# EE2000 Logic Circuit Design

Chapter 5– VHDL

## 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

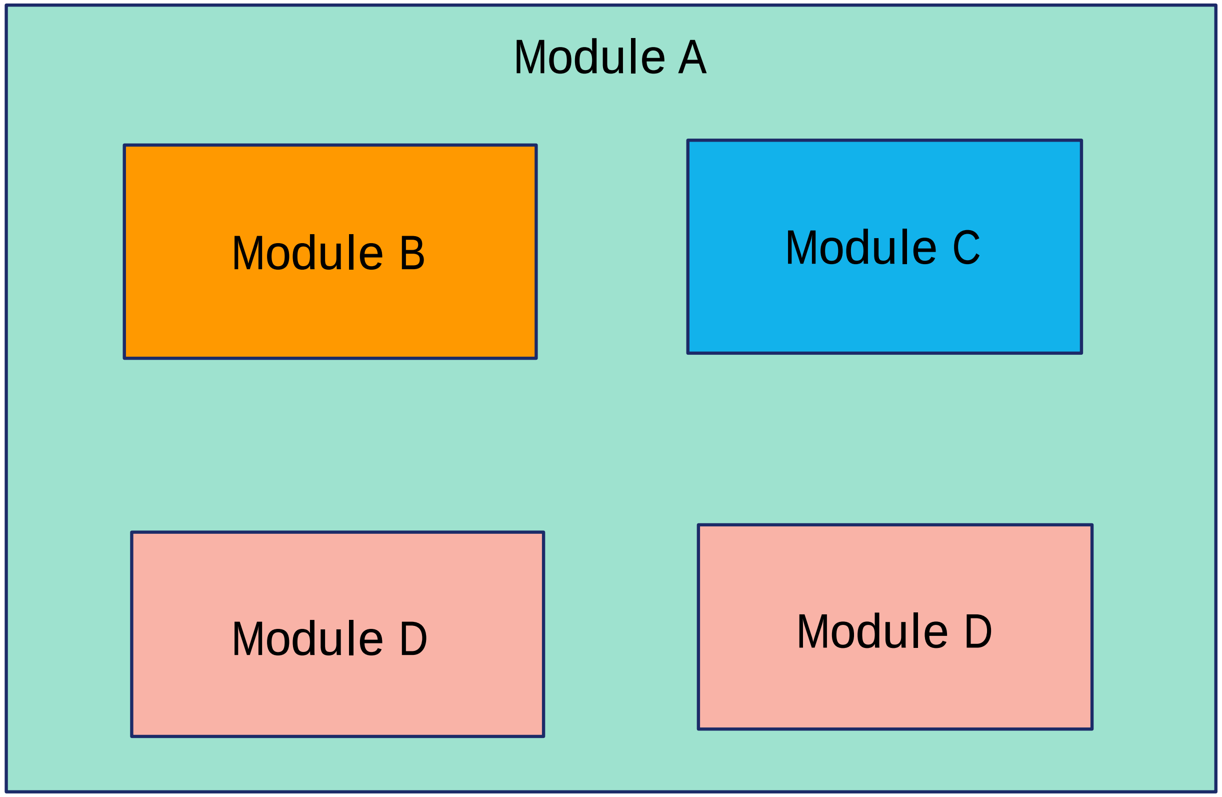
## 5.1 Hardware Description Language (HDL)

- 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.

