Chapter 2: Hardware Design Flow Using Verilog in Quartus II

2.1 Introduction to Quartus II System Development Software

This chapter is an introduction to the Quartus II software that will be used for analysis and synthesis of the DE2-115 Development and Education Board. Throughout this chapter hardware description languages like Verilog will be used for coding. The Altera Quartus II design software provides a complete, multiplatform design environment for system-on-a-programmable-chip (SOPC) designs. Also an example will be implemented in a tutorial using the hardware description language (Verilog) and the DE2-115. Below are some suggested readings before going into the next section.

Quartus II Development Software Reading Resources:

(In suggested chronological reading/watching order)

1) Introduction to Quartus II Software

→ Version 11.0 (Latest):

http://www.altera.com/literature/manual/quartus2_introduction.pdf

- *NOTE:* The link to the newer version of the later version (11.0) provides a very brief overview, whereas the older version (listed below) gives more in depth information.

→ Version 10.0:

http://www.altera.com/literature/manual/archives/intro_to_quartus2.pdf

- *focus:* Emphasis is placed on the following sections, although a greater knowledge base is achieved by reviewing the entire document:

    a) Design Flow- Introduction (Page No. 11), Graphical User Interface Design Flow (Page No. 12)

b) Design Entry (Page No. 29) Introduction, Creating a Project(Page No. 30), Creating a Design(Page No. 31), *later this document can be used for a specific method of design entry (like Verilog, Block Diagram, VHDL, etc.)*

c) Programming & Configuration (Page No. 93) Introduction, Creating and Using Programming Files

2) Using Verilog for Quartus II Design:

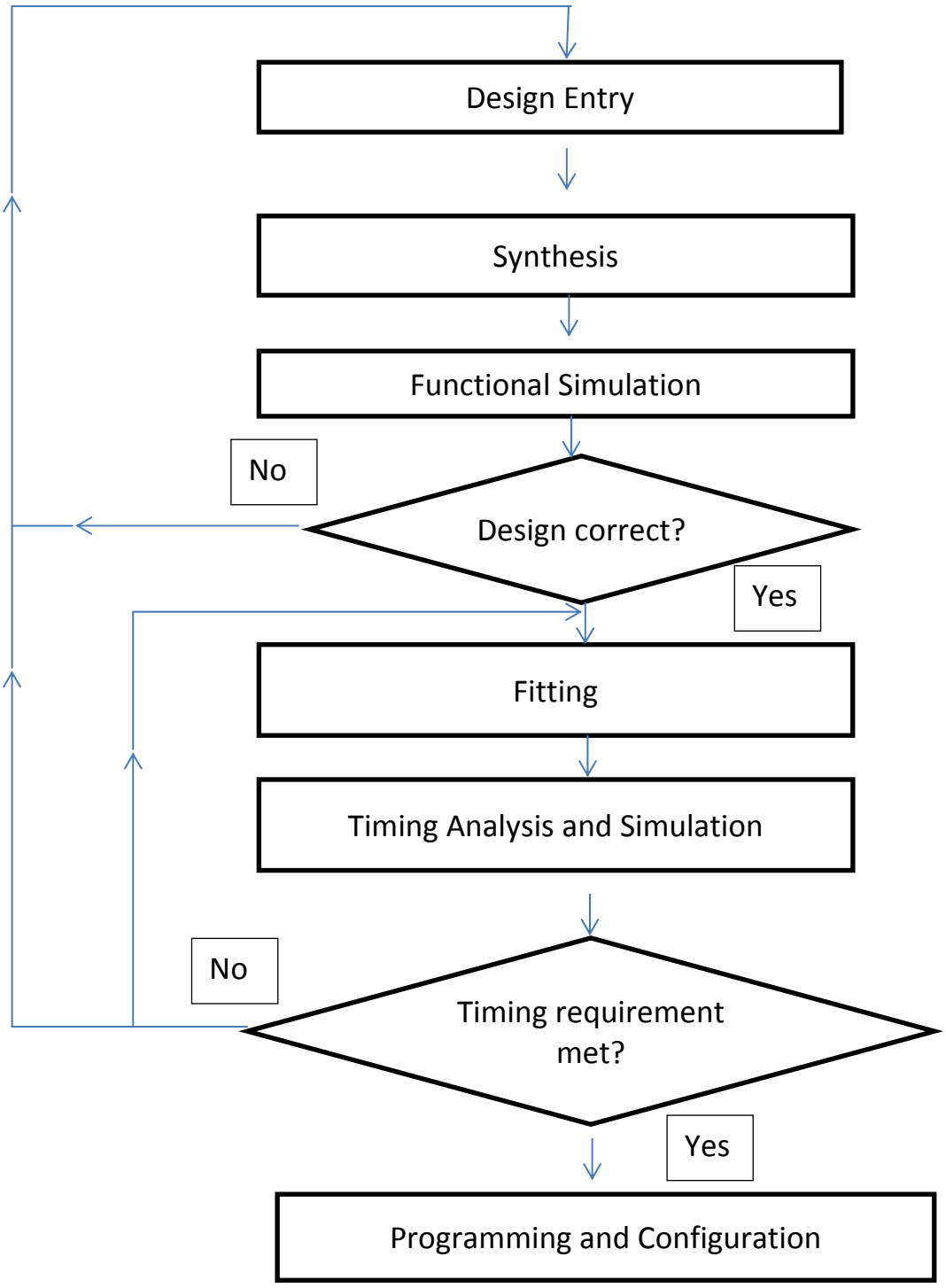
<*system cd*>\DE2_115_tutorials\tut_quartus_intro_verilog.pdf

- *focus*: This tutorial guides through the simulation process so that the project can be implemented without needing access to the DE2-115.(familiar with quartus and Verilog) (PG No 1-21)

3) Quartus II Handbook: http://www.altera.com/literature/hb/qts/quartusii_handbook.pdf

- *NOTE:* This resource is in depth and is only necessary to briefly overview the material in order to know where information can be found on an *as needed* basis.

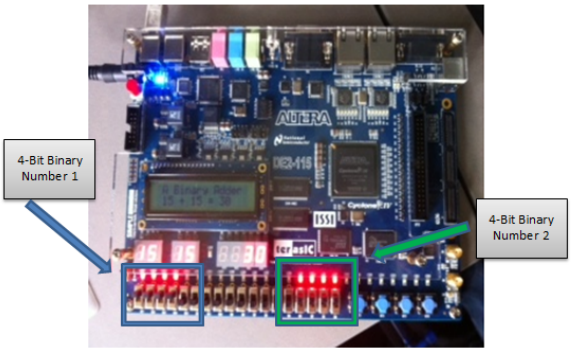
2.2 Design Flow (Hardware Only)



2.3 Binary Adder Example

Now that you are getting familiar with Quartus II and the DE2-11 a tutorial discussing the basic steps for using Quartus II is discussed below.

