Lab Assignment 3
Controlling the 7-Segment Displays with Object-Oriented Programming

Levi Kaplan
kaplan.l@northeastern.edu

Submit Date: 10/05/20
Due Date: 10/05/20

Abstract
In this lab, we worked on writing hex values and letters to the 7 segment LED displays. We also worked on abstracting code into header, source, and makefile files, and running the code through the makefile. Finally, we worked on timers and writing to the board at a certain interval of time.

Introduction

This lab involved heavy use of object oriented programming, extracting code not only into multiple classes, but also into different header and source files and having these files communicate with one another. We had to make use of the information learned from previous labs to write, read, and alter data on the board, now doing so continuously at a fixed interval.

Software and Hardware Used

Hardware

- DE1-SoC board
- Ethernet cable

Software

- Atom text editor
- Online C++ compiler (https://www.onlinegdb.com/online_c++_compiler)
- ssh and scp

Lab Steps

Lab 3.1

1. We first created the header and cpp file for DE1SoCfpga
2. We abstracted the class declaration into the .h file
3. We added the method stubs and constants into this file
4. We made sure this file hadn't been defined earlier
5. We prepared the .cpp file

Lab 3.2

1. We first created the header and cpp file for SevenSegment
2. We abstracted the class declaration into the .h file
3. We made sure this file hadn't been defined earlier
4. We prepared the .cpp file
5. We added the main.cpp file
6. We added the makefile
7. We tested the functionality of our program

Lab 3.3

1.

Lab Discussion

Prelab

1. We replaced main.cpp with the starter code
2. We created an array for storing the letter values in decimal
3. We experimented with writing to the board during the timing
    a. We were able to get the board to flip back and forth between 0 and 1
4. We added a counter to loop through an array and display those values
5. We tried writing the rotating values to the board
6. We realized that our Hex_ClearSpecific method didn't work
7. We fixed the Hex_ClearSpecific method

Lab 3.1

1. We first created the header and cpp file for DE1SoCfpga
    a. we copied the DE1SoCfpga class from the previous assignment
    b. we created a header (.h) file and source (.cpp) file
2. We abstracted the class declaration into the .h file
3. We added the method stubs and constants into this file
    a. We made the constants static so they can be accessed outside this class
    b. We made the actual methods public so they can be accessed by the cpp file
4. We made sure this file hadn't been defined earlier
    a. We used #ifndef to check that the file wasn't defined, and #endif at the end of the file to end this if
5. We prepared the .cpp file
    a. We added a DE1SoCfpga:: before the methods
    b. we abstracted all comments and constants out

Lab 3.2

1. We first created the header and cpp file for SevenSegment

   a. We copied the SevenSegment class from the prelab
   b. We created a header (.h) file and source (.cpp) file
2. We abstracted the class declaration into the .h file
3. We added the method stubs and constants into this file
   a. We made reg0_hexValue and reg1_hexValue private but the rest of the methods public
   b. We added all comments and documentation to the method stubs
   c. We made the actual methods public to allow for access by the source file
4. We made sure this file hadn't been defined earlier
   a. We used #ifndef to check that the file wasn't defined, and #endif at the end of the file to end this if
5. We prepared the .cpp file
   a. We added a SevenSegment:: before the methods
   b. we abstracted all comments and constants out
6. We added the main.cpp file
   a. We abstracted main from SevenSegment to its own file
   b. We included SevenSegment.h so that we could run the methods in it
7. We added the makefile
   a. We created a series of commands to compile our program properly
   b. We looked at the examples from lecture to properly format the makefile
   c. We discovered that makefile does not need a file extension
8. We tested the functionality of our program
   a. We found an error with makefile not having proper spacing
   b. We edited makefile to have proper whitespace
   c. The program compiled and ran as expected

