Stanford University

Winter Quarter 2015

Vincent Deo

EE 264 DSP Project Report

Audio Compressor and De-Esser Design and Implementation on the DSP Shield

STANFORD
UNIVERSITY

Introduction

Gain Manipulation - Compressors - Gates - Expanders

On top of the many famous time, amplitude or frequeny-dependent manipulations over an audio signal, a lesser known but very common practice in Audio mastering is the use of gain (or signal envelope) dependent effects. Compression -or it's reverse, expansion- allows to map the dynamic range of the input to the one wanted for the output.

In this project, I have been studying and programming possible implementations for a digital audio compressor, and one common application of such a processor, De-Essing, a process by which hard-attacking or sibilant sounds can be removed from audio in real time.

Figure 1 – The features of a common commercial analog compressor.

The gain mapping is defined by **threshold** and **ratio** values (which define the piece-wise mapping as shown in 2). Also, two time values determine very largely the behavior of the compressor, the **attack time** and the **release time**. These time constants caracterise the way the input gain is computed by envelope detection of the rectified input, and although other posibilities exist in digital processing (e.g. block RMS), this analog-like processing offers easy control and smooth to hear results.

Sidechaining and De-Essing

A sidechain is a driving channel for a compressor, but which is not meant to be heard at any point. Technically, the envelope can be computed on any signal, and the resulting compression factor can be applied to any other signal, as the processes are independent.

Given the possibility, we use the sidechain to isolate certain features of the input signal, typically to analyse the energy at certain frequecies. Most commonly, the desired use is to use the sidechain to isolate the excessive energy at high frequencies (5-8 kHz) of sibilant consonants in speech or vocal singing, and then compress the whole signal (broadband de-essing) or the trebles (split-band de-essing) when such a consonant occurs.

By using such processes or their numerous complexified extensions, great improvements in audio quality can be achieved with a bit of fine tuning. During this project and after compression features were up and running, I was able to set up all the standard features of broadband de-essing.

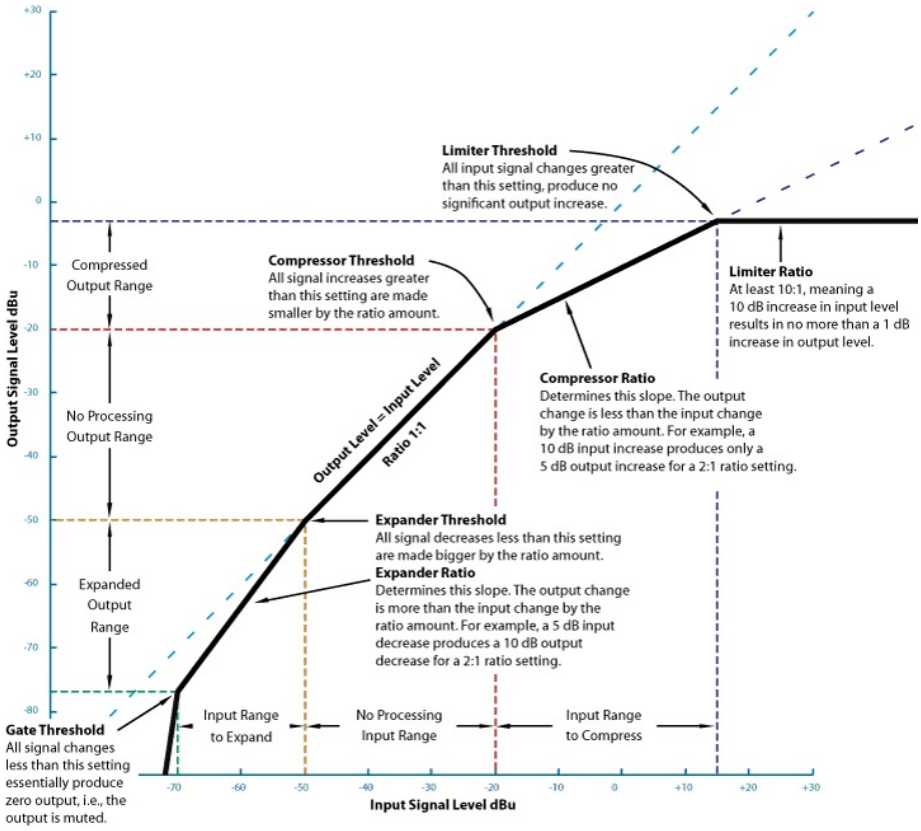

Figure 2 – The different common possibilities in gain mapping, namely from left to right gating, expansion, natural, compression and limitation. Our focus here goes on the compression part, although the look-up table mode (mode 2) allows to write any gain mapping.

