ECSE426 - Microprocessor Systems
Lab 5: Optimized FIR Filtering and Decimation Using CMSIS-DSP

1 Background

When dealing with any sensor device, there is often a minimum sampling rate required to mitigate aliasing distortion. Aliasing causes distortion because frequency components above half the sampling rate "fold" back over and corrupt the frequency components of interest. This rate, half the sampling rate, is called the *Nyquist frequency* and must *always* be obeyed to maintain signal integrity.

In the case of your accelerometer, you have a minimum sampling rate of 100Hz. For applications such as your tilt detection, this is much faster than you need. You can only practically update this value a few times of second, as any changes faster that are likely to be caused by shaking or sensor noise. A common technique to reduce the sampling rate to a rate better suited to your application is called *decimation*. This refers to the low-pass filtering of the signal to below the Nyquist frequency of your target rate to allow for re-sampling of the signal. Note, that in the process some of the samples are dropped to achieve this rate.

FIR Filtering

Finite impulse response (FIR) filtering is a process using a simple filter type, which is guaranteed to be stable. It consists of a simple convolution of the input signal x with an impulse response h of length N. The output y of the filter is computed as follows:

$$y[n] = h[0]x[n] + h[1]x[n-1] + \ldots + h[N-1]x[n-(N-1)]$$

Depending on the chosen coefficients (h), the filter can have different behaviour. In this case, we are interested in a simple low-pass filter. You actually already created a low-pass FIR filter in your first lab. The moving average filter is an FIR filter where all coefficients are $1/N$. Naturally, most solutions employed some optimizations which aren't applicable in the more general case.

Decimation

Decimation by M is a simple operation to reduce a sampling rate (re-sampling) from f_s to f_s/M. Once filtering has been done to ensure that there is no significant energy between $f_s/2M$ and $f_s/2$ (the original Nyquist rate), the operation involves dropping $M-1$ samples of every M samples so long as M is an integer.

Fixed-point Arithmetic

Floating-point arithmetic is convenient because it has a huge dynamic range, and yet also has improved precision at the expense of greater computational load. It does this by storing both an exponent and a collection of normalized (significant) digits. In this sense, the variable exponent allows variable precision, where the radix point (generalization of "decimal point") floats, hence the name "floating point". This is in contrast to *fixed-point arithmetic*, where, as the name would imply, that point is fixed. As such, we have a fixed number of integer bits and fixed number of fractional bits. This still allows for representation of non-integers but using fast integer arithmetic.

The designer can choose different parameters of their fixed-point words. For example, for greater precision it might be preferable to use a 32-bit word instead of 16. Similarly, the number of fractional bits has to be chosen to ensure that the required range of numbers be representable. A 32-bit word with 2 integer bits is called *Q2.30*. For the common case that there is only one integer bit (the sign bit in 2's complement), it is called *Q1.31* or simply *Q31*. The range of such a type would be from -1(0x80000000) to ~ 0.9999999995343387(0x7FFFFFFF).

In fixed-point arithmetic, addition and subtraction operate as with any integers. Multiplication, however, effectively moves the radix point. A $Q2.30 \times Q2.30$ operation will produce a $Q4.60$. To store this then in a 32-bit register, we would shift right by 30 and truncate the two MSbs. We may also opt to keep it in this intermediate format to maintain precision.

Even though the Cortex-M4 has a floating-point unit (FPU), it can perform integer operations faster and even supports a number of single-instruction multiple-data (SIMD) instructions that allow for multiple data points to be handled in the same cycle. One issue that fixed-point has which isn't such a big deal in floating point is the idea of overflow. An overflow in integer arithmetic can mean an error of the full scale of the word length. For this reason, the M4 also provides *saturating arithmetic* instructions, where the results are clamped to avoid such an error.

CMSIS-DSP

The CMSIS libraries, which give you access to low-level instructions in C also provides a rather comprehensive library of DSP functions, optimized for your hardware. The documentation is excellent reference, indicating among others, which operations are supported for many different data types. The source is provided but an optimized binary is also distributed and this can be used to reduce compile time.

2 Objectives

This lab build on the previous lab and should have the same functionality. Your goal is to add better signal conditioning by neatly down-sample your accelerometer readings to limit all data to 5Hz (f_s=10Hz). You are given a set of floating-point coefficients for a linear-phase FIR filter to down-sample accordingly. First, implement your own fixed-point FIR filter/decimator to perform this operation. Like in lab 1, a "state" must be maintained between function calls and a similar API is suggested. For the sake of simplicity, your filter should take in 10 samples (decimation by 10) at a time and output a single new value. Your function must take 16-bit values and return similar but the fractional length is up to you. Next, you will perform the same operation using the CMSIS-DSP `arm_fir_decimate_q15` function. You will have to be able to switch between use of either filter implementation so it would be in your best interests to use a similar API.

3 Demonstration

The demonstration will be simple and will consist of showing the proper functionality of your filter, buffering scheme, and an inspection of your design methodology. Since you're going to such lengths for data integrity, you must also convince us that your timing (100Hz input rate, 10Hz output rate) is perfect within the tolerances of your crystal resonator (50ppm).

You will have time to demo your lab until March 15, 2013. Note, that this is the last lab you are doing your old group of two. Lab 6 as well as the final project to follow will be done in the groups of 4.

4 Report

A report detailing your developments and observations in this lab will be required, though of reduced length.

5 Reference Material

- For CMSIS-DSP documentation and examples, look inside the CMSIS folder in your base project for lab 4. The HTML Doxygen output is also here

6 Appendix

```
1 // Please convert these to an appropriate fixed-point format
2 // For some formats, CMSIS-DSP provides conversions for you
3 float const coeffs[] = {
4   0.07416114002096, 0.03270293265285, 0.03863816947328,
5   0.04444276039114, 0.0497985768307,  0.05449897183339,
6   0.05841301164775, 0.06134989872289, 0.06313835392955,
7   0.06373119762273, 0.06313835392955, 0.06134989872289,
8   0.05841301164775, 0.05449897183339, 0.0497985768307,
9   0.04444276039114, 0.03863816947328, 0.03270293265285,
10  0.07416114002096
11 };
```