Lab 3.3

1. We replaced main.cpp with the starter code
2. We created an array for storing the letter values in decimal
   a. We put the array in SevenSegment.cpp
3. We experimented with writing to the board during the timing
   a. We were able to get the board to flip back and forth between 0 and 1
4. We added a counter to loop through an array and display those values

   a. We came across an issue where the values wouldn't loop through
   b. We later discovered this was because at the start of the while loop we were resetting the value of the counter
   c. Once we moved the counter outside the while loop we were able to loop through the values
5. We tried writing the rotating values to the board
   a. We first tried using Hex_WriteSpecific, but after struggling with it not working properly, we tried using Hex_WriteNumber
   b. After speaking with a TA, we realized we should in fact be using Hex_WriteSpecific
   c. We came up with a solution where we first looped through the old values of the board, writing them in the appropriate spot, and then wrote the new value
6. We realized that our Hex_ClearSpecific method didn't work
   a. Instead of clearing one cell, it cleared an entire block of cells, which meant we couldn't properly rotate through them
7. We fixed the Hex_ClearSpecific method

Results

Prelab 3

Lab 3.1

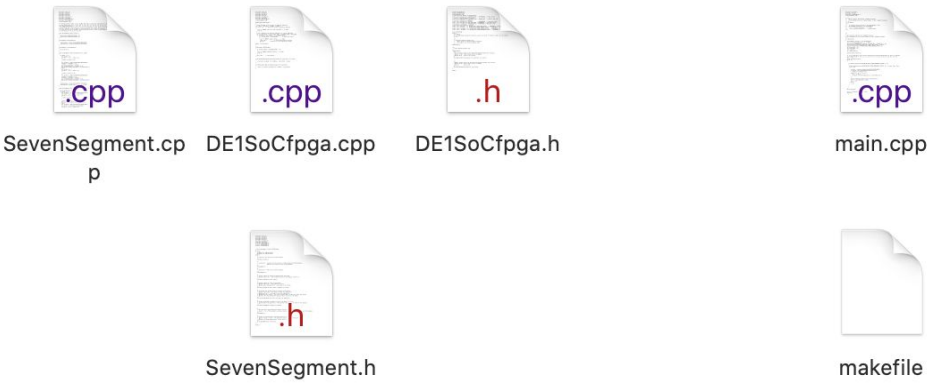
file contents

Lab 3.2

interfacing with the board after all headers/source files were made, using makefile

```
1. Erase all elements
2. Clear specific element
3. Write a specific element to a specific index
4. Write a number to the display
5. Exit
Select an option: _4
Write a number to the display
Enter the number: _100
Main menu:
1. Erase all elements
2. Clear specific element
3. Write a specific element to a specific index
4. Write a number to the display
5. Exit
Select an option: _2
Clear specific element
Enter the index to clear: _4
Main menu:
1. Erase all elements
2. Clear specific element
3. Write a specific element to a specific index
4. Write a number to the display
5. Exit
Select an option: _
```

file contents



Lab 3.3

running make clean and make on the board, then running the program

```
root@de1soclinux:~/Labs/Lab3# ls
DE1SoCfpga.cpp  SevenSegment      main.cpp            pushbuttons.cc
DE1SoCfpga.h    SevenSegment.cpp  main.o              pushbuttonsclass.cc
DE1SoCfpga.o    SevenSegment.h    makefile            pushbuttonscpp
Prelab3         SevenSegment.o    pushbuttonclass.cc
root@de1soclinux:~/Labs/Lab3# make clean
rm DE1SoCfpga.o main.o SevenSegment.o SevenSegment
root@de1soclinux:~/Labs/Lab3# make
g++    -c -o DE1SoCfpga.o DE1SoCfpga.cpp
g++ -g -Wall -c main.cpp
main.cpp: In function 'int main()':
main.cpp:31:7: warning: unused variable 'entervalue' [-Wunused-variable]
main.cpp:32:7: warning: unused variable 'enterindex' [-Wunused-variable]
main.cpp:34:7: warning: unused variable 'reg0_hexValue' [-Wunused-variable]
g++ -g -Wall -c SevenSegment.cpp
SevenSegment.cpp: In member function 'int SevenSegment::DisplayMenu(char)':
SevenSegment.cpp:239:1: warning: control reaches end of non-void function [-Wret
urn-type]
g++ DE1SoCfpga.o main.o SevenSegment.o -o SevenSegment
root@de1soclinux:~/Labs/Lab3# ./SevenSegment
Program Starting...!
```

