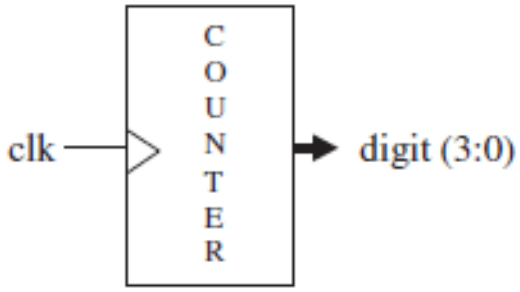
Example One-digit Counter #1



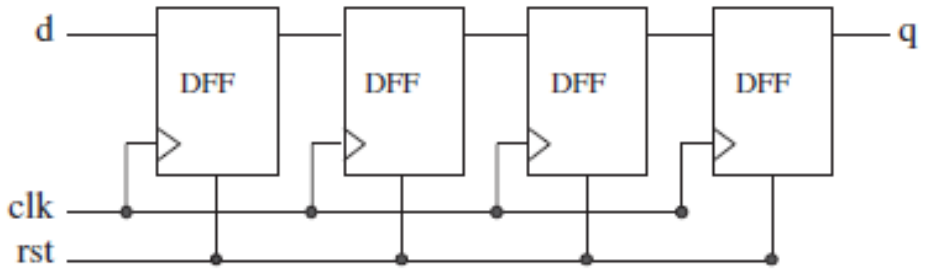
The code below implements a progressive 1-digit decimal counter (0 -> 9 -> 0).

```
1 ------------------------------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ------------------------------------------
5 ENTITY counter IS
6       PORT (clk : IN STD_LOGIC;
7             digit : OUT INTEGER RANGE 0 TO 9);
8 END counter;
9 ------------------------------------------
10 ARCHITECTURE counter OF counter IS
11    BEGIN
12          count: PROCESS(clk)
13          VARIABLE temp : INTEGER RANGE 0 TO 10;
14                BEGIN
15                      IF (clk'EVENT AND clk='1') THEN
16                      temp := temp + 1;
17                            IF (temp=10) THEN temp := 0;
18                            END IF;
19                      END IF;
20          digit <= temp;
21          END PROCESS count;
22 END counter;
23 ------------------------------------------
```

The initial value of temp in the physical circuit can be any 4-bit value. If such value is below 10 (see line 17), the circuit will count correctly from there.

Example: Shift Register

In the Figure below we have a shift register where the output bit (q) must be four positive clock edges behind the input bit (d). It also contains an asynchronous reset, which must force all flip-flop outputs to '0' when asserted. In this example, the IF statement is employed.




```
1 -------------------------------------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 -------------------------------------------------
5 ENTITY shiftreg IS
6       GENERIC (n: INTEGER := 4); -- # of stages
7       PORT (d, clk, rst: IN STD_LOGIC;
8                q: OUT STD_LOGIC);
9 END shiftreg;

10 -------------------------------------------------
11 ARCHITECTURE behavior OF shiftreg IS
12            SIGNAL internal: STD_LOGIC_VECTOR (n-1 DOWNTO 0);
13    BEGIN
14            PROCESS (clk, rst)
15              BEGIN
16                    IF (rst='1') THEN
17                          internal <= (OTHERS => '0');
18                    ELSIF (clk'EVENT AND clk='1') THEN
19                          internal <= d & internal(internal'LEFT DOWNTO 1);
20                    END IF;
21              END PROCESS;
22    q <= internal(0);
23 END behavior;
24 -------------------------------------------------
```

Solution-2:

```
1 ------------------------------------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ------------------------------------------------
5 ENTITY shift_register IS
6       GENERIC (N: INTEGER := 4); --number of stages
7       PORT (din, clk, rst: IN STD_LOGIC;
8             dout: OUT STD_LOGIC);
9 END ENTITY;
10 ------------------------------------------------
11 ARCHITECTURE shift_register OF shift_register IS
12    BEGIN
13          PROCESS (clk, rst)
14                VARIABLE internal: STD_LOGIC_VECTOR(0 TO N-1);
15          BEGIN
16                IF (rst='1') THEN
17                      internal := (OTHERS => '0');
18                ELSIF (clk'EVENT AND clk='1') THEN
19                      internal := din & internal (0 TO N-2);
20                END IF;
21    dout <= internal (N-1);
22          END PROCESS;
23 END ARCHITECTURE;
24 ------------------------------------------------
```

The WAIT Statement

WAIT is another sequential statement. It is available in three forms, of which two are for synthesis and one is for simulation. When WAIT is employed, the PROCESS cannot have a sensitivity list. Simplified syntaxes for all three forms follow.

