Laboratory Report

DESIGNING DIGITAL CONTROLLER FOR ELECTRONIC DUMBWAITER LIFT (EDL) SYSTEM

EEX5351

Digital Electronic System

LAB 02

By

S.A.P. Kavinda

617143597

Submitted to

Department of Electrical & Computer Engineering

Faculty of Engineering Technology

The Open University of Sri Lanka

On

11/12/2022

Table of Contents

COMBINATIONAL LOGIC DIGITAL SYSTEM DESIGNING AND IMPLEMENTATION

1. Learning Outcome

- Apply the concepts of basic timing issues, including clocking, timing constraints, and propagational delays during design process.
- Develop Complex digital systems in a hierarchical fashion using top-down and bottom-up design approaches.
- Verify the digital system design via digital circuit simulation using an HDL.
- Implement digital system design using a programmable platform.
- Explain the types and characteristics of common fault that occur a in digital electronic systems.
- Discuss different computer-aid testing tools for circuit simulation, fault diagnosis and Automatic Test Pattern Generator.
-

2. Apparatus

- Anvyls Spartan 6 FPGA board
- Xlinx ISE Design Suite 14.7
- Active-HDL Simulation Tool
- Visual Studio Code – Timing Diagram Plugin for visualize timing diagrams
- SIMPLIS 8.20 – For drawing circuit diagrams

3. Theory Related to Design

3.1 Digital Electronic Systems

We can call the physical systems as the set of interconnected objects or elements that realize some functions using a set of input and output signals. The systems where all input and outputs are digital signals are called Digital Systems. In modern days digital systems are presents in everywhere. We can say the simplicity of design approaches, the higher efficiency, high accuracy, and the modeling capability using algorithms are the key points of digital electronic systems that involve to this success. Therefore, the design and implementation is an important part for Engineers for developing sustainable systems and analysis. Digital Electronic systems use two levels of voltages or currents for represents the logic 0 or 1 that in binary number system. These are the basic components of digital systems. Also, the switches are the main controlling elements of digital electronic systems. Normally MOS Transistors are uses for developing this switches. By using MOS transistors, we can implement Inverters, NAND gates and NOR gates. These NAND and NOR gates are called as universal gate because other all logic gates (AND, OR, XOR, XNOR) could implement using one of these.

Why use NAND or NOR gates instead of AND and OR gates? Mainly there are three reasons for that. In the laboratory, we only need one kind of gate. Because we can implement all kinds of gate using sing NAND or NOR gates. Also, nMOS switches are good for transmitting 0V. pMOS switches are transmitting 1V. So in COMS technology, AND gates are implemented using NAND gates and OR gates are Implemented using NOR gates. The 3rd reason is within an IC, NAND and NOR gates are cheaper than AND and OR gates.

3.2 Delay Timing

In reality, the output of a piece of real hardware does not change instantaneously when an input changes. Inevitably, there is a delay. This is due to various things such as capacitance of transistors, Transmission line capacitance etc… In VHDL digital system modelling, there are three prescribed delays can be identified. Any signal assignment will incur a delay of one the three types of below.

1. Transport Delay

In this delay type consider propagation delay only. This signal will assume the new value after specific time.

$$output <= TRANSPORT\ x\ AFTER\ 10ns;$$

This Transport delay is like a infinite bandwidth Transmission line.

2. Inertial Delay

This Delay deal with the minimum input pulse with and propagation delay. Inertial delay acts like a real gate, It absorbs pulses narrow than in width than the propagation delay. This is the default in VHDL statements which contain “AFETER” clause.

$$output <= x\ AFETR\ 10ns$$

Also, In VHDL we can define the Pulse Reject time specifically also. This REJECT keyword can be used only with the keyword INTERNAL.

$$output = REJECT\ 5ns\ INTERNAL\ x\ AFETR\ 10ns;$$

3. Delta Delay

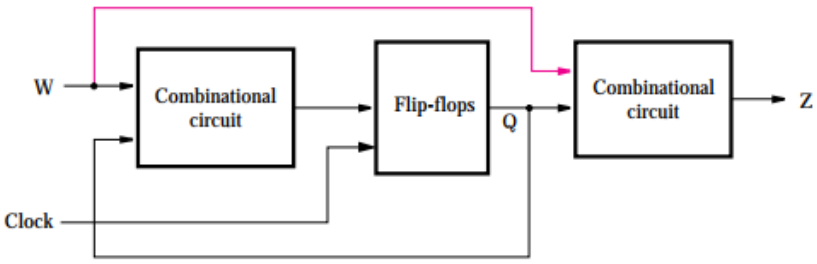
This type represents the zero-delay time, that’s means ideal components. In VHDL not included any keyword in the signal assignment.

3.3 Anvyl FPGA Development board

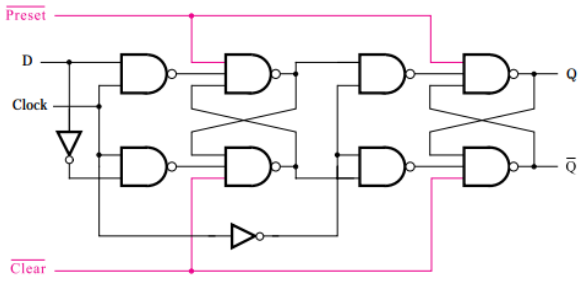
FPGA (Field Programmable Gate Arrays) are the semiconductor devices that are based on a matrix of configurable logic block (CLBs) connected via programmable interconnects. Although there are various types of FPGA available, SRAM based FPGAs are dominant.

The Anvyl FPGA development platform is a complete, ready to use digital circuit development platform based on a speed grade -3 Xilinx Spaetan-6 LX45 FPGA. In this laboratory session we are used this demonstration board to demonstrate our designs.

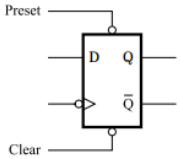
3.4 Sequential Circuits or Finite State Machine



Many of designs required a make output depended not only current input, but also depended old inputs. So achieve this requirement, digital circuits have to be a combination of both combination circuits and memory devices. When considering the structure of design, sequential circuit can be divided into two categories known as synchronous circuits and Asynchronous circuits. In synchronous circuits use a clock and all memory devices are work in order to this clock. Asynchronous circuits are working without a clock and the memory devices are working one after one manner. Sequential circuits are also can divide into two categorize as considering the design method, Mealy machine and Moore machine. In Moore machines output depends only with the current state and states are changing as current input. In Mealy machines. Output depended on both current input and current state. There are numerous ways to implement memory blocks such as flip flips, static memories and Opto-magnetic memory devices. A simplest memory block is a D-Flip Flop and it can implement in that way,



(a) Circuit



(b) Graphical symbol

4. Introduction to the Design

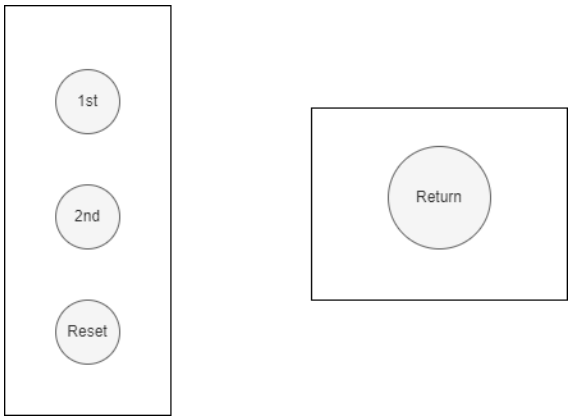
For the LAB to, I had to design a digital controller for a Electronic Dumbwaiter Lift(EDL) for a hotel. The typical view of EDL controller as below,

And the given specifications are,

1. The hotel owner initially plan's to implements the system for three stories (ground floor, first floor, and second floor).
2. Push buttons are used for selecting the appropriate floor to move the lift from the default location (ground floor kitchen).
3. Limit Switches will be used for identifying the floor when the lift reaches.
4. Return switches are used for sending the EDL to the default location.
5. RESET can be used to reset the EDL system.

