# *LAB 6*

## *Part I*

2. *Is resetn synchronous or asynchronous?*

*Ans – The resetn signal is a synchronous reset. It has an active low. To reset the FSM to the starting state, the clock has to be set to high whenever we want to reset it.*

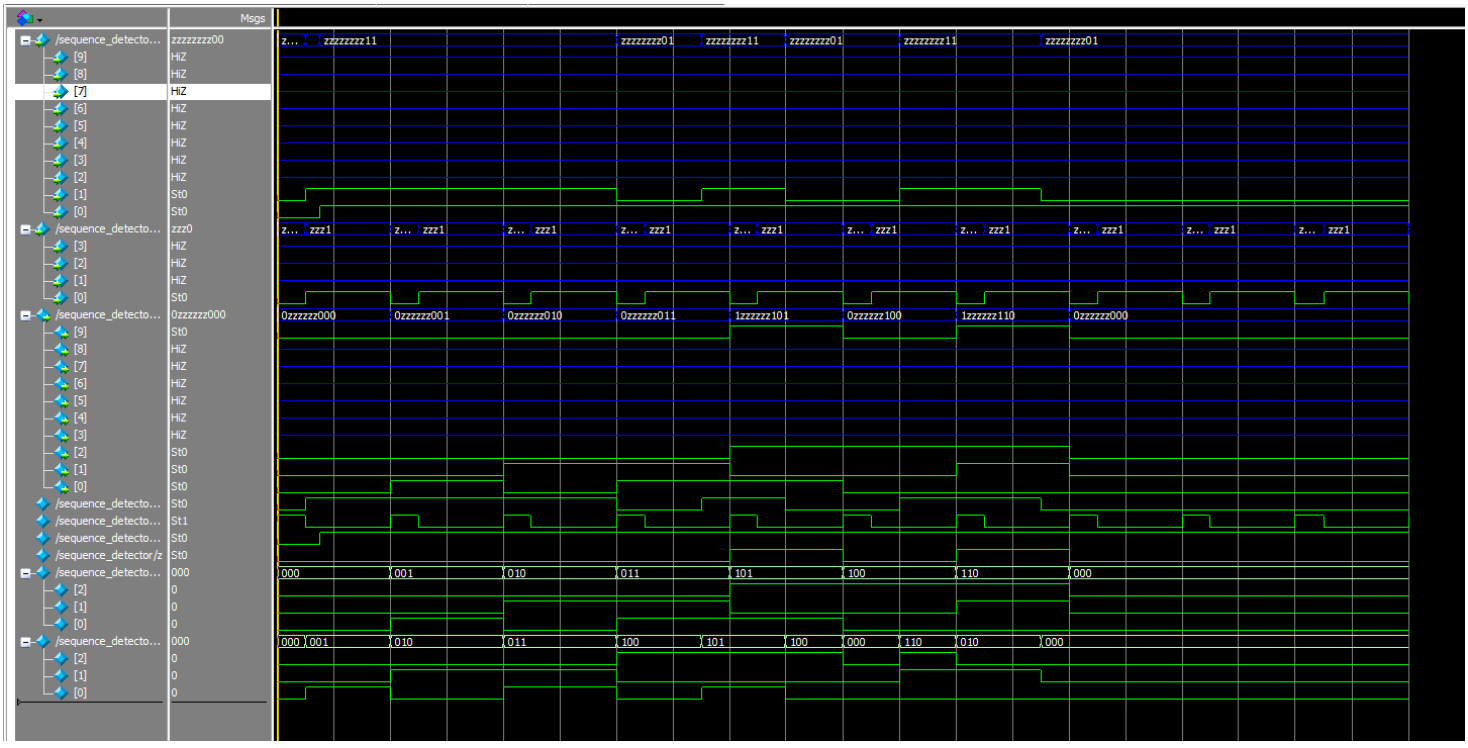
3. *Complete the State Table and Show the Code*

*Ans – For the completed code, please refer to sequence_detector.v*

| Current State | Switch Input | Output State |
|---|---|---|
| 000 | 0 | 000 |
| 000 | 1 | 001 |
| 001 | 0 | 000 |
| 001 | 1 | 010 |
| 010 | 0 | 100 |
| 010 | 1 | 011 |
| 011 | 0 | 110 |
| 011 | 1 | 101 |
| 100 | 0 | 000 |
| 100 | 1 | 110 |
| 101 | 0 | 100 |
| 101 | 1 | 101 |
| 110 | 0 | 000 |
| 110 | 1 | 010 |

