

SEVEN SEGMENT DISPLAY



Mentors

Prasad Trimukhe
Aditya Gudla
Simranjeet Singh

Interns

Ajay Chaudhari
Chethan T Bhat
Ritvik Tiwari
Karthik A Shet

Introduction

The Seven Segment display has seven LEDs arranged into the number eight. They are both cost-effective and easy to use. The Figure 1 shows a standard seven-segment display.

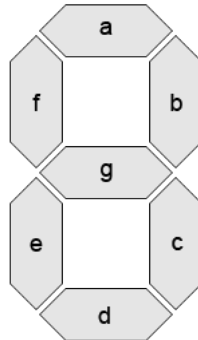


Figure 1: Seven Segment Display

There are two types of seven-segment displays, **Common Anode** and **Common Cathode**. The Internal structure of each of these types is nearly the identical. However, the polarity of the LEDs and common terminal are different. The diagram below shows the internal structure of the common cathode seven-segment display (the one is used in this tutorial) and common anode seven-segment display.

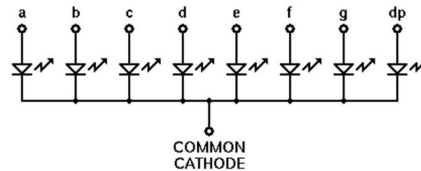


Figure 2: Seven Segment Display with Common Cathode

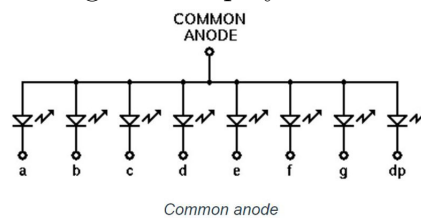


Figure 3: Seven Segment Display with Common Anode

- The Common Cathode Display (CCD) – In the common cathode display, all the cathode connections of the LED's are joined together to logic "0" or ground. The individual segments are illuminated by application of a "HIGH", logic "1" signal to the individual Anode terminals.
- The Common Anode Display (CAD) – In the common anode display, all the anode connections of the LED's are joined together to logic "1" and the individual segments are illuminated by connecting the individual Cathode terminals to a "LOW", logic "0" signal.

But, seven segment display does not work by directly supplying voltage to different segments of LEDs. First, our decimal number is changed to its BCD equivalent signal then BCD to seven segment decoder converts that signals to the form which is fed to seven segment display. This BCD to seven segment decoder has four input lines (A, B, C and D) and 7 output lines (a, b, c, d, e, f and g), this output is given to seven segment LED display which displays the decimal number depending upon inputs.

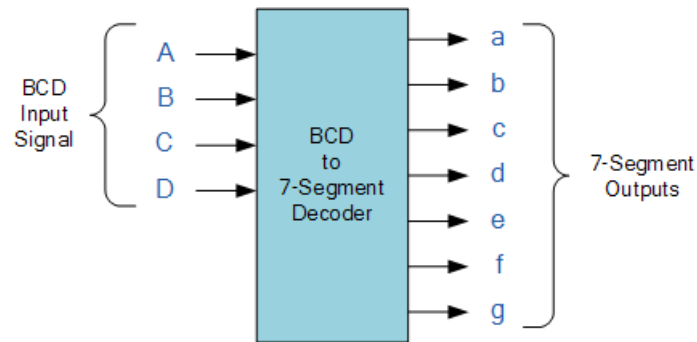


Figure 4: Design of BCD Decoder 4:7

Truth Table for Seven Segment Display									
NO.	BCD	Display	a	b	c	d	e	f	g
0	0000		1	1	1	1	1	1	0
1	0001		0	1	1	0	0	0	0
2	0010		1	1	0	1	1	0	1
3	0011		1	1	1	1	0	0	1
4	0100		0	0	1	0	0	1	1
5	0101		1	0	1	1	0	1	1
6	0110		1	0	1	1	1	1	1
7	0111		1	1	1	0	0	0	0
8	1000		1	1	1	1	1	1	1
9	1001		1	1	1	1	0	1	1

Table 1: Truth Table for Seven Segment Display

Code

Below is the VHDL code is for Seven Segment Display

```
-----
-- File:      Seven_Segment_Display.vhd
-- Description: This is an implementation of a Seven Segment Display using a
--              using Case Statement or behavioural modelling
--              Seven display consist of 7 led segments to display 0 to 6 which
--              will be named as A to G.
--              We will design a BCD to 7 Segment Decoder converts which converts
--              4 bit binary to 7 bit control signal which can be displayed on
--              7 segment display.
-----

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

-- Entity Declaration
-- This is where all the Inputs & Outputs are specified, the BCD bit
-- will be stored in BCDNos and 7 LEDs on the Segment are stored in DisplayLEDs
ENTITY Seven_Segment_Display IS
    PORT(
        BCDNos          : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
        DisplayLEDs      : OUT STD_LOGIC_VECTOR (6 DOWNTO 0)
    );
END Seven_Segment_Display;

-- Architecture Body
-- This is where we describe the behavior of the Seven Segment Display
ARCHITECTURE BEHAVIOURAL OF Seven_Segment_Display IS
BEGIN
    PROCESS(BCDNos)
    BEGIN
        CASE BCDNos IS
            WHEN "0000" =>
                DisplayLEDs <= "1111110"; ---0
            WHEN "0001" =>
                DisplayLEDs <= "0110000"; ---1
            WHEN "0010" =>
                DisplayLEDs <= "1101101"; ---2
            WHEN "0011" =>
                DisplayLEDs <= "1111001"; ---3
            WHEN "0100" =>
                DisplayLEDs <= "0010011"; ---4
            WHEN "0101" =>
                DisplayLEDs <= "1011011"; ---5
            WHEN "0110" =>
                DisplayLEDs <= "1011111"; ---6
            WHEN "0111" =>
                DisplayLEDs <= "1110000"; ---7
            WHEN "1000" =>
                DisplayLEDs <= "1111111"; ---8
            WHEN "1001" =>
                DisplayLEDs <= "1111011"; ---9
            WHEN others =>
                DisplayLEDs <= "0000000"; ---null
        END CASE;
    END PROCESS;
END BEHAVIOURAL;
```

