Lab 4
Andrew Barba
Hardik Kapoor

Pre-Lab

`Initialize()`
Allocates a chunk of shared memory at /dev/mem, with read and write capabilities to that file.

`Finalize()`
Closes the allocated memory at file descriptor /dev/mem, and then closes the file descriptor.

`RegisterRead()`
Reads from a register using pointer arithmetic to find the register

`RegisterWrite()`
Writes to a register using pointer arithmetic to find the register

Assignment 2: LedOnOff.c

```
[sudo] password for user202:
Enter a value less than 256: 7
Value = 7
user202@localhost:~$ sudo ./LedNumber
[sudo] password for user202:
Enter a value less than 256: 6
Value = 6
user202@localhost:~$ ls
LedNumber  LedNumber.c  \  a.out  examples.desktop  hello.c  sort.c
user202@localhost:~$ vi LedOnOff.c
user202@localhost:~$ ls
LedNumber  LedNumber.c  LedOnOff.c  \  a.out  examples.desktop  hello.c  sort.c
user202@localhost:~$ rm LedOnOff.c
user202@localhost:~$ ls
LedNumber  LedNumber.c  \  a.out  examples.desktop  hello.c  sort.c
user202@localhost:~$ cp LedNumber ./LedOnOff.c
user202@localhost:~$ ls
LedNumber  LedNumber.c  LedOnOff.c  \  a.out  examples.desktop  hello.c  sort.c
user202@localhost:~$ vi LedOnOff.c
user202@localhost:~$ rm LedOnOff.c
user202@localhost:~$ cp LedNumber.c ./LedOnOff.c
user202@localhost:~$ vi LedOnOff.c
user202@localhost:~$ gcc LedOnOff.c  o LedOnOff && ./LedOnOff
LedOnOff.c:111:1: error: expected identifier or '(' before '{' token
 LedOnOff.c: In function 'main':
LedOnOff.c:131:1: warning: format '%d' expects argument of type 'int *', but argumen
t 2 has type 'int' [-Wformat]
user202@localhost:~$ vi LedOnOff.c
user202@localhost:~$ gcc LedOnOff.c -o LedOnOff && ./LedOnOff
LedOnOff.c:111:1: error: expected identifier or '(' before '{' token
user202@localhost:~$ vi LedOnOff.c
user202@localhost:~$ gcc LedOnOff.c -o LedOnOff && ./LedOnOff
Mapping I/O memory failed - Did you run with 'sudo'?
: Bad file descriptor
user202@localhost:~$ sudo ./LedOnOff
[sudo] password for user202:
Enter an LED (1 - 7): 5
Enter on or off (0-1): 0
user202@localhost:~$ sudo ./LedOnOff
[sudo] password for user202:
user202@localhost:~$ sudo ./LedOnOff
[sudo] password for user202:
Enter an LED (1 - 7): 4
Enter on or off (0-1): 1
user202@localhost:~$ vi LedOnOff.c
user202@localhost:~$ sudo ./LedOnOff
[sudo] password for user202:
user202@localhost:~$ gcc LedOnOff.c  o LedOnOff && sudo ./LedOnOff
[sudo] password for user202:
Enter an LED (1 - 7): 5
Enter on or off (0-1): 1
user202@localhost:~$ vi LedOnOff.c
user202@localhost:~$ gcc LedOnOff.c -o LedOnOff && sudo ./LedOnOff
[sudo] password for user202:
```

Assignment 3: SwitchToLed.c

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/mman.h>
// 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 ptr Base address returned by 'mmap'.
 * @parem offset Offset where device is mapped.
 * @param value Value to be written.
 */
void RegisterWrite(char *ptr, int offset, int value)
{
* (int *) (ptr + offset) = value;
}
/**
 * Read a 4-byte value from the specified general-purpose I/O location.
 *
 * @param ptr Base address returned by 'mmap'.
 * @param offset Offset where device is mapped.
```

