SMITH ENGINEERING
Queen's University
Electrical and Computer Engineering

Queen's University

ELEC 374 – DIGITAL SYSTEMS ENGINEERING

INTRODUCTION TO INTEL QUARTUS PRIME DESIGN SOFTWARE, MODELSIM-INTEL FPGA STARTER EDITION, AND DE0-CV DEVELOPMENT BOARD

UPDATED JANUARY 2024
K. AGGARWAL, A. SOJOODI, AND A. AFSAHI

UPDATED JANUARY 2023
A. AFSAHI

UPDATED JANUARY 2020
P. ALIZADEH AND A. AFSAHI

UPDATED JANUARY 2017
M. FREGEAU

UPDATED JANUARY 2016:
K. KUMAR AND A. AFSAHI

UPDATED JANUARY 2012-2013:
S. KARADEMIR AND A. AFSAHI

UPDATED DECEMBER 2010:
G. VIJAY, Z. SHEN, AND A. AFSAHI

UPDATED JANUARY 2006-2009:
R. GRANT

UPDATED JANUARY 2005:
M. JARVIN

TUTORIAL DEVELOPED BY:
R. HOSHINO AND A. AFSAHI

TUTORIAL REVISION 12
JANUARY 2024

Table of Contents

1 ABOUT THIS TUTORIAL

Intel Quartus Prime Design Software and ModelSim-Intel FPGA simulator have been installed on the machines in Smith Engineering BMH 212 and BMH 214, where ELEC 374 labs will be held. In addition the software is available remotely through AppsAnywhere.

This tutorial is intended to give you a brief introduction to using Intel Quartus Prime Lite Edition Design Software. For students who have had no previous experience with Quartus, there is an online tutorial that can be accessed directly from the Help menu in the program (see the *Additional References* section at the end of this tutorial).

Section 2 of this tutorial will show you how to install Intel Quartus Prime on your Windows machine. As such, instructions are provided on how to download the free, Lite Edition of the software.

Section 3 will describe the process of creating a new project, consisting of a simple VHDL Ripple Carry Adder and its compilation.

Simulating designs requires the creation of a testbench and the use of a simulator. Section 4 will show how to write a VHDL testbench and set up the required paths for this file to be used during the simulation.

Section 5 will guide you through the setup of ModelSim-Intel on your system to work with Quartus Prime. You will learn how to simulate your VHDL design.

Section 6 will provide you with a Verilog datapath module to help with starting your CPU design project. The datapath consists of a few Registers, a Bus, and a Ripple Carry Adder. This section will show you how to initialize a register, transfer the contents of the register over the bus to another register or to the adder, store the adder result in a register, and transfer it to the output register through the bus, along with the corresponding testbench and simulation results. The methodology presented in Sections 4 and 5 was used to compile and simulate the entire datapath.

Section 7 deals with the hardware details of the DE0-CV Development Board and shows how to assign I/O pins, select hardware devices, and program the FPGA chip.

Section 8 provides some references.

Appendix A is provided for legacy reasons and completeness. Although it is not recommended to use a schematic or a mixed HDL/schematic design for this project, this Appendix describes the process of creating a Schematic design or a mixed HDL/Schematic design in Quartus, v13.0 SP1. Similar design styles can be used in Quartus Prime Lite Edition, v18.1.

2 *INSTALLING INTEL QUARTUS PRIME LITE EDITION 18.1 AND MODELSIM-INTEL FPGA STARTER EDITION*

Intel provides a free Lite edition of Intel Quartus Prime Design Software and a Starter Edition of ModelSim-Intel. Although some advanced features may have been disabled, it is quite sufficient for the requirements of this lab.

- Open a web browser and go to the Intel FPGA Software Download Center for Windows:

Intel Quartus Prime Lite Edition Design Software Version 18.1 for Windows

- Select Quartus Prime Lite, Version 18.1, for Windows.

- Under Downloads, select the "Multiple Download" tab.

    - Download Intel® Quartus® Prime Lite Edition Software (Device support included)
        - **Intel Quartus-Lite-18.1.0.625-windows.tar**.
    - Extract the files into the same temporary directory.
    - Run the setup.bat file, and select at least the following packages during the installation process:
        - Intel Quartus Prime Lite Edition
        - Intel Quartus Prime Help
        - ModelSim-Intel FPGA Starter Edition
        - Intel Cyclone V

- After installation, under Download, select the Updates tab.

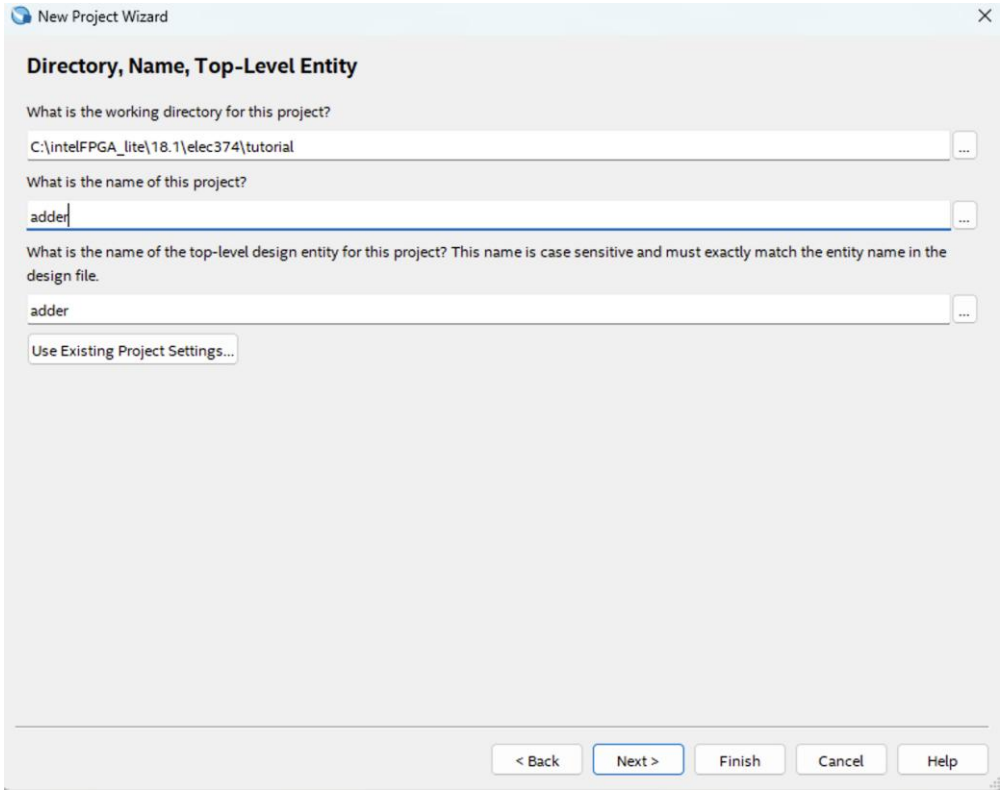
    - Download Intel® Quartus® Prime Software v18.1 Update 1
        - **Intel QuartusSetup-18.1.1.646-windows.tar**.
    - Extract content from downloaded file.
    - Run the .exe file.

3 A SIMPLE VHDL RIPPLE CARRY ADDER

In order to illustrate the basic working of the Quartus design software, we first create a simple VHDL project. The same methodology can be used for Verilog. In Section 6, we will provide a more complete Verilog design example.

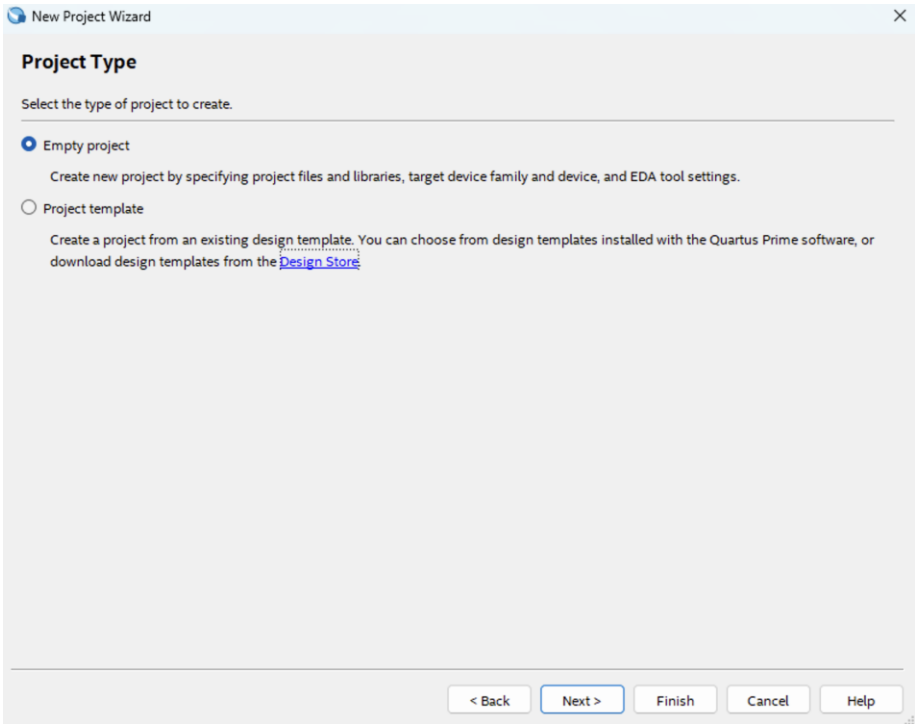
3.1 CREATING A NEW PROJECT

- Start the Quartus Prime program.
- Click on **File → New Project Wizard**.
- After clicking **Next**, the following dialog box will pop up:

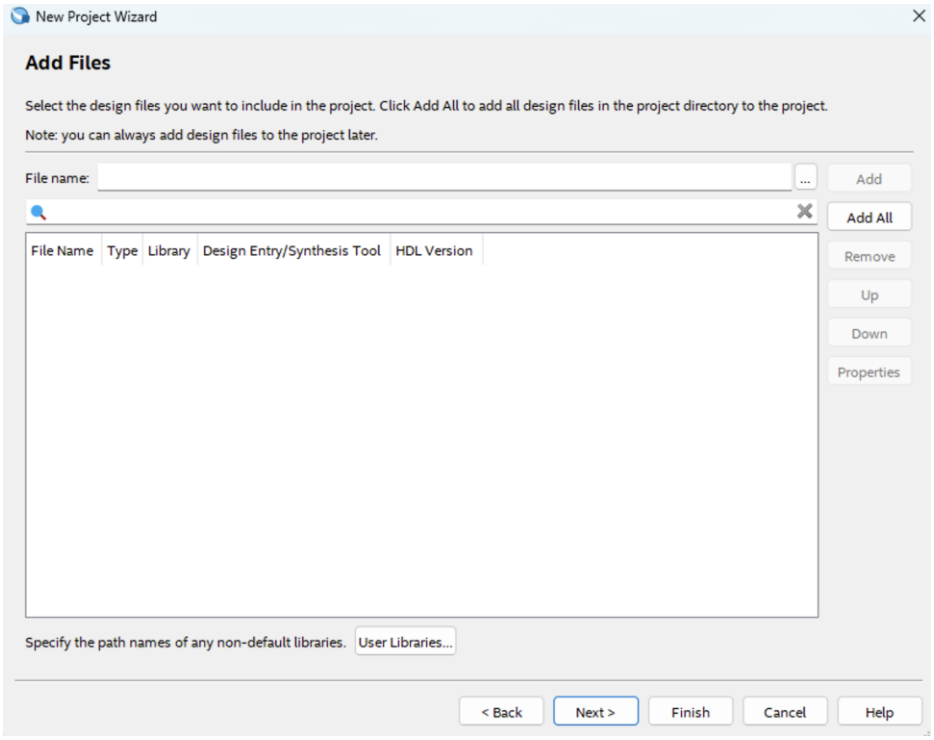


- For the *working directory*, create a new directory on your computer and select it.
- *Note: Since Quartus will create a bunch of project files, make sure you create a separate directory for your design, such as C:\intelFPGA_lite\18.1\elec374\tutorial".*
    1) For the *name of the project*, pick a reasonably descriptive name to describe the project *(for your CPU, you may want to call it "CPU"; for the example in this tutorial, we call it "**adder**")*.
    2) For the *name of the top-level design entity*, pick a name to describe your top level design *(In this example, we use the default name "**adder**", which is the same as the project name)*.

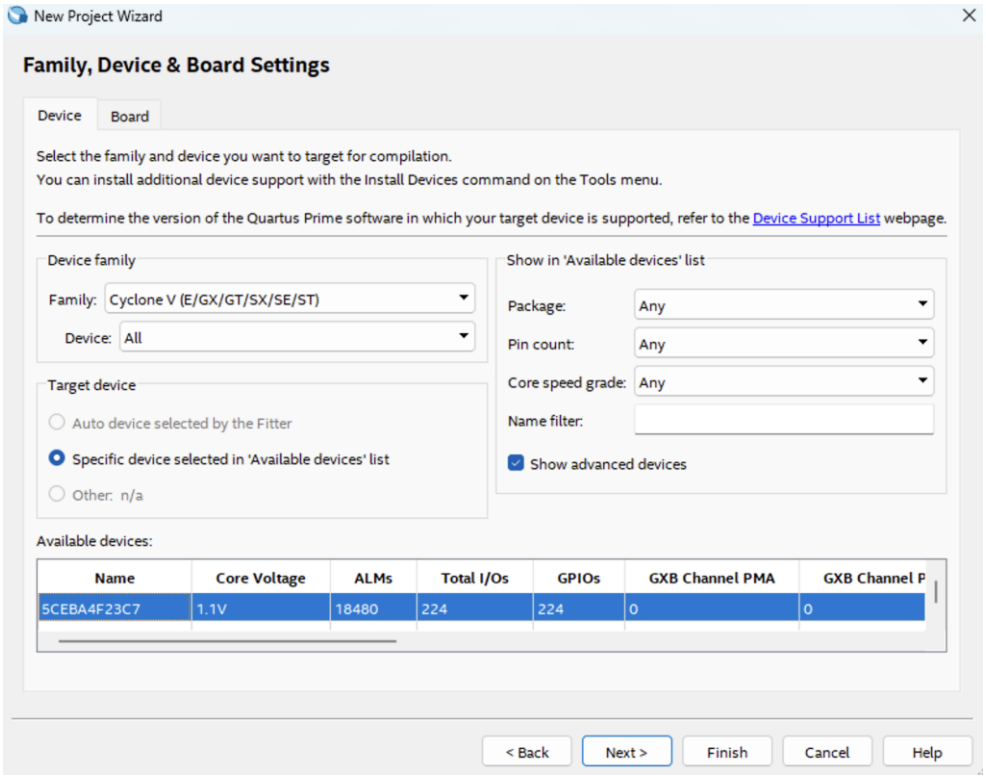
- After clicking **Next**, the following dialog box pops up. Select "Empty project", and click **Next**.



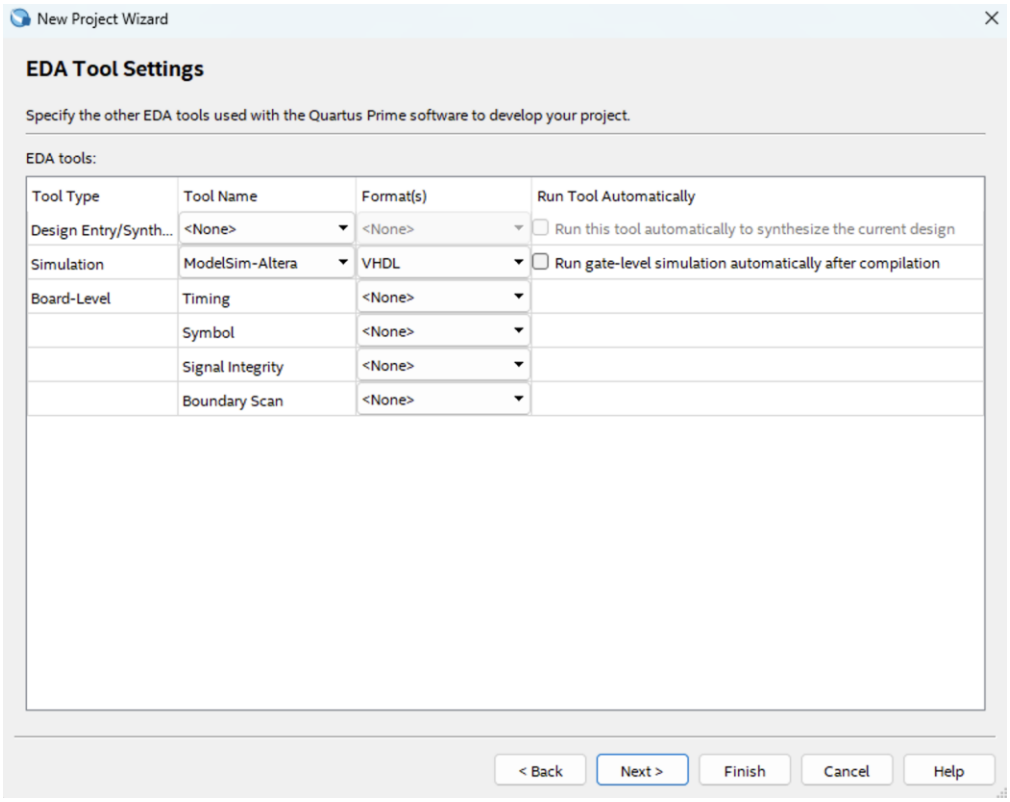
- After clicking **Next**, the following dialog box pops up. Since no design files currently exist, you can simply ignore this step and click **Next**.



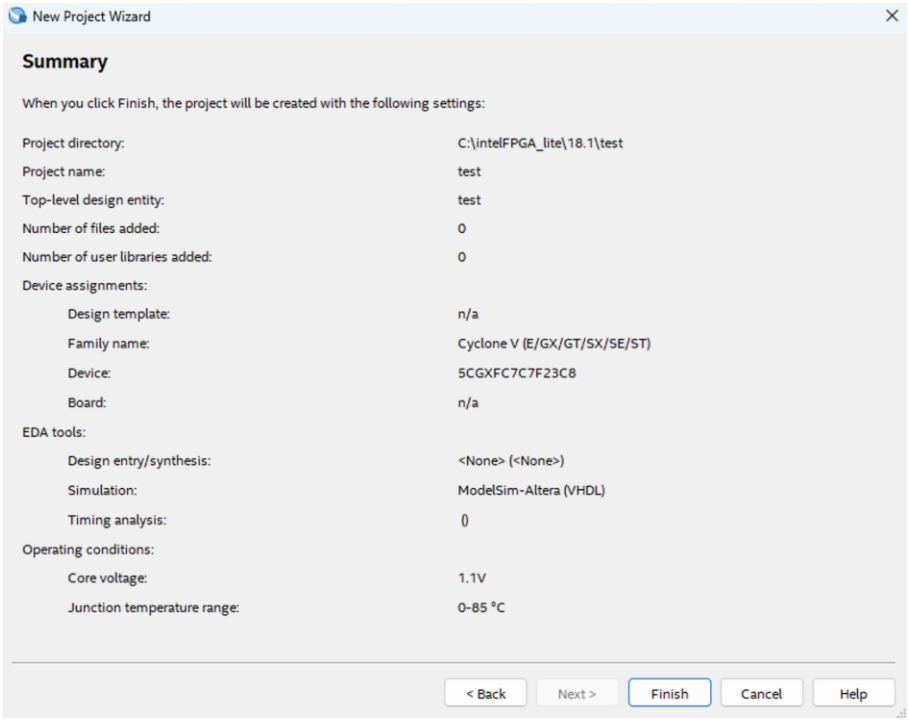
- After clicking **Next**, the following dialog box pops up. For DE0-CV board, select *Cyclone V* from the **Device Family** drop down list, and ensure the *"Specific device selected in Available devices list"* is checked under **Target device** category. Select *5CEBA4F23C7* from the **Available devices** list. If this device is not listed, ensure that all the **Filters** are set to *Any*, as shown below.



- After clicking **Next**, the following dialog box pops up. We will be using an external EDA (Electronic Design Automation) tool, ModelSim-Altera, for our simulation. Hence, make sure you choose this tool in the **Simulation** section under **Tool Name** category. Depending on your choice of HDL, you may choose **Verilog** or **VHDL** from the **Format** menu. For this example, we choose VHDL.



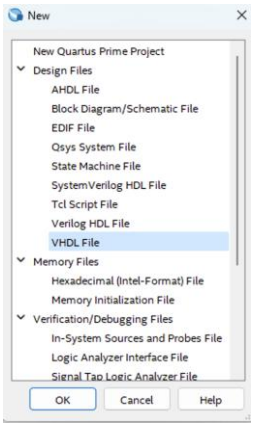
- After clicking **Next**, the following dialog box pops up. Verify your settings and click **Finish** to complete the New Project wizard.



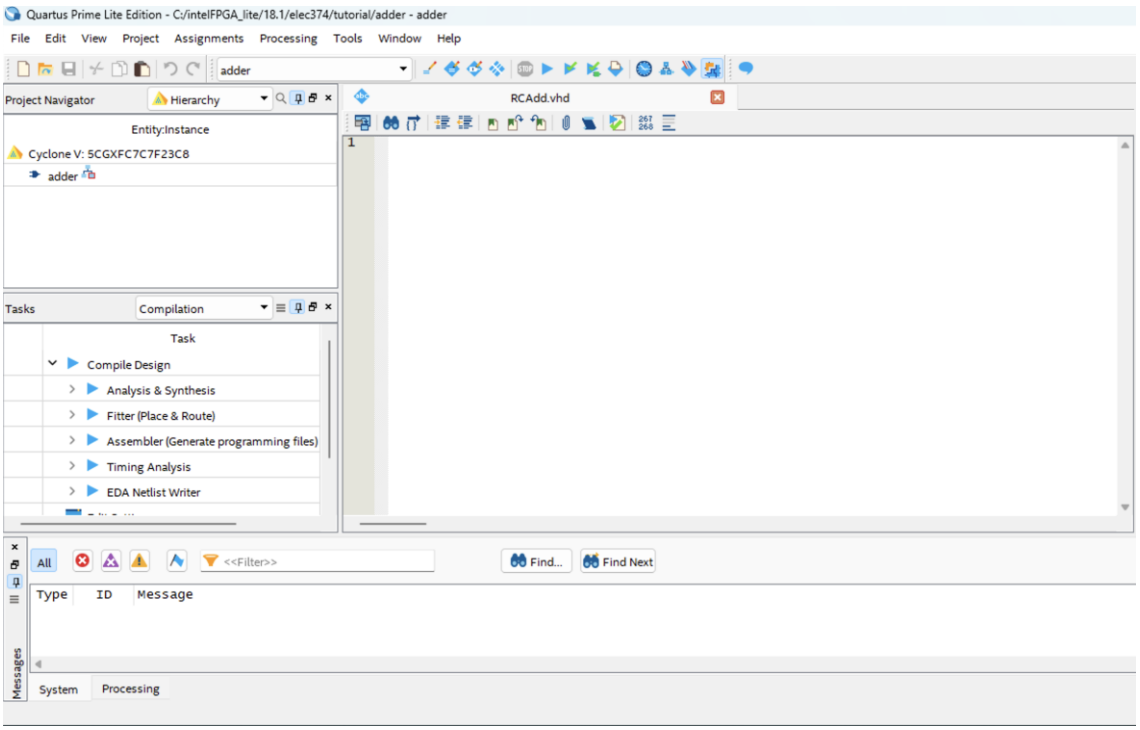
3.2 *ADDING A NEW VHDL FILE*

In this section, we will illustrate how to write a VHDL code for your design. A simple VHDL version of a Ripple Carry Adder will be implemented and will be added to the project.

- In the Quartus main window, Click on **File → New** to bring up the **New** dialog box. Choose the **VHDL** file option (or Verilog file option, depending on your choice of HDL) from *Design Files* tab.



- After clicking **OK**, you will see a new text window. Click on **File → Save As** and save this file as RCAdd.vhd. Ensure the box "**Add file to current project**" is checked.
    - *Note: The file name you provide here will be the **entity name** that you will have to use for your VHDL component*



- The code for the Ripple Carry Adder in Section 3.3 will be added to this file.

3.3 PROGRAMMING A RIPPLE CARRY ADDER

- Enter this code in RCAdd.vhd file and save the design.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY RCAdd IS
    PORT (A, B    : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
          Result  : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END RCAdd;

ARCHITECTURE behavioral OF RCAdd IS
BEGIN
    RippleCarryAdder:
    PROCESS (A, B)
    VARIABLE LocalCarry : STD_LOGIC_VECTOR(8 DOWNTO 0);
    BEGIN
        LocalCarry(0) := '0';
        FOR i IN 0 TO 7 LOOP
            Result(i) <= A(i) XOR B(i) XOR LocalCarry(i);
            LocalCarry(i+1) := (A(i) AND B(i)) OR
                               (LocalCarry(i) AND (A(i) OR B(i)));
        END LOOP;
    END PROCESS;
END behavioral;
```

- The Ripple Carry Adder, RCAdd, is made up of three distinct blocks. Refer to the screenshot below.

Line Number(s)	Function
1-2	Setup libraries (needed for standard logic signals)
4-7	Setup RCAdd Entity and declare all I/O ports
9-22	Declare RCAdd Architecture and define the operation of the ripple carry adder

RCAdd.vhd

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.ALL;
3
4   ENTITY RCAdd IS
5       PORT (A, B    : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
6             Result : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
7   END RCAdd;
8
9   ARCHITECTURE behavioral OF RCAdd IS
10  BEGIN
11      RippleCarryAdder:
12      PROCESS (A, B)
13      VARIABLE LocalCarry : STD_LOGIC_VECTOR(8 DOWNTO 0);
14      BEGIN
15          LocalCarry(0) := '0';
16          FOR i IN 0 TO 7 LOOP
17              Result(i) <= A(i) XOR B(i) XOR LocalCarry(i);
18              LocalCarry(i+1) := (A(i) AND B(i)) OR
19                                 (LocalCarry(i) AND (A(i) OR B(i)));
20          END LOOP;
21      END PROCESS;
22  END behavioral;
```

- Consult the Lecture Slides on VHDL or Verilog in the course. Also, for good references on HDL programming, refer to the *Additional References* section in this tutorial.

