Muhammad Umar
Computer Systems Engineering
NED University of Engineering & Technology Karachi

Design Project FPGA Controller Design for an Elevator

Aim and Objectives

The primary objective of this project is to gain practical experience in the design and implementation of sequential systems, which were covered in the digital logic design course. The aim of the project is to deepen our understanding of fundamental digital systems and to develop the ability to design them according to given requirements and constraints.

To achieve this aim, we will apply the concepts of digital logic design to demonstrate the operation of constrained sequential systems using Verilog HDL. We will also implement our designs on an FPGA board and test them in a real-world environment. By completing this project, we will not only gain hands-on experience in designing and implementing digital systems, but also enhance our problem-solving and critical thinking skills. Additionally, we will develop a deeper understanding of the restrictions and limitations of sequential circuits, which will be useful in future projects and career endeavors.

Introduction

Elevators are an essential part of modern-day buildings, enabling easy and efficient transportation between floors. Digital controllers play a crucial role in the safe and reliable operation of elevators. This project aims to design a digital controller for an elevator system that meets specific requirements, including floor selection, door operation, motor control, overload protection, and response to fire alarms.

Finite State Machine (FSM)

Finite State Machines (FSMs) are widely used in digital system design to model and control the behavior of a system. There are two main types of FSMs: **Mealy** and **Moore** machines.

A **Mealy** machine's output depends on both the present state and input signals. On the other hand, a **Moore** machine's output depends only on the present state of the system, regardless of any input signals.

In our project, we will be using a Moore machine to design the digital controller for the elevator. This approach is more commonly used in practical digital systems due to its increased reliability and resilience to input glitches. Since the output of the elevator controller depends only on the current state of the system, it will not be affected by any input glitches, making it a more robust design.

Using a Moore machine for our project will also require us to carefully consider the states of the elevator controller and how they relate to the inputs and outputs of the system. By designing the elevator controller in this way, we can ensure that it operates correctly and efficiently, while also accounting for any potential issues or malfunctions.

Equipment Used for Elevator design

- **The Artix-7 FPGA board (Nexys A7)**

  The Artix-7 FPGA board (Nexys A7) is a hardware platform used for implementing digital circuits and designs. It contains the Artix-7 FPGA chip which provides the programmable logic for the implementation of various digital circuits. The Nexys A7 board also includes many peripherals, such as switches, buttons, LEDs, VGA, USB, Ethernet, and more, which can be used for interfacing and communication with other devices.

- **Xilinx Vivado**

  Xilinx Vivado 2020.2 is a software used for designing and implementing digital circuits on the Artix-7 FPGA board. It includes a wide range of tools and features for designing and testing digital circuits, such as block design, simulation, synthesis, implementation, and more. Vivado also provides a user-friendly interface for programming the FPGA chip and configuring the board's peripherals.

Design Method

1. Define the system requirements: In this step, we identified the requirements of the elevator control system, such as the ability to control the elevator's movement between different floors, monitor the elevator's weight capacity like overload condition and door status, and indicate the floor number and any emergency situations such as fire alarms or overloading.
2. Design the system architecture: Based on the requirements, we designed the system architecture using an Artix-7 FPGA board with input ports for floor selection buttons and request floors, output ports for floor number display, door status, and emergency indicators, and logic circuits to control the elevator's movement.
3. Develop the control logic: We developed the control logic for the elevator's movement using a state machine that takes into account the current floor and the selected floor from the input buttons to control the elevator's motor.
4. Implement the design: Once the control logic was developed, we implemented the design on the Artix-7 FPGA board using Verilog and tested it to ensure it meets the system requirements.
5. Verify and validate the design: We performed testing and verification of the implemented design to ensure it functions correctly and meets the system requirements. We also validated the design by comparing it with the original requirements to ensure that all the requirements were met.
6. Document the design: We documented the design by creating a design specification, test

plan,to provide a comprehensive understanding of the system design and usage instructions.

List of Variables Abbreviation

FPGA	Field Programmable Gate Array IP
openG	Open ground
closed	Closed ground
openF1	Opened 1st floor
closedF1	Closed 1st floor
openF2	Opened 2nd floor
closedF2	Closed 2nd floor
openF3	Opened 3rd floor
closedF3	Closed 3rd floor
FAG	fire-alarm at ground floor
FA1	fire-alarm at 1st floor
FA2	fire-alarm at 2nd floor
FA3	fire-alarm at 3rd floor
OLG	overload at ground floor
OLF1	overload at 1st floor
OLF2	overload at 2nd floor
OLF3	overload at 3rd floor
BG	Input for the ground floor button inside the elevator.
BF1	Input for the 1st floor button inside the elevator.
BF2	Input for the 2nd floor button inside the elevator.
BF3	Input for the 3rd floor button inside the elevator.
reqG	Input for the ground floor button outside the elevator.
reqF1	Input for the 1st floor button outside the elevator.
reqF2	Input for the 2nd floor button outside the elevator.
reqF3	Input for the 3rd floor button outside the elevator.

FSM State Diagram

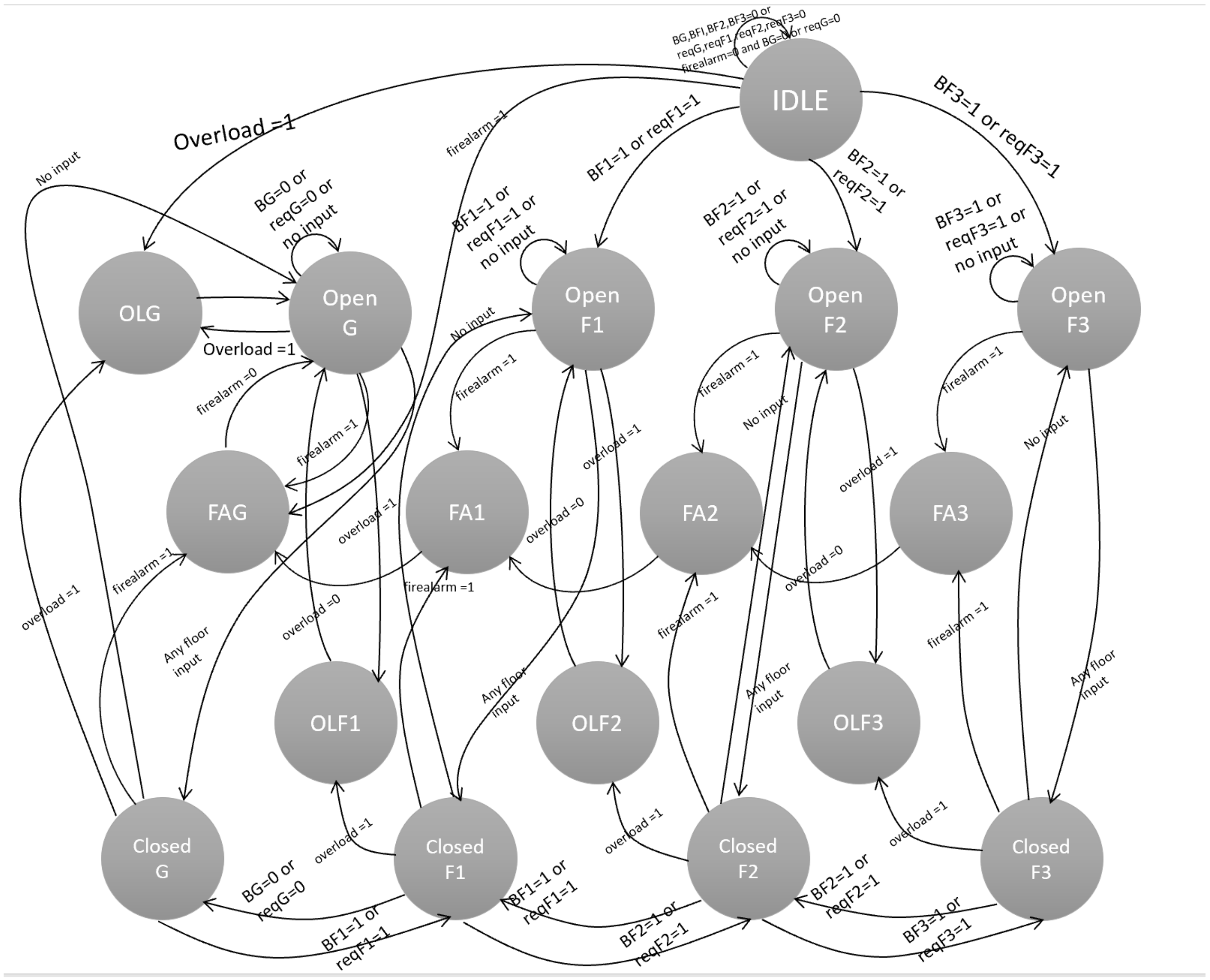


Figure 1 Moore FSM of Elevator

- The elevator control system consists of a total of 17 states.
- Each floor has four states associated with it: opened door, closed door, overload condition, and fire-alarm condition.
- The state names decoding is as follows: Open ground (openG), Closed ground (closedG), Opened 1st floor (openF1), Closed 1st floor (closedF1), Opened 2nd floor (openF2), Closed 2nd floor (closedF2), Opened 3rd floor (openF3), Closed 3rd floor (closedF3), fire-alarm at ground floor (FAG), fire-alarm at 1st floor (FA1), fire-alarm at 2nd floor (FA2), fire-alarm at 3rd floor (FA3), overload at ground floor (OLG), overload at 1st floor (OLF1), overload at 2nd floor (OLF2), overload at 3rd floor (OLF3).
- The elevator is assumed to be in idle state by default and becomes available with an opened door at any floor where it is requested.
- The elevator closes the door when a floor is chosen, moves to the desired floor, opens the door, and stays there with the opened door.
- The operation of the elevator at overload condition: the elevator should not accept any requests and stay at the same floor from where it was accessed, with opened doors. The overload sensor becomes active, the FSM goes to the state OLF1, displays the status on 7-segment display, turns on overload LED signal on the FPGA board, and returns to state o1 with opened doors at 1st floor. If any input is applied at this stage, the elevator will not respond to that and stays there until the overload signal is released.
- The fire-alarm response: when it becomes active, the elevator goes to the ground floor irrespective of its initial state and opens the doors at ground floor. The fire-alarm signal is applied, the FSM moves to state FA1. The doors are closed at FA1 state, turns on the FAout signal LED to indicate fire-alarm, displays the status on 7-segments, and moves to state FAG. From FAG, the elevator arrives at OG state and opens its doors. During this alarm, none of the requests are expected until the fire-alarm signal is de-asserted.
- The overload and fire-alarm sensors work in a similar way as explained above whenever they get triggered.

