



SLTC
Research University

EEE212-Digital Logic Circuit Design

Design of An Elevator Controller Report

Shakil Arifeen - 21UG0057

B. Tharish - 21UG0099

Ometh Perera - 21UG0108

Kaminda Perera - 21UG0153

Table of Contents

1. Abstract	2
2. Introduction	2
3. Operation Principle	2
4. State Diagram representing the movement of elevator from each floor.....	3
4.1 State Assignment	3
4.2 State table	4
5. State Diagram For The Opening And Closing Of The Elevator Door	5
6. Operation of Elevator Control.....	6
7. VHDL Codes and ModelSim Altera Simulations for the Elevator Controller	8
7.1 Floor Counter	8
7.2 Comparator	10
7.3 Code Register	12
7.4 Timer	14
7.5 Seven – Segment.....	16
7.6 Call\Request FF	18
7.7 Elevator Control	20
8. State Diagram With Emergency Button	22
9. Conclusion	23
10. References.....	23

1. Abstract

The following report consists of the designing of an elevator control system designed for a 4-story building with 5 floor buttons (0 - basement, 1st floor, 2nd floor, 3rd floor & 4th floor) by the aid of digital logic electronic components. The system has been modelled using VHDL and the system operation has been verified by RTL simulation.

2. Introduction

An elevator (also known as a lift) is a type of vertical transportation device that moves people or goods between floors of a building. Elevators are typically powered by electric motors and use cables, pulleys, and counterweights to move a car or platform up and down along a guide rail or track.

Elevators are a common feature in multi-story buildings and are essential for providing convenient and accessible transportation between floors. They are also used in other structures, such as towers and bridges, to move people and goods to different levels.

Elevators are equipped with safety features, such as emergency stop buttons, to ensure the safety of passengers. They are also regulated by building codes to ensure that they are designed, installed, and maintained to meet certain standards for safety and reliability.

The elevator system in this report is designed to allow passengers to request a lift to a specific floor, and for the lift to transport them to their desired floor in a safe and efficient manner. The system consists of several components, including a floor counter, a call comparator, a code register, a timer, a seven-segment display, and a toggle.

To design the required system, the following assumptions have been made;

- Initially, the elevator rests on the ground floor from where the elevator starts its cycle.
- A cycle is a period starting from the point at which the passenger calls the elevator to a specific floor to the point the passenger is delivered to the requested floor.
- For simplicity, it is considered there is only one floor call and one floor request for each elevator cycle.

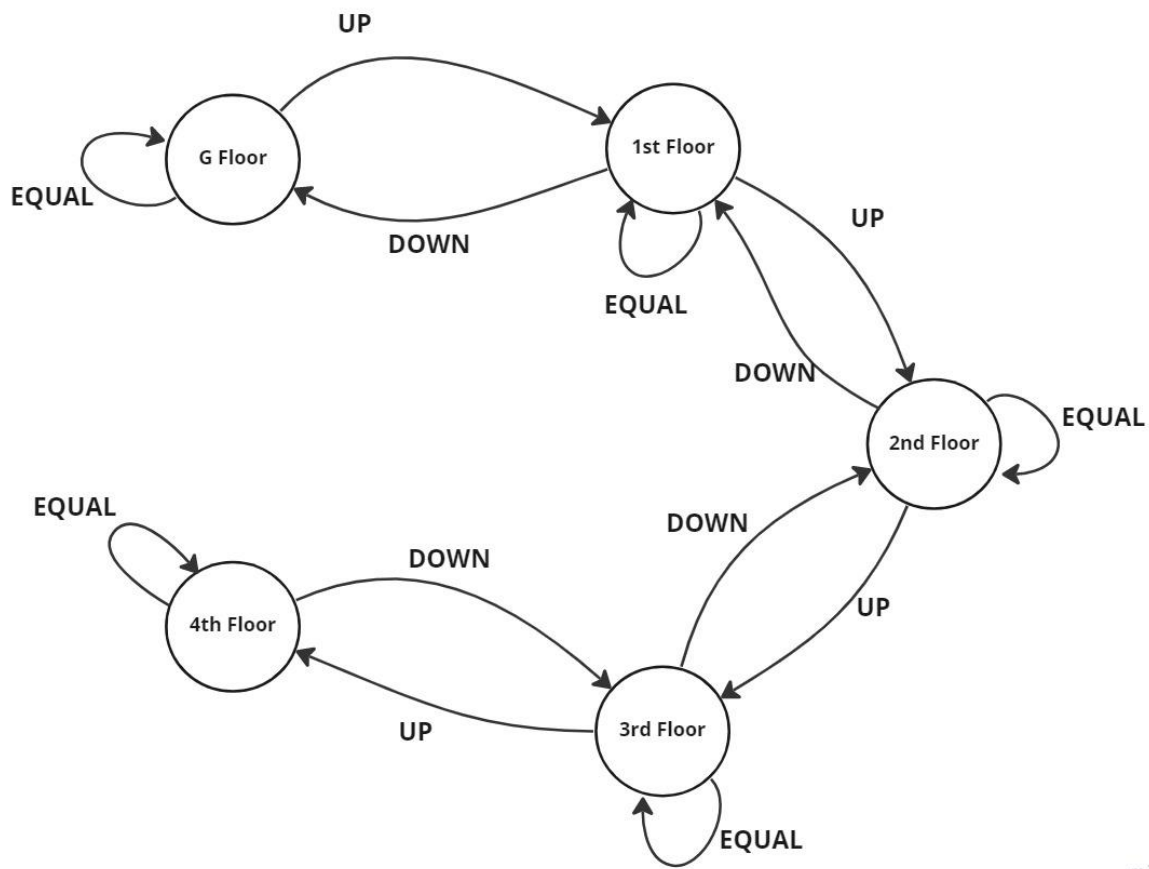
3. Operation Principle

The elevator is called to the floor which the passenger wants to be picked up from by pressing a button placed in that specific floor. The system collects the called floor value and compares it to the value of the floor the elevator currently is. In the system included in this report, since the cycle starts from the ground floor this would always mean the called floor is either itself or a floor above. Hence the elevator stays at its position or moves upward. If the elevator is to move upwards, it moves up one floor and once again compares its current floor value with the floor value it was called to. The comparing happens through the comparator of the system. Once the current floor value and called floor value are equal after having moved the elevator upwards, the elevator stops and opens the door.

From the point at which the elevator stops and opens the door, a timer is initiated 10 seconds holding the door open. After the said timer reaches 10 seconds the elevator door closes. This is a waiting time added as a safety measure to the passengers entering and exiting the elevator.

After the passenger enters the elevator, he/she can request the floor they want to go to. Depending on if the requested floor is greater than or less than to the current floor the comparator sends a signal to either move the elevator up or down a floor. Once again after comparison of the two values, if the values are equal the elevator stops, opens the door, and activates the timer.

4. State Diagram representing the movement of elevator from each floor



miro

4.1 State Assignment

States (Q2_Q1_Q0)				
Ground Floor	1 st Floor	2 nd Floor	3 rd Floor	4 th Floor
000	001	010	011	100

A comparator compares the current state of the floor and the requested floor and gives an output which makes the floor move either upwards or downwards.

INPUTS	EQUAL	DOWN	UP
UD	00	01	10

4.2 State table

	Present State			Next State		Output		
	Q2	Q1	Q0	U	D	Q2+	Q1+	Q0+
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	X	X	X
2	0	0	0	1	0	0	0	1
3	0	0	0	1	1	X	X	X
4	0	0	1	0	0	0	0	1
5	0	0	1	0	1	0	0	0
6	0	0	1	1	0	0	1	0
7	0	0	1	1	1	X	X	X
8	0	1	0	0	0	0	1	0
9	0	1	0	0	1	0	0	1
10	0	1	0	1	0	0	1	1
11	0	1	0	1	1	X	X	X
12	0	1	1	0	0	0	1	1
13	0	1	1	0	1	0	1	0
14	0	1	1	1	0	1	0	0
15	0	1	1	1	1	X	X	X
16	1	0	0	0	0	1	0	0
17	1	0	0	0	1	0	1	1
18	1	0	0	1	0	1	0	0
19	1	0	0	1	1	X	X	X
20	1	0	1	0	0	X	x	X
21	1	0	1	0	1	X	x	X
22	1	0	1	1	0	X	x	X
23	1	0	1	1	1	X	x	X
24	1	1	0	0	0	x	x	X
25	1	1	0	0	1	x	x	X
26	1	1	0	1	0	x	x	X
27	1	1	0	1	1	x	x	X
28	1	1	1	0	0	x	x	X
29	1	1	1	0	1	x	x	X
30	1	1	1	1	0	x	x	X
31	1	1	1	1	1	x	x	X

