Patricia Gavelek, Anders Dahl EECE2160	Embedded Design: Enabling Robotics Lab Assignment

Memory-Mapped I/O and Object-Oriented Programming

Patricia Gavelek, Anders Dahl
Gavelek.p@husky.neu.edu
dahl.a@husky.neu.edu

Submit date: Feb. 26, 2016
Due Date: Feb. 26, 2016

Abstract

This was the first lab using devices on the Zedboard. It introduced the concept of memory-mapped I/O to access devices available on the ZedBoard, including the LEDs, the switches, and buttons. We learned how to control these devices, first by writing a program that turned on the lights then by modifying the program to have the switches and buttons control the LEDs. In using the buttons, debouncing was a problem that had to be dealt with. Lastly, C++ was introduced, showing how the C++ classes could be used for abstraction when we changed our code to C++.

Introduction

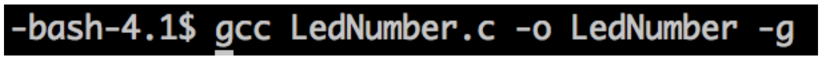
In this lab we introduce the concept of memory-mapped I/O to access devices available on the ZedBoard, including the LEDs, the switches, and the push buttons. We learned how to control these devices, first by writing a program that turned on the lights then by modifying the program to have the switches and buttons control the LEDs. In using the buttons, debouncing was a problem that had to be dealt with. Lastly, C++ was introduced, showing how the C++ classes could be used for abstraction when we changed our code to C++. The lab also provided familiarity with the concept of a root user and access control. The root user was used to give a normal user sudo privileges on the ZedBoard. The ZedBoard offers a variety of tools to use. The LEDs, switches, and push buttons were all controllable through the use of code. The lab scratched the surface of the cool things one can accomplish with the ZedBoard.

Lab Discussion

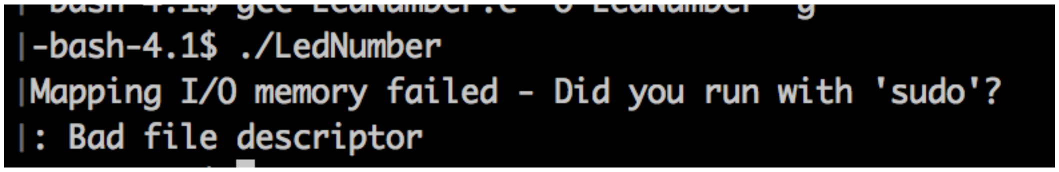
In order to develop the programs, the C programming language, the gcc compiler, the gdb debugger, and the ZedBoard were all used. A variety of C libraries were needed – stdio, stdlib, fcntl, unistd, and sys/mman.

The pre-lab showed what would happen if the C program ran on a normal linux machine rather than the ZedBoard. It would not work. It also went through the functions seen in LedNumber.c, and the student had to describe what all the functions did. Here are the results:

a).



```
-bash-4.1$ gcc LedNumber.c -o LedNumber -g
```



```
-bash-4.1$ ./LedNumber
Mapping I/O memory failed - Did you run with 'sudo'?
: Bad file descriptor
```

A file descriptor is an abstract indicator used to access an input/output source such as a file. The program fails during initialization. It tries to open "/dev/mem" for reading and writing, but it fails and returns -1 as the file descriptor. Next, the call to mmap exits with a perror message and the entire program shuts down.

b).

Initialize(): opens a file descriptor to “/dev/mem” with read and write access. It then returns an mmap which maps files or devices into memory. It wants to map the size of gpio_size, and each address should have an offset of gpio_address in the virtual address space. The allocated space should be readable, writable, and shared, and it should be linked to the file descriptor fd. The entire initialization returns the address to virtual memory which is mapped to physical or it gives an error.

Finalize(): closes previously open mappings and file descriptors. Removes the mapping previously allocated during initialize. Removes the entirety of it or the size of gpio_size. The finalize function then proceeds to close to opened file descriptor.

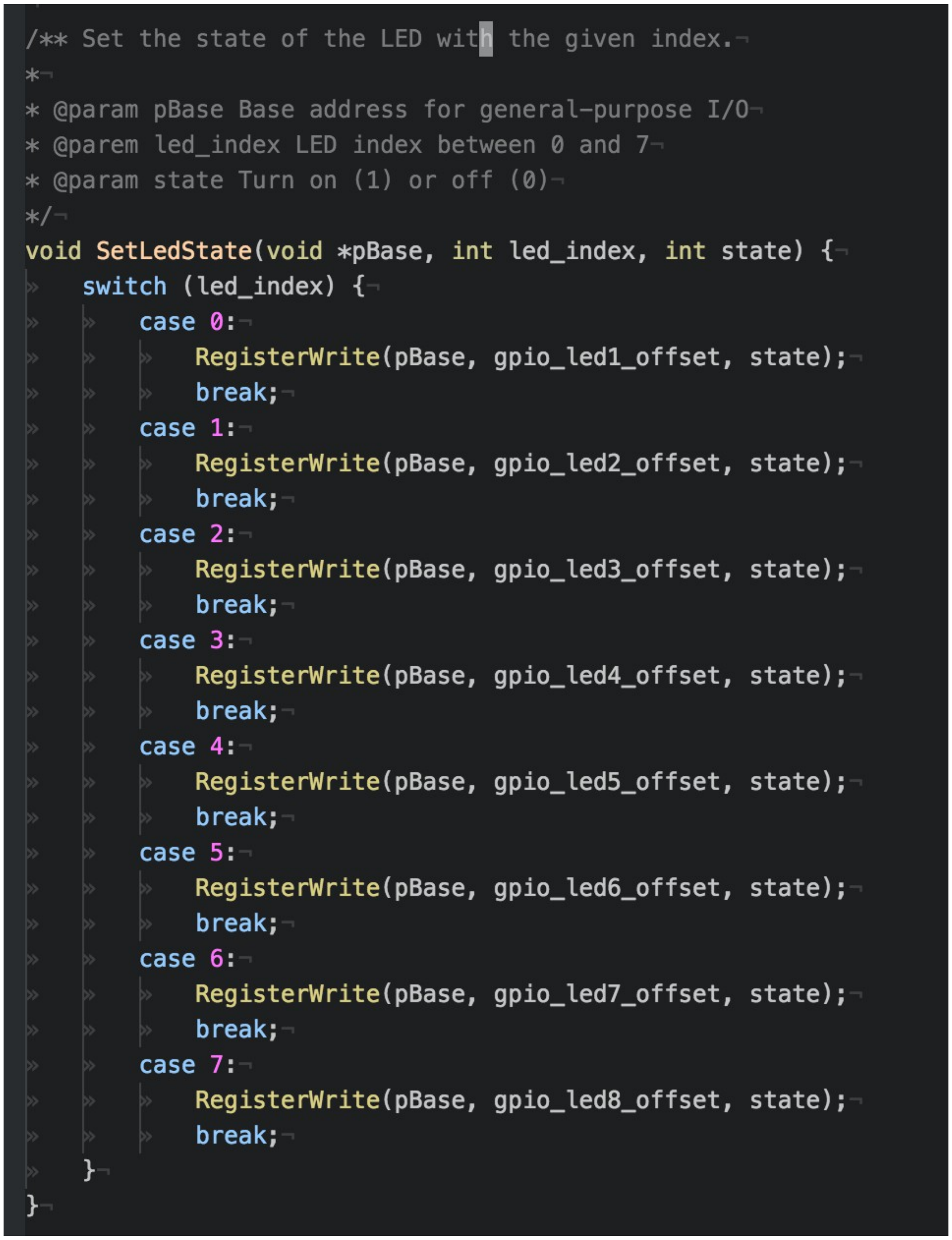
RegisterRead(): read a 4-byte value from the specified address in the virtual address space. You need to enter the base address returned from mmap and then the offset from that address.

RegisterWrite(): write a 4-byte value from the specified address in the virtual address space. You need to enter the base address returned from mmap and then the offset from that address. Now, at the correct location, write the value to that address space.

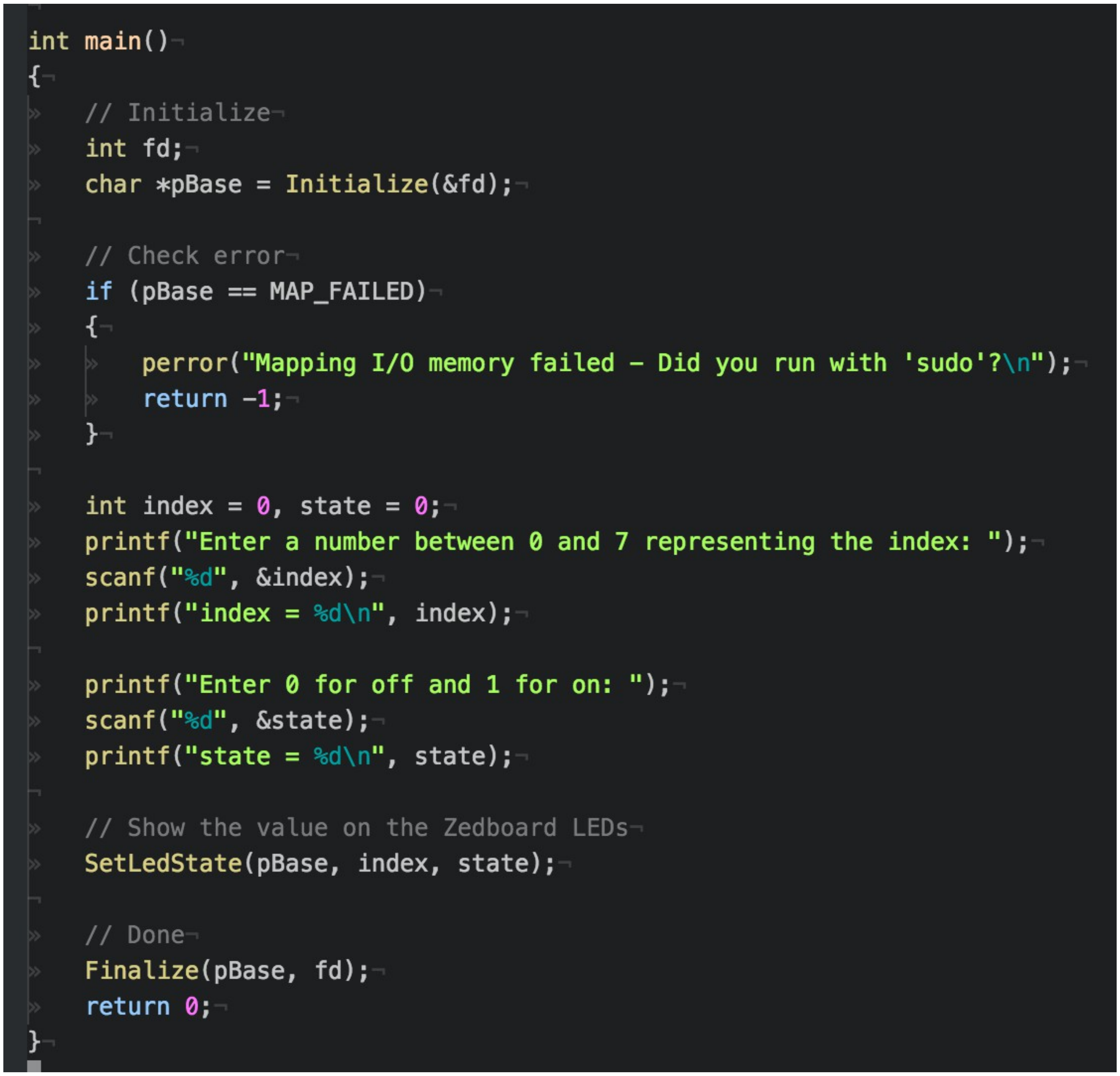
SetLedNumber(): Show the lower 8 bits of integer value on LEDs. This is done by writing to each address in control of the LEDs. The value of an individual bit can be calculated by utilizing modulus and division.