In this example, the components from the DE2-115 Board that will be used are:



→ 7 Segment Hex Display,

→ Switches,

→ 8 Red LEDs, and the

→ LCD Display

As shown in the picture above the switches and LED's are synchronized and represent a 4 bit binary number. The values of these binary numbers are displayed on the 7 segment display and LCD. Moreover the addition of these two binary numbers is also displayed on the seven segment display and LCD.

*To learn more in detail about the 7 Segment Hex Display also there is a short video about 7 segment display () and LCD refer to the last 5 pages of this example

The Binary Adder tutorial teaches how to

- Connect the conputer with the DE2-115.
- Create a new project using Quartus II.
- Create a Verilog file.
- Put I/O pin locations in the assignment editor.
- Synthesize your design.
- Use system builder.

1. The youtube video for the complete procedure can be accessed from the link given below:

   http://www.youtube.com/watch?v=rpFxGuRB0xk

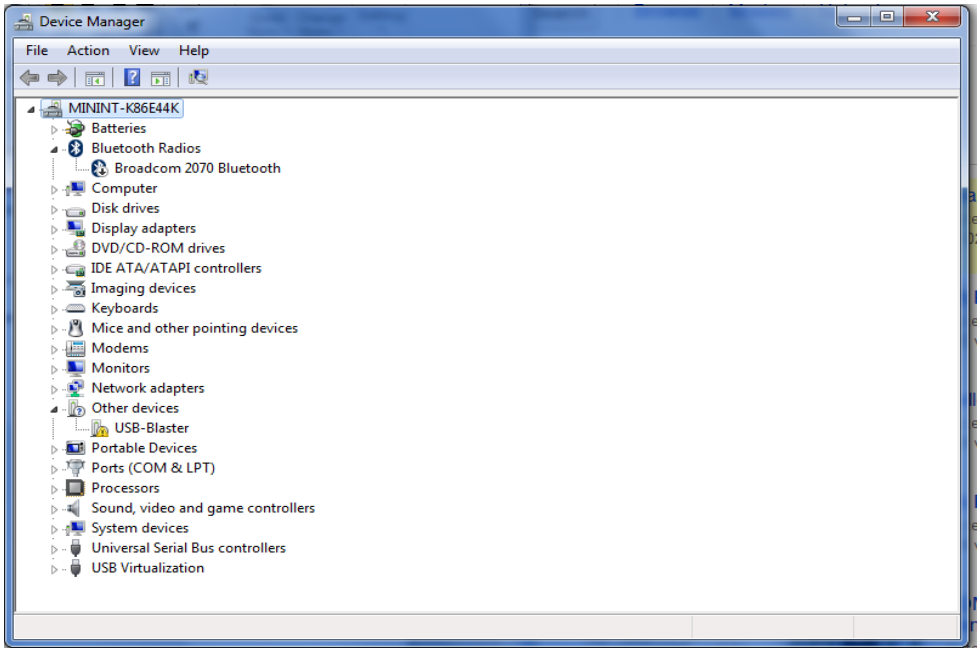
2. The example can also be implemented by using the written instructions given below:

Binary Adder Tutorial

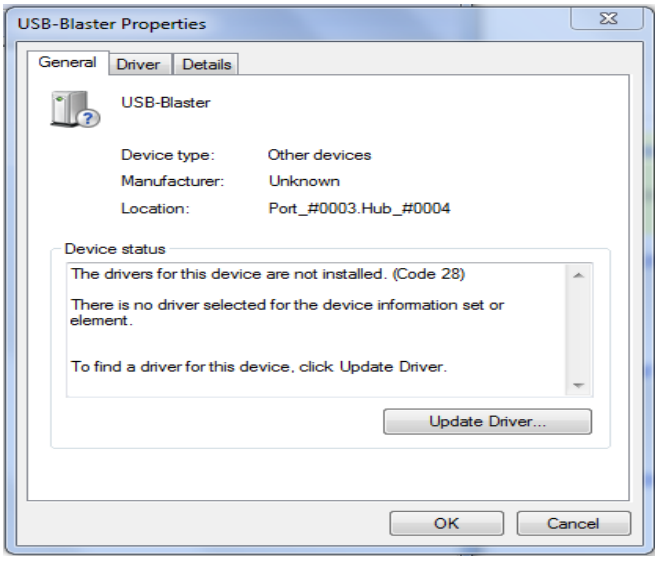
Step 1: Install the USB driver for the FPGA development board. This step will only be done for the first time the FPGA board is used.

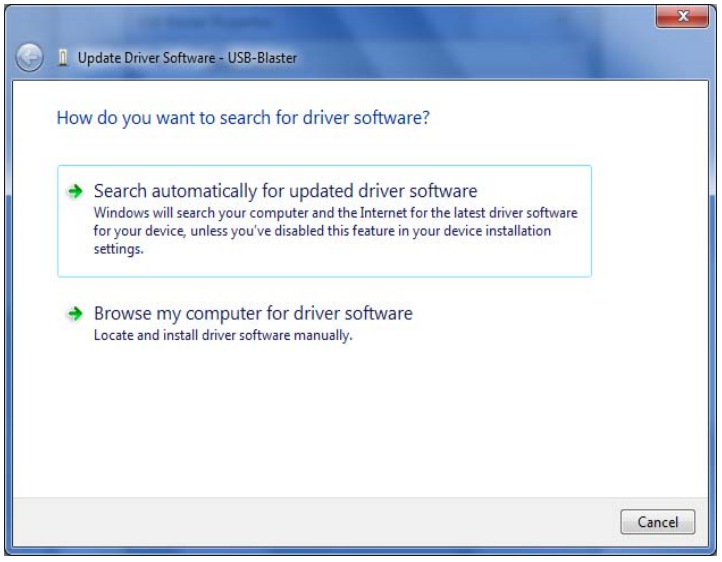
a) On the FPGA board, connect the power plug to an outlet. Connect the USB cable from your computer to the FPGA board in port J9 (closest to the power outlet).

b) Open the start Menu and Search Windows for “Device Manager”-> Scroll down to “Other Devices”-> A new window called “USB Blaster Properties” will open.

