# MUMT-307/501 Final Project Report

David “Graham” Smith

04/22/2019

# Intro

The aim of this project was to explore how dynamic range controllers work in a more detailed fashion. Out of all the different types of audio effects I've always found these to be the most interesting due to their complexity. The basics of the different effects like equalizing, chorus and reverb were always fairly easy to understand in terms of their behaviour and uses. Dynamic range controllers on the other hand seemed quite esoteric at first and their behaviour and uses were not immediately obvious. Dissecting one to see how it operates seemed like a good idea to mitigate this.

Initially the project started just as an implementation but a larger questions began to stem from this. How do I verify my implementation and is my design correct? Is what I consider "correct" the same as what an audio engineer or musician would look for in a dynamic range controller? Are the desirable properties of the different types of dynamic range controllers the same or does it vary from compressor to expander (beyond the obvious functional differences)? Considering all the options was a bit much given the time constraints, so this aspect of the project was limited to strictly compressors.

This report is intended to be read with an accompanying set of Matlab scripts. Run the script *main.m* and follow the prompts.

# Theory

What does a dynamic range controller do? Well, as the name would imply it attempts to control the maximum and minimum levels a signal can take. This is typically achieved through a time-varying gain function whose gain value is determined based in the input signal of the level. Dynamic range controllers come in two varieties: compressors and expanders. Compressors are used to reduce the dynamic range of a signal while expanders are used to increase the dynamic range of a signal. These can be configured into their extreme forms called limiters and gates. A limiter doesn't allow a signal to go beyond a specified level while a gate silences sounds below a certain level.

This implies that there is a there is some level detection going on in the controller. Indeed there is, and there are several different approaches to monitoring the level of a signal. They range from simple peak detection algorithms using the absolute value of a signal to more complicated methods liek computing the RMS value of a signal. In practice however the RMS computation can introduce significant delay so approximation techniques can be used.

Dynamic range controllers can come in two topologies: feed-forward and feedback. Each has their own advantages and disadvantages depending on the application and whether the implementation is digital or analog. Feedback topologies consider the level of the signal after the gain has been applied. Feed-foward systems perform the level detection on the unaltered input signal. For a further analysis of the different advantages / disadvantages to each variation any of the sources listed in the bibliography section should provide you with sufficient information.

The *Threshold Level* is the level at which the dynamic range controller will activate. The *Ratio* parameter determines the amount off attenuation to be applied to the overshoot of the signal. The *Overshoot* is the amount of the signal which goes over the threshold. Controllers can either have a

“hard-knee”, where the transition region around the threshold is immediate, or a “soft-knee”, where the transition is more gradual. The *Width* parameter determines the width of the knee (in the soft-knee case) which allows the transition from above and below the threshold to be achieved more naturally. This is generally achieved through interpolation (of which there are many varieties).

One of the important characteristics of any piece audio equipment is that it doesn’t add any sort of undesired clicks/pops or distortions to the signal being processed. These generally result from discontinuities and rapid changes in signal values. In order to prevent this from happening, dynamic range controllers have *Attack Time* and *Release Time* controls. The values adjust the growth and decay rate of smoothing functions internal to the controller. Where this smoothing is applied in the signal chain as well as how it is applied is another thing to consider. Control of these parameters is crucial to eliminate artifacts like “pumping” and “breathing”.

For the purposes of this report the *Attack Time* and *Release Time* are based on the rise time being considered to be the time it takes the function to go from 10% and 90% of it’s final value (or 90% to 10%). Additionally we only implement a feed-forward design with smoothing applied in the level detector pre-dB conversion. What these terms represent is something that a lot of the literature disagrees upon.

# System Characteristics

The following systems analysis is specific to the compressor architecture implemented in the report.

## Linearity

The system is non-linear. Consider the scenario where you have signals $x_1[n]$ and $x_2[n]$ which are slightly below the threshold value of the controller in question. Each signal will pass through unaltered since the device will be operating the in region where no attenuation is applied. However, The sum of the signals though would push the total signal level to be above the threshold causing controller to activate and alter the input’s level.

