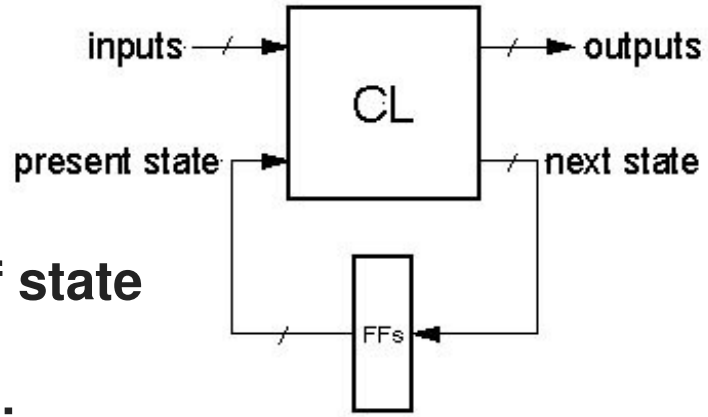
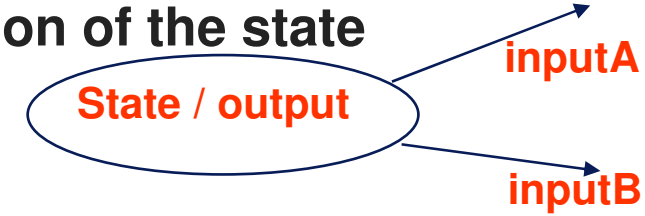
Using, Modeling and Implementing FSMs

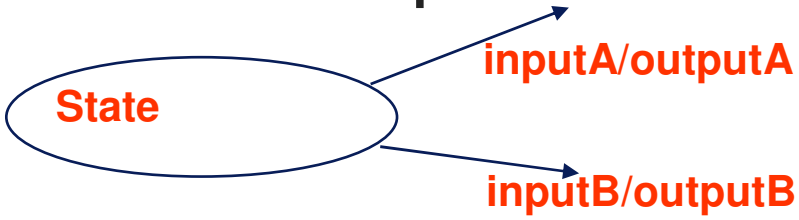
Review: What's an FSM?

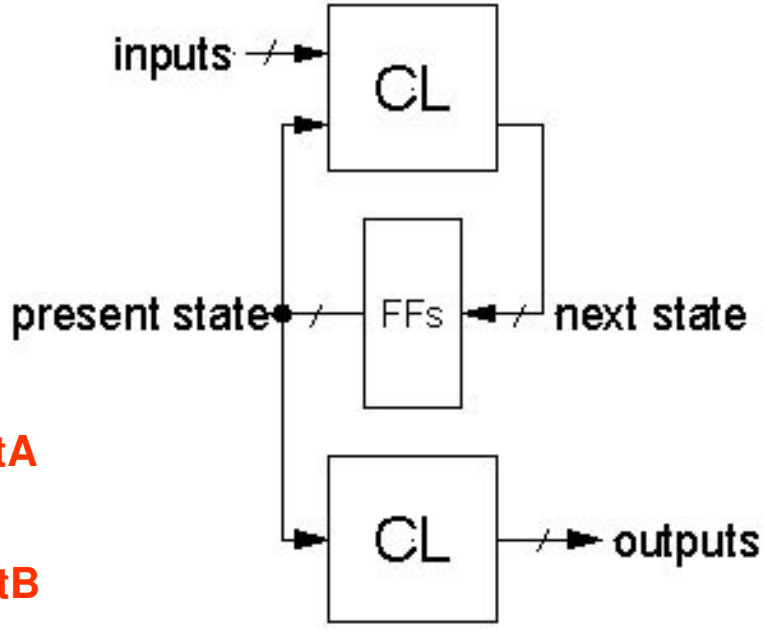


- Next state is function of state and input
- Moore Machine: output is a function of the state