- Present states Q2_Q1_Q0 represent the current floor of the elevator.
- Q2+_Q1+_Q0+ represents the next state which depends on the value of U and D.
- UD = 10 means the elevator should go up by one floor, while UD= 01 means that the elevator should go down by one floor.
- If the compared floor values are equal the floor remains on the same floor.
- Invalid states are represented by XXX

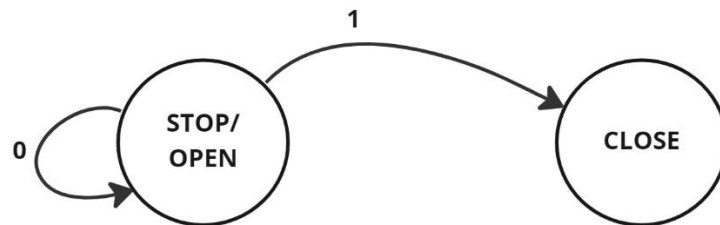
The Output States are represented by the following equations

$$Q0+ \rightarrow Q0'.D + Q2'.Q0'.U + Q0.U'.D'$$

$$Q1+ \rightarrow Q2.D + Q1'.Q0.U + Q1.Q0'.D' + Q1.Q0.U'$$

$$Q2+ \rightarrow Q2.D' + Q1.Q0.U$$

5. State Diagram For The Opening And Closing Of The Elevator Door



miro

The input for this comes from the timer (counts to 10 seconds – waiting time) which is enabled when the current floor and requested floor are equal.

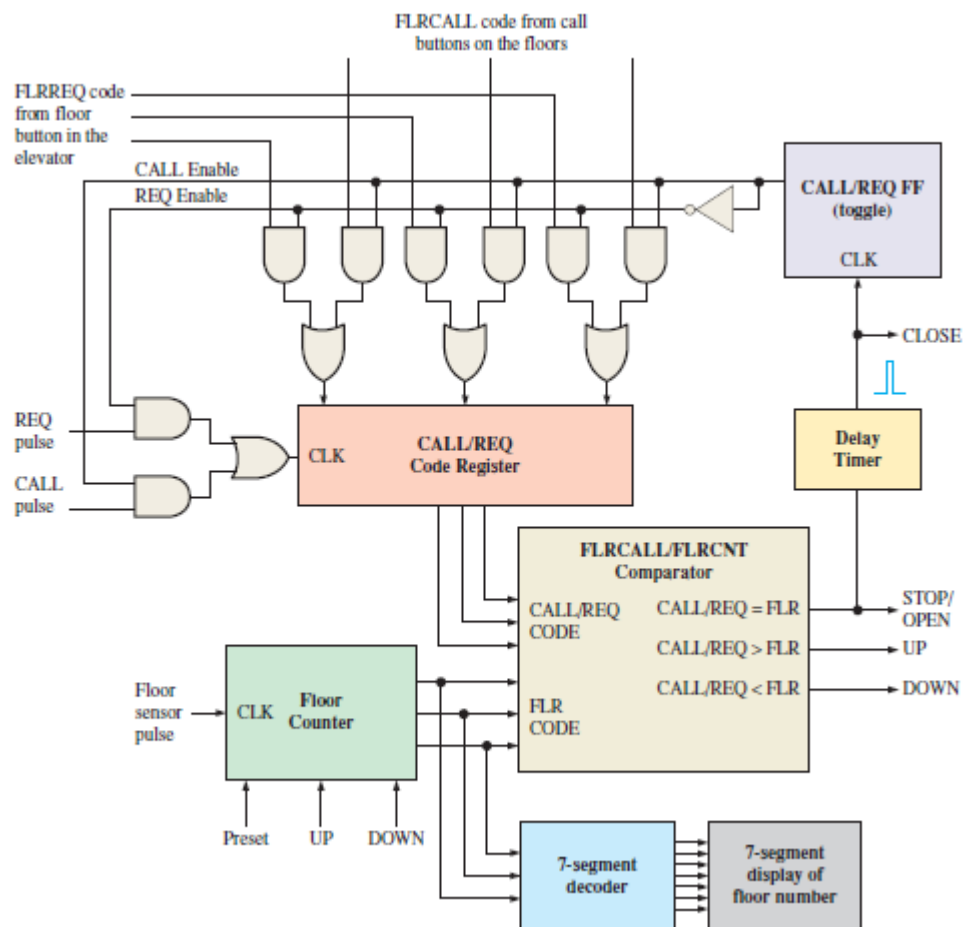
State	Input	Next State
0 (Stop/Open)	0	0
0 (Stop/Open)	1	1
1 (Close)	0	x
1 (Close)	1	x

- When the current state is 0 (Stop/Open) an input of 0 makes the elevator door remain in the same state. Therefore, the next state is 0 (Stop/Open). An input of 1 makes the elevator door close. Therefore, the next state is 1 (Close)
- When the current state is 1 (Close) there is no input from the timer as it only gets activated when the comparator sends out a signal to stop and open the door when the current floor and called/requested floor are equal.

After drawing the Karnaugh Map for this truth table it gives the following equation.

Next State → Input

6. Operation of Elevator Control



-Elevator Controller Logic Diagram-

- This displays the logic diagram for the elevator controller. Either a floor request or a floor call (FLRCALL) initiates elevator operation (FLRREQ). Keep in mind that FLRCALL refers to the act of calling the elevator to a certain floor. In FLRREQ, a passenger in the elevator, ask to be taken to a particular floor. This streamlined operation is built on the CALL/REQ sequence, which is composed of a call, a request, and another call.
- The 3-bit codes FLRCALL and FLRREQ designate particular floors. The unique 3-bit code for that floor is placed on the inputs of the CALL/REQ code register when a call button is pressed on a particular floor, and a CALL pulse is created to enter the code into the register. When a request button is pressed inside the elevator, an identical procedure takes place. An REQ pulse is created to save the code in the CALL/REQ code register after it has been input.
- The difference between a call and a request is not recognized by the elevator. The comparator assesses whether the elevator's current floor is larger, lower, or equal to the destination floor number. Following this comparison, the elevator motor control is given one of three commands: UP, DOWN, or OPEN. The floor counter is either increased at each floor as the elevator ascends or decreased at each floor as it descends as it travels toward the intended floor. A STOP/OPEN order is sent to both the door control and the elevator motor control once the elevator has arrived at the designated floor.
- The delay timer sends a CLOSE signal to the elevator door control after a certain amount of time. As previously stated, this elevator's design may only call and request one floor at a time.

7. VHDL Codes and ModelSim Altera Simulations for the Elevator Controller

7.1 Floor Counter

This VHDL code defines an entity called "FLOORCOUNTER" with three input ports: "UP", "DOWN", and "Sensor", and one output port "FLRCODE". The "UP" and "DOWN" inputs are used to control the direction of the floor counter, and the "Sensor" input is used to trigger the counting. The "FLRCODE" output is a 3-bit vector that represents the current floor count.

The code also defines an internal signal called "FloorCnt", which is an integer ranging from 0 to 4. This signal keeps track of the current floor count.

The code then defines a process that is sensitive to changes on the "UP", "DOWN", and "Sensor" inputs. Inside the process, the "FLRCODE" output is set to the current value of "FloorCnt" converted to a 3-bit vector. Then, if the "Sensor" input is '1', the code checks the values of the "UP" and "DOWN" inputs to determine the direction of the floor counter. If "UP" is '1' and "DOWN" is '0', the floor counter is incremented by 1. If "UP" is '0' and "DOWN" is '1', the floor counter is decremented by 1.

Finally, the code includes two conditional statements to check if the floor count has gone above 4 or below 0. If either of these conditions is true, the floor count is reset to the appropriate limit (4 or 0).

Overall, this code implements a simple floor counter that increments or decrements in response to the "UP" and "DOWN" inputs and is reset to the appropriate limit if the count goes out of range. The current floor count is output on the "FLRCODE" port as a 3-bit vector.

