

MSc Course on Analogue and Digital IC Design  
Imperial College London

## Laboratory Experiment – Mastering Digital Design

(webpage: <https://github.com/Mastering-Digital-Design/Lab-Module>)

This Laboratory Experiment is intended to ensure all students on the ADIC MSc course reach an expected level of competence in digital design using a hardware description language (such as SystemVerilog) implemented on an FPGA. Those students with limited or no prior experience in both SystemVerilog and/or FPGA may need to devote time outside the scheduled period to complete the experiment.

The experiment is organized in four parts, each with clearly defined learning outcomes. See the document entitled: “*Mastering Digital Design – Experiment Specification Document*”.

### Objectives

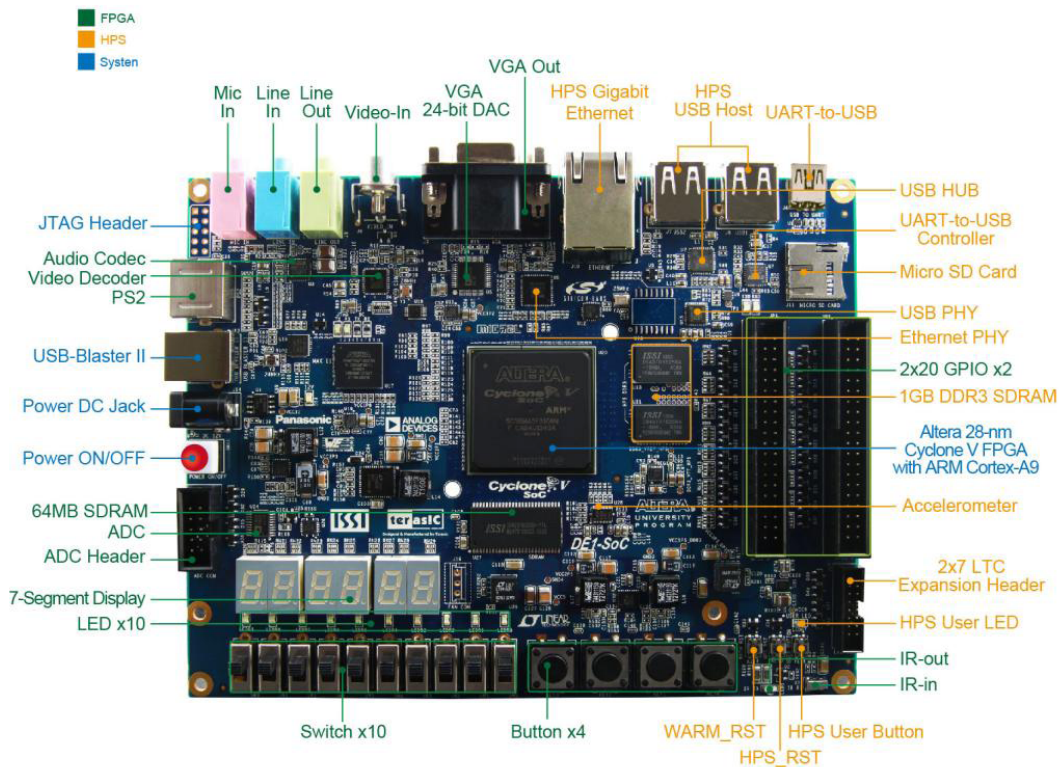
By the end of this experiment, you should have learned:

- How to design digital circuits using Intel/Altera’s Quartus II Design software;
- How to design digital circuits targeting Altera’s Cyclone V FPGA using Terasic’s DE1-SoC Board;
- How to design digital circuits in efficient, synthesizable SystemVerilog HDL;
- How to evaluate your design in terms of resource utilization and clock speed;
- How to use the DE1-SoC FPGA board with its custom daughter board for analogue I/O functions;
- Have designed something yourself for the Cyclone V FPGA.

### Before you start

Before you come to the laboratory, you are expected to:

- Have understood the lectures on SystemVerilog
- Be familiar with the basic architecture inside the FPGA
- Have read through this Laboratory instructions



Both the experimental board and a PC would be made available to you during your allotted period in the laboratory. In addition, you may also borrow a DE1-SoC board to use at home for the duration of this experiment.

This instruction manual is divided into four parts, one for each week. Each part has its own goals and learning outcomes.

Some students will find this experiment harder or easier than average, depending on your prior experience with digital logic. Therefore, the four Lab sessions contain the compulsory as well as optional exercises. If you fall behind this experiment during any week, it is wise to find a bit of spare time to catch up outside the official laboratory sessions and restrict yourself to the compulsory parts only.

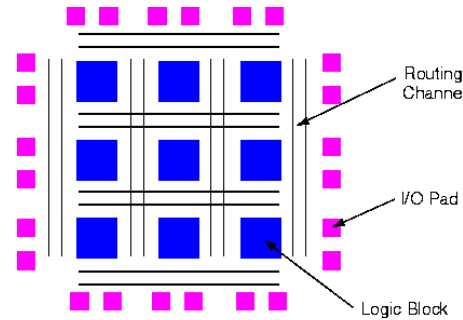
## PART I – Schematic to Verilog (Learning Outcomes)

Basic competence in using Intel/Altera's Quartus design systems for Cyclone-V FPGA; appreciate the superiority of hardware description language over schematic capture for digital design; use of case statement to specify combinatorial circuit; use higher level constructs in SystemVerilog to specify complex combinatorial circuits; develop competence in taking a design from description to hardware.

