Madeline Coco, Kaite O’Flaherty	Embedded Design
10/12/2021	Lab 4

Controlling the 7-Segment Displays With Object-Oriented Programming

Assignment 1:

MakeFile Code:

```
lab3a: DE1SoCfpga.o SevenSegment.o main.o
	g++ DE1SoCfpga.o SevenSegment.o main.o -o lab3a

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

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

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

clean:
	rm DE1SoCfpga.o SevenSegment.o lab3a
```

DE1SoCfpga.h Code:

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <iostream>
#ifndef DE1SoCfpga_H
#define DE1SoCfpga_H


class DE1SoCfpga
{
   // Cyclone V FPGA device addresses
   public:
   char *pBase;
   int fd;

   DE1SoCfpga();
   ~DE1SoCfpga();
   void RegisterWrite(unsigned int offset, int value);
   int RegisterRead(unsigned int offset);
};


   #endif
```

DE1SoCfpga.cpp Code:

```
#include "DE1SoCfpga.h"
#include <iostream>

using namespace std;

const unsigned int bitArray[16] = {63, 6, 91, 79, 102, 109, 125, 7, 127, 111, 119, 124, 57, 94, 121,
113};
const unsigned int mask[6] = {281474976710528, 281474976678143, 281474968387583,
281472846004223, 280929515864063, 141836999983103};

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

const unsigned int HEX3_HEX0_OFFSET= 0xFF200020 - LW_BRIDGE_BASE;//;const unsigned int
HEX5_HEX4_OFFSET= //;
const unsigned int HEX5_HEX4_OFFSET= 0xFF200030 - LW_BRIDGE_BASE;//;

     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 offset, int value)
    {
    cout << "in REg WR\n";
        * (volatile unsigned int *)(pBase + offset) = value;
    }

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

SevenSegment.h Code:

```
#include "DE1SoCfpga.h"
#ifndef SEVENSEGMENT_H
#define SEVENSEGMENT_H

using namespace std;

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 index, int value);
    void Hex_WriteNumber(int number);


//turn off all the displays ~SevenSegment() { //Hex_ClearAll();
};

#endif
```

SevenSegment.cpp Code:

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

using namespace std;

const unsigned int bitArray[16] = {63, 6, 91, 79, 102, 109, 125, 7, 127, 111, 119, 124, 57, 94, 121,
113};
const unsigned int mask[6] = {281474976710528, 281474976678143, 281474968387583,
281472846004223, 280929515864063, 141836999983103};

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

const unsigned int HEX3_HEX0_OFFSET= 0xFF200020 - LW_BRIDGE_BASE;//;const unsigned int
HEX5_HEX4_OFFSET= //;
const unsigned int HEX5_HEX4_OFFSET= 0xFF200030 - LW_BRIDGE_BASE;//;

SevenSegment::SevenSegment()
   {
     reg0_hexValue=0;
     reg1_hexValue=0;
DE1SoCfpga::RegisterWrite(HEX3_HEX0_OFFSET, reg0_hexValue);
DE1SoCfpga::RegisterWrite(HEX5_HEX4_OFFSET, reg1_hexValue);

   }

//turn off all the displays
SevenSegment::~SevenSegment()
{
SevenSegment::Hex_ClearAll();
}
```

```
void SevenSegment::Hex_ClearAll()
{
  int val = 0;
  DE1SoCfpga::RegisterWrite(HEX3_HEX0_OFFSET, val);
  DE1SoCfpga::RegisterWrite(HEX5_HEX4_OFFSET, val);
}

//clears (turns off) a specified 7-segment display specified by indexwhere the index (0 to 5)
represents one of the six displays.
void SevenSegment::Hex_ClearSpecific(int index)
{
   reg0_hexValue = DE1SoCfpga::RegisterRead(HEX3_HEX0_OFFSET);
   reg1_hexValue = DE1SoCfpga::RegisterRead(HEX5_HEX4_OFFSET);

  if ((index >=0)&&(index <= 3))
  {
     reg0_hexValue = (mask[index] & reg0_hexValue);
     DE1SoCfpga::RegisterWrite(HEX3_HEX0_OFFSET, reg0_hexValue);
  }
  if ((index >=4)&&(index<=5))
  {
     reg1_hexValue = (mask[index-4] & reg1_hexValue);
     DE1SoCfpga::RegisterWrite(HEX5_HEX4_OFFSET, reg1_hexValue);
  }
}

