# **Microprocessors (662-133) NAME: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Lab #9 – Library Routines

Description

This lab will detail how to create a complete C program in the Altera NIOS EDS. We will take the program written in assembly language in lab 4 and rewrite it using C.

Learning objectives

1. Create the proper file structure needed by Eclipse.
2. Use expanded debugging techniques available in C.
3. Create a C program.
4. Use the expression window to debug and evaluate program operation.
5. Observe function calls.
6. Use the disassembly to view the assembly program created by the complier.
7. Expand the use of If..Else statements.
8. Explore the system.h library file.

**Procedure**

For this lab, we'll re-do (and then tweak or expand) one of the labs we did in assembly, lab 4 (the simple calculator).

So create a new project called lab 9, and create a new source file named 'lab9.c', and then add the contents of the included lab9.c file. Note that very little is provided this time, just the #include statements and the main() function block. At the end of this lab are some tips and strategy for creating the structure of a C program when starting from scratch.

Recalling the lab....

The program will have a main loop (use a while(1) { \*code goes here\* } loop structure), at the beginning, we'll wait for a pushbutton to be pressed, and then depending on the value of the pushbuttons:

If KEY0 is pressed, set our storage variable to be equal to the slider switches value.

If KEY1 was pressed, add the value from the switches with the stored value

If KEY2 was pressed, logically AND the value of the switches with the stored value

If KEY3 was pressed, multiply the value of the switches with the stored value

Whatever the result of those operations (either the result of the math operation, or just the value we're storing) gets displayed on the red LEDs

Then wait for the pushbuttons to be released, then close the loop so your program goes back to the beginning of the while(1) { } code block.

Finding register addresses

We've already seen the function used to load values from an IO port (such as the switches or pushbuttons), that is IORD\_ALTERA\_AVALON\_PIO\_DATA(base\_address). You can use it assigning an integer value like so:

int buttons;

buttons = IORD\_ALTERA\_AVALON\_PIO\_DATA(PUSHBUTTONS\_BASE);

// This loads the value on the pushbuttons and puts it into the integer buttons.

There is also a write version, like this:

int value = 500;

IOWR\_ALTERA\_AVALON\_PIO\_DATA(RED\_LEDS\_BASE,value);

// This writes the value of 500 into the register for the RED leds.

To use these two functions, you must add the following #include statements at the beginning of your program:

#include <altera\_avalon\_pio\_regs.h>

#include <system.h>

The first include has definitions for the IO functions, the second has definitions for all the devices on the NIOS II processor, like PUSHBUTTONS\_BASE, RED\_LEDS\_BASE, GREEN\_LEDS\_BASE, SLIDER\_SWITCHES\_BASE. They are defined much as we did with .EQU statements in assembly, using #define directives, you can examine the system.h file in the lab9\_bsp project, this contains all the addresses.

Bitmasking in C

If you recall in assembly, we used masking (using bitwise AND) to select individual bits to examine, we do the same in C, and it is even simpler to apply.

For instance, to check if bit 3 (value 2^3 or 8) is set in an integer variable named 'x'...

if (x & 8)

{

// Code goes here

}

The & symbol is the bitwise AND symbol, see the others on page 69 of C in a nutshell.

So the value of x is taken, ANDed with the number 8 (bit 3, or 0b00001000), which will give a result of either 0 or 8, depending on if bit 3 was set in the variable x. When we use a number in an if statement, remember that 0 is considered false, and anything other than 0 is true. So in this case, if bit 3 was not set, the result is 0, and so the if condition is false, and the code does not run. If bit 3 was set, the result will be non-zero (8 in this case), and the code runs.

Be careful NOT to use the && symbol, that is a logical AND, which evaluates each side as true or false FIRST, and then ands the two true/false values, which is not at all what we want when masking

A few hints

Remember to allocate all the variables you need at the beginning of a code block (main in this case). In particular, since we want the 'stored value' variable to survive from loop to loop, it must be OUTSIDE of the main while(1) loop. If it were inside it would be re-created each time.

You can also chain together the if statements checking each of the buttons with if...else if....else if.... if you wish. Then you could put the output (the IOWR\_AVALON\_.... function call) either once at the end (using a variable), or put it inside each of the if blocks.

The trickiest part of this program may be the wait for a button press... wait for a button release. Use this pseudocode:

int x;

x = read the value from the pushbuttons;

while (x is pressed/not pressed , ie nonzero/zero)

{

x = read the value from the pushbuttons;

}

The read command appears twice, this is on purpose, once is the initial check, and after that, we check over and over until the condition in the while() loop FAILS (evaluates to false).

