ECE 451 LAB 7
Subway Signal Control Logic II

Ishani Gowaikar
CSU ID 831702439

Objective:

In this lab we extend the previously designed Subway logic controller to include provisions for Direction (D) control for a given configuration.

Lab Description (specs):

This is the continuation of Lab 3. In Lab 3, the direction signal D is given as a primary input signal. Now, let's design a finite state machine to generate signal D. Let's assume that the traffic pattern for the track is such when two consecutive left-to-right trains passed by, the next train must go from right-to-left. After that, there will be two more left-to-right trains, etc. To detect the train direction through the station, we set up two light beams just above the rail track and place two photocells, P1 and P2, some distance apart. Assume the train can fit inside of the two beams. When the beam shines on a photocell, it produces a 0, and when the beam is blocked, it produces a 1. The train may stop at the middle of the photocells. If this happens, no change is made to signal D. If the train stopping at the middle and then back to the direction it came from, then this train does not count as a passing train. Design a logic circuit to generate D.

Approach:

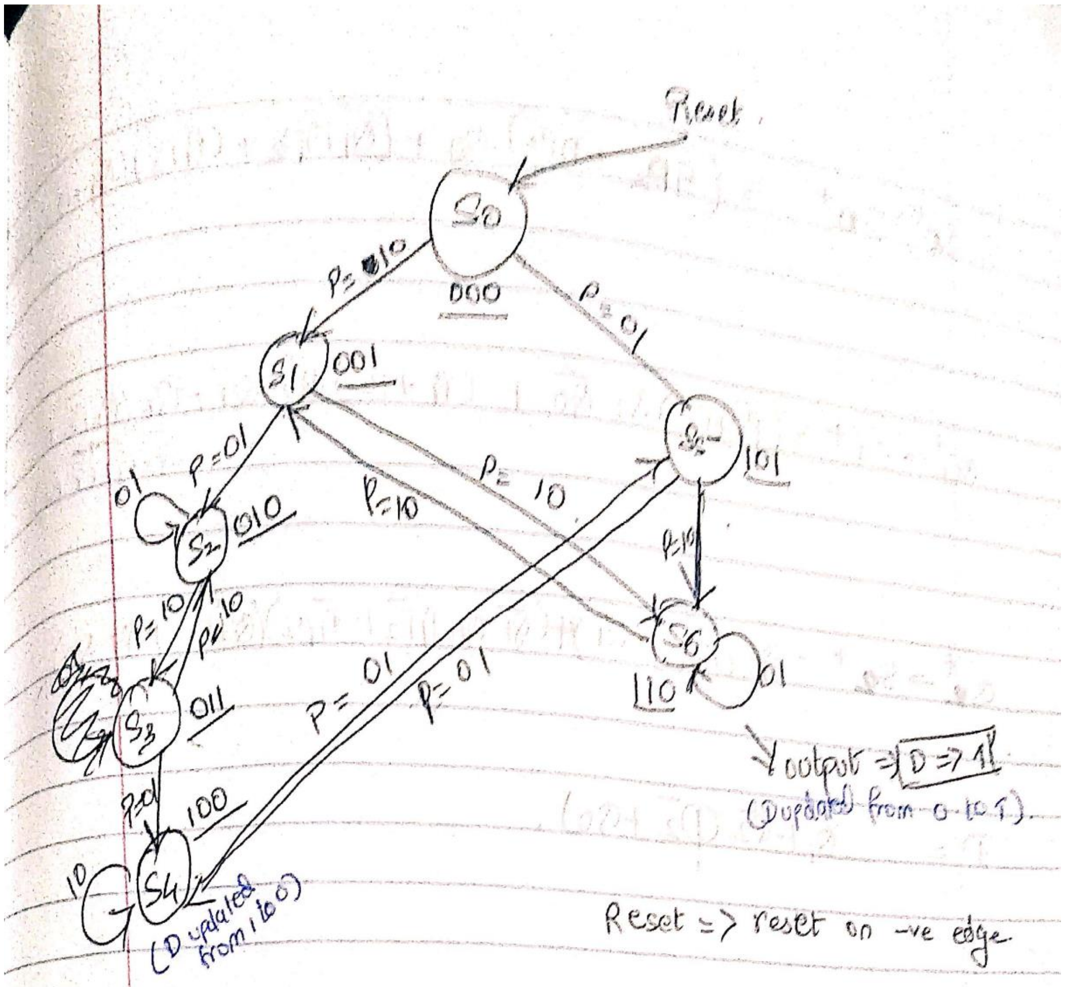
- We first construct the state transition diagram (refer prelab) for the given specs and use state priority as discussed in class to perform state encoding
- We then write the state transition table using the state transition diagram and decided state encoding.
- After deciding upon the type of flip-flop we intend to use, we make the truth table and then generate the equations for combinational logic blocks using Quine-Mclusky method and use these to prepare the cadence schematic.
- For the Verilog model, we use the transition diagram and construct the model using if-else and case constructs.

