

## Tutorial 1

### Introduction to VHDL, and ISE 10.1 on the Digilent Spartan-3E Starter Kit board

## Introduction

These labs will be using what is known as a Field Programmable Gate Array (FPGA). The FPGA will be on a Digilent Spartan3E board. FPGAs are different than traditional integrated circuits, because interconnects between the gates are programmable, which means the **hardware** itself is configurable. This tutorial will be the first in a series of tutorials as shown in figure 1.

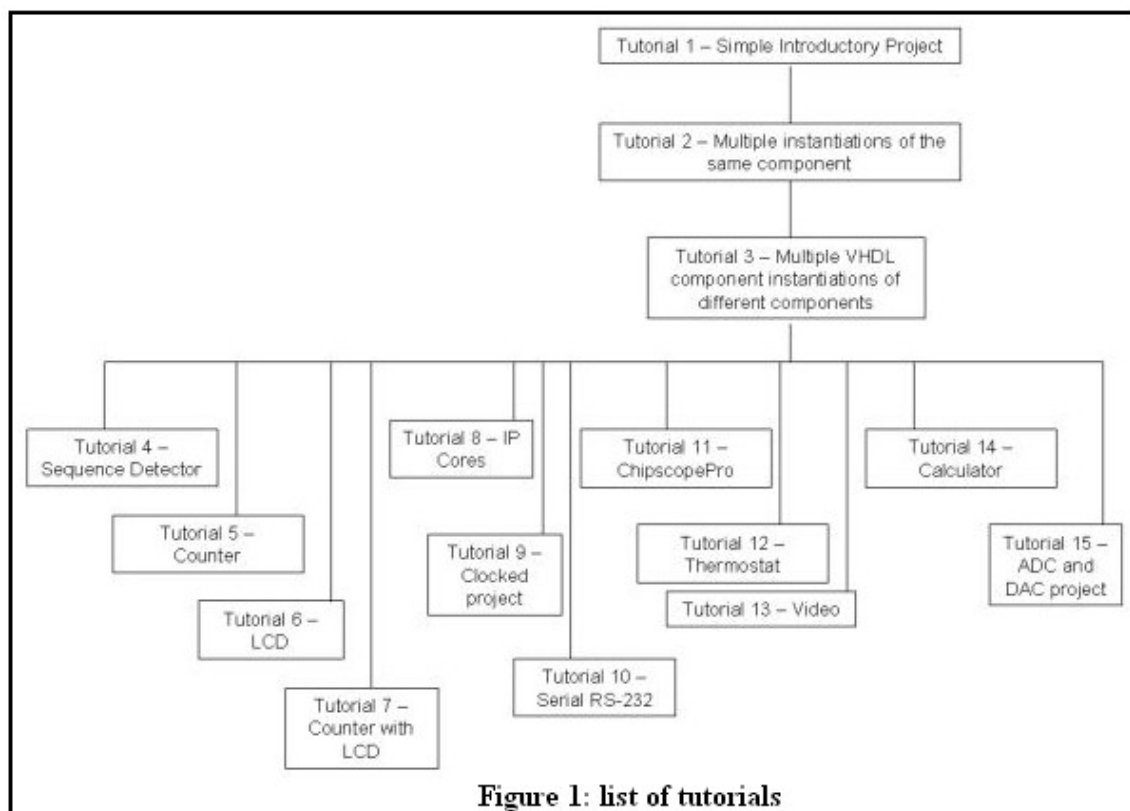


Figure 1: list of tutorials

It is assumed that the individual wishing to learn programmable logic will complete tutorials 1-3 in order. Once that series is complete then the individual could proceed to any of the remaining tutorials (4-15) in any order desired based on their individual needs.

This lab will be an introduction to design techniques for FPGAs using a hardware descriptive language (VHDL) design. Xilinx ISE 10.1 is the design tool provided by Xilinx for this purpose. Xilinx makes a free version of this tool called Webpack which would be virtually identical for this purpose. The board will be a Digilent Spartan3E board with a XC3S500E chip. Please avoid putting your fingers on the chips, as they are extremely electrostatic discharge sensitive.

There are four fundamental steps in all digital logic design. These consist of:

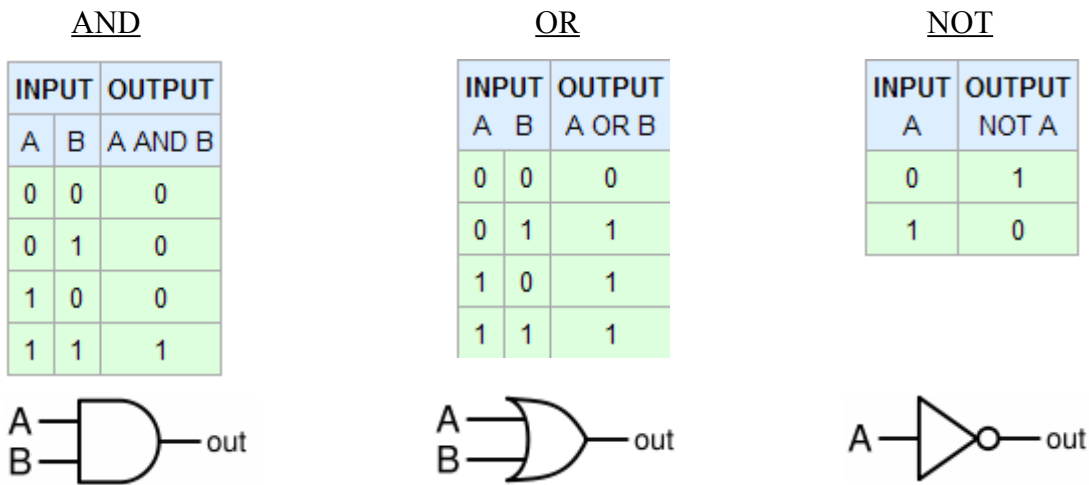
1. **Design** – The schematic or code that describes the circuit.
2. **Synthesis** – The intermediate conversion of human readable circuit description to FPGA code (EDIF) format. It involves syntax checking and combining of all the separate design files into a single file.
3. **Place & Route** – Where the layout of the circuit is finalized. This is the translation of the EDIF into logic gates on the FPGA.
4. **Program** – The FPGA is updated to reflect the design through the use of programming (.bit) files.

