Integrated Circuits Design by FPGA

م.م. أحمد مؤيد عبدالحسين
جامعة الفرات الأوسط التقنية / الكلية التقنية الهندسية / نجف

Lecture 6

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.
- To make comparison between **IF , CASE ,** and **WHEN** statements.
- To indicate the main reasons behind **bad clocking**.

Contents of this Lecture

- Sequential VHDL code inside **Processes\ Loop**
- Sequential VHDL code **IF** vs. **CASE**
- Sequential VHDL code **CASE** vs. **WHEN**
- Bad Clocking

Sequential VHDL code inside Processes\ Loop

- As the name says, **LOOP** is useful when a piece of code must be instantiated several times.

- **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 VHDL code inside Processes\ Loop

- 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];
```

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

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

Sequential VHDL code inside Processes\ Loop

- **Example**

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

Sequential VHDL code inside Processes\ Loop

- **EXIT**: Used for ending.

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

- **NEXT**: Used for skipping loop steps.

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

Sequential VHDL code inside Processes\ Loop

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;
```

Sequential VHDL code inside Processes\ Loop

Example 6.8: Carry Ripple Adder

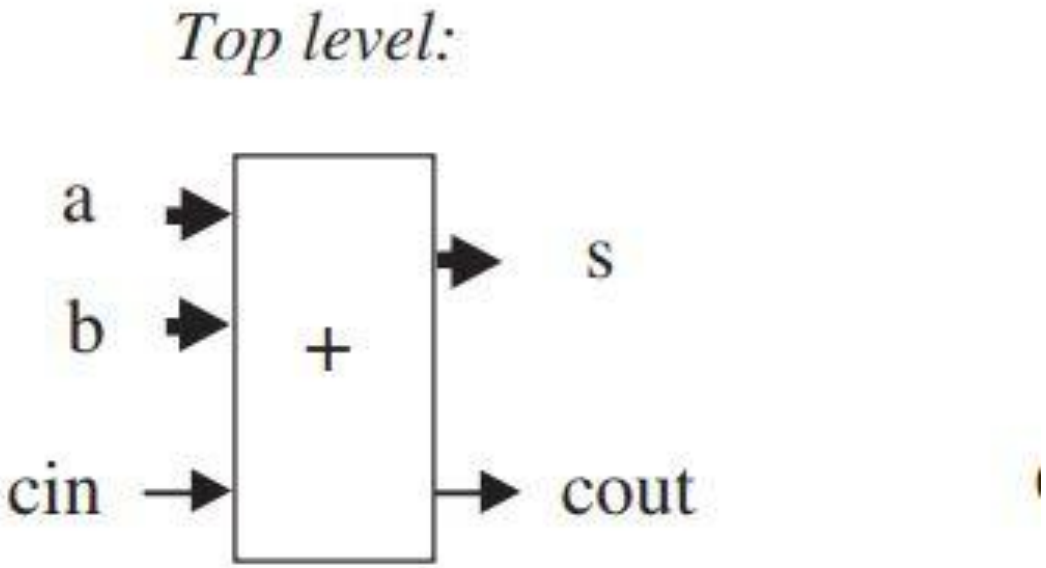
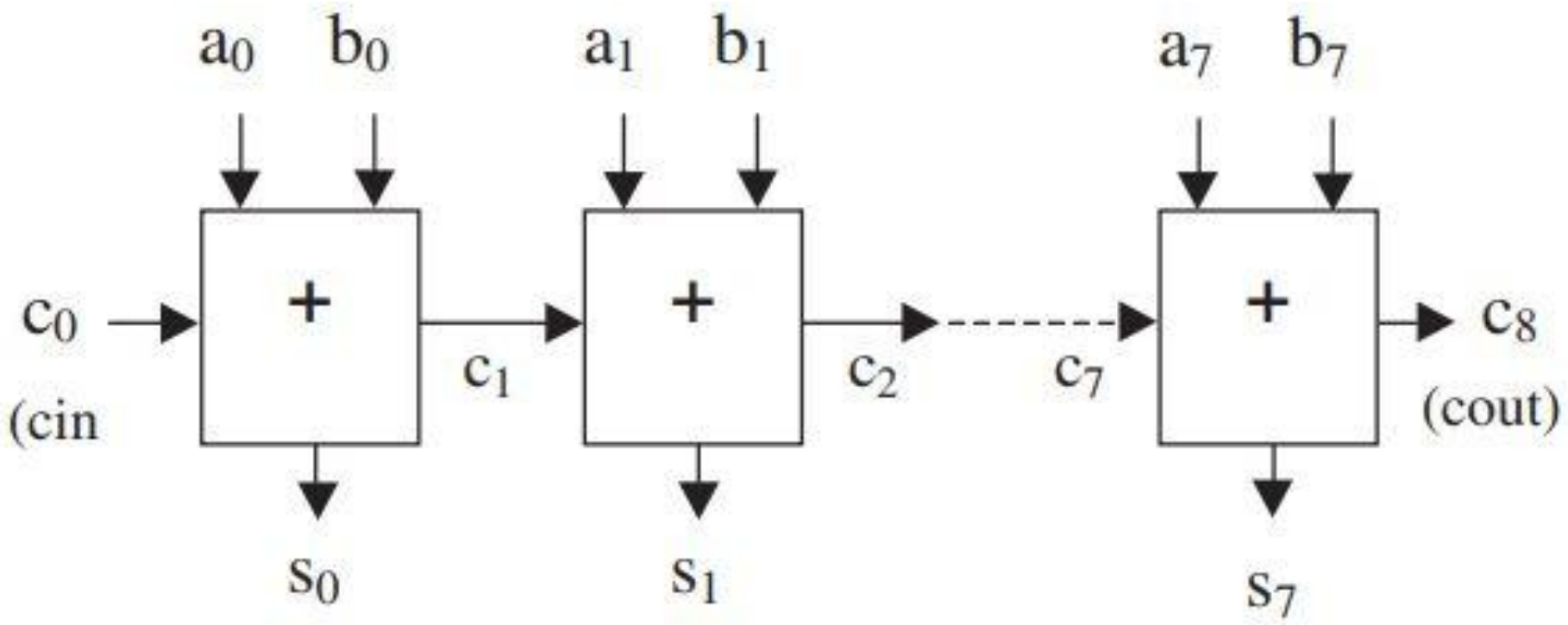


