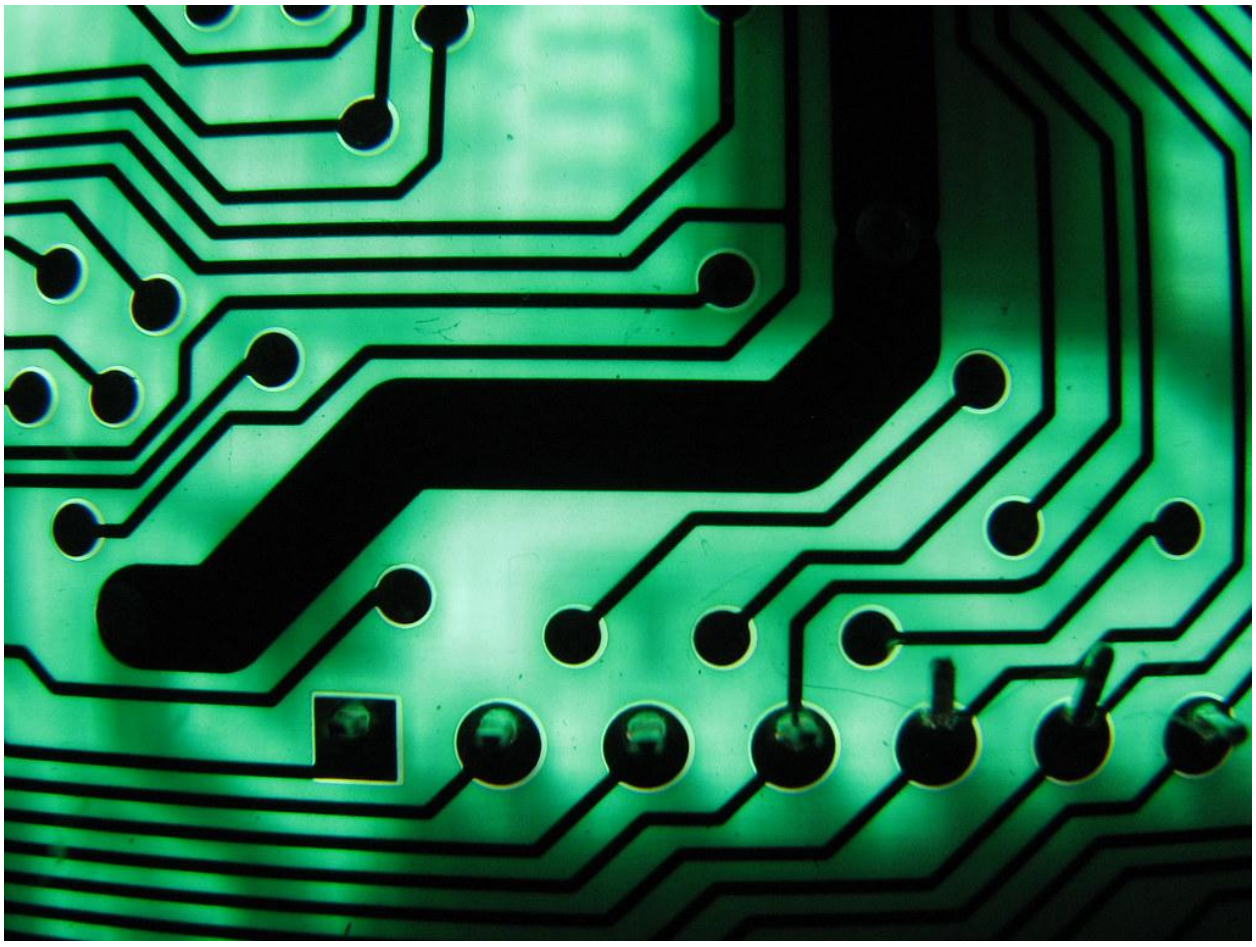
EEC 180 DIGITAL SYSTEMS II

Lab #1
Implementing Combinational Logic in the MAX10 FPGA



Introduction

The purpose of this lab is to program an FPGA to do basic combinational circuits, use switches, push buttons, and 7-segment led displays.

Tobin Joseph

Results

1 Prelab

Part 1.1 Install Quartus

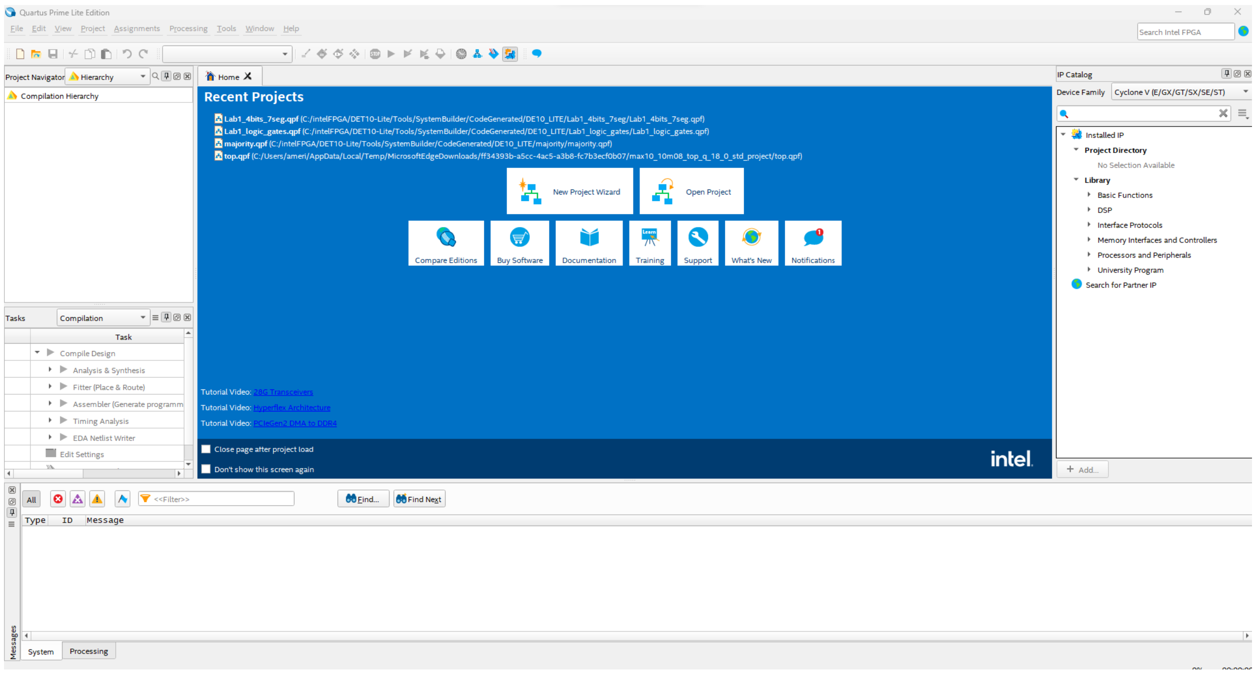
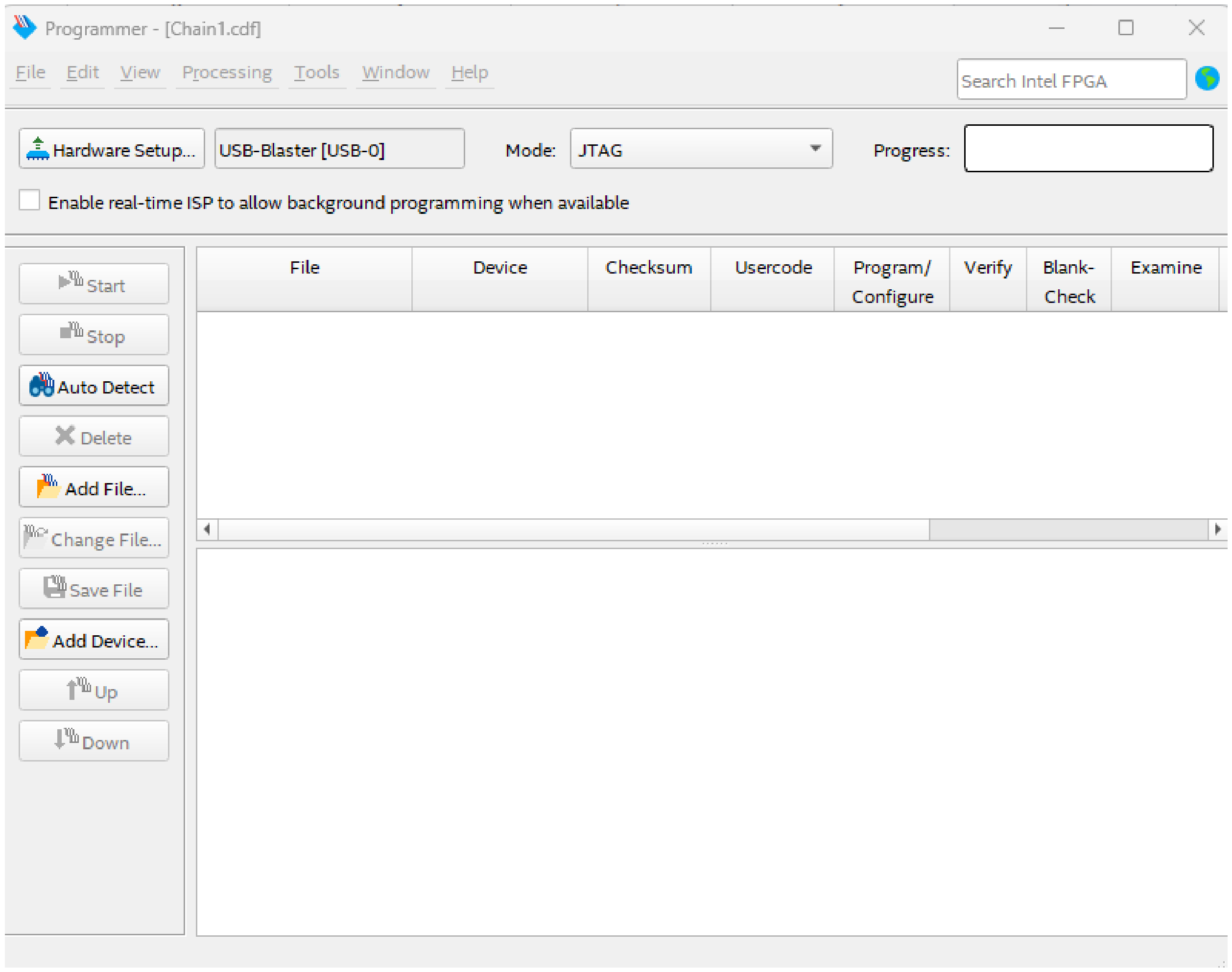

