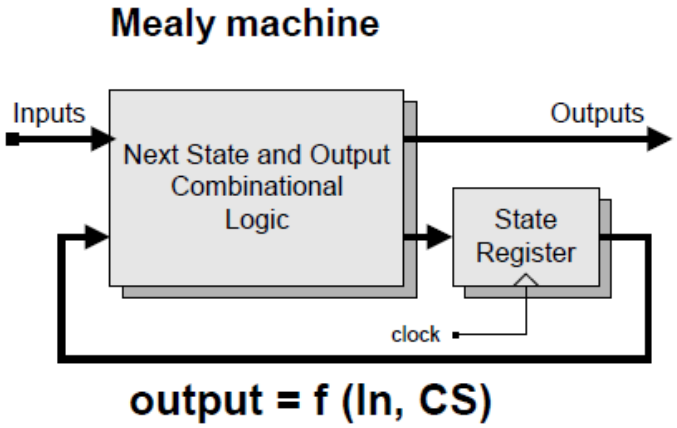
DCS Lab 4
Sequence

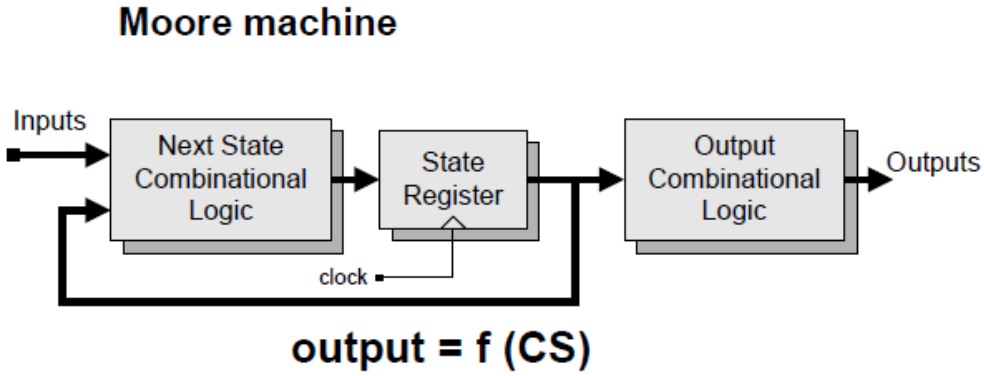
Finite state machine

郭晏誠

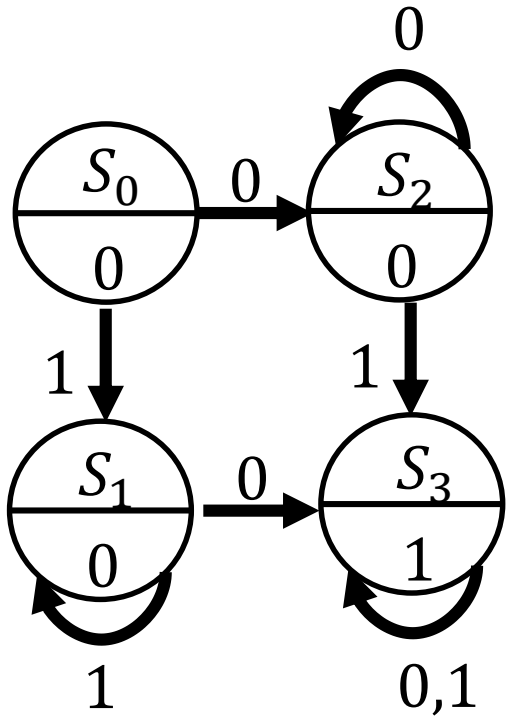
Mealy and Moore Machine

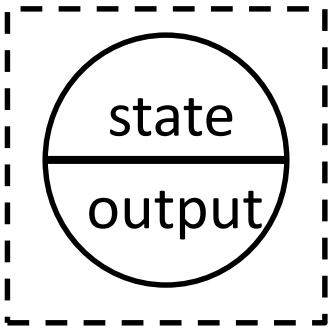
- Mealy machine
  - The outputs depend on the current state and inputs
- Moore machine
  - The outputs depend on the current state only





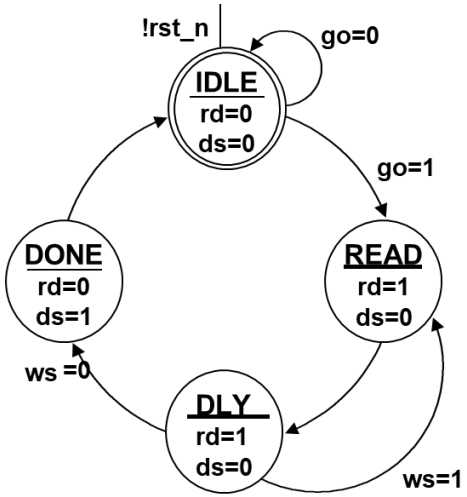
FSM coding style (example1)






```
input clk,rst_n,in_data;
output logic out;
logic  [1:0] cur_state,next_state;
logic out_comb;
parameter S_0 = 2'd0;
parameter S_1 = 2'd1;
parameter S_2 = 2'd2;
parameter S_3 = 2'd3;

always_ff @(posedge clk or negedge rst_n) begin
    if(!rst_n) begin
        cur_state <= S_0;
    end else begin
        cur_state <= next_state;
    end
end

always_comb  begin
    case(cur_state)
        S_0:     next_state = ( in_data == 0 ) ? S_2 : S_1;
        S_1:     next_state = ( in_data == 0 ) ? S_3 : S_1;
        S_2:     next_state = ( in_data == 0 ) ? S_2 : S_3;
        S_3:     next_state = ( in_data == 0 ) ? S_3 : S_3;
        default:next_state = cur_state;
    endcase
end

assign out_comb = ( next_state == S_3 ) ? 1 : 0;
always_ff @(posedge clk or negedge rst_n) begin
    if(!rst_n) begin
        out <= 0;
    end else begin
        out <= out_comb;
    end
end
```

FSM coding style (example2)



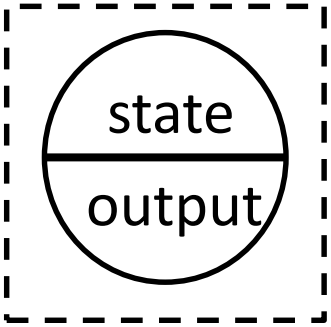

```
module fsm_cc1_2
(output logic rd, ds,
input go, ws, clk, rst_n);
//state parameter declaration
Parameter IDLE = 2'b00,
          READ = 2'b01,
          DLY  = 2'b11,
          DONE = 2'b10;

logic [1:0] curr_state, next_state;

always_ff@(posedge clk or negedge rst_n) begin
    if (!rst_n)      curr_state <= IDLE;
    else             curr_state <= next_state;
end
```

