VE270 Recitation Class

2020.7.8

Yiqing Zhao

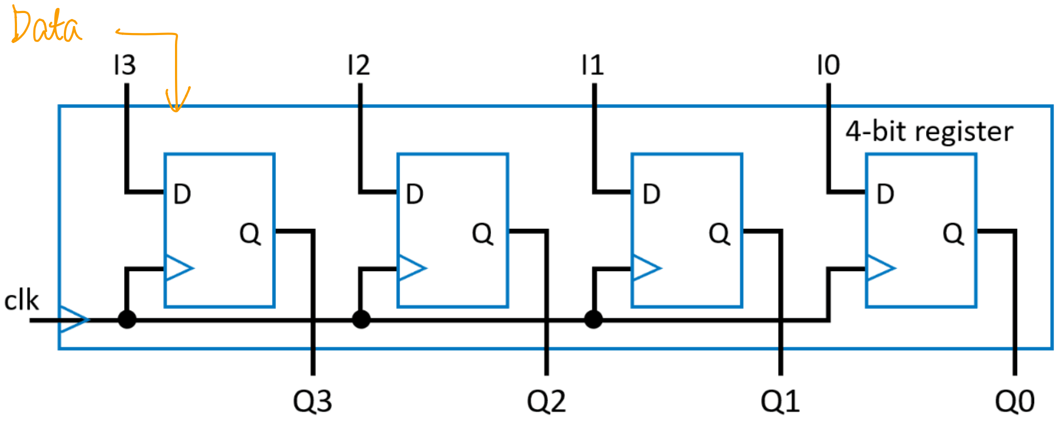
Outline

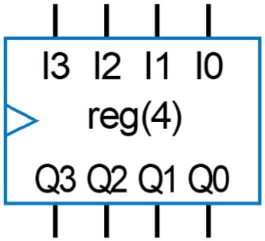
- Registers
  A. Register with Load Signal
  B. Shift Register&Rotate Register
  C. Universal Shift Register
  D. Register Examples
- Shifters
  A. Bigger Shifter
- Finite State Machine
  A. FSM (Controller) Design

Registers

General Description

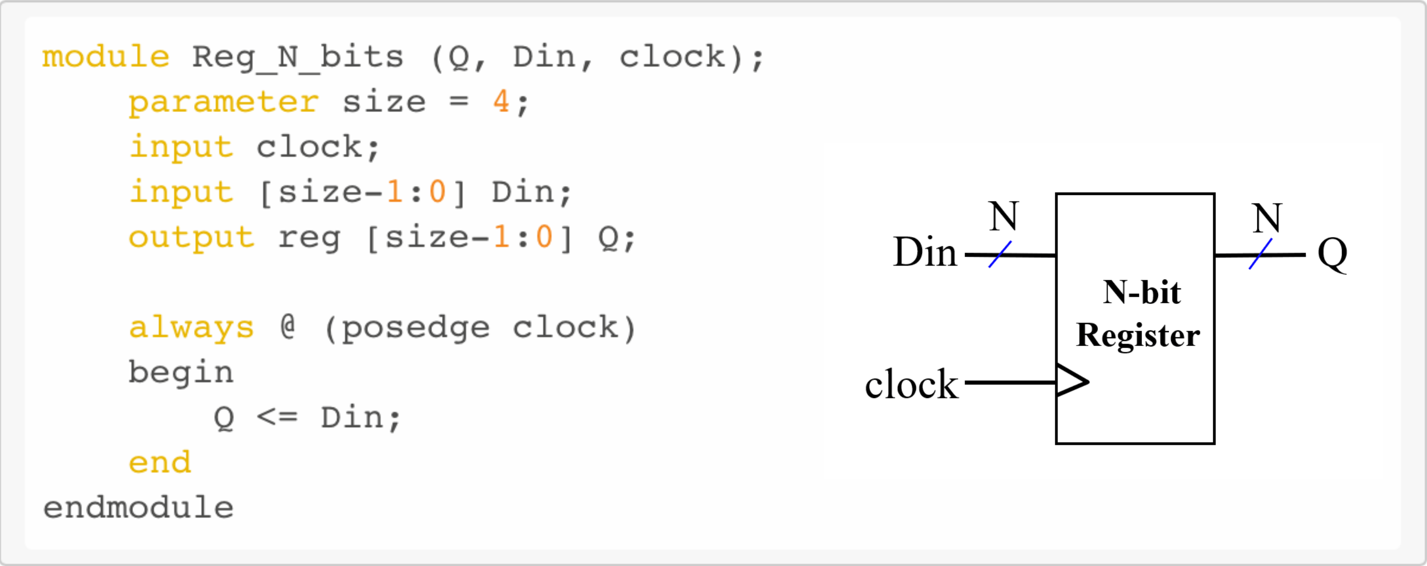
- Registers: Sets of Flip-Flops, can store data





