Controlling Seven Segment Displays Using Object-Oriented Programming
Embedded Design: Enabling Robotics
EECE2160

Michael Brodskiy
Brodskiy.M@Northeastern.edu

April 19, 2023

Date Performed:	April 12, 2023
Partner:	Dylan Powers
Instructor:	Professor Shazli

Abstract

This laboratory experiment was intended to be a conclusion to the course, which integrates all concepts covered, including, but not limited to, bits and hexadecimal, digital logic, object-oriented C++, and headers and makefiles. By integrating all of these concepts together, with minimal assistance, the course leaves us with proficient knowledge of them. As a result of the lab, three header files, `DE1SoCfpga.h`, `LEDControl.h`, and `SevenSegment.h`, and their respective `.cpp` files were created, in addition to a `main.cpp` file containing code to interact with headers, and a makefile to compile everything together.

KEYWORDS: bits, hexadecimal, digital logic, object-oriented, headers, makefiles, `DE1SoCfpga`, `LEDControl`, `SevenSegment`

1 Equipment

Available equipment included:

- DE1-SoC board
- DE1-SoC Power Cable
- USB-A to USB-B Cable
- Computer
- MobaXTerm SSH Terminal
- USB-to-ethernet Adapter
- gcc compiler

2 Introduction

This lab project has the primary goal of controlling the 7-segment displays using object-oriented programming in C++. In the project, the 7-segment displays on the DE1-SoC board were utilized to display characters, decimal values and hexadecimal values being controlled by two parallel ports. The first port controls the displays (`HEX3`, `HEX2`, `HEX1`, & `HEX0`), and the second parallel port controls the last two displays (`HEX5` & `HEX4`). Data can be written into these two ports (registers) and read back by using word operations.

3 Discussion & Analysis

3.1 Assignment 1

The purpose of assignment one was simply logic-based. It was necessary to consider the 7-bit logic behind seven-segment displays, and generate a table representing a hexadecimal number or letter in decimal, binary, and hexadecimal. The table is shown below.

#	6	5	4	3	2	1	0	Decimal	Hex
0	0	1	1	1	1	1	1	63	0x3F
1	0	0	0	0	1	1	0	6	0x6
2	1	0	1	1	0	1	1	91	0x5B
3	1	0	0	1	1	1	1	79	0x4F
4	1	1	0	0	1	1	0	102	0x66
5	1	1	0	1	1	0	1	109	0x6D
6	1	1	1	1	1	0	1	125	0x7D
7	0	0	0	0	1	1	1	7	0x7
8	1	1	1	1	1	1	1	127	0x7F
9	1	1	0	1	1	1	1	111	0x6F
A	1	1	1	0	1	1	1	119	0x77
b	1	1	1	1	1	0	0	124	0x7C
C	0	1	1	1	0	0	1	57	0x39
d	1	0	1	1	1	1	0	94	0x5E
e	1	1	1	1	0	0	1	121	0x79
f	1	1	1	0	0	0	1	113	0x71

3.2 Assignment 2

In Part 2 of this project, the `DE1SoCfpga` class from the previous lab was converted into an independent class with its own header file and source file. The two files that were created are shown below in Listing 1 and 2. The independent class `DE1SoCfpga` served as the base class for initializing memory and controlling register access.

