Lab Assignment 3

Memory-Mapped I/O and
Object-Oriented Programming

Tianqi Li
Email Address: li.tianq@northeastern.edu

Submit date: 8/2/2020
Due Date: 7/31/2020

Abstract

In this lab, we learned the concept of memory-mapped I/O to access devices available on the DE1-SoC, including the LEDs, the switches, and the push buttons. We learned how to control the state of the LEDs through give input or change the switches (turn on/off) and the push key buttons (move left/right and up/down the value to change the LEDs state).

Introduction

Using memory mapped I/O in the device DE1-SoC. we can read or write the state of a certain I/O at a given address. For example (const unsigned int LEDR_BASE = 0x00000000;

const unsigned int SW_BASE = 0x00000040;

const unsigned int KEY_BASE = 0x00000050;)

The LEDR_BASE is an address for whole LEDs state, SW_BASE is for whole switches and the KEY_BASE is for the whole key push buttons states.

We write the functions for changing their state.

Lab Discussion

Computer system: Window 10

DE1-SoC board

One 32 GB Micro SD Card (Insert in DE1-SoC board)

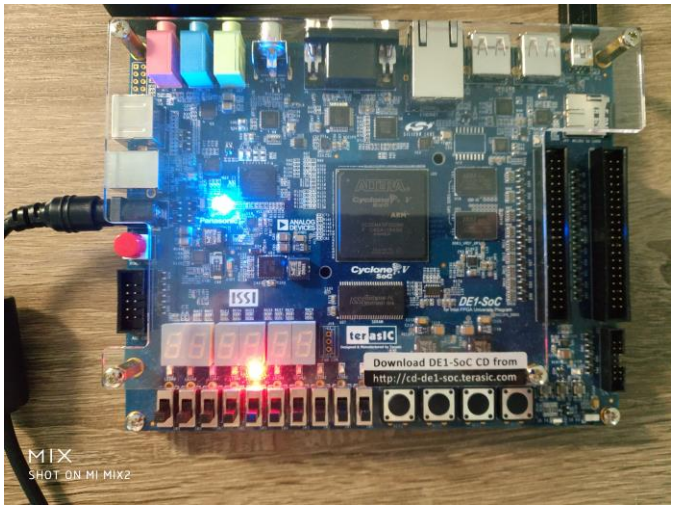
One 64 GB USB flash drive (write with the Win32DiskImager)

Results and Analysis

Lab 3.1 Lab Assignment, Lab 3.2 Controlling all LEDs and Lab 3.3 Controlling all the switches (I include all thing together we ran the LedNumber)

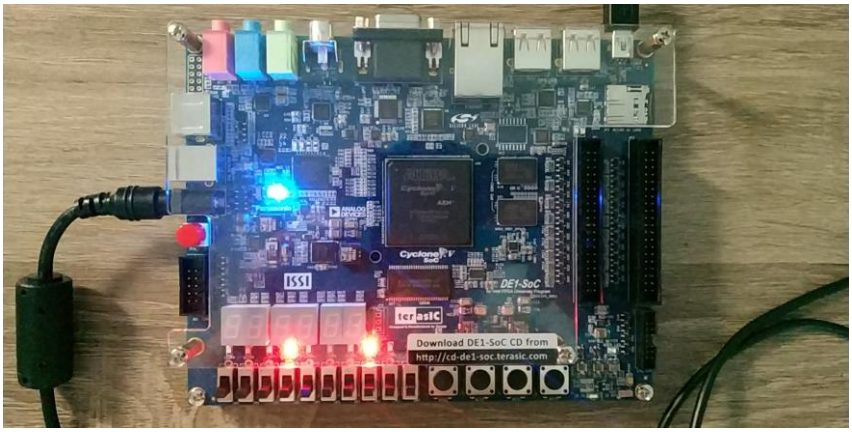
```
Type a ledNum between 0 to 9: 5
Type 0 to close the led : 0
Type a value between 0 to 1023: 68
Type 1 to continue once you move well with the switch: 1
298
Type a switchNum to read between 0 to 9 (1 is on, 0 is off): 4
0
```

