Design with Hardware Description Languages (HDL)

(DHDL-94952)

Amin Mehranzadeh, Ph.D.

CE Department of
Islamic Azad University of Dezful

mehranzadeh@iaud.ac.ir
Mehran.students@gmail.com

Sequential Code

Sequential Code:

- As mentioned before, VHDL code is inherently concurrent. PROCESSES, FUNCTIONS, and PROCEDURES are the only sections of code that are executed sequentially. However, as a whole, any of these blocks is still concurrent with any other statements placed outside it.
- Sequential code is also called behavioral code.

Sequential Code:

- The statements discussed in this section are all sequential, that is, allowed only inside PROCESSES, FUNCTIONS, or PROCEDURES. They are: IF, WAIT, CASE, and LOOP.
- VARIABLES are also restricted to be used in sequential code only (that is, inside a PROCESS, FUNCTION, or PROCEDURE). Thus, contrary to a SIGNAL, a VARIABLE can never be global, so its value can not be passed out directly.

PROCESS:

- A PROCESS is a sequential section of VHDL code.
- It is characterized by the presence of IF, WAIT, CASE, or LOOP, and by a sensitivity list (except when WAIT is used).
- A PROCESS must be installed in the main code, and is executed every time a signal in the sensitivity list changes (or the condition related to WAIT is fulfilled).

PROCESS:

- Its syntax is shown below:

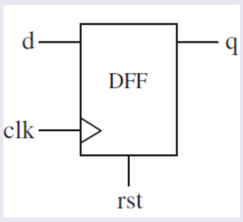
```
[label:] PROCESS (sensitivity list)
  [VARIABLE name type [range] [:= initial_value;]]
BEGIN
  (sequential code)
END PROCESS [label];
```

Notes:

- VARIABLES are optional. If used, they must be declared in the declarative part of the PROCESS (before the word BEGIN, as indicated in the syntax above).
- The initial value is not synthesizable, being only taken into consideration in simulations.
- The use of a label is also optional. Its purpose is to improve code readability. The label can be any word, except VHDL reserved words.

Example 1: DFF with Asynchronous Reset #1

- A D-type flip-flop (DFF, figure 6.1) is the most basic building block in sequential logic circuits. In it, the output must copy the input at either the positive or negative transition of the clock signal (rising or falling edge).



Example 1: DFF with Asynchronous Reset #1

```
--------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
--------------------------------------
ENTITY dff IS
   PORT (d, clk, rst: IN STD_LOGIC;
         q: OUT STD_LOGIC);
END dff;
--------------------------------------
ARCHITECTURE behavior OF dff IS
BEGIN
   PROCESS (clk, rst)
   BEGIN
      IF (rst='1') THEN
         q <= '0';
      ELSIF (clk'EVENT AND clk='1') THEN
         q <= d;
   END IF;
   END PROCESS;
END behavior;
--------------------------------------
```

Signals and Variables:

- VHDL has two ways of passing non-static values around: by means of a SIGNAL or by means of a VARIABLE.
- A SIGNAL can be declared in a PACKAGE, ENTITY or ARCHITECTURE (in its declarative part), while a VARIABLE can only be declared inside a piece of sequential code (in a PROCESS, for example).
- Therefore, while the value of the former can be global, the latter is always local.

Signals and Variables:

- The value of a VARIABLE can never be passed out of the PROCESS directly; if necessary, then it must be assigned to a SIGNAL. On the other hand, the update of a VARIABLE is immediate, that is, we can promptly count on its new value in the next line of code.
- That is not the case with a SIGNAL (when used in a PROCESS), for its new value is generally only guaranteed to be available after the conclusion of the present run of the PROCESS.

