CSC258 Lab6 Report

Tianle Wang

Part I

1. Begin with the starter circuit provided in `lab6_starter.circ`. Answer the following questions in your prelab:
   - Given the starter circuit, is the **Reset** signal a synchronous or asynchronous reset?
     - Since the **Reset** signal does not depend on clock, it's a asynchronous reset
   - Is it active high, or active low signal?
     - Since the **Enable** signal need to be `1` to active whole circuit, it's active high signal.
   - How should the **Reset** signal feature in the tests that you run on your FSM?
     - To clean the state to the default
   - Hint: if you're not sure of some of the answers to the first two questions, try experimenting with the circuit to confirm the behaviour you suspect.
2. Before modifying the Logisim starter circuit, assign flip-flop values to each of the states in Figure 2 and create a state table that illustrates the state transitions in response to the input signal w.

w	curr_state	next_state
0	000	000
1	000	001
0	001	000
1	001	010
0	010	100
1	010	011
0	011	100
1	011	101
0	100	000
1	100	110
0	101	100
1	101	101
0	110	000
1	110	010

According to the table above, we can conclude a K-Map following:

z_2

-	$c_2'c_1'$	$c_2'c_1$	c_2c_1	c_2c_1'
$w'c_0'$	0	1	0	0
$w'c_0$	0	1	0	1
wc_0	0	1	0	1
wc_0'	0	0	0	1

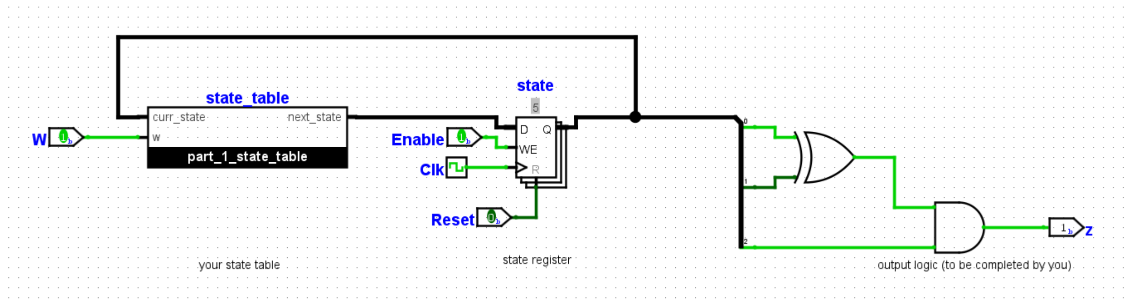
z_1

-	$c_2'c_1'$	$c_2'c_1$	c_2c_1	c_2c_1'
$w'c_0'$	0	0	0	0
$w'c_0$	0	0	0	0
wc_0	1	0	0	0
wc_0'	0	1	1	1

z_0

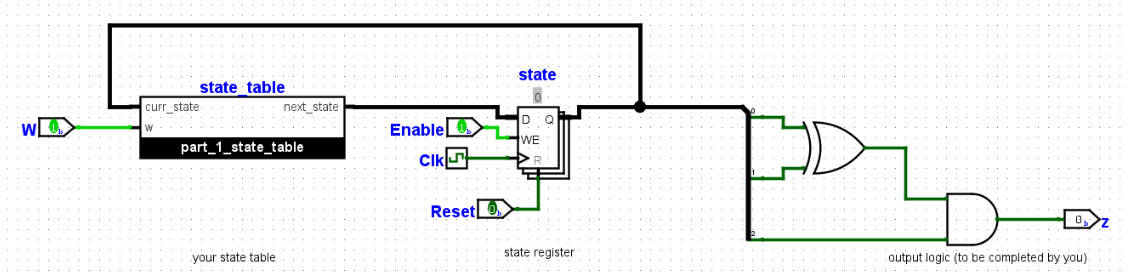
-	$c_2'c_1'$	$c_2'c_1$	c_2c_1	c_2c_1'
$w'c_0'$	0	0	0	0
$w'c_0$	0	0	0	0
wc_0	0	1	0	1
wc_0'	1	1	0	0