Quartus Prime Lite Edition
File Edit View Project Assignments Processing Tools Window Help
Search Intel FPGA
Project Navigator
Hierarchy
Compilation Hierarchy
Home
Recent Projects
Lab1_4bits_7seg.qpf (C:/intelFPGA/DET10-Lite/Tools/SystemBuilder/CodeGenerated/DE10_LITE/Lab1_4bits_7seg/Lab1_4bits_7seg.qpf)
Lab1_logic_gates.qpf (C:/intelFPGA/DET10-Lite/Tools/SystemBuilder/CodeGenerated/DE10_LITE/Lab1_logic_gates/Lab1_logic_gates.qpf)
majority.qpf (C:/intelFPGA/DET10-Lite/Tools/SystemBuilder/CodeGenerated/DE10_LITE/majority/majority.qpf)
top.qpf (C:/Users/ameri/AppData/Local/Temp/MicrosoftEdgeDownloads/ff34393b-a5cc-4ac5-a3b8-fc7b3ecf0b07/max10_10m08_top_q_18_0_std_project/top.qpf)
New Project Wizard
Open Project
Compare Editions
Buy Software
Documentation
Training
Support
What's New
Notifications
Tutorial Video: 10G Transceivers
Tutorial Video: Hyperflex Architecture
Tutorial Video: PCIeGen2 DMA to DDR4
Close page after project load
Don't show this screen again
intel
IP Catalog
Device Family
Cyclone V (E/GX/GT/SX/SE/ST)
Installed IP
Project Directory
No Selection Available
Library
Basic Functions
DSP
Interface Protocols
Memory Interfaces and Controllers
Processors and Peripherals
University Program
Search for Partner IP
Add...
Tasks
Compilation
Task
Compile Design
Analysis & Synthesis
Fitter (Place & Route)
Assembler (Generate programm
Timing Analysis
EDA Netlist Writer
Edit Settings
All
<<Filter>>
Find...
Find Next
Type
ID
Message
Messages
System
Processing

Figure 1.1.1 Quartus 22.1.



USB-Blaster [USB-0]

Mode: JTAG

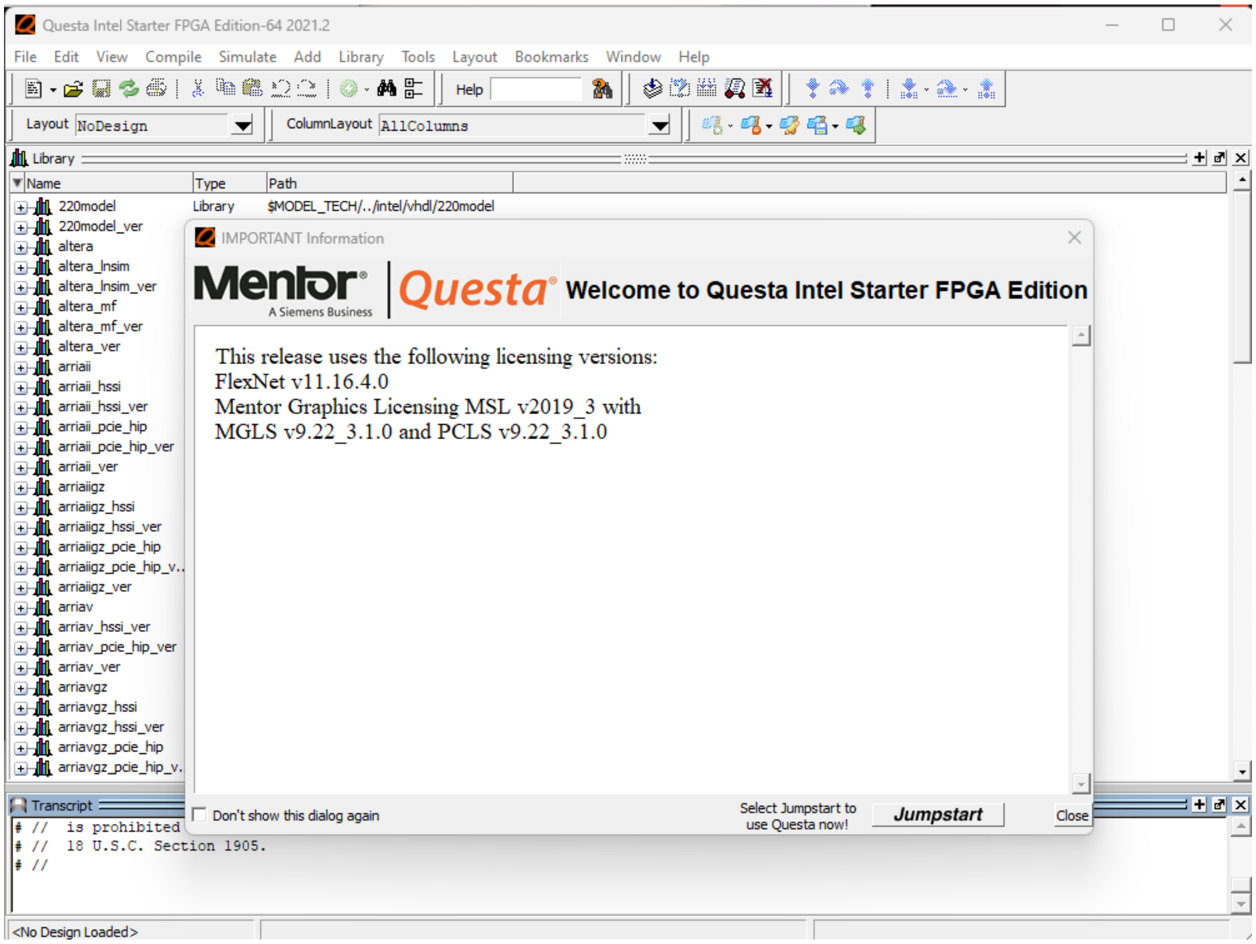
Progress:

Enable real-time ISP to allow background programming when available

Figure 1.1.2 Quartus 22.1 Programer.

Part 1.2 Install ModelSim

Modelsim was substituted for Questa.


Questa Intel Starter FPGA Edition-64 2021.2
File Edit View Compile Simulate Add Library Tools Layout Bookmarks Window Help
Layout NoDesign
ColumnLayout AllColumns
Library
Name Type Path
IMPORTANT Information
Mentor A Siemens Business
Questa
Don't show this dialog again
Select Jumpstart to use Questa now!
Jumpstart
Close
Transcript
<No Design Loaded>

Name	Type	Path
220model	Library	$MODEL_TECH/../intel/vhdl/220model
220model_ver		
altera		
altera_lnsim		
altera_lnsim_ver		
altera_mf		
altera_mf_ver		
altera_ver		
arriaii		
arriaii_hssi		
arriaii_hssi_ver		
arriaii_pcie_hip		
arriaii_pcie_hip_ver		
arriaii_ver		
arriaiigz		
arriaiigz_hssi		
arriaiigz_hssi_ver		
arriaiigz_pcie_hip		
arriaiigz_pcie_hip_v..		
arriaiigz_ver		
arriav		
arriav_hssi_ver		
arriav_pcie_hip_ver		
arriav_ver		
arriavgz		
arriavgz_hssi		
arriavgz_hssi_ver		
arriavgz_pcie_hip		
arriavgz_pcie_hip_v..		

