Integrated Circuits Design by FPGA

م.م. أحمد مؤيد عبدالحسين

جامعة الفرات الأوسط التقنية / الكلية التقنية الهندسية / نجف

Lecture 5

VHDL Sequential Code

Objectives of this Lecture

- To understand what is the Sequential VHDL code.

- Also, this lecture is very important, for it allows a better understanding of where the parallel VHDL code or sequential VHDL code, as well as the consequences of using one or the other.

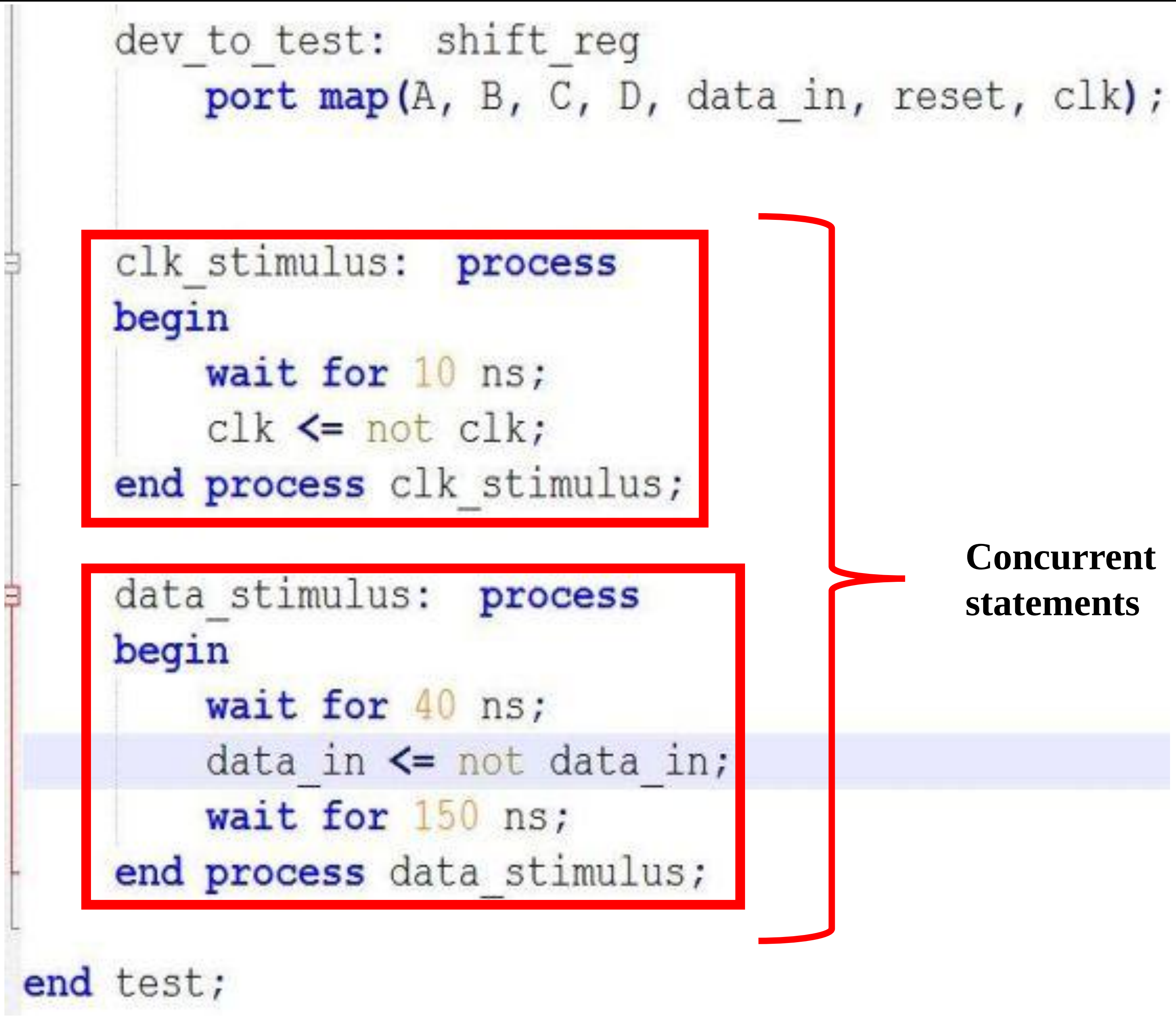
Contents of this Lecture

- Introduction about Sequential VHDL code.
- Sequential VHDL code inside **Processes**
- Sequential VHDL code inside **Processes**\ **IF**
- Sequential VHDL code inside **Processes**\ **WAIT**
- Sequential VHDL code inside **Processes**\ **CASE**

Introduction about Sequential VHDL code

- VHDL code can be concurrent (parallel) or sequential.
- As mentioned in lecture 4, 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 .

Introduction about Sequential VHDL code



```
    dev_to_test:  shift_reg
        port map(A, B, C, D, data_in, reset, clk);


    clk_stimulus:  process
    begin
        wait for 10 ns;
        clk <= not clk;
    end process clk_stimulus;

    data_stimulus:  process
    begin
        wait for 40 ns;
        data_in <= not data_in;
        wait for 150 ns;
    end process data_stimulus;

end test;
```

Concurrent statements

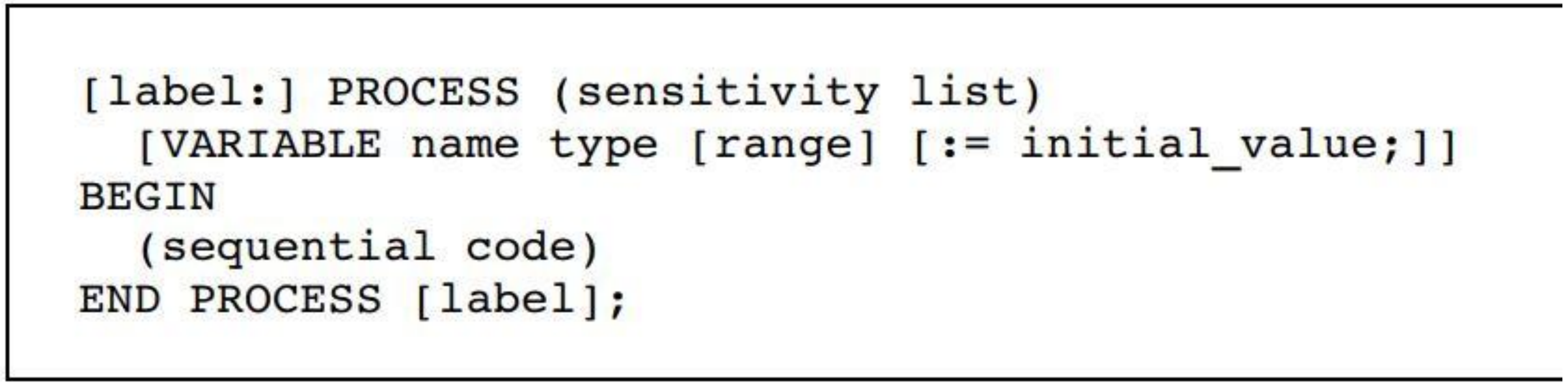
Introduction about Sequential VHDL code

- 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).

