CPE 462
VHDL: Simulation and Synthesis

Topic #06 - c) Wait, Case, Loop

Wait operation

- The operation of WAIT is sometimes similar to that of IF.
- PROCESS cannot have a sensitivity list when WAIT is employed.
- Two synthesizable forms: **WAIT ON** and **WAIT UNTIL**

```
WAIT UNTIL signal_condition;
```

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

WAIT UNTIL

- The WAIT UNTIL statement accepts only one signal
- 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 of a 8-bit register with synchronous reset.

```
entity waituntil is
  port (clk,rst : in bit;
  input : in bit_vector(7 downto 0);
  output : out bit_vector(7 downto 0)
  );
end entity;

architecture myarch of waituntil is
begin
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;
end architecture;
```

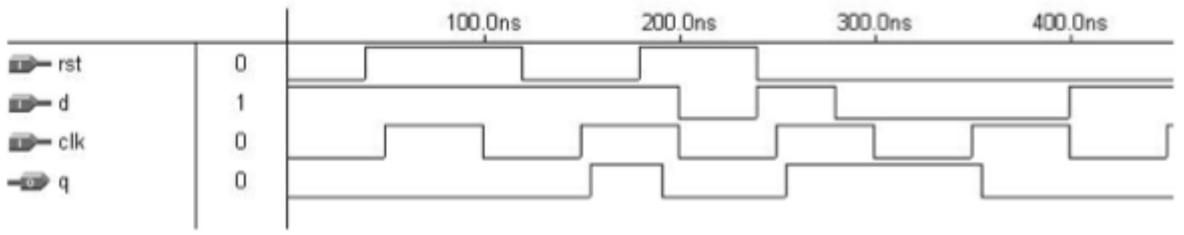
WAIT ON

- 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, the PROCESS will continue execution whenever a change in rst or clk occurs.
- Example of a 8-bit register with synchronous reset.

```
entity waiton is
  port (clk,rst : in bit;
  input : in bit_vector(7 downto 0);
  output : out bit_vector(7 downto 0)
  );
end entity;

architecture myarch of waiton is
begin
  process          -- no sensitivity list
    begin
      wait on clk, rst;
      if (rst='1') then
        output <= "00000000";
      elsif (clk'event and clk='1') then
        output <= input;
      end if;
end process;
```

DFF with Asynchronous Reset



- This code shows the behavior of a standard DFF.
- The output q becomes d whenever clock is high.

```
entity dff is
   port (d, clk, rst: in bit;
   q : out bit);
end entity;


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;
```

CASE

- CASE is another statement intended exclusively for sequential code (along with IF, LOOP, and WAIT).
- Very similar to concurrent WHEN statements
- Its syntax is shown below:

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

CASE example

- 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

```
library ieee; use ieee.std_logic_1164.all;

entity casestatement is
  port (a,b,c : in std_logic_vector (3 downto 0);
       control : in std_logic_vector(1 downto 0);
       x,y : out std_logic_vector(3 downto 0));
end entity;

architecture myarch of casestatement is
begin
  process (control)
    begin
      case control IS
         when "00" => x<=a; y<=b;
         when "01" => x<=b; y<=c;
         when others => x<="0000"; y<="ZZZZ";
      end case;
   end process;

end architecture;
```

CASE example

- The **=>** symbol looks strange but it is part of the when line.

```
library ieee; use ieee.std_logic_1164.all;

entity casestatement is
  port (a,b,c : in std_logic_vector (3 downto 0);
        control : in std_logic_vector(1 downto 0);
        x,y : out std_logic_vector(3 downto 0));
end entity;

architecture myarch of casestatement is
begin
  process (control)
    begin
      case control IS
         when "00" => x<=a; y<=b;
         when "01" => x<=b; y<=c;
         when others => x<="0000"; y<="ZZZZ";
      end case;
   end process;

end architecture;
```

