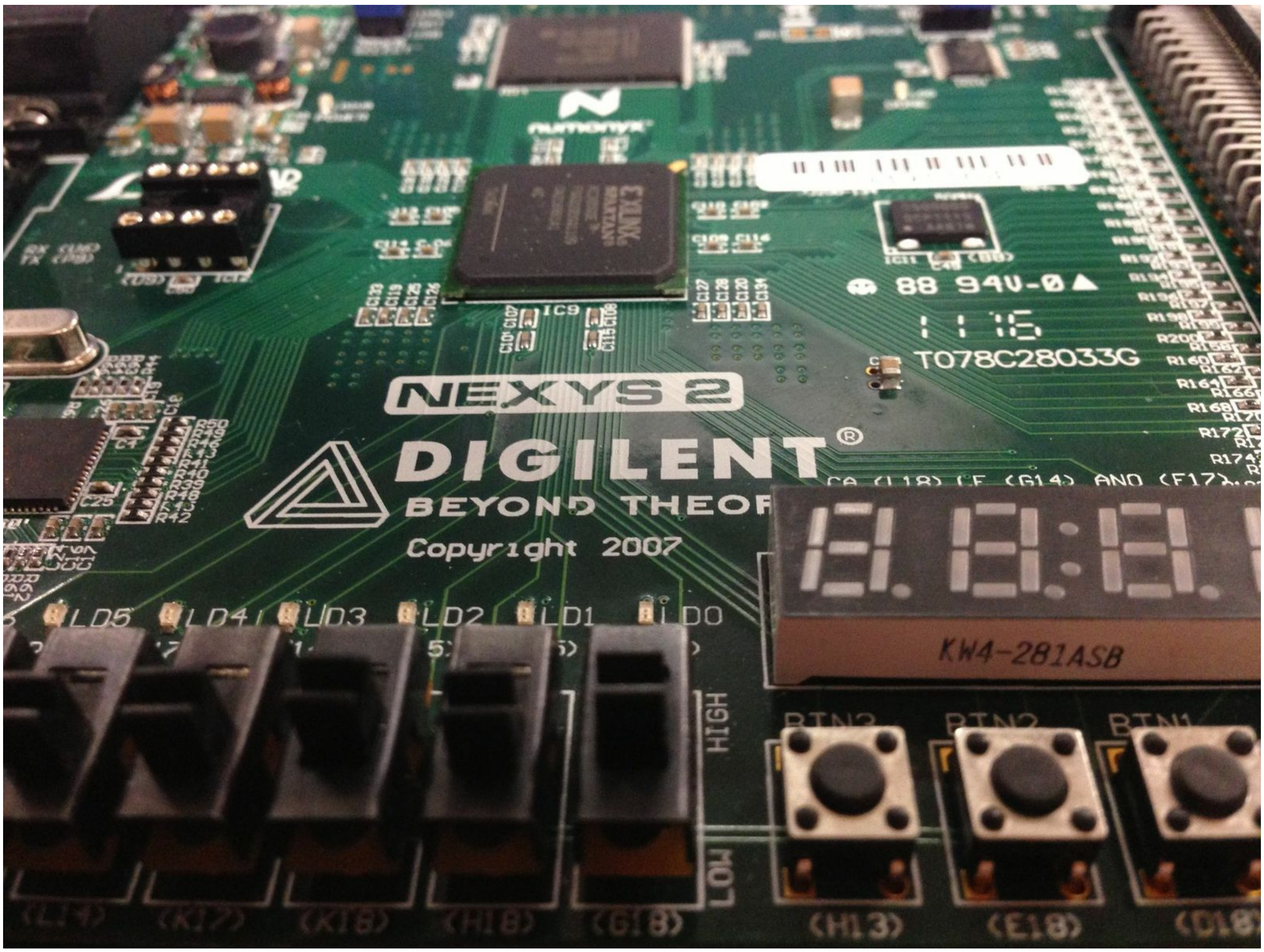
Elevator Project

Madison Traynham

4/23/13



Contents

Problem Description 3
FSM 3
Implementation of the FSM 5
  Truth Table: 5
  Karnaugh Map: 5
Cases 6
    Things that worked 6
    FSM Routine: 6
    Things that did not work 6
Future Work 7
Appendix A: Elevator Controller FSM 8
Appendix B: High Level State Machine 9
    Controller: 9
    Datapath: 10
Appendix C: 11

