Budapest University of Technology and Economics
Department of Electron Devices

Lab. 1
Digital system design
Introduction, language structure, examples

Osama Ali
osamaalisalman.khafajy@edu.bme.hu
Ahmad Halal
ahmadhalalftesah@edu.bme.hu

6 of May 2022

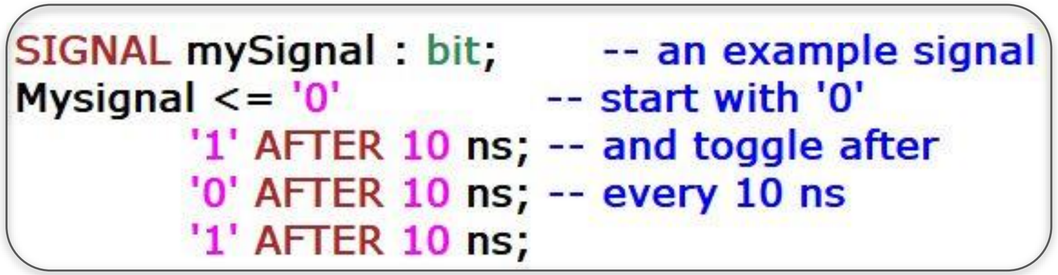
Outline

- VHDL language structure
- Architecture
- Deferent styles
- Signal & Variables

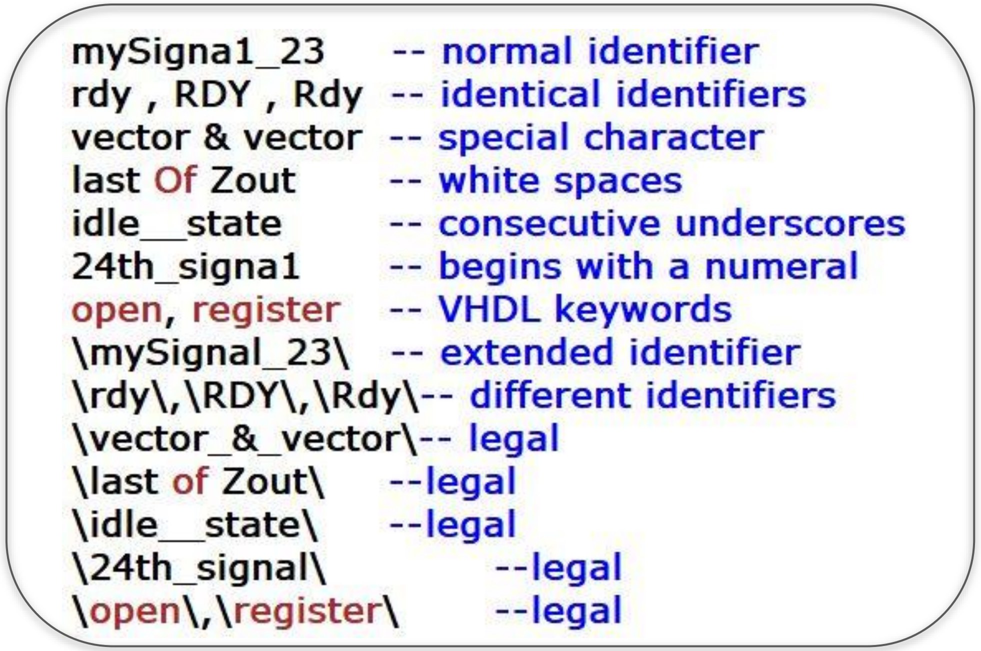
General Considerations

- Case insensitive
- Comments: ’ --’ until end of line
- Statements are terminated by ‘**;**’
- List delimiter: ‘**,**’
- Signal assignment: ‘**<=**’
- User defined names:
  - letters, numbers, underscores
  - start with a letter!
  - underscores

It is an example don't write like that in VHDL



```
SIGNAL mySignal : bit;       -- an example signal
Mysignal <= '0'            -- start with '0'
        '1' AFTER 10 ns; -- and toggle after
        '0' AFTER 10 ns; -- every 10 ns
        '1' AFTER 10 ns;
```



```
mySigna1_23       -- normal identifier
rdy , RDY , Rdy   -- identical identifiers
vector & vector   -- special character
last Of Zout      -- white spaces
idle__state       -- consecutive underscores
24th_signa1       -- begins with a numeral
open, register    -- VHDL keywords
\mySignal_23\     -- extended identifier
\rdy\,\RDY\,\Rdy\-- different identifiers
\vector_&_vector\-- legal
\last of Zout\    --legal
\idle__state\     --legal
\24th_signal\         --legal
\open\,\register\     --legal
```

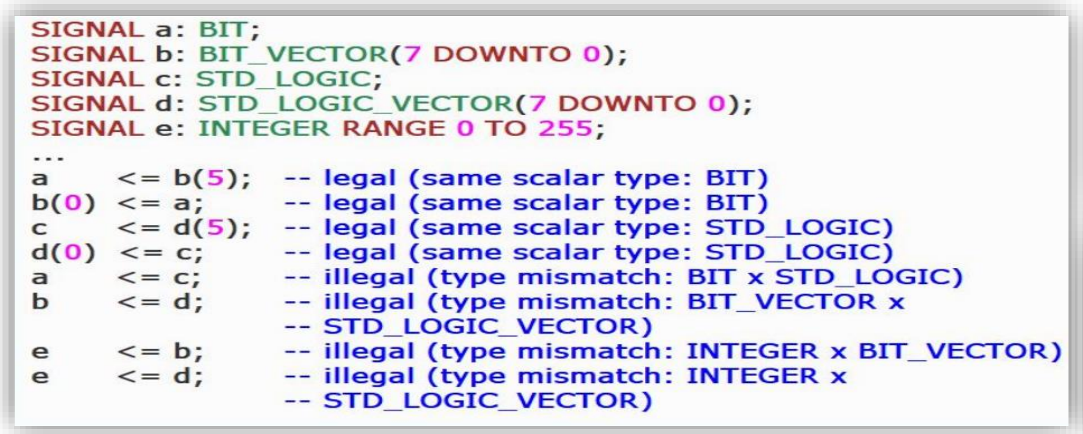
Data Types

- There is a series of pre-defined data types in VHDL through the standard and the IEEE libraries.
- Not all data types are synthesizable.

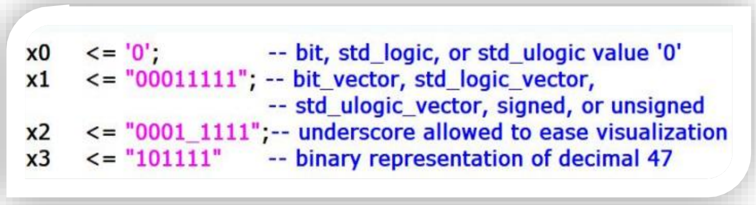
Package / library	Defined data types
Package standard of library std	BIT, BOOLEAN, INTEGER, and REAL
Package std_logic_1164 of library ieee	STD_LOGIC and STD_ULOGIC
Package std_logic_arith of library ieee	SIGNED and UNSIGNED

STD_LOGIC & STD_ULOGIC	
'1'	Logic 1 or High 1
'0'	Logic 0 or High 0
'Z'	High impedance
'W'	Weak signal, can't tell if 0 or 1
'L'	Weak 0, pulldown
'H'	Weak 1, pullup
'-'	Don't care
'U'	Uninitialized
'X'	Unknown, multiple drivers