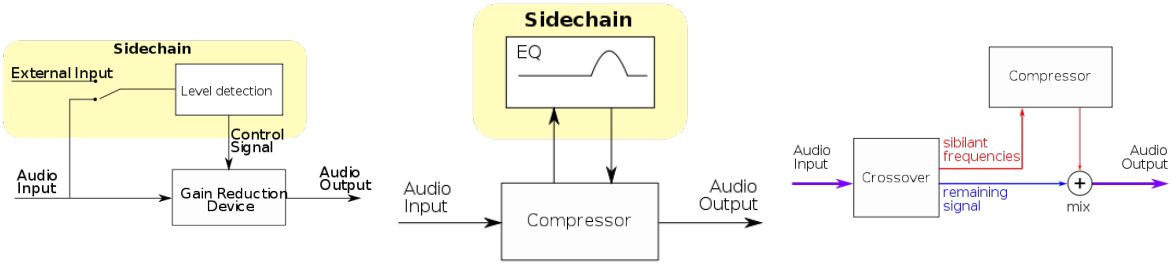

Figure 3 – Top level processing blocks of a sidechain-controlled compressor, in general, and for broad and split band de-essing.

Administrative

Timeline

Tentative Weekly Schedule:

- Week 1: Envelope detection and basic compression features (focus on adaptive gain adjustment)
- Week 2: Complete piecewise dynamic gain control
- Week 3: Standard broadband de-essing by using equalized sidechain. If not too heavy, an adaptive filtering of the side-chain will be aimed for.
- Week 4: De-essing by narrow-band parametric equalization

Actual schedule:

- Week 1: No time to get started, thinking about it a lot.
- Week 2: Envelope detection, try at algebraic compression, look-up table compression.
- Week 3: Corrected and fine sounding algebraic compression, definitive.
- Week 4: Sidechaining, spectral study of undesirable sounds, setup of HW filters.

Compression by dynamic gain adjustment of the Audio Codec, and De-Essing by application of cut-band filters were abandoned as it appeared it was not possible to control quickly and effieciently the Audio Codec from the data processing ISR.
Also, the DAC DRC was studied but not used, in which case this project would just have consisted in setting up the DRC registers as far as its possibilities allow (and which do not seemingly include native sidechaining).

EE 264 Concepts Involved

Lectures:

- Filter design (Elliptic, biquad grouping and Parks-McClellan)
- Random signals, power, envelopes
- Spectrum analysis for the speech frequency investigation

Labs:

- Flow control, real time audio operation, ISRs, Serial Communication
- Fixed point arithmetic for everything
- Audio Codec hardware accelerators for biquad or FIR filtering.

Implementation & Design

Envelope detection

The first key step towards the implementation of a compressor is to determine the input gain level. There are multiple ways to do this, among which I selected to use a rectified signal follower. This technique has the advantage to compute the envelope of the signal with light computations and on a sample by sample basis.

Given the chosen attack and release time, expressed in number of samples as parameters A and R ($\geq$1), the envelope follower follows the following first order equations, where $E[n]$ is the envelope and $x[n]$ the input signal:

$$E[n] = \frac{1}{A}|x[n]| + (1 - \frac{1}{A})E[n-1] \quad \text{if|x[n]|} > \text{E[n} - 1]$$
$$E[n] = \frac{1}{R}|x[n]| + (1 - \frac{1}{R})E[n-1] \quad \text{if|x[n]|} \leq \text{E[n} - 1]$$

Which make $E[n]$ track $|x[n]|$ with exponential behavior with given time constants in the rising or releasing cases.

Even though this is not close to the mathematical definition of an envelope (i.e., a Hilbert Transform), this method provides good results and has some legitimity. Most of the time, attack times used are in the range of 4 - 50 ms, and release times in 20 - 1000 ms. For that case, most of the spectrum lies at higher frequencies than the envelope evolution, which in turn makes the peak level consistently representative of the RMS at such time scales.

