VHDL

Very High Speed Integrated Circuit
Hardware Description Language

Introduction to VHDL

- VHDL is a language that is used to describe the behavior of digital circuit designs
- VHDL provides everything that is necessary in order to describe any digital circuit
- VHDL design can be simulated and translated into a form suitable for hardware implementation
- Hierarchical use of VHDL designs permits the rapid creation of complex digital circuit designs

Terminology (1)

- Simulation is the prediction of the behavior of a design
  - VHDL provides many features suitable for the simulation of digital circuit designs
  - Functional simulation approximates the behavior of a hardware design by assuming that all outputs change at the same tie
  - Timing simulation predicts the exact behavior of a hardware design
- Synthesis is the translation of a design into a netlist file that describes the structure of a hardware design
  - **VHDL was not designed for the purpose of synthesis**

Terminology (2)

- Field-Programmable Gate Arrays (FPGAs) are programmable logic devices that permit the rapid prototyping of a digital circuit design
  - Configuring a device allows the FPGA to implement virtually any digital circuit design
  - VHDL designs may be created for the purpose fo generation a bitstream file to configure of a device
- Application-Specific Integrated Circuits (ASICs) are custom integrated circuits designed to implement a specific application
  - VHDL designs may be created for the purpose of generating the detailed layout files necessary to fabricate an ASIC

VHDL is a hardware-design language

1. When you are working with VHDL, you are ***not programming***, you are "***designing hardware***".
2. Your VHDL code should reflect this fact. It means that unless you are inside certain constructs, your code lines will be ***executed almost all at once***.
3. If your VHDL code appears too similar to code of a higher-level computer language, it is probably ***bad VHDL code***. **This is vitally important.**

Have a general concept of what your hardware should look like

1. Although VHDL is vastly powerful, if you do not understand basic digital constructs, you will probably be ***unable to generate efficient digital circuits***.
2. Digital design is similar to higher-level language programming in that even the most complicated programming at any level can be broken down into some simple programming constructs. There is a strong analogy to digital design in that even the most complicated digital circuits can be described in terms of basic digital constructs.
3. In other words, if you are not able to roughly envision the digital circuit you are trying to model in terms of basic digital circuits, you will probably ***misuse VHDL***, thus angering the VHDL gods. VHDL is cool, but it is not as magical as it initially appears to be.

Starting Point of VHDL

- VHDL is not case sensitive
  - ABCD, AbCd, abcd are the same
- VHDL is not sensitive to white space (spaces and tabs)
  - “ab or c” is equal to “ab   or   c”
- Comments in VHDL begin with the symbol “--”
  - The VHDL synthesizer ignores anything after the two dashes and up to the end of the line in which the dashes appear.
- VHDL is relatively lax (寬鬆) on its requirement for using parentheses
- VHDL statement is terminated with a semicolon (“;”)

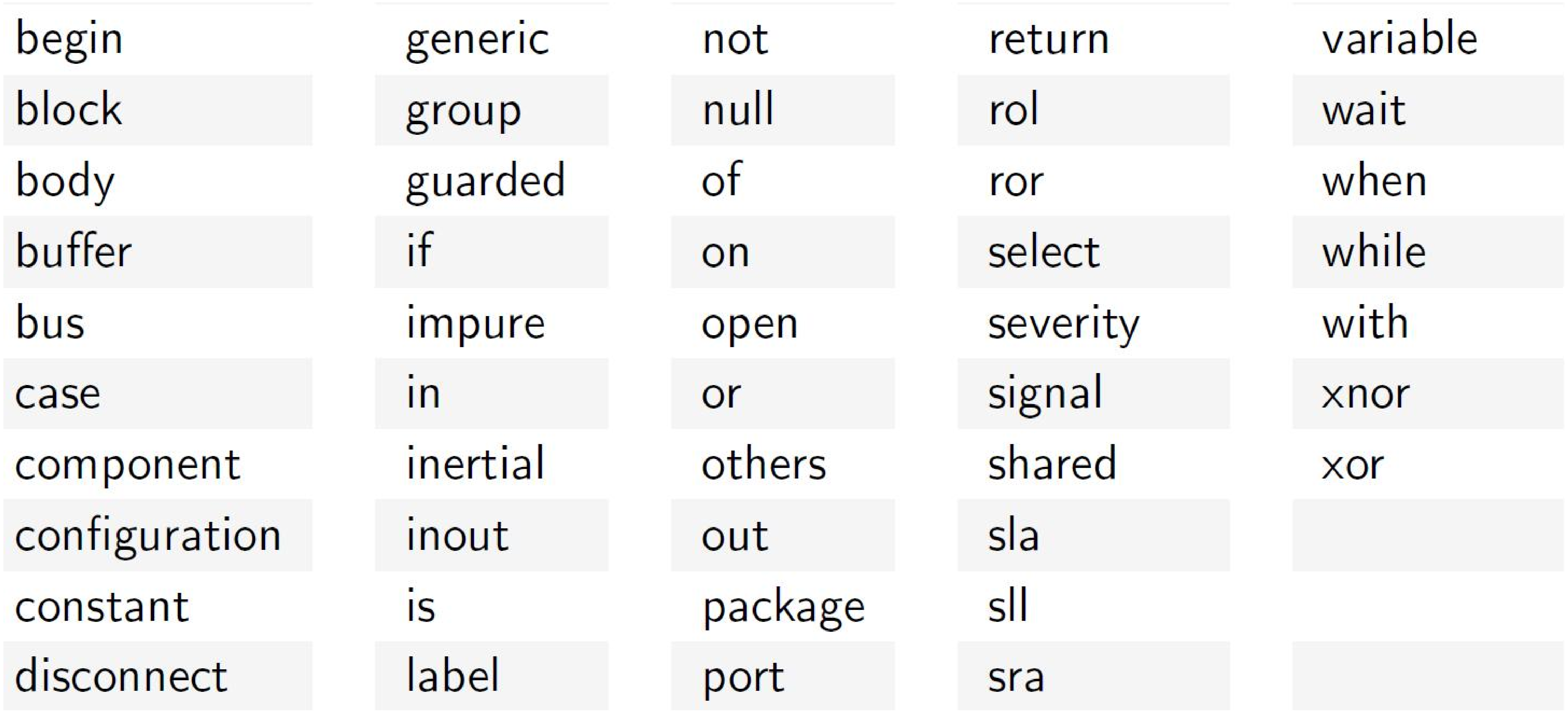
Identifiers (names)