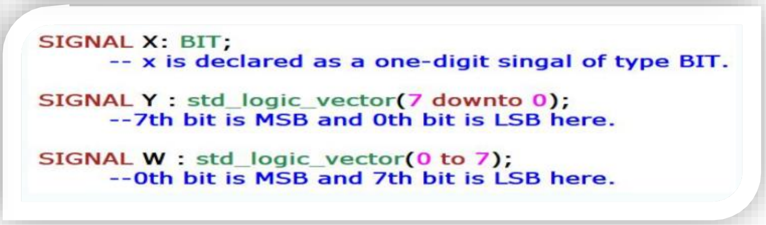
- STD_LOGIC (and STD_LOGIC_VECTOR): **8-valued** logic system introduced in the IEEE 1164 standard.
- BOOLEAN: True, False.
- INTEGER: 32-bit integers (from -2,147,483,647 to +2,147,483,647).



```
SIGNAL a: BIT;
SIGNAL b: BIT_VECTOR(7 DOWNTO 0);
SIGNAL c: STD_LOGIC;
SIGNAL d: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL e: INTEGER RANGE 0 TO 255;
...
a     <= b(5);  -- legal (same scalar type: BIT)
b(0)  <= a;     -- legal (same scalar type: BIT)
c     <= d(5);  -- legal (same scalar type: STD_LOGIC)
d(0)  <= c;     -- legal (same scalar type: STD_LOGIC)
a     <= c;     -- illegal (type mismatch: BIT x STD_LOGIC)
b     <= d;     -- illegal (type mismatch: BIT_VECTOR x
                -- STD_LOGIC_VECTOR)
e     <= b;     -- illegal (type mismatch: INTEGER x BIT_VECTOR)
e     <= d;     -- illegal (type mismatch: INTEGER x
                -- STD_LOGIC_VECTOR)
```



```
x0   <= '0';           -- bit, std_logic, or std_ulogic value '0'
x1   <= "00011111"; -- bit_vector, std_logic_vector,
                       -- std_ulogic_vector, signed, or unsigned
x2   <= "0001_1111";-- underscore allowed to ease visualization
x3   <= "101111"       -- binary representation of decimal 47
```



```
SIGNAL X: BIT;
    -- x is declared as a one-digit singal of type BIT.

SIGNAL Y : std_logic_vector(7 downto 0);
    --7th bit is MSB and 0th bit is LSB here.

SIGNAL W : std_logic_vector(0 to 7);
    --0th bit is MSB and 7th bit is LSB here.
```

VHDL Operators

- Operators can be used to implement any combinational circuit.

Operator type	Operators	Data types
Logical	NOT, AND NAND, OR, NOR, XOR, XNOR	BIT, BIT_VECTOR, STD_LOGIC, STD_LOGIC_VECTOR, STD_ULOGIC, STD_ULOGIC_VECTOR
Arithmetic	+, -, *, /, **, MOD, REM, ABS	INTEGER, SIGNED, UNSIGNED
Comparison	=, /=, <, >, <=, >=	All above
Shift	SLL, SRL, SLA, SRA, ROL, ROR	BIT_VECTOR
Concatenation	&, (, , ,)	Same as for logical operators, plus SIGNED and UNSIGNED

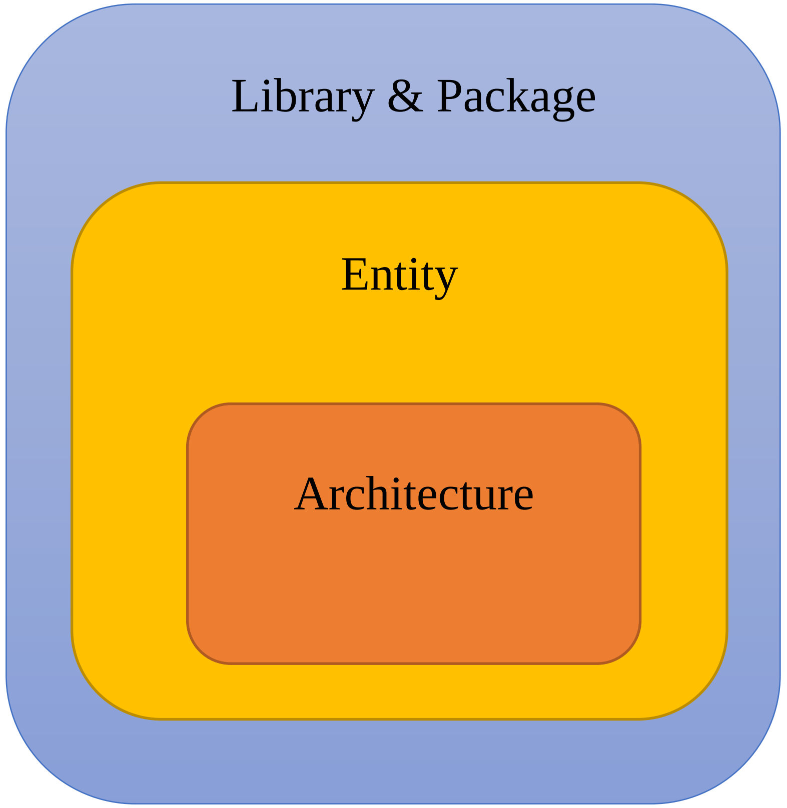
A VHDL design consist of three fundamental design units:

1. **Library and Package Declaration**
   - Library and packages are collection of commonly used items, such as data **types**, **subprograms**, and **components**.

```
library IEEE;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
```

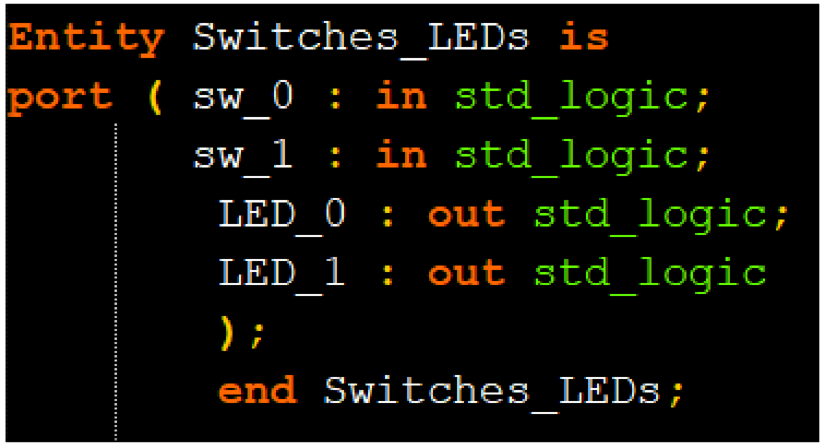
   - One can create package that includes several data types, constants, and components. After the IEEE library we can declare the new package.

```
library work;
use work.DataTypes_pkg.all;
```