We ran the Write1Led to open the led5 (the state 1 is open),



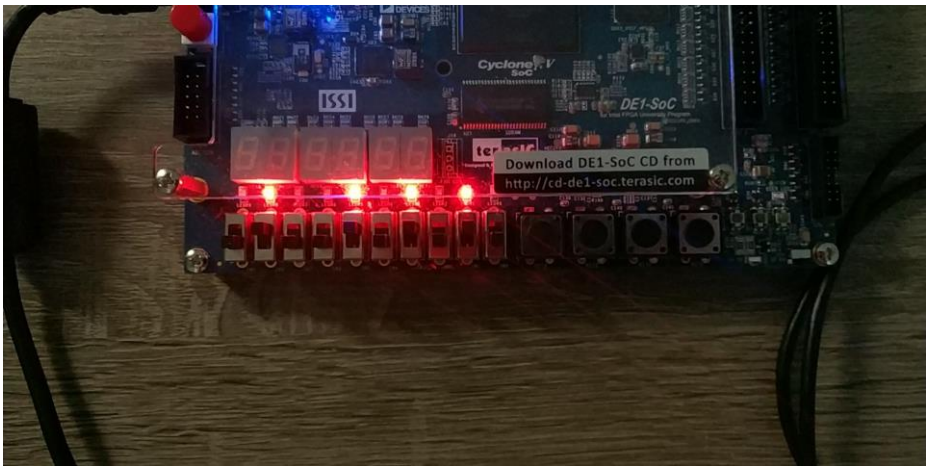
Then we Type 0 close it **(the state 0 is close in Write1Led function, this part is test for the Write1Led function in lab 3.1)**

Then we enter 68 as a value to run WriteALLLeds



It show 0001000100 which is 68 = (64 + 4) = 2^6 + 2^2 **(this part is testing for WriteAllLeds function in lab 3.2)**

Then we switch up the1, 3, 5, 8 led button, and type 1 to run with ReadAllLeds and using the result for ReadAllLeds ran the WriteAllLeds to open the Leds.



It shows the 0100101010 (1,3.5,8) we turn on the led with the value given by ReadAllLeds function **(this part is testing ReadAllLeds and WriteAllLeds functions which is in lab 3.2 and lab 3.3)**

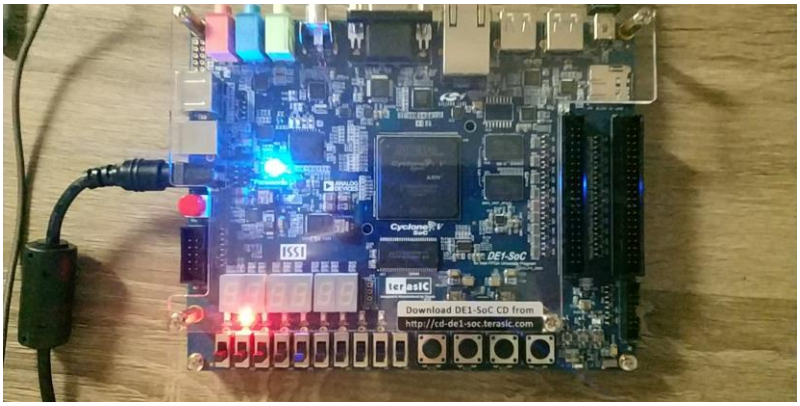
Finally, we used Read1Switch to read the state, we type 4 (show in the last part of terminal picture) it output 0 which means the LED4 was turn off, as above, we just opened LED 1, 3, 5, 8 led. So that the LED4 ‘s state was 0 or turned off. **(this part is test Read1Switch function which is in lab 3.1)**

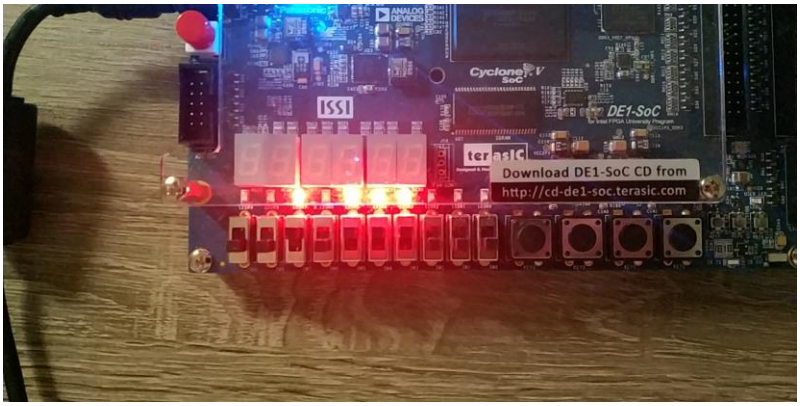
There are another 2 similar tests for the LedNumber.cpp (include lab3.1.3.2,3.3)

