Group: Raymond You and Ethan Carpenter
Professor Kimani
EECE2160 SEC 05 8am
Lab 1 Memory-Mapped I/O and Object-Oriented Programming

Assignment 2

```
/** Set the state of the LEDs with the given value.
*
* @param pBase Base address for general-purpose I/O
* @param value Value between 0 and 255 written to the LEDs
*/
void WriteAllLeds(void *pBase, int value) {
    if(value < 0 || value > 255) {
            return;
    }
    for(int i = 0; i < 7; i++) {
          Write1Led((char*) pBase, i , value % 2);
          value /= 2;
    }
}

  //WriteAllLeds(pBase, 0) -> ALL OFF
  //WriteAllLeds(pBase, 255) -> ALL ON
  //WriteAllLeds(pBase, 1) -> LED1 ON
  //WriteAllLeds(pBase, 2) -> LED2 ON
  //WriteAllLeds(pBase, 3) -> LED1 and LED2 ON
  //WriteAllLeds(pBase, 4) -> LED3 ON
```

Assignment 3

```
/** Reads all the switches and returns their value in a single integer.
*
* @param pBase Base address for general-purpose I/O
* @return A value that represents the value of the switches
*/
int ReadAllSwitches(void *pBase) {
   int total = 0;
   for(int i = 7; i >= 0; i--) {
          total *= 2;
          total += Read1Switch((char*) pBase, i);
   }
   return total;
}
```

```
  //ReadAllSwitches(pBase) with switch 0 on -> 1;
  //ReadAllSwitches(pBase) with switch 1 on -> 2;
  //ReadAllSwitches(pBase) with switch 2 on -> 4;
  //ReadAllSwitches(pBase) with switch 0,1 on -> 3;
  //ReadAllSwitches(pBase) with switch 0,1,2 on -> 7;
  //ReadAllSwitches(pBase) with all switches on -> 255;
```

