Embedded Des Enabling Robotics
Lab 3 report

Shiyu Wang
Hao Jiang

Fall 2016
September 30

Introduction

This lab further familiar us with the use of ZedBoard. In addition, we learnt how to use general purpose memory mapped I/O on the ZedBoard. We wrote code on Linux system and controlled the LEDs, switches and buttons on the ZedBoard. We wrote a primary program as pre-lab and did a lot of research on the use of functions such as *open()*, *mmap(), REG_WRITE(),* and munmap(). We incorporated our program *userio_ledSet()* which was written as pre-lab into the program given on BlackBoard. This gave us a result for 3.1. We then did some revision on this program, and used *REG_READ()* rather than *REG_WRITE()* in the program since we learnt in class that LEDs can only be write to, while switches can only be written on. We then write to read from the push button and printed which button we pushed on the window. Lab 3.5 is a little tricky. We didn't figure it out in class, but after we attended Friday class on the 29th September, and learnt about Macro, we finally created a functional program.

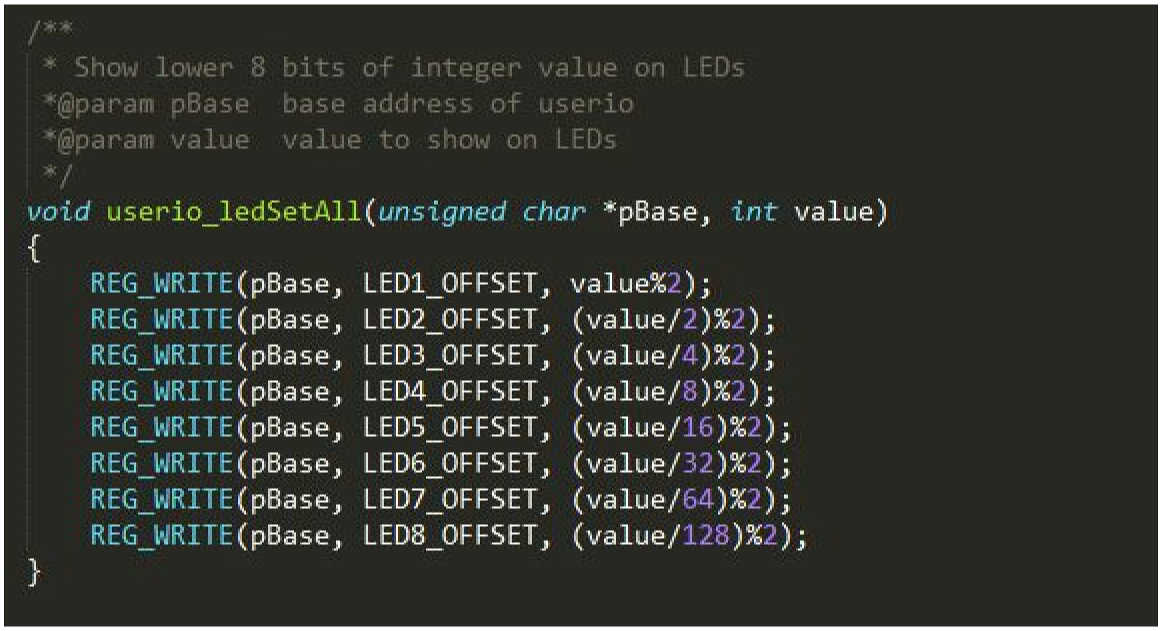
Lab Preparation

We first logged onto the computer using our credentials. We then connected the Zedboard to the PC host via Ethernet to the RJ45 connector. The ZedBoard was boosted after connected to the computer. We made an SSN connection to the ZedBoard where the IP is 192.168.1.10, and port 22.

Apart from the usual routine above, we did something unusual. We logged into ZedBoard as root rather than the usual <user550>. We then add <user550> to allow sudo access by the command: adduser <user550> sudo. We logged off from the system as root and relogged as sudo user550. This time, when we ran our program, we have to prefix the program as sudo <program name>

