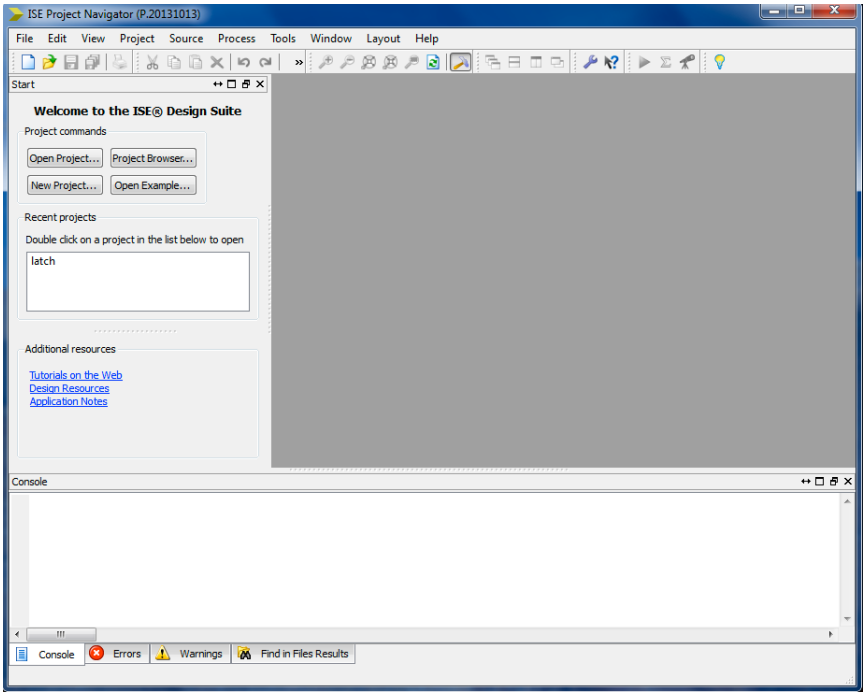
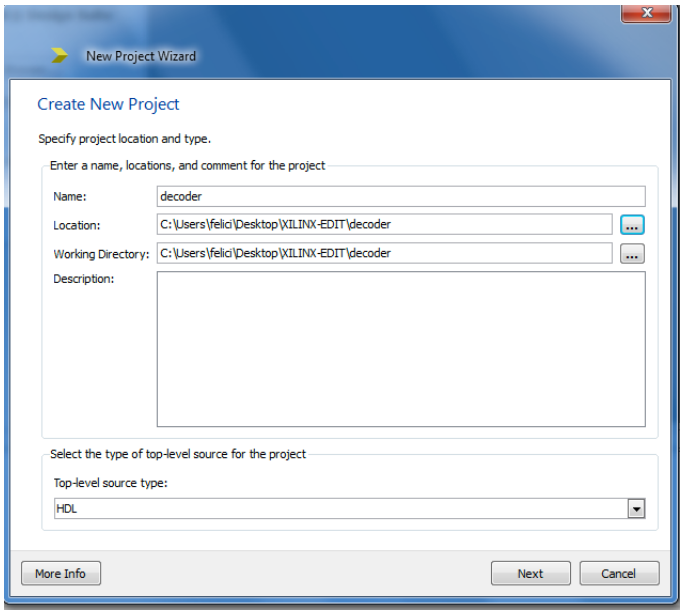
XILINX ISE AND SPARTAN 3AN TUTORIAL

SYNTETIZE AND SIMULATION----------

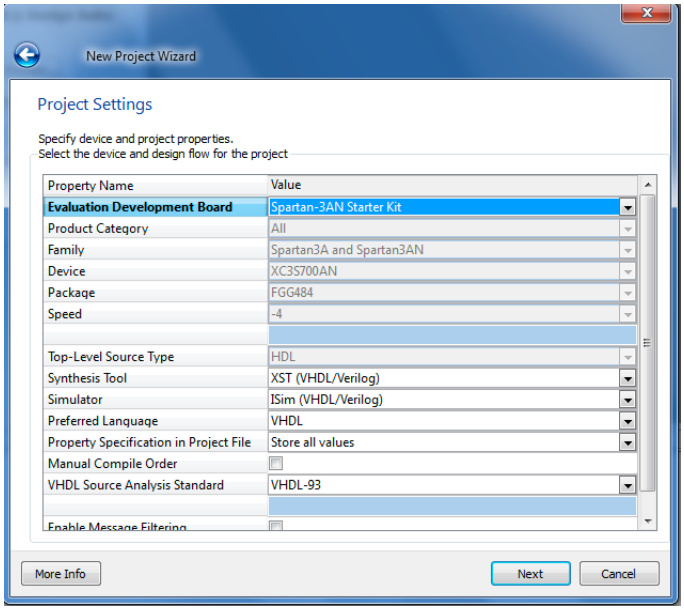
This tutorial will show you how to create a simple Xilinx ISE project based on the Spartan-3 Board. It will be implemented a simple decoder circuit that uses the switches on the board as inputs and the eight LEDs as outputs. There are several parts in this tutorial. Part 1 shows the basics of creating a simulating a project in Xilinx ISE. Part 2 shows how to generate the bit file that will be used to program the SPARTAN 3AN/3A BOARD FPGA internal memory. Part 3 shows how to program the board.