Below is the Verilog HDL code is for Seven Segment Display

```
//Verilog code for seven segment display
//Define module

module sevSegment_display(
    input [3:0]bcd,                //Define all I/O ports
    input clk,
    output reg [6:0]seg);

//Define Always block ,this block is executed every time the
//input BCD value changes.
always @ (bcd)
begin
    case(bcd)
        4'b0000:seg<=7'b1111110;//ouput port 'seg' is assigned with seven
                                //segment equivalent of input BCD (0)
        4'b0001:seg<=7'b0110000;//ouput port 'seg' is assigned with seven
                                //segment equivalent of input BCD (1)
        4'b0010:seg<=7'b1101101;//ouput port 'seg' is assigned with seven
                                //segment equivalent of input BCD (2)
        4'b0011:seg<=7'b1111001;//ouput port 'seg' is assigned with seven
                                //segment equivalent of input BCD (3)
        4'b0100:seg<=7'b0110011;//ouput port 'seg' is assigned with seven
                                //segment equivalent of input BCD (4)
        4'b0101:seg<=7'b1011011;//ouput port 'seg' is assigned with seven
                                //segment equivalent of input BCD (5)
        4'b0110:seg<=7'b1011111;//ouput port 'seg' is assigned with seven
                                //segment equivalent of input BCD (6)
        4'b0111:seg<=7'b1110000;//ouput port 'seg' is assigned with seven
                                //segment equivalent of input BCD (7)
        4'b1000:seg<=7'b1111111;//ouput port 'seg' is assigned with seven
                                //segment equivalent of input BCD (8)
        4'b1001:seg<=7'b1111011;//ouput port 'seg' is assigned with seven
                                //segment equivalent of input BCD (9)

        default: seg<=7'b0000000;

    endcase
end
endmodule
```

Test Bench Code

Below is the VHDL Test Bench code is for Seven Segment Display

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

ENTITY Seven_Segment_Display_TB IS
END Seven_Segment_Display_TB;

ARCHITECTURE behavior OF Seven_Segment_Display_TB IS

-- Component Declaration for the Unit Under Test (UUT)
COMPONENT Seven_Segment_Display
    PORT(
        BCDNos          : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
        DisplayLEDs      : OUT STD_LOGIC_VECTOR (6 DOWNTO 0)
    );
END COMPONENT;

SIGNAL BCDNos          : STD_LOGIC_VECTOR (3 DOWNTO 0);    --INPUT BCD NO
SIGNAL DisplayLEDs     : STD_LOGIC_VECTOR (6 DOWNTO 0);    --OUTPUT LED SIGNAL

BEGIN
-- Initiate the Unit Under Test (UUT)
    uut: Seven_Segment_Display PORT MAP (
        BCDNos          => BCDNos,
        DisplayLEDs     => DisplayLEDs
    );

-- Stimulus process
    stim_proc: PROCESS
    BEGIN
        BCDNos <= "0000";
        WAIT FOR 100 ns;
        BCDNos <= "0001";
        WAIT FOR 100 ns;
        BCDNos <= "0010";
        WAIT FOR 100 ns;
        BCDNos <= "0011";
        WAIT FOR 100 ns;
        BCDNos <= "0100";
        WAIT FOR 100 ns;
        BCDNos <= "0101";
        WAIT FOR 100 ns;
        BCDNos <= "0110";
        WAIT FOR 100 ns;
        BCDNos <= "0111";
        WAIT FOR 100 ns;
        BCDNos <= "1000";
        WAIT FOR 100 ns;
        BCDNos <= "1001";
        WAIT FOR 100 ns;
        WAIT;
    END PROCESS;
END;
```

Below is the Verilog HDL Test Bench code is for Seven Segment Display

```
// Verilog code for test bench of sevensegment
//Define module
module sevSegment_tb;

reg [3:0]bcd; reg clk;           //Define the input
wire [6:0]seg;                  //Define the outputs

//Map the I/O ports with DUT
sevSegment_display uut(.bcd(bcd),.clk(clk),.seg(seg));

// Define Initial block
initial begin
    clk =0;                     //Initialise the value of clock to '0'
end

//Define Always block
always
//clock input toggles for every 10 time units
begin clk = ~clk;#10; end