For implement this designed controller, I used a mealy machine with a encoder, In my design have 7 inputs and 2 output to control the motor direction.

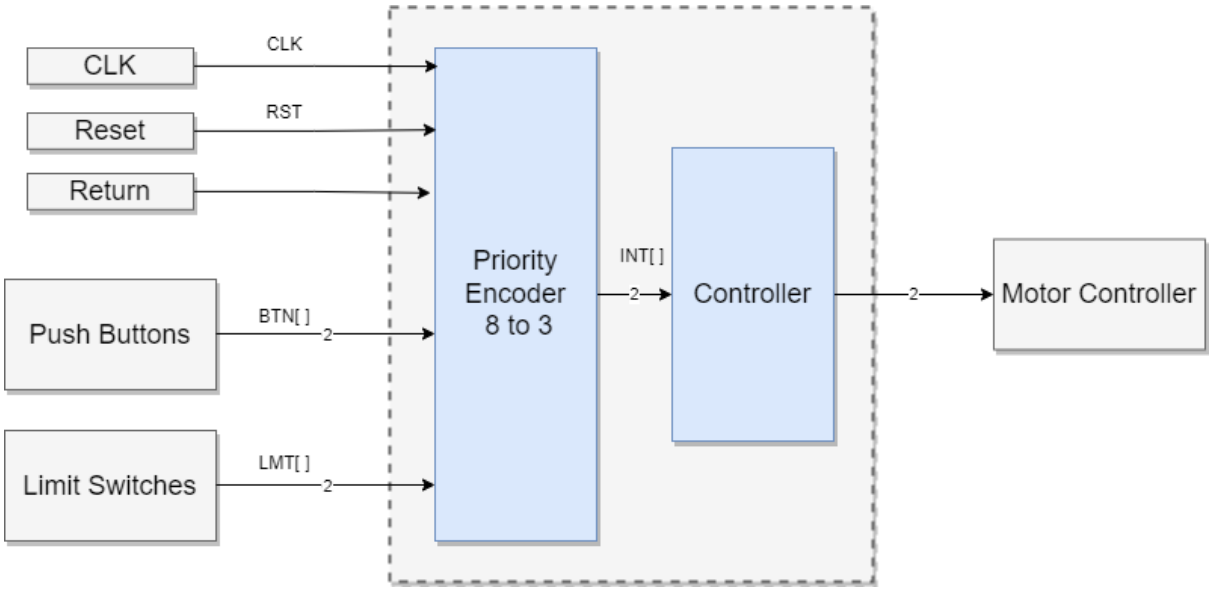
With this controller, There are two control panel required to located in a kitchen and on EDL as below.



1st panel is for kitchen and 2nd panel is for EDL. I made following initial assumptions for continue the deign tasks.

- EDL can transport goods between floors from/to kitchen and a floor only. It can't go intermediate floors.(Ex. It can't go from 1st floor to 2nd)
- When start the system (Powered up),EDL went to ground floor.
- If press the reset, controller goes to initial state.(If system not work properly, can press 'reset').
- Assume initially not pressed any button.
- After start the transition, Buttons can't change the destination, other than with press reset.

5. Internal Functional Components of the Controller



This controller has two components known as, priority encoder and a controller. Priority encoders reduce the number of inputs to the controller, and it maintain the input priority. Controllers hold the current state and make decisions according to the inputs.

Inputs:

Table 1

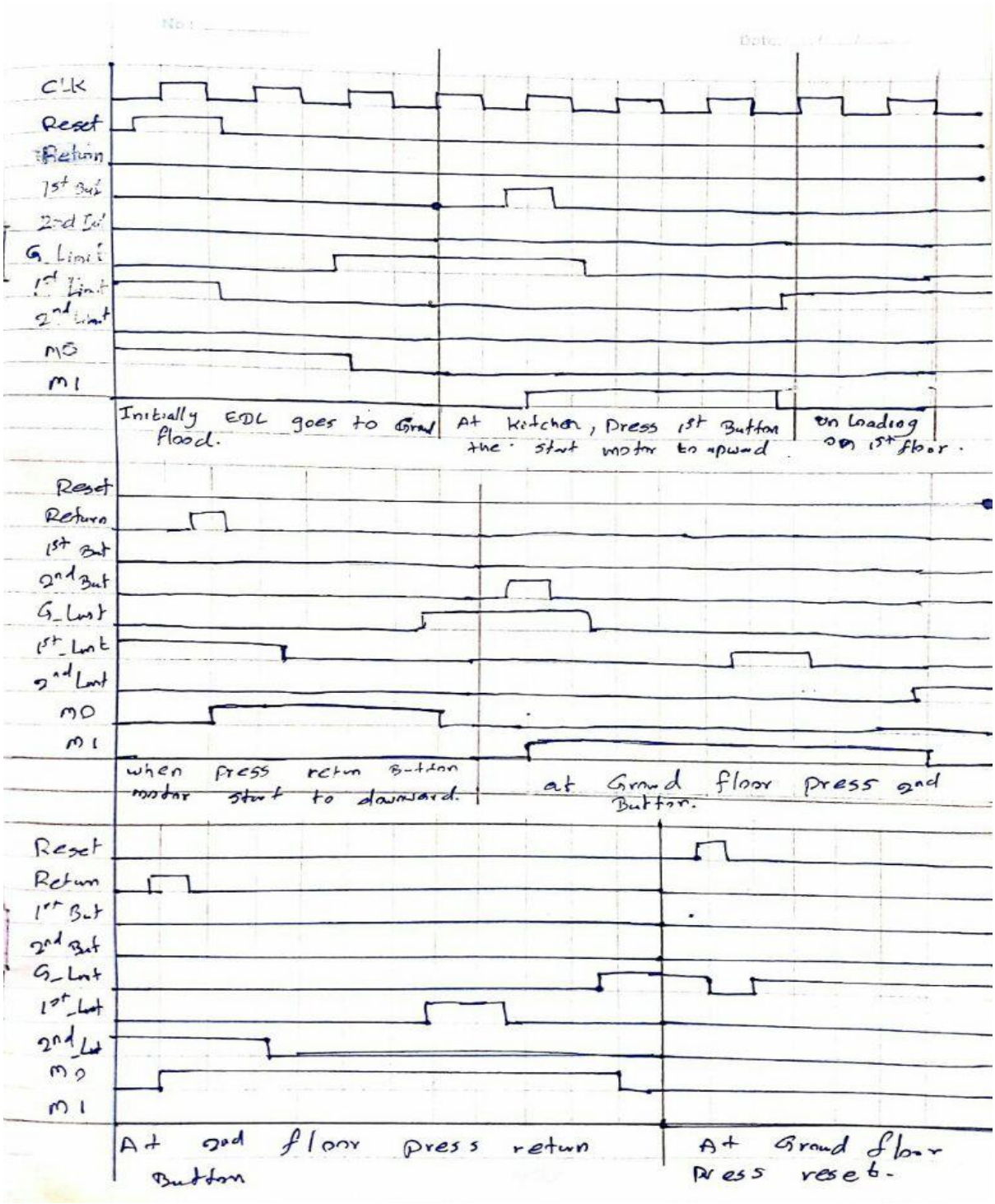
#	Input	Encoder Output Code (pqr)	Description
1	CLK	-	Synchronize Clock
2	RST	000	Reset - reset the EDL system.
3	P[0]	001	1st push button. Use to select 1st floor
4	P[1]	010	2nd push button. Use to select 2nd floor
5	L[0]	011	Ground Limit switch – activated when Lift reaches the kitchen
6	L[1]	100	1st Limit switch – activated when Lift reaches 1st Floor
7	L[2]	101	2nd Limit switch – activated when Lift reaches 2nd Floor
8	RET	110	Return – use to return lift to the kitchen
9	RUN	111	Nothing (Implement at the hardware level by using a push-up resistors)

Outputs

Table 2

#	Output	Code	Function
1	M[1],M[0]	00	Stop Motor
2	M[1],M[0]	01	Start Motor Upward
3	M[1],M[0]	10	Start Motor Downward
4	M[1],M[0]	11	Stop Motor

6. Timing Diagram for EDL



7. State Diagram

Here I used Implication Chart method to minimize the states.

