Budapest University of Technology and Economics
Department of Electron Devices

Introduction to VHDL language structure, examples

Digital Lab.1

Osama Ali

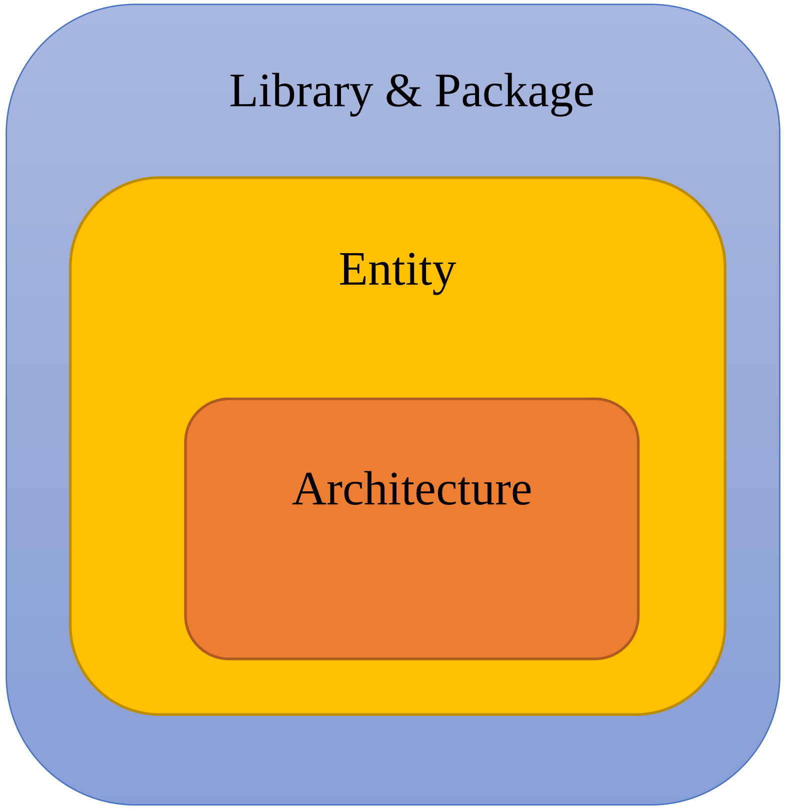
2024

Introduction to VHDL

- Higher-level computer languages are used to describe algorithms
  - Sequential execution
- Hardware Description Languages (HDL) are used to describe hardware
  - Not for programming, but for designing hardware
  - Most popular: VHDL, Verilog
  - Parallel (concurrent) execution
    - Instructions are all executed at the same time

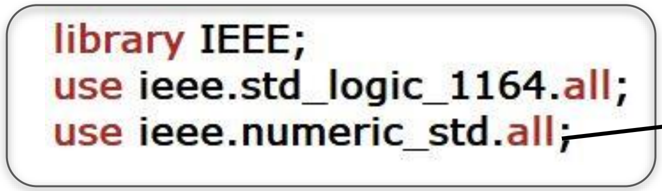
A VHDL design consist of three fundamental design units

1. **Library and Package Declaration**
2. **Entity Declaration**
3. **Architecture**



1. Library and Package Declaration

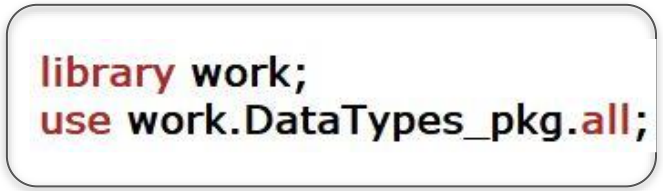
- Library and packages are collection of commonly used items, such as data **types**, **subprograms**, and **components**.



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

Defines arithmetic operations on arrays of STD_LOGIC

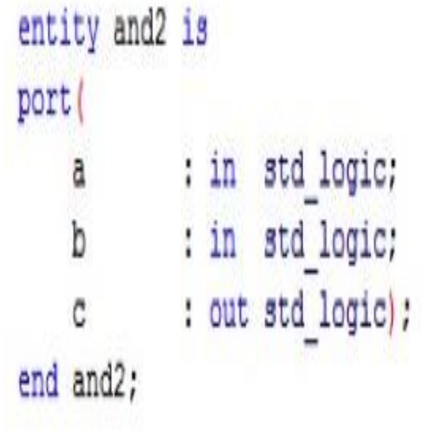
- One can create package that includes several data types, constants, and components. After the IEEE library we can declare the new package.



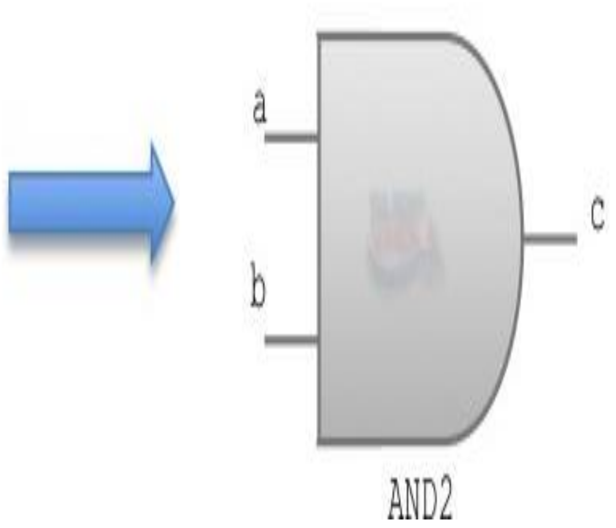
```
library work;
use work.DataTypes_pkg.all;
```

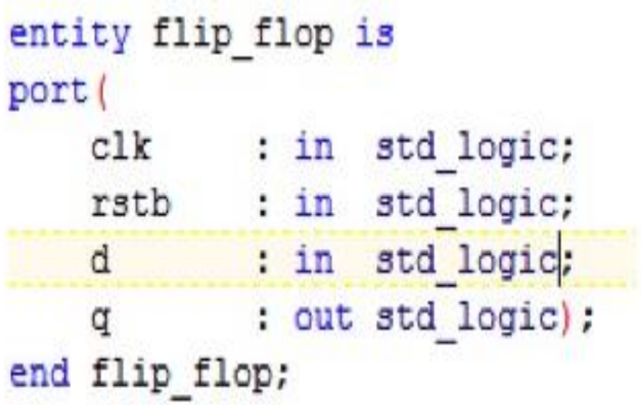
2. Entity Declaration

- **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**
- **Data_Type**: define the data type of the port which can be **bit**, **integer**, **std_logic**, and many other types

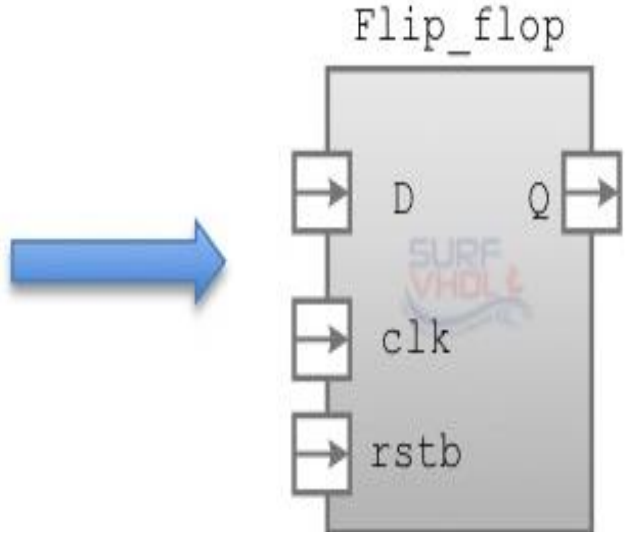


```
entity and2 is
port(
    a      : in  std_logic;
    b      : in  std_logic;
    c      : out std_logic);
end and2;
```