A different form for CASE

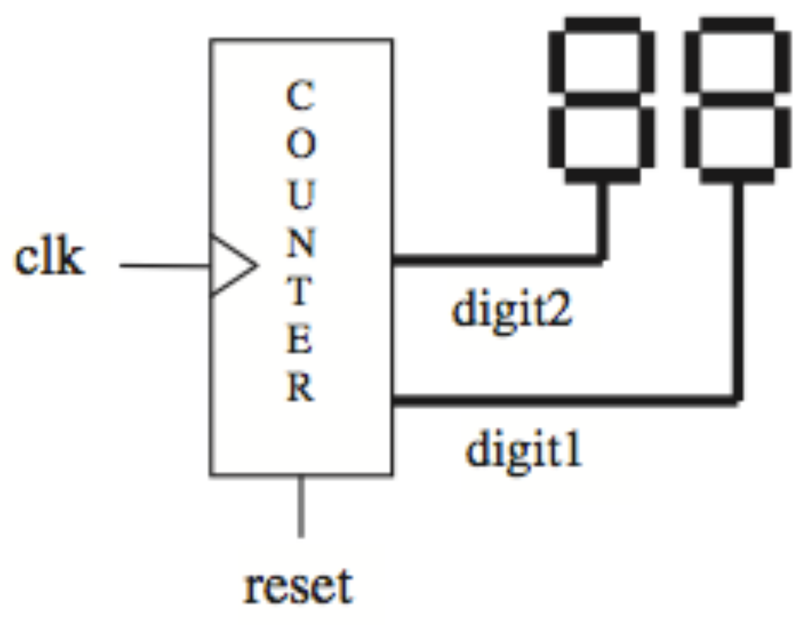
- ‘WHEN value” can take up three forms:

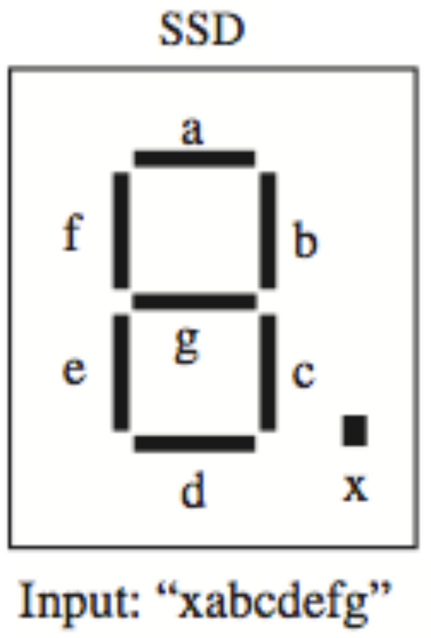
```
WHEN value                   -- single value
WHEN value1 to value2        -- range, for enumerated data types
                             -- only
WHEN value1 | value2 |...    -- value1 or value2 or ...
```

```
----------------------------------------------
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;
----------------------------------------------
```

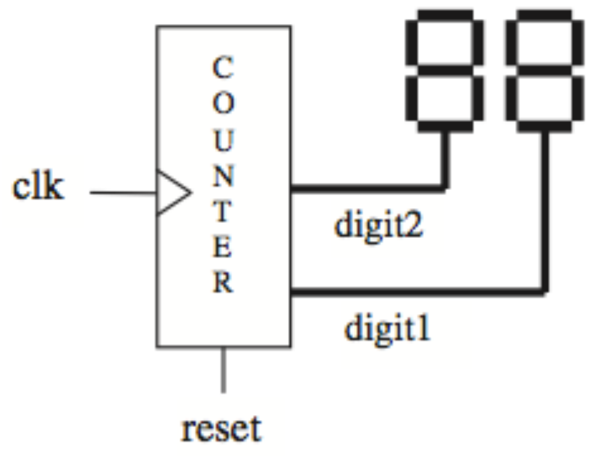
Two-digit Counter with SSD Output

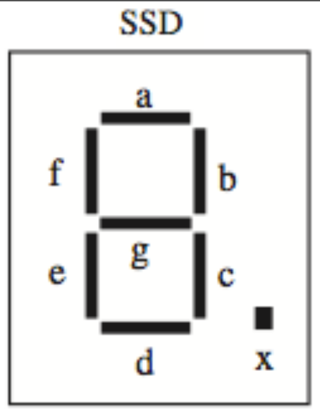
- We want a progressive 2-digit decimal counter (0 to 99 to 0), with external asynchronous reset plus binary-coded decimal (BCD) to seven-segment display (SSD) conversion.