There are two sections of “design” for this lab. The first is the VHDL program design of what the project has to do, where the circuit in question will be implemented. The second step is the testbench simulation. As its name implies, it is used for testing the project by simulating the result of driving the inputs and observing the outputs to verify your design. The software package to be used is the Xilinx ISE 10.1. ISE has the capability of implementing a variety of different design methodologies including: Schematic Capture, Finite State Machines, and Hardware Descriptive Language (VHDL or Verilog). ISE also provides its own simulation tool to test your designs before setting them on the FPGA. For additional details on the various tools and their uses, view the video on the FMAC website ([www.fpgamac.com](http://www.fpgamac.com)).

## Logic Gates

In everyday activities, people use a number system with the digits zero through nine, for a total of ten digits, thus, base ten. In the digital world a number system called binary (base two) is used. There are only two digits, Zero (0) and One (1). In hardware, Zero is represented as a low voltage, and one as a high voltage.

In digital logic design, logic gates are used extensively. There are three fundamental logic gates, consisting of the following: AND, OR, NOT (also known as invert). They all work in similar but unique ways. The following are truth tables that list what the output will be with various inputs.



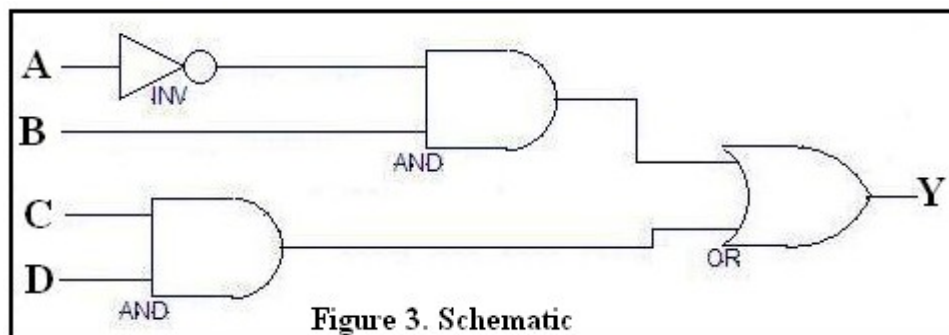
**Figure 2. Boolean Logic**

The AND gate is used to find if both input values are high (one). If the statement is true the gate will output a one and if false (one input being low) it will output a low (zero). An OR gate is used to see if either of the inputs are high and then will output a high if true and a low if false. The NOT gate is used to directly invert the input. If the input is high then the output will be low and vice versa. In normal digital logic, a high is a positive voltage and the low is ground.

## Objective

The objective is to understand the basics of digital logic and design. This will be accomplished through the implementation of the small circuit below in an FPGA. The circuit will be designed in such a way that only specific combinations of switches will turn on a light emitting diode (LED) on the board.

The Boolean expression for this circuit is  $Y=(\text{NOT}(A) \text{ AND } B) \text{ OR } (C \text{ AND } D)$ . The schematic can be seen in figure 3.