Results and Analysis

Assignment 2 – LedOnOff.c

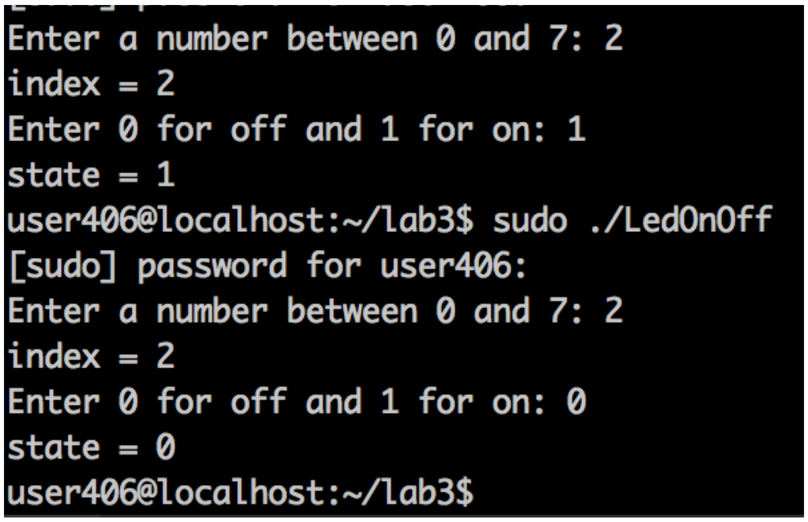


```c
/** Set the state of the LED with the given index.
*
* @param pBase Base address for general-purpose I/O
* @parem led_index LED index between 0 and 7
* @param state Turn on (1) or off (0)
*/
void SetLedState(void *pBase, int led_index, int state) {
    switch (led_index) {
        case 0:
            RegisterWrite(pBase, gpio_led1_offset, state);
            break;
        case 1:
            RegisterWrite(pBase, gpio_led2_offset, state);
            break;
        case 2:
            RegisterWrite(pBase, gpio_led3_offset, state);
            break;
        case 3:
            RegisterWrite(pBase, gpio_led4_offset, state);
            break;
        case 4:
            RegisterWrite(pBase, gpio_led5_offset, state);
            break;
        case 5:
            RegisterWrite(pBase, gpio_led6_offset, state);
            break;
        case 6:
            RegisterWrite(pBase, gpio_led7_offset, state);
            break;
        case 7:
            RegisterWrite(pBase, gpio_led8_offset, state);
            break;
    }
}
```



```
int main()
{
    // Initialize
    int fd;
    char *pBase = Initialize(&fd);

    // Check error
    if (pBase == MAP_FAILED)
    {
        perror("Mapping I/O memory failed - Did you run with 'sudo'?\n");
        return -1;
    }

    int index = 0, state = 0;
    printf("Enter a number between 0 and 7 representing the index: ");
    scanf("%d", &index);
    printf("index = %d\n", index);

    printf("Enter 0 for off and 1 for on: ");
    scanf("%d", &state);
    printf("state = %d\n", state);

    // Show the value on the Zedboard LEDs
    SetLedState(pBase, index, state);

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

Assignment 2 – Running the Code

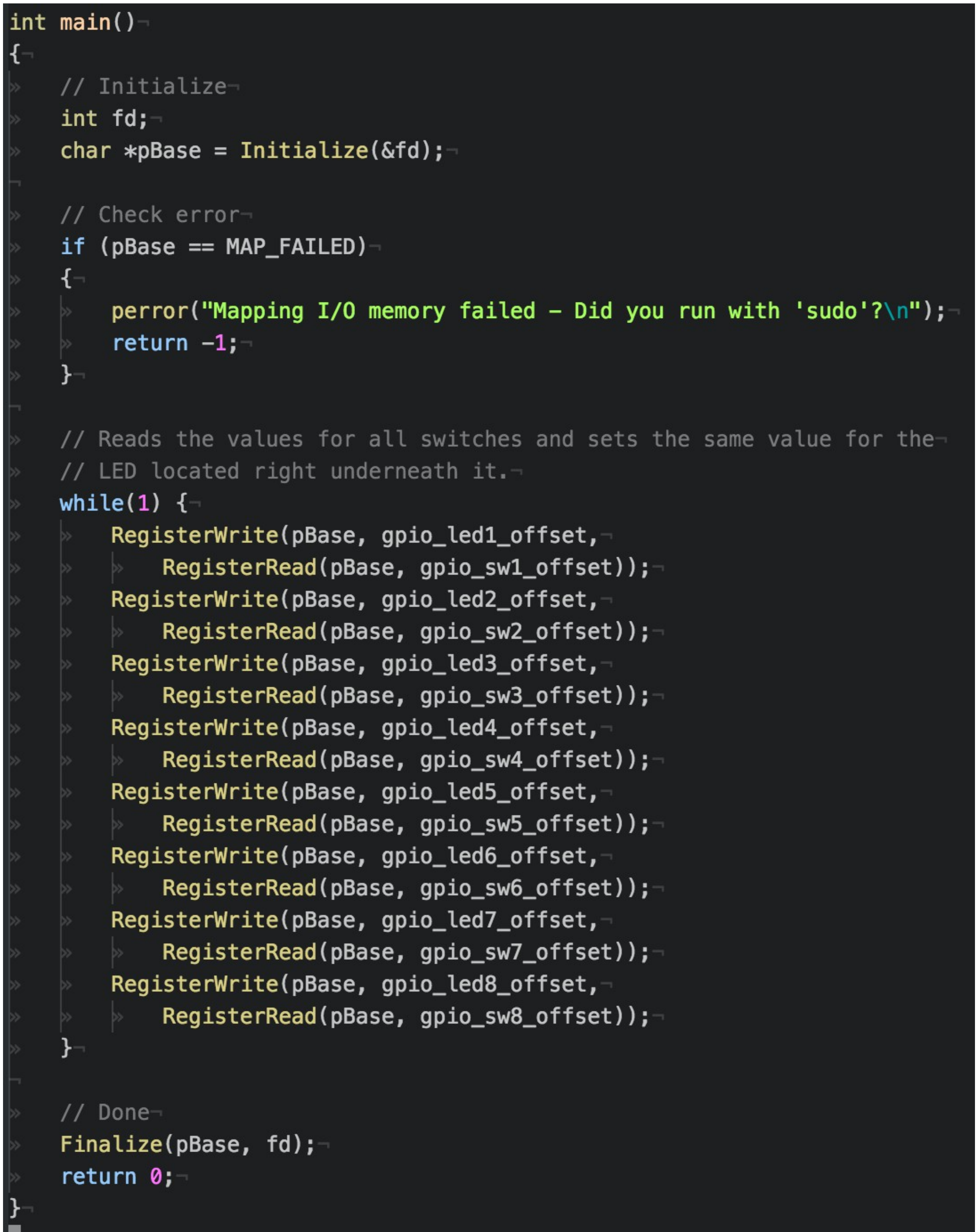


```
Enter a number between 0 and 7: 2
index = 2
Enter 0 for off and 1 for on: 1
state = 1
user406@localhost:~/lab3$ sudo ./LedOnOff
[sudo] password for user406:
Enter a number between 0 and 7: 2
index = 2
Enter 0 for off and 1 for on: 0
state = 0
user406@localhost:~/lab3$
```

We tested on a variety of inputs, including the ones seen in the picture. We were able to turn individual LEDs on or off. We tried turning the same LED on twice. The expected behavior occurred – no change.

Assignment 3 – SwitchToLed.c:

a)



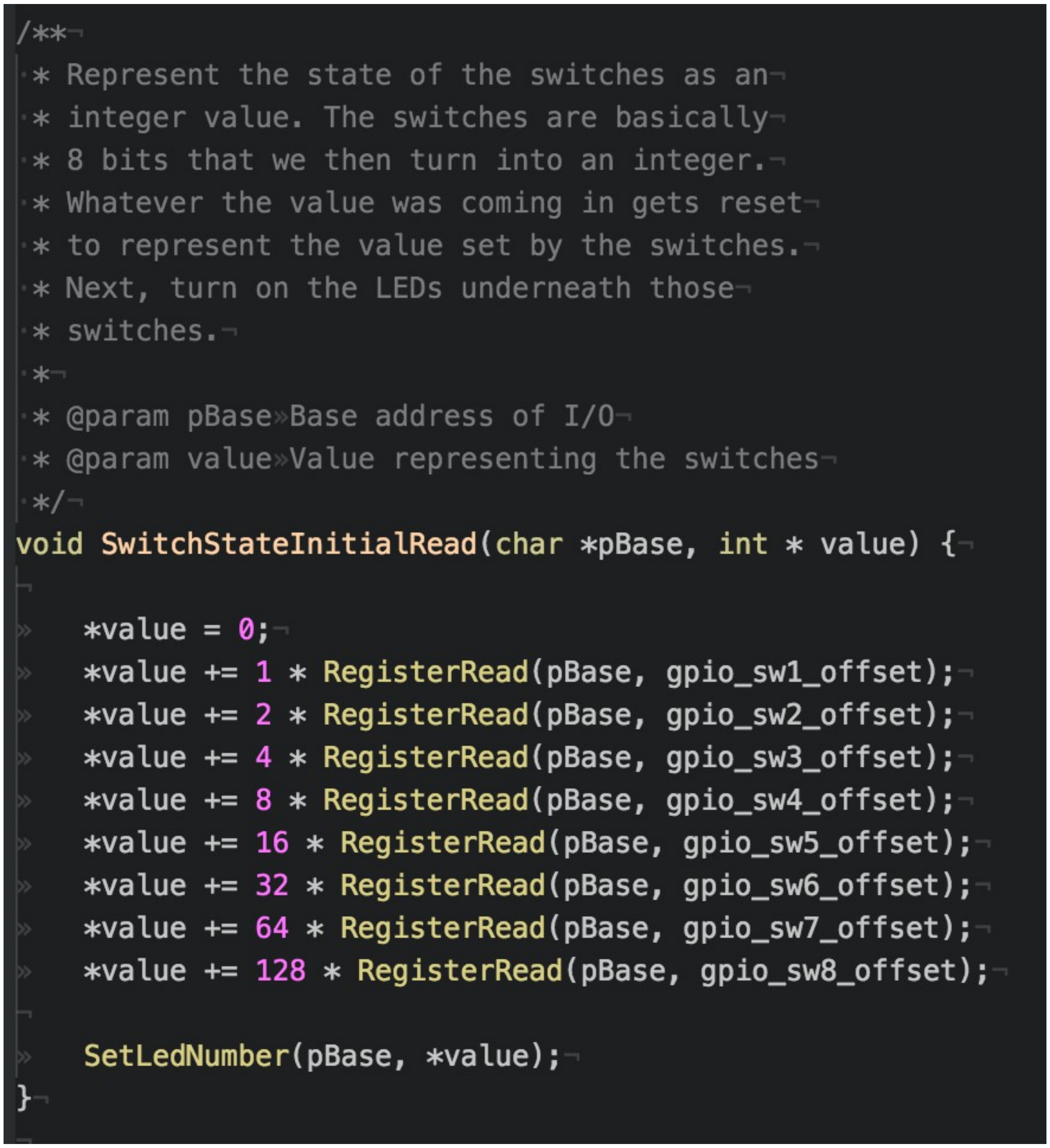
```c
int main()
{
    // Initialize
    int fd;
    char *pBase = Initialize(&fd);