- *Mealy Machine*: output is a function of state and input



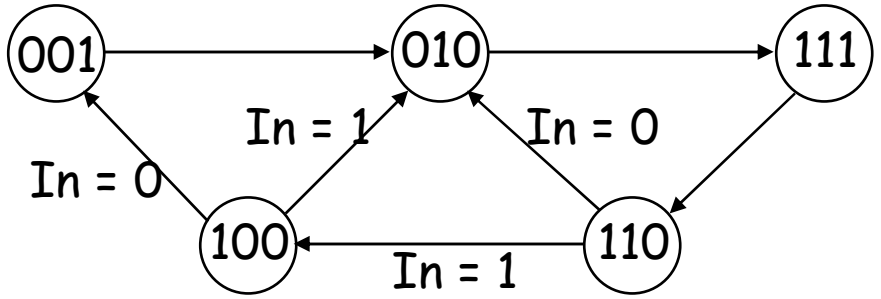


Review: Formal Design Process

- Review of Design Steps:
  1. Circuit functional specification
  2. State Transition Diagram
  3. Symbolic State Transition Table
  4. Encoded State Transition Table
  5. Derive Logic Equations
  6. Circuit Diagram
     - FFs for state
     - CL for NS and OUT

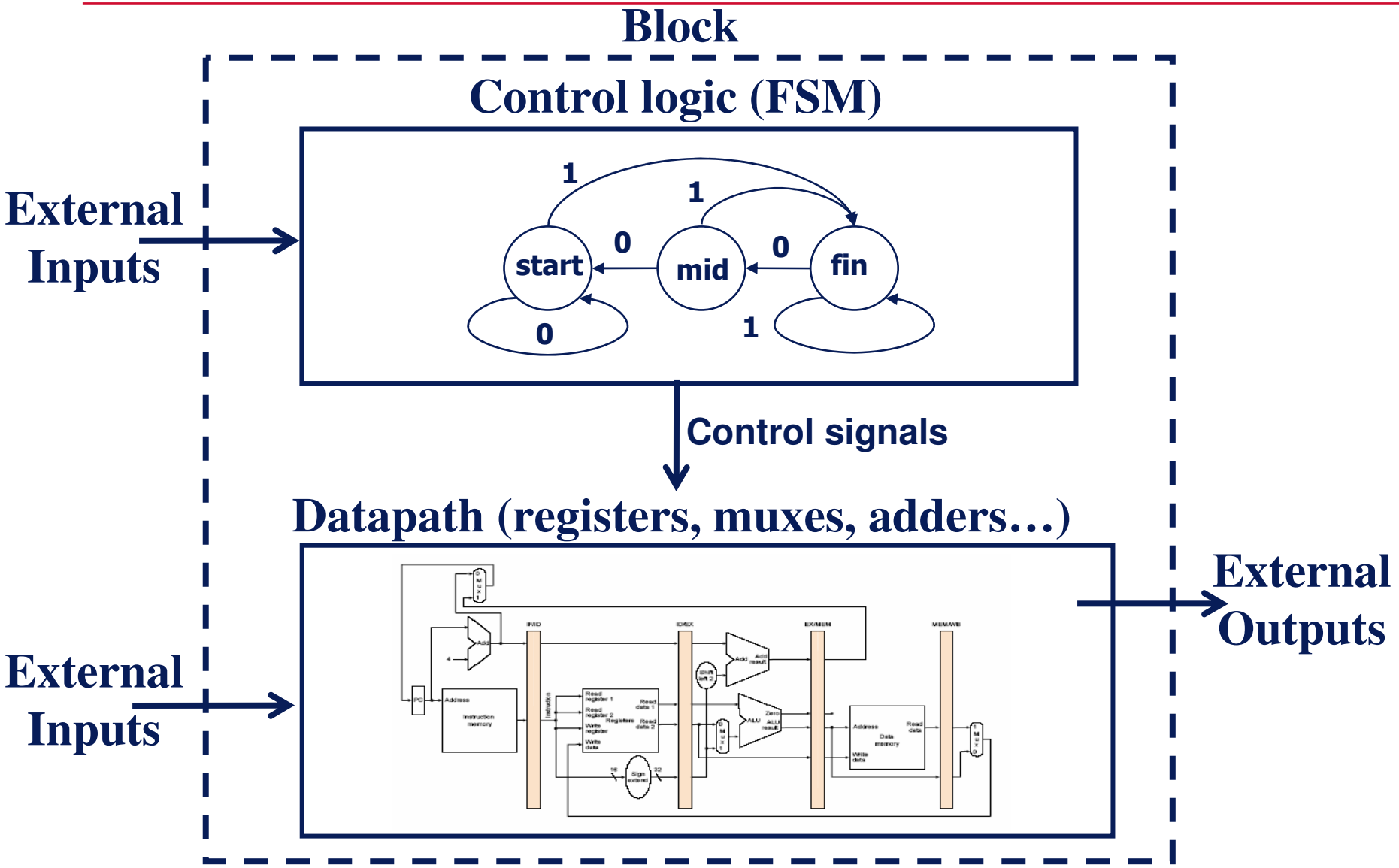
Finite State Machine Representations

- States: determined by possible values in sequential storage elements
- Transitions: change of state
- Clock: controls when state can change by controlling storage elements



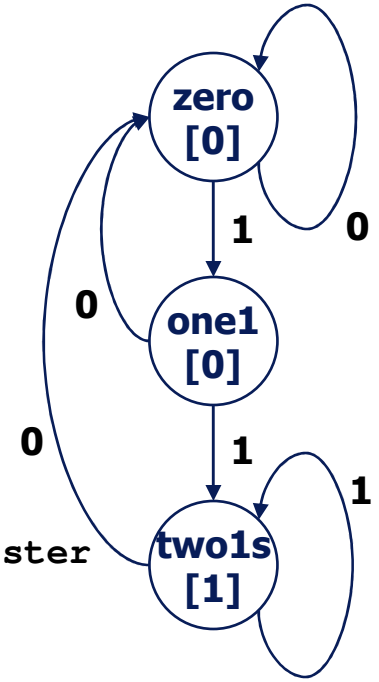
- Sequential Logic
  - Sequences through a series of states
