Introduzione alle FPGA

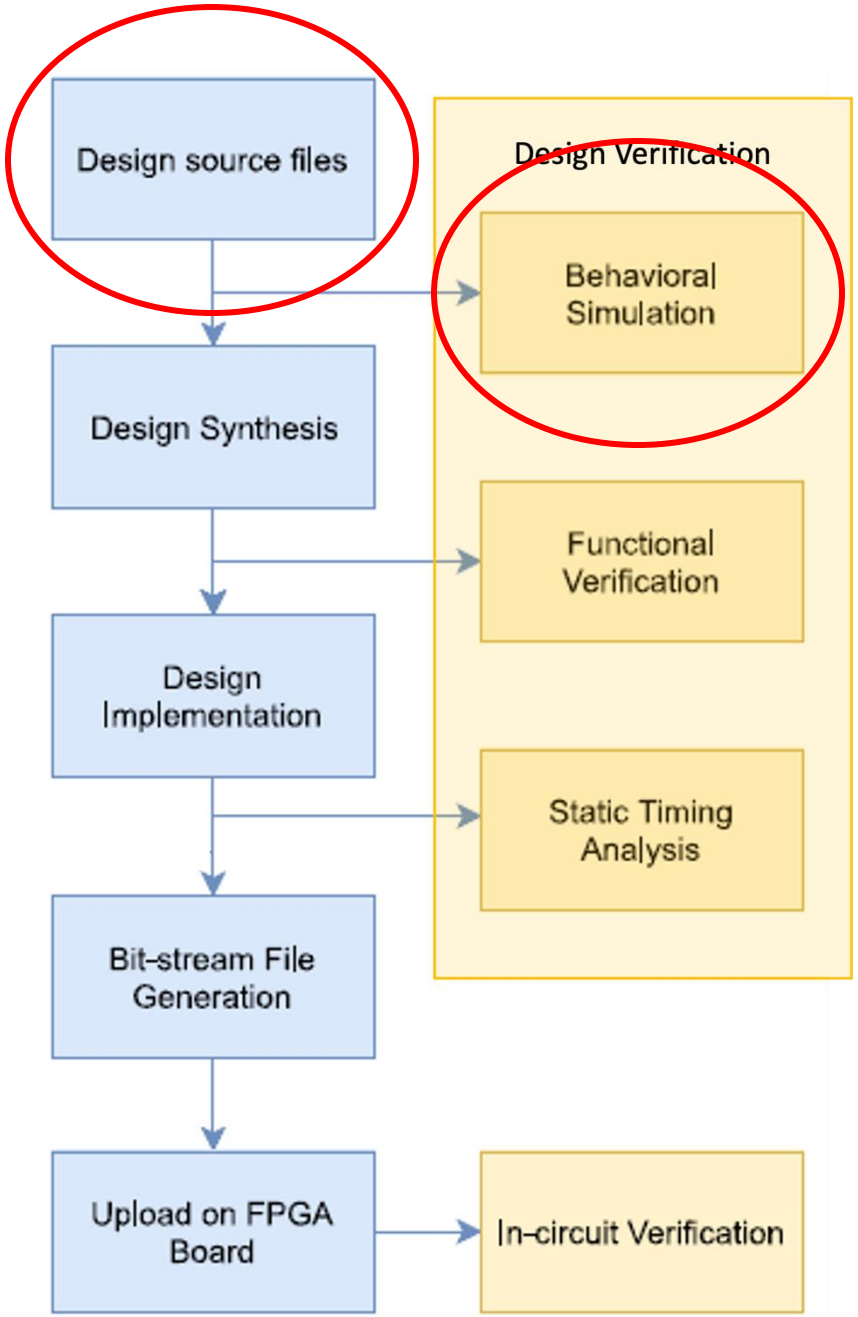
Lab 1

Prof. Laura Gonella

Laboratorio di Acquisizione e Controllo Dati
a.a. 2024-25

- In these two lab sessions we will learn the basics of VHDL design and behavioral simulation using examples of simple combinatorial and sequential circuits

- We will use a free online software, EDA Playground, that allows to write VHDL code to design and simulate digital blocks



Examples and exercises

- Example: OR gate
- Exercise 1: AND gate
- Exercise 2: 2to1 MUX

- Example: OR gate with “process”
- Example: D-FF
- Exercise 3: 2to1 MUX with “process” and “if” statement

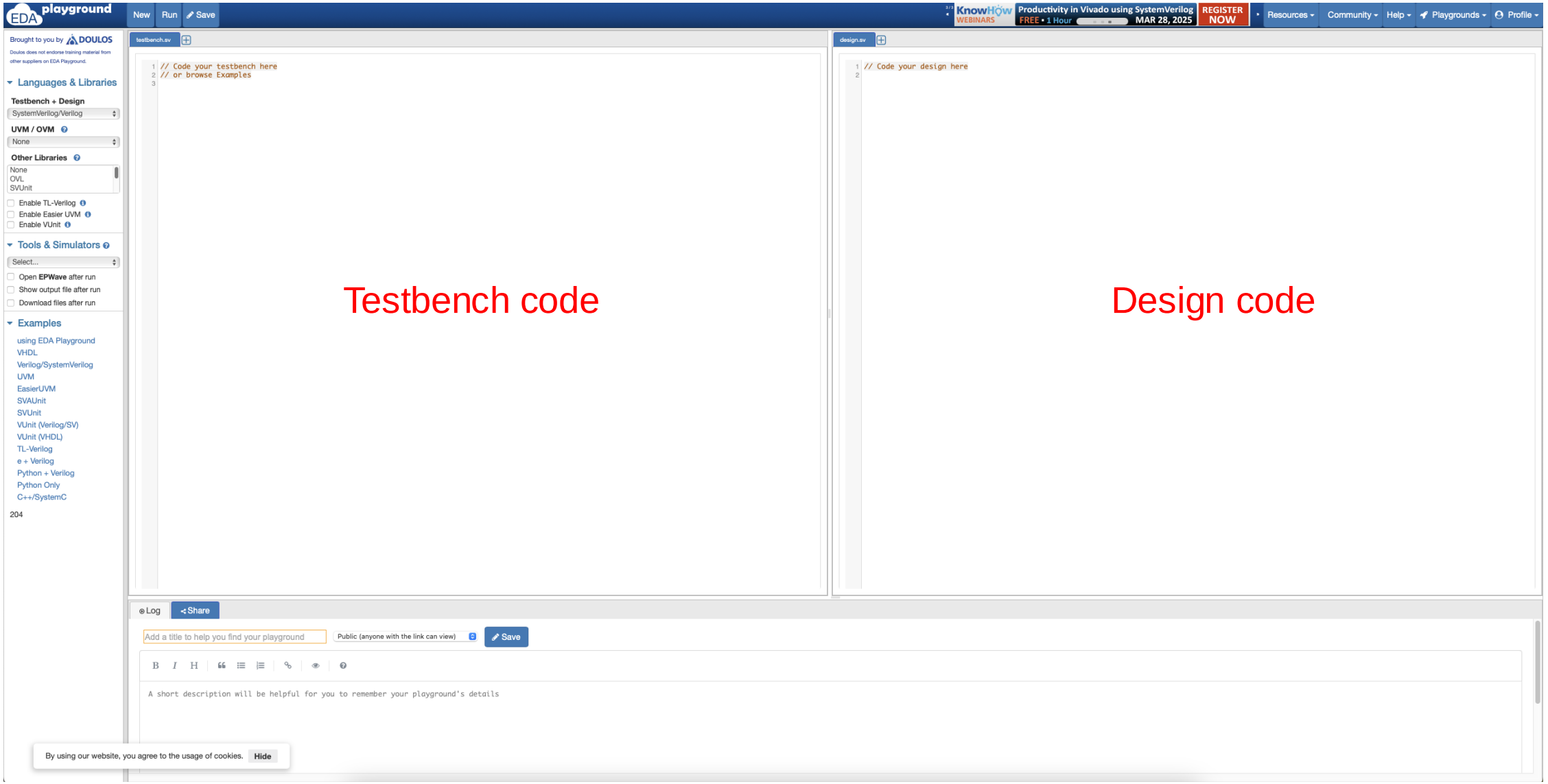
- Exercise 4: D-FF with synchronous reset
- Exercise 5: Shift register