    // Check error
    if (pBase == MAP_FAILED)
    {
        perror("Mapping I/O memory failed - Did you run with 'sudo'?\n");
        return -1;
    }

    // Reads the values for all switches and sets the same value for the
    // LED located right underneath it.
    while(1) {
        RegisterWrite(pBase, gpio_led1_offset,
            RegisterRead(pBase, gpio_sw1_offset));
        RegisterWrite(pBase, gpio_led2_offset,
            RegisterRead(pBase, gpio_sw2_offset));
        RegisterWrite(pBase, gpio_led3_offset,
            RegisterRead(pBase, gpio_sw3_offset));
        RegisterWrite(pBase, gpio_led4_offset,
            RegisterRead(pBase, gpio_sw4_offset));
        RegisterWrite(pBase, gpio_led5_offset,
            RegisterRead(pBase, gpio_sw5_offset));
        RegisterWrite(pBase, gpio_led6_offset,
            RegisterRead(pBase, gpio_sw6_offset));
        RegisterWrite(pBase, gpio_led7_offset,
            RegisterRead(pBase, gpio_sw7_offset));
        RegisterWrite(pBase, gpio_led8_offset,
            RegisterRead(pBase, gpio_sw8_offset));
    }

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

b)

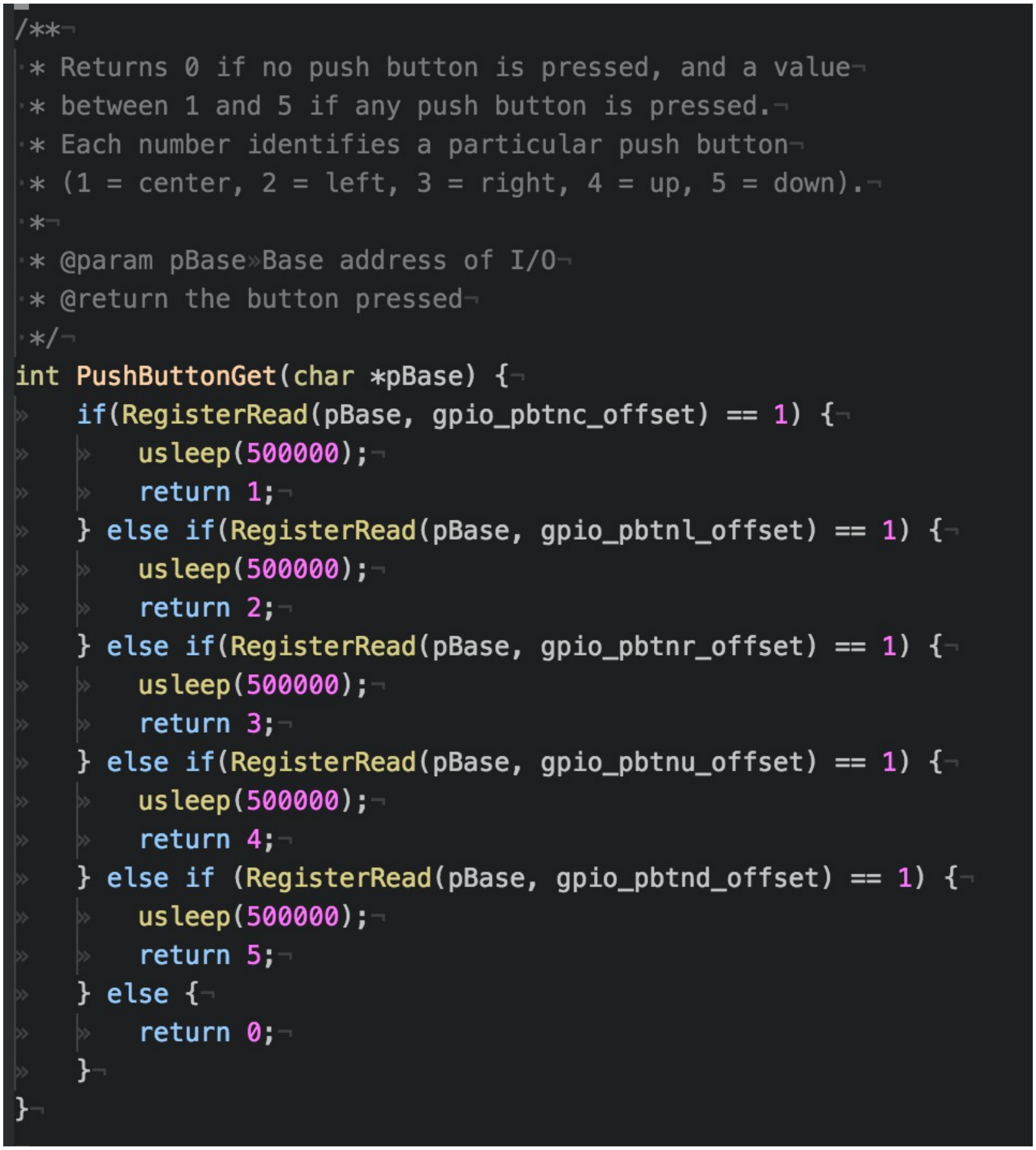
The SwitchToLed.c along with all other programs were shown to the instructor or a TA.

Assignment 4 – PushButton.c:

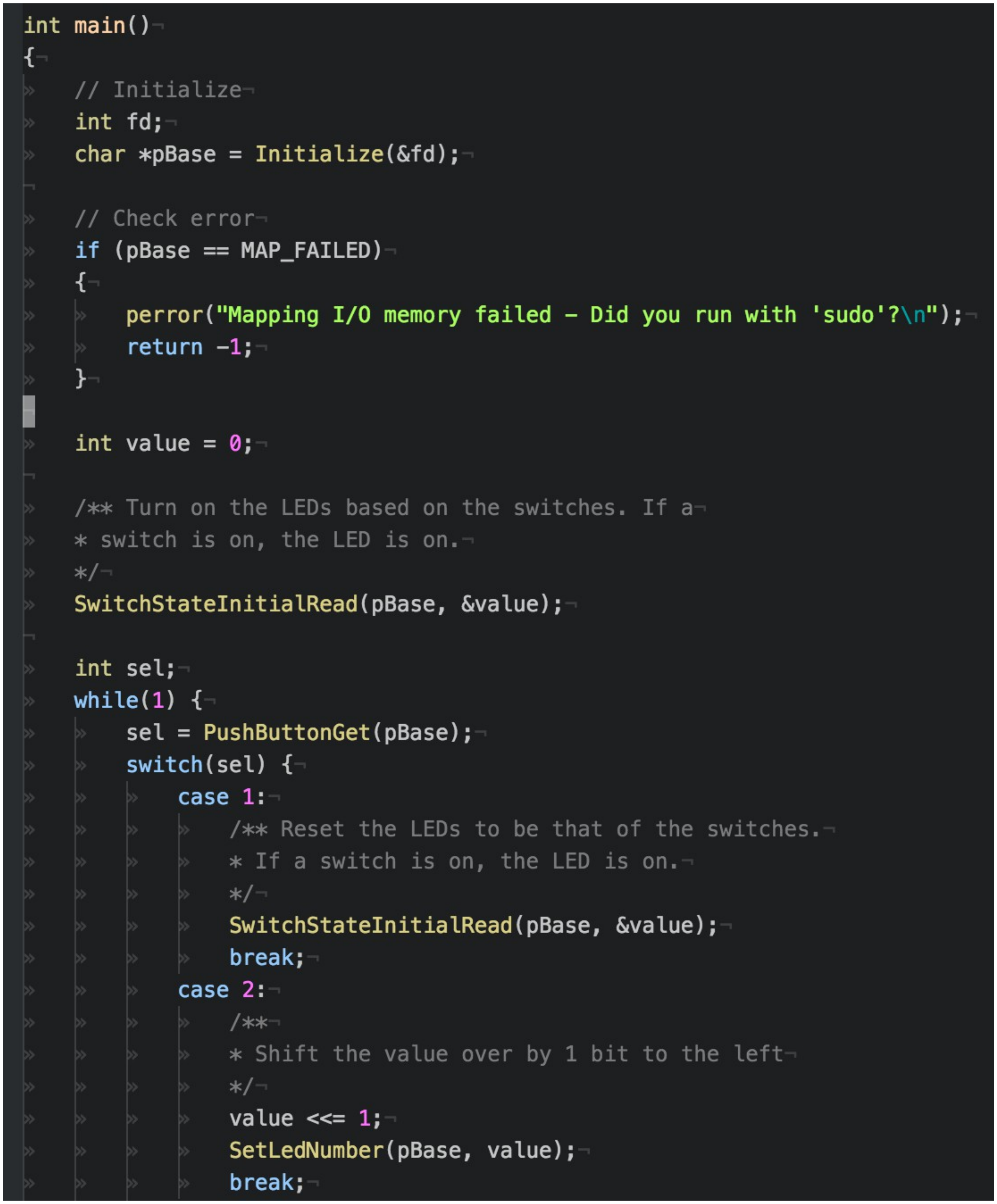
a)



