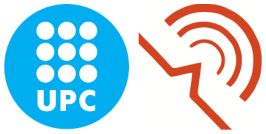

Universitat Politècnica de Catalunya (UPC)
Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona (ETSETB)

Implement a chorus effect in a real-time embedded system

by

Gerard Amposta Boquera

Advisor: **Asunción Moreno Bilbao**

Barcelona, September 2016

"I was left with an urge to make the guitar sound like things it shouldn't be able to sound like."

Adrian Belew (1949 -)

Universitat Politècnica de Catalunya (UPC)

Abstract

Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona (ETSETB)
Departament de Teoria del Senyal i Comunicacions

by **Gerard Amposta Boquera**

An implementation of a chorus effect in an embedded device has been devised. Some objective and subjective qualities of this implementation have been analyzed, with the purpose of being tested for their adequacy for usage in the music industry. These qualities include, but are not limited to: maximum delay, number of delayed copies of the original signal, waveshape of the signal modulating the delay, dependency of parameters of the modulating signal from parameters from the input signal, adaptability of the architecture to accommodate further effects without increasing significantly resources usage. It is encountered that a scientific approach to this study has either not been taken, or the results of this research are privately held, therefore being adequate to start this research for the public interest. The system has been implemented in a dsPIC33FJ256GP506 digital signal controller using a standard development board, and results show that this is a promising approach that could be industrializable using audio-grade components for the analog stages and a slightly more powerful DSC.

Acknowledgements

When my family and friends finally managed to overcome my stubbornness and convinced me to put it together and write this thesis, I knew what I was heading into was though. What I happily ignored at that moment was *how* though. This last year has been a challenging *tour de force* on juggling with my time: time for work, time for the thesis, time for resting, when feasible, and all while trying (and failing) to avoid losing contact with the same family and friends that talked me into writing my thesis. It's been some difficult times, but I also learnt a lot, so I am grateful that I actually did it.

None of this would have been possible without the help of a lot of people. First and foremost, I would like to thank my advisor, Asunción Moreno, for her time, her patience, her wise advice, her kindness, and specially for stepping up at crucial times, steering the ship in the right direction when it was necessary. For the very same reasons, I would also like to thank Antonio Bonafonte, whose early insights about the project were incredibly helpful.

A large share of the gratitude that I owe for being able to finish this thesis is due to the sole person that has been with me every single day during this period: thanks Montse for your kindness, patience, love and understanding. I also need to thank all my family, specially Mom and Dad, for their unconditional support.

Is difficult to go through this without your peers comprehension, so is lovely to be able to count upon friends who have gone through the same as you to for some morale boosting. Thanks Marc, Jordi, Marc, Víctor, Miquel, Dani, Jesús, Albert, and all of you guys, for being there during the last 10 years, and for the years to come.

Since this is, in the end, about music, I would also like to thank my "road family", for allowing me to skip rehearsals and such. Thanks Manel, Esteve, Josep and Maurici for making me laugh like crazy during that last year every single time we met.

Being now a quite experienced engineer, I have had the opportunity to observe in others how determinant is the influence of senior engineers in juniors, and the importance of blossoming in the right environment, and picking the right role models. For this reason, although they have not been linked to this project whatsoever, I would not feel right finishing this acknowledgements section without thanking Sergio Casanova and Oriol Solà. I feel fortunate to have had the opportunity to grow up with them both personally and professionally, and I hope I can keep learning from them for the years being.

And, well... the little furry dudes can't actually read, but their unparalleled comforting abilities have helped me to keep my sanity during those very stressful times, so thanks Neko and Pam.

This page is intentionally left blank

Contents

List of Figures

List of Tables

Acronyms

AGC Automatic Gain Compensation. 23

CLT Central Limit Theorem. 33

CPU Central Processing Unit. v, 6, 26, 27, 34

DAW Digital Audio Workstation. 3

DCI Data Converter Interface. 28, 29

DF1 Direct Form 1. 25, 41

DSC Digital Signal Controller. ii, 29, 30, 41

DSP Digital Signal Processor. 25, 26, 27, 28, 29, 30, 32

FFT Fast Fourier Transform. 25

FIR Finite Impulse Response. 16

FPU Floating Point Unit. 25, 26, 28, 29, 30

HAL Hardware Abstraction Layer. 6

HW Hardware. 25, 26, 27, 28, 29, 32, 41

I2C Inter-Integrated Circuit. 28

IDE Integrated Development Environment. 25, 26

IEEE Institute of Electrical and Electronics Engineers. 46

IIR Infinite Impulse Response. 16, 18, 21, 25, 30, 31, 34, 41

LFO Low Frequency Oscillator. 5, 10, 11, 13

LPF Low Pass Filter. 20

LSb Least Significant Bit. 33

LUT Least Significant Bit. 29, 30

MAC Multiply-and-Accumulate. 25, 26, 28

MCU MicroController Unit. v, 2, 24, 25, 26, 27, 28, 29, 30, 33, 34

PC Personal Computer. 39

RNG Random Number Generator. v, 11, 13, 15, 19, 33

SIMD Single Instruction, Multiple Data. 26

SW Software. 28, 29, 33, 34

TI Texas Instruments. 26

TRS Tip, Ring, Sleeve. 25, 29

VST Virtual Studio Technology. 2, 3, 39

Dedicated to Montse, for her support, dedication and patience during this journey.

Chapter 1

Introduction

1.1 Motivation

Short and clear, the motivation for this project is to conjugate my passion for embedded systems programming and digital signal processing with my passion for music.

I am myself a regular user of chorus and other effects units. When using them, I often wonder about the design choices taken on their design. Why the range of this parameter is constrained to those values? Why did not they include a low-pass filter in this side-chain? Why there isn't any analog input to allow for dynamic variation of effect parameters? Those are frequent questions that pop up on my mind.

Sometimes the reasons may be based upon engineering and component costs. Sometimes they are based upon ease of use for the average musician or live performer, or upon suppositions about what musicians really and really don't want. Sometimes the reasons are as illogical as "it has always been done this way". But, more often than not, the reasons for keeping it simple actually hold right - nobody really wants to accidentally dial their settings to extreme values while performing live.

Nevertheless, sometimes you just need to get into the wilderness of sonic exploration. While trying to arrange some passage that it has been resisting you for so long. When working on production on the studio, searching for *that* texture that is on your mind, but not on any of your instruments. When you don't know how to get back from you "C" part to the bridge. Those are the times when a little dose of original, unnatural sounds can save a production, giving the touch that is unmistakably tied to that song for times to come.

