Basys 2 Setup Tutorial

Berke Dinç, Yalçın İşler

October 16, 2024

Abstract

This is a tutorial that explains how to setup the Digilent Basys 2 FPGA board with Xilinx ISE Webpack.

1 Install Xilinx ISE Webpack and Digilent Adept.

You can get Xilinx ISE Webpack here: http://www.xilinx.com/products/design-tools/ise-design-suite/ise-webpack. htm You can get Digilent Adept here: http://www.digilentinc.com/Products/Detail.cfm?Prod=ADEPT2

2 Launch Xilinx ISE Webpack from the Start menu.

Figure 1: Enter Caption

3 Create New Project In ISE

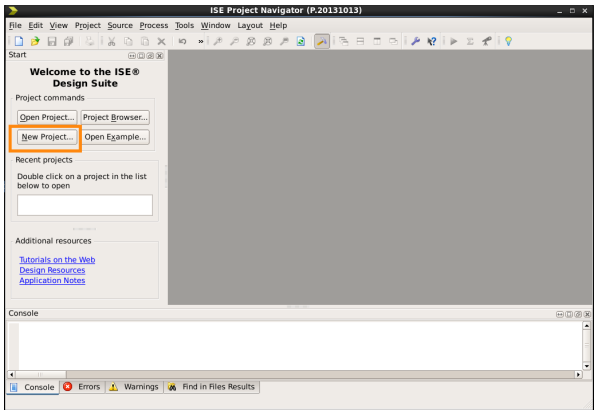


Figure 2: New Project

4 Configuring Project

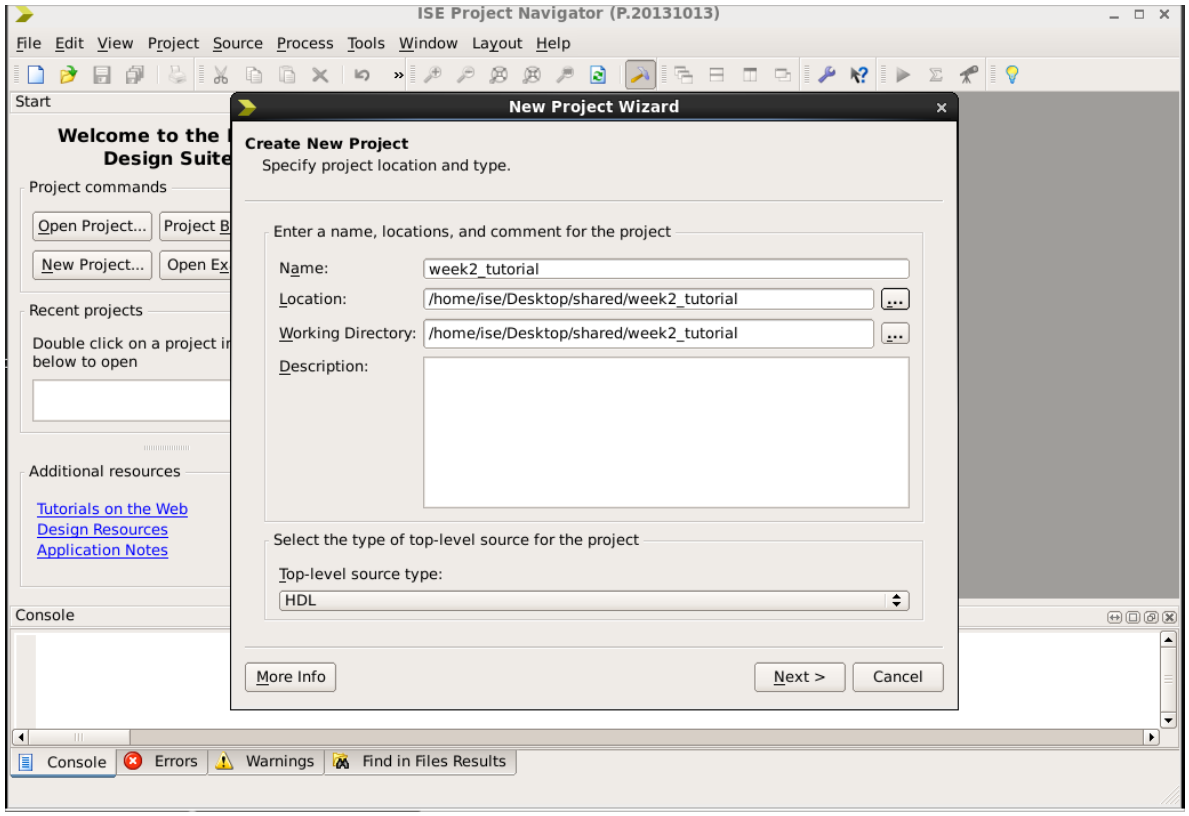


Figure 3: folder and name configuration

- Name: Name of the Project. Be careful to avoid using spaces and special characters when naming.
- Location: Location of project. Use an diretory that is not inside your Users home in windows systems because of avoiding long file paths and special characters inside username. better if you use Drive 'D' . Use this directory for all your projects.
- Working Directory: For most cases must be same with location of project
- Top Level Source Type: For most cases needs to be HDL for this lesson, there is another options like schematic design and EDIF.

4.1 Configuring Board And IC Settings

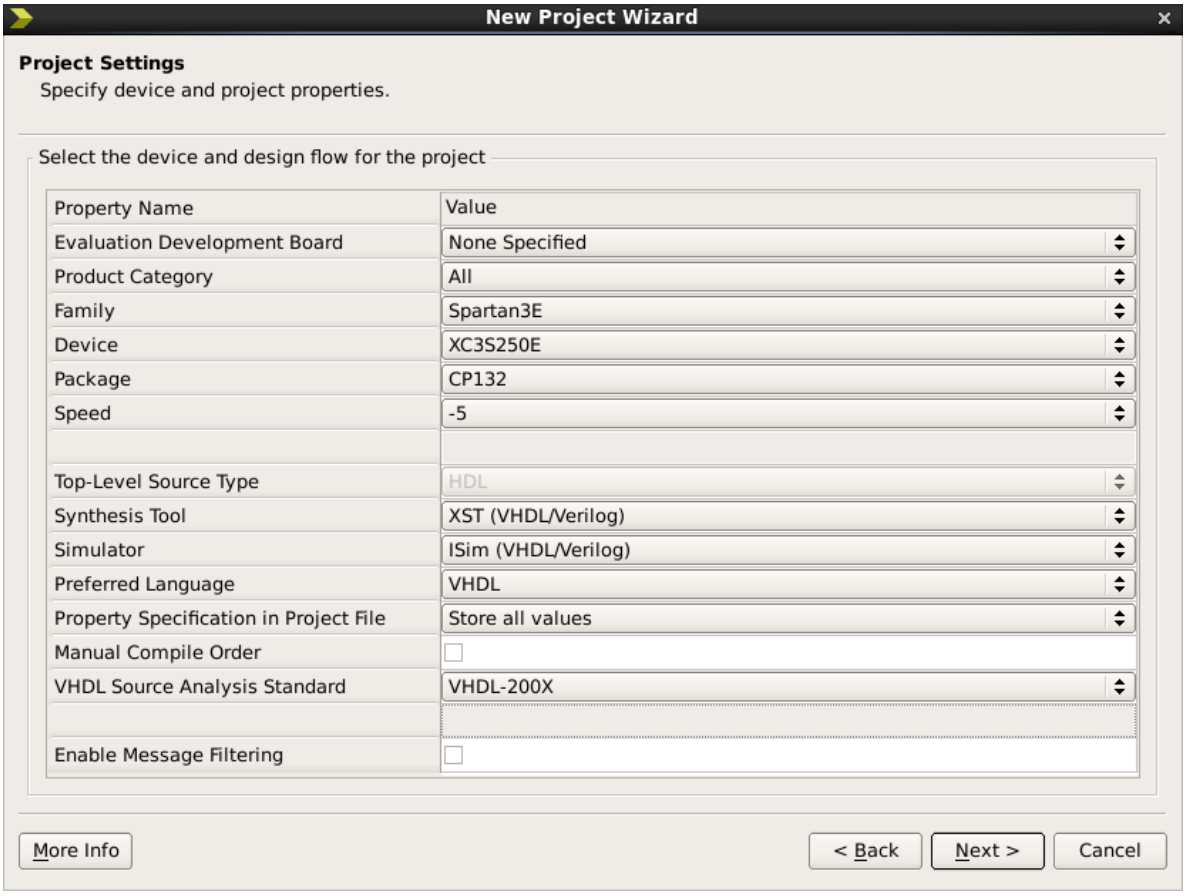


Figure 4: Settings For Basys2 Board

- Family: Basys2 Boards have xilinx spartan 3E family fpga's inside.
- Device: Model name of fpga IC, mine board have XC3S250E IC. 250 means that IC has 250.000 nand gates inside. some boards has 100.000 nands gates so check your board and small IC that middle of the front face of board. If it says XC3S100E on the IC, change this option.
- package: Most of the basys2 boards uses cp132 package. Check this from same IC.
- Prefered language: You can use verilog and VHDL at the same project (not recommended) but we need to declare main module's language in here. For this Lecture We are using VHDL so we select VHDL option.

Other items can be default. if you realy want to learn about other options go and search about it on google. Press next, And then Finish.

