# EDA322
# Digital Design

Lecture 1:
**Introduction and VHDL basics**

Ioannis (Yiannis) Sourdis

# Course Objectives

- Understand how digital hardware components operate and how they are designed.
  - Which are the basis for every digital computer system
- Be a successful designer of digital circuits
  - A necessary step to be able to design your own hardware.
  - Or at least understand how the hardware you will be using (program) actually looks like.

Prerequisites:

- EDA432/DIT790 Fundamentals of Digital & Computer Engineering (or equivalent)

# Course Organization

- **Lectures*:** check PingPong regularly for updates
  - Mon 13:00-15:00, @ HB2
  - Wed 15:00-17:00, @ HB2
  - Thu 15:00-17:00, @ HB2 (only for this week)
  - Fri 10:00-12:00, @ HA4
- **Office hours:** check PingPong
- **Exercise sessions** (Starting next week)**:**
  - Wed 8:00-10:00 or Thu 10:00-12:00
- **Lab** – once a week
- **Exam** – written based on the Lectures (**50% to pass**)
- **Final Grade**
  - Chalmers: 0, 3, 4, 5
  - GU: Fail (U), Pass (G) & Pass w/ Distinction (VG)

# Outline of the Lectures

- **Week 1:**
  - **Lecture 1:** Intro, **VHDL** basics
  - **Lecture 2:** Boolean logic, logic minimization, adders
  - **Lecture 3:** Combinational logic **VHDL**
  - **Lecture 4:** Simple processor (the one you will implement in the lab), structural **VHDL**
  - **Lab1:** Tutorial for the tools
- **Week 2:**
  - **Lecture 5:** Sequential circuits
  - **Lecture 6:** Sequential circuits -**VHDL**
  - **Exercises** Boolean Algebra (S. Knutsson)
  - **Lab2:** Implement an ALU
- **Week 3:**
  - **Lecture 7:** Finite State Machines
  - **Lecture 8:** FSM **VHDL**
  - **Lecture 9:** Testbenches (**VHDL**)
  - **Exercises** Sequential Circuits (S. Knutsson)
  - **Lab3:** Toplevel of the processor
- **Week 4:**
  - **Lecture 10:** Technologies –ASIC (guest lecture Lars Svensson)
  - **Lecture 11:** Technologies –FPGAs
  - **Lecture 12:** Arithmetic Units,
  - **Exercises** FSMs (S. Knutsson)
  - **Lab4:** Control of the processor

# Outline of the Lectures

- **Week 5:**
  - **Lecture 13:** **HDL** in industry (guest lecture Aeroflex Gaisler)
  - **Lecture 14:** System Design (Interfaces)
  - **Lecture 15:** Memories & Interconnects
  - **Exercises** Arithmetic op. (S. Knutsson)
  - **Lab5:** Testbench for the processor
- **Week 6:**
  - **Lecture 16:** Testing & Design for Testing
  - **Lecture 17:** Pipelines
  - **Lecture 18:** Timing, Delay, power of circuits
  - **Exercises** Timing of circuits (S. Knutsson)
  - **Lab6:** Download your design to an FPGA board and test it
- **Week 7:**
  - **Lecture 18:** Asynchronous circuits
  - **Exercises** testing (S. Knutsson)
  - **Lab7:** Measure performance, power, area of your processor
- **Week 8:**
  - **Lecture 20:** Summary of the lectures
  - **Exercises** on (students') demand (S. Knutsson)
  - **Extra Lab slots**

# Registration

- Register for the course for this quarter
- Register in PingPong (today!)
  - **EDA322 on PingPong is the official course webpage.**
  - **Check PingPong regularly for announcements**
- Register and book a lab time-slot (today!) in PingPong
  - Labs start at study week 1 (tomorrow!)
- Register to the exercise sessions
- Can buy course book – Cremona (recommended)

# Labs

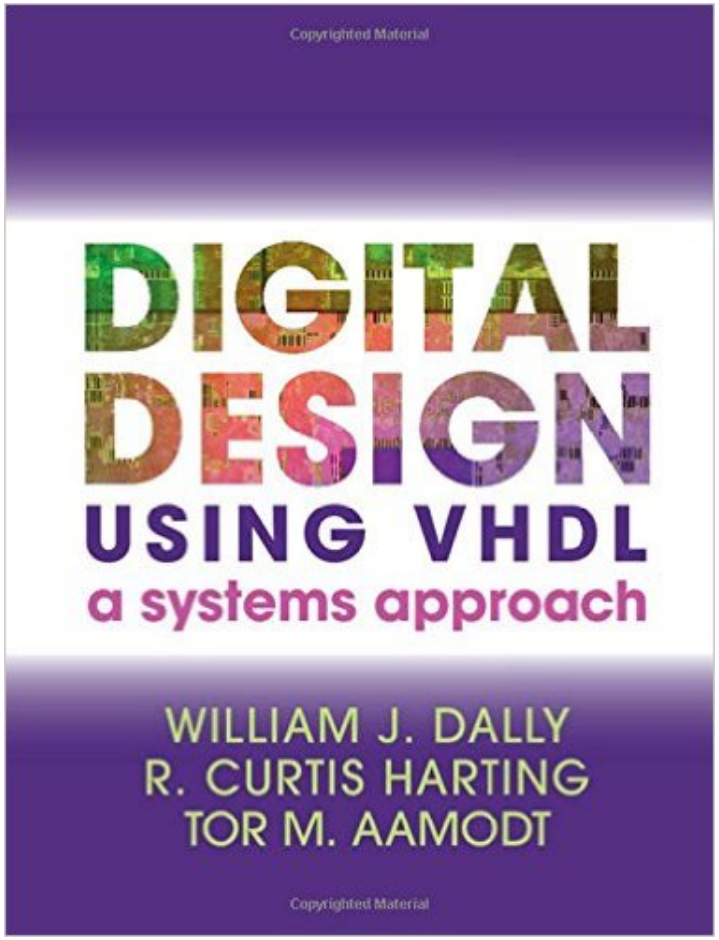
- Be there on time
- Do not bring drinks, coffees, food in the lab room
- Check the Lab assignments and prepare before the lab
  - Bring the preparation-exercises to the labs in order to start your assignment
- Register today!
  - In teams of 2 students

- To successfully pass the Labs you need to successfully:
  - complete at least Labs 2-5, and
    - Successfully completing all 7 labs gives a **bonus of 10 (out of 100) points** to the course exam
  - write the final Lab report
    - **Deadline: the week of the exam**

# Course Material

- Lecture slides
  - Made detailed enough for you to study them after the lectures
- Recommended Book
  - Digital Design Using VHDL: A Systems Approach, 1st Edition by William J. Dally, R. Curtis Harting, Tor M. Aamodt (NEW)
- Online VHDL tutorials
  - Uploaded in PingPong
- Wikipedia – The Free On-line Encyclopedia
  - http://en.wikipedia.org/wiki/VHDL

# Recommended book




- Cremona

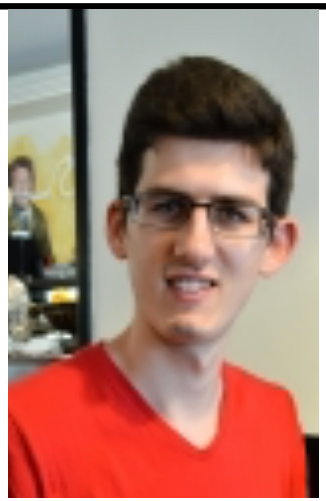
# People

Ioannis Sourdis
(Yiannis / Jannis / Γιάννης)

Associate Professor
Examiner

Sven Knutsson
Teaching Assistant

Ahsen Ejaz
Teaching Assistant



Stefano Ribes
Teaching Assistant

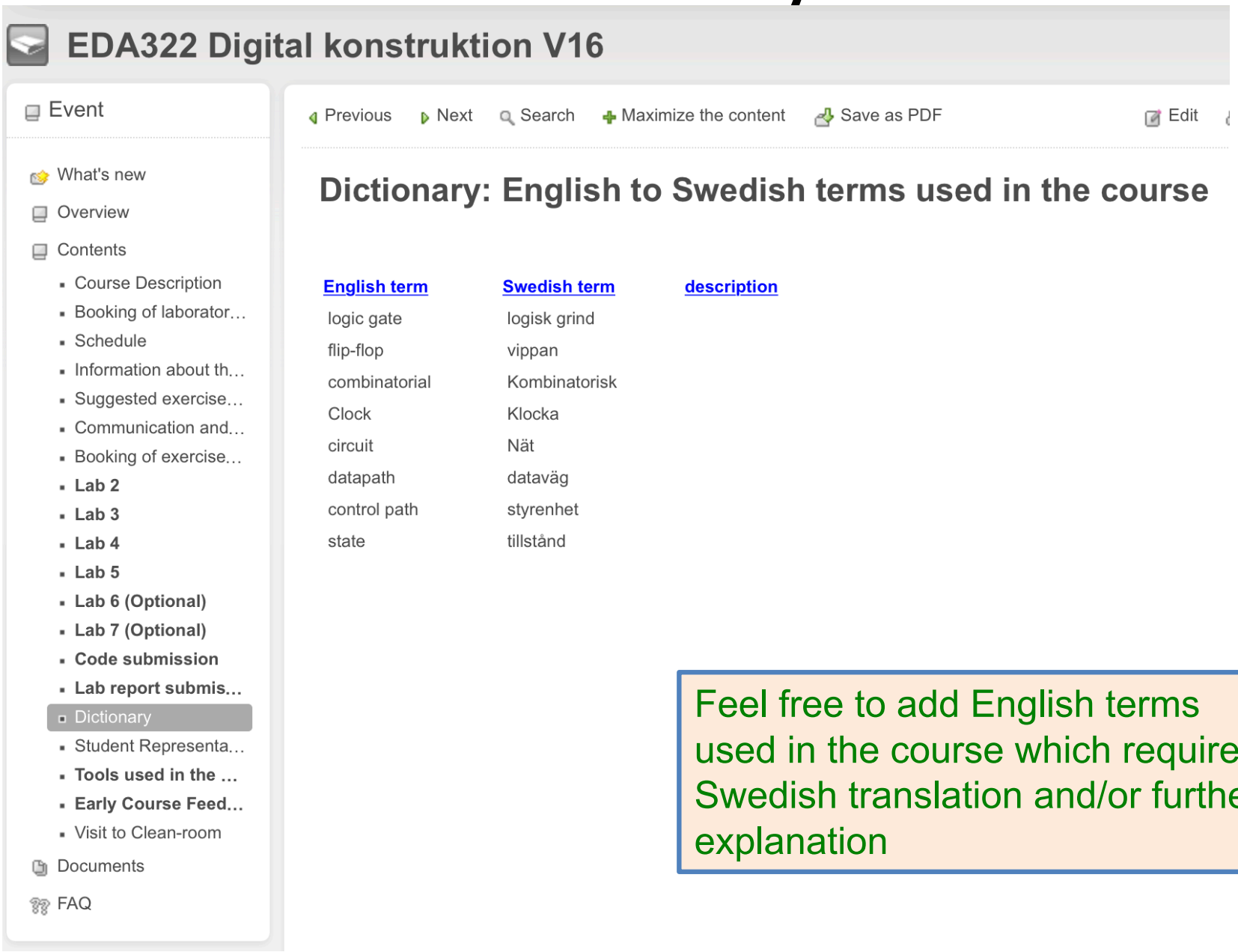
Petros Voudouris
Teaching Assistant

Evangelos Vasilakis
Teaching Assistant

# Communication

- Contact me and course assistants:
  - Through PingPong
    - Toolbox, “ask questions” function
  - Use contact hours (see PingPong)

# Dictionary




Feel free to add English terms used in the course which require Swedish translation and/or further explanation

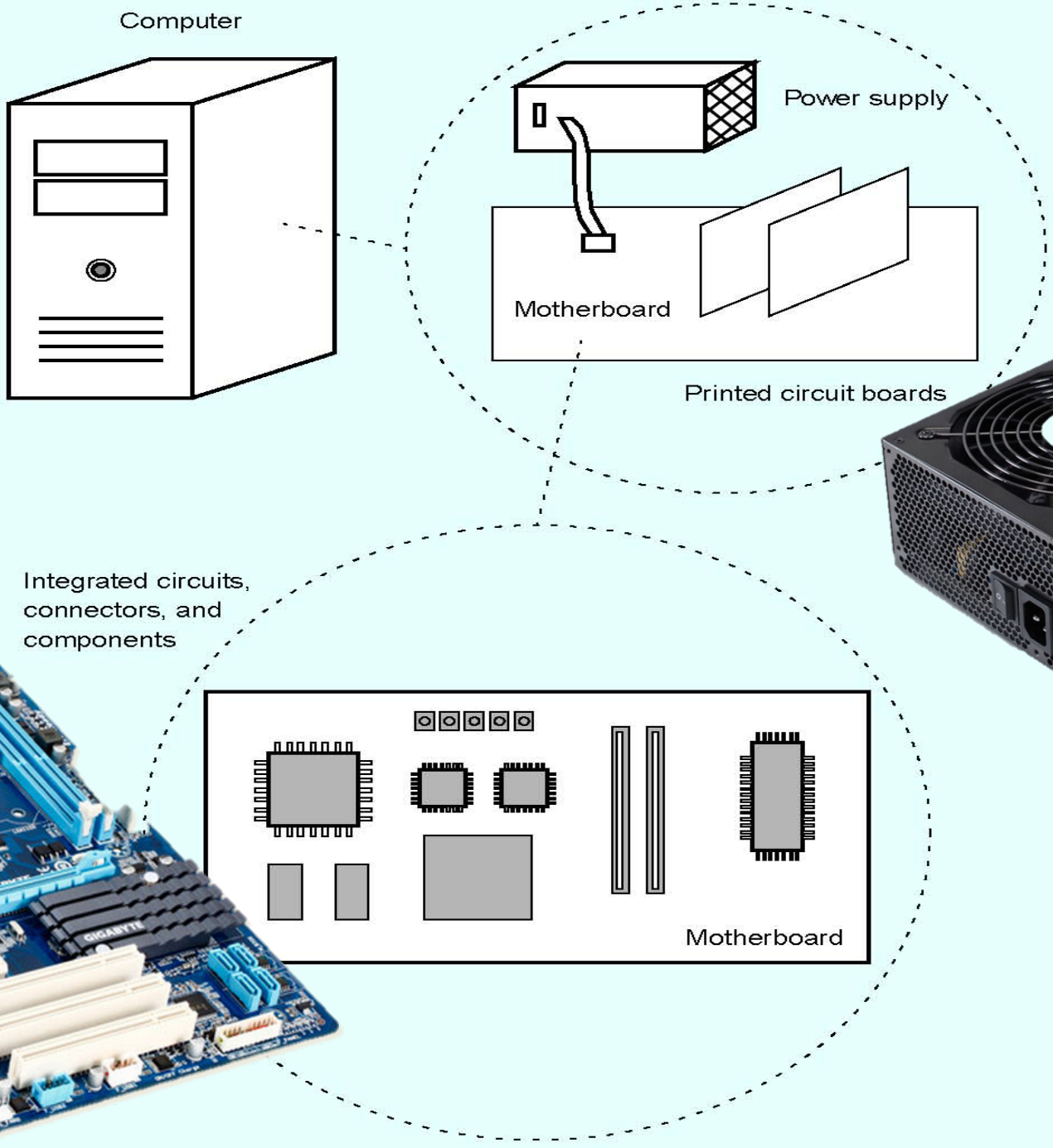
# Outline of Lecture 1

- Use of digital hardware
- The general design flow of digital hardware
- VHDL basics
  - General VHDL format for describing a digital circuit
    - Entity, architecture, ports
  - Describing gates in VHDL
  - Wires and busses
  - Different styles of VHDL