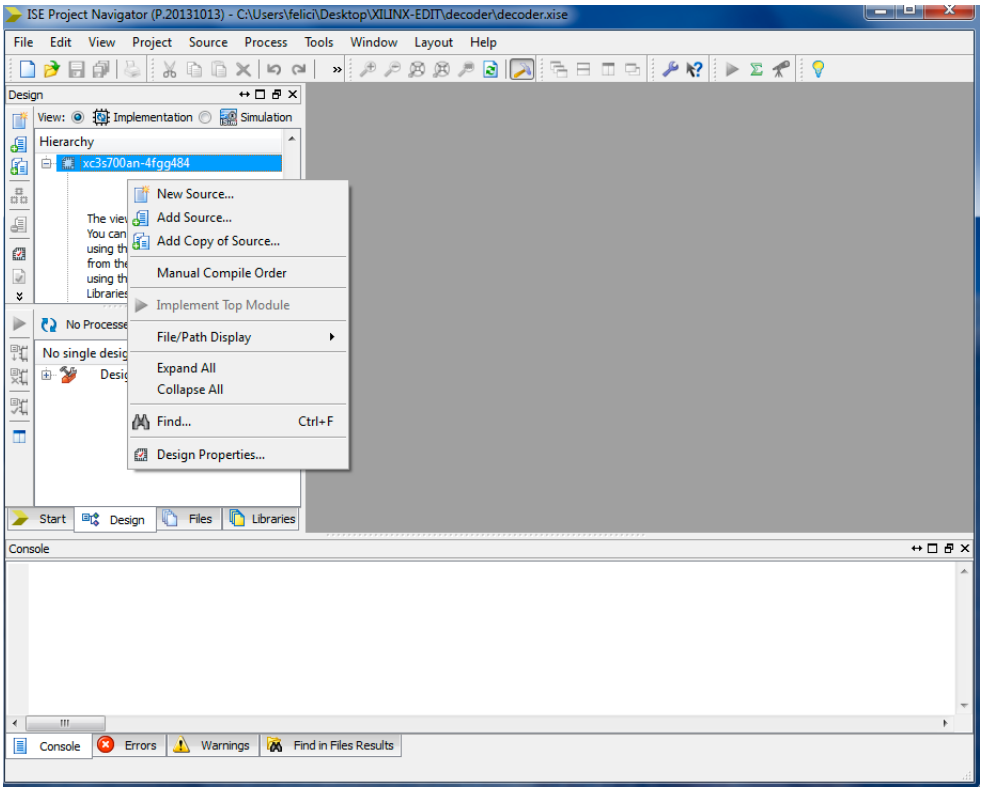
Select **File → New Project**



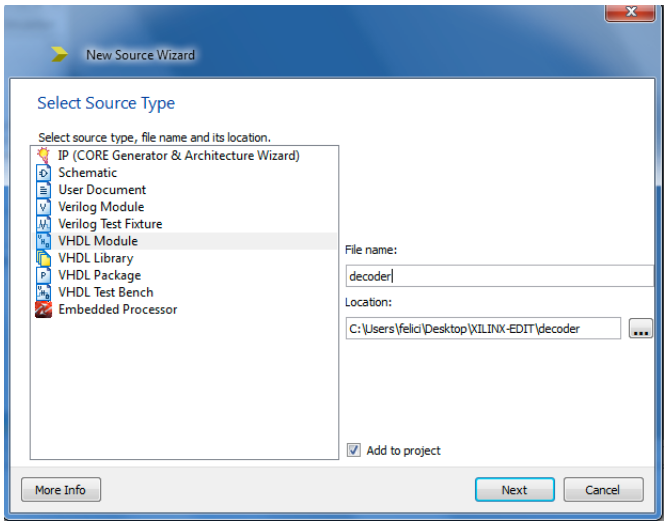
Select a project location and name. For this tutorial we will name the project "decoder".
Click **Next** and select the **Spartan 3AN Starter** Kit (or the **Spartan 3A Kit**)in the Evaluation Development Board category (check Preferred Language field: must be VHDL).



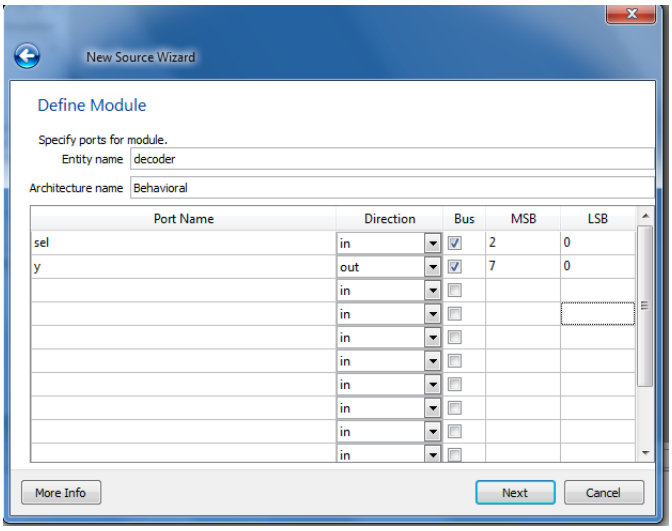
Click **Next** and **Finish.**
Right-click and select **New Source**.



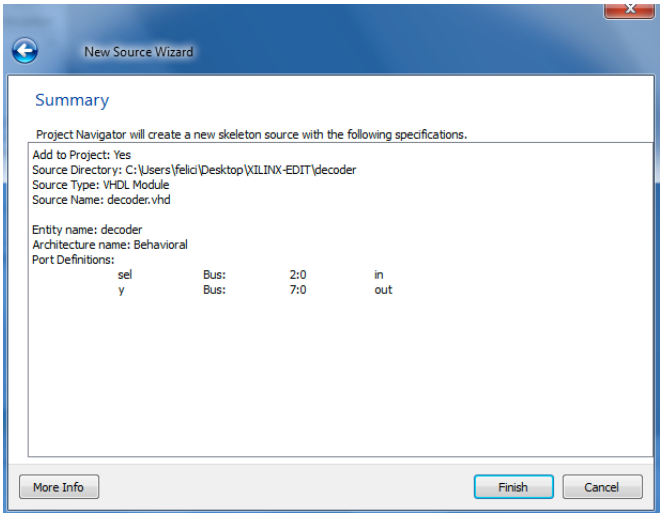
Select **VHDL Module** and assign **"decoder"** name.



You can now specify the inputs and outputs for the decoder. These will be inserted into an automatically generated template of the VHDL file. We have one 3-bit input ("sel") and one 8- bit output ("y"):



Click **Next**. A summary window will be shown.

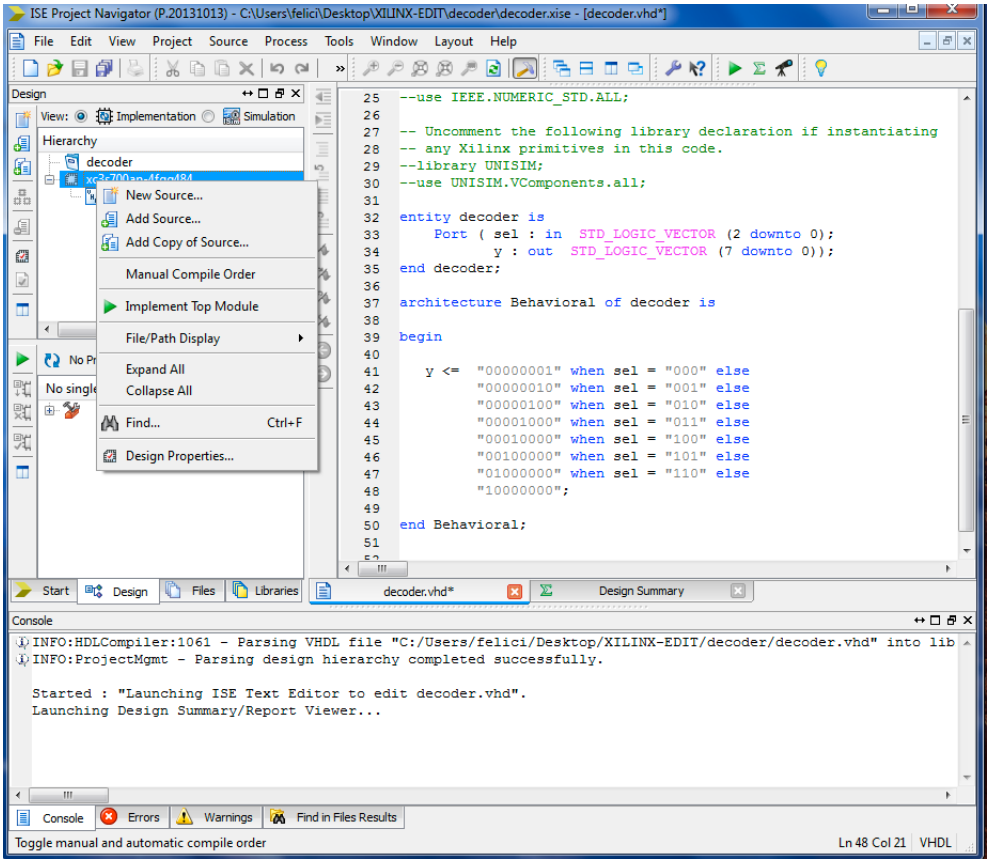


Click **Finish** and decoder.vhd will be shown in the top-left sources and on the window editor.

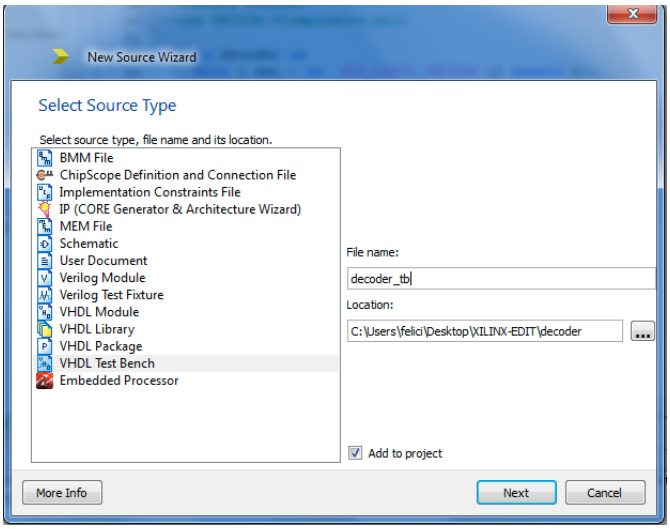
You are required to describe the behavior of the decoder using statements in the architecture body. In this example we will use conditional signal assignment statements; insert the code between **begin** and **end**.

