

NATIONAL UNIVERSITY OF HO CHI MINH CITY
UNIVERSITY OF INFORMATION TECHNOLOGY
FACULTY OF COMPUTER ENGINEERING

LECTURE

Subject:

VERILOG Hardware Design Language

Chapter7: State Machine

Lecturer: Lam Duc Khai



Agenda

1. Chapter 1: Introduction
2. Chapter 2: Fundamental concepts
3. Chapter 3: Modules and hierarchical structure
4. Chapter 4: Primitive Gates – Switches – User defined primitives
5. Chapter 5: Structural model
6. Chapter 6: Behavioral model – Combination circuit & Sequential circuit
7. Chapter 7: State machines
8. Chapter 8: Testbench and verification
9. Chapter 9: Tasks and Functions



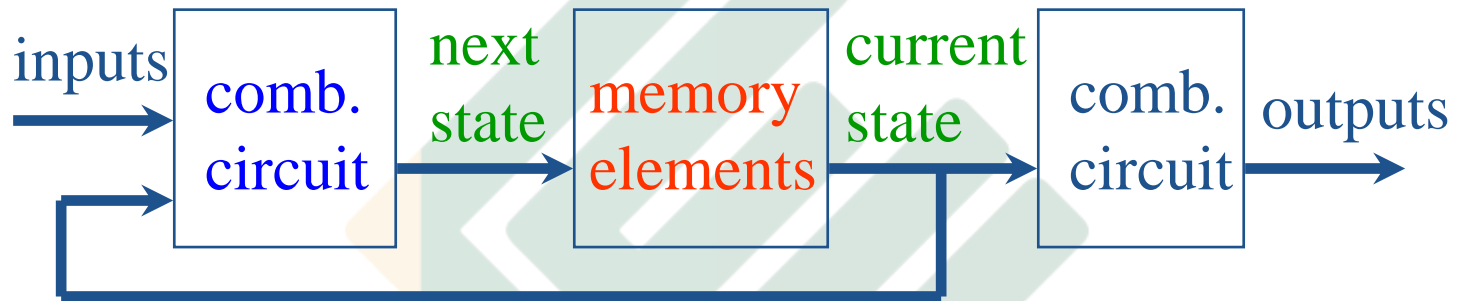
Why FSM ?

Finite State Machines (FSMs)

- Useful for designing many different types of circuits
- 3 basic components:
 - Combinational logic (next state)
 - Sequential logic (store state)
 - Output logic
- Different encodings for state:
 - Binary (min FF's), Gray, One hot (good for FPGA), One cold, etc

