Finite State Machines

Shi Ran 1004793495

October 29, 2020

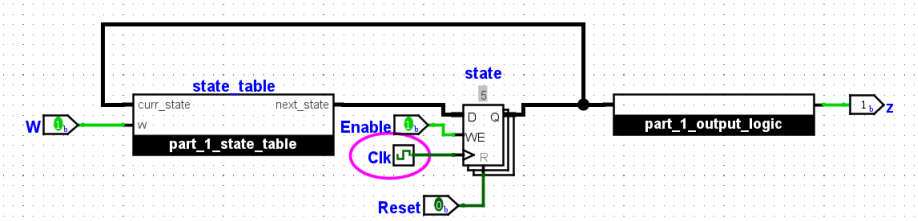
Part I

1. *Reset* signal is synchronous; it is active low signal; it will "clean" the current sequence, that is, making *curr_state* 0.

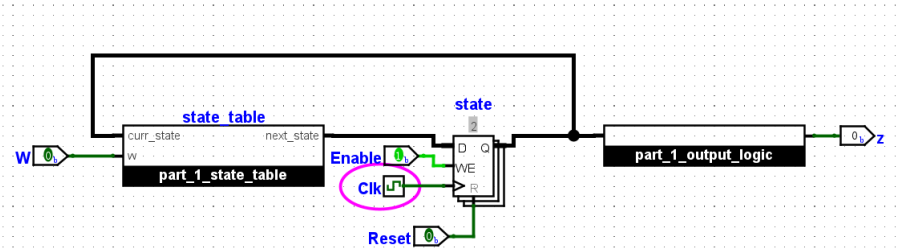
2.

curr	w	next	z
A: 000	0	A: 000	0
A: 000	1	B: 110	0
B: 110	0	A: 000	0
B: 110	1	C: 011	0
C: 011	0	E: 010	0
C: 011	1	D: 111	0
D: 111	0	E: 010	0
D: 111	1	F: 101	1
E: 010	0	A: 000	0
E: 010	1	G: 001	1
F: 101	0	E: 010	0
F: 101	1	F: 101	1
G: 001	0	A: 000	0
G: 001	1	C: 011	0

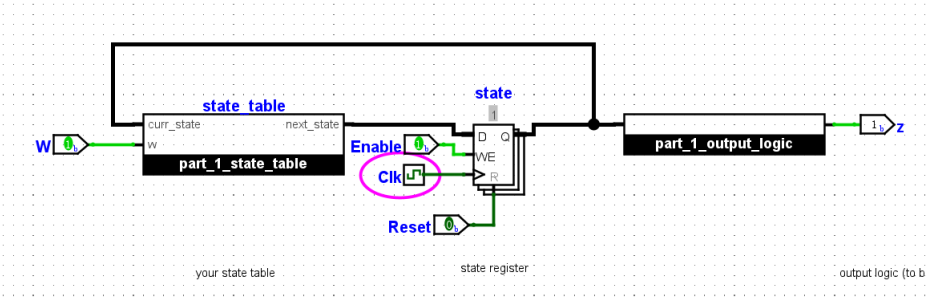
5. With $w = 1$, after the clock ticks for more than 4 times, the output remains high and register has a state 5 (F respectively).



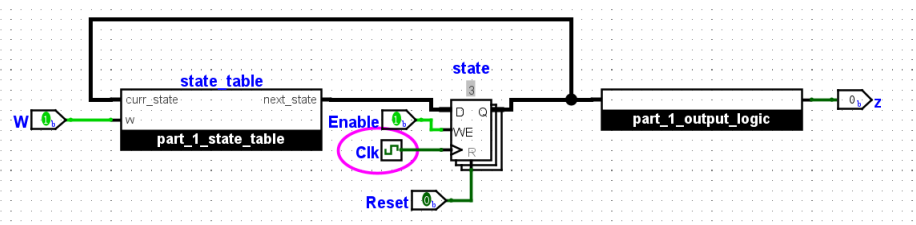
Continue with the previous case, set $w = 0$, after the clock ticks, the output becomes low and register has a state 2 (E respectively).