Problem Description

A four story building has been designed that needs a single bidirectional elevator to travel between the floors. The elevator will have a call button on each floor and floor select buttons on the inside of the elevator. Other buttons inside the elevator include an emergency stop and open door button. An elevator sensor is included and it will determine when the cart is in line with the doors. The sensor will allow open the doors on the correct floor when it is high.

FSM

The elevator used in the project contains a controller heavy FSM and has very little happening in the datapath. The controller logic is dictated by the FSM given in class and it was slightly adapted to fit the needs of the Nexys 2 board. The final FSM of the Controller can be seen in Figure 1 below.

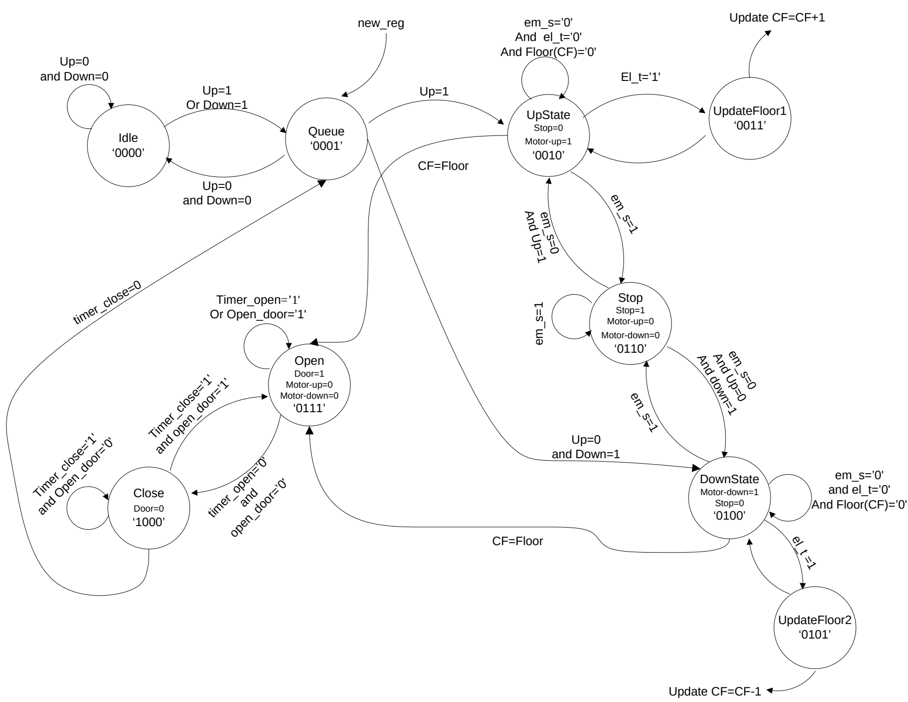


Figure 1: Controller FSM adapted from the one given in class.

There are nine states included in the FSM for the controller those are Idle, Queue, Up State, Update Floor 1, Stop, Down State, Update Floor 2, Open, and Close. These states are fairly self-explanatory and are used to control the movement and direction of the elevator. The inputs and outputs of the Controller can be referenced in the VHDL code included in the zip file.

The high level state machine, which contains the datapath and controller components for the elevators operation can be seen in Appendix B and below. The datapath FSM feeds in the new_req into the queue. The update states are taken care of inside of the datapath FSM. The timers that control the door opening and closing are also inside of the datapath.

