Lab 8
Verilog – Sequential Design(3)

Johnson Liu

Department of Electrical Engineering
National Cheng Kung University

Outline

1. Finite-State Machine (FSM)
   - Moore Machine
   - Mealy Machine
2. Traffic Light
3. TA Checking & Laboratory Report
4. Reference

Recall: Blocking vs. Non-Blocking

Name	Blocking	Non-Blocking
Symbol	=	<=
Ordering	Order Sensitive	Order Independent
Always Block Type	Combinational Block	Sequential Block
LHS Synthesize Result	Wire	Register (Memory element)

Recall: Blocking vs. Non-Blocking

1. Blocking

```
initial
begin
    ...
    A = 1;
    B = 0;
    ...
    A = B;
    B = A;
```

B = 0 is used
A = 0 is used

```
initial
begin
    ...
    A = 1;
    B = 0;
    ...
    B = A;
    A = B;
```

A = 1 is used
B = 1 is used

---

2. Non-Blocking

```
initial
begin
    ...
    A <= 1;
    B <= 0;
    ...
    A <= B;
    B <= A;
```

B = 0 is used
A = 1 is used

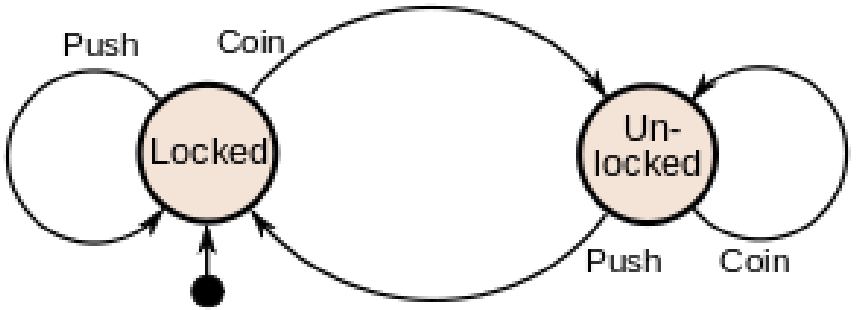
```
initial
begin
    ...
    A <= 1;
    B <= 0;
    ...
    B <= A;
    A <= B;
```

A = 1 is used
B = 0 is used

Finite-State Machine (FSM)

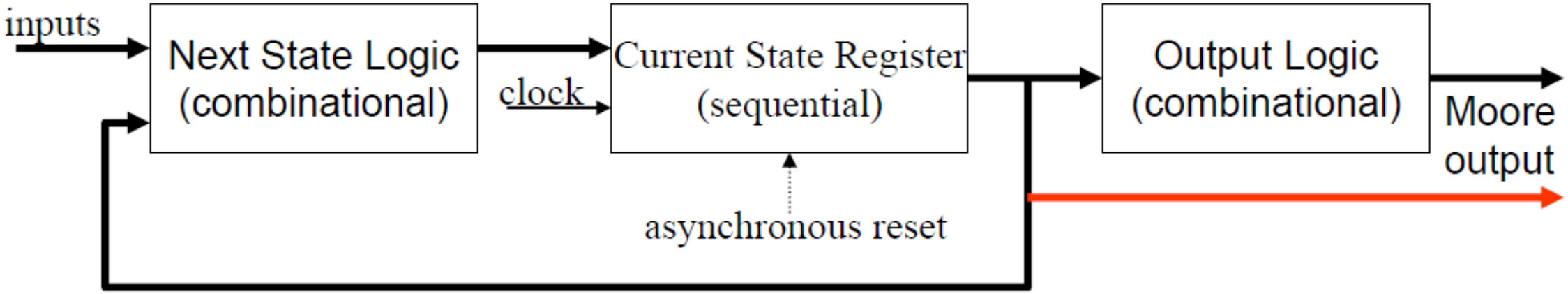
Finite-State Machine(FSM)

- 在進行控制電路(Controller)的設計時，可以使用FSM的方式設計。
- 將電路的行為切分為數個**state**，並定義每個state的行為以及state間轉換的條件。
- 可分為Moore & Mealy 兩種machine。



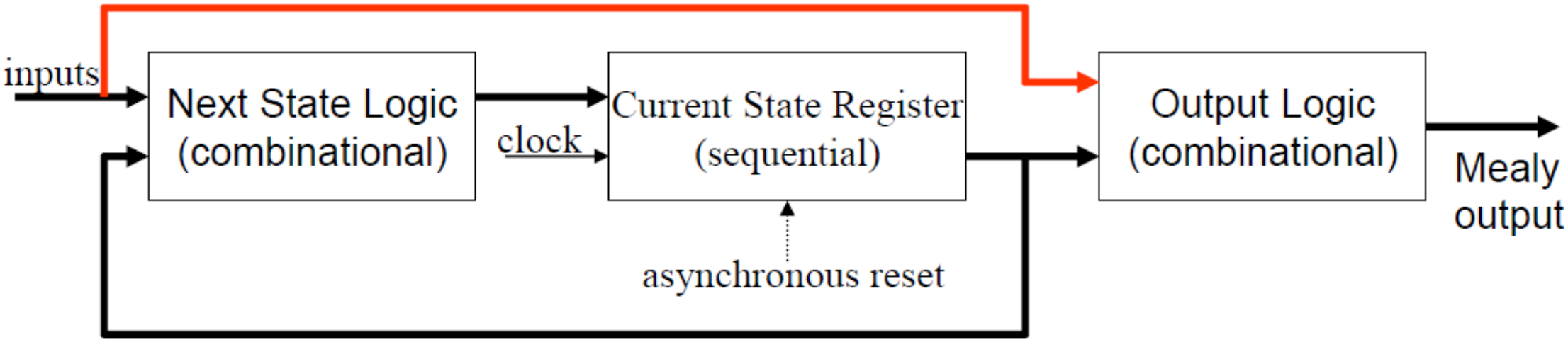
Moore vs Mealy

- Moore machine : Output只由當前的state決定。



Moore vs Mealy

- Mealy machine : Output由當前的state和input決定。

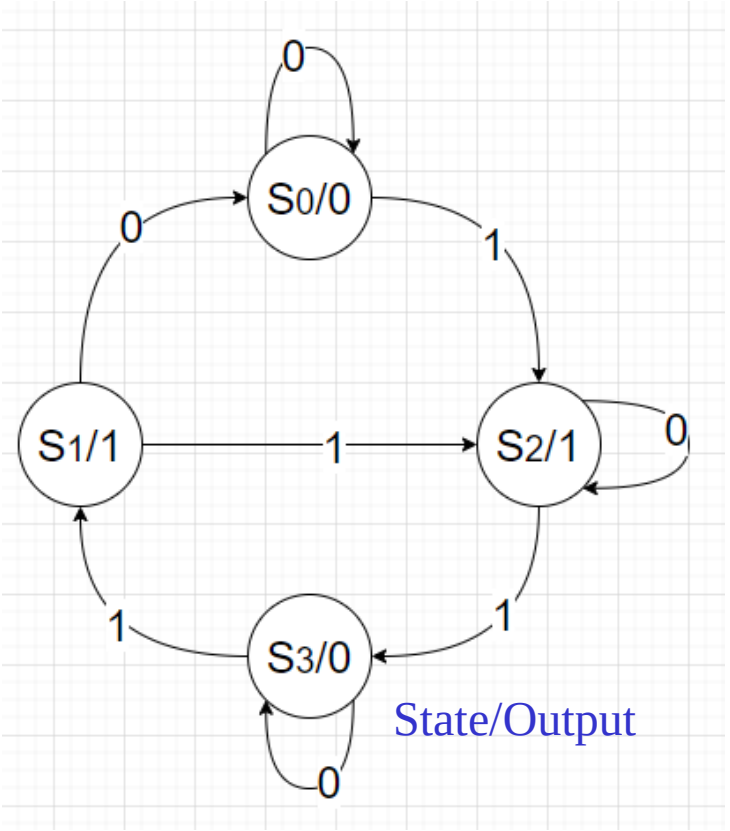


- **Verilog Example**

Moore Machine

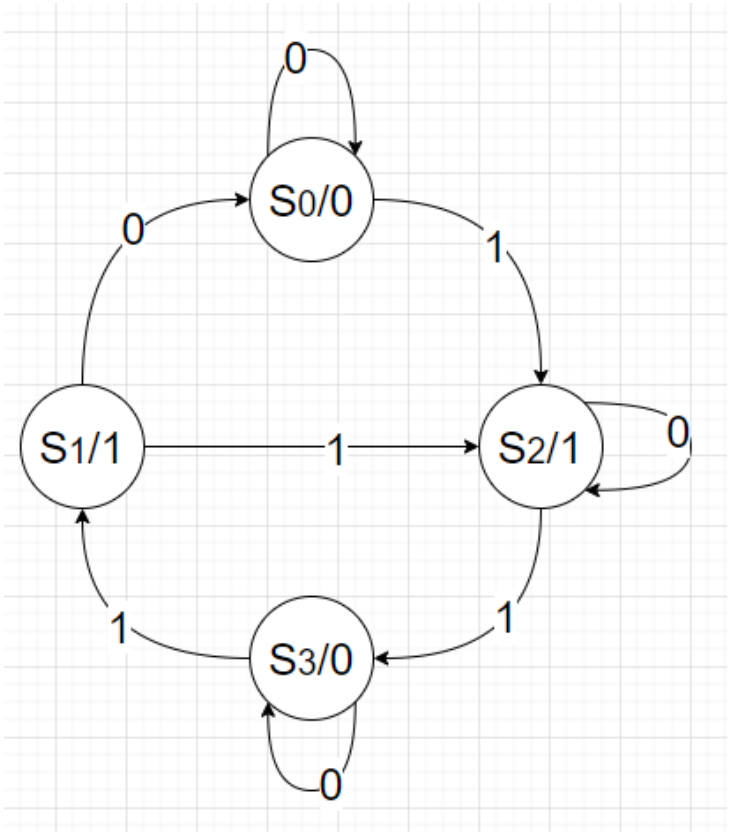
- Moore machine is a finite-state machine whose output values are determined only by its current state.
- Moore machine: S → O (S : state, O: output)