Another advantage is this envelope method integrates the effect of attack and decay times right in the envelope detection, rather than having an instantaneous power computation, and then requiring to process again this power value.

On the DSP shield, envelopes are computed in Q15 fixed point arithmetic, audio buffer per audio buffer. A memorized sample is stored within the compressor class structure to transfer the $E[n-1]$ value from the last sample of a buffer to the first of the next one. This is particularly important as the time coonstant may correspond to a large number of buffers.

The DSP shield directly recieves values `att = round(2^15/A)`, and equivalently `rel`; From there values `32768 - att` and `32768-rel` are stored, and used to compute envelope evolution with the least possible number of repeated computations.

Sidechain Filtering

In the sidechain modes of operation, the left input channel is treated as an input channel, and the right input channel as the sidechain. For consistent de-essing, the analog input should be identical on both channel. ADC filters are used to filter the sidechain, on which envelope detection is made. The derived compression factor is then applied to the mono input.

The developed API offers both a sidechain compression mode (mode 4) which copies the compressed mono input to both outputs, and a sidechain listen mode (mode 5), which copies the filtered sidechain to both outputs. This features allows the user to adjust the design of the ADC filters for best results, on top of already having control on the four main parameters of the compressor.

For the design of the ADC filter, the first step was to go through some speech analysis. I made 2-sec long recording of me saying sounds associated with the 26 letters of the alphabet, some of which spectrograms are shown in figure 4. The conclusion of this analysis is that for a relatively low-pitched male voice such as mine, the excess energy caracterizing sibilants (the “ssss” sound) lies in the 5 - 7 kHz band.

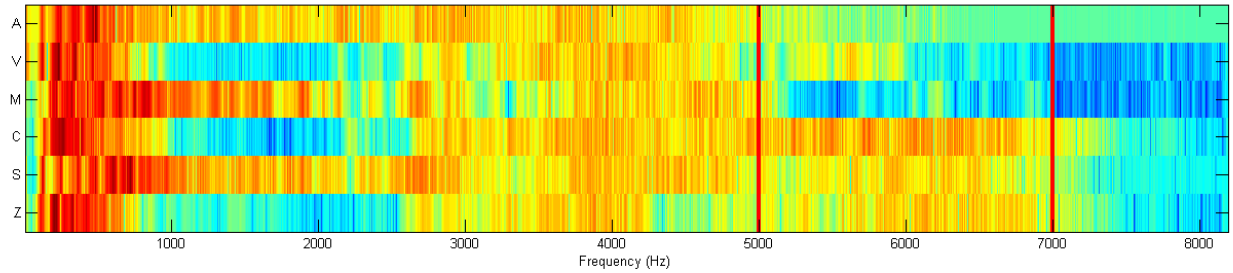

Figure 4 – Spectrograms of some alphabetical sounds. Top to bottom: a vowel (“a”), two non-sibilants (“vee”, “emm”) and three sibilants (“cee”, “ess”, “zhee”).

First, I tried using an 10-order elliptic high-pass to obtain the desired filtering. Using an elliptic filter allowed great stopband gain reduction, however the excessive gain of certain biquads in the factorization introduced hard-clipping in the signal path, which is most undesirable.

I therefore chose to use a 25-tap FIR filter. At 16 kHz sampling, this number of taps is sufficient to provide a cut between 4 kHz and 5 kHz. A filter was designed with matlab using the Parks-McClellan algorithm, then directly integrated into the DSP API code, taking advantage of the adaptive filtering switch capability of the Audio Codec. Sidechain filters (through for left, high-pass for right) are stored at compressor setup, but the adaptive filters are not switched. Then when changing mode from stereo to sidechaining (or reversely), the program just calls an adaptive filter switch on the ADC. At any time, the user has the opportunity to change the sidechain filter to a custom one using command 33.

Figures 5, 6 and 7 shows the properties default filter for 16, 32 and 48 kHz. At 16 kHz, we are only using a high-pass filter, and for 32 and 48 kHz we changed it into a band-pass

(5 kHz - 10 kHz) to avoid possible digital distortion noise from very high frequencies. One can notice that as the number of taps is fixed, the ripples increase with the sampling rate. This is not of great importance as the filtered sidechain is not to be listened to.