My goal is to provide musicians, arrangers and producers with an effects unit that truly enhances their creativity, offering possibilities that they may had never thought about

before. I want them to decide where the limits are, and to avoid assuming anything about what they might want. To this end, a no-limits approach has been taken in designing a chorus unit that tries to improve on previous pioneering units efforts, while maintaining simplicity in the measure that is possible.

To understand why did I chose to implement an effects unit in an embedded system instead of just integrating it in a VST package, one needs to look back at my professional career. If there is a single discipline that I might be enjoying even more than music, this is embedded systems design and development, and I have been fortunate enough to have spent most of my career tied to microcontrollers and the friends. Therefore, I did not want to miss the chance to get into yet another embedded project.

Working with embedded systems often consists on doing more with less, and knowing how more you can do with the less you have is an integral part of system dimensioning. Therefore, as much exciting as it can sound to implement a phase vocoder or a fast convolution based reverb, practical experience dictated that I should settle for something more modest for this project, to avoid putting too much strain on the microcontroller to the point that it is not able to perform the required operations per sample.

A chorus unit is simple enough to be conceptually easy to understand, its implementation in digital systems is quite streamlined [1], and trades processing power requirements for memory requirements, a trade-off that most MCUs of today will gladly take. Do not be deceived into thinking that this is a boring option, since chorus effects still can pack an array of wildly modulated sounds if let go rampant, and the implementation of the required modules to make it work is less trivial than it may seem initially.

Finally, when you decide to work in an audio project, is obvious that some digital signal processing is going to be necessary. For me this had some special implications, since I had wanted to be an audio engineer during my undergraduate years, and I only abandoned pursuing such a career after I started growing fond of embedded systems design. When I started considering doing an audio project as my thesis, I thought that this would be a nice way to provide some closure to that era.

So, here it is. Music. Embedded systems. Digital signal processing. I hope that the reader finds this piece of work enjoyable, as much as I did while writing it.

1.2 A brief introduction to music effects

Music effects are systems that take an audio signal as an input, modify it according to some sound control parameters, and output the modified audio signal, with the aim to enhance or confer a desired musical property in the audio signal. It is generally accepted that modifications usable in audio signals fall into one of the following seven categories: signal distortion, dynamics alteration, filtering, modulation, pitch modification, time-based modifications, and output feedback.

Distortion is the effect of increasing the input gain of the signal to the point that it starts saturating to its maximum value, creating inharmonic overtones and compressing the sound. It is very appreciated by electric guitar users. Overdrive, distortion and fuzz are common distortion-based effects.

Dynamics alteration refers to altering the output level of certain input passages according to some control rule. When this control rule is frequency-based, it is known as filtering. When this control rule is amplitude-based, it is known as dynamics alteration. For instance, compressors or noise gates are popular dynamics effects, while equalizers and wah-wah's are popular filtering effects.

Modulation effects are those effects based on the multiplication of one generally independent signal to the input signal, and are the main topic of this thesis. They can be modulation-only effects, such as tremolo, or modulation combined with another effect, such as chorus, vibrato or flanger. Nonetheless, when one effect is the result of different effects combined and at least one of them is a modulation effect, the global effect is usually considered to still be a modulation effect.

Pitch modification effects produce a frequency displacement effect in the input signal across all the signal duration. Harmonizer effects are the usual example. Pitch correction software such as Auto-Tune could also be considered to fall into this category, although in pitch change modification effects the frequency displacement it is applied in a selective manner only across certain sections of the input signal.

Time-based modifications include all effects based on storing samples of the input signal and mixing them back later. Echo effects, loopers, and room simulation effects like reverberations ("reverbs") usually fall into this category.

Feedback effects are produced when enough positive feedback is introduced in the audio path to create a sustained oscillation without excessive increase of volume. EBow is a popular application of this kind of effect.

Music effects are usually commercialized in four formats. Stompbox units are stand-alone units intended to be placed on the floor, and to be activated or deactivated using a foot switch. Usually they are able to be battery-powered, and don't include too much configuration parameters: ease of use is prioritized. Rack-mounted units are designed to be screwed into the 19-inch rack standard in telecommunications, computing and music industries, are always outlet-powered, and tend to be more complex or more flexible than stompbox units. Built-in units tend to be simple and inexpensive effect units built into instruments or amplifiers. At last, VST effects are pieces of software designed to be used from within a DAW, and tend to be used only in recording studios.

Music effects are extremely popular in studio settings and are ubiquitous for use in instruments that produce an electric output, such as the electric guitar. There is virtually no recording that goes to the market without going through some extensive effect-applying stage, so this is a field that is destined to grow as long as the music industry is still alive.

1.2.1 A brief history of music effects

Music effects date back from the late 1940s, when pioneering recording engineers started manipulating reel-to-reel recording tapes to create echo effects and microphone placement techniques to simulate echo chambers. Back then music effects were solely a studio thing, since most stand-alone units were expensive, bulky and impractical to use in a live setting of the time.

By 1950, tremolo, vibrato and reverb were available as built-in effects in some guitar amplifiers, using mechanical components such as springs. In addition, some musicians started to experiment with distortion, over-driving the tube valves on their amplifiers through increasing the input signal gain.

It wasn't until 1958 that the first stand-alone unit became popular, the Watkins Copicat, a relatively portable tape echo effect. In 1964 it was documented the first known attempt to achieve distortion through slitting the paper cones of the amplifier with a razor blade, and in 1966 it was manufactured the first amplifier specifically designed to achieve distortion.

In parallel, the development of the electronic transistor allowed manufacturing actually portable effect units. In 1962, the Maestro Fuzz Tone was released as the first transistor-based effect unit to hit the market. In 1967, the first wah-wah pedal, the Clyde McCoy, was released, along with the first octave effect, which Jimi Hendrix dubbed "Octavio". It was in 1968 where the first big hitter of the effects market, the Uni-Vibe, was released, becoming a Leslie rotating speaker emulating unit that is still highly sought-after today.

By midway 1970, a wide variety of solid-state based effects was available, including new pedals such as flangers, ring modulators, phase shifters and choruses.

In the 1980s, digital effects started replacing solid-state based effects. It was not until 1991 when the publication of Nirvana's "Nevermind" album relaunched interest in analog effects, a trend that lasts until today. Nonetheless, as of 2010s, studio-quality digital-based stompboxes are regaining lost ground to the simpler transistor-based effects due to the efforts of high-quality music effects manufacturers.

1.2.2 Modulation effects explained

In this section, the most relevant modulation and time-based effects related to this thesis are explained. Please refer to [2] for further information about music effects.