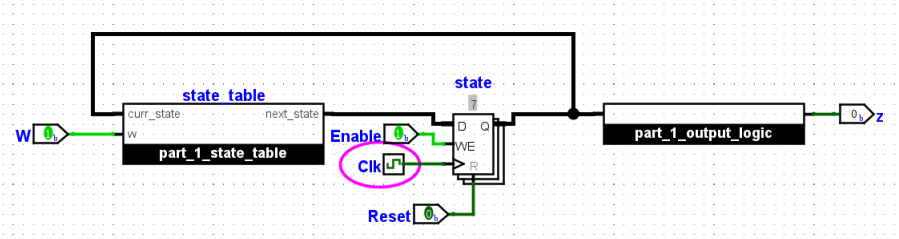
Continue with the previous case, set $w = 1$, after the clock ticks, the output becomes high and register has a state 1 (G respectively).



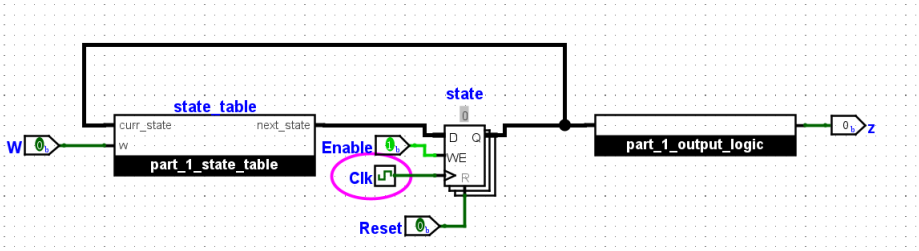
Continue with the previous case, set $w = 1$, after the clock ticks, the output becomes low and register has a state 3 (C respectively).



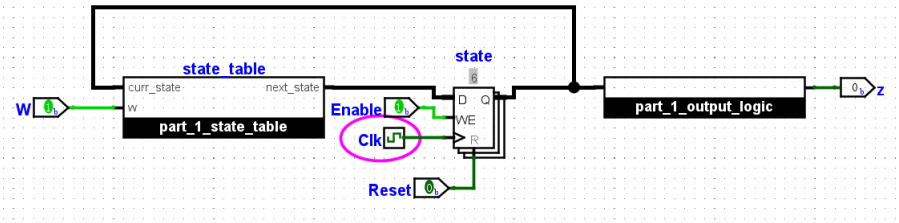
Continue with the previous case, set $w = 1$, after the clock ticks, the output becomes low and register has a state 7 (D respectively).



Continue with the previous case, set $w = 0$, after the clock ticks 2 times, the output becomes low and register has a state 0 (A respectively).



Continue with the previous case, set $w = 0$, after the clock ticks, the output becomes low and register has a state 6 (B respectively).



