Activity 1: Xilinx ISE 101

Objective

To introduce students to Xilinx ISE environment and make them comfortable with processes of compilation and synthesis.

Theory

The process of programming an FPGA board using XILINX ISE design suit requires further steps to be performed before we can run it.

1. **Design Entry –** the first step in creating a new design is to specify its structure and functionality. This can be done either by writing an HDL model using some text editor or drawing a schematic diagram using schematic editor.

2. **Design Synthesis** – next step in the design process is to transform design specification into a more suitable representation that can be further processed in the later stages in the design flow. This representation is called the netlist (A **netlist** is a textual description of a circuit made of components.). Prior to netlist creation synthesis tool checks the model syntax and analyze the hierarchy of your design which ensures that your design is optimized for the design architecture you have selected. The resulting netlist is saved to a Native Generic Circuit (NGC) file (for Xilinx® Synthesis Technology (XST) compiler) or an Electronic Design Interchange Format (EDIF) file (for Precision, or Synplify/Synplify Pro tools).

3. **Design Implementation**

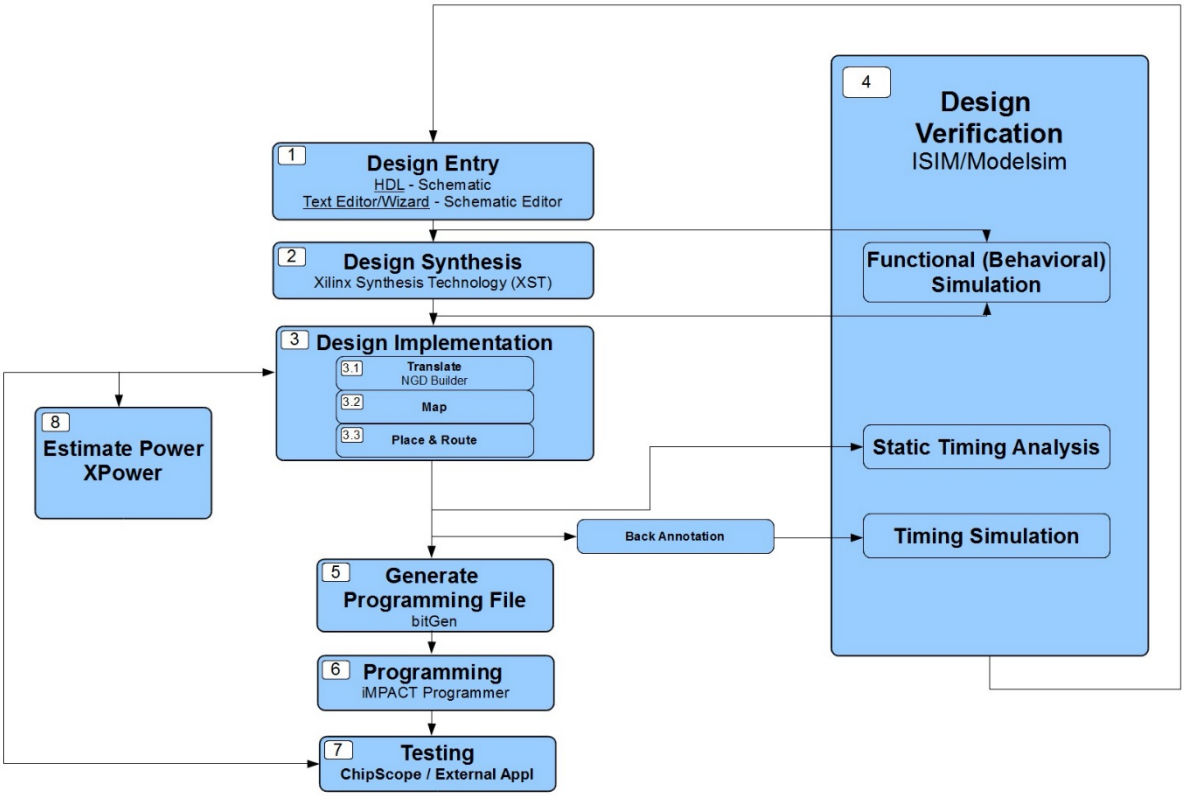
   Implementation step maps netlist produced by the synthesis tool onto particular device's internal structure. It consists from three steps:

   - **Translate step** – merges all incoming netlists and constraints into a Xilinx Native Generic Database (NGD) file.
   - **Map step** - maps the design, specified by an NGD file, into available resources on the target FPGA device, such as LUTs, Flip-Flops, BRAMs etc. As a result, a Native Circuit Description (NCD) file is created.
   - **Place and Route step** - takes a mapped Native Circuit Description (NCD) file, places and routes the design, and produces an NCD file that is used as input for bitstream generation.