Analysis

Lab 3.1

We found the actual abstraction into header and source files to be fairly straightforward, but struggled a bit in the actual implementation and had to consult lecture notes throughout the process to be sure we were doing things correctly. We were a bit confused about the #ifndef and #endif clauses and looked into the C++ documentation to understand this better. We were also unsure where to put method documentation but decided to put it in the header and not the source file to follow the Java conventions we were familiar with.

Lab 3.2

This portion of the lab went fairly similarly to the previous portion, and we were able to draw upon our new knowledge of how to abstract C++ code to quickly separate SevenSegments into a header and source file. In this portion of the lab, we also created a main file that called the SevenSegments instance, and a makefile. The main went fairly smoothly but we had some trouble with the makefile. We were confused about how we were meant to name the makefile, and whether it needed an extension. After doing some research we came across the fact that the makefile doesn't use an extension. In trying to test this on the board, we came across an issue with spacing and discovered that the spacing of the file matters, and the compiler is particular about where newlines and tabs are. Once we figured this out, we were able to smoothly run the program and it worked as it did before!

Lab 3.3

We had a really difficult time with this part of the lab, exacerbated by our board freezing every time we tested a build, greatly slowing our progress. We were confused about how to use the timer, and our confusion was amplified by a bug in our code we attributed to not properly understanding the timing when in fact it was a simple and easy-to-miss error with counting and iteration. We spent a long time with this portion of the lab, but we did eventually get a somewhat working solution. However, this solution was one that didn't work, because our prelab implementation of ClearSpecific was broken, which broke the final solution for our code.

Conclusion

In this lab, we learned more object oriented design principles as we further abstracted our code. Additionally, we learned more about writing and interfacing with the board, this time making use of the six different seven segmented displays. We were able to write to these displays, displaying hex codes and letters, and we learned how to use timers to cycle between different displays and write to the board at fixed intervals.

Appendix

Lab 3.1

```
#ifndef DE1SOCFPGA_H
#define DE1SOCFPGA_H
using namespace std;
// Physical base address of FPGA Devices
static const unsigned int LW_BRIDGE_BASE  = 0xFF200000;  // Base offset
// Length of memory-mapped IO window
static const unsigned int LW_BRIDGE_SPAN  = 0x00DEC700;  // Address map size
// Cyclone V FPGA device addresses
static const unsigned int SW_BASE         = 0x00000040;  // Switches offset
static const unsigned int HEX3_HEX0_BASE = 0x00000020; // HEX Reg1 offset
static const unsigned int HEX5_HEX4_BASE = 0x00000030;// HEX Reg2 offset
//0xFFFEC600 -0xFF200000 = 0xDEC600
static const unsigned  int  MPCORE_PRIV_TIMER_LOAD_OFFSET  =  0xDEC600;//  Points
to  LOAD Register
static const unsigned  int  MPCORE_PRIV_TIMER_COUNTER_OFFSET  =  0xDEC604;//
Points  to COUNTER Register
static const  unsigned  int  MPCORE_PRIV_TIMER_CONTROL_OFFSET  =  0xDEC608;//
Points  to CONTROL Register
static const  unsigned  int  MPCORE_PRIV_TIMER_INTERRUPT_OFFSET  =  0xDEC60C;//
Points  to INTERRUPT Register

class DE1SoCfpga {
public:
  char *pBase;
  // File descriptor passed by reference, where the result of function 'open' will be stored.
  int fd;

  /**
   * Initialize general-purpose I/O
   *  - Opens access to physical memory /dev/mem
   *  - Maps memory into virtual address space
   */
DE1SoCfpga();

/**
  * Close general-purpose I/O.
*/
~DE1SoCfpga();
```

```
 /**
 * 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 RegisterWrite(unsigned int reg_offset, int value);


 /**
 * Read a 4-byte value from the specified general-purpose I/O location.
 * @param offset  Offset where device is mapped.
 * @return        Value read.
 */
 int RegisterRead(unsigned int reg_offset);
};

#endif

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <iostream>
#include "DE1SoCfpga.h"
using namespace std;

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;
}

DE1SoCfpga::~DE1SoCfpga()
{
  if (munmap (pBase, LW_BRIDGE_SPAN) != 0)
  {
   cout << "ERROR: munmap() failed..." << endl;
   exit(1);
   }
 close (fd);    // close memory
}

void DE1SoCfpga::RegisterWrite(unsigned int reg_offset, int value)
{
  * (volatile unsigned int *)(pBase + reg_offset) = value;
}

int DE1SoCfpga::RegisterRead(unsigned int reg_offset)
{
   return * (volatile unsigned int *)(pBase + reg_offset);
}
```