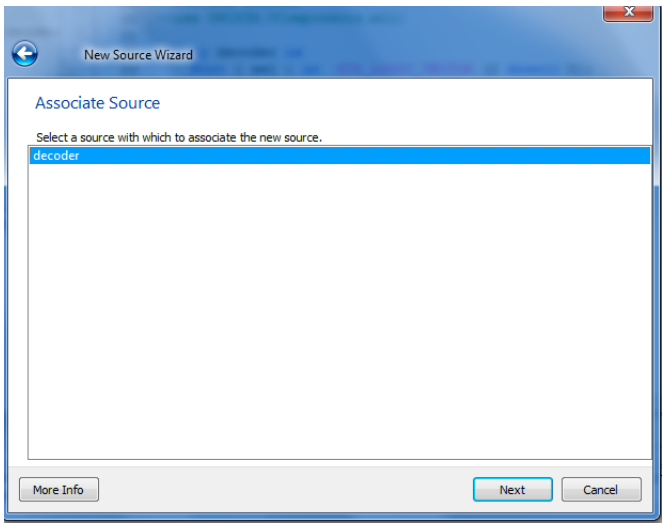
```
25  --use IEEE.NUMERIC_STD.ALL;
26
27  -- Uncomment the following library declaration if instantiating
28  -- any Xilinx primitives in this code.
29  --library UNISIM;
30  --use UNISIM.VComponents.all;
31
32  entity decoder is
33      Port ( sel : in  STD_LOGIC_VECTOR (2 downto 0);
34             y : out  STD_LOGIC_VECTOR (7 downto 0));
35  end decoder;
36
37  architecture Behavioral of decoder is
38
39  begin
40
41     y <=  "00000001" when sel = "000" else
42           "00000010" when sel = "001" else
43           "00000100" when sel = "010" else
44           "00001000" when sel = "011" else
45           "00010000" when sel = "100" else
46           "00100000" when sel = "101" else
47           "01000000" when sel = "110" else
48           "10000000";
49
50  end Behavioral;
51
```

Select again **New Source**



Select **VHDL Test Bench** and assign the name **decoder_tb**.
Selecting **Next** the "Associate Source" panel will be shown. Click **Next** and **Finish**.
The "decoder_tb.vhd" is shown in the editor panel.





Look at the file produced. Insert the instructions after the comment
-- insert stimulus here.

```
76
77
78      -- Stimulus process
79      stim_proc: process
80      begin
81         -- hold reset state for 100 ns.
82         wait for 100 ns;
83         -- insert stimulus here
84         wait for clk_period*5;
85         sel <= "000"
86         wait for clk_period*5;
87         sel <= "001"
88         wait for clk_period*5;
89         sel <= "010"
90         wait for clk_period*5;
91         sel <= "100"
92         wait for clk_period*5;
93         sel <= "101"
94         wait for clk_period*5;
95         sel <= "110"
96         wait for clk_period*5;
97         sel <= "111"
98
99
100        wait;
101     end process;
```

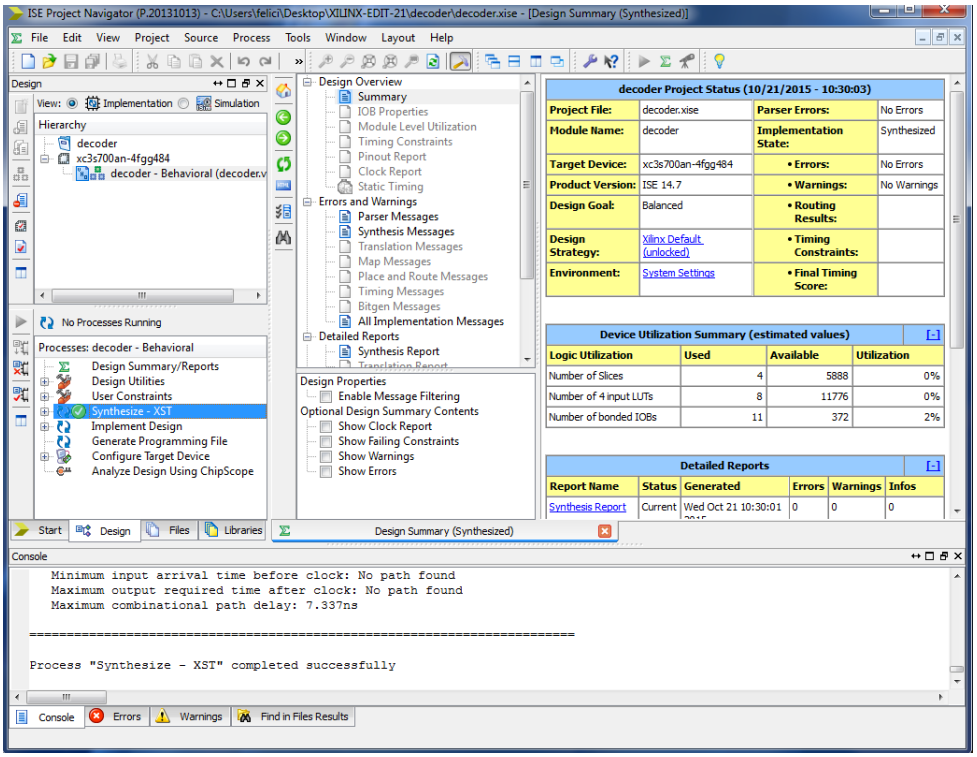
Modify also the **clk** sections of code as shown

```
--Inputs
signal sel : std_logic_vector(2 downto 0) := (others => '0');

--Outputs
signal y : std_logic_vector(7 downto 0);
-- No clocks detected in port list. Replace <clock> below with
-- appropriate port name

constant clk_period : time := 10 ns;
signal clk: std_logic;
```

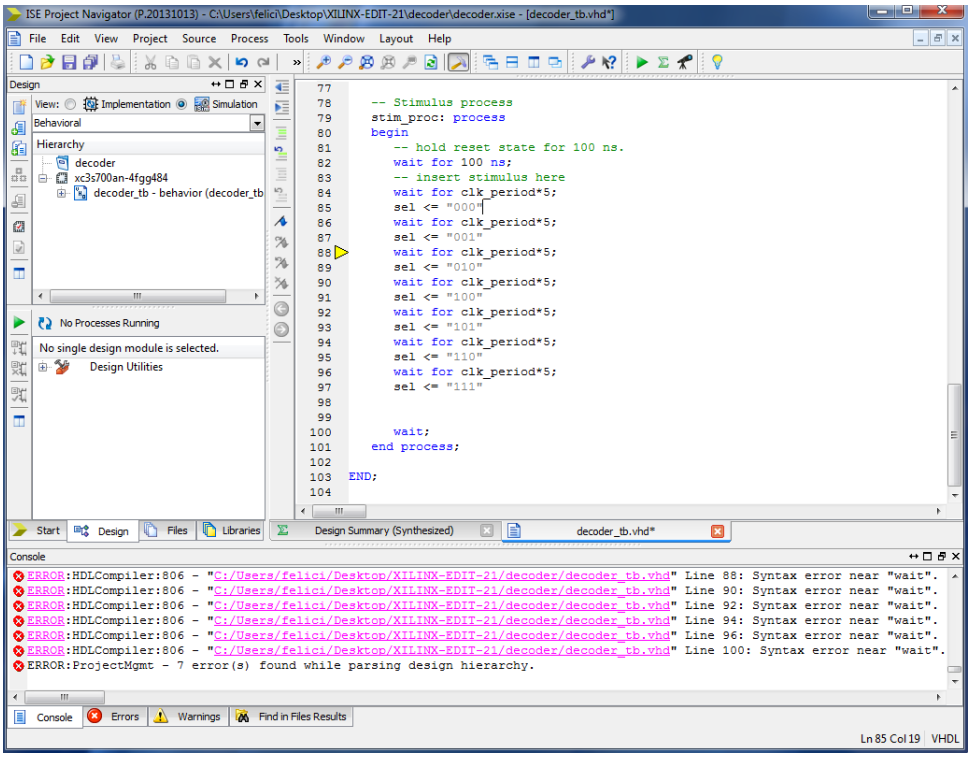
```
BEGIN

   -- Instantiate the Unit Under Test (UUT)
   uut: decoder PORT MAP (
          sel => sel,
          y => y
        );


--    -- Clock process definitions
   clk_process :process
   begin
      clk <= '0';
      wait for clk_period/2;
      clk <= '1';
      wait for clk_period/2;
   end process;
```