Code:

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

entity FLOORCOUNTER is
    port (UP, DOWN, Sensor: in std_logic;
          FLRCODE: out std_logic_vector(2 downto 0));
end entity FLOORCOUNTER;

architecture LogicOperation of FLOORCOUNTER is
    signal FloorCnt: integer range 0 to 4 := 0;

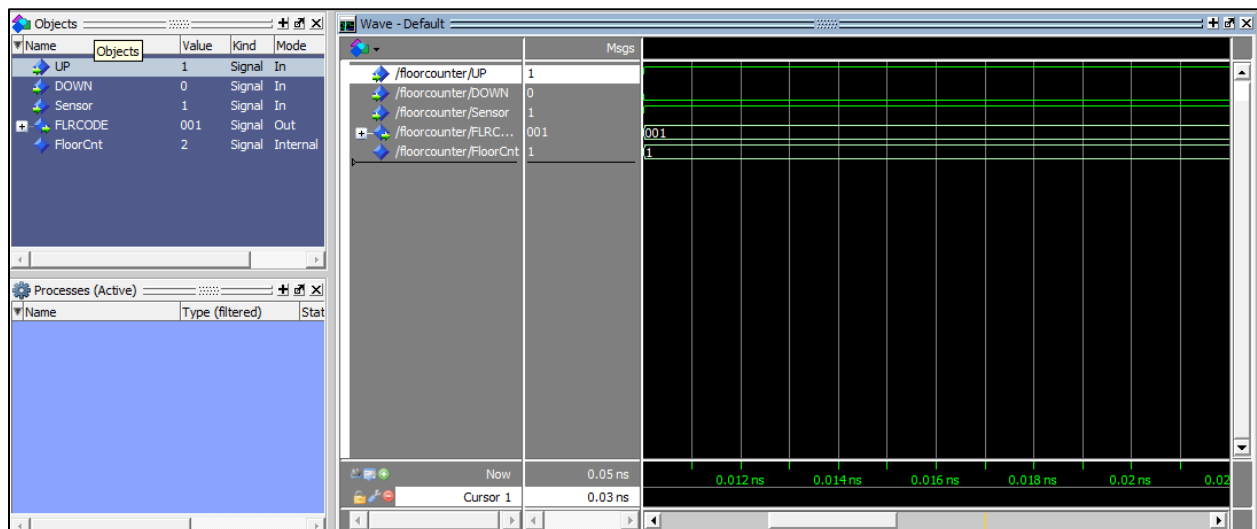
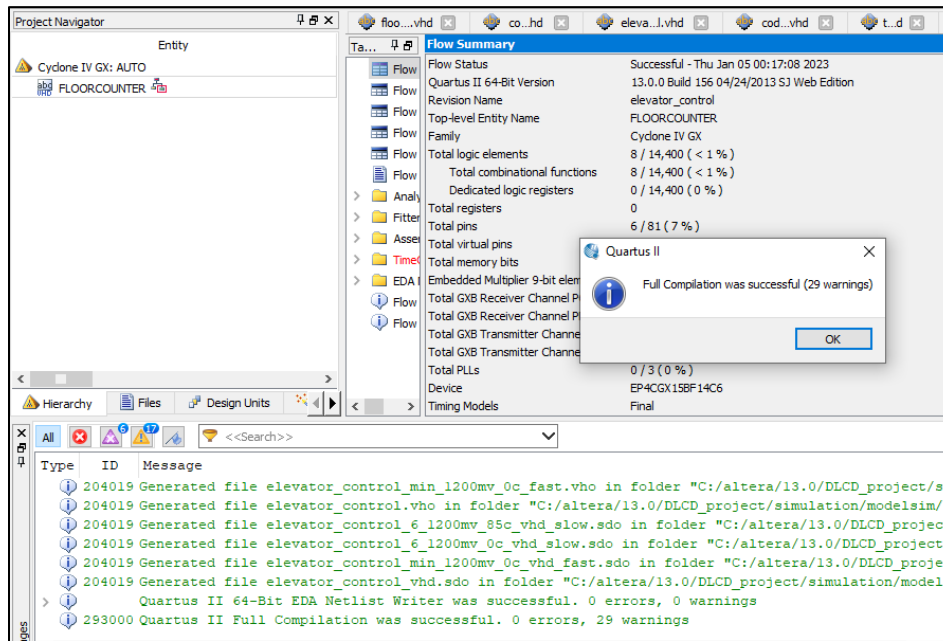
begin
    process(UP, DOWN, Sensor)
    begin
        FLRCODE <= std_logic_vector(to_unsigned(FloorCnt, 3));
        if (Sensor = '1') then
            if UP = '1' and DOWN = '0' then
                FloorCnt <= FloorCnt + 1;
            elsif UP = '0' and DOWN = '1' then
                FloorCnt <= FloorCnt - 1;
            end if;
        end if;
```

```

end if;
if FloorCnt > 4 then
    FloorCnt <= 4;
elsif FloorCnt < 0 then
    FloorCnt <= 0;
end if;
end process;

```

Simulation:



7.2 Comparator

This VHDL code defines an entity called "FLRCALLCOMPARATOR" with four input ports: "FlrCodeCall", "FlrCodeCnt", "UP", and "DOWN", and three output ports: "UP", "DOWN", and "STOP". The "FlrCodeCall" and "FlrCodeCnt" inputs are 3-bit vectors representing the requested floor and the current floor, respectively. The "UP" and "DOWN" inputs are used to control the direction of the elevator, and the "STOP" output is used to indicate when the elevator should stop at the requested floor.

The code then defines an architecture for the "FLRCALLCOMPARATOR" entity, which contains three concurrent assignments to the "UP", "DOWN", and "STOP" output ports.

The first assignment sets the "STOP" output to '1' when the requested floor ("FlrCodeCall") is equal to the current floor ("FlrCodeCnt"). Otherwise, it sets the "STOP" output to '0'.

The second assignment sets the "UP" output to '1' when the requested floor is greater than the current floor, and the current floor is not equal to "111" (the maximum floor). Otherwise, it sets the "UP" output to '0'.

The third assignment sets the "DOWN" output to '1' when the requested floor is less than the current floor, and the current floor is not equal to "000" (the minimum floor). Otherwise, it sets the "DOWN" output to '0'.

Overall, this code compares the requested floor and the current floor, and sets the "UP", "DOWN", and "STOP" outputs appropriately to move the elevator to the requested floor.

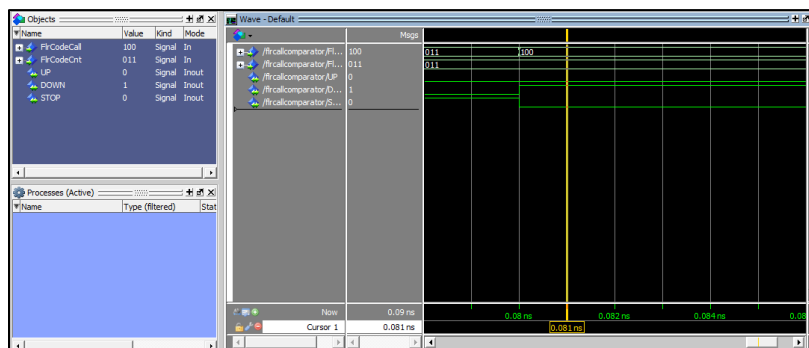
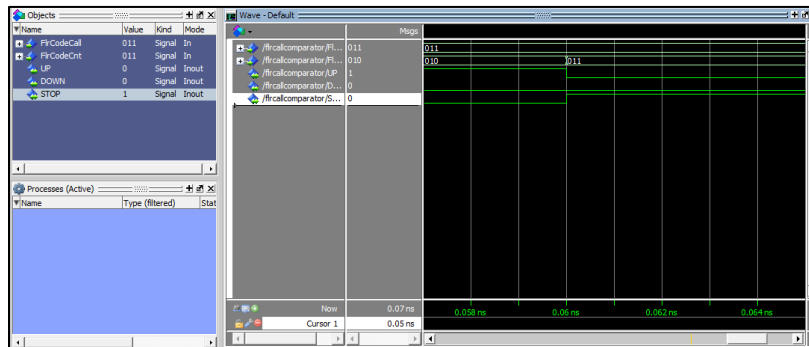
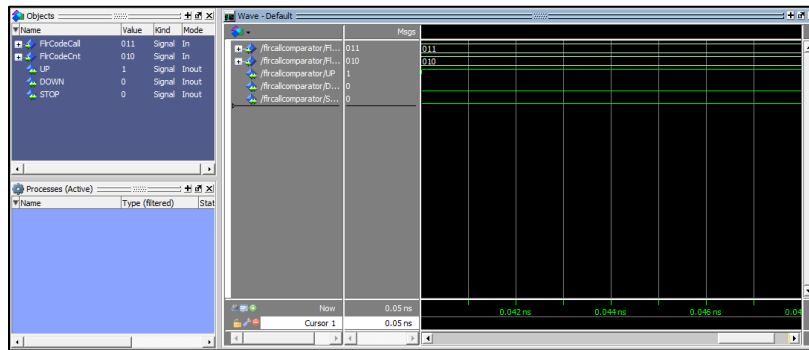
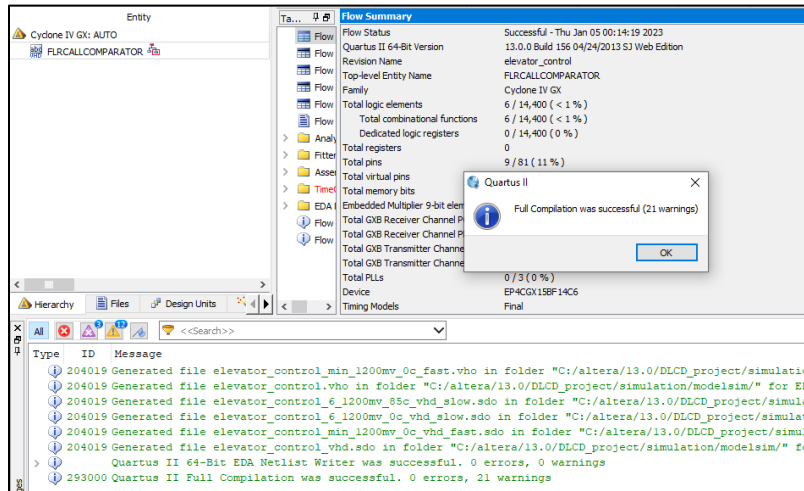
Code:

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

entity FLRCALLCOMPARATOR is
    port (
        FlrCodeCall: in std_logic_vector(2 downto 0);
        FlrCodeCnt: in std_logic_vector(2 downto 0);
        UP: inout std_logic;
        DOWN: inout std_logic;
        STOP: inout std_logic
    );
end entity FLRCALLCOMPARATOR;

architecture LogicOperation of FLRCALLCOMPARATOR is
begin
    STOP <= '1' when (FlrCodeCall = FlrCodeCnt) else '0';
    UP <= '1' when (FlrCodeCall > FlrCodeCnt and FlrCodeCnt /= "111") else
'0';
    DOWN <= '1' when (FlrCodeCall < FlrCodeCnt and FlrCodeCnt /= "000") else
'0';
end architecture LogicOperation;
```

