State Machines

- Two types of state machine as classified by output formation
  - Moore: outputs formed by present state only
  - Mealy: outputs formed by present state and input
- Also classified by state encoding
  - Binary: 000, 001, 010, 011, 100, 101,....
  - Grey: 000, 001, 011, 010, 110, 100,....
  - One Hot: 000001, 000010, 000100, 001000, 010000,....
  - Random: 001000, 100010, 000110, 101000, 110110,....
State Machines

- A Moore-type state machine has the structure seen below.

- Present state is typically held by D-type flip-flops.
- The machine will transition to the next state as determined by the current inputs and the present state.
- The next state decoder is comprised entirely of combinatorial logic.
- The outputs are a function of the present state only.
- The outputs may also simply be the state flip-flop outputs.
State Machines

- A Mealy-type state machine has the structure seen below.

![Mealy State Machine Diagram]

- Present state is typically held by D-type flip-flops.
- The machine will transition to the next state as determined by the current inputs and the present state.
- The next state decoder is comprised entirely of combinatorial logic.
- The outputs are a function of present state and current inputs.
State Machines

- State machine diagrams document how the state machine operates
  - States are given names (enumerated types) inside the bubbles
  - Transitions between states are indicated by arcs
  - Conditions for taking the arc given by a logic equation
  - Outputs asserted in states are given by a list beside the state
  - Mealy outputs formed are described by a separate equation
  - State machines with no explicit control outputs should state that it only has "state only" outputs.
Moore state machine coding - state encoding

- State encoding done with enumerated types
- Looking up state "01001" is inefficient and leads to errors
- Enumerated states show up in the simulator output
- Declare type of state vectors directly after defining the encoding
- `arbiter_ps, arbiter_ns` are the present and next state vectors
- These are defined directly after the type declaration

```vhdl
// enumerated states
enum logic [1:0]{
    IDLE = 2'b00,
    BBUSY = 2'b01,
    BWAIT = 2'b10,
    BFREE = 2'b11,
} arbiter_ps, arbiter_ns;
```
State Machines

- Moore state machine coding - state storage
  - State storage is implemented with one `always_ff` clause
  - Inferring rising edge triggered D-FFs to hold arbiter present state
  - Asynchronous reset, active low
  - Transitions between states occur only at the clock edge

```verilog
//infer the present state vector flip flops
always_ff @(posedge clk, negedge rst_n)
  if (!rst_n) arbiter_ps <= IDLE;       //at reset, go to idle state
  else        arbiter_ps <= arbiter_ns; //otherwise, go to the next state
```
State Machines

- Moore state machine coding - next state decoder

```verilog
always_comb begin
    gnt = 1'b0; //default assignment, output signal
    unique case (arbiter_ps)
        IDLE :
            if (req) arbiter_ns = BBUSY;
            else arbiter_ns = IDLE;
        BBUSY: begin
            gnt = 1'b1; //assert gnt
            if (!done) arbiter_ns = BBUSY;
            else if (dly) arbiter_ns = BWAIT;
            else arbiter_ns = BFREE;
        end
        BWAIT: begin
            gnt = 1'b1; //assert gnt
            if (!dly) arbiter_ns = BFREE;
            else arbiter_ns = BWAIT;
        end
        BFREE:
            if (req) arbiter_ns = BBUSY;
            else arbiter_ns = IDLE;
    endcase
end //always
```
State Machines

- Arbiter synthesis results
State Machines

- Arbiter operation
State Machines

- Moore state machine coding - the whole enchilada

module arbiter0(
    output logic gnt,
    input    clk,   //clock input
    input    rst_n, //asynchronous active low reset
    input    dly,   //delay input
    input    done,  //done input
    input    req    //request input
);

//define enumerated types and vectors for ps, ns
enum logic [1:0] {
    IDLE = 2'b00,
    BBUSY = 2'b01,
    BWAIT = 2'b10,
    BFREE = 2'b11,
} arbiter_ps, arbiter_ns;

//infer the present state vector flip flops
always_ff @(posedge clk, negedge rst_n)
    if (!rst_n) arbiter_ps <= IDLE;
    else       arbiter_ps <= arbiter_ns;

always_comb begin
    gnt = 1'b0; //default, output signal
    case (arbiter_ps)
        IDLE : begin
            if (req) arbiter_ns = BBUSY;
            else    arbiter_ns = IDLE;
        end
        BBUSY: begin
            gnt = 1'b1; //assert gnt
            if (!done) arbiter_ns = BBUSY;
            else if (dly) arbiter_ns = BWAIT;
            else arbiter_ns = BFREE;
        end
        BWAIT: begin
            gnt = 1'b1; //assert gnt
            if (!dly) arbiter_ns = BFREE;
            else arbiter_ns = BWAIT;
        end
        BFREE: begin
            if (req) arbiter_ns = BBUSY;
            else arbiter_ns = IDLE;
        end
    endcase
end //always
endmodule
State Machines

