules to simulate real-world elevator behavior:

1. **Debouncer Module**: Provides stable button inputs by filtering noise.
2. **SS_Decoder Module**: Implements the main elevator logic, including request handling, state transitions, movement control, and error management.
3. **XDC File**: Maps all logical signals to physical FPGA board resources.

Operational Flow:

1. The system starts in the IDLE state, monitoring for floor requests and button presses.
2. Upon receiving a request and pressing the start button, the elevator transitions to DOOR_CLOSING and begins moving.
3. It moves floor-by-floor, stops at requested floors, and plays arrival sounds.
4. Errors (e.g., overload or emergency stop) are detected, halting the elevator and displaying "Err" until cleared.
5. The system updates the 7-segment display, LEDs, and audio signals to reflect real-time status.

Note: For full implementation and detailed code, refer to the attached files (or git-hub URL above).

In summary, the technology and design of the project revolve around using Verilog HDL to implement custom elevator logic on an FPGA platform. The system leverages on-board peripherals like switches, buttons, 7-segment displays, LEDs, and audio output, integrated through careful module design, a defined development process, and adherence to standard FPGA workflows.

3. Achievements and Results

3. Achievements and Results

Test Results

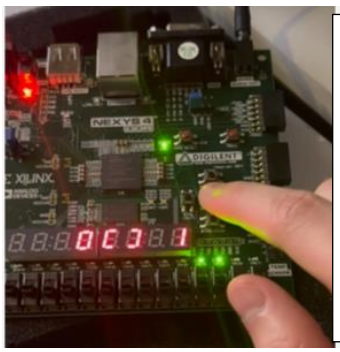
The main objectives of this project were to implement and validate the following functionalities:

- **Elevator Floor Selection Using Switches**: Simulate floor requests via switches, with corresponding LEDs indicating requested floors.
- **Elevator Status Display Using LEDs and 7-Segment Displays**: Display elevator status such as current floor, moving direction (up or down), and door open/close status.
- **Weight Simulation Using Buttons**: Represent the number of passengers or weight using buttons, simulating weight restrictions, with error handling for overload.
- **Emergency Stop Functionality**: Enable an emergency stop button to immediately halt the elevator and signal an error visually and audibly.

Operational Results

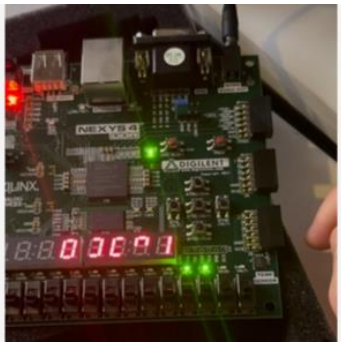
The project successfully demonstrated all target functionalities. Detailed results are outlined below:

1. **Floor Selection and Indication**



When a floor switch is activated, the corresponding floor LED lights up, signaling a valid floor request. The current floor, however, is ignored to prevent redundant requests. This ensures efficient floor handling and avoids unnecessary stops.

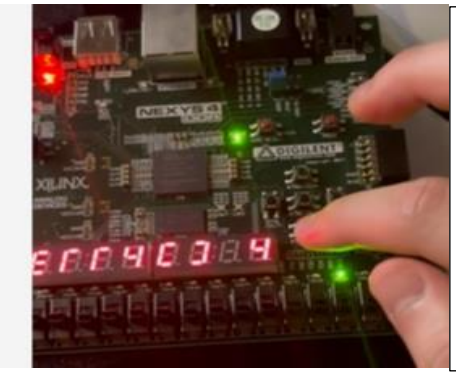
2. **Elevator Movement (Up/Down)**



Upon pressing the start button, the elevator begins moving. The system displays the current floor, direction (up or down), and door status (open/close) on the 7-segment display.

- **Simultaneous Requests**: If multiple floor requests are made, the elevator prioritizes the nearest floor and sequentially handles remaining requests.
- **Arrival Indication**: A PWM-generated audio signal is played upon reaching a floor to simulate an arrival chime.

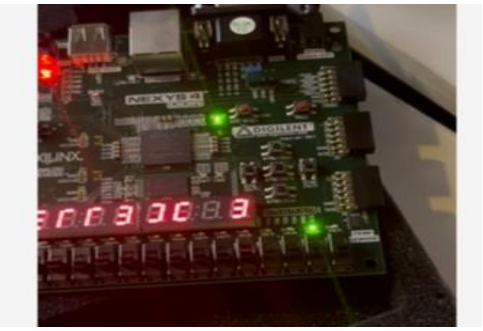
3. **Weight Limit Simulation**



Although the FPGA board lacks an actual weight sensor, the system uses a button to simulate passenger weight.