State Transition Table

Present State	Inputs	Next State	Outputs
	BG, BF1, BF2, BF3, reqG, reqF1, reqF2, reqF3, overload, firealarm		door_open , door_closed, [1:0]prox, FAout, OLout
idle	BG==1 || reqG==1	idle	100000
	BF1==1 || reqF1==1	openF1	100100
	BF2==1 || reqF2==1	openF2	101000
	BF3==1 || reqF3==1	openF3	101100
	overload == 1	OLG	100001
	firealarm==1	FAG	100010
openG	BG==1 || reqG==1	openG	100000
	(BF1==1 || reqF1==1) || (BF2==1 || reqF2==1) || (BF3==1 || reqF3==1)	closedG	010000
	overload == 1	OLG	100001
	firealarm == 1	FAg	100010
closedG	BG==1||reqG== 1	openG	100000
	(BF1==1 || reqF1==1) || (BF2==1 || reqF2==1) || (BF3==1 || reqF3==1)	closedF1	010100
	overload == 1	OLG	100001
	firealarm==1	FAG	100010
openF1	BG==1 || reqG==1	closedF1	010100
	BF1==1 || reqF1==1	openF1	100100
	BF2==1 || reqF2==1	closedF1	010100
	BF3==1 || reqF3==1	closedF1	010100
	overload == 1	OLF1	100101
	firealarm==1	FA1	010110
closedF1	BG==1 || reqG==1	closedG	010000
	BF1==1 || reqF1==1	openF1	100100
	BF2==1 || reqF2==1	closedF2	011000
	BF3==1 || reqF3==1	closedF2	011000
	overload == 1	OLF1	100101
	firealarm==1	FA1	010110

Present State	Inputs	Next State	Outputs
openF2	BG==1 || reqG==1	closedF2	011000
	BF1==1 || reqF1==1	closedF2	011000
	BF2==1 || reqF2 ==1	openF2	101000
	BF3==1 || reqF3==1	closeF2	011000
	overload == 1	OLF2	101001
	firealarm == 1	FA2	011010
closedF2	BG==1 || reqG==1	closedF1	010100
	BF1==1 || reqF1==1	closedF1	010100
	BF2==1 || reqF2 ==1	openF2	101000
	BF3==1 || reqF3==1	closedF3	011100
	overload == 1	OLF2	101001
	firealarm == 1	FA2	011010
openF3	BG==1 || reqG==1	closedF3	011100
	BF1==1 || reqF1==1	closedF3	011100
	BF2==1 || reqF2 ==1	closedF3	011100
	BF3==1 || reqF3==1	openF3	101100
	overload == 1	OLF3	101101
	firealarm == 1	FA3	011110
closedF3	BG==1 || reqG==1	closedF2	011000
	BF1==1 || reqF1==1	closedF2	011000
	BF2==1 || reqF2 ==1	closedF2	011000
	BF3==1 || reqF3==1	openF3	101100
	overload == 1	OLF3	101101
	firealarm == 1	FA3	011110
OLG	X (don’t care)	openG	100000
OLF1	X (don’t care)	openF1	100100
OLF2	X (don’t care)	openF2	101000
OLF3	X (don’t care)	openF3	101100
FAG	X (don’t care)	openG	100000
FA1	X (don’t care)	FAG	100010
FA2	X (don’t care)	FA1	010110
FA3	X (don’t care)	FA2	011010

Module Block Diagram

The overall module block diagram of the system is as follows:

Inputs Explanation

The elevator's input button is represented by a 4-bit input using the one-hot encoding scheme. The four bits correspond to the following inputs:

BG: Input for the ground floor button inside the elevator.

BF1: Input for the 1st floor button inside the elevator.

BF2: Input for the 2nd floor button inside the elevator.

BF3: Input for the 3rd floor button inside the elevator.

If all the bits are 0, this indicates that the elevator is currently at BG. If only the BF1 bit is asserted (value of 1), the elevator will go to the 1st floor. Similarly, if only the BF2 bit is asserted, the elevator will go to the 2nd floor. If only the BF3 bit is asserted, the elevator will go to the 3rd floor. It is important to note that only one of these bits can be asserted at a time to ensure that the elevator goes to the desired floor.

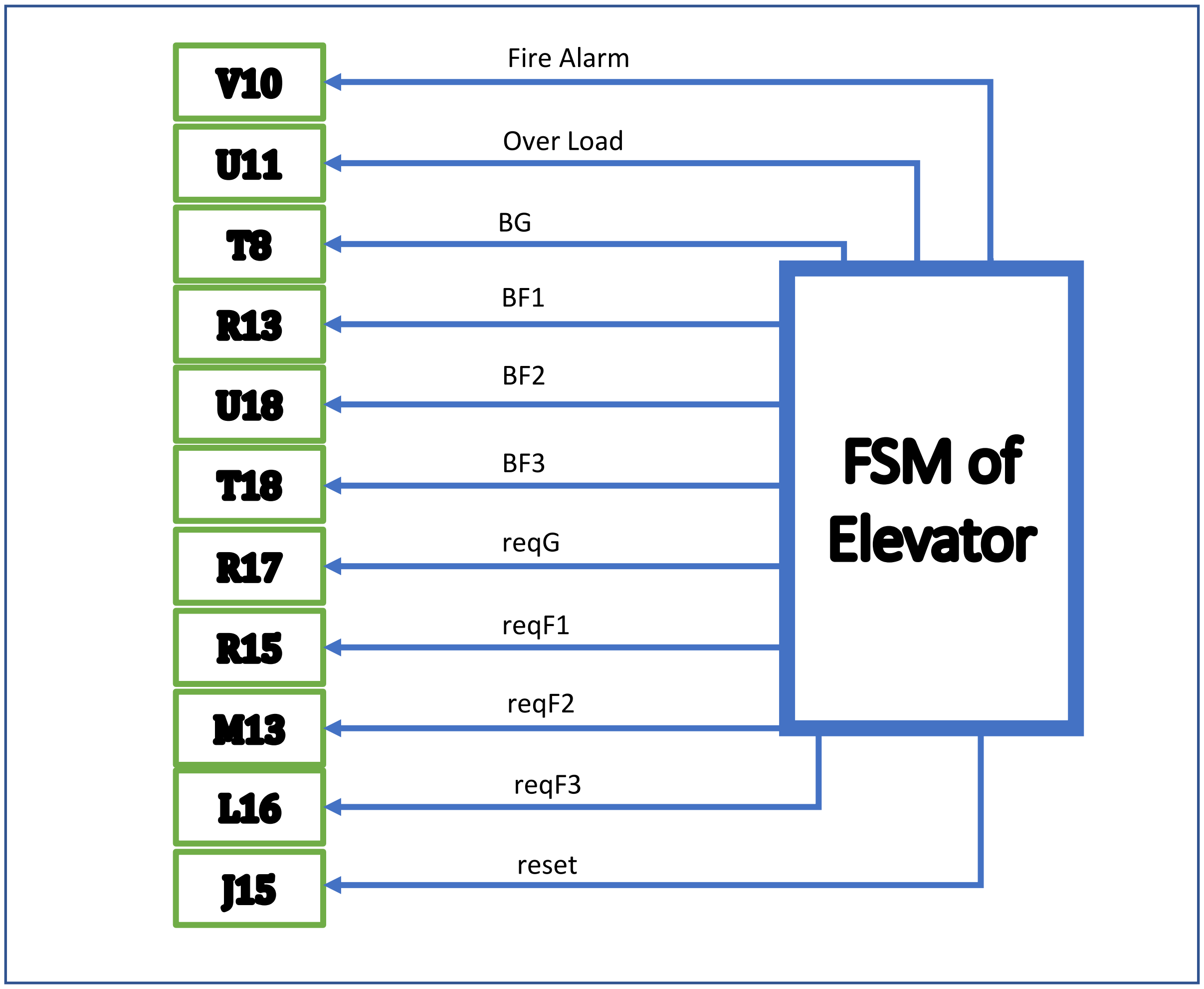
reqG: Input for the ground floor button outside the elevator.

reqF1: Input for the 1st floor button outside the elevator.

reqF2: Input for the 2nd floor button outside the elevator.

reqF3: Input for the 3rd floor button outside the elevator.

The input ports reqG, reqF1, reqF2, reqF3 represent the input buttons outside the elevator, and they also use the same one-hot encoding scheme. If reqG is asserted, it means someone has pressed the button to request the elevator at the ground floor. If reqF1 is asserted, it means someone has pressed the button to request the elevator to go to the 1st floor. Similarly, if reqF2 is asserted, it means someone has pressed the button to request the elevator to go to the 2nd floor, and if reqF3 is asserted, it means someone has pressed the button to request the elevator to go to the 3rd floor.



Input switches Connections of Artix-7 FPGA Board

Output Explanation

Output ports are used to communicate the state of a system or device to external components. In the context of an elevator control system, output ports are used to indicate various states of the elevator, such as whether the door is open or closed, the current floor number, and whether the elevator is overloaded or the fire alarm is triggered.