Inputs:
P1, P2: Photocell inputs (clubbed into 2 bit bus P [P1, P2] for simulations)
Reset: global reset signal (reset on negative edge)

Outputs:
D: Direction signal for the subway controller
State (Q): State outputs (S0 to S6: 3 bit state outputs)

Description of States:
S0: Reset state (next state S1 if P = 10, S5 if P=01)
S1: 1 Train from left crossed sensor P1 **(next state S2 if P=01, S6 if P=10 (train goes back to left, thus not counted)).**
S2: Train coming from left crosses sensor P2 **(1 train passed from left to right, next state S3 if P =10, else self-transition).**
S3: 2nd train coming from left crosses P1 **(next state = S4 if P = 01, S2 if P =10 (train goes back to left, not counted)).**
S4: 2nd train from left crosses P2 (**2nd train successfully goes from left to right, Direction (D) updated from 1 to 0, next state = S5 if P=01, else self-transition**)
S5: 1st train from right crosses sensor P2 first (**next state= S6 if P =10, S4 if P =01(train from right back to right, not counted**))
S6: 1st train from right crosses sensor P1 (**successfully passed from right to left, Direction D updated to 1, next state = 1 if P =10, else self-transition with output D =1**).

State transition diagram and table:



Reset => reset on -ve edge

Current State		P1	P2		Next State	D
S0	000	0	1	S5	101	1
		1	0	S1	001	1
S1	001	0	1	S2	010	1
		1	0	S6	110	1
S2	010	0	1	S2	010	1
		1	0	S3	011	1
S3	011	0	1	S4	100	1
		1	0	S2	010	1
S4	100	0	1	S5	101	0
		1	0	S4	100	0
S5	101	0	1	S4	100	0
		1	0	S6	110	0
S6	110	0	1	S6	110	0
		1	0	S1	001	1

$$Q_0^+ \to S_0^+ = (\overline{P_1}P_2 + P_1P_2) \cdot Q_0 + [\overline{Q_1}\,\overline{P_1}P_2 + (P_1\overline{P_2})(Q_1+Q_2)]$$

