Implementation of Universal Shift Register in Verilog HDL/VHDL

Mentors
Prasad T
Aditya Gudla
Simranjeet Singh

Interns
Ajay Chaudhari
Chethan T Bhat
Ritvik Tiwari
Karthik A Shet

Introduction

What is a Register?

Flip flops can be used to store a single bit of binary data (1 or 0). However, in order to store multiple bits of data, we need multiple flip flops. N flip flops are to be connected in an order to store n bits of data. A **Register** is a device which is used to store such information. It is a group of flip flops connected in series used to store multiple bits of data.

Shift Registers

The information stored within these registers can be transferred with the help of **Shift Registers**. Shift Register is a group of flip flops used to store multiple bits of data. The bits stored in such registers can be made to move within the registers and in/out of the registers by applying clock pulses. An n-bit shift register can be formed by connecting n flip-flops where each flip flop stores a single bit of data.
Shift registers are basically of 4 types. These are:

1. Serial In Serial Out shift register
2. Serial In parallel Out shift register
3. Parallel In Serial Out shift register
4. Parallel In parallel Out shift register

Serial-In Serial-Out Shift Register

The shift register, which allows serial input (one bit after the other through a single data line) and produces a serial output is known as Serial-In Serial-Out shift register. Since there is only one output, the data leaves the shift register one bit at a time in a serial pattern, thus the name Serial-In Serial-Out Shift Register.
The logic circuit given below shows a serial-in serial-out shift register. The circuit consists of four D flip-flops which are connected in a serial manner. All these flip-flops are synchronous with each other since the same clock signal is applied to each flip flop.

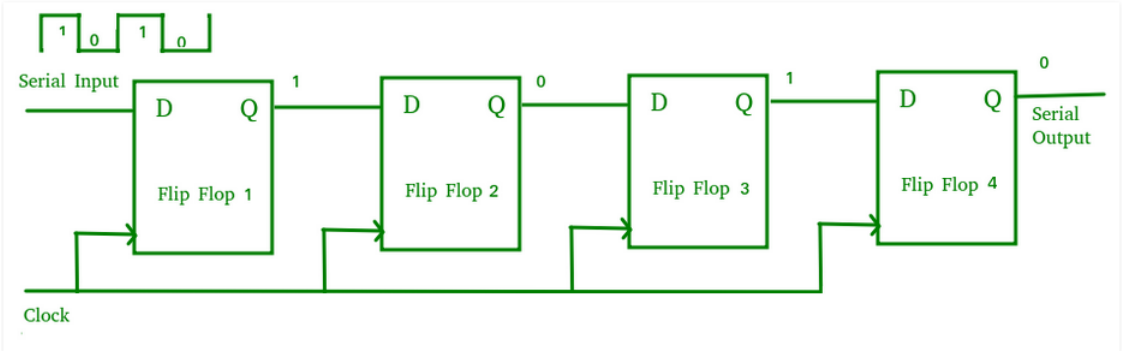

Figure 1: SISO Shift Register

Serial-In Parallel-Out Shift Register

The shift register, which allows serial input (one bit after the other through a single data line) and produces a parallel output is known as Serial-In Parallel-Out shift register.

The logic circuit given below shows a serial-in-parallel-out shift register. The circuit consists of four D flip-flops which are connected. The clear (CLR) signal is connected in addition to the clock signal to all the 4 flip flops in order to RESET them. The output of the first flip flop is connected to the input of the next flip flop and so on. All these flip-flops are synchronous with each other since the same clock signal is applied to each flip flop.

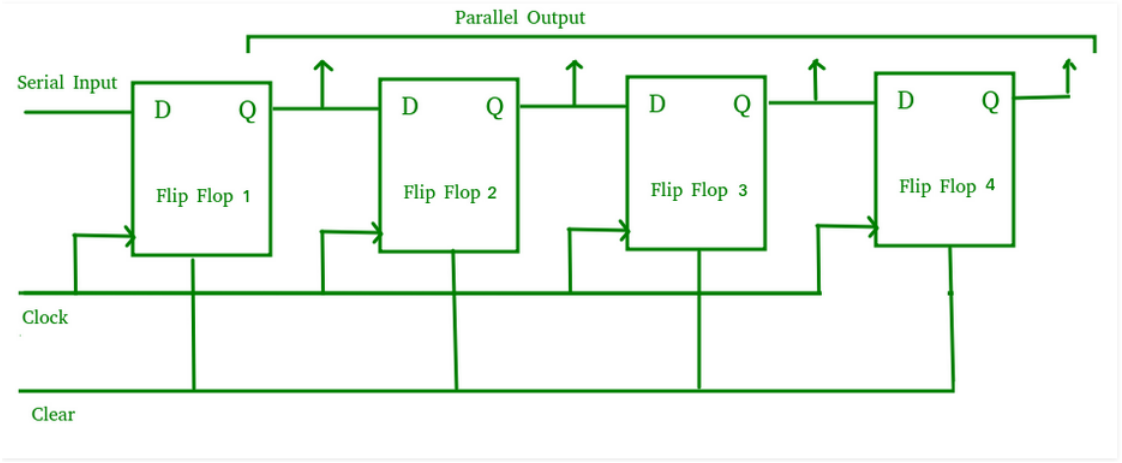

Figure 2: SIPO Shift Register

Parellel-In Serial-Out Shift Register

The PISO(Parellel-in Serial-out) shift register has one of the most important application among the shift register, and it is to convert parellel data into serial data, which is extensively used in communication.
The circuit consists of four D flip-flops which are connected. The clock input is directly connected to all the flip flops but the input data is connected individually to each flip flop through a multiplexer at the input of every flip flop. The output of the previous flip flop and parallel data input are connected to the input of the MUX and the output of MUX is connected to the next flip flop. All these flip-flops are synchronous with each other since the same clock signal is applied to each flip flop.

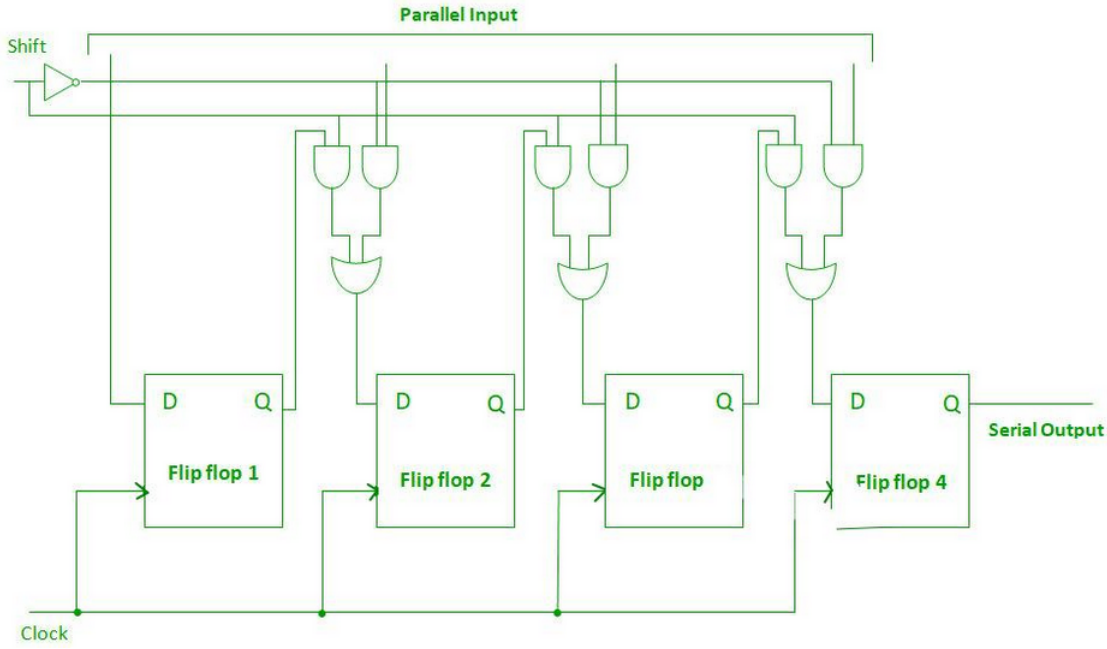

Figure 3: PISO Shift Register

Parallel-In Parallel-Out Shift Register

The shift register, which allows parallel input (data is given separately to each flip flop and in a simultaneous manner) and also produces a parallel output is known as Parallel-In parallel-Out shift register.
The logic circuit given below shows a parallel-in-parallel-out shift register. The circuit consists of four D flip-flops which are connected. The clear (CLR) signal and

clock signals are connected to all the 4 flip flops. In this type of register, there are no interconnections between the individual flip-flops since no serial shifting of the data is required. Data is given as input separately for each flip flop and in the same way, output also collected individually from each flip flop.

