Lab Assignment 3
Memory-Mapped I/O and Object Oriented Programming

Saul Reyna, Yuheng Dong
reyna.s@husky.neu.edu
dong.yuh@husky.neu.edu

Submit date: 2/12/2020
Due Date: 2/12/2020

Abstract
The lab consisted of accessing the ZedBoard's I/O such as the lights, buttons and switches. Throughout the lab, the I/O is controlled or examined independently, such as controlling only the lights or only reading the position of the switches. After all the characteristics of the board I/O is observed, these parts are made to interact with one another, such as using buttons and switches to dictate which lights remain on and off.

Introduction

Using memory mapped I/O in the ZedBoard, the user is able to read or write the state of a certain I/O at a given address. Eventually, the interaction with the board's I/O is abstracted to classes, that allows different functions being called on a ZedBoard object, rather than having to pass around the memory address of the I/O around through functions.

Lab Discussion

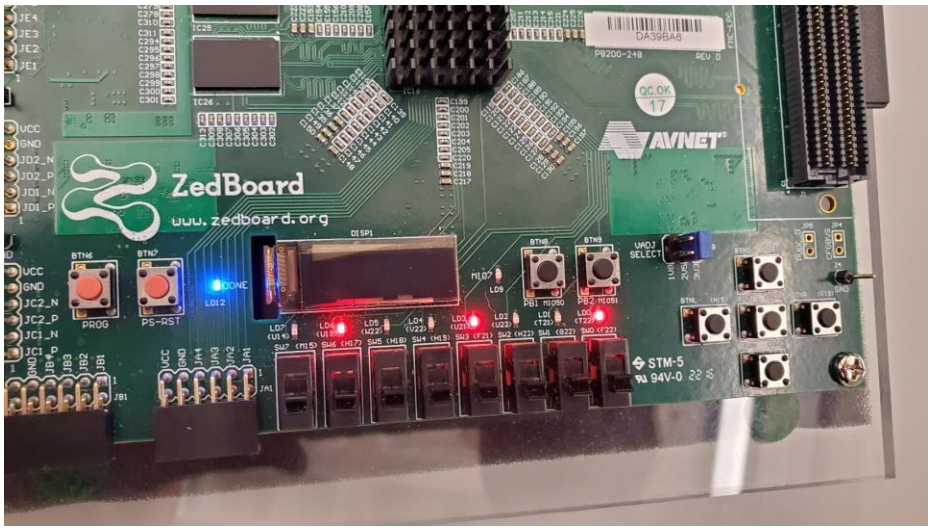
Equipment used in this lab include

- Zedboard (ZYNQ SoC)
    - Dual ARM Cortex – A9 MPCore with CoreSight
    - Zynq-7000 AP SoC XC7Z020-1CLG484
- Xillinux OS
- CoE lab desktop