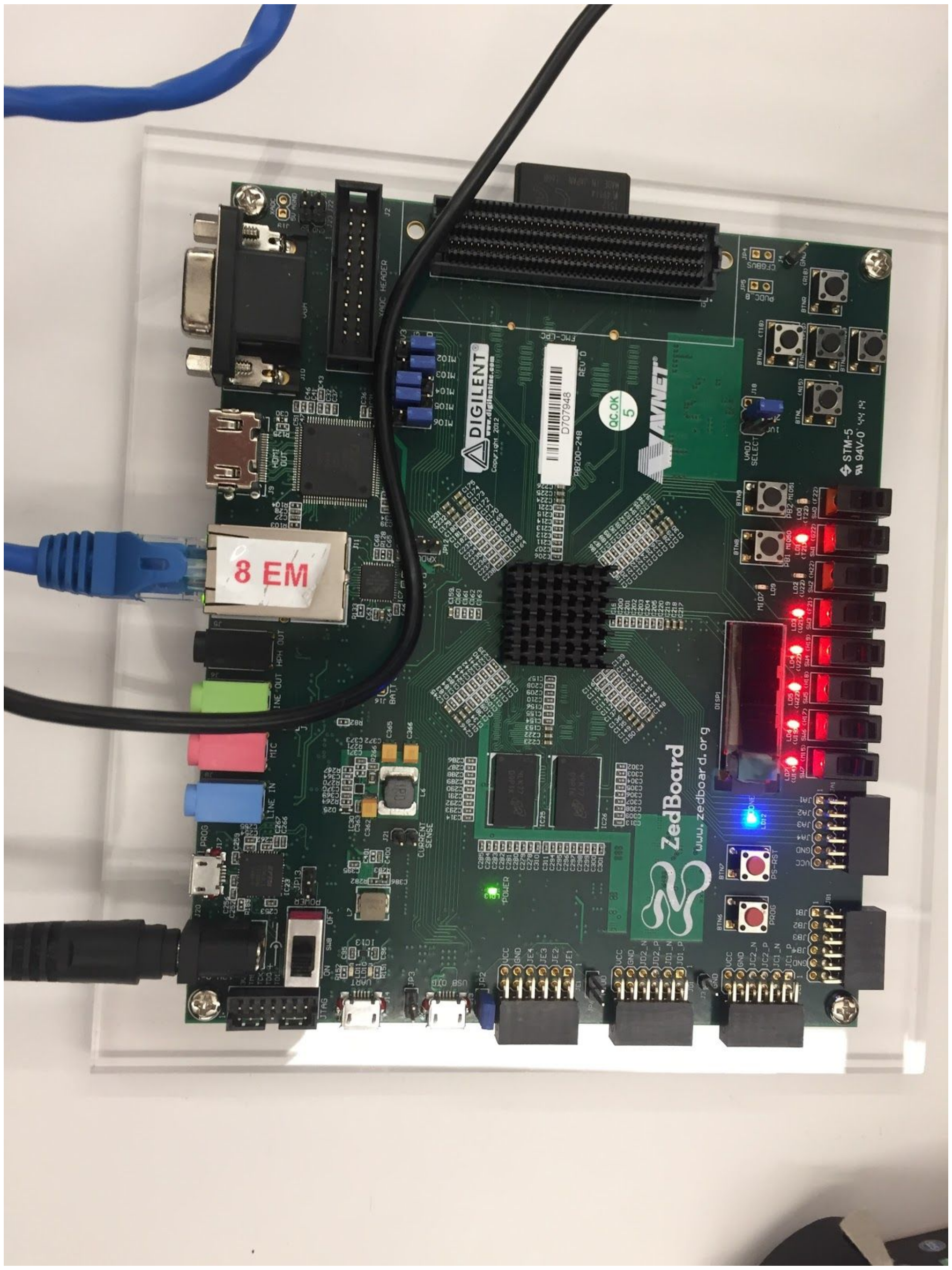
Lab 3.1

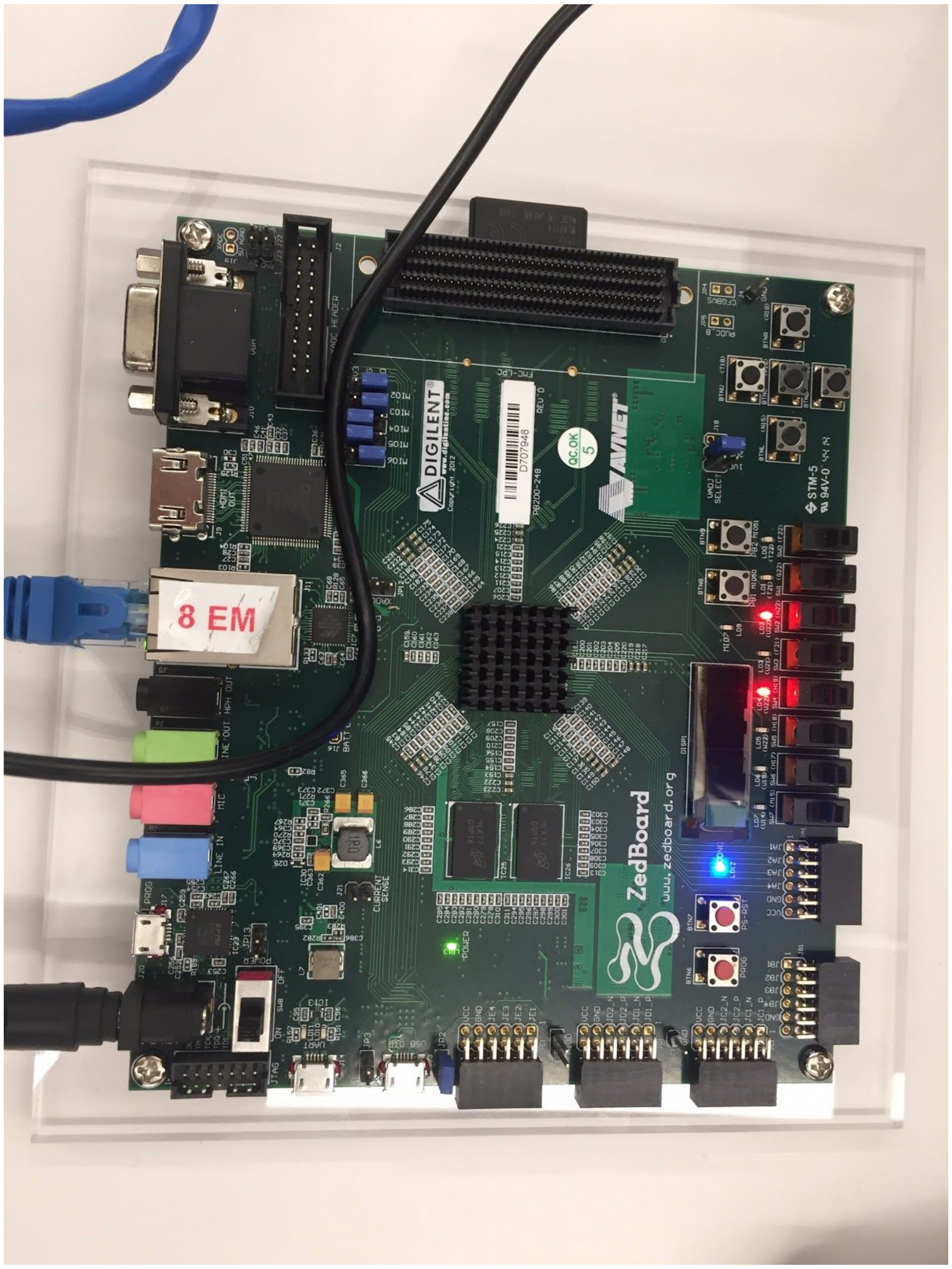
In this section, we wrote a program to control individual LED, what we did was to imitate the use of function *userio_ledSetAll()* which lights up all the LEDs at the same time.



```c
/**
 * Show lower 8 bits of integer value on LEDs
 *@param pBase  base address of userio
 *@param value  value to show on LEDs
 */
void userio_ledSetAll(unsigned char *pBase, int value)
{
    REG_WRITE(pBase, LED1_OFFSET, value%2);
    REG_WRITE(pBase, LED2_OFFSET, (value/2)%2);
    REG_WRITE(pBase, LED3_OFFSET, (value/4)%2);
    REG_WRITE(pBase, LED4_OFFSET, (value/8)%2);
    REG_WRITE(pBase, LED5_OFFSET, (value/16)%2);
    REG_WRITE(pBase, LED6_OFFSET, (value/32)%2);
    REG_WRITE(pBase, LED7_OFFSET, (value/64)%2);
    REG_WRITE(pBase, LED8_OFFSET, (value/128)%2);
}
```

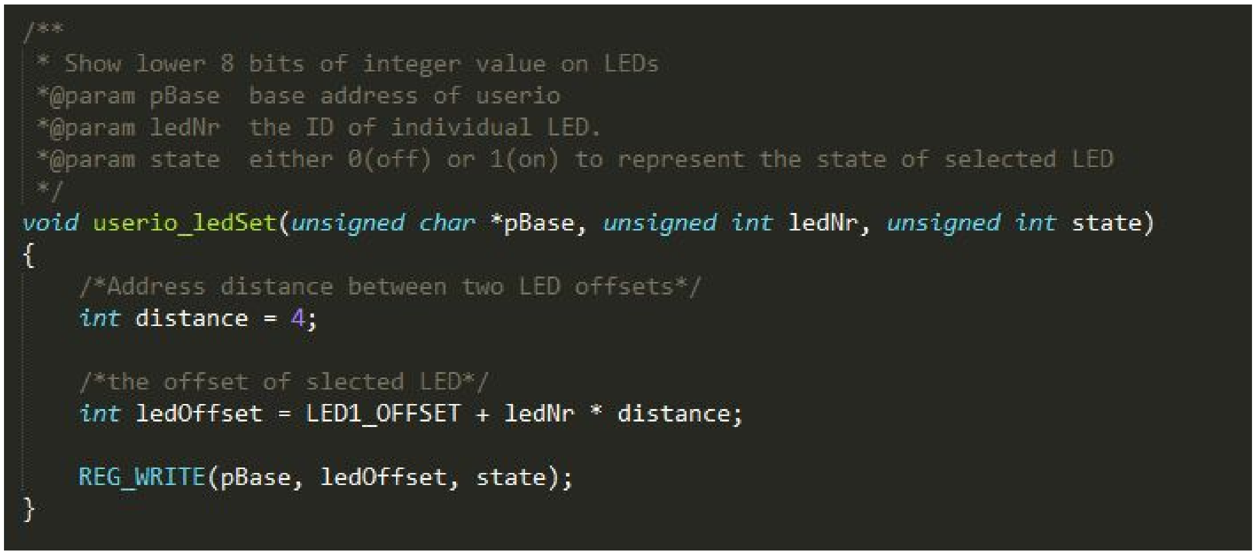
First, we simply test the *userio_ledSetAll()* by compiling and running. Here is the output LED:


8 EM
DIGILENT
AVNET
ZedBoard
www.zedboard.org



We found it just simply doing some some division (by 2, 4, 8, etc) then modulo 2. If the output is 1 the light will be on, 0 for off.