5 Project Page

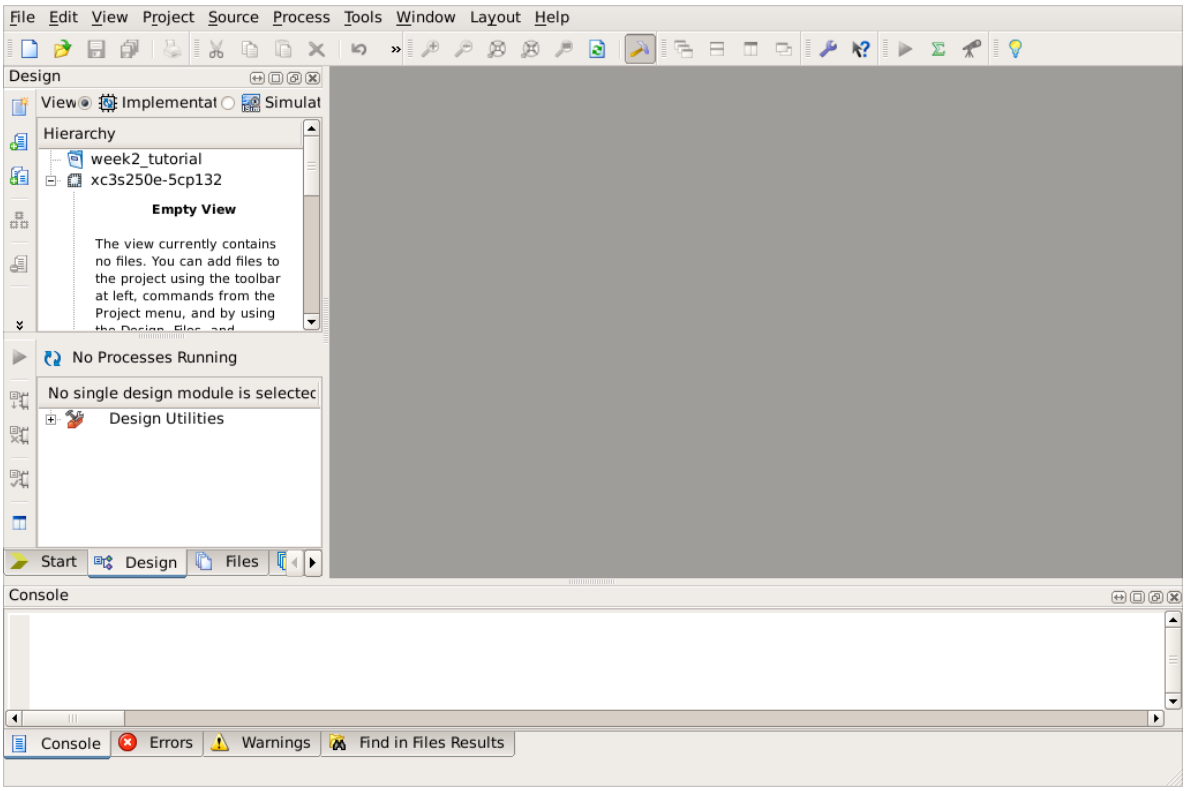
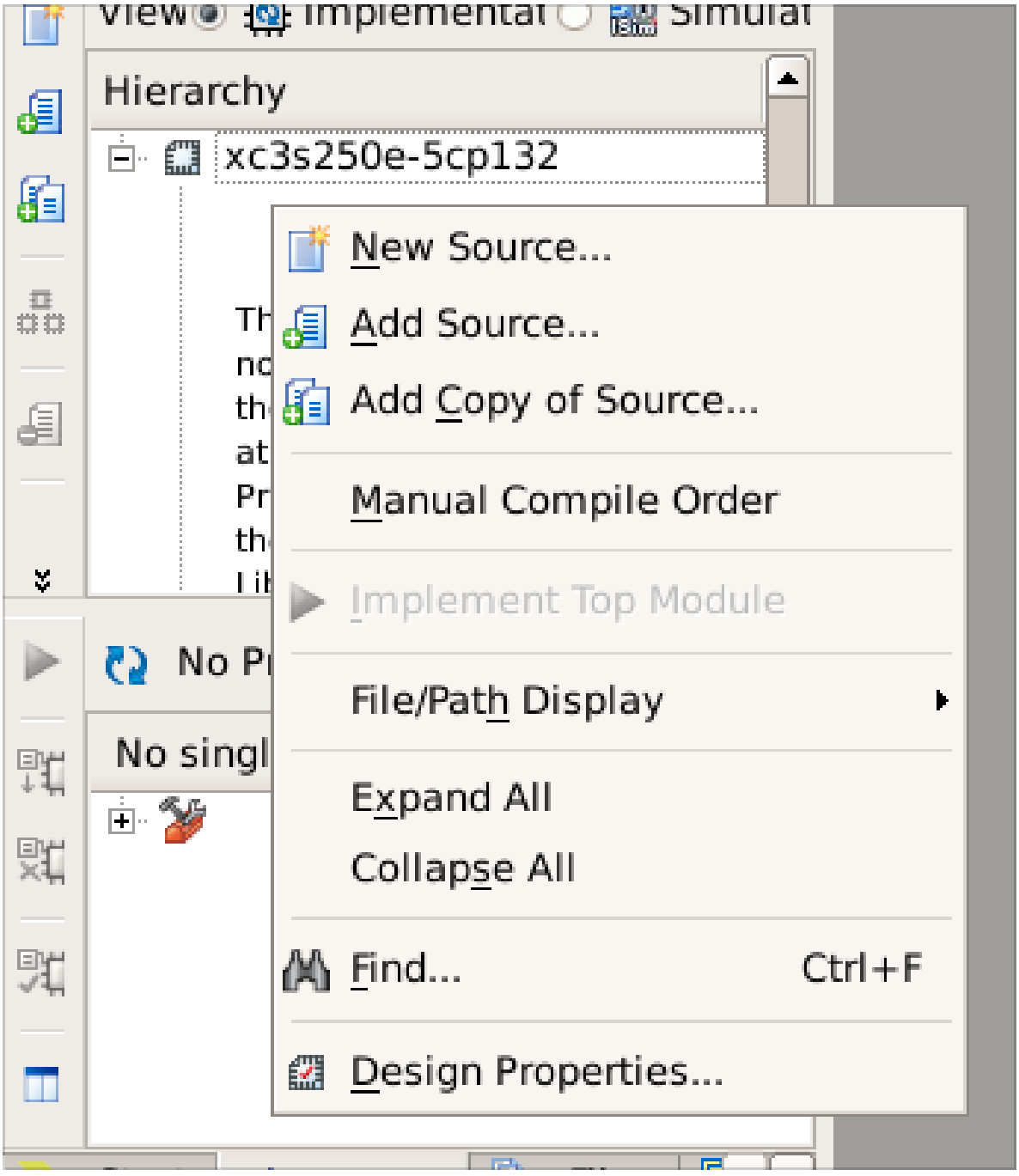


Figure 5: Project Page

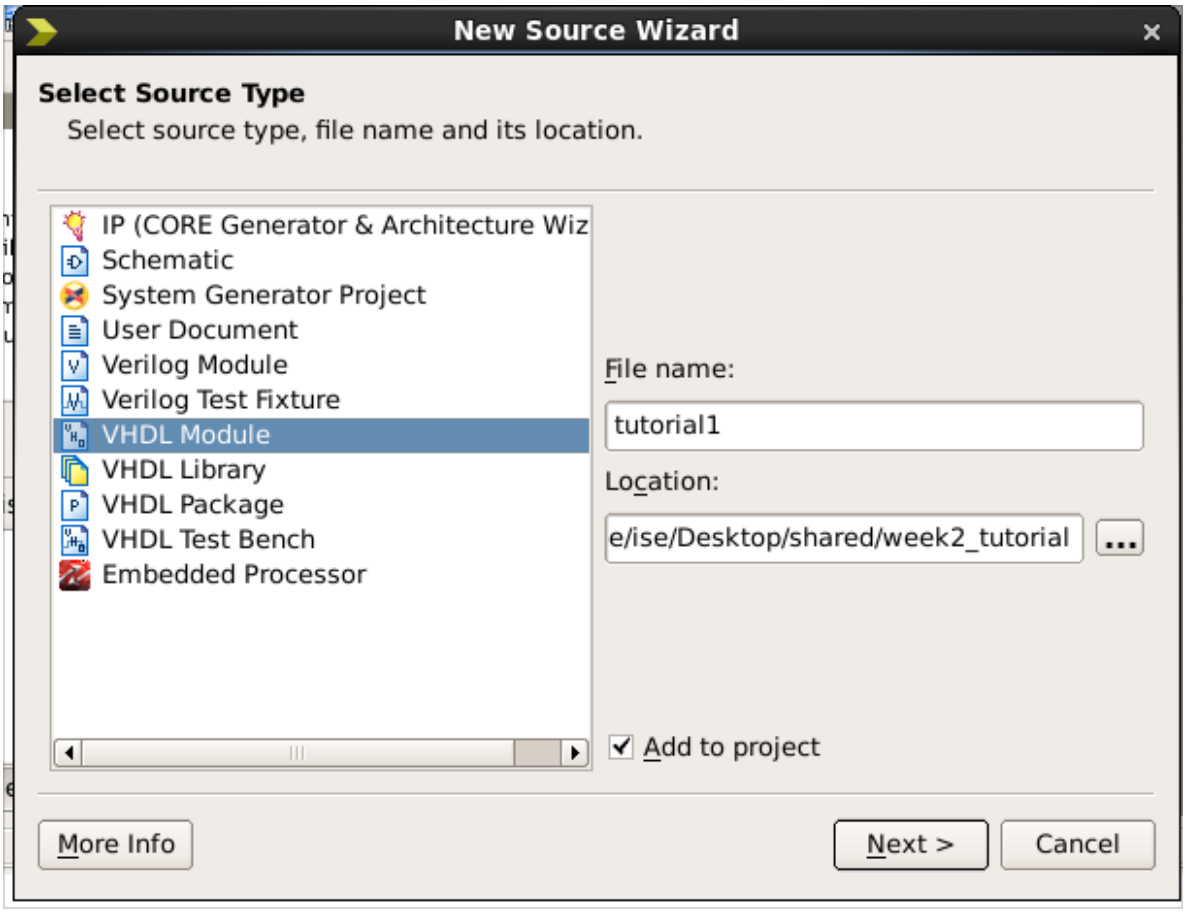
6 Adding Source

- right click to device(xc3s250e...) inside Hierarchy table. When popup shows select the new source.