2. Entity Declaration

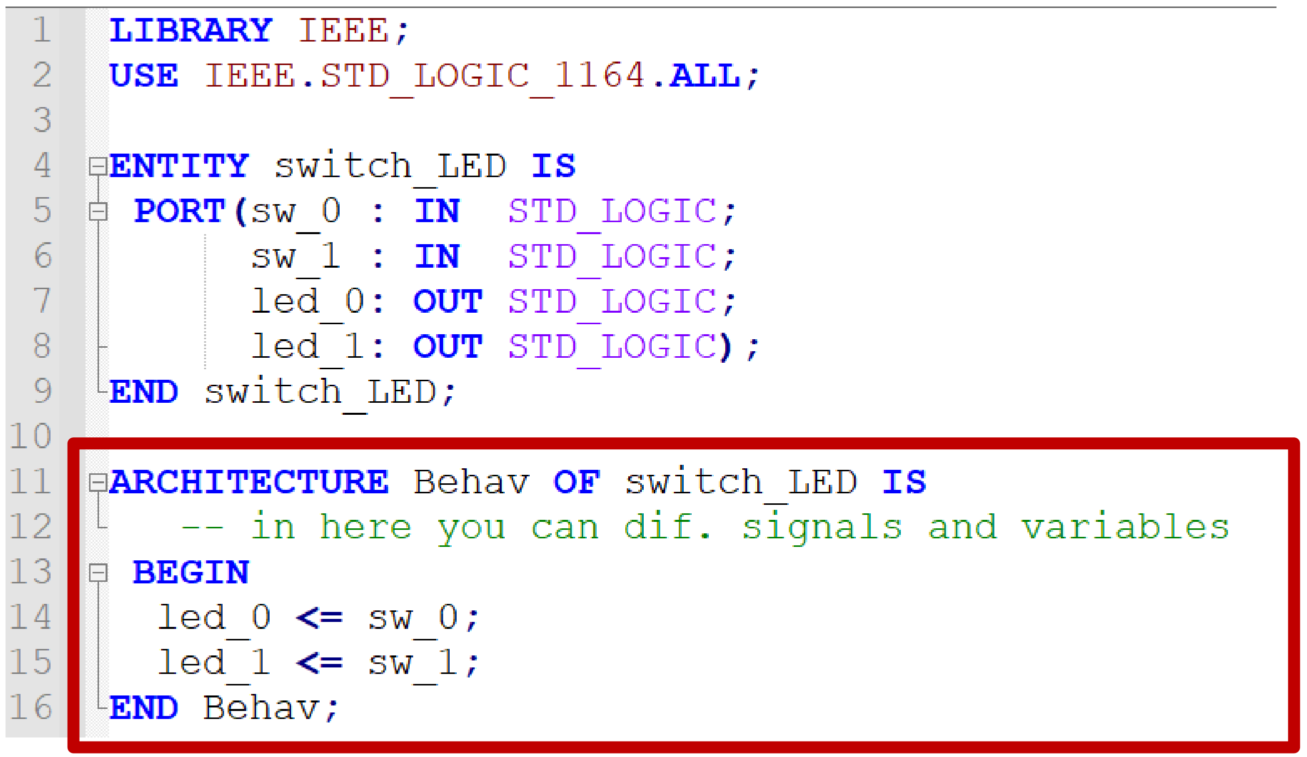
- **port_name**: used to identify pin(s) and providing the ability to connect it to the design unit or other designs units.
- **mode**: give the direction of the port. It can be **in**, **out**, or **inout**, as will be discussed later
- **data_type**: define the data type of the port which can be **bit**, **integer**, **std_logic**, and many other types.



```
Entity Switches_LEDs is
port ( sw_0 : in std_logic;
       sw_1 : in std_logic;
        LED_0 : out std_logic;
        LED_1 : out std_logic
        );
        end Switches_LEDs;
```

3. Architecture

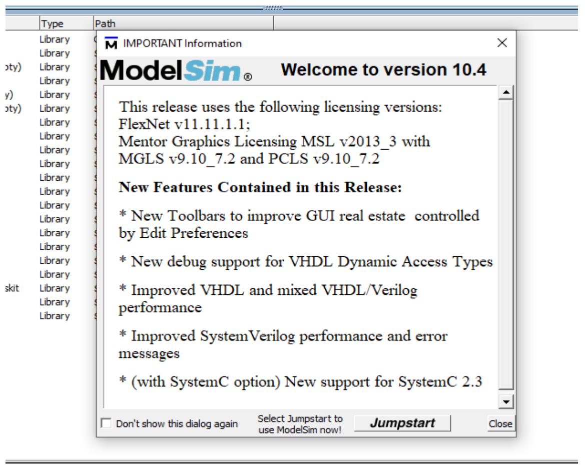
- **Implementation** of the design Always connected with a specific entity
  - One entity can have several architectures
  - Entity ports are available as signals within the architecture
- Contains concurrent statements



```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY switch_LED IS
 PORT(sw_0 : IN  STD_LOGIC;
      sw_1 : IN  STD_LOGIC;
      led_0: OUT STD_LOGIC;
      led_1: OUT STD_LOGIC);
END switch_LED;

ARCHITECTURE Behav OF switch_LED IS
   -- in here you can dif. signals and variables
 BEGIN
  led_0 <= sw_0;
  led_1 <= sw_1;
END Behav;
```

Creating a VHDL Project

- Start the ModelSim
- If the welcome screen popped-up, just press jumpstart.
- Then press Create a Project if you want to create a new one or select Open a Project to continue working on an already existing project. For our first practice we are going to create a new project.
- As shown in the below figure, we have to select the project name and also the project location where it will be saved. We can leave work as the as a library to save out design after compilation.


Type
Path
IMPORTANT Information
ModelSim
Don't show this dialog again
Select Jumpstart to use ModelSim now!
Jumpstart
Close

Welcome to version 10.4

This release uses the following licensing versions:
FlexNet v11.11.1.1;
Mentor Graphics Licensing MSL v2013_3 with
MGLS v9.10_7.2 and PCLS v9.10_7.2

New Features Contained in this Release:

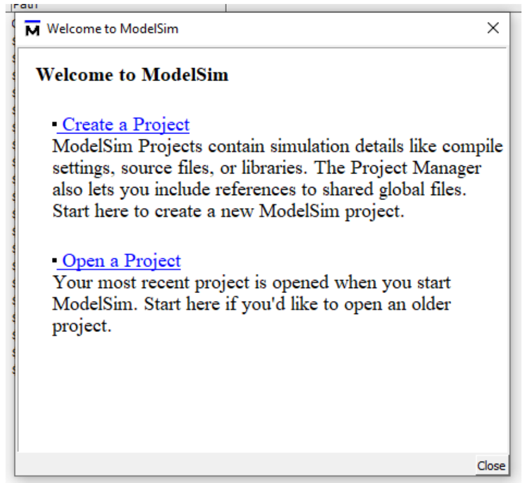
* New Toolbars to improve GUI real estate controlled by Edit Preferences