7.1 State Diagram Before Minimizing

Table 3 State Description before minimizing

State	Description
II	Initial State
GG	Ground Floor State
F1	1st Floor State
F2	2nd Floor State
TG	Go to Ground Floor State
T1	Go to 1st Floor State
T2	Go to 2nd Floor State

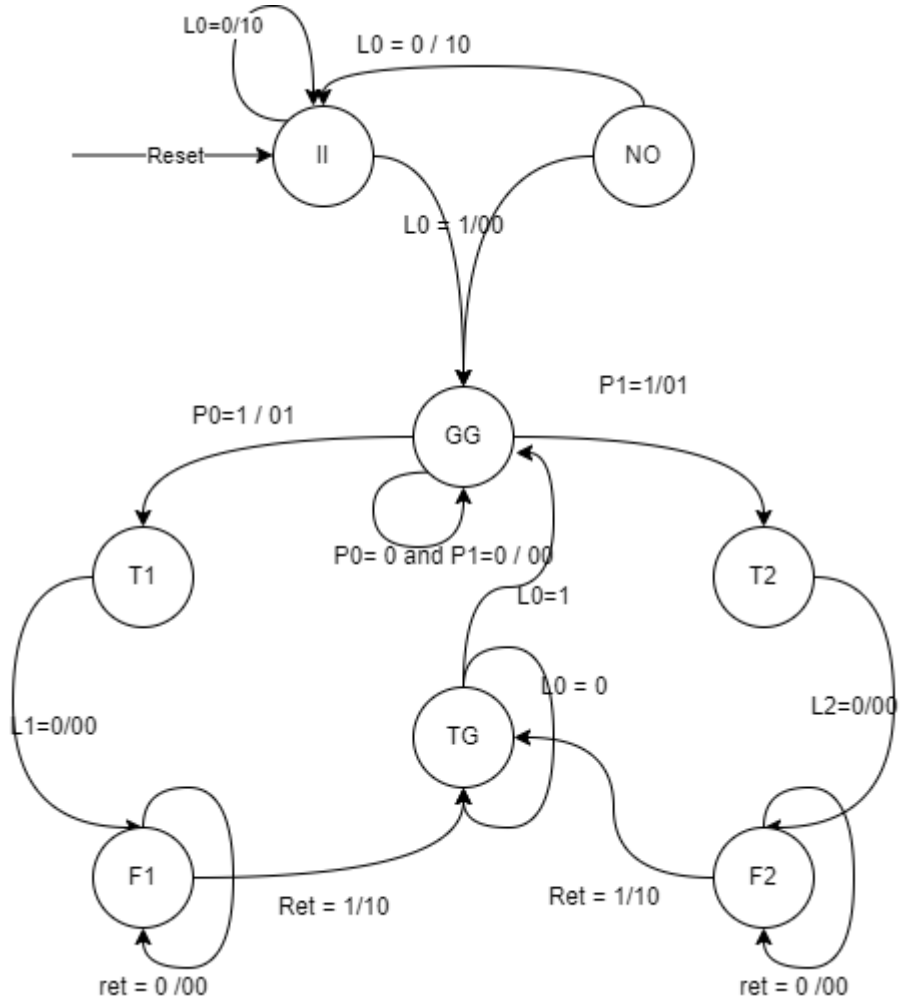
Table 4 State Table before minimizing

Present State	Next State, Output							
	RST	**P [0]**	**P [1]**	**L [0]**	**L [1]**	**L [2]**	**RET**	**RUN ***
II	II,00	-,-	-,-	GG,00	II,10	II,10	-,-	II,10
GG	II,00	T1,01	T2,01	GG,00	-,-	-,-	GG,00	GG,00
F1	II,00	F1,00	F1,00	-,-	F1,00	-,-	TG,10	F1,00
F2	II,00	F2,00	F2,00	-,-	-,-	F2,00	TG,10	F2,00
TG	II,00	TG,10	TG,10	GG,00	TG,10	TG,10	TG,10	TG,10
T1	II,00	T1,01	T1,01	T1,01	F1,00	-,-	T1,01	T1,01
T2	II,00	T2,01	T2,01	T2,01	T2,01	F2,00	T2,01	T2,01
NO*	II,10	II,10	II,10	GG,00	II,10	II,10	II,00	II,00

* If not at a given states, takes redundant state.

> Assuming initially not pressed any button.

State Diagram before minimizing



7.2 Minimizing States using Implication Chart

Table 5

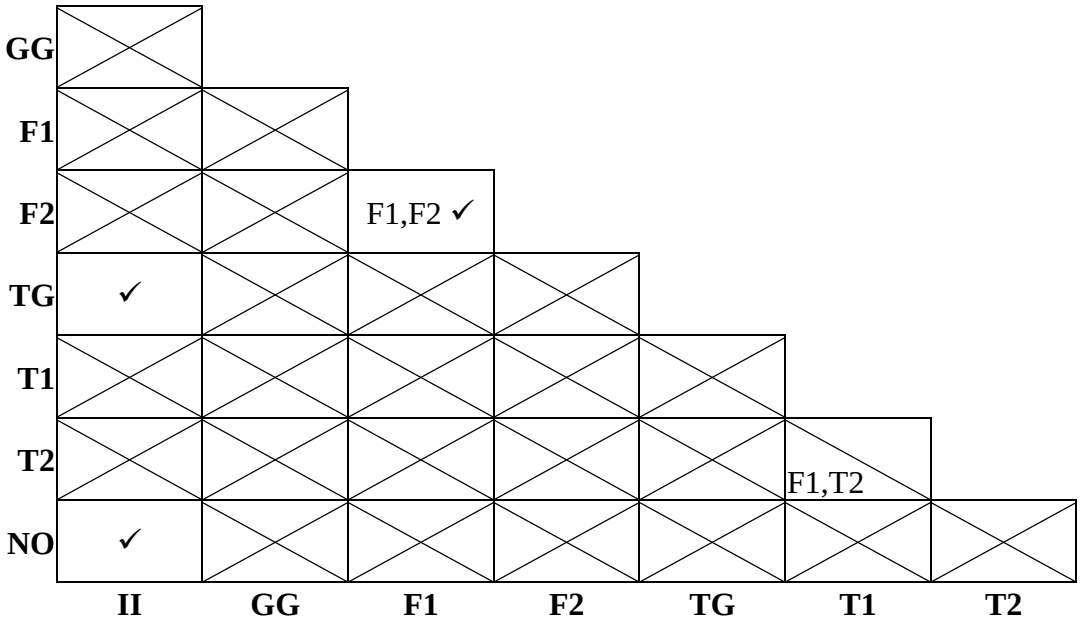
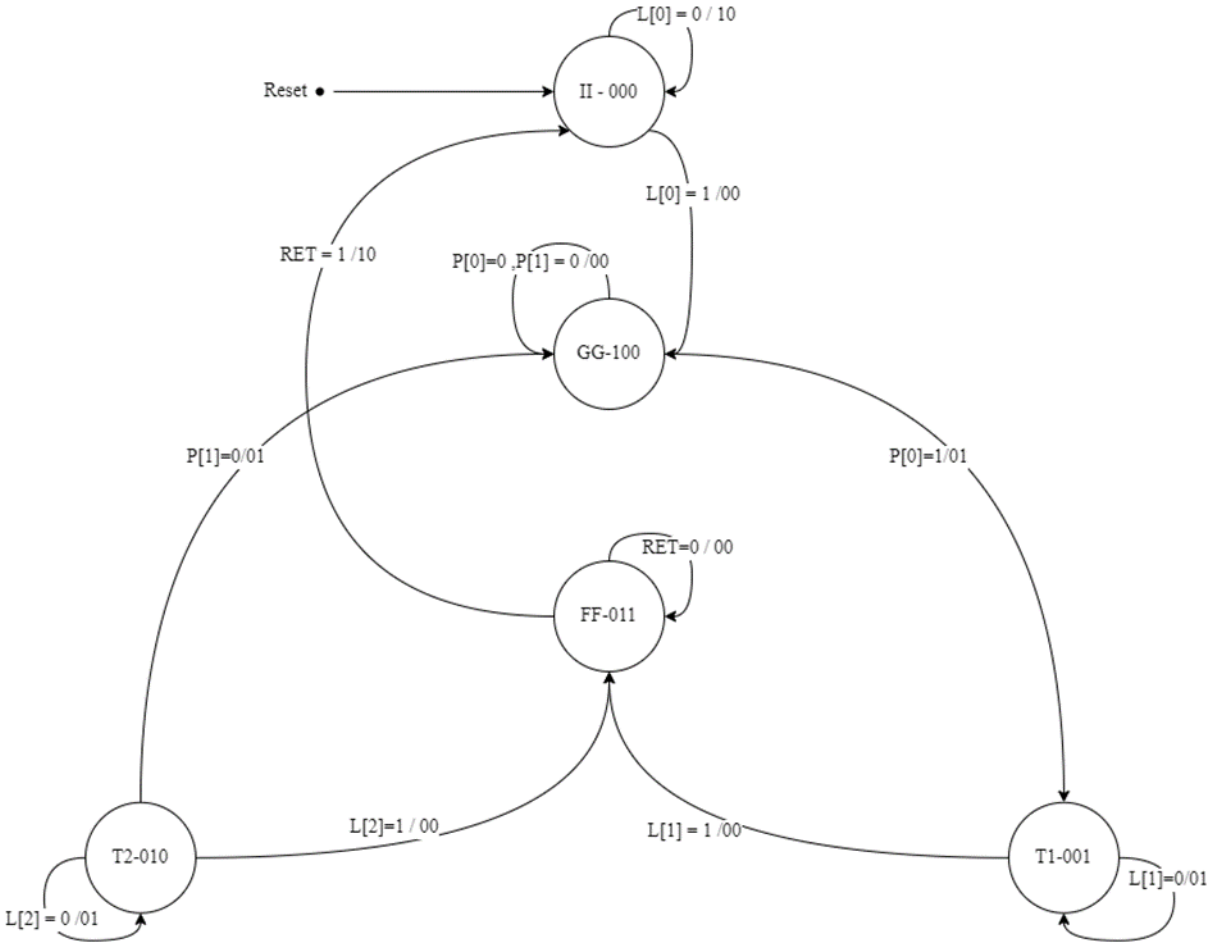