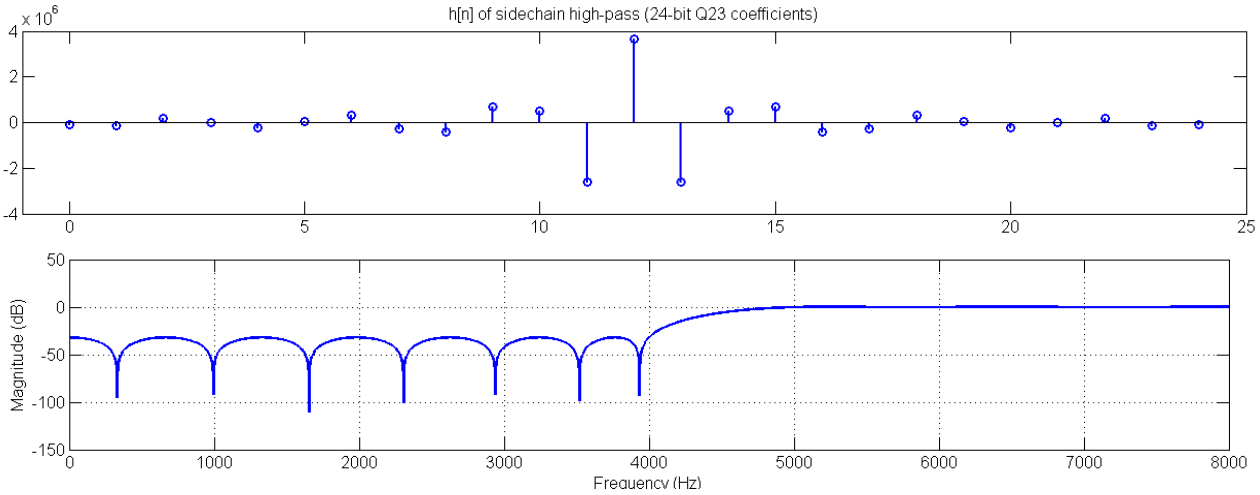

Figure 5 – Top: Impulse response, and Bottom: frequency response, of the default sidechain 25 tap linear phase FIR for 16 kHz sampling.

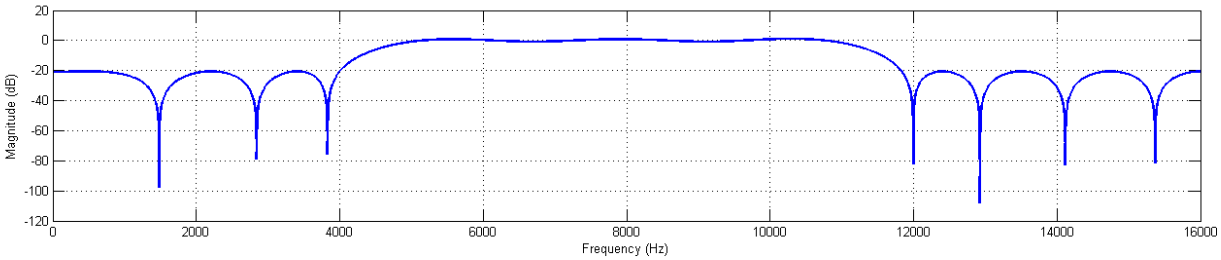

Figure 6 – Frequency response of the default sidechain 25 tap linear phase FIR for 32 kHz sampling.

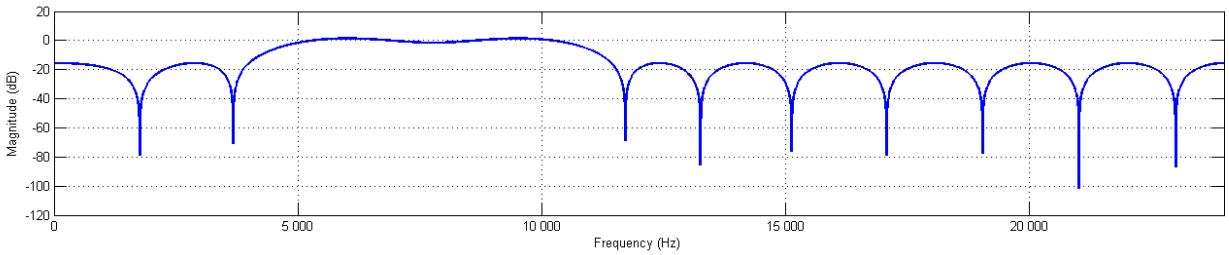

Figure 7 – Frequency response of the default sidechain 25 tap linear phase FIR for 48 kHz sampling.