WESTERN NEW ENGLAND UNIVERSITY WNE





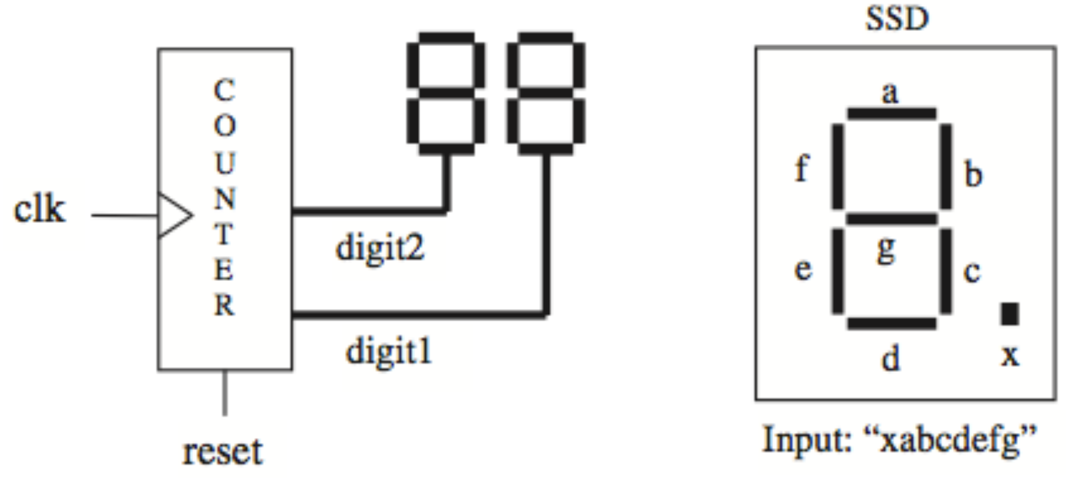
Input: “xabcdefg”

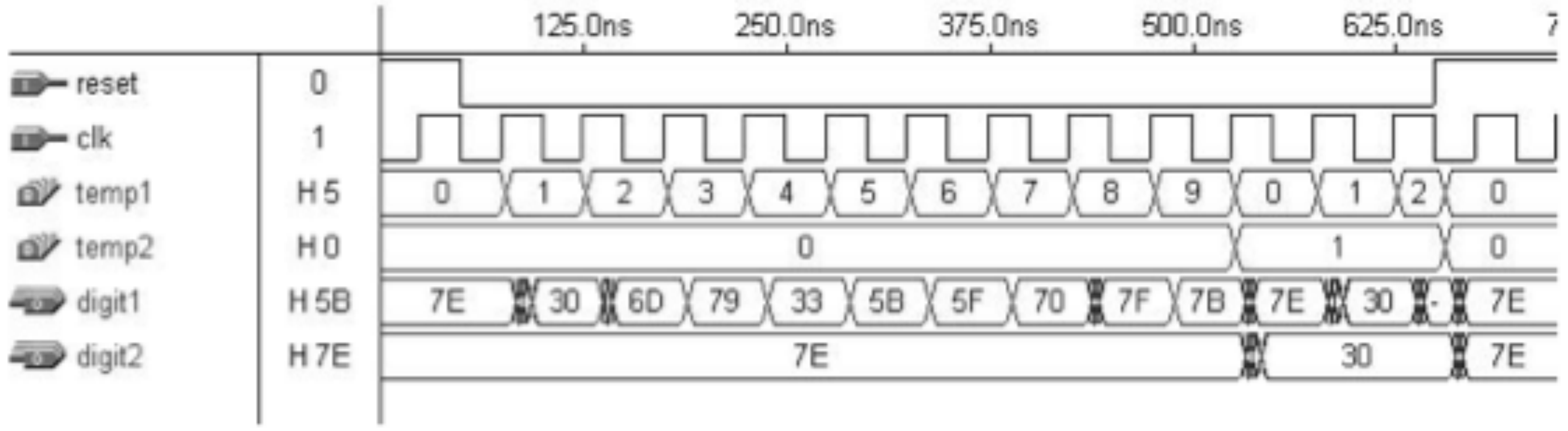
```
--------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
--------------------------------------------------
ENTITY counter IS
   PORT (clk, reset : IN STD_LOGIC;
         digit1, digit2 : OUT STD_LOGIC_VECTOR (6 DOWNTO 0));
END counter;
--------------------------------------------------
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;
            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;
```

redundant code?