7 New Source Wizard

- Here you can select what type of file you want to create.
- for this lesson we are going to use VHDL so we need to add VHDL module(you are always free to try other file types).
- Select the 'VHDL MODULE' option and give it a file name. check the location that it needs to be inside of project directory.
- If its okey press Next.



8 Defining Module

We can do this step by coding and personaly I will recomend the coding way but for demonstration purposes, here we can define inputs and outputs of module. From the table we see 5 columns that are

- Port Name: Name of the port that can be anything you want. Here we are going to make 'simple 3 bit counter example' so we need 3 input ports as 'enable, reset, clock'. And one output port that name is 'counterValue'.
- Direcction: Here we define direction of port. only the counter value was output so change direction to output for 'counterValue'.
- Bus: If our port needs to contain more than 1 bit we need to check this. then we can declare LSB and MSB values.
- LSB and MSB: as the name defines here we are declaring MSB and LSB values of port. for 3 bit counter, counter value needs to be contain 3 bits so we are saying that MSB = 2 and LSB = 0.
- select next and finish to create a file.
- Architecture: you can write anything here. I am going to use default which is Behavioral. I think they did this because most of the time we use behavioral modeling style but there is structural and dataflow modeling styles that we can use with VHDL and there is something called hls which is out of our scope.

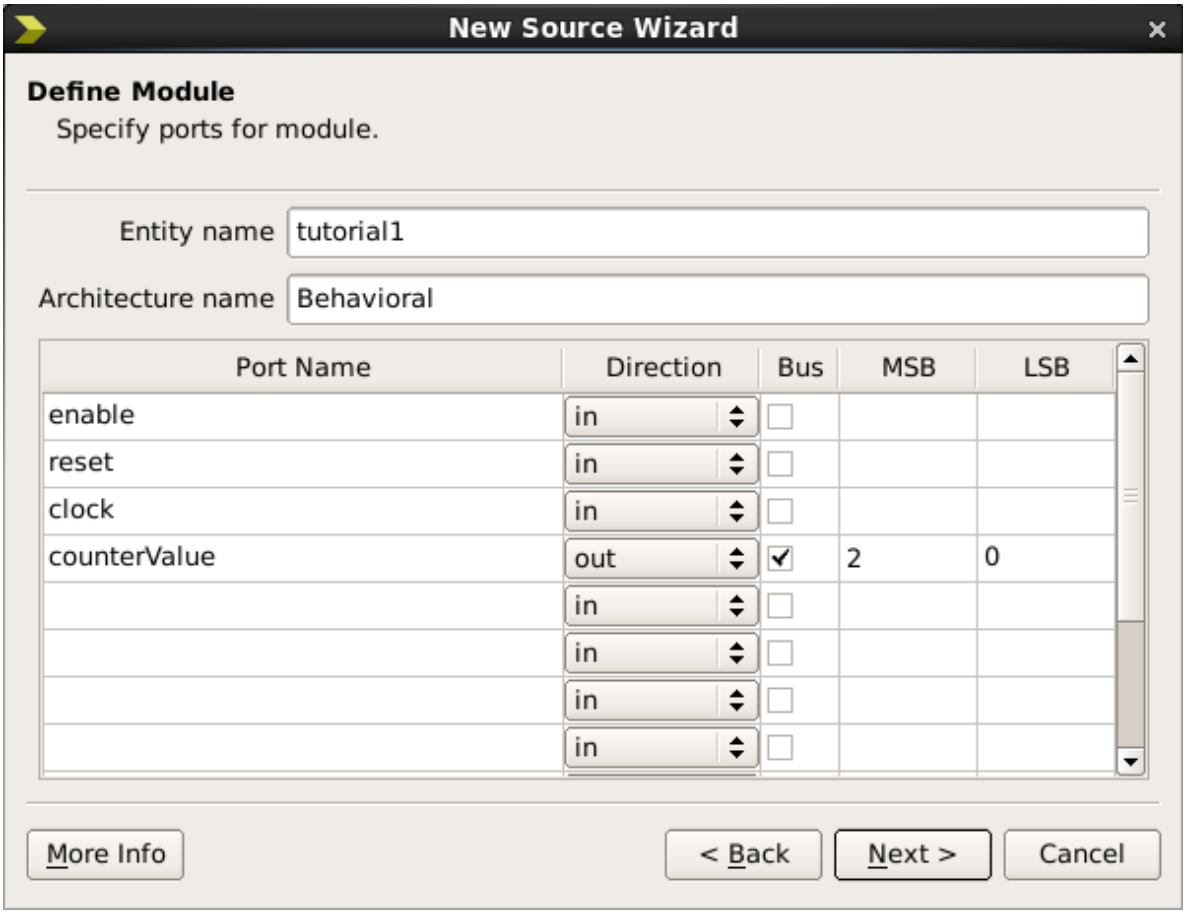


Figure 6: Defining module inputs and outputs

9 Getting Started to Defining Hardware

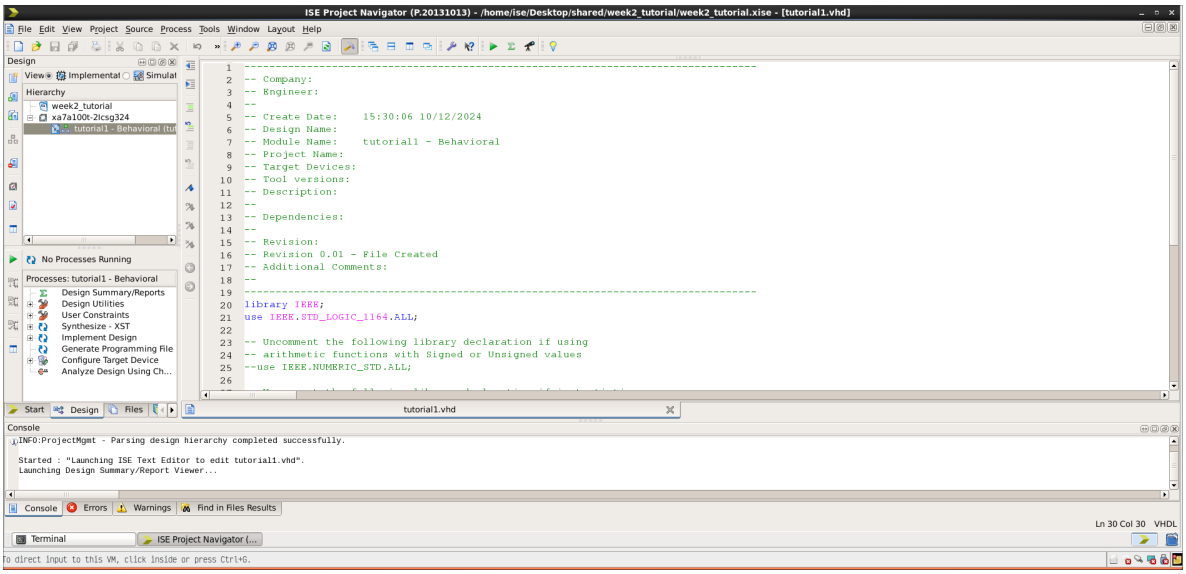


Figure 7: the code

- here we have the prewritten code based of the decleration before we done. I am going to delete comment line for better readability.

10 Code of Decleration Before We Done

```
entity tutorial1 is
    Port ( enable : in  STD_LOGIC; -- define enable as input
           reset : in  STD_LOGIC; -- define reset as input
           clock : in  STD_LOGIC; -- define clock as input
           counterValue : out  STD_LOGIC_VECTOR (2 downto 0)); -- define counter value as 3 bit vector output
end tutorial1;
```

Figure 8: prewritten code

11 Libraries

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.numeric_std.all;
```

Figure 9: Libraries

- Add this libraries

12 Code of 3 Bit Counter

