DSP - Audio Demodulation and Reconstruction

Group 12 of Evan Sutcliffe, Will Panton, Jamie McDonald

I. Introduction

THIS abstract outlines an implementation of a computationally and memory efficient digital signal processing algorithm, which demodulates and reconstructs an audio stereo signal. The aim was to achieve real-time processing given the constraints faced by using a Raspberry Pi (e.g. 2GB of RAM). The code was prototyped in MATLAB before being implemented in *C*. The following justifies how we achieved the design specification in each section of the code.

A. Initial Decimation

The Nyquist-Shannon theorem says the sufficient sampling frequency to capture all the information of the input signal is 2.14MHz (twice the maximum frequency component). Given the input is sampled at 10MHz there would be significant processing of overabundant samples. For greater efficiency the code discards of every 3 in 4 samples thus decimating by 4 to achieve a new sample rate of 2.5 MHz. This speeds up subsequent signal processing actions as 4 times less samples are processed. This type of decimation will producing aliasing across the signal, however the next section (I-B) describes filtering action which removes alias frequencies in any case, so no pre-filtering is required by this section as this would add unnecessary computational expense.

B. Demodulation

Demodulation of the signal was performed using a time domain convolution of the decimated audio signal using an FIR filter. This was chosen as traditional envelope detectors cannot be used for DSB SC. The FIR filters (seen in Table I) were generated using an implementation of the Parks-Mcclellan design algorithm. For this the we specified a ripple of 10dB for the bandpass and an attenuation of -20dB for other frequencies. This was chosen due to the low level of aliasing from the demodulation making small-scale stop-band attenuation acceptable. From the specifications, an FIR filter of order 86 was generated. The benefit of using a smaller filter allowed for a faster computation as the computational complexity of the convolution is $O(n^2)$ [1], [2].

TABLE I
FIR Designed Filter Perameters

Name	Passband ripple (dB)	Stopband ripple (dB)	Passband	Transition region	FIR order
DIFF	10	-20	1.03-1.07MHz	1KHz	86
SUM	10	-20	0.98-1.02MHz	1KHz	86
LP	1.23	-41.96	0-3KHz	2KHz	36

After filtering the signal is multiplied by a local oscillator which is designed to never be out of phase with the input (see system diagram page 2) as this would provide deficient sound quality.

C. Re-sampling

In order to achieve the specified output sampling frequency of 48 kHz the demodulated signals (which have f_s = 2.5 MHz) were re-sampled. However since these signals can't be decimated by an integer factor M to yield the desired frequency, they must first be interpolated by a factor L.

Interpolation adds unwanted spectral images at multiples of the original sampling frequency. Therefore a filter is required. A filter is also needed for the input to the decimator to prevent aliasing. Because these filters occur sequentially, they can be combined into a single low pass filter, with a cutoff frequency equal to the lower of the two cut-off frequencies (in this case f_c of the decimator).

First attempts to implement this re-sampling function involved naïve algorithms, that interpolated by 'padding' zeros between samples, low-pass filtering with graphical convolution, then decimating by discarding samples. Though functional, this method this would not allow exact re-sampling by factor of 12/625 without running slowly. Instead a factor of 1/52 was used, yielding an output with f_s 48077 Hz (justified based on the <1% difference from the desired f_s) that 'sounded good' and could be generated quickly.

To remedy this, an FIR interpolator, comprising of L poly-phase filters ("sub-filters") was used. In the naïve implementation, the stuffed zeros that are being multiplied with coefficients of the filter produce outputs that have no effect. Using poly-phase filters avoids this unnecessary computation by selecting the coefficients that will produce a useful output (every Lth coefficient, starting at 0, through L-1). Figure I-C illustrates this operation

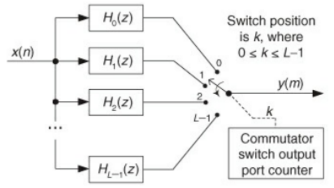
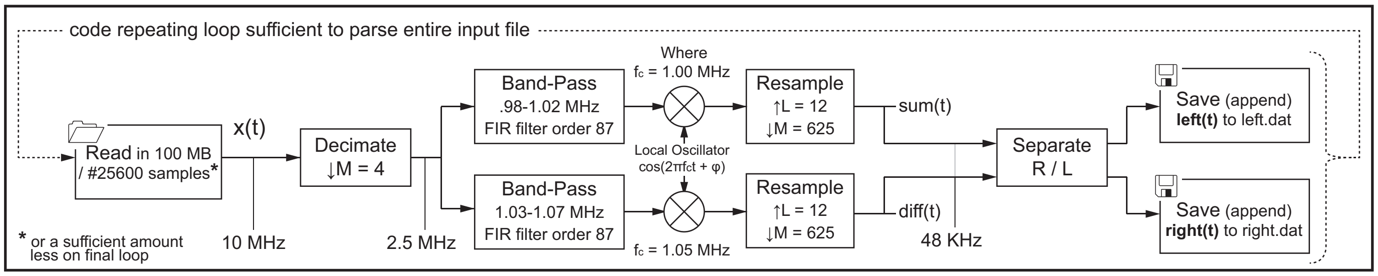