- Chorus: The main character of this thesis. A chorus effect is achieved whenever one or multiple delayed copies of the input signal are mixed back to the dry signal, often scaled by a certain value, with the delay time being modulated by an independent LFO.
- Delay: also known as echo, this effect is achieved whenever one or multiple copies of the input signal are added to the dry signal, often scaled by a certain value, with fixed delay values. This can be, in practical terms, a chorus without modulating the delay.
- Flanger: a flanging effect is achieved when a single delayed copy of the input signal is mixed back to the dry signal without being scaled, with the delay time being modulated by an independent LFO, and the delay time is small enough that the effect is not perceived as a chorus but as a single signal being comb-filtered. Usually some output signal is fed back to the input to further enhance the effect. This is, in practical terms, a chorus effect with smaller delay time and output feedback. How smaller? [2] recommends less than 15ms for flanger and 15ms to 25ms for chorus.
- Vibrato: a vibrato effect is achieved whenever a single delayed copy of the input signal is output, with the delay time being modulated by an independent LFO. This is, in practical terms, a chorus effect without the dry signal.
- Tremolo: a tremolo effect is achieved through modulating the audio signal with an independent LFO.

1.3 State of the art

Although judging from the number of different chorus units in the market one may be inclined to think that a lot has been done in chorus units implementations, actually not too much has been written, or nothing is specially original. [1] keeps being the seminal reference for implementation of modulation effects, being the *de facto* industry standard when it comes to implementing chorus effects in particular. In this thesis, the basic structure and recommendations from [1] have been followed as much as possible.

More generic references such as [2] simply reference [1] when it comes to chorus design. Further improvement attempts in [1] such as [3] focus on interpolation algorithms for the fractional delay line concept, which is not implemented in this design. [4] delves a bit into the idea of using random data to decrease predictability of the modulating effect and using multiple voices to increase the chorus sensation. They opted for a multiband filtering and band-pass chorusing approach with the aim to decrease noise, which is a very interesting approach, although I opted out of this implementation for CPU bandwidth constraints.

1.4 Objectives

The main objectives to be achieved during the execution of this project are:

- Analyze the perceptual effect of increased delay size in chorus effects.
- Analyze the adequacy of certain basic waveforms (sine, square, pulse, sawtooth, triangle) and their linear combinations as modulating signals for a chorus effect.
- Analyze the musical effect of superposition of multiple delayed copies of the same signal while modulating the delay time, possibly with different delay times and different modulating signals for each copy.
- Analyze the musical effect of modulating parameters of the modulating signal with parameters of the input signal.
- Determine an adequate system architecture for implementing the required features such that the system is modular, scalable, and hardware dependencies can be bounded to HAL modules.
- Determine an adequate system architecture for implementing the required features such that the system is able to implement, with few modifications, other music

effects besides from chorus that require the same building blocks (delay, vibrato, tremolo...).

- Analyze how much of the signal quality degradation is attributable to using an audio path not adequate for Hi-Fi, and how much is attributable to inadequate handling of the digital signal, and minimize this last component.

Chapter 2

Technical description

The purpose of this chapter is to describe the system to be implemented from the digital signal processing point of view. Therefore, it is entirely focused on what, and not how, is to be implemented: refer to Chapter 3 for implementation details.

2.1 System design

Starting from the basics, the most elemental chorus effect possible needs to:

- Store N samples from the input signal $x[n]$ in a delay line.
- Independently generate an oscillating signal $o[n]$.
- Modulate some base delay k with the oscillating signal, $k_o = k \cdot o[n]$.
- Add to $x[n]$ a scaled and delayed version of itself, $y[n] = x[n] + g_k \cdot x[n - k_o]$.

In its simplicity, it already requires a few digital building blocks:

- A delay line of up to N samples (z^{-N}).
- An oscillator.
- A modulator ($\times$).
- An amplifier.
- A mixer ($+$).

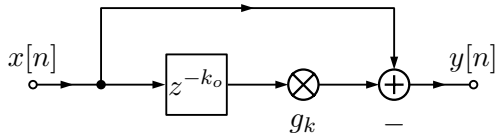

FIGURE 2.1: Basic chorus architecture

Taking as starting point the basic chorus architecture depicted in Figure 2.1, it is legitimate to ask how much this architecture can be reasonably expanded without having to design new digital modules [4], and without changing the core architecture to the point that is not a chorus any more. For instance:

- The range of k can be increased beyond what is usually considered for a chorus effect [2]. This requires a larger delay line.
- With a sufficiently large delay, if no modulation is applied to k, the system behaves exactly like a delay unit [5]. For further delay-like sound, a portion of the output signal can be fed back to the input [6]. Adding feedback requires a mixer.
- If the feed-forward path of $x[n]$ is suppressed and k is small enough, a vibrato effect is achieved [7].
- If the delay path is suppressed and the amplitude of $x[n]$ is modulated, a tremolo effect is achieved [8].
- Multiple delayed copies of the input signal can be aggregated, instead of adding a single delayed copy to the original signal. For this to be of any perceptible effect, k_o values would need to be different for each delayed copy. This means that either k or $o[n]$ need to be different for each delayed copy. This requires additional processing time, but no additional memory, since the same delay line for $x[n]$ is used for all the delayed copies.
- If different $o[n]$ values are to be fed to each delayed copy, this requires the capability of generating additional, different oscillator samples. This can be achieved in several different manners:
  - Using delayed and/or scaled copies of the same oscillator signal $o[n]$. Using delayed copies requires a delay line for $o[n]$, while using scaled copies does not require additional infrastructure except for a gain unit. If certain properties are required for the delays (for instance, with equidistant phases) additional processing capabilities may be required.
  - Increasing the number of oscillators. Those new oscillators can have different parameters (frequency/period, duty) or generate a different oscillating function (sinusoidal, square, triangular, sawtooth...). Duplicating an existing

oscillator increases memory usage and requires additional bandwidth for the signal generation. Implementing a new oscillator with a different waveform requires, in addition, the implementation of another building block.

  - Using delayed and/or scaled copies of multiple oscillators. This combines both approaches previously mentioned.

- $o[n]$ parameters (frequency/period, duty) can be modulated. For instance, they can be dependent on:

  - An independent oscillator, $u_o[n]$. This oscillator can have any waveform and frequency.
  - A random signal, $u_r[n]$.
  - Input signal amplitude.
  - Input signal frequency.
  - Input signal frequency change.

  Moreover, each parameter can be modulated independently from an combination of those parameters. This may require the implementation of additional oscillators, and for sure needs that a random number generator, an envelope follower and/or a module capable of characterizing the frequency content of the input signal are implemented.

