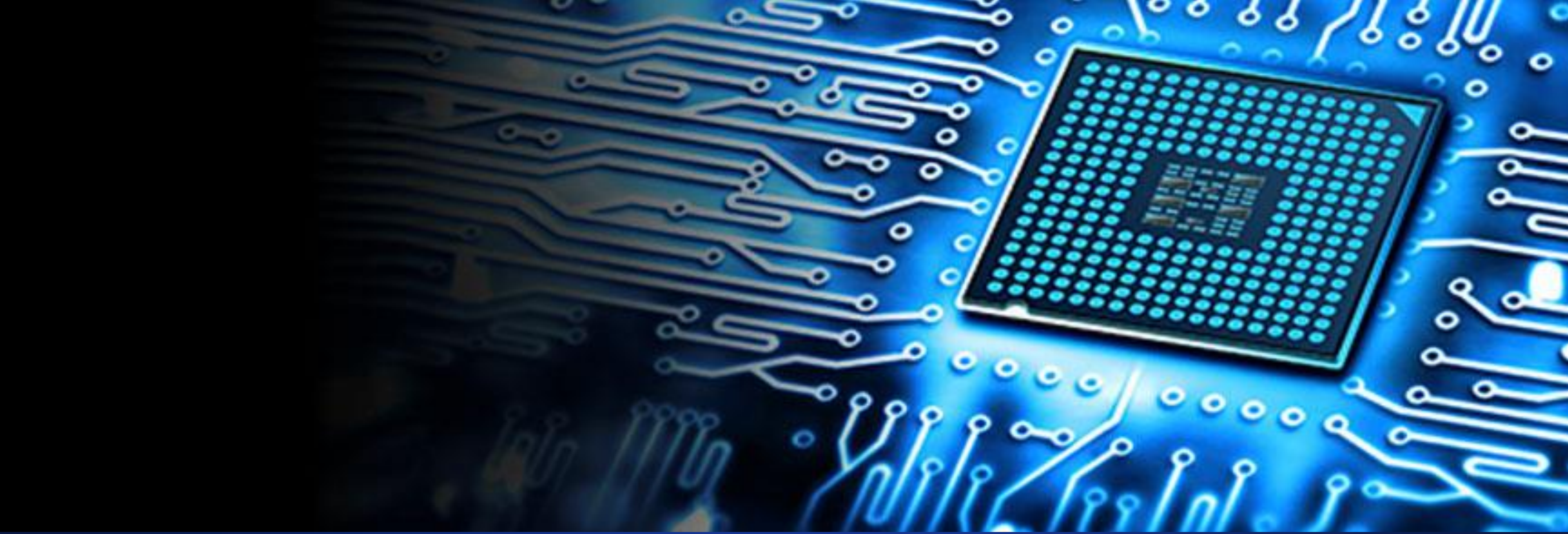


# Chapter 2.1-2.3
Introduction to VHDL

**Version: 2023/11/21**

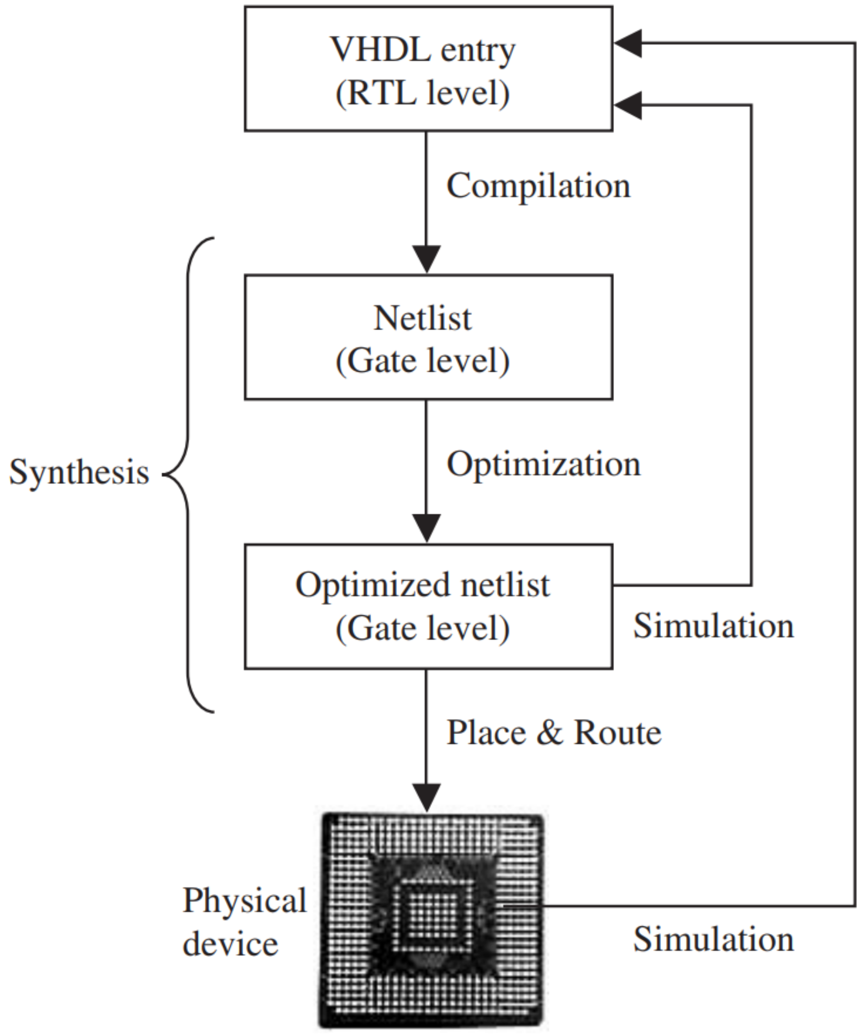
# Chapter 2 Introductin to VHDL

## 2.1 Computer-aided design

- As integrated circuit technology has improved to allow more and more components on a chip, digital systems have continued to grow in complexity
- As digital systems have become more complex, detailed design of the systems at the gate and flip-flop level has become very tedious and time-consuming
- Now, hardware design often involves no hands-on tasks with bread-boards and wires

- Computer-aided design (CAD) tools have advanced significantly in the past decade, and nowadays, digital design is performed using a variety of software tools
- Prototypes or even final designs can be created without discrete components and interconnection wires