IF:

- The syntax of IF is shown below:

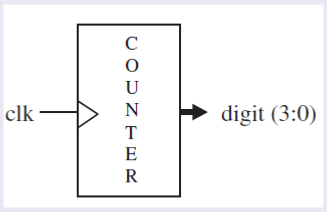
```
IF conditions THEN assignments;
ELSIF conditions THEN assignments;
...
ELSE assignments;
END IF;
```

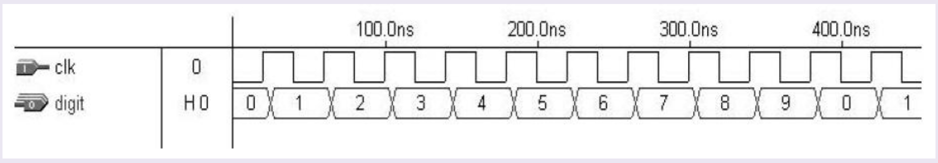
Example:

```
IF (x<y) THEN temp:="11111111";
ELSIF (x=y AND w='0') THEN temp:="11110000";
ELSE temp:=(OTHERS =>'0');
```

Example 2: One-digit Counter #1

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



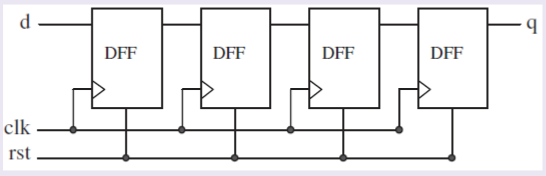


Example 2: One-digit Counter #1

```
---------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------------
ENTITY counter IS
   PORT (clk : IN STD_LOGIC;
         digit : OUT INTEGER RANGE 0 TO 9);
END counter;
---------------------------------------------
ARCHITECTURE counter OF counter IS
BEGIN
   count: PROCESS(clk)
      VARIABLE temp : INTEGER RANGE 0 TO 10;
   BEGIN
      IF (clk'EVENT AND clk='1') THEN
         temp := temp + 1;
         IF (temp=10) THEN temp := 0;
         END IF;
      END IF;
      digit <= temp;
   END PROCESS count;
END counter;
---------------------------------------------
```

Example 3: Shift Register

- Figure below shows a 4-bit shift register. 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.



Example 3: Shift Register

```
----------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
----------------------------------------------
ENTITY shiftreg IS
   GENERIC (n: INTEGER := 4);    -- # of stages
   PORT (d, clk, rst: IN STD_LOGIC;
         q: OUT STD_LOGIC);
END shiftreg;
--------------------------------------------------
ARCHITECTURE behavior OF shiftreg IS
   SIGNAL internal: STD_LOGIC_VECTOR (n-1 DOWNTO 0);
BEGIN
   PROCESS (clk, rst)
   BEGIN
      IF (rst='1') THEN
         internal <= (OTHERS => '0');
      ELSIF (clk'EVENT AND clk='1') THEN
         internal <= d & internal(internal'LEFT DOWNTO 1);
      END IF;
   END PROCESS;
   q <= internal(0);
END behavior;
--------------------------------------------------
```

WAIT:

- The operation of WAIT is sometimes similar to that of IF. Moreover, contrary to when IF, CASE, or LOOP are used, the PROCESS cannot have a sensitivity list when WAIT is employed. Its syntax (there are three forms of WAIT) is shown below.

```
WAIT UNTIL signal_condition;
```

```
WAIT ON signal1 [, signal2, ... ];
```

```
WAIT FOR time;
```

WAIT UNTIL:

- The WAIT UNTIL statement accepts only one signal, thus being more appropriate for synchronous code than asynchronous. Since the PROCESS has no sensitivity list in this case, WAIT UNTIL must be the first statement in the PROCESS. The PROCESS will be executed every time the condition is met.

Example: 8-bit register with synchronous reset.

```
PROCESS          -- no sensitivity list
BEGIN
   WAIT UNTIL (clk'EVENT AND clk='1');
   IF (rst='1') THEN
      output <= "00000000";
   ELSIF (clk'EVENT AND clk='1') THEN
      output <= input;
   END IF;
END PROCESS;
```

WAIT ON:

- WAIT ON, on the other hand, accepts multiple signals.
- The PROCESS is put on hold until any of the signals listed changes.
- In the example below, the PROCESS will continue execution whenever a change in rst or clk occurs.

```
PROCESS
BEGIN
   WAIT ON clk, rst;
   IF (rst='1') THEN
      output <= "00000000";
   ELSIF (clk'EVENT AND clk='1') THEN
      output <= input;
   END IF;
END PROCESS;
```

WAIT FOR:

- Finally, WAIT FOR is intended for simulation only (waveform generation for testbenches).
- Example: WAIT FOR 5ns;