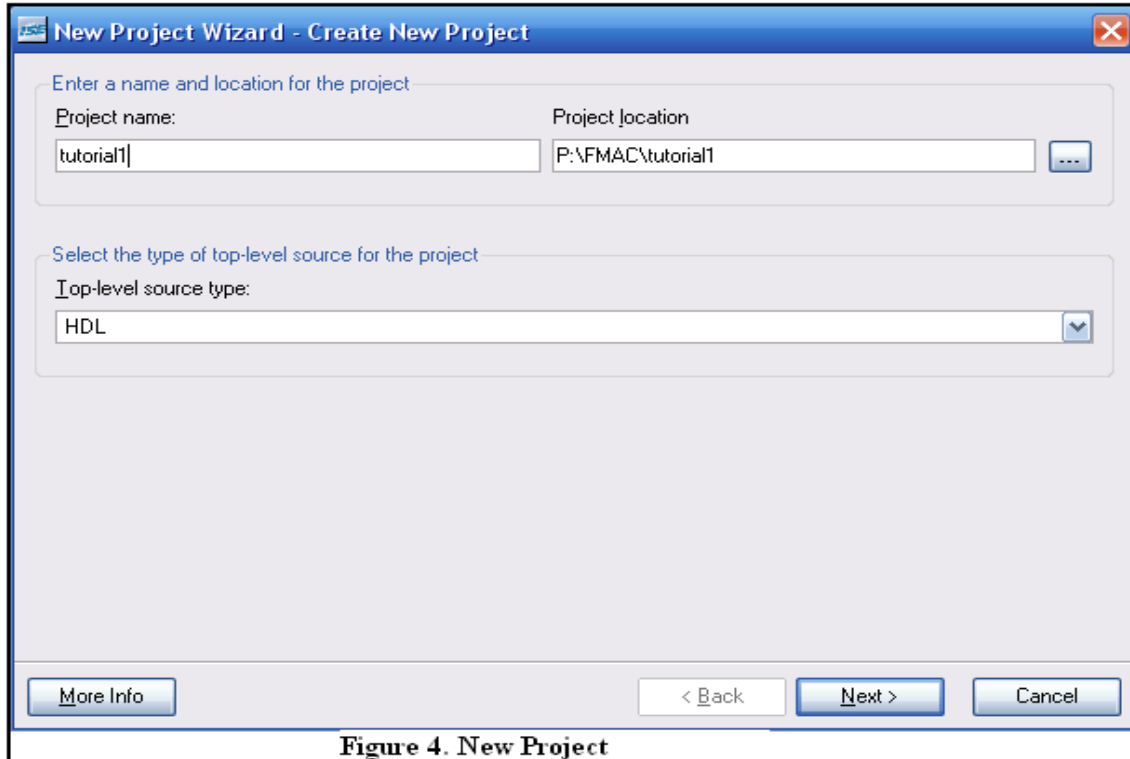
**Figure 3. Schematic**

## Process

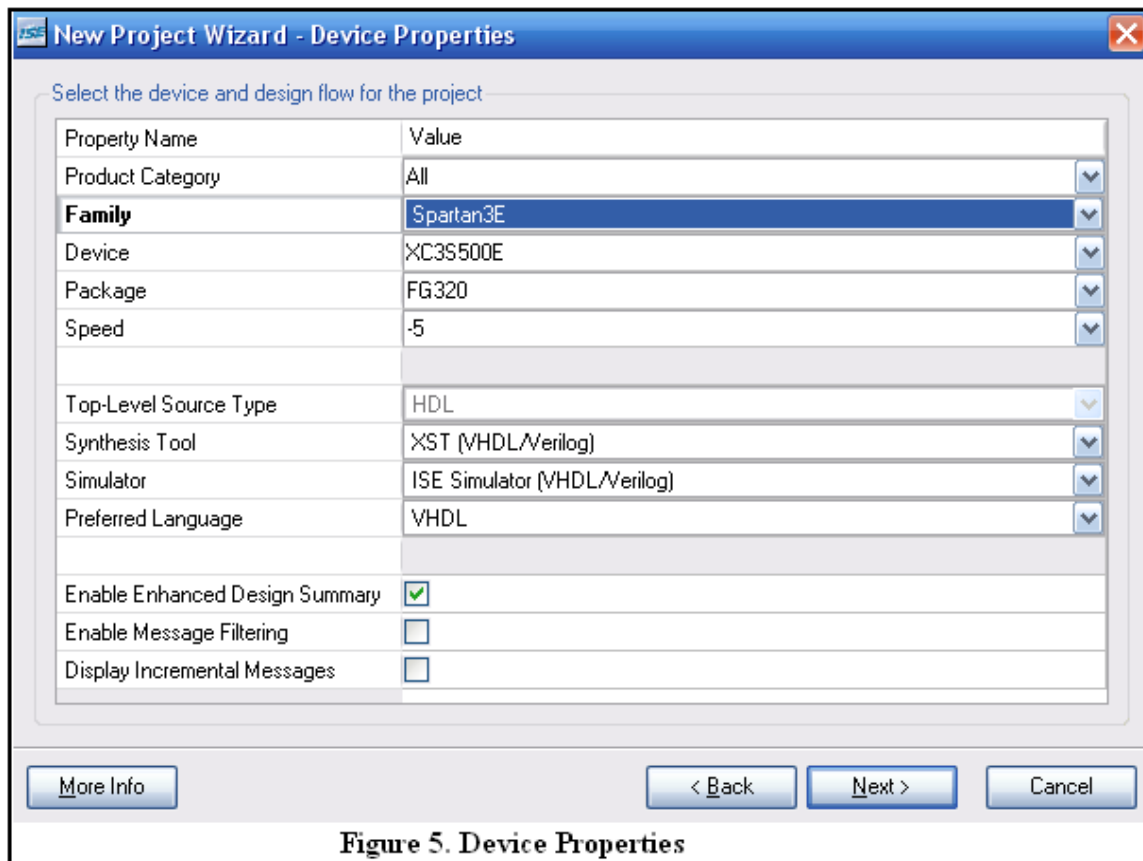
1. Analyze the VHDL.
2. Verify inputs and outputs with test bench waveform.
3. Program FPGA with circuit design.

## Implementation

1. Open the **Xilinx ISE Project Navigator** by selecting it from the start menu or from the desktop icon.
2. Within the project navigator, select from the File Menu, New Project. Name the project “tutorial1”. See figure 4. To put the project in a different location, either directly type the file path into Project Location or search by clicking the Three-dotted Browse button. The location of the project can always be seen on the top of the Project Navigator screen. Click next and it will show the device properties window.