Assignment 4

a) PushButton.cpp

```
#include <iostream>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/mman.h>
using namespace std;

// 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 pBase                 Base address returned by 'mmap'.
 * @parem offset        Offset where device is mapped.
 * @param value         Value to be written.
 */
void RegisterWrite(char *pBase, int offset, int value)
{
        * (int *) (pBase + 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, 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 pBase       Virtual 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(ledNum == 0) {
            int ledOffset = gpio_led1_offset;
            RegisterWrite(pBase, ledOffset, state);
    }
    if(ledNum == 1) {
            int ledOffset = gpio_led2_offset;
            RegisterWrite(pBase, ledOffset, state);
    }
    if(ledNum == 2) {
            int ledOffset = gpio_led3_offset;
            RegisterWrite(pBase, ledOffset, state);
    }
    if(ledNum == 3) {
            int ledOffset = gpio_led4_offset;
            RegisterWrite(pBase, ledOffset, state);
    }
    if(ledNum == 4) {
            int ledOffset = gpio_led5_offset;
            RegisterWrite(pBase, ledOffset, state);
    }
    if(ledNum == 5) {
            int ledOffset = gpio_led6_offset;
            RegisterWrite(pBase, ledOffset, state);
```

```
    }
    if(ledNum == 6) {
            int ledOffset = gpio_led7_offset;
            RegisterWrite(pBase, ledOffset, state);
    }
    if(ledNum == 7) {
            int ledOffset = gpio_led8_offset;
            RegisterWrite(pBase, ledOffset, state);
    }
}



/** Reads the value of a switch
* - Uses 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(switchNum == 0) {
            int switchOffset = gpio_sw1_offset;
            return RegisterRead(pBase, switchOffset);
   }
   if(switchNum == 1) {
            int switchOffset = gpio_sw2_offset;
            return RegisterRead(pBase, switchOffset);
   }
   if(switchNum == 2) {
            int switchOffset = gpio_sw3_offset;
            return RegisterRead(pBase, switchOffset);
   }
   if(switchNum == 3) {
            int switchOffset = gpio_sw4_offset;
            return RegisterRead(pBase, switchOffset);
   }
   if(switchNum == 4) {
            int switchOffset = gpio_sw5_offset;
            return RegisterRead(pBase, switchOffset);
   }
   if(switchNum == 5) {
            int switchOffset = gpio_sw6_offset;
            return RegisterRead(pBase, switchOffset);
```

```
    }
    if(switchNum == 6) {
            int switchOffset = gpio_sw7_offset;
            return RegisterRead(pBase, switchOffset);
    }
    if(switchNum == 7) {
            int switchOffset = gpio_sw8_offset;
            return RegisterRead(pBase, switchOffset);
    }
//edge case
      if(switchNum > 7 || switchNum < 0) {
              return 0;
      }