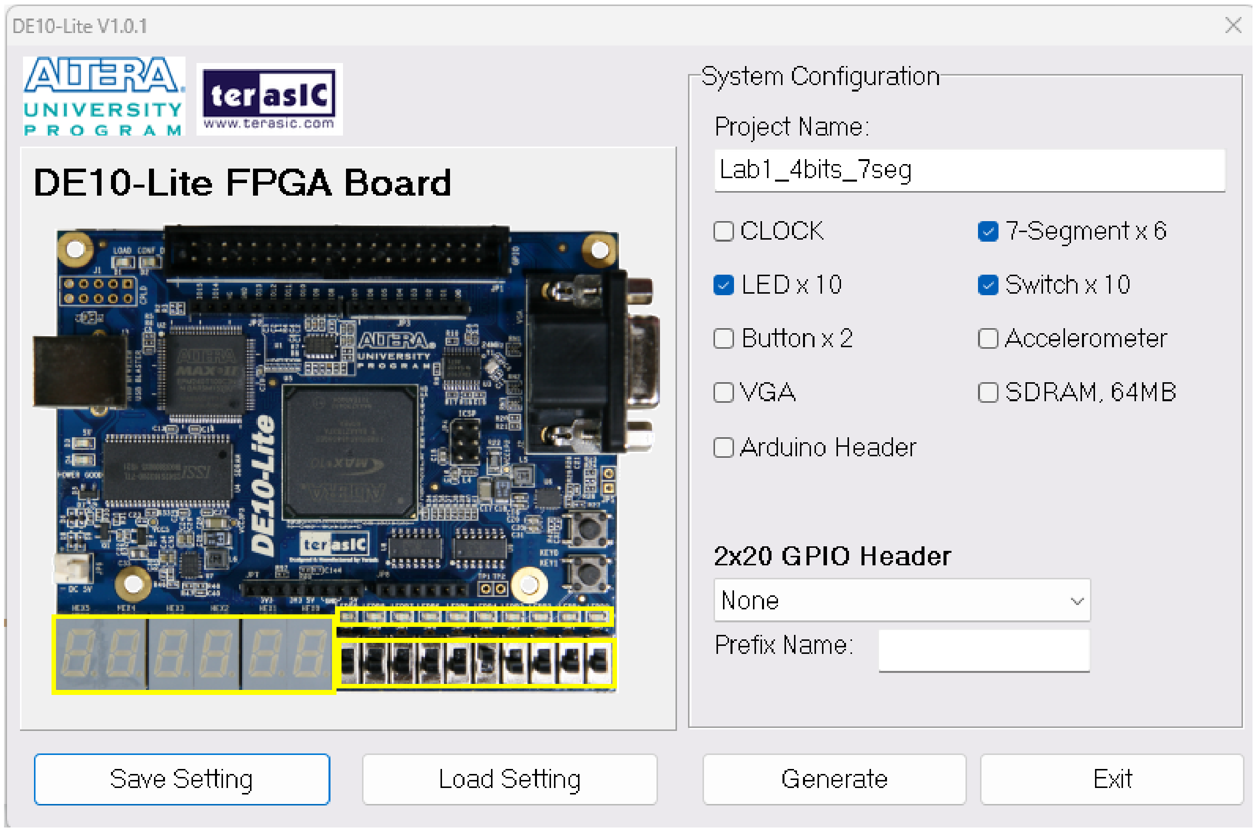
Welcome to Questa Intel Starter FPGA Edition

This release uses the following licensing versions:
FlexNet v11.16.4.0
Mentor Graphics Licensing MSL v2019_3 with
MGLS v9.22_3.1.0 and PCLS v9.22_3.1.0

```
# //  is prohibited
# //  18 U.S.C. Section 1905.
# //
```

Figure 1.2.1 Questa Lite 2021.2.

1.3 Install SystemBuilder



System Configuration

Project Name: Lab1_4bits_7seg

- [ ] CLOCK
- [x] 7-Segment x 6
- [x] LED x 10
- [x] Switch x 10
- [ ] Button x 2
- [ ] Accelerometer
- [ ] VGA
- [ ] SDRAM, 64MB
- [ ] Arduino Header

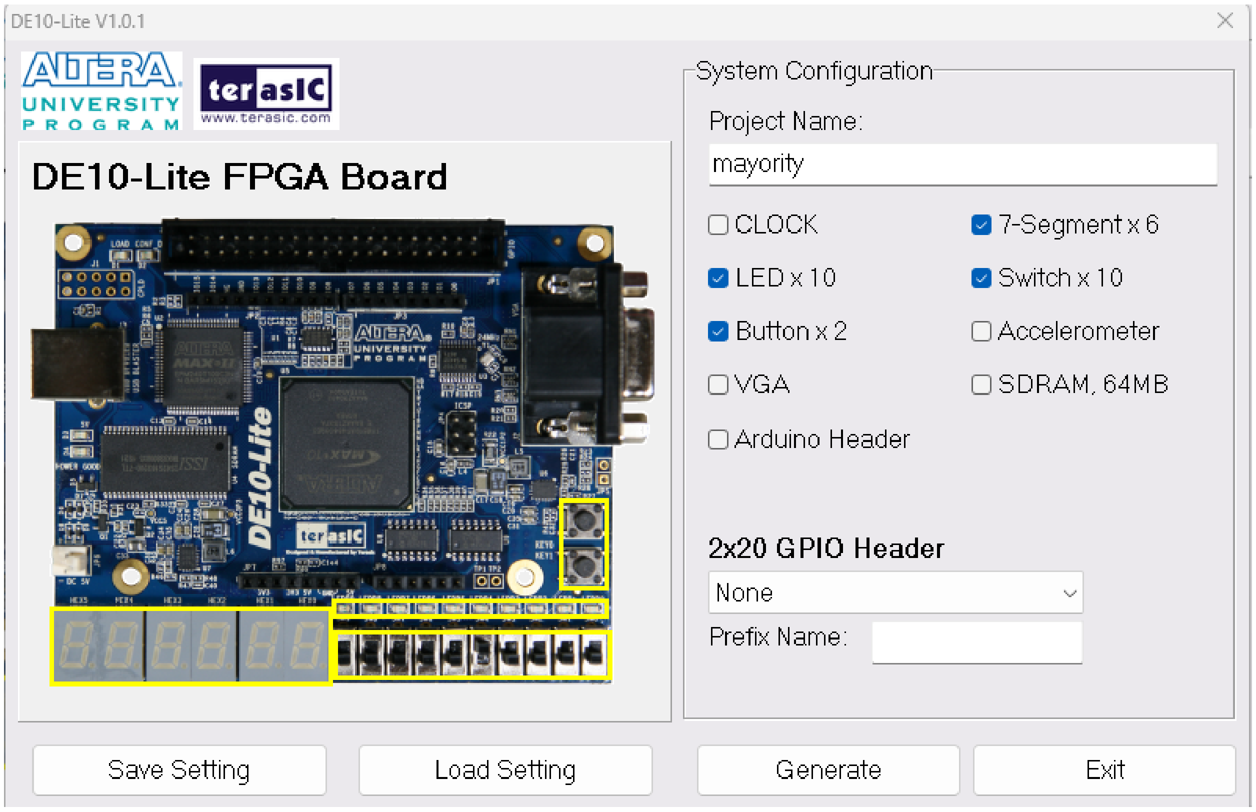
2x20 GPIO Header: None

Prefix Name:

Figure 1.3.1 System Builder.

2 Design Flow using System Builder, Quartus Prime and ModelSim

Part 2.1 Creating Project Files Using System Builder



System Configuration

Project Name: mayority

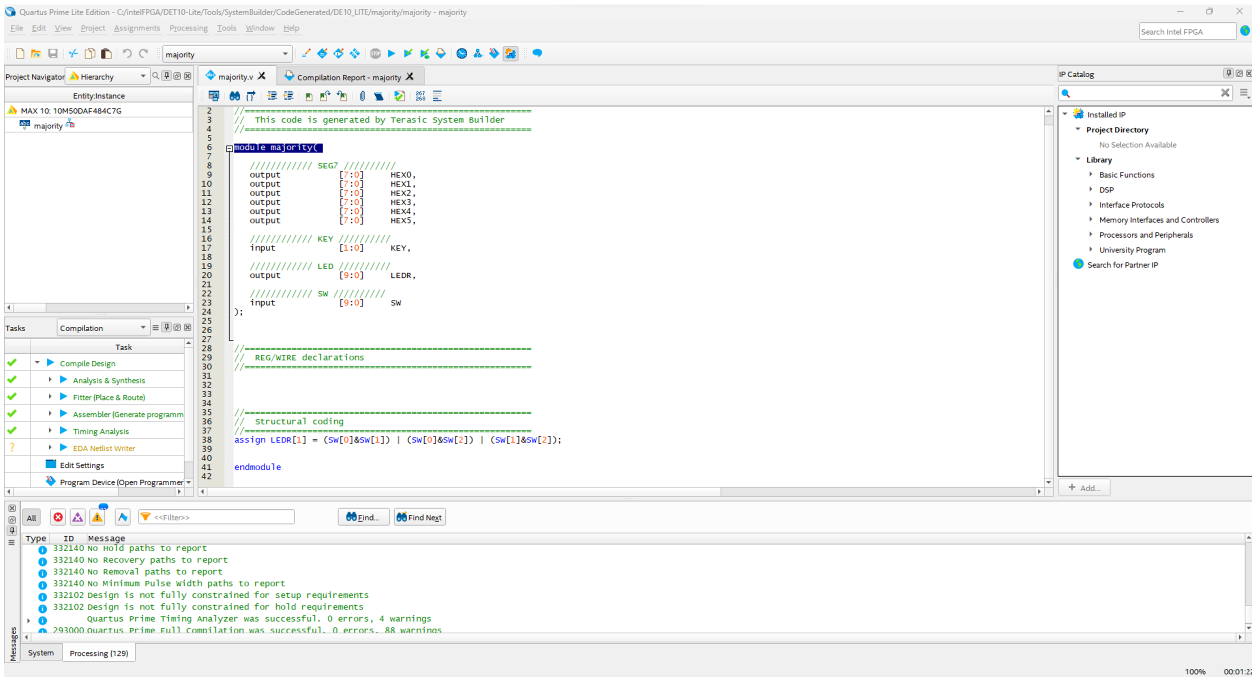
- [ ] CLOCK
- [x] 7-Segment x 6
- [x] LED x 10
- [x] Switch x 10
- [x] Button x 2
- [ ] Accelerometer
- [ ] VGA
- [ ] SDRAM, 64MB
- [ ] Arduino Header

2x20 GPIO Header: None

Prefix Name:

Figure 2.1.1 Creating a project with System Builder.