EDA Playground

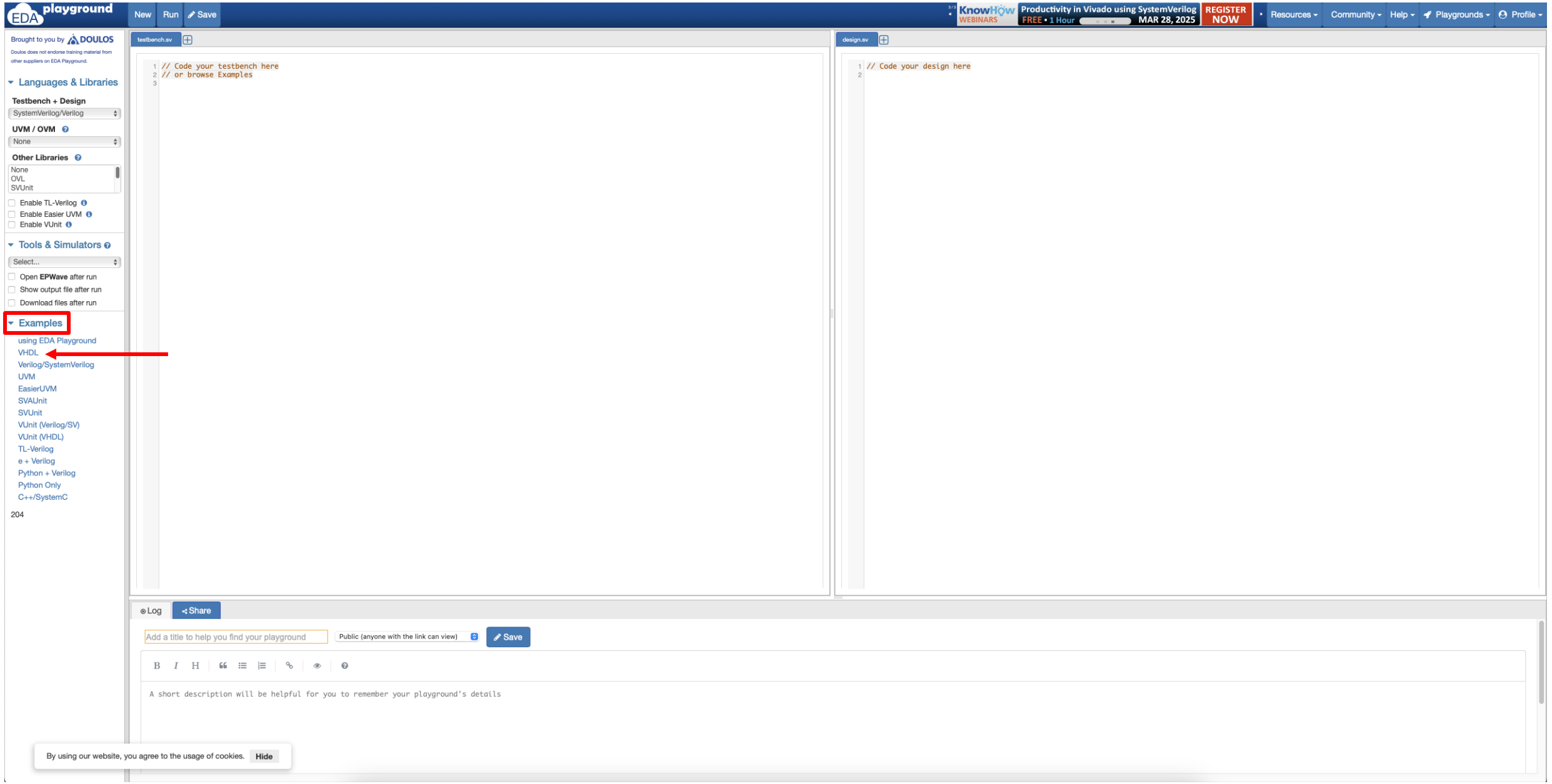
- https://www.edaplayground.com

- Create an account

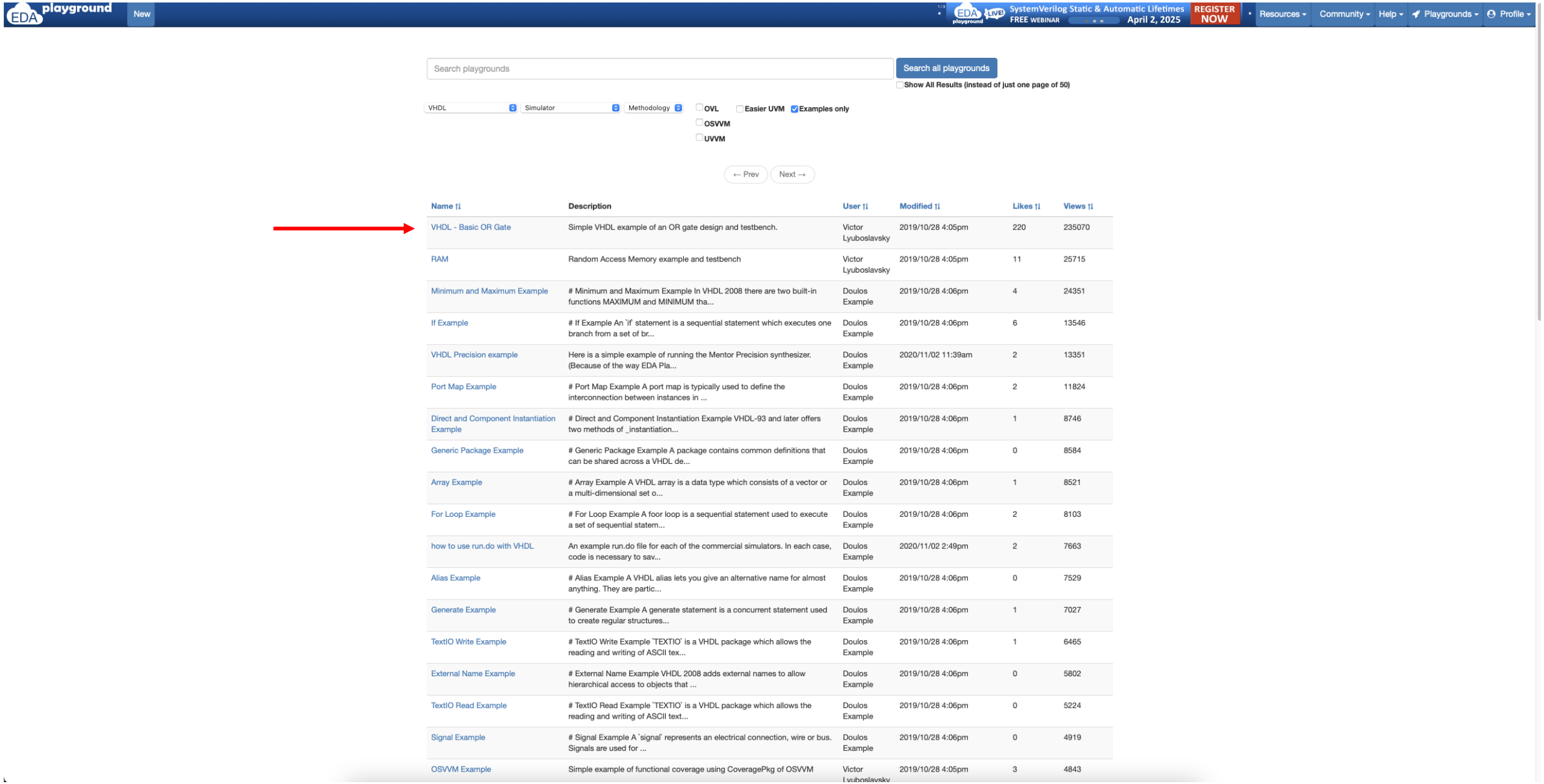
EDA Playground



EDA Playground: Examples

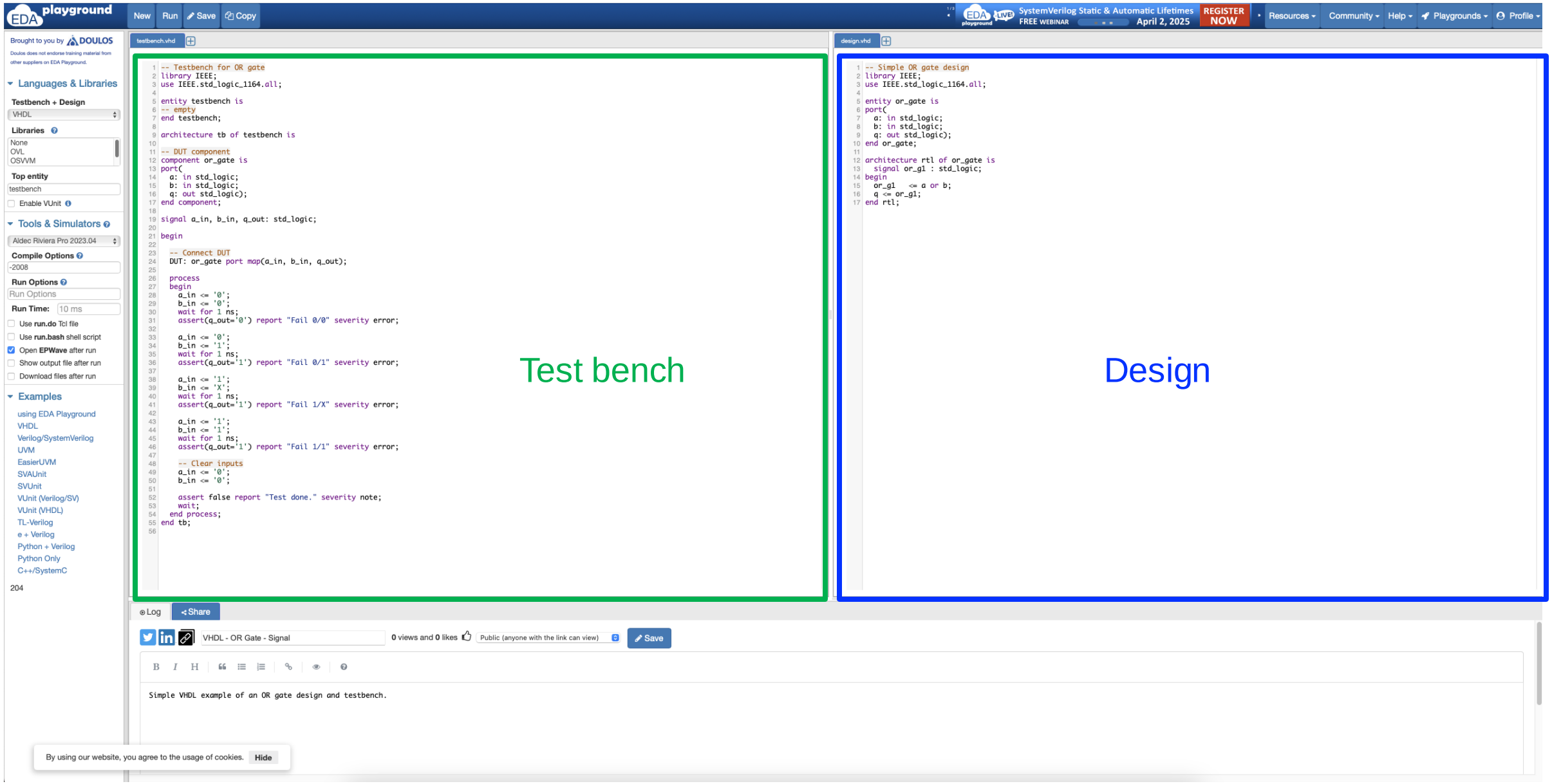


EDA Playground: Examples



Name	Description	User	Modified	Likes	Views
VHDL - Basic OR Gate	Simple VHDL example of an OR gate design and testbench.	Victor Lyuboslavsky	2019/10/28 4:05pm	220	235070
RAM	Random Access Memory example and testbench	Victor Lyuboslavsky	2019/10/28 4:05pm	11	25715
Minimum and Maximum Example	# Minimum and Maximum Example In VHDL 2008 there are two built-in functions MAXIMUM and MINIMUM tha...	Doulos Example	2019/10/28 4:06pm	4	24351
If Example	# If Example An `if` statement is a sequential statement which executes one branch from a set of br...	Doulos Example	2019/10/28 4:06pm	6	13546
VHDL Precision example	Here is a simple example of running the Mentor Precision synthesizer. (Because of the way EDA Pla...	Doulos Example	2020/11/02 11:39am	2	13351
Port Map Example	# Port Map Example A port map is typically used to define the interconnection between instances in ...	Doulos Example	2019/10/28 4:06pm	2	11824
Direct and Component Instantiation Example	# Direct and Component Instantiation Example VHDL-93 and later offers two methods of _instantiation...	Doulos Example	2019/10/28 4:06pm	1	8746
Generic Package Example	# Generic Package Example A package contains common definitions that can be shared across a VHDL de...	Doulos Example	2019/10/28 4:06pm	0	8584
Array Example	# Array Example A VHDL array is a data type which consists of a vector or a multi-dimensional set o...	Doulos Example	2019/10/28 4:06pm	1	8521
For Loop Example	# For Loop Example A foor loop is a sequential statement used to execute a set of sequential statem...	Doulos Example	2019/10/28 4:06pm	2	8103
how to use run.do with VHDL	An example run.do file for each of the commercial simulators. In each case, code is necessary to sav...	Doulos Example	2020/11/02 2:49pm	2	7663
Alias Example	# Alias Example A VHDL alias lets you give an alternative name for almost anything. They are partic...	Doulos Example	2019/10/28 4:06pm	0	7529
Generate Example	# Generate Example A generate statement is a concurrent statement used to create regular structures...	Doulos Example	2019/10/28 4:06pm	1	7027
TextIO Write Example	# TextIO Write Example `TEXTIO` is a VHDL package which allows the reading and writing of ASCII tex...	Doulos Example	2019/10/28 4:06pm	1	6465
