NUS National University of Singapore

Department of Electrical and Computer Engineering

EE4218 EMBEDDED HARDWARE SYSTEMS

A Step-by-Step Guide for VHDL Simulation and FPGA Implmentation Using Xilinx Vivado 2014.2

VHDL Design and Simulation Using Xilinx Vivado 2014.2:
This manual illustrates step-by-step instructions on how to use Xilinx's Vivado 2014.2 software for simulation and synthesis of digital circuits using VHDL.

NOTE:

- **Save all your work in D:\MyWork if you are using a lab PC. Any files saved on the hard drive of the computers in the lab will be cleared on a daily basis.**

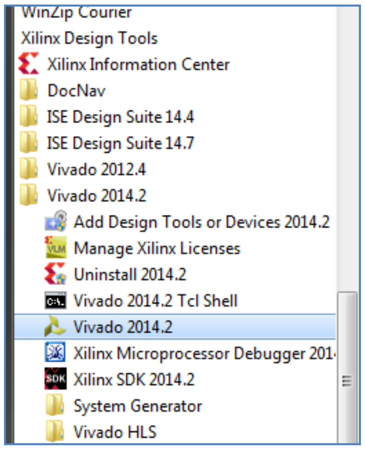
The procedures described in this manual involve 4 major steps:

1. Creating a new VHDL project using Xilinx Vivado 2014.2.
2. Using the VHDL Editor for coding and syntax checking.
3. Simulating the design using VHDL Test Bench and ISE Simulator.
4. FPGA implementation of a VHDL design.

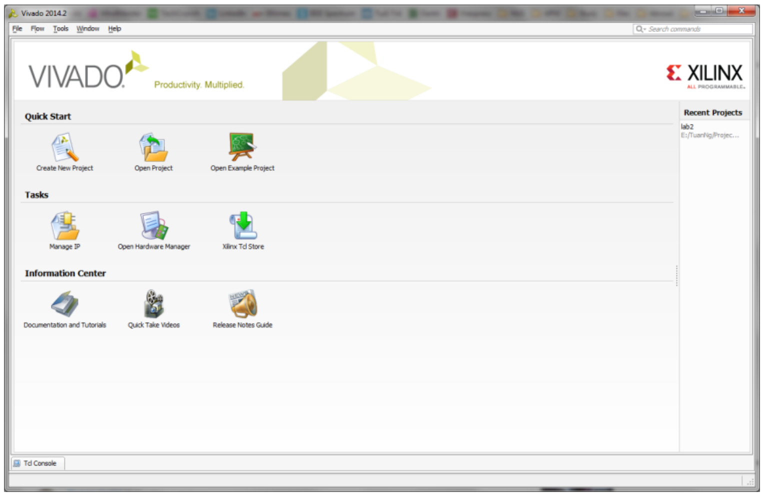
After that, there is **one assignment** specified in **Section 5** and **some notes in Section 6.**

1. Xilinx Vivado: Creating a New VHDL Project

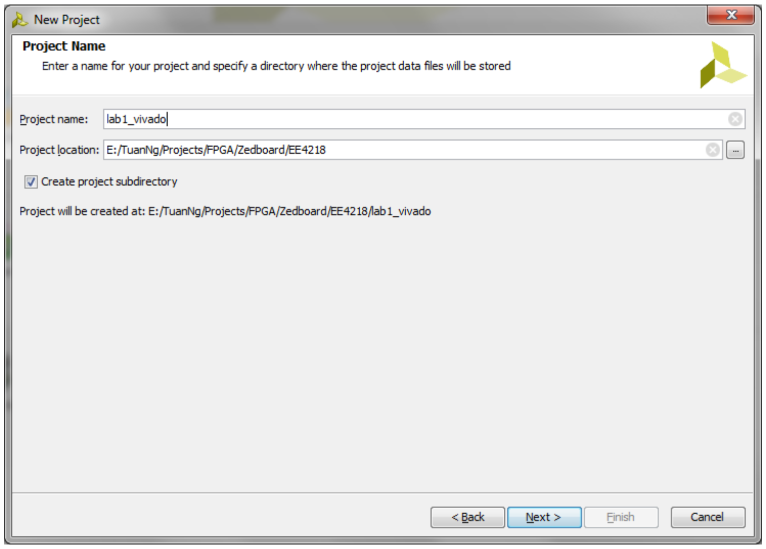
1. You can run the Xilinx Vivado 2014.2 either by clicking the icon on Desktop or going to Start Menu → All Programs → Xilinx Design Tools → Vivado 2014.2 → Vivado 2014.2



2. After launching the ***Vivado***, click on **Create New Project**.
   The ***Create New Project*** window will appear.



3. Click **Next** then enter the **Project Name** and **Project Location**
4. Click **Next** again and make sure the **RTL Project** is selected and the **Do not specify sources at this time** box is checked. Then **Next**



Project Name

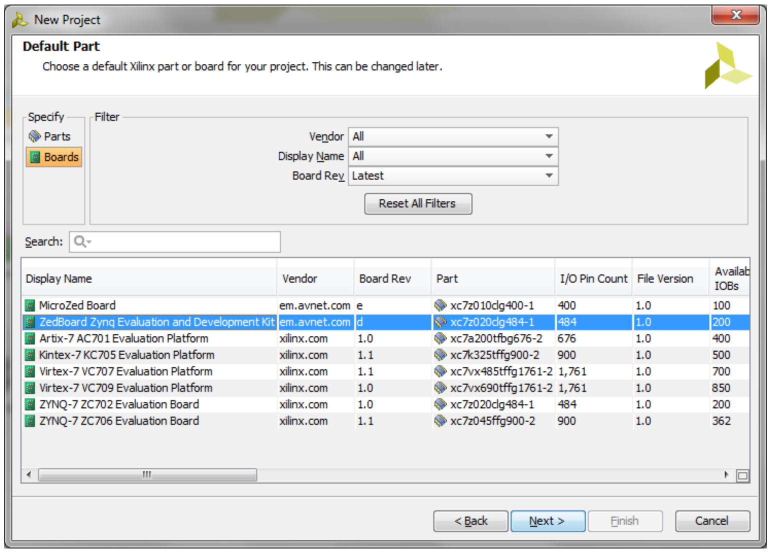
Enter a name for your project and specify a directory where the project data files will be stored

Project name: lab1_vivado

Project location: E:/TuanNg/Projects/FPGA/Zedboard/EE4218

☑ Create project subdirectory

Project will be created at: E:/TuanNg/Projects/FPGA/Zedboard/EE4218/lab1_vivado



New Project

Default Part

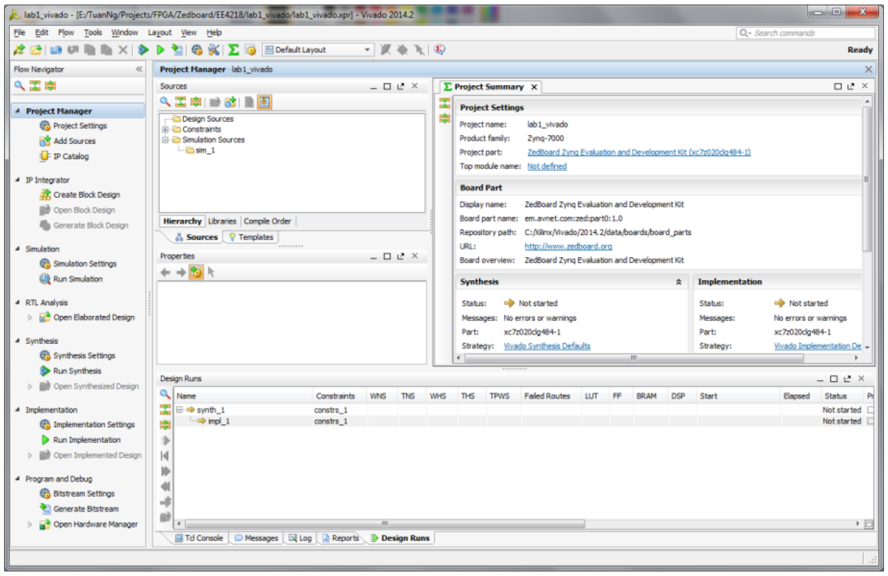
Choose a default Xilinx part or board for your project. This can be changed later.

Specify: Parts, Boards

Filter: Vendor: All; Display Name: All; Board Rev: Latest

Display Name	Vendor	Board Rev	Part	I/O Pin Count	File Version	Available IOBs
MicroZed Board	em.avnet.com	e	xc7z010clg400-1	400	1.0	100
ZedBoard Zynq Evaluation and Development Kit	em.avnet.com	d	xc7z020clg484-1	484	1.0	200
Artix-7 AC701 Evaluation Platform	xilinx.com	1.0	xc7a200tfbg676-2	676	1.0	400
Kintex-7 KC705 Evaluation Platform	xilinx.com	1.1	xc7k325tffg900-2	900	1.0	500
Virtex-7 VC707 Evaluation Platform	xilinx.com	1.1	xc7vx485tffg1761-2	1,761	1.0	700
Virtex-7 VC709 Evaluation Platform	xilinx.com	1.0	xc7vx690tffg1761-2	1,761	1.0	850
ZYNQ-7 ZC702 Evaluation Board	xilinx.com	1.0	xc7z020clg484-1	484	1.0	200
ZYNQ-7 ZC706 Evaluation Board	xilinx.com	1.1	xc7z045ffg900-2	900	1.0	362

5. Now we need to select the FPGA device that we are working on. Since we use the Zedboard which is officially supported by Xilinx, we can select the board from the provided list.
   The board is **Zedboard Zynq Evaluation and Development Kit**.
   Click **Next** to go to **New Project Summary** and **Finish** to create the project.



lab1_vivado - [E:/TuanNg/Projects/FPGA/Zedboard/EE4218/lab1_vivado/lab1_vivado.xpr] - Vivado 2014.2

Project Summary

Project Settings

Project name: lab1_vivado
Product family: Zynq-7000
Project part: ZedBoard Zynq Evaluation and Development Kit (xc7z020clg484-1)
Top module name: Not defined

Board Part

Display name: ZedBoard Zynq Evaluation and Development Kit
Board part name: em.avnet.com:zed:part0:1.0
Repository path: C:/Xilinx/Vivado/2014.2/data/boards/board_parts
URL: http://www.zedboard.org
Board overview: ZedBoard Zynq Evaluation and Development Kit

Synthesis

Status: Not started
Messages: No errors or warnings
Part: xc7z020clg484-1
Strategy: Vivado Synthesis Defaults

Implementation

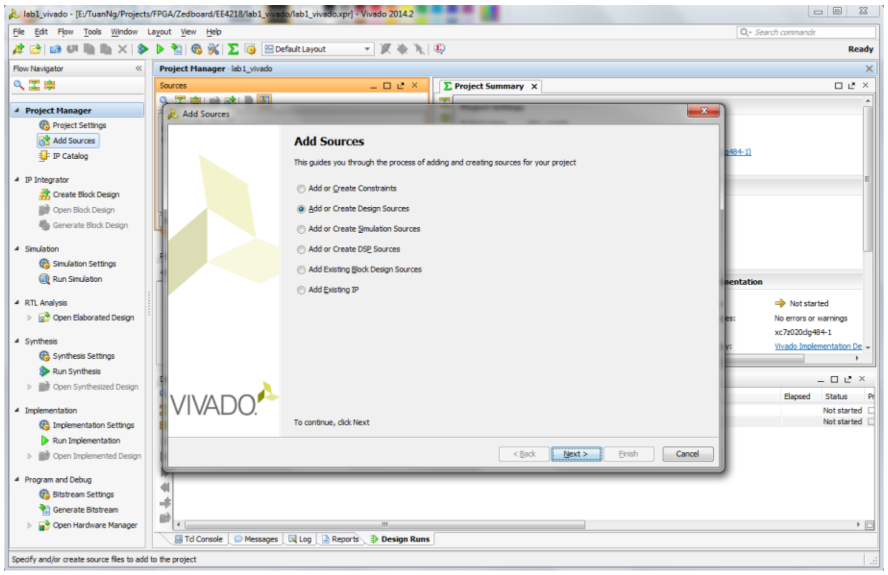
Status: Not started
Messages: No errors or warnings
Part: xc7z020clg484-1
Strategy: Vivado Implementation De

Design Runs

Name	Constraints	WNS	TNS	WHS	THS	TPWS	Failed Routes	LUT	FF	BRAM	DSP	Start	Elapsed	Status
synth_1	constrs_1													Not started
impl_1	constrs_1													Not started

6. This is the main workspace of the project that we have just created.
   The left pane, **Flow Navigator**, lists all the tools we need to implement the system from making design files to simulating the system, generating the bitfile and finally, downloading it to the device.
   The **Sources** window shows all the files used in the project.