Registers

General Description

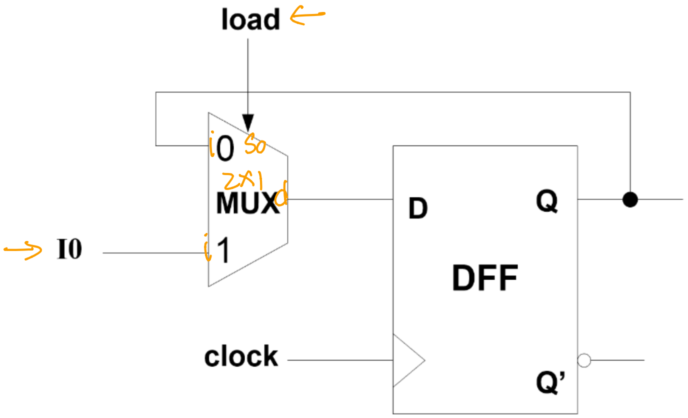


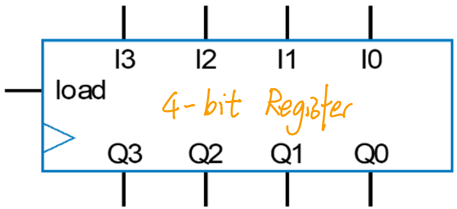
```
module Reg_N_bits (Q, Din, clock);
    parameter size = 4;
    input clock;
    input [size-1:0] Din;
    output reg [size-1:0] Q;

    always @ (posedge clock)
    begin
        Q <= Din;
    end
endmodule
```

Registers

Register with Load Signal

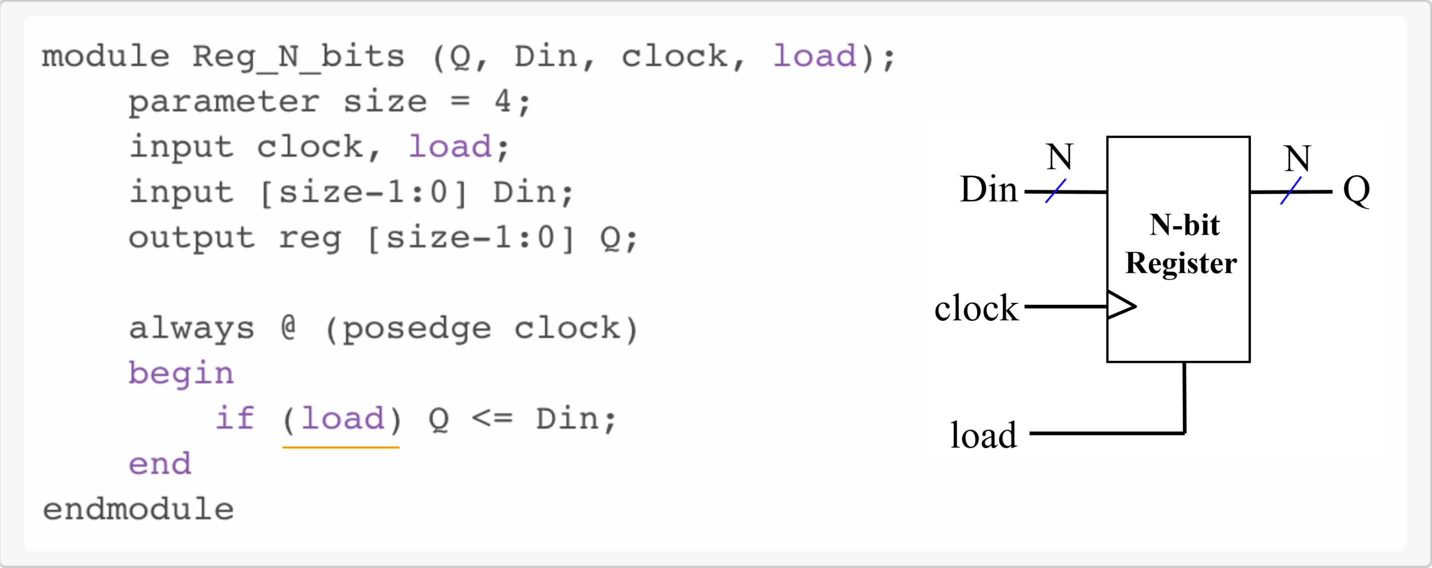
- When load = 1, we can load the value of the current input





Registers

Register with Load Signal

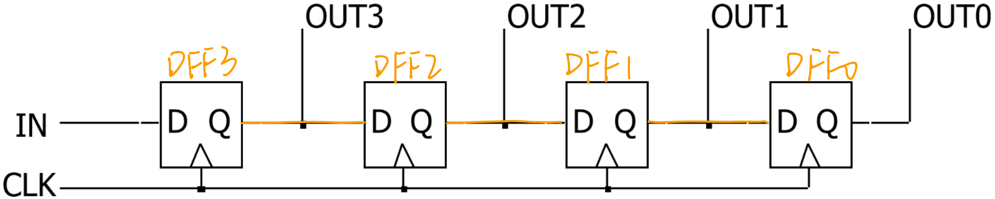


```
module Reg_N_bits (Q, Din, clock, load);
    parameter size = 4;
    input clock, load;
    input [size-1:0] Din;
    output reg [size-1:0] Q;

    always @ (posedge clock)
    begin
        if (load) Q <= Din;
    end
endmodule
```

