

EE2000 Logic Circuit Design

VHDL

VHDL and EDA

- Digital design could be very complicated and time-consuming. Modern complex system designs rely on EDA (Electronic Design Automation_ tools.
- EDA resources can be reused and it shorten the complexity and design time. Design time is the most critical factor in determining the market share and profits.
- EDA tools are used at all stages in the design process, including design capture, logic synthesis, simulation, layout placement and routing, layout verification.
- VHDL is one of the most commonly EDA tools.

Basic Logic Gates using VHDL

- 5.1 Hardware Description Language (HDL)
- 5.2 VHDL Code Structure
- 5.3 Basic Logic Gates Using VHDL
- 5.4 VHDL Data Types
- 5.5 VHDL Operators - Boolean / Relational / Arithmetic
- 5.6 VHDL Syntax for Boolean expressions / Logic Circuits
- 5.7 VHDL Module
- 5.8 VHDL Format & Syntax

3

5.1 Hardware Description Language (HDL)

- Two popular HDLs—**VHDL** and **Verilog**
- VHDL – Very High Speed Integrated Circuit Hardware Description Language (VHSIC-HDL).
- Developed by U.S. Department of Defense (DoD).
- Standardized by IEEE.
- Widely used to translate designs into bit patterns that configure the actual devices.

4

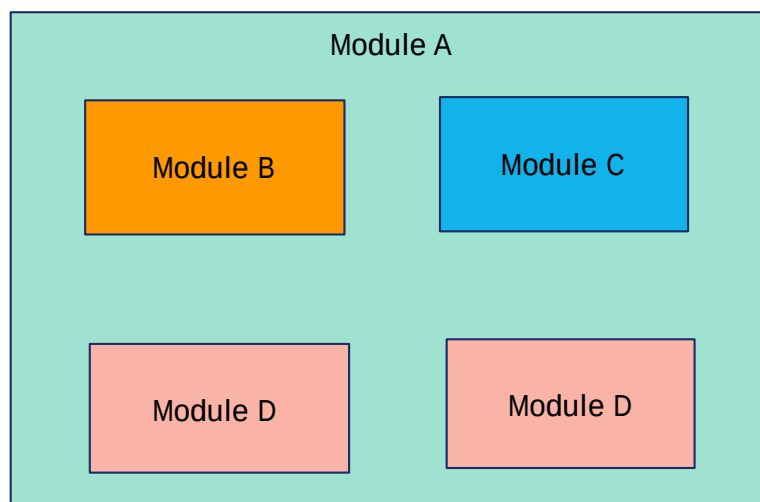
5.1 Hardware Description Language (HDL)

- HDL – Hardware Description Language is a software programming language that is used to model a piece of hardware
- VHDL is an HDL used to describe a digital system at several different levels: behavioral, data flow, and structural.
- VHDL leads naturally to a top-down design methodology.
- The Vivado design tool from Xilinx will be used to design simulation and synthesize the logic circuit/system in this course.



5

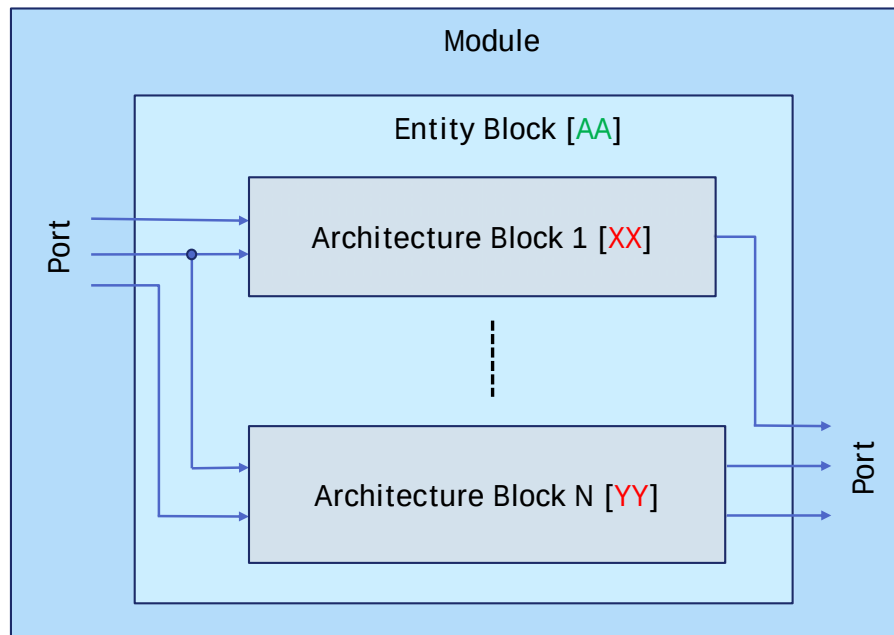
5.2 VHDL Code Structure - Illustration



Like breaking down a complicated circuit into a number of smaller circuits, with each module describe a smaller circuit. The modules are connected through **ports** to form the final circuit.