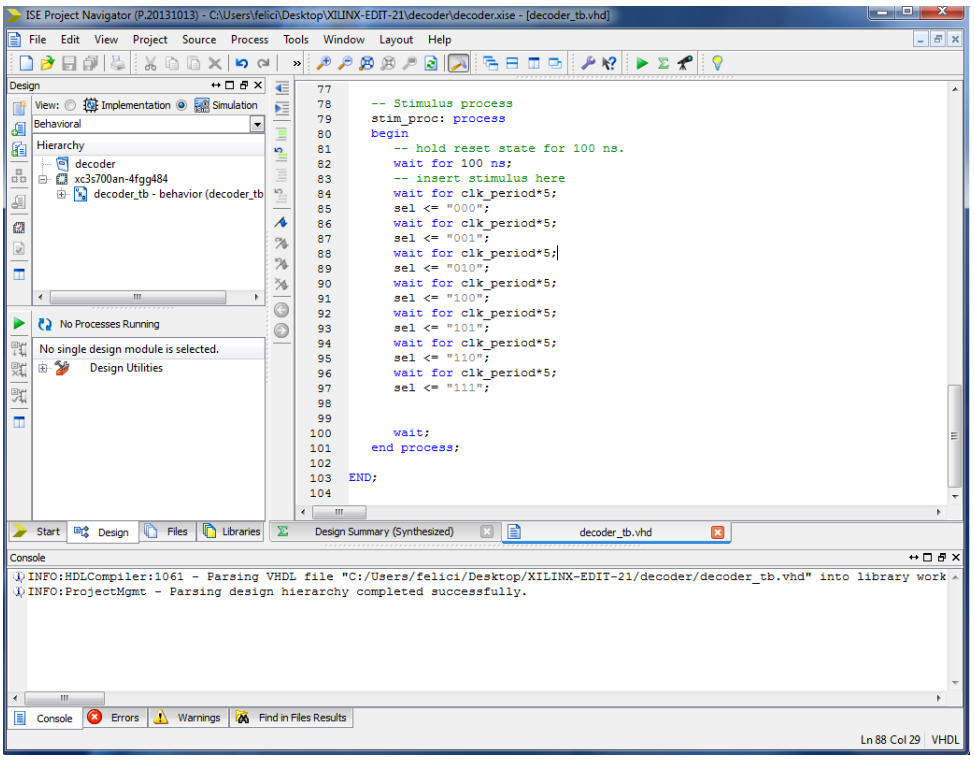
or remove all the **clk** sections and change the “wait for clk_period*5;” to “wait for 50 ns”.

Check that **Implementation** is selected in the **Project Manager View** panel that and double-click **Syntetize-XST.**

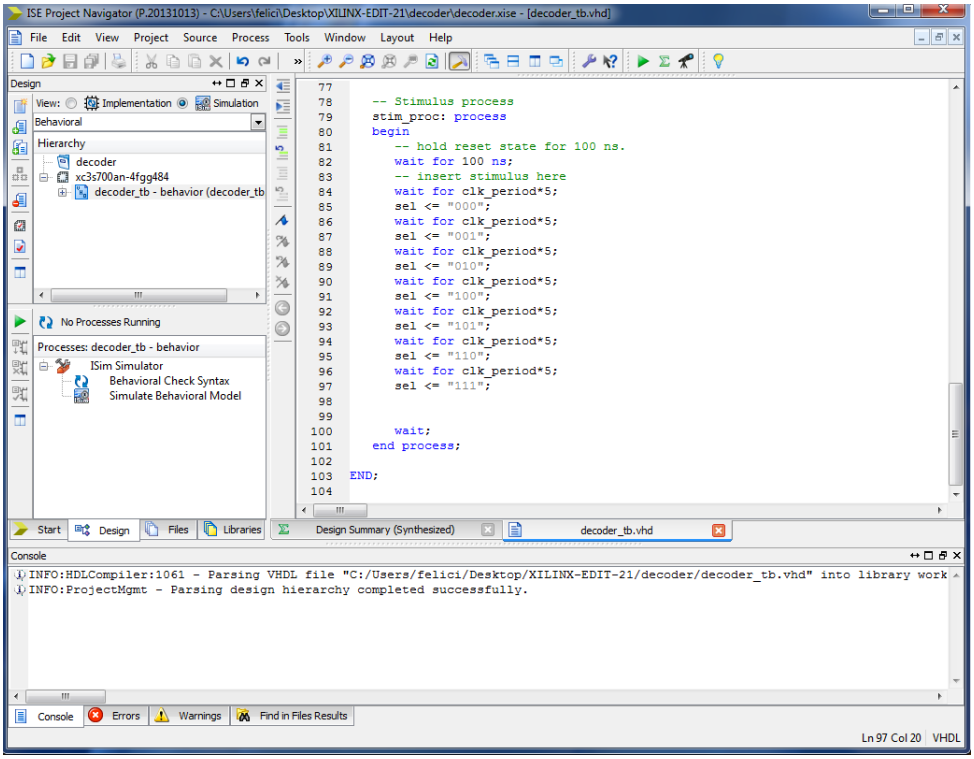


Select **Simulate** in the **Project Manager View**. Some errors will be shown in the console. Correct them (add semicolon after **sel** statement) and save the project. The errors should disappear.