//Initialize input ports with different combination of BCD data
initial begin
    bcd = 4'd5;#100;
    bcd = 4'd1;#100;
    bcd = 4'd0;#100;
    bcd = 4'd9;#100;
    bcd = 4'd8;#100;
    bcd = 4'd3;#100;
    #100;
end                               //end of initial block.
endmodule                         //end of module.
```

Implementing on quartus II

For in-depth guide for creating a **New Project** follow **Quick Start Guide For Quartus Prime Lite Software**. Both Verilog HDL and VHDL code is provided in this tutorial, hence continue the tutorial by selecting only one language to avoid confusion.

1. Create a **New Project** with the following settings

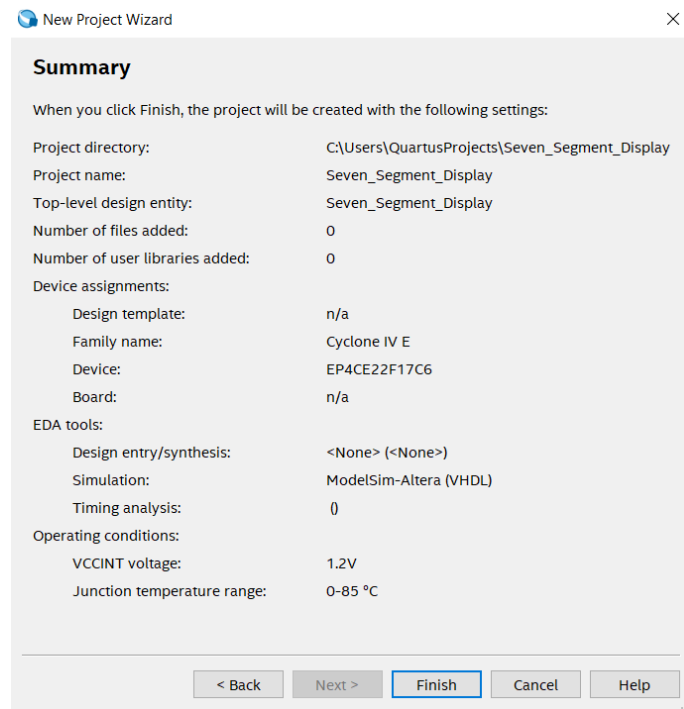


Figure 5: Project Setting

2. Create a **New VHDL** file. Type the code for Seven Segment Display mention above in the code section(code on page 2) in this file.

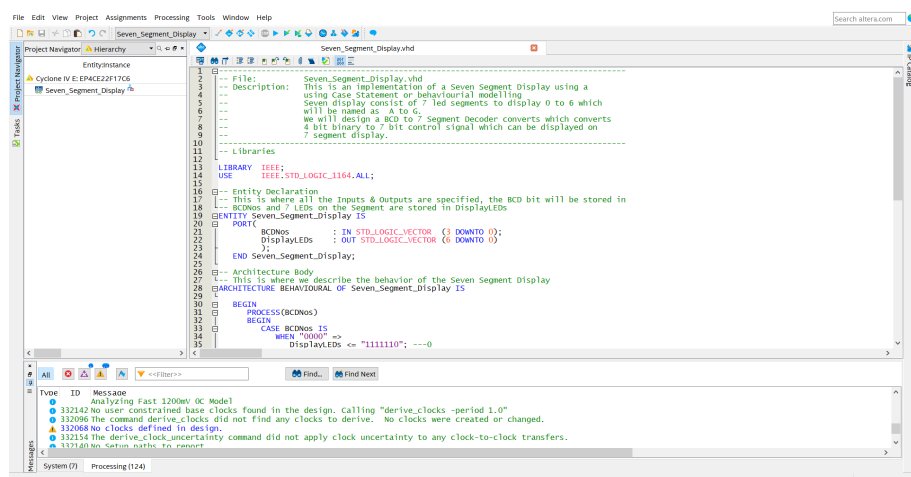


Figure 6: Create a VHDL file

- Go to **File**→**Save as**. Enter the Name of file (it should be same as module/entity name), Enter correct file extension, for VHDL it is **.vhd** and for Verilog HDL it is **.v** and then Click on **Save**

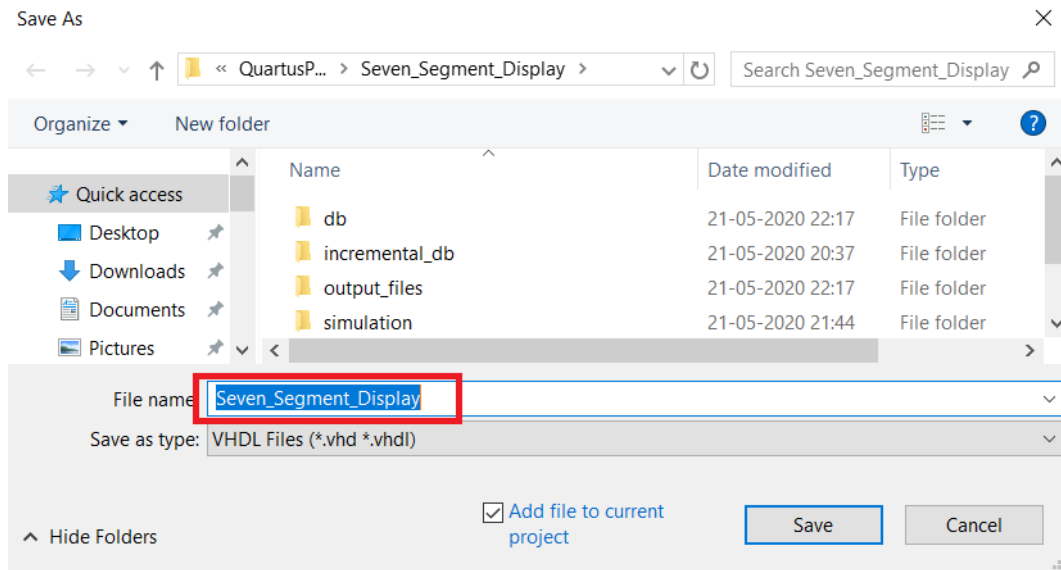


Figure 7: Save the file

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

