Circuit Design with VHDL

Chapter 10

Instructor: Ali Jahanian

Outline

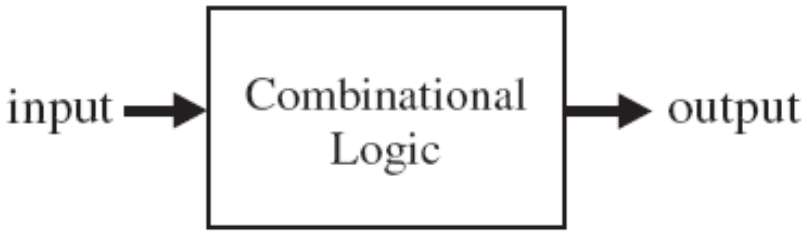
- I-CIRCUIT DESIGN
  - 5 Concurrent Code
    - 5.1 Concurrent versus Sequential
    - 5.2 Using Operators
    - 5.3 WHEN (Simple and Selected)
    - 5.4 GENERATE
    - 5.5 BLOCK
    - 5.6 Problems

VHDL code

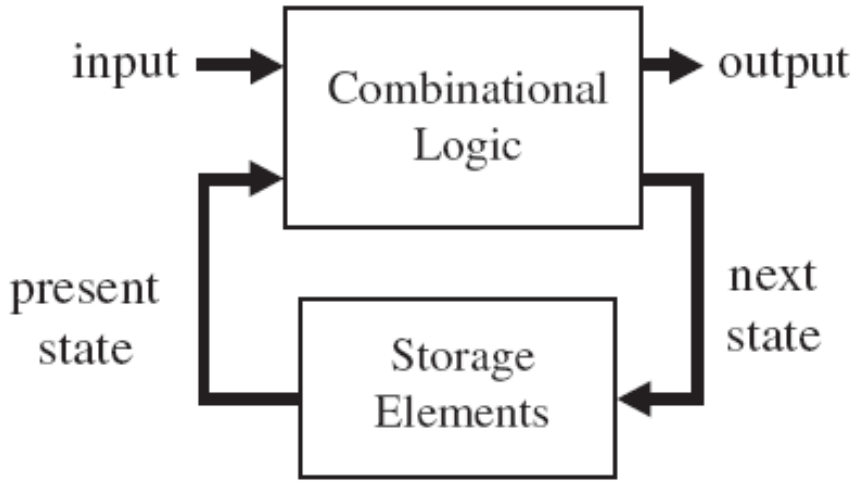
- VHDL code can be:
  - concurrent (parallel)
    - This chapter
  - Sequential
    - Chapter 6

Concurrent versus Sequential

- **Combinational** *versus* **Sequential** *Logic*



(a)



(b)

Figure 5.1
Combinational (a) versus sequential (b) logic.