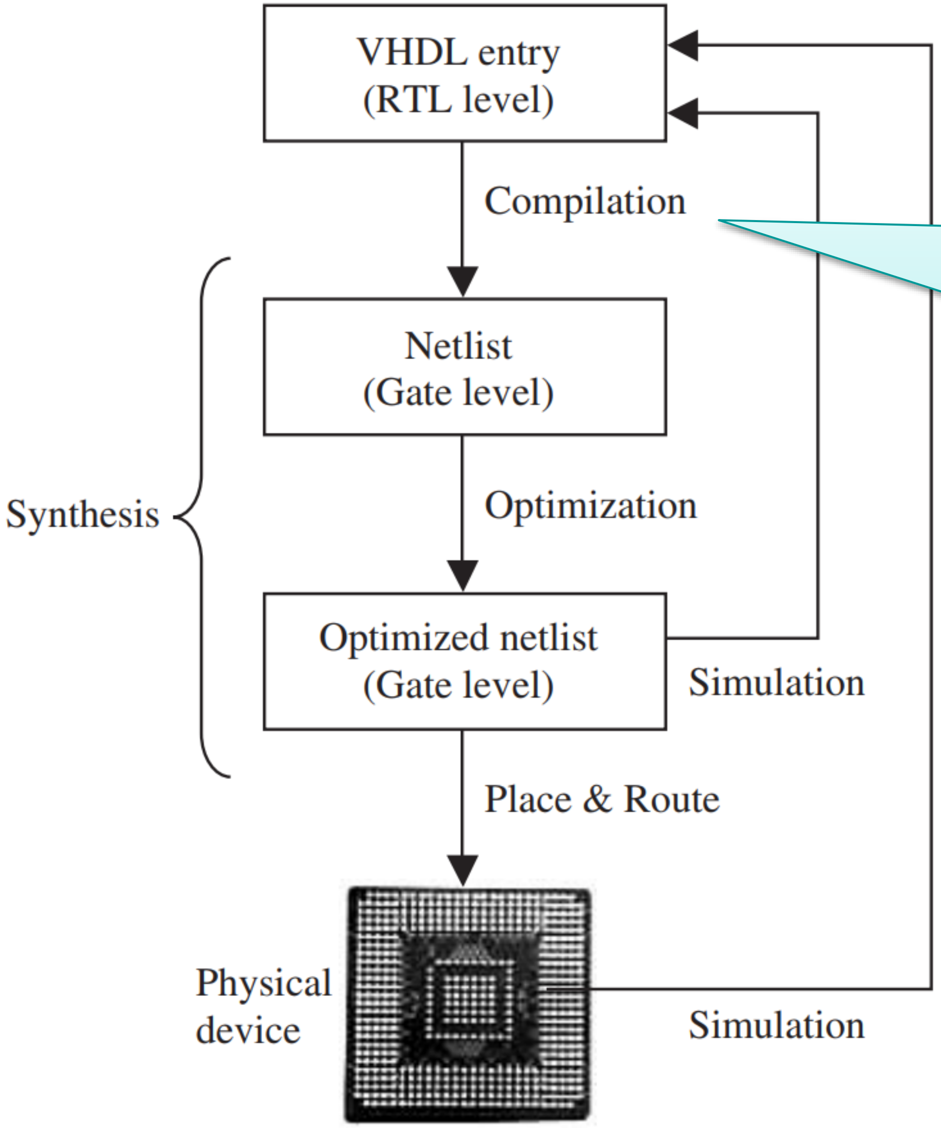
# 2.1 Computer-aided design



Writing the VHDL code

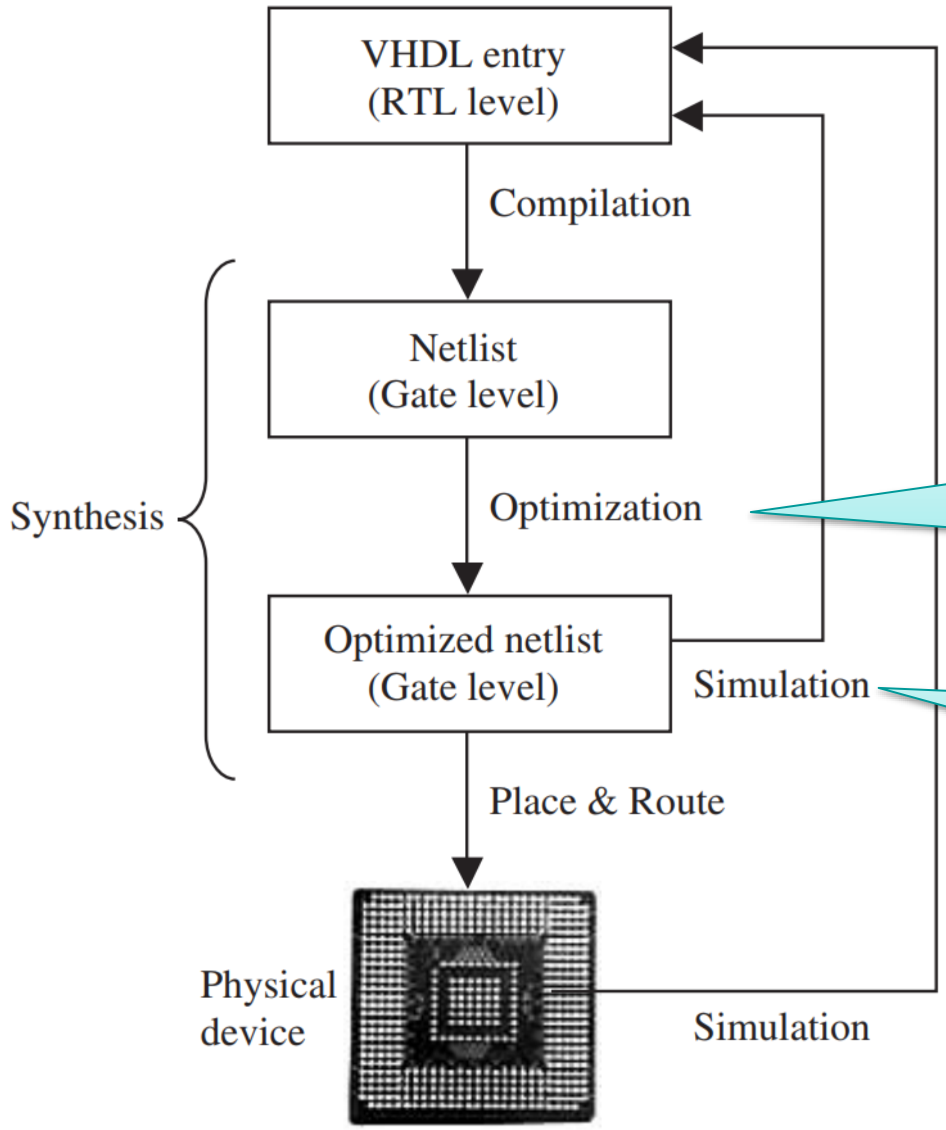
- Register Transfer Level（寄存器传输级）
- In RTL design, a circuit is described as a set of registers and a set of transfer functions describing the flow of data between the registers
- The registers are implemented directly as flip-flops, whilst the transfer functions are implemented as blocks of combinational logic

## 2.1 Computer-aided design



**Complication (编译)：** the conversion of the high-level VHDL language, which describes the circuit at the RLT into a netlist at the gate level

