Lab Assignment 3

Lab Report

Matthew Springer

springer.m@husky.neu.edu

Submit date: 2/8/17

Due Date: 2/8/2017

**Abstract**

This lab was an introduction to programming memory-mapped Input/Ouput on the ZedBoard. Using a virtual memory map, we were able to read from switches and push buttons, as well as write to LEDs.

This is a lab report detailing the work done during the lab and the results we were able to obtain.

# Introduction

This laboratory session explored the basics of interaction with the ZedBoard. During the lab we demonstrated our ability to use Object-Oriented Programming in C++ to create a virtual memory map, on the ZedBoard and then read and write to virtual memory. This allowed us to turn the ZedBoard LEDs on and off, read the states of the ZedBoard switches, and determine which ZedBoard push buttons have been pressed. All of these lessons culminated in the final assignment, where we had to read an integer either from user input or represented in binary from the 8 ZedBoard switches, and then use the 8 ZedBoard LEDs to display a binary representation of the integer. The ZedBoard push buttons could also be used to manipulate the number: increasing it, decreasing it, bit-shifting it left or right, and resetting it to the value represented on the ZedBoard switches.

# Results and Analysis

*Assignment 1*

Sample Output:

user409@localhost:~/lab03$ Please enter an LED number between 0-7: 5

user409@localhost:~/lab03$ Please enter a state for LED #5 (0 or 1): 1

user409@localhost:~/lab03$ Please enter a switch number between 0-7: 4

user409@localhost:~/lab03$ The value of switch #4 is: 0

user409@localhost:~/lab03$ Please enter an LED number between 0-7: 2

user409@localhost:~/lab03$ Please enter a state for LED #5 (0 or 1): 1

user409@localhost:~/lab03$ Please enter a switch number between 0-7: 6

user409@localhost:~/lab03$ The value of switch #4 is: 1

user409@localhost:~/lab03$ Please enter an LED number between 0-7: 3

user409@localhost:~/lab03$ Please enter a state for LED #5 (0 or 1): 1

user409@localhost:~/lab03$ Please enter a switch number between 0-7: 0

user409@localhost:~/lab03$ The value of switch #4 is: 0

*Assignment 2*

/\*\* Changes the state of an LED (ON or OFF) to represent a given int in binary

\* @param pBase base address of I/O

\* @param numVal the number (0-255) to be represented in binary on the LEDs

\* @param state

State to change to (ON or OFF)

\*/

void WriteAllLeds(char \*pBase, int numVal) {

int remainder = numVal;

int length = 8;

int \* binaryRep = new int[length];

bool converting = true;

for (int i = length - 1; i >= 0; i--) {

if (converting) {

int place = pow(2, i);

if (remainder >= place) {

// cout << "Led #" << i << ": 1" << endl;

binaryRep[i] = 1;

remainder = remainder - place;

if (remainder == 0) {

converting = false;

}

}

else {

// cout << "Led #" << i << ": 0" << endl;

binaryRep[i] = 0;

}

}

else {

// cout << "Led #" << i << ": 0" << endl;

binaryRep[i] = 0;

}

}

for (int j = 0; j < length; j++) {

Write1Led(pBase, j, binaryRep[j]);

}

delete [] binaryRep;

}

In order to test the function WriteAllLeds, I tested the numbers 0 (all LEDs off) and 255 (all LEDs on), as well as the powers of 2 (each light individually). I then computed the binary representations of 135 and 46 in order to make sure that it handled multiple lights.

*Assignment 3*

/\*\* Reads all the switches and returns their value in a single integer.

\*

\* @param pBase Base address for general-purpose I/O

\* @return A value that represents the value of the switches

\*/

int ReadAllSwitches(char \*pBase) {

int sum = 0;

int length = 8;

bool converting = true;

for (int i = 0; i < length; i++) {

int place = pow(2, i);

sum += place \* Read1Switch(pBase, i);

}

return sum;

}

In order to test the function ReadAllSwitches, I first verified that all switches in the off position registered 0, and that all switches in the on position registered 255. I then checked each of the individual switches to make sure that they registered the corresponding value of 2.

