CMPE 310 - Assembly Language Project 3

Assigned: Fri March 07, 2008. Due Tue April 1, 2008

Project Description

Write a Linux Assembly Language program that reads a waveform from a file (the filename given as a command-line argument), applies a second-order digital filter, and writes the output to a file. The emphasis of this project is on using the Floating-Point Unit (FPU) to process floating point numbers.

Efficient use of the FPU registers will be part of the grading criteria. Specifically, there need only be one memory read (from the input array) per loop iteration. Constant multiples *and* previous points (amplitudes) in the waveform can be retained in the FPU register stack. In other words, there is one amplitude read in from memory and one which is discarded from the FPU register stack in each loop iteration.

Waveform File Format Specification

The format of the waveform files (same for input and output) is designed to be friendly with `GNUPLOT`. Each line is either a (time, amplitude) pair or a comment. Comment lines start with the `#` pound character, and the remainder of the line should be ignored.

The `fprintf()` / `fscanf()` format string should look like `"%G\t%G\n"`. The `%G` matches either a floating-point number with no exponent (ex: `[-]ddd.ddd`), or a floating point number in scientific notation (`[-]d.dddE±dd`, note the 'E' is capitalized). Each "pair line" will consist of a floating-point number followed by a tab (`\t`), then another floating-point number followed by a newline character.

There will be no size-of-waveform indicator included; you will have to check the output of `fscanf()` to see when the input file has ended. Following is an example input file with two points (pairs),

```
# time      amplitude
1152936000 242
1.15380E+9 309
```

For clarity, the ASCII representation of the above format is as follows:

```
0000000   #       t   i   m   e  \t  \t   a   m   p   l   i   t   u   d
0000010   e  \n   1   1   5   2   9   3   6   0   0   0  \t   2   4   2
0000020  \n   1   .   1   5   3   8   0   E   +   9  \t   3   0   9  \n
```

There will be a `GNUPLOT` script available with will plot an input file and an output file simultaneously.

Description of Digital Filter

The procedure applied to the input waveform should implement a low-pass filter with a cutoff frequency of one quarter of the (Nyquist) sample rate. This is a normalized frequency of $\frac{1}{4}$. The filter will be a 2nd-order approximation to a Butterworth filter, which means each value of the output depends on three previous values of the input and two previous values of the output (making it causal).

The general form of a digital filter specification is:

$$H(z)=\frac{B(z)}{A(z)}=\frac{b(1)+b(2)z^{-1}+...+b(n+1)z^{-n}}{1+a(2)z^{-1}+...+a(n+1)z^{-n}} \quad .$$

In this case, the coefficients of $b(n)$ and $a(n)$ are given by elements of the following matrices:

$$B=[0.0976, 0.1953, 0.0976]$$

$$A=[1.0000, -0.9428, 0.3333]$$

By substitution, we obtain

$$H(z)=\frac{0.0976+0.1953z^{-1}+0.0976z^{-2}}{1-0.9428z^{-1}+0.3333z^{-2}} \quad .$$

To implement the digital filter, we rewrite $H(z)$ as a difference equation:

$$y[n]=0.0976\,x[n]+0.1953x[n-1]+0.0976x[n-2]+0.9428y[n-1]-0.3333y[n-2] \quad .$$

This is the function $y[n]$ which we want to implement. This is a function which depends on previous values of both the input $x[n-1]$, $x[n-2]$ etc., and the output $y[n-1]$, $y[n-2]$ etc., as well as their current values. Computed properly, this transfer function should multiply the frequency components of the input waveform by the components shown in Illustration 1. This is called the magnitude of the frequency response (of the transfer function).

It is not imperative that you understand the underlying theory for this project, but what should be clear is that this difference equation implements a low-pass filter, and should have the effect on a square wave (the input) as shown in the output in Illustration 2.

