University of California, Davis
Department of Electrical and Computer Engineering

EEC180B DIGITAL SYSTEMS Spring Quarter 2018

LAB 2: FPGA Synthesis and Combinational Logic Design

Objective: This lab covers the steps Quartus Prime uses to analyze a Verilog design, synthesize that design into Boolean expressions, and map the resulting netlist onto the FPGA.

- **Part 1** provides an overview of design compilation and illustrates how to view the results of logic synthesis, technology mapping, and placement/routing using a simple 2-to-1 MUX. The students are then asked to repeat these steps for increasingly complex designs.
- **Part 2** will give more practice designing combinational circuits in Verilog and writing testbenches.
- **Part 3** shows an easier way to implement the design for Part 2 of Lab 1.

Prelab

Please complete the following tasks before coming to lab. Refer to the course webpage for prelab guidelines and grading criteria.

1. Read through Part 1. Write out the differences between a look-up table (LUT), a logic element (LE), and a logic array block (LAB) for the MAX10 FPGA. More information can be found on pages 3-8 of the following document: `https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/max-10/m10_architecture.pdf`

2. Write the Verilog for the 2-to-4 decoder for Part 1. You may write your code using `assign` statements, a `case` statement, or any other synthesizable Verilog construct. Your code must be able to copied and pasted straight into a template generated by SystemBuilder and work correctly.

3. Read through Part 2. Create truth tables for the combinational module outputs h and i. You may use the following as a template:

Inputs			Outputs		
a	b	c	g	h	i
0	0	0	1		
0	0	1	0		
0	1	0	0		
0	1	1	1		
1	0	0	0		
1	0	1	1		
1	1	0	1		
1	1	1	0		

Part 1 – Design Compilation

Many steps must be performed in order to convert a high-level Verilog design into a form that can be implemented on the FPGA. This conversion process is known as design compilation and most of the heavy lifting is done by the Quartus Prime software. This section will begin with a brief, simplified overview of the design compilation process, demonstrate design compilation with a simple gate, and then ask you to investigate more complex designs.

A simplified summary of the Quartus Prime compilation flow[1] is shown in Figure 1. These steps are summarized as follows:

- **Design:** This includes all the files that make up the design. These can be files written in a hardware description language (Verilog or VHDL), schematic driven designs, memory initialization files, and more.
- **Analysis & Synthesis** Design files are analyzed for completeness and consistency. Design elements such as flipflops, latches, and state machines are inferred from hardware description languages such as Verilog. Constructs in the design are converted into Boolean expressions and logic optimizations are applied.

  Once a design has been synthesized into digital logic, it must be mapped to the primitive hardware elements of the FPGA including look-up tables (LUTs), memory blocks, hardware multipliers, and I/O pins. This process is called *technology mapping*.
- **Fitter:** After a design has been technology mapped, the Fitter attempts to match the synthesized logic and design timing requirements to the available resources on the specific selected device. Each logic function is mapped to a specific logic cell (e.g. LUT, flipflop, multiplier etc.) in the FPGA. An appropriate interconnect path is chosen between logic functions and configured. When this step is complete, the design has been completely mapped to the FPGA and is ready to be implemented.
- **Assembler:** Once the location for all logic functions and routing paths have been determined, the bitstream that actually configures the LUTs and routing switches in the FPGA can be generated.
- **TimeQuest Analyzer:** This tool analyzes the timing paths in the final design, calculates propagation delay along each path, checks for timing violations, and reports timing results.

1.1 Compilation of a 2-to-1 MUX

We will now follow the compilation process using a simple 2-to-1 multiplexor.

- Run **DE10_Lite_SystemBuilder.exe**
- Set the project name to any of your choice. Select the LED and Switch I/O devices. When you generate the project from SystemBuilder, make sure that you save the project to your home directory. By default, SystemBuilder saves to a public folder accessible to all students using your computer.
- Exit System Builder.
- Use Quartus Prime to open the project you just created.

A) Implement Using an `assign` Statement

At this point, your design should consist solely of the System Builder Verilog Template. Implement a 2-to-1 multiplexor with select line `SW[0]`, inputs `SW[1]` and `SW[2]`, and output `LEDR[0]` by placing the following line of code in the structural coding section of your top level design file:

[1]A more detailed description can be found at: `http://quartushelp.altera.com/15.0/mergedProjects/comp/comp/comp_view_comp.htm`

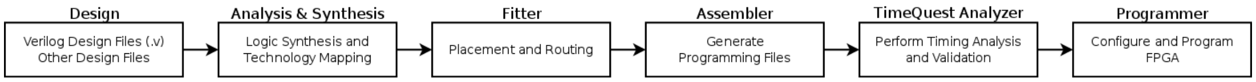
Design
Verilog Design Files (.v) Other Design Files
Analysis & Synthesis
Logic Synthesis and Technology Mapping
Fitter
Placement and Routing
Assembler
Generate Programming Files
TimeQuest Analyzer
Perform Timing Analysis and Validation
Programmer
Configure and Program FPGA

Figure 1: Simplified flow for design compilation in Quartus Prime.

