



Integrated Circuits Design by FPGA

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

Lecture 4

VHDL Parallel Code

Lecture 4

Parallel \equiv Concurrent

Objectives of this Lecture

- To understand what is the Parallel (Concurrent) VHDL code.
- 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 Parallel VHDL code.
- Parallel VHDL code using Operators .
- Parallel VHDL code using **WHEN** .
- Parallel VHDL code using **GENERATE**.

Introduction about Parallel VHDL code

- VHDL code can be concurrent (parallel) or sequential.
- The concurrent statements in VHDL are **WHEN** and **GENERATE** . Besides them, assignments using only operators (**AND**, **NOT**, +, *, etc.) can also be used to construct concurrent code .
- VHDL code is inherently concurrent (parallel). Only statements placed inside a **PROCESS**, **FUNCTION**, or **PROCEDURE** are sequential. Still, though within these blocks the execution is sequential, the block, as a whole, is concurrent with any other (external) statements .
- concurrent code only can be used outside **PROCESSES**, **FUNCTIONS**, or **PROCEDURE**.

Parallel VHDL code using Operators

Operators

```
graph LR; A[Operators] --> B[Logical: AND, OR, NOT, XOR, XNOR, ...]; A --> C[Mathematical: +, -, *, /];
```

Logical: AND,
OR, NOT, XOR,
XNOR, ...

Mathematical: +
, -, *, /

Parallel VHDL code using Operators

Example 5.1

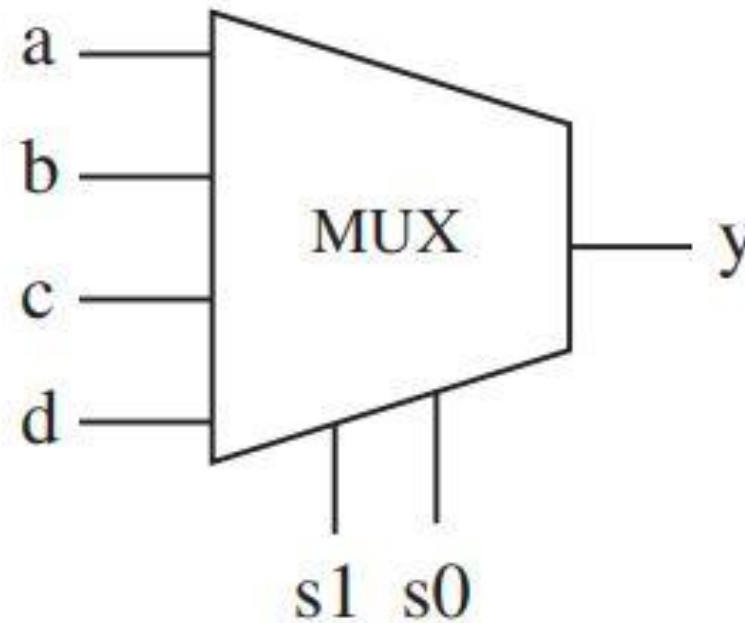


Figure 5.3
Multiplexer of example 5.1.

Parallel VHDL code using Operators

```
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  -----
5  ENTITY mux IS
6      PORT ( a, b, c, d, s0, s1: IN STD_LOGIC;
7            y: OUT STD_LOGIC);
8  END mux;
9  -----
10 ARCHITECTURE pure_logic OF mux IS
11 BEGIN
12     y <= (a AND NOT s1 AND NOT s0) OR
13          (b AND NOT s1 AND s0) OR
14          (c AND s1 AND NOT s0) OR
15          (d AND s1 AND s0);
16 END pure_logic;
17 -----
```