Sequential VHDL code inside Processes

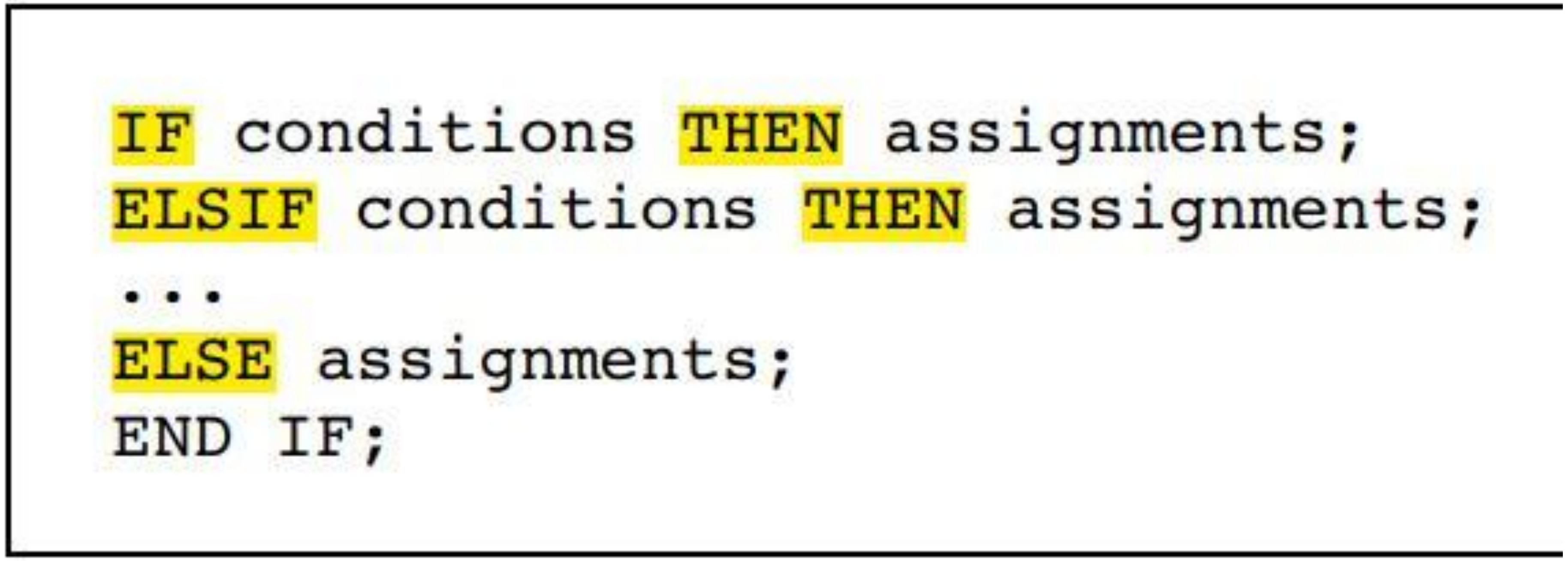
- A PROCESS is a sequential section of VHDL code. It is characterized by the presence of **IF**, **WAIT**, **CASE**, or **LOOP**, and by a sensitivity list (except when WAIT is used).
- A PROCESS must be installed in the main code, and is executed every time a signal in the **sensitivity list changes**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



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

Sequential VHDL code inside Processes \IF

IF structure



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

Sequential VHDL code inside Processes \IF

Example 6.1

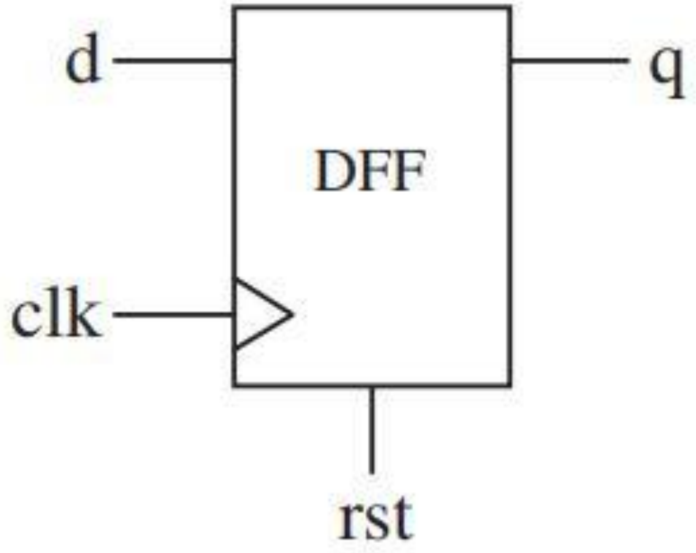
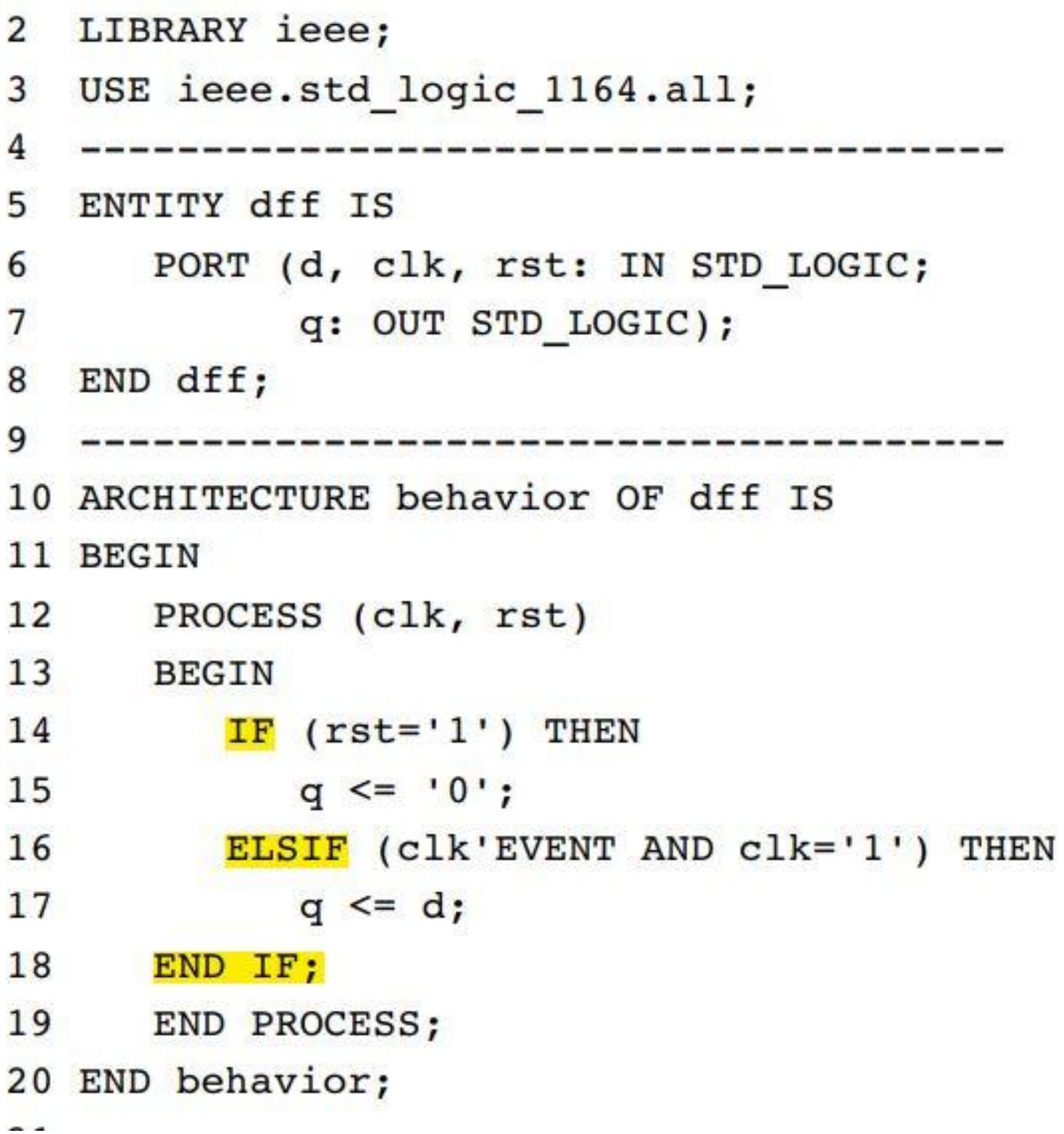


Figure 6.1
DFF with asynchronous reset of example 6.1.