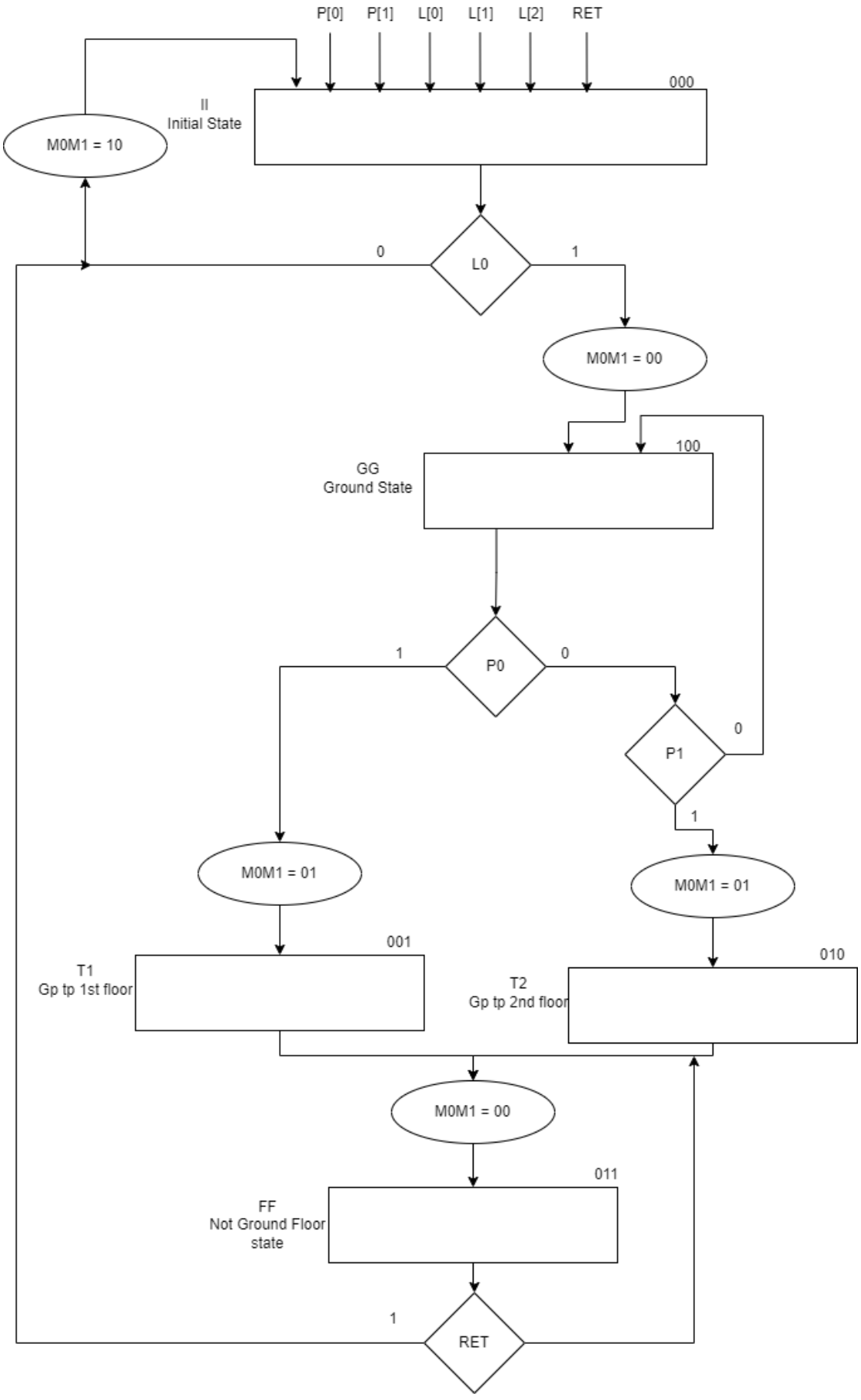
Table 6

	State	Description	Code
1	II	Initial State	000
2	GG	Ground Floor State	100
3	FF	NOT Ground Floor State	011
4	T1	Go to 1st Floor State	001
5	T2	Go to 2nd Floor State	010

7.3 State Diagram After Minimizing



8. ASM Chart for Controller



Required Number of Flip Flops for implement this circuit = 3 (Because $2^3 > 5 > 2^2$)