```
11
12 architecture Behavioral of tutorial1 is
13    signal output:  unsigned( 2 downto 0 ) ; -- define counter value as output signal
14 begin --begin the module
15    counterValue <= std_logic_vector(output); -- turn unsigned to vector
16    process(clock, reset) -- Process sensitive to clock and reset signals
17    begin
18       if reset = '1' then -- if reset is one then do the following
19          output <= (others => '0'); -- '<=' means assign, other means all bits inside q, => '0' means map to 0.
20                                     -- its a bit confusing when to use '=>' or '<= 'or ':=' etc. We will cover that later.
21                                     -- so this line says define 0 to all bits inside q
22
23          elsif(rising_edge(clock))then -- when clock pulses then do the following
24                                        -- we dont include reset inside here so reset will be async interupt.
25             if enable = '0' then
26             output <= output + 1;
27             else
28             output <= output - 1;
29             end if;
30          end if;
31    end process;
32
33 end Behavioral; --end the module
34
```

Figure 10: the code

13 Synthesize

- Double click to Synthesize - XST and run it. if everything was okey green tick was appear.

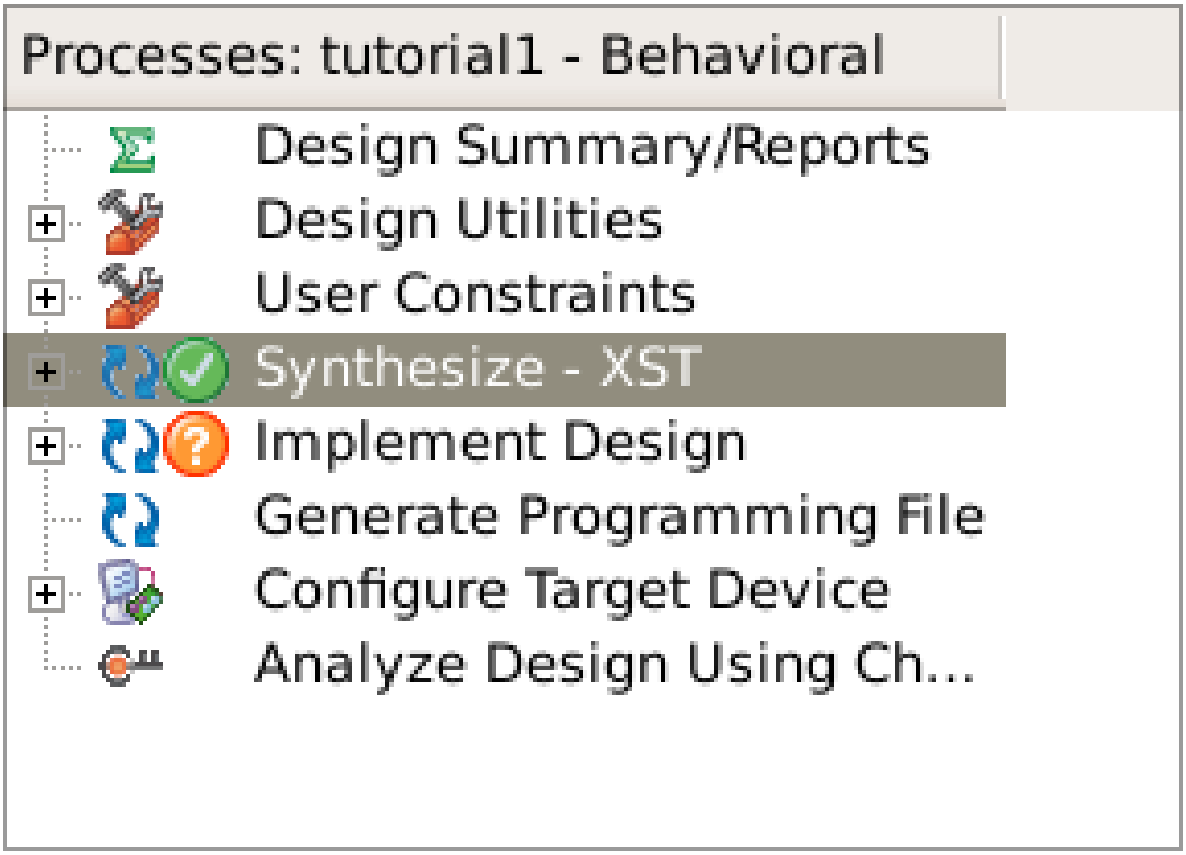


Figure 11: Enter Caption

14 Creating Simulation File

- Before uploading our code to the basys2 board, we have to simulate our code for debugging.
- For this we are going to add new source but this time selecting 'VHDL Test Bench' type, name file to 'tutorial1_test' you can name the file what you want.
- press next

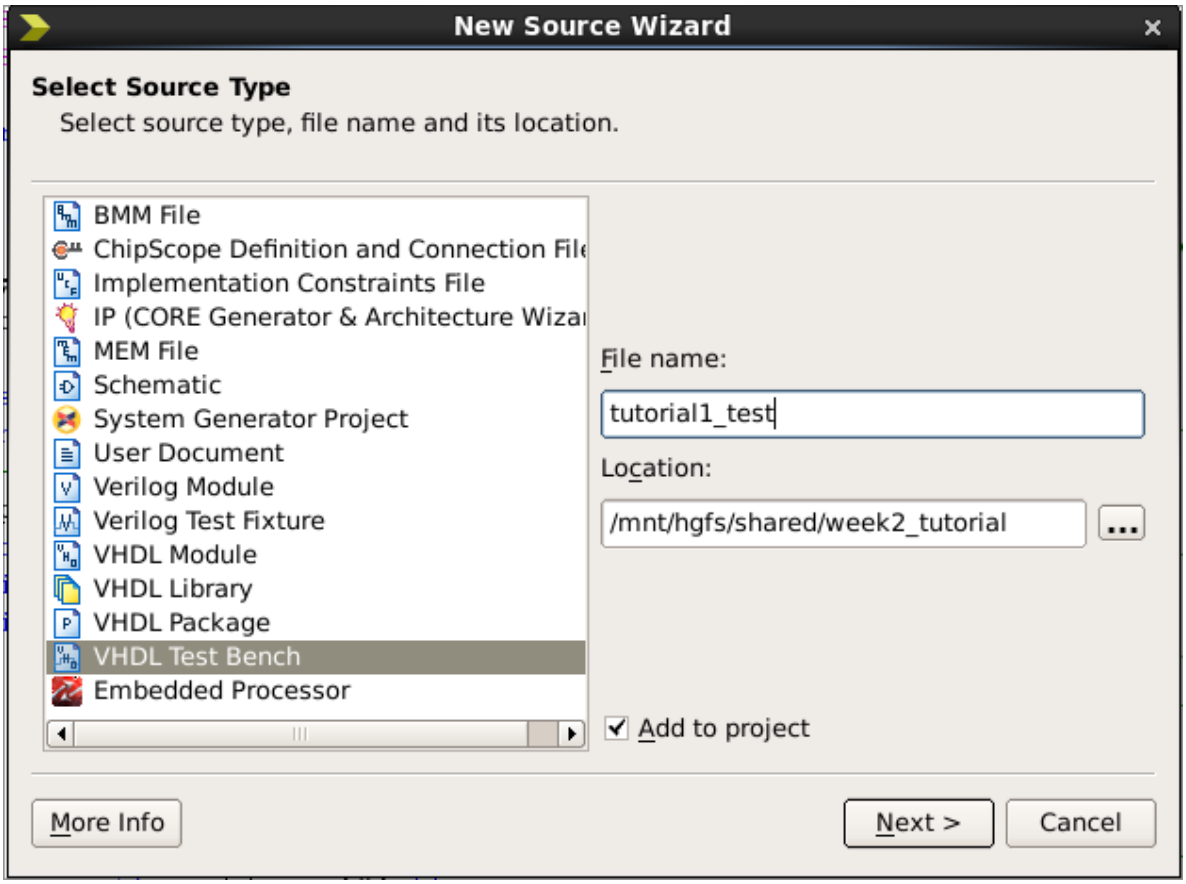


Figure 12: Adding Test Source

- in the next window we need to select source module that we want to simulate, there was an only one module which is 'main module'. For more complicated project you might have more than one module.
- for this lesson we select the 'tutorial1' module and press next. and than finish.

Figure 13: Enter Caption

15 Simulation Code

- ISE will create file named 'tutorial1_test.vhd' we can see this file by selecting 'simulation' at the view section.

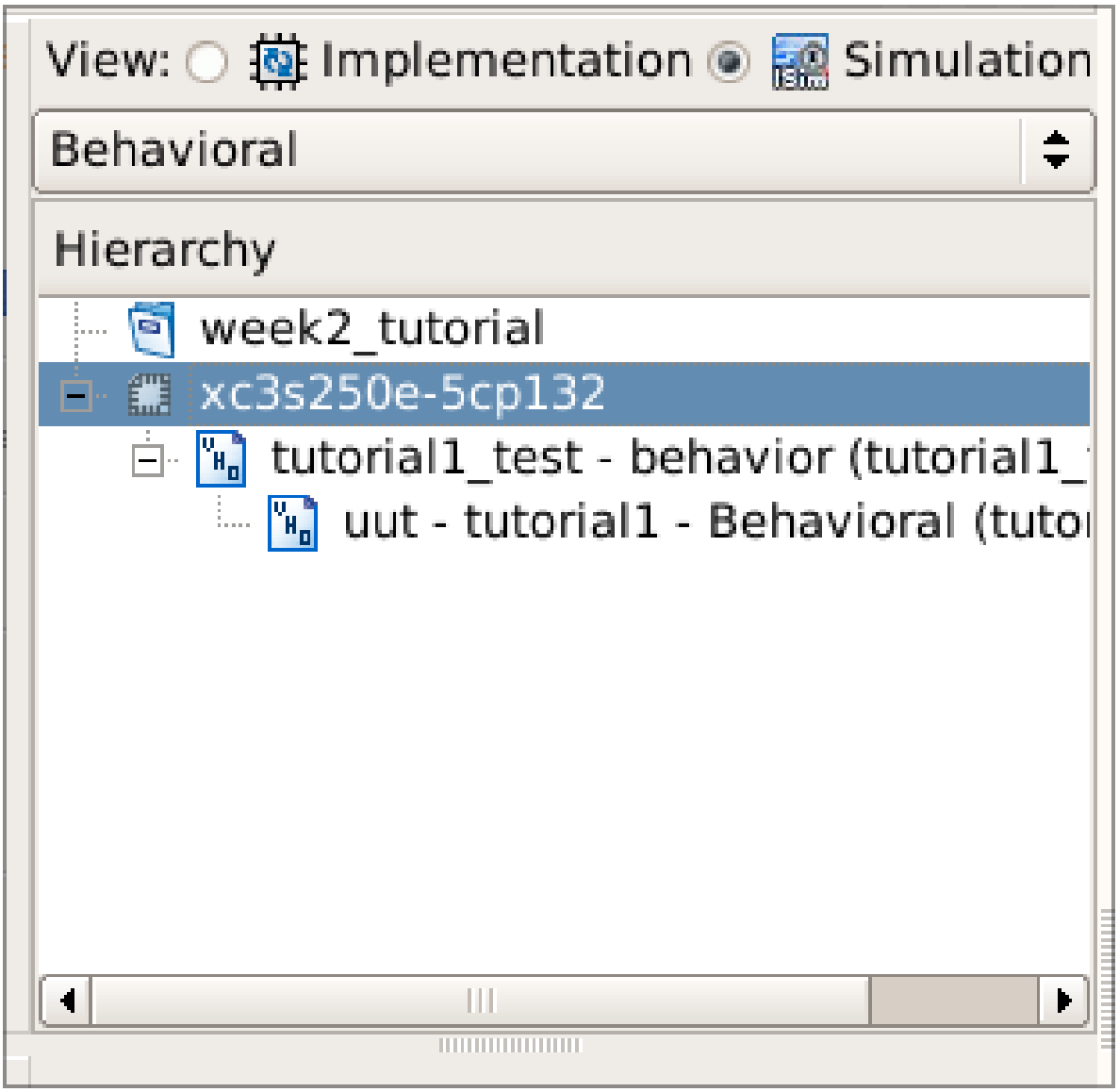


Figure 14: Simulation View

- At the same time simulation file must be opened inside editor.
- I am going to delete all comment lines at the starting of the file for better readability.

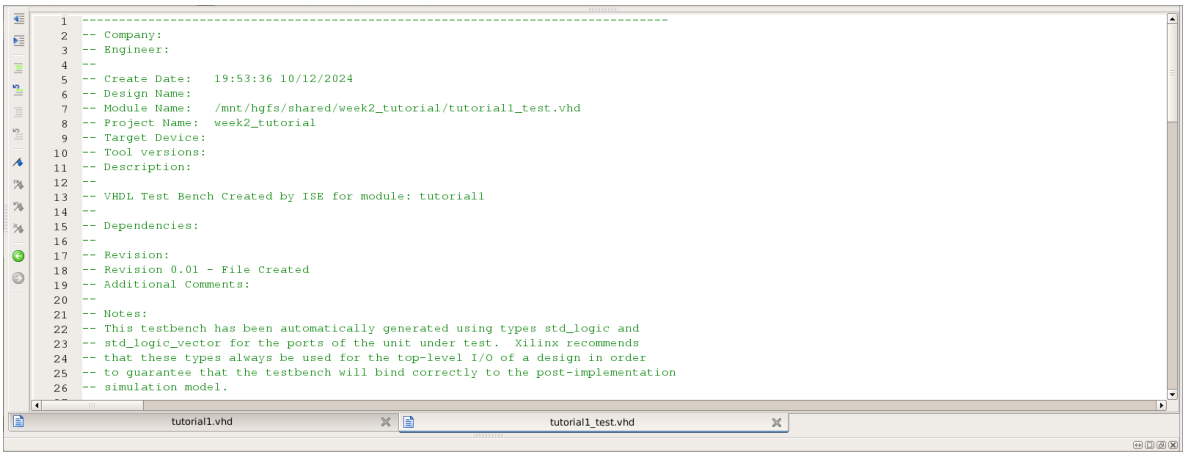


Figure 15: Enter Caption

- We see that all our main modules inputs and outputs, as well as clock period, clock process(how clock works) etc. was declared so we don't need to do this stuff.
- You are always free to study all code line by line but...
- More interesting part of the code was at the end of file. scroll to the very end of the file and you will see the part named 'Stimulus process'

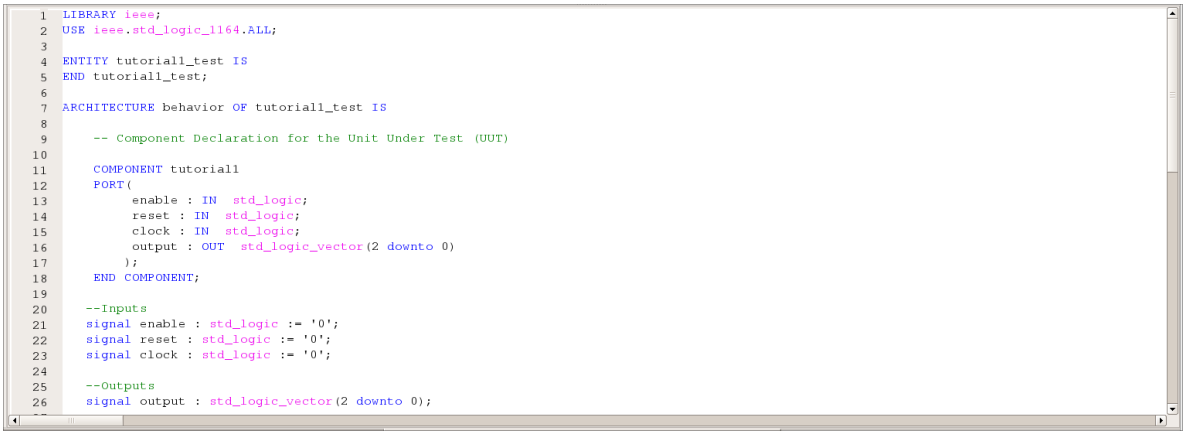


Figure 16: simulation declarations

16 Writing Stimulus Process

- from here we can declare how the inputs are manipulated during simulation.