Concurrent versus Sequential …

- A common **mistake**

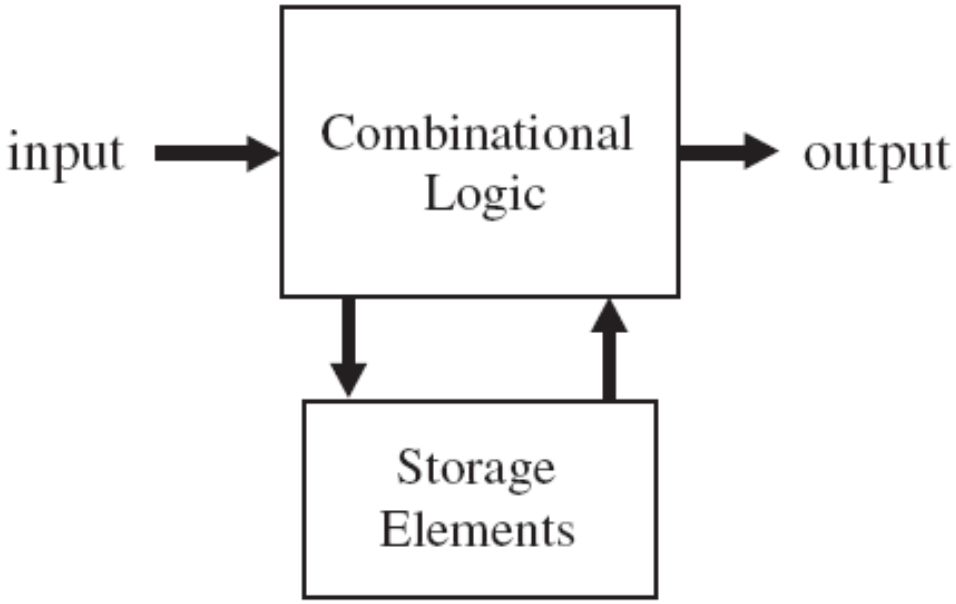


Figure 5.2
RAM model.

Concurrent versus Sequential …

- **Combinational** *versus* **Sequential** *Code*
  - VHDL code is inherently concurrent (parallel)
- Only statements placed inside a
  - PROCESS,
  - FUNCTION,
  - PROCEDURE

  are sequential.

These are concurrent with any other (external) statements.

- Concurrent code is also called dataflow code.

Concurrent versus Sequential …

- **As an example:**

stat1		stat3		stat1		
stat2	≡	stat2	≡	stat3	≡	etc.
stat3		stat1		stat2		

- **In general we can only build,**
  - combinational logic circuits with concurrent code.
  - sequential logic circuits with sequential code.

Concurrent versus Sequential …

- **In summary, in concurrent code the following can be used:**
  - Operators;
  - The WHEN statement ;
  - The GENERATE statement;
  - The BLOCK statement.

Using Operators

- Section 4.1

Table 5.1
Operators.

Operator type	Operators	Data types
Logical	NOT, AND, NAND, OR, NOR, XOR, XNOR	BIT, BIT_VECTOR, STD_LOGIC, STD_LOGIC_VECTOR, STD_ULOGIC, STD_ULOGIC_VECTOR
Arithmetic	+, –, *, /, ** (mod, rem, abs)	INTEGER, SIGNED, UNSIGNED
Comparison	=, /=, <, >, <=, >=	All above
Shift	sll, srl, sla, sra, rol, ror	BIT_VECTOR
Concatenation	&, (,,,)	Same as for logical operators, plus SIGNED and UNSIGNED

Using Operators (Example 5.1: Multiplexer #1)

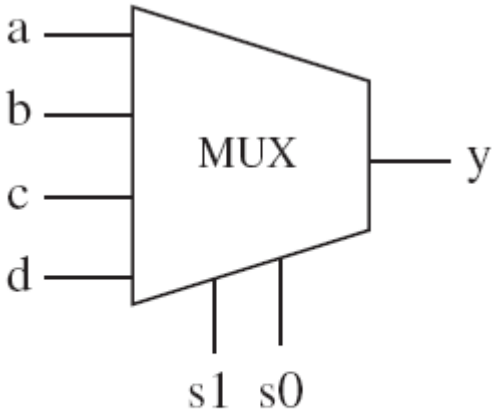


Figure 5.3
Multiplexer of example 5.1.