- If the simulated weight exceeds the limit (e.g., 4 passengers), the 7-segment display shows an error (ERR), and the elevator halts.
- Normal operation resumes once the weight is reduced below the limit.

4. **Emergency Stop Functionality**



During operation, pressing the emergency stop button halts the elevator immediately, and the system enters an error state.

- The 7-segment display shows ERR, and the system emits an error sound via the PWM audio output.
- After releasing the emergency stop, the elevator resumes its operation from the last interrupted state.

Comparative Analysis

The implemented system meets the fundamental objectives of an elevator controller, offering efficient and reliable operation. However, the current design has some limitations compared to commercial elevator controllers:

- **Sensor Integration**: Real-world systems use advanced sensors such as weight sensors, IR or laser sensors for passenger detection, which are absent in this FPGA simulation.

- **Simultaneous Requests**: While the system handles basic multi-floor requests effectively, more complex scenarios (e.g., simultaneous requests in large buildings like shopping malls) may require optimized algorithms for better efficiency.

Despite these limitations, the system performs exceptionally well for its scope, demonstrating high accuracy in request handling, button responsiveness, and timing consistency.

Performance Summary

The project successfully met all predefined goals, validating the elevator functionalities on the FPGA platform:

- The system accurately handles floor requests, manages sequential movements, and displays real-time statuses via LEDs and 7-segment displays.
- Weight and emergency stop simulations function reliably, halting operations when required and resuming seamlessly.
- Button and switch inputs are processed rapidly, with minimal delays, ensuring smooth and responsive operations.

Demonstration: For detailed demonstrations of the project, refer to the attached video or the following link:

https://youtu.be/ID3XCSdN02A?list=LL

Overall, this project demonstrates a robust FPGA-based elevator control system that integrates fundamental functionalities with a modular design. While further enhancements like sensor integration could expand its applicability, the current implementation effectively simulates elevator behavior within the given scope.

4. Conclusion & Applicability

4. Conclusion & Applicability

Conclusion

This project successfully demonstrated a functional FPGA-based elevator control system capable of simulating essential elevator operations such as floor requests, movement between floors, weight limitations, and emergency handling. While the current implementation is a simulation using switches, buttons, LEDs, and 7-segment displays, it highlights the potential for further development into a practical elevator controller.

By integrating additional hardware components such as motors and sensors (e.g., weight sensors, infrared sensors), the system can evolve into a fully functional elevator controller capable of operating in real-world

scenarios. The modular design and reliable performance in handling basic elevator operations lay a strong foundation for future expansions.

Future Plans

To expand the scope and functionality of this project, the following enhancements are planned:

1. **Integration with Motors**: Adding motor control to enable physical movement of an elevator cabin, synchronizing the motor with floor selection and stop signals.
2. **Sensor Integration**: Incorporating weight sensors, infrared passenger detection, or laser sensors to monitor real-time conditions inside the elevator and improve safety.
3. **Algorithm Optimization**: Refining the elevator scheduling algorithm to handle high-traffic scenarios, such as in shopping malls or office buildings, where simultaneous and frequent requests occur.
4. **Enhanced Display Systems**: Expanding the 7-segment display to include more advanced displays or touch interfaces for user interaction and additional status updates.

Applicability

The system's simplicity and modularity make it suitable for specific use cases where basic elevator functionalities are sufficient. Potential applications include:

1. **Residential Elevators**: Small apartment buildings or villas with lower traffic demands can benefit from this system due to its reliability and cost-effectiveness.
2. **Freight Elevators**: Warehouses or industrial sites requiring basic floor navigation and load monitoring without complex scheduling can utilize this system effectively.
3. **Educational Purposes**: This project can serve as a learning tool for students and professionals in the fields of digital design, embedded systems, and FPGA programming.

While the current design is not ideal for high-traffic environments such as shopping malls or commercial complexes due to its simple scheduling logic, future algorithmic improvements could extend its applicability to such settings.

In summary, this project provides a solid framework for a reliable and efficient elevator control system. With further enhancements in hardware and algorithm design, it has the potential to evolve into a versatile and practical solution for residential and industrial elevator applications.