Lab Assignment 3b

Jack Fenton
fenton.j@husky.neu.edu
Nicholas Abadir
abadir.n@husky.neu.edu

Submit date: 1/13/2020
Due Date: 1/13/2020

Abstract

In this lab we further delved into memory mapping I/O instead focusing on the buttons within the zedboard to control the LED's versus terminal input. We then converted the program to use classes as to begin building our skills in object oriented design. With object oriented design, it allows other people to collaborate and read the program with ease versus a continuous main with abundant functions.

Introduction

The goal of this lab was to further solidify our knowledge of memory mapping I/O by using the button to manipulate other circuit elements such as the LEDs. Using the ZED board buttons to manipulate its own elements brings us to a new level of embedded designs and gives us insight to how many things we use today, such as a tv remote or controller, work to manipulate its own elements as well as other programs. We also incorporated object oriented design to make our code more secure while also making it easier to manipulate for other programs and uses. This modularity helps in real world areas as people can more easily read our code and understand it as we break it up piece by piece in classes versus a multitude of functions and lengthy main.

Lab Discussion

Assignment 1 was to incorporate the button functionality to control the LEDs while also representing a binary number. The order was that the up and down button respectively increased the number 1 or decreased the number by 1. The left and right would move the binary number by one digit respectively left or right, with the center resetting the LED's to the number represented on the switch. This was achieved within the function **ButtonCommands** and a switch case which would be activated depending on the address received which was nicely converted into numbers 1-5.

Assignment 2 was to then convert our current program into one that uses classes so as to have our program in object oriented design. This was done through the class **ZedBoard**, **LEDs, Switches, Buttons**, and **ZedMenu**. These all communicate with each other to perform the functionality of the previous program and their names indicate the elements the classes refer to. This allows us to organize the functions and commands relevant to each element while also further increasing security.

Assignment 3 was then to copy all PushButtonClass.cpp into a new file named CounterSpeed.cpp with new functionality for the buttons. The center keeps its functionality while the up and down button increment or decrement respectively the tick speed of the counter by 1 tick/sec. The left and right buttons now choose the direction in which the counter works in their respective directions: left or right. This automatic counter will change the LED's depending on its tick speed rather than user controlled with the buttons controlling the speed and direction of this automatic function.

Results and Analysis

Lab 3.4 Controlling the push buttons

Continuing off the code for Lab 3a, we used the previous file and added a function void ButtonCommands which takes in pBase and then takes the address of the button and assigns it a value 1-5 to be used in the switch case which can be seen below.

```
void ButtonCommands(char *pBase)
{
   int cur_butt=0,last_butt=0,count=0;

   WriteAllLeds(pBase,ReadAllSwitches(pBase));
   count=ReadAllSwitches(pBase);
   while(true)
   {
       //Getting Current Button
       cur_butt=PushButtonGet(pBase);
       usleep(100000);
       if (cur_butt!=last_butt)
       {
           switch(cur_butt)
           {
               case 1:
                   //Shift Digits Right
                   count*=2;
                   WriteAllLeds(pBase, count);
                   break;

               case 2:
                   //Shift Digits Left
                   count/=2;
                   WriteAllLeds(pBase, count);
                   break;

               case 3:
                   //Incriment LEDS +1
                   count++;
                   WriteAllLeds(pBase, count);
                   ReadAllLeds(pBase);
                   break;

               case 4:
                   //Incriment LEDS -1
                   count--;
                   WriteAllLeds(pBase, count);
                   break;

               case 5:

                   //Set LEDs to Switches Num
                   WriteAllLeds(pBase,ReadAllSwitches(pBase));
                   break;
               case 6:
                   //exit
                   return;
           }
       }
       last_butt=cur_butt;

   }
}

int PushButtonGet(char *pBase)
{
   int numPress=0;
   for(int i=0; i<5;i++)
   {
       numPress+=(i+1)*Read1Butt(pBase,i);
   }
   return numPress;
}

int Read1Butt(char *pBase, int buttNum)
{
   int buttNum_offset = gpio_pbtnl_offset + 4*buttNum;
   return RegisterRead(pBase, buttNum_offset);
}
```

The function is initialized with the LEDs representing the switches as well as the counter being set to the value represented by the switches. Usleep is then used to delay the loop so to keep the button from overflowing with useless memory. Case 1 Shifts the digits right by multiplying the counter by 2, Case 2 shifts the digits left by dividing the counter by 2, Case 3 increments the counter by 1, Case 4 decrements the counter by 1, and case 5 makes the counter reset and the LEDs to represent the number on the switches.