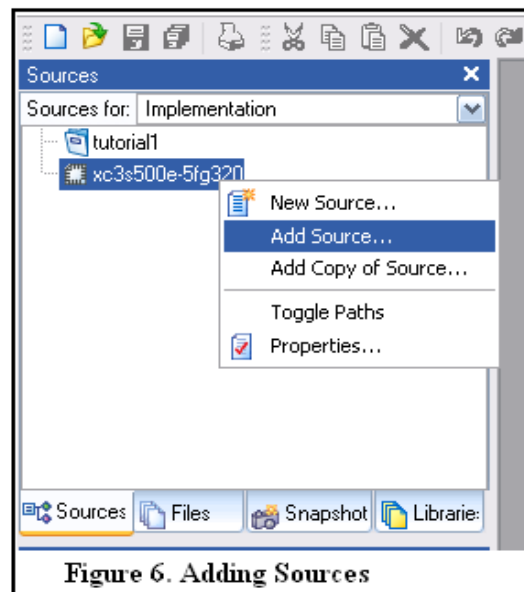


3. Set the device properties exactly as shown in figure 5 and continue to select Next until the project is created.

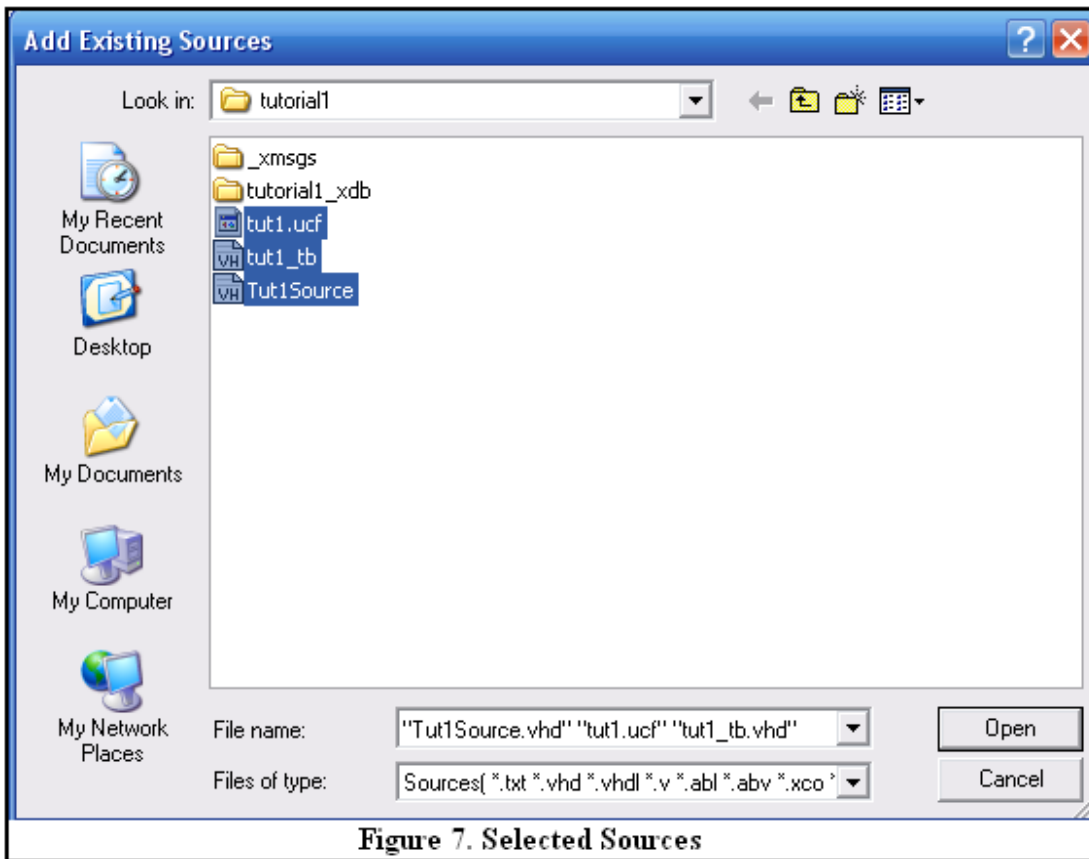


**Figure 5. Device Properties**

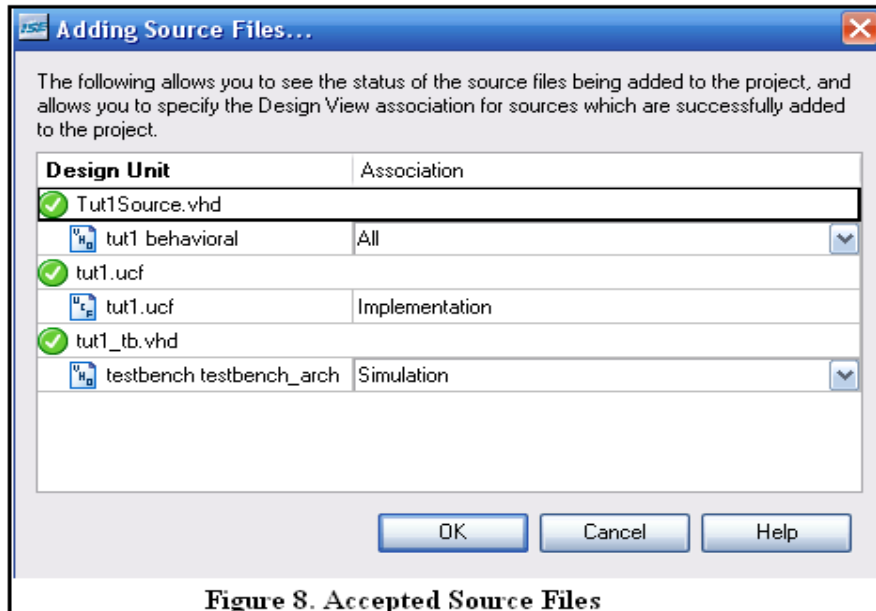
4. The project will be a blank project. For this project, there are three source files. Copy them into the directory for Tutorial 1. The files are: Tut1Source.vhd (the source file, tut1\_tb.vhd (the testbench) and tut1.ucf (the pin assignments). When naming testbenches, it is common practice to add the “\_tb” to the file name. Right click into the Sources pane and select Add Source as shown in figure 6.



5. Select the three source files shown in figure 7.



6. The green check marks, as seen in figure 8, identify that the program knows what type of files they are and what to do with them. The Tut1Source.vhd file can be used for Synthesis, Implementation and Simulation. The tut1.ucf constraint file can't be used for simulation. The tut1\_tb testbench can only be used for simulation.



7. Figure 9 shows that there are two major sections where most of the work is done. The windows are the Sources tab and the Processes tab. In the Sources tab, the tree structure is displayed. If you check on the “+” symbol you will see how the .ucf file falls below the source file.

8. By double-clicking on the Tut1Source.vhd file you can see the source file in the text editor screen. The source code is commented and somewhat easy to follow. In the sources tab, click on the down arrow and select Behavioral Simulation. See figure 10.

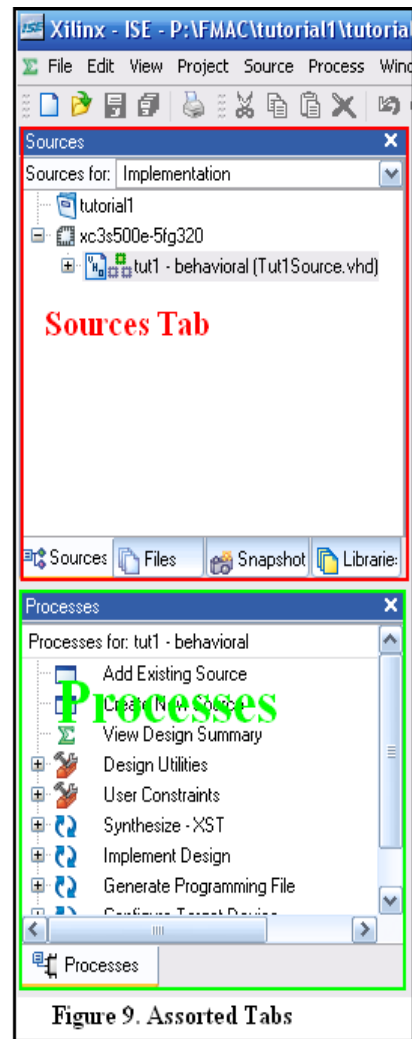


Figure 9. Assorted Tabs

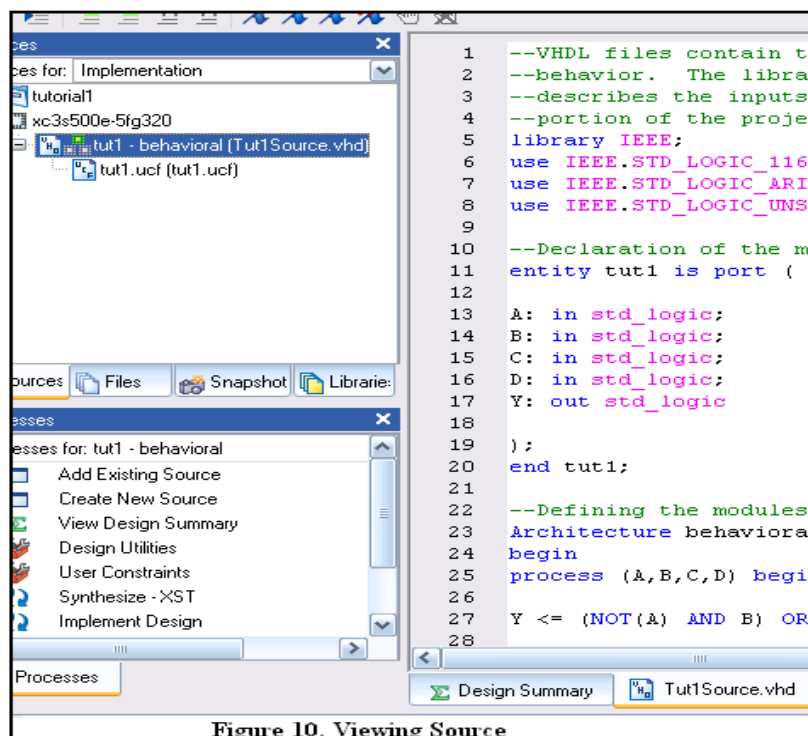


Figure 10. Viewing Source

- In the Sources window, the tree structure is again seen. This time, if you click on the “+” symbol, the source file is shown as a child to the testbench file. Highlight the testbench file and double-click on the “Simulate Behavioral Model”.