Part 2.2 Compiling and Programming Using Quartus Prime

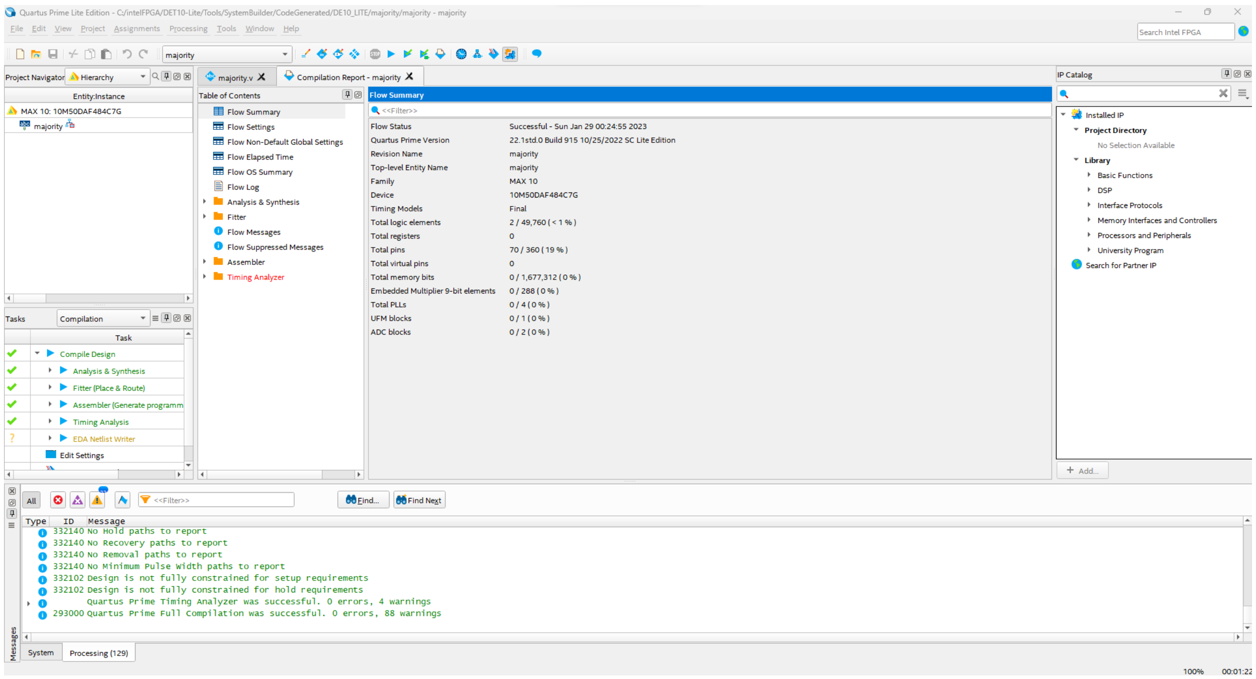


Quartus Prime Lite Edition - C:/intelFPGA/DET10-Lite/Tools/SystemBuilder/CodeGenerated/DE10_LITE/majority/majority - majority

```
//=======================================================
//  This code is generated by Terasic System Builder
//=======================================================

module majority(

	//////////// SEG7 //////////
	output		     [7:0]		HEX0,
	output		     [7:0]		HEX1,
	output		     [7:0]		HEX2,
	output		     [7:0]		HEX3,
	output		     [7:0]		HEX4,
	output		     [7:0]		HEX5,

	//////////// KEY //////////
	input 		     [1:0]		KEY,

	//////////// LED //////////
	output		     [9:0]		LEDR,

	//////////// SW //////////
	input 		     [9:0]		SW
);



//=======================================================
//  REG/WIRE declarations
//=======================================================




//=======================================================
//  Structural coding
//=======================================================
assign LEDR[1] = (SW[0]&SW[1]) | (SW[0]&SW[2]) | (SW[1]&SW[2]);


endmodule
```

```
Type  ID     Message
      332140 No Hold paths to report
      332140 No Recovery paths to report
      332140 No Removal paths to report
      332140 No Minimum Pulse Width paths to report
      332102 Design is not fully constrained for setup requirements
      332102 Design is not fully constrained for hold requirements
                   Quartus Prime Timing Analyzer was successful. 0 errors, 4 warnings
      293000 Quartus Prime Full Compilation was successful. 0 errors, 88 warnings
```

Figure 2.2.1 Programing using Quartus Prime.



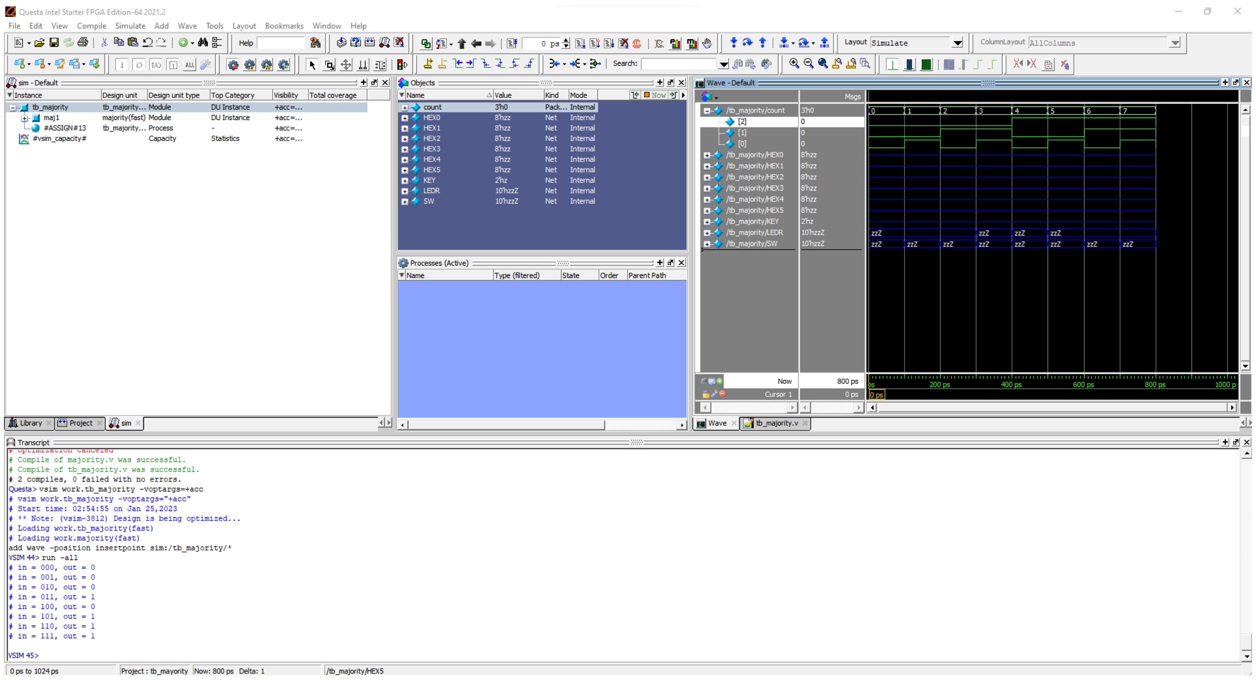
Flow Summary

Flow Status	Successful - Sun Jan 29 00:24:55 2023
Quartus Prime Version	22.1std.0 Build 915 10/25/2022 SC Lite Edition
Revision Name	majority
Top-level Entity Name	majority
Family	MAX 10
Device	10M50DAF484C7G
Timing Models	Final
Total logic elements	2 / 49,760 (< 1 %)
Total registers	0
Total pins	70 / 360 (19 %)
Total virtual pins	0
Total memory bits	0 / 1,677,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 288 (0 %)
Total PLLs	0 / 4 (0 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0 / 2 (0 %)

```
Type  ID     Message
      332140 No Hold paths to report
      332140 No Recovery paths to report
      332140 No Removal paths to report
      332140 No Minimum Pulse Width paths to report
      332102 Design is not fully constrained for setup requirements
      332102 Design is not fully constrained for hold requirements
                   Quartus Prime Timing Analyzer was successful. 0 errors, 4 warnings
      293000 Quartus Prime Full Compilation was successful. 0 errors, 88 warnings
```

Figure 2.2.2 Compiling using Quartus Prime.

Part 2.3 Simulating in ModelSim-Intel Using a Testbench



```
# optimization canceled
# Compile of majority.v was successful.
# Compile of tb_majority.v was successful.
# 2 compiles, 0 failed with no errors.
Questa> vsim work.tb_majority -voptargs=+acc
# vsim work.tb_majority -voptargs="+acc"
# Start time: 02:54:55 on Jan 25,2023
# ** Note: (vsim-3812) Design is being optimized...
# Loading work.tb_majority(fast)
# Loading work.majority(fast)
add wave -position insertpoint sim:/tb_majority/*
VSIM 44> run -all
# in = 000, out = 0
# in = 001, out = 0
# in = 010, out = 0
# in = 011, out = 1
# in = 100, out = 0
# in = 101, out = 1
# in = 110, out = 1
# in = 111, out = 1

VSIM 45>
```

0 ps to 1024 ps | Project : tb_mayority | Now: 800 ps Delta: 1 | /tb_majority/HEX5

Figure 2.3.1 Wave Results for simulation of tb_mayority.v.

```
# in = 000, out = 0
# in = 001, out = 0
# in = 010, out = 0
# in = 011, out = 1
# in = 100, out = 0
# in = 101, out = 1
# in = 110, out = 1
# in = 111, out = 1
```

Figure 2.3.2 Report results for simulation of tb_mayority.v.