Sequential VHDL code inside Processes \IF



```
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;
```

Sequential VHDL code inside Processes \IF

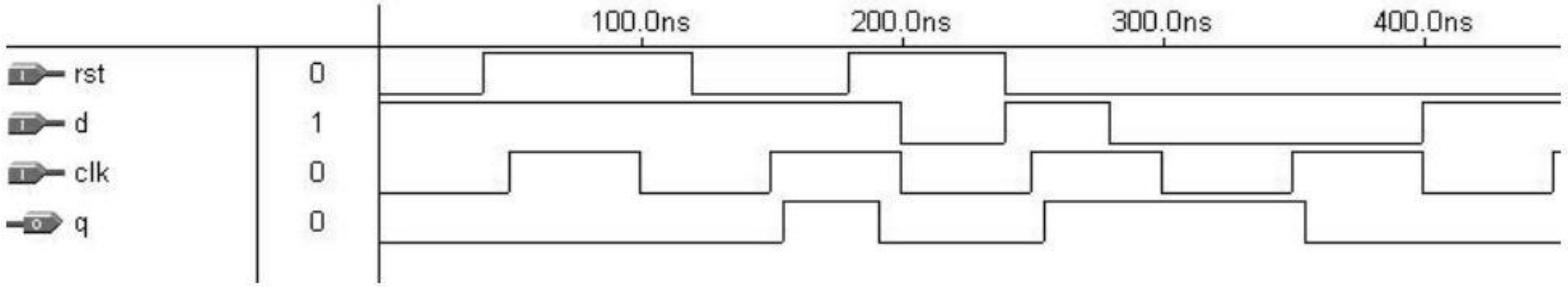


Figure 6.2
Simulation results of example 6.1.

DFF : D Flip Flop

Sequential VHDL code inside Processes \IF

Example 6.2

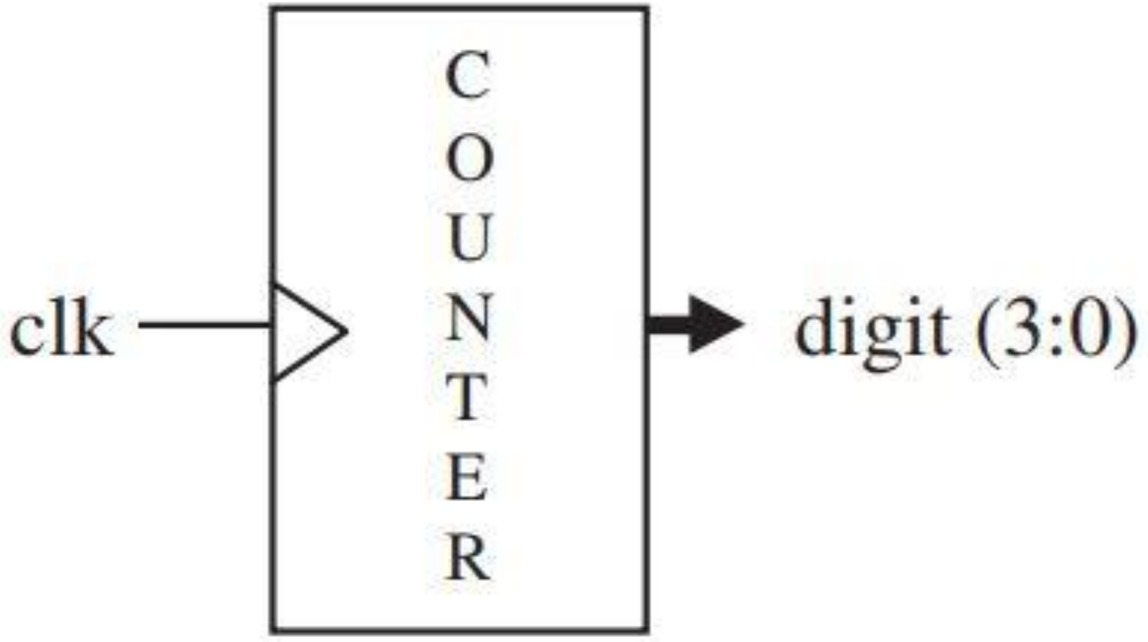
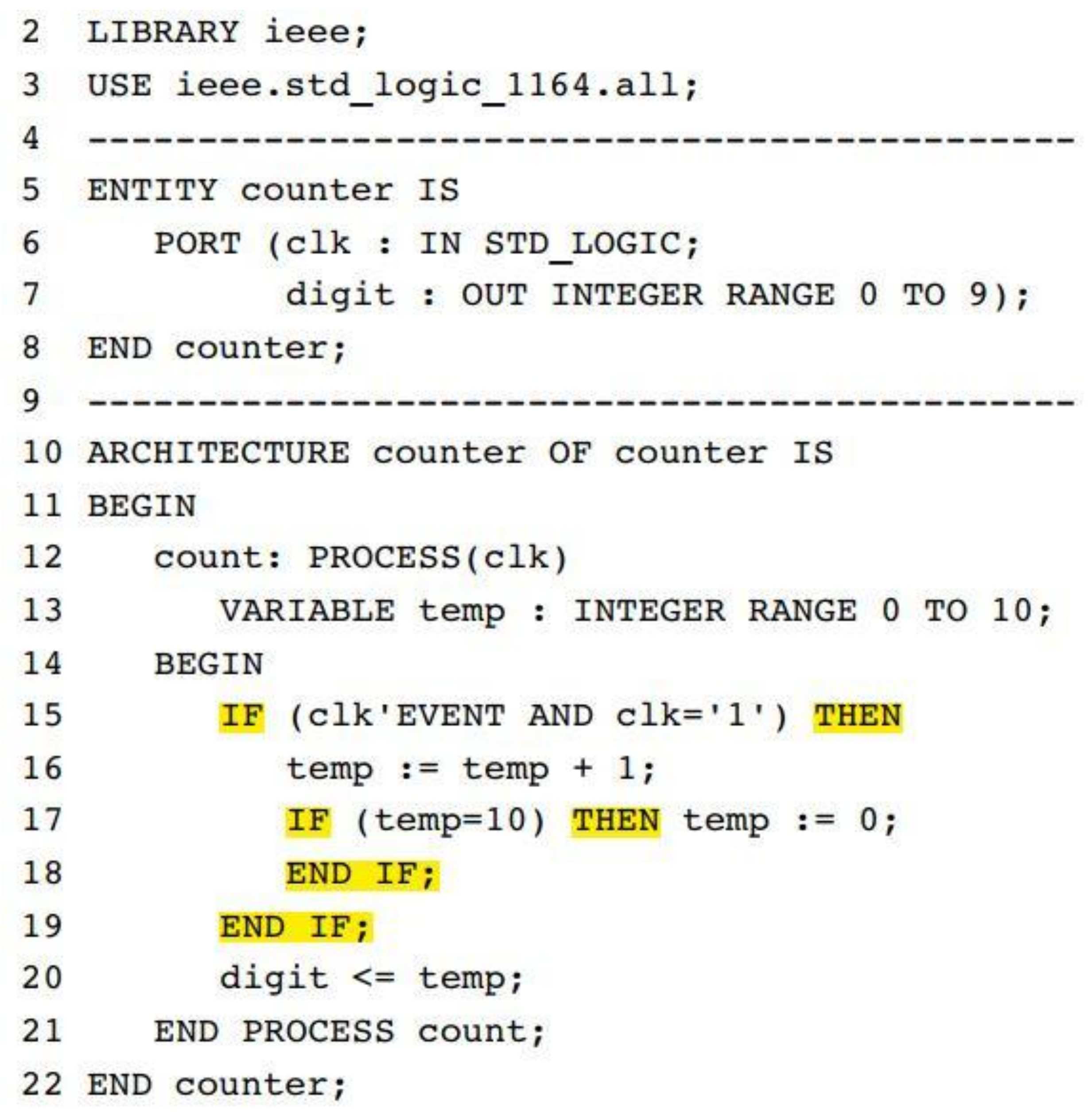