Finite State Machine

- Moore FSM model



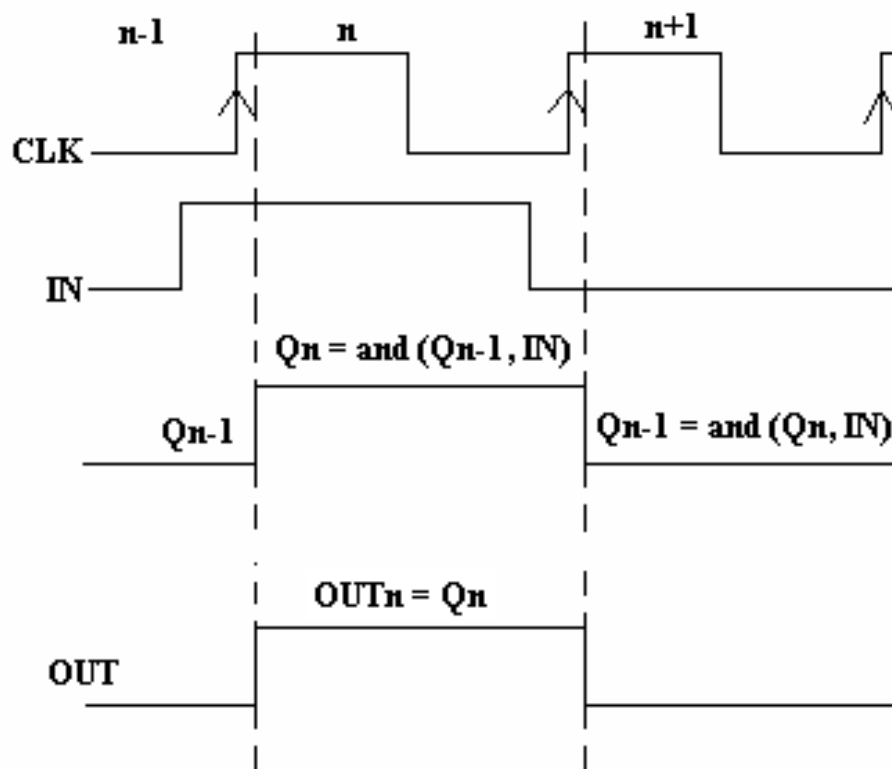
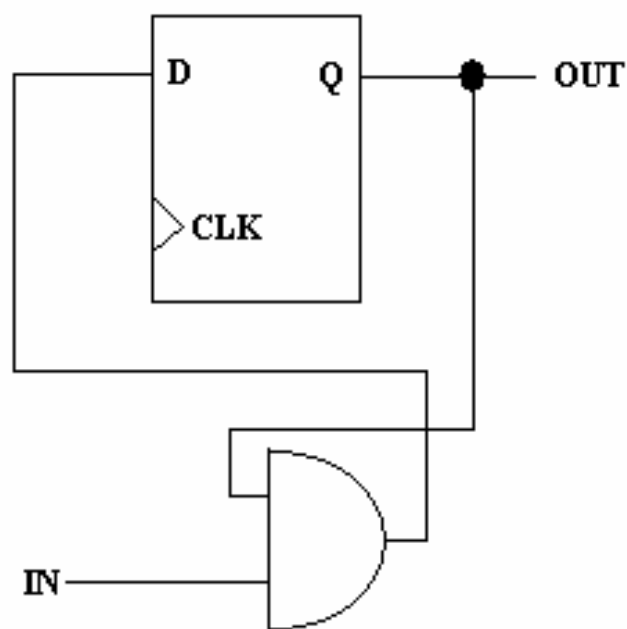
Next state = F (current state, inputs)
Outputs = G (current state)

