1

Coding Styles for if Statements and case Statements

This chapter begins with the structure implied by simple if statements and case statements, and it proceeds to more complex examples that use a combination of if statements and case statements.

Basic if and case Statements

Examine the following basic if and case statements to understand the structure they imply and the differences, if any, between the two.

Example 1-1 and Example 1-2 show Verilog and VHDL versions of four sequential if statements. These designs result in a priority encoded structure as shown in Figure 1-1.

Example 1-1 Verilog Example of Priority Encoded if Statement

```
module mult_if(a, b, c, d, sel, z);
input a, b, c, d;
input [3:0] sel;
output z;
reg z;

always @(a or b or c or d or sel)
begin
    z = 0;
    if (sel[0]) z = a;
    if (sel[1]) z = b;
    if (sel[2]) z = c;
    if (sel[3]) z = d;
end
endmodule
```

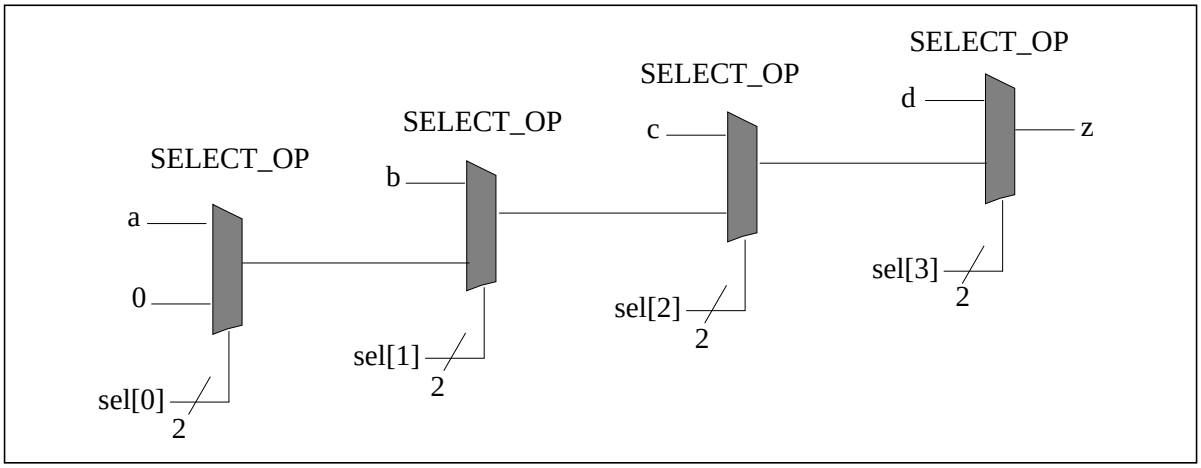
Example 1-2 is the equivalent VHDL example.

Example 1-2 VHDL Example of Priority Encoded if Statement

```
library IEEE;
use IEEE.std_logic_1164.all;

entity mult_if is
port (a, b, c, d: in std_logic;
    sel: in std_logic_vector(3 downto 0);
    z: out std_logic);
end mult_if;

architecture one of mult_if is
begin
    process (a, b, c, d, sel)
    begin
        z <= 0;
        if (sel(0) = '1') then
            z <= a;
        end if;
        if (sel(1) = '1') then
            z <= b;
        end if;
        if (sel(2) = '1') then
            z <= c;
        end if;
        if (sel(3) = '1') then
            z <= d;
        end if;
    end process;
end one;
```

Figure 1-1 shows the priority encoded, cascaded structure generated for Example 1-1 and Example 1-2.

Figure 1-1 Structure for Priority Encoded if Statement



Note:

The generic SELECT_OP components shown in Figure 1-1 are used by VHDL Compiler to implement conditional operations in the HDL (for example, if statements or case statements). SELECT_OP components behave like one-hot multiplexers; the control lines are mutually exclusive, and each SELECT_OP control input allows the data on the corresponding SELECT_OP data input to pass to the output of the SELECT_OP. Do not confuse these SELECT_OP components with one-hot multiplexers in a technology library. Design Compiler does not map to one-hot multiplexers in a technology library.