It will show in here,

Test 2:

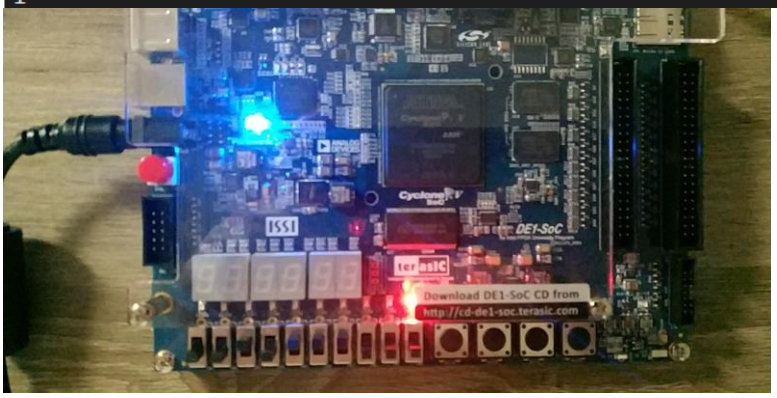
```
root@de1soclinux:~/Desktop/lab3# ./LedNumber
Type a ledNum between 0 to 9: 3
Type 0 to close the led : 0
Type a value between 0 to 1023: 256
Type 1 to continue once you move well with the switch: 1
184
Type a switchNum to read between 0 to 9 (1 is on, 0 is off): 6
0
```

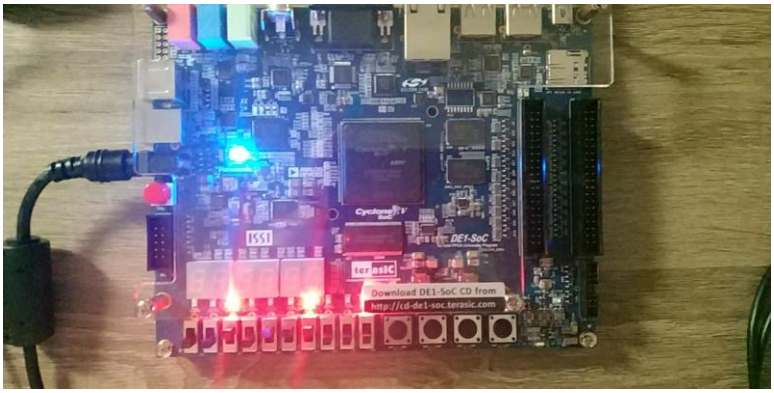




Test 3:

```
root@de1soclinux:~/Desktop/lab3# ./LedNumber
Type a ledNum between 0 to 9: 0
Type 0 to close the led : 0
Type a value between 0 to 1023: 137
Type 1 to continue once you move well with the switch: 1
679
Type a switchNum to read between 0 to 9 (1 is on, 0 is off): 2
1
```





Lab 3.4 Controlling the push buttons

There is a video to show the LED change (here is the output of terminal)

```
root@de1soclinux:~/Desktop/lab3# ./PushButtonClass
Tpying 1 if you are switch well the value with the button: 1
New buttons: 0000010100
New value: 20
New buttons: 0000101000
New value: 40
New buttons: 0001010000
New value: 80
New buttons: 0001001111
New value: 79
New buttons: 0000100111
New value: 39
New buttons: 0000010011
New value: 19
New buttons: 0000100110
New value: 38
New buttons: 0000100111
New value: 39
New buttons: 0000101000
New value: 40
New buttons: 0000100111
New value: 39
New buttons: 0000010011
New value: 19
New buttons: 0000100110
New value: 38
New buttons: 0001001100
New value: 76
New buttons: 0010011000
New value: 152
New buttons: 0010011001
New value: 153
New buttons: 0010011010
New value: 154
New buttons: 0010011001
New value: 153
New buttons: 0001001100
New value: 76
New buttons: 0000100110
New value: 38
New buttons: 0000010011
New value: 19
New buttons: 0000100110
New value: 38
```

```
New buttons: 0001001100
New value: 76
New buttons: 0010011000
New value: 152
New buttons: 0100110000
New value: 304
New buttons: 1001100000
New value: 608
New buttons: 0011000000
New value: 192
New buttons: 0110000000
New value: 384
New buttons: 0101111111
New value: 383
New buttons: 0010111111
New value: 191
New buttons: 0001011111
New value: 95
New buttons: 0001011110
New value: 94
New buttons: 0001011101
New value: 93
New buttons: 0001011100
New value: 92
New buttons: 0001011101
New value: 93
New buttons: 0001011100
New value: 92
New buttons: 0000101110
New value: 46
New buttons: 0000010111
New value: 23
New buttons: 0000001011
New value: 11
```