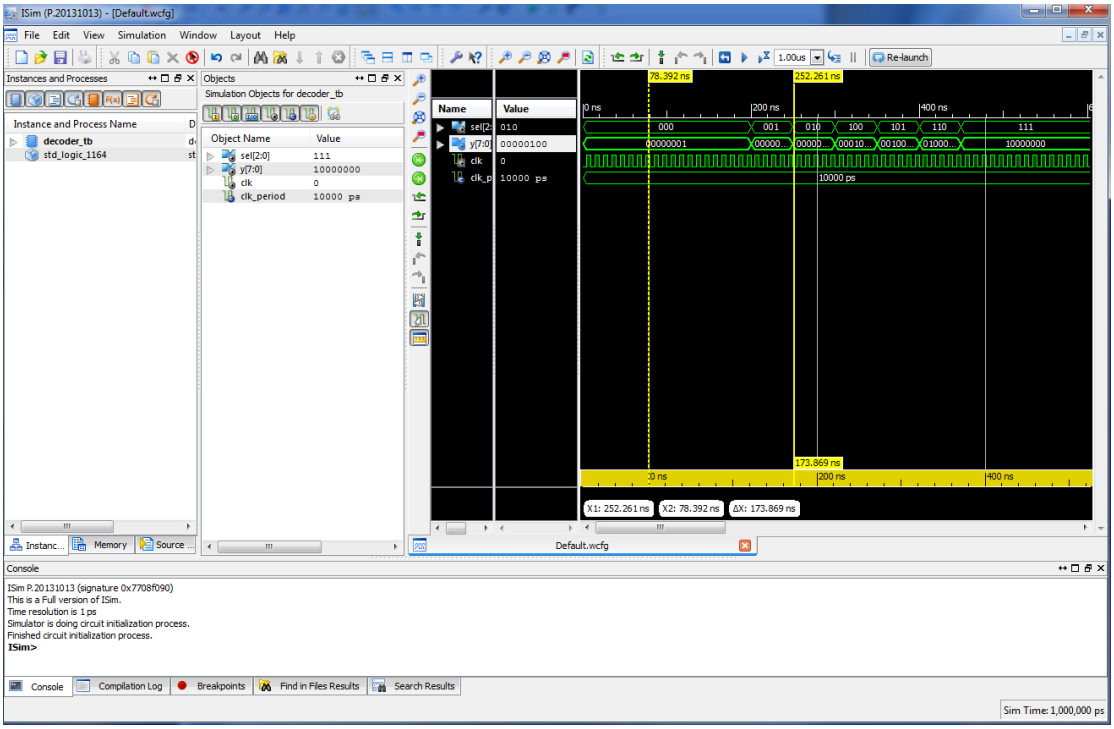




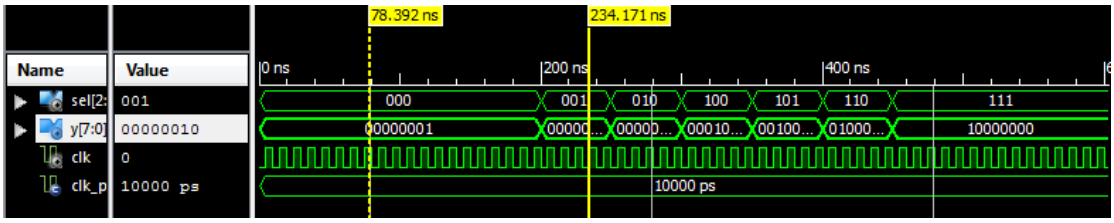
Select “decoder_tb” in the hierarchy window and expand **ISim Simulator**



Double-click Simulate Behavioral Model.

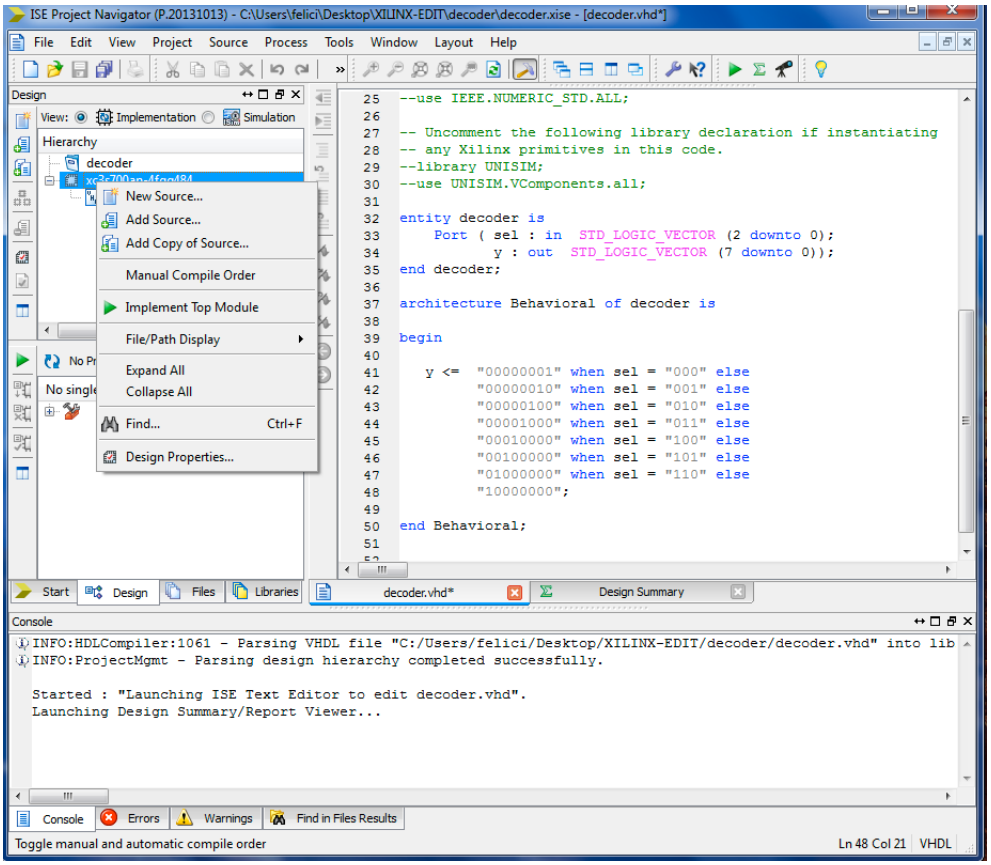


Simulation window will show the result of simulation.

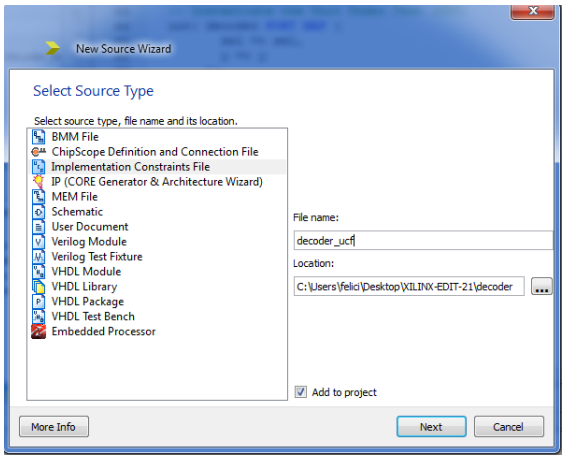


PROJECT IMPLEMENTATION----------

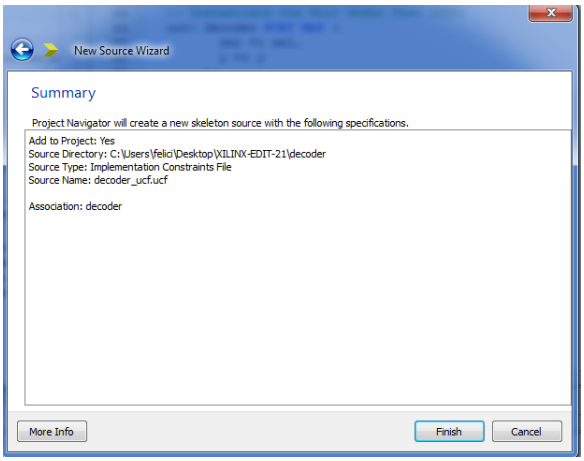
Again select **New Source**



select **Implementation Constraint Files** (you will have to check the Spartan 3AN specifications to find out UCF constraints for **slide switches** and **leds**) and assign the name "decoder_ucf" to the file, click next



A summary window shows up, click finish.



Now we have to map our signals to the board input/output. From the board user guide

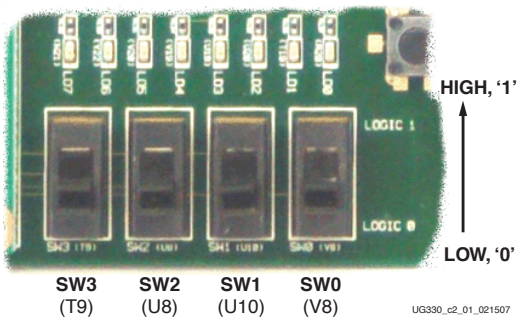


Figure 2-1: **Four Slide Switches**

we get the map of the UCF constraint file for the Slide Switches and leds.

```
NET "SW<0>" LOC = "V8" | IOSTANDARD = LVCMOS33 ;
NET "SW<1>" LOC = "U10"| IOSTANDARD = LVCMOS33 ;
NET "SW<2>" LOC = "U8" | IOSTANDARD = LVCMOS33 ;
NET "SW<3>" LOC = "T9" | IOSTANDARD = LVCMOS33 ;
```