*Note: One's marked in yellow output the result (ie. 1)*

4. *Simulate the circuit and post screenshots –*

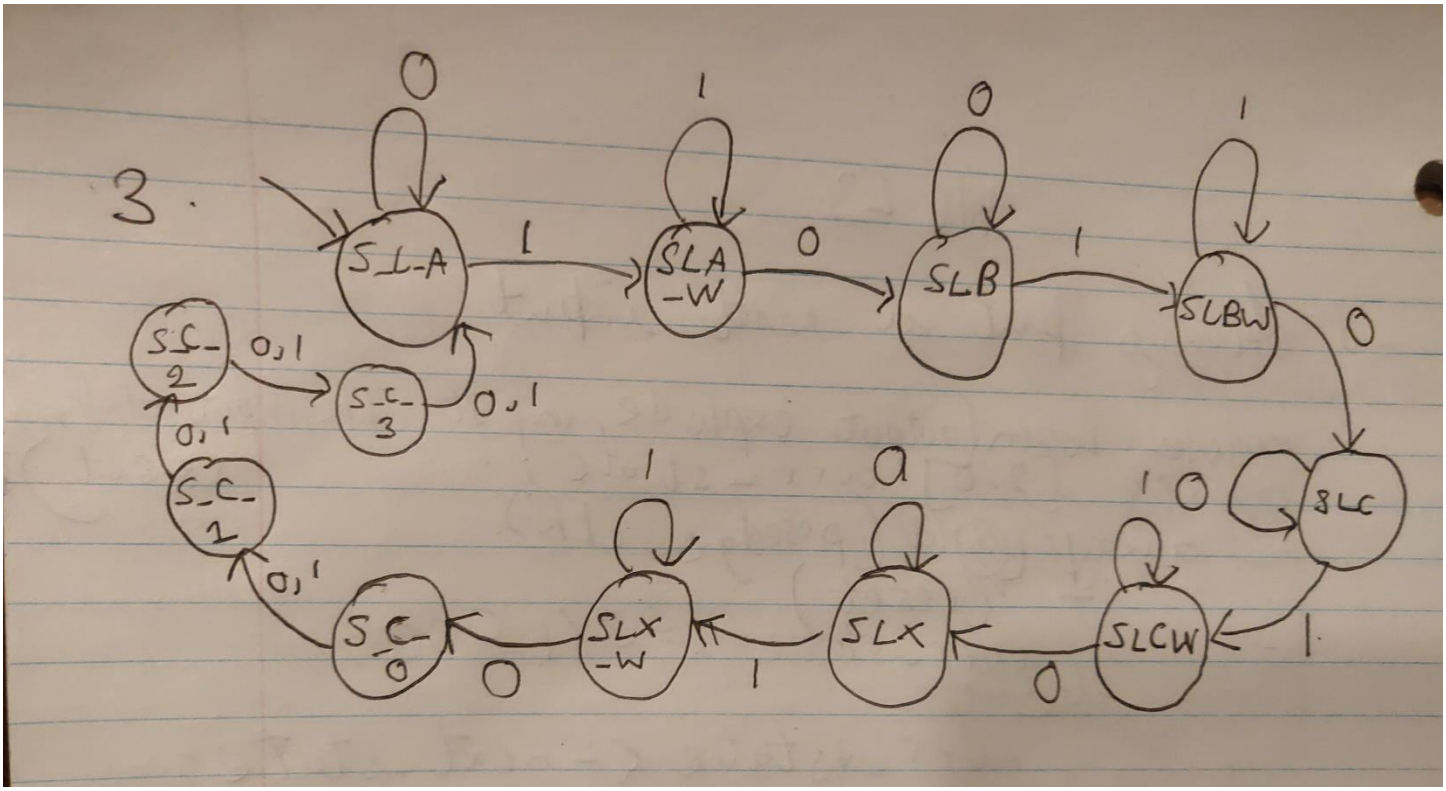


# *Part II*

2. *Draw a table that shows the state of the Registers and control signals for each cycle of your computation*

| Cycle | Value | Alu_select_a | Alu_select_b | B | R |
|---|---|---|---|---|---|
| 0 | A*X | 00 | 11 | B | - |
| 1 | A*X+B | 00 | 01 | A*X+B | - |
| 2 | X*(A*X+B) | 11 | 01 | X*(A*X+B) | - |
| 3 | X*(A*X+B)+C | 01 | 10 | X*(A*X+B) | X*(A*X+B) + C |