### 1.0 Introduction

You should have done some background reading before attending the laboratory session as suggested at the Lecture.

FPGAs is a type of programmable logic devices introduced by Xilinx in 1985. It is now the predominant technology for implementing digital logic in low to moderate volume production. The basic structure of an FPGA is shown below. It consists of three main types of resources: 1) Logic Blocks (or Elements); 2) Routing Resources; 3) I/O Pad. For more information about FPGA, see Lecture 1 notes available on the E2 Digital Electronics course webpage.



#### 1.1 Quartus Design Suite

Quartus provides a complete environment for you to implement your design on an Altera FPGA. It supports all aspects of the design flow, which is typically following the flow diagram shown here. The best way to learn Quartus is to go through this experiment step-by-step. After you have learned the basics, you can start to explore other aspects of the Quartus system.

#### 1.2 DE1-SoC Board

DE1-SoC Board is designed and made by Terasic. It is based around a Cyclone V FPGA from Altera. Include on the DE1 board are various I/O devices such as 7-segment LED displays, LED, switches, VGA port, RS232 port, SD card slot etc. A block diagram of the DE1 board is shown below. Although the Cyclone V includes a dual-core ARM processor, we will only be using the FPGA part of the FPGA for this experiment.

#### 1.4 SystemVerilog Hardware Description Language

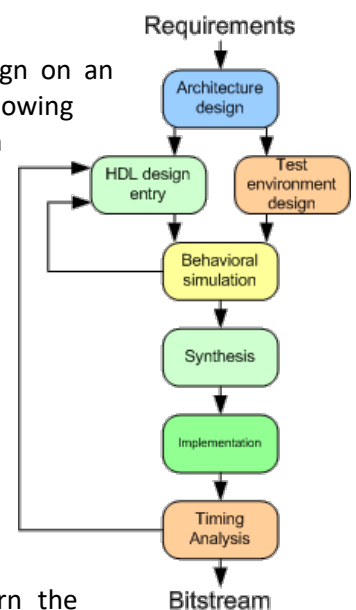
One of the key learning objectives of this experiment is for you to learn the SystemVerilog Hardware Description Language (SV), which is commonly used to specify FPGA and other types of chip designs. An excellent tutorial can be found on:

<https://www.asic-world.com/systemverilog/tutorial.html>

#### 1.5 Using Quartus Prime Lite software and DE1 at home

If your own laptop is sufficiently powerful (at least 4GB of RAM) and has plenty of free disk space (at least 1GB of free disk space), you may want to install a copy of the Quartus design software on your own computer. It is called Quartus Prime Lite. The latest version that supports Cyclone V FPGA is version 23. The earliest is version 18. You can download and use the software free on:

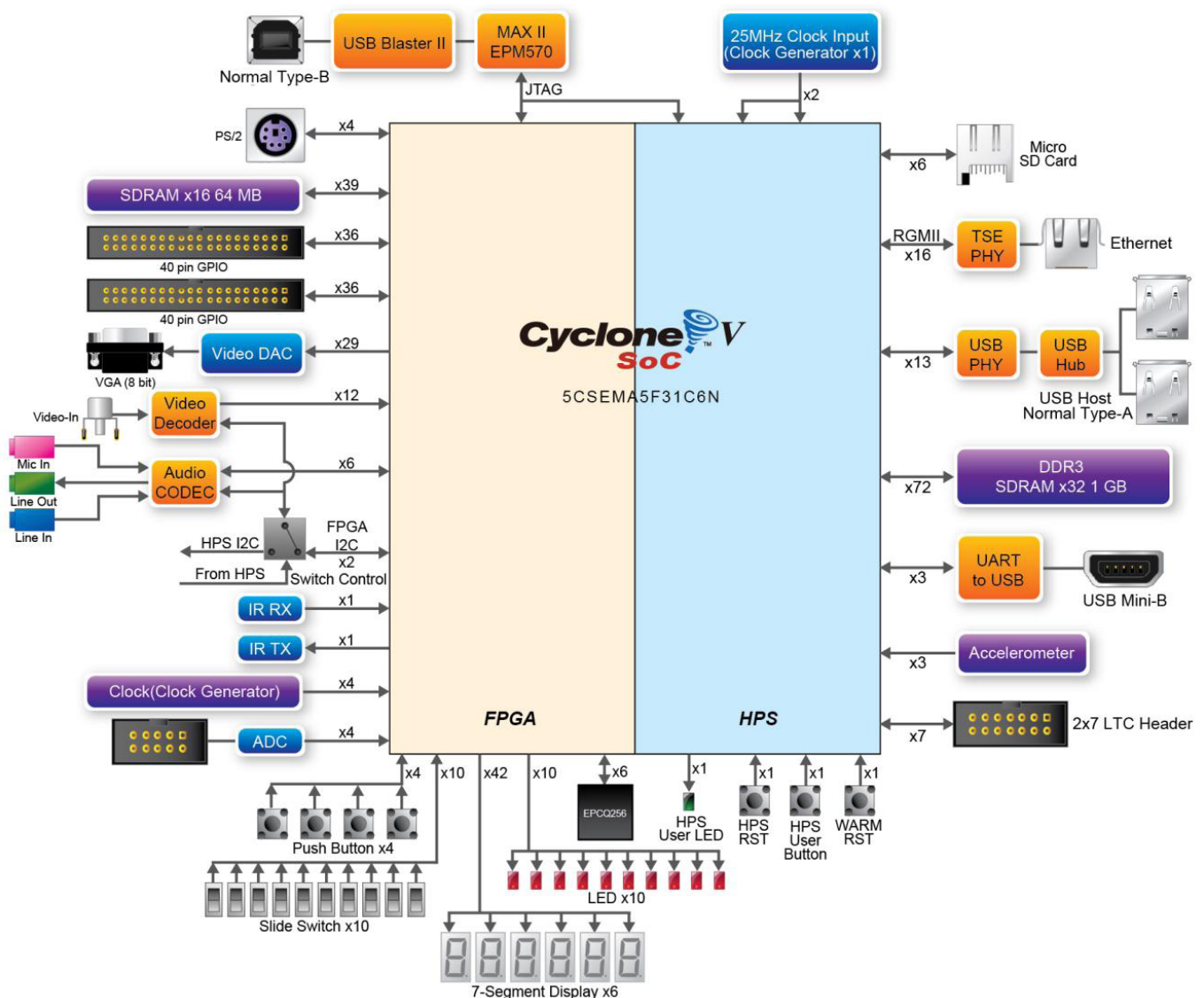
<https://www.intel.com/content/www/us/en/software-kit/795188/intel-quartus-prime-lite-edition-design-software-version-23-1-for-windows.html>