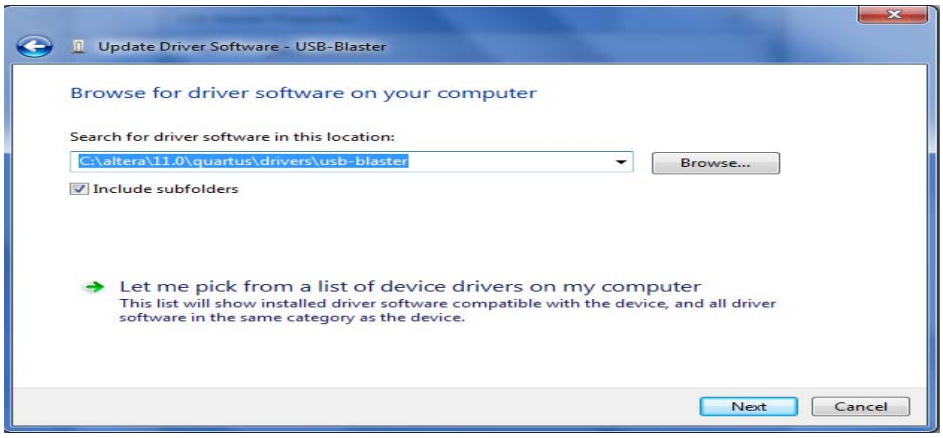


c) Under the tab “Driver” select “Update Driver” -> A new window will pop up and you’ll select “Browse my computer for driver software





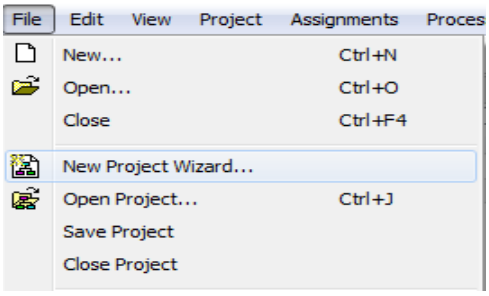
d) In the field “Search for Drivers in this location” browse your computer to create the following path: C: - > Altera -> 11.0 -> Quartus -> Drivers -> USB Blaster then select “Browse”



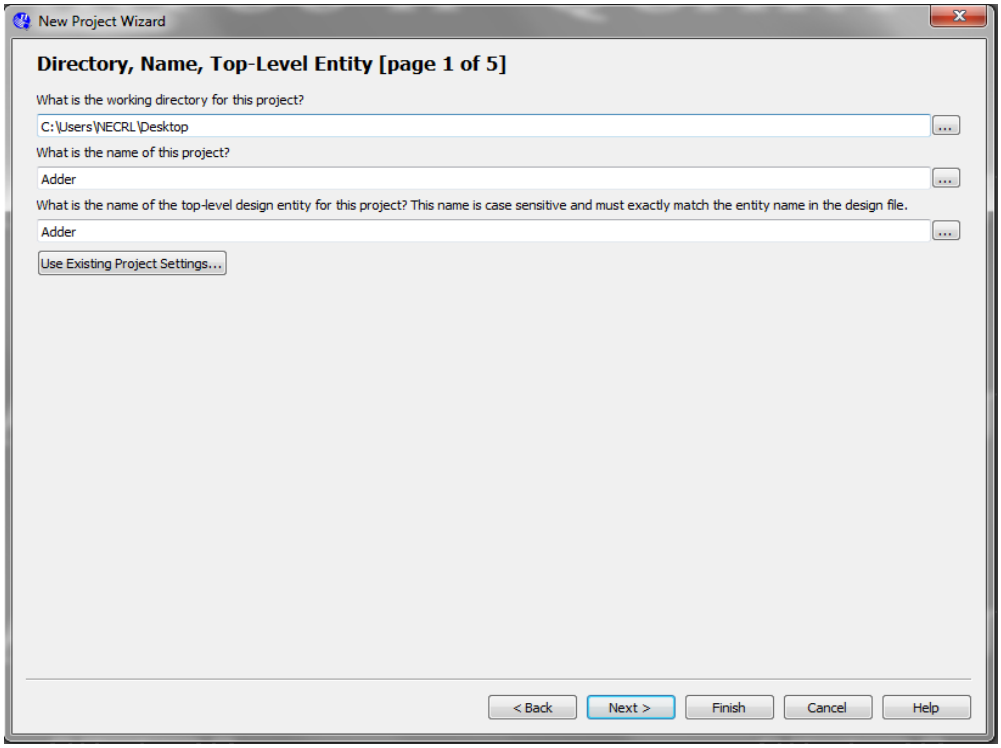
e) You may need to click “allow” to complete the process.

Step 2: Open the Quartus II software

a) Select “Create New Project Wizard”



b) In the first step (1 of 5) you will need to create a directory for your project and name your new project.



c) In step 2 of 5, you will add any previously created files to your project. Make sure to go to the lower portion of screen and select “Add User Libraries”.

   i. A new window opens. Go to “Global Library Name” and to the right of Global libraries click on “…”

   ii. Go to “Computer” then go to the “C drive” (where the Altera folder is located)

   iii. Go to on the Altera folder then go to the “quartus” folder

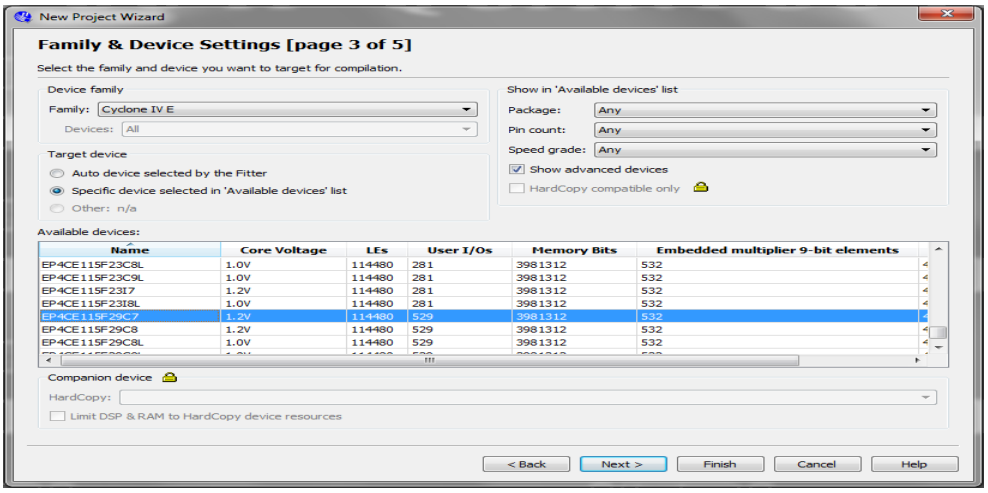
   iv. Go to on the “libraries” folder

   v. Add the “MegaFunctions” library and click “Select folder” then “OK”

d) In step 3 of 5, “Family & Device Settings” you will adjust the family and device you want to target for compilation.

i. Device family is Cyclone IV E.

ii. Target device is “Specific” and select our device from “Available Devices”→EP4CE115F29C7. Click “Next”

