Lab Assignment 7
Controlling the 7-Segment Displays with Object-Oriented Programming

Lab 7.0 Introduction

In this lab, you will utilize the 7-segment displays in the DE1-SoC board to display characters, decimal, and hexadecimal values using object-oriented design in C++. The DE1-SoC has six 7-segment displays controlled by two parallel ports. Each segment (6 to 0) is controlled by one bit with the first parallel port controlling the first four displays (HEX3, HEX2, HEX1, & HEX0), and the second parallel port controlling the last two displays (HEX5 & HEX4) as shown below. Data can be written into these two registers, and read back, by using word operations.

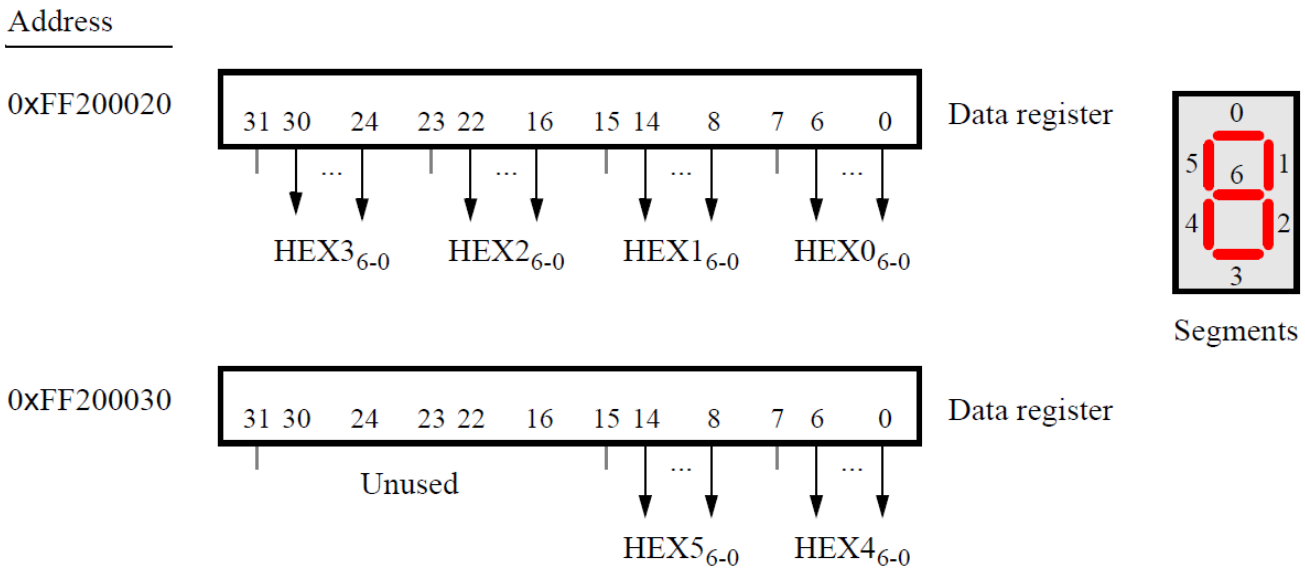

Figure 1: Bit locations for the 7-segment displays port

Figure 2: Hexadecimal digits to display on the 7-segment displays

Group

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

Pre-Lab Assignment

1. Complete the table below to indicate which segments will be turned ON (1) vs OFF (0) for the hexadecimal digits shown in Figure 2 above. The first one has been done for you using the segments in Figure 1.

Character	OUTPUTS - Segment 0/1								
#	6	5	4	3	2	1	0	Decimal	Hex
0	0	1	1	1	1	1	1	63	0x3F
1									
2									
3									
4									
5									
6									
7									
8									
9									
A									
b									
C									
d									
E									
F									

Table 1: Hexadecimal digits to display on the 7-segment displays

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

Bigger Picture for Prelab Question #3 (on the next page):

What the Prelab question #3 is asking you to do is to project/display the data stored in its systems to the outside world.

