Lab Assignment 6 Memory-Mapped I/O and Object-Oriented Programming

Introduction

In this lab we introduce the concept of memory-mapped I/O to access devices available on the DE1-SoC, including the LEDs, the switches, and the push buttons. Starting with a simple program given in this lab manual that controls the state of the LEDs, we will inspect the state of the other input devices, and make them all interact. Finally, we will use object-oriented programming to abstract some of the functionality related with memory-mapped I/O into a C++ class providing additional data protection and modularity.

Lab 6.0 LEDs, Switches, and Push Button's Parallel Ports on DE1-SoC

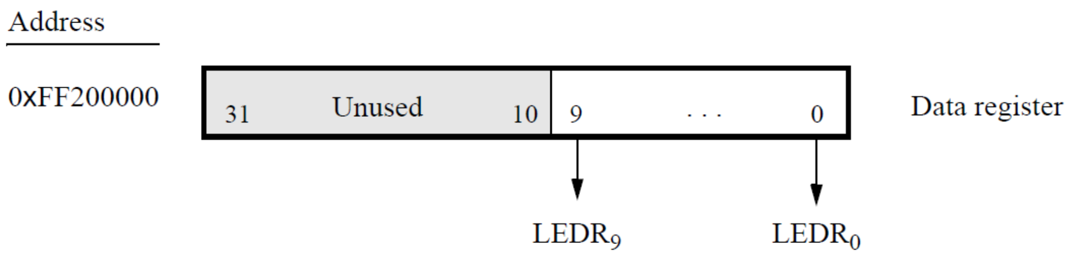

Figure 1: Output parallel port for LEDR

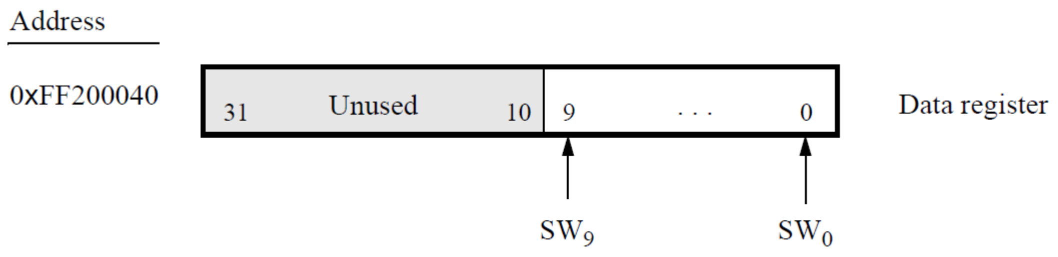

Figure 2: Data register in the slider switch parallel port

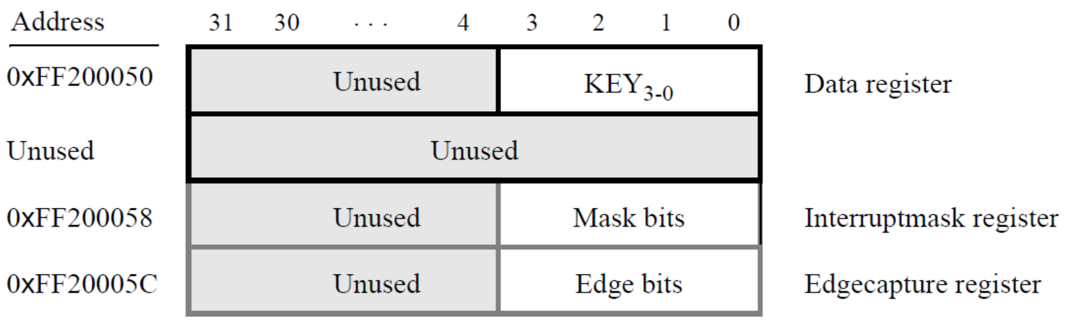

Figure 3: Registers used in the pushbutton parallel port. Note: For this lab, we will only use the Data register to determine the push button pressed.

Group

This assignment is intended to be completed individually but you are allowed to form a group of two to complete this assignment.

Controlling the LEDs

The starting program is available on Canvas. The Prelab assignment part e) and f) is our first attempt to control the devices present in the DE1-SoC, starting with the LEDs. We will write a program that controls the LEDs, switches, and push buttons using memory mapped I/O. Study the program found on Canvas and complete the pre-lab.

The code found on Canvas uses a technique called memory-mapped I/O in order to access devices. This technique consists of accessing a virtual file representing a set of I/O devices using the open() system call, and then mapping a set of control flags into memory locations using function mmap(), which lets you modify or read the device state by just dereferencing a pointer.