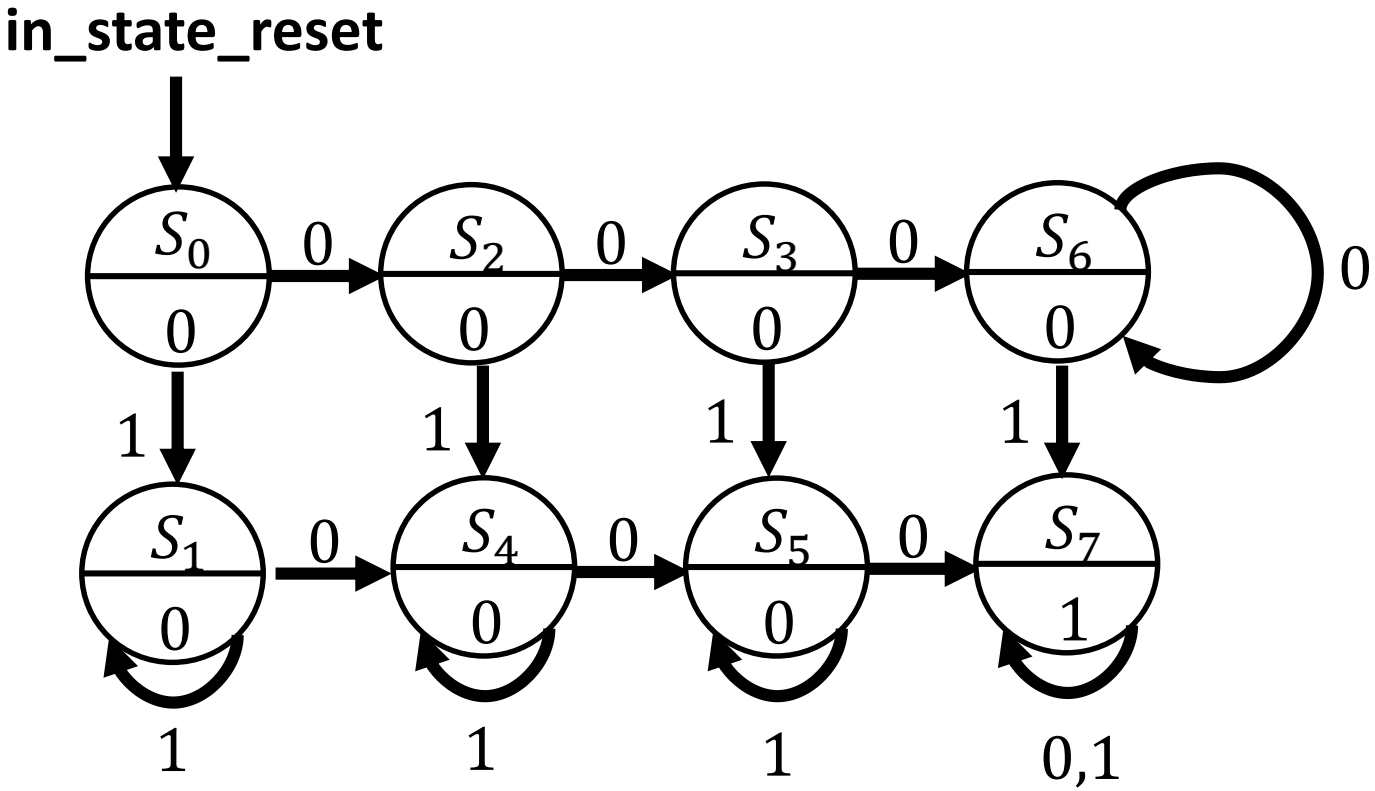
```
//next state and output logic
always_comb begin
    next = 'bx;
    rd = 1'b0;
    ds = 1'b0;
    case (curr_state)
    IDLE: begin
          if (go)    next_state = READ;
          else       next_state = IDLE;
    end
    READ: begin
          rd = 1'b1;
          next_state = DLY;
    end
    DLY : begin
          rd = 1'b1;
          if (!ws)   next_state = DONE;
          else       next_state = READ;
    end
    DONE: begin
          ds = 1'b1;
          next_state = IDLE;
    end
    endcase
end
endmodule
```

偵測Sequence 的設計

- Input 一個任意長度Sequence
- 設計FSM偵測任意長度Sequence
  是否至少有3個bit是0和1個bit是1

參考FSM


state
output


in_state_reset
S_0
0
S_2
0
S_3
0
S_6
0
0
1
1
1
1
S_1
0
S_4
0
S_5
0
S_7
1
0
0
0
1
1
1
0,1

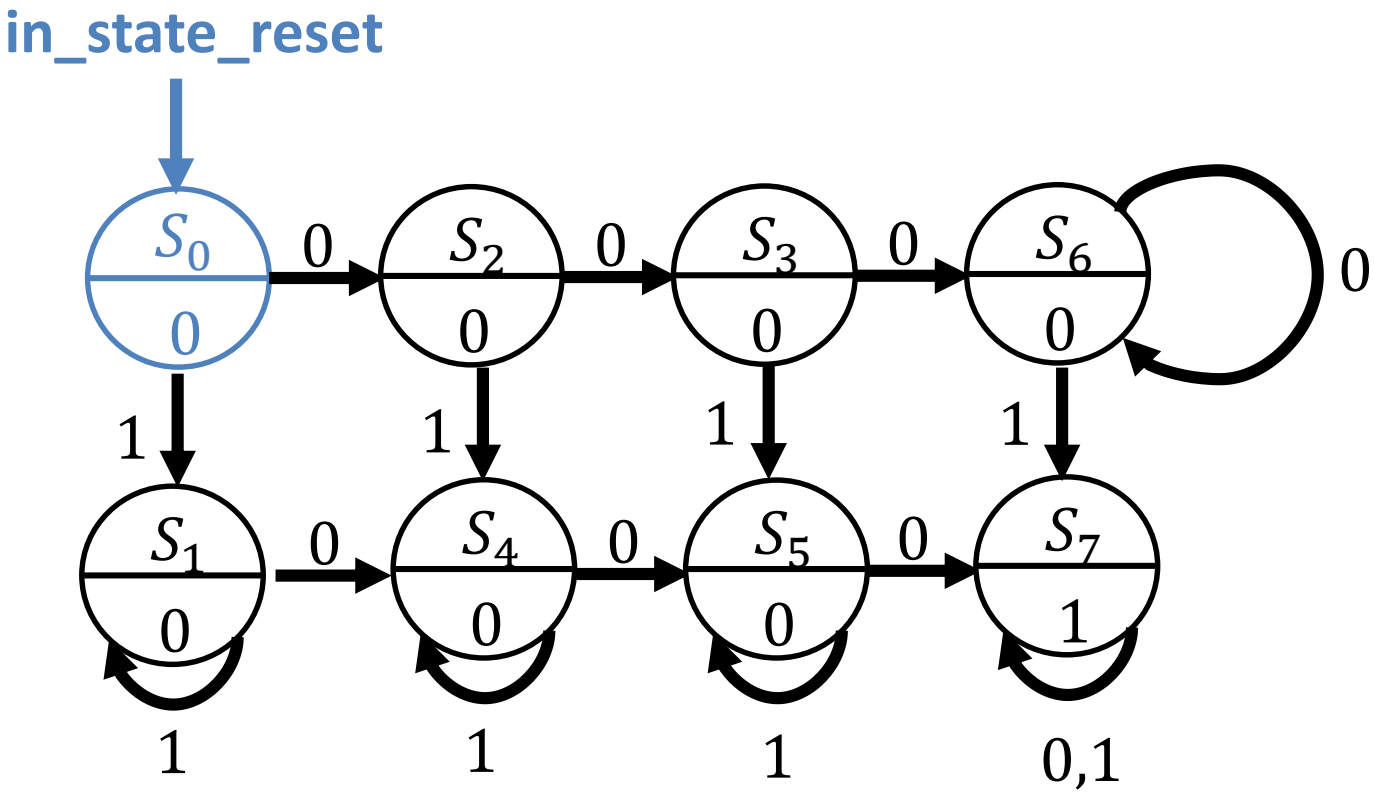
Seq.sv

Input Signal	Bit Width	Definition
clk	1	Clock
rst_n	1	Asynchronous active-low reset
in_data	1	FSM的input
in_state_reset	1	FSM的reset訊號 當in_state_reset == 1，out_cur_state=S_0(0)