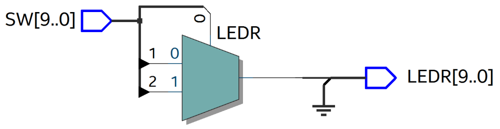

Figure 2: RTL Viewer result for the 2-to-1 mux both Verilog implementations.

```
reg mux_output;
always @(SW) begin
   if (SW[0]) begin
      mux_output = SW[2];
   end
   else begin
      mux_output = SW[1];
  end
end
assign LEDR[0] = mux_output;
```

Figure 3: Skeleton behavioral description for a 2-to-1 mux.

```
// LEDR is already declared in the port list
assign LEDR[0] = SW[0] ? SW[2] : SW[1];
```

Compile your design and program the DE10-Lite board. It should compile without errors. Verify that the design works as expected using the switches on the development board.

RTL Netlist Viewer

You can see the results of design analysis using the Netlist viewers in Quartus Prime. Open the RTL Viewer using

- Tools > Netlist Viewers > RTL Viewer.

The RTL Viewer displays a schematic view of the design netlist after Analysis is performed by Quartus, but before technology mapping or fitting occurs. The view is not the final design because no optimizations have been performed, but closely represents the original source design. If a design was synthesized using Quartus, this viewer shows how Quartus interpreted your design.

The expected output for the RTL Viewer for the mux is shown in Figure 2. Verify that what you have matches the figure. As can be seen, Quartus correctly inferred our design as a 2-to-1 mux.

B) Implement Using an `always` Block and `reg` Signals

Now, replace the implementation of the 2-to-1 mux with the code given in Figure 3. This is simply another way of describing a 2-to-1 mux using Verilog. Compile the design in Quartus and verify that it works as expected on the DE10-Lite board. Once again, open the RTL Viewer and verify that the RTL Viewer shows the same design as it did previously. This shows that different Verilog implementations of logic **can get synthesized to the same final design.**

Technology Mapped Netlist Viewer

Logic designs are ultimately implemented in look-up tables and other "atom primitives" on the FPGA. You can view this using the Technology Map Viewer. Navigate here using

- Tools > Netlist Viewers > Technology Map Viewer (Post-Mapping)

The Technology Map View provides a technology-specific graphical representation of your design after Analysis & Synthesis. It shows the hierarchy of atom primitives (LUTS, I/O ports, memories etc) in your design.

Verify that your results are similar to those shown in Figure 4. As can be seen, the entire design is implemented in a single logic element because a 2-to-1 mux easily fits into a 4-input LUT. Observe that the output pins LEDR[9:1] are tied to ground. Right click on the single logic cell in the design and select "Properties". The "Properties" window will appear on the left. This will show that logic-gate equivalent of the look-up table, the truth table for the LUT, its equivalent Boolean equation, and more.

