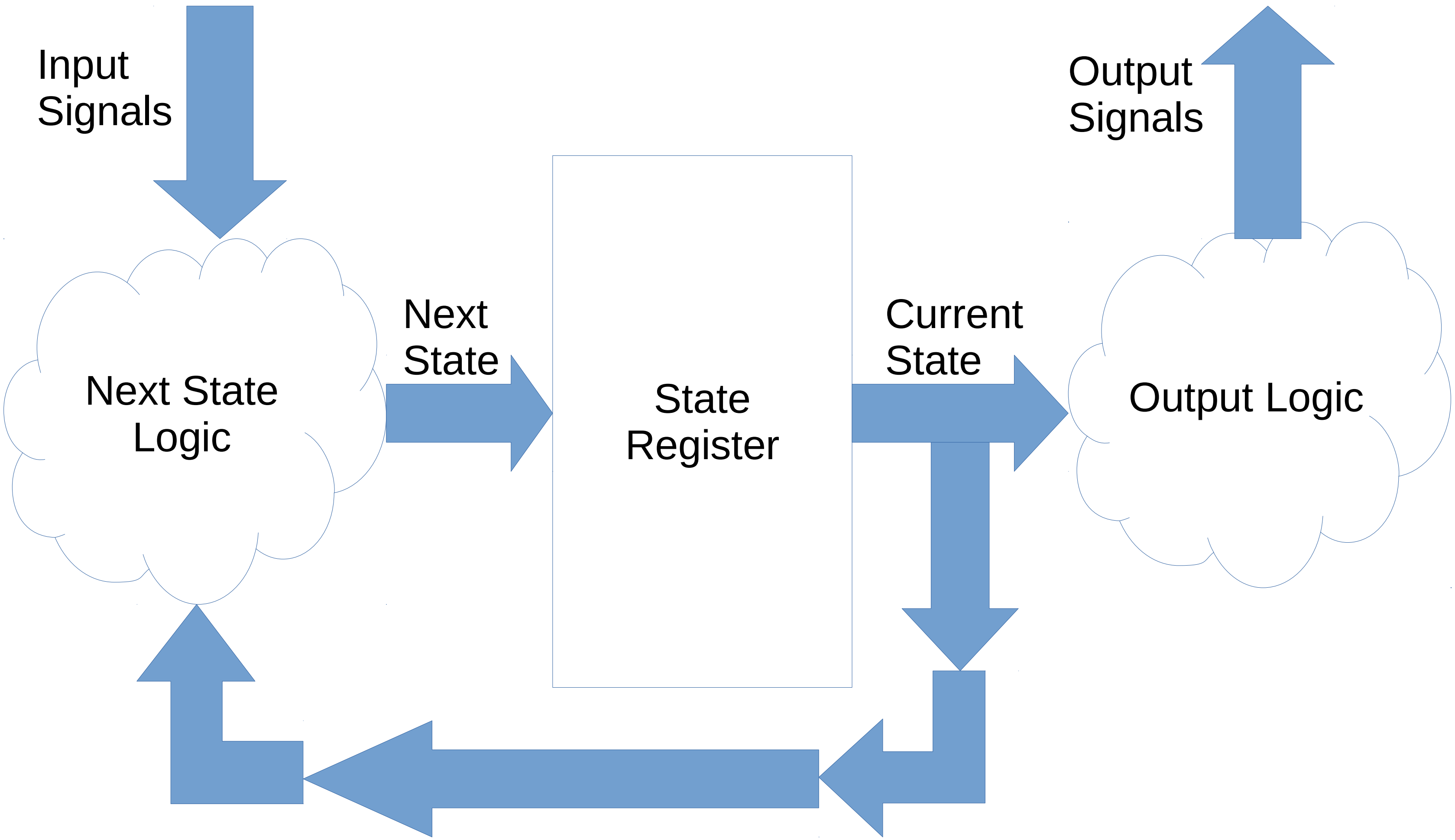
Introduction to Verilog and ModelSim

(Part 6 – State Machines)

State Machine

- Actually, a Finite State Machine (FSM)
  - mathematical model of computation
  - abstract machine with finite states
  - can only be in ONE state at any given time
- Consists of
  - next state logic calculation (combinational)
  - current state logic (sequential)
  - output / control logic (combinational)