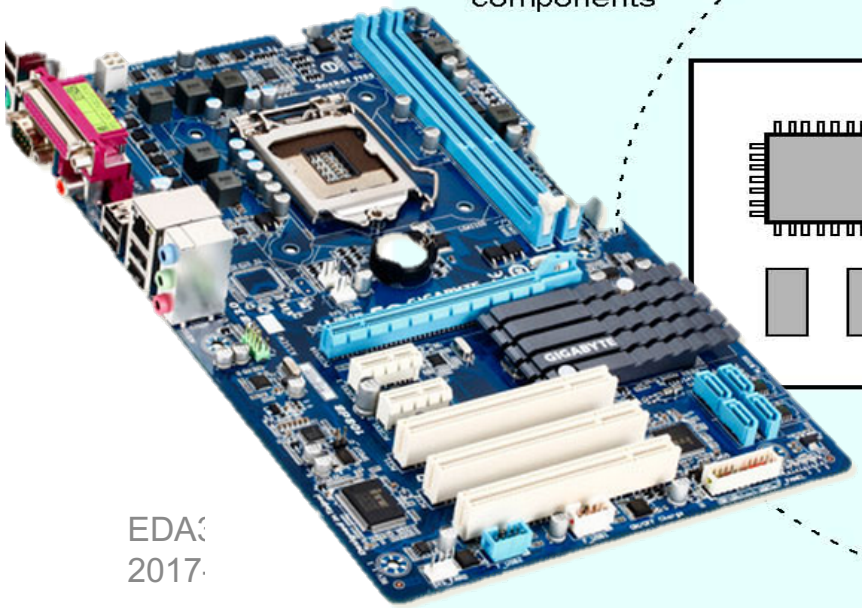
# Outline of Lecture 1

- Use of digital hardware
- The general design flow of digital hardware
- VHDL basics
  - General VHDL format for describing a digital circuit
    - Entity, architecture, ports
  - Describing gates in VHDL
  - Wires and busses
  - Different styles of VHDL

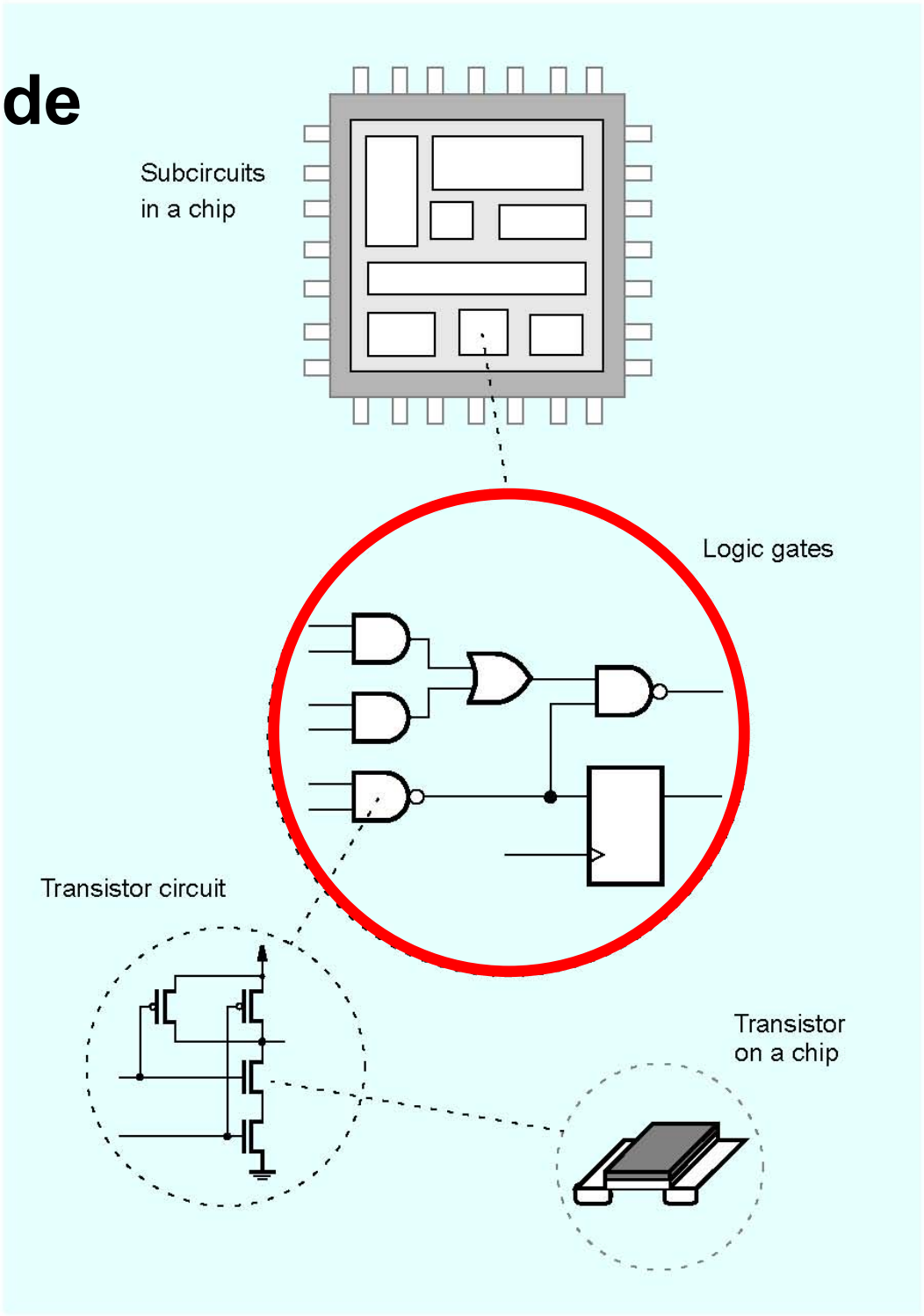
# What's inside a computer box?

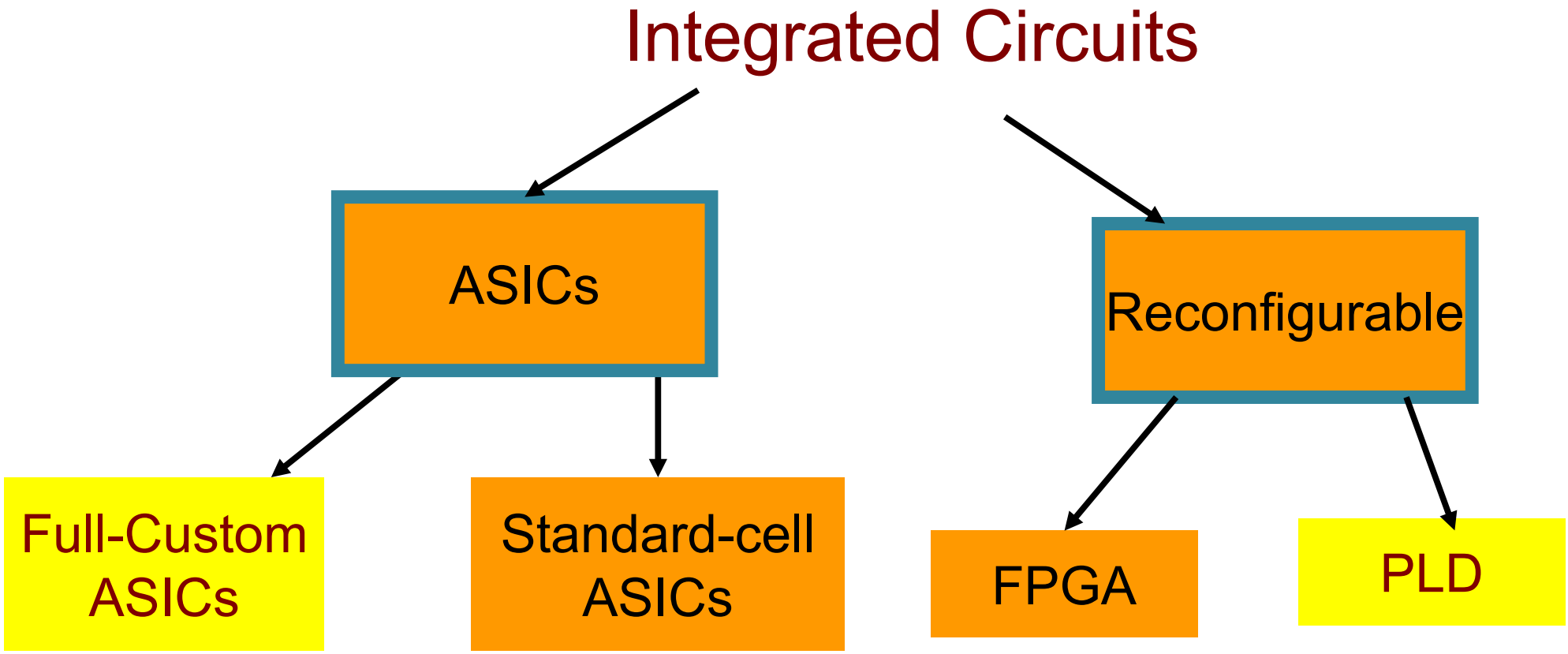




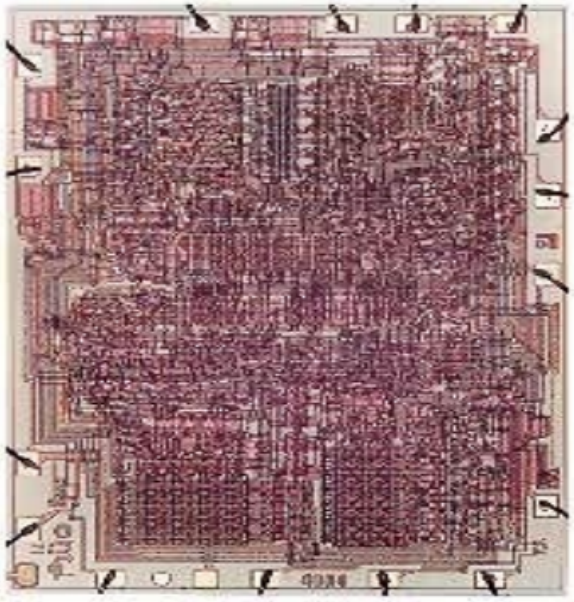
# What's inside a chip?



