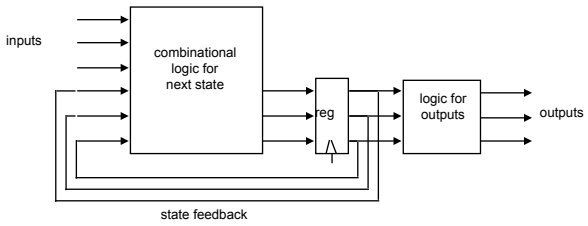
# Verilog for Finite State Machines

- Strongly recommended style for FSMs
- Works for both Mealy and Moore FSMs
- You can break the rules
  - But you have to live with the consequences

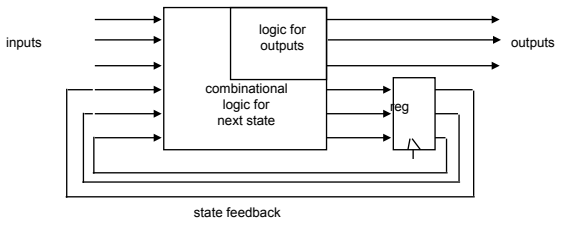


# Mealy and Moore machines

- Moore




- Mealy



## Constructing State Machines in Verilog

- We need register to hold
- the current state
  - always @(posedge clk)  block
- We need next state function
  - Where do we go from each state given the inputs
  - state by state case analysis
    - next state determined by current state and inputs
- We need the output function
  - State by state analysis
  - Moore: output determined by current state only
  - Mealy: output determined by current state and inputs



## State Register

- Declare two values
  - state : current state – output of state register
  - nxtState : next state –  input to state register
  - We rely on next state function to give us  nxtState
- Declare symbols for states with state assignment

```
localparam IDLE=0, WAITFORB=1,
           DONE=2, ERROR=3;

reg [1:0] state,    // Current state
          nxtState; // Next state
```

## State Register

- Simple code for register
  - Define reset state
  - Otherwise, just move to nxtState on clock edge

```
localparam IDLE=0, WAITFORB=1,
           DONE=2, ERROR=3;
reg [1:0] state,    // Current state
          nxtState; // Next state

always @(posedge clk) begin
  if (reset) begin
    state <= IDLE;   // Initial state
  end else begin
    state <= nxtState;
  end
end
```



## Next State Function

- Combinational logic function
  - Inputs : state, inputs
  - Output : nxtState
- We could use assign statements
- We will use an **`always @(*)`** block instead
  - Allows us to use more complex statements
  - **`if`**
  - **`case`**

## always @(*) Block

- Used for combinational logic functions
- Is always active – like assign statements
- Assignment ( = ) used to assign function value
- Output can be assigned more than once
  - e.g. multiple if statements
  - The last one counts
- ***All outputs must be assigned at least once***
  - No matter how ifs and cases are executed
  - Otherwise function is undefined
  - Use default assignments to help you



## Next State Function

- Describe what happens in each state
- Case statement is natural for this

```
always @(*) begin
  nxtState = state;  // Default next state: don't move
  case (state)
    IDLE : begin
      if (B) nxtState = ERROR;
      else if (A) nxtState = WAITFORB;
    end
    WAITFORB : begin
      if (B) nxtState = DONE;
    end
    DONE : begin
    end
    ERROR : begin
    end
  endcase
end
```

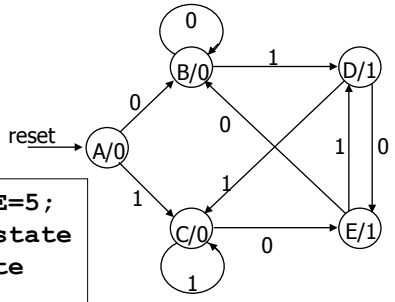
# Output Function

- Describe the output of each state

```
always @(*) begin
  nxtState = state;  // Default next state: stay where we are
  out = 0;           // Default output
  case (state)
    IDLE : begin
      if (B) nxtState = ERROR;
      else if (A) nxtState = WAITFORB;
    end
    WAITFORB : begin
      if (B) nxtState = DONE;
    end
    DONE : begin
      out = 1;
    end
    ERROR : begin
    end
  endcase
end
```