Here are some common output ports used in elevator control systems:

door_open: activates when the elevator door is opened.

door_closed: activates when the elevator door is closed

prox[1:0]: represents the floor numbers using binary LEDs

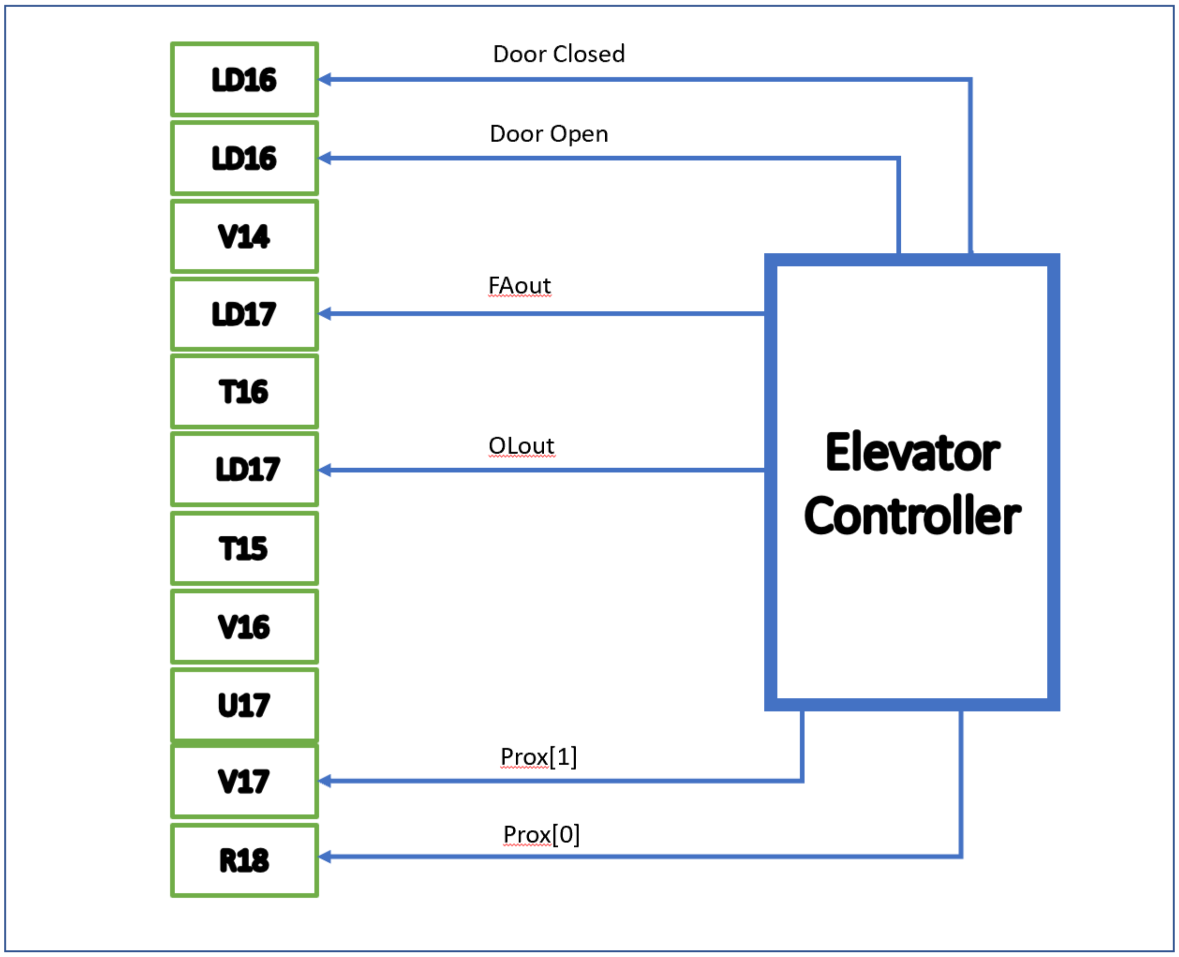
FAout: activates when the fire alarm is triggered

OLout: activates when the elevator is overloaded an[7:0]: used to activate the seven-segment display

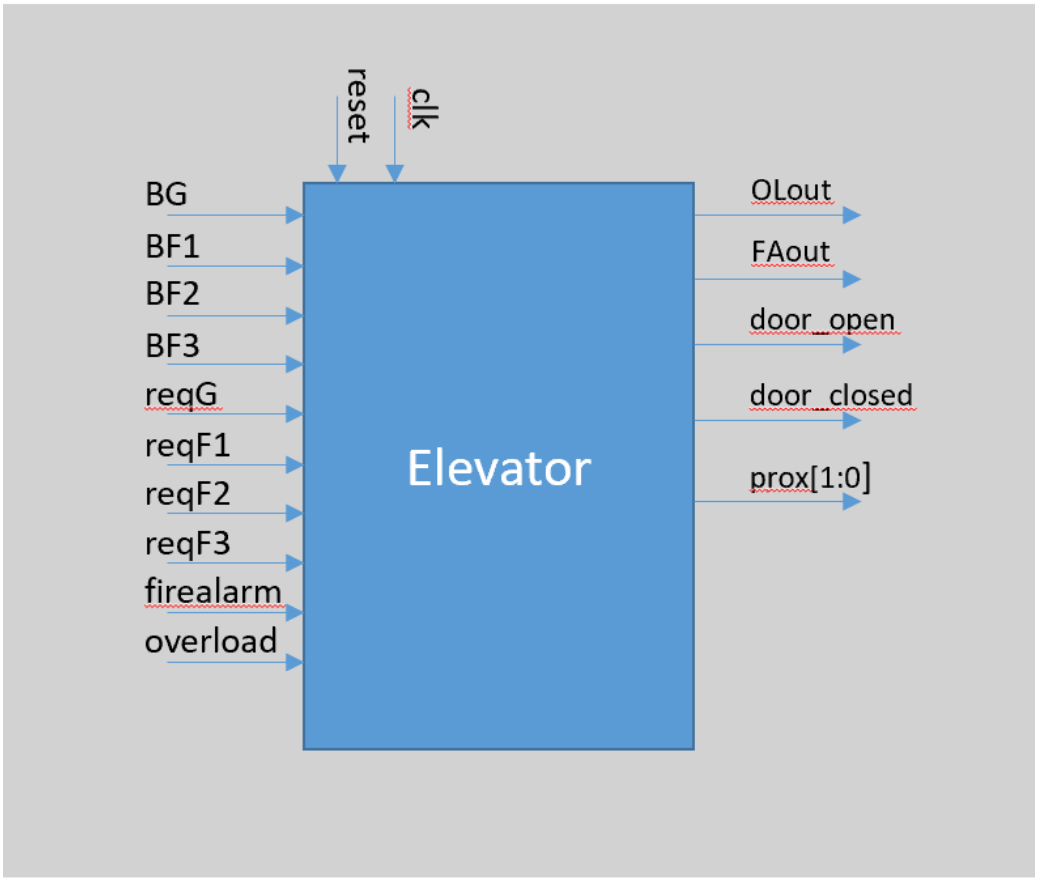
seg[6:0]: cathode-driven output port used to create a signal or a pattern.

The output port prox[1:0] uses binary LEDs to represent the floor numbers. Prox[1] is connected to the V17 LED, while prox[0] is connected to the R18 LED. When prox[1:0] is 00, it indicates that the elevator is on the Ground Floor. If prox[1:0] is 01, the elevator is on the First Floor. Similarly, if prox[1:0] is 10, the elevator is on the Second Floor, and if prox[1:0] is 11, it means the elevator is on the Third Floor, with both LEDs turned on.

The output ports, FAout and OLout, are utilized as RGB LED indicators for fire-alarm and overload sensors, respectively.