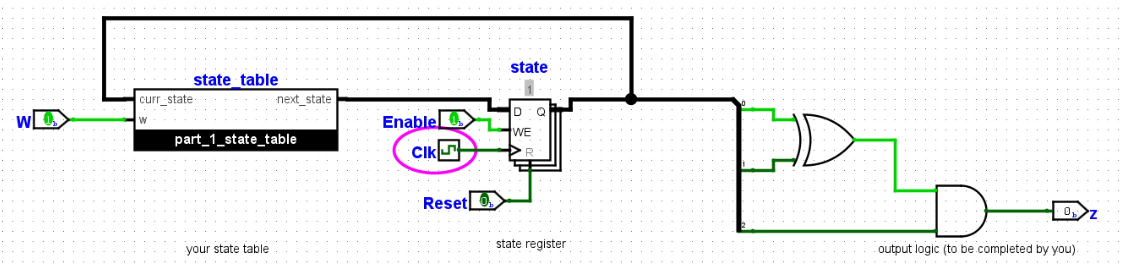
3. Fill in the rest of the circuit in the part1 state table module to implement the state table you derived in Step 2. This module will implement the state logic (the combinational circuit that determines what the new flip-flop values should be, based on the previous flip-flop values and the input w).
4. Implement the output value circuit for z in part1 FSM.

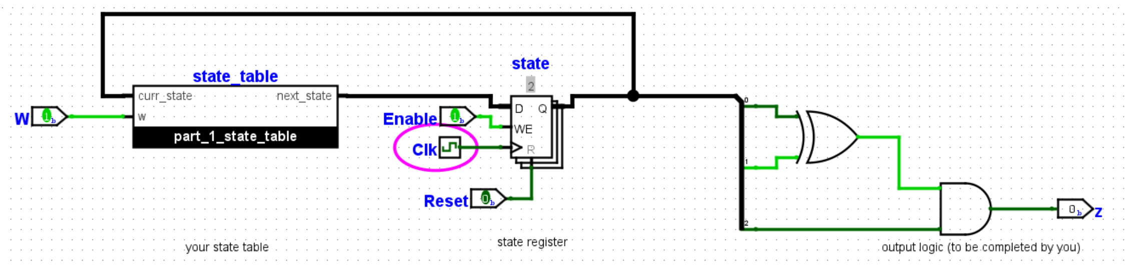


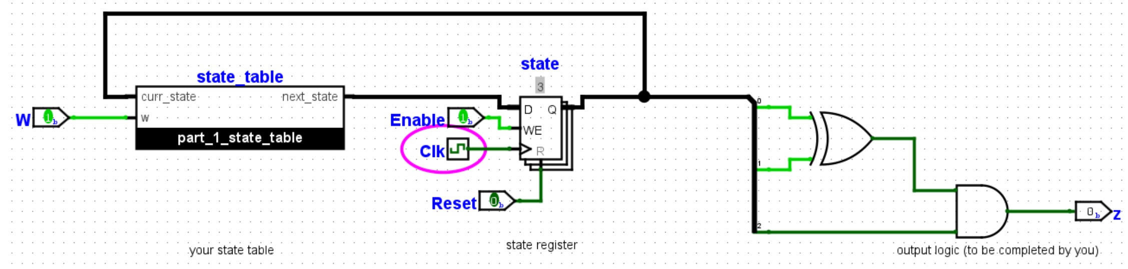
5. Outline the test plan for your circuit in your prelab report and why these test cases verify the correctness of your circuit. Use this plan to test your modules with Poke( ) to confirm its expected behaviour. Include screenshots of your simulation output that illustrates key test cases.

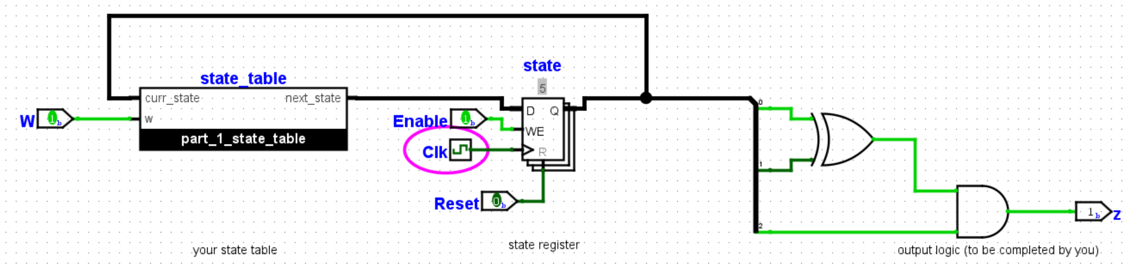
Since the requirement of `z` is that clock sequence `1111` or `1101` , therefore I start from all zero, and 4 clock cycle with `w = 1111` and then restart and go 4 clock with `1101`