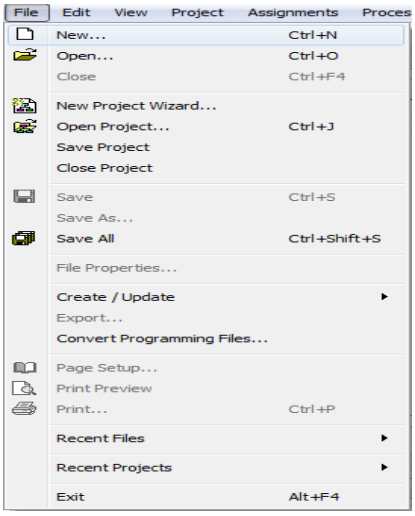


e) In step 4 of 5, EDA Tool Settings do not make any adjustments. Click “Next”

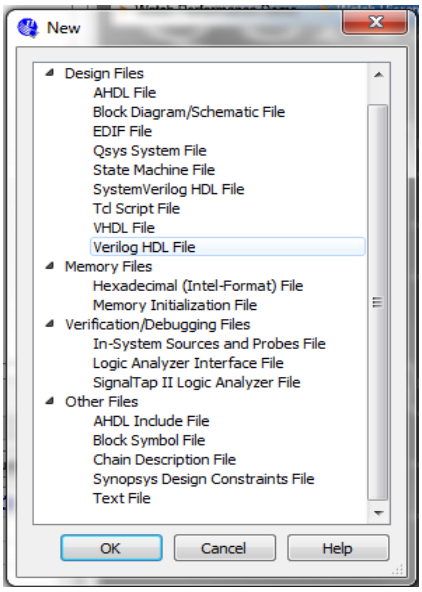
f) In step 5 of 5, Summary, click “Finish to create your new project.

Step 3: You will need to create a new Verilog file for your project.

a) Under “File” select “New”



b) Under “Design Files” select “Verilog HDL File”



c) Click “OK”

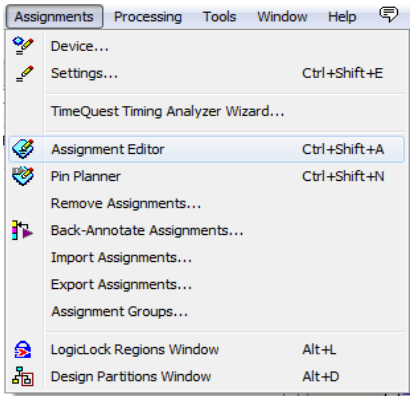
d) A new Verilog file will open. An asterisk will appear near the file name whenever unsaved changes have been made.

~ This tutorial focuses on Verilog (a hardware description language), In order to program the Altera DE2-115

Step 4: Copy the Verilog Code from the file Binary_Adder.txt file into Quartus II

Note: Binary_Adder.txt is located in the Codes folder

Step 5: You will use the DE2-115 manual to determine ports and PIN assignments. Assignments->assignment editor (Ctrl+Shift+A) set all components to their appropriate locations and voltage



	tatu	From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag
1	✓		CLOCK_50	Location	PIN_Y2	Yes			
2	✓		CLOCK_50	I/O Standard	3.3-V LVTTL	Yes			
3	✓		LEDR[0]	Location	PIN_G19	Yes			
4	✓		LEDR[0]	I/O Standard	2.5 V	Yes			
5	✓		LEDR[1]	Location	PIN_F19	Yes			
6	✓		LEDR[1]	I/O Standard	2.5 V	Yes			
7	✓		LEDR[2]	Location	PIN_E19	Yes			
8	✓		LEDR[2]	I/O Standard	2.5 V	Yes			
9	✓		LEDR[3]	Location	PIN_F21	Yes			
10	✓		LEDR[3]	I/O Standard	2.5 V	Yes			
11	✓		KEY[0]	Location	PIN_M23	Yes			
12	✓		KEY[0]	I/O Standard	3.3-V LVTTL	Yes			
13	✓		KEY[1]	Location	PIN_M21	Yes			
14	✓		KEY[1]	I/O Standard	3.3-V LVTTL	Yes			
15	✓		KEY[2]	Location	PIN_N21	Yes			
16	✓		KEY[2]	I/O Standard	3.3-V LVTTL	Yes			
17	✓		KEY[3]	Location	PIN_R24	Yes			
18	✓		KEY[3]	I/O Standard	3.3-V LVTTL	Yes			
19	✓		SW[0]	Location	PIN_AB28	Yes			
20	✓		SW[0]	I/O Standard	3.3-V LVTTL	Yes			
21	✓		SW[1]	Location	PIN_AC28	Yes			
22	✓		SW[1]	I/O Standard	3.3-V LVTTL	Yes			
23	✓		SW[2]	Location	PIN_AC27	Yes			
24	✓		SW[2]	I/O Standard	3.3-V LVTTL	Yes			
25	✓		SW[3]	Location	PIN_AD27	Yes			
26	✓		SW[3]	I/O Standard	3.3-V LVTTL	Yes			
27	✓		HEX0[0]	Location	PIN_G18	Yes			
28	✓		HEX0[0]	I/O Standard	2.5 V	Yes			
29	✓		HEX0[1]	Location	PIN_F22	Yes			
30	✓		HEX0[1]	I/O Standard	2.5 V	Yes			
31	✓		HEX0[2]	Location	PIN_E17	Yes			
32	✓		HEX0[2]	I/O Standard	2.5 V	Yes			
33	✓		HEX0[3]	Location	PIN_L26	Yes			
34	✓		HEX0[3]	I/O Standard	3.3-V LVTTL	Yes			
35	✓		HEX0[4]	Location	PIN_L25	Yes			
36	✓		HEX0[4]	I/O Standard	3.3-V LVTTL	Yes			
37	✓		HEX0[5]	Location	PIN_J22	Yes			
38	✓		HEX0[5]	I/O Standard	3.3-V LVTTL	Yes			
39	✓		HEX0[6]	Location	PIN_H22	Yes			