Lab 3.2

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <iostream>
#include "DE1SoCfpga.h"
#ifndef SEVENSEGMENT_H
#define SEVENSEGMENT_H
```

```
class SevenSegment : public DE1SoCfpga
{
private:
  unsigned int reg0_hexValue;
  unsigned int reg1_hexValue;
public:
  /*
   * Clears all the values on to the displays.
   */
  void Hex_ClearAll();

  /*
   * Constructor - assigns intitial values to reg0_hexValue and reg1_hexValue
   *               based on the intital state of the displays.
   */
  SevenSegment();

  /*
   * Destructor - Clears the current displays.
   */
  ~SevenSegment();

  /*
   * Clears a specific display as specified by the index.
   * @Param index (int) - the 0-indexed index of the display, from 0 to 5
   */
  void Hex_ClearSpecific(int index);

  /*
   * Writes a specific value to the board.
   * @Param index (int) - the index to write to
   * @Param value (unsigned int) - the value to be written
   */
  void Hex_WriteSpecific(int index, unsigned int value);

  /*
   * Calculates the display value to display on the board
   * @param start (int) - The index to start the indexing at
   * @param end (int) - The index to end the indexing at
   * @param digits[] (array) - The list of digits who's values will query bit_values
```

```
 * @Returns int representing the display value to write to the board
 */
int CalculateDisplay(int start, int end, int digits[]);

/*
 * Writes the passed-in number in hex to the board's display.
 * @param number (unsigned int) - The base-10 integer value to write to the display
 */
void Hex_WriteNumber(unsigned int number);


/*
 * Runs the menu, displaying menu options and getting user input.
 * @return : int if the program is exited, representing the user wanting to quit the program
 */
int RunMenu();

/*
 * Allows for functionality selection. Returns int when quit.
 * @param selection (char) - the character value for the selected option
 * @Returns int when program quit to pass back to main
 */
int DisplayMenu(char selection);
};

#endif

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <iostream>
#include "SevenSegment.h"
#include "DE1SoCfpga.h"
using namespace std;

const unsigned int bit_values[16] = {63, 6, 91, 79,102, 109, 125, 7, 127, 111, 119, 124, 57, 94,
121, 113};
```

```
const string menuString = "Main menu:\n1. Erase all elements\n2. Clear specific element\n3. Write a specific element to a specific index\n4. Write a number to the display\n5. Exit \nSelect an option: _";


void SevenSegment::Hex_ClearAll()
{
  RegisterWrite(HEX3_HEX0_BASE, 0);
  RegisterWrite(HEX5_HEX4_BASE, 0);
}

SevenSegment::SevenSegment()
{
  reg0_hexValue = RegisterRead(HEX3_HEX0_BASE);
  reg1_hexValue = RegisterRead(HEX5_HEX4_BASE);
}