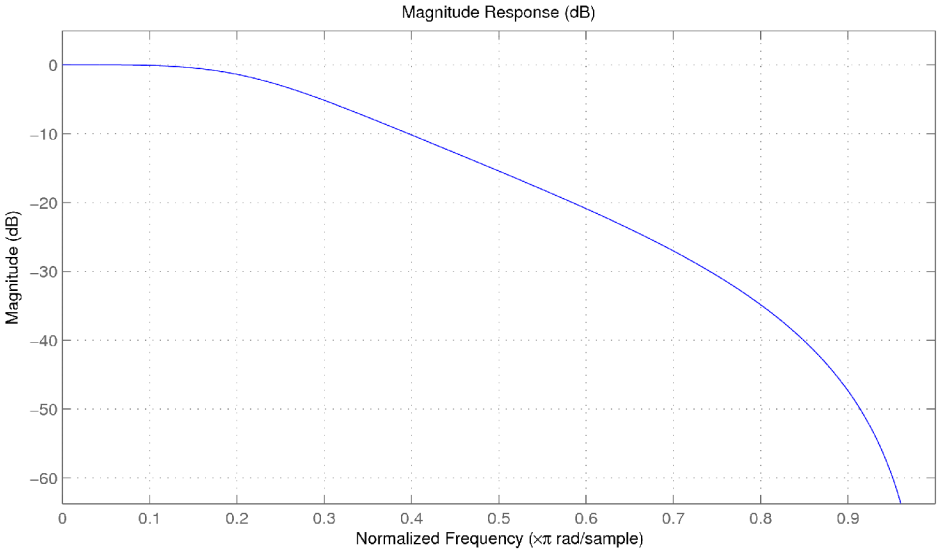

Illustration 1: Magnitude response of 2nd-order Butterworth digital filter

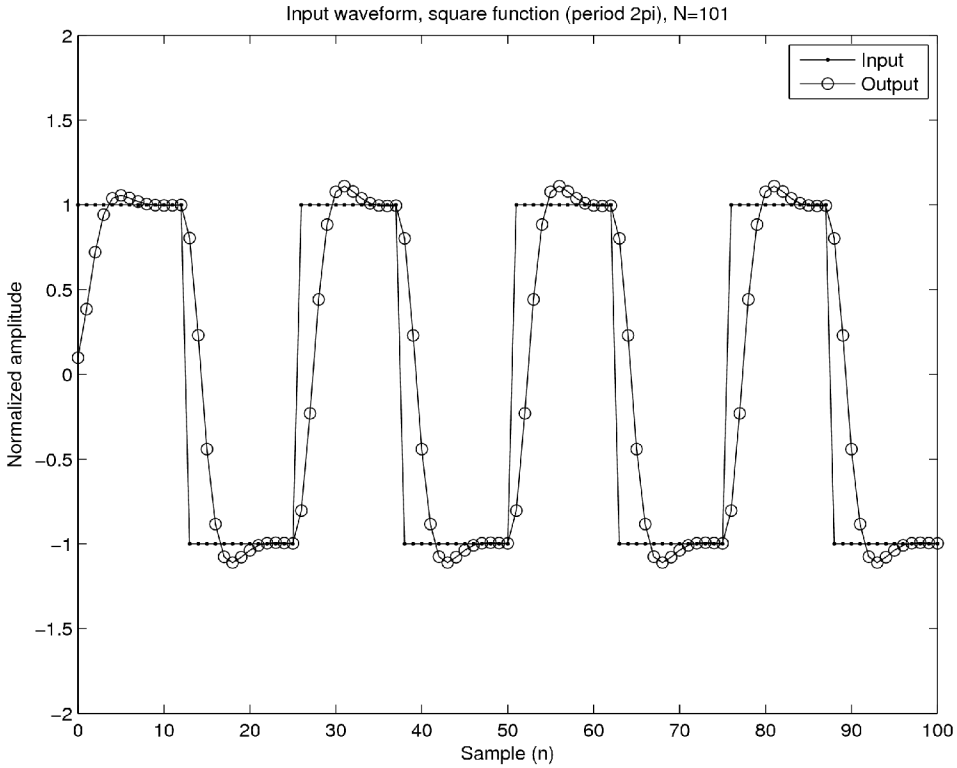

Illustration 2: Example of transformation with a square wave input

NOTE: You may use the C library functions for this project. The relevant functions are fopen, fscanf and printf. You will need to use gcc to link your source code if you use the C library functions.

THE LABS ARE INDIVIDUAL EFFORTS: INSTANCES OF CHEATING WILL RESULT IN YOU FAILING THE COURSE.