Lab 3.5 Using C++ objects

All code is show in the Appendix

Conclusion

In this lab, we mainly learned how to use the memory-mapped I/O to access devices available on the DE1-SoC, including the LEDs, the switches, and the push buttons. We learned how to control the state of the LEDs through give input or change the switches (turn on/off) and the push key buttons (move left/right and up/down the value to change the LEDs state) through function.

References

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

Appendix

LedNumber.cpp

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

using namespace std;

// Physical base address of FPGA Devices
const unsigned int LW_BRIDGE_BASE = 0xFF200000; // Base offset

// Length of memory-mapped IO window
const unsigned int LW_BRIDGE_SPAN = 0x00005000; // Address map size

// Cyclone V FPGA device addresses
const unsigned int LEDR_BASE = 0x00000000; // Leds offset
const unsigned int SW_BASE = 0x00000040; // Switches offset
const unsigned int KEY_BASE = 0x00000050; // Push buttons offset

/**
* 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.
* @param value  Value to be written.
*/
void RegisterWrite(char *pBase, unsigned int reg_offset, int value)
{
  * (volatile unsigned int *)(pBase + reg_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, unsigned int reg_offset)
{
  return * (volatile unsigned int *)(pBase + reg_offset);
}

/**
* Initialize general-purpose I/O
* - Opens access to physical memory /dev/mem
* - Maps memory 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)
{
  // Open /dev/mem to give access to physical addresses
  *fd = open( "/dev/mem", (O_RDWR | O_SYNC));
  if (*fd == -1) // check for errors in openning /dev/mem
  {
    cout << "ERROR: could not open /dev/mem..." << endl;
  }
  // Get a mapping from physical addresses to virtual addresses
  char *virtual_base = (char *)mmap (NULL, LW_BRIDGE_SPAN, (PROT_READ | PROT_WRITE),
MAP_SHARED, *fd, LW_BRIDGE_BASE);
  if (virtual_base == MAP_FAILED) // check for errors
  {
    cout << "ERROR: mmap() failed..." << endl;
    close (*fd); // close memory before exiting
    exit(1); // Returns 1 to the operating system;
  }
  return virtual_base;
}
```

```
/**
* 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)
{
  if (munmap (pBase, LW_BRIDGE_SPAN) != 0){
    cout << "ERROR: munmap() failed..." << endl;
    exit(1);
  }
  close (fd); // close memory
}


void Write1Led(char *pBase, int ledNum, int state) {
   if(0 <= ledNum <= 9) {
     int value = 1;
     int count = ledNum;
     if(state == 1) {
       while(count > 0) {
         value = 2 * value;
         count --;
       }
       RegisterWrite(pBase, LEDR_BASE, value);
     }
     else if(state == 0){
       RegisterWrite(pBase, LEDR_BASE, 0);
     }
     else{
       cout << "ERROR: state should only be 0 and 1 means off and on" << endl;
     }
  }
  else {
    cout << "ERROR: ledNum should be in 0-9" << endl;
  }
}
```

```
int Read1Switch(char *pBase, int switchNum) {
  if(0 <= switchNum <= 9) {
    int value = RegisterRead(pBase, SW_BASE);
    int state;
    while (switchNum >= 0){
      state = value % 2;
      value = (value - state) / 2;
      switchNum--;
    }
    return state;
  }
  else{
    cout << "ERROR: switchNum should be in 0-9" << endl;
  }
}


void WriteAllLeds(char *pBase, int value){
//  1111111111 is 1023 means all turn on
  if(0 <= value <= 1023){
    RegisterWrite(pBase, LEDR_BASE, value);
  }
  else{
    cout << "out rage" << endl;
  }
}


int ReadAllSwitches(char *pBase){
  //bitset<10> bits;
  //for (int i = 0; i < 10; i++) {
  //  bits.set(i, Read1Switch(pBase, i));
  //}
  //int result = (int)(bits.to_ulong());
  //return result;
  return RegisterRead(pBase, SW_BASE);

