BEE 271 Digital circuits and systems
Summer 2016
Lab 2: Hex adding machine

1 Objectives

This lab will present you with a combinatorial logic design problem building a hex adding machine and lead you through the steps to solving it in Verilog.

As shown in figures 1 through 3, your adding machine will:

1. Read inputs A and B as 5-bit binary numbers on the slide switches, displaying them on the LEDs.
2. *Continuously* calculate the 6-bit sum C = A + B.
3. Display A, B and C as 2-digit hex numbers on the 7-segment displays with leading zero suppression.

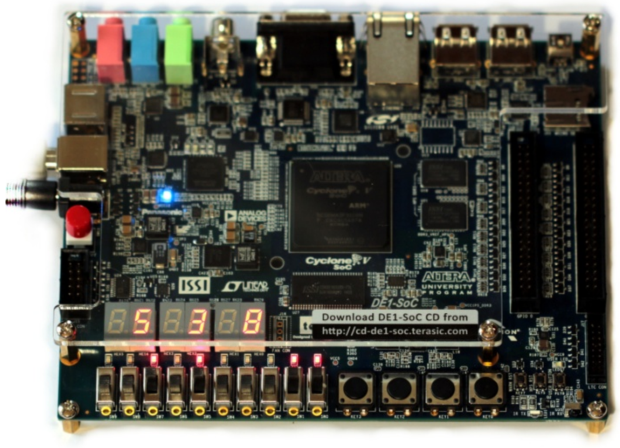


Figure 1. 5 + 3 = 8

By the end of the lab, you should feel fairly comfortable analyzing truth tables and Karnaugh maps, synthesizing combinatorial logic and then writing, compiling and debugging a Verilog module that implements it.

2 Required

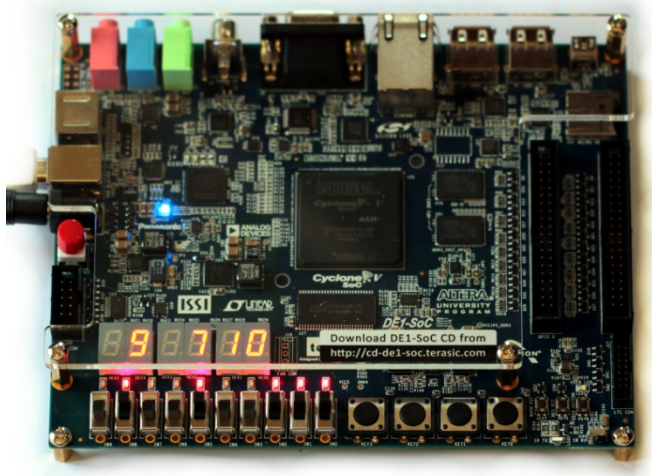


Figure 2. 9 + 7 = 0x10

To do this lab, you will need a Terasic DE0-SoC board and a PC with the following installed.

1. Quartus Prime release 15.1 or 16.0.
2. Altera's USB Blaster driver.
3. The DE1-SoC CDROM, which should be installed in the C:\altera directory.

3 Procedure

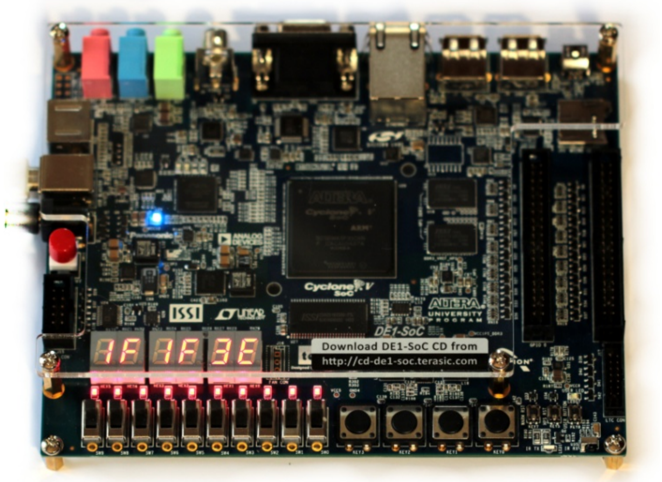


Figure 3. 0x1F + 0x1F = 0x3E

Here are the steps you'll follow.

1. Create a truth table specification for a seven-segment display driver.
2. Use Karnaugh maps to write the combinatorial equations for each of the segments.
3. Use Quartus II and the System Builder tool to create an empty Verilog project.
4. Create a deliberately trivial Verilog program that wires the switches to the LEDs and to the segments of one of the displays to verify how they work and that you can successfully compile a Verilog program and get it to run on the DE1-SoC board.
5. Write and debug a hex to 7-segment decoder module that implements the combinatorial equations you wrote and use it to display the hex value on switches SW[3:0] on one of the displays.
6. Add additional logic to add A + B and to suppress leading zeros.

4 Demo and report

Once you have you have your design working, you must demo it. You must also submit a report containing the following. This is all you need to submit. Your truth table and Karnaugh maps can be hand-drawn and should be submitted as a PDF.

1. Your truth table.
2. Karnaugh maps and equations for each segment.
3. Your Verilog program as a .v (Verilog) or .sv (SystemVerilog) file.

5 7-segment displays

In this step, you'll develop the logic equations for a 7-segment decoder.

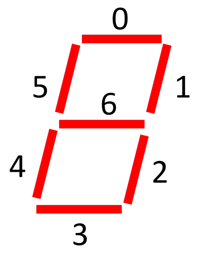


Figure 4. Numbering of the segments.

5.1 Truth table

The individual segments in 7-segment display are numbered from 0 at the top, clockwise around the outside, then the middle, as shown in figure 4.