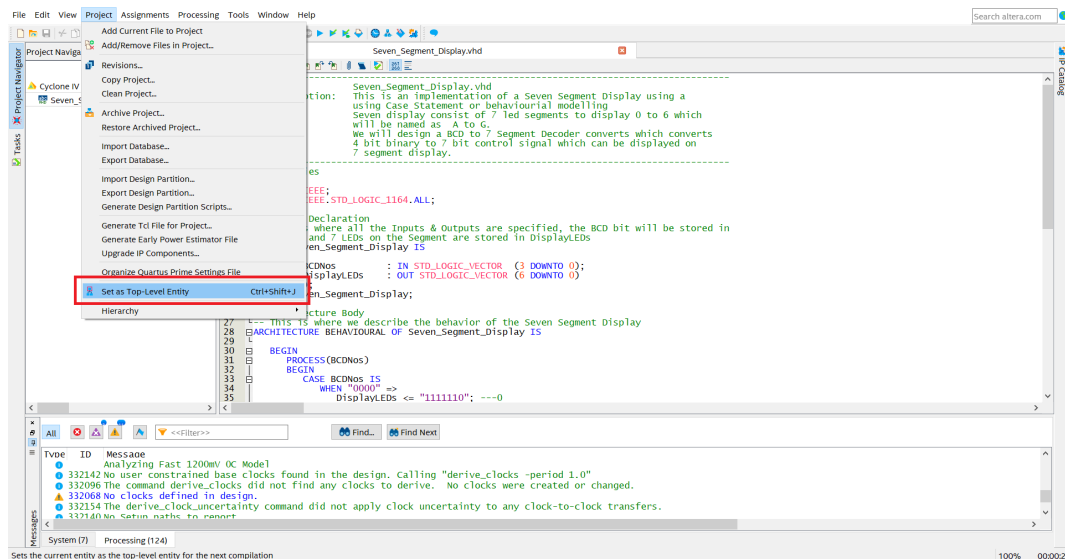


Figure 8: Assign Top level Entity

5. Goto **Processing**→**Start Compilation**. It takes sometime to complete. You can verify whether all the files are compiled successfully by checking the highlighted tabs i.e. **Messages** and **Tasks** tab.

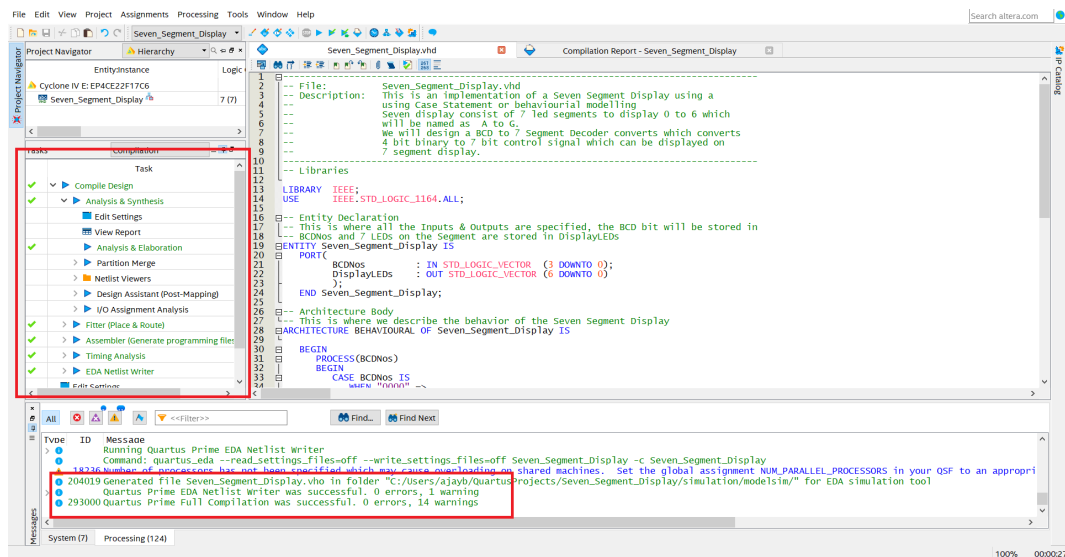


Figure 9: Verification

RTL Circuit of the implemented design

The Figure shown in step 2 below shows the RTL design of the Seven Segment Display circuit that we design using the above files. Here we can 7 select modules i.e 2:1 Multiplier, and one output and input block. The select lines for the Multiplier are from the 4-Bit DataBus input and the 7-Bit DataBus is the data line. If the input and the select lines matches for a particular Multiplier the the its 7 bit data is passed to the output.

Steps to get RTL circuit.

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

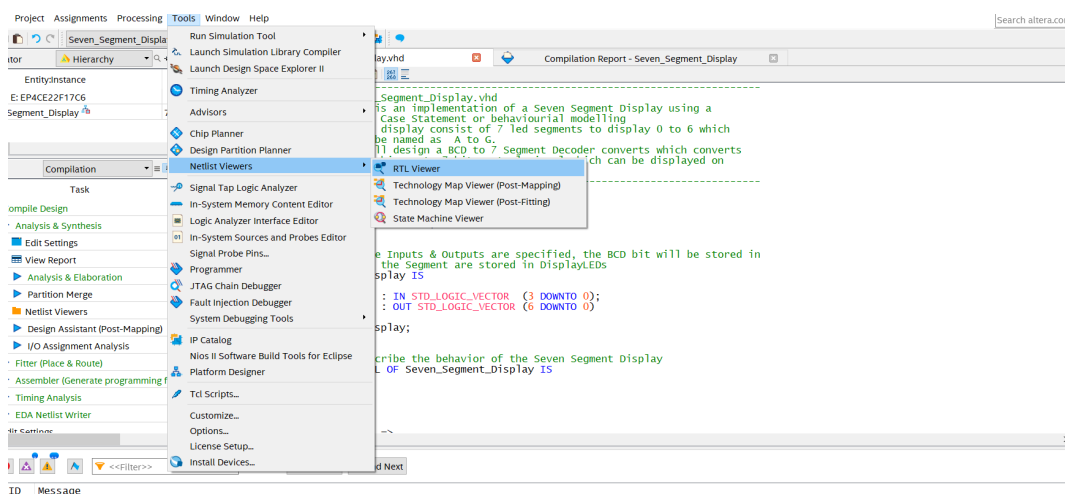


Figure 10: Start RTL Viewer

- The below figure shows the equivalent RTL circuit of BCD decoder (4:7) or the design of Seven Segment Display using behavioral modelling.

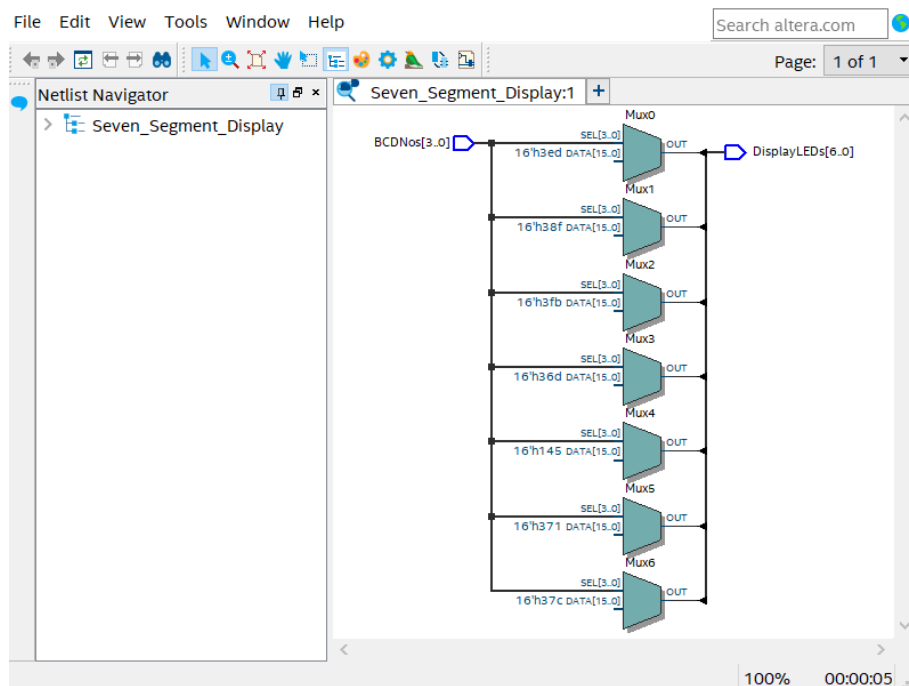


Figure 11: RTL Design

Pin Assignment