## 5.1 Hardware Description Language (HDL)

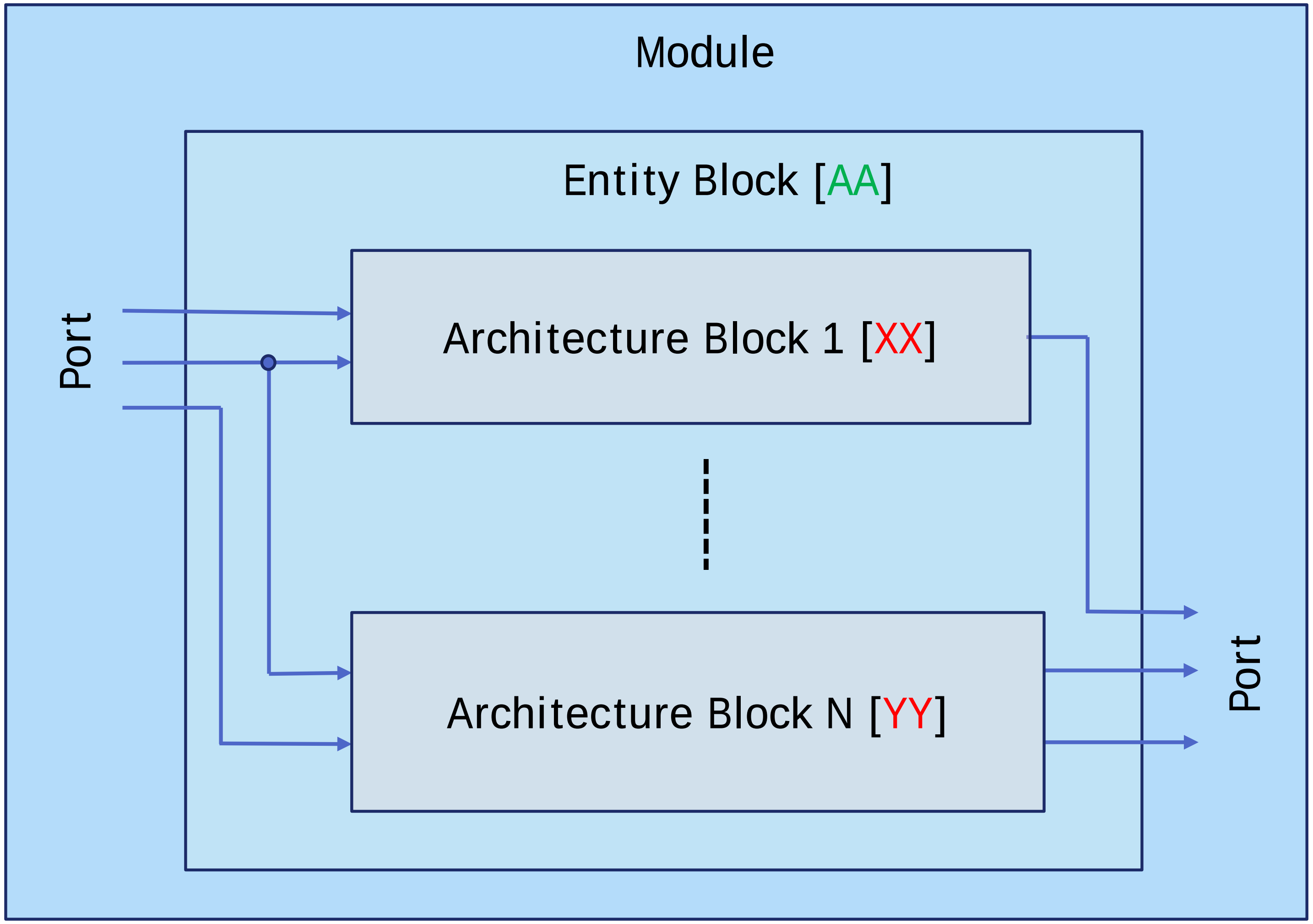
- HDL – Hardware Description Language is a software programming language that is used to model a piece of hardware
- VHDL is an HDL used to describe a digital system at several different levels: behavioral, data flow, and structural.
- VHDL leads naturally to a top-down design methodology.
- The Vivado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

## 5.2 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.

## 5.2 VHDL Code Structure - Illustration



## 5.2 VHDL Code Structure

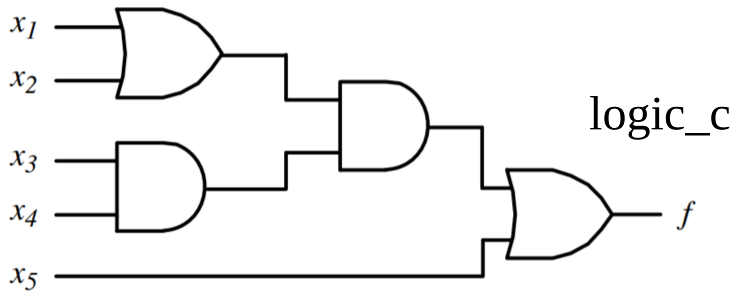
{***Library Declaration***}
{***Entity Declaration***}
{***Architecture Declaration***}