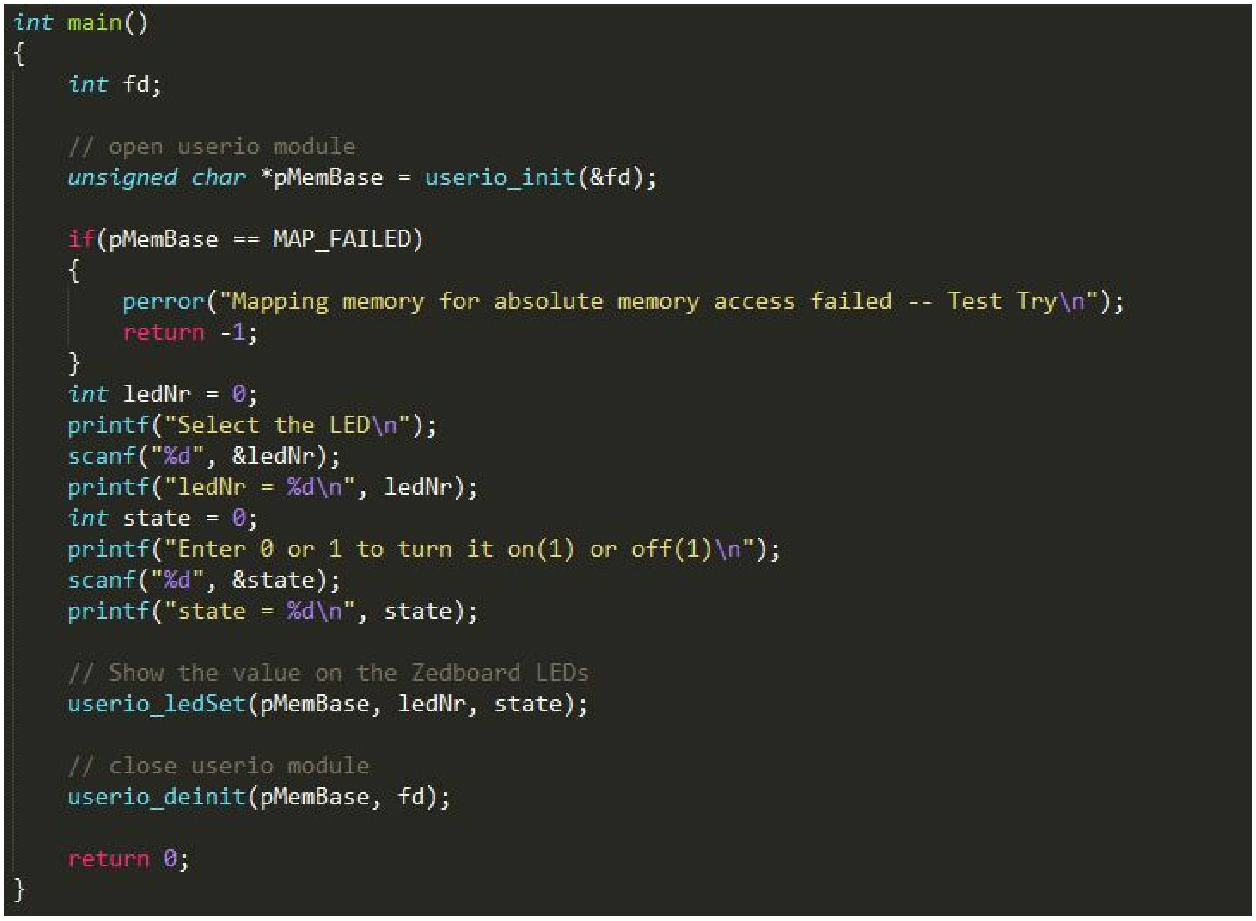
According to it, We wrote a simple function *userio_ledSet()*. It takes the number of the LED we want to light up, and writes to that LED. The full function is as shown:



```
/**
 * Show lower 8 bits of integer value on LEDs
 *@param pBase  base address of userio
 *@param ledNr  the ID of individual LED.
 *@param state  either 0(off) or 1(on) to represent the state of selected LED
 */
void userio_ledSet(unsigned char *pBase, unsigned int ledNr, unsigned int state)
{
    /*Address distance between two LED offsets*/
    int distance = 4;

    /*the offset of slected LED*/
    int ledOffset = LED1_OFFSET + ledNr * distance;

    REG_WRITE(pBase, ledOffset, state);
}
```

This function needs input *pBase which is the base address which has been initiated in function unsigned char *userio_init(int *fd). Another input is ledNwom prompting user to enter in main function. The last input would be state which is either 1(switch on) or 0(switch off) depending on what user enters. This function calculates the address (pBase+Offset) for a certain LED and writes to it using REG_WRITE() to make it light or off. Another key point in this function is the distance between LED addresses. We use led1's address + 4 * the id of led to find the specific led needed to be switched.

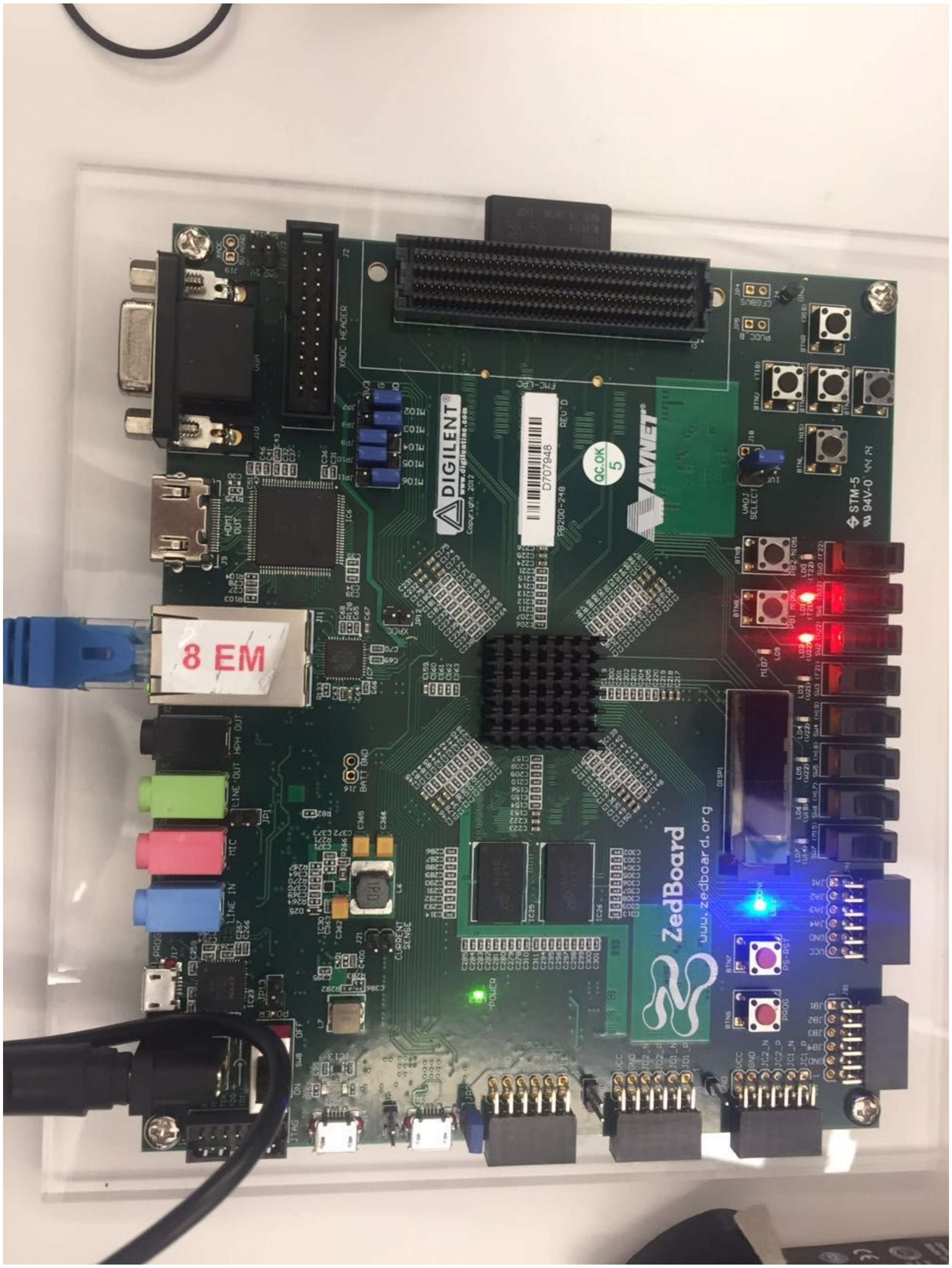
In main function, this program will be utilized in this way:



```c
int main()
{
    int fd;

    // open userio module
    unsigned char *pMemBase = userio_init(&fd);

    if(pMemBase == MAP_FAILED)
    {
        perror("Mapping memory for absolute memory access failed -- Test Try\n");
        return -1;
    }
    int ledNr = 0;
    printf("Select the LED\n");
    scanf("%d", &ledNr);
    printf("ledNr = %d\n", ledNr);
    int state = 0;
    printf("Enter 0 or 1 to turn it on(1) or off(1)\n");
    scanf("%d", &state);
    printf("state = %d\n", state);

    // Show the value on the Zedboard LEDs
    userio_ledSet(pMemBase, ledNr, state);

    // close userio module
    userio_deinit(pMemBase, fd);

    return 0;
}
```

After we ran this program and enter the number of LED we wanted to light up, we found this program worked. Full program is attached to the appendix.