4 CREATING AND SETTING UP THE TESTBENCH FOR SIMULATION

- We will write a VHDL testbench to simulate the design using an Electronic Design Automation (EDA) simulation tool, ModelSim-Intel. The same methodology can be used for Verilog.

4.1 CREATING A SIMPLE VHDL TESTBENCH

The following steps will illustrate how to create a simple VHDL testbench:

- To add a VHDL file to the design:
  - Click on **File → New**
  - Select *VHDL File* from the **Design Files** tab and click **OK** .
  - Click on **File → Save As** and save it as *RCAdd_tb.vhd*.
  - Enter the code below in *RCAdd_tb.vhd* as the testbench code and save the design.

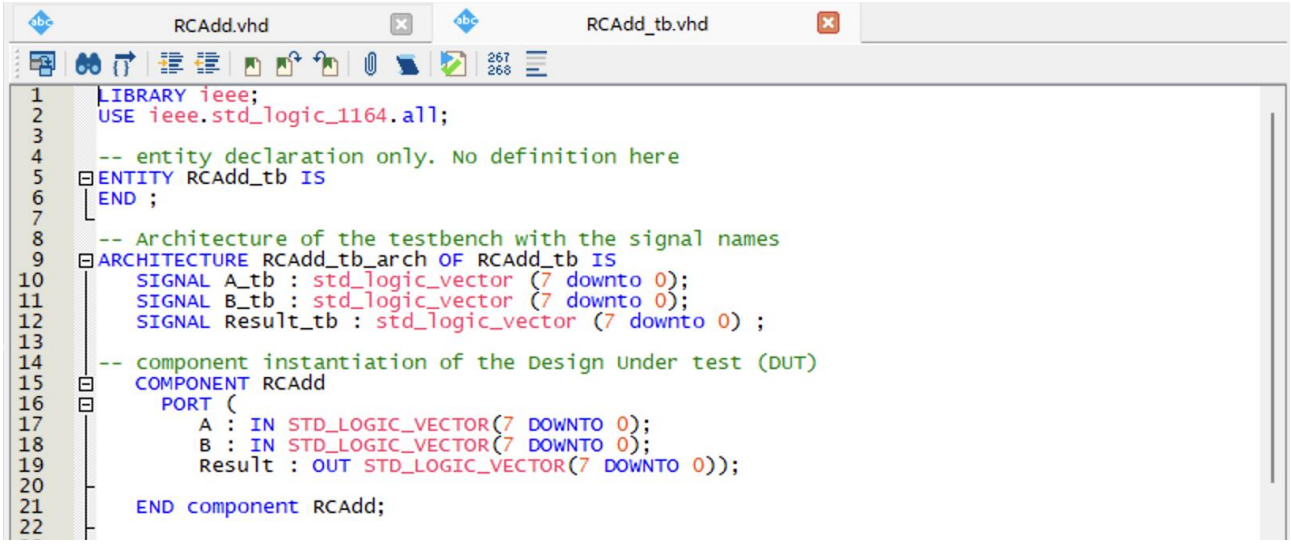
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

-- entity declaration only. No definition here
ENTITY RCAdd_tb IS
END ;

-- Architecture of the testbench with the signal names
ARCHITECTURE RCAdd_tb_arch OF RCAdd_tb IS
   SIGNAL A_tb : std_logic_vector (7 downto 0);
   SIGNAL B_tb : std_logic_vector (7 downto 0);
   SIGNAL Result_tb : std_logic_vector (7 downto 0) ;

-- component instantiation of the Design Under test (DUT)
   COMPONENT RCAdd
     PORT (
        A : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        B : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        Result : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
   END component RCAdd;

  BEGIN
     DUT : RCAdd

--port mapping: between the DUT and the testbench signals
     PORT MAP (
          A => A_tb ,
          B => B_tb ,
          Result => Result_tb ) ;

--add test logic here
   sim_process: process
   begin
       wait for 0 ns;
       A_tb <= b"0000_0000";
       B_tb <= b"0000_0000";

      wait for 20 ns;
       A_tb <= b"0010_1010"; -- decimal 42
```

```
        B_tb <= b"0011_1010"; -- decimal 58

       wait for 200 ns;
        A_tb <= b"01101001"; -- decimal 105
        B_tb <= b"00010101"; -- decimal 21
        wait;
      end process sim_process;
end;
```

Note: Port names in the design file of the DUT (A, B, and Result) will be mapped to the signal names (A_tb, B_tb, and Result_tb) of the testbench file.

- The program is made up of the following distinct blocks. Refer to the screenshot below.

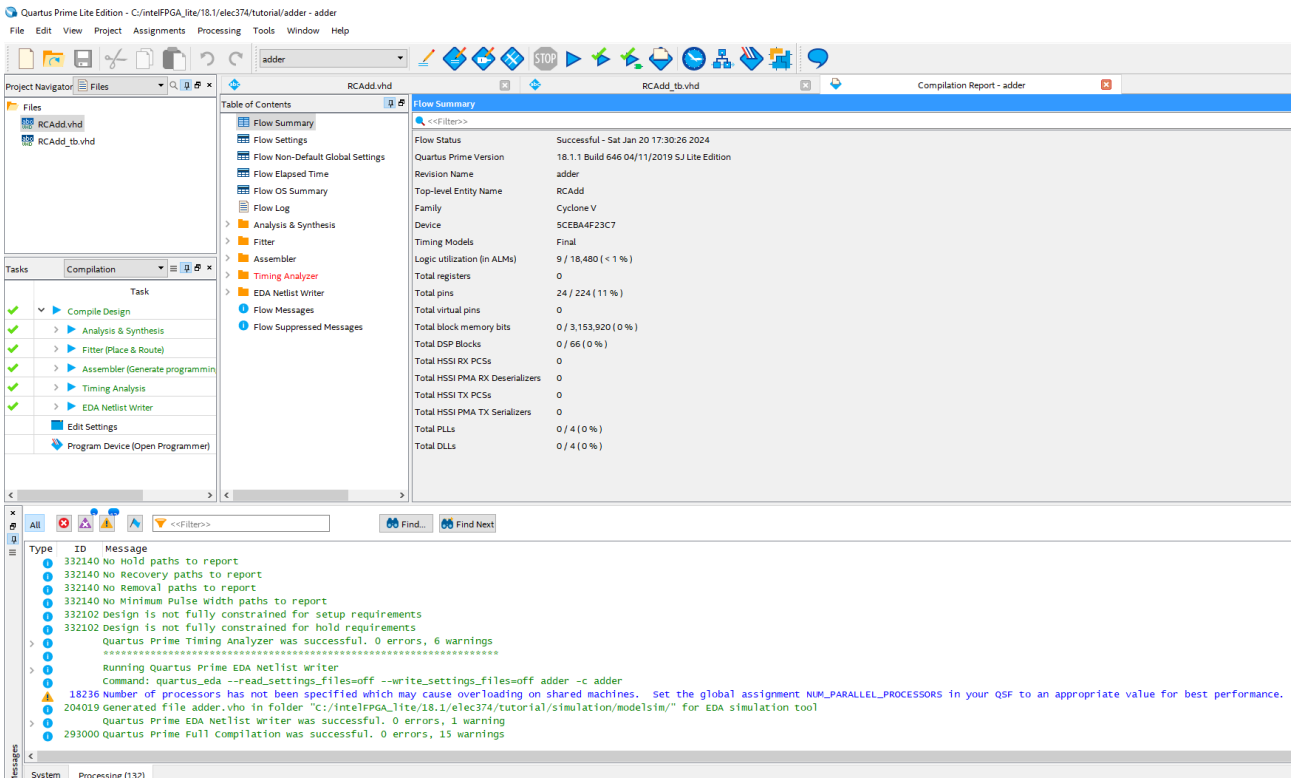
Line Number(s)	Function
1-2	Setup libraries (needed for standard logic signals)
5-5	Entity declaration
8-21	Architecture declaration and Component Instantiation (RCAdd VHDL module)
27-30	Port mapping between the TB signals and the DUT
33-47	Test logic to verify functionality of the generated VHDL output file



```
23      BEGIN
24        DUT : RCAdd
25
26    --port mapping: between the DUT and the testbench signals
27        PORT MAP (
28              A => A_tb ,
29              B => B_tb ,
30              Result => Result_tb ) ;
31
32              --add test logic here
33        sim_process: process
34        begin
35           wait for 0 ns;
36           A_tb <= b"0000_0000";
37           B_tb <= b"0000_0000";
38
39           wait for 20 ns;
40           A_tb <= b"0010_1010"; -- decimal 42
41           B_tb <= b"0011_1010"; -- decimal 58
42
43           wait for 200 ns;
44           A_tb <= b"01101001"; -- decimal 105
45           B_tb <= b"00010101"; -- decimal 21
46           wait;
47        end process sim_process;
48    end;
49
```

- Now set **RCAdd.vhd** as the Top-Level Entity by right clicking on RCAdd.vhd in the **Files** tab, and selecting *Set as Top-Level Entity*
- Save and Compile the design by clicking on

The main window will show various statistics on the design, as shown below.

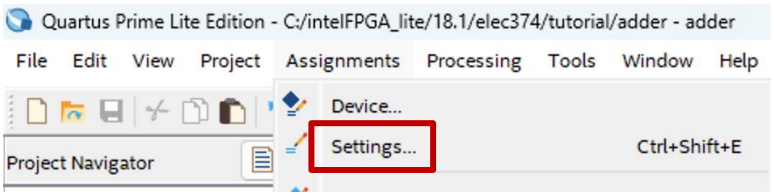


These can be very helpful in analyzing your design. For instance, if the compilation generates errors or warnings, check the messages pane at the bottom of the window for details on what those errors or warning are, and how you might fix them.
By browsing through the reports and **Timing Analyzer**, one can find such information as the anticipated worst-case propagation delay through your circuit, which determines the worst-case maximum clock speed for correct operation.

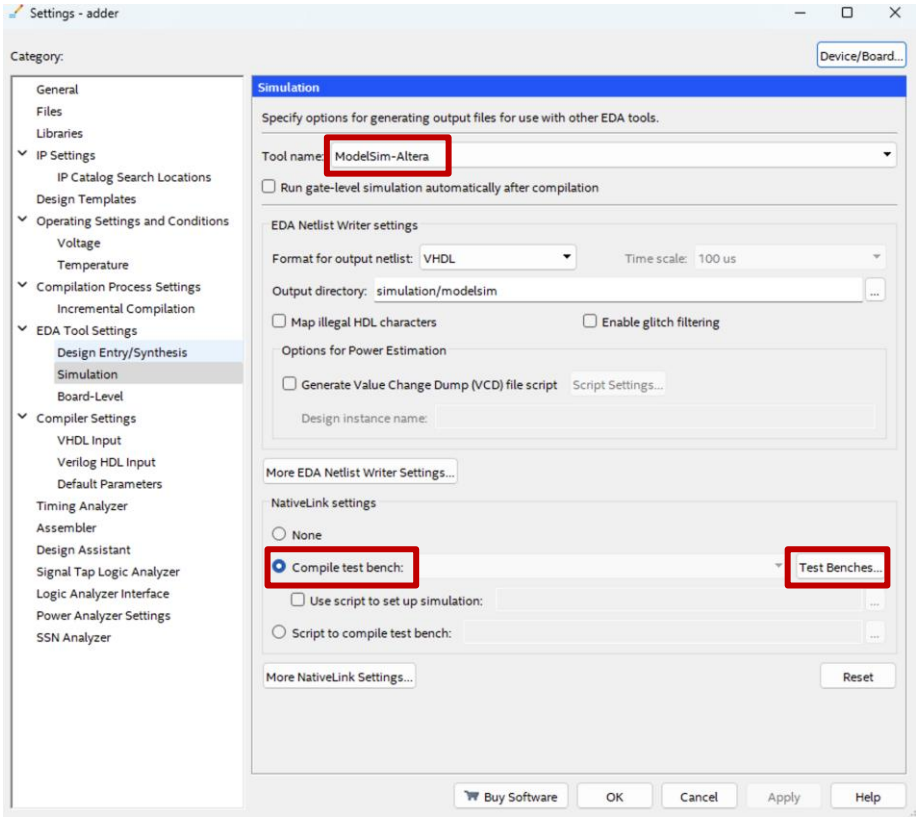
4.2 SETTING UP THE TESTBENCH FOR SIMULATION

- After compilation goes through without errors, we now have to setup the testbench file to be identified by the ModelSim-Intel simulation tool to be used for providing stimulus for the DUT. Here are the steps to setup the Testbench file for simulation:

  - Click on the "**Assignments**" menu and go to **Settings.**

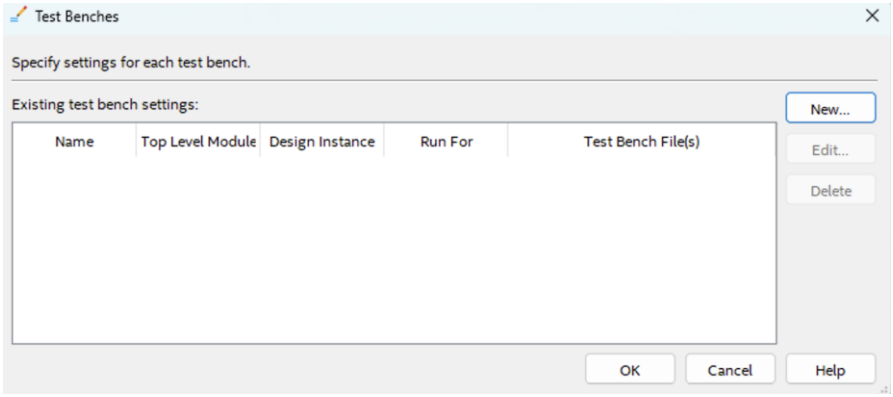


  - In the **Category** list, select **Simulation,** under **EDA Tool Settings**. The **Simulation** page appears.



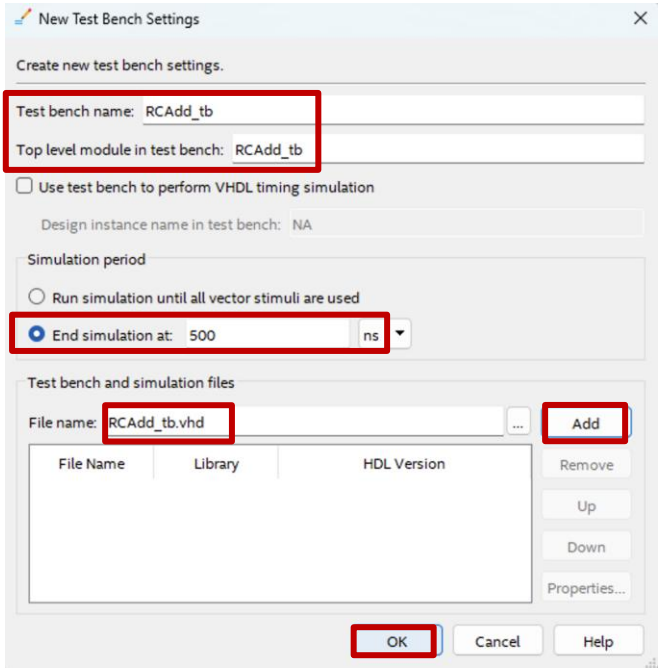
  - In the **Tool name** list, select **ModelSim-Altera**

- Click on the **Compile test bench** option to choose it. This option helps to compile simulation models, design files, testbench files, and starts simulation. You can use different testbench setups to specify different test scenarios.
- Click on **Test Benches**. The **Test Benches** dialog box pops-up.
- Click **New**. The **New Test Bench Settings** dialog box appears

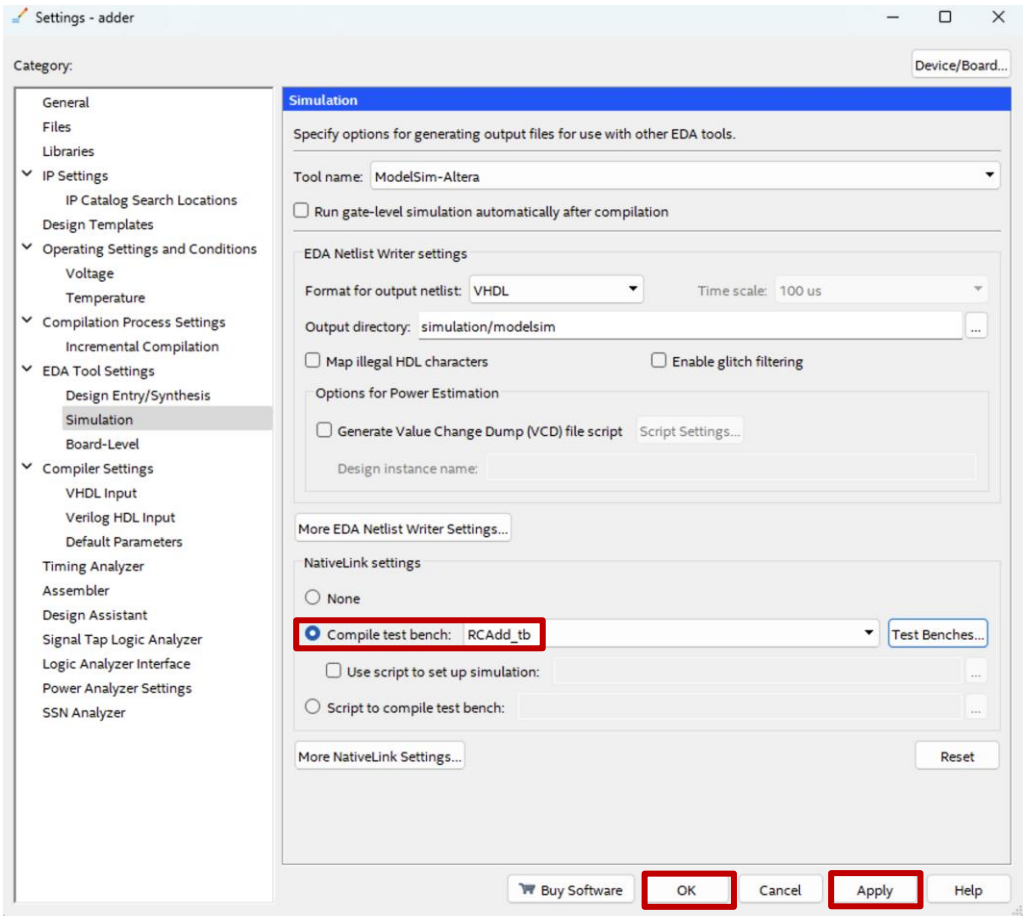


- In the **Top level module in test bench** box, type the top-level testbench entity or module name RCAdd_tb, refer to the screenshot in the next page).
- Under **Simulation period**, select **End simulation at** and specify the time as 500ns.

  Note: If you try to make your simulation too long, Quartus may crash.
- Under **Test bench files**, browse and add the testbench files in the **File name** box. Use the **Up** and **Down** buttons to reorder the files when there are more than one testbench files. The files will be compiled in order from top to bottom. Click **OK**.
- In the **Test benches** dialog box, click **OK**.



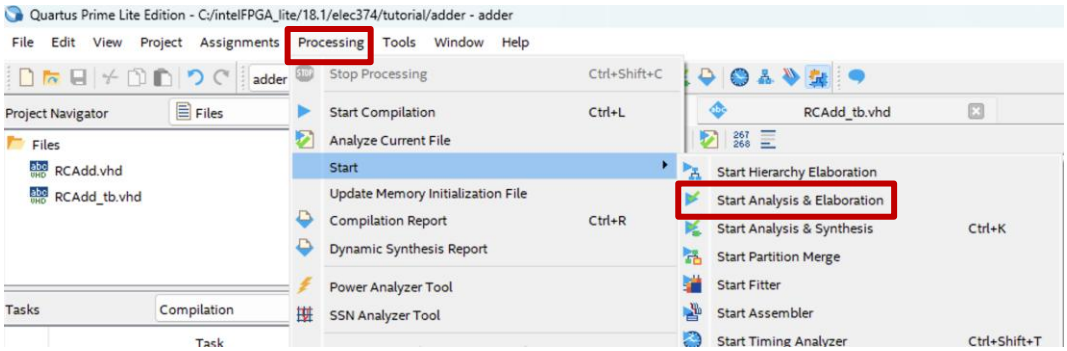
- You will now be back to the **Settings Menu,** where you will find the testbench name you just added next to the **Compile Test bench** option.
- Click on **Apply** and then **OK,** and you will be done with setting up the testbench file for simulation.



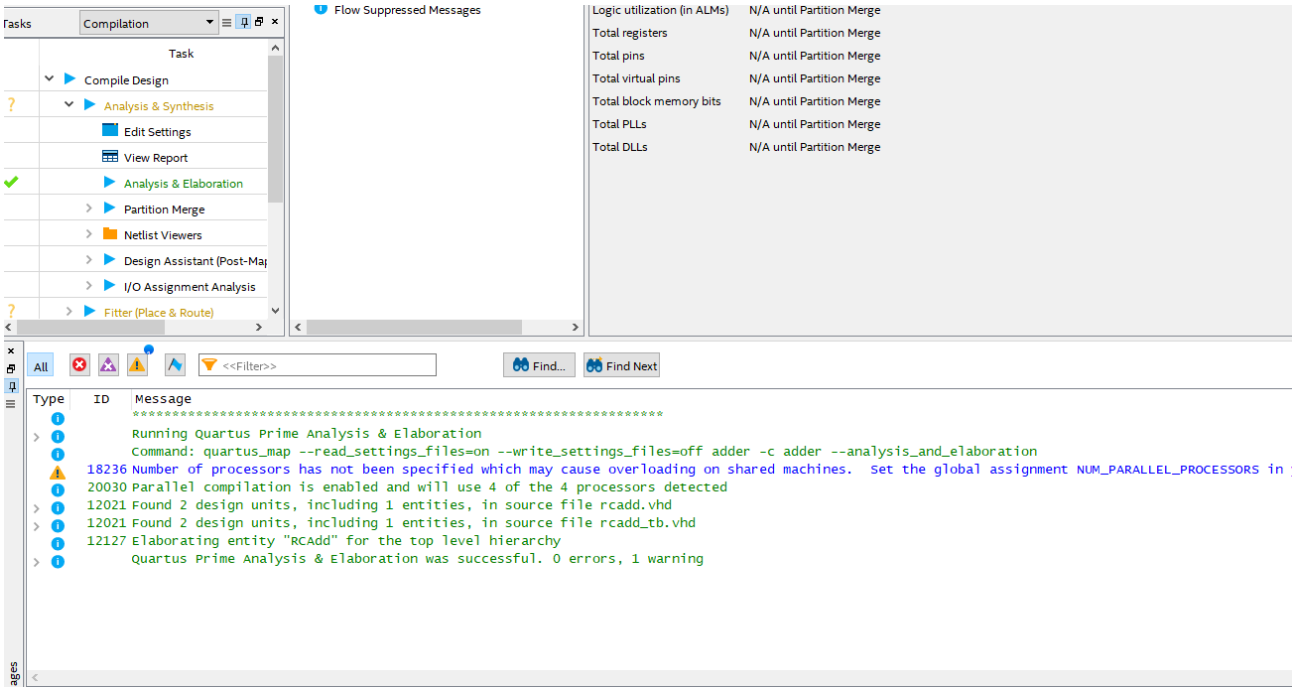
- Now we will see the steps to run the functional simulation using ModelSim-Intel.

4.3 ANALYZING AND ELABORATING THE DESIGN

- On the **Processing** menu, point to **Start** and click on **Start Analysis & Elaboration**. This command collects all your file name information and builds your design hierarchy in preparation for simulation.



- On successful completion of this step, you will see a similar dialogue box, as shown below:



- Warnings may be ignored, as we are only trying to verify the functionality of the design at this point. The warnings may have to be fixed when we try to implement the design in hardware (this is called Design Synthesis).

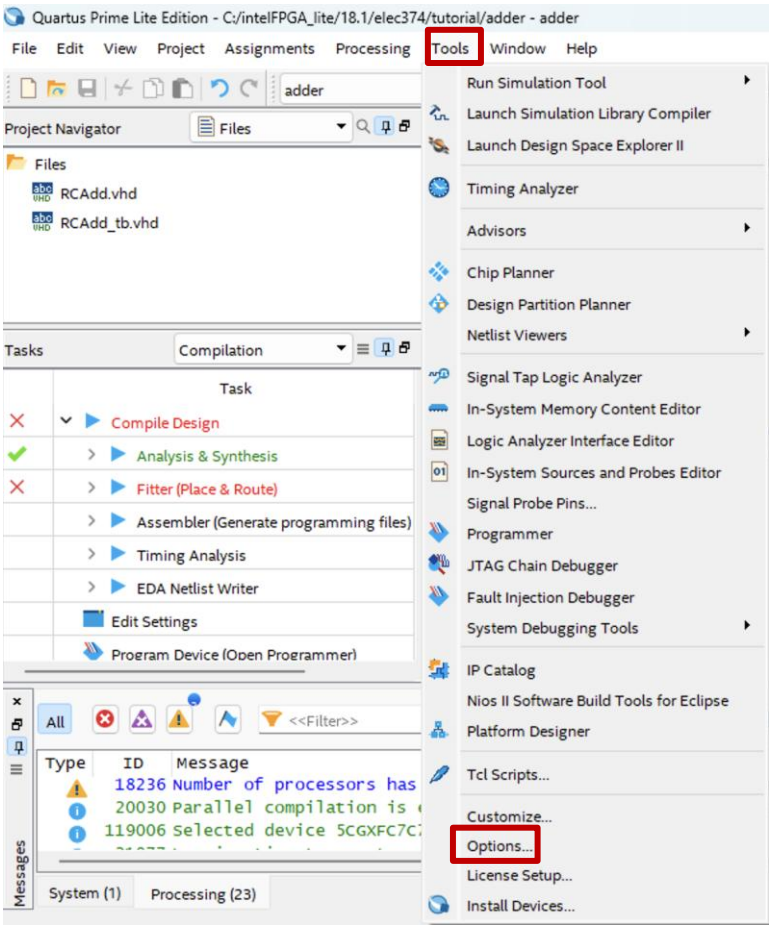
- To run the simulation, we will need to install ModelSim-Intel.

5 FUNCTIONAL SIMULATION WITH MODELSIM-INTEL

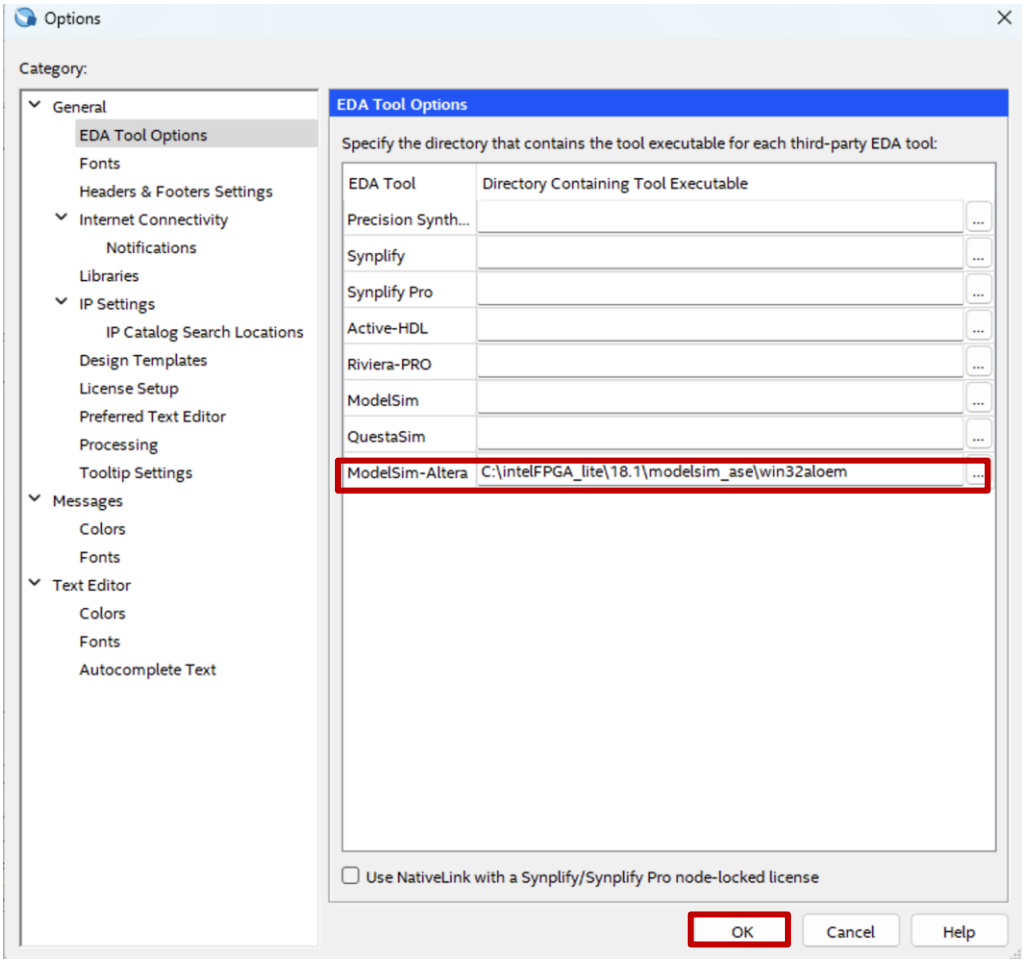
The following steps will guide you through the use of ModelSim for simulation of the design that we have created. The same methodology can be used for Verilog. If you have correctly followed the steps, then ModelSim-Intel should have already been installed in your machine, and you should be able to find it in the Windows startup menu.

5.1 SETTING UP THE EDA SIMULATOR EXECUTION PATH

- ModelSim-Intel can be set as the default EDA (Electronic Design Automation) simulation tool to work with Quartus Prime.
- We can now simulate the testbench using ModelSim-Intel with the following steps:
  - On the **Tools** menu in Quartus, click **Options** as shown in the snapshot.
  - The **Options** dialog box appears



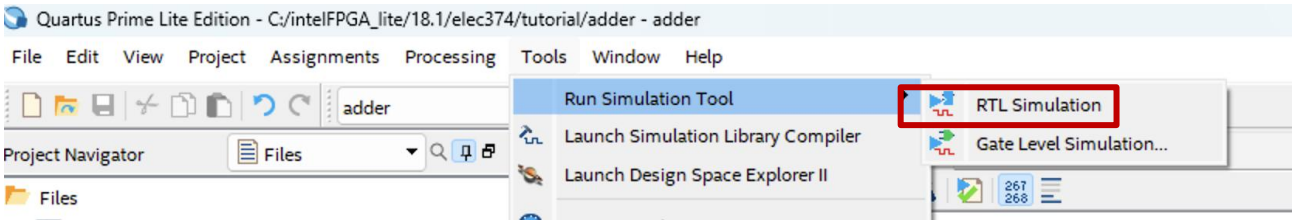
  - In the **Category** list, select **EDA Tool Options**.



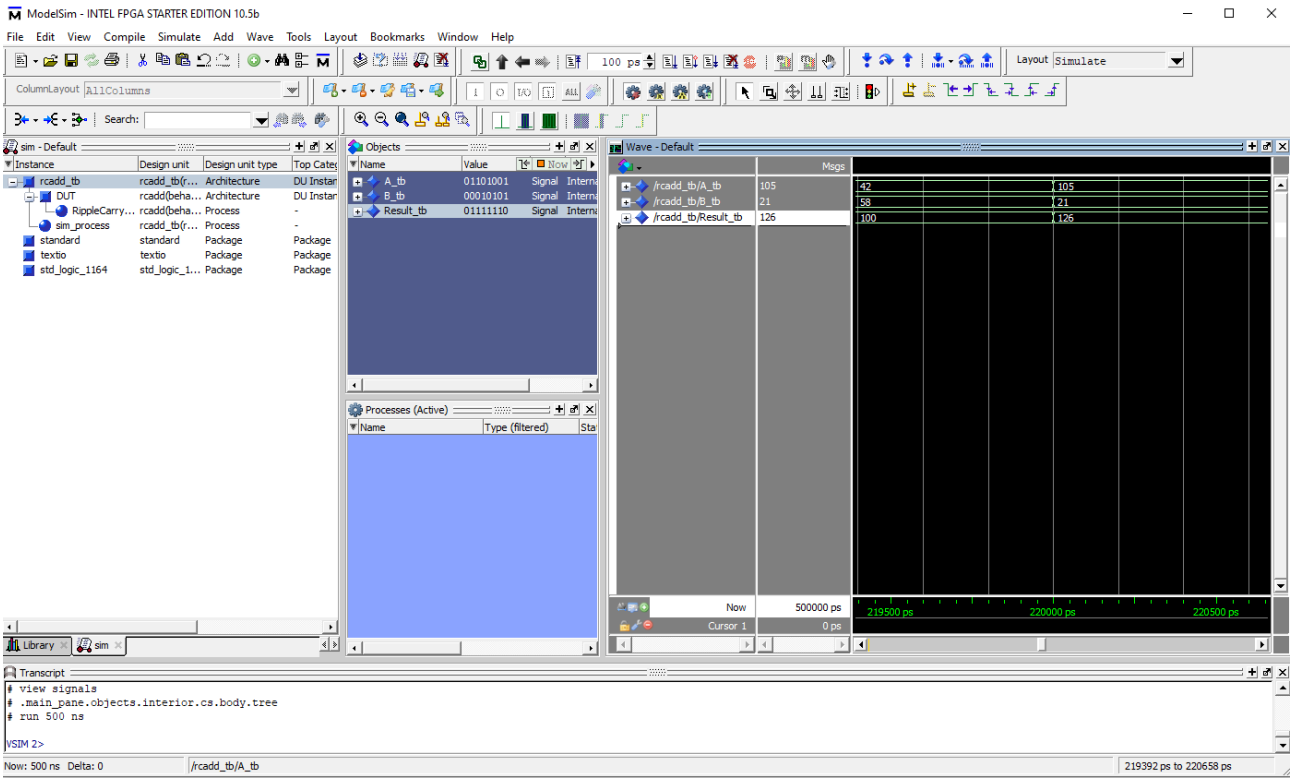
- Click on the button next to the **Location of Executable** entry corresponding to the **ModelSim-Altera** option.
- Type the path or browse to the directory containing the executables of the EDA tool. (The default location is **C:\intelFPGA_lite\18.1\modelsim_ase\win32aloem**)
- Click **OK.**

5.2 LAUNCHING THE MODELSIM-INTEL SIMULATION

- Ensure the required VHDL design file is set as **top-level entity.**
- To launch the simulation (after ensuring analysis and elaboration has gone through without errors), on the **Tools** menu, point to **Run Simulation Tool** and click **RTL Simulation** to automatically run the EDA simulator, compile all necessary design files, and complete a simulation.



1. Now the ModelSim window pops-up and automatically loads all your design files and runs the simulation for you. You should be able to see the simulation results in the window that pops-up.
2. You will notice that the output: **Result_tb** is the result of the adder module, which is capable of addition.
3. This completes the testing of the Adder module you created.



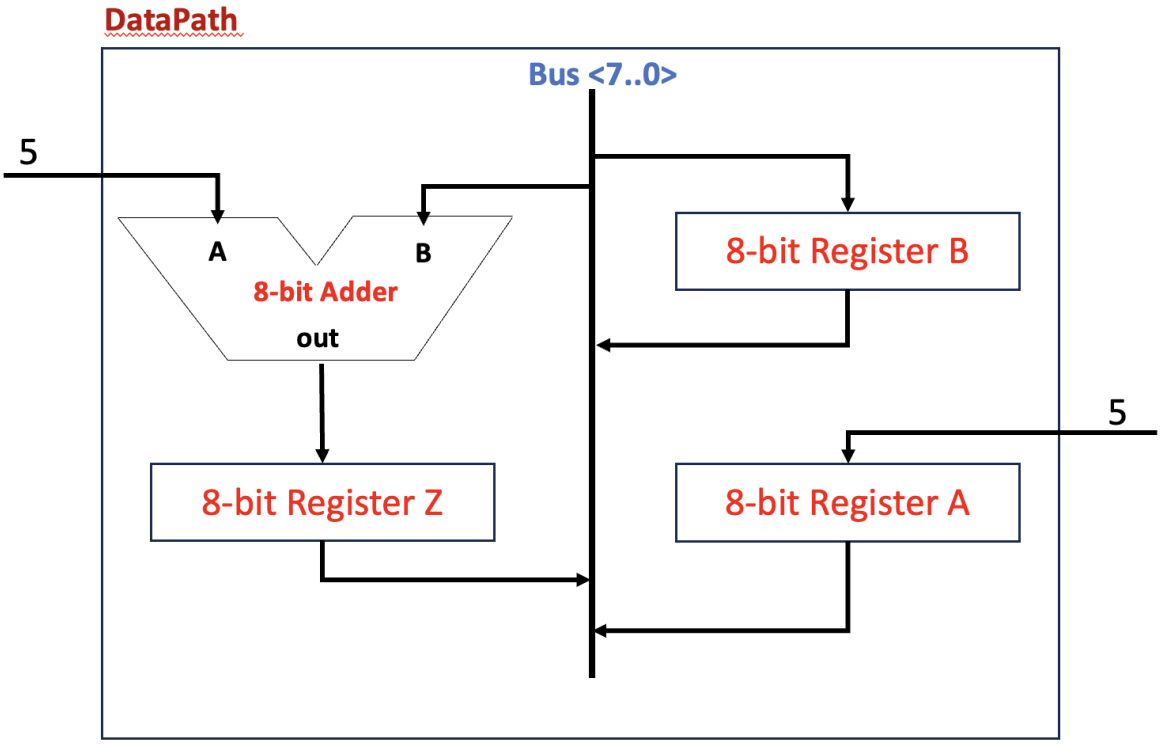
4. The radix of the results can be changed to decimal to make the results easily readable, by right-clicking on the binary numbers and changing the radix to decimal.

Note: Ensure that the ModelSim-Intel window is closed every time you launch a simulation from Quartus Prime. Otherwise, there might be a conflict with the existing open window and ModelSim-Intel may not launch.

6 A SIMPLE DATAPATH DESIGN IN VERILOG

In Phase 1 and Phase 2 of the CPU design project, you will be involved with the design and implementation of a **Datapath** consisting of number of modules, including 32-bit general-purpose registers, Instruction Register (IR), Program Counter (PC), a 32-bit bi-directional Bus, Arithmetic Logic Unit (ALU), memory interface registers (MAR and MDR), and input and output ports, among others. Creating these modules, transferring register contents over the bus, and connecting the modules together in the top-level entity, datapath, is crucial to the success of your project.

In this section, we will show you how you can build a mini-datapath consisting of three 8-bit registers, an 8-bit bi-directional Bus, and an 8-bit adder, and how to instantiate them in the top-level entity datapath, as shown in the figure below. This example shows a simple datapath to help students understand the purpose and function of the bus and datapath. This gives an example of the project structure to help with the understanding of how to use modules in the top-level entity, and how the testbench will interact with the top-level entity.



The testbench runs through two instructions, one to load registers, and the other to do the addition.

1. Load Register A with an immediate value of 5.

2. Add the contents of Register A (by putting the contents of Register A on the bus) with an immediate value of 5 using the Adder module, and then store the result of the Adder in Register Z.

3. Put the contents of Register Z on the bus, and store that value in Register B.

6.1 Verilog Ripple Carry Adder Module

The following code shows a simple 8-bit Ripple Carry Adder in Verilog.