* New debug support for VHDL Dynamic Access Types

* Improved VHDL and mixed VHDL/Verilog performance

* Improved SystemVerilog performance and error messages

* (with SystemC option) New support for SystemC 2.3



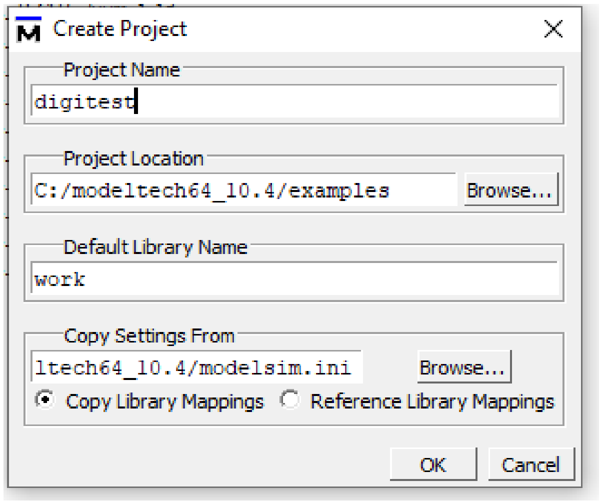
Welcome to ModelSim

- Create a Project

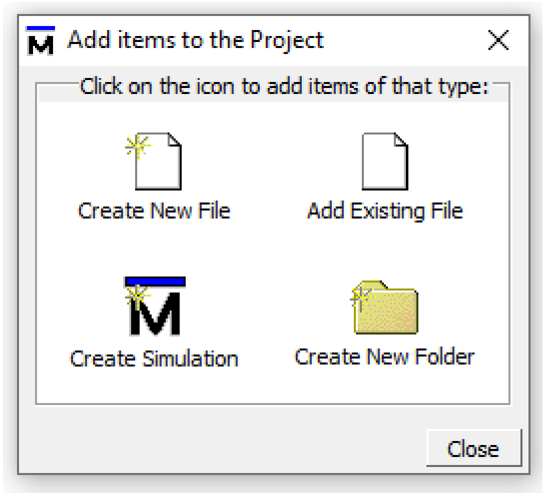
ModelSim Projects contain simulation details like compile settings, source files, or libraries. The Project Manager also lets you include references to shared global files. Start here to create a new ModelSim project.

- Open a Project

Your most recent project is opened when you start ModelSim. Start here if you'd like to open an older project.

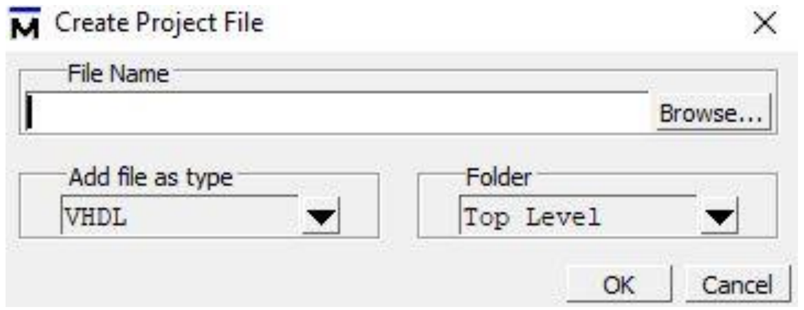


Project Name: digitest
Project Location: C:/modeltech64_10.4/examples
Default Library Name: work
Copy Settings From: ltech64_10.4/modelsim.ini
[✓ Copy Library Mappings] Reference Library Mappings



Adding the source code

- We are free to select any name for the project. However, it is preferred to be a descriptive name.
- It could be more convenient to make a dedicated folder (in advance) for saving the new project all files in a specific location. Let's make a new folder on the desktop, then we choose it as the project location. Then press OK.
- From the next window we add the source VHDL code. We have two options, if we have an already existing VHDL code (.vhd file), we can choose Add Existing File and we browse to it; the other option is to initiate a new source code by selecting Create New File . In the latest case we have to enter a name for the source code.


Create Project File

File Name

Add file as type: VHDL

Folder: Top Level

- If there are no more files, we can close the current window and start editing the source code.

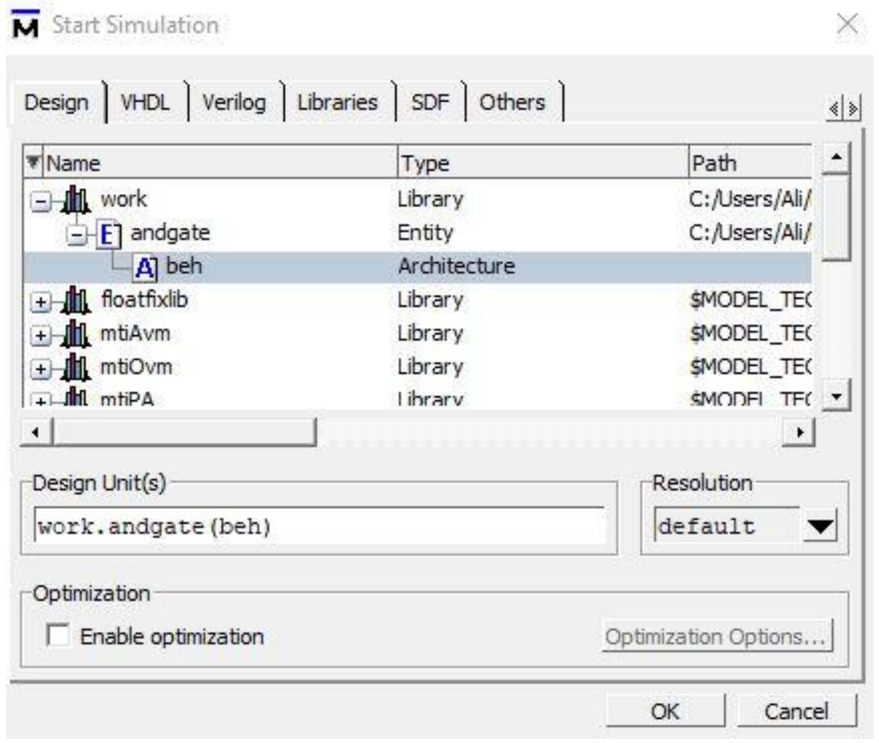
Editing the source code

- For editing the source code, we can use the build in editor of the ModelSim (double click will open the source file) or use any other editor like Sublime.
- After finish editing the code, we have to compile it to make sure that the syntax is correct and also to convert the source code to the objective code and move to the next step.
- For compiling the code there are different ways:
  - right click on the source file and chose compile,
  - from the Compile drop down menu, or
  - press the icon on the tool bar on the top of the window.
- After the first try for compilation, it is probable to get an error message related to some typos. To correct these error, double click on the red message appears in the Transcript sub-window and read the error description, then edit your code.
- After correcting all errors, you should see a message in green that states (Compile of source file was successful).

Start simulation

- For verifying the functional behavior of the design, we have to simulate the compiled design.
- To start the simulation, from the drop-down menu of Simulate we choose **Start Simulation**.

▪ A new window will appear to select the objective code that we want to simulate from the corresponding library.
▪ Recall that we have chosen work as the default library. Press on the + beside work and select the name of the entity and under it the name of the architecture that you want to simulate.
▪ Unselect the option of Enable Optimization and press OK.



Name	Type	Path
work	Library	C:/Users/Ali/
andgate	Entity	C:/Users/Ali/
beh	Architecture	
floatfixlib	Library	$MODEL_TEC
mtiAvm	Library	$MODEL_TEC
mtiOvm	Library	$MODEL_TEC
mtiPA	Library	$MODEL_TEC

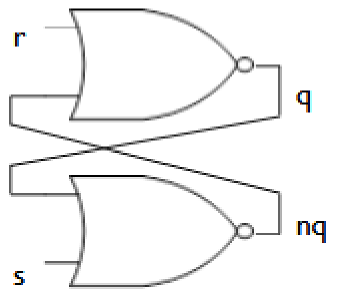
Design Unit(s): work.andgate(beh)
Resolution: default
Optimization: ☐ Enable optimization

Verifying the design

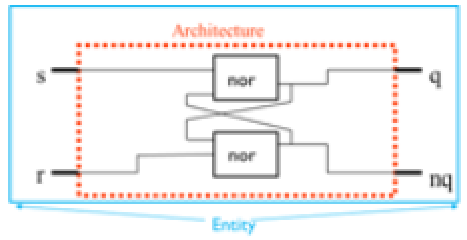
- To verify the design by applying different input pattern and observe the output we have to add the related signals to the wave window.
- From the Objects sub-window, select the signals that you want to monitor by keep pressing the Ctrl from the keyboard and the lift mouse button, then right click on any of them and choose Add » To Wave » Selected Signals. The wave window will appear with the selected signals.
- To verify the design, we have to provide the input signals with different pattern and check the output. From the wave window, right click on any of the input signals and select Force. In the value field enter the data according to the required test pattern and press OK.
- After doing this for input signals, now its time to start running the simulation by pressing the Run icon from the related tool bar.