If you declare a variable as an int at C++, you do not need to worry about doing 2's complement.

The int will have a 32-bit number stored (using 2's complement), and we ask you to project/display the lower bits via seven-segment displays in Hex format.

3. Modify your PushButtonClasses.cpp from the previous lab assignment by removing LEDControl class and adding SevenSegment class. The class SevenSegment inherits the class DE1SoCfpga and used the functions RegisterWrite(…) and RegisterRead(…) as needed. The functions will take a decimal number (positive or negative) and display the corresponding decimal number to Hexadecimal.

The SevenSegment class will have the following functionality:

- Two private data members: unsigned int reg0_hexValue and unsigned int reg1_hexValue representing the state of the two 7-segment display registers. These variables should be updated every time a new value is written to the corresponding registers.
- A class constructor that initializes the private data members.
- A class destructor that calls Hex_ClearAll() function to turn off all the displays
- A public function called Hex_ClearAll() that clears (turns off) all the 7-segment displays.
- A public function called Hex_ClearSpecific(int index) that clears (turns off) a specified 7-segment display specified by index where the index (0 to 5) represents one of the six displays.

- A public function called Hex_WriteSpecific(int index, int value) that writes the digit or character value (from Figure 2 above) to the specified 7-segment display specified by index where the index (0 to 5) represents one of the six displays.
  - Entering a decimal number 0 as **index** and a decimal number -1 as **value** will cause the first seven-segment display (the very far right) to display "F" as we are using 2's complement.
  - Entering a decimal number 0 as **index** and a decimal number 15 as **value** will cause the first seven-segment display (the very far right) to display "F" as we are using 2's complement.
  - Entering a decimal number 5 as **index** and a decimal number 2 as **value** will cause the last seven-segment display (the very far left) to display "2" as we are using 2's complement.

- A public function called Hex_WriteNumber(int number) that writes a positive or negative number to the 7-segment displays.
  - Entering a decimal number -1 as **number** will cause all the seven-segment displays to display "FFFFFF" as we are using 2's complement
  - Entering a decimal number 15 as **number** will cause all the seven-segment displays to display "00000F" as we are using 2's complement

From **Table 1**, create a global const array with the appropriate digit and character codes representing the segments to be turned ON or OFF when writing to a display. The index corresponds to the digits to write on the 7-segment display. The arrays should have the following format:

const unsigned int bit_values[16] = {…};

Do not forget to add the following constants to your code:

const unsigned int HEX3_HEX0_OFFSET = //;

const unsigned int HEX5_HEX4_OFFSET = //;

To help you with the development, **the starting code** for the main() function in a file main.cpp can be found below:

```
/**
 * Main operates the DE1-SoC 7-Segment Displays
 * This program writes an integer number on the 7-Segment Displays
 */
int main(void)
{

	// Create a pointer object of the SevenSegment class
	SevenSegment *display = new SevenSegment;
	cout << "Program Starting...!" << endl;

	//display->Hex_WriteNumber(hex_value);   // display value

	cout << "Terminating...!" << endl;
	return 0;

}
```

Write a program that would allow a user to interact with the seven-segment displays utilizing all of the available functions. This means that you need to create a menu.

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 **prelab7.pdf**.

Read the information about ARM A9 MPCore Timers from one of the manuals on Canvas:

DE1-SoC_Computer_System_with_ARM_Cortex_A9.pdf, section 2.4.1.

Make sure you read the example code.

//End Prelab Assignment

Lab 7.1 Creating the Base Class

In a folder called *lab7*, convert your DE1SoCfpga class from the previous lab into an independent class in its own header file (.h file) and source file (.cpp file). Write the DE1SoCfpga class declaration in a file named DE1SoCfpga.h, and its implementation in a file named DE1SoCfpga.cpp. We'll need the following functions to be in class DE1SoCfpga as implemented in the previous lab:

- Constructor initializing the memory-mapped I/O.
- Destructor finalizing the memory-mapped I/O.
- Function RegisterWrite(offset, value), writing a value into a register given its offset.
- Function RegisterRead(offset), returning the value read from a register given its offset.