COMPUTER ENGINEERING



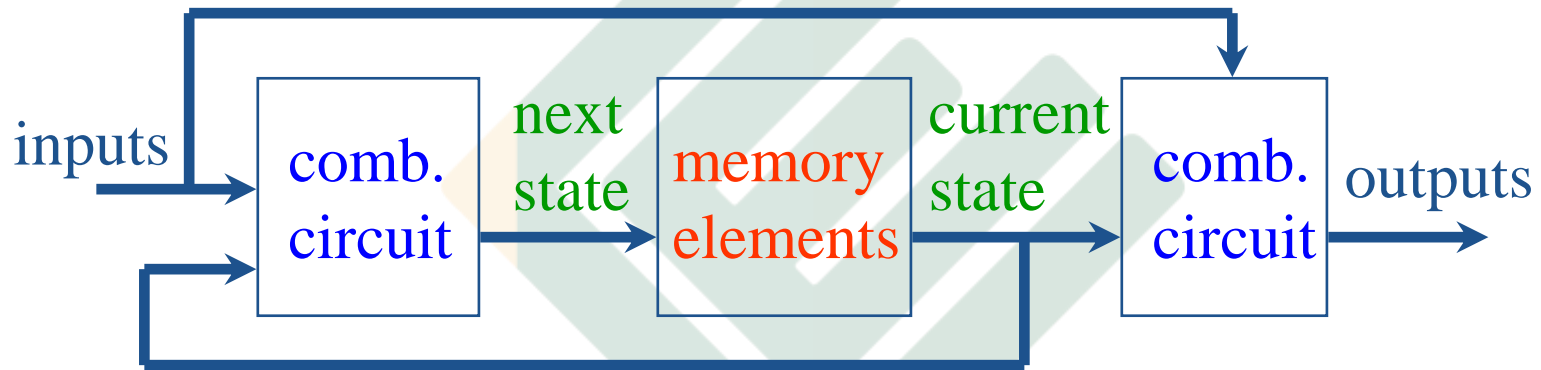
Finite State Machine

- Moore FSM model



Finite State Machine

- Mealy FSM model



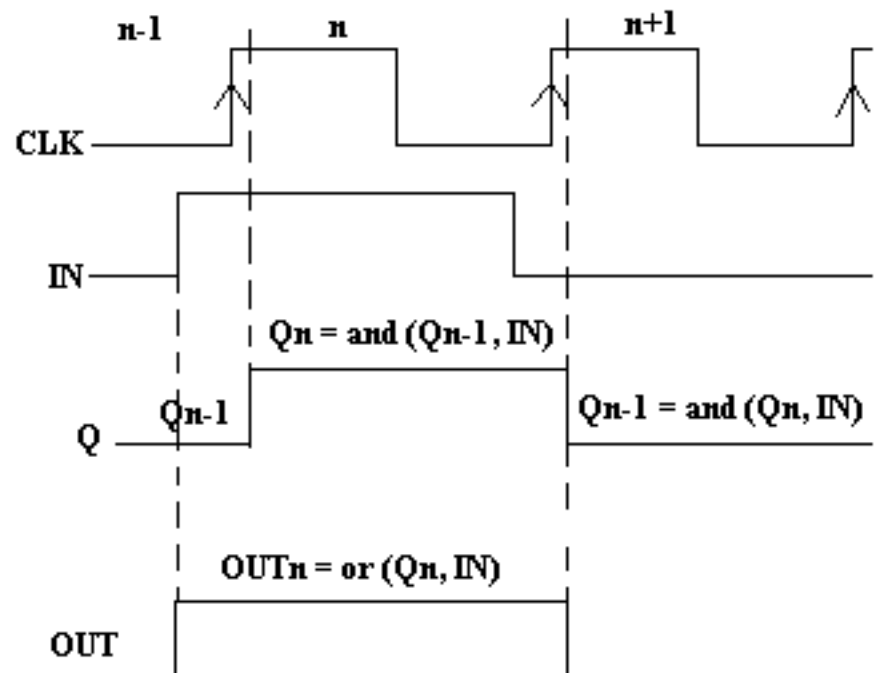
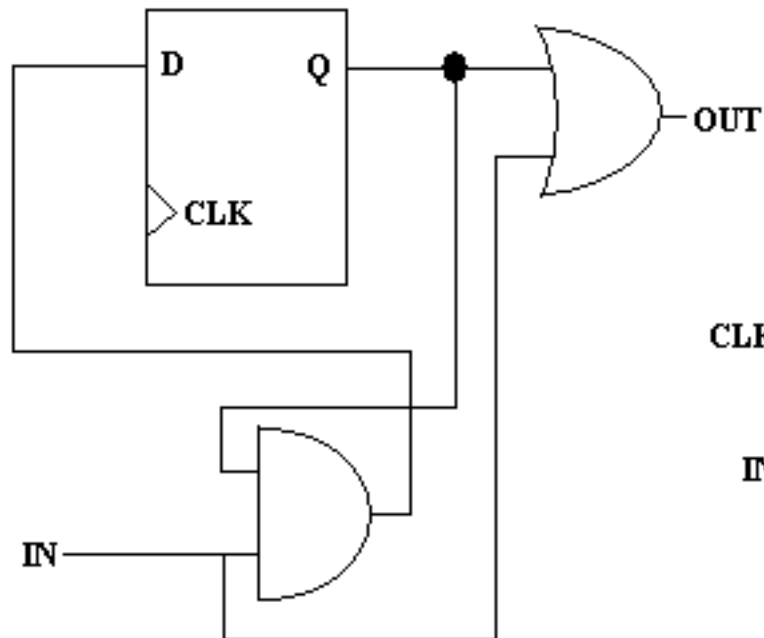
Next state = F (current state, inputs)

Outputs = G (current state, inputs)



Finite State Machine

- Mealy FSM model





FSMs modeling

- There are many ways to model FSMs:
 - ✓ Method1: Define the next-state logic combinatorially and define the state-holding latches explicitly
 - ✓ Method2: Define the behavior in a single always `@(posedge clk)` block
- Variations on these themes

COMPUTER ENGINEERING



FSMs modeling

➤ Method 1:

```
module FSM(o, a, b, reset);  
output o;  
reg o;  
input a, b, reset;  
reg [1:0] state, nextState;
```

Output o is declared a reg because it is assigned procedurally, not because it holds state

```
always @(a or b or state)  
case (state)  
2'b00: begin  
    nextState = a ? 2'b00 : 2'b01;  
    o = a & b;  
end  
2'b01: begin  
    nextState = 2'b10;  
    o = 0;  
end  
endcase
```

Combinational block must be sensitive to any change on any of its inputs

(Implies state-holding elements otherwise)

Latch implied by sensitivity to the clock or reset only

```
always @(posedge clk or reset)  
if (reset)  
    state <= 2'b00;  
else  
    state <= nextState;
```



