Chapter 5

Introduction to VHDL

5.1 Hardware Description Language (HDL)

- A computer language used to design circuits with text-based descriptions of the circuits.

- **VHSIC**： **V**ery **H**igh **S**peed **I**ntegrated **C**ircuit

- **VHDL** (**V**HSIC **H**ardware **D**escription **L**anguage) is the industry-standard language used for programming PLDs.

VHDL History

- ❑Developed by defense contractors (承包商) as a standard for programming circuits.
- ❑Currently defined by IEEE Standard 1076-1993.
- ❑Related standard for certain data types is IEEE Standard 1164-1993.

VHDL

- ❑Used to describe the structure or behavior of hardware.
- ❑Describes how the hardware should operate (modeling).
- ❑Describes how the hardware should be built (synthesis).
- ❑In VHDL, the designer enters text according to the **syntax** of the language.
- ❑**Syntax:** The rules of construction, or “grammar”, of a programming language.

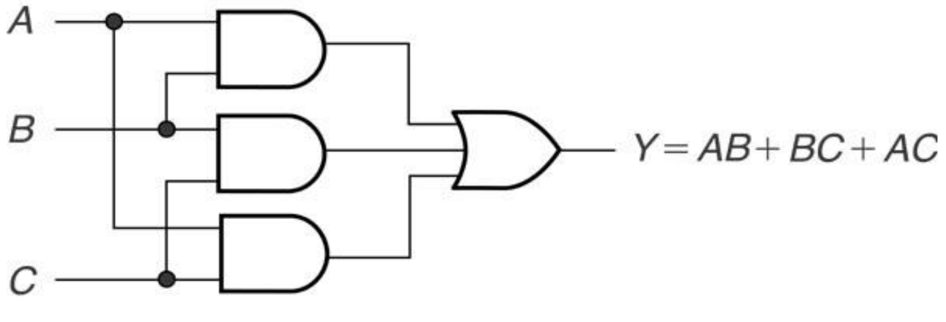
Entity and Architecture

- ❑Two basic constructs required for all VHDL code.
- ❑The **entity（實體） declaration** describes the inputs and outputs.
- ❑The **architecture body** defines the relationships between the inputs and outputs.

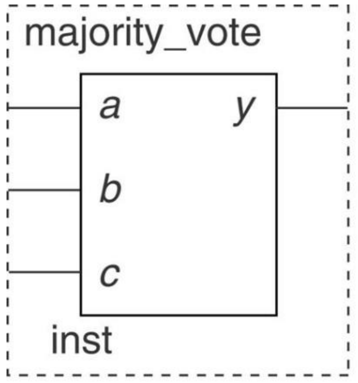
VHDL Entity

- ❑ Defines the external aspects(外觀) of the function.
- ❑ Each input or output is a **port**.
- ❑ The type of port is defined by **mode**.

- ❑ Majority Vote Circuit
  Output is **High** with two or more **High** inputs.
- ❑ Graphic symbol representing the VHDL file in a Quartus II Block Diagram File.



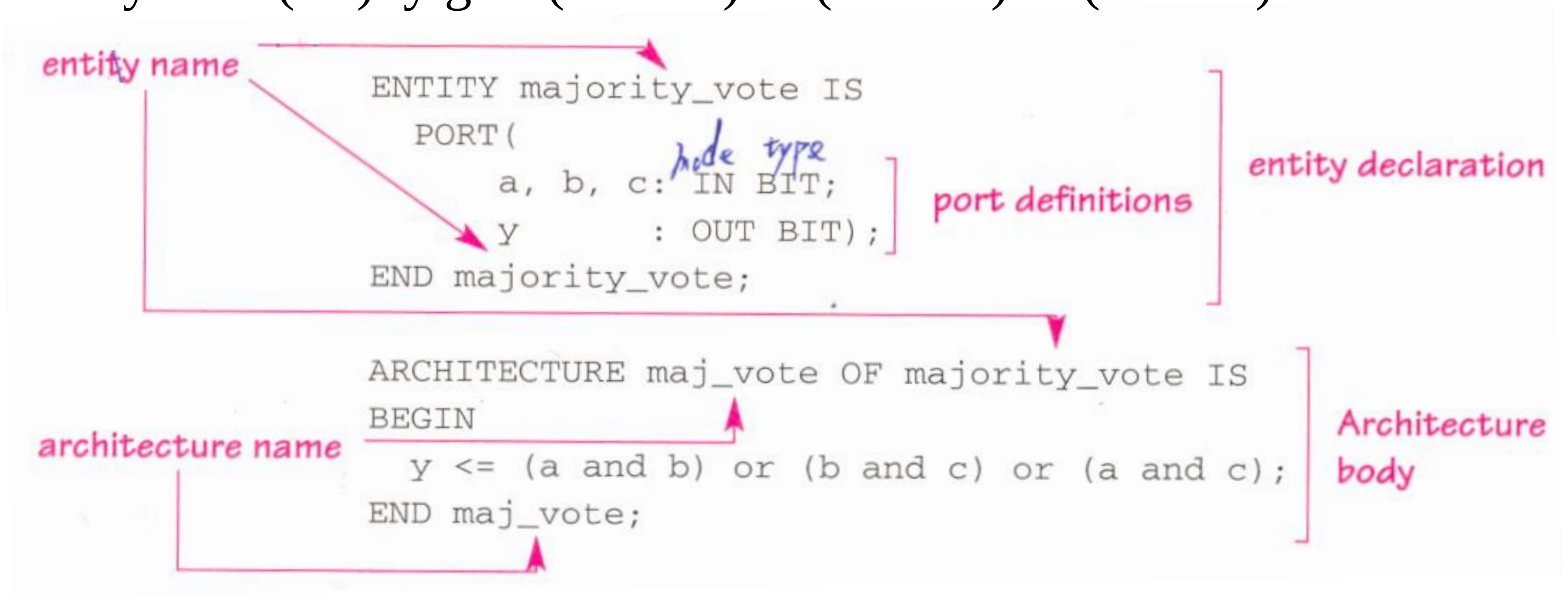
a. Logic circuit



b. Symbol for VHDL majority vote design

VHDL Entity Declaration and Architecture Body

- Mode: data direction
- Type: range of values (BIT: “0” and “1”)
- Symbol (<=): y gets (a and b) or (b and c) or (a and c)



```
ENTITY majority_vote IS
  PORT(
      a, b, c: IN BIT;
      y      : OUT BIT);
END majority_vote;

ARCHITECTURE maj_vote OF majority_vote IS
BEGIN
  y <= (a and b) or (b and c) or (a and c);
END maj_vote;
```

Port Modes and Types

- **IN** refers to a port used only for input.
- **OUT** refers to a port used only for output.
- **BIT** refers to the port type.
- A port designated as type **BIT** can have a value of either ‘0’ or ‘**1**’ .

Boolean Operators in VHDL

- **AND, OR, NOT, NAND, NOR, XOR**, and **XNOR** are represented as written.
- VHDL has no order of precedence for Boolean operators.
- Expressions must be written explicitly with parentheses.

Boolean Operators Example

- $Y = A\overline{B} + \overline{A}B\overline{C}$

- *Y* <= (a and(not b)) or ((not a) and b and (not c));

- $Y = \overline{AB + \overline{A}\,\overline{C} + D}$

- *Y* <=not((a and b) or ((not a) and (not c)) or d);

Example 5.1

Write a VHDL file for the logic circuit in Figure 5.2, assuming all ports are of type BIT.

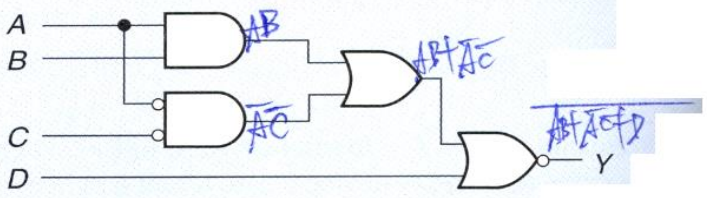


FIGURE 5.2 Example 5.1: Logic Circuit

■ Solution

The Boolean expression for the circuit in Figure 5.2 is $Y = \overline{AB + \bar{A}\bar{C} + D}$. This is described by the following VHDL design entity.

```
ENTITY logic_circuit IS
   PORT(
       a, b, c, d : IN BIT;
       y          : OUT BIT);
END logic_circuit;
```

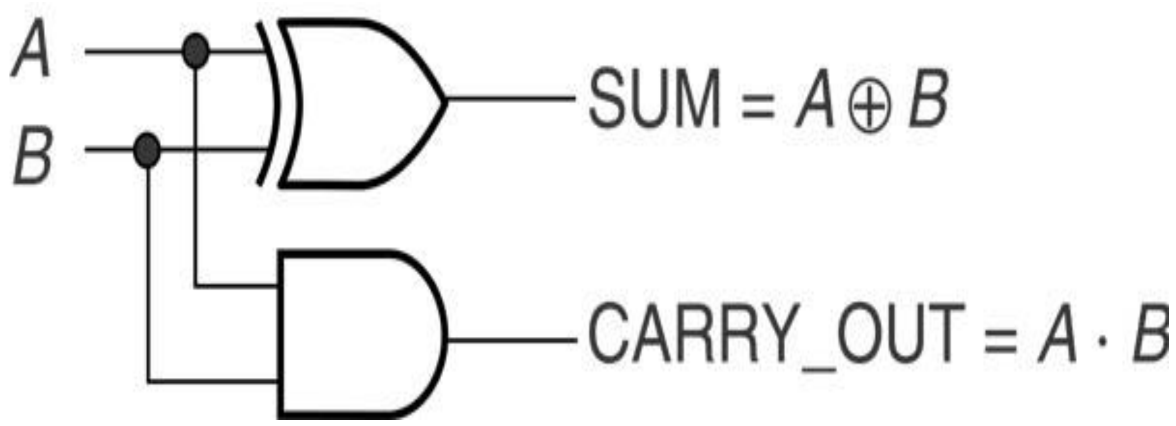
```
ARCHITECTURE cct OF logic_circuit IS
BEGIN
   y <= not((a and b) or ((not a) and (not c)) or d);
END cct;
```

Signal Concurrency(同時發生)

- ❑ Concurrent means simultaneous.
- ❑ The statements in an architecture body are evaluated at the same time, they are not dependent on the order in which they are written.
- ❑ A change in one input common to several circuits affects all the circuits at the same time.

Signal Concurrency

- Half adder
- If A=1, B=1:
  SUM=0, CARRY_OUT=1
- If A=1, B=0:
  SUM=1, CARRY_OUT=0



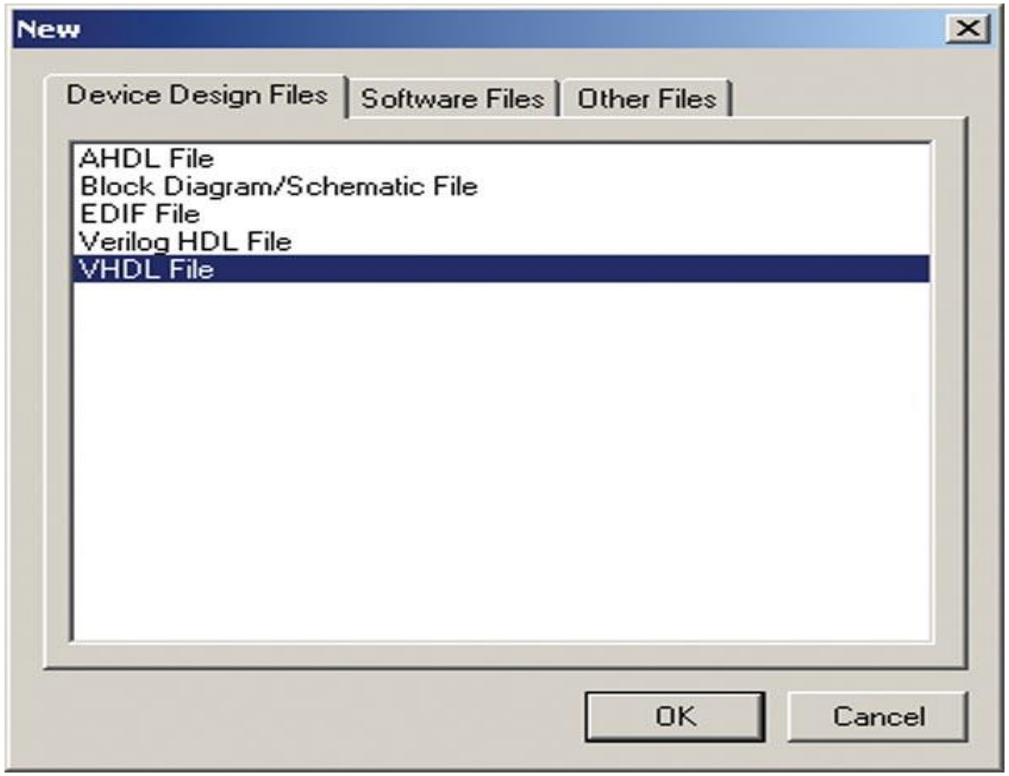
Signal Concurrency Example

```
sum          <=  a xor  b;
carry_out    <=  a and  b;
```

- ❑The order in which the statements are written is not important.(順序不重要)
- ❑Both are executed at the same time.
- ❑This is how the hardware behaves.

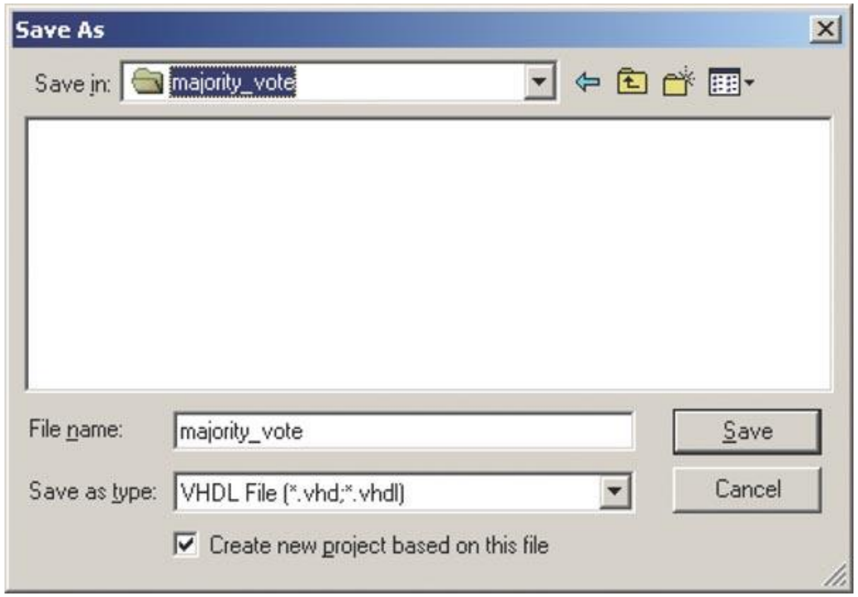
5.2 Making a VHDL File in Quartus II – 1

- Created using the Quartus II Text Editor
- Start a **New File.**
- Select VHDL File from the **Device Design Files** tab.
- The text editor automatically opens.



Making a VHDL File in Quartus II – 2

- ❑ Name the file and save as type VHDL.
- ❑ Check Create new project based on this file.
- ❑ Click Save.
- ❑ Click Yes when asked if you want to create a new project from this file.