Listing 1: `DE1SoCfpga` Class Declaration in Header File `DE1SoCfpga.h`

```
/*
 * =====================================================================
 *
 *       Filename:  DE1SoCfpga.h
 *
 *    Description:  Header file for the DE1SoCfpga class
 *
 *        Version:  1.0
 *        Created:  04/06/2023
 *       Revision:  none
 *       Compiler:  GCC
 *
 *         Author:  Michael Brodskiy (Brodskiy.M@Northeastern.edu)
 *                  Dylan Powers (Powers.D@Northeastern.edu)
 *
 * =====================================================================
 */
```

```
18
19 #ifndef DE1SOCFPGA_H
20 #define DE1SOCFPGA_H
21 #include <stdio.h>
22 #include <unistd.h>
23 #include <stdlib.h>
24 #include <fcntl.h>
25 #include <sys/mman.h>
26 #include <iostream>
27 #include <cmath>
28 #include <unistd.h>
29
30 using namespace std;
31
32 // Physical base address of FPGA Devices
33  const unsigned int LW_BRIDGE_BASE = 0xFF200000; // Base
       offset
34 // Length of memory-mapped IO window
35 const unsigned int LW_BRIDGE_SPAN = 0x00005000; // Address
     map size
36
37 // Cyclone V FPGA device addresses
38 const unsigned int LEDR_BASE = 0x00000000; // Leds offset
39 const unsigned int SW_BASE = 0x00000040; // Switches offset
40 const unsigned int KEY_BASE = 0x00000050; // Push buttons
     offset
41
42 const unsigned int HEX3_HEX0_BASE =  0x00000020; // HEX
     Reg1 offset
43 const unsigned int HEX5_HEX4_BASE =  0x00000030; // HEX
     Reg2 offset
44
45 class DE1SoCfpga {
46
47     public:
48
49         char *pBase;
50         int fd;
51         DE1SoCfpga();
52         ~DE1SoCfpga();
53         void RegisterWrite(unsigned int offset, int value);
54         int RegisterRead(unsigned int offset);
55
56 };
57
58 #endif
```

Listing 2: DE1SoCfpga Implementation file DE1SoCfpga.cpp

```
/*
 * =====================================================================
 *
 *       Filename:  DE1SoCfpga.cpp
 *
 *    Description:  Contains function definitions for DE1SoCfpga header file
 *
 *        Version:  1.0
 *        Created:  03/30/2023
 *       Revision:  none
 *       Compiler:  GCC
 *
 *         Authors:  Michael Brodskiy (Brodskiy.M@Northeastern.edu)
 *                   Dylan Powers (Powers.D@Northeastern.edu)
 *
 * =====================================================================
 */

#include "DE1SoCfpga.h"

    /**
    * Initialize general-purpose I/O
    * - Opens access to physical memory /dev/mem
    * - Maps memory into virtual address space
    *
    * @return Address to virtual memory which is mapped to physical,
    * or MAP_FAILED on error.
    */
DE1SoCfpga::DE1SoCfpga() {

        // Open /dev/mem to give access to physical addresses
        fd = open( "/dev/mem", (O_RDWR | O_SYNC));
        if (fd == -1) // check for errors in openning /dev/mem
        {
            cout << "ERROR: could not open /dev/mem..." << endl;
            exit(1);
        }
        // Get a mapping from physical addresses to virtual
```

```
            addresses
        char *virtual_base = (char *)mmap (NULL,
            LW_BRIDGE_SPAN, (PROT_READ | PROT_WRITE),
            MAP_SHARED, fd, LW_BRIDGE_BASE);
        if (virtual_base == MAP_FAILED) // check for errors
        {
            cout << "ERROR: mmap() failed..." << endl;
            close (fd); // close memory before exiting
            exit(1); // Returns 1 to the operating system;
        }
        pBase = virtual_base;

    }


    /**
    * Close general-purpose I/O.
    *
    * @param pBase Virtual address where I/O was mapped.
    * @param fd File descriptor previously returned by '
      open'.
    */
DE1SoCfpga::~DE1SoCfpga() {

        if (munmap (pBase, LW_BRIDGE_SPAN) != 0) {

            cout << "ERROR: munmap() failed..." << endl;
            exit(1);

        }

        close (fd); // close memory

    }

    /**
    * Write a 4-byte value at the specified general-purpose
       I/O location.
    *
    * @parem offset Offset where device is mapped.
    * @param value Value to be written.
    */
    void DE1SoCfpga::RegisterWrite(unsigned int offset, int
        value) {

        * (volatile unsigned int *)(pBase + offset) = value
```

```
            ;
79
80      }
81
82      /**
83      * Read a 4-byte value from the specified general-
            purpose I/O location.
84      *
85      * @param offset Offset where device is mapped.
86      * @return Value read.
87      */
88      int DE1SoCfpga::RegisterRead(unsigned int offset) {
89
90          return * (volatile unsigned int *)(pBase + offset);
91
92      }
```