[label:] WAIT UNTIL condition;

[label:] WAIT ON sensitivity_list;

[label:] WAIT FOR time_expression;

WAIT UNTIL: This statement causes the process or subprogram to hold until the expressed condition is fulfilled.

---DFF process with IF:----------

```
PROCESS (clk)
BEGIN
        IF (clk'EVENT AND clk='1') THEN
                IF (clr='1') THEN
                        q <= '0';
                ELSE
                        q <= d;
                END IF;
        END IF;
END PROCESS;
```

---DFF process with WAIT UNTIL:-------

```
PROCESS
BEGIN
        WAIT UNTIL (clk'EVENT AND clk='1');
        IF (clr='1') THEN
                q <= '0';
        ELSE
                q <= d;
        END IF;
END PROCESS;
```

WAIT ON: This statement causes the process or subprogram to hold until any listed signal changes. In the example below, WAIT ON monitors the clock. Since a single WAIT ON statement at the beginning or at the end of a process is equivalent to using a process with the same signals listed in the sensitivity list, the two processes below for a DFF with synchronous clear are equivalent.

```
---DFF process with IF:-----------
PROCESS (clk)
BEGIN
        IF (clk'EVENT AND clk='1') THEN
                IF (clr='1') THEN
                        q <= '0';
                ELSE
                        q <= d;
                END IF;
        END IF;
END PROCESS;
---------------------------------

---DFF process with WAIT ON:------
PROCESS
BEGIN
        IF (clk'EVENT AND clk='1') THEN
                IF (clr='1') THEN
                        q <= '0';
                ELSE
                        q <= d;
                END IF;
        END IF;
WAIT ON clk;
END PROCESS;
---------------------------------
```

WAIT FOR: This statement is for simulations. The declaration below creates a clock waveform with period 80 ns.

```
WAIT FOR 40ns;
clk <= NOT clk;
```

Example: DFF with Asynchronous Reset.
The code below implements the DFF. However, here WAIT ON is used instead of IF only.

```
1 ------------------------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ------------------------------------
5 ENTITY dff IS
6       PORT (d, clk, rst: IN STD_LOGIC;
7               q: OUT STD_LOGIC);
8 END dff;
9 ------------------------------------
10 ARCHITECTURE dff OF dff IS
11     BEGIN
12        PROCESS
13            BEGIN
14                  WAIT ON rst, clk;
15            IF (rst='1') THEN
16                  q <= '0';
17            ELSIF (clk'EVENT AND clk='1') THEN
18                  q <= d;
19            END IF;
20       END PROCESS;
21 END dff;
22 ------------------------------------
```

Example: One-digit Counter
The code below implements the same progressive 1-digit decimal counter. However, WAIT UNTIL was used instead of IF only.

```
1 ------------------------------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ------------------------------------------
5 ENTITY counter IS
6       PORT (clk : IN STD_LOGIC;
7               digit : OUT INTEGER RANGE 0 TO 9);
8 END counter;
9 ------------------------------------------
```

```
10 ARCHITECTURE counter OF counter IS
11     BEGIN
12     PROCESS -- no sensitivity list
13             VARIABLE temp : INTEGER RANGE 0 TO 10;
14     BEGIN
15             WAIT UNTIL (clk'EVENT AND clk='1');
16             temp := temp + 1;
17             IF (temp=10) THEN temp := 0;
18             END IF;
19             digit <= temp;
20     END PROCESS;
21 END counter;
22 ---------------------------------------------
```

CASE

CASE is another statement intended exclusively for sequential code (along with IF, LOOP, and WAIT). Its syntax is shown below.

```
CASE identifier IS
        WHEN value => assignments;
        WHEN value => assignments;
        ...
END CASE;
```

Example

```
CASE control IS
        WHEN "000" => x<=a; y<=b;
        WHEN "000" | "111" => x<=b; y<= '0';
        WHEN OTHERS => x<='0'; y<='1';
END CASE;
```

Example:

```
CASE control IS
        WHEN "00" => x<=a; y<=b;
        WHEN "01" => x<=b; y<=c;
        WHEN OTHERS => x<="0000"; y<="ZZZZ";
END CASE;
```

Another important keyword is NULL which should be used when no action is to take place.
For example, WHEN OTHERS =>NULL;

Like SELECT, CASE too allows the use of multiple values, which can be grouped with "|" (means "or") or "TO" (for range), as shown.

