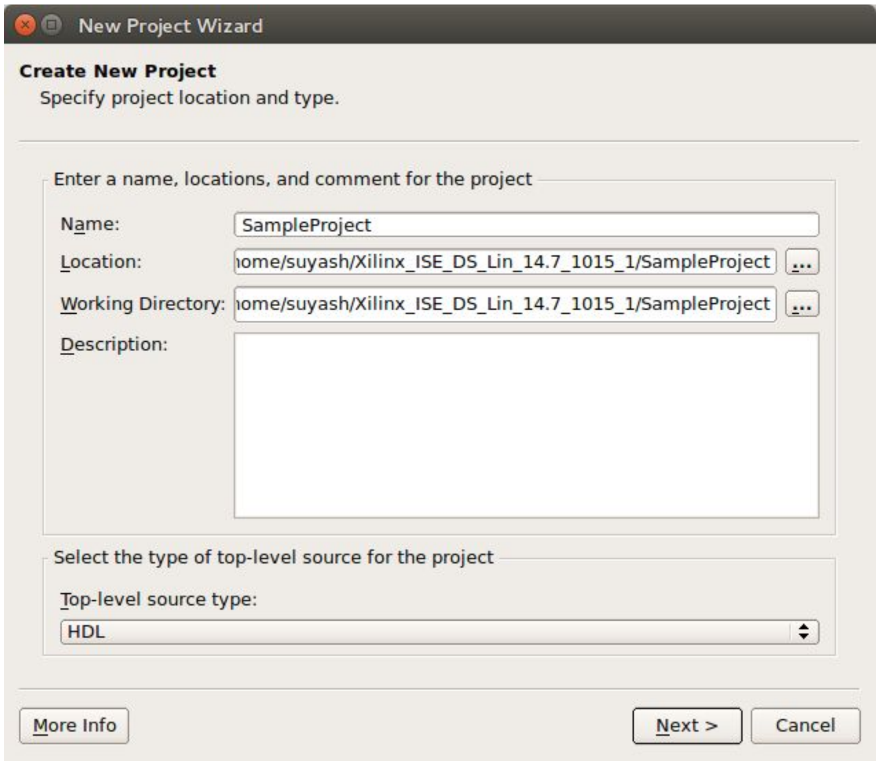
Getting Started with Xilinx ISE

- Start the Xilinx ISE
- Create a new Project by clicking on 'New Project' (on the top left window panel)



New Project Wizard

Create New Project

Specify project location and type.

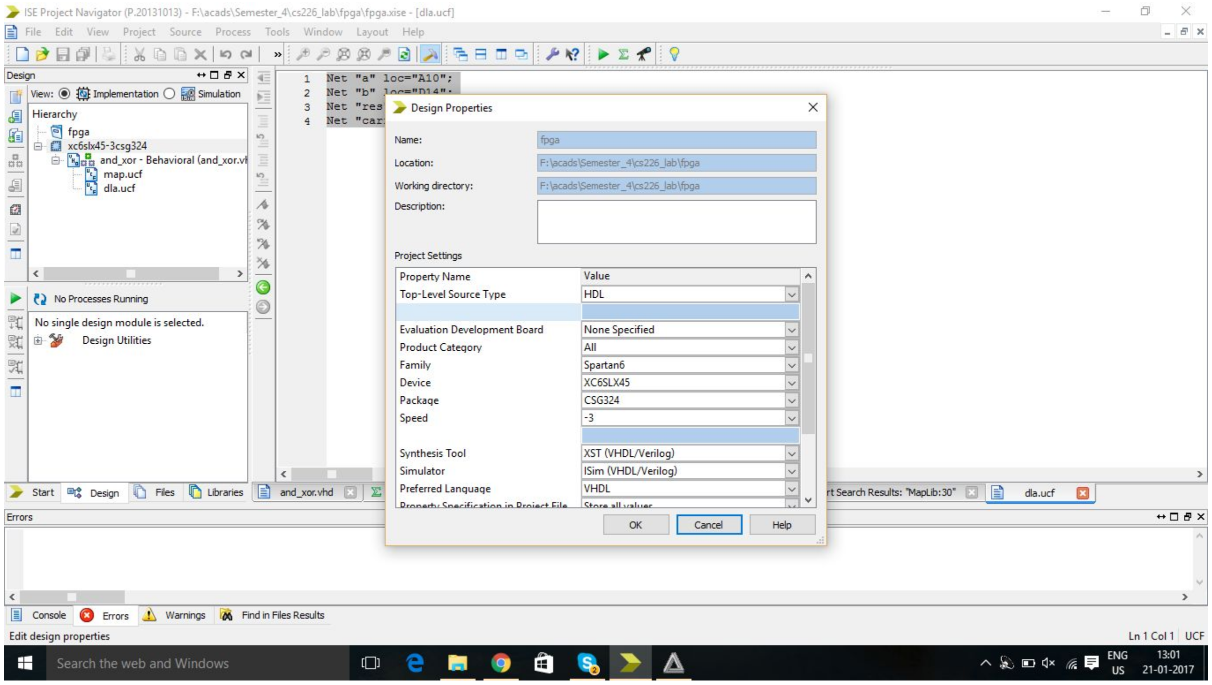
Enter a name, locations, and comment for the project

Name:	SampleProject
Location:	ome/suyash/Xilinx_ISE_DS_Lin_14.7_1015_1/SampleProject
Working Directory:	ome/suyash/Xilinx_ISE_DS_Lin_14.7_1015_1/SampleProject
Description:	

Select the type of top-level source for the project

Top-level source type: HDL

- Enter the project details as shown here



ISE Project Navigator (P.20131013) - F:\acads\Semester_4\cs226_lab\fpga\fpga.xise - [dla.ucf]

```
Net "a" loc="A10";
Net "b" loc="D14";
Net "res
Net "car
```

Hierarchy: fpga / xc6slx45-3csg324 / and_xor - Behavioral (and_xor.vh / map.ucf / dla.ucf

No Processes Running

No single design module is selected.