Your first step is to fill in the rest of the truth table in figure 5, indicating which segments should light up for each hex value from 0x0 to 0xF. The values for s0 are already filled in. You will need to do the rest. Hex values 0xB and 0xD should display as lower-case b and d.

b3	b2	b1	b0	Hex	s0	s1	s2	s3	s4	s5	s6
0	0	0	0	0	1						
0	0	0	1	1	0						
0	0	1	0	2	1						
0	0	1	1	3	1						
0	1	0	0	4	0						
0	1	0	1	5	1						
0	1	1	0	6	1						
0	1	1	1	7	1						
1	0	0	0	8	1						
1	0	0	1	9	1						
1	0	1	0	A	1						
1	0	1	1	B	0						
1	1	0	0	C	1						
1	1	0	1	D	0						
1	1	1	0	E	1						
1	1	1	1	F	1						

Figure 5. Truth table for the individual segments.

5.2 Karnaugh maps

The second step is to copy the desired values for each output s0 through s6 into a 4-variable Karnaugh map and then use it to write a sum of products or product of sums equation.

Figure 6 shows how this might be done for s0. Based on the groups chosen:

s0 = b1 b2 + b1' b2' b3 + b0' b3 +
    b0' b2' + b0 b2 b3' + b1 b3'

You will need to work out the equations for s1 through s6. ***Please do not just copy them off the internet!***

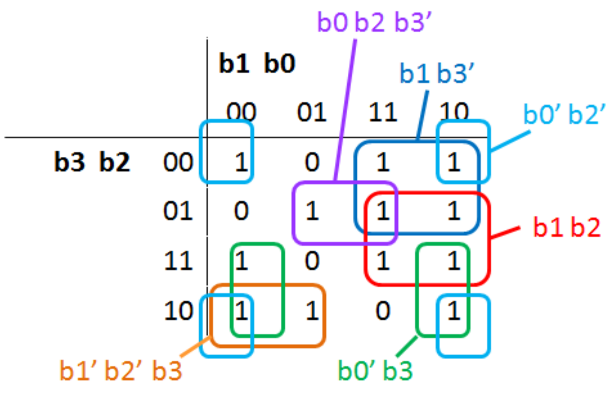


Figure 6. A Karnaugh map for s0.

6 Create a project in Quartus Prime

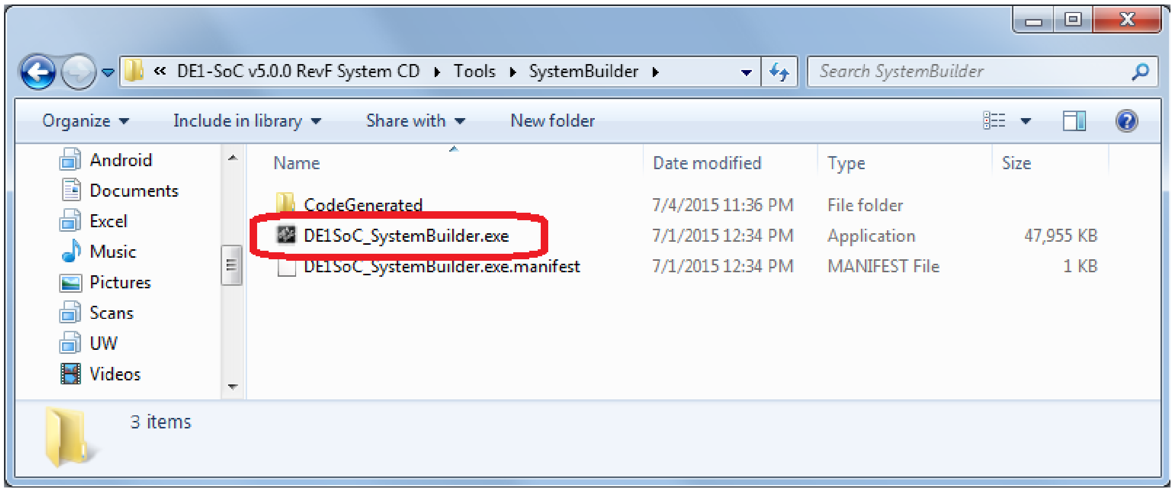
In this step, you'll create a trivial skeleton Verilog project in Quartus that ties the slide switches to the LEDs and to the segments of one of the displays. You'll then program the board and observe which LEDs and segments turn on as you flip the switches.

These are the steps.

1. Use the DE1-SoC SystemBuilder tool to create a skeleton project with the inputs and outputs you need.
2. Open it in Quartus and add the skeleton Verilog file to the project. (SystemBuilder generates this file but doesn't add it to the project automatically.)
3. Edit the skeleton Verilog to connect the switches, LEDs and one of the displays.
4. Compile your project.
5. Program the device.
6. Verify that everything works as expected.

6.1 Use SystemBuilder to create a skeleton

The first step is to use the System Builder tool create a skeleton project with inputs and outputs connected to the devices on the DE1-SoC board that your logic needs to use.