Your class is divided into Groups of two students. You may choose your own group partner. If you don't find anyone to work with, let Dr Aaron Zhao know and he will pair you up with someone. Each pair of students shares one DE1-SoC board with an analogue I/O board for the duration of this experiment. You can check one out with your ID card from Level 1 stores. There may be sufficient DE1-SoC boards available for some of pairs to have one each depending how many we can spare. You may take the board home with you or use it in Room 909 (MSc Lab). However, you are responsible for the board until it is returned at the end of the term.

To install your own copy of Quartus Prime Lite, you should go to Altera's website to register, then download the free Quartus Prime Light from [here](#). Note that Quartus and the DE1 board only works with MS Windows or Linux.

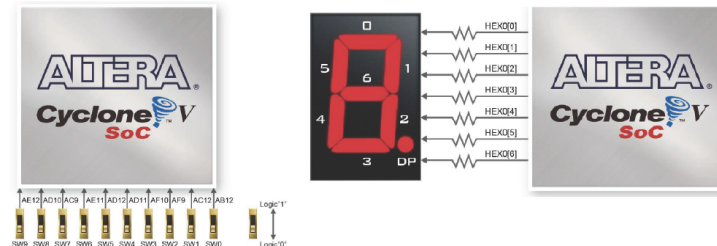
Plug the DE1 board to a USB port on your computer and turn it ON (red button). It will ask you for a device driver, which can be found in the Quartus software directory ....\drivers. See "DE1-SoC Getting Started Guide" from the experiment webpage.



Block Diagram of the DE1-SoC Board

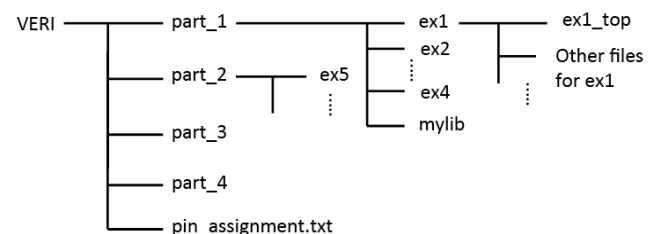
## Experiment 1: Schematic capture using Quartus – 7-Segment Display

If you have come to the laboratory session prepared, Part I of experiment VERI should take no more than ONE 3-hour session. It will lead you through the entire design of a 7-segment decoder using schematic entry method. It will use four slide switches on the right (SW3 to SW0) on the DE1 board as input, and display the 4-bit binary number as a hexadecimal digit on the right-most 7-segment display (HEX0).



### Step 1: Creating a good directory structure

Before you start carrying out any design for this exercise, it would be very helpful if you first create in your home directory a directory structure on the h:\ drive for this experiment. Shown on the right is a possible directory structure that you may choose to create. Each folder is empty for now, but as you progress through the four Lab Sessions, you will be creating each design in each of the folders.



### Step 2: See what you are aiming for

Go to the Experiment webpage (see above) and download a copy of the solution for Exercise 1: “**ex1sol.sof**” to your home directory (or wherever that is). Now turn ON the DE1 board.

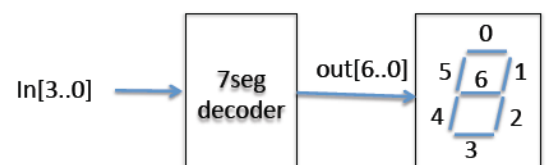
### Step 3: Programme the FPGA

Start up Quartus software on your computer. Click command: **Tools > Programmer**. In the popup window, click: **Hardware Setup ....** You should see something like the diagram on the right. Then select: **DE-SOC [USB-1]**. This is to tell Quartus software that you are using the DE1-SoC USB interface to program (or blast) the FPGA. Then click **Auto Detect** button on the left. A window will pop up and you need to select SCSEMA5 radio button to tell the system which type of Cyclone V FPGA chip you are using (which is 5CSEMA5).