40	✓		HEX0[6]	I/O Standard	3.3-V LVTTL	Yes			
41	✓		HEX1[0]	Location	PIN_M24	Yes			
42	✓		HEX1[0]	I/O Standard	3.3-V LVTTL	Yes			
43	✓		HEX1[1]	Location	PIN_Y22	Yes			
44	✓		HEX1[1]	I/O Standard	3.3-V LVTTL	Yes			
45	✓		HEX1[2]	Location	PIN_W21	Yes			
46	✓		HEX1[2]	I/O Standard	3.3-V LVTTL	Yes			
47	✓		HEX1[3]	Location	PIN_W22	Yes			
48	✓		HEX1[3]	I/O Standard	3.3-V LVTTL	Yes			
49	✓		HEX1[4]	Location	PIN_W25	Yes			
50	✓		HEX1[4]	I/O Standard	3.3-V LVTTL	Yes			
51	✓		HEX1[5]	Location	PIN_U23	Yes			
52	✓		[5]	I/O Standard	3.3-V LVTTL	Yes			
53	✓		HEX1[6]	Location	PIN_U24	Yes			
54	✓		HEX1[6]	I/O Standard	3.3-V LVTTL	Yes			
55	✓		HEX2[0]	Location	PIN_AA25	Yes			
56	✓		HEX2[0]	I/O Standard	3.3-V LVTTL	Yes			
57	✓		HEX2[1]	Location	PIN_AA26	Yes			
58	✓		HEX2[1]	I/O Standard	3.3-V LVTTL	Yes			
59	✓		HEX2[2]	Location	PIN_Y25	Yes			
60	✓		HEX2[2]	I/O Standard	3.3-V LVTTL	Yes			
61	✓		HEX2[3]	Location	PIN_W26	Yes			
62	✓		HEX2[3]	I/O Standard	3.3-V LVTTL	Yes			
63	✓		HEX2[4]	Location	PIN_Y26	Yes			
64	✓		HEX2[4]	I/O Standard	3.3-V LVTTL	Yes			
65	✓		HEX2[5]	Location	PIN_W27	Yes			
66	✓		HEX2[5]	I/O Standard	3.3-V LVTTL	Yes			
67	✓		HEX2[6]	Location	PIN_W28	Yes			
68	✓		HEX2[6]	I/O Standard	3.3-V LVTTL	Yes			
69	✓		HEX3[0]	Location	PIN_V21	Yes			
70	✓		HEX3[0]	I/O Standard	3.3-V LVTTL	Yes			
71	✓		HEX3[1]	Location	PIN_U21	Yes			
72	✓		HEX3[1]	I/O Standard	3.3-V LVTTL	Yes			
73	✓		HEX3[2]	Location	PIN_AB20	Yes			
74	✓		HEX3[2]	I/O Standard	3.3-V LVTTL	Yes			
75	✓		HEX3[3]	Location	PIN_AA21	Yes			
76	✓		HEX3[3]	I/O Standard	3.3-V LVTTL	Yes			
77	✓		HEX3[4]	Location	PIN_AD24	Yes			
78	✓		HEX3[4]	I/O Standard	3.3-V LVTTL	Yes			

Status: Ok

118	✓		HEX6[3]	I/O Standard	3.3-V LVTTL	Yes			
119	✓		HEX6[4]	Location	PIN_AB15	Yes			
120	✓		HEX6[4]	I/O Standard	3.3-V LVTTL	Yes			
121	✓		HEX6[5]	Location	PIN_AA15	Yes			
122	✓		HEX6[5]	I/O Standard	3.3-V LVTTL	Yes			
123	✓		HEX6[6]	Location	PIN_AC17	Yes			
124	✓		HEX6[6]	I/O Standard	3.3-V LVTTL	Yes			
125	✓		HEX7[0]	Location	PIN_AD17	Yes			
126	✓		HEX7[0]	I/O Standard	3.3-V LVTTL	Yes			
127	✓		HEX7[1]	Location	PIN_AE17	Yes			
128	✓		HEX7[1]	I/O Standard	3.3-V LVTTL	Yes			
129	✓		HEX7[2]	Location	PIN_AG17	Yes			
130	✓		HEX7[2]	I/O Standard	3.3-V LVTTL	Yes			
131	✓		HEX7[3]	Location	PIN_AH17	Yes			
132	✓		HEX7[3]	I/O Standard	3.3-V LVTTL	Yes			
133	✓		HEX7[4]	Location	PIN_AF17	Yes			
134	✓		HEX7[4]	I/O Standard	3.3-V LVTTL	Yes			
135	✓		HEX7[5]	Location	PIN_AG18				
136	✓		HEX7[5]	I/O Standard	3.3-V LVTTL	3.3-V LVTTL Status: Ok			
137	✓		HEX7[6]	Location	PIN_AA14	Yes			
138	✓		HEX7[6]	I/O Standard	3.3-V LVTTL	Yes			
139	✓		LEDR2[0]	Location	PIN_F15	Yes			
140	✓		LEDR2[0]	I/O Standard	2.5 V	Yes			
141	✓		LEDR2[1]	Location	PIN_G15	Yes			
142	✓		LEDR2[1]	I/O Standard	2.5 V	Yes			
143	✓		LEDR2[2]	Location	PIN_G16	Yes			
144	✓		LEDR2[2]	I/O Standard	2.5 V	Yes			
145	✓		LEDR2[3]	Location	PIN_H15	Yes			
146	✓		LEDR2[3]	I/O Standard	2.5 V	Yes			
147	✓		SW2[0]	Location	PIN_AA23	Yes			
148	✓		SW2[0]	I/O Standard	3.3-V LVTTL	Yes			
149	✓		SW2[1]	Location	PIN_AA22	Yes			
150	✓		SW2[1]	I/O Standard	3.3-V LVTTL	Yes			
151	✓		SW2[2]	Location	PIN_Y24	Yes			
152	✓		SW2[2]	I/O Standard	3.3-V LVTTL	Yes			
153	✓		SW2[3]	Location	PIN_Y23	Yes			
154	✓		SW2[3]	I/O Standard	3.3-V LVTTL	Yes			

Step 6: For any project it is required to create pin assignment from the DE2-115 manual.

a) Under “Assignments” select “Assignment Editor”

b) Add each port under “Assignment Name” –each port will need two assignments:

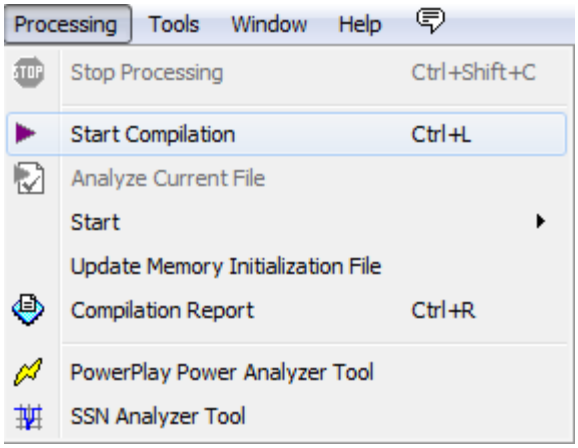
i. PIN location

ii. I/O requirement.

Note: This process is very lengthy and in the future can be bypassed using “System Builder”(PG No. 15).

Step 7: When the Verilog code is finished, and all assignments are done, you will be ready to compile your design and program the device.