When looking at the implementation file, `DE1SoCfpga.cpp`, the functions for the previous lab were implemented from the previous lab. The constructor `DE1SoCfpga()` was included to initialize the memory-mapped I/O, the destructor `~DE1SoCfpga()` was included to finalize the memory-mapped I/O, the function `RegisterWrite(offset, value)` was a included to write a value to into a register given the offset, and the function `RegisterRead(offset)` was included to return the value read from a register given its offset.

3.3 Assignment 3

In Part 3 of the lab project, a new class called `SevenSegment` was created. This class had a declaration file or header file called `SevenSegment.h` and an implementation file or source file called `SevenSegment.cpp` both of which were in the same sevenLED directory. The two files are shown below in Listing 3 and 4.

Listing 3: `SevenSegment` class declaration in header file `SevenSegment.h`

```
1   /*
2    * =====================================================
3    *
4    *       Filename:  SevenSegment.h
5    *
6    *    Description:  Header file for the SevenSegment class
7    *
8    *        Version:  1.0
9    *        Created:  04/06/2023
10   *       Revision:  none
11   *       Compiler:  GCC
12   *
13   *         Author:  Michael Brodskiy (Brodskiy.
         M@Northeastern.edu)
```

```
 *                  Dylan Powers (Powers.D@Northeastern.edu)
 *
 * ================================================================
 */

#include "DE1SoCfpga.h"

const unsigned int bit_values[17] = {0x3F, 0x6, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x7, 0x7F, 0x6F, 0x77, 0x7C, 0x39, 0x5E, 0x79, 0x71, 0x40};

class SevenSegment : public DE1SoCfpga {

    private:

        unsigned int reg0_hexValue;
        unsigned int reg1_hexValue;

    public:

        SevenSegment();
        ~SevenSegment();
        void Hex_ClearAll();
        void Hex_ClearSpecific(int index);
        void Hex_WriteSpecific(int display_id, int value);
        void Hex_WriteNumber(int number);

};
```