9. Derive Boolean Expression for each Flip Flops of Controller

Transition Table - Controller

Table 7

Present State			Input			Next State (D)			Output	
S3	S2	S1	P	Q	R	S3+	S2+	S1+	M2	M1
000			0	0	0	0	0	0	1	0
			0	0	1	0	0	0	1	0
			0	1	0	0	0	0	1	0
			0	1	1	1	0	0	0	0
			1	0	0	0	0	0	1	0
			1	0	1	0	0	0	1	0
			1	1	0	0	0	0	1	0
			1	1	1	0	0	0	1	0
001			0	0	0	0	0	1	0	1
			0	0	1	0	0	1	0	1
			0	1	0	0	0	1	0	1
			0	1	1	0	0	1	0	1
			1	0	0	0	1	1	0	0
			1	0	1	0	0	1	0	1
			1	1	0	0	0	1	0	1
			1	1	1	0	0	1	0	1
010			0	0	0	0	1	0	0	1
			0	0	1	0	1	0	0	1
			0	1	0	0	1	0	0	1
			0	1	1	0	1	0	0	1
			1	0	0	0	1	0	0	1
			1	0	1	0	1	1	0	0
			1	1	0	0	1	0	0	1
			1	1	1	0	1	0	0	1
011			0	0	0	0	1	1	0	0
			0	0	1	0	1	1	0	0
			0	1	0	0	1	1	0	0
			0	1	1	0	1	1	0	0
			1	0	0	0	1	1	0	0
			1	0	1	0	1	1	0	0
			1	1	0	0	0	0	1	0
			1	1	1	0	1	1	0	0
100			0	0	0	1	0	0	0	0
			0	0	1	0	0	1	0	1
			0	1	0	0	1	0	0	1
			0	1	1	1	0	0	0	0
			1	0	0	1	0	0	0	0
			1	0	1	1	0	0	0	0
			1	1	0	1	0	0	0	0
			1	1	1	1	0	0	0	0

Minimizing S3:

Minimal QuineMcCluskey Expression = s2's1'p'qr + s3q'r' + s3p

Minimizing S2:

Minimal QuineMcCluskey Expression = s1pq'r' + s2s1' + s2p' + s2r + s3p'qr'

Minimizing S1:

Minimal QuineMcCluskey Expression = s2's1 + s2pq'r + s1p' + s1q' + s1r + s3p'q'r

Minimizing M2:

Minimal QuineMcCluskey Expression = s3's2's1' + s2s1pqr'

Minimizing M1:

Minimal QuineMcCluskey Expression = s2's1p' + s2's1r + s2's1q + s2s1'p' + s2s1'r' + s2s1'q + s3p'q'r + s3p'qr'

Transition Table – Priority Encoder

Table 8

Inputs							Outputs		
RST	P[0]	P[1]	L[0]	L[1]	L[2]	RET	P	Q	R
1	x	x	x	x	x	x	0	0	0
0	1	x	x	x	x	x	0	0	1
0	0	1	x	x	x	x	0	1	0
0	0	0	1	x	x	x	0	1	1
0	0	0	0	1	x	x	1	0	0
0	0	0	0	0	1	x	1	0	1
0	0	0	0	0	0	1	1	1	0
0	0	0	0	0	0	0	1	1	1

Minimizing P:

P = RST’ . P0’ . P1’.L0’ [L1 + L1’.L2+L1’.L2’.RET]

P = RST’ . P0’ . P1’.L0’[L1+L2+ L1’.L2’.RET]

Minimizing Q:

Q = RST’ . P0’ . P1 + RET’ . P0’ . P1’ . L0 + RST’ . P0’. P1’.L0’.L1’.L2’

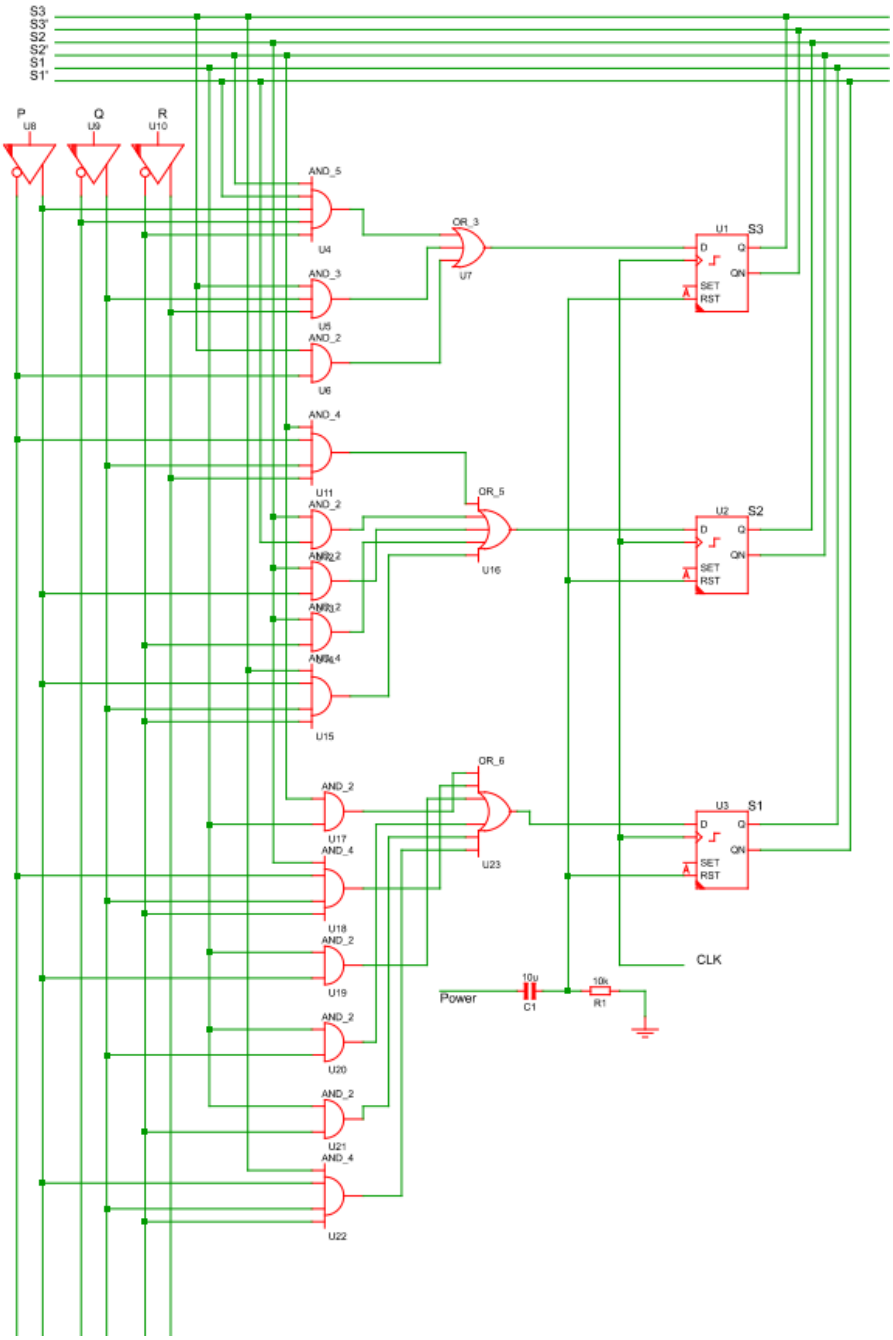
Minimizing R:

R = RST’ [P0 + P0’.P1’.L0 + P0’.P1’.L0’.L1’.L2 + P0’ P1’ L0’ L1’ L2’ RET]

