Introduction to Xilinx Vivado tools

This document is meant to be a starting point for users who are new to using the Xilinx Vivado tools. The document will describe the basic steps to start, create, simulate, synthesize, implement and program an FPGA using Vivado through a series of screenshots and an example design which is a simple binary counter.

Starting the tool:

Use the windows application browser to start Vivado 2014.2. If a different version of the tool is installed on your computer, start the corresponding tool. The steps described in this document are very similar to other versions of the tool.

Once the tool is started, you will see a page as shown in Fig. 1. Spend some time looking at the quick take videos which talk about tool features on a wide variety of subjects.

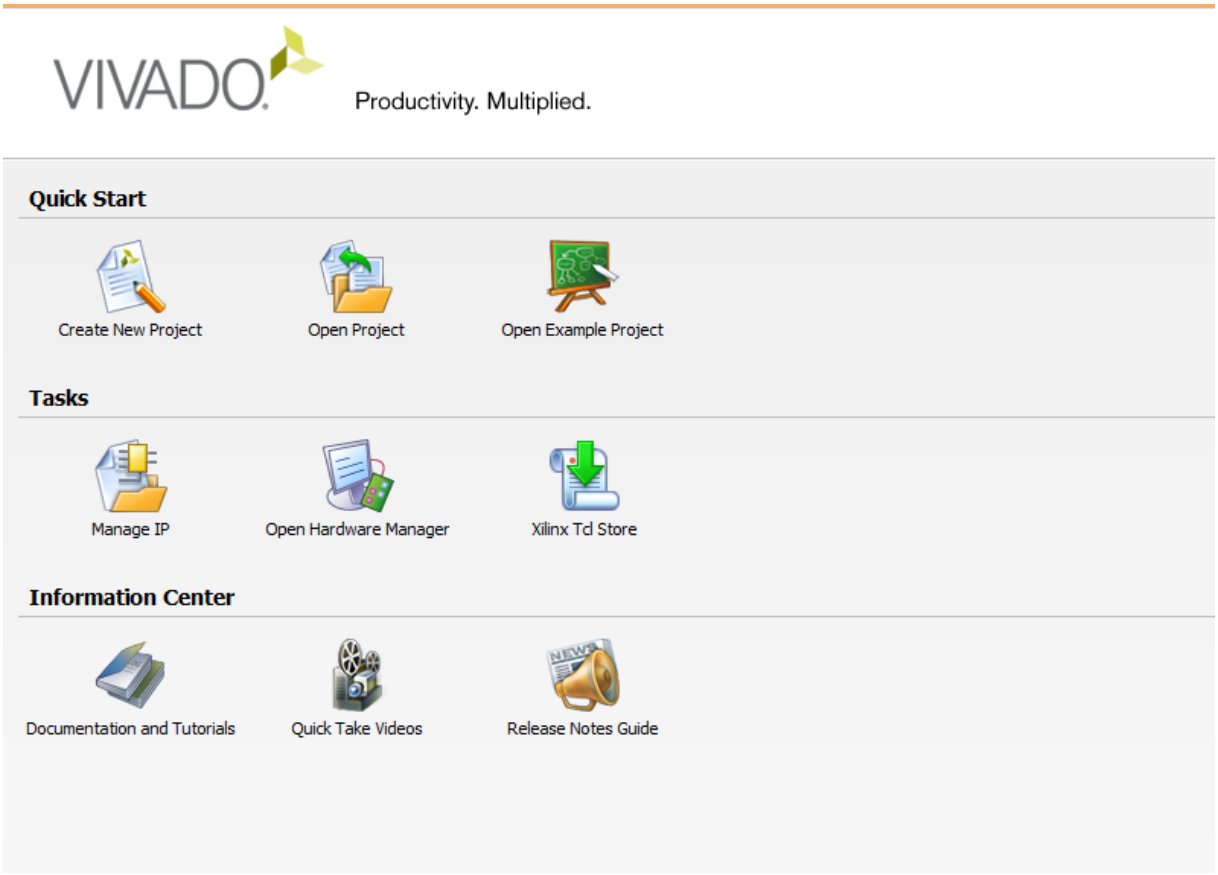

Figure 1: Vivado start page

Click on Create New Project to start a new project. You will be prompted to specify a directory and a project name. Use a meaningful name so that it is easier to open and remember what the project was about at a later point of time.

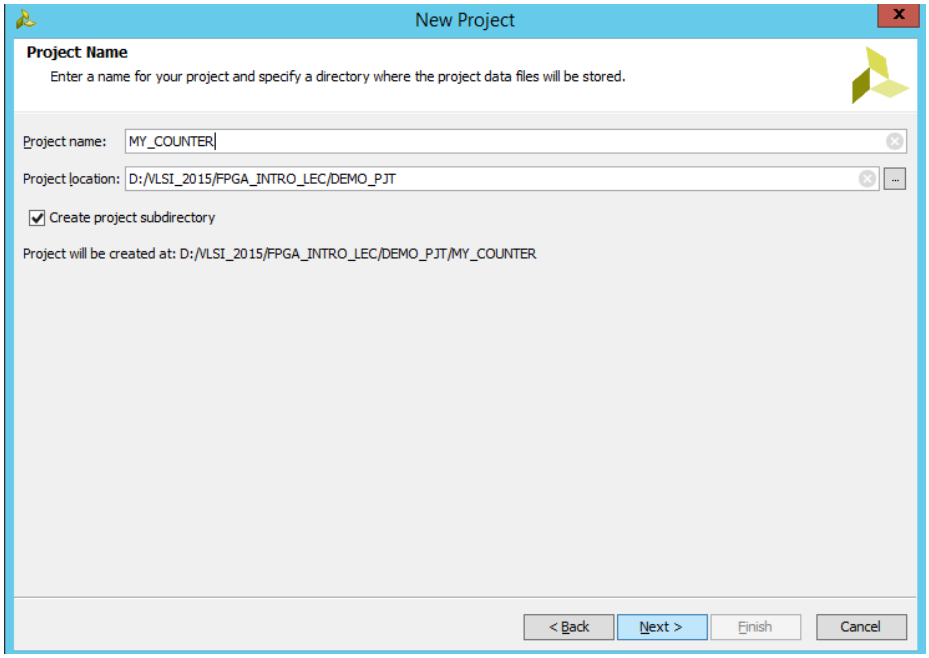

Figure 2: Choose project name and directory

Navigate through the project creation assistant by clicking on Next. Add sources, IPs and constraints if you have any. You can always add new sources, IPs and constraints later into the project.