6

5.2 VHDL Code Structure - Illustration



7

5.2 VHDL Code Structure

```
{Library      Declaration}  
{Entity      Declaration}  
{Architecture Declaration}
```

Library declaration: locate the system library, IEEE Standard library is often included in the VHDL code. STD_LOGIC_1164 contains STD_LOGIC types & related functions. You can create your own library.

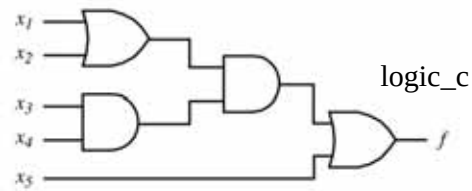
Entity declaration: defines the external interface to the current design [input (port) / output (port)].

Architecture declaration: describes the internal circuit of the corresponding entity [logic circuit].

8

5.2 VHDL Code Structure - Illustration

```
library ieee;  
use ieee.std_logic_1164.all;
```



```
entity logic_c is  
    port(  
        x1,x2,x3,x4,x5 : in  bit;  
        f               : out bit  
    );  
end logic_c;
```

```
architecture Behavior of logic_c is  
begin  
    f <=((x1 or x2) and (x3 and x4) or x5);  
end Behavior;
```

9

5.3 Basic Logic Gates using VHDL

Circuit description with an architecture

The description of the **logic circuit operation as in “Functional code”** is enclosed by **BEGIN** and **END**.

```
ARCHITECTURE XX of AA IS
```

```
BEGIN
```

```
    Functional code
```

```
END XX;
```

```
ARCHITECTURE YY of AA IS
```

```
BEGIN
```

```
    Functional code
```

```
END YY;
```

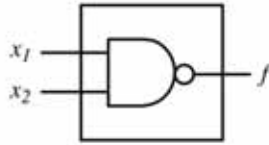
AA: entity name

XX, YY: architecture name

10

5.3 Basic Logic Gates using VHDL

Example:



```
architecture Behavior of nand_gate is  
begin
```

```
f <= not (x1 and x2);
```

```
end Behavior;
```

Alternative:

```
f <= x1 nand x2;
```

11

5.3 Basic Logic Gates using VHDL

VHDL is “case insensitive”.

```
architecture Behavior of nand_gate is  
begin
```

```
f <= x1 nand x2;
```

```
end Behavior;
```

The above code is identical to the following:

```
ARCHITECTURE Behavior of NAND_GATE IS  
BEGIN
```

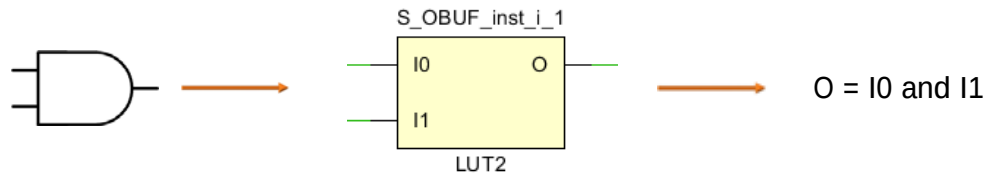
```
F <= x1 NAND x2;
```

```
END Behavior;
```

12

5.3 Basic Logic Gates using VHDL

VHDL language translates logic circuit into Look-up Table

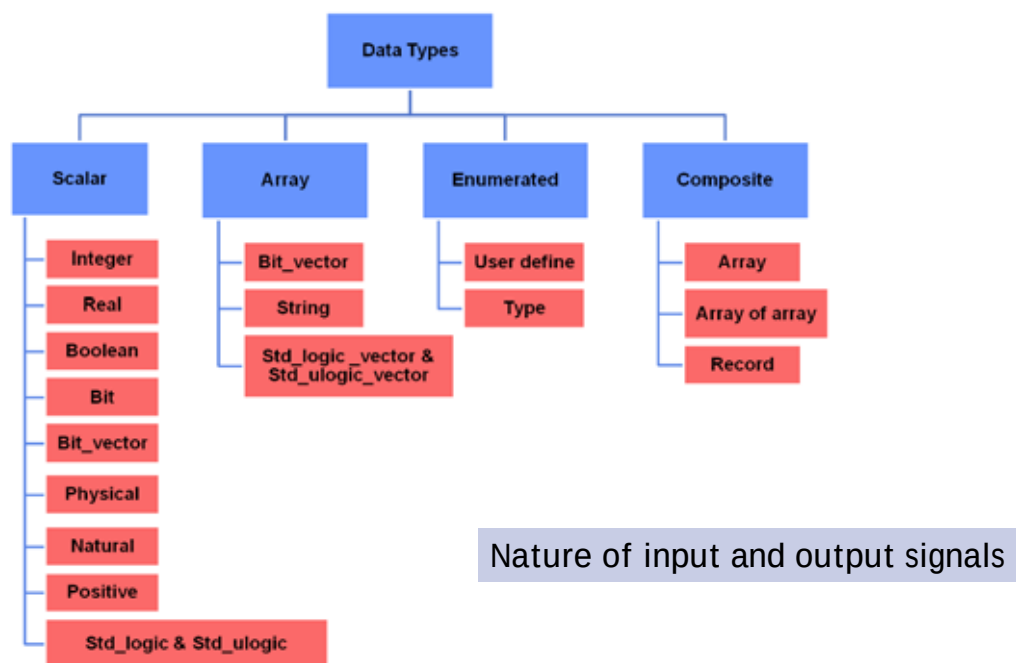


A typical Schematic Diagram for a logic gate in the CAD tool –
e.g. Xilinx Vivado Software