Algebraic Compression in Fixed Point

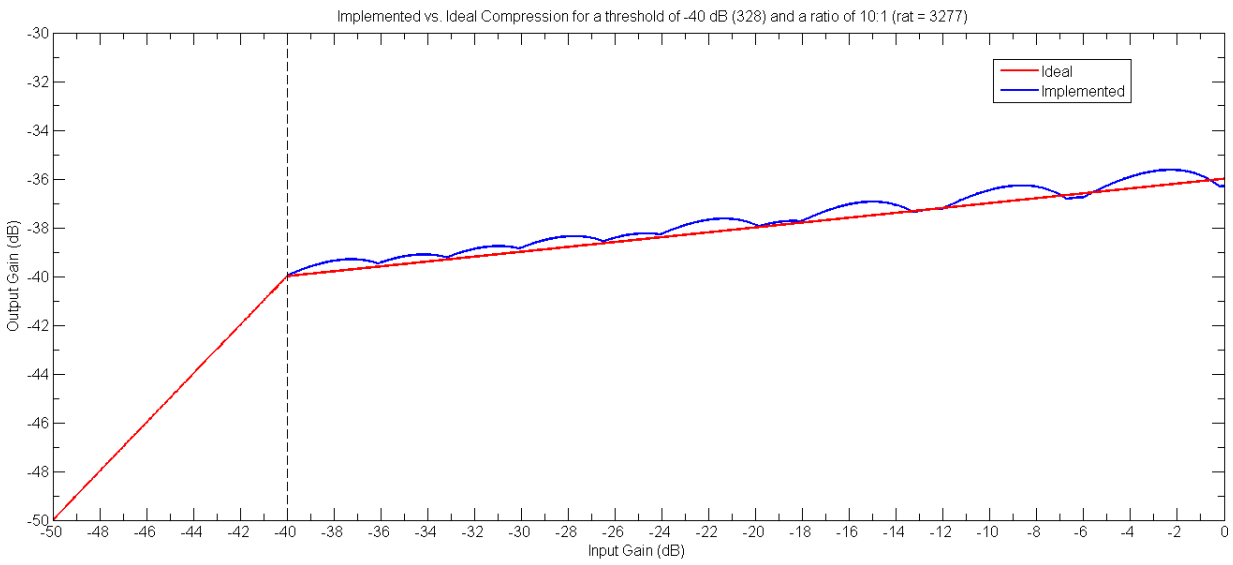

Figure 8 – Ideal and Fixed-point Implemented compression curves for a example case.

First, we derive the theoretical expression of the compression factor, which is the multiplier to apply to input samples values, i.e the ratio between the natural function and the function represented in figure 8, equal to 1 below the theshold and < 1 above.

Let T be the value of the threshold for amplitudes and T_{dB} its value in dBs, related by $T_{dB} = 20\log_{10}(T/2^{15})$. Let R be the value of the compression ratio, ie. $R = \frac{d\ in_{dB}}{d\ out_{dB}}$.

It is to be noted that our implementation stores values $\texttt{thr} = T$ and $\texttt{rat} = 2^{15}/R$, and that these numbers must be in the range $[0, 2^{15} - 1]$. The allowed values are therefore $T_{dB} \in$ [-90.3 dB, -0.0003 dB] and $R > 1.00003$.

The in/out gain relationship above threshold is the following:

$$out_{dB} = \frac{1}{R}(in_{dB} - T_{dB}) + T_{dB}$$

which for amplitudes turns in successively:

$$20\log_{10}(out) = \frac{1}{R}(20\log_{10}(in) - 20\log_{10}(T)) + 20\log_{10}(T)$$

$$out = \frac{in^{1/R}T}{T^{1/R}}$$

hence a compression multiplier C given by:

$$C = \frac{out}{in} = \frac{in^{1/R-1}}{T^{1/R-1}}$$

From here arises the idea to use binary logarithms to compute the exponentiation:

$$C = 2\hat{}\left[\left(\frac{1}{R} - 1\right)(\log_2(in) - \log_2(T))\right]$$

However with the envelope, ie the in gain value possibly changing at each sample, we need an extremely fast and efficienty way to compute binary logarithms and exponentials. Therefore, we implement a piecewise linear approximation of these functions, by mapping linearly between exact powers of two.

