RTL Coding Guidelines
Think Synchronous

- *Synchronous* designs run smoothly through synthesis, simulation and place-and-route
- Isolate necessary *Asynchronous* logic into separate blocks

Asynchronous Address Decoder

How am I going to synthesize this?
RTL Coding Guidelines
Think RTL

- Describe the circuits in terms of its registers and the combinational logic between them

module GIZMO (A, CLK, Z);
...
always @(A) begin : COMBO1...
always @(posedge CLK)...
always @(B) begin : COMBO2...
always @(posedge CLK) ...
end module;
RTL Coding Guidelines
Separate Combinational from Sequential

- Follows RTL coding style
- Easy to read and self-documenting

module EXAMPLE (DATA1, DATA2, CLK, Q)
input DATA1, DATA2, CLK;
output Q;
reg DATA, Q;
always @(DATA1 or DATA2)
begin: COMBO
    DATA <= GOBBLEYGOOK(DATA1, DATA2);
end
always @(posedge CLK)
begin: SEQUENTIAL
    Q <= DATA;
end
endmodule

Combinational Logic
Sequential Logic
IF Statements

- **IF statements infer multiplexer logic**

```vhdl
always @(SEL or A or B)
  if (SEL)
    D <= A;
  else
    D <= B;
```

- **Latches are inferred unless all variables are assigned in all branches**

```vhdl
always @(SEL or A)
  if (SEL)
    D <= A;
```

![Diagram of multiplexer logic](image1)

![Diagram of latch logic](image2)
IF Statements (cont.)

- IF-ELSE-IF statements infer priority-encoded multiplexers

```verilog
always @(SEL or A or B or C or D)
    if (SEL[2] == 1'b1)
        OUT <= A;
    else if (SEL[1] == 1'b1)
        OUT <= B;
    else if (SEL[0] == 1'b1)
        OUT <= C;
    else
        OUT <= D;
```

Long delay from D to OUT
IF Statements (cont.)

- Remove redundant conditions
- Use CASE statements if conditions are mutually exclusive

Don’t

```vhdl
always @(A or B or C or D or E)
  if (A < B)
    OUT <= C;
  else if (A > B)
    OUT <= D;
  else if (A == B)
    OUT <= E;
```

Do

```vhdl
always @(A or B or C or D or E)
  if (A < B)
    OUT <= C;
  else if (A > B)
    OUT <= D;
  else
    OUT <= E;
```
CASE Statements
Verilog Directives

- **full_case** indicates that all *user-desired* cases have been specified
- Do not use *default* for one-hot encoding

```verilog
always @(SEL or A or B or C)
begin
  case (SEL) //synopsys full_case
    3'b001 : OUT <= A;
    3'b010 : OUT <= B;
    3'b100 : OUT <= C;
  endcase
end
```

Does not infer latches

```verilog
always @(SEL or A or B)
begin
  case (SEL)
    3'b001 : OUT <= A;
    3'b010 : OUT <= B;
    3'b100 : OUT <= C;
  endcase
end
```

Infers latches for OUT because not all cases are specified
CASE Statements
Verilog Directives (cont.)

- `parallel_case` indicates that all cases listed are mutually exclusive to prevent priority-encoded logic.
CASE Statements
“CASE” vs. “IF-ELSE IF”

- Use IF-ELSE for 2-to-1 multiplexers
- Use CASE for n-to-1 multiplexers where n > 2
- Use IF-ELSE IF for priority encoders
- Use CASE with //synopsys parallel_case when conditions are mutually exclusive
- Use CASE with //synopsys full_case when not all conditions are specified
- Use CASE with //synopsys full_case parallel_case for one-hot Finite State Machines (FSMs)
CASE Statements
FSM Encoding

- Use CASE statements to describe FSMs
- Use //synopsys parallel_case to indicate mutual exclusivity
- Use //synopsys full_case when not all possible states are covered (one-hot)
- Do not use default unless recovery state is desired
CASE Statements
FSM Encoding (cont.)