Registers

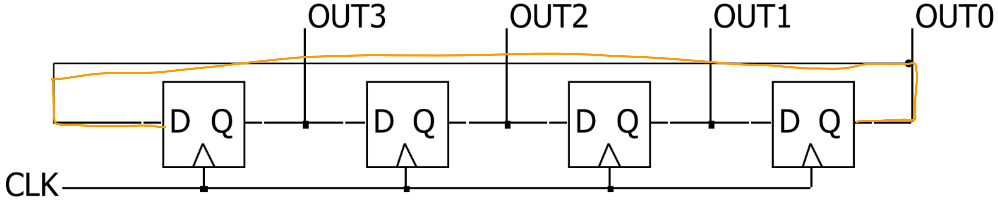
Shift Registers & Rotate Registers

- Shift Registers: Connect Q output of one flip-flop to the D input of the next flip-flop
- Rotate Registers: The same as shift registers, but connect Q output of last flip-flop to the D input of the first flip-flop



Suppose IN = 1, Init = 0011:

0011->1001->1100->1110->1111

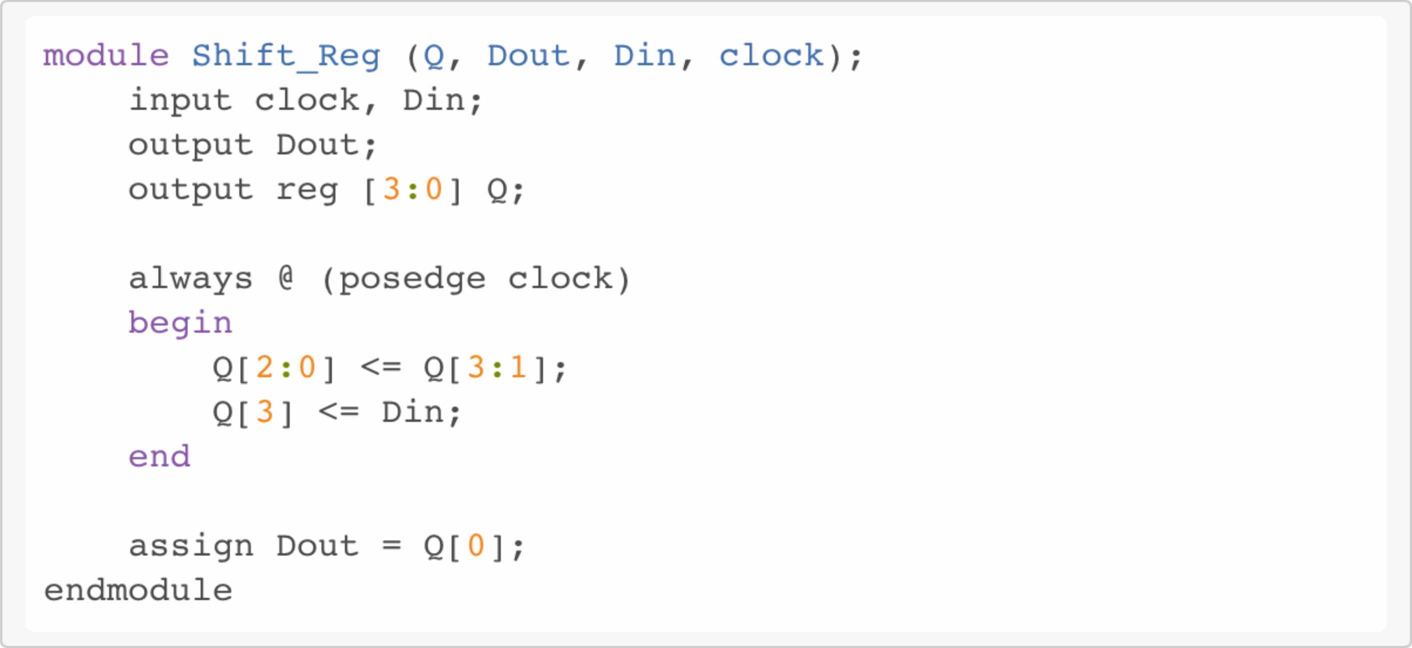


Suppose Init = 0011:

0011->1001->1100->0110->0011

Registers

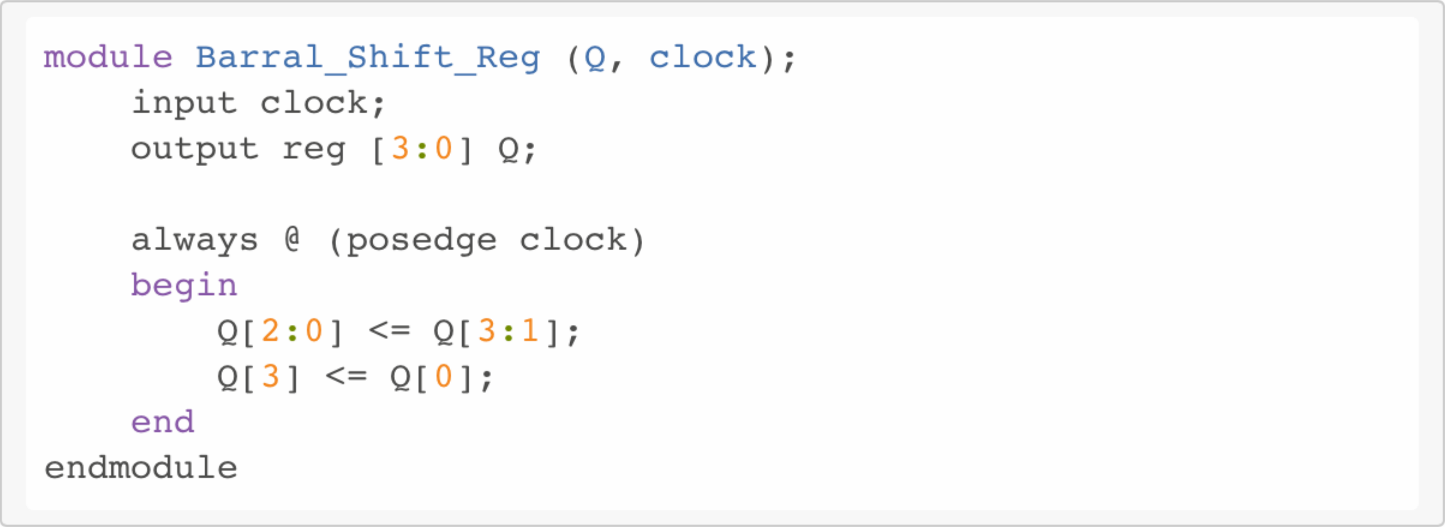
Shift Registers & Rotate Registers



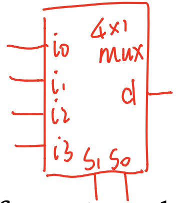
```
module Shift_Reg (Q, Dout, Din, clock);
    input clock, Din;
    output Dout;
    output reg [3:0] Q;

    always @ (posedge clock)
    begin
        Q[2:0] <= Q[3:1];
        Q[3] <= Din;
    end

    assign Dout = Q[0];
endmodule
```

Registers

Shift Registers & Rotate Registers



```
module Barral_Shift_Reg (Q, clock);
    input clock;
    output reg [3:0] Q;

    always @ (posedge clock)
    begin
        Q[2:0] <= Q[3:1];
        Q[3] <= Q[0];
    end
endmodule
```



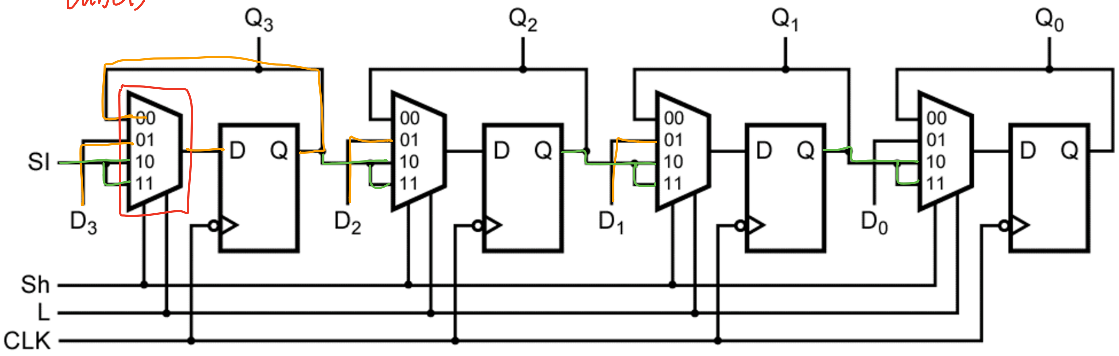
Registers

☆ Universal Shift Register

- Multi-functional Shift Register: Shift Right, Load, Hold, ...