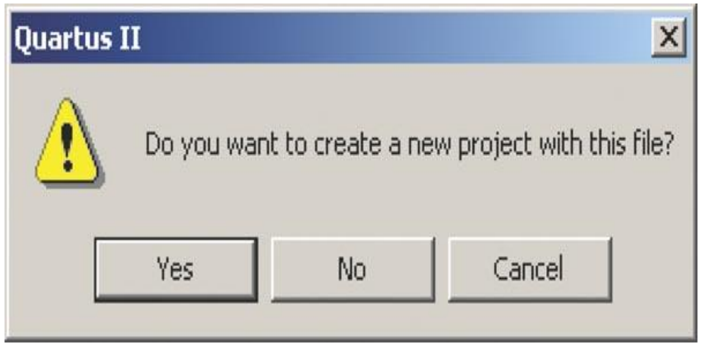

Save As

Save in: majority_vote

File name: majority_vote

Save as type: VHDL File (*.vhd;*.vhdl)

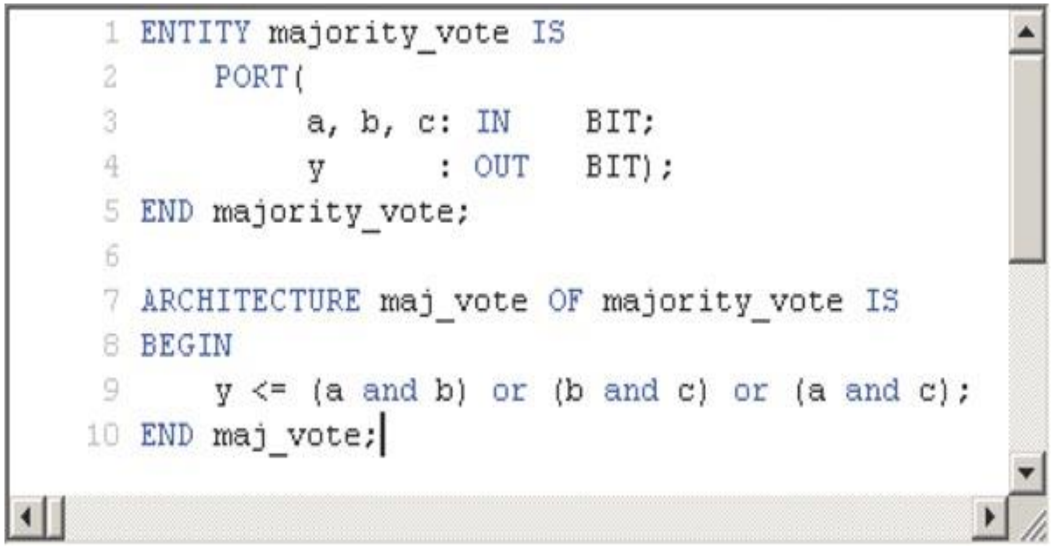
☑ Create new project based on this file


Quartus II

Do you want to create a new project with this file?

Making a VHDL File in Quartus II – 3

- VHDL code is entered in the Text Editor window.
- For reference, the text editor will number each line of code.
- Save and compile your completed VHDL code.

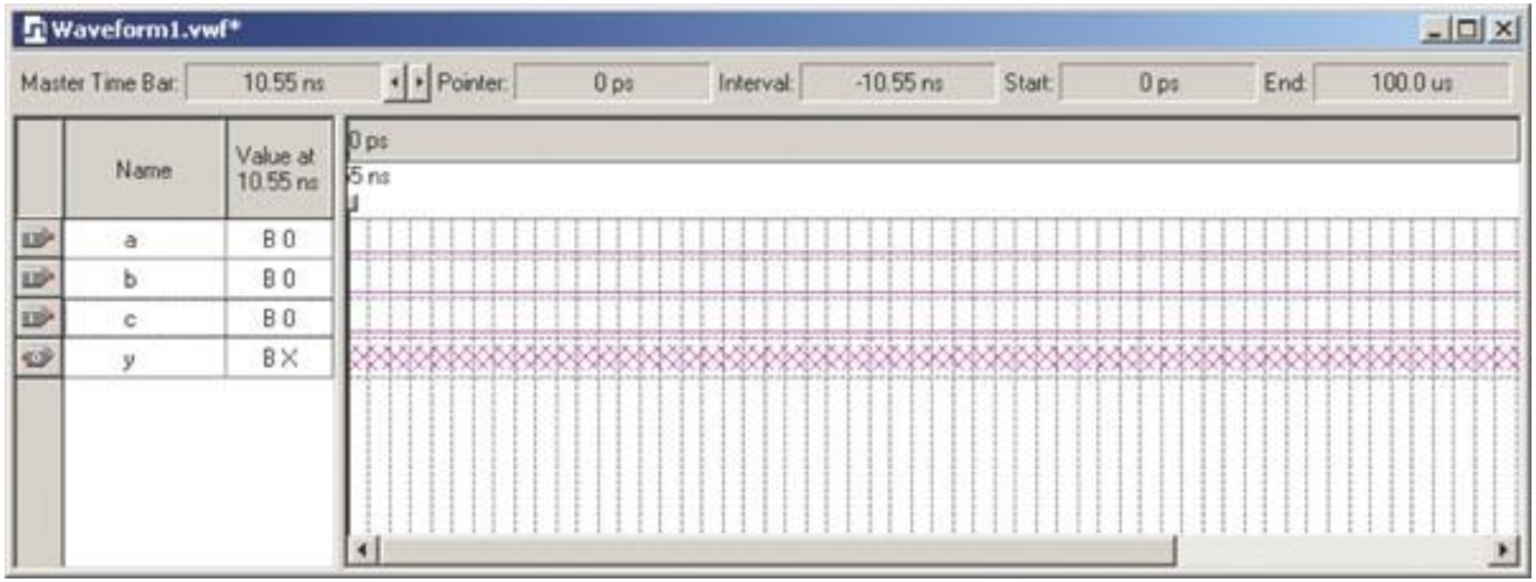


```
ENTITY majority_vote IS
    PORT(
         a, b, c: IN    BIT;
         y      : OUT   BIT);
END majority_vote;

ARCHITECTURE maj_vote OF majority_vote IS
BEGIN
    y <= (a and b) or (b and c) or (a and c);
END maj_vote;
```

Making a VHDL File in Quartus II – 4

- New file → Vector Waveform File
- Node finder → for input a, b, c and output y
- Set End Time → 100 us

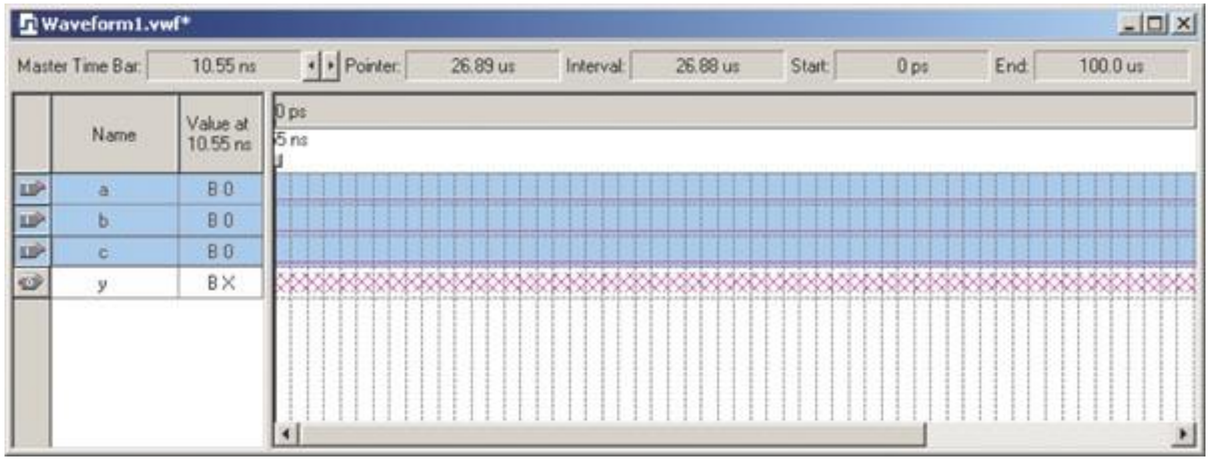

Waveform1.vwf*

Master Time Bar: 10.55 ns Pointer: 0 ps Interval: -10.55 ns Start: 0 ps End: 100.0 us

Name	Value at 10.55 ns
a	B 0
b	B 0
c	B 0
y	B X

Making a VHDL File in Quartus II – 5

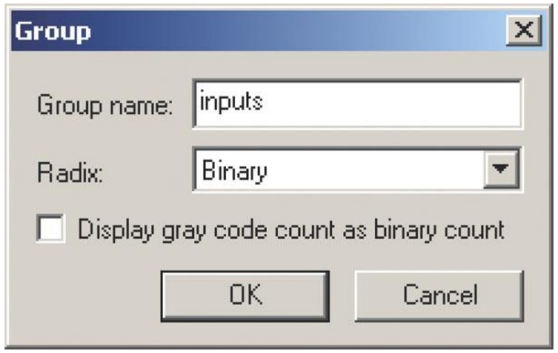
- Group a, b, and c



Waveform1.vwf*

Master Time Bar: 10.55 ns Pointer: 26.89 us Interval: 26.88 us Start: 0 ps End: 100.0 us

Name	Value at 10.55 ns
a	B 0
b	B 0
c	B 0
y	B X

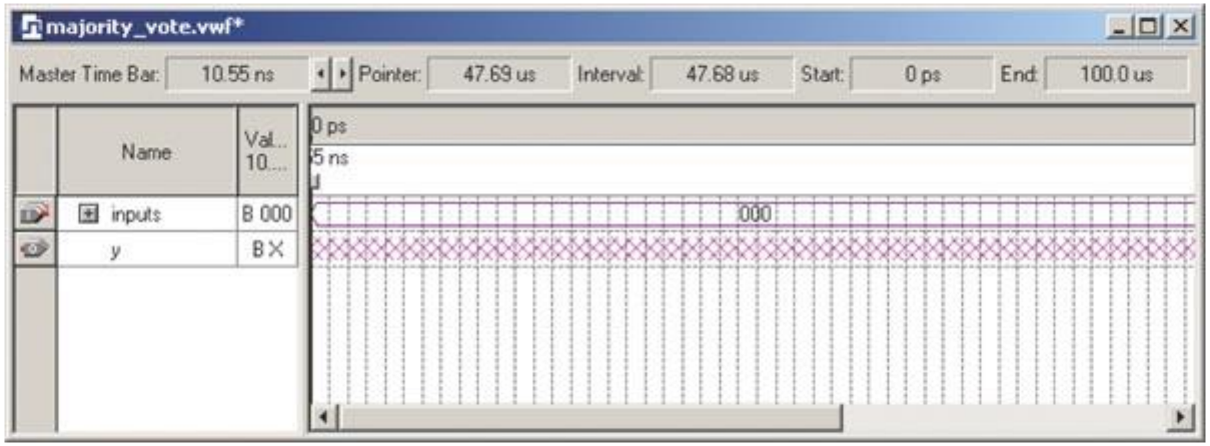


Group

Group name: inputs

Radix: Binary

Display gray code count as binary count [ ]



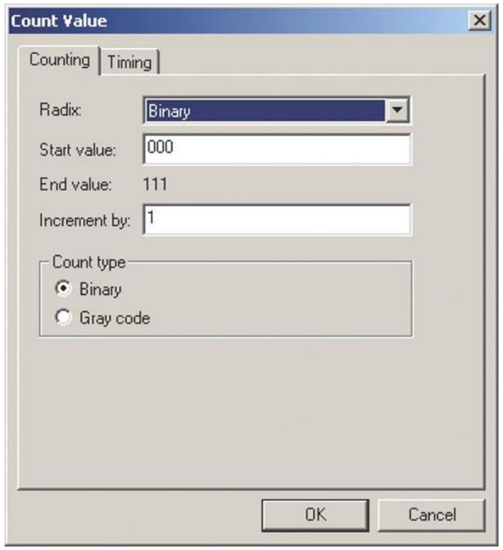
majority_vote.vwf*

Master Time Bar: 10.55 ns Pointer: 47.69 us Interval: 47.68 us Start: 0 ps End: 100.0 us

Name	Val... 10....
inputs	B 000
y	B X

Making a VHDL File in Quartus II – 6

- Set Count Value
- Set Timing
- Fit Window



Count Value

Counting | Timing

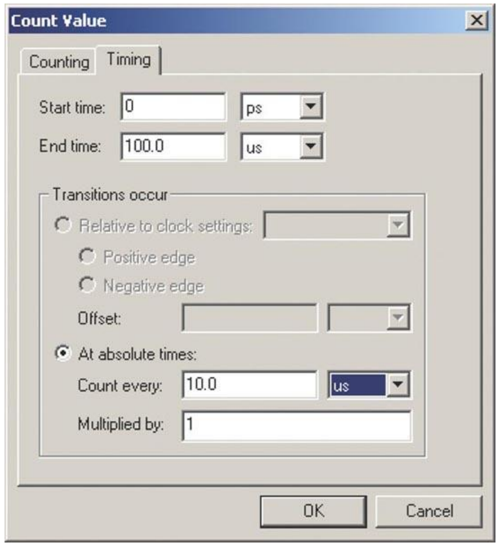
Radix: Binary

Start value: 000

End value: 111

Increment by: 1

Count type
- [✓ Binary]
- Gray code



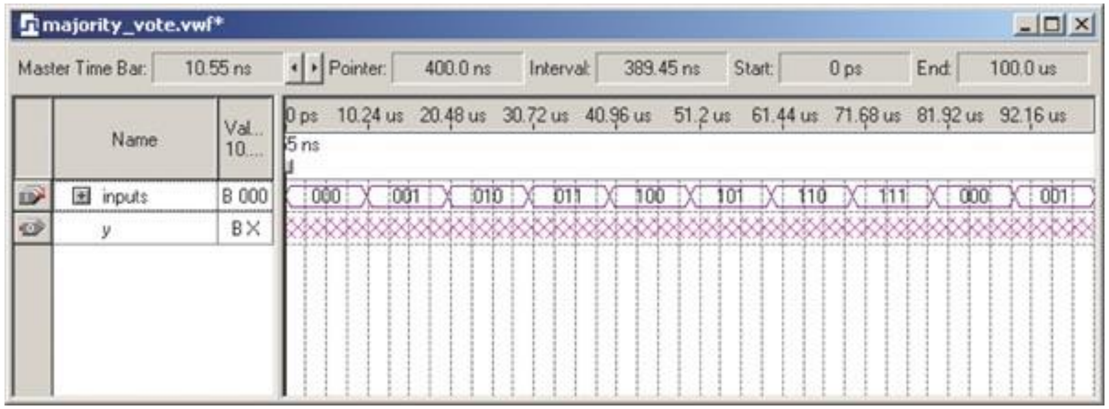
Count Value

Counting | Timing

Start time: 0 ps

End time: 100.0 us

Transitions occur
- Relative to clock settings:
  - Positive edge
  - Negative edge
  - Offset:
- [✓ At absolute times:]
  - Count every: 10.0 us
  - Multiplied by: 1



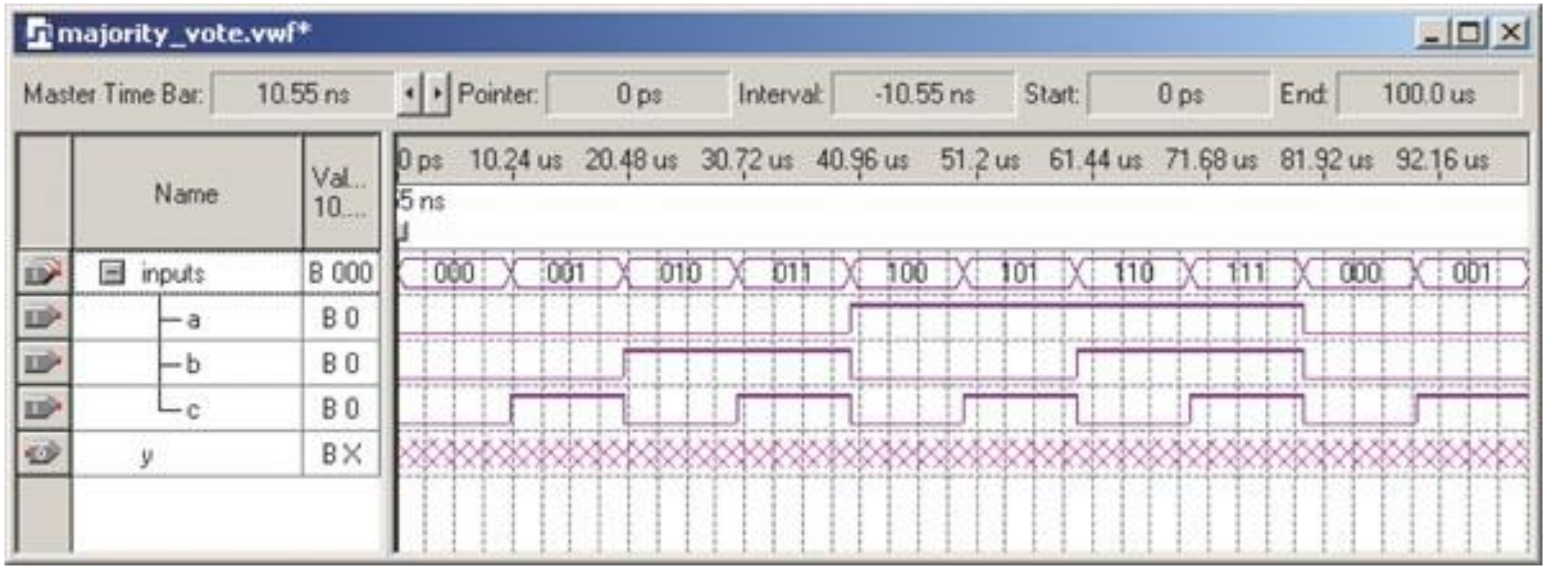
majority_vote.vwf*

Master Time Bar: 10.55 ns | Pointer: 400.0 ns | Interval: 389.45 ns | Start: 0 ps | End: 100.0 us

