UNIVERSITÀ DEGLI STUDI DI TRIESTE

Dipartimento di
Fisica
Dipartimento d'Eccellenza 2023-2027

VHDL
Part 1

Laboratorio di acquisizione e controllo dati

UniTs a.a. 2025-2026

Introduction to VHDL

- VHDL is used to describe **hardware** → **Concurrent** language
  - Code lines are executed all at once
  - It helps to be able to envisage how the final circuit looks like
  - Writing good code means designing a circuit that will work

- VHDL is used to **model** digital circuits
  - Design, testing, implementation in hardware (e.g. in an FPGA)
  - Different levels of abstraction

- Implementation of a VHDL-based system
  - Code writing; Compiling; Simulation; Synthesis
  - Software and hardware tools provided by FPGA manufacturers

VHDL coding style tips

- Make the code readable
  - Indentation, self-describing identifiers, comments, consistent style
- **Do not write VHDL code with high-level language style**
  - Resulting circuits will not work well; lots of unnecessary hardware, inefficient
- Keep the **VHDL model simple and clear** (does not mean short)
  - A short VHDL code does not mean a better implementation in HW
- Remember: you are designing hardware
  - **Partition the circuit** you want to implement into smaller units, with less functionality
  - **Draw a diagram** of the circuit you want to implement

VHDL basics

- VHDL is case insensitive
- VHDL is insensitive to white spaces
- Comments begin with “--” and do not span multiple lines
- Every VHDL statement ends with a semicolon
- Parentheses are used to make the code more readable and to specify operators' precedence

- Identifiers (i.e. name given to various items in VHDL)
  - Only letters, digits (0-9), “_”
  - Must start with alphabetic character
  - Must not end with underscore
  - Must not have two consecutive underscores

- Reserved words
  - Words that have a special meaning
  - Cannot be used as identifiers

Table A.1 provides a complete list of VHDL reserved words.

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

Table A.1: A complete list of VHDL reserved words.

Data objects

- Data objects: **signals**, **variables**, **constants**
  - The signal represents a wire
  - The variable is used to store information
  - The constant is a variable with a fixed value

- Objects must be **declared** before using them
  - For each object, specific syntax and places in the code to declare it

- All objects in VHDL must have a **type**
  - Objects can only have **value** of that type
  - Operations are allowed only between same-type objects

Data types

- VHDL provides **built-in** data types
- Built-in data-types work well for simulations but are not enough to describe an actual circuit
- Types can be extended using external **libraries** (Third parties or User-defined)
- The most widely used VHDL library is the **IEEE library**

Examples of built-in data types	
integer	whole numbers
real	floating point numbers
time	used to specify delays - important in simulation
bit	scalar 1-bit signal (allowed values ‘0’ or ‘1’)
bit_vector	multiple bits (buses) signals
boolean	boolean number (true or false)

IEEE library

- The IEEE library is one of the most common library used in VHDL designs

- **Standardized data types** for the representation of **logic signals**
  - **std_logic** for single bit signals
  - **std_logic_vector** for multiple-bit signals
  - Additional logic values to model three-state buffers, pull-up and pull-down networks, high impedance state, etc.

Additional logic values	
0	logic 0
1	logic 1
U	unitialised
X	unknown logic value
Z	high-impedance
W	weak signal
L	weak low
H	weak high
-	don't care

VHDL building blocks

- **Library**: Imports data types and functions
- **Entity**: Declares the digital block and defines the **interface** to the outside world through **port statements**
- **Architecture**: Implements the **functionality** of the block through **concurrent statements**
  - There are four types of concurrent statements
    - **Concurrent** signal assignment statement
    - **Conditional** signal assignment statement
    - **Selected** signal assignment statement
    - **Process** statement

Note: statements represent finite quantities of actions to be taken

Library

```vhdl
-- EXAMPLE: VHDL CODE FOR OR GATE WITH COMMENTS

-- library declaration
    -- to import definitions (types, functions, etc.) from a library, use the "library" statement
    -- to include a package from a library, use the "use" statement
        -- the package "std_logic_1164" is needed to import the data types "std_logic" and "std_logic_vector"

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


-- entity

    -- entity declaration syntax
        -- each entity has an identifier (e.g. a name)
entity or_gate is

    -- Port(); contains the list of signals that belong to the entity, that interface to the external world
        -- Each line in Port(); is a statement declaring a signal
Port (
    -- sintax for signal declaration in port
        -- <signal name> : mode type;
    A : in STD_LOGIC;
    B : in STD_LOGIC;
    Y : out STD_LOGIC  -- no semicolumn after the last port definition
);
end or_gate;

-- architecture

    -- architecture declaration syntax
        -- each architecure has an identifier (e.g. a name)
architecture or_arch of or_gate is

    -- declarative part between the "is" and the "begin" keywords
        -- empty in this example

begin
    -- implementation part between the "begin" and the "end" keywords
        -- this is where the functionality of the entity is described

    -- concurrent signal assignment statement
        -- <signal name> <= <expression>;
Y <= A or B;

        -- the signal assignment operator "<=" is used to assign a new value to a signal
            -- it specifies the relationship between signals
            -- the signal on the left side of the signal assignment operator is dependent upon the signals on the right side of the operator

        -- "or" is a reserved word in VHDL

end or_arch;
```

