2-Bit Ripple Adder

Mentors
Prasad T
Aditya Gudla
Simranjeet Singh

Interns
Ajay Chaudhari
Chethan T Bhat
Ritvik Tiwari
Karthik A Shet

Introduction

Full Adder is the adder which adds three inputs and produces two outputs. The first two inputs are A and B and the third input is an input carry as C-IN. The output carry is designated as C-OUT and the normal output is designated as S which is SUM.

Multiple full adder circuits can be cascaded in parallel to add an N-bit number. For an N-bit parallel adder, there must be N number of full adder circuits. A ripple carry adder is a logic circuit in which the carry-out of each full adder is the carry in of the succeeding next most significant full adder. It is called a ripple carry adder because each carry bit gets rippled into the next stage. We obtain an output carry (C-OUT) and an N- bit sum (S).

For this project we will be using the full adder design for our 2-bit ripple carry adder

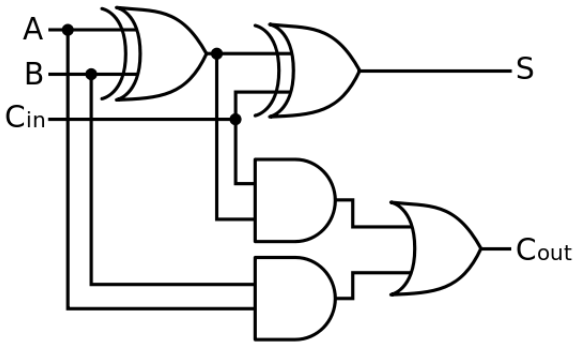

Figure 1: Full Adder

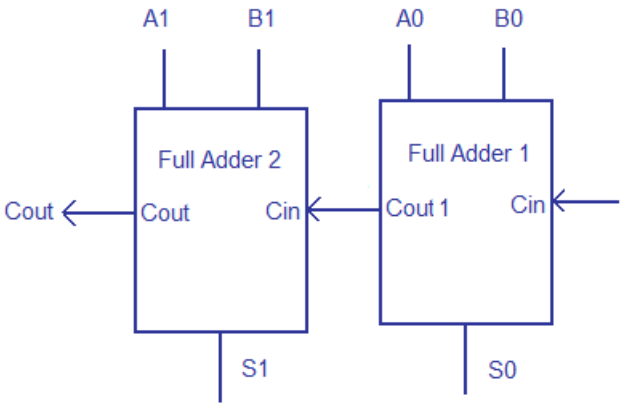

Figure 2: 2-bit Ripple Carry Adder

Truth Table for Full Adder				
A	B	Cin	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 1: Truth Table for Full Adder

VHDL Codes

Code for Full Adder Dataflow Modelling:

```
--Full Adder VHDL Code
--Including Libraries
LIBRARY IEEE;
USE     IEEE.STD_LOGIC_1164.ALL;
USE     IEEE.NUMERIC_STD.ALL;

--Full Adder Entity Declaration
ENTITY full_adder IS
        PORT(
                A       : IN    STD_LOGIC;  --Input A
                B       : IN    STD_LOGIC;  --Input B
                CIN     : IN    STD_LOGIC;  --Input CIN
                S       : OUT STD_LOGIC;    --Output SUM
                COUT: OUT STD_LOGIC     --Output CARRY
                );
        END  full_adder;

--Architecture Declaration
ARCHITECTURE BEHAVIOUR OF full_adder IS
        --Main Logic
        BEGIN

                S   <=  A XOR B XOR CIN;
                COUT<= (A AND B) OR (CIN AND (A XOR B));

END ARCHITECTURE BEHAVIOUR;
```

Code for 2 Bit Ripple Carry Full Adder

```
--We have incorporated the full adder entity from the previous
--code for the 2 bit ripple carry full adder

LIBRARY          IEEE;
USE              IEEE.STD_LOGIC_1164.ALL;
USE              IEEE.NUMERIC_STD.ALL;

--Ripple Carry Adder Entity
ENTITY ripple_carry_adder IS
  PORT(
          A: IN STD_LOGIC_VECTOR (1 DOWNTO 0); --2 bit Vector input A
          B: IN STD_LOGIC_VECTOR (1 DOWNTO 0); --2 bit Vector input B
          CIN: IN STD_LOGIC;
          S: OUT STD_LOGIC_VECTOR (1 DOWNTO 0);--2 bit Vector output SUM
          COUT: OUT STD_LOGIC
          );

END ENTITY ripple_carry_adder;

 --Architecture Declaration
ARCHITECTURE BEHAVIOURAL OF ripple_carry_adder IS
        COMPONENT full_adder
                PORT(
                        A       : IN    STD_LOGIC;
                        B       : IN    STD_LOGIC;
                        CIN     : IN    STD_LOGIC;
                        S       : OUT STD_LOGIC;
                        COUT : OUT STD_LOGIC
                        );
                                  END COMPONENT;

                SIGNAL C1       : STD_LOGIC;
                --Main Architecture
                --We have used the previously designed full_adder modules here
                BEGIN
                        --Over here we have used PORT MAP for mapping the current
                        --inputs with the full_adder module
                        FA0: full_adder PORT MAP( A(0),B(0),CIN,S(0), C1 );
                        FA1: full_adder PORT MAP( A(1),B(1), C1,S(1), COUT);

END ARCHITECTURE BEHAVIOURAL;
```

TestBench

Test Bench for Ripple Carry Adder

```
--VHDL Test Bench

--Including Libraries
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

--Entity Declaration
 ENTITY tb_ripple_carry_adder IS
END tb_ripple_carry_adder;


--Architecture Declaration
ARCHITECTURE behavior OF tb_ripple_carry_adder IS

-- Component Declaration for the Unit Under Test (UUT)

COMPONENT ripple_carry_adder
PORT(
A : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
B : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
CIN : IN STD_LOGIC;
S : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
COUT : OUT STD_LOGIC
);
END COMPONENT;

--Inputs
SIGNAL A : STD_LOGIC_VECTOR(1 DOWNTO 0) := (OTHERS => '0');
SIGNAL B : STD_LOGIC_VECTOR(1 DOWNTO 0) := (OTHERS => '0');
SIGNAL CIN : STD_LOGIC := '0';

--Outputs
SIGNAL S : STD_LOGIC_VECTOR(1 DOWNTO 0);
SIGNAL COUT : STD_LOGIC;

BEGIN

-- Instantiate the Unit Under Test (UUT)
uut: ripple_carry_adder PORT MAP (
A => A,
B => B,
CIN => CIN,
S => S,
COUT => COUT
);

-- Stimulus process
stim_proc: PROCESS
BEGIN
-- hold reset state for 100 ns.
WAIT FOR 100 ns;
```

```
--Different combinations of two bit inputs which are fed to the adder
A <= "01";
B <= "11";

WAIT FOR 100 ns;
A <= "11";
B <= "11";
CIN <= '1';

WAIT FOR 100 ns;
A <= "10";
B <= "01";

WAIT FOR 100 ns;
A <= "00";
B <= "11";


WAIT;

END PROCESS;

END;
```

Verilog Codes

You can perform the same experiment using **VERILOG** language. Please refer the codes given below.

RTL Description - Full Adder

```
// Verilog code for Full Adder
// Define Full Adder module
module full_adder(
input A,B,C_IN ,            // Define input ports A, B and C_IN
output S,C_OUT );           // Define output ports S and C_OUT

assign  S    = A ^ B ^ C_IN;                  // Define Sum logic
assign C_OUT = ((A & B) | (C_IN & (A ^ B))); // Define Carry_out logic

endmodule
```

RTL Description - Ripple Carry Adder

```
// Verilog code for 2 bit ripple_carry_adder
// Define module

module ripple_carry_adder(


input [1:0]A,B,
input C_IN,                 //Define all input ports
output [1:0]S,
output C_OUT);              // Define all ouput ports

wire C1;                   //Define intermediate carry as C1


full_adder FA0(A[0],B[0],C_IN,S[0],C1); // instantiate full_adder (FA0)
full_adder FA1(A[1],B[1],C1,S[1],C_OUT);// instantiate full_adder (FA1)


endmodule                           // end of module
```

Testbench

```
// Verilog code for Test bench
//Define module

module tb_ripple_carry_adder;

reg [1:0]A;                         //Define all I/O ports
reg [1:0]B;
reg C_IN;
wire [1:0]S;
wire C_OUT;

// Map all th I/O ports with DUT
ripple_carry_adder uut(.A(A) , .B(B), .C_IN(C_IN), .S(S), .C_OUT(C_OUT));

initial begin //Initialize the pins with different combination of inputs.

        A =2'b01; B=2'b11;              # 100;
        A =2'b11; B=2'b11; C_IN =1'b1; # 100;
        A =2'b10; B=2'b01;              # 100;
        A =2'b00; B=2'b11;              # 100;

        end                             // End of initial block
endmodule                           // End of module
```

Implementing on quartus II

Follow the below steps :