Output Signal	Bit Width	Definition
out_cur_state	3	FSM current state (每個cycle進行檢查) If (!rst_n) out_cur_state =S_0(0) 當in_state_reset == 1，out_cur_state=S_0(0)
out	1	FSM output (每個cycle進行檢查) If (!rst_n) out =0 根據p6 FSM進行output

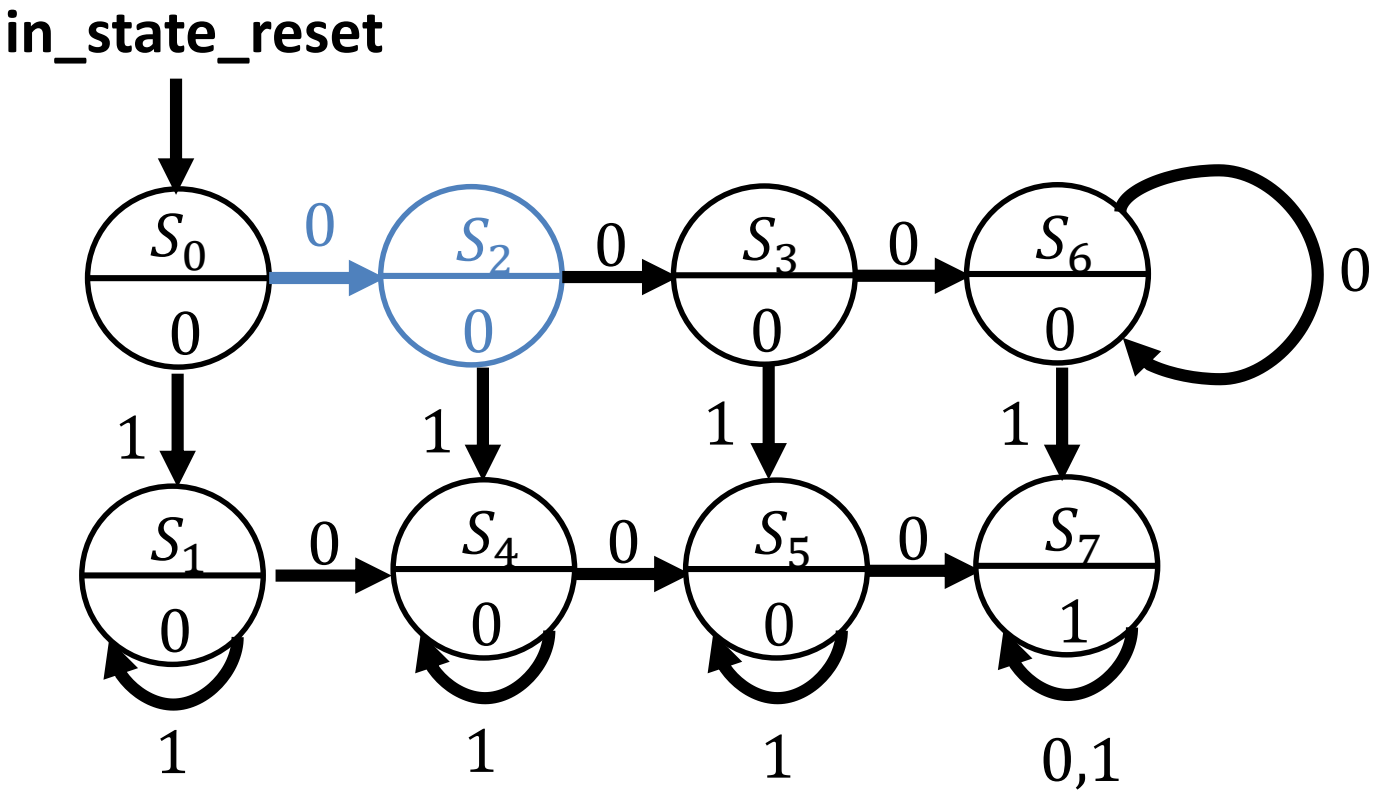
FSM example

Cycle	1	2	3	4	5	6	7	8	9	10	11
in_state_reset	1										
In_data	0										
out_cur_state		0									
out		0									