- Another state machine...

```verilog
module glitch(output logic rd, ds
    input go, ws, clk, reset_n);

    enum reg [1:0]{ IDLE = 2'b00, READ = 2'b01, DLY = 2'b10,
        DONE = 2'b11} glitch_ps, glitch_ns;

    always_ff @(posedge clk, negedge reset_n)
        if (!reset_n) glitch_ps <= IDLE;
        else glitch_ps <= glitch_ns;

    always_comb begin
        rd = 1'b0; //default state
        ds = 1'b0; //default state
        unique case (glitch_ps)
            IDLE : if (go) glitch_ns = READ;
                else glitch_ns = IDLE;
            READ : begin
                rd = 1'b1;
                glitch_ns = DLY;
            end
            DLY : begin
                rd = 1'b1;
                if (!ws) glitch_ns = DONE;
                else glitch_ns = READ;
            end
            DONE : begin
                ds = 1'b1;
                glitch_ns = IDLE;
            end
        endcase
    end
endmodule
```
State Machines

- How is the output "rd" formed?
  - Which Q output changes first?
  - Which flip flop gets the clock edge first?
  - Where are the flip flops located relative to each other?
  - Which is faster Q_n or Q?
  - Which inputs on the OR or OAI gate are faster?
State Machines

- What can happen... gate simulation
  - BTW, the RTL simulation looks perfect
  - Glitches formed when going between states 10 → 01
  - Also note _ns vector transitions
State Machines

- Anytime we decode FF outputs glitches could occur
- Unequal propagation times will create the glitches
- Grey encoding can prevent this, but requires care

- Do we care?
  - In many cases, no.
  - OK as long as inputs stabilize prior to \( (\text{clock period} + \text{tsu}) \)
  - Deadly if output goes to edge sensitive circuitry, eg. FIFO, DRAM
State Machines

- To eliminate the possibility of glitches, force the outputs to come from flip-flop outputs.
- How about a "clean-up" flip-flop?
State Machines

▷ What else could we do?
  ▷ By looking at the inputs and the next state vector we know what the next state will be.
  ▷ We also know if an output should be asserted in that next state.
  ▷ If we are going to transition to a state with an asserted output, set an output flip-flop to be one at that time.
  ▷ What would this look like?
State Machines

- Add an extra always block to make the output flip-flops
  - Note that the case is evaluating _ns not _ps

```verilog
always_ff @(posedge clk, negedge reset_n)
  if (!reset_n) begin
    rd <= 1'b0;
    ds <= 1'b0;
  end
  else begin
    rd = 1'b0;
    ds = 1'b0;
    case (glitch_ns)
      READ : rd = 1'b1;
      DLY   : rd = 1'b1;
      DONE  : ds = 1'b1;
    endcase
  end
end
```
Another clearer way using the same idea
  - To the present state vector, add extra "state bits" that are actually outputs which are asserted in the correct state
  - This eliminates the extra always block
  - Outputs will only come from flip-flop outputs
  - Strip off outputs from present state vector
State Machines

- Glitchless state machine code

module no_glitch( output rd, ds,
        input  go, ws, clk, reset_n);

// state encoding outputs | present state
// ds   rd | ps[1] ps[0]
enum reg [3:0]{
    IDLE = 4'b00_00,
    READ = 4'b01_01,
    DLY = 4'b01_10,
    DONE = 4'b10_11,
} no_glitch_ns, no_glitch_ps;

always_ff @(posedge clk, negedge reset_n)
    if (!reset_n) no_glitch_ps <= IDLE;
    else no_glitch_ps <= no_glitch_ns;

always_comb begin
    unique case (no_glitch_ps)
        IDLE : if (go) no_glitch_ns = READ;
        else no_glitch_ns = IDLE;
        READ : no_glitch_ns = DLY;
        DLY : if (!ws) no_glitch_ns = DONE;
        else no_glitch_ns = READ;
        DONE : no_glitch_ns = IDLE;
    endcase
end
assign {ds,rd} = no_glitch_ps[3:2];
endmodule
State Machines

- Synthesis output from glitchless state machine
  - Outputs "rd" and "ds" come directly from flip-flop outputs
  - Penalty is slight increase in gate count (ff vs gates)