Lab Assignment 10

Controlling the 7-Segment Displays with

Object-Oriented Programming

Zhenming Yang, Connor McEleney

[yang.zhenm@northeastern.edu](mailto:yang.zhenm@northeastern.edu), [mceleney.c@northeastern.edu](mailto:mceleney.c@northeastern.edu)

Instructor: Prof. Zagieboylo

[d.zagieboylo@northeastern.edu](mailto:d.zagieboylo@northeastern.edu)

Due Date: November 28th, 2023

Submit Date: November 28th, 2023

**Abstract**

In this lab, we used the DE1-SoC board to display characters and numbers in C++. Two data registers controlled these displays: Register 1 for HEX0 to HEX3 and Register 2 for HEX4 and HEX5, enabling word operations.

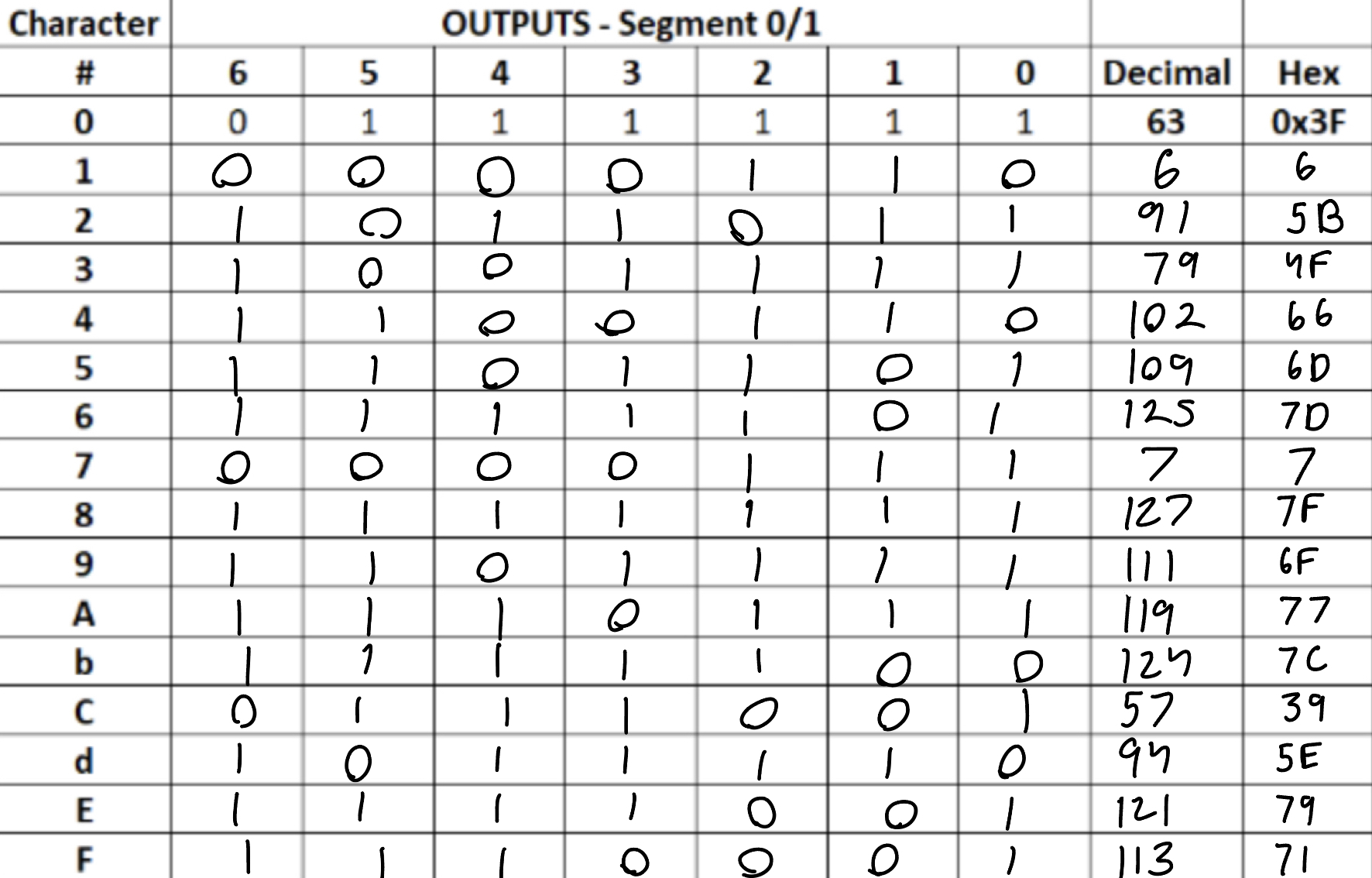
# Introduction

# In this lab, we worked with the DE1-SoC board's 7-segment displays to showcase characters, decimals, and hexadecimal numbers using object-oriented programming in C++. Recalling in lab 2, we created digital logic circuits for operating these displays on the DE1-SoC board. This board featured six such displays, managed by two data registers. Each segment of the displays, labelled from 0 to 6, was manipulated by a bit in these registers. The first four displays (HEX0, HEX1, HEX2, and HEX3) were operated by Data Register 1, while the remaining two (HEX4 and HEX5) were governed by Data Register 2. We could write to and read from these registers through word operations.

