DS5110 Final Project

Tyler Beaulieu, Tim Paylor

December 3, 2025

Abstract

Within both FPGA labs 1 and 2, we became acquainted with our Altera Cyclone V FPGA board and Quartus II FPGA editing software through various projects.These projects included building: various circuits to test out the LEDs on the board, various circuits to test out logic gates, a decode circuit, a grey code decoder, a 4-bit adder, an 8-bit squarer, and an add on to our 8-bit squarer to display the results in base 10 as opposed to hexadecimal.

1 Introduction

FPGAs (Field Programmable Gate Arrays) are standard devices used in instrumental design. The advantage of the FPGA is that it has no set architecture (unlike a Raspberry Pi or some other similar device) and so can be customized to fit a wide number of applications through wiring and rewiring the gates on the board. Altera Boards are compatible with the software Quartus, used to set the architecture of the gates and connect them to other devices, whether those built into the board or those connected externally.

2 Circuits

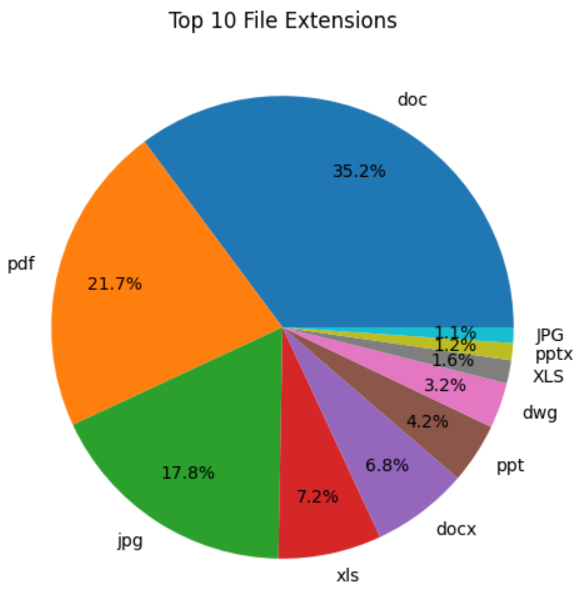
2.1 Simple Circuits

The first circuit we built and tested in Lab 1 was a simple connection between our dip switches on the FPGA board and the boards LEDs. The purpose of this circuit was to get us used to what the LED components and dip switch components look like in the Quartus software as well as to learn how to draw connections between components in the Quartus software. Through modifying this circuit by connecting an LED to ground then after connecting the same LED to a voltage supply we were able to discover that when the dip switch is in the position that turns the LED on, the dip switch is providing the LED with a voltage supply and not ground.

Lab 1.2 dealt with learning how to use the bus drawing tool by connecting a collection of dip switches to a collection of LEDs using a bus. The final result was a circuit that performed the same as the one built in Lab 1.1 except using a bus connection instead of wire connections.

In Lab 1.3 we learned how to insert components such as NAND gates and built a few circuits out of these gates wiring the output to the LEDs. Some of the circuits we built in this lab include: an OR circuit built out of NAND gates, and an XOR circuit built out of NAND gates. See Figure 1. The OR gate circuit has a truth table of 1 whenever one of the inputs is 1 and 0 otherwise while the XOR gate circuit has a truth table of one only when one of the inputs is one while the other is 0, and zero otherwise.

The final simple circuit we designed was 7-segment LED circuit. This circuit was design to output an 8-bit number in hexadecimal on two 7-segment LED displays. In this circuit we used a custom component called a 7segment. This component converts an input nibble to a nibble that lights up the correct LEDs of the 7-segment display to display the hexadecimal digit of the corresponding input nibble. The schematic for the circuit is shown in Figure 2 while the truth table for the custom component is shown in Figure 3. The first column from the left in the truth table is corresponds to the signal read from the blank input line, the second column from the left corresponds to the input nibble in hex, while the third column is the output from the component in binary. After playing around with the truth table for the 7seg component we were able to determine which LEDs corresponded to which bits of an input passed to the display device; for the specific device a bit set to zero lights up the corresponding LED. The correspondence can be seen in Figure 4. This helped us fix the bugs in the 7seg component truth that caused hexadecimal digits: 3, 8 and E to display incorrectly.