**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].

## 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;
```

## 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;
```

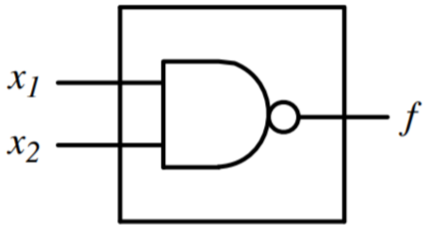
```
ARCHITECTURE YY of AA IS

BEGIN
    Functional code
END YY;
```

**AA: entity name**
**XX, YY: architecture name**

## 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;
```

## 5.3 Basic Logic Gates using VHDL

VHDL is “case insensitive”.

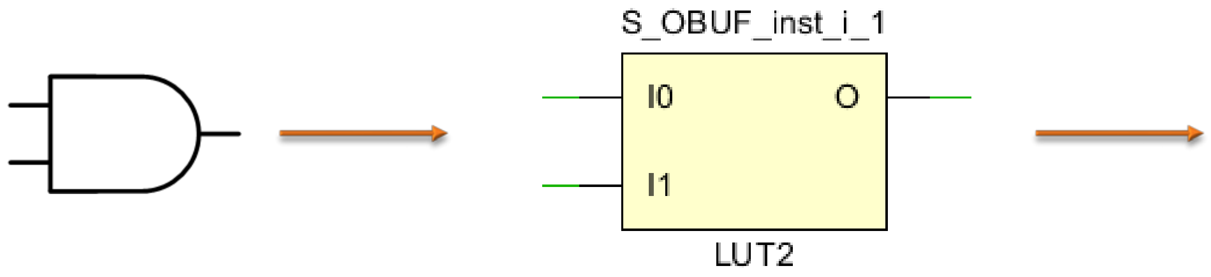
```
architecture Behavior of nand_gate is
begin
                    f <= x1 nand x2;
end Behavior;
```

The above code is identical to the following:

```
ARCHITECTURE Behavior of NAND_GATE IS
BEGIN
                    F <= x1 NAND x2;
END Behavior;
```

## 5.3 Basic Logic Gates using VHDL

VHDL language translates logic circuit into Look-up Table

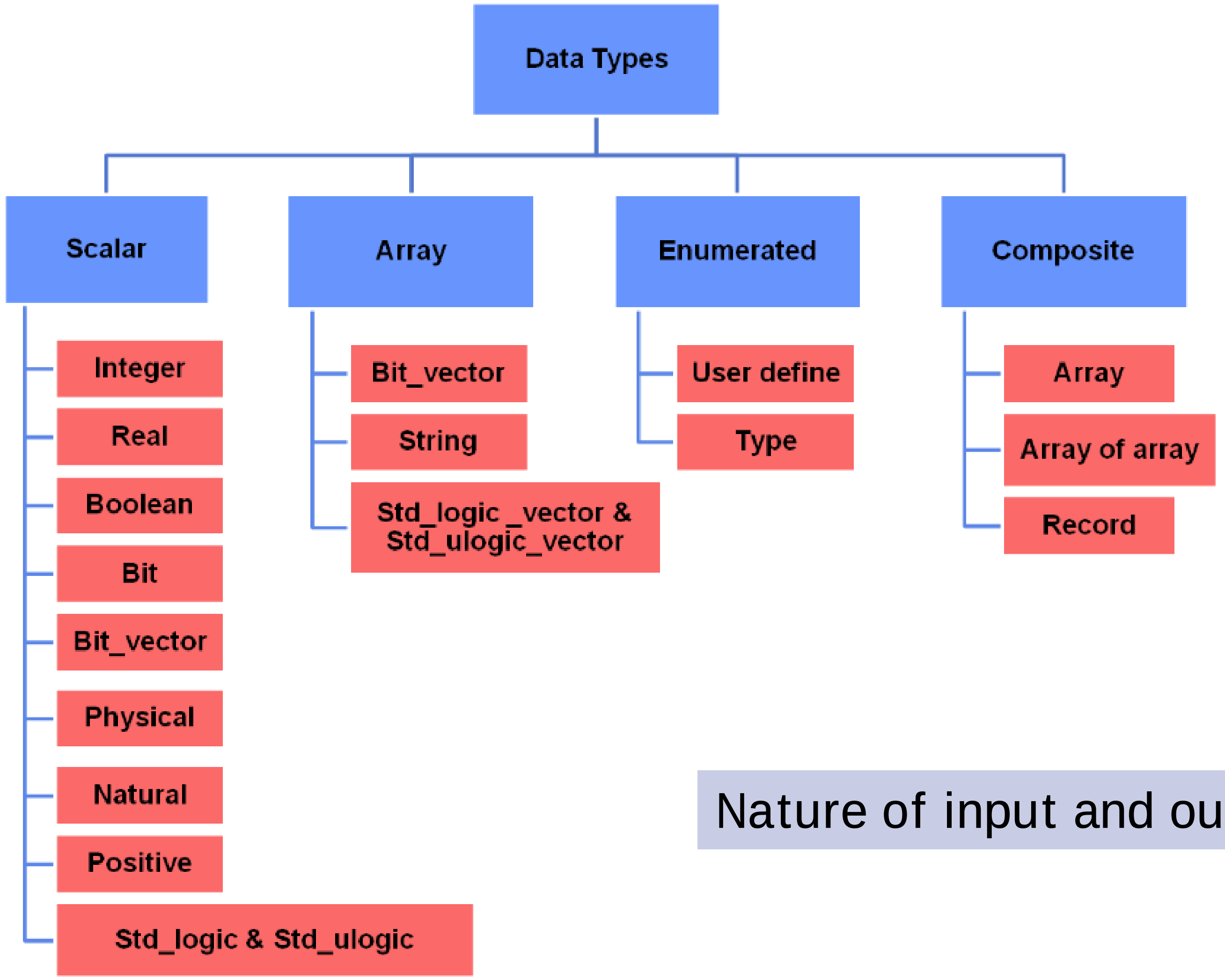


O = I0 and I1

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

## 5.4 VHDL Data Types



Nature of input and output signals

## 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 |

## 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;
```

## 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

## 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.

## 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
```

## 5.4 VHDL Data Types – (Number type)

| VHDL | Bit Pattern | Decimal equivalent |
|---|---|---|
| B”101” | 101 | 5 |
| X”101” | 100000001 | 257 |
| 101 | 1100101 | 101 |

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".

## 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 |

## 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

## 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**

## 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**

## 5.5 VHDL Operator - Arithmetic Operators

This part will be covered in the later chapter.

| Arithmetic operation | Operator | Example |
|---|---|---|
| Addition | + | `Z <= A + B;` |
| Subtraction | - | `Z <= A - B;` |
| Multiplication | * | `Z <= A * B;` |
| Division | / | `Z <= A / B;` |
| Exponentiating | ** | `Z <= 4 ** 2;` |
| Modulus | MOD | `Z <= A MOD B;` |
| Remainder | REM | `Z <= A REM B;` |
| Absolute value | ABS | `Z <= ABS A;` |

## 5.5 VHDL Operator - Arithmetic Operators

REM:

5 rem 3 = 2 → 5/3 = 1 remainder is 2

(-5) rem 3 = -2

5 rem (-3) = 2

(-5) rem (-3) = -2

MOD:

5 mod 3 = 2

(-5) mod 3 = 1 ← different from REM

5 mod (-3) = -1 ← different from REM

(-5) 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.

## 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

## 5.5 VHDL Operators - Example

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

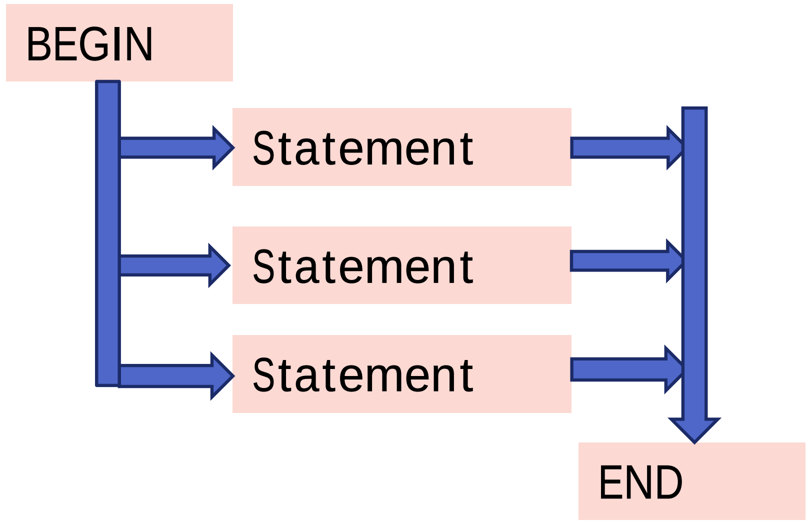
Example:

```
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
```

## 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 )**.



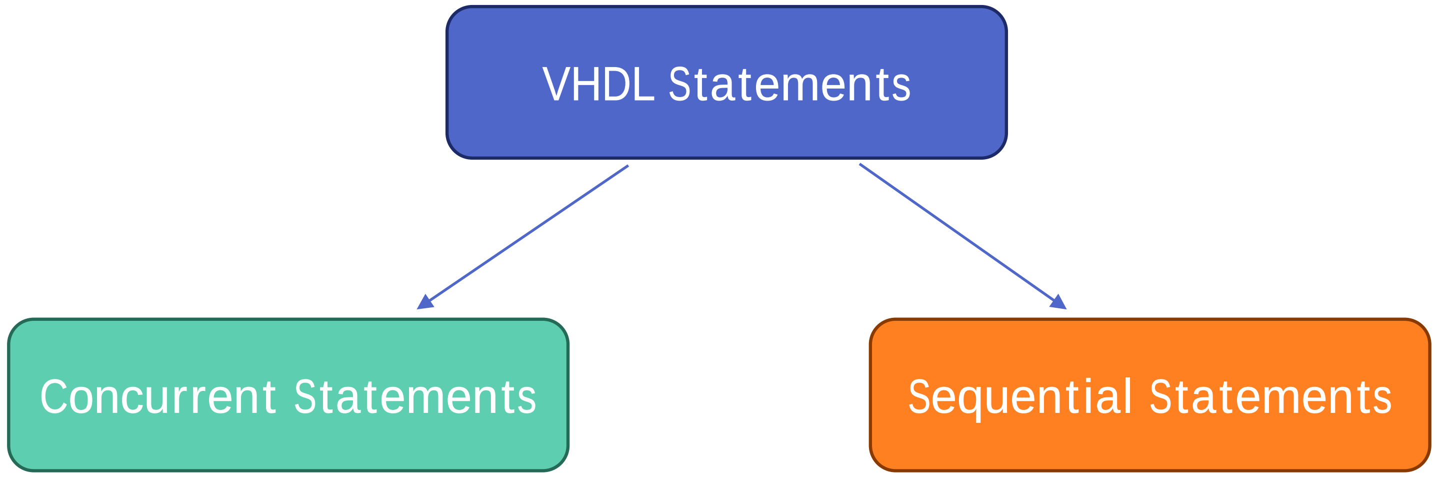
## 5.6 VHDL Syntax for Boolean Expressions

VHDL models **combinational circuit** using **concurrent statements**.

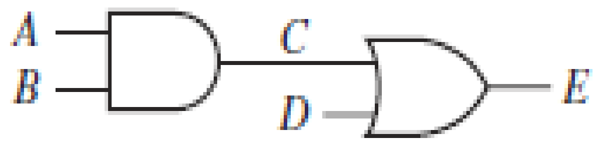
Concurrent statements:

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

*Sequential statements will be covered later.*



## 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.

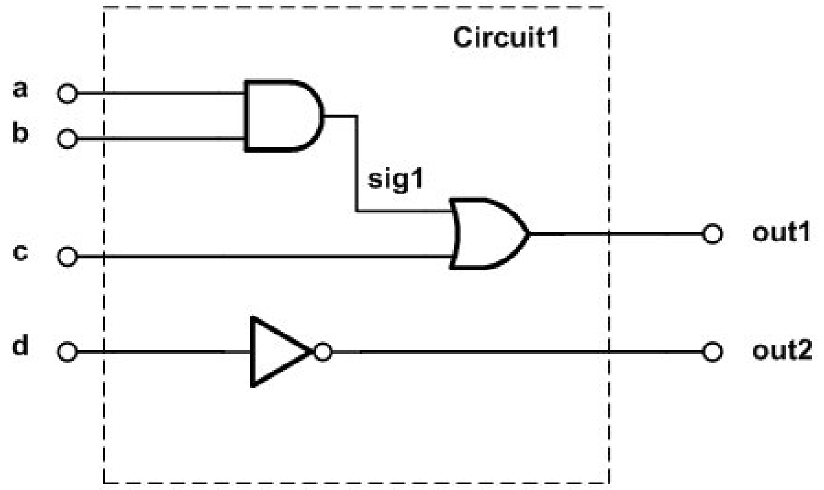
## 5.6 VHDL Syntax for Boolean Expressions

Examples of Boolean expressions converted into VHDL Boolean expressions.

| Boolean | VHDL Boolean |
|---|---|
| $Y = \overline{AB}$ | `Y <= NOT(A AND B);`<br>`  or`<br>`Y <= A NAND B;` |
| $Y = \overline{A + B}$ | `Y <= NOT(A OR B);`<br>`  or`<br>`Y <= A NOR B;` |
| $Y = A + BC$ | `Y <= A OR (B AND C);` |
| $Y = C\overline{X + D}$ | `Y <= C AND NOT (X OR D);` |
| $Y = A\bar{B}C + \bar{A}\,\bar{B}C + \overline{AB}C$ | `Y <= (A AND NOT B AND C) OR`<br>`     (NOT A AND NOT B AND C) OR`<br>`     (NOT (A AND B) AND C);` |

## 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);
```