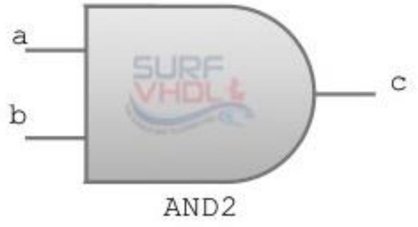


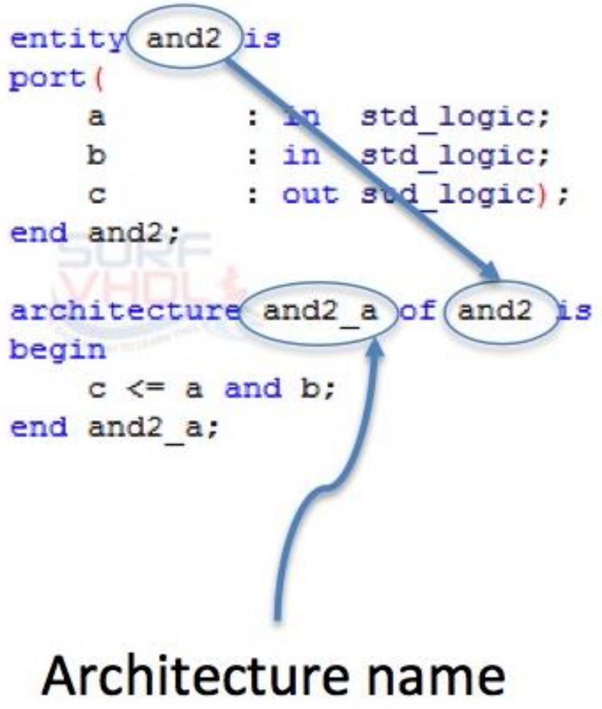
```
entity flip_flop is
port(
    clk      : in  std_logic;
    rstb     : in  std_logic;
    d        : in  std_logic;
    q        : out std_logic);
end flip_flop;
```



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





```
entity and2 is
port(
    a        : in   std_logic;
    b        : in   std_logic;
    c        : out  std_logic);
end and2;

architecture and2_a of and2 is
begin
    c <= a and b;
end and2_a;
```

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

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 (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).

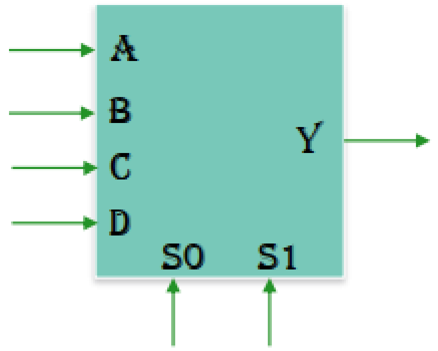
```
architecture sim of UnresolvedTb is
    signal Sig1 : std_ulogic := '0';
begin

    -- Driver A
    Sig1 <= '0';

    -- Driver B
    Sig1 <= '1' after 20 ns;

end architecture;
```

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

Combinational statements - using operators ***Supplementary***

Operators can be used to implement any combinational circuit.

```
y <= (a AND NOT s1 AND NOT s0) OR
     (b AND NOT s1 AND s0) OR
     (c AND s1 AND NOT s0) OR
     (d AND s1 AND s0);
```



Combinational statements - using WHEN

Supplementary

- WHEN is another fundamental concurrent statements.
- It appears in two forms: WHEN / ELSE (simple WHEN) and WITH / SELECT / (selected WHEN)

WHEN / ELSE:

```
ASSIGNMENT WHEN CONDITION ELSE
ASSIGNMENT WHEN CONDITION ELSE
...... ;
```

WITH / SELECT / WHEN:

```
WITH IDENTIFIER SELECT
     ASSIGNMENT WHEN VALUE,
     ...;
```

Example 5.10: PHYSICA data type

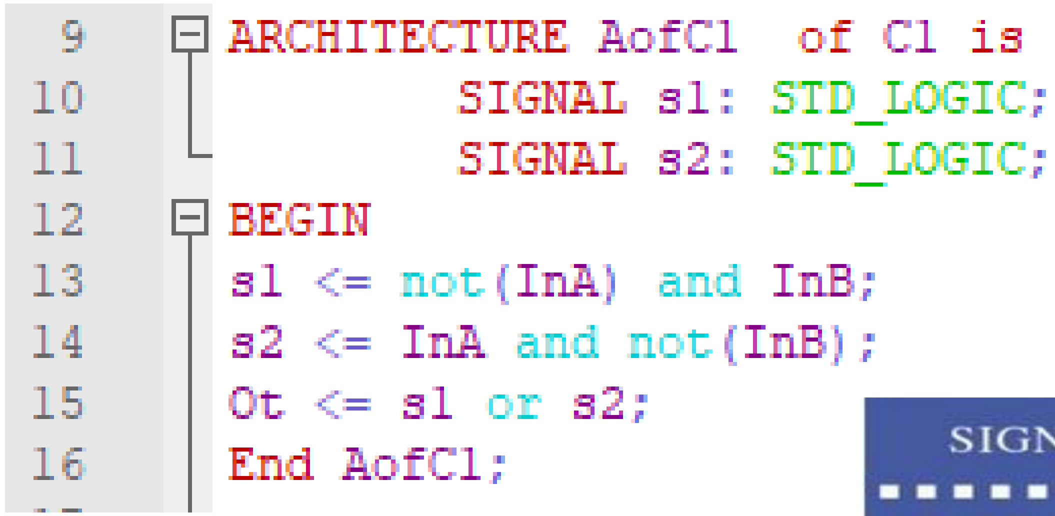
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY mux IS
PORT ( a, b, c, d: IN STD_LOGIC;
        sel: IN STD_LOGIC_VECTOR (1 DOWNTO 0);
        y  : OUT STD_LOGIC);
END mux;
ARCHITECTURE mux1 OF mux IS
BEGIN
y <= a WHEN sel="00" ELSE
     b WHEN sel="01" ELSE
     c WHEN sel="10" ELSE
     d;
END mux1;
```

Example 5.10: PHYSICA data type

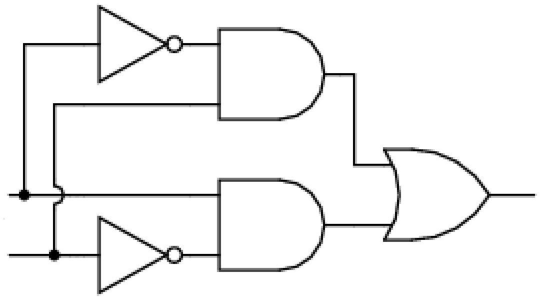
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY mux IS
PORT ( a, b, c, d: IN STD_LOGIC;
       sel: IN STD_LOGIC_VECTOR (1 DOWNTO 0);
       y: OUT STD_LOGIC);
END mux;
ARCHITECTURE mux2 OF mux IS
BEGIN
   WITH sel SELECT
    y <= a WHEN "00",
         b WHEN "01",
         c WHEN "10",
         d WHEN OTHERS; --
END mux2;
```