Load = $D_3 D_2 D_1 D_0$

Labels



Sh(Shift)	L(Load)	Action
0	0	Hold
0	1	Load
1	X	Shift Right

Registers

Universal Shift Register

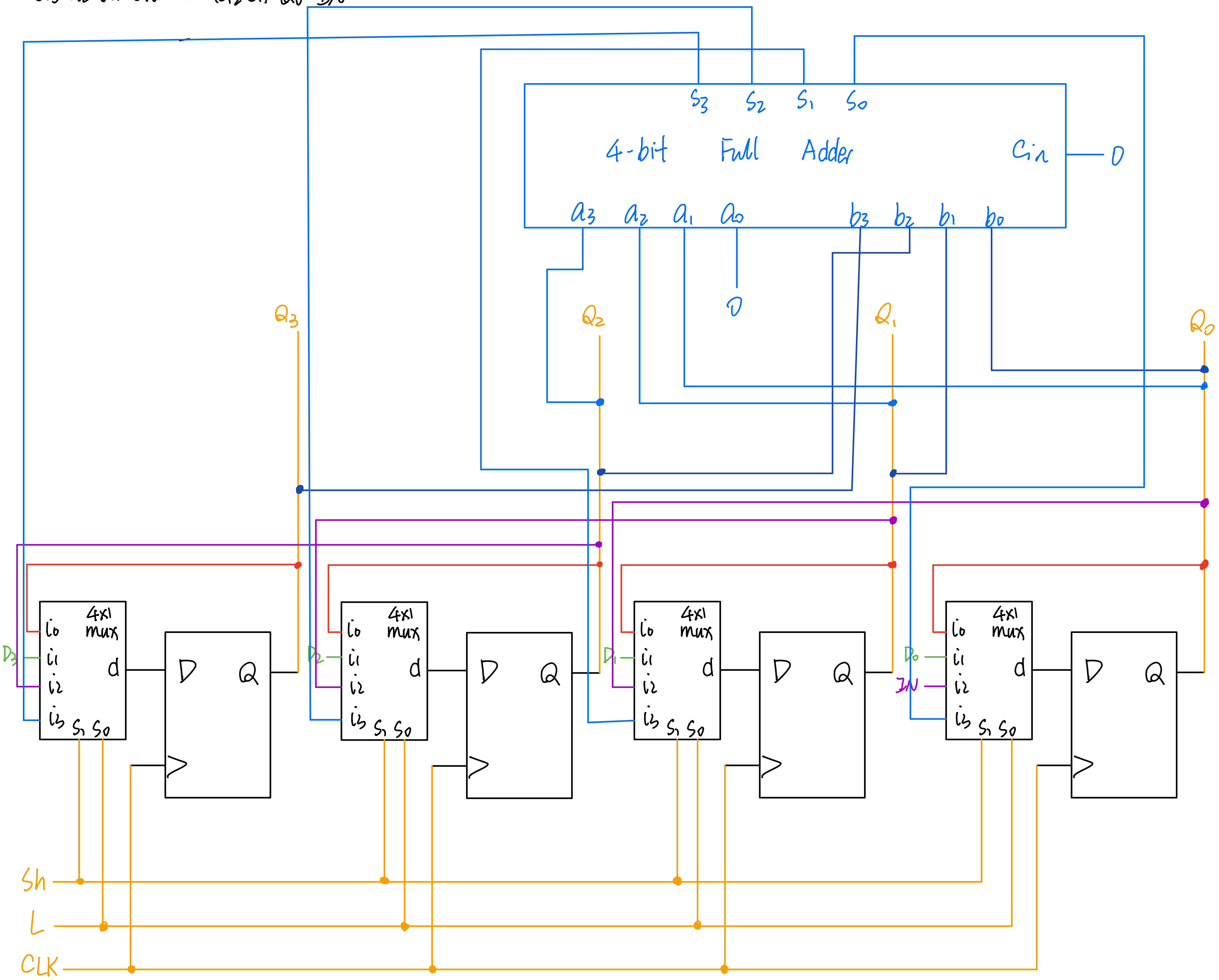
- Design a register with the following control signals:

Sh(Shift)	L(Load)	Action
0	0	**Hold**
0	1	**Load**
1	0	**Shift Left**
1	1	**3*Content**

$\underline{Q_3Q_2Q_1Q_0 \times 2} + Q_3Q_2Q_1Q_0$

$Q_2Q_1Q_0\ 0$

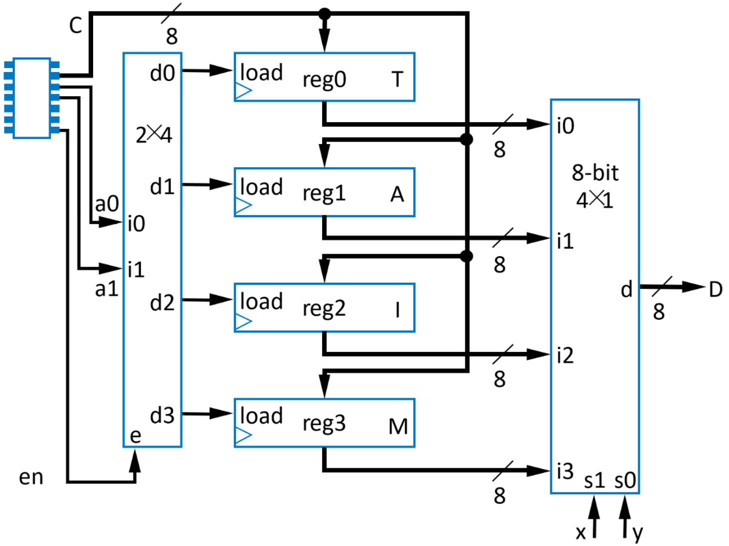
$Q_3 Q_2 Q_1 Q_0 \Rightarrow Q_2 Q_1 Q_0$ IN



