/\*

\* @project lab 04 pre-lab assignment

\* @author Matthew Springer

\* @date February 7, 2017

\*/

\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* \*\*

\*\* Part a) \*\*

\*\* \*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Shell script code //

#!/bin/bash

# CreateDir.sh

mkdir $1

echo "Directory $1 created on `date`"

// Console output //

bash-4.3$ ./CreateDir.sh new\_folder

Directory new\_folder created on Tue Feb 7 17:26:08 EST 2017

\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* \*\*

\*\* Part b) \*\*

\*\* \*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// C++ code //

// WiimoteBtns.h //

/\*

\* @project lab04

\* @file WiimoteBtns.h

\* @author Matthew Springer

\* @created February 7, 2017

\* @purpose Header file for class WiimoteBtns

\*/

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

#include <iostream>

class WiimoteBtns {

private:

int fd;

public:

WiimoteBtns();

~WiimoteBtns();

void Listen();

void ButtonEvent(int code, int value);

};

// WiimoteBtns.cpp //

/\*

\* @file WiimoteBtns.cpp

\* @author Matthew Springer

\* @date February 7, 2017

\* @purpose function definitions for WiimoteBtns.h

\*/

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

#include <iostream>

#include "WiimoteBtns.h"

using namespace std;

WiimoteBtns::WiimoteBtns() {

fd = open("dev/input/event2", O\_RDONLY);

if (fd == -1) {

cerr << "Error: Could not open event file - forgot sudo?\n";

exit(1);

}

}

void WiimoteBtns::Listen() {

while (true) {

// read a packet of 32 bytes from Wiimote

char buffer[32];

read(fd, buffer, 32);

// extract code (byte 10) and value (byte 12) from packet

int code = buffer[10];

int value = buffer[12];

// print them

WiimoteBtns::ButtonEvent(code, value);

}

}

void WiimoteBtns::ButtonEvent(int code, int value) {

cout << "Code = " << code << ", value = " << value << endl;

}

WiimoteBtns::~WiimoteBtns() {

close(fd);

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* \*\*

\*\* Part c) \*\*

\*\* \*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// C++ code //

// WiimoteAccel.h //

/\*

\* @project lab04

\* @file WiimoteAccel.cpp

\* @author Matthew Springer

\* @date February 7, 2017

\*/

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

#include <iostream>

#include "WiimoteAccel.h"

using namespace std;

WiimoteAccel::WiimoteAccel() {

fd = open("dev/input/event0", O\_RDONLY);

if (fd == -1) {

cerr << "Error: Could not open event file - forgot sudo?\n";

exit(1);

}

}

WiimoteAccel::~WiimoteAccel() {

close(fd);

}

void WiimoteAccel::Listen() {

// read a packet of 16 bytes from Wiimote

char buffer[16];

read(fd, buffer, 16);

// extract code (byte 10) and value (byte 12) from packet

int code = buffer[10];

short acceleration = \* (short \*) (buffer + 12);

// print them

WiimoteAccel::AccelerationEvent(code, acceleration);

}

void WiimoteAccel::AccelerationEvent(int code, short value) {

cout << "Code = " << code << ", acceleration = " << value << endl;

}

// WiimoteAccel.cpp //

/\*

\* @project lab04

\* @file WiimoteAccel.cpp

\* @author Matthew Springer

\* @date February 7, 2017

\*/

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

#include <iostream>

#include "WiimoteAccel.h"

using namespace std;

WiimoteAccel::WiimoteAccel() {

fd = open("dev/input/event0", O\_RDONLY);

if (fd == -1) {

cerr << "Error: Could not open event file - forgot sudo?\n";

exit(1);

}

}

WiimoteAccel::~WiimoteAccel() {

close(fd);

}

void WiimoteAccel::Listen() {

// read a packet of 16 bytes from Wiimote

char buffer[16];

read(fd, buffer, 16);

// extract code (byte 10) and value (byte 12) from packet

int code = buffer[10];

short acceleration = \* (short \*) (buffer + 12);

// print them

WiimoteAccel::AccelerationEvent(code, acceleration);

}

void WiimoteAccel::AccelerationEvent(int code, short value) {

cout << "Code = " << code << ", acceleration = " << value << endl;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* \*\*

\*\* Part d) \*\*

\*\* \*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// C++ Code //

// ZedBoard.h //

/\*\*

\* @file ZedBoard.h

\* @author John Kimani (j.kimani@neu.edu)

\* @date October, 2016

\* @brief Process GPIO input and output for the Zedboard.

\*

\* Contains a ZedBoard class that opens GPIO ports through

\* memory-mapping for reading switches and push buttons and

\* writing to LEDs

\*/

#ifndef ZEDBOARD\_H

#define ZEDBOARD\_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

// Class Definition

class ZedBoard {

private:

char \*pBase; // virtual address where I/O was mapped

int fd; // file descriptor for dev memory

int dummyValue; // for testing without a Zedboard

public:

ZedBoard(); // Default Constructor

~ZedBoard(); // Destructor

void RegisterWrite(int offset, int value);

int RegisterRead(int offset);

void Write1Led(int ledNum, int state);

void WriteAllLeds(int value);

int Read1Switch(int switchNum);

int ReadAllSwitches();

};

#endif

// ZedBoard.cpp //

/\*\*

\* @file ZedBoard.cpp

\* @author John Kimani (j.kimani@neu.edu)

\* @date October, 2016

\* @brief Process GPIO input and output for the Zedboard.