# Lab Setup

## Pre-Lab

Character binary logic for later implementation



## Equipment

DE1-SoC:

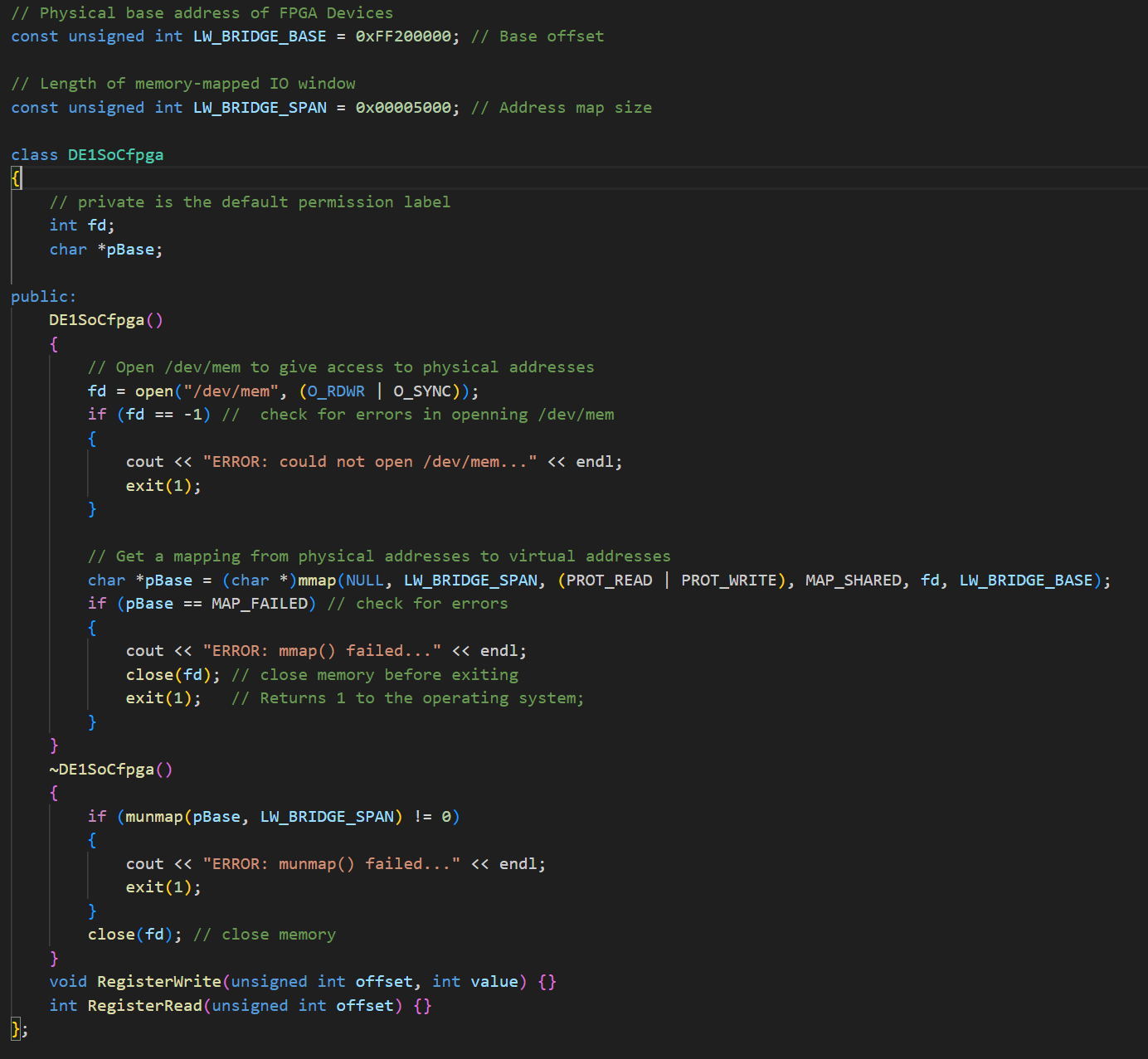
* The DE1-SoC is a hardware design platform built around the Altera System-on-Chip (SoC) FPGA. The DE1-SoC is designed for experiments on computer organization and embedded systems. It includes embedded processors, memory, audio and video devices, and some simple I/O peripherals.