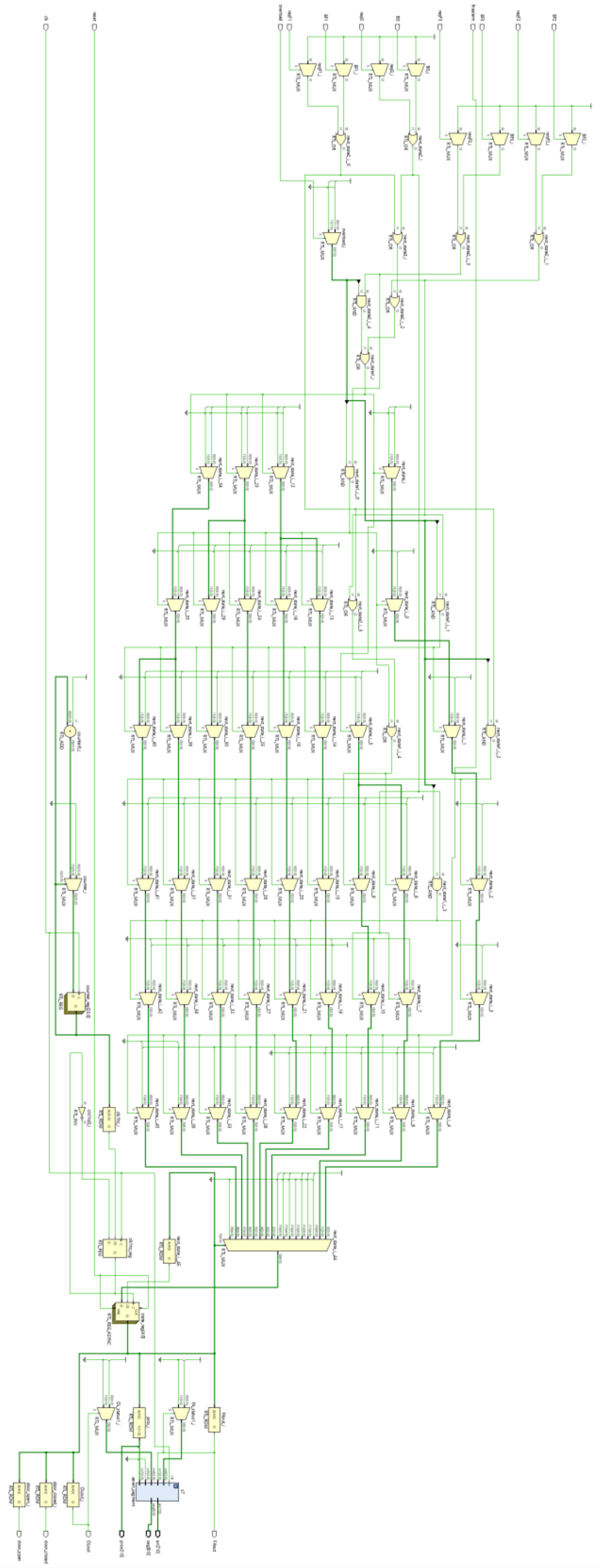


Output LEDs Connections of Artix-7 FPGA Board



Elevator Controller Block Diagram

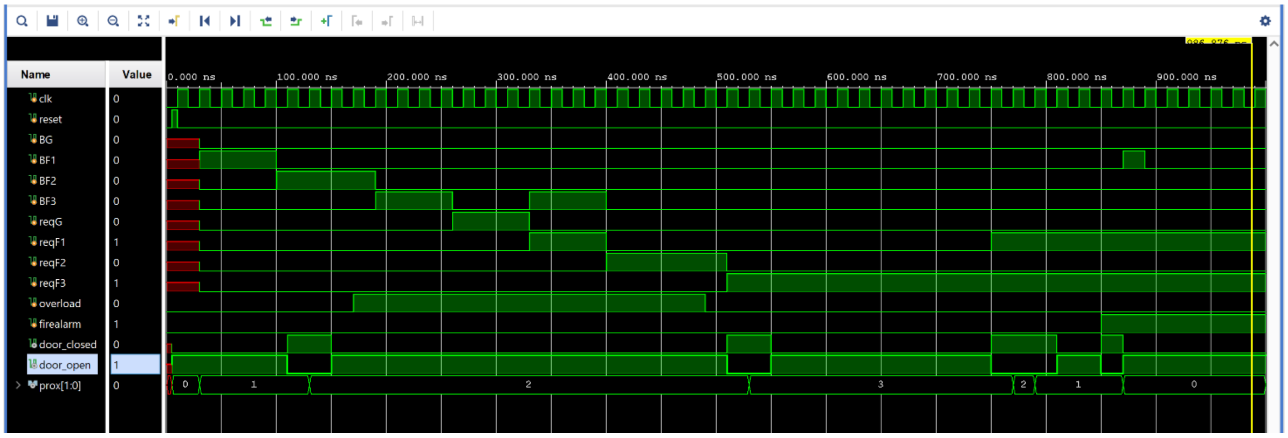
RTL Schematic of FPGA Controller Design for an Elevator



FPGA Results & Simulation

This simulation pattern initializes the input signals of the elevator controller module and simulates different elevator scenarios. It includes the following steps:

- The system received a signal BF1 which is asserted it means the button inside the elevator is pressed and want to move to 1st Floor then the state of an elevator is changed and move to the 1st Floor and the door is open as it can be visible that door_open is asserted.
- After some time BF2 is pressed and the door will be closed for short span and after that the state or position of an elevator is changed to 2nd Floor and the door_open is also asserted.
- When BF3 is asserted means request to 3rd Floor from 2nd Floor and overload signal is asserted it means it will remain on 2nd Floor till the overload signal has been down to 0.
- After checking all the conditions of reqG, reqF1, reqF2 is asserted respectively there was no action taken place and the elevator is still on 2nd Floor because overload signal is asserted.
- When the overload signal is down and reqF3 is asserted then door is closed for short span of time and elevator moved to 3rd Floor and the position can be visible by prox[1:0] bits.
- When reqF1 is asserted then elevator will move first to 2nd Floor then to desired floor which is 1st one.
- When firealarm signal is asserted it means that elevator should go to ground floor wherever the elevator is like if it is on 3rd Floor then it will straight go to ground Floorwhich is in our case is 0.

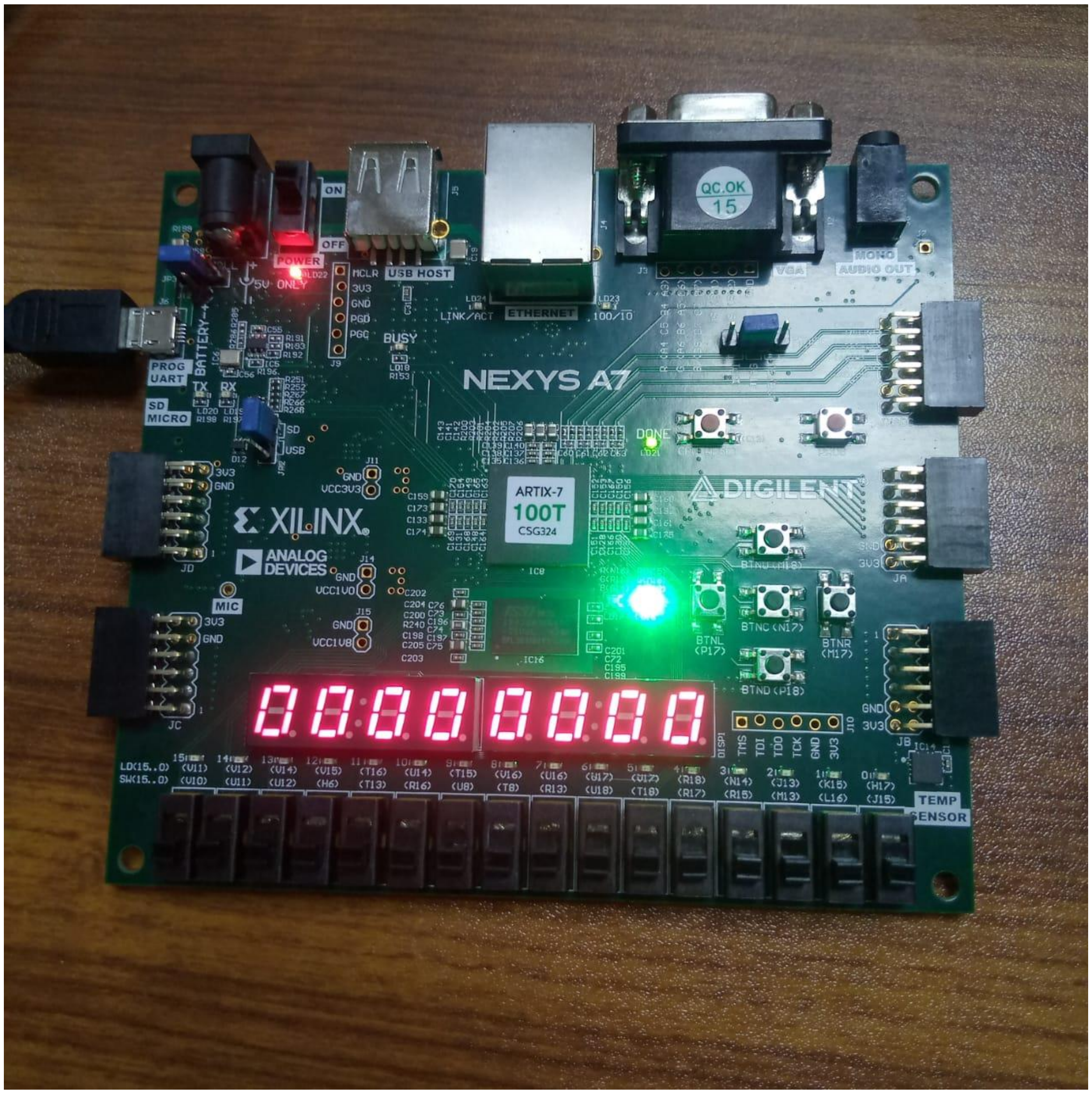


Simulation Results of Elevator Module using Testbench

FPGA Demonstration

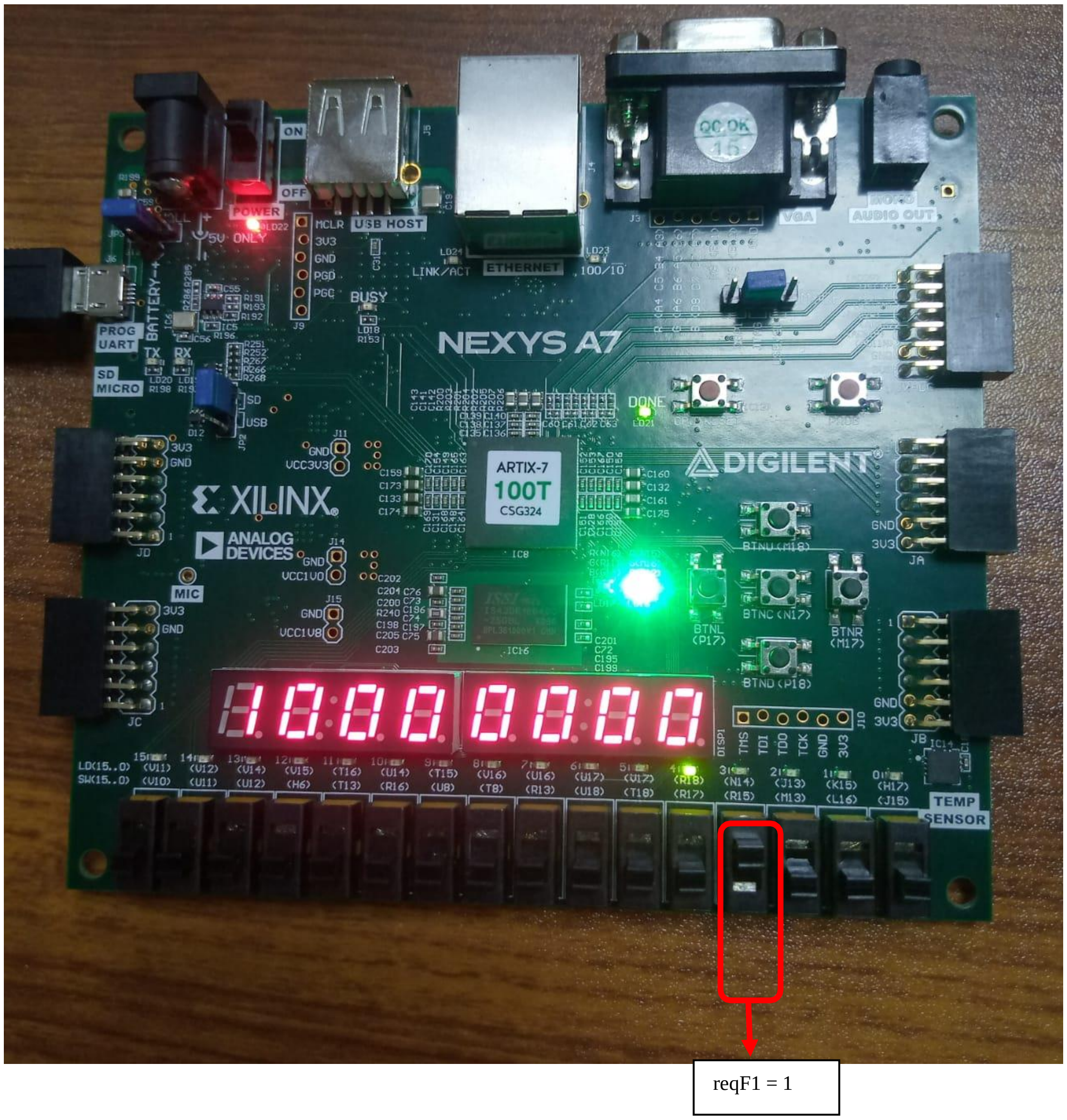
1st Test Case

The system was reset first by asserting J15 switch hence it is in idle state and remaining all signals are not asserted. LD16 which turns green always as the door is open.



