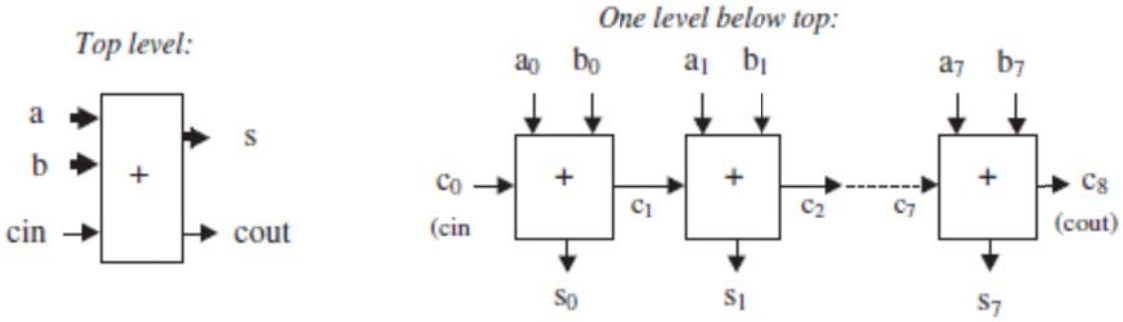
Example: Carry Ripple Adder

For the carry ripple adder shown below a and b are the input vectors to be added, cin is the carry-in bit, s is the sum vector, and cout is the carry-out bit.



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

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

```
1 ----- Solution 1: Generic, with VECTORS --------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ---------------------------------------------
5 ENTITY adder IS
6               GENERIC (length : INTEGER := 8);
7               PORT (a, b: IN STD_LOGIC_VECTOR (length-1 DOWNTO 0);
8                       cin: IN STD_LOGIC;
9                       s: OUT STD_LOGIC_VECTOR (length-1 DOWNTO 0);
10                      cout: OUT STD_LOGIC);
11 END adder;
12 ---------------------------------------------
```

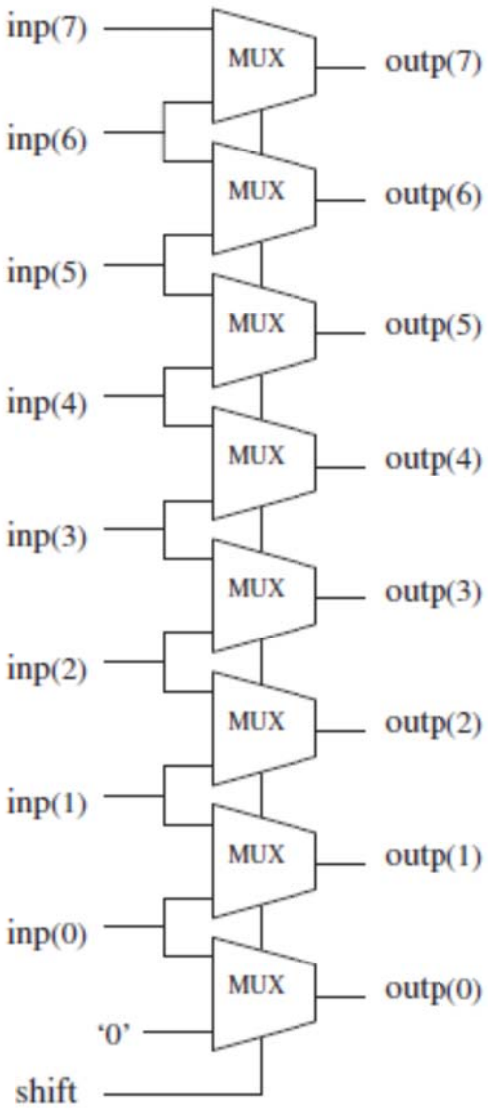
```
13 ARCHITECTURE adder OF adder IS
14 BEGIN
15          PROCESS (a, b, cin)
16          VARIABLE carry : STD_LOGIC_VECTOR (length DOWNTO 0);
17          BEGIN
18          carry(0) := cin;
19          FOR i IN 0 TO length-1 LOOP
20                  s(i) <= a(i) XOR b(i) XOR carry(i);
21                  carry(i+1) := (a(i) AND b(i)) OR (a(i) AND
22                  carry(i)) OR (b(i) AND carry(i));
23          END LOOP;
24          cout <= carry(length);
25 END PROCESS;
26 END adder;
27 ------------------------------------------------
1 ---- Solution 2: non-generic, with INTEGERS ----
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ------------------------------------------------
5 ENTITY adder IS
6           PORT (a, b: IN INTEGER RANGE 0 TO 255;
7                   c0: IN STD_LOGIC;
8                   s: OUT INTEGER RANGE 0 TO 255;
9                   c8: OUT STD_LOGIC);
10 END adder;
```

```
11 ------------------------------------------------
12 ARCHITECTURE adder OF adder IS
13     BEGIN
14             PROCESS (a, b, c0)
15              VARIABLE temp : INTEGER RANGE 0 TO 511;
16             BEGIN
17              IF (c0='1') THEN temp:=1;
18              ELSE temp:=0;
19              END IF;

20              temp := a + b + temp;

21              IF (temp > 255) THEN
22              c8 <= '1';
23              temp := temp---256;
24              ELSE c8 <= '0';
25              END IF;
26              s <= temp;
27 END PROCESS;
28 END adder;
29 ------------------------------------------------
```