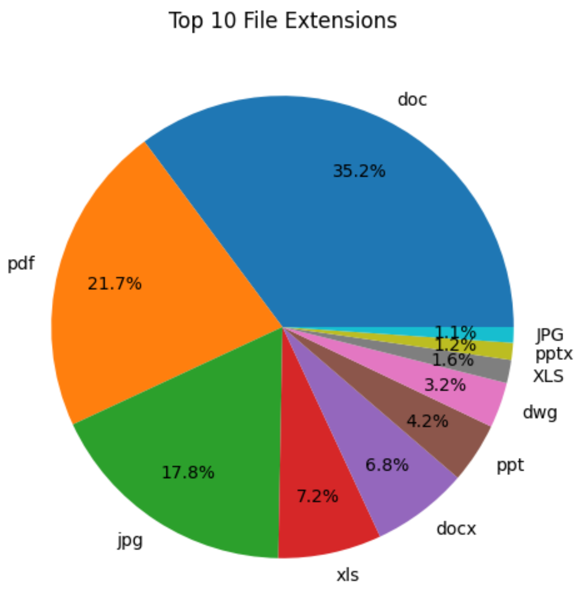


Figure 1: OR and XOR gate
OR gate is the circuit above while the XOR gate is the circuit below

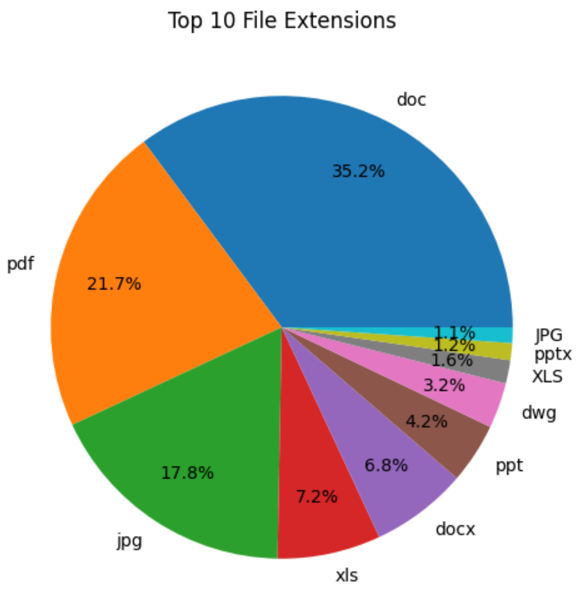


Figure 2: 7-segment LED circuit

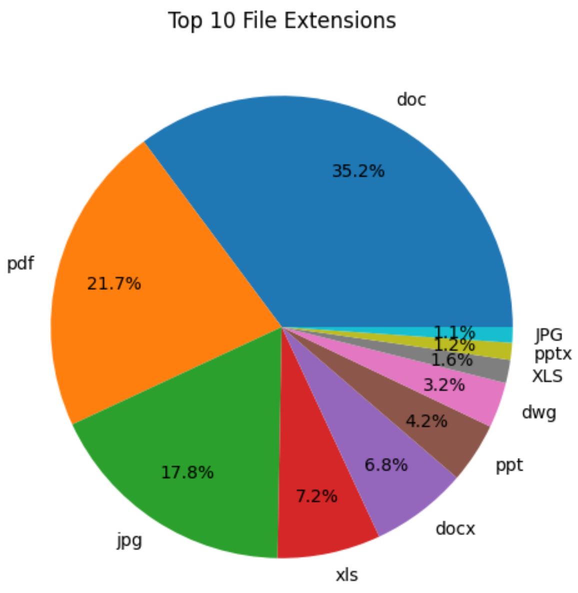


Figure 3: 7seg Component truth table

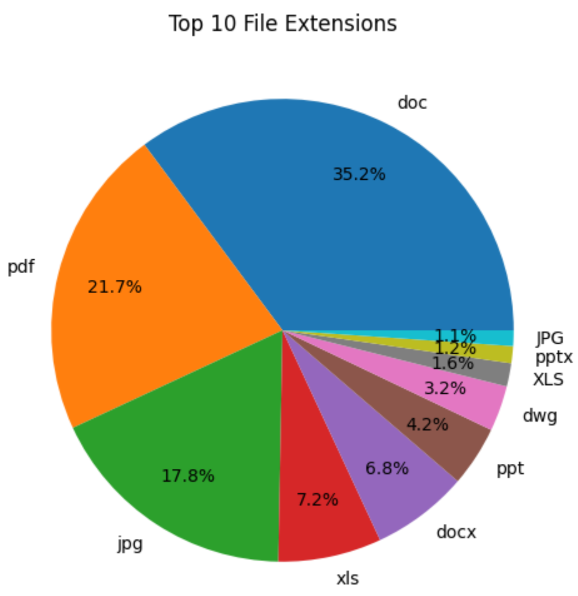


Figure 4: 7 segment LED display

2.2 Decoder

We then used NAND gates to construct a 2-bit decoder. A decoder takes a 2-bit number (from 00 to 11) and turns only the corresponding bit in a 4-bit number "on". For example, if the input is 01, the output is 0100. This can be constructed by combining inverters and AND gates, both of which are easily constructed from NANDs. Our schematic is shown in Figure 5.

