case Statement

- case also creates combinatorial logic and is easier to read when if statements are nested more than three deep or when many checks are made against the same expression.
- When the case selection item matches the case expression, the corresponding assignments are made.

```haskell
case (case_expression)
  case_selection_item1 : case_item_statement1;
  case_selection_item2 : begin
    case_item_statement2a;
    case_item_statement2b;
    case_item_statement2c;
  end
  case_selection_item3 : case_item_statement3;
[default : case_item_statement5;] // optional
endcase
```
case Statement

- Looks like a completely specified case statement. But, what happens when `sel` takes on the value `2'b0X`? Is that possible? What would happen?

```verilog
module mux4_1 (
    input  sel,
    input  din_0, din_1, din_2, din_3;
    output logic d_out);
always_comb
    case (sel)
        2'b00 : d_out = din_0;
        2'b01 : d_out = din_1;
        2'b10 : d_out = din_2;
        2'b11 : d_out = din_3;
    endcase
endmodule
```
Case Statement

- Common variations for case statement
- Verilog has an implied `break` statement unlike C

```verilog
always_comb
  case (data_in)
    0 : a = 1; // simple match
    1 : begin // bracket multiple statements with begin/end
        a = 1;
        b = 1;
      end
    2,3,4 : c = 1; // match multiple values
    default: c = 0; // catch other possibilities
  endcase
```
case Statement

- Not accounting for all possibilities causes problems

```verilog
//incomplete case, 3x1 mux
module case2 (  
    input [7:0] a_in,  
    input [7:0] b_in,  
    input [7:0] 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
```

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)
case Statement

- SystemVerilog unique and priority clarify imcompletely specified case statements.
- 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.
**case Statement**

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

- Incomplete case with unique

```verilog
// incomplete case, 3x1 mux
module incomplete_case(
    input [7:0] a_in,
    input [7:0] b_in,
    input [7:0] c_in,
    input [1:0] sel,
    output logic [7:0] d_out);

always_comb
    unique case (sel)
        2'b00 : d_out = a_in;
        2'b01 : d_out = b_in;
        2'b10 : d_out = c_in;
    endcase
endmodule
```
In this case, **unique case** asserts that...

- **sel** should only take on values of 2'b00, 2'b01, or 2'b10. If any other values are encountered in simulation, an error will occur.
- The selection items, 2'b00, 2'b01, or 2'b10 are mutually exclusive, thus only one item should match **sel**.
- No priority logic is necessary.
case Statement

- Priority is an *assertion* which implies:
  - All legal values for `case_expression` are listed in `case_items`.
  - At least one `case_item` should match `case_expression`.
  - If more than one select expression matches the case expression, the first matching branch must be taken.

- Priority guides synthesis
  - All other possibilities for `case_items` are don’t cares and may be used to simplify logic.

- Priority usage
  - Explicitly says 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

- Interrupt priority encoder

```verilog
always_comb begin
  priority_case (1'b1)
    irq0 : irq = 4'b0001; // Highest Priority
    irq1 : irq = 4'b0010;
    irq2 : irq = 4'b0100;
    irq3 : irq = 4'b1000; // Lowest Priority
  endcase
end
```

- priority indicates:
  - One of the case_items must be true.
  - More than one interrupt request is legal.
  - If multiple requests occur, evaluate them in order.