Present State	Next State		Output	
	I=0	I=1	I=0	I=1
S_0	S_0	S_2	0	0
S_1	S_0	S_2	1	1
S_2	S_2	S_3	1	1
S_3	S_3	S_1	0	0



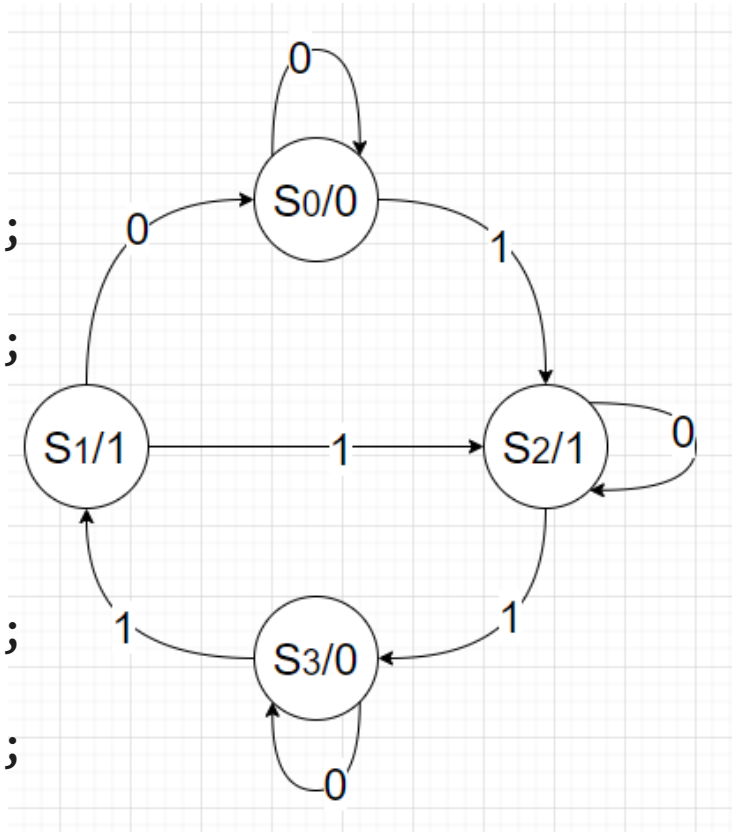
Moore - State Register (Sequential)

```
module moore(clk, rst, In_Data, Out_Data);
    input clk, rst, In_Data;
    output [1:0] Out_Data;
    reg [1:0] Out_Data;
    reg [1:0] State, NextState;
    parameter S0 = 2'b00, S1 = 2'b01,
              S2 = 2'b10, S3 = 2'b11;
    // State Register (Flip-Flops)
    always @(posedge clk or posedge rst)
    begin
        if(rst)
            State <= S0;
        else
            State <= NextState;
    end
```



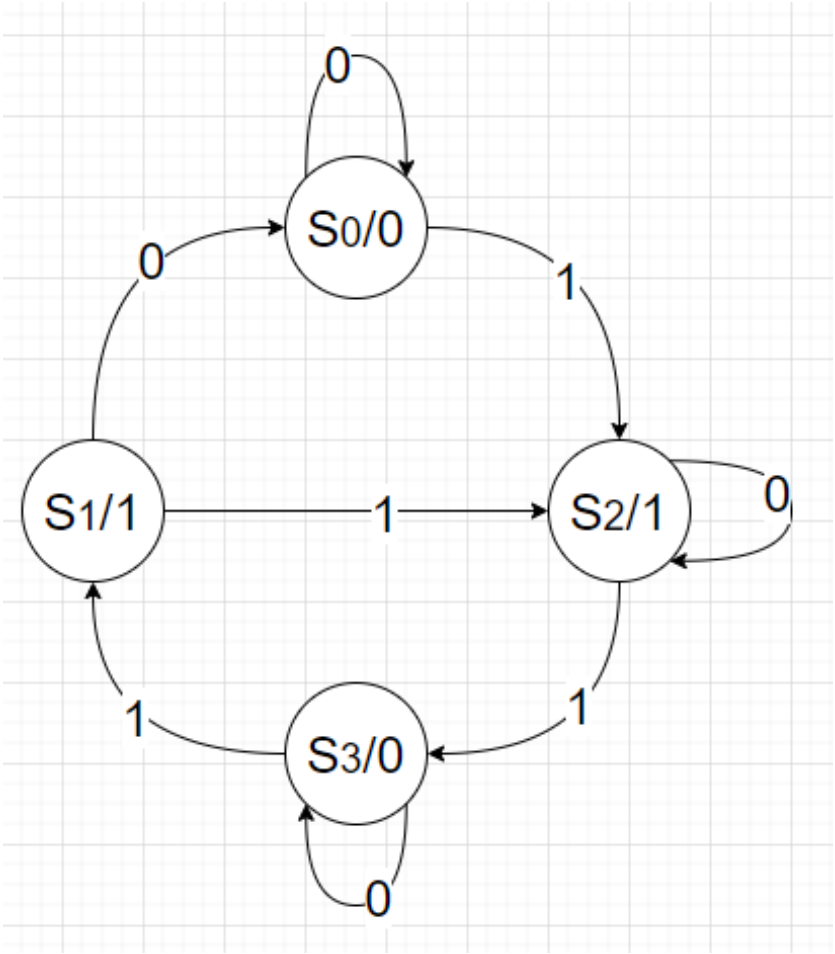
Moore - Next State Logic (Combinational)

```
// Next State Logic
always @(In_Data or State)
begin
    case(State)
        S0:
        begin
            if(In_Data == 1)
                NextState = S2;
            else
                NextState = S0;
        end
        S1:
        begin
            if(In_Data == 1)
                NextState = S2;
            else
                NextState = S0;
            end
        S2:
        begin
            if(In_Data == 1)
                NextState = S3;
            else
                NextState = S2;
        end
        S3:
        begin
            if(In_Data == 1)
                NextState = S1;
            else
                NextState = S3;
        end
    endcase
```



Moore - Output Logic (Combinational)

```
    // Output Logic
    always @(State)
    begin
        case(State)
            S0:Out_Data = 0;
            S1:Out_Data = 1;
            S2:Out_Data = 1;
            S3:Out_Data = 0;
        endcase
    end
endmodule
```

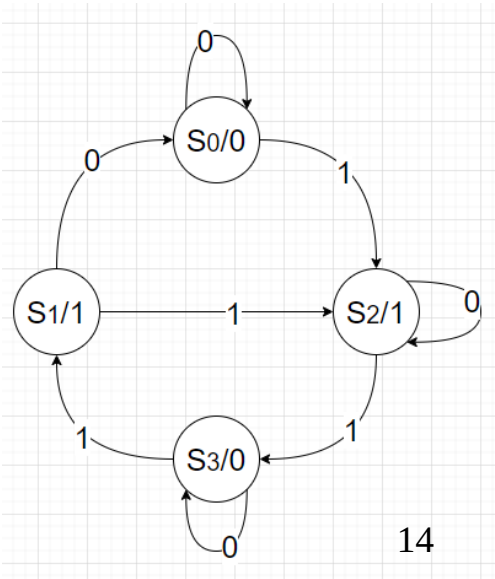


Moore - Bad Example (All Sequential)