- Identifiers can be as long as you want
- Identifiers can only contain a combination of letters (A-Z and a-z), digits (0-9) and the underscore character (“_”)
- Identifiers must start with an alphabetic character (A-Z and a-z)
- Identifiers must not end with an underscore (“_”) and must never have two consecutive underscores (“__”)

Reserved Words (1)

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

Reserved Words (2)

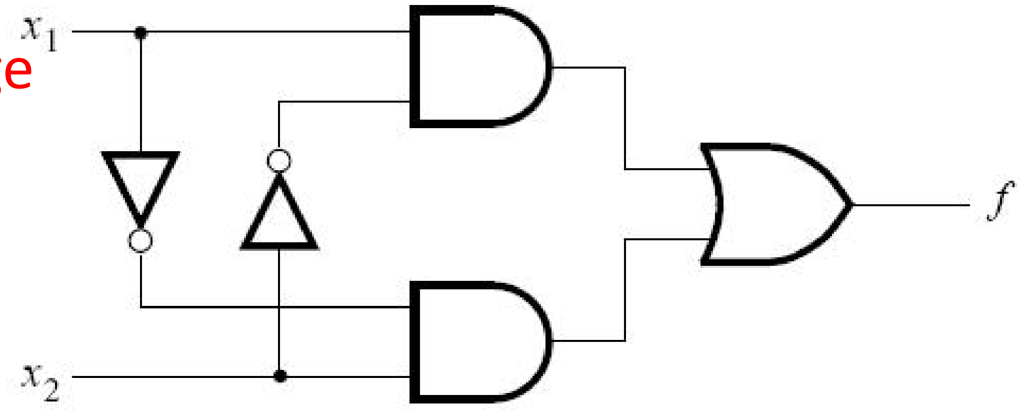


begin	generic	not	return	variable
block	group	null	rol	wait
body	guarded	of	ror	when
buffer	if	on	select	while
bus	impure	open	severity	with
case	in	or	signal	xnor
component	inertial	others	shared	xor
configuration	inout	out	sla	
constant	is	package	sll	
disconnect	label	port	sra	

VHDL Example

$$f = (x_1\,\overline{x_2}) + (\overline{x_1}\,x_2)$$

```
LIBRARY ieee ; -- predefined commonly used items
USE ieee.std_logic_1164.all ; -- individual package
ENTITY light IS
    PORT (  x1, x2 : IN STD_LOGIC ;
            f : OUT STD_LOGIC ) ;
END light ;

ARCHITECTURE LogicFunction OF light IS
BEGIN
    f <= (x1 AND NOT x2) OR (NOT x1 AND x2);
END LogicFunction ;
```



x_1	x_2	f
0	0	0
0	1	1
1	0	1
1	1	0

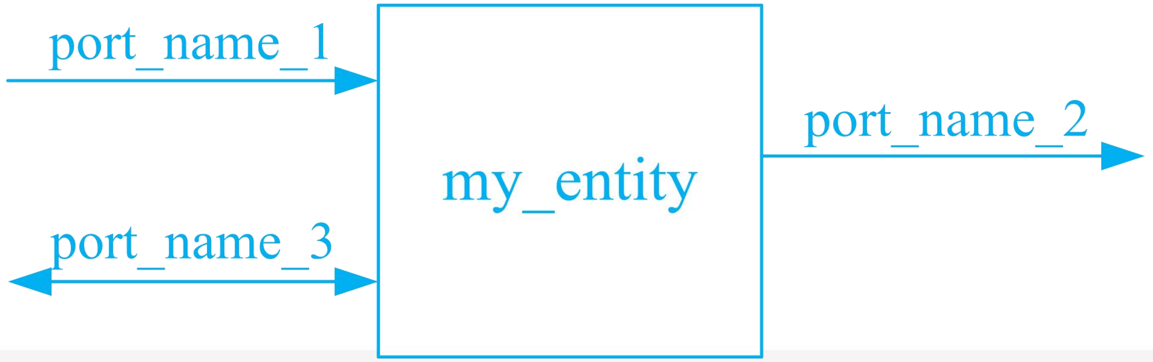
Library

- Libraries provide a set of packages, components, and functions that simplify the task of designing hardware
- Packages provide a collection of related data types and subprograms
- The following is an example of the use of the ***ieee*** library and its ***std_logic_1164*** package:

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

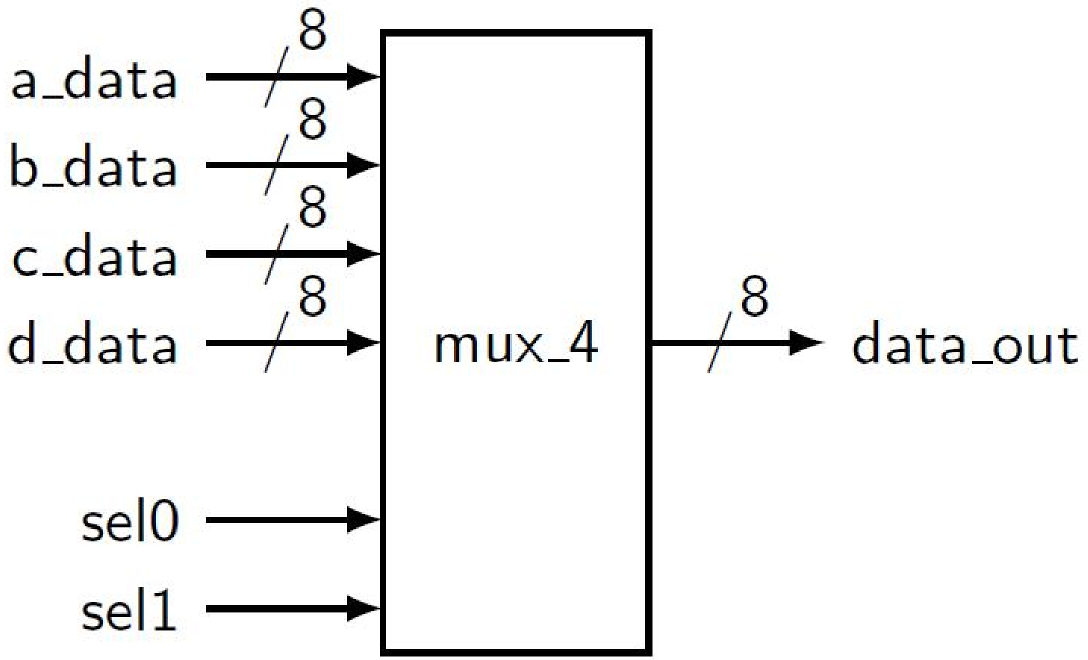
Entities

- An entity is a specification of the design's external interface
- Entry declarations specify the following:
    1. Then name of the entity
    2. A set of generic declarations specifying instance-specific parameters



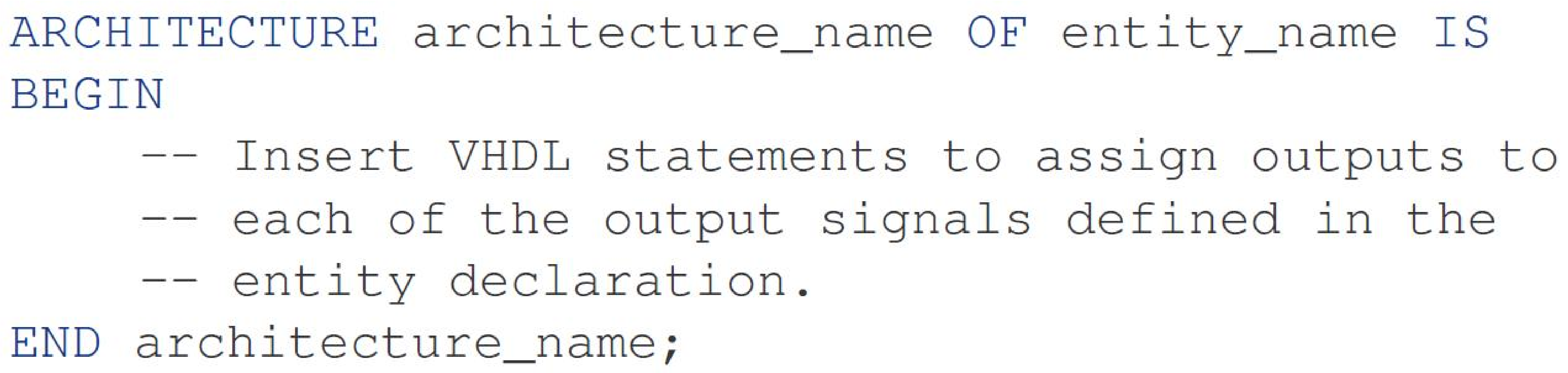
```
entity my_entity is
port(
     port_name_1 : in    std_logic ;
     port_name_2 : out   std_logic;
     port_name_3 : inout std_logic ); --do not forget the semicolon
end my_entity; -- do not forget this semicolon either
```

```
entity mux4 is
port (  a_data     : in      std_logic_vector(0 to 7);
        b_data     : in      std_logic_vector(0 to 7);
        c_data     : in      std_logic_vector(0 to 7);
        d_data     : in      std_logic_vector(0 to 7);
        sel1,sel0  : in      std_logic;
        data_out   : out     std_logic_vector(7 downto 0));
end mux4;
```