```
-- Stimulus process
stim_proc: process
begin
    -- hold reset state for 100 ns.
    wait for 100 ns;

    wait for clock_period*10;

    -- insert stimulus here

    wait;
end process;
```

Figure 17: running simulation

17 Declaring Stimulus Process

- Write this code inside 'stim_proc' process. Comments are describing what will code do line by line.

```
51  -- Stimulus process
52  stim_proc: process
53  begin
54     reset <= '1'; --set reset 1
55     wait for clock_period*5; -- wait 5 clock pulses for initialization
56     reset <= '0'; --set reset to 0 so program can start to count.
57     wait for clock_period*10; -- wait for 2^3=8 clock cycles and 2 more for see what was going to happen
58     enable <= '1'; -- set enable to 1 so we can see what gonna happen. we are expecting program gonna count down on every clock pulse
59     wait for clock_period*10; -- wait for 2^3=8 clock cycles and 2 more for see what was going to happen
60  end process;
```

Figure 18: Rewritten Stimulus Process

18 Running Simulation

- First select 'tutorial1_test' file.
- Then run Behavioral Check Syntax. if there is no error green tick was shown near it.
- If there was an error red cross will appear, in that situation read the console message and fix the error.
- Lastly run Simulate Behavioral Model and simulation windows appears.

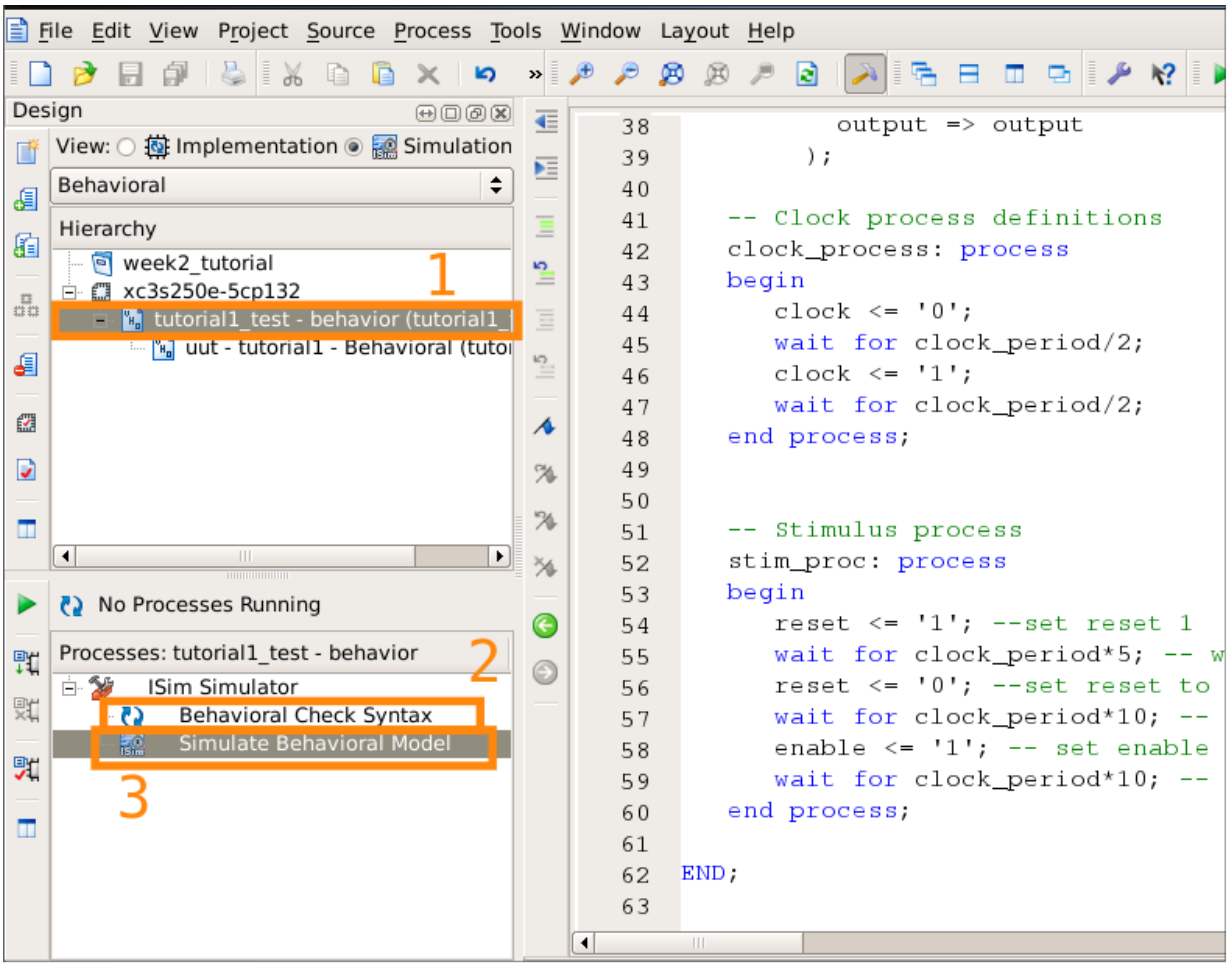


Figure 19: running simulation

19 Analyzing Simulation

- First set simulation time to 1ms or any other value that is enough to see all conditions.
- Then press the button left of it. marked by 2. this button runs simulation by defined time in our case 1ms.
- After simulation was finished we can check the outputs.
- In ideal conditions all possible input values needs to be declared and outputs needs to be checked.

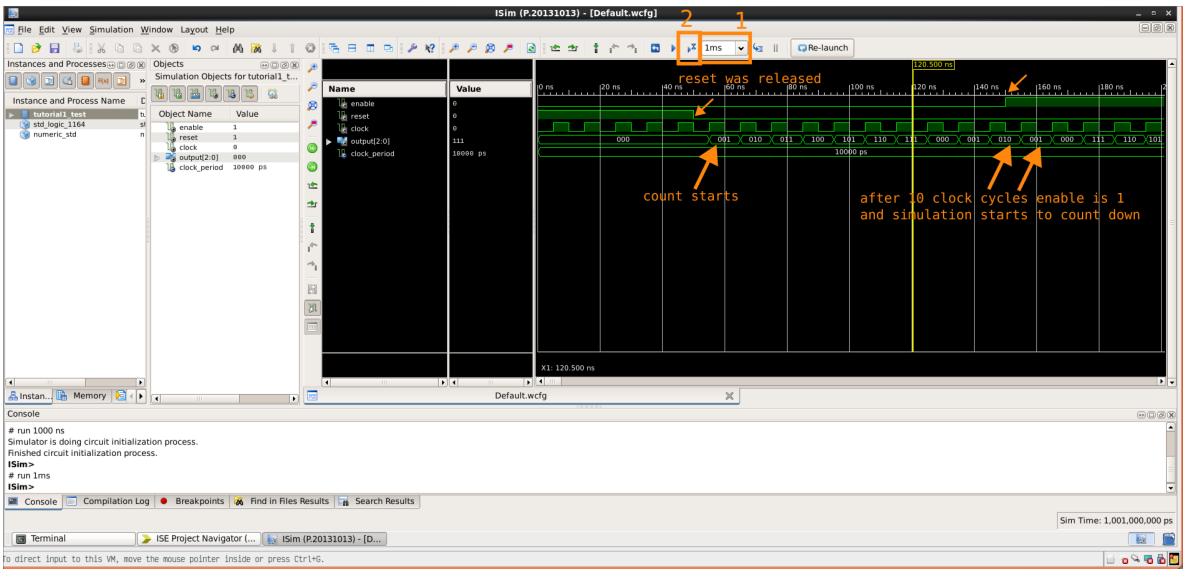


Figure 20: ISIM window

20 Implementing Design to Board

- Now we need to declare pins of fpga ic to inputs and outputs of our design.
- For this we are going to add new source 'implementation constraints file'.

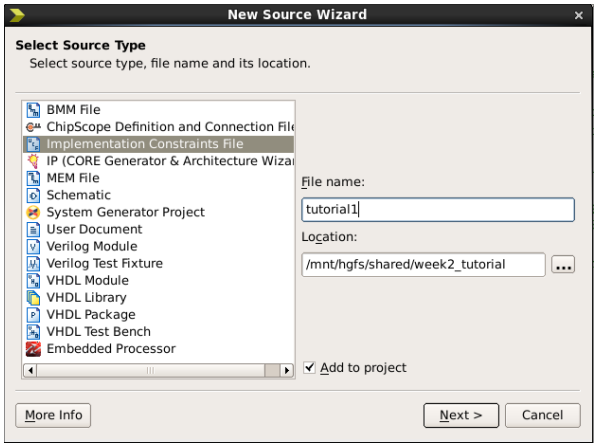


Figure 21: UCF New Source Wizard

21 Opening PlanAhead