1. Start a **New Project** in Quartus Lite software

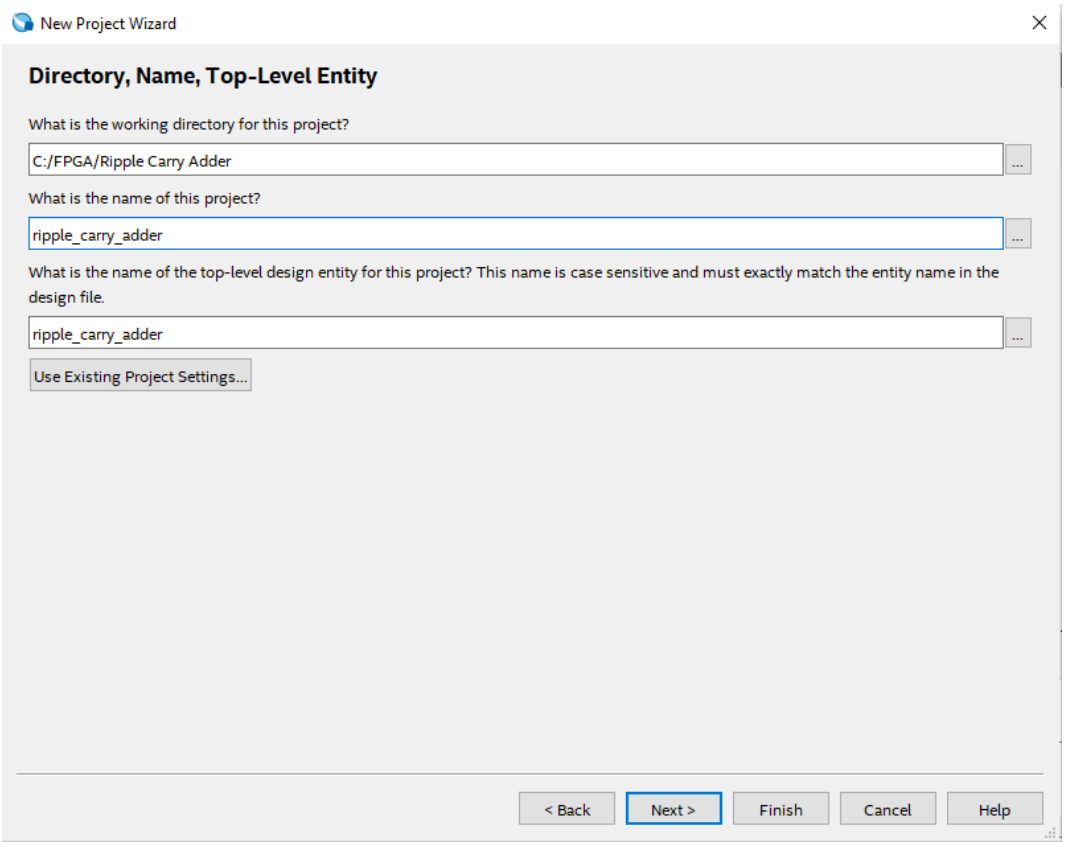

Figure 3: Creating New Project

2. You will see this screen after completing all steps. For detailed steps refer the **'Quick Start Guide to Quartus and ModelSim Software'** document.
Note: We will be using DE0 Nano Board (Cyclone IV Family EP4CE22F17C6) for entire project.