For in-depth guide for **Pin Assignment** a **New Project** follow **Quick Start Guide For Quartus Prime Lite Software Section 4.2**. This guide also contains a detailed pin description for DE0-Nano Board, so that you can select pin as per your requirement

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

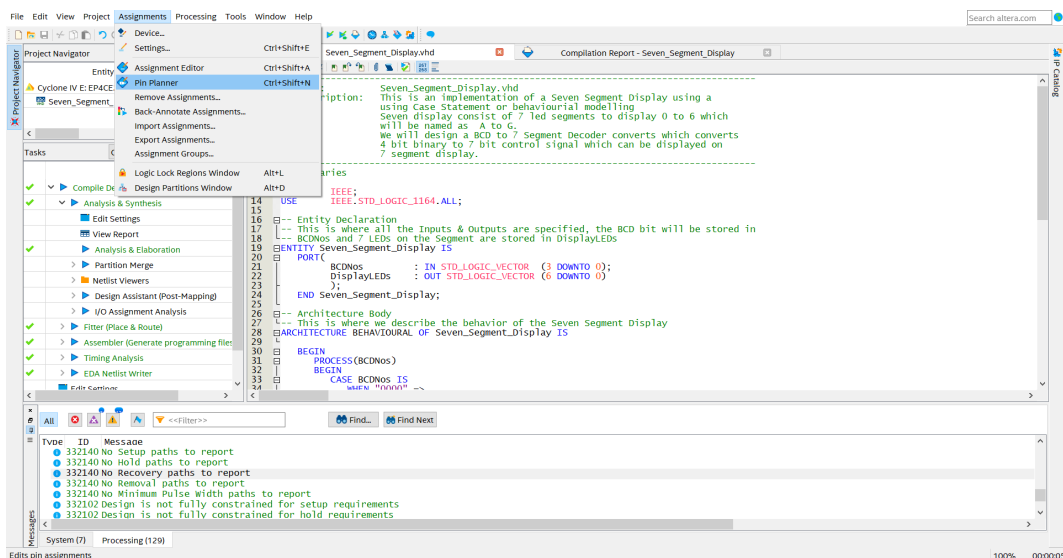


Figure 12: Start Pin Planner

2. 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 . If you a Seven Segment Display Module then you can use the GPIOs pins of the board and connect them to it. For this demonstration we will assign the **Output** to the LEDs[0 to 6] And assign all the **Inputs**(BCD) as the Slide Switch[0 to 3]

Pin Name for sliding Switches		
Signal Name	FPGA PinNo.	Description
DW[0]	PIN_M1	BCDnos[0]
DW[1]	PIN_T8	BCDnos[1]
DW[2]	PIN_B9	BCDnos[3]
DW[3]	PIN_M15	BCDnos[4]
Pin Name for LEDs		
Signal Name	FPGA PinNo.	Description
LED[0]	PIN_A15	DisplayLEDs[0]
LED[1]	PIN_A13	DisplayLEDs[1]
LED[2]	PIN_B13	DisplayLEDs[2]
LED[3]	PIN_A11	DisplayLEDs[3]
LED[4]	PIN_D1	DisplayLEDs[4]
LED[5]	PIN_F3	DisplayLEDs[5]
LED[6]	PIN_B1	DisplayLEDs[6]

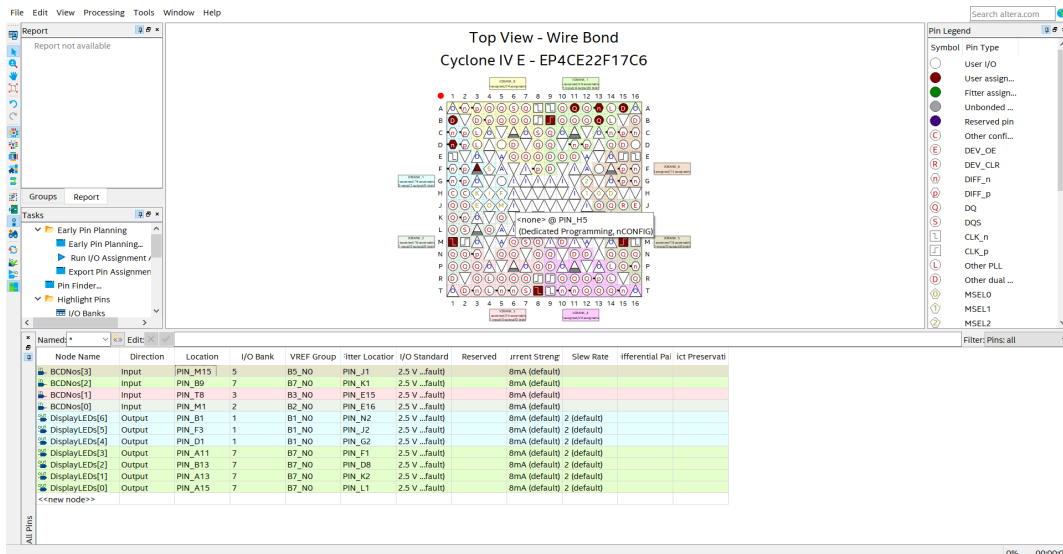


Figure 13: Pin assignment

3. 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.

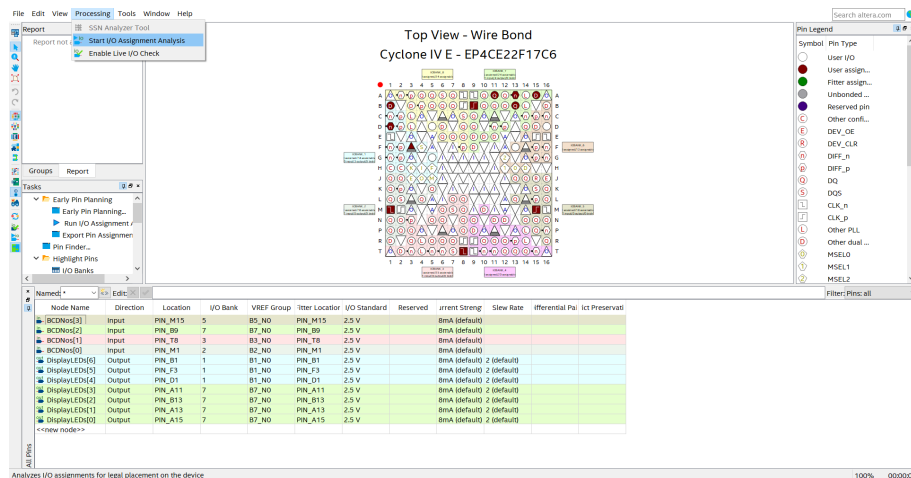