Entity

```vhdl
-- EXAMPLE: VHDL CODE FOR OR GATE WITH COMMENTS

-- library declaration
    -- to import definitions (types, functions, etc.) from a library, use the "library" statement
    -- to include a package from a library, use the "use" statement
        -- the package "std_logic_1164" is needed to import the data types "std_logic" and "std_logic_vector"

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- entity

    -- entity declaration syntax
        -- each entity has an identifier (e.g. a name)
entity or_gate is

    -- Port(); contains the list of signals that belong to the entity, that interface to the external world
        -- Each line in Port(); is a statement declaring a signal
Port (
    -- sintax for signal declaration in port
        -- <signal name> : mode type;
    A : in STD_LOGIC;
    B : in STD_LOGIC;
    Y : out STD_LOGIC  -- no semicolumn after the last port definition
);
end or_gate;

-- architecture

    -- architecture declaration syntax
        -- each architecure has an identifier (e.g. a name)
architecture or_arch of or_gate is

    -- declarative part between the "is" and the "begin" keywords
        -- empty in this example

begin
    -- implementation part between the "begin" and the "end" keywords
        -- this is where the functionality of the entity is described

    -- concurrent signal assignment statement
        -- <signal name> <= <expression>;
Y <= A or B;

        -- the signal assignment operator "<=" is used to assign a new value to a signal
            -- it specifies the relationship between signals
            -- the signal on the left side of the signal assignment operator is dependent upon the signals on the right side of the operator

        -- "or" is a reserved word in VHDL

end or_arch;
```

Architecture

```vhdl
-- EXAMPLE: VHDL CODE FOR OR GATE WITH COMMENTS

-- library declaration
    -- to import definitions (types, functions, etc.) from a library, use the "library" statement
    -- to include a package from a library, use the "use" statement
        -- the package "std_logic_1164" is needed to import the data types "std_logic" and "std_logic_vector"

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


-- entity

    -- entity declaration syntax
        -- each entity has an identifier (e.g. a name)
entity or_gate is

    -- Port(); contains the list of signals that belong to the entity, that interface to the external world
        -- Each line in Port(); is a statement declaring a signal
Port (
    -- sintax for signal declaration in port
        -- <signal name> : mode type;
    A : in STD_LOGIC;
    B : in STD_LOGIC;
    Y : out STD_LOGIC  -- no semicolumn after the last port definition
);
end or_gate;

-- architecture

    -- architecture declaration syntax
        -- each architecure has an identifier (e.g. a name)
architecture or_arch of or_gate is

    -- declarative part between the "is" and the "begin" keywords
        -- empty in this example

begin
    -- implementation part between the "begin" and the "end" keywords
        -- this is where the functionality of the entity is described

    -- concurrent signal assignment statement
        -- <signal name> <= <expression>;
Y <= A or B;

        -- the signal assignment operator "<=" is used to assign a new value to a signal
            -- it specifies the relationship between signals
            -- the signal on the left side of the signal assignment operator is dependent upon the signals on the right side of the operator

        -- "or" is a reserved word in VHDL

end or_arch;
```

VHDL test bench

- A test bench is a **VHDL code written to test the code** before implementing it in an FPGA
  - The test bench can be seen as a test setup that connects stimuli to the inputs of the Device Under Test (DUT) and observes the output of the DUT (the DUT is the entity you want to test)

- VHDL code in the testbench will never be synthesized
  - **Non synthesizable code** can be used, typically makes simulation easier and better

Note: In this course we will not delve into how to code test benches. Test benches for most examples are provided and only briefly discussed