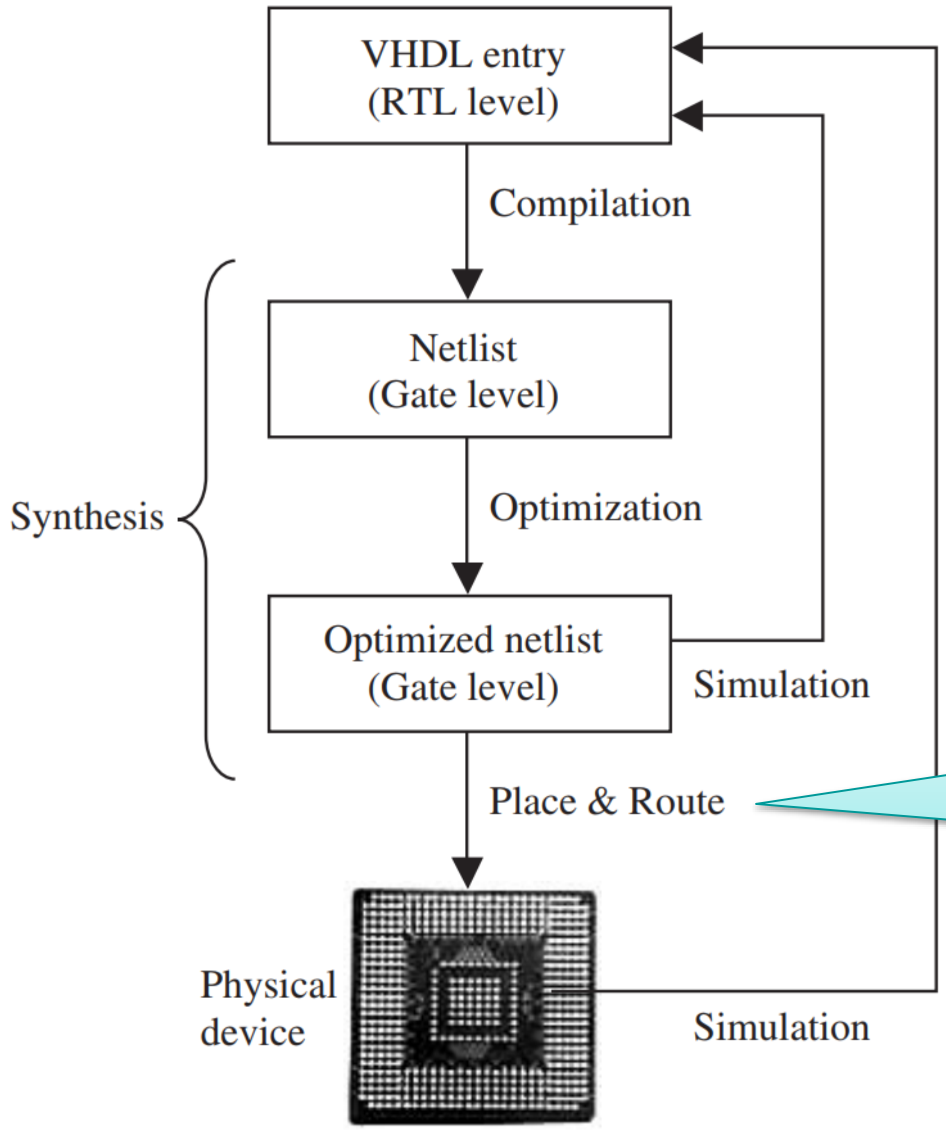
# 2.1 Computer-aided Design



**Optimization** is performed on the gate-level netlist for speed or for area

The design can be **simulated**

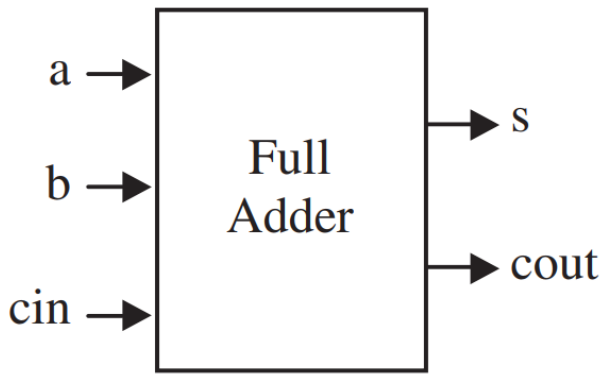
# 2.1 Computer-aided Design



Finally, a **place-and-route** software will generate the physical layout for a FPGA chip

# Translation of VHDL code into a circuit

## Example: Full-adder diagram and truth table



| a b | cin | s | cout |
|---|---|---|---|
| 0 0 | 0 | 0 | 0 |
| 0 1 | 0 | 1 | 0 |
| 1 0 | 0 | 1 | 0 |
| 1 1 | 0 | 0 | 1 |
| 0 0 | 1 | 1 | 0 |
| 0 1 | 1 | 0 | 1 |
| 1 0 | 1 | 0 | 1 |
| 1 1 | 1 | 1 | 1 |

| | |
|---|---|
| ➢ a, b | : input bits to be added |
| ➢ cin | : carry-in bit |
| ➢ s | : sum bit |
| ➢ cout | : carry-out bit |

- s = 1 whenever the number of inputs that are high is odd
- cout = 1 when two or more inputs are high

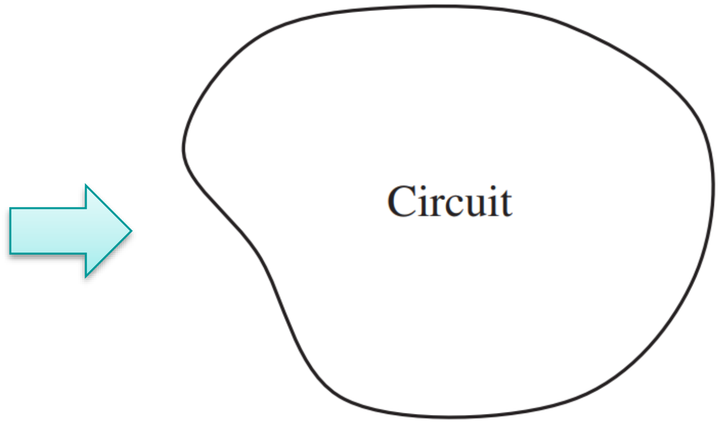
## Translation of VHDL code into a circuit

### Example of VHDL code for the full-adder unit

```
ENTITY full_adder IS
PORT (a, b, cin: IN BIT;
        s, cout: OUT BIT);
END full_adder;
--------------------------------------
ARCHITECTURE dataflow OF full_adder IS
BEGIN
   s <= a XOR b XOR cin;
   cout <= (a AND b) OR (a AND cin) OR
           (b AND cin);
END dataflow;
```

- Entity(实体) : a description of the pins (PORTS) of the circuit

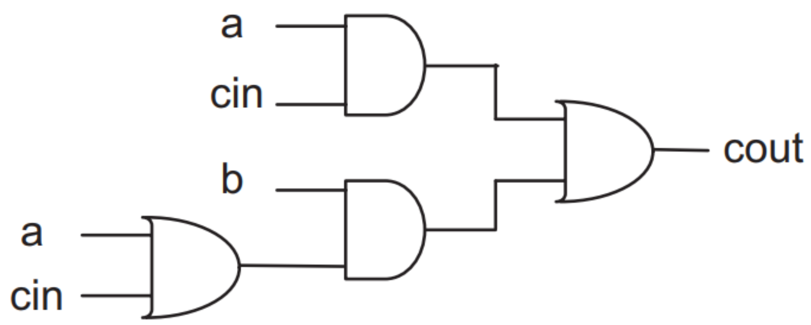
- Architecture(结构体) : description of how the circuit should function

