DSP Project

Online filtering on embedded hardware

This document provides an outline for a DSP project to be completed by the end of the course. You are expected to investigate, in detail, methods related to solving the problem. There is a substantial design element to the project, and a quantitative evaluation of the performance of the proposed methods must be performed and presented. You are to write up a comprehensive report (at most 10 pages) describing your method and results.

For this project you may work in groups of up to three. You will need access to a microprocessor with ADC and DAC functionality, and someone in the group should have experience in programming it. It should support floating-point arithmetic unless you're comfortable with getting software-based floating-point libraries to work on an integer processor.

The Problem

Some microprocessors have ADCs and DACs and can be used to filter analog signals: digitise the signal, process it using discrete-time filtering, and send the result to the analog output. This project aims to investigate methods of implementing filters for causal systems that operate on streaming input data. Such online operation is different from batch processing, where one has access to all the data in storage before processing begins. Both finite impulse response (FIR) filters and infinite impulse response (IIR) filters are to be considered.

FIR filtering

Any discrete-time linear time-invariant (LTI) system can be represented by its impulse response $h[n]$ according to the input-output relation

$$y[n] = \sum_{k=-\infty}^{\infty} h[k]x[n-k].$$

If the system is causal then $h[n] = 0$ for $n < 0$. If it is additionally FIR with length $q+1$ then $h[n] = 0$ for $n > q$, then

$$y[n] = \sum_{k=0}^{q} h[k]x[n-k] = h[0]x[n] + h[1]x[n-1] + \cdots + h[q]x[n-q].$$

The current output is a weighted sum of the current input and the q previous inputs, where the weights are the impulse response values.

Filter design packages can be used to find impulse responses for particular filters. For example in Matlab

```
hv = remez(20,[0,.5,.55,1],[0,0,1,1]);
```

gives the coefficients

```
hv = [0.0387 -0.0874 -0.0463 0.0288 0.0202 -0.0591 -0.0248
      0.1023 0.0248 -0.3171 0.4749 -0.3171 0.0248 0.1023
     -0.0248 -0.0591 0.0202 0.0288 -0.0463 -0.0874 0.0387]
```

These values correspond to $h[0]$ to $h[20]$, and completely specify a particular filter. To implement this filter, for every new input you need to calculate the sum of the current and the previous 20 inputs weighted by the impulse response values to produce the current output value. Computationally each output requires 21 floating-point multiply operations and 20 additions. Typically the impulse response and input values are real, although complex values are permitted with an increase in computational cost.

Preliminary task

The first task is to implement an analog signal processor from the FIR filter with given coefficients. Those given above can be used as typical.

Every time a new input becomes available we must calculate and set the new output, using the filter weights and additional stored inputs.

On a typical microprocessor you can run an ADC at a specified rate and have it generate an interrupt when the next sample arrives. Inside the interrupt service routine (ISR) you store the new value, implement the filter calculation, and update the DAC. Alternatively you might use a timer interrupt to trigger a single-shot ADC sample before calculating and setting the output. This adds latency but might be fine for low frequencies.

The sampling frequency is variable and is limited by the available computing capacity. Aim to use audio frequencies around $F_s = 8$kHz if possible. Your ISR must be as efficient as possible to ensure that it returns before the next sample arrives. If your processor can't manage this at 8kHz then use a lower rate.

Determine the frequency response of the filter. The Matlab `freqz` can be used, or one can simply calculate $H(e^{j\omega})$ directly for a dense set of values of ω. Connect your device to a signal generator at the input and a scope at the output. Use input

sinusoids of varying frequency to ensure that you understand the gain and phase properties of the system: generate a Bode plot experimentally and verify it against theory. You should also be able to observe the periodicity of the frequency response.

Objective

Once you get a basic FIR filter working you can do an investigation around it. Anything interesting or informative will do, but some options are:

1. Use a filter design package to generate impulse responses for different filters, and test them using your implementation.
2. Try to maximise the sampling frequency of your filter. This is probably limited by the processor, especially for filters with long impulse responses. Making the ISR as efficient as possible is critical.
3. Build a first-order RC lowpass filter in the lab with a cutoff of say 2kHz, and design and implement its digital equivalent. Compare their behaviour in the lab.
4. Extend your implementation to incorporate previous output values into the calculation of the current output, essentially adding poles to the system function. The issue of initialisation will have to be considered.
5. Filtering in the time domain using convolution is expensive. Modify your implementation to use a FFT (don't write your own!) for frequency-domain filtering using overlap-add or overlap-save. This requires that the sampled data be processed in batches, which adds latency but can increase throughput.
6. Many processors have DSP functionality in the form of specialised instructions, which are probably most easily used through a vendor-supplied library. Look around for such resources and see if you can get the filter to run at a higher sampling rate or with longer impulse responses.
7. If you can get filtering going at audio rates you might choose to digitally implement some standard audio effects.