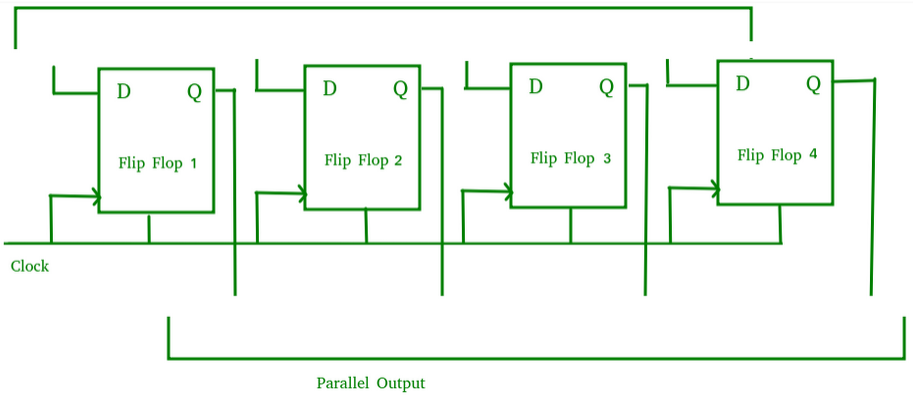

Figure 4: PIPO Shift Register

Universal Shift Register

A Universal shift register is a register is a combination of all the above 4 shift registers with. Universal shift registers are used as memory elements in computers. A Unidirectional shift register is capable of shifting in only one direction. A bidirectional shift register is capable of shifting in both the directions. The Universal shift register is a combination design of bidirectional shift register and a unidirectional shift register with parallel load provision.

S1	S0	REGISTER OPERATION
0	0	No changes
0	1	Shift right
1	0	Shift left
1	1	Parallel load

Figure 5: Choosing the operation through MUX select lines

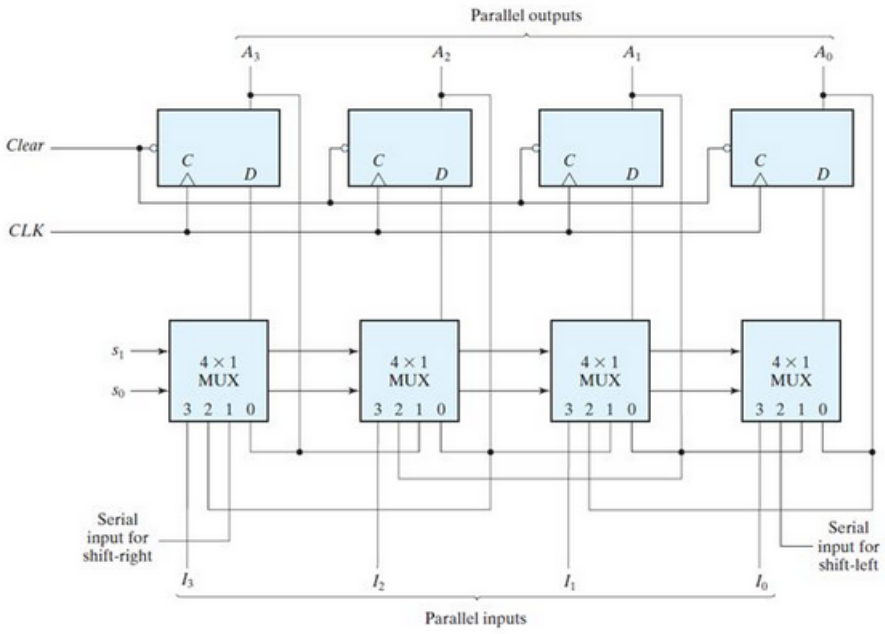

Figure 6: Universal Shift Register

Verilog HDL Code for Universal Shift Register

The Universal Shift Register has been implemented as it contains all different types of shift registers in itself.

RTL Description

```
module universal_shiftreg(
       input [3:0]i, //Define pins for parellel input
       input [1:0]s, //Define select line pins to choose the operation
       input clk,rst,il,ir, //Define clock,reset, serial input for
                           //shift left and serial input for shift right
       output reg [3:0]b); //Define output pins



always @ (posedge clk, negedge rst ) //Execute the logic whenever positive edge of
                                    //clock or negative edge of Reset is encountered
begin
       if(rst==0) //if rst = 0, then clear the output
              b=4'b000;
       else
       begin
              case(s) //check select lines to determine type of
                      //operation
                     2'b00:begin end //Retain the data
                     2'b01:begin
                            b={ir,b[3:1]}; //shift right operation
                            end
                     2'b10:begin
                            b={b[2:0],il}; //Shift left operation
                            end
                     2'b11:begin
                            b = i; //parellel input-output operation
                            end
                     default:begin end
              endcase
       end
end
endmodule
```

Testbench

Verilog:

```
module universalshiftreg_tb;
reg [3:0]i; reg[1:0]s; reg clk; reg rst; reg il;reg ir; //Define all input ports
wire [3:0]b; //Define all output ports

universal_shiftreg uut(.i(i), //Map testbench ports with DUT ports
                       .s(s),
                                          .clk(clk),
                                          .rst(rst),
                                          .b(b),
                                          .il(il),
                                          .ir(ir));

initial begin

       rst = 1'b0; //Initialze values on all input pins
       clk = 0;
       ir = 0;
       il = 0;
       s = 2'b00;
       i = 4'b0000;
end

always
begin
       clk = ~clk; #5; //Define Clock operation
end

initial begin
       #50;
       rst = 1; #50 //Different Combinations of input
       ir = 1;
       s = 2'b01;#10;
       ir = 0;#10;
       ir = 1;#10;
       ir = 0;#10;
       il = 1;
       s = 2'b00;#50
```

```
        s = 2'b10;
        il = 1;#10
        il = 0;#10
        il = 0;#10
        s = 2'b11;#5
        i = 4'b1111;
        #100;
end
endmodule
```

Implementation in Quartus Prime

To Implement the design in Quartus Software, follow these steps(For more detailed information on the following steps, refer 'Quick Start Guide to Quartus Prime Lite Software' Document):
Note: The language used is Verilog HDL. Device is DE0-Nano FPGA Board