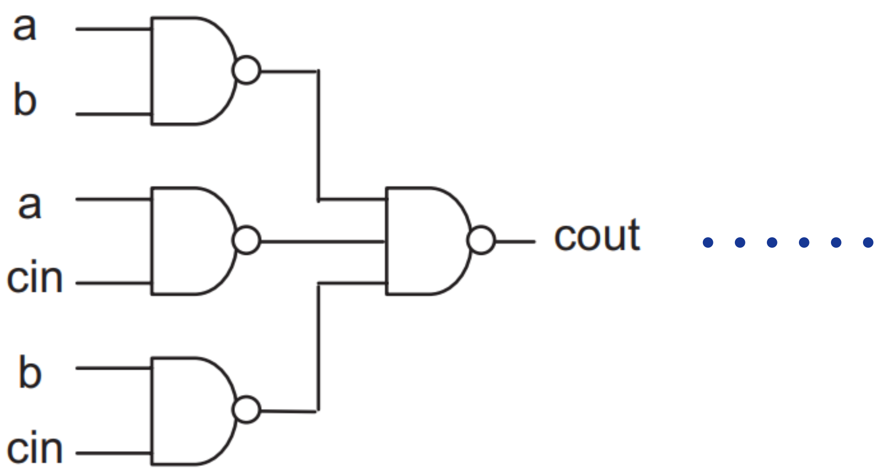


From the VHDL code, a physical circuit is inferred

# Translation of VHDL code into a circuit

## Example of possible circuits obtained from the full-adder VHDL code



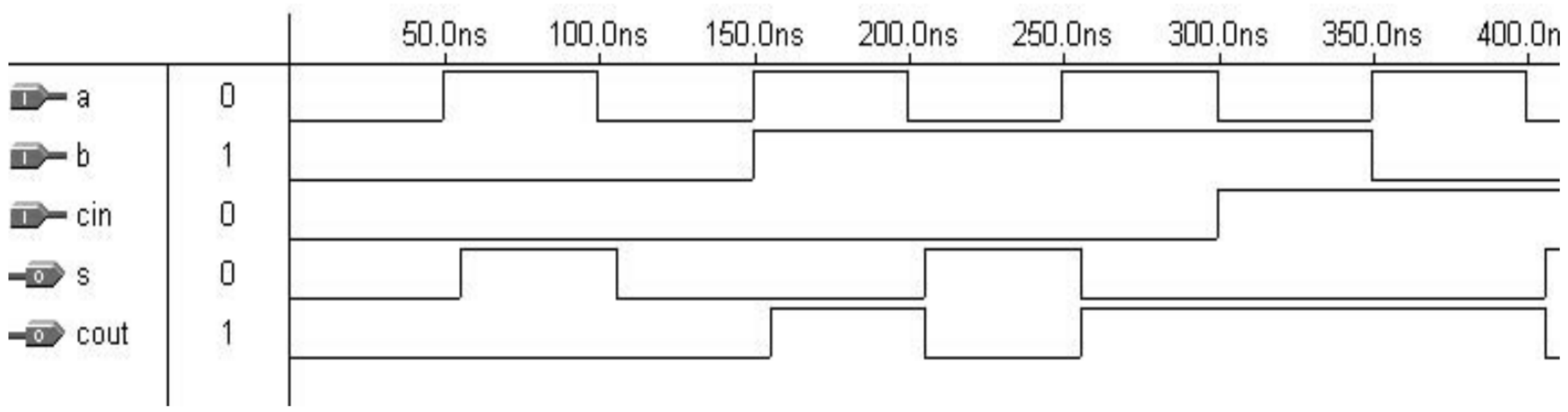


There are several ways of implementing the equations described in the Architecture

- ➢ The actual circuit will depend on the **compiler/optimizer** being used
- ➢ Synthesis tool can be set to optimize the layout for area or for speed, which obviously also affects the final circuitry

# Translation of VHDL code into a circuit

## Simulation results from the VHDL design



Whatever the final circuit inferred from the code is, its operation should always be verified

When testing, waveforms will be displayed by the simulator

# **Chapter 2** Introductin to VHDL

| | | | |
|---|---|---|---|
| 1 | Computer-aided design | 11 | Simple synthesis examples |
| 2 | **Hardware description language** | 12 | VHDL models for multiplexers |
| 3 | VHDL description of combinational circuits | 13 | VHDL libraries |
| 4 | VHDL modules | 14 | Modeling registers and counters using VHDL processes |
| 5 | Sequential statements and VHDL processes | 15 | Behavioral and structural VHDL |
| 6 | Modeling flip-flop using VHDL processes | 16 | Variables, signals, and constants |
| 7 | Processes using wait statemetns | 17 | Arrays |
| 8 | Two types of VHDL delays: Transport and inertial delays | 18 | Loops in VHDL |
| 9 | Complication,simulation, and synthesis of VHDL code | 19 | Assert and report statements |
| 10 | VHDL data types and operatos | | |

# 2.2 Hardware description languages (HDL)

## What is VHDL?

- **HDL** describes behavior (or/and structure) of an electronic circuit or system, from which the physical circuit or system can then be implemented

- **VHDL** is a general-purpose HDL that can be used to describe and simulate the operation of a wide variety of digital systems
- The systems can range in complexity from a few gates to an interconnection of many complex integrated circuits

# 2.2 Hardware description languages

## History of VHDL

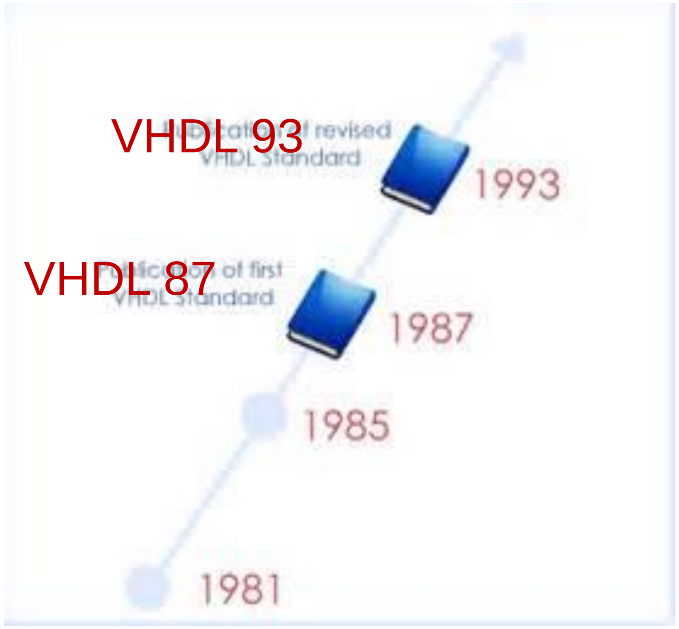
- VHDL was originally developed under funding from US-DoD (United State - Department of Defence) to allow a uniform method for specifying digital systems