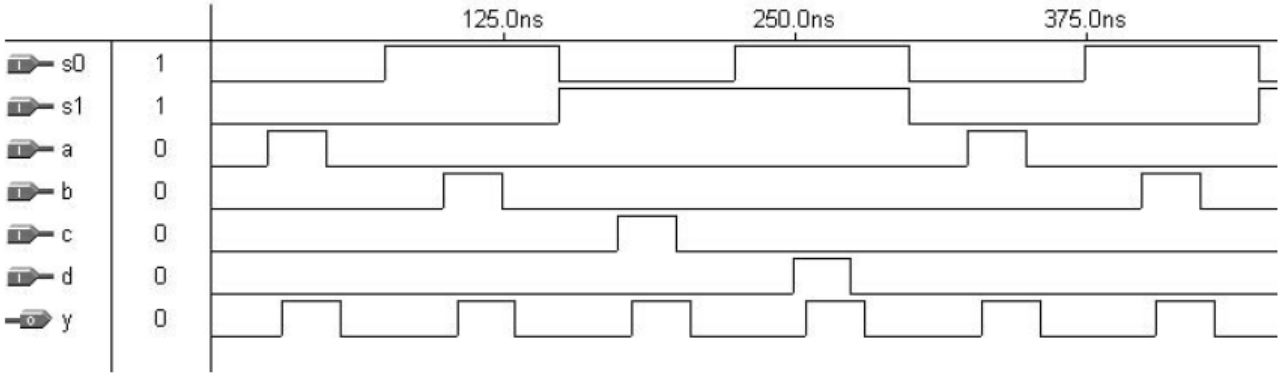


Figure 5.4
Simulation results of example 5.1.

Using Operators (Example 5.1: Multiplexer #1)

```
1  ----------------------------------------
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 ----------------------------------------
```

WHEN (Simple and Selected)

- **WHEN / ELSE:**

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

- **WITH / SELECT / WHEN:**

OTHERS, UNAFFECTED

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

WHEN (Examples)

```
------ With WHEN/ELSE -------------------------
outp <= "000" WHEN (inp='0' OR reset='1') ELSE
        "001" WHEN ctl='1' ELSE
        "010";

---- With WITH/SELECT/WHEN --------------------
WITH control SELECT
   output <= "000" WHEN reset,
             "111" WHEN set,
             UNAFFECTED WHEN OTHERS;
-----------------------------------------------
```

WHEN (Example 5.2: Multiplexer #2)

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

WHEN (Example 5.2: Multiplexer #2) . . .

```
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;    -- cannot be "d WHEN "11" "
18 END mux2;
19 ----------------------------------------
```

WHEN (Example 5.2: Multiplexer #2) . . .

```
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;
```

WHEN (Example 5.2: Multiplexer #2) . . .

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

WHEN (Example 5.3: Tri-state Buffer)

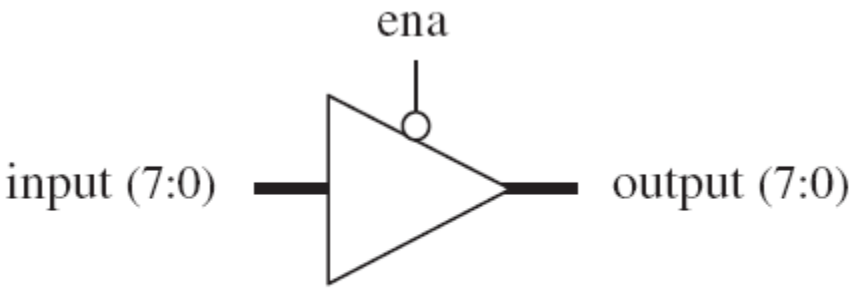


Figure 5.6
Tri-state buffer of example 5.3.

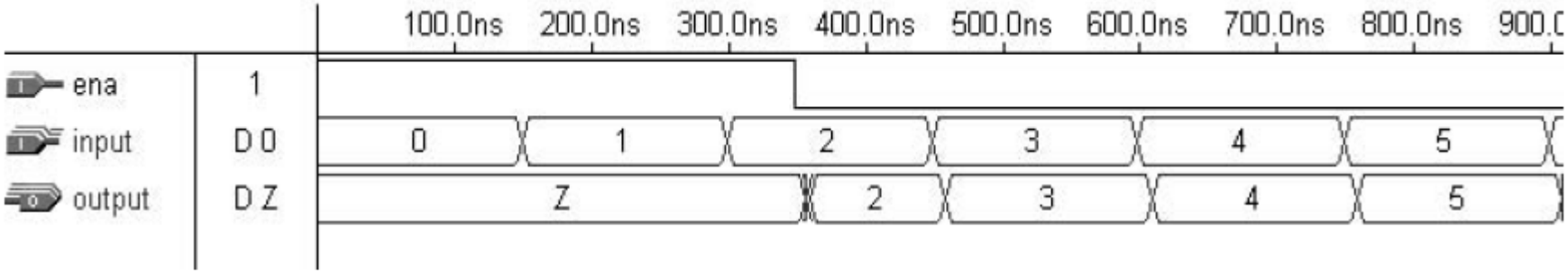


Figure 5.7
Simulation results of example 5.3.