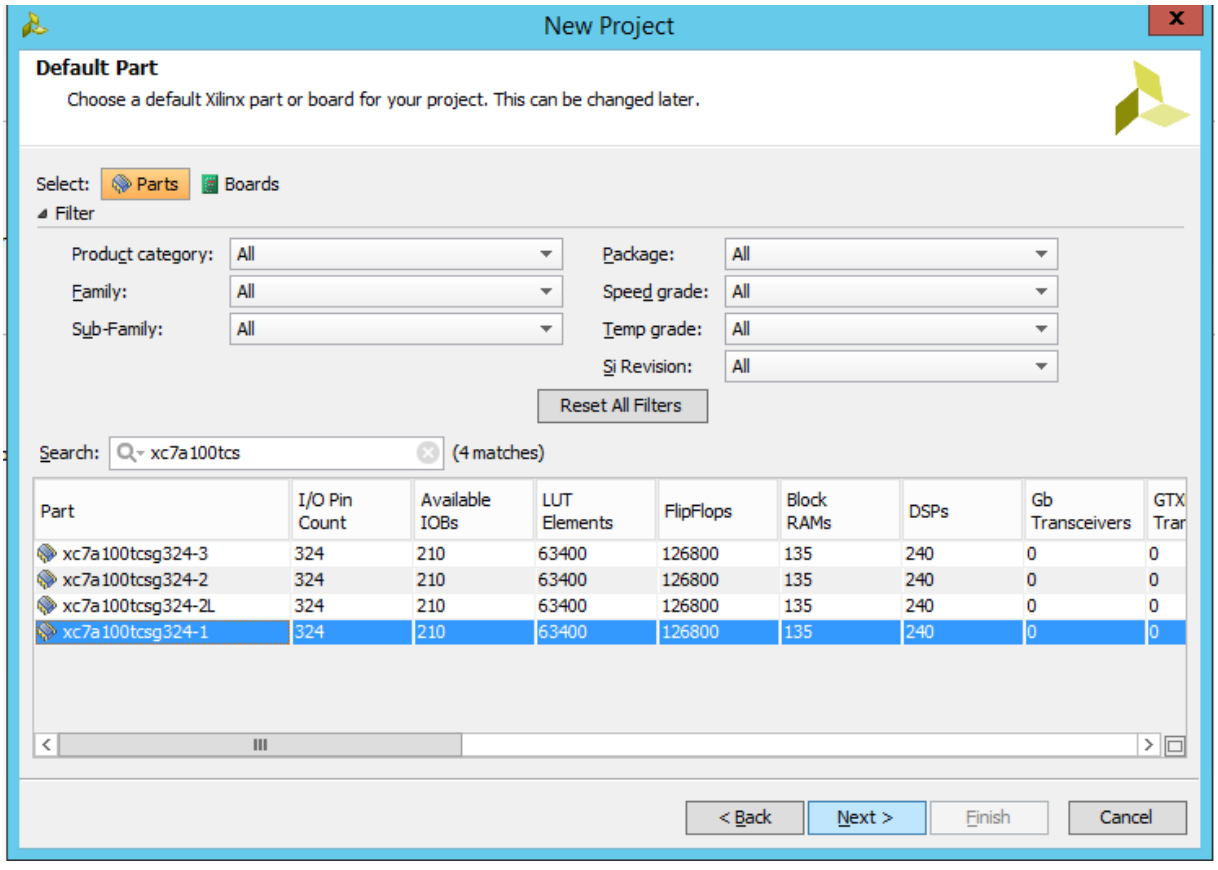

Figure 3:Choose the target FPGA

Choose the target FPGA. We will use the Xilinx Artix-7 FPGA with a speed grade of 1.

If you are successful in following the above steps, you will end up with a screen which looks like the one shown in Fig. 4.

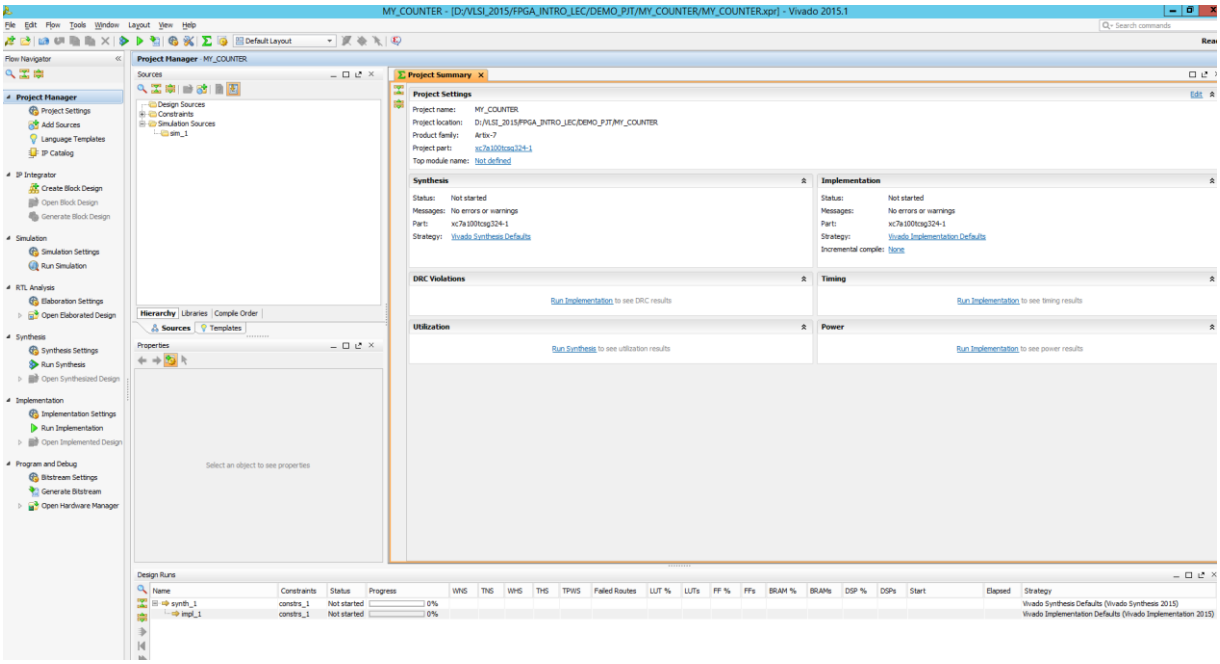

Figure 4: Project home page

The left hand pane is called the Flow navigator. It enables you to go through the FPGA flow from simulation, synthesis and implementation and finally to generate the bitstream and program the FPGA. More details are shown in Fig.5 .