```
1   // Ripple Carry Adder
2   module adder(A, B, Result);
3
4   input [7:0] A, B;
5   output [7:0] Result;
6
7   reg [7:0] Result;
8   reg [8:0] LocalCarry;
9
10  integer i;
11
12  always@(A or B)
13     begin
14        LocalCarry = 9'd0;
15        for(i = 0; i < 8; i = i + 1)
16        begin
17              Result[i] = A[i]^B[i]^LocalCarry[i];
18              LocalCarry[i+1] = (A[i]&B[i])|(LocalCarry[i]&(A[i]|B[i]));
19        end
20  end
21  endmodule
22
```

6.2 Edge-Triggered Register Module

The following code shows a simple 8-bit Register in Verilog.

```
1   module register #(parameter DATA_WIDTH_IN = 8, DATA_WIDTH_OUT = 8, INIT = 8'h0)(
2      input clear, clock, enable,
3      input [DATA_WIDTH_IN-1:0]BusMuxOut,
4      output wire [DATA_WIDTH_OUT-1:0]BusMuxIn
5   );
6    reg [DATA_WIDTH_IN-1:0]q;
7    initial q = INIT;
8    always @ (posedge clock)
9          begin
10            if (clear) begin
11               q <= {DATA_WIDTH_IN{1'b0}};
12            end
13            else if (enable) begin
14               q <= BusMuxOut;
15            end
16         end
17      assign BusMuxIn = q[DATA_WIDTH_OUT-1:0];
18   endmodule
19
```

6.3 Bi-Directional Bus Module

The following code shows an 8-bit bi-directional bus. For a better understanding of the function and organization of the bus, consult the **CPU Phase 1** document and its Figure 1 in onQ.

```
module Bus (
    //Mux
    input [7:0]BusMuxInRZ, input [7:0]BusMuxInRA, input [7:0]BusMuxInRB,
    //Encoder
    input RZout, RAout, RBout,

    output wire [7:0]BusMuxOut
);

reg [7:0]q;

always @ (*) begin
    if(RZout) q = BusMuxInRZ;
    if(RAout) q = BusMuxInRA;
    if(RBout) q = BusMuxInRB;
end
assign BusMuxOut = q;
endmodule
```

6.4 DATAPATH MODULE

The following code shows the **Datapath** module that instantiates three 8-bit Registers, an 8-bit Adder, and an 8-bit bi-directional Bus, using the adder, register, and Bus modules.

```
module DataPath(
    input wire clock, clear,
    input wire [7:0] A,
    input wire [7:0] RegisterAImmediate,
    input wire RZout, RAout, RBout,
    input wire RAin, RBin, RZin
);

wire [7:0] BusMuxOut, BusMuxInRZ, BusMuxInRA, BusMuxInRB;

wire [7:0] Zregin;

//Devices
register RA(clear, clock, RAin, RegisterAImmediate, BusMuxInRA);
register RB(clear, clock, RBin, BusMuxOut, BusMuxInRB);

// adder
adder add(A, BusMuxOut, Zregin);
register RZ(clear, clock, RZin, Zregin, BusMuxInRZ);

//Bus
Bus bus(BusMuxInRZ, BusMuxInRA, BusMuxInRB, RZout, RAout, RBout, BusMuxOut);

endmodule
```

6.5 TESTBENCH

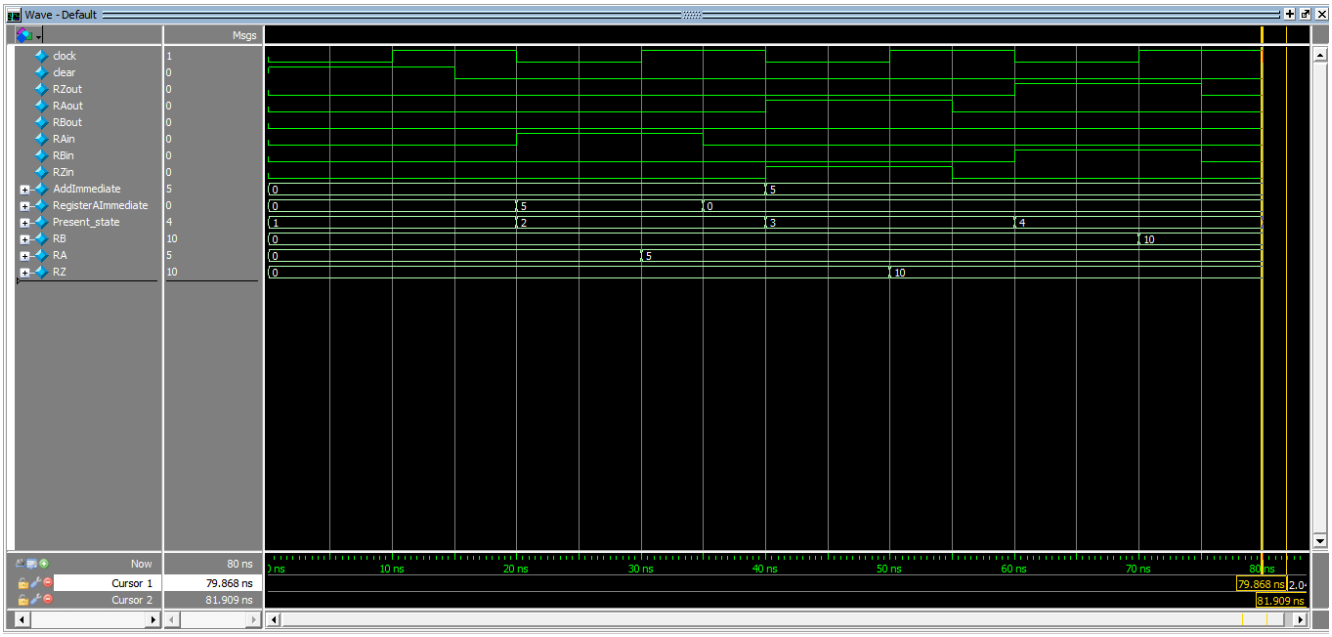
The following code shows the Verilog testbench for the top-level entity Datapath. The structure of the testbench is very similar to the structure of the testbench shown for Verilog (Page 11 - 13) and VHDL (Page 8 -10) of the **CPU Phase 1** document. It first instantiates the Datapath, initializes the state and registers, load Register A with the value 5, add the immediate value 5 to the contents of the Register A using the adder, and store the result of addition in Register Z, and finally transfer the value of

Register Z to Register B.

```
`timescale 1ns/10ps
module tutorial_tb();
reg clock, clear, RZout, RAout, RBout, RAin, RBin, RZin;
reg [7:0] AddImmediate;
reg [7:0] RegisterAImmediate;

reg [3:0] present_state;

DataPath DP(
   clock, clear,
   AddImmediate,
   RegisterAImmediate,
   RZout, RAout, RBout,
   RAin, RBin, RZin
);

parameter init = 4'd1, T0 = 4'd2, T1 = 4'd3, T2 = 4'd4;

initial begin clock = 0; present_state = 4'd0; end
always #10 clock = ~clock;
always @ (negedge clock) present_state = present_state + 1;