} parallel
code

Parallel VHDL code using Operators

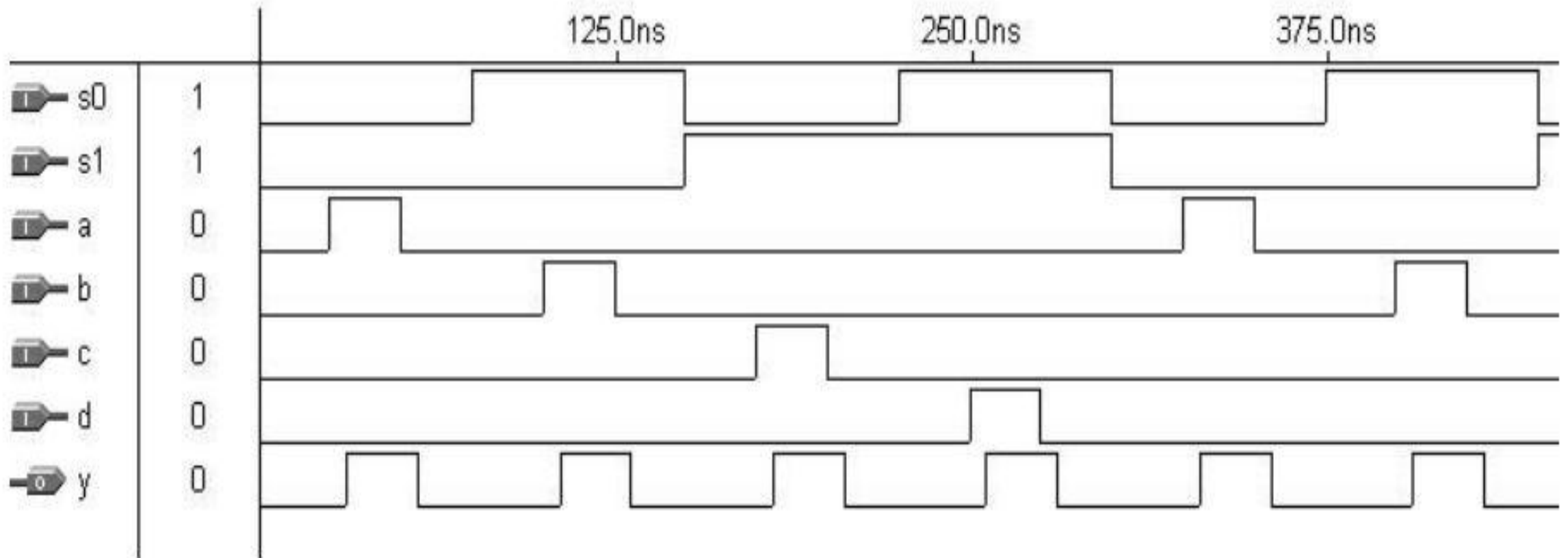


Figure 5.4
Simulation results of example 5.1.

Multiplexer

Parallel VHDL code using WHEN

- As mentioned above, **WHEN** is one of the fundamental concurrent statements (along with operators and **GENERATE**).
- It appears in two forms: **WHEN / ELSE** (simple WHEN) and **WITH/SELECT / WHEN** (selected WHEN).

Parallel VHDL code using WHEN

WHEN / ELSE:

```
assignment WHEN condition ELSE  
assignment WHEN condition ELSE  
...;
```

WITH / SELECT / WHEN:

```
WITH identifier SELECT  
assignment WHEN value,  
assignment WHEN value,  
...;
```

Parallel VHDL code using WHEN

Example 5.2

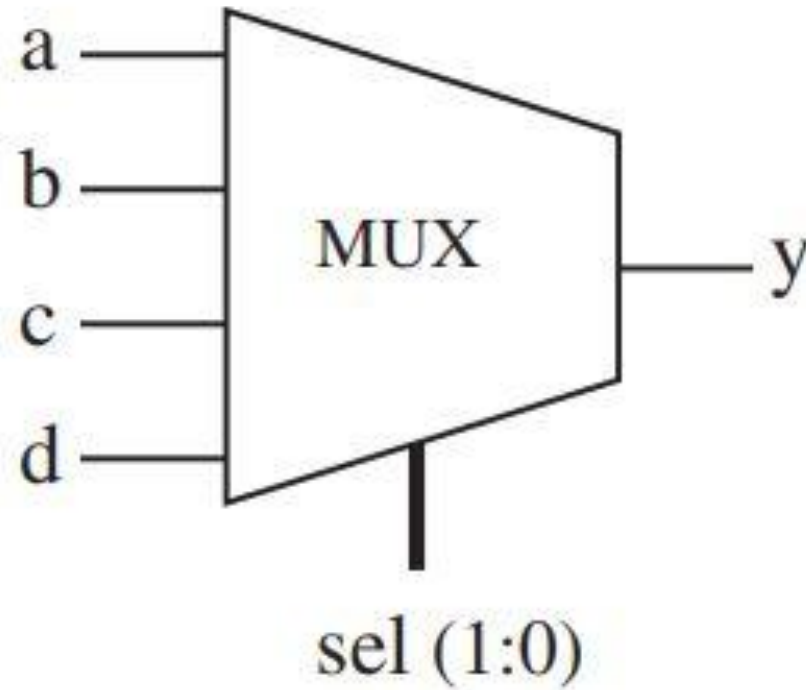



Figure 5.5
Multiplexer of example 5.2.