Part II

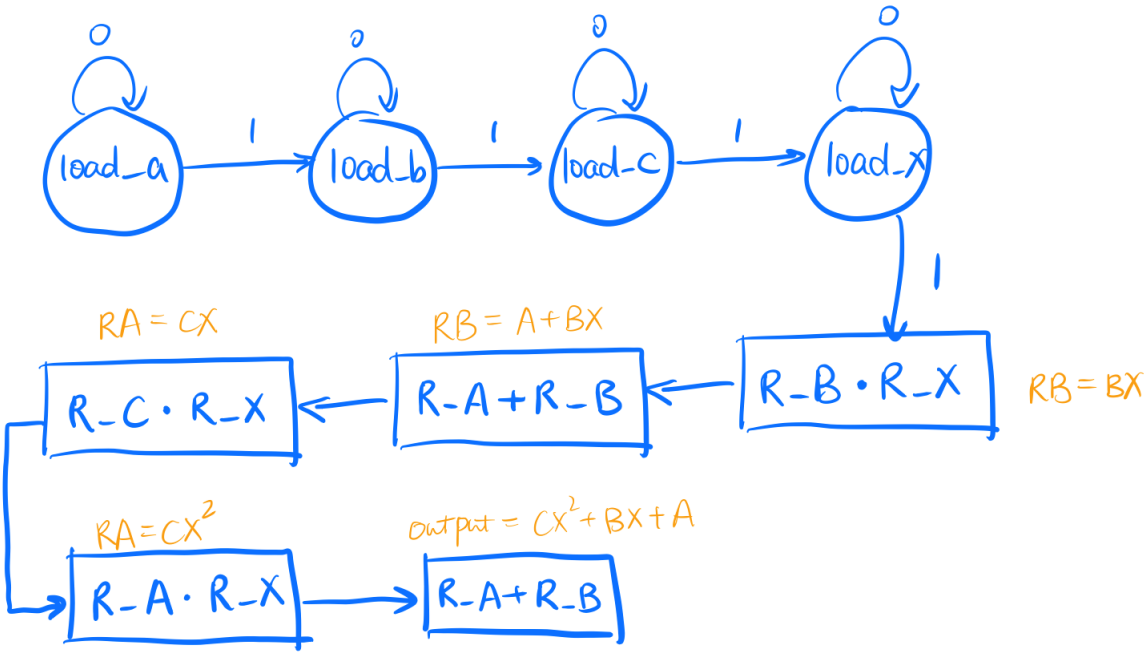
1. *data_in* is the data input by user, if WE for the register is high, the input value will be loaded into that register. Then two mux *alu_select_a* and *alu_select_b* determines A and B values that get input into the ALU. The alu_op button determines whether the alu will perform addition or multiplication. The output of ALU unit can be input again into the registers when Id_alu_out is on to perform furthers steps of calculation. The output can also be loaded into register R_R when ld_r is on. The value will be the output. The reset buttons will make all values in registers to be 0.

The FSM: when *Reset* is on, *Enable* is on, there are 5 states in total to perform the calculation. First, input value is loaded into R_C, after the clock ticks, the input value is loaded into R_A, then multiply R_A by R_A and store the value in R_A. Then add R_C to R_A, store the value in R_B. Then add R_A to R_B again, this value gets output.

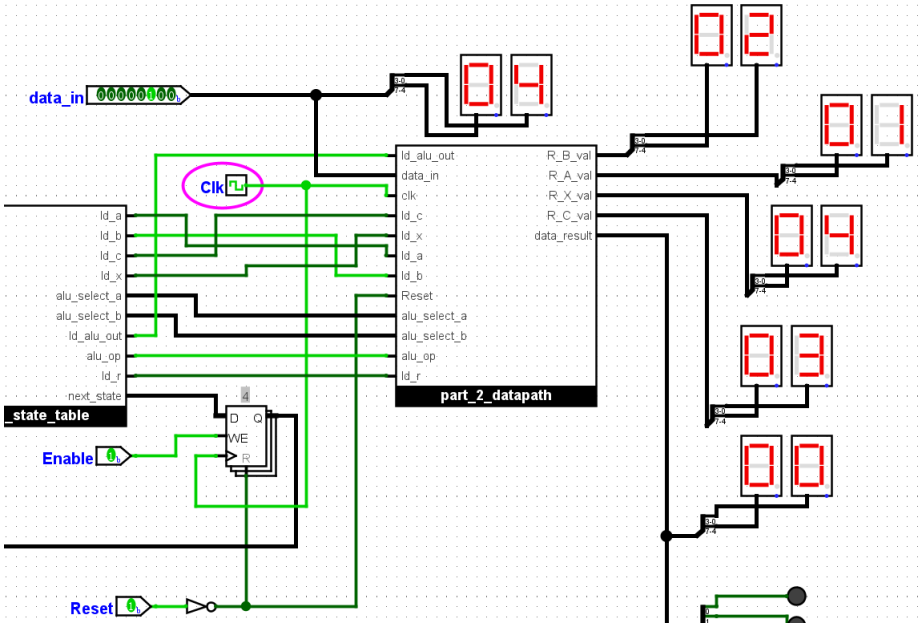
2.