}

/** Set the state of the LEDs with the given value.
*
* @param pBase Base address for general-purpose I/O
* @param value Value between 0 and 255 written to the LEDs
*/
void WriteAllLeds(void *pBase, int value) {
     if(value < 0 || value > 255) {
            return;
     }
     for(int i = 0; i < 7; i++) {
           Write1Led((char*) pBase, i , value % 2);
           value /= 2;
     }
}

/** Reads all the switches and returns their value in a single integer.
*
* @param pBase Base address for general-purpose I/O
* @return A value that represents the value of the switches
*/
int ReadAllSwitches(void *pBase) {
    int total = 0;
    for(int i = 7; i >= 0; i--) {
           total *= 2;
           total += Read1Switch((char*) pBase, i);
    }
    return total;
}
```

```
int PushButtonGet(void *pBase) {
   int out = 6;
   for(int i = 0; i < 5; i++) {
         if(RegisterRead((char*)pBase, i*4 + gpio_pbtnl_offset)) {
                              out = (i+4)%5;
         }
   }
   return out;
}



/**
 * Main function to interact with I/O Interfaces
 */
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); // Returns 1 to the operating system;
         }

         //cout << "Please enter a LED number (0 to 7) and the state (0 or 1) to turn on or off any
LED" << endl;
  //int ledNum;
  //int stateNum;
  //cin >> ledNum;
  //cin >> stateNum;
  //Write1Led(pBase, ledNum, stateNum);
  //cout << "Please enter a switch number (0 to 7)" << endl;
  //int switchNum;
  //cin >> switchNum;
  //cout << Read1Switch(pBase, switchNum) << endl;

  //cout << "Please enter a value to set the state of the LEDs with the given value" << endl;
  //int ledValue;
```

```
//cin >> ledValue;
//WriteAllLeds(pBase, ledValue);
//WriteAllLeds(pBase, 0) -> ALL OFF
//WriteAllLeds(pBase, 255) -> ALL ON
//WriteAllLeds(pBase, 1) -> LED1 ON
//WriteAllLeds(pBase, 2) -> LED2 ON
//WriteAllLeds(pBase, 3) -> LED1 and LED2 ON
//WriteAllLeds(pBase, 4) -> LED3 ON



//cout << ReadAllSwitches(pBase) << endl;
//ReadAllSwitches(pBase) with switch 0 on -> 1;
//ReadAllSwitches(pBase) with switch 1 on -> 2;
//ReadAllSwitches(pBase) with switch 2 on -> 4;
//ReadAllSwitches(pBase) with switch 0,1 on -> 3;
//ReadAllSwitches(pBase) with switch 0,1,2 on -> 7;
//ReadAllSwitches(pBase) with all switches on -> 255;

int count = ReadAllSwitches(pBase), oldPress = 6, newPress = 0, bad = 1;
while(1) {
	newPress = PushButtonGet(pBase);
	if(oldPress != newPress) {
		if(newPress == 1)
			count++;
		if(newPress == 2)
			count--;
		if(newPress == 3)
			count = ReadAllSwitches(pBase);
		if(newPress == 0)
			count /=2;
		if(newPress == 4)
			count *=2;
		oldPress = newPress;
		if(newPress != 6) {
			cout << count << endl;
			WriteAllLeds(pBase, count);
		}
	}
}
 cout << PushButtonGet(pBase);
```

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

b) Video attached with zip file.