SevenSegment::~SevenSegment()
{
  Hex_ClearAll();
}


void SevenSegment::Hex_ClearSpecific(int index)
{
  int curValue = RegisterRead(HEX5_HEX4_BASE);
  if(index > 3) {
    int bitVal = curValue;
    if(index == 4) {
      bitVal = bitVal << 8;
      bitVal = bitVal >> 8;
    } else {
      bitVal = bitVal >> 8;
      bitVal = bitVal << 8;
    }
    bitVal = ~bitVal;

    int clearedState4_5 = curValue & bitVal;
    RegisterWrite(HEX5_HEX4_BASE, clearedState4_5);
  } else {
    int bitVal = curValue;
```

```
   for(int i = 0; i < index; i++)
   {
     bitVal = bitVal << 8;
   }
   int curValue = RegisterRead(HEX3_HEX0_BASE);
   curValue = curValue << 4;
   bitVal = bitVal | curValue;
   int clearedState3_0 = bitVal >> 4;
   RegisterWrite(HEX3_HEX0_BASE, clearedState3_0);
  }

  reg0_hexValue = RegisterRead(HEX3_HEX0_BASE);
  reg1_hexValue = RegisterRead(HEX5_HEX4_BASE);
}

void SevenSegment::Hex_WriteSpecific(int index, unsigned int value)
{
  Hex_ClearSpecific(index);
  if(index > 3) {
    int curValue = RegisterRead(HEX5_HEX4_BASE);
    int bitVal = bit_values[value];
    for(int i = 4; i < index; i++)
    {
      bitVal = bitVal << 8;
    }

    curValue = curValue | bitVal;
    RegisterWrite(HEX5_HEX4_BASE, curValue);
  } else {
    int curValue = RegisterRead(HEX3_HEX0_BASE);
    int bitVal = bit_values[value];
    for(int i = 0; i < index; i++)
    {
      bitVal = bitVal << 8;
    }
    curValue = curValue | bitVal;
    RegisterWrite(HEX3_HEX0_BASE, curValue);
  }

}
```

```
int SevenSegment::CalculateDisplay(int start, int end, int digits[]) {
  bool first = true;
  int display = 0;
  // for the first four displays, display the proper hex value
  for(int j = start; j > end; j--) {
    // get the decimal value that represents the display for the j'th hex digit
    int value = bit_values[digits[j]];
    // if it's the first display, we don't want to shift over the bits,
      // we just want toset the display as the bits
    if(first == true) {
      display = value;
      first = false;
    // otherwise, we want to shift over the bits by 8 and then add the bits to the display
    } else {
      display = display << 8 | value;
    }
  }
  return display;
}

void SevenSegment::Hex_WriteNumber(unsigned int number)
{
  // create an array of 6 elements representing the value to write to each separate display
  int digits[6];
  // there are 6 possible display values
  for (int i = 0; i < 6; i++) {
    // get the first hexadecimal number by using modulo to calculate the base-16 representation
    digits[i] = number % 16;
    // divide the number by 16 to shrink it down to the next power of 16
    number = number / 16;
  }

  // calculate the values to display on the two sets of displays
  int display = CalculateDisplay(3, -1, digits);
  int display2 = CalculateDisplay(5, 3, digits);
  // display = 0 | bit_values[digits[0]];
  // cout << "bits: " << bit_values[digits[0]] << endl;

  RegisterWrite(HEX3_HEX0_BASE, display);
  RegisterWrite(HEX5_HEX4_BASE, display2);
```

```
  reg1_hexValue = RegisterRead(HEX3_HEX0_BASE);
  reg0_hexValue = RegisterRead(HEX5_HEX4_BASE);
}


int SevenSegment::RunMenu()
{
  char selection;
  cout << menuString;
  cin >> selection;
  return DisplayMenu(selection);
}

int SevenSegment::DisplayMenu(char selection)
{
  int clearIndex = 0;
  int writeIndex = 0;
  int writeElement = 0;
  int number = 0;
  switch (selection) {
    // clear all elements
    case '1':
      cout << "Clear all elements" << endl;
      Hex_ClearAll();
      RunMenu();
      break;
    // clear specific element
    case '2':
      cout << "Clear specific element" << endl;
      cout << "Enter the index to clear: _";
      cin >> clearIndex;
      Hex_ClearSpecific(clearIndex);
      RunMenu();
      break;
    // write specific element
    case '3':
      cout << "Write a specific element to a specific index" << endl;
      cout << "Enter the index to write to: _";
      cin >> writeIndex;
      cout << endl;
      cout << "Enter the element to write: _";
```

