


EE461 Verilog-HDL

Week 10 Logic Synthesis

Alex Yang, Engineering School, NPU

Week 10 Outlines

- Logic Synthesis
  - Synthesis supported/non-supported verilog structures
  - Directives: full case/parallel case
  - Coding Guideline

Synthesis supported/non-supported

- synthesis supported
  - Ports: input, output, inout
  - wire, reg, tri
  - parameter
  - continuous assignment
  - always
  - task(w/o timing info) & function
  - for loop, if-else(if-else if-else), case

- synthesis non-supported
  - === , !==
  - / (division)
  - % (modulus)
  - real, time
  - arrays of integer
  - defparam
  - delay (like #10)
  - assign & deassign/ force & release
  - primitive
  - initial block
  - repeat, forever, while
  - wait
  - fork / join
  - event

full_case/parallel_case

- "full_case" Directive in Case Statement
  - "full" in case block without "//synopsys full_case"
  - "full" in case block with "//synopsys full_case"

```
module ex(y, a, b, c, sel);
   output       y;
   input [1:0]  sel;
   input        a, b, c;
   reg y;

always @(a or b or c or sel)
      case (sel)     // synopsys full_case
           2'b00: y = a;
           2'b01: y = b;
           2'b10: y = c;
     endcase
endmodule
```

directive: // synopsys full_case

For verilog simulator: it is a comment

For logic synthesizer: it is to tell that all outputs(y in above code) are equal to x for non-covered case, such as sel=2'b11.

full_case/parallel_case

- "full_case" Directive in Case Statement
  - "not full" in case block w/o "//synopsys full_case"

```
module ex(y, a, b, c, sel);
   output       y;
   input [1:0]  sel;
   input        a, b, c;
   reg y;

always @(a or b or c or sel)
      case (sel)
            2'b00: y = a;
            2'b01: y = b;
            2'b10: y = c;
     endcase
endmodule
```

No "// synopsys full_case" &w/o default:

It will be inferred a latch for sure.

full_case/parallel_case

- "full_case" Directive in Case Statement
  - "full" in case block w/o "//synopsys full_case"

```
module ex(y, a, b, c, sel);
   output       y;
   input [1:0]  sel;
   input        a, b, c;
   reg y;

always @(a or b or c or sel)
      case (sel)
            2'b00: y = a;
            2'b01: y = b;
            2'b10: y = c;
            //default:  y = 1'bx;
            default:    y = 1'b0;
      endcase
endmodule
```

No "// synopsys full_case" with default:

No latch inferred for y=1'bx, but mismatch exists between synthsis(x in synthesis: don't care; mininization involved) and simulation(x in simulation: unknow; logic operation involved, such as b=a&~a => b=0 but a=1'bx => b=1'bx).

Correction default statement: all outputs = const. or one of values in other case statement

full_case/parallel_case

- "full_case" Directive in Case Statement
  - "full" in case block w/o "//synopsys full_case"

```
module ex(y, a, b, c, sel);
   output       y;
   input [1:0]  sel;
   input        a, b, c;
   reg y;

always @(a or b or c or sel)
      y = 1'b0;
      case (sel)
            2'b00: y = a;
            2'b01: y = b;
            2'b10: y = c;
     endcase
endmodule
```

no "// synopsys full_case" & no default:

No latch inferred in case statement with output values assignment to const. or one of values in other case statements.

It is perfect match between synthesis and simulation.

full_case/parallel_case

- "full_case" Directive in Case Statement
  - "not full" in case block w/ "//synopsys full_case"

```
module ex(y, a, b, c, sel);
   output       y;
   input [1:0]  sel;
   input        a, b, c;
   reg y;

always @(a or b or c or sel)
      case (sel)  // synopsys full_case
           2'b00: y = a;
           2'b01: y = b;
           2'b10: y = c;
     endcase
endmodule
```

"// synopsys full_case" & not full case

No latch inferred after synthesis (gate level module), but mismatch exists between synthsis and simulation(a latch inferred in RTL level module).

full_case/parallel_case

- "full_case" Directive in Case Statement
  - "full" in case block w/ "//synopsys full_case"

```
module addrDecode1a (
                mce0_n,
                mce1_n,
                rce_n, addr);
   output     mce0_n, mce1_n,
rce_n;
   input [31:30]    addr;
   reg          mce0_n, mce1_n,
rce_n;

  always @(addr)begin
     casez (addr) // synopsys full_case
        2'b10: {mce1_n, mce0_n} = 2'b10;
        2'b11: {mce1_n, mce0_n} = 2'b01;
        2'b0?:                  rce_n = 1'b0;
     endcase
  end
endmodule
```

"// synopsys full_case" & full case

latch inferred after synthesis (gate level module), because not all outputs are assigned values.