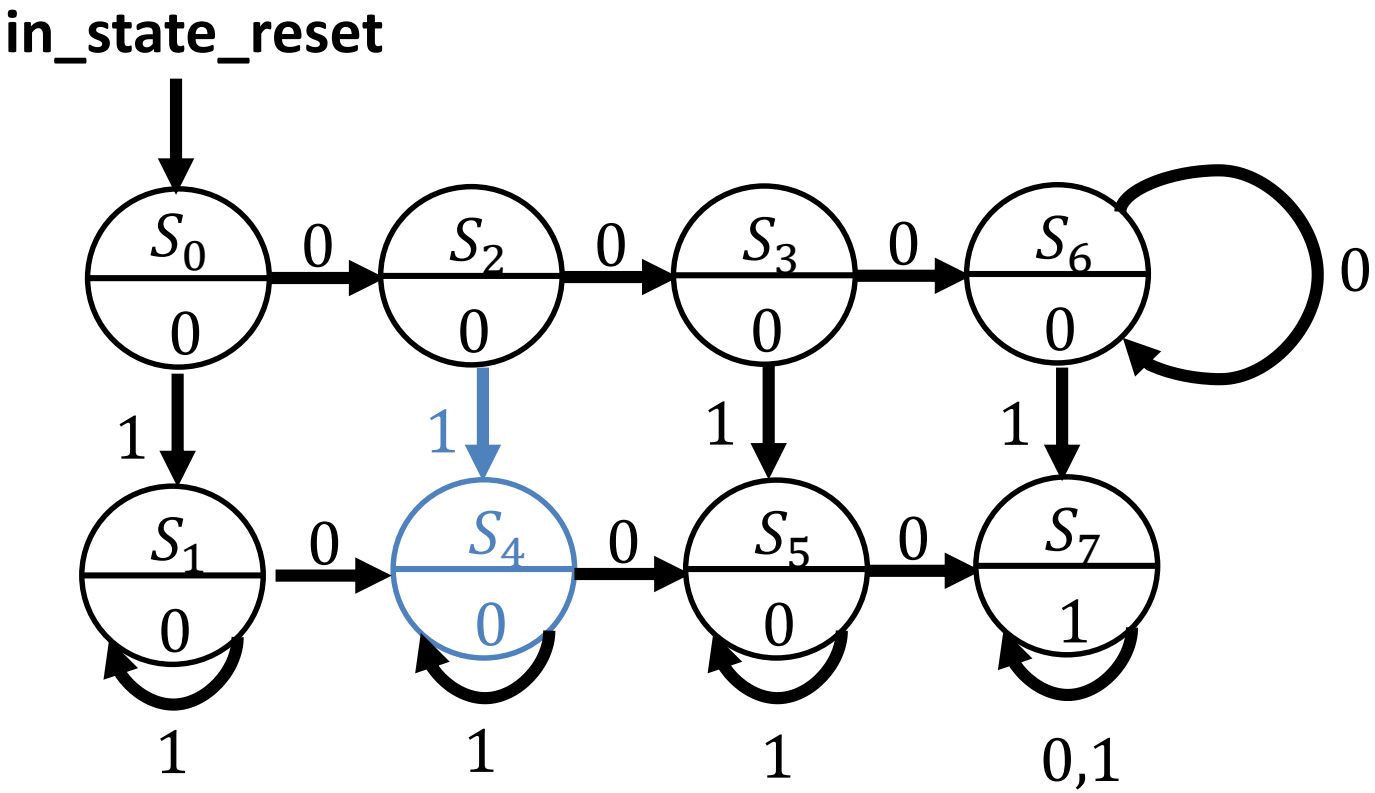
FSM example

Cycle	1	2	3	4	5	6	7	8	9	10	11
in_state_reset	1	0									
In_data	0	0									
out_cur_state		0	2								
out		0	0								



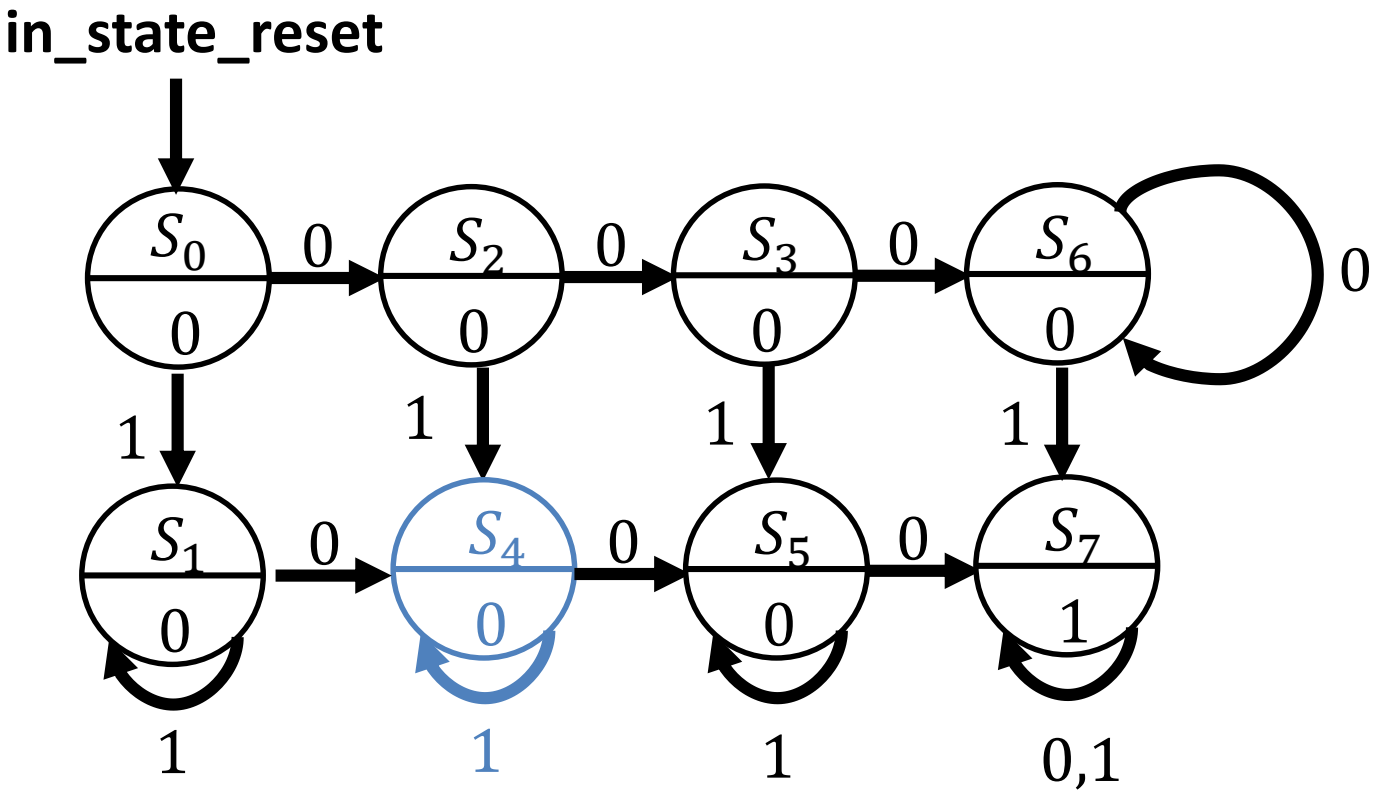
FSM example

Cycle	1	2	3	4	5	6	7	8	9	10	11
in_state_reset	1	0	0								
In_data	0	0	1								
out_cur_state		0	2	4							
out		0	0	0							