Design Properties

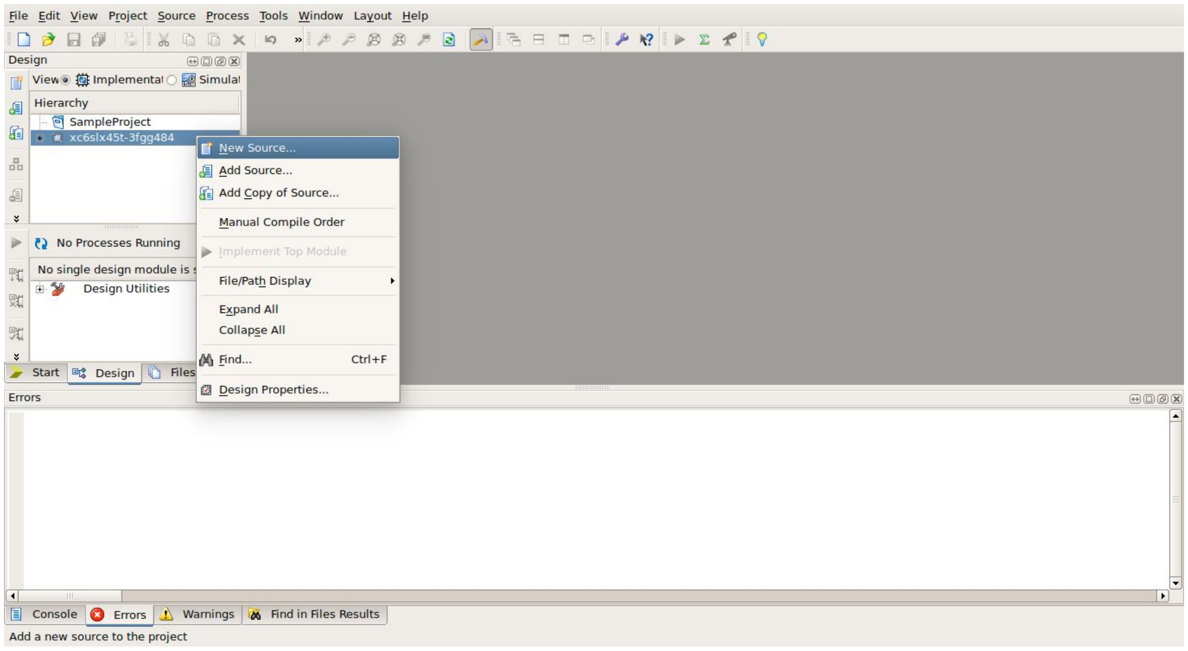
Name:	fpga
Location:	F:\acads\Semester_4\cs226_lab\fpga
Working directory:	F:\acads\Semester_4\cs226_lab\fpga
Description:	

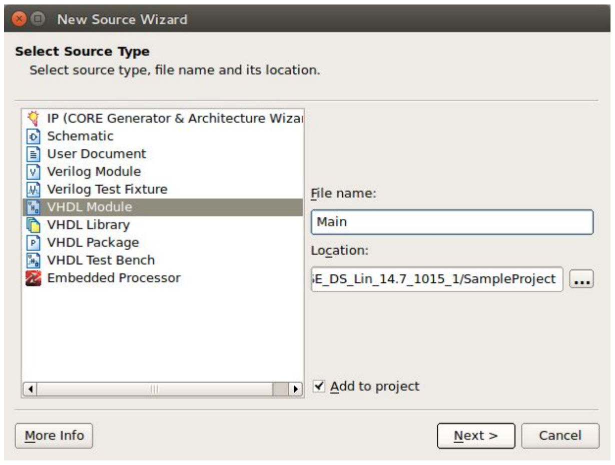
Project Settings

Property Name	Value
Top-Level Source Type	HDL
Evaluation Development Board	None Specified
Product Category	All
Family	Spartan6
Device	XC6SLX45
Package	CSG324
Speed	-3
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISim (VHDL/Verilog)
Preferred Language	VHDL
Property Specification in Project File	Store all values

And thus finish the creation of the project.

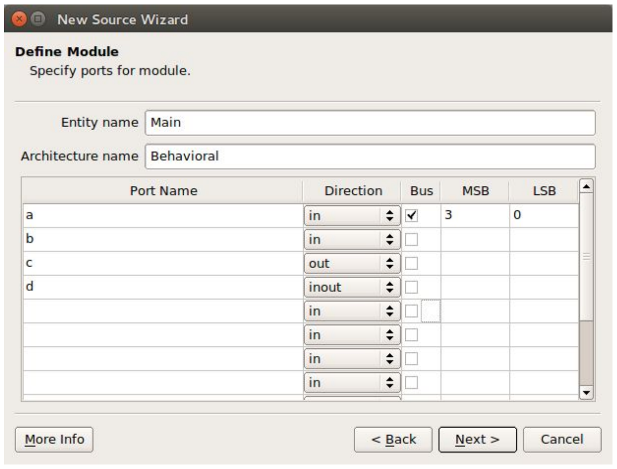
- Add a new source, VHDL module to your project and enter details of the input/output variables. Also note how a vector can be declared here itself.





Select Source Type
Select source type, file name and its location.

File name: Main
Location: E_DS_Lin_14.7_1015_1/SampleProject
✓ Add to project



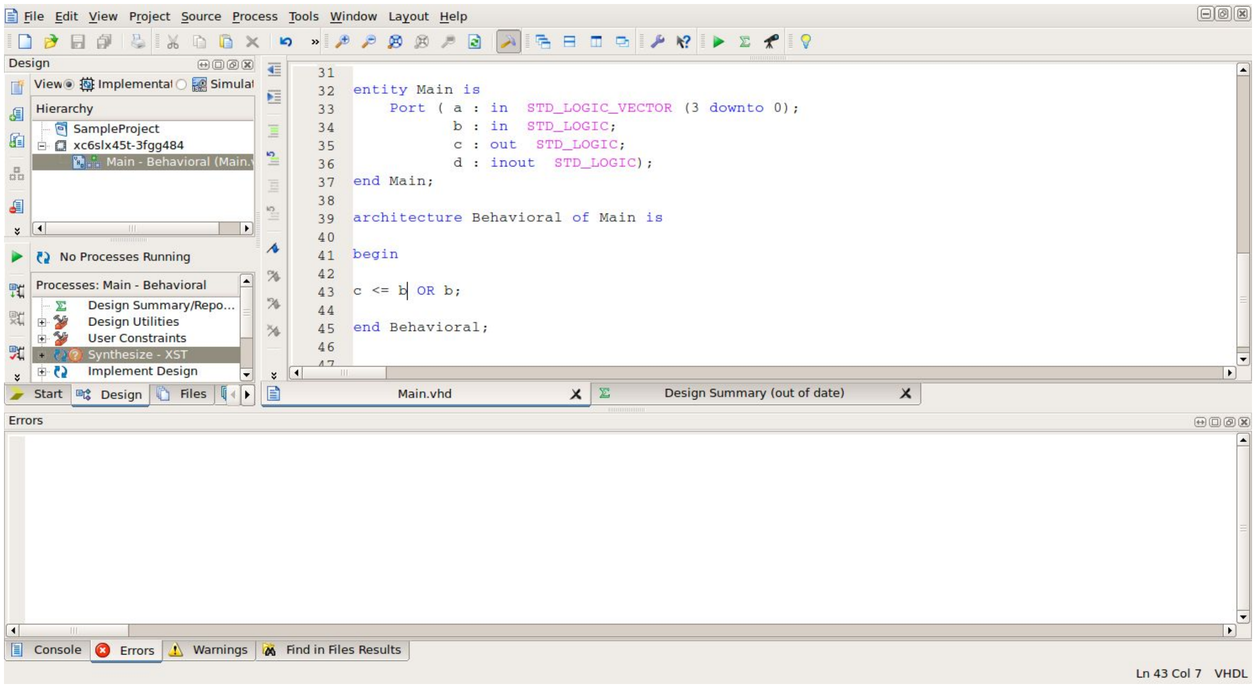
Define Module
Specify ports for module.