Output LED:

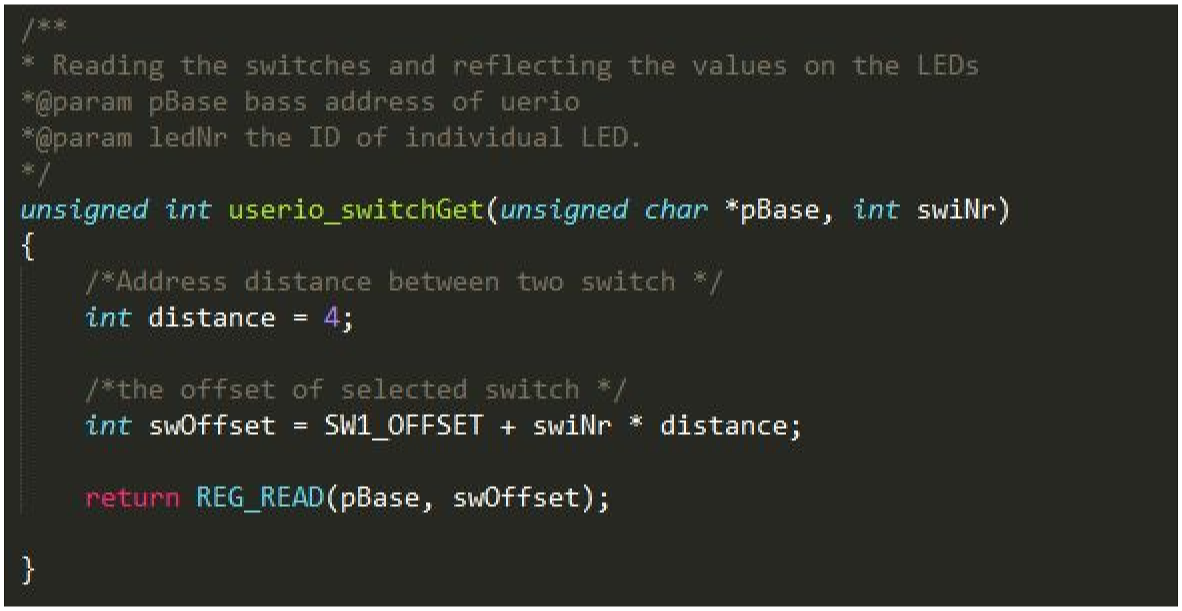


Lab 3.2

We copied program from last section into lab3-SwitchToLed.c using the command: < cp lab3-singleLed.c lab3-SwitchToLed.c>. We copied the userio_SwitchGet() prototype into this program and then build off that. Prototype we wrote is:

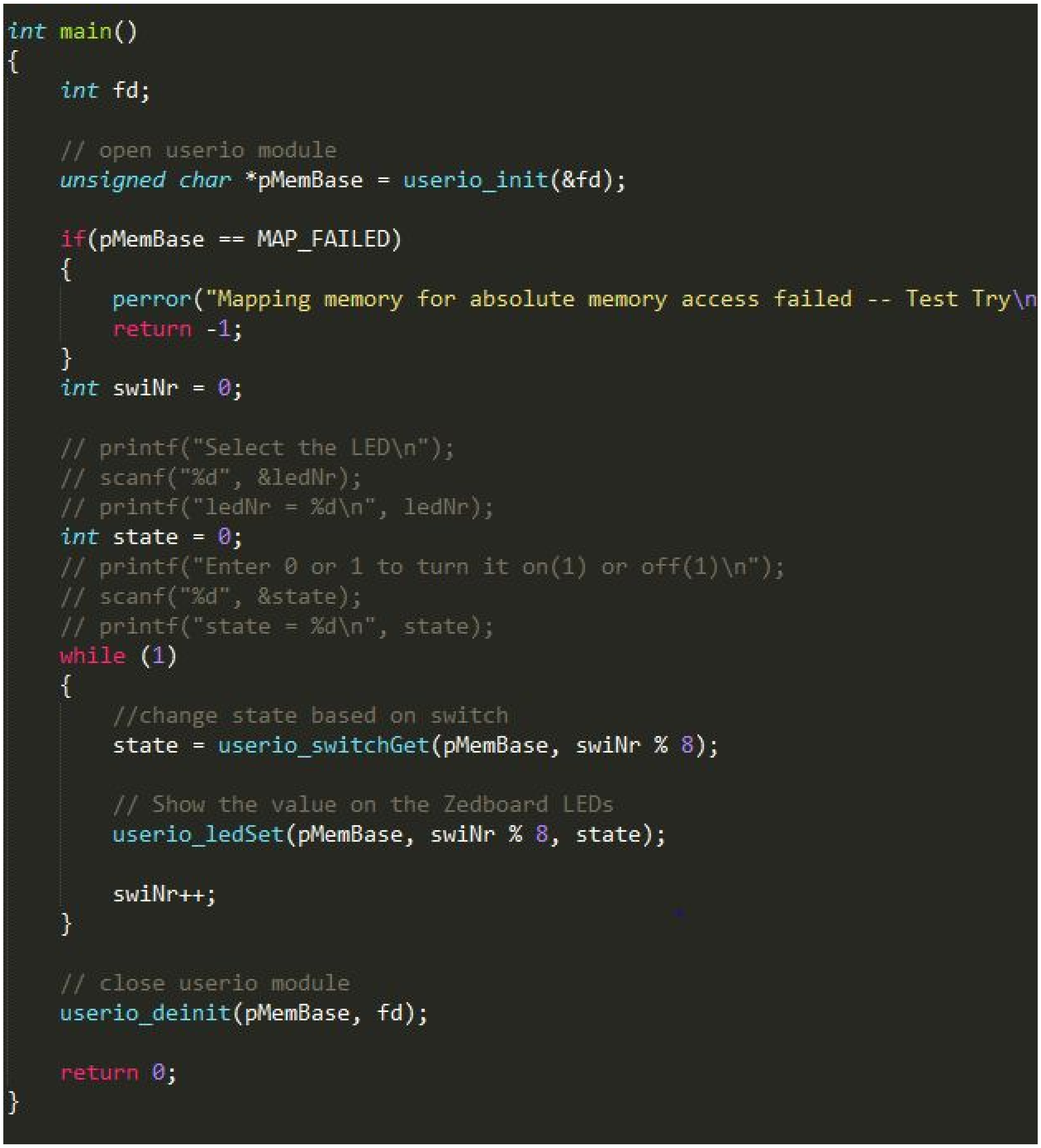
*unsigned int userio_switchGet(unsigned char *pBase, int swiNr)*

The whole function is:



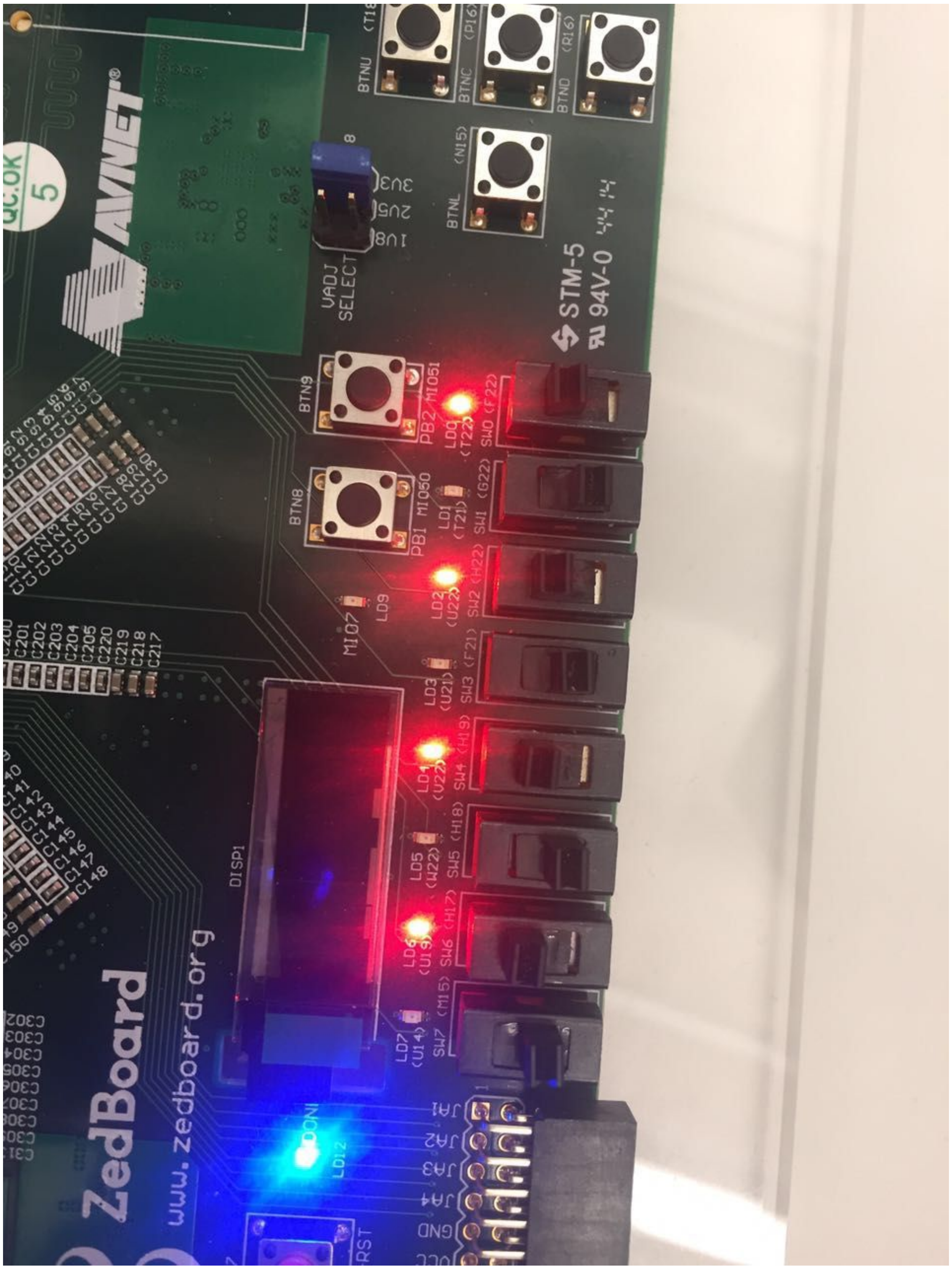
```
/**
* Reading the switches and reflecting the values on the LEDs
*@param pBase bass address of uerio
*@param ledNr the ID of individual LED.
*/
unsigned int userio_switchGet(unsigned char *pBase, int swiNr)
{
    /*Address distance between two switch */
    int distance = 4;

    /*the offset of selected switch */
    int swOffset = SW1_OFFSET + swiNr * distance;

    return REG_READ(pBase, swOffset);

}
```