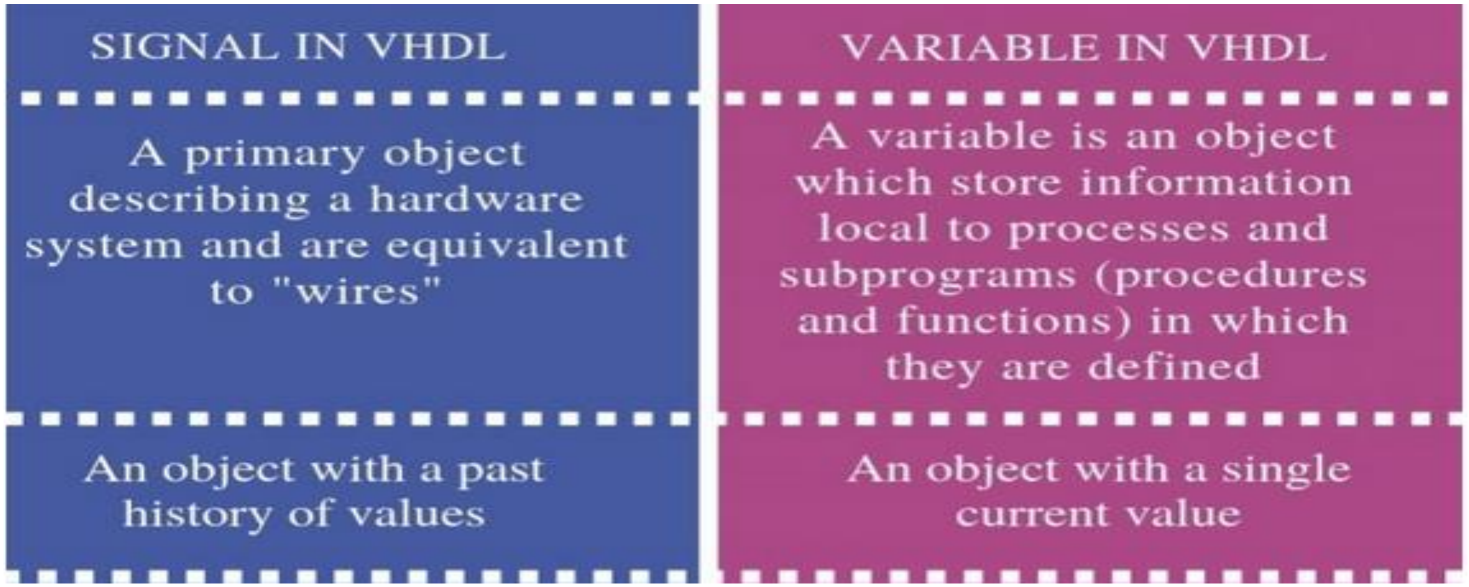
Signals and Variable

- Represent wires within the circuit.



```
ARCHITECTURE AofCl  of Cl 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 AofCl;
```

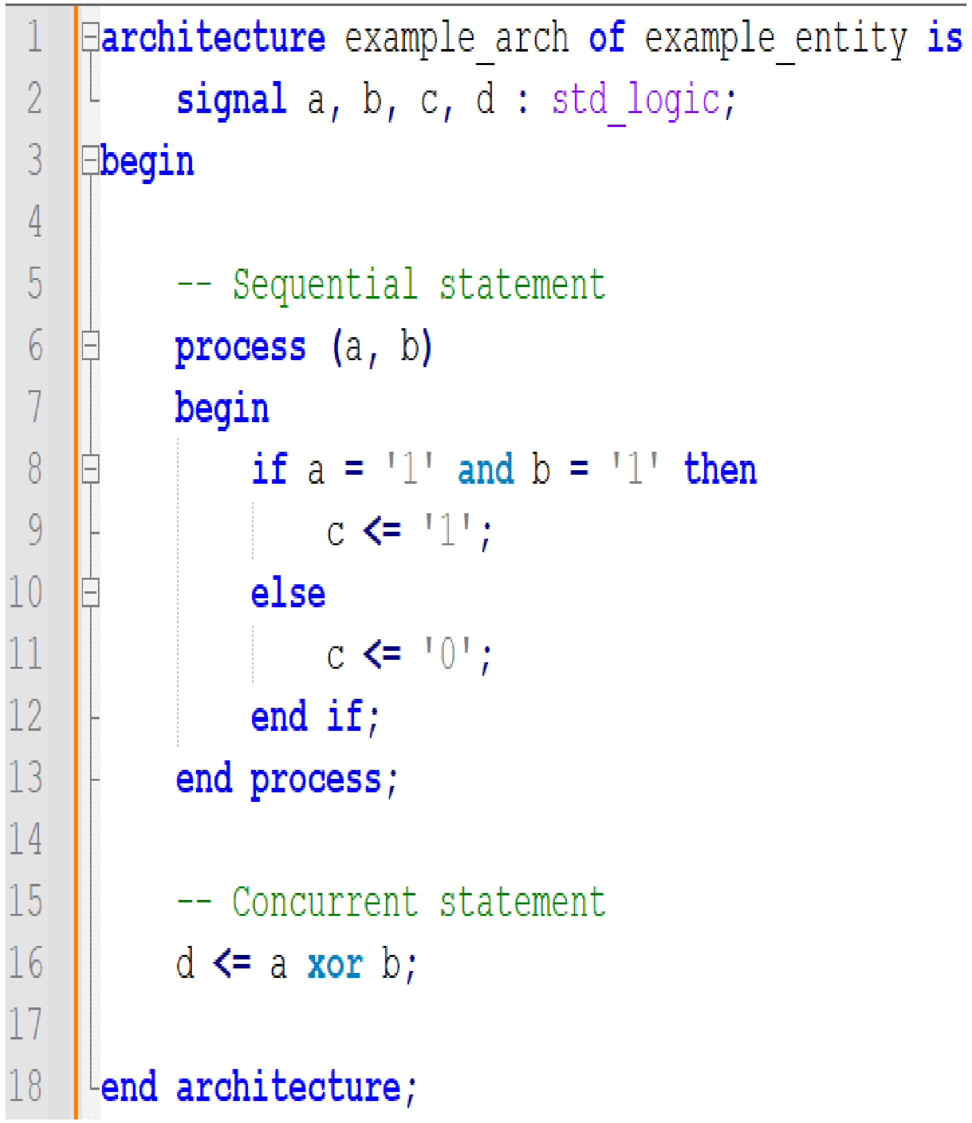




SIGNAL IN VHDL	VARIABLE IN VHDL
A primary object describing a hardware system and are equivalent to "wires"	A variable is an object which store information local to processes and subprograms (procedures and functions) in which they are defined
An object with a past history of values	An object with a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

Sequential statements (Process) *Supplementary*

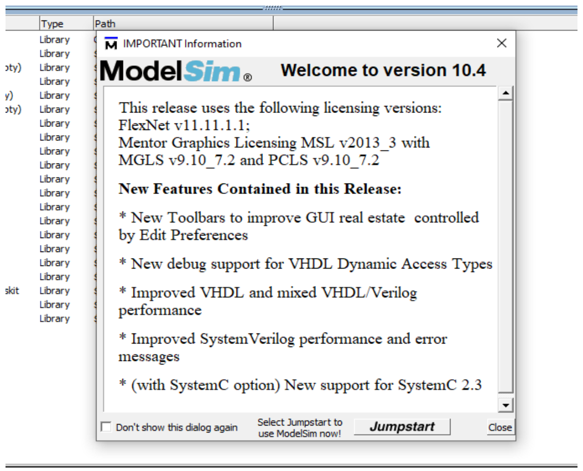
- A *process* statement itself is a **concurrent statement**
  - All statements in a process are executed sequentially until the process is suspended via a *wait* statement
  - Within a process, procedures and functions can partition the sequential statements
- A process can be a single signal assignment statement or a series of sequential statements
- Upon initialization, all processes are executed once
- Processes are executed in a data-driven manner, and activated
  - by events on signals in the process *sensitivity* list or
  - by waiting for the occurrence of specific events using the *wait* statement
- The **sensitivity list** – being next to the process keyword – is a list of those input signals to the component to which the process is sensitive



```
architecture example_arch of example_entity is
    signal a, b, c, d : std_logic;
begin

    -- Sequential statement
    process (a, b)
    begin
        if a = '1' and b = '1' then
            c <= '1';
        else
            c <= '0';
        end if;
    end process;

    -- Concurrent statement
    d <= a xor b;

end architecture;
```

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.



