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.
- SVerilog case has implied "break" statement unlike "C".
- case items are not-necessarily non-overlapping.

```verilog
case (case_expression)
  case_item1 : case_item_statement1;
  case_item2 : begin
    case_item_statement2a;
    case_item_statement2b;
    case_item_statement2c;
    end
  case_item3 : case_item_statement3;
  default    : case_item_statement5;
endcase
```
case Statement

//8-wide, 4:1 multiplexer
module case1 (  
    input [7:0] a_in, b_in, c_in, d_in,
    input [1:0] sel,
    output logic [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
Is the simulation correct? What do we do with the "X"?

// 8-wide, 4:1 multiplexer
module case1 (
    input [7:0] a_in, b_in, c_in, d_in,
    input [1:0] sel,
    output logic [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**

We can use a *default* to propagate the "X". What does this do for us?

```verilog
module case1 (  
    input [7:0] a_in, b_in, c_in, d_in,  
    input [1:0] sel,  
    output logic [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
```

<table>
<thead>
<tr>
<th>case1/a_in</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>case1/b_in</td>
<td>20</td>
</tr>
<tr>
<td>case1/c_in</td>
<td>30</td>
</tr>
<tr>
<td>case1/d_in</td>
<td>40</td>
</tr>
<tr>
<td>case1/sel</td>
<td>0</td>
</tr>
<tr>
<td>case1/d_out</td>
<td>10</td>
</tr>
</tbody>
</table>
// incomplete case, 8-wide, 3:1 mux
module case2 (  
    input [7:0] a_in, b_in, c_in,  
    input [1:0] sel,  
    output logic [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
case Statement

//incomplete case statement
//with default case_item
module case2(
    input [7:0] a_in, b_in, c_in,
    input [1:0] sel,
    output logic [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? What does synthesis do with the 8’bx?
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.
Unique Case

- Unique is an *assertion* which implies:
  - All possible values of `case_expression` are in the `case_items`
  - At simulation run time, a match must be found in `case_items`
  - At run time, only one match will be found in `case_items`

- Unique guides synthesis
  - It indicates that no priority logic is necessary
  - This produces parallel decoding which may be smaller/faster

- Unique usage
  - Use when each `case_item` is unique and only one match should occur.
  - Using a "default" `case_item` removes the testing for non-existent matches, but the uniqueness test remains.
case Statement - unique and priority

Priority Case

- Priority is an *assertion* which implies:
  - All possible values for `case_expression` are in `case_items`
- Priority guides synthesis
  - It indicates that all other testable conditions are don’t cares and may be used to simplify logic
  - This produces logic which is possibly smaller/faster
- Priority usage
  - Use to explicitly say that priority is important even though the Verilog case statement is a priority statement.
  - Using a "default" `case_item` will cause priority requirement to be dropped since all cases are available to be matched.
  - Use of a "default" also indicates that more than one match in `case_item` is OK.
- Priority is a bad name. Case is already a priority structure.
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 logic [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

Synthesis reminds us of the implicit claims made by priority:

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

System Verilog priority Modifier
▶ OK, so now what happens now when sel is 2'b11?
▶ RTL Simulator issues a run-time warning.
▶ Go back, fix the RTL error that caused the disallowed case to occur.
▶ After fixing, the RTL and gate simulation will match.
case Statement

// incomplete case priority modifier
module case2 (
    input     [7:0] a_in, b_in, c_in,
    input     [1:0] sel,
    output logic [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

Simulator issues warning at sel==2’11, and with sel==2’1x

** 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.
casez Statement

- casez is a special version of the case expression
- Allows don’t care’s in *case expression* or *case item*
- casez uses ? or Z as don’t care instead of as a logic value
- Recommendation: use ”?” as don’t care, not ”Z”.
- For example:
  - case item 2b1? can match the case expressions: 2b10, 2b11, 2b1x, or 2b1z
- casez has overlapping case items. If more than one case item matches a case expression, the first matching case item has priority.
- Use caution when using this statement
This is a interrupt priority encoder. Simultaneous interrupt requests may be asserted, but returns only the highest priority request.

```verilog
module priority_encoder_casez (  
    input [7:0] irq, //interrupt requests  
    output logic [3:0] highest_pri); //encoded highest priority interrupt  
always_comb begin  
    priority casez (irq)  
        8'b1??????? : highest_pri = 4'h8; //interrupt 8  
        8'b?1?????? : highest_pri = 4'h7; //interrupt 7  
        8'b??1????? : highest_pri = 4'h6; //interrupt 6  
        8'b???1???? : highest_pri = 4'h5; //interrupt 5  
        8'b????1??? : highest_pri = 4'h4; //interrupt 4  
        8'b?????1?? : highest_pri = 4'h3; //interrupt 3  
        8'b??????1? : highest_pri = 4'h2; //interrupt 2  
        8'b???????1 : highest_pri = 4'h1; //interrupt 1  
        default : highest_pri = 4'h0; //no interrupts  
    endcase  
endendmodule
```
case x Statement

- One more case expression exists: case x.
- Can be useful for testbenches.
- Current recommendations are to not use it in synthesizable code.