1. Create a New Project using the Project Wizard.

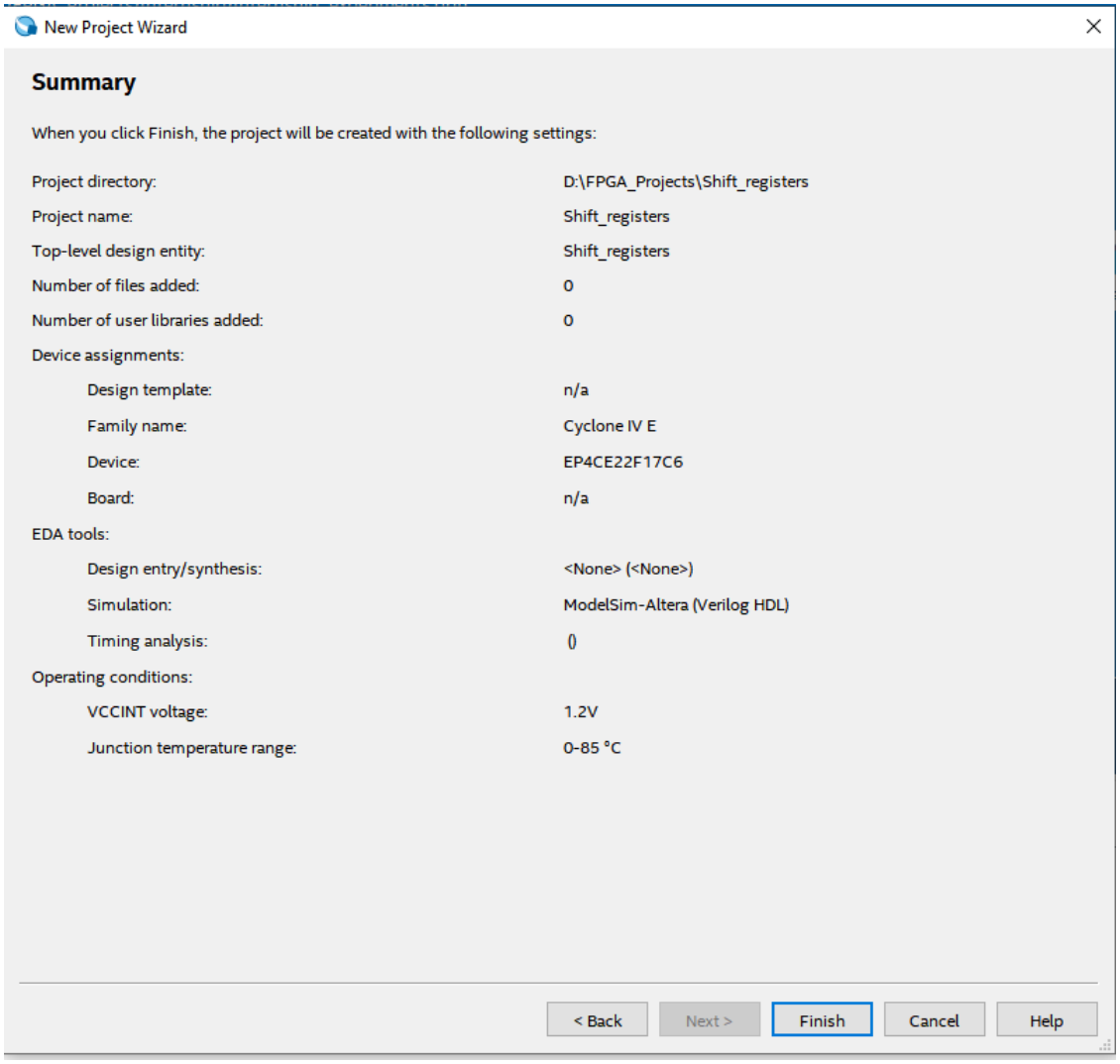

Figure 7: Project Summary

2. Create a New Verilog File

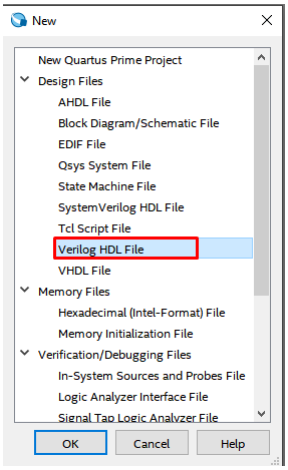

Figure 8: Creating a Verilog File

3. Type in the Verilog code provided in this Document and save the file(make sure file name is same as module name)

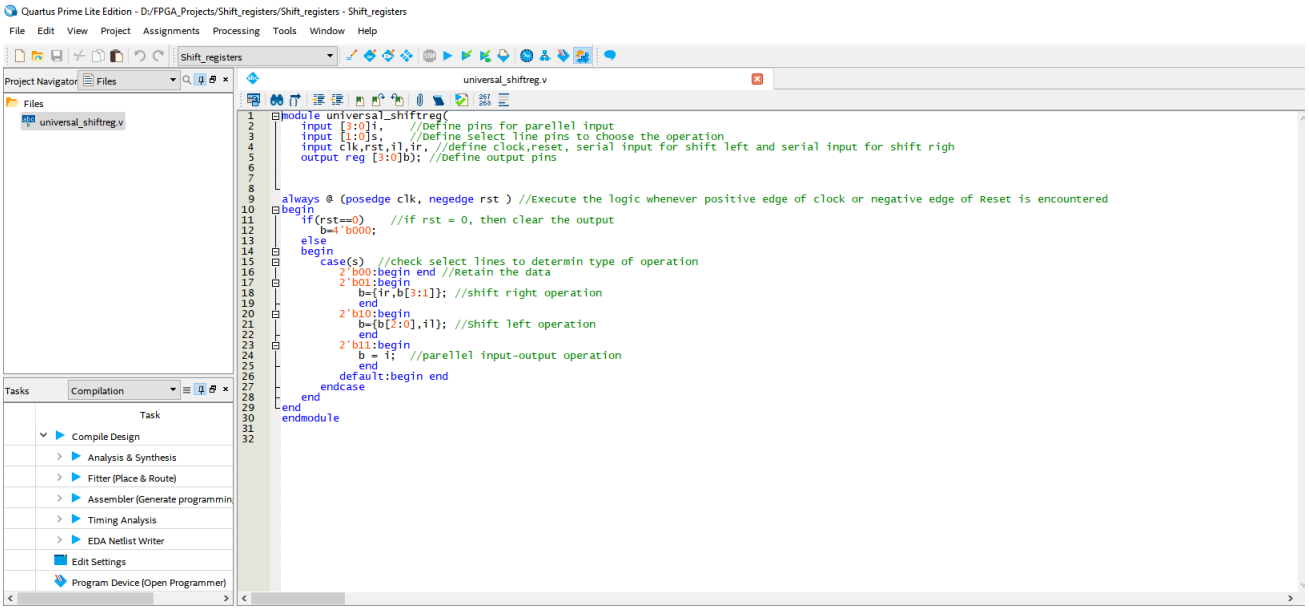

Figure 9: Code typed and saved

Note: If module name is different from project name, then manually set the verilog file as Top-Level Module by:Right Click on '<**File_name**>'→**Set as Top-Level Entity**

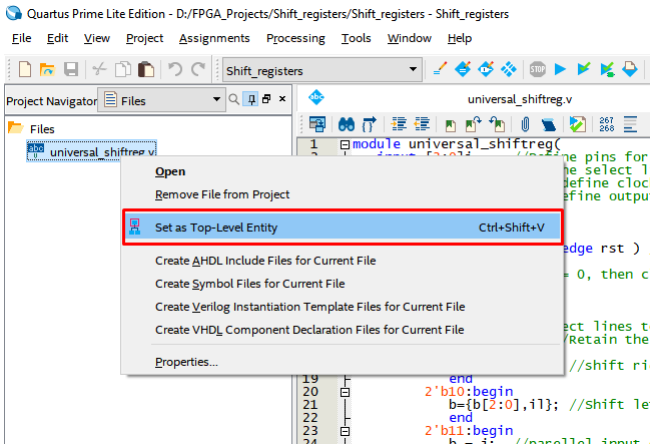

Figure 10: Setting top-level entity

4. Compile the Design by either by double clicking '**COMPILE DESIGN**' or clicking on the '**Play**' button

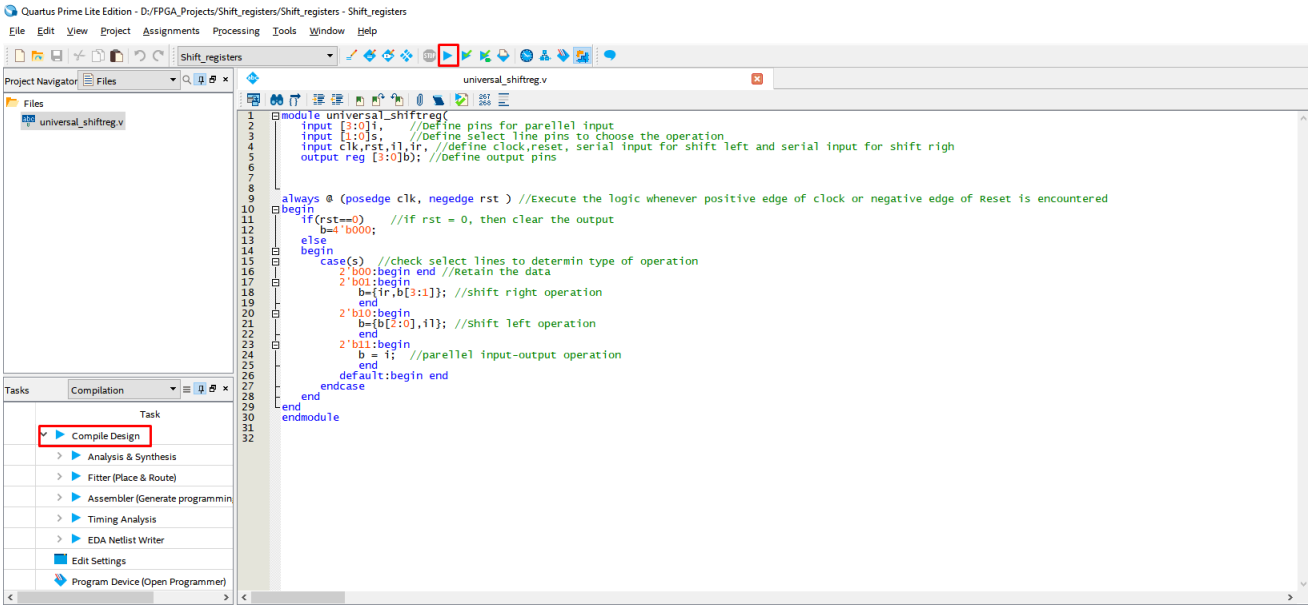

Figure 11: Code compilation

Note: The Compilation is complete when you see these **Green Tick** marks.If even one of them have **Red Cross** it means the compilation has failed