4. **Design Verification** – is very important step in design process. A verification is comprised of seeking out problems in the HDL implementation in order to make it compliant with the design specification. A verification process reduces to extensive simulation of the HDL code. Design Verification is usually performed using two approaches: Simulation and Static Timing Analysis.

   There are two types of simulation:

- **Functional (Behavioral) Simulation** – enables you to simulate or verify a code syntax and functional capabilities of your design. This type of simulation tests your design decisions before the design is implemented and allows you to make any necessary changes early in the design process. In functional (behavioral) simulation no timing information is provided.
- **Timing Simulation** – allows you to check does the implemented design meet all functional and timing requirements and behaves as you expected. The timing simulation uses the detailed information about the signal delays as they pass through various logic and memory components and travel over connecting wires. Using this information it is possible to accurately simulate the behaviour of the implemented design. This type of simulation is performed after the design has been placed and routed for the target PLD, because accurate signal delay information can now be estimated. A process of relating accurate timing information with simulation model of the implemented design is called **Back-Annotation**.



- **Static Timing Analysis** – helps you to perform a detailed timing analysis on mapped, placed only or placed and routed FPGA design. This analysis can be useful in evaluating timing performance of the logic paths, especially if your design doesn't meet timing requirements. This method doesn't require any type of simulation.

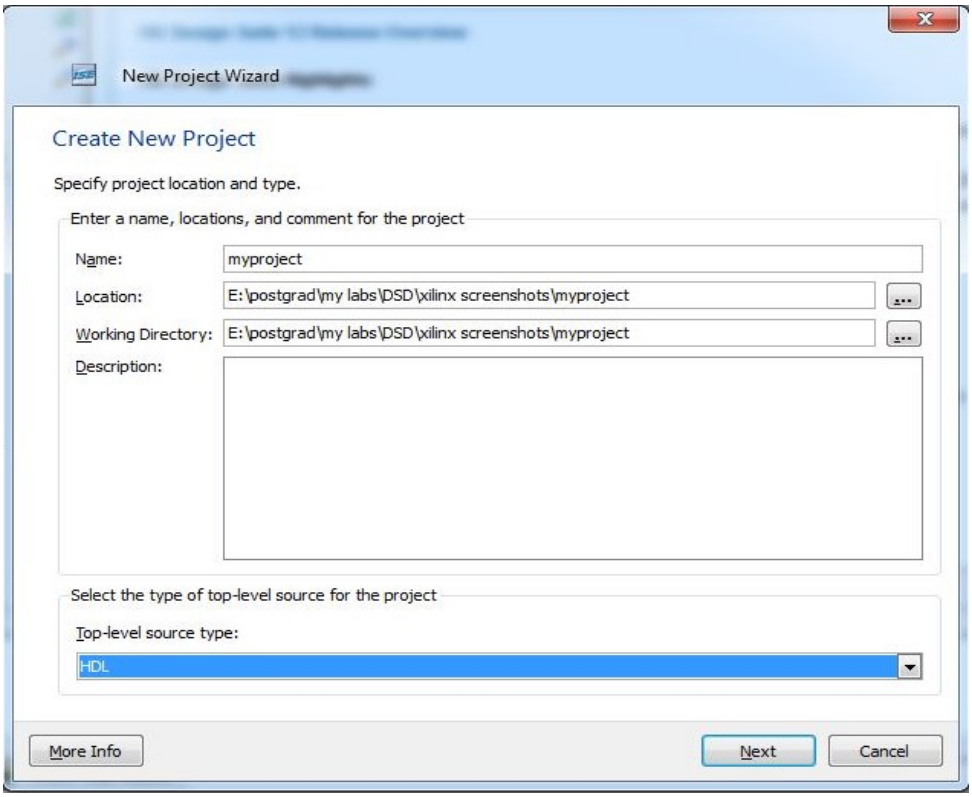
5. **Generate Programming File** – this option runs BitGen, the Xilinx bitstream generation program, to create a bitstream file that can be downloaded to the device.

6. **Programming** – iMPACT Programmer uses the output from the Generate Programming File process to configure your target device.