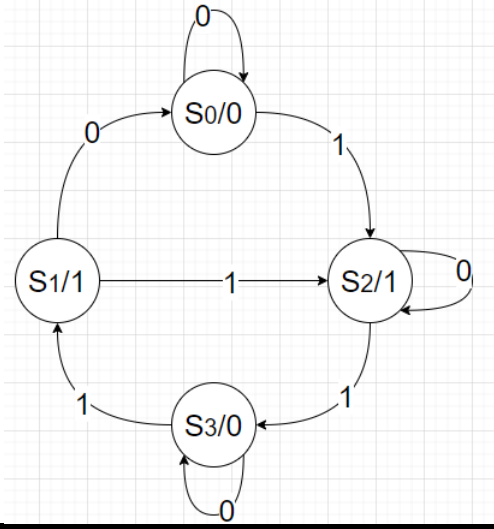
```
module moore_bad(clk, rst, In_Data, Out_Data);
    input clk, rst, In_Data;
    output [1:0] Out_Data;
    reg [1:0] Out_Data;
    reg [1:0] State;
    parameter S0 = 2'b00, S1 = 2'b01,
              S2 = 2'b10, S3 = 2'b11;
    always @(posedge clk)
    begin
        if(rst)
        begin
            State <= S0;
        end
        else
        begin
            case(State)
                S0:
                begin
                    Out_Data <= 0;
                    if(In_Data == 1)
                        State <= S2;
                    else
                        State <= S0;
                end
```

```
                S1:
                begin
                    Out_Data <= 1;
                    if(In_Data == 1)
                        State <= S2;
                    else
                        State <= S0;
                end
                S2:
                begin
                    Out_Data <= 1;
                    if(In_Data == 1)
                        State <= S3;
                    else
                        State <= S2;
                end
```

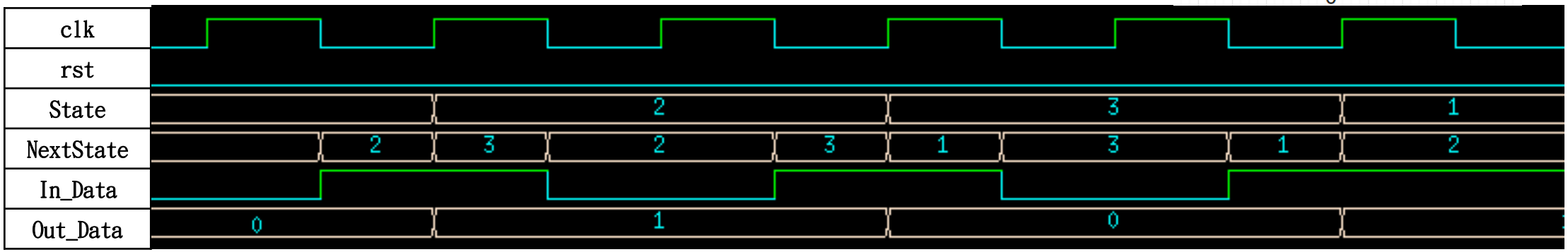
```
                S3:
                begin
                    Out_Data<= 0;
                    if(In_Data == 1)
                        State <= S1;
                    else
                        State <= S3;
                end
```



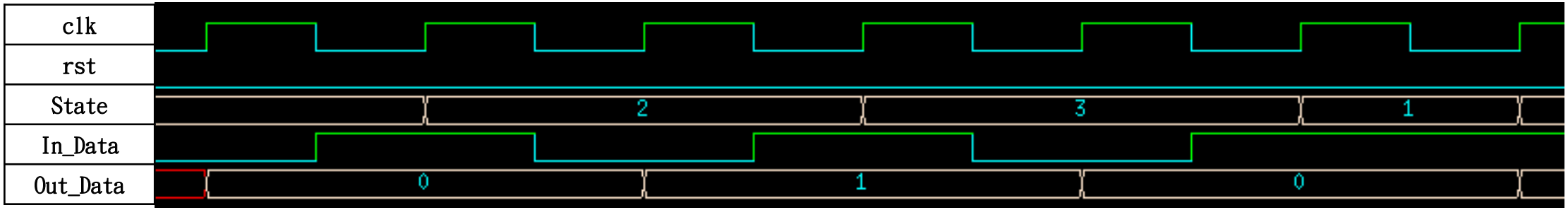
Moore Waveform



- Good Moore



- Bad Moore Output delay了1個cycle

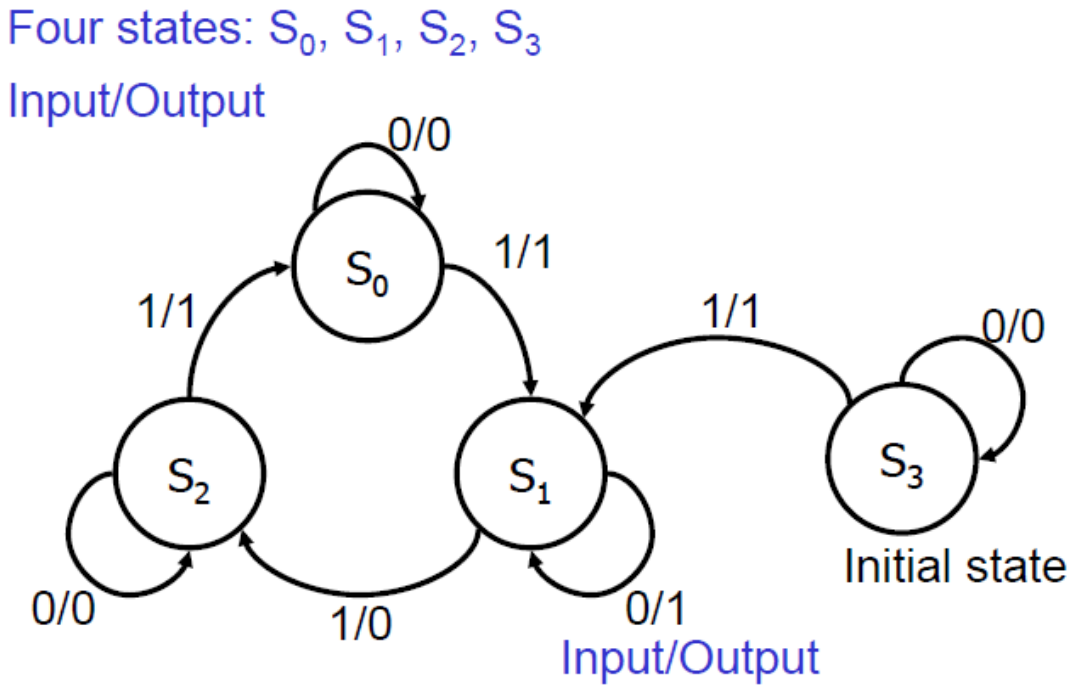


- **Mealy Machine**

Mealy Machine

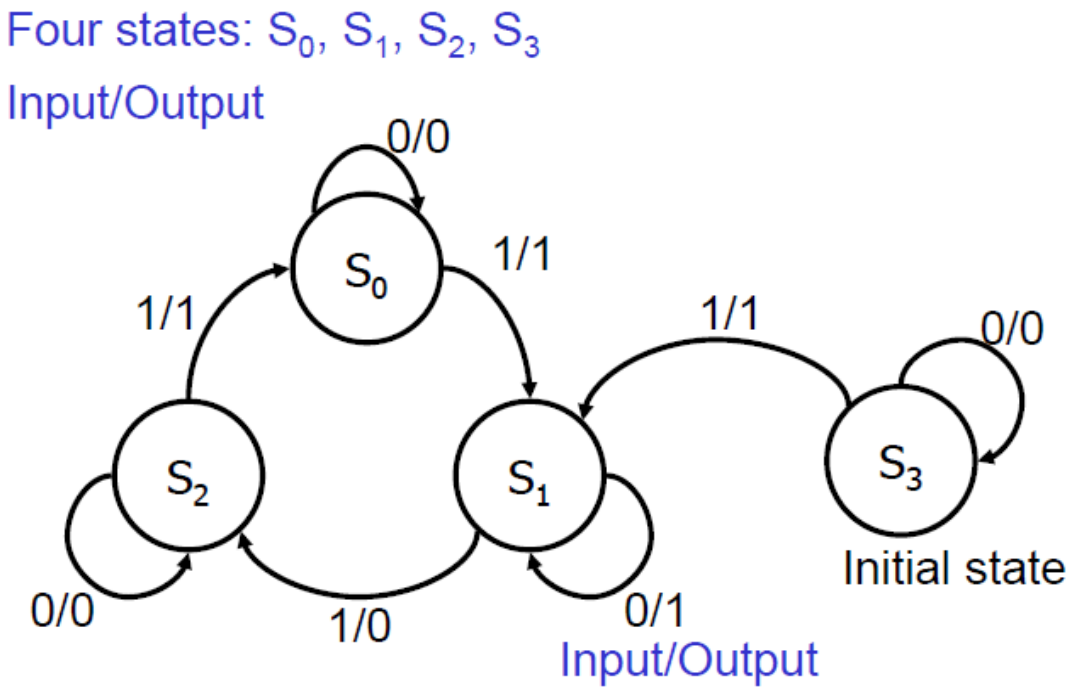
- Mealy machine is a finite-state machine whose output values are determined both by its current state and the current inputs.
- Mealy machine: $S \times I \rightarrow O$ (S : state, I: Input, O: output)

Present State	Next State		Output	
	I=0	I=1	I=0	I=1
S_0	S_0	S_1	0	1
S_1	S_1	S_2	1	0
S_2	S_2	S_0	0	1
S_3	S_3	S_1	0	1