3 Implementing Basic Combinational Logic Gates on the DE10-Lite Board

Part 3.1 Verilog Code

```
//=======================================================

//  This code is generated by Terasic System Builder

//=======================================================


module Lab1_logic_gates(


//////////// SEG7 //////////

output                  [7:0]       HEX0,

output                  [7:0]       HEX1,

output                  [7:0]       HEX2,

output                  [7:0]       HEX3,

output                  [7:0]       HEX4,

output                  [7:0]       HEX5,


//////////// KEY //////////

input                   [1:0]       KEY,


//////////// LED //////////

output                  [9:0]       LEDR,


//////////// SW //////////

input                   [9:0]       SW

);
```

```
//=======================================================
//  REG/WIRE declarations
//=======================================================



//=======================================================
//  Structural coding
//=======================================================

// 2 input and gate
assign LEDR[0] = (SW[0]&SW[1]);

// 3 input or gate
assign LEDR[1] = (SW[2]|SW[3]|SW[4]);

// turn on led segment with switch
assign HEX2[3] = ~SW[9];

// turn on led segment with button 0
assign HEX0[1] = KEY[0];

// turn on led segment with button 1
assign HEX5[6] = KEY[1];

endmodule
```

Figure 3.1.1 Verilog code for implementing sample basic combinational logic gates.

Part 3.2 Testbench Module Verilog Code

```
`timescale 1ps/1ps

module tb_Lab1_logic_gates;

reg [9:0] switches;

reg [1:0] keys;

wire [7:0] HEX0;

wire [7:0] HEX1;

wire [7:0] HEX2;

wire [7:0] HEX3;

wire [7:0] HEX4;

wire [7:0] HEX5;

wire [1:0] KEY;

wire [9:0] LEDR;

wire [9:0] SW;


assign SW[9:0] = switches;

assign KEY[1:0] = keys;


// The part of the testbench that instantiates the hardware to be tested

Lab1_logic_gates lg1 (.HEX0(HEX0), .HEX1(HEX1), .HEX2(HEX2),

.HEX3(HEX3), .HEX4(HEX4), .HEX5(HEX5),

.KEY(KEY), .LEDR(LEDR), .SW(SW));


initial begin


      // 2 input and gate
```

```
$display("***************************************");
$display("2 input and gate");
switches[1] = 1'b0;
switches[0] = 1'b0;
#100
$display("in = %b, led[0] = %b", switches, LEDR[0]);

switches[1] = 1'b0;
switches[0] = 1'b1;
#100
$display("in = %b, led[0] = %b", switches, LEDR[0]);

switches[1] = 1'b1;
switches[0] = 1'b0;
#100
$display("in = %b, led[0] = %b", switches, LEDR[0]);

switches[1] = 1'b1;
switches[0] = 1'b1;
#100
$display("in = %b, led[0] = %b", switches, LEDR[0]);



// 3 input or gate
$display("***************************************");
$display("TEST: 3 input or gate");
switches[4] = 1'b0;
```

```
    switches[3] = 1'b0;

    switches[2] = 1'b0;

    #100

    $display("in = %b, led[1] = %b", switches, LEDR[1]);


    switches[4] = 1'b0;

    switches[3] = 1'b0;

    switches[2] = 1'b1;

    #100

    $display("in = %b, led[1] = %b", switches, LEDR[1]);


    switches[4] = 1'b0;

    switches[3] = 1'b1;

    switches[2] = 1'b0;

    #100

    $display("in = %b, led[1] = %b", switches, LEDR[1]);


    switches[4] = 1'b0;

    switches[3] = 1'b1;

    switches[2] = 1'b1;

    #100

    $display("in = %b, led[1] = %b", switches, LEDR[1]);


    switches[4] = 1'b1;

    switches[3] = 1'b0;

    switches[2] = 1'b0;

    #100
```

```
$display("in = %b, led[1] = %b", switches, LEDR[1]);


switches[4] = 1'b1;

switches[3] = 1'b0;

switches[2] = 1'b1;

#100

$display("in = %b, led[1] = %b", switches, LEDR[1]);


switches[4] = 1'b1;

switches[3] = 1'b1;

switches[2] = 1'b0;

#100

$display("in = %b, led[1] = %b", switches, LEDR[1]);


switches[4] = 1'b1;

switches[3] = 1'b1;

switches[2] = 1'b1;

#100

$display("in = %b, led[1] = %b", switches, LEDR[1]);


// turn on led segment with switch 9

$display("***************************************");

$display("TEST: turn on led segment with switch 9");

switches[9] = 1'b0;

#100

$display("in = %b, HEX2 = %b", switches, HEX2);

switches[9] = 1'b1;
```

```
#100
$display("in = %b, HEX2 = %b", switches, HEX2);
switches[9] = 1'b0;
#100
$display("in = %b, HEX2 = %b", switches, HEX2);

// turn on led segment with button 0
$display("***************************************");
$display("TEST: turn on led segment with button 0");
keys[1] = 1'b1;
#100
$display("push buttons = %b, HEX0 = %b", keys, HEX0);
keys[0] = 1'b1;
#100
$display("push buttons = %b, HEX0 = %b", keys, HEX0);
keys[0] = 1'b0;
#100
$display("push buttons = %b, HEX0 = %b", keys, HEX0);
keys[0] = 1'b1;
#100
$display("push buttons = %b, HEX0 = %b", keys, HEX0);


// turn on led segment with button 1
$display("***************************************");
$display("TEST: turn on led segment with button 1");
keys[0] = 1'b1;
```

```
        #100
        $display("push buttons = %b, HEX5 = %b", keys, HEX5);
        keys[1] = 1'b0;
        #100
        $display("push buttons = %b, HEX5 = %b", keys, HEX5);
        keys[1] = 1'b1;
        #100
        $display("push buttons = %b, HEX5 = %b", keys, HEX5);
        keys[1] = 1'b0;
end
endmodule
```

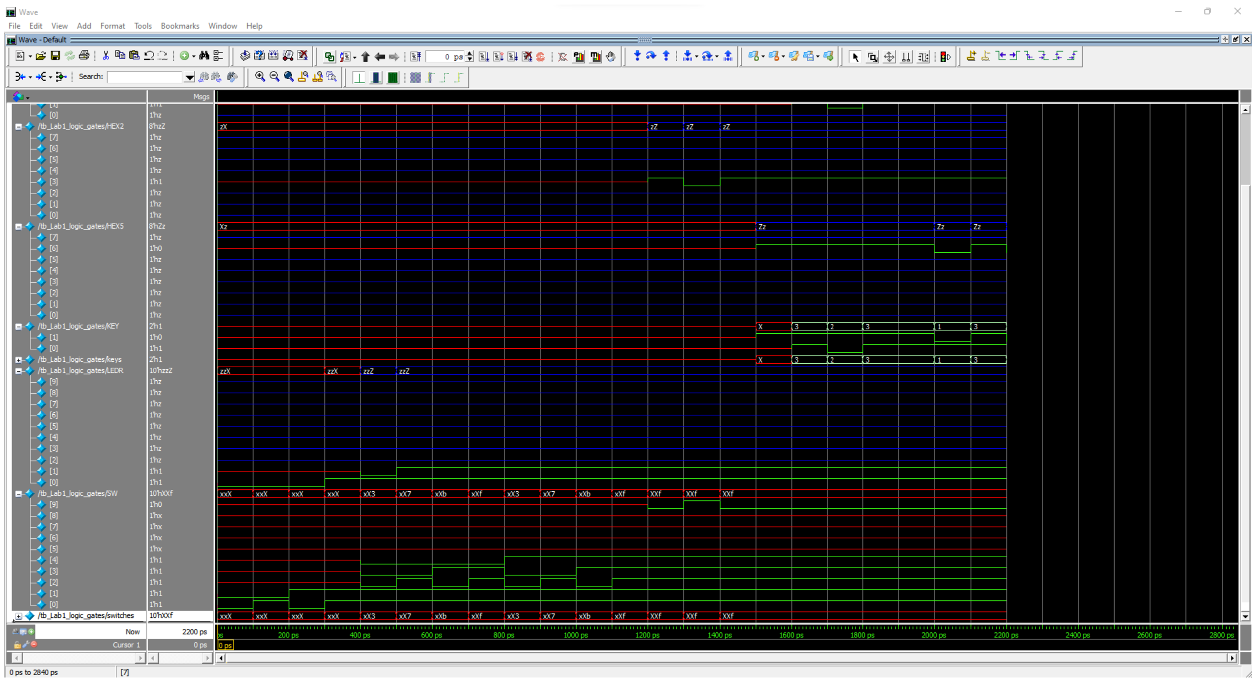
Figure 3.2.1 Verilog code for implementing a test bench for sample basic combinational logic gates.

Part 3.3 Testbench Results Report

```
# ***************************************
# 2 input and gate
# in = xxxxxxxx00, led[0] = 0
# in = xxxxxxxx01, led[0] = 0
# in = xxxxxxxx10, led[0] = 0
# in = xxxxxxxx11, led[0] = 1
# ***************************************
# TEST: 3 input or gate
# in = xxxxx00011, led[1] = 0
# in = xxxxx00111, led[1] = 1
# in = xxxxx01011, led[1] = 1
# in = xxxxx01111, led[1] = 1

# in = xxxxx10011, led[1] = 1

# in = xxxxx10111, led[1] = 1

# in = xxxxx11011, led[1] = 1

# in = xxxxx11111, led[1] = 1

# ***************************************

# TEST: turn on led segment with switch 9

# in = 0xxxx11111, HEX2 = zzzz1zzz

# in = 1xxxx11111, HEX2 = zzzz0zzz

# in = 0xxxx11111, HEX2 = zzzz1zzz

# ***************************************

# TEST: turn on led segment with button 0

# push buttons = 1x, HEX0 = zzzzzzxz

# push buttons = 11, HEX0 = zzzzzz1z

# push buttons = 10, HEX0 = zzzzzz0z

# push buttons = 11, HEX0 = zzzzzz1z

# ***************************************

# TEST: turn on led segment with button 1

# push buttons = 11, HEX5 = z1zzzzzz

# push buttons = 01, HEX5 = z0zzzzzz

# push buttons = 11, HEX5 = z1zzzzzz
```

Figure 3.3.1 Report from test bench for basic combinational logic gates.