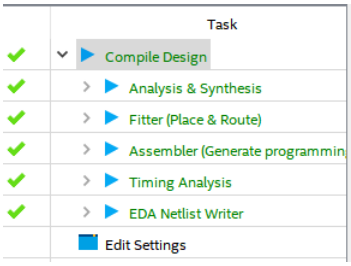

Figure 12: Successful compilation

5. Upon successful compilation,to view the RTL Circuit, Go to **TOOLS→NETLIST VIEWERS→RTL VIEWER**

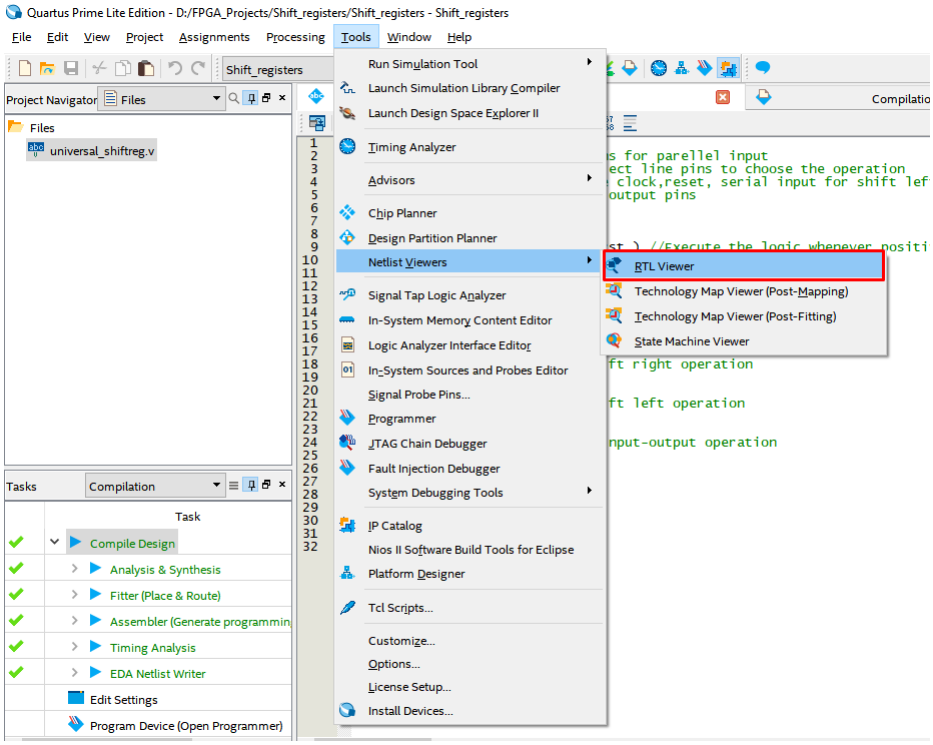

Figure 13: Opening the RTL Viewer

RTL Circuit of the Implemented Design

The below RTL circuit

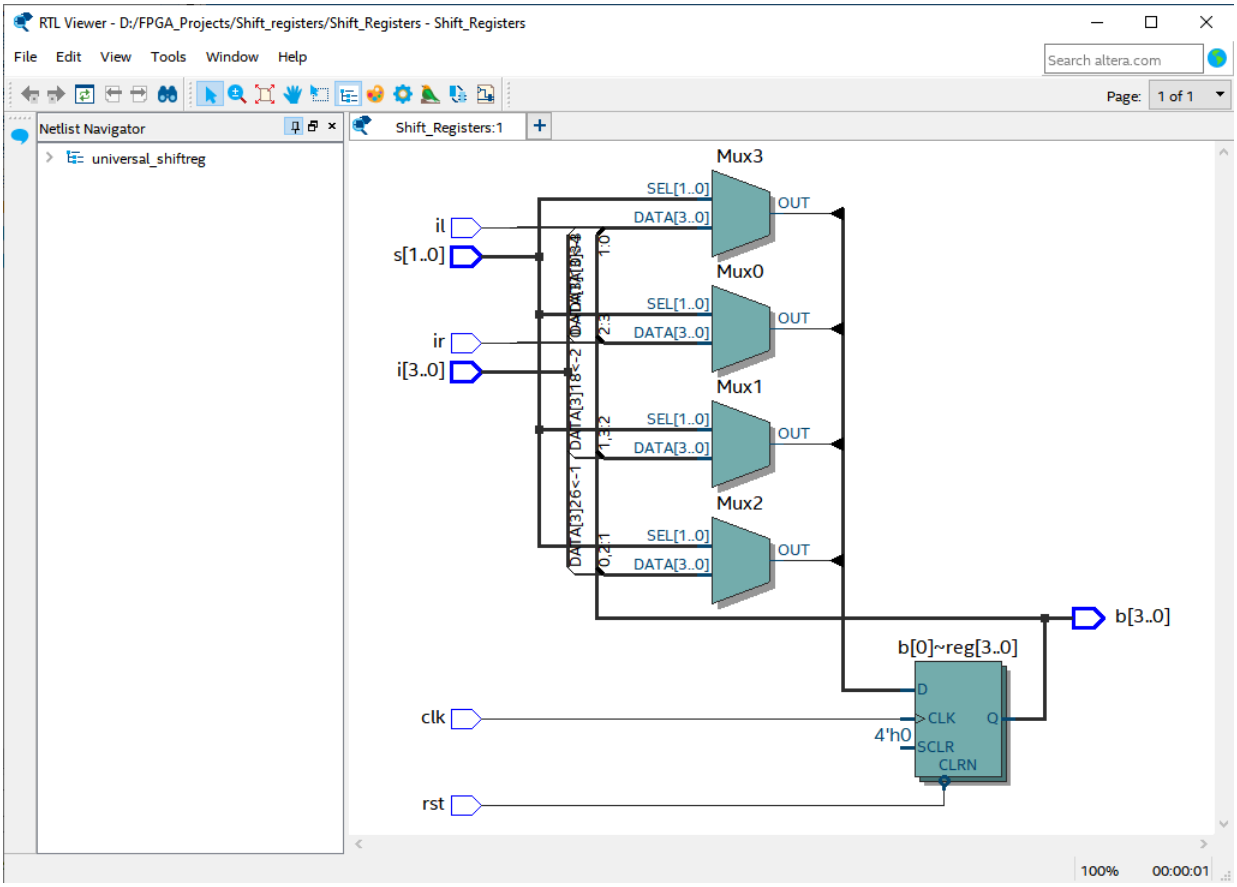

Figure 14: RTL Circuit of Universal Shift Register

Pin Assignment

1. Open the **Pin Planner** by going to **ASSIGNMENTS→PIN PLANNER**

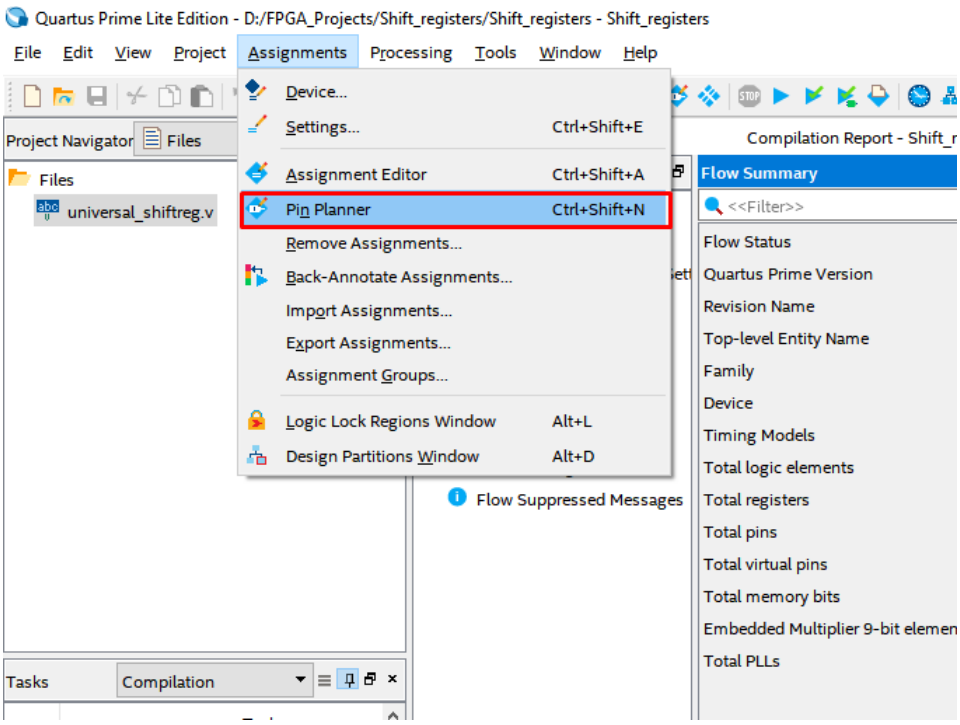

Figure 15: Opening the PIN PLANNER

2. Now assign the pins by either dragging and dropping the nodes(Blue box) on the required pins(**Black** box), or click on the drop down menu(Red box) to search and select a pin. To find out which pin to use, refer the manual of your board. For the DE0-Nano FPGA Board, the manual can be found at `https://www.ti.com/lit/ug/tidu737/tidu737.pdf`