Example 4: DFF with Asynchronous Reset #2

Example 4: DFF with Asynchronous Reset #2

- The code below implements the same DFF of example1.
- However, here WAIT ON is used instead of IF only.

```
---------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------
ENTITY dff IS
   PORT (d, clk, rst: IN STD_LOGIC;
         q: OUT STD_LOGIC);
END dff;
---------------------------------------
ARCHITECTURE dff OF dff IS
BEGIN
   PROCESS
   BEGIN
      WAIT ON rst, clk;
      IF (rst='1') THEN
         q <= '0';
      ELSIF (clk'EVENT AND clk='1') THEN
         q <= d;
      END IF;
   END PROCESS;
END dff;
---------------------------------------
```

Example 5: One-digit Counter #2

- The code below implements the same progressive 1-digit decimal counter of example2.
- However, WAIT UNTIL was used instead of IF only.

```
---------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------------
ENTITY counter IS
   PORT (clk : IN STD_LOGIC;
         digit : OUT INTEGER RANGE 0 TO 9);
END counter;
---------------------------------------------
ARCHITECTURE counter OF counter IS
BEGIN
   PROCESS         -- no sensitivity list
      VARIABLE temp : INTEGER RANGE 0 TO 10;
   BEGIN
      WAIT UNTIL (clk'EVENT AND clk='1');
      temp := temp + 1;
      IF (temp=10) THEN temp := 0;
      END IF;
      digit <= temp;
   END PROCESS;
END counter;
---------------------------------------------
```

Amin Mehranzadeh, Ph.D.

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

Amin Mehranzadeh, Ph.D.

CASE:

- The CASE statement (sequential) is very similar to WHEN (combinational).
- Here too all permutations must be tested, so the keyword OTHERS is often helpful.
- Another important keyword is NULL (the counterpart of UNAFFECTED), which should be used when no action is to take place. For example, WHEN OTHERS =>NULL;.
- However, CASE allows multiple assignments for each test condition, while WHEN allows only one.

CASE:

- Like in the case of WHEN, here too “WHEN value” can take up three forms:

```
WHEN value                  -- single value
WHEN value1 to value2       -- range, for enumerated data types
                            -- only
WHEN value1 | value2 |...   -- value1 or value2 or ...
```

Example 6: DFF with Asynchronous Reset #3

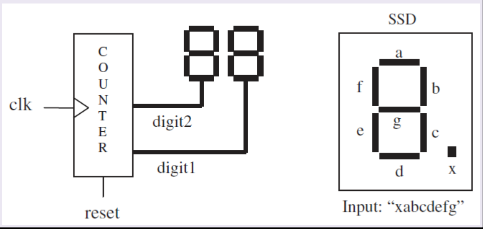
- The code below implements the same DFF of example1.
- However, here CASE was used instead of IF only.

```
---------------------------------------------
LIBRARY ieee;                    -- Unnecessary declaration,
                                 -- because
USE ieee.std_logic_1164.all;     -- BIT was used instead of
                                 -- STD_LOGIC
---------------------------------------------
ENTITY dff IS
   PORT (d, clk, rst: IN BIT;
         q: OUT BIT);
END dff;
---------------------------------------------
ARCHITECTURE dff3 OF dff IS
BEGIN
   PROCESS (clk, rst)
   BEGIN
      CASE rst IS
         WHEN '1' => q<='0';
         WHEN '0' =>
            IF (clk'EVENT AND clk='1') THEN
               q <= d;
            END IF;
         WHEN OTHERS => NULL;     -- Unnecessary, rst is of type
                                  -- BIT
      END CASE;
   END PROCESS;
END dff3;
---------------------------------------------
```

Amin Mehranzadeh, Ph.D.

Example 7: 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. Notice that we have chosen the following connection between the
- circuit and the SSD: <u>xabcdefg</u> (that is, the MSB feeds the decimal point, while the LSB feeds segment g).



Amin Mehranzadeh, Ph.D.

