IIT Bombay

EE224 + PH230 Project

Optimal Elevator Control System

Student Name	Student ID
1. Nahush Rajesh Kolhe	210260034
2. Spandan Sachin Anaokar	210260055

Lecturer in charge:

Prof. Pradeep Sarin

Date : 9/04/23

Contents

1 Project Summary

We have built a working model of the Optimal Elevator Control System. This system consists of an Elevator with four states i.e. Ground floor, 1^{st} floor, 2^{nd} floor, and 3^{rd} floor. We have 4 call buttons which are present outside each floor, and 4 destination buttons which are all present inside the Elevator, along with a fan installed. There is a Close button inside the elevator which forcefully closes the elevator. All the 8 call and destination buttons (together called request buttons), close button, fan, and the states of the elevator are represented by LEDs.

Features of the Elevator Control System :

- When a request button is pressed it will stay active (the request is still pending) until the Elevator reaches the corresponding floor.
- If the Elevator has to change states to satisfy the request, it will either move 1 floor above or below in 2 seconds (clock cycle), depending on the request.
- At any moment of time, when any set of request buttons are active the Elevator takes the next step in a path which requires least amount of time to satisfy all the request buttons. i.e. It takes the optimal path, minimizing "time" parameter.
- Whenever elevator satisfies a request or when someone presses and releases the button representing i^{th} floor at a time when elevator is itself at i^{th} floor, then the elevator will open to let the user in and close automatically after 4 sec. The person may press and release that button continuously to hold the elevator for longer period of time.
- Whenever the elevator is open, person inside the elevator can press the Close button to instantly close the elevator.
- If one or more destination buttons are active, then the fan will switch ON, else will remain OFF. This not only automates the fan but also acts as an indicator to the person inside the Elevator that a destination button has been recorded or not.

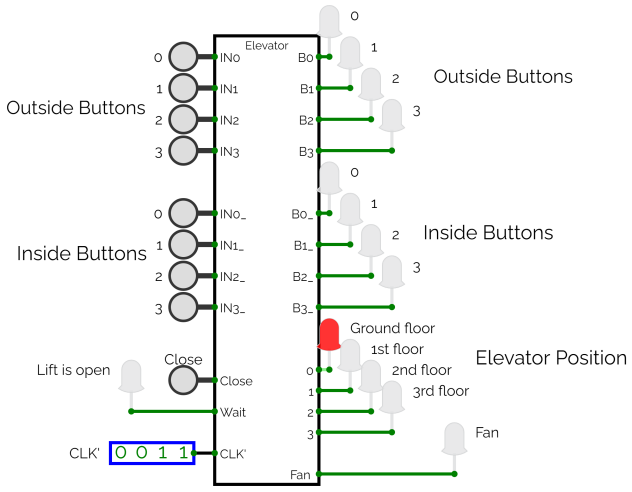


Figure 1: Final elevator Model

2 Introduction

2.1 Elevator Control Systems in General

An Elevator Control System is an integral component of modern buildings, providing efficient transportation of people and goods between different floors. The system's reliability, safety, and performance are critical to ensuring smooth and uninterrupted operations.

However, elevator systems often suffer from inefficiencies that can result in long wait times, overcrowding, and wasted energy. To address these issues, an optimal elevator control system can be designed using digital electronics to improve elevator performance, reduce energy consumption, and enhance user experience.

An Optimal Elevator Control System is a system that efficiently and effectively manages the operation of elevators in a building, ensuring that the elevator system is used to its full potential. Such a system takes into account factors such as the number of people waiting for an elevator, their destination floors, the elevator's current position, and the time of day, among others, to optimize elevator usage and reduce wait times.

2.2 Our Work

For the sake of simplicity the word "Optimal" in our project will only cater to the time parameter, which means that the Elevator will minimize the total wait time. Wait time is defined (here) as the total time required for the Elevator to satisfy a given set of requests.

In this project, we will be designing, simulating, and implementing an Optimal Elevator Control System using digital electronics, which involves the use of logic gates, flip-flops, and other digital components. By leveraging digital electronics, we can achieve a high level of precision and accuracy in elevator control while reducing the system's complexity and wait time. The project will cover the design, implementation, testing, and performance of the system, under different operating conditions.

3 Theoretical Design

The entire circuit design is broken down into different components namely Forward Edge, Button, Delay, Main, and Elevator. Then these all are put together into Final Elevator which neatly displays the entire elevator control system.

We will look at each component one by one and provide an explaination of its working and use. We have set all the clocks(CLK) used to 125ms.

Note: Given a variable Q, Q' represents the complement of Q. For a given variable A representing a state, A^* represents the new state.

3.1 Forward Edge Detector

Figure 2 represents the Forward Edge Detector which is a simple circuit that we will use multiple times in the rest of our components. Its function is to detect the forward edge of the input

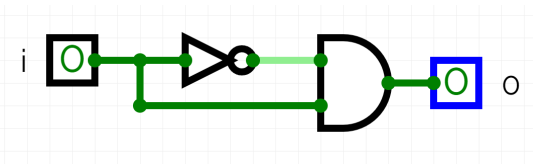


Figure 2: Circuit Diagram of Forward Edge Detector

Whenever Input(i) goes from logic 0 to logic 1, output(o) becomes logic 1 for a very short amount of time. For all other cases, the output is logic 0.
This happens due to the delay present in the NOT gate. Hence whenever input goes from logic 0 to logic 1, the output is logic 1 during the time when the NOT gate is yet to update.

3.2 Button

A Button circuit represents a request button present inside and outside of the elevator. There are 8 such buttons and all of them have identical circuits. The main function of the button is to record a user request, which can then be passed on for further processing.

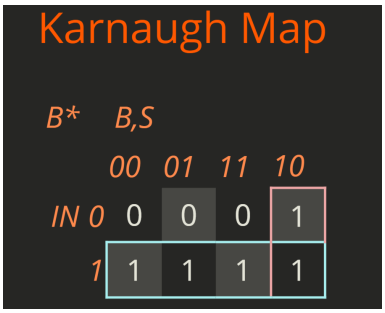
B_0, B_1, B_2, B_3 represents the buttons present outside the elevator at each corresponding floor. $B_{0\_}, B_{1\_}, B_{2\_}, B_{3\_}$ represents the buttons present inside the elevator.IN_i for i = 0,1,2,3,0_,1_,2_,3_ represents the input for the corresponding button, which is generally a short press by the user, i.e. a short square wave pulse.
S_i represents the state of the elevator. When the elevator is in state i, only then S_i will be logic 1, else it will be logic 0.

We want the Button to work in the following way:

- User has access to Input(IN) which when he presses, we want the Button(B*) to be logic 1, irrespective of B and S. This represents calling the elevator or defining the destination of the elevator.

	IN	B	S	B*
0	0	0	0	0
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

(a) Truth Table



(b) K-Map

Figure 3: Karnaugh-Map for Button B

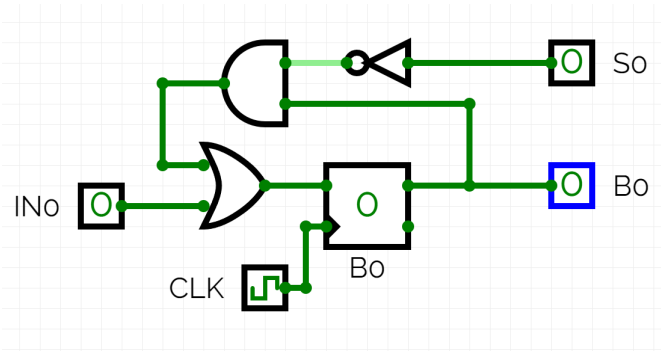


Figure 4: Circuit Diagram of Button B_0

- When the IN is logic 0, S is logic 0 and Button B is logic 1 we want the Button B* to remain in logic 1. This represents that if the elevator hasn't yet fulfilled the request then the button must remain active.
- When S is 1 in the case of IN being logic 0, we want button B* to be logic 0. this represents when the lift has completed the request by reaching a state, that particular button must go OFF.
- Lastly the trivial case when all are logic 0, we want B* also in logic 0.

