

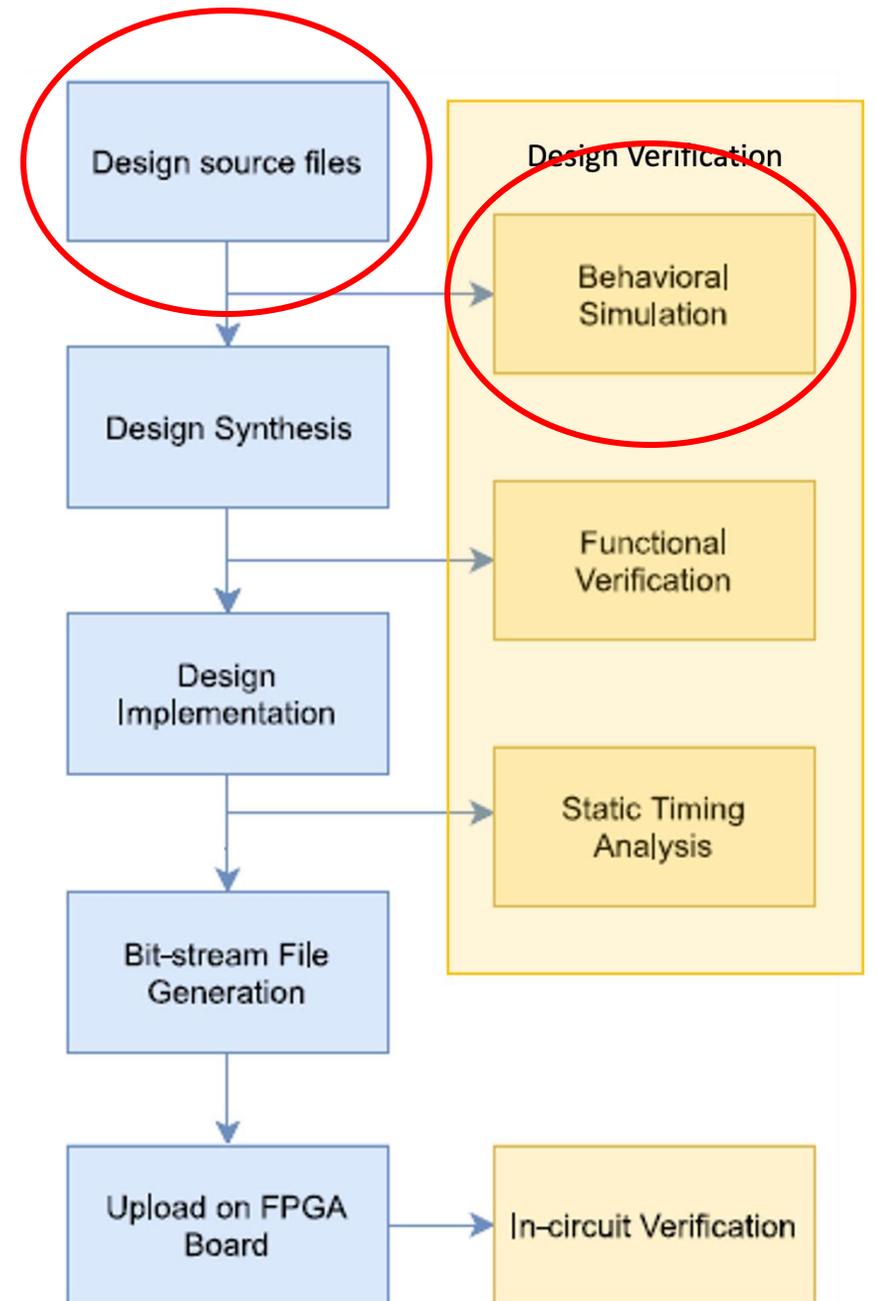
Introduzione alle FPGA

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

The screenshot shows the EDA Playground interface. At the top, there is a navigation bar with 'New', 'Run', and 'Save' buttons. A banner for 'KnowHow WEBINARS' is visible, advertising a 'FREE • 1 Hour' webinar on 'Productivity in Vivado using SystemVerilog' on 'MAR 28, 2025', with a 'REGISTER NOW' button. The main area is split into two code editors. The left editor, titled 'testbench.sv', contains the text: '1 // Code your testbench here', '2 // or browse Examples', and '3'. The right editor, titled 'design.sv', contains the text: '1 // Code your design here' and '2'. A large red text overlay 'Testbench code' is centered over the left editor, and 'Design code' is centered over the right editor. On the left side, there is a sidebar with 'Languages & Libraries' and 'Tools & Simulators' sections. The 'Languages & Libraries' section includes 'Testbench + Design' (SystemVerilog/Verilog), 'UVM / OVM' (None), and 'Other Libraries' (None, OVL, SVUnit) with checkboxes for 'Enable TL-Verilog', 'Enable Easier UVM', and 'Enable VUnit'. The 'Tools & Simulators' section has a 'Select...' dropdown and checkboxes for 'Open EPWave after run', 'Show output file after run', and 'Download files after run'. Below this is an 'Examples' list with categories like 'using EDA Playground', 'VHDL', 'Verilog/SystemVerilog', 'UVM', 'EasierUVM', 'SVAUnit', 'SVUnit', 'VUnit (Verilog/SV)', 'VUnit (VHDL)', 'TL-Verilog', '+ Verilog', 'Python + Verilog', 'Python Only', and 'C++/SystemC'. At the bottom, there is a 'Log' and 'Share' button, a text input field for a title, a 'Public (anyone with the link can view)' dropdown, and a 'Save' button. A rich text editor with formatting options (B, I, H, quote, list, link, image) is also present, with a placeholder text: 'A short description will be helpful for you to remember your playground's details'. A cookie notice at the bottom left reads: 'By using our website, you agree to the usage of cookies. Hide'.