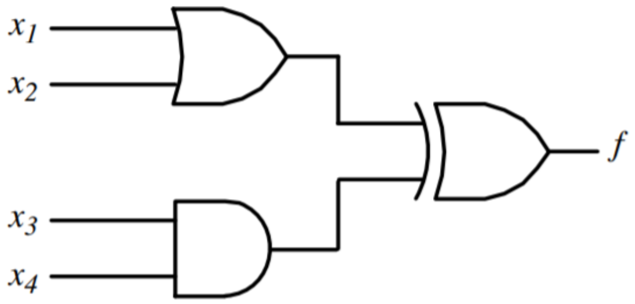
≡

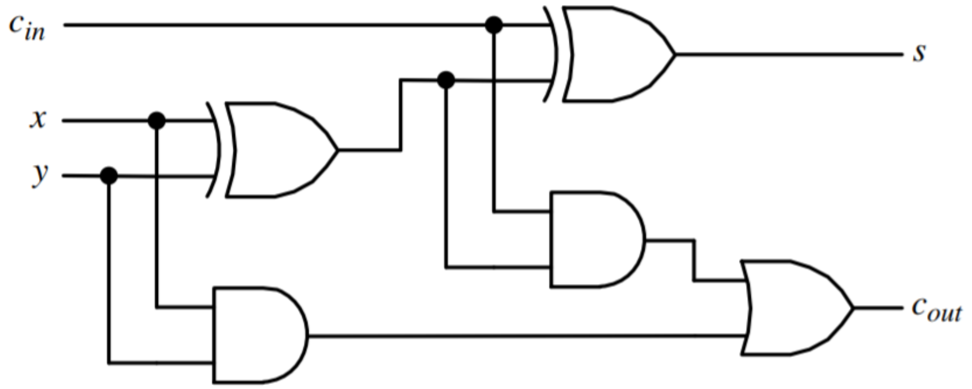
```
out1 <= (sig1 or c);

sig1 <= (a and b);

out2 <= (not d);
```

