EL2208 Problem Solving with C Lab. Lab #14

Creating personal library and complex numbers to analyze the highpass filter.

Use X-ming+Putty as you did in Lab #5. Reopen the module if you forget how to use it.

This weeks lab is an exercise of creating your own personal library and complex numbers to analyze the highpass filter shown in **Figure 1.** over a wide range of frequencies. The circuit is composed of a capacitor (C) that stores energy due to an electric field and a resistor (R) that impedes current in the circuit.

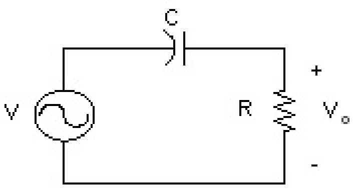

Figure 1. Highpass Circuit

The highpass filter will pass signals at high frequencies called the "pass band". At frequencies below the pass band, the output voltage (**V_o**) approaches zero so signals are attenuated. This region in the frequency response is called the "stop band". The equation for computing the magnitude of the output voltage is

$$|V_o| = \left| \frac{j2\pi fRCV}{1 + j2\pi fRC} \right| \tag{1}$$

where **V** is the input voltage, **f** is the frequency in Hertz (Hz), and $j = \sqrt{-1}$.

In this lab we will write our own library to implement the complex number type declaration and complex number functions. The first step to create our own library is to write a header file. The header file for this exercise is called "**complex.h**". In this file you will need to write comment about the library, for example:

```
/************************************************************************/
/* complex.h                                                            */
/* a library of complex number definition and functions                 */
/* created by :                                                         */
/* date       :                                                         */
/* version    :                                                         */
/************************************************************************/
/* Complex type definition                                              */
/************************************************************************/
....
/************************************************************************/
/* Function:                                                            */
/************************************************************************/
```

Use a structure to store the complex numbers. The structure definition is

```
typedef struct
{
   double real;
   double imaginary;
} complex;
```

Complete the **complex.h** file with two function header for the following two functions:

1. The first will perform a complex division. Both the numerator and denominator in Equation 1 are complex numbers; the real part of the numerator is zero and the imaginary part of the numerator is **j2πfRCV** (**0 + j2πfRCV**). Write (and call) a function which performs this division and returns the quotient. The prototype is

   complex divide(complex numerator, complex denominator);

2. The second function will compute the magnitude of a complex number. The return value will be a floating-point (**double**) number. The prototype is

   double magnitude(complex a);

   The magnitude of a complex number is the square root of the sum of the square of the real part and the square of the imaginary part.

Remember we need to put the keyword **extern** before the function header.

The second step is to write the function body in an implementation file called "**complex.c**" along with some comment and the necessary library to make this implementation file.

```
/***********************************************************************/
/* complex.c                                                           */
/* a library of complex number function declared in complex.h         */
/* created by :                                                        */
/* date       :                                                        */
/* version    :                                                        */
/***********************************************************************/
/* Include the necessary libraries                                     */
/***********************************************************************/
#include <stdio.h>
#include "complex.h"
/***********************************************************************/
/* Function body:                                                      */
/***********************************************************************/
```

Now write your main program and make sure to include "complex.h" as one of your library. For this exercise we want to compute the magnitude of the output voltage ($|V_o|$) over a frequency range from 0Hz through 10MHz at 100Hz increments. Use the values **R=50000**, **C=1.0e-9**, and **V=10.0**. You can read these values in at runtime or use **#define** statements to initialize these quantities. To compute the magnitude of the output voltage, make a call to the complex number division function and the complex number magnitude function.

Print the frequency and the magnitude of the output voltage in two columns to a file called **lab14.out** in preparation for plotting the data with **Gnuplot**. Do not put any column headings in your output file. You do not need to write a separate function to write to the output file.

To compile your program, there are two ways to do this. The first one is :

1. Compile your implementation file using the following command:
   ```
   gcc -c complex.c
   ```
2. Compile your program:
   ```
   gcc -o lab14 lab14.c complex.o
   ```

The second method is to compile them together using the following command:

```
   gcc -o lab14 lab14.c complex.c
```

When you have the code running and the output file created, start **gnuplot** and type the following commands into gnuplot command prompt the file:

```
gnuplot> set logscale xy
gnuplot> set xlabel "Frequency (Hz)"
gnuplot> set ylabel "Output Voltage (V)"
gnuplot> set title "Frequency Response of a Highpass Filter"
gnuplot> plot "lab14.out" with lines
```

Because of the very wide range in frequencies (0 to 10MHz), we used a log-log plot. This will make the distances between orders of magnitude equal on the graph. For instance, it is the same distance along the x-axis between 10 and 100 Hz as it is between 10kHz and 100kHz.

Show your lab instructor the finished plot.

Submit the C code using lab submitter.