If n is a positive 16-bit signed integer, the floor of the binary logarithm of n is the position of the most significant 1 in the binary decomposition of n. A linear interpolation of $\log_2(n)$ between $\lfloor \log_2(n) \rfloor$ and $\lceil \log_2(n) \rceil$ is made by taking all the lower weighted bits of n and shifting them as the decimal part of $\log_2 n$. Of course, a truncation error is possibly made when these bits are right shifted, and a flooring error when they are left shifted and padded with zeros. In any case, such a function does an efficient job as it can be programmed with a very quick loop and bitshift operations only (see `binlog` in the code documentation). The interpolation is shown in figure 9, and the approximate computation overall is shown in figure 8.

Binary logarithm of 16-bit integers are stored as 16-bits in a Q11 manner. The 4 highest weighted bits after the sign bit store $\lfloor \log_2 n \rfloor$, and the 11 remaining bits store the shifted decimal part. Reversely, we implemented a `binExp` function that does the reverse operation.

Finally, the operation implemented is not exactly the one written above, as we wanted to keep sure the arguments of the functions were positive at all times,using the positive value $1 - \frac{1}{R}$ rather than its opposite.

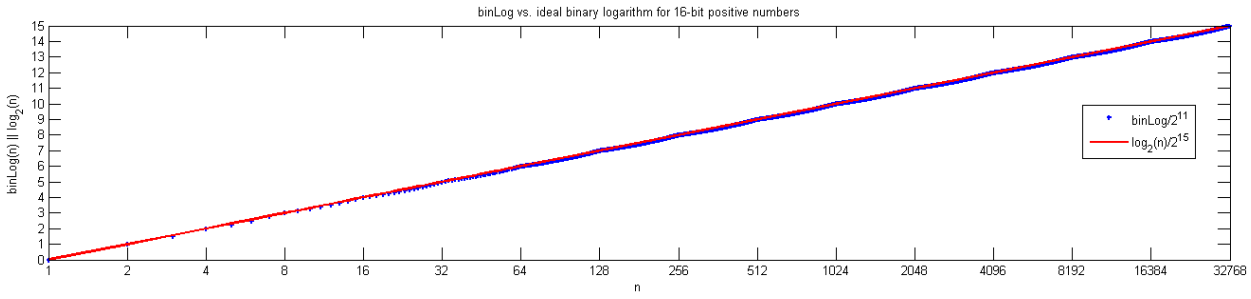

Figure 9 – Results of the binLog interpolation vs. the exact binary logarithm.