Name	Val... 10....	0 ps	10.24 us	20.48 us	30.72 us	40.96 us	51.2 us	61.44 us	71.68 us	81.92 us	92.16 us
inputs	B 000	000	001	010	011	100	101	110	111	000	001
y	B X										

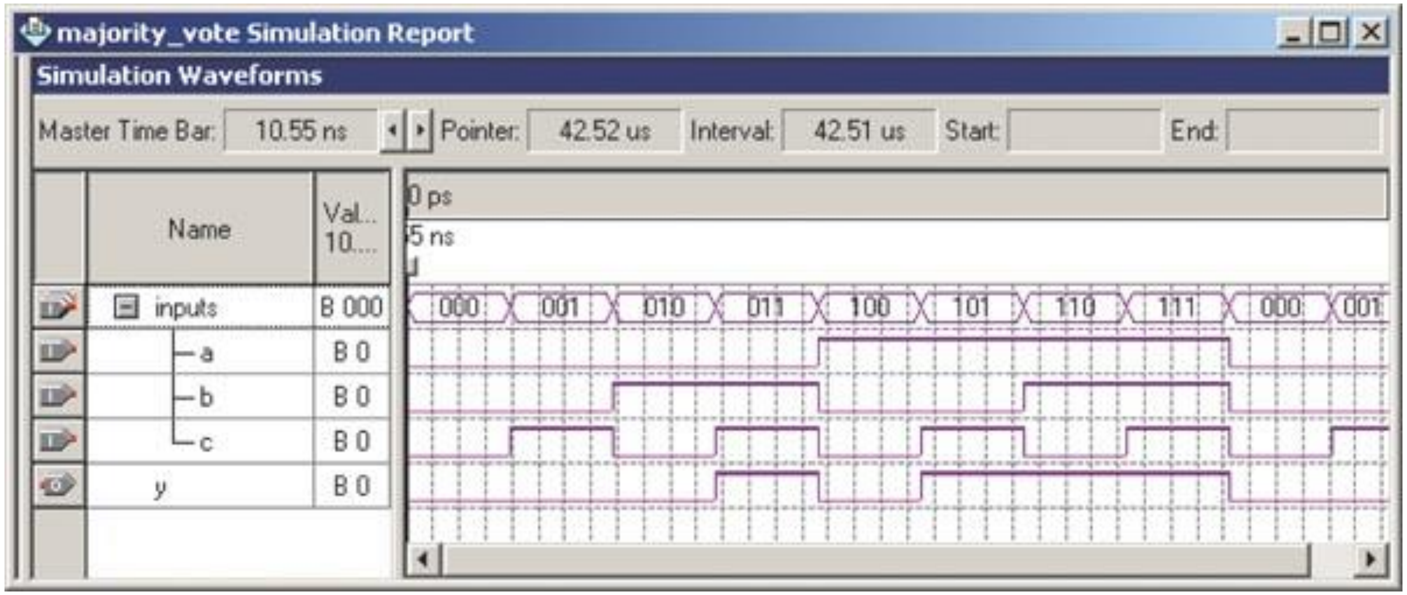
Making a VHDL File in Quartus II – 7

- Group and individual input waveform



Making a VHDL File in Quartus II – 8

- Run simulation



5.3 Valid Names in VHDL – 1

- ❑A valid name in Quartus is called a **name identifier**.
- ❑All ports, signals, variables, entity names, architecture bodies, or similar objects must use names that are recognized by Quartus.

Valid Names in VHDL – 2

- VHDL is **not case sensitive. (不分大小寫)**
- Name identifiers consists of a **letter** followed by any number of letters or numbers.
- A **space** in a name is considered **invalid**.
- VHDL keywords should be capitalized. **(大寫)**
- User names should be written in lowercase.
- An underscore can be written within a name but cannot start or end the name.
- Two consecutive underscores are not permitted.

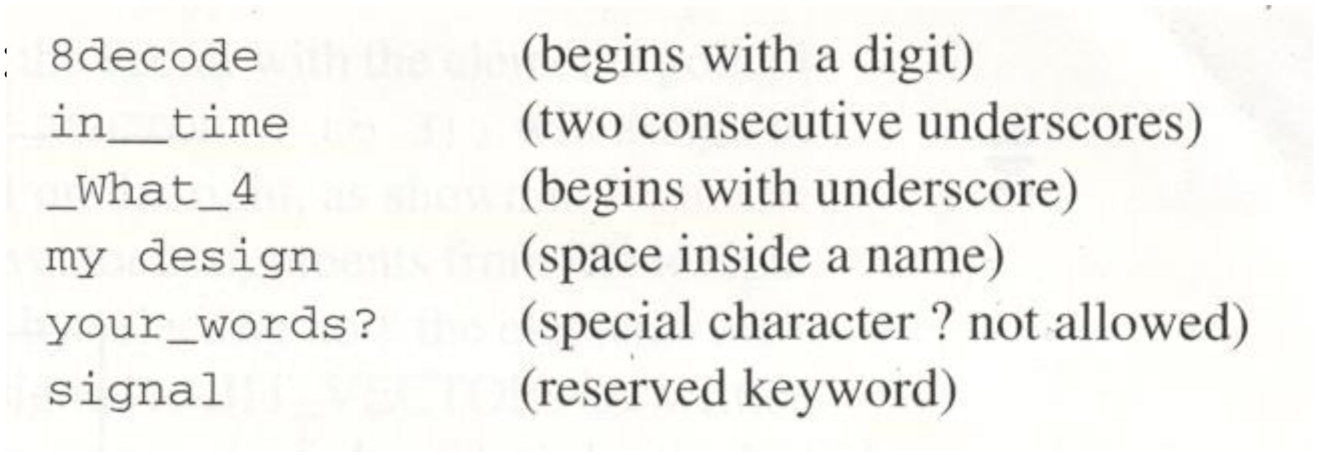
Reserved Keywords

- Reserved keywords are words that have a specific function in VHDL.
- They cannot be used as object names.
- A complete listing of the VHDL reserved keyword can be found in the Quartus II Help File.

Example

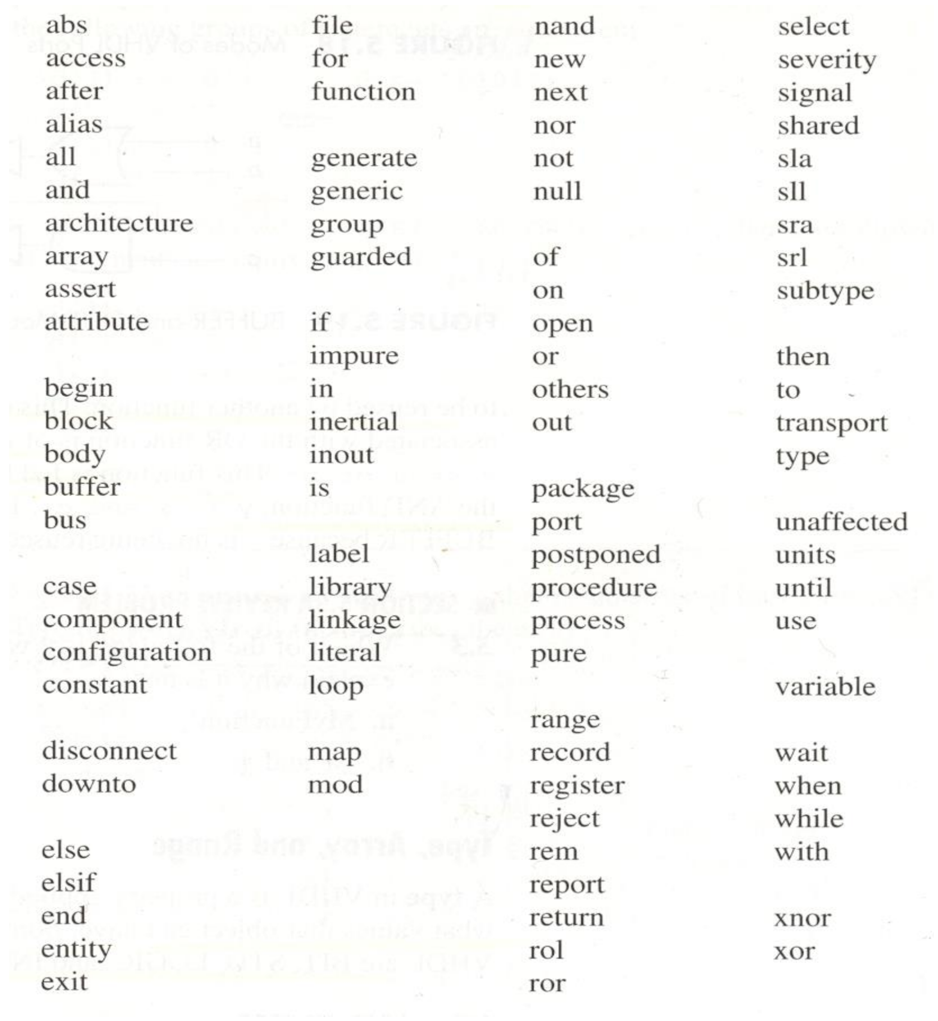
- Valid names: decode8
  just_in_time
  What_4

- Invalid names:



8decode	(begins with a digit)
in__time	(two consecutive underscores)
_What_4	(begins with underscore)
my design	(space inside a name)
your_words?	(special character ? not allowed)
signal	(reserved keyword)

Reserved Keywords for VHDL 1993



abs	file	nand	select
access	for	new	severity
after	function	next	signal
alias		nor	shared
all	generate	not	sla
and	generic	null	sll
architecture	group		sra
array	guarded	of	srl
assert		on	subtype
attribute	if	open	
	impure	or	then
begin	in	others	to
block	inertial	out	transport
body	inout		type
buffer	is	package	
bus		port	unaffected
	label	postponed	units
case	library	procedure	until
component	linkage	process	use
configuration	literal	pure	
constant	loop		variable
		range	
disconnect	map	record	wait
downto	mod	register	when
		reject	while
else		rem	with
elsif		report	
end		return	xnor
entity		rol	xor
exit		ror	

Comments

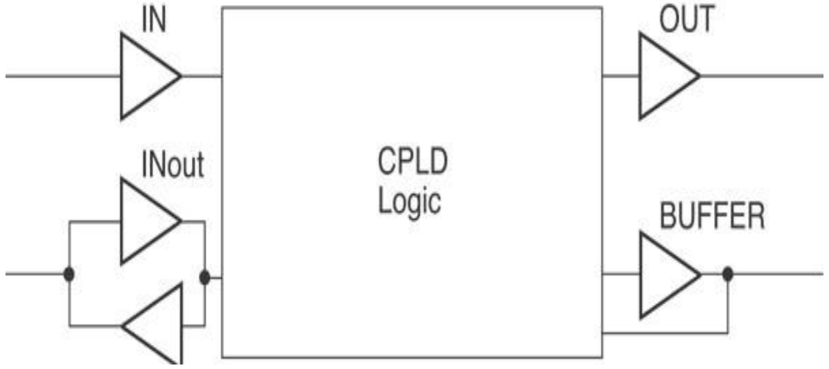
- A comment is explanatory text that is ignored by the compiler.
- Comments are preceded by two consecutive hyphens.
- Example: -- this is a comment.

Port Modes – 1

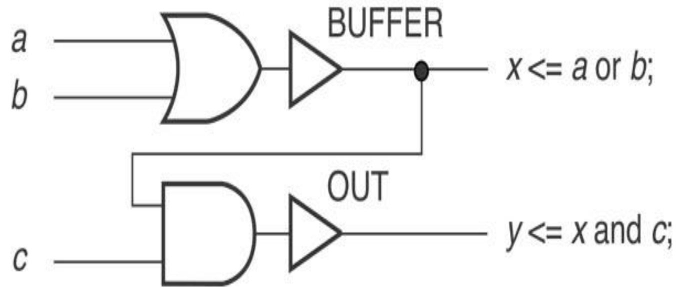
- Defines the ports direction of data flow.
- **IN** - data flows from an INPUT pin to the CPLD's logic.
- **OUT** – data flows from the CPLD's logic to an OUTPUT.
- **INOUT** - refers to a bidirectional port that allows data to flow in both directions.

Port Modes – 2

- **BUFFER** refers to a special case of OUT that has a feedback connection back into the CPLD logic that allows the port value to be changed by the CPLD.



Modes of VHDL ports



BUFFER and OUT modes

Type

- A **type** in VHDL is a property applied to a port, signal or variable that defines what values the object can have.

- Common types: BIT, STD_LOGIC and INTEGER.**(常用的)**

BIT

- BIT can have only two values ‘0’ and ‘1’.
- Values are placed in single quotes.
- VHDL treats them like ASCII characters.

BIT_VECTOR

- BIT_VECTOR: a one-dimensional **array** of elements, each of type BIT.
- The range of the array is indicated by listing its upper and lower bounds.
- d: IN BIT_VECTOR (3 **downto** 0).
- d: IN BIT_VECTOR (0 **to** 3).

IN BIT_VECTOR (3 downto 0)

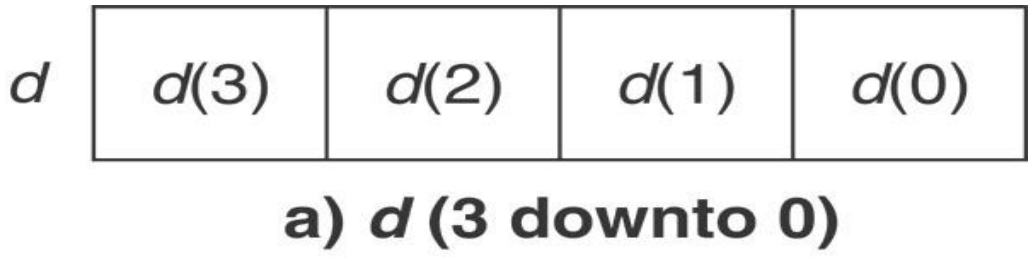
d(3) <= ‘0’;	d3d2d1d0 d <= “0101”; (一次設定) 高位在左邊
d(2) <= ‘1’;	
d(1) <= ‘0’;	
d(0) <= ‘1’	

IN BIT_VECTOR (0 to 3)

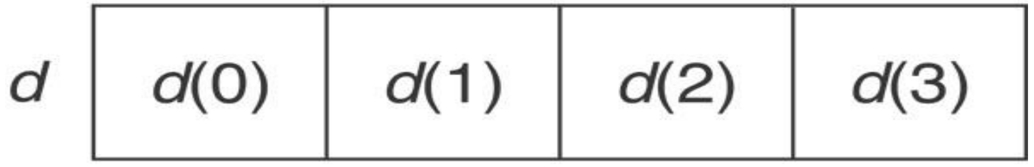
d(3) <= ‘0’;	d0d1d2d3 d <= “1010”; 高位在右邊
d(2) <= ‘1’;	
d(1) <= ‘0’;	
d(0) <= ‘1’	

IN BIT_VECTOR (0 to 3)

- One-dimensional array



a) *d* (3 downto 0)



b) *d* (0 to 3)

Example 5.2 -1

- Bit

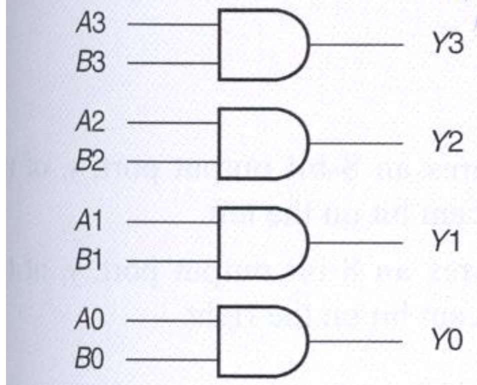


FIGURE 5.21 4-Bit AND Array

Figure 5.21 shows a 4-bit AND array, that is, an array of four 2-bit AND gates. The following VHDL file describes the array.