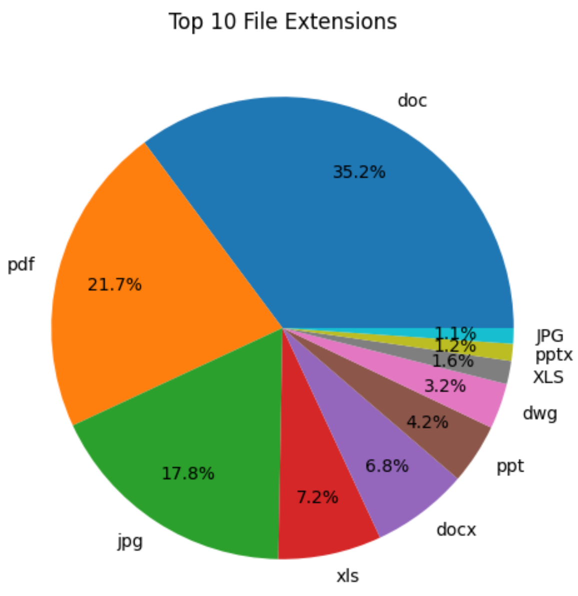


Figure 5: 2-bit decoder

2.3 Grey Code Translator

A Grey Code Translator was created, converting a 4-bit grey code to a hexadecimal output on the 7 segment display. The schema was taken from pg. 480 of Horowitz and Hill, 2nd Edition.

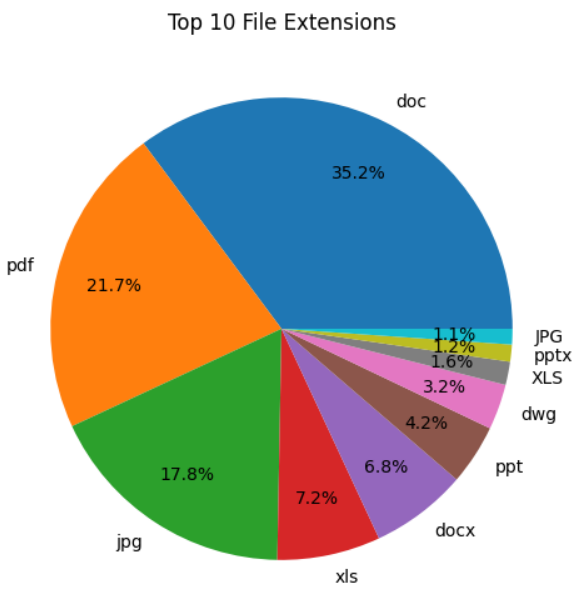


Figure 6: Grey Code Translator

2.4 Adder

Using the Adder Submodule provided in Quartus, we constructed an Adder circuit to add two 4-bit numbers. The numbers are set with the dip switches, and the output appears in hexadecimal on the 7-segment display. When the sum exceeds 15, the carry bit is set to high and is added to the sum of the next digit.

The "mf constant" input on the last 7-segment always inputs "0001" into the display HEX1 (as seen in Figure 7). However, if the "blank" input is set to high, it will just display 0. To display the last carry for sums larger than 15 (F), an inverter is added to the carry bit of the last adder. Then when the carry is high, the blank input is low and HEX1 displays "1".