Figure 6.9
8-bit carry ripple adder of example 6.8

a b cin	s cout
0 0 0	0 0
0 1 0	1 0
1 0 0	1 0
1 1 0	0 1
0 0 1	1 0
0 1 1	0 1
1 0 1	0 1
1 1 1	1 1

Sequential VHDL code inside Processes\ Loop

Example 6.8: Carry Ripple Adder



Sequential VHDL code inside Processes\ Loop

Each section of the latter diagram is a full-adder unit (section 1.4). Thus its outputs can be computed by means of:

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

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

```
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 ---------------------------------------------
```

Sequential VHDL code inside Processes\ Loop

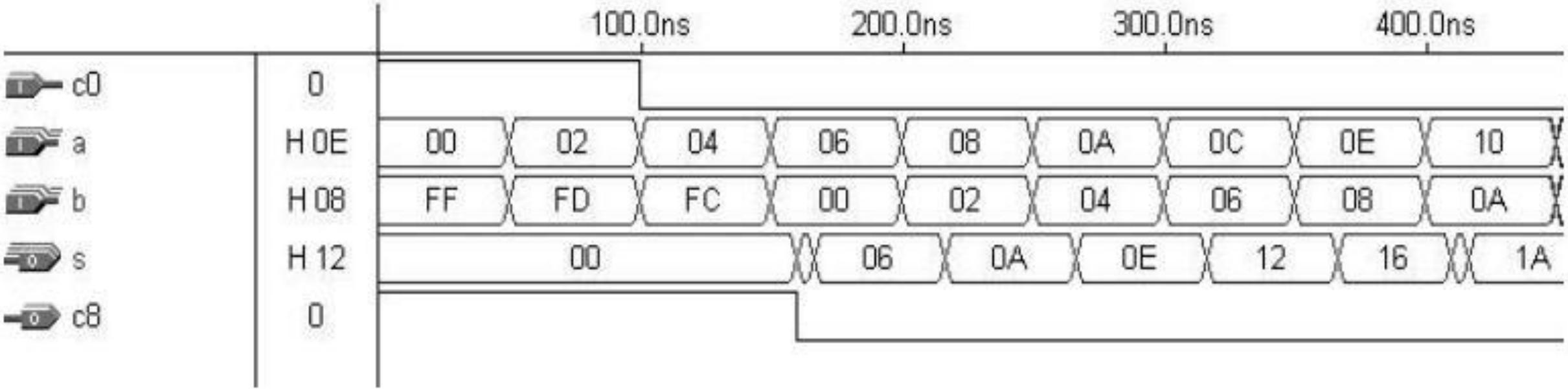


Figure 6.10
Simulation results of example 6.8.