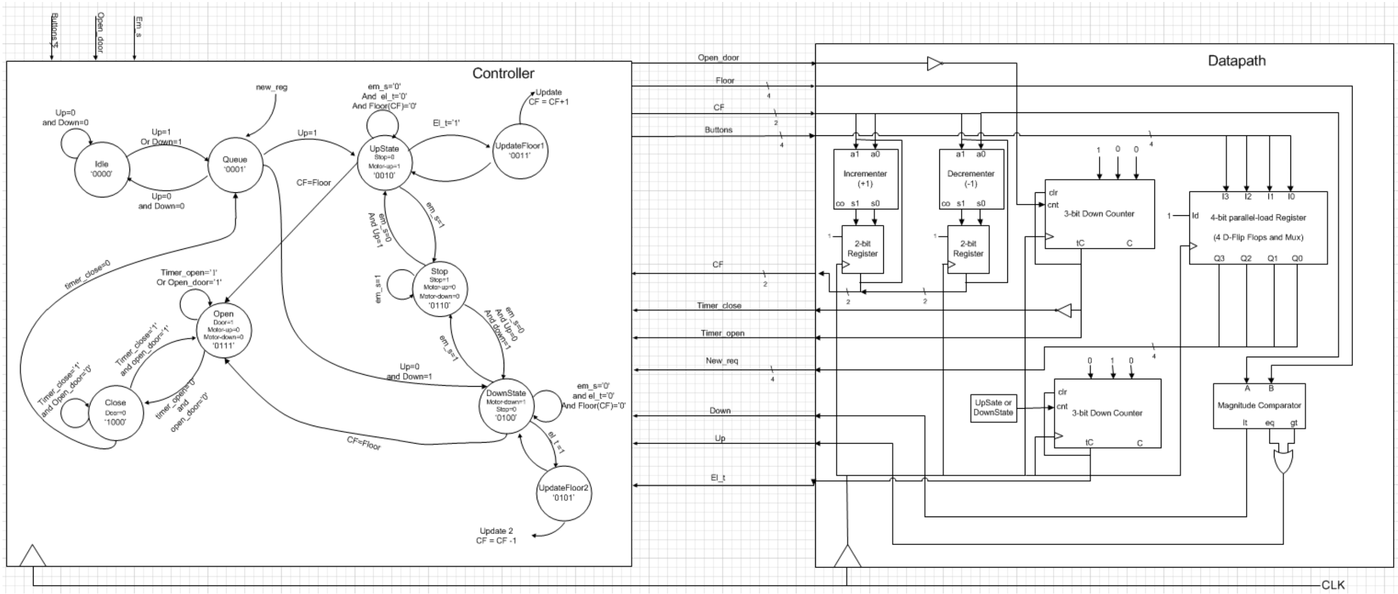


Figure 2: HLSM of the Elevator Controller

Implementation of the FSM

Truth Table:

In order to follow the RTL design process a truth table must be derived in order for the Karnaugh maps and circuit diagrams to be created. The truth table was derived by following each state and its state bits in the FSM. The input and output values along with don't care values were recorded in the table as seen below.