# Integrated Circuits

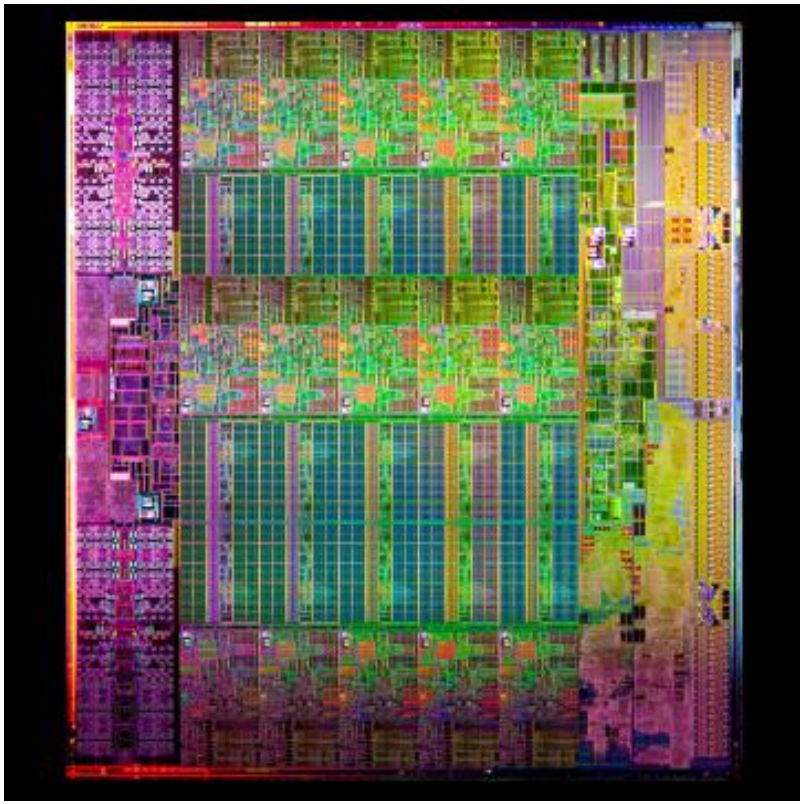


# ASIC:
# Application Specific Integrated Circuits

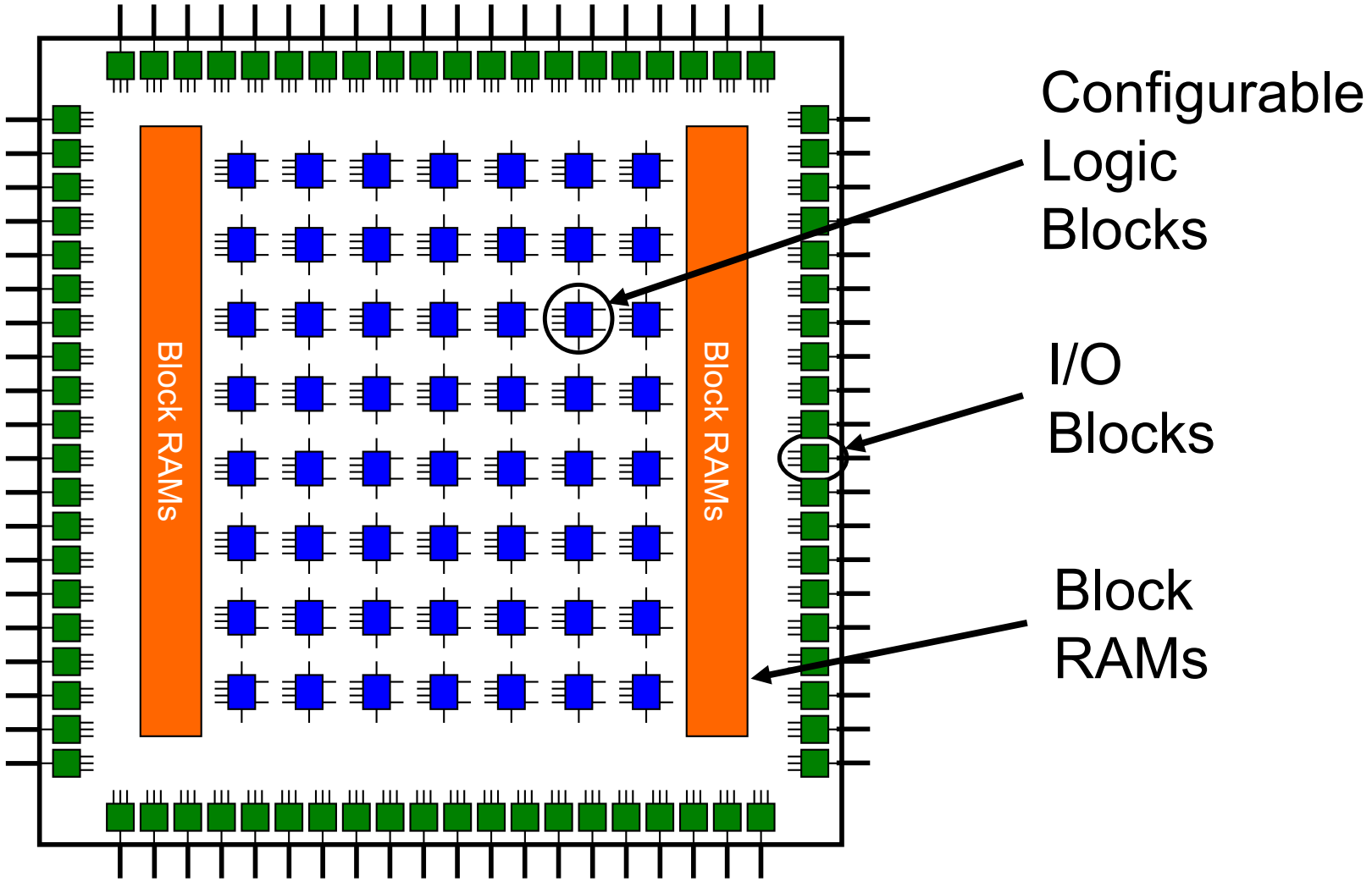


Intel 4004 (1971)
108 KHz 2,300 transistors

Intel Xeon Broadwell-E5 (2016)
3.6GHz,  7.2 billion transistors,
22cores



# FPGA (a Reconfigurable chip): Field Programmable Gate Arrays



# Two competing implementation approaches

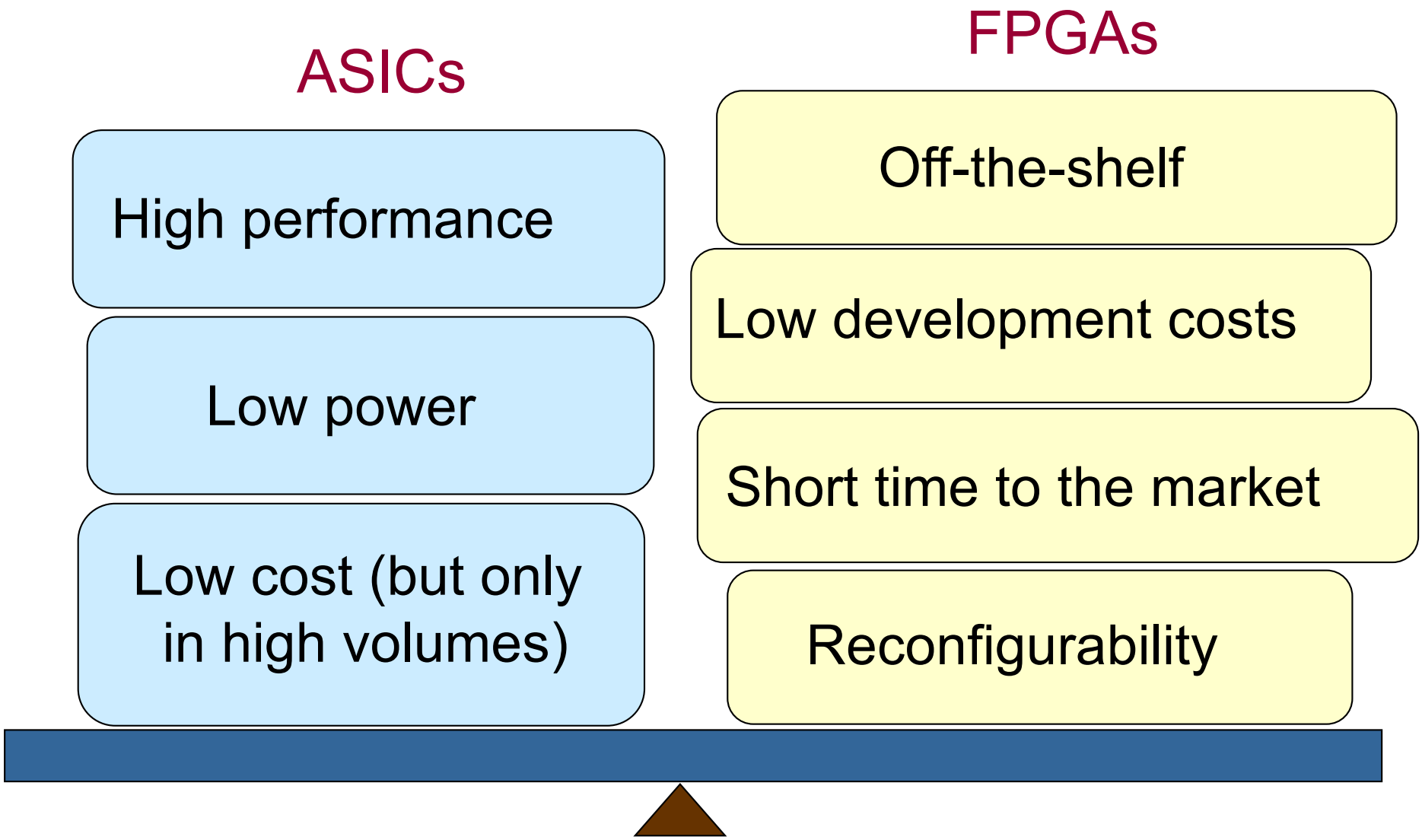
## ASIC
## Application Specific Integrated Circuit

- **once they are fabricated, the hardware design cannot be changed**
- all the way from behavioral description to physical layout (VLSI)
- expensive and time consuming fabrication in semiconductor foundry

## FPGA
## Field Programmable Gate Array

- **generic, can support different hardware designs (reconfigurable)**
- The hardware description is "translated" to a bitstream used to (re)configure the FPGA device
- bought off the shelf and reconfigured by designers themselves

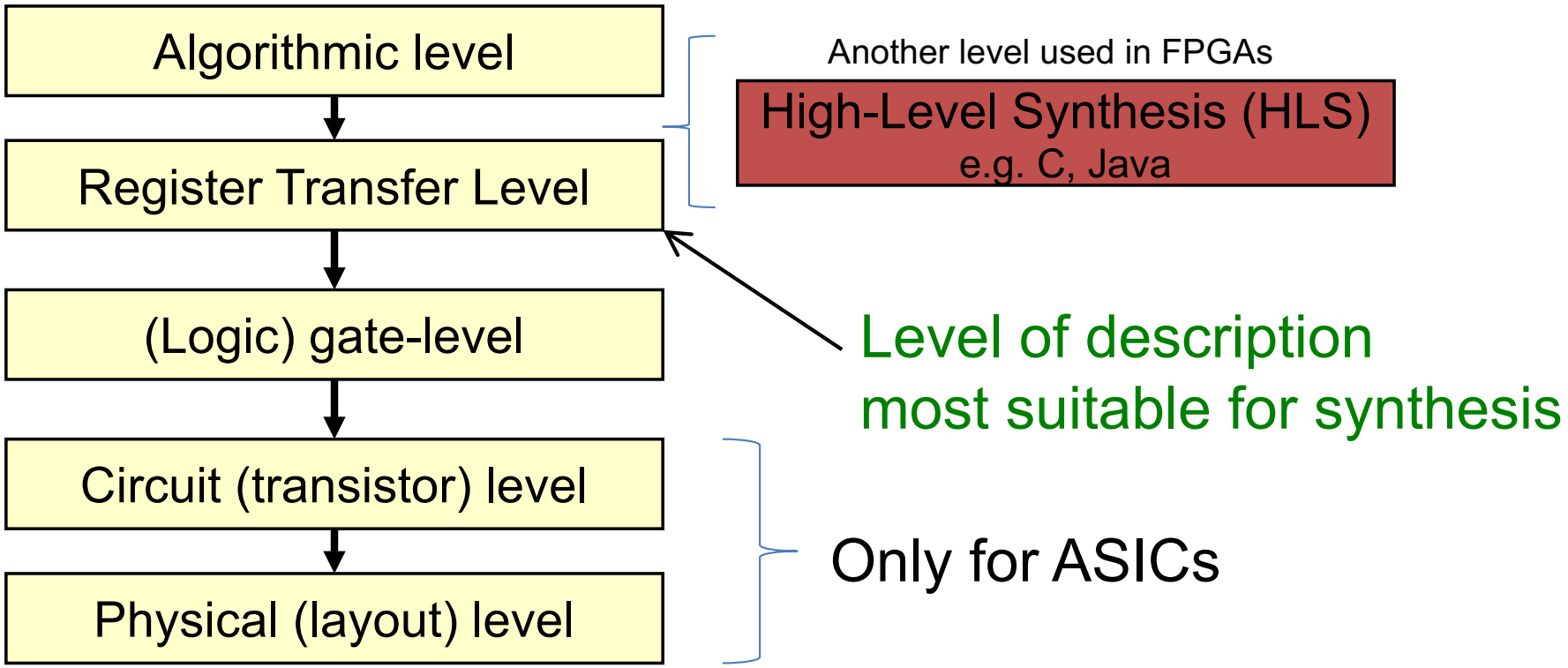
# FPGAs vs. ASICs



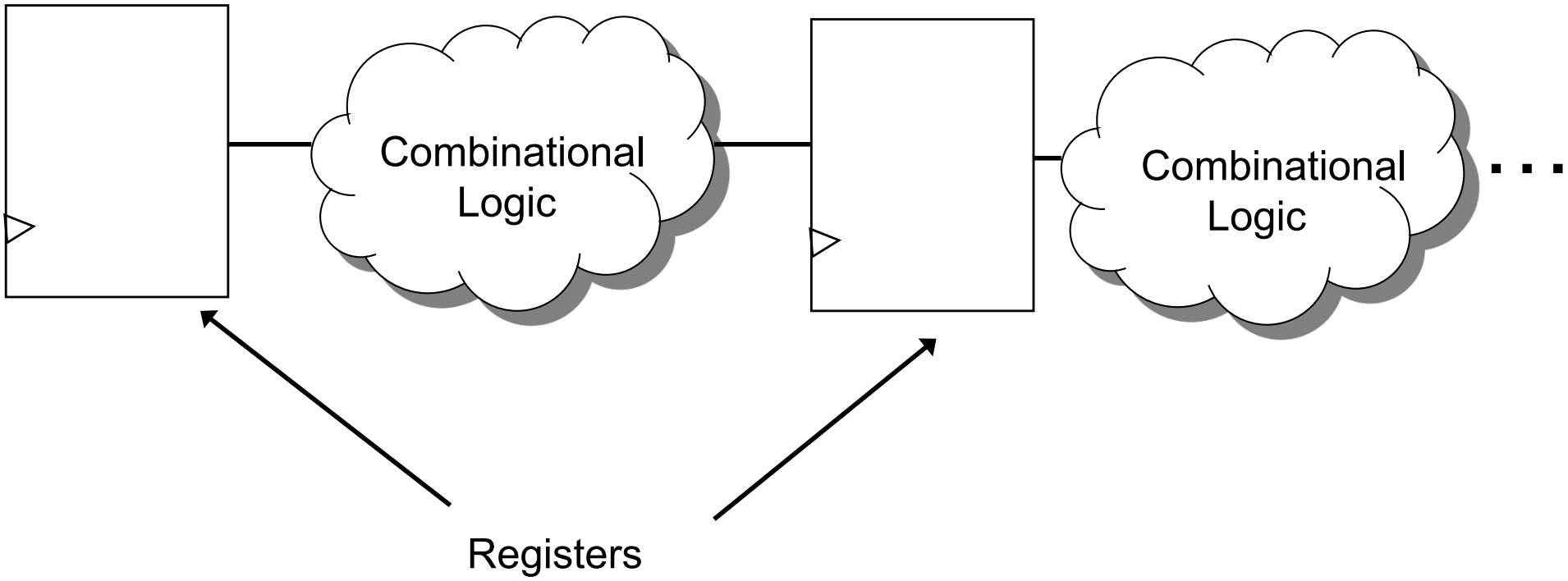
# Outline of Lecture 1

- Use of digital hardware
- The general design flow of digital hardware
- VHDL basics
  - General VHDL format for describing a digital circuit
    - Entity, architecture, ports
  - Describing gates in VHDL
  - Wires and busses
  - Different styles of VHDL

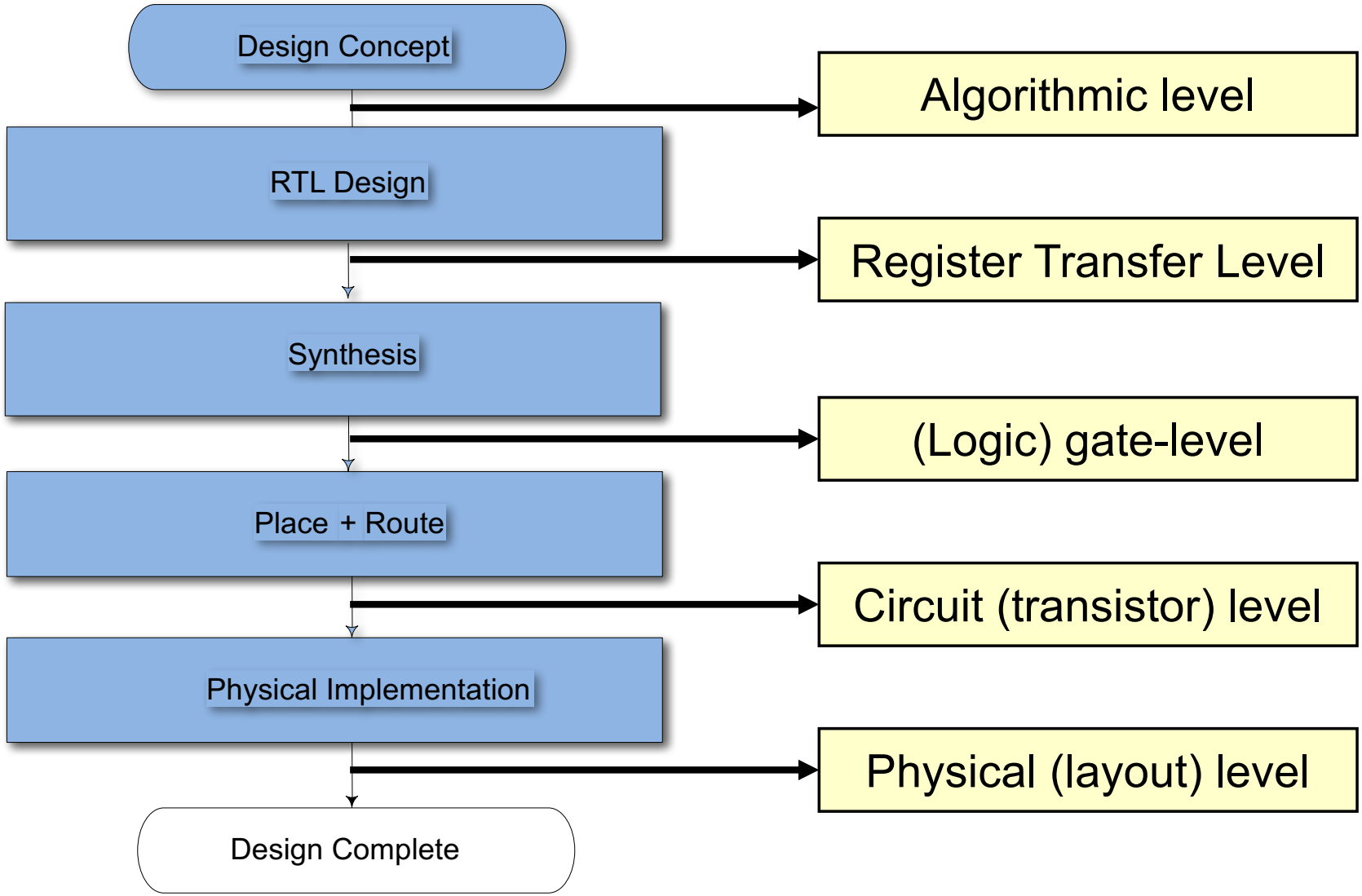
# Levels of hardware design description



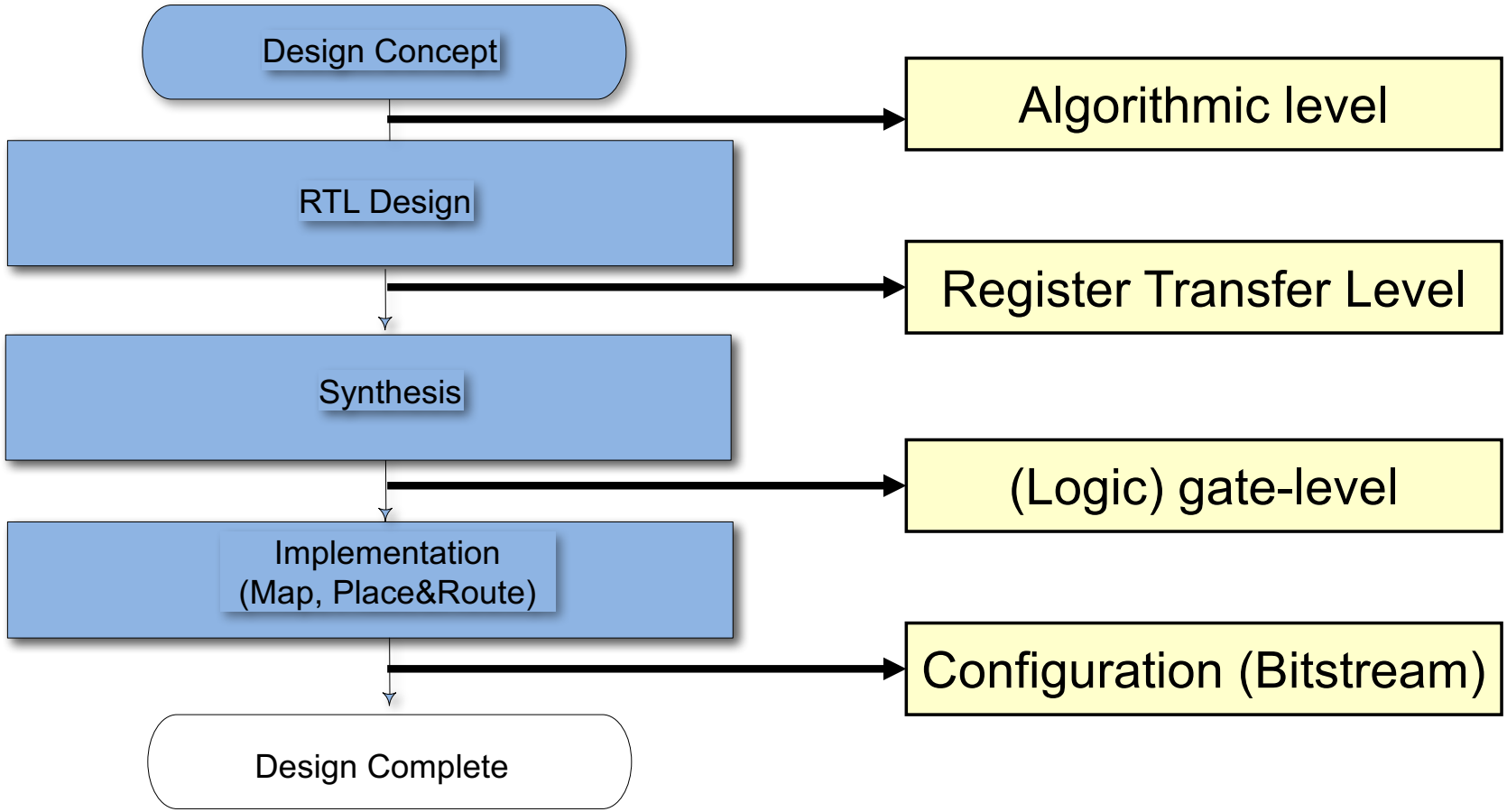
# Register Transfer Level (RTL) Design Description