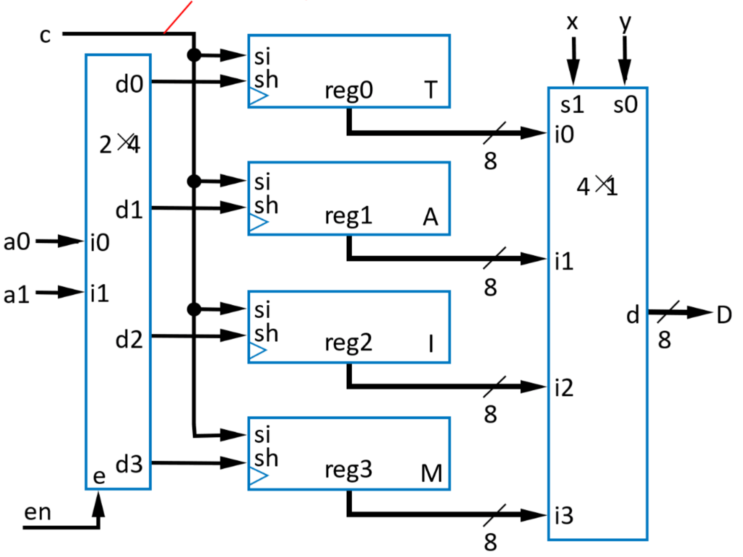
Registers

Register Examples

- Above-Mirror Display



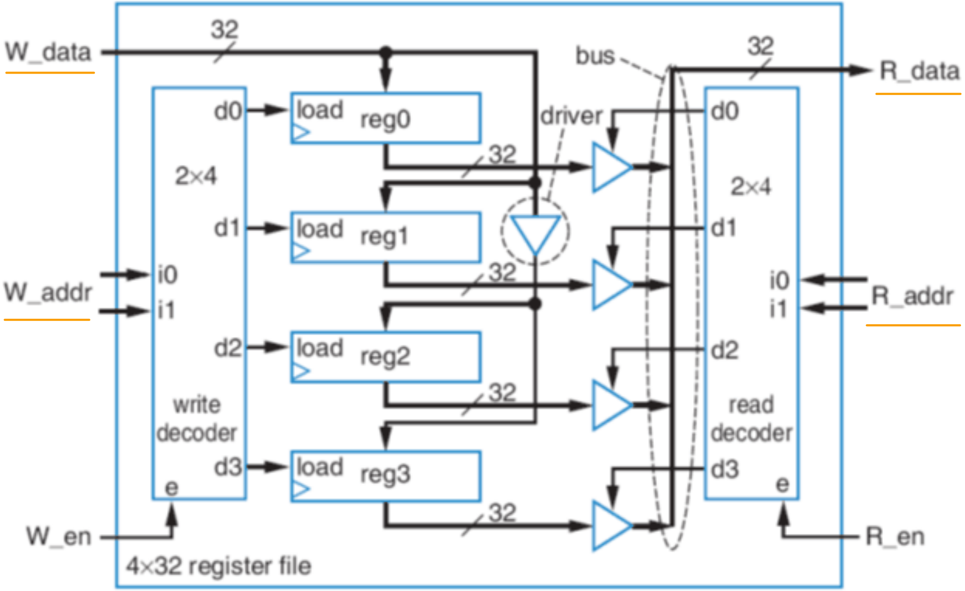
Note: this line is 1 bit, rather than 8 bits like before



Registers

Register Examples

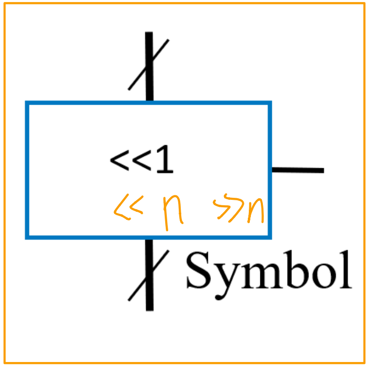
- Register File



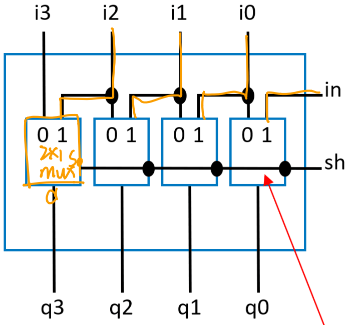
Shifters

General Description

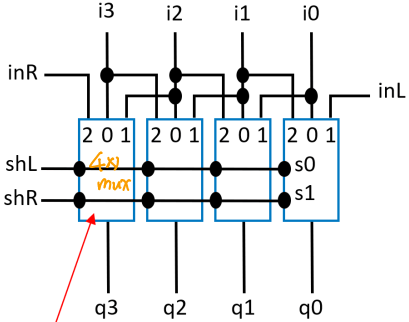
- Shifters are not Shift Registers: Shifters use MUX, and change value immediately
- X>>1 equals X/2
- X<<1 equals X*2



Symbol



Shifter with left shift or no shift



Shifter with left shift, right shift, and no shift

MUX

Shifters

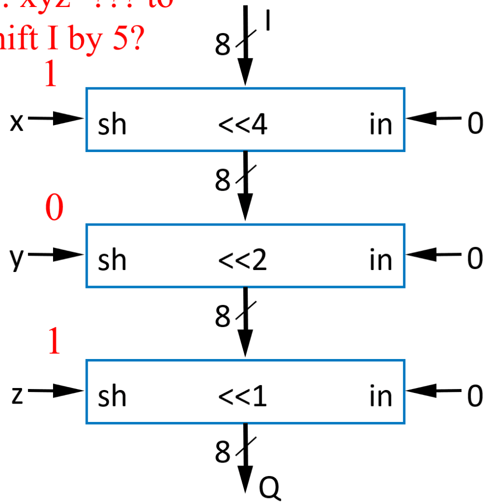
Bigger Shifter

- A shifter that can shift by any amount - By using multiple shifters
- To shift I by N, 4x+2y+z = N, where x, y, z = 0 or 1.

$\boxed{\ll 7}$

Q: xyz=??? to shift I by 5?

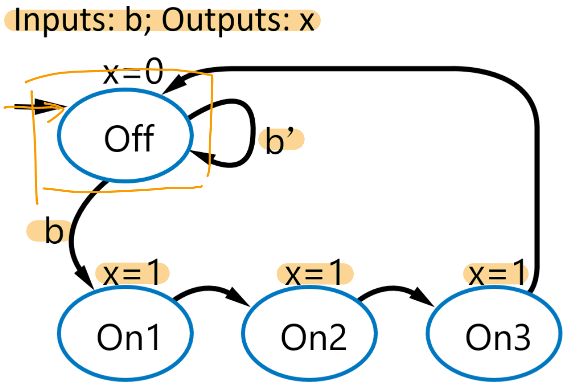
$\boxed{101}$



Finite State Machine

General Description

- Finite State Machine (FSM) has states, inputs, outputs, initial state, transitions.
- State Diagram & State Table:



Initial States

State diagram

State table.