$H_0(z)$
$H_1(z)$
$H_2(z)$
$H_{L-1}(z)$
$x(n)$
$y(m)$
Switch position is k, where $0 \le k \le L-1$
0
1
2
$L-1$
k
Commutator switch output port counter

Fig. 1. FIR interpolator with L poly-phase filters and commutator switch [3]

This generates L outputs per input without having to use memory to store zeros, and works by loading an input into delay line, calculating sum of products using the relevant sub-filters and then shifting delay line. Then, rather than generating unnecessary outputs that were destined to be discarded, the commutator switch can be rotated to every Mth (modulo L) ouput to produce the same effect as decimating by M.


code repeating loop sufficient to parse entire input file
Read in 100 MB / #25600 samples*
x(t)
Decimate ↓M = 4
Band-Pass .98-1.02 MHz FIR filter order 87
Band-Pass 1.03-1.07 MHz FIR filter order 87
Where fc = 1.00 MHz
Local Oscillator cos(2πfct + φ)
fc = 1.05 MHz
Resample ↑L = 12 ↓M = 625
Resample ↑L = 12 ↓M = 625
sum(t)
diff(t)
Separate R / L
Save (append) left(t) to left.dat
Save (append) right(t) to right.dat
* or a sufficient amount less on final loop
10 MHz
2.5 MHz
48 KHz

D. Measuring Sound Performance

To assess the sound performance quality was assessed using a correlation. To quantify the accuracy, the correlation was normalised against the autocorrelation of the true audio file. From this it was found that the 86 order filter had normalised value of 0.2. 39, 87, 125, 199 and 323 order filters implemented in the code showed correlation intuitively correlates with filter order and the execution time scales at a higher rate than sound correlation (see appendix Table II). It was deemed best to adopt a filter of 'decent' sound correlation but fast execution time.

E. Measuring Speed Performance

1) Convolution

Dominant in the code is the convolution function accounting for 82% of execution time. Deciding whether to use standard or frequency domain convolution was decided experimentally. Note: frequency domain convolution was implemented using a pre-built FFT algorithm designed by *Nayuki* [4]. Figure 2 shows execution time for the standard convolution is directly proportional to the filter order in comparison to FFT convolution which increases very slowly. The crossover occurs at an impulse response length of 100 for the RPi 4 hardware which justifies reasoning to use standard convolution when the filter order is 86 as seen in Table **??**. When testing filters of order 500 it was sensible to use the FFT implementation (included in the source code). For justification seen in section I-B, sub-100 order filters boast acceptable sum/diff filtering action. Although larger order filters (>700) are easily implemented with the code and offer near perfect attenuation of non-desired frequencies, the execution-time saving of sub-100 order filters was decided to be adequate to achieve good balance of sound quality of separated channels and fast computation time.

2) Parallelisation

Parallelisation was utilised to reduce the execution time of the code. Parallelisation can be a good optimisation method for DSP systems as they are computationally complex and separated audio blocks can be computed independently. The theoretical improvements for running 4 threads is a 2-3x speed improvement was more than the observed 1.67x improvement. This could be due to the model having to wait for the slowest thread before fully parsing its own thread [5]. By computing separate audio blocks on different threads a higher average CPU and I/O utilisation was realised. Furthermore, the Raspberry Pi uses a 4 core CPU where single thread processes will run only one of the cores[6].

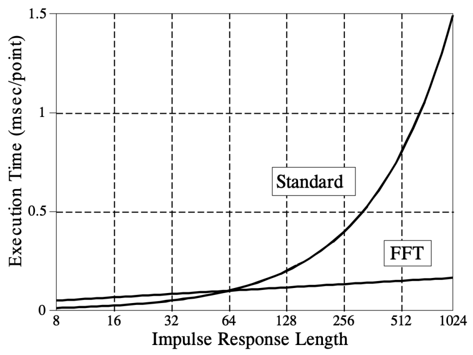


Fig. 2. Execution times for FFT convolution. FFT convolution is faster than the standard 1 method when the filter kernel is longer than about 60 points.