External Name Example	# External Name Example VHDL 2008 adds external names to allow hierarchical access to objects that ...	Doulos Example	2019/10/28 4:06pm	0	5802
TextIO Read Example	# TextIO Read Example `TEXTIO` is a VHDL package which allows the reading and writing of ASCII text...	Doulos Example	2019/10/28 4:06pm	0	5224
Signal Example	# Signal Example A `signal` represents an electrical connection, wire or bus. Signals are used for ...	Doulos Example	2019/10/28 4:06pm	0	4919
OSVVM Example	Simple example of functional coverage using CoveragePkg of OSVVM	Victor Lyuboslavsky	2019/10/28 4:05pm	3	4843

Example of VHDL code: OR gate



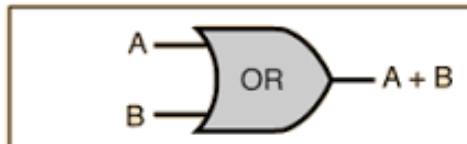
VHDL basics

- VHDL is a strongly typed language (like C)
  - All objects in VHDL must have a type (signals, variables, ports, etc...)
  - Objects can only have value of that type
  - Objects must be declared before using them
  - Operations are allowed only between same-type objects
  - Advantage: Less mistakes. Disadvantage: Longer codes
- VHDL is case insensitive
- Every VHDL statement ends with a semicolumn ;
- Not sensitive to white spaces between statements
- Empty lines ignored by compiler
- Comments start with two dashes --

Building blocks of VHDL

design.vhd

```
1  -- Simple OR gate design
2  library IEEE;
3  use IEEE.std_logic_1164.all;
4
5  entity or_gate is
6  port(
7    a: in std_logic;
8    b: in std_logic;
9    q: out std_logic);
10 end or_gate;
11
12 architecture rtl of or_gate is
13   signal or_g1 : std_logic;
14 begin
15   or_g1   <= a or b;
16   q <= or_g1;
17 end rtl;
```



A	B	Out
0	0	0
0	1	1
1	0	1
1	1	1

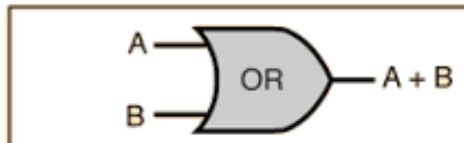
- Library and packages
- Entity
- Architecture

[Words in purple are VHDL keywords]

Entity, port

design.vhd

```
1  -- Simple OR gate design
2  library IEEE;
3  use IEEE.std_logic_1164.all;
4
5  entity or_gate is
6  port(
7    a: in std_logic;
8    b: in std_logic;
9    q: out std_logic);
10 end or_gate;
11
12 architecture rtl of or_gate is
13   signal or_g1 : std_logic;
14 begin
15   or_g1   <= a or b;
16   q <= or_g1;
17 end rtl;
```



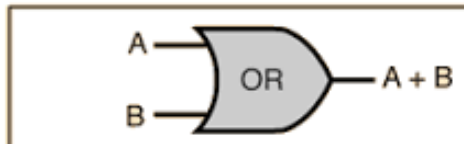
A	B	Out
0	0	0
0	1	1
1	0	1
1	1	1

- The **entity** declares the digital block
  - **or_gate** in this example
- The **port** within the entity contains statements to define the interface to the outside world