Architecture Different styles

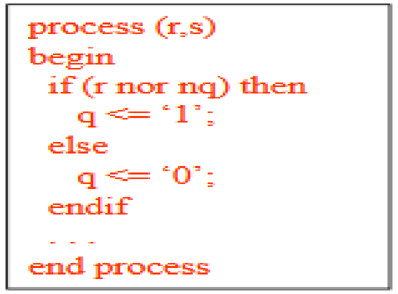
- **Data flow**
  - Descrubes how data flows from input to output



- Structural
  - How components are put together

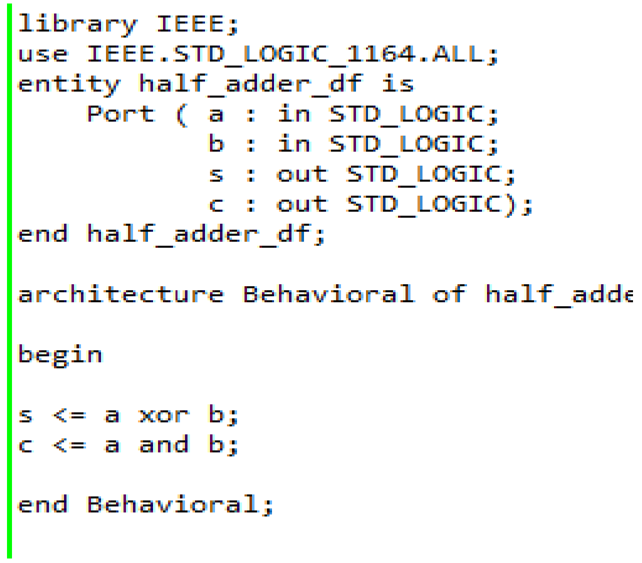


- Behavioral
  - Describes the behavior of the circuit most likely withing a process



```
process (r,s)
begin
 if (r nor nq) then
    q <= '1';
 else
    q <= '0';
 endif
 ...
end process
```

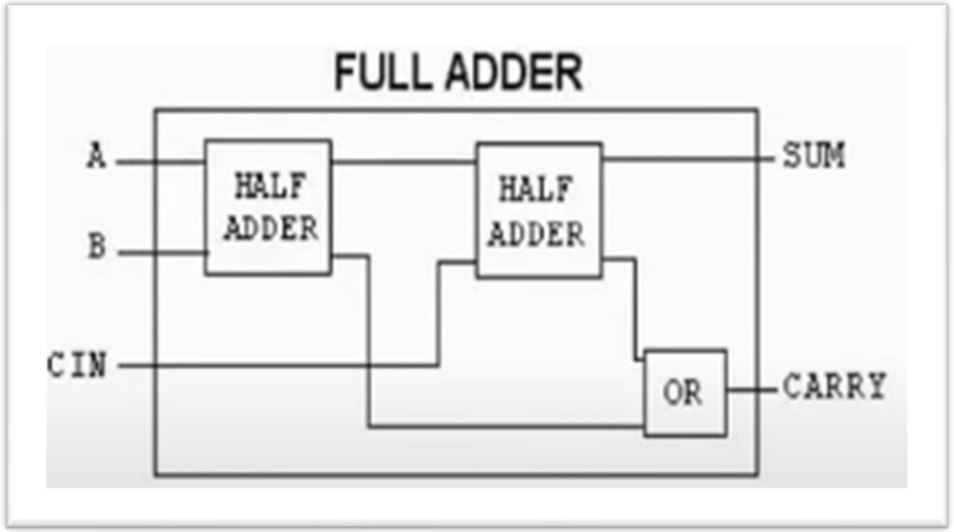
Data flow



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity half_adder_df is
    Port ( a : in STD_LOGIC;
           b : in STD_LOGIC;
           s : out STD_LOGIC;
           c : out STD_LOGIC);
end half_adder_df;

architecture Behavioral of half_adde

begin

s <= a xor b;
c <= a and b;