Figure 2-2: **UCF Constraints for Slide Switches**

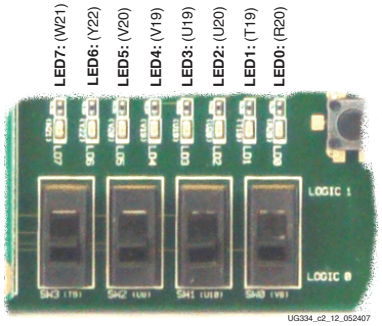


Figure 2-12: **Eight Discrete LEDs**

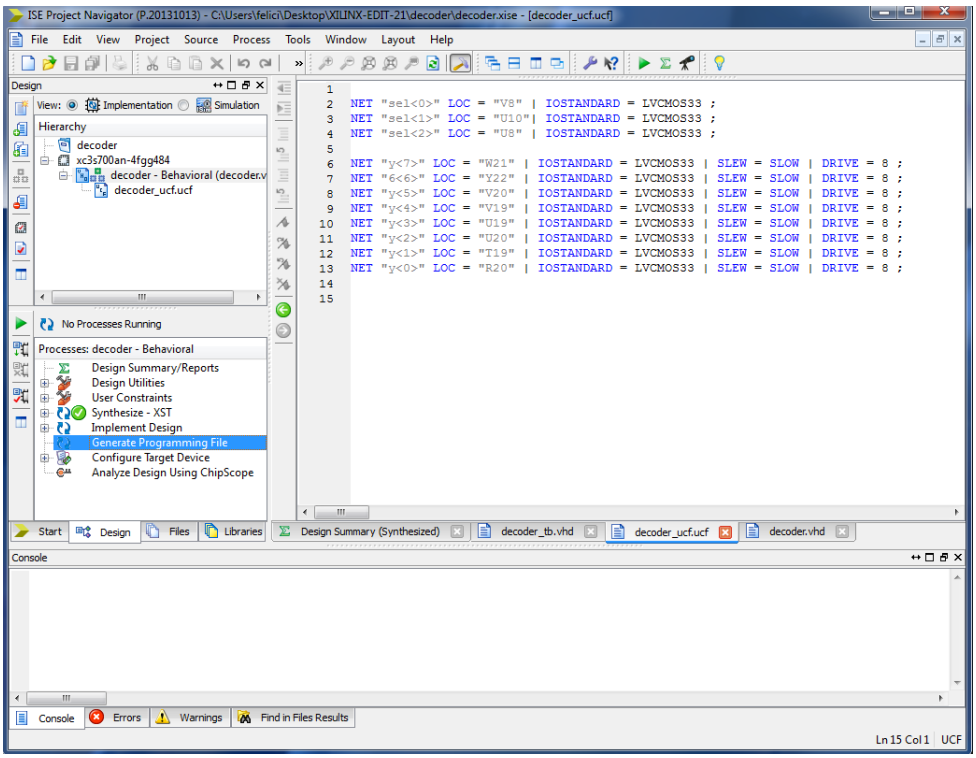
```
NET "LED<7>" LOC = "W21" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
NET "LED<6>" LOC = "Y22" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
NET "LED<5>" LOC = "V20" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
NET "LED<4>" LOC = "V19" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
NET "LED<3>" LOC = "U19" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
NET "LED<2>" LOC = "U20" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
NET "LED<1>" LOC = "T19" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
NET "LED<0>" LOC = "R20" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
```

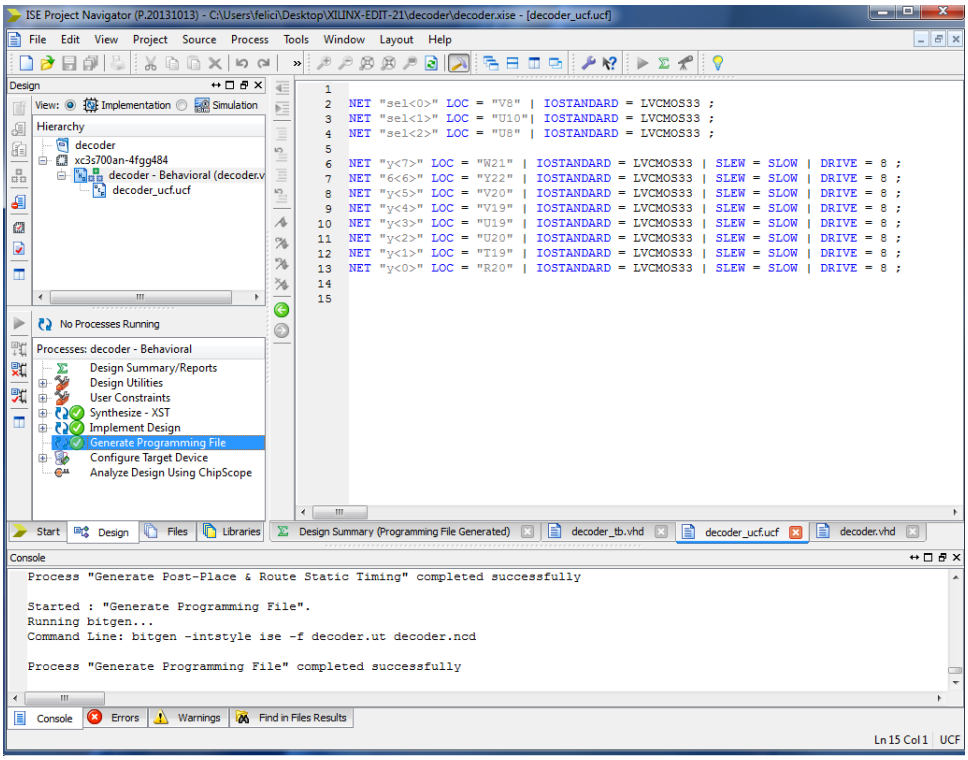
Figure 2-13: **UCF Constraints for Eight Discrete LEDs**

Add our constraint file assigning the nets of our project

```
NET "sel<0>" LOC = "V8" | IOSTANDARD = LVCMOS33 ;
NET "sel<1>" LOC = "U10"| IOSTANDARD = LVCMOS33 ;
NET "sel<2>" LOC = "U8" | IOSTANDARD = LVCMOS33 ;

NET "y<7>" LOC = "W21" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
NET "6<6>" LOC = "Y22" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
NET "y<5>" LOC = "V20" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
NET "y<4>" LOC = "V19" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
NET "y<3>" LOC = "U19" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
NET "y<2>" LOC = "U20" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
NET "y<1>" LOC = "T19" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
NET "y<0>" LOC = "R20" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
```

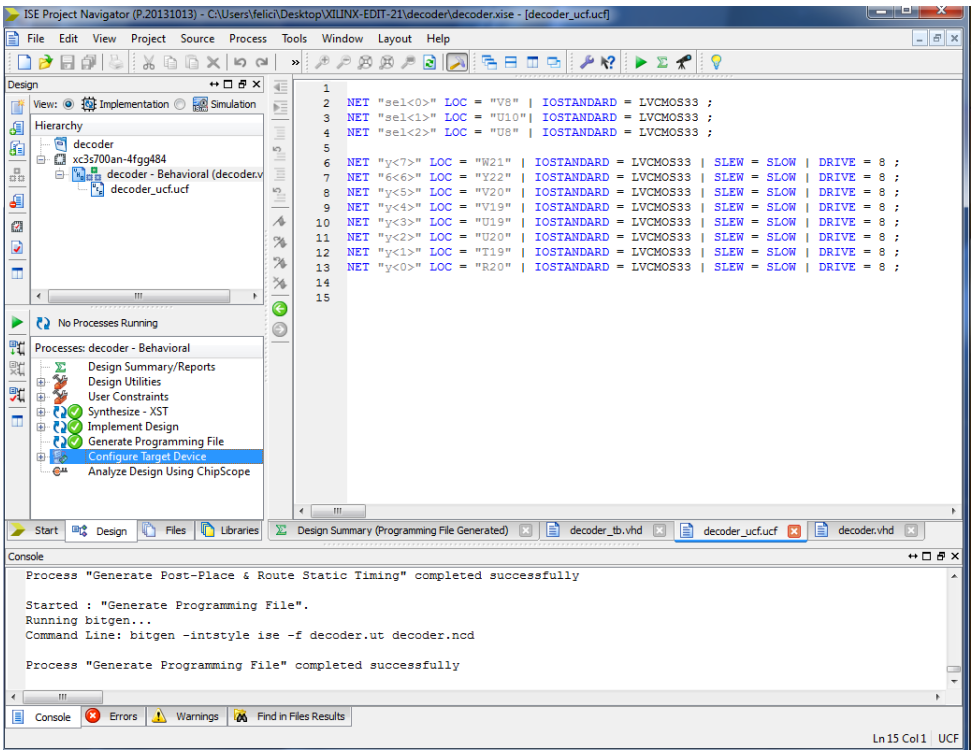
Now we're ready to implement our project. Double-click generate programming file