I0, I1, O **internal port** names are generated by Vivado software

13

5.4 VHDL Data Types



14

5.4 VHDL Data Types

Selected Data types:

- 1) Predefined types
- 2) User-defined types

Type	Description
bit	'0' or '1'
boolean	FALSE or TRUE
integer	an integer in the range -2^{31} to $+(2^{31} - 1)$
natural	integer in the range 0 to $+(2^{31} - 1)$
positive	integer in the range 1 to $+(2^{31} - 1)$
real	floating-point number in the range $-1.0E38$ to $+1.0E38$
character	any legal ASCII character including upper and lowercase letters, digits, and special characters
time	An integer with units fs, ps, ns, us, ms, sec, min, or hr

15

5.4 VHDL Data Types

A physical type allows user to define measurement units for some physical quantity, like length, time, pressure, capacity, etc.

```
-- EXAMPLE : physical type definitions

type Distance is range 0 to INTEGER'HIGH
units
  micron;
  millimetre = 1000 micron;
  centimetre = 10 millimetre;
  metre = 100 centimetre;
  inch = 25400 micron;
  foot = 12 inch;
  yard = 3 foot;
  mile = 5280 foot;
end units Distance;
```

AFTER the type **Distance** defined, "inch", "millimetre", "Distance" can be recognized by the VHDL design tool and we can declare signal Dist2 as following:

```
-- EXAMPLE : signal declaration and calculation

signal Dist2 : Distance := 2 inch - 1
millimetre;
```

16

5.4 VHDL Data Types

Let's begin with simple things first.

Signal declaration syntax:

```
signal <name> : <type>;
```

Signal declaration examples for architecture:

```
signal x1 : bit;  
signal C  : bit_vector (3 to 0);  
signal D  : bit_vector (7 downto 0);
```

bit & bit_vector are predefined in VHDL standards IEEE library.

bit: a bit object is either of value 0 or 1.

bit_vector: represent multi-bit data by an array of bit objects

17

5.4 VHDL Data Types - bit / bit_vector

```
signal x1 : bit;  
signal C  : bit_vector (3 to 0);  
signal D  : bit_vector (7 downto 0);
```

Signal assignment examples:

```
x1 <= '1';  
C  <= "1010";  
D  <= "11001010";
```

- Signal represent physical interconnect (wire) that communicate between processes/modules
- Signal assignment operator "**<=**"
- You can assign '1' to a single bit value, and "1010" to an array of bit values.
- You can use (3 to 0) or (7 downto 0) to index the array values.

18

5.4 VHDL Data Types - bit_vector

Data array is assigned as bit vector in VHDL.
Here is a data array **A**:

A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]
------	------	------	------	------	------	------	------

```
signal C: bit_vector (0 to 3);  
signal D: bit_vector (3 downto 0);  
signal A: bit_vector (7 downto 0);
```

```
C <= "1101";  
D <= C;  
A(6 downto 3) <= D;
```

```
C(0)=1 C(1)=1 C(2)=0 C(3)=1  
D(3)=1 D(2)=1 D(1)=0 D(0)=1  
A(6)=1 A(5)=1 A(4)=0 A(3)=1
```

19

5.4 VHDL Data Types - (Number type)

VHDL	Decimal equivalent	Bit Pattern
B"101"	$1 \times 2^2 + 0 \times 2 + 1 = 5$	101
X"101"	$1 \times 16^2 + 0 \times 16 + 1 = 257$	100000001
101	101	1100101

These numeric values are referred to as scalars or literals. Binary (prefix of B), Hex (prefix of X), Decimal (no prefix).

A signal may optionally be declared with an initial value:

Syntax: **signal** <name> : <type> := <initial_value>;

Example:

```
signal d : bit_vector(15 downto 0) := X"1234";
```

:= ← is signal initialization operator

Bit array or bit vector is assigned to a signal (d) with hexadecimal value of **X"1234"**.