You will now see two lines in the Programmer window as shown on the right. Since we are only configuring (i.e. sending a bit-stream to) the FPGA part of the Cyclone V chip, we need to **delete** the SOCVHPS (stands for System-on-Chip V High Performance System, which is the ARM processor) from the programmer set up.

Next click the **AddFile** button. Navigate to the folder containing the **ex1sol.sof** file. Select this. Finally click the **Start** button.

The **ex1sol.sof** file contains the solution to Exercise 1 designed by me. It has the bit-stream to configure (or





programme) the FPGA part of Cyclone V. Once the bit-stream is successfully sent to the FPGA chip, this design will take over the function of the chip. You should be able to change the least significant four switches and see a hexadecimal number displayed on rightmost 7-segment display.

You should leave the programmer utility running in the background for ease of sending another design to the FPGA later. Return to Quartus software by clicking its window.

#### Step 4: Paper Design

The overall block diagram for the decoder is shown below. The decoder outputs **out[6..0]** drive the seven segments on the display. Note that the LED segments are **low active**, meaning that the LED will light up (ON) if the corresponding digital signal is at 0V.

The truth-table for the decoder is shown here:

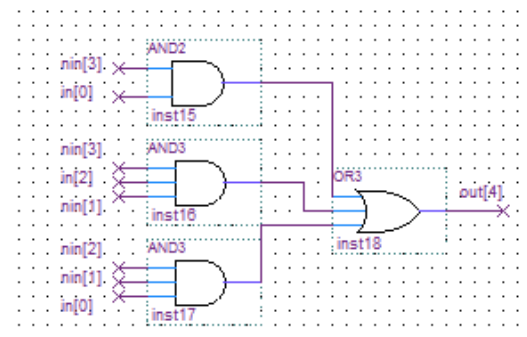
With what you have learned in the first year, you should be able to design the decoder in the form of seven Boolean equations, and then use K-map to minimise the logic. In order to save time, only derive the Boolean equation for out[4] as a Boolean function of in[3..0].

in[3..0]	out[6:0]	Digit	in[3..0]	out[6:0]	Digit
0000	1000000	0	1000	0000000	8
0001	1111001	1	1001	0010000	9
0010	0100100	2	1010	0001000	A
0011	0110000	3	1011	0000011	b
0100	0011001	4	1100	1000110	C
0101	0010010	5	1101	0100001	d
0110	0000010	6	1110	0000110	E
0111	1111000	7	1111	0001110	F

You also should **not** use K-map to perform any optimization. Quartus (and other modern CAD design software) will perform logic minimization for you and will do a much better job, taking into account the architecture of the FPGA chip.

#### Step 5: Create the project “ex1”

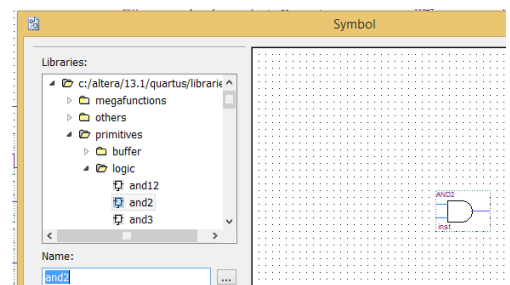
- Create in your home directory the folder **../part\_1/ex1**.
- Click **file>New Project Wizard**, complete the form. Use **ex1** as the project name and **ex1\_top** as top-design name.
- Select the FPGA device as Cyclone V 5CSEMA5F31C6. Then click **Finish**.




#### Step 6: Specify the 7-segment decoder as schematic

- Download from the website the file: **My7Seg\_incomplete.bdf.zip** and unzip in the folder **../part\_1/ex1**. This is a **partially completed** schematic for the 7-segment decoder circuit with circuit for **out[4]** missing. You are now ready to enter the circuit to produce **out[4]** as gates using the schematic editor. This is shown on the right and it implements the equation:

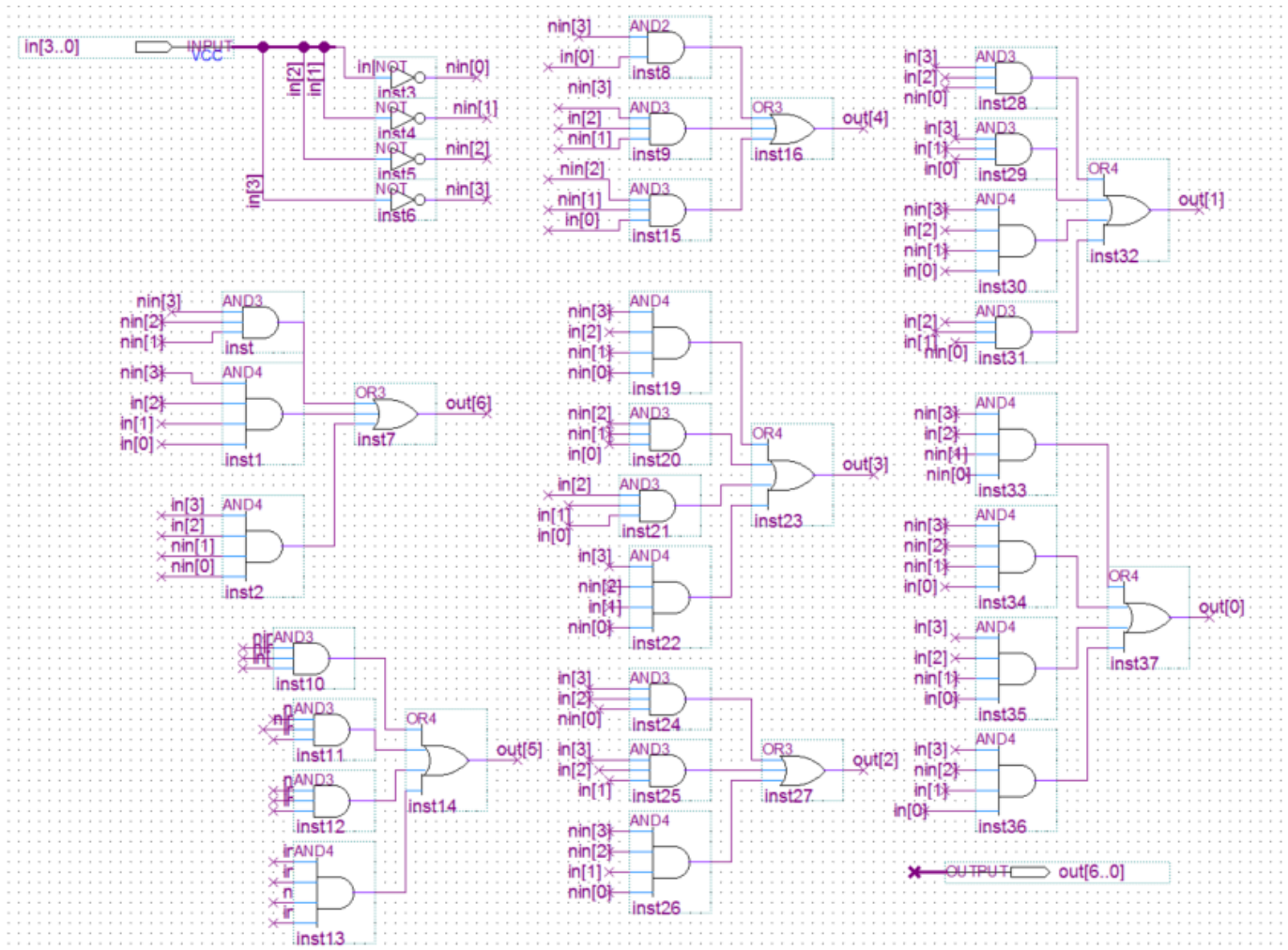
$$\text{out4} = \text{in3} * \text{in0} + \text{in3} * \text{in2} * \text{in1} + \text{in2} * \text{in1} * \text{in0}$$



The Graphic Editor provides a number of libraries which include circuit elements that can be imported into a schematic. Double-click on the blank space in the Graphic Editor window, or click on the icon  in the toolbar that looks like an AND gate. A pop-up box will appear.

Expand the hierarchy in the Libraries box as shown in the figure. First expand libraries, then expand the library primitives, followed by expanding the library logic which comprises the logic gates. Select “and2”, which is a two-input AND gate, and click OK. Now, the AND symbol will appear in the Graphic Editor window. Using the mouse, move the symbol to a desirable location and click to place it there.

- Repeat and place two “and3” and one “or3” gates on the schematic. Change the names of all the input and output nodes accordingly. (It is quickest to put down all the gates first before wiring them up later.)
- Now wire up the gates by click and drag on the input nodes of the gates to extend a wire out, and then simply type the name of the node on the keyboard.
- When completed, you will see the entire schematic diagram for the decoder circuit as shown here:



#### Step 7: Include this file in project

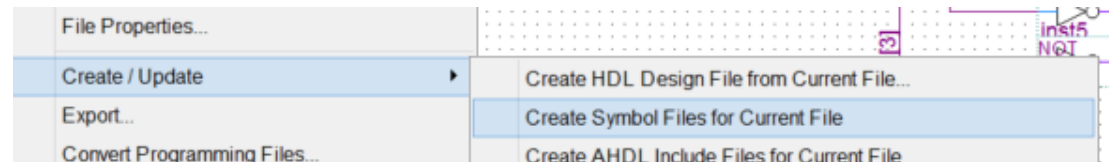
Every time you create a new entity or module as part of your design, you must include the file in the project.

- Click: **Project > Add Current Files to Project ....**,




### Step 7: Make a symbol for the decoder

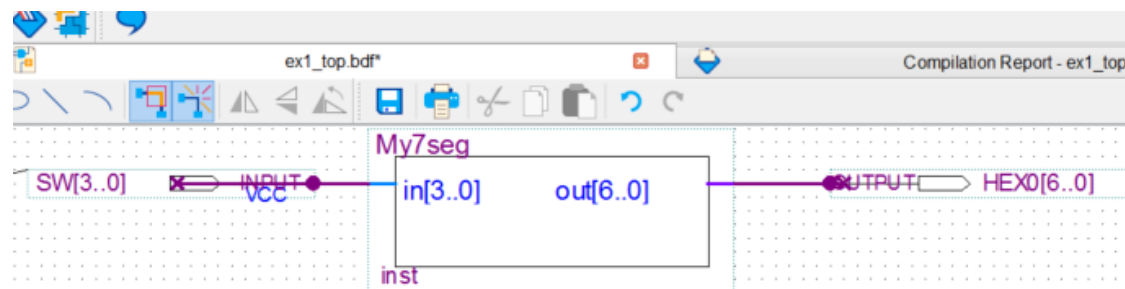
It is often convenient to encapsulate a circuit into a module (sometimes known as an “entity”), which is then used multiple times in a design. For us to do so, we need to create a symbol for **My7seg** decoder module.