Project settings such as synthesis, implementation and bit generation options can be modified by clocking on the Project settings tab.

We will not use the IP integrator in this course, but this is used to package your design into your own custom IP which can be used by others.

Simulations can be launched by clicking on Run Simulation tab. Remember to set the correct top level for running simulations. Often it is a testbench which is created by you are provided by the course instructor to verify your design. This can be done by right clicking on the module name in the “Sources” window and choosing the right module. Usually, the top level for simulation is different from the top level file for synthesis and implementation as the testbench contains non synthesizable constructs.

Synthesis can be launched by clicking on Run Synthesis. Synthesis nor simulation will work if there are syntax errors, and Vivado will show errors in the “Sources” window as well as the “Messages” window. A design which has been synthesized can be examined further by clicking on the “Open Synthesized Design” tab. This will enable you to examine the schematics, add additional timing constraints and other taskts.

A synthesized design can be implemented for the target FPGA by clicking on the “Run Implementation” tab. You can examine the design placement on the FPGA, and timing results once implementation is completed by opening the implemented design.

Finally a bitstream can be generated and the “Open Hardware Manager” tab can be used to connect to the FPGA and program the generated bitstream for execution on the FPGA.

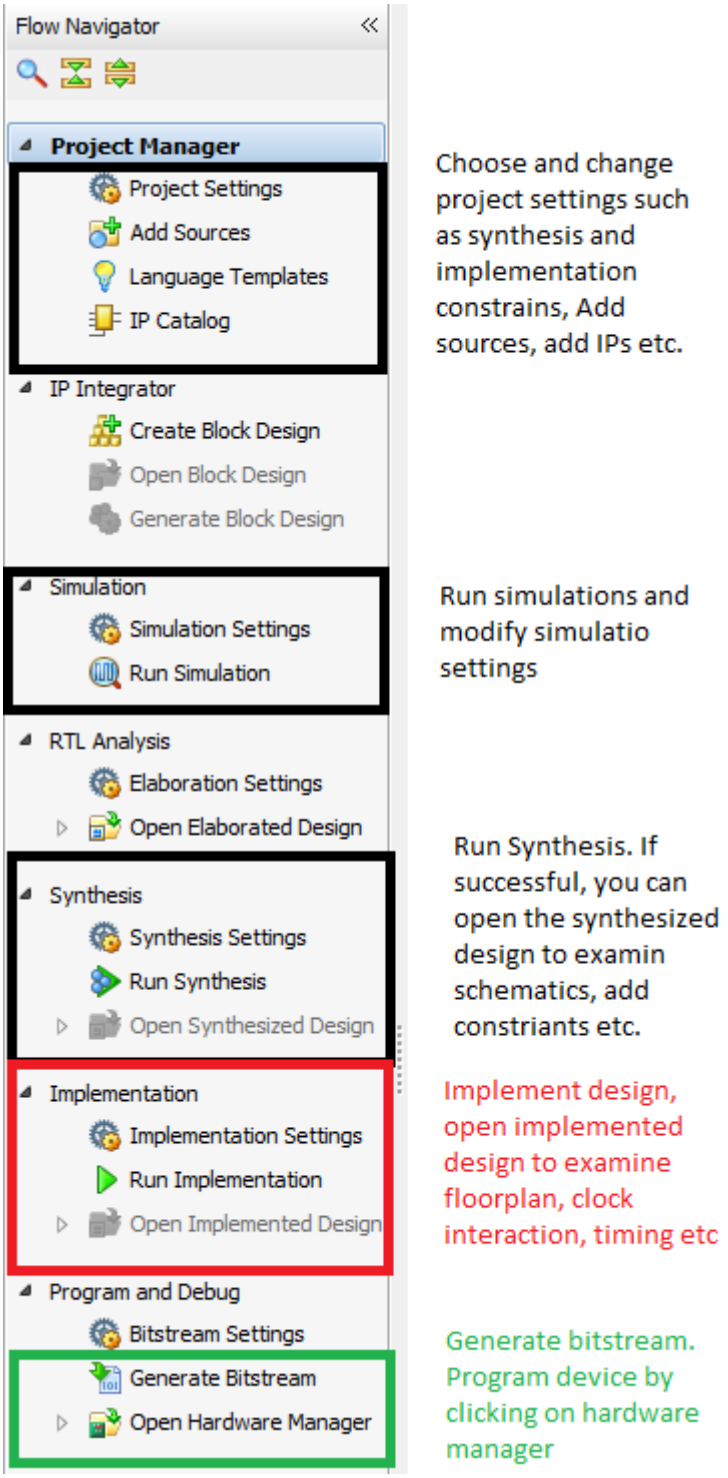

Figure 5: Project Flow navigator

The Counter example that we will use for this document uses an internal clock generator for running the counter. The code for the counter module is shown below. It contains a clk_wiz_0 component which is the clock generator we will use. There is one input clock, connected to the top level port along with two output clocks, one of which will be used by our design.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

entity my_counter is
    Port ( start_count : in STD_LOGIC;
           rst : in STD_LOGIC;
           clk_in1 : in STD_LOGIC;
           count_val : out unsigned (2 downto 0));
end my_counter;

architecture Behavioral of my_counter is

component clk_wiz_0
port
 (-- Clock in ports
  clk_in1           : in     std_logic;
  -- Clock out ports
  clk_out1          : out    std_logic;
  clk_out2          : out    std_logic;
  -- Status and control signals
  reset             : in     std_logic;
  locked            : out    std_logic
 );
end component;

signal  clk_out1    : std_logic;
signal  clk_out2    : std_logic;
signal  locked      : std_logic;
signal  local_count : unsigned (19 downto 0);
signal  local_probe : std_logic_vector(7 downto 0);
signal  local_count_val : std_logic_vector(3 downto 0);