# Results and Analysis

**Results**

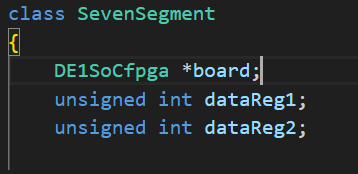
## Part 1: Using Object-Oriented Programming to Control the 7-Segment Displays

To first program the 7-segment displays, we first converted the DE1SoCfpga class from lab 9 into its individual class. We split the implementation into the header file and a cpp file with a 1. Constructor sets up the memory-mapped I/O, a Destructor shuts down the memory-mapped I/O, a Function RegisterWrite(offset, value) writes a value to a register at a specified offset, and a Function RegisterRead(offset) retrieves the value from a register at a specified offset (Figure 1).



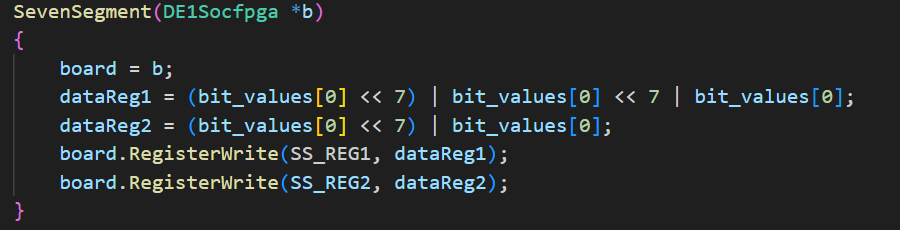
**Figure 1**: The header file for the class DE1SoCfpga. Every functions and constants are from the lab 9.

We then created a new class called SevenSegment. The class includes three private data members: DE1SoCfpga \*board, unsigned int dataReg1, and unsigned int dataReg2, which serve as the Board object and hold the values for two data registers (Figure 2).



**Figure 2**: The attributes of the SevenSegment class.

These data register variables are updated each time a new value is written to their respective registers. The class is equipped with a constructor that takes a DE1SoCfpga\* as an argument, initializing these private members to display zeroes on all six 7-segment displays of the board (Figure 3).



**Figure 3**: Constructor of the SevenSegment class. It takes a pointer to the DE1Socfpga object and write the bit\_values for 0 into the specified memory address. Noted the bit\_values is an array of bit representations of numbers for the display we got from the pre lab (Figure 4).



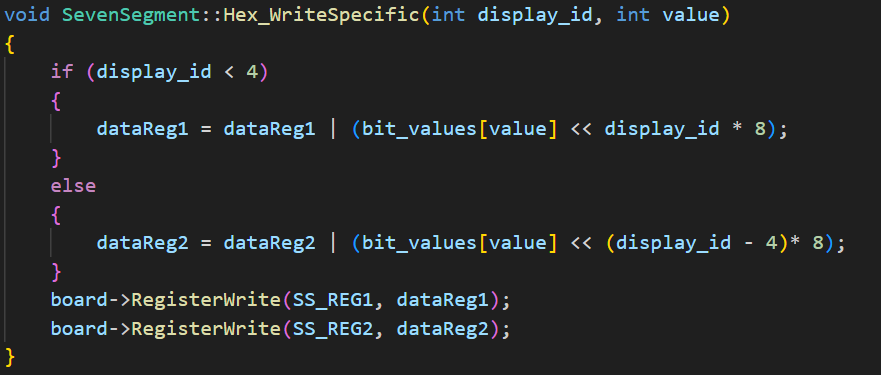
**Figure 4**: The bit\_value array has all the bit representations of the number regarding their index.

The class also offers a public method Hex\_ClearAll(), which turns off all the 7-segment displays, and Hex\_ClearSpecific(int index), which turns off a specific display based on the index ranging from 0 to 5 (Figure 5).



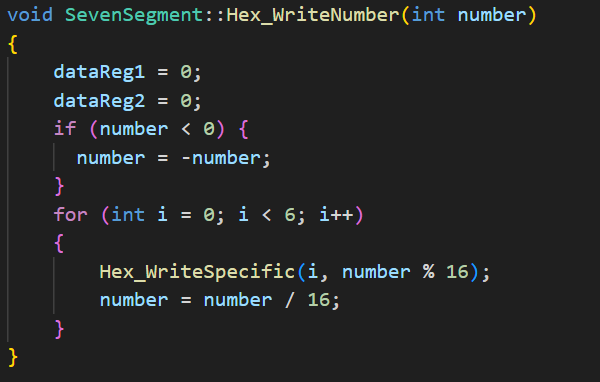
**Figure 5**: The Hex\_ClearAll and Hex\_Clear functions that turns all data registers to 0 or a specific indexes to 0.