Example 7: Two-digit Counter with SSD Output

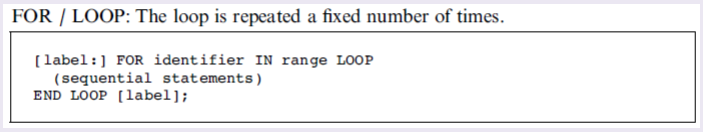
```
---------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------------------
ENTITY counter IS
   PORT (clk, reset : IN STD_LOGIC;
         digit1, digit2 : OUT STD_LOGIC_VECTOR (6 DOWNTO 0));
END counter;
---------------------------------------------------
ARCHITECTURE counter OF counter IS
BEGIN
   PROCESS(clk, reset)
      VARIABLE temp1: INTEGER RANGE 0 TO 10;
      VARIABLE temp2: INTEGER RANGE 0 TO 10;
   BEGIN
   ---- counter: ----------------------
      IF (reset='1') THEN
         temp1 := 0;
         temp2 := 0;
      ELSIF (clk'EVENT AND clk='1') THEN
         temp1 := temp1 + 1;
         IF (temp1=10) THEN
            temp1 := 0;
            temp2 := temp2 + 1;
            IF (temp2=10) THEN
               temp2 := 0;
```

Example 7: Two-digit Counter with SSD Output

```
            END IF;
         END IF;
      END IF;
      ---- BCD to SSD conversion: --------
      CASE temp1 IS
         WHEN 0 => digit1 <= "1111110";    --7E
         WHEN 1 => digit1 <= "0110000";    --30
         WHEN 2 => digit1 <= "1101101";    --6D
         WHEN 3 => digit1 <= "1111001";    --79
         WHEN 4 => digit1 <= "0110011";    --33
         WHEN 5 => digit1 <= "1011011";    --5B
         WHEN 6 => digit1 <= "1011111";    --5F
         WHEN 7 => digit1 <= "1110000";    --70
         WHEN 8 => digit1 <= "1111111";    --7F
         WHEN 9 => digit1 <= "1111011";    --7B
         WHEN OTHERS => NULL;
      END CASE;
      CASE temp2 IS
         WHEN 0 => digit2 <= "1111110";    --7E
         WHEN 1 => digit2 <= "0110000";    --30
         WHEN 2 => digit2 <= "1101101";    --6D
         WHEN 3 => digit2 <= "1111001";    --79
         WHEN 4 => digit2 <= "0110011";    --33
         WHEN 5 => digit2 <= "1011011";    --5B
         WHEN 6 => digit2 <= "1011111";    --5F
         WHEN 7 => digit2 <= "1110000";    --70
         WHEN 8 => digit2 <= "1111111";    --7F
         WHEN 9 => digit2 <= "1111011";    --7B
         WHEN OTHERS => NULL;
      END CASE;
   END PROCESS;
END counter;
---------------------------------------------------
```

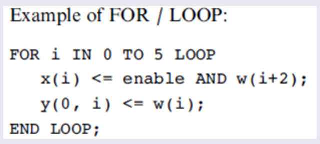
LOOP:

- As the name says, LOOP is useful when a piece of code must be instantiated several times. There are several ways of using LOOP, as shown in the syntaxes below.


FOR / LOOP: The loop is repeated a fixed number of times.

```
[label:] FOR identifier IN range LOOP
  (sequential statements)
END LOOP [label];
```

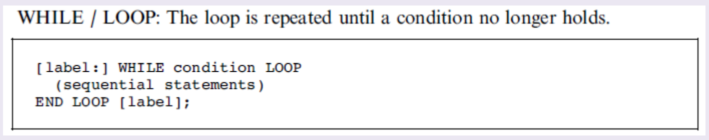
Note:
One important remark regarding FOR / LOOP (similar to that made for GENERATE) is that both limits of the range must be static.


Example of FOR / LOOP:

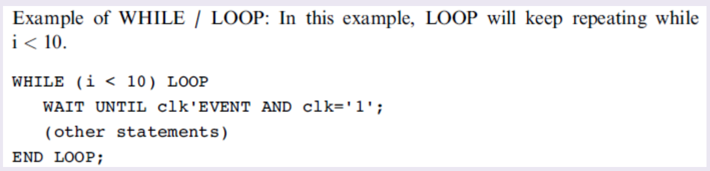
```
FOR i IN 0 TO 5 LOOP
   x(i) <= enable AND w(i+2);
   y(0, i) <= w(i);
END LOOP;
```

Amin Mehranzadeh, Ph.D.   CE Department of Islamic Azad University of Dezful

LOOP:


WHILE / LOOP: The loop is repeated until a condition no longer holds.

```
[label:] WHILE condition LOOP
  (sequential statements)
END LOOP [label];
```


Example of WHILE / LOOP: In this example, LOOP will keep repeating while $i < 10$.