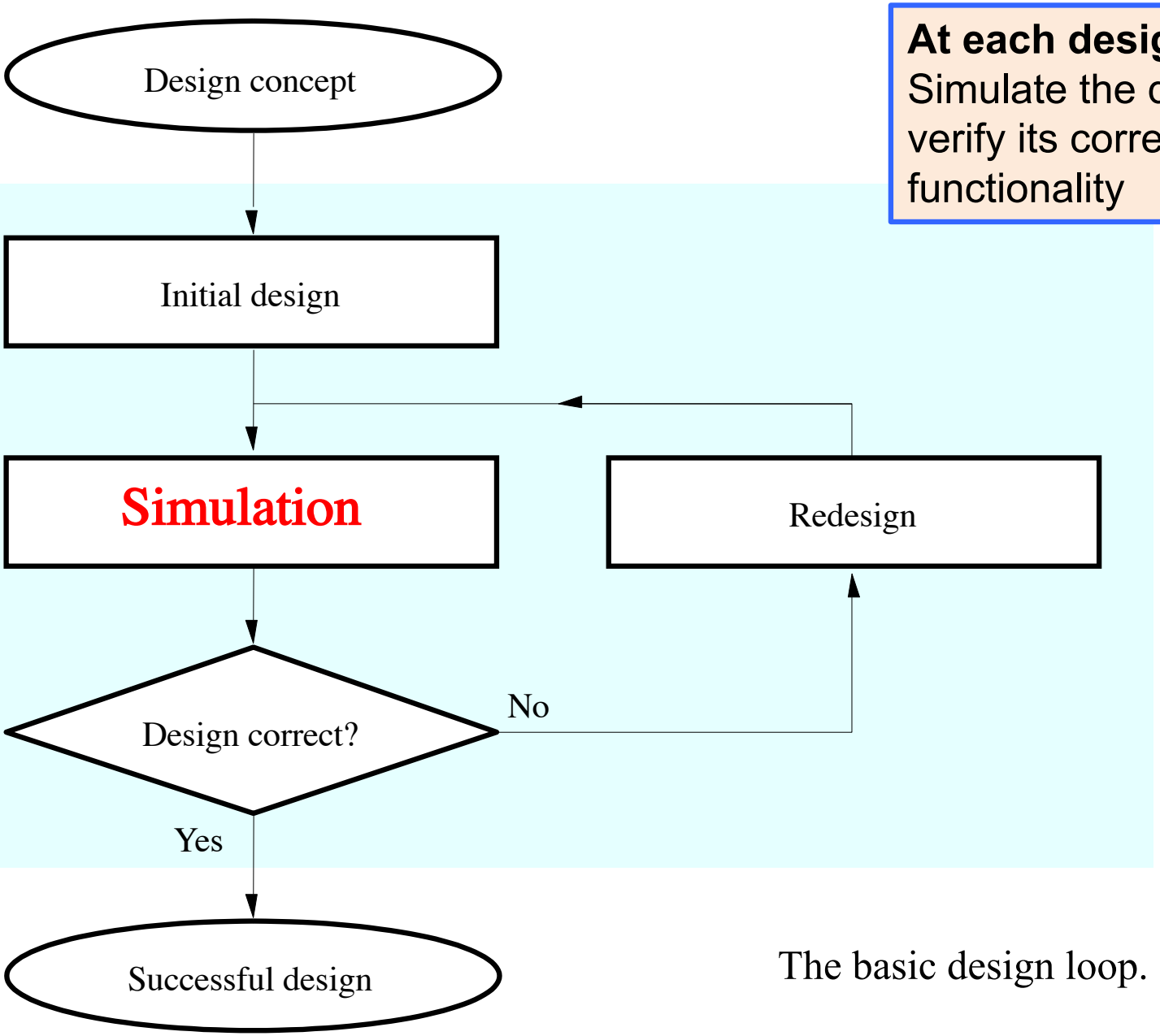


# Top Level ASIC Digital Design Flow



# Top Level FPGA Digital Design Flow






**At each design step:**
Simulate the design to verify its correct functionality

The basic design loop.

# FPGA Design process (1/2)

Specifications

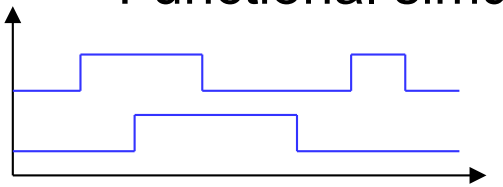
On-paper hardware design
(Block diagram & ASM chart)

**At each design step:**
Simulate the design to verify its correct functionality

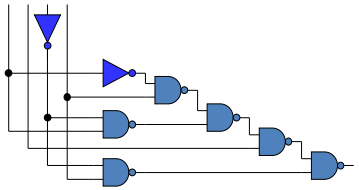
VHDL description (RTL)

```
Library IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity RC5_core is
        port(
                clock, reset, encr_decr: in std_logic;
                data_input: in std_logic_vector(31 downto 0);
                data_output: out std_logic_vector(31 downto 0);
                out_full: in std_logic;
                key_input: in std_logic_vector(31 downto 0);
                key_read: out std_logic;
        );
end AES_core;
```

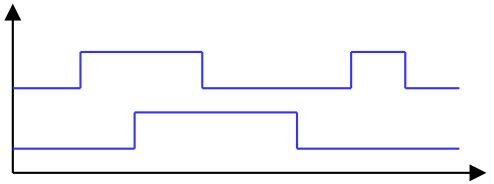
Functional simulation



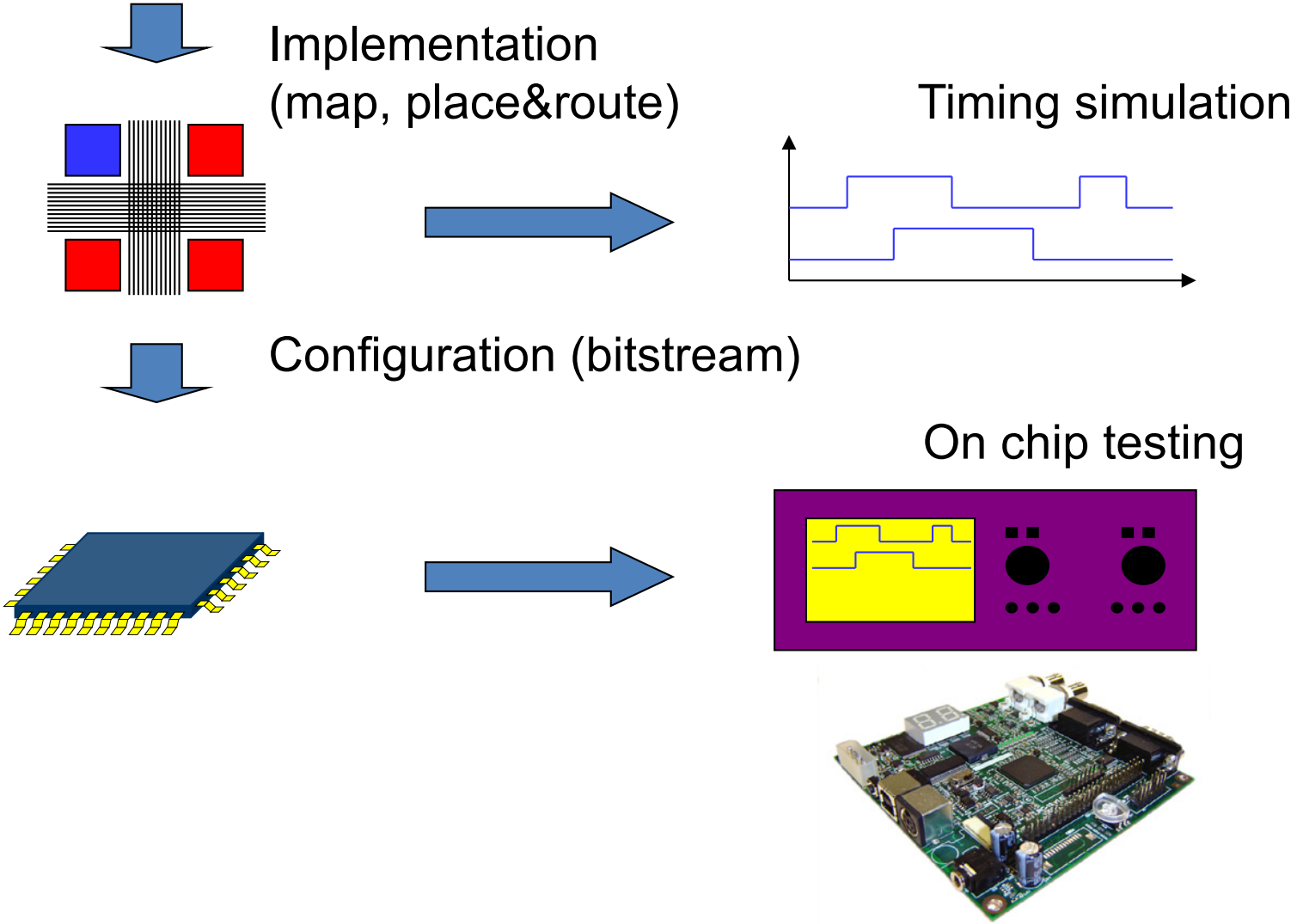
Synthesis (gate-level)



Post-synthesis simulation



# FPGA Design process (2/2)



# Simulation Tools

ModelSim®

# FPGA Synthesis Tools



# Quiz 1-1

http://m.socrative.com/student/#joinRoom

room number: **713113**

- Q1-4 (True/False):
  a. An ASIC can change its hardware after fabrication?
  b. An FPGA is more expensive to fabricate than an ASIC?
  c. An ASIC is faster and more power efficient?
  d. An ASIC has shorter time to market?

- Q5: Put the design steps in the right order
  1. RTL Design
  2. Physical Implementation
  3. Design concept
  4. Place & Route
  5. Synthesis

**write the just the numbers separated with spaces, e.g. 1 2 3 4 5**

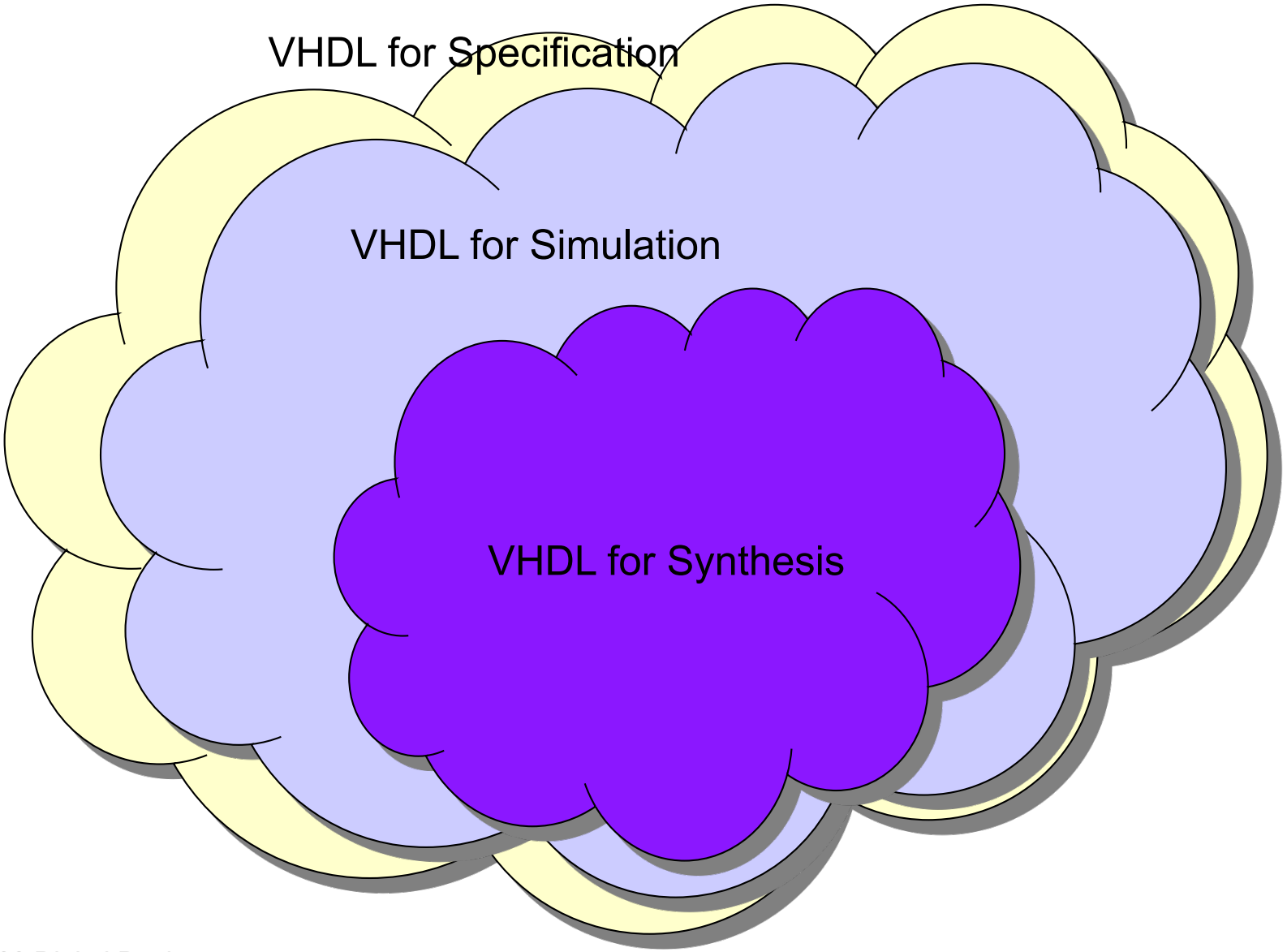
- Q6: Which of the above steps produces:
  a. RTL representation,
  b. Gate-level
  c. Transistor-level
  d. Layout of a design
  e. Algorithmic design