  - Common port modes: **in, out, inout**
  - In this example
    - **a, b** are the inputs to the digital block
    - **q** is the output of the digital block
    - **a, b, q** are of type **std_logic** (one of the most common data type in VHDL, from the IEEE library, more in a bit)
  - No semicolumn (;) after the last port definition

Architecture

design.vhd

```
1  -- Simple OR gate design
2  library IEEE;
3  use IEEE.std_logic_1164.all;
4
5  entity or_gate is
6  port(
7    a: in std_logic;
8    b: in std_logic;
9    q: out std_logic);
10 end or_gate;
11
12 architecture rtl of or_gate is
13   signal or_g1 : std_logic;
14 begin
15   or_g1   <= a or b;
16   q <= or_g1;
17 end rtl;
```



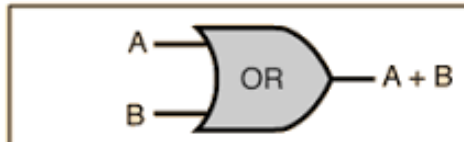
A	B	Out
0	0	0
0	1	1
1	0	1
1	1	1

- The **architecture** implements the functionality of the entity/digital block
- It is composed of two parts
  - Declarative part between the **is** and the **begin** keywords
  - Implementation part between the **begin** and the **end** keywords

Architecture

design.vhd

```
1  -- Simple OR gate design
2  library IEEE;
3  use IEEE.std_logic_1164.all;
4
5  entity or_gate is
6  port(
7    a: in std_logic;
8    b: in std_logic;
9    q: out std_logic);
10 end or_gate;
11
12 architecture rtl of or_gate is
13   signal or_g1 : std_logic;
14 begin
15   or_g1   <= a or b;
16   q <= or_g1;
17 end rtl;
```



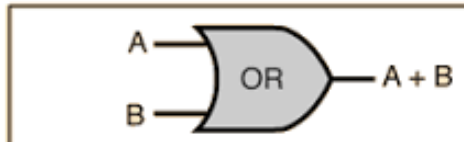
A	B	Out
0	0	0
0	1	1
1	0	1
1	1	1

- The functionality of the digital block is described in the implementation part of the architecture
- In this example, the functionality implemented by the architecture is the OR function
- **or** reserved keyword in VHDL

Signal declaration

design.vhd

```
1  -- Simple OR gate design
2  library IEEE;
3  use IEEE.std_logic_1164.all;
4
5  entity or_gate is
6  port(
7    a: in std_logic;
8    b: in std_logic;
9    q: out std_logic);
10 end or_gate;
11
12 architecture rtl of or_gate is
13   signal or_g1 : std_logic;
14 begin
15   or_g1   <= a or b;
16   q <= or_g1;
17 end rtl;
```



A	B	Out
0	0	0
0	1	1
1	0	1
1	1	1

- All signals that are used by the architecture must be defined in the declarative part
  - In this example, the architecture uses the signal **or_g1** of type **std_logic**
  - Optionally declare an initial value (if no initial value, derived from type definition)
  - Examples of signal declaration

```
signal a : std_logic;
signal b : integer;
```