F. Importing input file in blocks

The brief states "no set limit on length of input stream". The code acts on the input samples held in finite, programmer configurable length blocks. Allowing memory allocation to be released for future blocks to utilise. Non-accumulating memory space means the implementation will operate for input streams significantly larger than the host computers on-board memory.

A short-fall of using blocks is the pass-over between sequential blocks. When the convolution based filters reach the end of a block of data, the result becomes skewed from the correct values as information about the next block is not present. Because this pass-over region is small, it is almost inaudible to the listener and therefore acceptable to not amend saving further computation.

The affect of sweeping block size was measured by the execution time the implementation took to run as seen in figure 3. The results shows very small (< 1MB) blocks break the code because insufficient samples in each block disallow convolution between the fixed length filters. Increasing block size beyond 80 MB shows an increasing time relationship likely because the pointer size needed to address such large blocks during convolution becomes slower to handle. Beyond the size of the input stream there is negligible time saving to operate at larger block size. The implementation uses 64 MB blocks for the fastest results.

II. Summary

On reflection the code offers high fidelity without compromising too heavily on fast computational time. Testing shows execution time of 30 seconds on the Pi (or real time on faster PCs).

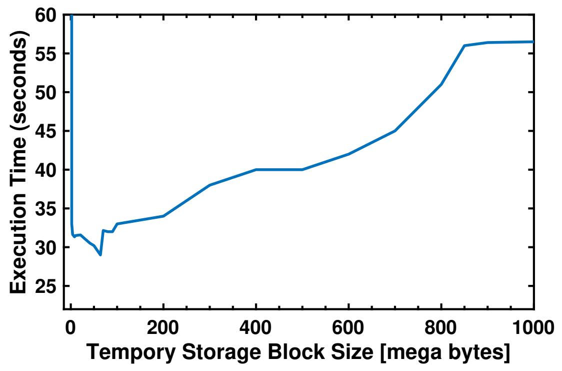


Fig. 3. Execution times for sweep of the 'buffer' parameter computed with quad-threading on a 2 GB, Quad-core, RPi4

References

[1] P. Embree, *C++ Algorithms for Digital Signal Processing*. Pearson Education, 1998.
[2] S. W. Smith, *The scientist and engineer's guide to digital signal processing*. California Technical Pub., 1997.
[3] R. G. Lyons, *Understanding Digital Signal Processing*. Pearson Education International, 2013.
[4] Nayuki, "Project nayuki," 2020. [Online]. Available: https://www.nayuki.io
[5] J. R. Michael McCool, Arch D. Robison, *Chapter 2 - Background*, J. R. Michael McCool, Arch D. Robison, Ed. Boston: Morgan Kaufmann, 2012.
[6] "Raspberry pi 4 tech specs," https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/, accessed: 30-01-2020.

Appendix

TABLE II
CORRELATION BY MATLAB® AGAINST ORIGINAL SOUND TRACK
(SCORE TAKEN AS AVERAGE OF LEFT AND RIGHT CORRELATION)

Order	Correlation Score (relative)	stopband attenuation(dB)	Execution Time (s)
39	0.050	-10	16
87	0.120	-20	20
125	0.140	-30	23
199	0.16	-40	26

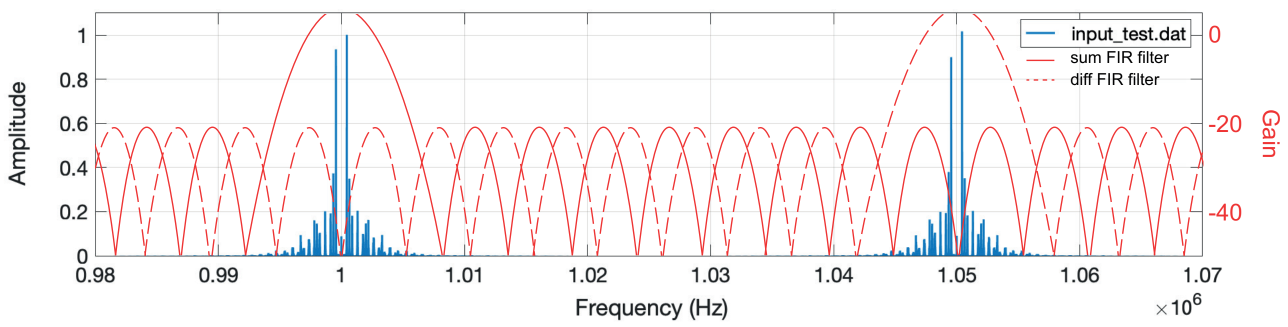


Fig. 4. Frequency plot of input.dat file showing sum and diff and signal centred at 1MHz and 1.05MHz with 20KHz sidebands respectively (blue) | Impulse response of FIR filters shown (red)