Listing 4: `SevenSegment` source file `SevenSegment.cpp`

```
/*
 * ================================================================
 *
 *       Filename:  SevenSegment.cpp
 *
 *    Description:  Function definition file for the SevenSegment class
 *
 *        Version:  1.0
 *        Created:  04/06/2023
 *       Revision:  none
 *       Compiler:  GCC
 *
 *         Author:  Michael Brodskiy (Brodskiy.M@Northeastern.edu)
```

```
14  *                    Dylan Powers (Powers.D@Northeastern.edu
        )
15  *
16  * ================================================================
17  */
18
19 #include "SevenSegment.h"
20
21 SevenSegment::SevenSegment() {
22
23     reg0_hexValue = RegisterRead(HEX3_HEX0_BASE);
24     reg1_hexValue = RegisterRead(HEX5_HEX4_BASE);
25
26 }
27
28 SevenSegment::~SevenSegment() {
29
30     Hex_ClearAll();
31
32 }
33
34 void SevenSegment::Hex_ClearAll() {
35
36     RegisterWrite(HEX3_HEX0_BASE, 0);
37     RegisterWrite(HEX5_HEX4_BASE, 0);
38
39 }
40
41 void SevenSegment::Hex_ClearSpecific(int index) {
42
43     unsigned int mask;
44
45     if (index < 4) {
46
47         mask = 0xFF << (8 * index);
48         RegisterWrite(HEX3_HEX0_BASE, (RegisterRead(
            HEX3_HEX0_BASE) & ~mask));
49
50     } else if (index < 6) {
51
52         mask = 0xFF << (8 * (index - 4));
53         RegisterWrite(HEX5_HEX4_BASE, (RegisterRead(
            HEX5_HEX4_BASE) & ~mask));
54
55     } else {
56
```

```
57          cout << "Not a valid index value." << endl;
58
59      }
60
61  }
62
63  void SevenSegment::Hex_WriteSpecific(int display_id, int
        value) {
64
65      unsigned int mask;
66
67      if (display_id < 4) {
68
69          reg0_hexValue &= ~((~bit_values[value]) << (8 *
                display_id));
70          reg0_hexValue |= bit_values[value] << (8 *
                display_id);
71          RegisterWrite(HEX3_HEX0_BASE, reg0_hexValue);
72
73      } else if (display_id < 6) {
74
75          reg1_hexValue &= ~((~bit_values[value]) << (8 * (
                display_id - 4)));
76          reg1_hexValue |= bit_values[value] << (8 * (
                display_id - 4));
77          RegisterWrite(HEX5_HEX4_BASE, reg1_hexValue);
78
79      } else {
80
81          cout << "Not a valid index value." << endl;
82
83      }
84
85  }
86
87  void SevenSegment::Hex_WriteNumber(int number) {
88
89      int count = 0;
90      bool negative = false;
91
92      if (number < 0) {
93
94          number = -1 * number;
95          negative = true;
96
97      } else if (number == 0) {
```

```
98
99          Hex_WriteSpecific(0, 0);
100
101     }
102
103     while (number > 0) {
104
105
106         Hex_WriteSpecific(count, number % 10);
107         number /= 10;
108
109         count++;
110
111     }
112
113     if (negative) { Hex_WriteSpecific(count, 16); }
114
115 }
```

In these two files, there was important functionality included to achieve the goal of controlling the 7-segment displays on the DE1-SoC board. In the `SevenSegment.h` file two new private, unsigned int data members `reg0_hexValue` and `reg1_hexValue` were created with the ability to update every time a new value was written to the corresponding register as well as a global array with 16 elements with each element representing the display segments to be turned ON or OFF when writing to a 7-segment display. Here in the `SevenSegment.h` file is also where the class `SevenSegment` inherits the class `DE1SoCfpga` and uses the functions `RegisterWrite()` and `RegisterRead()` as needed.

Then in the `SevenSegment.cpp` file, the constructor `SevenSegment()` was included as well as the destructor `~SevenSegment()` which called a new function `Hex_ClearAll()`. `Hex_ClearAll()` is a function included in the `SevenSegment.cpp` source file that clears all the 7-segment displays. This was done by writing the value zero to the two data registers for the HEX displays with the `RegisterWrite()` function. Along with the function `Hex_ClearAll()`, the functions `Hex_ClearSpecific(int index)`, `Hex_WriteSpecific(int display_id, int value)` and `Hex_WriteNumber(int number)` were created and included in the `SevenSegment.cpp` file. Starting with the function `Hex_ClearSpecific()` this function turns off a specific 7-segment display specified by an index (0 to 5) using a bit mask and bit shifting. Next with the function `Hex_WriteSpecific()`, this public function enables the ability to write a hexadecimal digit specified by a decimal value (0-15) to a specified display indicated by a `display_id` (0-5). This function was created using an if else statement and the concepts of bit masking and bit shifting. Lastly the function `Hex_WriteNumber(int number)` writes a positive or negative number on the 7-segment displays as shown in the `SevenSegment` source file `SevenSegment.cpp` shown above.

All functions described above were verified using the `main.cpp` file shown below in

Listing 5 to ensure correct behavior.