}
```

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

  int ledNum;
  cout << "Type a ledNum between 0 to 9: ";
  cin >> ledNum;
  Write1Led(pBase, ledNum, 1);

  int close;
  cout << "Type 0 to close the led : ";
  cin >> close;
  if(close == 0){
    Write1Led(pBase, ledNum, 0);
  }

  int value;
  cout << "Type a value between 0 to 1023: ";
  cin >> value;

  WriteAllLeds(pBase, value);
  int option;
  cout << "Type 1 to continue once you move well with the switch: ";
  cin >> option;
  if(option == 1){
    cout << ReadAllSwitches(pBase) << endl;
    value = ReadAllSwitches(pBase);
    WriteAllLeds(pBase, value);

    int switch1;
    cout << "Type a switchNum to read between 0 to 9 (1 is on, 0 is off): ";
    cin >> switch1;
    cout << Read1Switch(pBase, switch1) << endl;
  }
  else{
    cout << "fail to continue" << endl;
  }
// Done
  Finalize(pBase, fd);
}
```

PushButtonClass.cpp

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

using namespace std;

// Physical base address of FPGA Devices
const unsigned int LW_BRIDGE_BASE = 0xFF200000; // Base offset

// Length of memory-mapped IO window
const unsigned int LW_BRIDGE_SPAN = 0x00005000; // Address map size

// Cyclone V FPGA device addresses
const unsigned int LEDR_BASE = 0x00000000; // Leds offset
const unsigned int SW_BASE = 0x00000040; // Switches offset
const unsigned int KEY_BASE = 0x00000050; // Push buttons offset



/**
* 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.
* @param value  Value to be written.
*/
void RegisterWrite(char *pBase, unsigned int reg_offset, int value)
{
  * (volatile unsigned int *)(pBase + reg_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, unsigned int reg_offset)
{
  return * (volatile unsigned int *)(pBase + reg_offset);
}

/**
* Initialize general-purpose I/O
* - Opens access to physical memory /dev/mem
* - Maps memory 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)
{
  // Open /dev/mem to give access to physical addresses
  *fd = open( "/dev/mem", (O_RDWR | O_SYNC));
  if (*fd == -1) // check for errors in openning /dev/mem
  {
    cout << "ERROR: could not open /dev/mem..." << endl;
  }
  // Get a mapping from physical addresses to virtual addresses
  char *virtual_base = (char *)mmap (NULL, LW_BRIDGE_SPAN, (PROT_READ | PROT_WRITE),
MAP_SHARED, *fd, LW_BRIDGE_BASE);
  if (virtual_base == MAP_FAILED) // check for errors
  {
    cout << "ERROR: mmap() failed..." << endl;
    close (*fd); // close memory before exiting
    exit(1); // Returns 1 to the operating system;
  }
  return virtual_base;
}

/**
* 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)
{
  if (munmap (pBase, LW_BRIDGE_SPAN) != 0){
    cout << "ERROR: munmap() failed..." << endl;
    exit(1);
  }
  close (fd); // close memory
}

void Write1Led(char *pBase, int ledNum, int state) {
  if(0 <= ledNum <= 9) {
     int value = 1;
     int count = ledNum;
     if(state == 1) {
       while(count > 0) {
         value = 2 * value;
         count --;
       }
       RegisterWrite(pBase, LEDR_BASE, value);
     }
     else if(state == 0){
       RegisterWrite(pBase, LEDR_BASE, 0);
     }
     else{
       cout << "ERROR: state should only be 0 and 1 means off and on" << endl;
     }
  }
  else {
    cout << "ERROR: ledNum should be in 0-9" << endl;
  }
}

int Read1Switch(char *pBase, int switchNum) {
  if(0 <= switchNum <= 9) {
    int value = RegisterRead(pBase, SW_BASE);
    int state;
    while (switchNum >= 0){
      state = value % 2;
      value = (value - state) / 2;
```

