Flow Control Statements

Introductory VHDL Methodology

Objectives

After completing this module, you will be able

- Write an ‘if/else’ conditional statement
- Write an ‘if/else if’ conditional statement
- Write a ‘case’ conditional statement
- Use the default ‘else’ or ‘others’ clause
- Detect and avoid conditions where latches may be inferred
- Write ‘for loop’ statements for repetitive operations

Language Structure

```
   architecture  RTL  of  ENTITY_1  is
  .   .   .
   begin
 concurrent statements ;
   . . .
process
   begin
       case (  ) is
      when…
             . . .
    end case ;
end process ;
   . . .
    . . .
process
  begin
     if (sel = “00”) then
       . . .
    else….
  end if ;
end process ;
 ...
end architecture RTL ;
```

An if/else, case or loop statement <u>must</u> be within a process

If / Else Statements

- The if / else statement allows operations to be performed based on certain conditions
- It has three basic forms

```
process
begin
if ( boolean expression ) then
sequential statements;
end if ;
```

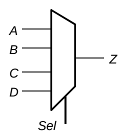
```
process
begin
if ( boolean expression ) then
sequential statements ;
else
sequential statements ;
end if ;
```

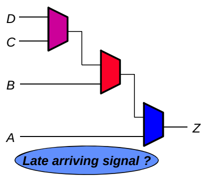
```
process
begin
if  ( boolean expression )  then
sequential statements ;
elsif  ( boolean expression ) then
sequential statements ;
elsif  ( boolean expression ) then
sequential statements ;
end if ;
```

If / Elsif Example and Rules

1. **The first condition found to be true will be executed**
2. **Conditions can overlap**
3. **The first condition of an if /elsif has priority**

```
process   ( A, B, C, D, Sel )
begin
If       ( Sel = “00” )  then
                Z <= A ;
elsif  ( Sel = “01” )  then
                Z <= B ;
elsif  ( Sel = “10” )  then
                Z <= C ;
elsif  ( Sel = “11” )  then
                Z <= D ;
end if;
end process ;
```





Case Statement

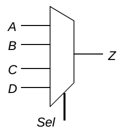
- **The case statement allows operations to be performed based on the value of a single expression, as indicated by the “selector expression”**

```
process   (…)
begin
case  ( selector expression )  is
when ..    . =>
              sequential statements ;
when ... =>
              sequential statements ;
when ... =>
              sequential statements ;
end case ;
. . .
end process ;
```

```
process (...)
begin
case  ( selector expression )  is
when ... =>
              sequential statements ;
. . .
when others =>
              sequential statements ;
end case ;
. . .
end process ;
```

Case Examples and Rules

1. All possible conditions must be specified
2. No conditions can overlap
3. All range specifications must be of a discrete type



```
process  (A, B, C, D, Sel )
begin
case  Sel  is
  when  "00"  =>   Z  <= A ;
  when  "01"  =>   Z  <= B ;
  when  "10"  =>   Z  <= C ;
  when  "11"  =>   Z  <= D ;
end  case ;
. . .
end process ;
```

Case statements are preferable for LUT architectures given that most synthesis tools will produce a mux, or similarly minimal logic level structure

Is this sufficient for std_logic?

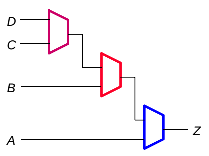
Overlapping Conditions

- **If condition overlap, then if/else if statement is necessary**

Relational Operator

```
process   ( A, B, C, D, Sel )
begin
If       ( Sel <=3)  then
           Z <= A ;
elsif  ( Sel <= 5 )  then
           Z <= B ;
elsif  ( Sel <= 7)  then
           Z <= C ;
elsif  ( Sel <= 9 )  then
           Z <= D ;
end if;
end process ;
```

'Assignment' Operator



Specifying Ranges

- **If a range is to specified as a condition of a case or if/else statement, it must be of a discrete type**

```
process   (...)
begin
if (x =  12 to 14) then                  sequential statements ;
. . .

case  ( selector expression )  is
when  0 to 7                =>     sequential statements ;
when  4.3 to 7.7            =>     sequential statements ;
when  "1000" | "1010"       =>     sequential statements ;
when  "1000"  to "1010"     =>     sequential statements ;
. . .
end process ;
```