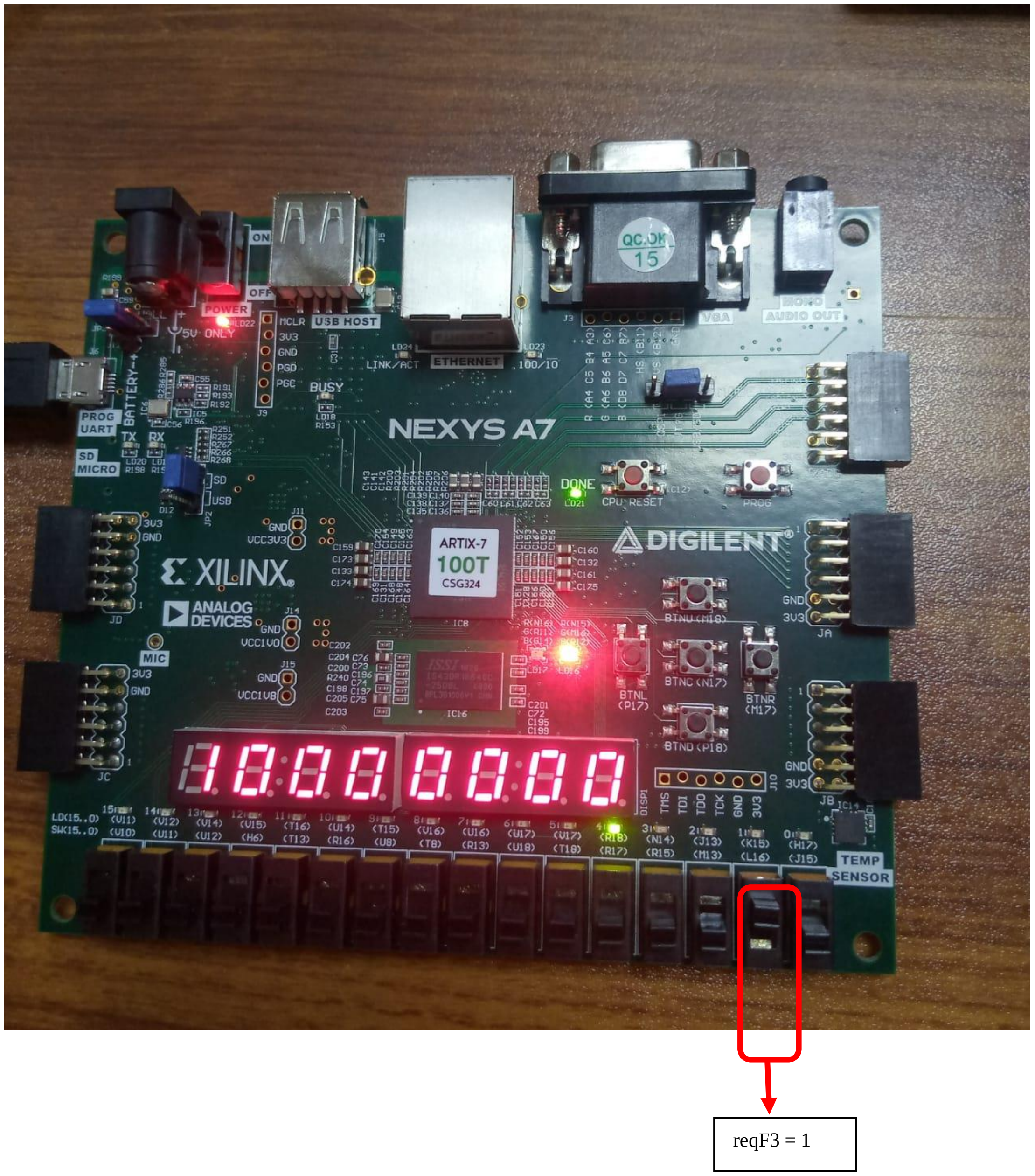
2nd Test Case

Switch **R15** means **reqF1** is asserted and input is applied to the elevator from switches. At this stage, the elevator is at 1nd floor indicated by ‘1’ at the extreme left 7 segment display and also the ‘1’ position is shown by proximity sensor values ‘01’ represented by On LED at **V17** (prox[1]) and OFF at **R18** (prox[0]). The door is opened at this state represented LD16 which turns Green always as the door is open, in contrast when the door is closed then LD16 RGB LED will turn Red.



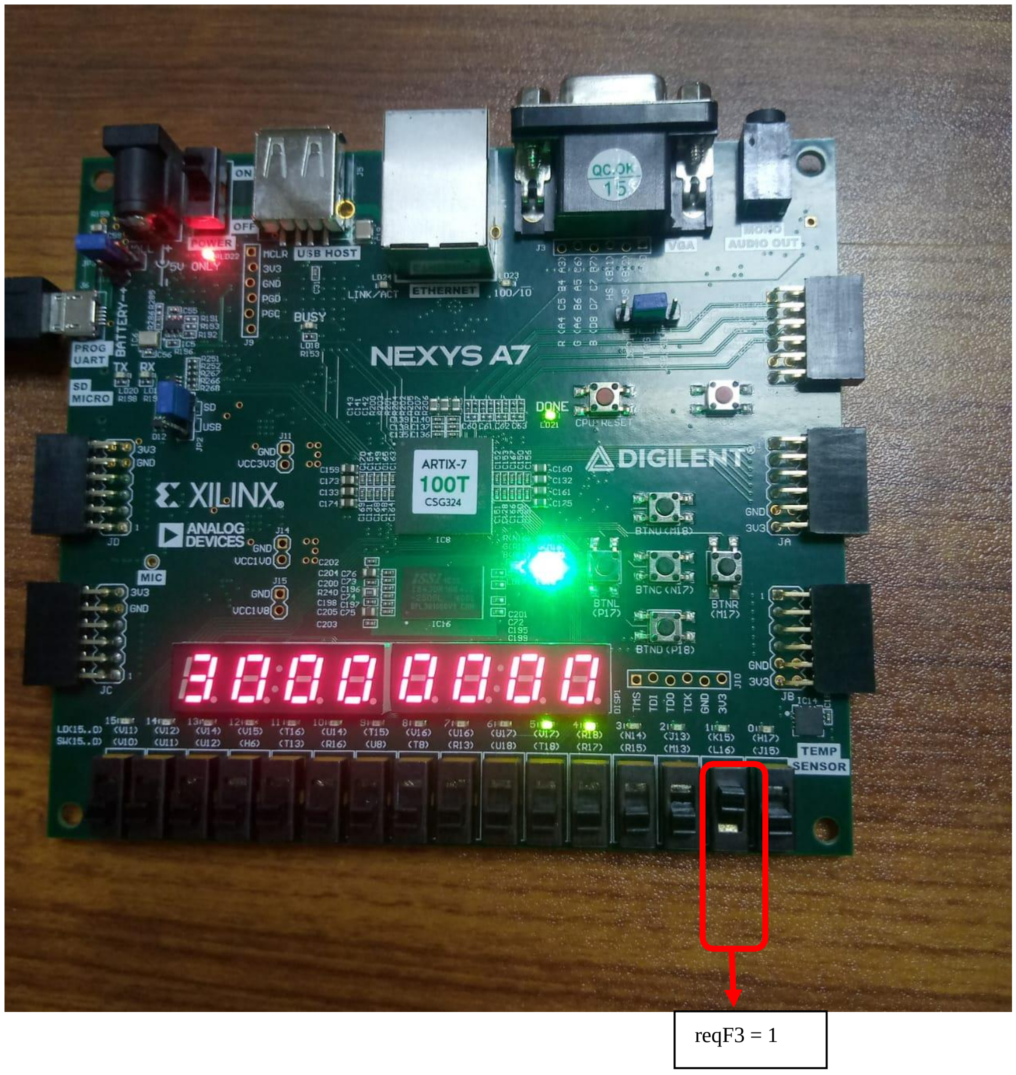
3rd Test Case

Switch **L16** means **reqF3** is asserted and input is applied to the elevator from switches. At this stage, the elevator is at 1nd floor indicated by '1' at the extreme left 7 segment display and also the '1' position is shown by proximity sensor values '01' represented by On LED at **V17** (prox[1]) and OFF at **R18** (prox[0]). The door is closed for small unit of time which could be observed in board and it turns Red, whereas this state will be changed according to current input which is reqF3 then LD16 will turn Green as the door is open,