Listing 5: `main.cpp` file for verifying behavior of functions in `SevenSegment.cpp`

```
/**
 * Main operates the DE1-SoC 7-Segment Displays
 * This program writes an integer number on the 7-Segment
    Displays
 */

int main(void) {

  int count = -25;

  // Create a pointer object of the SevenSegment class
  SevenSegment *display = new SevenSegment;

  cout << "Program Starting...!" << endl;

  // Update the display every second
  while( count <= 25 ) {

    int hex_value = count*count*count; // Value to display
    cout << "Count: " << count << ",  Value = " <<
       hex_value << endl;
    display->Hex_WriteNumber(hex_value);   // display value

    sleep(1); // wait for 1 second

    count++; // increment count

  }

  delete display; // delete class object
  cout << "Terminating...!" << endl;
  return 0;

}
```

In order to complete the verification process, a Makefile had to be created to compile the programs `DE1SoCfpga.h`, `DE1SoCfpga.cpp`, `SevenSegment.h`, `SevenSegment.cpp`, and `main.cpp`.

```
main: Main.o SevenSegment.o DE1SoCfpga.o
        g++ -g -Wall Main.o SevenSegment.o DE1SoCfpga.o -o
            main

Main.o: Main.cpp SevenSegment.h DE1SoCfpga.h
```

```
5         g++ -g -Wall Main.cpp -c
6
7   SevenSegment.o: SevenSegment.cpp SevenSegment.h DE1SoCfpga.
        h
8         g++ -g -Wall SevenSegment.cpp -c
9
10  DE1SoCfpga.o: DE1SoCfpga.cpp DE1SoCfpga.h
11        g++ -g -Wall DE1SoCfpga.cpp -c
12
13  clean:
14        rm main Main.o SevenSegment.o DE1SoCfpga.o
```

Results of the verification test are shown in a video that can be accessed by the first QR code below.

Figure 1: Verification video of `SevenSegment.cpp` functions

3.4 Assignment 4

The objective of Part 4 for the project was to combine a program written in a previous lab with the program written in parts 2 and 3 of this project so that the value controlled by push buttons is displayed on the 7-segment displays in addition to being displayed on the LEDs. This meant using the class `DE1SoCfpga` as a base class with both classes `LEDControl` and `SevenSegment` inheriting this class. To start, the `LEDControl` class was converted into an independent class with its own header and source file shown in Listing 6 and 7 respectively.

Listing 6: `LEDControl` class declaration in header file `LEDControl.h`

```
1   /*
```

```
 * ========================================================
 *
 *       Filename:  LEDControl.h
 *
 *    Description:  Header file for the LEDControl class
 *
 *        Version:  1.0
 *        Created:  04/13/2023
 *       Revision:  none
 *       Compiler:  GCC
 *
 *         Author:  Michael Brodskiy (Brodskiy.M@Northeastern.edu)
 *                  Dylan Powers (Powers.D@Northeastern.edu)
 *
 * ========================================================
 */

#include "DE1SoCfpga.h"

class LEDControl : public DE1SoCfpga {

    private:

        unsigned int leds_regValue;

    public:

        LEDControl();
        ~LEDControl();
        int Read1Switch(int switchNum);
        void Write1Led(int ledNum, int state);
        int ReadAllSwitches();
        void WriteAllLeds(int value);
        int PushButtonGet();

};
```

Listing 7: `LEDControl` source file `LEDControl.cpp`

```
/*
 * ========================================================
 *
 *       Filename:  LEDControl.cpp
 *
 *    Description:  File containing LED and push button
```

```
    controls
 *
 *         Version:  1.0
 *         Created:  04/13/2023
 *        Revision:  none
 *        Compiler:  GCC
 *
 *          Author:  Michael Brodskiy (Brodskiy.M@Northeastern.edu)
 *                   Dylan Powers (Powers.D@Northeastern.edu)
 *
 * =====================================================
 */

#include "LEDControl.h"

bool prevVal[4] = {false, false, false, false};
bool butValue[4] = {false, false, false, false};

LEDControl::LEDControl() {

    leds_regValue = RegisterRead(SW_BASE);

}

// Destroy the board and LEDControl objects
LEDControl::~LEDControl() {

    cout << "Closing LEDs, Switches, & Buttons..." << endl;

}

/**Reads the value of a switch
* -Uses base address of I/O
* @param switchNum Switch number (0 to 9)
* @return Switch value read
*/
int LEDControl::Read1Switch(int switchNum) {

    // Read the switch register
    int switchRegisterValue = RegisterRead(SW_BASE);

    // Mask the value to extract the specified switch bit
    int switchBitMask = 1 << switchNum;