always @(present_state) begin
   case(present_state)
      init: begin
         clear <= 1;
         AddImmediate <= 8'h00; RegisterAImmediate <= 8'h00;
         RZout <= 0; RAout <= 0; RBout <= 0; RAin <= 0; RBin <= 0; RZin <= 0;
         #15 clear <= 0;
      end
      // ldi A, 5
      T0: begin
         RegisterAImmediate <= 8'h5; RAin <= 1;
         #15 RegisterAImmediate <= 8'h00; RAin <= 0;
      end
   // addi B, A, 5 - 2 steps
      // add value in register A and immediate 5 and then save in Z
      T1: begin
         RAout <= 1; AddImmediate <= 8'h5; RZin <= 1;
         #15 RAout <= 0; RZin <= 0;
      end
      // mv B, Z - move value in Z to B
      T2: begin
         RZout <= 1; RBin <= 1;
         #15 RZout <= 0; RBin <= 0;
      end
   endcase
end
endmodule

```

6.6 SIMULATION RESULTS

After compilation and simulation, as discussed in Sections 4 and 5, you should see the following waveform in ModelSim-Intel.


Wave - Default
Msgs
clock
clear
RZout
RAout
RBout
RAin
RBin
RZin
AddImmediate
RegisterAImmediate
Present_state
RB
RA
RZ
Now
80 ns
Cursor 1
79.868 ns
Cursor 2
81.909 ns
10 ns
20 ns
30 ns
40 ns
50 ns
60 ns
70 ns

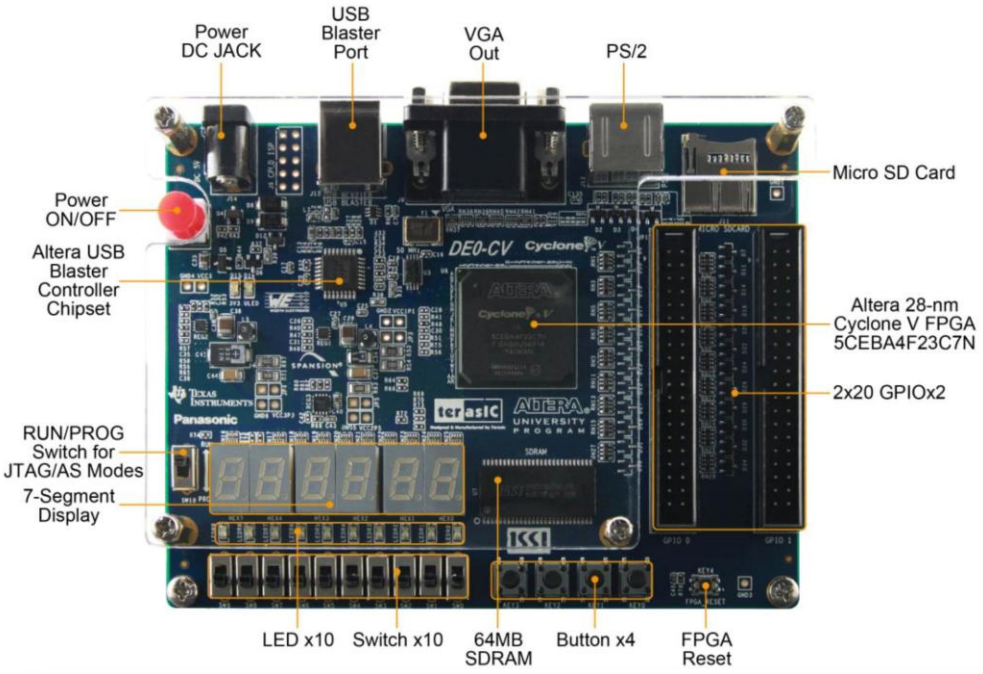
7 TESTING DESIGN IN HARDWARE

In Phase 4 of the Lab CPU design project, you will be testing your final processor design on the DE0-CV development board. You may wish to just skim over this section and when it is time to program your design, you can refer back to this section and read it in more detail.

Note: Be sure to read over the Altera documentation for DE0-CV User Manual, available in ELEC 374 onQ page. You will need to refer to it to find the pin numbers that are explained in the following sections.

7.1 DE0-CV EVALUATION BOARD

The DE0-CV evaluation board provided in the lab will look similar to the diagram below:



Important: Make sure the Run/Prog switch is set to Run, and never change this switch!

7.2 ASSIGNING I/O PINS TO EXTERNAL PINS

To test your design, you need to connect your external switches and LEDs on the evaluation board to the pins on the FPGA.

- The DE0-CV evaluation board has one bank of slide switches, pushbutton switches and LEDs, the descriptions of which can be found in the three tables below, also available on Pages 24-25 in the DE0-CV User Manual.

Table 3-3 Pin Assignment of Slide Switches

Signal Name	*FPGA Pin No.*	*Description*
SW0	PIN_U13	Slide Switch[0]
SW1	PIN_V13	Slide Switch[1]
SW2	PIN_T13	Slide Switch[2]
SW3	PIN_T12	Slide Switch[3]
SW4	PIN_AA15	Slide Switch[4]
SW5	PIN_AB15	Slide Switch[5]
SW6	PIN_AA14	Slide Switch[6]
SW7	PIN_AA13	Slide Switch[7]
SW8	PIN_AB13	Slide Switch[8]
SW9	PIN_AB12	Slide Switch[9]

Table 3-2 Pin Assignment of Push-buttons

Signal Name	*FPGA Pin No.*	*Description*
KEY0	PIN_U7	Push-button[0]
KEY1	PIN_W9	Push-button[1]
KEY2	PIN_M7	Push-button[2]
KEY3	PIN_M6	Push-button[3]
RESET_N	PIN_P22	Push-button which connected to DEV_CLRN Pin of FPGA

Table 3-4 Pin Assignment of LEDs

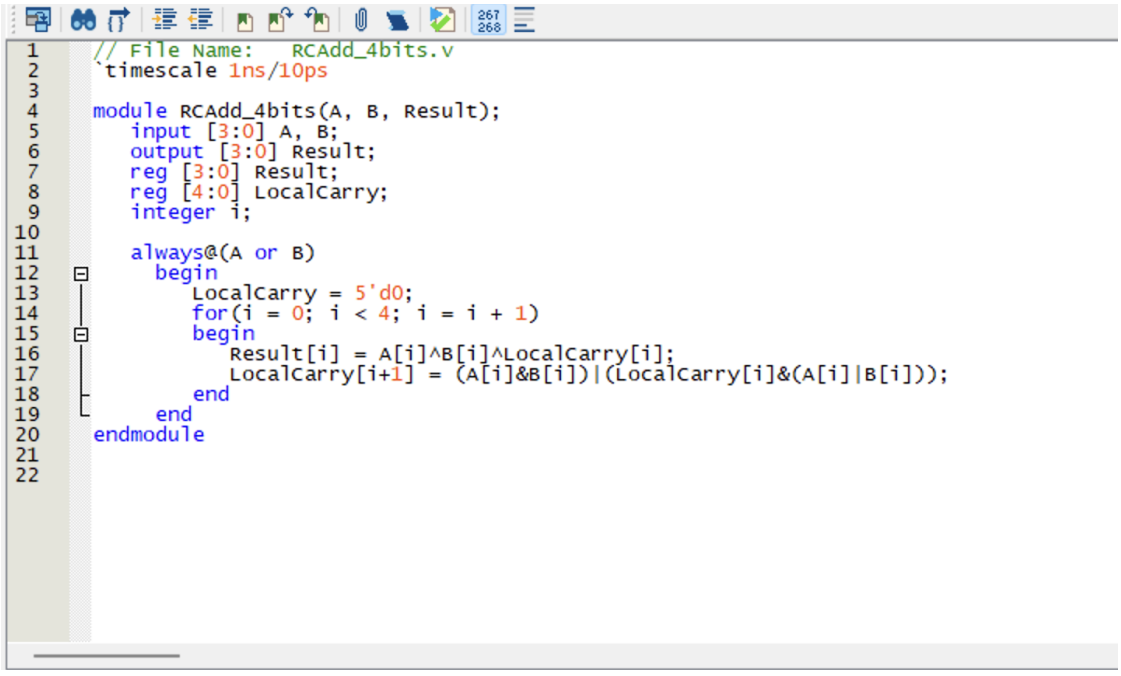
Signal Name	*FPGA Pin No.*	*Description*
LEDR0	PIN_AA2	LED [0]
LEDR1	PIN_AA1	LED [1]
LEDR2	PIN_W2	LED [2]
LEDR3	PIN_Y3	LED [3]
LEDR4	PIN_N2	LED [4]
LEDR5	PIN_N1	LED [5]
LEDR6	PIN_U2	LED [6]
LEDR7	PIN_U1	LED [7]
LEDR8	PIN_L2	LED [8]
LEDR9	PIN_L1	LED [9]

- Referring to the tables in the documentation (DE0-CV User Manual), you can see which pin numbers correspond to which components.
- To assign each of the pins to a specific I/O pin, open the project.

- Since DE0-CV has 10 slide switches, we cannot use the previous Verilog code as it uses two 8-bit input ports, which requires a minimum of 16 switches. For this section of the tutorial, you can create a new project called **RCAdd_4bits**, add a Verilog file to it and compile it. You can use the following Verilog code in the new file:

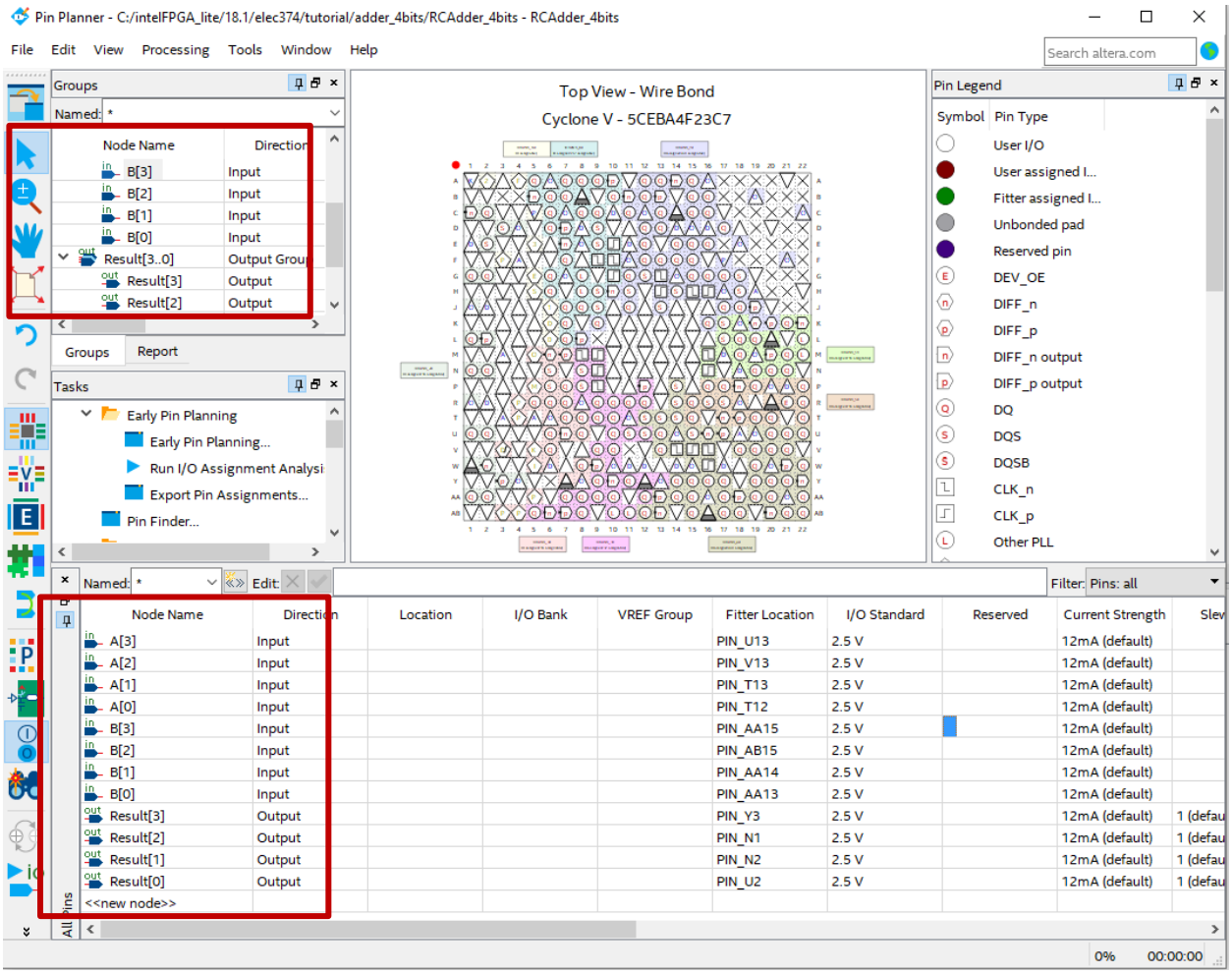
```
// File Name:   RCAdd_4bits.v
`timescale 1ns/10ps
module RCAdd_4bits(A, B, Result);
   input [3:0] A, B;
   output [3:0] Result;
   reg [3:0] Result;
   reg [4:0] LocalCarry;
   integer i;
   always@(A or B)
     begin
         LocalCarry = 5'd0;
         for(i = 0; i < 4; i = i + 1)
         begin
            Result[i] = A[i]^B[i]^LocalCarry[i];
            LocalCarry[i+1] = (A[i]&B[i])|(LocalCarry[i]&(A[i]|B[i]));
         end
     end
endmodule
//end
```

The screenshot of the Verilog code is shown below.



STEP 1: Make sure the RDAdd_4bits Verilog file is open and included in the project. The next step assumes that your project has compiled successfully.

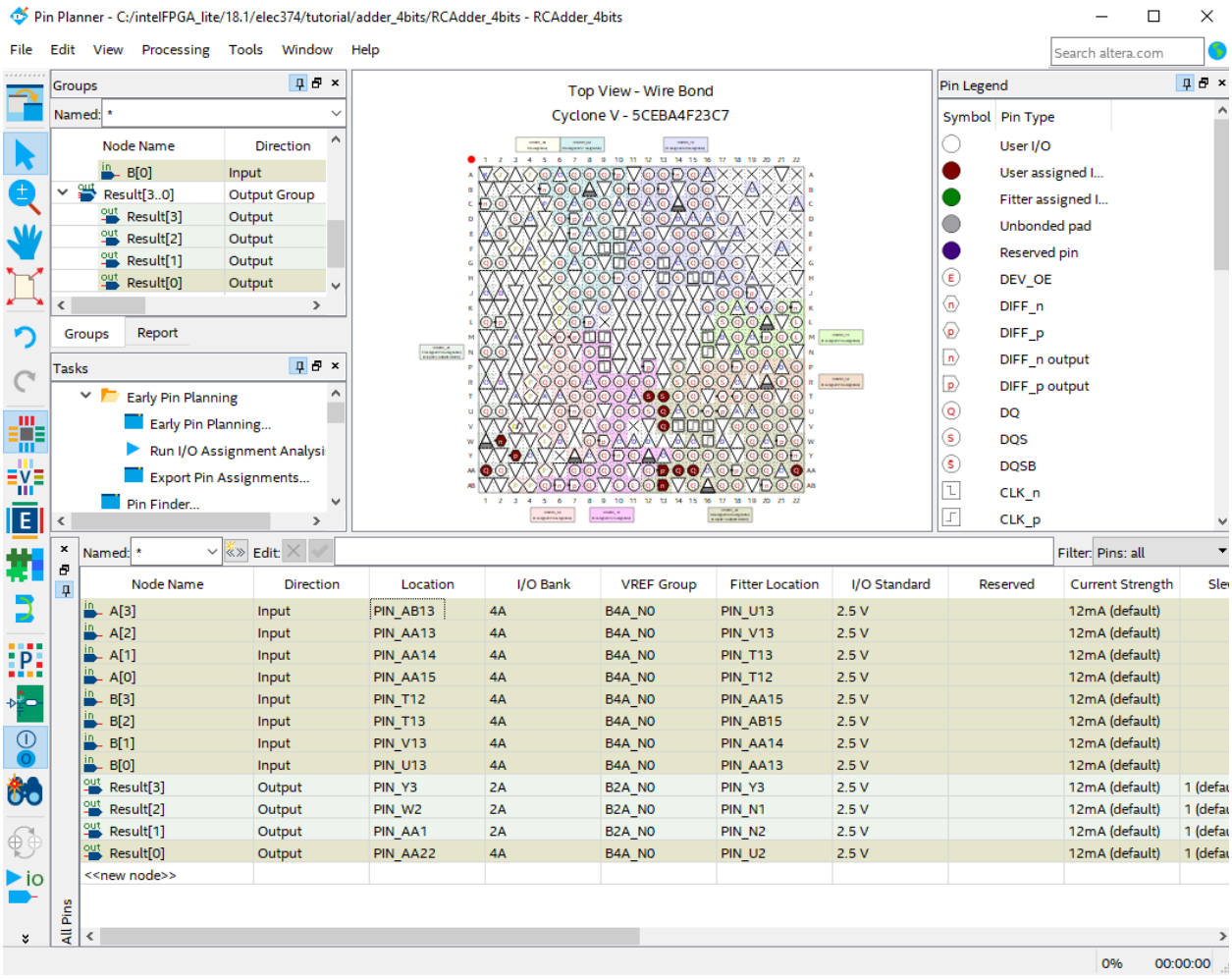
STEP 2: From the *Assignments* menu, select *Pin Planner*. The following screen will be displayed:



As you can see, the ports defined by your design (inputs 'A', 'B' and outputs 'Result') have already been added in this window.

STEP 3: Set up the pin assignment by giving the location for each pin of the ports according to the table below. For example, A[3] can be assigned as 'PIN_AB13'.

When you finish this step correctly, the *Pin Planner* window should be updated as follows:



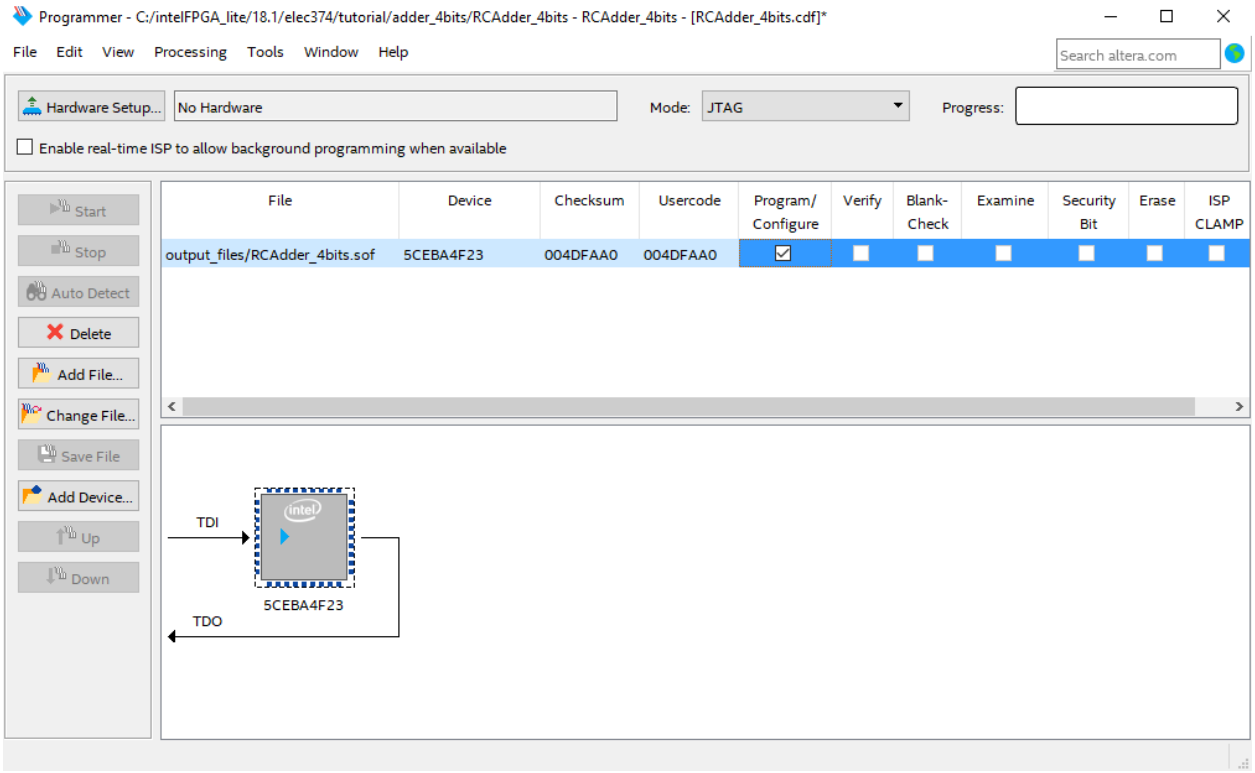
STEP 4 (OPTIONAL): To make sure your pinout setting is correct, you may generate a spreadsheet of it by selecting menu item *Export* under menu *File* of the *Pin Planner* window. You can then open the created file by Excel and examine whether it is consistent to the table used in **STEP 3**.

If the setting is OK, save all the files and close the *Pin Planner* window. You can also re-compile the whole project again at this moment to make it ready for the hardware test.

7.3 Assigning a Device

STEP 1: Click on *Assignments → Device* to bring up the *Settings* window. Ensure that the device selected is the *5CEBA4F23C7* of *Cyclone V*, and click OK.

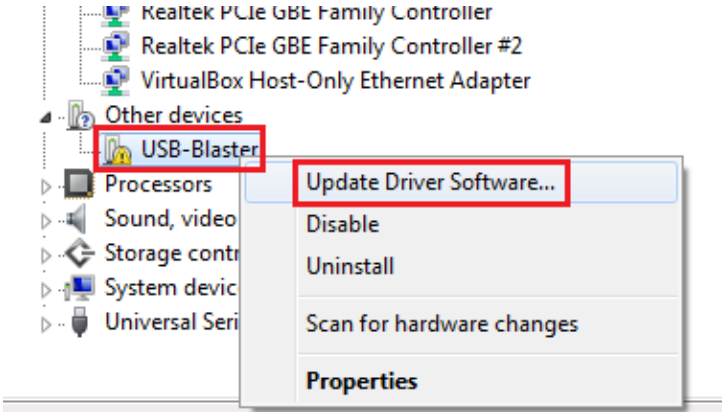
STEP 2: Click on *Tools → Programmer* to bring up the *programmer* window:



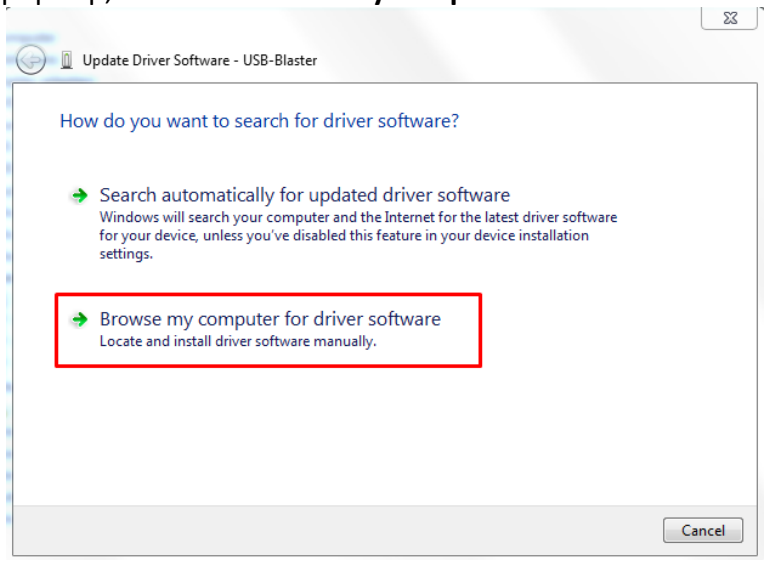
STEP 3: Now Power on the DE0-CV board and connect the USB cable to the USB port near the power connector. Next, connect the USB cable to your computer. If the "**Found new hardware wizard"** reports that it needs to install the drivers for the USB-Blaster device, follow these instructions; otherwise, skip ahead to the **Hardware Setup** section **(Step 4)**.

- Click on the Windows Start button and type "**Device Manager**" in the search bar. The search should find the Device Manager program. Click on it to show a list of devices and peripherals attached to your system.

- In the Device Manager window that opens up, right click on "**USB-Blaster**" under "**Other Devices**" and click on "**Update Driver Software…**".



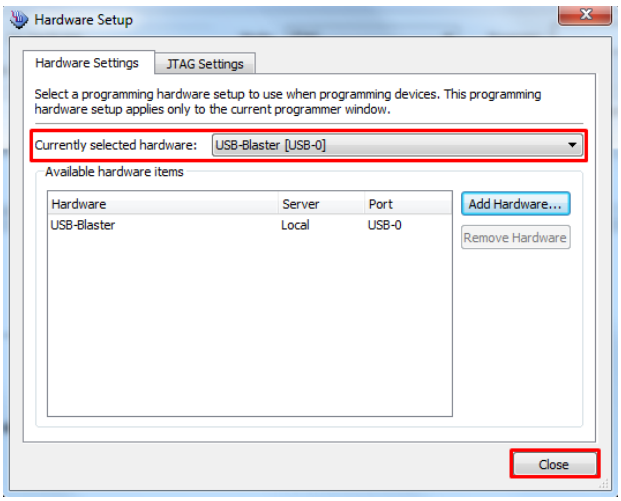
- In the window that pops up, click on "**Browse my computer for device software**".