Lab 3.5 Using C++ Objects

The program was reconfigured with an object oriented approach. This was done using a hierarchy of classes. ZedBoard was the base class for this, which included the initialization and finalization of the ZedBoard in its constructor and destructor respectively. It also included RegisterRead and RegisterWrite. LEDs, which included all functionality of the LEDs, and Switches, which included all functionality of the Switches, were derived from this base class. Buttons included all the functionality needed by the buttons and all the events they could trigger. Because of this, Buttons inherited from ZedBoard, LEDs, and Switches, so the inheritances had to be virtual. The final derived class, which inherited from LEDs, Switches, and Buttons, was the ZedMenu, which was the user interface that allowed for selection of different functionality.
The initialization of all the classes can be seen below. The full contents of their methods can be found in the appendix.

```
class ZedBoard
{
private:
   int fd;
   char *pBase;

public:
   ZedBoard();
   ~ZedBoard();

   void RegisterWrite(int offset, int value);
   int RegisterRead(int offset);

}; //Methods for ZedBoard on lines 207-264


class LEDs : public virtual ZedBoard
{

private:
   int LED_base; //offset of the first led, set in constructor

public:
   LEDs(){LED_base = 0x12C;}
   ~LEDs(){}
   void Write1Led(int ledNum, int state);
   void WriteAllLeds(int numLEDs);

   int Read1Led(int ledNum);
   int ReadAllLeds();

   void Display1Led();
   void DisplayAllLeds();

}; //Methods for LEDs on lines 269-348

class Switches : public virtual ZedBoard
{

private:
   int Switch_base; //offset of the first switch, set in constructor

public:
   Switches(){Switch_base = 0x14C;}
   ~Switches(){}

   int Read1Switch(int switchNum);
   int ReadAllSwitches();

   void Output1Switch();
   void OutputAllSwitches();

}; //Methods for Switches on lines 356-401


class Buttons : public virtual ZedBoard, public virtual LEDs, public virtual Switches
{
private:
   int Button_base; //offset of the first button, set in constructor

   int cur_butt; //current button reading
   int last_butt; //previous button reading
   int count; //current number for display
   int countRate; //number of times count increases per second
   int countDirection; //1 or -1, gives positive or negative direction
   float time_pass; //in seconds, time since last LED update

public:
   Buttons(){Button_base = 0x16C, cur_butt=0, last_butt=0, count=0; countRate=0;
countDirection=1;}
   ~Buttons(){}

   int Read1Butt(int buttNum);
   void PushButtonGet();
   void ButtonCommands();
   void ButtonSelection();
   void Counter();
   void CounterSpeed();
   void CounterChange();


}; //Methods for Buttons on lines 407-579


class ZedMenu : private virtual LEDs, private virtual Switches, private Buttons
{
private:
   int cur_case; //current user choice from the menu, 1-7

public:
   ZedMenu(){cur_case=0;}
   ~ZedMenu(){}

   int Current(){return cur_case;}
   void PromptSelection();
   void ChooseOption();
   void Selection();


}; //Methods for ZedMenu on lines 582-635
```

This approach is more scalable, and prevents the problem of having an absurd number of functions in a single class, but the inheritance is not as simple as it would ideally be. This is only managed using the inheritances as virtual, which allow for branching classes to reconnect without double defining the base class or making multiple instances of it.