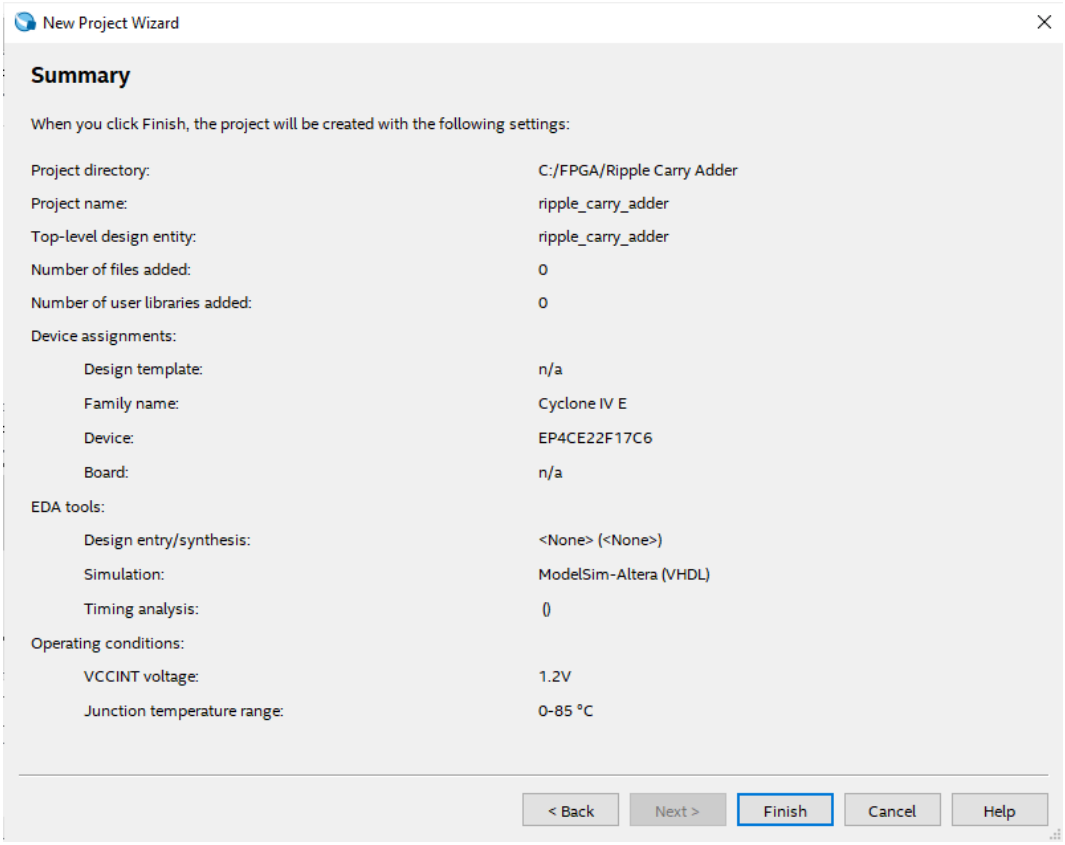

Figure 4: Summary

3. We will be using **VHDL** throughout this project. Create a **New VHDL file**.

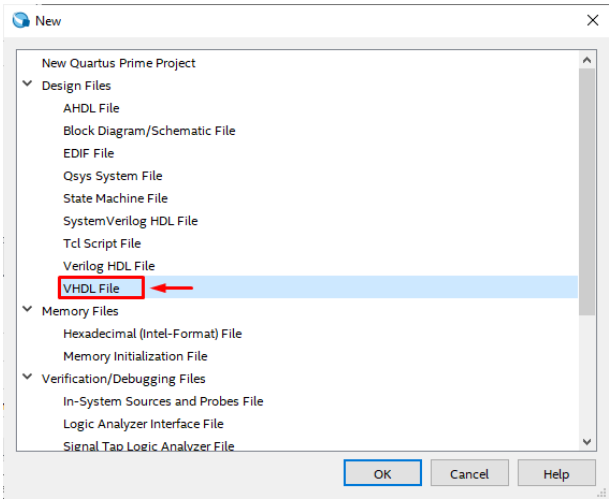

Figure 5: New VHDL File

4. Type the code for full adder(code on page 2) in this file.

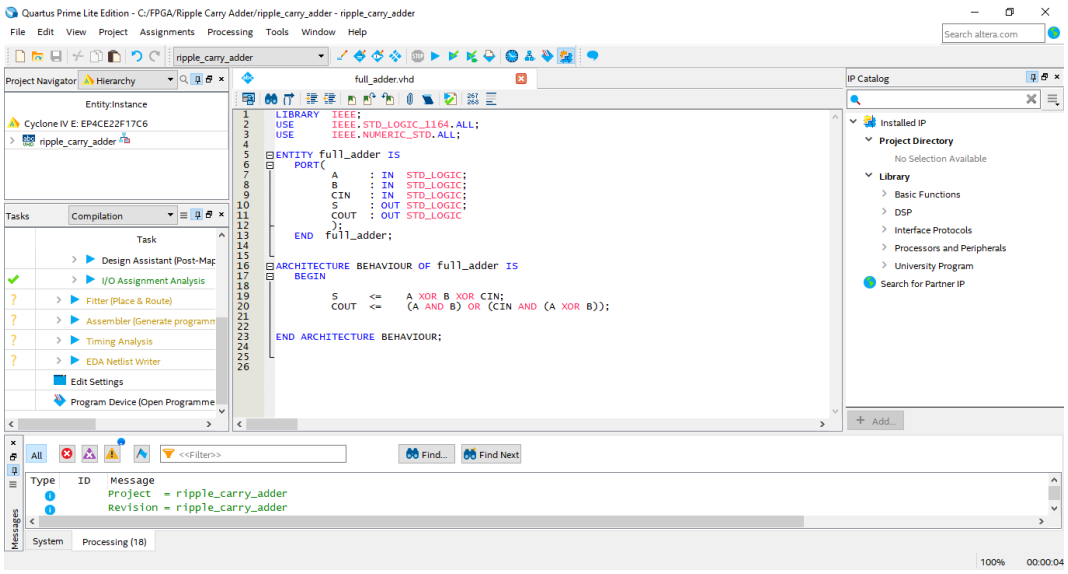

Figure 6: VHDL Code:Full Adder

5. Go to **File→Save as** and save the file.**Note:** File name should be same as entity name.

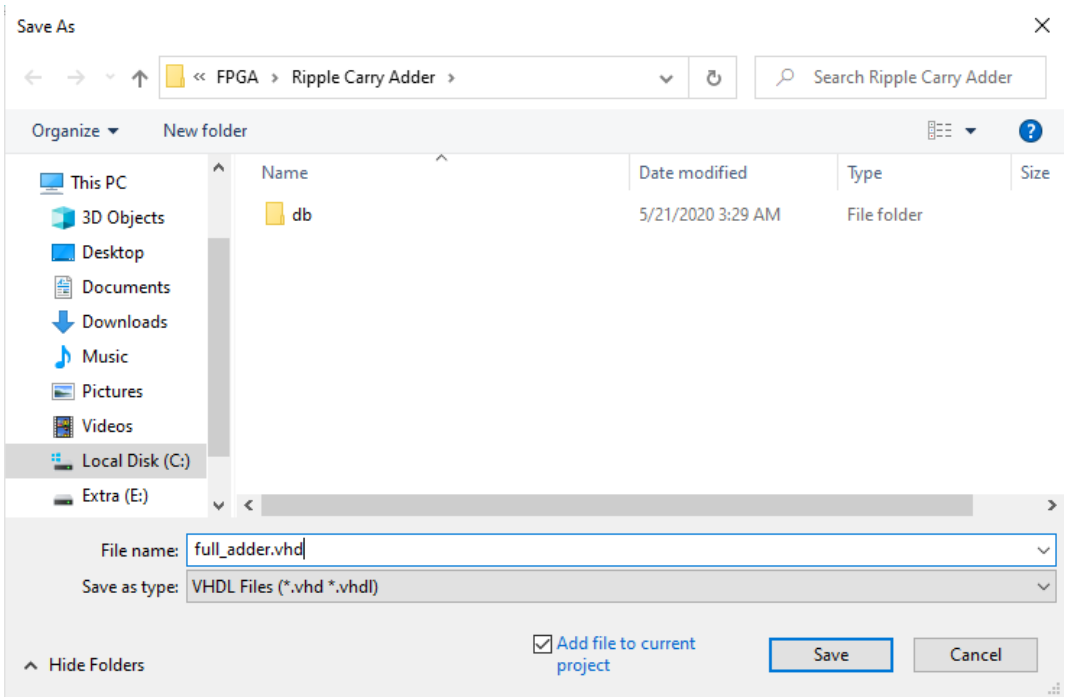

Figure 7: Saving the file

6. Now, Create a **New VHDL file** again and type the code for 2-bit Ripple Carry Adder(code on page 3) in this file.

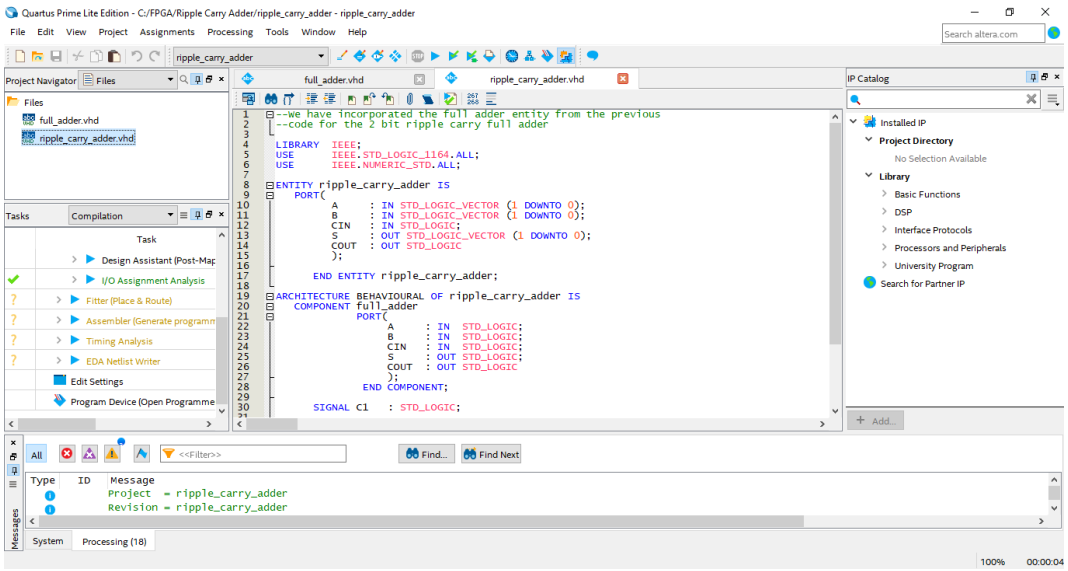

Figure 8: —VHDL Code: 2-bit Ripple Carry Adder

7. Go to **File→Save as** and save the file.

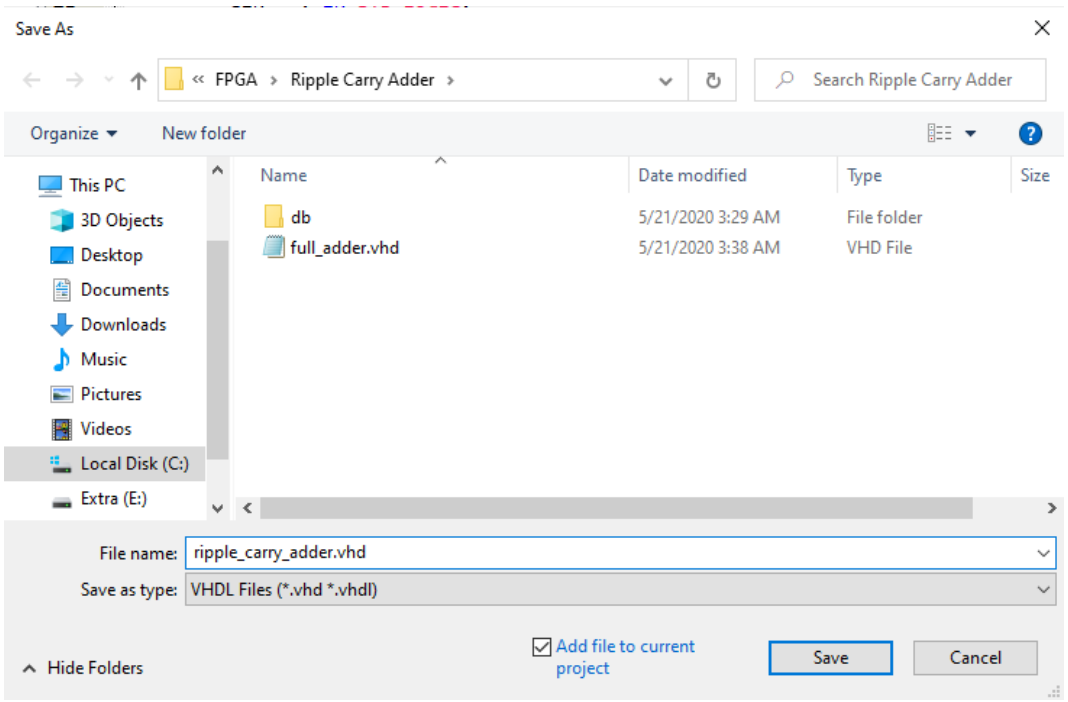

Figure 9: Saving the file

8. Goto **Project→Set as Top-Level Entity**. The **ripple_carry_adder** file is our main file and make sure you have selected this file while setting the top level entity.

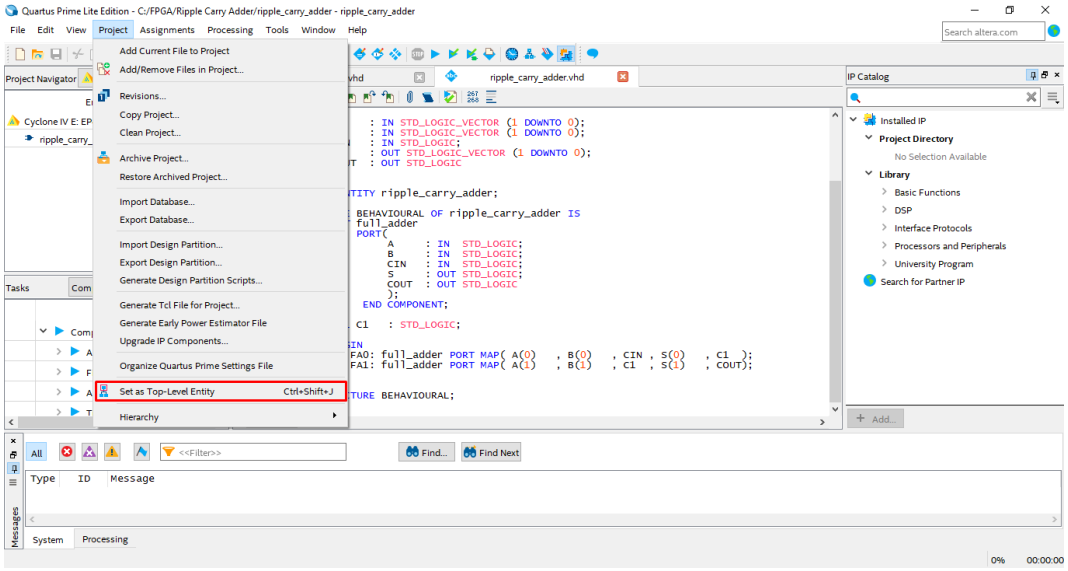

Figure 10: Setting Top level entity

9. Goto **Processing→Start Compilation**.

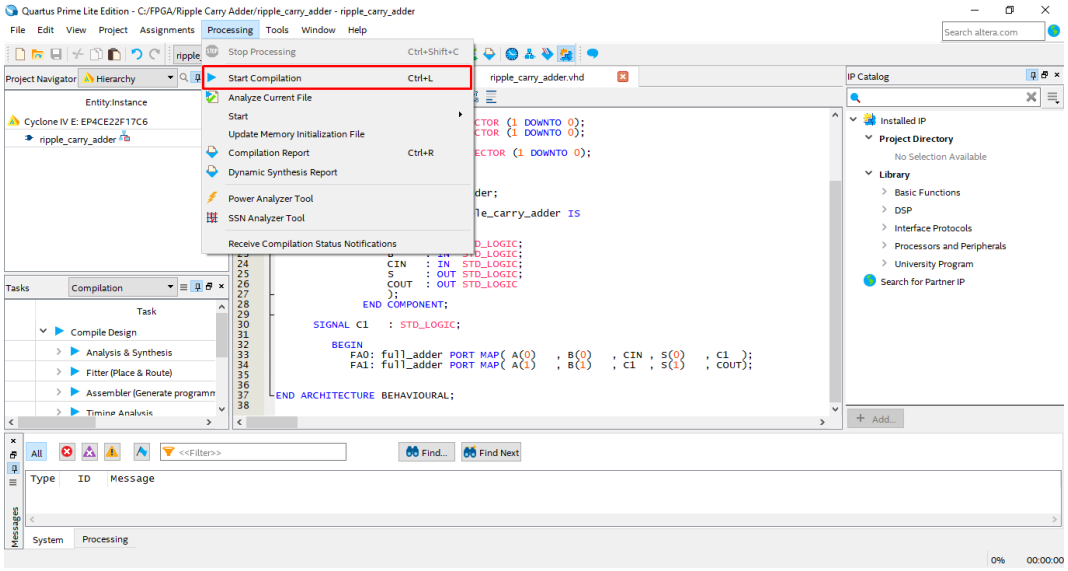

Figure 11: Compiling the Design

10. You can verify whether all the files are compiled successfully by checking the highlighted tabs i.e. **Messages** and **Tasks** tab.