- In the next window, click on "**Browse…**" and browse to the **Intel** directory on the local drive. Drill down into the base **Quartus** directory and select the **USB-Blaster** directory under **drivers**. Press **OK** on the file browser window and click **Next** on the new driver installation wizard. If you see a Windows Security dialog box, click on **Install.**

STEP 4: Now, back to the Programmer window, click on *Hardware Setup…* to bring up the *Hardware Setup* window:

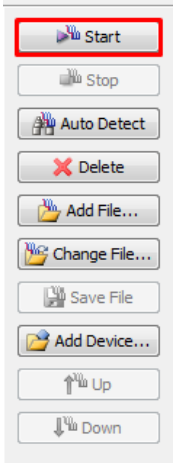
- Ensure that the *Currently selected hardware* is the *USB-Blaster (USB-#)* and then click *Close*. If the USB-blaster does not appear on the list, it may need to be installed, refer to the previous instructions on how to do this.



- Click on *Program/Configure* checkbox:

Important: Make sure the Run/Prog switch is set to Run!

STEP 5: Click on the *Start Programming* icon on the side menu:



The evaluation board should light up, the program should get downloaded, and you should now be able to test your design in hardware.

For example, you can turn on or off the switches to change the values of input ports. Then, you can observe the outputs presented by LED to see if they change accordingly or not. Also, you can simulate this 4-bit adder in ModelSim-Intel, and compare the results to the hardware behavior as an assessment of your design.

8 ADDITIONAL REFERENCES

There is an excellent online tutorial for learning how to use some of the more advanced features of the Quartus Prime package.

- Click on *Help → PDF Tutorials* to find the *PDF Tutorial for Verilog HDL Users,* and *PDF Tutorial for VHDL Users*.

The following are some Intel and Quartus related links:

- Intel FPGAs and Programmable Devices
- Intel Quartus Prime Software Support
- Intel FPGA Support Resources

The following links are some references to Intel VHDL and Verilog design examples:

- Intel's VHDL Design Examples
- Intel's Verilog Design Examples

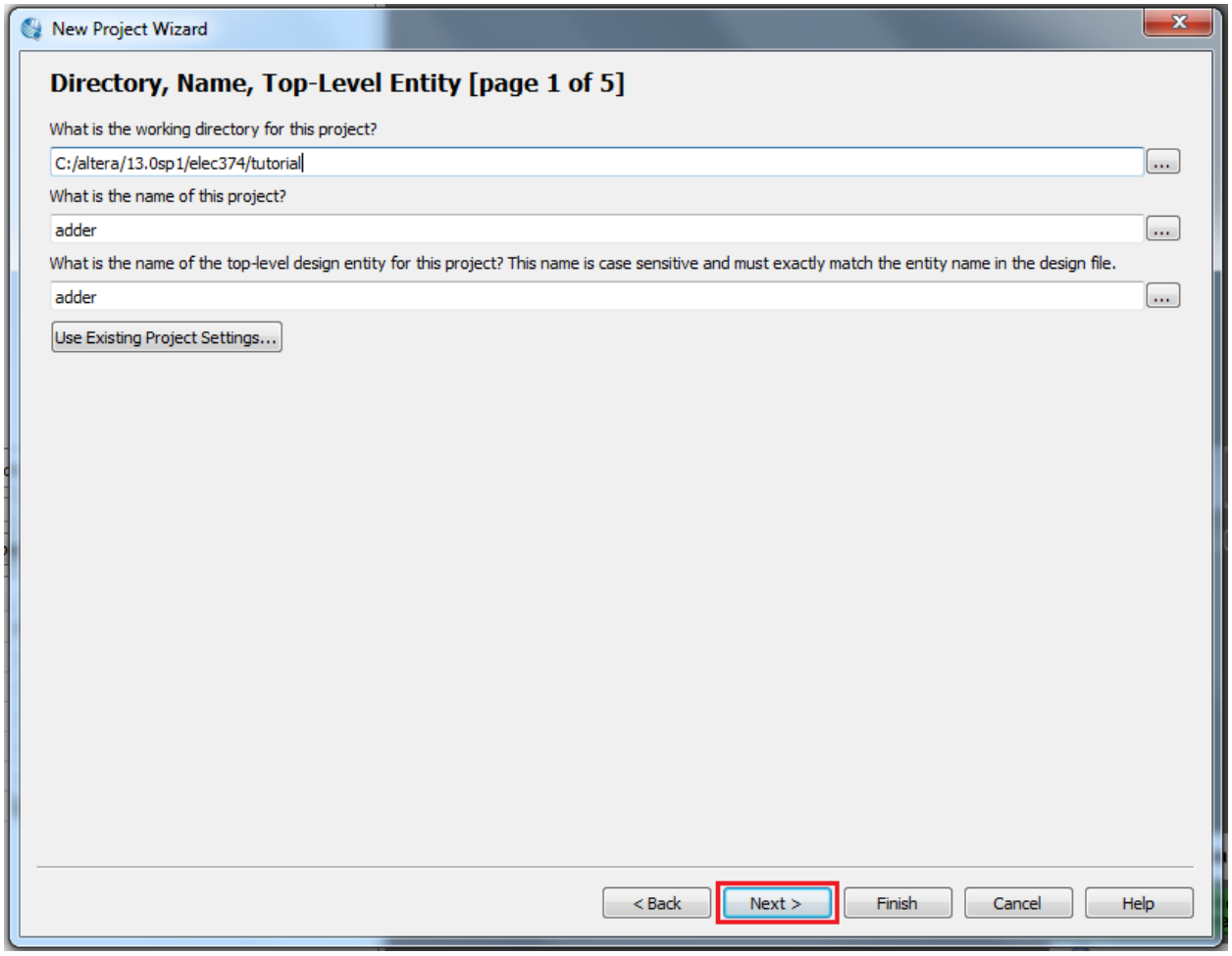
APPENDIX A - SCHEMATIC AND MIXED HDL-SCHEMATIC DESIGN

Appendix A is provided for legacy reasons and completeness. Although it is not recommended to use a schematic design or a mixed HDL/schematic design for this project, this section describes the process of creating simple Schematic or HDL/Schematic design in Altera Quartus II, v13.0 SP1. Similar design styles can be used in Intel Quartus Prime Lite Edition, v18.1.

In order to illustrate the basic working of the program, a simple schematic project will be created using a pre-made adder provided by Altera.

1. CREATING A NEW PROJECT

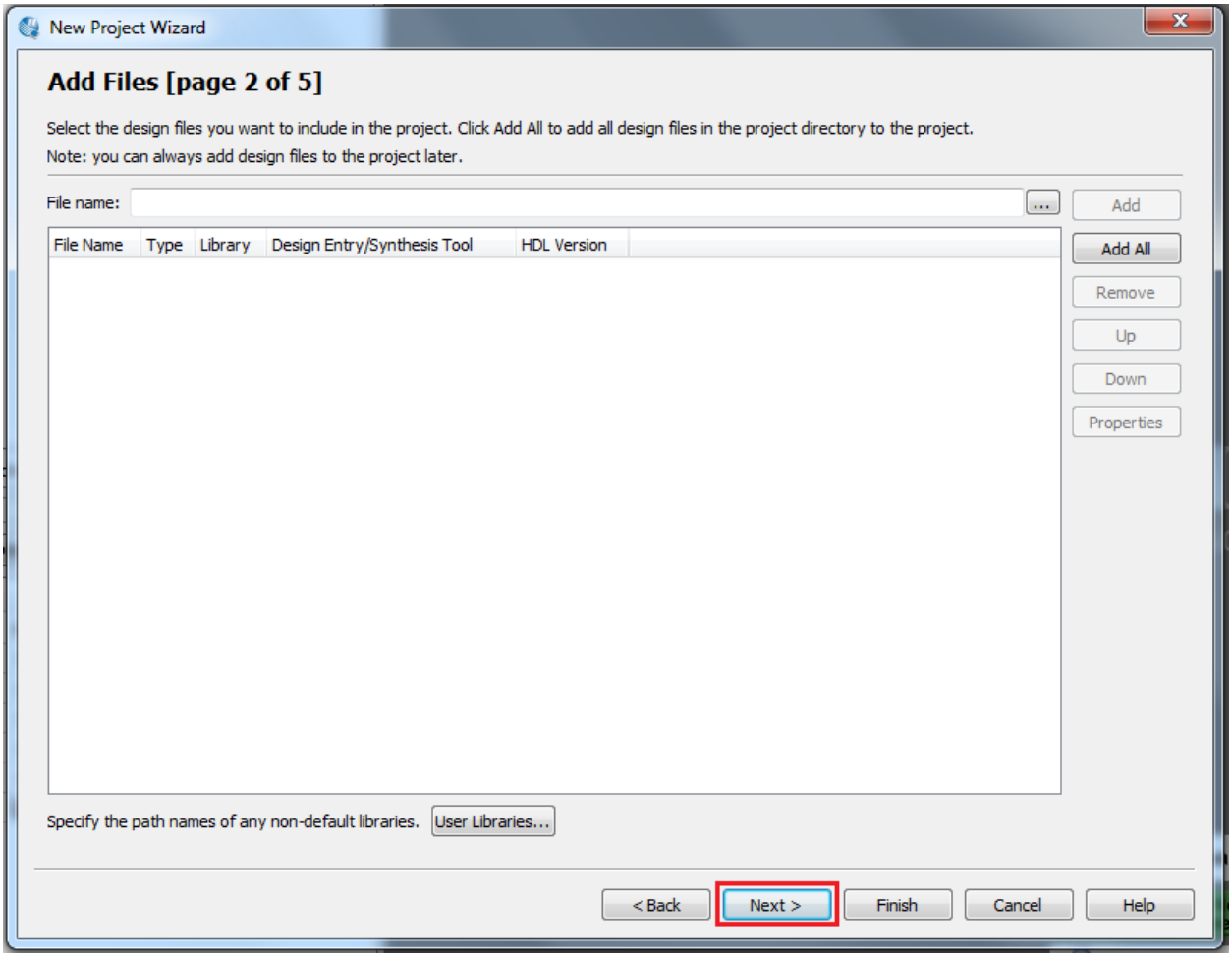
- Start the Quartus II program. Choose either the 64-bit version (if available) or the 32-bit version. In terms of functionality, you shouldn't face any difference.
- Click on **File → New Project Wizard**.
- After clicking **Next**, the following dialog box will pop up:



- For the *working directory*, create a new directory on your computer and select it.

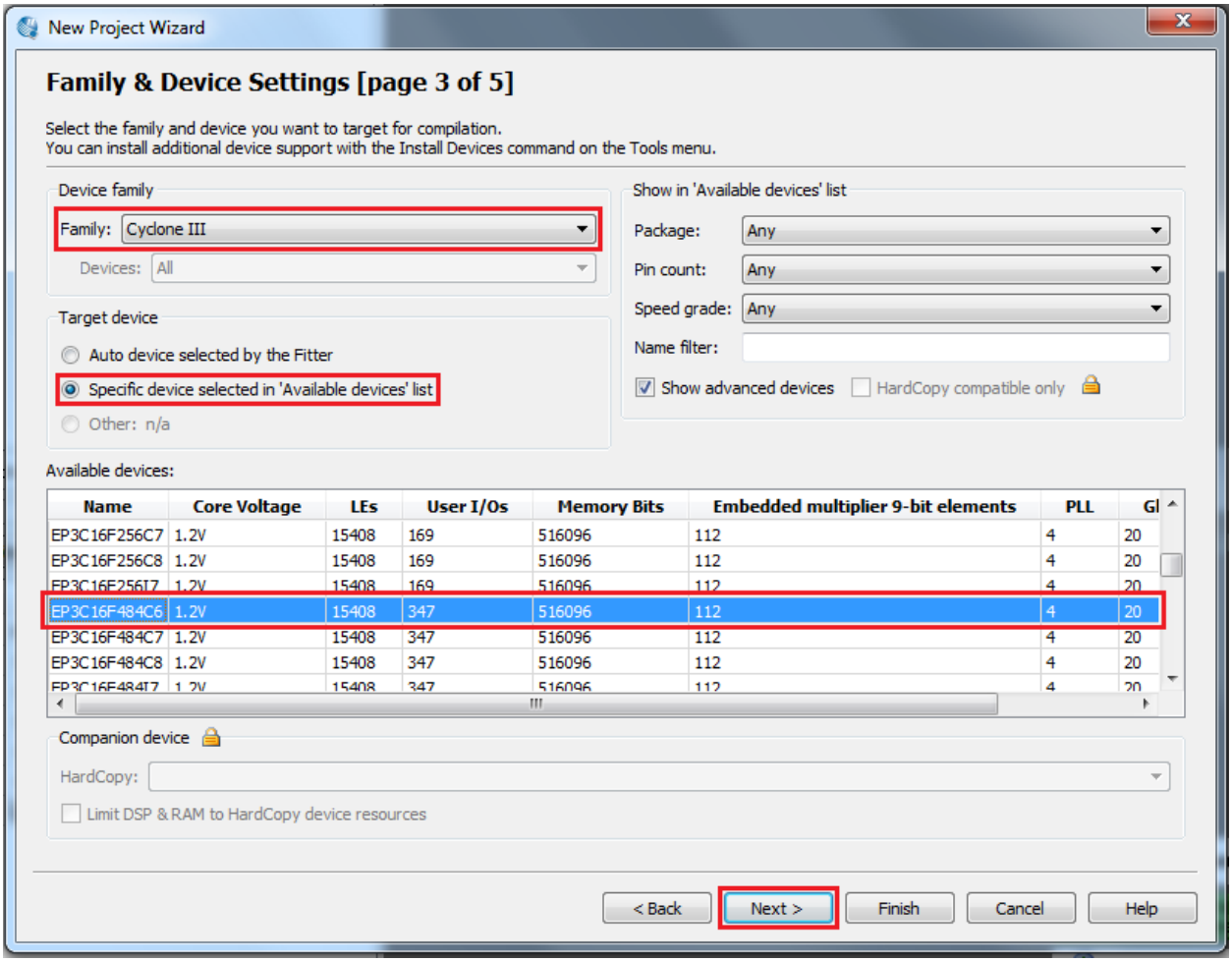
- *Note: Since Altera will create a bunch of project files, make sure you create a separate directory for your design (do not save everything in your root directory). For your first example, you may want to use "C:/altera/13.0sp1/elec374/tutorial".*
    1) For the *name of the project*, pick a reasonably descriptive name to describe the project *(for your CPU, you may want to call it "CPU"; for the example in this tutorial, we call it "**adder**")*.
    2) For the *name of the top-level design entity*, pick a name to describe your top level design *(In this example, we use the default name "**adder**", which is the same as the project name)*.

- After clicking **Next**, the following dialog box pops up. Since no design files currently exist, you can simply ignore this step and click **Next**.



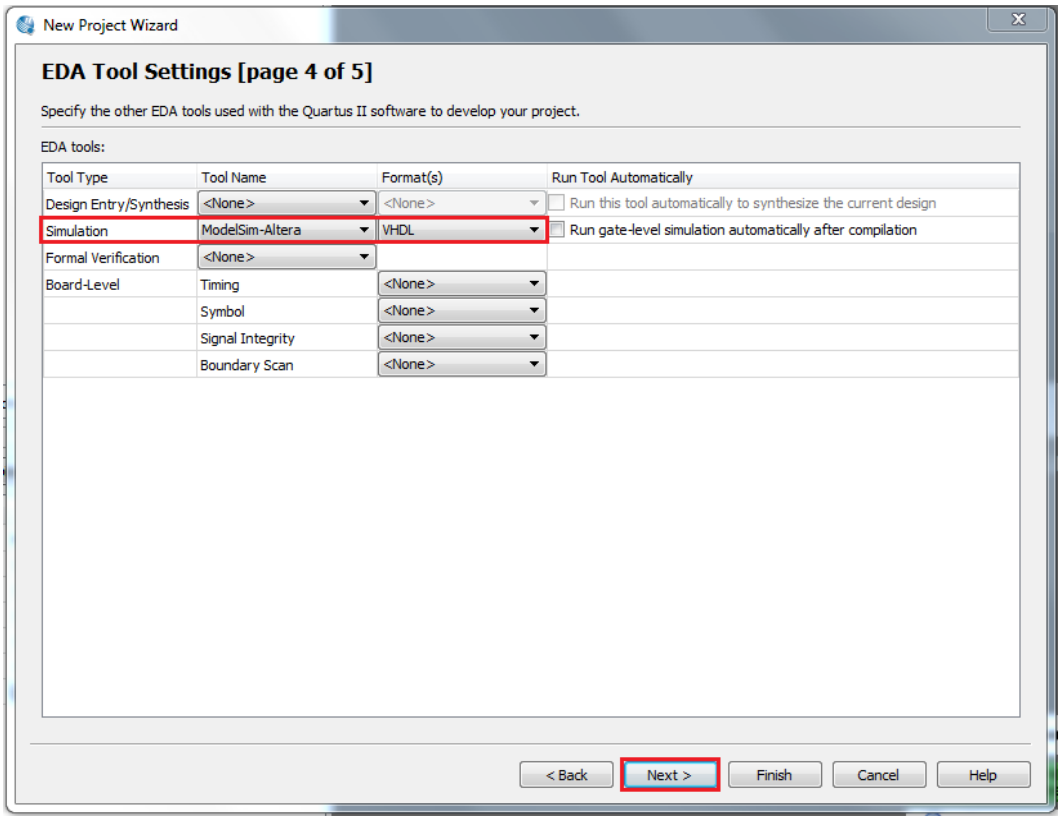
- After clicking **Next**, the following dialog box pops up. For DE0 board, select *Cyclone III* from the **Device Family** drop down list and ensure *"Specific device selected in Available devices list"* is selected under **Target device** category. Select *EP3C16F484C6* from the **Available**

devices list. If this device is not listed, ensure that all the **Filters** are set to *Any* or are set as shown below.

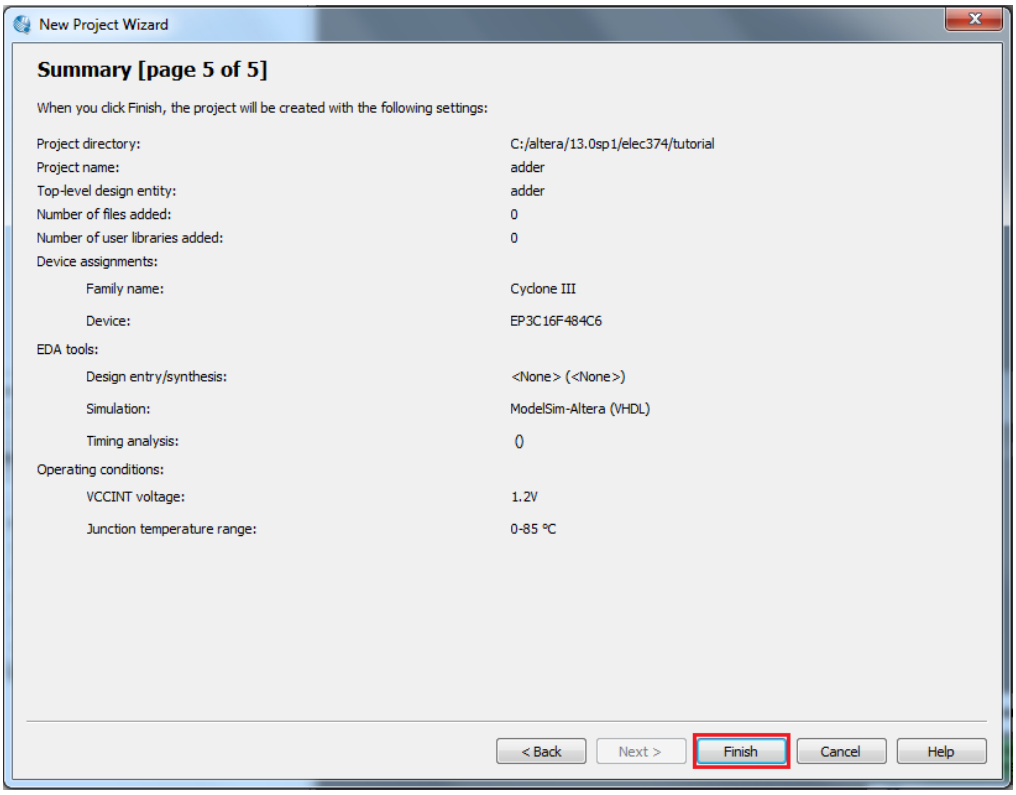


- Note: For DE0-CV board, select *Cyclone V* from the **Device Family** drop down list, and select *5CEBA4F23C7* from the **Available devices** list.

- After clicking **Next**, the following dialog box pops up. We will be using an external EDA (Electronic Design Automation) tool, ModelSim-Altera, for our simulation. Hence, make sure you choose this tool in the **Simulation** section under **Tool Name** category. Depending on your choice of HDL, you may choose **Verilog** or **VHDL** from the **Format** menu.

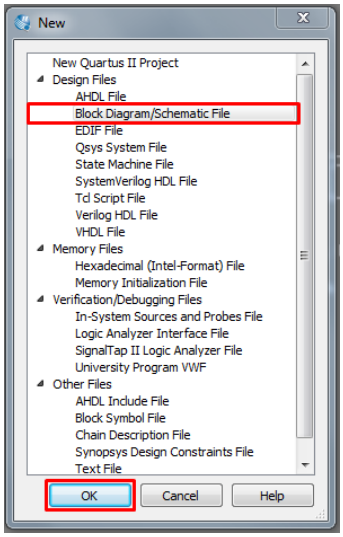


- After clicking **Next**, the following dialog box pops up. Verify your settings and click **Finish** to complete the New Project wizard.

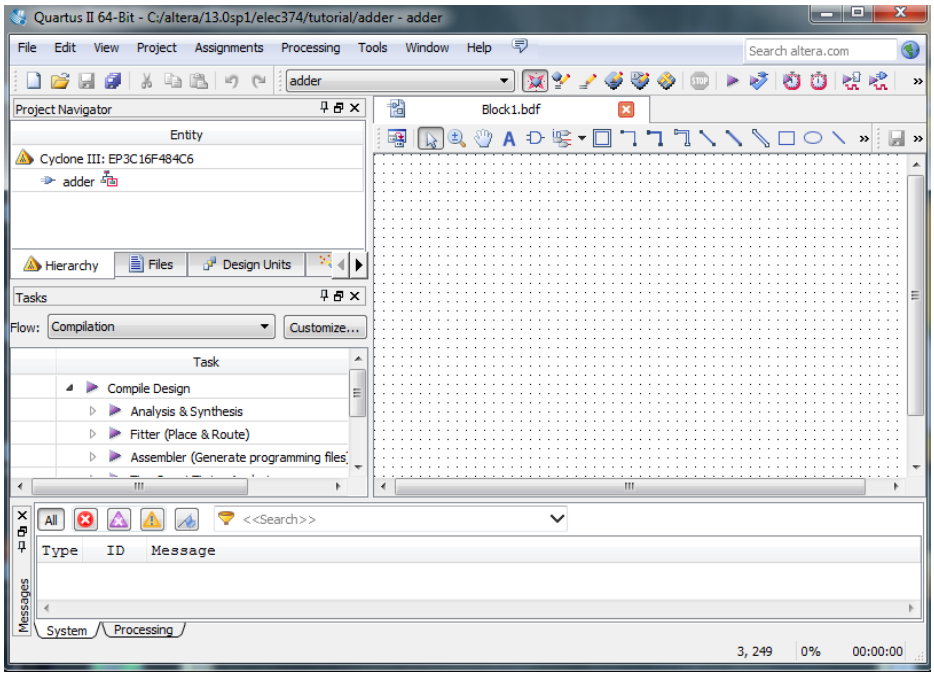


2. *CREATING A BLOCK DIAGRAM FILE*

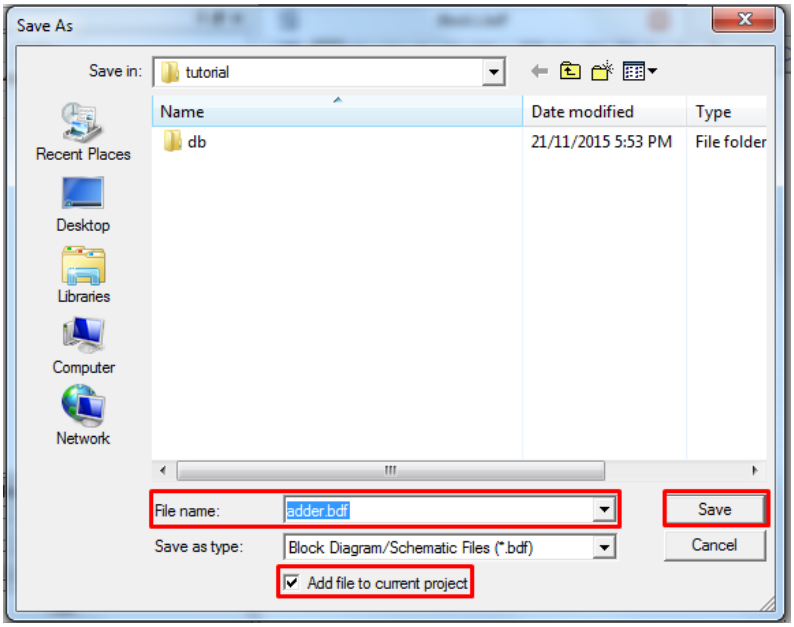
- Click on **File → New**.
- The following dialog box will pop up. Select *Block Diagram/Schematic File* from the **Design Files** tab.



- After clicking **OK**, you will see a new drawing window:

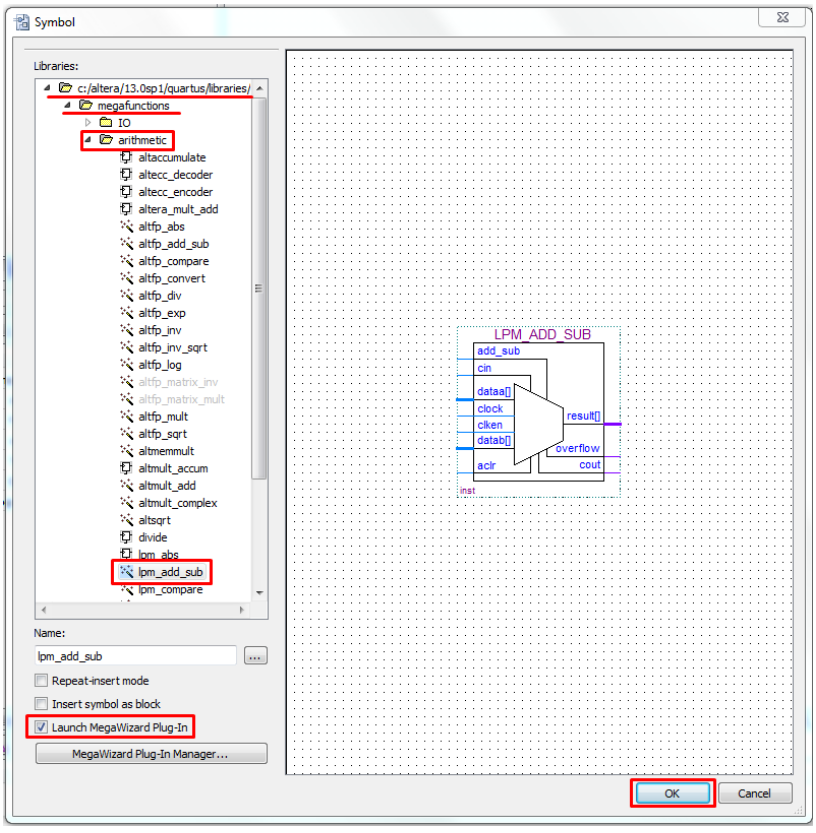


- Since this will be the top-level entity of this project, click on **File → Save As** and save it as *<top level entity>.bdf* where *<top level entity>* is the case-sensitive name you specified as your top-level design entity in Step 1 of the **New Project Wizard** (*e.g.*, adder.bdf for this tutorial). Ensure you check the box "*Add file to current project*".



3. *ADDING NEW SYMBOLS*

- After saving, double-click somewhere in the middle of the design window to bring up the *Symbol* wizard:

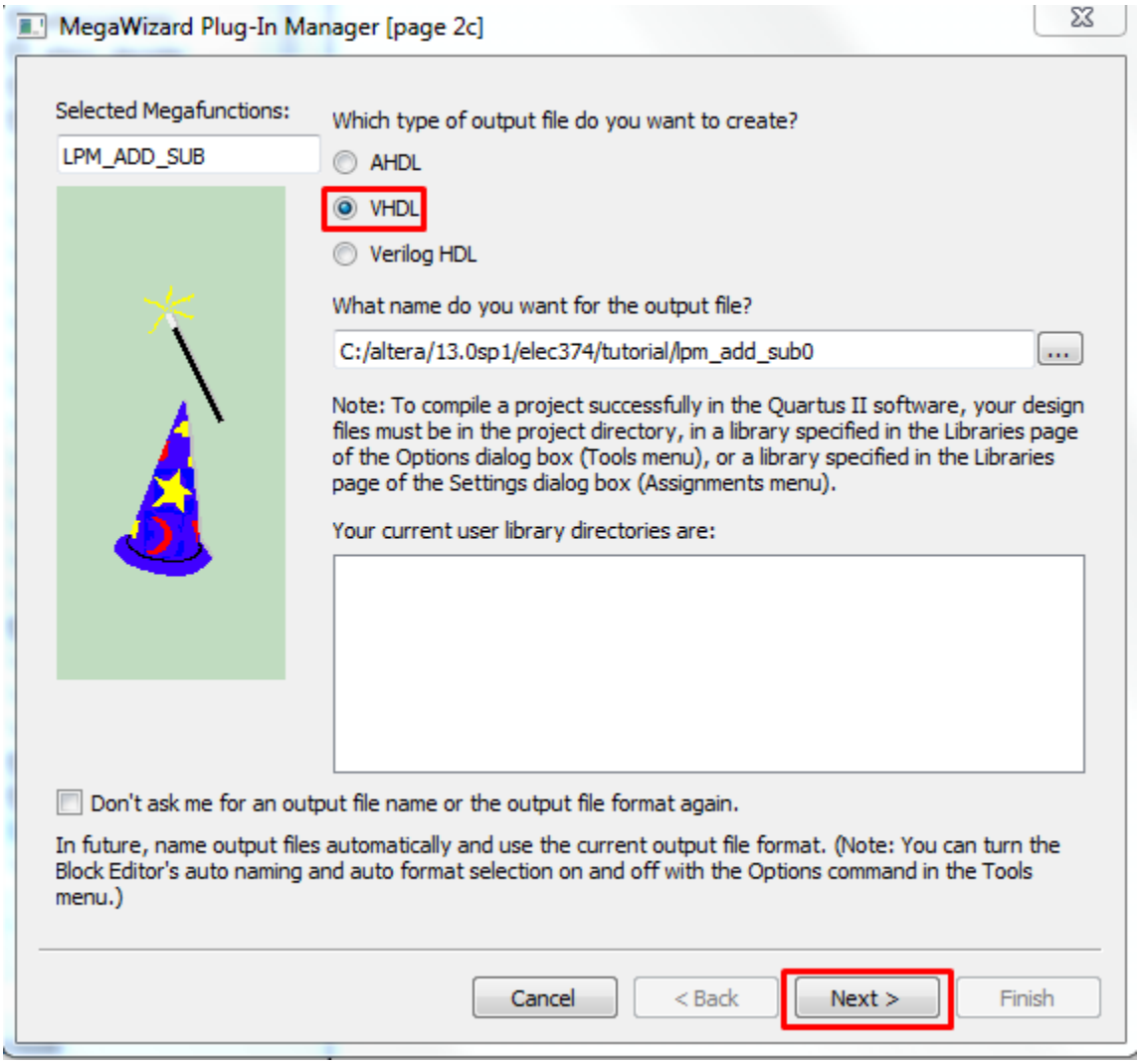


- Click on the libraries path [C:/Altera/13.0sp1/quartus/libraries/] → **megafunctions** → **Arithmetic** → **lpm_add_sub**. Ensure the "*Launch MegaWizard Plug-In*" is checked and click

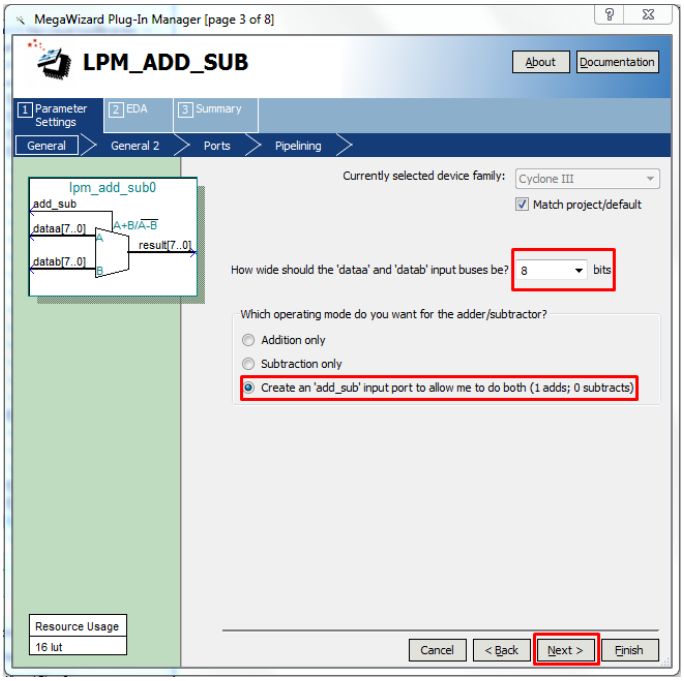
OK. *(Note: You can select basic discrete components directly by navigating the Libraries tree directly or by typing the name of the component into the Name dialog box to search the Libraries tree. However, for megafunction blocks – such as ALUs, Shifters, MUXs, Memory, etc. – be sure to use the Wizard instead even though those components also can be found in the tree.)*

- In the MegaWizard manager, ensure you choose the type of output file as **VHDL** or **Verilog** HDL based on your choice of programming language. Your testbench will also have to be in the same language that you choose here.

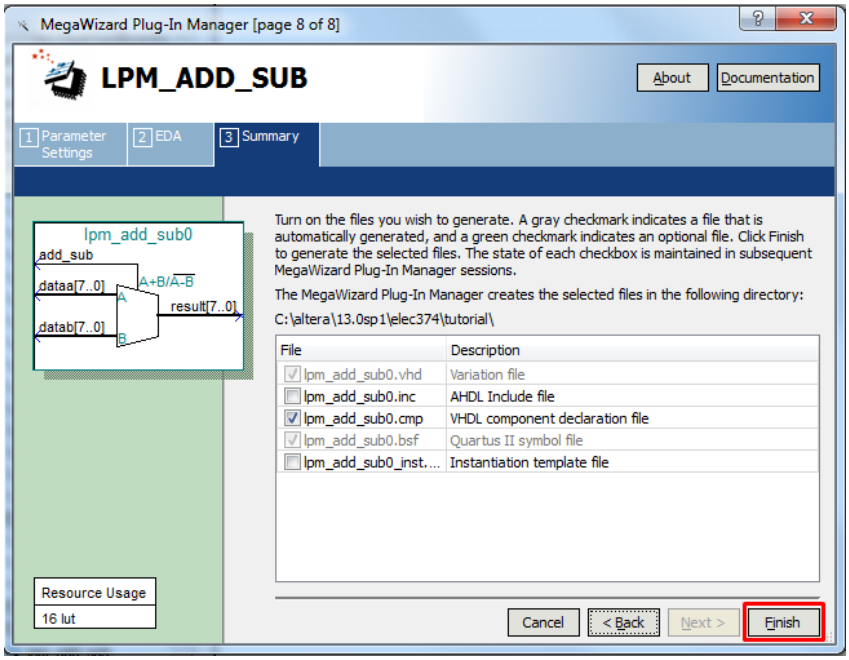