**write the just the numbers separated with spaces, e.g. 1 2 3 4 5**

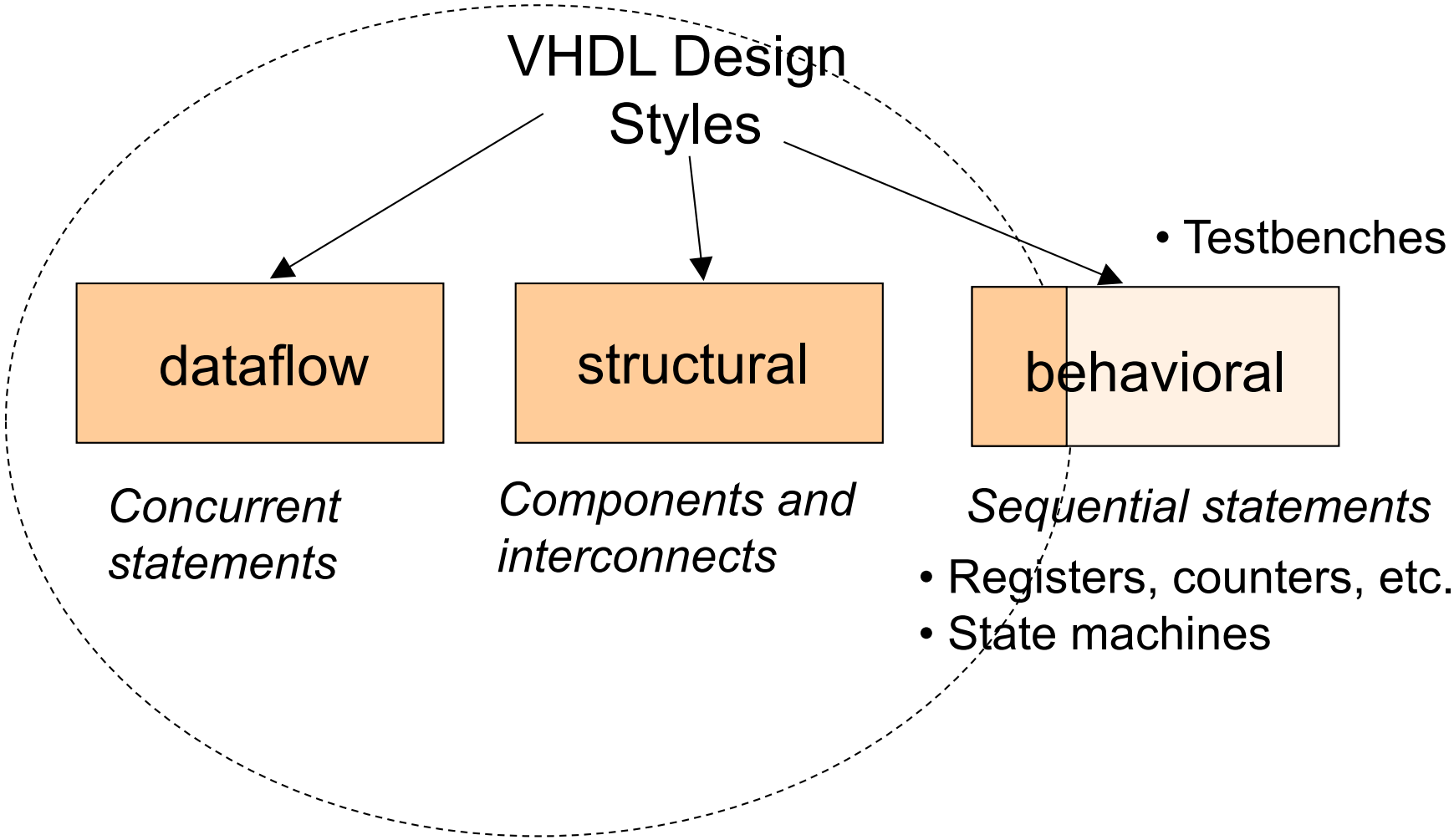
# Outline of Lecture 1

- Use of digital hardware
- The general design flow of digital hardware
- VHDL basics
  - General VHDL format for describing a digital circuit
    - Entity, architecture, ports
  - Describing gates in VHDL
  - Wires and busses
  - Different styles of VHDL

# VHDL basics


VHDL for Specification
VHDL for Simulation
VHDL for Synthesis

# VHDL Design Styles



# VHDL Fundamentals

# Naming and Labeling (1)

- VHDL is case insensitive

  Example:

  Names or labels

  ```
  databus
  Databus
  DataBus
  DATABUS
  ```

  are all equivalent

# Naming and Labeling (2)

General rules of thumb

1. All names should start with an alphabet character (a-z or A-Z)
2. Use only alphabet characters (a-z or A-Z) digits (0-9) and underscore (_)
3. Do not use any punctuation or reserved characters within a name (!, ?, ., &, +, -, etc.)
4. Do not use two or more consecutive underscore characters (__) within a name (e.g., Sel__A is invalid)
5. All names and labels in a given entity and architecture must be unique

# Free Format

- VHDL is a "free format" language

No formatting conventions, such as spacing or indentation imposed by VHDL compilers. Space and carriage return treated the same way.

Example:

```
    if (a=b) then
```

*or*

```
    if (a=b)        then
```

*or*

```
    if (a =
    b) then
```

are all equivalent

# Readability standards & coding style

Adopt readability standards based on one of the VHDL tutorials uploaded in PingPong

Use coding style recommended in
OpenCores Coding Guidelines:
http://cdn.opencores.org/downloads/opencores_coding_guidelines.pdf

Strongly recommended!

# Comments

- Comments in VHDL are indicated with
  a "double dash", i.e., "--"
  - Comment indicator can be placed anywhere in the line
  - Any text that follows in the same line is treated as a comment
  - "Carriage return"- "enter" terminates a comment
  - No method for commenting a block extending over a couple of lines

Examples:

```
-- main subcircuit
Data_in <= Data_bus;     -- reading data from the input FIFO
```

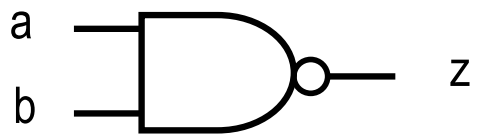
# Comments

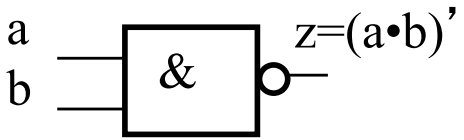
- Explain function of module to other designers

- Explanatory; **not** just restatement of code
- Described at close-to-code location
  - Put near executable code, not just in a header

# Design Entity

# Example: NAND Gate

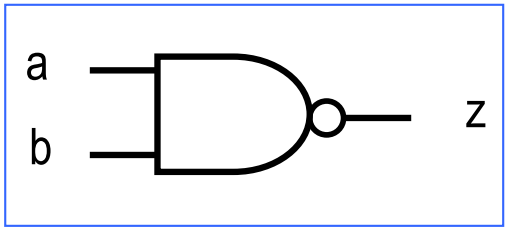



NAND

| a | b | z |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

# Example VHDL Code




- File extension for a VHDL file is **.vhd**
- 3 sections to a piece of VHDL code
- Name of the file should be the same as the entity name (nand_gate.vhd) [OpenCores Coding Guidelines]

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY nand_gate IS
     PORT(
          a    : IN STD_LOGIC;
          b    : IN STD_LOGIC;
          z    : OUT STD_LOGIC);
END nand_gate;

ARCHITECTURE model OF nand_gate IS
BEGIN
     z <= a NAND b;
END model;
```

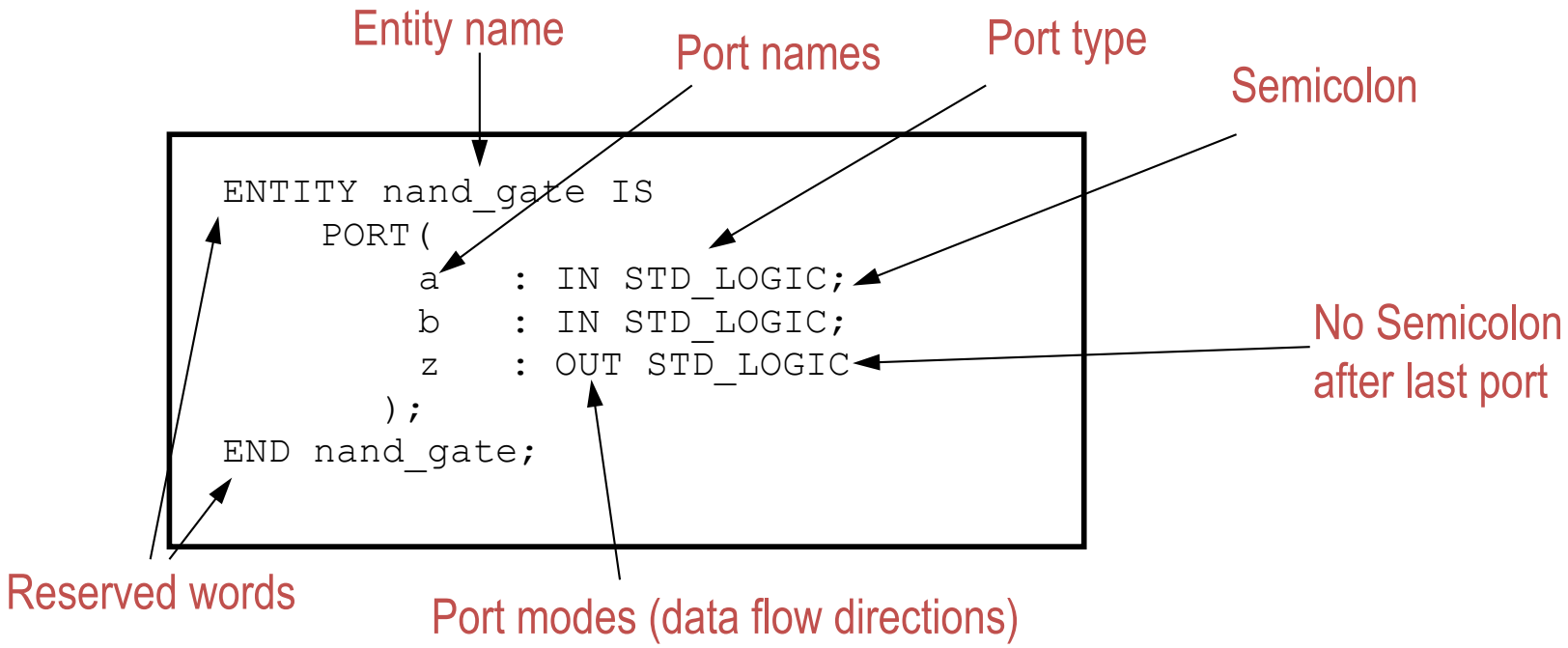
LIBRARY DECLARATION

ENTITY DECLARATION

ARCHITECTURE BODY

# Entity Declaration

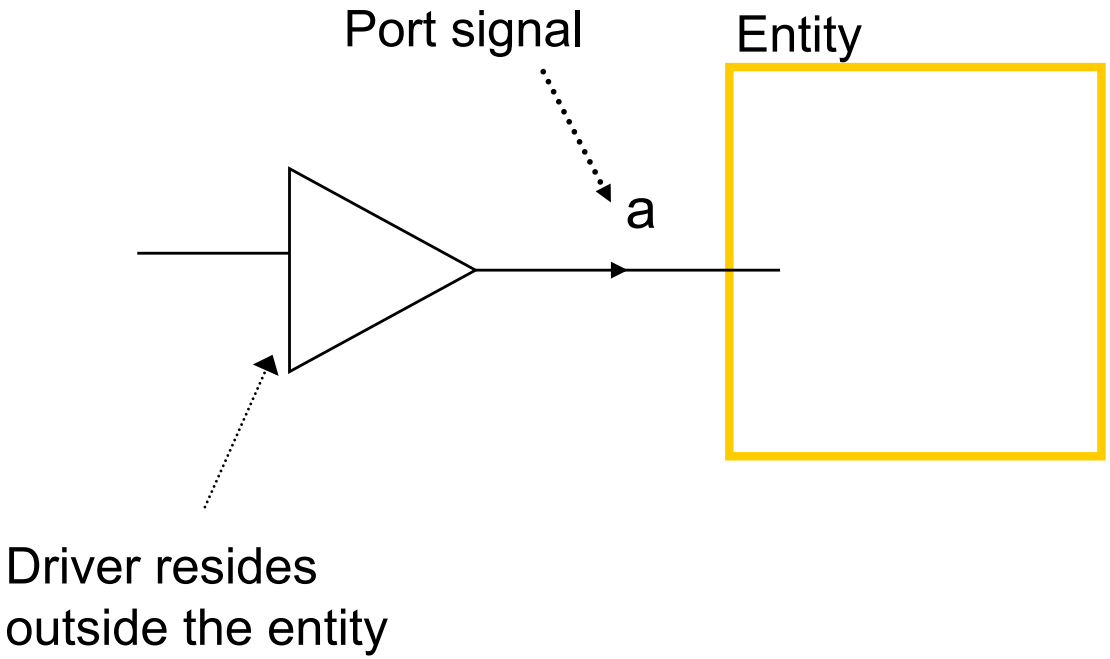
- Entity Declaration describes the interface of the component, i.e. input and output ports.



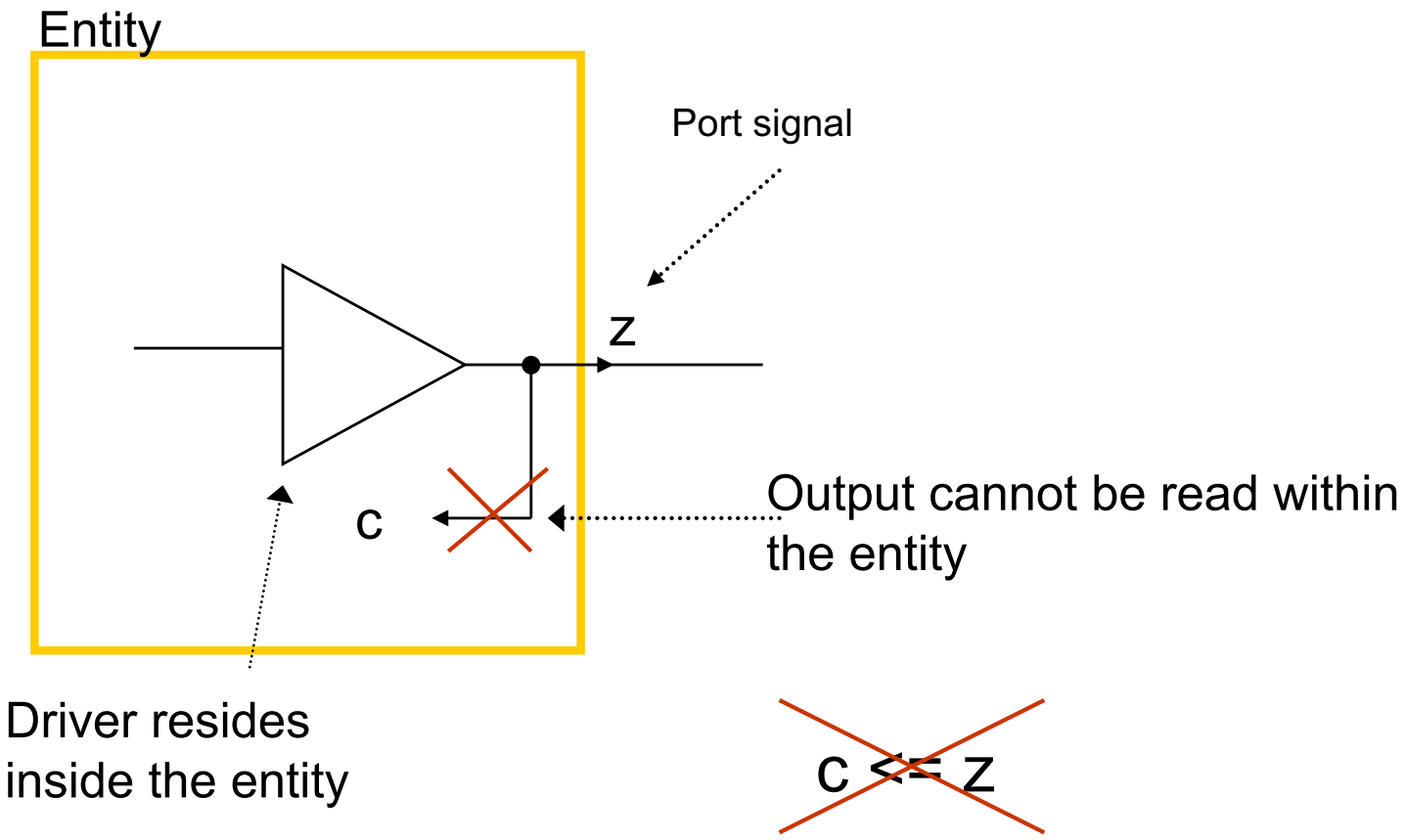
# Entity declaration