Parallel VHDL code using WHEN

```
1  ----- Solution 1: with WHEN/ELSE -----
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  -----
5  ENTITY mux IS
6      PORT ( a, b, c, d: IN STD_LOGIC;
7             sel: IN STD_LOGIC_VECTOR (1 DOWNTO 0);
8             y: OUT STD_LOGIC);
9  END mux;
10 -----
11 ARCHITECTURE mux1 OF mux IS
12 BEGIN
13     y <=  a WHEN sel="00" ELSE
14           b WHEN sel="01" ELSE
15           c WHEN sel="10" ELSE
16           d;
17 END mux1;
18 -----
```



Parallel code

Parallel VHDL code using WHEN

```
1  --- Solution 2: with WITH/SELECT/WHEN -----
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  -----
5  ENTITY mux IS
6      PORT ( a, b, c, d: IN STD_LOGIC;
7             sel: IN STD_LOGIC_VECTOR (1 DOWNTO 0);
8             y: OUT STD_LOGIC);
9  END mux;
10 -----
11 ARCHITECTURE mux2 OF mux IS
12 BEGIN
13     WITH sel SELECT
14         y <=  a WHEN "00",      -- notice "," instead of ";"
15              b WHEN "01",
16              c WHEN "10",
17              d WHEN OTHERS;
18 END mux2;
19 -----
```



Parallel code

Parallel VHDL code using WHEN

```
1  -----
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  -----
5  ENTITY mux IS
6      PORT ( a, b, c, d: IN STD_LOGIC;
7              sel: IN INTEGER RANGE 0 TO 3;
8              y: OUT STD_LOGIC);
9  END mux;
10 ----- Solution 1: with WHEN/ELSE -----
11 ARCHITECTURE mux1 OF mux IS
12 BEGIN
13     y <=  a WHEN sel=0 ELSE
14           b WHEN sel=1 ELSE
15           c WHEN sel=2 ELSE
16           d;
17 END mux1;
```


Parallel VHDL code using WHEN

```
18 -- Solution 2: with WITH/SELECT/WHEN -----
19 ARCHITECTURE mux2 OF mux IS
20 BEGIN
21     WITH sel SELECT
22         y <=  a WHEN 0,
23             b WHEN 1,
24             c WHEN 2,
25             d WHEN 3;      -- here, 3 or OTHERS are equivalent,
26 END mux2;                -- for all options are tested anyway
27 -----
```

Parallel VHDL code using WHEN

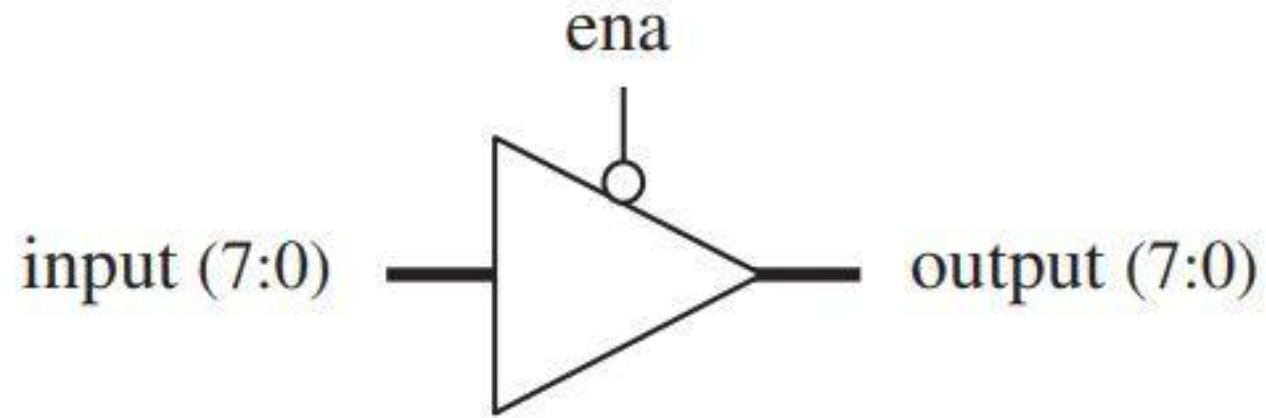


Figure 5.6
Tri-state buffer of example 5.3.

Example 5.3: Tri-state Buffer

Parallel VHDL code using WHEN

output = input when ena (enable) is low, or output = "ZZZZZZZZ" (high impedance) otherwise.