Bit (--) Comments

```
-- 4-bit bitwise and function
-- y3 = a3 and b3; y2 = a2 and b2; etc.
ENTITY bitwise_and_4 IS
   PORT(
        a3, a2, a1, a0 : IN BIT;
        b3, b2, b1, b0 : IN BIT;
        y3, y2, y1, y0 : OUT BIT);
END bitwise_and_4;

ARCHITECTURE and_gate OF bitwise_and_4 IS
BEGIN
   y3 <= a3 and b3;
   y2 <= a2 and b2;
   y1 <= a1 and b1;
   y0 <= a0 and b0;
END and_gate;
```

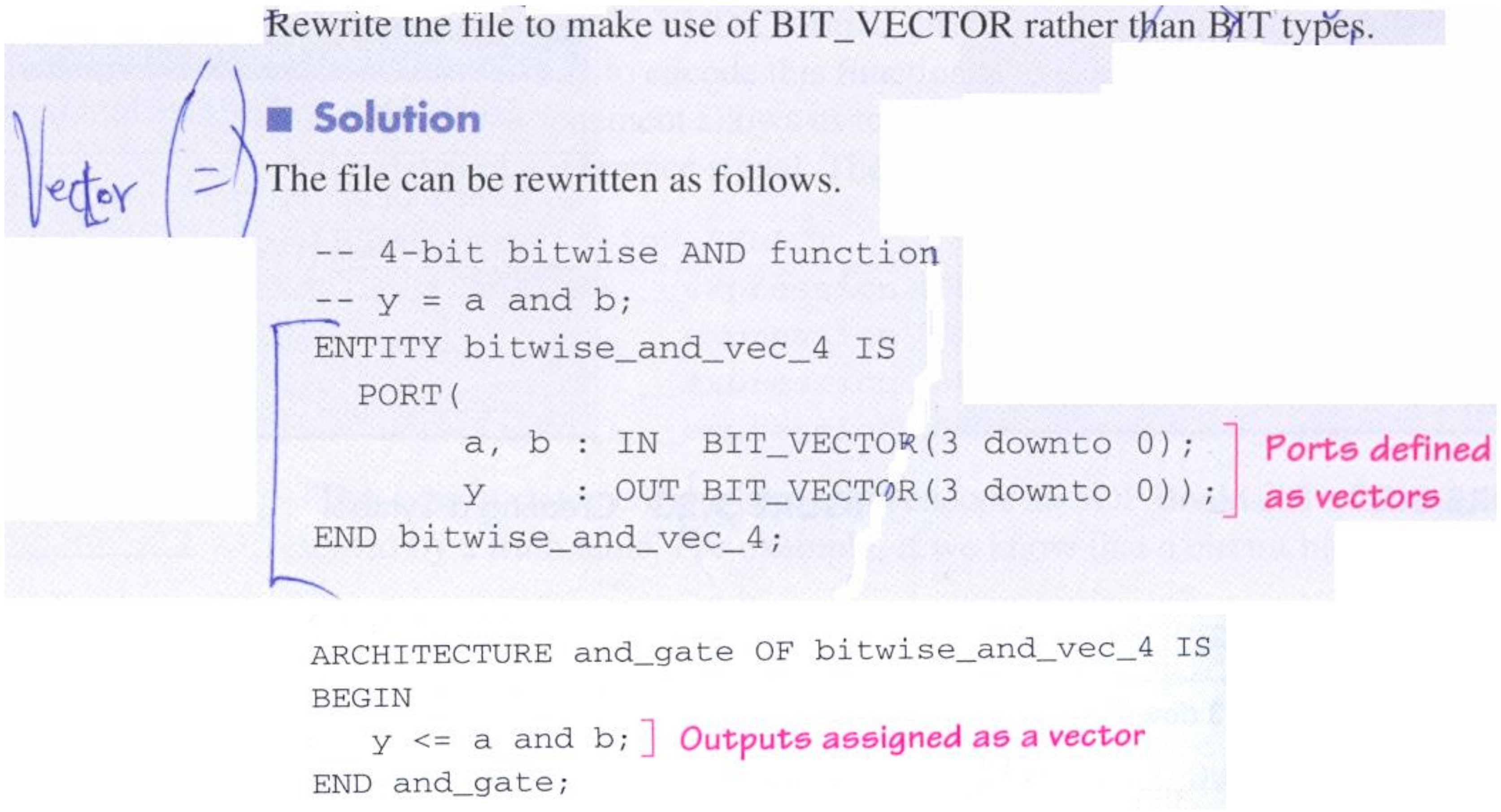
mode type

Ports defined individually

Outputs assigned individually

Example 5.2 -2

- Bit Vector



Rewrite the file to make use of BIT_VECTOR rather than BIT types.

Vector (=)

Solution

The file can be rewritten as follows.

```
-- 4-bit bitwise AND function
-- y = a and b;
ENTITY bitwise_and_vec_4 IS
  PORT(
        a, b : IN  BIT_VECTOR(3 downto 0);
        y    : OUT BIT_VECTOR(3 downto 0));
END bitwise_and_vec_4;
```

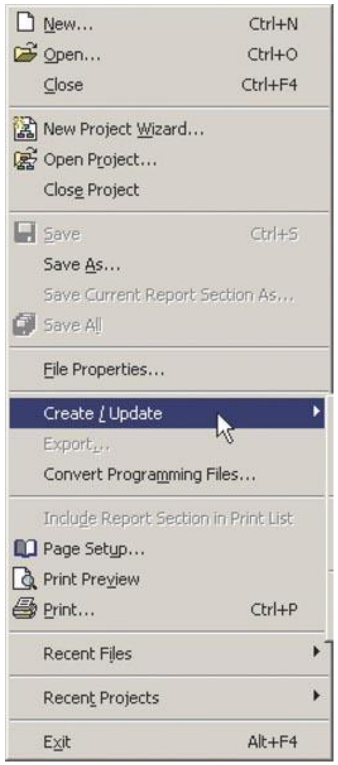
Ports defined as vectors

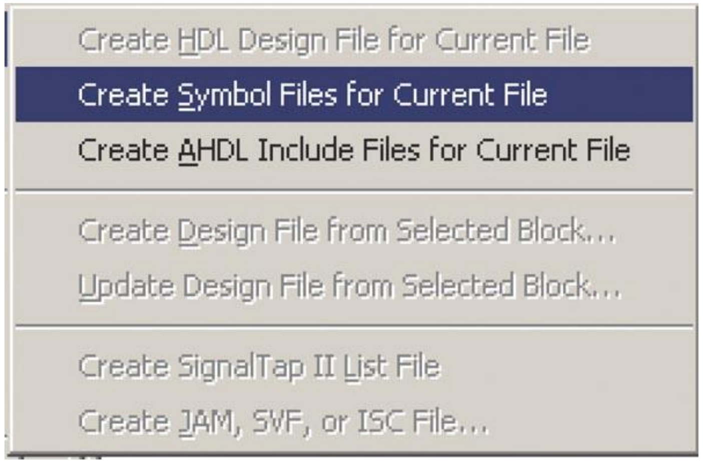
```
ARCHITECTURE and_gate OF bitwise_and_vec_4 IS
BEGIN
   y <= a and b;
END and_gate;
```

Outputs assigned as a vector

Making a Graphic Symbol from VHDL

- Open the VHDL file and its associated project.
- Select **Create/Update** from the **File** menu.
- Select **Create Symbol Files for the Current File**

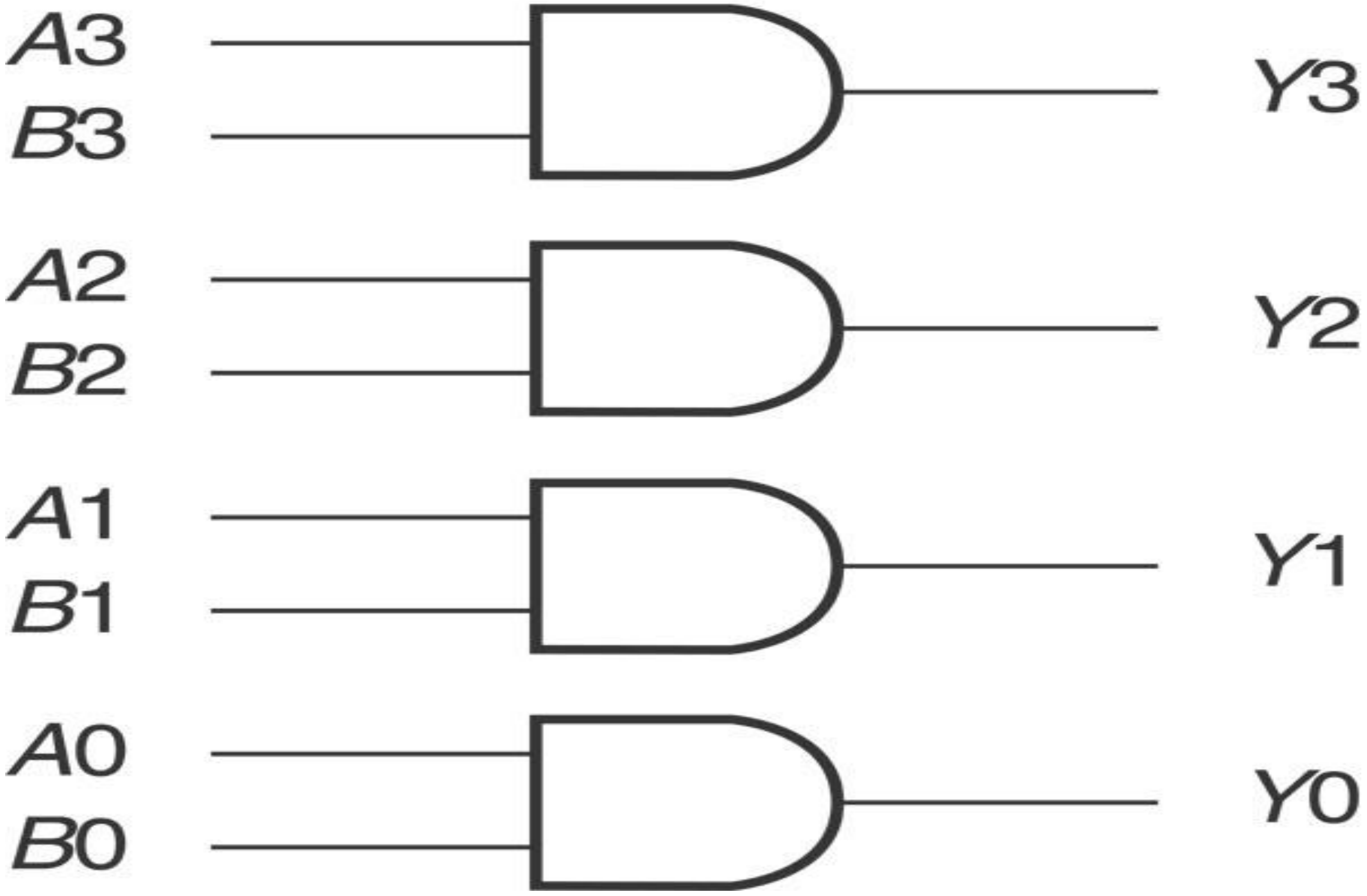



Create HDL Design File for Current File
Create Symbol Files for Current File
Create AHDL Include Files for Current File
Create Design File from Selected Block...
Update Design File from Selected Block...
Create SignalTap II List File
Create JAM, SVF, or ISC File...

VHDL Input & Output Definition -1

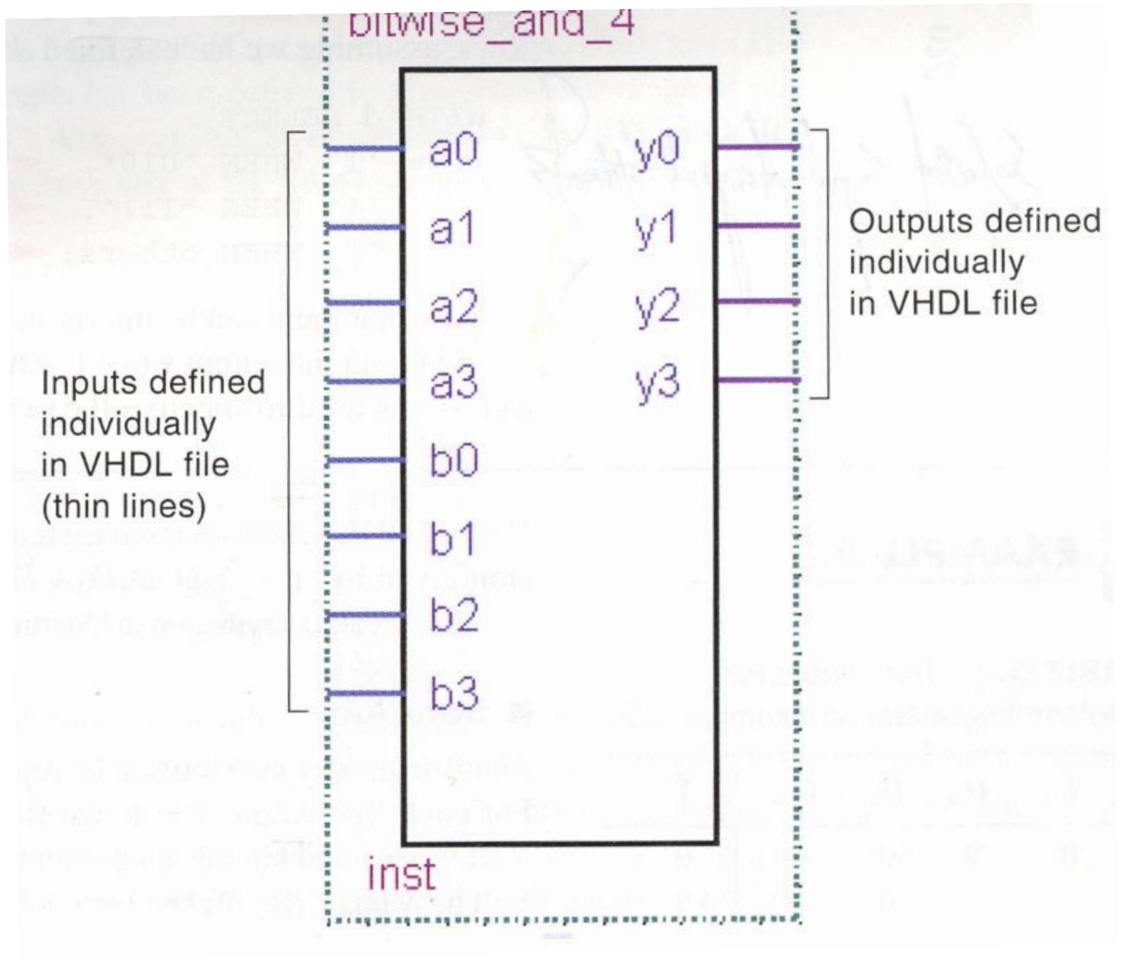
- A graphic symbol is derived from VHDL code:
  - VHDL code defining the inputs and outputs as separate ports shows the inputs and outputs as thin lines. (細線)
  - VHDL code defining inputs and outputs as vectors shows the inputs and outputs as thick lines. (粗線)