Assignment 5

a) PushButtonClass.cpp

```
#include <iostream>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/mman.h>
using namespace std;

// 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

class ZedBoard {
     char *pBase;
     int fd;

public:
      ZedBoard();
      ~ZedBoard();
      void RegisterWrite(int offset, int value);
      int RegisterRead(int offset);
      void Write1Led(int ledNum, int state);
      int Read1Switch(int switchNum);
      void WriteAllLeds(int value);
      int ReadAllSwitches();
      int PushButtonGet(void);
};

/**
 * 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.
 */
ZedBoard::ZedBoard () {
          this->fd = open( "/dev/mem", O_RDWR);
          this->pBase = (char *) mmap(NULL, gpio_size, PROT_READ | PROT_WRITE,
MAP_SHARED,
                          this->fd, gpio_address);

  // Check error
          if (this->pBase == MAP_FAILED)
          {
```

```
                cerr << "Mapping I/O memory failed - Did you run with 'sudo'?\n";
                exit(1); // Returns 1 to the operating system;
        }
}

/**
 * Close general-purpose I/O.
 *
 * @param pBase      Virtual address where I/O was mapped.
 * @param fd  File descriptor previously returned by 'open'.
 */
ZedBoard::~ZedBoard () {
        munmap(this->pBase, gpio_size);
        close(this->fd);
}

/**
 * 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 ZedBoard::RegisterWrite(int offset, int value)
{
        * (int *) (this->pBase + 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 ZedBoard::RegisterRead(int offset)
{
        return * (int *) (this->pBase + offset);
}

/** 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 ZedBoard::Write1Led(int ledNum, int state) {
    if(ledNum == 0) {
            int ledOffset = gpio_led1_offset;
            RegisterWrite(ledOffset, state);
    }
    if(ledNum == 1) {
            int ledOffset = gpio_led2_offset;
            RegisterWrite(ledOffset, state);
    }
    if(ledNum == 2) {
            int ledOffset = gpio_led3_offset;
            RegisterWrite(ledOffset, state);
    }
    if(ledNum == 3) {
            int ledOffset = gpio_led4_offset;
            RegisterWrite(ledOffset, state);
    }
    if(ledNum == 4) {
            int ledOffset = gpio_led5_offset;
            RegisterWrite(ledOffset, state);
    }
    if(ledNum == 5) {
            int ledOffset = gpio_led6_offset;
            RegisterWrite(ledOffset, state);
    }
    if(ledNum == 6) {
            int ledOffset = gpio_led7_offset;
            RegisterWrite(ledOffset, state);
    }
    if(ledNum == 7) {
            int ledOffset = gpio_led8_offset;
            RegisterWrite(ledOffset, state);
    }
}



/** Reads the value of a switch
* - Uses base address of I/O
* @param pBase base address of I/O
* @param switchNum Switch number (0 to 7)
```

```
* @return Switch value read
*/
int ZedBoard::Read1Switch(int switchNum) {
   if(switchNum == 0) {
            int switchOffset = gpio_sw1_offset;
            return RegisterRead(switchOffset);
   }
   if(switchNum == 1) {
            int switchOffset = gpio_sw2_offset;
            return RegisterRead(switchOffset);
   }
   if(switchNum == 2) {
            int switchOffset = gpio_sw3_offset;
            return RegisterRead(switchOffset);
   }
   if(switchNum == 3) {
            int switchOffset = gpio_sw4_offset;
            return RegisterRead(switchOffset);
   }
   if(switchNum == 4) {
            int switchOffset = gpio_sw5_offset;
            return RegisterRead(switchOffset);
   }
   if(switchNum == 5) {
            int switchOffset = gpio_sw6_offset;
            return RegisterRead(switchOffset);
   }
   if(switchNum == 6) {
            int switchOffset = gpio_sw7_offset;
            return RegisterRead(switchOffset);
   }
   if(switchNum == 7) {
            int switchOffset = gpio_sw8_offset;
            return RegisterRead(switchOffset);
   }
//edge case
      if(switchNum > 7 || switchNum < 0) {
               return 0;
      }
}

/** Set the state of the LEDs with the given value.
*
```

```
* @param pBase Base address for general-purpose I/O
* @param value Value between 0 and 255 written to the LEDs
*/
void ZedBoard::WriteAllLeds(int value) {
    if(value < 0 || value > 255) {
            return;
    }
    for(int i = 0; i < 7; i++) {
            Write1Led(i , value % 2);
            value /= 2;
    }
}

/** Reads all the switches and returns their value in a single integer.
*
* @param pBase Base address for general-purpose I/O
* @return A value that represents the value of the switches
*/
int ZedBoard::ReadAllSwitches(void) {
   int total = 0;
   for(int i = 7; i >= 0; i--) {
           total *= 2;
           total += Read1Switch(i);
   }
   return total;
}

int ZedBoard::PushButtonGet(void) {
   int out = 6;
   for(int i = 0; i < 5; i++) {
           if(RegisterRead(i*4 + gpio_pbtnl_offset)) {
                                   out = (i+4)%5;
           }
   }
   return out;
}



/**
 * Main function to interact with I/O Interfaces
 */
int main()
```

```
{
  ZedBoard *zb = new ZedBoard;

  int count = zb->ReadAllSwitches(), oldPress = 6, newPress = 0, bad = 1;
  while(1) {
        newPress = zb->PushButtonGet();
        if(oldPress != newPress) {
                if(newPress == 1)
                        count++;
                if(newPress == 2)
                        count--;
                if(newPress == 3)
                        count = zb->ReadAllSwitches();
                if(newPress == 0)
                        count /=2;
                if(newPress == 4)
                        count *=2;
                oldPress = newPress;
                if(newPress != 6) {
                        cout << count << endl;
                        zb->WriteAllLeds(count);
                }
        }
  }
   cout << zb->PushButtonGet();
}
```