FSMs modeling

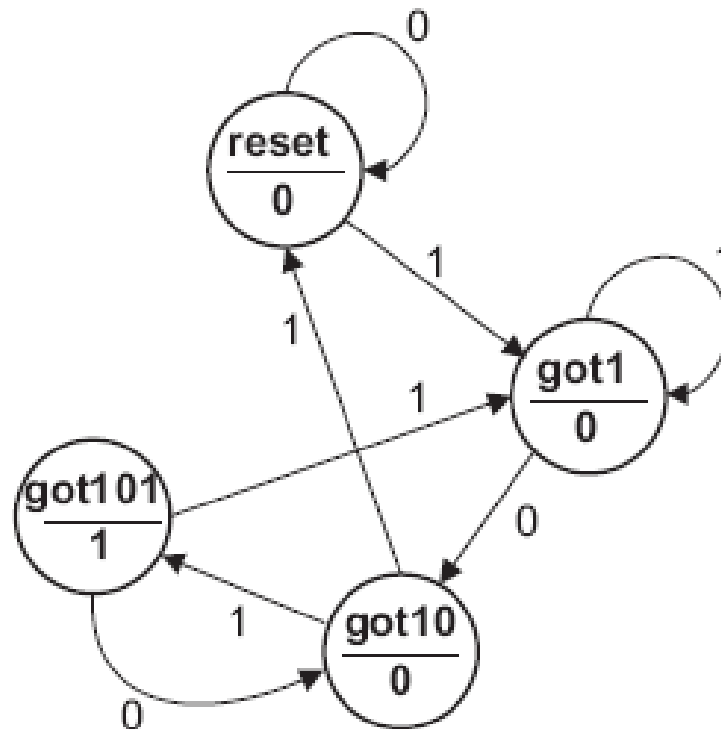
➤ Method2:

```
module FSM(o, a, b);  
output o;  
reg o;  
input a, b;  
reg [1:0] state;  
  
always @(posedge clk or reset)  
if (reset) state <= 2'b00;  
else case (state)  
2'b00: begin  
state <= a ? 2'b00 : 2'b01;  
o <= a & b;  
end  
2'b01: begin state <= 2'b10; o <= 0; end  
endcase
```



Example1: A Moore 101 Detector

A Moore 101 Detector



Example1: A Moore 101 Detector (Cont'd)

```
module Moore101Detector (dataIn, found, clock, reset);
```

```
//Input and Output Declarations
```

```
input          dataIn;
```

```
input          clock;
```

```
input          rst;
```

```
output         found;
```

```
//DataInternal Variables
```

```
reg [3:0]      state;
```

```
reg [3:0]      next_state;
```

```
//State Declarations
```

```
parameter reset = 3'b000;
```

```
parameter got1  = 3'b001;
```

```
parameter got10 = 3'b010;
```

```
parameter got101 = 3'b101;
```

JG

Example1: A Moore 101 Detector (Cont'd)

//Combinational Next State Logic

```
always @(state or dataIn)
```

```
case (state)
```

```
reset:
```

```
if (dataIn)
```

```
    next_state = got1;
```

```
else
```

```
    next_state = reset;
```

```
got1:
```

```
if (dataIn)
```

```
    next_state = got1;
```

```
else
```

```
    next_state = got10;
```

```
got10:
```

```
if (dataIn)
```

```
    next_state = got101;
```

```
else
```

```
    next_state = reset;
```

```
got101:
```

```
if (dataIn)
```

```
    next_state = got1;
```

```
else
```

```
    next_state = got10;
```

```
default:
```

```
    next_state = reset;
```

```
endcase // case(state)
```

//State FF Transition

```
always @(posedge clock)
```

```
if (rst == 1)
```

```
    state <= reset;
```

```
else
```

```
    state <= next_state;
```