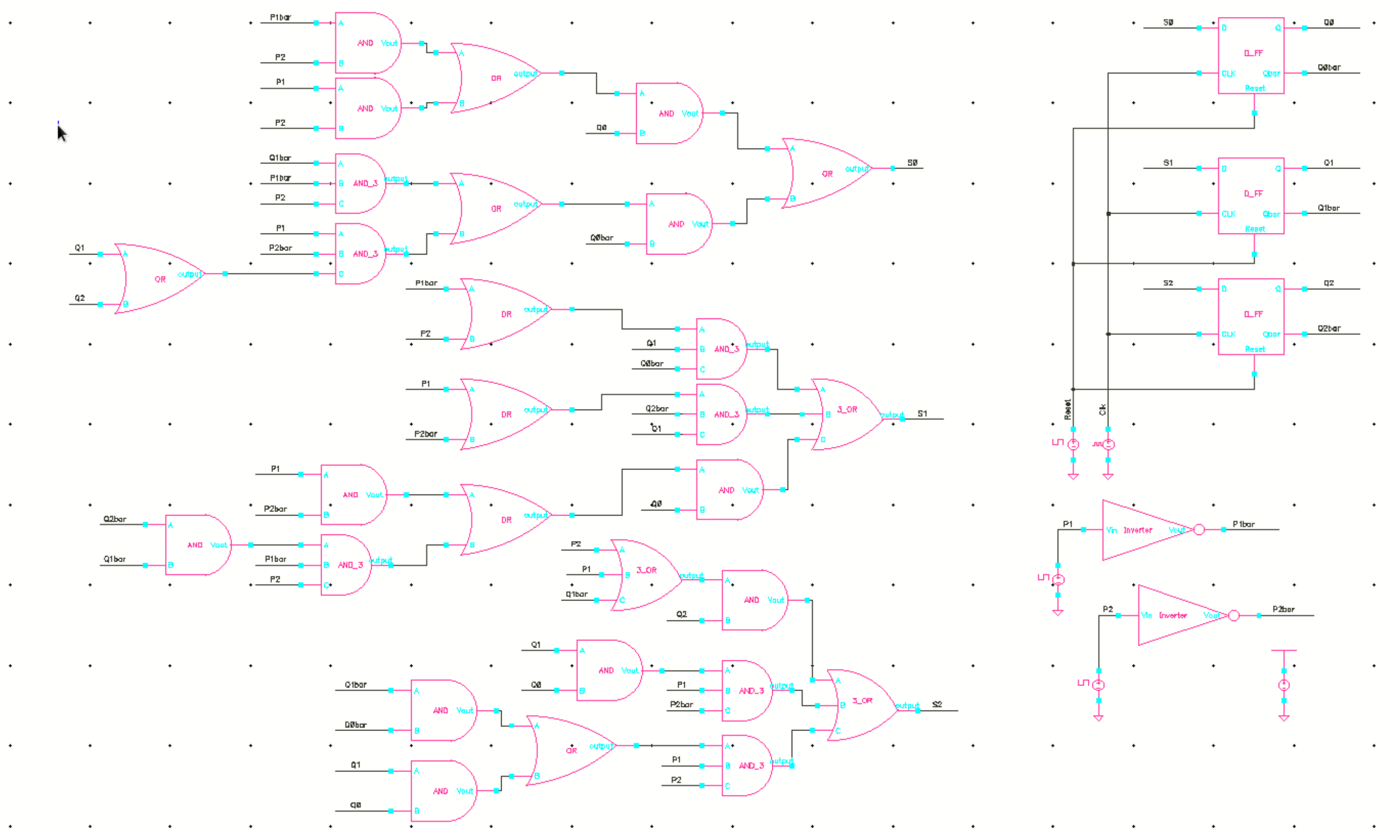
$$Q_1^+ \to S_1^+ = (\overline{P_1} + P_2)\,Q_1 \cdot \overline{Q_0} + (P_1 + \overline{P_2})\,\overline{Q_2}\,Q_1 + Q_0(P_1\overline{P_2} + \overline{Q_2}\,\overline{Q_1}\,\overline{P_1}\,P_2)$$

$$Q_2^+ \to S_2^+ = Q_2(P_2 + P_1 + \overline{Q_1}) + (\overline{Q_1}\,Q_0P_1\overline{P_2} + \overline{P_1}P_2)(\overline{Q_1}\,\overline{Q_0} + Q_1Q_0)$$

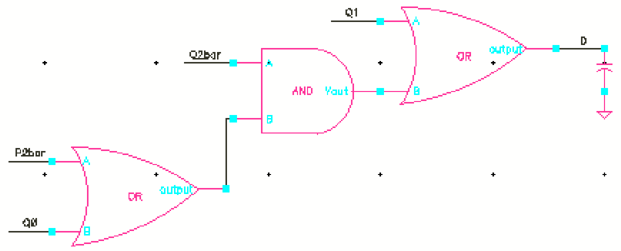
$$D = Q_1 + \overline{Q_2}(\overline{P_2} + Q_0).$$