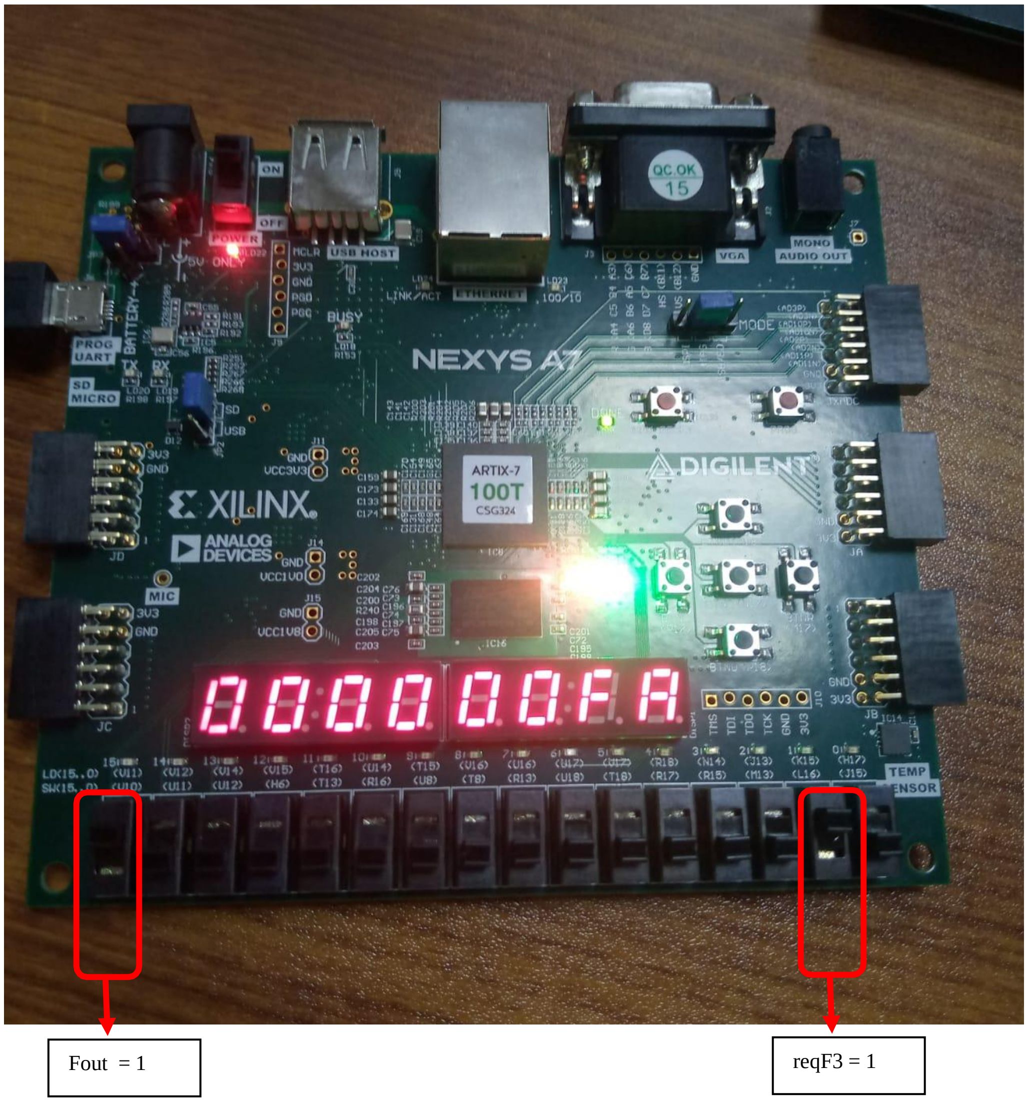
4th Test Case

When the condition of reqF3 meets then Green RGB LED will on as it can be observed in board.



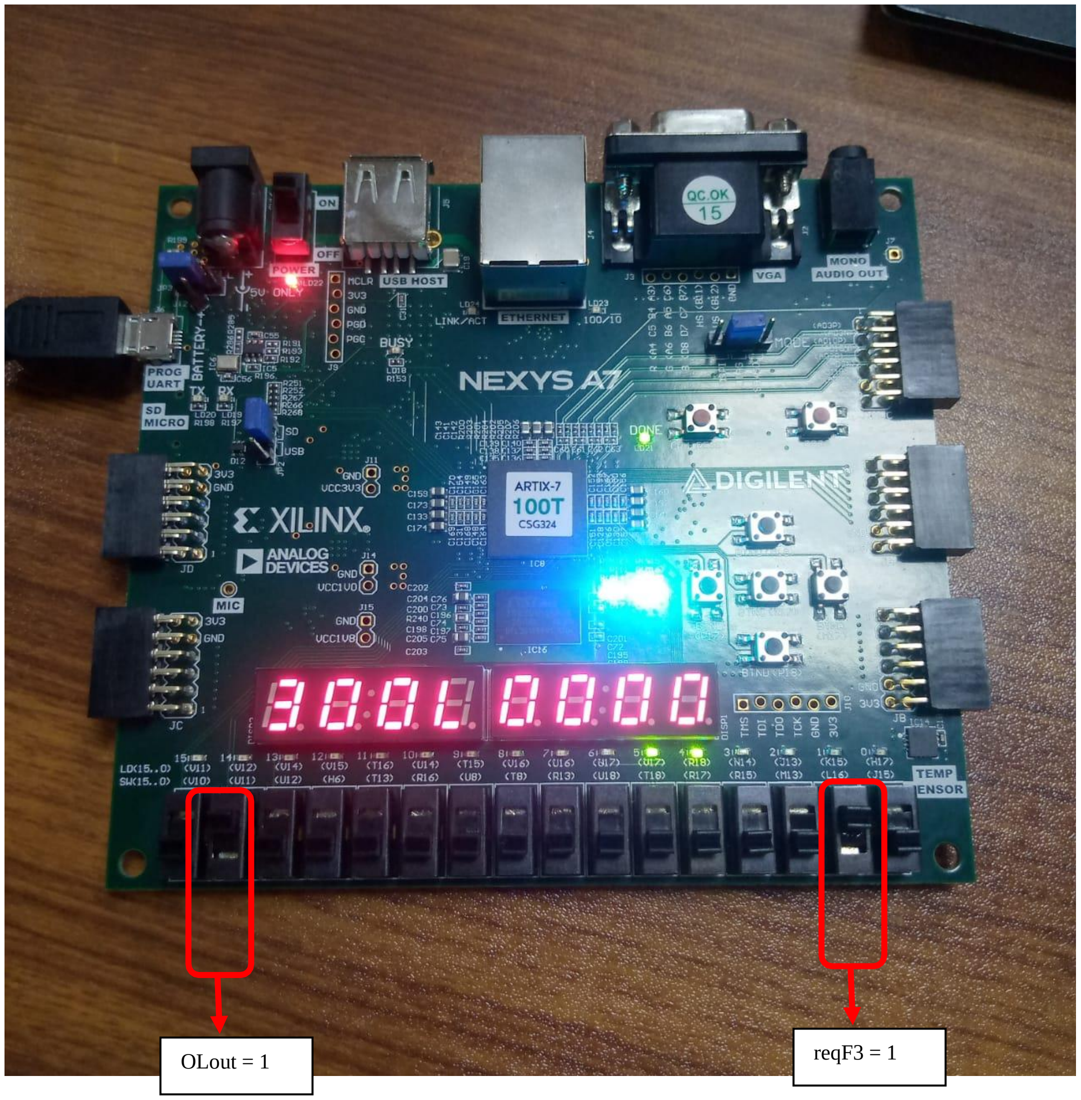
5th Test Case

Switch **V10** means FAout (FireAlarmOut) is asserted and input is applied to the elevator from switches where **L16** switch which is **reqF3** also asserted. At this stage, the elevator was at 3rd Floor. It goes directly to the ground Floor because it is asserted and RGB LED at location **LD17** is also blinking Red indicates that Firealarm is asserted. The door is opened at this state represented by **LD16 RGB LED** as Green.



6th Test Case

Switch **U11** means **OLout** is asserted and input is applied to the elevator from switches. At this stage, the elevator is at 3rd floor indicated by '3' at the extreme left 7 segment display and also the '3' position is shown by proximity sensor values '11' represented by On LED at **V17** (prox[1]) and OFF at **R18** (prox[0]). The door is opened at this state represented RGB LED LD16 which turns Blue when the Overload condition arised, in contrast when the door is closed then LD16 RGB LED will turn Red.



Appendix
Verilog Code of Elevator Design

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Engineer:
//
// Create Date: 04/27/2023 01:28:58 AM
// Design Name:
// Module Name: elevator
// Project Name: FPGA Controller Design for an Elevator
// Target Devices: Nexys-A7
// Tool Versions: Vivado 2020.2
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module elevator(clk,  reset, BG, BF1, BF2, BF3, reqG, reqF1, reqF2, reqF3, overload, firealarm,
an, seg, door_open, door_closed, prox, FAout, OLout
);
 input clk, reset, BG, BF1, BF2, BF3, reqG, reqF1, reqF2, reqF3,
overload, firealarm;

 output reg [1:0] prox;
 output reg OLout, FAout;
 output [6:0] seg;
 output reg door_open , door_closed ;
 output [7:0] an;

 parameter openG = 5'b00000, closedG = 5'b00001, openF1 = 5'b00010, closedF1 =
 5'b00011, openF2 = 5'b00100, closedF2 = 5'b00101, openF3 = 5'b00110, closedF3 =
 5'b00111, idle = 5'b01000;
 parameter OLG = 5'b01001, OLF1 = 5'b01010, OLF2 = 5'b01011,
 OLF3 = 5'b01100, FAG = 5'b01101, FA1 = 5'b01110, FA2 =
 5'b01111, FA3 = 5'b10000;
```

```
reg[4:0] state, next_state;

 reg [25:0] counter;
 reg clk1hz;
 reg [17:0] countr;
 reg clk250hz;


 initial begin
 counter = 0;
 clk1hz = 1;
 countr = 0;
 clk250hz = 1;
 end


 //Module for generating 1 Hz of clock from the 100Mhz of clock

 always @(posedge clk)
 begin
 if(counter == 49_999999)
 begin counter <= 0;
 clk1hz <= ~clk1hz;
 end
 else
 begin
 clk1hz <= clk1hz;
 counter <= counter + 1; end
 end