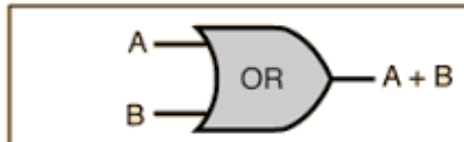
```
signal be_signal : std_logic_vector(11 downto 0);
```

```
signal A : std_logic := '0';
signal B : std_logic_vector(11 downto 0) := (others => '0');
```

Signal assignment

design.vhd

```vhdl
-- Simple OR gate design
library IEEE;
use IEEE.std_logic_1164.all;

entity or_gate is
port(
  a: in std_logic;
  b: in std_logic;
  q: out std_logic);
end or_gate;

architecture rtl of or_gate is
  signal or_g1 : std_logic;
begin
  or_g1   <= a or b;
  q <= or_g1;
end rtl;
```



A	B	Out
0	0	0
0	1	1
1	0	1
1	1	1

- Signal assignment is done with the **<=** operator, in the architecture implementation part
- Note: signal assignments are evaluated concurrently to each VHDL statement, when placed outside of a process block (more on process block in a bit)
  - Evaluated at the same time
  - The order in which they are written does not count
  - Equivalent implementations:

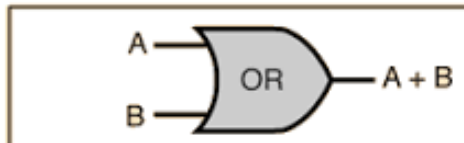
```vhdl
architecture Behavioral of MyModule is

begin
  S <= A;
  B <= C;
end Behavioral;
```

```vhdl
architecture Behavioral of MyModule is

begin
  B <= C;
  S <= A;
end Behavioral;
```

Library and packages

design.vhd

```
-- Simple OR gate design
library IEEE;
use IEEE.std_logic_1164.all;

entity or_gate is
port(
  a: in std_logic;
  b: in std_logic;
  q: out std_logic);
end or_gate;

architecture rtl of or_gate is
  signal or_g1 : std_logic;
begin
  or_g1   <= a or b;
  q <= or_g1;
end rtl;
```



A	B	Out
0	0	0
0	1	1
1	0	1
1	1	1

- **Library** and **packages**
  - Import definitons (data types, functions, keywords, etc) from other files
  - **Use** statement to add packages from a library
- In this example we use the **IEEE** library, one of the most common library used in VHDL designs
  - **IEEE.std_logic_1164** is one of two main packages to import when using this library
  - The other is **IEEE.numeric_std**

Data types

- Data types
  - Built-in; Third parties; User-defined

- 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)

- Built-in data-types are not enough to describe an actual circuit
  - Types can be extended using external libraries (Third parties or User-defined)

IEEE packages

- VHDL logic types
  - **std_logic** for single bit signals
  - **std_logic_vector** for multiple-bit signals (buses)
  - **signed**: vector representation of signed binary numbers
  - **unsigned**: vector representation of unsigned binary numbers

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

Testbench

- Remember: VHDL code in the testbench will never be synthesized!
  - Non synthesizable code can be used, typically makes simulation easier and better
- The test bench has **library**, **entity** and **architecture**
- Testbench entity typically empty

testbench.vhd

```
-- Testbench for OR gate
library IEEE;
use IEEE.std_logic_1164.all;

entity testbench is
-- empty
end testbench;

architecture tb of testbench is

-- DUT component
component or_gate is
port(
  a: in std_logic;
  b: in std_logic;
  q: out std_logic);
end component;

signal a_in, b_in, q_out: std_logic;

begin

  -- Connect DUT
  DUT: or_gate port map(a_in, b_in, q_out);

  process
  begin
    a_in <= '0';
    b_in <= '0';
    wait for 1 ns;
    assert(q_out='0') report "Fail 0/0" severity error;

    a_in <= '0';
    b_in <= '1';
    wait for 1 ns;
    assert(q_out='1') report "Fail 0/1" severity error;

    a_in <= '1';
    b_in <= 'X';
    wait for 1 ns;
    assert(q_out='1') report "Fail 1/X" severity error;

    a_in <= '1';
    b_in <= '1';
    wait for 1 ns;
    assert(q_out='1') report "Fail 1/1" severity error;

    -- Clear inputs
    a_in <= '0';
    b_in <= '0';

    assert false report "Test done." severity note;
    wait;
  end process;
end tb;
```

Component declaration & instantiation

- Declare the component to be simulated in the declarative region of your architecture
- Instantiate the component to be simulated in the implementation region architecture
- DUT – Device Under Test
  - Sometimes UUT – Unit Under Test

testbench.vhd

```
1  -- Testbench for OR gate
2  library IEEE;
3  use IEEE.std_logic_1164.all;
4
5  entity testbench is
6  -- empty
7  end testbench;
8
9  architecture tb of testbench is
10
11 -- DUT component
12 component or_gate is
13 port(
14   a: in std_logic;
15   b: in std_logic;
16   q: out std_logic);
17 end component;
18
19 signal a_in, b_in, q_out: std_logic;
20
21 begin
22
23   -- Connect DUT
24   DUT: or_gate port map(a_in, b_in, q_out);
25
26   process
27   begin
28     a_in <= '0';
29     b_in <= '0';
30     wait for 1 ns;
31     assert(q_out='0') report "Fail 0/0" severity error;
32
33     a_in <= '0';
34     b_in <= '1';
35     wait for 1 ns;
36     assert(q_out='1') report "Fail 0/1" severity error;
37
38     a_in <= '1';
39     b_in <= 'X';
40     wait for 1 ns;
41     assert(q_out='1') report "Fail 1/X" severity error;
42
43     a_in <= '1';
44     b_in <= '1';
45     wait for 1 ns;
46     assert(q_out='1') report "Fail 1/1" severity error;
47
48     -- Clear inputs
49     a_in <= '0';
50     b_in <= '0';
51
52     assert false report "Test done." severity note;
53     wait;
54   end process;
55 end tb;
56
```

Simulating input signals

- The stimuli to be applied to the inputs of the DUT are defined in the architecture using a **process** block (more about process block in a bit)

testbench.vhd

```
-- Testbench for OR gate
library IEEE;
use IEEE.std_logic_1164.all;

entity testbench is
-- empty
end testbench;

architecture tb of testbench is

-- DUT component
component or_gate is
port(
  a: in std_logic;
  b: in std_logic;
  q: out std_logic);
end component;

signal a_in, b_in, q_out: std_logic;

begin

  -- Connect DUT
  DUT: or_gate port map(a_in, b_in, q_out);

  process
  begin
    a_in <= '0';
    b_in <= '0';
    wait for 1 ns;
    assert(q_out='0') report "Fail 0/0" severity error;

    a_in <= '0';
    b_in <= '1';
    wait for 1 ns;
    assert(q_out='1') report "Fail 0/1" severity error;

    a_in <= '1';
    b_in <= 'X';
    wait for 1 ns;
    assert(q_out='1') report "Fail 1/X" severity error;

    a_in <= '1';
    b_in <= '1';
    wait for 1 ns;
    assert(q_out='1') report "Fail 1/1" severity error;

    -- Clear inputs
    a_in <= '0';
    b_in <= '0';

    assert false report "Test done." severity note;
    wait;
  end process;
end tb;
```