Step 8: At the top of the screen, select the "Play" button to begin the automatic compilation process. Watch in the lower left screen as the compilation process occurs. This may take several minutes.



(Step 7)

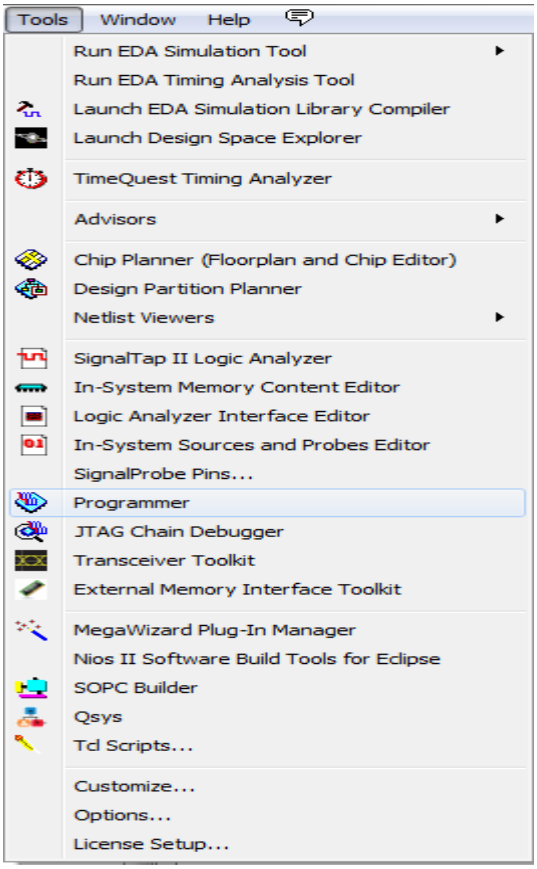
Step 9: When it has compiled, double click on "Program Device".

a) Push the large red button on the FPGA board to turn on the power.

b) Programmer will open, and at the top left "USB Blaster" will appear. If it does not, click on "Hardware Setup". Select "USB Blaster" and click ok.

c) When "USB Blaster" appears next to "Hardware Setup" select "Start" and watch the upper right corner as the design is implemented.

d) When the "Progress" bar has reach 100% you may test your design on the FPGA board.

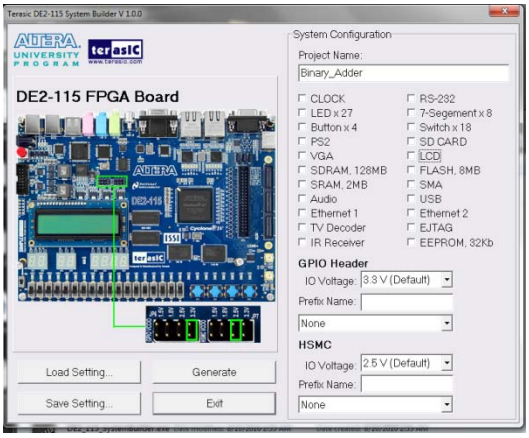

Tools
Window
Help
Run EDA Simulation Tool
Run EDA Timing Analysis Tool
Launch EDA Simulation Library Compiler
Launch Design Space Explorer
TimeQuest Timing Analyzer
Advisors
Chip Planner (Floorplan and Chip Editor)
Design Partition Planner
Netlist Viewers
SignalTap II Logic Analyzer
In-System Memory Content Editor
Logic Analyzer Interface Editor
In-System Sources and Probes Editor
SignalProbe Pins...
Programmer
JTAG Chain Debugger
Transceiver Toolkit
External Memory Interface Toolkit
MegaWizard Plug-In Manager
Nios II Software Build Tools for Eclipse
SOPC Builder
Qsys
Tcl Scripts...
Customize...
Options...
License Setup...

System Builder

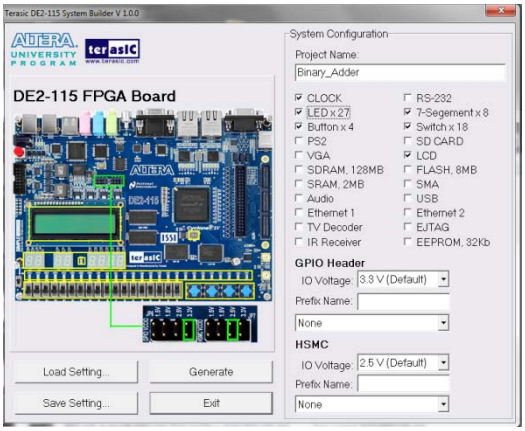
Alternate way to do pin assignments with the help of System Builder

System builder is a GUI that creates pin assignment by selecting the components that will be needed for a project. System builder saves time by creating the pin assignments for you and letting you choose what components you need. For Example:-

1) Open DE2_115_tools->DE2_115_system_builder to find DE2_115_SystemBuilder.exe
2) Name the project under Project Name: in this Tutorial we name or project Binary_Adder

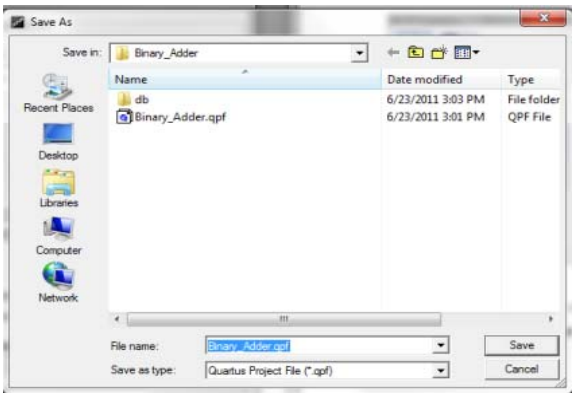


3) Check all Components that you will be using: in this Tutorial we are using CLOCK, LEDx27, Buttonx4, 7-Segementx8, Switchx18, and of course the LCD.