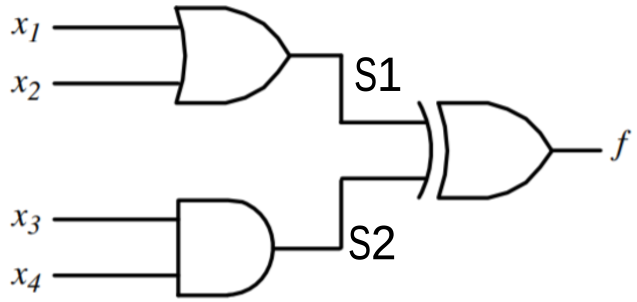
## 5.6 VHDL Syntax for Logic Circuit - Exercise





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

## 5.6 VHDL Syntax for Logic Circuit - Exercise



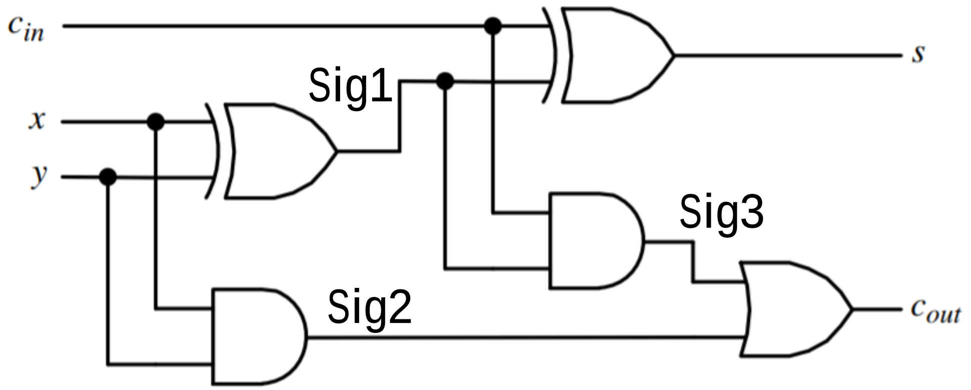
```
S1 <= x1 OR x2;
S2 <= x3 AND x4;
F <= S1 XOR S2;
```

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.

## 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;
```

## 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;
```

## 5.8 VHDL Format and Syntax - Example

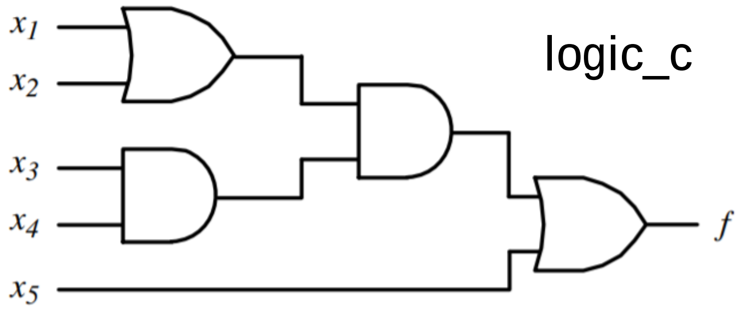
```
ENTITY and_gate IS
Port (
        A : in  STD_LOGIC;
        B : in  STD_LOGIC;
        S : out STD_LOGIC
);
END and_gate;
```

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

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

## 5.7 VHDL Modules – Entity Example

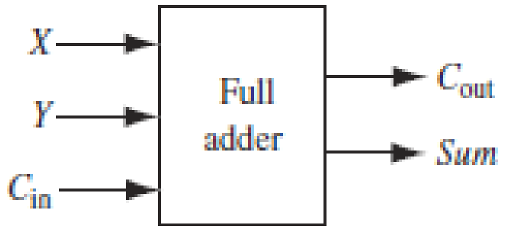
Example for a circuit named logic_c :



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

## 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.

## 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;
```

## 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;
```