```
 * @return Value read.
 */
int RegisterRead(char *ptr, int offset)
{
return * (int *) (ptr + 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 ptr Virtual address where I/O was mapped.
 * @param fd File descriptor previously returned by 'open'.
 */
void Finalize(char *ptr, int fd)
{
munmap(ptr, gpio_size);
close(fd);
}

/**
 * Show lower 8 bits of integer value on LEDs
 *
 * @param ptr Base address of I/O
 * @param value Value to show on LEDs
 */
void SetLedNumber(char *ptr, int value)
{
RegisterWrite(ptr, gpio_led1_offset, value % 2);
```

```
RegisterWrite(ptr, gpio_led2_offset, (value / 2) % 2);
RegisterWrite(ptr, gpio_led3_offset, (value / 4) % 2);
RegisterWrite(ptr, gpio_led4_offset, (value / 8) % 2);
RegisterWrite(ptr, gpio_led5_offset, (value / 16) % 2);
RegisterWrite(ptr, gpio_led6_offset, (value / 32) % 2);
RegisterWrite(ptr, gpio_led7_offset, (value / 64) % 2);
RegisterWrite(ptr, gpio_led8_offset, (value / 128) % 2);
}

/** Set the state of the LED with the given index.
 *
 * @param ptr 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 *ptr, int led_index, int state)
{
  int led;

switch (led_index) {
 case 0:
        led = gpio_led1_offset;
        break;
 case 1:
        led = gpio_led2_offset;
        break;
 case 2:
        led = gpio_led3_offset;
        break;
 case 3:
        led = gpio_led4_offset;
        break;
 case 4:
        led = gpio_led5_offset;
        break;
 case 5:
        led = gpio_led6_offset;
        break;
 case 6:
        led = gpio_led7_offset;
        break;
 case 7:
        led = gpio_led8_offset;
        break;
}


  RegisterWrite(ptr, led, state);
```

```
}

int main()
{
// Initialize
int fd;
char *ptr = Initialize(&fd);
// Check error
if (ptr == MAP_FAILED)
{
perror("Mapping I/O memory failed - Did you run with 'sudo'?\n");
return -1;
}

printf("enter while loop...\n");

while (1) {

  SetLedState(ptr, 0, RegisterRead(ptr, gpio_sw1_offset));
  SetLedState(ptr, 1, RegisterRead(ptr, gpio_sw2_offset));
  SetLedState(ptr, 2, RegisterRead(ptr, gpio_sw3_offset));
  SetLedState(ptr, 3, RegisterRead(ptr, gpio_sw4_offset));
  SetLedState(ptr, 4, RegisterRead(ptr, gpio_sw5_offset));
  SetLedState(ptr, 5, RegisterRead(ptr, gpio_sw6_offset));
  SetLedState(ptr, 6, RegisterRead(ptr, gpio_sw7_offset));
  SetLedState(ptr, 7, RegisterRead(ptr, gpio_sw8_offset));


/**
  SetLedState(ptr, 0, 0);
  SetLedState(ptr, 1, 1);
  SetLedState(ptr, 2, 0);
  SetLedState(ptr, 3, 1);
  SetLedState(ptr, 4, 0);
  SetLedState(ptr, 5, 1);
  SetLedState(ptr, 6, 0);
  SetLedState(ptr, 7, 1);

*/
}

printf("End loop.\n");

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

Assignment 4: PushButton.c

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/mman.h>
// 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 ptr Base address returned by 'mmap'.
 * @parem offset Offset where device is mapped.
 * @param value Value to be written.
 */
void RegisterWrite(char *ptr, int offset, int value)
{
* (int *) (ptr + offset) = value;
}
/**
 * Read a 4-byte value from the specified general-purpose I/O location.
 *
 * @param ptr Base address returned by 'mmap'.
 * @param offset Offset where device is mapped.
```

```
 * @return Value read.
 */
int RegisterRead(char *ptr, int offset)
{
return * (int *) (ptr + 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 ptr Virtual address where I/O was mapped.
 * @param fd File descriptor previously returned by 'open'.
 */
void Finalize(char *ptr, int fd)
{
munmap(ptr, gpio_size);
close(fd);
}

