Centurion
UNIVERSITY
Shaping Lives...
Empowering Communities...

Architecture Area, Speed & Power

Centurion
UNIVERSITY
Shaping Lives
Empowering Communities

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*.

Two Always Block FSM Style (Good Style)

Centurion University
Shaping Lives...
Empowering Communities...

```
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;
```

Two Always Block FSM Style (Good Style)

Centurion University
Shaping Lives...
Empowering Communities...

```
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
```

Centurion
UNIVERSITY
Shaping Lives...
Empowering Communities

Making default *next* equal all X's assignment

- 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

Centurion
UNIVERSITY
Shaping Lives
Empowering Communities

One Always Block FSM Style (Avoid This Style!)

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.

Centurion
UNIVERSITY
Shaping Lives...
Empowering Communities...

One Always Block FSM 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:

Centurion UNIVERSITY
Shaping Lives...
Empowering Communities...

```
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;

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

One Always Block FSM Style

Centurion University
Shaping Lives...
Empowering Communities...

```
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
```

Centurion UNIVERSITY
Shaping Lives... Empowering Communities...

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.

Onehot FSM Coding Style

```
module fsm_cc4_fp
  (output reg gnt,
   input 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
```

Index into the state register, not state encodings

Onehot requires larger declarations

Reset modification

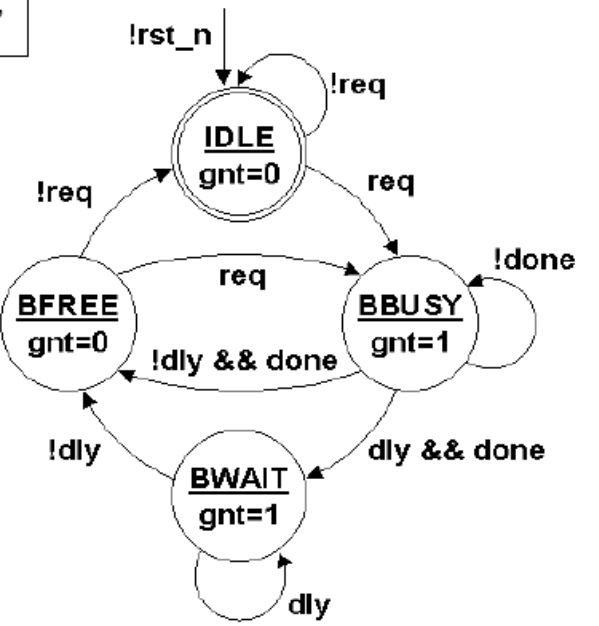
Must make all-0's assignment

case (1'b1)

Add "full" & "parallel" case

state[*current_state*] case items

Only update the next[*next_state*] bit



Onehot FSM Coding Style

Centurion University
Shaping Lives...
Empowering Communities...

```
            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;
                          end
          endcase
        end
      endmodule
```

state[*current_state*] case items

Only update the next[*next state*] bit

Centurion
UNIVERSITY
Shaping Lives...
Empowering Communities...

Onehot FSM Coding Style

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).

Centurion
UNIVERSITY
Shaping Lives...
Empowering Communities...

Registered FSM Outputs (Good Style)

The Only Change one has to do is

```
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
```