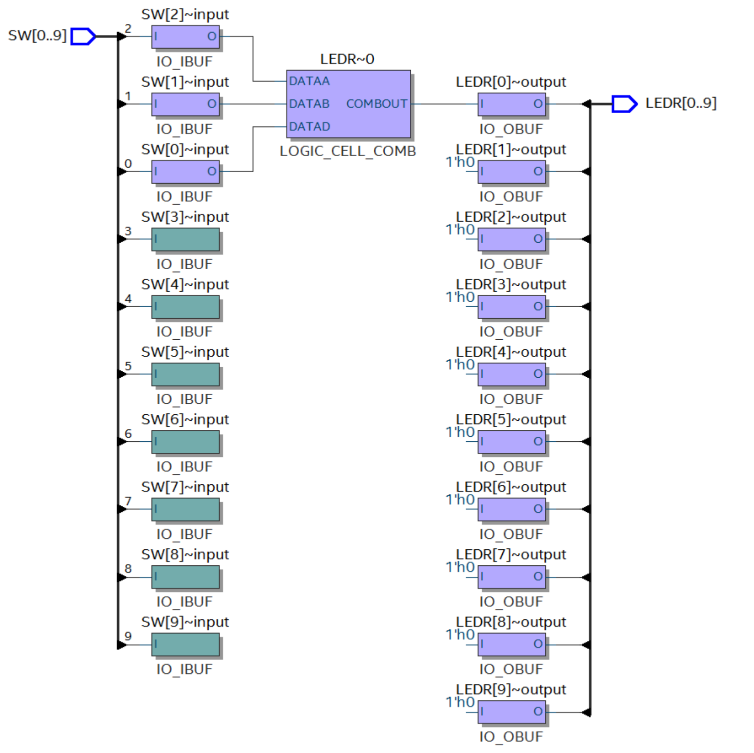

Figure 4: Expected results of the Technology Map Viewer for the 2-to-1 mux. Ten input switches are shown on the left, the combinational logic cell in the middle, and ten LEDs are shown on the right side the figure.

Chip Planner

Finally, Quartus allows the user to view the results of fitting the design to a target FPGA. Naviage to the Chip Planner program by

- Tools > Chip Planner

Your results should appear as in Figure 5. This is a graphical depiction of the physical resources on the actual MAX10 FPGA on the DE10-Lite board. There is a lot going on in this figure:

- The **light-blue** rectangles are the Logic Array Blocks. In the MAX10 FPGA we are using:
  - Each **Logic Array Block** (LAB) consists of 16 Logic Elements.
  - Each **Logic Element** (LE) contains a 4-input LUT, a flipflop and routing muxes. If you want to jump ahead, Figure 7 shows a LE.
- The **white** rectangles are hardware 18x18 multipliers.
- The **yellow** rectangles are M9K memory blocks. These memory blocks each contain 8,192 bits of memory. Additional parity bits (a form of error detection) brings the total capacity of each block to 9,216.
- The **green** rectangle is the on-chip flash memory. This memory can contain the bitstream that programs the FPGA when it is powered on, reset, or reconfigured.
- The **brown** blocks on the border of the chip are the I/O ports and drivers, which take in signals from components such as switches and drive outputs like the LEDs.
- The **dark-blue** rectangle is the LAB that is being used by the current design.

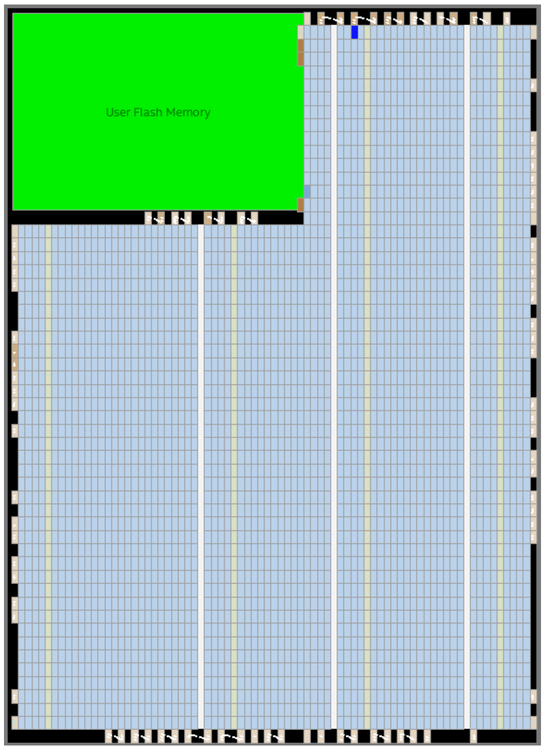

Figure 5: Chip Planner image for the 2-to-1 mux. *Note - this image does not render well in black-and-white. Best viewing is done in the online PDF.*

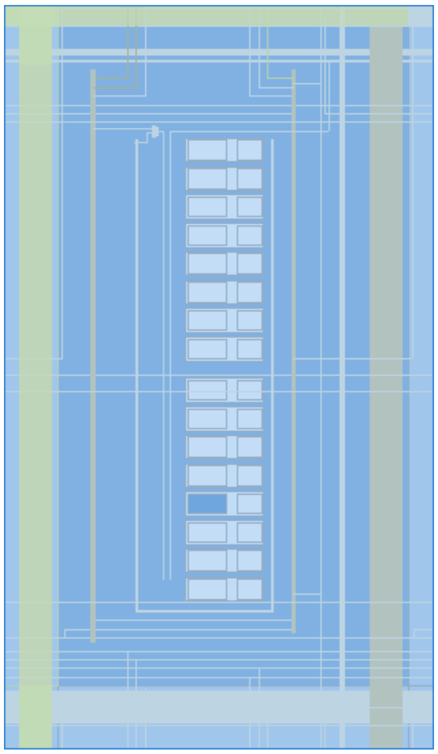

Figure 6: Zoom on the single occupied LAB block for the 2-to-1 mux design. Each LAB contains 16 LEs (shown as white rectangles). The LE that the 2-to-1 mux is assigned to is highlighted in blue. *Note - this image does not render well in black-and-white. Best viewing is done in the online PDF.*

Use the Chip Planner software to zoom in on the dark-blue rectangle. The image on your screen should look like Figure 6. The LUT that implements the 2-to-1 mux only occupies a single LE in the LAB.