  - Based on sequence of values on input signals
  - Clock period defines elements of sequence

How do I use FSMs ?



Example1 : Reduce 1s example

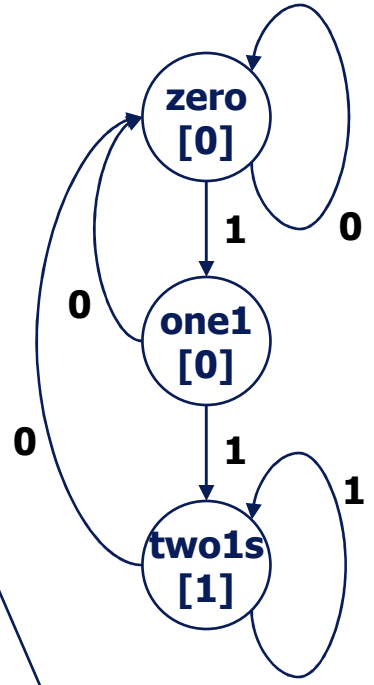
- Change the first 1 to 0 in each string of 1's
  - Example Moore machine implementation




```
module Reduce(Out, Clock, Reset, In);
    output                Out;
    input                 Clock, Reset, In;

    reg      Out;
    reg [1:0]             CurrentState;     // state register
    reg [1:0]             NextState;

    // State assignment
    localparam            STATE_Zero =      2'h0,
                          STATE_One1 =      2'h1,
                          STATE_Two1s =     2'h2,
                          STATE_X =         2'hX;
```

