2160 Lab 4

Patrick Taylor and James Wachala

Section Number: 1

February 13th, 2019

Pre-lab

a

It fails at line 98, as that is where it catches if the inizialization fails

b

Initialize

This function's job is to 'set things up' for the board. I essentially sets up addresses for things to happen, using nmap to keep things spaced right for the size of the gpio (see man mman)

Finalize

It undoes initialise. It unmaps what mmap did, and also closes the file where this stuff was being written/read.

RegisterRead

Reads value of a register. It takes in the base memory address and the actual register's offset from that, and returns the content of the register. This is essentially reading a signal or state.

RegisterWrite

Functions like RegisterRead, but instead of returning a value it sets taht value to that register. Used to turn on a light, for instance.

1 LedOnOff

We tested several positions on and off.

```
// LedOnOff.cc
#include <fcntl.h>

#include <iostream>

#include <stdio.h>

#include <sys/mman.h>

#include <unistd.h>
```

```
// 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

/**

 * Write a 4-byte value at the specified general-purpose I/O location.
 *
 * @param ptr Base address returned by 'mmap'.
 * @parem offset Offset where device is mapped.
 * @param value Value to be written.
 */
void RegisterWrite(char *ptr, int offset, int value) {
    *(int *)(ptr + offset) = value;
}
/**
 * Read a 4-byte value from the specified general-purpose I/O location.
 *
 * @param ptr Base address returned by 'mmap'.
 * @param offset Offset where device is mapped.
 * @return Value read.
 */
int RegisterRead(char *ptr, int offset) { return *(int *)(ptr + 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 ptr Virtual address where I/O was mapped.

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

 */

void Finalize(char *ptr, int fd) {

    munmap(ptr, gpio_size);

    close(fd);

}


 /** Set the state of the LED with the given index.
 *
 * @param ptr Base address for general-purpose I/O
 * @parem led_index LED index between 0 and 7
 * @param state Turn on (1) or off (0)
 */
void SetLedState(char *ptr, int led_index, int state) {
     RegisterWrite(ptr, gpio_led1_offset+4*led_index, state);


}
/**

 * Show lower 8 bits of integer value on LEDs

 *

 * @param ptr Base address of I/O
 * @param value Value to show on LEDs

 */
void SetLedNumber(char *ptr, int value) {

    RegisterWrite(ptr, gpio_led1_offset, value % 2);

    RegisterWrite(ptr, gpio_led2_offset, (value / 2) % 2);

    RegisterWrite(ptr, gpio_led3_offset, (value / 4) % 2);

    RegisterWrite(ptr, gpio_led4_offset, (value / 8) % 2);

    RegisterWrite(ptr, gpio_led5_offset, (value / 16) % 2);

    RegisterWrite(ptr, gpio_led6_offset, (value / 32) % 2);

    RegisterWrite(ptr, gpio_led7_offset, (value / 64) % 2);

    RegisterWrite(ptr, gpio_led8_offset, (value / 128) % 2);

}

int main() {

    // Initialize

    int fd;

    char *ptr = Initialize(&fd);

    // Check error

    if (ptr == MAP_FAILED) {

        perror("Mapping I/O memory failed - Did you run with 'sudo'?\n");

        return -1;

    }

    int placeValue;

    std::cout << "Enter a value between 0 and 7: ";

    std::cin >> placeValue;

    int state;

    std::cout << "Enter a value 0 (off) or 1 (on): ";

    std::cin >> state;
    // Show the value on the Zedboard LEDs

    SetLedState(ptr, placeValue,state);

    // Done

    Finalize(ptr, fd);

    return 0;

}
```

