Memory-Mapped I/O and Object-Oriented Programming
Embedded Design: Enabling Robotics
EECE2160

Michael Brodskiy
Brodskiy.M@Northeastern.edu

April 6, 2023

Date Performed:	March 30, 2023
Partner:	Dylan Powers
Instructor:	Professor Shazli

Abstract

This laboratory experiment, intended to introduce memory mapping and object-oriented programming concepts in C++, involved reading and writing to the DE1-SoC FPGA board. Starting with some pre-made register read and write code, conversions between binary and decimal number systems were used to infer the status of switches, LEDs, and push-buttons on the board. Functions were then written to interface with the board; these functions were then converted to an object-oriented program, with two classes: `DE1SoCfpga` and `LEDControl`.

KEYWORDS: memory mapping, object-oriented, read and write, DE1-SoC, binary, decimal, function, class

1 Equipment

Available equipment included:

- DE1-SoC board
- DE1-SoC Power Cable
- USB-A to USB-B Cable
- Computer
- MobaXTerm SSH Terminal
- USB-to-ethernet Adapter

2 Introduction

3 Discussion & Analysis

3.1 Pre-lab

Existing Functions:

a. `char *Initialize(int *fd)`

   First and foremost, `Initialize()` points the `fd` pointer at the location of a physical device in memory, and gives it read, write, and synchronization access. Next, an `if` statement checks for the possibility of errors; that is, if the pointer `fd` is equal to -1, then an error is printed, and the program exits with exit code 1 (error). The physical device is then mapped to a virtual device, and given a virtual memory address, using the `mmap()` function. Another `if` statement then checks whether the memory mapping was successful or not; if yes, the virtual memory location of the device is returned. Otherwise, an error is printed, the `fd` pointer connection to memory is closed, and exit code 1 is returned.

b. `void Finalize(char *pBase, int fd)`

   The `Finalize()` function checks whether the device attached to the memory address pointer `pBase` is successfully unmapped from memory. If successful, it closes the connection to the device using address `fd`. Otherwise, an error is printed, and exit code 1 is returned.

c. `int RegisterRead(char *pBase, unsigned int reg_offset)`

   By combining the base address pointer `pBase` and the device mapping offset value (`reg_offset`), the value that is read from the device at the memory address `pBase` is returned by `RegisterRead()`.

d. `void RegisterWrite(char *pBase, unsigned int reg_offset, int value)`

Similar to `RegisterRead()`, `RegisterWrite` combines the base address pointer `pBase` and the device mapping offset, `reg_offset`, to find the device, and then assigns a specified value to that address.

Writing a switch read function:

Listing 1: Switch Reading Code

```
/**Reads the value of a switch
 * -Uses base address of I/O
 * @param pBaseBase address returned by'mmap'
 * @param switchNumSwitch number (0 to 9)
 * @returnSwitch value read
 */
int Read1Switch(char *pBase, int switchNum) {

    // Read the switch register
    int switchRegisterValue = RegisterRead(pBase, SW_BASE);

    // Mask the value to extract the specified switch bit
    int switchBitMask = 1 << switchNum;

    // if the result is non-zero, then the switch is on,
       off otherwise
    int switchValue;
    //use the bitwise AND operator and compare result
       against the bit mask
    if (switchRegisterValue & switchBitMask) {
        switchValue = 1;
    } else {
        switchValue = 0;
    }

    return switchValue;

}
```

Writing a switch write function:

Listing 2: Switch Writing Code

```
/** Changes the state of an LED (ON or OFF)
 *  @param pBase Base address returned by 'mmap'
 *  @param ledNum LED number (0 to 9)
 *  @param state State to change to (ON or OFF)
 */
void Write1Led(char *pBase,int ledNum, int state) {

```

```
8       if (ledNum < 0 || ledNum > 9) {
9           cout << "ERROR: Invalid LED number. Only LED
                numbers 0 to 9 are valid." << endl;
10          return;
11
12      }
13
14      // Read the LED redister
15      int ledRegisterValue = RegisterRead(pBase, LEDR_BASE);
16
17      // Set the state of the specified LED based on the
            state parameter
18      if (state == 1) {
19          ledRegisterValue = (1 << ledNum);
20      } else {
21          ledRegisterValue &= ~(1 << ledNum);
22      }
23
24      // Write the new LED register value
25      RegisterWrite(pBase, LEDR_BASE, ledRegisterValue);
26
27      // Read the LED register again and print out the value
            to double check (this can be commented out)
28      int UpdatedRegisterValue = RegisterRead(pBase,
            LEDR_BASE);
29      cout << "LED Register Updated Value: " <<
            UpdatedRegisterValue << endl << endl;
30
31  }
```

3.2 Assignment 1

3.3 Assignment 2

3.4 Assignment 3

3.5 Assignment 4

3.6 Assignment 5

4 Conclusion

Overall, this lab was an effective introduction to the concept of C++ object-oriented programming. By first having the user create functions to interface with the board, and then having the user create a class, it was easier to grasp the idea of a class. The

conversion of the program from procedural to object-oriented was, in this manner, facilitated.