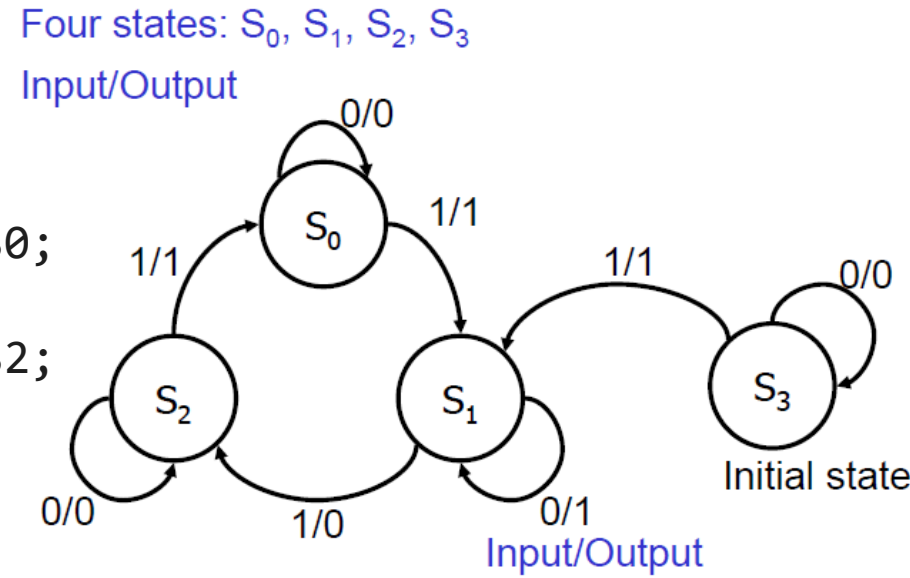
Mealy - State Register (Sequential)

```
module mealy(clk, rst, In_Data, Out_Data);
    input clk, rst, In_Data;
    output [1:0] Out_Data;
    reg [1:0] Out_Data;
    reg [1:0] State, NextState;
    parameter S0 = 2'b00, S1 = 2'b01,
              S2 = 2'b10, S3 = 2'b11;
    // State Register (Flip-Flops)
    always @(posedge clk or posedge rst)
    begin
        if(rst)
            State <= S3;
        else
            State <= NextState;
    end
```



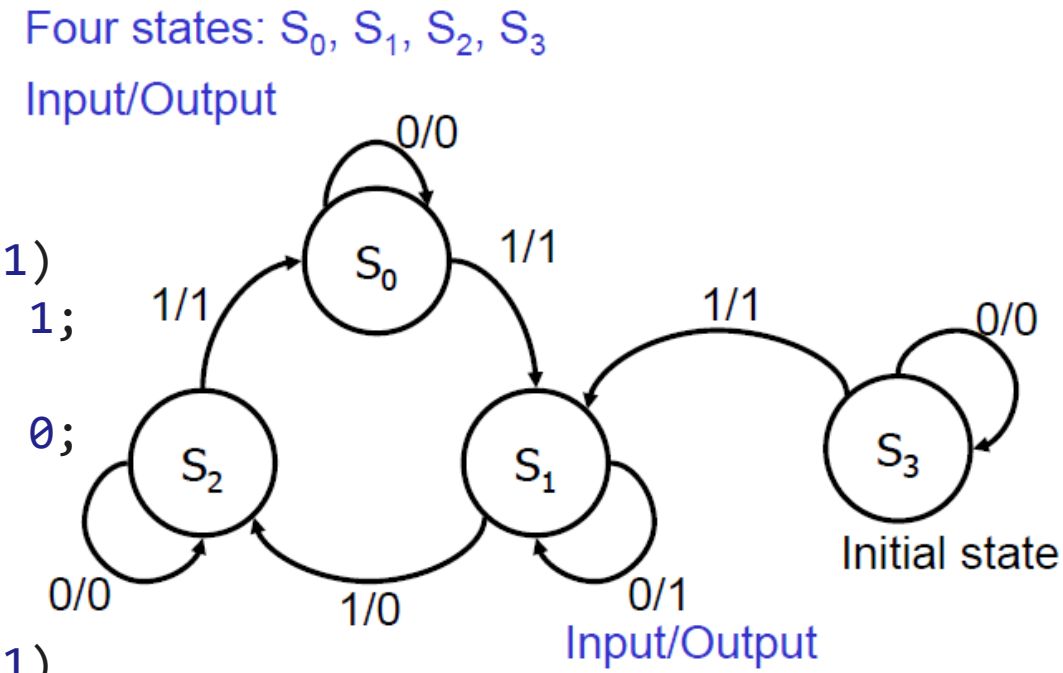
Mealy - Next State Logic (Combinational)

```
// Next State Logic
always @(In_Data or State)
begin
    case(State)
        S0:
        begin
            if(In_Data == 1)
                NextState = S1;
            else
                NextState = S0;
        end
        S1:
        begin
            if(In_Data == 1)
                NextState = S2;
            else
                NextState = S1;
            end
        S2:
        begin
            if(In_Data == 1)
                NextState = S0;
            else
                NextState = S2;
        end
        S3:
        begin
            if(In_Data == 1)
                NextState = S1;
            else
                NextState = S3;
        end
    endcase
```

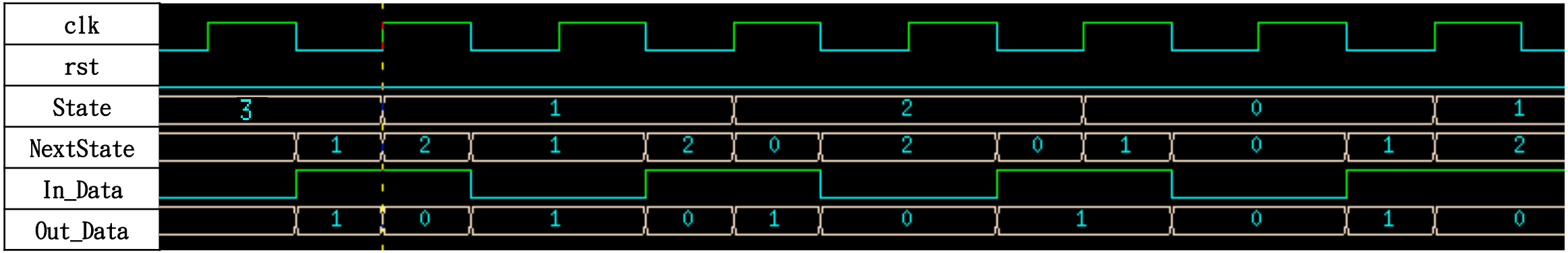


Mealy - Output Logic (Combinational)

```
// Output Logic
always @(In_Data or State)
begin
    case(State)
        S0:
        begin
            if(In_Data == 1)
                Out_Data = 1;
            else
                Out_Data = 0;
        end
        S1:
        begin
            if(In_Data == 1)
                Out_Data = 0;
            else
                Out_Data = 1;
        end
        S2:
        begin
            if(In_Data == 1)
                Out_Data = 1;
            else
                Out_Data = 0;
        end
        S3:
        begin
            if(In_Data == 1)
                Out_Data = 1;
            else
                Out_Data = 0;
        end
    endcase
```