EDA Playground: Examples

The screenshot displays the EDA Playground web interface. At the top, there is a navigation bar with 'New', 'Run', and 'Save' buttons. On the right side of the top bar, there are promotional banners for 'KnowHow WEBINARS' and 'Productivity in Vivado using SystemVerilog' with a 'REGISTER NOW' button. The main interface is split into two panes: 'testbench.sv' on the left and 'design.sv' on the right. The left pane contains a code editor with three lines of comments: '// Code your testbench here', '// or browse Examples', and a blank line. The right pane contains a code editor with two lines of comments: '// Code your design here' and a blank line. On the left side, there is a sidebar menu with several sections: 'Languages & Libraries', 'Tools & Simulators', and 'Examples'. The 'Examples' section is highlighted with a red box, and a red arrow points to the 'VHDL' option within it. Other options in the 'Examples' list include 'using EDA Playground', 'Verilog/SystemVerilog', 'UVM', 'EasierUVM', 'SVAUnit', 'SVUnit', 'VUnit (Verilog/SV)', 'VUnit (VHDL)', 'TL-Verilog', 'e + Verilog', 'Python + Verilog', 'Python Only', and 'C++/SystemC'. At the bottom of the interface, there is a sharing section with a 'Log' button, a 'Share' button, a text input field for a title, a dropdown for visibility (set to 'Public'), and a 'Save' button. Below this is a rich text editor with a toolbar and a placeholder text: 'A short description will be helpful for you to remember your playground's details'. A cookie notice is visible at the very bottom of the page.

EDA Playground: Examples

Search playgrounds Show All Results (instead of just one page of 50)

- VHDL Simulator Methodology OVL Easier UVM Examples only OSVVM UVVM

← Prev Next →

| 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 for 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 |