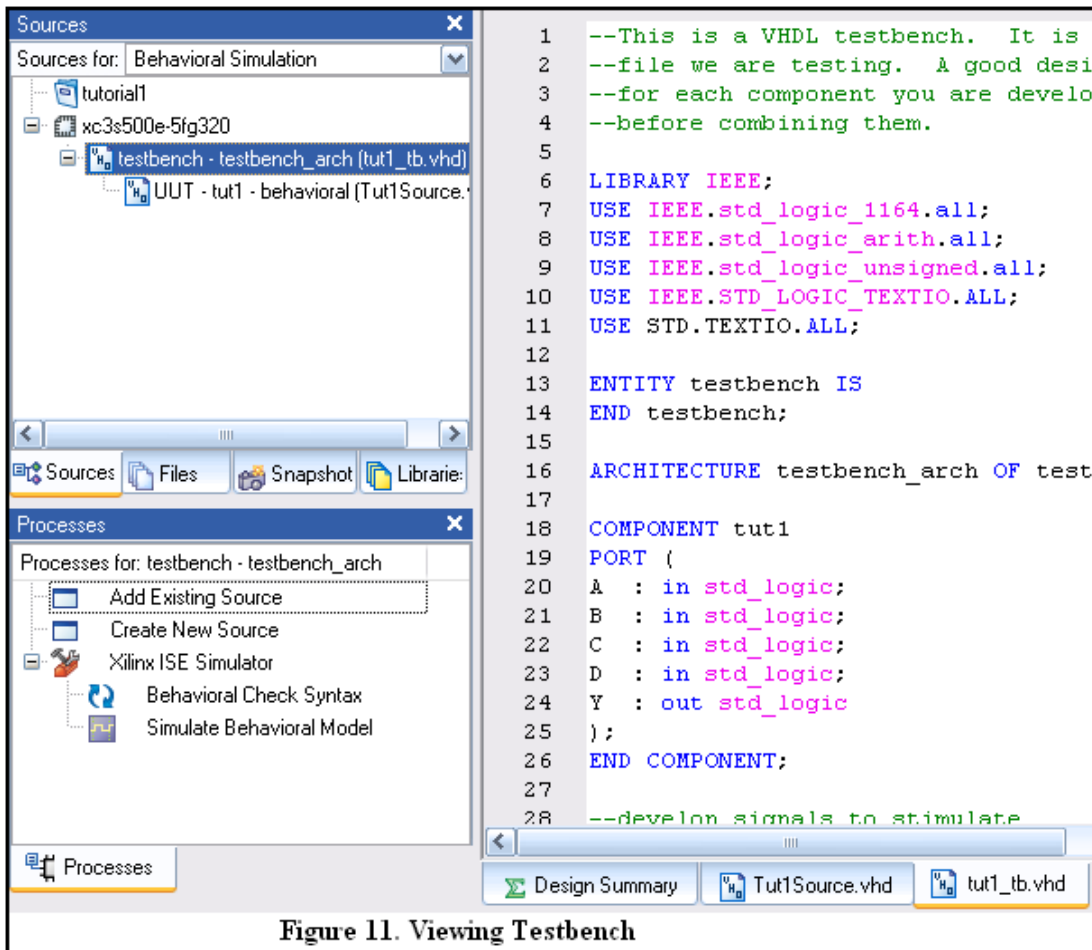


Figure 11. Viewing Testbench

- The Xilinx ISE will launch as shown in figure 12. You can zoom in by right clicking on the waveform and selecting zoom in. See figure 13.

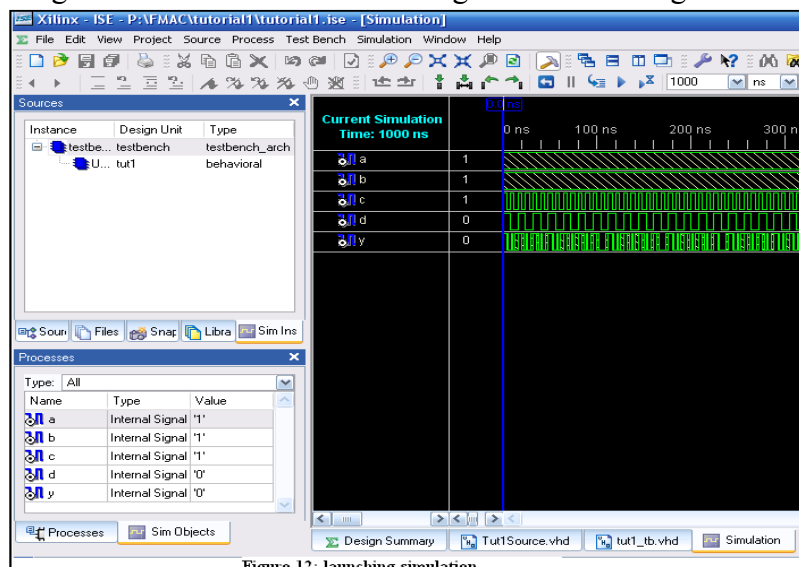
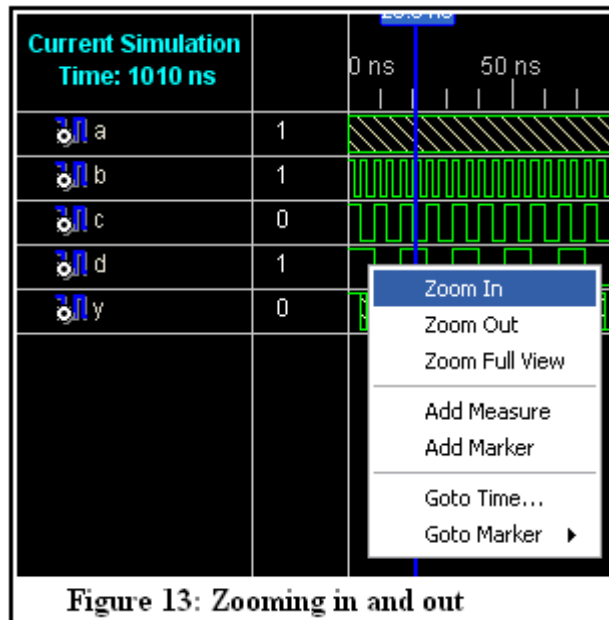
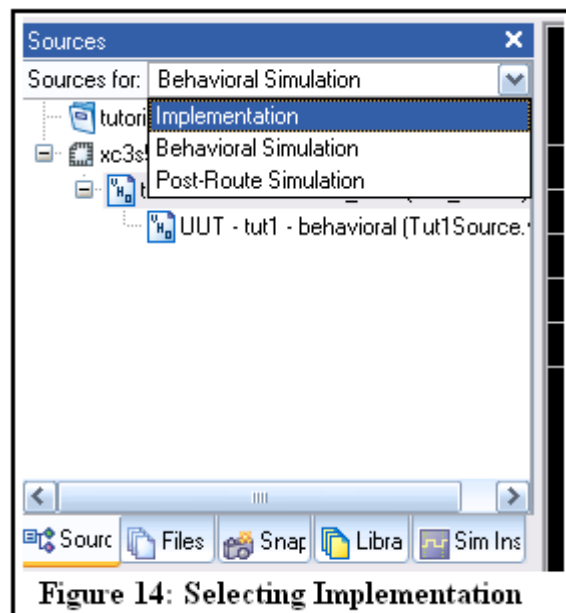