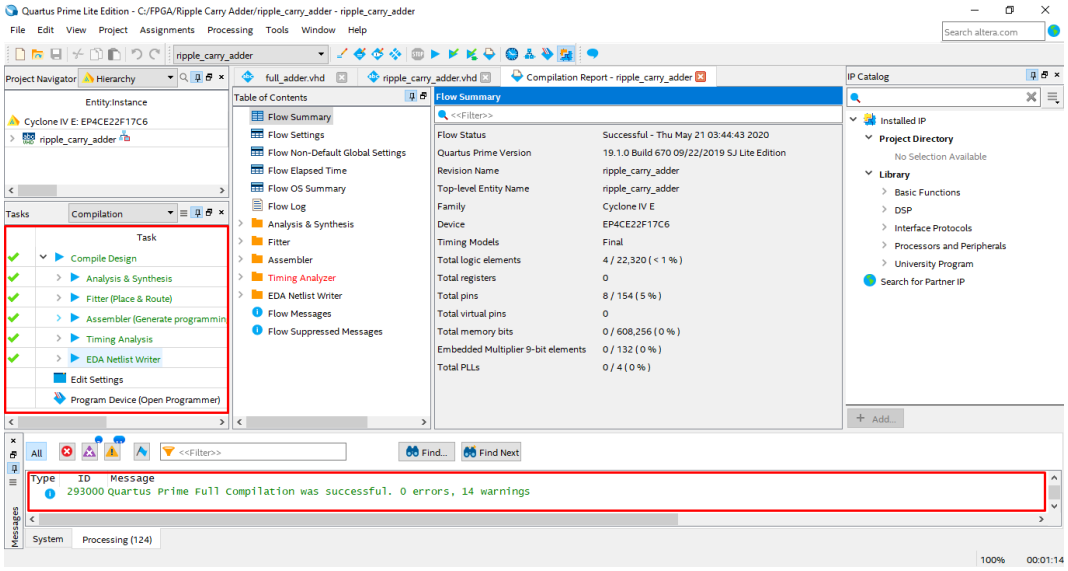

Figure 12: Verification(Flow Summary)

RTL Circuit of the implemented design

The Figure shown in step 2 below shows the RTL design of the ripple carry adder circuit. Here you can see 2 full adder modules are used. The design for these modules is incorporated from the full_adder file.

Steps to get RTL circuit.

1. Goto **Tools→Netlist Viewers→RTL Viewer**.

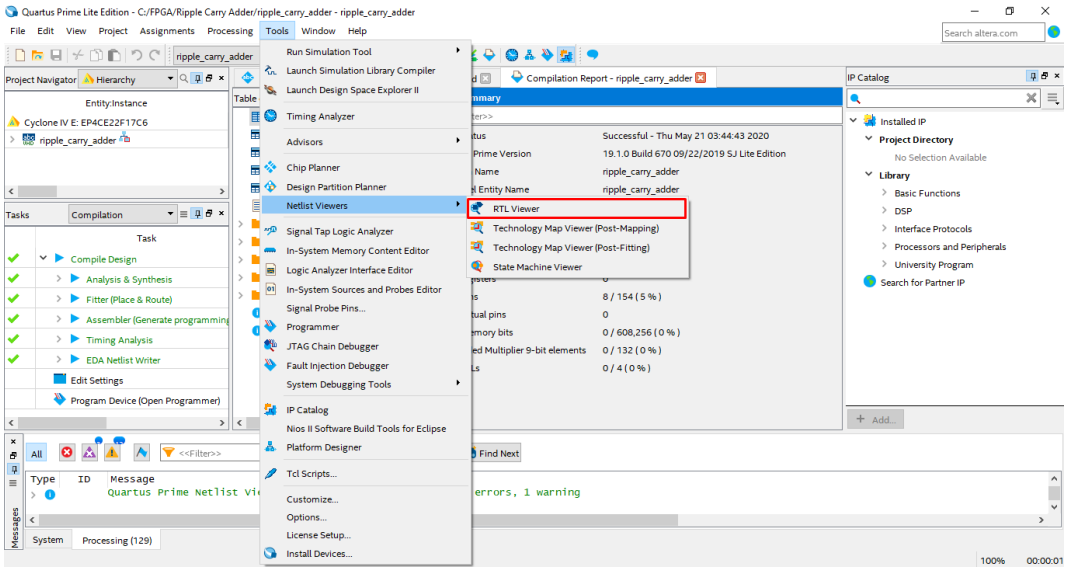

Figure 13: RTL Viewer

2. The below figure shows the equivalent RTL circuit of full adder.

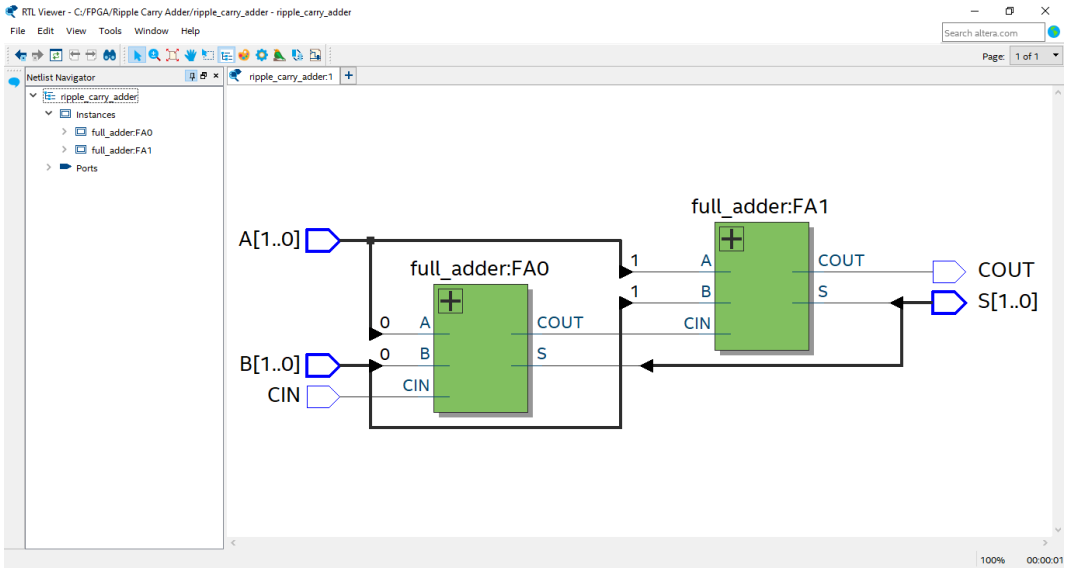

Figure 14: RTL Design for 2-bit Ripple Carry Adder

Pin Assignment

1. Click on **Assignments→Pin Planner**,This Pin Planner shows the I/O ports that we have created in our design.

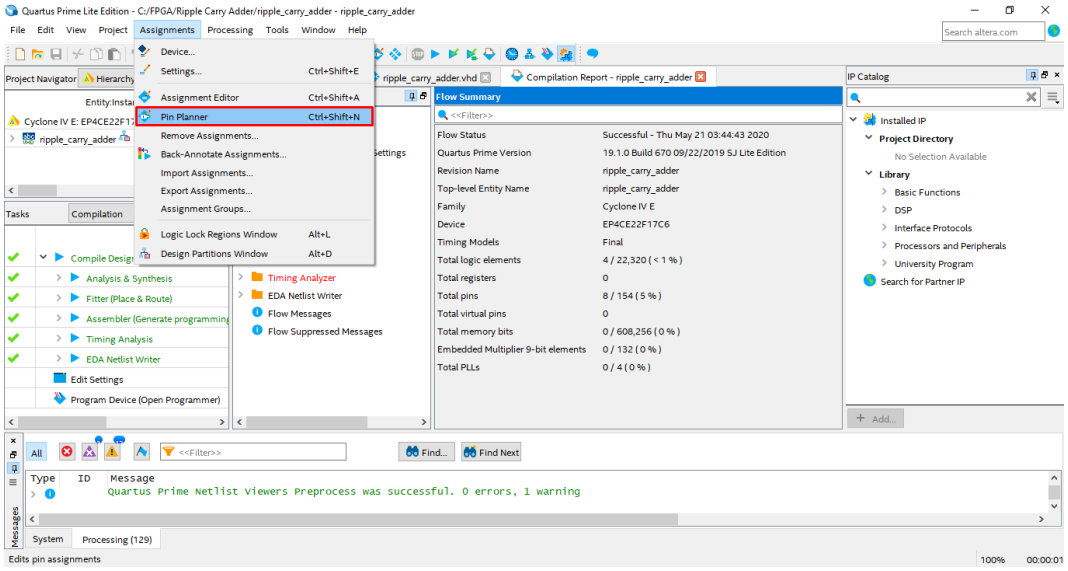

Figure 15: Pin Planner

2. We will now assign the pins for Input and Output in the Location tab present in the highlighted window.

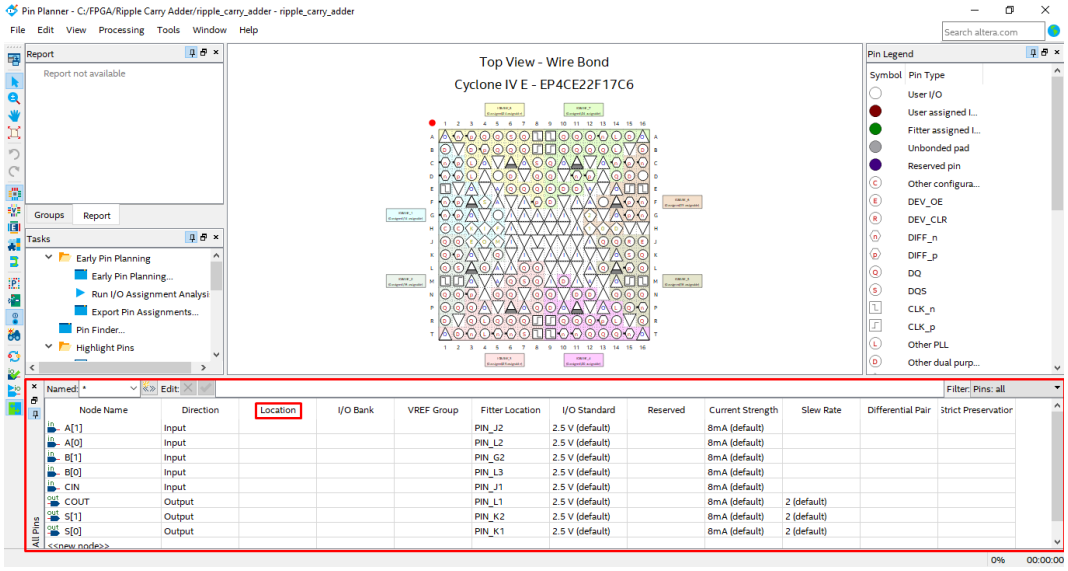

Figure 16: Assigning Pins

3. Now we have to assign each I/O ports with the respective Pin numbers, which can be found on the Device Manual which we are implementing.Here we are referring to the DEO NANO Board . We have demonstrated how to assign the DIP[0] switch to the first input(A[1]) in the following image. You can assign all the other **Inputs** and **Outputs** in similar manner.