This can be represented by a truth table given in Figure 3.(a), and the corresponding K-Map is given in Figure 3.(b). Finally, we get the following equation.

$$B^* = IN + S'B$$

The circuit representing Button B_0 is given in Figure 4

3.3 Delay Circuit

This circuit will be used in the Main circuit to cause a delay in the state transition. This delay time represents the time taken by the lift to open and close itself to let the user in. The Delay Circuit plays an important role in many situations which we will see in a bit. Let's first look at the mechanism of the circuit.

$$Bi\_f = B_0 + B_{0\_}$$

In the delay circuit initially, x' is logic 1. The first part of the circuit detects the Negative Edge of any B_{i_f} (i.e. Forward edge of the B'_{i_f}) and connects it

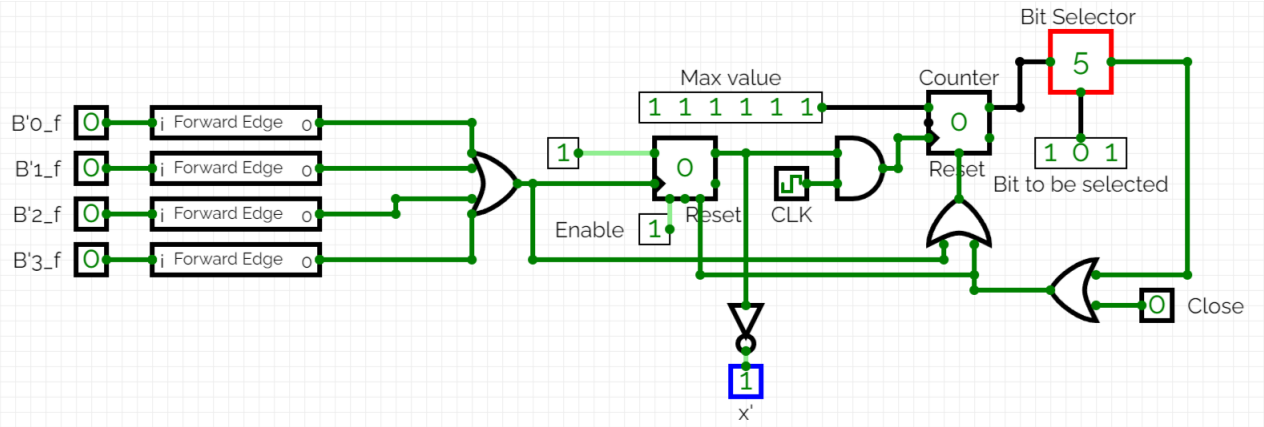


Figure 5: Delay Circuit Diagram

to the clock of a D register. When it is detected the D register then stores the constant value 1 which in turn activates the clock of the Counter. This counter then counts till 32 i.e. the 6th bit of the output of the counter becomes 1.

The Bit selector's function is to output a particular bit from a string of bits. Note that when we assign "Bit to be selected" as 5, it will pick up the 6th bit, because it counts from 0.

When the counter is counting from 0 to 32, we can see that the output x' is logic 0. As soon as the output of the Bit selector goes high the Counter and D register are made to reset and hence making the x' logic 0 again.

Hence using the Negative edge of Buttons we have created a pulse of width 32*(Clock Cycle width) = 4sec
These 4 seconds precisely denote the time required for the lift to open and close and "x" represents nothing but the status of the lift is open.

Also notice that a "Close" input has been added which can reset the Counter and D register when it is logic 1. This "Close" represents nothing but the Close button to forcefully close the lift without completing its usual 4-second time.

Features of Delay Circuit :

- Whenever any button Bi_f goes from logic 1 to logic 0, "x" becomes active for 4 seconds. This represents whenever the elevator completes a request the Button switches OFF automatically which in turn leads to x becoming active and opening the elevator for the person to come in or leave. This is exactly when we want the elevator to open.
- Consider a person on the floor i inside or outside the lift and the lift is also on the floor i. If this person presses and releases the button on that floor then this will cause the elevator to open, which is what we expect of it. Also if the elevator is in the open state and the person presses and releases the button, then the elevator will reset the time for being open, which is exactly what we want the elevator to do in situations when the person wants to hold the elevator.

- Note that the above point only works when he releases a previously pressed button because we are detecting a negative edge. We have opted for this convention due to its simple design, nevertheless, it makes no difference, it's just that we have a little different rule compared to commercial elevators.

- Whenever the elevator is open i.e. x is active. A person inside the lift has the power to close the lift instantly using the "Close" button.

3.4 Main

This is the main part of the circuit which governs the state change of the elevator. Here we will deal with the effective buttons $B_{i\_f's}$, because it doesn't matter if the button was pressed from outside or inside, the elevator just wants to know where the request is. We will also introduce a slow clock(CLK') for the state change of the elevator. Finally, we will introduce the Delay circuit and Forward Edge Detector to introduce the required features of the elevator.

To optimize the wait time, we will do case by case analysis of every possible situation and make its K-Map. Let F_0 be the 0th bit (LSB) and F_1 be 1st bit (MSB) of the state variables. Hence $F_1F_0 = 00$ represents the 0th floor, 01 represents the 1st floor, 10 represents the 2nd floor and 11 represents the 3rd floor respectively. Input variables are the 4 $B_{i\_f's}$. Let's start constructing logic for the truth table :

- When the elevator is in state i and no buttons are active then the elevator should remain in state i

- When the elevator is in state i and at least the i^{th} button is active, the elevator should remain in state i.

- When the elevator is at the ground floor (state 0) and a subset of $\{B_{1\_f}$, B_{2_f}, B_{3_f} $\}$ buttons are active, the shortest path will be to directly go towards the required floors in a straight line. Hence the new state of the elevator must be state 1.

- When the elevator is on the 1st floor and a subset of $\{B_{2\_f}$, B_{3_f} $\}$ buttons are active, the shortest path will again be the straight line, hence the new state of the elevator must be state 2.

- When the elevator is on the 1st floor and B_{0_f} is active, the new state of the elevator must be state 0.

- When the elevator is at 1st floor and $\{B_{0\_f}$, B_{3_f} $\}$ or $\{B_{0\_f}$, B_{2_f}, B_{3_f} $\}$ are active then we can see 2 obvious paths, one is to go to 3rd floor and then go to ground floor, other is first go to ground floor and then to 3rd floor. The first path requires 5 steps, whereas the later one requires only 4 steps. No other path will give less than 4 steps, hence this is the optimal path. Hence the new state will be state 0. Note that we ignore the time required to open and close the elevator in our calculations because in any path we encounter same number of such events.

- When the elevator is on the 1st floor and $\{B_{0\_f}$, $B_{2\_f}\}$ is active, again we have 2 obvious paths which both require 2 steps. However, there is a subtle

difference. In the first path where the elevator goes to the 2nd floor first, if at this time some person randomly presses a button then now the average time to fulfill the requests will be $\frac{2+2+2+4}{4} = 10/4$. These 4 cases represent the random person pressing the button

of 0th,1st,2nd and 3rd floors. Similarly for the second path where the elevator goes to the ground floor first, if at this time a random button is pressed, the average time to fulfill the requests will be $\frac{2+2+2+3}{4} = 9/4$. Hence the second path is more optimal when we take into account future requirements. So the new state will be state 0.

- Rest of the cases are symmetric due to the symmetry of the elevator.

The following Figure 6 and Figure 7 represent the truth table generated by this logic, and Figure 8 represents Karnaugh-maps for them.

	F1	F0	B0_f	B1_f	B2_f	B3_f	F0*
0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	1
2	0	0	0	0	1	0	1
3	0	0	0	0	1	1	1
4	0	0	0	1	0	0	1
5	0	0	0	1	0	1	1
6	0	0	0	1	1	0	1
7	0	0	0	1	1	1	1
8	0	0	1	0	0	0	0
9	0	0	1	0	0	1	0
10	0	0	1	0	1	0	0
11	0	0	1	0	1	1	0
12	0	0	1	1	0	0	0
13	0	0	1	1	0	1	0
14	0	0	1	1	1	0	0
15	0	0	1	1	1	1	0
16	0	1	0	0	0	0	0
17	0	1	0	0	0	1	0
18	0	1	0	0	1	0	0
19	0	1	0	0	1	1	0
20	0	1	0	1	0	0	1
21	0	1	0	1	0	1	1
22	0	1	0	1	1	0	1
23	0	1	0	1	1	1	1
24	0	1	1	0	0	0	0
25	0	1	1	0	0	1	0
26	0	1	1	0	1	0	0
27	0	1	1	0	1	1	0
28	0	1	1	1	0	0	1
29	0	1	1	1	0	1	1
30	0	1	1	1	1	0	1

(a)

31	0	1	1	1	1	1	1
32	1	0	0	0	0	0	0
33	1	0	0	0	0	1	1
34	1	0	0	0	1	0	0
35	1	0	0	0	1	1	0
36	1	0	0	1	0	0	1
37	1	0	0	1	0	1	1
38	1	0	0	1	1	0	0
39	1	0	0	1	1	1	0
40	1	0	1	0	0	0	1
41	1	0	1	0	0	1	1
42	1	0	1	0	1	0	0
43	1	0	1	0	1	1	0
44	1	0	1	1	0	0	1
45	1	0	1	1	0	1	1
46	1	0	1	1	1	0	0
47	1	0	1	1	1	1	0
48	1	1	0	0	0	0	1
49	1	1	0	0	0	1	1
50	1	1	0	0	1	0	0
51	1	1	0	0	1	1	1
52	1	1	0	1	0	0	0
53	1	1	0	1	0	1	1
54	1	1	0	1	1	0	0
55	1	1	0	1	1	1	1
56	1	1	1	0	0	0	0
57	1	1	1	0	0	1	1
58	1	1	1	0	1	0	0
59	1	1	1	0	1	1	1
60	1	1	1	1	0	0	0
61	1	1	1	1	0	1	1
62	1	1	1	1	1	0	0
63	1	1	1	1	1	1	1