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

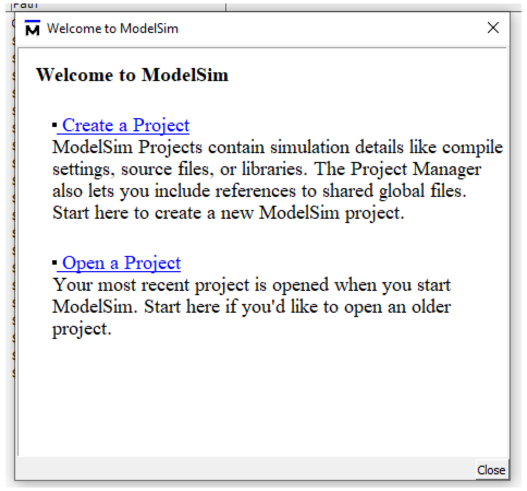
* New debug support for VHDL Dynamic Access Types

* Improved VHDL and mixed VHDL/Verilog performance

* Improved SystemVerilog performance and error messages

* (with SystemC option) New support for SystemC 2.3

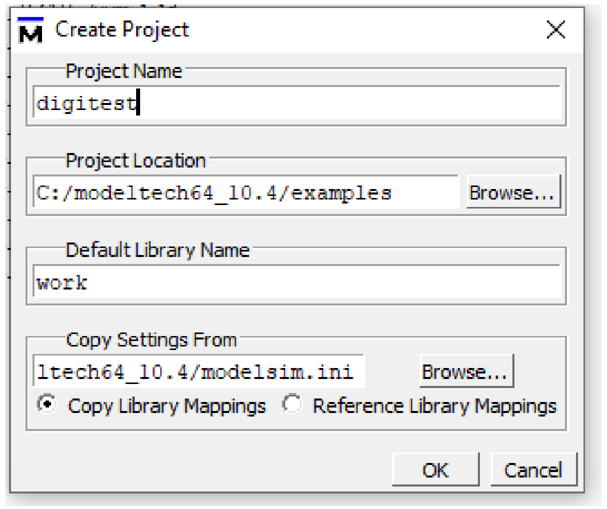
Don't show this dialog again
Select Jumpstart to use ModelSim now!
Jumpstart
Close



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.



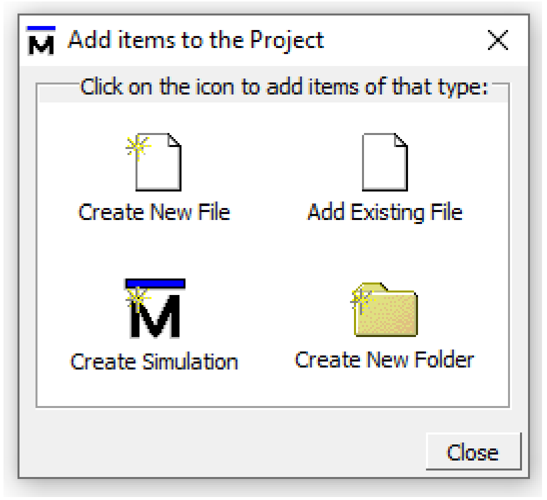
Project Name: digitest

Project Location: C:/modeltech64_10.4/examples

Default Library Name: work

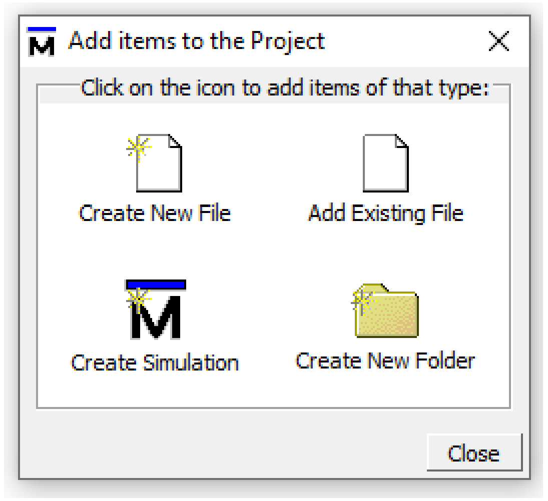
Copy Settings From: ltech64_10.4/modelsim.ini

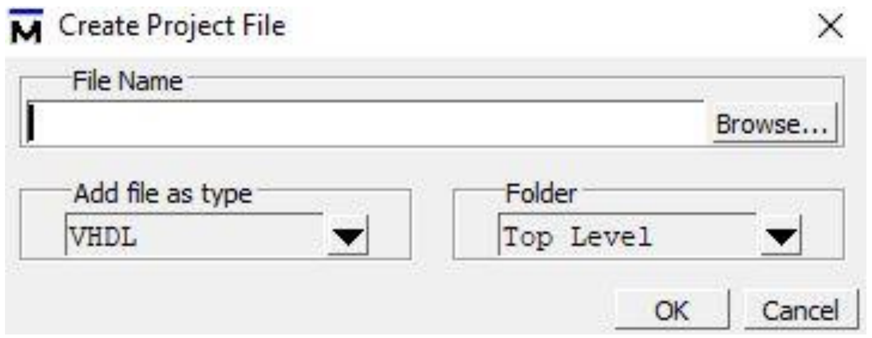
[✓ Copy Library Mappings] Reference Library Mappings



Adding the source code

- 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.