Figure 6.3
Counter of example 6.2.

Sequential VHDL code inside Processes \IF



```vhdl
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;
```

Sequential VHDL code inside Processes \IF

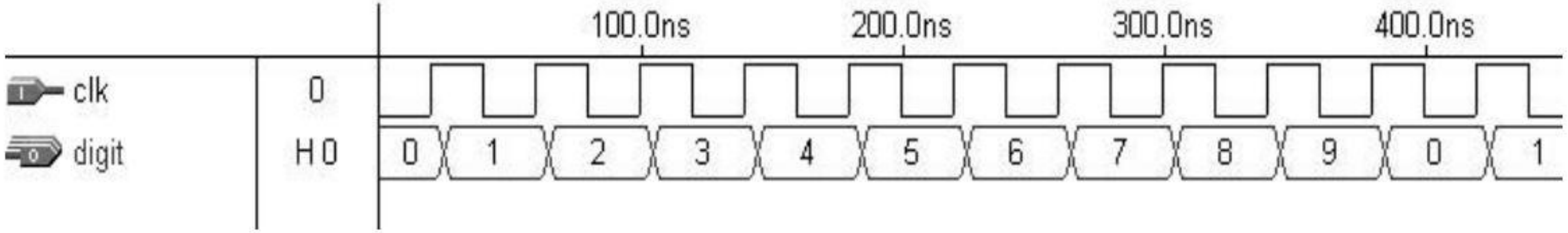
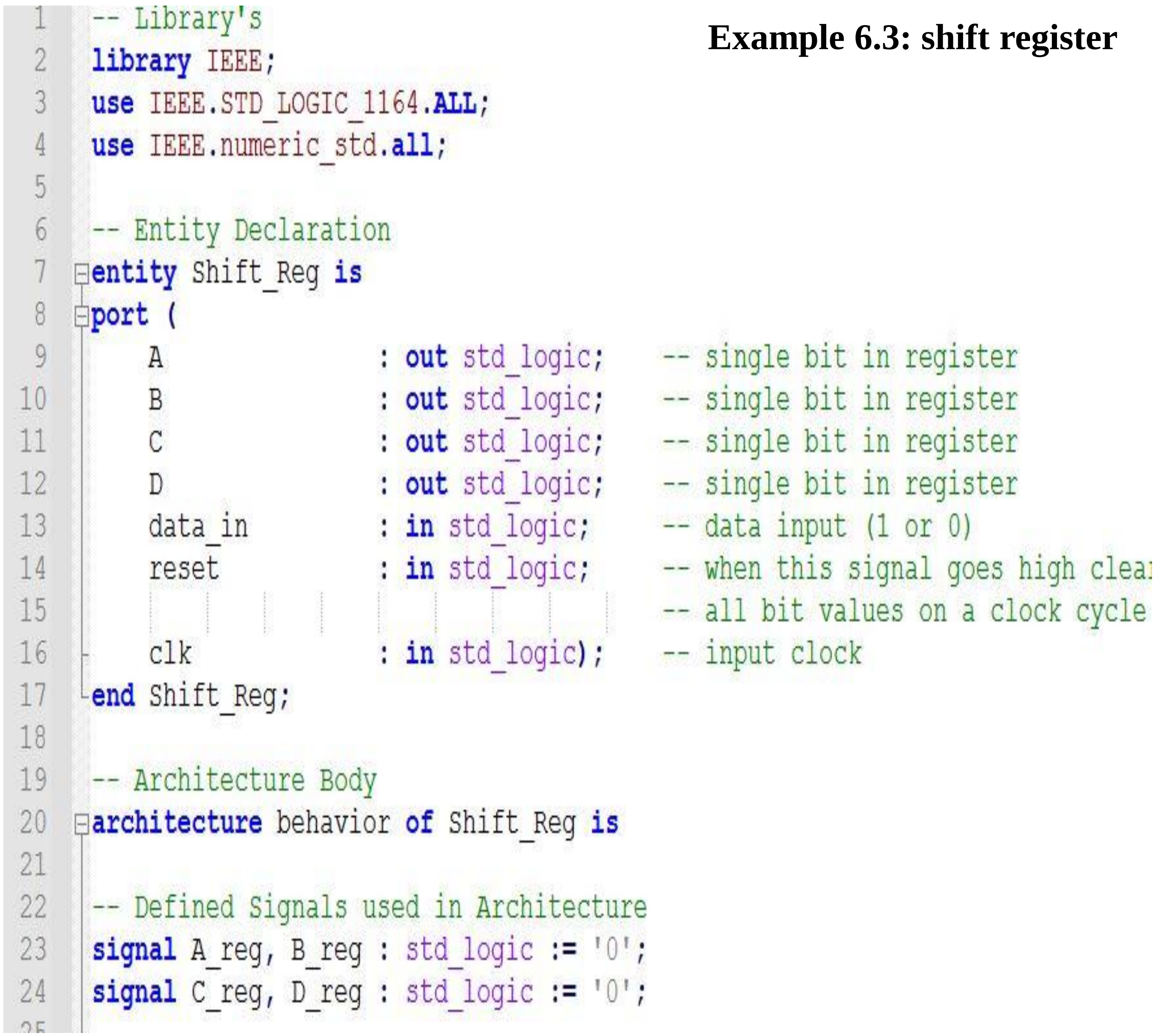


Figure 6.4
Simulation results of example 6.2.