# Example #2 : Edge Detector (Moore)

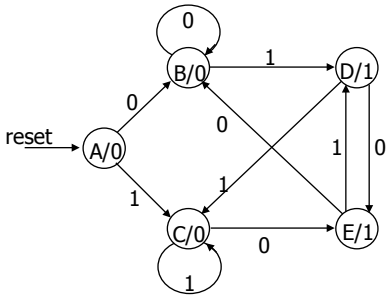



```
localparam A=0, B=1, C=2, D=4, E=5;
reg [2:0] state,    // Current state
          nxtState; // Next state

always @(posedge clk) begin
  if (reset) begin
    state <= A;   // Initial state
  end else begin
    state <= nxtState;
  end
end
```

# Example #2 : Edge Detector (Moore)

```
always @(*) begin
  nxtState = state;
  out = 0;
  case (state)
    A : if (in) nxtState = C;
        else    nxtState = B;
    B : if (in) nxtState = D;
    C : if (~in) nxtState = E;
    D : begin
      out = 1;
      if (in) nxtState = C;
      else    nxtState = E;
    end
    E : begin
      out = 1;
      if (in) nxtState = D;
      else    nxtState = B;
    end
    default : begin
      out = 1'bX;
      nxtState = 3'bX;
    end
  endcase
end
```

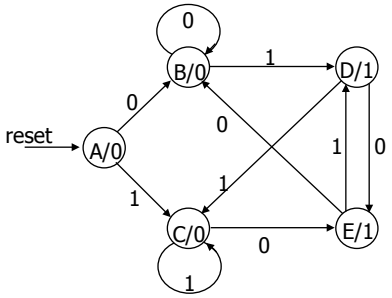




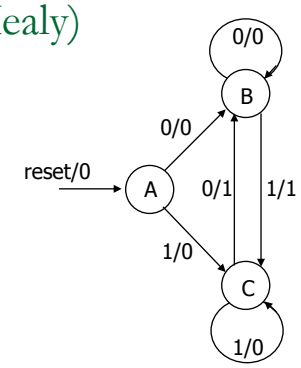

# Example #2 : Edge Detector (Moore)

- Using state assignment for output
  - Only works for Moore FSM

```
always @(*) begin
  nxtState = state;
  out = state[2];
  case (state)
    A : if (in) nxtState = C;
        else    nxtState = B;
    B : if (in) nxtState = D;
    C : if (~in) nxtState = E;
    D : if (in) nxtState = C;
        else    nxtState = E;
    E : if (in) nxtState = D;
        else    nxtState = B;
    default : nxtState = 3'bX;
  endcase
end
```



# Example #2 : Edge Detector (Mealy)



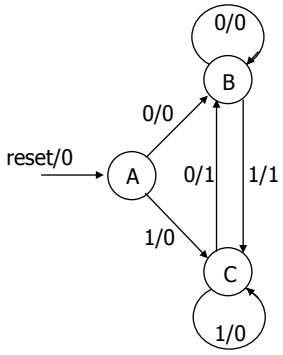

```
localparam A=0, B=1, C=2;
reg [1:0] state,    // Current state
          nxtState; // Next state

always @(posedge clk) begin
  if (reset) begin
    state <= A;   // Initial state
  end else begin
    state <= nxtState;
  end
end
```



# Example #2 : Edge Detector (Moore)

- Output depends on state and input




```
always @(*) begin
  nxtState = state;
  out = 0;
  case (state)
    A : if (in) nxtState = C;
        else    nxtState = B;
    B : if (in) begin
          out = 1;
          nxtState = C;
        end
    C : if (~in) begin
          out = 1;
          nxtState = B;
        end
    default : begin
      out = 1'bX;
      nxtState = 3'bX;
    end
  endcase
end
```

# Summary

- Please use “standard” FSM Verilog style shown here
  - Do not be beguiled into thinking this is programming C!
- always `@(posedge clk)` block
  - Use only for registers with simple logic
  - e.g. shifter, counter, enabled register, etc.
- Miscellaneous combinational logic
  - **`assign`** statements (always safe)
  - **`always @(*)`** block (be very careful)
    - Think of this as a complex **`assign`** statement
    - Fine to have more than one