4) Click Generate

5) Create a directory for your project and then click save



6) To open this project open the .qpf file

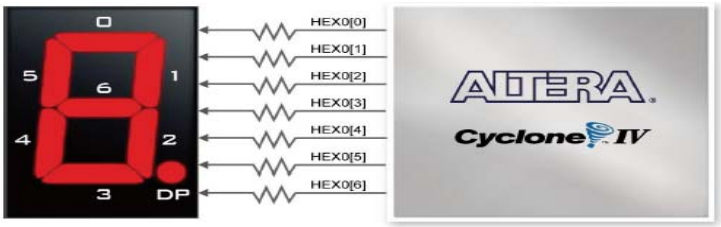
7) Delete the verilog code that System Builder created then copy the code from Binary_Adder_System_Builder

8) Go to steps 8 and 9

7 Segment Hex Display

```
if(!reset_n)begin
 number1 = 0;
 number2 = 0;
end
else begin
 number1 = SW;
 number2 = SW2;
 sum = (number1 + number2);
begin
 //  XX   X#   XXXX
 hex4 = DISPLAYNUMBERS(number1%10);
 //  XX   #X   XXXX
 hex5 = DISPLAYNUMBERS(number1/10);
 //  X#   XX   XXXX
 hex6 = DISPLAYNUMBERS(number2%10);
 //  #X   XX   XXXX
 hex7 = DISPLAYNUMBERS(number2/10);
 //  XX   XX   XXX#
 hex0 = DISPLAYNUMBERS(sum%10);
 //  XX   XX   XX#X
 hex1 = DISPLAYNUMBERS(sum/10);
end
end
end
function [7:0] DISPLAYNUMBERS;
```

```
 input [3:0] value;
begin
 if (value == 0)
 DISPLAYNUMBERS = 7'b1000000;//0
 else if (value == 1)
 DISPLAYNUMBERS = 7'b1111001;//1
 else if (value == 2)
 DISPLAYNUMBERS = 7'b0100100;//2
 else if (value == 3)
 DISPLAYNUMBERS = 7'b0110000;//3
 else if (value == 4)
 DISPLAYNUMBERS = 7'b0011001;//4
 else if (value == 5)
 DISPLAYNUMBERS = 7'b0010010;//5
 else if (value == 6)
 DISPLAYNUMBERS = 7'b0000010;//6
 else if (value == 7)
 DISPLAYNUMBERS = 7'b1111000;//7
 else if (value == 8)
 DISPLAYNUMBERS = 7'b0000000;//8
 else if (value == 9)
 DISPLAYNUMBERS = 7'b0011000;//9
end
endfunction
 assign LEDR = number1;     //Links switch orientation with respective LEDs
 assign LEDR2 = number2;
 assign HEX0 = hex0;        //Links sum value to display output signal
 assign HEX1 = hex1;
 assign HEX4 = hex4;        //Links 1st number value to display output signal
 assign HEX5 = hex5;
 assign HEX6 = hex6;        //Links 2nd number value to display output signal
 assign HEX7 = hex7;
 assign HEX3 = 7'b1111111;
 assign HEX2 = 7'b1111111;
 endmodule
```



In this project we used four 7-segment displays to show the values of switches being turned on in binary. In a 7- segment display a high logic level will turn off the led and a low logic level to a segment will turn the led on. To represent an LED with a seven-bit value we use the values zero through six. To display a zero to a segment we set the hex value to be equal to

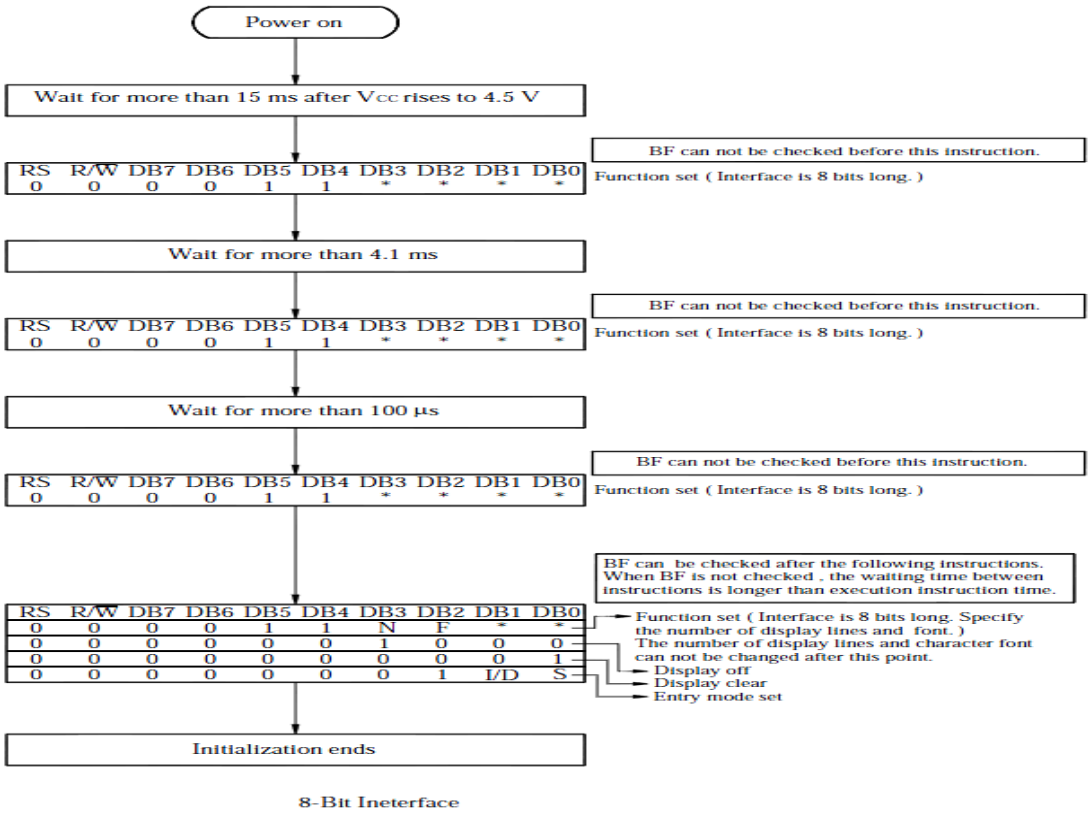
7b’1000000. This is because a zero will have all led on but the center led (number 6 on the figure above). The code also uses a function to simplify the task of representing a bit value to a hex value. Since the function DISPLAYNUMBERS only has one output it seemed like a function instead of a task. In the function we have only one input value that represents a 4 bit switch value, this value is passed through a series of if else statements to determine the hex value. At the end of this program we assign all appropriate values to the represented LEDs.