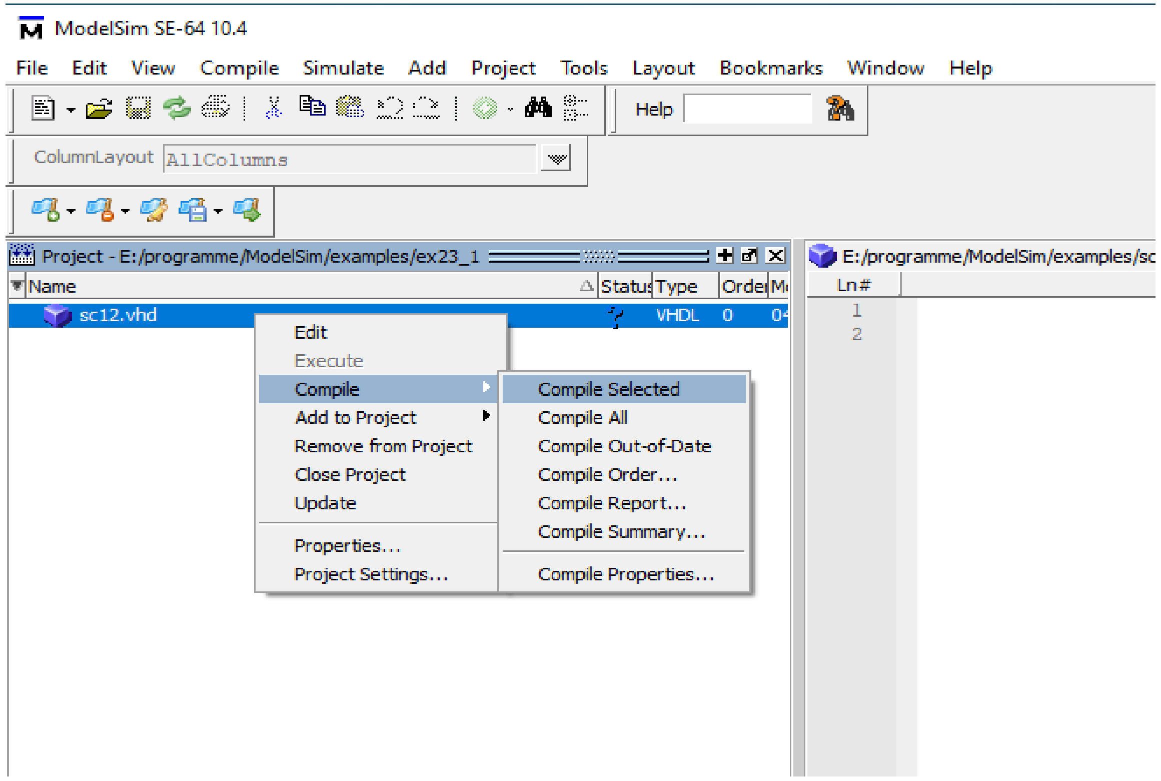




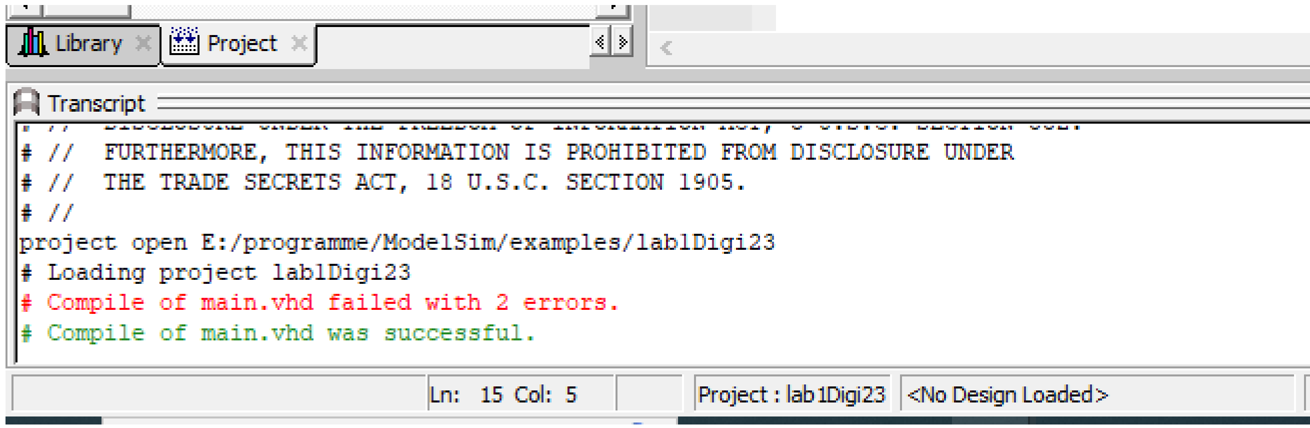
- If there are no more files, we can close the current window and start editing the source code.

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 Notepad++
- After finish editing the code, we need to compile it to make



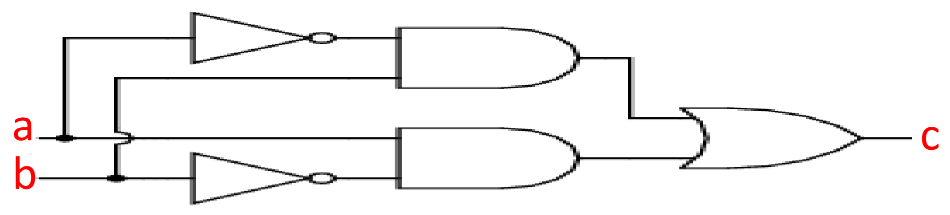
- After the first try for compilation, it is probable to get an error message related to some typos. To correct these error, double click on the red message appears in the Transcript sub-window and read the error description, then edit your code.
- After correcting all errors, you should see a message in green that states (Compile of source file was successful).



```
# //  FURTHERMORE, THIS INFORMATION IS PROHIBITED FROM DISCLOSURE UNDER
# //  THE TRADE SECRETS ACT, 18 U.S.C. SECTION 1905.
# //
project open E:/programme/ModelSim/examples/lab1Digi23
# Loading project lab1Digi23
# Compile of main.vhd failed with 2 errors.
# Compile of main.vhd was successful.
```