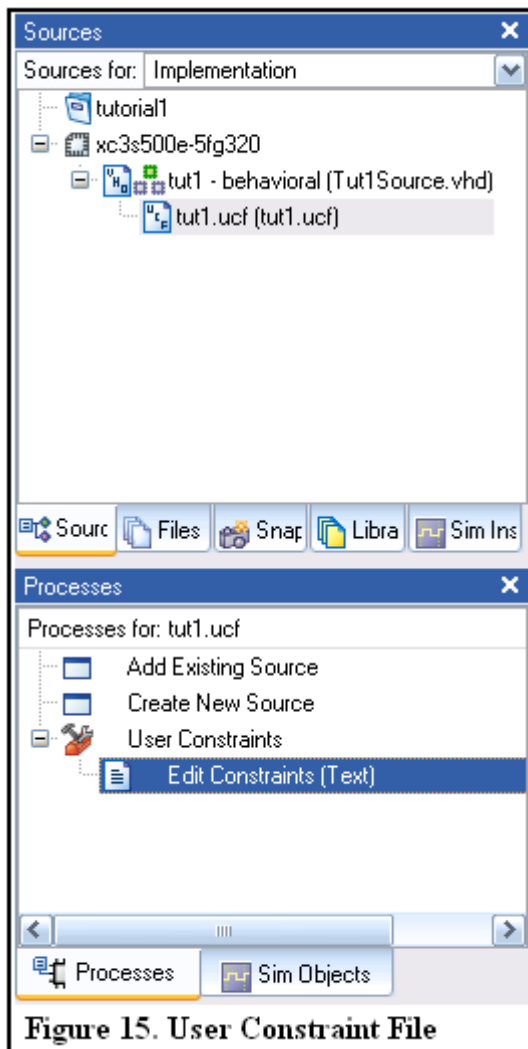
Figure 12: launching simulation



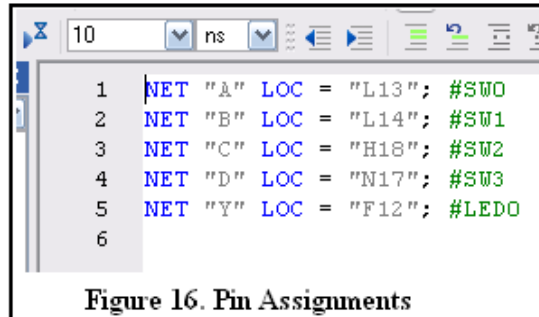
- By clicking into the waveform the blue marker will appear. By scrolling back and forth, you can see what combinations of inputs cause a high on the output.
- In the drop down section of the sources pane, choose Implementation. See figure 14. The source file is used to describe the behavior of the hardware. The third file that is used in this project is the tut1.ucf file. The “ucf” stands for **user constraint file**. What this file does is tie the source file inputs and outputs to the actual pins on the device. There are many more things that a UCF file can be used for but they are beyond the scope of this tutorial.



- Highlight the ucf file in the sources section as shown in figure 15. In the processes tab, expand the user constraints section. Double-click on the Edit Constraints (Text) button. This will bring up the five line ucf file. See figure 16.