Elevator Truth Table

	Inputs												Outputs								
Current State	s0	s1	s2	s3	Up	Down	Em_s	El_top	Floor(CF)	Open/Close	OpenTimer	CloseTimer	Stop	Door	Motor-Up	Motor-Down	n0	n1	n2	n3	Next State
Idle	0	0	0	0	0	0	X	X	X	X	X	X	0	0	0	0	0	0	0	0	Idle
	0	0	0	0	1	X	X	X	X	X	X	X	0	0	0	0	0	0	0	1	Queue
	0	0	0	0	X	1	X	X	X	X	X	X	0	0	0	0	0	0	0	1	Queue
Queue	0	0	0	1	0	0	X	X	X	X	X	X	0	0	0	0	0	0	0	0	Idle
	0	0	0	1	1	X	X	X	X	X	X	X	0	0	0	0	0	0	1	0	UpState
	0	0	0	1	0	1	X	X	X	X	X	X	0	0	0	0	0	1	0	0	DownState
UpState	0	0	1	0	X	X	0	0	0	X	X	X	0	0	1	0	0	0	1	0	Upstate
	0	0	1	0	X	X	X	1	X	X	X	X	0	0	1	0	0	0	1	1	Update-Up
	0	0	1	0	X	X	1	X	X	X	X	X	0	0	1	0	0	1	1	0	Emergency Stop
	0	0	1	0	X	X	X	X	1	X	X	X	0	0	1	0	0	1	1	1	Open
Update-Up	0	0	1	1	X	X	X	X	X	X	X	X	0	0	0	0	0	0	1	0	Upstate
DownState	0	1	0	0	X	X	0	0	0	X	X	X	0	0	0	1	0	1	0	0	DownState
	0	1	0	0	X	X	X	1	X	X	X	X	0	0	0	1	0	1	0	1	Update-Down
	0	1	0	0	X	X	1	X	X	X	X	X	0	0	0	1	0	1	1	0	Emergency Stop
	0	1	0	0	X	X	X	X	1	X	X	X	0	0	0	1	0	1	1	1	Open
Update-Down	0	1	0	1	X	X	X	X	X	X	X	X	0	0	0	0	0	0	1	0	DownState
Emergency Stop	0	1	1	0	X	X	1	X	X	X	X	X	1	0	0	0	0	1	1	0	Emergency Stop
	0	1	1	0	1	X	0	X	X	X	X	X	1	0	0	0	0	0	1	0	UpState
	0	1	1	0	0	1	0	X	X	X	X	X	1	0	0	0	0	1	0	0	DownState
Open	0	1	1	1	X	X	X	X	X	1	X	X	0	1	0	0	0	1	1	1	Open
	0	1	1	1	X	X	X	X	X	X	1	X	0	1	0	0	0	1	1	1	Open
	0	1	1	1	X	X	X	X	X	0	0	X	0	1	0	0	1	0	0	0	Closed
Closed	1	0	0	0	X	X	X	X	X	0	X	1	0	0	0	0	1	0	0	0	Closed
	1	0	0	0	X	X	X	X	X	1	X	1	0	0	0	0	0	1	1	1	Open
	1	0	0	0	X	X	X	X	X	X	X	0	0	0	0	0	0	0	0	1	Queue

Table 1: Truth Table Implementation of the Controller FSM.

Karnaugh Map:

After the truth table has been completed the Karnaugh map can be generated. There are 12 input bits so the Karnaugh map will be 6 bits by 6 bits resulting in a table that is 64 by 64 wide for 4 separate outputs, or 4096 instances times 4. This is pretty excessive for a Karnaugh map and there appeared to be an easier way using the Quine-McCluskey (QM) Method. The QM method uses the minterms of each output and determines the most reduced Boolean equation. Due to constraints in time and knowledge gaps I was unable to complete the QM method in reducing all of the don't care instances. The Karnaugh maps became too cumbersome and other priorities took its place instead.

Cases

Throughout the project I ran into many issues with the coding of the elevator. The biggest issue that I had was the switch from thinking in an object oriented programming language like C++ or Java and thinking in a VHDL state of mind. When coding the syntax issues that arose were due to trying to execute code in a sequence like in C++ instead of VHDL's "everything at once" approach.

Things that worked

I managed to get the code working correctly in the same fashion that it is displayed on the FSM. A test bench was also designed to control the possible situations that might arise in testing the elevator. I created a test bench to use the following routine. This routine tested most of the possible situations that would be encountered when using the elevator.

FSM Routine:

1. Go to 1st Floor
2. Clear button input
3. Go to 4th floor
4. Clear button input
5. Go down to 3rd floor
6. Clear button input
7. Go down to 1st floor from 3rd
8. Emergency Stop
9. Continue going to 1st
10. Clear button input
11. Set Open/Close to 1 and hold door open on the 1st floor
12. Clear button input
13. Goes from 1st floor to 3rd and 4th with multiple up requests.
14. Clear button input
15. Goes from 4th floor down to 2nd and to 1st with multiple down requests
16. Clear button input