```

```
50      // if the result is non-zero, then the switch is on,
            off otherwise
51      int switchValue;
52      //use the bitwise AND operator and compare result
            against the bit mask
53      if (switchRegisterValue & switchBitMask) {
54          switchValue = 1;
55      } else {
56          switchValue = 0;
57      }
58
59      return switchValue;
60
61  }
62
63  /** Changes the state of an LED (ON or OFF)
64  *  @param ledNum LED number (0 to 9)
65  *  @param state State to change to (ON or OFF)
66  */
67  void LEDControl::Write1Led(int ledNum, int state) {
68
69      if (ledNum < 0 || ledNum > 9) {
70          cout << "ERROR: Invalid LED number. Only LED
                numbers 0 to 9 are valid." << endl;
71          return;
72
73      }
74
75      // Read the LED register
76      int ledRegisterValue = RegisterRead(LEDR_BASE);
77
78      // Set the state of the specified LED based on the
            state parameter
79      if (state == 1) {
80          ledRegisterValue = (1 << ledNum);
81      } else {
82          ledRegisterValue &= ~(1 << ledNum);
83      }
84
85      // Write the new LED register value
86      RegisterWrite(LEDR_BASE, ledRegisterValue);
87
88      // Read the LED register again and print out the value
            to double check (this can be commented out)
89      int UpdatedRegisterValue = RegisterRead(LEDR_BASE);
90      cout << "LED Register Updated Value: " <<
```

```
        UpdatedRegisterValue << endl << endl;
91
92  }
93
94  /** Reads all the switches and returns their value in a
        single integer
95  * @return value that represents the value of the switches
96  */
97  int LEDControl::ReadAllSwitches() {
98
99      return RegisterRead(SW_BASE);
100
101 }
102
103 /** Set the state of the LEDs with the given value
104 * @param value  Value between 0 and 1023 written to the
        LEDs
105 */
106 void LEDControl::WriteAllLeds(int value) {
107
108     RegisterWrite(LEDR_BASE, value);
109
110 }
111
112 int LEDControl::PushButtonGet() {
113
114     // Read the switch register
115     int butRegisterValue = RegisterRead(KEY_BASE);
116
117     // Mask the value to extract the specified switch bit
118     int butBitMask[4];
119     for (int i = 0; i < 4; i++) {
120
121         butBitMask[i] = 1 << i;
122
123     }
124
125     // if the result is non-zero, then the switch is on,
            off otherwise
126     //use the bitwise AND operator and compare result
            against the bit mask
127     for (int i = 0; i < 4; i++) {
128
129         if (butRegisterValue & butBitMask[i]) {
130
131             butValue[i] = true;
```

```
132             if (butValue[i] != prevVal[i]) { prevVal[i] = !
                    prevVal[i]; }
133
134         } else {
135
136             butValue[i] = false;
137             if (butValue[i] != prevVal[i]) { prevVal[i] = !
                    prevVal[i]; }
138
139         }
140     }
141
142     int quantDiff = 0;
143     bool diff[] = {false, false, false, false};
144
145     for (int i = 0; i < 4; i++) {
146
147      if (butValue[i] && prevVal[i]) {
148
149          diff[i] = true;
150          quantDiff++;
151
152      }
153
154     }
155
156     if (quantDiff == 2) return 4;
157     else if (quantDiff == 0) return -1;
158     else {
159
160      for (int i = 0; i < 4; i ++) {
161
162          if (diff[i]) return i;
163
164      }
165
166     }
167
168 }
```