(b)

Figure 6: Truth Table for F_0^*

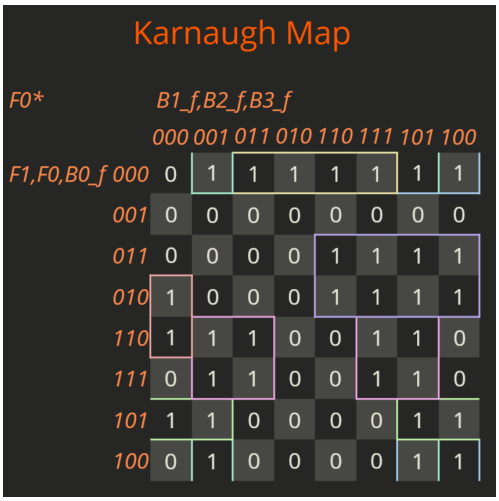
	F1	F0	B0_f	B1_f	B2_f	B3_f	F1*
0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	0
2	0	0	0	0	1	0	0
3	0	0	0	0	1	1	0
4	0	0	0	1	0	0	0
5	0	0	0	1	0	1	0
6	0	0	0	1	1	0	0
7	0	0	0	1	1	1	0
8	0	0	1	0	0	0	0
9	0	0	1	0	0	1	0
10	0	0	1	0	1	0	0
11	0	0	1	0	1	1	0
12	0	0	1	1	0	0	0
13	0	0	1	1	0	1	0
14	0	0	1	1	1	0	0
15	0	0	1	1	1	1	0
16	0	1	0	0	0	0	0
17	0	1	0	0	0	1	1
18	0	1	0	0	1	0	1
19	0	1	0	0	1	1	1
20	0	1	0	1	0	0	0
21	0	1	0	1	0	1	0
22	0	1	0	1	1	0	0
23	0	1	0	1	1	1	0
24	0	1	1	0	0	0	0
25	0	1	1	0	0	1	0
26	0	1	1	0	1	0	0
27	0	1	1	0	1	1	0
28	0	1	1	1	0	0	0
29	0	1	1	1	0	1	0
30	0	1	1	1	1	0	0

(a) Truth Table

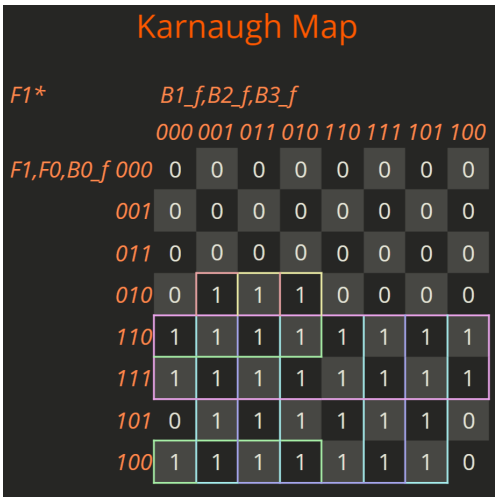
31	0	1	1	1	1	1	0
32	1	0	0	0	0	0	1
33	1	0	0	0	0	1	1
34	1	0	0	0	1	0	1
35	1	0	0	0	1	1	1
36	1	0	0	1	0	0	0
37	1	0	0	1	0	1	1
38	1	0	0	1	1	0	1
39	1	0	0	1	1	1	1
40	1	0	1	0	0	0	0
41	1	0	1	0	0	1	1
42	1	0	1	0	1	0	1
43	1	0	1	0	1	1	1
44	1	0	1	1	0	0	0
45	1	0	1	1	0	1	1
46	1	0	1	1	1	0	1
47	1	0	1	1	1	1	1
48	1	1	0	0	0	0	1
49	1	1	0	0	0	1	1
50	1	1	0	0	1	0	1
51	1	1	0	0	1	1	1
52	1	1	0	1	0	0	1
53	1	1	0	1	0	1	1
54	1	1	0	1	1	0	1
55	1	1	0	1	1	1	1
56	1	1	1	0	0	0	1
57	1	1	1	0	0	1	1
58	1	1	1	0	1	0	1
59	1	1	1	0	1	1	1
60	1	1	1	1	0	0	1
61	1	1	1	1	0	1	1
62	1	1	1	1	1	0	1
63	1	1	1	1	1	1	1

(b) K-Map

Figure 7: Truth Table for F_1^*



(a) For F_0^*



(b) For F_1^*

Figure 8: Karnaugh Map for new states

These K-maps give us the following equations :

$$F_0^* = F_1'F_0'B_{0\_f}'B_{2\_f} + F_1'F_0B_{1\_f} + F_1F_0'B_{0\_f}B_{2\_f}' + F_1F_0B_{3\_f} + F_0'B_{0\_f}'B_{2\_f}'B_{3\_f} + F_0'B_{0\_f}'B_{1\_f}B_{2\_f}' + F_0B_{0\_f}'B_{1\_f}'B_{2\_f}'B_{3\_f}' \quad (1)$$

$$F_1^* = F_0B_{0\_f}'B_{1\_f}'B_{3\_f} + F_0B_{0\_f}'B_{1\_f}'B_{2\_f} + F_1B_{0\_f}'B_{1\_f}' + F_1B_{3\_f} + F_1B_{2\_f} + F_1F_0 \quad (2)$$

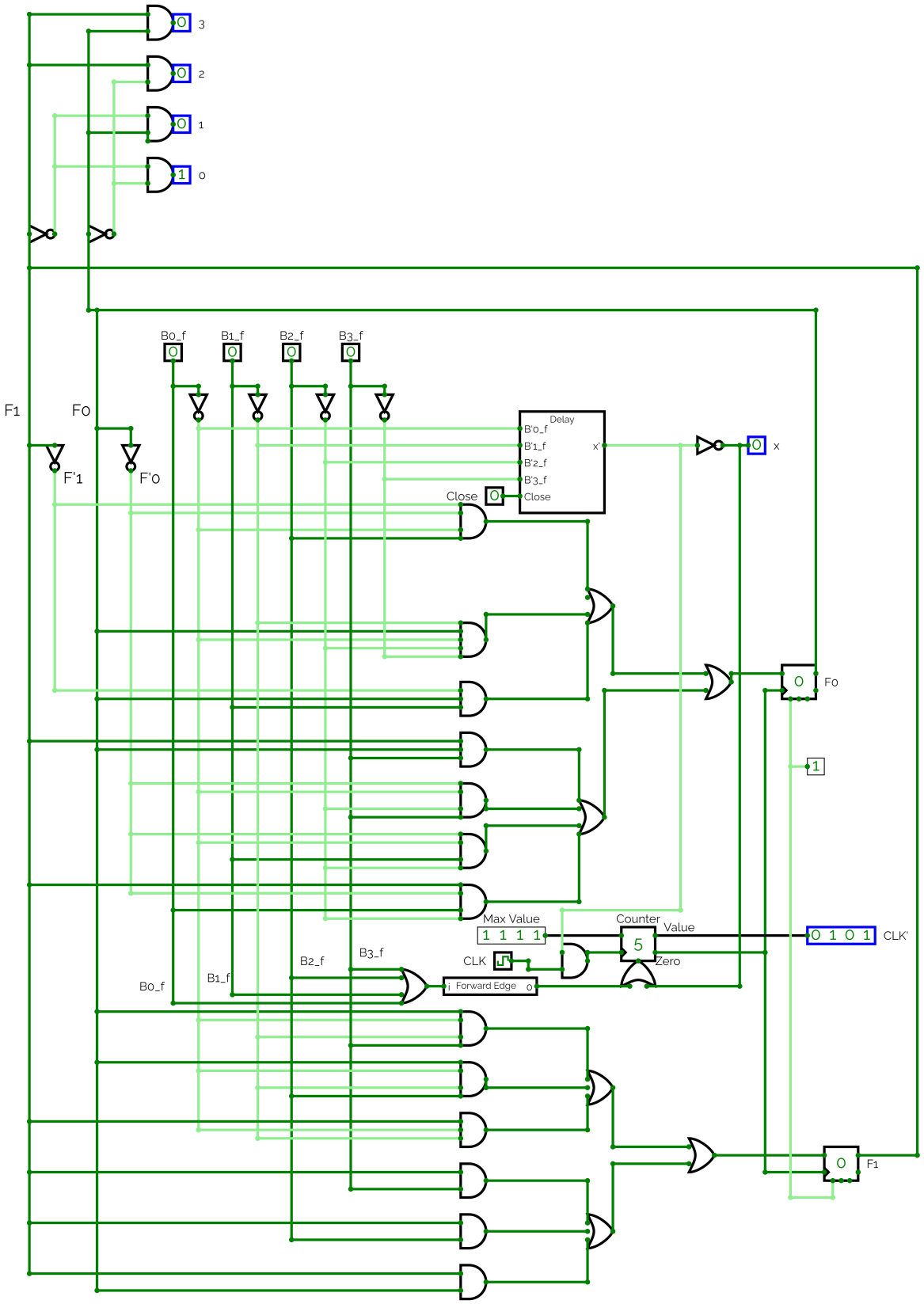


Figure 9: Main Circuit Diagram

Figure 9 gives the complete circuit diagram for the Main circuit.
Features of the Main circuit :

