Lab 7

Memory-Mapped I/O and Controlling HW with SW

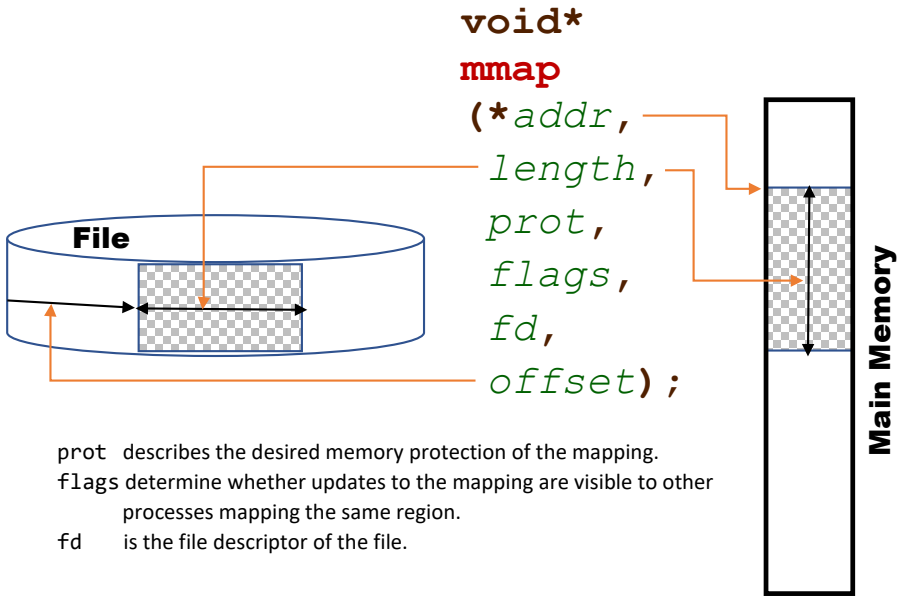
1. Introduction

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. In later labs, we will also use MMIO to control the spider robot's servos. 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.

2. Memory Mapping

A processor can communicate with Input/Output devices (e.g., keyboard, monitor) the same way it communicates with the memory (i.e., through *Load* (read) and *Store* (write) instructions). In this case a portion of the memory "address" space is reserved for I/O devices rather than the regular memory that is used to store running programs' data and instructions. This method of communicating with I/O devices is called memory mapped I/O. The (now less common) alternative is for processors to use dedicated instructions to access I/O devices.

Memory mapping is a fast and efficient method to **securely** allow programs to access physical devices. Normally, the Operating System oversees physical devices, and every I/O access must go through the OS (via a **system call**). Unfortunately, system calls are very slow operations. Memory mapping makes an I/O device directly accessible as a file in the program's address space (I.e., memory addresses that it is allowed to read and write without system calls). In this way, the program only needs **one system call** to set up the mapping, and thereafter can access the I/O device directly. The standard Linux function for establishing a memory map is called mmap(). The following figure illustrates how mmap() works.



The addr parameter is a pointer to *where* the application would like the device to be mapped; if NULL is passed, then the kernel chooses the address at which to create the mapping.

The other important parameter is fd, which stands for *file descriptor* and is the C/C++ representation of a particular file. In this lab, we will memory map the file /dev/mem, which is a special file that gives us access to the physical memory on our DE1-SoC board. Part of this memory space contains the addresses of the board I/O device registers. We will use the mmap() function to map the device registers into the user address space so that we can read/write them with pointers.

2.1 Address of DE1-SoC LEDs, Switches, and Push Button

The following diagrams are taken from the **DE1-SoC** manual:

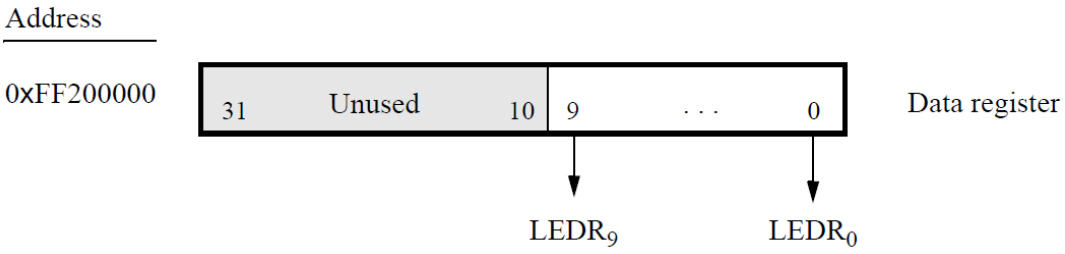


Figure 1: Output parallel port for LEDR

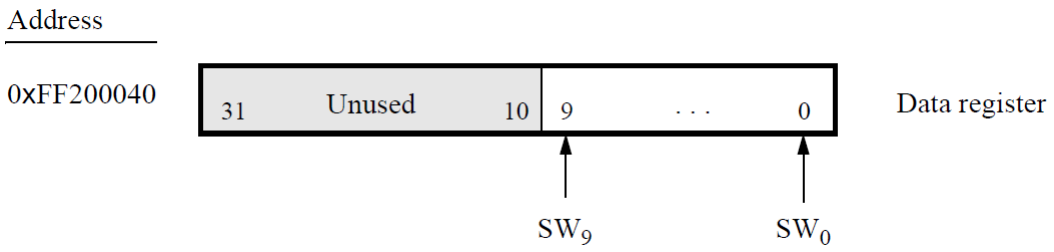


Figure 2: Data register in the slider switch parallel port

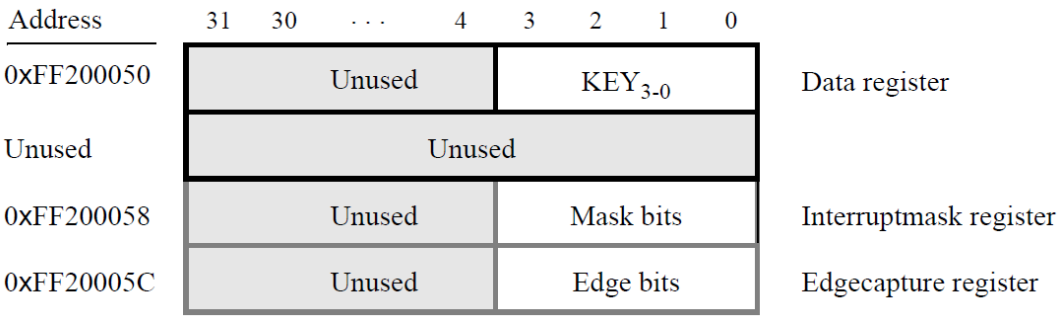


Figure 3: Registers used in the pushbutton parallel port. Note: For this lab, we will only use the Data register to determine the push button pressed.

2. Interfacing with the LEDs and Switches

The program **LedNumber.cpp on Canvas**, which is supplied with this lab, is the beginning of a program to control the devices present on the DE1-SoC. We will modify this program to interface with the LEDs and switches using memory mapped I/O. Study the program on Canvas and **complete the pre-lab**. The given code uses memory mapped I/O to access the DE1 devices. It starts by accessing a virtual file representing a set of I/O devices using the open() system call, and then mapping the file directly into user-space memory into memory locations using function mmap(); mapping files to addresses lets you read or write the device state **by just dereferencing a pointer**.

Pre-Lab Assignment

Before this lab session, please prepare the following material:

a) By using any external references necessary (e.g., to lookup keywords like 'volatile'), explain the code in functions RegisterRead() and RegisterWrite().

b) Implement a function that reads the current integer value represented by the board's 10 switches. The function should have the interface below. The function needs to call RegisterRead(…) to read the state of the 32-bit data register of 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)
```

Hint: You need to use bit masking to clear the higher 22 bits of the 32-bit data register.

c) Implement a function to write an integer value to the board LEDs data register. The function should have the interface below. The function needs to call RegisterWrite(…) to write the value to the 32-bit data register of the LEDs.

```
/** Set the state of the LEDs with the given value.
 *
 * @param pBase      Base address for general-purpose I/O
 * @param value      Value between 0 and 1023 written to the LEDs
 */
void WriteAllLeds(char *pBase, int value)
```

Hint: As the program carries out system administration tasks, you may need to add *sudo* before executing the program (e.g., sudo ./a.out). If you are asked to enter a password use the password of your board Linux account.

Have the prelab assignment answers ready to be checked at the beginning of the lab session.

2.1 Controlling the LEDs by the Switches

- Download LedNumber.cpp from Canvas and then copy and paste your ReadAllSwitches and WriteAllLeds functions from the pre-lab into the appropriate sections.
- Upload the modified file to the DE1-SoC in a directory named *Lab7*.
- Modify the main function in your program to test these two functions.
- Compile your program and run it on the DE1-SoC to test.
  You will need to add the **sudo** command in front of the normal command to execute this program since **mmap** requires high system privilege to access the device memory:
  sudo ./LedNumber.
  This will prompt you for a password; enter the password you used to log onto the DE1.
- Congratulations, you have written your first piece of software controlling a device on the DE1-SoC!

Assignment 1

Modify your program so that the value returned from your ReadAllSwitches function is written to the LEDs using your WriteAllLeds function. This way the LEDs will reflect the status of the switches. Take **2 photos** of the board showing your code working with **different switch states.**

2.2 Controlling One LED

Our next goal is to write a function that controls a specific LED by turning it on or off. The function should have the interface below:

```
/** Changes the state of an LED (ON or OFF)
*  @param pBase      Base address of I/O
*  @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, 5, 1); Should turn on LED number 5.