Assignment 6

a) CounterSpeed.cpp

```
#include <iostream>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/mman.h>
#include <cmath>
using namespace std;

// 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

class ZedBoard {
     char *pBase;
     int fd;

public:
      ZedBoard();
      ~ZedBoard();
      void RegisterWrite(int offset, int value);
      int RegisterRead(int offset);
      void Write1Led(int ledNum, int state);
      int Read1Switch(int switchNum);
      void WriteAllLeds(int value);
      int ReadAllSwitches();
      int PushButtonGet(void);
};

/**
 * 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.
 */
ZedBoard::ZedBoard () {
        this->fd = open( "/dev/mem", O_RDWR);
        this->pBase = (char *) mmap(NULL, gpio_size, PROT_READ | PROT_WRITE,
MAP_SHARED,
                        this->fd, gpio_address);

  // Check error
        if (this->pBase == MAP_FAILED)
        {
                cerr << "Mapping I/O memory failed - Did you run with 'sudo'?\n";
                exit(1); // Returns 1 to the operating system;
        }
}

/**
 * Close general-purpose I/O.
 *
 * @param pBase      Virtual address where I/O was mapped.
 * @param fd  File descriptor previously returned by 'open'.
 */
ZedBoard::~ZedBoard () {
        munmap(this->pBase, gpio_size);
        close(this->fd);
}

/**
 * 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 ZedBoard::RegisterWrite(int offset, int value)
{
        * (int *) (this->pBase + 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 ZedBoard::RegisterRead(int offset)
{
        return * (int *) (this->pBase + offset);
}

/** 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 ZedBoard::Write1Led(int ledNum, int state) {
    if(ledNum == 0) {
            int ledOffset = gpio_led1_offset;
            RegisterWrite(ledOffset, state);
    }
    if(ledNum == 1) {
            int ledOffset = gpio_led2_offset;
            RegisterWrite(ledOffset, state);
    }
    if(ledNum == 2) {
            int ledOffset = gpio_led3_offset;
            RegisterWrite(ledOffset, state);
    }
    if(ledNum == 3) {
            int ledOffset = gpio_led4_offset;
            RegisterWrite(ledOffset, state);
    }
    if(ledNum == 4) {
            int ledOffset = gpio_led5_offset;
            RegisterWrite(ledOffset, state);
    }
    if(ledNum == 5) {
            int ledOffset = gpio_led6_offset;
            RegisterWrite(ledOffset, state);
```

```
    }
    if(ledNum == 6) {
            int ledOffset = gpio_led7_offset;
            RegisterWrite(ledOffset, state);
    }
    if(ledNum == 7) {
            int ledOffset = gpio_led8_offset;
            RegisterWrite(ledOffset, state);
    }
}



/** Reads the value of a switch
* - Uses base address of I/O
* @param pBase base address of I/O
* @param switchNum Switch number (0 to 7)
* @return Switch value read
*/
int ZedBoard::Read1Switch(int switchNum) {
   if(switchNum == 0) {
            int switchOffset = gpio_sw1_offset;
            return RegisterRead(switchOffset);
   }
   if(switchNum == 1) {
            int switchOffset = gpio_sw2_offset;
            return RegisterRead(switchOffset);
   }
   if(switchNum == 2) {
            int switchOffset = gpio_sw3_offset;
            return RegisterRead(switchOffset);
   }
   if(switchNum == 3) {
            int switchOffset = gpio_sw4_offset;
            return RegisterRead(switchOffset);
   }
   if(switchNum == 4) {
            int switchOffset = gpio_sw5_offset;
            return RegisterRead(switchOffset);
   }
   if(switchNum == 5) {
            int switchOffset = gpio_sw6_offset;
            return RegisterRead(switchOffset);
```

```
    }
    if(switchNum == 6) {
            int switchOffset = gpio_sw7_offset;
            return RegisterRead(switchOffset);
    }
    if(switchNum == 7) {
            int switchOffset = gpio_sw8_offset;
            return RegisterRead(switchOffset);
    }
//edge case
      if(switchNum > 7 || switchNum < 0) {
              return 0;
      }
}

/** Set the state of the LEDs with the given value.
*
* @param pBase Base address for general-purpose I/O
* @param value Value between 0 and 255 written to the LEDs
*/
void ZedBoard::WriteAllLeds(int value) {
     if(value < 0 || value > 255) {
            return;
     }
     for(int i = 0; i < 7; i++) {
           Write1Led(i , value % 2);
           value /= 2;
     }
}

/** Reads all the switches and returns their value in a single integer.
*
* @param pBase Base address for general-purpose I/O
* @return A value that represents the value of the switches
*/
int ZedBoard::ReadAllSwitches(void) {
    int total = 0;
    for(int i = 7; i >= 0; i--) {
           total *= 2;
           total += Read1Switch(i);
    }
    return total;
}
```

```
int ZedBoard::PushButtonGet(void) {
    int out = 6;
    for(int i = 0; i < 5; i++) {
            if(RegisterRead(i*4 + gpio_pbtnl_offset)) {
                                    out = (i+4)%5;
            }
    }
    return out;
}



/**
 * Main function to interact with I/O Interfaces
 */
int main()
{
  ZedBoard *zb = new ZedBoard;

  int count = zb->ReadAllSwitches(), press = 0,counterSpeed=0;
  while(1) {
          sleep(1);
          count+=counterSpeed;
          press = zb->PushButtonGet();
                  if(press == 1)
                          counterSpeed++;
                  if(press == 2)
                          counterSpeed--;
                  if(press == 3)
                          count = zb->ReadAllSwitches();
                  if(press == 4)
                          counterSpeed = -abs(counterSpeed);
                  if(press == 0)
                          counterSpeed = abs(counterSpeed);
                  cout << count << endl;
          }
}
```

b) Video attached with zip file.