Entity name: Main
Architecture name: Behavioral

Port Name	Direction	Bus	MSB	LSB
a	in	✓	3	0
b	in			
c	out			
d	inout			
	in			
	in			
	in			
	in			

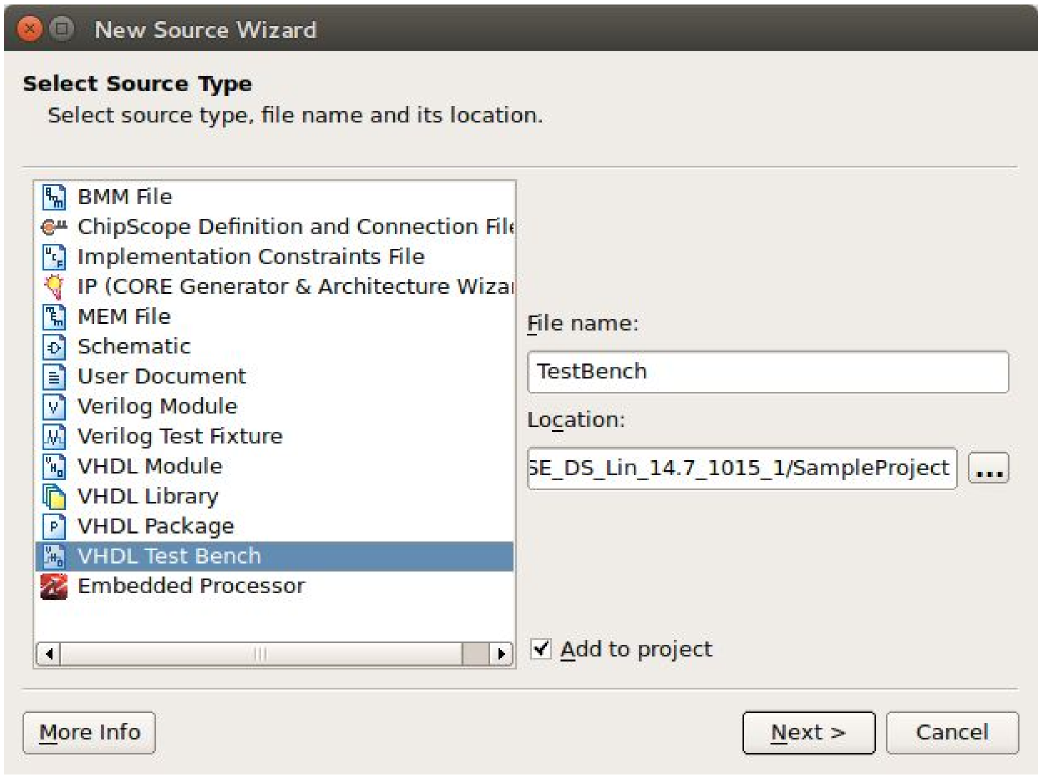
Thus finish the creation of this module.

- Then you can write the code you need in the architectural section as shown below:



```vhdl
entity Main is
    Port ( a : in  STD_LOGIC_VECTOR (3 downto 0);
           b : in  STD_LOGIC;
           c : out  STD_LOGIC;
           d : inout  STD_LOGIC);
end Main;

architecture Behavioral of Main is

begin

c <= b OR b;

end Behavioral;
```

- Save your code. Synthesize, which should then change to a green tick if all goes right. If not, go through the errors and warnings dumped on the console and debug.
- Then add a testbench for testing and simulation of your program by right clicking on the project and then 'New Source' as we did in step1 while adding VHDL module



File name: TestBench

Location: SE_DS_Lin_14.7_1015_1/SampleProject

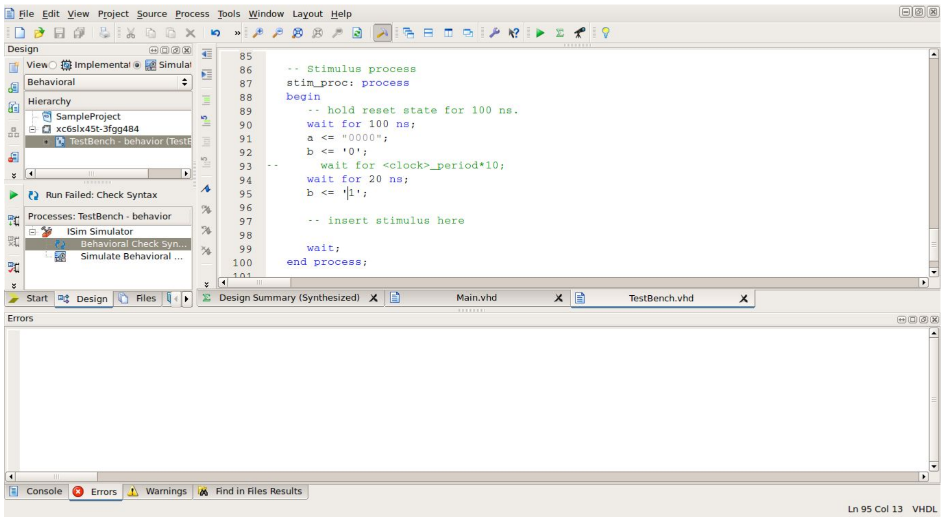
[✓] Add to project

Then click on Next -> Next -> Finish to see the testbench added to your project.

- If you are don’t need clock in your simulation, you can remove it from the testbench. Comment the clock_process entirely out of the code as shown below (shortcut: Alt+C)

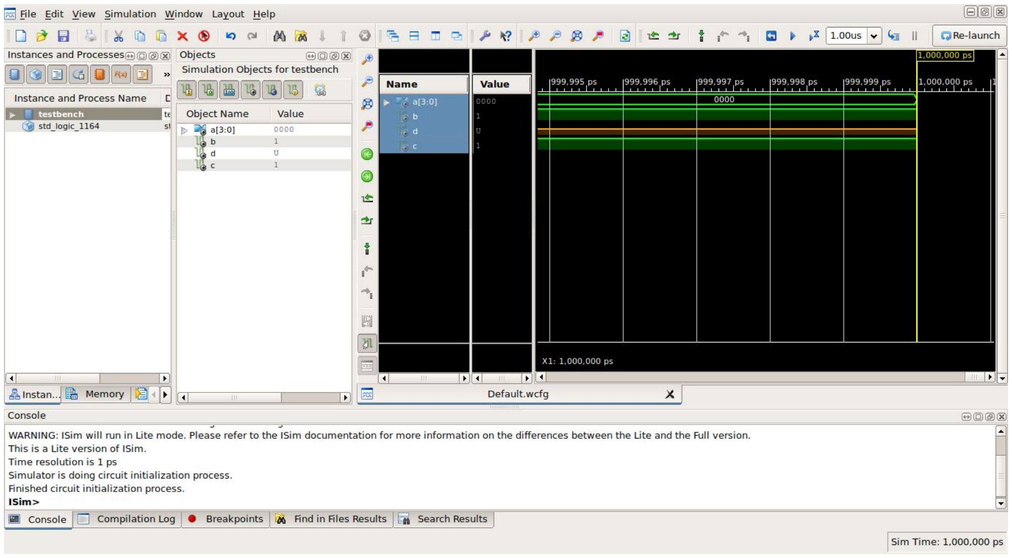
```
--   constant <clock>_period = 2*10ns

--   <clock>_process :process
--   begin
--                        <clock> <= '0';
--                        wait for <clock>_period/2;
--                        <clock> <= '1';
--                        wait for <clock>_period/2;
--   end process;
```