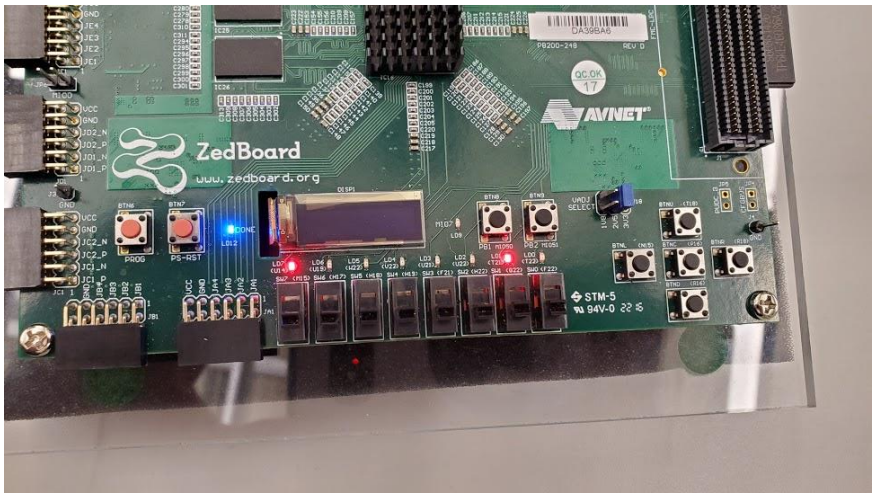
Results and Analysis

Assignment 3.1

```
user418@localhost:~/lab3$ sudo ./ledNumber
[sudo] password for user418:
LED 0, 3, 6 on
```



```
user418@localhost:~/lab3$ sudo ./ledNumber
[sudo] password for user418:
LED 1, 7 on
user418@localhost:~/lab3$
```

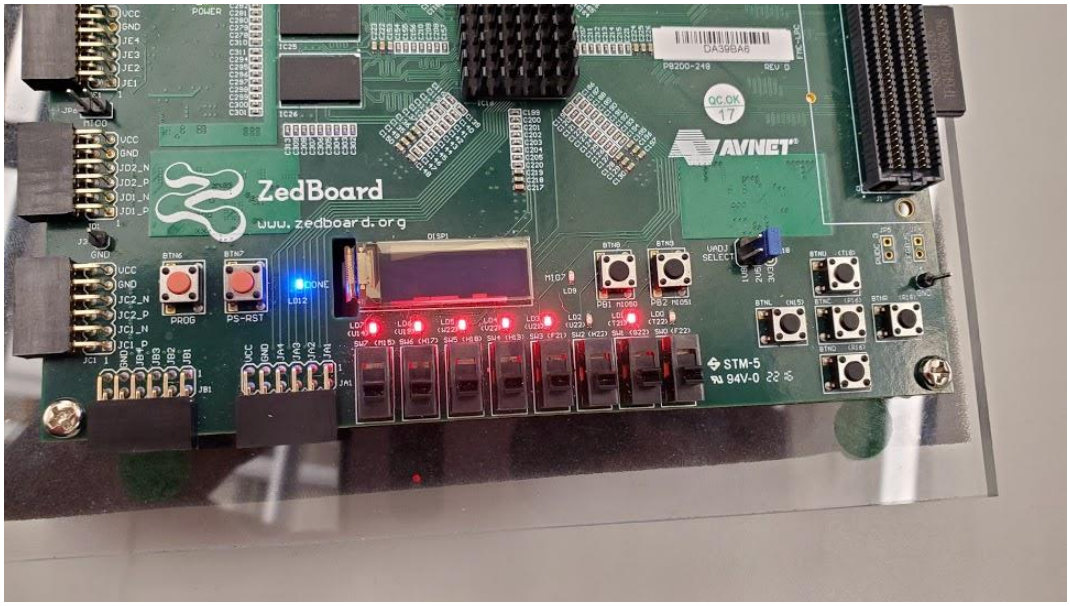



```
user418@localhost:~/lab3$ sudo ./ledNumber
[sudo] password for user418:
Switch 3: 1
Switch 7: 1
Switch 0: 1
```

