Prelab 6

Part I

2. Answer the following questions in your prelab: given the starter code, is the `resetn` signal is an synchronous or asynchronous reset? Is it active high, or active low? Given this, what do you have to do in simulation to reset the FSM to the starting state? **(PRELAB)**

1) It's a synchronous reset

2) It's active low

3) Make reset_n (SW[0]) = 0 , then create a posedge of clock (KEY[0])

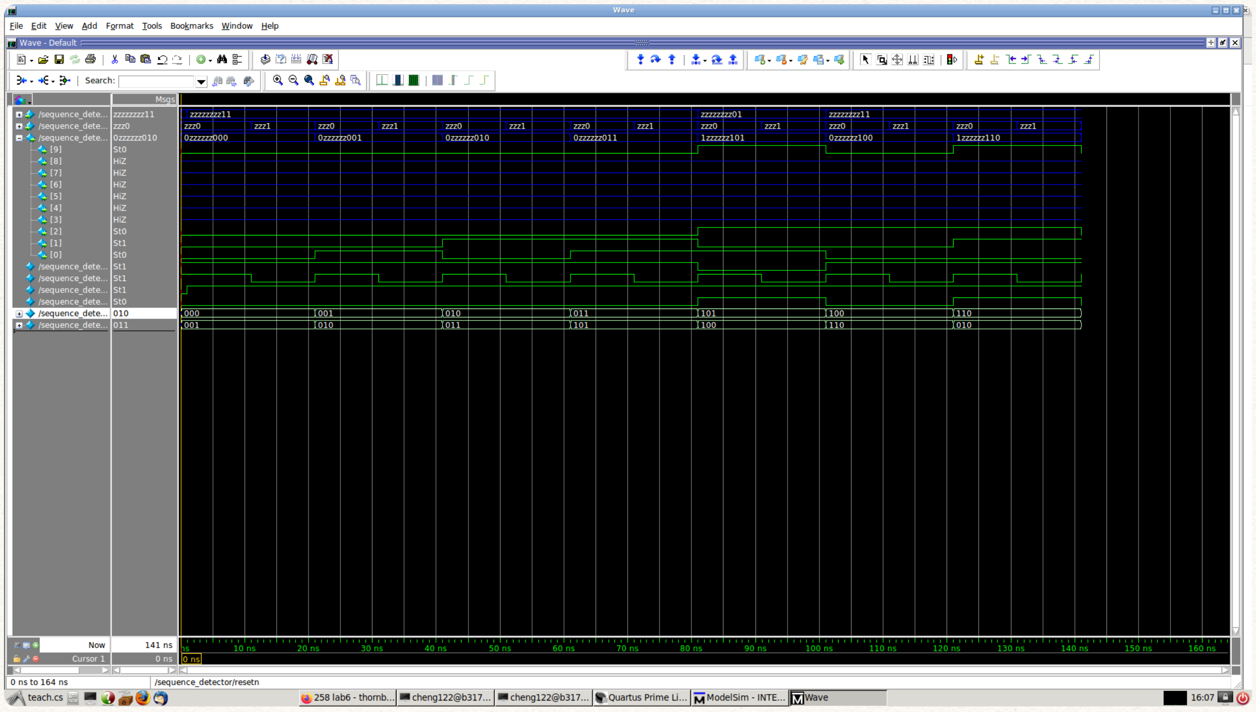
3. Complete the state table showing how the present state and input value determine the next state and the output value. Fill in all the missing parts of the template code to implement the FSM based on the state table you derived. Include completed code in your prelab. **(PRELAB)**

Current State	Input (w)	Next State
A	0	A
A	1	B
B	0	A
B	1	C
C	0	E
C	1	D
D	0	E
D	1	F
E	0	A
E	1	G
F	0	E
F	1	F
G	0	A
G	1	C

see sequence_detector.v

for completed Verilog code.

4. Simulate your circuit using ModelSim for a variety of input settings, ensuring the output waveforms are correct. Include a few screenshots that shows the simulation output. **(PRELAB)**



```
/sequence_dete...   zzzzzzzz11
/sequence_dete...   zzz0
/sequence_dete...   0zzzzzz010
    [9]             St0
    [8]             HiZ
    [7]             HiZ
    [6]             HiZ
    [5]             HiZ
    [4]             HiZ
    [3]             HiZ
    [2]             St0
    [1]             St1
    [0]             St0
/sequence_dete...   St1
/sequence_dete...   St1
/sequence_dete...   St1
/sequence_dete...   St0
/sequence_dete...   010
/sequence_dete...   011

Now        141 ns
Cursor 1   0 ns
0 ns to 164 ns    /sequence_detector/resetn
```