20

5.4 VHDL Data Types - std_logic /std_logic_vector

Standard Library defined by IEEE

```
library ieee;  
use ieee.std_logic_1164.all;
```

Type	Description
U	uninitialized. This signal hasn't been set yet
X	unknown. Impossible to determine this value/result
0	logic 0
1	logic 1
Z	High Impedance (Tri-state)
W	Weak signal, can't tell if it should be 0 or 1
L	Weak signal that should probably go to 0
H	Weak signal that should probably go to 1
-	Don't care

21

5.4 VHDL Data Types - Logic signal

Two multi-bit signals can be AND-ed to form an output signal. They all have the same bit length.

```
SIGNAL A : BIT_VECTOR (7 DOWNTO 0);  
SIGNAL B : BIT_VECTOR (7 DOWNTO 0);  
SIGNAL S : BIT_VECTOR (7 DOWNTO 0);  
BEGIN  
    S <= A and B;  
END;
```

BIT_VECTOR:
With well defined
logic value [0 / 1]

```
SIGNAL A : STD_LOGIC_VECTOR (7 DOWNTO 0);  
SIGNAL B : STD_LOGIC_VECTOR (7 DOWNTO 0);  
SIGNAL S : STD_LOGIC_VECTOR (7 DOWNTO 0);  
BEGIN  
    S <= A and B;  
END;
```

STD_LOGIC_VECTOR:
logic output [0 / 1 / Z
/ U / etc...] as shown
in previous slide

22

5.5 VHDL Operator - **Boolean Operators**

Logical operation	Operator	Example
AND	AND	Z <= (A AND B);
NAND	NAND	Z <= (A NAND B);
NOR	NOR	Z <= (A NOR B);
NOT	NOT	Z <= NOT (A);
OR	OR	Z <= (A OR B);
XNOR	XNOR	Z <= (A XNOR B);
XOR	XOR	Z <= (A XOR B);

VHDL provides predefined operators for modeling hardware units.
Logical (Boolean), Arithmetic, Relational Operators

23

5.5 VHDL Operator - **Relational Operators**

Relational operation	Operator	Example
Equal to	=	If (A = B) Then
Not equal to	/=	If (A /= B) Then
Less than	<	If (A < B) Then
Less than or equal to	<=	If (A <= B) Then
Greater than	>	If (A > B) Then
Greater than or equal to	>=	If (A >= B) Then

VHDL provides predefined operators for modeling hardware units.
Logical (Boolean), Arithmetic, Relational Operators

24

5.5 VHDL Operator - Arithmetic Operators

This part will be covered in the later chapter.

Arithmetic operation	Operator	Example
Addition	+	$Z \leq A + B;$
Subtraction	-	$Z \leq A - B;$
Multiplication	*	$Z \leq A * B;$
Division	/	$Z \leq A / B;$
Exponentiating	**	$Z \leq 4 ** 2;$
Modulus	MOD	$Z \leq A \text{ MOD } B;$
Remainder	REM	$Z \leq A \text{ REM } B;$
Absolute value	ABS	$Z \leq \text{ABS } A;$

25

5.5 VHDL Operator - Arithmetic Operators

REM:

$5 \text{ rem } 3 = 2 \rightarrow 5/3 = 1 \text{ remainder is } 2$

$(-5) \text{ rem } 3 = -2$

$5 \text{ rem } (-3) = 2$

$(-5) \text{ rem } (-3) = -2$

MOD:

$5 \text{ mod } 3 = 2$

$(-5) \text{ mod } 3 = 1 \leftarrow \text{different from REM}$

$5 \text{ mod } (-3) = -1 \leftarrow \text{different from REM}$

$(-5) \text{ mod } (-3) = -2$

$$5 = 1*3 + 2$$

$$-5 = -2*3 + 1$$

$$5 = -2(-3) - 1$$

$$-5 = 1*(-3) - 2$$

Then what is MOD used for?

The mod operator gives the residue for a division to round down.

26

5.5 Precedence of VHDL Operators

VHDL Operators:

1. Binary logical operators: and, or, nand, nor, xor, xnor
2. Relational operators: =, /=, <, <=, >, >=
3. Shift operators: sll, srl, sla, sra, rol, ror
4. Adding operators: +, -, & (concatenation)
5. Unary sign operators: +, -
6. Multiplying operators: *, /, mod, rem
7. Miscellaneous operators: not, abs, **

Sll = Shift Left Logical Srl = Shift Right Logical Sra = Shift Right Arithmetic rol = Rotate Right logical

Class 7 has highest precedence, then 6, 5, 4, ...

Operators in the same class are applied left to right

Parentheses change the order of precedence

27

5.5 VHDL Operators - Example

Operators with the highest precedence are applied first, without the presence of parentheses.

Example:

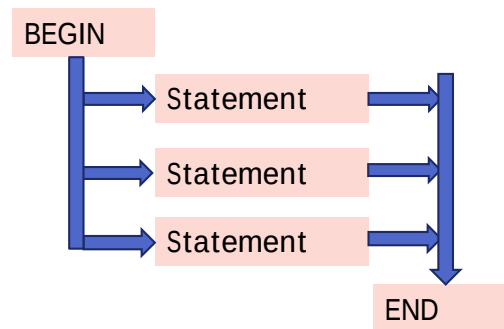
```
f <= a AND NOT(b) OR NOT(a) AND b;  
ab' + a'b
```

```
f <= a AND (NOT(b) OR NOT(a)) AND b;  
a(b' + a')b
```

28

5.6 VHDL Syntax for Boolean Expressions

- VHDL code describes the circuit design with Boolean expressions.
- It is different from computer program which is composed of a sequence of instructions for CPU to execute.
- VHDL Boolean expressions are **statements only for configuring the FPGA chip (no CPU to execute these statements)**.