Click **File > Creat/Update > Create Symbol ...**



### Step 8: Use this module at the top-level design schematic

- Now we will use the newly created entity **My7seg** in the top-level design.
- Click **File > New ....** and select Block Diagram /Schematic File as shown here:
- Use the  button to select and place the **My7seg** module, input port and output port on the schematic.
- Double click the port symbol  to edit the input and output pin names as **SW[3..0]** and **HEX0[6..0]** respectively.
- Use the bus wiring tool  to wire up the ports to the module as two busses as shown below.
- Save this file.



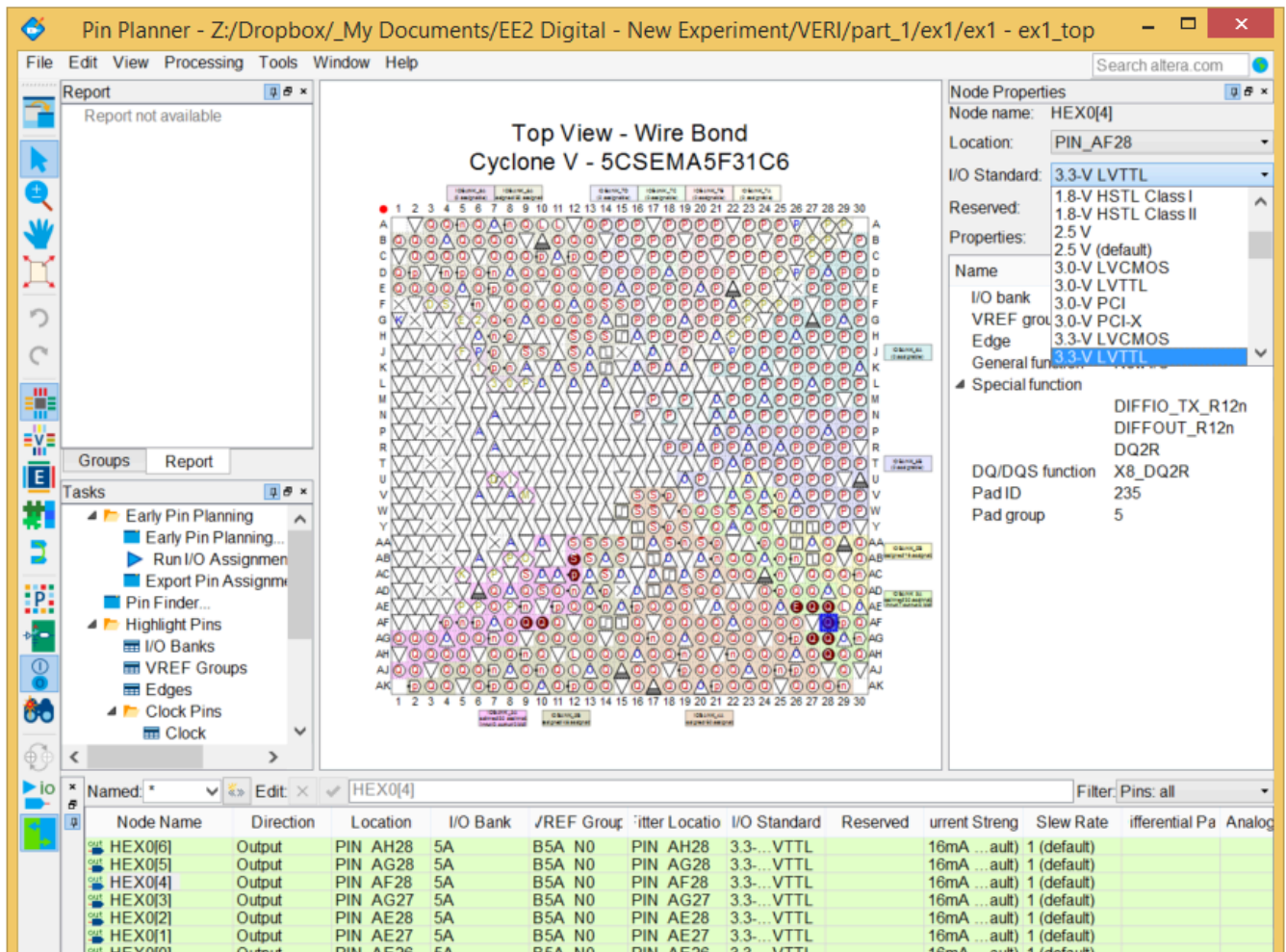
### Step 9: Pin assignment & Compilation

You need to associate your design with the **physical pins** of the Cyclone V FPGA on the DE1-SOC board.

- Check that the device is corrected assigned as 5CSEMA5F31C6 using: **Assignments > Device ...**
- Click: **Processing > Start > Start Analysis and Elaboration.** This will work out the input/output port names for your design. This should complete without error. Otherwise, fix all errors and re-analyse. (There will be many warnings – generally warnings are not important. But there MUST not be errors, which will be shown in RED.)
- Click **Assignment > Pin Planner** and a new window with the chip package diagram. You should also see the top-level input/output ports shown as a list.