Architecture

- a specification of the design's internal implementation
- can be written by means of three modeling techniques plus any combination of these three. There is the data-flow model, the behavioral model, the structural model and the hybrid models.



```
ARCHITECTURE architecture_name OF entity_name IS
BEGIN
    -- Insert VHDL statements to assign outputs to
    -- each of the output signals defined in the
    -- entity declaration.
END architecture_name;
```

Signal and Variable

Signal

- Signal
  - represent wires and storage elements
  - may only be defined inside architectures
  - associated with a data type
  - have attributes
- signal sig_1 : std_logic;
  - sig_1 <= A;

Variable

- Variable
  - represent storage elements
  - used to store local information
  - may only be defined inside process
  - associated with a data type
  - have attributes
- variable var_1 : integer;
  - var_1 := 34;

VHDL is a strongly-typed language

Synthesis vs Simulation

- All synthesizable disigns can be simulted
- Not all simulation designs can be synthesized
- Consider the following VHDL code:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY simple_buffer IS
     PORT (   din       : IN     std_logic;
              dout      : OUT    std_logic );
END simple_buffer;

ARCHITECTURE behavioural1 OF simple_buffer IS
BEGIN
     dout <= din AFTER 10 ns;
END behavioural1;
```

Synthesis vs Simulation

- The input din is assigned to dout after 10ns
  - Can this represent a real-world system? **YES**
  - Can this be implemented in a device? **PERHAPS**
  - Can this be implemented in all devices? **NO**
- This architecture can be simulated but not synthesized
- Some VHDLdesign entry tools only permit the use of synthesizable keywords
- Most tools understand a synthesizable subset of VHDL93

Logic Operators

- Logical operators supported by VHDL
  - **AND**, **OR**, **NAND**, **NOR**, **XOR**, **XNOR**, **NOT**
- Relational operators
  - **=** (equal), **/=** (not equal), **<** (less than), **>** (greater than), **<=** (less than or equal to), **>=** (greater than or equal to)
- Mathematical operators
  - **+** (addition), **-** (subtraction), ***** (multiplication), **/** (division), **ABS** (absolute value), **MOD** (modulus), **REM** (remainder), ****** (exponent)
- VHDL also supports the overloading of existing operators and the creation of new operators using functions
  - Different library may provide different operators

Assignment Statements

```
SIGNAL a, b, c              : std_logic;
SIGNAL avec, bvec, cvec     : std_logic_vector(7 DOWNTO 0);

-- Concurrent Signal Assignment Statements
-- NOTE: Both a and avec are produced concurrently
a        <= b AND c;
avec     <= bvec OR cvec;

-- Alternatively, signals may be assigned constants
a        <= ’0’;
b        <= ’1’;
c        <= ’Z’;
avec     <= "00111010";              -- Assigns 0x3A to avec
bvec     <= X"3A";                   -- Assigns 0x3A to bvec
```

Assignment Statements

```
SIGNAL a, b, c, d          :std_logic;
SIGNAL avec                :std_logic_vector(1 DOWNTO 0);
SIGNAL bvec                :std_logic_vector(2 DOWNTO 0);

-- Conditional Assignment Statement
-- NOTE: This implements a tree structure of logic gates
a <=     '0'      WHEN avec = "00" ELSE
         b        WHEN avec = "11" ELSE
         c        WHEN d = '1' ELSE
         '1';