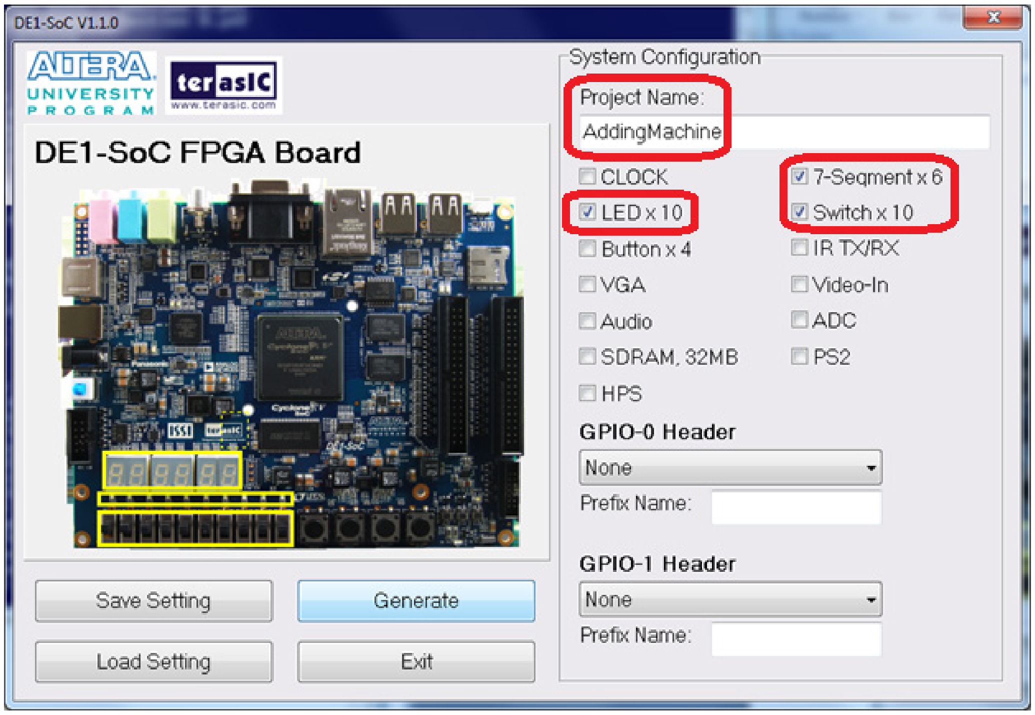


Name	Date modified	Type	Size
CodeGenerated	7/4/2015 11:36 PM	File folder	
DE1SoC_SystemBuilder.exe	7/1/2015 12:34 PM	Application	47,955 KB
DE1SoC_SystemBuilder.exe.manifest	7/1/2015 12:34 PM	MANIFEST File	1 KB

Figure 7. SystemBuilder in the Tools directory on DE1-SoC System CD.

SystemBuilder is on the System CD in the Tools\SystemBuilder directory. You can ignore any security warnings when you start it.

System Builder knows what devices are on a DE1-SoC board and lets you check the ones you want to use. You will need to use the LEDs, the 7-segment displays and the slide switches, as shown in figure 8. Give your project any name you like. I called mine AddingMachine. Save it into a separate directory.


DE1-SoC V1.1.0

DE1-SoC FPGA Board

System Configuration

Project Name: AddingMachine

- [ ] CLOCK
- [✓] 7-Segment x 6
- [✓] LED x 10
- [✓] Switch x 10
- [ ] Button x 4
- [ ] IR TX/RX
- [ ] VGA
- [ ] Video-In
- [ ] Audio
- [ ] ADC
- [ ] SDRAM, 32MB
- [ ] PS2
- [ ] HPS

GPIO-0 Header: None

Prefix Name:

GPIO-1 Header: None

Prefix Name:

Figure 8. Using SystemBuilder to create a project named AddingMachine using the LEDs, 7-segment displays and the ten slide switches.

6.2 Open it in Quartus Prime

Start Quartus Prime and open the project you created with SystemBuilder as shown in figure 10.

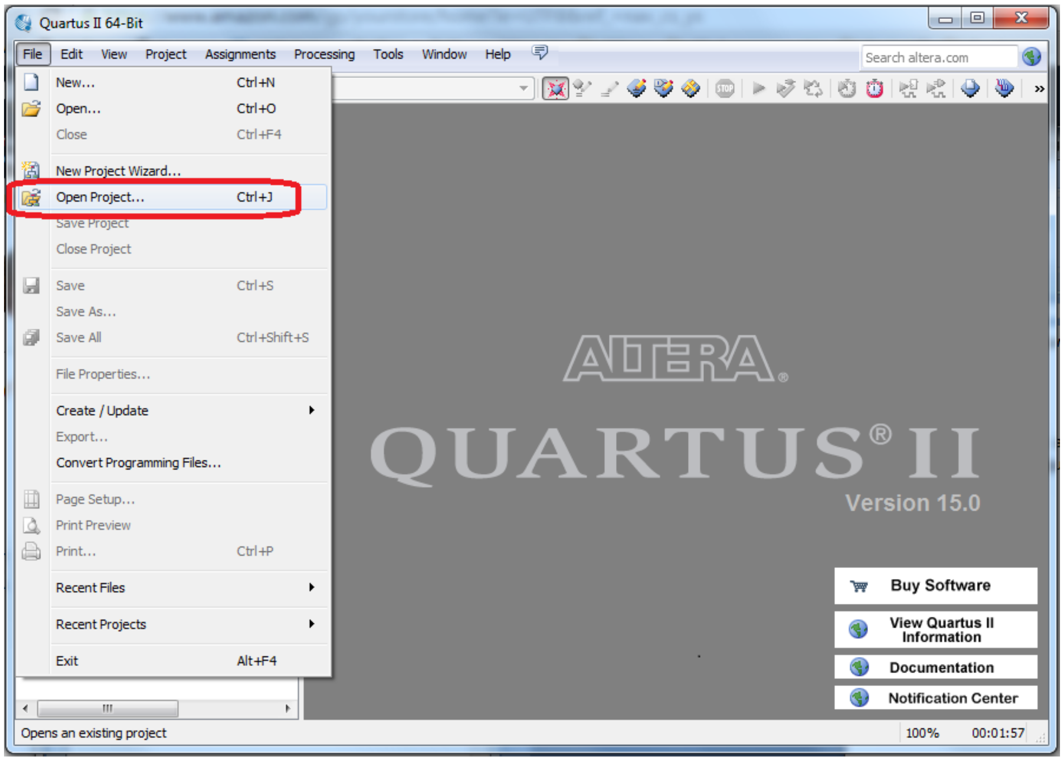


Figure 10. Start Quartus and open the project you created with SystemBuilder.