If you double-click on the blue rectangle in the LAB, the "Resource Property Editor" should appear and you can see how Quartus has configured the LE to implement the 2-to-1 mux. Verify that your design is similar to that shown in Figure 7.

1.2 Assignment

Implement each of the following simple designs in your top level module. Compile each design and download it to the DE10-Lite board to ensure each works:

1. **Simple 2-to-4 Decoder**. Implement a decoder with inputs SW[1:0] and outputs LEDR[3:0]. The exact behavior of the decoder should follow the truth table:

SW[0]	SW[1]	LEDR[0]	LEDR[1]	LEDR[2]	LEDR[3]
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

2. **5-Bit Adder**. Implement a 5-bit adder. One operand should be SW[4:0] and the other should be SW[9:5]. Output the results to the LEDs. This can be done in a single line of Verilog using: `assign LEDR[9:0] = SW[4:0] + SW[9:5];`

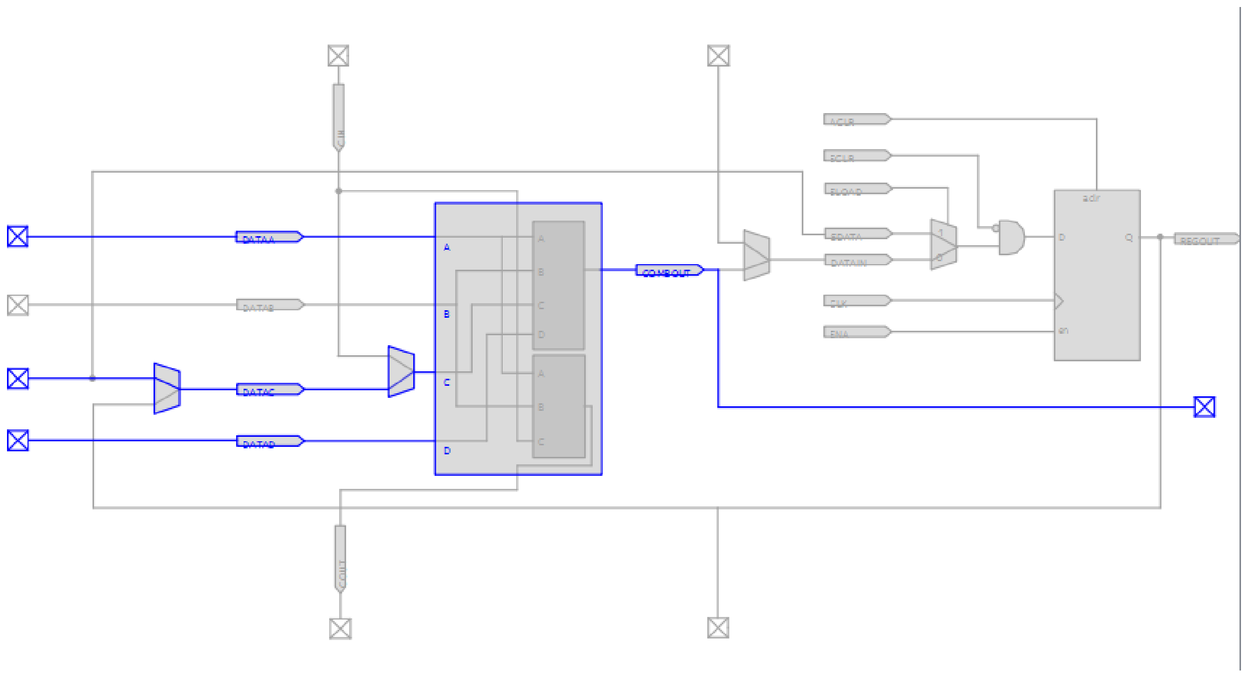
Figure 7: Resource Property Editor for the logic-element containing the 2-to-1 mux. As you can see, the LE contains a 4-input LUT, a flipflop, and routing muxes. An additional LUT is included for fast arithmetic operations.