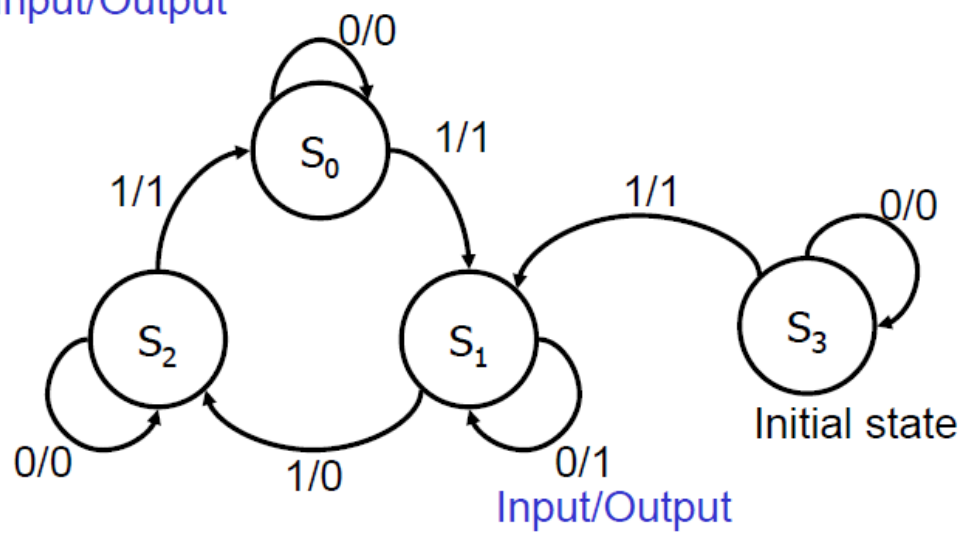


Mealy Waveform



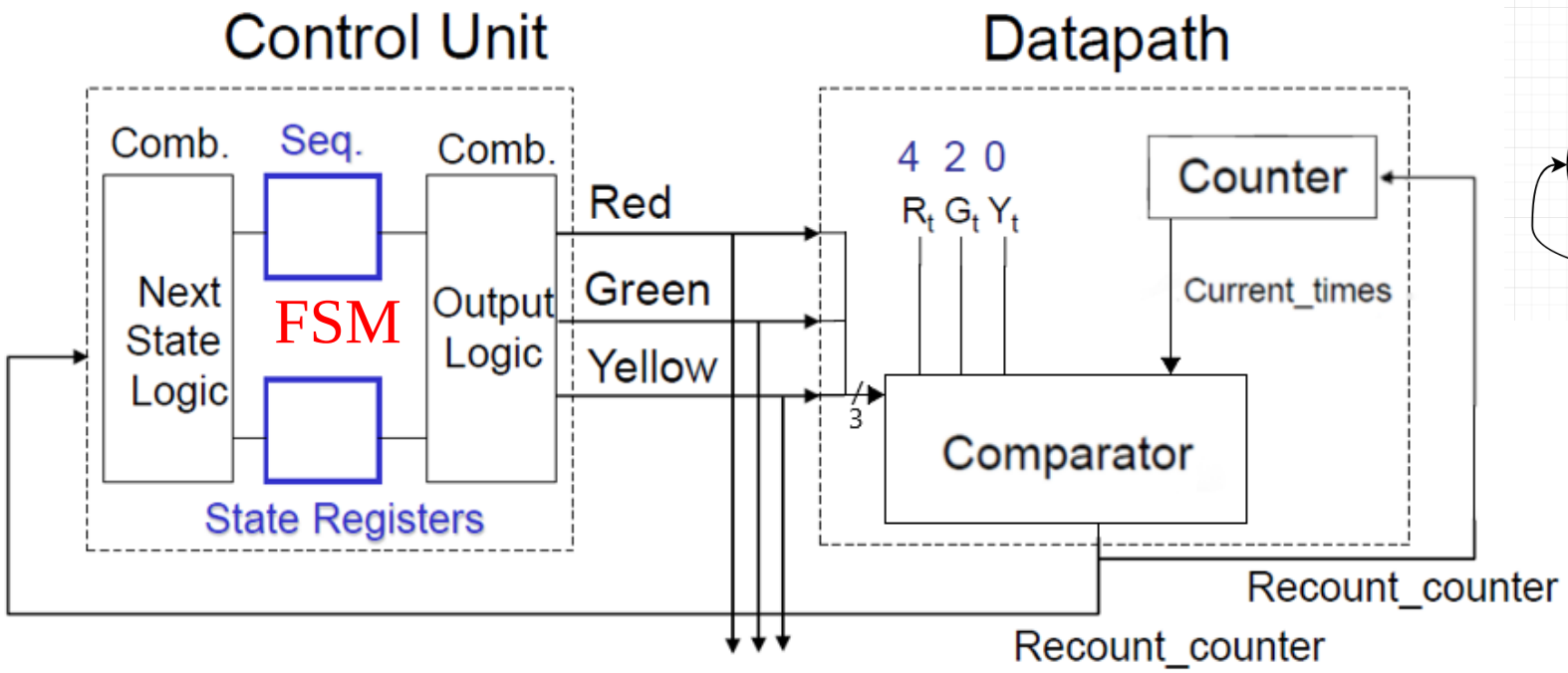
Four states: S_0, S_1, S_2, S_3

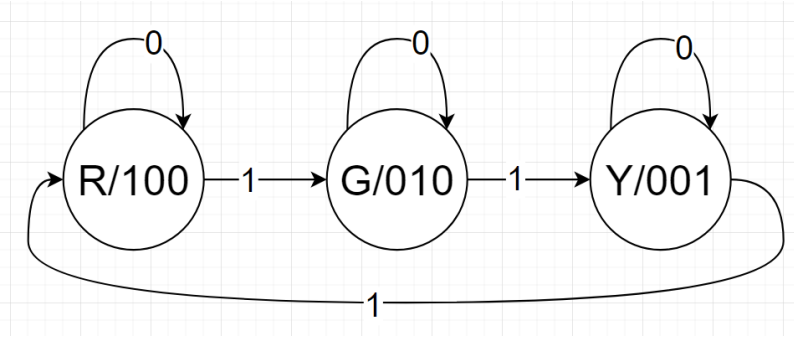
Input/Output



Example : Traffic Light

Traffic Light Circuit

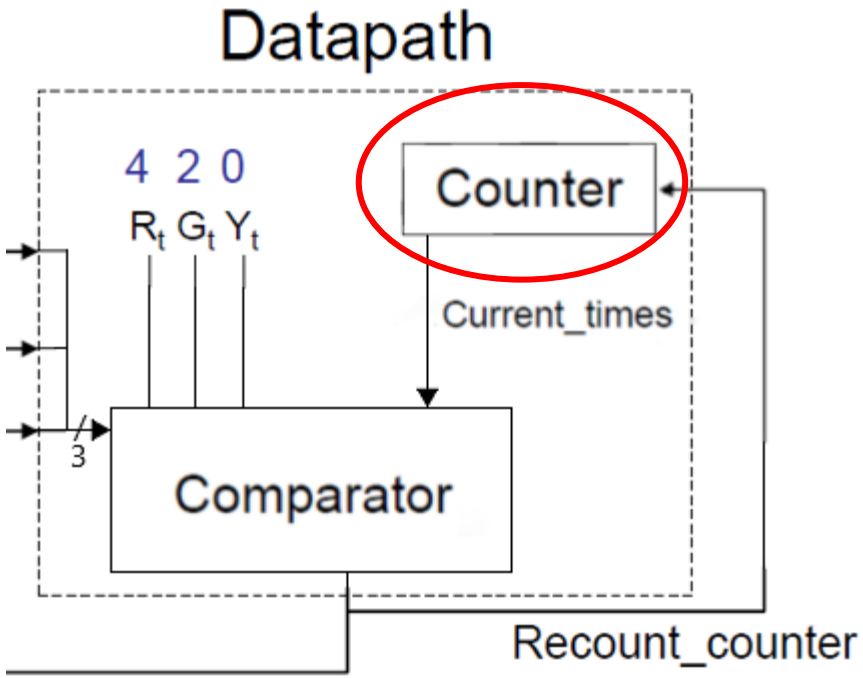




R_time: 4+1=5 cycles
G_time: 2+1=3 cycles
Y_time: 0+1=1 cycles

Datapath—Counter (Sequential)

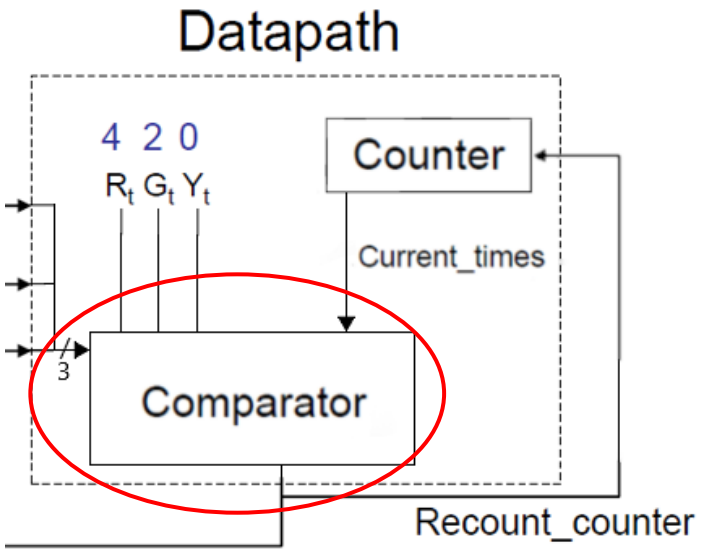
```
module Counter(clk, rst, Recount_Counter, Count_Out);
    input clk, rst, Recount_Counter;
    output [3:0] Count_Out;
    reg [3:0] Count_Out;
    always @(posedge clk)
    begin
        if(rst)
        begin
            Count_Out <= 0;
        end
        else
        begin
            if(Recount_Counter)
                Count_Out <= 0;
            else
                Count_Out <= Count_Out + 1;
        end
```



Datapath—Comparator (Combinational)