-- Selected Signal Assignment Statement
-- NOTE: The selection values must be constants
bvec <= d & avec;
WITH bvec SELECT
a <=     '0'      WHEN "000",
         b        WHEN "011",
         c        WHEN "1--",      -- Some tools won't synthesize '-' properly
         '1'      WHEN OTHERS;
```

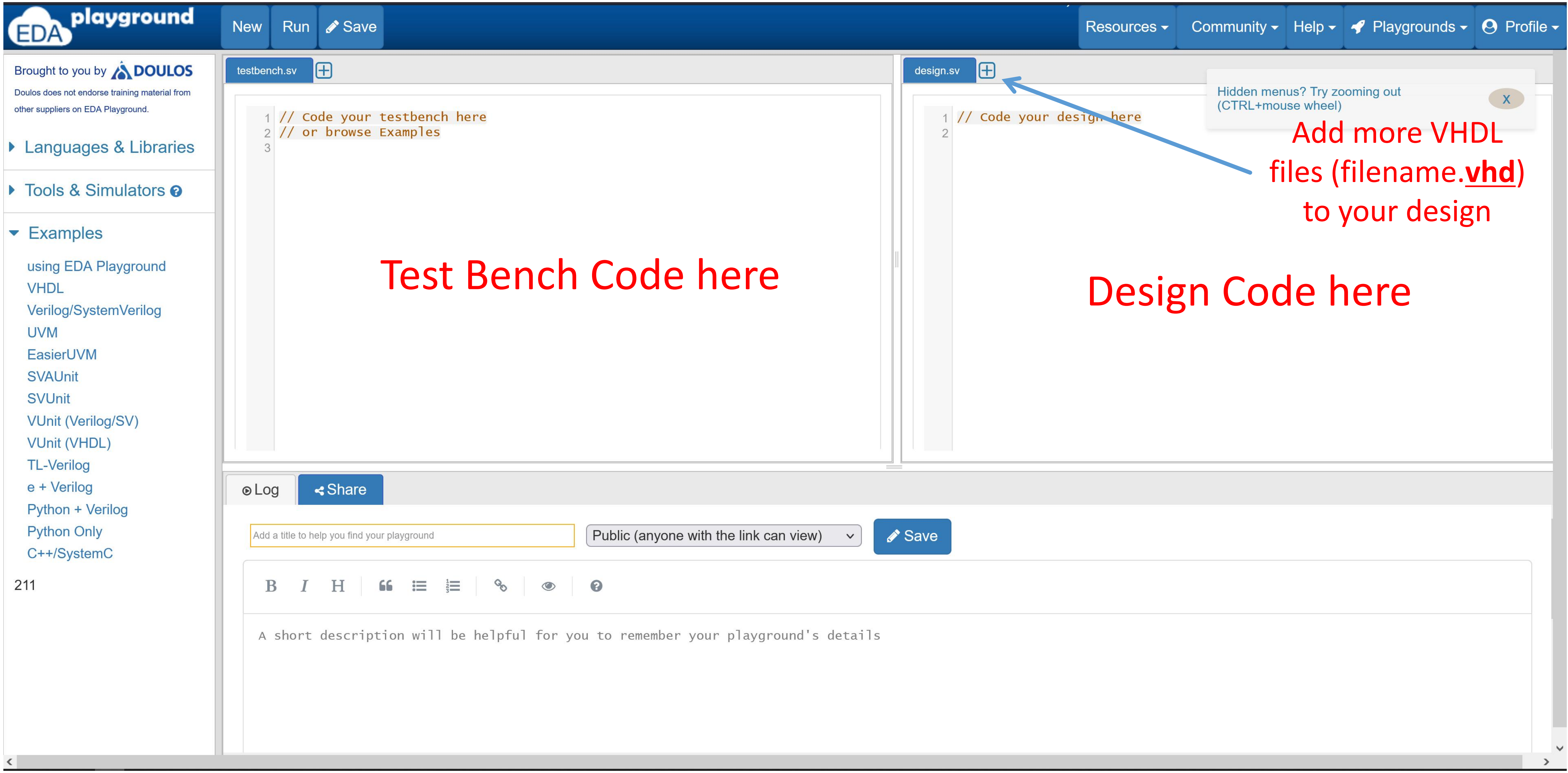
```
-- Selected Signal Assignment Statement
-- NOTE: Selected signal assignments also work
--        with vectors
WITH a SELECT
avec <= "01010101"        WHEN '1',
        bvec              WHEN OTHERS;
```

EDA Playground

EDA Playground lets you type in and run HDL code (using a selection of free and commercial simulators and synthesizers). It's great for learning HDLs, it's great for testing out unfamiliar things and it's great for sharing code.

EDA Playground account

- https://www.edaplayground.com/home
- sign in / log in
  - You can start typing straight away. But to run your code, you'll need to sign or log in. Logging in with a Google account gives you access to all non-commercial simulators and some commercial simulators.
  - To run commercial simulators, you need to register and log in with a username and password. Registration is free, and only pre-approved email's will have access to the commercial simulators.
  - To prevent your validation from being disabled, please supply your company or institution email address. Access will not be granted to freely available email addresses (eg gmail, 126, etc.).
  - You may use your um email account to do registration.