Lab 3.6 Counter with Speed Control

Instead of making an entirely new program, the speed control functionality is made an additional option in the menu. Code for this control, which is added to the buttons class, is shown below

```
/*
Main Function for displaying the changing count on LEDs
-Checks for buttons being pressed
-If different than last check, makes an update based on counterspeed
-Counter Change checks if enough time has passed to change display
*/
void Buttons::Counter()
{
   count=ReadAllSwitches();
   WriteAllLeds(count);
   cout << countRate<<endl;
   while(true)
   {
       //Getting Current Button
       PushButtonGet();
       usleep(100000);
       time_pass+=0.1;
       if (cur_butt!=last_butt)
       {
           CounterSpeed();
       }
       last_butt=cur_butt;
       CounterChange();

   }
}

/*
Increases the count represented on the LEDs
determines when to increase based on time since last increase
*/
void Buttons::CounterChange()
{
   if (countRate!=0)
       {
           if((time_pass)>=(float)1/countRate)
```

```
            {
                count+=countDirection;
                time_pass=0;
            }
            WriteAllLeds(count);
        }

}

/*
All possible actions that can be taken based on buttons pressed
*/
void Buttons::CounterSpeed()
{
   switch(cur_butt)
   {
       case 1: //Right Button--Counts Up
           countDirection=1;
           break;

       case 2: //Left Button--Count Down
           countDirection=-1;
           cout<<"Count Backward \n";
           break;

       case 3: //Up Button--Increase Count Speed
           countRate++;
           cout<<"Count Faster \n";
           break;

       case 4: //Down Button--Incriment LEDS -1
           if (countRate>0) countRate--;
           cout<<"Count Slower \n";
           break;

       case 5: //Center Button--Set LEDs to Switches Num
           count=ReadAllSwitches();
           break;

       case 6:
           //exit
```

```
            return;
   }
}
```

Just like in Lab 3.4, the program constantly checks for user input from the buttons, with a small delay to prevent misreadings. When a new button input has been detected, the CounterSpeed function is run, which runs through the switch case, and based on the button pressed, can increase or decrease the rate or flip the direction of counting. CounterChange then checks if a sufficient amount of time has passed since the last change in the LEDs, the threshold of which is 1/countRate, which for a rate of 2 counts/sec would return 0.5. The time is then reset and the count begins again.

Conclusion

This lab further delved into memory mapping I/O and its applications. With the ability of using the circuit boards own buttons to manipulate the LEDs, it is easy to see the further applications of this as we see examples of this every day, for example controlling the brightness on your phone through a slider. Then with object oriented design in mind, implementing classes into our program further allowed us to organize to make our code easier to read and use while increasing the security. Considering this is done throughout the world today, the advantages are obvious as it makes it easier to work between larger groups of people without close contact.