2

```
#include <fcntl.h>

#include <iostream>

#include <stdio.h>

#include <sys/mman.h>

#include <unistd.h>

// 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

/**

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

 *

 * @param ptr Base address returned by 'mmap'.

 * @parem offset Offset where device is mapped.

 * @param value Value to be written.

 */

void RegisterWrite(char *ptr, int offset, int value) {

    *(int *)(ptr + offset) = value;

}

/**

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

 *

 * @param ptr Base address returned by 'mmap'.

 * @param offset Offset where device is mapped.
 * @return Value read.
 */
int RegisterRead(char *ptr, int offset) { return *(int *)(ptr + 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 ptr Virtual address where I/O was mapped.
 * @param fd File descriptor previously returned by 'open'.
 */
```

```
void Finalize(char *ptr, int fd) {

    munmap(ptr, gpio_size);

    close(fd);

}


 /** Set the state of the LED with the given index.
 *
 * @param ptr Base address for general-purpose I/O
 * @parem led_index LED index between 0 and 7
 * @param state Turn on (1) or off (0)
 */
void SetLedState(char *ptr, int led_index, int state) {
     RegisterWrite(ptr, gpio_led1_offset+4*led_index, state);


}
/**

 * Show lower 8 bits of integer value on LEDs

 *

 * @param ptr Base address of I/O

 * @param value Value to show on LEDs

 */
void SetLedNumber(char *ptr, int value) {

    RegisterWrite(ptr, gpio_led1_offset, value % 2);

    RegisterWrite(ptr, gpio_led2_offset, (value / 2) % 2);

    RegisterWrite(ptr, gpio_led3_offset, (value / 4) % 2);

    RegisterWrite(ptr, gpio_led4_offset, (value / 8) % 2);

    RegisterWrite(ptr, gpio_led5_offset, (value / 16) % 2);

    RegisterWrite(ptr, gpio_led6_offset, (value / 32) % 2);

    RegisterWrite(ptr, gpio_led7_offset, (value / 64) % 2);

    RegisterWrite(ptr, gpio_led8_offset, (value / 128) % 2);

}

int main() {

    // Initialize
    int fd;

    char *ptr = Initialize(&fd);

    // Check error

    if (ptr == MAP_FAILED) {

        perror("Mapping I/O memory failed - Did you run with 'sudo'?\n");

        return -1;

    }
    bool running = true;
    while(running){
        for(int i = 0; i < 8; i++){
                int val = RegisterRead(ptr, gpio_sw1_offset+4*i);
                 SetLedState(ptr, i,val);
        }

    }
    /*int placeValue;

    std::cout << "Enter a value between 0 and 7: ";

    std::cin >> placeValue;

    int state;

    std::cout << "Enter a value 0 (off) or 1 (on): ";

    std::cin >> state;

    // Show the value on the Zedboard LEDs

    SetLedState(ptr, placeValue,state);
*/
    // Done

    Finalize(ptr, fd);

    return 0;

}
```