```
// Code your testbench here
// or browse Examples
```

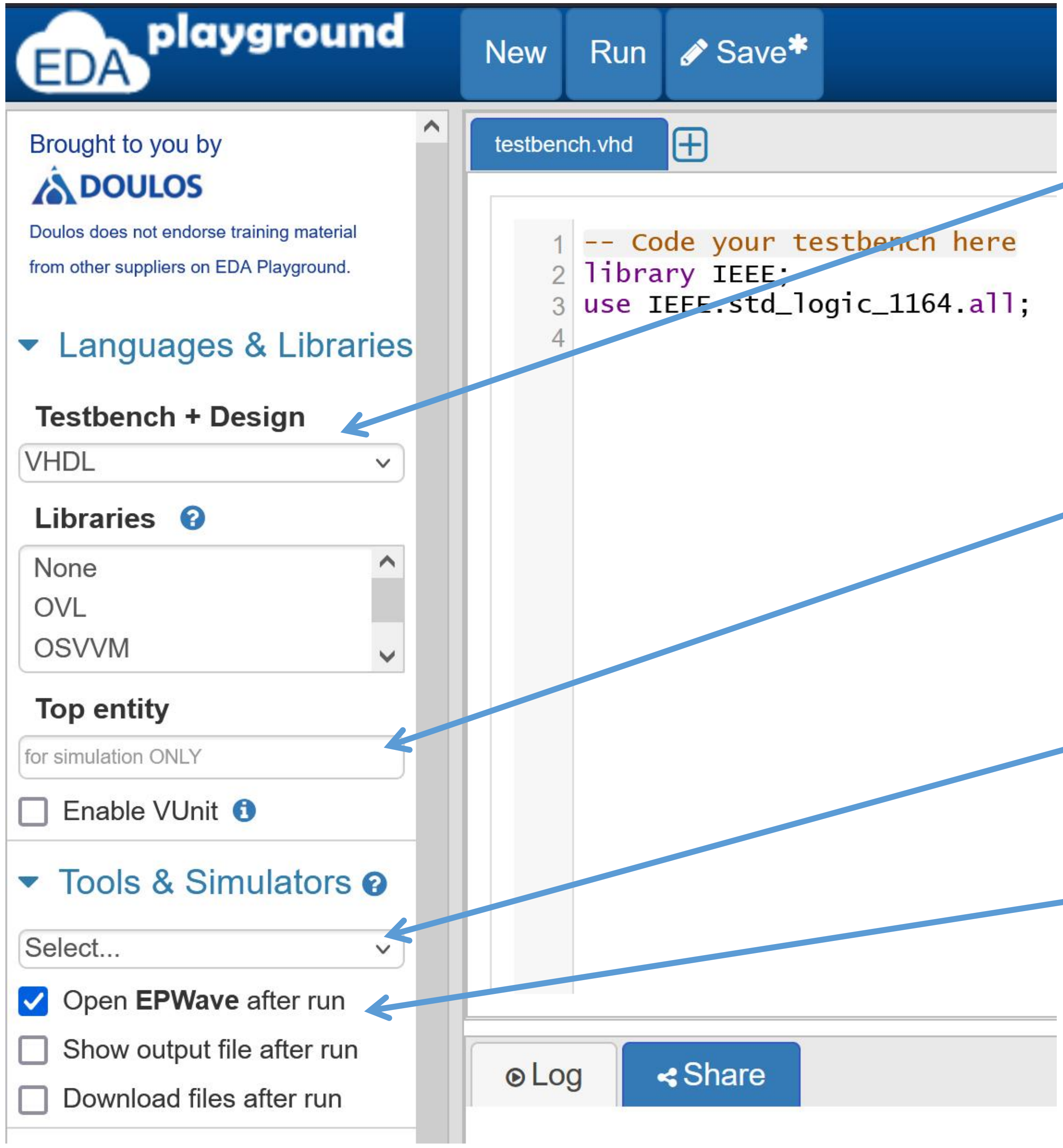
```
// Code your design here
```

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

Add more VHDL files (filename.**vhd**) to your design

Test Bench Code here

Design Code here