Initially, there's nothing in the project except a mostly empty .SDC (Synoptics Design Constraint) file that can be used (not by us) to specify timing constraints.

SystemBuilder creates a Verilog skeleton but doesn't add it to the project automatically, presumably because there's nothing in it yet except the module interface definition. You'll need to add it yourself, as shown in figure 11.

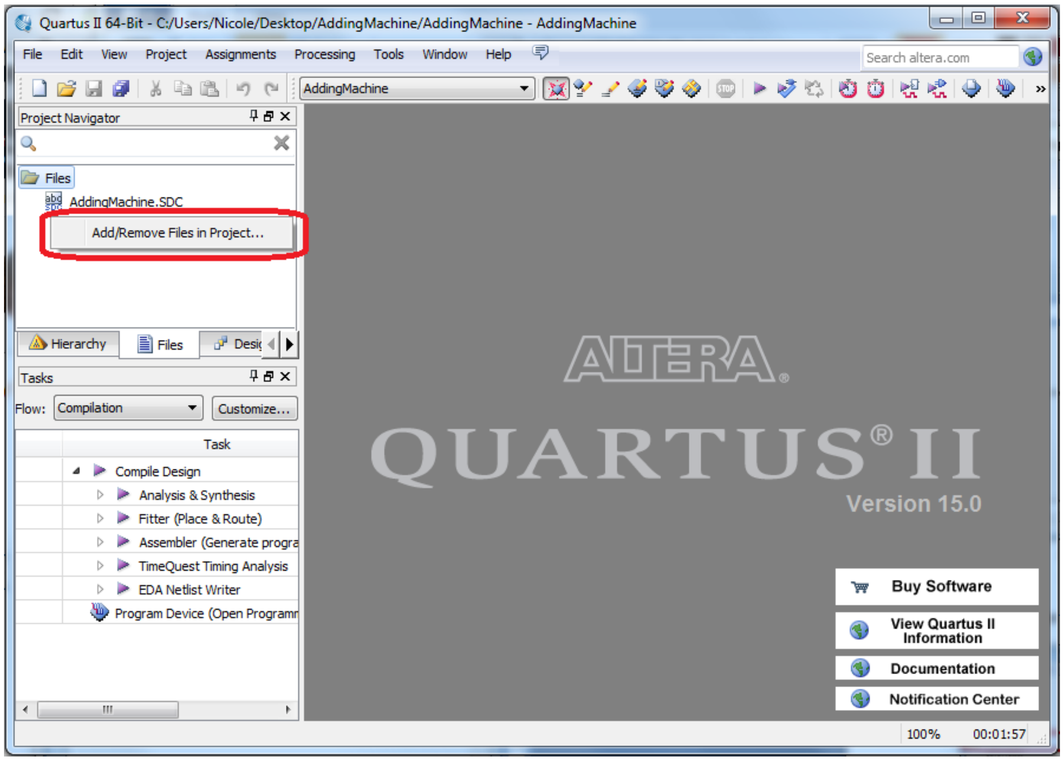
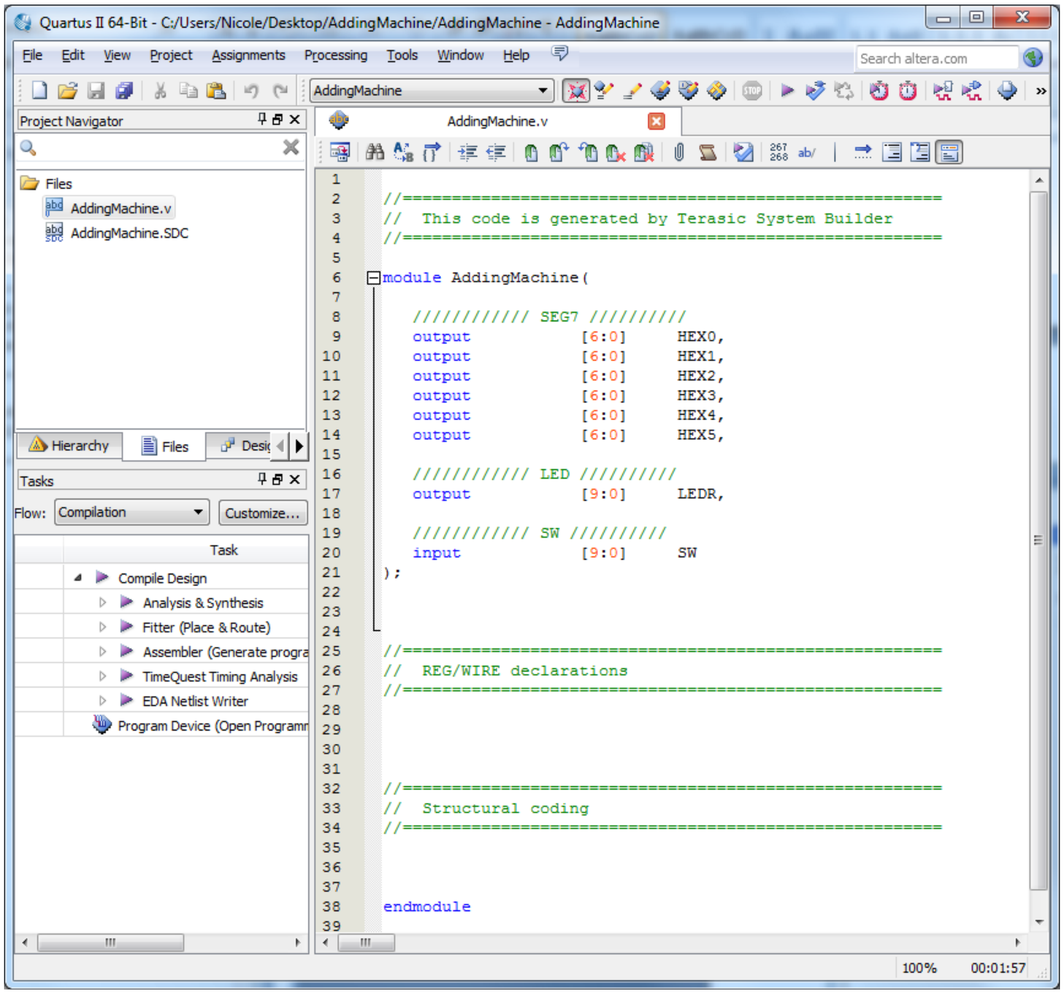