Steps	Id_c	Id_x	Id_a	Id_b	alu_out	sa	sb	op	ld_r
Load value into R_A	0	0	1	0	0	00	00	0	0
Load value into R_B	0	0	0	1	0	00	00	0	0
Load value into R_C	1	0	0	0	0	00	00	0	0
Load value into R_X	0	1	0	0	0	00	00	0	0
Multiply R_B by R_X, store result in R_B	0	0	0	1	1	01	11	1	0
Add R_A to R_B, store result in R_B	0	0	0	1	1	00	01	0	0
Multiply R_C by R_X, store value in R_A	0	0	1	0	1	10	11	1	0
Multiply R_A by R_X, store value in R_A	0	0	1	0	1	00	11	1	0
Add R_A to R_B, ALU output is $Cx^2 + Bx + A$	0	0	0	0	1	00	01	0	1

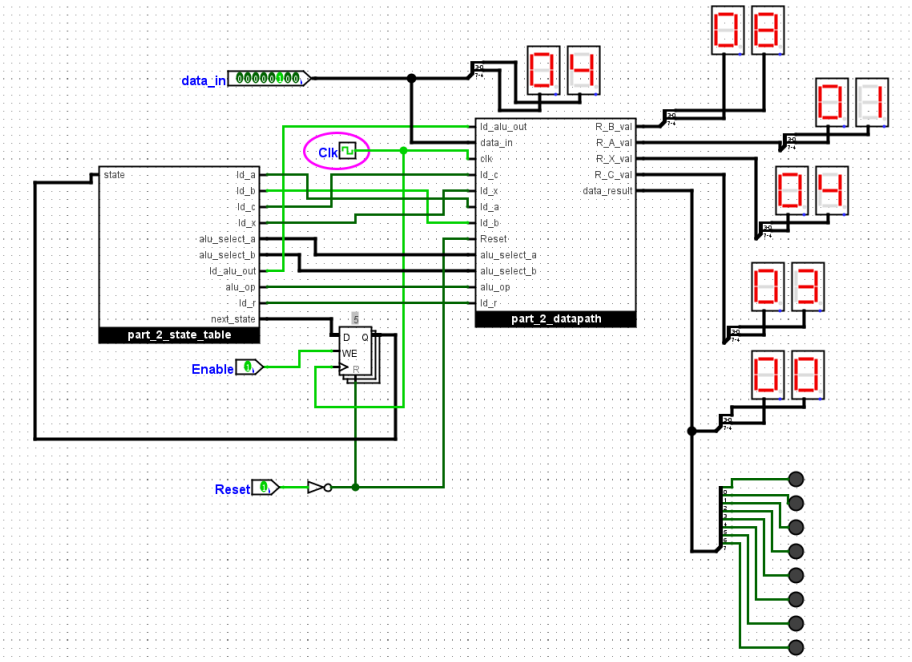
3.