end Behavioral;
```

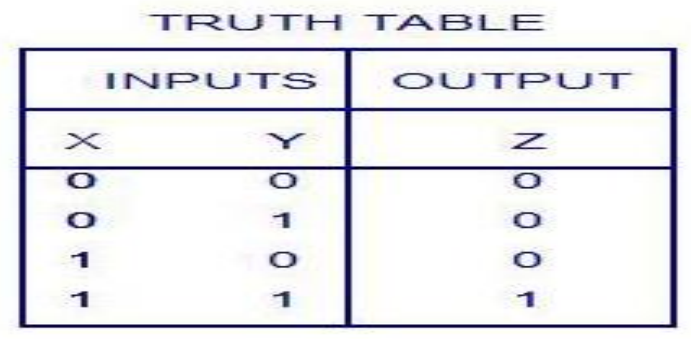
Structural

- Components from libraries are connected together
- Design are hierarchical
- Each component can be individually simulated



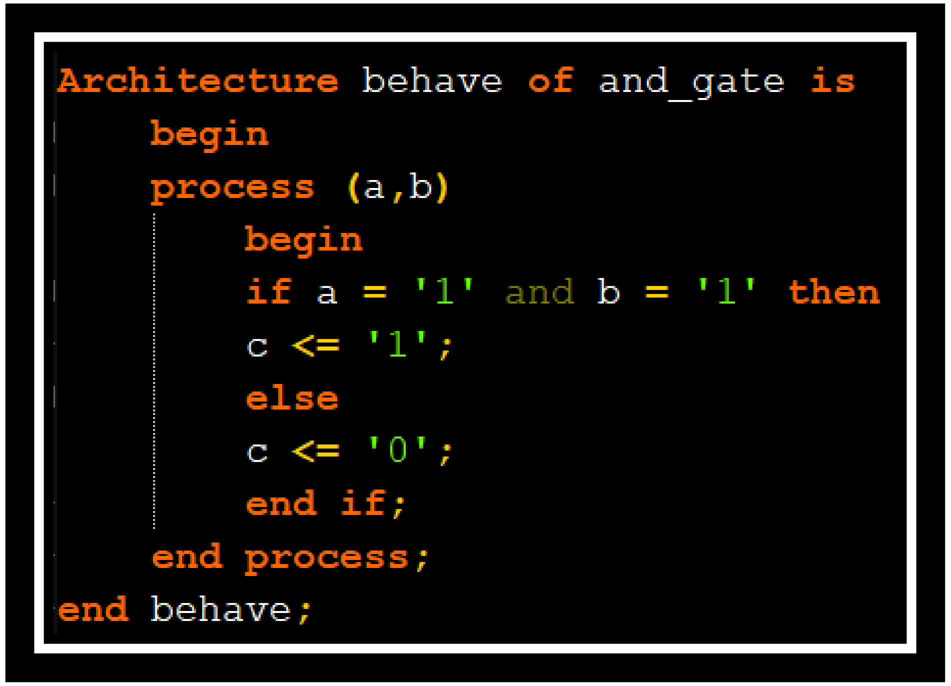
Behavioral

- Also known as High-level Descriptions
- Consists of a set of an assignment statements to represent the behavior
- No need to focus on the gate level implementation of a design
- But the execution most likely will be sequential



TRUTH TABLE

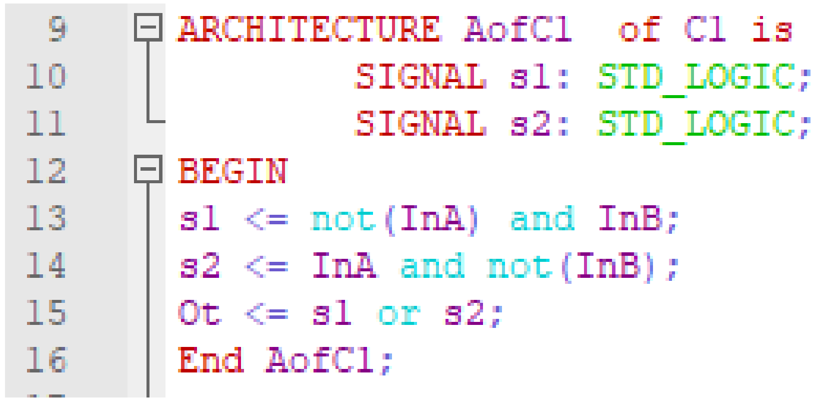
INPUTS		OUTPUT
X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1



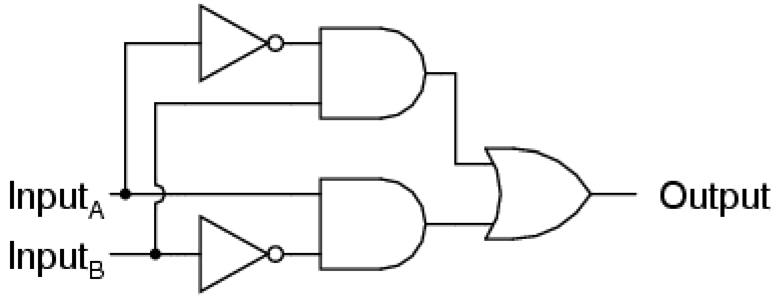
```
Architecture behave of and_gate is
    begin
    process (a,b)
        begin
        if a = '1' and b = '1' then
        c <= '1';
        else
        c <= '0';
        end if;
    end process;
end behave;
```

Signals and Variable

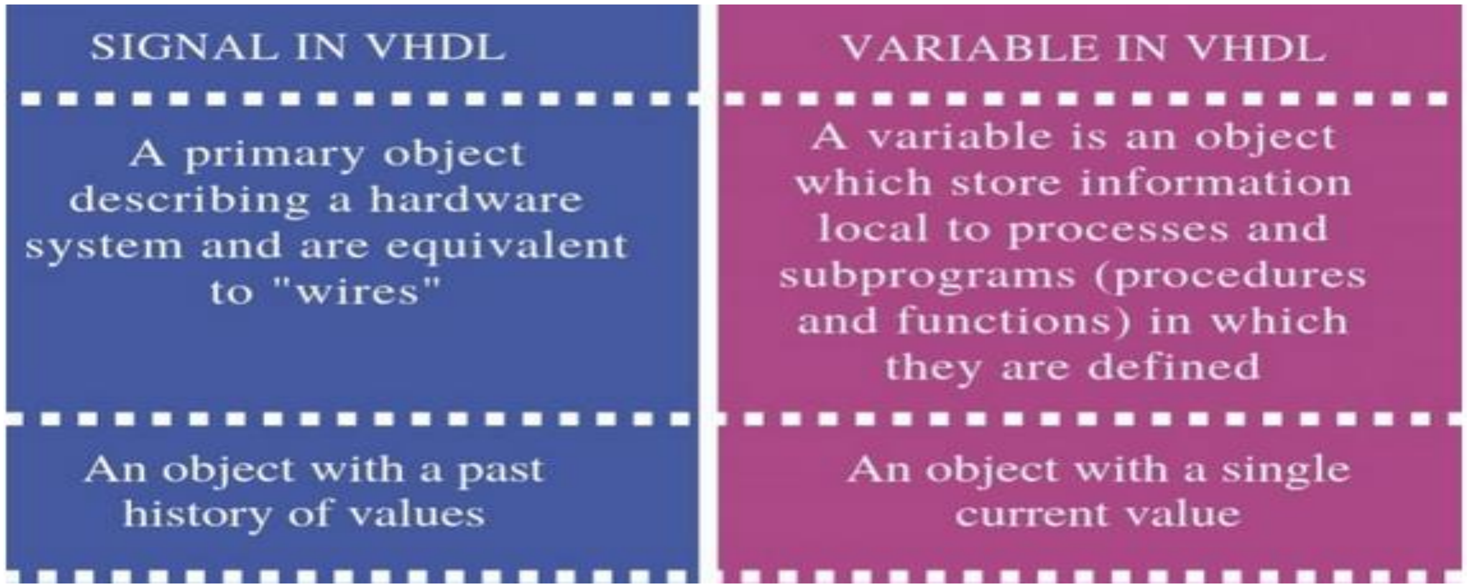
- Represent wires within the circuit.