Part II

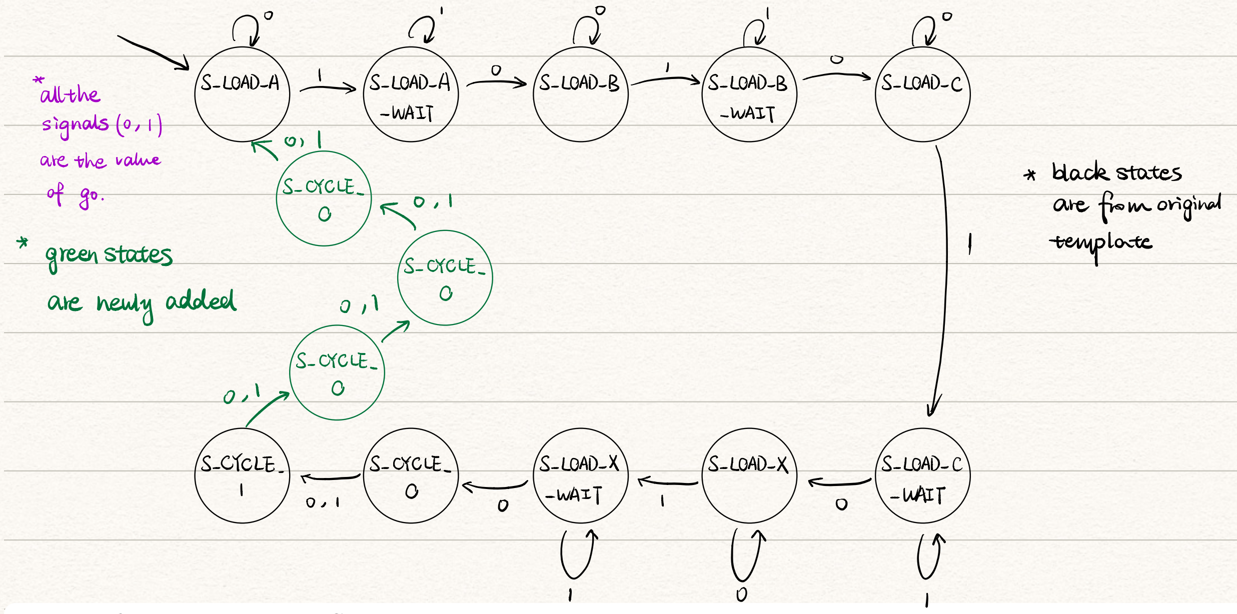
1. Examine the provided starter Verilog code for this part (`poly_function.v`, which is available on Quercus, as well as in the appendices of this handout). This is a major step in this part of the lab. You will not need to write much Verilog yourself, but you will need to fully understand the circuitry described by the provided Verilog to be able to make your modifications. **(PRELAB)**

I've read and understand the circuitry.

2. Determine a sequence of steps similar to the datapath example shown in lecture that controls your datapath to perform the required computation. You should draw a table that shows the state of the Registers and control signals for each cycle of your computation. Include this table in your prelab. **(PRELAB)**

Cycle	Computation	alu-op	ld-a	ld-b	ld-r	ld-alu-out	alu-select-a	alu-select-b	A	B	R
0	Bx	1	0	1	0	1	01	11	B	x	/
1	Bx + A	0	0	1	0	1	01	00	Bx	A	/
2	Cx	1	1	0	0	1	10	11	C	x	/
3	Cx^2	1	1	0	0	1	00	11	Cx	x	/
4	Cx^2+Bx+A	0	0	0	1	0	01	00	Cx^2	Bx+A	Cx^2+Bx+A