Assignment 3.2

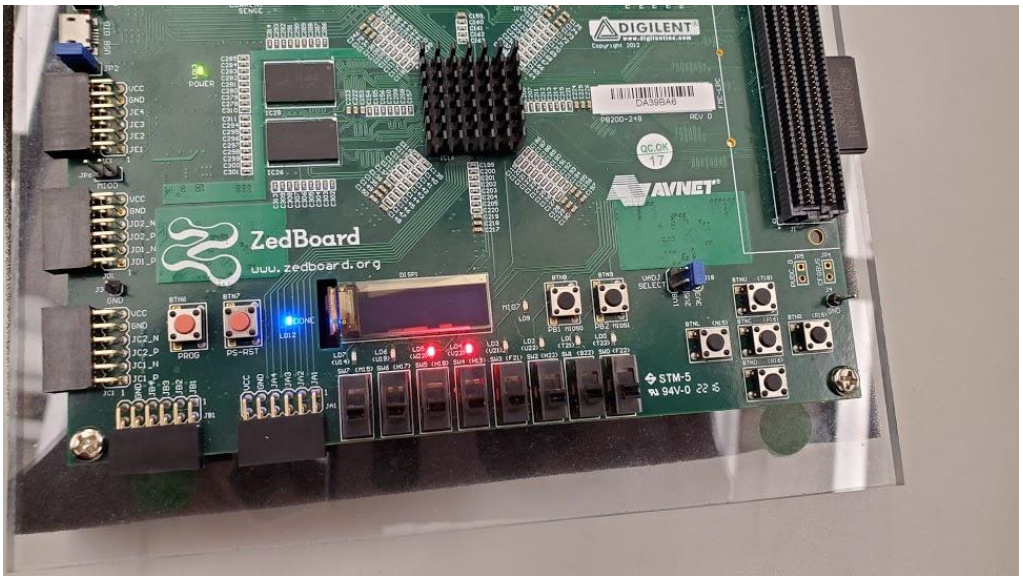
```
user418@localhost:~/lab3$ sudo ./ledNumber
[sudo] password for user418:
Enter a value from 0 to 255
255
Enter a value from 0 to 255
100
Enter a value from 0 to 255
48
Enter a value from 0 to 255
250
Enter a value from 0 to 255
```

The input values are the values that determine whether a light is on or off. Each value gets converted to its associated binary number and if the digit of the binary number is a one, it turns the light on, otherwise, it sets the value of the light to zero. For example, the binary for 255 is 11111111, a total of eight ones. This would result in a light turning on. In the other hand, 0, would turn all the lights off, since all the digits of the binary number of 0 would be 0.

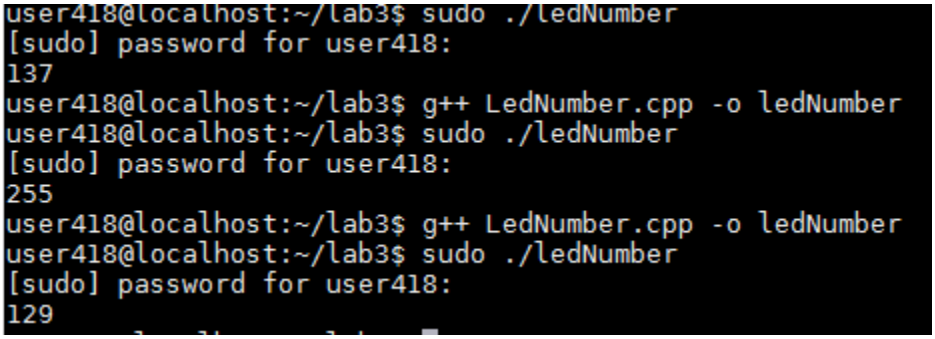


The image above is from when the input was the decimal number 250, which binary number is 11111010, meaning that LEDs 1, 3, 4, 5, 6, and 7 should be turned on.

Similarly, below, the board lit up LEDs 4 and 5 when the input was 48, whose binary number is 00110000.



Assignment 3.3



What ReadAllSwitches would do was convert the state of the switches (turned up or down) and convert it to a binary number. If a switch was turned up, a 1 would be in the number digit of said switch. If switch is turned down, it would be a 0. There were 3 main cases used to test ReadAllSwitches. The first case was with all the switches turned off, which returned 00000000 as intended. The next case was with all the switches turned on, which returned 11111111 as intended. The final case was with varying switches on and off. All combinations of on and off switches tested returned the correct bitset.

Assignment 3.4

Attached video.

Assignment 3.5

Code for this and all previous assignments are in the appendix, in their respective order.

Conclusion

Using memory mapping and the physical address of the input/output of the board, one can read and write values to any address to within the mapped window, where the addresses represent the LEDs, switches, and buttons. To reach those addresses, an offset address is added to the base address of the I/O so that the program knows where the given I/O is. Using this concept, the different I/O can now interact with one another. In this lab, some of these interactions involved turning the LEDs on depending if the switch was turned up or down. Additionally, a menu of buttons controlled the lights by manipulating a binary number representing the state of all the LEDs(1 for on and 0 for off).