2. Xilinx VHDL Editor: Adding Sources – Programming – Syntax Checking



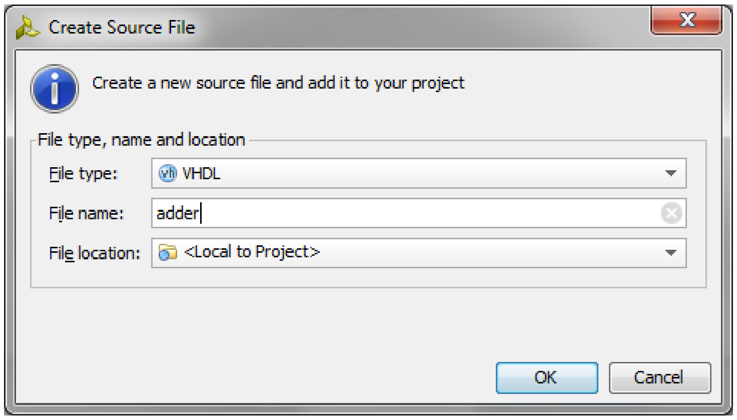
Add Sources

This guides you through the process of adding and creating sources for your project

- ( ) Add or Create Constraints
- [✓ Add or Create Design Sources]
- ( ) Add or Create Simulation Sources
- ( ) Add or Create DSP Sources
- ( ) Add Existing Block Design Sources
- ( ) Add Existing IP

To continue, click Next

1. Click on **Add Sources** under **Project Manager** in the **Flow Navigator** pane to add the design sources.
   Make sure that the **Add or Create Design Sources** is selected.



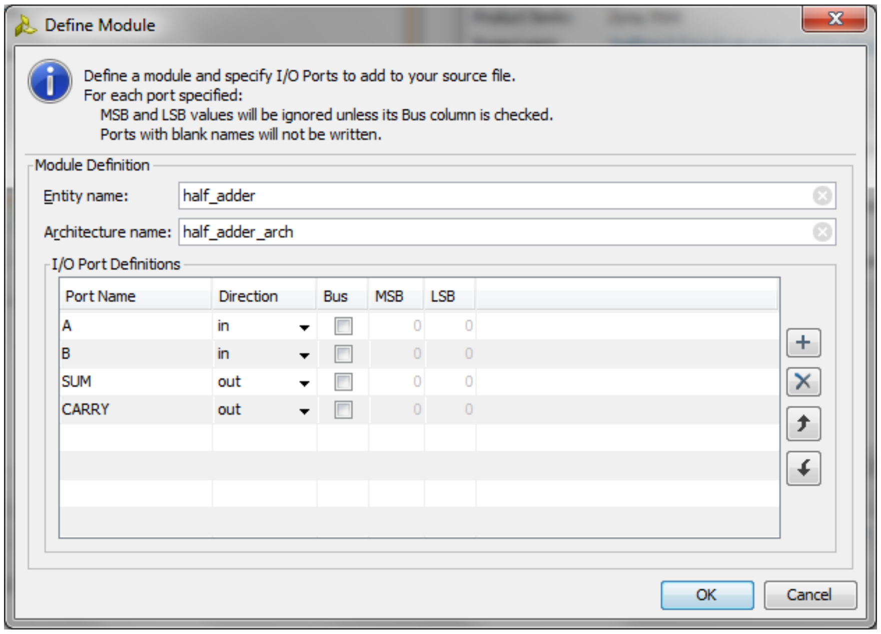
Create Source File

Create a new source file and add it to your project

File type, name and location

File type:	VHDL
File name:	adder
File location:	<Local to Project>

2. Click **Next** then **Create File.**
   Select **VHDL** in the drop down list.
   Specify the desired file name, for example, **adder**
   Click **OK** then **Finish**



Define Module

Define a module and specify I/O Ports to add to your source file.
For each port specified:
MSB and LSB values will be ignored unless its Bus column is checked.
Ports with blank names will not be written.

Module Definition

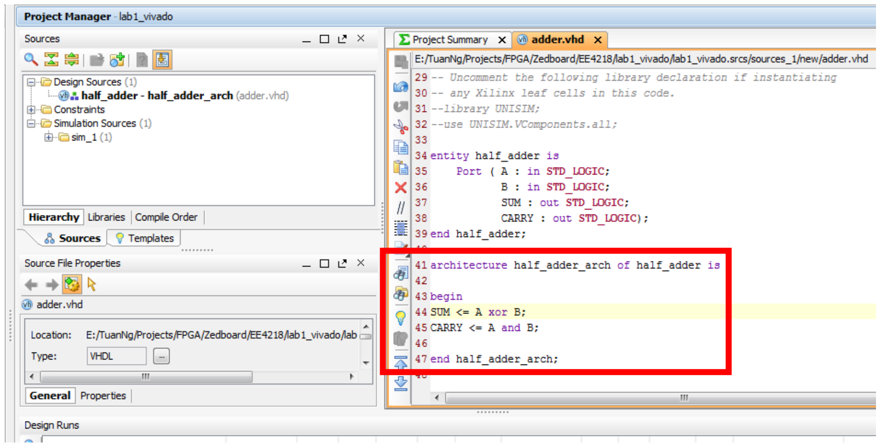
Entity name: half_adder

Architecture name: half_adder_arch

I/O Port Definitions

Port Name	Direction	Bus	MSB	LSB
A	in	☐	0	0
B	in	☐	0	0
SUM	out	☐	0	0
CARRY	out	☐	0	0

3. After that, we define the module as in the picture on the left.
   Since this is the simple half adder, the input and output ports are only one bit.
   Click **OK** to generate the VHDL file.

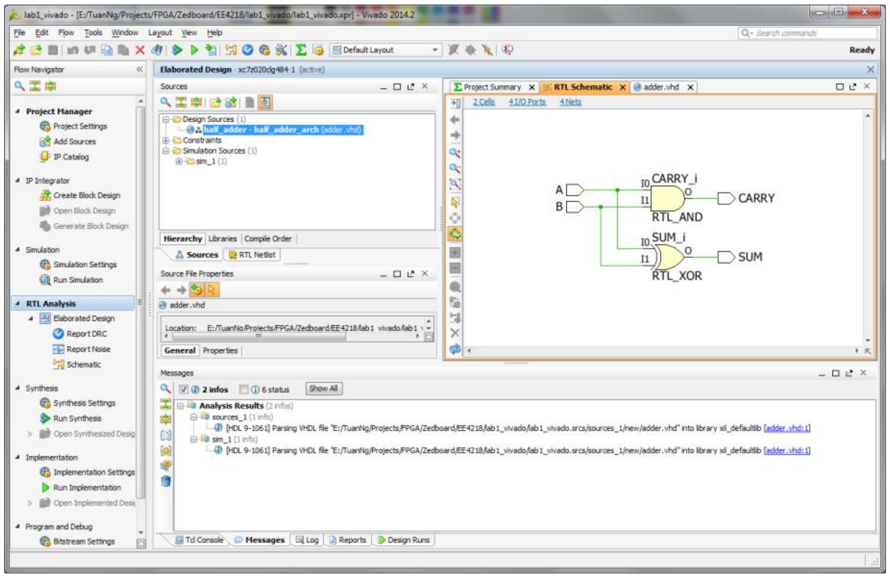


E:/TuanNg/Projects/FPGA/Zedboard/EE4218/lab1_vivado/lab1_vivado.srcs/sources_1/new/adder.vhd

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

entity half_adder is
    Port ( A : in STD_LOGIC;
           B : in STD_LOGIC;
           SUM : out STD_LOGIC;
           CARRY : out STD_LOGIC);
end half_adder;

architecture half_adder_arch of half_adder is

begin
SUM <= A xor B;
CARRY <= A and B;

