Prelab Assignment 3
Seven Segment

Levi Kaplan
kaplan.l@northeastern.edu

Submit Date: 10/1/20
Due Date: 10/1/20

Levi Kaplan EECE2160	Embedded Design: Enabling Robotics Prelab Assignment 3

Digit Table

Character	**Outputs - Segment 0/1**								
#	**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**

Prelab Code

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
```

```
#include <fcntl.h>
#include <sys/mman.h>
#include <iostream>
using namespace std;
// Physical base address of FPGA Devices
const unsigned int LW_BRIDGE_BASE  = 0xFF200000;  // Base offset
// Length of memory-mapped IO window
const unsigned int LW_BRIDGE_SPAN  = 0x00DEC700;  // Address map size
// display offset
const unsigned int HEX3_HEX0_BASE = 0x00000020; // HEX Reg1 offset
const unsigned int HEX5_HEX4_BASE = 0x00000030;// HEX Reg2 offset
// const unsigned int bit_values[16] = {00111111, 00000110, 01011011, 01001111, 01100110,
01101101, 01111101,
//                          00000111, 01111111, 01101111, 01110111, 01111100, 00111001,
01011110,
//                          0111001, 011110001};

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: _";

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()
  {
    // 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.
*/
~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 RegisterWrite(unsigned int reg_offset, int value)
{
  * (volatile unsigned int *)(pBase + reg_offset) = 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)
 {
   return * (volatile unsigned int *)(pBase + reg_offset);
 }


};

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()
 {
  RegisterWrite(HEX3_HEX0_BASE, 0);
  RegisterWrite(HEX5_HEX4_BASE, 0);
 }

 /*
  * Constructor - assigns intitial values to reg0_hexValue and reg1_hexValue
  *               based on the intital state of the displays.
  */
 SevenSegment()
 {
  reg0_hexValue = RegisterRead(HEX3_HEX0_BASE);
  reg1_hexValue = RegisterRead(HEX5_HEX4_BASE);
 }

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

```
}



/*
 * 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)
{

  if(index > 4) {
    int clearedState5_4 = reg1_hexValue & index;
    RegisterWrite(HEX5_HEX4_BASE, clearedState5_4);
  } else {
    int clearedState3_0 = reg0_hexValue & index;
    RegisterWrite(HEX3_HEX0_BASE, clearedState3_0);
  }

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

/*
 * 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)
{
  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);
  }

}

/*
 * 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[]) {
  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;
}

/*
 * 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)
{
  // 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);
}


/*
 * 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()
{
  char selection;
  cout << menuString;
  cin >> selection;
  return DisplayMenu(selection);
}

int 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;
   }
  }
};


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