```
    switchNum--;
   }
   return state;
  }
  else{
   cout << "ERROR: switchNum should be in 0-9" << endl;
  }

}


void WriteAllLeds(char *pBase, int value){
//  1111111111 is 1023 means all turn on
  if(0 <= value <= 1023){
   RegisterWrite(pBase, LEDR_BASE, value);
  }
  else{
   cout << "out rage" << endl;
  }

}

int ReadAllSwitches(char *pBase){
  //bitset<10> bits;
  //for (int i = 0; i < 10; i++) {
  //  bits.set(i, Read1Switch(pBase, i));
  //}
  //int result = (int)(bits.to_ulong());
  //return result;
  return RegisterRead(pBase, SW_BASE);
}


int PushButtonGet(char *pBase) {
   if(RegisterRead(pBase, KEY_BASE) == 0){
    return -1;
   }
   if(RegisterRead(pBase, KEY_BASE) == 1) {
    return 0;
   }
   else if(RegisterRead(pBase, KEY_BASE) == 2) {
        return 1;
```

```
  }
  else if(RegisterRead(pBase, KEY_BASE) == 4) {
        return 2;
  }
  else if(RegisterRead(pBase, KEY_BASE) == 8) {
        return 3;
  }
  //push more than one key buttons
  else {
    return 4;
  }
}

int ChangeValue(int pushButtonValue, int value){
  bitset<10> buttons(value);
  if(pushButtonValue == 0){
      sleep(1);
      value = value + 1;
      bitset<10> newB(value);
      cout << "New buttons: " << newB  << endl;
      cout << "New value: " << value << endl;
      return value;
  }
  else if(pushButtonValue == 1){
      sleep(1);
      value = value - 1;
      bitset<10> newB(value);
      cout << "New buttons: " << newB  << endl;
      cout << "New value: " << value << endl;
      return value;

  }
  else if(pushButtonValue == 2){
     sleep(1);
     buttons >>= 1;
     cout << "New buttons: " << buttons  << endl;
     int temp = (int)(buttons.to_ulong());
     cout << "New value: " << temp << endl;
     return temp;
  }
  else if(pushButtonValue == 3){
    sleep(1);
```

```
    buttons <<= 1;
    cout << "New buttons: " << buttons << endl;
    int temp = (int)(buttons.to_ulong());
    cout << "New value: " << temp << endl;
    return temp;
  }
  else if(pushButtonValue == 4){
    sleep(1);
    return value;
  }
}