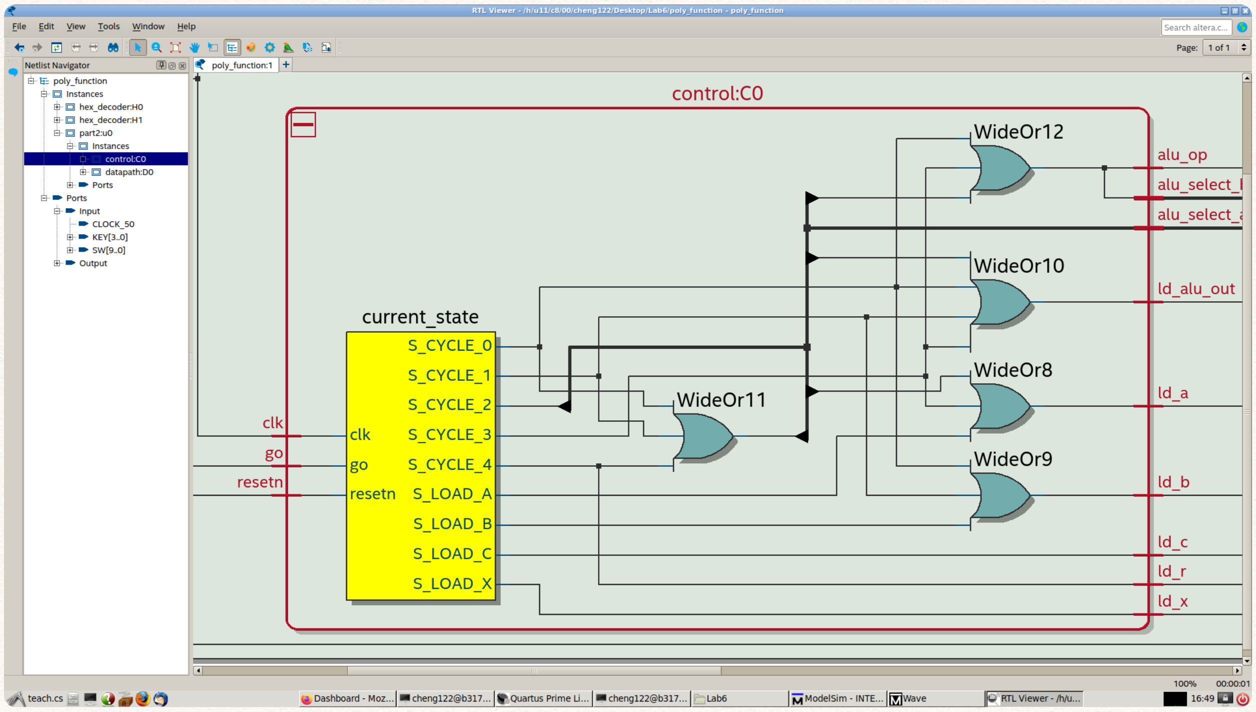
3. Draw a state diagram for your controller starting with the register load states provided in the example FSM. Include the state diagram in your prelab. **(PRELAB)**

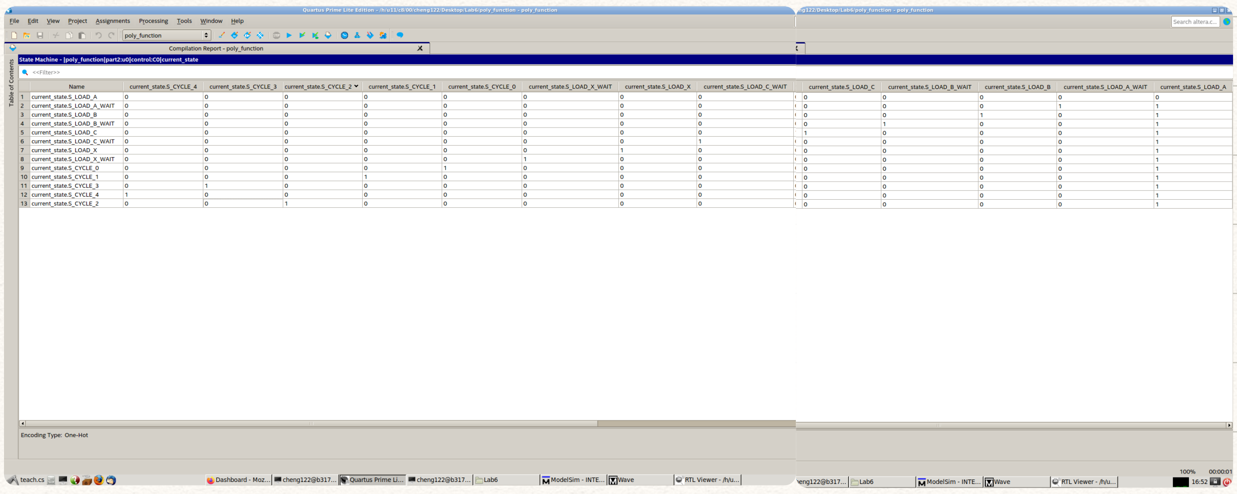

S-LOAD-A
S-LOAD-A -WAIT
S-LOAD-B
S-LOAD-B -WAIT
S-LOAD-C
S-LOAD-C -WAIT
S-LOAD-X
S-LOAD-X -WAIT
S-CYCLE_ 0
S-CYCLE_ 1
0
1
0, 1
* all the signals (0, 1) are the value of go.
* green states are newly added
* black states are from original template

4. Modify the provided FSM code to implement your controller and synthesize it. You should only modify the control module. Include your modified code in the prelab. **(PRELAB)**

see poly-function.v

5. To examine the circuit produced by Quartus Prime open the RTL Viewer tool (Tools > Netlist Viewers > RTL Viewer). Find (on the left panel) and double-click on the box shown in the circuit that represents the finite state machine, and determine whether the state diagram that it shows properly corresponds to the one you have drawn. To see the state codes used for your FSM, open the Compilation Report, select the Analysis and Synthesis section of the report, and click on State Machines. Include a screenshot of the generated FSM in your prelab. **(PRELAB)**