Counter

Sequential VHDL code inside Processes \IF

Example 6.3: shift register

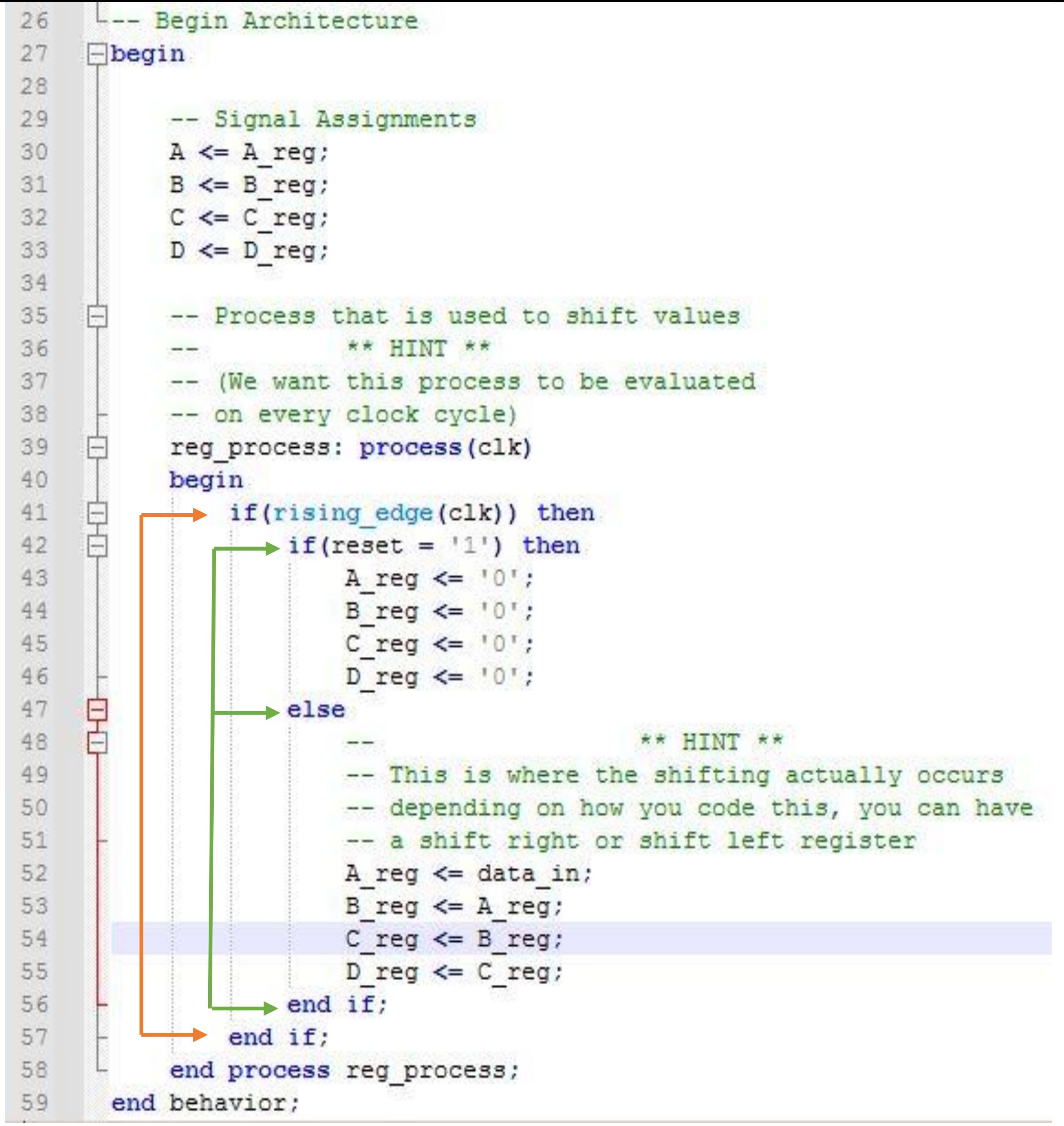


```vhdl
-- Library's
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

-- Entity Declaration
entity Shift_Reg is
port (
        A               : out std_logic;    -- single bit in register
        B               : out std_logic;    -- single bit in register
        C               : out std_logic;    -- single bit in register
        D               : out std_logic;    -- single bit in register
        data_in         : in std_logic;     -- data input (1 or 0)
        reset           : in std_logic;     -- when this signal goes high clea
                                            -- all bit values on a clock cycle
        clk             : in std_logic);    -- input clock
end Shift_Reg;

-- Architecture Body
architecture behavior of Shift_Reg is

-- Defined Signals used in Architecture
signal A_reg, B_reg : std_logic := '0';
signal C_reg, D_reg : std_logic := '0';
```

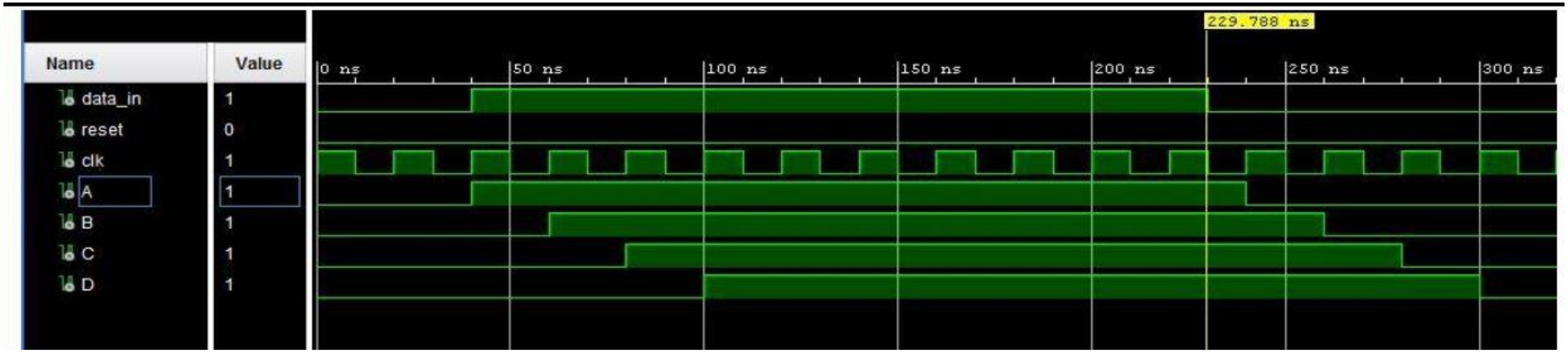


```vhdl
-- Begin Architecture
begin

    -- Signal Assignments
    A <= A_reg;
    B <= B_reg;
    C <= C_reg;
    D <= D_reg;

    -- Process that is used to shift values
    --            ** HINT **
    -- (We want this process to be evaluated
    -- on every clock cycle)
    reg_process: process(clk)
    begin
        if(rising_edge(clk)) then
            if(reset = '1') then
                A_reg <= '0';
                B_reg <= '0';
                C_reg <= '0';
                D_reg <= '0';
            else
                --                  ** HINT **
                -- This is where the shifting actually occurs
                -- depending on how you code this, you can have
                -- a shift right or shift left register
                A_reg <= data_in;
                B_reg <= A_reg;
                C_reg <= B_reg;
                D_reg <= C_reg;
            end if;
        end if;
    end process reg_process;
end behavior;
```