begin

 my_clock_gen: clk_wiz_0
   port map (
  -- Clock in ports
  clk_in1 => clk_in1,
 -- Clock out ports
  clk_out1 => clk_out1,
  clk_out2 => clk_out2,
 -- Status and control signals
  reset => '0',
  locked => locked
);

process(clk_out1, rst)
begin
  if(rst = '0') then
    local_count <= (others => '0');
  elsif(rising_edge(clk_out1)) then
    if(start_count = '1') then
        local_count <= ((local_count) + 1);
    else
        local_count <= local_count;
  end if;
  end if;
 end process;

 count_val <= local_count(2 downto 0) + local_count(19 downto 17);
 local_count_val <= std_logic_vector(local_count(3 downto 0));

end Behavioral;
```

The following snippet contains the code for the counter testbench.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity counter_testbench is
--  Port ( );
end counter_testbench;

architecture Behavioral of counter_testbench is

component my_counter
    Port ( start_count : in STD_LOGIC;
           rst : in STD_LOGIC;
           clk_in1 : in STD_LOGIC;
           count_val : out unsigned (2 downto 0));
end component;

signal start_count_tb   : std_logic;
signal rst_tb           : std_logic;
signal clk_in_tb        : std_logic := '0';
signal count_val_tb     : unsigned(2 downto 0);
constant clk_period     : time := 2.5ns;
constant delay_val      : time := 10ns;

begin

my_counter_inst : my_counter
port map(
    start_count     => start_count_tb,
    rst             => rst_tb,
    clk_in1         => clk_in_tb,
    count_val       => count_val_tb);

clk_in_tb           <= not clk_in_tb after clk_period;

rst_tb              <= '0' after delay_val,'1' after 1500*delay_val;
start_count_tb      <= '0' after delay_val, '1' after 2000*delay_val;

end Behavioral;
```

The constraints file looks like this

```
set_property PACKAGE_PIN E3 [get_ports clk_in1]
set_property IOSTANDARD LVCMOS33 [get_ports clk_in1]
#create_clock -add -name clk_in1 -period 10.00 -waveform {0 5} [get_ports clk_board]

set_property PACKAGE_PIN C12 [get_ports {rst}]
set_property IOSTANDARD LVCMOS33 [get_ports {rst}]

set_property PACKAGE_PIN U9 [get_ports {start_count}]
set_property IOSTANDARD LVCMOS33 [get_ports {start_count}]
# LED 0
set_property PACKAGE_PIN T8 [get_ports {count_val[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {count_val[0]}]
# LED 1
set_property PACKAGE_PIN V9 [get_ports {count_val[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {count_val[1]}]
# LED 2
set_property PACKAGE_PIN R8 [get_ports {count_val[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {count_val[2]}]

set_property CFGBVS Vcco [current_design]
set_property config_voltage 3.3 [current_design]
```

Since we have a clock generator as a component we will need to instantiate/create this for our design. We can do this by clicking on “IP Catalog” in the the “Flow Navigator”. Search for an IP starting with “Clocking Wizard” in the Catalog and double click on it. You should see a window which lets you customize the IP. Choose two output clocks and leave the rest of the setting unchanged. Click “OK” to generate the Clock generator IP. Fig. 6 shows a snapshot of this process.

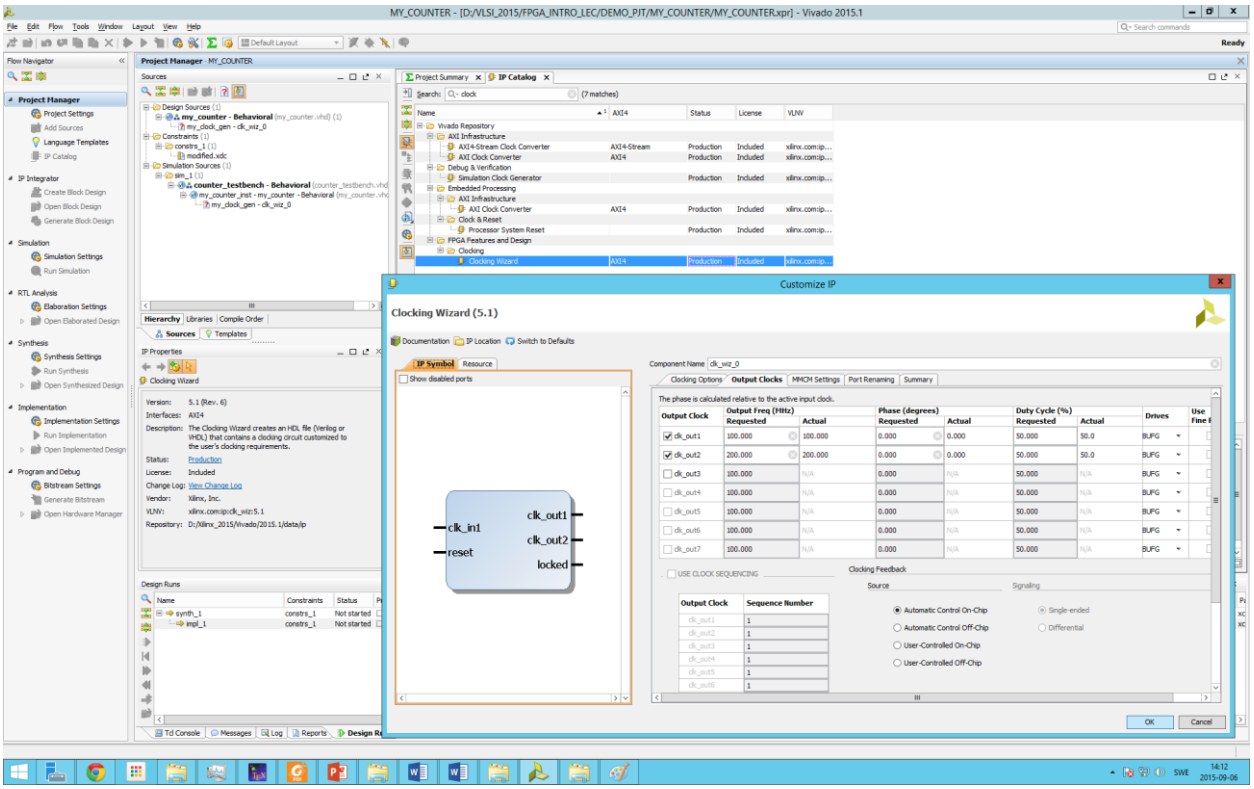

Figure 6: Generating a Clocking IP

Once the generation is complete, you will no longer see a ? mark under the “my_clock_gen” instance name. You are not ready to launch the simulation. Click on “Run Simulation” in the “Flow Navigator”. Make sure that “counter_testbench” is the top level in your simulation sources.