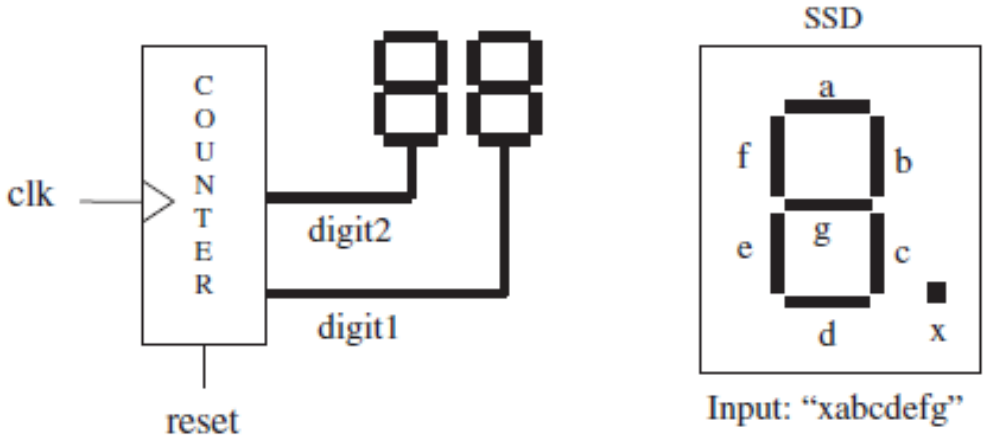
WHEN value1 | value2 |... --value1 or value2 or ...
WHEN value1 TO value2 --range (for enumerated types only)

Example: DFF with Asynchronous Reset. The code below implements the DFF. However, here CASE was used instead of IF only. Notice that a few unnecessary declarations were intentionally included in the code to illustrate their usage.

```
1 ---------------------------------------------
2 LIBRARY ieee;                                  -- Unnecessary declaration,
3                                                -- because
4 USE ieee.std_logic_1164.all;                   -- BIT was used instead of
5                                                -- STD_LOGIC
6 ---------------------------------------------
7 ENTITY dff IS
8       PORT (d, clk, rst: IN BIT;
9                 q: OUT BIT);
10 END dff;
11 ---------------------------------------------
12 ARCHITECTURE dff3 OF dff IS
13     BEGIN
14             PROCESS (clk, rst)
15             BEGIN
16                     CASE rst IS
17                     WHEN '1' => q<='0';
18                     WHEN '0' =>
19                     IF (clk'EVENT AND clk='1') THEN
20                             q <= d;
21                     END IF;
22                     WHEN OTHERS => NULL; -- Unnecessary, rst is of type
23                                                -- BIT
24                     END CASE;
25             END PROCESS;
26 END dff3;
27 ---------------------------------------------
```