## Memory

As smoothing is implemented using a 1-pole filter this means the system requires memory to operate. The gain function and the rest of the system’s components are memory-less however.

## Stability

The system is BIBO stable as the gain function is memory-less and only depends on the current time instant. From that and the static characteristics of the gain function we will always know what the output will be once the attack time has passed.

## Causality

The system is causal as future values are never considered, only current and one sample in the past.

### Time-Variance

Whether the system is time-invariant or not depends on what the attack and release times are set to. In the extremely unlikely event that both of them are 0 seconds, then the system would be time-invariant. However it most applications this is almost never the case and the system will be time-varying. Consider the scenario where an incoming transient hits the input of the controller before the release time has fully had a chance to decay from the attenuation applied to the previous input. The level of the new signal will be altered. Had the system been completely "at rest" when the new signal signal was applied, the resulting output would be much different.

# Design Considerations

In the following sections the process of determining which algorithms to use for the sub-components will be discussed. In general, the main criteria used was based on the ease at which the sub-components behaviour could be objectively verified. This criteria was established to make sure the behaviour was fully understood in a way that didn't depend on having "golden ears".

## Level Detection Method

Three different options were considered for this: a simple peak detection algorithm, RMS and an RMS approximation algorithm. The RMS algorithm was deemed to be unusable as it was hard to tune the window value to match the behaviour of the other signals in terms of attack and release times and make it comparable. The "so-called" RMS approximation proved to give results extremely similar to the peak detection algorithm but it proved to be difficult in terms of objectively verifying that the attack and release times were properly set. It was for these reasons, and the straight-forwardness of verifying the peak detection algorithm, that it was chosen to be used in the design. Time-permitting an implementation of each algorithm would be developed so their behaviour could be compared but this can be saved for future work.

## Gain Function

Implementing the gain function was relatively straight-forward. The sources listed in the bibliography contain the requisite mathematical equations for implementing the dynamic range controllers in code. One thing to watch out for was how you calculate the dB values and take into consideration the bit-depth of the system you're working on. When you have an amplitude value of 0 in Matlab, the dB formula returns a value of -∞. This proved to making graphing and verification difficult so these values are capped using the formula $-\text{floor}(20*\log_{10}(2^{Q})$, where Q is the bit-depth of the system (assuming PCM with uniform quantization steps). The system was chosen to be 32-bit based on previous experience with such systems.

Another important characteristic of the gain function is how the knee is calculated. To keep things simple, parabolic interpolation was used. In the future, this is something that could be explored more fully.

# Verifying Correct Behaviour

One of the most important parts when designing and implementing any system is to make sure what you've built actually works. This is especially true in DSP applications where it's not always so clear that a system is working correctly (as compared to verifying a calculator's functionality). The entire job of an Audio Validation Engineer is to provide objective evidence and measurements that an audio system has been properly implemented.

The overall testing strategy was unit test all the individual functions for a range of values first. After ensuring their correction operation, the overall system's characteristics would be verified but aiming to treat it as a black-box. Eventually errors were found though where this condition had to be violated.

## Motivations for Testing Strategy

There are several reasons why this testing strategy was chosen. The first is that by ensuring all the individual components are working correctly first, a lot more insight into the operation of the compressor can be gained in general. The second is that it allows you to more quickly establish where an error occurred. By ensuring the individual parts are all functioning correctly, it allows you to more quickly diagnose that something went wrong in the "integration" process.

A third motivation is that this type of "black-box" situation can often occur in the real-world. Suppose that you have a compressor that you want to attempt to "clone". In order to do that, you would need to adjust its parameters and send a variety of test-signals through to probe it trying to establish it's behavioural characteristics. I've personally been in this scenario. I worked for a company that hired another company to configure various DSP aspects of an audio system and my job became to ensure we got what we paid for. A compressor was one aspect of the system.