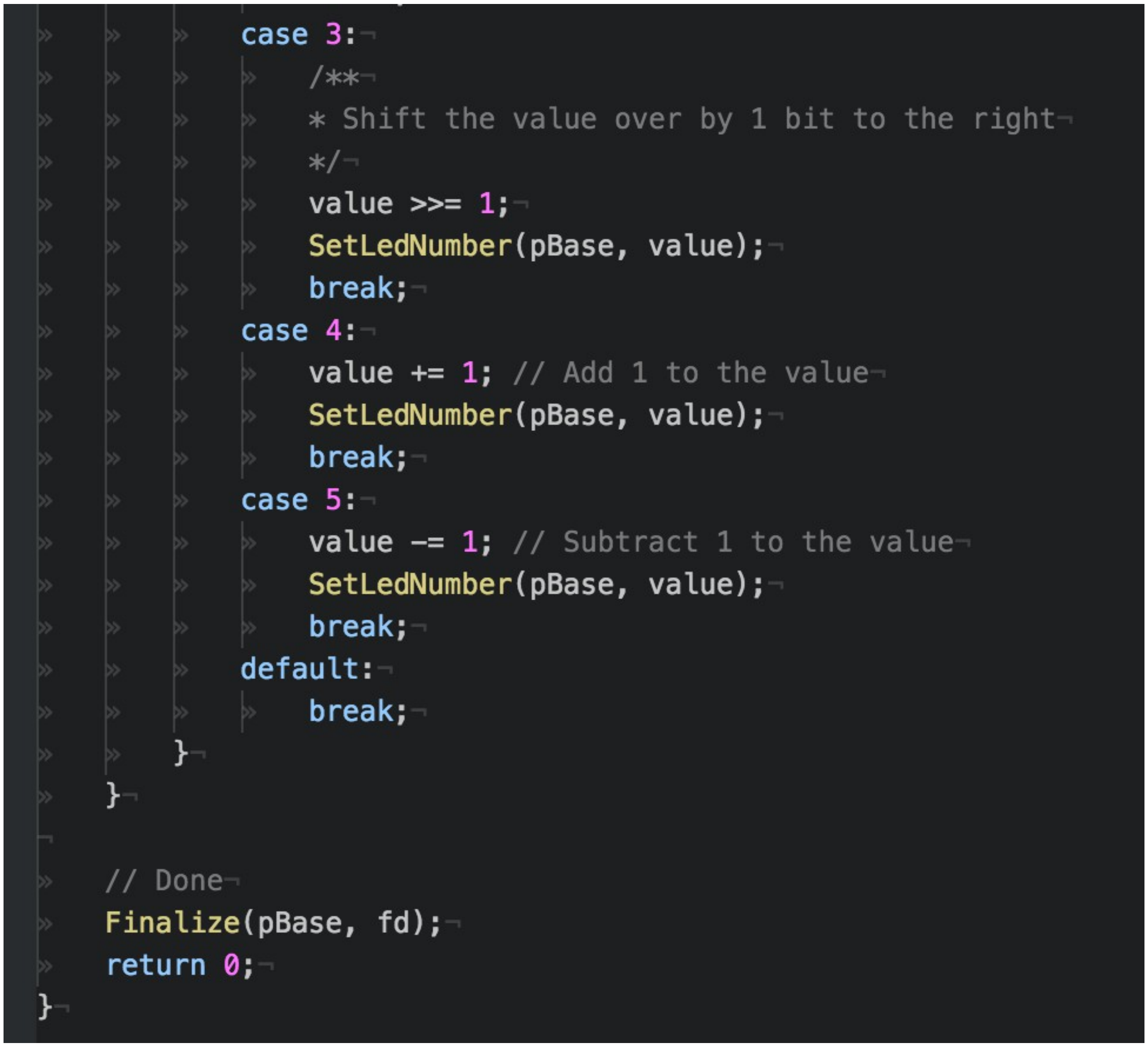
```
/**
 * Represent the state of the switches as an
 * integer value. The switches are basically
 * 8 bits that we then turn into an integer.
 * Whatever the value was coming in gets reset
 * to represent the value set by the switches.
 * Next, turn on the LEDs underneath those
 * switches.
 *
 * @param pBase	Base address of I/O
 * @param value	Value representing the switches
 */
void SwitchStateInitialRead(char *pBase, int * value) {

	*value = 0;
	*value += 1 * RegisterRead(pBase, gpio_sw1_offset);
	*value += 2 * RegisterRead(pBase, gpio_sw2_offset);
	*value += 4 * RegisterRead(pBase, gpio_sw3_offset);
	*value += 8 * RegisterRead(pBase, gpio_sw4_offset);
	*value += 16 * RegisterRead(pBase, gpio_sw5_offset);
	*value += 32 * RegisterRead(pBase, gpio_sw6_offset);
	*value += 64 * RegisterRead(pBase, gpio_sw7_offset);
	*value += 128 * RegisterRead(pBase, gpio_sw8_offset);

	SetLedNumber(pBase, *value);
}
```



```c
/**
 * Returns 0 if no push button is pressed, and a value
 * between 1 and 5 if any push button is pressed.
 * Each number identifies a particular push button
 * (1 = center, 2 = left, 3 = right, 4 = up, 5 = down).
 *
 * @param pBase	Base address of I/O
 * @return the button pressed
 */
int PushButtonGet(char *pBase) {
	if(RegisterRead(pBase, gpio_pbtnc_offset) == 1) {
		usleep(500000);
		return 1;
	} else if(RegisterRead(pBase, gpio_pbtnl_offset) == 1) {
		usleep(500000);
		return 2;
	} else if(RegisterRead(pBase, gpio_pbtnr_offset) == 1) {
		usleep(500000);
		return 3;
	} else if(RegisterRead(pBase, gpio_pbtnu_offset) == 1) {
		usleep(500000);
		return 4;
	} else if (RegisterRead(pBase, gpio_pbtnd_offset) == 1) {
		usleep(500000);
		return 5;
	} else {
		return 0;
	}
}
```



```c
int main()
{
    // Initialize
    int fd;
    char *pBase = Initialize(&fd);

    // Check error
    if (pBase == MAP_FAILED)
    {
        perror("Mapping I/O memory failed - Did you run with 'sudo'?\n");
        return -1;
    }

    int value = 0;

    /** Turn on the LEDs based on the switches. If a
    * switch is on, the LED is on.
    */
    SwitchStateInitialRead(pBase, &value);

    int sel;
    while(1) {
        sel = PushButtonGet(pBase);
        switch(sel) {
            case 1:
                /** Reset the LEDs to be that of the switches.
                * If a switch is on, the LED is on.
                */
                SwitchStateInitialRead(pBase, &value);
                break;
            case 2:
                /**
                * Shift the value over by 1 bit to the left
                */
                value <<= 1;
                SetLedNumber(pBase, value);
                break;
```