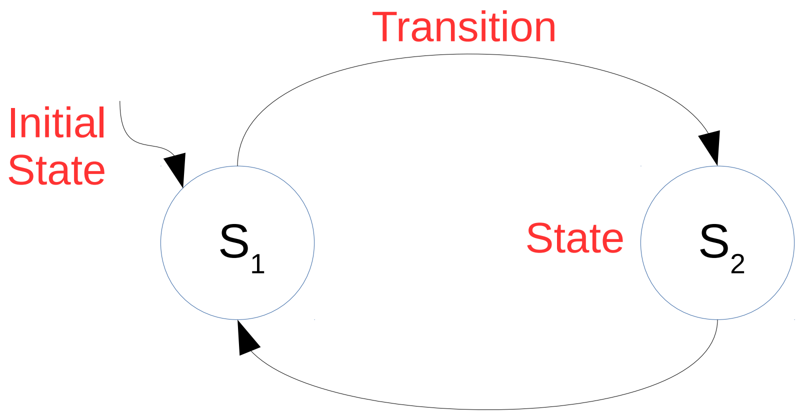
State Machine (cont.)



State Machine (cont.)

- State machines for control applications
  - 2 distinct types: Moore and Mealy machines
- Moore Machine
  - output depends only on state
  - entry actions only
- Mealy Machine
  - output depends on both state and input
  - input actions only

State Diagram



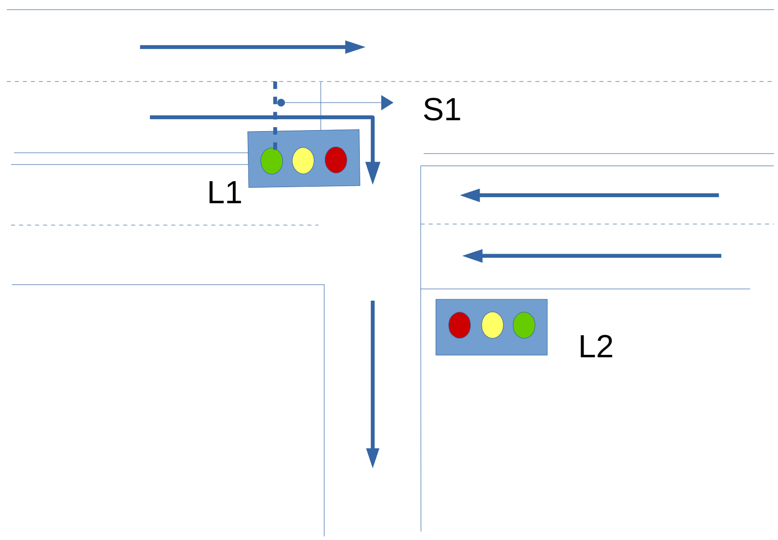
- Many variants
  - Use the one used here
- Circles are states
  - Actions (e.g. entry, exit) tied to states
- Arrows are transitions
  - Specify cause triggers
  - Special arrow for initial state (reset@default)

State Encoding

- Binary encoding
  - Normal binary counter (e.g. 00, 01, 10, 11)
- Gray encoding
  - Basically a counter with a single bit transition
  - (e.g. 00, 01, 11, 10)
- One-{hot,cold} encoding
  - Single bit 'active' at a time (hot-1,cold-0)
  - (e.g. 0001, 0010, 0100, 1000)

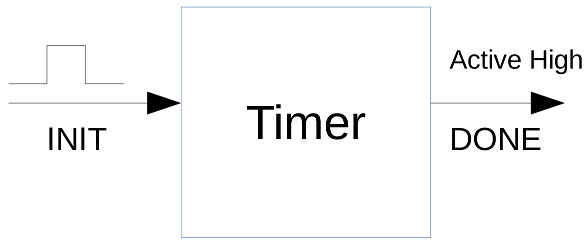
FSM Implementation

- Example: Traffic Light Controller
  - L2 will be at GREEN (G), L1 at RED (R)
  - When S1 triggers → L2 changes after tD time unit
  - L2=Y for tY time unit, L2=R for tR time unit
  - L1=G for tG time unit
  - L1=Y for tY time unit
  - L1=R for tR time unit