```
ENTITY entity_name IS
   PORT (
      port_name :  port_mode  signal_type;
      port_name :  port_mode  signal_type;
      ……..
      port_name :  port_mode  signal_type);
END entity_name;
```

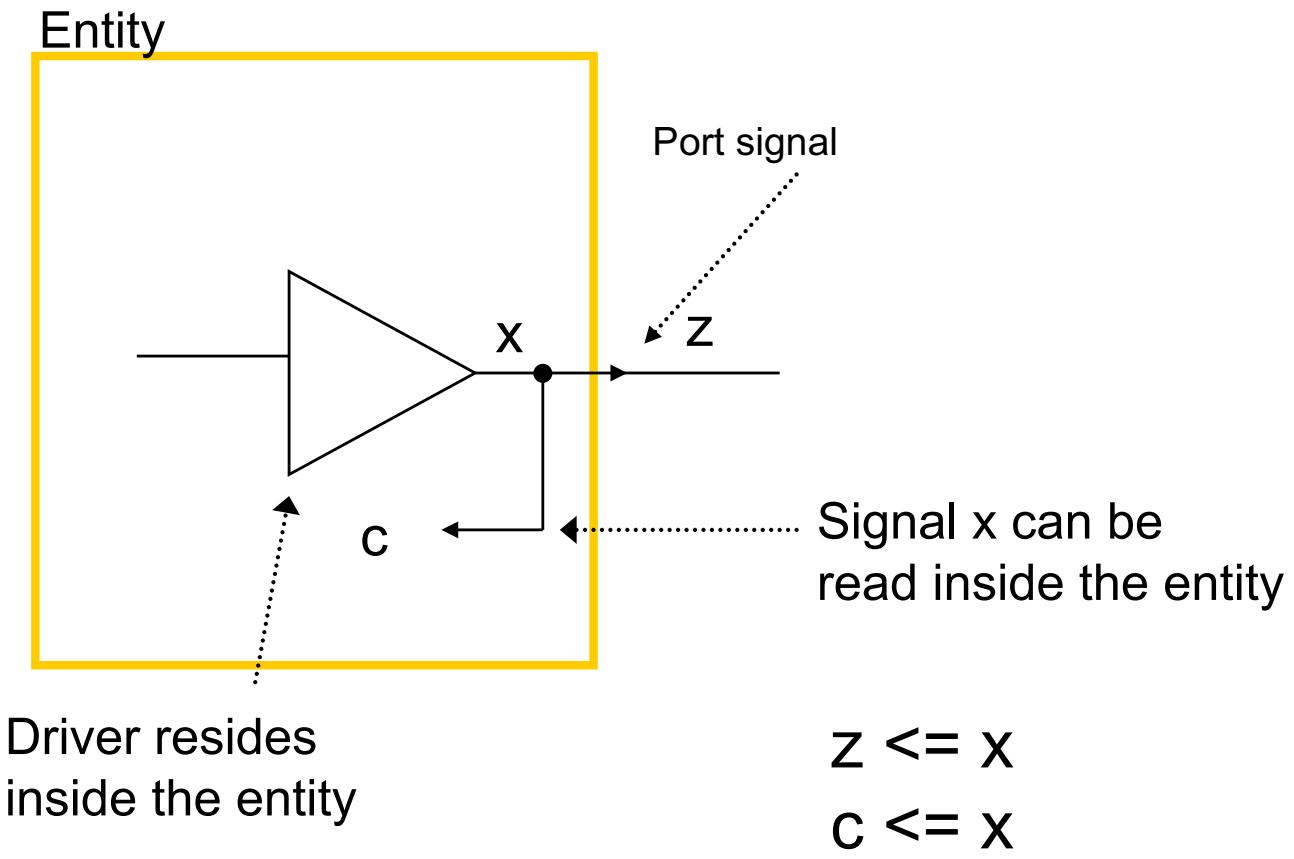
# Port Mode IN



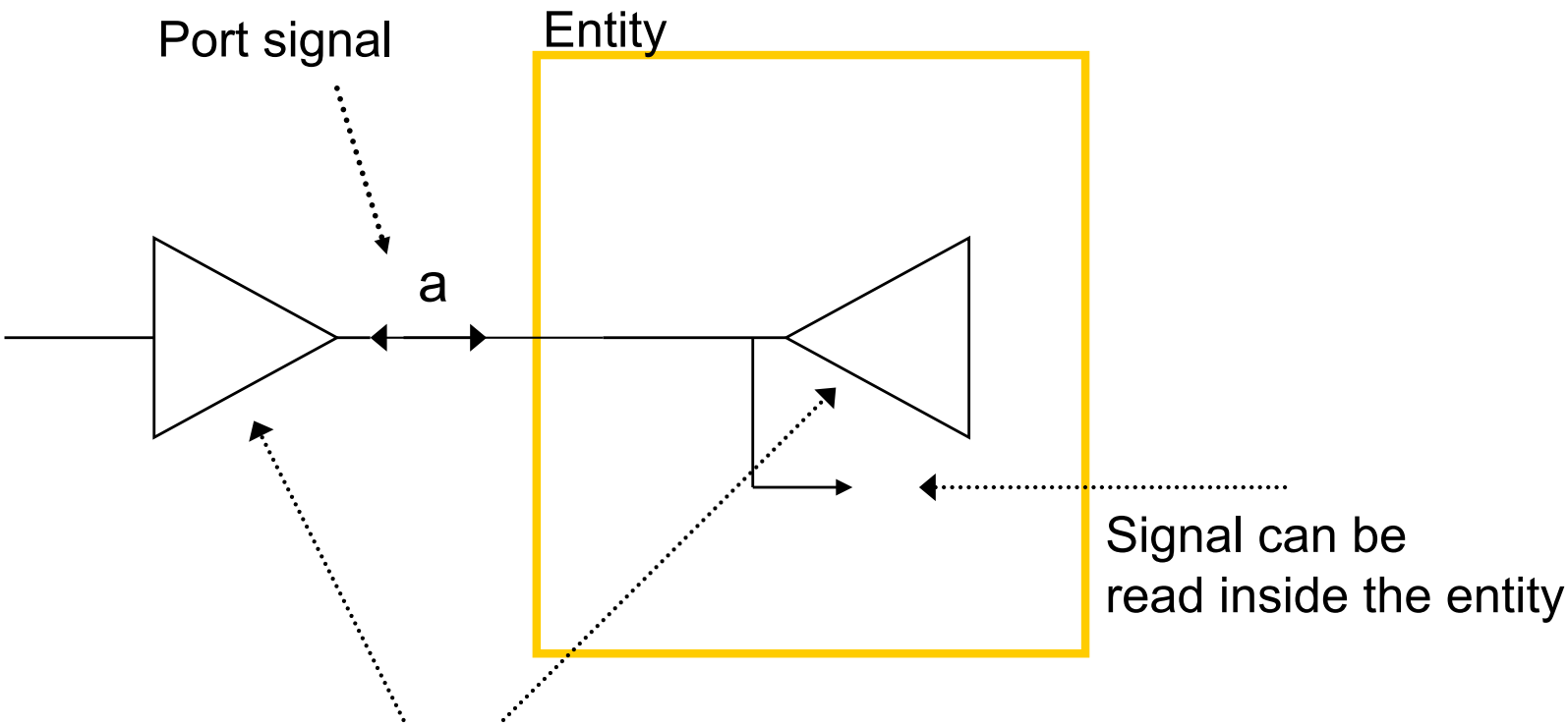
# Port Mode OUT



# Port Mode OUT (with extra signal)



# Port Mode INOUT



# Port Modes - Summary

The *Port Mode* of the interface describes the direction in which data travels with respect to the c*omponent*

- **In**: Data comes into this port and can only be read within the entity. It can appear **only on the right side** of a signal or variable assignment.
- **Out**: The value of an output port can only be updated within the entity. **It cannot be read**. It can only appear **on the left side** of a signal assignment.
- **Inout**: The value of a bi-directional port can be read and updated within the entity model. It can appear on **both sides** of a signal assignment.
- **Buffer:** Used for a signal that is an output from an entity. The value of the signal can be used inside the entity, which means that in an assignment statement the signal can appear on the left and right sides of the <= operator. Not recommended to be used in the synthesizable code.

# Architecture (Architecture body)

- Describes an implementation of a design entity
- Architecture example:

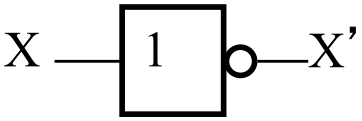
```
ARCHITECTURE model OF nand_gate IS
BEGIN
     z <= a NAND b;
END model;
```

# Architecture – simplified syntax

```
ARCHITECTURE architecture_name OF entity_name IS
    [ declarations ]
BEGIN
     code
END architecture_name;
```

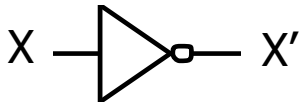
# Other gates

```
ARCHITECTURE arch_not OF not IS
BEGIN
    f<= NOT x;
END arch_not;
```

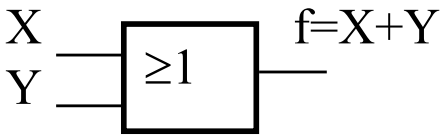



Inverter (not)

| X | f |
|---|---|
| 0 | 1 |
| 1 | 0 |

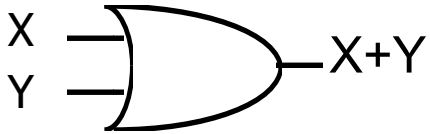


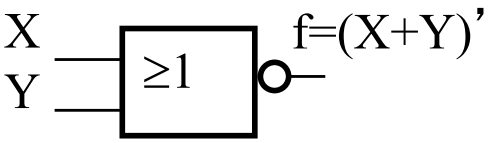

```
ARCHITECTURE arch_or2 OF or2 IS
BEGIN
    f<=x OR y;
END arch_or2;
```




OR

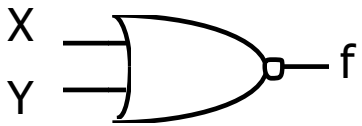
| X | Y | f |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |



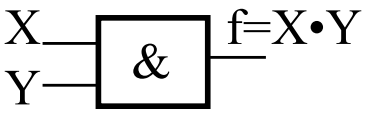



| X | Y | f |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

## NOR

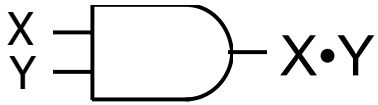



**VHDL:** `f <= x NOR y;`

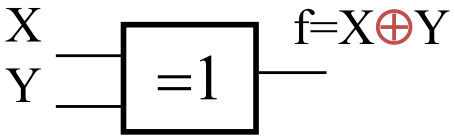



| X | Y | f |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

## AND

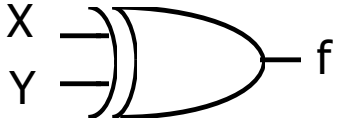



**VHDL:** `f <= x AND y;`

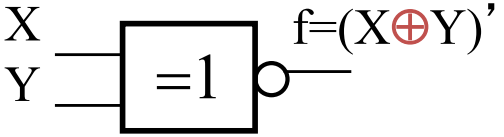



| X | Y | f |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

## XOR

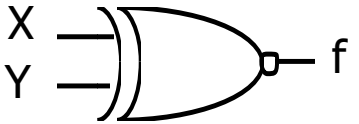



**VHDL:** `f <= x XOR y;`




| X | Y | f |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

## XNOR




f=X' ⊕Y = (X⊕Y)'

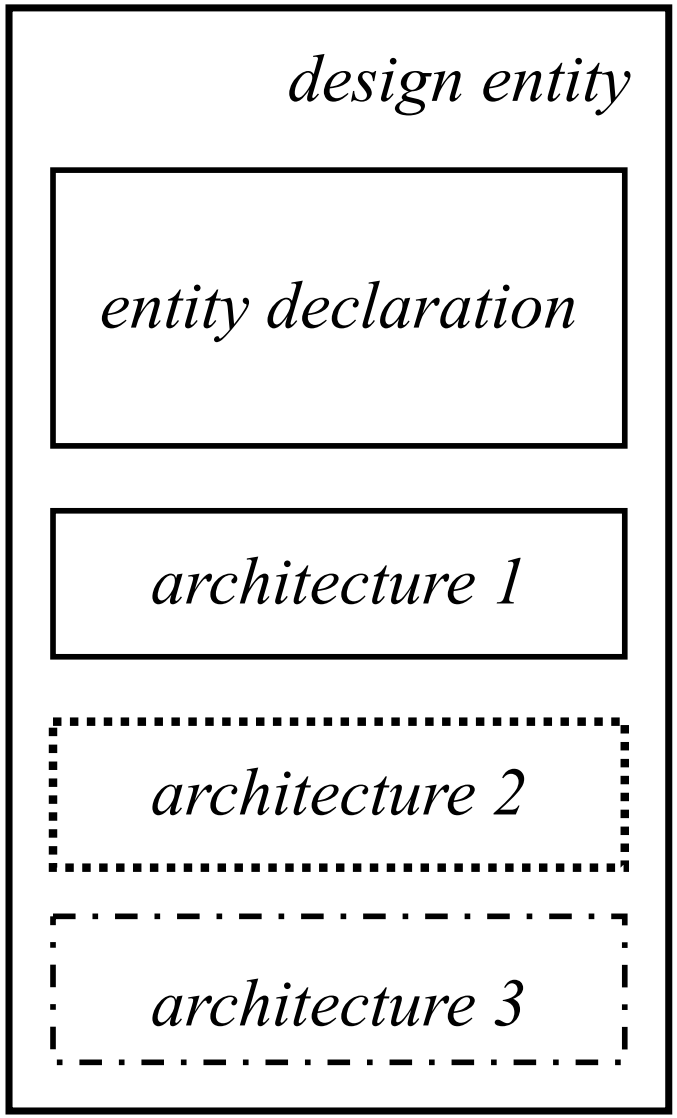
**VHDL:** `f <= x XNOR y;`

# Entity Declaration & Architecture

nand_gate.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY nand_gate IS
    PORT(
        a   : IN STD_LOGIC;
        b   : IN STD_LOGIC;
        z   : OUT STD_LOGIC);
END nand_gate;

ARCHITECTURE dataflow OF nand_gate IS
BEGIN
    z <= a NAND b;
END dataflow;
```

# Design Entity: Entity & Architecture




*Design Entity* - most basic building block of a design.

One *entity* can have many different *architectures.*

# Tips & Hints

Place each entity in a different file.

The name of each file should be exactly the same as the name of an entity it contains.

These rules are not enforced by all tools but are worth following in order to increase readability and portability of your designs

# Tips & Hints

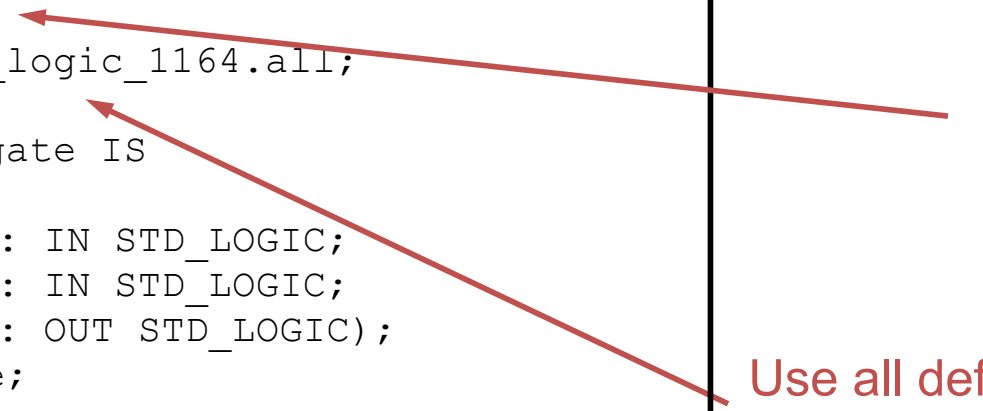
Place the declaration of each port, signal, constant, and variable in a separate line

These rules are not enforced by all tools but are worth following in order to increase readability and portability of your designs