*Hint: This function **should call** your previous function WriteAllLeds after using the appropriate bit masking to turn on or off a specific bit in the LEDs data register. The other LEDs should remain unchanged indicating that you should be reading the current LED values using RegisterRead.*

2.3. Reading One Switch

Now we need to be able to read and modify only 1 switch/LED at a time.
Write a function that has the interface below:

```
/** Reads the value of a switch
 * @param pBase          Base address of I/O
 * @param switchNum      Switch number (0 to 9)
 * @return Switch value read
 */
int Read1Switch(char *pBase, int switchNum)
```

The function returns 1 if the switch is on and 0 otherwise.
*Hint: This function **should call** your previous function ReadAllSwitches and then use the appropriate bit masking to check the status of the specific switch.*

2.4 Putting it all together

Finally, **modify your main function** so that it prompts the user for an integer; if the **integer is –1** then all LEDS will be set to the value of their respective switches; if the integer is **between 0 and 9** then *only the LED with that index* should be modified to reflect the state of the switch with the same index.

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 switches and different commands for the user and **upload photos** that verify the behavior is correct (**you should caption your photos so that the previous LED state and user prompt are clear**).

3. Interfacing with the Push Buttons

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

- KEY3, KEY2, KEY1, KEY0

Write a program that:

- Interprets the current value of the switches as the initial value of a counter.
- The program starts by reflecting this counter value on the LEDs, using the functions you implemented in the previous assignments.
- **All changes to the counter value** should be reflected immediately on the LEDs so that the LEDs always display the current binary value of our counter.
- When the user presses the *KEY0* button,
  - The current counter value should be incremented by 1.
- When the user presses the *KEY1* button,
  - The counter should be decremented by 1.
- When the user presses the *KEY2* button,
  - The current count should be shifted right one bit position, inserting one 0 on the left (e.g., 00010111 → 00001011).
- When the user presses the *KEY3* button,
  - The current count should be shifted left one bit position, inserting a 0 on the right (e.g., 00010111 → 00101110).
- Finally, when the user presses multiple buttons, the counter should be reset to the value specified by the current state of the switches.

Perform the following tasks:

- Copy file LedNumber.cpp into a new file called PushButton.cpp in the same directory.
- Write a function named PushButtonGet(…) that **returns -1** if no push button is pressed, and a value between 0 and 3 identifying the push button pressed. If multiple push buttons (any two or more) are pressed it should return a value of 4.
- Implement the **main** program described above.
- *Hint 1: To detect a* button press *you will need to keep track of the state of the buttons the **LAST TIME** that you checked them. If a button is held down, then PushButtonGet will repeatedly*

*return the same value; this **does not** mean you should repeatedly update the counter value, it should only be updated **once per press**.*

Hint 2: To keep the program running, use an infinite loop in your main function.

Assignment 3

Compile and test your program. Demonstrate the implemented features by recording a short video. Make sure that you demonstrate all features, including incrementing the counter, decrementing it, and resetting it to the values given by the switches.

4. Lab Submission Instructions

- Each lab consists of a set of **pre-lab questions** and **lab assignments**. You need to submit one lab report with the **answers** to the **pre-lab questions** and the **lab assignments**. If working in a group of maximum two students, only one student must submit one version of the lab report and the source code files with the names of both students on the report cover page and at the top of each source code.
- Write your lab report following the report template provided on Canvas.
- Any source code must be **well-commented** by explaining the purpose of your code. Every function should have a comment explaining its inputs, outputs, and effects. At the beginning of your source code files write your full name, students ID, and any special compiling/running instruction (if any).
- For each lab, submit the following on Canvas before the announced due date/time:
- The lab report, as a single Word or PDF document.
- The source code (.cpp, .cc, or .h files) for lab assignments with C/C++ programming.
- Any files the assignments might ask for (e.g., a demonstration video file).
- Submit each of the above files separately (do not upload compressed files).
- You can submit multiple attempts for this lab; however, only what you submit in the last attempt will be graded (i.e., all required reports and files must be included in this last attempt).

5. Grading Rubric

Section	Items	Points
Prelab	Explanation of ReadRegisters() and WriteRegisters() functions	5
	Commented source code of ReadAllSwitches() function	5
	Commented source code of WriteAllLeds() function	5
Assignment 1	Commented source code of program to read all switches and write to all LEDs	10
	Screen shots of program	5
	Demonstration	10
Assignment 2	Commented source code of program control individual LEDs	10
	Screen shots of program	5
	Demonstration	15
Assignment 3	Commented source code of program to control LEDs with pushbuttons	10
	Screen shots of program	5
	Demonstration	15
Lab Report Quality		5
	Total Points Lab Report	60
	Total Points Demonstrations	40
	Grand Total Points	100