```
-- Code your testbench here
library IEEE;
use IEEE.std_logic_1164.all;
```

Testbench + Design: VHDL

Libraries: None, OVL, OSVVM

Top entity: for simulation ONLY

[ ] Enable VUnit

Tools & Simulators: Select...

[✓] Open **EPWave** after run

[ ] Show output file after run

[ ] Download files after run

1. Select VHDL here

2. type your top entity name here (the test bench entity name)

3. Select GHDL here

4. Check this option

EDA Playground Help

- Quick Start
  - Log in. Click the Log in button (top right) Then either
    - click on Google or Facebook or
    - register by clicking on ‘Register for a full account’ (which enables all the simulators on EDA Playground)
  - Select your language from the Testbench + Design menu.
  - Select your simulator from the Tools & Simulators menu. Using certain simulators will require you to supply additional identifcation information.
  - Type in your code in the testbench and design windows.
  - Click Run.
- Tutorial http://eda-playground.readthedocs.io/en/latest/tutorial.html

Example in EDA Playground

3 to 8 decoder

design.vhd

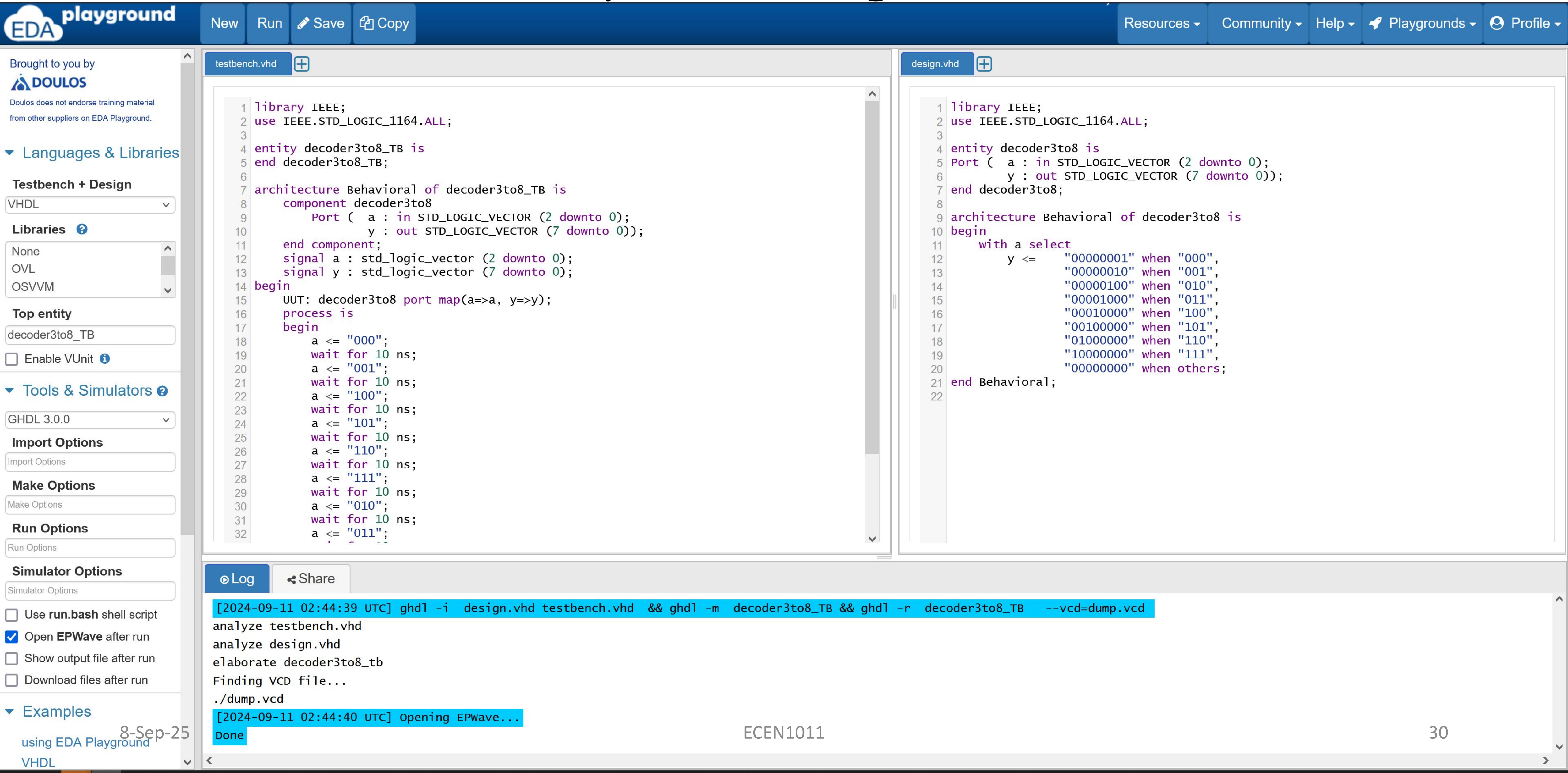
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity decoder3to8 is
Port (a : in STD_LOGIC_VECTOR (2 downto 0);
        y : out STD_LOGIC_VECTOR (7 downto 0));
end decoder3to8;
```

```
architecture Behavioral of decoder3to8 is
begin
    with a select
        y <=    "00000001" when "000",
                "00000010" when "001",
                "00000100" when "010",
                "00001000" when "011",
                "00010000" when "100",
                "00100000" when "101",
                "01000000" when "110",
                "10000000" when "111",
                "00000000" when others;
end Behavioral;
```

testbench.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity decoder3to8_TB is
end decoder3to8_TB;

architecture Behavioral of decoder3to8_TB is
    component decoder3to8
        Port (   a : in STD_LOGIC_VECTOR (2 downto 0);
                 y : out STD_LOGIC_VECTOR (7 downto 0));
    end component;
    signal a : std_logic_vector (2 downto 0);
    signal y : std_logic_vector (7 downto 0);
begin
    UUT: decoder3to8 port map(a=>a, y=>y);