Moore Verilog FSM: combinational part

```
always @(In or CurrentState) begin
    NextState =                     CurrentState;
    Out =                           1'b0;
    case (CurrentState)
        STATE_Zero: begin // last input was a zero
            if (In) NextState =   STATE_One1;
        end
        STATE_One1: begin // we've seen one 1
            if (In) NextState =   STATE_Two1s;
            else NextState =      STATE_Zero;
        end
        STATE_Two1s: begin // we've seen at least 2 ones
            Out =                 1;
            if (~In) NextState =  STATE_Zero;
        end
        default: begin // in case we reach a bad state
            Out =                 1'bx;
            NextState =           STATE_X;
        end
    endcase
end
```



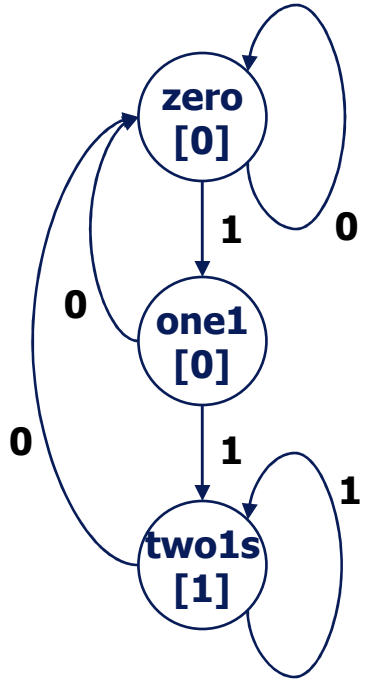
include all signals that are input to state and output equations

Compute: output = G(state)

next state = F(state, in)

Moore Verilog FSM: state part

```
    // Implement the state register
    always @ (posedge Clock) begin
       if (Reset) CurrentState <= STATE_Zero;
       else CurrentState <=       NextState;
    end
endmodule
```



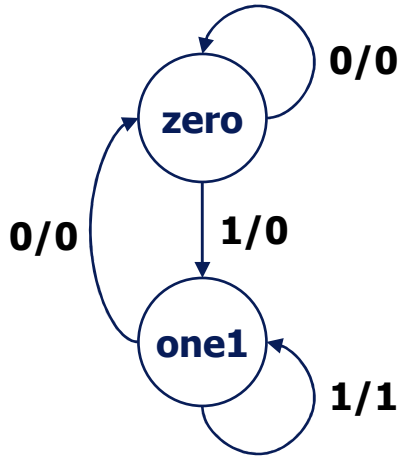
Note: posedge Clock requires NONBLOCKING ASSIGNMENT.

Blocking Assignment <-> Combinational Logic

Nonblocking Assignment <-> Sequential Logic (Registers)