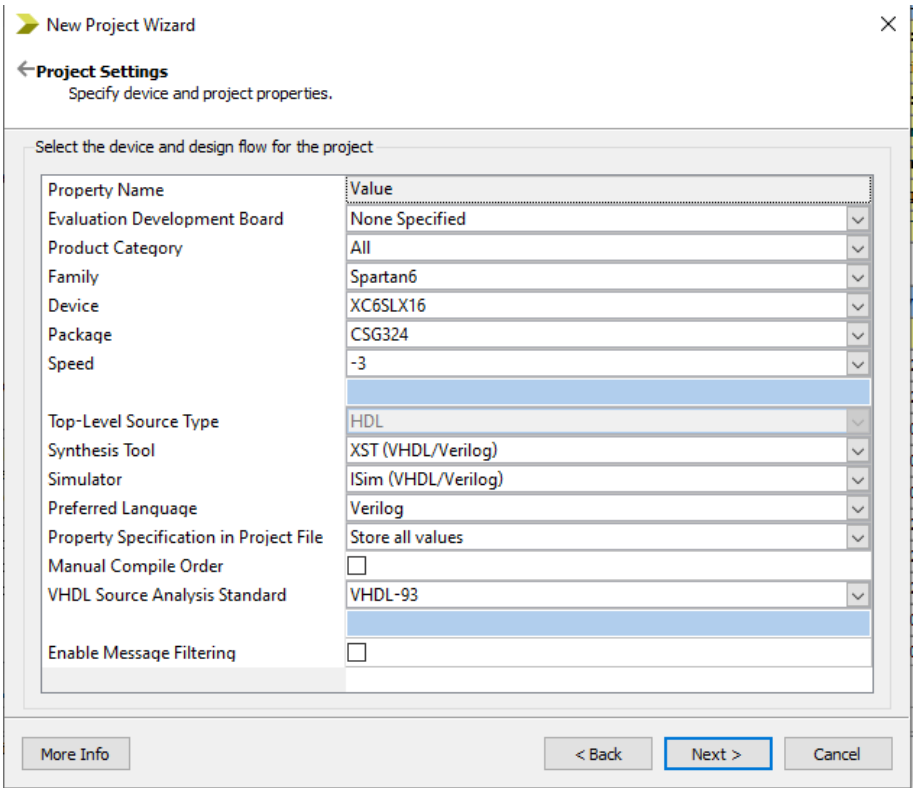
7. **Testing** – after configuring your device, you can debug your FPGA design using the Xilinx ChipScope Pro tool or some external logic analyzer.

Practical Steps:

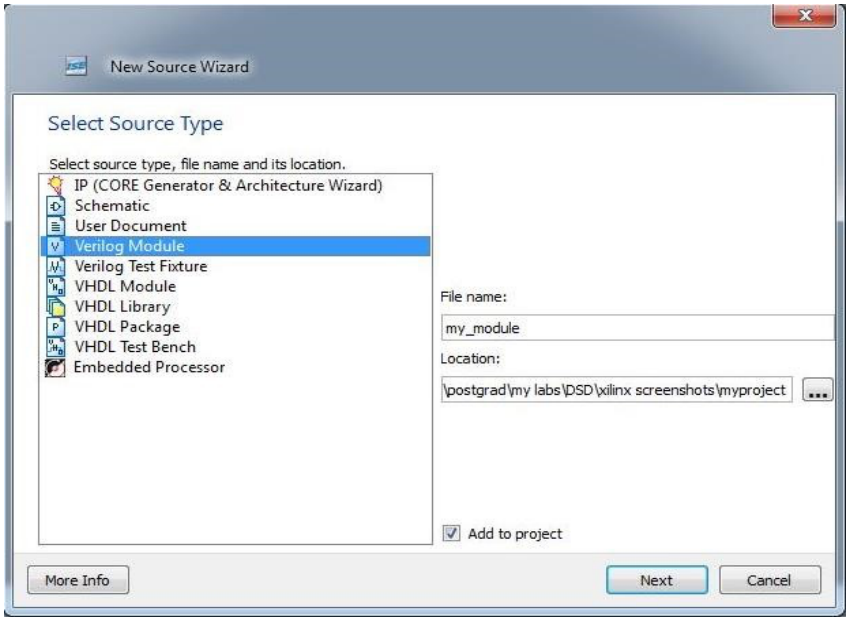
1. Create a project by going to File->New Project