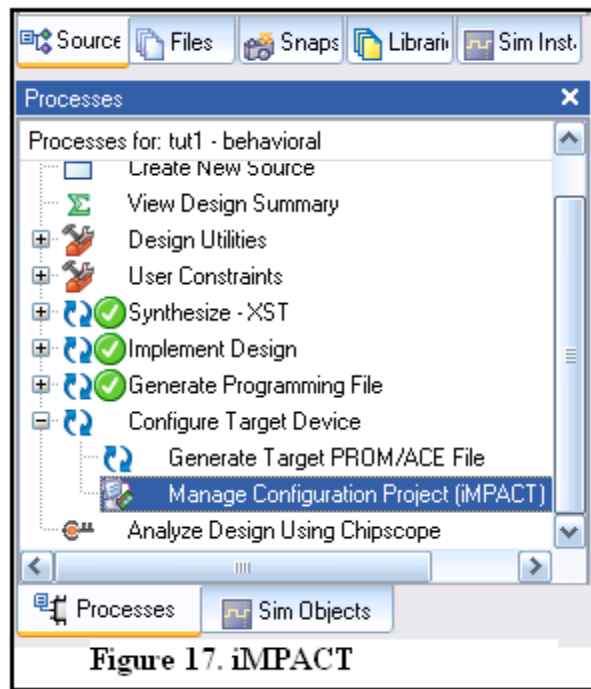
**Figure 15. User Constraint File**



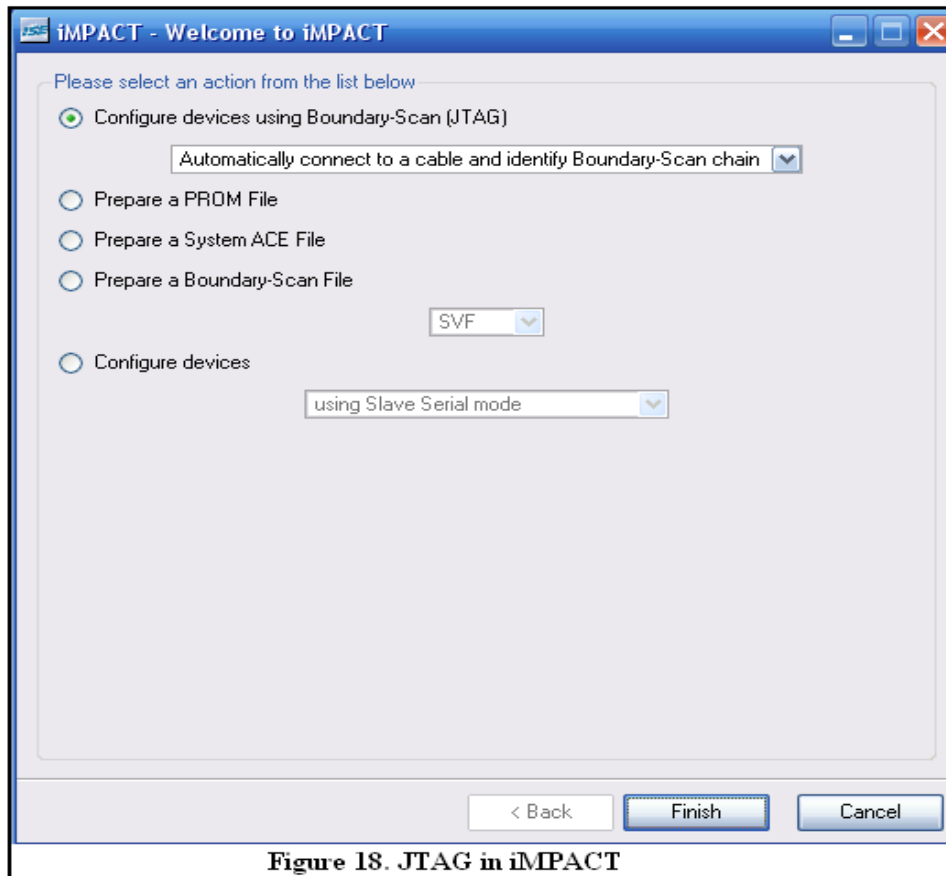
**Figure 16. Pin Assignments**

- The first line ties the input A to pin L13 on the FPGA board. Pin L13 is tied to switch 0 on the FPGA board. The “NET” identifies a signal input or output. The “LOC” identifies a pin number on the FPGA and the green text is comments. The entire listing of all the UCF possible connections is listed on the Digilent web site.

- As mentioned earlier, there are four distinct steps to any FPGA project: design, synthesis, implementation, and programming. The design portion is now finished. It would be possible to now click on Synthesize. When it is done, double click on Implement Design. Once both process have completed double click on generate programming file. This will create the “.bit” file that will be used to program the FPGA. See Figure 17. Finally open the “configure target device” and select iMPACT. iMPACT is the software that the Xilinx tools will use to communicate through the JTAG chain to the programming pins on the FPGA.



16. The box that opens is shown in figure 18. In this instance we will not program anything but the FPGA so click on finish. In later tutorials this will be where you will make choices about programming additional items on the FPGA board.



17. The boundary scan performed on the board finds three items on the JTAG chain. The items are (in order from left to right): the FPGA, the PROM and the CPLD. For the purposes of this project choose the “.bit” file and then Open. For the PROM and the CPLD choose “Bypass”. Finally click OK on the final screen. See Figure 19.

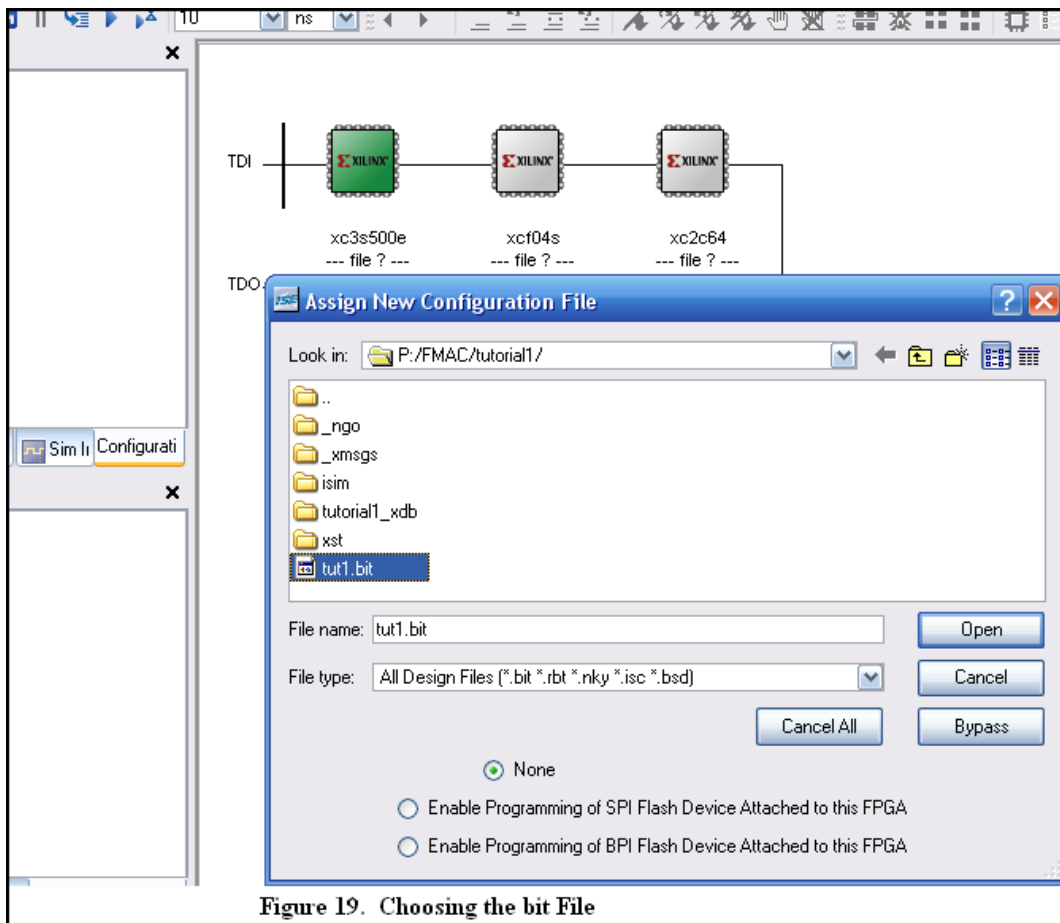


Figure 19. Choosing the bit File

18. Right click on the FPGA and choose “Program”. At this point the programming file is loaded into the device. There are a couple of additional choices that are available that are not necessary at this point. See figure 20.