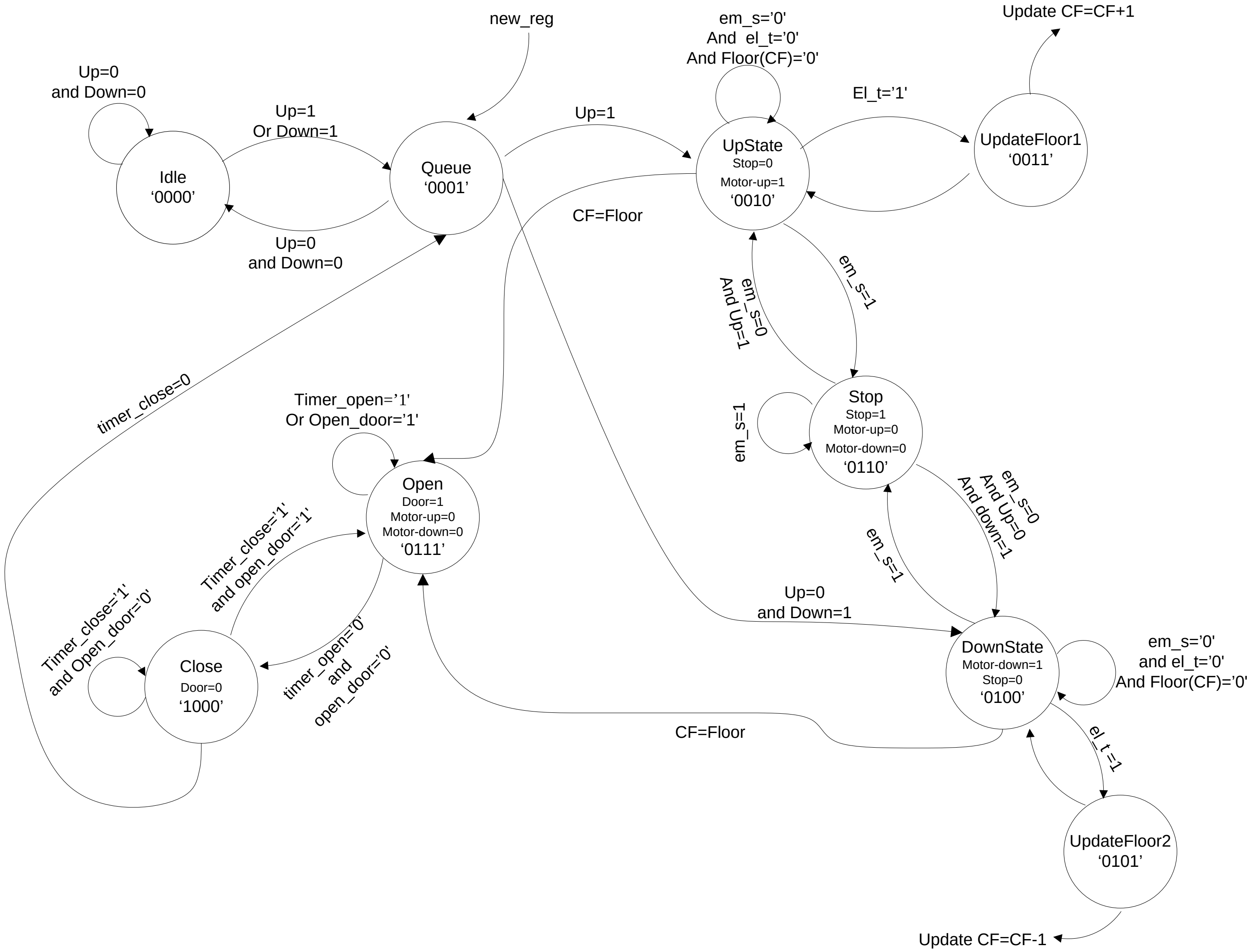
Things that did not work

After completing the entirety of the elevator code that was due I decided to try and implement the seven segment display to display not only the current floor but also current state of the doors. The door display brought errors because of the shared anode that the seven segment displays share. To counter act this issue a mux must be implemented to control the input lines to the anode and switch between the possibilities at the speed of the clock. This would switch between the two active anodes at such a fast rate that the eye would not be able to tell the difference. I ran out of time before I was able to get it completed.

Future Work

In the future if time allows I would like to implement the seven segment display method described above. Other interesting work that could be done on the elevator includes two elevator supports, algorithms to control better queue priority and possibly the implementation of a speaker to announce the floor that the elevator approaches.