Results

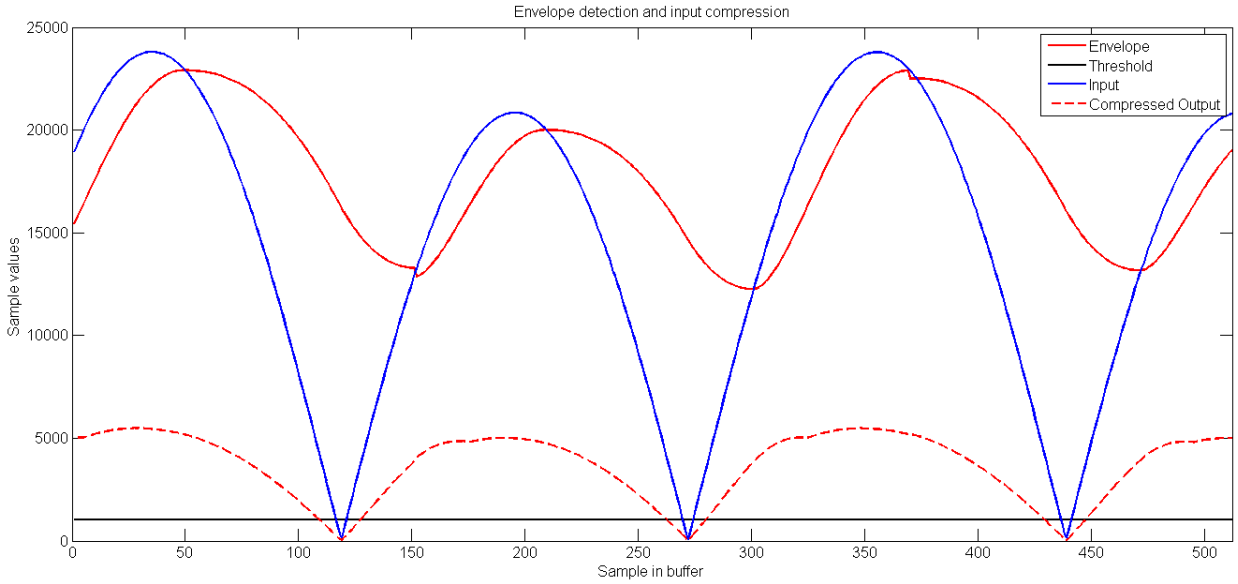

Figure 10 – Envelope detection and compression example over a sine wave of frequency 50 Hz at 16 kHz sampling. The attack time is 1 ms and the release time 5 ms. The threshold is set at sample value 1036 and the ratio is 2:1.

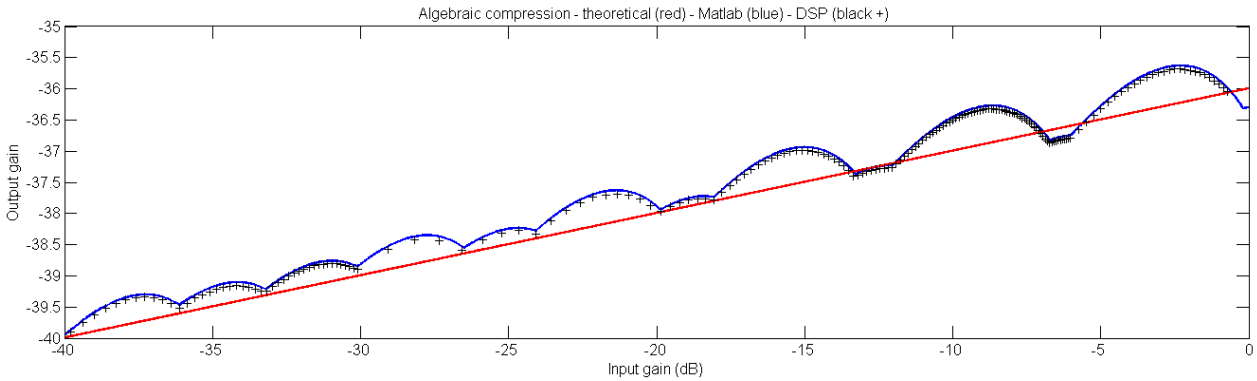

Figure 11 – Compression between input and output using approximate binary exponentiation as computed with Matlab (blue, solid), and on the DSP shield (black +). Rouding errors make a small difference between computations. The rounding errors are much more critical at low envelope value (<500), but such cases do not arise in practice.