- When VHDL was developed, the main purpose was to have a machanism to describe and document hardware unambiguously
- Synthesizing hardware from high-level descriptions was not one of the original purposes

## 2.2 Hardware description languages

### History of VHDL

- VHDL has since become an IEEE standard, and it is widely used in industry


VHDL 93
Publication of revised VHDL Standard
1993
VHDL 87
Publication of first VHDL Standard
1987
1985
1981

- Further revision were done to the standard in 2000 and 2002

## 2.2 Hardware description languages

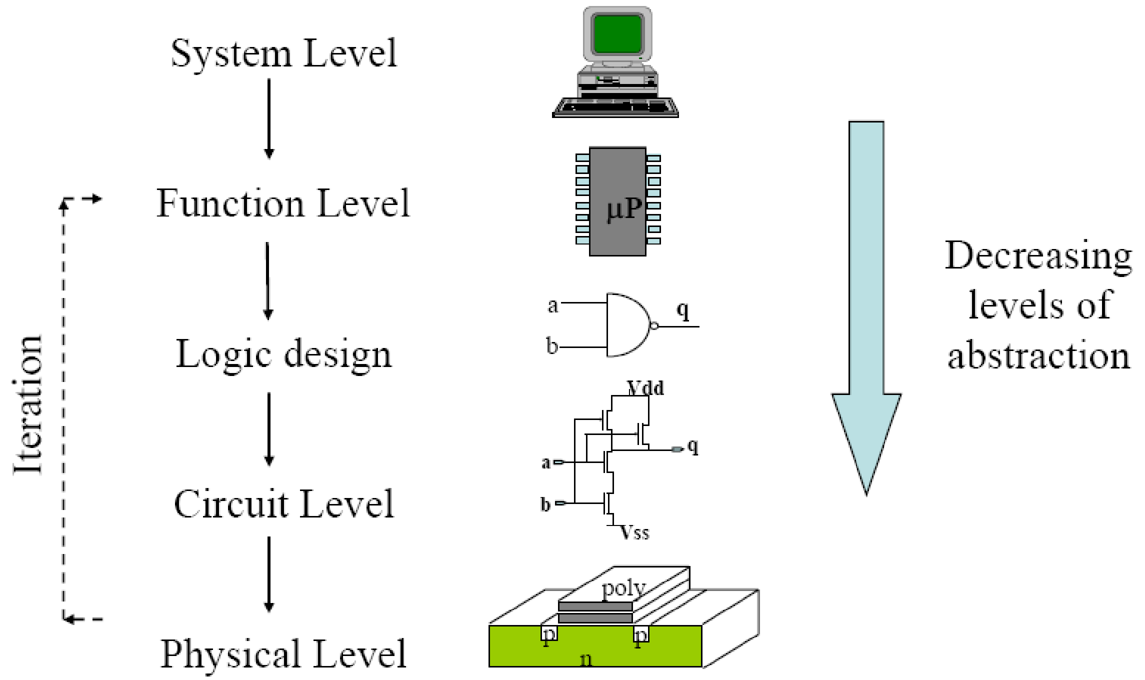
### What does VHDL can do?

VHDL can describe a digital system at several different levels

| | Using adder as an example |
|---|---|
| **Behavioral level (行为级)** | Its function of adding two binary numbers without giving any implementation details |
| **Data flow level (数据流级)** | Giving the logic equation of the adder |
| **Structural level (结构级)** | Specifying the gates and the interconnections between the gates that comprise the adder |

# 2.2 Hardware description languages

## VHDL & Top-down design



The design can gradually be refined, eventually leading to a structrual description closely related to the acutal hardware implementation

## 2.2 Hardware description languages

### VHDL: technology independent

- If a design is described in VHDL and implemented in today's technology, the same VHDL description could be used as a starting point for a design in some future technology

- Although initially conceived as a hardware documentation language, most of VHDL can now be used for simulation and logic synthesis

- VHDL is intended for circuit **synthesis** as well as circuit **simulation**
- However, though VHDL is **fully simulatable**, not all constructs are synthesizable

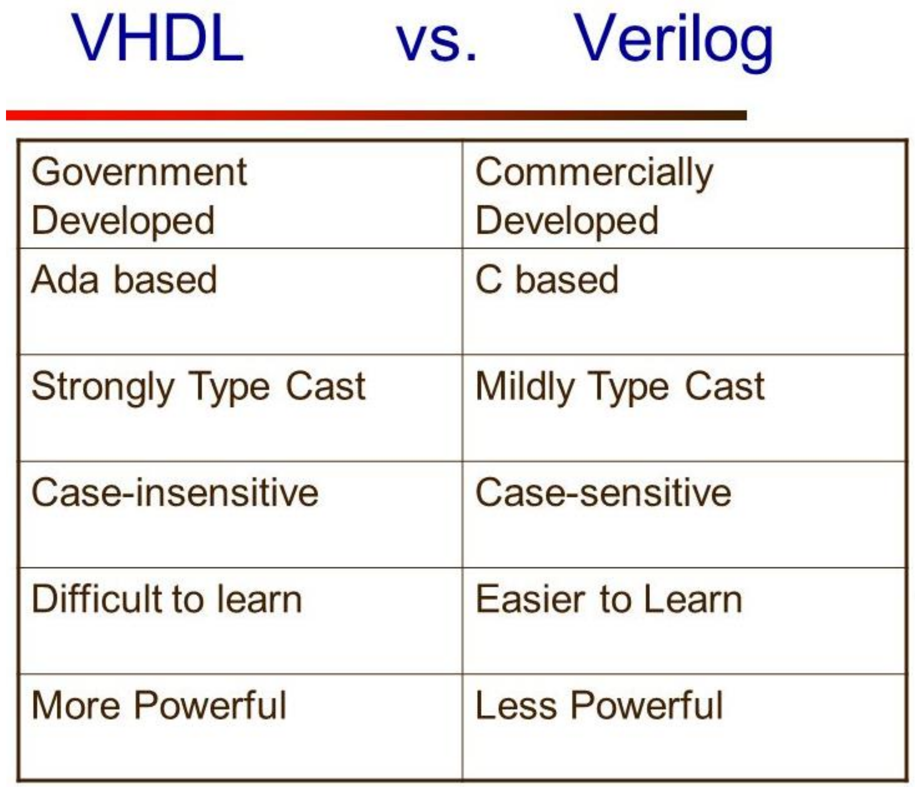
## 2.2 Hardware description languages

### Verilog

- **Verilog: Verify Logic**
- Verilog is another popular HDL
- It was developed by the industry at about the same time the U.S. DoD was funding the creation of VHDL