Example: Simple Barrel Shifter

If shift=0, then outp = inp;

if shift = 1, then outp(0) = '0' and outp(i) = inp(i - 1), for 1<=i<=7.



```
1 ----------------------------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ----------------------------------------
5 ENTITY barrel IS
6               GENERIC (n: INTEGER := 8);
7               PORT (inp: IN STD_LOGIC_VECTOR (n-1 DOWNTO 0);
8                       shift: IN INTEGER RANGE 0 TO 1;
9                       outp: OUT STD_LOGIC_VECTOR (n-1 DOWNTO 0));
10 END barrel;
11 ----------------------------------------
12 ARCHITECTURE RTL OF barrel IS
13 BEGIN
14              PROCESS (inp, shift)
15              BEGIN
16                      IF (shift=0) THEN
17                              outp <= inp;
18                      ELSE
19                              outp(0) <= '0';
20                              FOR i IN 1 TO inp'HIGH LOOP
21                                      outp(i) <= inp(i-1);
22                              END LOOP;
23                      END IF;
24 END PROCESS;
25 END RTL;
26 ----------------------------------------
```

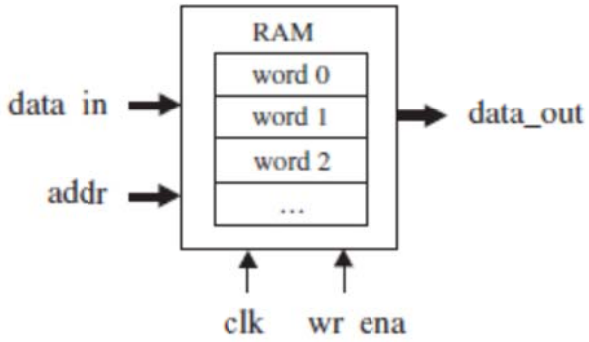
Example: Leading Zeros

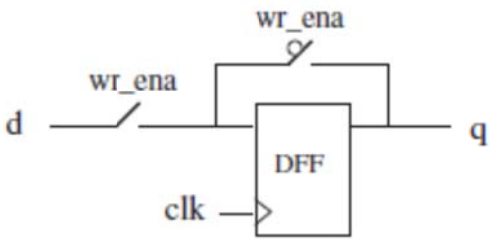
Design a circuit that counts the number of leading zeros in a binary vector, starting from its left end (MSB).

If a '1' is found in the data vector, then EXIT will terminate the loop.

```
1 ----------------------------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ----------------------------------------
5 ENTITY LeadingZeros IS
6       PORT ( data: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
7                 zeros: OUT INTEGER RANGE 0 TO 8);
8 END LeadingZeros;
9 ----------------------------------------
10 ARCHITECTURE behavior OF LeadingZeros IS
11 BEGIN
12     PROCESS (data)
13             VARIABLE count: INTEGER RANGE 0 TO 8;
14     BEGIN
15             count := 0;
16             FOR i IN data'RANGE LOOP
17                     CASE data(i) IS
18                             WHEN '0' => count := count + 1;
19                             WHEN OTHERS => EXIT;
20                     END CASE;
21             END LOOP;
22             zeros <= count;
23 END PROCESS;
24 END behavior;
25 ----------------------------------------
```

Example: RAM
Below is another example using sequential code, particularly the IF statement. We show the implementation of a RAM (random access memory).



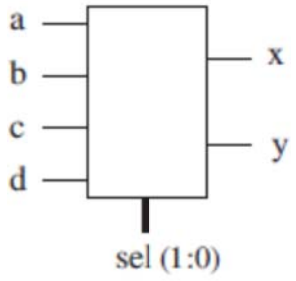


When wr_ena is asserted, at the next rising edge of clk the vector present at data_in must be stored in the position specified by addr. The output, data_out, on the other hand, must constantly display the data selected by addr.

```
1 ------------------------------------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ------------------------------------------------
5 ENTITY ram IS
6       GENERIC ( bits: INTEGER := 8; -- # of bits per word
7                       words: INTEGER := 16); -- # of words in the memory
8       PORT ( wr_ena, clk: IN STD_LOGIC;
9                 addr: IN INTEGER RANGE 0 TO words-1;
10                data_in: IN STD_LOGIC_VECTOR (bits-1 DOWNTO 0);
11                data_out: OUT STD_LOGIC_VECTOR (bits-1 DOWNTO 0));
12 END ram;
13 ------------------------------------------------
14 ARCHITECTURE ram OF ram IS
15              TYPE vector_array IS ARRAY (0 TO words-1) OF
16                      STD_LOGIC_VECTOR (bits-1 DOWNTO 0);
17              SIGNAL memory: vector_array;
18      BEGIN
```

```
19          PROCESS (clk, wr_ena)
20          BEGIN
21                  IF (wr_ena='1') THEN
22                          IF (clk'EVENT AND clk='1') THEN
23                                  memory(addr) <= data_in;
24                          END IF;
25                  END IF;
26          END PROCESS;
27          data_out <= memory(addr);
28 END ram;
29 ---------------------------------------------------
```