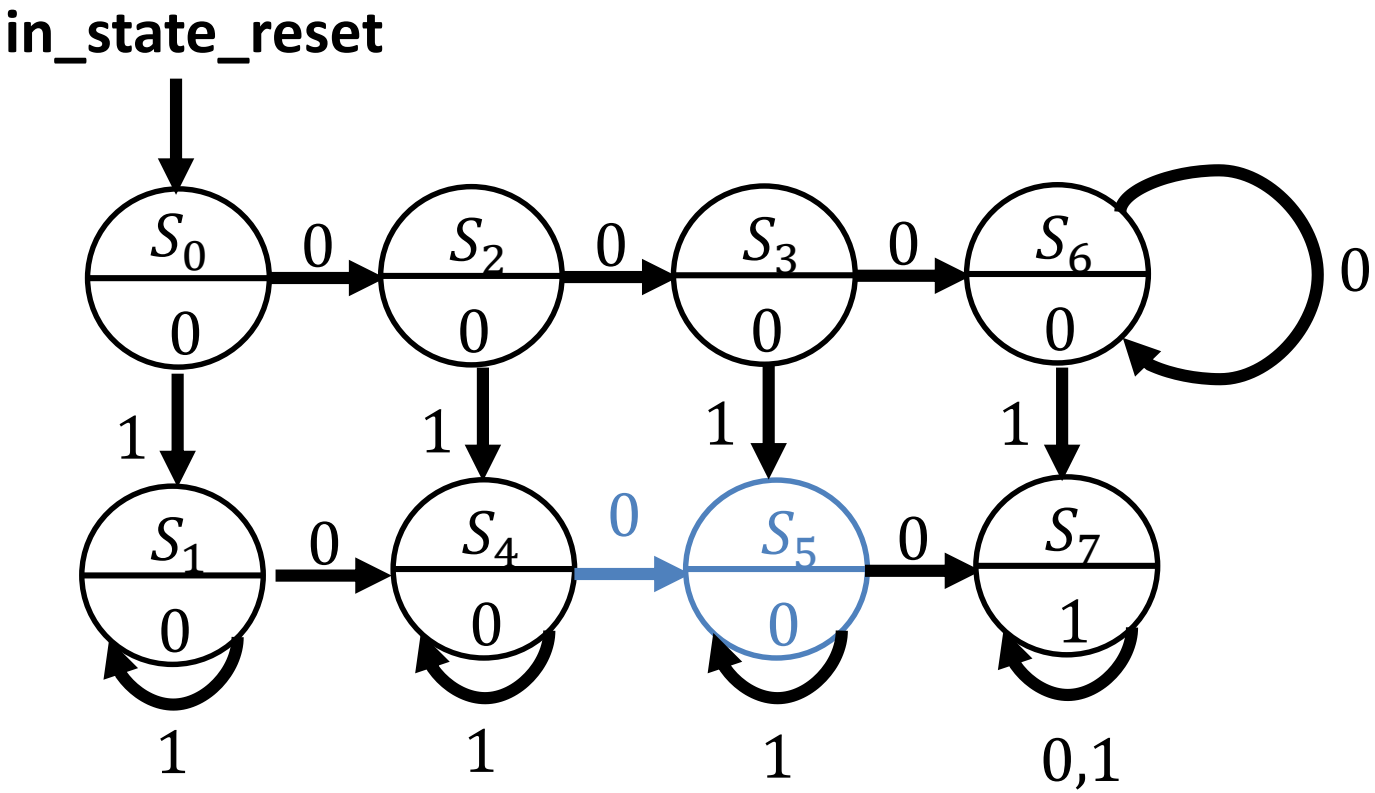
FSM example

Cycle	1	2	3	4	5	6	7	8	9	10	11
in_state_reset	1	0	0	0							
In_data	0	0	1	1							
out_cur_state		0	2	4	4						
out		0	0	0	0						



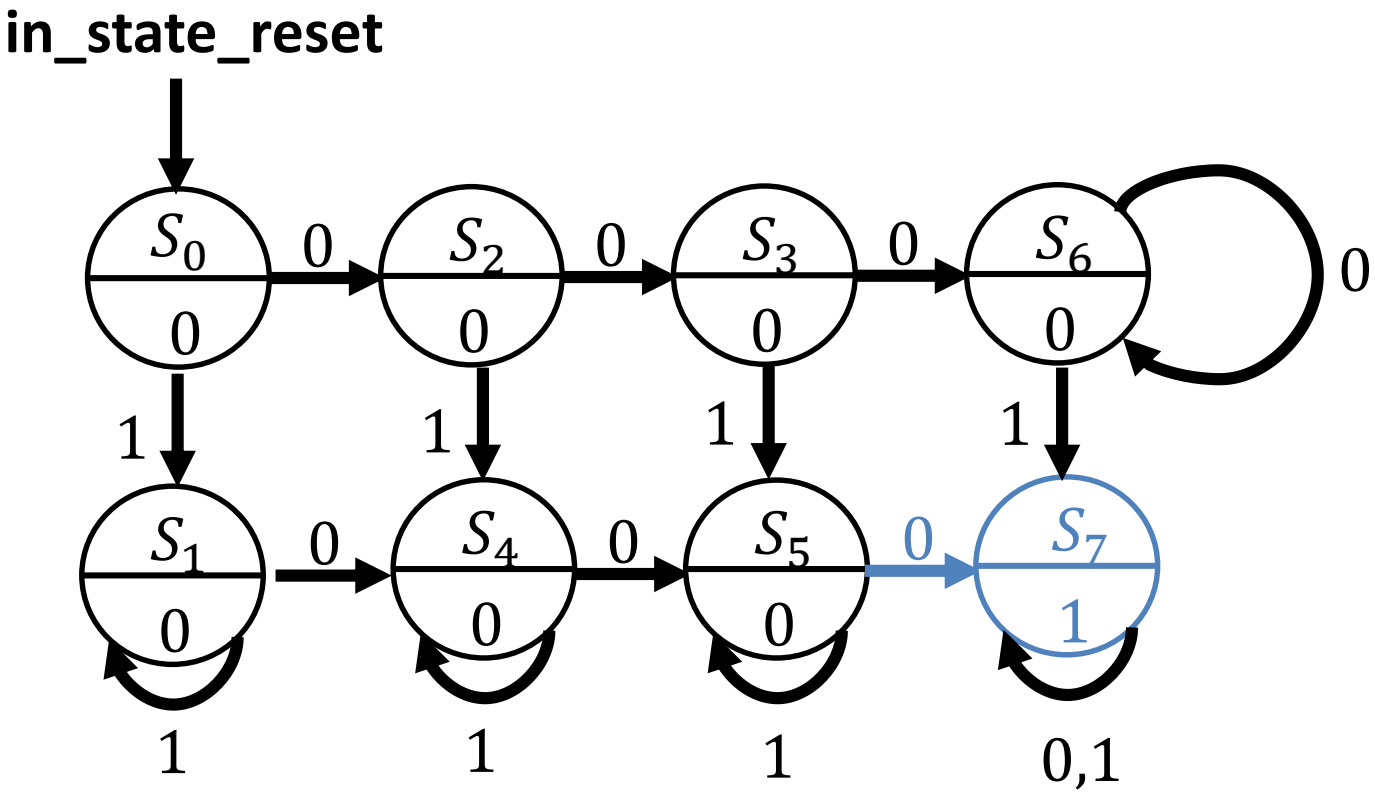
FSM example

Cycle	1	2	3	4	5	6	7	8	9	10	11
in_state_reset	1	0	0	0	0						
In_data	0	0	1	1	0						
out_cur_state		0	2	4	4	5					
out		0	0	0	0	0					