- Using two D-registers, we have implemented the K-map of F_1 and F_0 in the circuit. These new states represent the next step that the elevator takes, and due to the logical truth table constructed using optimal cases, the elevator will chose the optimal path. This optimal path will minimize the wait time and reach its global minima. The topmost part of the circuit represents the output, which is the floor that the elevator is currently on. When F_1,F_0 are 0,0 then the elevator is on the ground floor. F_1,F_0 are 0,1 then the elevator is in 1st floor ..etc.

- The D registers are fed using clock CLK'. This clock is constructed using a counter, which counts till Max Value = 16 in this case. The zero of CLK' produces a short pulse when the counter goes from 16 to 0. Hence CLK' = CLK*16 = 2sec. This means that the elevator will take 2 sec to change its state in general.

- Notice that the reset of Counter is connected with Forward Edge of {OR of all the buttons}. This is to ensure that whenever no buttons are active and someone presses a button the CLK' gets reset so that it will take 2 complete seconds to change state. If this was not done, then the first state change will take an arbitrary amount of time between 0 and 2 sec.

  This is the main reason why we have constructed CLK' using a counter and not just directly fed a 2sec clock. Using this circuitry we can reset the clock by resetting the counter. Theoretically, it's not possible to reset a clock because it makes no sense, so then why does this simple construction seem to work? The answer to this is that we are actually not resetting the clock exactly, rather we are reducing the error so that it is negligible. This error reduction factor is nothing but the Max Value of the counter which is 16 here.

- We have also used the Delay circuit here. This detects the negative edge of any button and produces an output x' which turns OFF for 4 seconds after the negative edge. The rest of the time x' is active. The clock that is fed to the counter is made by x'·CLK. This means that the counter will not count when x' is OFF. Also whenever x' turns OFF, x turns ON which is fed to reset the counter. Hence whenever the elevator is in state i and someone presses and releases the button i the elevator immediately opens up(the counter inside the delay starts counting), and simultaneously the CLK' is made to reset so that only after the closing of the elevator the main counter starts counting and it will take exactly 2 sec to change state. In this process, at any point in time if the button i is pressed and released once again, then the delay counter will get reset and the same procedure follows.

- The Close button works as explained in the Delay circuit

3.5 Elevator

In this circuit, we will finally bring all the components together and assemble them using the correct logic.

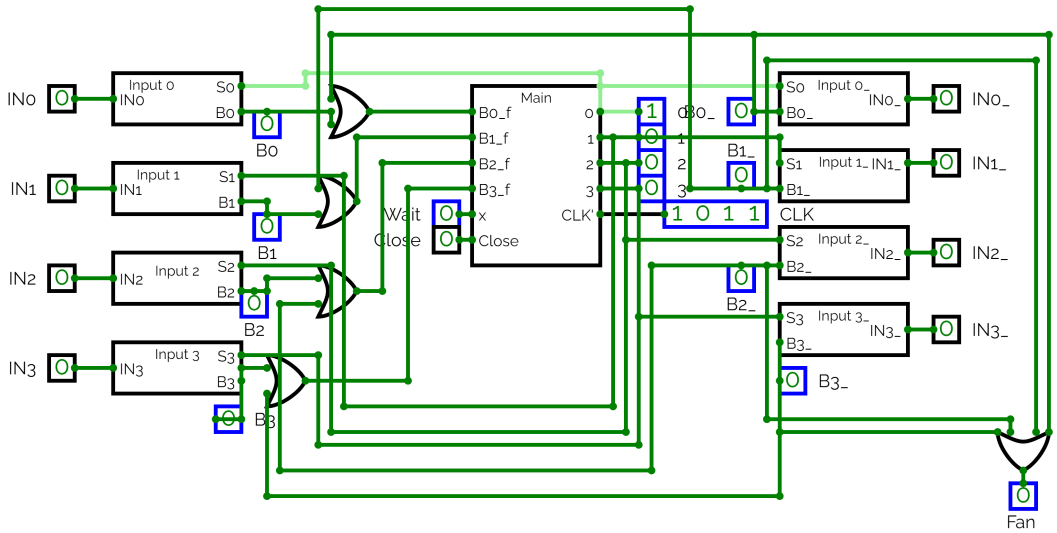


Figure 10: Elevator Circuit Diagram

Features of Elevator circuit :

- All the button circuits and Main circuits are introduced here. $B_{i\_f's}$ in the Main circuit are connected to B_i+$B_{i\_}$ for all i's.
- S_i in the buttons are connected to the State of the elevator i, which is the output of the Main circuit.
- A fan is constructed using $B_0i\_$+$B_{1\_}$+$B_{2\_}$+$B_{3\_}$. Hence whenever any button inside the elevator is active, the fan will be ON, else will remain OFF.

3.6 Final Elevator

The elevator completes the entire circuit of the optimal elevator control system that we require. The Final Elevator is the final model which represents the elevator with appropriate inputs and outputs and displays the state using LED's.

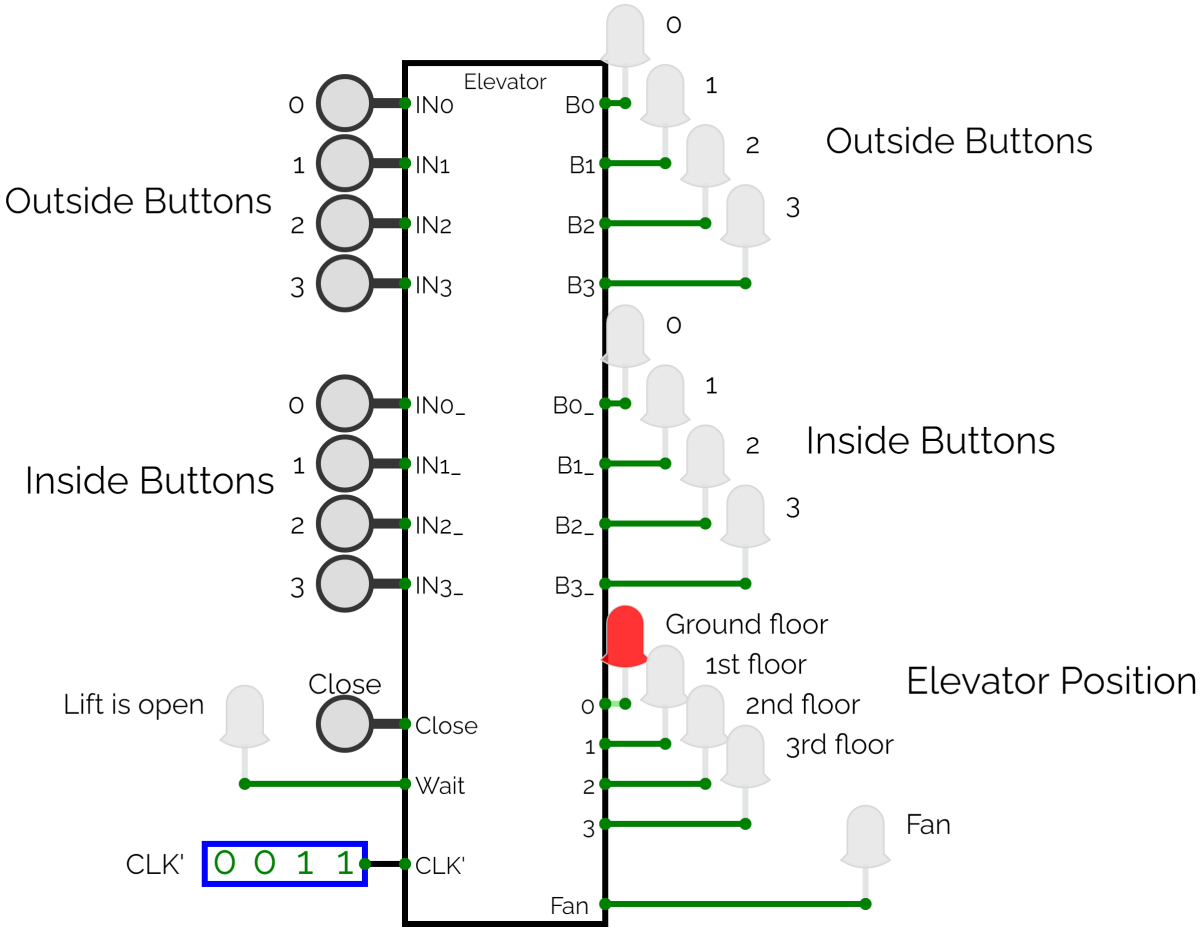


Figure 11: Final Elevator Model

3.7 FPGA Block Diagram

We have a big Module called Final Elevator which is in turn composed of multiple submodules. Both the Waiter and the Fedge are general modules that are used as detectors to detect specific signals. On the other hand, the button is an important module that is responsible for taking in the input and storing it in a convenient form. BigClk is a module that is responsible for all events related to the clock of the elevator system. The combination of all these results in our elevator control system. This system is for a 4-floor system, however, it can be easily extended for a large number of floors.