In the `LEDControl.cpp` source file, the constructor `LEDControl()` and the destructor ` LEDControl` were created where the destructor also displayed a message "Closing LEDs, Switches, & Buttons...". As for the functions `Write1Led()`, `WriteAllLeds()`, `Read1Switch()` and `ReadAllSwitches()` were included from the previous lab where they were tested and verified for correct behavior.
Then, in order to test the functions, a `main.cpp` file was created with the functionality of controlling the LEDs and Hex displays with the switches and the push

buttons on the DE1-SoC board. The `main.cpp` file can be seen below.

Listing 8: `Main.cpp` for Part 4

```
#include "SevenSegment.h"
#include "LEDControl.h"

int main() {

    SevenSegment display;
    LEDControl LEDs;
    int ledDisp = LEDs.ReadAllSwitches();
    // ************** Put your code here
        **********************
    while(true) {

            usleep(225000);
            int change = LEDs.PushButtonGet();
            if (change == 0) {

                ledDisp++;
                LEDs.WriteAllLeds(ledDisp);
                display.Hex_WriteNumber(ledDisp);
                if (ledDisp > 1023) ledDisp = 0;

            } else if (change == 1) {

                ledDisp -= 1;
                LEDs.WriteAllLeds(ledDisp);
                display.Hex_WriteNumber(ledDisp);
                if (ledDisp < 0) ledDisp = 1023;

            } else if (change == 2) {

                ledDisp *= 2;
                if (ledDisp > 1023) ledDisp -= 1024;
                LEDs.WriteAllLeds(ledDisp);
                display.Hex_WriteNumber(ledDisp);

            } else if (change == 3) {

                ledDisp /= 2;
                LEDs.WriteAllLeds(ledDisp);
                display.Hex_WriteNumber(ledDisp);

            } else if (change == 4) {

                ledDisp = LEDs.ReadAllSwitches();
```

```
                    LEDs.WriteAllLeds(ledDisp);
                    display.Hex_WriteNumber(ledDisp);

                }

    }

}
```

After this was completed, a Makefile was created to compile the programs, including a clean rule accessible through the command make clean in order to get rid of all generated binary files.

Listing 9: Makefile for Part 4

```
main: Main.o LEDControl.o SevenSegment.o DE1SoCfpga.o
        g++ -g -Wall Main.o LEDControl.o SevenSegment.o
            DE1SoCfpga.o -o main

Main.o: Main.cpp LEDControl.h SevenSegment.h DE1SoCfpga.h
        g++ -g -Wall Main.cpp -c

LEDControl.o: LEDControl.cpp LEDControl.h DE1SoCfpga.h
        g++ -g -Wall LEDControl.cpp -c

SevenSegment.o: SevenSegment.cpp SevenSegment.h DE1SoCfpga.
    h
        g++ -g -Wall SevenSegment.cpp -c

DE1SoCfpga.o: DE1SoCfpga.cpp DE1SoCfpga.h
        g++ -g -Wall DE1SoCfpga.cpp -c

clean:
        rm main Main.o LEDControl.o SevenSegment.o
            DE1SoCfpga.o
```

With the Makefile, the program was compiled and tested to verify the LEDs displayed the inputs binary value while the 7-segment displays displayed the same value in decimal, reflecting changes to the value when the push buttons are pressed. Below is a video demonstrating the working design.

4 Conclusion

Overall, this laboratory project was an effective way to finish off the course. By having us draw from concepts learned throughout the entirety of the course, we were able to effectively work with a hardware device integrated with C++. As such, through the creation of new code, as well as integration of code from previous

Figure 2: Verification Video

labs, DE1SoCfpga board interaction was converted to a fully object-oriented C++ program, encompassing all course concepts.