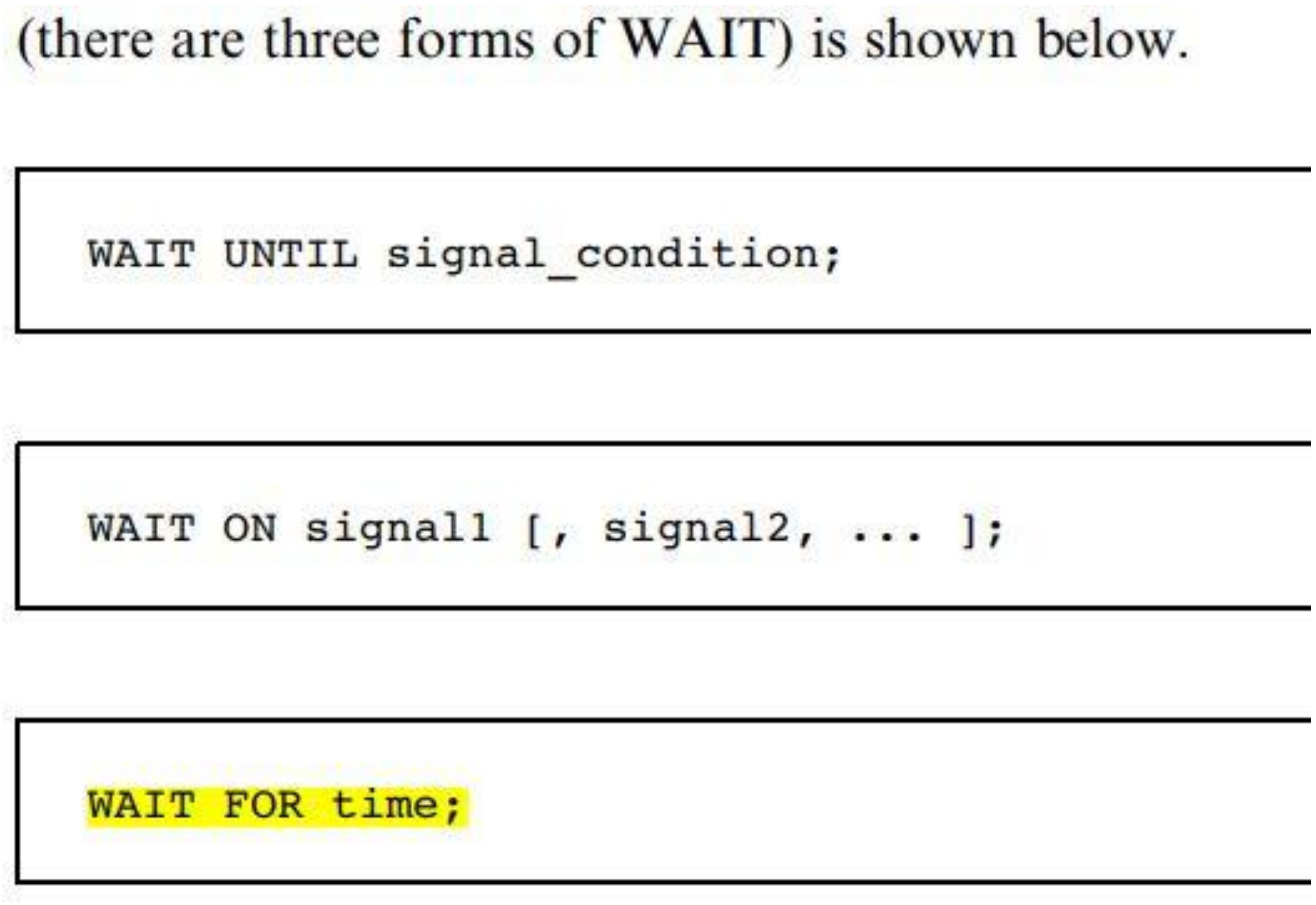
Sequential VHDL code inside Processes \IF



Name	Value
data_in	1
reset	0
clk	1
A	1
B	1
C	1
D	1

Example 6.3: shift register

Sequential VHDL code inside Processes \WAIT



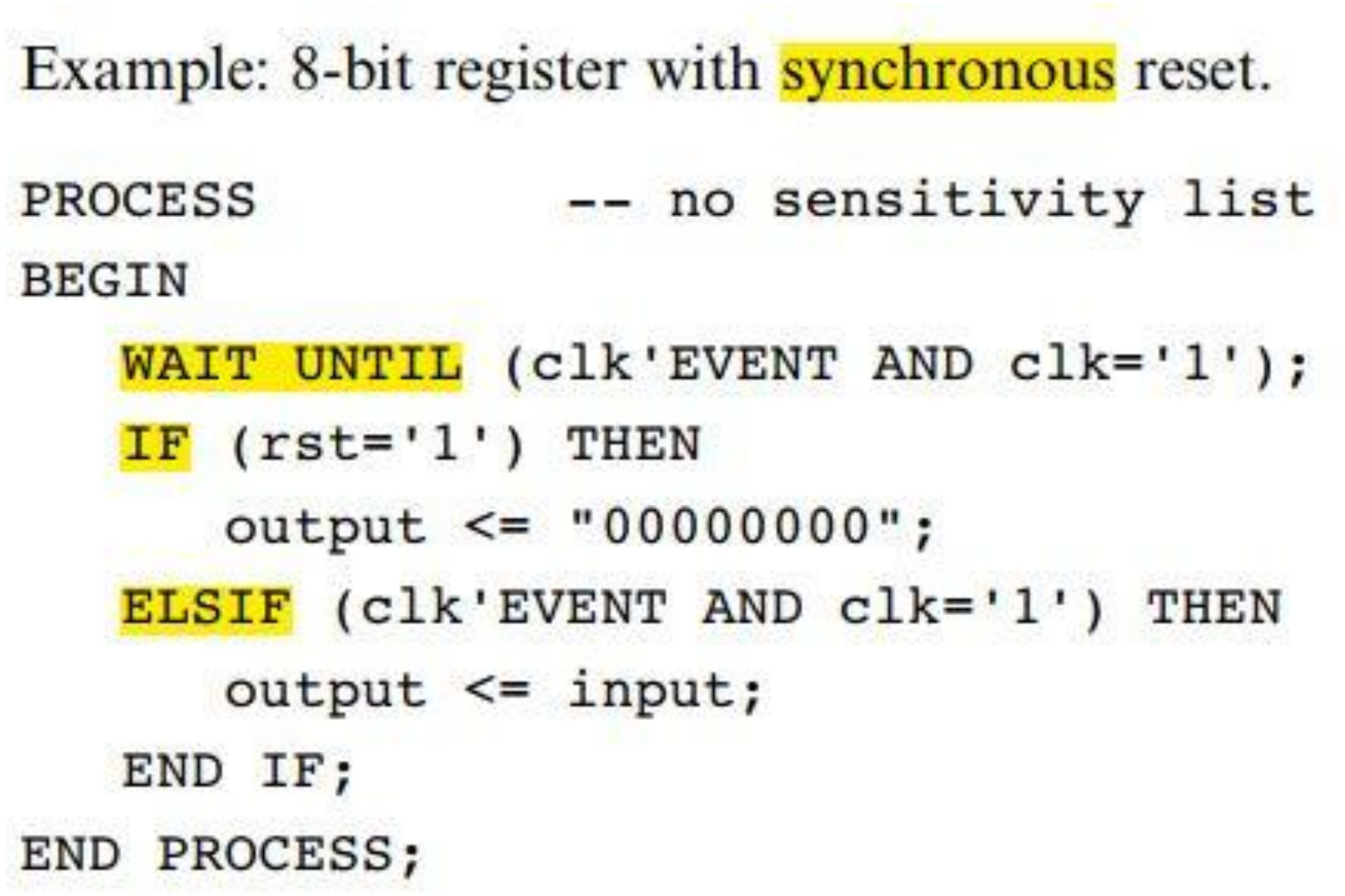
(there are three forms of WAIT) is shown below.

```
WAIT UNTIL signal_condition;
```