Signal Name	Pin Location
HEX0[6]	PIN_AH28
HEX0[5]	PIN_AG28
HEX0[4]	PIN_AF28
HEX0[3]	PIN_AG27
HEX0[2]	PIN_AE28
HEX0[1]	PIN_AE27
HEX0[0]	PIN_AE26
SW[3]	PIN_AF10
SW[2]	PIN_AF9
SW[1]	PIN_AC12
SW[0]	PIN_AB12





- Click on the appropriate pins one by one, and select the corresponding location from a dropdown list according to the list shown in the pin assignment table above. The I/O standard (i.e. interface voltages) should be “3.3V LVTTTL”.
- Click: **Processing > Start Compilation**, to build the entire design, and to generate all the necessary files. There should be NO error, but there will be many warnings.

#### Step 10: Program the FPGA on the DE1 Board

- You have now created in the `../part_1/ex1/output_files/` folder the file `ex1_top.sof`, which contain your design. (This should be the same design as the one I gave you to try out in Step 2 of this exercise.)
- Program the DE1 board with your version of `ex1_top.sof` and test that it is working properly.

#### Step 11: Propagation Delay from inputs to outputs

- Click: **Tools > TimeQuest Timing Analyzer** to invoke the built in timing analyzer of Quartus. A new TimeQuest window will appear.
- Click: **Netlist > Create Timing Netlist**. Then select post-fit and slow-corner, then OK.
- In the “**Set Operating Conditions**” window, select “Slow 1100mV 0°C model”.
- Now click: **Netlist > Update Timing Netlist ...** This will use the specified timing model and condition to produce a set of timing data.
- Click: **Report > Datasheet > Report Datasheet**. This will produce a table showing the input-to-output propagation delay for various combination of rise and fall times (RR, RF, FR and FF). Make sure that you understanding what this table means.

- Repeat the procedure again but for “Slow 1100mV 85°C Model”. What is the delay difference at these two temperature extremes? Why?

#### Step 12: Examine the resources used

- Now examine the Compilation Report. You should see something as shown here.
- It shows that this design used only 4 out of 32,070 ALMs (Adaptive Logic Modules), 11 of the 457 I/O pins and none of the other resources.

Flow Summary	
Flow Status	Successful - Sun Oct 09 15:29:33 2016
Quartus Prime Version	16.0.0 Build 211 04/27/2016 S.J. Lite Edition
Revision Name	ex1_top
Top-level Entity Name	ex1_top
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	4 / 32,070 ( < 1 % )
Total registers	0
Total pins	11 / 457 ( 2 % )
Total virtual pins	0
Total block memory bits	0 / 4,065,280 ( 0 % )
Total DSP Blocks	0 / 87 ( 0 % )
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 ( 0 % )
Total DLLs	0 / 4 ( 0 % )

**Congratulations! You have now completed your first FPGA design!**

## Experiment 2: 7-Segment decoder in SystemVerilog HDL

I hope you now appreciate how limiting and slow it is to enter a design as a schematic diagram. Modern digital designers DO NOT USE schematic as a method of entry any more. Instead a designer would either use Verilog or VHDL hardware description language, or some high level language such as OpenCL or Vivaldo HLS to specify the design. In this experiment, you will design the Verilog version of what you have done in Experiment 1. Hopefully this will convince you never to use schematic capture for digital design again!

### Step 1: hex\_to\_7seg.sv

- Create a new project **ex2** as before and a top-level module **ex2\_top** as before in **ex2** folder.
- In Quartus, create a design file in SV known as **hex\_to\_7seg.sv** using:  
**File > New ....** and select SystemVerilog HDL from the list.
- Type the Verilog source file as shown below. (You should have seen this during one of the Lectures earlier). Make sure you pay attention to the syntax of SV. Save your file.
- A full compilation can take a long time. A far more efficient way to check the syntax of your code by clicking: **Process > Analyze current file**. You should get into a habit of ALWAYS perform this step to make sure that the new Verilog module you have created is as error free as possible. It will save you a lot of time.

```

1  module hexto7seg (
2      output logic [6:0] out, // low-active output
3      input logic [3:0] in // 4-bit binary input
4  );
5
6      always_comb
7      case (in)
8          4'h0: out = 7'b1000000;
9          4'h1: out = 7'b1111001; // -- 0 ---
10         4'h2: out = 7'b0100100; // |   |
11         4'h3: out = 7'b0110000; // 5   1
12         4'h4: out = 7'b0011001; // |   |
13         4'h5: out = 7'b0010010; // -- 6 ---
14         4'h6: out = 7'b0000010; // |   |
15         4'h7: out = 7'b1111000; // 4   2
16         4'h8: out = 7'b0000000; // |   |
17         4'h9: out = 7'b0011000; // -- 3 ---
18         4'ha: out = 7'b0001000;
19         4'hb: out = 7'b0000011;
20         4'hc: out = 7'b1000110;
21         4'hd: out = 7'b0100001;
22         4'he: out = 7'b0000110;
23         4'hf: out = 7'b0001110;
24         default: out = 7'b0000000; // default all
25     endcase
26 endmodule