  Else use this clock to give waits.
- Visualize the testbench as a script to give various inputs to the program with timed waits. Consider the example given below:



```
   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      wait for 100 ns;
      a <= "0000";
      b <= '0';
--       wait for <clock>_period*10;
      wait for 20 ns;
      b <= '1';

      -- insert stimulus here

      wait;
   end process;
```

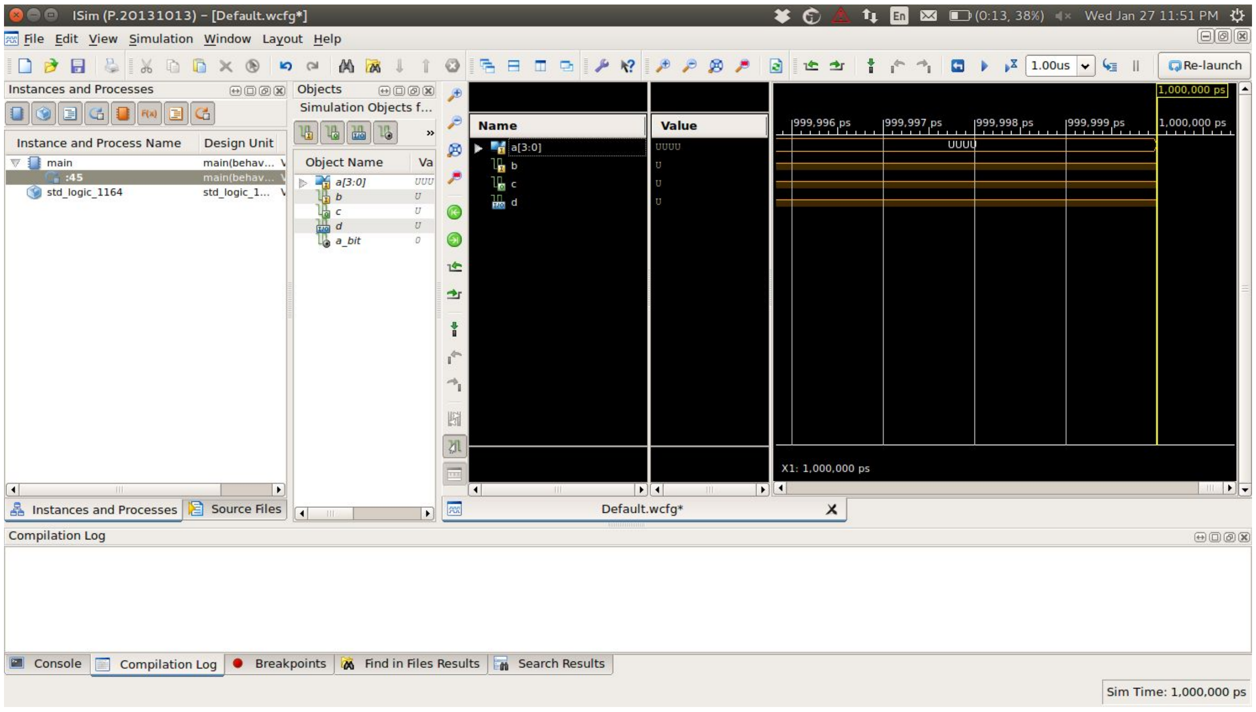
- Click on ‘Behavioral Check Syntax’. It should change to a green tick if all goes right. If not, go through the errors and warnings dumped on the console and debug.



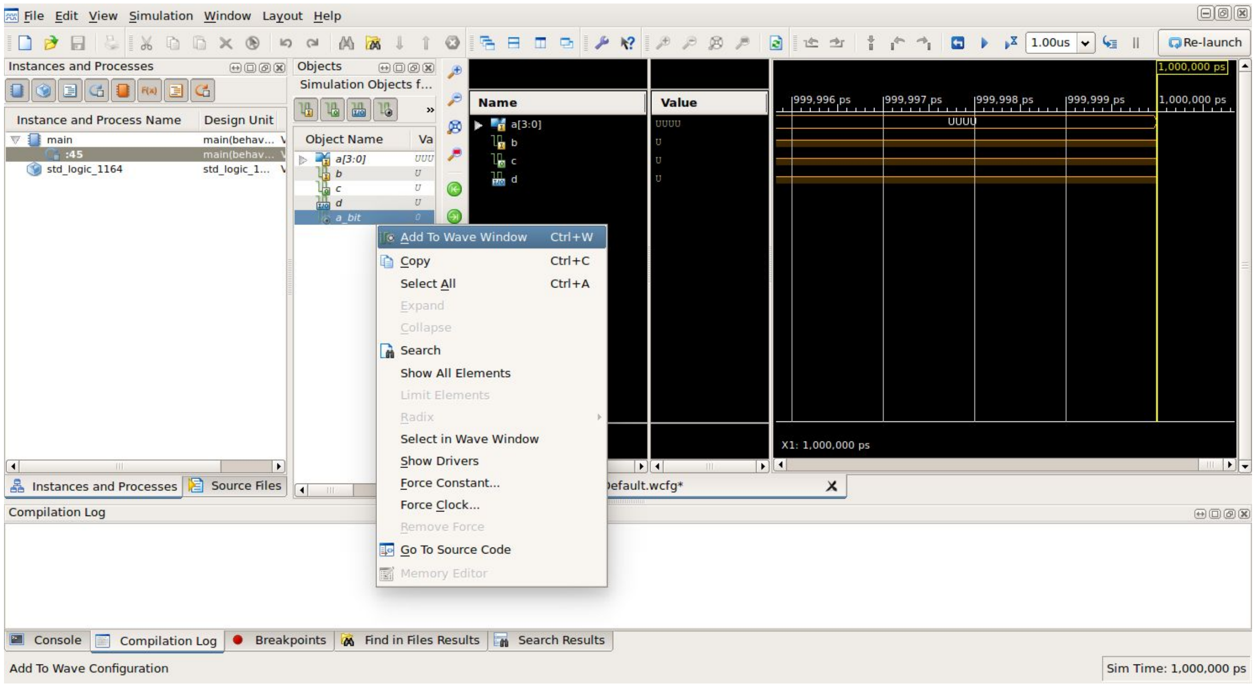
```
WARNING: ISim will run in Lite mode. Please refer to the ISim documentation for more information on the differences between the Lite and the Full version.
This is a Lite version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim>
```