There's also Hex\_WriteSpecific(int display\_id, int value), which displays a hexadecimal digit on a chosen 7-segment display, with the digit determined by a decimal value (0 to 15) and display\_id selecting the display (Figure 6).



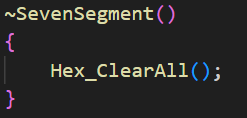
**Figure 6**: The Hex\_WriteSpecific function write the specific display to the given value. The function gets the bit representation of that number and shift it to the specific index and write that value in to it.

Another method, Hex\_WriteNumber(int number), is designed to show a hexadecimal value of a positive or negative integer across the six 7-segment displays, within the range of -FFFFF to FFFFFF, without displaying the '+' sign for positive numbers (Figure 7).



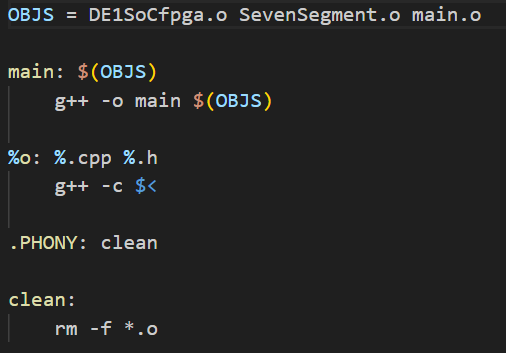
**Figure 7**: The Hex\_WriteNumber uses the Hex\_WriteSpecific to first break down the number into multiple single digits hex numbers by doing arithmatics and display them at varies index.

Lastly, the class destructor calls the Hex\_ClearAll() function to turn off all the displays (Figure 8).



**Figure 8**: Destructor that runs the Hex\_ClearAll function.

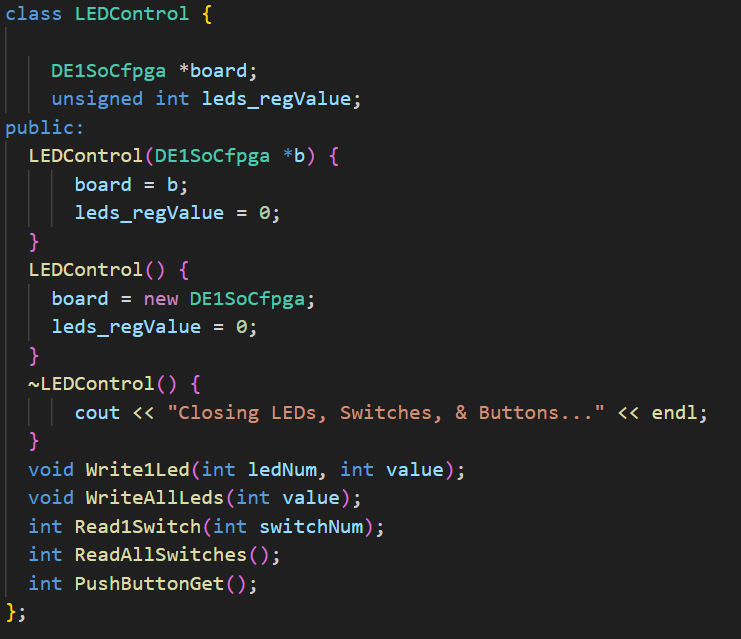
We tested our implementation on the board using a Makefile and it gave us the expected results (Figure 9).



**Figure 9**: Makefile that configure the compiler to create object files from header files and cpp files. The clean command remove the created object files.