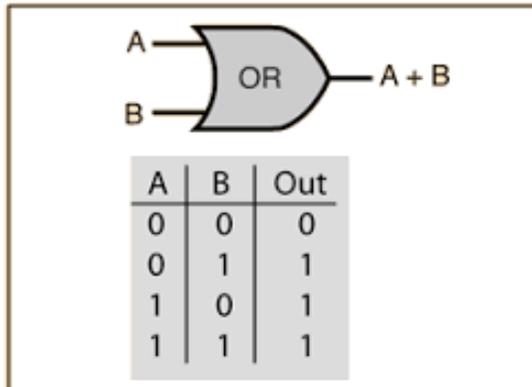


VHDL basics

- 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 **semicolon ;**
- **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;
```

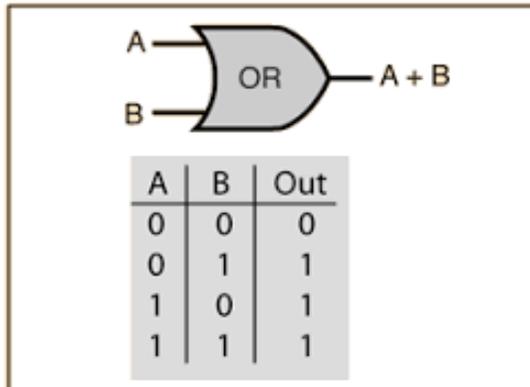


- Library and packages
- Entity
- Architecture

[Words in purple are VHDL keywords]

Entity, port

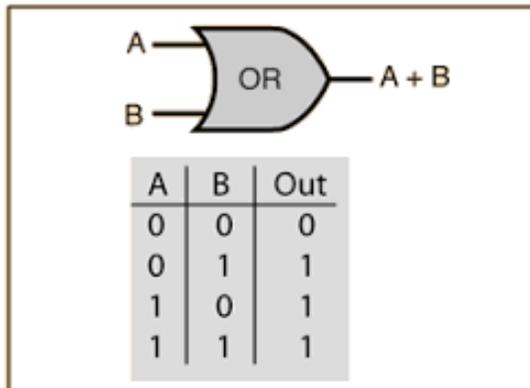
```
design.vhd +
1  -- Simple OR gate design
2  library IEEE;
3  use IEEE.std_logic_1164.all;
4
5  entity or_gate is
6  port(
7    a: in std_logic;
8    b: in std_logic;
9    q: out std_logic);
10 end or_gate;
11
12 architecture rtl of or_gate is
13   signal or_g1 : std_logic;
14 begin
15   or_g1  <=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



- 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 semicolon (;) after the last port definition

Architecture

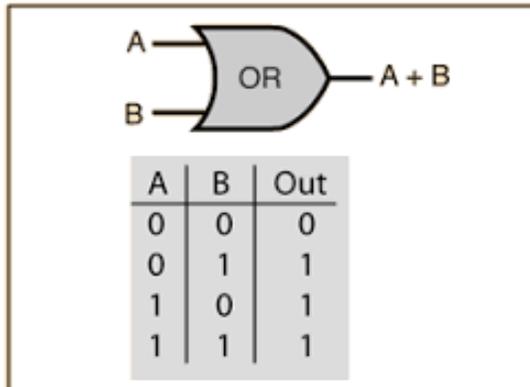
```
design.vhd +
1  -- Simple OR gate design
2  library IEEE;
3  use IEEE.std_logic_1164.all;
4
5  entity or_gate is
6  port(
7    a: in std_logic;
8    b: in std_logic;
9    q: out std_logic);
10 end or_gate;
11
12 architecture rtl of or_gate is
13   signal or_g1 : std_logic;
14   begin
15     or_g1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



- 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

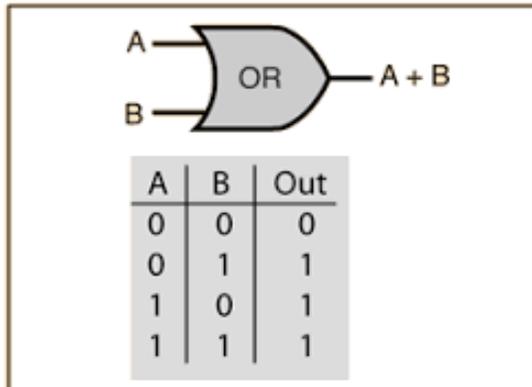
```
design.vhd +
1  -- Simple OR gate design
2  library IEEE;
3  use IEEE.std_logic_1164.all;
4
5  entity or_gate is
6  port(
7    a: in std_logic;
8    b: in std_logic;
9    q: out std_logic);
10 end or_gate;
11
12 architecture rtl of or_gate is
13   signal or_g1 : std_logic;
14 begin
15   or_g1  <=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