- Verilog was introduced by Gateway Design Automation in 1984 as a proprietary HDL
- Synopsis created synthesis tools for Verilog around 1988
- Verilog became an IEEE standard in 1995

## 2.2 Hardware description languages



VHDL vs. Verilog

| VHDL | Verilog |
|---|---|
| Government Developed | Commercially Developed |
| Ada based | C based |
| Strongly Type Cast | Mildly Type Cast |
| Case-insensitive | Case-sensitive |
| Difficult to learn | Easier to Learn |
| More Powerful | Less Powerful |

ADA was a general-purpose programming language, also sponsored by the DoD

Due to the similarity with C, some find Verilog easier or less intimidating to learn

Many find VHDL to be excellent for supporting design and documentation of large systems

If you know one of these languages, it is not difficult to transition to the other

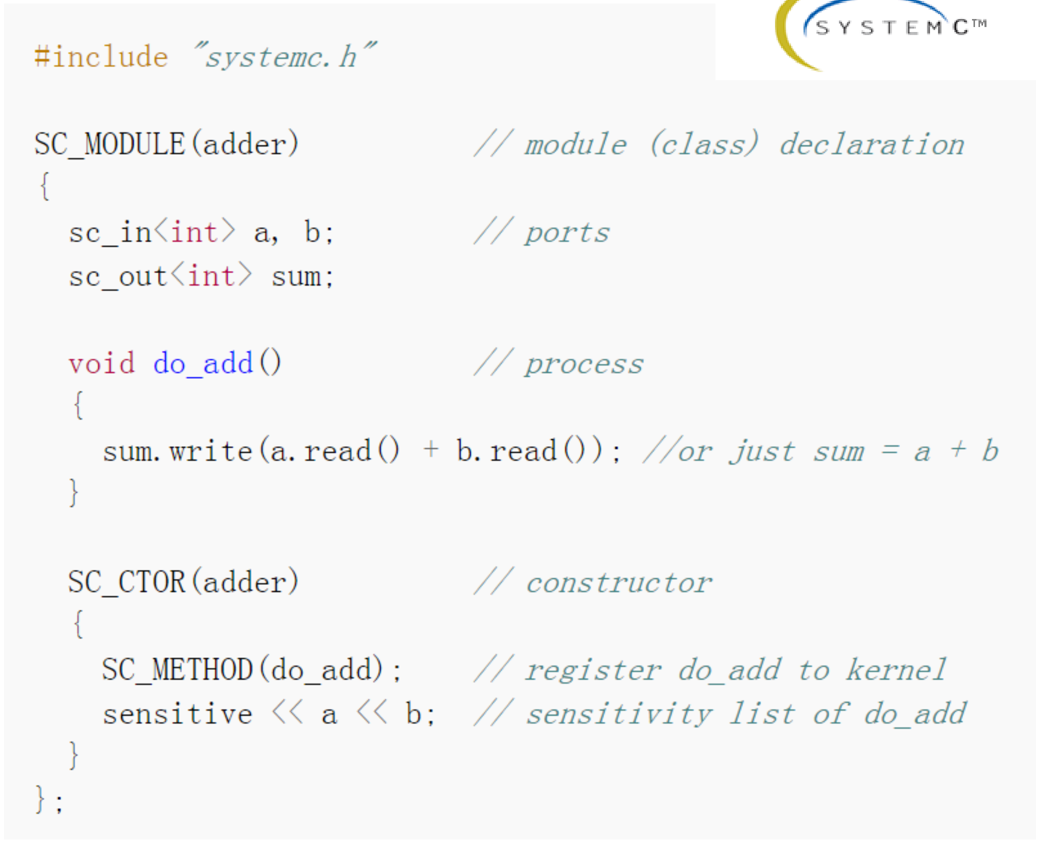
## 2.2 Hardware description languages

| Ada Lovelace | |
|---|---|
| Birthday | Sunday, December 10, 1815 |
| Birthplace | London, England |
| Died | Saturday, November 27, 1852 |
| Nationality | British |

- Ada Lovelace
- The daughter of Lord Byron, who became the world's first programmer while cooperating with **Charles Babbage** on the design of his mechanical computing engines in the mid-1800s
- The language **Ada** was named after her

## 2.2 Hardware description languages

### More: System C, Handel-C, System Verilog



```
#include "systemc.h"

SC_MODULE(adder)          // module (class) declaration
{
  sc_in<int> a, b;        // ports
  sc_out<int> sum;

  void do_add()           // process
  {
    sum.write(a.read() + b.read()); //or just sum = a + b
  }

  SC_CTOR(adder)          // constructor
  {
    SC_METHOD(do_add);    // register do_add to kernel
    sensitive << a << b;  // sensitivity list of do_add
  }
};
```

**System C** is created as an extention to C++

## 2.2.1 Learning a language

| C | VHDL/Verilog |
|---|---|
| “C” is an mix of “High-level language” and “Assembly language” | “VHDL” is an “Hardware description language” |
| “C” is only handle “Sequential instructions” | “VHDL” allows both “Sequential & concurrent executions” |

- Most importantly, VHDL has statements that execute concurrently (parallelly)
- HDL must model real hardware in which the components are all in operation at the same time

For that reason, VHDL is usually referred to as a **code** rather than a program

## 2.2.1 Learning a language

- It is important to distinguish the descriptions represent combinational hardware versus sequential hardware
- Just like fluency in a natural language comes by speaking, reading, and writing the language, mastery of VHDL comes by **repeated use** to create models for various digital systems

# Chapter 2 Introductin to VHDL

## 2.3 VHDL description of combinational circuits

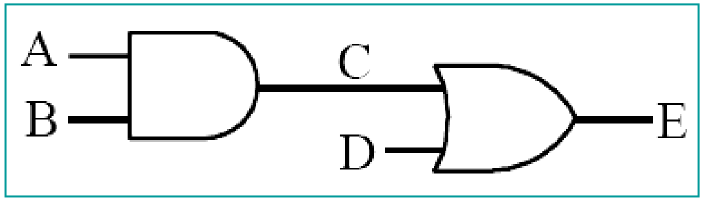
| Computer program (C or Java) | Hardware description language |
|---|---|
| ➢ Sequence of instructions with a well-defined order<br>➢ Encounter and execute different parts of the program sequentially | ➢ Represent concurrently operating hardware<br>➢ “Simulate” the execution of several parts of the circuit at the same time |

**Concurrent statement in VHDL （并发语句）**

- ➢ Concurrent statements are statements which are always ready to execute
- ➢ Concurrent statements get evaluated any time and every time a signal on the right side of the statement changes