		Inputs			Outputs		
	Present State	s1	s0	b	x	n1	n0
00	*Off*	0	0	0	0	0	0
		0	0	1	0	0	1
01	*On1*	0	1	0	1	1	0
		0	1	1	1	1	0
10	*On2*	1	0	0	1	1	1
		1	0	1	1	1	1
11	*On3*	1	1	0	1	0	0
		1	1	1	1	0	0

next state

Finite State Machine

FSM (Controller) Design

- Five steps:
  1. Capture the FSM
  2. Create the architecture
  3. Encode the states
  4. Create the state table
  5. Implement the combinational logic

Finite State Machine

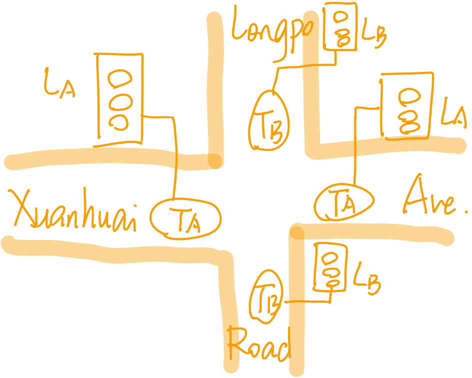
FSM (Controller) Design

- One day, Engineering students are rushing to Long Bin Building from their dorms on Xuanhuai Avenue. They are busy reading about FSMs in their favorite textbook and aren’t looking where they are going.
- At the same time, Math students are rushing the 5th Canteen to get their favorite Pan Fried dumplings on Longpo Road. They are solving differential equations in their minds and aren’t looking where they are going either.
- Then, a serious injury occurs at the intersection of the two roads. A student, Bit, decides to solve this problem by using FSM.

Finite State Machine

FSM (Controller) Design

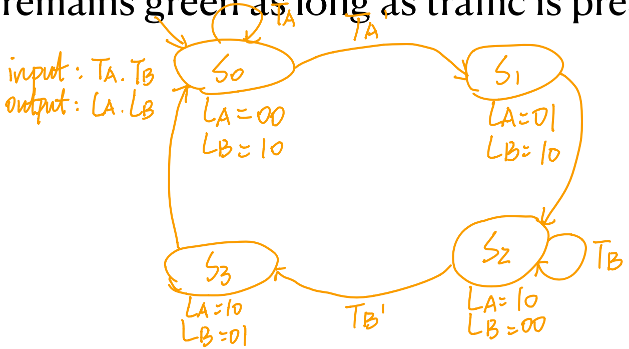
- Bit uses two traffic sensors, T_A and T_B, on Xuanhuai Ave. and Longpo Road, respectively.
  - Sensors show 0: students present
  - Sensors show 1: students not present
- Bit uses two traffic lights, L_A and L_B, to control traffic. The traffic lights has three mode. (Green: 00, Yellow: 01, Red: 10)



Finite State Machine

FSM (Controller) Design

- The FSM controller updates on the rising edge of the clock signal:
  - At beginning, the lights are green on Xuanhuai Ave. and red on Longpo Road.
  - If traffic appears on Xuanhuai $T_A=1$ Ave., the lights do not change. When there is no longer traffic on Xuanhuai $T_A=0$ Ave., the light on Xuanhuai Ave. becomes yellow for a clock cycle before it turns red and Longpo road's light turns green.
  - Similarly, the Longpo road's light remains green as long as traffic $T_B=1$ is present, then turns yellow and eventually red.

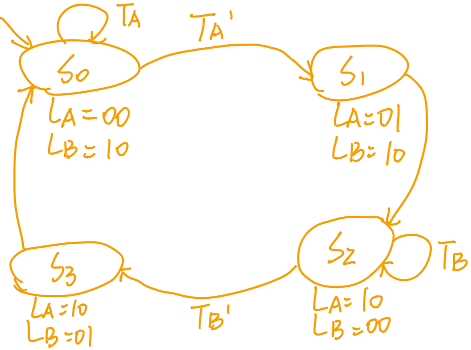


Finite State Machine

FSM (Controller) Design

- Step 1: Capture the FSM
  - Among all transitions leaving a state, only one condition should be true.
  - Among all transitions leaving a state, one condition must be true.
  - All conditions must be considered when leaving a state.