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

```
WAIT FOR time;
```

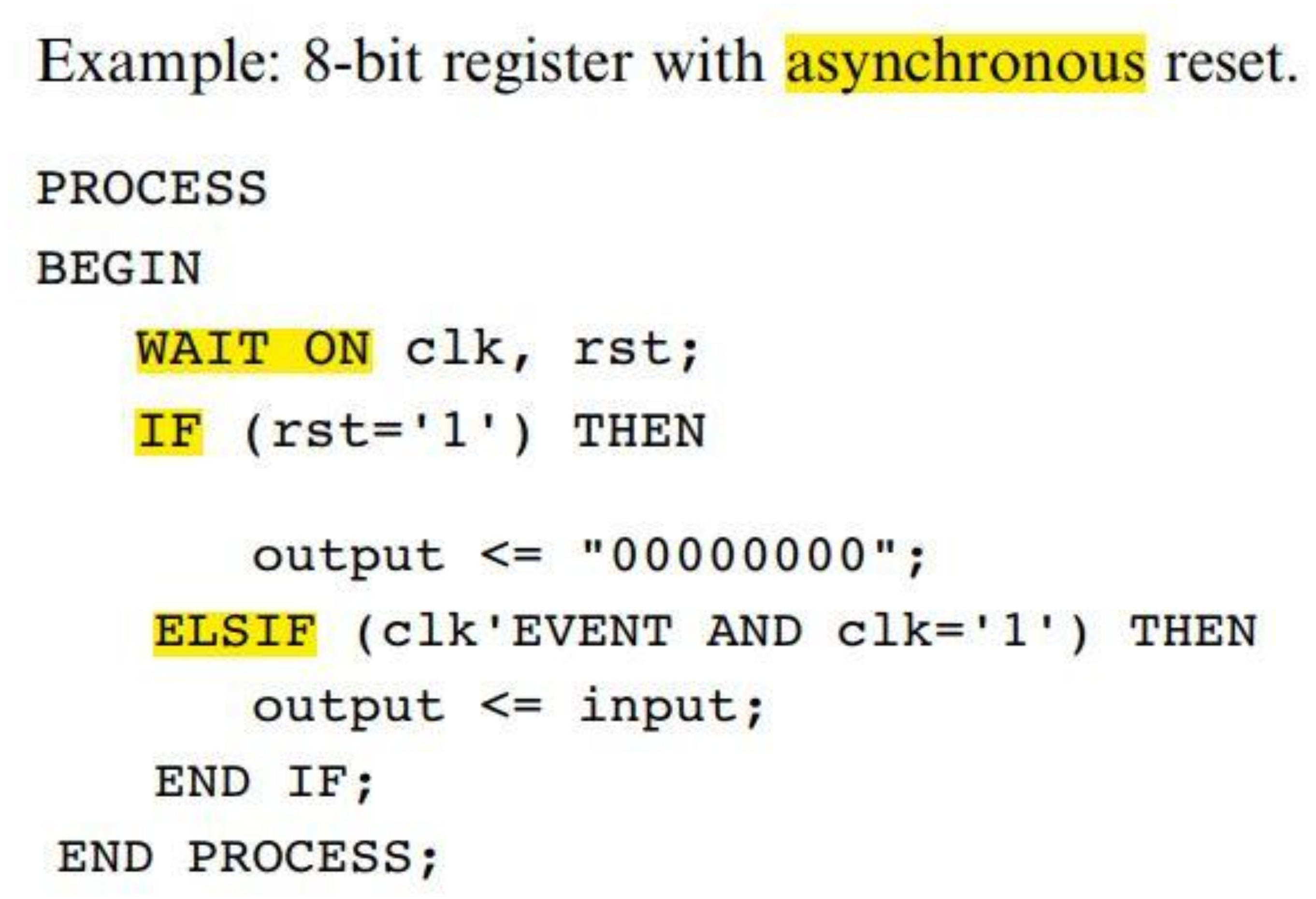
Sequential VHDL code inside Processes \WAIT



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;
```

Sequential VHDL code inside Processes \WAIT



Example: 8-bit register with asynchronous reset.

```
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;
```

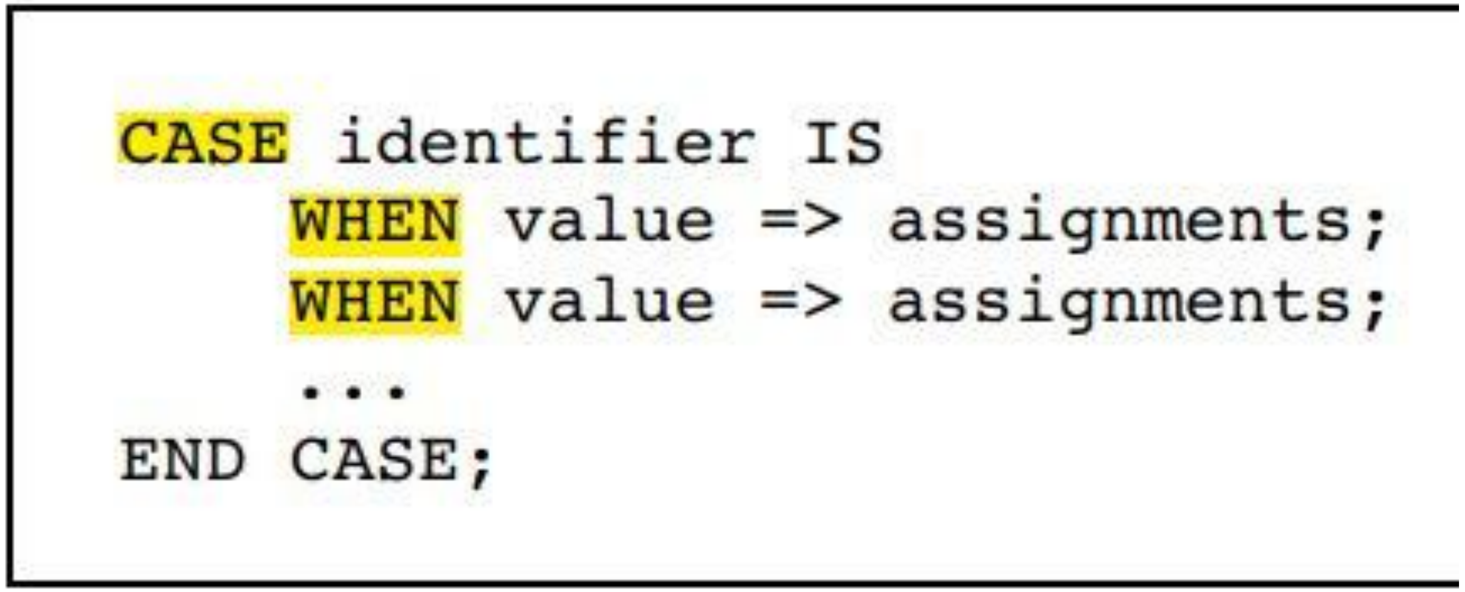
Sequential VHDL code inside Processes \WAIT

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

WAIT FOR 5ns;

Sequential VHDL code inside Processes \CASE

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



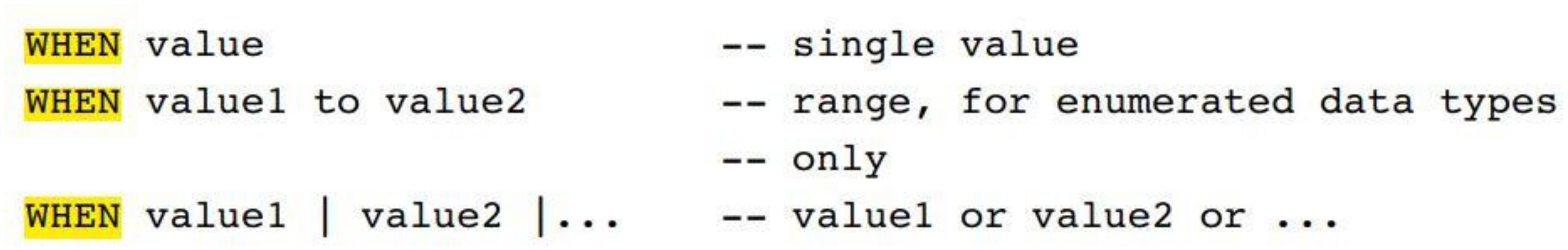
```
CASE identifier IS
    WHEN value => assignments;
    WHEN value => assignments;
    ...
END CASE;
```

- The **CASE** statement (sequential) is very similar to **WHEN** (combinational).

Note : Parallel = Concurrent = Combinational

Sequential VHDL code inside Processes \CASE

- Here too all permutations must be tested.
- However, **CASE** allows multiple assignments for each test condition (as shown in the previous slide), while **WHEN** allows only one.
- Like in the case of **WHEN** (lecture 4), 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 ...
```