Figure 14: Pin Analysis

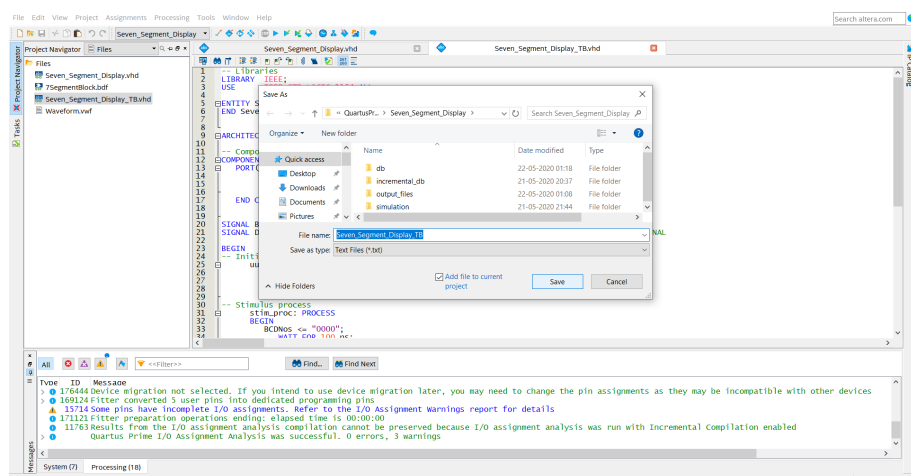
Downloading the code to DE0 Nano FPGA Board

Refer **Quick Start Guide to Quartus and ModelSim Software Section 4.3**. The procedure to download the code to your FPGA board remains same, just make sure to **Compile** the correct file i.e **Seven_Segment_Display** which needs to be set as **Top Level Entity** while generating the .sof file.

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 test bench code below, make sure to follow only one language in this entire tutorial.

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



2. Go to **Assignments**→**Settings**.
3. Navigate to **Simulation** under **EDA Tool Settings**. Set the language as VHDL. Select **Compile Test Bench** and then click on **Test Benches**. You can find a detailed tutorial for setting a Test Bench file for simulation in the **Quick Start Guide to Quartus and ModelSim Software Document** in **Section 5.2**
4. Click on **Processing**→**Start Compilation** This process will check for any illegal statement occurred in any of the files and give the error messages in the Messages section.

Without TestBench