Part 3.4 Testbench Wave



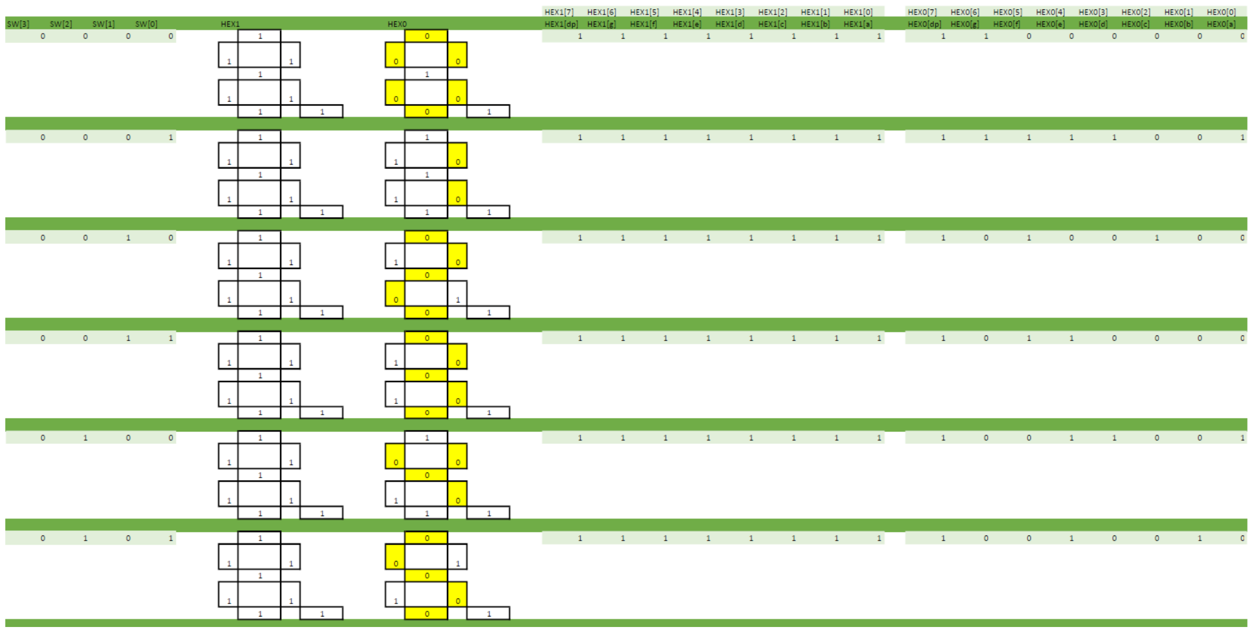
```
                                      Msgs
    └─ [0]                            1'hz
⊟─ /tb_Lab1_logic_gates/HEX2          8'hzZ
    ├─ [7]                            1'hz
    ├─ [6]                            1'hz
    ├─ [5]                            1'hz
    ├─ [4]                            1'hz
    ├─ [3]                            1'h1
    ├─ [2]                            1'hz
    ├─ [1]                            1'hz
    └─ [0]                            1'hz
⊟─ /tb_Lab1_logic_gates/HEX5          8'hZz
    ├─ [7]                            1'hz
    ├─ [6]                            1'h0
    ├─ [5]                            1'hz
    ├─ [4]                            1'hz
    ├─ [3]                            1'hz
    ├─ [2]                            1'hz
    ├─ [1]                            1'hz
    └─ [0]                            1'hz
⊟─ /tb_Lab1_logic_gates/KEY           2'h1
    ├─ [1]                            1'h0
    └─ [0]                            1'h1
⊞─ /tb_Lab1_logic_gates/keys          2'h1
⊟─ /tb_Lab1_logic_gates/LEDR          10'hzzZ
    ├─ [9]                            1'hz
    ├─ [8]                            1'hz
    ├─ [7]                            1'hz
    ├─ [6]                            1'hz
    ├─ [5]                            1'hz
    ├─ [4]                            1'hz
    ├─ [3]                            1'hz
    ├─ [2]                            1'hz
    ├─ [1]                            1'h1
    └─ [0]                            1'h1
⊟─ /tb_Lab1_logic_gates/SW            10'hXXf
    ├─ [9]                            1'h0
    ├─ [8]                            1'hx
    ├─ [7]                            1'hx
    ├─ [6]                            1'hx
    ├─ [5]                            1'hx
    ├─ [4]                            1'h1
    ├─ [3]                            1'h1
    ├─ [2]                            1'h1
    ├─ [1]                            1'h1
    └─ [0]                            1'h1
⊞─ /tb_Lab1_logic_gates/switches      10'hXXf

                               Now    2200 ps
                          Cursor 1       0 ps

0 ps to 2840 ps    [7]
```

Figure 3.4.1 Wave result of test bench for sample basic combinational logic gates.

4 Implementing a Combinational Logic Decimal 7-segment Display for 4-bit Switch Inputs

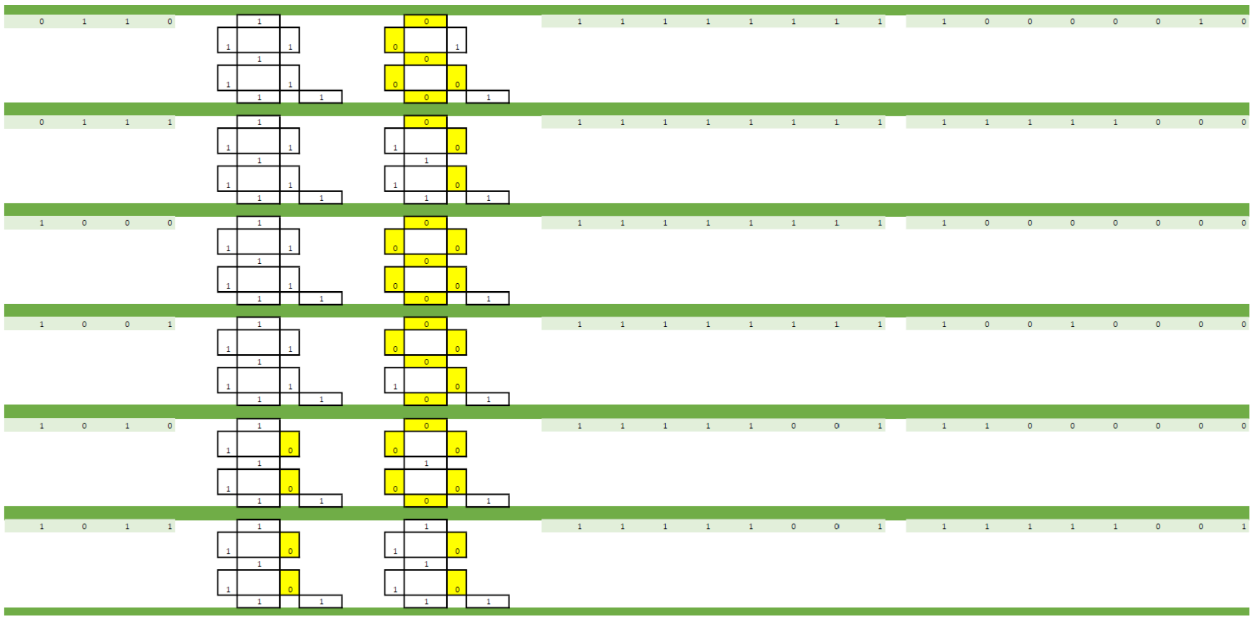
Part 4.1 Truth Table for Design

We used an Excel spreadsheet to graphically depict the digits on the 7 segment displays and come up with the truth tables.