*Assignment 4*

/\*

\* Project: EECE 2160 Lab #03 Part 2

\* Author: Matthew Springer

\* Date: February 1, 2017

\*/

#include <iostream>

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/mman.h>

#include <cmath>

using namespace std;

// 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

struct PushButtonConfig {

int left;

int right;

int up;

int down;

int center;

};

/\*\*

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

\*

\* @param pBase Base address returned by 'mmap'.

\* @param offset Offset where device is mapped.

\* @param value Value to be written.

\*/

void RegisterWrite(char \*pBase, int offset, int value)

{

\* (int \*) (pBase + offset) = value;

}

/\*\*

\* Read a 4-byte value from the specified general-purpose I/O location.

\*

\* @param pBase Base address returned by 'mmap'.

\* @param offset Offset where device is mapped.

\* @return Value read.

\*/

int RegisterRead(char \*pBase, int offset)

{

return \* (int \*) (pBase + offset);

}

/\*\*

\* Initialize general-purpose I/O

\* - Opens access to physical memory /dev/mem

\* - Maps memory at offset 'gpio\_address' into virtual address space

\*

\* @param fd File descriptor passed by reference, where the result

\* of function 'open' will be stored.

\* @return Address to virtual memory which is mapped to physical,

\* or MAP\_FAILED on error.

\*/

char \*Initialize(int \*fd)

{

\*fd = open( "/dev/mem", O\_RDWR);

return (char \*) mmap(NULL, gpio\_size, PROT\_READ | PROT\_WRITE, MAP\_SHARED,

\*fd, gpio\_address);

}

/\*\*

\* Close general-purpose I/O.

\*

\* @param pBase Virtual address where I/O was mapped.

\* @param fd File descriptor previously returned by 'open'.

\*/

void Finalize(char \*pBase, int fd)

{

munmap(pBase, gpio\_size);

close(fd);

}

/\*\* Changes the state of an LED (ON or OFF)

\* @param pBase base address of I/O

\* @param ledNum LED number (0 to 7)

\* @param state

State to change to (ON or OFF)

\*/

void Write1Led(char \*pBase, int ledNum, int state) {

int ledOffset = 0x12C + (ledNum \* 0x004);

RegisterWrite(pBase, ledOffset, state);

}

/\*\* Reads the value of a switch

\* - Uses base address of I/O

\* @param pBase base address of I/O

\* @param switchNum Switch number (0 to 7)

\* @return switch value read

\*/

int Read1Switch(char \*pBase, int switchNum) {

int switchOffset = 0x14C + (switchNum \* 0x004);

return RegisterRead(pBase, switchOffset);

}

/\*\* Changes the state of an LED (ON or OFF) to represent a given int in binary

\* @param pBase base address of I/O

\* @param numVal the number (0-255) to be represented in binary on the LEDs

\* @param state

State to change to (ON or OFF)

\*/

void WriteAllLeds(char \*pBase, int numVal) {

int remainder = numVal;

int length = 8;

int \* binaryRep = new int[length];

bool converting = true;

for (int i = length - 1; i >= 0; i--) {

if (converting) {

int place = pow(2, i);

if (remainder >= place) {

// cout << "Led #" << i << ": 1" << endl;

binaryRep[i] = 1;

remainder = remainder - place;

if (remainder == 0) {

converting = false;

}

}

else {

// cout << "Led #" << i << ": 0" << endl;

binaryRep[i] = 0;

}

}

else {

// cout << "Led #" << i << ": 0" << endl;

binaryRep[i] = 0;

}

}

for (int j = 0; j < length; j++) {

Write1Led(pBase, j, binaryRep[j]);

}

delete [] binaryRep;

}

/\*\* Reads all the switches and returns their value in a single integer.