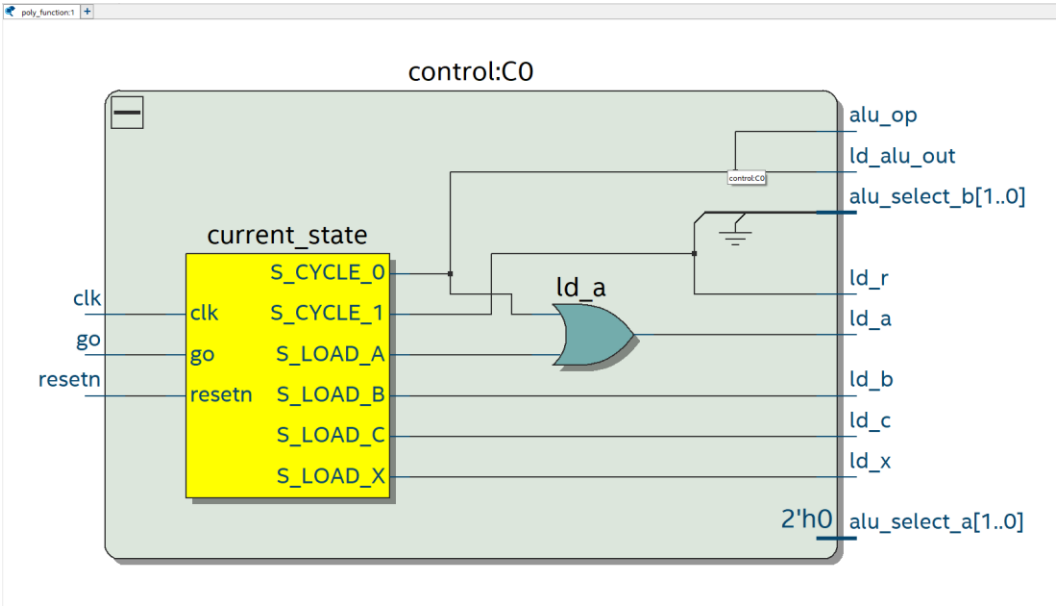
3. *Draw a state diagram for your controller starting with the register load states provided in the example FSM*

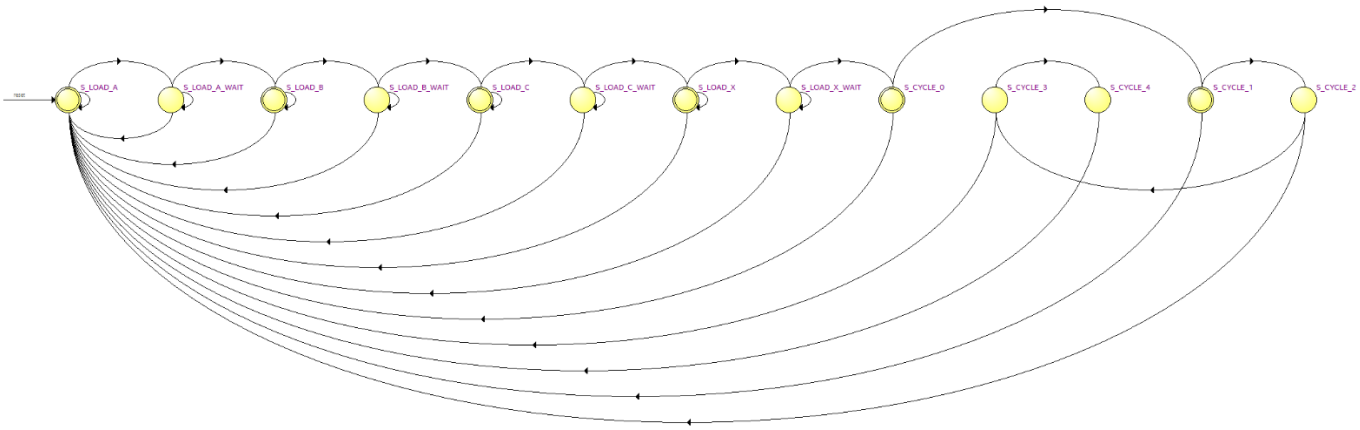



4. *Modify the provided FSM code to implement your controller and synthesize it. You should only modify the control module.*

*Ans – Please check poly_function.v*

5. *Include a screenshot of the generated FSM in your prelab.*



State Machine - |poly_function|part2:u0|control:C0|current_state

<<Filter>>

| | Name | state.S_C | state.S_( | state.S_C | tate.S_ | ate.S_ | :S_LC | tate.S_ | te.S_LO/ | state.S_ | e.S_LO. | current_state.S_LOAD_B | current_state.S_LOAD_A_WAIT | current_state.S_LOAD_A |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | current_state.S_LOAD_A | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | current_state.S_LOAD_A_WAIT | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 3 | current_state.S_LOAD_B | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 4 | current_state.S_LOAD_B_WAIT | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 5 | current_state.S_LOAD_C | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 6 | current_state.S_LOAD_C_WAIT | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 7 | current_state.S_LOAD_X | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 8 | current_state.S_LOAD_X_WAIT | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 9 | current_state.S_CYCLE_1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 10 | current_state.S_CYCLE_2 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 11 | current_state.S_CYCLE_3 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 12 | current_state.S_CYCLE_4 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 13 | current_state.S_CYCLE_0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

6. *Simulate the circuit and post screenshots –*