Playing around with inputs

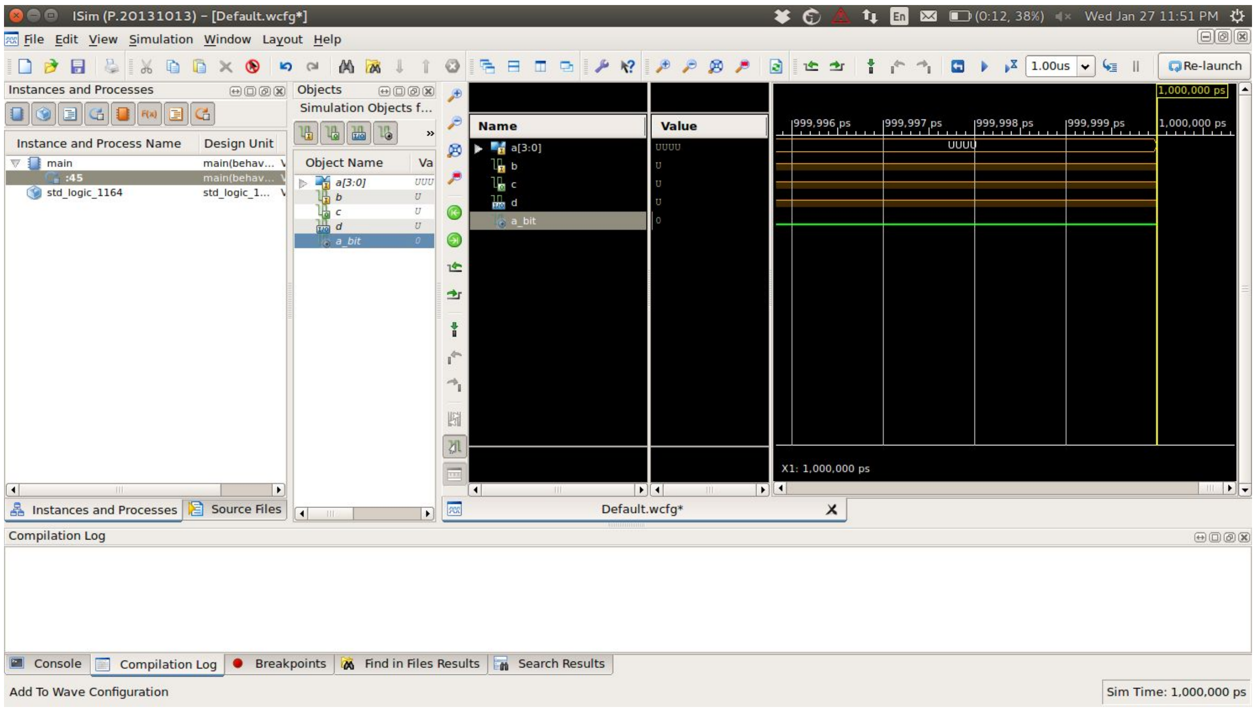
- You can also add signals and variables to waveform viewer window as shown
  Click on the main instance on left side of screen. Expand it to be able to see the objects



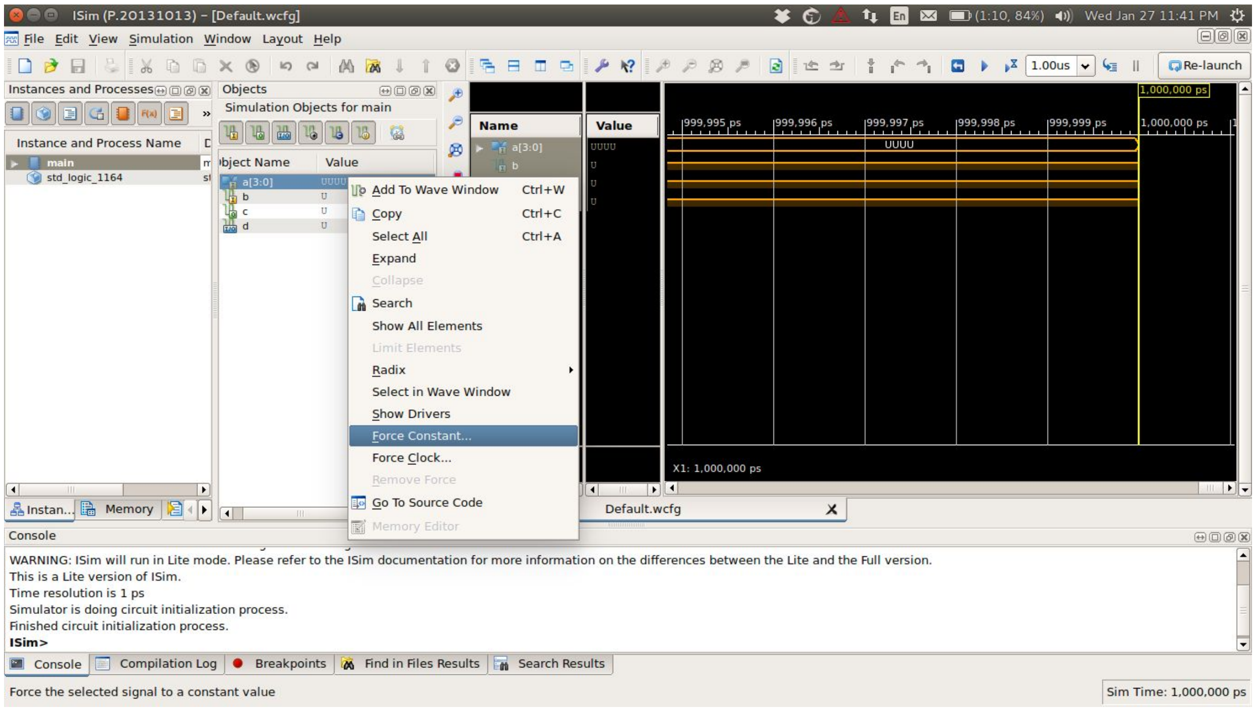
Right click on the variable/signal you wish to see in waveform. Click on 'Add to Wave Window' option



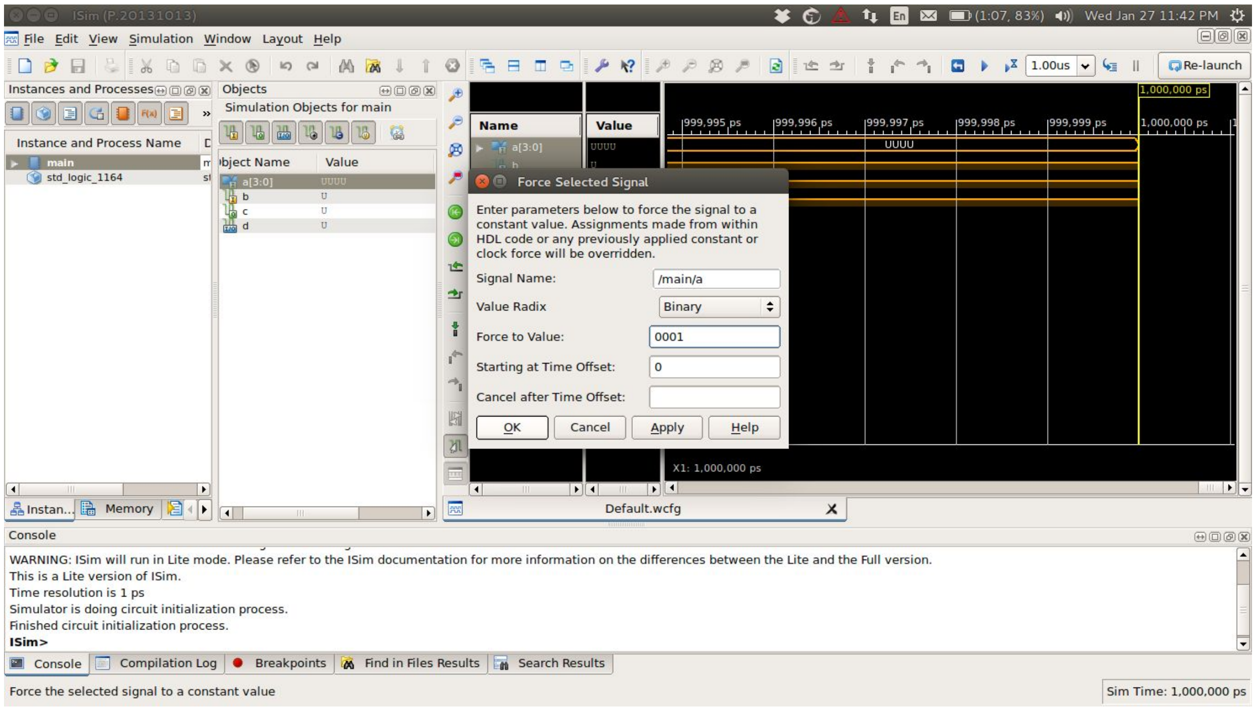
You can see your variable/signal added to the waveform.