Appendix

Appendix 1: Code for program lab3b2.cpp

```
#include <iostream>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/mman.h>
```

```
#include <string>
#include <math.h>
#include <time.h>

//Jack Fenton and Nicholas Abadir
//Northeastern University
//EECE2150 Embedded Design
//Lab 3b Object Oriented
//13 February 2020


using namespace std;

/* ***** Initialize Global Variables ***** */

// Physical base address of GPIO
const unsigned gpio_address = 0x400d0000;

// Length of memory-mapped IO window
const unsigned gpio_size = 0xff;

/*
const int gpio_led1_offset = 0x12C; // Offset for LED1
const int gpio_led2_offset = 0x130; // Offset for LED2
const int gpio_led3_offset = 0x134; // Offset for LED3
const int gpio_led4_offset = 0x138; // Offset for LED4
const int gpio_led5_offset = 0x13C; // Offset for LED5
const int gpio_led6_offset = 0x140; // Offset for LED6
const int gpio_led7_offset = 0x144; // Offset for LED7
const int gpio_led8_offset = 0x148; // Offset for LED8
const int gpio_sw1_offset = 0x14C; // Offset for Switch 1
const int gpio_sw2_offset = 0x150; // Offset for Switch 2
const int gpio_sw3_offset = 0x154; // Offset for Switch 3
const int gpio_sw4_offset = 0x158; // Offset for Switch 4
const int gpio_sw5_offset = 0x15C; // Offset for Switch 5
const int gpio_sw6_offset = 0x160; // Offset for Switch 6
const int gpio_sw7_offset = 0x164; // Offset for Switch 7
const int gpio_sw8_offset = 0x168; // Offset for Switch 8
const int gpio_pbtnl_offset = 0x16C; // Offset for left push button
const int gpio_pbtnr_offset = 0x170; // Offset for right push button
const int gpio_pbtnu_offset = 0x174; // Offset for up push button
const int gpio_pbtnd_offset = 0x178; // Offset for down push button
const int gpio_pbtnc_offset = 0x17C; // Offset for center push button
*/

/* ***** Class Definitions ***** */

class ZedBoard
{
private:
   int fd;
   char *pBase;

public:
   ZedBoard();
   ~ZedBoard();

   void RegisterWrite(int offset, int value);
   int RegisterRead(int offset);

}; //Methods for ZedBoard on lines 207-264


class LEDs : public virtual ZedBoard
{

private:
   int LED_base; //offset of the first led, set in constructor

public:
   LEDs(){LED_base = 0x12C;}
   ~LEDs(){}

   void Write1Led(int ledNum, int state);
   void WriteAllLeds(int numLEDs);

   int Read1Led(int ledNum);
   int ReadAllLeds();

   void Display1Led();
   void DisplayAllLeds();
```

```
}; //Methods for LEDs on lines 269-348

class Switches : public virtual ZedBoard
{

private:
   int Switch_base; //offset of the first switch, set in constructor

public:
   Switches(){Switch_base = 0x14C;}
   ~Switches(){}

   int Read1Switch(int switchNum);
   int ReadAllSwitches();

   void Output1Switch();
   void OutputAllSwitches();

}; //Methods for Switches on lines 356-401


class Buttons : public virtual ZedBoard, public virtual LEDs, public virtual Switches
{
private:
   int Button_base; //offset of the first button, set in constructor

   int cur_butt; //current button reading
   int last_butt; //previous button reading
   int count; //current number for display
   int countRate; //number of times count increases per second
   int countDirection; //1 or -1, gives positive or negative direction
   float time_pass; //in seconds, time since last LED update

public:
   Buttons(){Button_base = 0x16C, cur_butt=0, last_butt=0, count=0; countRate=0;
countDirection=1;}
   ~Buttons(){}

   int Read1Butt(int buttNum);
   void PushButtonGet();
   void ButtonCommands();
   void ButtonSelection();
   void Counter();
   void CounterSpeed();
   void CounterChange();


}; //Methods for Buttons on lines 407-579


class ZedMenu : private virtual LEDs, private virtual Switches, private Buttons
{
private:
   int cur_case; //current user choice from the menu, 1-7

public:
   ZedMenu(){cur_case=0;}
   ~ZedMenu(){}

   int Current(){return cur_case;}
   void PromptSelection();
   void ChooseOption();
   void Selection();


}; //Methods for ZedMenu on lines 582-635


/*  ***** Initialize Non-Class Functions ***** */

int isInt (int min, int max, string prompt);




/* ************************************** */
/* ***** MAIN FUNCTION FOR PROGRAM ***** */
int main()
{
   ZedMenu myZedMenu;

   myZedMenu.ChooseOption();
   while(myZedMenu.Current()!=7)
   {
       myZedMenu.Selection();
       myZedMenu.ChooseOption();
   }
}






/*
Prints a prompt and takes input, returns input when it is int between min and max

@param min Minimum value the function will allow the user to give
@param max Maximum value the function will allow the user to give