29

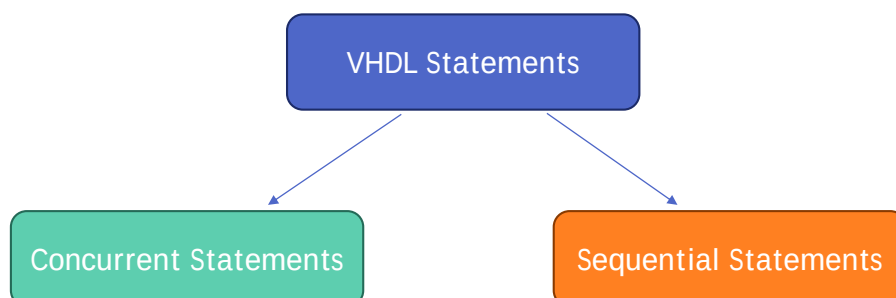
5.6 VHDL Syntax for Boolean Expressions

VHDL models combinational circuit using concurrent statements.

Concurrent statements:

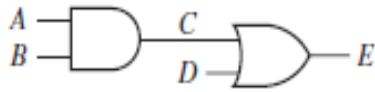
A change(s) at the input(s) will cause the output change accordingly.

Sequential statements will be covered later.



30

5.6 VHDL Syntax for Boolean Expressions



```
C <= A and B after 5 ns;
E <= C or D after 5 ns;
```

Concurrent statements (continuous assignments) example:
A signal assignment statement has the form:

```
signal_name <= expression [after delay];
```

Square bracket is optional and it is for indicating the propagation delay.

Signal on the left is updated after delay, if included.

31

5.6 VHDL Syntax for Boolean Expressions

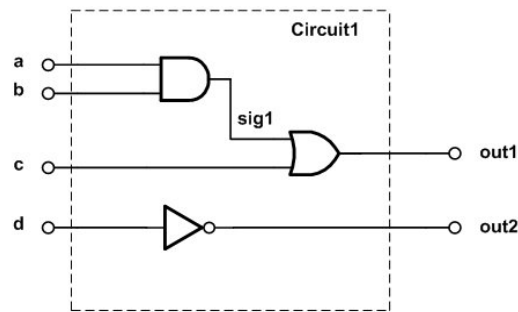
Examples of Boolean expressions converted into VHDL Boolean expressions.

Boolean	VHDL Boolean
$Y = \overline{AB}$	Y <= NOT(A AND B); or Y <= A NAND B;
$Y = \overline{A + B}$	Y <= NOT(A OR B); or Y <= A NOR B;
$Y = A + BC$	Y <= A OR (B AND C);
$Y = C\overline{X} + \overline{D}$	Y <= C AND NOT (X OR D);
$Y = A\overline{B}C + \overline{A}\overline{B}C + \overline{A}B\overline{C}$	Y <= (A AND NOT B AND C) OR (NOT A AND NOT B AND C) OR (NOT (A AND B) AND C);

32

5.6 VHDL Syntax for Logic Circuit

Example of 3 concurrent statements:



Precedence order is NOT important for concurrent statements !!

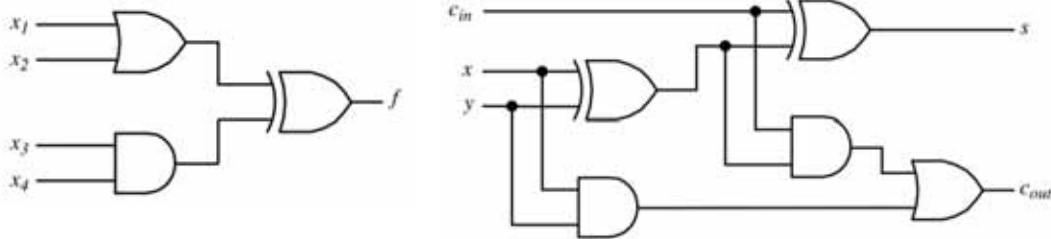
```
sig1 <= (a and b);
out1 <= (sig1 or c);
out2 <= (not d);
```

≡

```
out1 <= (sig1 or c);
sig1 <= (a and b);
out2 <= (not d);
```

33

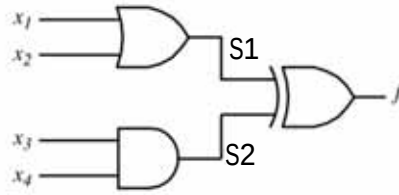
5.6 VHDL Syntax for Logic Circuit - Exercise



Write your own VHDL statement to describe the above two logic circuits.