Sequential VHDL code inside Processes \CASE

Example 6.7:
2-digit counter

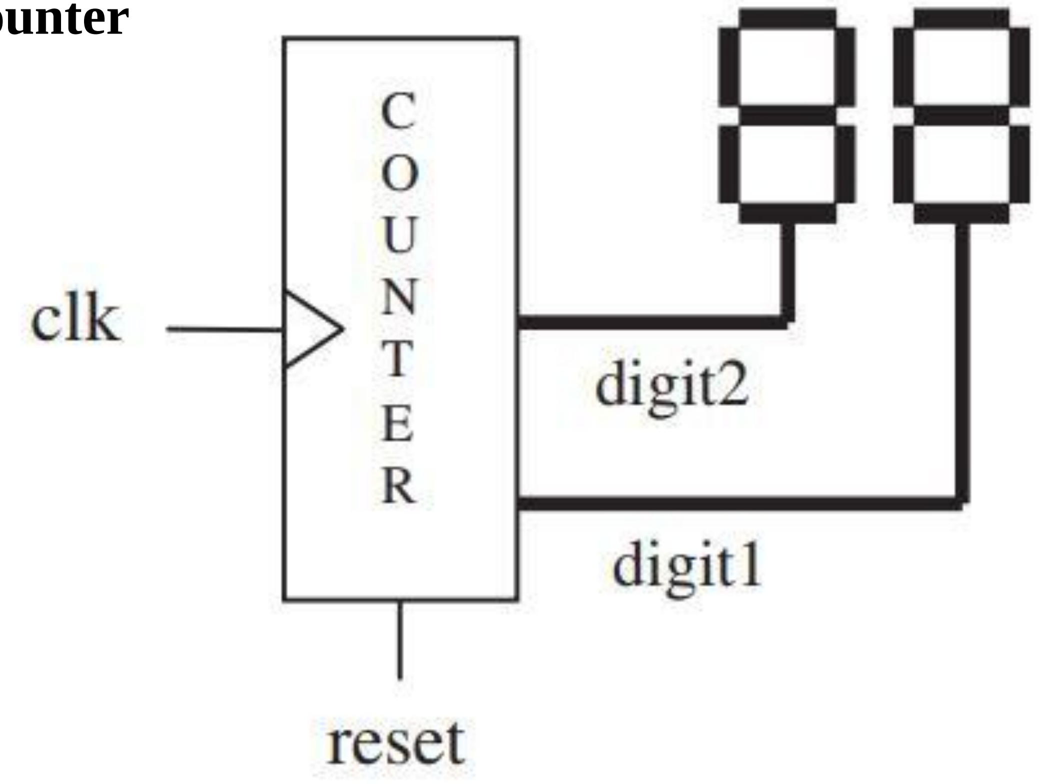
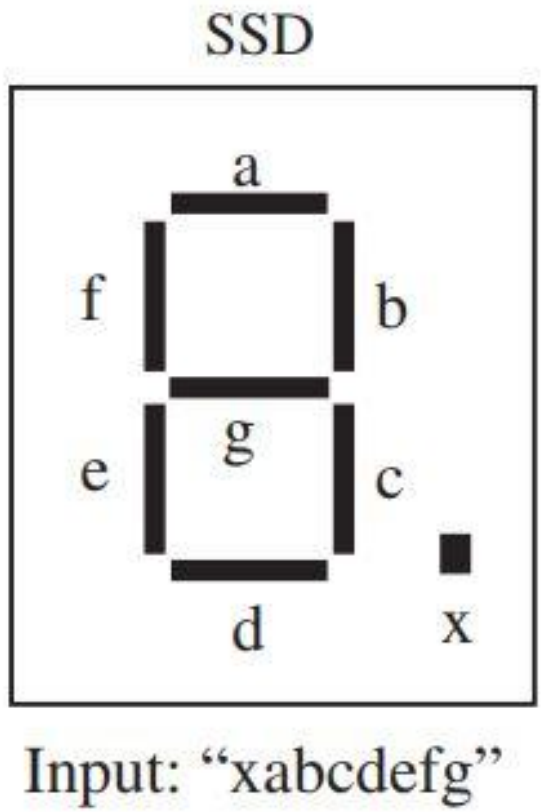


Figure 6.7
2-digit counter of example 6.7.



a	b	c	d	e	f	g
1	1	1	1	1	1	1

Sequential VHDL code inside Processes \CASE

```
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;
```

Sequential VHDL code inside Processes \CASE

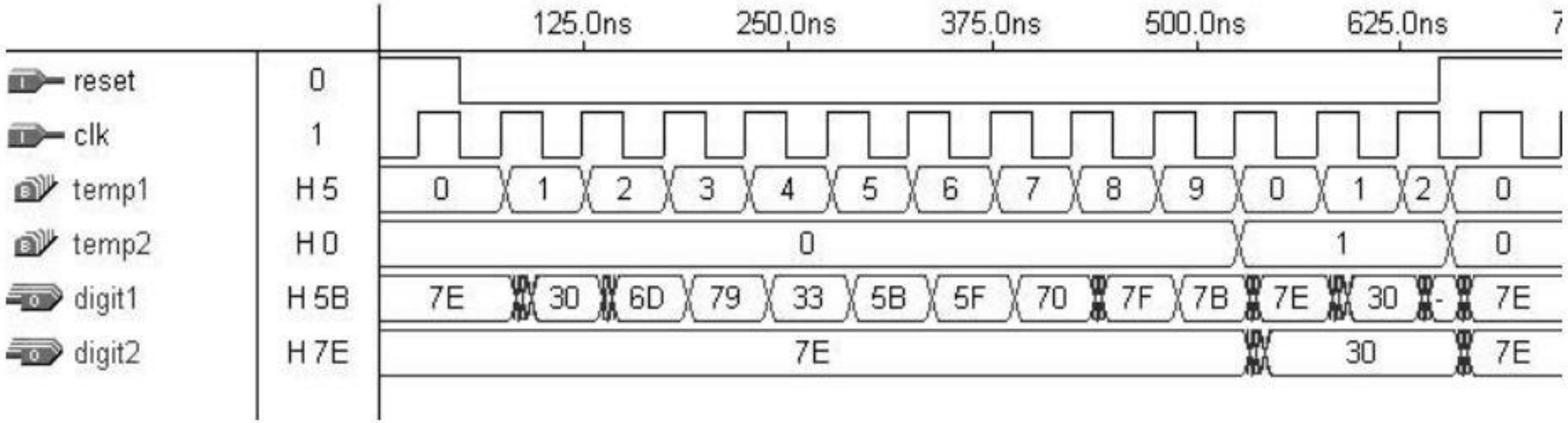


Figure 6.8
Simulation results of example 6.7.

2-digit counter

Assignments

- Design the same **DFF** of example 6.1 by using **WAIT** statement.
- Design the same progressive 1-digit decimal **counter** of example 6.2 by using **WAIT** statement.

End of lecture 5

Any Questions ?