Pre-Lab Assignment

Before the start of your lab session, please submit your pre-lab assignment:

a) In the starting code, you will find the following line:

   char *virtual_base = (char *) mmap (NULL, LW_BRIDGE_SPAN, (PROT_READ | PROT_WRITE), MAP_SHARED, *fd, LW_BRIDGE_BASE);

   In your own words, explain the significance of each argument to mmap (NULL, LW_BRIDGE_SPAN, etc.)

   To help you with this question, you can find information in this YouTube video, this website and this other website.

b) Write down the page number in *DE1-SoC_Computer_System_with_ARM_Cortex_A9.pdf* (posted on Canvas) where you can find Figure 1, Figure 2 and Figure 3 found on the first page of this lab assignment.

c) Run the program below to turn on Red LED2, LED1 and LED0. Does the program work? Why/Why not? In your own words please explain why it works/does not work. Tie your answer to the information provided in this YouTube video.

```
#include <iostream>
using namespace std;
#define LEDR_BASE  0xFF200000
int main()
{
  int *red_LED_ptr = (int *)LEDR_BASE;   //from page 60 of IntelDataSheet
  *(red_LED_ptr) = 0x7; // Does it turn on LEDR[2], LEDR[1], LEDR[0]?
  return 0;
}
```

d) Run the program below to read the values of Switch9 (SW9) down to Switch0 (SW0). Try to slide several switches up and the rest of them down before running the program. Does the program work? Why/Why not? In your own words please explain why it works/does not work. Tie your answer to the information provided in this YouTube video.

```
#include <iostream>
using namespace std;
#define SW_BASE  0xFF200040

int main()
{
  int SW_value;
  int *SW_switch_ptr = (int *)SW_BASE;  //from page 37 of IntelDataSheet
  SW_value = *(SW_switch_ptr);  // reading the value of switches
  cout << SW_value << endl;
  return 0;
}
```

e) Write a function to control each LED individually. The function should have the interface below: The function calls RegisterRead(…) to read the current value of the LEDs, and RegisterWrite(…) to write the state to the specified LED.

```
/** 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);
```

In this function, *ledNum* selects LED to control (0 … 9) and *state* controls if LED is off (0), or on (1). For example:

Write1Led(pBase, 0, 1);   Should turn on LED 0.

Hints: How can you use bit masking to turn on or off a specific bit in a register?

f) Write a function analogous to the one above but this time the function needs to read all the switches. The function should have the interface below. The function calls RegisterRead(…) to read the switches register and returns the value of all the switches.

```
/** 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);
```

Submit **a single PDF file** with your responses and pre-lab code to Canvas before coming to the lab. The name of your prelab report needs to be **prelab6.pdf**.

Lab 6.1 Lab Assignment

- Copy and paste the code above into a file named LedNumber.cpp on the DE1-SoC (or download it from Canvas), created in a new directory named *Lab6*.

- Copy and paste your Write1Led(…) and ReadAllSwitches(…) functions from the pre-lab and confirm their functionality from the main function.

    - In the main function, implement code to ask the user to enter an LED number (0 to 9) and the state (0 or 1) to turn on or off any LED. Confirm the correct behavior of Write1Led(…).

    - After writing the value of a LED, you need to read the value of all switches by calling ReadAllSwitches on the main function. Confirm the correct behavior of ReadAllSwitches(…).

- Compile your program and run it on the DE1-SoC.

Assignment 1

Run your program at least three times, entering different numbers for LEDs and different kind of combination of switches for each test.

Congratulations, you have written your first piece of software controlling a device on the DE1-SoC!

Lab 6.2 Reading One Switch

Our next goal is to write a function that reads a switch. The function should have the interface below. In this function *switchNum* selects the switch to read from (0 … 9). The function calls RegisterRead(…) to read the switch register and returns the value of the specified switch. The output value of 0 represents the switch is on the "OFF" position and the output value of 1 represents the switch is on the "ON" position.

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

Assignment 2

Compile and run your program on the DE1-SoC and verify that its behavior is correct from the main function. Test your code for different settings of the switches.

In your report, copy and paste the contents of the function Read1Switch(…), and describe the combinations of switch settings tested during the execution of your program.

Lab 6.3 Controlling the push buttons

Push buttons are another simple kind of input devices available on the DE1-SoC. There are four push buttons to the right of the switches identified as follows:

- KEY0
- KEY1
- KEY2
- KEY3

Assuming that a user can only push one button at a time, write a program that will do the following:

When the user presses the *KEY0* button, the value of the LEDs needs to be incremented by one
When the user presses the *KEY1* button, the value of the LEDs needs to be decremented by one
When the user presses the *KEY2* button, the value of the LEDs needs to be shifted right by one bit, inserting one 0 on the left (e.g., 00010111 → 00001011).
When the user presses the *KEY3* button, the value of the LEDs needs to be shifted left by one bit, inserting a 0 on the right (e.g., 00010111 → 00101110).

