case Statement

- Nesting if past two levels is error prone and possibly inefficient.
- case excels when many tests are performed on the same expression.
- case works well for muxes, decoders, and next state logic.

CASE Statement Structure

case (case_expression)
    case_item1 : case_item_statement1;
    case_item2 : case_item_statement2;
    case_item3 : case_item_statement3;
    case_item4 : case_item_statement4;
    default    : case_item_statement5;
endcase

case is shorthand to describe a complicated if/then/else structure.
case Statement

//"full case" case statement
module case1 (
    input [7:0] a_in, b_in, c_in, d_in,
    input [1:0] sel,
    output reg [7:0] d_out);

    always_comb
    case (sel)
        2'b00 : d_out = a_in;
        2'b01 : d_out = b_in;
        2'b10 : d_out = c_in;
        2'b11 : d_out = d_in;
    endcase
endmodule
case Statement

Is the simulation correct?

//"full case" case statement
module case1 (  
    input [7:0] a_in, b_in, c_in, d_in,  
    input [1:0] sel,  
    output reg [7:0] d_out);

    always_comb  
    case (sel)  
        2'b00 : d_out = a_in;  
        2'b01 : d_out = b_in;  
        2'b10 : d_out = c_in;  
        2'b11 : d_out = d_in;  
    endcase  
endmodule
case Statement

Use default case to propagate "x"

//"full case" case statement
module case1 (  
    input [7:0] a_in, b_in, c_in, d_in,  
    input [1:0] sel,  
    output reg [7:0] d_out);

always_comb  
    case (sel)  
        2’b00 : d_out = a_in;  
        2’b01 : d_out = b_in;  
        2’b10 : d_out = c_in;  
        2’b11 : d_out = d_in;  
        default : d_out = 8’bx;
    endcase
endmodule
//incomplete case statement
module case2 (
    input [7:0] a_in, b_in, c_in,
    input [1:0] sel,
    output reg [7:0] d_out);

always_comb
    case (sel)
        2’b00 : d_out = a_in;
        2’b01 : d_out = b_in;
        2’b10 : d_out = c_in;
    endcase
endmodule

Inferred memory devices in process in routine case2 line 6 in file 'case2.sv'.
===========================================================================
<table>
<thead>
<tr>
<th>Register Name</th>
<th>Type</th>
<th>Width</th>
<th>Bus</th>
<th>MB</th>
<th>AR</th>
<th>AS</th>
<th>SR</th>
<th>SS</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>d_out_reg</td>
<td>Latch</td>
<td>8</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
===========================================================================
Warning: Netlist for always_comb block contains a latch. (ELAB-974)
Presto compilation completed successfully.
//incomplete case statement
//with default case_item
module case2 (  
    input [7:0] a_in, b_in, c_in,  
    input [1:0] sel,  
    output reg [7:0] d_out);

always_comb  
    case (sel)  
    2'b00 : d_out = a_in;  
    2'b01 : d_out = b_in;  
    2'b10 : d_out = c_in;  
    default : d_out = 8'bx;  
endcase
endmodule

Does RTL and gate simulation differ when sel = 2'b11?
case Statement

System Verilog priority Modifier
▶ SystemVerilog introduced two case and if statement modifiers
  ▶ priority
  ▶ unique
▶ Both give information to synthesis to aid optimization.
▶ Both are assertions (simulation error reporting mechanisms)
▶ Both imply that there is a case_item for all the possible legal values that case_expression might assume.
▶ Priority
  ▶ Priority tells the simulator that if all is well, you will find a match
  ▶ If a match is not found at run-time, emit an error or warning.
  ▶ Since all legal combinations are listed, synthesis is free to optimize any other unlisted case_items since they are effectively "don’t-cares".
  ▶ Priority does not guarantee removal of unintended latches.
//incomplete case statement
//with systemverilog priority modifier
module case2 (  
    input [7:0] a_in, b_in, c_in,  
    input [1:0] sel,  
    output reg [7:0] d_out);  

always_comb  
    priority case (sel)  
        2’b00 : d_out = a_in;  
        2’b01 : d_out = b_in;  
        2’b10 : d_out = c_in;  
    endcase  
endmodule

Warning: case2.sv:7: Case statement marked priority does not cover all possible conditions. (VER-504)
case Statement

System Verilog priority Modifier

▶ OK, so what happens in simulation with 2’b11?
▶ No x propagation in RTL simulation, but warning is emitted
▶ Go back, fix the error that caused the unallowed case to occur.
▶ After fixing, the RTL and gate simulation should match.
case Statement

//incomplete case statement
//with systemverilog priority modifier
module case2 (
    input [7:0] a_in, b_in, c_in,
    input [1:0] sel,
    output reg [7:0] d_out);

always_comb
    priority case (sel)
        2'b00 : d_out = a_in;
        2'b01 : d_out = b_in;
        2'b10 : d_out = c_in;
endcase
endmodule

** Warning: (vsim-8315) case2.sv(7): No condition is true in the unique/priority if/case statement.
** Warning: (vsim-8315) case2.sv(7): No condition is true in the unique/priority if/case statement.
case Statement

System Verilog priority Modifier
- Bottom line...
  - If case is full, use default expression.
  - If case is not full, use priority modifier.
casez Statement

- casez is a special version of the case expression
- Allows don’t care bits in the comparison
- Uses ? or Z values as don’t care instead of as a logic value (weird!)
- Recommendation: use ”?” as don’t care
- Use caution when using this statement

casez Example

```verilog
module interrupt_ctl1 (output reg int2, int1, int0, input [2:0] irq);

always_comb begin
  {int2, int1, int0} = 3'b0; //default
  casez (irq)
    3'b1?? : int2 = 1'b1;
    3'b?1? : int1 = 1'b1;
    3'b??1 : int0 = 1'b1;
  endcase
end
endmodule
```
## casez Statement

### casez Example

module interrupt_ctl1 (  
    output reg int2, int1, int0,  
    input [2:0] irq );  
always_comb begin  
    {int2, int1, int0} = 3’b0; //default  
    casez (irq)  
        3’b1?? : int2 = 1’b1;  
        3’b?1? : int1 = 1’b1;  
        3’b??1 : int0 = 1’b1;  
    endcase  
end
casez Statement

Parallel or Unique encoding

- If the irq bus has any of the values: 3'b011, 3'b101, 3'b110 or 3'b111, more than one case item could match the irq value.
- This is known as a non-parallel case statement.
- Synthesis will produce a priority encoder structure.
- If code could be rewritten to have parallel or unique case items, no priority structure would be needed. Thus, faster/smaller implementation.
- We indicate this situation with the modifier unique.
module interrupt_ctl2a (  
    output reg int2, int1, int0,  
    input [2:0] irq );  
always_comb begin  
    {int2, int1, int0} = 3'b0; //default  
    unique casez (irq)  
    3'b1?? : int2 = 1'b1;  
    3'b01? : int1 = 1'b1;  
    3'b001 : int0 = 1'b1;  
    endcase  
end
casez Statement

Now what?

▶ Further work has shown unique to be of at most little benefit.
▶ I only got it to work properly once.
▶ Synthesis does not stick strictly to priority encoding, its smarter.
▶ Using unique, I get about 10% decrease in delay and area at best.
▶ At this point, until I can bet a better handle on it, avoid unique.
One more case expression exists: `casex`.
Can be useful for testbenches.
Current recommendations are to not use it in synthesizable code.