Examine the “log” window in at the bottom to see if there are any errors or warnings. If simulation is successful, the simulator is launched and you can see some signals in the waveform viewer. You can choose additional signals to debug by clicking on the corresponding module instance name, then the signal and right-click-> add to waveform window.

Run the simulation for 22us by typing *run 22us* in the “Tcl Console” at the bottom. You can zoom in zoom out using the tools in the waveform viewer.

Fig. 7. Shows a snapshot of the simulation

Once you are satisfied with the simulation behaviour, you can “Run Synthesis”. Make sure you keep an eye on the “log” window to see the messages that appear during synthesis. You can always view Reports by using the Reports tab at the bottom as shown in Fig. 8.

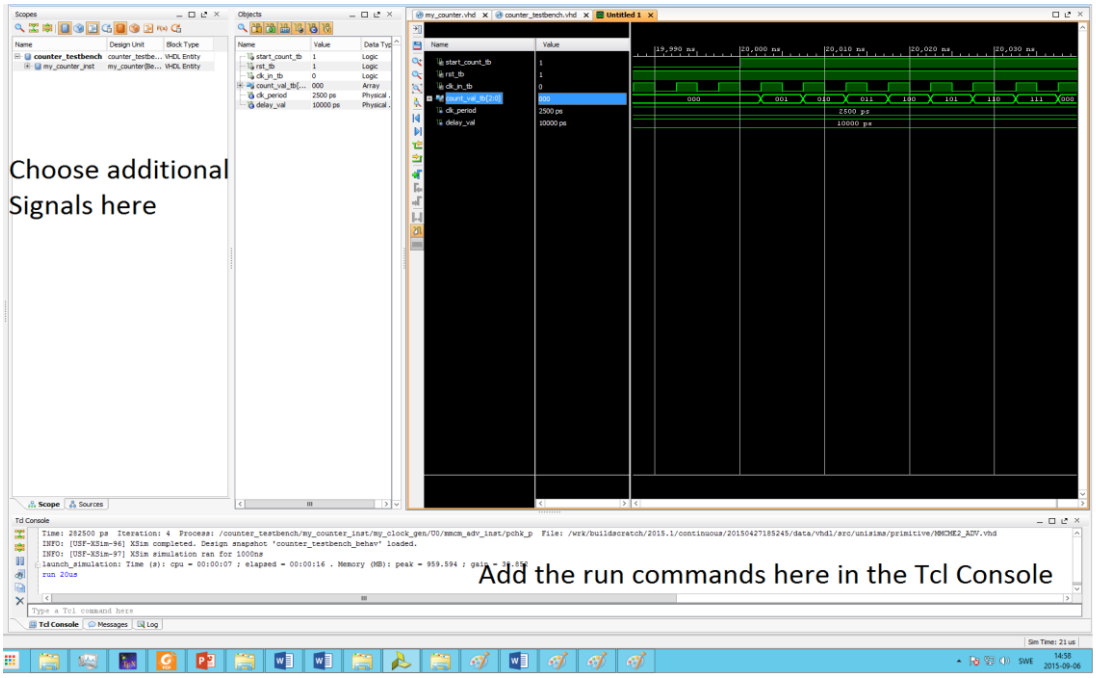

Figure 7: Simulation window

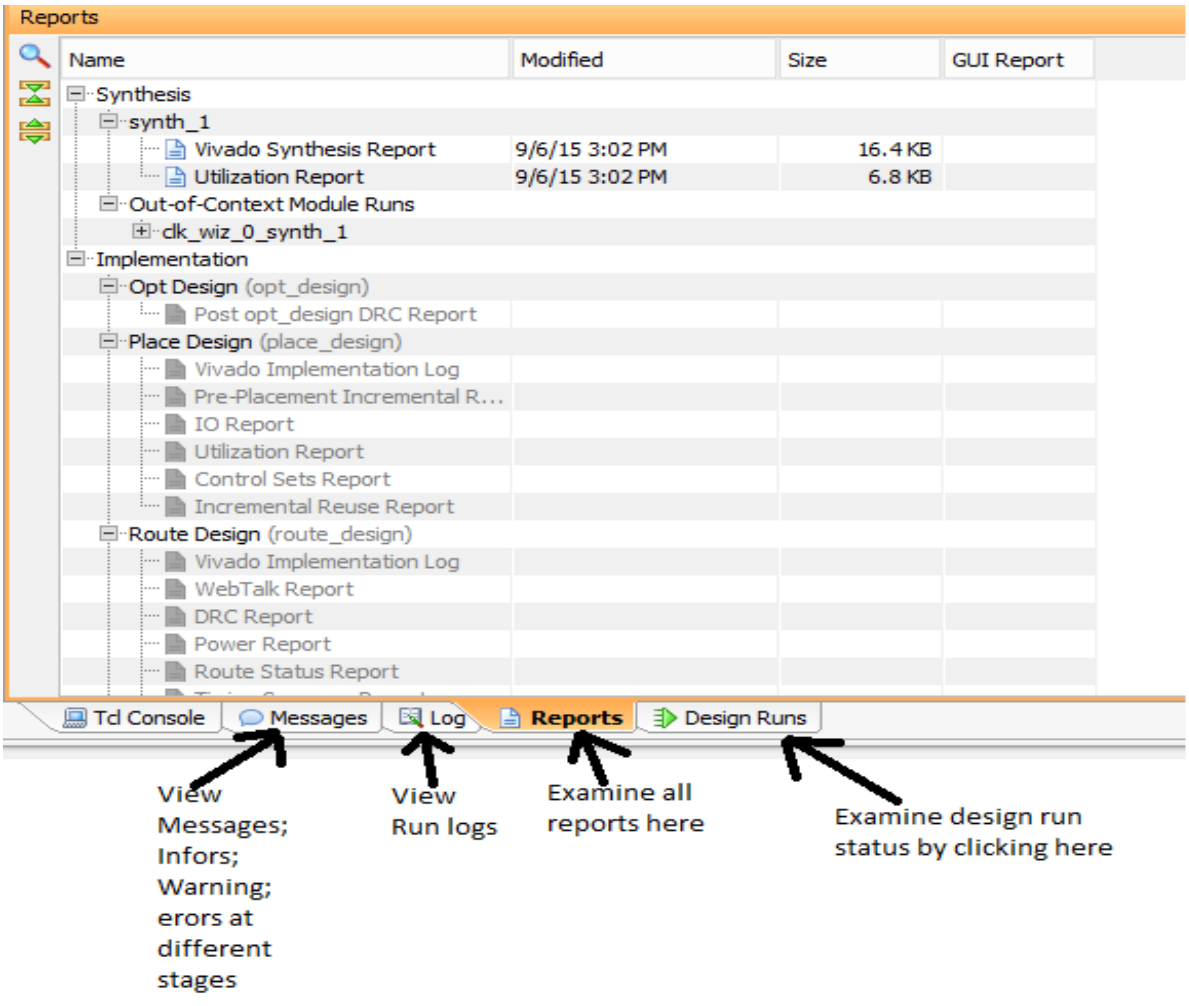

Figure 8: Logs and Messages Viewer

Once synthesis is complete, you can “Open Synthesized Design” and examine the various menus such as creating additional constraints, viewing the schematic etc.

“Run Implementation” when you are ready and wait for implementation to finish. It is very important to make sure that the design constraints are met when implementation ends. Keep an eye on the “log” window and look at the “Messages” to understand any Warning and Errors that you might encounter. Once Implementation is complete, you can click on “Open Implemented