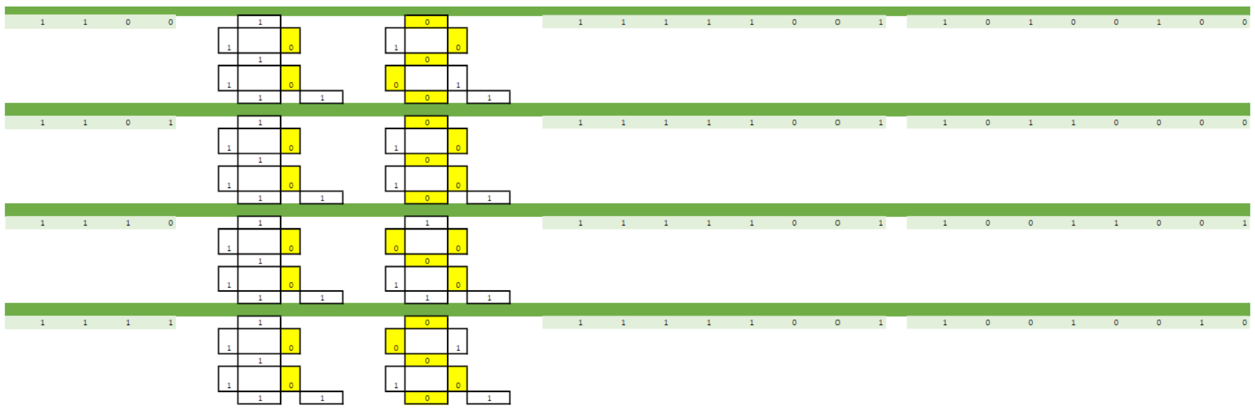
						HEX1[7]	HEX1[6]	HEX1[5]	HEX1[4]	HEX1[3]	HEX1[2]	HEX1[1]	HEX1[0]	HEX0[7]	HEX0[6]	HEX0[5]	HEX0[4]	HEX0[3]	HEX0[2]	HEX0[1]	HEX0[0]
SW[3]	SW[2]	SW[1]	SW[0]	HEX1	HEX0	HEX1[dp]	HEX1[g]	HEX1[f]	HEX1[e]	HEX1[d]	HEX1[c]	HEX1[b]	HEX1[a]	HEX0[dp]	HEX0[g]	HEX0[f]	HEX0[e]	HEX0[d]	HEX0[c]	HEX0[b]	HEX0[a]
0	0	0	0			1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
0	0	0	1			1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1
0	0	1	0			1	1	1	1	1	1	1	1	1	0	1	0	0	1	0	0
0	0	1	1			1	1	1	1	1	1	1	1	1	0	1	1	0	0	0	0
0	1	0	0			1	1	1	1	1	1	1	1	1	0	0	1	1	0	0	1
0	1	0	1			1	1	1	1	1	1	1	1	1	0	0	1	0	0	1	0

Figure 4.1.1 Graphical depictions of digits 0 - 5 for generating truth tables.



0	1	1	0			1	1	1	1	1	1	1	1	1	0	0	0	0	0	1	0
0	1	1	1			1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
1	0	0	0			1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
1	0	0	1			1	1	1	1	1	1	1	1	1	0	0	1	0	0	0	0
1	0	1	0			1	1	1	1	1	0	0	1	1	1	0	0	0	0	0	0
1	0	1	1			1	1	1	1	1	0	0	1	1	1	1	1	1	0	0	1

Figure 4.1.2 Graphical depictions of digits 6 - 11 for generating truth tables.



1	1	0	0	1	1	1	1	1	0	0	1	1	0	1	0	0	1	0	0
1	1	0	1	1	1	1	1	1	0	0	1	1	0	1	1	0	0	0	0
1	1	1	0	1	1	1	1	1	0	0	1	1	0	0	1	1	0	0	1
1	1	1	1	1	1	1	1	1	0	0	1	1	0	0	1	0	0	1	0

Figure 4.1.3 Graphical depictions of digits 12 - 15 for generating truth tables.

SW [3]	SW [2]	SW [1]	SW [0]	HEX1 [7]	HEX1 [6]	HEX1 [5]	HEX1 [4]	HEX1 [3]	HEX1 [2]	HEX1 [1]	HEX1 [0]
0	0	0	0	1	1	1	1	1	1	1	1
0	0	0	1	1	1	1	1	1	1	1	1
0	0	1	0	1	1	1	1	1	1	1	1
0	0	1	1	1	1	1	1	1	1	1	1
0	1	0	0	1	1	1	1	1	1	1	1
0	1	0	1	1	1	1	1	1	1	1	1
0	1	1	0	1	1	1	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	1	1	1	1
1	0	1	0	1	1	1	1	1	0	0	1
1	0	1	1	1	1	1	1	1	0	0	1
1	1	0	0	1	1	1	1	1	0	0	1
1	1	0	1	1	1	1	1	1	0	0	1
1	1	1	0	1	1	1	1	1	0	0	1
1	1	1	1	1	1	1	1	1	0	0	1

SW [3]	SW [2]	SW [1]	SW [0]	HEX0 [7]	HEX0 [6]	HEX0 [5]	HEX0 [4]	HEX0 [3]	HEX0 [2]	HEX0 [1]	HEX0 [0]
0	0	0	0	1	1	0	0	0	0	0	0
0	0	0	1	1	1	1	1	1	0	0	1
0	0	1	0	1	0	1	0	0	1	0	0
0	0	1	1	1	0	1	1	0	0	0	0
0	1	0	0	1	0	0	1	1	0	0	1
0	1	0	1	1	0	0	1	0	0	1	0
0	1	1	0	1	0	0	0	0	0	1	0
0	1	1	1	1	1	1	1	1	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	1	0	0	1	0	0	0	0
1	0	1	0	1	1	0	0	0	0	0	0
1	0	1	1	1	1	1	1	1	0	0	1
1	1	0	0	1	0	1	0	0	1	0	0
1	1	0	1	1	0	1	1	0	0	0	0
1	1	1	0	1	0	0	1	1	0	0	1
1	1	1	1	1	0	0	1	0	0	1	0

Figure 4.1.4 Truth tables for HEX1 and HEX0.

Part 4.2 Karnaugh map for each output function

HEX1[7]

SW[3], SW[2] \ SW[1], SW[0]	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

HEX1[7] = 1

HEX1[6]

SW[3], SW[2] \ SW[1], SW[0]	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

HEX1[6] = 1

HEX1[5]

SW[3], SW[2] \ SW[1], SW[0]	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

HEX1[5] = 1

HEX1[4]

SW[3], SW[2] \ SW[1], SW[0]	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

HEX1[4] = 1

HEX1[3]

SW[3], SW[2] \ SW[1], SW[0]	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

HEX1[3] = 1

HEX1[2]

SW[3], SW[2] \ SW[1], SW[0]	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	0	0	0	0
10	1	1	0	0

HEX1[2] = $\underline{\text{SW[3]}}$ + $\underline{\text{SW[1]}}$ SW[3] $\underline{\text{SW[2]}}$

HEX1[1]

SW[3], SW[2] \ SW[1], SW[0]	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	0	0	0	0
10	1	1	0	0

HEX1[1] = $\underline{\text{SW[3]}}$ + $\underline{\text{SW[1]}}$ SW[3] $\underline{\text{SW[2]}}$

HEX1[0]

SW[3], SW[2] \ SW[1], SW[0]	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

HEX1[0] = 1

HEX0[7]

SW[3], SW[2] \ SW[1], SW[0]	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

HEX0[7] = 1

HEX0[6]

SW[3], SW[2] \ SW[1], SW[0]	00	01	11	10
00	1	1	0	0
01	0	0	1	0
11	0	0	0	0
10	0	0	1	1

HEX0[6]= $\underline{\text{SW[3]}}$ $\underline{\text{SW[2\}}}$ $\underline{\text{SW[1]}}$ + SW[3] $\underline{\text{SW[2]}}$ SW[1] + $\underline{\text{SW[3]}}$ SW[2] SW[1] SW[0]

HEX0[5]

SW[3], SW[2] \ SW[1], SW[0]	00	01	11	10
00	0	1	1	1
01	0	0	1	0
11	1	1	0	0
10	0	0	1	0

HEX0[5]= SW[3] SW[2] $\underline{\text{SW[1]}}$ + $\underline{\text{SW[3]}}$ $\underline{\text{SW[2]}}$ SW[1] + $\underline{\text{SW[3]}}$ $\underline{\text{SW[2]}}$ SW[0] + $\underline{\text{SW[3]}}$ SW[1] SW[0] + SW[3] $\underline{\text{SW[2]}}$ SW[1] SW[0]

HEX0[4]

SW[3], SW[2] \ SW[1], SW[0]	00	01	11	10
00	0	1	1	0
01	1	1	1	0
11	0	1	1	1
10	0	1	1	0

HEX0[4]= SW[0] + $\underline{\text{SW[3]}}$ SW[2] $\underline{\text{SW[1]}}$ + SW[3] SW[2] SW[1]

HEX0[3]

SW[3], SW[2] \ SW[1], SW[0]	00	01	11	10
00	0	1	0	0
01	1	0	1	0
11	0	0	0	1
10	0	0	1	0

HEX0[3] = $\underline{\text{SW[3]}}$ $\underline{\text{SW[2]}}$ $\underline{\text{SW[1]}}$ SW[0] + $\underline{\text{SW[3]}}$ SW[2] $\underline{\text{SW[1]}}$ $\underline{\text{SW[0]}}$ + $\underline{\text{SW[3]}}$ SW[2] SW[1] SW[0] + SW[3] SW[2] SW[1] $\underline{\text{SW[0]}}$ + SW[3] $\underline{\text{SW[2]}}$ SW[1] SW[0]

HEX0[2]

SW[3], SW[2] \ SW[1], SW[0]	00	01	11	10
00	0	0	0	1
01	0	0	0	0
11	1	0	0	0
10	0	0	0	0

HEX0[2] = SW[3] SW[2] $\underline{\text{SW[1]}}$ $\underline{\text{SW[0]}}$ + $\underline{\text{SW[3]}}$ $\underline{\text{SW\{2]}}$ SW[1] $\underline{\text{SW[0]}}$

HEX0[1]

SW[3], SW[2] \ SW[1], SW[0]	00	01	11	10
00	0	0	0	0
01	0	1	0	1
11	0	0	1	0
10	0	0	0	0

HEX0[1] = $\underline{\text{SW[3]}}$ SW[2] $\underline{\text{SW[1]}}$ SW[0] + $\underline{\text{SW[3]}}$ SW[2] SW[1] $\underline{\text{SW[0]}}$ + SW[3] SW[2] SW[1] SW[0]

HEX0[0]

SW[3], SW[2] \ SW[1], SW[0]	00	01	11	10
00	0	1	0	0
01	1	0	0	0
11	0	0	0	1
10	0	0	1	0

HEX0[0] = $\underline{\text{SW[3]}}$ $\underline{\text{SW[2]}}$ $\underline{\text{SW[1]}}$ SW[0] + $\underline{\text{SW[3]}}$ SW[2] $\underline{\text{SW[1]}}$ $\underline{\text{SW[0]}}$ + SW[3] SW[2] SW[1] $\underline{\text{SW[0]}}$ + SW[3] $\underline{\text{SW[2]}}$ SW[1] SW[0]

Figure 4.2.1 Karnaugh maps for each output function with derived equations.