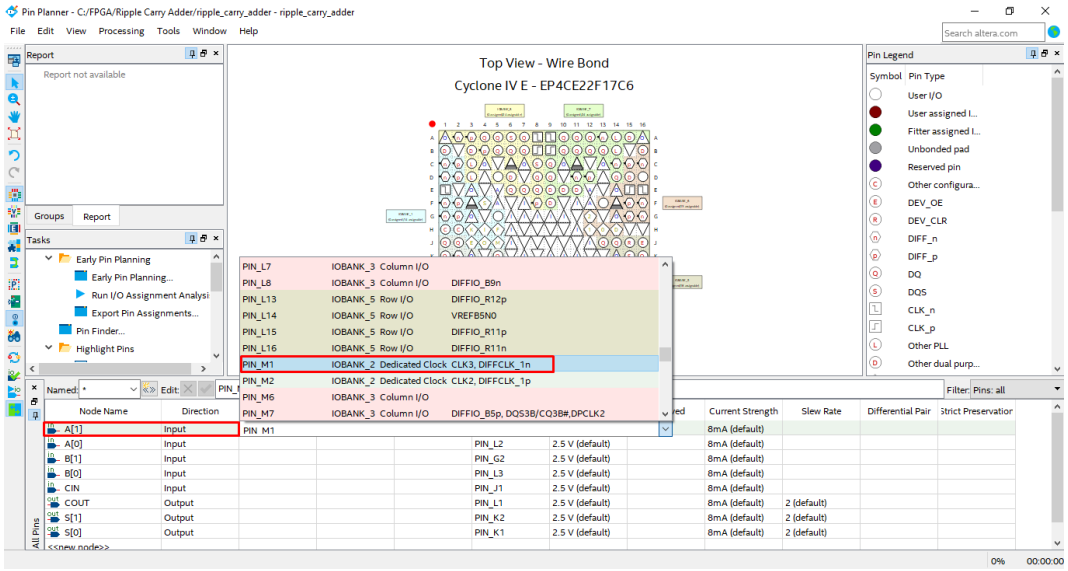

Figure 17: Assigning Pins

4. After assigning all the pins your window would be similar to the following image. You can verify the pin assignments from the following image.

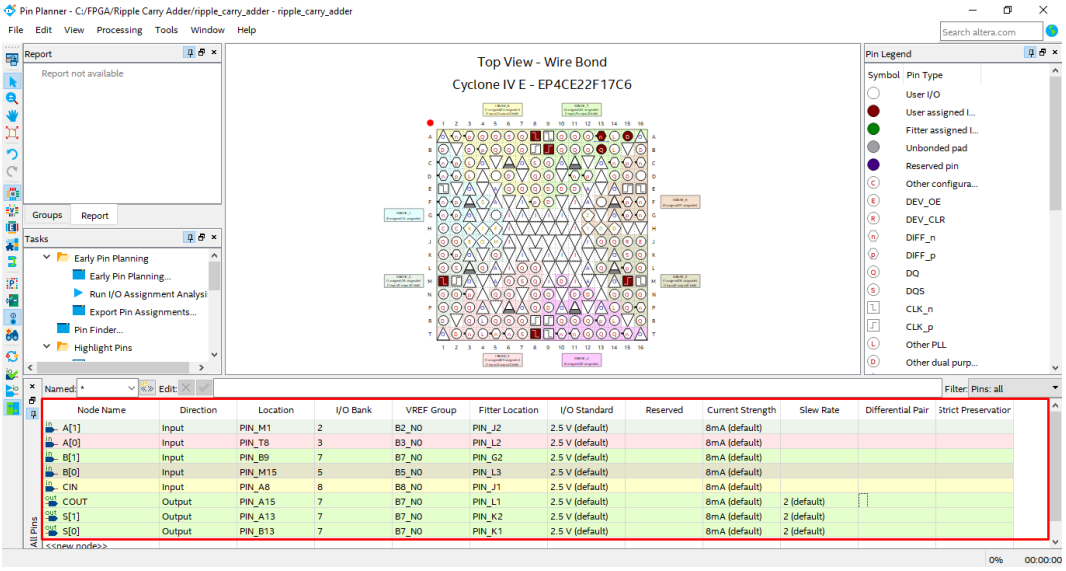

Figure 18: Verifying Pin Assignment

5. Go to **Processing→Start I/O Assignment Analysis**.

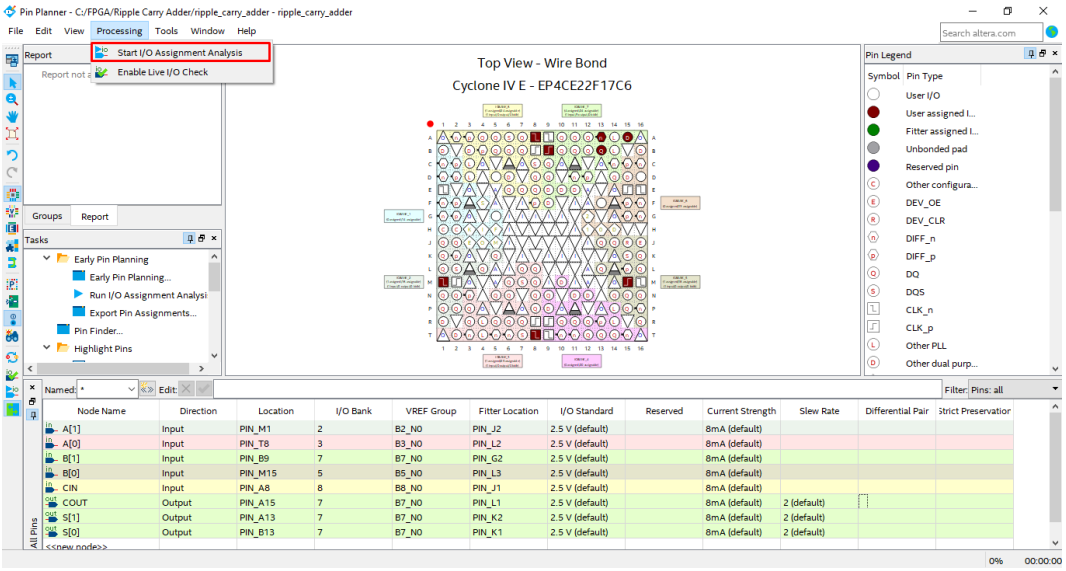

Figure 19: Start I/O Analysis

Downloading the code to DE0 Nano FPGA Board

Before starting, Make sure the board is powered ON and connected to the computer through an USB Cable

1. Compile the Project by double clicking on **Compile** or the **Play** button. This creates an SRAM object file(.SOF file). This file is used to program the Device