3

```
#include <fcntl.h>

#include <iostream>
```

```
#include <stdio.h>

#include <sys/mman.h>

#include <unistd.h>

#include <cmath>

#include <unistd.h>

// 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

/**

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

 *

 * @param ptr Base address returned by 'mmap'.

 * @parem offset Offset where device is mapped.

 * @param value Value to be written.

 */

void RegisterWrite(char *ptr, int offset, int value) {

    *(int *)(ptr + offset) = value;

}

/**

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

 *

 * @param ptr Base address returned by 'mmap'.

 * @param offset Offset where device is mapped.

 * @return Value read.

 */

int RegisterRead(char *ptr, int offset) {
        return *(int *)(ptr + 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 ptr Virtual address where I/O was mapped.
 * @param fd File descriptor previously returned by 'open'.
 */
void Finalize(char *ptr, int fd) {
    munmap(ptr, gpio_size);
    close(fd);
}
/**
 * Show lower 8 bits of integer value on LEDs
 *
 * @param ptr Base address of I/O
 * @param value Value to show on LEDs

 */

void SetLedNumber(char *ptr, int value) {

    RegisterWrite(ptr, gpio_led1_offset, value % 2);

    RegisterWrite(ptr, gpio_led2_offset, (value / 2) % 2);

    RegisterWrite(ptr, gpio_led3_offset, (value / 4) % 2);

    RegisterWrite(ptr, gpio_led4_offset, (value / 8) % 2);

    RegisterWrite(ptr, gpio_led5_offset, (value / 16) % 2);

    RegisterWrite(ptr, gpio_led6_offset, (value / 32) % 2);

    RegisterWrite(ptr, gpio_led7_offset, (value / 64) % 2);

    RegisterWrite(ptr, gpio_led8_offset, (value / 128) % 2);

}
/**
*
*/
int PushButtonGet(char *ptr){
    //0 = none pressed
    //1 = left
    //2 = right
    //3 = up
    //4 = down
    //5 = center
    for(int i = 0; i < 5; i++){
        int val = RegisterRead(ptr,gpio_pbtnl_offset+4*i);
        if(val ==1){
            return i+1;
        }
    }
    return 0;
}

int main() {

    // Initialize

    int fd;

    char *ptr = Initialize(&fd);

    // Check error

    if (ptr == MAP_FAILED) {
        perror("Mapping I/O memory failed - Did you run with 'sudo'?\n");

        return -1;

    }
    int count = 0;
    int lastState = 7;
    while(true){
        usleep(1000);
        if(count > 255){
                 count = 255;
        }
        if(count < 0){
                 count = 0;
        }
        SetLedNumber(ptr, count);
        int state = PushButtonGet(ptr);
        if(lastState == 7){
            state = 5;
            }
        if(state!=lastState){

        switch(state){
            case 1:{
                if(count>127){
                count-=128;
                }
                count*=2;
                break;
            }
            case 2:{
                count/=2;
                break;
            }
            case 3:{
                 count++;
                 break;
            }
            case 4:{
                count--;
                break;
            }
            case 5:{
            count = 0;
                 for(int i = 0; i < 8; i++){
                     int val = RegisterRead(ptr, gpio_sw1_offset+4*i);
                     count+= val*std::pow(2,i);
                 }
                 break;
            }
        }
        lastState = state;
        }
    }
    /*int value;

    std::cout << "Enter a value less than 256: ";

    std::cin >> value;

    // Show the value on the Zedboard LEDs

    SetLedNumber(ptr, value);

    // Done

    Finalize(ptr, fd);

    return 0;

}
```

4

```

#include "ZedBoard.h"

// 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*/

/**
 * Show lower 8 bits of integer value on LEDs
 *
 * @param ptr Base address of I/O
 * @param value Value to show on LEDs
 */
void SetLedNumber(int value,ZedBoard &board) {
    board.RegisterWrite( gpio_led1_offset, value % 2);
    board.RegisterWrite( gpio_led2_offset, (value / 2) % 2);
    board.RegisterWrite( gpio_led3_offset, (value / 4) % 2);
    board.RegisterWrite( gpio_led4_offset, (value / 8) % 2);
    board.RegisterWrite( gpio_led5_offset, (value / 16) % 2);
    board.RegisterWrite( gpio_led6_offset, (value / 32) % 2);
    board.RegisterWrite( gpio_led7_offset, (value / 64) % 2);
    board.RegisterWrite( gpio_led8_offset, (value / 128) % 2);
```

```
}
/**
*
*/
int PushButtonGet(ZedBoard &board){
    //0 = none pressed
    //1 = left
    //2 = right
    //3 = up
    //4 = down
    //5 = center
    for(int i = 0; i < 5; i++){
        int val = board.RegisterRead(gpio_pbtnl_offset+4*i);
        if(val ==1){
            return i+1;
        }

    }
    return 0;
}

int main() {

    // Initialize

    //int fd;
    ZedBoard board;
    //char *ptr = Initialize(&fd);

    // Check error

    /*if (ptr == MAP_FAILED) {

        perror("Mapping I/O memory failed - Did you run with 'sudo'?\n");

        return -1;

    }*/
    int count = 0;
    int lastState = 7;
    while(true){
        usleep(1000);
        if(count > 255){
                 count = 255;
        }
        if(count < 0){
                 count = 0;
        }
        SetLedNumber(count, board);
        int state = PushButtonGet(board);
        if(lastState == 7){
            state = 5;
            }
        if(state!=lastState){
```

```
        switch(state){
            case 1:{
                if(count>127){
                count-=128;
                }
                count*=2;
                break;
            }
            case 2:{
                count/=2;
                break;
            }
            case 3:{
                 count++;
                 break;
            }
            case 4:{
                count--;
                break;
            }
            case 5:{
            count = 0;
                 for(int i = 0; i < 8; i++){
                     int val = board.RegisterRead( gpio_sw1_offset+4*i);
                     count+= val*std::pow(2,i);
                 }
                 break;
            }
        }
        lastState = state;
        }
    }
    /*int value;

    std::cout << "Enter a value less than 256: ";

    std::cin >> value;

    // Show the value on the Zedboard LEDs

    SetLedNumber( value);
*/
    // Done

    //Finalize( fd);

    return 0;

}
```