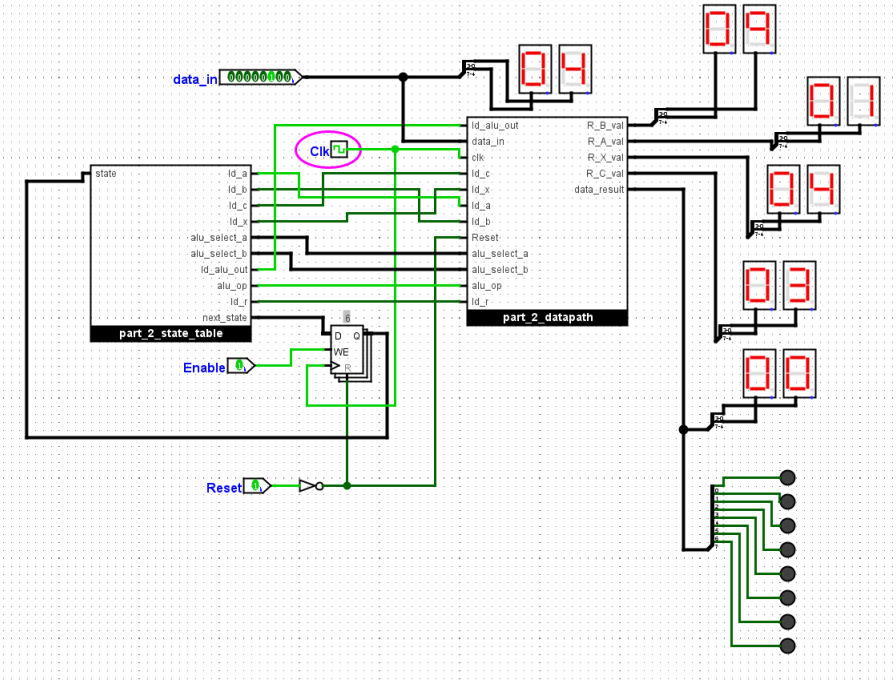
5. Load $1, 2, 3, 4$ into A, B, C, D respectively with clock ticks 4 times.



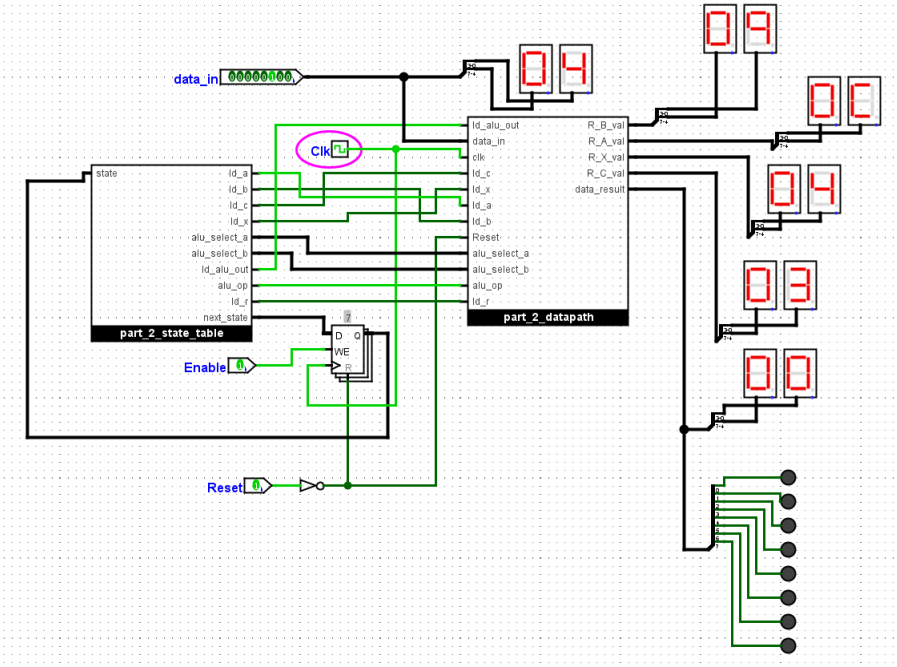
After one clock cycle, we have $R_B = R_B \cdot R_X = 08$.