## 2.3 VHDL description of combinational circuits

### Concurrent statements

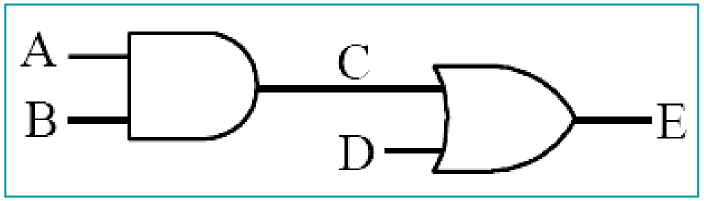


5-ns propagation delay for both gates

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

- A, B, C, D, E: signals

# 2.3 VHDL description of combinational circuits



## Concurrent statements

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

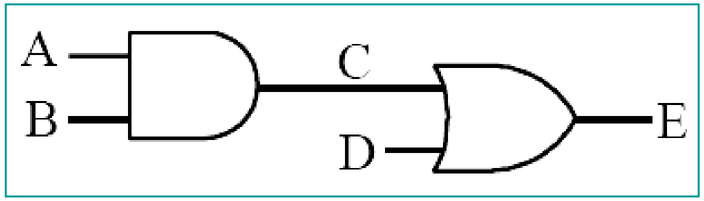
evaluated anytime **A or B** changes

evaluated anytime **C or D** changes

signal assignment operator

## 2.3 VHDL description of combinational circuits

### Concurrent statements

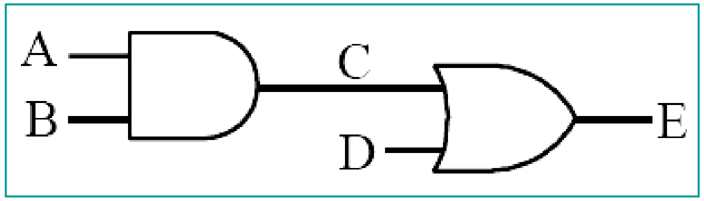


```
C <= A and B after 5 ns;
E <= C or D after 5 ns;
```

| t(ns) | A | B | C | D | E |
|---|---|---|---|---|---|
| $0^-$ | 1 | 0 | 0 | 0 | 0 |
| 0 | | | | | |
| 5 | | | | | |
| 10 | | | | | |

## 2.3 VHDL description of combinational circuits

### Concurrent statements

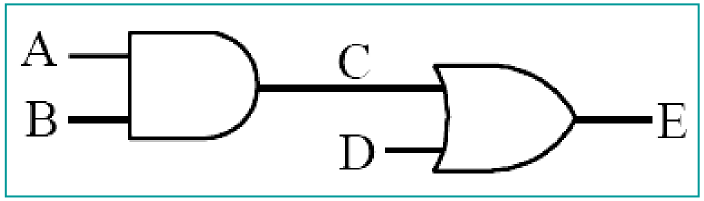


```
C <= A and B after 5 ns;
E <= C or D after 5 ns;
```

| t(ns) | A | B | C | D | E |
|---|---|---|---|---|---|
| $0^-$ | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 5 | | | | | |
| 10 | | | | | |

## 2.3 VHDL description of combinational circuits

### Concurrent statements

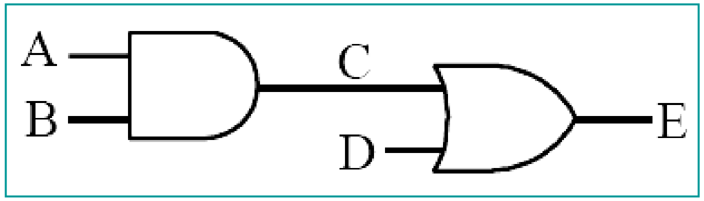


```
C <= A and B after 5 ns;
E <= C or D after 5 ns;
```

| t(ns) | A | B | C | D | E |
|---|---|---|---|---|---|
| $0^-$ | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 5 | 1 | 1 | 1 | 0 | 0 |
| 10 | | | | | |

## 2.3 VHDL description of combinational circuits

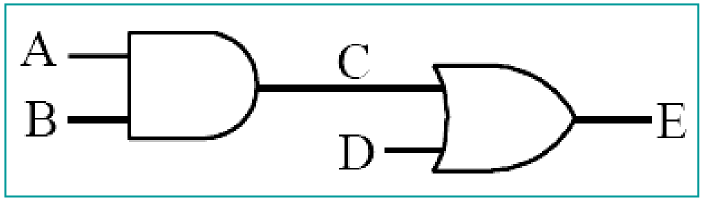
### Concurrent statements



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

| t(ns) | A | B | C | D | E |
|---|---|---|---|---|---|
| $0^-$ | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 5 | 1 | 1 | 1 | 0 | 0 |
| 10 | 1 | 1 | 1 | 0 | 1 |

## 2.3 VHDL description of combinational circuits

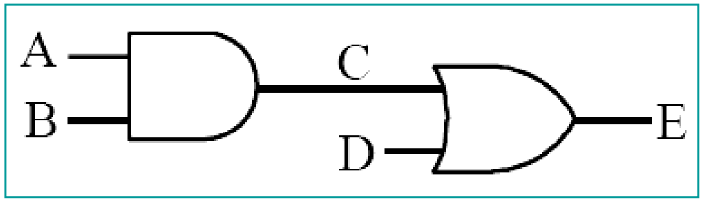


```
C <= A and B;
E <= C or D;
```

If delay is not specified, "delta" delay is assumed

| t(ns) | A | B | C | D | E |
|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 |
| 1 + Δ | 1 | 1 | 1 | 0 | 0 |
| 1 + 2 Δ | 1 | 1 | 1 | 0 | 1 |

## 2.3 VHDL description of combinational circuits



```
C <= A and B;
E <= C or D;
```

=

```
E <= C or D;
C <= A and B;
```

# 2.3 VHDL description of combinational circuits

## General form of signal assignment statement

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

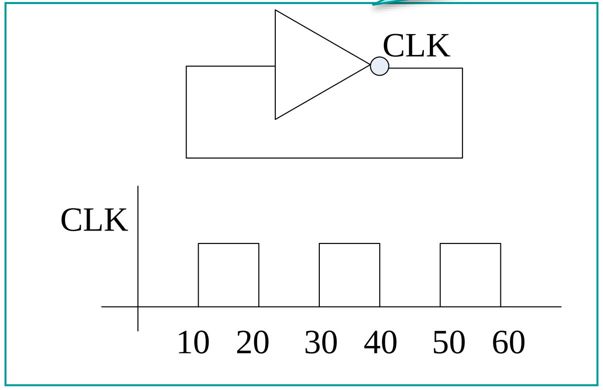
Optional

```
signal_name <= expression;
```