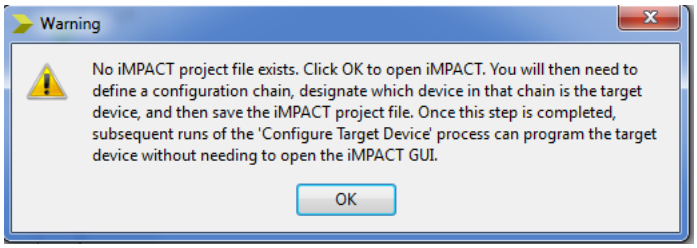


PROGRAM THE DEVICE----------

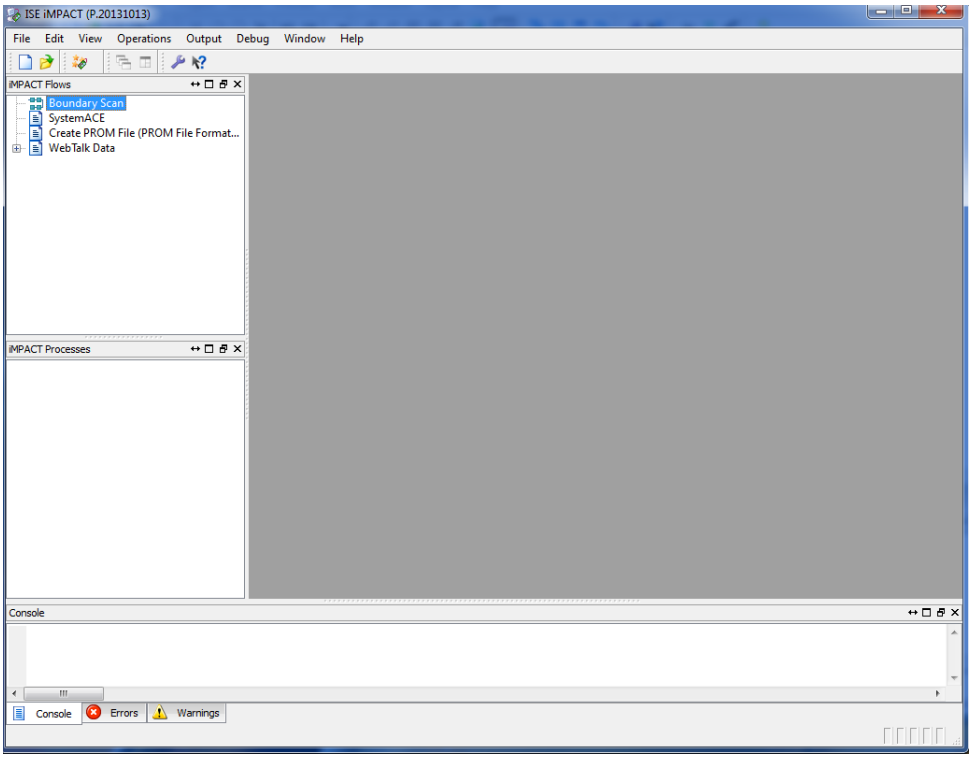
Double-click **Configure Target Device**

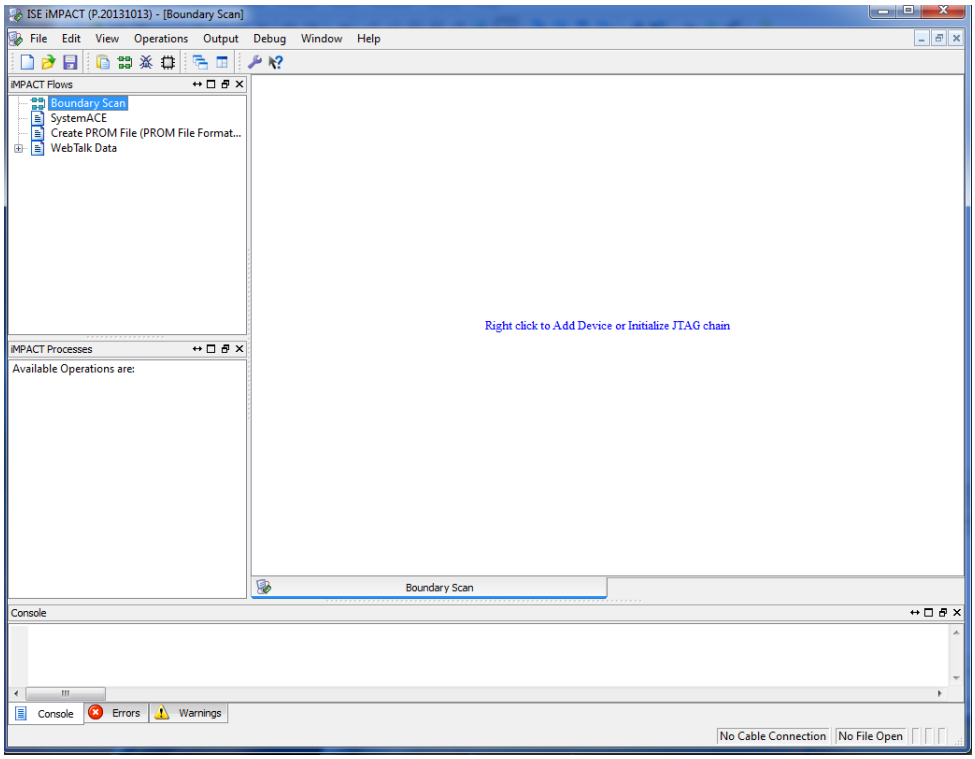


A warning window appear. Click **OK**

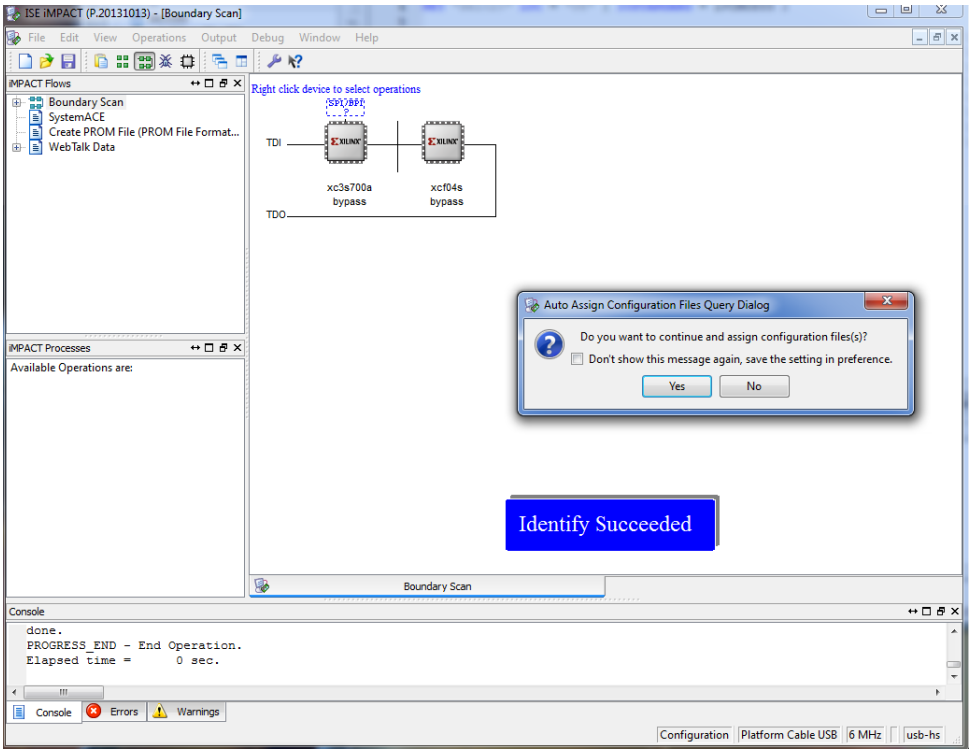


The **Impact** window shows up. Double-click **Boundary Scan** to detect connected devices and then initialize the chain.