3.7.1 Button

Input: *b, s, clk*
Output: *q*

- The logic of the diagram is the same as in the circuit design for the button. We make a $q* = b + s'q$ circuit.
- It involves just storing q in a register to be able to update it in each clock cycle.
- We make use of the button module multiple times in the final circuit. We use this button for each of the 4 inner buttons and 4 outer buttons of the lift.

3.7.2 Fedge

Input: *in, clk*
Output: *edg*

- We use a similar Forward Edge Detector Logic as in the circuit design.
- We just need to store the *in* in a d-register which has a set time delay. This leads to us getting a pulse in the output whenever there is a forward edge present in the input.
- A variant of this module can be used for negative edge detection too. We just need to provide the complement of the input as *in*.

3.7.3 Waiter

Input: *w0, w1, w2, w3, clk*
Output: *o*

- The purpose of this module is to detect whenever we require the elevator to stop at a floor so that people can enter or leave.
- We detect this by using the fact that we will only open the elevator doors when the *q*(combined of inside and outside button) for that floor changes from 1 to 0.
- We just take the complement of all these values and detect the forward edge in them. In the end, we take the OR for all floors.
- If there is a pulse in the output 0 we need to stop at that floor.

3.7.4 Bigclk

Input: *close, reset, w, clk*
Output: *open, c clkout*

- The purpose of this module is to act as a Big Clock that determines when should the elevator move to the next floor. The design of this a bit different from the circuit design as Verilog has an inbuilt ability to be able to perform this task. As a result, even though the code is simple, the size of the block diagram has increased greatly.
- We use a counter which starts at a large value and decrements by 1 for each cycle of the inbuilt clk of the FPGA(rawclk).
- Along with this we need to consider different conditions. First is that when the elevator moves for the first time, we need to make sure the counter restarts at that value. To achieve this we use a positive edge detection of the OR of the button states q.
- In such cases we increase the counter to the double of the normal value and set the output *clkout* as 0. This is because each time the counter goes to 0 we invert the *clkout* and thus even if we reset the counter we may not be sure if *clkout* is 1 or 0.
- The second is the case when an elevator needs to stop at a floor. We use the same logic as above and make sure that the elevator stops for a certain amount of time whenever the output from waiter shows a pulse.
- The third case relates to when in the middle of the second case we want to suddenly end the waiting. In this case the counter is reset to the original value if the counter was previously greater. If it was smaller then waiting had already ended and so we need to make no changes.
- The result of the above calculations is sent out of the module. *clkout* refers to the bigclk we have developed, while *open* is when we are waiting. c denotes the clock cycle and is connected to an LED. Hence c changes it denotes our *clkout*.

3.7.5 Main

Input: *q0, q1, q2, q3, clk*
Output: *s0, s1, s2, s3*

- This module is the main part of our state machine. The module just performs the tasks designed by the KMaps and uses a reg to store the $f0$, and $f1$ values which denote the floor number.
- This module is also the only one that runs on the big clock. This is because the changes in this module should only take place after a variable amount of time depending on whether the lift has stopped and other factors.

3.7.6 Final Elevator

Input: $b0, b1, b2, b3, b0\_, b1\_, b2\_, b3\_, close, clkraw$
Output: $s0\_, s1\_, s2\_, s3\_, q0, q1, q2, q3, q0\_, q1\_, q2\_, q3\_, blink, fan, open$

- This module is what contains all other modules. All the input values from outside the FPGA are given here.
- b and $b\_$ represent the outside and inside buttons. Similarly, the q and $q\_$ represent the lit state of the buttons.
- The $s\_$ reflects the floor at which the elevator is present while also depicting which internal s value is what.
- The other reg *blink* indicates the big clock comes along, fan indicates if the lift has anyone inside it and *open* refers to if the lift has stopped at a floor or not.
- This module calls all the above functions and joins all components of our system together.

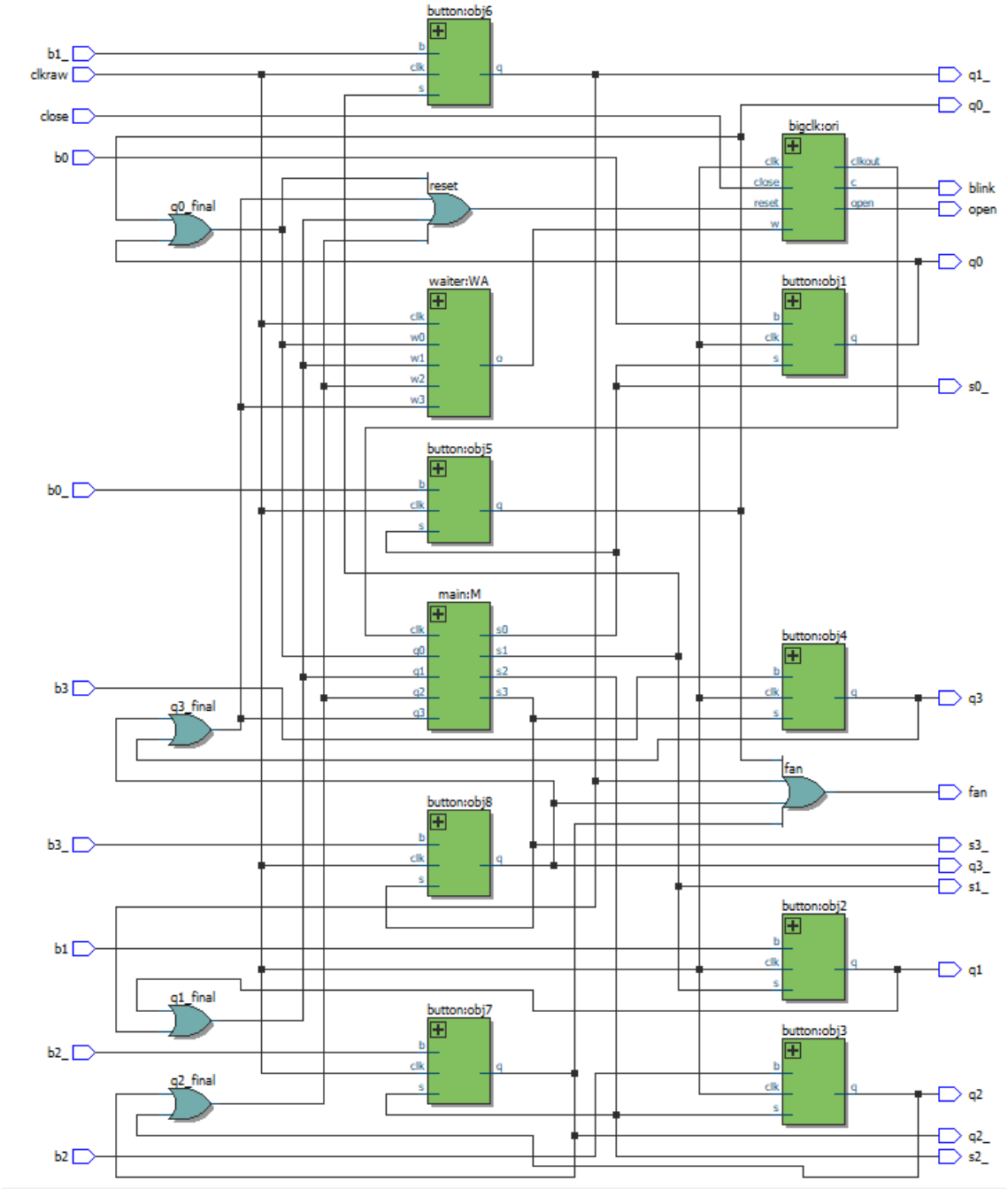
3.8 Inputs and Outputs

INPUT:

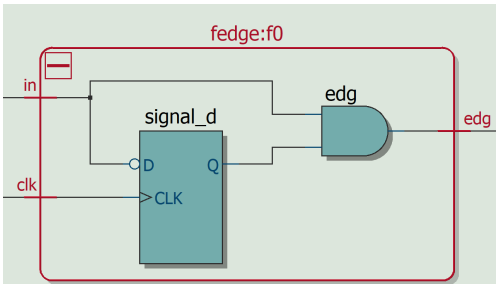
- b_i: denote the outside button of the i^{th} floor
- $b_{i\_}$: denote the inside button of the i^{th} floor
- *close*: denote if we want to forcefully want to close the lift doors(cancel the lift stop)
- *clkraw*: takes the high-frequency clock from the in-built clock of FPGA

OUTPUT:

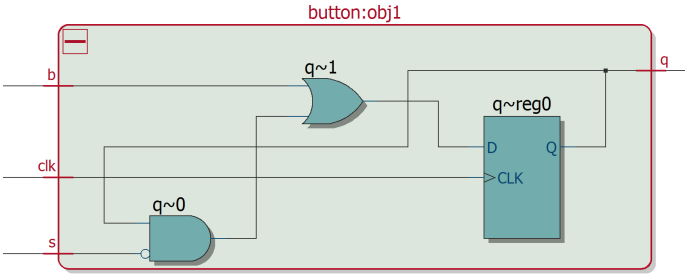
- q_i: denote the outside button state of the i^{th} floor
- $q_{i\_}$: denote the inside button state of the i^{th} floor
- $s_{i\_}$: denote whether lift is at i^{th} floor
- blink: clock cycle concerning the big clock
- fan: denotes whether any inside lift button state is on
- open: denotes whether the lift doors are open at a floor(lift is stopped at a floor)