//Combinational Output Logic

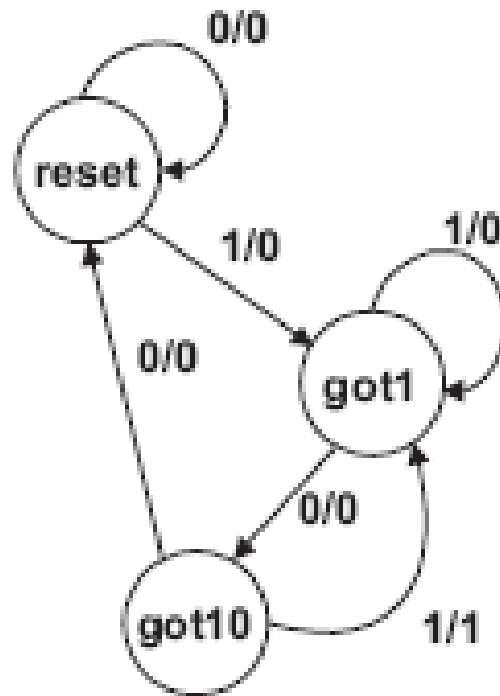
```
assign found = (state == got101) ? 1: 0;
```

```
endmodule // Moore101Detector
```



Example2: A Mealy 101 Detector

A 101 Mealy Machine



Example2: A Mealy 101 Detector (Cont'd)

```
module Mealy101Detector (dataIn, found, clock, reset);
```

```
//Input and Output Declarations
```

```
input          dataIn;  
input          clock;  
input          reset;  
output         found;
```

```
//DataInternal Variables
```

```
reg [3:0]      state;  
reg [3:0]      next_state;
```

```
//State Declarations
```

```
parameter reset = 3'b000;  
parameter got1  = 3'b001;  
parameter got10 = 3'b010;
```

JG

Example2: A Mealy 101 Detector (Cont'd)

//Combinational Next State Logic

```
always @(state or dataIn)
  case (state)
    reset:
      if (dataIn)
        next_state = got1;
      else
        next_state = reset;
    got1:
      if (dataIn)
        next_state = got1;
      else
        next_state = got10;
    got10:
      if (dataIn)
        next_state = got1;
      else
        next_state = reset;
```

default:

```
  next_state = reset;
endcase // case(state)
```

//State FF Transition

```
always @(posedge clock)
  if (reset == 1)
    state <= reset;
  else
    state <= next_state;
```

//Combinational Output Logic

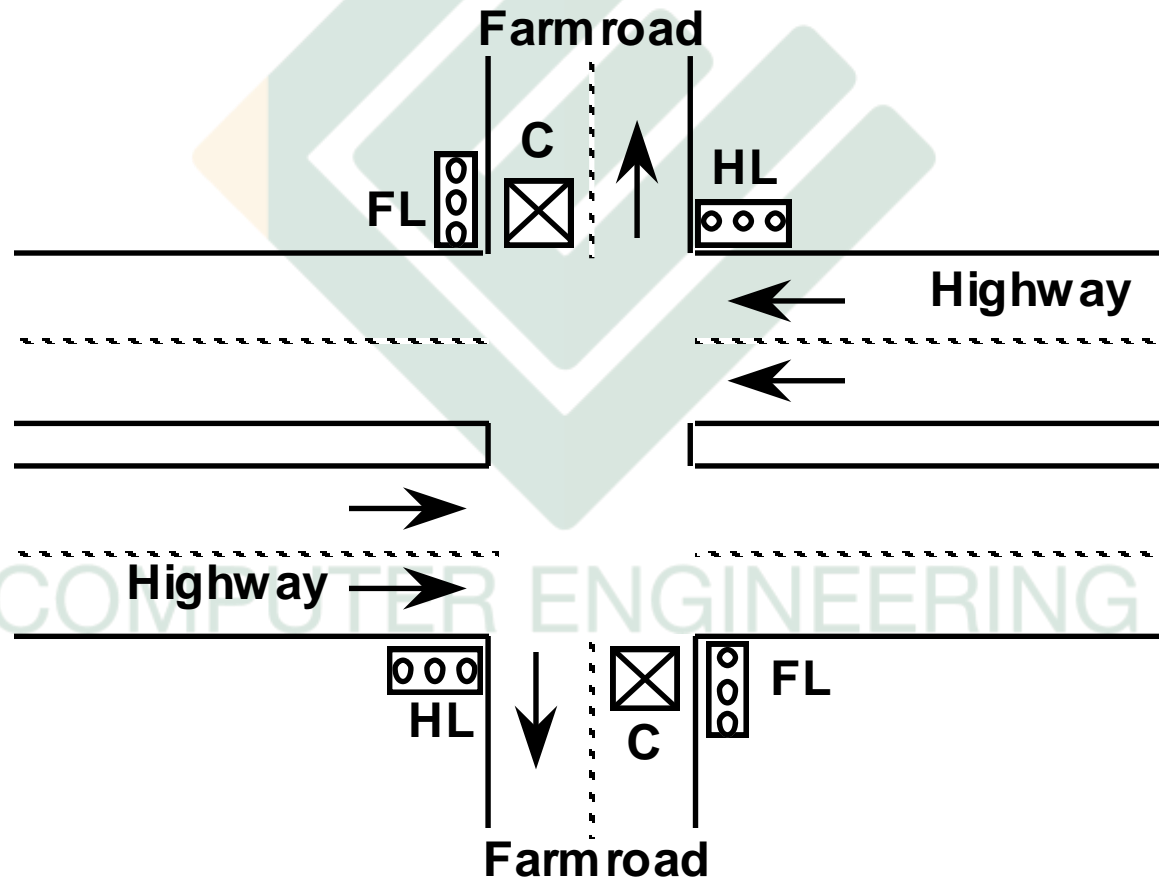
```
assign found = (state == got10 &&
  dataIn == 1) ? 1: 0;
```

```
endmodule // Mealy101Detector
```