References

[1] Michael Benjamin, "*Lab Report Guide*", Northeastern University, January 18 2006.

Appendix

LedNumber

```
#include <iostream>
#include <cstdlib>
#include <fcntl.h>
#include <unistd.h>
#include <sys/mman.h>
#include <bitset>
#include <string>

using namespace std;

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

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

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 pBase        Base address returned by 'mmap'.
 * @parem offset       Offset where device is mapped.
* @ return Switch value read

* @param value         Value to be written.
*
* @param switchNum Switch number (0 to 7)
*/
void RegisterWrite(char *pBase, int offset, int value)
```

```
{
        * (int *) (pBase + offset) = value;
}/** Reads the value of a switch
* - Users base address of I/O
* @param pBase base address of I/O


/**
 * 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 pBaseVirtual 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) {
        if (0 >= ledNum > 7) {
                cerr << "Trying to write to a non-existing LED" << endl;
        }
        int ledOffset = 0x12C + (ledNum * 4);
        RegisterWrite(pBase, ledOffset, state);
}

/** Reads the value of a switch
* - Users 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) {
        if (0 > switchNum && switchNum > 7)
        {
                cerr << "Trying to read a non-existing switch" << endl;
        }
        int switchOffset = 0x14C + (switchNum * 4);
        RegisterRead(pBase, switchOffset);
}

void WriteAllLeds(char *pBase, int value) {
     bitset<8> bits(value);
     for(int i = 0; i < 8; i++) {
             Write1Led(pBase, i, bits[i]);
     }
}

int ReadAllSwitch(char *pBase) {
        bitset<8> bits;
        for (int i = 0; i < 8; i++) {
                bits.set(i, Read1Switch(pBase, i));
        }

        int switch_list= (int)(bits.to_ulong());
        return switch_list;
}

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);
        }

// ************** Put your code here **********************
```

```
  cout << ReadAllSwitch(pBase) << endl;

        // Done
        Finalize(pBase, fd);
        return 0;
}
```

PushButton

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

using namespace std;

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

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

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 pBase        Base address returned by 'mmap'.
 * @parem offset       Offset where device is mapped.
```

```
* @ return Switch value read

* @param value         Value to be written.
*
* @param switchNum Switch number (0 to 7)
*/
void RegisterWrite(char *pBase, int offset, int value)
{
        * (int *) (pBase + offset) = value;
}/** Reads the value of a switch
* - Users base address of I/O
* @param pBase base address of I/O


/**
 * 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 pBaseVirtual 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) {
        if (0 >= ledNum > 7) {
                cerr << "Trying to write to a non-existing LED" << endl;
        }
        int ledOffset = 0x12C + (ledNum * 4);
        RegisterWrite(pBase, ledOffset, state);
}

/** Reads the value of a switch
* - Users 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) {
        if (0 > switchNum && switchNum > 7)
        {
                cerr << "Trying to read a non-existing switch" << endl;
        }
        int switchOffset = 0x14C + (switchNum * 4);
        RegisterRead(pBase, switchOffset);
}

void WriteAllLeds(char *pBase, int value) {
     bitset<8> bits(value);
     for(int i = 0; i < 8; i++) {
             Write1Led(pBase, i, bits[i]);
     }
}

int ReadAllSwitch(char *pBase) {
        bitset<8> switches;
        for (int i = 0; i < 8; i++) {
                switches.set(i, Read1Switch(pBase, i));
        }

        int switch_list= (int)(switches.to_ulong());
        return switch_list;
}

int ReadButton(char *pBase, int value) {
        bitset<8> buttons(value);
                cout << "Previous value: " << value << endl;
                if(RegisterRead(pBase, 0x16C) == 1) {
                        sleep(1);
                        buttons <<= 1;
                         cout << "New bitset: " << buttons << endl;
                }
                else if(RegisterRead(pBase, 0x170) == 1) {
```