VHDL Input & Output Definition -2



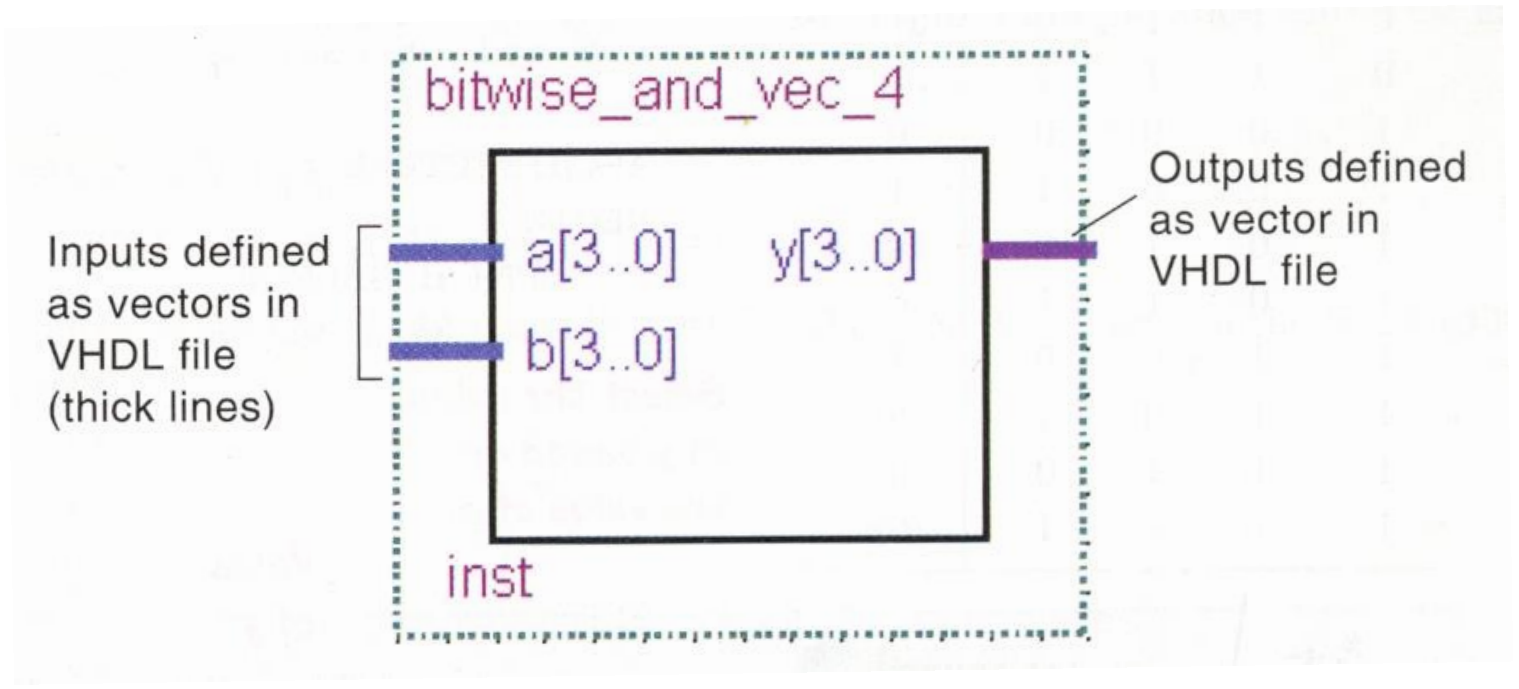
VHDL Input & Output Definition -3

- Symbol for 4-bit AND array (Separate Inputs)


bitwise_and_4
a0
a1
a2
a3
b0
b1
b2
b3
y0
y1
y2
y3
inst
Inputs defined individually in VHDL file (thin lines)
Outputs defined individually in VHDL file

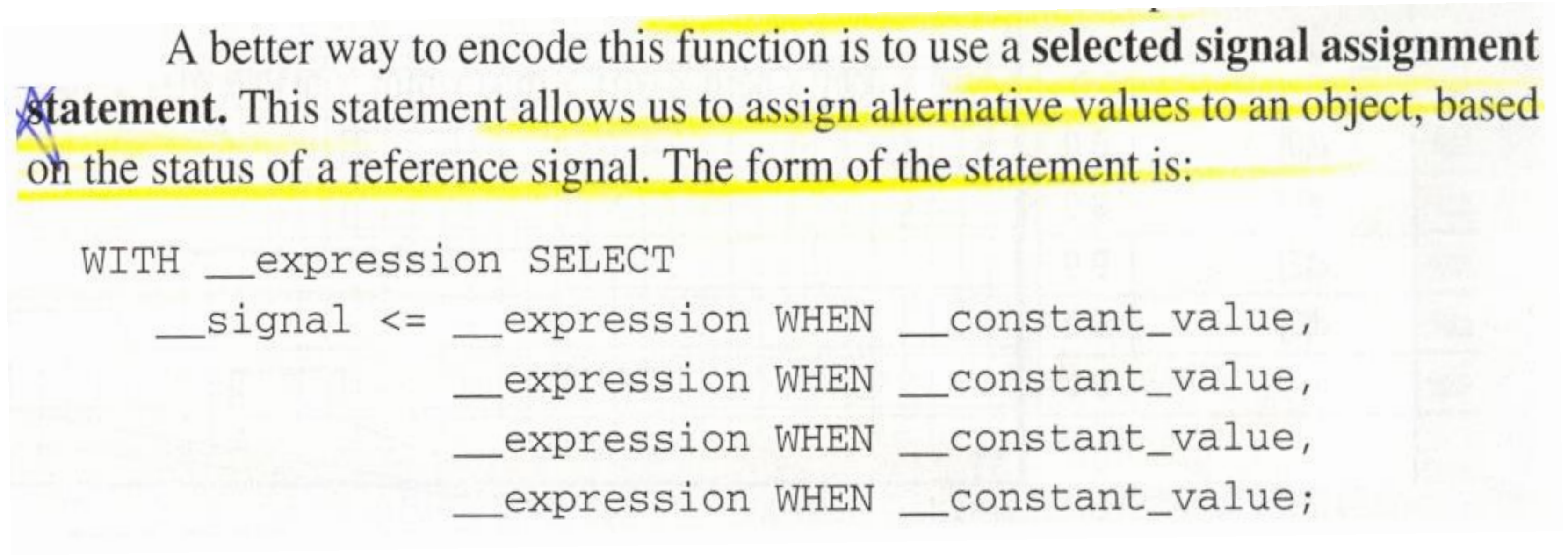
VHDL Input & Output Definition -4

- Symbol for 4-bit AND array (Combined Inputs)


bitwise_and_vec_4
Inputs defined as vectors in VHDL file (thick lines)
a[3..0]
b[3..0]
y[3..0]
Outputs defined as vector in VHDL file
inst

Selected Signal Assignment Statements -1

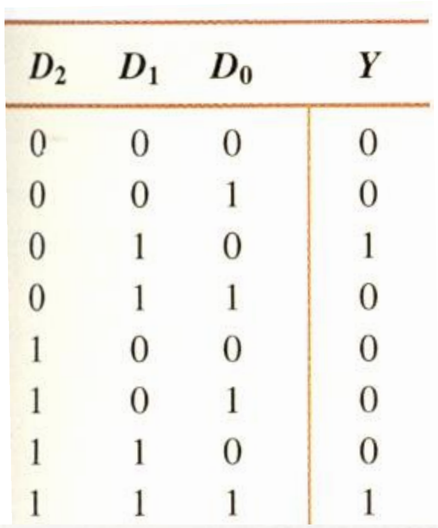
- **Selected Signal Assignment Statements** list alternatives that are available for each value of an expression, then select a course（路線）of action based on the value of the expression.



A better way to encode this function is to use a **selected signal assignment statement.** This statement allows us to assign alternative values to an object, based on the status of a reference signal. The form of the statement is:

```
WITH __expression SELECT
   __signal <= __expression WHEN __constant_value,
               __expression WHEN __constant_value,
               __expression WHEN __constant_value,
               __expression WHEN __constant_value;
```

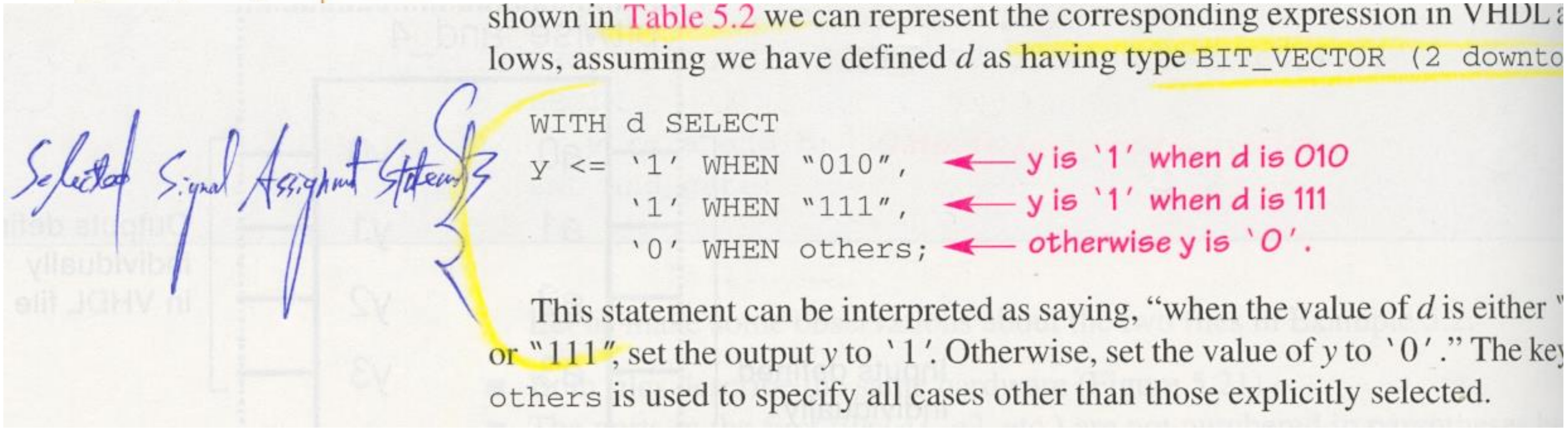
Selected Signal Assignment Statements -2



D_2	D_1	D_0	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Boolean Expression

$$Y = \overline{D^2} D^1 \overline{D^0} + D^2 D^1 D^0$$



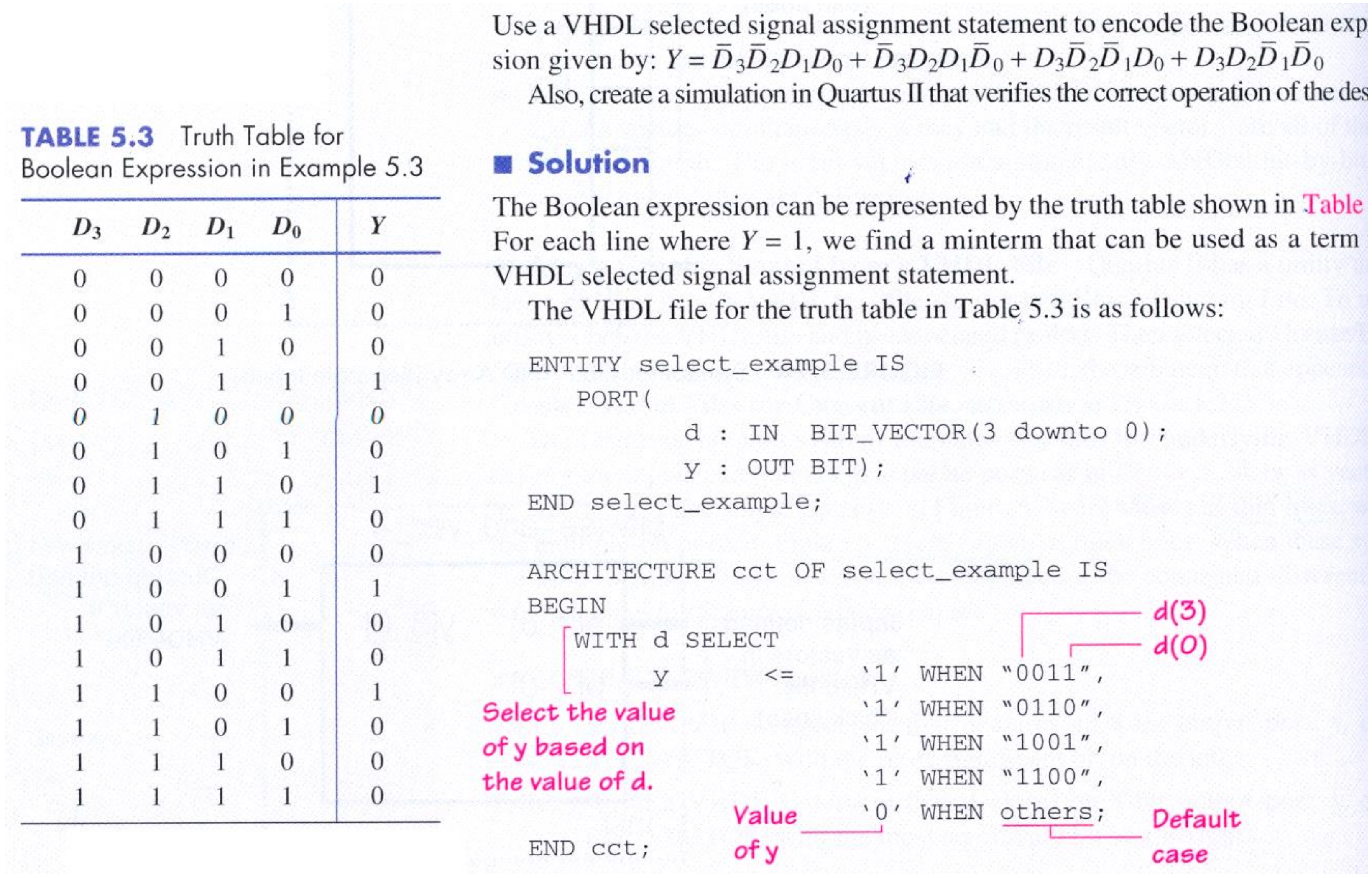
shown in Table 5.2 we can represent the corresponding expression in VHDL a
lows, assuming we have defined *d* as having type `BIT_VECTOR (2 downto`

Selected Signal Assignment Statements

```
WITH d SELECT
y <= '1' WHEN "010",     ← y is '1' when d is 010
     '1' WHEN "111",     ← y is '1' when d is 111
     '0' WHEN others;    ← otherwise y is '0'.
```

This statement can be interpreted as saying, "when the value of *d* is either "
or "111", set the output *y* to '1'. Otherwise, set the value of *y* to '0'." The key
`others` is used to specify all cases other than those explicitly selected.

Example 5.3 -1



Use a VHDL selected signal assignment statement to encode the Boolean exp
sion given by: $Y = \bar{D}_3\bar{D}_2D_1D_0 + \bar{D}_3D_2D_1\bar{D}_0 + D_3\bar{D}_2\bar{D}_1D_0 + D_3D_2\bar{D}_1\bar{D}_0$
Also, create a simulation in Quartus II that verifies the correct operation of the des

TABLE 5.3 Truth Table for Boolean Expression in Example 5.3

D_3	D_2	D_1	D_0	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

■ Solution

The Boolean expression can be represented by the truth table shown in Table
For each line where $Y = 1$, we find a minterm that can be used as a term
VHDL selected signal assignment statement.