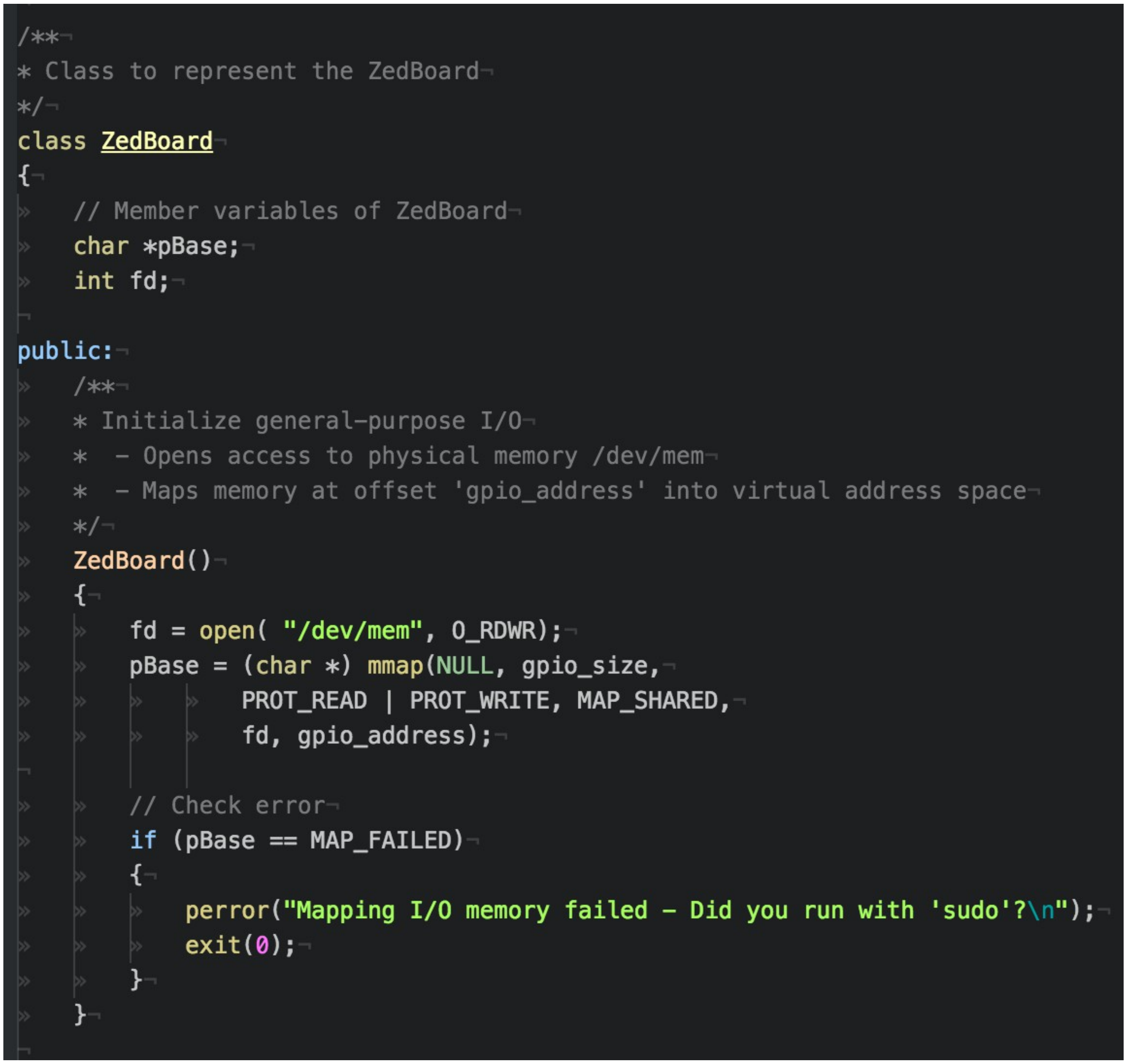
```
            case 3:
                /**
                * Shift the value over by 1 bit to the right
                */
                value >>= 1;
                SetLedNumber(pBase, value);
                break;
            case 4:
                value += 1; // Add 1 to the value
                SetLedNumber(pBase, value);
                break;
            case 5:
                value -= 1; // Subtract 1 to the value
                SetLedNumber(pBase, value);
                break;
            default:
                break;
        }
    }

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

b)

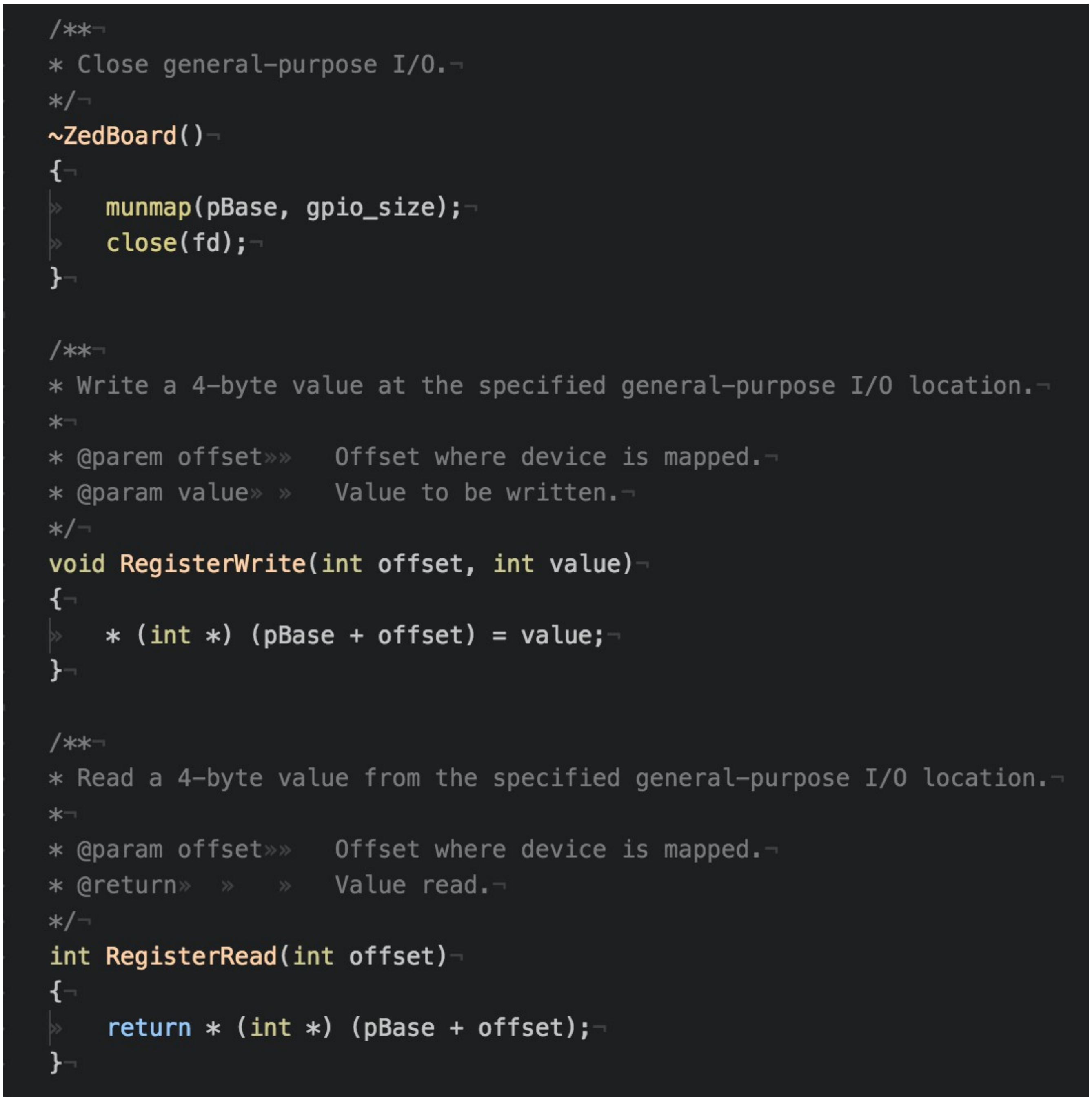
The PushButton.c along with all other programs were shown to the instructor or a TA.

Assignment 5 – PushButtonCpp.cpp:

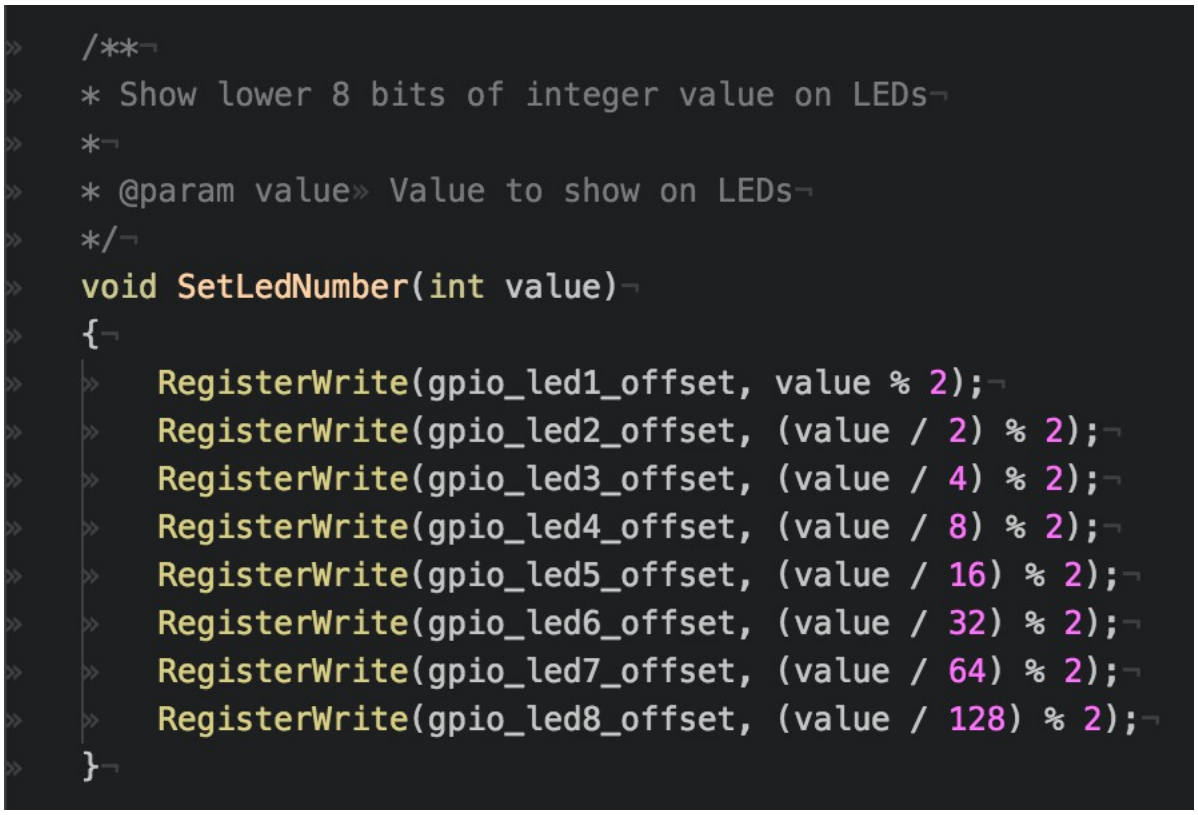
a)



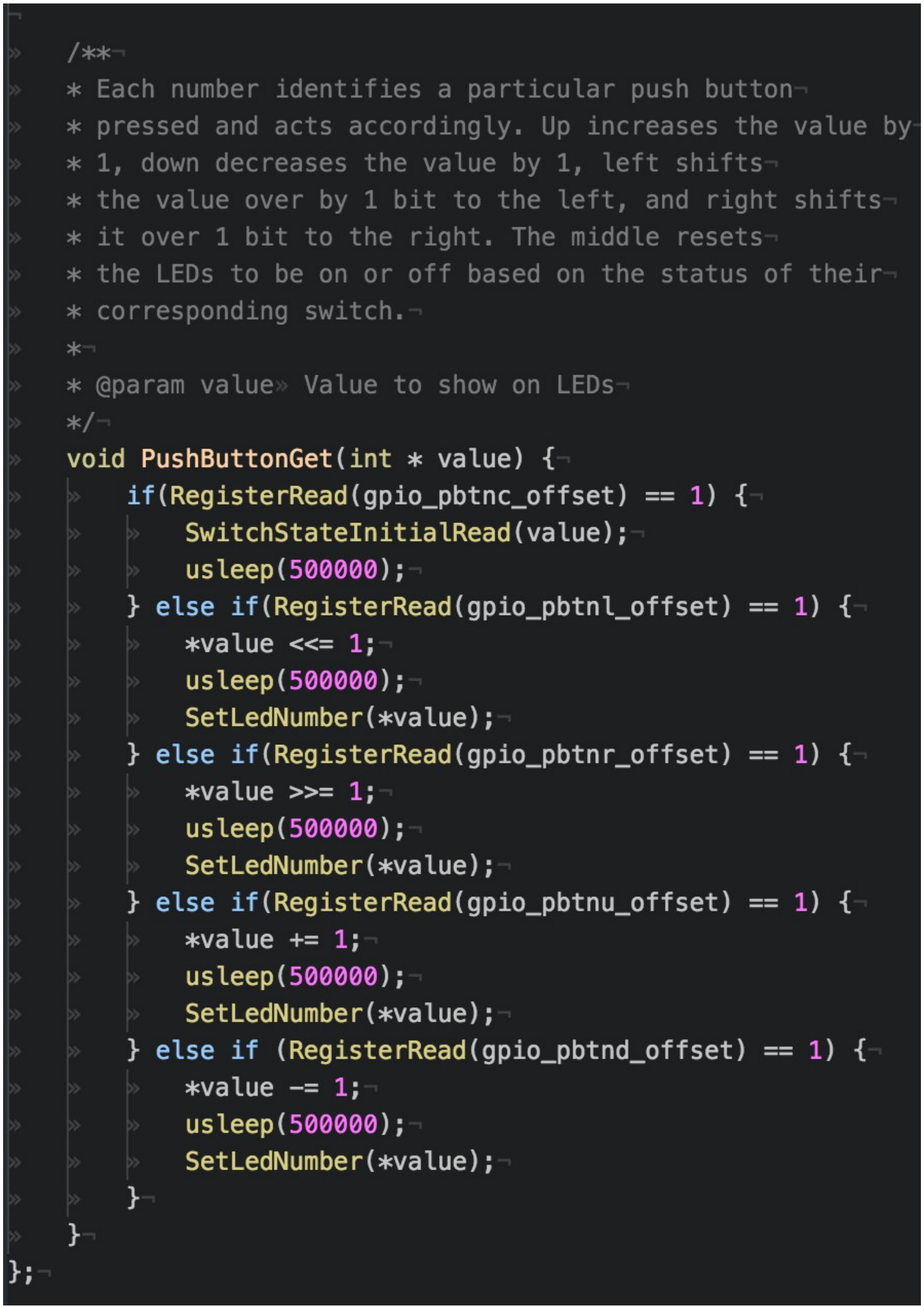
```
/**
* Class to represent the ZedBoard
*/
class ZedBoard
{
	// Member variables of ZedBoard
	char *pBase;
	int fd;

public:
	/**
	* Initialize general-purpose I/O
	*  - Opens access to physical memory /dev/mem
	*  - Maps memory at offset 'gpio_address' into virtual address space
	*/
	ZedBoard()
	{
		fd = open( "/dev/mem", O_RDWR);
		pBase = (char *) mmap(NULL, gpio_size,
				PROT_READ | PROT_WRITE, MAP_SHARED,
				fd, gpio_address);