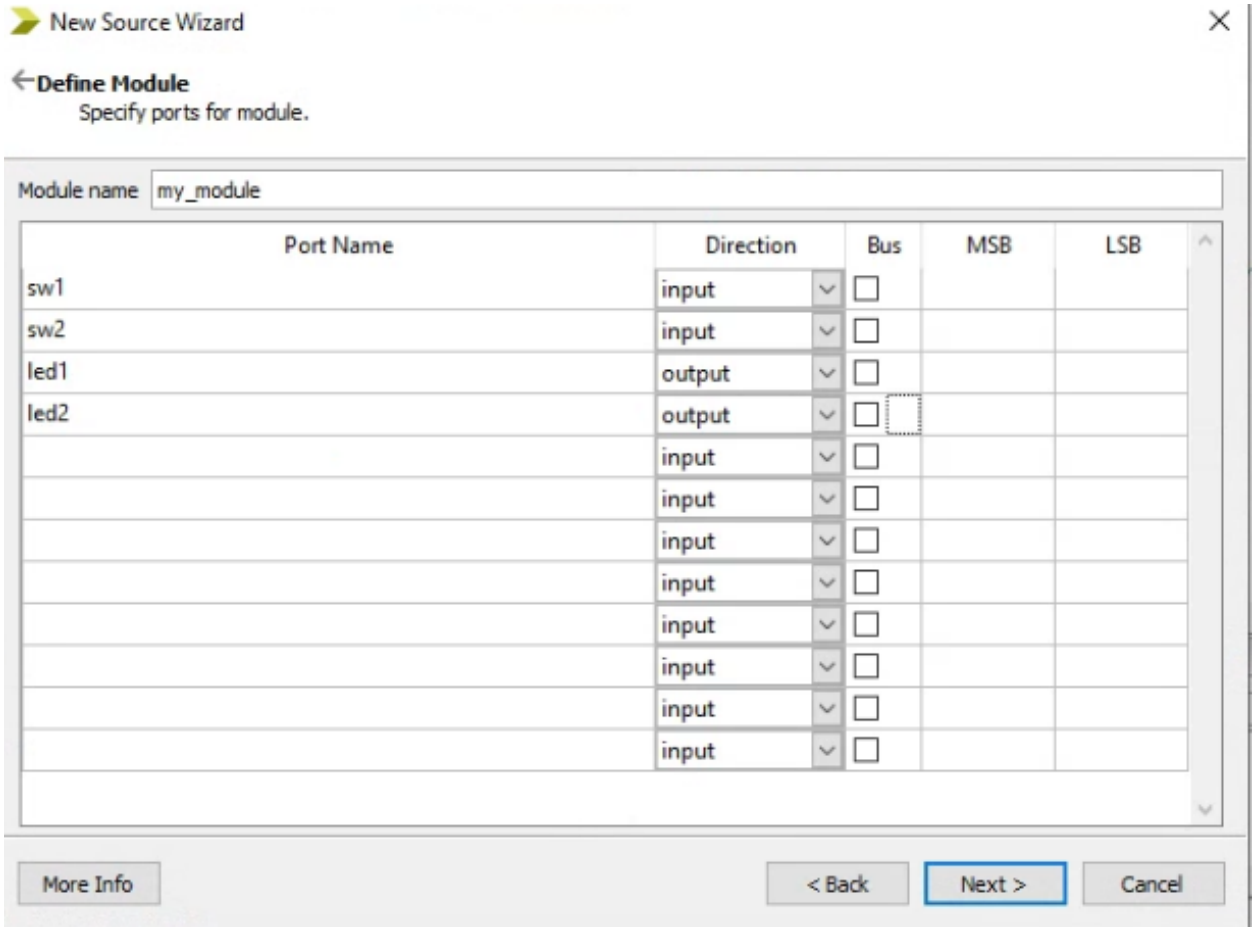
2. After creating a top level HDL project, use the following settings