- Copy file LedNumber.cpp into a new file called PushButton.cpp in the same directory.

- Write a function named PushButtonGet(…) that returns any values it receives from the push buttons.

- Notice that reading the state of the push buttons is tricky because of debouncing on the push buttons. You want your program to increment or decrement or right/left shift the value of the LEDs only once for each time a push button is pressed. In order to deal with this situation, you will need to store the current state of the push buttons, and only act if you detect a difference in this state.

Assignment 3

Compile and thoroughly test your program.

List the content of file **PushButton.cpp** in your report.

Lab 6.4 Using C++ objects with 1 class

In the last part of this lab, we will use object-oriented programming to abstract the functionality related with I/O operations on the DE1-SoC, including the initialization and finalization of the I/O memory maps, and the read or write operations on I/O memory locations.

We will do this with a new C++ class called DE1SoCfpga, whose constructor and destructor will take care of initialization and finalization operations, respectively—those implemented in functions Initialize() and Finalize(). In the previous C++ code, the functions shared information through arguments and return values, including the base pointer to the I/O memory (variable pBase) and the file descriptor of the virtual device file (variable fd). These two variables should now be public members of class DE1SoCfpga.

Functions ReadRegister() and WriteRegister() also took the base pointer pBase as the first argument. Now, these functions will also be part of class DE1SoCfpga, and will read the base pointer from class member pBase. This approach avoids having to pass this argument around.

Your class should look like the one shown below.

```
class DE1SoCfpga
{

public:
        char *pBase;
        int fd;

        DE1SoCfpga ()
        {
                ...
        }

        ~DE1SoCfpga ()
        {
                ...
        }

        void RegisterWrite(unsigned int offset, int value)
        {
                ...
        }

        int RegisterRead(unsigned int offset)
        {
                ...
        }

        // You need to add Write1Led(), WriteAllLeds(), Read1Switch(),
        // ReadAllSwitches(), PushButtonGet()


};
```

- Copy your program PushButton.cpp into a new file named PushButtonClass.cpp located in directory Lab2.
- Modify the code in PushButtonClass.cpp to convert it into a proper C++ object oriented program with one class and a main() function. Class DE1SoCfpga should be defined as provided above that also includes Write1Led(…), WriteAllLeds(…), Read1Switch(…), ReadAllSwitches(…), and PushButtonGet(…)
- Create the necessary class object in the main() function, and edit all the function arguments as needed to pass objects around.

Assignment 4

Compile and thoroughly test your program.

List the content of file **PushButtonClass.cpp** in your report.

Lab 6.5 Using C++ objects with 2 classes

Now that you have completed section 6.4, we can partition the code even more.

Your new base class needs to look like the one shown below.

```
class DE1SoCfpga
{

public:
        char *pBase;
        int fd;

        DE1SoCfpga ()
        {
                ...
        }

        ~DE1SoCfpga ()
        {
                ...
        }

        void RegisterWrite(unsigned int offset, int value)
        {
                ...
        }

        int RegisterRead(unsigned int offset)
        {
                ...
        }
};
```

- Copy your program PushButtonClass.cpp into a new file named PushButtonClasses.cpp located in directory Lab2.
- Modify the code in PushButtonClasses.cpp to convert it into a proper C++ object-oriented program with two classes and a main() function.
- The base class DE1SoCfpga should be defined as provided above
- The second class, LEDControl inherits the property of the base class DE1SoCfpga
- The second class, LEDControl needs to include a default constructor, destructor, and functions Write1Led(…), WriteAllLeds(…), Read1Switch(…), ReadAllSwitches(…), and PushButtonGet(…)
- Create the necessary class objects in the main() function, edit the LEDControl function arguments as needed to pass objects around and your program should work as it did in Assignment 4 above.

Assignment 5

Compile and thoroughly test your program.

List the content of file **PushButtonClasses.cpp** in your report

Laboratory Report

You need to follow the lab report outline provided on Canvas.
Submit your final and working version of PushButton.cpp, PushButtonClass.cpp, and PushButtonClasses.cpp into Canvas.
Your grader will run your code.

For individual submission:
Submit a document with the following naming convention: *FirstNameLastName*-Lab6.docx (for example: JohnDoe-Lab6.docx) that contains the code found in PushButton.cpp, PushButtonClass.cpp, and PushButtonClasses.cpp. Turnitin will check the genuineness of your work.

For a group of two submission:
Submit a document with the following naming convention: *YourFirstNameLastName-YourLabPartnerFirstNameLastName*-Lab6.docx (for example: JohnDoe-JaneDoe-Lab6.docx) that contains the code found in PushButton.cpp, PushButtonClass.cpp, and PushButtonClasses.cpp. Turnitin will check the genuineness of your work. Each individual is responsible to submit their own file(s)/document(s).