```
      cin >> writeElement;
      Hex_WriteSpecific(writeIndex, writeElement);
      RunMenu();
      break;
    // write a number to the display
    case '4':
      cout << "Write a number to the display" << endl;
      cout << "Enter the number: _";
      cin >> number;
      Hex_WriteNumber(number);
      RunMenu();
      break;
    // exit the program
    case '5':
      Hex_ClearAll();
      return 0;
      break;
    // if invalid, display an error and run the menu again
    default:
      cout << "Error: please enter a valid option" << endl;
      RunMenu();
      break;
  }
}
};

int main(void)
{
  // Create a pointer object of the SevenSegment class
  int hex_value = 0;
  SevenSegment *display = new SevenSegment;
  cout << "Program Starting...!" << endl;
   return display->RunMenu();   // display menu
}

# SevenSegment program Makefile
SevenSegment: DE1SoCfpga.o main.o SevenSegment.o
        g++ DE1SoCfpga.o main.o SevenSegment.o -o SevenSegment

main.o: main.cpp SevenSegment.h
        g++ -g -Wall -c main.cpp
```

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

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

Lab 3.3

```
#include <iostream>
#include <string>
#include "SevenSegment.h"
using namespace std;

// /*
//  * Main operates the DE1-SoC 7-Segment Displays
//  * This program writes an integer number on the 7-Segment Displays
//  */
// int main()
// {
//   // Create a pointer object of the SevenSegment class
//   SevenSegment *display = new SevenSegment;
//   cout << "Program Starting...!" << endl;
//    return display->RunMenu();   // display menu
// }


/*
*
* Main operates the DE1-SoC 7-Segment Displays
* This program writes an integer number on the 7-Segment Displays
*/
int main(void)
{
  SevenSegment *display = new SevenSegment;
  cout << "Program Starting...!" << endl;
  int counter= 200000000;// timeout = 1/(200 MHz) x 200x10^6 = 1 sec
```

```
 display->RegisterWrite(MPCORE_PRIV_TIMER_LOAD_OFFSET, counter);
 display->RegisterWrite(MPCORE_PRIV_TIMER_CONTROL_OFFSET, 3);
 int entervalue = 0;
 int enterindex = 0;
 int curValue;
 int reg0_hexValue;

 display->Hex_ClearAll();

 // list of indices of the letters should query from bit_alpha_values in order or display
 int letterVals[18] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17};
 int i = 0;
 //cin >> bitentervalue;
 while (1)
 {

  if (display->RegisterRead(MPCORE_PRIV_TIMER_INTERRUPT_OFFSET) != 0)
  {
   display->RegisterWrite(MPCORE_PRIV_TIMER_INTERRUPT_OFFSET, 1); // reset timer
flag
    if(i < 18)
    {
     curValue = display->RegisterRead(HEX3_HEX0_BASE);
     // display ->Hex_WriteNumber(letterVals[i]);
       curValue = curValue << 8;
     for(int j = 1; j < 5; j++) {
       if(i - j > 0) {
         display ->Hex_WriteSpecific(j, letterVals[i-j]);
       }
     }
     display ->Hex_WriteSpecific(0, letterVals[i]);
     cout << letterVals[i] << endl;
     i = i + 1;
    } else {
     i = 0;
    }