- 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;
```



- 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;
```

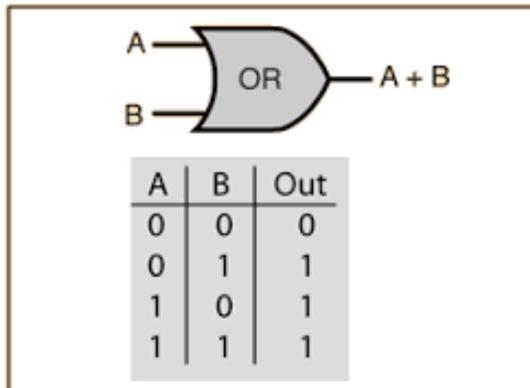
```
signal be_signal : std_logic_vector(11 downto 0);
```

```
signal A : std_logic := '0';
```

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

Signal assignment

```
design.vhd +
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;
```



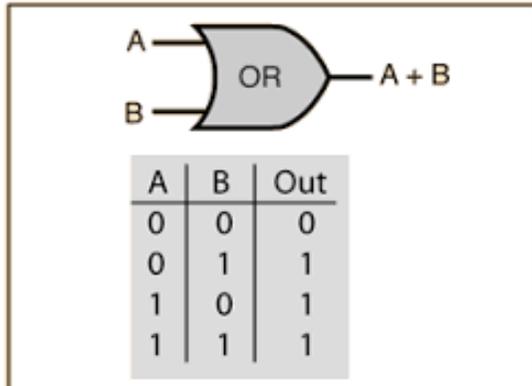
- **Signal assignment** is done with the `<=` operator, in the architecture implementation part
- Note: **signal assignments are evaluated concurrently to each VHDL statement**, when placed outside of a process block (more on process block in a bit)
 - Evaluated at the same time
 - The order in which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

```
architecture Behavioral of MyModule is
begin
  S <= A;
  B <= C;
end Behavioral;
```

```
architecture Behavioral of MyModule is
begin
  B <= C;
  S <= A;
end Behavioral;
```

Library and packages

```
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;
```



- **Library and packages**

- Import definitions (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'**: uninitialised
 - **'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

```
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
```

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



```
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
```

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



```
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
```

Terminating the simulation

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

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
```

EDA Playground: Run a simulation

The screenshot shows the EDA Playground interface. On the left, there is a sidebar with sections for Languages & Libraries, Tools & Simulators, and Examples. The 'Top entity' dropdown is highlighted with a red box and an arrow labeled '1'. The main editor shows two files: 'testbench.vhd' and 'design.vhd'. The 'testbench.vhd' code is highlighted with a red box and an arrow labeled '2'. The code in 'testbench.vhd' is as follows:

```
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
12
13
14
15
16
17
18
19
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 <= '1';
34   b_in <= '0';
35   wait
36   assert
37     report "Fail 0/1" severity error;
38
39   a_in <= '1';
40   b_in <= '1';
41   wait for 1 ns;
42   assert(q_out='1') report "Fail 1/X" severity error;
43
44   a_in <= '1';
45   b_in <= '1';
46   wait for 1 ns;
47   assert(q_out='1') report "Fail 1/1" severity error;
48
49 -- Clear inputs
50 a_in <= '0';
51 b_in <= '0';
52
53 assert false report "Test done." severity note;
54 wait;
55 end process;
56 end tb;
```

The 'design.vhd' code is as follows:

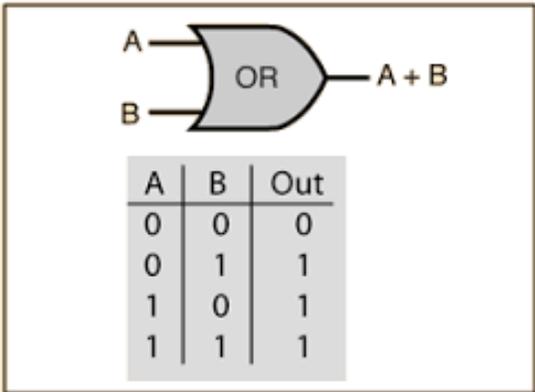
```
1 -- Simple OR gate design
2 library IEEE;
3 use IEEE.std_logic_1164.all;
4
5 entity or_gate is
6 port(
7   a: in std_logic;
8   b: in std_logic;
9   q: out std_logic);
10 end or_gate;
11
12 architecture rtl of or_gate is
13   signal or_g1: std_logic;
14 begin
15   or_g1 <= a or b;
16   q <= or_g1;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