# Libraries

# Library Declarations

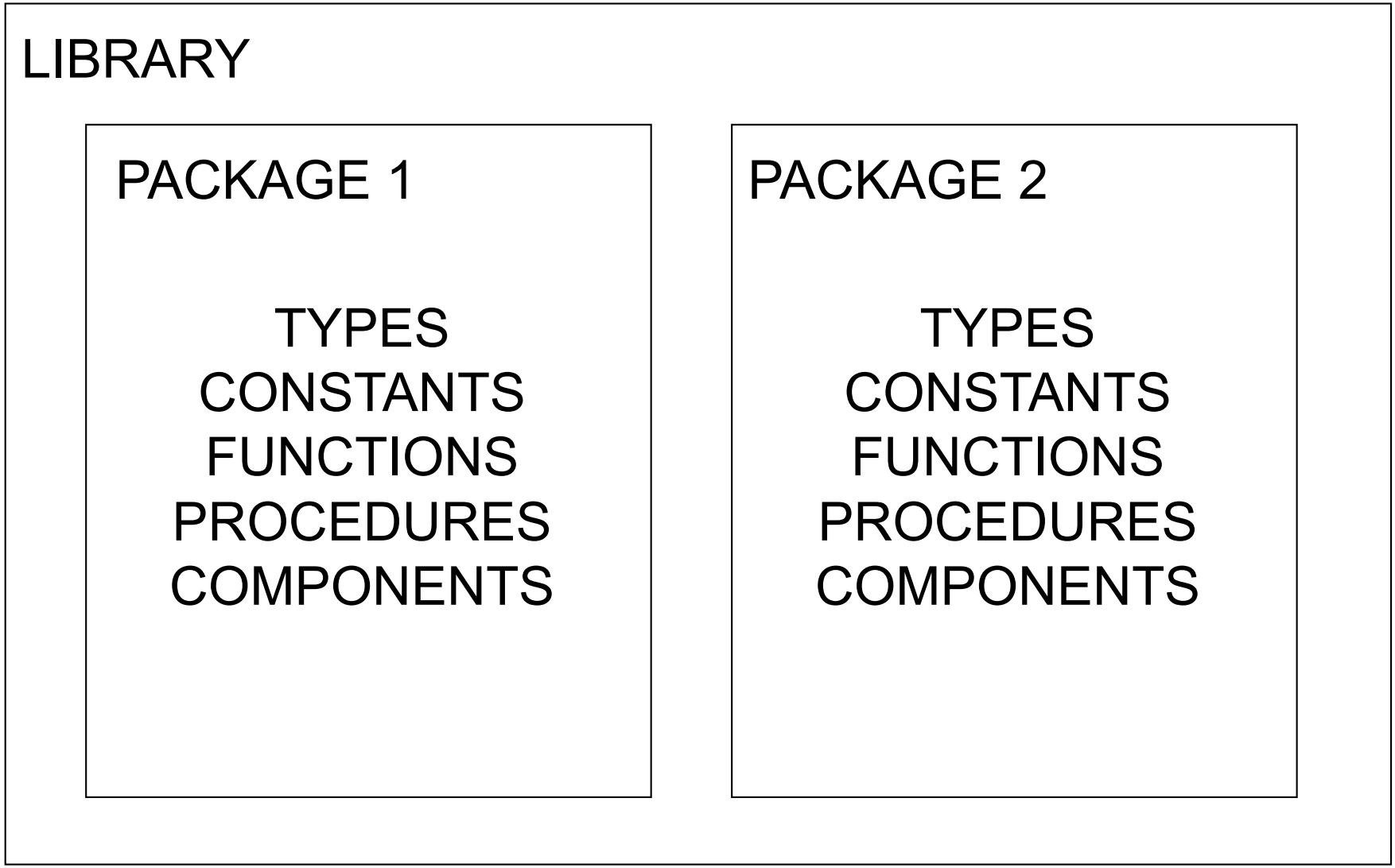
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY nand_gate IS
    PORT(
        a   : IN STD_LOGIC;
        b   : IN STD_LOGIC;
        z   : OUT STD_LOGIC);
END nand_gate;

ARCHITECTURE model OF nand_gate IS
BEGIN
    z <= a NAND b;
END model;
```



# Library declarations - syntax

```
LIBRARY  library_name;
USE library_name.package_name.package_parts;
```

# Fundamental parts of a library



# Libraries

- ieee

  Specifies multi-level logic system, including STD_LOGIC, and STD_LOGIC_VECTOR data types

Need to be explicitly declared

- std

  Specifies pre-defined data types (BIT, BOOLEAN, INTEGER, REAL, SIGNED, UNSIGNED, etc.), arithmetic operations, basic type conversion functions, basic text i/o functions, etc.

- work

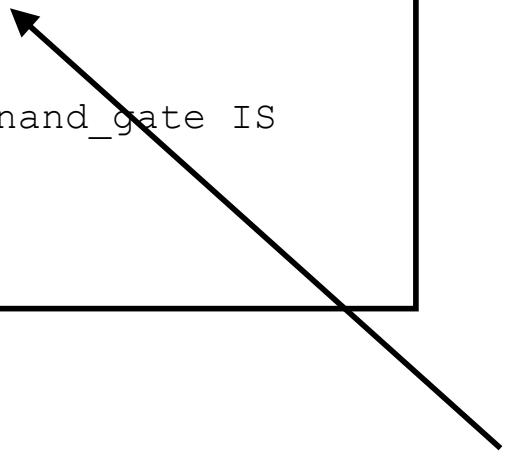
  Holds current designs after compilation

Visible by default

# STD_LOGIC

# STD_LOGIC

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY nand_gate IS
     PORT(
          a    : IN STD_LOGIC;
          b    : IN STD_LOGIC;
          z    : OUT STD_LOGIC);
END nand_gate;

ARCHITECTURE dataflow OF nand_gate IS
BEGIN
     z <= a NAND b;
END dataflow;
```



What is STD_LOGIC?

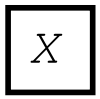
# BIT versus STD_LOGIC

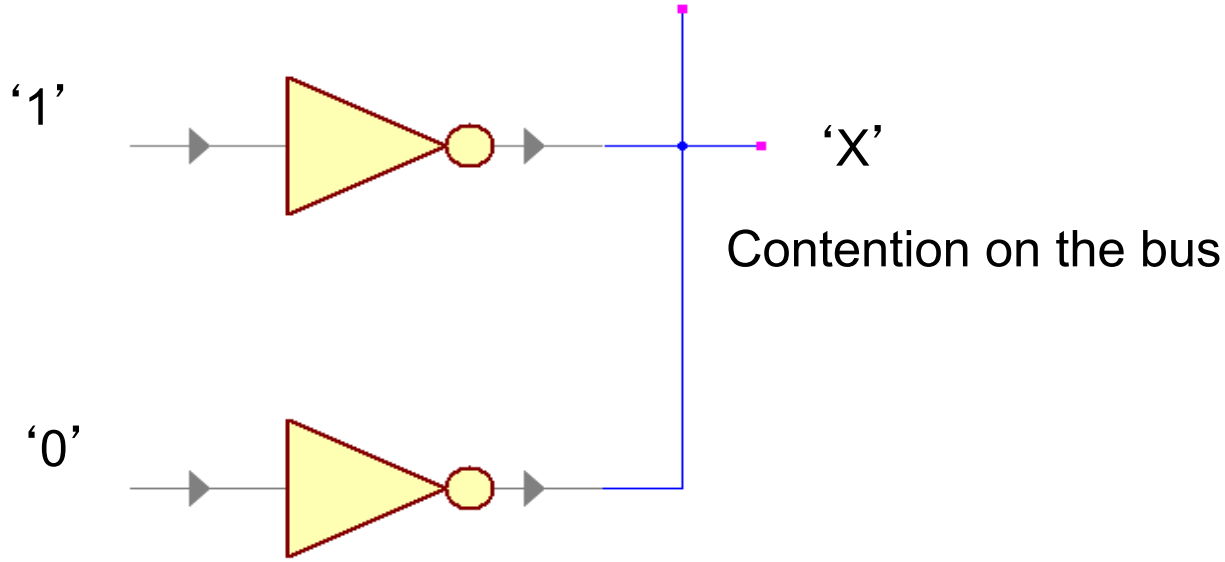
- BIT type can only have a value of ‘0’ or ‘1’
- STD_LOGIC can have eight values
  - ‘0’, ’1’, ’X’, ’Z’, ’W’, ’L’, ’H’, ’-’
  - Useful mainly for simulation
  - ‘0’, ’1’, ’-’, and ‘Z’ are synthesizable (**your codes should contain only these 4 values**)
  - ‘Z’ indicates the off (*high impedance*) state of a tri
  - ‘-’ indicates a don’t care value

# STD_LOGIC *type* demystified

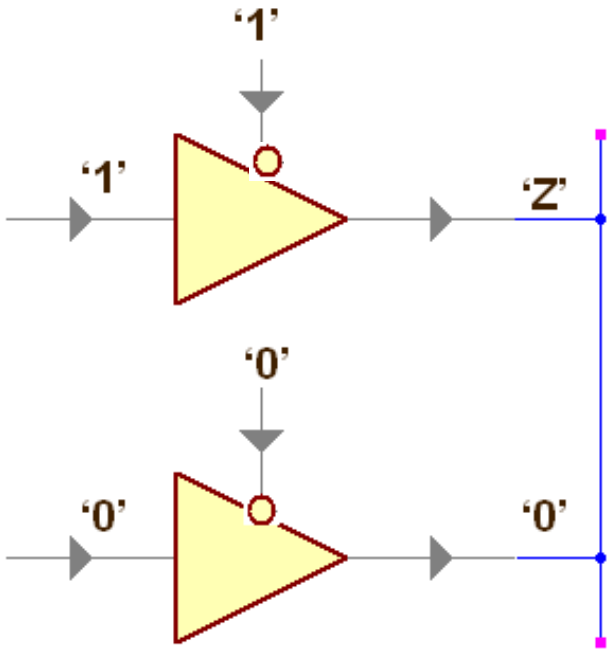
| Value | Meaning |
|---|---|
| ‘X’ | Forcing (Strong driven) Unknown |
| ‘0’ | Forcing (Strong driven) 0 |
| ‘1’ | Forcing (Strong driven) 1 |
| ‘Z’ | High Impedance |
| ‘W’ | Weak (Weakly driven) Unknown |
| ‘L’ | Weak (Weakly driven) 0.<br>Models a pull down. |
| ‘H’ | Weak (Weakly driven) 1.<br>Models a pull up. |
| ‘-’ | Don't Care |

# More on STD_LOGIC Meanings (1)



# More on STD_LOGIC Meanings (2)



# More on STD_LOGIC Meanings (3)

‘_’

- Do not care.
- Can be assigned to outputs for the case of invalid inputs (may produce significant improvement in resource utilization after synthesis).
- Must be used with great caution.

  For example in VHDL, the comparison

  ‘1’ = ‘-’

  gives FALSE.

# Resolving logic levels

| | **X** | **0** | **1** | Z | W | L | H | – |
|---|---|---|---|---|---|---|---|---|
| X | **X** | **X** | **X** | **X** | X | X | X | X |
| 0 | **X** | **0** | **X** | **0** | 0 | 0 | 0 | X |
| 1 | **X** | **X** | **1** | **1** | 1 | 1 | 1 | X |
| Z | **X** | **0** | **1** | **Z** | W | L | H | X |
| W | X | 0 | 1 | W | W | W | W | X |
| L | X | 0 | 1 | L | W | L | W | X |
| H | X | 0 | 1 | H | W | W | H | X |
| – | X | X | X | X | X | X | X | X |

# STD_LOGIC Rules

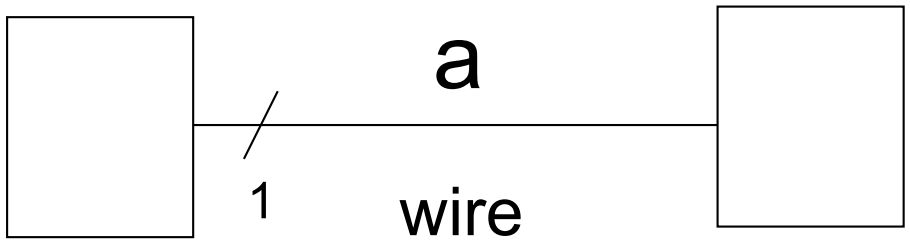
- Use **std_logic** or **std_logic_vector** for all entity input or output ports
  - Do not use integer, unsigned, signed, bit for ports
  - You can use them inside of architectures if desired
  - You can use them in generics
- Instead use std_logic_vector and a conversion function inside of your architecture

  [Consistent with OpenCores Coding Guidelines]

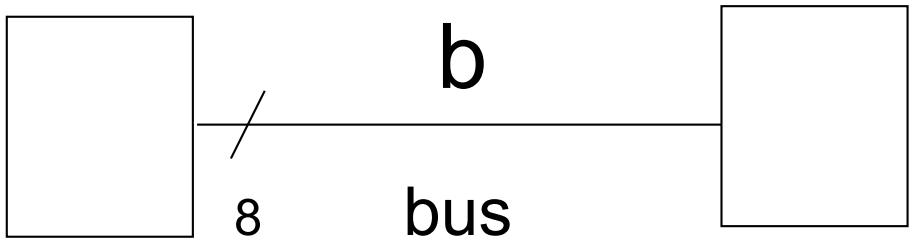
# Modeling Wires and Buses

# Signals

SIGNAL a : STD_LOGIC;




SIGNAL b : STD_LOGIC_VECTOR(7 DOWNTO 0);



# Standard Logic Vectors

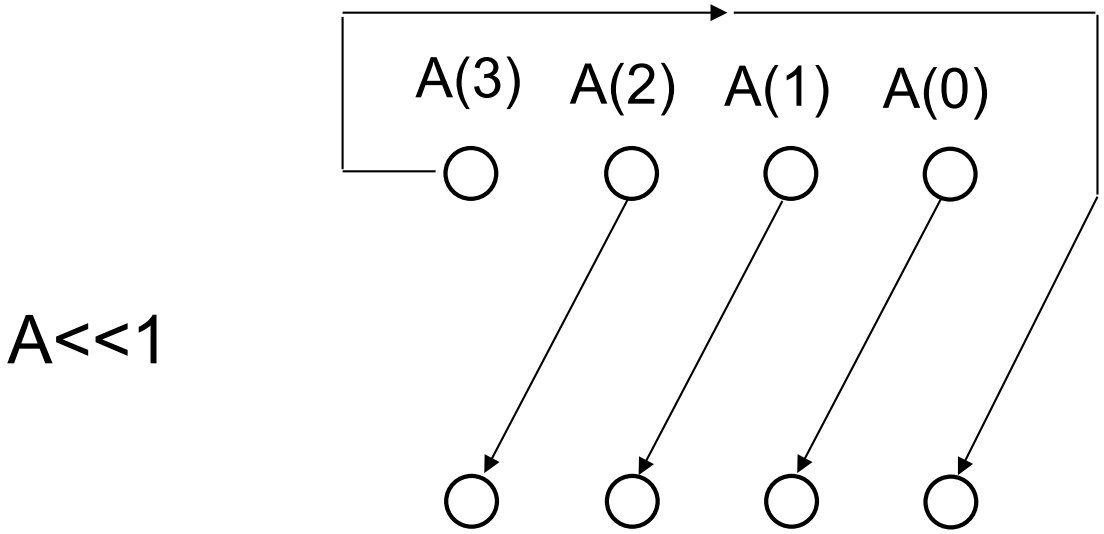
```
SIGNAL a: STD_LOGIC;
SIGNAL b: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL c: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL d: STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL e: STD_LOGIC_VECTOR(8 DOWNTO 0);
                                .....
a <= '1';
b <= "0000";          -- Binary base assumed by default
c <= B"0000";         -- Binary base explicitly specified
d <= X"AF67";         -- Hexadecimal base
e <= O"723";          -- Octal base
```

# Vectors and Concatenation

```
SIGNAL a: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL b: STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL c, d, e: STD_LOGIC_VECTOR(7 DOWNTO 0);

a <= "0000";
b <= "1111";
c <= a & b;                    -- c = "00001111"

d <= '0' & "0001111";          -- d <= "00001111"

e <= '0' & '0' & '0' & '0' & '1' & '1' &
     '1' & '1';
                               -- e <= "00001111"
```

# Fixed Rotation in VHDL

```
SIGNAL A :      STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL ArotL: STD_LOGIC_VECTOR(3 DOWNTO 0);
```

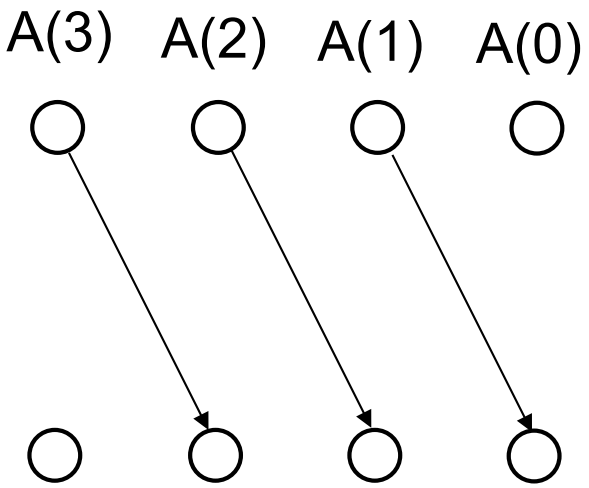



```
ArotL <=  A(2) & A(1) & A(0) & A(3);
```

# Fixed Shift in VHDL

```
SIGNAL A :      STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL AshiftR: STD_LOGIC_VECTOR(3 DOWNTO 0);
```