- You can let the default file output name be chosen or provide a name of your choice to the generated HDL output file. Then click on **next**.



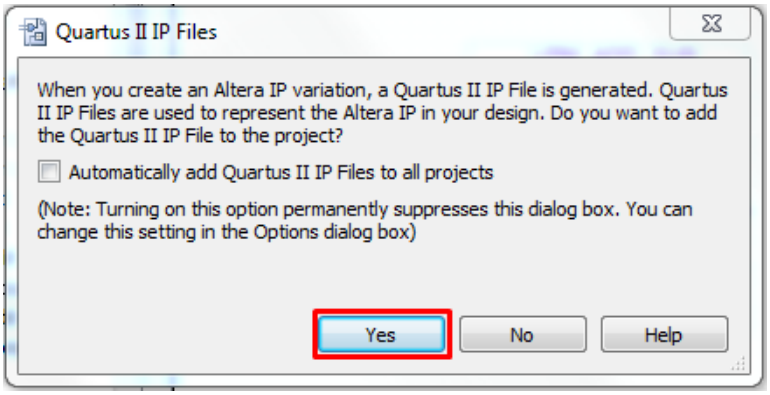
- The **lpm_add_sub** module pops up, letting you choose the data inputs and outputs from a GUI interface. Choose "**Create an 'add_sub' input port to allow me to do both (1 adds; 0 subtracts)**" and "**8 bits**" for the data width of '*dataa'* and '*datab'* and click '**Next'**.



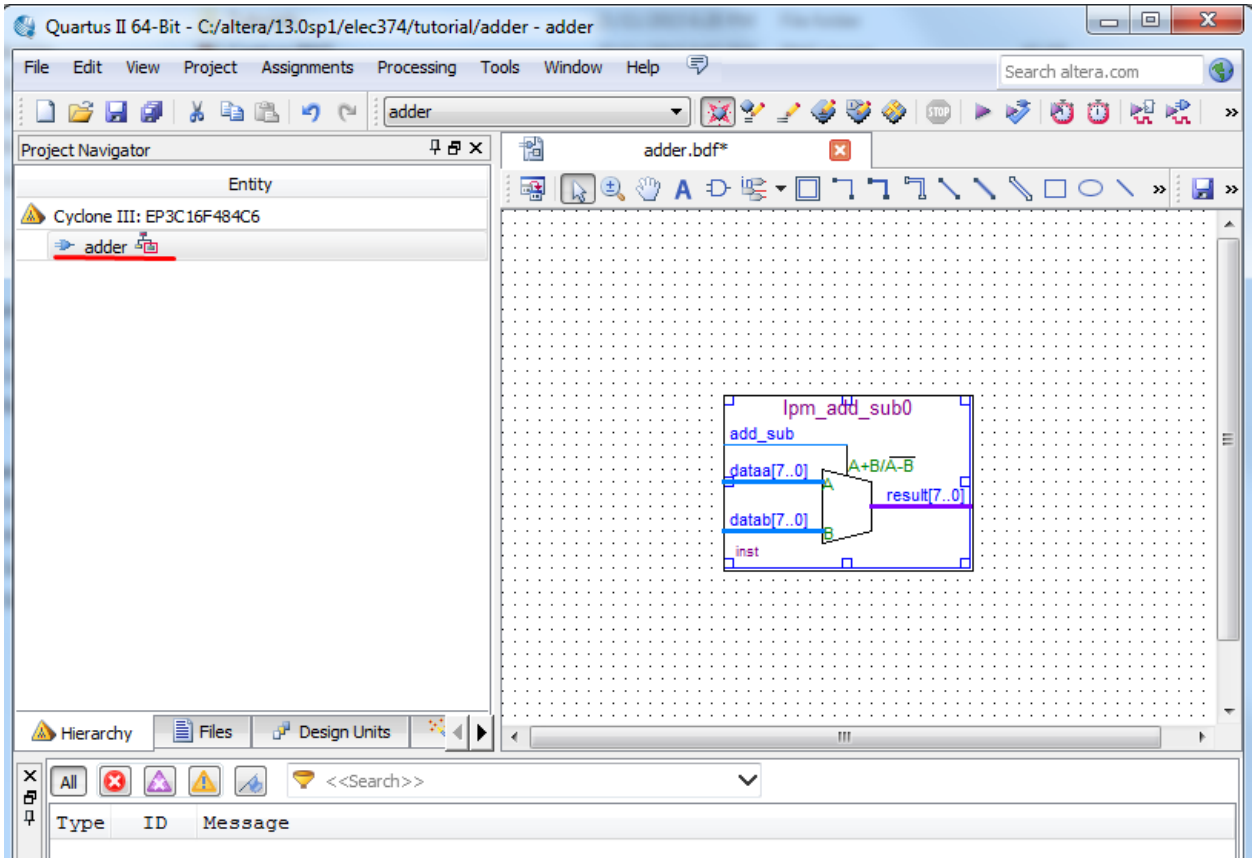
- Continue clicking '**Next'** to accept the default setting in the next few screens, till you reach the following page and click '**Finish'**.



- For the next pop-up, just click **Yes**.

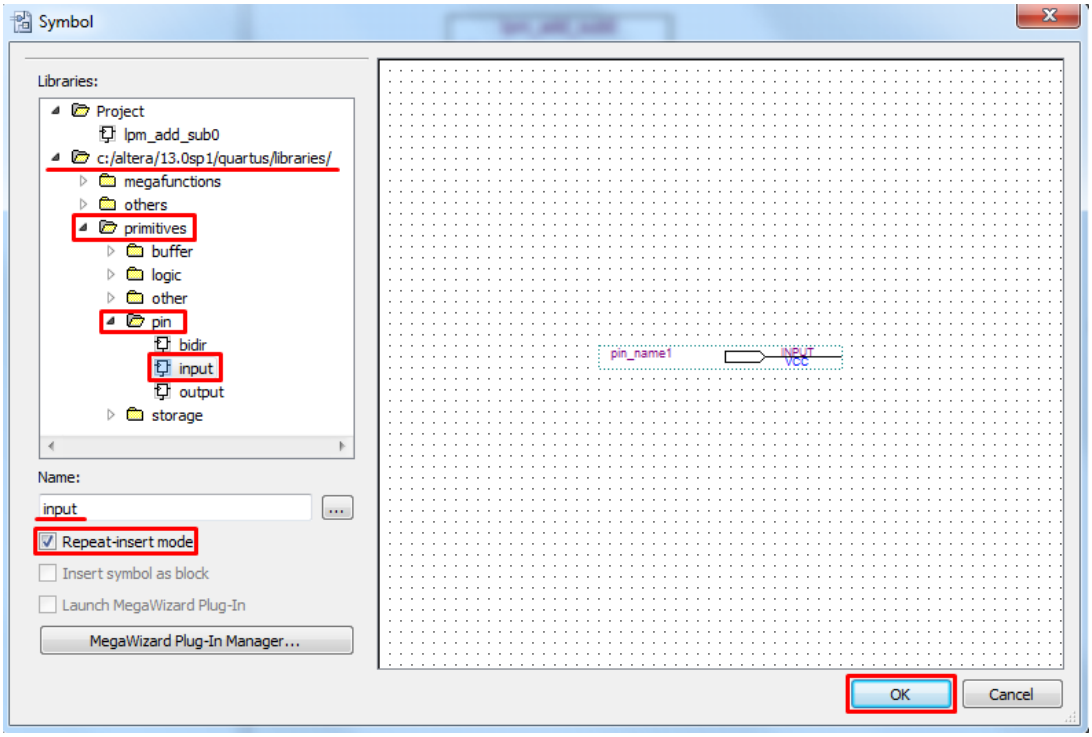


- Now you'll be back to the main window where you can decide where you want to place your newly created symbol. Click somewhere in the middle of your design window to place the new *lpm_add_sub0* symbol.

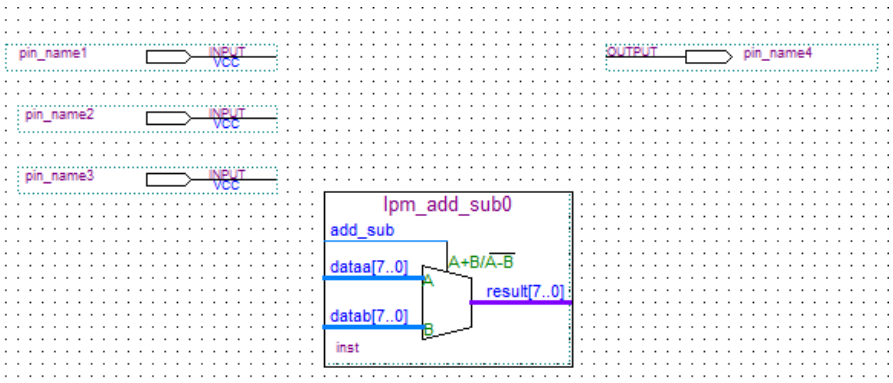


- To create *inputs* and *outputs* for the adder, double click in an empty area of the diagram to bring up the **symbols wizard** again.
    1) Under the libraries path, navigate to *primitives → pin*, select *input* or simply type "*input*" into the **Name** dialog box.

2) Since we need three inputs for the adder; two for data inputs, and another to control the mode of operation (add/sub), we check the *Repeat-insert mode* box and click **OK**.
3) Click *thrice* somewhere above the LPM_ADD_SUB module in your block diagram to add three *input* pins (make sure the symbols do not overlap) and then press *ESC*.

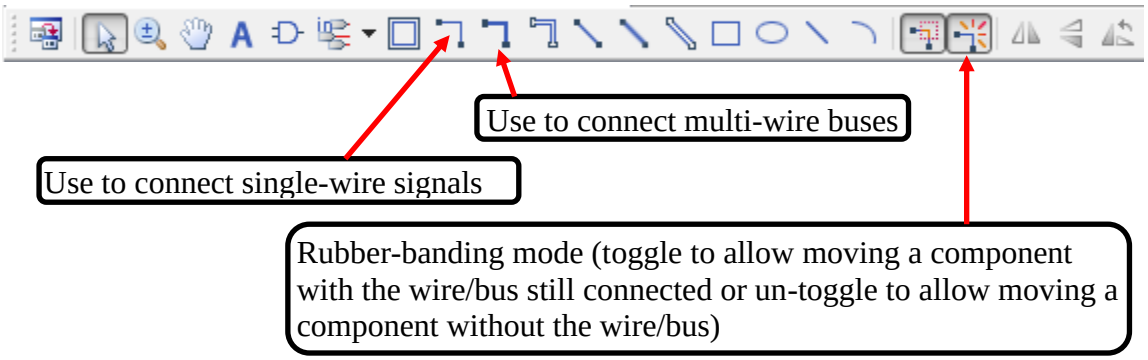


- Follow the same procedure to add one *output* pin (select *output* from the symbols menu instead of *input* and uncheck the *Repeat-insert mode* since we only want one output).
- Click once somewhere above the adder in your block diagram to add an *output* pin.
- The resulting block diagram should look something like this:



4. *CONNECTING SYMBOLS*

- To connect the symbols, we use the menu on the top:

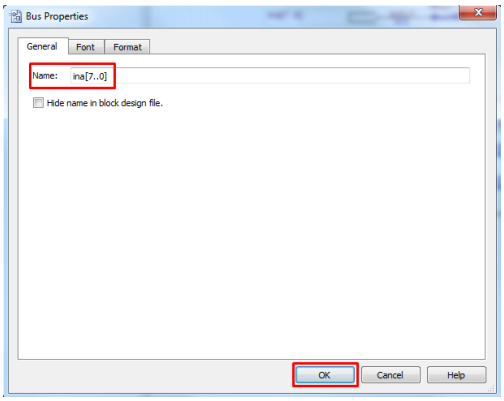


- Since the *adder* we are using has an 8-bit bus, we want to use multi-wire buses by selecting . Click on the icon, then click on the edge of any of the input pin components, where there is a line and a label, and drag from there outwards to draw a line. Repeat this procedure for one other input pin, the output pin, two input data ports of the adder *lpm_add_sub0* and the output data port of the same adder.

- Similarly, for single-bit signals, select , click on the edge of each component where there is a line and a label, and drag from there outwards to draw a line. Do this for the remaining input pin and the add_sub input control port of the adder.

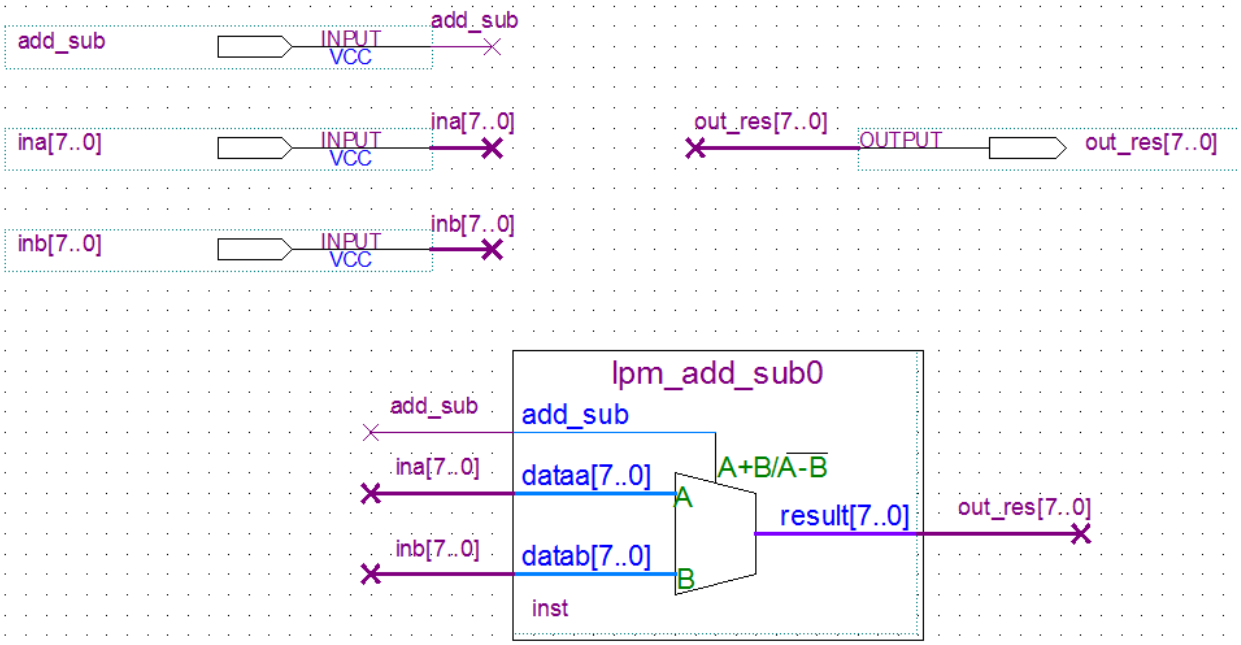
- In terms of actually connecting the components, you have two options: you can either connect the wires directly or you can leave them unconnected and use labels to accomplish the same thing. The latter option is preferred in a more complicated design as it keeps the layout cleaner and easier to follow (you do not have to follow long, crisscrossing wires to determine what signal is associated with which wire). *In your CPU design, use the labeling method rather than wiring everything up directly unless you're absolutely certain it would be just as clear or clearer to wire the blocks directly together.*

- To label a line:
  - select the line by clicking on it
  - right click and choose **Properties**
  - type the label next to the **Name** tab as shown below.



- For single wire signals, you can enter a regular label name like *add_sub* as used in this example.
- For buses, you have to specify the size of the bus and the order of the wires in the bus. For this example, use a label such as *ina[7..0]* to signify an 8-bit bus, with the most significant bits being entered first *(do NOT put ina[0..7] as the bits will be reversed)*. Also, note that if you want to look at specific signals of a bigger bus, you can enter labels such as *ina[3]* or *ina[5..3]*.
  Note: When working with buses, make sure you use the bus wires and not the single line wire. Also, ensure that you label them correctly using the wire_name[n..0] notation.
  WARNING: Avoid ending bus and pin labels with numbers as the system can interpret them as pins of a different bus. This warning will be repeated again as it is the source of many design faults.
- To label an I/O pin, click on the label to rename it (using the same bus notation described above).
- Finish labeling all input and output pins as well as the connector labels (12 labels in total).
  - To make design entry faster, try selecting a bus and its name, copying it, and pasting it – this way, you don't need to draw the bus and type its name twice. You can also copy elements by holding the *ctrl* key while dragging them to their new locations.

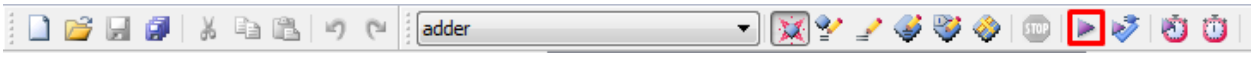
- On connecting all the symbols, your schematic should look as shown in the following figure:



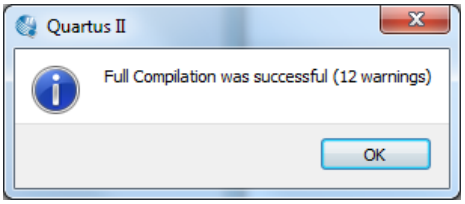
- Once complete, click on **File → Save** to save your design.
  Note: Quartus II is known for crashing at the least convenient times. Save your designs often and save separate backup copies once you have something working in case your design files ever get corrupted in a crash.

5. *Compiling the Design*

- To start compiling a design, click on:

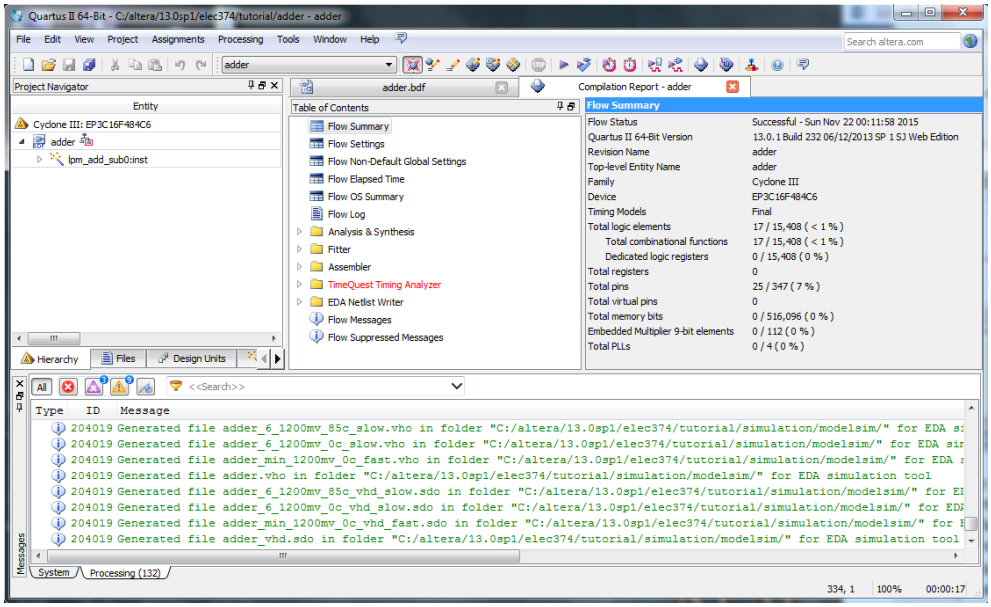


- When the compilation is complete, you will see the following message:

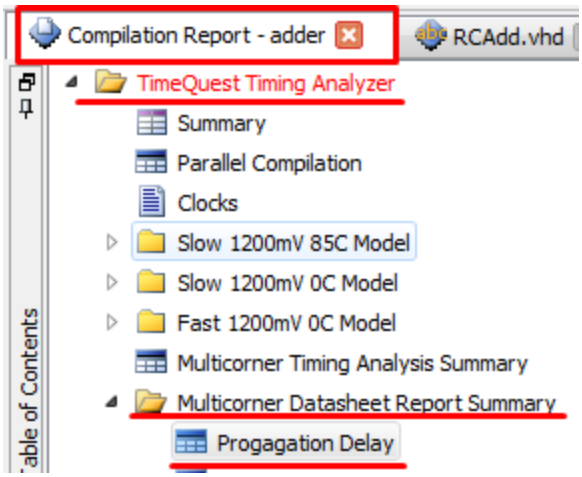


If you received errors or warnings apart from the expected 12 warnings, check your design to ensure you haven't misspelled any of the labels and that all bus wires are properly connected to the wires of the components. You should only see an "X" on the unconnected end of the wire; if you see an "X" between the wire and the component, the wire is not connected to the component.

- The main window will show various statistics on the design, as shown below.



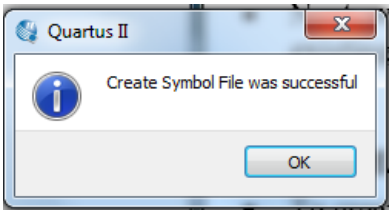
- These can be very helpful in analyzing your design. For instance, if the compilation generates errors or warnings, check the messages pane at the bottom of the window for details on what those errors or warning are and how you might fix them. By browsing through the reports, one can find such information as the anticipated worst-case propagation delay (t_{PD}) through your circuit, which determines the worst-case maximum clock speed for correct operation (**Compilation Report tab → TimeQuest Timing Analyzer → Multicorner Datasheet Report Summary → Propagation Delay**).



This completes the section on creation of a Block Schematic and its compilation.

6. *Creating a Block Symbol from HDL file*