WHEN (Example 5.3: Tri-state Buffer) . . .

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

WHEN (Example 5.4: Encoder)

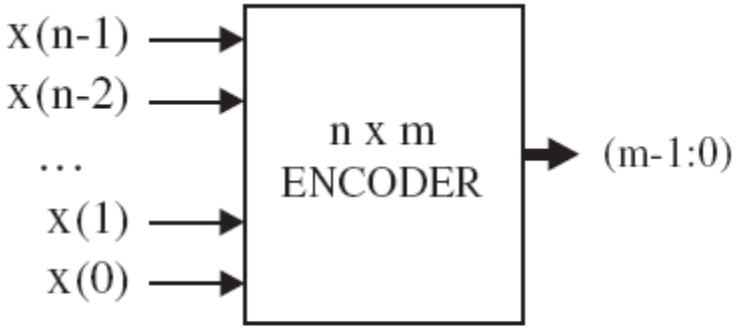


Figure 5.8
Encoder of example 5.4.

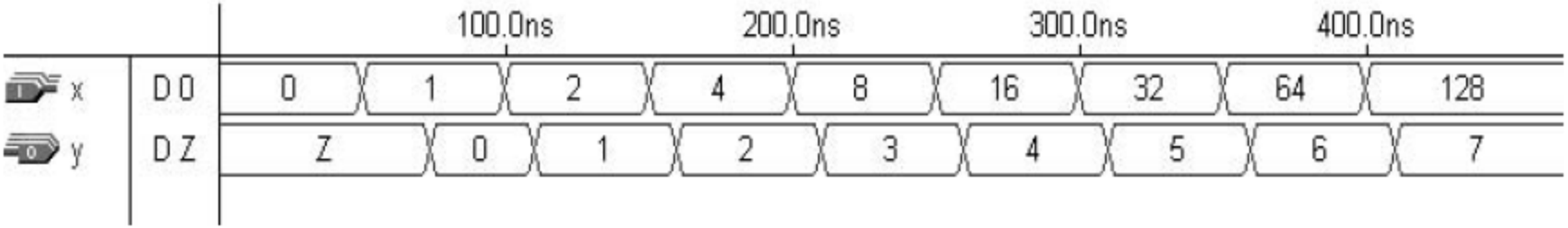


Figure 5.9
Simulation results of example 5.4.

WHEN (Example 5.4: Encoder) . . .