CS Scanned with CamScanner

Schematic:

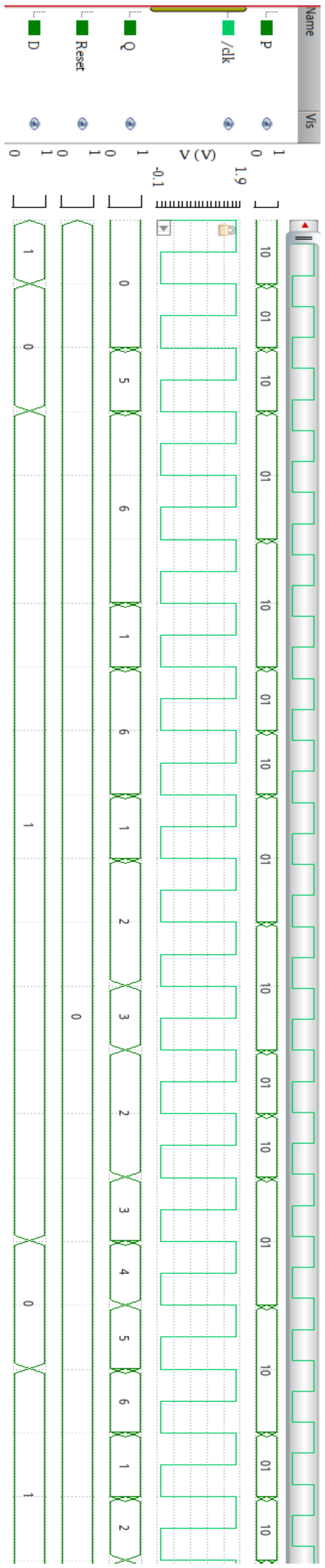


P1bar
P2
P1
P2
AND Vout
OR output
AND Vout
Q0
OR output
S0
Q1bar
P1bar
P2
AND_3
P1
P2bar
Q1
Q2
OR
Q0bar
P1bar
P2
P1
P2bar
Q1
Q0bar
Q2bar
3_OR
S1
Q2bar
Q1bar
P1bar
P2
P2
P1
Q1bar
Q2
Q1
Q0
P1
P2bar
Q1bar
Q0bar
Q1
Q0
P1
P2
S2
S0
S1
S2
D_FF
Q0
Q0bar
Q1
Q1bar
Q2
Q2bar
Reset
Clk
Inverter
P1bar
P2bar



Q1
Q2bar
AND Vout
OR output
D
P2bar
Q0

Results:



Name
Vis
P
/clk
Q
Reset
D
V (V)
1.9
-0.1
P: 10 01 10 01 10 01 10 01 10 01 10 01 10 01 10
Q: 0 5 6 1 6 1 2 3 2 3 4 5 6 1 2
Reset: 0
D: 1 0 1 0 1

Conclusion:

In this lab, we successfully designed a direction control unit for a subway station controller previously designed in lab3. We learnt how to design finite state machines and implement them in Cadence. We successfully verified its working of the design and verified it with the given specs.

Post-lab Questions:

1. Compare and contrast a Moore Finite State Machine (FSM) with a Mealy FSM.

A:

Mealy Machine: A Mealy Machine is an FSM whose output is dependent on the present state as well as the present input.

Moore Machine: Moore machine is an FSM whose output is dependent on only the present state.

Moore FSM:	Mealy FSM:
• In Moore machine, the outputs depend on current state only.	• In Mealy FSM, the outputs depend on current state and inputs.
• There is more hardware requirement.	• There is less hardware requirement.
• They react slower to inputs (One clock cycle later)	• They react faster to inputs.
• Synchronous output and state generation.	• Asynchronous output generation.
• Output is placed on states	• Output is placed on transitions.
• Easy to design	• It is difficult to design.