Example3: Traffic Light Controller

➤ Picture of Highway/Farmroad Intersection:





Example3: Traffic Light Controller (Cont'd)

➤ Specifications

? Tabulation of Inputs and Outputs:

Input Signal

reset

C

TS

TL

Description

place FSM in initial state

detect vehicle on farmroad

short time interval expired

long time interval expired

Output Signal

HG, HY, HR

FG, FY, FR

ST

Description

assert green/yellow/red highway lights

assert green/yellow/red farmroad lights

start timing a short or long interval

? Tabulation of Unique States: Some light configuration imply others

State

S0

S1

S2

S3

Description

Highway green (farmroad red)

Highway yellow (farmroad red)

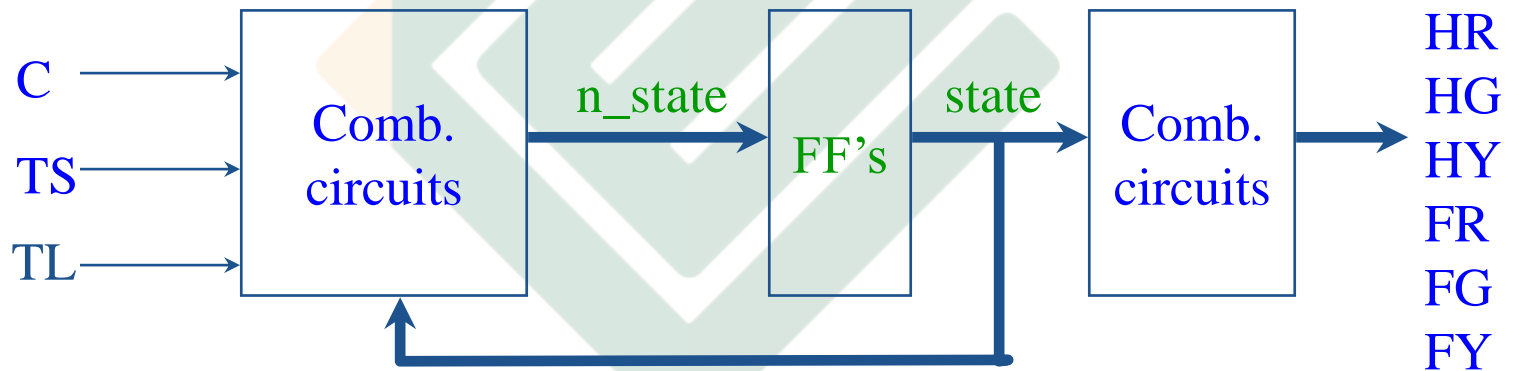
Farmroad green (highway red)

Farmroad yellow (highway red)



Example3: Traffic Light Controller (Cont'd)