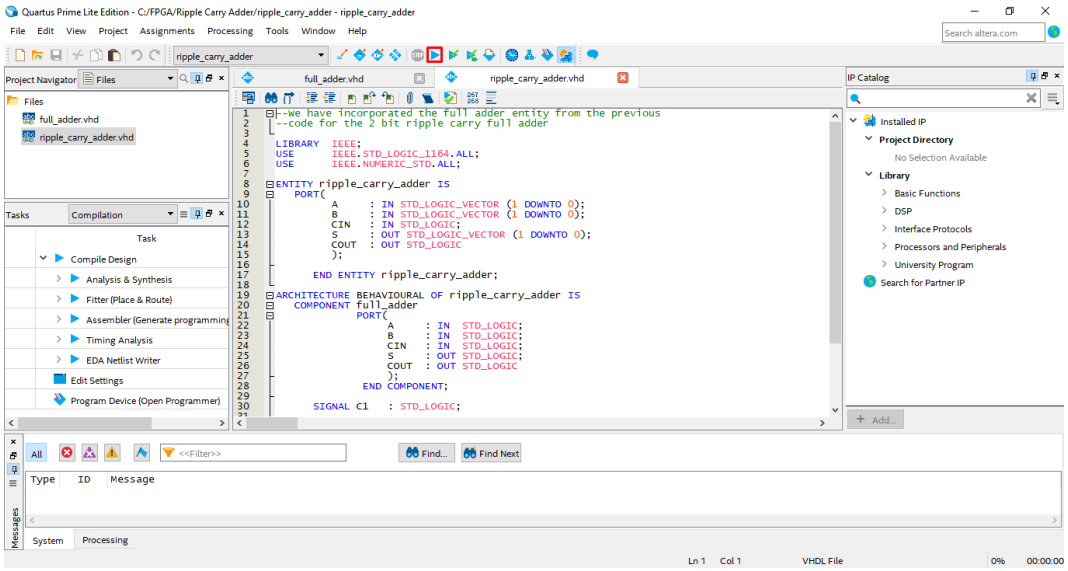

Figure 20: Creating .SOF file

2. Open the programmer by going to **Tools→Programmer**

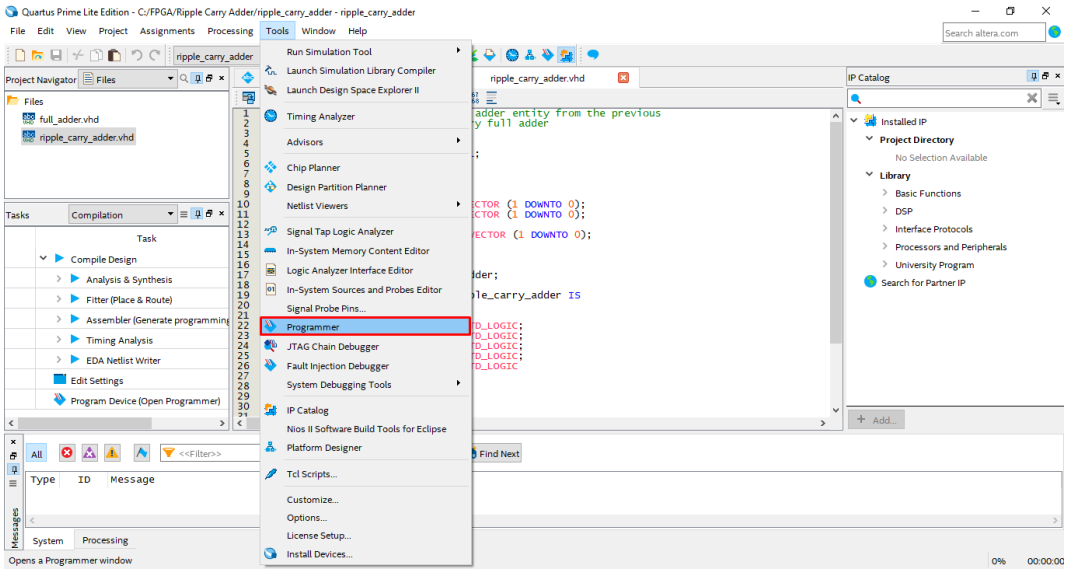

Figure 21: Programmer

3. Click on **Hardware Setup**. The Device required must be listed under "Currently available hardware". If not, check if the device drivers are correctly installed. Choose the hardware from the dropdown menu

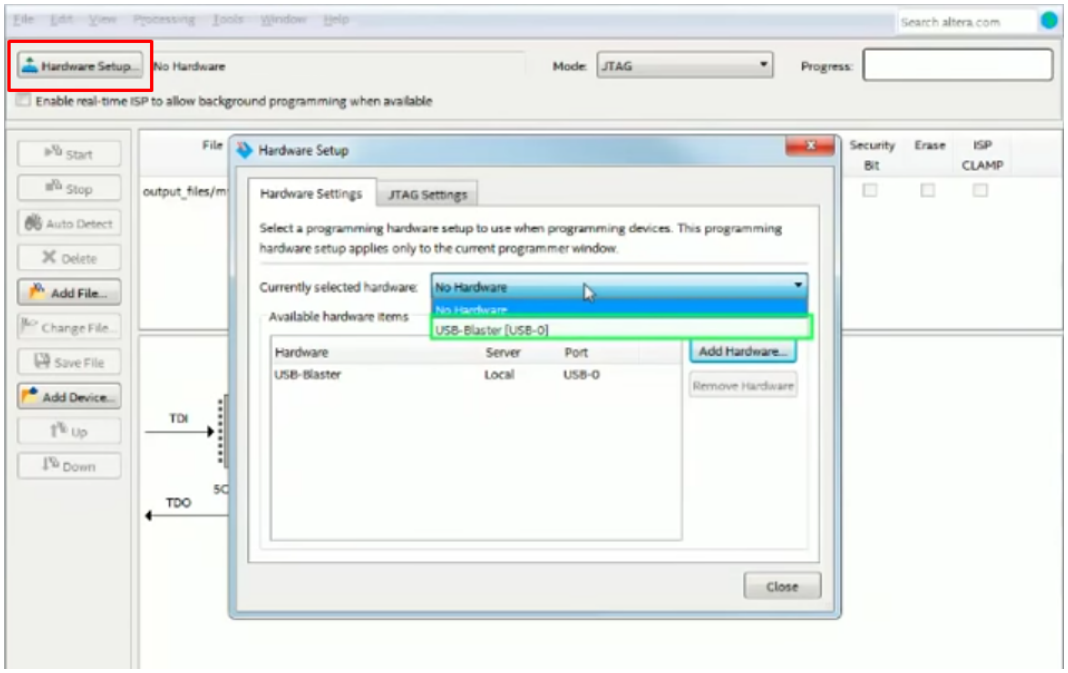

Figure 22: Hardware setup

4. If the file is not listed, it can be manually added by clicking on **Add File**. The .SOF file can be found the output directory inside the project directory

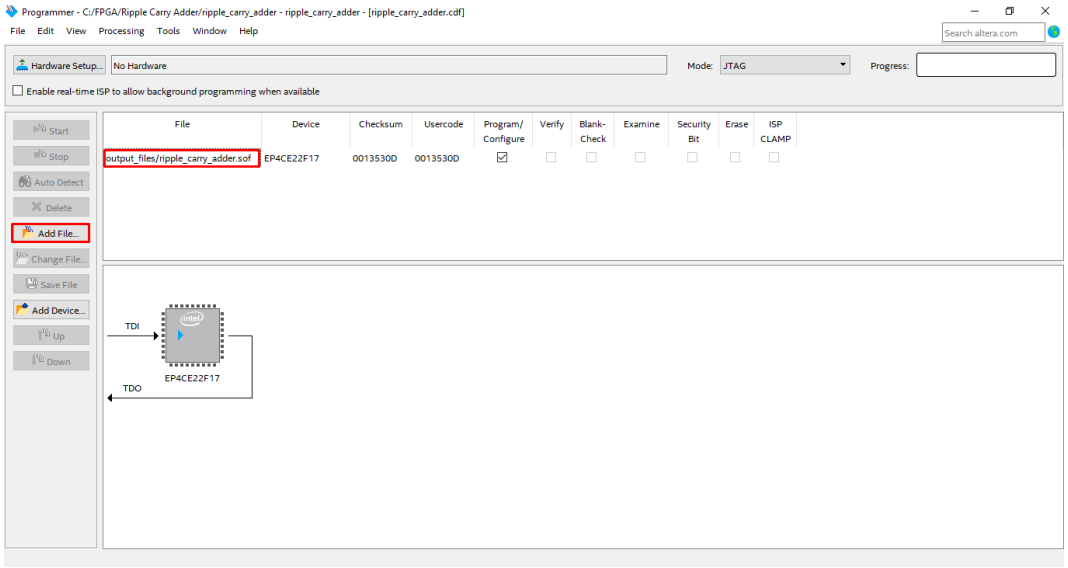

Figure 23: Add file

5. Make sure the Program/Configure checkbox is ticked

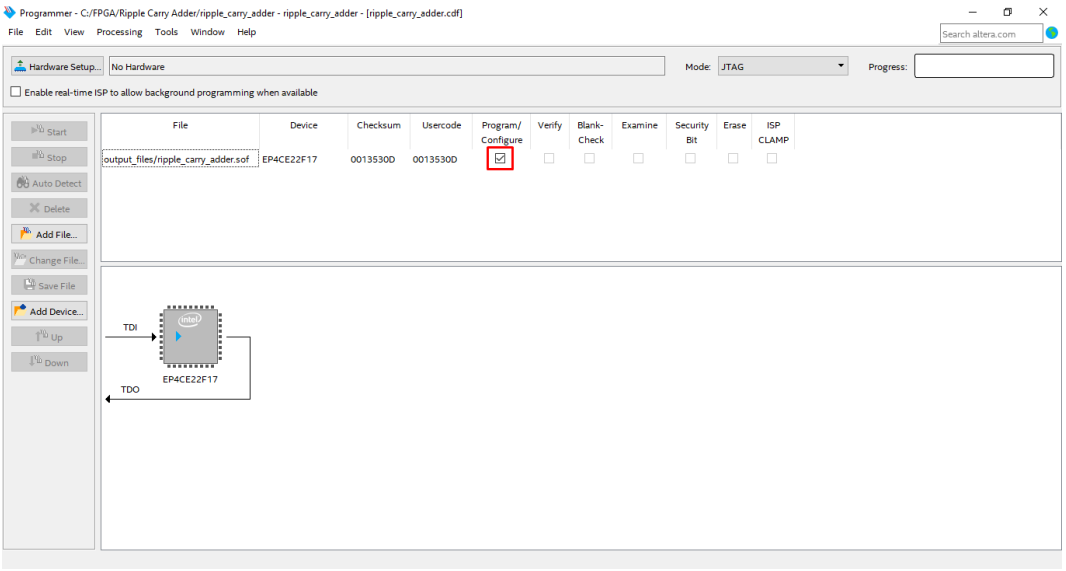

Figure 24: Setting Program Configure

6. When ready, click on **Start** to start the programming process. **Start** button will be enabled when the 'DEO NANO' board is connected to USB port of your device.