5 Appendix

Listing 3: Object-Oriented Source Code

```
/*
 * ========================================================
 *
 *       Filename:  PushButtonClass.cpp
 *
 *    Description:  Interfaces with switches and push-buttons on the De1-SoC board using
 *                  class De1-SoC
 *
 *        Version:  1.0
 *        Created:  03/30/2023
 *       Revision:  none
 *       Compiler:  GCC
 *
 *        Authors:  Michael Brodskiy (Brodskiy.M@Northeastern.edu)
 *                  Dylan Powers (Powers.D@Northeastern.edu)
 *
 * ========================================================
 */

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <iostream>
#include <cmath>
#include <unistd.h>

using namespace std;

class DE1SoCfpga {

private:

    char *pBase;
    int fd;
```

```
37
38  public:
39
40
41      /**
42      * Initialize general-purpose I/O
43      * - Opens access to physical memory /dev/mem
44      * - Maps memory into virtual address space
45      *
46      * @return Address to virtual memory which is mapped to
            physical,
47      * or MAP_FAILED on error.
48      */
49      DE1SoCfpga() {
50
51          // Open /dev/mem to give access to physical
                addresses
52          fd = open( "/dev/mem", (O_RDWR | O_SYNC));
53          if (fd == -1) // check for errors in openning /dev/
                mem
54          {
55              cout << "ERROR: could not open /dev/mem..." <<
                    endl;
56              exit(1);
57          }
58          // Get a mapping from physical addresses to virtual
                addresses
59          char *virtual_base = (char *)mmap (NULL,
                LW_BRIDGE_SPAN, (PROT_READ | PROT_WRITE),
                MAP_SHARED, fd, LW_BRIDGE_BASE);
60          if (virtual_base == MAP_FAILED) // check for errors
61          {
62              cout << "ERROR: mmap() failed..." << endl;
63              close (fd); // close memory before exiting
64              exit(1); // Returns 1 to the operating system;
65          }
66          pBase = virtual_base;
67
68      }
69
70
71      /**
72      * Close general-purpose I/O.
73      *
74      * @param pBase Virtual address where I/O was mapped.
```

```
75      * @param fd File descriptor previously returned by '
            open'.
76      */
77      ~DE1SoCfpga() {
78
79          if (munmap (pBase, LW_BRIDGE_SPAN) != 0) {
80
81              cout << "ERROR: munmap() failed..." << endl;
82              exit(1);
83
84          }
85
86          close (fd); // close memory
87
88      }
89
90      /**
91      * Write a 4-byte value at the specified general-purpose
            I/O location.
92      *
93      * @parem offset Offset where device is mapped.
94      * @param value Value to be written.
95      */
96      void RegisterWrite(unsigned int offset, int value) {
97
98          * (volatile unsigned int *)(pBase + offset) = value
                ;
99
100     }
101
102     /**
103     * Read a 4-byte value from the specified general-
            purpose I/O location.
104     *
105     * @param offset Offset where device is mapped.
106     * @return Value read.
107     */
108     int RegisterRead(unsigned int offset) {
109
110         return * (volatile unsigned int *)(pBase + offset);
111
112     }
113
114 };
115
116 class LEDControl {
```

```
117
118 private:
119
120     DE1SoCfpga *board; // Declare a DE1SoCfpga class
121     bool prevVal[4]; // Store previous button values
        globally
122     bool butValue[4]; // Store current button values
        globally
123
124     // Physical base address of FPGA Devices
125     const unsigned int LW_BRIDGE_BASE = 0xFF200000; // Base
         offset
126
127     // Length of memory-mapped IO window
128     const unsigned int LW_BRIDGE_SPAN = 0x00005000; //
        Address map size
129
130     // Cyclone V FPGA device addresses
131     const unsigned int LEDR_BASE = 0x00000000; // Leds
        offset
132     const unsigned int SW_BASE = 0x00000040; // Switches
        offset
133     const unsigned int KEY_BASE = 0x00000050; // Push
        buttons offset
134
135 public:
136
137     // Initialize the LEDControl class
138     LEDControl() {
139
140         board = new DE1SoCfpga();
141         for (int i = 0; i < 4; i++) {
142
143             prevVal[i] = false;
144             butValue[i] = false;
145
146         }
147
148     }
149
150     // Destroy the board and LEDControl objects
151     ~LEDControl() {
152
153         delete board;
154
155     }
```

```
156
157     /**Reads the value of a switch
158     * -Uses base address of I/O
159     * @param switchNum Switch number (0 to 9)
160     * @return Switch value read
161     */
162     int Read1Switch(int switchNum) {
163
164         // Read the switch register
165         int switchRegisterValue = board->RegisterRead(
                SW_BASE);
166
167         // Mask the value to extract the specified switch
                bit
168         int switchBitMask = 1 << switchNum;
169
170         // if the result is non-zero, then the switch is on
                , off otherwise
171         int switchValue;