Simulation:



7.3 Code Register

This VHDL code defines an entity called "CODEREGISTER" with two input ports: "Clk" and "FlrCodeIn", and one output port "FlrCodeOut". The "Clk" input is used as a clock signal to control the updating of the "FlrCodeOut" output, and the "FlrCodeIn" input is the value that should be registered in the output.

The code also defines an internal signal called "FlrCodeOut_prev", which is used to store the previous value of the "FlrCodeOut" output.

The code then defines a process that is sensitive to changes on the "Clk" input. Inside the process, the "FlrCodeOut" output is updated to the value of "FlrCodeIn" when a rising edge occurs on the "Clk" input.

Outside the process, the "FlrCodeOut_prev" signal is continuously updated to the value of "FlrCodeOut".

Overall, this code implements a register that stores the value of the "FlrCodeIn" input and updates the "FlrCodeOut" output on the rising edge of the "Clk" input. The "FlrCodeOut_prev" signal is used to store the previous value of the output.

Code:

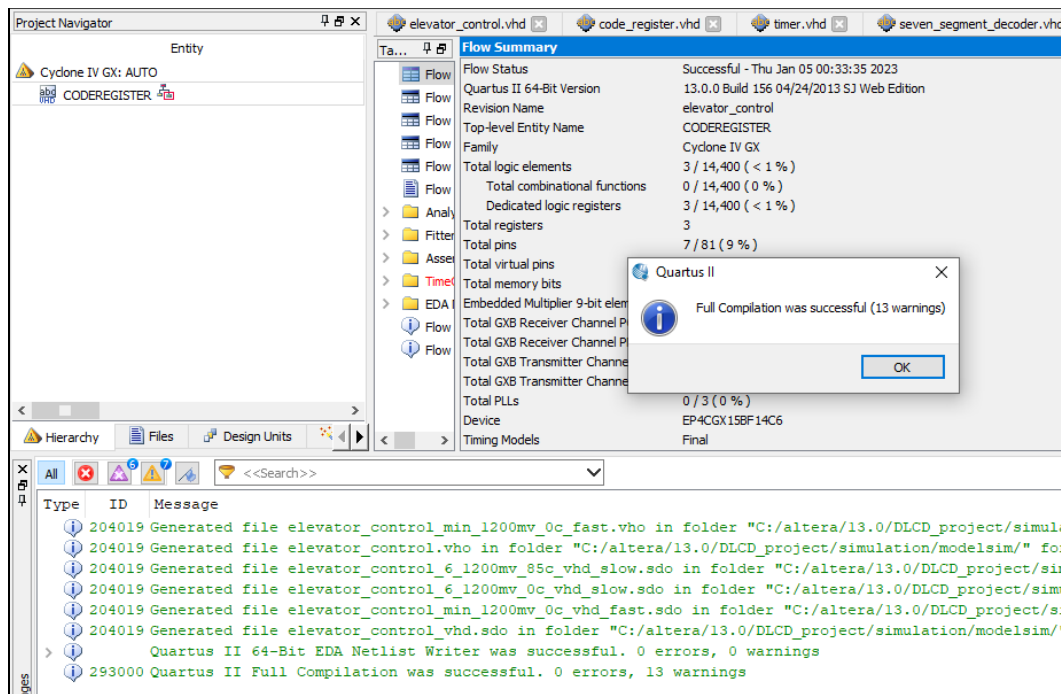
```
library ieee;
use ieee.std_logic_1164.all;

entity CODEREGISTER is
    port (
        Clk: in std_logic;
        FlrCodeIn: in std_logic_vector(0 to 2);
        FlrCodeOut: buffer std_logic_vector(0 to 2)
    );
end entity CODEREGISTER;

architecture LogicOperation of CODEREGISTER is
    signal FlrCodeOut_prev: std_logic_vector(0 to 2);
begin
    process(Clk)
    begin
        if (Clk'event and Clk = '1') then
            FlrCodeOut <= FlrCodeIn;
        end if;
    end process;

    FlrCodeOut_prev <= FlrCodeOut;
end architecture LogicOperation;
```

Simulation:

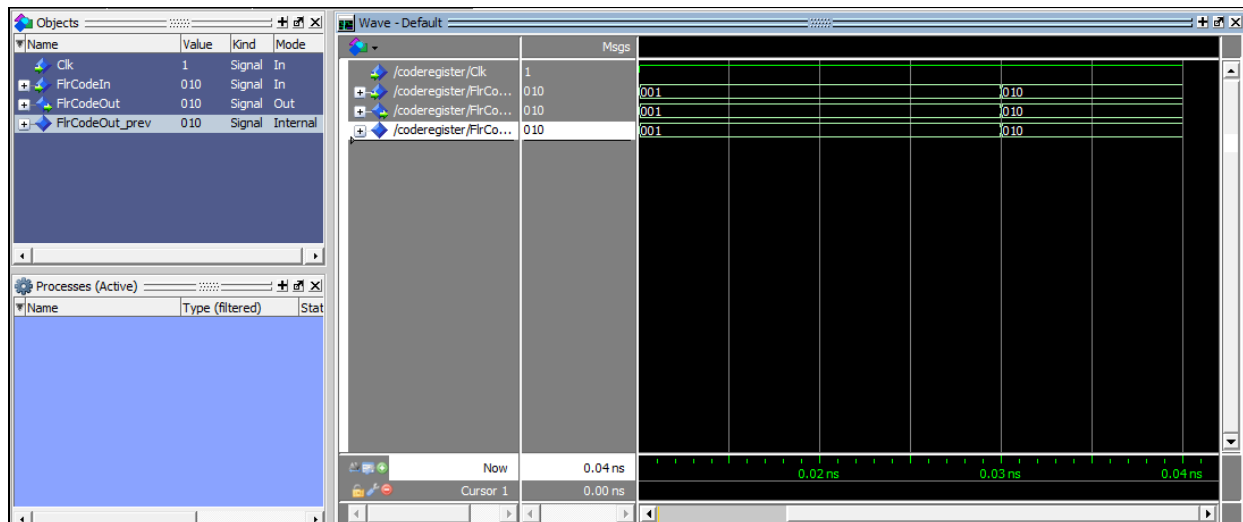


The image shows the Quartus II compilation summary and message window. The 'Flow Summary' tab is active, displaying the following information:

Flow Status	Successful - Thu Jan 05 00:33:35 2023
Quartus II 64-Bit Version	13.0.0 Build 156 04/24/2013 S3 Web Edition
Revision Name	elevator_control
Top-level Entity Name	CODEREGISTER
Family	Cyclone IV GX
Total logic elements	3 / 14,400 (< 1 %)
Total combinational functions	0 / 14,400 (0 %)
Dedicated logic registers	3 / 14,400 (< 1 %)
Total registers	3
Total pins	7 / 81 (9 %)
Total virtual pins	
Total memory bits	
Embedded Multiplier 9-bit elements	0 / 3 (0 %)
Total GXB Receiver Channel P	
Total GXB Receiver Channel P	
Total GXB Transmitter Channel	
Total GXB Transmitter Channel	
Total PLLs	0 / 3 (0 %)
Device	EP4CGX15BF14C6
Timing Models	Final

A message window titled 'Quartus II' is overlaid on the summary, stating: 'Full Compilation was successful (13 warnings)'. The 'Messages' window at the bottom shows the following messages:

```
204019 Generated file elevator_control_min_1200mv_0c_fast.vho in folder "C:/altera/13.0/DLCD_project/simul
204019 Generated file elevator_control.vho in folder "C:/altera/13.0/DLCD_project/simulation/modelsim/" fo
204019 Generated file elevator_control_6_1200mv_85c_vhd_slow.sdo in folder "C:/altera/13.0/DLCD_project/si
204019 Generated file elevator_control_6_1200mv_0c_vhd_slow.sdo in folder "C:/altera/13.0/DLCD_project/sim
204019 Generated file elevator_control_min_1200mv_0c_vhd_fast.sdo in folder "C:/altera/13.0/DLCD_project/s
204019 Generated file elevator_control_vhd.sdo in folder "C:/altera/13.0/DLCD_project/simulation/modelsim/
> Quartus II 64-Bit EDA Netlist Writer was successful. 0 errors, 0 warnings
293000 Quartus II Full Compilation was successful. 0 errors, 13 warnings
```



The image shows the Quartus II simulation waveforms. The 'Objects' window on the left lists the signals: Clk, FirCodeIn, FirCodeOut, and FirCodeOut_prev. The 'Wave - Default' window displays the waveforms for these signals. The 'Messages' window shows the following messages:

```
/coderegister/Clk 1
/coderegister/FirCo... 010
/coderegister/FirCo... 010
/coderegister/FirCo... 010
```

The waveforms show the signals over time, with a time scale of 0.04 ns. The 'Processes (Active)' window is also visible, showing the active processes during the simulation.

7.4 Timer

This VHDL code defines an entity called "Timer" with two input ports: "Enable" and "Clk", and one output port "QOut". The "Enable" input is used to start and stop the timer, and the "Clk" input is used as a clock signal to control the timing of the timer. The "SetCount" input is an integer value that determines how long the timer will run for before outputting a pulse on the "QOut" output.

The code defines an architecture for the "Timer" entity, which contains a process that is sensitive to changes on the "Enable" and "Clk" inputs. Inside the process, the "QOut" output and a variable called "Cnt" are both initialized to '0' when the "Enable" input is '0'.

When a rising edge occurs on the "Clk" input, the process checks if "SetCount" is equal to "Cnt". If it is, the "QOut" output is set to '1' and "Cnt" is reset to 0. If "SetCount" is not equal to "Cnt", "Cnt" is incremented by 1.

Overall, this code implements a timer that generates a pulse on the "QOut" output after a certain number of clock cycles, as determined by the "SetCount" input. The timer can be started and stopped using the "Enable" input.

Code:

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

entity Timer is
    port (
        Enable: in std_logic;
        Clk: in std_logic;
        SetCount: in integer range 0 to 1023;
        QOut: out std_logic
    );
end entity Timer;

architecture TimerBehavior of Timer is
begin
    process(Enable, Clk)
        variable Cnt: integer range 0 to 1023;
    begin
        if (Clk'EVENT and Clk = '1') then
            if Enable = '0' then
                Cnt := 0; QOut <= '0';
            end if;
            if SetCount = Cnt then
                QOut <= '1';
                Cnt := 0;
            else
                Cnt := Cnt + 1;
            end if;
        end if;
    end process;
end architecture TimerBehavior;
```

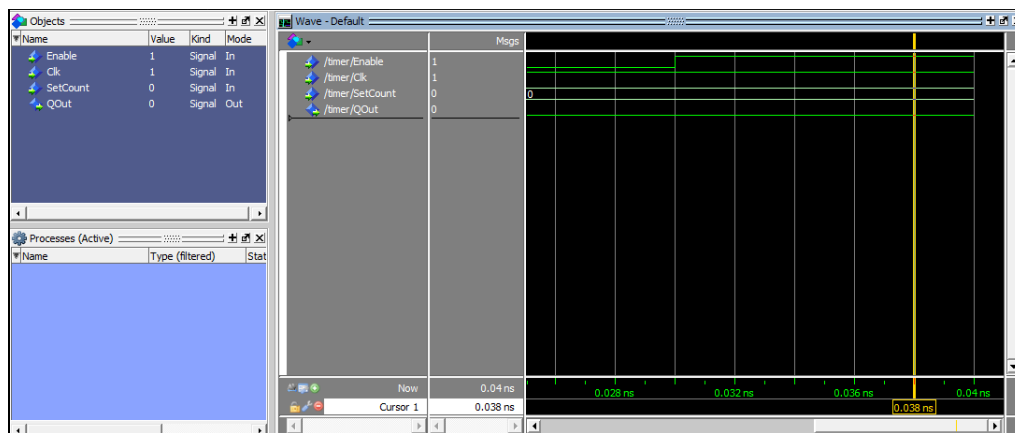
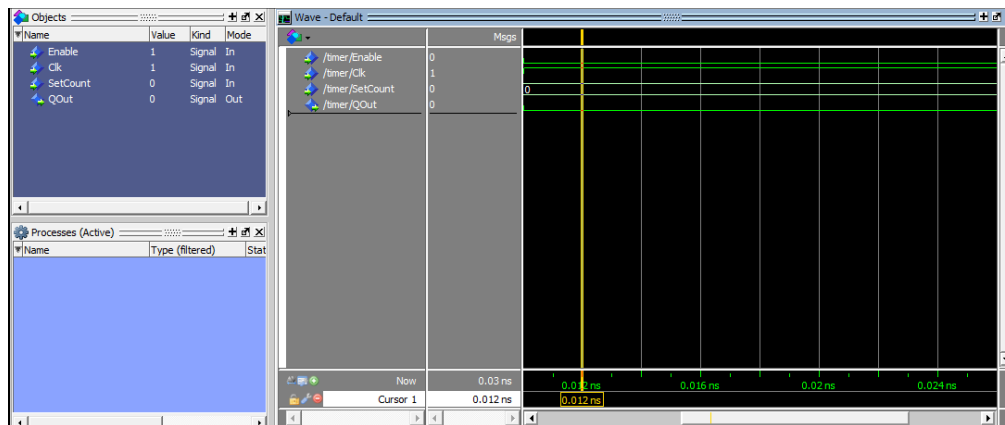
Simulation:

The screenshot shows the Quartus II simulation results window. The 'Flow Summary' tab is active, displaying the following information:

- Flow Status: Successful - Thu Jan 05 00:43:09 2023
- Quartus II 64-Bit Version: 13.0.0 Build 156 04/24/2013 SJ Web Edition
- Revision Name: elevator_control
- Top-level Entity Name: Timer
- Family: Cyclone IV GX
- Total logic elements: 22 / 14,400 (< 1 %)
- Total combinational functions: 22 / 14,400 (< 1 %)
- Dedicated logic registers: 11 / 14,400 (< 1 %)
- Total registers: 11
- Total pins: 13 / 81 (16 %)
- Total virtual pins: 0 / 3 (0 %)
- Total memory bits: 0 / 3 (0 %)
- Embedded Multiplier 9-bit elements: 0 / 3 (0 %)
- Total GXB Receiver Channel P: 0 / 3 (0 %)
- Total GXB Receiver Channel P: 0 / 3 (0 %)
- Total GXB Transmitter Channel: 0 / 3 (0 %)
- Total PLLs: 0 / 3 (0 %)
- Device: EP4CGX15BF14C6
- Timing Models: Final

A message box titled 'Quartus II' is overlaid on the summary, stating: 'Full Compilation was successful (12 warnings)'. The 'Messages' window at the bottom shows the following log:

```
204019 Generated file elevator_control_min_1200mv_0c_fast.vho in folder "C:/altera/13.0/DLCD_project/simulation/
204019 Generated file elevator_control.vho in folder "C:/altera/13.0/DLCD_project/simulation/modelsim/" for EDA
204019 Generated file elevator_control_6_1200mv_85c_vhd_slow.sdo in folder "C:/altera/13.0/DLCD_project/simulation/
204019 Generated file elevator_control_6_1200mv_0c_vhd_slow.sdo in folder "C:/altera/13.0/DLCD_project/simulation/
204019 Generated file elevator_control_min_1200mv_0c_vhd_fast.sdo in folder "C:/altera/13.0/DLCD_project/simulation/
204019 Generated file elevator_control_vhd.sdo in folder "C:/altera/13.0/DLCD_project/simulation/modelsim/" for
Quartus II 64-Bit EDA Netlist Writer was successful. 0 errors, 0 warnings
293000 Quartus II Full Compilation was successful. 0 errors, 12 warnings
```



7.5 Seven – Segment

This VHDL code defines an entity called "SevenSegment" with three input ports: "H0", "H1", and "H2", and seven output ports: "a", "b", "c", "d", "e", "f", and "g". These output ports correspond to the segments of a 7-segment display, and the input ports correspond to the inputs of a 7-segment decoder.

The code defines an architecture for the "SevenSegment" entity, which contains seven concurrent assignments to the output ports. Each of these assignments defines the value of an output segment based on the values of the input ports "H0", "H1", and "H2".

Overall, this code implements a 7-segment display using a 7-segment decoder. The input ports "H0", "H1", and "H2" control which digits are displayed on the 7-segment display, and the output ports correspond to the segments of the display.

Code:

```
library ieee;
use ieee.std_logic_1164.all;
entity SevenSegment is
    port (a, b, c, d, e, f, g: out std_logic; H0, H1, H2: inout
std_logic);
end entity SevenSegment;

architecture SevenSegmentBehavior of SevenSegment is
begin
    a <= H1 or (H2 and H0) or (not H2 and not H0);
    b <= not H2 or (not H0 and not H1) or (H0 and H1);
    c <= H0 or not H1 or H2;
    d <= (not H0 and not H2) or (not H2 and H1) or
(H1 and not H0) or (H2 and not H1
and H0);
    e <= (not H0 and not H2) or (H1 and not H0);
    f <= (not H1 and H2) or (not H1 and not H0) or (H2 and not H0);
    g <= (not H2 and H1) or (H1 and not H0) or (H2 and not H1);

end architecture SevenSegmentBehavior;
```

Simulation:

The screenshot shows the Quartus II simulation results. The **Flow Summary** window displays the following information:

Category	Value	Percentage
Flow Status	Successful	-
Quartus II 64-Bit Version	13.0.0 Build 156 04/24/2013 SJ Web Edition	-
Revision Name	elevator_control	-
Top-level Entity Name	SevenSegment	-
Family	Cyclone IV GX	-
Total logic elements	7 / 14,400	(< 1 %)
Total combinational functions	7 / 14,400	(< 1 %)
Dedicated logic registers	0 / 14,400	(0 %)
Total registers	0	-
Total pins	10 / 81	(12 %)
Total virtual pins	-	-
Total memory bits	-	-
Embedded Multiplier 9-bit elements	-	-
Total GXB Receiver Channel P	-	-
Total GXB Receiver Channel P	-	-
Total GXB Transmitter Channel	-	-
Total GXB Transmitter Channel	-	-
Total PLLs	0 / 3	(0 %)
Device	EP4CGX15BF14C6	-
Timing Models	Final	-

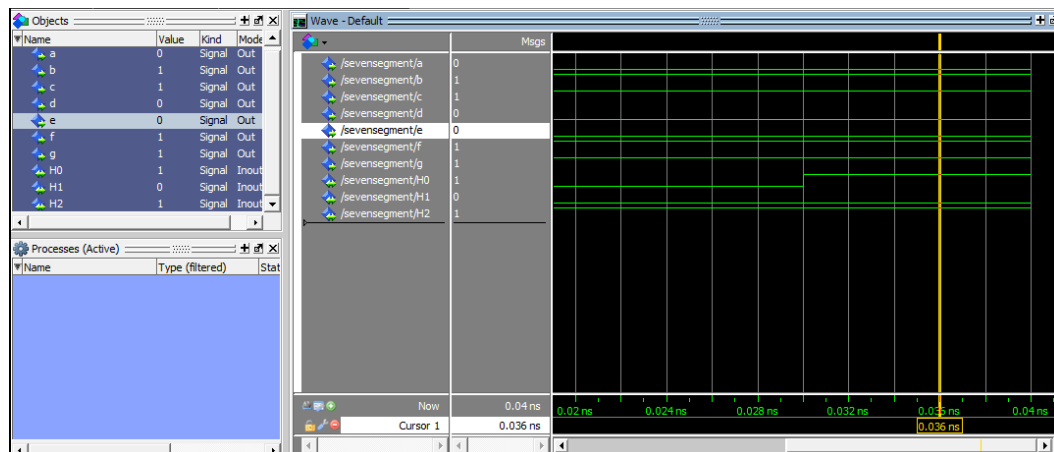
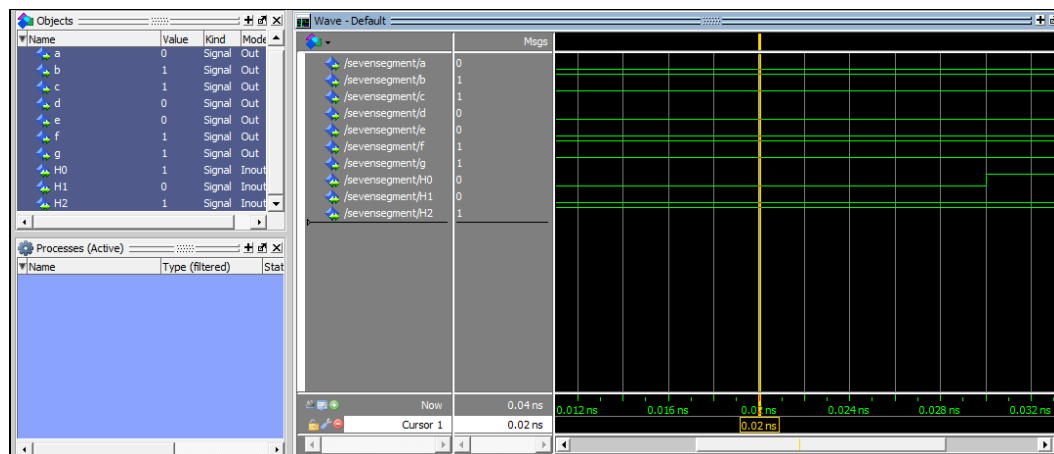
A message window titled "Quartus II" displays the following message:

Full Compilation was successful (17 warnings)

OK

The bottom pane shows the message log with the following entries:

- 204019 Generated file elevator_control_min_1200mv_0c_fast.vho in folder "C:/altera/13.0/DLCD_project/simulation/mode
- 204019 Generated file elevator_control.vho in folder "C:/altera/13.0/DLCD_project/simulation/mode
- 204019 Generated file elevator_control_6_1200mv_85c_vhd_slow.sdo in folder "C:/altera/13.0/DLCD_p
- 204019 Generated file elevator_control_6_1200mv_0c_vhd_slow.sdo in folder "C:/altera/13.0/DLCD_p
- 204019 Generated file elevator_control_min_1200mv_0c_vhd_fast.sdo in folder "C:/altera/13.0/DLCD
- 204019 Generated file elevator_control_vhd.sdo in folder "C:/altera/13.0/DLCD_project/simulation/
- Quartus II 64-Bit EDA Netlist Writer was successful. 0 errors, 0 warnings
- 204019 Quartus II Full Compilation was successful. 0 errors, 17 warnings



7.6 Call\Request FF

This VHDL code defines a digital circuit called a JK flip-flop, which has two inputs (J and K) and two outputs (Q and Q'). It operates on a clock signal, "Clk", and updates its outputs according to a set of rules based on the values of the inputs when a rising edge occurs on the clock signal. The circuit uses internal signals "QTemp" and "QNot" to temporarily store and update the values of the outputs. The JK flip-flop can be used in a variety of applications, including data storage, signal synchronization, and state machines. In the context of an elevator system, it could be used to store the current state of the elevator (e.g. which floor it is on) and to control the movement of the elevator according to input signals (e.g. button presses).

Code:

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

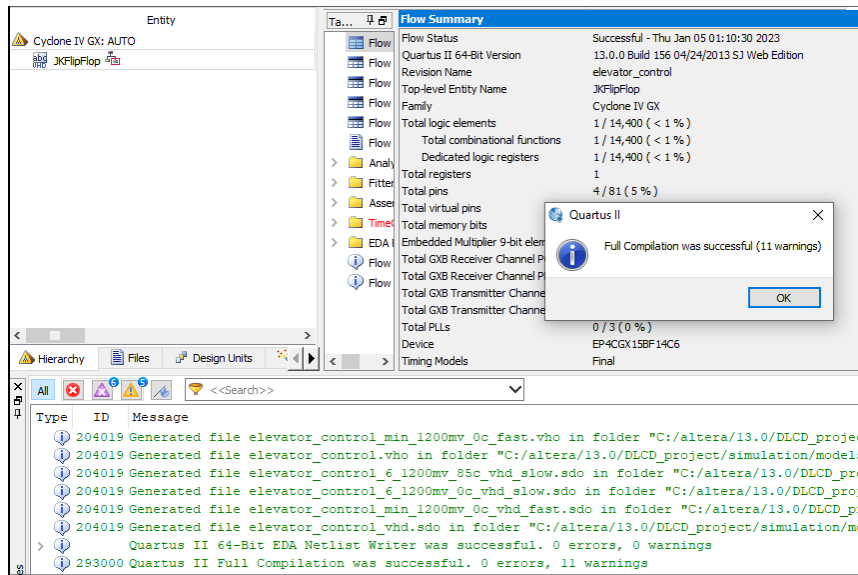
entity JKFlipFlop is
    port (
        J: in std_logic;
        K: in std_logic;
        Clk: in std_logic;
        Q: buffer std_logic
    );
end entity JKFlipFlop;

architecture LogicOperation of JKFlipFlop is
    signal QNot: std_logic := '1';
    signal QTemp: std_logic;

begin
    process (J, K, Clk)
    begin
        if (Clk'EVENT and Clk = '1') then
            if J = '1' and K = '0' then
                QTemp <= '1';
            elsif J = '0' and K = '1' then
                QTemp <= '0';
            elsif J = '1' and K = '1' then
                QTemp <= QNot;
            end if;
        end if;
    end process;

    Q <= QTemp;
    QNot <= not Q;
end architecture LogicOperation;
```