Write this code also Test and Verify it

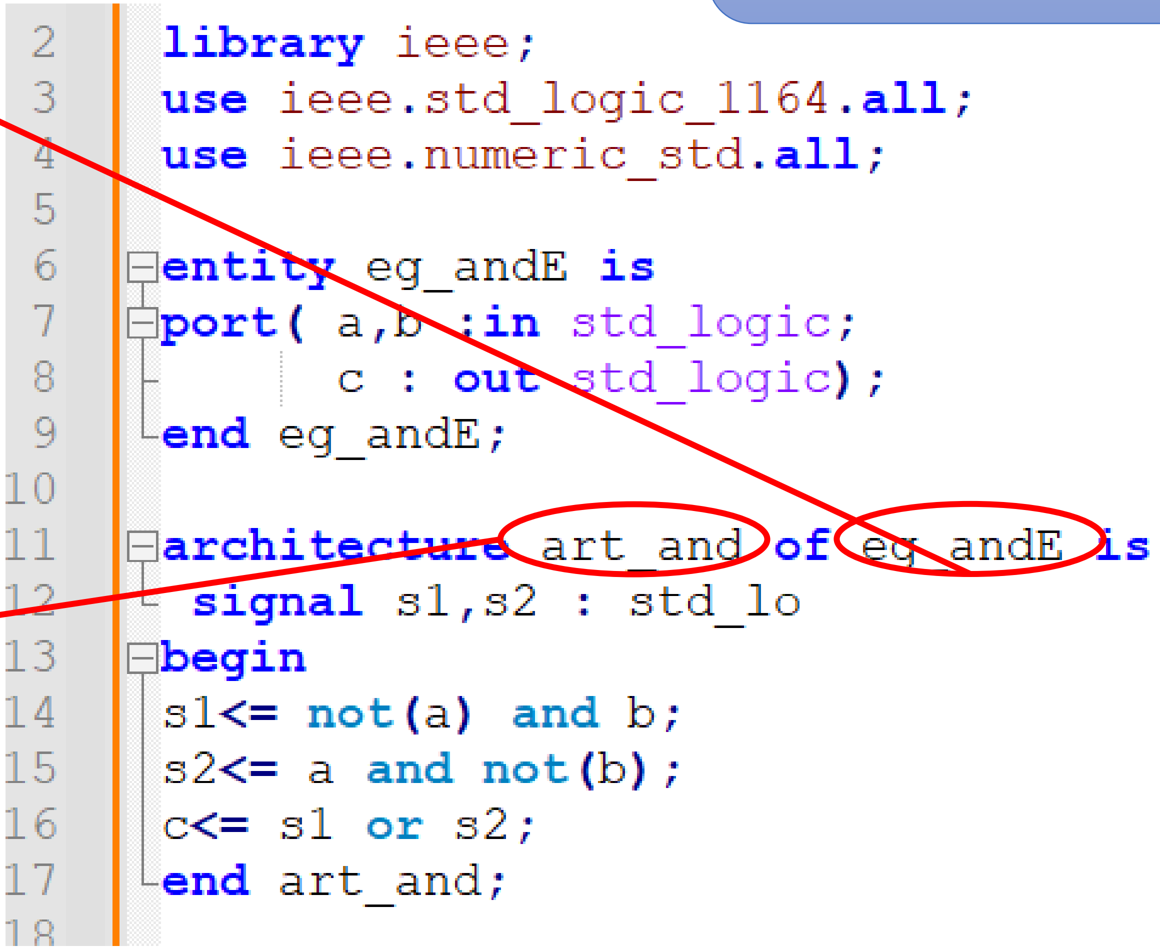
Entity name



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

Architecture name

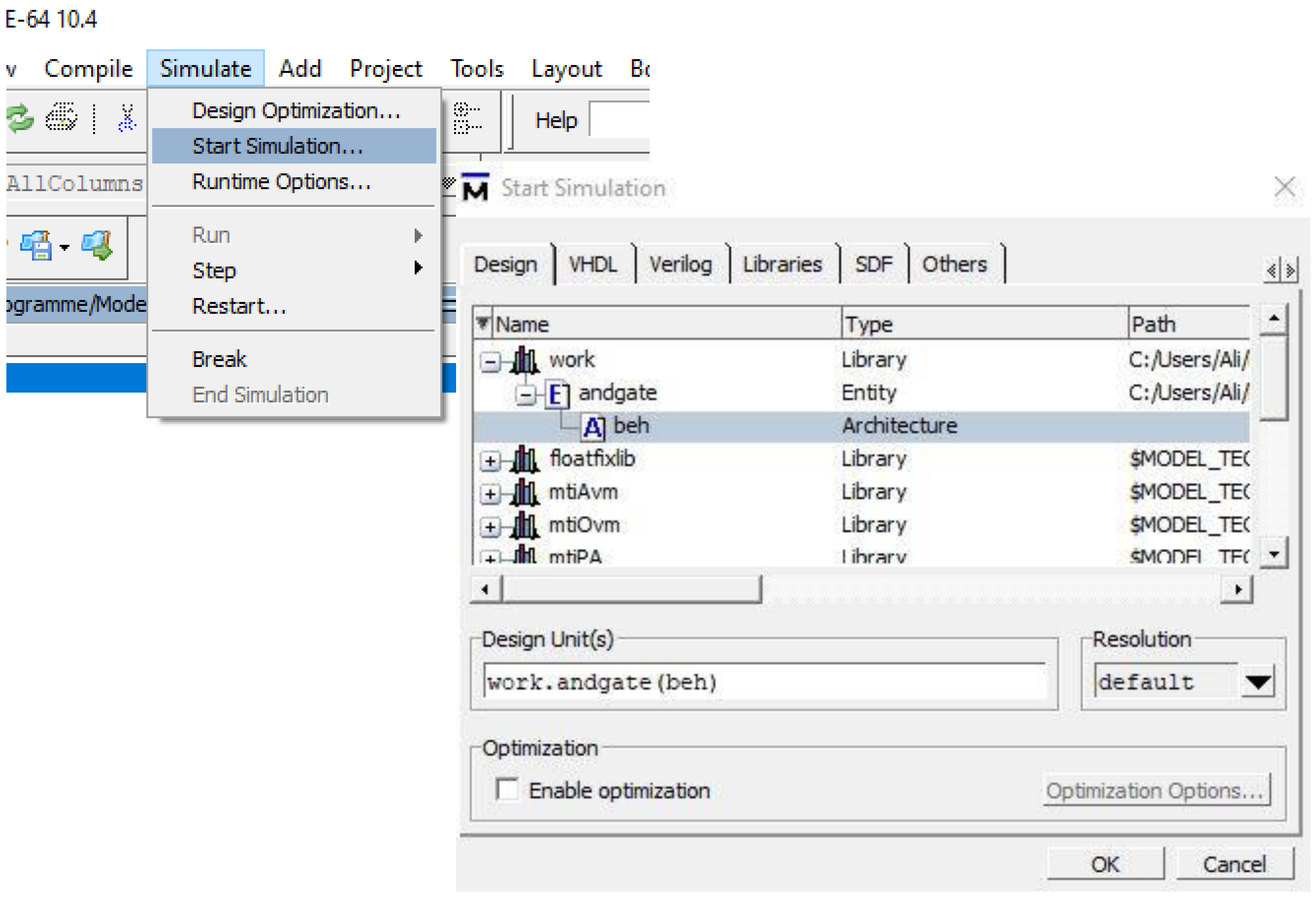
Library & Package
Entity



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

entity eg_andE is
port( a,b  :in std_logic;
        c  : out std_logic);
end eg_andE;