Quartus II 64-Bit - C:/Users/Nicole/Desktop/AddingMachine/AddingMachine - AddingMachine
File Edit View Project Assignments Processing Tools Window Help
Search altera.com
AddingMachine
Project Navigator
Files
AddingMachine.SDC
Add/Remove Files in Project...
Hierarchy
Files
Desi
Tasks
Flow: Compilation
Customize...
Task
Compile Design
Analysis & Synthesis
Fitter (Place & Route)
Assembler (Generate progra
TimeQuest Timing Analysis
EDA Netlist Writer
Program Device (Open Programr
ALTERA
QUARTUS® II
Version 15.0
Buy Software
View Quartus II Information
Documentation
Notification Center
100%
00:01:57

Figure 11. Add your Verilog file.

6.3 Connect the switches, LEDs and segments

In this step, you'll wire up your module. The only part we care about in the empty skeleton is the module definition and the list of arguments. You can delete some of the rest of the comments.

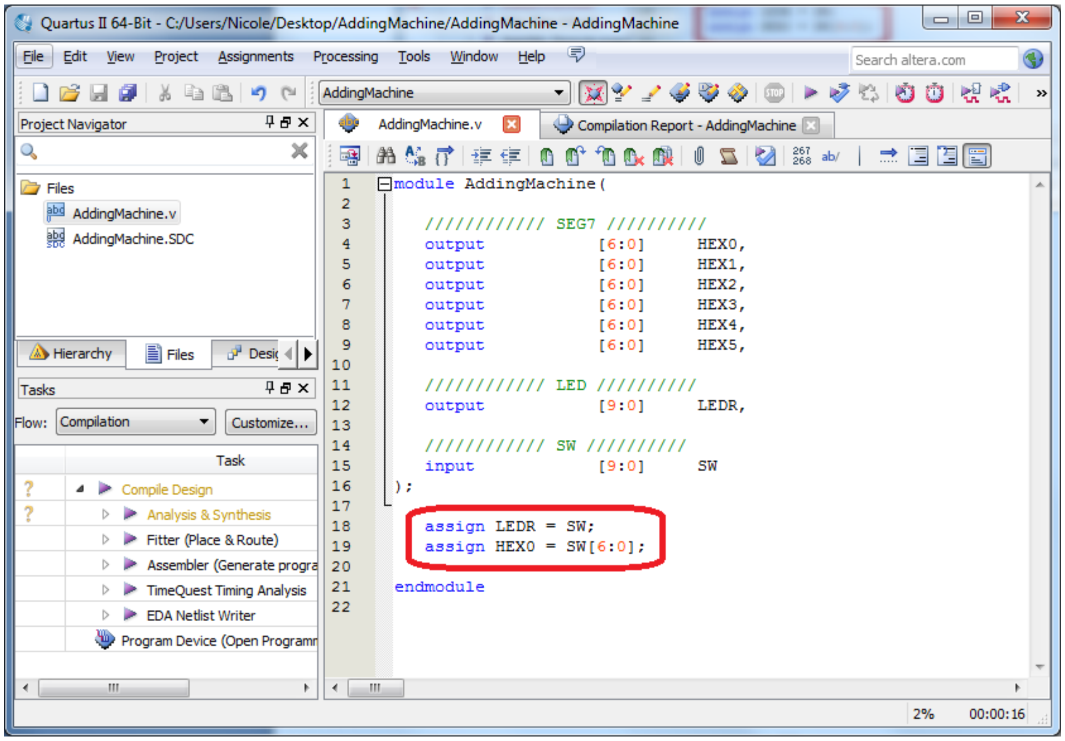

Quartus II 64-Bit - C:/Users/Nicole/Desktop/AddingMachine/AddingMachine - AddingMachine
File Edit View Project Assignments Processing Tools Window Help
Project Navigator
Files
AddingMachine.v
AddingMachine.SDC
Hierarchy
Files
Tasks
Flow: Compilation
Customize...
Task
Compile Design
Analysis & Synthesis
Fitter (Place & Route)
Assembler (Generate progra
TimeQuest Timing Analysis
EDA Netlist Writer
Program Device (Open Programr
100%
00:01:57

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

module AddingMachine(

    //////////// SEG7 //////////
    output          [6:0]     HEX0,
    output          [6:0]     HEX1,
    output          [6:0]     HEX2,
    output          [6:0]     HEX3,
    output          [6:0]     HEX4,
    output          [6:0]     HEX5,

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

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


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




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



endmodule
```

Figure 14. The empty skeleton with machine-generated comments.

Wire the switches to the LEDs and to the segments of one of the displays by adding these lines, as shown in figure 15, and then save your file.