This function gets input of pBase and switch number which can be read from the function *REG_READ()*. Unlike function *userio_ledSet()*, it will output the state (0 or 1) of a certain switch which can be produced by function *REG_READ(pBase, swOffset).* This function only check single switch, so when implemented in main function, we will use a infinite while loop to check every switch. In the main function, the function is called in this way:



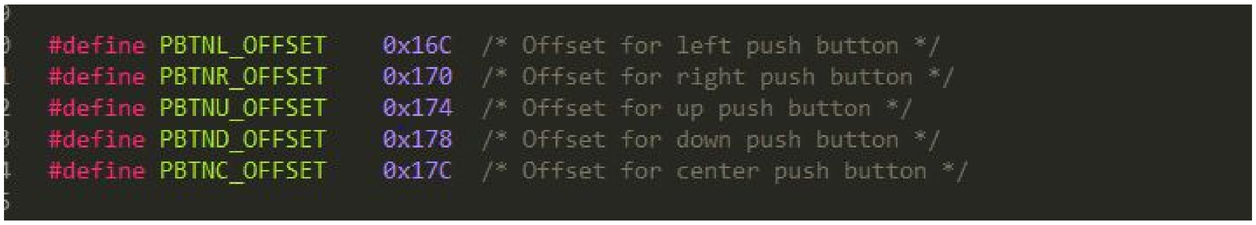
```
int main()
{
    int fd;

    // open userio module
    unsigned char *pMemBase = userio_init(&fd);

    if(pMemBase == MAP_FAILED)
    {
        perror("Mapping memory for absolute memory access failed -- Test Try\n
        return -1;
    }
    int swiNr = 0;

    // printf("Select the LED\n");
    // scanf("%d", &ledNr);
    // printf("ledNr = %d\n", ledNr);
    int state = 0;
    // printf("Enter 0 or 1 to turn it on(1) or off(1)\n");
    // scanf("%d", &state);
    // printf("state = %d\n", state);
    while (1)
    {
        //change state based on switch
        state = userio_switchGet(pMemBase, swiNr % 8);

        // Show the value on the Zedboard LEDs
        userio_ledSet(pMemBase, swiNr % 8, state);

        swiNr++;
    }

    // close userio module
    userio_deinit(pMemBase, fd);

    return 0;
}
```

Loop *While(1)* will always be true which means this is a infinite loop and will check the state of switches over and over again. If one switch was read to be 1, the corresponding light will be turned on. After our multiple tests, we found it worked. Full program is attached to the appendix.
Output LED:


AVNET
ZedBoard
www.zedboard.org
VADJ SELECT
STM-5
94V-0

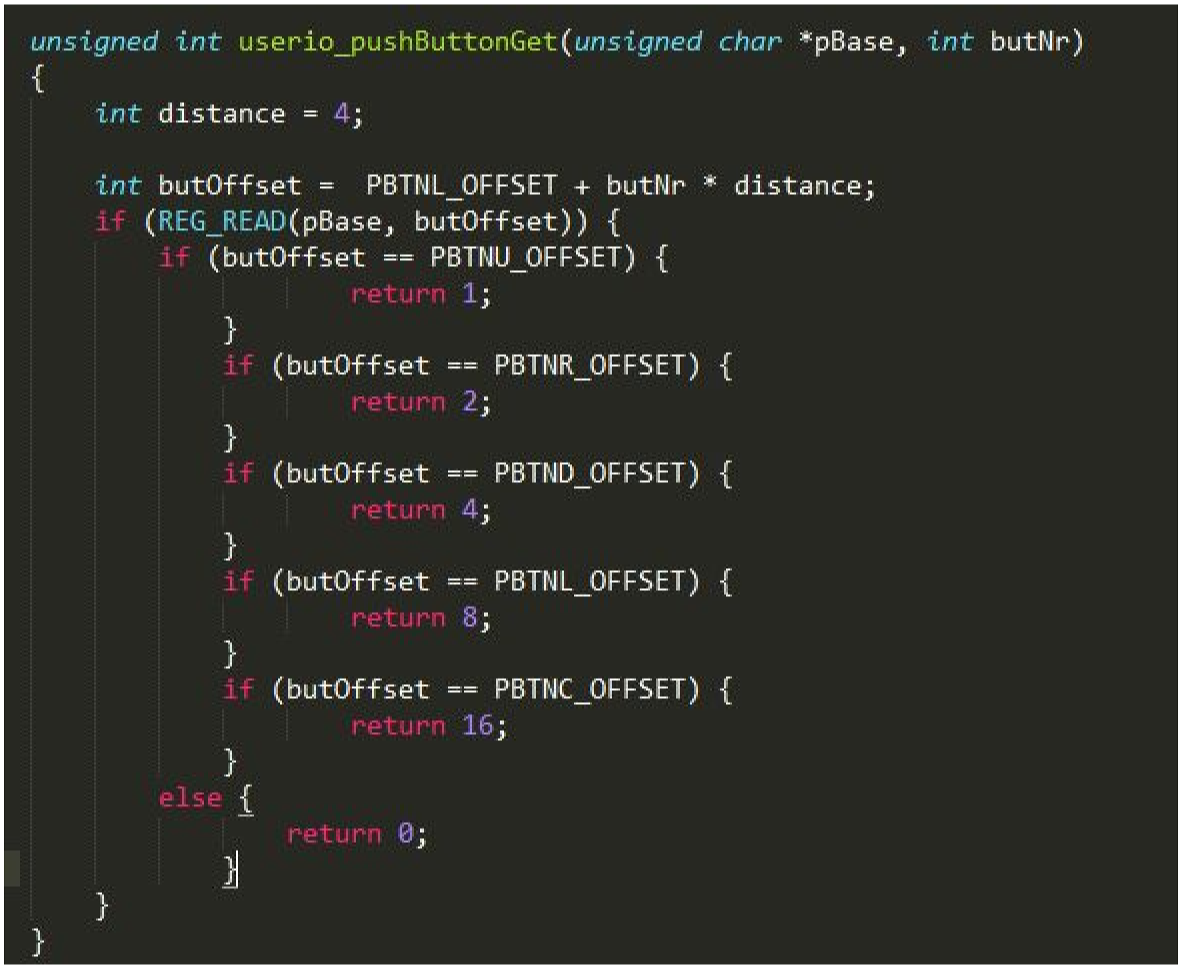
Lab 3.3

This program read from push buttons and prints the number of that button to the interface. We first copied this program to lab3-pushButton.c and built off that. This function should read the state of the push buttons and print the number to the one that is being pushed. Since each push button has a certain number associated with it, we define the address of them into program:



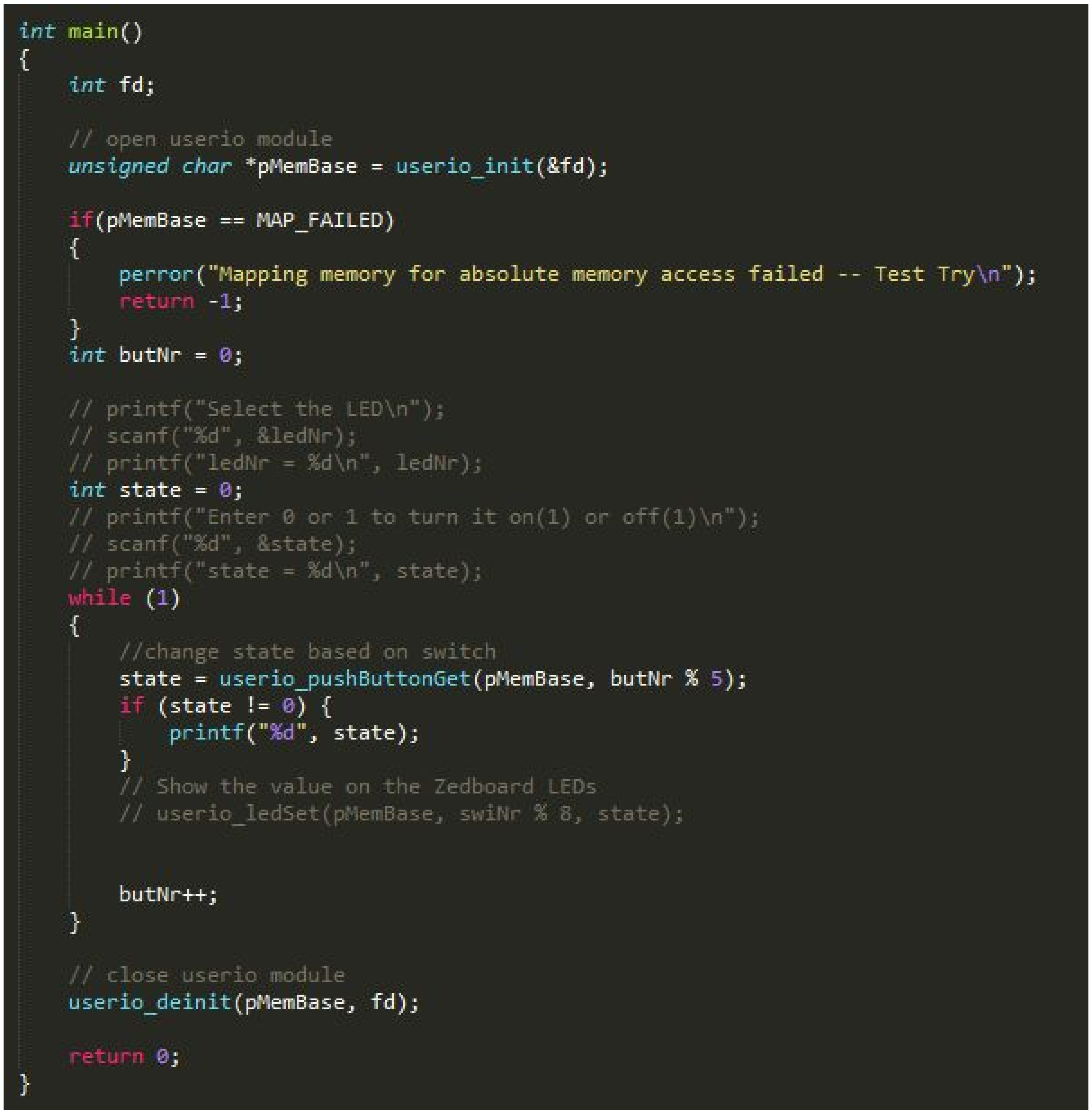
```c
#define PBTNL_OFFSET    0x16C  /* Offset for left push button */
#define PBTNR_OFFSET    0x170  /* Offset for right push button */
#define PBTNU_OFFSET    0x174  /* Offset for up push button */
#define PBTND_OFFSET    0x178  /* Offset for down push button */
#define PBTNC_OFFSET    0x17C  /* Offset for center push button */
```

We used if statements in this function to find the pressed button:



```c
unsigned int userio_pushButtonGet(unsigned char *pBase, int butNr)
{
    int distance = 4;

    int butOffset =  PBTNL_OFFSET + butNr * distance;
    if (REG_READ(pBase, butOffset)) {
        if (butOffset == PBTNU_OFFSET) {
                return 1;
            }
            if (butOffset == PBTNR_OFFSET) {
                return 2;
            }
            if (butOffset == PBTND_OFFSET) {
                return 4;
            }
            if (butOffset == PBTNL_OFFSET) {
                return 8;
            }
            if (butOffset == PBTNC_OFFSET) {
                return 16;
            }
        else {
            return 0;
            }
    }
}
```

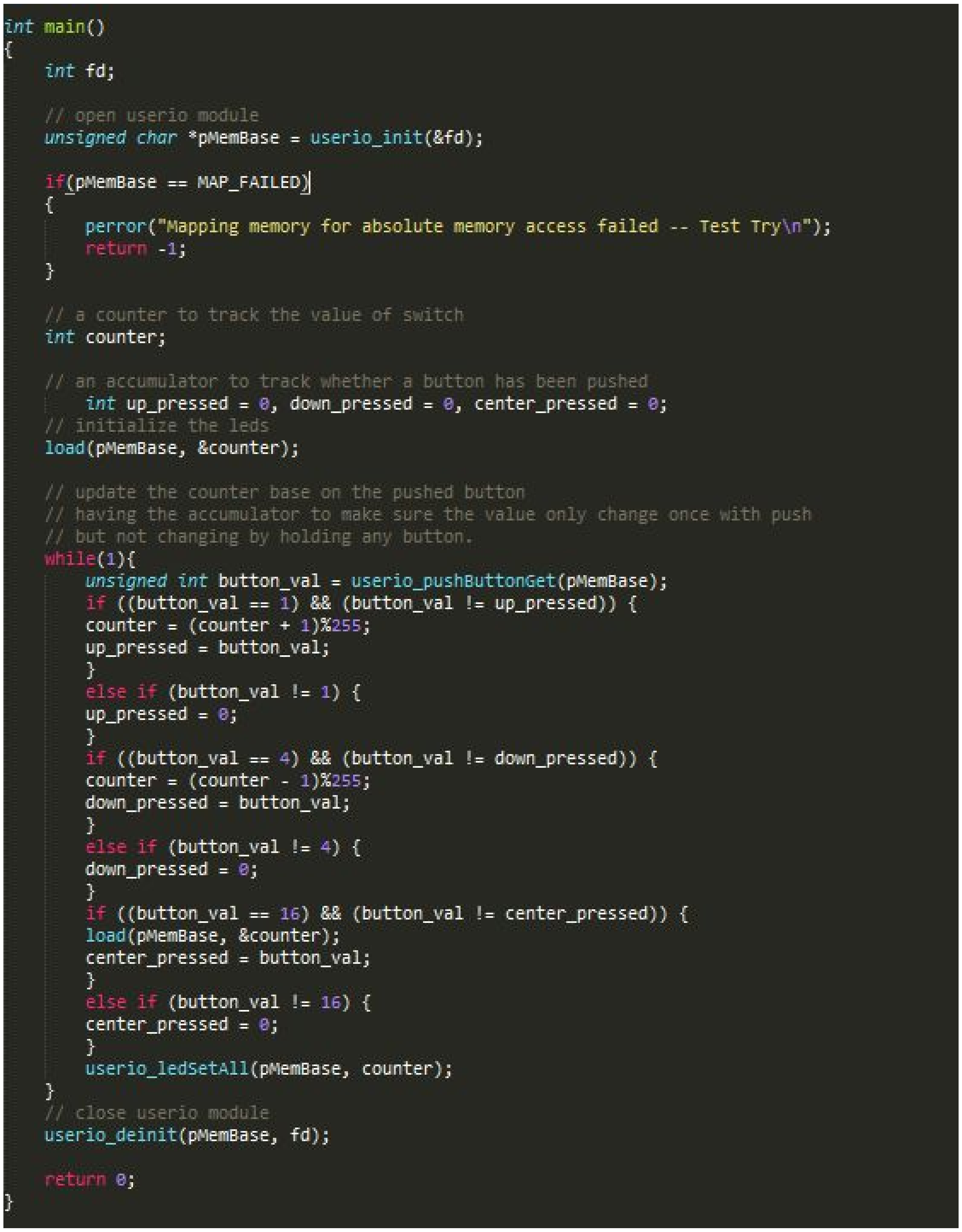
This function gets inputs pBase and button number, and returns the number of button being pushed. First, it will check the state of a certain button by *REG_READ(pBase, butOFFSet)*. If it returns one, then the function checks the address of the button and gives out the number that represent it. Similar to *SwitchGet*, we have to use a while loop to check the state of every button and print it out:



```
int main()
{
    int fd;

    // open userio module
    unsigned char *pMemBase = userio_init(&fd);

    if(pMemBase == MAP_FAILED)
    {
        perror("Mapping memory for absolute memory access failed -- Test Try\n");
        return -1;
    }
    int butNr = 0;

    // printf("Select the LED\n");
    // scanf("%d", &ledNr);
    // printf("ledNr = %d\n", ledNr);
    int state = 0;
    // printf("Enter 0 or 1 to turn it on(1) or off(1)\n");
    // scanf("%d", &state);
    // printf("state = %d\n", state);
    while (1)
    {
        //change state based on switch
        state = userio_pushButtonGet(pMemBase, butNr % 5);
        if (state != 0) {
            printf("%d", state);
        }
        // Show the value on the Zedboard LEDs
        // userio_ledSet(pMemBase, swiNr % 8, state);


        butNr++;
    }

    // close userio module
    userio_deinit(pMemBase, fd);

    return 0;
}
```