end half_adder_arch;
```

4. New the newly created file is added to the project under **Design Sources.** Double click the file to open it and add the following lines to the architecture specification of the entity the save the file.

   SUM <= A xor B;
   CARRY <= A and B;



lab1_vivado - [E:/TuanNg/Projects/FPGA/Zedboard/EE4218/lab1_vivado/lab1_vivado.xpr] - Vivado 2014.2

Elaborated Design - xc7z020clg484-1 (active)

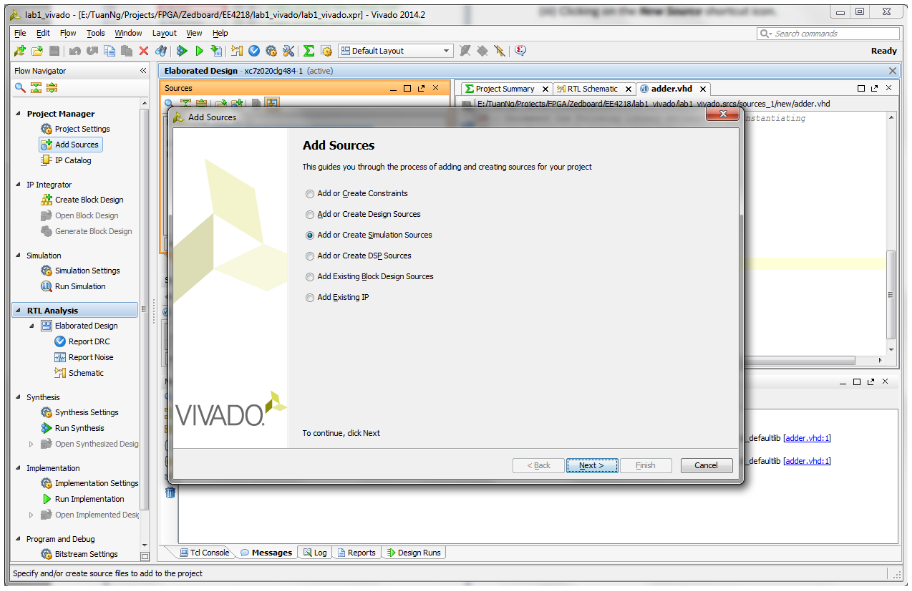
Messages

Analysis Results (2 infos)
- sources_1 (1 info)
  - [HDL 9-1061] Parsing VHDL file "E:/TuanNg/Projects/FPGA/Zedboard/EE4218/lab1_vivado/lab1_vivado.srcs/sources_1/new/adder.vhd" into library xil_defaultlib [adder.vhd:1]
- sim_1 (1 info)
  - [HDL 9-1061] Parsing VHDL file "E:/TuanNg/Projects/FPGA/Zedboard/EE4218/lab1_vivado/lab1_vivado.srcs/sources_1/new/adder.vhd" into library xil_defaultlib [adder.vhd:1]

5. Basically, Vivado checks the syntax of VHDL code whenever you save the file. If you want further analysis of the design files without going through the lengthy complete synthesis process, you can click **Open Elaborated Design** under **RTL Analysis** in the **Flow Navigator**. It will also show the schematic of the design if there is no problem with it.

3. A) Behavioural Simulation: Using VHDL Test Bench for Combinational Circuits

Behavioural simulation is performed before actual design implementation on hardware to verify that the logic created is correct.



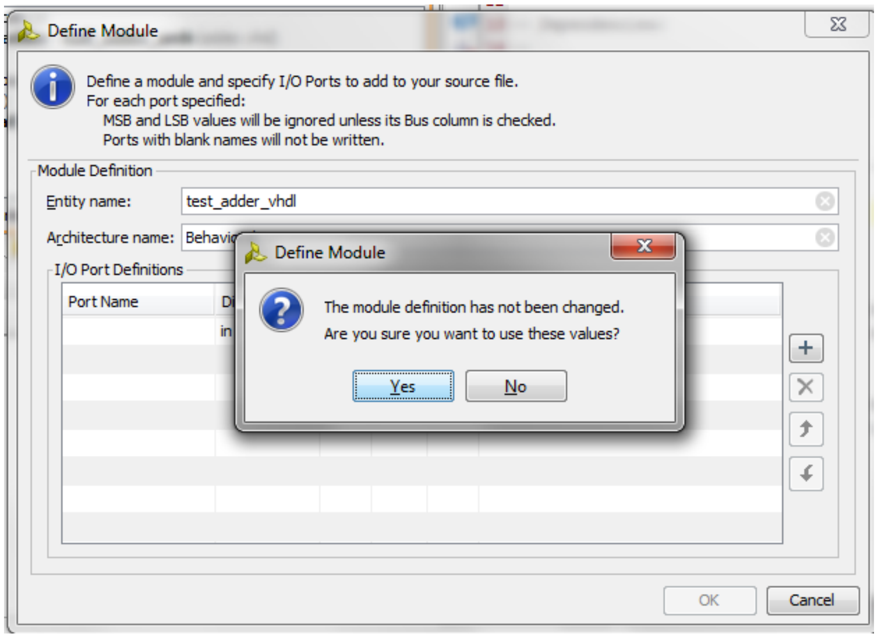
Add Sources

This guides you through the process of adding and creating sources for your project

( ) Add or Create Constraints
( ) Add or Create Design Sources
(●) Add or Create Simulation Sources
( ) Add or Create DSP Sources
( ) Add Existing Block Design Sources
( ) Add Existing IP

To continue, click Next

1. Click **Add Sources** and select **Add or Create Simulation Sources.**
Click **Next** then **Create File** to create the new VHDL file.
Choose the **VHDL** file type and specify the name of the testbench, **test_adder_vhdl.**
Click **OK** then **Finish**.



Define Module

Define a module and specify I/O Ports to add to your source file.
For each port specified:
MSB and LSB values will be ignored unless its Bus column is checked.
Ports with blank names will not be written.

Module Definition

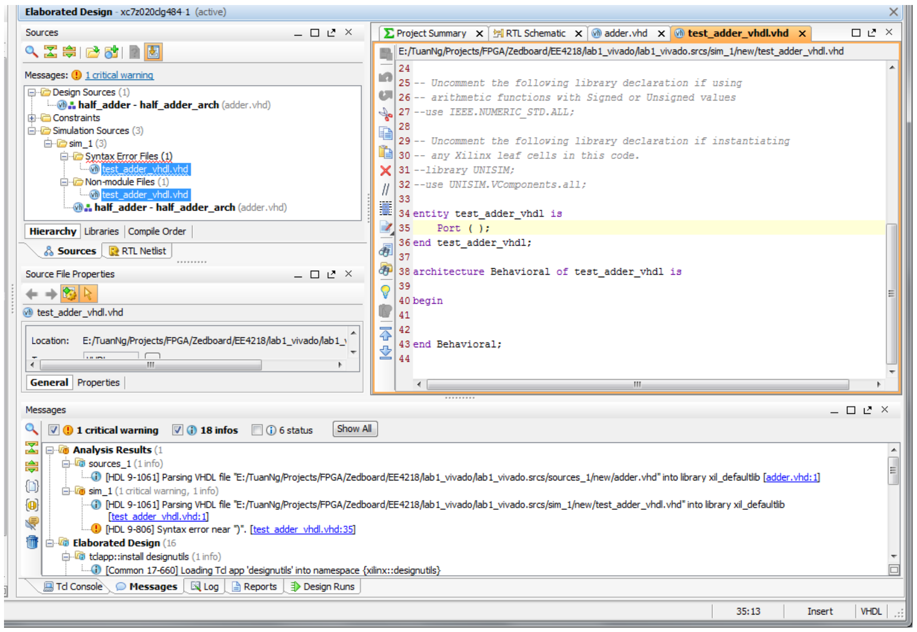
Entity name: test_adder_vhdl

Architecture name: Behavi

Define Module

The module definition has not been changed.
Are you sure you want to use these values?

2. The **Define Module** will appear but we do not need to define the I/O ports because this is the testbench.
Click **OK** then **Yes** to create the file.

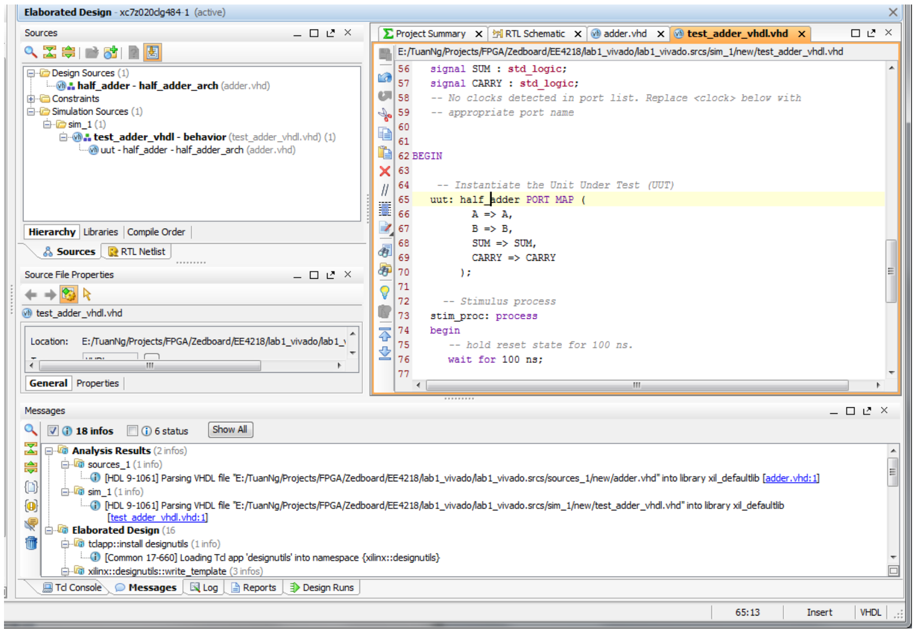


E:/TuanNg/Projects/FPGA/Zedboard/EE4218/lab1_vivado/lab1_vivado.srcs/sim_1/new/test_adder_vhdl.vhd

```
-- 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 test_adder_vhdl is
    Port ( );
end test_adder_vhdl;

architecture Behavioral of test_adder_vhdl is

begin


end Behavioral;
```

```
1 critical warning   18 infos   6 status
Analysis Results (1
  sources_1 (1 info)
    [HDL 9-1061] Parsing VHDL file "E:/TuanNg/Projects/FPGA/Zedboard/EE4218/lab1_vivado/lab1_vivado.srcs/sources_1/new/adder.vhd" into library xil_defaultlib [adder.vhd:1]
  sim_1 (1 critical warning, 1 info)
    [HDL 9-1061] Parsing VHDL file "E:/TuanNg/Projects/FPGA/Zedboard/EE4218/lab1_vivado/lab1_vivado.srcs/sim_1/new/test_adder_vhdl.vhd" into library xil_defaultlib
      [test_adder_vhdl.vhd:1]
    [HDL 9-806] Syntax error near ")". [test_adder_vhdl.vhd:35]
Elaborated Design (16
  tclapp::install designutils (1 info)
    [Common 17-660] Loading Tcl app 'designutils' into namespace {xilinx::designutils}
```

3. Then you see that there is a Syntax Error in the just created file. Try to open the **Messages** tab and locate the error.

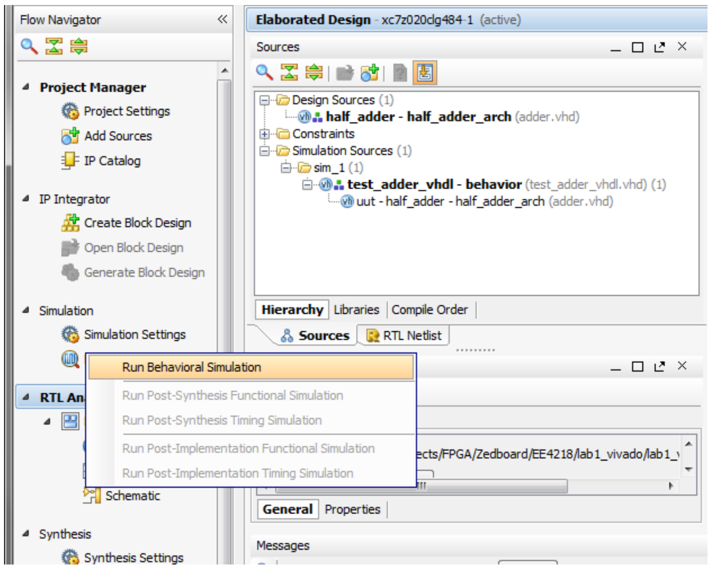


E:/TuanNg/Projects/FPGA/Zedboard/EE4218/lab1_vivado/lab1_vivado.srcs/sim_1/new/test_adder_vhdl.vhd

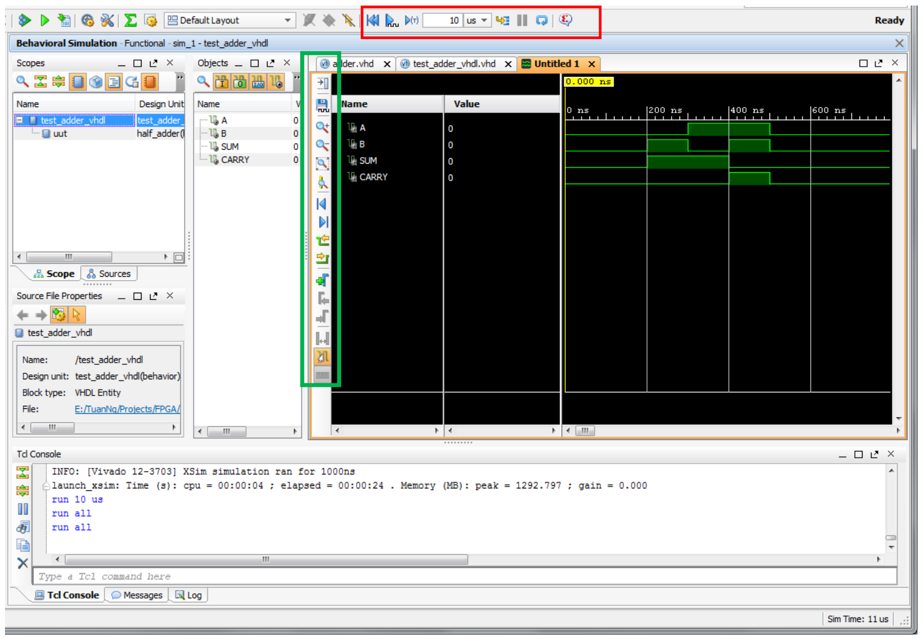
```
  signal SUM : std_logic;
  signal CARRY : std_logic;
  -- No clocks detected in port list. Replace <clock> below with
  -- appropriate port name


BEGIN

   -- Instantiate the Unit Under Test (UUT)
  uut: half_adder PORT MAP (
         A => A,
         B => B,
         SUM => SUM,
         CARRY => CARRY
       );