Example 6.8: Carry Ripple Adder

Sequential VHDL code inside Processes\ Loop

Example 6.9: Simple Barrel Shifter

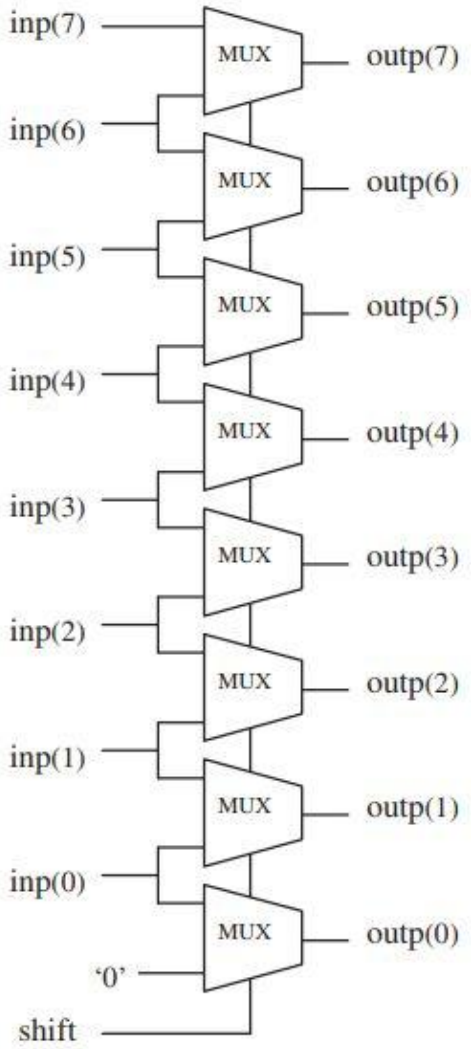


Figure 6.11
Simple barrel shifter of example 6.9.

Sequential VHDL code inside Processes\ Loop

Example 6.9: Simple Barrel Shifter

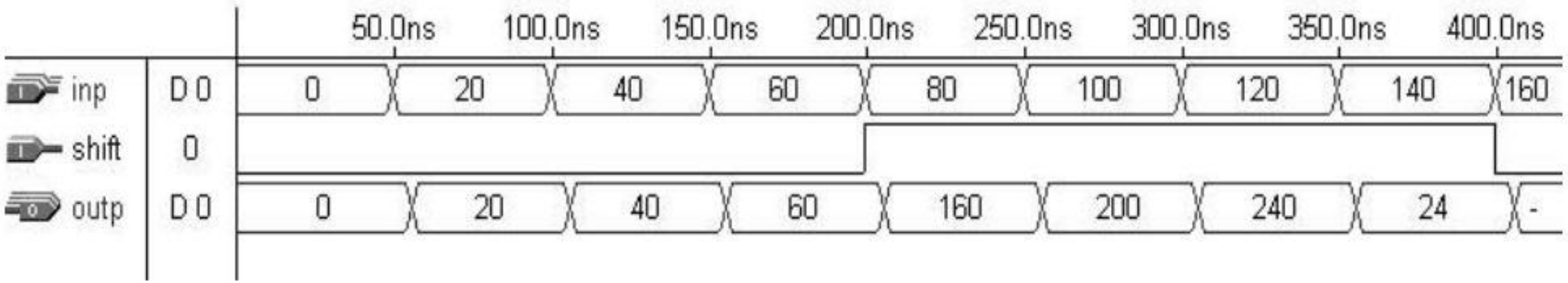


Figure 6.12
Simulation results of example 6.9.