As it can be seen, there are a few enhancements that can be applied to the basic structure of a chorus that, while keeping being essentially a chorus, they allow it to perform a wealth of additional functions. Most of the already mentioned have actually been implemented in our system: see section 2.1.1 for additional details.

2.1.1 Features

The system implements the following features:

- Basic chorus capabilities, as depicted in Figure 2.1.
- Basic delay capabilities, up to 1024 samples, resulting in 23ms of maximum delay time. This is barely enough to be perceived as a delay [2], although the effect is perceived as such when high delay and no modulation is applied.
- An oscillator bank consisting of sinusoidal, square, pulse, sawtooth and triangle oscillators. There are three independent sinusoidal oscillators. One of them is able to generate up to 8 sinusoidal signals, with their relative phases equally distributed in the unit circle (see section 3.3.4).

- Delay lines of up to 1024 samples for 6 of the 7 oscillators. The sinusoidal oscillator used as LFO for the other oscillators parameter modulation can not be latched.
- Basic vibrato capabilities.
- Basic tremolo capabilities.
- Up to 8 delayed copies of the input signal can be mixed into the output, plus the original, dry signal.
- Basic random number generation capabilities.
- Possibility to use envelope of the input signal for modulating oscillator parameters.
- Possibility to use input signal frequency content variation for modulating oscillator parameters.

Some features that have been discussed but have not been implemented are:

- Possibility to return a portion of the output signal to the input, creating a feedback loop.
- Possibility to use input signal frequency content for modulating oscillator parameters.

Since the system can be reconfigured to achieve such different functionalities, some convenient mechanism for storing parameters and system configuration needs to be provided. Hence, some memory space has been reserved for storing these configurations, or programs.

System parameters in Table 2.1 are fixed and cannot be modified by programs. On the other hand, parameters in Table 2.2 are reconfigurable on-the-fly, allowing for the flexibility to change the system functionality from one program to another.

Parameter	Notation	Default value	Units
Delay line size	N	1024	-
Maximum number of voices	V	8	-
Oscillator update parameters filter order	N_u	16	-
Pre-modulation high-pass filter order	N_h	2	-
Phase accumulator sin oscillator gain	g^i_{s0}	1.0	-
LFO frequency	f_{LFO}	1	Hz

TABLE 2.1: Fixed system parameters

Parameter	Notation	Default value	Units
Default oscillator frequency	f_{oD}	20	Hz
Default oscillator period	T_{oD}	50	ms
Default oscillator duty	τ_{oD}	10	ms
Frequency sensitivity to LFO	S_{fo}	16	-
Frequency sensitivity to RNG	S_{fr}	16	-
Frequency sensitivity to input amplitude	S_{fe}	16	-
Frequency sensitivity to input pitch change	S_{fp}	16	-
Period sensitivity to LFO	S_{To}	16	-
Period sensitivity to RNG	S_{Tr}	16	-
Period sensitivity to input amplitude	S_{Te}	16	-
Period sensitivity to input pitch change	S_{Tp}	16	-
Duty sensitivity to LFO	$S_{\tau o}$	16	-
Duty sensitivity to RNG	$S_{\tau r}$	16	-
Duty sensitivity to input amplitude	$S_{\tau e}$	16	-
Duty sensitivity to input pitch change	$S_{\tau p}$	16	-
Voice base delay	k_i	0	# samples
Voice gain	g_i	1.0	-
Voice sin oscillator gain	g^i_{s1}	0.0	-
Voice square oscillator gain	g^i_{sq}	0.0	-
Voice pulse oscillator gain	g^i_{p}	0.0	-
Voice sawtooth oscillator gain	g^i_{sw}	0.0	-
Voice triangle oscillator gain	g^i_{tr}	0.0	-

TABLE 2.2: Reconfigurable parameters and their default values

Option	Accepted values	Default value
`bModulateDelay`	`0`: Modulate delay (chorus mode) `1`: Modulate signal (tremolo mode)	`0`
`bFilterOscParams`	`0`: Not filter oscillator parameters `1`: Low-pass filter oscillator parameters	`1`
`bOutputOsc`	`0`: Output $y[n]$ `1`: Output $o[n]$	`0`
`bCoupledSinOsc`	`0`: Oscillator is a combination of basic waveforms `1`: Oscillator is equidistant phase sinusoidal	`0`
`bAddDry`	`0`: Do not add $x[n]$ to $y[n]$ `1`: Add $x[n]$ to $y[n]$	`0`

TABLE 2.3: Configurable options

2.2 System description

The implemented design is a non-linear, time-variant system that performs complex operations to an input signal $x[n]$ with the aim to provide certain desired musical properties at the output.

To better understand what the system is doing, it may be helpful to describe operations performed during processing of a single sample:

1. Sample $x[n]$ is received, and is stored in the input data buffer $\underline{\mathbf{x}}$. $\underline{\mathbf{x}}$ is a circular buffer of length N, where $N = 2^{10} = 1024$ samples:

$$\underline{\mathbf{x}} = \begin{bmatrix} x[0] \\ x[1] \\ \vdots \\ x[n] \\ \vdots \\ x[N] \end{bmatrix} \tag{2.1}$$

2. Sample $x[n]$ is fed to the input of an envelope follower $h_e[n]$, to a pitch change detector $h_p[n]$, and to a RNG. Those modules update their internal states, and provide their output samples $u_e[n]$, $u_p[n]$ and $u_r[n]$, respectively. See section 2.3 for details on those modules.

3. An independent sinusoidal LFO updates its internal state and provides its output sample, $u_o[n]$. Along with the previous outputs, it can be put together in vector form as:

$$\underline{\mathbf{u}} = \begin{bmatrix} u_o[n] \\ u_r[n] \\ u_e[n] \\ u_p[n] \end{bmatrix} \tag{2.2}$$

4. Oscillator parameters frequency, period and duty (f_o, T_o and τ_o, respectively) are updated such as:

$$\begin{bmatrix} f_o \\ T_o \\ \tau_o \end{bmatrix} = \begin{bmatrix} f_{oD} \\ T_{oD} \\ \tau_{oD} \end{bmatrix} + \begin{bmatrix} f_{oD} \cdot 2^{-S_{fo}} & 2^{-S_{fr}} & 2^{-S_{fe}} & 2^{-S_{fp}} \\ T_{oD} \cdot 2^{-S_{To}} & 2^{-S_{Tr}} & 2^{-S_{Te}} & 2^{-S_{Tp}} \\ \tau_{oD} \cdot 2^{-S_{\tau o}} & 2^{-S_{\tau r}} & 2^{-S_{\tau e}} & 2^{-S_{\tau p}} \end{bmatrix} \cdot \begin{bmatrix} u_o[n] \\ u_r[n] \\ u_e[n] \\ u_p[n] \end{bmatrix} \tag{2.3}$$

   where f_{oD}, T_{oD} and τ_{oD} are the default oscillator frequency, period and duty, respectively, and S-factors are the sensitivities of f_o, T_o and τ_o to LFO, RNG, envelope follower and pitch change detector coefficients, $\underline{\mathbf{u}}$.