34

5.6 VHDL Syntax for Logic Circuit - Exercise



```
S1 <= x1 OR x2;
S2 <= x3 AND x4;
F <= S1 XOR S2;
```

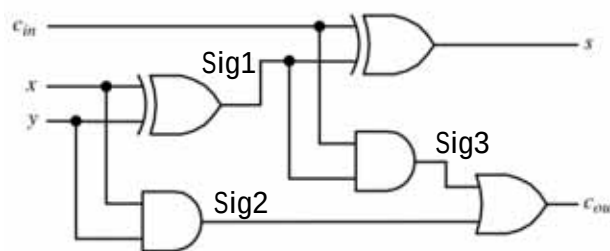
OR

```
F <= (x1 OR x2) XOR (x3 AND x4);
```

- Left hand style is more readable and traceable but need to declare two more signals.
- In contrast, Right hand style use less lines of code but a bit difficult for debugging.
- They both have same result after synthesize.

35

5.6 VHDL Syntax for Logic Circuit - Exercise



```
Sig1 <= x XOR y;
Sig2 <= x AND y;
Sig3 <= Sig1 AND cin;
s <= cin XOR Sig1;
cout <= Sig3 OR Sig2;
```

36

5.7 VHDL Module

A Logic circuit consists of one or multiple modules.

Each module consists of only one **entity** with **architecture(s)**.

Entity description declares the input and output signals.

Each ENTITY has a unique **name** !!

Port declaration specifies the input signals and output signals for the module.

```
ENTITY name IS  
  PORT( ... );  
END name;
```

37

5.8 VHDL Format and Syntax - Example

```
ENTITY and_gate IS  
  Port (  
    A : in  STD_LOGIC;  
    B : in  STD_LOGIC;  
    S : out STD_LOGIC  
  );  
END and_gate;
```

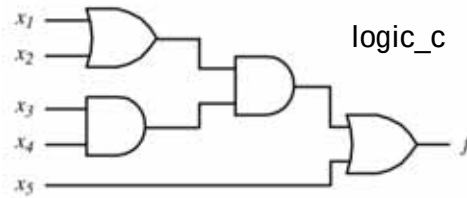
Mode indicates the direction (**in**, **out**, and **inout**) of port signals

Type specifies the data type (**bit**, **bit_vector**) that can be communicated

38

5.7 VHDL Modules - Entity Example

Example for a circuit named logic_c :

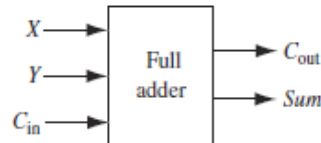


```
entity logic_c is
  port(
    x1,x2,x3,x4,x5 : in std_logic;
    f : out std_logic
  );
end logic_c;
```

39

5.7 VHDL Modules - Entity Example

Example for a Full Adder:



```
entity FullAdder is
  port(
    X, Y, C_in   : in std_logic;
    C_out, Sum   : out std_logic
  );
end FullAdder;
```

In this example, the VHDL assignment statements for Sum and C_{out} represent the logic expressions for the full adder. Sum and C_{out} are described by architectures.

40

5.8 VHDL Format and Syntax - Example

Every ENTITY must at least contain one ARCHITECTURE associated with it.

Each ARCHITECTURE has a unique **name**.

```
ARCHITECTURE ckt OF and_gate IS
BEGIN
    y <= a AND b;
END ckt;
```

41

5.8 VHDL Format and Syntax - Complete Module Code

```
library ieee;
use ieee.std_logic_1164.all;

ENTITY and_gate IS
PORT (    a,b    :IN  STD_LOGIC;
        s        :OUT STD_LOGIC)
    ;)
END and_gate;

ARCHITECTURE ckt OF and_gate IS
BEGIN
    s <= a AND b;
END ckt;
```

42

* Example

Write a complete VHDL design module to implement a circuit with the following Boolean expressions. Assign a signal name sigW1 to represent the common logic term in your design.

$$A = (XYZ')' + XZ$$

$$B = (XYZ')'(X + Z)$$

$$C = ((XYZ')' + X')'$$

43

* Example

```
ENTITY Prob_21 IS
    PORT
    (
        X, Y, Z : IN BIT;
        A, B, C : OUT BIT
    );
END Prob_21;

ARCHITECTURE arc_21 OF Prob_21 IS
    SIGNAL sigW1 : BIT;
    BEGIN
        sigW1 <= X NAND (Y NAND NOT Z);
        A <= sigW1 OR (X AND Z);
        B <= sigW1 AND (X AND Y);
        C <= sigW1 NOR NOT X;
    END arc_21;
```

$A = (XYZ')' + XZ$
 $B = (XYZ')'(X + Z)$
 $C = ((XYZ')' + X')'$

44

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL; -- Octal decoder with enable --
                                -- using IF-THEN-ELSE      --
                                -- and CASE statement        --

ENTITY decoder_d IS
    PORT(en : IN    std_logic;
          a  : IN    STD_LOGIC_VECTOR (2 downto 0);
          y  : OUT   STD_LOGIC_VECTOR (7 downto 0));
END decoder_d;