\*

\* @param pBase Base address for general-purpose I/O

\* @return A value that represents the value of the switches

\*/

int ReadAllSwitches(char \*pBase) {

int sum = 0;

int length = 8;

bool converting = true;

for (int i = 0; i < length; i++) {

int place = pow(2, i);

sum += place \* Read1Switch(pBase, i);

}

cout << "The integer representation of the current switch configuration is: " << sum << endl;

return sum;

}

/\*\* Reads the value of a push button

\* - Uses base address of I/O

\* @param pBase base address of I/O

\* @param buttonNum push button number (0 to 4)

\* 0 - left

\* 1 - right

\* 2 - up

\* 3 - down

\* 4 - center

\* @return button value read

\*/

int Read1Button(char \*pBase, int buttonNum) {

int buttonOffset = 0x16C + (buttonNum \* 0x004);

return RegisterRead(pBase, buttonOffset);

}

/\*\* Reads the current button configuration and modifies the given counter if any are pressed.

\*

\* @param pBase Base address for general-purpose I/O

\* @param config the current button state configuration

\* @param counter the integer to adjust

\*/

void PushButtonGet(char \*pBase, PushButtonConfig \* config, int \*counter) {

int left, right, up, down, center, currentNum;

left = Read1Button(pBase, 0);

right = Read1Button(pBase, 1);

up = Read1Button(pBase, 2);

down = Read1Button(pBase, 3);

center = Read1Button(pBase, 4);

currentNum = \*counter;

if (config->left == 0 && left == 1) {

\*counter = min(255, currentNum << 1);

cout << "The current value of the counter is: " << \*counter << endl;

}

config->left = left;

if (config->right == 0 && right == 1) {

\*counter = max(0, currentNum >> 1);

cout << "The current value of the counter is: " << \*counter << endl;

}

config->right = right;

if (config->up == 0 && up == 1) {

\*counter = min(255, currentNum + 1);

cout << "The current value of the counter is: " << \*counter << endl;

}

config->up = up;

if (config->down == 0 && down == 1) {

\*counter = max(0, currentNum - 1);

cout << "The current value of the counter is: " << \*counter << endl;

}

config->down = down;

if (config->center == 0 && center == 1) {

\*counter = ReadAllSwitches(pBase);

cout << "The current value of the counter is: " << \*counter << endl;

}

config->center = center;

}

/\*\*

\* Main function to interact with I/O Interfaces

\*/

int main()

{

// Initialize

int fd;

char \*pBase = Initialize(&fd);

// Check error

if (pBase == MAP\_FAILED)

{

cerr << "Mapping I/O memory failed - Did you run with 'sudo'?\n";

exit(1); // Returns 1 to the operating system;

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\* Put your code here \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int \* counter = new int;

\*counter = ReadAllSwitches(pBase);

PushButtonConfig \* pbConfig = new PushButtonConfig;

pbConfig->left = 0;

pbConfig->right = 0;

pbConfig->up = 0;

pbConfig->down = 0;

pbConfig->center = 0;

cout << "The current value of the counter is: " << \*counter << endl;

while (true) {

PushButtonGet(pBase, pbConfig, counter);

WriteAllLeds(pBase, \*counter);

}

Finalize(pBase, fd);

return 0;

}

Sample Output:

user409@localhost:~/lab03$ g++ PushButton.cpp -o PushButton

user409@localhost:~/lab03$ sudo ./PushButton

user409@localhost's password:

The current value of the counter is: 32

The current value of the counter is: 31

The current value of the counter is: 30

The current value of the counter is: 29

The current value of the counter is: 28

The current value of the counter is: 14

The current value of the counter is: 7

The current value of the counter is: 8

The current value of the counter is: 9

The current value of the counter is: 10

The current value of the counter is: 11

The current value of the counter is: 12

The current value of the counter is: 13

The current value of the counter is: 26

The current value of the counter is: 52

The current value of the counter is: 104

The current value of the counter is: 52

The current value of the counter is: 26

The current value of the counter is: 32

^C

user409@localhost:~/lab03$

*Assignment 5*

/\*

\* Project: EECE 2160 Lab #03 Part 3

\* Author: Matthew Springer

\* Date: February 1, 2017

\*/

#include <iostream>

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/mman.h>

#include <cmath>

using namespace std;

// 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

struct PushButtonConfig {

int left;

int right;

int up;

int down;

int center;

};

class ZedBoard

{

int fd;

char \*pBase;

int \* counter;

PushButtonConfig \*pbConfig;

private:

/\*\*

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

\*

\* @param pBase Base address returned by 'mmap'.