- You can also force values to the input variables directly in the simulation as shown below



```
WARNING: ISim will run in Lite mode. Please refer to the ISim documentation for more information on the differences between the Lite and the Full version.
This is a Lite version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim>
```



Force Selected Signal

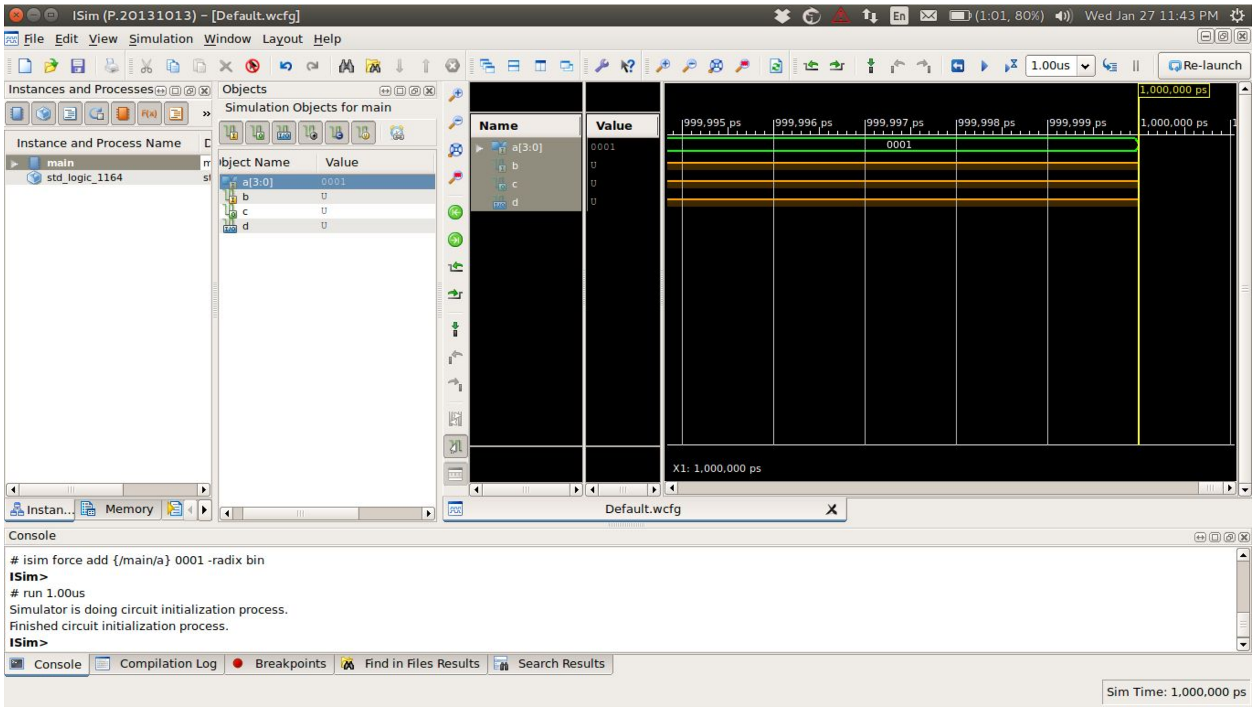
Enter parameters below to force the signal to a constant value. Assignments made from within HDL code or any previously applied constant or clock force will be overridden.

Signal Name:	/main/a
Value Radix	Binary
Force to Value:	0001
Starting at Time Offset:	0
Cancel after Time Offset:	

OK Cancel Apply Help

```
WARNING: ISim will run in Lite mode. Please refer to the ISim documentation for more information on the differences between the Lite and the Full version.
This is a Lite version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim>
```

Force the selected signal to a constant value



```
# isim force add {/main/a} 0001 -radix bin
ISim>
# run 1.00us
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim>
```

You can see the value ‘0001’ been forced on to the variable a.

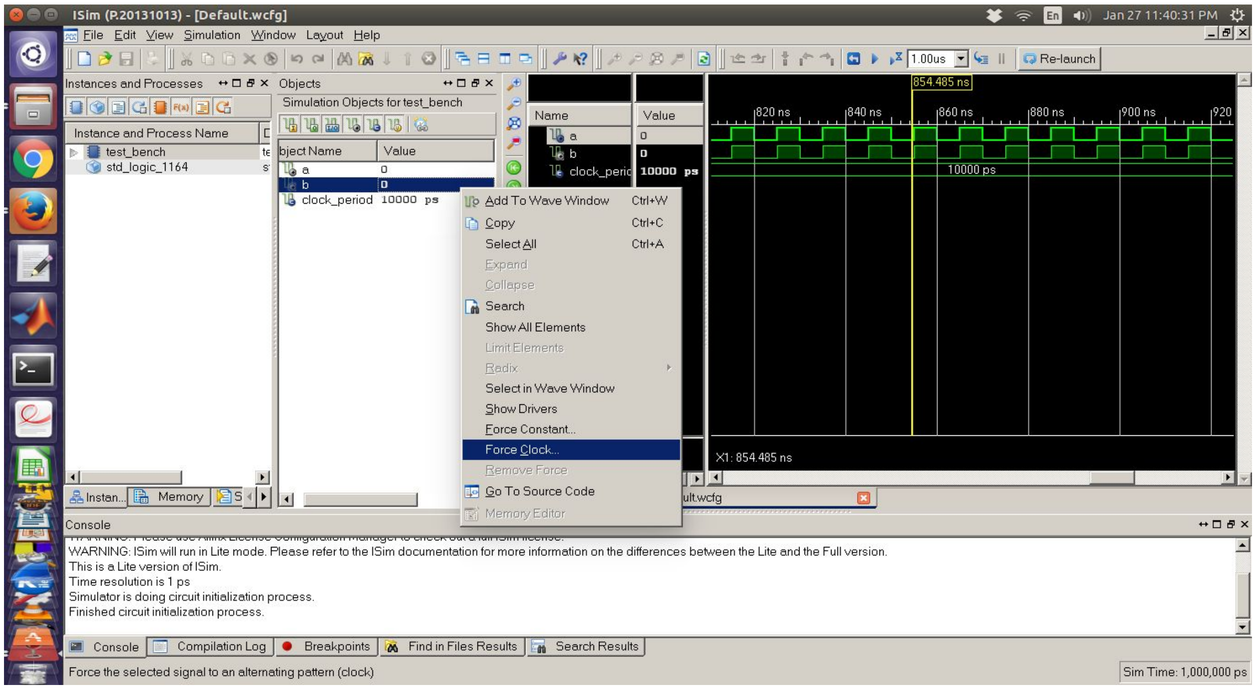
Modularity: Multiple files in same project