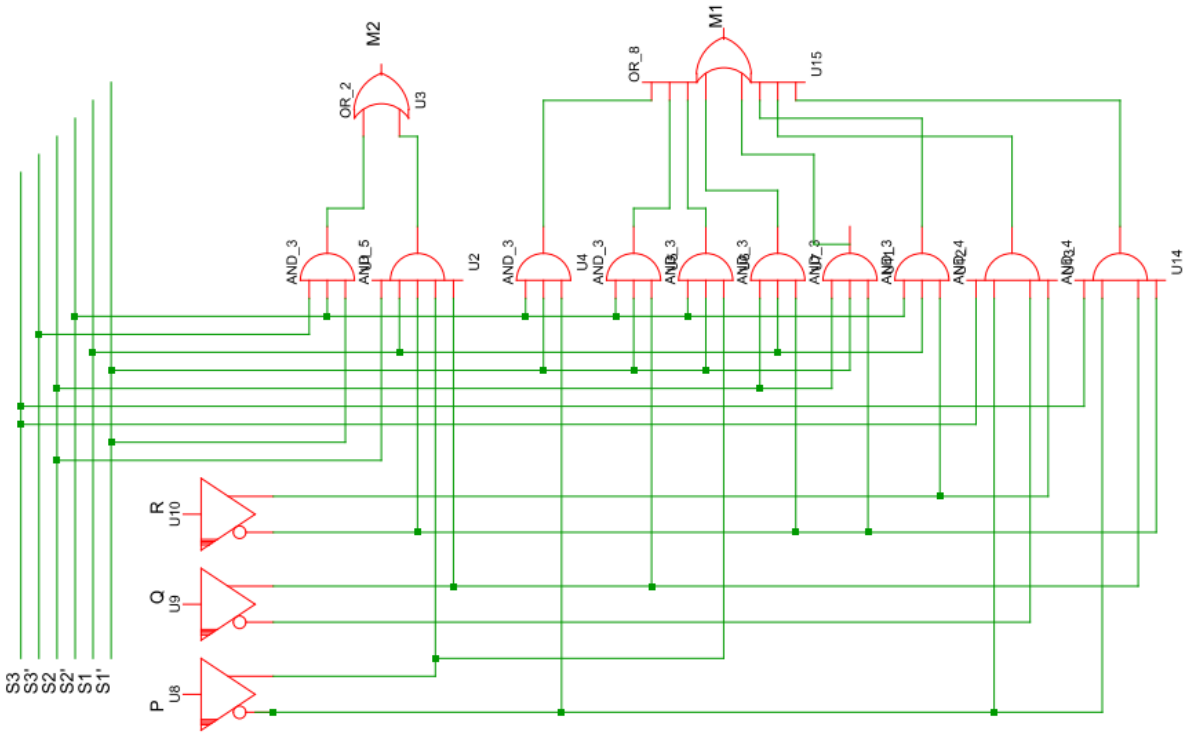
R = RST’ [P0 + P1’.L0’ + P0’.P1’.L0’.L1’ [L2+RET]]

10. FSM Circuits

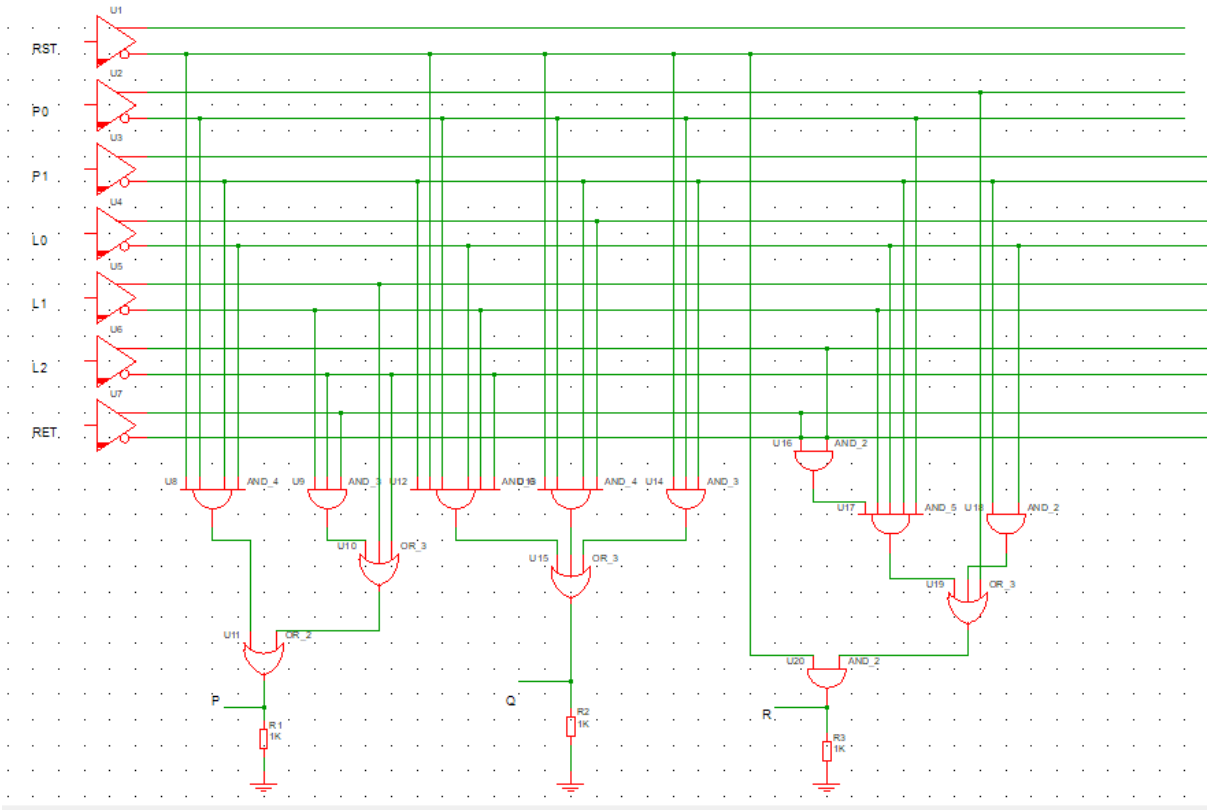
10.1 Circuit of controller's States



10.2 Circuit of Controller's output



10.3 Circuit of Priority Encoder



- Controller has 5 states and developed by using D Flip flop.
- Output also depended on the current states and input (Merely machine).
- Included Power-on-reset circuit. So, when power up the system, system goes to initial state.

11. Behavioral Model VHDL Code

11.1 Code For Encoder

```
----------------------------------------------------------------------------
library IEEE; --Not a Picture
use IEEE.STD_LOGIC_1164.ALL;
use ieee.NUMERIC_STD.all;

entity Encorder is
    Port ( BUT1 : in  STD_LOGIC;
           BUT2 : in  STD_LOGIC;
           G_lmt : in  STD_LOGIC;
           lmt_1 : in  STD_LOGIC;
           lmt_2 : in  STD_LOGIC;
           ret : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           Y : out  STD_LOGIC_VECTOR (2 downto 0));
end Encorder;

architecture Behavioral of Encorder is

begin

process(BUT1,BUT2,G_lmt,lmt_1,lmt_2,ret,reset)
begin

if (reset = '1') then Y <= "000";
elsif (ret = '1') then Y <= "001";
elsif (BUT1 = '1') then Y<="010";
elsif (BUT2 = '1') then Y<="011";
elsif (G_lmt = '1') then Y<="100";
elsif (lmt_1 = '1') then Y<="101";
elsif (lmt_2 = '1') then Y<="110";
else Y<="111";

end if;
end process;



end Behavioral;
```

11.2 Code for State Controller

```
library ieee; -- Not a picture
use IEEE.STD_LOGIC_1164.ALL;


entity EDL_2 is
port(
clk: in std_logic;
encod : in std_logic_vector(2 downto 0);
M :out std_logic_vector(1 downto 0)
);
end EDL_2;

architecture arch_EDL_2 of EDL_2 is
type state_type is (II,GG,FF,T1,T2);
signal state_reg, state_next : state_type;

begin
    process(clk)
    begin
        if encod = "000" then state_reg <=II;
        elsif (rising_edge(clk)) then state_reg <= state_next;
        end if;
    end process;


    process(encod,state_reg)
    begin
        state_next <= state_reg;
        M <= "00";

        case state_reg is
            when II =>
            if (encod = "100") then M <= "00";
                state_next <= GG;
            else
                M <= "10";
                state_next <= II;
            end if;

            when GG =>
            if (encod = "010") then M <= "01";
                state_next <= T1;
            elsif (encod = "011") then M <= "01";
                state_next <= T2;
            else
                M <= "00";
                state_next <= GG;
            end if;

            when FF =>
            if(encod = "001") then M <= "10";
                state_next <= II;
            else
                M <= "00";
                state_next <= FF;
            end if;

            when T1 =>
            if(encod = "101") then M <= "00";
                state_next <= FF;
            else
                M <= "01";
                state_next <= T1;
            end if;
```

```
            when T2 =>
            if(encod = "110") then M <= "00";
                state_next <= FF;
            else
                M <= "01";
                state_next <= T2;
            end if;
            end case;
            end process;



    end arch_EDL_2;
```