input : $T_A . T_B$
output : $L_A . L_B$

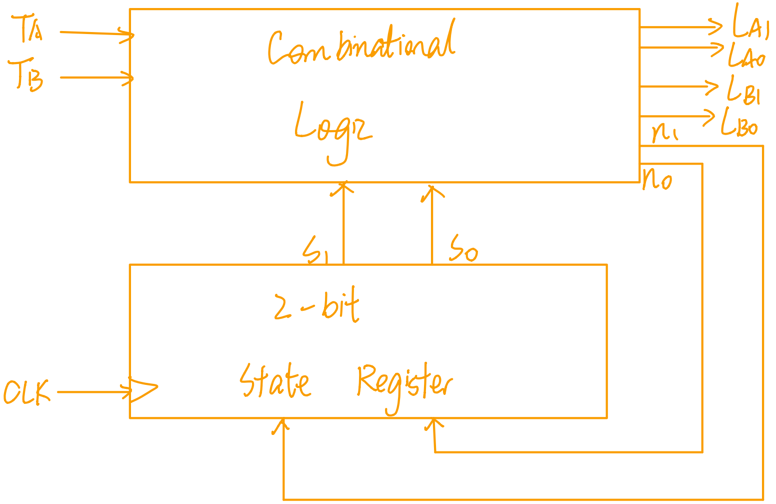


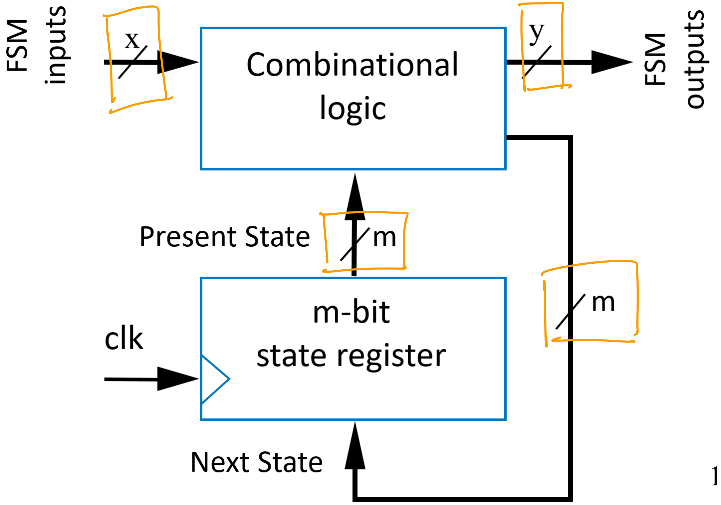
L_A L_B = 2 bits
T_A T_B 1 bit

Finite State Machine

FSM (Controller) Design

- Step 2: Create the architecture L_{A1} L_{A0} L_{B1} L_{B0}





Finite State Machine

FSM (Controller) Design

- Step 3: Encode the states

$S_0 = 00$

$S_1 = 01$

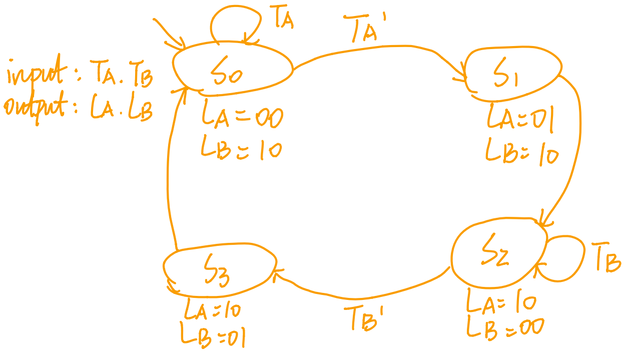
$S_2 = 10$

$S_3 = 11$

Finite State Machine

FSM (Controller) Design

- Step 4: Create the state table



Input				Output					
Present State		FSM In		Next State		FSM Output			
S_1	S_0	T_A	T_B	N_1	N_0	L_{A1}	L_{A0}	L_{B1}	L_{B0}
0	0	0	X	0	1	0	0	1	0
0	0	1	X	0	0	0	0	1	0
0	1	X	X	1	0	0	1	1	0
1	0	X	0	1	1	1	0	0	0
1	0	X	1	1	0	1	0	0	0
1	1	X	X	0	0	1	0	0	1

Finite State Machine

FSM (Controller) Design

- Step 5: Implement the combinational logic

$N_1 = S_1'S_0 + \underline{S_1S_0'T_B' + S_1S_0'T_B} = S_1'S_0 + S_1S_0' = S_1 \oplus S_0$

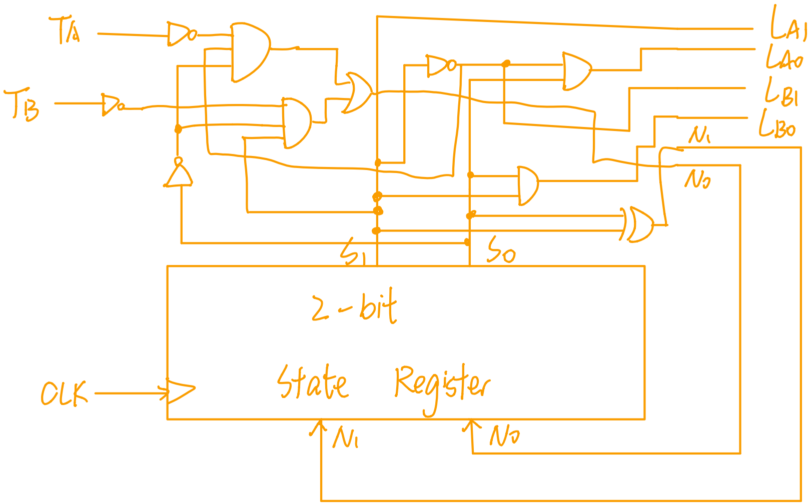
$N_0 = \underset{S_1'S_0'T_A'}{} + S_1S_0'T_B'$

$L_{A1} = S_1S_0' + S_1S_0 = S_1$

$L_{A0} = S_1'S_0$

$L_{B1} = S_1'S_0' + S_1'S_0 = S_1'$

$L_{B0} = S_1S_0$



Any Questions?