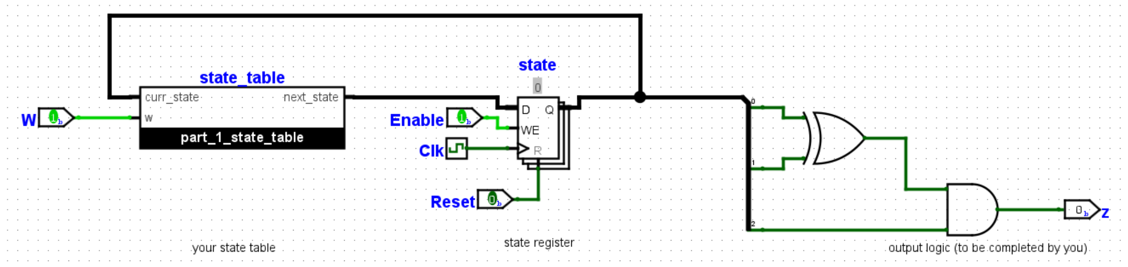


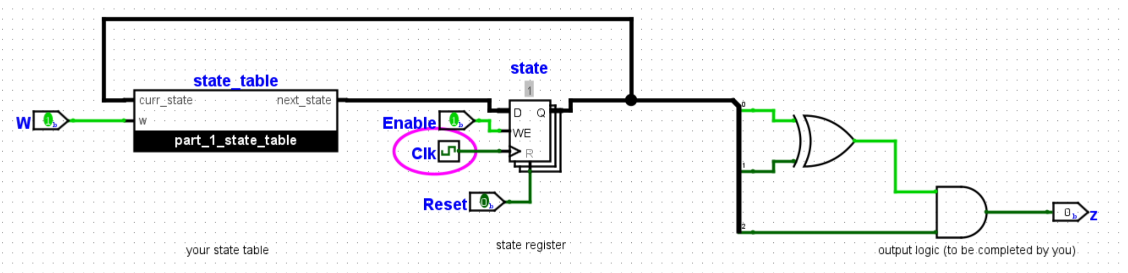


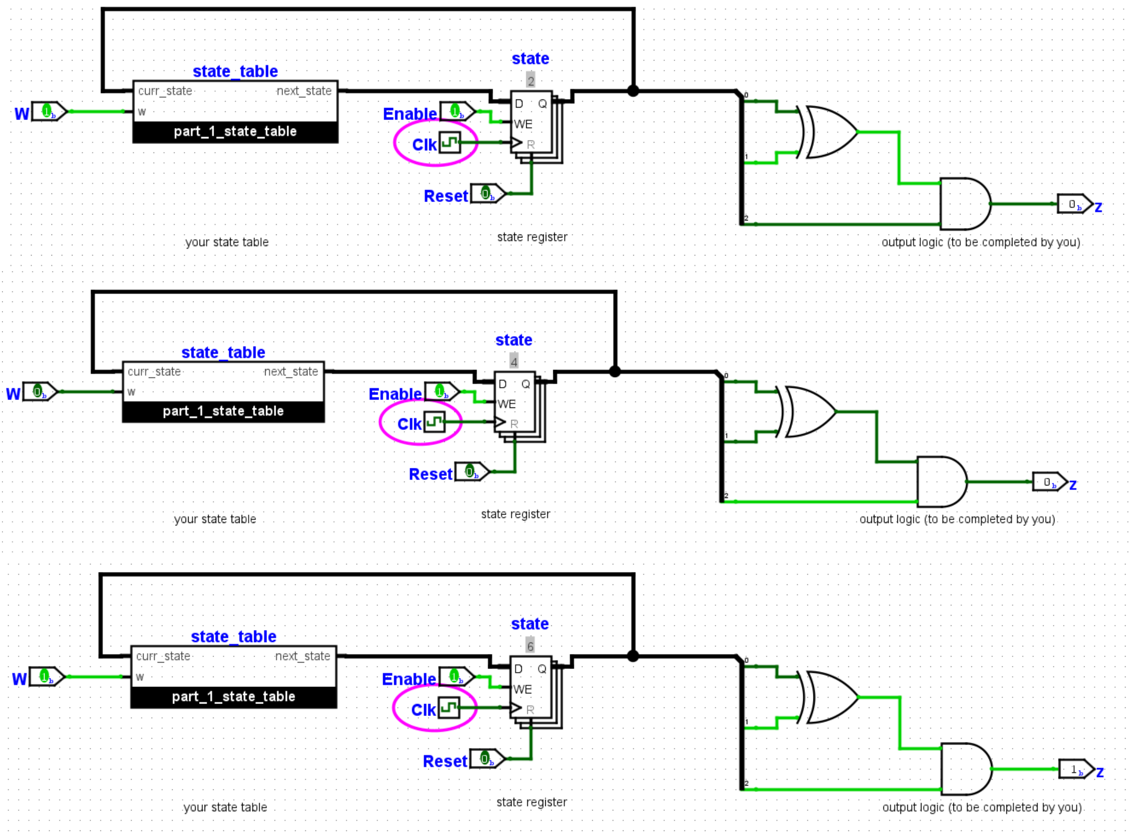










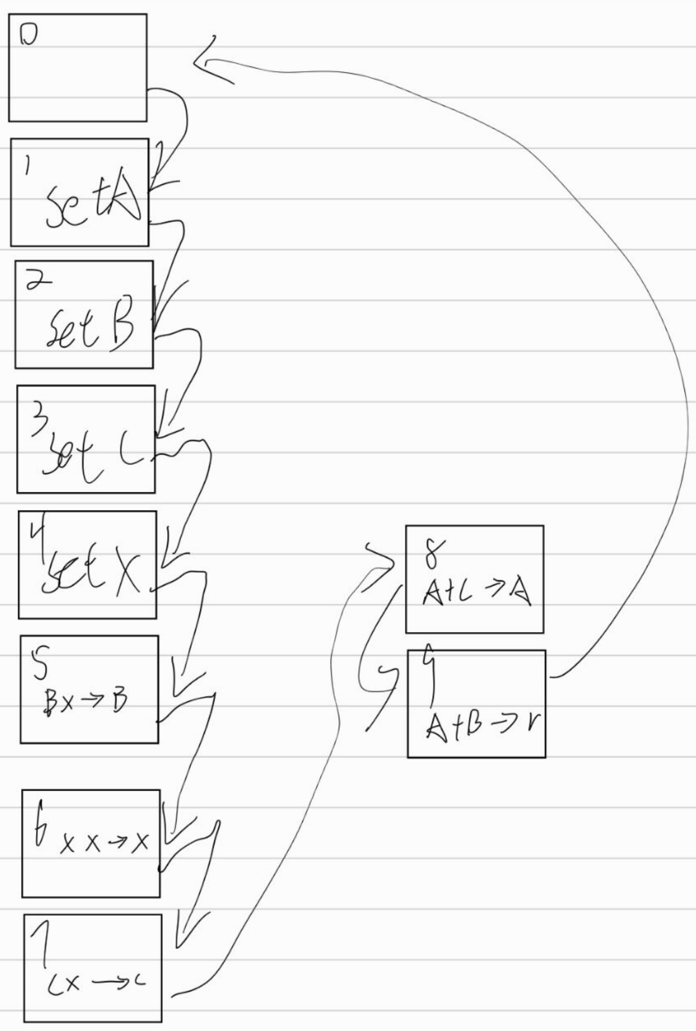


Part II

1. Examine the provided starter circuits (all the modules for this part start with `part_2`), which is available on Quercus). This is a major step in this part of the lab. You will not need to build the datapath yourself, but you will need to fully understand the datapath embodied by the starter circuit to be able to make your modifications.
2. Determine a sequence of steps similar to the datapath example shown in lecture to control the datapath to perform the required computation. You should draw a table that shows the contents of the registers and the control signal values for each cycle of your computation. Include this table in your prelab.

