FSM and Efficient Synthesizable FSM Design using Verilog
Introduction

- There are many ways to code FSMs including many very poor ways to code FSMs.
- This lecture offers guidelines for doing efficient coding, simulation and synthesis of FSM designs.
- In this lecture multiple references are made to combinational always blocks and sequential always blocks.
Introduction

- Combinational *always* blocks are *always* blocks that are used to code combinational logic functionality and are strictly coded using *blocking assignments*. A combinational always block has a combinational sensitivity list, a sensitivity list without "posedge" or "negedge" Verilog keywords.

- Sequential *always* blocks are *always* blocks that are used to code clocked or sequential logic and are always coded using *nonblocking* assignments. A sequential always block has an edge-based sensitivity list.
A common classification used to describe the type of an FSM is Mealy and Moore state machines.

A Moore FSM is a state machine where the outputs are only a function of the present state.

A Mealy FSM is a state machine where one or more of the outputs is a function of the present state and one or more of the inputs.
Mealy & Moore FSMs (contd.)

Mealy Machine Only

Combinational Logic

Next State Logic

Next

Sequential Logic Circuit

Present State FF’s

State

Output Logic

outputs

CLK
Binary Encoded or One Hot Encoding

Binary Encoded FSM (Highly Encoded)

One Hot Encoding
A binary-encoded FSM design only requires as many flip-flops as are needed to uniquely encode the number of states in the state machine.

Number of FF

if(log2(number of states) == integer)
    required FF = log2(number of states)
else
    required FF = integer(log2(#states))+1;
Binary Encoded or One Hot Encoding

- A onehot FSM design requires a flip-flop for each state in the design and only one flip-flop (the flip-flop representing the current or "hot" state) is set at a time in a onehot FSM design.

- For a state machine with 9-16 states, a binary FSM only requires 4 flip-flops while a onehot FSM requires a flip-flop for each state in the design (9-16 flip-flops).
What constitutes an efficient FSM coding style?
Identify HDL coding goals and why they are important.
Quantify the capabilities of various FSM coding styles.
The FSM coding style should be easily modified to change state encodings and FSM styles.

The coding style should be compact.

The coding style should be easy to code and understand.

The coding style should facilitate debugging.

The coding style should yield efficient synthesis results.
Two Always Block FSM Style (Good Style)

One of the best Verilog coding styles is to code the FSM design using two always blocks, one for the sequential state register and one for the combinational next-state and combinational output logic.
module fsm_4states
    (output reg gnt,
     input dly, done, req, clk, rst_n);

parameter [1:0] IDLE = 2'b00,
    BBUSY = 2'b01,
    BWAIT = 2'b10,
    BFREE = 2'b11;
reg [1:0] state, next;

always @(posedge clk or negedge rst_n)
    if (!rst_n) state <= IDLE;
    else state <= next;
always @(state or dly or done or req) begin
next = 2'bx;
gnt = 1'b0;
case (state)
  IDLE : if (req) next = BBUSY;
          else next = IDLE;
  BBUSY: begin
          gnt = 1'b1;
          if (!done) next = BBUSY;
          else if (dly) next = BWAIT;
          else next = BFREE;
          end
  BWAIT: begin
          gnt = 1'b1;
          if (!dly) next = BFREE;
          else next = BWAIT;
          end
  BFREE: if (req) next = BBUSY;
         else next = IDLE;
endcase end endmodule
Placing a default next state assignment on the line immediately following the always block sensitivity list is a very efficient coding style. This default assignment is updated by next-state assignments inside the case statement.

There are three types of default next-state assignments that are commonly used: (1) next is set to all X's, (2) next is set to a predetermined recovery state such as IDLE, or (3) next is just set to the value of the state register.

By making a default next state assignment of X's, pre-synthesis simulation models will cause the state machine outputs to go unknown if not all state transitions have been explicitly assigned in the case statement.

This is a useful technique to debug state machine designs, plus the X's will be treated as "don't cares" by the synthesis tool.

Some designs require an assignment to a known state as opposed to assigning X's. Examples include: satellite applications, medical applications, designs that use the FSM flip-flops as part of a diagnostic scan.
One of the most common FSM coding styles in use today is the one sequential always block FSM coding style.