(a) Final Elevator

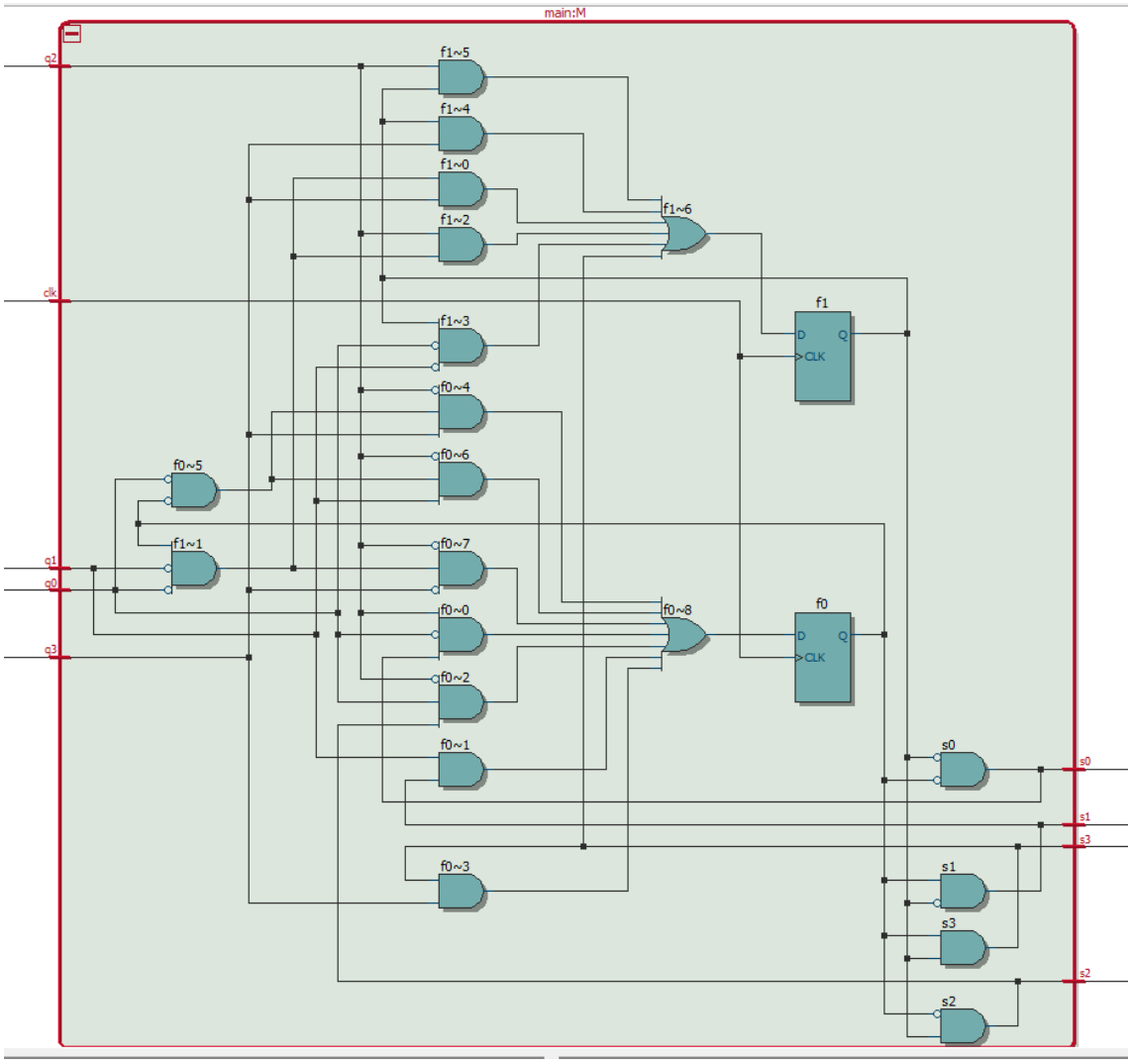


(b) Forward Edge Detector



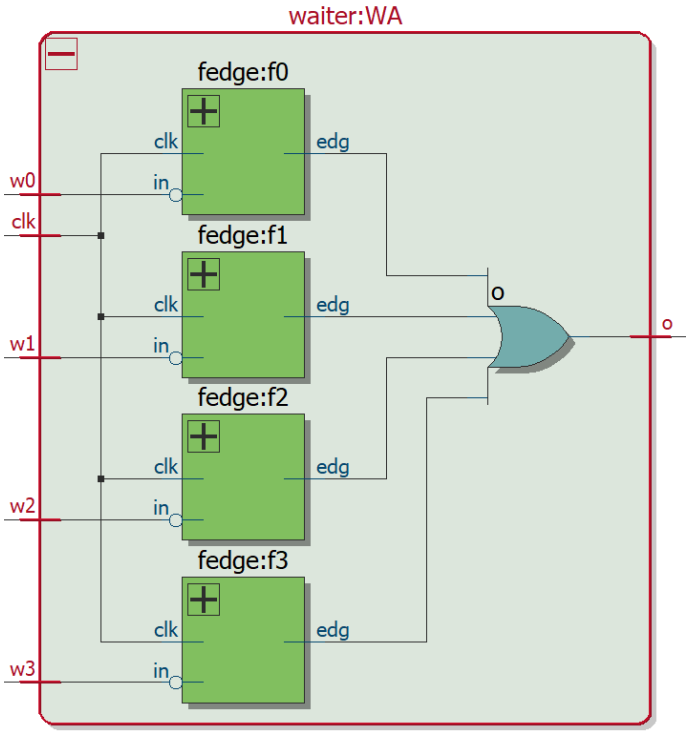
(c) Button

Figure 12: Block Diagram

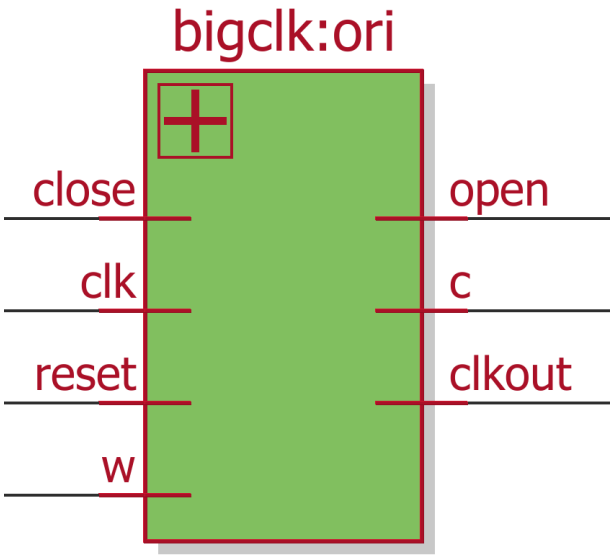


(d) Main

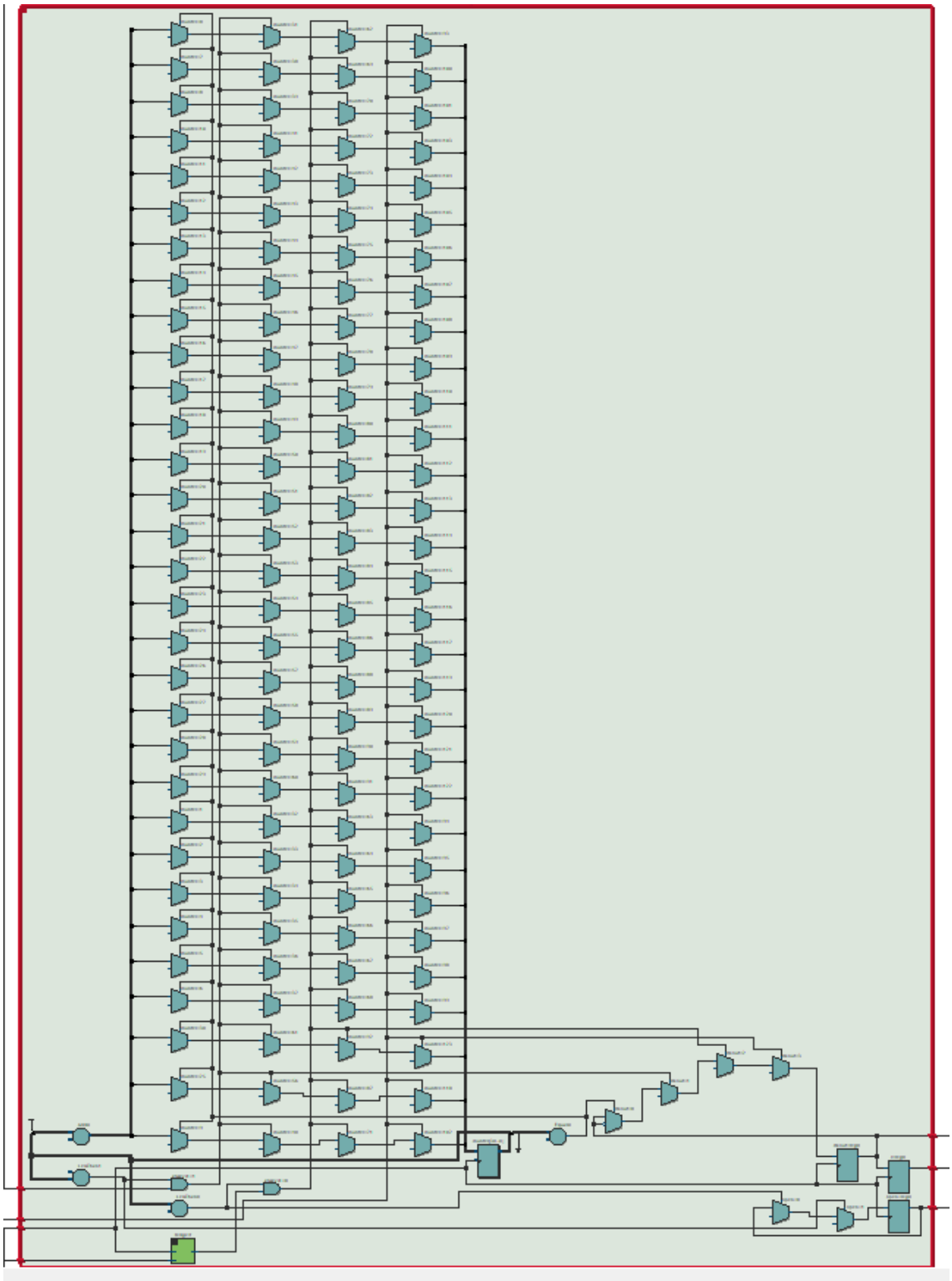
Figure 12: Block Diagram



(e) Waiter



(f) Bigclk(schematic)



(g) Bigclk

Figure 12: Block Diagram

3.9 Timing Diagram

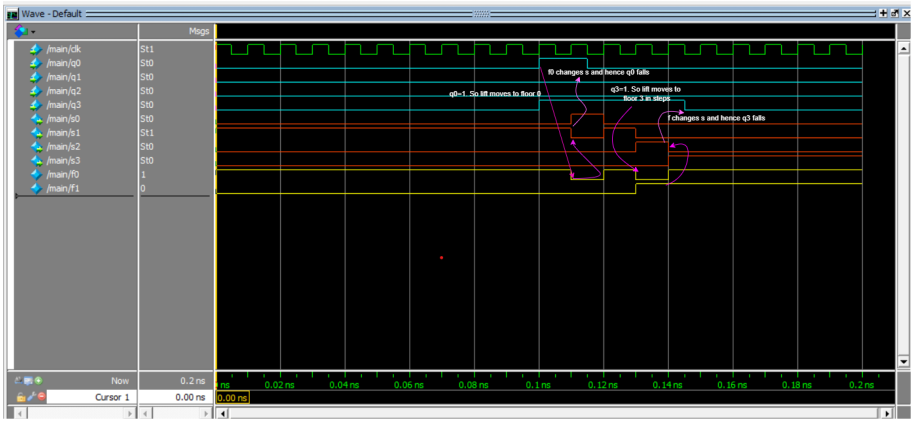
3.9.1 Button Timing Diagram

- In this we have first incresed b from 0 to 1. This causes q which represents the storing of the request to also go to 1.
- When we press s later on, this refers to the fact that the elevator has reached that floor and the button (q) stops lighting

3.9.2 Main Timing Diagram

- Here we first keep the lift on floor 1. This makes it such that $s1$ is 1. Now we make $q0$ and $q3 = 1$, so this makes $f0$ and $f1$ change according to the formula which further induces a change in the s's, and hence $q0$ becomes 0.
- In the same way, the lift travels to the third floor by influencing the $f0$ and $f1$ which lead to change in s's, and thus $q3$ also is changed to 0.

(a) Button



(b) Main

Figure 13: Timing Diagram

4 Experiment Implementation

We plan to demonstrate the following situations which will cover all the important aspects of our project. All the outputs are displayed using appropriate LED's

1. Let the initial state of the elevator be 0. Now press the button for IN_0 for some time and then release it. Keep doing this with varying press times.

   As long as the button is pressed B_0 remains active. Whenever it is not pressed, B_o is at logic 0. This represents when the elevator is at the ground floor and the button is pressed, it doesn't store the value, but rather just displays the raw input.

2. Let the initial state of the elevator be 0. Now short-press $IN_{3\_}$.

   Until the elevator reaches the 3rd floor, the button $B_{3\_}$ and the fan remains active. Once the elevator reaches state 3, the fan and button $B_{3\_}$ automatically becomes logic 0. The elevator will travel in 3*CLK' = 6 sec. This represents when a person inside the elevator short-press for the 3rd floor, the elevator takes him to his destination in 6 sec and the fan and the button remain ON during his journey.

3. Let the initial state of the elevator be 0. Short-press IN_1, IN_2, IN_3.

   The elevator will go to state 1 in 2 sec. Then the "Lift is open" will be active for 4 sec during which CLK' will be reset and will be stopped, i.e. will remain at logic 0 for 4 sec. Then the elevator takes another 2 sec to go to state 2. Again the elevator will be open for 4 sec and 2 sec later the elevator will go to state 3(fulfilling the request) and will open for 4 sec. This represents when calls are made on floor 1,2,3 when the elevator is at 0. The elevator goes to each floor and opens for some time to let the user in.

4. Let the initial state of the elevator be 2. Short-press $IN_{3\_}$ and IN_0.

   The elevator will first go to state 3 in 2sec, during which the fan will be active. Then the elevator opens for 4 sec i.e. "Lift is open" LED will turn ON for 4sec. Then the elevator goes to state 0 in 3*CLK' = 6sec and then opens again for 4 additional sec, during which the fan is OFF. This represents that when the elevator has a call and destination on the 0th and 3rd floor while being on the 2nd floor, it will first fulfill the 3rd-floor request and then the ground-floor request to minimize the wait time.

5. Let the initial state of the elevator be 3. Short-press IN_0, IN_1, IN_2. When the elevator reaches state 1, short-press the Close button

   The elevator goes to state 2 in 2 sec. Then it opens for 4 sec during which "Lift is open" is active. Then the elevator goes to state 1 in 2 sec. Then it opens and the "Lift is open" becomes active, but the elevator then instantly turns OFF due to the "Close" button that user has pressed. Then the

elevator goes to state 0 in 2 sec and again the elevator opens for 4 sec. This represents whenever the user presses the close button at a time when the elevator is open, it instantly closes.

6. Let the initial state of the elevator be 0. Short-press IN_2. When the elevator reaches state 1, keep short-pressing IN_1 for a suitable amount of time.