A fourth reasons stems from the original plan. The original (ambitious) goal was to develop a suite of sub-functions which could be used to implement the different types of dynamic range controllers. A general model would be developed for the various topology configurations which could then be modified using function handles to change the model's behaviour. For example to change the system from a compressor to an expander, you would change the function handle for the gain calculation. To change the level detection to use the RMS approximation algorithm, you would change the corresponding level detection function handle. Unfortunately, time did not permit this to be fully realized. As a result, there several unit tests for an expander's functionality but the gain function didn't get tested in a full system. However, their development did provide an interesting contrast to a compressor's and as a result more insight was gained into the operation of both.

## Unit Testing Dynamic Range Controller Parameters

It was important to determine which order to test the different parameters for the compressor as their operations are coupled together. The threshold was the easiest to test and since the other parameter's behaviours are defined by it, it was chosen first. Then the ratio was measured for a system with a hard-knee. After those two were tested the knee-width parameter was verified to be operational.

The attack and release times are implemented in a separate functional block than the gain calculations so their order doesn't matter relative to the other parameters.

### Threshold

Creating tests for the threshold parameter was straight forward. In order to facilitate ease of verification a hard-knee was chosen and the ratio parameter was kept fixed. This way the verification could be done with a simple visual inspection ensuring that the gains changed at the correct dB level.

### Ratio

With the threshold behaviour verified I could now more easily confirm correct operation of the ratio parameter. To facilitate ease of verification in this case a hard-knee was used as well as a fixed threshold value. Visual confirmation became more involved though as it involved a few mental calculations to ensure operation was successful. Additionally, Matlab had trouble plotting the curve for when the expander operated as a gate (i.e. $R = \infty$) so this curve is not displayed in the plot. The values were verified by examining the function output though.

### Knee Width

With the two preceding components now verified I could move on to ensuring that the knee-width was being correctly calculated it. This was once again done though visual inspection by plotting the static characteristic gain curves while keeping the other parameters fixed. Proving that the actual numerical calculations involved here are correct is a rather difficult task so verification (without effectively rewriting the same code again) was limited to ensuring that the interpolation worked by verifying the curves with soft-knees intersected the hard-knee curve at the dB value half the width away. For instance with a threshold of -30 dBFS and a width of 4 dB, the soft-knee curve should intersect the hard-knee curve at -32 dBFS and -28 dBFS.

### Attack/Release Time

Confirming correct operation of these parameters was done by running a step function through the peak detection algorithm. As the step-function is limited on the range 0 to 1, verification was a simple as finding the points on each curve which corresponded to .1 and .9 on the Y-axis and taking the difference of the corresponding X-values to calculate the times. One crucial aspect to note is that the duration of the step response has to be long enough to allow the different curves to meet those rise and fall to those points.

## Testing Complete System Behaviour

With each of the sub-components now verified, I could construct the entire compressor and attempt to characterize its behaviour. This proved to be an interesting task as it really tested my knowledge of the operation of compressors. Only a single value for each parameter was tested as the previous unit testing covered a range of parameters.

## Measuring Threshold

Measuring the threshold was a fairly straight-forward task. Set the knee width to 0, keep the ratio constant and set the attack and decay times to 0. Then, instead of using a step function, use a level sweep as a test signal. The output should diverge from the input curve at the threshold point. There is however the case where the compressor you're dealing with doesn't support zero values for the attack and release times. This scenario is more complicated as the attack time will cause input to delay when it crosses the threshold and the attenuation beings to apply. The resulting curve isn't as easy to analyze. Future work can be done in developing a verification method for this scenario as it is a situation that is likely to occur.

## Measuring Ratio

Measuring the ratio was fairly simple and followed a similar manner to the threshold method just described. However, in this scenario the impact of the attack and release times didn't matter as we were interested in the attenuation amount above the threshold. From there it was a simple matter of comparing the input/output values (taking into account the threshold value) and making sure the overshoot was properly attenuated.

