Levi Kaplan EECE2160	Embedded Design: Enabling Robotics Prelab Assignment 2

Prelab Assignment 2
The Main Program

Levi Kaplan
kaplan.l@northeastern.edu

Submit Date: 9/23/20
Due Date: 9/24/20

Levi Kaplan EECE2160	Embedded Design: Enabling Robotics Prelab Assignment 2

Method Explanation

Initialize():

Initialize sets up the data to be written to memory and mapped to virtual addresses. It opens access to physical memory as well as maps the memory to virtual addresses using mmap. It also properly handles the case when data is not successfully written to memory. It returns the address to the virtual memory in the form of a character.

Finalize():

Finalize closes the input and output, or throws an error if not able to properly unmap the memory through munmap.

RegisterRead():

RegisterRead reads a 4-byte value from the base address, taking into account the offset, and returns the read value.

RegisterWrite():

RegisterWrite writes the passed-in value at the passed-in address, taking into account the passed-in offset where the device is mapped.

Prelab Code

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <iostream>
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  = 0x00DEC700;  // Address map size
// Cyclone V FPGA device addresses
const unsigned int LEDR_BASE       = 0x00000000;  // Leds offset
const unsigned int SW_BASE         = 0x00000040;  // Switches offset
```

Levi Kaplan EECE2160	Embedded Design: Enabling Robotics Prelab Assignment 2

```
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)
{
  return * (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);
}

void WriteAllLeds(char *pBase, int value)
{
   RegisterWrite(pBase, LEDR_BASE, value);
}


/**
 * 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;
   exit(1);
  }
  // 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
}

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

  // ************** Put your code here **********************
  int value = 0;
 cout << "Enter an int value between 0 to 1023: " << endl;
  cin >> value; cout << "value to be written to LEDs = " << value << endl;
  WriteAllLeds(pBase, value);
  int readLEDs = RegisterRead(pBase, LEDR_BASE);
  cout << "value of LEDS read = " << readLEDs << endl;
  // Done
  Finalize(pBase, fd);
}

/*
 * Changes the state of an LED (ON or OFF)
 *  @param pBase          Base address returned by 'mmap'
 *  @param ledNumLED      Number (0 to 9)
 *  @param state          State to change to (ON or OFF)
 */
void Write1Led(char *pBase,int ledNum, int state)
{
  // write the given led num as the given state
  RegisterWrite(*pBase, LEDR_BASE + ledNum, state);
}

/*
 * 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(char *pBase)
{
  // set default value bewfore reading
  int value = 0;
  // read the value of every switch and add its value to the total
  for (int i = 0; i < 10; i++) {
    value = value + RegisterRead(pBase, SW_BASE + i);
  }

  return value;
}
```