The VHDL file for the truth table in Table 5.3 is as follows:

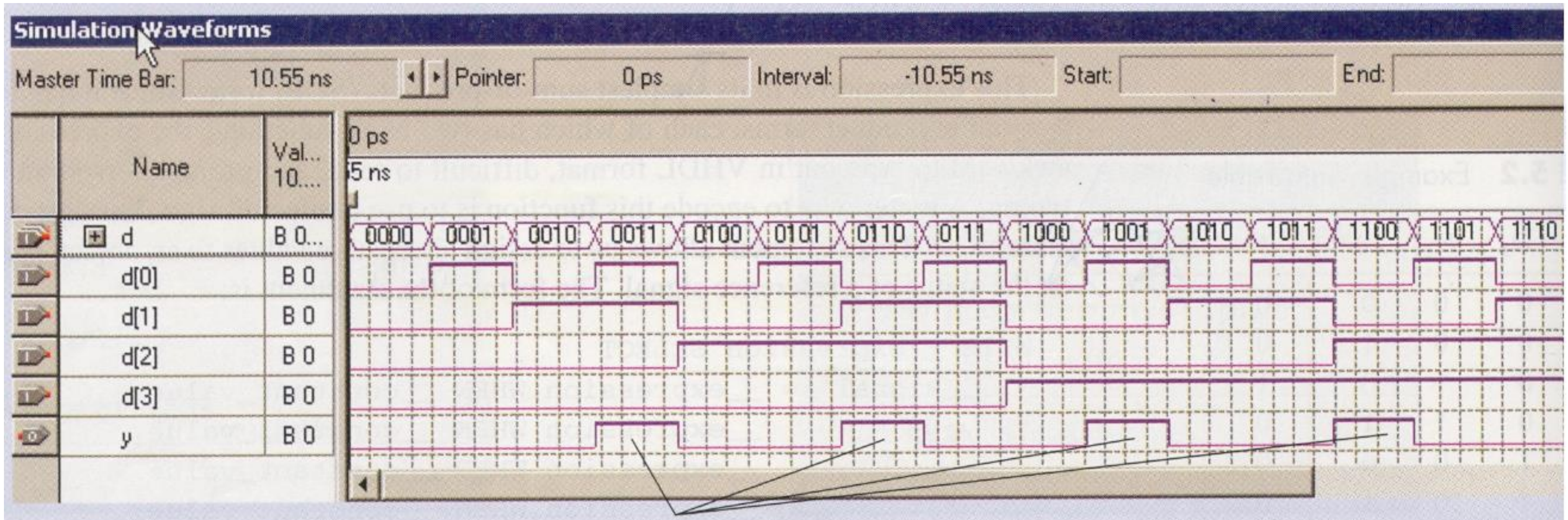
```
ENTITY select_example IS
   PORT(
            d : IN  BIT_VECTOR(3 downto 0);
            y : OUT BIT);
END select_example;

ARCHITECTURE cct OF select_example IS
BEGIN
   WITH d SELECT
        y        <=     '1' WHEN "0011",
                        '1' WHEN "0110",
                        '1' WHEN "1001",
                        '1' WHEN "1100",
                        '0' WHEN others;

END cct;
```

Example 5.3 -2

Figure 5.26 shows the simulation of the design, indicating that the outputs HIGH where selected in the VHDL file and LOW elsewhere.



Output HIGH for selected values of d (0011, 0110, 1000, 1100), otherwise LOW.

STD_LOGIC or STD_LOGIC VECTOR

- **IEEE Std. 1164 Multi-Valued Logic**.
- Gives a broader range of output values than just ‘0’ and ‘1’.
- Can be any of nine values.

IEEE Std. 1164 Multi-Valued Logic – 1

- “Forcing” level: gate output
- “Weak” level: specified by a pull-up or pull-down resistor
  not important to us
- “Z” state: high-impedance state of tristate buffer
- Majority of applications : “X”, “0”, “1”, and “Z”

' U '	Uninitialized
' X '	Forcing Unknown
' 0 '	Forcing 0
' 1 '	Forcing 1
' Z '	High Impedance
' W '	Weak Unknown
' L '	Weak 0
' H '	Weak 1
' - '	Don't Care

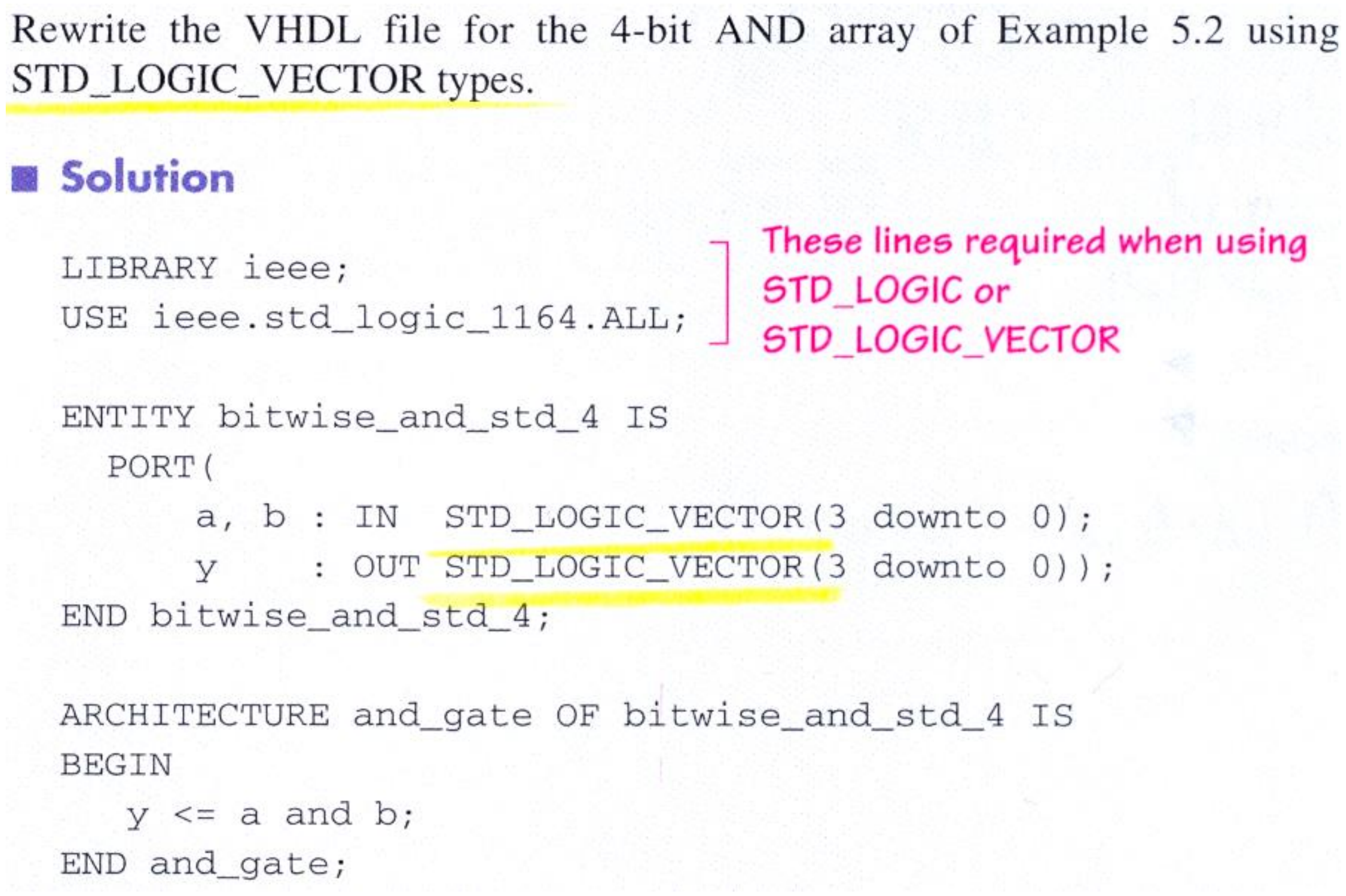
IEEE Std. 1164 Multi-Valued Logic – 2

- To use STD_LOGIC in a VHDL file:
  - Include reference to the **ieee** VHDL library and the **std_logic_1164** package before the entity declaration.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL
```

- The **std_logic_1164** package contains all type definitions of the STD_LOGIC types.

Example 5.4



Rewrite the VHDL file for the 4-bit AND array of Example 5.2 using STD_LOGIC_VECTOR types.

■ Solution

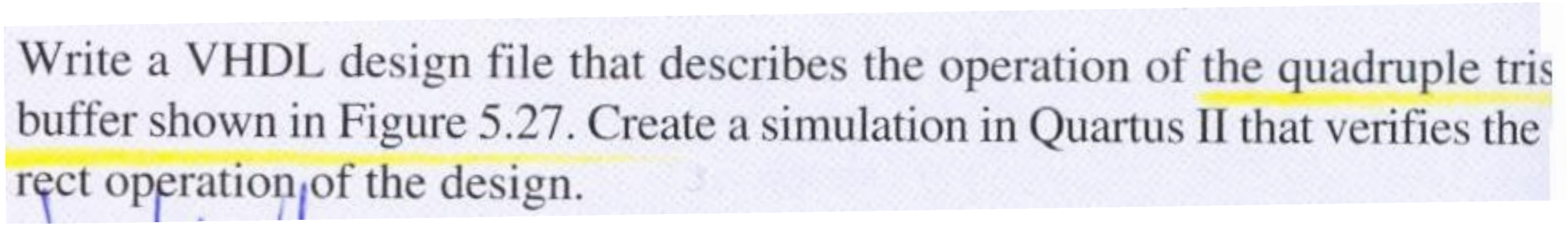
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
```

These lines required when using STD_LOGIC or STD_LOGIC_VECTOR

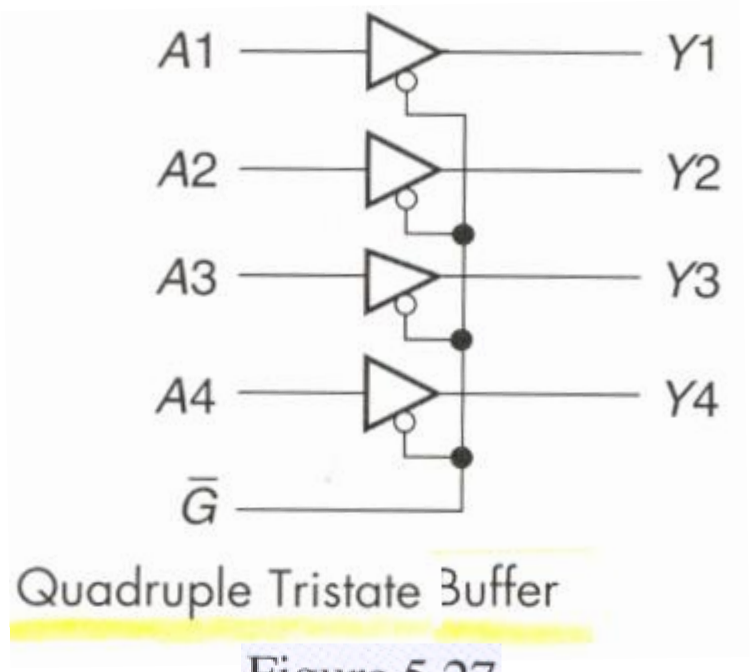
```
ENTITY bitwise_and_std_4 IS
   PORT(
         a, b : IN  STD_LOGIC_VECTOR(3 downto 0);
         y    : OUT STD_LOGIC_VECTOR(3 downto 0));
END bitwise_and_std_4;

ARCHITECTURE and_gate OF bitwise_and_std_4 IS
BEGIN
    y <= a and b;
END and_gate;
```

Example 5.5 -1

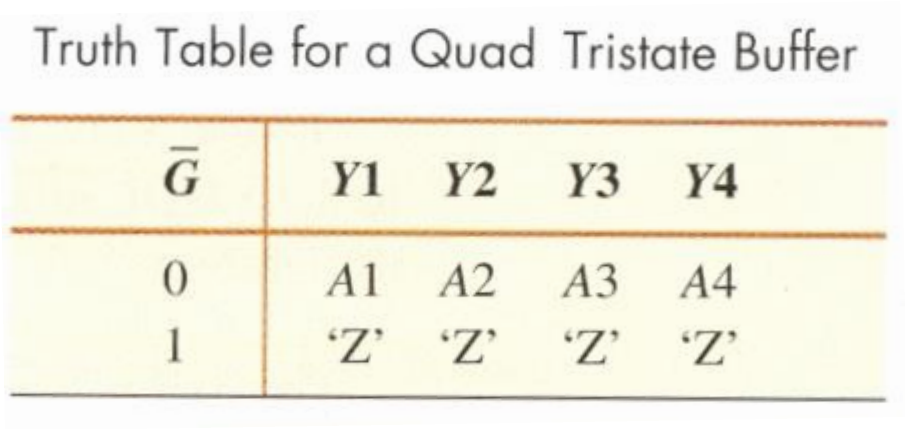


Write a VHDL design file that describes the operation of the quadruple tris
buffer shown in Figure 5.27. Create a simulation in Quartus II that verifies the
rect operation of the design.



Quadruple Tristate Buffer

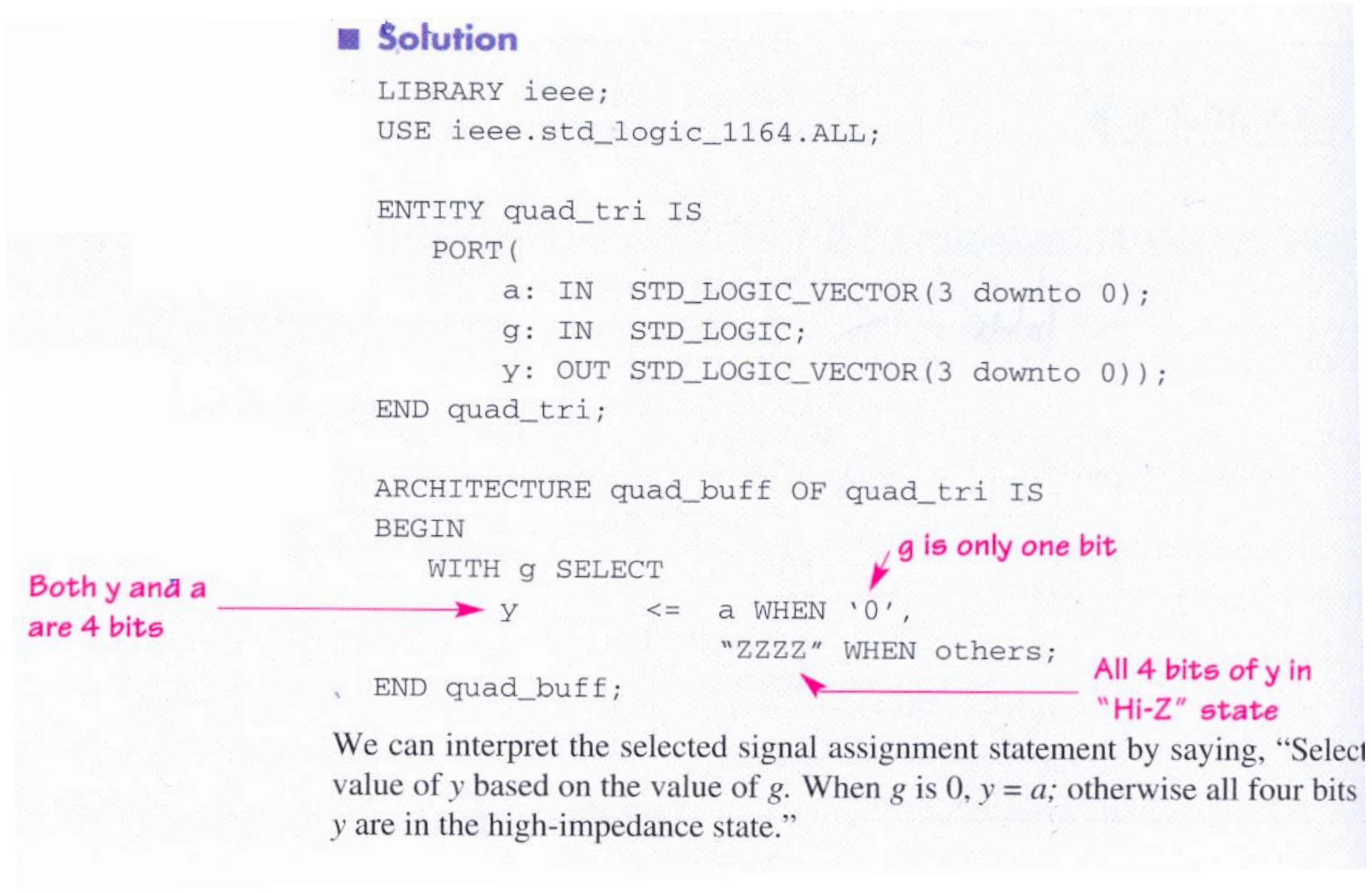
Figure 5.27



Truth Table for a Quad Tristate Buffer

$\overline{G}$	Y1	Y2	Y3	Y4
0	A1	A2	A3	A4
1	'Z'	'Z'	'Z'	'Z'