Simulation:

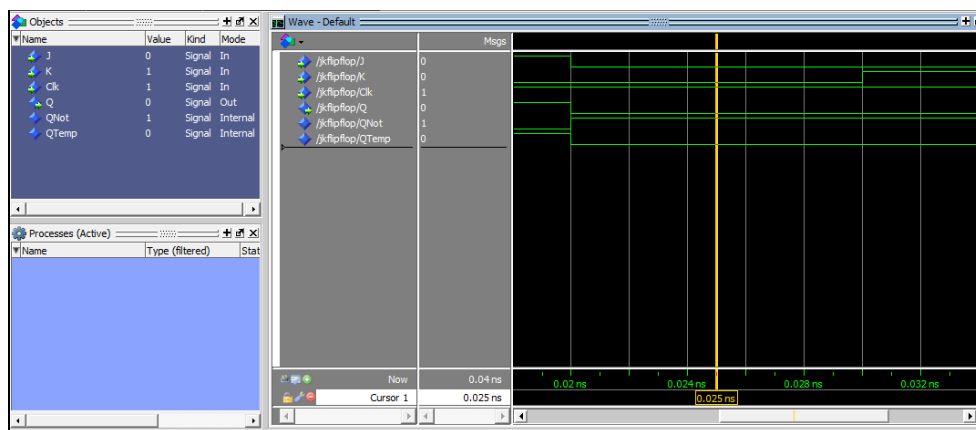
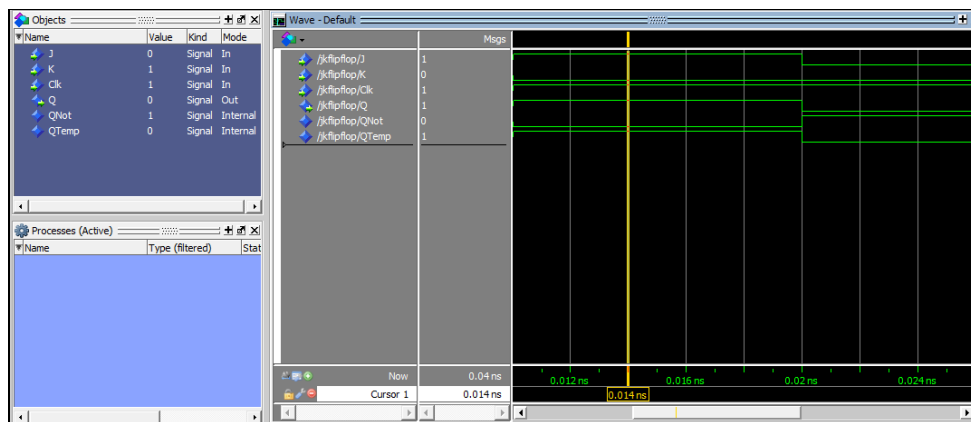


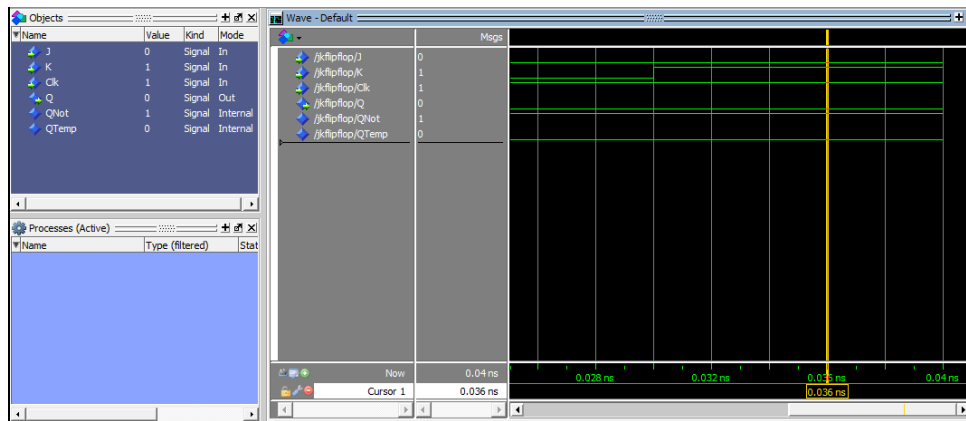
The image shows the Quartus II Flow Summary window and a message dialog box. The Flow Summary window displays the following information:

Category	Value
Flow Status	Successful - Thu Jan 05 01:10:30 2023
Quartus II 64-Bit Version	13.0.0 Build 156 04/24/2013 SJ Web Edition
Revision Name	elevator_control
Top-level Entity Name	JKFlipFlop
Family	Cyclone IV GX
Total logic elements	1 / 14,400 (< 1 %)
Total combinational functions	1 / 14,400 (< 1 %)
Dedicated logic registers	1 / 14,400 (< 1 %)
Total registers	1
Total pins	4 / 81 (5 %)
Total virtual pins	
Total memory bits	
Embedded Multiplier 9-bit elements	
Total GXB Receiver Channel Pairs	
Total GXB Receiver Channel Pairs	
Total GXB Transmitter Channel Pairs	
Total PLLs	0 / 3 (0 %)
Device	EP4CGX15BF14C6
Timing Models	Final

The message dialog box displays the following information:

Quartus II
Full Compilation was successful (11 warnings)
OK





7.7 Elevator Control

This VHDL code describes a digital circuit for controlling an elevator system. It has several inputs and outputs for communicating with the various components of the elevator, and it has several internal components for performing specific tasks such as storing and comparing data, generating timer events, and displaying information. The circuit is triggered by changes to the "Emergency" input and determines the destination floor code and the direction the elevator should move based on the status of the call and panel buttons. The current floor is updated based on the direction of movement and the output of the floor sensors, and it is displayed on the seven-segment display.