Appendix A: Elevator Controller FSM

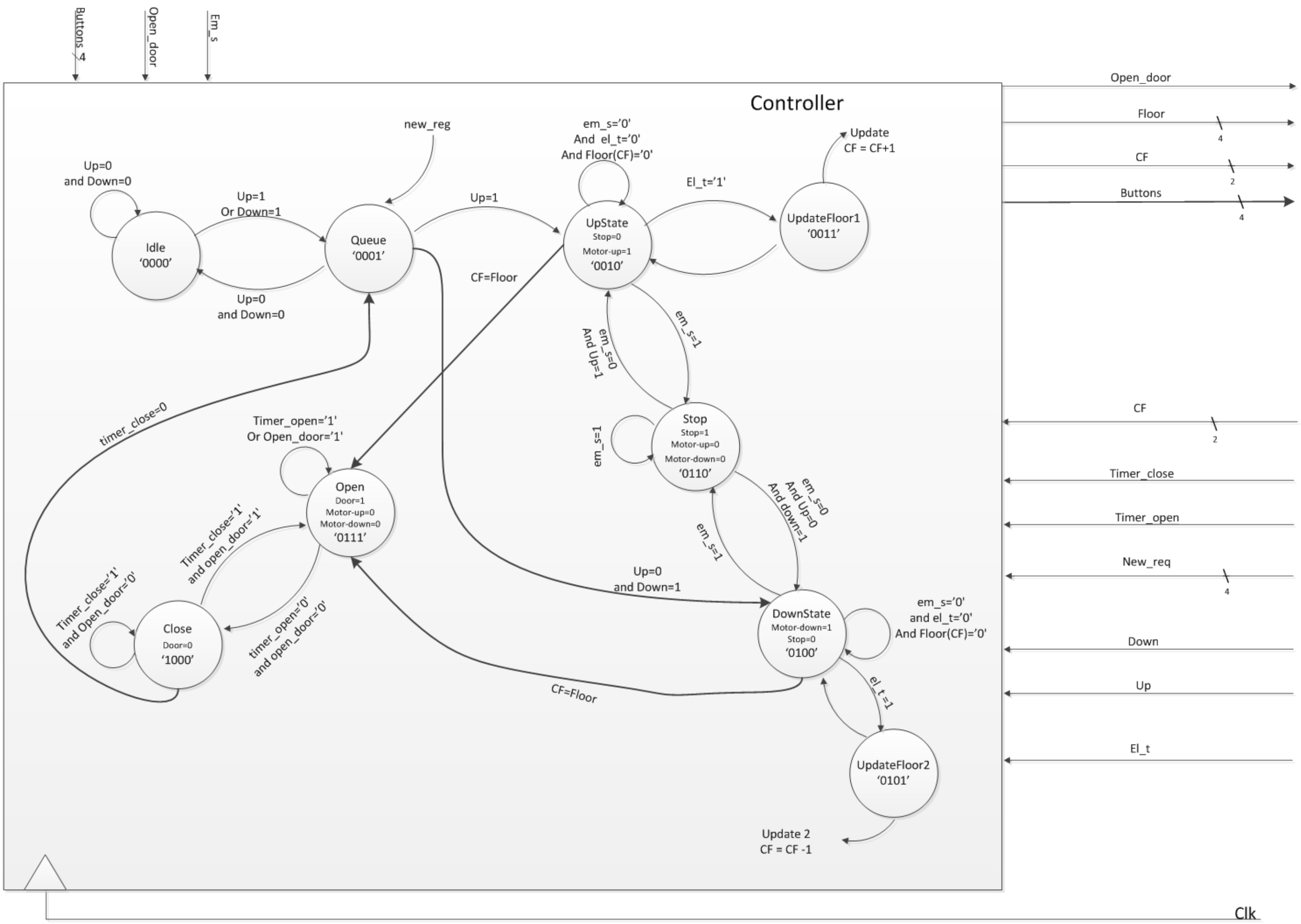


Inputs: s0, s1, s2, s3, Up, Down, Em_s, El_top,Floor(CF), Open/Close, and CloseTimer