		// Check error
		if (pBase == MAP_FAILED)
		{
			perror("Mapping I/O memory failed - Did you run with 'sudo'?\n");
			exit(0);
		}
	}
```



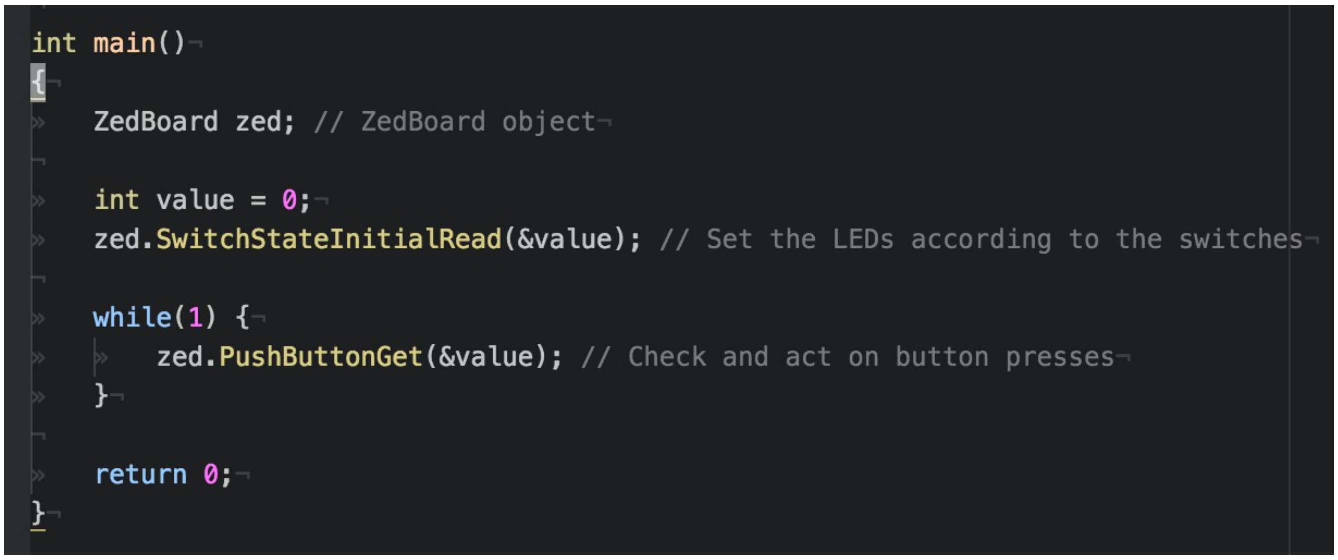
```
/**
* Close general-purpose I/O.
*/
~ZedBoard()
{
    munmap(pBase, gpio_size);
    close(fd);
}

/**
* Write a 4-byte value at the specified general-purpose I/O location.
*
* @parem offset      Offset where device is mapped.
* @param value       Value to be written.
*/
void RegisterWrite(int offset, int value)
{
    * (int *) (pBase + offset) = value;
}

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



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



```
    /**
    * Each number identifies a particular push button
    * pressed and acts accordingly. Up increases the value by
    * 1, down decreases the value by 1, left shifts
    * the value over by 1 bit to the left, and right shifts
    * it over 1 bit to the right. The middle resets
    * the LEDs to be on or off based on the status of their
    * corresponding switch.
    *
    * @param value	Value to show on LEDs
    */
    void PushButtonGet(int * value) {
        if(RegisterRead(gpio_pbtnc_offset) == 1) {
            SwitchStateInitialRead(value);
            usleep(500000);
        } else if(RegisterRead(gpio_pbtnl_offset) == 1) {
            *value <<= 1;
            usleep(500000);
            SetLedNumber(*value);
        } else if(RegisterRead(gpio_pbtnr_offset) == 1) {
            *value >>= 1;
            usleep(500000);
            SetLedNumber(*value);
        } else if(RegisterRead(gpio_pbtnu_offset) == 1) {
            *value += 1;
            usleep(500000);
            SetLedNumber(*value);
        } else if (RegisterRead(gpio_pbtnd_offset) == 1) {
            *value -= 1;
            usleep(500000);
            SetLedNumber(*value);
        }
    }
};
```



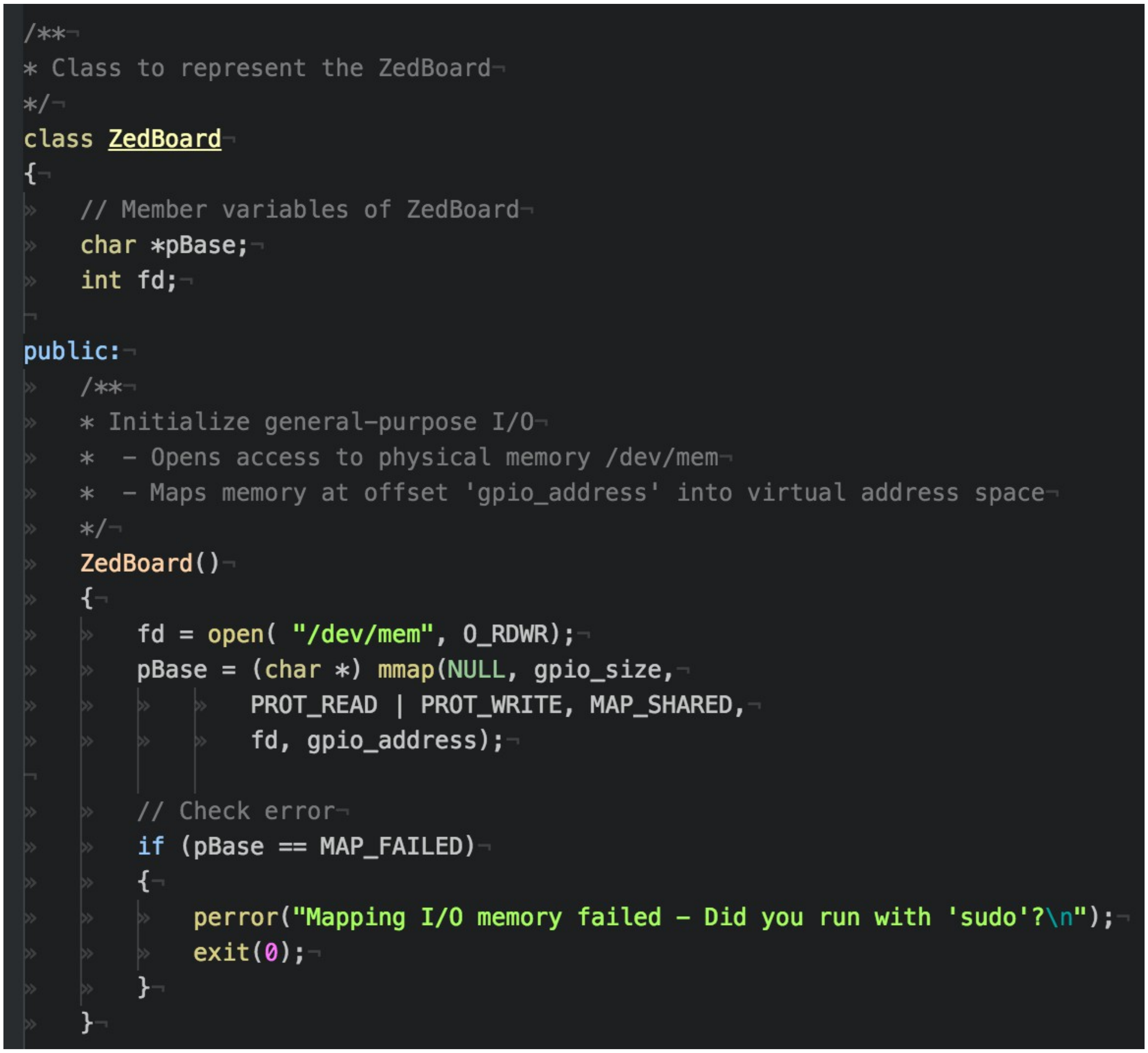
```
int main()
{
    ZedBoard zed; // ZedBoard object

    int value = 0;
    zed.SwitchStateInitialRead(&value); // Set the LEDs according to the switches

    while(1) {
        zed.PushButtonGet(&value); // Check and act on button presses
    }

    return 0;
}
```

b)

The PushButtonCpp.cpp along with all other programs were shown to the instructor or a TA.