    process is
    begin
        a <= "000";
        wait for 10 ns;
        a <= "001";
        wait for 10 ns;
        a <= "010";
        wait for 10 ns;
        a <= "011";
        wait for 10 ns;
        a <= "100";
        wait for 10 ns;
        a <= "101";
        wait for 10 ns;
        a <= "110";
        wait for 10 ns;
        a <= "111";
        wait for 10 ns;
        wait;
    end process;
end Behavioral;
```

Save and “RUN” your design



Testbench + Design: VHDL

Libraries: None, OVL, OSVVM

Top entity: decoder3to8_TB

☐ Enable VUnit

Tools & Simulators: GHDL 3.0.0

Import Options: Import Options

Make Options: Make Options

Run Options: Run Options

Simulator Options: Simulator Options

☐ Use **run.bash** shell script
☑ Open **EPWave** after run
☐ Show output file after run
☐ Download files after run

testbench.vhd

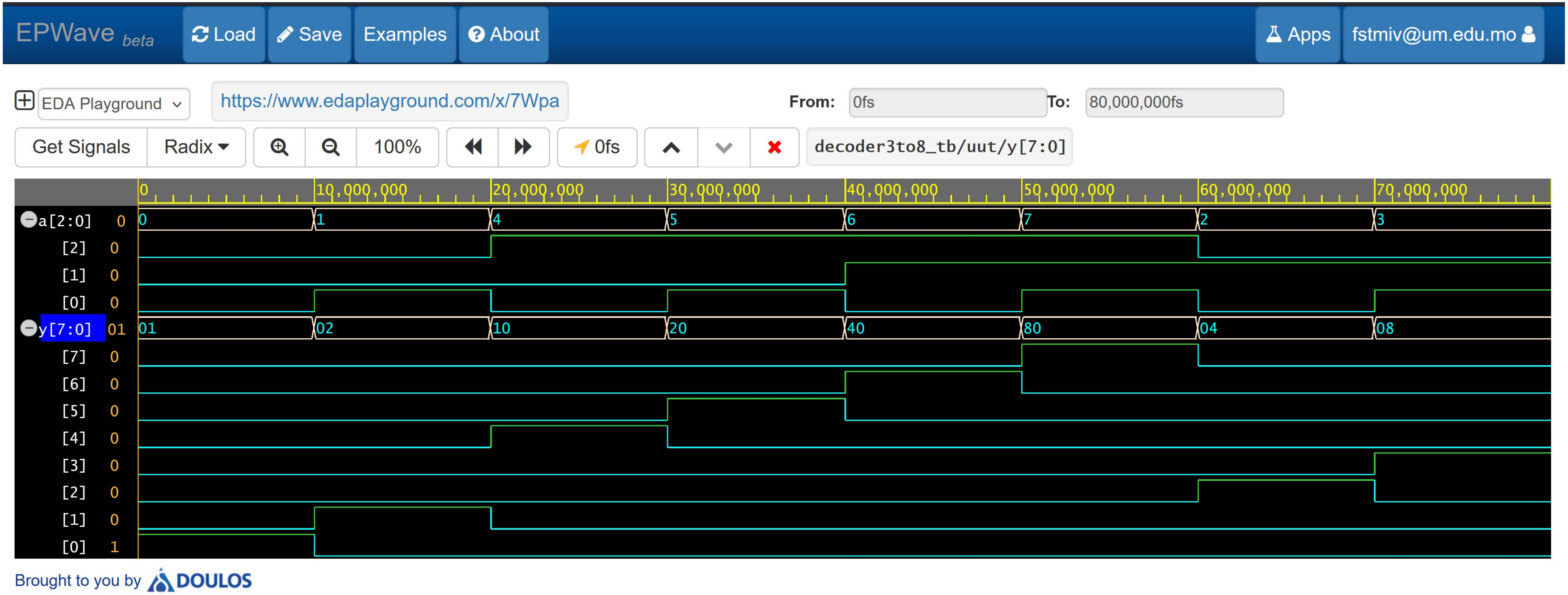
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity decoder3to8_TB is
end decoder3to8_TB;

architecture Behavioral of decoder3to8_TB is
    component decoder3to8
        Port (  a : in STD_LOGIC_VECTOR (2 downto 0);
                y : out STD_LOGIC_VECTOR (7 downto 0));
    end component;
    signal a : std_logic_vector (2 downto 0);
    signal y : std_logic_vector (7 downto 0);
begin
    UUT: decoder3to8 port map(a=>a, y=>y);
    process is
    begin
        a <= "000";
        wait for 10 ns;
        a <= "001";
        wait for 10 ns;
        a <= "100";
        wait for 10 ns;
        a <= "101";
        wait for 10 ns;
        a <= "110";
        wait for 10 ns;
        a <= "111";
        wait for 10 ns;
        a <= "010";
        wait for 10 ns;
        a <= "011";
```

design.vhd

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity decoder3to8 is
Port (  a : in STD_LOGIC_VECTOR (2 downto 0);
        y : out STD_LOGIC_VECTOR (7 downto 0));
end decoder3to8;

architecture Behavioral of decoder3to8 is
begin
    with a select
        y <=    "00000001" when "000",
                "00000010" when "001",
                "00000100" when "010",
                "00001000" when "011",
                "00010000" when "100",
                "00100000" when "101",
                "01000000" when "110",
                "10000000" when "111",
                "00000000" when others;
end Behavioral;

```

```
[2024-09-11 02:44:39 UTC] ghdl -i  design.vhd testbench.vhd  && ghdl -m  decoder3to8_TB && ghdl -r  decoder3to8_TB   --vcd=dump.vcd
analyze testbench.vhd
analyze design.vhd
elaborate decoder3to8_tb
Finding VCD file...
./dump.vcd
[2024-09-11 02:44:40 UTC] Opening EPWave...
Done
```

Simulation result



From: 0fs To: 80,000,000fs

decoder3to8_tb/uut/y[7:0]

```
              0           10,000,000  20,000,000  30,000,000  40,000,000  50,000,000  60,000,000  70,000,000
a[2:0]   0    0           1           4           5           6           7           2           3
  [2]    0
  [1]    0
  [0]    0
y[7:0]   01   01          02          10          20          40          80          04          08
  [7]    0
  [6]    0
  [5]    0
  [4]    0
  [3]    0
  [2]    0
  [1]    0
  [0]    1
```