```
assign LEDR = SW;
assign HEX0 = SW[6:0];
```

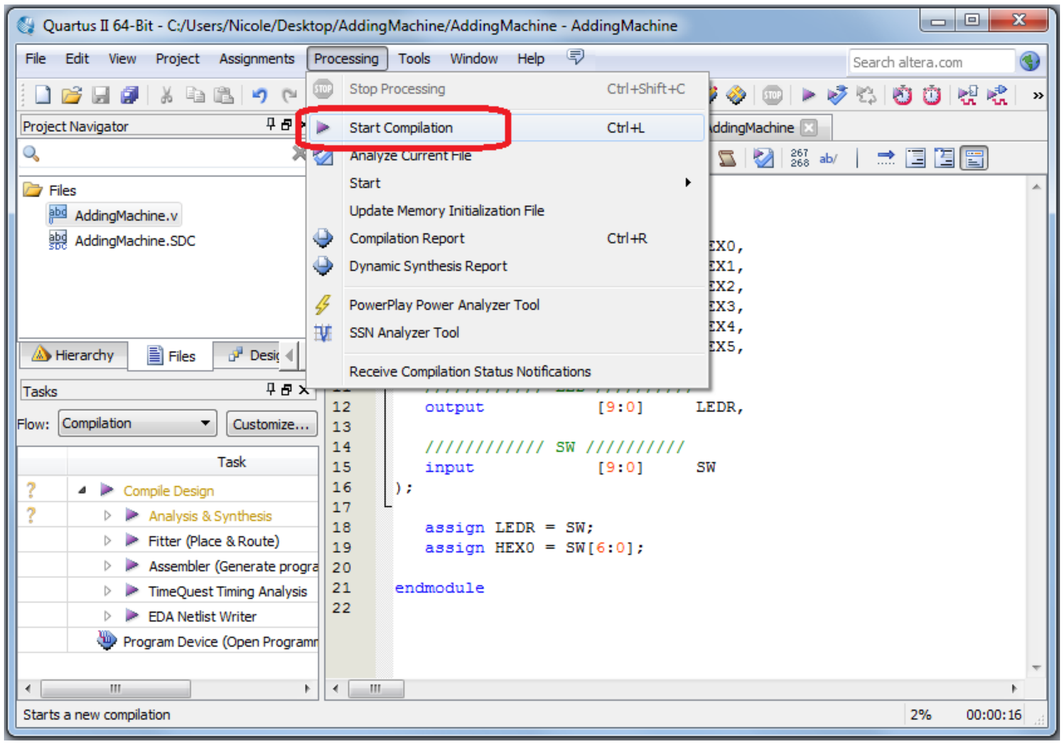

Quartus II 64-Bit - C:/Users/Nicole/Desktop/AddingMachine/AddingMachine - AddingMachine
File Edit View Project Assignments Processing Tools Window Help
Search altera.com
AddingMachine
Project Navigator
Files
AddingMachine.v
AddingMachine.SDC
Hierarchy
Files
Desig
Tasks
Flow: Compilation
Customize...
Task
Compile Design
Analysis & Synthesis
Fitter (Place & Route)
Assembler (Generate progra
TimeQuest Timing Analysis
EDA Netlist Writer
Program Device (Open Programr
AddingMachine.v
Compilation Report - AddingMachine
2%
00:00:16

```
module AddingMachine(

    //////////// SEG7 //////////
    output          [6:0]       HEX0,
    output          [6:0]       HEX1,
    output          [6:0]       HEX2,
    output          [6:0]       HEX3,
    output          [6:0]       HEX4,
    output          [6:0]       HEX5,

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

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

    assign LEDR = SW;
    assign HEX0 = SW[6:0];

endmodule
```

Figure 15. Delete the extraneous comments and add statements to connect the switches to the LEDs and to the HEX0 7-segment display.

6.4 Compile

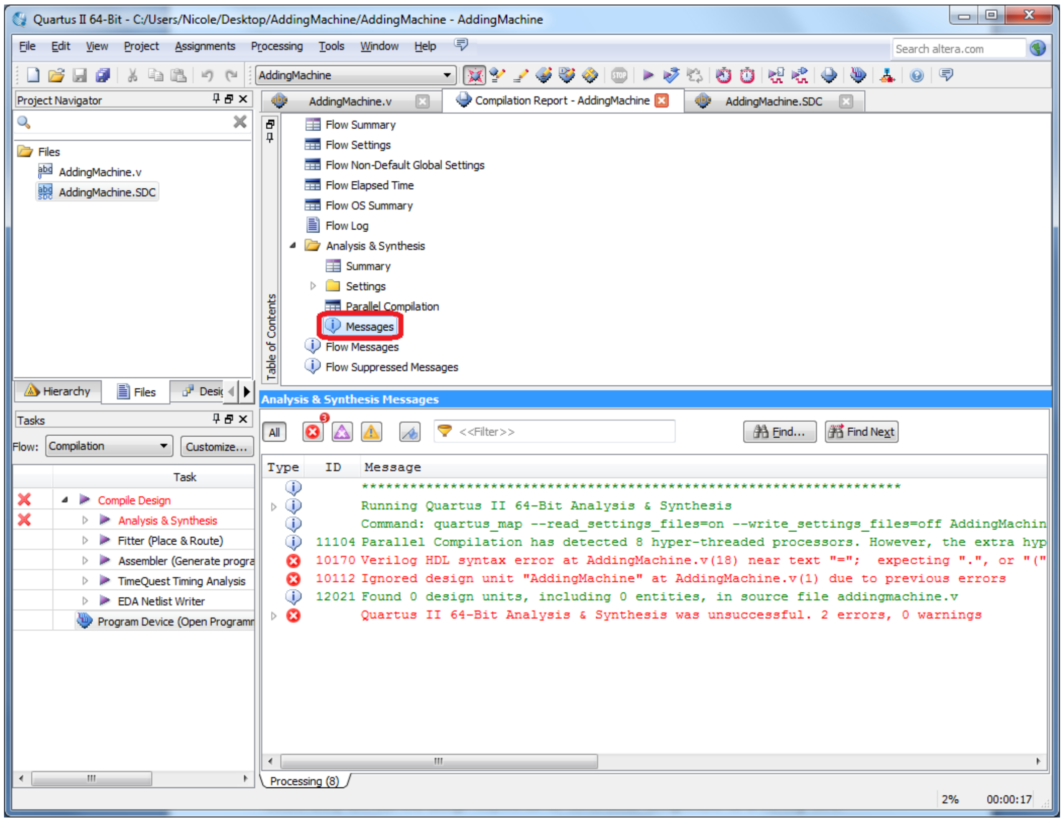
The next step is to compile your program, as shown in figure 16.


Quartus II 64-Bit - C:/Users/Nicole/Desktop/AddingMachine/AddingMachine - AddingMachine
File Edit View Project Assignments Processing Tools Window Help
Search altera.com
Stop Processing Ctrl+Shift+C
Start Compilation Ctrl+L
Analyze Current File
Start
Update Memory Initialization File
Compilation Report Ctrl+R
Dynamic Synthesis Report
PowerPlay Power Analyzer Tool
SSN Analyzer Tool
Receive Compilation Status Notifications
Project Navigator
Files
AddingMachine.v
AddingMachine.SDC
Hierarchy
Files
Tasks
Flow: Compilation
Customize...
Task
Compile Design
Analysis & Synthesis
Fitter (Place & Route)
Assembler (Generate progra
TimeQuest Timing Analysis
EDA Netlist Writer
Program Device (Open Programr
Starts a new compilation
2%
00:00:16