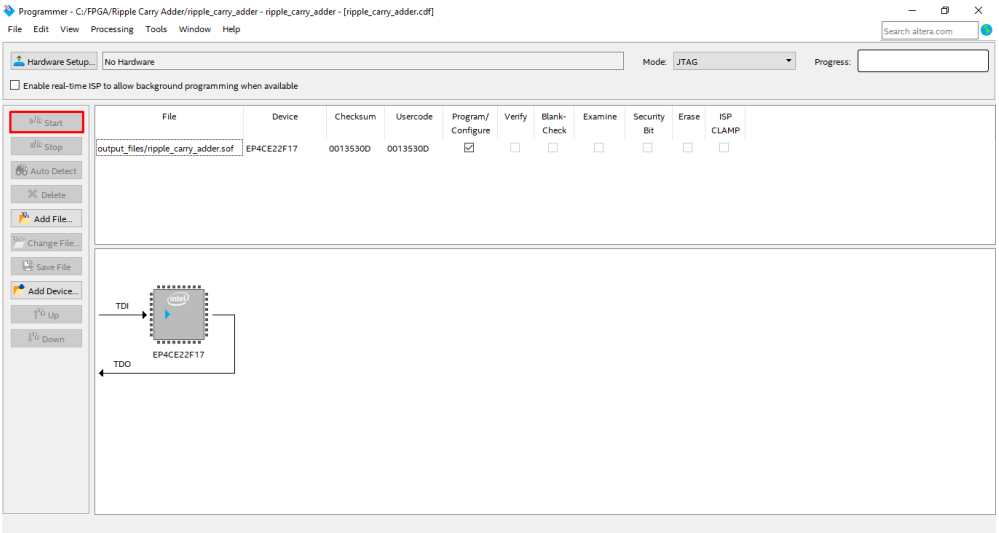

Figure 25: Initiating Design Download into FPGA

Implementing on Modelsim

For more detailed procedure on using ModelSim, refer **Quick Start Guide to Quartus and ModelSim Software** Document. You can find both Verilog HDL and VHDL TestBench code below, make sure to follow only one language in this entire tutorial.

1. Create a New VHDL file in Quartus Prime.Type in the TestBench code provided in this document and save the file with the same name as the module name

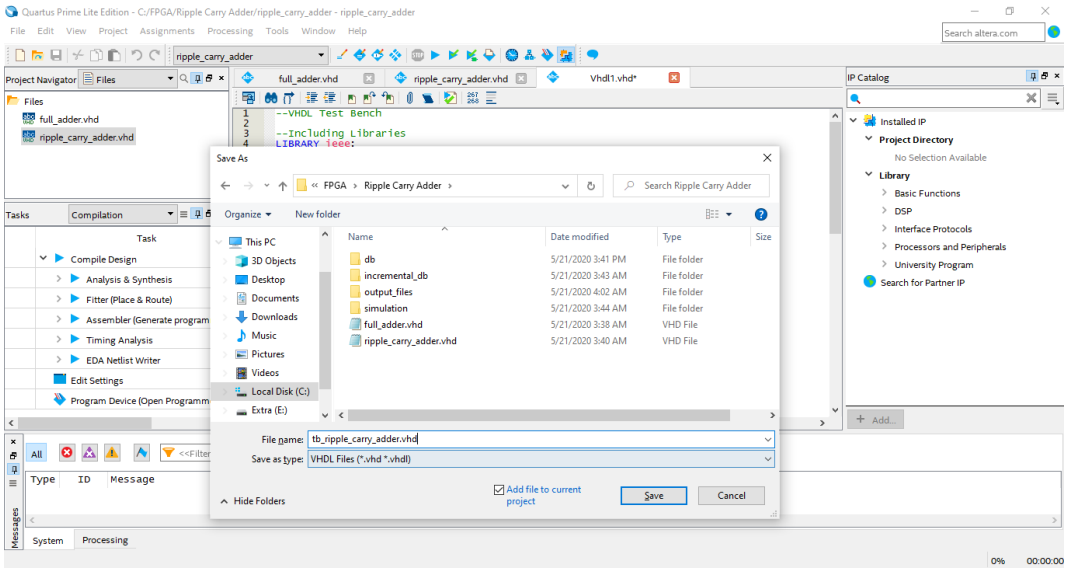

Figure 26: Save TestBench file

2. Go to **Assignments→Settings**.

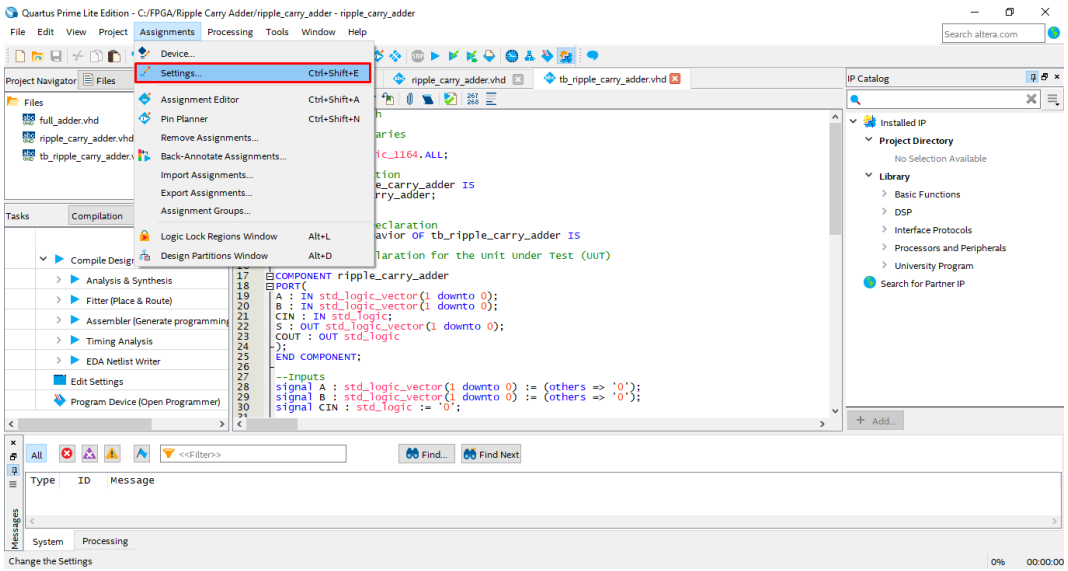

Figure 27: Settings

3. Navigate to **Simulation** under **EDA Tool Settings**.Set the language as VHDL. Select **Compile TestBench** and then click on **Test Benches**.

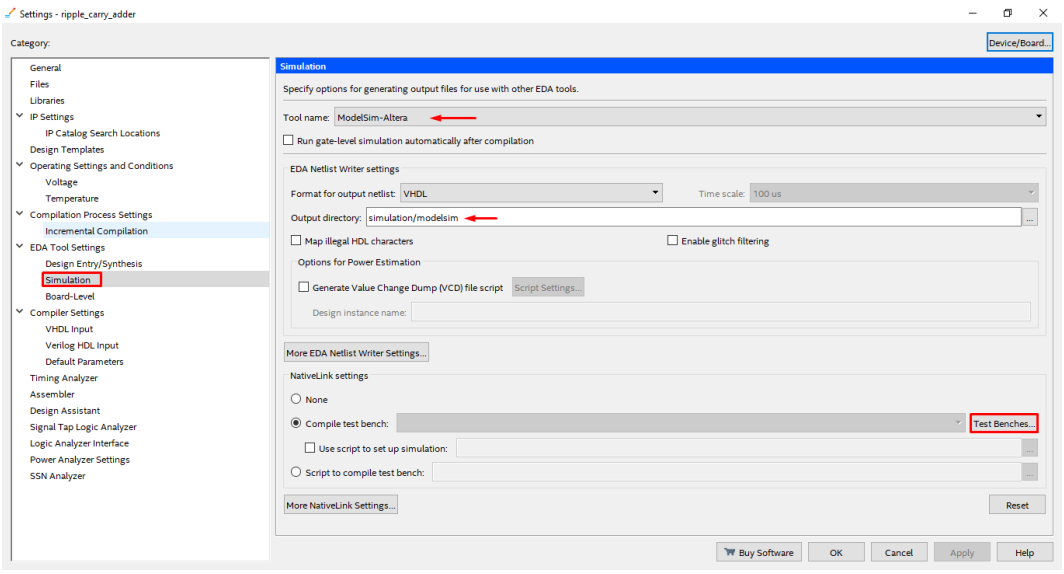

Figure 28: Adding the TestBench file

4. Click on **New**, this opens another dialogue box.

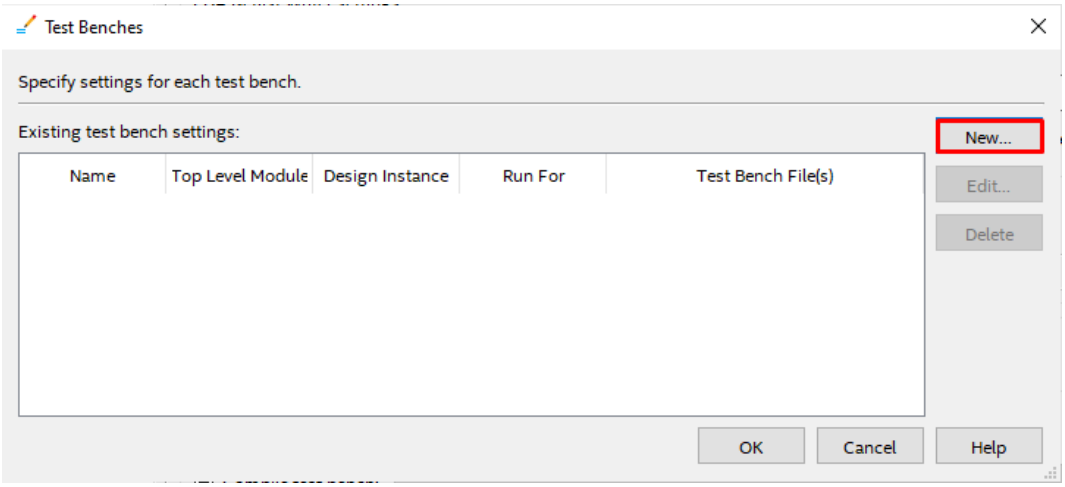

Figure 29: Adding the TestBench file

5. Now type in the TestBench name(In this design , its **tb_ripple_carry_adder**). Now click on the highlighted Browse button.Find the TestBench file(it can be found in the project directory) and click on **Open**. Now click on **Add**,then **OK**.