Code:

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

entity ELEVATOR is
    port (CallCode, PanelCode: in std_logic_vector(1 downto 0);
          Call, Request, Sensor, Clk, Emergency: in std_logic;
          UP, DOWN, STOPOPEN, CLOSE: inout std_logic;
          a, b, c, d, e, f, g: out std_logic);
end entity ELEVATOR;

architecture LogicOperation of ELEVATOR is
    component FLOORCOUNTER is
        port (UP, DOWN, Sensor: in std_logic;
              FLRCODE: out std_logic_vector(1 downto 0));
    end component FLOORCOUNTER;

    component FLRCALLCOMPARATOR is
        port (FlrCodeCall, FlrCodeCnt: in std_logic_vector(1 downto 0);
              UP, DOWN, STOP : inout std_logic);
    end component FLRCALLCOMPARATOR;

    component CODEREGISTER is
        port (Clk: in std_logic;
              FlrCodeIn: in std_logic_vector(0 to 1);
              FlrCodeOut: out std_logic_vector(0 to 1));
    end component CODEREGISTER;
```

```

end component CODEREGISTER;

component Timer is
  port (Enable, Clk: in std_logic;
        SetCount: in integer range 0 to 1023;
        QOut: inout std_logic);
end component Timer;

component SevenSegment is
  Port (a, b, c, d, e, f, g: out std_logic;
        H0, H1: inout std_logic);
end component SevenSegment;

component JKFlipFlop is
  port (J, K, Clk: in std_logic;
        Q: out std_logic);
end component JKFlipFlop;

signal FRCNT, FRCLOUT, FRCL, FRIN: std_logic_vector(0 to 1);
signal CallEn: std_logic;
signal EmergencyActive: std_logic;

begin

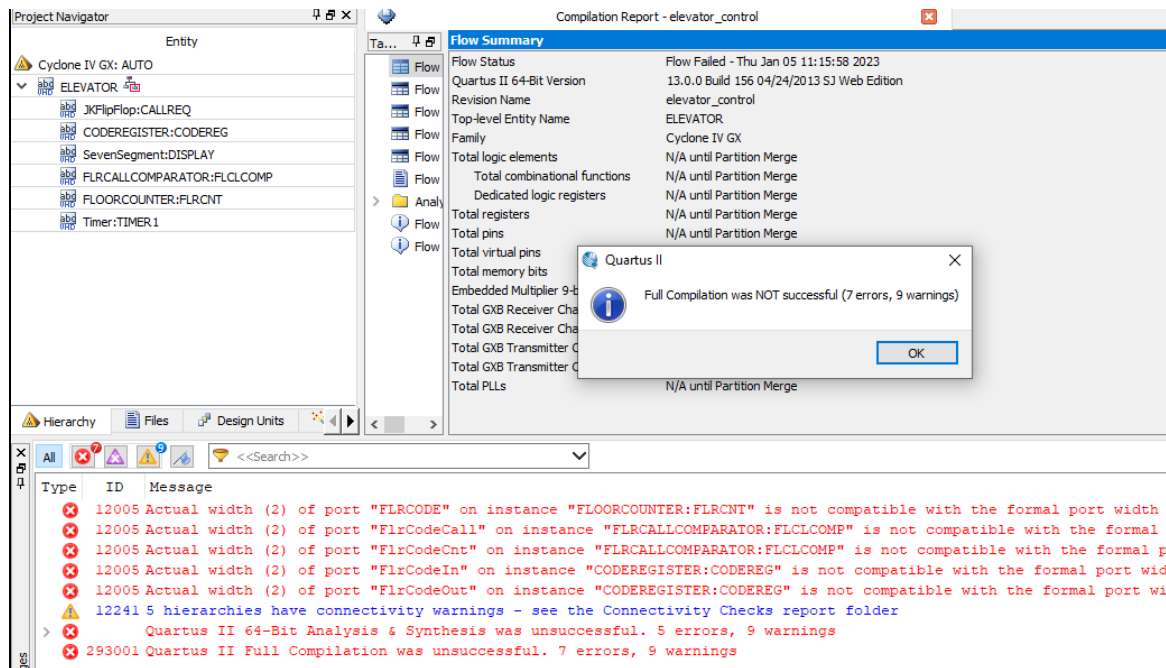
  process (Emergency)
  begin
    if Emergency = '1' then
      EmergencyActive <= '1';
    end if;
  end process;

  process (CallEn, EmergencyActive, CallCode, PanelCode)
  begin
    if EmergencyActive = '1' then
      FRIN <= "00";
    elsif (CallEn = '1') then
      FRIN <= CallCode;
    else
      FRIN <= PanelCode;
    end if;
  end process;

  CALLREQ: JKFlipFlop port map(J => '1', K => '1', Clk => CLOSE, Q =>
CallEn);
  CODEREG: CODEREGISTER port map(Clk => Clk, FlrCodeIn => FRIN, FlrCodeOut =>
FRCL);
  FLCLCOMP: FLRCALLCOMPARATOR port map(FlrCodeCall => FRCL, FlrCodeCnt =>
FRCNT, Up => UP, Down => DOWN, Stop => STOPOPEN);
  FLRCNT: FLOORCOUNTER port map(UP => UP, DOWN => DOWN, Sensor => Sensor,
FLRCODE => FRCNT);
  DISPLAY: SevenSegment port map(a=>a,b=>b,c=>c,d=>d,e=>e,f=>f,g=>g,
                                H0=>FRCNT(1),H1=>FRCNT(0));
  TIMER1: Timer port map (Enable=>STOPOPEN, Clk=> Clk,SetCount=>10,
                          QOut=>Close);

end architecture LogicOperation;

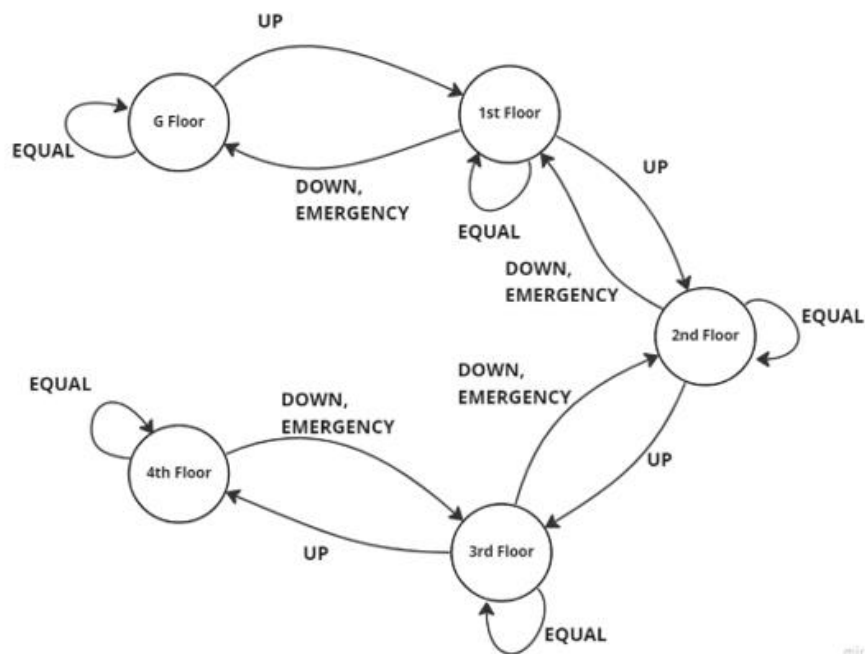
```



8. State Diagram With Emergency Button

The design can be further improved by adding an emergency button which will bring the elevator to the ground floor no matter which floor the elevator resided on at the moment.

The state diagram for the design which includes an emergency button:



9. Conclusion

A digital logic-based elevator control system for a four storied building has been proposed in this report. Considering the events that occur during the movement of an elevator, two state diagrams were designed. These state diagrams were used to come up with the functions that are needed to be performed to control the elevator. These functions were translated into logic equations and were implemented using basic logic gates and available digital logic ICs into sub circuits. Combining these sub circuits, the full control circuit had been created.

10. References

- T. L. Floyd, *Digital Fundamentals*. Harlow: Pearson Education, 2015.
- “A simulation study of an elevator control system using digital logic.” [Online]. Available:
https://www.researchgate.net/publication/322052244_A_Simulation_Study_of_an_Elevator_Control_System_using_Digital_Logic. [Accessed: 09-Jan-2023].
- *IOSR Journal of Engineering*, vol. 3, no. 12, 2013.
- “Sequential design example: Elevator,” *YouTube*, 01-Oct-2020. [Online]. Available:
<https://www.youtube.com/watch?v=DwKKly9M1cE>. [Accessed: 09-Jan-2023].
- “Finite State Machine example 2,” *YouTube*, 02-Sep-2020. [Online]. Available:
https://www.youtube.com/watch?v=_F_RrPRQOgU. [Accessed: 09-Jan-2023].