ARCHITECTURE arc OF decoder_d IS
    BEGIN
        PROCESS (a,en) -- sensitivity list
        BEGIN
            IF (en='1') -- Enable must be HIGH to
                        -- execute the THEN clause
            THEN
                CASE a IS
                    WHEN "000" =>y<="00000001"; -- when a = 000,
                    WHEN "001" =>y<="00000010"; -- y = 00000001
                    WHEN "010" =>y<="00000100";
                    WHEN "011" =>y<="00001000";
                    WHEN "100" =>y<="00010000";
                    WHEN "101" =>y<="00100000";
                    WHEN "110" =>y<="01000000";
                    WHEN "111" =>y<="10000000";
                    WHEN others=>y<="00000000";
                END CASE;
            ELSE y<="00000000"; -- all LOW outputs
                                -- if en is not HIGH
            END IF;
        END PROCESS;
    END arc;

```

* 3-to-8-Line Decoder

Inputs			Outputs							
A ₂	A ₁	A ₀	D ₀	D ₁	D ₂	D ₃	D ₄	D ₅	D ₆	D ₇
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

6. Components and Instantiation

- Structural or modular design of a complex system
- Starting from small module or subcircuits.
- Resources reused make easier.

Component Declaration

- In modular design, an architecture may contain multiple components.
- They need to be define globally.

```
architecture [name] ...  
[signal]  
  
    component X          -- Component X declaration  
    ...  
    end component;  
  
    component Y          -- Component Y declaration  
    ...  
    end component;  
  
begin  
    ...                -- Component instantiation  
end [name];
```

47

Full Adder Design

There are two half-adders(HAs) in a full adder, use half-adder as a component for modular design. The HA logic is:

$$s = a \oplus b \quad c = ab$$

Inputs		Outputs	
<i>a</i>	<i>b</i>	<i>c</i>	<i>s</i>
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

```
entity HA is  
port (a : in  STD_LOGIC;  
      b : in  STD_LOGIC;  
      s : out  STD_LOGIC;  
      c : out  STD_LOGIC  
      );  
end HA;  
  
architecture Behavioral of HA is begin  
s <= a xor b;  
c <= a and b;  
end Behavioral;
```

48

```

entity FA is
  port (a : in std_logic;
        b : in std_logic;
        cin : in std_logic;
        sum : out std_logic;
        cout : out std_logic
        );
end FA;

```

architecture behavior of FA is

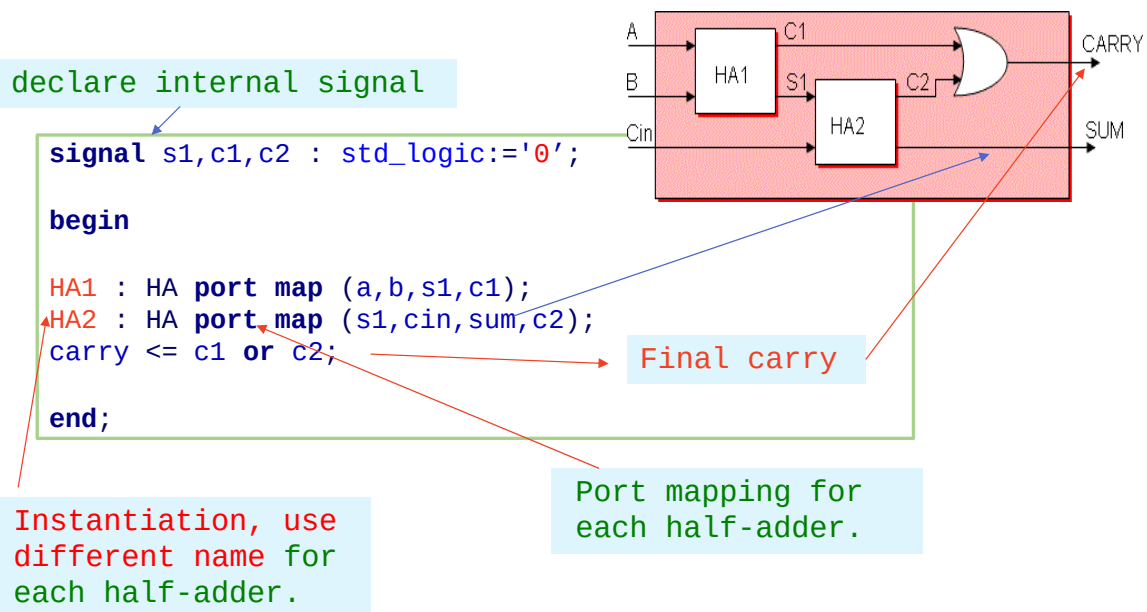
```

component HA
  port(
    a : in std_logic;
    b : in std_logic;
    sum : out std_logic;
    cout : out std_logic
  );
end component;

```

49

- Two HAs are needed to be connected in certain way.
- Use internal signals s1,c1,c2 to connect the two HAs.
- The signal pass to HAs via **port map**.



50

```

entity FA is
    port (a : in std_logic;
          b : in std_logic;
          cin : in std_logic;
          sum : out std_logic;
          cout : out std_logic
    );
end FA;

architecture behavior of FA is
    component HA
        port(
            a : in std_logic;
            b : in std_logic;
            sum : out std_logic;
            cout : out std_logic
        );
    end component;

    signal s1,c1,c2 : std_logic:='0';

begin
    HA1 : halfadder port map (a,b,s1,c1);
    HA2 : halfadder port map (s1,c1,sum,c2);
    carry <= c1 or c2;
end;

```