architecture art_and of eg_andE is
 signal s1,s2 : std_lo
begin
s1<= not(a) and b;
s2<= a and not(b);
c<= s1 or s2;
end art_and;
```

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**.

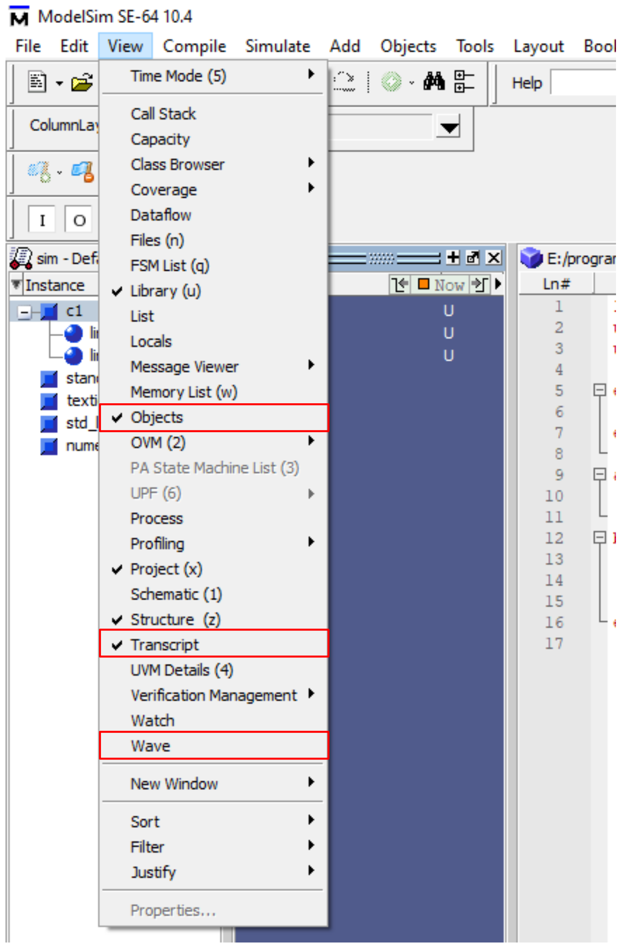


Name	Type	Path
work	Library	C:/Users/Ali/
andgate	Entity	C:/Users/Ali/
beh	Architecture	
floatfixlib	Library	$MODEL_TEC
mtiAvm	Library	$MODEL_TEC
mtiOvm	Library	$MODEL_TEC
mtiPA	Library	$MODEL_TEC

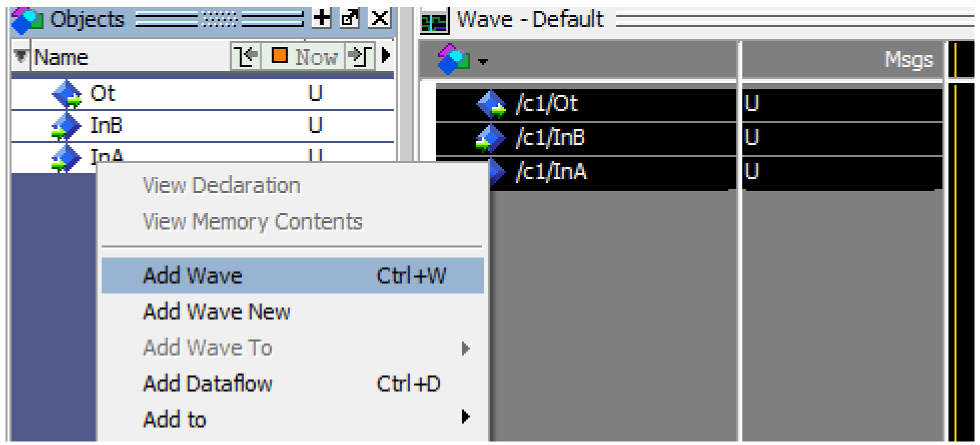
Design Unit(s): work.andgate(beh)

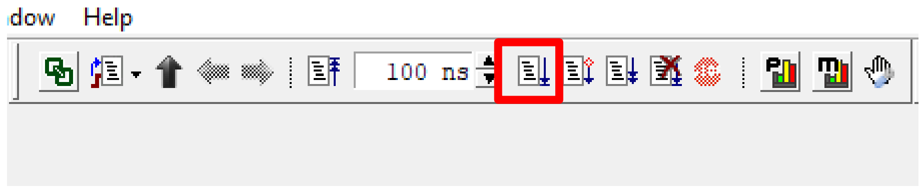
Resolution: default

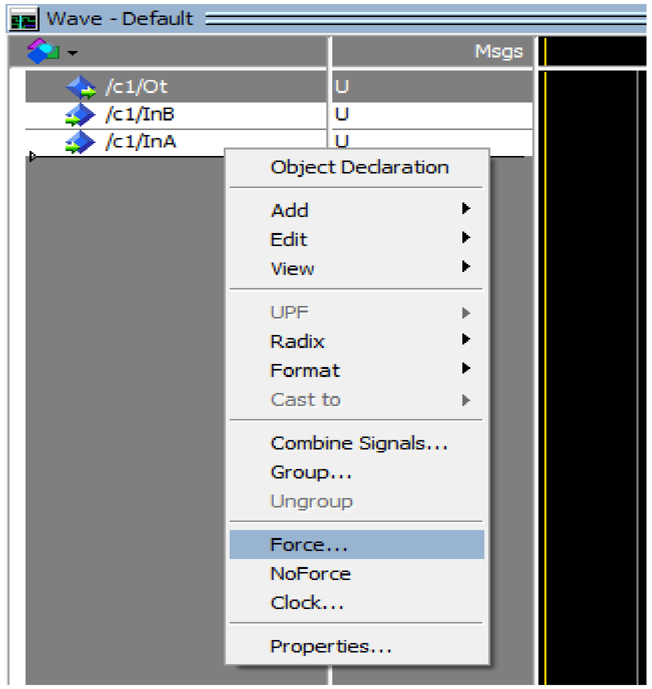
[ ] Enable optimization

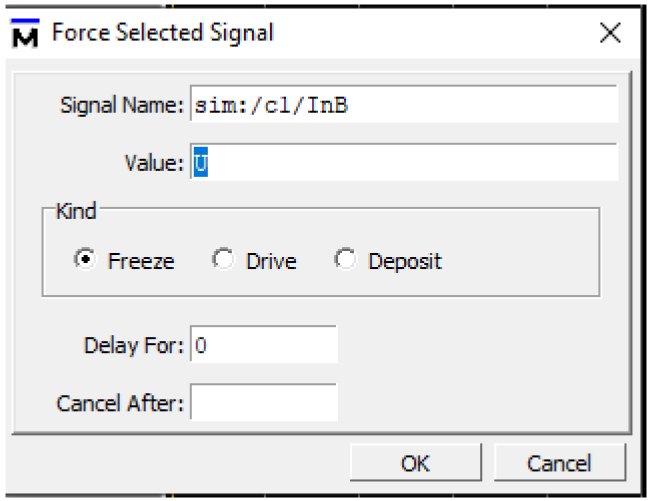


Add to Wave & Force

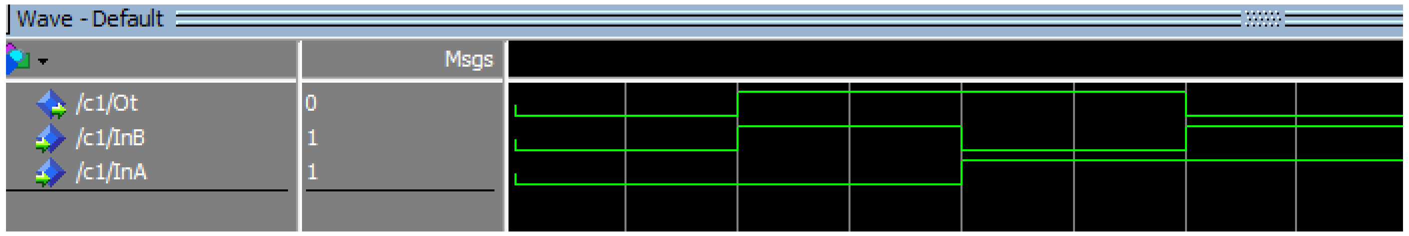








Signal Name: sim:/c1/InB
Value: U
Kind: [✓ Freeze] Drive Deposit
Delay For: 0
Cancel After:

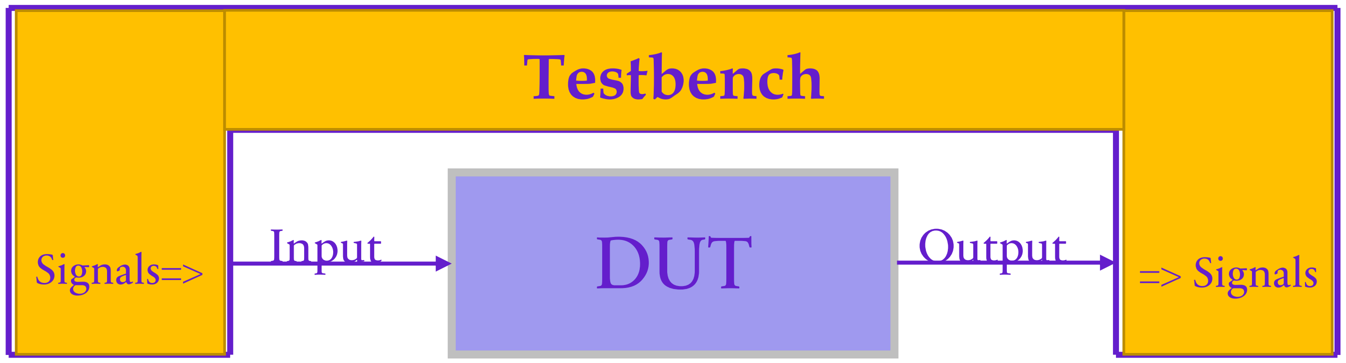


Testbench

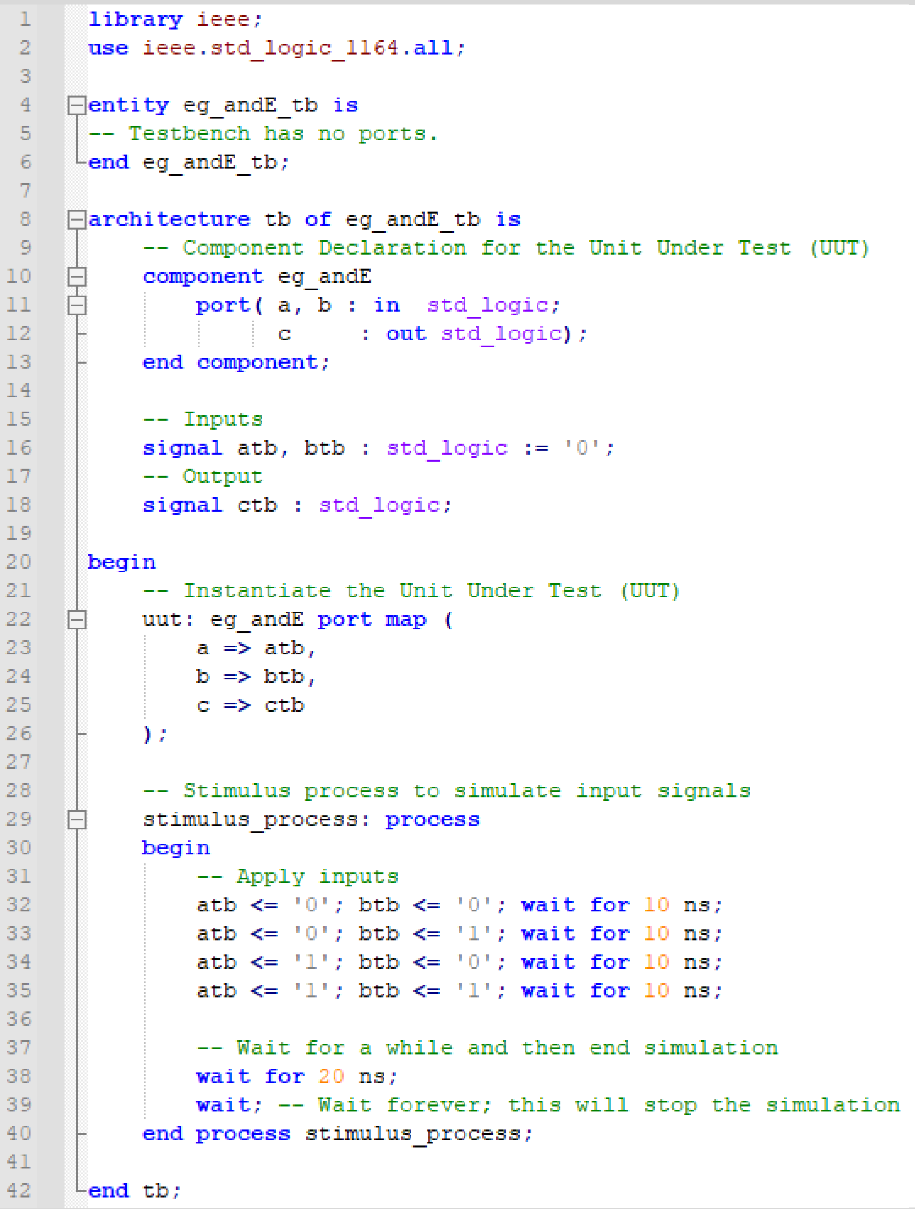
- Testbench is an important part of VHDL design to check the functionality of Design through simulation waveform.
- Testbench provides a stimulus for **design under test** DUT or **Unit Under Test** UUT to check the output result.

A TestBench consists of:

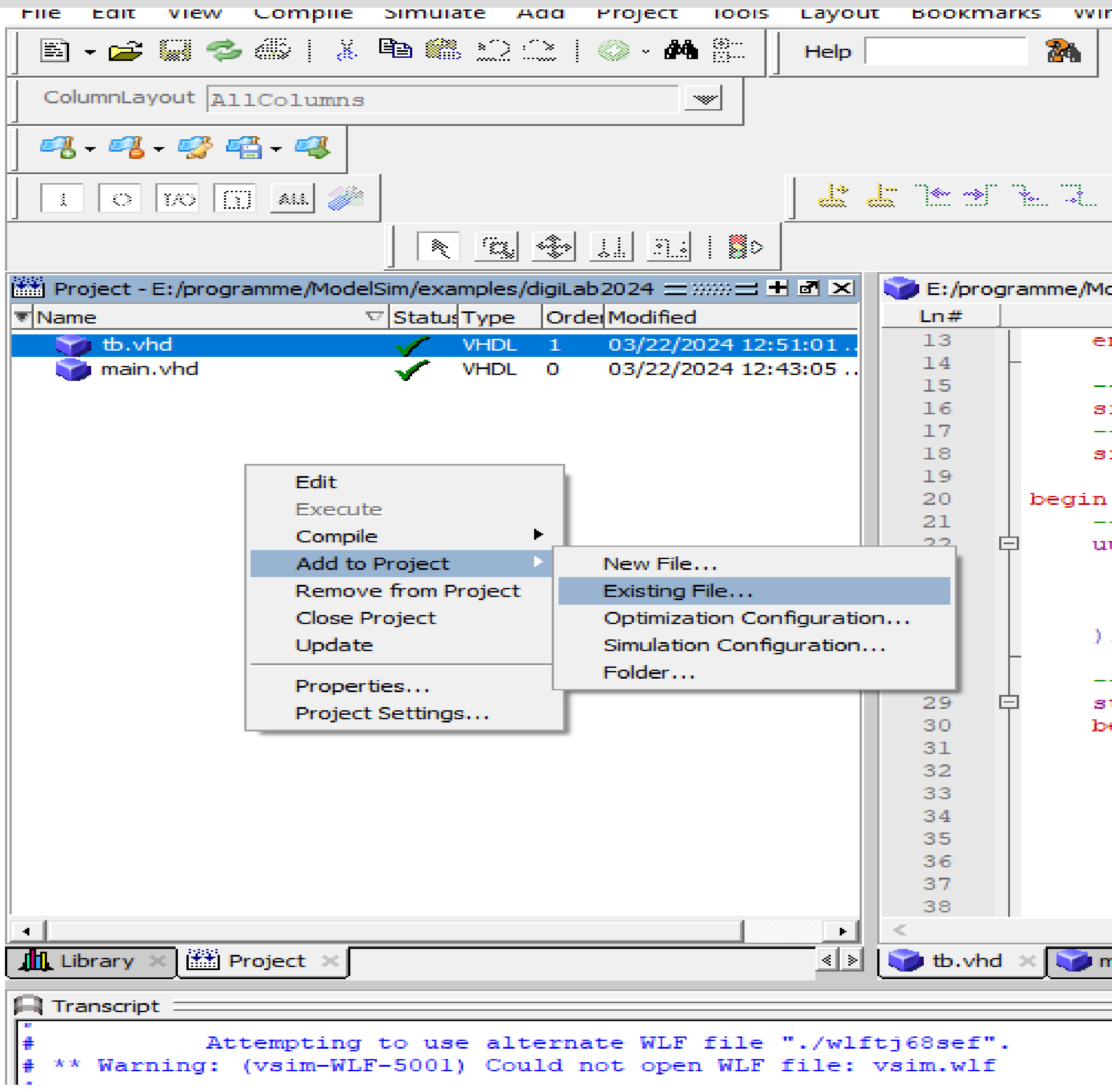
- Entity
  - has no ports (empty entity header)
- Architecture
  - declares, instantiates, and wires together the driver model and the model under test
  - driver model provides the stimulus and verifies model responses



Testbench File