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

```

```
18 infos   6 status
Analysis Results (2 infos)
  sources_1 (1 info)
    [HDL 9-1061] Parsing VHDL file "E:/TuanNg/Projects/FPGA/Zedboard/EE4218/lab1_vivado/lab1_vivado.srcs/sources_1/new/adder.vhd" into library xil_defaultlib [adder.vhd:1]
  sim_1 (1 info)
    [HDL 9-1061] Parsing VHDL file "E:/TuanNg/Projects/FPGA/Zedboard/EE4218/lab1_vivado/lab1_vivado.srcs/sim_1/new/test_adder_vhdl.vhd" into library xil_defaultlib
      [test_adder_vhdl.vhd:1]
Elaborated Design (16
  tclapp::install designutils (1 info)
    [Common 17-660] Loading Tcl app 'designutils' into namespace {xilinx::designutils}
  xilinx::designutils::write_template (3 infos)
```

4. Make the testbench for the **half_adder** module we've just created.
You can make a testbench by yourself or use any other tool to make a template for you base on the I/O interfaces of your design.
Add the following stimuli to the VHDL test bench code:
A <= '0';
B <= '0';
wait for 100ns;
A <= '0';
B <= '1';
wait for 100ns;
A <= '1';
B <= '0';
wait for 100ns;
A <= '1';
B <= '1';
wait for 100ns;
A <= '0';
B <= '0';


Flow Navigator
Project Manager
Project Settings
Add Sources
IP Catalog
IP Integrator
Create Block Design
Open Block Design
Generate Block Design
Simulation
Simulation Settings
Run Behavioral Simulation
Run Post-Synthesis Functional Simulation
Run Post-Synthesis Timing Simulation
Run Post-Implementation Functional Simulation
Run Post-Implementation Timing Simulation
Schematic
Synthesis
Synthesis Settings
Elaborated Design - xc7z020clg484-1 (active)
Sources
Design Sources (1)
half_adder - half_adder_arch (adder.vhd)
Constraints
Simulation Sources (1)
sim_1 (1)
test_adder_vhdl - behavior (test_adder_vhdl.vhd) (1)
uut - half_adder - half_adder_arch (adder.vhd)
Hierarchy
Libraries
Compile Order
Sources
RTL Netlist
General
Properties
Messages

5. Now, we can simulate our design.
Click **Run Simulation** under **Simulation** then **Run Behavioral Simulation.**

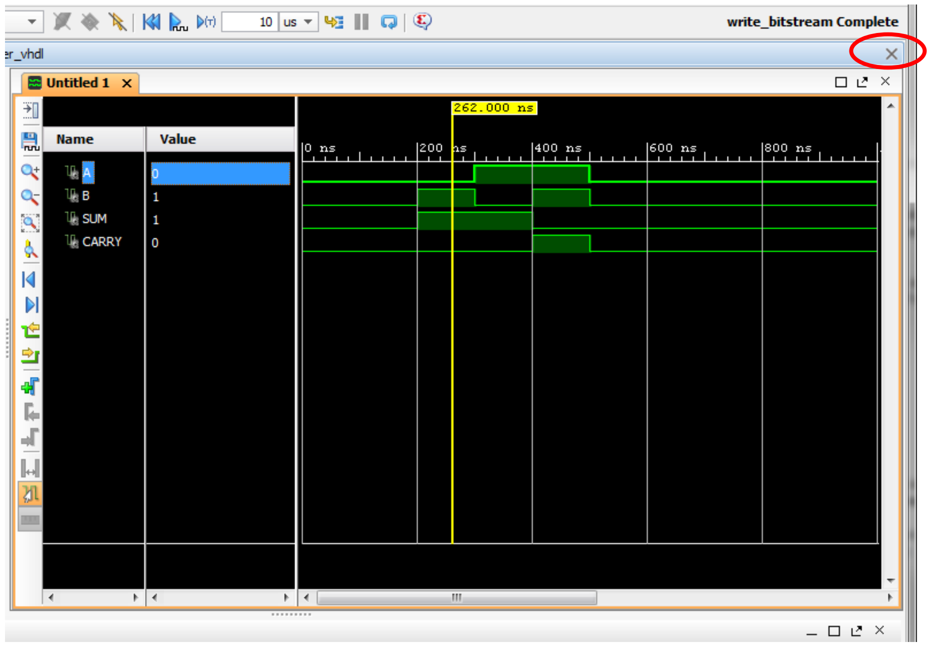


```
INFO: [Vivado 12-3703] XSim simulation ran for 1000ns
launch_xsim: Time (s): cpu = 00:00:04 ; elapsed = 00:00:24 . Memory (MB): peak = 1292.797 ; gain = 0.000
run 10 us
run all
run all
```

6. The waveform tab will be opened. Try to play with the tools inside the green/red boxes to navigate around the waveform.

   Initially, you may not see the waveform as on the figure because the time resolution is too high, **ps** in this case.

   You need to switch to full view. Select **View → Zoom Fit** or the **Zoom Fit** icon inside the green box.

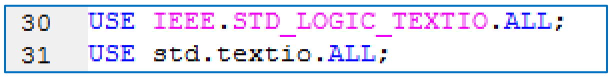


7. You can click on any place inside the waveform to move the yellow vertical line along the timeline, the corresponding values of input and output are shown in the **Value** column.
   a and ***b*** are the stimuli value, in other words, the input.
   sum and ***carry*** are the results of the stimuli, in other words, the output.
   To exit the Simulation, click the **cross** icon as shown in red circle.

B) VHDL Test Bench using TEXTIO

This section illustrates using text files for giving stimuli for the test bench. This enables fast verification of complex designs where a number of cases have to be tested.

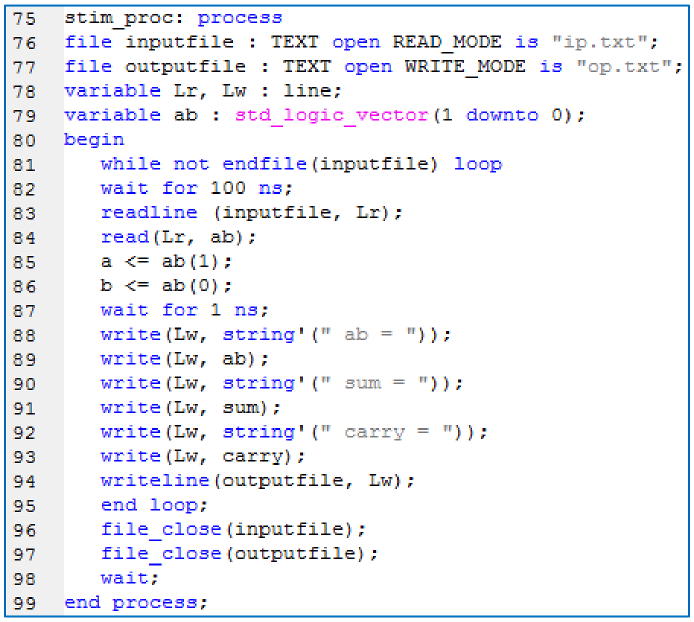
1. Add the 2 lines to the library declarations



```
USE IEEE.STD_LOGIC_TEXTIO.ALL;
USE std.textio.ALL;
```

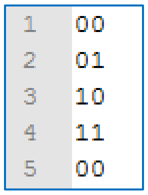
2. Change the stimulus process (from section 3.A) to the following

```
stim_proc: process
file inputfile : TEXT open READ_MODE is "ip.txt";
file outputfile : TEXT open WRITE_MODE is "op.txt";
variable Lr, Lw : line;
variable ab : std_logic_vector(1 downto 0);
begin
   while not endfile(inputfile) loop
            wait for 100 ns;
            readline (inputfile, Lr);
            read(Lr, ab);
            a <= ab(1);
            b <= ab(0);
            wait for 1 ns;
            write(Lw, string'(" ab = "));
   write(Lw, ab);
   write(Lw, string'(" sum = "));
            write(Lw, sum);
            write(Lw, string'(" carry = "));
            write(Lw, carry);
            writeline(outputfile, Lw);
   end loop;
   file_close(inputfile);
   file_close(outputfile);
   wait;
end process;
```



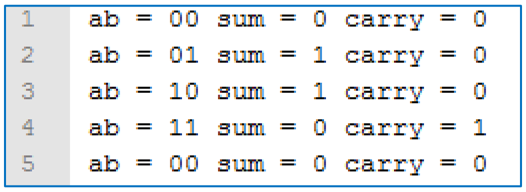
```
stim_proc: process
file inputfile : TEXT open READ_MODE is "ip.txt";
file outputfile : TEXT open WRITE_MODE is "op.txt";
variable Lr, Lw : line;
variable ab : std_logic_vector(1 downto 0);
begin
   while not endfile(inputfile) loop
   wait for 100 ns;
   readline (inputfile, Lr);
   read(Lr, ab);
   a <= ab(1);
   b <= ab(0);
   wait for 1 ns;
   write(Lw, string'(" ab = "));
   write(Lw, ab);
   write(Lw, string'(" sum = "));
   write(Lw, sum);
   write(Lw, string'(" carry = "));
   write(Lw, carry);
   writeline(outputfile, Lw);
   end loop;
   file_close(inputfile);
   file_close(outputfile);
   wait;
end process;
```

3. Create a text file ip.txt using a text editor (eg: Notepad) with the following contents, and save it in your project folder as ip.txt.



```
00
01
10
11
00
```

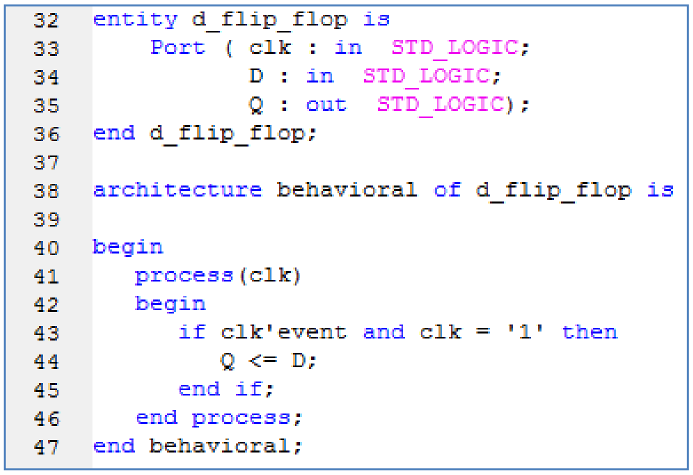
4. Proceed with the simulation (See step 5 onwards in section 3A). You should be able to see the simulation window identical to that in section 3.A.
Also, in the project folder, you should be able to see a file **op.txt** with the contents as shown in the figure.