```
                        sleep(1);
                        buttons >>= 1;
                        cout << "New bitset: " << buttons << endl;
                }
                else if(RegisterRead(pBase, 0x174) == 1) {
                        sleep(1);
                        cout << "New value: " << value + 1 << endl;
                        return value + 1;
                }
                else if(RegisterRead(pBase, 0x178) == 1) {
                        sleep(1);
                        cout << "New value: " << value - 1 << endl;
                        return value - 1;

                }
                else if(RegisterRead(pBase, 0x17C) == 1) {
                        sleep(1);
                        buttons = ReadAllSwitch(pBase);
                        cout << "New bitset: " << buttons << endl;

                }
                int button_list = (int)(buttons.to_ulong());
                return button_list;
}

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);
        }

// ************** Put your code here **********************
        int value = 0;
        while (true) {
                if (value == ReadButton(pBase, value)) {
                        value = value;
                }
                else {
                        value = ReadButton(pBase, value);
                }
                WriteAllLeds(pBase, value);
        }
        Finalize(pBase, fd);
        return 0;
}
```

PushButtonClass

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

using namespace std;

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

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

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 ZedBoard {
        char *pBase;
        int fd;

public:
        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);
                }
        }

        ~ZedBoard() {
                munmap(pBase, gpio_size);
                close(fd);
        }

        void RegisterWrite(int offset, int value) {
                * (int *) (pBase + offset) = value;
        }

        int RegisterRead(int offset) {
                return * (int *) (pBase + offset);
        }

  void Write1Led(int ledNum, int state) {
        if (0 >= ledNum > 7) {
                cerr << "Trying to write to a non-existing LED" << endl;
        }
        int ledOffset = 0x12C + (ledNum * 4);
        this->RegisterWrite(ledOffset, state);
 }

 int Read1Switch(int switchNum) {
        if (0 > switchNum && switchNum > 7)
        {
                cerr << "Trying to read a non-existing switch" << endl;
        }
        int switchOffset = 0x14C + (switchNum * 4);
        this->RegisterRead(switchOffset);
 }

 void WriteAllLeds(int value) {
     bitset<8> bits(value);
     for(int i = 0; i < 8; i++) {
             this->Write1Led(i, bits[i]);
     }
}

 int ReadAllSwitch() {
        bitset<8> switches;
        for (int i = 0; i < 8; i++) {
                switches.set(i, this->Read1Switch(i));
        }

        int switch_list= (int)(switches.to_ulong());
        return switch_list;
 }
```

```
 int ReadButton(int value) {
        bitset<8> buttons(value);
                cout << "Previous value: " << value << endl;
                if(this->RegisterRead(0x16C) == 1) {
                        sleep(1);
                        buttons <<= 1;
                                cout << "New bitset: " << buttons << endl;
                }
                else if(this->RegisterRead(0x170) == 1) {
                        sleep(1);
                        buttons >>= 1;
                        cout << "New bitset: " << buttons << endl;
                }
                else if(this->RegisterRead(0x174) == 1) {
                        sleep(1);
                        cout << "New value: " << value + 1 << endl;
                        return value + 1;
                }
                else if(this->RegisterRead(0x178) == 1) {
                        sleep(1);
                        cout << "New value: " << value - 1 << endl;
                        return value - 1;

                }
                else if(this->RegisterRead(0x17C) == 1) {
                        sleep(1);
                        buttons = this->ReadAllSwitch();
                        cout << "New bitset: " << buttons << endl;

                }
                int button_list = (int)(buttons.to_ulong());
                return button_list;
 }
};

int main()
{
        ZedBoard zb;
// ************** Put your code here **********************
        int value = 0;
        while (true) {
                if (value == zb.ReadButton(value)) {
                        value = value;
                }
                else {
                        value = zb.ReadButton(value);
                }
                zb.WriteAllLeds(value);
        }

        zb.~ZedBoard();
}
```