```
                                  EX0,
                                  EX1,
                                  EX2,
                                  EX3,
                                  EX4,
                                  EX5,

    output              [9:0]     LEDR,

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

    assign LEDR = SW;
    assign HEX0 = SW[6:0];

endmodule
```

Figure 16. Compile your program.

If you have any syntax or other errors, you'll see something like figure 17, where I've deliberately left out the assign keywords for continuous assignment to a Verilog "wire". (You do that only with a "reg" temporary variable type.) To read the compiler error messages, go to the Compilation Report → Analysis & Synthesis section and click on messages, which will open in another tile. Fix your mistake and try again.

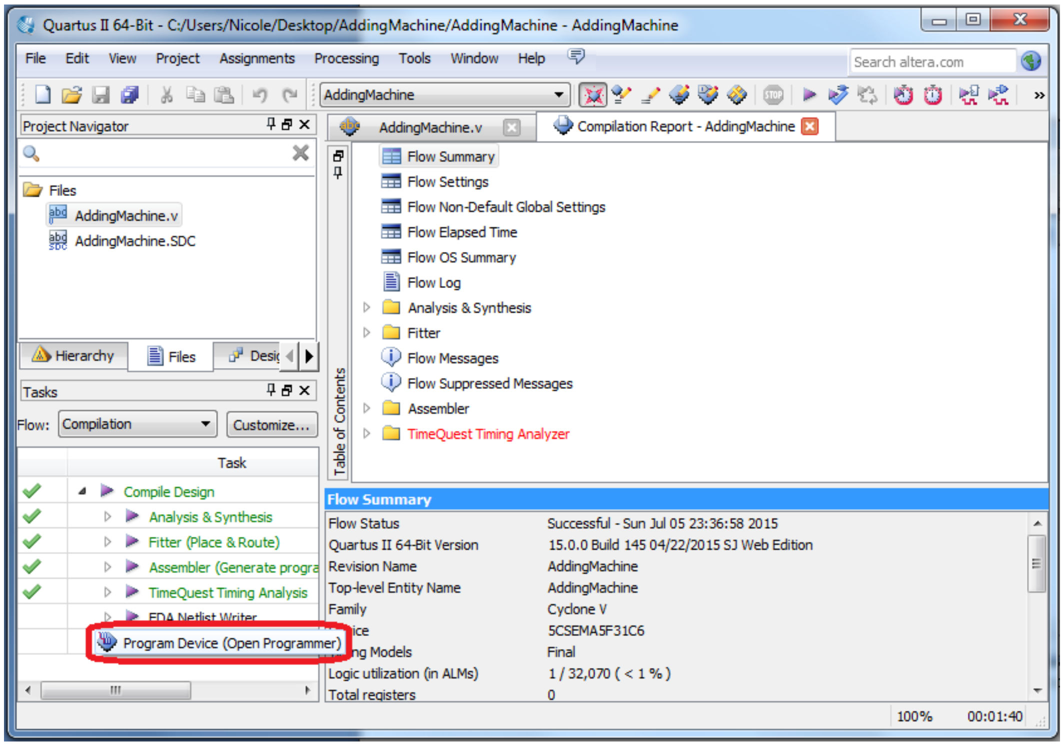