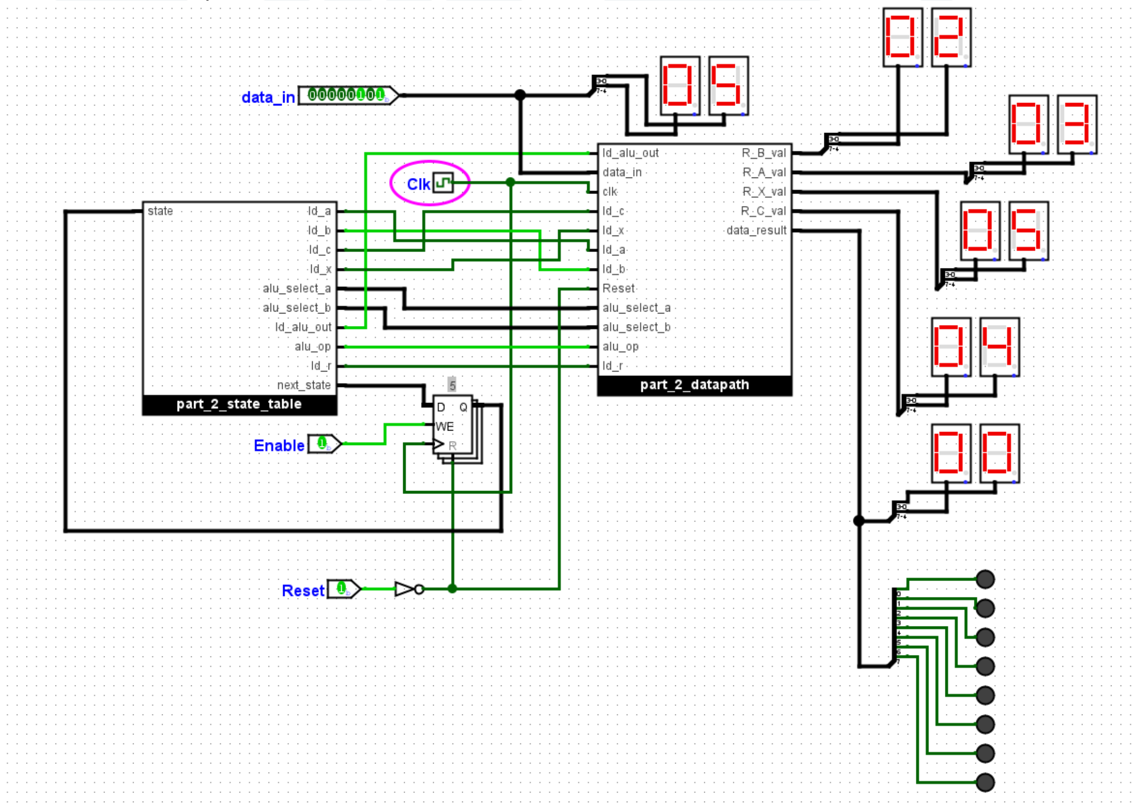
curr_state	next_state	Feature
0000	0001	Leave empty, keep everything no change
0001	0010	`id_a = 1` set `data_in` into `A`
0010	0011	`id_b = 1`set `data_in` into `B`
0011	0100	`id_c = 1`set `data_in` into `C`
0100	0101	`id_x = 1`set `data_in` into `X`
0101	0110	calculate `BX` and store the result into `B`
0110	0111	calculate `X^2` and store the result into `X`
0111	1000	calculate `CX` and store the result into `C`
1000	1001	calculate `C + A` and store the result into `A`
1001	0000	`Id_r = 1` open D flip-flop and write result in `data_result`

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.