```
ab = 00 sum = 0 carry = 0
ab = 01 sum = 1 carry = 0
ab = 10 sum = 1 carry = 0
ab = 11 sum = 0 carry = 1
ab = 00 sum = 0 carry = 0
```

C) VHDL Test Bench for Clocked Circuits

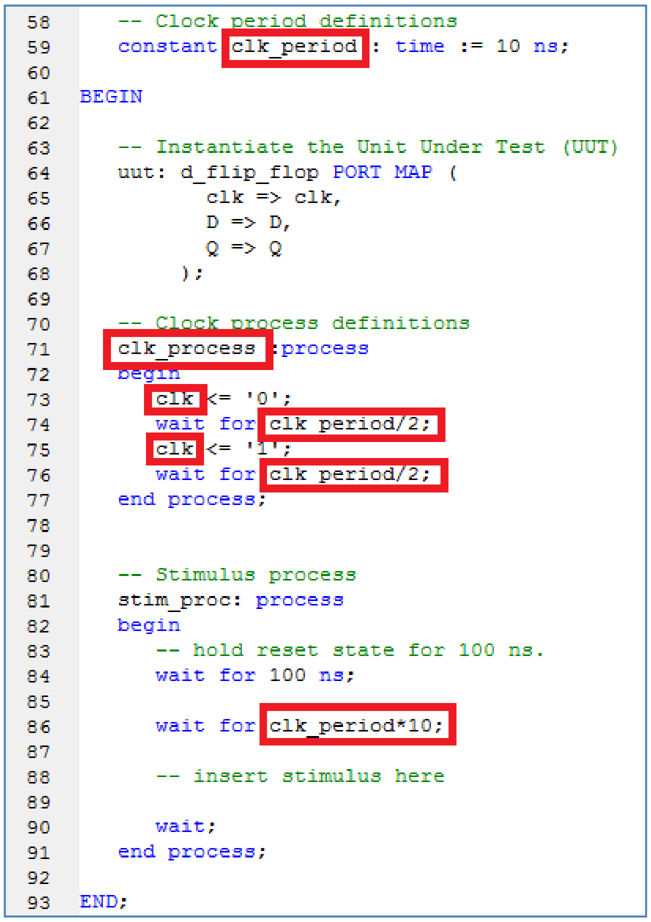
This section will show how to write a VHDL Test Bench for a clocked circuit, illustrated using a positive edge triggered D Flip-Flop



```
entity d_flip_flop is
    Port ( clk : in  STD_LOGIC;
           D : in  STD_LOGIC;
           Q : out  STD_LOGIC);
end d_flip_flop;

architecture behavioral of d_flip_flop is

begin
   process(clk)
   begin
      if clk'event and clk = '1' then
         Q <= D;
      end if;
   end process;
end behavioral;
```

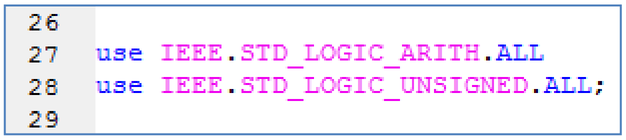
1. Similar to step **Error! Reference source not found.** to 3 of section **Error! Reference source not found.**, create the VHDL code for a D-flip-flop. The VHDL codes will look like:

```
entity d_flip_flop is
    Port ( clk : in  STD_LOGIC;
           D : in  STD_LOGIC;
           Q : out  STD_LOGIC);
end d_flip_flop;

architecture behavioral of d_flip_flop is

begin
   process(clk)
   begin
      if clk'event and clk = '1' then
         Q <= D;
      end if;
   end process;
end behavioral;
```



```
   -- Clock period definitions
   constant clk_period : time := 10 ns;

BEGIN

   -- Instantiate the Unit Under Test (UUT)
   uut: d_flip_flop PORT MAP (
          clk => clk,
          D => D,
          Q => Q
        );

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


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

      wait for clk_period*10;

      -- insert stimulus here

      wait;
   end process;

END;
```

2. After the syntax has been determined to be correct, create a VHDL Test Bench for this ***d_flip_flop***. The steps are similar to steps **Error! Reference source not found.** to **Error! Reference source not found.** of section 3.
   In this case, we need to make a **clk_process** to generate the clock to the design.
   Please use the clock period of **100 ns**.

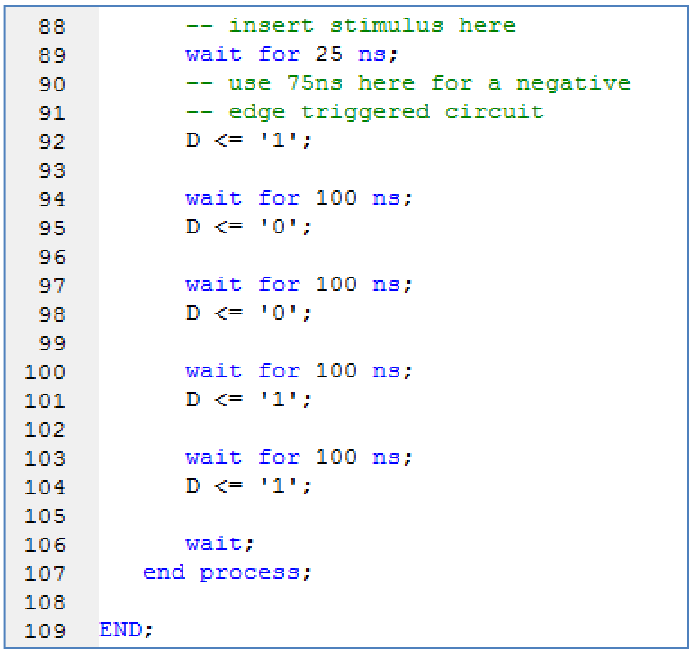


```

use IEEE.STD_LOGIC_ARITH.ALL
use IEEE.STD_LOGIC_UNSIGNED.ALL;

```

3. If mathematical operations are being used for giving stimuli in an automated fashion, the following two libraries might have to be included as well:

   use IEEE.STD_LOGIC_ARITH.ALL
   use IEEE.STD_LOGIC_UNSIGNED.ALL;

   In the example described in this section, the stimuli given does not require the use of the above two libraries.



```
        -- insert stimulus here
        wait for 25 ns;
        -- use 75ns here for a negative
        -- edge triggered circuit
        D <= '1';

        wait for 100 ns;
        D <= '0';

        wait for 100 ns;
        D <= '0';

        wait for 100 ns;
        D <= '1';

        wait for 100 ns;
        D <= '1';

        wait;
    end process;

END;
```

4. Insert the following stimuli in the VHDL test bench

   wait for 25 ns;
   D <= '1';

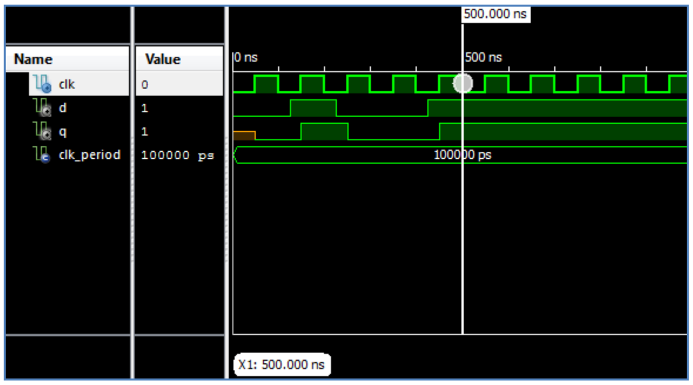
   wait for 100 ns;
   D <= '0';

   wait for 100 ns;
   D <= '0';

   wait for 100 ns;
   D <= '1';

   wait for 100 ns;
   D <= '1';

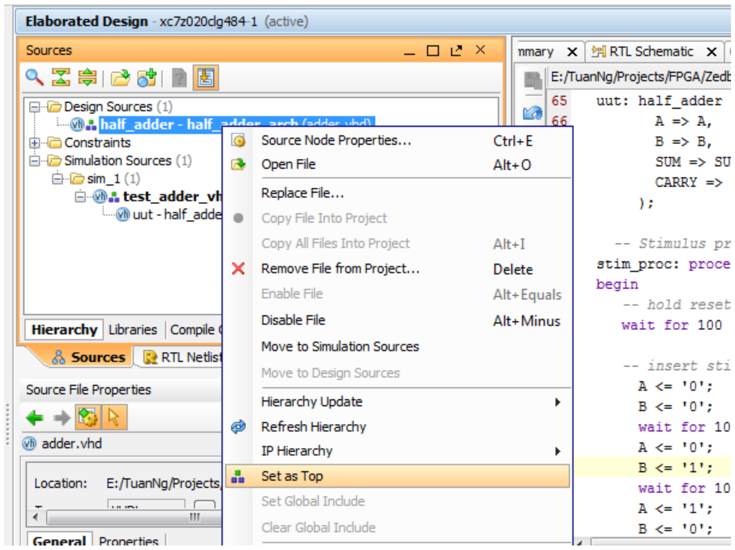
   Note that the ***wait for 25 ns*** should preferably be ***wait for 75 ns*** if negative edge triggered circuits are used. If there is more than one input, these inputs can be changed simultaneously in between the stimuli. In this example, D is the only input that is being changed in between stimuli.



5. Do the Behavioral Simulation similar to Step 5 onwards of Section 3A to simulate the design.
   In this figure, the D-flip-flop is triggered at every positive clock cycle

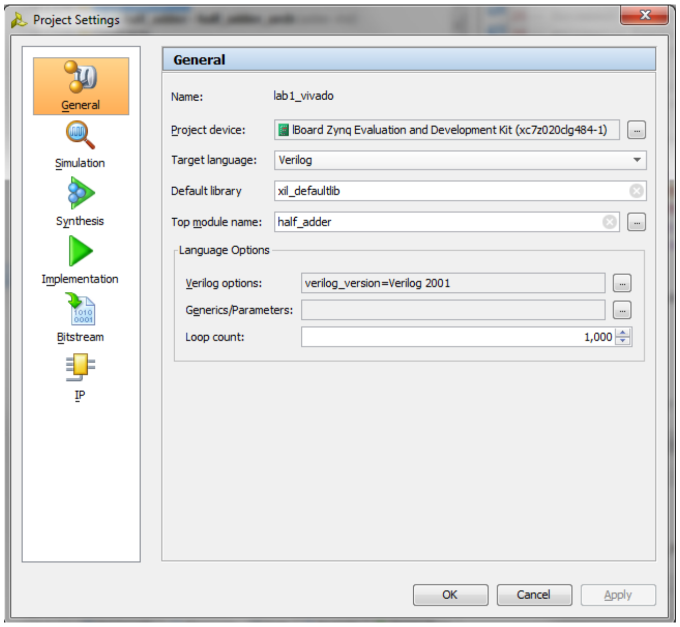
4. FPGA Implementation: Half-Adder

This section describes how to implement the design made in section 1 on an actual FPGA hardware.



```
uut: half_adder
        A => A,
        B => B,
        SUM => SU
        CARRY =>
     );

   -- Stimulus pr
   stim_proc: proce
   begin
      -- hold reset
      wait for 100