3. Create a Verilog module.

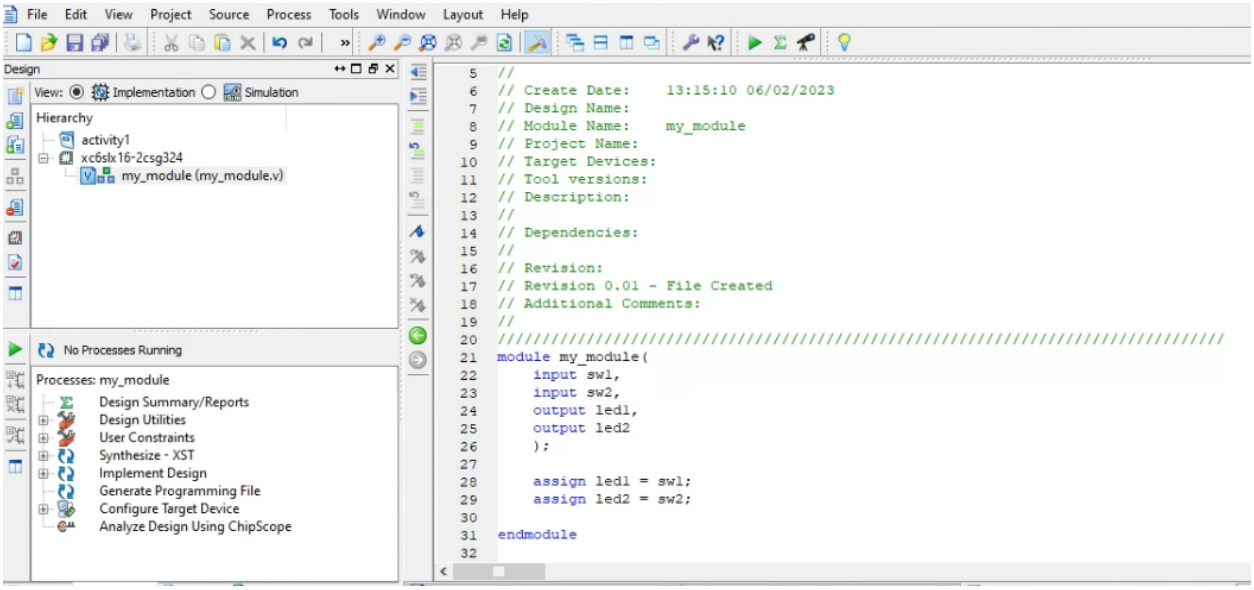


4. Populate the port list and write the Verilog code.

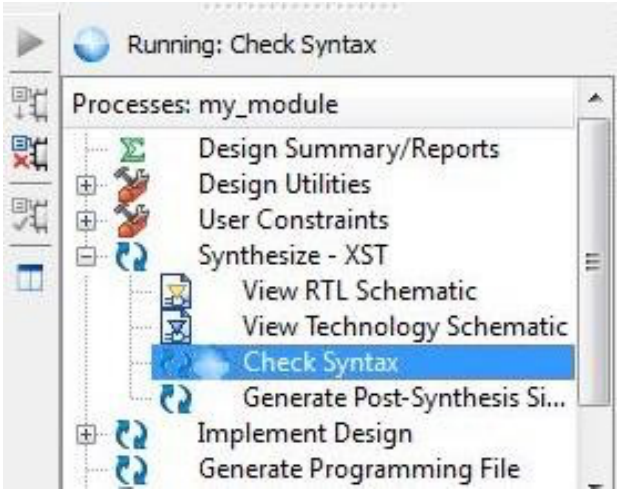


Add following lines of code:

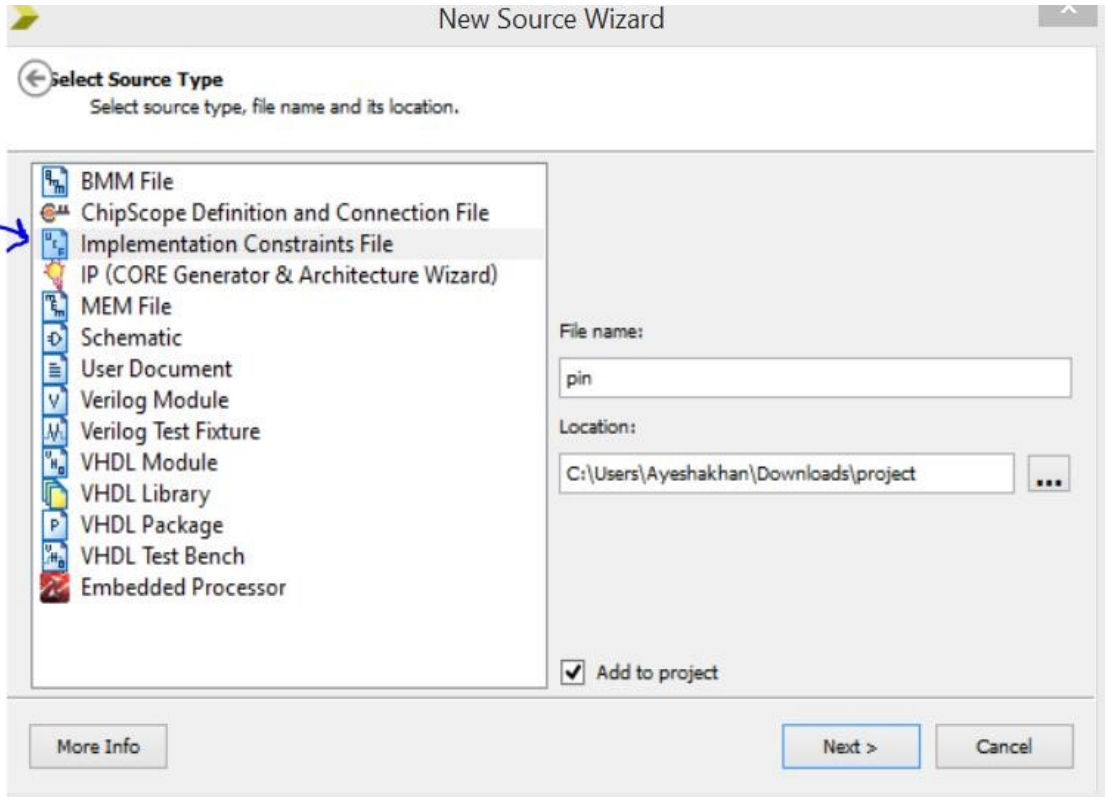
```
assign led1 = sw1;
assign led2 = sw2;
```



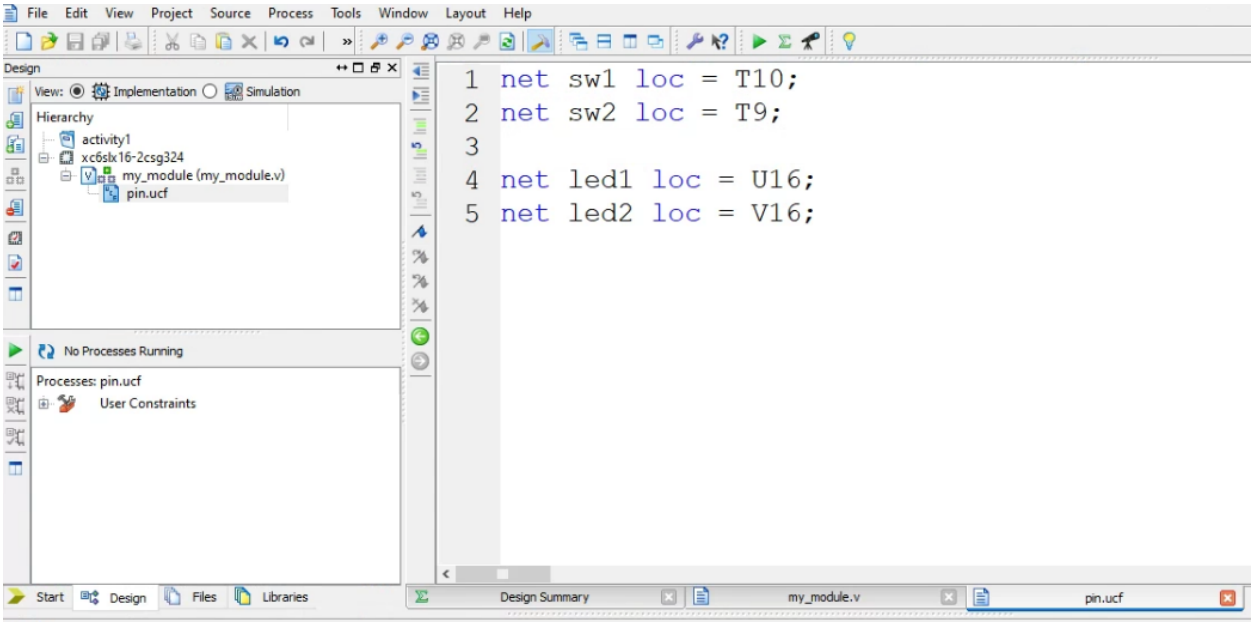
5. Once the code is final, check for syntax errors.