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  -----
4  ENTITY tri_state IS
5      PORT ( ena: IN STD_LOGIC;
6             input: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
7             output: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
8  END tri_state;
9  -----
10 ARCHITECTURE tri_state OF tri_state IS
11 BEGIN
12     output <= input WHEN (ena='0') ELSE
13         (OTHERS => 'Z');
14 END tri_state;
15 -----
```

Parallel VHDL code using WHEN

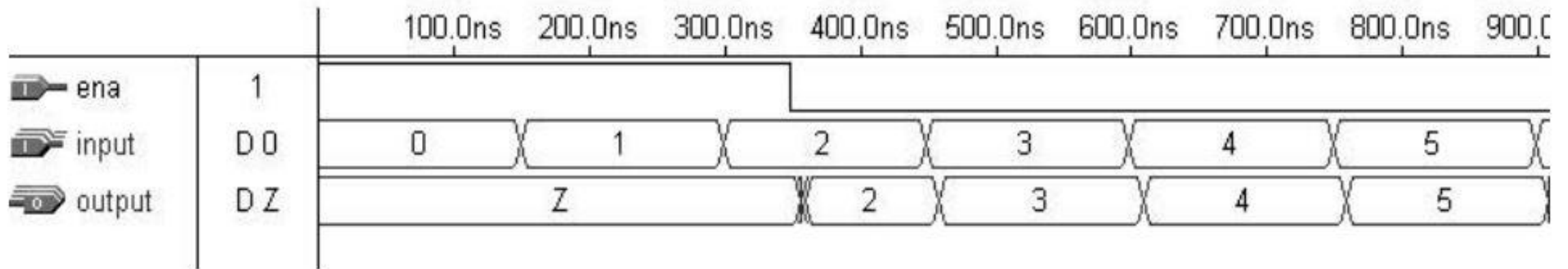


Figure 5.7
Simulation results of example 5.3.

Tri-state Buffer

Parallel VHDL code using WHEN

$n = 8$

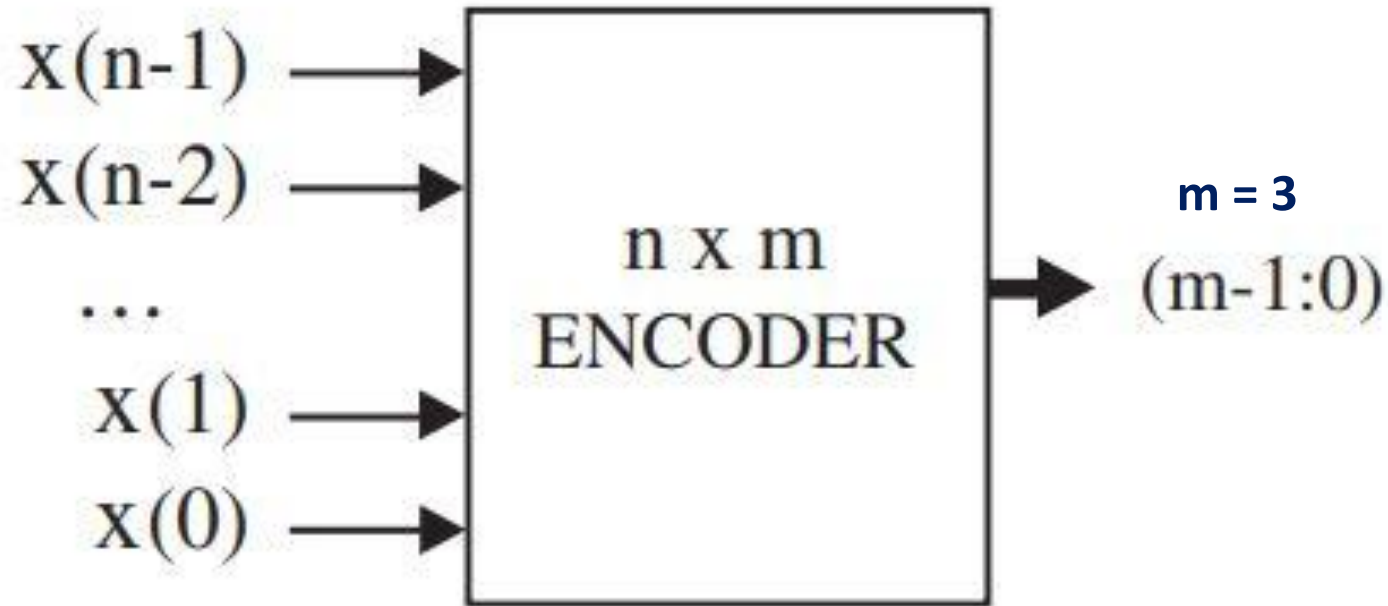


Figure 5.8

Encoder of example 5.4.

Parallel VHDL code using WHEN