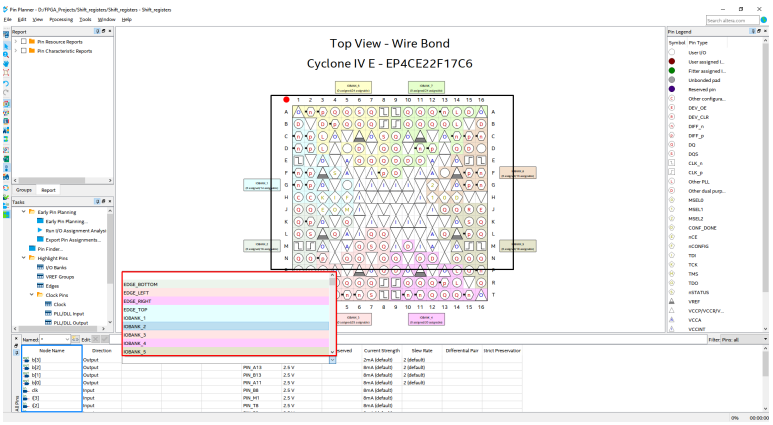

Figure 16: Assigning Pins

Here is an example of assigning a pin for the output 'b' in our design

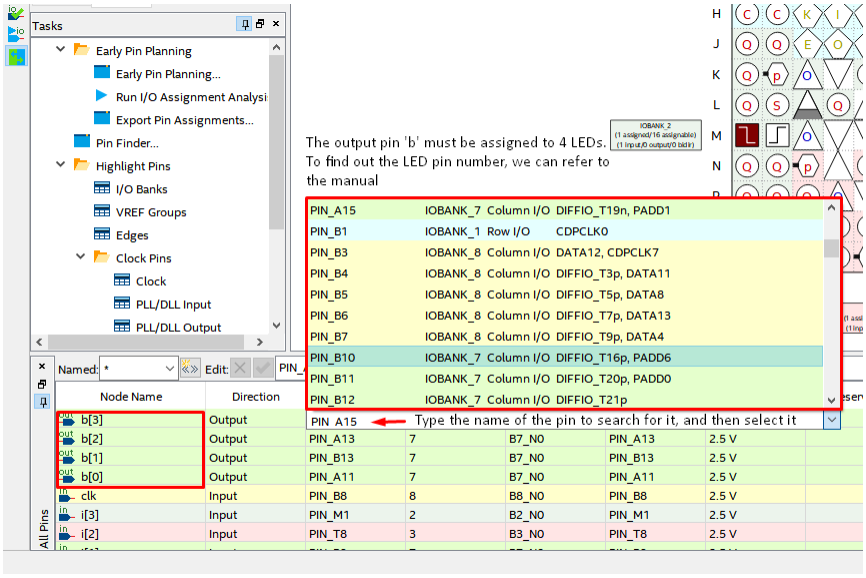

Figure 17: Assigning pin for b[0]

Note: Make sure to select the appropriate I/O standard by referring to the board manual

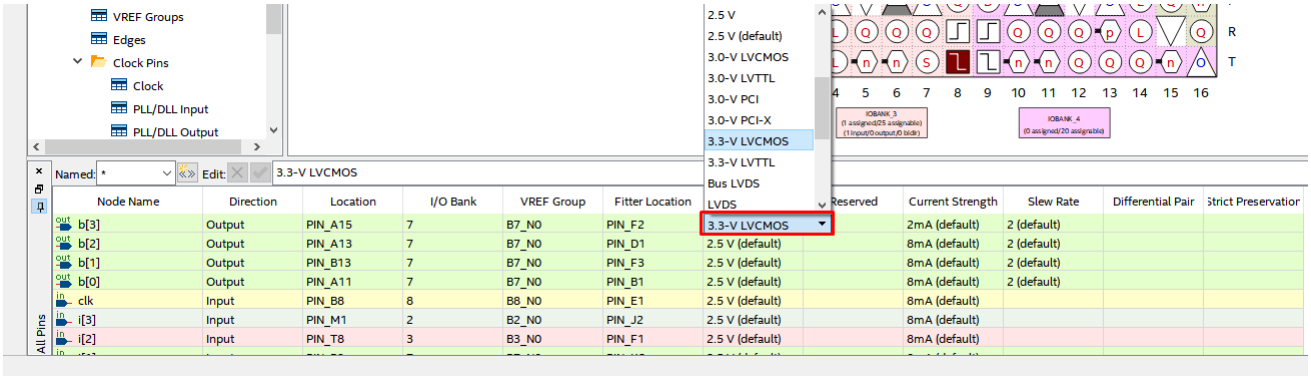

Figure 18: Assigning I/O Standards

3. Once all the pins are assigned, Go to **PROCESSING→START I/O ASSIGNMENT ANALYSIS**. The analysis checks pin assignments and surrounding logic for illegal assignments and violations of board layout rules.

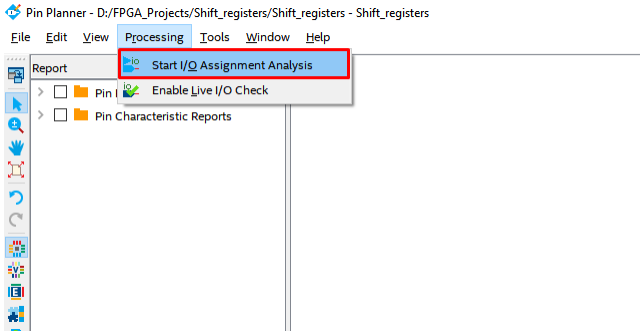

Figure 19: Starting the I/O Assignment Analyser

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. A file with .SOF extension is created during compilation. This file is used to load the program to the board