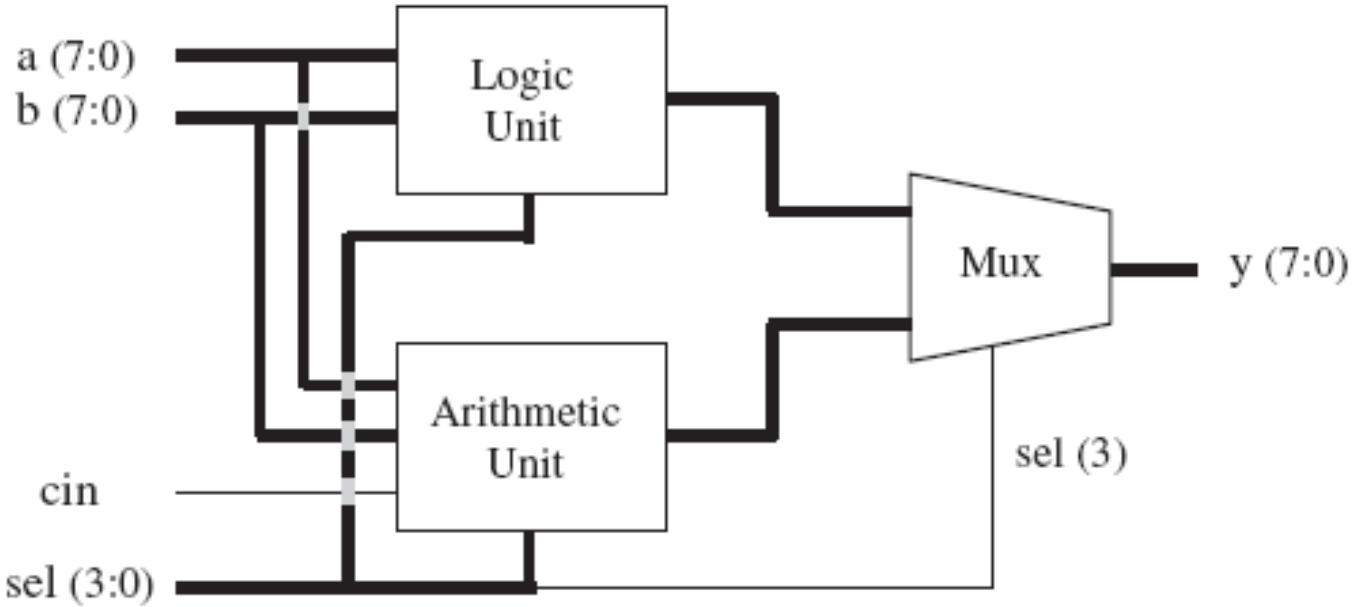
```
1  ---- Solution 1: with WHEN/ELSE -------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ---------------------------------------------
5  ENTITY encoder IS
6     PORT ( x: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
7            y: OUT STD_LOGIC_VECTOR (2 DOWNTO 0));
8  END encoder;
9  ---------------------------------------------
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";
21 END encoder1;
22 ---------------------------------------------
```

WHEN (Example 5.4: Encoder) …

```
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;
23 ----------------------------------------
```

Do you know any solution for bigger Encoder?

WHEN (Example 5.5: ALU)



WHEN (Example 5.5: ALU) …

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

WHEN (Example 5.5: ALU) …

```
1  ----------------------------------------------
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
```

WHEN (Example 5.5: ALU) …

```
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;
```

WHEN (Example 5.5: ALU) …

```
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;
```

WHEN (Example 5.5: ALU) …

```
36     -------- Mux: ---------------
37     WITH sel(3) SELECT
38        y <=  arith WHEN '0',
39              logic WHEN OTHERS;
40 END dataflow;
41 ----------------------------------------
```

GENERATE

- It is equivalent to the sequential statement LOOP
  - FOR / GENERATE:

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

  - IF / GENERATE nested inside FOR / GENERATE:

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

GENERATE (Example)

```
SIGNAL x: BIT_VECTOR (7 DOWNTO 0);
SIGNAL y: BIT_VECTOR (15 DOWNTO 0);
SIGNAL z: BIT_VECTOR (7 DOWNTO 0);
...
G1: FOR i IN x'RANGE GENERATE
    z(i) <= x(i) AND y(i+8);
END GENERATE;
```

```
NotOK: FOR i IN 0 TO choice GENERATE
    (concurrent statements)
END GENERATE;
```

GENERATE (Example 5.6: Vector Shifter)