```

### Step 2: Create Top-Level Specification in SV

- Instead of using schematic capture for the top-level module (that connects to physical pins on the FPGA), we will do this also in Verilog by creating the file: "**ex2\_top.sv**" as shown here. Set this as Top-Level entity.
- Click: **Project > Add/Remove Files**, and remove the .bdf file as part of this project.
- This allows you to remove the .bdf file and replace it with the .sv file for the top-level specification.
- Verify that everything works properly with: **Process > Start > Start analysis & elaboration**. Make sure that there is no error. (Warnings often capture potential errors. However, the Quartus system generates many warnings, and nearly all of which are not important. Once you have gain confidence on the system, you may start ignoring the warning, but never ignore any error.)

```

1  //-----
2  // Module name:  ex2_top
3  // Function: Top level module for Lab 3 Task 1
4  //      .... to display 4-bit value a 7-seg display
5  // Creator: Peter Cheung
6  // Version: 3.0
7  // Date: 6 Oct 2024
8  //-----
9  module ex2_top (
10      input logic [3:0] SW, // declare input/output ports
11      output logic [6:0] HEX0
12  );
13      hexto7seg SEG0 (.out(HEX0), .in(SW[3:0]));
14  endmodule

```

Annotations in the image:

- Instance name:** Points to `SEG0` in line 13.
- Internal name:** Points to `hexto7seg` in line 13.
- External name:** Points to `SEG0` in line 13.

You will save a lot of time if you ALWAYS use these two steps: analyze, and analysis & elaboration, and ensure that ALL errors are dealt with (and warning understood).

### Step 3: Pin Assignment – the quick way

- Earlier you used the **pin assignment editor** to associate pins on the package to your signals. This is a tedious process. In ex1, if you have correctly completed the design, the pin assignment would have been stored in a file: “**ex1\_top.qsf**” file.
- Open this file, either using Quartus’ built-in editor by clicking: **File > Open file...** or use your own favourite edit on your PC.
- You will find lines of statement such as:

```
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX0[4]  
set_location_assignment PIN_AF28 -to HEX0[4]
```

- The first line defines the voltage standard used by the HEX0[4] signal (3.3V logic).
- The second line defines the physical pin location of HEX0[4] is PIN\_AF28.
- Now open the **ex2\_top.qsf** file. You will see that there is no pin assignment for this design yet. Before full compilation, we need to tell Quartus which signal is connect to which physical pin on the FPGA.
- Instead of using the tedious **pin assignment editor** in **ex1**, we will modify the **ex2\_top.qsf** file with our text editor to include the pin assignment information. To do this, first download from the experiment webpage the file: **pin\_assignment.txt** to the VERI directory.
- Then use: **Edit > Insert File ...** in Quartus to insert the whole of **pin\_assignment.txt** in **ex2\_top.qsf**.
- Note that we only use 7 pins in **ex2\_top.sv**, but **pin\_assignment.txt** defines **all** pins used by the four parts of Experiment VERI. Quartus will generate lots of warnings which you may ignore about these unused pins not being driven. It will not create any error and the pin assignments for unused pins will be ignored.

### Step 5: Test your design

- Recompile your design.
- Go to the Programmer window (assuming that you still have it opened). Delete the **.sof** file entry and add the current **.sof** file.
- Test your design on the board.

### Step 6: Put module in mylib

Over the four weeks in the Lab, you will design and verify various SystemVerilog modules which you will reuse. You should copy these to the “**mylib**” folder and include them in your new design as necessary.

**Note:** When you perform a compilation, there may be a popup window informing you that some “Chain\_x.cdf” file has been modified, and ask if you wish to save it. Just click NO.

### Experiment 3: Test yourself - 10-bit binary switch values on three 7-segment displays

Here is a “test yourself” exercise. Create your own design to display all 10-bit sliding switches as hexadecimal on three of the 7-segment LED displays.

**Checkpoint:** You should get to this point by the end of the 3-hour Lab Session or earlier.

### Experiment 4: Displaying 10-bit binary as BCD digits on the 7-segment displays

In one of the lectures, you have been taught how to convert binary numbers to binary-code-decimal digits using the “**shift and add 3**” algorithm. You have been shown how to implement an 8-bit binary to BCD converter using Verilog. Furthermore in problem sheet 1, you have been asked to extend this to a 10-bit converter (**bin2bcd\_10.sv**).

For this optional exercise, you are required to display the 10-bit binary number as specified by the 10 sliding switches SW[9:0] as a decimal number using your 10-bit converter module and the 7-segment decoder. Record the resource usage of your design.

- Now download from the experiment website a 16-bit binary to BCD converter module provided (**bin2bcd\_16.v**), and replace your 10-bit converter with this one.
- When instantiating the 16-bit converter, but only using 10 of the 16 bits, you should specify the input ports as: {6'b0, SW[9:0]}. (Remember that the {...} operator is for bit-concatenation.)
- Test your design on the DE1 Board.
- Compare the resource usage by this design (with **bin2bcd\_16.v**) with that using the 10-bit version (**bin2bcd\_10.sv**). You will find that in fact the number of ALMs used will be the same.
- Basically Quartus optimizer removes unused resources. The module **bin2bcd\_16.sv** has six of its input connected to 0, and only 12 of its output connected to output pins. The CAD software will eliminate all the redundant logic. This should result in the same number of ALM being used as that with a 10-bit converter. In other words, for such combinational circuit, you only need to keep the 16-bit version for any numbers with 16 bits or lower.

Before you move onto Part II of the experiment, you should copy the components (modules) you have designed to the “mylib” folder. In the following sessions, you will be using the various .sv files from this repository of your own design. You will also be adding to it later.