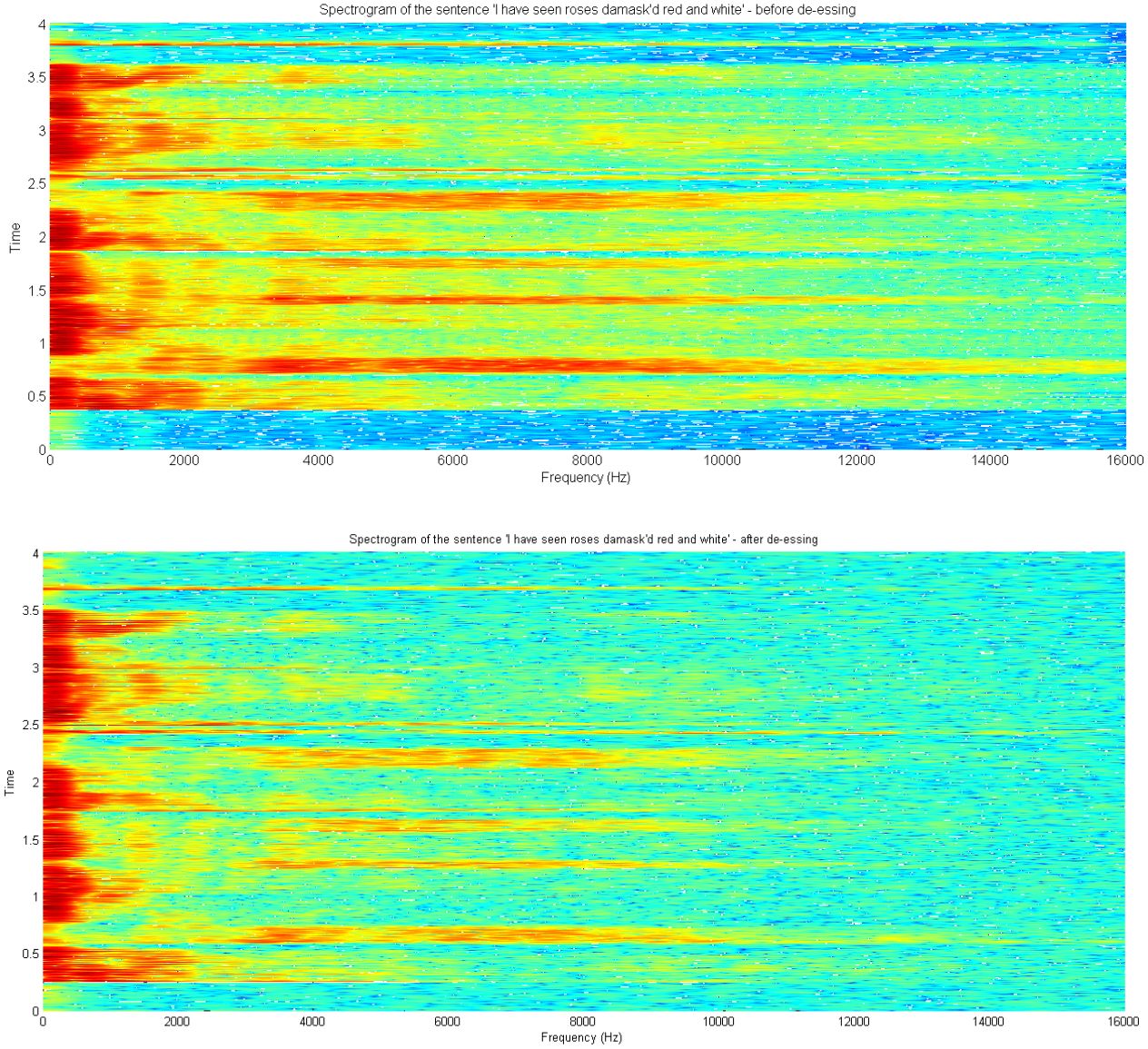

Figure 12 – Spectrogram of the sentence 'I have seen roses damask'd red and white' before and after it goes through the DSP shield in de-essing mode. It can be seen with attention that the relative amplitude of formants including a strong > 5 kHz amplitude is reduced compared to vowel sounds.

Unsolved challenges during the project

Coming back on a couple of difficulties that have arisen during the project, I would like to mention the following:

- At first, I planned to do the sidechain filtering with an elliptic filter factored into 4 biquads. As it happened, factored biquads had high overshooting ripples closed to the band edge, and introduced a lot of clipping. A lot of distortion on the sidechain is completely acceptable, but unfortunately not clipping and therefore IIR filter design was abandoned for this purpose.
- Fixed point arithmetic introduces a singularity during the computation of the envelope with small `att` or `rel` values, corresponding in real case for > 40 ms release time. What happens is that a given value is stable for the computation and the envelope is stuck there. To overcome that issue, a force release was added if the envelope is not computed to change when it should.
- At some point, I also wanted to implement the compression in semi-hardware, by changing dynamically the output gain depending on the envelope. This had to be abandoned as well, as the I2C bus used to communicate with the codec is not reactive enough, and is unstable when called during an ISR.

Conclusion

This project (and the labs before) has been an amazing opportunity for me to discover audio DSP implementation, through this choice of de-essing feature I made. Overall, I consider this project to be a great success, even though the de-esser at this point in time only implements a basic broadband de-essing, but which can already be very efficient with a bit of fine tuning (and probably analog knobs would make that extremely faster). The compressor part on its side works smoothly even at 48 kHz rate, making the low processing cost decision worth the slight non-linearity in gain.

Finally, there is a variety of features that I'd like to add to this de-esser, and also a variety of audio effects that'd I'd like to implement in the future. The DSP Shield is a great tool for that, which is why if possible I'd like to keep it!