Often you would feel the need to modularize your code by putting different functional entities into different files. These files can be added the same way you add vhdl codes and testbenches in your project. To link them however, you would need to link their ports via what is called port mapping. Below is an example where the entity *arthur* is composed of internal VHDL modules *barney* and *elmo*. We define the components first with their port types and then map the ports to those *arthur*.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY arthur IS
      PORT( clk : IN STD_LOGIC; --clock input
            enbl : IN STD_LOGIC; --enable for arthur
            adv : IN STD_LOGIC; --advance the state
            pulse : OUT STD_LOGIC; --pulse the dog collar
            open_door : OUT STD_LOGIC --output to door driver
            );
END arthur;

ARCHITECTURE struct OF arthur IS
      SIGNAL ask_me : STD_LOGIC; --barney asks elmo
      SIGNAL clear : STD_LOGIC; --elmo clears request to ask

      COMPONENT barney PORT(
                  clk : IN STD_LOGIC;
                  enable : IN STD_LOGIC;
                  adv : IN STD_LOGIC;
                  clear : IN STD_LOGIC;
                  outa : OUT STD_LOGIC;
                  ask_me : OUT STD_LOGIC
                  );
      END COMPONENT;

      COMPONENT elmo PORT(
                  ask_me : IN STD_LOGIC;
                  go : OUT STD_LOGIC;
                  clear : OUT STD_LOGIC
                  );
      END COMPONENT;

      BEGIN
      U1 : barney PORT MAP(
                  clk => clk,
                  enable => enbl,
                  adv => adv,
                  clear => clear,
                  outa => pulse,
                  ask_me => ask_me
                  );

      U2 : elmo PORT MAP(
                  ask_me => ask_me,
                  go => open_door,
                  clear => clear
                  );
END struct;
```

Forcing Signal via ISim

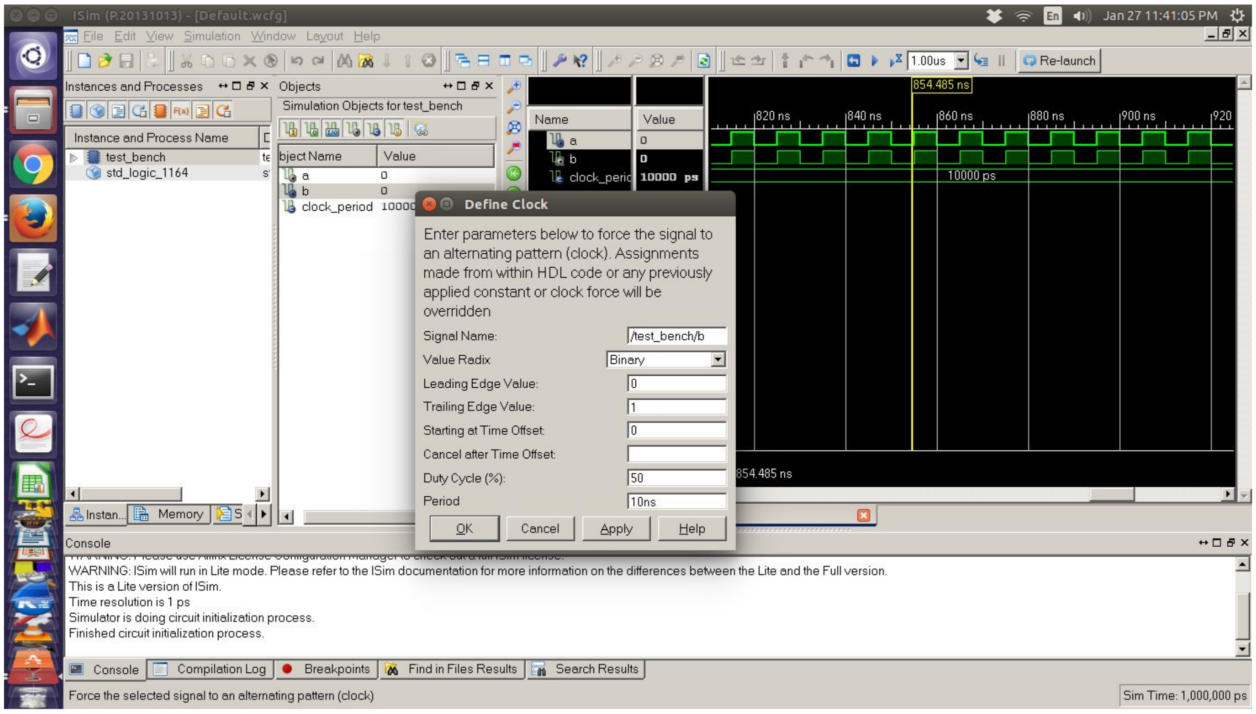
Right click on the variable you want to change -> select Force Clock/Force Constant option -> Fill the required details -> click OK -> Refresh.
Force to Value : 1
Starting at Time Offset : 10ns
Cancel after Time Offset : 30ns



```
WARNING: ISim will run in Lite mode. Please refer to the ISim documentation for more information on the differences between the Lite and the Full version.
This is a Lite version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
```

Force the selected signal to an alternating pattern (clock)

Sim Time: 1,000,000 ps



Enter parameters below to force the signal to an alternating pattern (clock). Assignments made from within HDL code or any previously applied constant or clock force will be overridden

Signal Name:	/test_bench/b
Value Radix	Binary
Leading Edge Value:	0
Trailing Edge Value:	1
Starting at Time Offset:	0
Cancel after Time Offset:	
Duty Cycle (%):	50
Period	10ns

```
WARNING: ISim will run in Lite mode. Please refer to the ISim documentation for more information on the differences between the Lite and the Full version.
This is a Lite version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
```

Force the selected signal to an alternating pattern (clock)

Sim Time: 1,000,000 ps

For more details view this video : https://www.youtube.com/watch?v=EhepIQIa9hg

Common Errors

In Entities and Architectures:

reference: **VHDL for Programmable_Logic by Kevin Skahill**

There are several common errors that are worth mentioning. Identifying them early may prevent misconceptions. Some of the errors are with syntax, others with semantics. Following is a code example with several errors. See if you can identify the errors:

```
entity many_errors is port --line 1
      a: bit _vector (3 to 0) ; --line 2
      b: out std_logic_vector(O to 3) ; --line 3
      c: in bit _vector (6 downto 0) ;) --line 4 \
end many_errors --line 5
                  line 6
architecture not_so_good of many_ errors --line 7
begin --line 8
my_ label: process --line 9
      begin --line 10
      if c x"F" then --line 11
      b <= a --line 12
      else --line 13
      b <= ' 0101' ; --line 14
      end if --line 15
end process; --line 16
end not_so_good --line 17
```

We'll take this design one line at a time because there are so many errors. The port declaration requires an "(" at the end of line I or beginning of line 2. In line 2, "to" should read "downto." The lack of the keyword IN to identify the mode is acceptable. If the mode is not explicitly declared, then the default of IN is assumed. Line 3 is ok. The semicolon on line 4 should appear after the second ")". The omission of a semicolon is one of the most common syntax errors. A semicolon is required at the end of line 5. There's even an error in line 6: The comment character " __ " is required. Line 7 is missing the keyword "is" after the name of the entity. Line 8 is ok. The process sensitivity list is missing from line 9. Line 10 is ok. The comparison of line 11 will always evaluate to FALSE because x"F" represents "1111" not "001111 "-six bits must be compared with six bits (this is not error, but is probably not what the designer wanted). In line 12, a signal of one type may be assigned to a signal only of the same base type, so we will have to change a or b to be of the same type. Line 13 is ok. The single quote marks (') of line 14 should be replaced with double quote marks ("). Line 15 requires a semicolon. Line 16 is ok. Line 17 requires a semicolon. The design is corrected and listed below.

Corrected code:

```
entity many_errors is port(
      a: std_logic_vector (3 downto 0);
      b: out std_logic_vector(O to 3);
      c: in bit_vector(6 downto 0));
end many_errors;
architecture not_so_good of many_errors is
begin
my_label: process(c, a)
      begin
      if c "001111" then
      b <= a;
      else
      b <= "0101";
      end if;
end process;
end not_so_good;
```

Initialisation of signals in architecture:

Credits: Team **techNologic**

```
architecture Behavioral of find_smallest_i is
      signal b_new : STD_LOGIC_VECTOR(N downto 0);
      signal i_current : STD_LOGIC_VECTOR(N-2 downto 0);

begin
      process(clk, reset)
      variable flag : boolean := false;
      b_new <= b; -- b is an input signal from entity definition
```

This snippet gives an error about initialising of b_new. Stating both b_new<3> and b<3> are active.
This can be simply fixed by initialising b_new in a proper fashion.

```
architecture Behavioral of find_smallest_i is
      signal b_new : STD_LOGIC_VECTOR(N downto 0) := b; -- changed line
      signal i_current : STD_LOGIC_VECTOR(N-2 downto 0);

begin
      process(clk, reset)
      variable flag : boolean := false;
```

In Creating Combinational and Synchronous Logic

reference: **VHDL for Programmable_Logic by Kevin Skahill**

Usage of variables:

In terms of processing VHDL for simulation, variables are only meaningful in a process and do not retain their values between periods of time when the processes are inactive. Variables must be reinitialized each time the process is activated. Synthesis creates logic based on these assumptions. Thus, the following code does not describe a counter:

```
architecture incorrect of bad_counter
begin
count: process (rst, clk)
      variable cnt: integer;
begin
      if rst = '1' then
            cnt := 0;
      elsif (clk'event and clk='l') then
            cnt := cnt + 1;
      end if;
end process;
end incorrect;
```

The variable ent does not retain its value for the time during which the process is inactive.

In using Arithmetic and Comparison operations

These functions are defined in the library 'IEEE.Std_logic_unsigned.ALL'.
reference: https://www.cs.sfu.ca/~ggbaker/reference/std_logic/unsigned/
The functions in this library take STD_LOGIC_VECTOR arguments and treat them as unsigned integers when performing the operations. Hence, this library must be included whenever any arithmetic operations are to be performed.

In Synthesis :

Signal is connected to multiple drivers

Consider the following code:

```
entity test is
port (  clk1 : in std_logic;
        clk2 : in std_logic;
        out1 : out std_logic_vector(7 downto 0)
    );
end test;
architecture Behavioral of test is
```

```
signal out2: std_logic_vector(7 downto 0);
begin
out1 <= out2;

process(clk1)
begin
if(clk1'event and clk1='1') then
out2 <= out2 + "00000001";   --increment by '1'
end if;
end process;

process(clk2)
begin
if(clk2'event and clk2='1') then
out2 <= out2 + "00000011" ;      -- --increment by '3'
end if;
end process;

end Behavioral;
```

The above code can be successfully compiled without any errors or warnings.But when you try to synthesis it you will get the following errors:

```
ERROR:Xst:528 - Multi-source in Unit on signal <0>>; this signal is connected to multiple drivers.
ERROR:Xst:528 - Multi-source in Unit on signal ; this signal is connected to multiple drivers.
...
ERROR:Xst:528 - Multi-source in Unit on signal ; this signal is connected to multiple drivers.
```

This error occurs when we try to change a signal in two different processes.As you can see from the above code the signal 'out2' is driven by two clocks,named clk1 and clk2.A multi-driven signal cannot be realized in hardware.If the clk1'event and clk2'event occurs at the same time then,the hardware doesn't know which statement to execute.That is why this kind of code is not synthesizable.And there is no way to solve this error.Only option is to change your logic in some way that you can get your things done without using a multi-driven signal.

For more details visit :
http://vhdlguru.blogspot.in/2010/03/synthesis-error-signal-is-connected-to.html

The Golden Rules of Debugging

reference : **https://www.doulos.com/knowhow/fpga/debugging/**

1) **Always fix the first error first:**

```
# Error: COMP96_0078: mux4_tb.vhd : (8, 16): Unknown identifier "STD_LOGIC_VECTOR".
```

This error generally pops on the top, the fix is most of the times

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

the error comes as the compiler can't resolve logic vector, so similarly when ever you get an unknown identifier try to find if you are including the right libraries

2) **Don't ignore warnings**

```
Warning (10492): VHDL Process Statement warning at alu.vhd(32):
signal "Op" is read inside the Process Statement but isn't in the Process Statement's
sensitivity list
```

This warning is interesting because most synthesis tools will actually automatically "fix" your sensitivity list, that is they will "pretend" that the missing signal is actually there. However simulation tools are not allowed to "pretend", so you will get a mis-match between simulation and synthesis. Your simulation may appear to be doing what you expect, but your hardware will behave differently. Or perhaps even more confusing, your hardware may behave correctly but your simulation will fail!
In that case the fix is easy, and the consequences are sufficiently bad you should fix the problem.

In synthesis you may be warned about latches - again this is an important warning that you should take notice of it.

3) **Try rebuilding**
Sometimes tools go wrong for no apparent reason.

4) **Don't Panic!**

Additional reference:

For syntax: http://www.ics.uci.edu/~jmoorkan/vhdlref/vhdl_golden_reference_guide.pdf
For further more errors:
http://class.ece.iastate.edu/cpre583/ref/VHDL/Common_VHDL_mistakes.pdf

Compiled By:

Utkarsh Kumar
Chandra Maloo
Aditya Kusupati
Anand Dhoot
Jash Dave