```
ARCHITECTURE AofC1  of C1 is
        SIGNAL s1: STD_LOGIC;
        SIGNAL s2: STD_LOGIC;
BEGIN
s1 <= not(InA) and InB;
s2 <= InA and not(InB);
Ot <= s1 or s2;
End AofC1;
```

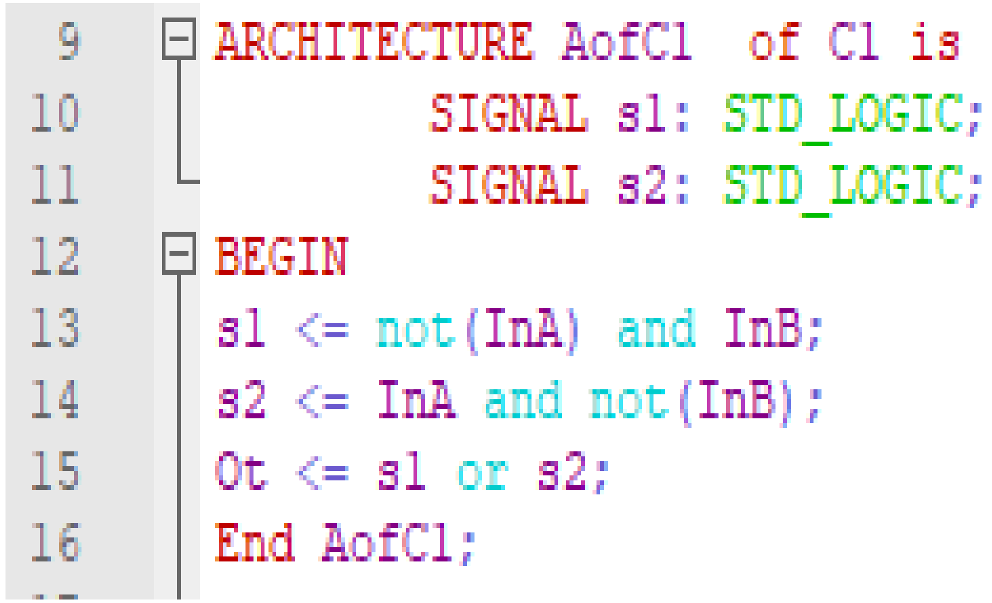


A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

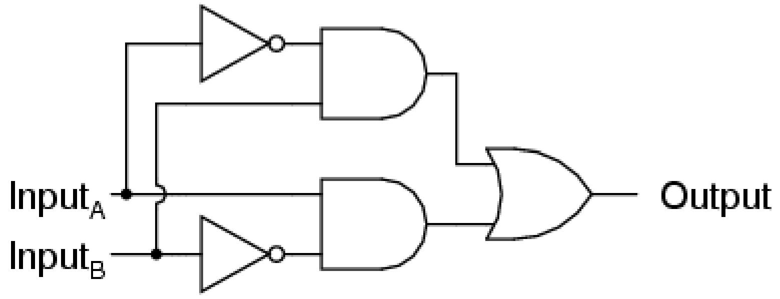


SIGNAL IN VHDL	VARIABLE IN VHDL
A primary object describing a hardware system and are equivalent to "wires"	A variable is an object which store information local to processes and subprograms (procedures and functions) in which they are defined
An object with a past history of values	An object with a single current value

Example: Write this also Test and Verify

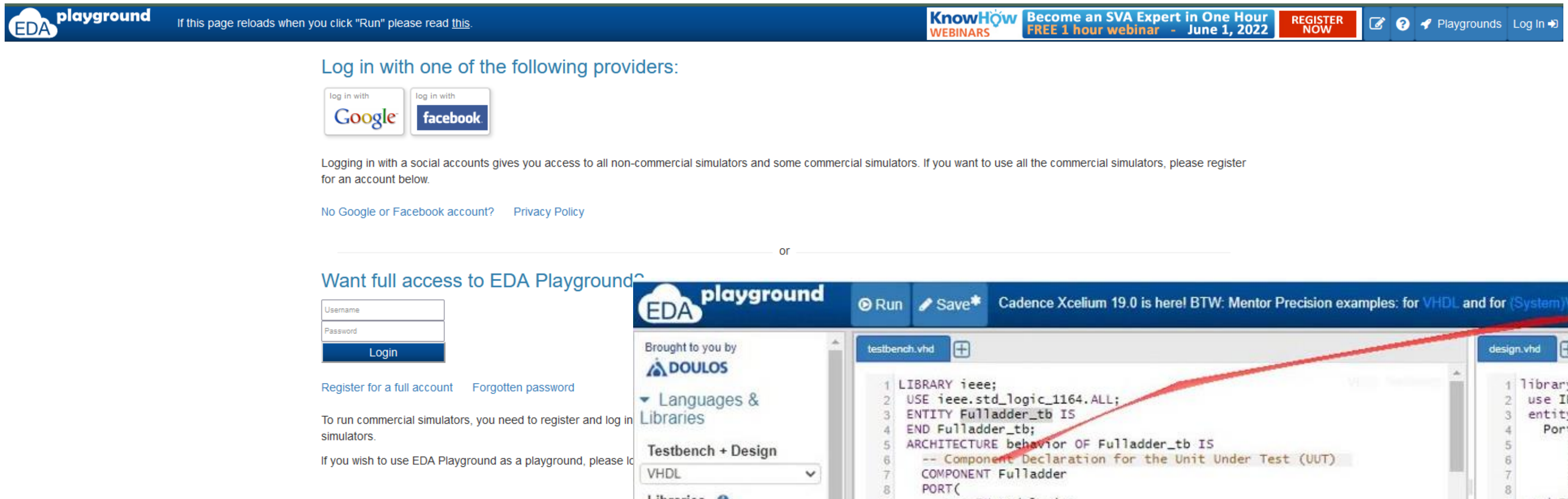


```
ARCHITECTURE AofC1  of C1 is
        SIGNAL s1: STD_LOGIC;
        SIGNAL s2: STD_LOGIC;
BEGIN
s1 <= not(InA) and InB;
s2 <= InA and not(InB);
Ot <= s1 or s2;
End AofC1;
```



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

VHDL online Editor, Compiler and Simulator


EDA playground
If this page reloads when you click "Run" please read this.
KnowHow WEBINARS
Become an SVA Expert in One Hour
FREE 1 hour webinar - June 1, 2022
REGISTER NOW
Playgrounds
Log In

Log in with one of the following providers:

log in with Google
log in with facebook

Logging in with a social accounts gives you access to all non-commercial simulators and some commercial simulators. If you want to use all the commercial simulators, please register for an account below.

No Google or Facebook account? Privacy Policy

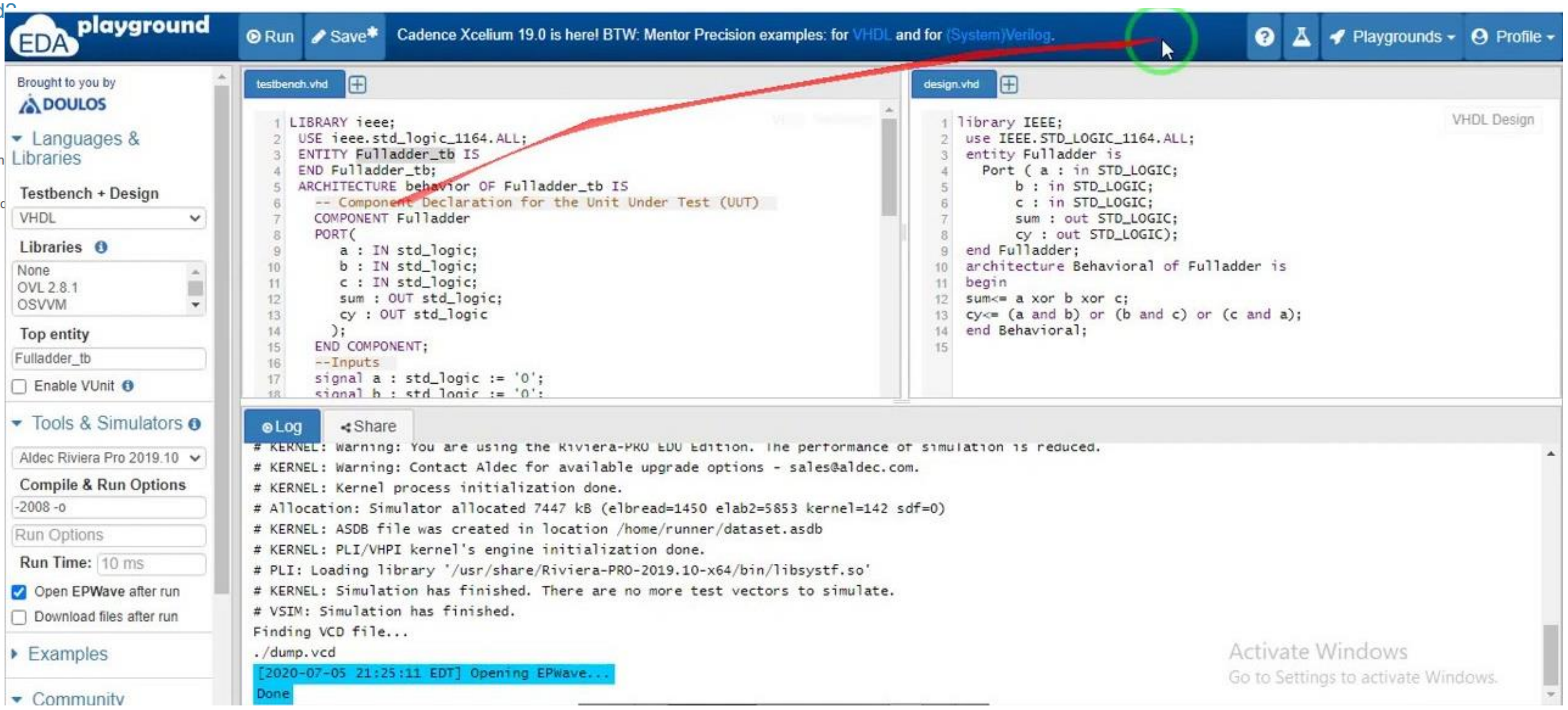
or

Want full access to EDA Playground?

Register for a full account Forgotten password

To run commercial simulators, you need to register and log in simulators.

If you wish to use EDA Playground as a playground, please lo



Testbench + Design: VHDL

Top entity: Fulladder_tb

Tools & Simulators: Aldec Riviera Pro 2019.10

Compile & Run Options: -2008 -o

Run Time: 10 ms

testbench.vhd