Mealy Verilog FSM for Reduce-1s example

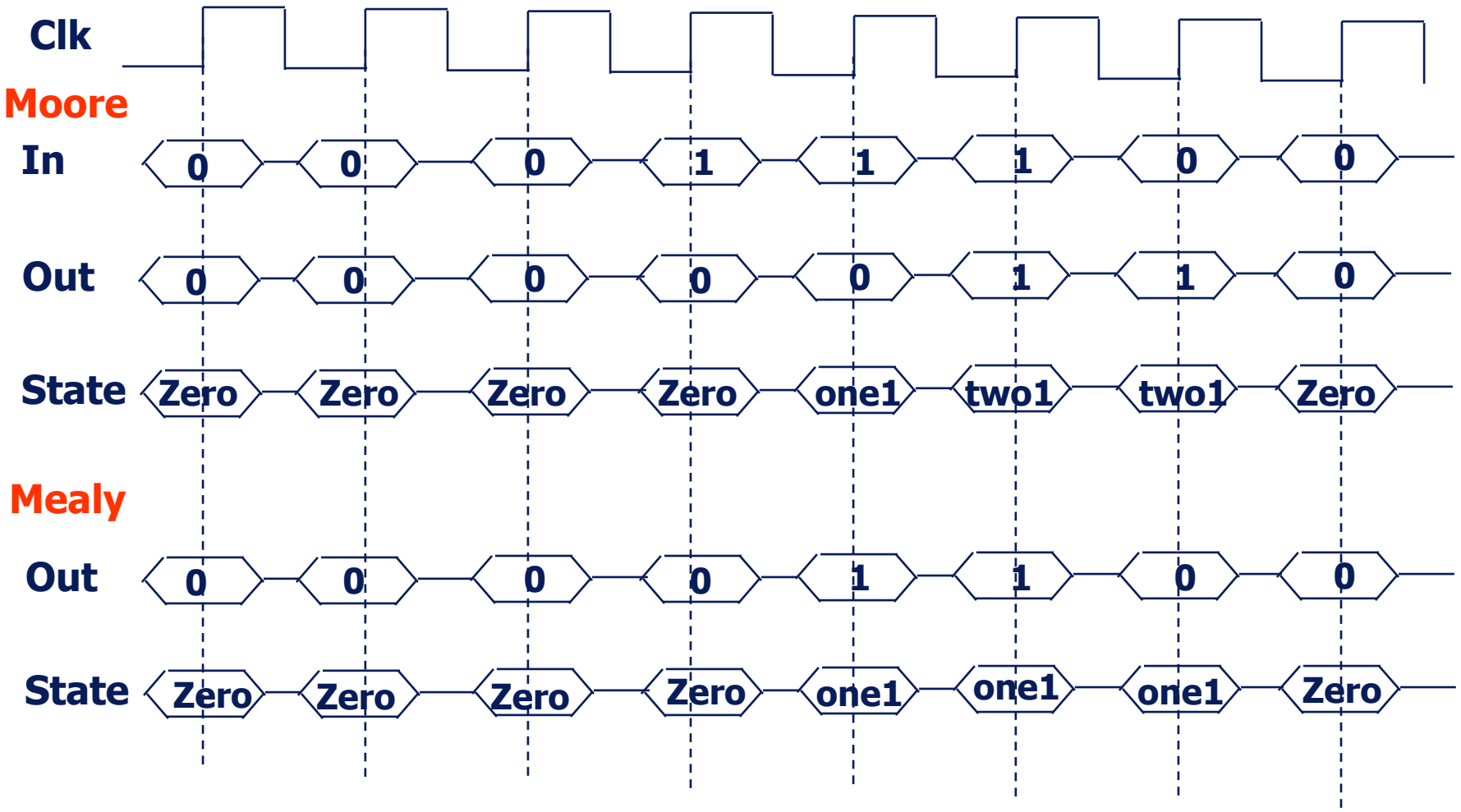
```
module Reduce(Clock, Reset, In, Out);
    input    Clock, Reset, In;
    output   Out;
    reg      Out;
    reg      CurrentState;          // state register
    reg      NextState;

    localparam          STATE_Zero =       1'b0,
                        STATE_One =        1'b1;

    always @(posedge Clock) begin
        if (Reset) CurrentState <=         STATE_Zero;
        else CurrentState <=               NextState;
    end

    always @ (In or CurrentState) begin
        NextState =                        CurrentState;
        Out =                              1'b0;
        case (CurrentState)
            State_Zero: if (In) NextState =        STATE_One;
            State_One: begin               // we've seen one 1
                       if (In) NextState = STATE_One;
                       else NextState =    STATE_Zero;
                       Out =               In;
            end
        endcase
    end
endmodule
```