We can simulate our design without using a testbench file by using creating a **University Program VWF** file. You can find detailed steps to create and add Input and Output nodes in this files in the **Quick Start Guide to Quartus and ModelSim Software Document** in **Section 5.1**

1. Assign clock pulse to the inputs BCDNos[0 to 3] as per the given table and run the simulation

Clock Pulse For Inputs	
Input	Clock
DisplayLEDs[0]	50ns
DisplayLEDs[1]	100ns
DisplayLEDs[2]	200ns
DisplayLEDs[3]	400ns

2. After running the simulation you can see the following results, also you can notice for BCDNos "1010" to "1111" the output is zero because we mention the condition for input greater than 9 the output is null or zero.

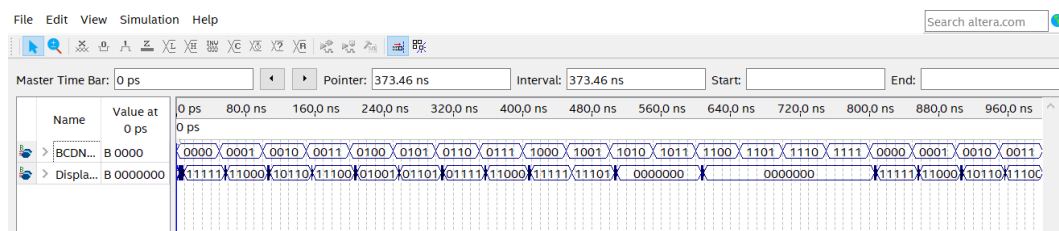


Figure 16: Output Waveform

Expanded form of above image

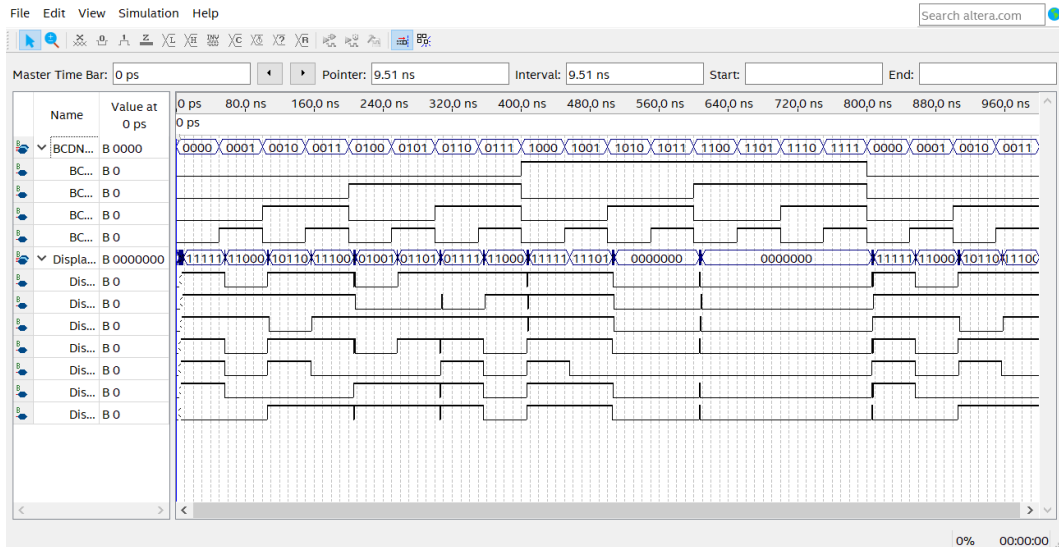


Figure 17: Expanded Output Waveform

Functional Simulation using NativeLink Feature

1. Then Go to **Tools**→**Run Simulation Tool**→**RTL Simulation** to automatically run the EDA simulator(ModelSim-Altera) and to compile all necessary design files.

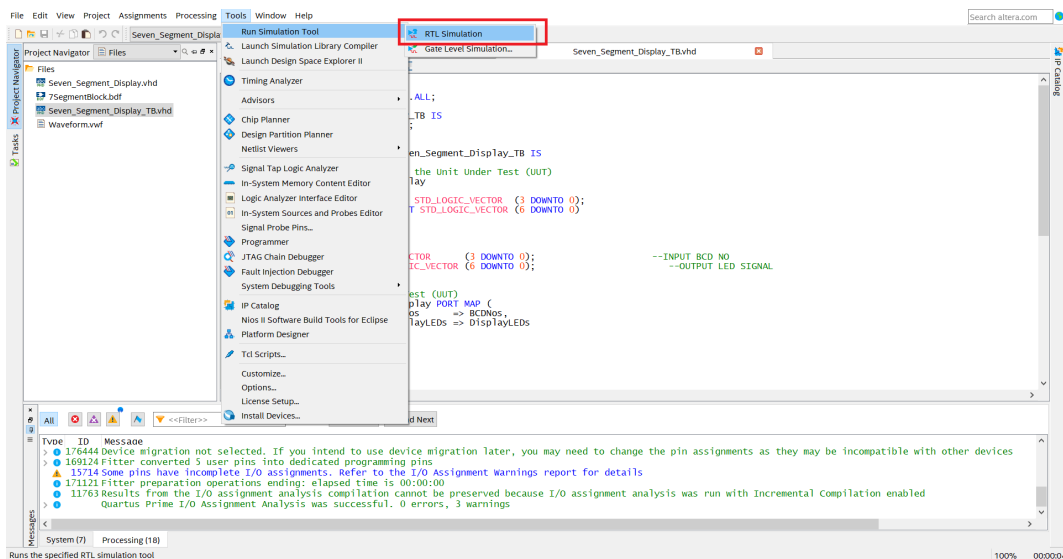


Figure 18: Start RTL Simulation

2. Finally ModelSim-Altera tool opens,
Add the I/O pins as shown in the below figure. Double click on the testbench file under 'Library' section, this adds the signals under 'Object' section.