1. Open the programmer by going to 'Tools'→ 'Programmer'

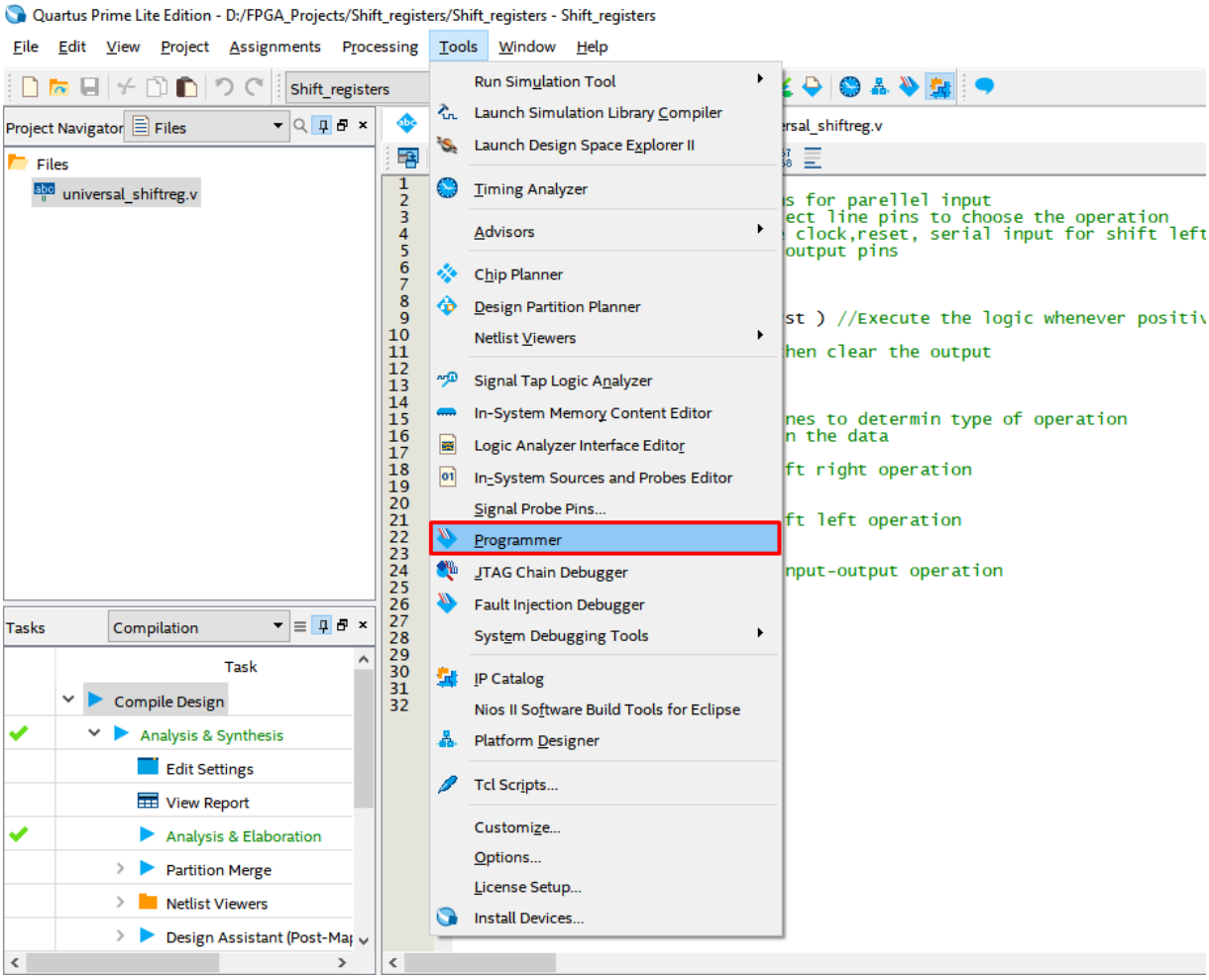

Figure 20: Opening programmer

2. 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

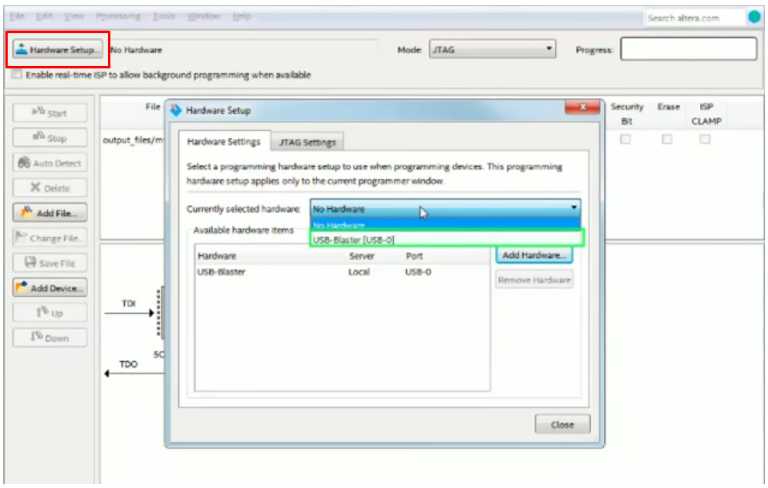

Figure 21: Choosing Hardware

3. 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

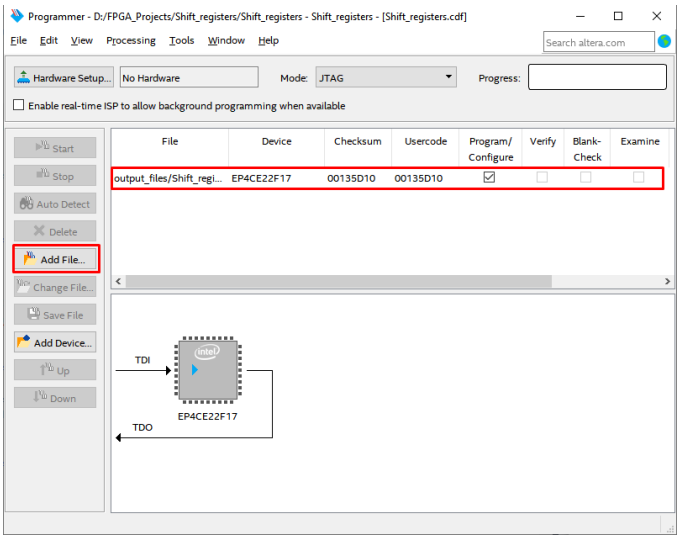

Figure 22: Adding file if its not already listed

4. Make sure the program/configure checkbox is ticked

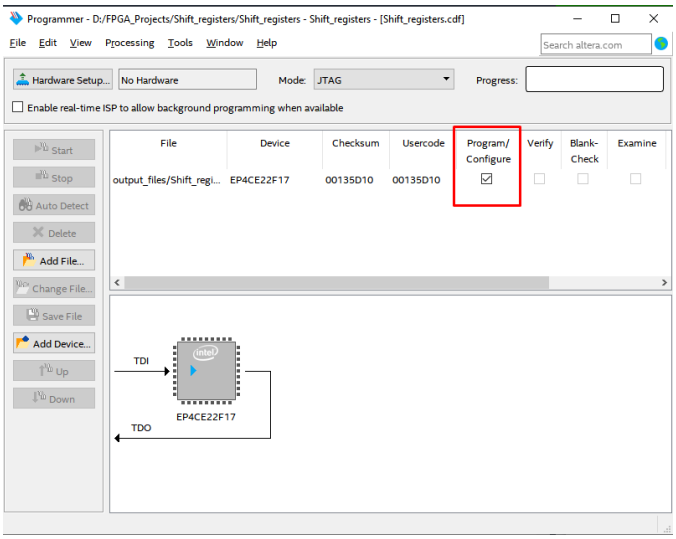

Figure 23: Program/Configure is checked

5. 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. **Note:**The start button will be highlighted when the board is connected

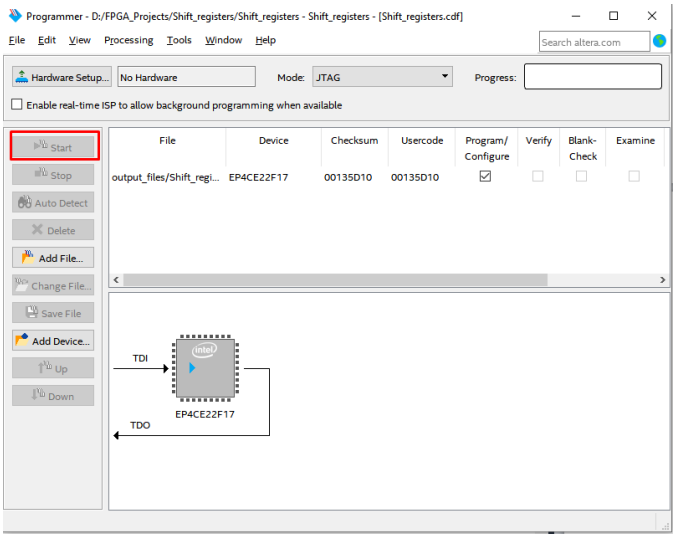

Figure 24: Starting the download

Implementing in ModelSim

The testbench shown here is a Verilog Testbench. For more detailed procedure on using ModelSim, refer 'Quick Start Guide to Quartus and ModelSim Software' Document