      -- insert sti
        A <= '0';
        B <= '0';
        wait for 10
        A <= '0';
        B <= '1';
        wait for 10
        A <= '1';
        B <= '0';
```

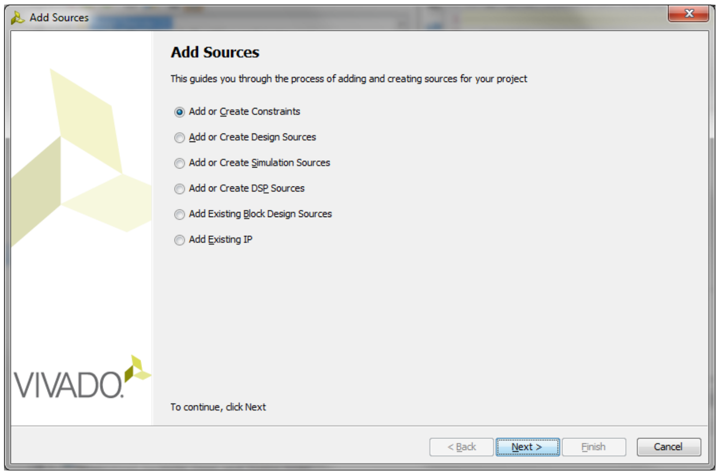
1. First, make sure that the **half_adder** entity is selected as **Top.**
   Check whether the icon is shown next to the design file.
   If not, right click at choose **Set as Top**.



General

Name: lab1_vivado
Project device: IBoard Zynq Evaluation and Development Kit (xc7z020clg484-1)
Target language: Verilog
Default library: xil_defaultlib
Top module name: half_adder
Verilog options: verilog_version=Verilog 2001
Generics/Parameters:
Loop count: 1,000

2. Then click **Project Settings** in the Flow Navigator to to verify that the correct device is selected.
   This is very important because the configuration file generated by Vivado is unique for each device.


Add Sources

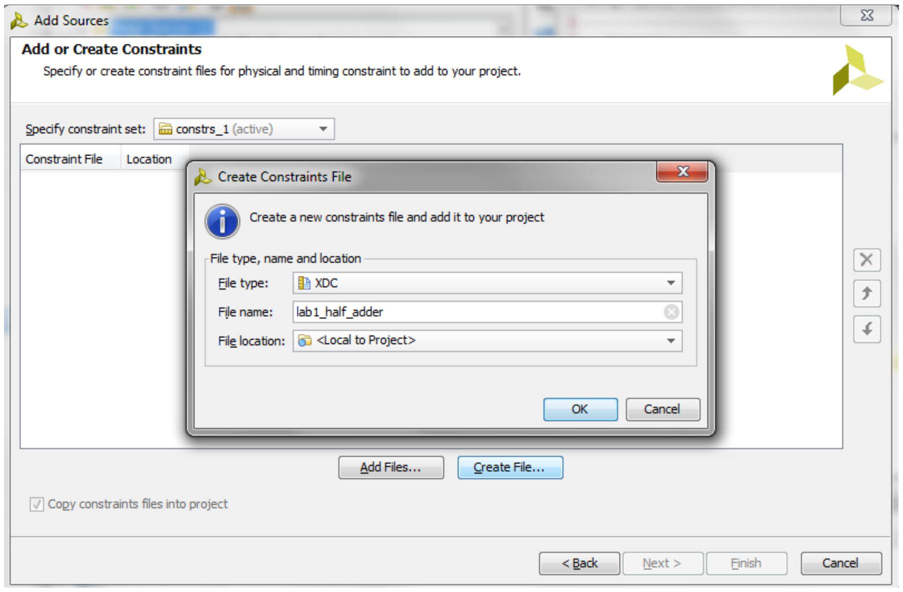
Add Sources

This guides you through the process of adding and creating sources for your project

- [✓ Add or Create Constraints]
- Add or Create Design Sources
- Add or Create Simulation Sources
- Add or Create DSP Sources
- Add Existing Block Design Sources
- Add Existing IP

To continue, click Next

3. We need to add a constraint file to tell Vivado where we want to map the input/output ports of our design to the actual physical I/O of the FPGA chip.
   Click **Add Sources** and select **Add or Create Constraints**.


Add Sources

Add or Create Constraints

Specify or create constraint files for physical and timing constraint to add to your project.

Specify constraint set: constrs_1 (active)

Create Constraints File

Create a new constraints file and add it to your project

File type, name and location

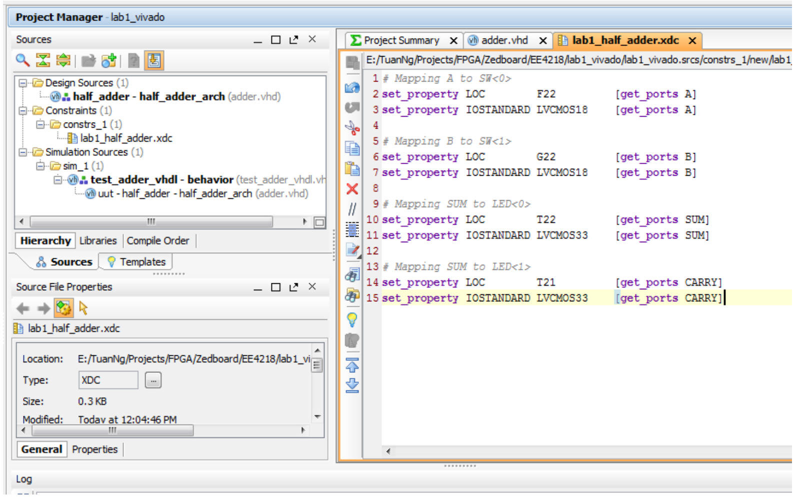
File type: XDC

File name: lab1_half_adder

File location: <Local to Project>

[✓] Copy constraints files into project

4. Specify the configurations as the picture on the left. And click **Finish** to generate the file.


Project Manager - lab1_vivado

lab1_half_adder.xdc

E:/TuanNg/Projects/FPGA/Zedboard/EE4218/lab1_vivado/lab1_vivado.srcs/constrs_1/new/lab1

```
# Mapping A to SW<0>
set_property LOC        F22         [get_ports A]
set_property IOSTANDARD LVCMOS18    [get_ports A]

# Mapping B to SW<1>
set_property LOC        G22         [get_ports B]
set_property IOSTANDARD LVCMOS18    [get_ports B]

# Mapping SUM to LED<0>
set_property LOC        T22         [get_ports SUM]
set_property IOSTANDARD LVCMOS33    [get_ports SUM]