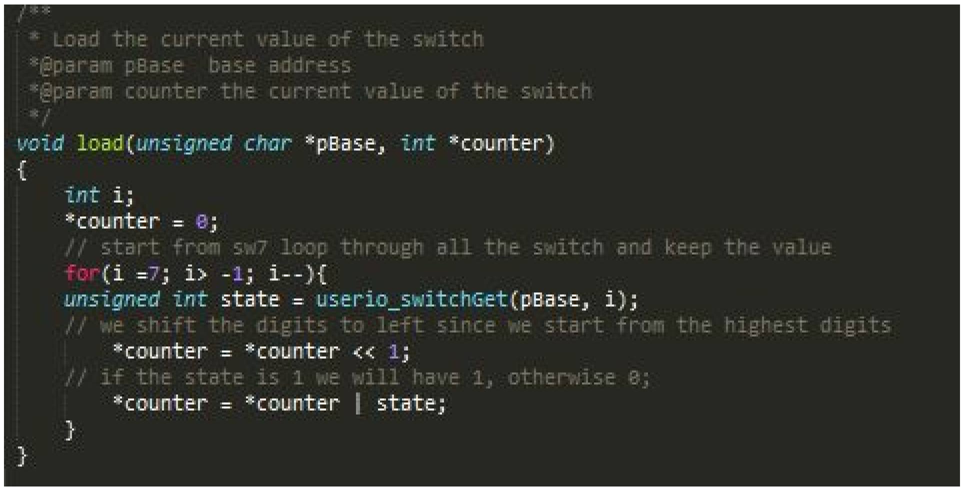
Then we can just simply output the value to the LEDs by applying userio_ledSet instead of printf This program will run infinitely as long as you push some button and print the number of the button we push. We tested it, and it worked. We then used contl-C to exit the program. Full program is attached to the appendix

After that, we implement the required new pushButton.c, which reads the value of switches firstly, then display the counter value on the led by buttons. Here is our changes in main function:



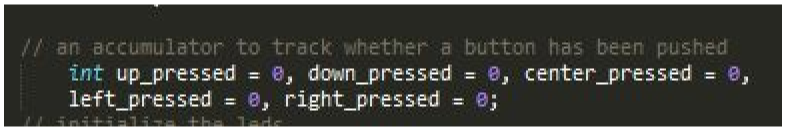
```c
int main()
{
    int fd;

    // open userio module
    unsigned char *pMemBase = userio_init(&fd);

    if(pMemBase == MAP_FAILED)
    {
        perror("Mapping memory for absolute memory access failed -- Test Try\n");
        return -1;
    }

    // a counter to track the value of switch
    int counter;

    // an accumulator to track whether a button has been pushed
        int up_pressed = 0, down_pressed = 0, center_pressed = 0;
    // initialize the leds
    load(pMemBase, &counter);

    // update the counter base on the pushed button
    // having the accumulator to make sure the value only change once with push
    // but not changing by holding any button.
    while(1){
        unsigned int button_val = userio_pushButtonGet(pMemBase);
        if ((button_val == 1) && (button_val != up_pressed)) {
        counter = (counter + 1)%255;
        up_pressed = button_val;
        }
        else if (button_val != 1) {
        up_pressed = 0;
        }
        if ((button_val == 4) && (button_val != down_pressed)) {
        counter = (counter - 1)%255;
        down_pressed = button_val;
        }
        else if (button_val != 4) {
        down_pressed = 0;
        }
        if ((button_val == 16) && (button_val != center_pressed)) {
        load(pMemBase, &counter);
        center_pressed = button_val;
        }
        else if (button_val != 16) {
        center_pressed = 0;
        }
        userio_ledSetAll(pMemBase, counter);
    }
    // close userio module
    userio_deinit(pMemBase, fd);

    return 0;
}
```

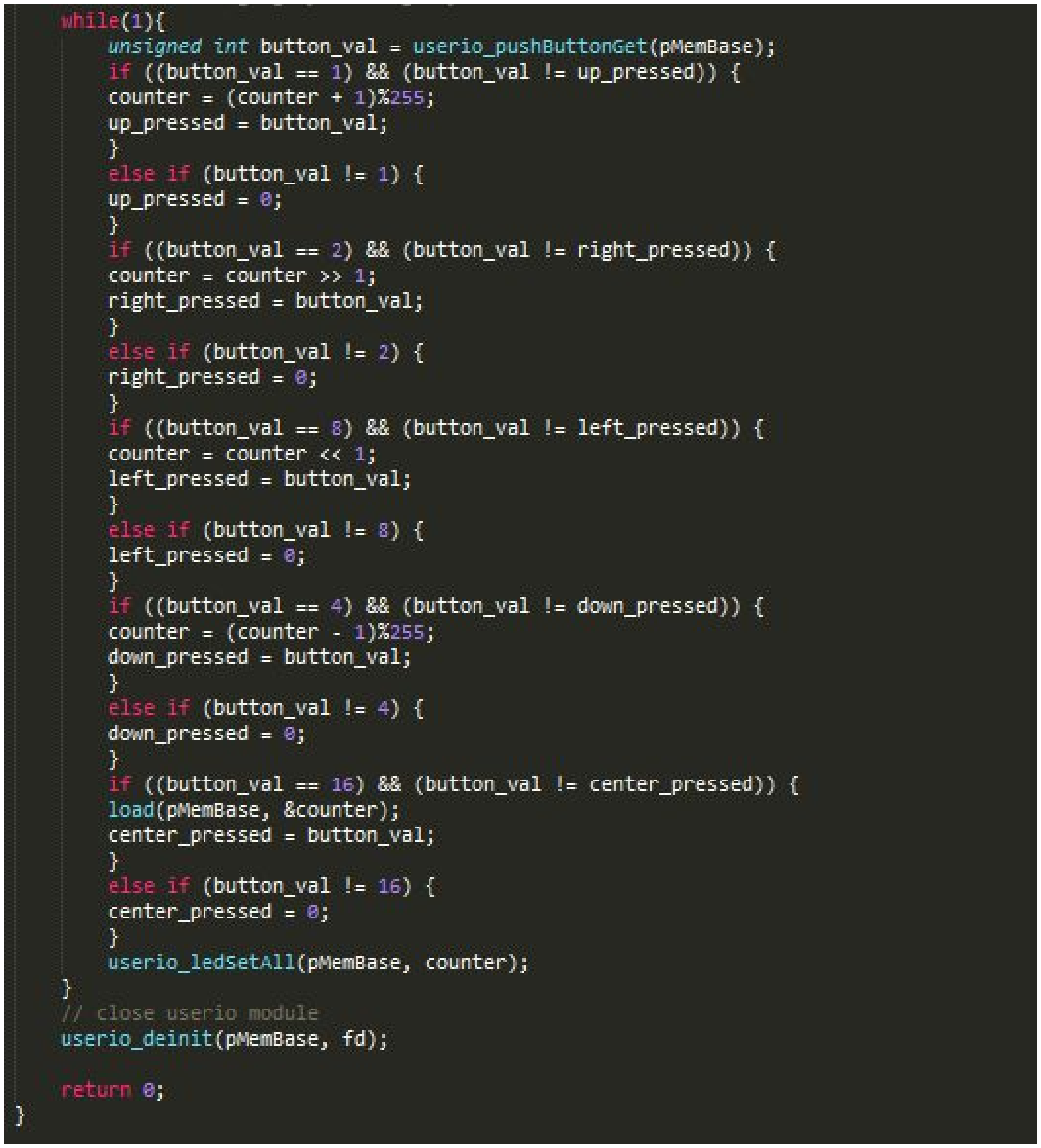
Firstly, we use a counter to track the state of switch. Also we modulo 255 to solve the overflow. Then, we use 3 counters to count up, down and center button. Finally, we can use if cases to determine which button was pushed. Full program is attached to the appendix. And a load function is used to load the state of the switch


```
/**
 * Load the current value of the switch
 *@param pBase  base address
 *@param counter the current value of the switch
 */
void load(unsigned char *pBase, int *counter)
{
    int i;
    *counter = 0;
    // start from sw7 loop through all the switch and keep the value
    for(i =7; i> -1; i--){
    unsigned int state = userio_switchGet(pBase, i);
    // we shift the digits to left since we start from the highest digits
        *counter = *counter << 1;
    // if the state is 1 we will have 1, otherwise 0;
        *counter = *counter | state;
    }
}
```