1. Create a new verilog file in Quartus Prime

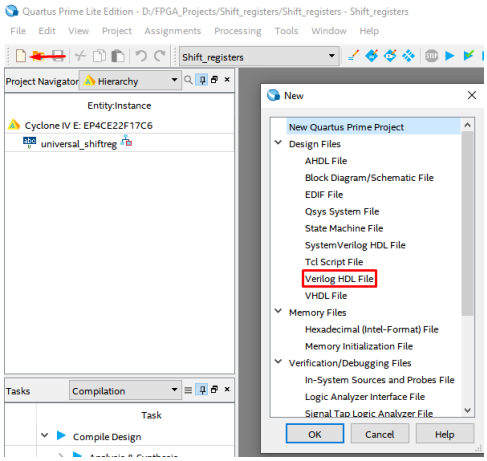

Figure 25: Creating new verilog file

2. Type in the Testbench code provided in this document and save the file with the same name as the module name

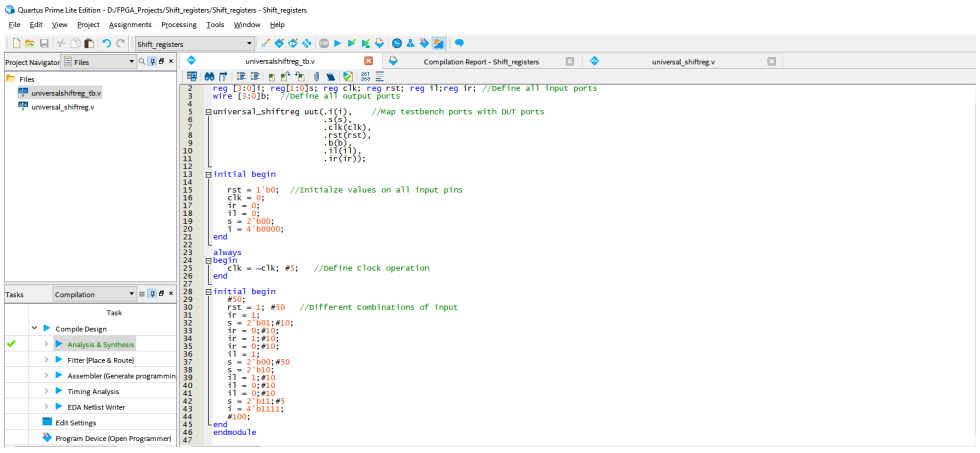

Figure 26: typing the testbench code

3. Go to **ASSIGNMENTS→SETTINGS**

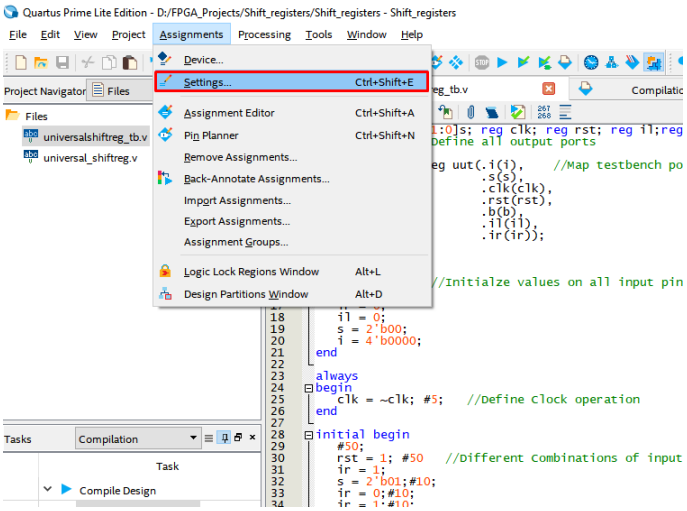

Figure 27: opening settings

4. Navigate to **SIMULATION** under **EDA TOOL SETTINGS**

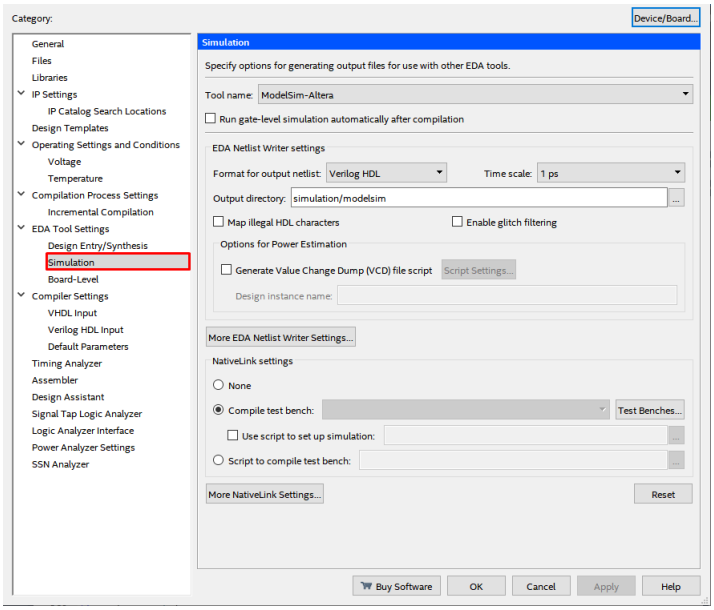

Figure 28: Navigate to simulation settings

5. set the language(**FORMAT FOR OUTPUT NETLIST**) as Verilog HDL. Select **COMPILE TEST BENCH** and then click on **TEST BENCHES...**

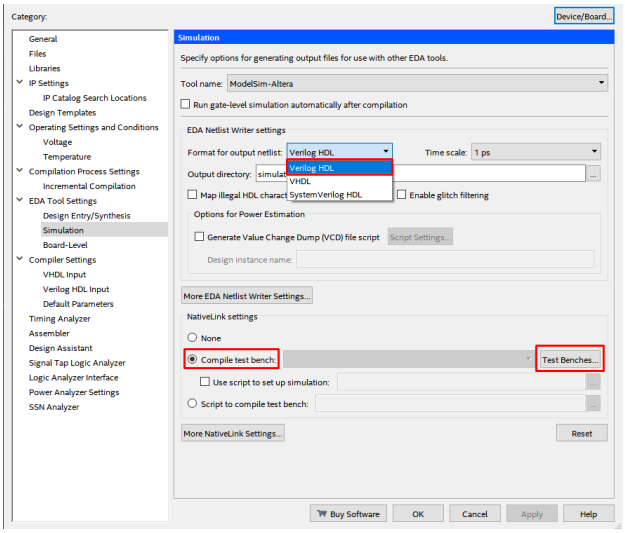

Figure 29: Configuring the Settings

6. Click on **NEW**, this opens another dialogue box. Now type in the testbench name(In this design , its **universal_tb**). Now click on the highlighted browse button

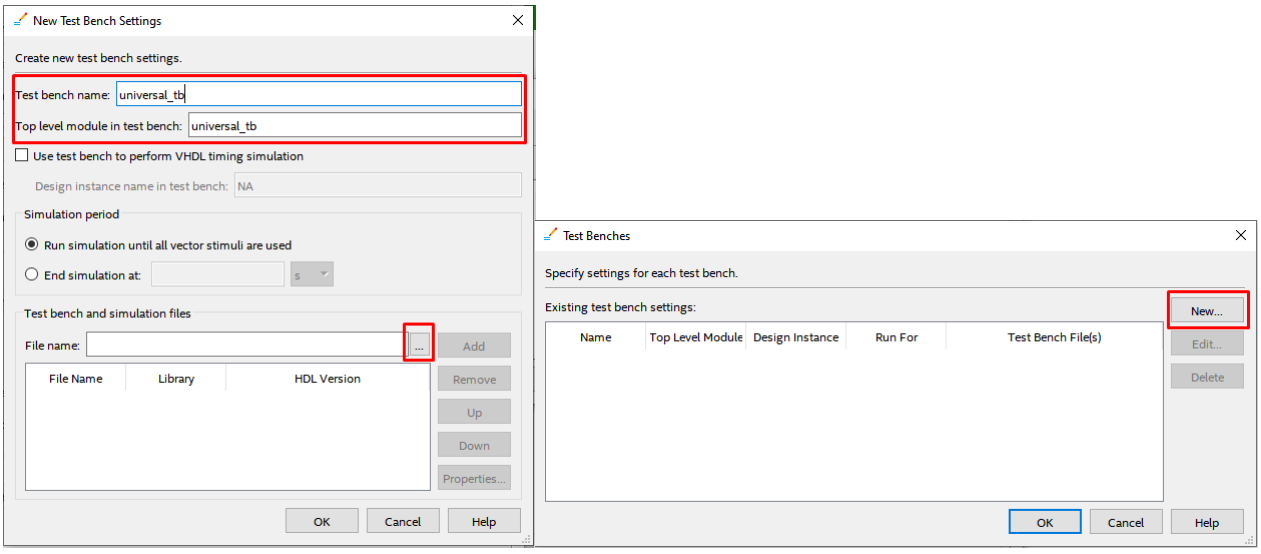

Figure 30: Adding the Tesbench file

7. Find the testbench file(it can be found in the project directory) and click on **OPEN**

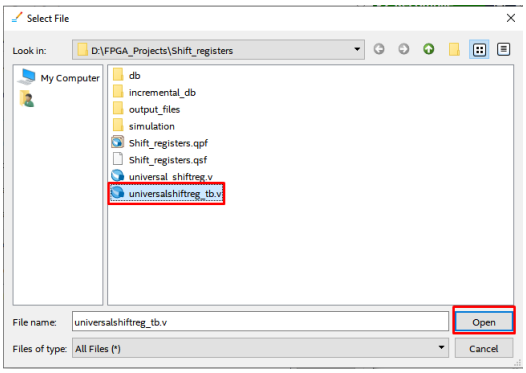

Figure 31: Finding the file in the project directory

8. Now click on **ADD**,then **OKAY**,then **OKAY** again and finally click on **APPLY** **Note:**The **APPLY** button will be highlighted after **OKAY** has been clicked on all previous windows

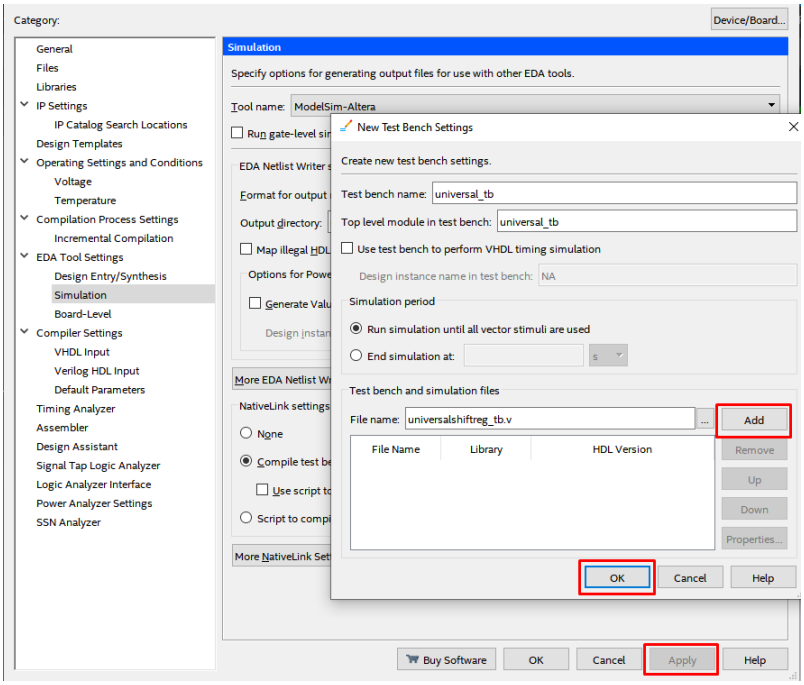

Figure 32: Finalising the testbench adding process

9. To start the simulation, go to **TOOLS→RUN SIMULATION TOOL→RTL SIMULATION**. This will open ModelSim and display the simulation waveform