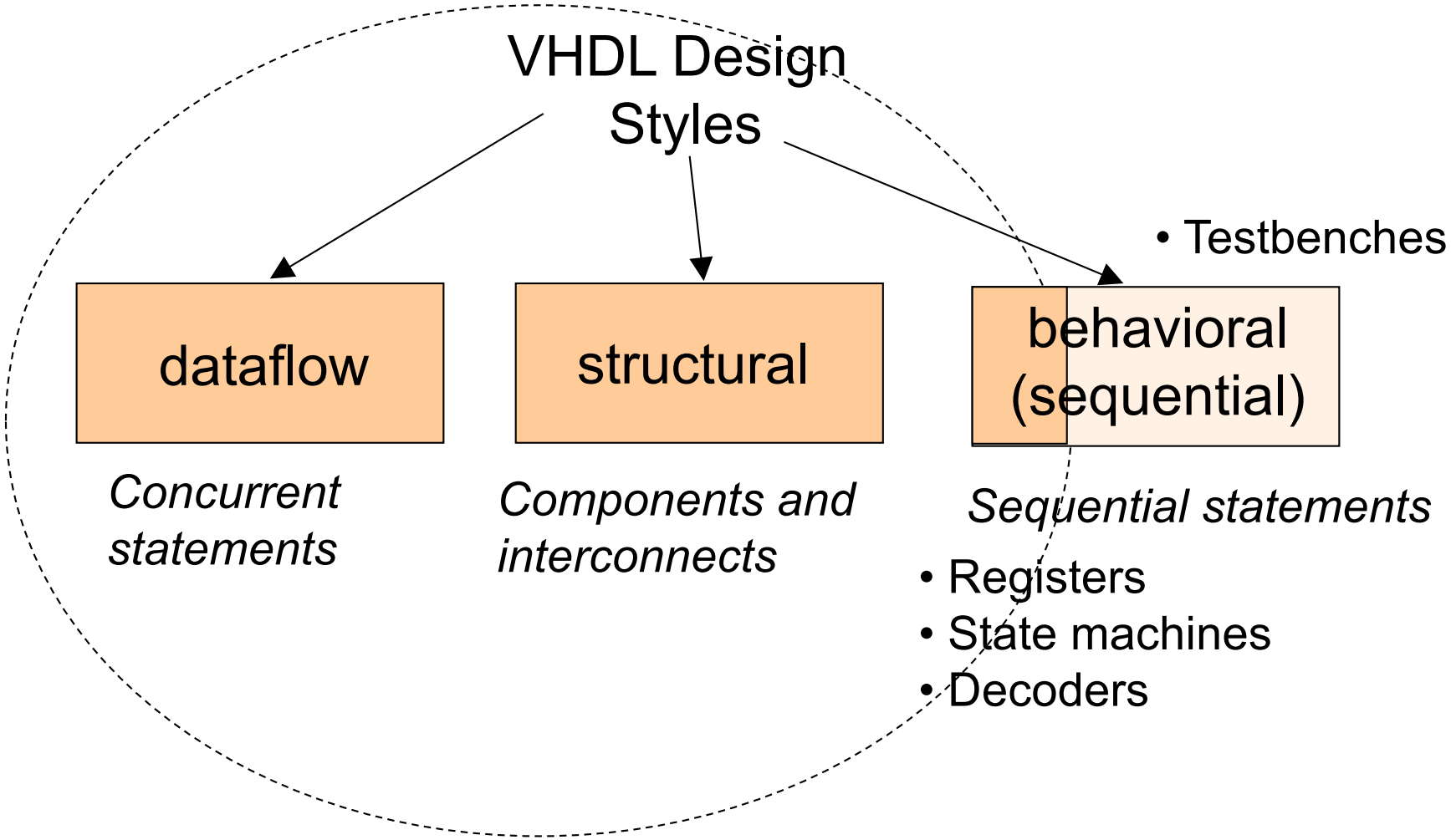
A>>1



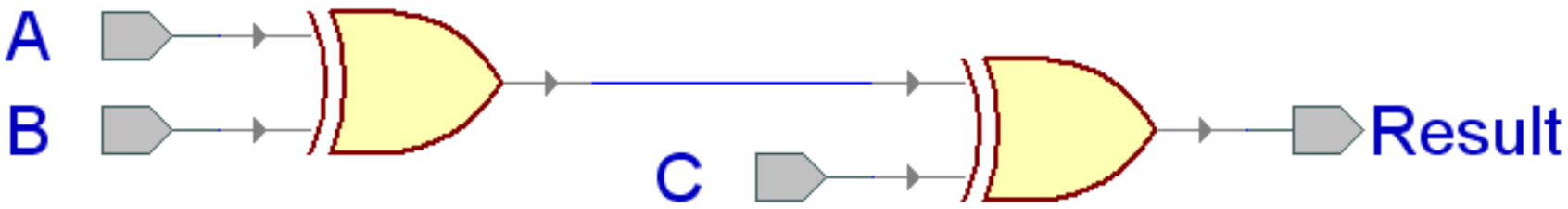

AshiftR <= ‘0’ & A(3) & A(2) & A(1);

# VHDL Design Styles

# VHDL Design Styles



# xor3 Example

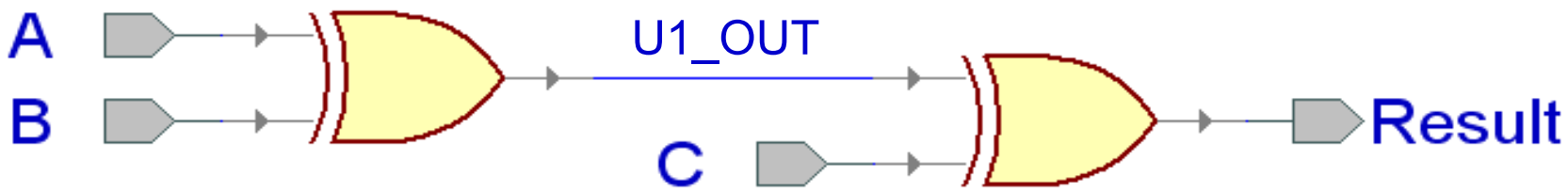


# Entity xor3_gate

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY xor3_gate IS
    PORT(
        A : IN STD_LOGIC;
        B : IN STD_LOGIC;
        C : IN STD_LOGIC;
        Result : OUT STD_LOGIC
    );
end xor3_gate;
```

# Dataflow Architecture (xor3_gate)

```
ARCHITECTURE dataflow OF xor3_gate IS

SIGNAL U1_OUT: STD_LOGIC;

BEGIN
        U1_OUT <= A XOR B;
        Result  <= U1_OUT XOR C;
END dataflow;
```



# Dataflow Description

- Describes how data moves through the system and the various processing steps.
  - Dataflow uses series of concurrent statements to realize logic.
  - Dataflow is most useful style when series of Boolean equations can represent a logic → used to implement simple combinational logic
  - Dataflow code also called "concurrent" code
- Concurrent statements are evaluated at the same time; thus, the order of these statements doesn't matter
  - This is not true for sequential/behavioral statements

This order…

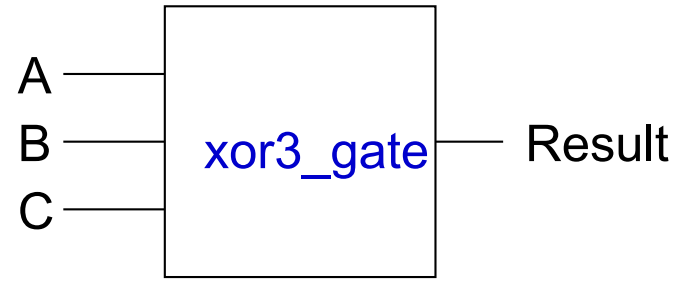
```
U1_out <= A XOR B;
Result <= U1_out XOR C;
```

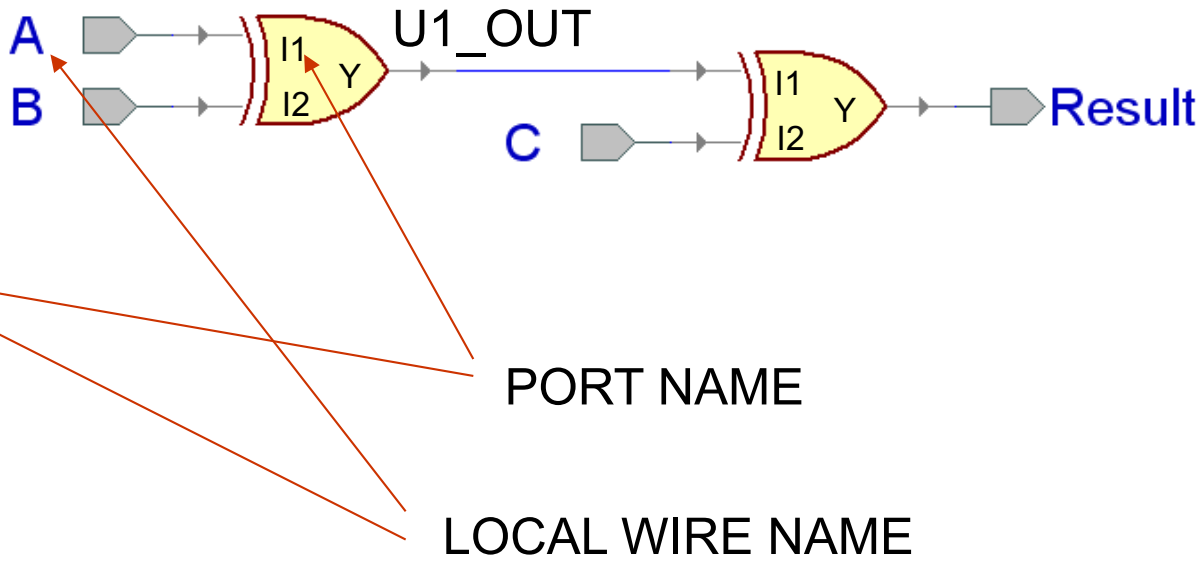
Is the same as this order…

```
Result <= U1_out XOR C;
U1_out <= A XOR B;
```

# Structural Architecture in VHDL 93

```
ARCHITECTURE structural OF xor3_gate IS
SIGNAL U1_OUT: STD_LOGIC;

COMPONENT xor2
      PORT(
            I1 : IN STD_LOGIC;
            I2 : IN STD_LOGIC;
            Y : OUT STD_LOGIC
         );
END COMPONENT;

BEGIN
     U1: entity work.xor2(dataflow)
            PORT MAP (I1 => A,
                      I2 => B,
                       Y  => U1_OUT);

     U2:  entity work.xor2(dataflow)
            PORT MAP (I1 => U1_OUT,
                      I2 => C,
                      Y  => Result);
END structural;
```



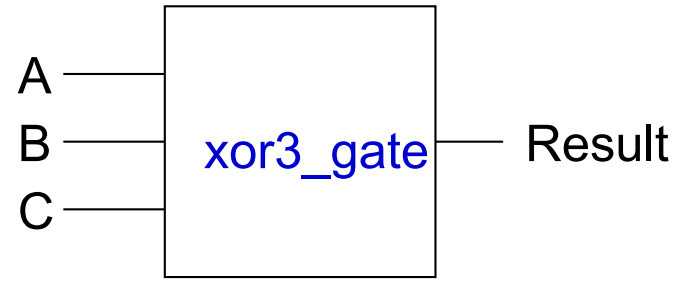
# xor2

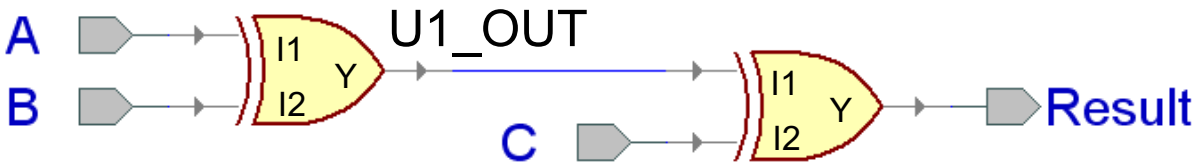
xor2.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY xor2 IS
    PORT(
        I1   : IN STD_LOGIC;
        I2   : IN STD_LOGIC;
        Y    : OUT STD_LOGIC);
END xor2;

ARCHITECTURE dataflow OF xor2 IS
BEGIN
    Y <= I1 xor I2;
END dataflow;
```

# Structural Architecture in VHDL 93

```
ARCHITECTURE structural OF xor3_gate IS
SIGNAL U1_OUT: STD_LOGIC;

COMPONENT xor2
      PORT(
            I1 : IN STD_LOGIC;
            I2 : IN STD_LOGIC;
            Y : OUT STD_LOGIC
         );
END COMPONENT;

BEGIN
     U1: entity xor2(dataflow)
            PORT MAP (A,
                      B,
                      U1_OUT);

     U2:  entity xor2(dataflow)
            PORT MAP (U1_OUT,
                      C,
                      Result);
END structural;
```



# Structural Description

- Structural design is the simplest to understand. This style is the closest to schematic capture and utilizes simple building blocks to compose logic functions.
- Components are interconnected in a **strict** hierarchical manner.
- Structural descriptions may connect simple gates or complex components.
- Structural style is useful when expressing a design that is naturally composed of sub-blocks.

# Behavioral Architecture (xor3 gate)

```
ARCHITECTURE behavioral OF xor3 IS
BEGIN

xor3_behave: PROCESS (A, B, C)
BEGIN
    IF ((A XOR B XOR C) = '1') THEN
        Result <= '1';
    ELSE
        Result <= '0';
    END IF;
END PROCESS xor3_behave;

END behavioral;
```

# Behavioral Description

- It accurately models what happens on the inputs and outputs of the black box (no matter what is inside and how it works).
- This style uses PROCESS statements in *VHDL.*

# Quiz 1-2

http://m.socrative.com/student/#joinRoom

room number: **713113**

- Q1,2,3: Which VHDL style is suitable for:
  - Connecting multiple HW blocks to make a bigger one
  - Describing a HW block in an algorithmic/software style
  - Implementing Boolean functions directly
- Q4: What are the VHDL symbols for:
  a. Logic one
  b. Logic zero
  c. Don't care
  d. Unknown
  e. High Impedance

Note: put the symbols in the correct sequence separated with a space, e.g. @ # $ % &

- Q5: which one of the following namings are valid in VHDL:

  a. 7segment_display
  b. A87372477424
  c. Adder/Subtractor
  d. /reset
  e. And_or_gate
  f. AND__OR__NOT
  g. Kogge-Stone-Adder
  h. Ripple&Carry_Adder
  i. My adder

# Valid or invalid?

~~7segment_display~~

A87372477424

~~Adder/Subtractor~~

~~/reset~~

And_or_gate

~~AND__OR__NOT~~

~~Kogge-Stone-Adder~~

~~Ripple&Carry_Adder~~

~~My adder~~

# Summary of Lecture 1

- Why is hardware design important
- The general flow for designing hardware
  - ASICs vs. FPGAs
- VHDL basics
  - General VHDL format for describing a digital circuit
    - Entity, architecture, ports
  - Describing gates in VHDL
  - Libraries
  - Wires and busses

- Reading (complimentary to the slide lectures):
  - Chapter 2

- Next Lecture 2 (this Wednesday):
  - Boolean functions and minimization
  - Design of Adders
    - **to be used for next week's Lab**

# Backup slides

# Brief History of VHDL

# VHDL

- VHDL is a language for describing digital hardware used by industry worldwide
  - **VHDL** is an acronym for **V**HSIC (**V**ery **H**igh **S**peed **I**ntegrated **C**ircuit) **H**ardware **D**escription **L**anguage

# Genesis of VHDL

State of art 1980

- Multiple design entry methods and hardware description languages in use
- No or limited portability of designs between CAD tools from different vendors
- Objective: shortening the time from a design concept to implementation from 18 months to 6 months

# Genesis of VHDL

State of art 1980

- Multiple design entry methods and hardware description languages in use
- No or limited portability of designs between CAD tools from different vendors
- Objective: shortening the time from a design concept to implementation from 18 months to 6 months

# A Brief History of VHDL

- June 1981: Woods Hole Workshop
- July 1983: contract awarded to develop VHDL
  - Intermetrics
  - IBM
  - Texas Instruments
- August 1985: VHDL Version 7.2 released
- December 1987:

  VHDL became IEEE Standard 1076-1987 and in 1988 an ANSI standard

# Four versions of VHDL

- Four versions of VHDL:
  - IEEE-1076 1987
  - IEEE-1076 1993 ← most commonly supported by CAD tools
  - IEEE-1076 2000 (minor changes)
  - IEEE-1076 2002 (minor changes)

# Verilog

# Verilog

- Essentially identical in function to VHDL
  - No generate statement
- Simpler and syntactically different
  - C-like

- Gateway Design Automation Co., 1985
- Gateway acquired by Cadence in 1990
- IEEE Standard 1364-1995
- Early *de facto* standard for ASIC programming

- Programming language interface to allow connection to non-Verilog code

# VHDL vs. 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 |

# Features of VHDL and Verilog

- Technology/vendor independent
- Portable
- Reusable