\* @param offset Offset where device is mapped.

\* @param value Value to be written.

\*/

void registerWrite(int offset, int value)

{

\* (int \*) (this->pBase + offset) = value;

}

/\*\*

\* Read a 4-byte value from the specified general-purpose I/O location.

\*

\* @param pBase Base address returned by 'mmap'.

\* @param offset Offset where device is mapped.

\* @return Value read.

\*/

int registerRead(int offset)

{

return \* (int \*) (this->pBase + offset);

}

/\*\*

\* Initialize general-purpose I/O

\* - Opens access to physical memory /dev/mem

\* - Maps memory at offset 'gpio\_address' into virtual address space

\*

\* @param fd File descriptor passed by reference, where the result

\* of function 'open' will be stored.

\* @return Address to virtual memory which is mapped to physical,

\* or MAP\_FAILED on error.

\*/

char \* initialize()

{

this->fd = open( "/dev/mem", O\_RDWR);

return (char \*) mmap(NULL, gpio\_size, PROT\_READ | PROT\_WRITE, MAP\_SHARED,

this->fd, gpio\_address);

}

/\*\*

\* Close general-purpose I/O.

\*

\* @param pBase Virtual address where I/O was mapped.

\* @param fd File descriptor previously returned by 'open'.

\*/

void finalize()

{

munmap(this->pBase, gpio\_size);

close(this->fd);

}

/\*\* Changes the state of an LED (ON or OFF)

\* @param pBase base address of I/O

\* @param ledNum LED number (0 to 7)

\* @param state

State to change to (ON or OFF)

\*/

void write1Led(int ledNum, int state) {

int ledOffset = 0x12C + (ledNum \* 0x004);

this->registerWrite(ledOffset, state);

}

/\*\* Reads the value of a switch

\* - Uses base address of I/O

\* @param pBase base address of I/O

\* @param switchNum Switch number (0 to 7)

\* @return switch value read

\*/

int read1Switch(int switchNum) {

int switchOffset = 0x14C + (switchNum \* 0x004);

return this->registerRead(switchOffset);

}

/\*\* Reads all the switches and returns their value in a single integer.

\*

\* @param pBase Base address for general-purpose I/O

\* @return A value that represents the value of the switches

\*/

int readAllSwitches() {

int sum = 0;

int length = 8;

bool converting = true;

for (int i = 0; i < length; i++) {

int place = pow(2, i);

sum += place \* this->read1Switch(i);

}

cout << "The integer representation of the current switch configuration is: " << sum << endl;

return sum;

}

/\*\* Reads the value of a push button

\* - Uses base address of I/O

\* @param pBase base address of I/O

\* @param buttonNum push button number (0 to 4)

\* 0 - left

\* 1 - right

\* 2 - up

\* 3 - down

\* 4 - center

\* @return button value read

\*/

int read1Button(int buttonNum) {

int buttonOffset = 0x16C + (buttonNum \* 0x004);

return this->registerRead(buttonOffset);

}

public:

/\*\* Changes the state of an LED (ON or OFF) to represent a given int in binary

\* @param pBase base address of I/O

\* @param numVal the number (0-255) to be represented in binary on the LEDs

\* @param state

State to change to (ON or OFF)

\*/

void writeAllLeds(int numVal) {

int remainder = numVal;

int length = 8;

int \* binaryRep = new int[length];

bool converting = true;

for (int i = length - 1; i >= 0; i--) {

if (converting) {

int place = pow(2, i);

if (remainder >= place) {

// cout << "Led #" << i << ": 1" << endl;

binaryRep[i] = 1;

remainder = remainder - place;

if (remainder == 0) {

converting = false;

}

}

else {

// cout << "Led #" << i << ": 0" << endl;

binaryRep[i] = 0;

}

}

else {

// cout << "Led #" << i << ": 0" << endl;

binaryRep[i] = 0;

}

}

for (int j = 0; j < length; j++) {

this->write1Led(j, binaryRep[j]);

}

delete [] binaryRep;

}

/\*\* Reads the current button configuration and modifies the given counter if any are pressed.