@param prompt optional-reprints this message every time user gives invalid input
*/
int isInt(int min, int max, string prompt=" ")
{
   int input=-1;
   while((input<min) || (input>max))
   {
       cout << prompt;
       cin >> input;
       if (cin.fail())
       {
           cin.clear();
           cin.ignore();
       }
       else break;
   }
   return input;
}




/*
Initialize general-purpose I/O
- Opens access to physical memory /dev/mem
- Maps memory at offset 'gpio_address' into virtual address space
fd File descriptor where the result of function 'open' will be stored.
pBase set to virtual memory which is mapped to physical, or MAP_FAILED on error.
*/
ZedBoard::ZedBoard()
{
   fd = open( "/dev/mem", O_RDWR);


   pBase = (char *) mmap(NULL, gpio_size, PROT_READ | PROT_WRITE, MAP_SHARED,
           fd, gpio_address);

   if (pBase == MAP_FAILED)
   {
       cerr << "Mapping I/O memory failed - Did you run with 'sudo'?\n";
       exit(1); // Returns 1 to the operating system;
   }
}

/*
Close general-purpose I/O.
pBase Virtual address where I/O was mapped
*fd File descriptor previously returned by 'open'
*/
ZedBoard::~ZedBoard()
{
   munmap(pBase, gpio_size);
   close(fd);
}

/*
Write a 4-byte value at the specified general-purpose I/O location.

pBase Base address returned by 'mmap'.
@parem offset Offset where device is mapped.
@param value Value to be written.
*/
void ZedBoard::RegisterWrite(int offset, int value)
{
   * (int *) (pBase + offset) = value;
}
```

```

/*
Read a 4-byte value from the specified general-purpose I/O location.
pBase Base address returned by 'mmap'.
@param offset Offset where device is mapped.
@return Value read.
*/
int ZedBoard::RegisterRead(int offset)
{
   return * (int *) (pBase + offset);
}





/*
Changes the state of an LED (ON or OFF)
address of specified LED, n, is based on the 0th LEDs address + (size of LED
address)*n

@param ledNum LED number (0 to 7)
@param state State to change to (ON or OFF)
*/
void LEDs::Write1Led(int ledNum, int state)
{
   int ledNum_offset = LED_base + 4*ledNum;
   RegisterWrite(ledNum_offset, state);
}

/*
Changes all 8 LEDs to reflect an 8 bit number

@param numLEDs base 10 integer that will be represented by LEDs
*/
void LEDs::WriteAllLeds(int numLEDs)
{
   for (int i=0;i<=7;i++)
   {
       Write1Led(i, numLEDs%2);
       numLEDs/=2;
   }
```

```
}

/*
Reads the value of an LED
- Uses base address of I/O
@param ledNum LED number (0 to 7)
@return LED value read
*/
int LEDs::Read1Led(int ledNum)
{
   int ledNum_offset = LED_base + 4*ledNum;
   return RegisterRead(ledNum_offset);
}

/*
Read the value of the 8 bit number represented by the LEDs
treats each LED as representing a different bit
LED 0 represents the 2^0 bit

@returns integer value of number represented
*/
int LEDs::ReadAllLeds()
{
   int tempInt=0;
   for (int i=0;i<=7;i++)
   {
       tempInt+=Read1Led(i)*(int)pow(2,i);
   }
   return tempInt;
}



/*
Prompts user for an LED to write to and a state
-isInt() only allows for valid inputs to pass
Sends information to Write1LED
*/
void LEDs::Display1Led()
{
   int numIn, stateIn;
   numIn=isInt(0,7,"Choose an LED to set: ");
   stateIn=isInt(0,1,"Choose state for the LED: ");
   Write1Led(numIn, stateIn);
}

/*
Prompts user for number that can be displayed on 8 bits
Sends user input to WriteAllLEDs()
*/
void LEDs::DisplayAllLeds()
{
   int numIn;
   numIn=isInt(0,255,"Choose number to display on LEDs: ");
   WriteAllLeds(numIn);
}









/*
Reads the value of a switch
- Uses base address of I/O
@param switchNum Switch number (0 to 7)
@return Switch value read
*/
int Switches::Read1Switch(int switchNum)
{
   int switchNum_offset = Switch_base + 4*switchNum;
   return RegisterRead(switchNum_offset);
}



/*
Read the value of the 8 bit number represented by the Switches
treats each switch as representing a different bit
switch 0 represents the 2^0 bit

@returns integer value of number represented
*/
```

```
int Switches::ReadAllSwitches()
{
   int tempInt=0;
   for (int i=0;i<=7;i++)
   {
       tempInt+=Read1Switch(i)*(int)pow(2,i);
   }
   return tempInt;
}



void Switches::Output1Switch()
{
   int numIn, stateOut;
   numIn=isInt(0,7,"Choose a Switch to read: ");
   stateOut=Read1Switch(numIn);
   cout<<"The state of Switch "<<numIn<<" is "<<stateOut<<endl;
}



void Switches::OutputAllSwitches()
{
   cout<<"The 8 bit number made by the switches is "<<ReadAllSwitches()<<endl;
}





/*
Reads the value of an button
- Uses base address of I/O
@param buttNum Button number (0 to 5)
@return Button value read
*/
int Buttons::Read1Butt(int buttNum)