full_case/parallel_case

- "full_case" Directive in Case Statement
  - Myth (Wrong): "//synopsys full_case" could remove all latches.
  - Truth: The easiest way to eliminate latches is to make initial default value assignments to all outputs immediately beneath the sensitivity list, before executing the case statement.

full_case/parallel_case

- "full_case" Directive in Case Statement

Correct coding style:

```
module addrDecode1d (mce0_n, mce1_n, rce_n, addr);
        output      mce0_n, mce1_n, rce_n;
        input [31:30]  addr;
        reg         mce0_n, mce1_n, rce_n;

        always @(addr) begin
                // Assign values to all outputs
                {mce1_n, mce0_n, rce_n} = 3'b111;
                casez (addr)
                        2'b10: {mce1_n, mce0_n} = 2'b10;
                        2'b11: {mce1_n, mce0_n} = 2'b01;
                        2'b0?: rce_n = 1'b0;
                endcase
        end
endmodule
```

full_case/parallel_case

- "parallel_case" Directive
  - A "parallel" case statement is a case statement in which it is only possible to match a case expression to one and only one case item. If it is possible to find a case expression that would match more than one case item, the matching case items are called "overlapping" case items and the case statement is not "parallel."

full_case/parallel_case

- "parallel_case" Directive in Case
  - "non parallel" in case block w/o "//synopsys parallel_case"

```
module intctl1a (int2, int1, int0, irq);
   output int2, int1, int0;
   input [2:0] irq;
   reg int2, int1, int0;
   always @(irq) begin
      { int2, int1, int0} = 3'b0;
      casez (irq)
         3'b1??: int2 = 1'b1;
         3'b?1?: int1 = 1'b1;
         3'b??1: int0 = 1'b1;
      endcase
  end
endmodule
```

No "// synopsys parallel_case":

Simulation: a priority encoder

Sythesis: inferred a priority encoder

full_case/parallel_case

- "parallel_case" Directive in Case
  - "non parallel" in case block w/ "//synopsys parallel_case"

```
module intctl1a (int2, int1, int0, irq);
  output int2, int1, int0;
  input [2:0] irq;
  reg int2, int1, int0;
  always @(irq) begin
    { int2, int1, int0} = 3'b0;
    casez (irq) // synopsys parallel_case
      3'b1??: int2 = 1'b1;
      3'b?1?: int1 = 1'b1;
      3'b??1: int0 = 1'b1;
    endcase
  end
endmodule
```

w/ "// synopsys parallel_case":

Simulation: a priority encoder

Sythesis: non-priority encoder

mismatch between both

full_case/parallel_case

- "parallel_case" Directive in Case
  - "parallel" in case block w/o "//synopsys parallel_case"

```
module intctl1a (int2, int1, int0, irq);
   output int2, int1, int0;
   input [2:0] irq;
   reg int2, int1, int0;
   always @(irq) begin
      { int2, int1, int0} = 3'b0;
      casez (irq)
         3'b1??: int2 = 1'b1;
         3'b01?: int1 = 1'b1;
         3'b001: int0 = 1'b1;
      endcase
  end
endmodule
```

w/o "// synopsys parallel_case":

Simulation: non-priority encoder

Sythesis: a priority encoder

mismatch between both

full_case/parallel_case

- "parallel_case" Directive in Case
  - "parallel" in case block w/ "//synopsys parallel_case"

```
module intctl1a (int2, int1, int0, irq);
  output int2, int1, int0;
  input [2:0] irq;
  reg int2, int1, int0;
  always @(irq) begin
    { int2, int1, int0} = 3'b0;
    casez (irq) // synopsys parallel_case
      3'b1??: int2 = 1'b1;
      3'b01?: int1 = 1'b1;
      3'b001: int0 = 1'b1;
    endcase
  end
endmodule
```

w/ "// synopsys parallel_case":

Simulation: non-priority encoder

Sythesis: a priority encoder

mismatch between both

Redundancy: "synopsys parallel_case"

full_case/parallel_case

- Summary
  - Conclusion: "full_case" and "parallel_case" directives are most dangerous when they work! It is better to code a full and parallel case statement than it is to use directives to make up for poor coding practices.

full_case/parallel_case

- Coding Style Guidelines
  - Exercise caution when using the Verilog casez statement in RTL design
  - Do not use casex in RTL design
  - In general, do not use "full_case parallel_case" directives
  - There are exceptions, but you have to know what you're doing if you plan to add "full_case parallel_case" directives.
  - Coding with case statements is recommended when a truth-table-like structure makes the Verilog code more concise and readable.
  - only use full_case parallel_case to optimize onehot FSM designs.
  - When coding a case statement with "don't cares," use a casez statement and use "?" characters instead of "z" characters in the case items to indicate "don't care" bits.

Coding Guidelines

- Look at word file