11.3 Structured Code

```
library ieee; --not a Picture
use ieee.std_logic_1164.all;

entity EDL is
    port(
    clk, reset :in std_logic;
    BUT1,BUT2,G_lmt,lmt_1,lmt_2,RET :in std_logic;          -- p - push button , l - limit
switch , -M motor, ret-return
    M : out std_logic_vector(1 downto 0)
    );
end EDL;

architecture arch_EDL of EDL is

signal P : std_logic_vector(2 downto 0);      -- internal signals

component EDL_2 is
    port(
    clk: in std_logic;
    encod : in std_logic_vector(2 downto 0);
    M :out std_logic_vector(1 downto 0)
    );

end component;
component Encorder is

     Port ( BUT1 : in  STD_LOGIC;
            BUT2 : in  STD_LOGIC;
            G_lmt : in  STD_LOGIC;
            lmt_1 : in  STD_LOGIC;
            lmt_2 : in  STD_LOGIC;
            ret : in  STD_LOGIC;
            reset : in  STD_LOGIC;
            Y : out  STD_LOGIC_VECTOR (2 downto 0));

 end component;

 -- port map

 begin
    controller :    EDL_2 port map (encod => P, M =>M, clk =>clk);
    encod:  Encorder port map (BUT1 => BUT1,BUT2 => BUT2,G_lmt => G_lmt, lmt_1 =>
lmt_1,
    lmt_2 =>lmt_2, ret => ret, reset => reset, Y => P);
 end arch_EDL;
```

12. Test Bench Code for Controller

12.1 Test Bench for Encoder

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.NUMERIC_STD.all;
    -- Add your library and packages declaration here ...

entity encorder_tb is
end encorder_tb;

architecture TB_ARCHITECTURE of encorder_tb is
    -- Component declaration of the tested unit
    component encorder
    port(
        BUT1 : in STD_LOGIC;
        BUT2 : in STD_LOGIC;
        G_lmt : in STD_LOGIC;
        lmt_1 : in STD_LOGIC;
        lmt_2 : in STD_LOGIC;
        ret : in STD_LOGIC;
        reset : in STD_LOGIC;
        Y : out STD_LOGIC_VECTOR(2 downto 0) );
    end component;

    -- Stimulus signals - signals mapped to the input and inout ports of tested entity
    signal BUT1 : STD_LOGIC;
    signal BUT2 : STD_LOGIC;
    signal G_lmt : STD_LOGIC;
    signal lmt_1 : STD_LOGIC;
    signal lmt_2 : STD_LOGIC;
    signal ret : STD_LOGIC;
    signal reset : STD_LOGIC;
    -- Observed signals - signals mapped to the output ports of tested entity
    signal Y : STD_LOGIC_VECTOR(2 downto 0);

    -- Add your code here ...

begin

    -- Unit Under Test port map
    UUT : encorder
        port map (
            BUT1 => BUT1,
            BUT2 => BUT2,
            G_lmt => G_lmt,
            lmt_1 => lmt_1,
            lmt_2 => lmt_2,
            ret => ret,
            reset => reset,
            Y => Y
        );

    reset <= '1', '0' after 50ns;
    BUT1 <= '0', '1' after 100 ns, '0' after 150ns;
    BUT2 <= '0', '1' after 200 ns, '0' after 250ns;
--  G_lmt <= '0', '1' after 100ns , '0' after 150ns;


end TB_ARCHITECTURE;

configuration TESTBENCH_FOR_encorder of encorder_tb is
    for TB_ARCHITECTURE
        for UUT : encorder
            use entity work.encorder(behavioral);
        end for;
```

```
    end for;
end TESTBENCH_FOR_encorder;
```

12.2 Test Bench for State Controller

```
library ieee;
use ieee.std_logic_1164.all;

    -- Add your library and packages declaration here ...

entity edl_2_tb is
end edl_2_tb;

architecture TB_ARCHITECTURE of edl_2_tb is
constant clk_period : time := 10 ns; --defining clock
    -- Component declaration of the tested unit
    component edl_2
    port(
        clk : in STD_LOGIC;
        encod : in STD_LOGIC_VECTOR(2 downto 0);
        M : out STD_LOGIC_VECTOR(1 downto 0) );
    end component;

    -- Stimulus signals - signals mapped to the input and inout ports of tested entity
    signal clk : STD_LOGIC;
    signal encod : STD_LOGIC_VECTOR(2 downto 0);
    -- Observed signals - signals mapped to the output ports of tested entity
    signal M : STD_LOGIC_VECTOR(1 downto 0);

    -- Add your code here ...

begin

    -- Unit Under Test port map
    UUT : edl_2
        port map (
            clk => clk,
            encod => encod,
            M => M
        );

    clock_process: process
    begin
        clk <='0';
        wait for clk_period/2;
        clk <= '1';
        wait for clk_period/2;
    end process;

    encod <= "000", "100" after 100ns, "110" after 200ns;

end TB_ARCHITECTURE;

configuration TESTBENCH_FOR_edl_2 of edl_2_tb is
    for TB_ARCHITECTURE
        for UUT : edl_2
            use entity work.edl_2(arch_edl_2);
        end for;
    end for;
end TESTBENCH_FOR_edl_2;
```

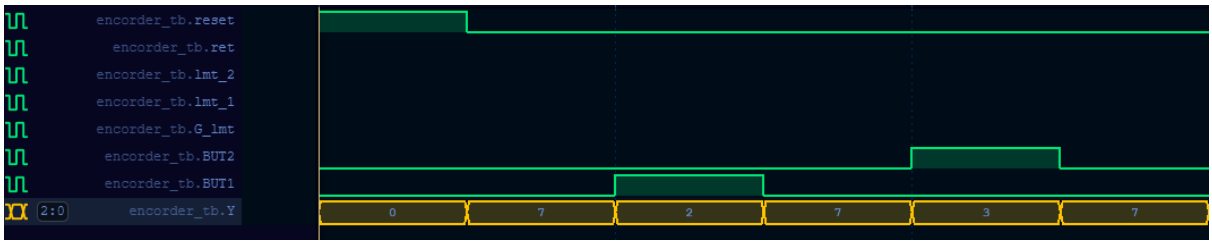
12.3 Test Bench for the Controller

```
library ieee;
use ieee.std_logic_1164.all;

    -- Add your library and packages declaration here ...

entity edl_tb is
end edl_tb;

architecture TB_ARCHITECTURE of edl_tb is
-- Component declaration of the tested unit
        constant clk_period : time := 10 ns; --defining clock
    component edl
    port(
        clk : in STD_LOGIC;
        reset : in STD_LOGIC;
        BUT1 : in STD_LOGIC;
        BUT2 : in STD_LOGIC;
        G_lmt : in STD_LOGIC;
        lmt_1 : in STD_LOGIC;
        lmt_2 : in STD_LOGIC;
        RET : in STD_LOGIC;
        M : out STD_LOGIC_VECTOR(1 downto 0) );
    end component;

    -- Stimulus signals - signals mapped to the input and inout ports of tested entity
    signal clk : STD_LOGIC;
    signal reset : STD_LOGIC;
    signal BUT1 : STD_LOGIC;
    signal BUT2 : STD_LOGIC;
    signal G_lmt : STD_LOGIC;
    signal lmt_1 : STD_LOGIC;
    signal lmt_2 : STD_LOGIC;
    signal RET : STD_LOGIC;
    -- Observed signals - signals mapped to the output ports of tested entity
    signal M : STD_LOGIC_VECTOR(1 downto 0);