Further work:

1) Try changing the output to, instead of the red LEDS, use a printf() command. Use the format from an earlier lab, and remember when dealing with integers, use the integer format in the string(%d).

2) Use the structure of the calc() function from an earlier lab, and have your program use that function for one of the pushbutton presses. Modify the function to do whatever mathematical operation you wish.

3) Change that function to return a float type, and modify the math so that it gives a floating-point result (include a division by a floating-point number, for example: return x / 2.5;) Then use printf() in the main program to display it, changing the format string to %f.

4) Try changing one of the variables to be a global. Does it have any impact on your program?

Here is a short guide to starting out writing a program:

Fundamentally, a C program is a bunch of code blocks, that form sections of code enclosed in { and } braces. Each code block is a series of statements that will be executed in order. The primary example of this we've seen so far is the main() function. This code block is mandatory, and is where program execution starts and ends. We can write other functions (each of which is a code block) and we can write code blocks within main() that are executed conditionally (using IF statements or loops) as well.

The other main concept in C programs are variables, these are representations of all the data in a C program. Variables are either declared outside of all the code blocks (global variables) or they are declared at the very beginning of a code block, and they exist only inside that particular code block (local variables).

Lets take a quick look at the structure of a C program with most of the statements removed for simplicity, and a few comments (line-comments start with // ):

testprogram.c

#include <file.h> // Include any libraries we need

int a,b,c; // Global variable declarations happen outside any code blocks

int main()

{

// The code block for the main function starts here

int i; // A local variable declaration

statement; // Each code block includes as many statements as we want

statement; // Each statement ends in a semicolon

while(a > b) // The while block has its own code block

{

// The code inside this code block runs multiple times

// As long as (a > b) evaluates to true

statement;

statement;

}

if (c >= b) // An if statement has a code block that only executes if the condition is true

{

// The statements in this code block will run only if (c >= b) is true

statement;

c = calc(a); // this code block includes a function call

}

else

{

// If the expression (c >= b) was not true, this code block runs instead

statement;

c = calc(b);

}

// The code block for the main function ends here, as does the program

}

int calc(int x) // This function takes a single integer parameter, that will be named x locally

{

// The code block for the calc function starts here

int z; // A variable local to this code block

statement;

statement;

// The code block for the calc function ends here, with a return statement

return z;

}

When starting out with a blank file, it can be difficult to know where to start with a program. Usually it's helpful to envision writing a set of step-by-step instructions if you were going to have another person perform the calculations you want the processor to do. Writing pseudo code or using a flowchart for this can be helpful.

For instance, for the calculator program in this lab, you might give a set of instructions like this:

At the start:

Wait until a pushbutton is pressed

Read the value of the pushbutton

Read the value of the switches

if button 0 was pressed, store the current value of the switches

if button 1 was pressed, add the value from the switches with the stored value

if button 2 was pressed, logically AND the value of the switches with the stored value

if button 3 was pressed, multiply the value of the switches with the stored value

Whatever the result of that that operation was, display it on the red LEDs

Wait until the pushbuttons have been released

go back to the start

When you're trying to translate something like this into C (or maybe pseudo code first), look for key concepts or phrases:

Whenever you mention getting a value of something or using it, you're probably talking about a variable that will need to be defined. Above we see the pushbutton's value, the switches' value, and the stored value, as well as maybe the output/result we will display.

Whenever you see the word if, you're probably talking (not surprisingly!) about an if statement. In particular are the four button checks. You might also chain them together in 'else if' statements, if when one condition is true, you don't want to bother checking for others. In this case someone could press multiple pushbuttons at the same time, yet we only want to deal with one operation at a time, so we would use 'else if' to only process the first 'command' we see.

Whenever you see something like 'wait for...' or 'once this happens...' or 'go back to..' you're talking about a loop. For the simple 'wait for' it'll probably be a small while loop, with the only thing inside is the reading/updating of the value (in this case reading the pushbuttons) and the expression inside the while(expression) statement is the OPPOSITE of the condition we're waiting for (so the loop runs as long as the condition we're waiting for has not yet happened).

With a main loop in the program, like we have here 'go back to the start' we probably have a while(true) type of loop, in this case while(1) is typically used. (Anything not 0 is true). You should consider what variables should be declared before, or what statements should be executed before the loop starts. Make sure everything is setup or initialized as needed, since those statements don't need to execute every time.