The class DE1SoCfpga will be the base class initializing memory and controlling register access. To access the Hex displays, add the HEX3_HEX0_BASE and HEX5_HEX4_BASE register offsets for the 7-segment displays in the DE1SoCfpga.cpp below the LEDs, Switches, and push button key offsets (or you can add the HEX register offsets to SevenSegment.cpp, please see 7.2).

Lab 7.2 Using Object-Oriented Programming to Control the 7-Segment Displays

Create a new class SevenSegment in a header file SevenSegment.h and source file SevenSegment.cpp with the functionality given to you on the Prelab assignment.

Include the array in the SevenSegment.cpp file to use when writing to the 7-segment displays.

Assignment 1

1. In your *lab7* folder, you should have the program implemented in the following files: DE1SoCfpga.h, DE1SoCfpga.cpp, SevenSegment.h, SevenSegment.cpp, & main.cpp. Create a Makefile to compile your programs as discussed in class, including a clean rule accessible through command make clean in order to get rid of all generated binary files.
2. Modify the main function as needed to test and verify the correct operation of the SevenSegment functions. The main function given above can be used to test the function Hex_WriteNumber(int number).
3. Download all the .h and .cpp files, and Makefile into your local machine. Include these files with your report submission.

Lab 7.3 Using Timer to Control 7-Segment Displays

The objective of this last experiment is to use an MPCore timer to control 7-Segment Displays.

Go back to your prelab and use it as your starting point. You will only need to submit one file for this Lab 7.3 called SevenSegmentTimer.cpp.

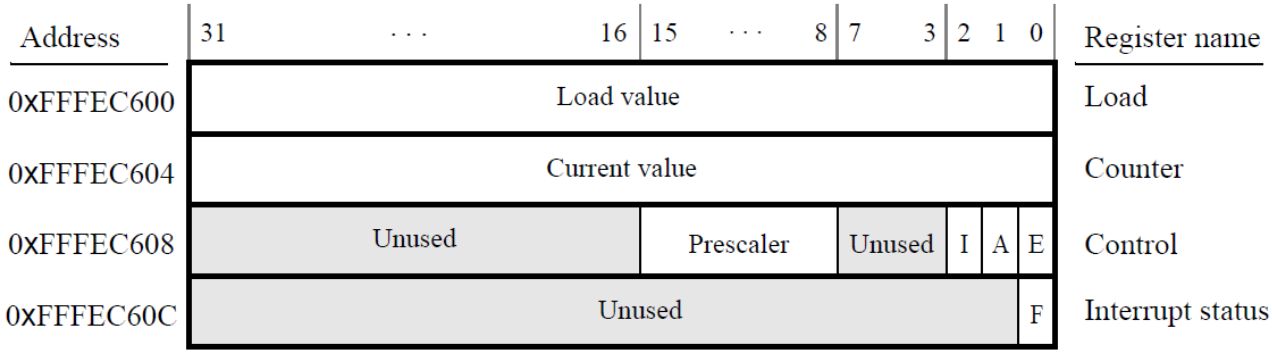

Figure 2. ARM A9 Private Timer Port

Add the following constants to your program based on the information provided in Figure 2:

```
//0xFFFEC600 - 0xFF200000 = 0xDEC600
const unsigned int MPCORE_PRIV_TIMER_LOAD_OFFSET = 0xDEC600;        // Points to LOAD Register
const unsigned int MPCORE_PRIV_TIMER_COUNTER_OFFSET = ;      // Points to COUNTER Register
const unsigned int MPCORE_PRIV_TIMER_CONTROL_OFFSET = ;      // Points to CONTROL Register
const unsigned int MPCORE_PRIV_TIMER_INTERRUPT_OFFSET = ;  // Points to INTERRUPT Register
```

In dealing with a Timer, it is always a good idea to save the current state of the timer registers prior to entering the program and restoring them after we are done with them.