At the bottom of the page, there is a social sharing section with a title 'VHDL - OR Gate - Signal', a description 'Simple VHDL example of an OR gate design and testbench.', and a cookie notice at the very bottom: 'By using our website, you agree to the usage of cookies. Hide'.

EDA Playground: Simulation waveforms

```

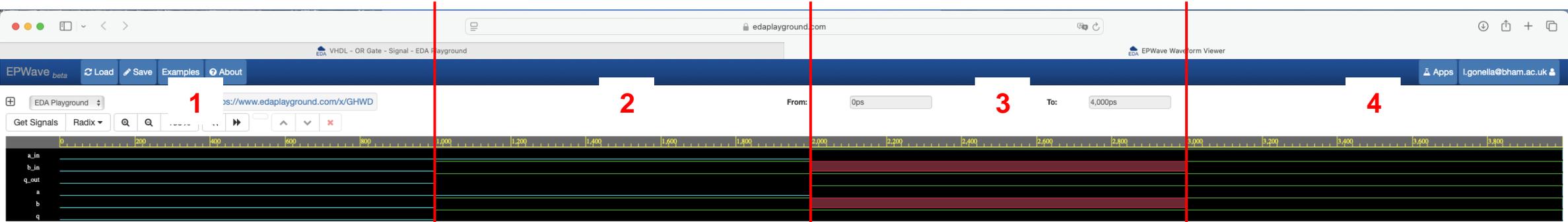
1  -- Simple OR gate design
2  library IEEE;
3  use IEEE.std_logic_1164.all;
4
5  entity or_gate is
6  port(
7    a: in std_logic;
8    b: in std_logic;
9    q: out std_logic);
10 end or_gate;
11
12 architecture rtl of or_gate is
13   signal or_g1 : std_logic;
14 begin
15   or_g1 <= a or b;
16   q <=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



```

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 <= '0';
40   wait for 1 ns;
41   assert(q_out='1') report "Fail 1/0" 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;
    
```

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

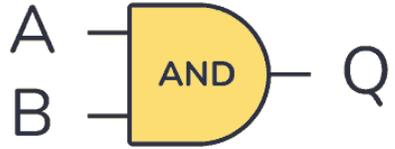
1 – Create a copy of the OR project



4 – Modify tb

3 – Modify design

2 – Change the name to AND gate and save



| A | B | Q |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

5 – Save and run the simulation

Exercise 2: Design and simulate a 2to1 MUX

2 Languages & Libraries

testbench + Design

VHDL

Libraries

None
OVL
OSVVM

Top entity

testbench

Enable VUnit

3 Tools & Simulators

Aldec Riviera Pro 2023.04

Compile Options

-2019 -o

Run Options

Run Time: 10 ms

Use run.do Tcl file

Use my bench shell script

4 Open EPWave after run

Download files after run

Examples

using EDA Playground

VHDL

Verilog/SystemVerilog

UVM

EasierUVM

SVAUnit

SVUnit

VUnit (Verilog/SV)

VUnit (VHDL)

TL-Verilog

e + Verilog

Python + Verilog

Python Only

C++/SystemC

1- Create a new project

```

1 -- Code your testbench here
2 library IEEE;
3 use IEEE.std_logic_1164.all;
4

```

Remember that here you need the name of the testbench entity

```

1 -- Code your design here
2 library IEEE;
3 use IEEE.std_logic_1164.all;
4
5
6
7
8
9
10
11
12
13
14
15
16
17

```

Output = $\bar{S}A + SB$

| 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 |

5 **6**

VHDL - MUX - LACD

0 views and 0 likes

Public (anyone with the link can view)

Save*

A short description will be helpful for you to remember your playground's details

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 syntax:

```
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