➤ Block diagram

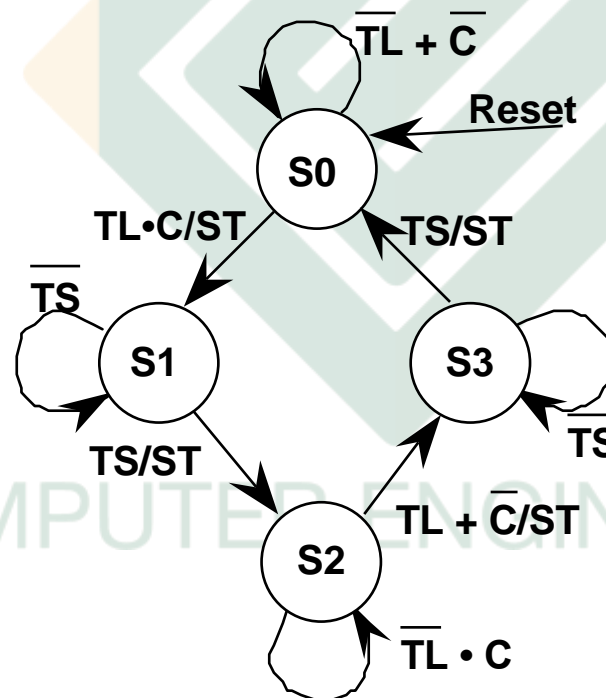


COMPUTER ENGINEERING



Example3: Traffic Light Controller (Cont'd)

➤ State transition diagram



S0: HG

S1: HY

S2: FG

S3: FY



Example3: Traffic Light Controller (Cont'd)

➤ Verilog FSM Description

```
module traffic_light(HG, HY, HR, FG, FY, FR, ST_o,  
                    tl, ts, clk, reset, c) ;  
    output HG, HY, HR, FG, FY, FR, ST_o;  
    input tl, ts, clk, reset, c ;  
    reg ST_o, ST ;  
    reg[0:1] state, next_state ;  
    parameter EVEN= 0, ODD=1 ;  
    parameter S0= 2'b00, S1=2'b01, S2=2'b10, S3=2'b11;  
    assign HG = (state == S0) ;  
    assign HY = (state == S1) ;  
    assign HR = ((state == S2)|| (state == S3));  
    assign FG = (state == S2) ;  
    assign FY = (state == S3) ;  
    assign FR = ((state == S0)|| (state == S1)) ;
```



Example3: Traffic Light Controller (Cont'd)

➤ Verilog FSM Description (Cont'd)

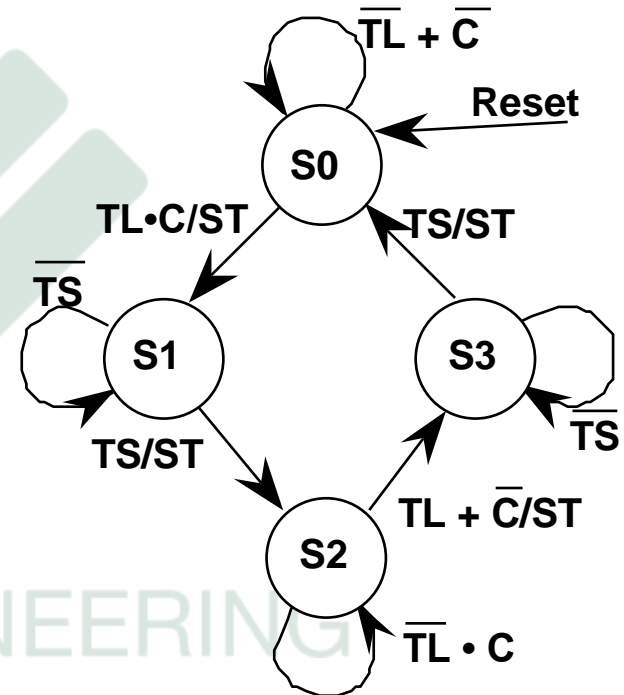
```
// flip-flops
always@ (posedge clk or posedge reset)
    if(reset)          // an asynchronous reset
        begin
            state = S0 ;
            ST_o = 0 ;
        end
    else
        begin
            state = next_state ;
            ST_o = ST ;
        end
    end
```



Example3: Traffic Light Controller (Cont'd)

➤ Verilog FSM Description (Cont'd)

```
always@ (state or c or tl or ts)
  case(state)                                // state transition
    S0:
      if(tl & c)
        begin
          next_state = S1 ;
          ST = 1 ;
        end
      else
        begin
          next_state = S0 ;
          ST = 0 ;
        end
  end
```



Example3: Traffic Light Controller (Cont'd)

➤ Verilog FSM Description (Cont'd)

S1:

```
if (ts) begin
```

```
    next_state = S2 ;
```

```
    ST = 1 ;
```

```
end
```

```
else begin
```

```
    next_state = S1 ;
```

```
    ST = 0 ;
```

```
end
```