   //writes the digit or character value(from Figure 2 above) to the specified 7-segment display
specified by indexwhere the index (0 to 5) represents one of the six displays.
void SevenSegment::Hex_WriteSpecific(int index, int value)
   {
   //RegisterWrite(HEX3_HEX0_OFFSET, bitArray[value]);
    SevenSegment::Hex_ClearSpecific(index);
    value = value & 0xf;
    int original;
   if ((index >=0)&&(index <= 3))
  {
  //int current;
     //current= object1.RegisterRead(HEX3_HEX0_OFFSET);
     original = DE1SoCfpga::RegisterRead(HEX3_HEX0_OFFSET);
     reg0_hexValue = (original | (bitArray[value] << (index * 8)));
     DE1SoCfpga::RegisterWrite(HEX3_HEX0_OFFSET, reg0_hexValue);
  }
  if ((index >=4)&&(index<=5))
  {
     original = DE1SoCfpga::RegisterRead(HEX5_HEX4_OFFSET);
     reg1_hexValue  = (original | (bitArray[value] << ((index-4) * 8)));
     DE1SoCfpga::RegisterWrite(HEX5_HEX4_OFFSET, reg1_hexValue);
  }


   }

   //writes a positive or negative number tothe 7-segment displays.
   void SevenSegment::Hex_WriteNumber(int number)
   {
           int x = 0;
           int counter = 1;
       for(int loop = 0; loop <6; loop++)
{
           x = (((0x00000F << (loop * 4)) & number) / counter);
           counter = counter * 16;
           SevenSegment::Hex_WriteSpecific(loop, x);
}

   }
```

Main.cpp Code:

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


int main()
{
    int choice;
    int index;
    int val;
    int number;
    SevenSegment *display = new SevenSegment;

    while (choice !=5)
    {
```

```
    cout << "Please choose an option below:" << endl;
    cout<< "1. Turn off a specific index (0-5)."<<endl;
    cout<<"2. Write a new value to a specific index(0-5)."<< endl;
    cout << "3. Write a number to the display." << endl;
    cout << "4. Clear all index values." << endl;
    cout << "5. Exit!" << endl;
    cin >> choice;

    switch(choice)
    {
    case 1:
        cout << "Which index do you want to clear?" << endl;
        cin >> index;

        if ((index <= 5) && (index >= 0))
        {
            display->Hex_ClearSpecific(index);
        }
        else
        {
            cout << "You chose an invalid option." << endl;
        }
break;

    case 2:
        cout << "Which index do you want to replace?" << endl;
        cin >> index;

        if ((index <= 5) && (index >= 0))
        {
            cout << "What is the decimal value you want?" << endl;
            cin >> val;
            display->Hex_WriteSpecific(index, val);
        }
        else
        {
            cout << "You chose an invalid option." << endl;
        }
break;

    case 3:
        cout<< "Please enter the number you want to display"<<endl;
        cin>>number;
        display->Hex_WriteNumber(number); break;
    case 4:
            cout << "Clearing all values from the display..." << endl;
            display->Hex_ClearAll(); break;
    default:
            cout << "Wrong." << endl;
break;

}
}

}
```

Timer Code:

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

using namespace std;

const unsigned int bitArray[16] = {63, 6, 91, 79, 102, 109, 125, 7, 127, 111, 119, 124, 57, 94,
121, 113};
const unsigned int mask[6] = {281474976710528, 281474976678143, 281474968387583,
281472846004223, 280929515864063, 141836999983103};

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

const unsigned int SW_OFFSET = 0xFF200040 - LW_BRIDGE_BASE;
const unsigned int HEX3_HEX0_OFFSET= 0xFF200020 - LW_BRIDGE_BASE;
const unsigned int HEX5_HEX4_OFFSET= 0xFF200030 - LW_BRIDGE_BASE;

const unsigned int MPCORE_PRIV_TIMER_LOAD_OFFSET = 0xFFFEC600 -
LW_BRIDGE_BASE;             //points to load register
const unsigned int MPCORE_PRIV_TIMER_COUNTER_OFFSET = 0xFFFEC604 -
LW_BRIDGE_BASE;                         //points to COUNTER register
const unsigned int MPCORE_PRIV_TIMER_CONTROL_OFFSET = 0xFFFEC608 -
LW_BRIDGE_BASE;                         //points to control register
const unsigned int MPCORE_PRIV_TIMER_INTERRUPT_OFFSET = 0xFFFEC60C -
LW_BRIDGE_BASE;             //points to interrupt register
```

```
class DE1SoCfpga
{
public:
    char *pBase;
    int fd;
    int value;
    DE1SoCfpga() //Constructor - Initialize
    {
        // 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() //Destructor - Finalize
    {
        if (munmap (pBase, LW_BRIDGE_SPAN) != 0)
        {
            cout << "ERROR: munmap() failed..." << endl;
            exit(1);
        }
        close (fd); // close memoryC700;
    }
    void RegisterWrite(unsigned int offset, int value)
    {
        * (volatile unsigned int *)(pBase + offset) = value;
    }
    int RegisterRead(unsigned int offset)
    {
        return * (volatile unsigned int *)(pBase + offset);
    }
};


class SevenSegment:public DE1SoCfpga
{
private:
    unsigned int reg0_hexValue;
    unsigned int reg1_hexValue;
    unsigned int initialvalueLoadMPCore;
    unsigned int initialvalueControlMPCore;
    unsigned int initialvalueInterruptMPCore;