5. f_o, T_o and τ_o are, optionally, low-pass filtered, using one $N_u = 16$ low-pass filter as described in section 2.3.1.

6. f_o, T_o and τ_o are fed to the oscillator bank, consistent of two sinusoidal oscillators (o_{s0} and o_{s1}), one square oscillator (o_{sq}), one pulse oscillator (o_p), one sawtooth

oscillator (o_{st}) and one triangle oscillator (o_{tr}). Those oscillators update their internal parameters, and their output samples are generated and stored in the oscillator bank circular buffer, which can be expressed in matrix form as:

$$\underline{\mathbf{O}} = \begin{bmatrix} o_{s0}[0] & o_{s1}[0] & o_{sq}[0] & o_p[0] & o_{st}[0] & o_{tr}[0] \\ o_{s0}[1] & o_{s1}[1] & o_{sq}[1] & o_p[1] & o_{st}[1] & o_{tr}[1] \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ o_{s0}[n] & o_{s1}[n] & o_{sq}[n] & o_p[n] & o_{st}[n] & o_{tr}[n] \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ o_{s0}[N] & o_{s1}[N] & o_{sq}[N] & o_p[N] & o_{st}[N] & o_{tr}[N] \end{bmatrix} \tag{2.4}$$

7. $\underline{\mathbf{x}}$ and $\underline{\mathbf{O}}$ are input to the voice processing subsystem. For each voice v_i:

   (a) If the oscillator signal is set to modulate the delay value k_i (chorus mode), no signal is still picked as input signal, and k_i is considered to be 0 for oscillator calculations. Instead, if the oscillator signal is set to modulate the actual input signal $x[n]$ (tremolo mode), then $x[n-k_i]$ is picked as input signal, and k_i is used as delay for oscillator calculations.

   (b) If the oscillation mode is different to coupled sinusoidal oscillators, the compounded oscillator signal for v_i is:

$$o_i[n] = \begin{bmatrix} o_{s1}[n-k_i] & o_{sq}[n-k_i] & o_p[n-k_i] & o_{st}[n-k_i] & o_{tr}[n-k_i] \end{bmatrix} \cdot \begin{bmatrix} g^i_{s1} \\ g^i_{sq} \\ g^i_p \\ g^i_{st} \\ g^i_{tr} \end{bmatrix} \tag{2.5}$$

   with k_i being either k_i or 0 depending on the modulation mode, as aforementioned.

   Otherwise, if the oscillation mode is coupled sinusoidal oscillators, then $o_i[n] = cos(o_{s0}[n-k_i] + \frac{2 \cdot \pi \cdot i}{V})$, where V is the total number of voices, and $i \in [0, V)$.

   (c) If the oscillator signal is set to modulate the delay value k (chorus mode), then $x[n - \lfloor k \cdot o_i[n] \rfloor]$ is picked as the input signal - no interpolation is implemented, contrary to what is suggested in [1]. If the oscillator signal is set to modulate the actual input signal $x[n]$ (tremolo mode), then $x[n-k_i]$ is kept as the input signal choice.

   The recommendation in [1] to interpolate samples when the modulated delay is not exactly an integer value is not followed due to implementation constraints. See section 3.3.4 for further details.

(d) $x[n - k_i]$ can be optionally high-pass filtered to obtain $\hat{v}_i[n]$. The filter is the high-pass filter described in section 2.3.1, with $N_h = 2$.

(e) $v_i[n]$ is equal to:

$$v_i[n] = B_{[0,1]} \cdot \hat{v}_i[n] + \overline{B_{[0,1]}} \cdot o_i[n] \tag{2.6}$$

where $B_{[0,1]}$ can be either 0 or 1, and $\overline{B_{[0,1]}}$ is exactly the value that $B_{[0,1]}$ is not. The option to output the oscillator signal directly is also available for its applicability to use the system as input of another synthesizer.

8. The output of all voices is aggregated and scaled accordingly:

$$y[n] = B_{[0,1]} \cdot x[n] + \sum_{i=0}^{V-1} g_i \cdot v_i[n] \tag{2.7}$$

where $B_{[0,1]}$ can be either 0 or 1. As it can be observed, the original sample $x[n]$, without any gain or delay, can also be added to the output signal. This is specially useful in coupled oscillator mode, to boost thickness of the original signal.

Notice how the noise floor is increased if the total sum of gains, including the original gain of 1.0 for the dry signal, exceeds by too much the 1.0 mark. A different approach based in multi-band filtering and processing is tested in [4].

The following sub-modules are described in more detail in section 2.3:

- Filters
- RNG
- Envelope follower
- Pitch change detector
- Oscillators

2.3 Modules

This section delves into detail of each of the aforementioned modules.

2.3.1 Filters

Filters of different functions and orders are required to shape the frequency content of signals, and are one of the seminal building blocks for digital systems. In this product, three different filter topologies have been used:

- First-order, low-pass IIR filters, in a leaky-integrator topology (see [9]). Those filters emulate the behaviour of moving average FIR filters (see [10]), but instead of latching N samples, the approximation $x[N-1] \approx y[n-1]$ is used, being able to reduce the number of latched samples from N to 1. The difference equation that describes leaky integrators is:

$$y[n] = \frac{(N-1) \cdot y[n-1] + x[n]}{N} \tag{2.8}$$

  which is implemented by the system depicted in Figure 2.2.

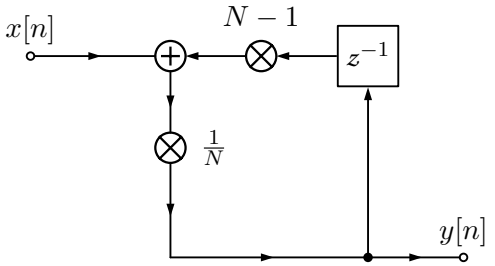

FIGURE 2.2: Leaky integrator

  Its transfer function is:

$$H(z) = \frac{1}{N - (N-1) \cdot z^{-1}} \tag{2.9}$$

  This transfer function has a pole at $z = \frac{N-1}{N}$, and a zero at ∞. Its frequency response, for different N parameters, is depicted in Figure 2.3.