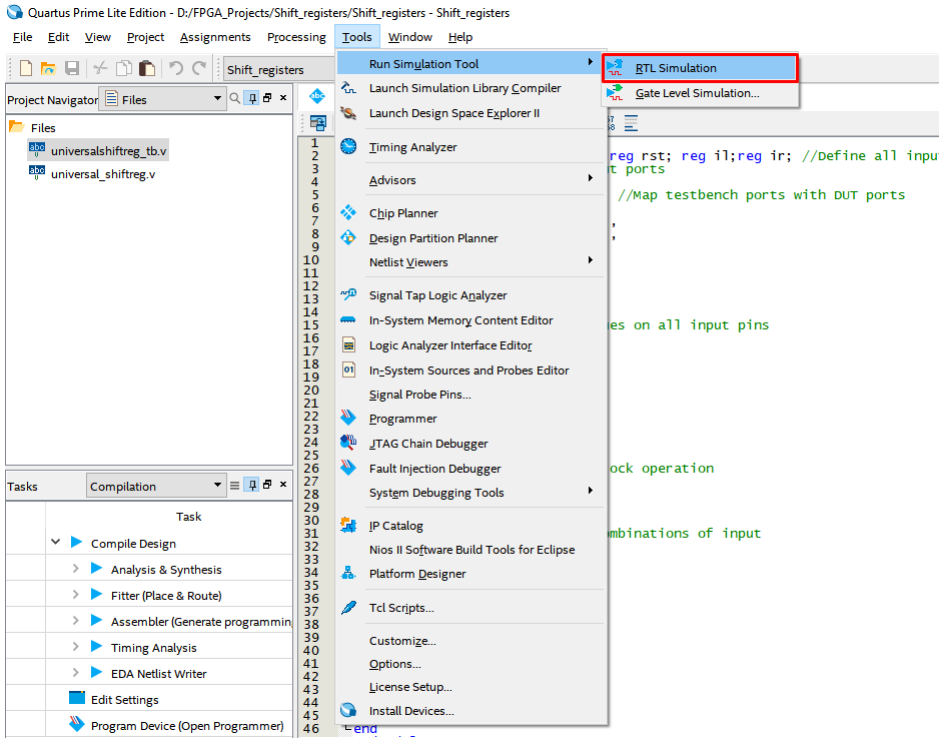

Figure 33: Running the Simulation

Simulation Results

The waveform shown below is the result of the simulation.
Initially when **rst=0**, the output is reset to '**0000**'. After **50 ns**, **rst** is set and the register is in normal operation. When **s = 00**, the register retains the previous data. At **0.1 ns**, **s=01**, so the shift register is set to **'Shift right operation'**. At every consecutive clock cycle until **0.14 ns**, it can be seen that the shift register ouput is shifted right while the MSB is loaded with the value of **'ir'**. At **0.14ns**, **s=00**, so until **0.19ns**, the shift register retains the value. At **0.19 ns**, **s=10**, The shift register is set to **'Shift left Operation'** and at each consecutive clock cycle, the Shift register shifts the data left and loads the LSB with data on the **'il'** pin. Finally, at **0.22 ns**, **s=11**, this sets the shift register to **Parellel input mode**, so for every consecutive clock cycle, the shift register is updated with the data on the 4 bit parellel input **'i'**.

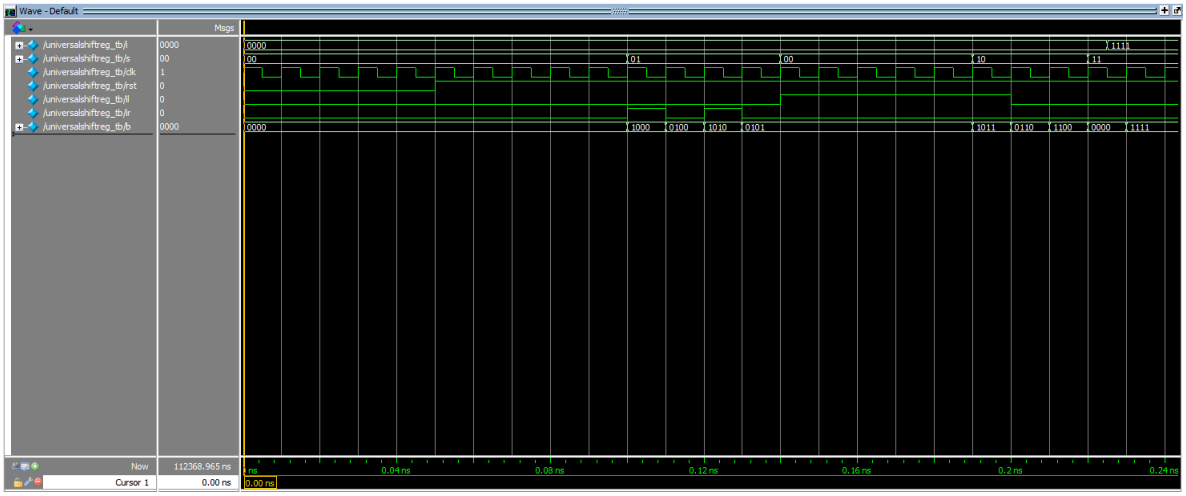

Figure 34: Simulation Waveform

1 VHDL Code for the Universal Shift Register

The Entire design can also be implemented using a VHDL code. Irrespective of whether its VHDL or Verilog, the implementation process remains the same

1.1 RTL Description

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY uni_shift IS
PORT (clock, clear, sl_in, sr_in : in bit;
mode : IN BIT_VECTOR ( 1 DOWNTO 0 );
data : IN BIT_VECTOR ( 3 DOWNTO 0 );
q : INOUT BIT_VECTOR (3 DOWNTO 0 ));
end uni_shift;

ARCHITECTURE behav OF uni_shift IS

BEGIN
PROCESS (clock, clear)
BEGIN -- Asynchronous, active-low Clear input:
IF clear = '0' THEN
q <= "0000" ; -- Rising edge-triggered D flip-flops:
ELSEIF clock'event AND clock = '1' THEN

CASE mode IS
WHEN "00" => null; -- "Do Nothing" mode: retain current flip-flop outputs
WHEN "01" => q <= (q srl 1) OR (sr_in & "000") ; -- Shift Right Serial Input
WHEN "10" => q <= (q sll 1) OR ("000" & sl_in) ; -- Shift Left Serial Input
WHEN "11" => q <= data ; -- Parallel (Broadside) Load
END case;
END if;
END process;
END behav;
```

1.2 TestBench

```
--VHDL Test Bench

--Including Libraries
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

--Entity Declaration
 ENTITY tb_uni_shift IS
END tb_uni_shift;


--Architecture Declaration
ARCHITECTURE behavior OF tb_uni_shift IS

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

COMPONENT uni_shift
PORT(clock, clear, sl_in, sr_in : IN bit;

mode : IN BIT_VECTOR ( 1 DOWNTO 0 );

data : IN BIT_VECTOR ( 3 DOWNTO 0 );

q : INOUT BIT_VECTOR (3 DOWNTO 0 )

);
END COMPONENT;

--Inputs
SIGNAL clear : BIT:= '0';
SIGNAL clock : BIT:= '0';
SIGNAL sl_in : BIT:= '0';
SIGNAL sr_in : BIT:= '0';
SIGNAL mode : BIT_VECTOR(1 DOWNTO 0) := (OTHERS => '0');
SIGNAL data : BIT_VECTOR(3 DOWNTO 0) := (OTHERS => '0');

--Outputs
SIGNAL q: BIT_VECTOR(3 DOWNTO 0) := (OTHERS => '0');


BEGIN

-- Instantiate the Unit Under Test (UUT)
uut: uni_shift PORT MAP (
```

```
clear=> clear,
clock => clock,
sl_in => sl_in,
sr_in => sr_in,
mode => mode,
data => data,
q => q
);

 -- Clock process definitions
   clk_process :PROCESS
   BEGIN
               clock <= '0';
               WAIT FOR 5 ns;
               clock <= '1';
               WAIT FOR 5 ns;
   END PROCESS;

-- Stimulus process
stim_proc: PROCESS
BEGIN
--Different Combinations of Input
clear <= '0';
WAIT FOR 50 ns;
clear <= '1';
sr_in <= '1';
mode <= "01";
WAIT FOR 10 ns;
sr_in <= '0';
WAIT FOR 10 ns;
sr_in <= '1';
WAIT FOR 10 ns;
sr_in <= '0';
WAIT FOR 10 ns;
sl_in <= '1';
mode <= "00";
WAIT FOR 50 ns;
mode <= "10";
sl_in <= '1';
WAIT FOR 10 ns;
sl_in <= '0';
WAIT FOR 10 ns;
sl_in <= '0';
WAIT FOR 10 ns;
```

```
mode <= "11";
WAIT FOR 5 ns;
data <= "1111";
WAIT FOR 100 ns;

END PROCESS;

END;
```