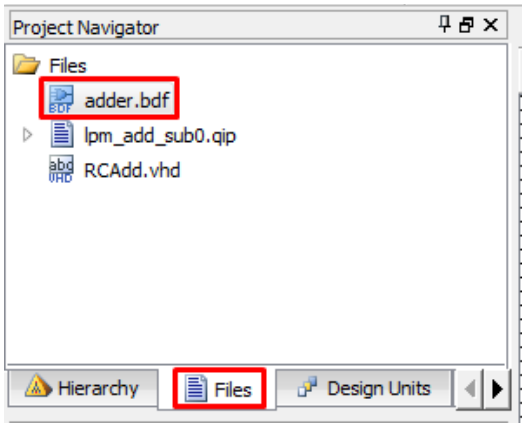
- To create a block schematic symbol for the newly created ripple carry adder, open the RCAdd.vhd file under the **Files** tab in the **Project Navigator** window (this file will likely be already open at this point) and click on **File → Create/Update → Create Symbol Files for Current File.**
- When the process is complete, you should see the following message:



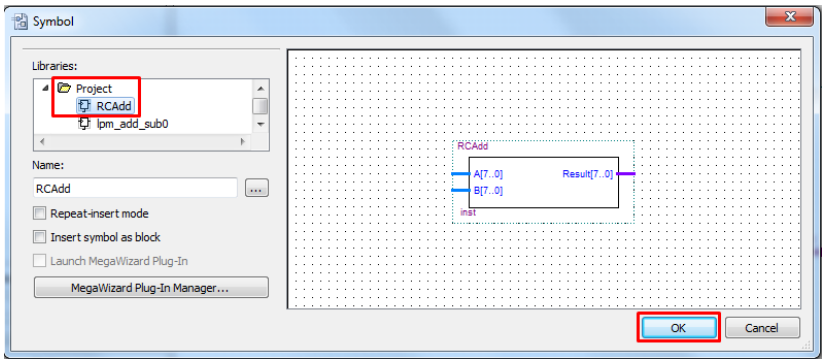
- If you received any errors, make sure you didn't mistype any of the code.

Adding Block Symbol to Project

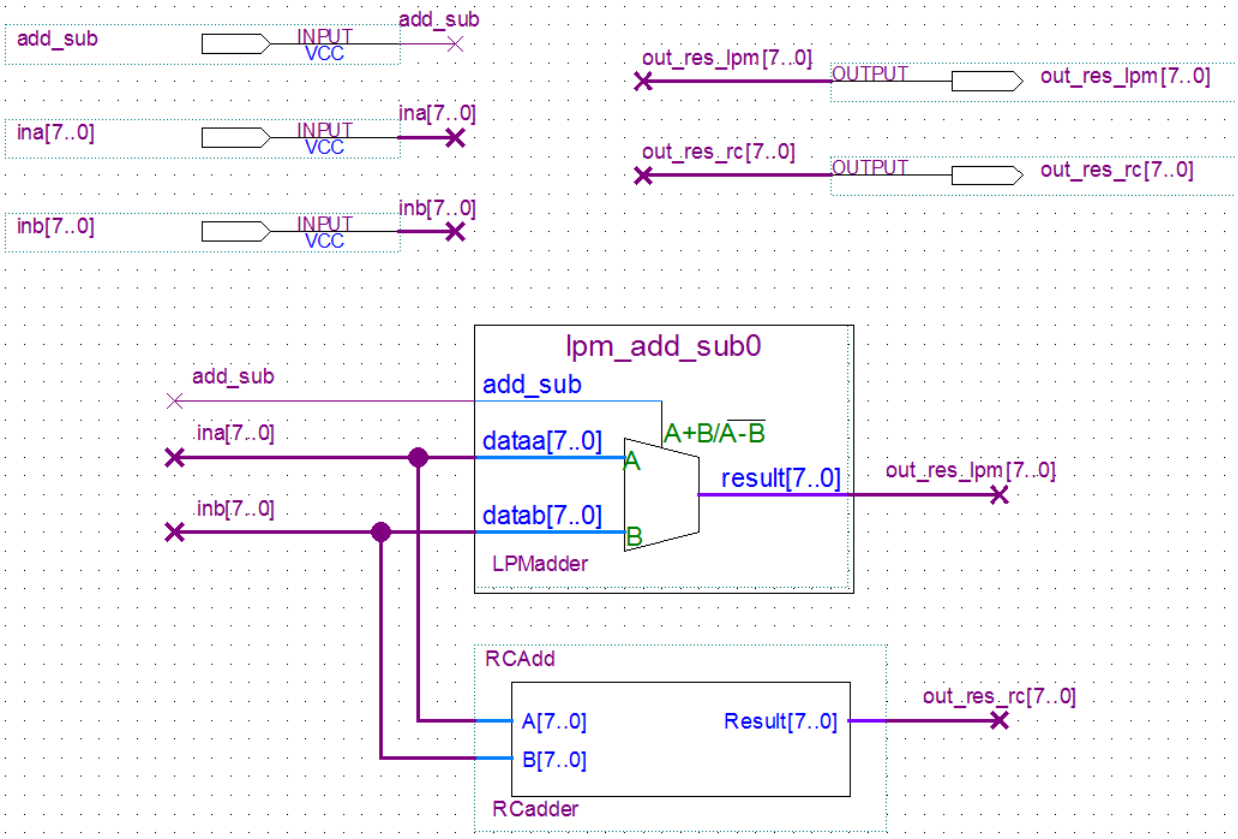
- Open the original block diagram file *adder.bdf*.
- If all your windows were closed and you do not see your block diagram file, you can click on the *Files* tree under the *Files* tab and double click the file to open it.



- Double click somewhere in an open space of the design window of the *adder.bdf* file to bring up the *Symbol* wizard:



- Under the *Project* tree, you should now see *RCAdd*.
- Select it and click *OK*.
- Now place this component in the design window (adder.bdf) along with the LPM_ADD_SUB module's schematic.
- Rename the two adders by clicking on their 'inst' textboxes to 'LPMadder' and 'RCadder' respectively.
- Connect up the components to create a combined schematic that is connected as shown in the figure below: (create a new output pin out_res_rc[7..0] as well). This will have the LPM_ADD_SUB block schematic and the schematic of the RC Adder (RCAdd) generated from the VHDL code.



Note: Be careful what you call the second output pin. If your first output pin was called *out_res[7..0]*, do *NOT* call the second output pin *out_res2[7..0]*, as this will cause a duplicate naming conflict (Quartus II seems to internally refer to each individual signal of bus *out_res[7..0]* as *out_res0*, *out_res1*, *out_res2*, etc., and this causes a conflict with the second bus signal *out_res2[7..0]*). As a rule, **always** ensure that one node name is not the prefix of another node name that is followed immediately by a single number as in the above example. In general, don't include numbers as the last character of a node or bus name. In this instance, the output pins were called *out_res_lpm[7..0]* and *out_res_rc[7..0]* to avoid this problem.

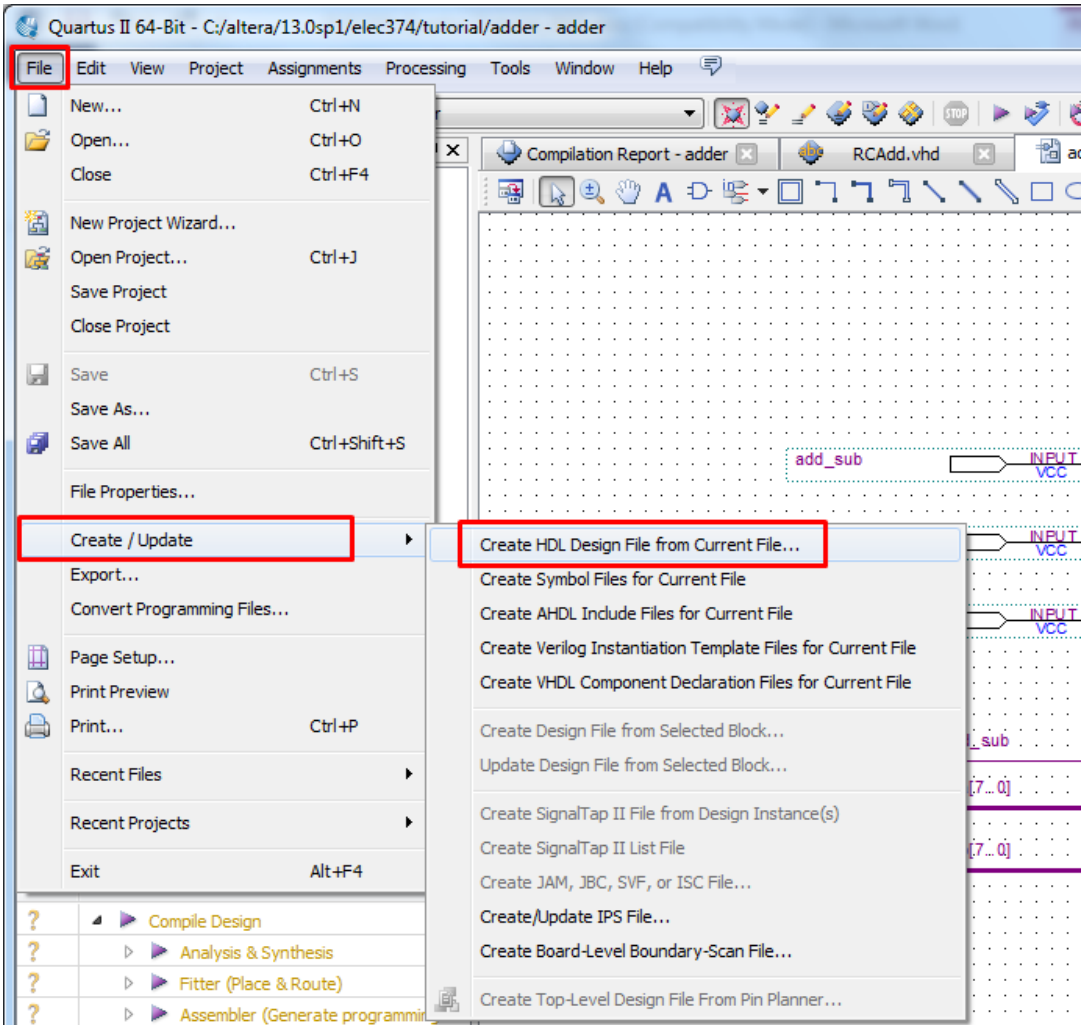
- Save (**File → Save As** and not **File → Save**) the block schematic design file with a new name **adder_lpm_rc.bdf**.

7. *Creating And Setting up the TestBench for Simulation*

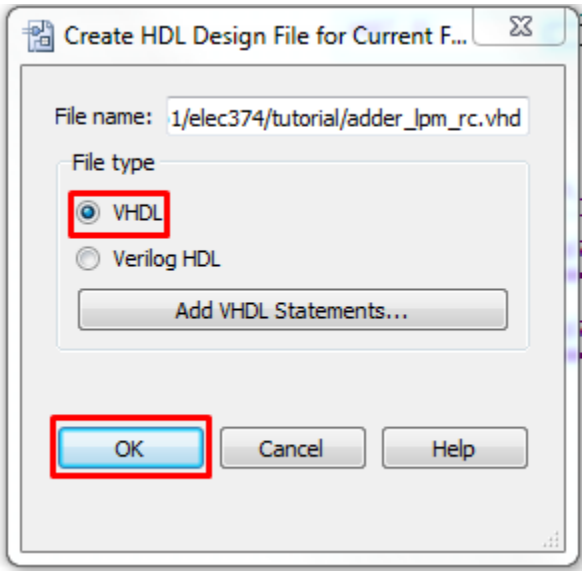
- To simulate the design, we will first generate a HDL file of the block schematic that we just created.
- We will then write a corresponding HDL testbench file and use it to simulate the design using an Electronic Design Automation (EDA) simulation tool like ModelSim.

Generating an HDL File from a Block Schematic

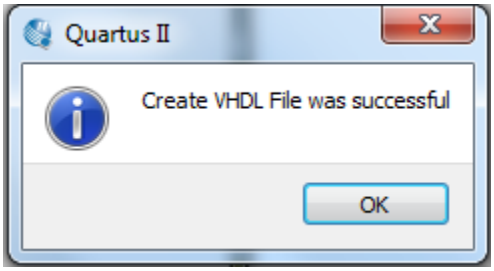
- We will use a HDL file to simulate a design using Quartus II V12.0 and above. Hence, we generate a HDL code from the combined schematic **adder_lpm_rc.bdf**
- We open the combined schematic of LPM_ADD_SUB module and RCAdd: "**adder_lpm_rc.bdf**" and navigate from **File-> Create/Update-> Create HDL Design File from Current File** to create a HDL output file.



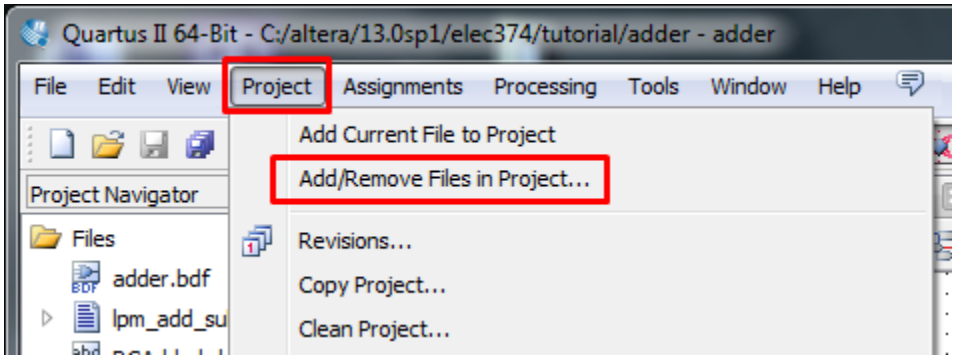
- Choose **VHDL** format for the output file in the sub-sequent pop-up window.



- On successfully generating a VHDL file, the following window pops-up:

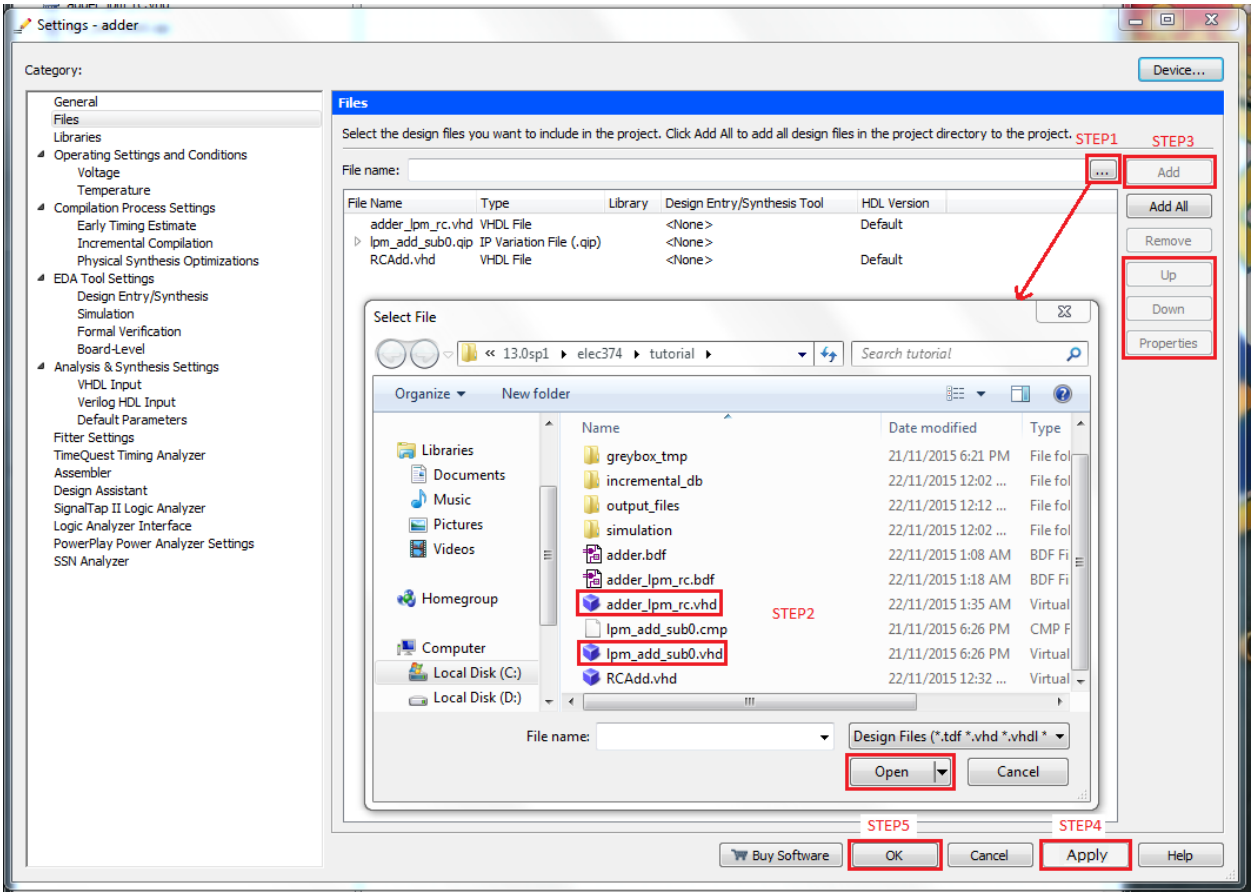


- You can confirm the generation of the HDL file by looking for "**adder_lpm_rc.vhd**" directly in the directory which you created for the project.
- To view the contents of the HDL file, navigate from the **Project** drop-down menu,

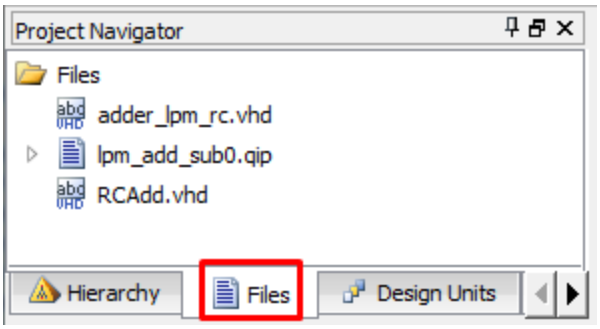


click on **"Add/Remove Files in Project"** to bring up the settings window, and follow Step 1 through 5 from the screenshot below:

- Step 1: Click on the browse button [**…**] to open the "**Select File**" dialogue box shown in Step 2.
- Step 2: Select the file that we generated from the block schematic: "**adder_lpm_rc.vhd**". You will also notice a system generated VHDL file: "**lpm_add_sub0.vhd**". This is the VHDL code of the LPM (Library of Parameterized Modules) module**: LPM_ADD_SUB.** Select this file too by clicking on CTRL+<Filename> and open both files by clicking on **OPEN**.
- Step 3: If the **Add** button is active then click on it and you will see both the files added in the **Files** menu. Use the **Remove** button on the right to remove all files, except the two files that you just added and the RCAdd.vhd file. Use the **Up/Down** buttons to move adder_lpm_rc.vhd above lpm_add_sub0.qip. RCAdd.vhd stays at the bottom of the three files.
- Step 4: Click on **Apply.**
- Step 5: Click on **OK.**



- Now you should be able to view the files in the **Project Navigator** window, under the **Files** menu:



- To simulate the design, we will need a testbench through which we will be providing stimulus for specific signals in the design. This testbench will be written in VHDL and has the following sections: Entity, Architecture, Component Instantiation, Port Mapping and Test Logic.

Creating A Simple VHDL Testbench

The following steps will illustrate how to create a simple VHDL testbench:

- A VHDL file is added to the design as described in Section 4.1; that is,
  - Click on **File → New**
  - Select *VHDL File* from the **Design Files** tab and click **OK** .
  - Click on **File → Save As** and save it as *<filename_tb.vhd> adder_lpm_rc_tb.vhd* in this example.
  - Enter the below code in *adder_lpm_rc_tb.vhd* as testbench code for the combined adders, and save the design.
- The following code describes a simple testbench for the combined adders.

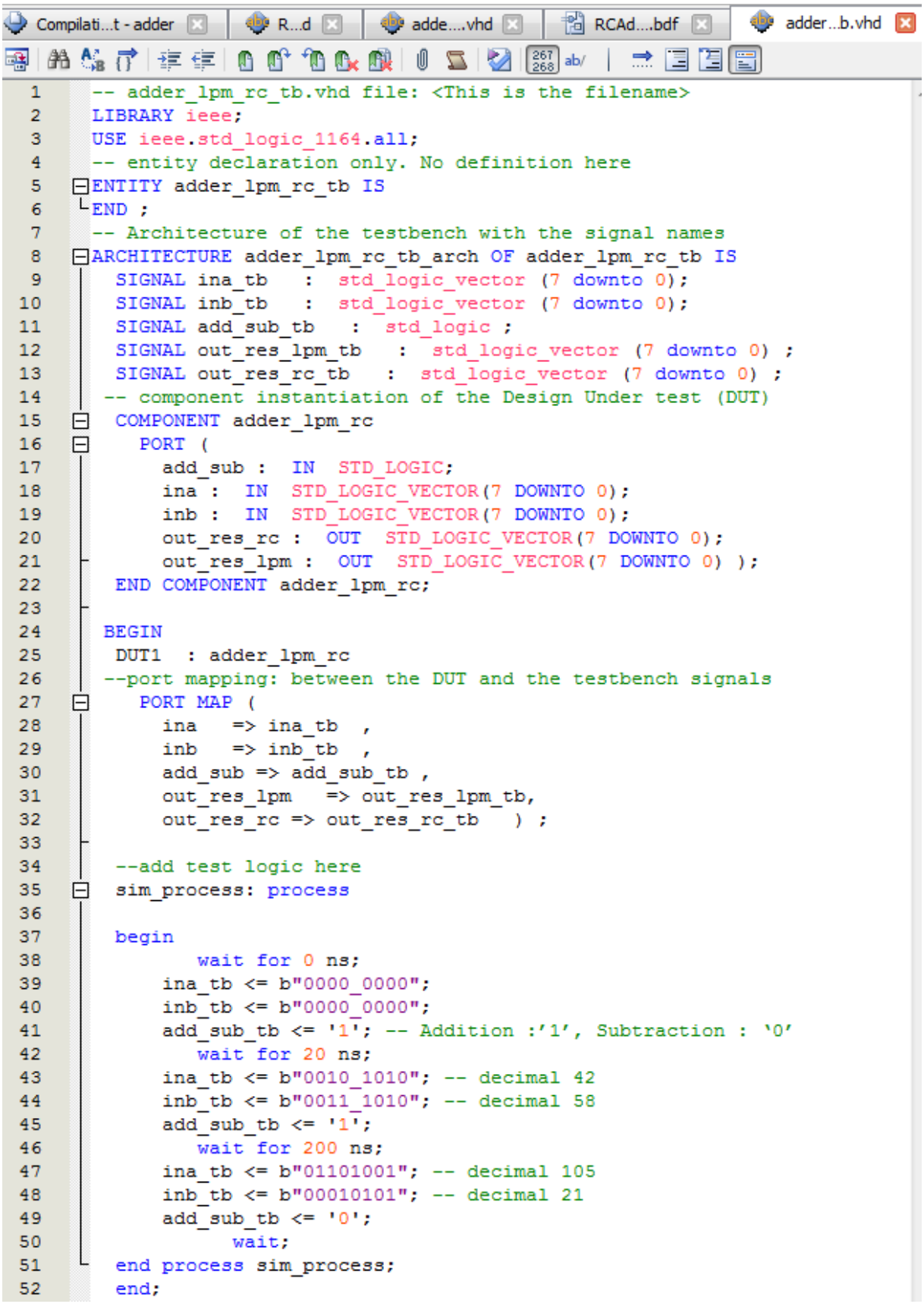
```
-- adder_lpm_rc_tb.vhd file: <This is the filename>
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-- entity declaration only. No definition here
ENTITY adder_lpm_rc_tb IS
END ;
-- Architecture of the testbench with the signal names
ARCHITECTURE adder_lpm_rc_tb_arch OF adder_lpm_rc_tb IS
  SIGNAL ina_tb    :  std_logic_vector (7 downto 0);
  SIGNAL inb_tb    :  std_logic_vector (7 downto 0);
  SIGNAL add_sub_tb    :  std_logic ;
  SIGNAL out_res_lpm_tb    :  std_logic_vector (7 downto 0) ;
  SIGNAL out_res_rc_tb    :  std_logic_vector (7 downto 0) ;
 -- component instantiation of the Design Under test (DUT)
  COMPONENT adder_lpm_rc
    PORT (
     add_sub :  IN  STD_LOGIC;
     ina :  IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
     inb :  IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
     out_res_rc :  OUT  STD_LOGIC_VECTOR(7 DOWNTO 0);
     out_res_lpm :  OUT  STD_LOGIC_VECTOR(7 DOWNTO 0) );
  END COMPONENT adder_lpm_rc;