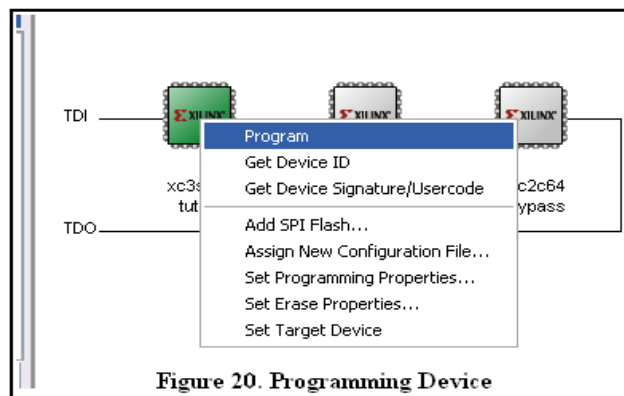
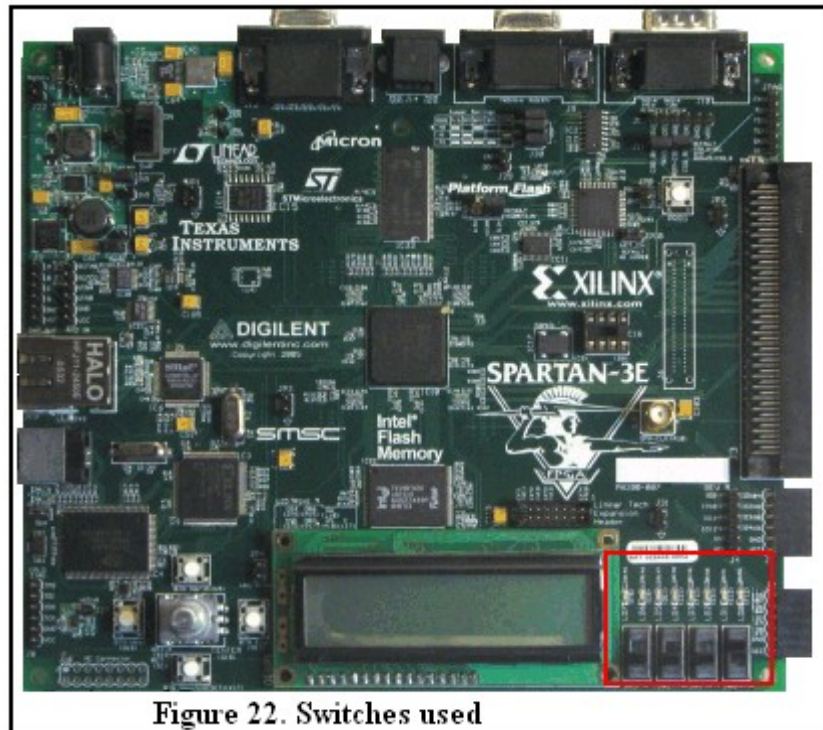
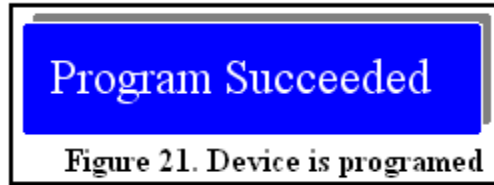


Figure 20. Programming Device

19. When the device has successfully programmed you should be able to use the button to verify the hardware works properly. This is done by sliding the switches to the center for on and towards the board's edge for off. See figures 21 and 22



About the author: Craig Kief is the Deputy Director of the FPGA Mission Assurance Center and can be reached at [DeptDirector@fpgamac.com](mailto:DeptDirector@fpgamac.com).

## VHDL Source File

```
--VHDL files contain three parts: library declarations, entity and
--behavior. The library describes what each function will do. Entity
--describes the inputs and outputs. Behavior is the working
--portion of the project.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--Declaration of the module's inputs and outputs for part 1 of tutorial 2
entity tut1 is port (

A: in std_logic;
B: in std_logic;
C: in std_logic;
D: in std_logic;
Y: out std_logic

);
end tut1;

--Defining the modules behavior
Architecture behavioral of tut1 is
begin
process (A,B,C,D) begin

Y <= (NOT(A) AND B) OR (C AND D);

end process;
end behavioral;
```

## VHDL Testbench

--This is a VHDL testbench. It is used to stimulate the inputs to  
--file we are testing. A good design practice is to create a testbench  
--for each component you are developing. Test each component separately  
--before combining them.

```
LIBRARY IEEE;  
USE IEEE.std_logic_1164.all;  
USE IEEE.std_logic_arith.all;  
USE IEEE.std_logic_unsigned.all;  
USE IEEE.STD_LOGIC_TEXTIO.ALL;  
USE STD.TEXTIO.ALL;
```

```
ENTITY testbench IS  
END testbench;
```

```
ARCHITECTURE testbench_arch OF testbench IS
```

```
COMPONENT tut1  
PORT (  
A : in std_logic;  
B : in std_logic;  
C : in std_logic;  
D : in std_logic;  
Y : out std_logic  
);  
END COMPONENT;
```

```
--develop signals to stimulate  
SIGNAL A : std_logic := '0';  
SIGNAL B : std_logic := '0';  
SIGNAL C : std_logic := '0';  
SIGNAL D : std_logic := '0';  
SIGNAL Y : std_logic := '0';
```

```
--UUT means unit under test  
BEGIN  
UUT : tut1
```

```
--map signals on right to entities on the left  
PORT MAP (  
A => A,  
B => B,  
C => C,  
D => D,  
Y => Y  
);
```

```
signal_A: process
```

```
begin
A <= NOT A;
wait for 1 ns;
end process;

signal_B: process
begin
B <= NOT B;
wait for 2 ns;
end process;

signal_C: process
begin
C <= NOT C;
wait for 4 ns;
end process;

signal_D: process
begin
D <= NOT D;
wait for 8 ns;
end process;

END testbench_arch;
```

## UCF File

```
NET "A" LOC = "L13"; #SW0  
NET "B" LOC = "L14"; #SW1  
NET "C" LOC = "H18"; #SW2  
NET "D" LOC = "N17"; #SW3  
NET "Y" LOC = "F12"; #LED0
```