    -- Add your code here ...

begin

    -- Unit Under Test port map
    UUT : edl
        port map (
            clk => clk,
            reset => reset,
            BUT1 => BUT1,
            BUT2 => BUT2,
            G_lmt => G_lmt,
            lmt_1 => lmt_1,
            lmt_2 => lmt_2,
            RET => RET,
            M => M
        );

    -- Add your stimulus here ...
    clock_process: process
    begin
        clk <='0';
        wait for clk_period/2;
        clk <= '1';
        wait for clk_period/2;
    end process;
```

```
    stim: process
    begin
        ---------------Case 00-----------------------------
         reset <= '1';  --Initially reset occur at hardware
         wait for 200ns;
         reset <='0';
         wait for 300ns;       -- Initillay find lift go to kitchen
        G_lmt <= '1';   --Lift  reach the Ground Floor
        wait for 100ns;
        assert M = "00" report "Check Case 00 - 1"  severity error;
        wait for 400ns; -- Remain at kitchen

        ----------------------Case 01------------------------

        BUT1 <= '1'; --Press 1st button
        wait for 100ns;
        BUT1 <='0';
        G_lmt <= '0';
        wait for 100ns;
        assert M = "01" report "Check Case 01 - 1"  severity error;
        wait for 300ns;
        lmt_1 <= '1'; --Reach 1st floor
        wait for 500ns;
        assert M = "00" report "Check Case 01 - 2"  severity error;
        RET <= '1';  --press return button
        wait for 100ns;
        RET <= '0';
        lmt_1 <='0';
        wait for 400ns;
        assert M = "10" report "Check Case 01 - 3"  severity error;
        G_lmt <= '1'; -- Reach kitchen
        assert M = "00" report "Check Case 01 - 4"  severity error;

        wait;

        end process;



end TB_ARCHITECTURE;

configuration TESTBENCH_FOR_edl of edl_tb is
    for TB_ARCHITECTURE
        for UUT : edl
            use entity work.edl(arch_edl);
        end for;
    end for;
end TESTBENCH_FOR_edl;
```

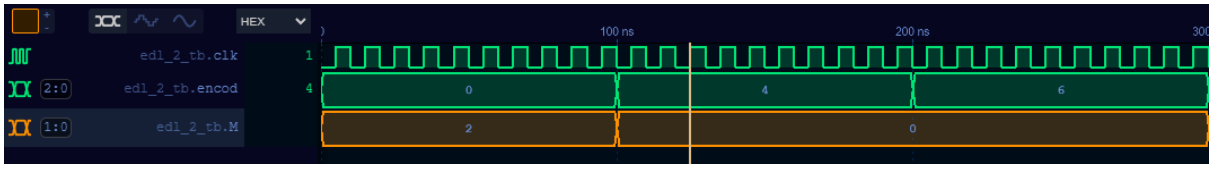
13. Simulation Results

13.1 For Encoder



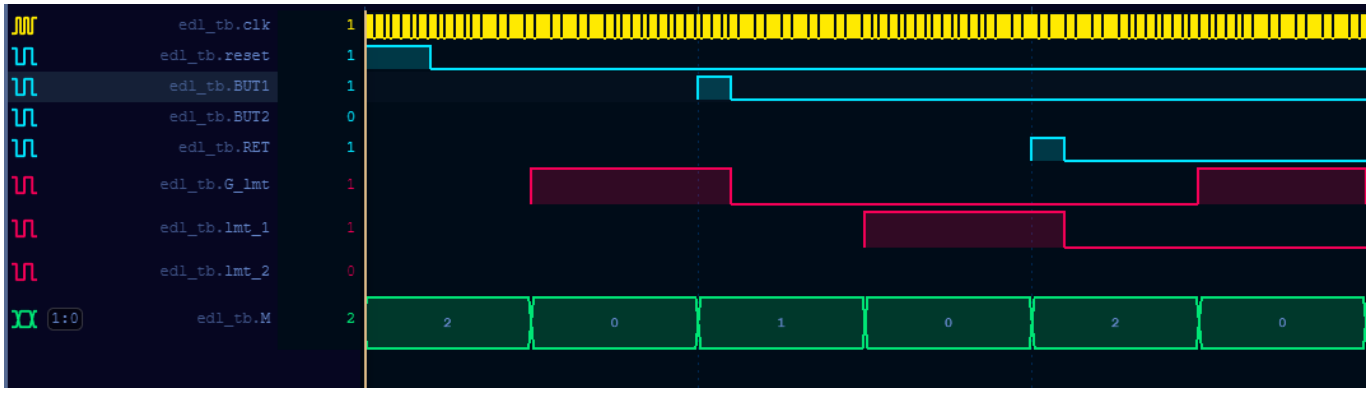
As shown in above graph, Encoder output is varying with different input as expected.

13.2 For State Controller



As shown in above graph, State control provide motor output, according to the encoder input. That's mean it change its state properly.

13.3 For controller

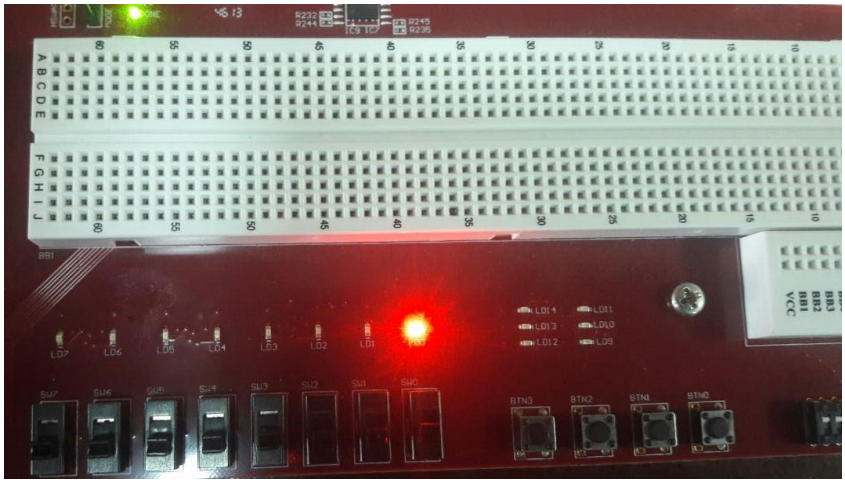


Initially reset is hold sometimes trough hardware (Included it in the circuit diagram). Then the motor output provided as downward (M[]=2) until reach the kitchen (G_lmt). Then motor was stopped. Then press 1st floor button (BUT1). Then motor was start to upward direction (M[]=1). The ground limit went to zero. When reach 1st floor 1st limit switch(lmt_1) turn on. Then Motor was stopped. Finally press the return button(RET), Then motor goes to downward until reach the ground limit.

Blue color indicate the user pressing buttons. Red color indicates the system inputs such as limit switches. Green color indicates the motor output.

14. Implementation

Here I implement the designed controller on Anvyls Spartan 6 FPGA board in the Lab. I used two LEDs to indicates the motor controller output and 3 buttons as push buttons. Also I used 3 slide switches as limit switches. It worked properly and provided expected outputs.



15. Conclusion

- During this laboratory experiment, I designed a sequential circuit for a simple lift. I take 3 attempts before making this design and learned lot of about state diagram drawing, ASM chart drawing and the VHDL coding techniques. Also, it provided a knowledge about the procedure of designing a digital system from sketch level. When implement design by using VHDL language, it's very convenient if use behavioral coding instead using structural coding specially for complex designs. Also, I learned about the state's reduction techniques and some best practices that using in circuit designing such as implementing reset via hardware and power on self-resets. Finally, I used test benches for verifying the design and implement a prototype on an FPGA for hardware-level verification.