SSD Simulation

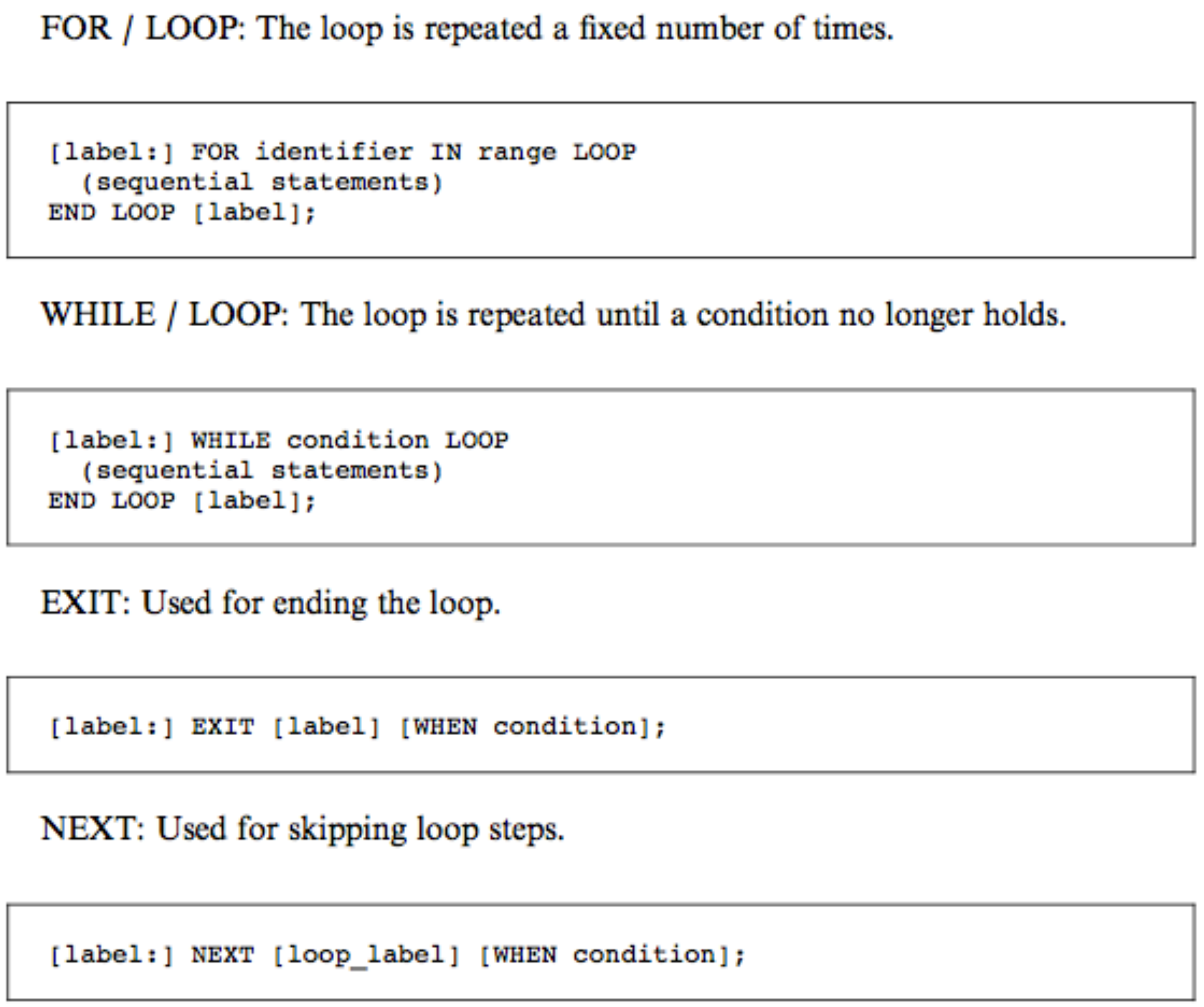
- Using CASE statements made it extremely easy to convert a binary signal into a seven segment display digit.