Design” to examine the actual mapping of your design onto FPGA resources, review timing reports and also to examine clock interaction if your design has multiple clocks. The implemented design for the counter is shown in Fig. 9.

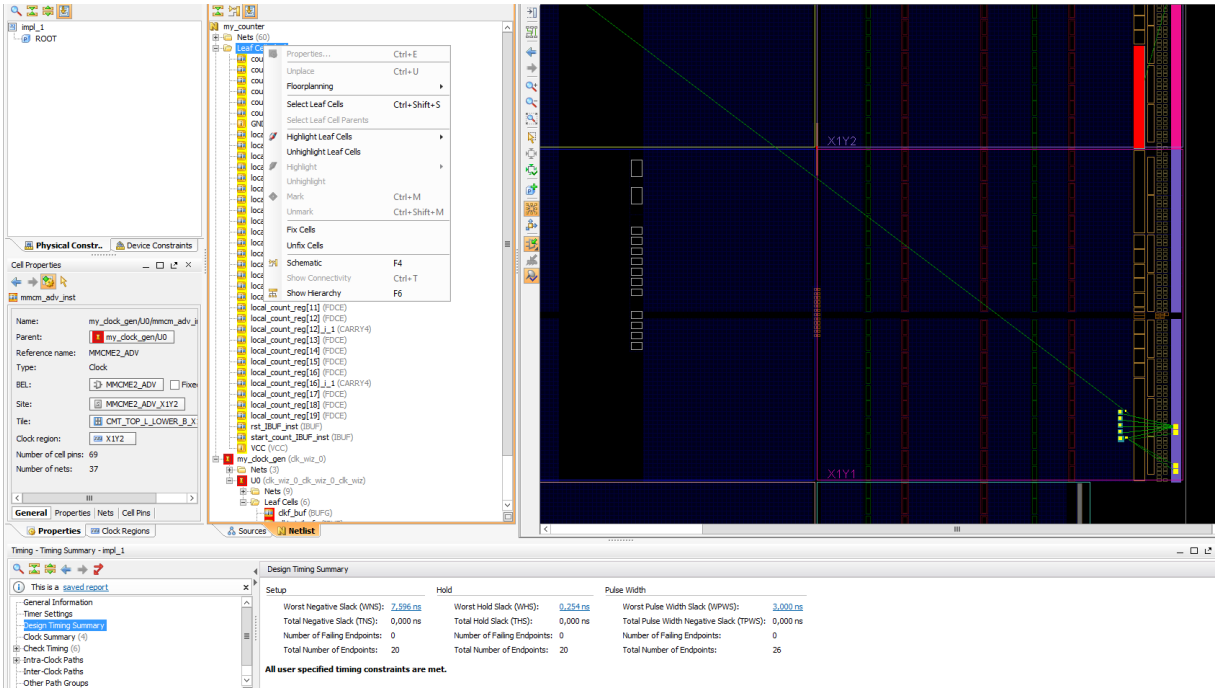

Figure 9: Implemented design

You can zoom in-out from different parts of your design as well as highlight different modules of your design by right clicking in the netlist view. Examine timing to make sure that both setup and hold slack are zero or positive. In Fig. 9 the clock generator is highlighted in red and the counter in yellow along with the pads which are used for output mapping. Note that the router places the design close to the pads to make routing easier. It is also possible to cross-probe into RTL code to examine which resource was mapped to which line of your RTL code by choosing the resource in the netlist view, right clicking and choosing “Go to source”.

Another important window in Vivado is the Project summary window. Here you can get an overview of all the different stages of your design flow and examine resource utilization as well as get a power consumption estimate for your design. Fig 10. Shows a snapshot from the project summary window for the counter design. Make sure that you have no critical warning/DRC errors before you generate your bit stream. Sometimes, bit stream generation will fail if you have not made sure that the design constraints and pins are mapped correctly.