 // memory element of MOORE FSM

 always@(posedge clk or posedge reset)
 begin
 if(reset==1)
```

```
state <= idle;
else
state <= next_state;
end




// Combinational logic for next state of MOORE FSM

always @(*)
begin
case (state)

idle: begin
  if (firealarm == 0)
  begin
  if ((BG==1 || reqG==1) & overload == 0)
      next_state = idle;
  else if ((BF1==1 || reqF1==1) & overload == 0)
      next_state = openF1;
  else if ((BF2==1 || reqF2==1) & overload == 0)
      next_state = openF2;
  else if ((BF3==1 || reqF3==1)& overload == 0)
      next_state = openF3;
  else if ((BG==1 || reqG==1) || (BF1==1 || reqF1==1) || (BF2==1 || reqF2==1) || (BF3==1 || reqF3==1) & overload == 1)
      next_state = OLG;
  else
      next_state = idle;
  end
  else
      next_state = FAG;
  end
openG: begin
  if (firealarm == 0)
  begin
  if ((BG==1 || reqG==1))
      next_state = openG;
  else if ( (BF1==1 || reqF1==1) || (BF2==1 || reqF2==1) || (BF3==1 || reqF3==1)& overload == 0)
      next_state = closedG;
  else if ((BG==1 || reqG==1) || (BF1==1 || reqF1==1) || (BF2==1 || reqF2==1) || (BF3==1 || reqF3==1) & overload == 1)
      next_state = OLG;
  else
      next_state = openG;
  end
  else
       next_state = FAG;
   end
closedG: begin
   if (firealarm == 0)
   begin
   if ((BG==1||reqG==1))
       next_state = openG;
   else if ((BF1==1 || reqF1==1) || (BF2==1 || reqF2==1) || (BF3==1 || reqF3==1)& overload == 0)
       next_state = closedF1;
   else if ((BG==1 || reqG==1) || (BF1==1 || reqF1==1) || (BF2==1 || reqF2==1) || (BF3==1 || reqF3==1) & overload == 1)
       next_state = OLG;
   else
       next_state = openG;
   end
   else
       next_state = FAG;
   end
openF1: begin
   if(firealarm == 0)
   begin
   if ((BG==1 || reqG==1) & overload == 0)
       next_state = closedF1;
   else if ((BF1==1 || reqF1==1) & overload == 0)
       next_state = openF1;
   else if ((BF2==1 || reqF2==1) & overload == 0)
       next_state = closedF1;
   else if ((BF3==1 || reqF3==1) & overload == 0 )
       next_state = closedF1;
   else if ((BG==1 || reqG==1) || (BF1==1 || reqF1==1) || (BF2==1 || reqF2==1) || (BF3==1 || reqF3==1) & overload == 1)
       next_state = OLF1;
   else
       next_state = openF1;
   end
   else
       next_state = FA1;
   end
closedF1: begin
   if (firealarm == 0)
   begin
   if ((BG==1 || reqG==1) & overload == 0)
       next_state = closedG;
   else if ((BF1==1 || reqF1==1) & overload == 0)
       next_state = openF1;
   else if ((BF2==1 || reqF2==1) & overload == 0)
```

```
     next_state = closedF2;
     else if ((BF3==1 || reqF3==1) & overload == 0)
        next_state = closedF2;
     else if ((BG==1 || reqG==1) || (BF1==1 || reqF1==1) || (BF2==1 || reqF2==1) || (BF3==1 || reqF3==1) & overload == 1)
        next_state = OLF1;
     else
        next_state = openF1 ;
     end
     else
        next_state = FA1;
     end
  openF2: begin
     if(firealarm == 0)
     begin
     if ((BG==1 || reqG==1) & overload == 0)
        next_state = closedF2;
     else if ((BF1==1 || reqF1==1) & overload == 0)
        next_state = closedF2;
     else if ((BF2==1 || reqF2 ==1) & overload == 0)
        next_state = openF2;
     else if ((BF3==1 || reqF3==1) & overload == 0)
        next_state = closedF2;
     else if ((BG==1 || reqG==1) || (BF1==1 || reqF1==1) || (BF2==1 || reqF2==1) || (BF3==1 || reqF3==1) & overload == 1)
        next_state = OLF2;
     else
        next_state = openF2;
     end
     else
        next_state = FA2;
     end
  closedF2: begin
     if (firealarm==0)
     begin
     if ((BG==1 || reqG==1) & overload == 0)
        next_state = closedF1;
     else if ((BF1==1 || reqF1==1) & overload == 0 )
        next_state = closedF1;
     else if ((BF2==1 || reqF2 ==1) & overload == 0 )
        next_state = openF2;
     else if ((BF3==1 || reqF3==1) & overload == 0 )
        next_state = closedF3;
     else if ((BG==1 || reqG==1) || (BF1==1 || reqF1==1) || (BF2==1 || reqF2==1) || (BF3==1 || reqF3==1) & overload == 1 )
        next_state = OLF2;
     else
        next_state = openF2;
     end
     else
        next_state = FA2;
     end
  openF3: begin
     if(firealarm == 0)
     begin
     if ((BG==1 || reqG==1) & overload == 0)
        next_state = closedF3;
     else if ((BF1==1 || reqF1==1) & overload == 0)
        next_state = closedF3;
     else if ((BF2==1 || reqF2 ==1) & overload == 0)
        next_state = closedF3;
     else if ((BF3==1 || reqF3==1) & overload == 0)
        next_state = openF3;
     else if ((BG==1 || reqG==1) || (BF1==1 || reqF1==1) || (BF2==1 || reqF2==1) || (BF3==1 || reqF3==1) & overload == 1)
        next_state = OLF3;
     else
        next_state = openF3;
     end
     else
        next_state = FA3;
     end
  closedF3: begin
     if(firealarm == 0)
     begin
     if ((BG==1 || reqG==1) & overload == 0)
        next_state = closedF2;
     else if ((BF1==1 || reqF1==1) & overload == 0)
        next_state = closedF2;
     else if ((BF2==1 || reqF2 ==1) & overload == 0)
        next_state = closedF2;
     else if ((BF3==1 || reqF3==1) & overload == 0)
        next_state = openF3;
     else if ((BG==1 || reqG==1) || (BF1==1 || reqF1==1) || (BF2==1 || reqF2==1) || (BF3==1 || reqF3==1) & overload == 1)
        next_state = OLF3;
     else
        next_state = openF3;
     end
     else
        next_state = FA3;
     end
  OLG: next_state = openG;

  OLF1: next_state = openF1;
```

```
OLF2: next_state = openF2;

OLF3: next_state = openF3;

FAG: next_state = openG;

FA1: next_state = FAG;

FA2: next_state = FA1;

FA3: next_state = FA2;

default : next_state = state;
endcase
end




// output logic for MOORE FSM

always @(state)
begin
door_open = 1; door_closed = 0;
prox = 0;
FAout = 0;
OLout = 0;

case (state) // door = 0 means it is open, door = 1 means it is closed

idle: begin
   door_open = 1;
   door_closed = 0;
   prox = 0;
   FAout= 0 ;
   OLout = 0;
   end

openG: begin
   door_open = 1;
   door_closed = 0;
   prox = 0;
   FAout = 0;
   OLout = 0;
   end
```

```
closedG: begin
   door_open = 0;
   door_closed = 1;
   prox =0;
   FAout = 0;
   OLout = 0;
   end

openF1: begin
   door_open = 1;
   door_closed = 0;
   prox =1;
   FAout = 0;
   OLout = 0;
   end

closedF1: begin
   door_open = 0;
   door_closed = 1;
   prox =1;
   FAout = 0;
   OLout = 0;
   end

openF2: begin
   door_open = 1;
   door_closed = 0;
   prox =2; FAout = 0;
   OLout = 0;
   end
closedF2: begin
  door_open = 0;
  door_closed = 1;
  prox =2;
  FAout = 0;
  OLout = 0;
end

openF3: begin
   door_open = 1;
   door_closed = 0;
   prox =3;
   FAout = 0;
   OLout = 0;
   end
```

```
closedF3: begin
   door_open = 0;
   door_closed = 1;
   prox =3;
   FAout = 0;
   OLout = 0;
   end

OLG: begin
   door_open = 1;
   door_closed = 0;
   prox =0;
   FAout = 0;
   OLout = 1;
   end

OLF1: begin
   door_open = 1;
   door_closed = 0;
   prox =1;
   FAout = 0;
   OLout = 1;
   end

OLF2: begin
   door_open = 1;
   door_closed = 0;
   prox =2;
   FAout = 0;
   OLout = 1;
   end

OLF3: begin
   door_open = 1;
   door_closed = 0;
   prox =3;
   FAout = 0;
   OLout = 1;
   end

FAG: begin
   door_open = 1;
   door_closed = 0;
   prox =0;
   FAout = 1;
```

```
  OLout = 0;
   end

FA1: begin
   door_open = 0;
   door_closed = 1;
   prox =1;
   FAout = 1;
   OLout = 0;
   end

FA2: begin
   door_open = 0;
   door_closed = 1;
   prox =2;
   FAout = 1;
   OLout = 0;
   end

FA3: begin
   door_open = 0;
   door_closed = 1;
   prox =3;
   FAout = 1;
   OLout = 0;
   end
endcase
end



always @(posedge clk)
begin
if(countr == 199999)
   begin
   countr <= 0;
   clk250hz <= ~clk250hz;
   end
else
   begin
   clk250hz <= clk250hz;
   countr <= countr + 1;
   end
end
```

```
reg [3:0] FA_status1, FA_status2, OL_status1, OL_status2, position;



always @(clk250hz)
begin
position <= prox;
if(FAout == 1)
   begin
   FA_status1 <= 4'hA;
   FA_status2 <= 4'hF;
   end
else
   begin
   FA_status1 <= 0;
   FA_status2 <= 0;
   end

if (OLout == 1)
   begin
   OL_status1 <= 4'h9;
   OL_status2 <= 4'h0;
   end
else
   begin
   OL_status1 <= 0;
   OL_status2 <= 0;
   end
end



seven_segment s7 (.clk(clk), .in0(FA_status1), .in1(FA_status2),
.in2(), .in3(), .in4(OL_status1), .in5(OL_status2), .in6(),
.in7(position), .seg(seg), .an(an));
```