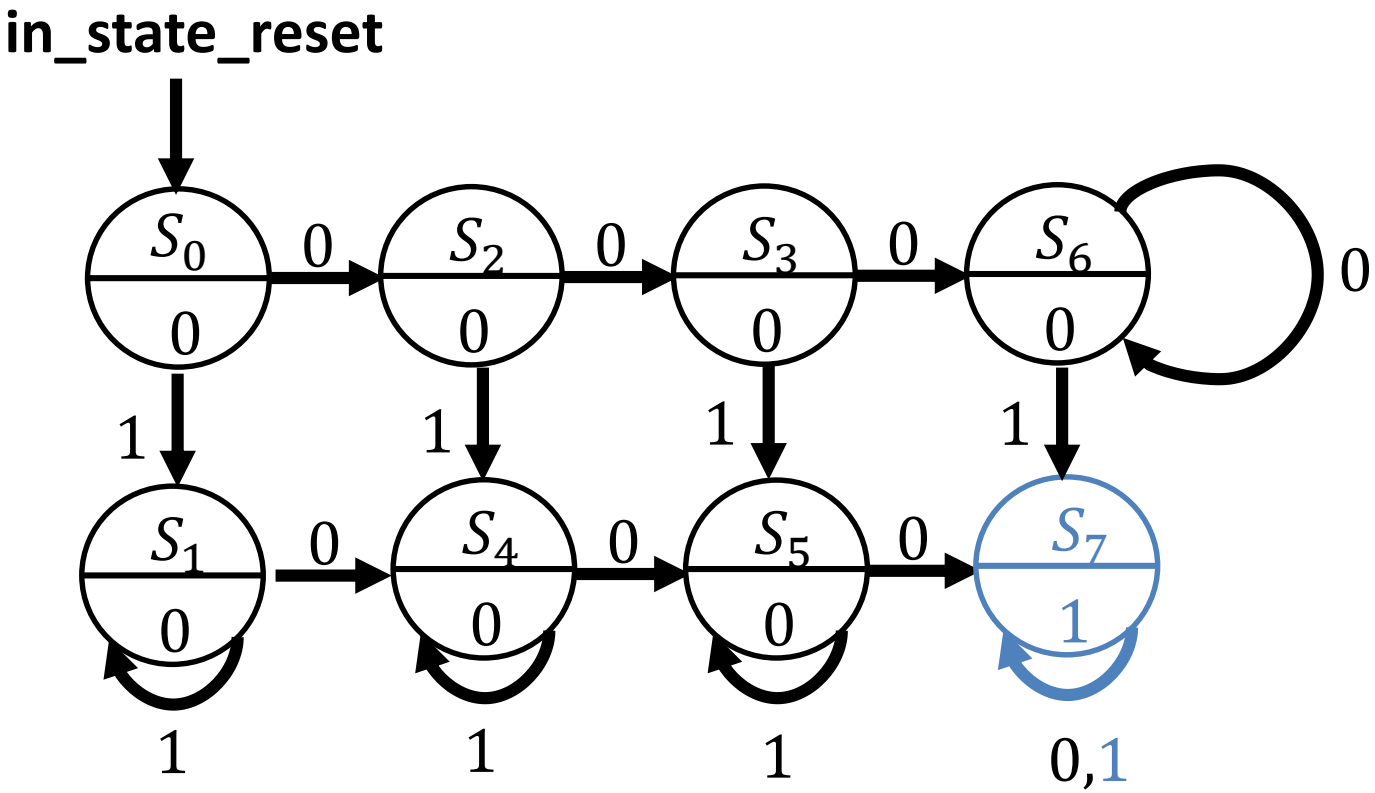
FSM example

Cycle	1	2	3	4	5	6	7	8	9	10	11
in_state_reset	1	0	0	0	0	0					
In_data	0	0	1	1	0	0					
out_cur_state		0	2	4	4	5	7				
out		0	0	0	0	0	1				



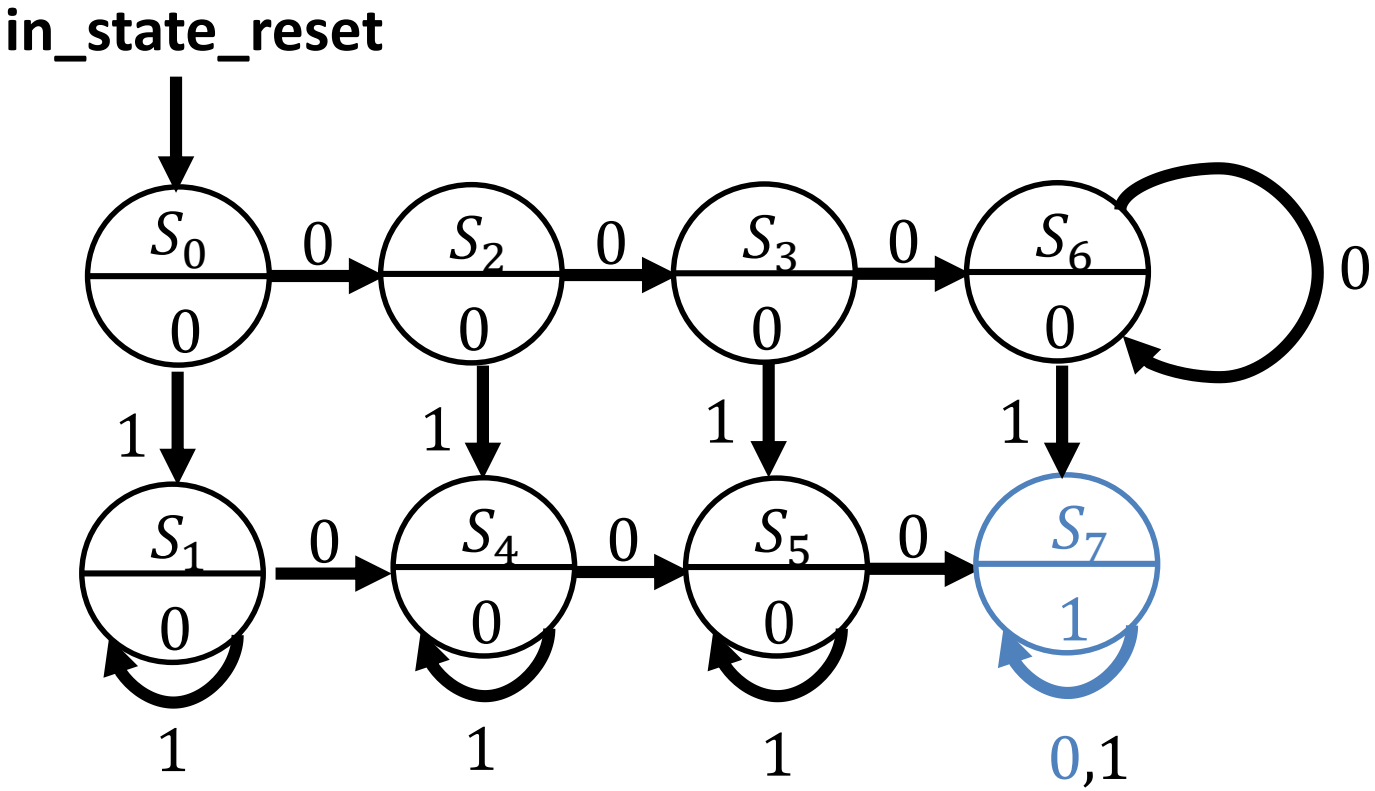
FSM example

Cycle	1	2	3	4	5	6	7	8	9	10	11
in_state_reset	1	0	0	0	0	0	0				
In_data	0	0	1	1	0	0	1				
out_cur_state		0	2	4	4	5	7	7			
out		0	0	0	0	0	1	1			