public:
    void Hex_ClearAll()
    {
        reg0_hexValue = 0;
        reg1_hexValue = 0;
        DE1SoCfpga::RegisterWrite(HEX3_HEX0_OFFSET, reg0_hexValue);
        DE1SoCfpga::RegisterWrite(HEX5_HEX4_OFFSET, reg1_hexValue);
    }
    SevenSegment() //Constructor - Initialize
    {
        DE1SoCfpga::RegisterRead(reg0_hexValue);
        DE1SoCfpga::RegisterRead(reg1_hexValue);
            initialvalueLoadMPCore = RegisterRead(MPCORE_PRIV_TIMER_LOAD_OFFSET);
            initialvalueControlMPCore =
RegisterRead(MPCORE_PRIV_TIMER_CONTROL_OFFSET);
            initialvalueInterruptMPCore =
RegisterRead(MPCORE_PRIV_TIMER_INTERRUPT_OFFSET);
    }
    ~SevenSegment()
    {
        Hex_ClearAll();
            RegisterWrite(MPCORE_PRIV_TIMER_LOAD_OFFSET, initialvalueLoadMPCore);
            RegisterWrite(MPCORE_PRIV_TIMER_CONTROL_OFFSET,
initialvalueControlMPCore);
            RegisterWrite(MPCORE_PRIV_TIMER_INTERRUPT_OFFSET,
initialvalueInterruptMPCore);
    }
    void Hex_ClearSpecific(int index)
    {
        if (index >= 0 & index <= 3)
        {
            reg0_hexValue = DE1SoCfpga::RegisterRead(HEX3_HEX0_OFFSET);
            reg0_hexValue = reg0_hexValue & ~( 0x7F << (index * 8));
            // cout << reg0_hexValue << endl;
            DE1SoCfpga::RegisterWrite(HEX3_HEX0_OFFSET, reg0_hexValue);
        }
        if (index >= 4 & index <= 5)
        {
            reg1_hexValue = DE1SoCfpga::RegisterRead(HEX5_HEX4_OFFSET);
            reg1_hexValue = reg1_hexValue & ~( 0x7F << ((index-4) * 8));
            //cout << reg1_hexValue << endl;
            DE1SoCfpga::RegisterWrite(HEX5_HEX4_OFFSET, reg1_hexValue);
        }
        else
        {
            // cout << "Please input a valid index" << endl;
        }
    }
    void Hex_WriteSpecific(int index, int value)
    {
//RegisterWrite(HEX3_HEX0_OFFSET, bitArray[value]);
Hex_ClearSpecific(index);
//value = value & 0xf;
int original;
  if ((index >=0)&&(index <= 3))
   {
   //int current;
       //current= object1.RegisterRead(HEX3_HEX0_OFFSET);
       original = RegisterRead(HEX3_HEX0_OFFSET);
reg0_hexValue = (original | (bitArray[value] << (index * 8)));
       RegisterWrite(HEX3_HEX0_OFFSET, reg0_hexValue);
   }
   if ((index >=4)&&(index<=5))
   {


       original = RegisterRead(HEX5_HEX4_OFFSET);
       reg1_hexValue  = (original | (bitArray[value] << ((index-4) * 8)));
       RegisterWrite(HEX5_HEX4_OFFSET, reg1_hexValue);
   }

    }
        void Hex_WriteNumber(int number)
    {
int x = 0;
int counter = 1;
        for(int loop = 0; loop <6; loop++)
{
x = (((0x00000F << (loop * 4)) & number) / counter);
counter = counter * 16;
Hex_WriteSpecific(loop, x);
}
    }
};


/**
* 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 = 100000000; // 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 enterindex = 0; //points to Hex0 display
int entervalue = 0; //points to segment0
int count = 20;
int switchvalue = 0;
int switch0 = 0;
int masking = 0xFFFFFF;
int name[11] = {55, 55, 119, 94, 94, 6, 121, 57, 63, 57, 63};

while ((count > 1) && (switch0 == 0))
{
 if (display->RegisterRead(MPCORE_PRIV_TIMER_INTERRUPT_OFFSET) != 0)
 {

 display->RegisterWrite(MPCORE_PRIV_TIMER_INTERRUPT_OFFSET, 1);
 // reset timer flag bit

for(int i = 0; i < 11; i++)
{
 entervalue=entervalue |  name[i]^0x1 ;
// display->Hex_WriteSpecific(enterindex, entervalue);
 display -> RegisterWrite(HEX3_HEX0_OFFSET,name[i]);
 //entervalue=entervalue;
}

switchvalue = display->RegisterRead(SW_OFFSET);
switch0 = (switchvalue & (0x1));
count = count - 1;
 }

}
delete display;

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


/*

name[i] i = 0 to 11

writeName(allvalue[])
fro(i = 0 to 10)

value = value | name[i]

registerWrite(HEX..,value)

value = value << 8
*/
```