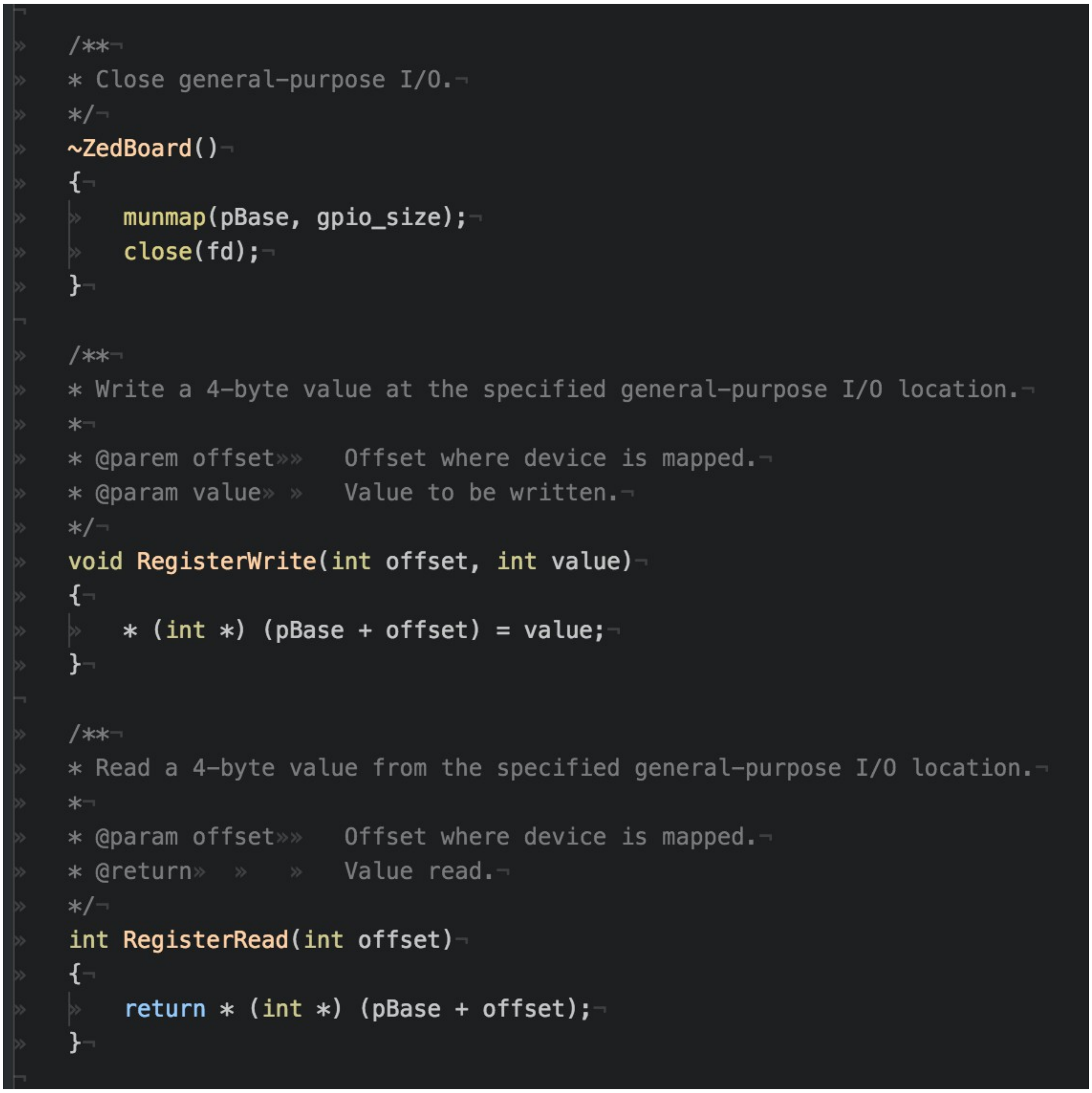
Extra Credit – CounterSpeed.cpp:

a)

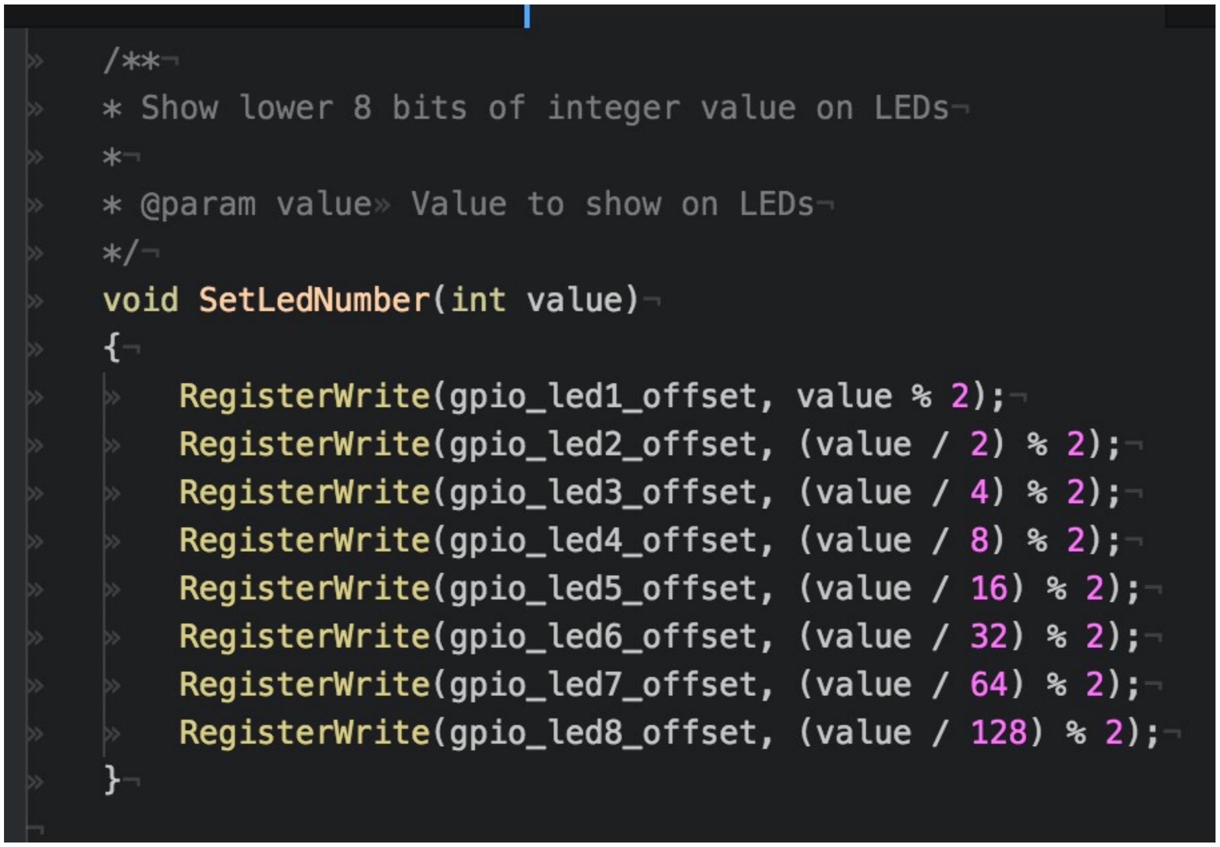


```
/**
* Class to represent the ZedBoard
*/
class ZedBoard
{
    // Member variables of ZedBoard
    char *pBase;
    int fd;

public:
    /**
    * Initialize general-purpose I/O
    *  - Opens access to physical memory /dev/mem
    *  - Maps memory at offset 'gpio_address' into virtual address space
    */
    ZedBoard()
    {
        fd = open( "/dev/mem", O_RDWR);
        pBase = (char *) mmap(NULL, gpio_size,
                PROT_READ | PROT_WRITE, MAP_SHARED,
                fd, gpio_address);

        // Check error
        if (pBase == MAP_FAILED)
        {
            perror("Mapping I/O memory failed - Did you run with 'sudo'?\n");
            exit(0);
        }
    }
```

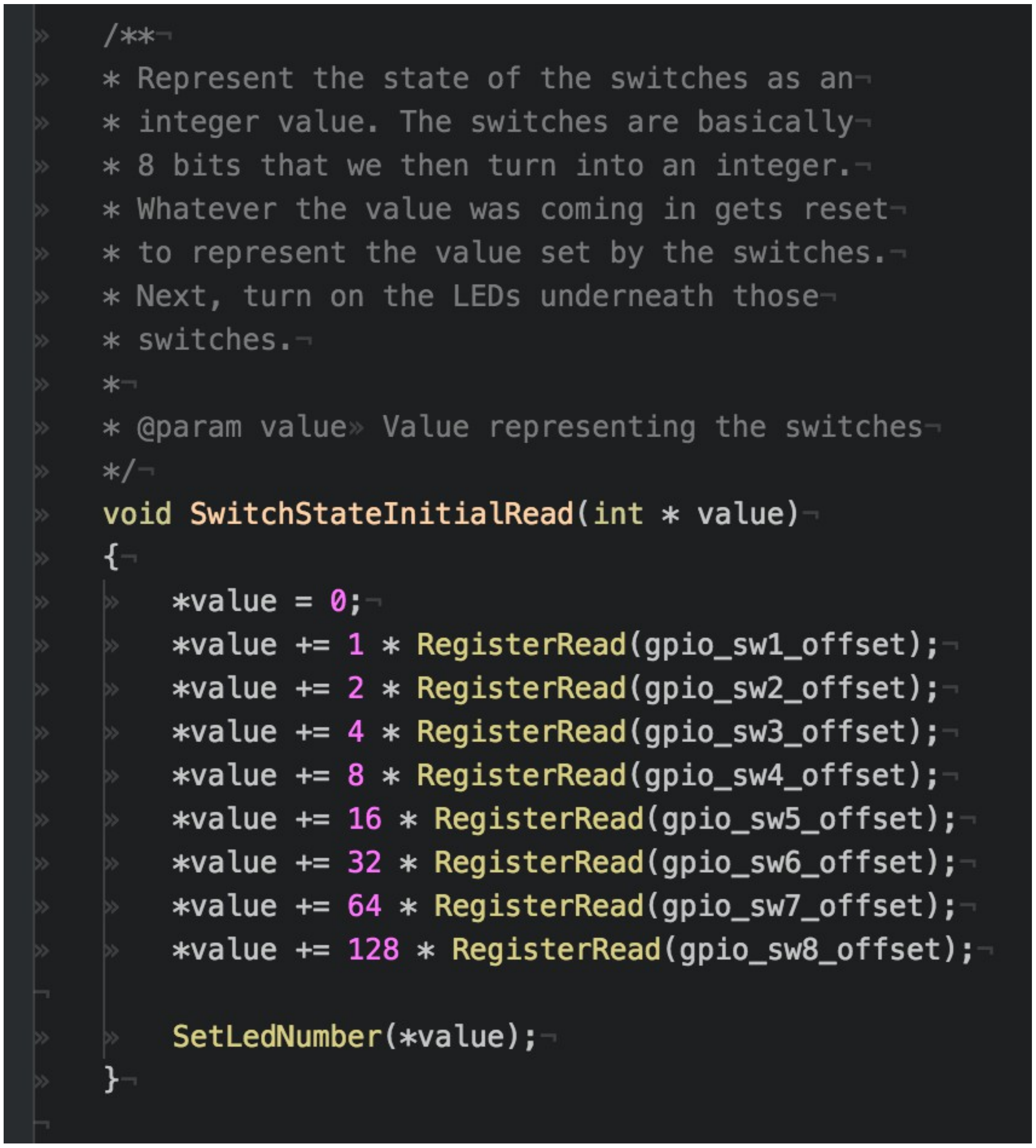
Patricia Gavelek, Anders Dahl EECE2160	Embedded Design: Enabling Robotics Lab Assignment