In your SevenSegment *constructor*, you need to save the initial state of each register, hence:

```
initialvalueLoadMPCore = RegisterRead(MPCORE_PRIV_TIMER_LOAD_OFFSET);
initialvalueControlMPCore = RegisterRead(MPCORE_PRIV_TIMER_CONTROL_OFFSET);
initialvalueInterruptMPCore = RegisterRead(MPCORE_PRIV_TIMER_INTERRUPT_OFFSET);
```

In your SevenSegment de*structor*, you need to restore the state of each register to their initial value, hence:

```
RegisterWrite(MPCORE_PRIV_TIMER_LOAD_OFFSET, initialvalueLoadMPCore);
RegisterWrite(MPCORE_PRIV_TIMER_CONTROL_OFFSET, initialvalueControlMPCore);
RegisterWrite(MPCORE_PRIV_TIMER_INTERRUPT_OFFSET, initialvalueInterruptMPCore);
```

To help you with the development, **the starting code** for the main() function can be found below:

```
/**
 * Main operates the DE1-SoC 7-Segment Displays
 * This program writes an integer number on the 7-Segment Displays
 */
int main(void)
{

	SevenSegment *display = new SevenSegment;
	cout << "Program Starting...!" << endl;

	int counter = 200000000;   // timeout = 1/(200 MHz) x 200x10^6 = 1 sec
	display->RegisterWrite(MPCORE_PRIV_TIMER_LOAD_OFFSET, counter);
	display->RegisterWrite(MPCORE_PRIV_TIMER_CONTROL_OFFSET, 3);

	int enterindex = 0;  //points to Hex0 display
	int entervalue = 0;  //points to segment0
	int count = 20;

	while (count > 1)
	{
		if (display->RegisterRead(MPCORE_PRIV_TIMER_INTERRUPT_OFFSET) != 0)
		  {
		    display->RegisterWrite(MPCORE_PRIV_TIMER_INTERRUPT_OFFSET, 1);
		      // reset timer flag bit
		    entervalue = entervalue ^ 0x1;
		    display->Hex_WriteSpecific(enterindex, entervalue);
		    count = count – 1;
		  }
	}

	delete display;

	cout << "Terminating...!" << endl;
	return 0;

}
```

Quick Note:
Adjust the value of LW_BRIDGE_SPAN by modifying the value of FINAL_PHYSICAL_ADDRESS should you need to.

Write a program that show your full name in a rotating manner using all the 7-segment displays, every 0.5 seconds as long as Toggle Switch 0 (SW0) is on the downward position. When SW0 is on the upward position, exit the program.
Assume our full name is John Doe.
The display order shall be:

Progress	HEX5	HEX4	HEX3	HEX2	HEX1	HEX0
t = 0 second						J
t = 0.5 second					J	o
t = 1 second				J	o	h
t = 1.5 second			J	o	h	n
t = 2 second		J	o	h	n	D
t = 2.5 second	J	o	h	n	d	o
t = 3 second	o	h	n	d	o	e
t = 3.5 second	h	n	d	o	e	J

Assignment 2

1. Modify the main function as needed to test and verify the correct operation of the SevenSegment functions using the MPCore timer.
2. Submit your SevenSegmentTimer.cpp into Canvas.

Laboratory Report

You need to follow the lab report outline provided on Canvas.

Submit your final and working version of all the files needed to complete Assignment 1 and 2.

Your grader will run your code.

For individual submission:

Submit a document with the following naming convention: FirstNameLastName-Lab7.docx (for example: JohnDoe-Lab7.docx) that contains all the files needed to complete Assignment 1 and 2. Turnitin will check the genuineness of your work.

For a group of two submission:

Submit a document with the following naming convention: YourFirstNameLastName-YourLabPartnerFirstNameLastName-Lab7.docx (for example: JohnDoe-JaneDoe-Lab7.docx) that contains all the files needed to complete Assignment 1 and 2. Turnitin will check the genuineness of your work. Each individual is responsible to submit their own file(s)/document(s).