## Measuring Knee Width

Determining the knee-width as correct was a bit more difficult here. In a more traditional black-box setup, I would generate an idealized curve and compare the output of the compressor in response to a level sweep. There would be a certain amount of acceptable tolerance to determine whether the parameter was in bounds or not. In this case however I would essentially be rewriting the math in the compressorGain() function. Instead, I took a similar approach to the unit-testing where I checked to see how the output curve behaved at the boundaries of the knee. At the lower boundary the knee is at the intersection between the two curves. At the upper boundary the knee is at the point where the interpolation ends and the full ratio value is used. Verification was done by checking that the ratio was correct in the "linear" region of the function (where the parabolic interpolation ends).

## Measuring The Attack/Release Time

This part of the verification process was the most difficult. The test setup involved keeping the other parameters constant and using a step signal with levels set so when it's "off" the threshold isn't triggered. It seems to fundamentally be a very difficult task to measure the attack time precisely due to it's nature. As the attack time slows the rise of the peak detection function, the actual input level will cross the threshold before the gain function has a chance to activate and attenuate it. This artifact was represented as "plateau" in the step-response output signal (as shown in the attached Matlab scripts). Additionally, the duration of the step-signal needs to be long enough such that it allows the rise-time to let the function get high enough to cross the threshold.

In contrast, the release time was fairly easy to verify. It's interaction with the threshold value doesn't create any sort of a "charging-time" so to speak. It was just necessary to make the test signal stay below the threshold long enough for the rise time to complete.

This helps illustrate how crucial the smoothing function(s) placement is. If smoothing was placed after the gain function instead it might be possible to measure the attack time more accurately as there wouldn't be any coupling between the attack time and when the threshold was triggered. A system with smoothing functions in the level detection, as well as the gain calculations, would prove to be very challenging to verify. Especially if the values were constrained and not capable of being set to 0. Further work could be done in evaluating the effects here and how to properly verify them.

# What is considered "correct"?

There are many objective ways using various signals to capture the behaviour of a system. These are all fantastic from the standpoint of the engineer creating the tool as they allow them to ensure the proper operation of their device. There are various other places where dynamic range controllers can be used beyond the recording studio (i.e. automotive audio systems, cochlear hearing implants, radio transmission systems) and having a strict set of objective criteria to meet is often available in those situations.

In the music production world things are a bit different. Non-linearities and non-ideal behaviour are often sought after for the unique musical characteristics these aspects of a device bring about (just think of guitar distortion). Given the intended audience and how they use of the dynamic range controllers more creatively, I think the true measure of success (beyond any fancy engineering graphs) is whether or not an audio engineer/producer would want to use the tool.

That begs the much larger question though, of what is it that audio engineers and musicians are looking for in a compressor anyways? In order to find out I spoke to a few and the results varied quite a bit. One wanted a "transparent" compressor to keep a signal "under control". In this case he specifically wanted something for the utility of dynamic range control to avoid constantly adjusting a fader to maintain a signal's place in the mix. The "plateauing" effect might have a negative effect here. Another said he didn't care about the utility at all and specifically used them for the colouring effect they have on a track. In a scenario like that, the "plateauing" effect might be considered beneficial. How is it you translate these ideas into a design? In the ideal case, where the compressor operates in the manner in which they were originally designed, that is easy. But what topologies are necessary to create a desirable colouring? What set of parameters will make creatively inspire an audio engineer? These are much larger questions. A great deal of work could be done in the future to more properly answer them.

# Bibliography

Pirkle, W.C. 2013. *Designing Audio Effect Plug-Ins In C++ With Digital Audio Signal Processing Theory*, Chapter 13. Burlington, MA, USA: Focal Press.

Reiss, J.D. 2015. *Audio Effects: Theory, Implementation and Application*, Chapter 6. Boca Raton, FL, USA: Taylor & Francis Group

Zölzer, U. 2011. *DAFX—Digital Audio Effects*, Chapter 4. West Sussex, U.K.: John Wiley & Sons.