\*

\* @param pBase Base address for general-purpose I/O

\* @param config the current button state configuration

\* @param counter the integer to adjust

\*/

void pushButtonGet() {

int left, right, up, down, center, currentNum;

left = this->read1Button(0);

right = this->read1Button(1);

up = this->read1Button(2);

down = this->read1Button(3);

center = this->read1Button(4);

currentNum = \*this->counter;

if (this->pbConfig->left == 0 && left == 1) {

\*this->counter = min(255, currentNum << 1);

cout << "The current value of the counter is: " << \*counter << endl;

}

this->pbConfig->left = left;

if (this->pbConfig->right == 0 && right == 1) {

\*this->counter = max(0, currentNum >> 1);

cout << "The current value of the counter is: " << \*counter << endl;

}

this->pbConfig->right = right;

if (this->pbConfig->up == 0 && up == 1) {

\*this->counter = min(255, currentNum + 1);

cout << "The current value of the counter is: " << \*counter << endl;

}

this->pbConfig->up = up;

if (this->pbConfig->down == 0 && down == 1) {

\*this->counter = max(0, currentNum - 1);

cout << "The current value of the counter is: " << \*counter << endl;

}

this->pbConfig->down = down;

if (this->pbConfig->center == 0 && center == 1) {

\*this->counter = this->readAllSwitches();

cout << "The current value of the counter is: " << \*counter << endl;

}

this->pbConfig->center = center;

}

int getCounter() {

return \*this->counter;

}

ZedBoard() {

// Initialize

this->pBase = this->initialize();

// Check error

if (this->pBase == MAP\_FAILED)

{

cerr << "Mapping I/O memory failed - Did you run with 'sudo'?\n";

exit(1); // Returns 1 to the operating system;

}

this->counter = new int;

\*this->counter = this->readAllSwitches();

this->pbConfig = new PushButtonConfig;

this->pbConfig->left = 0;

this->pbConfig->right = 0;

this->pbConfig->up = 0;

this->pbConfig->down = 0;

this->pbConfig->center = 0;

}

~ZedBoard() {

this->finalize();

delete this->pBase;

delete this->counter;

delete this->pbConfig;

}

};

/\*\*

\* Main function to interact with I/O Interfaces

\*/

int main()

{

ZedBoard \* zb = new ZedBoard();

cout << "The current value of the counter is: " << zb->getCounter() << endl;

while (true) {

zb->pushButtonGet();

zb->writeAllLeds(zb->getCounter());

}

delete zb;

return 0;

}

Sample Output:

user409@localhost:~/lab03$ g++ PushButtonClass.cpp -o PushButtonClass

user409@localhost:~/lab03$ sudo ./PushButtonClass

user409@localhost's password:

The current value of the counter is: 0

The current value of the counter is: 1

The current value of the counter is: 2

The current value of the counter is: 3

The current value of the counter is: 6

The current value of the counter is: 12

The current value of the counter is: 24

The current value of the counter is: 23

The current value of the counter is: 22

The current value of the counter is: 21

The current value of the counter is: 20

The current value of the counter is: 10

The current value of the counter is: 11

The current value of the counter is: 12

The current value of the counter is: 0

^C

user409@localhost:~/lab03$

# Conclusion

In conclusion, I found this to be a very enjoyable lab. I encountered some difficulties due to not having a lab partner, but none that I was not able to quickly resolve either with a Google search or by asking Prof. Kimani or one of the TAs. It was a very interesting introduction to Input/Output on the ZedBoard, and I look forward to building my skills further in future labs.

# References

None

1. Appendix

Use an appendix to present additional information such source code snippets, code organization or test run outputs.