Sequential if statements allow you to structure HDL for late arriving signals. In Figure 1-1, the inputs to the first SELECT_OP in the chain (`sel[0]` and a or `0`) have the longest delay to the output z. The inputs to the last SELECT_OP in the chain (`sel[3]` and d) have the shortest delay to the output. This is discussed further in Chapter 2, “Coding if and case Statements for Late Arriving Signals.”

An `if...else if` construct in Verilog and an `if...elsif` in VHDL are implemented with single SELECT_OP components. Therefore, they do not result in the cascaded structure shown in Figure 1-1.

Single if statements result in a parallel structure. Example 1-3 and Example 1-4 show the Verilog and VHDL examples that use the single if statement coding style. Figure 1-2 shows the parallel structure inferred for these examples.

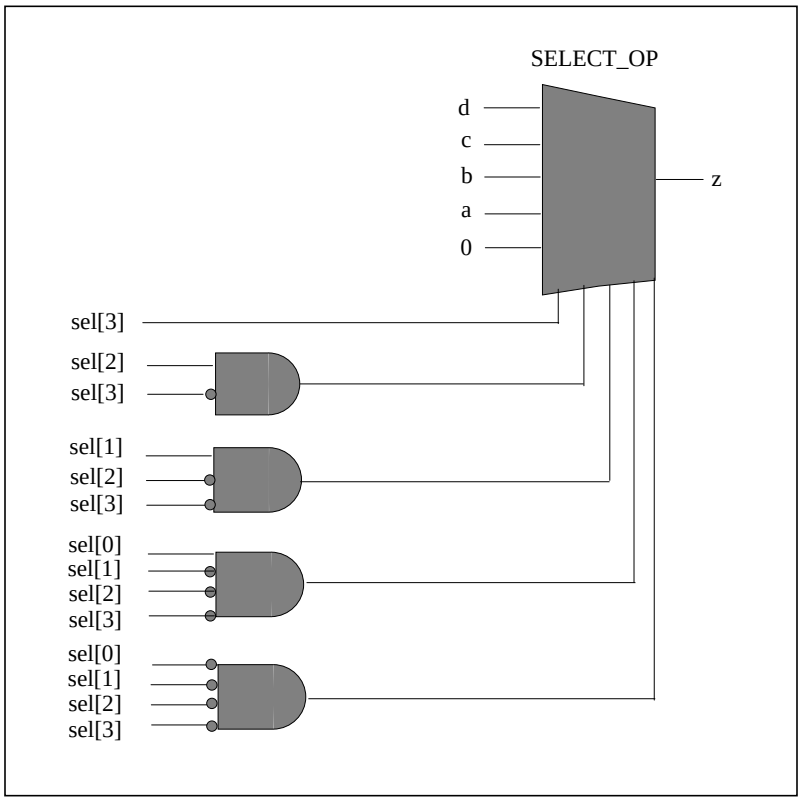
Example 1-3 Verilog Example for Single if Statement (Not Priority Encoded)

```
module single_if(a, b, c, d, sel, z);
input a, b, c, d;
input [3:0] sel;
output z;
reg z;

always @(a or b or c or d or sel)
begin
    z = 0;
    if (sel[3])
        z = d;
    else if (sel[2])
        z = c;
    else if (sel[1])
        z = b;
    else if(sel[0])
        z = a;
end
endmodule
```

Example 1-4 VHDL Example for Single if Statement (Not Priority Encoded)

```
library IEEE;
use IEEE.std_logic_1164.all;
entity single_if is
port (a, b, c, d: in std_logic;
    sel: in std_logic_vector(3 downto 0);
    z: out std_logic);
end single_if;

architecture one of single_if is
begin
    process(a, b, c, d, sel)
    begin
        z <= 0;
        if (sel(3) = '1') then
            z <= d;
        elsif (sel(2) = '1') then
            z <= c;
        elsif (sel(1) = '1') then
            z <= b;
        elsif (sel(0) = '1') then
            z <= a;
        end if;
    end process;
end one;
```

Figure 1-2 Structure Implied by Single if Statement



Now that you have seen some basic if statement examples, examine the structure implied by a simple case statement.