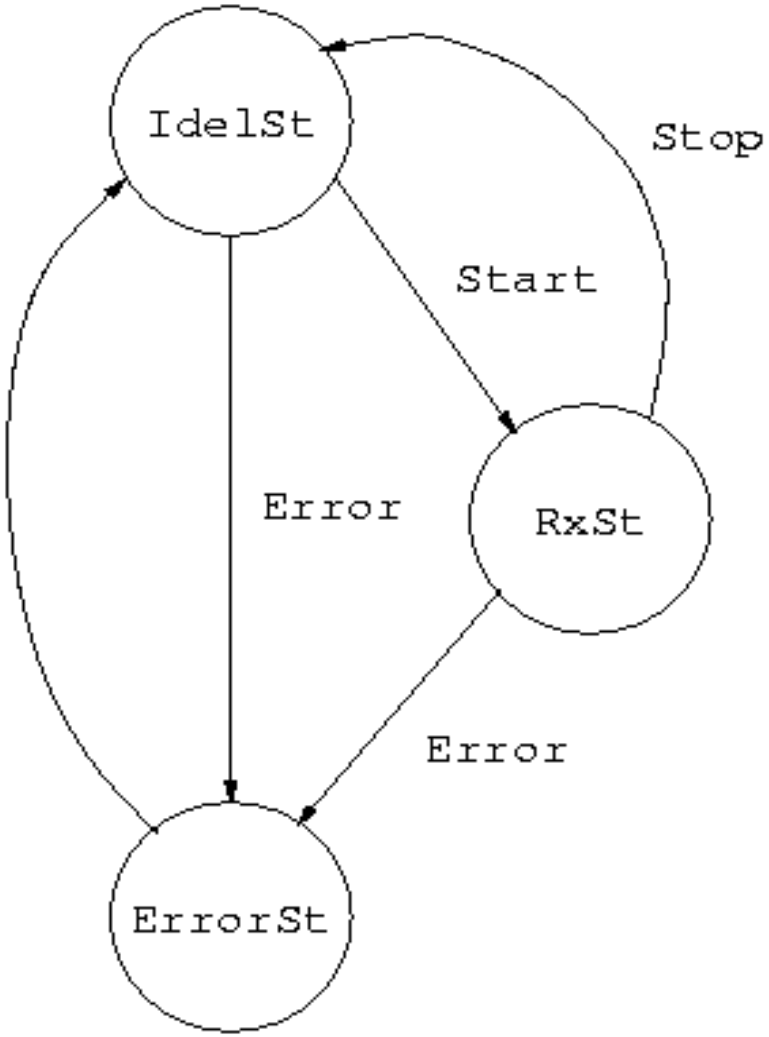
Note: smaller state machine

Output = G(state, input)

Moore vs Mealy



Example2: Transition Diagram



FSM 1/4 : CL

```
module fsmM(ReceiveSt, ErrorSt, Start,
            Stop, Error, Clk, Reset_);
//
input  Start, Stop, Error, Clk, Reset_;
output ReceiveSt, ErrorSt;
//
parameter [1:0] IdleState    = 0,
                ReceiveState = 1,
                ErrorState   = 3;
//
reg [1:0] FSMstate, nxtFSMstate;
//
always @(FSMstate or Start or Stop or Error) begin
//
   case(FSMstate)
```

FSM 2/4 : CL

```
   IdleState:
                  begin
                     if(Error) nxtFSMstate <= ErrorState;
                     else begin
                        if(Start) nxtFSMstate <= ReceiveState;
                        else nxtFSMstate <= IdleState;
                     end
                  end
//
   ReceiveState:
                  begin
                     if(Error) nxtFSMstate <= ErrorState;
                     else begin
                        if(Stop) nxtFSMstate <= IdleState;
                        else nxtFSMstate <= ReceiveState;
                     end
                  end
```

FSM 3/4: register

```
    ErrorState : nxtFSMstate <= IdleState;
    default    : nxtFSMstate <= IdleState;
    endcase
end

always @(posedge Clk) begin
    if (Reset) FSMstate <= #`dh IdleState;
    else FSMstate <= #`dh nxtFSMstate;
End
```

Mealy FSM 4/4 : outputs

```
reg  ReceiveSt;
wire SetRcvSt = (FSMstate==IdleState)&Start;
wire ClrRcvSt = (FSMstate==ReceiveState)&(Error|Stop);
//
always @(posedge Clk) begin
   if (~Reset_) ReceiveSt <= 0;
   else ReceiveSt <= (ReceiveSt | SetRcvSt)&~ClrRcvSt;
end
//
wire ErrorSt = FSMstate[1];
//
endmodule
```

Output "ReveiveSt" depends on inputs.