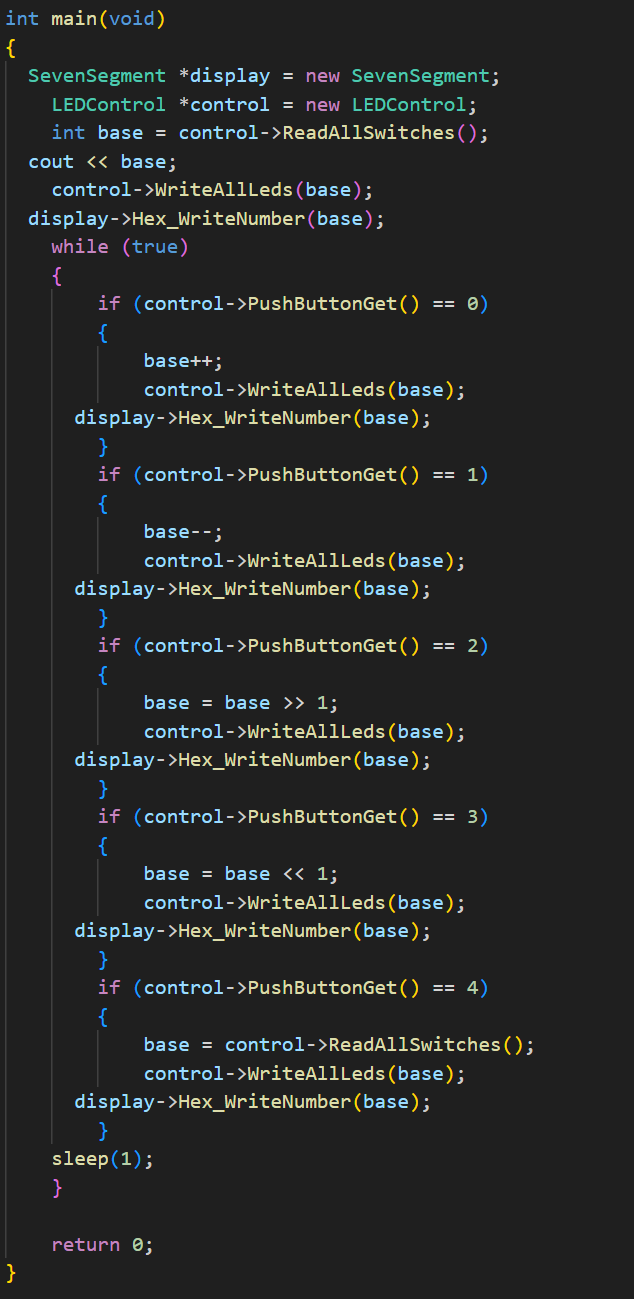
## Part 2: Combining the 7-Segment Displays with LED Control

To incorporate the LED control, we used for lab 9 into our newly programed 7-Segment Displays, we first created a LEDControl class that has all the functions and attributes form our last lab (Figure 10). Every function is programed the exact same way as what we did in the previous lab.



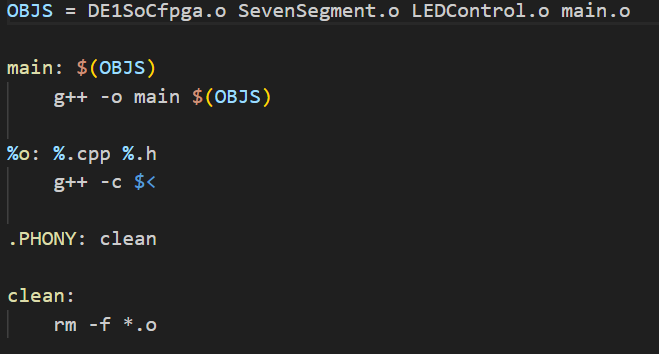
**Figure 10**: The LEDControl class contains all the functions in the last lab using a DE1SoCfpga class object pointer instead.

To make the test the functions, we modified the code from last lab and added a new object using the SevenSegment class to display the number out on the Seven Segment Display (Figure 11).



**Figure 11**: New main function that not only display the number through the LEDs but also the Seven Segment display.

The program is build using Makefile and tested and it gives us promising results (Figure 12).



**Figure 12**: Makefile that configure the compiler to create object files from header files and cpp files. The clean command remove the created object files.

**Analysis**

In this lab the code consists of a class DE1SoCfgpa that handles the communication between the software and hardware as well as two other classes. SevenSegment is one of the classes and its purpose is to handle memory management of the seven-segment display and includes methods to calculate which digit to display on what segment based off of the current digit being represented. The LEDControl class managed the memory of the board LED’s and displayed the binary representation of the current number as well as including methods to read the switches. In main these classes were used in conjunction to display both the binary (on the LED’s) and HEX (on the seven-segment display) and pushbuttons/switches were used to increment/decrement and set the value being represented as well as shift bits.

# Conclusion

This lab introduced us to key concepts of building C++ projects, such as the use of header, source, and make files as well as reinforcing our skills in object-oriented programming.

Using header and source files is an efficient way to organize the code structure, and using the make file to compile the files makes it automatic and efficient. While this particular project wasn’t large, for much larger projects to compile the entire thing would be a waste of time, the make file ensures that only the edited code gets re-compiled and creates object files and executables for everything based on the input parameters.