```
Type  ID    Message
            *******************************************************************
            Running Quartus II 64-Bit Analysis & Synthesis
            Command: quartus_map --read_settings_files=on --write_settings_files=off AddingMachin
      11104 Parallel Compilation has detected 8 hyper-threaded processors. However, the extra hyp
      10170 Verilog HDL syntax error at AddingMachine.v(18) near text "=";  expecting ".", or "("
      10112 Ignored design unit "AddingMachine" at AddingMachine.v(1) due to previous errors
      12021 Found 0 design units, including 0 entities, in source file addingmachine.v
            Quartus II 64-Bit Analysis & Synthesis was unsuccessful. 2 errors, 0 warnings
```

Figure 17. Error messages.

6.5 Program the device

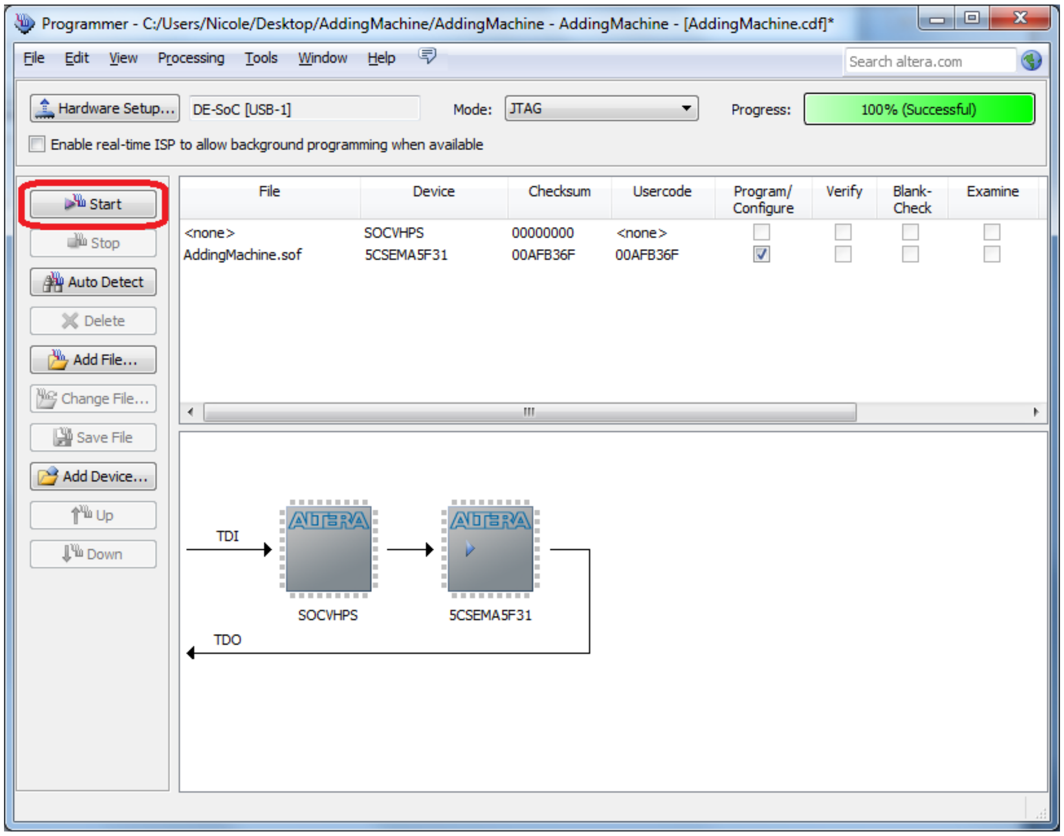
Once you have a successful compile, you can try programming the device, as shown in figures 18 and 19. You may need to auto detect the DE1-SoC board, in which case you'll need to be sure you end up with the configuration shown in figure 19, with the SOCVHPS (hard processor system) and 5CSEMA5F31 (Cyclone V FPGA).



Flow Summary

Flow Status	Successful - Sun Jul 05 23:36:58 2015
Quartus II 64-Bit Version	15.0.0 Build 145 04/22/2015 SJ Web Edition
Revision Name	AddingMachine
Top-level Entity Name	AddingMachine
Family	Cyclone V
ice	5CSEMA5F31C6
ng Models	Final
Logic utilization (in ALMs)	1 / 32,070 (< 1 %)
Total registers	0

Figure 18. After successful compile, program the device.


Programmer - C:/Users/Nicole/Desktop/AddingMachine/AddingMachine - AddingMachine - [AddingMachine.cdf]*
File Edit View Processing Tools Window Help
Search altera.com
Start
Stop
Auto Detect
Delete
Add File...
Change File...
Save File
Add Device...
Up
Down
TDI
TDO
SOCVHPS
5CSEMA5F31

Hardware Setup... DE-SoC [USB-1] Mode: JTAG Progress: 100% (Successful)

☐ Enable real-time ISP to allow background programming when available

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank- Check	Examine
<none>	SOCVHPS	00000000	<none>	☐	☐	☐	☐
AddingMachine.sof	5CSEMA5F31	00AFB36F	00AFB36F	☑	☐	☐	☐

Figure 18. Device programmer.

6.6 Test

Once you have board programmed, verify that it works as you expect.

Notice that the LEDs are active high (they go on when the switch is on) but that the display segments are active low (they go on when the switch is off). This is often done because power consumption is less for signals in the high state if they use a TTL interface, as these parts likely do. (Remember that a float looks like a 1, so it can't possibly require any current.)

This means that whatever equations you've worked out for s0 through s6 will need to be inverted at the end.

7 Create your 7-segment decoder

In this section, you'll create Verilog module which will take 4 bits = 1 hex digit as input and generate outputs to drive the individual segments of one display.

7.1 Basic skeleton

Here is the basic skeleton you will need to fill in. The equation for s0 has already been rewritten into Verilog syntax. You will need to fill in the rest. Notice that the outputs are inverted at the end for active low.

```
module SevenSegment( input [3:0] hexDigit, output [6:0] segments );

   wire b0, b1, b2, b3;
   wire [6:0] s;

   assign b0 = hexDigit[0];
   assign b1 = hexDigit[1];
   assign b2 = hexDigit[2];
   assign b3 = hexDigit[3];

   assign s[0] = b1 & b2 | ~b1 & ~b2 & b3 | ~b0 & b3 |
                 ~b0 & ~b2 | b0 & b2 & ~b3 | b1 & ~b3;

   assign segments = ~s;

endmodule
```

This module can be instantiated in your AddingMachine module and hooked up to some switches and one of the displays as follows.

```
   SevenSegment Hex0( SW[3:0], HEX0 );
```

7.2 Debug

Debug your design by verifying that as you flip switches, all the correct segments light up.

8 Create the adding machine

The final step is to turn your design into an adding machine that performs as shown in figures 1 through 3.

1. Modify your design to wire up all the switches and all the displays with additional instances of your SevenSegment module.
2. Implement the add function. It's built in to Verilog as the + operator but you'll need to consider how to add two 5-bit numbers from the switches to get a 6-bit result and how to split 5 and 6-bit numbers into two hex digits.
3. Verify that you have this working as an adding machine.
4. Modify your design to suppress leading zeros.
5. Demo your design and turn in your report.