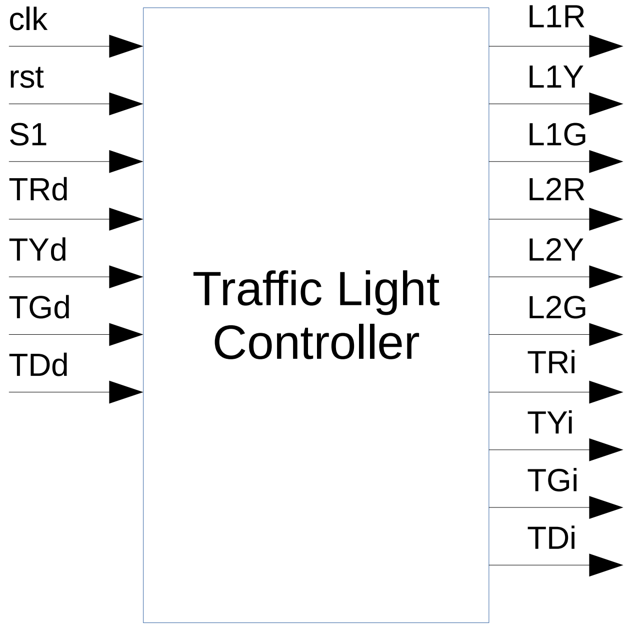


FSM Implementation (cont.)

- Clearly,
  - 6 output lines
  - 1 input line
- But, what about time?
  - Assume timers!
  - 4 inputs, 4 outputs

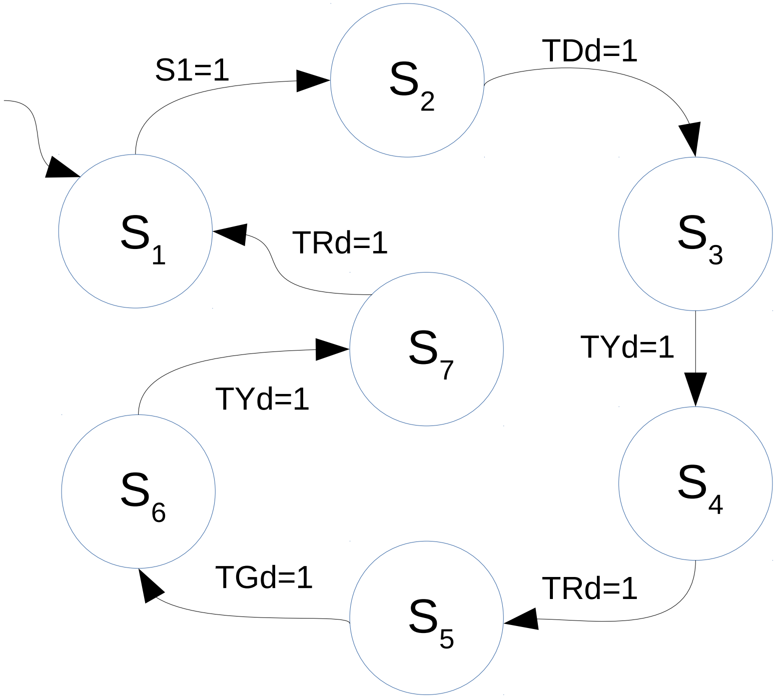


- So, target module:



FSM Implementation (cont.)

- Outputs and State Diagram
  - S1={1,0,0,0,0,1,0,0,0,0}
  - S2={1,0,0,0,0,1,0,0,0,1}
  - S3={1,0,0,0,1,0,0,1,0,0}
  - S4={1,0,0,1,0,0,1,0,0,0}
  - S5={0,0,1,1,0,0,0,0,1,0}
  - S6={0,1,0,1,0,0,0,1,0,0}
  - S7={1,0,0,1,0,0,1,0,0,0}



- Is this a Mealy Machine or a Moore Machine?

FSM Implementation (cont.)

- Decide state encoding
  - binary encoding (minimum number of register)
  - 1-hot encoding (easier to decode @ get output logic)
- Get equation for output logic
  - 1 equation per output line
  - Mealy (input &state) or Moore (state only)
- Get equation for next state logic
  - state register output(s) → input(s)
  - next state logic output → state register input

Let's choose these!

FSM Implementation (cont.)