Test bench (1/2)

```vhdl
-- EXAMPLE: TEST BENCH FOR OR GATE WITH COMMENTS

-- library
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- entity
    -- test bench entity typically empty
entity or_gate_tb is
end or_gate_tb;

-- architecture
architecture or_gate_tb_arch of or_gate_tb is

-- declative part

    -- component declaration
        -- this is the entity you want to test
component or_gate
Port (
A : in STD_LOGIC;
B : in STD_LOGIC;
Y : out STD_LOGIC
);
end component;

    -- declaration of test bench signals
        -- sintax for signal declaration in declaritive part of the architecture
            -- signal <signal_name> : type := <initial value>;
            -- Note: in a test bench it is typical to assign a value to the signal at declaration; this is not done in the code to be synthesized in hardware (more in another example)
signal A_tb : STD_LOGIC := '0';
signal B_tb : STD_LOGIC := '0';
signal Y_tb : STD_LOGIC;

begin
-- implementation part

    -- component instantiation
        -- this is where the test bench signals are connected to the device under test (DUT) that is your entity
        -- imagine this as connecting a pulse generator to A and B and an oscilloscope to Y
dut: or_gate
port map (
A => A_tb,
B => B_tb,
Y => Y_tb
);

```

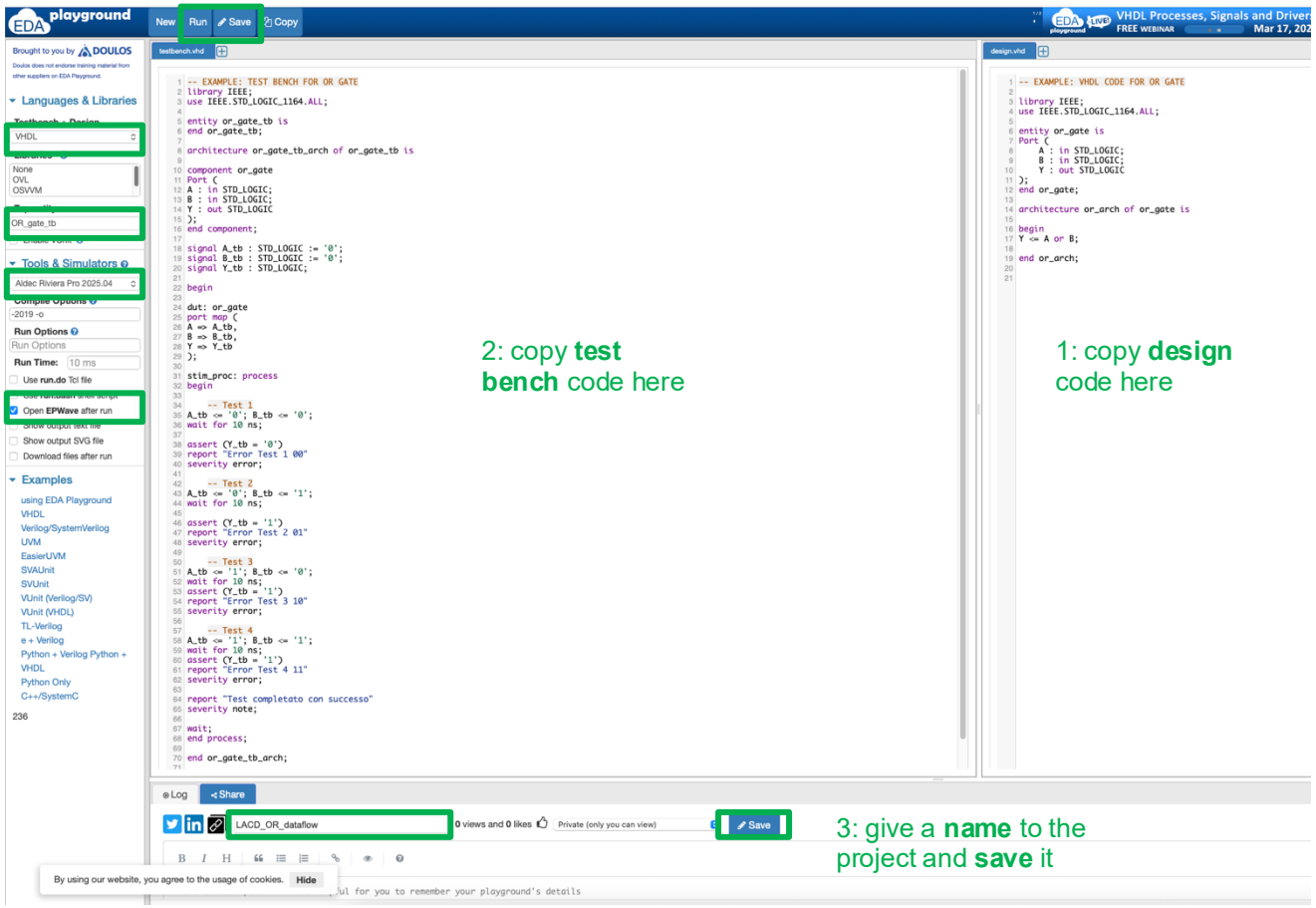
Test bench (2/2)

```
47
48      -- process statement - statements within process are exacuted sequentially, more on this later
49          -- this is the part where stimuli are applied to A and B and the output Y is observed
50  stim_proc: process
51  begin
52
53      -- Test 1
54  A_tb <= '0'; B_tb <= '0'; -- sequential signal assignment statement (more on this later)
55  wait for 10 ns; -- wait statements are used in test benches as delays to sequence inputs
56                  -- wait is non synthesizable code
57
58  assert (Y_tb = '0')
59  report "Error Test 1 00"
60  severity error;
61
62      -- Test 2
63  A_tb <= '0'; B_tb <= '1';
64  wait for 10 ns;
65
66  assert (Y_tb = '1')
67  report "Error Test 2 01"
68  severity error;
69          -- use the assert function to check signal values against some expectation
70          -- assert returns always a boolean value
71          -- default severity is error
72
73      -- Test 3
74  A_tb <= '1'; B_tb <= '0';
75  wait for 10 ns;
76  assert (Y_tb = '1')
77  report "Error Test 3 10"
78  severity error;
79
80      -- Test 4
81  A_tb <= '1'; B_tb <= '1';
82  wait for 10 ns;
83  assert (Y_tb = '1')
84  report "Error Test 4 11"
85  severity error;
86
87  report "Test completato con successo"
88  severity note;
89
90  wait; -- stopping all stimuli at the end of the process
91  end process;
92
93  end or_gate_tb_arch;
94
```