Example: Two-digit Counter with SSD Output

The code below implements a progressive 2-digit decimal counter (0 -> 99 -> 0), with external asynchronous reset plus binary-coded decimal (BCD) to seven-segment display (SSD) conversion.




```
1 ------------------------------------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ------------------------------------------------
5 ENTITY counter IS
6       PORT (clk, reset : IN STD_LOGIC;
7               digit1, digit2 : OUT STD_LOGIC_VECTOR (6 DOWNTO 0));
8 END counter;
9 ------------------------------------------------
10 ARCHITECTURE counter OF counter IS
11 BEGIN
12      PROCESS(clk, reset)
13              VARIABLE temp1: INTEGER RANGE 0 TO 10;
14              VARIABLE temp2: INTEGER RANGE 0 TO 10;
15      BEGIN
16 ---- counter: ----------------------
17      IF (reset='1') THEN
18              temp1 := 0;
19              temp2 := 0;
20      ELSIF (clk'EVENT AND clk='1') THEN
21              temp1 := temp1 + 1;
22              IF (temp1=10) THEN
23                      temp1 := 0;
```

```
24                      temp2 := temp2 + 1;

25                      IF (temp2=10) THEN
26                              temp2 := 0;
27                      END IF;
28              END IF;
29      END IF;

30 ---- BCD to SSD conversion: --------
31 CASE temp1 IS
32              WHEN 0 => digit1 <= "1111110"; --7E
33              WHEN 1 => digit1 <= "0110000"; --30
34              WHEN 2 => digit1 <= "1101101"; --6D
35              WHEN 3 => digit1 <= "1111001"; --79
36              WHEN 4 => digit1 <= "0110011"; --33
37              WHEN 5 => digit1 <= "1011011"; --5B
38              WHEN 6 => digit1 <= "1011111"; --5F
39              WHEN 7 => digit1 <= "1110000"; --70
40              WHEN 8 => digit1 <= "1111111"; --7F
41              WHEN 9 => digit1 <= "1111011"; --7B
42              WHEN OTHERS => NULL;
43 END CASE;

44 CASE temp2 IS
45              WHEN 0 => digit2 <= "1111110"; --7E
46              WHEN 1 => digit2 <= "0110000"; --30
47              WHEN 2 => digit2 <= "1101101"; --6D
48              WHEN 3 => digit2 <= "1111001"; --79
49              WHEN 4 => digit2 <= "0110011"; --33
50              WHEN 5 => digit2 <= "1011011"; --5B
51              WHEN 6 => digit2 <= "1011111"; --5F
52              WHEN 7 => digit2 <= "1110000"; --70
53              WHEN 8 => digit2 <= "1111111"; --7F
54              WHEN 9 => digit2 <= "1111011"; --7B
55              WHEN OTHERS => NULL;
56 END CASE;

57  END PROCESS;
58 END counter;
59 ------------------------------------------------
```

The LOOP Statement

As the name says, LOOP is used when a piece of code must be instantiated several times. It is the counterpart of the concurrent statement GENERATE. Like IF, WAIT, and CASE, LOOP also can only be used in sequential code (PROCESS and subprograms).

There are five cases involving the LOOP statements: unconditional, with FOR, with WHILE, with EXIT, and with NEXT.

Unconditional LOOP:

```
    [label:] LOOP
    sequential_statements
END LOOP [label];
```

LOOP with FOR:

```
    [label:] FOR identifier IN range LOOP
    sequential_statements
END LOOP [label];
```

LOOP with WHILE:

```
    [label:] WHILE condition LOOP
    sequential_statements
END LOOP [label];
```

LOOP with EXIT:

```
    [loop_label:] [FOR identifier IN range] LOOP
    ...
    [exit_label:] EXIT [loop_label] [WHEN condition];
    ...
END LOOP [loop_label];
```

LOOP with NEXT:

```
    [loop_label:] [FOR identifier IN range] LOOP
    ...
    [next_label:] NEXT [loop_label] [WHEN condition];
    ...
END LOOP [loop_label];
```

Example of unconditional LOOP:

```
LOOP
        WAIT UNTIL clk='1';
        count := count + 1;
END LOOP;
```

Example of LOOP with FOR:

```
FOR i IN 0 TO 5 LOOP
        x(i) <= a(i) AND b(5-i);
        y(0, i) <= c(i);
END LOOP;
```

Example of LOOP with WHILE: The loop below will keep repeating while i < 10.

```
WHILE (i<10) LOOP
        WAIT UNTIL clk'EVENT AND clk='1';
        ...
END LOOP;
```

Example of LOOP with EXIT: The loop will be terminated if a value di¤erent from '0' is found in data.

```
FOR i IN data'RANGE LOOP
        CASE data(i) IS
                WHEN '0' => count:=count+1;
                WHEN OTHERS => EXIT;
        END CASE;
END LOOP;
```

Example of LOOP with NEXT: NEXT will cause LOOP to skip one iteration if i=skip occurs.

```
FOR i IN 0 TO 15 LOOP
        NEXT WHEN i=skip;
        ...
END LOOP;
```