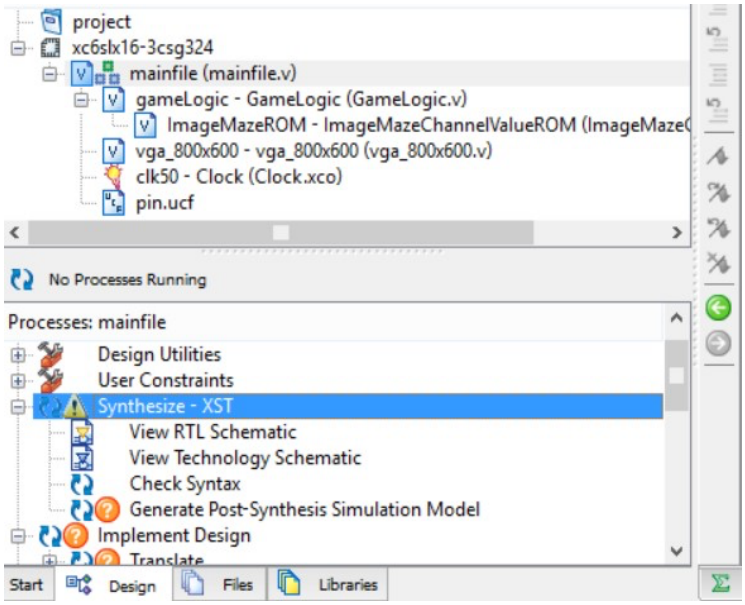
6. After setting up the XILINX Project as learnt in last lab, add a new source file: this time an Implementation Constraints File.



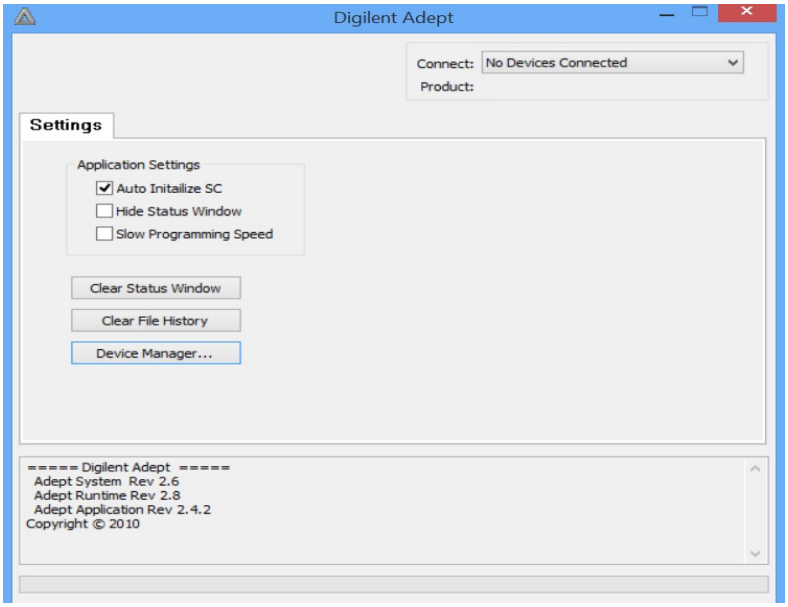
7. In order to assign make the code work with external/internal ports we need to add a new source file in the design by the name of user constraint file. Pin assignment to input and output ports in one of many operations of UCF. Now add the pin assignments according to your requirement as per the syntax shown below (NET and LOC are keywords, NET refers to name of port used in code whereas LOC refers to the physical location of pin on the board).



8. Once the code is final, Synthesize it to make it error free, then double click on Implement Design, It consists of three parts-(translate, Map, Place and route) and then generate the programming File by double clicking on Programming File Generation.



9. Once everything is checked, Plug in your FPGA board using USB cable and using Adept connect the device then add the .bit file generated in last step and click to program.



Controlling LEDs

Turn ON/OFF the LEDs by implementing the given code with the help of switches on Nexys3 FPGA board.

Verilog Module:

```
//
//////////////////////////////////////////////////////////////////////////////////
module my_module(
    input sw1,
    input sw2,
    output led1,
    output led2
    );

    assign led1 = sw1;
   assign led2 = sw2;

endmodule
```

Implementation Constraint File: (UCF)

```
net sw1 loc = T10;
net sw2 loc = T9;
net led1 loc = U16;
net led2 loc = V16;
```