S2:

```
if(tl | !c) begin
```

```
    next_state = S3 ;
```

```
    ST = 1 ;
```

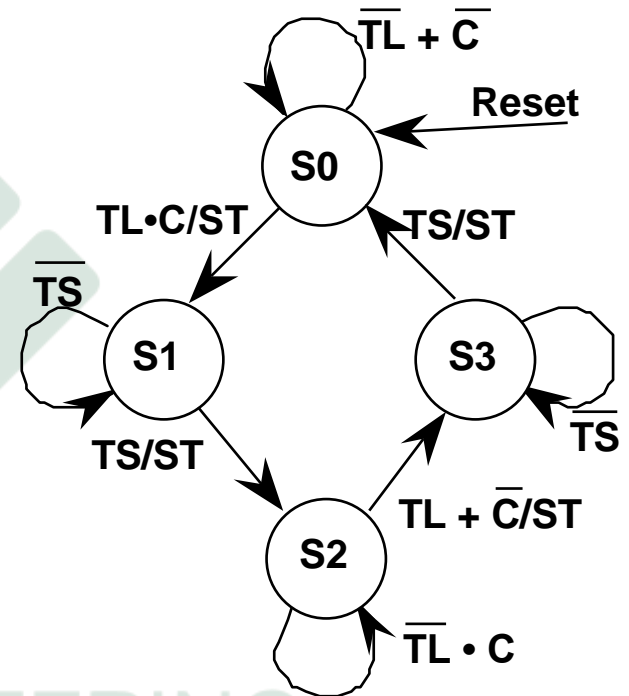
```
end
```

```
else begin
```

```
    next_state = S2 ;
```

```
    ST = 0 ;
```

```
end
```

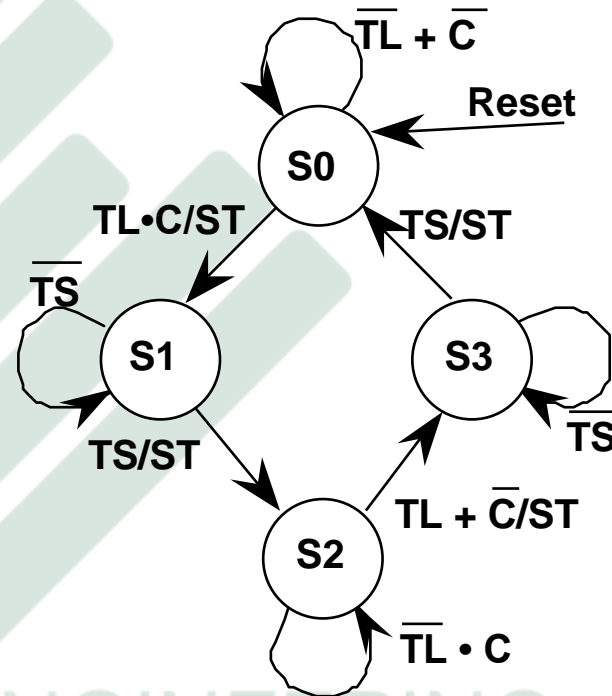




Example3: Traffic Light Controller (Cont'd)

➤ Verilog FSM Description (Cont'd)

```
S3:
  if(ts)
    begin
      next_state = S0 ;
      ST = 1 ;
    end
  else
    begin
      next_state = S3 ;
      ST = 0 ;
    end
  endcase
endmodule
```





Tips on FSM

- Don't forget to handle the default case
- Use two different always blocks for next state and state assignment
 - Can do it in one big block but not as clear
- Outputs can be a mix of combin. and seq.
 - Moore Machine: Output only depends on state
 - Mealy Machine: Output depends on state and inputs

COMPUTER ENGINEERING



To ensure proper recognition and inference of state machines and to improve the quality of results, Altera recommends that you observe the following guidelines, which apply to both Verilog HDL and VHDL:

- **Assign default values to outputs derived from the state machine so that synthesis does not generate unwanted latches.**

- Separate the state machine logic from all arithmetic functions and data paths, including assigning output values.

- If your design contains an operation that is used by more than one state, define the operation outside the state machine and cause the output logic of the state machine to use this value.

- Use a simple asynchronous or synchronous reset to ensure a defined power-up state. If your state machine design contains more elaborate reset logic, such as both an asynchronous reset and an asynchronous load, the Quartus II software generates regular logic rather than inferring a state machine.



END

COMPUTER ENGINEERING