  }
 }
 delete display;
```

```
 cout << "Terminating...!" << endl;
 return 0;
   }

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <iostream>
#include "SevenSegment.h"
#include "DE1SoCfpga.h"
using namespace std;

int bit_values[16] = {63, 6, 91, 79,102, 109, 125, 7, 127, 111, 119, 124, 57, 94, 121, 113};
int bit_alpha_values[18] = {119, 124, 57, 94, 121, 113, 118, 6, 14, 56, 84, 63, 115, 80, 109, 62,
102, 91};
char alpha_values[18] = {   'a', 'b', 'c','d','e', 'f', 'h','i','j','l','n','o', 'p','r', 's', 'u', 'y', 'z'};
string menuString = "Main menu:\n1. Erase all elements\n2. Clear specific element\n3. Write a
specific element to a specific index\n4. Write a number to the display\n5. Exit \nSelect an option:
_";
// create an array of 6 elements representing the value to write to each separate display
int curValue[6] = {0, 0, 0, 0, 0, 0};

void SevenSegment::Hex_ClearAll()
{
 RegisterWrite(HEX3_HEX0_BASE, 0);
 RegisterWrite(HEX5_HEX4_BASE, 0);
}

SevenSegment::SevenSegment()
{
 reg0_hexValue = RegisterRead(HEX3_HEX0_BASE);
 reg1_hexValue = RegisterRead(HEX5_HEX4_BASE);
}

SevenSegment::~SevenSegment()
{
 Hex_ClearAll();
}
```

```
void SevenSegment::Hex_ClearSpecific(int index)
{
  if(index > 3) {
    int bitVal = 0;
    for(int i = 4; i < index; i++)
    {
      bitVal = bitVal << 8;
    }
    int curValue = RegisterRead(HEX5_HEX4_BASE);
    curValue = curValue << 4;
    curValue = curValue | bitVal;
    int clearedState4_5 = curValue >> 4;
    RegisterWrite(HEX5_HEX4_BASE, clearedState4_5);
  } else {
    int bitVal = 0;
    for(int i = 0; i < index; i++)
    {
      bitVal = bitVal << 8;
    }
    int curValue = RegisterRead(HEX3_HEX0_BASE);
    curValue = curValue << 4;
    curValue = curValue | bitVal;
    int clearedState3_0 = curValue >> 4;
    RegisterWrite(HEX3_HEX0_BASE, clearedState3_0);
  }

  reg0_hexValue = RegisterRead(HEX3_HEX0_BASE);
  reg1_hexValue = RegisterRead(HEX5_HEX4_BASE);
}

void SevenSegment::Hex_WriteSpecific(int index, unsigned int value)
{
  Hex_ClearSpecific(index);
  if(index > 3) {
    int curValue = RegisterRead(HEX5_HEX4_BASE);
    int bitVal = bit_alpha_values[value];
    for(int i = 4; i < index; i++)
    {
      bitVal = bitVal << 8;
    }
```

```
    curValue = curValue | bitVal;
    RegisterWrite(HEX5_HEX4_BASE, curValue);
   } else {
    int curValue = RegisterRead(HEX3_HEX0_BASE);
    int bitVal = bit_alpha_values[value];
    for(int i = 0; i < index; i++)
    {
      bitVal = bitVal << 8;
    }
    curValue = curValue | bitVal;
    RegisterWrite(HEX3_HEX0_BASE, curValue);
   }

}

// int SevenSegment::CalculateDisplay(int start, int end, int values[]) {
//   bool first = true;
//   int display = 0;
//   // for the first four displays, display the proper hex value
//   for(int j = start; j > end; j--) {
//     // get the decimal value that represents the display for the j'th hex digit
//     int value = bit_alpha_values[values[j]];
//     // if it's the first display, we don't want to shift over the bits,
//       // we just want toset the display as the bits
//     if(first == true) {
//       display = value;
//       first = false;
//     // otherwise, we want to shift over the bits by 8 and then add the bits to the display
//     } else {
//       display = display << 8 | value;
//     }
//   }
//   return display;
// }
//
// void SevenSegment::Hex_WriteNumber(unsigned int number)
// {
//
//   // // there are 6 possible display values
//   // for (int i = 0; i < 6; i++) {
```