Start the simulation and add the test bench file. Then select all the signals (A , B and C) and add it to 'wave' section.

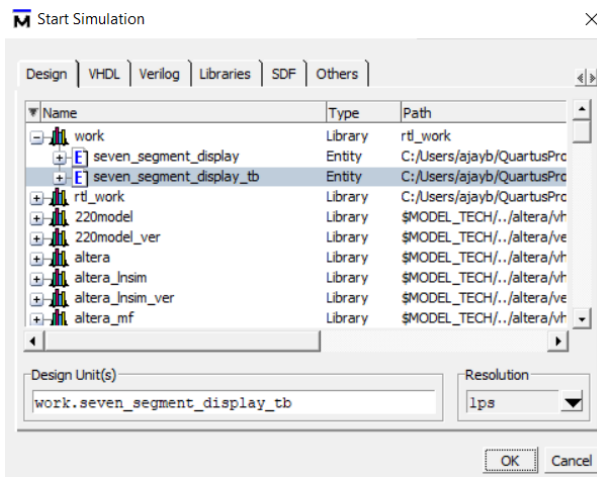


Figure 19: Select TestBench file

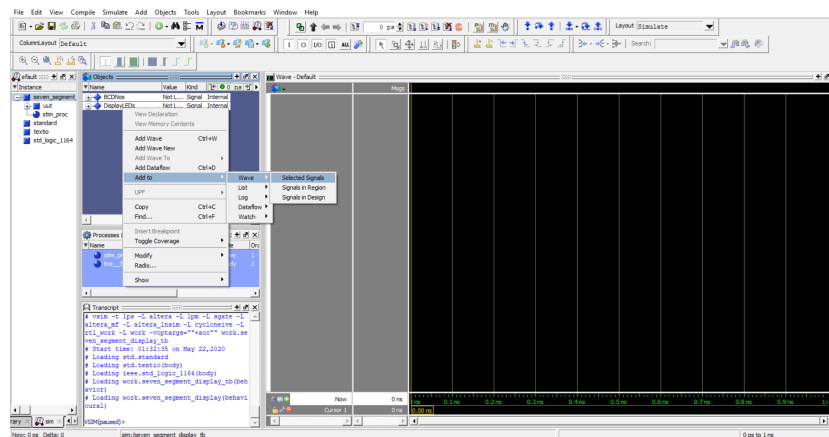


Figure 20: Add I/O Signals

3. Start the simulation by clicking on the run all button in tool bar

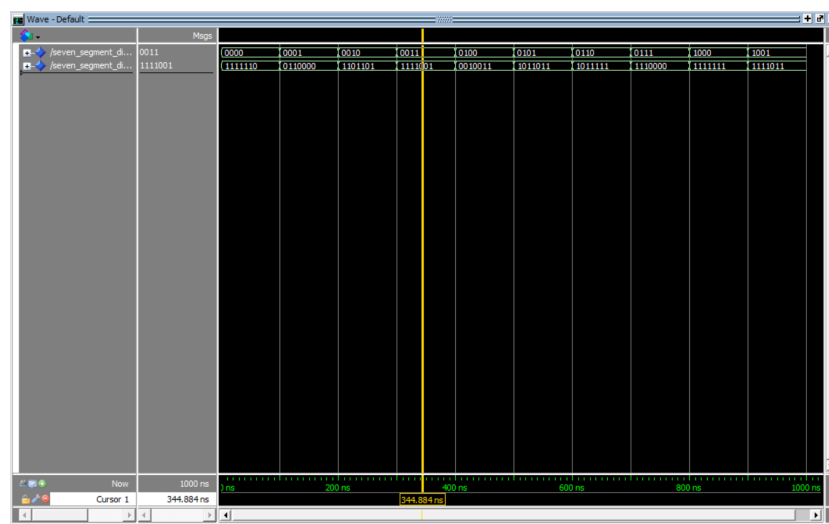


Figure 21: Simulation

4. **Navigating the simulation** At this point you should have successfully run the simulation, but the waveform window is rather small and hard to see. We can move around in the simulation and see the value of the signals. Look for the cursor, a yellow vertical line in the waveform viewer, with the time in yellow at the bottom. You can use this line to move left or right in the waveform viewer and also zoom-in and zoom-out.

Testing the Design

Simulation waveform of our Design

The Result shown below can be verified by comparing it with the Truth Table provided in page 2

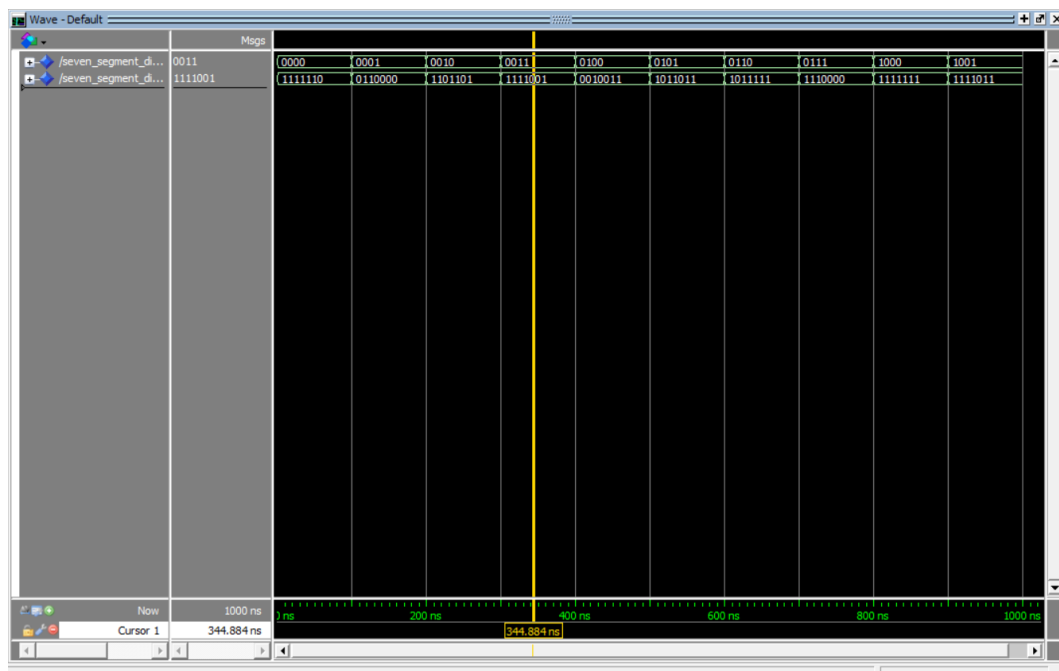


Figure 22: Final Output Waveform

You can perform the same experiment using **Verilog HDL** language by using the code and testbench provided in code section of this document.