Simple case Statement

Example 1-5 and Example 1-6 show, respectively, Verilog and VHDL examples of a single case statement.

Example 1-5 Verilog for Single case Statement

```
module case1(a, b, c, d, sel, z);
input a, b, c, d;
input [3:0] sel;
output z;
reg z;
always @(a or b or c or d or sel)
begin
    casex (sel)
        4'b1xxx: z = d;
        4'bx1xx: z = c;
        4'bxx1x: z = b;
        4'bxxx1: z = a;
        default: z = 1'b0;
    endcase
end
endmodule
```

Notice the use of the casex statement in Example 1-5. The casex statement enables you to take advantage of don't care conditions, so you don't have to specify all possible 0/1 combinations of `sel`. VHDL, on the other hand, does not let you take advantage of don't care conditions. The VHDL language requires that VHDL Compiler treat comparisons with don't care conditions as false. Example 1-6 shows the equivalent VHDL for Example 1-5. In Example 1-6, all conditions have to be specified in the branches of the case statement.

Example 1-6 VHDL for Single case Statement

```
library IEEE;
use IEEE.std_logic_1164.all;
entity case1 is
port (a, b, c, d: in std_logic;
     sel: in std_logic_vector(3 downto 0);
     z: out std_logic);
end case1;

architecture one of case1 is
begin
     process(a,b,c,d,sel)
     begin
          case sel is
               when ”1000”|”1001”|”1010”|”1011”|
                    ”1100”|”1101”|”1110”|”1111” => z <= d;
               when ”0100”|”0101”|”0110”|”0111” => z <= c;
               when ”0010”|”0011” => z <= b;
               when ”0001” => z <= a;
               when others => z <= ’0’;
          end case;
     end process;
end one;
```

The `others` branch is required by the language in the case statement in Example 1-6, because the case statement does not cover all possible values. The “0000” case is missing, as are combinations of other `std_logic` values (the `std_logic` type is defined to have nine possible values). The `others` branch is required to cover combinations in addition to the 0 and 1 combinations.

The structure implied for the single case statement in Example 1-5 and Example 1-6 is identical to that shown in Figure 1-2 for a single if statement. A case statement gives the same results as a single if statement if the selector in the case statement branches is the same as the selector in the if statement branches.

The preceding VHDL examples cover multiple if statements, single if statements, and single case statements. These are summarized as follows:

Multiple if

```
statement4;
if (cond = cond1) then
     statement1;
end if;
if (cond = cond2) then
     statement2;
end if;
if (cond = cond3) then
     statement3;
end if;
```

A multiple if statement style has the longest path. Both the data and the control signals cascade through multiple SELECT_OP constructs.

Single if

```
if (cond = cond1) then
     statement1;
elsif (cond = cond2) then
     statement2;
elsif (cond = cond3) then
     statement3;
else
     statement4;
end if;
```

A single if statement is represented by a single SELECT_OP, but the control signals are priority encoded. The delay through the SELECT_OP is the same for all the data inputs.

Single case

```
case (cond) is
     when cond1 =>
          statement1;
     when cond2 =>
          statement2;
     when cond3 =>
          statement3;
     when others =>
          statement4;
end case;
```

A single case statement is also represented by a single SELECT_OP. There is no priority encoding for the control signals. The delay through the SELECT_OP is the same for all the data inputs. A single if and a single case statement give identical results if the selector is the same in every branch of the if statement.