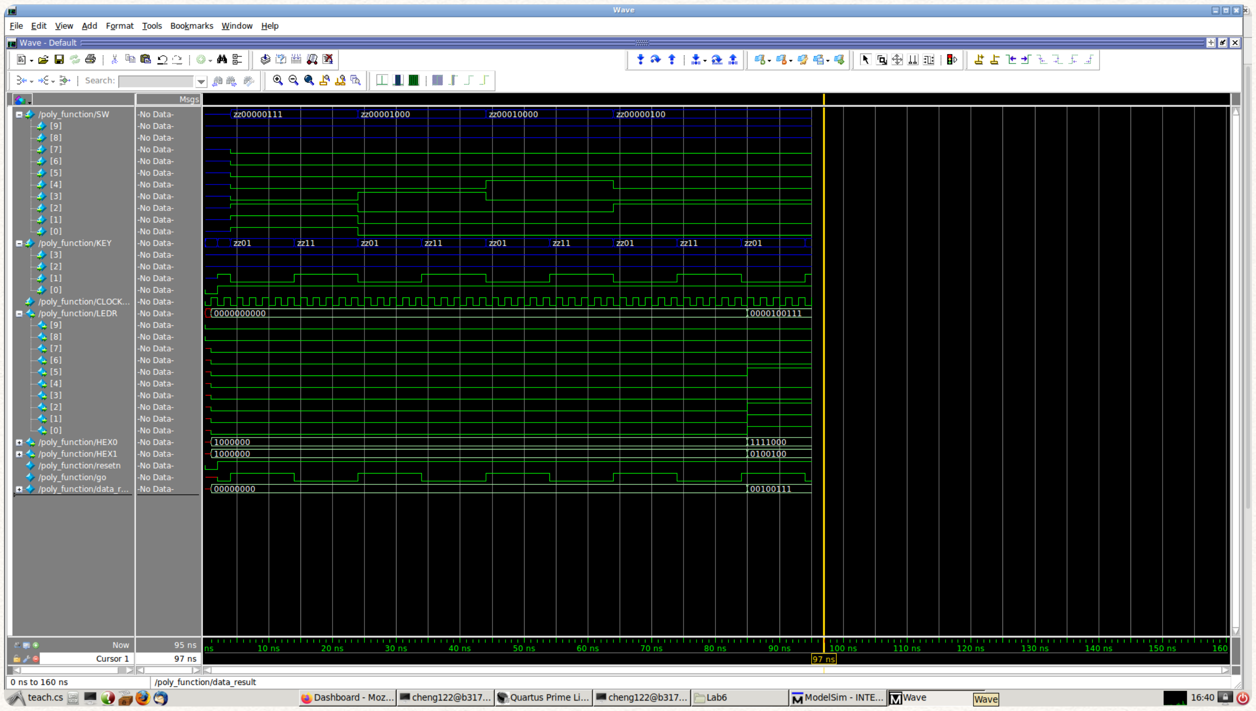

Quartus Prime Lite Edition - /h/u11/c8/00/cheng122/Desktop/Lab6/poly_function - poly_function
File
Edit
View
Project
Assignments
Processing
Tools
Window
Help
poly_function
Compilation Report - poly_function
Table of Contents
<<Filter>>

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

	Name	current_state.S_CYCLE_4	current_state.S_CYCLE_3	current_state.S_CYCLE_2	current_state.S_CYCLE_1	current_state.S_CYCLE_0	current_state.S_LOAD_X_WAIT	current_state.S_LOAD_X	current_state.S_LOAD_C_WAIT	current_state.S_LOAD_C	current_state.S_LOAD_B_WAIT	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_0	0	0	0	0	1	0	0	0	0	0	0	0	1
10	current_state.S_CYCLE_1	0	0	0	1	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_2	0	0	1	0	0	0	0	0	0	0	0	0	1

Encoding Type: One-Hot

6. Simulate your circuit with ModelSim for a variety of input settings, ensuring the output waveforms are correct. It is recommended that you start by simulating the datapath and controller modules separately. Only when you are satisfied that they are working individually should you combine them into the full design. Why is this approach better? (Hint: Consider the case when your design has 20 different modules.) Include few screenshots of simulation output in your prelab. **(PRELAB)**



Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
/poly_function/SW
/poly_function/KEY
/poly_function/CLOCK...
/poly_function/LEDR
/poly_function/HEX0
/poly_function/HEX1
/poly_function/resetn
/poly_function/go
/poly_function/data_r...
-No Data-
zz00000111
zz00001000
zz00010000
zz00000100
zz01
zz11
0000000000
0000100111
1000000
1111000
0100100
00000000
00100111
Now 95 ns
Cursor 1 97 ns
0 ns to 160 ns
/poly_function/data_result