 BEGIN
```

```
 DUT1   : adder_lpm_rc
--port mapping: between the DUT and the testbench signals
   PORT MAP (
     ina    => ina_tb   ,
     inb    => inb_tb   ,
     add_sub => add_sub_tb ,
     out_res_lpm    => out_res_lpm_tb,
     out_res_rc => out_res_rc_tb ) ;

 --add test logic here
 sim_process: process

 begin
        wait for 0 ns;
     ina_tb <= b"0000_0000";
     inb_tb <= b"0000_0000";
    add_sub_tb <= '1'; -- Addition :'1', Subtraction : '0'
        wait for 20 ns;
     ina_tb <= b"0010_1010"; -- decimal 42
     inb_tb <= b"0011_1010"; -- decimal 58
    add_sub_tb <= '1';
        wait for 200 ns;
     ina_tb <= b"01101001"; -- decimal 105
     inb_tb <= b"00010101"; -- decimal 21
    add_sub_tb <= '0';
           wait;
 end process sim_process;
 end;
```

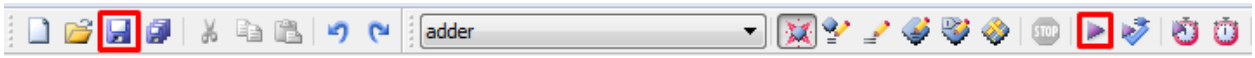
Note: Port names in the design file of the DUT (ina, inb, out_res_rc etc.) will be mapped to the signal names (ina_tb, inb_tb, out_res_rc_tb etc.) of the testbench file.

- The program is made up of the following distinct blocks (for convenience, refer to the screenshot in the next page):

Line Number(s)	Function
2-3	Setup libraries (needed for standard logic signals)
5	Entity declaration
8-22	Architecture declaration and Component Instantiation (`adder_lpm_rc VHDL module`)
27-32	Port mapping between the TB signals and the DUT
35-51	Test logic to verify functionality of the generated VHDL output file

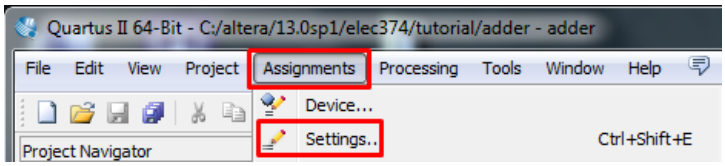

Compilati...t - adder
R...d
adde....vhd
RCAd....bdf
adder...b.vhd
-- adder_lpm_rc_tb.vhd file: <This is the filename>
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-- entity declaration only. No definition here
ENTITY adder_lpm_rc_tb IS
END ;
-- Architecture of the testbench with the signal names
ARCHITECTURE adder_lpm_rc_tb_arch OF adder_lpm_rc_tb IS
SIGNAL ina_tb : std_logic_vector (7 downto 0);
SIGNAL inb_tb : std_logic_vector (7 downto 0);
SIGNAL add_sub_tb : std_logic ;
SIGNAL out_res_lpm_tb : std_logic_vector (7 downto 0) ;
SIGNAL out_res_rc_tb : std_logic_vector (7 downto 0) ;
-- component instantiation of the Design Under test (DUT)
COMPONENT adder_lpm_rc
PORT (
add_sub : IN STD_LOGIC;
ina : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
inb : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
out_res_rc : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
out_res_lpm : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END COMPONENT adder_lpm_rc;
BEGIN
DUT1 : adder_lpm_rc
--port mapping: between the DUT and the testbench signals
PORT MAP (
ina => ina_tb ,
inb => inb_tb ,
add_sub => add_sub_tb ,
out_res_lpm => out_res_lpm_tb,
out_res_rc => out_res_rc_tb) ;
--add test logic here
sim_process: process
begin
wait for 0 ns;
ina_tb <= b"0000_0000";
inb_tb <= b"0000_0000";
add_sub_tb <= '1'; -- Addition :'1', Subtraction : '0'
wait for 20 ns;
ina_tb <= b"0010_1010"; -- decimal 42
inb_tb <= b"0011_1010"; -- decimal 58
add_sub_tb <= '1';
wait for 200 ns;
ina_tb <= b"01101001"; -- decimal 105
inb_tb <= b"00010101"; -- decimal 21
add_sub_tb <= '0';
wait;
end process sim_process;
end;

- Now set **adder_lpm_rc.vhd** as the top level entity by right clicking on adder_lpm_rc.vhd in the **Files** tab, and selecting *Set as Top-Level Entity*
- Save and Compile the design as before, by clicking on

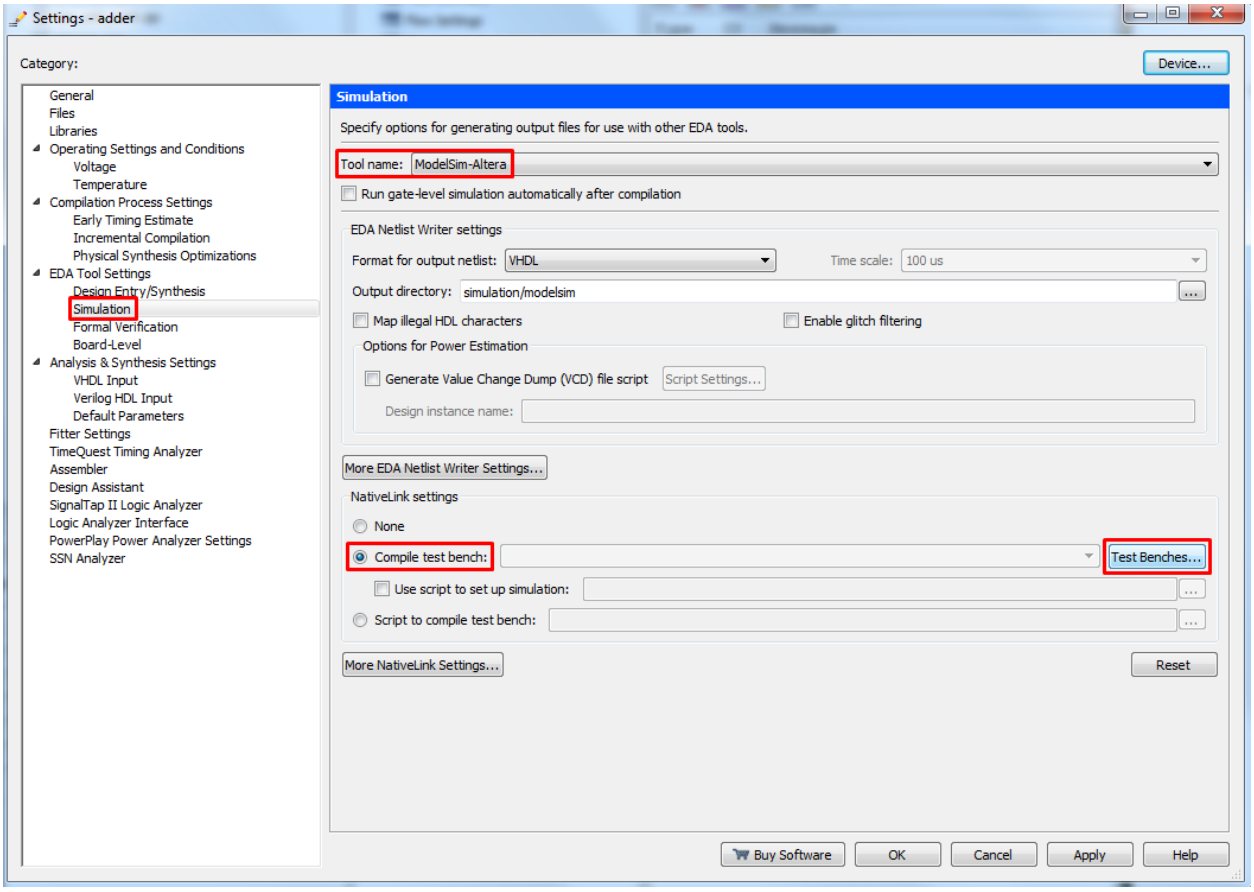


Setting up the TestBench for Simulation

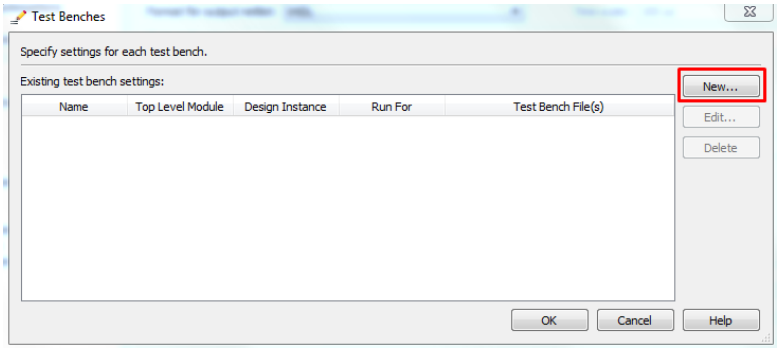
- After compilation goes through without errors, we now have to setup the testbench file to be identified by the simulation tool (ModelSim) to be used for providing stimulus for the DUT. Here are the steps to setup the Testbench file for simulation:
  - Click on the "**Assignments**" menu and go to **Settings.**



  - In the **Category** list, select **Simulation,** under **EDA Tool Settings**. The **Simulation** page appears.

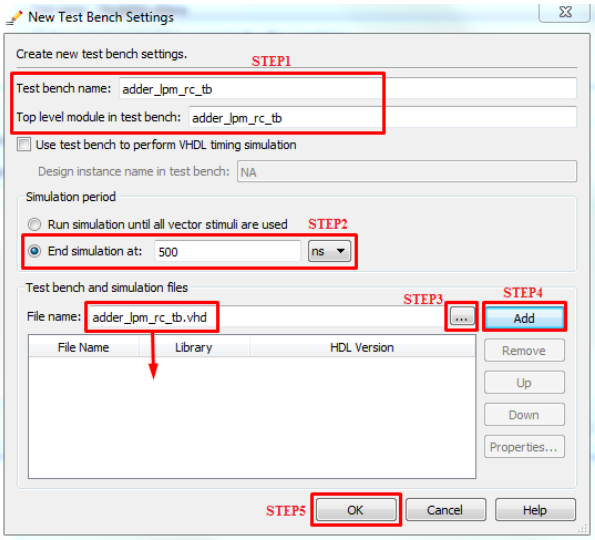


- In the **Tool name** list, select **ModelSim-Altera**
- Click on the **Compile test bench** option to choose it. This option helps to compile simulation models, design files, testbench files, and starts simulation. You can use different testbench setups to specify different test scenarios.
- Click on **Test Benches**. The **Test Benches** dialog box pops-up.
- Click **New**. The **New Test Bench Settings** dialog box appears

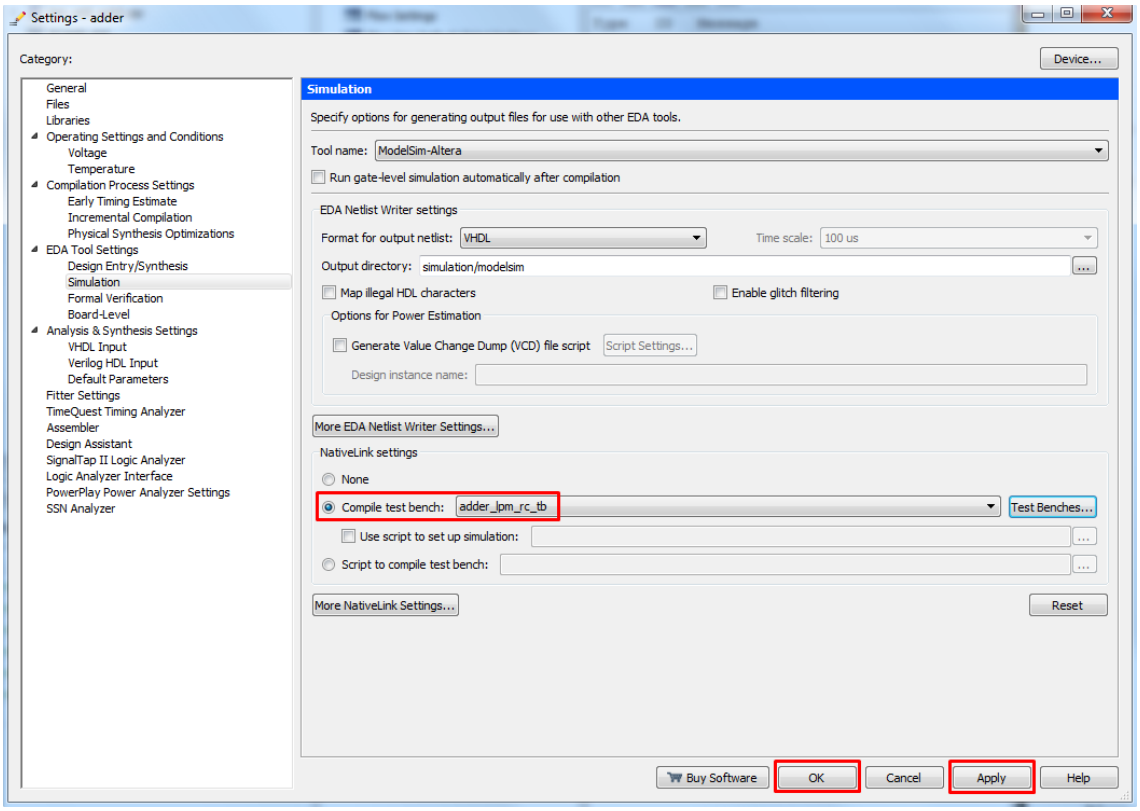


- In the **Top level module in test bench** box, type the top-level testbench entity or module name <*vhdl_testbench_name_tb*>. In this case, adder_lpm_rc_tb (refer to the screenshot in the next page).
- Under **Simulation period**, select **End simulation at** and specify the time as 500ns.

  Note: If you try to make your simulation too long (say, over 1 ms) Quartus II is likely to crash.
- Under **Test bench files**, browse and add the testbench files in the **File name** box. Use the **Up** and **Down** buttons to reorder the files when there are more than one testbench files. The files will be compiled in order from top to bottom.
- Click **OK**.
- In the **Test benches** dialog box, click **OK**.
- This setup is called: enabling the NativeLink feature in Quartus II



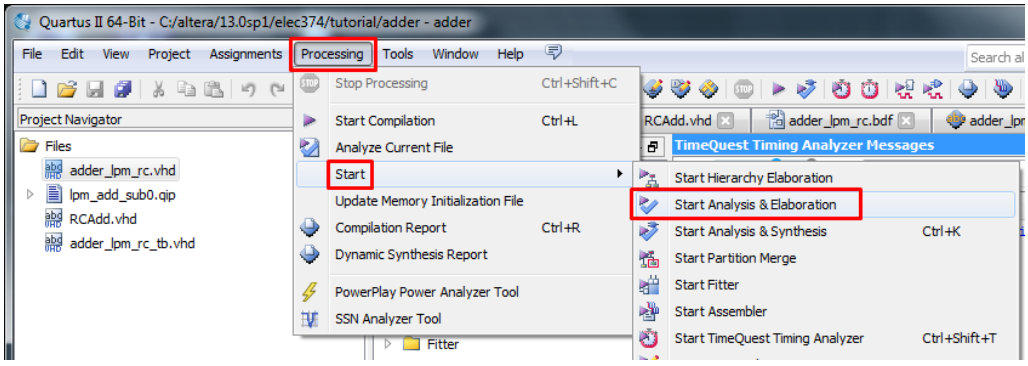
- You will now be back to the **Settings Menu,** where you will find the testbench name you just added next to the **Compile Test bench** option.
- Click on **Apply** and then **OK,** and you will be done with setting up the testbench file for simulation.



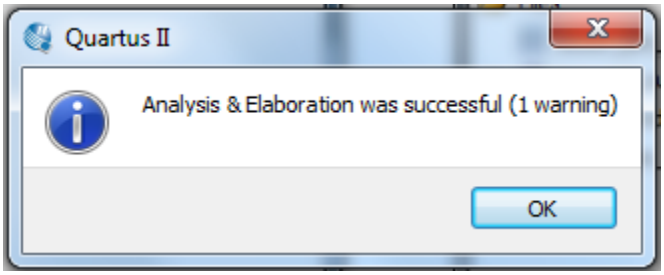
- Now we will see the steps to run the functional simulation using ModelSim.

Analyzing and Elaborating the Design

- On the **Processing** menu, point to **Start** and click on **Start Analysis & Elaboration**. This command collects all your file name information and builds your design hierarchy in preparation for simulation.



- On successful completion of this step, you will see the following dialogue box:



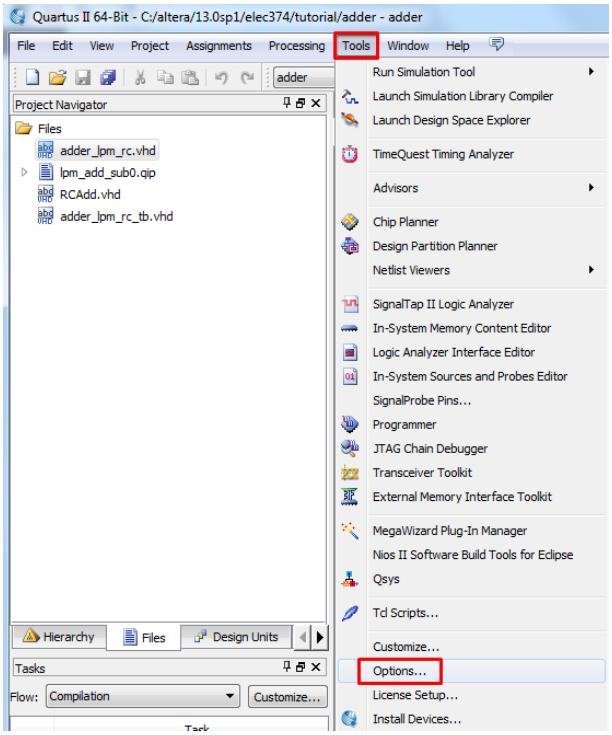
- This one warning may be ignored, as we are only trying to verify the functionality of the design at this point. The warning may have to be fixed when we try to implement the design in hardware (this is called Design Synthesis)
- To run the simulation, we will need to install ModelSim.

8. FUNCTIONAL SIMULATION WITH MODELSIM

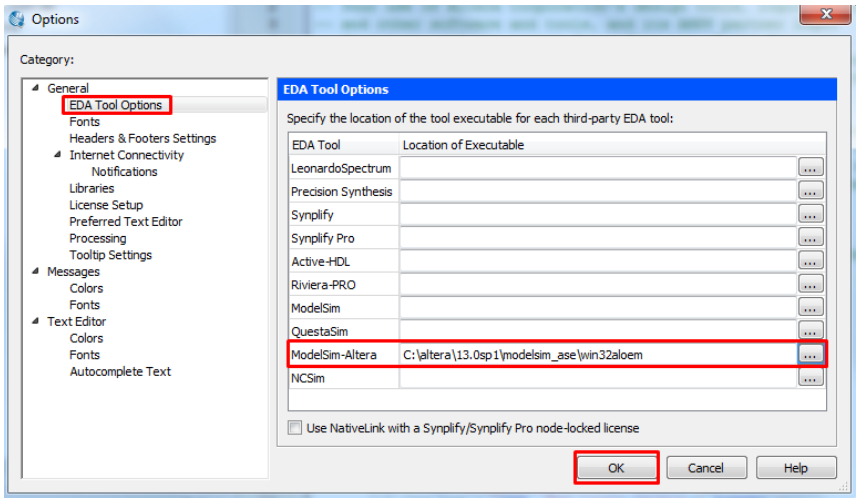
The following steps will guide you through the use of ModelSim for simulation of the design that we have created. If you have correctly followed the steps in Section 1, then ModelSim should be installed in your machine and you should be able to find it in the windows startup menu under "**All Programs → Altera 13.0.1.232 Web Edition → ModelSim-Altera 10.1d (Quartus II 13.0sp1)**".

Setting up the EDA Simulator Execution Path

- ModelSim can be set as the default EDA (Electronic Design Automation) simulation tool to work with Quartus II. In Section 3.1, we had chosen ModelSim-Altera as our choice of simulation tool and set VHDL as the desired format.
- We can now simulate the testbench using ModelSim with the following steps:
  - On the **Tools** menu in Quartus, click **Options** as shown in the snapshot.
  - The **Options** dialog box appears



- In the **Category** list, select **EDA Tool Options**.

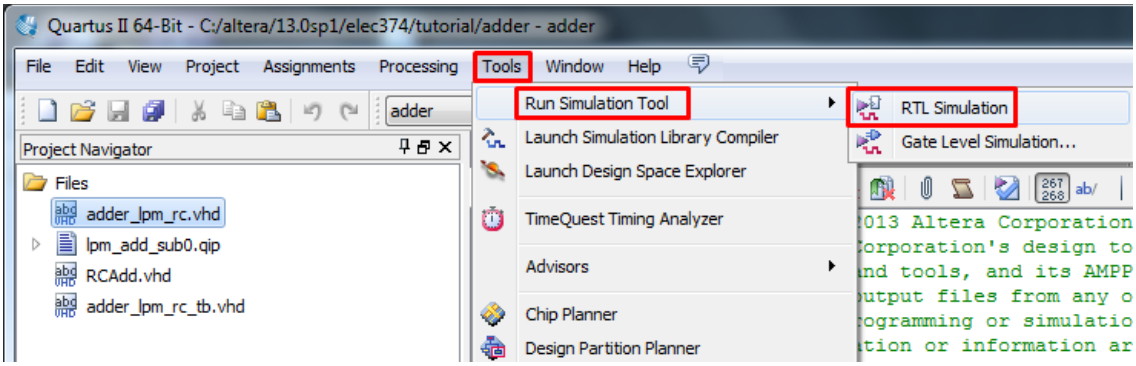


- Click on the button next to the **Location of Executable** entry corresponding to the **ModelSim-Altera** option.
- Type the path or browse to the directory containing the executables of the EDA tool. (The default location is as shown in the screen-shot: **C:\altera\13.0sp1\modelsim_ase\win32aloem**). Click **OK.**

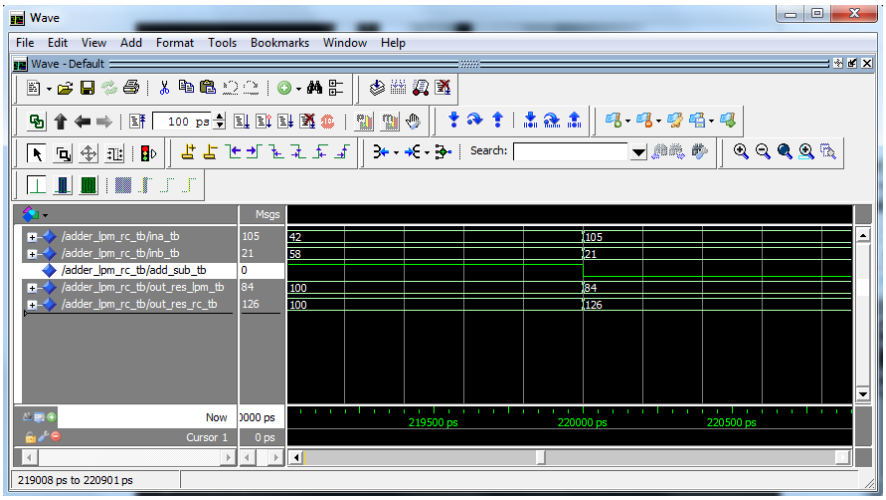
Launching the ModelSim Simulation

- Ensure the required VHDL design file is set as **top-level entity** (Section 5.2)**.**

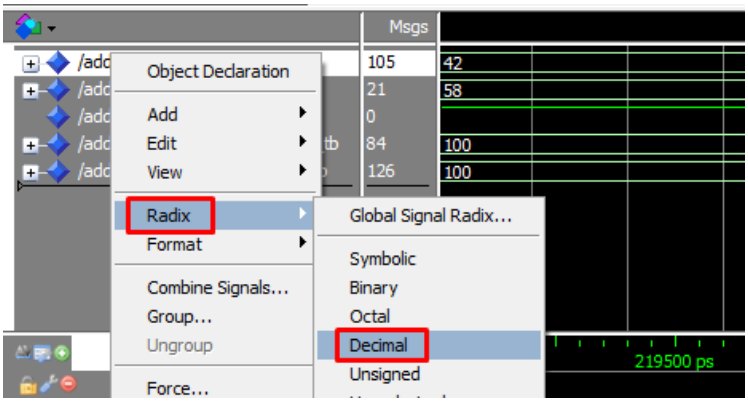
- To launch the simulation (after ensuring analysis and elaboration has gone through without errors), on the **Tools** menu, point to **Run Simulation Tool** and click **RTL Simulation** to automatically run the EDA simulator, compile all necessary design files, and complete a simulation.



5. Now the ModelSim window pops-up and automatically loads all your design files and runs the simulation for you. You should be able to see the simulation results in the window that pops-up.
6. You will notice that the output: **out_res_lpm_tb** is the result of the LPM_ADD_SUB module, which is capable of addition and subtraction. However, the RCAdder is only an adder and its output **out_res_rc_tb** is not affected by the change in signal level of **add_sub_tb** signal.
7. This completes the testing of the Adder module you created.



8. The radix of the results can be changed to decimal by right-clicking on the binary numbers and using the following settings, to make the results easily readable.



Note: Ensure that the ModelSim window is closed every time you launch a simulation from Quartus. Otherwise, there is a conflict with the existing open window and ModelSim will not launch.