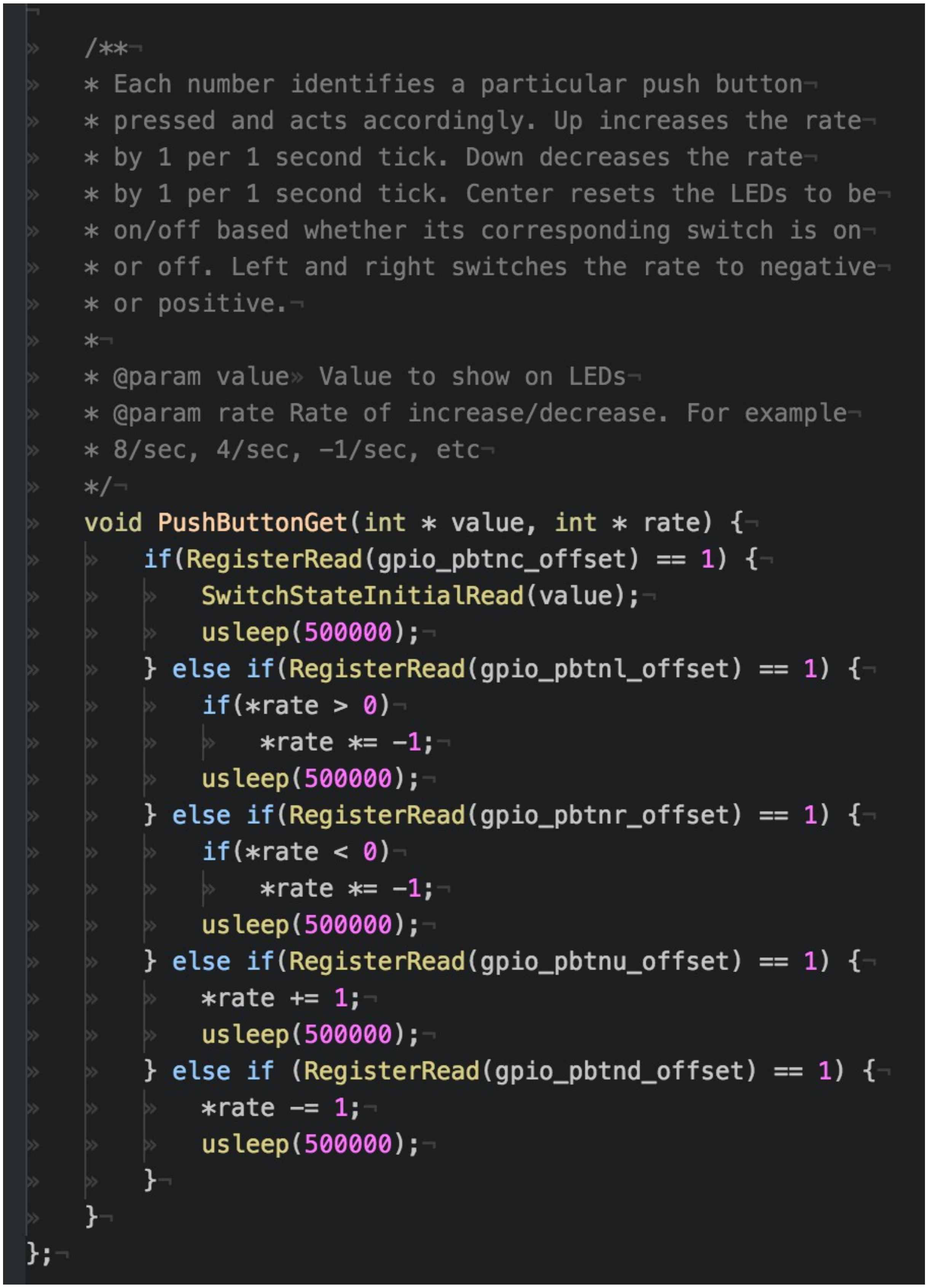
```
    /**
    * Close general-purpose I/O.
    */
    ~ZedBoard()
    {
        munmap(pBase, gpio_size);
        close(fd);
    }

    /**
    * Write a 4-byte value at the specified general-purpose I/O location.
    *
    * @parem offset      Offset where device is mapped.
    * @param value       Value to be written.
    */
    void RegisterWrite(int offset, int value)
    {
        * (int *) (pBase + offset) = value;
    }

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



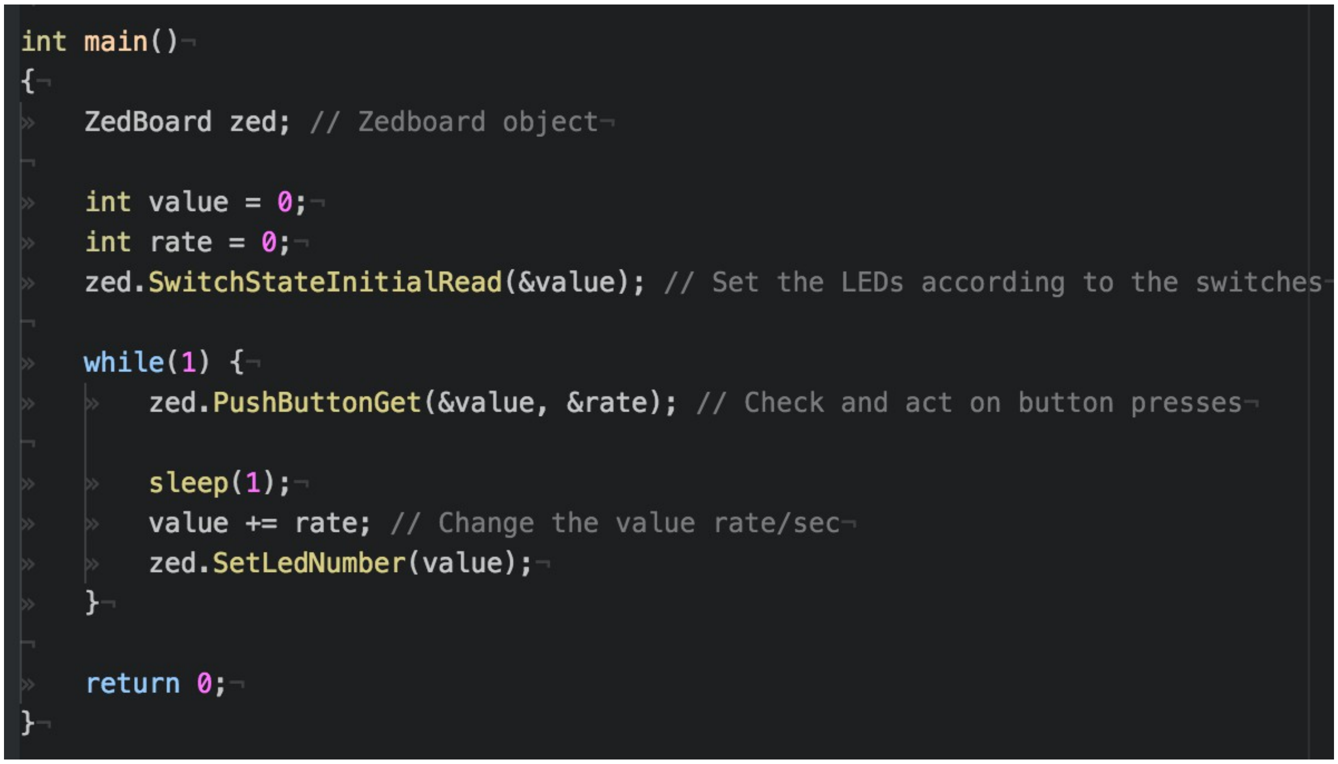
```c
/**
* Show lower 8 bits of integer value on LEDs
*
* @param value	Value to show on LEDs
*/
void SetLedNumber(int value)
{
	RegisterWrite(gpio_led1_offset, value % 2);
	RegisterWrite(gpio_led2_offset, (value / 2) % 2);
	RegisterWrite(gpio_led3_offset, (value / 4) % 2);
	RegisterWrite(gpio_led4_offset, (value / 8) % 2);
	RegisterWrite(gpio_led5_offset, (value / 16) % 2);
	RegisterWrite(gpio_led6_offset, (value / 32) % 2);
	RegisterWrite(gpio_led7_offset, (value / 64) % 2);
	RegisterWrite(gpio_led8_offset, (value / 128) % 2);
}
```



```
/**
 * Represent the state of the switches as an
 * integer value. The switches are basically
 * 8 bits that we then turn into an integer.
 * Whatever the value was coming in gets reset
 * to represent the value set by the switches.
 * Next, turn on the LEDs underneath those
 * switches.
 *
 * @param value	Value representing the switches
 */
void SwitchStateInitialRead(int * value)
{
	*value = 0;
	*value += 1 * RegisterRead(gpio_sw1_offset);
	*value += 2 * RegisterRead(gpio_sw2_offset);
	*value += 4 * RegisterRead(gpio_sw3_offset);
	*value += 8 * RegisterRead(gpio_sw4_offset);
	*value += 16 * RegisterRead(gpio_sw5_offset);
	*value += 32 * RegisterRead(gpio_sw6_offset);
	*value += 64 * RegisterRead(gpio_sw7_offset);
	*value += 128 * RegisterRead(gpio_sw8_offset);

	SetLedNumber(*value);
}
```



```
    /**
    * Each number identifies a particular push button
    * pressed and acts accordingly. Up increases the rate
    * by 1 per 1 second tick. Down decreases the rate
    * by 1 per 1 second tick. Center resets the LEDs to be
    * on/off based whether its corresponding switch is on
    * or off. Left and right switches the rate to negative
    * or positive.
    *
    * @param value	Value to show on LEDs
    * @param rate Rate of increase/decrease. For example
    * 8/sec, 4/sec, -1/sec, etc
    */
    void PushButtonGet(int * value, int * rate) {
        if(RegisterRead(gpio_pbtnc_offset) == 1) {
            SwitchStateInitialRead(value);
            usleep(500000);
        } else if(RegisterRead(gpio_pbtnl_offset) == 1) {
            if(*rate > 0)
                *rate *= -1;
            usleep(500000);
        } else if(RegisterRead(gpio_pbtnr_offset) == 1) {
            if(*rate < 0)
                *rate *= -1;
            usleep(500000);
        } else if(RegisterRead(gpio_pbtnu_offset) == 1) {
            *rate += 1;
            usleep(500000);
        } else if (RegisterRead(gpio_pbtnd_offset) == 1) {
            *rate -= 1;
            usleep(500000);
        }
    }
};
```



```
int main()
{
    ZedBoard zed; // Zedboard object

    int value = 0;
    int rate = 0;
    zed.SwitchStateInitialRead(&value); // Set the LEDs according to the switches

    while(1) {
        zed.PushButtonGet(&value, &rate); // Check and act on button presses

        sleep(1);
        value += rate; // Change the value rate/sec
        zed.SetLedNumber(value);
    }

    return 0;
}
```

b)

The CounterSpeed.cpp along with all other programs were shown to the instructor or a TA.

Conclusion

Overall, in this lab we learned how to write code that controls the LEDs, buttons and switches on the Zedboard. Knowing how to use the memory-mapping IO and controlling the Zedboard devices will be useful later when we start using the Wii remote and robotic arm. For the buttons, we took care of debouncing by using usleep, so that it would only read one value when the button was pushed. This way, a button could be pushed twice in a row rather than using a state field that changed when a button was pushed. In general, knowing how to read and manipulate states and values critical to embedded design. In the future, more programs could be written to control the LEDs in different ways. The programs in the lab could also be rewritten. The files could be split up into proper C++ class files – makefiles, main.cpp, class.cpp, class.h, etc.

References

N/A