Extra Credit

```
#include "ZedBoard.h"
void SetLedNumber(int value,ZedBoard &board) {

    board.RegisterWrite( gpio_led1_offset, value % 2);

    board.RegisterWrite( gpio_led2_offset, (value / 2) % 2);

    board.RegisterWrite( gpio_led3_offset, (value / 4) % 2);

    board.RegisterWrite( gpio_led4_offset, (value / 8) % 2);

    board.RegisterWrite( gpio_led5_offset, (value / 16) % 2);

    board.RegisterWrite( gpio_led6_offset, (value / 32) % 2);

    board.RegisterWrite( gpio_led7_offset, (value / 64) % 2);

    board.RegisterWrite( gpio_led8_offset, (value / 128) % 2);

}
/**
*
*/
int PushButtonGet(ZedBoard &board){
    //0 = none pressed
    //1 = left
    //2 = right
    //3 = up
    //4 = down
    //5 = center
    for(int i = 0; i < 5; i++){
        int val = board.RegisterRead(gpio_pbtnl_offset+4*i);
        if(val ==1){
            return i+1;
        }

    }
    return 0;
}

int main() {

    // Initialize

    //int fd;
    ZedBoard board;
    //char *ptr = Initialize(&fd);

    // Check error
    /*if (ptr == MAP_FAILED) {

        perror("Mapping I/O memory failed - Did you run with 'sudo'?\n");

        return -1;

    }*/
    int count = 0;
    int timeCount = 0;
    int lastState = 7;
    int mult = 0;
    while(true){
        usleep(1000);
        if(mult > 0 || mult < 0){
          timeCount++;
          if(timeCount%(1000/abs(mult)) == 0){
          if(mult >0){
            count++;
          }else{
              count--;
          }
            SetLedNumber(count, board);
          }
        }


        if(count > 255){
                 count = 255;
        }
        if(count < 0){
                 count = 0;
        }
        SetLedNumber(count, board);
        int state = PushButtonGet(board);
        if(lastState == 7){
            state = 5;
            }
        if(state!=lastState){

        switch(state){
            case 1:{

                if(mult > 0){
                    mult*=-1;
                }
                break;
            }
            case 2:{
                if(mult < 0){
                    mult*=-1;
                }
                break;
            }
            case 3:{
                 mult++;
                 break;
            }
            case 4:{
                mult--;
                break;
            }
            case 5:{
            count = 0;
                 for(int i = 0; i < 8; i++){
                     int val = board.RegisterRead( gpio_sw1_offset+4*i);
                     count+= val*std::pow(2,i);
                 }
                 break;
            }
        }
        lastState = state;
        }
    }
    /*int value;

    std::cout << "Enter a value less than 256: ";

    std::cin >> value;

    // Show the value on the Zedboard LEDs

    SetLedNumber( value);
*/
    // Done

    //Finalize( fd);

    return 0;

}
```