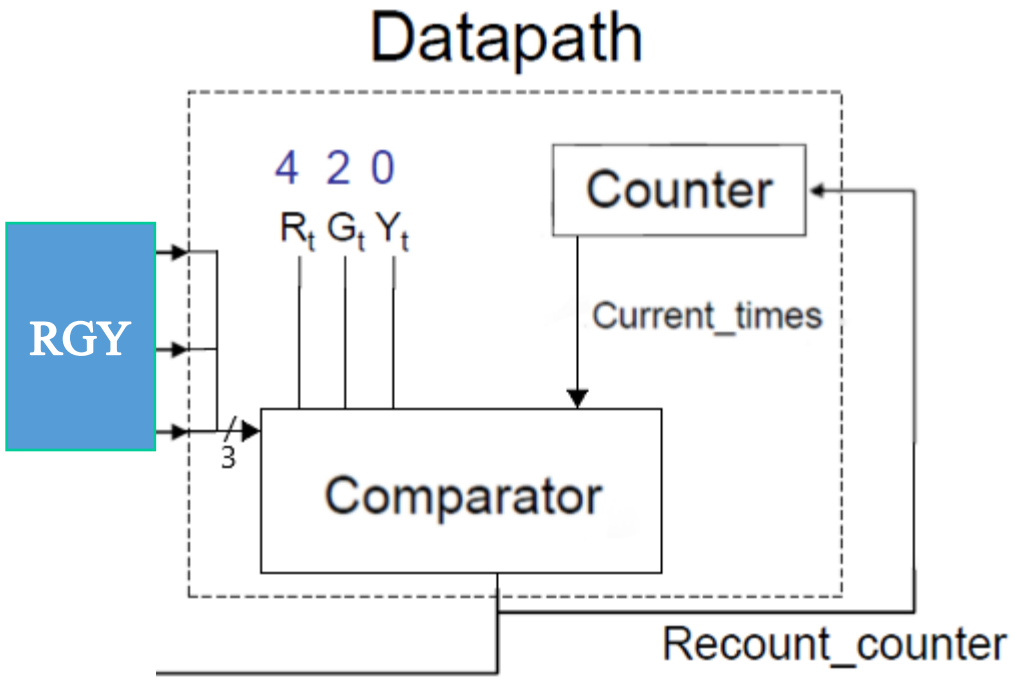
```
module Compare(current_times, RGY, Recount_counter);
    input [2:0] RGY;
    input [3:0] current_times;
    output Recount_counter;
    reg Recount_counter;
    parameter R_times = 4,
              G_times = 2,
              Y_times = 0;

    always @(*)
    begin
        case(RGY)
            3'b100:
            begin
                if(current_times == R_times)
                    Recount_counter = 1;
                else
                    Recount_counter = 0;
            end
            3'b001:
            begin
                if(current_times == Y_times)
                    Recount_counter = 1;
                else
                    Recount_counter = 0;
            end
            3'b010:
            begin
                if(current_times == G_times)
                    Recount_counter = 1;
                else
                    Recount_counter = 0;
            end
            default:
                Recount_counter = 1;
        endcase
```



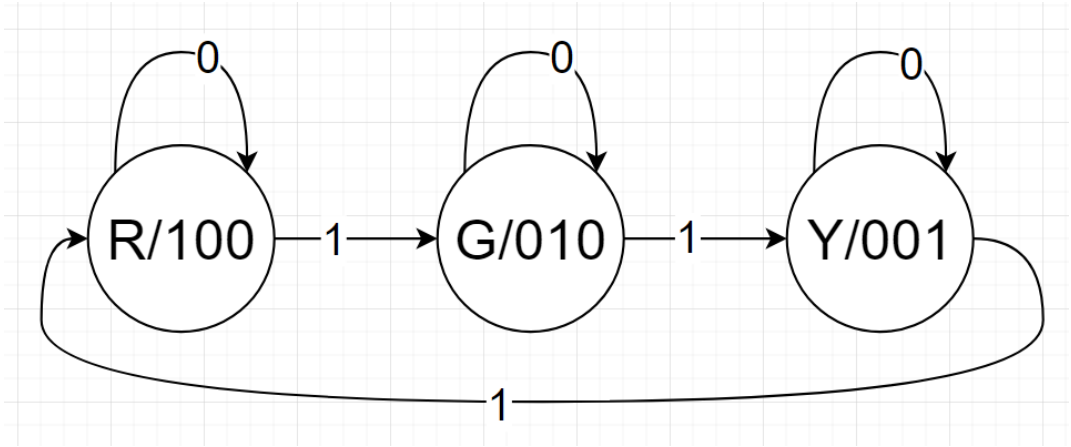
Datapath

```
module Datapath(clk, rst, RGY, Recount);
    input clk, rst;
    input [2:0] RGY;
    output Recount;
    wire [3:0] Current_times;
    // Module
    Compare compare
    (
        .current_times(Current_times),
        .RGY(RGY),
        .Recount_counter(Recount)
    );
    Counter counter
    (
        .clk(clk),
        .rst(rst),
        .Recount_Counter(Recount),
        .Count_Out(Current_times)
    );
endmodule
```



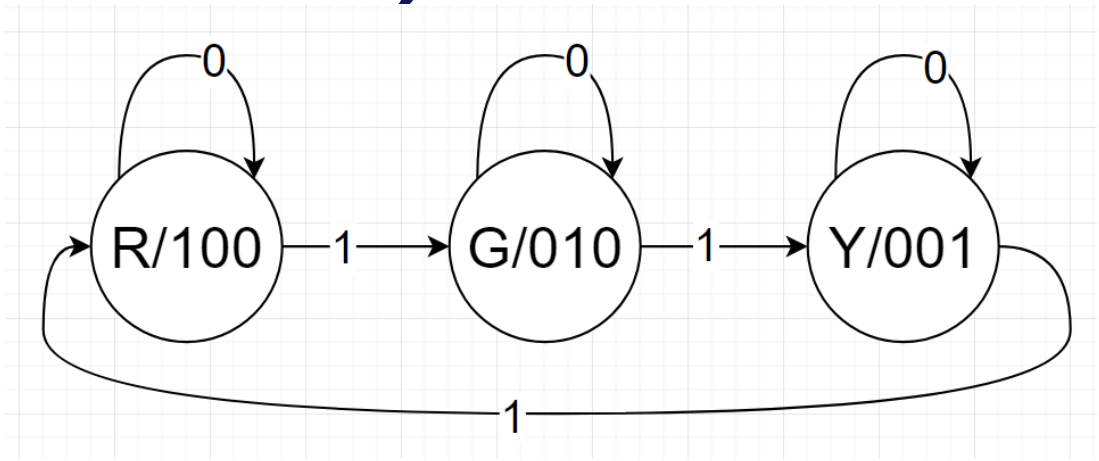
Control Unit—State Register (Sequential)

```
module Traffic_Control (clk, rst, Recount_Counter, Red, Green, Yellow);
    input clk, rst, Recount_Counter;
    output Red, Green, Yellow;
    reg Red, Green, Yellow;
    reg [1:0] currentstate, nextstate;
    parameter [1:0] Red_Light = 0,
                    Green_Light = 1,
                    Yellow_Light = 2;
    // State Register (Flip-Flops)
    always @(posedge clk)
    begin
        if(rst)
            currentstate <= Red_Light;
        else
            currentstate <= nextstate;
    end
```



Control Unit—Next State Logic (Combinational)

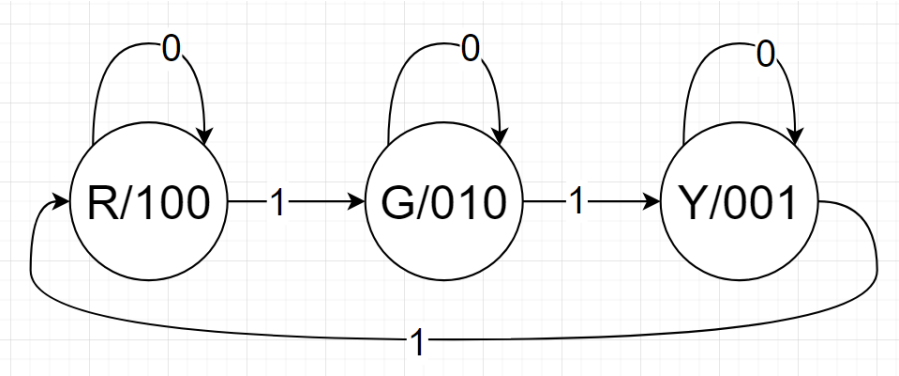
```
// Next State Logic
always @(*)
begin
    case(currentstate)
        Red_Light:
        begin
            if(Recount_Counter)
                nextstate = Green_Light;
            else
                nextstate = Red_Light;
        end
        Green_Light:
        begin
            if(Recount_Counter)
                nextstate = Yellow_Light;
            else
                nextstate = Green_Light;
        end
```




```
        Yellow_Light:
        begin
            if(Recount_Counter)
                nextstate = Red_Light;
            else
                nextstate = Yellow_Light;
        end
        default:
            nextstate = Red_Light;
    endcase
end
```