<u>Lab 3.4</u>

Here we copied the program from pushButton.c and expand two more counters and if cases in main function. We just simply and left and right case to shift the counter:

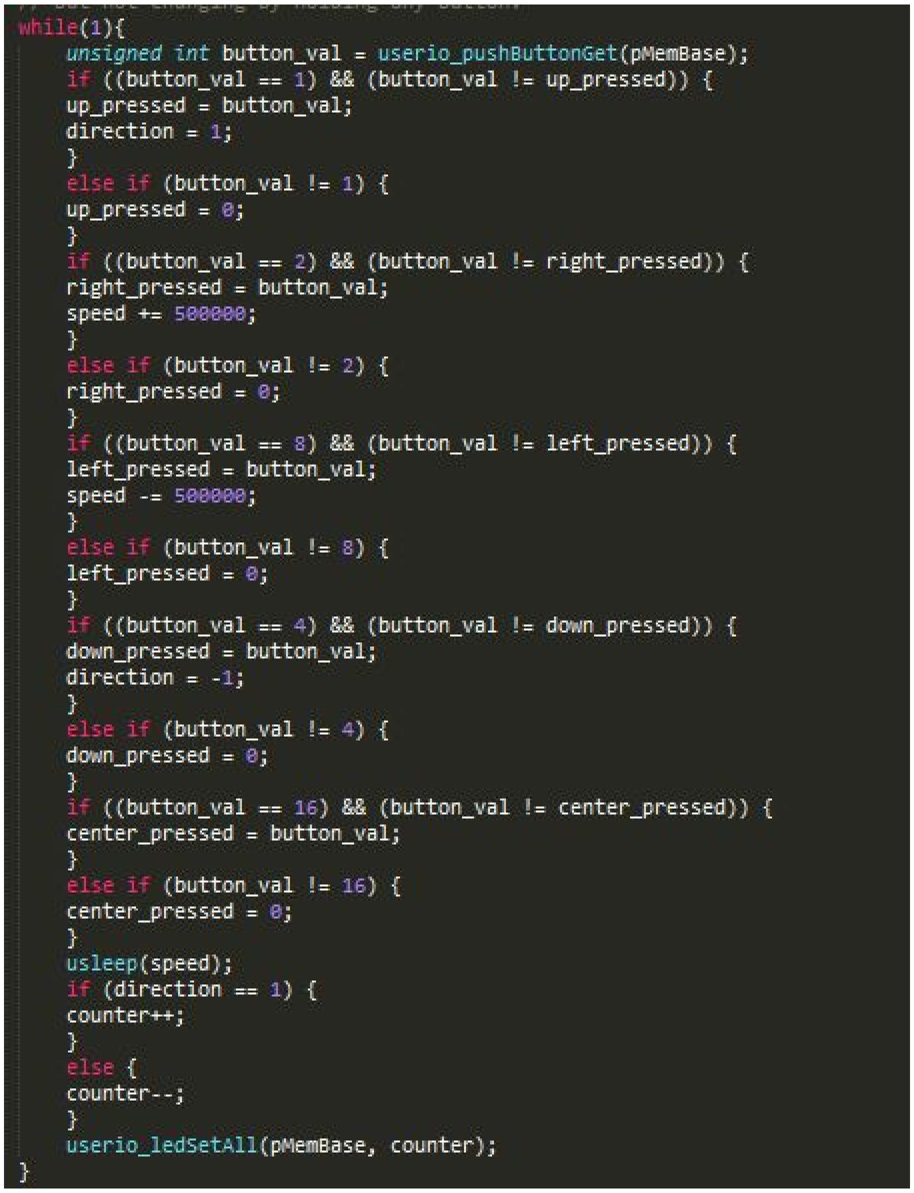

```
// an accumulator to track whether a button has been pushed
    int up_pressed = 0, down_pressed = 0, center_pressed = 0,
    left_pressed = 0, right_pressed = 0;
```



```
    while(1){
        unsigned int button_val = userio_pushButtonGet(pMemBase);
        if ((button_val == 1) && (button_val != up_pressed)) {
        counter = (counter + 1)%255;
        up_pressed = button_val;
        }
        else if (button_val != 1) {
        up_pressed = 0;
        }
        if ((button_val == 2) && (button_val != right_pressed)) {
        counter = counter >> 1;
        right_pressed = button_val;
        }
        else if (button_val != 2) {
        right_pressed = 0;
        }
        if ((button_val == 8) && (button_val != left_pressed)) {
        counter = counter << 1;
        left_pressed = button_val;
        }
        else if (button_val != 8) {
        left_pressed = 0;
        }
        if ((button_val == 4) && (button_val != down_pressed)) {
        counter = (counter - 1)%255;
        down_pressed = button_val;
        }
        else if (button_val != 4) {
        down_pressed = 0;
        }
        if ((button_val == 16) && (button_val != center_pressed)) {
        load(pMemBase, &counter);
        center_pressed = button_val;
        }
        else if (button_val != 16) {
        center_pressed = 0;
        }
        userio_ledSetAll(pMemBase, counter);
    }
    // close userio module
    userio_deinit(pMemBase, fd);

    return 0;
}
```

3.5

copied the program from pushButton.c to pushCounter.c. In this program, we didn't change much. The only changes are in while loop of main function. Instead of count by pushing, we make it auto count by changing the speed. Speed and direction are two new counter to count speed and directions (-1 or 1). As required, we used the up and down button to control the directions, left and right to decrease or increase the speed. The counter changes automatically by different speed and direction:



```
while(1){
    unsigned int button_val = userio_pushButtonGet(pMemBase);
    if ((button_val == 1) && (button_val != up_pressed)) {
    up_pressed = button_val;
    direction = 1;
    }
    else if (button_val != 1) {
    up_pressed = 0;
    }
    if ((button_val == 2) && (button_val != right_pressed)) {
    right_pressed = button_val;
    speed += 500000;
    }
    else if (button_val != 2) {
    right_pressed = 0;
    }
    if ((button_val == 8) && (button_val != left_pressed)) {
    left_pressed = button_val;
    speed -= 500000;
    }
    else if (button_val != 8) {
    left_pressed = 0;
    }
    if ((button_val == 4) && (button_val != down_pressed)) {
    down_pressed = button_val;
    direction = -1;
    }
    else if (button_val != 4) {
    down_pressed = 0;
    }
    if ((button_val == 16) && (button_val != center_pressed)) {
    center_pressed = button_val;
    }
    else if (button_val != 16) {
    center_pressed = 0;
    }
    usleep(speed);
    if (direction == 1) {
    counter++;
    }
    else {
    counter--;
    }
    userio_ledSetAll(pMemBase, counter);
}
```

Extra Credit

I think the most important part is the if cases in main function. If multiple buttons are pushed, more complex case will be applied. For example, if we press both buttons “4” and “16”. The output should be 4+ 16 = 20 or even more complex math. Therefore, we need to write more cases. Same for the buttonGet function, we also need more cases to handle it.