- Truth table for output logic

Logic for L1 Red

cS[2]	cS[1]	cS[0]	L1	L2	TXd
0	0	1	{ 1,0,0 }	{ 0,0,1 }	{ 0,0,0,0 }
0	1	0	{ 1,0,0 }	{ 0,0,1 }	{ 0,0,0,1 }
0	1	1	{ 1,0,0 }	{ 0,1,0 }	{ 0,1,0,0 }
1	0	0	{ 1,0,0 }	{ 1,0,0 }	{ 1,0,0,0 }
1	0	1	{ 0,0,1 }	{ 1,0,0 }	{ 0,0,1,0 }
1	1	0	{ 0,1,0 }	{ 1,0,0 }	{ 0,1,0,0 }
1	1	1	{ 1,0,0 }	{ 1,0,0 }	{ 1,0,0,0 }

FSM Implementation (cont.)

- Truth table for next state logic

Logic for D[2]

cS[2]	cS[1]	cS[0]	Trigger	nS[2]	nS[1]	nS[0]
0	0	1	S1	0	1	0
0	1	0	TDd	0	1	1
0	1	1	TYd	1	0	0
1	0	0	TRd	1	0	1
1	0	1	TGd	1	1	0
1	1	0	TYd	1	1	1
1	1	1	TRd	0	0	1

FSM in Verilog

```
module fsm_tlc (clk,rst,S1,TXd,
     L1,L2,TXi);

input clk, rst, S1;
input [3:0] TXd;
output [2:0] L1,L2;
output [3:0] TXi;
wire [2:0] L1,L2;
wire [3:0] TXi;
reg [2:0] cST;
wire [2:0] nST, nSTi;

always @(posedge clk or posedge rst)
begin
     if (rst) cST <= 3'b001;
     else cST <= nST;
end

// assign output L1
assign L1[2] = ~cST[2]|~(cST[1]^cST[0]);
assign L1[1] = cST[2]&cST[1]&~cST[0];
assign L1[0] = cST[2]&~cST[1]&cST[0];

// assign output L2
assign L2[2] = cST[2];
assign L2[1] = ~cST[2]&cST[1]&cST[0];
assign L2[0] = ~cST[2]&(cST[1]^cST[0]);

// assign output timer control
assign TXi[3] = cST[2]&~(cST[1]^cST[0]);
assign TXi[2] = cST[1]&(cST[2]^cST[0]);
assign TXi[1] = cST[2]&~cST[1]&cST[0];
assign TXi[0] = ~cST[2]&cST[1]&~cST[0];
```

```
// assign next state
assign nST[2] =
     (~cST[2]&cST[1]&cST[0]&TXd[1])|
     (cST[2]&~cST[1]&~cST[0]&TXd[0])|
     (cST[2]&~cST[1]&cST[0]&TXd[2])|
     (cST[2]&cST[1]&~cST[0]&TXd[1]);
assign nST[1] =
     (~cST[2]&~cST[1]&cST[0]&S1)|
     (~cST[2]&cST[1]&~cST[0]&TXd[3])|
     (cST[2]&~cST[1]&cST[0]&TXd[2])|
     (cST[2]&cST[1]&~cST[0]&TXd[1]);
assign nST[0] =
     (~cST[2]&cST[1]&~cST[0]&TXd[3])|
     (cST[2]&~cST[1]&~cST[0]&TXd[0])|
     (cST[2]&cST[1]&~cST[0]&TXd[1])|
     (cST[2]&cST[1]&cST[0]&TXd[0]);

endmodule
```