Example: Bad Combinational Design



sel	x y
00	a 0
01	b 1
10	c
11	d

The specifications provided for y are incomplete. Using just these specifications, the code could be the following:

```
1 -------------------------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 -------------------------------------
5 ENTITY example IS
6       PORT (a, b, c, d: IN STD_LOGIC;
7                 sel: IN INTEGER RANGE 0 TO 3;
8                 x, y: OUT STD_LOGIC);
9 END example;
10 -------------------------------------

11 ARCHITECTURE example OF example IS
```

```
12 BEGIN
13      PROCESS (a, b, c, d, sel)
14      BEGIN
15                      IF (sel=0) THEN
16                              x<=a;
17                              y<='0';
18                      ELSIF (sel=1) THEN
19                              x<=b;
20                              y<='1';
21                      ELSIF (sel=2) THEN
22                              x<=c;
23                      ELSE
24                              x<=d;
25                      END IF;
26      END PROCESS;
27 END example;
```

The above code can generate unexpected results in synthesis and simulation. So better to write the architecture part as below

```
11 ARCHITECTURE example OF example IS
12 BEGIN
13      PROCESS (a, b, c, d, sel)
14      BEGIN
15                      IF (sel=0) THEN
16                              x<=a;
17                              y<='0';
18                      ELSIF (sel=1) THEN
19                              x<=b;
20                              y<='1';
21                      ELSIF (sel=2) THEN
22                              x<=c; y<='X';
23                      ELSE
24                              x<=d; y<='X';
25                      END IF;
26      END PROCESS;
27 END example;
```

Signals and Variables

CONSTANT

CONSTANT serves to establish default values. Its syntax is shown below.

CONSTANT name : type := value;

Examples:
CONSTANT set_bit : BIT := '1';

CONSTANT datamemory : memory := (('0','0','0','0'),
('0','0','0','1'),
('0','0','1','1'));

A CONSTANT can be declared in a PACKAGE, ENTITY, or ARCHITECTURE.

When declared in a package, it is truly global, for the package can be used by several entities.

The most common places to find a CONSTANT declaration is in an ARCHITECTURE or in a PACKAGE.

SIGNAL

SIGNAL serves to pass values in and out the circuit, as well as between its internal units. In other words, a signal represents circuit interconnects (wires).

SIGNAL name : type [range] [:= initial_value];

Examples:

SIGNAL control: BIT := '0';

SIGNAL count: INTEGER RANGE 0 TO 100;

SIGNAL y: STD_LOGIC_VECTOR (7 DOWNTO 0);

A very important aspect of a SIGNAL, when used inside a section of sequential code (PROCESS, for example), is that its update is not immediate.

Another aspect that might aspect the result is when multiple assignments are made to the same SIGNAL. The compiler might complain and quit synthesis, or might infer the wrong circuit.

Example: Count Ones #1 (not OK)

We want to design a circuit that counts the number of '1's in a binary vector. Let us consider the solution below, which uses only signals.

This code has multiple assignments to the same signal, temp, in lines 15 (once) and 18 (eight times).

Moreover, since the value of a signal is not updated immediately, line 18 conflicts with line 15, for the value assigned in line 15 might not be ready until the conclusion of the PROCESS, in which case a wrong value would be computed in line 18.

In this kind of situation, the use of a VARIABLE is recommended.

```
1 ----------------------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ----------------------------------
5 ENTITY count_ones IS
6       PORT ( din: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
7                ones: OUT INTEGER RANGE 0 TO 8);
8 END count_ones;
9 ----------------------------------
10 ARCHITECTURE not_ok OF count_ones IS
11     SIGNAL temp: INTEGER RANGE 0 TO 8;
12 BEGIN
13     PROCESS (din)
14     BEGIN
15             temp <= 0;
16             FOR i IN 0 TO 7 LOOP
17                     IF (din(i)='1') THEN
18                             temp <= temp + 1;
19                     END IF;
20             END LOOP;
21             ones <= temp;
22     END PROCESS;
23 END not_ok;
24 ----------------------------------
```

With variable use the above code becomes as below

```
1 ---------------------------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ---------------------------------------
5 ENTITY count_ones IS
6       PORT ( din: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
7                 ones: OUT INTEGER RANGE 0 TO 8);
8 END count_ones;
9 ---------------------------------------
10 ARCHITECTURE ok OF count_ones IS
11 BEGIN
12     PROCESS (din)
13     VARIABLE temp: INTEGER RANGE 0 TO 8;
14     BEGIN
15             temp := 0;
16             FOR i IN 0 TO 7 LOOP
17             IF (din(i)='1') THEN
18                     temp := temp + 1;
19             END IF;
20             END LOOP;
21             ones <= temp;
22     END PROCESS;
23 END ok;
24 ---------------------------------------
```