  At the cost of a memory position and a sum for each sample processing, the zero at ∞ can be placed at $z = -1$, further blocking high frequencies, although this has not been implemented. The reason why these filter families have been chosen is for their high computational efficiency when $N = 2^m$ (see section 3.3.3), and therefore is not desired to compromise its performance due to increasing its computation time for a marginal gain in stopband attenuation, which is already quite satisfactory.

- First-order, high-pass IIR filters, in a topology complementary to the leaky integrator. The difference equation describing this system is:

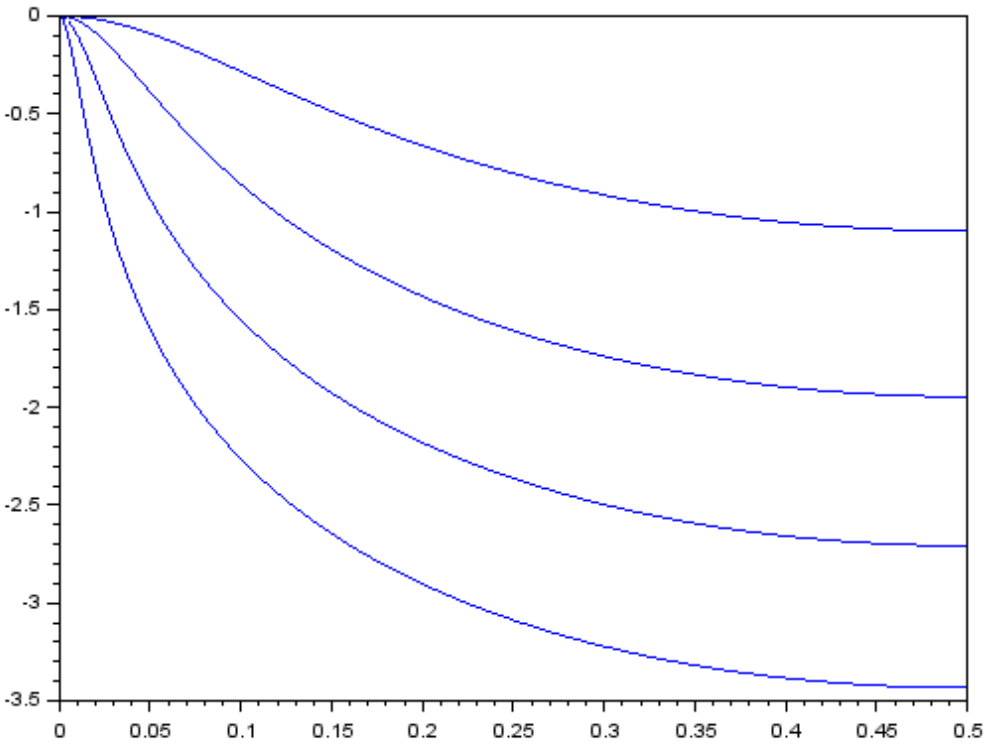

FIGURE 2.3: Low-pass filter frequency response for $N = 2$, 4, 8 and 16 (dB)

$$y[n] = \frac{x[n] - (N-1) \cdot y[n-1]}{N} \tag{2.10}$$

which is implemented by the system depicted in Figure 2.4.

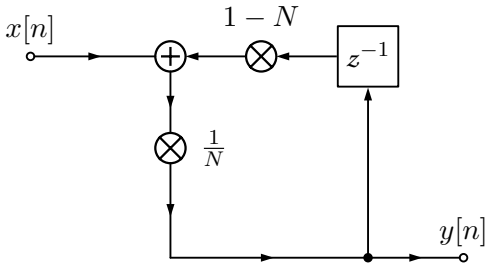

FIGURE 2.4: High-pass filter

Its transfer function is:

$$H(z) = \frac{1}{N + (N-1) \cdot z^{-1}} \tag{2.11}$$

This transfer function has a pole at $z = \frac{1-N}{N}$, and a zero at ∞. Its frequency response, for different N parameters, is depicted in Figure 2.5.

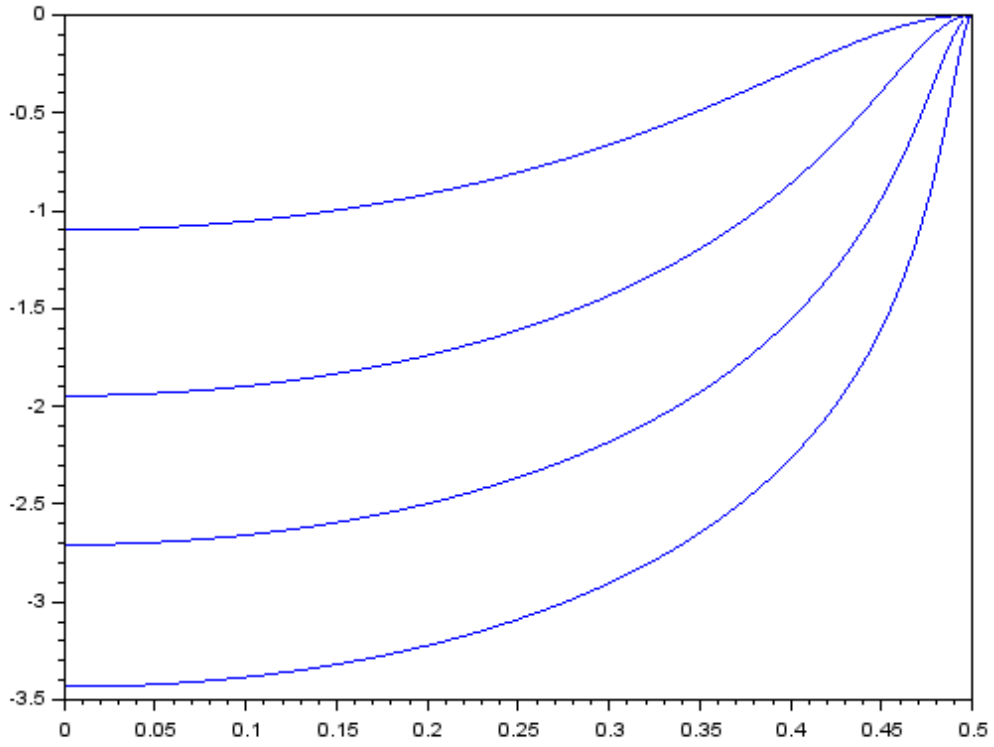

FIGURE 2.5: High-pass filter frequency response for $N = 2$, 4, 8 and 16 (dB)

Notice how the corner frequency barely changes when increasing the order N, instead increasing stopband attenuation. This is why in the system only high-pass filters of first order have been used.