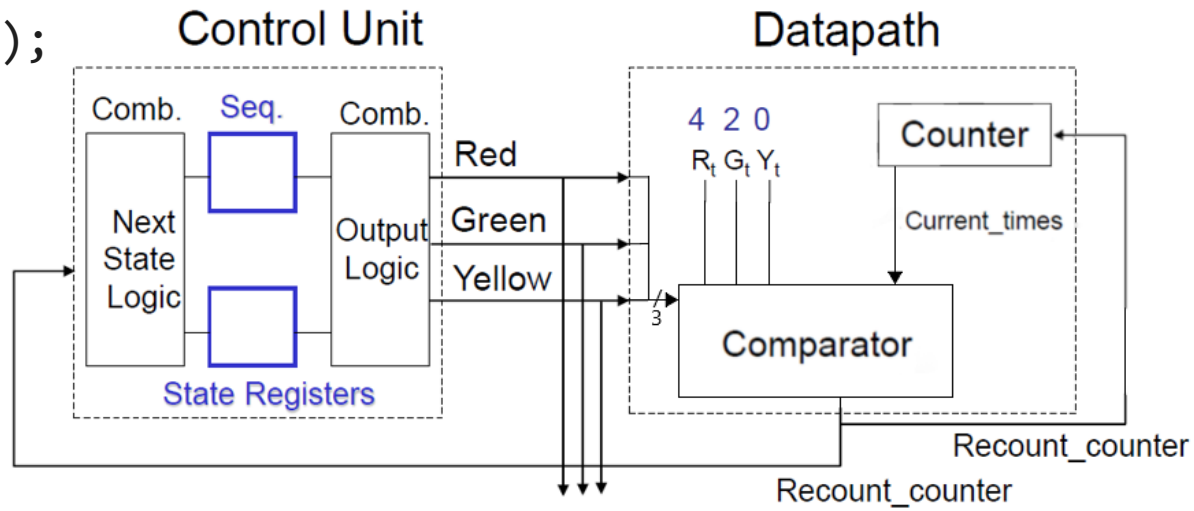
Control Unit—Output Logic (Combinational)

```
// Output Logic
always @(currentstate)
begin
    case(currentstate)
        Red_Light:
        begin
            Red = 1'b1;
            Green = 1'b0;
            Yellow = 1'b0;
        end
        Green_Light:
        begin
            Red = 1'b0;
            Green = 1'b1;
            Yellow = 1'b0;
        end
        Yellow_Light:
        begin
            Red = 1'b0;
            Green = 1'b0;
            Yellow = 1'b1;
        end
        default:
        begin
            Red = 1'b0;
            Green = 1'b0;
            Yellow = 1'b0;
        end
    endcase
end
```



Combine Control Unit & Datapath

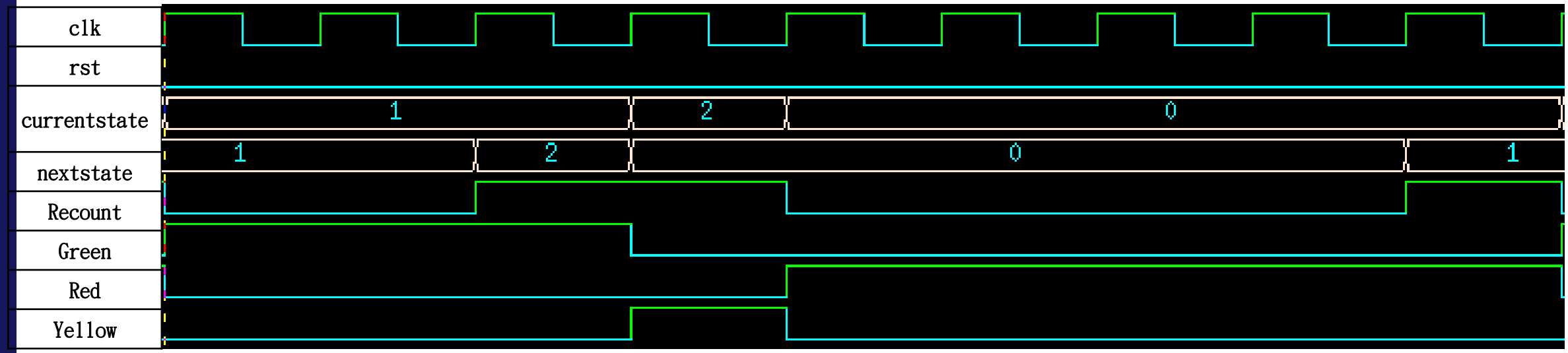
```
module traffic(clk, rst, Red, Green, Yellow);
    input clk, rst;
    output Red, Green, Yellow;
    wire Recount_counter;
    // Module
    Traffic_Control controller
    (
        .clk(clk),
        .rst(rst),
        .Recount_Counter(Recount_counter),
        .Red(Red),
        .Green(Green),
        .Yellow(Yellow)
    );
```



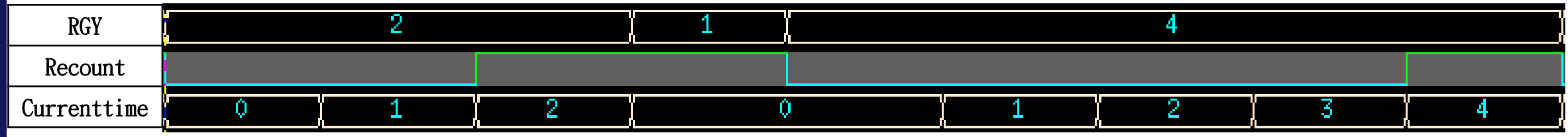

```
    Datapath datapath
    (
        .clk(clk),
        .rst(rst),
        .RGY({Red,Green,Yellow}),
        .Recount(Recount_counter)
    );
endmodule
```

Traffic Light Waveform

- Traffic Light



- Comparator



Implementation(1)
Microwave Oven

Microwave Oven

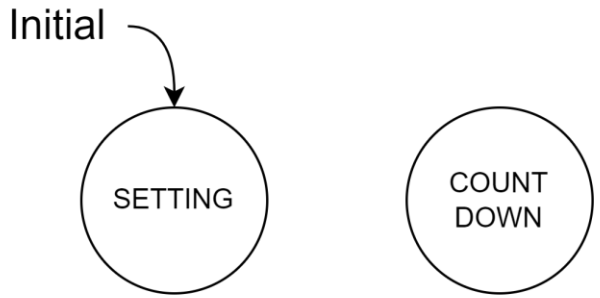
- A microwave oven is an electric oven that heats and cooks food by exposing it to electromagnetic radiation in the microwave frequency range.
- This induces polar molecules such as water in the food to rotate and produce thermal energy in a process known as electronic heating.
- The first microwave oven was created by Percy Spencer who was hired by Raytheon in 1945. He was at that time one of the world's leading experts in radar tube design. When he was working on powered radar set, he noticed that a chocolate bar that he had in his pocket was melting.
- Mechanisms
  Heat the food by microwave within the fixed time.

Implement Microwave Oven by State Machine

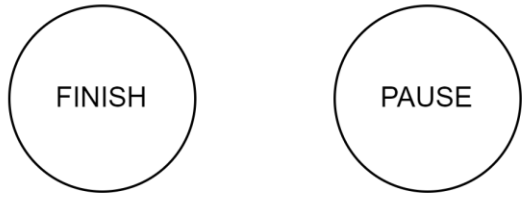
- Create a 10-bit register to store Count Down Time.
- Only one button work each time. If more than one button been activated, the button with highest priority will be chosen.
- Try to use following 4 States Moore Machine Template to create your own Finite State Machine. (By adding new transition to the template)

- 6 buttons on the control panel

Button Name	Priority
10 Minutes	1
1 Minute	2
10 Seconds	3
1 Second	4
Start	5
Pause	6



- 4 States Moore Machine Template



Implement Microwave Oven by State Machine

- *Setting*

  Set Count Down Time by Press ***10 Minutes***, ***1 Minute***, ***10 Seconds***, or ***1 Second*** Button. Press ***Pause*** Button to reset Count Down Time to 0. Press ***Start*** Button to start microwave food and count down. It is not necessary to microwave food and count down, if the Count Down Time is 0.

- *Count Down*

  Microwave food and count down one second each cycle. Press ***Pause*** to pause count down. It is not necessary to pause count down, if the Count Down Time is 0.

- *Pause*

  Pause count down. Press ***Pause*** to stop microwave and reset Count Down Time to 0. Press ***Start*** to resume microwave food and count down.

- *Finish*

  Notify user microwave has finished by one cycle, then resume setting.