wait statements

- **wait** statements are used in test benches as delays to sequence inputs
  - Not synthesizable code
  - **wait for <time>;** (this example)
  - **wait on <signal>;**
    - Waiting for an event (change of state in a signal)
  - **wait until <boolean expression>;**
    - Wait for a specific signal value

testbench.vhd

```
1  -- Testbench for OR gate
2  library IEEE;
3  use IEEE.std_logic_1164.all;
4
5  entity testbench is
6  -- empty
7  end testbench;
8
9  architecture tb of testbench is
10
11 -- DUT component
12 component or_gate is
13 port(
14   a: in std_logic;
15   b: in std_logic;
16   q: out std_logic);
17 end component;
18
19 signal a_in, b_in, q_out: std_logic;
20
21 begin
22
23   -- Connect DUT
24   DUT: or_gate port map(a_in, b_in, q_out);
25
26   process
27   begin
28     a_in <= '0';
29     b_in <= '0';
30     wait for 1 ns;
31     assert(q_out='0') report "Fail 0/0" severity error;
32
33     a_in <= '0';
34     b_in <= '1';
35     wait for 1 ns;
36     assert(q_out='1') report "Fail 0/1" severity error;
37
38     a_in <= '1';
39     b_in <= 'X';
40     wait for 1 ns;
41     assert(q_out='1') report "Fail 1/X" severity error;
42
43     a_in <= '1';
44     b_in <= '1';
45     wait for 1 ns;
46     assert(q_out='1') report "Fail 1/1" severity error;
47
48     -- Clear inputs
49     a_in <= '0';
50     b_in <= '0';
51
52     assert false report "Test done." severity note;
53     wait;
54   end process;
55 end tb;
56
```

Assert, report, severity

- **assert, report, severity**
  - Use the assert function to check signal values against some expectation
  - Assert returns always a boolean value
  - Default **severity** is **error**

testbench.vhd

```vhdl
-- Testbench for OR gate
library IEEE;
use IEEE.std_logic_1164.all;

entity testbench is
-- empty
end testbench;

architecture tb of testbench is

-- DUT component
component or_gate is
port(
  a: in std_logic;
  b: in std_logic;
  q: out std_logic);
end component;

signal a_in, b_in, q_out: std_logic;

begin

  -- Connect DUT
  DUT: or_gate port map(a_in, b_in, q_out);

  process
  begin
    a_in <= '0';
    b_in <= '0';
    wait for 1 ns;
    assert(q_out='0') report "Fail 0/0" severity error;

    a_in <= '0';
    b_in <= '1';
    wait for 1 ns;
    assert(q_out='1') report "Fail 0/1" severity error;

    a_in <= '1';
    b_in <= 'X';
    wait for 1 ns;
    assert(q_out='1') report "Fail 1/X" severity error;

    a_in <= '1';
    b_in <= '1';
    wait for 1 ns;
    assert(q_out='1') report "Fail 1/1" severity error;

    -- Clear inputs
    a_in <= '0';
    b_in <= '0';

    assert false report "Test done." severity note;
    wait;
  end process;
end tb;
```