```
1  ---- Solution 1: with WHEN/ELSE -----
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  -----
5  ENTITY encoder IS
6      PORT ( x: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
7            y: OUT STD_LOGIC_VECTOR (2 DOWNTO 0));
8  END encoder;
9  -----
10 ARCHITECTURE encoder1 OF encoder IS
11 BEGIN
12     y <=  "000" WHEN x="00000001" ELSE
13           "001" WHEN x="00000010" ELSE
14           "010" WHEN x="00000100" ELSE
15           "011" WHEN x="00001000" ELSE
16           "100" WHEN x="00010000" ELSE
17           "101" WHEN x="00100000" ELSE
18           "110" WHEN x="01000000" ELSE
19           "111" WHEN x="10000000" ELSE
20           "ZZZ";
      END encoder1;
```

Parallel VHDL code using WHEN

```
1  ---- Solution 2: with WITH/SELECT/WHEN -----
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  -----
5  ENTITY encoder IS
6      PORT ( x: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
7            y: OUT STD_LOGIC_VECTOR (2 DOWNTO 0));
8  END encoder;
9  -----
10 ARCHITECTURE encoder2 OF encoder IS
11 BEGIN
12     WITH x SELECT
13         y <=  "000" WHEN "00000001",
14              "001" WHEN "00000010",
15              "010" WHEN "00000100",
16              "011" WHEN "00001000",
17              "100" WHEN "00010000",
18              "101" WHEN "00100000",
19              "110" WHEN "01000000",
20              "111" WHEN "10000000",
21              "ZZZ" WHEN OTHERS;
22 END encoder2;
```

Parallel VHDL code using WHEN

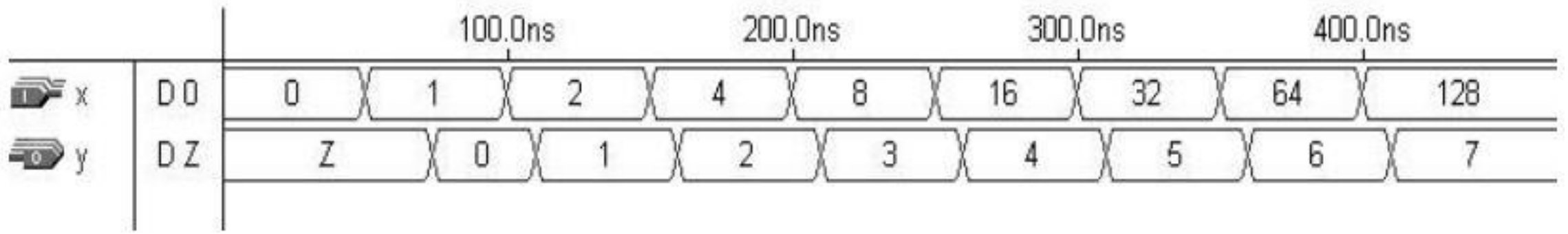


Figure 5.9
Simulation results of example 5.4.