FSM in Verilog (Test Bench 1)

```
module fsm_tlc_tb ();
reg clk, rst, S1;
reg [3:0] TXd;
wire [2:0] L1,L2;
wire [3:0] TXi;
reg init;
// reset block
initial begin
  clk=1'b0; rst=1'b0; S1=1'b0;
  TXd[3]=0; TXd[2]=0;
  TXd[1]=0; TXd[0]=0; init = 0;
  // generate clock here
  forever #5 clk = ~clk;
end
// timer RED
always @(TXi[3]) begin
  if (init&TXi[3]) begin
  $display("[%g] TR Init",$time);
    #40 TXd[3] = 1'b1;
  $display("[%g] TR Done",$time);
  end
  else TXd[3] = 1'b0;
end
// timer YELLOW
always @(TXi[2]) begin
  if (init&TXi[2]) begin
  $display("[%g] TY Init",$time);
    #20 TXd[2] = 1'b1;
  $display("[%g] TY Done",$time);
  end
  else TXd[2] = 1'b0;
end
```

```
// timer GREEN
always @(TXi[1]) begin
  if (init&TXi[1]) begin
  $display("[%g] TG Init",$time);
    #200 TXd[1] = 1'b1;
  $display("[%g] TG Done",$time);
  end
  else TXd[1] = 1'b0;
end
// timer DELAY
always @(TXi[0]) begin
  if (init&TXi[0]) begin
  $display("[%g] TD Init",$time);
    #20 TXd[0] = 1'b1;
  $display("[%g] TD Done",$time);
  end
  else TXd[0] = 1'b0;
end
// generate reset signal
initial begin
  #50; rst = 1'b1;
  $display("[%g] RESET!",$time);
  #50; rst = 1'b0;
  init = 1'b1;
  $display("[%g] INIT!",$time);
end
```

```
// generate stimuli
initial begin
  #150; // wait reset done
  $display("[%g] S1 NOW!",$time);
  S1 = 1'b1; // trigger S1
  #20;
  S1 = 1'b0;
  #500; $stop;
end
// monitor block - state
always @(dut.cST)
begin
  $display("[%g] Check: [%b]",
    $time,dut.cST);
end
// monitor block - clock
always @(negedge clk)
begin
  $display("[%g] State: [%b]",
    $time,dut.cST);
end
// instantiate DUT
fsm_tlc dut (clk,rst,S1,
  TXd,L1,L2,TXi);
endmodule
```

FSM in Verilog (Analysis 1)

- Notice some events look out of place!
  - race conditions in software events
  - within same simulation time, no precedence!
- Testbench design can be improved
  - should avoid race conditions (generate stimuli earlier than processing clock edges – DONE!)
  - use system task $strobe to propagate events towards the end of event queue (try replace $display in timer always block)

Practical Session 6.1

- Rewrite the FSM module for the given traffic light controller using 1-hot encoding
- Verify the design using the same testbench

FSM in Verilog (Analysis 2)

- There is an easier way to do this → YAY!
  - using always block for combinational logic
  - state transitions are obvious
- The previous example is still important
  - shows that next-state decision logic IS combinational logic!
  - still, not really practical to use in real design (time to market!)

FSM in Verilog (Alt.)

```
module fsma_tlc (clk,rst,S1,TXd,
     L1,L2,TXi);

input clk, rst, S1;
input [3:0] TXd;
output [2:0] L1,L2;
output [3:0] TXi;
wire [2:0] L1,L2;
wire [3:0] TXi;
reg [2:0] cST, nST;

// define useful parameters
parameter [2:0] STATE1=3'b001,
  STATE2=3'b010, STATE3=3'b011,
  STATE4=3'b100, STATE5=3'b101,
  STATE6=3'b110, STATE7=3'b111;

// state register
always @(posedge clk or
  posedge rst)
begin
  if (rst) cST <= STATE1;
  else cST <= nST;
end
```

```
// get next state logic
always @(S1 or TXd)
begin
  case (cST)
    STATE1: if (S1) nST=STATE2;
      else nST=STATE1;
    STATE2:
      if (TXd[0]) nST=STATE3;
      else nST=STATE2;
    STATE3:
      if (TXd[2]) nST=STATE4;
      else nST=STATE3;
    STATE4:
      if (TXd[3]) nST=STATE5;
      else nST=STATE4;
    STATE5:
      if (TXd[1]) nST=STATE6;
      else nST=STATE5;
    STATE6:
      if (TXd[2]) nST=STATE7;
      else nST=STATE6;
    STATE7:
      if (TXd[3]) nST=STATE1;
      else nST=STATE7;
    default: nST = STATE1;
  endcase
end
```

```
// assign output L1
assign L1[2] =
  ~cST[2]|~(cST[1]^cST[0]);
assign L1[1] =
  cST[2]&cST[1]&~cST[0];
assign L1[0] =
  cST[2]&~cST[1]&cST[0];
// assign output L2
assign L2[2] = cST[2];
assign L2[1] =
  ~cST[2]&cST[1]&cST[0];
assign L2[0] =
  ~cST[2]&(cST[1]^cST[0]);
// assign output timer control
assign TXi[3] =
  cST[2]&~(cST[1]^cST[0]);
assign TXi[2] =
  cST[1]&(cST[2]^cST[0]);
assign TXi[1] =
  cST[2]&~cST[1]&cST[0];
assign TXi[0] =
  ~cST[2]&cST[1]&~cST[0];

endmodule
```

That's all, folks!