125.0ns
250.0ns
375.0ns
500.0ns
625.0ns
reset 0
clk 1
temp1 H5: 0 1 2 3 4 5 6 7 8 9 0 1 2 0
temp2 H0: 0 1 0
digit1 H5B: 7E 30 6D 79 33 5B 5F 70 7F 7B 7E 30 7E
digit2 H7E: 7E 30 7E

Loop

- LOOP is useful when a piece of code must be instantiated several times. LOOP is exclusively for sequential code.
- Both limits of the loop range must be static!



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

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

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

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

EXIT: Used for ending the loop.

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

NEXT: Used for skipping loop steps.

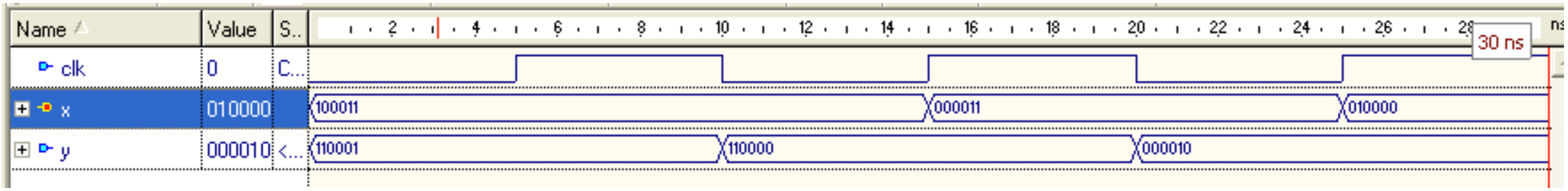
```
[label:] NEXT [loop_label] [WHEN condition];
```

Loop example

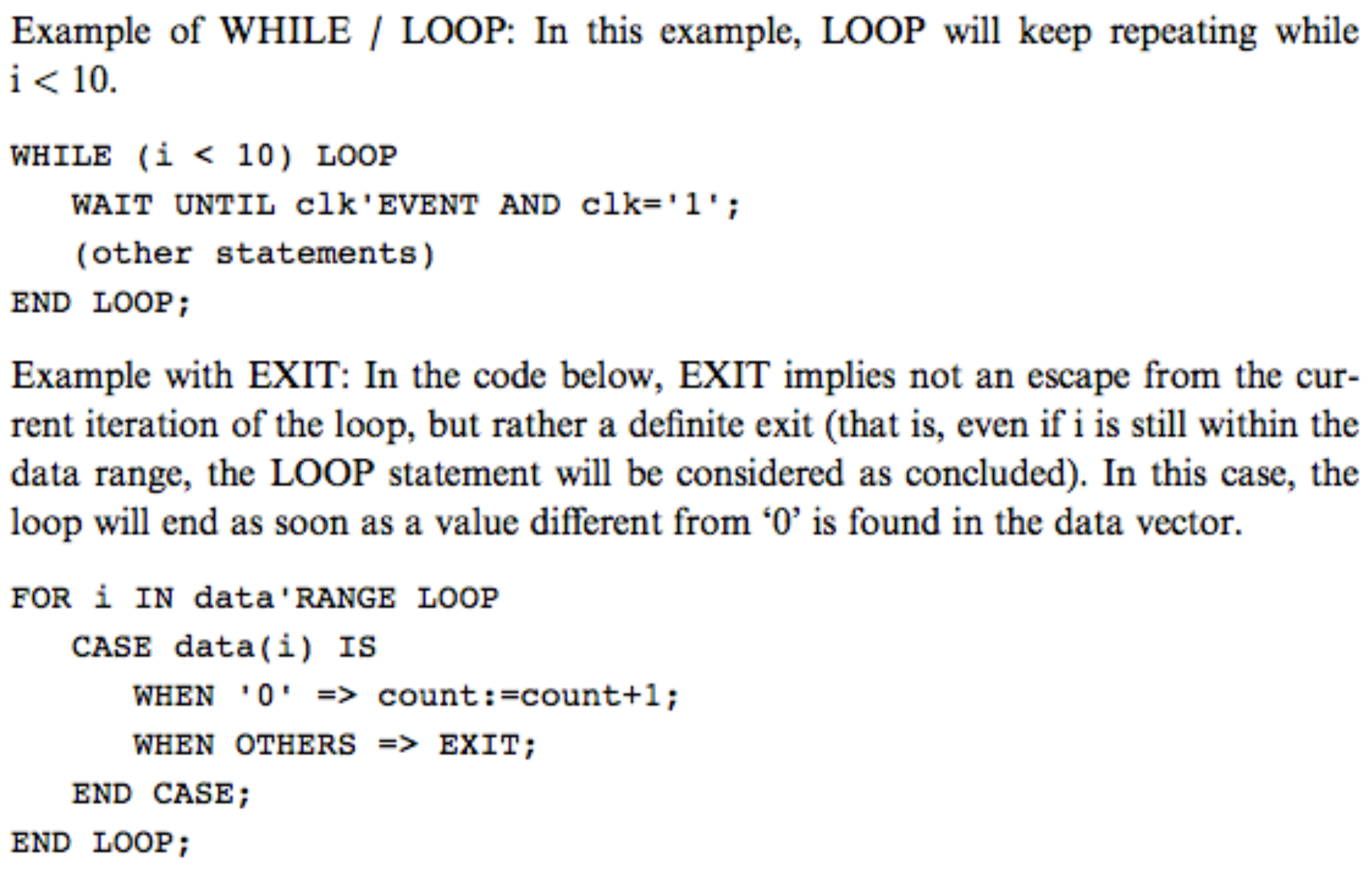
```
entity loopexample is
  port (y: in bit_vector (5 downto 0);
       clk : in bit;
       x : out bit_vector(5 downto 0));
end entity;

architecture myarch of loopexample is
begin
  process (clk)
  begin
   for i in 0 to 5 loop x(i) <= y(y'high-i); end loop;
  end process;

end architecture;
```