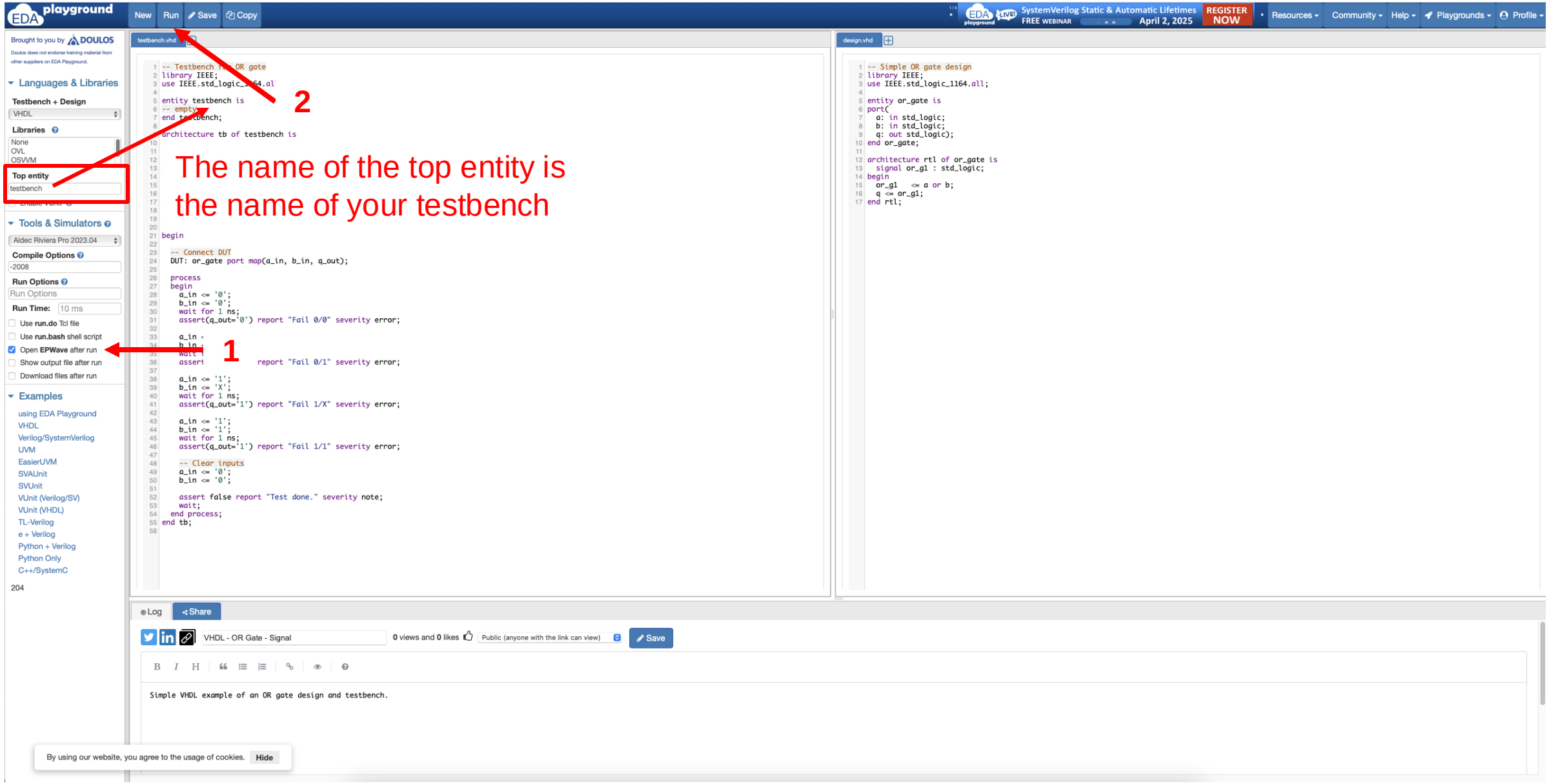
Terminating the simulation

- **wait;**
  - Stopping all stimuli at the end of the process

testbench.vhd

```
-- Testbench for OR gate
library IEEE;
use IEEE.std_logic_1164.all;

entity testbench is
-- empty
end testbench;

architecture tb of testbench is

-- DUT component
component or_gate is
port(
  a: in std_logic;
  b: in std_logic;
  q: out std_logic);
end component;

signal a_in, b_in, q_out: std_logic;

begin

  -- Connect DUT
  DUT: or_gate port map(a_in, b_in, q_out);

  process
  begin
    a_in <= '0';
    b_in <= '0';
    wait for 1 ns;
    assert(q_out='0') report "Fail 0/0" severity error;

    a_in <= '0';
    b_in <= '1';
    wait for 1 ns;
    assert(q_out='1') report "Fail 0/1" severity error;

    a_in <= '1';
    b_in <= 'X';
    wait for 1 ns;
    assert(q_out='1') report "Fail 1/X" severity error;

    a_in <= '1';
    b_in <= '1';
    wait for 1 ns;
    assert(q_out='1') report "Fail 1/1" severity error;

    -- Clear inputs
    a_in <= '0';
    b_in <= '0';

    assert false report "Test done." severity note;
    wait;
  end process;
end tb;
```

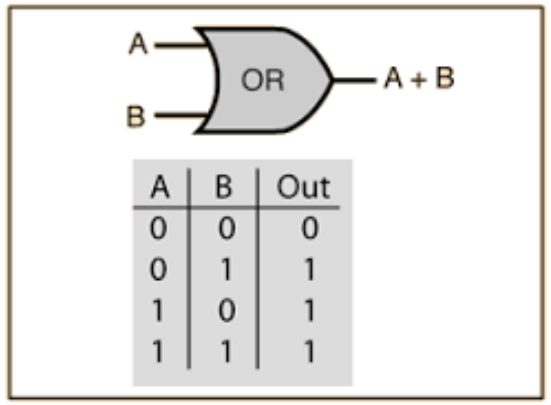
EDA Playground: Run a simulation



EDA Playground: Simulation waveforms

design.vhd

```
1  -- Simple OR gate design
2  library IEEE;
3  use IEEE.std_logic_1164.all;
4
5  entity or_gate is
6  port(
7    a: in std_logic;
8    b: in std_logic;
9    q: out std_logic);
10 end or_gate;
11
12 architecture rtl of or_gate is
13   signal or_g1 : std_logic;
14 begin
15   or_g1   <= a or b;
16   q <= or_g1;
17 end rtl;
```



A	B	Out
0	0	0
0	1	1
1	0	1
1	1	1

testbench.vhd

```
1  -- Testbench for OR gate
2  library IEEE;
3  use IEEE.std_logic_1164.all;
4
5  entity testbench is
6  -- empty
7  end testbench;
8
9  architecture tb of testbench is
10
11 -- DUT component
12 component or_gate is
13 port(
14   a: in std_logic;
15   b: in std_logic;
16   q: out std_logic);
17 end component;
18
19 signal a_in, b_in, q_out: std_logic;
20
21 begin
22
23   -- Connect DUT
24   DUT: or_gate port map(a_in, b_in, q_out);
25
26   process
27   begin
28     a_in <= '0';
29     b_in <= '0';
30     wait for 1 ns;
31     assert(q_out='0') report "Fail 0/0" severity error;
32
33     a_in <= '0';
34     b_in <= '1';
35     wait for 1 ns;
36     assert(q_out='1') report "Fail 0/1" severity error;
37
38     a_in <= '1';
39     b_in <= 'X';
40     wait for 1 ns;
41     assert(q_out='1') report "Fail 1/X" severity error;
42
43     a_in <= '1';
44     b_in <= '1';
45     wait for 1 ns;
46     assert(q_out='1') report "Fail 1/1" severity error;
47
48     -- Clear inputs
49     a_in <= '0';
50     b_in <= '0';
51
52     assert false report "Test done." severity note;
53     wait;
54   end process;
55 end tb;
56
```