# Mapping SUM to LED<1>
set_property LOC        T21         [get_ports CARRY]
set_property IOSTANDARD LVCMOS33    [get_ports CARRY]
```

Location: E:/TuanNg/Projects/FPGA/Zedboard/EE4218/lab1_vi

Type: XDC

Size: 0.3 KB

Modified: Today at 12:04:46 PM

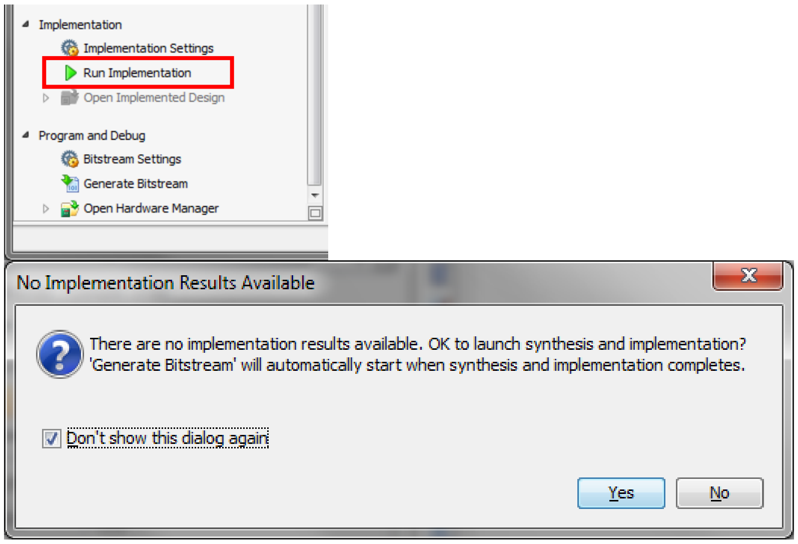
5. Now we can see the new file **lab1_half_adder.xdc** under the **Constraints.**
   Double click to open it, add the constraints below and save it.

# Mapping A to SW<0>
set_property LOC F22 [get_ports A]
set_property IOSTANDARD LVCMOS18 [get_ports A]

# Mapping B to SW<1>
set_property LOC G22 [get_ports B]
set_property IOSTANDARD LVCMOS18 [get_ports B]

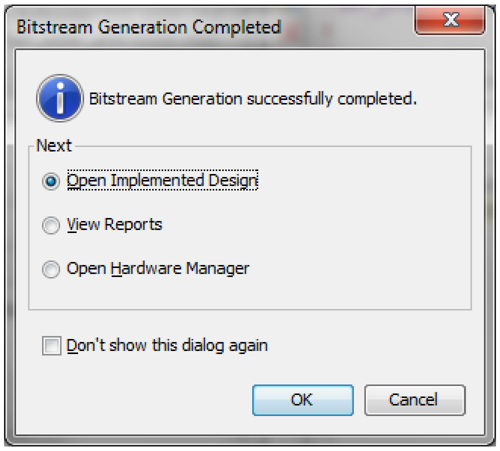
# Mapping SUM to LED<0>
set_property LOC T22 [get_ports SUM]
set_property IOSTANDARD LVCMOS33 [get_ports SUM]

# Mapping SUM to LED<1>
set_property LOC T21 [get_ports CARRY]
set_property IOSTANDARD LVCMOS33 [get_ports CARRY]



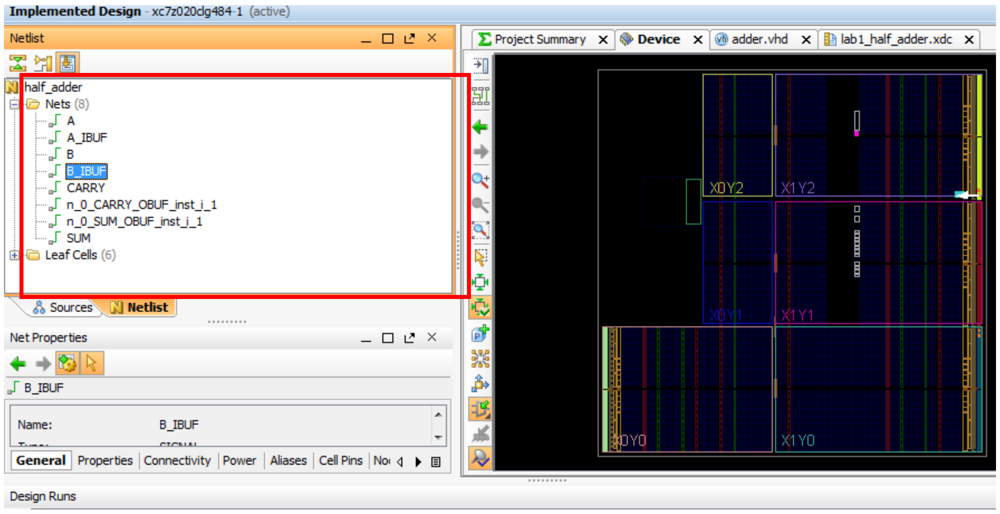
There are no implementation results available. OK to launch synthesis and implementation? 'Generate Bitstream' will automatically start when synthesis and implementation completes.

6. Now click the **Run Implementation** tool to generate the bitstream for FPGA.
There may be a dialog asking you to also launch the Synthesis and other processes before generating bitstream.
Choose **Yes**.
It takes a while to generate bitstream.

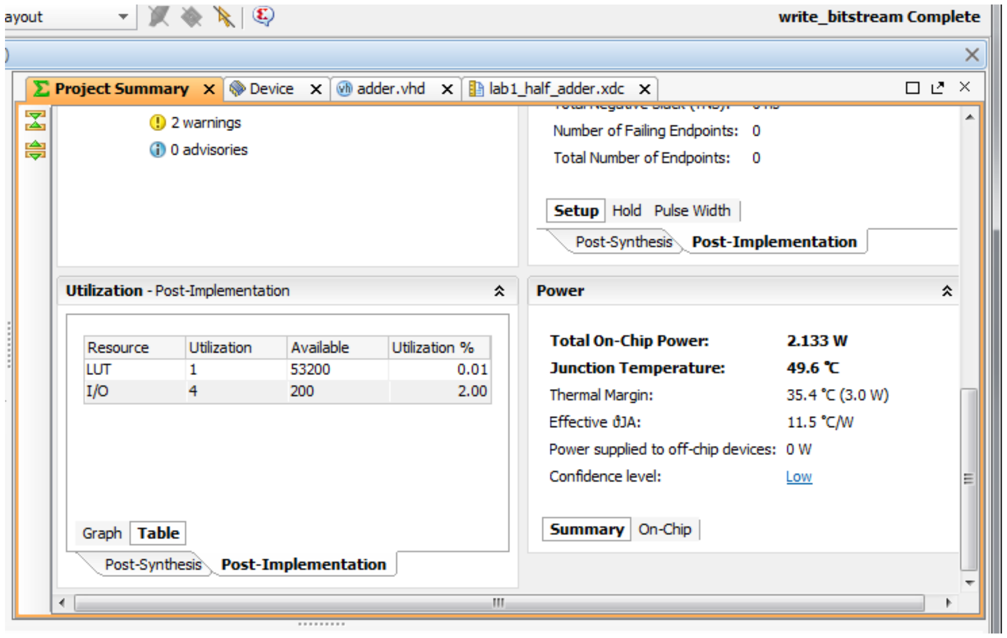


Bitstream Generation successfully completed.

7. After finishing, Vivado asks you to **Open Implemented Design** to view the layout on FPGA or only view the **Reports**.
Choose **Open Implemented Design** then **OK.**



8. The layout of the implemented design (**half_adder)** on FPGA will be shown.
Since this is a very small design, it's hard to see from the full view.
You can click on the instance listed in the red box, it will zoom in to its location on FPGA.



Number of Failing Endpoints: 0
Total Number of Endpoints: 0

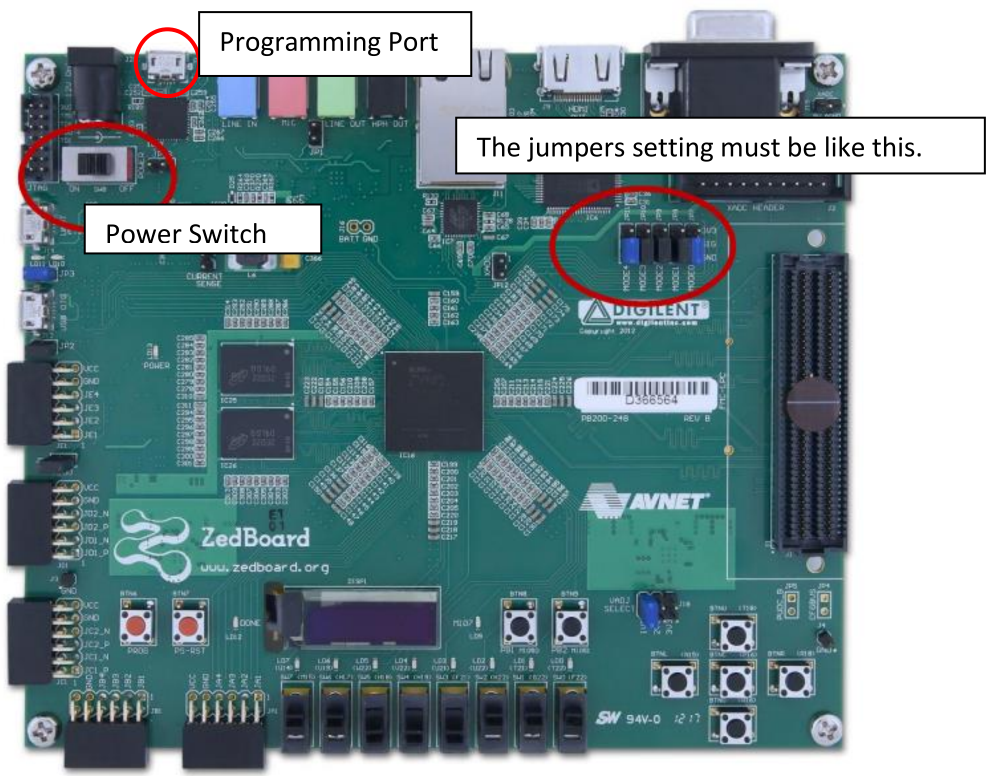
Utilization - Post-Implementation

Resource	Utilization	Available	Utilization %
LUT	1	53200	0.01
I/O	4	200	2.00

Power

Total On-Chip Power:	**2.133 W**
Junction Temperature:	**49.6 ℃**
Thermal Margin:	35.4 ℃ (3.0 W)
Effective ϑJA:	11.5 ℃/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

9. Click on **Project Summary** tab to view the ultilization of the design. In this design, we only need 1 LUT (Lookup Table) and 4 IO pins. Can you explain why?

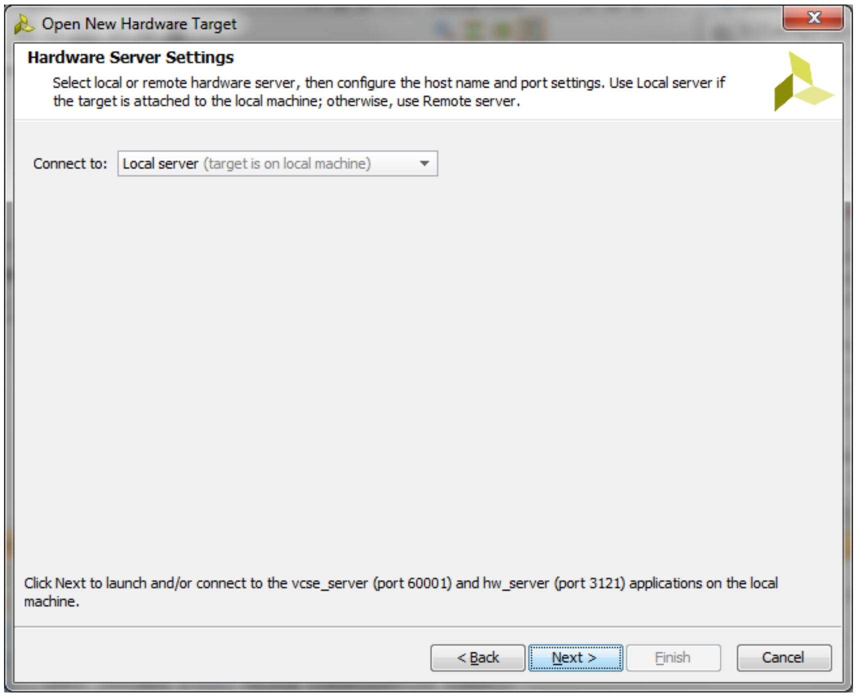


10. The next steps are to download the bitstream to FPGA.

Before going further, make sure you have connected the **Programming Port** on the board to your computer and it is powerred on. If you use the lab PC, all the drivers should be pre-installed.

In additions, verify the position of the jumpers as the left picture.

After that, in the **Flow Navigator,** click **Open Target → Open New Target**.



Hardware Server Settings

Select local or remote hardware server, then configure the host name and port settings. Use Local server if the target is attached to the local machine; otherwise, use Remote server.

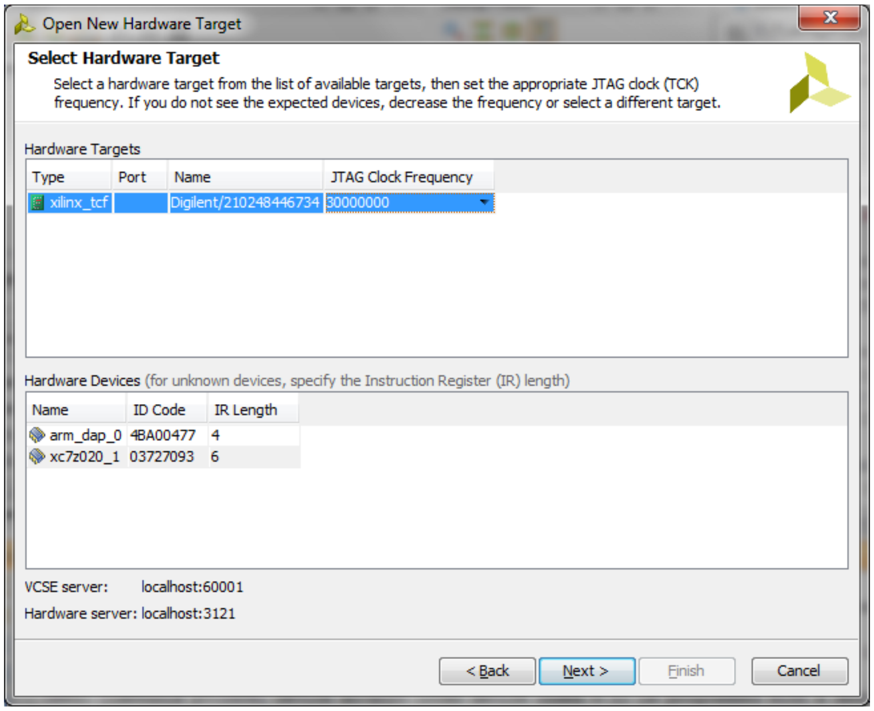
Connect to: Local server (target is on local machine)

Click Next to launch and/or connect to the vcse_server (port 60001) and hw_server (port 3121) applications on the local machine.

11. The **Open New Hardware Target** wizard will be shown. Click **Next.**

    In the **Hardware Server Settings,** select **Local server** then **Next**.

    It takes a while for Vivado to detect the connected device.



Select Hardware Target

Select a hardware target from the list of available targets, then set the appropriate JTAG clock (TCK) frequency. If you do not see the expected devices, decrease the frequency or select a different target.

Hardware Targets

Type	Port	Name	JTAG Clock Frequency
xilinx_tcf		Digilent/210248446734	30000000

Hardware Devices (for unknown devices, specify the Instruction Register (IR) length)

Name	ID Code	IR Length
arm_dap_0	4BA00477	4
xc7z020_1	03727093	6

VCSE server: localhost:60001

Hardware server: localhost:3121

12. If it can’t detect the device, try to power cycle the board, wait till the drivers are loaded and take the steps again.

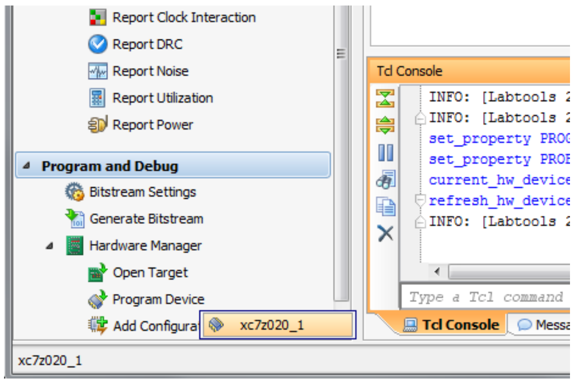
    If there is no problem, the wizard shows the list of devices connected. Here we have:

    - the JTAG programming interface in the **Hardware Targets (**this JTAG interface is built on the Zedboard)
    - and **2 hardware devices (**these devices are on Zynq FPGA chip).

    Click the dropdown box of **JTAG Clock Frequency** then select **30000000 Hz.**

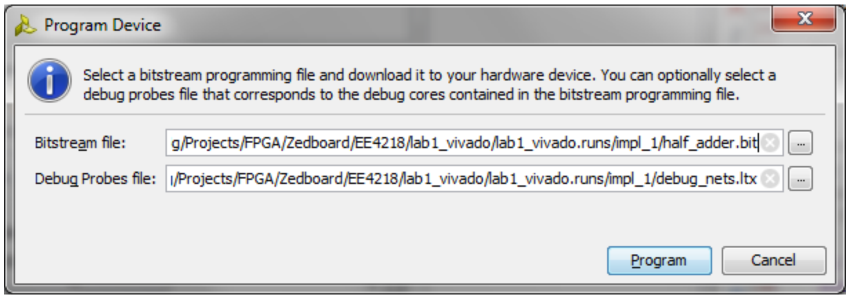
    Select the **xc7z020_1** device.

    Click **Next** and **Finish** to save the settings.



