Variables

- Variables are used to store intermediate calculations within a process. They exist only within sequential VHDL and cannot be declared or used directly in an architecture.
- A variable is declaration is very much like a signal declaration: Ex; (variable a, b, c : bit;).
- Variable declarations appear in the declarative part of a process (before the *begin*) and it can be of any type.

Note: Signals are not updated during process execution. That is , they don't change value while the process is running. This means that signals cannot be used to store intermediate values of calculations during process execution.

Example:

```
entity full_adder is
        port (a, b, c : in bit;
                cout, sum : out bit);
end entity;
```

```
architecture rtl of full_add is
  begin
      process (a, b, c)
          variable sum1, sum2, c1, c2 : bit;
        begin
          sum1 := a xor b;
          c1 := a and b;
          sum2 := sum1 xor c;
          c2 := sum1 and c;
          sum <= sum2;
          cout <= c1 or c2;
       end process;
  end
```

If_Statements

The *if* statement is the sequential equivalent of the conditional signal assignment. It allows the execution of a process to take one of a number of branches depending on a condition.

Example:

```
process (a, b)
begin
   if a = b then
      result <= 0;
```

```
    elsif a < b then
        result <= –1;
    else
        result <= –1;
    end if;
end process;
```

- The hardware interpretation of an *if* statement is a multiplexer.
- The conditions in the successive branches of an *if* statement are evaluated independently. However, the structure of the *if* statement ensures that the earlier conditions are tested first.

Note:

- With multiple-branch *if* statements, each condition can be, and will normally be, dependent on different signals and variables. If every branch is dependent on the same signal, then you probably need a *case* statement.
- There are two different situations where a signal does not receive a value: where there is a missing *else*, and where the signal is not assigned to in some branches. In both cases the interpretation is the same; **the previous value is preserved**.
- This opens up the question: what is the previous

value? if the signal has an earlier assignment, then the value comes from that assignment. If not, then the previous value comes from the previous execution of the process, **leading to feedback in the circuit** !!

Example:

- In this case, the *if* statement is **incomplete** because it is lacking an *else* part. Thus the signal gets a value in the *if* statement if the condition *enable*=*'1'* is satisfied, but remains unassigned if the condition is false. In this case the previous value comes from the unconditional assignment before the *if* statement.

This is equivalent to:

```
process (a, b, enable)
 begin
   if enable = '1' then
      z<= b;
   else
      z <= a;
   end if;
  end process;
```

- If there is no previous assignment, then there is feedback from the output of the circuit to an input. This because the value of the signal from the previous execution of the process

is preserved and becomes the value in this execution of the process.

Example:

```
process
 begin
   wait until ck='1';
   if en = '1' then
      q<= d;
   end if;
  end process;
```

- The signal *q* is updated with the new value of *d* if the condition is true, but is not updated at

all if the condition is false. In this case, the previous value of signal *q* is preserved.

The equivalent interpretation by VHDL will be

```
process
 begin
   wait until ck='1';
   if en = '1' then
      q<= d;
   else
      q<=q;
   end if;
  end process;
```

Note:

- One of the most pitfalls in writing VHDL for logic synthesis is the accident introduction of feedback in the circuit owing to an incomplete *if* statement.
- There are two ways to overcome that in practice: wither be careful to make sure that every signal assigned to in an *if* statement is assigned to in every branch and that there is always an *else* part; or to initialize signals with unconditional assignment before the *if* statement.

Example:

```
process
begin
wait on a, b, c;
if c='1' then
  z<=a;
else
  y<=b;
end if;
end process;
```

- In this example, although the if statement looks complete, different signal are being assigned a value in each branch of the if statement. Thus

both signals *z* and *y* will have asynchronous feedback.

Example:(redundant test)

```
process
begin
wait on a, b, c;
if c='1' then
  z<=a;
elseif c='0' then
  y<=b;
end if;
end process;
```

- In this case, although the if statement looks complete, each of the conditions in an if statement is synthesized in isolation from the other conditions. The synthesizer will therefore, not detect that this second condition is redundant. Thus the if statement is synthesized as a three-way multiplexer, the third input being the missing *else* condition which is the feedback of the previous value.

Case_Statement

Unlike the if statement, the condition of a case statement does not need to be boolean.

In general, the process that controls the combinational part of the sequencer is best described using a case statement:

Example:

```
process(light)
begin
  case (light) is
     when red =>
       next_light <= green;
     when amber =>
```

```
      next_light <= red;
    when green =>
      next_light <= amber;
  end case;
end process;
```