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

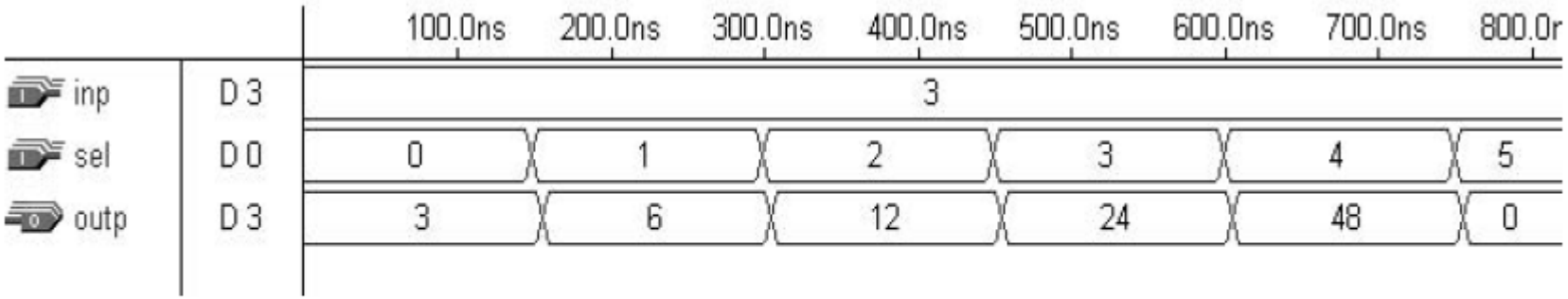


Figure 5.12
Simulation results of example 5.6.

GENERATE (Example 5.6: Vector Shifter) …

```
1  ------------------------------------------------
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 ------------------------------------------------
```

BLOCK

Turning the overall code more readable and more manageable

- Simple BLOCK

```
label: BLOCK
  [declarative part]
BEGIN
  (concurrent statements)
END BLOCK label;
```

- Guarded BLOCK

```
label: BLOCK (guard expression)
  [declarative part]
BEGIN
  (concurrent guarded and unguarded statements)
END BLOCK label;
```

Simple BLOCK

```
------------------------
ARCHITECTURE example ...
BEGIN
    ...
    block1: BLOCK
    BEGIN
        ...
    END BLOCK block1
    ...
    block2: BLOCK
    BEGIN
        ...
    END BLOCK block2;
    ...
END example;
------------------------
```

```
b1: BLOCK
    SIGNAL a: STD_LOGIC;
BEGIN
    a <= input_sig   WHEN ena='1' ELSE 'Z';
END BLOCK b1;
```

BLOCK …

- A BLOCK (simple or guarded) can be nested inside another BLOCK

```
label1: BLOCK
  [declarative part of top block]
BEGIN
  [concurrent statements of top block]
label2: BLOCK
  [declarative part nested block]
BEGIN
  (concurrent statements of nested block)
END BLOCK label2;
  [more concurrent statements of top block]
END BLOCK label1;
```

BLOCK (Example 5.7: Latch Implemented with a Guarded BLOCK)

```
1  ------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ------------------------------
5  ENTITY latch IS
6     PORT (d, clk: IN STD_LOGIC;
7           q: OUT STD_LOGIC);
8  END latch;
9  ------------------------------
10 ARCHITECTURE latch OF latch IS
11 BEGIN
12    b1: BLOCK (clk='1')
13    BEGIN
14       q <= GUARDED d;
15    END BLOCK b1;
16 END latch;
17 ------------------------------
```

BLOCK (Example 5.8: DFF Implemented with a Guarded BLOCK)

```
1  -------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  -------------------------------
5  ENTITY dff IS
6     PORT ( d, clk, rst: IN STD_LOGIC;
7            q: OUT STD_LOGIC);
8  END dff;
9  -------------------------------
10 ARCHITECTURE dff OF dff IS
11 BEGIN
12    b1: BLOCK (clk'EVENT AND clk='1')
13    BEGIN
14       q <= GUARDED '0' WHEN rst='1' ELSE d;
15    END BLOCK b1;
16 END dff;
17 -------------------------------
```

Thanks for your attention

- Don’t forget

Problems!!!

Next session

6 Sequential Code

- 6.1 PROCESS
- 6.2 Signals and Variables
- 6.3 IF
- 6.4 WAIT
- 6.5 CASE
- 6.6 LOOP
- 6.7 CASE versus IF
- 6.8 CASE versus WHEN
- 6.9 Bad Clocking
- 6.10 Using Sequential Code to Design Combinational Circuits