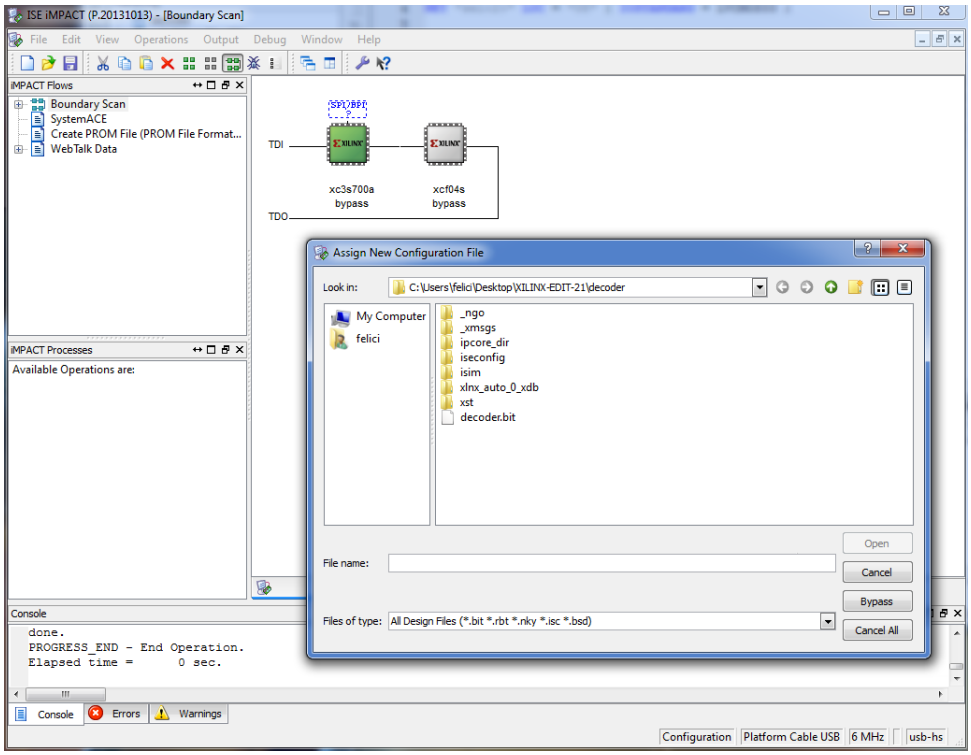




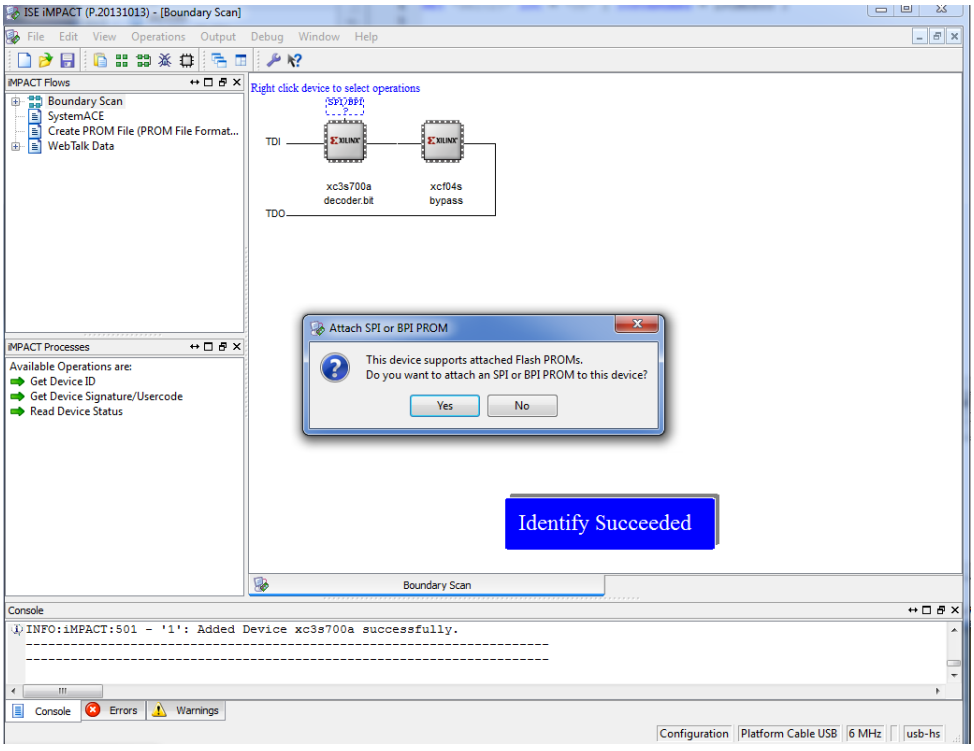
Two components are detected. The FPGA and The EEPROM. Click Yes to assign the files.



The FPGA is firstly selected together with a browser. Select the "decoder.bit" file and click **Open**.

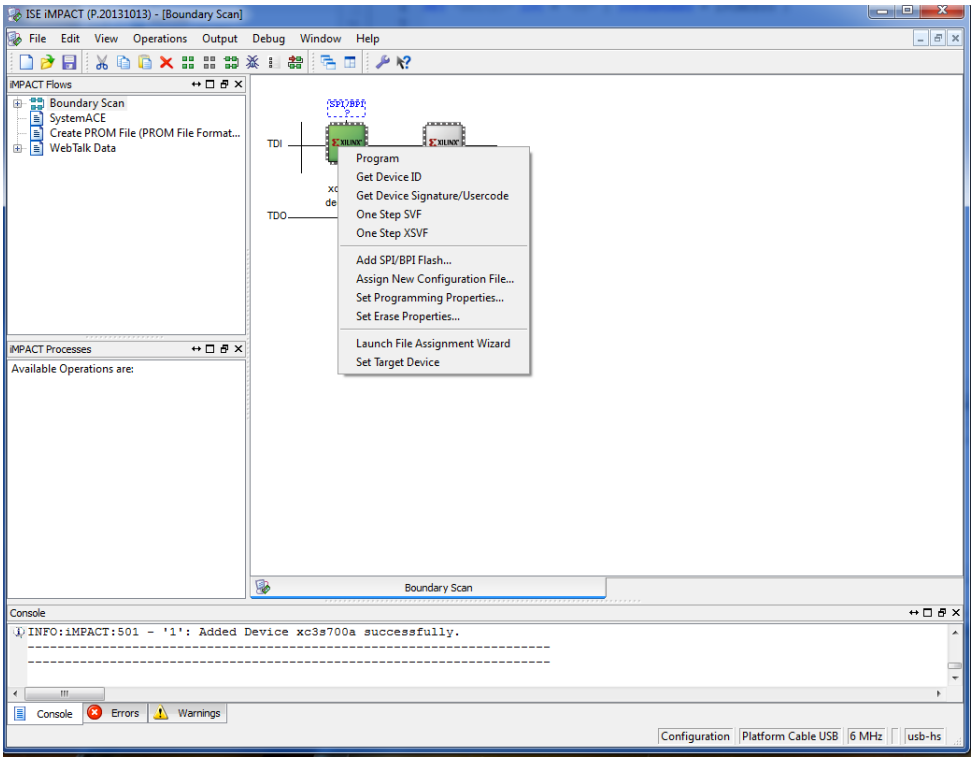


Click **No** in the next window and then **Bypass** (note that the decoder.bit file has been assigned to the FPGA)



A summary window appear. Click **OK**.

Finally we have to download the .bit file in the FPGA. Right-click the FPGA and select Program



A confirm window shows-up. Congratulation, you've programmed your first FPGA.