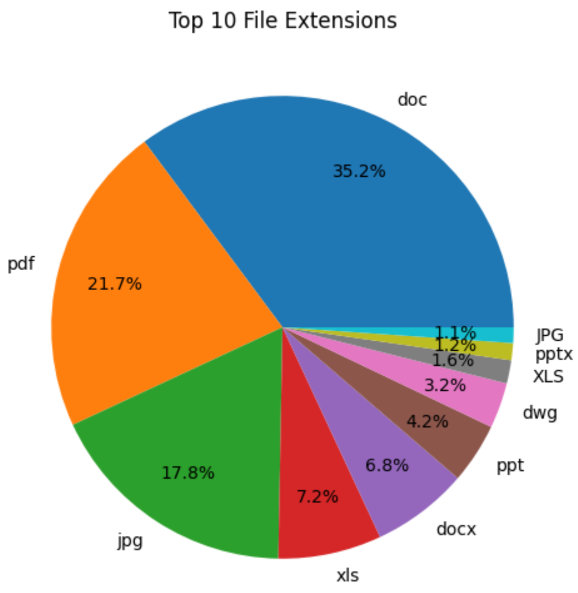


Figure 7: 4-Bit Adder

2.5 Multiplier

In Lab 2.4 we learned how to use the the FPGA softwares mega function tool. Using the mega function tool we built an 8-bit multiplier component and used it to square 8-bit inputs. The 8-bit input was then displayed in hexadecimal on two of the 7-segment LED displays while the 16-bit result was displayed in hexadecimal on four of the 7-segment LED displays on the FPGA board. The schematic for this circuit is shown in Figure 8

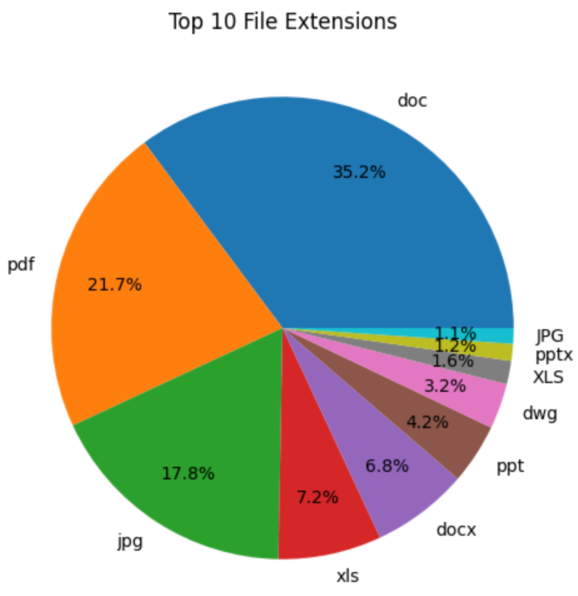


Figure 8: Squarer circuit

2.6 Multiplier with Base 10 Display

In the final section for the these two labs we modified our circuit from Lab 2.4 to display all numbers on the 7-segment LED displays in decimal as opposed to hexadecimal. We accomplished this by running all numbers through a series of custom divider components that divide their inputs by 10 and output the quotient and remainder. The remainder is decimal digit that can then be sent of to a 7seg component to display on 7-segment LED display.

Since we only had two 7-segment LED displays dedicated for our input number we could only display decimal numbers that contained two digits. This was a problem because our 8-bit input number can range from 0 to 255 which is a 3 digit decimal number. Since we were constrained by the amount of LED displays we had, our multiplier only works for numbers that contain two digits. We had a similar issue with our 16-bit output, which at its maximum is a 5 digit decimal number, and we only had 4 different LED displays to display it with. Therefore because of the constraints we had our squarer circuit only works for numbers for 0 through 99. See Figure 9 for the schematic for this circuit.

Aside from space constraints, it is much easier to translate binary into hex than binary into decimal. The first four digits of a binary number translate into the first digit of a hex number, and so on. This makes conversion convenient and easy. Decimal contains no such correspondence, making it more tedious to work with.

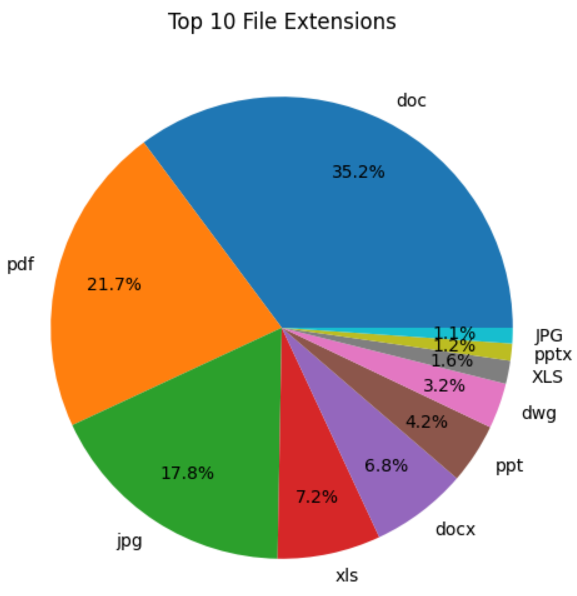


Figure 9: Squarer circuit with decimal display