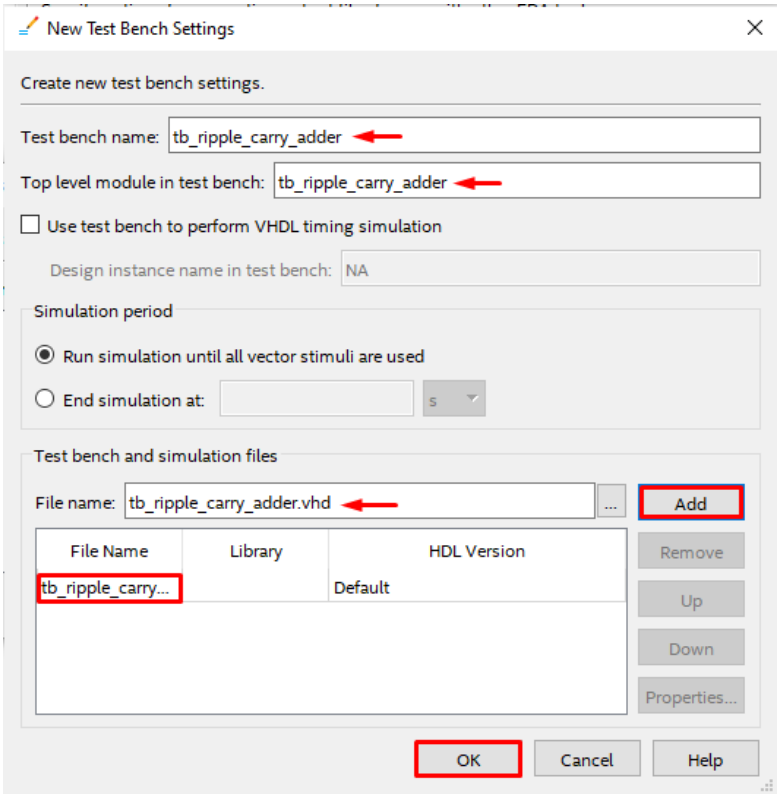

Figure 30: Adding the TestBench file

Functional Simulation using NativeLink Feature

1. Go to **Processing→Start Compilation**

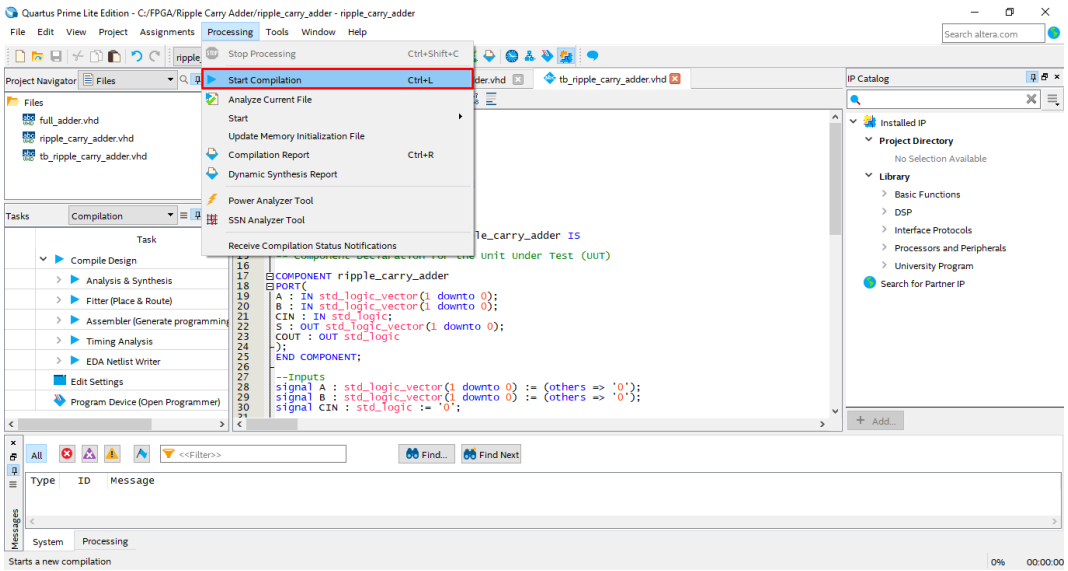

Figure 31: Compiling the project

2. Go to **Tools→Run Simulation Tool→RTL Simulation** to automatically run the EDA simulator(ModelSim-Altera) and to compile all necessary design files.**Note:** If you cannot see the graph, Click on the waveform window and select the **Zoom all** option.

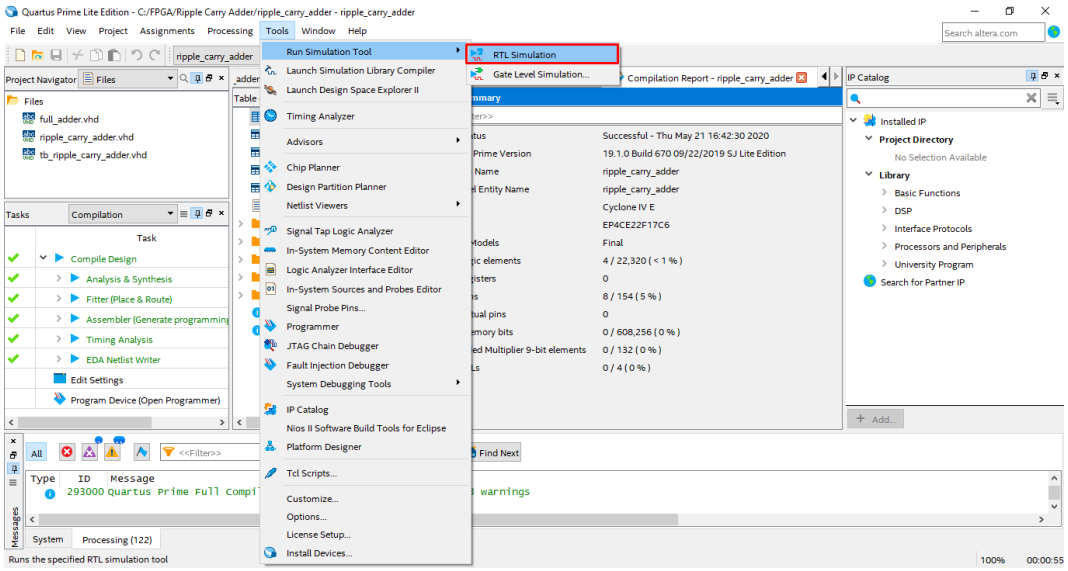

Figure 32: RTL Simulation

3. Finally ModelSim-Altera tool opens up with simulated waveform. click on **Run all** icon on the tool box to display the waveform.

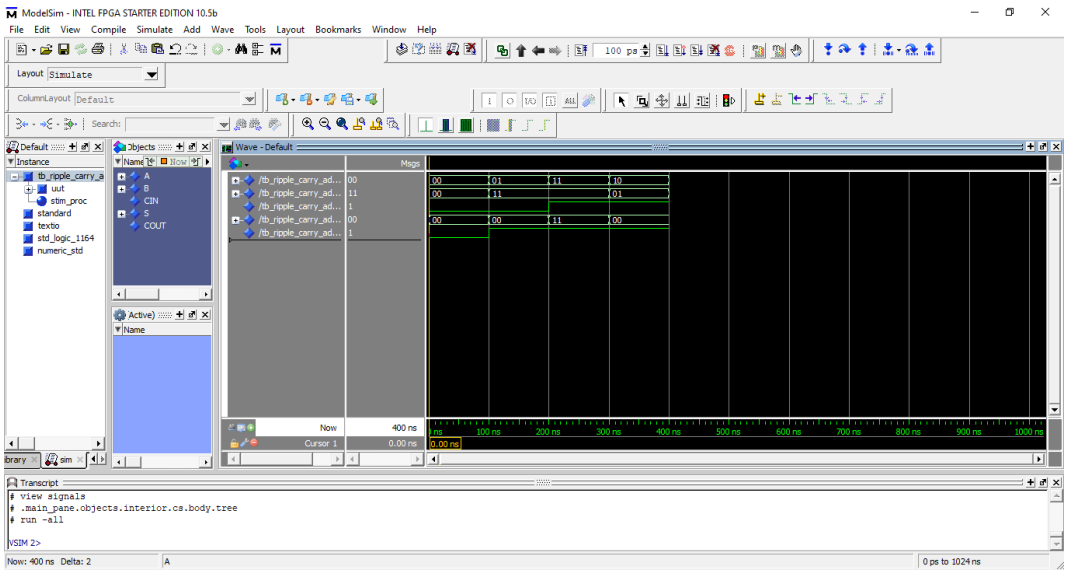

Figure 33: RTL Simulation

Testing the Design

Simulation waveform of the VHDL Design

The Result shown below can be verified by comparing it with the Truth Table provided in page 2 You can observe individual bits of a particular signal by clicking on the '+' icon.

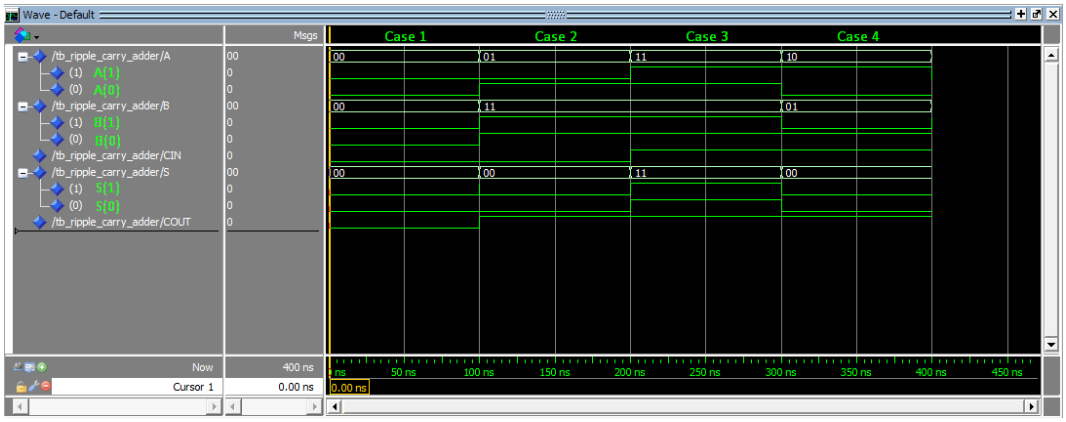

Figure 34: Simulation waveform