- We can do this by coding way. personaly I recommend doing this by coding way. but for demonstration purposes there was a tool called 'PlanAHead'. To run this tool select out main module and than extend 'User Constrains' by pressing small '+' sign near it.
- Double click to '(PlanAhead) - Post-Synthesis' because we are creating .ucf file after synthesis. In case we need to create .ucf file before synthesis, we need to select Pre-Synthesis option.
- Click yes if something appears without reading it.

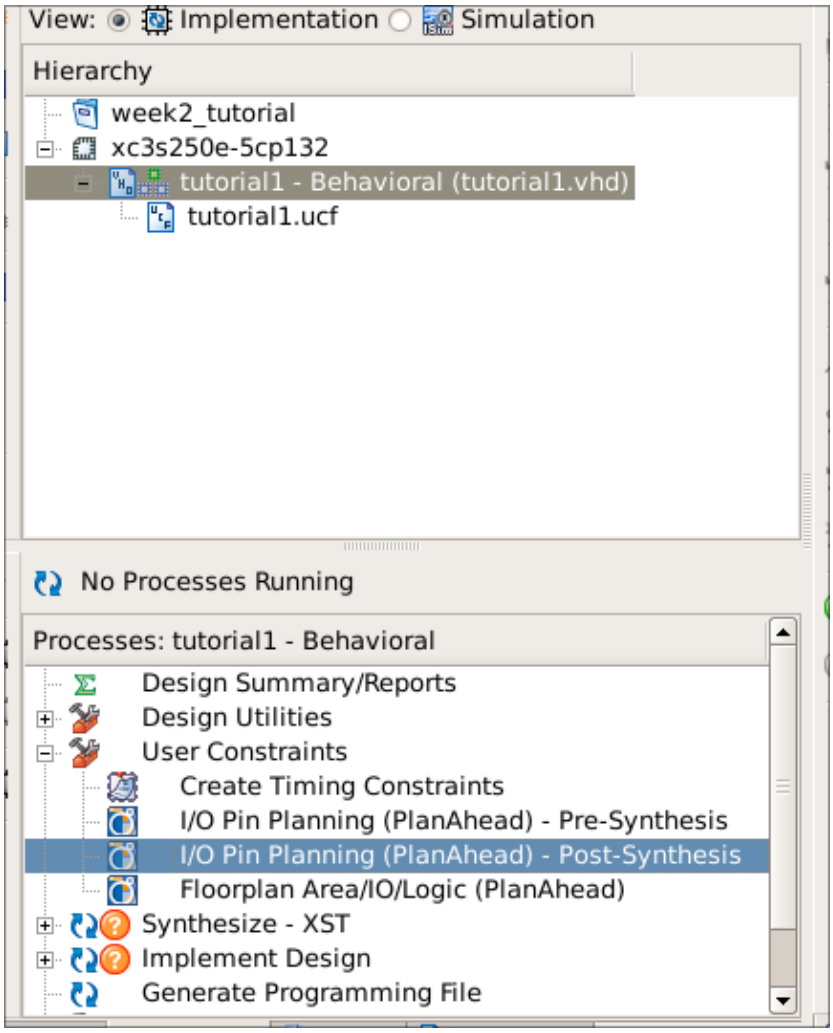


Figure 22: Opening PlanAhead

22 Which Pins to Use

- Here in front face of basys2 we can see bunch of components.
- Firstly we need to decide which buttons switches or leds we are going to use
- As you can see at the Figure 23, I am going to use 'LD0, LD1, LD2' for showing output of CounterValue
- SW0 as enable.
- BTN0 as reset.
- Green arrows shows that component was connected to which pin of the fpga ic.
- for example SW0 is connected to P11 pin of the fpga ic. We are going to use this information for creating .ucf file.

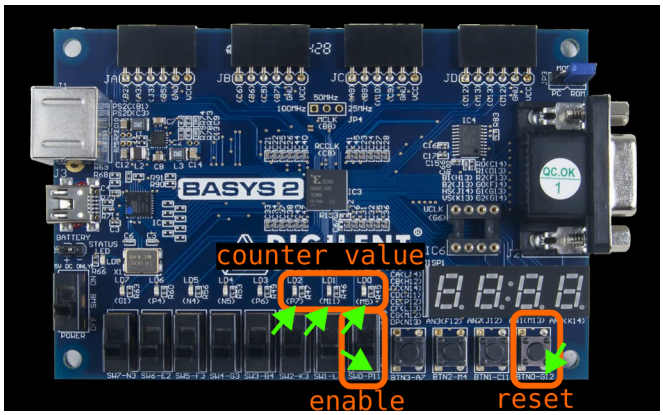


Figure 23: Basys2 Board

23 Using PlanAhead

- At the lower side of windows you will see our module's ports.
- From here extend all ports and from the site column select which pin you want to connect to that port.
- for example we want to connect enable to SW0, SW0 is connected to P11 pin of fpga as shown in the figure 23 by green arrow, so we are going to select P11 pin for enable port.
- For clock signal we are going to use basys2 boards onboard 50MHz which is connected to C8 pin of fpga.
- Repeat this for all ports as shown in the figure 24.
- Then save by clicking top left cassette icon or press 'ctrl-s'.
- Now we can close the PlanAhead.

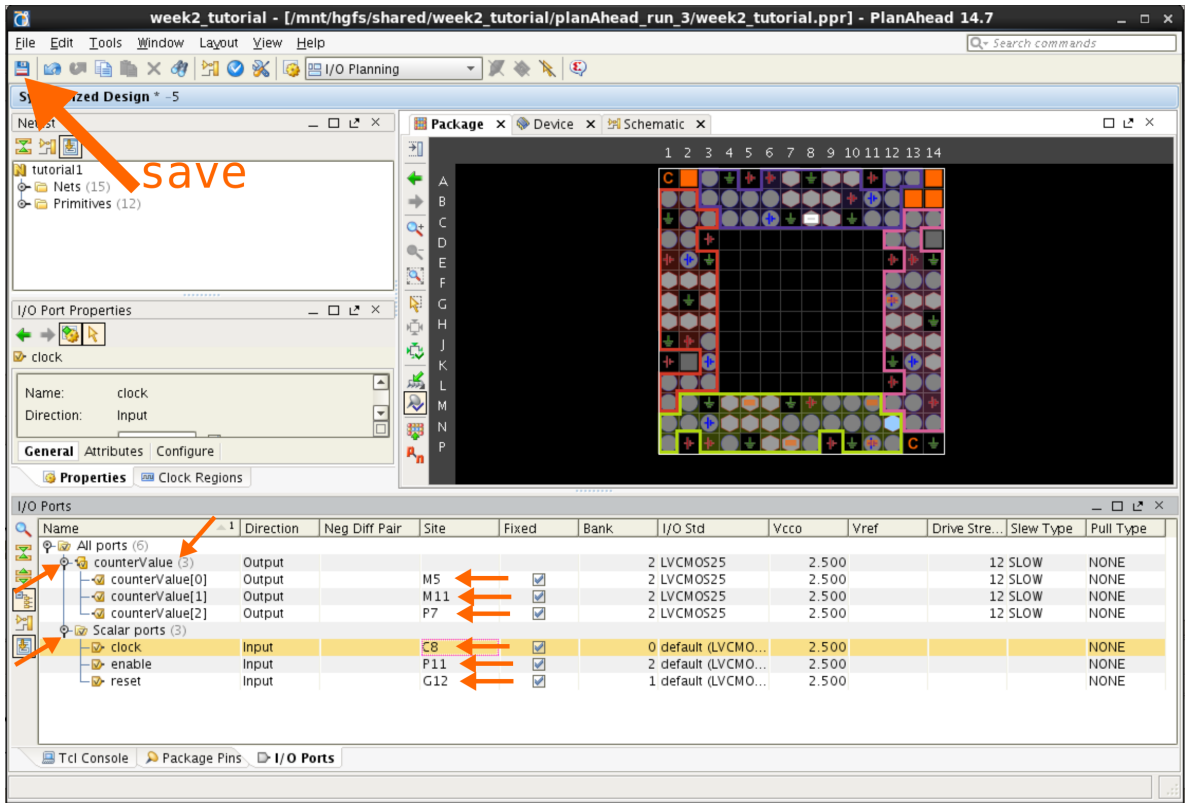


Figure 24: PlanAhead Window

24 Let's Look at .ucf File

- Here we can see all constrains we defined in PlanAhead in ucf format.
- I will recommend creating this file by hand without using PlanAhead tool.
- As you can see syntax is really easy to use. NET "PortName[Bit number if Port is vector/bus etc.]" LOC pin of FPGA;