```

entity HA is
    port (a : in      STD_LOGIC;
          b : in      STD_LOGIC;
          s : out      STD_LOGIC;
          c : out      STD_LOGIC
    );
end HA;

architecture Behavioral of HA is
begin
    s <= a xor b;
    c <= a and b;
end Behavioral;

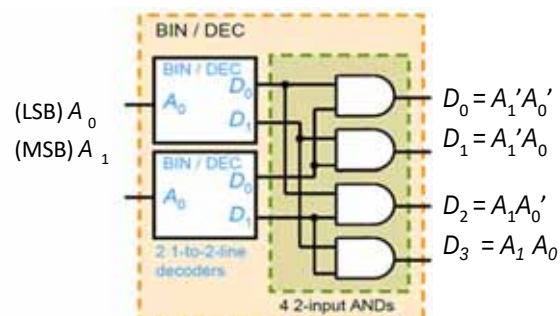
```

12

Design a 2-to-4 Line Decoder using 1-to-2 Line Decoder



Inputs		Outputs			
A_1	A_0	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



The flow:

The target module

```
43 architecture Behavioral of LDecoder2_4 is
44
45 component LDecoder1_2 is
46     Port ( A : in STD_LOGIC;
47           D0 : out STD_LOGIC;
48           D1 : out STD_LOGIC);
49 end component;
50
51 -- declare intermediate signals
52 signal SigA0: std_logic_vector(1 downto 0);
53 signal SigA1: std_logic_vector(1 downto 0);
54
55 begin
56
57 uut1: LDecoder1_2 port map(A0, SigA0(0), SigA0(1));
58 uut2: LDecoder1_2 port map(A1, SigA1(0), SigA1(1));
59
60 D0 <= SigA1(0) and SigA0(0);
61 D1 <= SigA1(0) and SigA0(1);
62 D2 <= SigA1(1) and SigA0(0);
63 D3 <= SigA1(1) and SigA0(1);
64
65 end Behavioral;
```

Pick a component



The component declaration is similar to an entity declaration.

The component declaration statement must be first declared in the architecture of a module so that the VHDL tools can check the component to have the same ports as the entity as declared.

Pick some wires for connection



The flow:

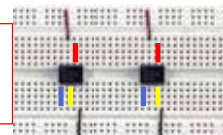
The target module

```
43 architecture Behavioral of LDecoder2_4 is
44
45 component LDecoder1_2 is
46     Port ( A : in STD_LOGIC;
47           D0 : out STD_LOGIC;
48           D1 : out STD_LOGIC);
49 end component;
50
51 -- declare intermediate signals
52 signal SigA0: std_logic_vector(1 downto 0);
53 signal SigA1: std_logic_vector(1 downto 0);
54
55 begin
56
57 uut1: LDecoder1_2 port map(A0, SigA0(0), SigA0(1));
58 uut2: LDecoder1_2 port map(A1, SigA1(0), SigA1(1));
59
60 D0 <= SigA1(0) and SigA0(0);
61 D1 <= SigA1(0) and SigA0(1);
62 D2 <= SigA1(1) and SigA0(0);
63 D3 <= SigA1(1) and SigA0(1);
64
65 end Behavioral;
```

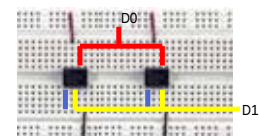
Pick a component

Component instantiation statement performance component numbering/labelling and signals name assignment such as the physical part placements and wire connections.

Place components and connecting wires to components

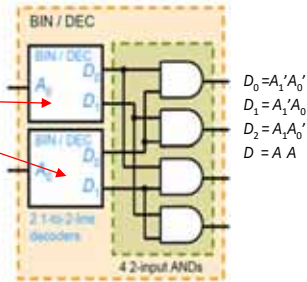


Connecting components' output to produce the circuit module output



Solution:

```
43 architecture Behavioral of LDecoder2_4 is
44
45     component LDecoder1_2 is
46         Port ( A : in STD_LOGIC;
47               D0 : out STD_LOGIC;
48               D1 : out STD_LOGIC);
49     end component;
50
51     -- declare intermediate signals
52     signal SigA0: std_logic_vector(1 downto 0);
53     signal SigA1: std_logic_vector(1 downto 0);
54
55     begin
56
57     uut1: LDecoder1_2 port map(A0, SigA0(0), SigA0(1));
58     uut2: LDecoder1_2 port map(A1, SigA1(0), SigA1(1));
59
60     D0 <= SigA1(0) and SigA0(0);
61     D1 <= SigA1(0) and SigA0(1);
62     D2 <= SigA1(1) and SigA0(0);
63     D3 <= SigA1(1) and SigA0(1);
64
65 end Behavioral;
```



Inputs		Outputs			
A ₁	A ₀	D ₀	D ₁	D ₂	D ₃
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1