LOOP snippets



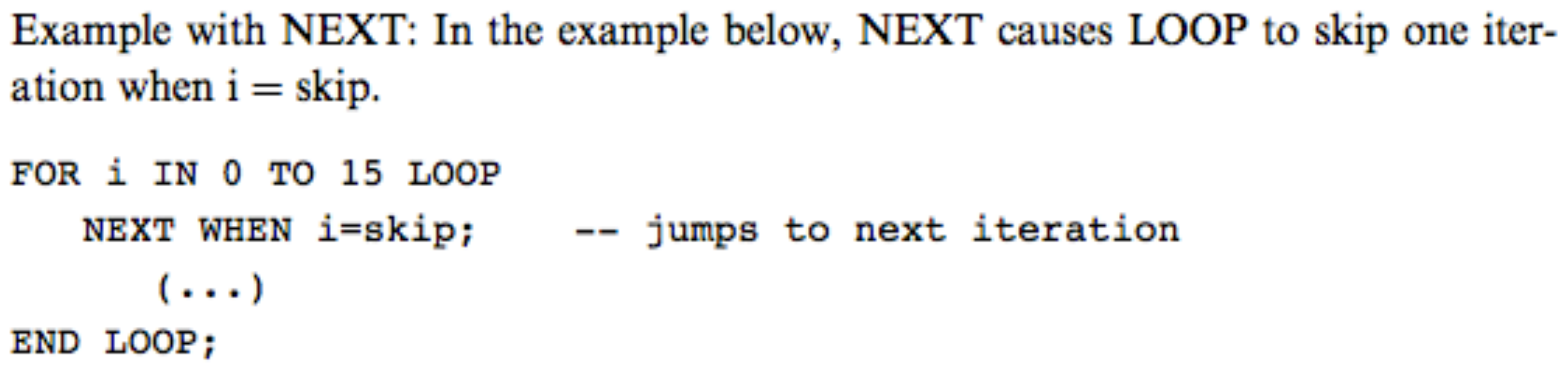
Example of WHILE / LOOP: In this example, LOOP will keep repeating while i < 10.

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

Example with EXIT: In the code below, EXIT implies not an escape from the current iteration of the loop, but rather a definite exit (that is, even if i is still within the data range, the LOOP statement will be considered as concluded). 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;
```

More LOOP snippets



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;
```