Lab Assignment 3a

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

Submit date: 1/6/2020
Due Date: 1/6/2020

Abstract

This laboratory experiment serves as an introduction to memory mapping I/O. This is achieved by using the addresses provided by the manufacturer on the zed board and in turn controlling or reading the state of the elements. This introduces us to further use on the zed board through memory mapping I/O and further kinds of embedded designs.

Introduction

The purpose of this lab was to learn how to take advantage of memory mapping I/O and controlling or reading elements on the ZED board. While the ZED board has simple elements to work with, it is also the first step into embedded design in which we mesh code and hardware to produce physical results. The program was provided with the addresses and the functions which initialize the map and allow us to manipulate the circuit elements on the board, as well as take information from it such as the number the switches may represent in binary or in contrast, telling the LED's to represent a number in binary.

Lab Discussion

The pre-lab assignment was to simply make the functions `Write1Led` and `Read1Switch` which are quite literal in the name as `Write1Led` will write the LED of your choosing based on number, either a 0 or 1, in which 1 is on and 0 is off. The `Read1Switch` returns a 0 or 1 which states the switch as on or off.

Then another function was created in order to turn all LEDs on in which a number is given and then converted into binary so that it can turn on each LED. This was done by setting each of the LEDs in order a binary digit and then dividing that number by 2 and checking for a remainder. If there was a remainder than the LED's state would switch to a 1 and stay 0 if not.

The final function for this lab was then added in which we read the switch and gave a number as if the switches in an order represented binary numbers. This was done in the same way as the `WriteAllLeds` function. The only difference is we took in a binary digit rather than express one.

Results and Analysis

Lab 3.1 Lab Assignment

The code provided for the prelab was used as the base for the assignment. In order to allow users to provide input for which action they wish to take, the same menu used in Lab 1 and Lab 2 was added into the `main()` function.

```
switch(cur_case)
{
   case 1:
       //Change LED
       numIn=isInt("Choose an LED to set: ",0,7);
```

```
        stateIn=isInt("Choose state for the LED: ",0,1);
        Write1Led(pBase, numIn, stateIn);

        break;

    case 2:
        //Read Switch

        numIn=isInt("Choose a Switch to read: ",0,7);
        stateOut=Read1Switch(pBase, numIn);
        cout<<"The state of Switch "<<numIn<<" is "<<stateOut<<endl;

        break;
```

Case 1 included the code for writing to a single LED. The user was asked for information by the function `isInt(prompt,min,max)` which blocks the program and asks the user for an integer value between its min and max values (appendix 1). When two valid inputs were given, they were sent to `Write1Led`. Case 2 asked the user which switch to output, using the same blocking input. The value at that switch was then returned by the prewritten code and printed.

Lab 3.2 Controlling all LEDs

The following function is used to write a user input number using 8 bit binary.

```
void WriteAllLeds(char *pBase, int numLeds)
{
    for (int i=7;i>=0;i--)
    {
        Write1Led(pBase, i, numLeds%2);
        numLeds/=2;
    }
}
```

The parameter `numLeds` is the integer value input by the user. To write to all of the LEDs, each one has to be written to in turn, which is done with a for loop. Knowledge of the behavior of binary numbers was used to manipulate the input value. The first LED written to is the 7th, last, LED, which is in the 2^0 place. The modulus used returns the remainder, which is 0 if the number is even and 1 if the number is odd. The value of `numLeds` is then floor divided by 2, effectively removing the digit just used. The next iteration of the loop checks if the halved value is divisible by 2, or if the original number was divisible by 4. This continues through all 8 LEDs. If the number 45 is selected, the iterations return 1,0,1,1,0,1,0,0 sequentially. When looking at the

LEDs, this appears as the binary number 00101101 when orienting so the 0th LED is on the left, which is the correct output.

Lab 3.3 Controlling all the switches

The following function is used to read the integer value represented by the sequence of binary switches.

```
int ReadAllSwitchs(char *pBase)
{
   int tempInt=0;
   for (int i=7;i>=0;i--)
   {
       tempInt+=Read1Switch(pBase,i)*(int)pow(2,7-i);
   }
   return tempInt;
}
```

The variable tempInt stores the element wise addition of the bits. Each LED represents a specific power of 2; if the switch is flipped on then that power of 2 is added to the value. Starting from the last switch, where i is 7, the power for the given switch is 2^{7-i}. This means that the last switch represents the 2^0 place. Each power of 2 is multiplied by the value at that switch, meaning that when the switch is off, that power of 2 is not added into the sum. When the binary sequence on the switches is 10010101, the first iteration of the code adds 1 because the digit furthest to the right is a one, then nothing is added for the second digit, and so on. The number returned by the program after all 8 digits are tried is 149, which is the correct conversion.

Conclusion

This lab gave an easy straight-forward introduction into I/O mapping and embedded design. This further allows us to expand our knowledge on the subject matter so we can move on to more advanced embedded designs. This can also be used further to improve our understanding of basic syntax and improving the speed of the program.

Appendix

Appendix 1: Code for function isInt()

```
int isInt(string prompt, int min, int max)
{
   int input=-1;
   while((input<min) || (input>max))
   {
       cout << prompt;
       cin >> input;
       if (cin.fail())
       {
           cin.clear();
           cin.ignore();
       }
       else break;
   }
   return input;
}
```