Encoder

Parallel VHDL code using WHEN

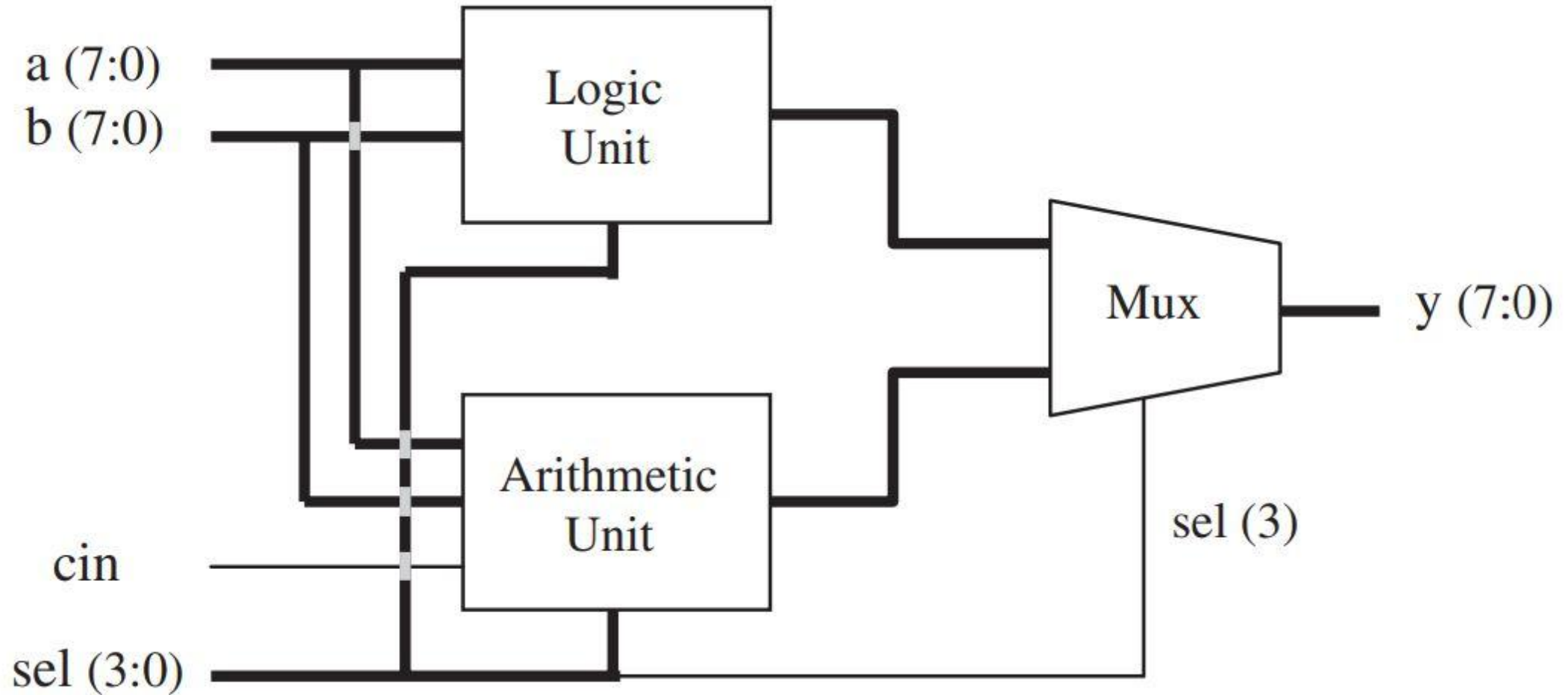


Figure: ALU (Arithmetic Logic Unit) of Example 5.5

Parallel VHDL code using WHEN

sel	Operation	Function	Unit
0000	$y \leq a$	Transfer a	Arithmetic
0001	$y \leq a+1$	Increment a	
0010	$y \leq a-1$	Decrement a	
0011	$y \leq b$	Transfer b	
0100	$y \leq b+1$	Increment b	
0101	$y \leq b-1$	Decrement b	
0110	$y \leq a+b$	Add a and b	
0111	$y \leq a+b+cin$	Add a and b with carry	
1000	$y \leq \text{NOT } a$	Complement a	Logic
1001	$y \leq \text{NOT } b$	Complement b	
1010	$y \leq a \text{ AND } b$	AND	
1011	$y \leq a \text{ OR } b$	OR	
1100	$y \leq a \text{ NAND } b$	NAND	
1101	$y \leq a \text{ NOR } b$	NOR	
1110	$y \leq a \text{ XOR } b$	XOR	
1111	$y \leq a \text{ XNOR } b$	XNOR	

Table: ALU operation of Example 5.5

Parallel VHDL code using WHEN

```
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 USE ieee.std_logic_unsigned.all;
5 -----
6 ENTITY ALU IS
7     PORT (a, b: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
8           sel: IN STD_LOGIC_VECTOR (3 DOWNTO 0);
9           cin: IN STD_LOGIC;
10          y: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
11 END ALU;
12 -----
13 ARCHITECTURE dataflow OF ALU IS
14     SIGNAL arith, logic: STD_LOGIC_VECTOR (7 DOWNTO 0);
15 BEGIN
16     ----- Arithmetic unit: -----
17     WITH sel(2 DOWNTO 0) SELECT
18         arith <=  a WHEN "000",
19                 a+1 WHEN "001",
20                 a-1 WHEN "010",
21                 b WHEN "011",
22                 b+1 WHEN "100",
```