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

  int option;
  cout << "Tpying 1 if you are switch well the value with the button: ";
  cin >> option;
  if(option == 1){
    int value = ReadAllSwitches(pBase);
    WriteAllLeds(pBase, value);
    int temp = value;
    while(true){
      int pushButtonValue = PushButtonGet(pBase);
      if(pushButtonValue != -1) {
        temp = ChangeValue(pushButtonValue, temp);
      }
      WriteAllLeds(pBase, temp);
    }
  }
  else{
    cout << "fail to continue" << endl;
  }
Finalize(pBase, fd);
}
```

PushButton.cpp

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

using namespace std;

// Physical base address of FPGA Devices
const unsigned int LW_BRIDGE_BASE = 0xFF200000; // Base offset

// Length of memory-mapped IO window
const unsigned int LW_BRIDGE_SPAN = 0x00005000; // Address map size

// Cyclone V FPGA device addresses
const unsigned int LEDR_BASE = 0x00000000; // Leds offset
const unsigned int SW_BASE = 0x00000040; // Switches offset
const unsigned int KEY_BASE = 0x00000050; // Push buttons offset


class
{
  char *pBase;
  int fd;

public:
  DE1SoCfpga()
  {
    // Open /dev/mem to give access to physical addresses
    fd = open( "/dev/mem", (O_RDWR | O_SYNC));
    if (fd == -1) // check for errors in openning /dev/mem
    {
      cout << "ERROR: could not open /dev/mem..." << endl;
      exit(1);
    }
```

```
  // Get a mapping from physical addresses to virtual addresses
  pBase = (char *)mmap (NULL, LW_BRIDGE_SPAN, (PROT_READ | PROT_WRITE), MAP_SHARED, fd,
LW_BRIDGE_BASE);
  if (pBase == MAP_FAILED) // check for errors
  {
    cout << "ERROR: mmap() failed..." << endl;
    close (fd); // close memory before exiting
    exit(1); // Returns 1 to the operating system;
  }
 }

 ~DE1SoCfpga()
 {
  if (munmap (pBase, LW_BRIDGE_SPAN) != 0){
    cout << "ERROR: munmap() failed..." << endl;
    exit(1);
  }
  close (fd); // close memory
 }

 void RegisterWrite(unsigned int reg_offset, int value)
 {
  * (volatile unsigned int *)(pBase + reg_offset) = value;
 }

 int RegisterRead(unsigned int reg_offset)
 {
  return * (volatile unsigned int *)(pBase + reg_offset);
 }

 void Write1Led(int ledNum, int state) {
  if(0 <= ledNum <= 9) {
    int value = 1;
    int count = ledNum;
    if(state == 1) {
      while(count > 0) {
        value = 2 * value;
        count --;
      }
      RegisterWrite(LEDR_BASE, value);
    }
    else if(state == 0){
```

```
      RegisterWrite(LEDR_BASE, 0);
    }
    else{
      cout << "ERROR: state should only be 0 and 1 means off and on" << endl;
    }
  }
  else {
    cout << "ERROR: ledNum should be in 0-9" << endl;
  }
}

int Read1Switch(int switchNum) {
  if(0 <= switchNum <= 9) {
    int value = RegisterRead(SW_BASE);
    int state;
    while (switchNum >= 0){
      state = value % 2;
      value = (value - state) / 2;
      switchNum--;
    }
    return state;
  }
  else{
    cout << "ERROR: switchNum should be in 0-9" << endl;
  }
}

void WriteAllLeds(int value){
//  1111111111 is 1023 means all turn on
  if(0 <= value <= 1023){
    RegisterWrite(LEDR_BASE, value);
  }
  else{
    cout << "out rage" << endl;
  }
}


int ReadAllSwitches(){
  return RegisterRead(SW_BASE);
}
```

```
int PushButtonGet() {
  if(RegisterRead(KEY_BASE) == 0){
    return -1;
  }
  if(RegisterRead(KEY_BASE) == 1) {
    return 0;
  }
  else if(RegisterRead(KEY_BASE) == 2) {
        return 1;
  }
  else if(RegisterRead(KEY_BASE) == 4) {
        return 2;
  }
  else if(RegisterRead(KEY_BASE) == 8) {
        return 3;
  }
  //push more than one key buttons
  else {
    return 4;
  }
}

int ChangeValue(int pushButtonValue, int value){
  bitset<10> buttons(value);
  if(pushButtonValue == 0){
     sleep(1);
     value = value + 1;
     bitset<10> newB(value);
     cout << "New buttons: " << newB  << endl;
     cout << "New value: " << value << endl;
     return value;
  }
  else if(pushButtonValue == 1){
     sleep(1);
     value = value - 1;
     bitset<10> newB(value);
     cout << "New buttons: " << newB  << endl;
     cout << "New value: " << value << endl;
     return value;

  }
```

```
    else if(pushButtonValue == 2){
        sleep(1);
        buttons >>= 1;
        cout << "New buttons: " << buttons  << endl;
        int temp = (int)(buttons.to_ulong());
        cout << "New value: " << temp << endl;
        return temp;
    }
    else if(pushButtonValue == 3){
        sleep(1);
        buttons <<= 1;
        cout << "New buttons: " << buttons << endl;
        int temp = (int)(buttons.to_ulong());
        cout << "New value: " << temp << endl;
        return temp;
    }
    else if(pushButtonValue == 4){
        sleep(1);
        return value;
    }
  }

};

int main()
{
  DE1SoCfpga de1;

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

  int option;
  cout << "Tpying 1 if you are switch well the value with the button: ";
  cin >> option;
  if(option == 1){
    int value = de1.ReadAllSwitches();
    de1.WriteAllLeds(value);
    int temp = value;
    while(true){
      int pushButtonValue = de1.PushButtonGet();
      if(pushButtonValue != -1) {
        temp = de1.ChangeValue(pushButtonValue, temp);
      }
```

```
      de1.WriteAllLeds(temp);
    }
  }
  else{
    cout << "fail to continue" << endl;
  }

// Done
return 0;
}
```