Delta delay is used

## 2.3 VHDL description of combinational circuits

Delay : 10ns



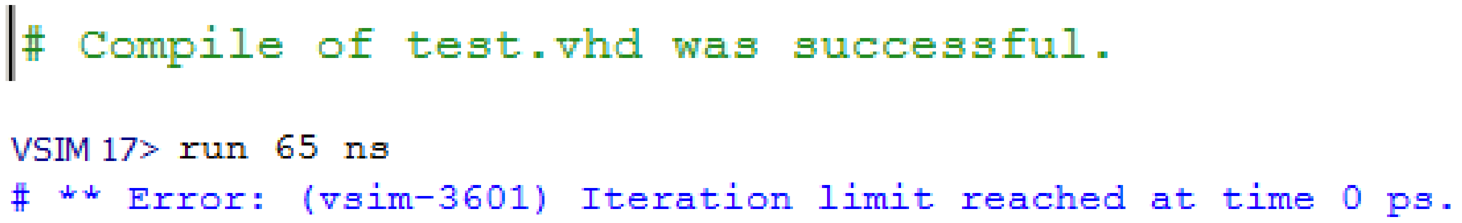
```
CLK <= not CLK after 10 ns;
```

## 2.3 VHDL description of combinational circuits

CLK <= **not** CLK **after** 10 ns;

a clock waveform with a half period of 10 ns

CLK <= **not** CLK;



```
# Compile of test.vhd was successful.

VSIM 17> run 65 ns
# ** Error: (vsim-3601) Iteration limit reached at time 0 ps.
```

CLK <= **not** CLK after Δ;

$\Delta + \Delta + \Delta + \Delta + \ldots << 1$

## 2.3 VHDL description of combinational circuits

√ Clk <= **NOT** clk **After** 10 ns;

√ CLK <= **not** CLK **after** 10 ns;

VHDL is **not case sensitive**

| **Legal identifiers** | **Illegal identifiers** |
|---|---|
| √ C123<br>√ ab_23 | × 1ABC<br>× ABC_ |

Signal names and other VHDL identifiers may contain **letters**, **numbers**, and _

1. An identifier must start with a **letter**
2. An identifier cannot end with _
3. Reserved word cannot be used as identifiers

# 2.3 VHDL description of combinational circuits

**Reserved words**

| | | | |
|---|---|---|---|
| *From VHDL 87:* | ENTITY | OPEN | WAIT |
| | EXIT | OR | WHEN |
| ABS | FILE | OTHERS | WHILE |
| ACCESS | FOR | OUT | WITH |
| AFTER | FUNCTION | PACKAGE | XOR |
| ALIAS | GENERATE | PORT | |
| ALL | GENERIC | PROCEDURE | *From VHDL 93:* |
| AND | GUARDED | PROCESS | |
| ARCHITECTURE | IF | RANGE | GROUP |
| ARRAY | IN | RECORD | IMPURE |
| ASSERT | INOUT | REGISTER | INERTIAL |
| ATTRIBUTE | IS | REM | LITERAL |
| BEGIN | LABEL | REPORT | POSTPONED |
| BLOCK | LIBRARY | RETURN | PURE |
| BODY | LINKAGE | SELECT | REJECT |
| BUFFER | LOOP | SEVERITY | ROL |
| BUS | MAP | SIGNAL | ROR |
| CASE | MOD | SUBTYPE | SHARED |
| COMPONENT | NAND | THEN | SLA |
| CONFIGURATION | NEW | TO | SLL |
| CONSTANT | NEXT | TRANSPORT | SRA |
| DISCONNECT | NOR | TYPE | SRL |
| DOWNTO | NOT | UNITS | UNAFFECTED |
| ELSE | NULL | UNTIL | XNOR |
| ELSIF | OF | USE | |
| END | ON | VARIABLE | |

## 2.3 VHDL description of combinational circuits

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

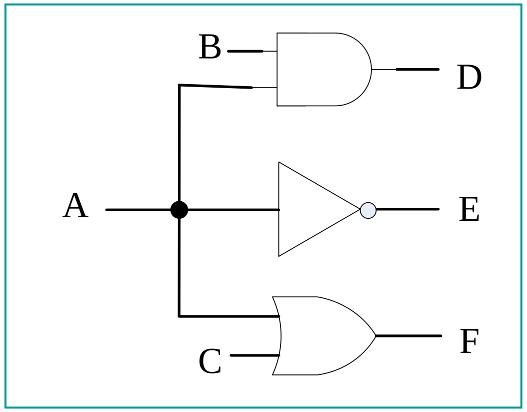
Every VHDL statement must be ended with a semicolon ;

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

```
signal1 <= expression1; signal2 <= expression2;
```

```
signal_name <= expression;             -- Comment started with --
```

# 2.3 VHDL description of combinational circuits



```
D <= A and B after 2 ns;
E <= not A after 1 ns;
F <= A or C after 3 ns;
```

When A changes, these concurrent statements all execute at the same time

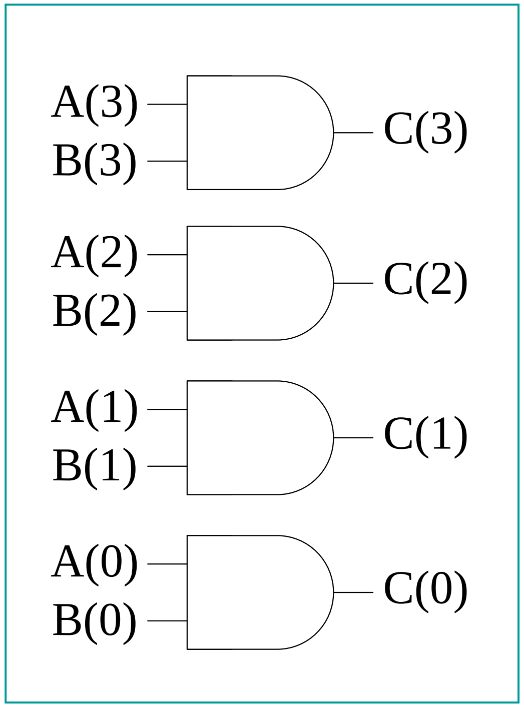
the time at which the statement executes

≠

the time at which the signal is updated

## 2.3 VHDL description of combinational circuits

B: **in** bit_vector (3 **downto** 0)



```
-- the hard way
C(3) <= A(3) and B(3);
C(2) <= A(2) and B(2);
C(1) <= A(1) and B(1);
C(0) <= A(0) and B(0);
```

```
-- the easy way
-- A, B : in bit_vector(3 downto 0)
-- C    : out bit_vector(3 downto 0)

C <= A and B;
```

perform the same operation on a group of signals

## 2.3 VHDL description of combinational circuits

B: **in** bit_vector (3 **downto** 0)

```
B <=   "1100"
```

```
B(3) <=   '1'
B(2) <=   '1'
B(1) <=   '0'
B(0) <=   '0'
```

B: **in** bit_vector (0 **to** 3)

```
B <=   "1100"
```

```
B(0) <=   '1'
B(1) <=   '1'
B(2) <=   '0'
B(3) <=   '0'
```

# 2.3 VHDL description of combinational circuits

```
signal up : out bit_vector (1 to 4);
signal down : in bit_vector (4 downto 1);
```

Consider two signals with different ranges

```
up <= down;
```

The assignment of one signal to the other is legal, since they are the same basetype and the same length, even though their actual ranges are different

```
up(1) <= down(4);
up(2) <= down(3);
up(3) <= down(2);
up(4) <= down(1);
```

The assignments are made **element by element** from left to right, this is equivalent to

- Notice that it is the position of the element in a bit-vector, not its index that determines the outcome of the assignment
- There is plenty of scope for pitfalls here