Part 4.3 Verilog Code for Design Implementation

```
//=======================================================
//  This code is generated by Terasic System Builder
//=======================================================

module Lab1_4bits_7seg(

      //////////// SEG7 //////////
      output                 [7:0]       HEX0,
      output                 [7:0]       HEX1,
      output                 [7:0]       HEX2,
      output                 [7:0]       HEX3,
      output                 [7:0]       HEX4,
      output                 [7:0]       HEX5,

      //////////// LED //////////
      output                 [9:0]       LEDR,

      //////////// SW //////////
      input                  [9:0]       SW
);

//=======================================================
//  REG/WIRE declarations
//=======================================================
//=========================================================

//  Structural coding

//=========================================================

// Turn off LEDS

assign LEDR = 10'b0000000000;


// Turn off all the segments not used

assign HEX2 = 8'b11111111;

assign HEX3 = 8'b11111111;

assign HEX4 = 8'b11111111;

assign HEX5 = 8'b11111111;


// set HEX1

assign HEX1[7] = 1'b1;

assign HEX1[6] = 1'b1;

assign HEX1[5] = 1'b1;

assign HEX1[4] = 1'b1;

assign HEX1[3] = 1'b1;

assign HEX1[2] = (~SW[3]) | (SW[3] & (~SW[2]) & (~SW[1]));

assign HEX1[1] = (~SW[3]) | (SW[3] & (~SW[2]) & (~SW[1]));

assign HEX1[0] = 1'b1;


// set hex0

assign HEX0[7] = 1'b1;

assign HEX0[6] = ((~SW[3]) & (~SW[2]) & (~SW[1])) | (SW[3] & (~SW[2]) &
SW[1]) | ((~SW[3]) & SW[2] & SW[1] & SW[0]);

assign HEX0[5] = (SW[3] & SW[2] & (~SW[1])) | ((~SW[3]) & (~SW[2]) & SW[1]) |
((~SW[3]) & (~SW[2]) & SW[0]) | ((~SW[3]) & SW[1] & SW[0]) | (SW[3] &
(~SW[2]) & SW[1] & SW[0]);
```

```
assign HEX0[4] = (SW[0]) | ((~SW[3]) & SW[2] & (~SW[1])) | (SW[3] & SW[2] &
SW[1]);

assign HEX0[3] = ((~SW[3]) & (~SW[2]) & (~SW[1]) & SW[0]) | ((~SW[3]) & SW[2]
& (~SW[1]) & (~SW[0])) | ((~SW[3]) & SW[2] & SW[1] & SW[0]) | (SW[3] & SW[2]
& SW[1] & (~SW[0])) | (SW[3] & (~SW[2]) & SW[1] & SW[0]);

assign HEX0[2] = (SW[3] & SW[2] & (~SW[1]) & (~SW[0])) | ((~SW[3]) & (~SW[2])
& SW[1] & (~SW[0]));

assign HEX0[1] = ((~SW[3]) & SW[2] & (~SW[1]) & SW[0]) | ((~SW[3]) & SW[2] &
SW[1] & (~SW[0])) + (SW[3] & SW[2] & SW[1] & SW[0]);

assign HEX0[0] = ((~SW[3]) & (~SW[2]) & (~SW[1]) & SW[0]) | ((~SW[3]) & SW[2]
& (~SW[1]) & (~SW[0])) | (SW[3] & SW[2] & SW[1] & (~SW[0])) | (SW[3] &
(~SW[2]) & SW[1] & SW[0]);

endmodule
```

Figure 4.3.1 Verilog Code for Design Implementation

Part 4.4 Verilog Code Test Bench of Design Implementation

```
`timescale 1ps/1ps

module tb_majority;

reg [3:0] count;

wire [7:0] HEX0;

wire [7:0] HEX1;

wire [7:0] HEX2;

wire [7:0] HEX3;

wire [7:0] HEX4;

wire [7:0] HEX5;

wire [9:0] LEDR;

wire [9:0] SW;

assign SW[3:0] = count;

// The part of the testbench that instantiates the hardware to be tested

Lab1_4bits_7seg seg7(.HEX0(HEX0), .HEX1(HEX1), .HEX2(HEX2),

.HEX3(HEX3), .HEX4(HEX4), .HEX5(HEX5),
 .LEDR(LEDR), .SW(SW));
// The part of the testbench that creates test input signals
initial begin
      count = 4'b0000;
      repeat (16) begin
            #100
            $display("in = %b, out = %b%b", count, HEX1, HEX0);
            count = count + 4'b0001;
      end
end
endmodule
```

Figure 4.4.1 Verilog Code Test Bench of Design Implementation

```
# in = 0000, out = 11111111000000
# in = 0001, out = 11111111111001
# in = 0010, out = 11111110100100
# in = 0011, out = 11111110110000
# in = 0100, out = 11111110011001
# in = 0101, out = 11111110010010
# in = 0110, out = 11111110000010
# in = 0111, out = 11111111111000
# in = 1000, out = 11111110000000
# in = 1001, out = 11111110010000
# in = 1010, out = 11110011100000
# in = 1011, out = 11110011111001
# in = 1100, out = 11110010100100
# in = 1101, out = 11110010110000
# in = 1110, out = 11110010011001
# in = 1111, out = 1111100110010010
```

Figure 4.4.2 Verilog Code Test Bench Results Report

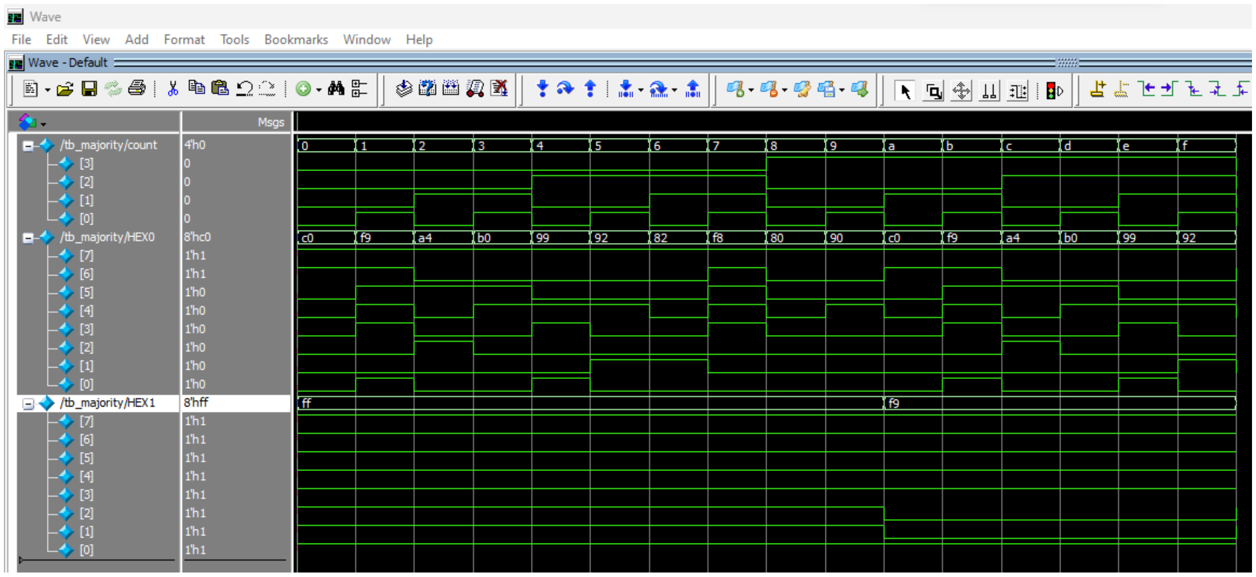


Figure 4.4.3 Verilog Code Test Bench Wave Results

Part 4.5 Testing it in the DE10-Lite board

We programmed the DE10-Lite board and tested the Verilog program. We showed the running circuit to the TA.

Discussion

The test bench code for the sample combinational circuits we created took us a lot of lines of code because the positions of the switches and that they were not related one to the other, so a sequential loop was not appropriate. The test bench code for the 4 bit to 7 segment 2 digits display decoder was shorter because we could use a loop.

We tried both approaches for debugging: the simulation and directly into the hardware. We found that only very little circuits can be debugged directly in hardware. The simulations through the test bench programs make it easier and automatically can assure you of full coverage of the code.

As we mentioned, the Quartus system currently uses Questa instead of ModelSim for Verilog simulations. They work very similar but we referenced the user documentation for the new Questa and were able to do things with a fairly easy conversion from the ModelSim tutorials and procedures specified in the lab.

We used Excel spreadsheets to make reference to the graphical positions of the led segments in the display all the way to the Karnaught maps to minimize errors due to data copying from one point to another. That worked very well for us as the equations came out without errors.

Conclusion

For doing combinational circuits with an FPGA board, the ability to use the Verilog language and simulate the circuit, makes for easier debugging and faster implementation. The use of the Verilog assign statement was enough to make simple combinational circuits that could translate inputs from manual switches to numbers displayed on 7 segment led displays.