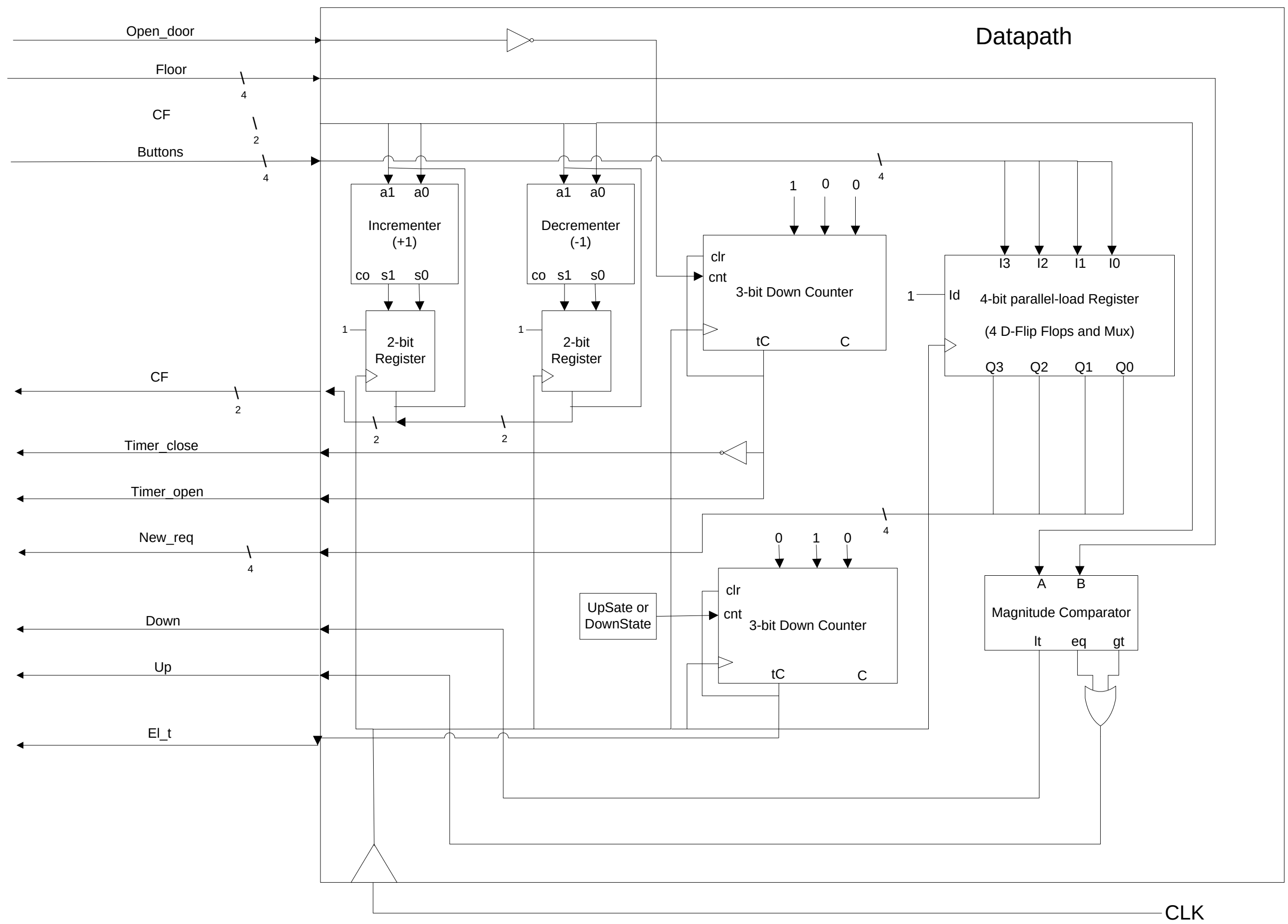
Outputs: Motor_up, Motor_down, Door, and Stop

Appendix B: High Level State Machine

Controller:



Datapath:



Datapath
Open_door
Floor
4
CF
2
Buttons
4
a1
a0
Incrementer
(+1)
co
s1
s0
a1
a0
Decrementer
(-1)
co
s1
s0
1
2-bit
Register
1
2-bit
Register
1 0 0
clr
cnt
3-bit Down Counter
tC
C
I3
I2
I1
I0
1
ld
4-bit parallel-load Register
(4 D-Flip Flops and Mux)
Q3
Q2
Q1
Q0
CF
2
2
2
Timer_close
Timer_open
New_req
4
4
0 1 0
clr
cnt
3-bit Down Counter
tC
C
UpSate or DownState
A
B
Magnitude Comparator
lt
eq
gt
Down
Up
El_t
CLK

Appendix C: Truth Table for Controller

Elevator Truth Table

	Inputs												Outputs								
Current State	**s0**	**s1**	**s2**	**s3**	**Up**	**Down**	**Em_s**	**El_top**	**Floor(CF)**	**Open/Close**	**OpenTimer**	**CloseTimer**	**Stop**	**Door**	**Motor-Up**	**Motor-Down**	**n0**	**n1**	**n2**	**n3**	**Next State**
Idle	0	0	0	0	0	0	X	X	X	X	X	X	0	0	0	0	0	0	0	0	Idle
	0	0	0	0	1	X	X	X	X	X	X	X	0	0	0	0	0	0	0	1	Queue
	0	0	0	0	X	1	X	X	X	X	X	X	0	0	0	0	0	0	0	1	Queue
Queue	0	0	0	1	0	0	X	X	X	X	X	X	0	0	0	0	0	0	0	0	Idle
	0	0	0	1	1	X	X	X	X	X	X	X	0	0	0	0	0	0	1	0	UpState
	0	0	0	1	0	1	X	X	X	X	X	X	0	0	0	0	0	1	0	0	DownState
UpState	0	0	1	0	X	X	0	0	0	X	X	X	0	0	1	0	0	0	1	0	Upstate
	0	0	1	0	X	X	X	1	X	X	X	X	0	0	1	0	0	0	1	1	Update-Up
	0	0	1	0	X	X	1	X	X	X	X	X	0	0	1	0	0	1	1	0	Emergency Stop
	0	0	1	0	X	X	X	X	1	X	X	X	0	0	1	0	0	1	1	1	Open
Update-Up	0	0	1	1	X	X	X	X	X	X	X	X	0	0	0	0	0	0	1	0	Upstate
DownState	0	1	0	0	X	X	0	0	0	X	X	X	0	0	0	1	0	1	0	0	DownState
	0	1	0	0	X	X	X	1	X	X	X	X	0	0	0	1	0	1	0	1	Update-Down
	0	1	0	0	X	X	1	X	X	X	X	X	0	0	0	1	0	1	1	0	Emergency Stop
	0	1	0	0	X	X	X	X	1	X	X	X	0	0	0	1	0	1	1	1	Open
Update-Down	0	1	0	1	X	X	X	X	X	X	X	X	0	0	0	0	0	0	1	0	DownState
Emergency Stop	0	1	1	0	X	X	1	X	X	X	X	X	1	0	0	0	0	1	1	0	Emergency Stop
	0	1	1	0	1	X	0	X	X	X	X	X	1	0	0	0	0	0	1	0	UpState
	0	1	1	0	0	1	0	X	X	X	X	X	1	0	0	0	0	1	0	0	DownState
Open	0	1	1	1	X	X	X	X	X	1	X	X	0	1	0	0	0	1	1	1	Open
	0	1	1	1	X	X	X	X	X	X	1	X	0	1	0	0	0	1	1	1	Open
	0	1	1	1	X	X	X	X	X	0	0	X	0	1	0	0	1	0	0	0	Closed
Closed	1	0	0	0	X	X	X	X	X	0	X	1	0	0	0	0	1	0	0	0	Closed
	1	0	0	0	X	X	X	X	X	1	X	1	0	0	0	0	0	1	1	1	Open
	1	0	0	0	X	X	X	X	X	X	X	0	0	0	0	0	0	0	0	1	Queue