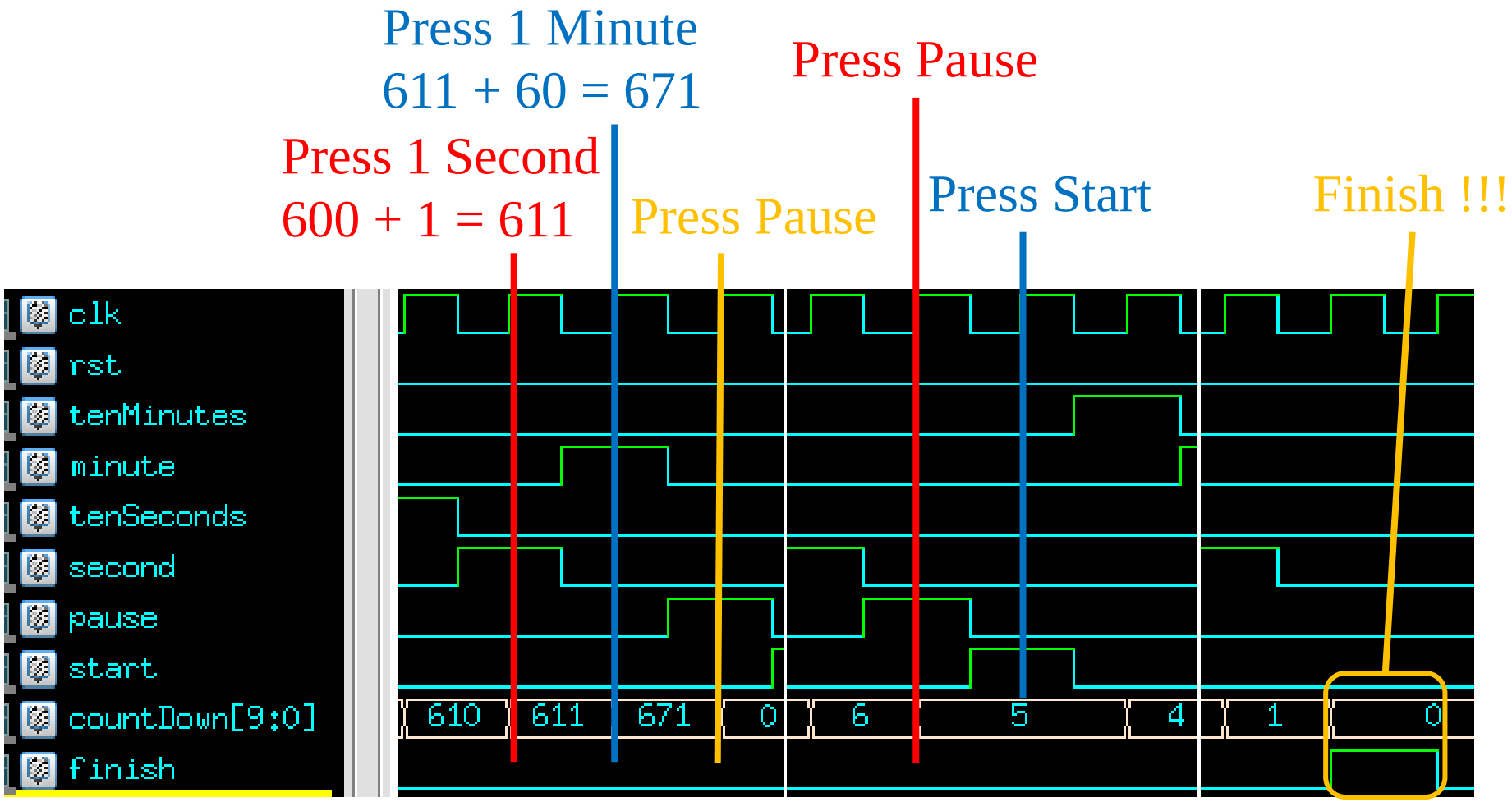
Implement Microwave Oven by State Machine

- I/O Interface of Module Microwave in Microwave.v

Name	I/O	Width	Description
clk	I	1	System clock signal. This system is synchronized with the positive edge of the clock.
rst	I	1	Active-high asynchronous reset signal.
tenMinutes	I	1	High when 10 Minutes Button is pressed.
minute	I	1	High when 1 Minute Button is pressed.
tenSeconds	I	1	High when 10 Seconds Button is pressed.
second	I	1	High when 1 Second Button is pressed.
pause	I	1	High when Pause Button is pressed.
start	I	1	High when Start Button is pressed.
countDown	O	10	Show the Count Down Time.
finish	O	1	Show microwave has finished.

Implement Microwave Oven by State Machine

- Waveform of Microwave Oven



Hint.

- 使用講義的state registers, next-state logic, output logic的範例進行改寫。(Moore machine)
- 使用if-else來設定按鈕的priority.
- 在setting state按下pause會reset倒數，在count down state按下則會暫停。

Implementation(2): Microwave Oven with Door

Implement Microwave Oven with Door

- Microwave Oven without door is very dangerous, because user would be microwaved and get burned.
- Microwave Oven with Door inherits Implementation(1)'s functions.
- Try to reuse Finite State Machine you created in Implementation(1), and add some constraints to transition.
- Setting
  The ***Start*** Button is valid only when the ***Door*** is closed.
- Count Down
  Microwave food and count down can only proceed with the ***Door*** being closed. If the ***Door*** is opened, the microwave oven will be forced to pause.
- Pause
  The ***Start*** Button is valid only when the ***Door*** is closed.

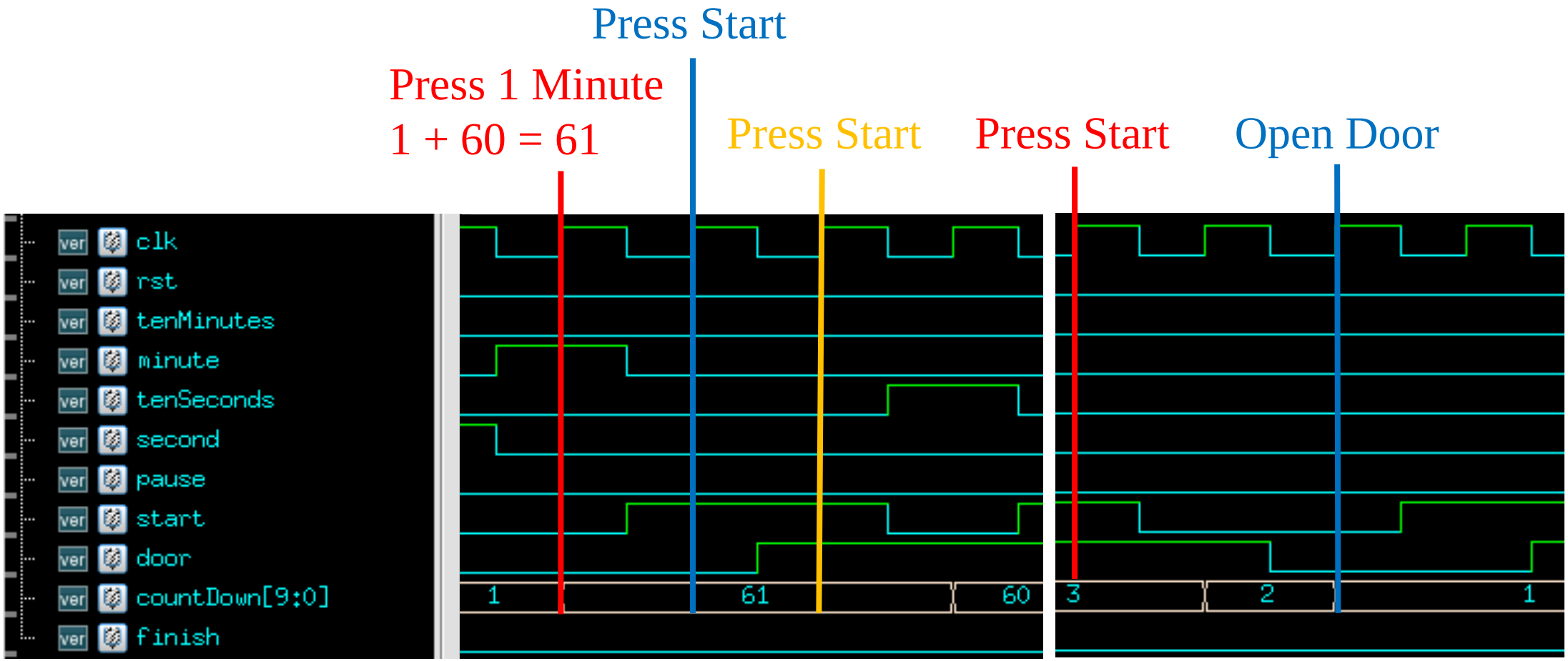
Implement Microwave Oven with Door

- I/O Interface of Module Microwave in Microwave.v

Name	I/O	Width	Description
clk	I	1	System clock signal. This system is synchronized with the positive edge of the clock.
rst	I	1	Active-high asynchronous reset signal.
tenMinutes	I	1	High when 10 Minutes Button is pressed.
minute	I	1	High when 1 Minute Button is pressed.
tenSeconds	I	1	High when 10 Seconds Button is pressed.
second	I	1	High when 1 Second Button is pressed.
pause	I	1	High when Pause Button is pressed.
start	I	1	High when Start Button is pressed.
door	I	1	High when Door is closed.
countDown	O	10	Show the Count Down Time.
finish	O	1	Show microwave has finished.

Implement Microwave Oven with Door

- Waveform of Microwave Oven



TA Checking
&
Laboratory Report

TA Checking & Laboratory Report

- TA Checking
  1. Implementation(1)’s Code & Pass Result
  2. Implementation(2)’s Code & Pass Result
- Laboratory Report
  1. Implementation(1)’s Finite State Machine and waveform
  2. Implementation(2)’s Finite State Machine and waveform
  3. How do you think about this time’s laboratory class?

Reference

Reference

- Finite-State Machine (FSM): https://reurl.cc/ye4gyM
- Verilog HDL Design: https://pse.is/3rypr5

Thank for Listening