Example 5.5 -2



Solution

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY quad_tri IS
   PORT(
        a: IN  STD_LOGIC_VECTOR(3 downto 0);
        g: IN  STD_LOGIC;
        y: OUT STD_LOGIC_VECTOR(3 downto 0));
END quad_tri;

ARCHITECTURE quad_buff OF quad_tri IS
BEGIN
   WITH g SELECT
        y        <=  a WHEN '0',
                     "ZZZZ" WHEN others;
END quad_buff;
```

Both y and a are 4 bits

g is only one bit

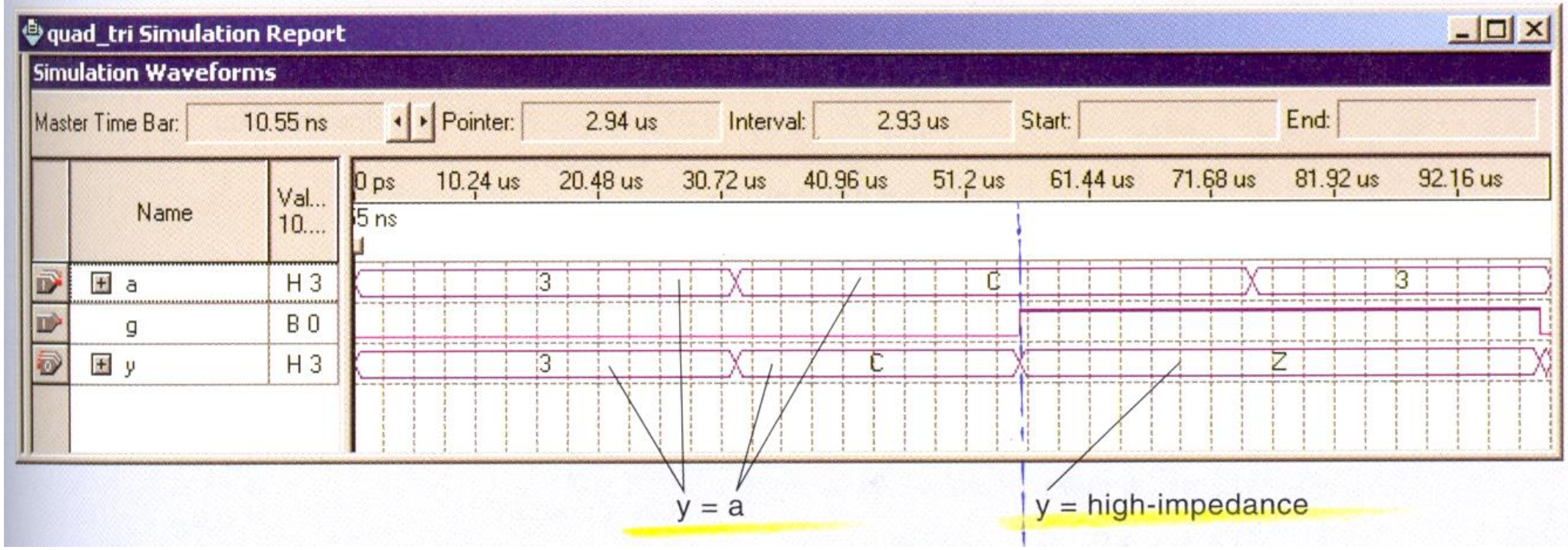
All 4 bits of y in "Hi-Z" state

We can interpret the selected signal assignment statement by saying, "Select value of *y* based on the value of *g*. When *g* is 0, *y* = *a;* otherwise all four bits *y* are in the high-impedance state."

Example 5.5 -3

Simulation of a 4-bit Tristate Buffer

Figure 5.28 shows a simulation of the 4-bit tristate buffer. As long as *g* is LOW, output *y* follows input *a*, first with a hexadecimal value of 3, then C. When *g* is HIGH, the output is in the high-impedance state, shown by Z.



Integers

- VHDL INTEGER types are represented by the range of 32-bit positive and negative numbers.

  –2,147,483,648 to +2,147,483,647.

- The following two expressions produce the same result in hardware:

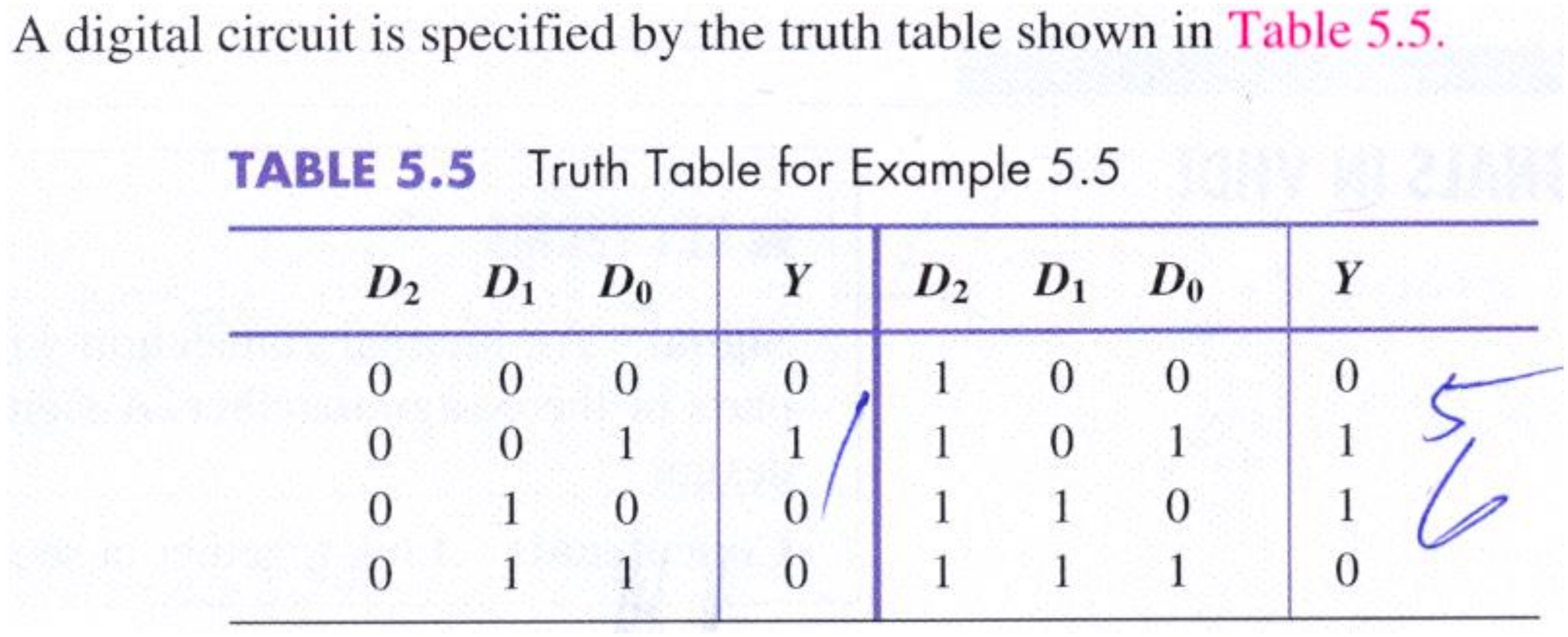
  ```
  d:  IN BIT_VECTOR (3 downto 0);
  d:  IN INTEGER RANGE (0 to 7);
  ```

NATURAL & POSITIVE Subtypes

- NATURAL: (自然數)
  - The set of all integers greater than or equal to ‘0’.
- POSITIVE: (正整數)
  - The set of all integers greater than or equal to 1.
- Constants in all these types are written in VHDL without quotes(“ ”) (e.g., y <= 3;).

Example 5.6 -1

- Write VHDL for the following truth table



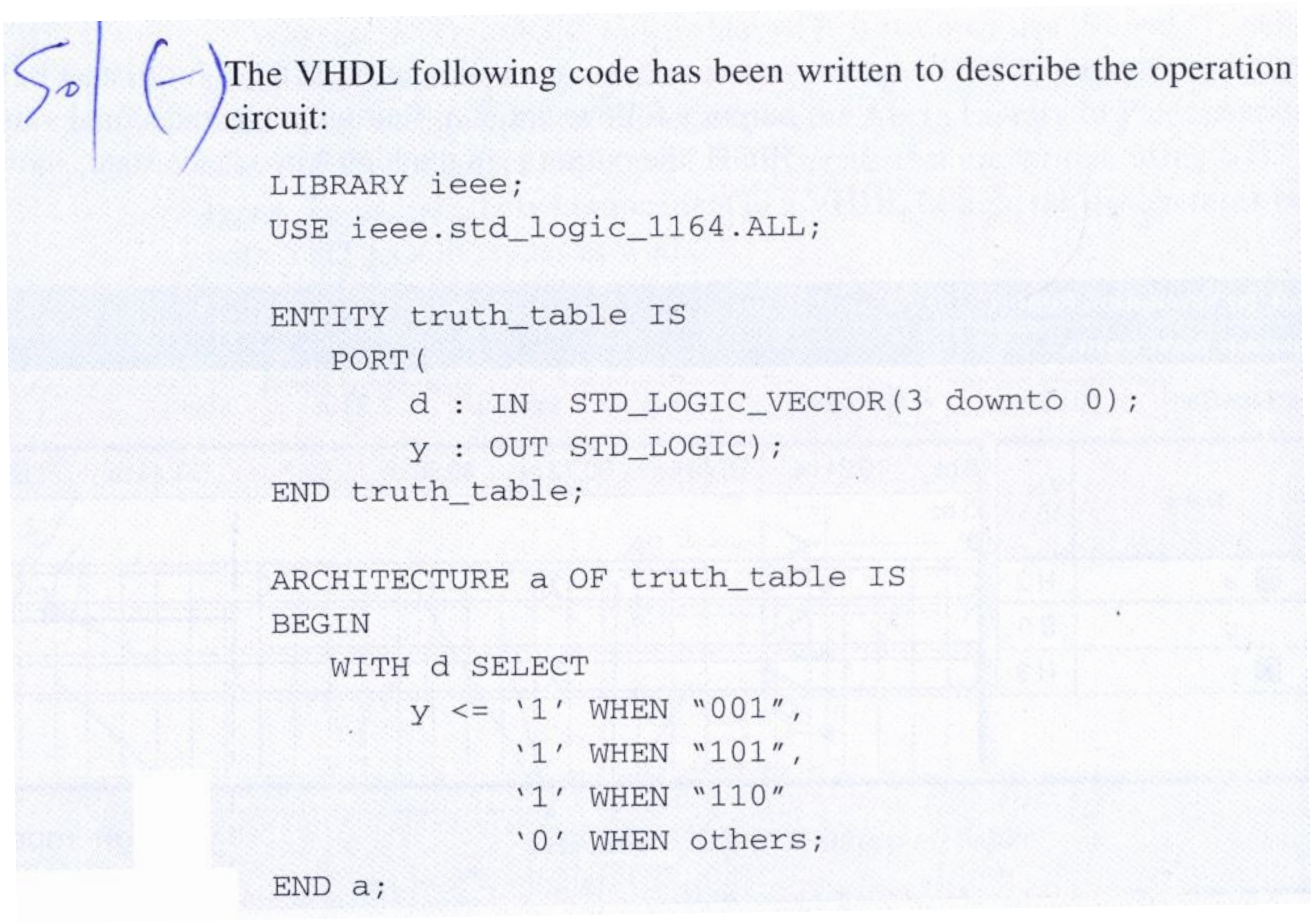
A digital circuit is specified by the truth table shown in Table 5.5.

TABLE 5.5 Truth Table for Example 5.5

D_2	D_1	D_0	Y	D_2	D_1	D_0	Y
0	0	0	0	1	0	0	0
0	0	1	1	1	0	1	1
0	1	0	0	1	1	0	1
0	1	1	0	1	1	1	0

5
6

Example 5.6 -2



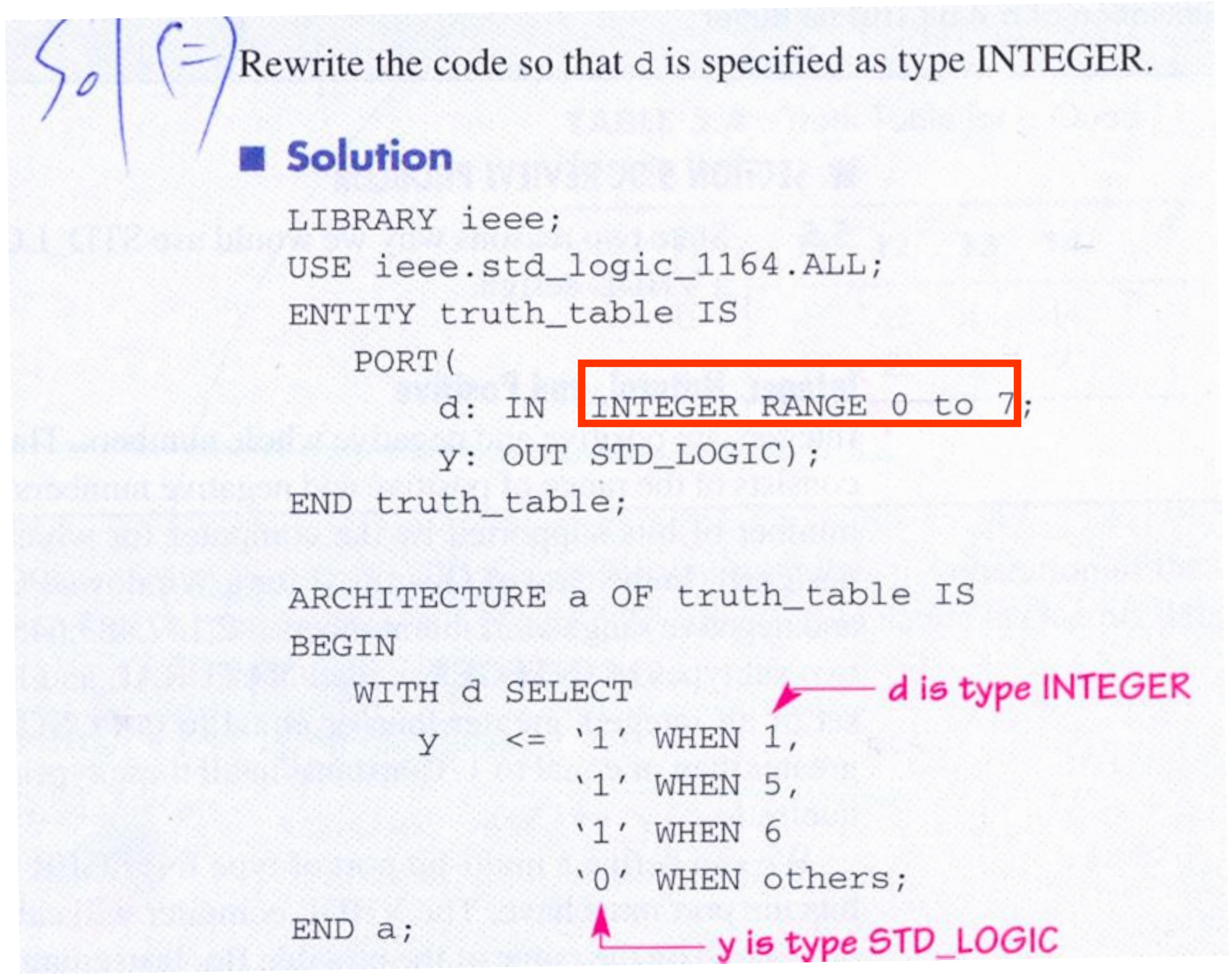
S_0 | (–) The VHDL following code has been written to describe the operation circuit:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY truth_table IS
   PORT(
       d : IN  STD_LOGIC_VECTOR(3 downto 0);
       y : OUT STD_LOGIC);
END truth_table;

ARCHITECTURE a OF truth_table IS
BEGIN
   WITH d SELECT
       y <= '1' WHEN "001",
            '1' WHEN "101",
            '1' WHEN "110"
            '0' WHEN others;
END a;
```

Example 5.6 -3



5o (=)

Rewrite the code so that d is specified as type INTEGER.