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

Amin Mehranzadeh, Ph.D.   CE Department of Islamic Azad University of Dezful

LOOP:

EXIT: Used for ending the loop.

```
[label:] EXIT [label] [WHEN condition];
```

Example with EXIT: In this case, the loop will end as soon as a value different from '0' is found in the data vector.

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

Amin Mehranzadeh, Ph.D.   CE Department of Islamic Azad University of Dezful

LOOP:

NEXT: Used for skipping loop steps.

```
[label:] NEXT [loop_label] [WHEN condition];
```

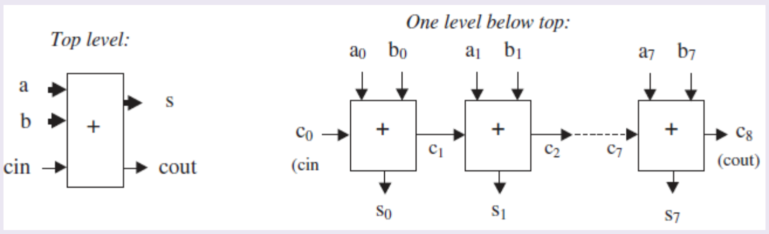
Example with NEXT: In the example below, NEXT causes LOOP to skip one iteration when i = skip.

```
FOR i IN 0 TO 15 LOOP
   NEXT WHEN i=skip;     -- jumps to next iteration
      (...)
END LOOP;
```

Amin Mehranzadeh, Ph.D.   CE Department of Islamic Azad University of Dezful

Example 8: Carry Ripple Adder

Figure below shows an 8-bit unsigned carry ripple adder.



Each section of the latter diagram is a full-adder unit Thus its outputs can be computed by means of:

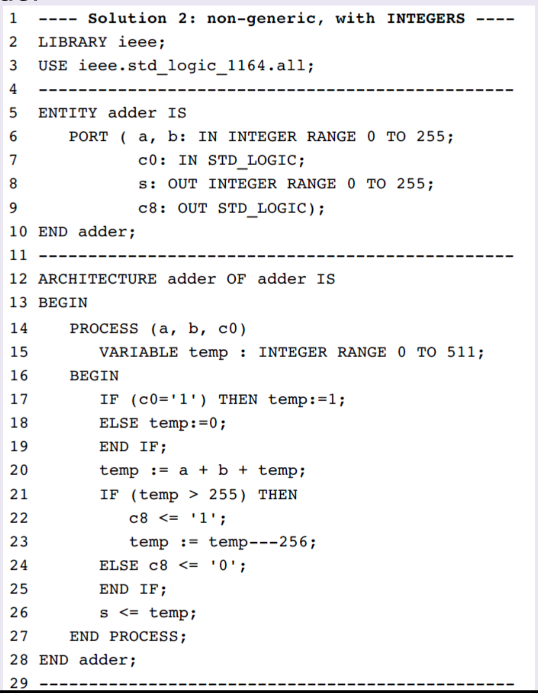
$$s_j = a_j \text{ XOR } b_j \text{ XOR } c_j$$

$$c_{j+1} = (a_j \text{ AND } b_j) \text{ OR } (a_j \text{ AND } c_j) \text{ OR } (b_j \text{ AND } c_j)$$

Example 8: Carry Ripple Adder

```
----- Solution 1: Generic, with VECTORS --------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
------------------------------------------------
ENTITY adder IS
   GENERIC (length : INTEGER := 8);
   PORT ( a, b: IN STD_LOGIC_VECTOR (length-1 DOWNTO 0);
          cin: IN STD_LOGIC;
          s: OUT STD_LOGIC_VECTOR (length-1 DOWNTO 0);
          cout: OUT STD_LOGIC);
END adder;
------------------------------------------------
ARCHITECTURE adder OF adder IS
BEGIN
   PROCESS (a, b, cin)
      VARIABLE carry : STD_LOGIC_VECTOR (length DOWNTO 0);
   BEGIN
      carry(0) := cin;
      FOR i IN 0 TO length-1 LOOP
         s(i) <= a(i) XOR b(i) XOR carry(i);
         carry(i+1) := (a(i) AND b(i)) OR (a(i) AND
                       carry(i)) OR (b(i) AND carry(i));
      END LOOP;
      cout <= carry(length);
   END PROCESS;
END adder;
------------------------------------------------
```