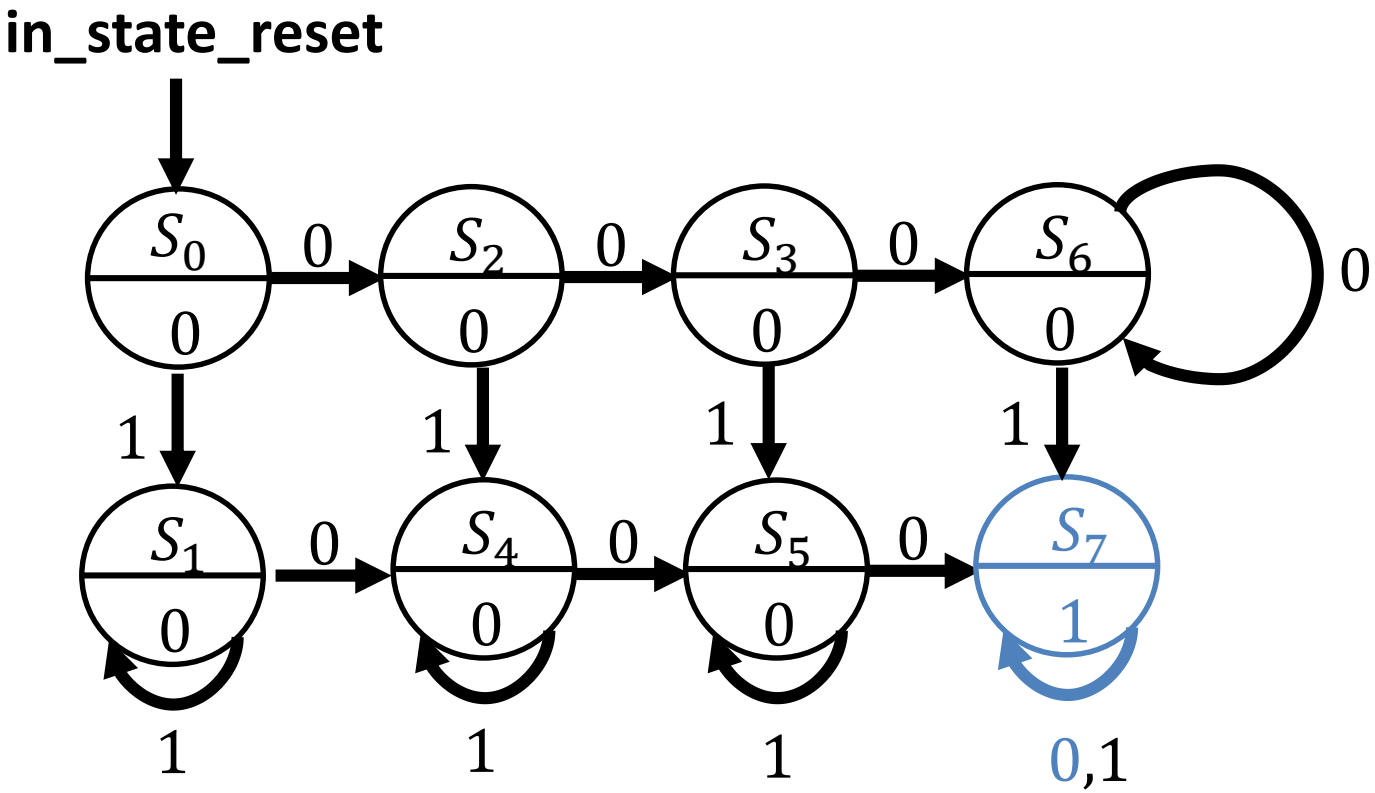
FSM example

Cycle	1	2	3	4	5	6	7	8	9	10	11
in_state_reset	1	0	0	0	0	0	0	0			
In_data	0	0	1	1	0	0	1	0			
out_cur_state		0	2	4	4	5	7	7	7		
out		0	0	0	0	0	1	1	1		



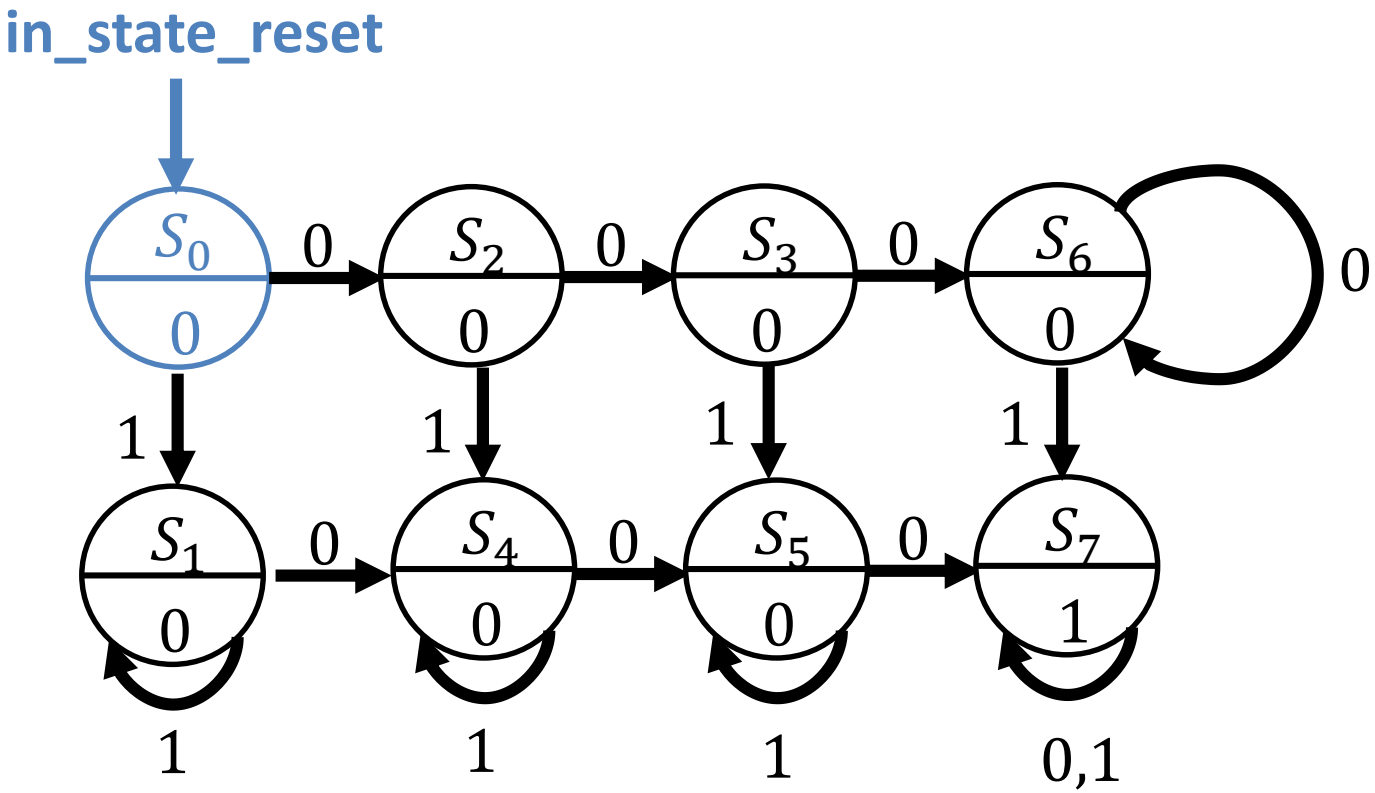
FSM example

Cycle	1	2	3	4	5	6	7	8	9	10	11
in_state_reset	1	0	0	0	0	0	0	0	0		
In_data	0	0	1	1	0	0	1	0	0		
out_cur_state		0	2	4	4	5	7	7	7	7	
out		0	0	0	0	0	1	1	1	1	



FSM example

Cycle	1	2	3	4	5	6	7	8	9	10	11
in_state_reset	1	0	0	0	0	0	0	0	0	1	
In_data	0	0	1	1	0	0	1	0	0	0	
out_cur_state		0	2	4	4	5	7	7	7	7	0
out		0	0	0	0	0	1	1	1	1	0