```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity eg_andE_tb is
-- Testbench has no ports.
end eg_andE_tb;

architecture tb of eg_andE_tb is
    -- Component Declaration for the Unit Under Test (UUT)
    component eg_andE
        port( a, b : in  std_logic;
              c     : out std_logic);
    end component;

    -- Inputs
    signal atb, btb : std_logic := '0';
    -- Output
    signal ctb : std_logic;

begin
    -- Instantiate the Unit Under Test (UUT)
    uut: eg_andE port map (
        a => atb,
        b => btb,
        c => ctb
    );

    -- Stimulus process to simulate input signals
    stimulus_process: process
    begin
        -- Apply inputs
        atb <= '0'; btb <= '0'; wait for 10 ns;
        atb <= '0'; btb <= '1'; wait for 10 ns;
        atb <= '1'; btb <= '0'; wait for 10 ns;
        atb <= '1'; btb <= '1'; wait for 10 ns;

        -- Wait for a while and then end simulation
        wait for 20 ns;
        wait; -- Wait forever; this will stop the simulation
    end process stimulus_process;

end tb;
```

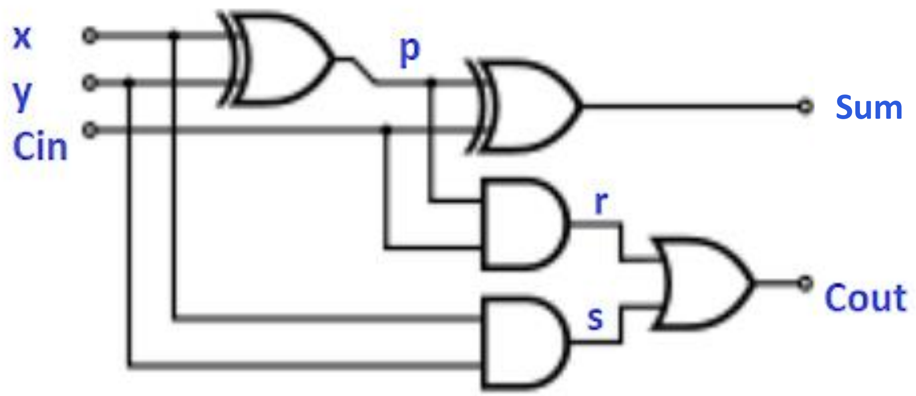


Name	Status	Type	Order	Modified
tb.vhd	✓	VHDL	1	03/22/2024 12:51:01 ..
main.vhd	✓	VHDL	0	03/22/2024 12:43:05 ..

```
#          Attempting to use alternate WLF file "./wlftj68sef".
# ** Warning: (vsim-WLF-5001) Could not open WLF file: vsim.wlf
```

Task

- 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	C_{in}	Sum	C_{out}
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