```
//   //   // get the first hexadecimal number by using modulo to calculate the base-16
representation
//   //   digits[i] = number % 16;
//   //   // divide the number by 16 to shrink it down to the next power of 16
//   //   number = number / 16;
//   // }
//
//   for(int i = 0; i < 6; i++) {
//     if(curValue[i] != 0) {
//       curValue[i] = number;
//     }
//   }
//
//   // calculate the values to display on the two sets of displays
//   int display = CalculateDisplay(3, -1, curValue);
//   int display2 = CalculateDisplay(5, 3, curValue);
//
//   RegisterWrite(HEX3_HEX0_BASE, display);
//   RegisterWrite(HEX5_HEX4_BASE, display2);
//
//   reg1_hexValue = RegisterRead(HEX3_HEX0_BASE);
//   reg0_hexValue = RegisterRead(HEX5_HEX4_BASE);
// }


int SevenSegment::CalculateDisplay(int start, int end, int digits[]) {
  bool first = true;
  int display = 0;
  // for the first four displays, display the proper hex value
  for(int j = start; j > end; j--) {
    // get the decimal value that represents the display for the j'th hex digit
    int value = bit_values[digits[j]];
    // if it's the first display, we don't want to shift over the bits,
      // we just want toset the display as the bits
    if(first == true) {
      display = value;
      first = false;
    // otherwise, we want to shift over the bits by 8 and then add the bits to the display
    } else {
      display = display << 8 | value;
    }
```

```
  }
  return display;
}

void SevenSegment::Hex_WriteNumber(unsigned int number)
{
  // create an array of 6 elements representing the value to write to each separate display
  int digits[6];
  // there are 6 possible display values
  for (int i = 0; i < 6; i++) {
    // get the first hexadecimal number by using modulo to calculate the base-16 representation
    digits[i] = number % 16;
    // divide the number by 16 to shrink it down to the next power of 16
    number = number / 16;
  }

  // calculate the values to display on the two sets of displays
  int display = CalculateDisplay(3, -1, digits);
  int display2 = CalculateDisplay(5, 3, digits);
  // display = 0 | bit_values[digits[0]];
  // cout << "bits: " << bit_values[digits[0]] << endl;

  RegisterWrite(HEX3_HEX0_BASE, display);
  RegisterWrite(HEX5_HEX4_BASE, display2);

  reg1_hexValue = RegisterRead(HEX3_HEX0_BASE);
  reg0_hexValue = RegisterRead(HEX5_HEX4_BASE);
}
int SevenSegment::RunMenu()
{
  char selection;
  cout << menuString;
  cin >> selection;
  return DisplayMenu(selection);
}

int SevenSegment::DisplayMenu(char selection)
{
  int clearIndex = 0;
  int writeIndex = 0;
  int writeElement = 0;
```

```
int number = 0;
switch (selection) {
  // clear all elements
  case '1':
    cout << "Clear all elements" << endl;
    Hex_ClearAll();
    RunMenu();
    break;
  // clear specific element
  case '2':
    cout << "Clear specific element" << endl;
    cout << "Enter the index to clear: _";
    cin >> clearIndex;
    Hex_ClearSpecific(clearIndex);
    RunMenu();
    break;
  // write specific element
  case '3':
    cout << "Write a specific element to a specific index" << endl;
    cout << "Enter the index to write to: _";
    cin >> writeIndex;
    cout << endl;
    cout << "Enter the element to write: _";
    cin >> writeElement;
    Hex_WriteSpecific(writeIndex, writeElement);
    RunMenu();
    break;
  // write a number to the display
  case '4':
    cout << "Write a number to the display" << endl;
    cout << "Enter the number: _";
    cin >> number;
    Hex_WriteNumber(number);
    RunMenu();
    break;
  // exit the program
  case '5':
    Hex_ClearAll();
    return 0;
    break;
  // if invalid, display an error and run the menu again
```

```
    default:
      cout << "Error: please enter a valid option" << endl;
      RunMenu();
      break;
  }
}
```