\*

\* Contains a ZedBoard class that opens GPIO ports through

\* memory-mapping for reading switches and push buttons and

\* writing to LEDs

\*/

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/mman.h>

#include <iostream>

#include "ZedBoard.h"

using namespace std;

/\*\*

\* Constructor Initialize general-purpose I/O

\* - Opens access to physical memory /dev/mem

\* - Maps memory at offset 'gpio\_address' into virtual address space

\*

\* @param None Default constructor does not need arguments.

\* @return None Default constructor does not return anything.

\*/

ZedBoard::ZedBoard(){

cout << "\nStarting...." << endl;

dummyValue = 99;

/\* // Uncomment this block of code when connected to the 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)

{

cerr << "Mapping I/O memory failed - Did you run with 'sudo'?\n";

exit(1); // Returns 1 to the operating system;

}

\*/

}

/\*\*

\* Destructor to close general-purpose I/O.

\* - Uses virtual address where I/O was mapped.

\* - Uses file descriptor previously returned by 'open'.

\*

\* @param None Destructor does not need arguments.

\* @return None Destructor does not return anything.

\*/

ZedBoard::~ZedBoard(){

/\* munmap(pBase, gpio\_size);

close(fd);

\*/

cout << "\nTerminating...." << endl;

}

/\*\*

\* Write a 4-byte value at the specified general-purpose I/O location.

\*

\* - Uses 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 \*) (pBase + offset) = value;

dummyValue = value;

}

/\*\*

\* Read a 4-byte value from the specified general-purpose I/O location.

\*

\* - Uses base address returned by 'mmap'.

\* @param offset Offset where device is mapped.

\* @return Value read.

\*/

int ZedBoard::RegisterRead(int offset)

{

//return \* (int \*) (pBase + offset);

return dummyValue;

}

/\*\*

\* Changes the state of an LED (ON or OFF)

\*

\* - Uses 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)

{

cout << "\nWriting to LED " << ledNum << ": LED state = " << state << endl;

//RegisterWrite(gpio\_led1\_offset + (ledNum \* 4), state);

}

/\*\*

\* Show lower 8 bits of integer value on LEDs

\*

\* - Calls Write1Led() to set all LEDs

\* @param value Value to show on LEDs

\*/

void ZedBoard::WriteAllLeds(int value)

{

cout << "\nWriting to all LEDs...." << endl;

for(int i = 0; i < 8; i++) {// write to all LEDs

Write1Led(i, (value / (1<<i)) % 2);

}

}

/\*\*

\* Reads the value of a switch

\*

\* - Uses base address of I/O

\* @param switchNum Switch number (0 to 7)

\* @return Switch value read

\*/

int ZedBoard::Read1Switch(int switchNum)

{

cout << "\nReading Switch " << switchNum << endl;

//return RegisterRead(gpio\_sw1\_offset + (switchNum \* 4));

return switchNum;

}

/\*\*

\* Reads the switch values into a decimal integer

\*

\* - Calls Read1Switch() to read all switches

\* @return Switches' value read

\*/

int ZedBoard::ReadAllSwitches()

{

int switchValue = 0;

cout << "\nReading all switches...." << endl;

for(int i = 7; i >= 0; i--) {// read all switches

switchValue = (switchValue << 1) + Read1Switch(i);

}

return switchValue;

}

// ZedMain.cpp //

/\*\*

\* @file ZedMain.cpp

\* @author John Kimani (j.kimani@neu.edu)

\* @date October, 2016

\* @brief Process GPIO input and output for the Zedboard.

\*

\* Contains a ZedBoard class that opens GPIO ports through

\* memory-mapping for reading switches and push buttons and

\* writing to LEDs

\*/

#include <iostream>

#include "ZedBoard.h"

using namespace std;

/\*\*

\* Main operates the Zedboard LEDs and switches

\*/

int main()

{

// Initialize

ZedBoard \*zed = new ZedBoard();

int value = 0;

cout << "Enter a value less than 256: ";

cin >> value;

cout << "value entered = " << value << endl;

// Show the value on the Zedboard LEDs

zed->WriteAllLeds(value);

delete zed;

// Done

} //end main

// Makefile //

OBJS = ZedMain.o ZedBoard.o

CC = g++

DEBUG = -g

CFLAGS = -Wall -c $(DEBUG)

LFLAGS = -Wall $(DEBUG)

ZedMain: $(OBJS)

$(CC) $(LFLAGS) $(OBJS) -o ZedMain

ZedMain.o: ZedBoard.h ZedMain.cpp

$(CC) $(CFLAGS) ZedMain.cpp

ZedBoard.o: ZedBoard.h ZedBoard.cpp

$(CC) $(CFLAGS) ZedBoard.cpp

clean:

rm \*.o

rm ZedMain

// Console output //

bash-4.3$ make

g++ -Wall -c -g ZedMain.cpp

g++ -Wall -c -g ZedBoard.cpp

g++ -Wall -g ZedMain.o ZedBoard.o -o ZedMain

bash-4.3$ ./ZedMain

Starting....

Enter a value less than 256: 235

value entered = 235

Writing to all LEDs.…

Writing to LED 0: LED state = 1

Writing to LED 1: LED state = 1

Writing to LED 2: LED state = 0

Writing to LED 3: LED state = 1

Writing to LED 4: LED state = 0

Writing to LED 5: LED state = 1

Writing to LED 6: LED state = 1

Writing to LED 7: LED state = 1

Terminating....

bash-4.3$