172         //use the bitwise AND operator and compare result
                against the bit mask
173         if (switchRegisterValue & switchBitMask) {
174             switchValue = 1;
175         } else {
176             switchValue = 0;
177         }
178
179         return switchValue;
180
181     }
182
183     /** Changes the state of an LED (ON or OFF)
184     *  @param ledNum LED number (0 to 9)
185     *  @param state State to change to (ON or OFF)
186     */
187     void Write1Led(int ledNum, int state) {
188
189         if (ledNum < 0 || ledNum > 9) {
190             cout << "ERROR: Invalid LED number. Only LED
                    numbers 0 to 9 are valid." << endl;
191             return;
192
193         }
194
195         // Read the LED register
```

```
196         int ledRegisterValue = board->RegisterRead(
                LEDR_BASE);
197
198         // Set the state of the specified LED based on the
                state parameter
199         if (state == 1) {
200             ledRegisterValue = (1 << ledNum);
201         } else {
202             ledRegisterValue &= ~(1 << ledNum);
203         }
204
205         // Write the new LED register value
206         board->RegisterWrite(LEDR_BASE, ledRegisterValue);
207
208         // Read the LED register again and print out the
                value to double check (this can be commented out
                )
209         int UpdatedRegisterValue = board->RegisterRead(
                LEDR_BASE);
210         cout << "LED Register Updated Value: " <<
                UpdatedRegisterValue << endl << endl;
211
212     }
213
214     /** Reads all the switches and returns their value in a
            single integer
215     * @return   A value that represents the value of the
            switches
216     */
217     int ReadAllSwitches() {
218
219         return board->RegisterRead(SW_BASE);
220
221     }
222
223     /** Set the state of the LEDs with the given value
224     * @param value      Value between 0 and 1023 written to
            the LEDs
225     */
226     void WriteAllLeds(int value) {
227
228         board->RegisterWrite(LEDR_BASE, value);
229
230     }
231
232     int PushButtonGet() {
```

```
233
234         // Read the switch register
235         int butRegisterValue = board->RegisterRead(KEY_BASE
                );
236
237         // Mask the value to extract the specified switch
                bit
238         int butBitMask[4];
239         for (int i = 0; i < 4; i++) {
240
241             butBitMask[i] = 1 << i;
242
243         }
244
245         // if the result is non-zero, then the switch is on
                , off otherwise
246         //use the bitwise AND operator and compare result
                against the bit mask
247         for (int i = 0; i < 4; i++) {
248
249             if (butRegisterValue & butBitMask[i]) {
250
251                 butValue[i] = true;
252                 if (butValue[i] != prevVal[i]) { prevVal[i]
                        = !prevVal[i]; }
253
254             } else {
255
256                 butValue[i] = false;
257                 if (butValue[i] != prevVal[i]) { prevVal[i]
                        = !prevVal[i]; }
258
259             }
260         }
261
262         int quantDiff = 0;
263         bool diff[] = {false, false, false, false};
264
265         for (int i = 0; i < 4; i++) {
266
267             if (butValue[i] && prevVal[i]) {
268
269                 diff[i] = true;
270                 quantDiff++;
271
272             }
```

```
273
274         }
275
276         if (quantDiff == 2) return 4;
277         else if (quantDiff == 0) return -1;
278         else {
279
280                 for (int i = 0; i < 4; i ++) {
281
282                     if (diff[i]) return i;
283
284                 }
285
286         }
287
288     }
289
290 };
291
292 int main() {
293
294     int ledDisp = 0;
295     LEDControl *control = new LEDControl();
296     // ************** Put your code here
        ***********************
297     while(true) {
298
299             usleep(225000);
300             int change = control->PushButtonGet();
301             if (change == 0) {
302
303                 ledDisp++;
304                 control->WriteAllLeds(ledDisp);
305                 if (ledDisp > 1023) ledDisp = 0;
306
307             } else if (change == 1) {
308
309                 ledDisp -= 1;
310                 control->WriteAllLeds(ledDisp);
311                 if (ledDisp < 0) ledDisp = 1023;
312
313             } else if (change == 2) {
314
315                 ledDisp *= 2;
316                 if (ledDisp > 1023) ledDisp -= 1024;
317                 control->WriteAllLeds(ledDisp);
```

```
318
319             } else if (change == 3) {
320
321                 ledDisp /= 2;
322                 control->WriteAllLeds(ledDisp);
323
324             } else if (change == 4) {
325
326                 control->WriteAllLeds(control->
                        ReadAllSwitches());
327                 ledDisp = control->ReadAllSwitches();
328
329             }
330
331     }
332     // Done
333     delete control;
334
335 }
```