Sequential VHDL code inside Processes\ Loop

```
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 ----------------------------------------
```

Sequential VHDL code inside Processes\ 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 6.10: Leading Zeros

Sequential VHDL code inside Processes\ Loop

Example 6.10: Leading Zeros

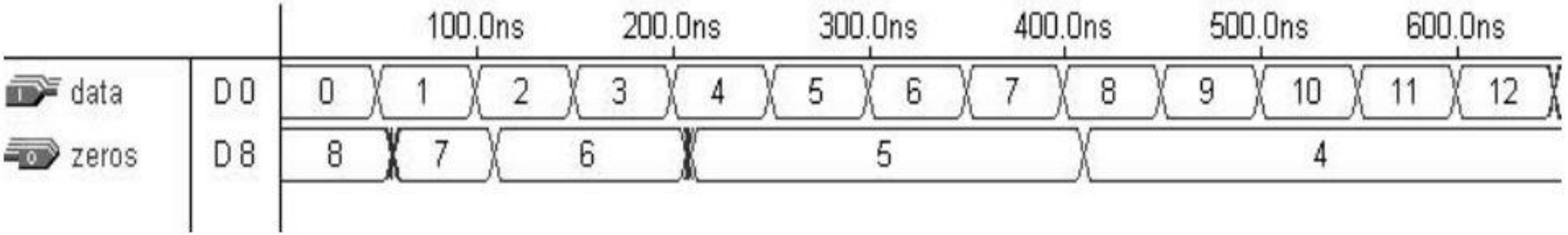


Figure 6.13
Simulation results of example 6.10.

Sequential VHDL code IF vs. CASE

Example: The codes below implement the same physical multiplexer circuit.

```
---- With IF: --------------
IF (sel="00") THEN x<=a;
ELSIF (sel="01") THEN x<=b;
ELSIF (sel="10") THEN x<=c;
ELSE x<=d;

---- With CASE: ------------
CASE sel IS
   WHEN "00" => x<=a;
   WHEN "01" => x<=b;
   WHEN "10" => x<=c;
   WHEN OTHERS => x<=d;
END CASE;
---------------------------
```

Sequential VHDL code WHEN vs. CASE

Example: From a functional point of view, the two codes below are equivalent.

```
---- With WHEN: ----------------
WITH sel SELECT
    x <=    a WHEN "000",
            b WHEN "001",
            c WHEN "010",
            UNAFFECTED WHEN OTHERS;

---- With CASE: ----------------
CASE sel IS
    WHEN "000" => x<=a;
    WHEN "001" => x<=b;
    WHEN "010" => x<=c;
    WHEN OTHERS => NULL;
END CASE;
-------------------------------
```

Sequential VHDL code WHEN vs. CASE

Table 6.1
Comparison between WHEN and CASE.

	WHEN	CASE
Statement type	Concurrent	Sequential
Usage	Only outside PROCESSES, FUNCTIONS, or PROCEDURES	Only inside PROCESSES, FUNCTIONS, or PROCEDURES
All permutations must be tested	Yes for WITH/SELECT/WHEN	Yes
Max. # of assignments per test	1	Any
No-action keyword	UNAFFECTED	NULL

Bad Clocking

```
PROCESS (clk)
BEGIN
    IF(clk'EVENT AND clk='1') THEN
        counter <= counter + 1;
    ELSIF(clk'EVENT AND clk='0') THEN
        counter <= counter + 1;
    END IF;
    ...
END PROCESS;
```

the compiler might display a message of the type **"signal does not hold value after clock edge"** or similar

Bad Clocking

- In this case, besides the messages already described, the compiler might also complain that the signal counter is multiply driven. In any case, compilation will be suspended.

Bad Clocking

```
PROCESS (clk)
BEGIN
    IF(clk'EVENT) THEN
        counter := counter + 1;
    END IF;
    ...
END PROCESS;
```

In this case the compiler assumes a default test value (say “AND clk='1'”) or issue a message of the type “clock not locally stable”.

Bad Clocking

- Finally, if a signal appears in the sensitivity list, but does not appear in any of the assignments that compose the PROCESS, then it is likely that the compiler will simply ignore it.

```
PROCESS (clk)
BEGIN
    counter := counter + 1;
    ...
END PROCESS;
```

However, a message of the type **"ignored unnecessary pin clk"** might be issued instead.

Assignments

- The assignments will be attached to your class room.

End of lecture 6

Any Questions ?