Example 8: Carry Ripple Adder



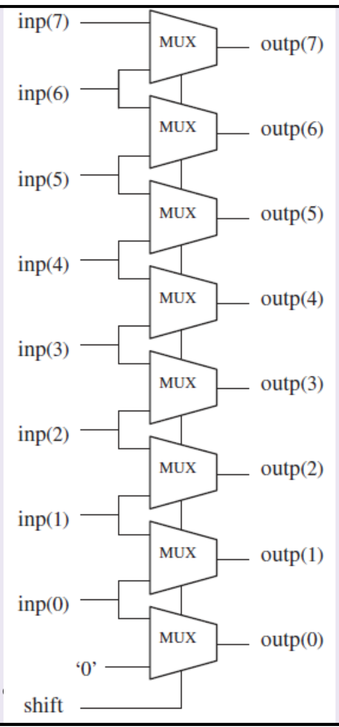
```
---- Solution 2: non-generic, with INTEGERS ----
LIBRARY ieee;
USE ieee.std_logic_1164.all;
------------------------------------------------
ENTITY adder IS
   PORT ( a, b: IN INTEGER RANGE 0 TO 255;
          c0: IN STD_LOGIC;
          s: OUT INTEGER RANGE 0 TO 255;
          c8: OUT STD_LOGIC);
END adder;
------------------------------------------------
ARCHITECTURE adder OF adder IS
BEGIN
   PROCESS (a, b, c0)
      VARIABLE temp : INTEGER RANGE 0 TO 511;
   BEGIN
      IF (c0='1') THEN temp:=1;
      ELSE temp:=0;
      END IF;
      temp := a + b + temp;
      IF (temp > 255) THEN
         c8 <= '1';
         temp := temp---256;
      ELSE c8 <= '0';
      END IF;
      s <= temp;
   END PROCESS;
END adder;
------------------------------------------------
```

Amin Mehranzadeh, Ph.D.

Example 9:

(Simple Barrel Shifter)

the circuit must shift the input vector (of size 8) either 0 or 1 position to the left. When actually shifted (shift = 1), the LSB bit must be filled with ‘0’ (shown in the botton left corner of the diagram). If shift = 0, then outp = inp; if shift = 1, then outp(0) = ‘0’ and outp(i) = inp(i - 1), for 1≤i≤7.



Amin Mehranzadeh, Ph.D.

Example 9:
(Simple Barrel Shifter)

```
---------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------------
ENTITY barrel IS
   GENERIC (n: INTEGER := 8);
   PORT ( inp: IN STD_LOGIC_VECTOR (n-1 DOWNTO 0);
          shift: IN INTEGER RANGE 0 TO 1;
          outp: OUT STD_LOGIC_VECTOR (n-1 DOWNTO 0));
END barrel;
---------------------------------------------
ARCHITECTURE RTL OF barrel IS
BEGIN
   PROCESS (inp, shift)
   BEGIN
      IF (shift=0) THEN
         outp <= inp;
      ELSE
         outp(0) <= '0';
         FOR i IN 1 TO inp'HIGH LOOP
           outp(i) <= inp(i-1);
         END LOOP;
      END IF;
   END PROCESS;
END RTL;
---------------------------------------------
```

Amin Mehranzadeh, Ph.D.

Example 10:
(Leading Zeros)

The design below counts the number of leading zeros in a binary vector, starting from the left end. In this example, the loop will end as soon as a '1' is found in the data vector. Therefore, it is appropriate for counting the number of zeros that precedes the first one.

```
---------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------------
ENTITY LeadingZeros IS
   PORT ( data: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
          zeros: OUT INTEGER RANGE 0 TO 8);
END LeadingZeros;
---------------------------------------------
ARCHITECTURE behavior OF LeadingZeros IS
BEGIN
   PROCESS (data)
      VARIABLE count: INTEGER RANGE 0 TO 8;
   BEGIN
      count := 0;
      FOR i IN data'RANGE LOOP
         CASE data(i) IS
            WHEN '0' => count := count + 1;
            WHEN OTHERS => EXIT;
         END CASE;
      END LOOP;
      zeros <= count;
   END PROCESS;
END behavior;
---------------------------------------------
```

Amin Mehranzadeh, Ph.D.