- Use parameter statements to define state values
- Use CASE statements and // synopsys parallel_case full_case to describe FSM

```verilog
module EXAMPLE (RESET, CLK, OUT);
input  RESET, CLK;
output  [1:0] OUT;
parameter IDLE=4'b0001, GO=4'b0010, YIELD=4'b0100,
STOP=4'b1000;
reg [3:0] CURRENT_STATE, NEXT_STATE;
always @(CURRENT_STATE)
begin: COMBO
  case (CURRENT_STATE) // synopsys full_case parallel_case
    IDLE: begin NEXT_STATE = GO; OUT <= 2'b01; end
    GO: begin NEXT_STATE = YIELD; OUT <= 2'b11; end
    YIELD: begin NEXT_STATE = STOP; OUT <= 2'b10; end
    STOP: begin NEXT_STATE = IDLE; OUT <= 2'b00; end
  endcase
end
always @(posedge CLK or negedge RESET)
begin: SEQUENTIAL
  if (~RESET)
    CURRENT_STATE <= IDLE;
  else
    CURRENT_STATE <= NEXT_STATE
end
endmodule
```
CASE Statements
Watch for Unintentional Latches

- Completely specify all branches for every case and if statement
- Completely specify all outputs for every case and if statement
- Use //synopsys full_case if all desired cases have been specified

What’s wrong with this example?

```
always @(SEL)
begin
  case (SEL)
  2'b00: A <= 1'b1;
  2'b01: A <= 1'b0;
  2'b10: B <= 1'b1;
  // Missing Case
endcase
end
```

(Missing Case)

(Missing Outputs)
Using cascade chains improves QoR significantly for multiplexers

Completely specify all possible cases for cascade chains to be inferred

```verilog
always @(SEL)
begin
  case (SEL)
    3'b000: OUT <= A;
    3'b001: OUT <= B;
    3'b010: OUT <= C;
    3'b011: OUT <= D;
    3'b100: OUT <= E;
    3'b101: OUT <= F;
    3'b110: OUT <= G;
    3'b111: OUT <= H;
  endcase
end
```
Multiplexers

- Use IF or continuous assignment when select is a single-bit signal

```vhdl
always @(SEL or A or B)
  if (SEL)
    D <= A;
  else
    D <= B;
-----------------------
assign D = SEL ? A : B;
```

- Use CASE statements when select is a multi-bit bus

```vhdl
always @(SEL or A or B or C or D)
begin
  case (SEL)
    2'b00 : OUT <= A;
    2'b01 : OUT <= B;
    2'b10 : OUT <= C;
    2'b11 : OUT <= D;
  endcase
end
```
Operators

- **Operators inferred from HDL**
  - Adder, Subtractor, AddSub (+, -), Multiplier (*)
  - Comparators (>, >=, <, <=, ==, !=)
  - Incrementer, Decrementer, IncDec (+1, -1)

- **Example**

```verilog
module add (sum, a, b);
  output [15:0] sum;
  input [15:0] a, b;
  assign sum = a + b + 1'b1;
endmodule
```

Design indicates two adders.

```verilog
module add (sum, a, b);
  output [15:0] sum;
  input [15:0] a, b;
  wire temp;
  assign {sum, temp} = {a, 1'b1} + {b, 1'b1};
endmodule
```

FE infers one adder with carry chain.
Operators
Operator Sharing

- Operators can be shared within an *always* block by default
- Users can disable sharing

```vhdl
always @(SEL or A or B or C)
begin
  if (SEL) then
    Z = A+B;
  else
    Z = A+C;
  end
end
```

**With Sharing**

**Without Sharing**

- Smaller
- Larger
Operators
Operator Balancing

- Use parenthesis to guide synthesis

\[ (A \times B) \times (C \times D) \quad \text{vs.} \quad A \times B \times C \times D \]