Also at the cost of a memory position and a sum, the zero can be placed at $z = 1$, further blocking low frequencies, although this has not been implemented for the same reasons that it has not been implemented in leaky integrators: the increase in stopband attenuation is not worth the extra operations.

- Second-order, band-pass IIR filters implementing a second-order resonator. These filters are designed to place conjugated pairs of poles and zeros over the frequency of interest, with the zero magnitude being lower than the pole magnitude. These filters are used solely for the filter bank used in pitch change detection.

  The difference equation that describes the resonator is:

$$y[n] = 2 \cdot k_p \cdot k_f \cdot y[n-1] - {k_p}^2 \cdot y[n-2] - 2 \cdot k_z \cdot k_f \cdot x[n-1] + {k_z}^2 \cdot x[n-2] + x[n] \quad (2.12)$$

where k_p and k_z are the magnitudes of the pole and the zero, respectively, and k_f is a factor related to the resonant frequency of the filter:

$$k_f = cos(\frac{2 \cdot \pi \cdot f_c}{f_s}) \quad (2.13)$$

f_c is the central frequency of the bandpass, and f_s is the sampling frequency. It is interesting to notice that, for implementation reasons (see section 3.3.3), these filters operate at a lower rate than f_s. Nevertheless, the decimation rate D_R affects both f and f_s and therefore is negligible for the quotient.

The system that implements this equation is depicted in Figure 2.6.

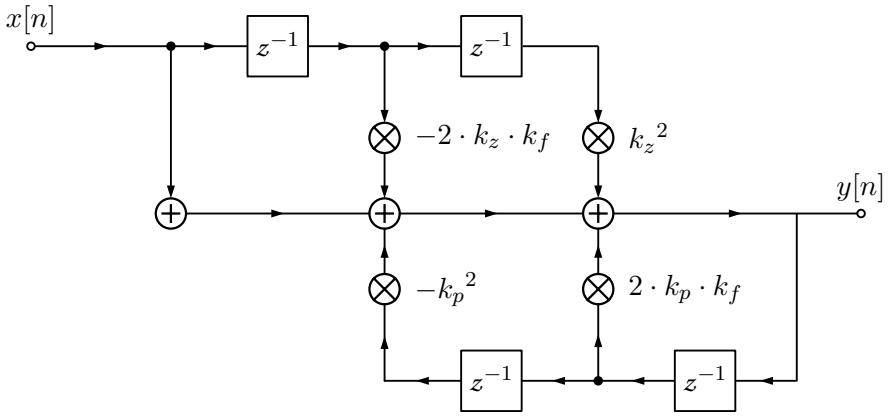

FIGURE 2.6: Band-pass resonator

Its transfer function is:

$$H(z) = \frac{1 - 2 \cdot k_z \cdot k_f \cdot z^{-1} + {k_z}^2 \cdot z^{-2}}{1 - 2 \cdot k_p \cdot k_f \cdot z^{-1} + {k_p}^2 \cdot z^{-2}} \quad (2.14)$$

This transfer function has a pair of conjugate zeros at $[k_z \cdot e^{-\frac{j \cdot 2 \cdot \pi f_c}{f_s}}, k_z \cdot e^{+\frac{j \cdot 2 \cdot \pi f_c}{f_s}}]$, and a pair of conjugate poles at $[k_p \cdot e^{-\frac{j \cdot 2 \cdot \pi f_c}{f_s}}, k_p \cdot e^{+\frac{j \cdot 2 \cdot \pi f_c}{f_s}}]$. And its frequency response, for $k_z = 0.45$ and $k_p = 0.995$ (which are the values used in the application) and different values of k_f, is depicted in Figure 2.7.

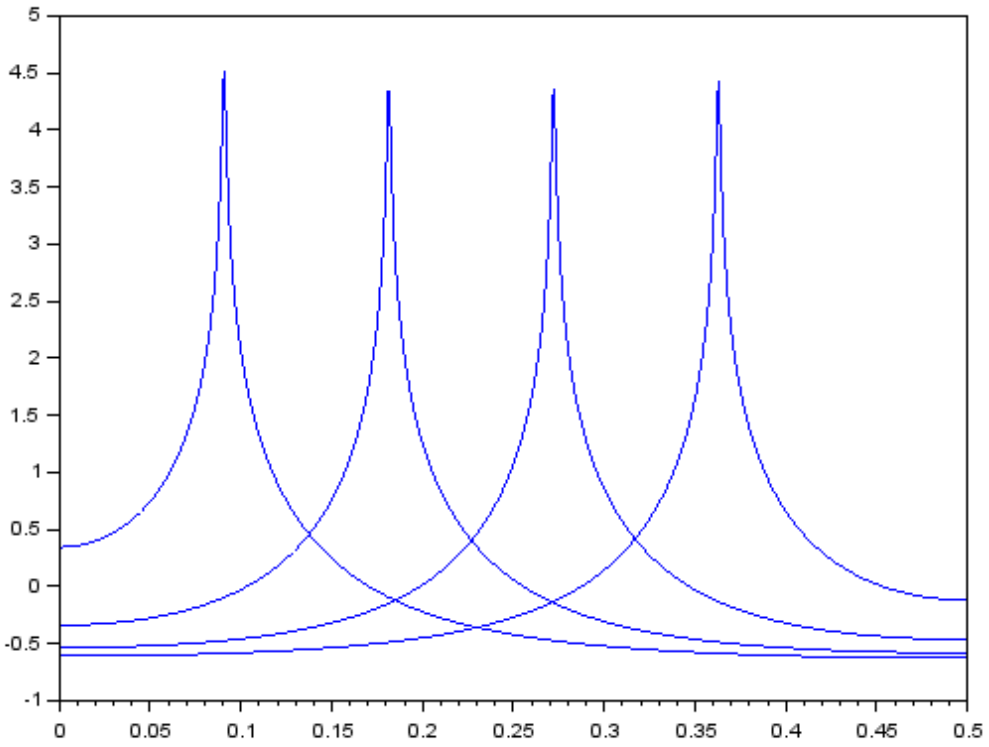

FIGURE 2.7: Band-pass resonator frequency response for $f_c = 4KHz$, $8KHz$, $12KHz$ and $16KHz$ (dB)

2.3.2 RNG

The component of $x[n]$ due to thermal noise and other noise factors is segregated from the part that contains information about the audio signal with the aim to fill a random number pool.

Random numbers from this pool are drawn when a random sample is required. Random samples are normalized to the interval $[0, 16)$.

See further details in section 3.3.5.

2.3.3 Envelope follower

A simple envelope follower is required to determine the amplitude of the input signal. To this end, a digital version of the ever-popular full wave rectifier is proposed, maintaining the classic scheme of using a rectification bridge followed by a low-pass filter, as depicted in Figure 2.8.