There is a quick example of getting the LED’s, Switches, Keys, and 7 segment Hex Display to function properly in the link below that goes more in detail about the 7 segment display.

http://www.youtube.com/user/montoya332?blend=6&ob=5#p/a/u/0/78JQ4IgF9yc

16 x 2 LCD

```
case(state)
        RESET1:
        begin
                LCD_EN          <= 1'b1;
                LCD_RS          <= 1'b0;
                LCD_RW          <= 1'b0;
                LCD_DATA <= 8'h38;
                state <= DROP_LCD_E;
                next_command <= RESET2;
        end
        RESET2:
        begin
                LCD_EN          <= 1'b1;
                LCD_RS          <= 1'b0;
                LCD_RW          <= 1'b0;
                LCD_DATA <= 8'h38;
                state <= DROP_LCD_E;
                next_command <= RESET3;
                end
        RESET3:
        begin
                LCD_EN          <= 1'b1;
                LCD_RS          <= 1'b0;
                LCD_RW          <= 1'b0;
                LCD_DATA <= 8'h38;
                state <= DROP_LCD_E;
                next_command <= FUNC_SET;
                end

        FUNC_SET:
        begin
   LCD_EN <= 1'b1;
   LCD_RS <= 1'b0;
   LCD_RW <= 1'b0;
   LCD_DATA <= 8'h38;
   state <= DROP_LCD_E;
   next_command <= DISPLAY_OFF;
  end

        DISPLAY_OFF:
  begin
   LCD_EN <= 1'b1;
   LCD_RS <= 1'b0;
   LCD_RW <= 1'b0;
   LCD_DATA <= 8'h08;
   state <= DROP_LCD_E;
   next_command <= DISPLAY_CLEAR;
  end

        DISPLAY_CLEAR:
  begin
   LCD_EN <= 1'b1;
   LCD_RS <= 1'b0;
   LCD_RW <= 1'b0;
   LCD_DATA <= 8'h01;
   state <= DROP_LCD_E;
                                17: LCD_DATA <= str17;
                                18: LCD_DATA <= hexConv(n1tens);
                                19: LCD_DATA <= hexConv(n1ones);
                                20: LCD_DATA <= str20;
                                21: LCD_DATA <= str21;
                                22: LCD_DATA <= str22;
                                23: LCD_DATA <= hexConv(n2tens);
                                24: LCD_DATA <= hexConv(n2ones);
                                25: LCD_DATA <= str25;
                                26: LCD_DATA <= str26;
                                27: LCD_DATA <= str27;
                                28: LCD_DATA <= hexConv(tens);
                                29: LCD_DATA <= hexConv(ones);
                                30: LCD_DATA <= str30;
                                31: LCD_DATA <= str31;
                                32: LCD_DATA <= str32;
                                default: LCD_DATA <= 8'h20;
                                endcase
                end
        else if (index == displaySize)
                begin
                next_command <= Print_String;
                index <= 5'b0;
                end
        end
        end

RETURN_HOME:
        begin
                LCD_EN <= 1'b1;
                LCD_RS <= 1'b0;
                LCD_RW <= 1'b0;
                LCD_DATA <= 8'h80;
                state <= DROP_LCD_E;
                next_command <= Print_String;
        end

CHANGE_LINE:
        begin
                LCD_EN <= 1'b1;
                LCD_RS <= 1'b0;
                LCD_RW <= 1'b0;
                LCD_DATA <= 8'h0C0;
                state <= DROP_LCD_E;
                next_command <= Print_String;
        end

DROP_LCD_E:
        begin
                LCD_EN <= 1'b0;
                state <= HOLD;
        end

HOLD:
        begin
                state <= next_command;
 next_command <= DISPLAY_ON;
end

        DISPLAY_ON:
begin
 LCD_EN <= 1'b1;
 LCD_RS <= 1'b0;
 LCD_RW <= 1'b0;
 LCD_DATA <= 8'h0C;
 state <= DROP_LCD_E;
 next_command <= MODE_SET;
end

        MODE_SET:
begin
 LCD_EN <= 1'b1;
 LCD_RS <= 1'b0;
 LCD_RW <= 1'b0;
                LCD_DATA <= 8'h06;
 state <= DROP_LCD_E;
 next_command <= Print_String;
                index <= 5'b0;
 end
        Print_String:
                begin
                index <= index + 1'b1;
 state <= DROP_LCD_E;
 LCD_EN <= 1'b1;
 LCD_RS <= 1'b1;
 LCD_RW <= 1'b0;
                begin
                if(index < displaySize)
                        begin
                                if(index < line1)
                                        case(index)
                                        1: LCD_DATA <= str1;
                                        2: LCD_DATA <= str2;
                                        3: LCD_DATA <= str3;
                                        4: LCD_DATA <= str4;
                                        5: LCD_DATA <= str5;
                                        6: LCD_DATA <= str6;
                                        7: LCD_DATA <= str7;
                                        8: LCD_DATA <= str8;
                                        9: LCD_DATA <= str9;
                                        10: LCD_DATA <= str10;
                                        11: LCD_DATA <= str11;
                                        12: LCD_DATA <= str12;
                                        13: LCD_DATA <= str13;
                                        14: LCD_DATA <= str14;
                                        15: LCD_DATA <= str15;
                                        default: LCD_DATA <= 8'h20;
                                        endcase
                                else if(index == line1)
                                        next_command <= CHANGE_LINE;
                                else if(index > line1)
                                        case(index)

                                end
  endcase
        end
end

function [8:0] hexConv;
        input [3:0] hex;
        begin
        if (hex == 4'b0000)//0
                hexConv = 8'b00110000;
                else if (hex == 4'b0001)//1
                hexConv = 8'b00110001;
                else if (hex == 4'b0010)//2
                hexConv = 8'b00110010;
                else if (hex == 4'b0011)//3
                hexConv = 8'b00110011;
                else if (hex == 4'b0100)//4
                hexConv = 8'b00110100;
                else if (hex == 4'b0101)//5
                hexConv = 8'b00110101;
                else if (hex == 4'b0110)//6
                hexConv = 8'b00110110;
                else if (hex == 4'b0111)//7
                hexConv = 8'b00110111;
                else if (hex == 4'b1000)//8
                hexConv = 8'b00111000;
                else if (hex == 4'b1001)//9
                hexConv = 8'b00111001;
                end
                endfunction
endmodule
```



8-Bit Ineterface

To display characters to an LCD there is a series of steps that need to be done before to initializing the LCD module. Since Verilog doesn't read code sequentially we created a case statement that will allow the initialization to be done in order. This is done by changing the state of the case to the next step in every case statement. The steps performed are RESET1, RESET2, RESET3, FUNCTION SET, DISPLAY OFF, DISPLAY CLEAR, RETURN HOME, CHANGE

LINE, DROP LCD, HOLD, DISPLAY ON, and MODE SET AND PRINT STRING. These reset needs to be done three time to because we need to initialize enable to high and register select and read/write to low signals. These steps are also done to communicate with the LCD to determine if it will be an 8 or 4 bit data bus, this is done by setting the data bus equal to the hex value eight(8'h38). Before we can write to the screen we need to clear the LCD display, this is done by changing the data bus equal to 8'h01 (Start of heading). Finally when we need to display the screen we set enable and read/write to high and reset to low, this is done because this allows us to write data to the LCD. In the print string case statement we added an else if (index ==line1) because without this the LCD wouldn't know when the next line begin or the first line starts.