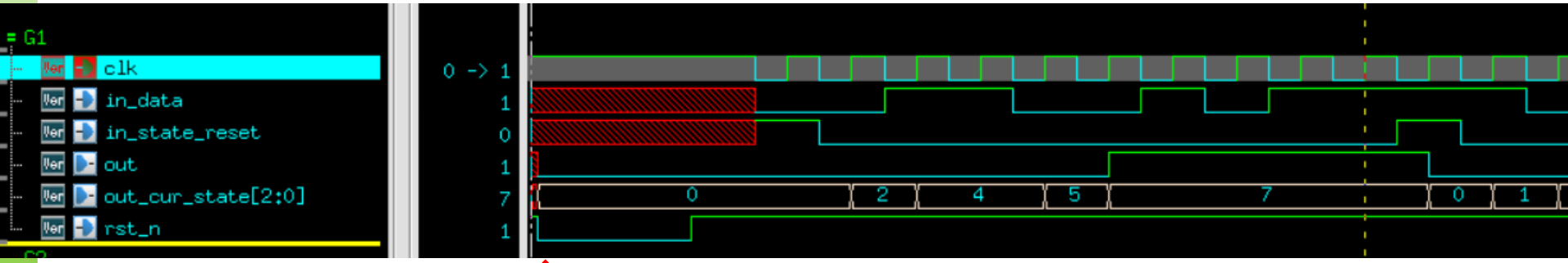


Spec

- 請使用FSM完成此次LAB，可參考pg 6。
- 所有output必須非同步負準位reset。
- 01_RTL 需要PASS。
- 02_SYN不能有error跟latches。
- 02_SYN時間timing slack必須為MET。
- 03_GATE 需要PASS

Waveform rst_n

- Waveform

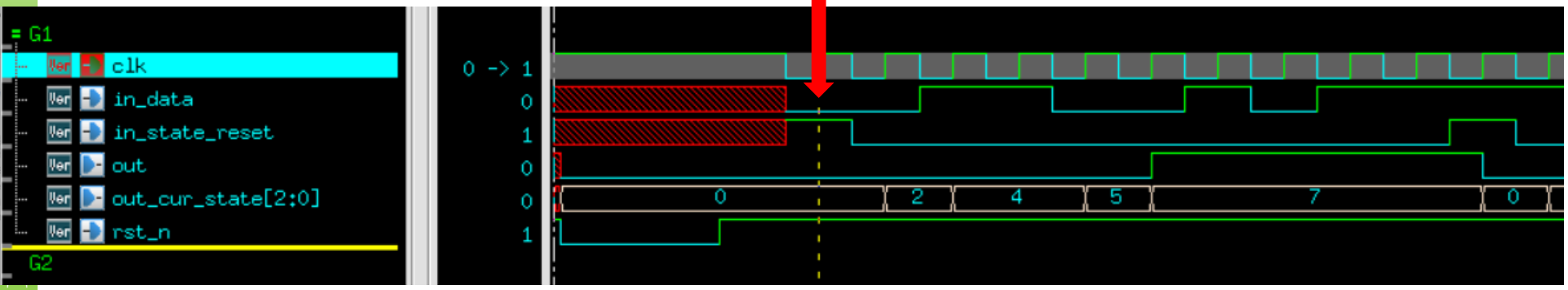


negative trigger asynchronous reset

Waveform 1st in_state_reset

- Waveform

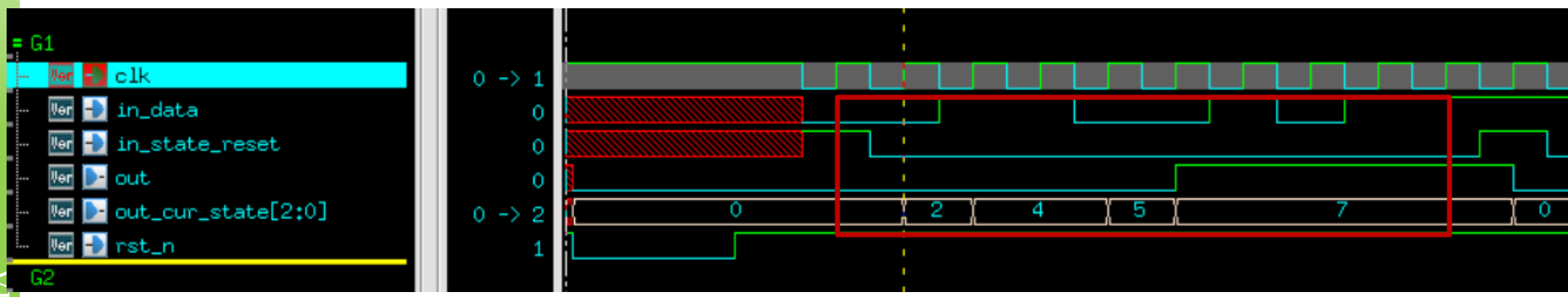
FSM reset, out_cur_state = 0, out = 0



Waveform in_data

- Waveform

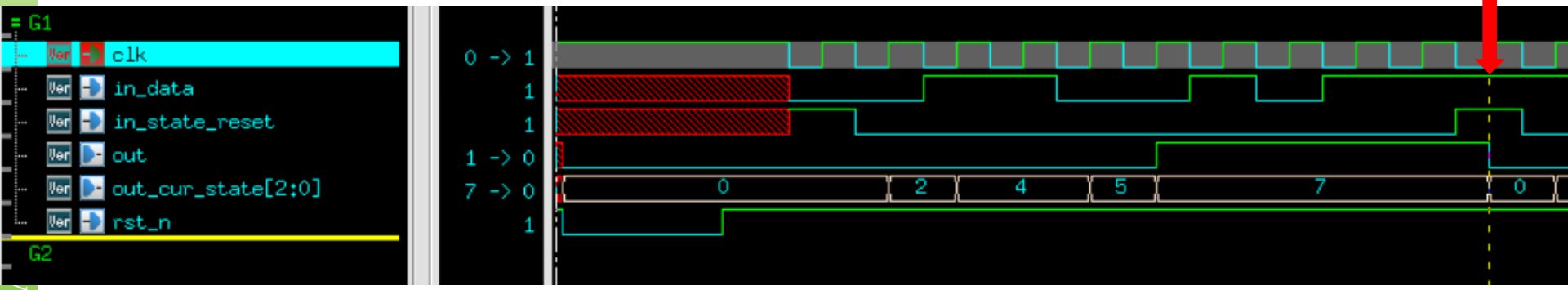
FSM，參考p7~p17



Waveform 2nd in_state_reset

- Waveform

FSM reset, out_cur_state = 0, out = 0



Command

- **不要更改FSM state parameter**

```
12  //-----------------------------------------------------------------------
13  //   INPUT AND OUTPUT DECLARATION
14  //-----------------------------------------------------------------------
15  input clk,rst_n,in_data,in_state_reset;
16  output logic [2:0] out_cur_state;
17  output logic out;
18
19
20  //-----------------------------------------------------------------------
21  //   FSM state
22  //-----------------------------------------------------------------------
23  parameter S_0 = 3'd0;
24  parameter S_1 = 3'd1;
25  parameter S_2 = 3'd2;
26  parameter S_3 = 3'd3;
27  parameter S_4 = 3'd4;
28  parameter S_5 = 3'd5;
29  parameter S_6 = 3'd6;
30  parameter S_7 = 3'd7;
31
32  //-----------------------------------------------------------------------
33  //   Your design
34  //-----------------------------------------------------------------------
35
36
37
38
39
```

Command

- tar -xvf ~dcsta01/Lab04.tar
- cd Lab04/01_RTL/
- Need 02_SYN
  - No Latch
  - No error
  - No timing violation (MET)
- Need 03_GATE
- Separate combinational and sequential blocks

Demo1: 3/24(四), 16:25:00

Demo2: 3/24(四), 23:59:59