   The elevator goes to state 1 in 2 sec. Now as soon as the user short-presses IN_1, the elevator opens and "Lift is open" becomes active, and CLK' resets to logic 0. Now if the user again short-presses before 4sec then the elevator remains open, else it will close and go to state 2 in 2 sec. If during that 2 sec, the user short-presses IN_1, then again the elevator will open, and the same result follows. This represents whenever a person calls a floor at which the elevator is present, the elevator opens to let him in. If the user wishes, he can hold the lift by continuously short-pressing that button.

5 Connection with Real-World Problems

Elevators have become a ubiquitous aspect of modern life, particularly in densely populated urban areas. The efficiency and safety of elevators rely on their control systems, and the optimal elevator control system is a technology that has gained prominence in recent years. This technology has wide-ranging applications in various fields, including optimization.

One of the primary applications of the optimal elevator control system is building automation. Buildings often feature numerous elevators, and the control system manages their operations efficiently. The optimal elevator control system can schedule the elevators' movements accordingly, which reduces waiting times and enhances the building's overall efficiency.

Another application of the optimal elevator control system is in transportation systems. Public transportation systems, such as subways and trains, use elevators to transport passengers between different levels. The optimal elevator control system can manage traffic flow and minimize waiting times for passengers, enhancing the overall efficiency of the transportation system.

The optimal elevator control system can also be applied in manufacturing facilities. Factories that have multiple floors require elevators to transport goods and equipment between different levels. The optimal elevator control system can optimize the elevator's movements, reducing the time required to transport goods and increasing the factory's productivity.

Optimization problems in elevators are relevant to other real-world problems, as the principles underlying elevator optimization are also applicable to other domains where resource allocation and optimization are critical. For example, optimization problems in elevators involve maximizing the efficient use of resources, such as elevators, to reduce wait times and improve passenger flow. This same principle applies to traffic management in cities, where traffic signals and road networks are optimized to reduce congestion and travel time.

The elevator problem is a well-known problem in computer science and involves optimization and scheduling, two fundamental concepts in computer science. The problem focuses on determining the most efficient way to move elevators between floors to minimize waiting times for passengers while also minimizing the energy consumption of the elevator system.

Compared to this, the scan algorithm, also known as the elevator algorithm, is a disk scheduling algorithm used to optimize data access on a hard disk drive. The scan algorithm works by moving the read/write head of the disk in a single direction, serving all requests in that direction until there are no more requests in that direction. Then, the head changes direction and moves in the opposite direction, serving all requests in that direction until there are no more requests in that direction. This process continues until all requests have been served.

Although the elevator problem and the scan algorithm are not directly related, they share similarities in terms of optimization and scheduling. Both problems require careful consideration of resource allocation and scheduling to achieve maximum efficiency. The application of optimization algorithms to both problems highlights the importance of algorithm design in solving real-world problems.

In conclusion, the optimal elevator control system is a crucial technology that plays a significant role in building automation, transportation systems, and manufacturing facilities. The system's ability to optimize the scheduling and movements of elevators reduces waiting times, improves passenger and goods flow, and increases overall productivity. The principles underlying elevator optimization apply to other real-world problems, such as traffic management, where resource allocation and optimization are critical. Furthermore, the elevator problem and the scan algorithm share similarities in terms of optimization and scheduling, emphasizing the importance of algorithm design in solving real-world problems. The optimization of elevator systems and related problems is an essential area of study that has the potential to significantly impact various domains and improve efficiency and safety.

6 Appendix

6.1 Model and Simulation

- For complete model and demonstration of our Optimal Elevator Control System click on See working model or go to url `https://drive.google.com/file/d/1GGIFM7KNHP8g0MXbFlU-d6KjrM3G9f-2/view?usp=sharing`
- For a full simulation of this elevator system, click on Launch Elevator simulation or go to the url `https://circuitverse.org/simulator/embed/elevator-3b7bc22d-15b0-46e5-8d0c-837a5bf0b769`

6.2 FPGA Code

The following code represents what was used in our project as a final product.

```
1  module button(input b, input s, output reg q, input clk);
2      always @ (posedge clk) begin
3          q = ((q&(~s))| b);
4      end
5  endmodule
6
7  module fedge(input in, output edg, input clk);
8      reg signal_d;
9      always @(posedge clk) begin
10         signal_d <= #1 ~in;
11     end
12     assign edg = in & (signal_d);
13 endmodule
14
15 module waiter(input w0,w1,w2,w3, clk, output o);
16     wire w0_,w1_,w2_,w3_;
17     wire x0, x1, x2, x3;
18     assign w0_=~w0;
19     assign w1_=~w1;
20     assign w2_=~w2;
21     assign w3_=~w3;
22     fedge f0(w0_, x0, clk);
23     fedge f1(w1_, x1, clk);
24     fedge f2(w2_, x2, clk);
25     fedge f3(w3_, x3, clk);
26     assign o = x0 | x1 | x2 | x3;
27 endmodule
28
29 module bigclk(input clk, input reset, input w, input close
       , output reg clkout, output reg c, output reg open);
30     reg [30:0] counter;
31
32     wire edge_detected;
33     fedge f(reset, edge_detected, clk);
34     always @ (posedge clk) begin
35         if(counter>99999999) begin
```

```
36              open <= 1;
37          end else if (counter<99999999) begin
38              open <= 0;
39          end
40      end
41
42
43      always @(posedge clk) begin
44           if (w) begin //wait if lift opens
45                counter <= 299999999;
46                clkout <=0;
47           end else if (edge_detected & counter<99999999)
                begin   //new cycle starts
48                counter <= 99999999;
49                clkout <=0;
50           end else if(close & (counter>99999999))begin    //
                forceful close
51                counter <= 99999999;
52                clkout <=0;
53           end else begin
54                if (counter == 0) begin
55                     counter <= 49999999;
56                     clkout <= ~clkout;
57                end else begin
58                     counter <= counter - 1;
59                end
60           end
61      end
62
63      always @(posedge clk) begin
64                c <= clkout;
65      end
66  endmodule
67
68  module main(q0, q1, q2, q3, s0, s1, s2, s3, clk);
69      input q0, q1, q2, q3, clk;
70      output s0, s1, s2, s3;
71      reg f0, f1;
72      always @ (posedge clk) begin
73           f0 <= (~f1 & ~f0 & ~q0 & q2) | (~f1 & f0 & q1) | (
                f1 & ~f0 & q0 & ~q2) | (f1 & f0 & q3) | (~f0 &
                ~q0 & ~q2 & q3) | (~f0 & ~q0 & q1 & ~q2) | (f0
                & ~q0 & ~q1 & ~q2 & ~q3);
74           f1 <= (f0 & ~q0 & ~q1 & q3) | (f0 & ~q0 & ~q1 & q2
                ) | (f1 & ~q0 & ~q1) | (f1 & q3) | (f1 & q2) |
                (f1 & f0);
75      end
76      assign s0 = ~f1 & ~f0;
77      assign s1 = ~f1 & f0;
78      assign s2 = f1 & ~f0;
```

```
79      assign s3 = f1 & f0;
80  endmodule
81
82
83  module FinalElevator(b0, b1, b2, b3, b0_, b1_, b2_, b3_,
        q0, q1, q2, q3, q0_, q1_, q2_, q3_, s0_, s1_, s2_ ,s3_,
        close, clkraw, blink, fan, open);
84      input b0, b1, b2, b3, b0_, b1_, b2_, b3_, close,
            clkraw;
85      wire q0_final, q1_final, q2_final, q3_final;
86      output s0_, s1_, s2_, s3_;
87      output blink, fan, open;
88      wire clk, k, reset, w;
89      output q0, q1, q2, q3, q0_, q1_, q2_, q3_;
90
91      waiter WA(q0_final, q1_final, q2_final, q3_final,
            clkraw, w);
92      bigclk ori(clkraw, reset, w, close, clk, k, open);
93      assign blink = k;
94
95      button obj1(b0, s0, q0, clkraw);
96      button obj2(b1, s1, q1, clkraw);
97      button obj3(b2, s2, q2, clkraw);
98      button obj4(b3, s3, q3, clkraw);
99
100     button obj5(b0_, s0, q0_, clkraw);
101     button obj6(b1_, s1, q1_, clkraw);
102     button obj7(b2_, s2, q2_, clkraw);
103     button obj8(b3_, s3, q3_, clkraw);
104
105     assign q0_final = q0 | q0_;
106     assign q1_final = q1 | q1_;
107     assign q2_final = q2 | q2_;
108     assign q3_final = q3 | q3_;
109
110     assign reset = q0_final | q1_final | q2_final |
            q3_final;
111     assign fan = q0_ | q1_ | q2_ | q3_;
112
113     main M(q0_final, q1_final, q2_final, q3_final, s0, s1,
            s2, s3, clk);
114
115     assign s0_ = s0;
116     assign s1_ = s1;
117     assign s2_ = s2;
118     assign s3_ = s3;
119
120 endmodule
```