For most FSM designs, the one always block FSM coding style is more verbose, more confusing and more error prone than a comparable two always block coding style.
module fsm_4states
    (output reg gnt,
     input dly, done, req, clk, rst_n);

parameter [1:0] IDLE = 2'd0,
    BBUSY = 2'd1,
    BWAIT = 2'd2,
    BFREE = 2'd3;

reg [1:0] state;
One Always Block FSM Style:

```verilog
class always @(posedge clk or negedge rst_n)
  if (!rst_n) begin
    state <= IDLE;
    gnt <= 1'b0;
  end
  else begin
    state <= 2'bx;
    gnt <= 1'b0;
  end

  case (state)
    IDLE : if (req) begin
      state <= BBUSY;
      gnt <= 1'b1;
    end
    else
      ...
  endcase
```

BBUSY: if (!done) begin
    state <= BBUSY;
gnt <= 1'b1;
end
else if (dly) begin
    state <= BWAIT;
gnt <= 1'b1;
end
else state <= BFREE;

BWAIT: if (dly) begin
    state <= BWAIT;
gnt <= 1'b1;
end
else state <= BFREE;

BFREE: if (req) begin
    state <= BBUSY;
gnt <= 1'b1;
end
else state <= IDLE;

endcase
end
endmodule
Onehot FSM Coding Style (Good Style)

- Efficient (small and fast) onehot state machines can be coded using an inverse case statement; a case statement where each case item is an expression that evaluates to true or false.
- Reconsider the fsm_4state design shown.
- The key to understanding the changes is to realize that the parameters no longer represent state encodings, they now represent an index into the state vector, and comparisons and assignments are now being made to single bits in either the state or next-state vectors.
- Notice how the case statement is now doing a 1-bit comparison against the onehot state bit.
module fsm_cc4_fp  
(output reg gnt, 
ininput dly, done, req, clk, rst_n); 

parameter [3:0] IDLE = 0, 
             BBUSY = 1, 
             BWAIT = 2, 
             BFREE = 3; 

reg [3:0] state, next; 

always @(posedge clk or negedge rst_n) 
    if (!rst_n) begin 
        state <= 4'b0; 
        state[IDLE] <= 1'b1; 
    end 
    else state <= next; 

always @(state or dly or done or req) begin 
    next = 4'b0; 
    gnt = 1'b0; 
    case (1'b1) // ambit synthesis case = full, parallel 
        state[IDLE] : if (req) next[BBUSY] = 1'b1; 
                      else next[IDLE] = 1'b1; 
        state[BBUSY] : begin 
                       gnt = 1'b1; 
                       if (!done) next[BBUSY] = 1'b1; 
                       else if (dly) next[BWAIT] = 1'b1; 
                       else next[BFREE] = 1'b1; 
        end 
        state[BWAIT] : begin 
                       gnt = 1'b1; 
                       if (!done) next[BWAIT] = 1'b1; 
                       else if (dly) next[BBUSY] = 1'b1; 
                       else next[BFREE] = 1'b1; 
        end 
end
Onehot FSM Coding Style

```verilog
state[BBUSY]: begin
    gnt = 1'b1;
    if (!done) next[BBUSY] = 1'b1;
    else if (dly) next[BWAIT] = 1'b1;
    else next[BFREE] = 1'b1;
end

state[BWAIT]: begin
    gnt = 1'b1;
    if (!dly) next[BFREE] = 1'b1;
    else next[BWAIT] = 1'b1;
end

state[BFREE]: begin
    if (req) next[BBUSY] = 1'b1;
    else next[IDLE] = 1'b1;
endcase
end
endmodule
```
This is the only coding style where one should use full_case and parallel_case statements.

The parallel case statement tells the synthesis tool to not build a priority encoder even though in theory, more than one of the state bits could be set.

(as engineers, we should know that this is a onehot FSM and that only one bit can be set so no priority encoder is required).
The Only Change one has to do is

```verilog
counter always @(posedge clk or negedge rst_n)
if (!rst_n) gnt <= 1'b0;
else begin
  gnt <= 1'b0;
  case (next)
  IDLE, BFREE: ; // default outputs
  BBUSY, BWAIT: gnt <= 1'b1;
  endcase
end
endmodule
```
Thanks

- Courtesy – International Cadence User Group 2002