```
23                 b-1 WHEN "101",
24                 a+b WHEN "110",
25                 a+b+cin WHEN OTHERS;
26     ----- Logic unit: -----
27     WITH sel(2 DOWNTO 0) SELECT
28         logic <= NOT a WHEN "000",
29                 NOT b WHEN "001",
30                 a AND b WHEN "010",
31                 a OR b WHEN "011",
32                 a NAND b WHEN "100",
33                 a NOR b WHEN "101",
34                 a XOR b WHEN "110",
35                 NOT (a XOR b) WHEN OTHERS;
36     ----- Mux: -----
37     WITH sel(3) SELECT
38         y <=  arith WHEN '0',
39             logic WHEN OTHERS;
40 END dataflow;
41 -----
```

Parallel VHDL code using WHEN

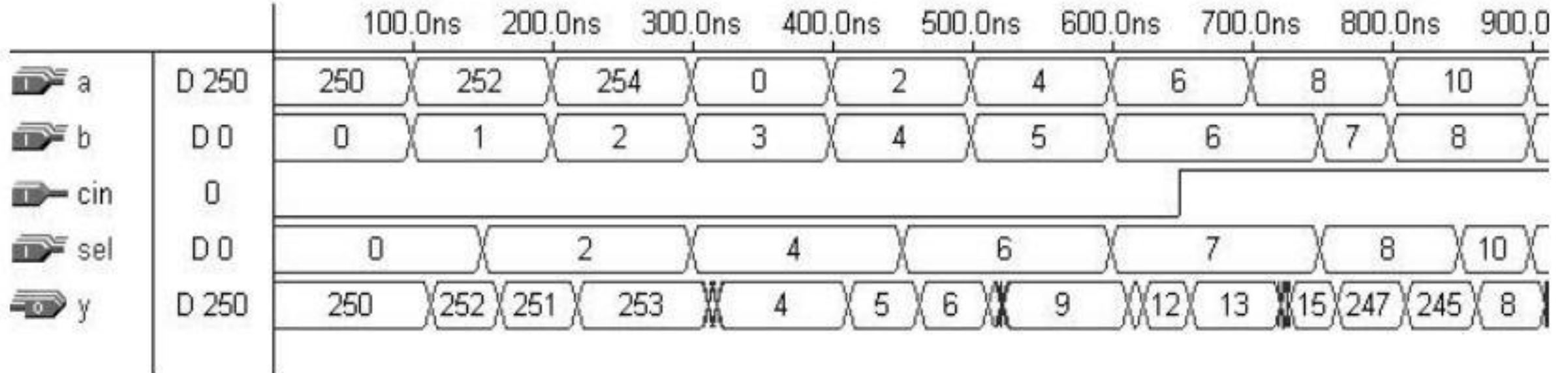


Figure 5.11
Simulation results of example 5.5.

ALU

Parallel VHDL code using GENERATE

- **GENERATE** is another concurrent statement (along with operators and WHEN) .
- It is equivalent to the sequential statement LOOP (chapter 6) in the sense that it allows a section of code to be repeated a number of times, thus creating several instances of the same assignments.
- Its regular form is the **FOR / GENERATE** construct, with the syntax shown below. Notice that GENERATE must be labeled.

FOR / GENERATE:

```
label: FOR identifier IN range GENERATE  
      (concurrent assignments)  
END GENERATE;
```

Parallel VHDL code using GENERATE

- An irregular form is also available, which uses **IF/GENERATE** (with an IF equivalent; recall that originally IF is a sequential statement). Here ELSE is not allowed. In the same way that **IF/GENERATE** can be nested inside **FOR/GENERATE** (syntax below), the opposite can also be done.

IF / GENERATE nested inside FOR / GENERATE:

```
label1: FOR identifier IN range GENERATE
    ...
    label2: IF condition GENERATE
        (concurrent assignments)
    END GENERATE;
    ...
END GENERATE;
```

Parallel VHDL code using GENERATE

Example 5.6: Vector Shifter

(the original vector is underscored):

row(0): 0 0 0 0 1 1 1 1

row(1): 0 0 0 1 1 1 1 0

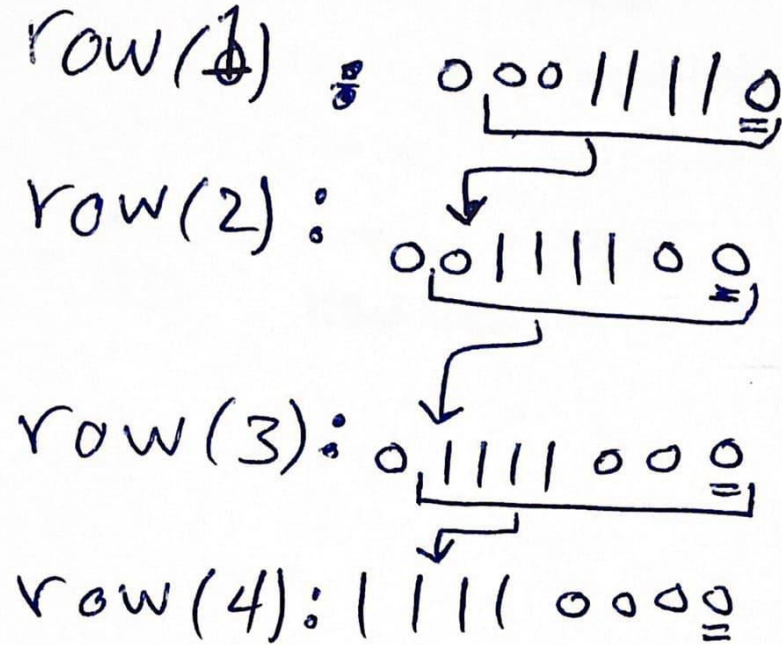
row(2): 0 0 1 1 1 1 0 0

row(3): 0 1 1 1 1 0 0 0

row(4): 1 1 1 1 0 0 0 0

Parallel VHDL code using GENERATE

```
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  -----
5  ENTITY shifter IS
6      PORT ( inp: IN STD_LOGIC_VECTOR (3 DOWNTO 0);
7            sel: IN INTEGER RANGE 0 TO 4;
8            outp: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
9  END shifter;
10 -----
11 ARCHITECTURE shifter OF shifter IS
12     SUBTYPE vector IS STD_LOGIC_VECTOR (7 DOWNTO 0);
13     TYPE matrix IS ARRAY (4 DOWNTO 0) OF vector;
14     SIGNAL row: matrix;
15 BEGIN
16     row(0) <= "0000" & inp;
17     G1: FOR i IN 1 TO 4 GENERATE
18         row(i) <= row(i-1)(6 DOWNTO 0) & '0';
19     END GENERATE;
20     outp <= row(sel);
21 END shifter;
22 -----
```



CS Scanned with CamScanner

Parallel VHDL code using GENERATE

Vector Shifter

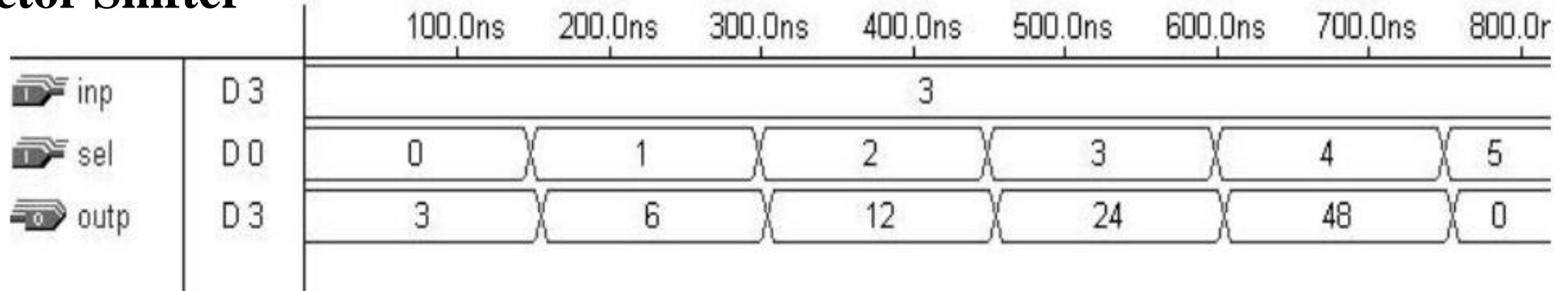


Figure 5.12

Simulation results of example 5.6.

Simulation results are presented in figure 5.12. As can be seen, $inp = "0011"$ (decimal 3) was applied to the circuit. The result was $outp = "00000011"$ (decimal 3) when $sel = 0$ (no shift), $outp = "00000110"$ (decimal 6) when $sel = 1$ (one shift to the left), $outp = "00001100"$ (decimal 12) when $sel = 2$ (two shifts to the left), and so on.

Assignments

- The assignments will be attached in your class room.

End of lecture 4

Any Questions ?