Solution

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY truth_table IS
   PORT(
       d: IN  INTEGER RANGE 0 to 7;
       y: OUT STD_LOGIC);
END truth_table;

ARCHITECTURE a OF truth_table IS
BEGIN
   WITH d SELECT                    ← d is type INTEGER
       y    <= '1' WHEN 1,
               '1' WHEN 5,
               '1' WHEN 6
               '0' WHEN others;
END a;
               ↑ y is type STD_LOGIC
```

5.5 Signals in VHDL

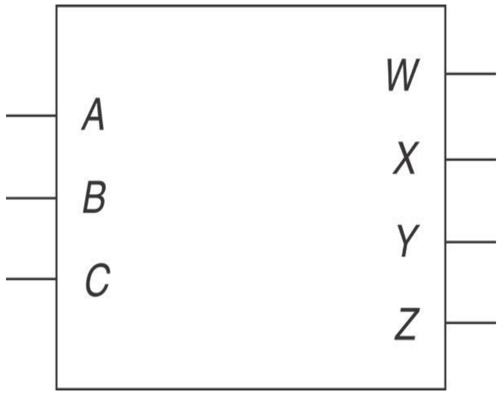
- ❑A signal is defined as an internal connection within a VHDL architecture that connects parts of the design together.

- ❑Acts like an internal wire inside the design.

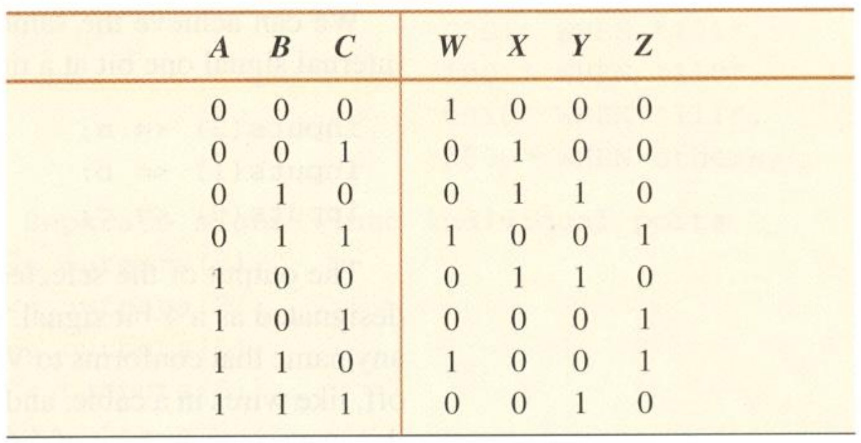
Concatenate (串接)

- ❑Bundling or linking the ports together.
- ❑Uses the **&** operator.
- ❑inputs <= a & b & c;

Signals in VHDL -1



Digital Function Block

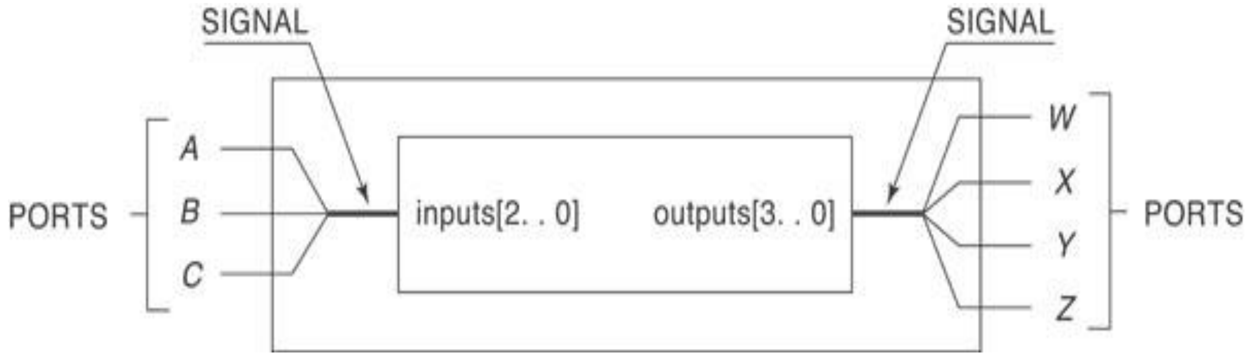


A	B	C	W	X	Y	Z
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	1	0	0	1	1	0
0	1	1	1	0	0	1
1	0	0	0	1	1	0
1	0	1	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	1	0

Truth table

$$W = \bar{A}\,\bar{B}\,\bar{C} + \bar{A}\,B\,C + A\,B\,C$$
$$X = \bar{A}\,\bar{B}\,C + \bar{A}\,B\,\bar{C} + A\,\bar{B}\,\bar{C}$$
$$Y = \bar{A}\,B\,\bar{C} + A\,\bar{B}\,\bar{C} + A\,B\,C$$
$$Z = \bar{A}\,B\,C + A\,\bar{B}\,C + A\,B\,\bar{C}$$

Signals in VHDL -2



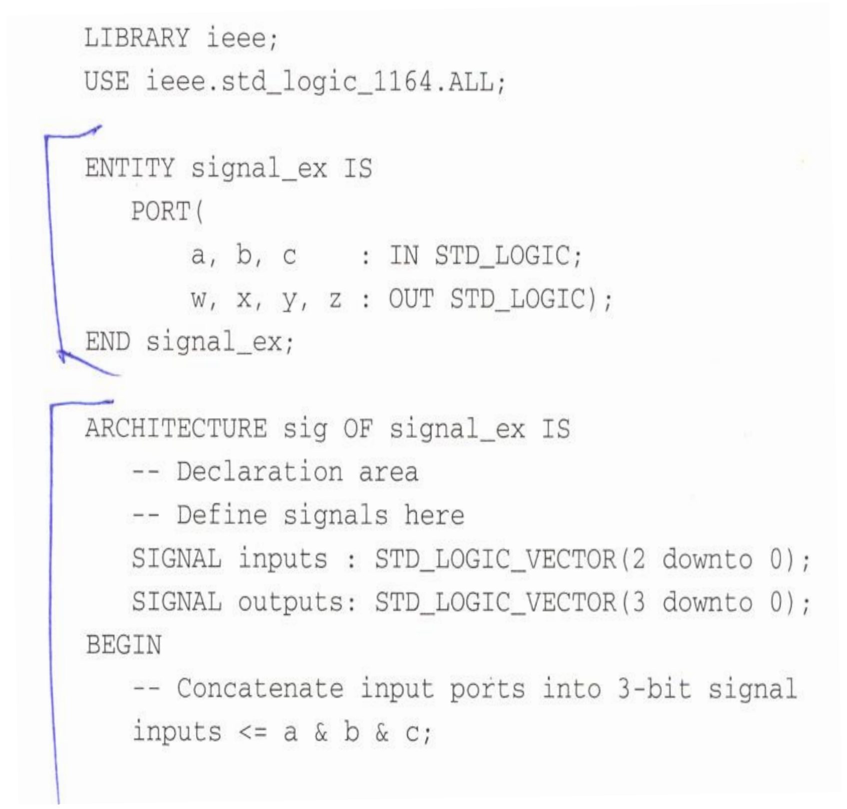
	inputs(2)	inputs(1)	inputs(0)
inputs	a	b	c

	outputs(3)	outputs(2)	outputs(1)	outputs(0)
outputs	w	x	y	z

```
inputs(2) <= a;
inputs(1) <= b;
inputs(0) <= c;
```

```
w <= outputs(3);
x <= outputs(2);
y <= outputs(1);
z <= outputs(0);
```

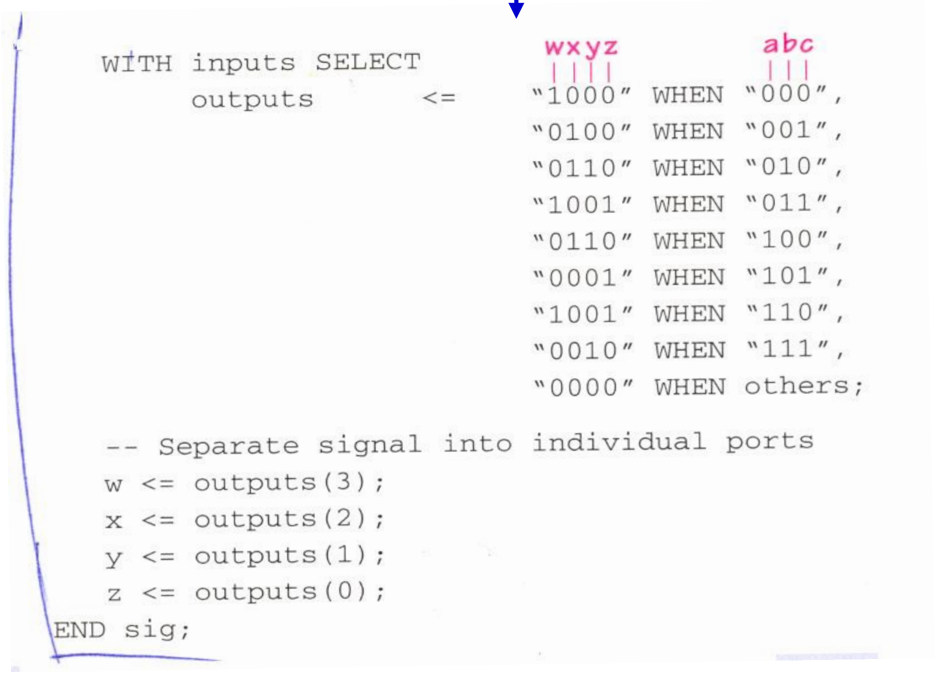
Signals in VHDL -3



```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY signal_ex IS
   PORT(
       a, b, c     : IN STD_LOGIC;
       w, x, y, z : OUT STD_LOGIC);
END signal_ex;

ARCHITECTURE sig OF signal_ex IS
   -- Declaration area
   -- Define signals here
   SIGNAL inputs : STD_LOGIC_VECTOR(2 downto 0);
   SIGNAL outputs: STD_LOGIC_VECTOR(3 downto 0);
BEGIN
   -- Concatenate input ports into 3-bit signal
   inputs <= a & b & c;
```

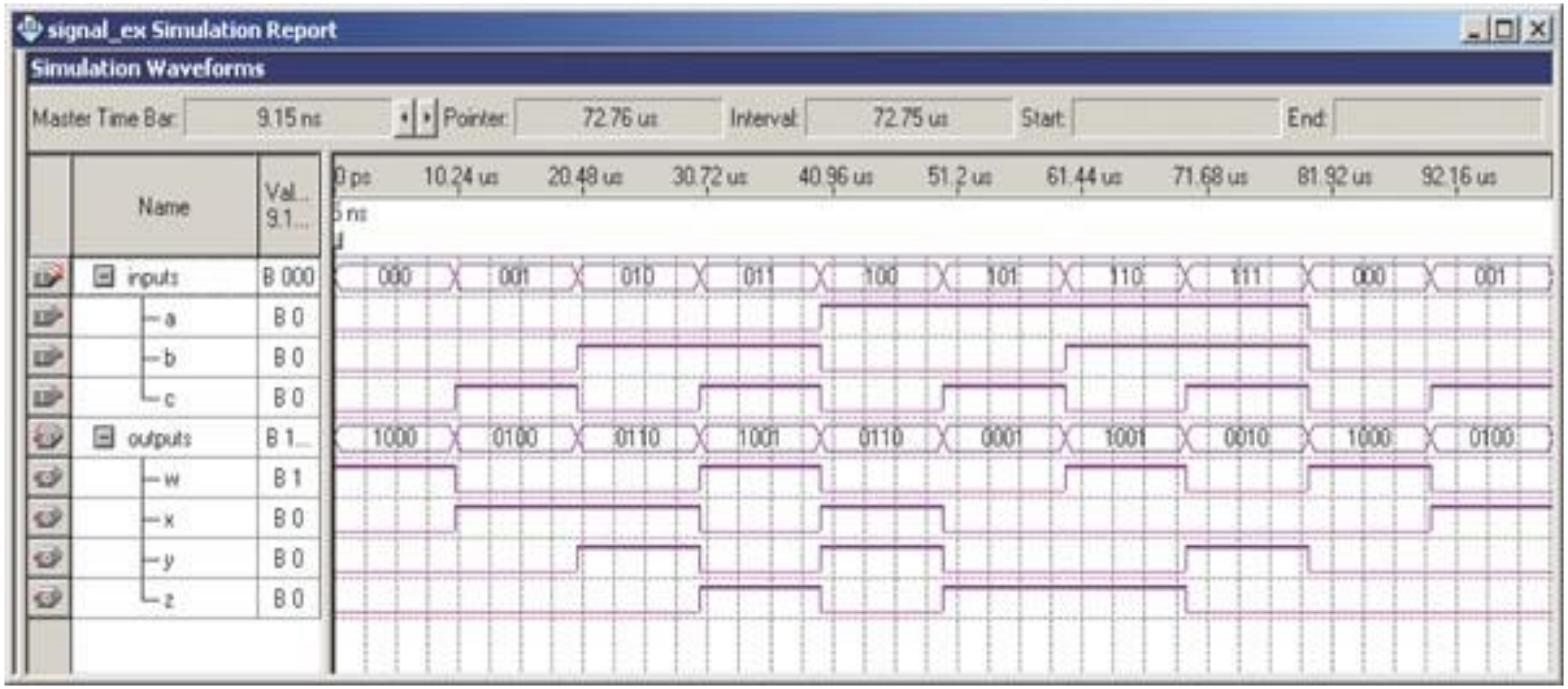


```
      WITH inputs SELECT
            outputs      <=     "1000" WHEN "000",
                                "0100" WHEN "001",
                                "0110" WHEN "010",
                                "1001" WHEN "011",
                                "0110" WHEN "100",
                                "0001" WHEN "101",
                                "1001" WHEN "110",
                                "0010" WHEN "111",
                                "0000" WHEN others;

      -- Separate signal into individual ports
      w <= outputs(3);
      x <= outputs(2);
      y <= outputs(1);
      z <= outputs(0);
END sig;
```

Signals in VHDL -4

The simulation for the example design, including internal signals



signal_ex Simulation Report

Simulation Waveforms

Master Time Bar: 9.15 ns   Pointer: 72.76 us   Interval: 72.75 us   Start: End:

0 ps   10.24 us   20.48 us   30.72 us   40.96 us   51.2 us   61.44 us   71.68 us   81.92 us   92.16 us

Name	Val... 9.1...										
inputs	B 000	000	001	010	011	100	101	110	111	000	001
a	B 0										
b	B 0										
c	B 0										
outputs	B 1...	1000	0100	0110	1001	0110	0001	1001	0010	1000	0100
w	B 1										
x	B 0										
y	B 0										
z	B 0										

Single- & Multiple-Bit Signals

- 3-bit port defined as:
  - d: IN   STD_LOGIC_VECTOR (2 downto 0);

- Single-bit port defined as:
  - enable: IN   STD_LOGIC;

Combining Single- & Multiple-Bit Signals

- Define the signal:
  - inputs : STD_LOGIC_VECTOR (3 downto 0);
- Concatenate the ports into a signal:
  - inputs <= enable & d;

HW