{
   int buttNum_offset = Button_base + 4*buttNum;
   return RegisterRead(buttNum_offset);
}

/*
Checks all 5 buttons and sets cur_butt to the number that is being pushed
If multiple buttons pushed, their sum will be returns
*/
void Buttons::PushButtonGet()
{
   int numPress=0;
   for(int i=0; i<5;i++)
   {
       numPress+=(i+1)*Read1Butt(i);

   }
   cur_butt=numPress;
}

/*
Constantly checks the buttons being pressed
If the button is different than it was before, takes corresponding action
*/
void Buttons::ButtonCommands()
{
   cout<<"Starting \n";
   count=ReadAllSwitches();
   WriteAllLeds(count);
   while(true)
   {
       //Getting Current Button
       PushButtonGet();
       usleep(100000);
       if (cur_butt!=last_butt)
       {
           cout<<"Selected "<<cur_butt<<endl;
           ButtonSelection();
       }
       last_butt=cur_butt;
   }
}
```

```
/*
All possible actions that can be taken based on buttons pressed
*/
void Buttons::ButtonSelection()
{
   switch(cur_butt)
   {
       case 1: //Right Button--Shift Digits Right
           count*=2;
           WriteAllLeds(count);
           break;

       case 2: //Left Button--Shift Digits Left
           count/=2;
           WriteAllLeds(count);
           break;

       case 3: //Up Button--Incriment LEDS +1
           count++;
           WriteAllLeds(count);
           ReadAllLeds();
           break;

       case 4: //Down Button--Incriment LEDS -1
           count--;
           WriteAllLeds(count);
           break;

       case 5: //Center Button--Set LEDs to Switches Num
           cout<<"Resetting \n";
           count=ReadAllSwitches();
           WriteAllLeds(count);
           break;

       case 6:
           //exit
           return;
   }
}

/*
```

```
Main Function for displaying the changing count on LEDs
-Checks for buttons being pressed
-If different than last check, makes an update based on counterspeed
-Chounter Change checks if enough time has passed to change display
*/
void Buttons::Counter()
{
   count=ReadAllSwitches();
   WriteAllLeds(count);
   cout << countRate<<endl;
   while(true)
   {
       //Getting Current Button
       PushButtonGet();
       usleep(100000);
       time_pass+=0.1;
       if (cur_butt!=last_butt)
       {
           CounterSpeed();
       }
       last_butt=cur_butt;
       CounterChange();

   }
}

/*
Increases the count represented on the LEDs
determines when to increase based on time since last increase
*/
void Buttons::CounterChange()
{
   if (countRate!=0)
       {
           if((time_pass)>=(float)1/countRate)
           {
               count+=countDirection;
               time_pass=0;
           }
           WriteAllLeds(count);
       }
}

/*
All possible actions that can be taken based on buttons pressed
*/
void Buttons::CounterSpeed()
{
   switch(cur_butt)
   {
       case 1: //Right Button--Counts Up
           countDirection=1;
           break;

       case 2: //Left Button--Count Down
           countDirection=-1;
           cout<<"Count Backward \n";
           break;

       case 3: //Up Button--Increase Count Speed
           countRate++;
           cout<<"Count Faster \n";
           break;

       case 4: //Down Button--Incriment LEDS -1
           if (countRate>0) countRate--;
           cout<<"Count Slower \n";
           break;

       case 5: //Center Button--Set LEDs to Switches Num
           count=ReadAllSwitches();
           break;

       case 6:
           //exit
           return;
   }
}


void ZedMenu::PromptSelection()
```

```
{
   cout << endl <<
           "Main Menu \n\n" <<
           "1. Change an LED \n" <<
           "2. Read a Switch State \n" <<
           "3. Show a Number on LEDs \n" <<
           "4. Display Number on Switches \n" <<
           "5. Button Pressing Bonanza \n" <<
           "6. Counter Speed Control \n" <<
           "7. Exit \n\n";
}

void ZedMenu::ChooseOption()
{
       PromptSelection();
       cur_case=isInt(1,7, "Select an option: ");
}

void ZedMenu::Selection()
{
   switch(cur_case)
   {
       case 1:
           //Change one LED
           Display1Led();
           break;

       case 2:
           //Read Switch
           Output1Switch();
           break;

       case 3:
           DisplayAllLeds();
           break;

       case 4:
           OutputAllSwitches();
           break;

       case 5:
           //Push Button Control
           ButtonCommands();
           break;
       case 6:
           Counter();
           break;
       case 7:
           //exit
           cout << "Exit \n\n";
           break;
   }
}
```