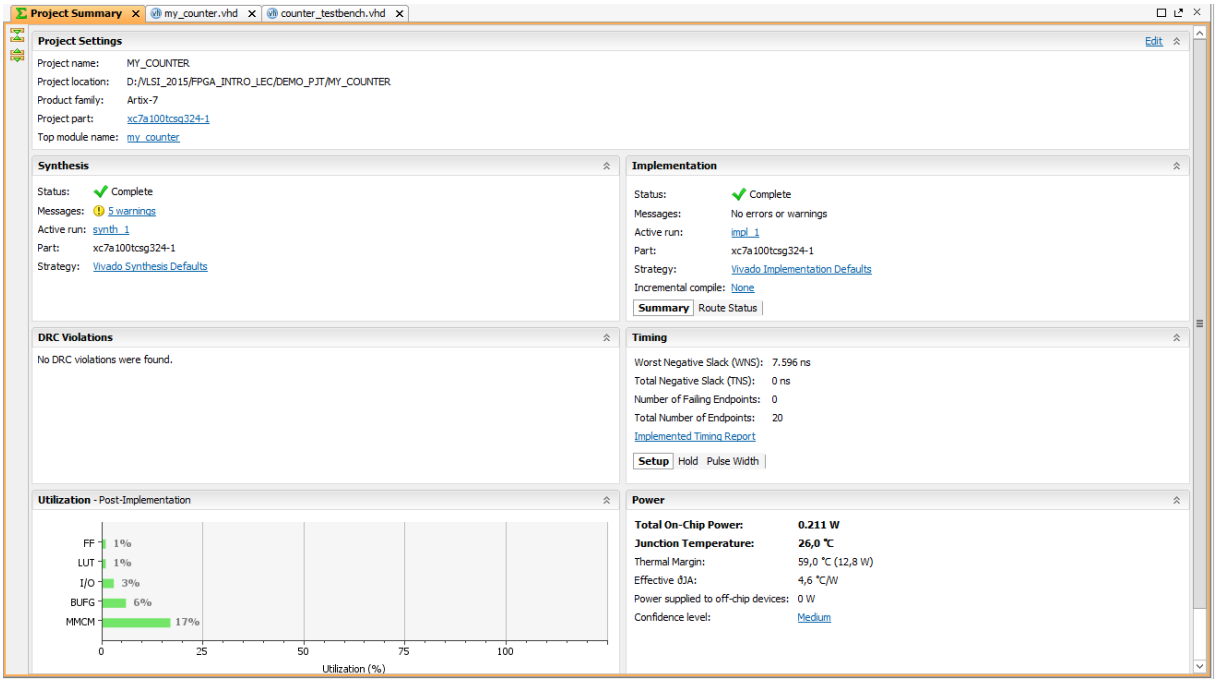

Figure 10: Project summary

Click on “Generate Bitstream” when ready. If bitstream generation is successful a file with the .bit extension will be created in “YOUR_DESIGN_PATH\PROJECT_NAME\PROJECT_NAME.runs\impl_1” folder where PROJECT_NAME is the name of your project.

On successful generation of the .bit file, click on “Open Hardware Manager” to launch the hardware manager to program the FPGA. Make sure that the FPGA is connected to your computer through a USB cable and that the FPGA is powered on.

When you click on the “Open hardware manger” you will get a promt saying “no hardware is connected”. Click on “Open Target” followed by “Open a New Target”. Choose “Local Server” as the FPGA will be connected locally to your machine and click “Next” as shown in Fig.11.

If the FPGA is recognized, a new window showing the name of the FPGA will pop us as shown in Fig.12. Click “Next” and the hardware manager will open with the FPGA connected to the computer.

Right Click on the FPGA, and choose bit file as shown in Fig. 13. You can now program the FPGA and verify that your design works on the FPGA.