/**
 * Show lower 8 bits of integer value on LEDs
 *
 * @param ptr Base address of I/O
 * @param value Value to show on LEDs
 */
void SetLedNumber(char *ptr, int value)
{
RegisterWrite(ptr, gpio_led1_offset, value % 2);
```

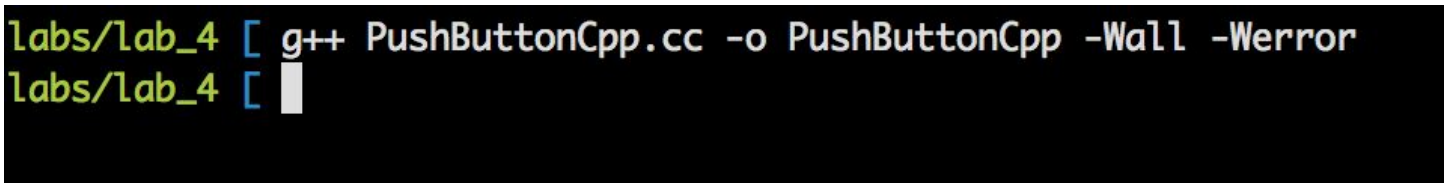
```
RegisterWrite(ptr, gpio_led2_offset, (value / 2) % 2);
RegisterWrite(ptr, gpio_led3_offset, (value / 4) % 2);
RegisterWrite(ptr, gpio_led4_offset, (value / 8) % 2);
RegisterWrite(ptr, gpio_led5_offset, (value / 16) % 2);
RegisterWrite(ptr, gpio_led6_offset, (value / 32) % 2);
RegisterWrite(ptr, gpio_led7_offset, (value / 64) % 2);
RegisterWrite(ptr, gpio_led8_offset, (value / 128) % 2);
}

/** Set the state of the LED with the given index.
 *
 * @param ptr 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 *ptr, int led_index, int state)
{
  int led;

switch (led_index) {
 case 0:
        led = gpio_led1_offset;
        break;
 case 1:
        led = gpio_led2_offset;
        break;
 case 2:
        led = gpio_led3_offset;
        break;
 case 3:
        led = gpio_led4_offset;
        break;
 case 4:
        led = gpio_led5_offset;
        break;
 case 5:
        led = gpio_led6_offset;
        break;
 case 6:
        led = gpio_led7_offset;
        break;
 case 7:
        led = gpio_led8_offset;
        break;
}


  RegisterWrite(ptr, led, state);
```

```
}

int main()
{
// Initialize
int fd;
char *ptr = Initialize(&fd);
// Check error
if (ptr == MAP_FAILED)
{
perror("Mapping I/O memory failed - Did you run with 'sudo'?\n");
return -1;
}

printf("enter while loop...\n");

int current = 0;
int up = 0;
int right = 0;
int down = 0;
int left = 0;
int center = 0;

while (1) {

 int _up = RegisterRead(ptr, gpio_pbtnu_offset);
 int _right = RegisterRead(ptr, gpio_pbtnr_offset);
 int _down = RegisterRead(ptr, gpio_pbtnd_offset);
 int _left = RegisterRead(ptr, gpio_pbtnl_offset);
 int _center = RegisterRead(ptr, gpio_pbtnc_offset);

if (up != _up) {
 up = _up;
 if (_up == 1) current++;
}

if (down != _down) {
 down = _down;
 if (_down == 1) current--;
}

if (right != _right) {
 right = _right;
 if (_right == 1) current /= 2;
}

if (left != _left) {
 left = _left;
```

```
 if (_left == 1) current *= 2;
}

if (center != _center) {
 center = _center;
 if (_center == 1) current = 0;
}









  SetLedNumber(ptr, current);
}

printf("End loop.\n");

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

Assignment 5:

No errors when compiling:



PushButtonCpp.cc

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/mman.h>
// 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_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 representing a single ZedBoard. Allows simple read and write
operations
 */
class ZedBoard {

  char *ptr;
  int fd;

  public:

    /**
     * Public constructor
```

```
     *
     * @param {char} ptr - Base address returned by 'mmap'
     * @param {int} fd - File descriptor passed by reference, where the result
of function 'open' will be stored
     */
    ZedBoard(char * ptr, int fd) {
      this->ptr = ptr;
      this->fd = fd;
    }

    /**
     * Destructor
     */
    ~ZedBoard() {
      munmap(this->ptr, gpio_size);
      close(this->fd);
    }

    /**
     * Set a value to a given register
     *
     * @param {int} offset - Offset where device is mapped
     * @param {int} value - Value to be written
     */
    void registerWrite(int offset, int value) {
      * (int *) (this->ptr + offset) = value;
    }

    /**
     * Read a value from a register
     *
     * @param {int} offset - Offset where device is mapped
     */
    int registerRead(int offset) {
      return * (int *) (this->ptr + offset);
    }

    /**
     * Show lower 8 bits of integer value on LEDs
     *
     * @param {int} value - Value to show on LEDs
     */
    void setLedNumber(int value) {
      this->registerWrite(gpio_led1_offset, value % 2);
      this->registerWrite(gpio_led2_offset, (value / 2) % 2);
      this->registerWrite(gpio_led3_offset, (value / 4) % 2);
      this->registerWrite(gpio_led4_offset, (value / 8) % 2);
      this->registerWrite(gpio_led5_offset, (value / 16) % 2);
```

```
      this->registerWrite(gpio_led6_offset, (value / 32) % 2);
      this->registerWrite(gpio_led7_offset, (value / 64) % 2);
      this->registerWrite(gpio_led8_offset, (value / 128) % 2);
    }

    /**
     * Set the state of the LED with the given index.
     *
     * @parem {int} led_index - LED index between 0 and 7
     * @param {int} state - Turn on (1) or off (0)
     */
    void setLedState(int led_index, int state) {
      int led;

      switch (led_index) {
      case 0:
           led = gpio_led1_offset;
           break;
      case 1:
           led = gpio_led2_offset;
        break;
      case 2:
           led = gpio_led3_offset;
           break;
      case 3:
        led = gpio_led4_offset;
        break;
      case 4:
        led = gpio_led5_offset;
        break;
      case 5:
        led = gpio_led6_offset;
        break;
      case 6:
        led = gpio_led7_offset;
        break;
      case 7:
        led = gpio_led8_offset;
        break;
      }

      this->registerWrite(led, state);
    }
};

/**
 * 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);
}

int main() {

  // Initialize
  int fd;
  char *ptr = Initialize(&fd);

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

  // Allocate zedboard on the heap
  ZedBoard *zboard = new ZedBoard(ptr, fd);

  int current = 0;
  int up = 0;
  int right = 0;
  int down = 0;
  int left = 0;
  int center = 0;

  while (1) {
    int _up = zboard->registerRead(gpio_pbtnu_offset);
    int _right = zboard->registerRead(gpio_pbtnr_offset);
    int _down = zboard->registerRead(gpio_pbtnd_offset);
    int _left = zboard->registerRead(gpio_pbtnl_offset);
    int _center = zboard->registerRead(gpio_pbtnc_offset);

    if (up != _up) {
```

```
      up = _up;
      if (_up == 1) current++;
    }

    if (down != _down) {
      down = _down;
      if (_down == 1) current--;
    }

    if (right != _right) {
      right = _right;
      if (_right == 1) current /= 2;
    }

    if (left != _left) {
      left = _left;
      if (_left == 1) current *= 2;
    }

    if (center != _center) {
      center = _center;
      if (_center == 1) current = 0;
    }

    zboard->setLedNumber(current);
  }

  // Free memory
  delete zboard;

  // Exit cleanly
  return 0;
}
```