```vhdl
LIBRARY ieee;
 USE ieee.std_logic_1164.ALL;
 ENTITY Fulladder_tb IS
 END Fulladder_tb;
 ARCHITECTURE behavior OF Fulladder_tb IS
   -- Component Declaration for the Unit Under Test (UUT)
   COMPONENT Fulladder
   PORT(
      a : IN std_logic;
      b : IN std_logic;
      c : IN std_logic;
      sum : OUT std_logic;
      cy : OUT std_logic
     );
   END COMPONENT;
   --Inputs
   signal a : std_logic := '0';
   signal b : std_logic := '0';
```

design.vhd

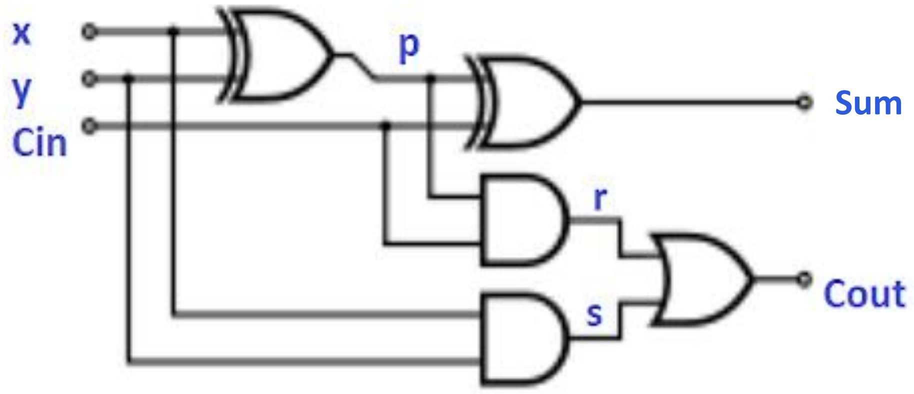
```vhdl
library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 entity Fulladder is
   Port ( a : in STD_LOGIC;
        b : in STD_LOGIC;
        c : in STD_LOGIC;
        sum : out STD_LOGIC;
        cy : out STD_LOGIC);
 end Fulladder;
 architecture Behavioral of Fulladder is
 begin
 sum<= a xor b xor c;
 cy<= (a and b) or (b and c) or (c and a);
 end Behavioral;
```

```
# KERNEL: Warning: You are using the Riviera-PRO EDU Edition. The performance of simulation is reduced.
# KERNEL: Warning: Contact Aldec for available upgrade options - sales@aldec.com.
# KERNEL: Kernel process initialization done.
# Allocation: Simulator allocated 7447 kB (elbread=1450 elab2=5853 kernel=142 sdf=0)
# KERNEL: ASDB file was created in location /home/runner/dataset.asdb
# KERNEL: PLI/VHPI kernel's engine initialization done.
# PLI: Loading library '/usr/share/Riviera-PRO-2019.10-x64/bin/libsystf.so'
# KERNEL: Simulation has finished. There are no more test vectors to simulate.
# VSIM: Simulation has finished.
Finding VCD file...
./dump.vcd
[2020-07-05 21:25:11 EDT] Opening EPWave...
Done
```

Task Group1

- Write a VHDL Code to Implement the below full-adder circuit using **data flow style.**

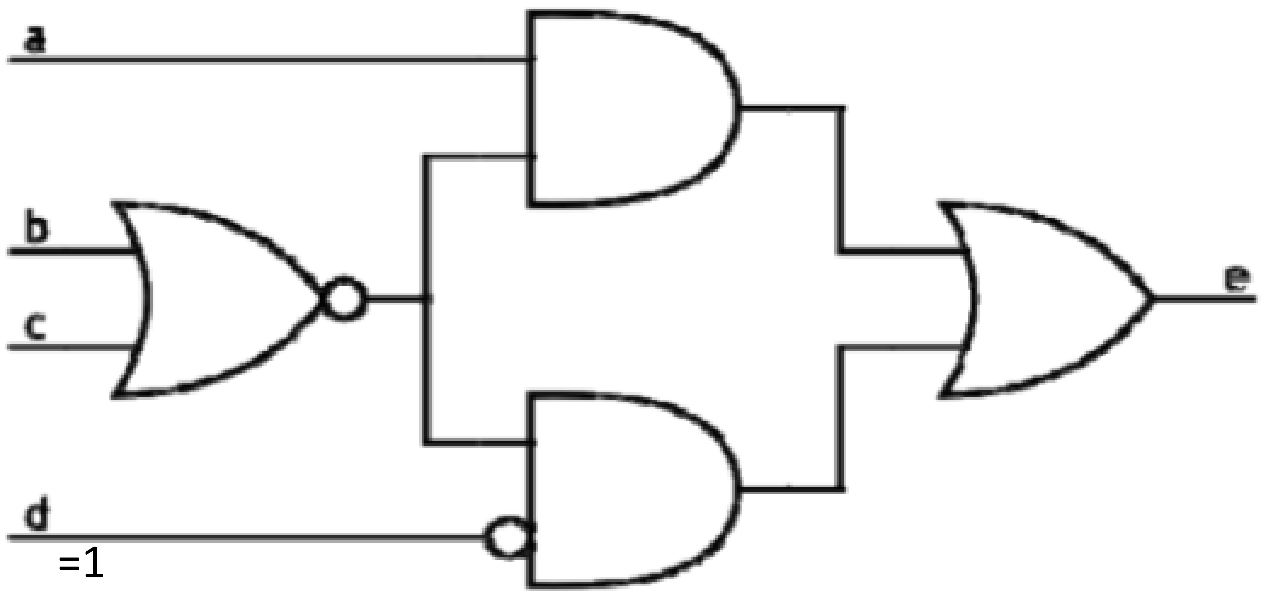
- Use **Signal** for the internal connection between gates.
- **Verify your design by simulation**, force different input patterns and check the output.



X	Y	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Task Group2

- Write a VHDL Code to Implement the below circuit function using **architecture data flow style.**
- Use **Signal** for the internal connection between gates.
- **Verify your design by simulation**, force different input patterns and check the output.



A	B	C	D	E
0	0	0	1	1
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	1	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0