Conditional Signal Assignment

- **A conditional signal assignment is effectively a concurrent form of the if / else statement**

```
architecture ...
 begin
   process  ( A,B, Sel )
     begin
       if  (Sel = '1')  then
         Z <= A;
        else
        Z <= B;
     end  if ;
   end  process ;
end  architecture ;
```

```
architecture ...
  begin
    Z <= A  when   Sel = '1'
           else  B ;
end architecture ;
```

- **Both statements produce the same results in simulation and synthesis. However, the latter does make the code less verbose**

Selected Signal Assignment

- **A selected signal assignment is effectively a concurrent form of the case statement**
- **The same rules exist as for the case statement:**
  (1) All conditions must be specified
  (2) No conditions may overlap

```
architecture ...
  begin
     process   ( A,B, Sel )
     begin
         case   (Sel)  is
         when   '1' => Z <= A ;
         when   '0' => Z <= B ;
      end  case ;
    end  process ;
end  architecture ;
```

```
architecture...
with  Sel  select
    Z <=  A   when   '1',
          B   when   '0' ;
end architecture ;
```

Loop Statements

- **Loop statements can be constructed for any repetitive operation. There are different forms, each having a different method of control. We shall examine the "for loop"**

```
process
variable COUNT : integer := 0;
 begin
  for  index in 0 to 15  loop
            COUNT := COUNT + 1 ;
end  loop ;
```

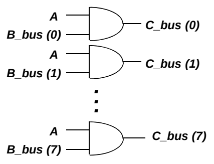
```
process  ( A, B_bus )
 begin
  for  I in 7 downto 0  loop
      C_bus (I) <= A  and  B_bus (I) ;
end  loop ;
```

*The loop variable **'index'** is not declared and is not visible outside the loop. It is treated as a constant*

With each loop iteration, it successively assumes the discrete values indicated in the range

Using Loop Statements

```
process  ( A, B_bus )
  begin
 for  I in 7 downto 0  loop
      C_bus (I) <= A  and  B_bus (I) ;
end  loop ;
```



- For synthesis, a loop is "unrolled" or logic resulting from each loop iteration is synthesized
- For execution (simulation), loop statements provide a very flexible tool for behavioral modeling
- Loop statements are also used extensively in subprograms (functions and procedures)

Review Questions

- In the notes section of your handout, write the corresponding case statement for the following if / elsif expression, and complete the sensitivity list

```
type User_Cond is  ( Rst, Wait, Jump, Fetch, Load) ;
Op_State : User_Cond ;
. . .
process  ( …)
begin
        If  Op_State = Rst  then
                Q_Out  <=  "0000" ;
        elsif  ( Op_State = Jump )  then
                Q_Out  <=  Data_Addr    ;
        elsif  ( Op_State = Load )  then
                 D_Reg  <=  Data_Stored ;
        else  D_Reg <= New_Data ;
   end if ;
end process ;
```

Bonus:

As presently written, both the case and if/else will produce unwanted additional logic, what is it and how can it be avoided?

Answers

```
type User_Cond is  ( Rst, Wait, Jump, Fetch, Load) ;
Op_State : User_Cond ;
. . .
process  (Op_state, Data_Addr, Data_Stored, New_Data)
    begin
        case ( Op_State) is
                 when    Rst    =>  Sig1  <=  "0000" ;
                 when    Jump =>  Sig1  <= Data_Addr ;
                 when    Load  =>  Sig2  <=  Data_Stored ;
                 when others  =>  Sig2  <= New_Data ;
       end case ;
end process ;
```

Sensitivity determined by which signals are read within the process

- Both Sig1 and Sig2 are not covered in each branch of the conditional statement; this produces a latch on both outputs. The default assignment to Sig2 does not prevent this because if an earlier branch condition is true, the later branch is never reached
- Solution: Either cover both signals in each branch, or create separate processes for each output