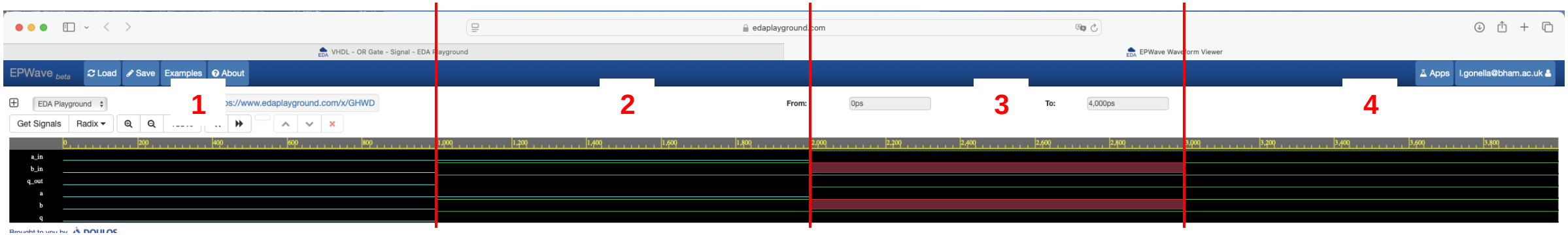
1

2

3

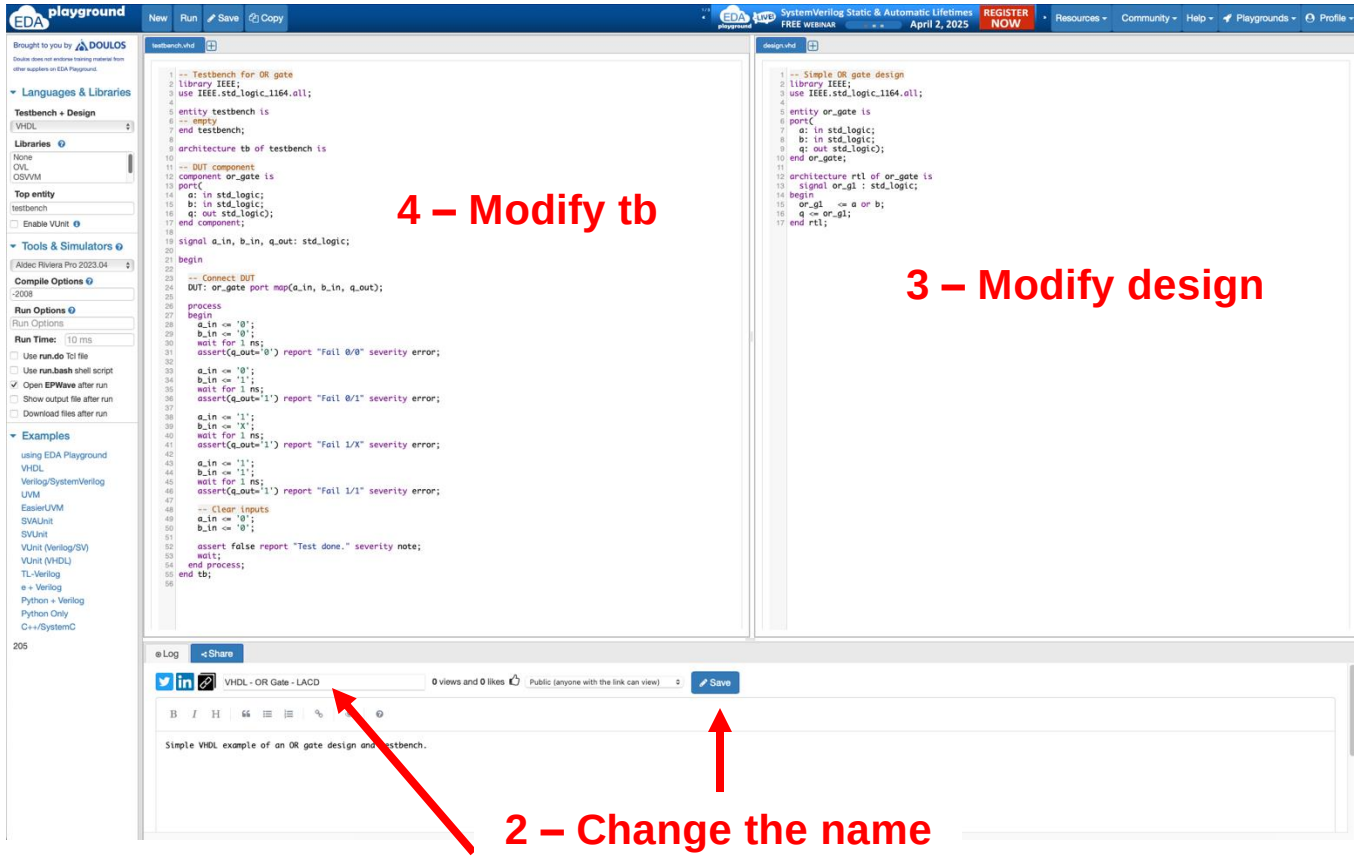
4

Simulation waveforms show changes to signal values as function of simulation time
a_in, b_in, q_out: internal signals - a, b, q: interface signals



Exercise 1: Design and simulate a 2-input AND gate

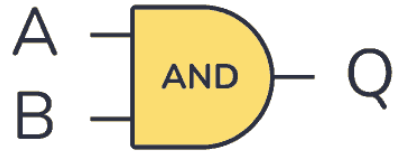
1 – Create a copy of the OR project



4 – Modify tb

3 – Modify design

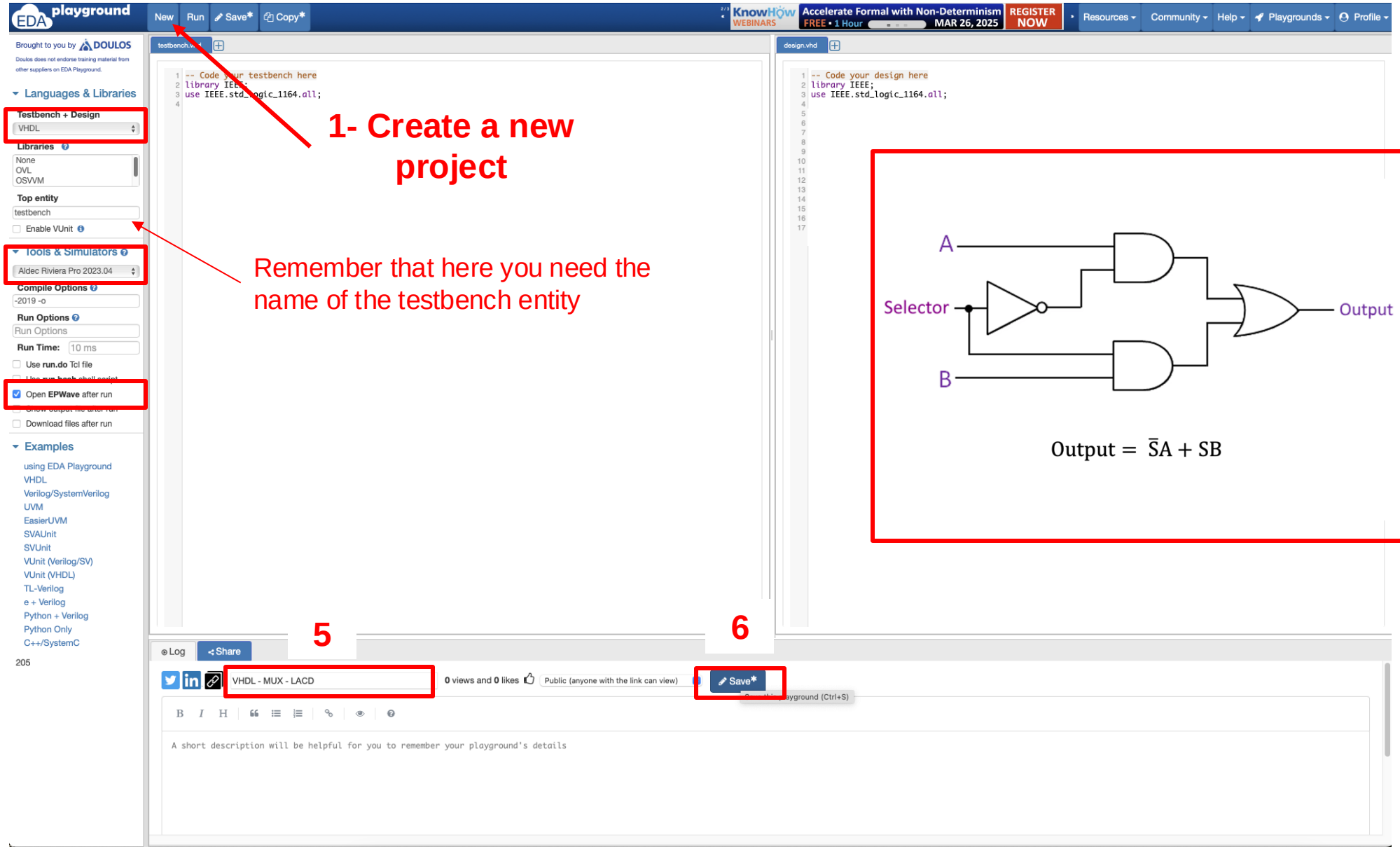
5 – Save and run the simulation



A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

2 – Change the name to AND gate and save

Exercise 2: Design and simulate a 2to1 MUX



A	B	S	Q
0	0	0	0
0	1	0	0
1	0	0	1
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	0
1	1	1	1

Conditional statement: when/else

- To describe the MUX functionality in the architecture, you need to use "when/else"
  - Conditional signal assignment (implicit if)
  - Examples of the sintax:

```
A <= '1' when B='1' else '0';
```

```
output <= input1 when mux_sel = "00" else
          input2 when mux_sel = "01" else
          (others => '0');
```

- Using these examples, implement the syntax needed for your MUX

End of part 1

- VHDL basics and building blocks
- Entity, port
- Architecture
- Signal declaration and assignment
- Library and packages
- Data types
- IEEE library
- Conditional statement: when/else

- Component declaration and instantiation in test bench
- Simulating the input signals
- wait statements
- Asserting, report, severity
- Terminating the simulation
- Run a simulation in EDA playground

- Example: OR gate
- Exercise 1: AND gate
- Exercise 2: 2to1 MUX