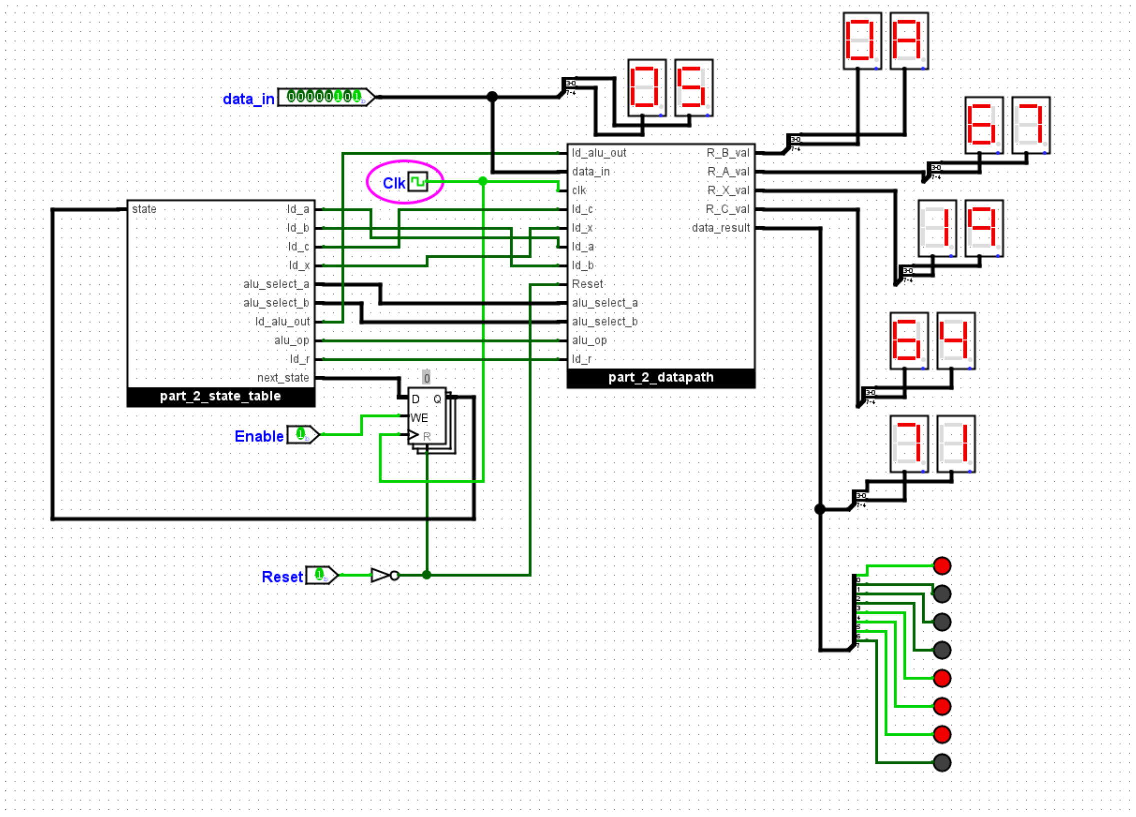


4. Modify the provided FSM to implement your controller and synthesize it. You should only modify the control module, not the datapath. Submit your modified circuit in the prelab.
5. Test your modules with Poke( ) to verify its correctness. Include a few screenshots that shows the simulation output.

For `cx^2+bx+a`, pick `a=3` `b=2` `c=4` `x = 5`, then `cx^2+bx+a = 113` in HEX `71`



data_in
00000101
Clk
Enable
Reset
state
ld_a
ld_b
ld_c
ld_x
alu_select_a
alu_select_b
ld_alu_out
alu_op
ld_r
next_state
part_2_state_table
D Q
WE
R
ld_alu_out
data_in
clk
ld_c
ld_x
ld_a
ld_b
Reset
alu_select_a
alu_select_b
alu_op
ld_r
R_B_val
R_A_val
R_X_val
R_C_val
data_result
part_2_datapath
05
02
03
05
04
00



data_in
00000101
Clk
Enable
Reset
state
ld_a
ld_b
ld_c
ld_x
alu_select_a
alu_select_b
ld_alu_out
alu_op
ld_r
next_state
part_2_state_table
D Q
WE
R
ld_alu_out
data_in
clk
ld_c
ld_x
ld_a
ld_b
Reset
alu_select_a
alu_select_b
alu_op
ld_r
R_B_val
R_A_val
R_X_val
R_C_val
data_result
part_2_datapath
05
0A
67
19
64
71