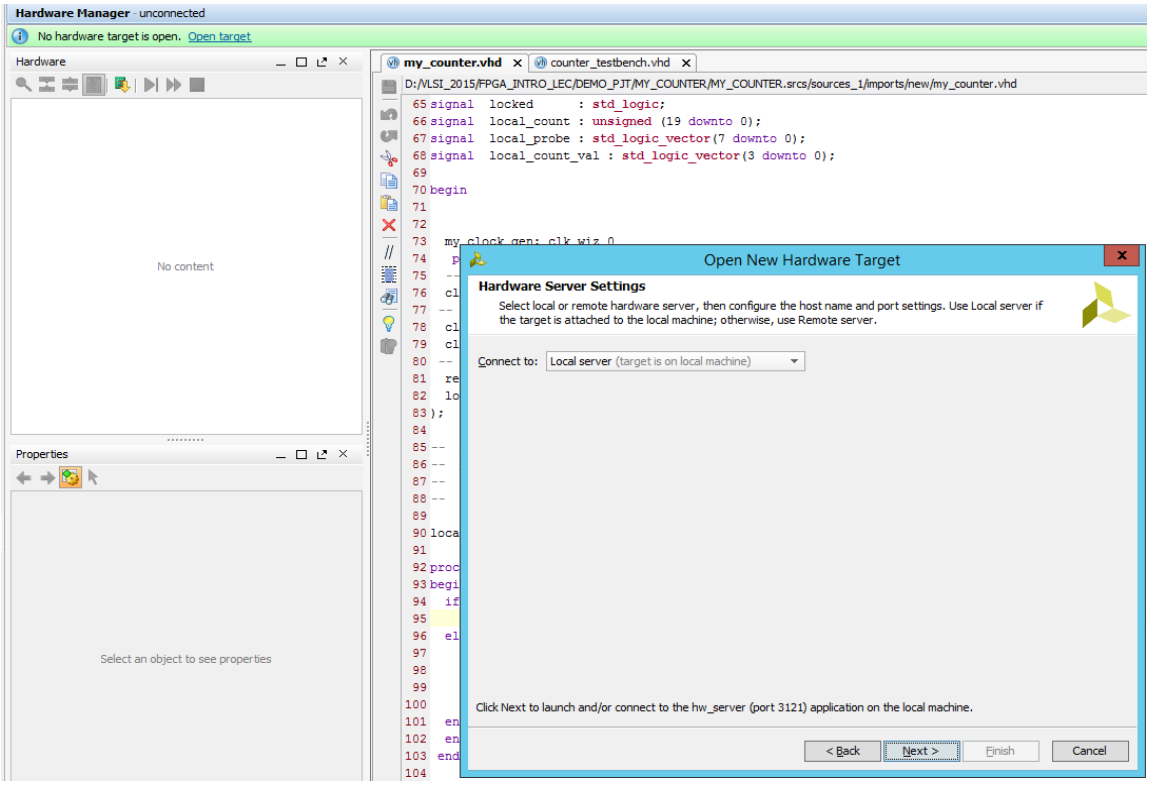

Figure 11: Connect to a hardware target

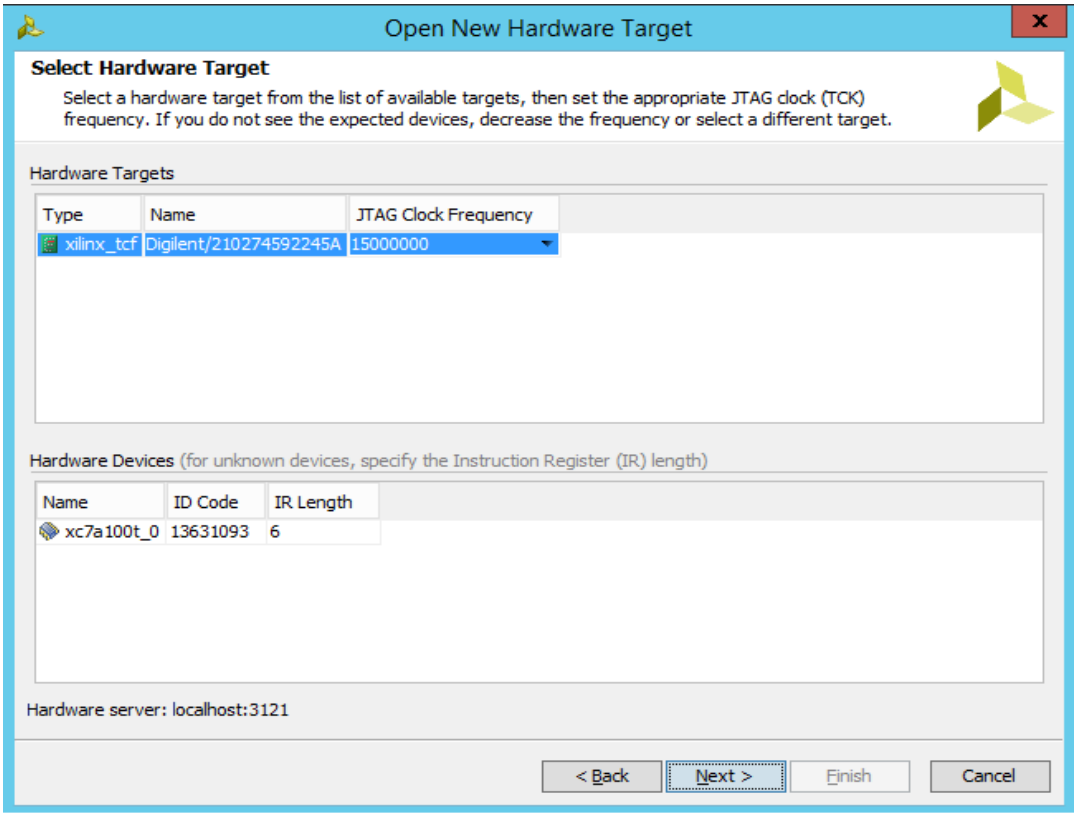

Figure 12: FPGA connected

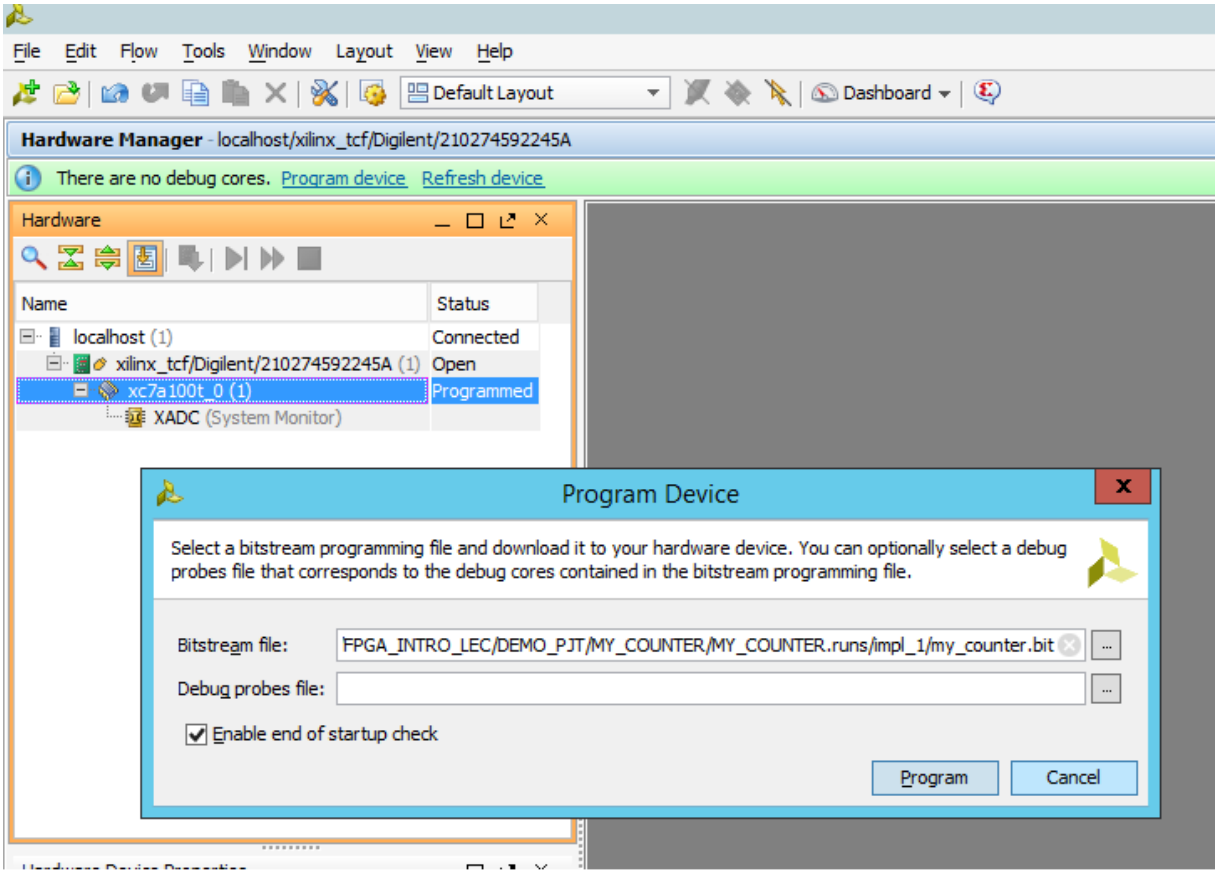

Figure 13: Choose and program bit file