```
1
2  # PlanAhead Generated physical constraints
3
4  NET "counterValue[2]" LOC = P7;
5  NET "counterValue[1]" LOC = M11;
6  NET "counterValue[0]" LOC = M5;
7  NET "enable" LOC = P11;
8  NET "reset" LOC = G12;
9
10 # PlanAhead Generated physical constraints
11
12 NET "clock" LOC = C8;
13
```

Figure 25: .ucf File

25 Generating Programming File

- Duble click to "Generate Programming File" inside process window.
- İf there was now errors green tick will appear. If there was an error red cross will appear.
- If there was an error check the console and correct errors.

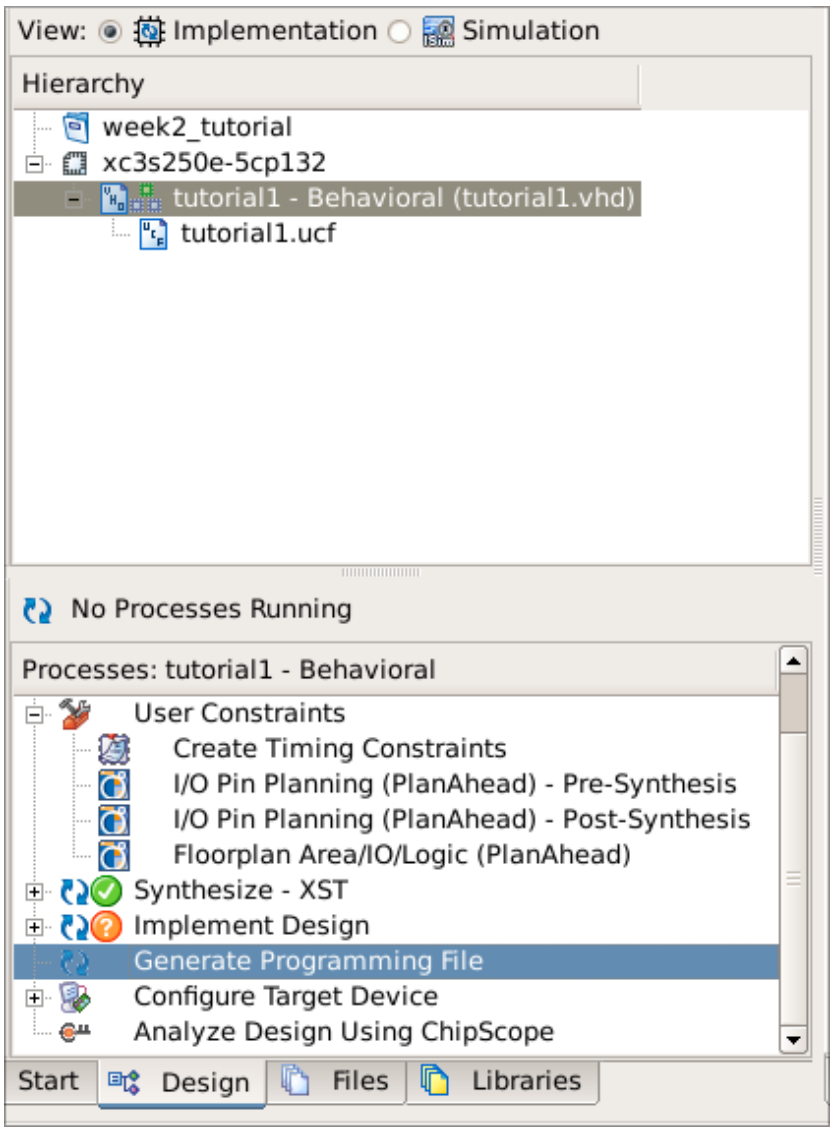


Figure 26: Generate Programming File

26 Uploading Bit File

- Now connect Basys 2 Board to Your pc.
- Open Adept 2 Software.
- Basys2 needs to be shown as connected, if not check your connection.
- here we see 2 options:
- the top one is for is for programming fpga without loading program to memory so if we restart the fpga the program will be gone.
- bottom one is for uploading our program to memory on the basys2 board.
- for this tutorial we are going to use top one.

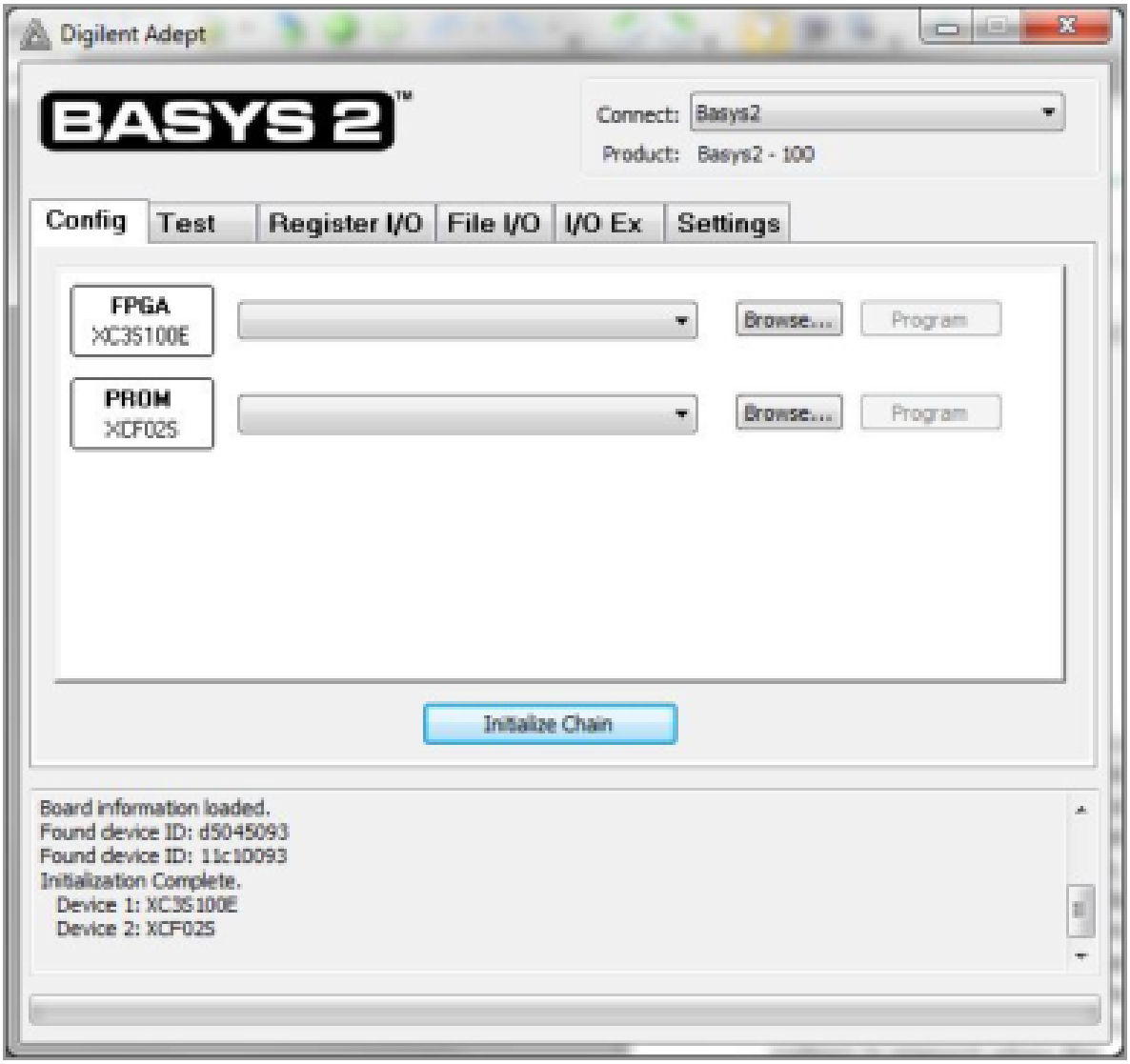


Figure 27: Adept 2 Window

27 Selecting '.bit' File

- Now select the ".bit" file inside your project directory which is defined when you are creating the project.
- now press Program button and program will be uploaded to your fpga board.

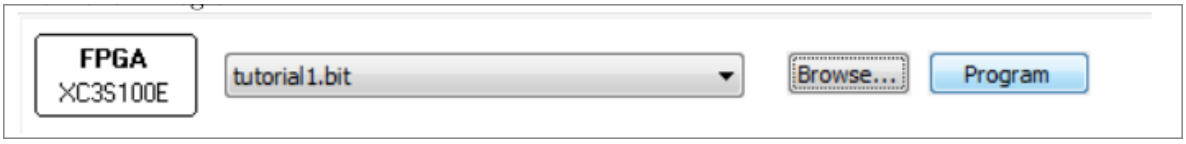


Figure 28: Selecting '.bit' File

28 CONGRATULATIONS!!!!

- Now play with buttons and switches so you can see how our program is working in real life.

29 Let's Do Same Thing With Verilog

- Most of the steps are same with minor differences.
- form New Project Wizard we need to select verilog language like shown below.

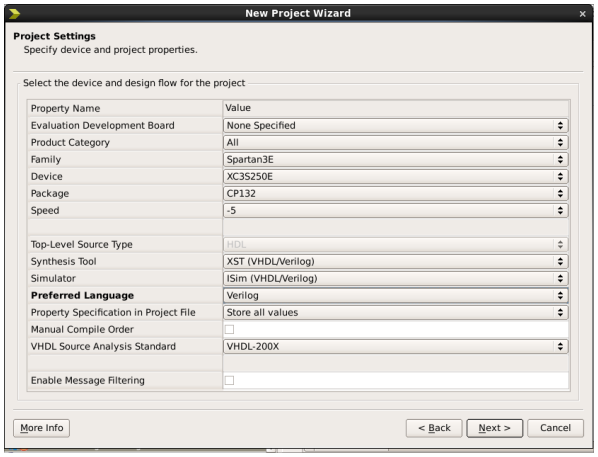


Figure 29: Creating Verilog Project

30 Adding New Source

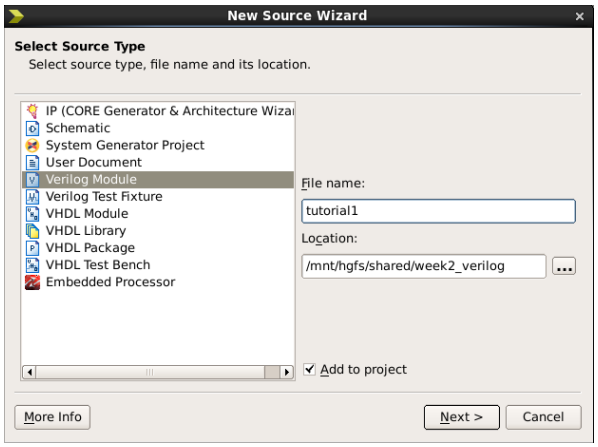


Figure 30: Select Verilog Source

31 Defining Ports (Actualy Not)

- This time we are not going to define ports here, we are going to define port by coding.
- but you can always use this option.

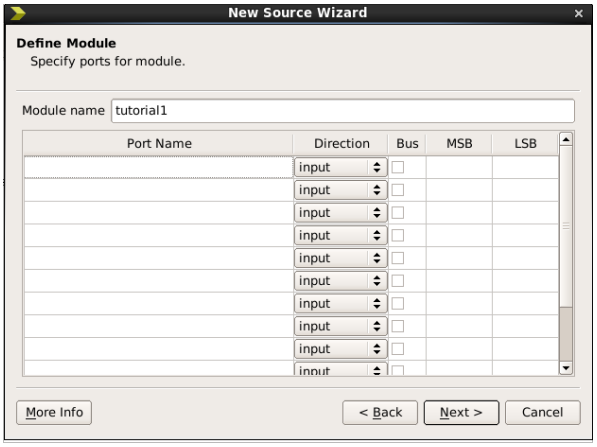


Figure 31: No Ports Defined

32 Writing Verilog Code

- You can see that we defined ports in the code.
- Comment lines describes the code. Feel free to read all lines and understand.