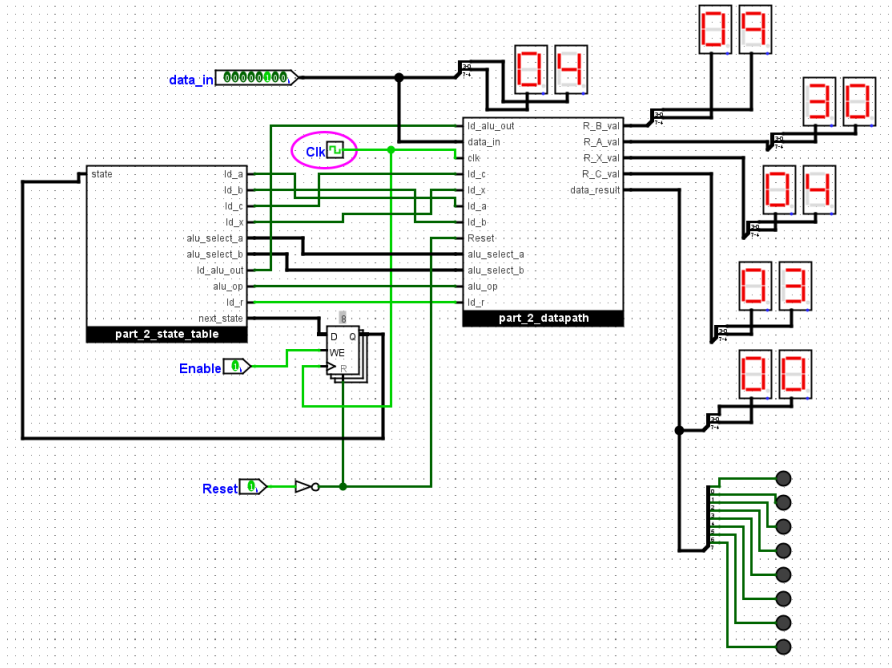
After one clock cycle, we have $R_B = R_B + R_A = 09$.



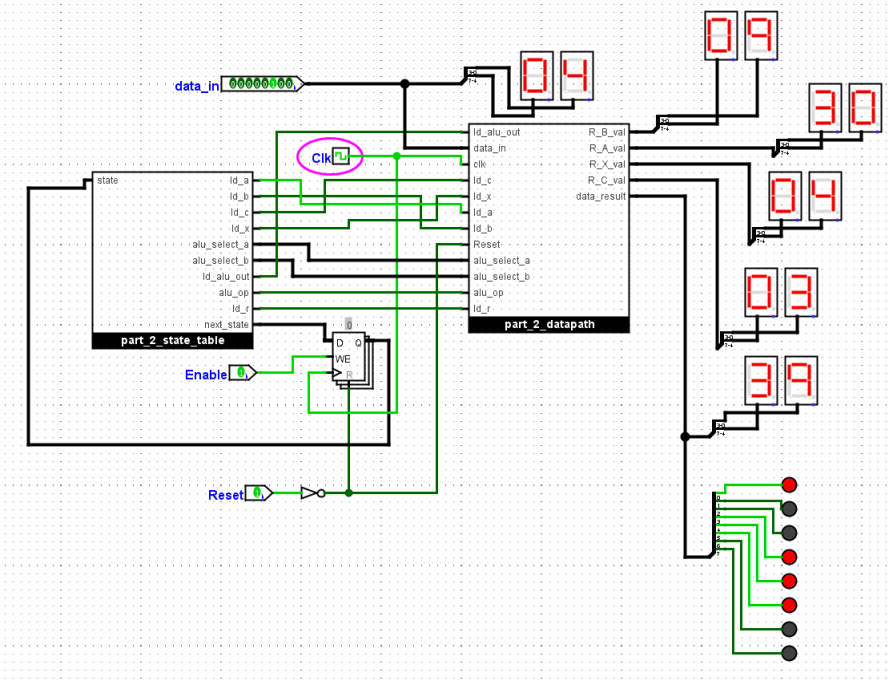
After one clock cycle, we have $R_A = R_C \cdot R_X = 0C$.



After one clock cycle, we have $R_A = R_C \cdot R_X = 30$.



After one clock cycle, we get the final output $R_A + R_B = 39$.


data_in
Clk
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
Enable
Reset
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
04
09
30
04
03
39