```
INFO: [Labtools 2
INFO: [Labtools 2
set_property PROC
set_property PROE
current_hw_device
refresh_hw_device
INFO: [Labtools 2
```

13. Now, we are ready to download the bitstream to FPGA.
    Click **Program Device** then **xc7z020_1**.



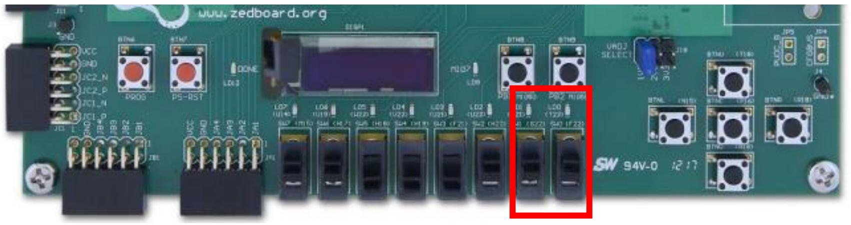
Program Device

Select a bitstream programming file and download it to your hardware device. You can optionally select a debug probes file that corresponds to the debug cores contained in the bitstream programming file.

Bitstream file: g/Projects/FPGA/Zedboard/EE4218/lab1_vivado/lab1_vivado.runs/impl_1/half_adder.bit

Debug Probes file: i/Projects/FPGA/Zedboard/EE4218/lab1_vivado/lab1_vivado.runs/impl_1/debug_nets.ltx

14. Click **Program** to download the bitstream.
    The process should take less than 30 seconds.



15. After that, try to adjust the switches on the board and see the changes of LEDs.

5. Assignments

A. **Implementation of a 4-bit counter** (Assignment **1A**, Classwork) **(4 marks)**

Implement a 4 bit counter on FPGA with the following specifications

1) The count direction (**UP/$\overline{\mathbf{DOWN}}$**) is controlled by a push button, and the count is shown on LEDs.
2) When a **LOAD** push-button is pressed, the counter is loaded asynchronously with a 4 bit **VALUE** set on a 4-bit DIP switch.
3) The counter had 2 modes, controlled using an **AUTO/$\overline{\mathbf{MANUAL}}$** push button. In AUTO mode, the count changes once every second. In MANUAL mode, the count changes by one when a **TICK** pushbutton is pressed (irrespective of the duration for which it is kept pressed, assuming it is > 1 second).

   [Hint: The frequency of the oscillator on the FPGA board is 100MHz. You will have to implement some kind of clock scaling. An apparent clock rate in the order of 1s is fine (i.e., need not be precisely 1s).]

B. **Air Traffic Control** (Assignment **1B**, Homework) **(6 marks)**

Implement an ATC system described below as a modular finite state machine.

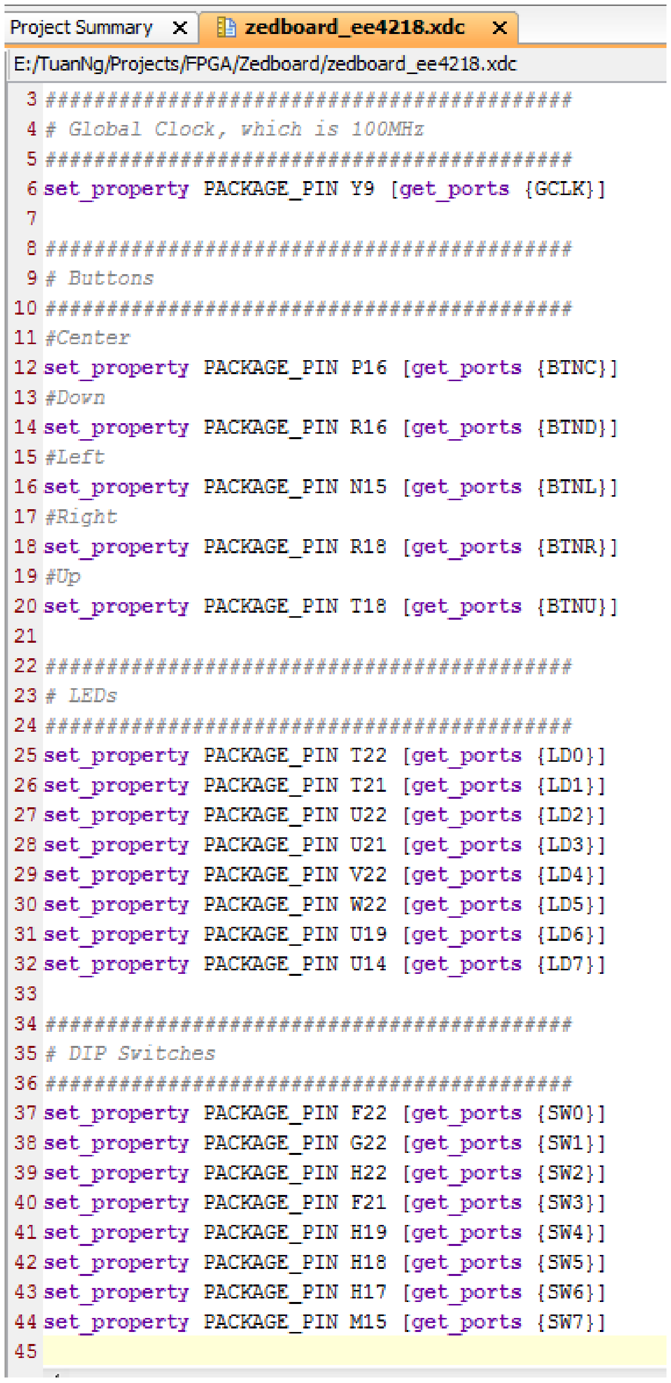
An automated take-off permission system is to be implemented in *De Morgan's Airport*, where one runway is dedicated for take-offs. The airport handles 8 different types of planes (numbered 0-7), broadly classified into two: narrow-body and wide-body; usually referred to as *light* and *heavy* respectively by the ATC. Plane types numbered 1, 3 and 7 are in the heavy category while the rest are light. A light jet taking off immediately behind a heavy jet is risky due to the wake turbulence left behind by the heavy jet.
A plane requests take off by pressing a **REQ** push-button after setting its **TYPE_NUMBER** on a 3-bit DIP switch. The request is **GRANTED** under the following 3 circumstances

1) The request is from a heavy jet
2) The request is from a light jet, and the previous jet which took off is also light
3) The request is from a light jet, the previous jet which took off is heavy, and 10 seconds have passed since the previous jet was granted permission to take off

and is **DENIED** otherwise.
Results (GRANTED/DENIED) are shown using separate LEDs, for exactly 3 seconds (i.e., exactly 3 cycles of the divided clock) immediately following the request, during which take-off requests (i.e pressing the REQ button) will have no effect.

[Hint : You will need to implement one or more counters, which should be used as component(s) controlled by a controller FSM]

6. Appendix

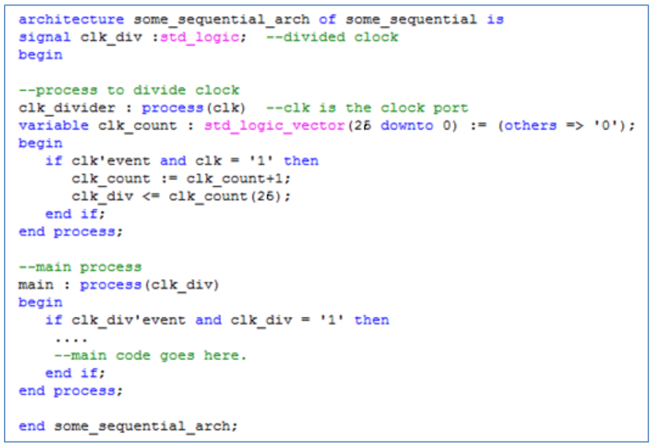


E:/TuanNg/Projects/FPGA/Zedboard/zedboard_ee4218.xdc

```
############################################
# Global Clock, which is 100MHz
############################################
set_property PACKAGE_PIN Y9 [get_ports {GCLK}]

############################################
# Buttons
############################################
#Center
set_property PACKAGE_PIN P16 [get_ports {BTNC}]
#Down
set_property PACKAGE_PIN R16 [get_ports {BTND}]
#Left
set_property PACKAGE_PIN N15 [get_ports {BTNL}]
#Right
set_property PACKAGE_PIN R18 [get_ports {BTNR}]
#Up
set_property PACKAGE_PIN T18 [get_ports {BTNU}]

############################################
# LEDs
############################################
set_property PACKAGE_PIN T22 [get_ports {LD0}]
set_property PACKAGE_PIN T21 [get_ports {LD1}]
set_property PACKAGE_PIN U22 [get_ports {LD2}]
set_property PACKAGE_PIN U21 [get_ports {LD3}]
set_property PACKAGE_PIN V22 [get_ports {LD4}]
set_property PACKAGE_PIN W22 [get_ports {LD5}]
set_property PACKAGE_PIN U19 [get_ports {LD6}]
set_property PACKAGE_PIN U14 [get_ports {LD7}]

############################################
# DIP Switches
############################################
set_property PACKAGE_PIN F22 [get_ports {SW0}]
set_property PACKAGE_PIN G22 [get_ports {SW1}]
set_property PACKAGE_PIN H22 [get_ports {SW2}]
set_property PACKAGE_PIN F21 [get_ports {SW3}]
set_property PACKAGE_PIN H19 [get_ports {SW4}]
set_property PACKAGE_PIN H18 [get_ports {SW5}]
set_property PACKAGE_PIN H17 [get_ports {SW6}]
set_property PACKAGE_PIN M15 [get_ports {SW7}]
```

The constraints for Zedboard.

```
## Constraint file for ZedBoard

############################################
# Global Clock, which is 100MHz
############################################
set_property PACKAGE_PIN Y9 [get_ports {GCLK}]

############################################
# Buttons
############################################
#Center
set_property PACKAGE_PIN P16 [get_ports {BTNC}]
#Down
set_property PACKAGE_PIN R16 [get_ports {BTND}]
#Left
set_property PACKAGE_PIN N15 [get_ports {BTNL}]
#Right
set_property PACKAGE_PIN R18 [get_ports {BTNR}]
#Up
set_property PACKAGE_PIN T18 [get_ports {BTNU}]

############################################
# LEDs
############################################
set_property PACKAGE_PIN T22 [get_ports {LD0}]
set_property PACKAGE_PIN T21 [get_ports {LD1}]
set_property PACKAGE_PIN U22 [get_ports {LD2}]
set_property PACKAGE_PIN U21 [get_ports {LD3}]
set_property PACKAGE_PIN V22 [get_ports {LD4}]
set_property PACKAGE_PIN W22 [get_ports {LD5}]
set_property PACKAGE_PIN U19 [get_ports {LD6}]
set_property PACKAGE_PIN U14 [get_ports {LD7}]

############################################
# DIP Switches
############################################
set_property PACKAGE_PIN F22 [get_ports {SW0}]
set_property PACKAGE_PIN G22 [get_ports {SW1}]
set_property PACKAGE_PIN H22 [get_ports {SW2}]
set_property PACKAGE_PIN F21 [get_ports {SW3}]
set_property PACKAGE_PIN H19 [get_ports {SW4}]
set_property PACKAGE_PIN H18 [get_ports {SW5}]
set_property PACKAGE_PIN H17 [get_ports {SW6}]
set_property PACKAGE_PIN M15 [get_ports {SW7}]
```



```
architecture some_sequential_arch of some_sequential is
signal clk_div :std_logic;  --divided clock
begin

--process to divide clock
clk_divider : process(clk)  --clk is the clock port
variable clk_count : std_logic_vector(26 downto 0) := (others => '0');
begin
   if clk'event and clk = '1' then
      clk_count := clk_count+1;
      clk_div <= clk_count(26);
   end if;
end process;

--main process
main : process(clk_div)
begin
   if clk_div'event and clk_div = '1' then
    ....
    --main code goes here.
   end if;
end process;

end some_sequential_arch;
```

Note for Clock circuit

The clock given by the Zedboard is 100MHz. To obtain a clock in the order of 1Hz, we will need to divide it using a process as shown on the left, and **clk_div** can be used as the clock of the system we are designing.
Please noted that the declaration of **clk_count** is:

variable clk_count : std_logic_vector (26 downto 0) …

Hint : You might want to do the behavioral simulation without the clock divider, and change the clock name in the main process after introducing the clock divider