```
endmodule



// seven segment driver
module seven_segment(clk,in0, in1, in2, in3,in4,in5,in6,in7,seg,an);
input [3:0] in0;
input [3:0] in1;
input [3:0] in2;
input [3:0] in3;
input [3:0] in4;
input [3:0] in5;
input [3:0] in6;
input [3:0] in7;
input clk;
wire [6:0] d0, d1, d2, d3, d4, d5, d6, d7;
output reg [6:0] seg;
output reg [7:0] an;
reg [17:0] count;
reg clk250hz;

initial begin
count = 0;
 clk250hz = 1;
end
always @(posedge clk)
begin
 if(count == 199999)
 begin count <= 0;
 clk250hz <= ~clk250hz; end
 else
 begin
 clk250hz <= clk250hz;
 count <= count + 1; end
end
reg [3:0] pstate,next;
initial begin
 pstate = 0;
 next = 0;
end
always @ (posedge clk250hz)
begin
 pstate <= next;
end
always @ (pstate)
```

```
begin
 case(pstate)
 0: begin next <= 1; an <= 8'b11111110; seg <= d0; end
 1: begin next <= 2; an <= 8'b11111101; seg <= d1; end
 2: begin next <= 3; an <= 8'b11111011; seg <= d2; end
 3: begin next <= 4; an <= 8'b11110111; seg <= d3; end
 4: begin next <= 5; an <= 8'b11101111; seg <= d4; end
 5: begin next <= 6; an <= 8'b11011111; seg <= d5; end
 6: begin next <= 7; an <= 8'b10111111; seg <= d6; end
 7: begin next <= 0; an <= 8'b01111111; seg <= d7; end
 default: begin next <= 0; an <= 8'b00000000; seg <= 7'b111_1111; end
 endcase
end

segdecoder D0(d0, in0);
segdecoder D1(d1, in1);
segdecoder D2(d2, in2);
segdecoder D3(d3, in3);
segdecoder D4(d4, in4);
segdecoder D5(d5, in5);
segdecoder D6(d6, in6);
segdecoder D7(d7, in7);

endmodule







// seven segment decoder
module segdecoder(out,in);
output reg [6:0] out;
input [3:0]in;
always @(in)
begin
 case(in)
 0: out = 7'b1000000;
 1: out = 7'b1111001;
 2: out = 7'b0100100;
 3: out = 7'b0110000;
 4: out = 7'b0001101;
 5: out = 7'b0100100;
 6: out = 7'b0000010;
 7: out = 7'b1111000;
 8: out = 7'b0000000;
 9: out = 7'b1000111; // to display L of OL (overload condition)
 10: out = 7'b0001000; // to display A of FA (firealarm)
 11: out = 7'b0000000;
 15: out = 7'b0001110; // to display F of FA (firealarm)
 default: out = 7'b1111111;
 endcase
end

endmodule
```

Test Bench Code for FPGA based Elevator Design

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/27/2023 02:57:26 PM
// Design Name:
// Module Name: tb_elevator
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module tb_elevator();
 reg clk, reset, BG, BF1, BF2, BF3, reqG, reqF1, reqF2, reqF3, overload, firealarm;
 wire door_closed, door_open;
 wire [1:0] prox;




 elevator dut (.clk(clk),.reset(reset),.BG(BG),.BF1(BF1),.BF2(BF2),.BF3(BF3),.reqG(reqG),.reqF1(reqF1),.reqF2(reqF2),
 .reqF3(reqF3),.overload(overload),.firealarm(firealarm),.door_open(door_open),.door_closed(door_closed),.prox(prox));


initial begin
  clk = 0;
end

always #10 clk = ~clk;

initial begin
firealarm = 0;
```

```
overload = 0;
reset = 0;
#5 reset = 1;
#5 reset = 0;
#20 BG = 0; BF1 = 1; BF2 = 0; BF3 = 0; reqG = 0;
reqF1 = 0; reqF2 = 0; reqF3 = 0;
#50
#20 BG = 0; BF1 = 0; BF2 = 1; BF3 = 0; reqG = 0;
reqF1 = 0; reqF2 = 0; reqF3 = 0;
#50
#20 overload = 1;
#20 BG = 0; BF1 = 0; BF2 = 0; BF3 = 1; reqG = 0;
reqF1 = 0; reqF2 = 0; reqF3 = 0;
#50
#20 BG = 0; BF1 = 0; BF2 = 0; BF3 = 0; reqG = 1;
reqF1 = 0; reqF2 = 0; reqF3 = 0;
#50
#20 BG = 0; BF1 = 0; BF2 = 0; BF3 = 1; reqG = 0;
reqF1 = 1; reqF2 = 0; reqF3 = 0;
#50
#20 BG = 0; BF1 = 0; BF2 = 0; BF3 = 0; reqG = 0;
reqF1 = 0; reqF2 = 1; reqF3 = 0;
#50
#40 overload = 0;
#20 BG = 0; BF1 = 0; BF2 = 0; BF3 = 0; reqG = 0;
reqF1 = 0; reqF2 = 0; reqF3 = 1;
#50
#100 reqF1 = 0;
#50
#40 reqF1 = 1;
#100 firealarm = 1;
#20 BF1 = 1;
#20 BF1 = 0;
#160 firealarm = 0;
#40 BF3 = 1;
#80 BF3 = 0;
end



endmodule
```

Universal Constraint File Code

```
set_property PACKAGE_PIN J17 [get_ports an[0]]
set_property PACKAGE_PIN J18 [get_ports an[1]]
set_property PACKAGE_PIN T9 [get_ports an[2]]
set_property IOSTANDARD LVCMOS33 [get_ports an[3]]
set_property IOSTANDARD LVCMOS33 [get_ports an[2]]
set_property IOSTANDARD LVCMOS33 [get_ports an[1]]
set_property IOSTANDARD LVCMOS33 [get_ports an[0]]
set_property IOSTANDARD LVCMOS33 [get_ports prox[1]]
set_property IOSTANDARD LVCMOS33 [get_ports prox[0]]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[5]}]

set_property IOSTANDARD LVCMOS33 [get_ports {seg[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports FAout]
set_property IOSTANDARD LVCMOS33 [get_ports firealarm]
set_property IOSTANDARD LVCMOS33 [get_ports BG]
set_property IOSTANDARD LVCMOS33 [get_ports BF1]
set_property IOSTANDARD LVCMOS33 [get_ports BF2]
set_property IOSTANDARD LVCMOS33 [get_ports BF3]
set_property IOSTANDARD LVCMOS33 [get_ports OLout]
set_property IOSTANDARD LVCMOS33 [get_ports overload]
set_property IOSTANDARD LVCMOS33 [get_ports reqG]
set_property IOSTANDARD LVCMOS33 [get_ports reqF1]
set_property IOSTANDARD LVCMOS33 [get_ports reqF2]
set_property IOSTANDARD LVCMOS33 [get_ports reqF3]
set_property IOSTANDARD LVCMOS33 [get_ports reset]

set_property IOSTANDARD LVCMOS33 [get_ports {an[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {an[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {an[5]}]


set_property IOSTANDARD LVCMOS33 [get_ports {an[4]}]
set_property PACKAGE_PIN J14 [get_ports {an[3]}]
set_property PACKAGE_PIN P14 [get_ports {an[4]}]
set_property PACKAGE_PIN T14 [get_ports {an[5]}]
set_property PACKAGE_PIN K2 [get_ports {an[6]}]
set_property PACKAGE_PIN U13 [get_ports {an[7]}]
set_property PACKAGE_PIN U18 [get_ports BF2]
set_property PACKAGE_PIN R13 [get_ports BF1]
set_property PACKAGE_PIN T8 [get_ports BG]
set_property PACKAGE_PIN T18 [get_ports BF3]
set_property PACKAGE_PIN E3 [get_ports clk]
set_property PACKAGE_PIN V10 [get_ports firealarm]
set_property PACKAGE_PIN R17 [get_ports reqG]
set_property PACKAGE_PIN R15 [get_ports reqF1]
set_property PACKAGE_PIN M13 [get_ports reqF2]
set_property PACKAGE_PIN L16 [get_ports reqF3]
set_property PACKAGE_PIN J15 [get_ports reset]
set_property PACKAGE_PIN V11 [get_ports door_closed]
set_property PACKAGE_PIN V12 [get_ports door_open]
#set_property PACKAGE_PIN V15 [get_ports FAout]
#set_property PACKAGE_PIN U14 [get_ports OLout]
set_property PACKAGE_PIN R10 [get_ports {seg[1]}]
set_property PACKAGE_PIN T10 [get_ports {seg[0]}]
set_property PACKAGE_PIN K16 [get_ports {seg[2]}]
set_property PACKAGE_PIN K13 [get_ports {seg[3]}]
set_property PACKAGE_PIN P15 [get_ports {seg[4]}]
set_property PACKAGE_PIN T11 [get_ports {seg[5]}]
set_property PACKAGE_PIN L18 [get_ports {seg[6]}]
set_property PACKAGE_PIN V17 [get_ports {prox[1]}]
set_property PACKAGE_PIN R18 [get_ports {prox[0]}]
set_property PACKAGE_PIN U11 [get_ports overload]

set_property -dict { PACKAGE_PIN M16   IOSTANDARD LVCMOS33 } [get_ports  door_open ]; #green led
set_property -dict { PACKAGE_PIN N15   IOSTANDARD LVCMOS33 } [get_ports  door_closed ];  #red led


set_property -dict { PACKAGE_PIN N16   IOSTANDARD LVCMOS33 } [get_ports  FAout ]; #red led


set_property -dict { PACKAGE_PIN G14   IOSTANDARD LVCMOS33 } [get_ports  OLout ]; #blue led
```