3. **5-Bit Multiplier**. Implement a 5-bit multplier. One operand should be SW[4:0] and the other should be SW[9:5]. Output the results to the LEDs. This can be done in a single line of Verilog using: `assign LEDR[9:0] = SW[4:0] * SW[9:5];`

For each design, include screen captures for the following images:

- The top level RTL Viewer result (similar to Figure 2).
- The top level Technology Mapped Viewer (Post-Mapping) (similar to Figure 4).
- The placed design as seen in Chip Planner. Zoom in one the LAB that your design occupies (similar to Figure 6).

Questions

1. A 2-to-4 Decoder only has two inputs. Why does the post-mapping design require more than 1 LE?
2. Look at the placement of the LEs for the 5-bit adders in the LAB. Does anything stand out to you?
3. Multipliers are considerably more complicated than adders. Why does the Technology Mapped View for the 5-bit multiplier look so much simpler than the Technology Mapped View for the 5-bit adder?

Checkoff: *There are two items to demonstrate for your TA. First, you must show the successful implementation of the 2-to-4 decoder on the DE10-Lite board. Second, when you've finished Part 1, show your TA all nine of the required images for checkoff.*

A Final Note about Netlist Viewers

In digital design, the netlist viewers reviewed here are generally not as useful as they first appear. This is because a digital system of even moderate complexity will consist of thousands of logic elements and flipflops. At this scale, it becomes impossible to deduce the behavior of the design as a whole using just the netlist viewers. This is why languages such as Verilog and VHDL exist. Language abstractions allow designers to create more complex systems without getting bogged down in the minutiae of the final implementation.

Part 2 – Combinational Gate Design and Verification

In this exercise, you will get more practice writing testbenches for designs. The importance of a testbench in digital system design for debugging and verification cannot be over-emphasized and thus the ability to write a good testbench is critical.

Perform the following steps:

- Open ModelSim-Intel FPGA and create a new project. Give it a meaningful project name.
- When presented with the option to add items to the project, create two new Verilog source files. Call one `comb.v` – this will be your gate. Call the other file `comb_tb.v` – this will be your testbench for `comb.v`.

In the file `comb.v`, create a module called `comb` with three one-bit inputs a, b, and c and three one-bit outputs g, h, and i that implements the following system of Boolean equations:

$$g = a \oplus b \oplus \overline{c}$$
$$h = a\overline{b} + c$$
$$i = g + c$$

Now, write a testbench in the file `comb_tb.v`. Your testbench should

- Instantiate the `comb` module.
- Apply all possible input stimulus to the module.
- Display the inputs, expected outputs, and actual outputs of the module using the `$display` or a similar command.
- Use the `$display` or a similar command to warn if one of the outputs is incorrect for a given input.

On the EEC180B course webpage, you can find a skeleton testbench called `part2_testbench.v` that outlines how to implement the self-checking. You will have to expand on it. Refer to the *Verilog:Testing* slides on the course webpage for more information.

Checkoff: *Demonstrate your circuit and testbench to your TA. First show the testbench simulation working correctly. Then, purposely break the logic in* `comb` *so it is incorrect and demonstrate to your TA that your testbench can detects the faulty circuit.*

Part 3 – Implementing a Decimal Display using a Case Statement

In Part II of Lab 1, you designed a decimal display for 4-bit switch inputs by generating logic for each output implementing Karnaugh maps. However, Verilog case statement allows you to directly specify the truth table for your logic function and allow the synthesis tools to perform the decomposition into Boolean equations and subsequent logic optimizations. Doing this procedure makes both writing and reading code for the given logic function easier.

Refer to the Verilog template `part3_template.v` which can be found on the EEC180B course webpage. You must:

- Complete the Verilog template to create a design with the same behavior as the one you made for Part II of Lab 1.
- Compile your Verilog design and test it on the DE10-Lite board.

Checkoff: *Demonstrate your working design on the DE10-Lite board to your TA for verification.*

Lab Report

Submit the following items for verification and grading:

1. Lab coversheet with TA verification for:
   - Part 1 - 2-to-4 Decoder Demonstration and Quartus screen shots.
   - Part 2 - Testbench Demonstration.
   - Part 3 - Implementation of the decimal display on the DE10-Lite board.
2. Requested screenshots for Part 1.
3. Answers to the questions given in Part 1.
4. Complete Verilog source and testbench code for Parts 2 and 3.

Grading - 50 Points Total

- [10 pts] – Prelab
- [25 pts] – Lab Check-off
  - [7 pts] – Part 1
  - [9 pts] – Part 2
  - [9 pts] – Part 3
- [15 pts] Lab Report
  - [3 pts] – Part 1 screenshots
  - [4 pts] – Part 1 answers to questions
  - [8 pts] – Complete Verilog Code