Exercise 1

- Simulate the OR gate example in EDA playground

- Instructions



Exercise 2

- Write the code for a 4-input AND gate and simulate it

- Instructions
  - Copy the OR gate project, change the name, save it
  - Modify the design code and the test bench code
  - Run the simulation

Intermediate signals

```vhdl
-- EXAMPLE: VHDL CODE FOR OR GATE WITH INTERNAL SIGNAL

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity or_gate is
Port (
    A : in STD_LOGIC;
    B : in STD_LOGIC;
    Y : out STD_LOGIC
);
end or_gate;

architecture or_arch of or_gate is

    -- declaration of an intermediate signal
        -- no link to the outside world, internal to the entity
        -- intermediate signals are often used in VHDL especially when modelling complex logic (not the case of this example)
            -- partition the function of complex logic into simpler functions based on the underlying hardware
            -- model circuits in the simplest way
            -- simple circuits have a higher probability of being synthesized correctly
            -- a short VHDL model is not necessarily the simplest
            -- longer VHDL code does not mean larger and more complex hardware

signal s1_or : STD_LOGIC;
    -- syntax for signal declaration
            -- signal <signal_name> : type := <initial value>;
            -- note that no initial value is assigned, the synthesizer would ingore it

begin
    -- concurrent signal assignment statements
    -- executed at the same time, order in which they are written does not matter
    -- can you see why this is the case?

s1_or <= A or B;

Y <= s1_or;

end or_arch;

```

Concurrent statements

- **Concurrent** signal assignment statement
- **Conditional** signal assignment statement
- **Selected** signal assignment statement
- **Process** statement

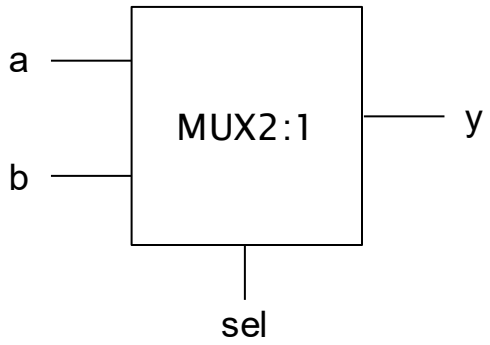
Concurrent signal assignment statements

```
-- CONCURRENT SIGNAL ASSIGNMENT STATEMENT
    -- Syntax for concurrent signal assignment statement
    <target_signal> <= <expression>;

    -- A concurrent signal assignment statement is executed any time there is a change in any of the signals listed on the right-hand side of the signal assignment operator
    -- The <target_signal> gets the value of the <expression> (constant, signal, operators acting on other signals)

    -- All the exercises seen up to now used concurrent signal assignment statements, e.g "Y <= A or B;" is a concurrent signal assignment statement,"Y <= A and B and C and D;" is a concurrent signal assignment statement
```