```
1  module tutorial1( counterValue, reset, clock, enable);
2  input enable; // Increment our counter on rising edge of 'count'
3  input reset; // Reset our counter on rising edge of 'reset'
4  input clock; // Synchronize outputs to clock
5  output counterValue; // Internally store the counter value
6  reg [2:0] counterValue;
7  // This takes care of synchronously driving the outputs
8  always @ ( posedge clock) begin //at the positive edge of the clock
9     if( reset == 1'b1 ) begin //if reset is high
10    counterValue <= 3'b000; //then set the counter to 0000
11    end
12    else if( enable == 1'b0) begin //if enable is low and reset is high
13    counterValue <= counterValue + 1'b1; //count up
14    end
15    else if( enable == 1'b1) begin //if enable is high and reset is high
16    counterValue <= counterValue - 1'b1; //count down
17    end
18 end
19 endmodule
```

Figure 32: Verilog Code

33 Simulating Code

- This time we are using 'verilog test fixture' file type for writing simulation code.
- As same as before write the simulation code inside our test file.

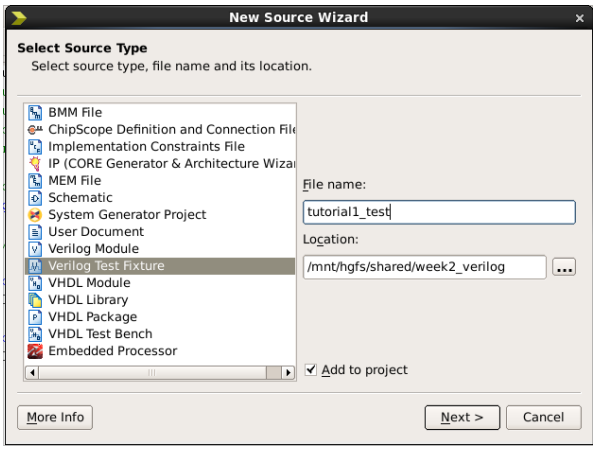


Figure 33: Create Test bench

```
43    initial clock = 0; // set clock to 0
44    always #20 clock = ~clock; //generating clock
45    initial begin
46       // Initialize Inputs
47       reset = 1; // set reset to 1
48       enable = 0; // set enable to 0
49       #100; //wait 5 clock pulses
50       reset = 0; //set reset to 0 so program can start to count
51       #200; //wait 10 clock pulses
52       enable = 1; // set enable to 1
53       #200; //wait 10 clock pulses
54    end
```

Figure 34: Verilog Simulation code

- Once we run the simulation we can see similar result as before.

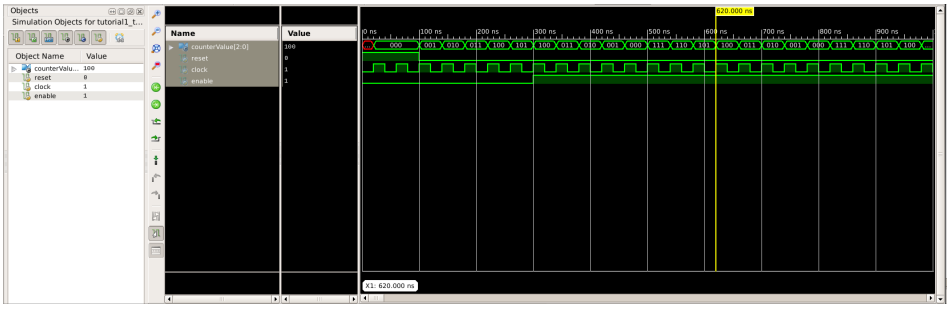


Figure 35: Verilog ISIM output

34 Creating .ucf File

- do the same steps from section 23 and 24.

35 Uploading The Code

- Do the same steps from section 25 to 28.

36 Codes Can Be Found at My Github

https://github.com/DALLI-KAKTUS/IKC-BME415-Lecture-Notes/tree/main