Conditional signal assignment statement

```
-- CONDITIONAL SIGNAL ASSIGNMENT STATEMENT
    -- Syntax for conditional signal assignment statement
    <target_signal> <=  <expression> when <condition> else
                        <expression> when <condition> else
                        <expression>;

    -- One target signal, multiple expressions with an associated condition

    -- A conditional signal assignment statement is executed any time a change occurs in the conditional signals
        -- Conditions are evaluated sequentially until one is evaluated as true
        -- The associated expression is evaluated and assigned to the target

    -- There is only one signal assignment operator for each conditional signal assignment statement

    -- The last expression in the conditional signal assignment statement is the catch-all condition
        -- If none of the conditions listed above the final expression evaluates as true, the last expression is assigned to the target

    -- Generally it is best to provide all the options in the conditional signal assignment statement and do rely on a catch all statement for intended signal assignment
```

MUX2:1 with conditional signal assignment statement



sel	a	b	y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

```vhdl
-- EXAMPLE: VHDL CODE FOR MUX2:1
-- WITH CONDITIONAL SIGNAL ASSIGNMENT STATEMENTS

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux2to1 is
    Port (
        a   : in  STD_LOGIC;
        b   : in  STD_LOGIC;
        sel : in  STD_LOGIC;
        y   : out STD_LOGIC
    );
end mux2to1;

architecture mux2to1_arch of mux2to1 is
begin
    -- conditional signal assignment statement
    y <= a when sel = '0' else
         b when sel = '1' else
         '0';

        -- this way of writing the MUX2:1 lists all possible conditions of SEL
            -- it is recommended to provide all the options in a conditional signal assignment statement
            -- '0' is a catch-all statement
                -- if none of the previous conditions is evaluated as true, then this statement is executed

        -- the conditional signal assignment statement could have been written as
            -- y <= a when sel = '0' else b;
            -- where else b is a catch-all statement
            -- this is a catch-all statement for intented signal assignment

        -- when using the relational operator =, single quotes are used to describe values associated with one-bit signals, as SEL in this example
end mux2to1_arch;

```

Exercise 3

- Simulate the **MUX2:1** in EDA playground

Exercise 4

- Write the code for a **MUX4:1** block with **conditional** signal assignment statement and simulate it in EDA playground

- Instructions
  - Use **std_logic_vector** for SEL
    - **SEL : in std_logic_vector (1 downto 0);**
  - You will need to use the **relational operator = for vectors**
    - **<signal_name> = "<values>"**

Selected signal assignment statement

```
-- SELECTED SIGNAL ASSIGNMENT STATEMENT
    -- Syntax for selected signal assignment statement
    with <choose_expression> select
    <target_signal> <=  <expression> when <choices>,
                        <expression> when <choices>,
                        <expression> when others;

    -- A selected signal assignment statement is evaluated each time there is a change in the <chooser_expression>
    -- Assignments are based upon the evaluation of one <expression>

    -- There is only one signal assignment operator for each selected signal assignment statement

    -- Generally it is best to to include all the expected cases in the selected signal assignment statement followed by the "when others" clause
```

Exercise 5

- Write the code for a **MUX4:1** block with **selected** signal assignment statement and simulate it in EDA playground

Summary part 1

- VHDL is used to model digital circuits that operate in parallel

- Building blocks
  - **Entity** – interface description
  - **Architecture** – functionality description

- Concurrent statements
  - **Concurrent** signal assignment statement
  - **Conditional** signal assignment statement
  - **Selected** signal assignment statement
  - **Process** statement

Summary part 1

- **IEEE** library and **IEEE_std_logic.1164** package to include data type **std_logic** and **std_logic_vector**

- **Signals**
  - Declared in **port** (**interface** signals) and **declarative** part of architecture (**intermediate** signals)
  - In simulations, an **initial value** can be assigned to a signal **at declaration** with the **assignment operator :=**
  - **New values** are assigned to signals with the **signal assignment operator <=**

- **Relational operator =**
  - One-bit signals: **<signal_name> = '<value>'**
  - Buses: **<signal_name> = "<values>"**

Summary part 1

- Exercises
  - 1: two input **OR** gate with **concurrent** signal assignment statement
  - 2: four input **AND** gate with **concurrent** signal assignment statement
  - 3: **MUX2:1** with **conditional** signal assignment statement
  - 4: **MUX4:1** with **conditional** signal assignment statement
  - 5: **MUX4:1** with **selected** signal assignment statement