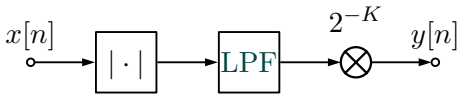

FIGURE 2.8: Envelope follower

The low-pass filter is a leaky integrator, as described in section 2.3.1, with $N = 16$. $|\cdot|$ simply indicates the absolute value of $x[n]$.

The purpose of this system is not to obtain a precise representation of the amplitude of the input signal, but to obtain a rough metric of the level of the input signal. To accommodate the whole dynamic range of the input signal, the output of the envelope follower module is scaled down by 2^{-K}, where K is the scaling factor, obtaining a logarithmic response which is more adequate to the nature of the sound.

2.3.4 Pitch change detector

The pitch change detector is a module aimed at obtaining an approximate metric of pitch changes in the input signal. To this end, a bank of IIR narrow band-pass filters centered at the frequencies f_m of all musical notes on two consecutive octaves when tuning in A=440Hz is implemented, as seen in Figure 2.9.

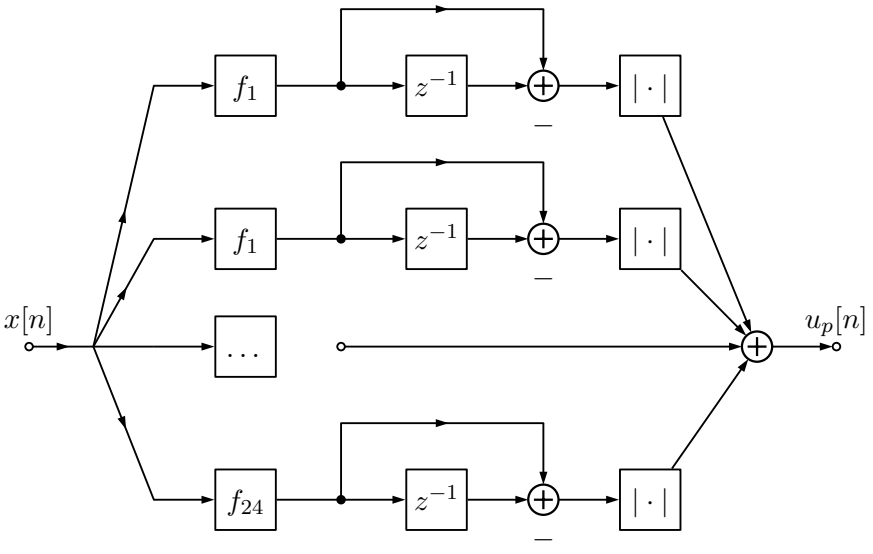

FIGURE 2.9: Pitch change detector

Central frequencies f_m for each band-pass filters are depicted in Table 2.4. Since there are 12 musical notes, and two octaves are covered, the total count of filters adds up to 24.

For bandwidth reasons, these filters are processed at a slower rate than the original signal (see section 3.3.7 for details). For each input sample $x[n]$, only the output of one

Musical note	Frequency f_m (Hz)	Musical note	Frequency f_m (Hz)
E_6	1318.51	E_7	2637.02
F_6	1396.91	F_7	2793.83
Gb_6	1479.98	Gb_7	2959.96
G_6	1567.98	G_7	3135.96
Ab_6	1661.22	Ab_7	3322.44
A_6	1760.00	A_7	3520.00
Bb_6	1864.66	Bb_7	3729.31
B_6	1975.53	B_7	3951.07
C_7	2093.00	C_8	4186.01
Db_7	2217.46	Db_8	4434.92
D_7	2349.32	D_8	4698.63
Eb_7	2489.02	Eb_8	4978.03

TABLE 2.4: Band-pass filter central frequencies $f_m(Hz)$

filter is processed. For all filters to operate on the same signal, $x[n]$ is decimated at a decimation rate D_R, and the input to the band-pass filters is $x[n \cdot D_R]$.

The output of each filter $x_{fm}[n \cdot D_R]$ filters the frequency content of $x[n \cdot D_R]$ around f_m. To obtain a metric of change of this frequency content for f_m, which we may call u^p_{fm}, the previous output of the filter $x_{fm}[(n-1) \cdot D_R]$ is subtracted from the current output of the filter $x_{fm}[n \cdot D_R]$, and the absolute value of the result is taken. Doing so for all f_m and summing the u^p_{fm} coefficients gives an indication of how much the frequency content of $x[n \cdot D_R]$ has changed between $x_{fm}[(n-1) \cdot D_R]$ and $x_{fm}[n \cdot D_R]$ around the whole spectrum, and is used to calculate the u_p coefficient:

$$u_p = \sum_{fm=1}^{24} |x_{fm}[n \cdot D_R] - x_{fm}[(n-1) \cdot D_R]| \tag{2.15}$$

2.3.5 Oscillators

The architecture of digital-like oscillators -square, pulse, sawtooth, triangle- does not really involve any math, being more of a microcontroller programming issue, and therefore being explained in section 3.3.4. Nevertheless, the implementation of the sinusoidal oscillator o_{s1} is worth a mention in this section, for the digital signal processing involved.

o_{s1} is different than o_{s0} in the fact that, while o_{s0} relies on a simple mechanism such as normalized phase accumulation -the phase of the oscillator is increased by $f_o \cdot T_s$ and normalized to $[0, 2\pi)$ interval, and then the cosine is calculated-, o_{s1} implements the second-order digital waveguide sinusoidal oscillator described in [11][12] and explained in [13][14].

The difference equation that rules the behaviour of the second order digital waveguide sinusoidal oscillator is:

$$\begin{bmatrix} y[n] \\ \hat{y}[n] \end{bmatrix} = \begin{bmatrix} cos(\theta) & cos(\theta) - 1 \\ cos(\theta) + 1 & cos(\theta) \end{bmatrix} \cdot \begin{bmatrix} y[n-1] \\ \hat{y}[n-1] \end{bmatrix} \tag{2.16}$$

where $\theta = \frac{2 \cdot \pi \cdot f_o}{f_s}$.

It is proven in [11] that this equation leads to oscillating outputs in both $y[n]$ and $\hat{y}[n]$. In addition, it is also shown that this oscillator fulfills some desirable properties, such as the fact that $y[n]$ is bounded between 0 and 1 (this is not the case for $\hat{y}[n]$, hence discarding $\hat{y}[n]$ for use in the system), and the fact that $y[n]$ and $\hat{y}[n]$ are exactly $\frac{\pi}{4}$ out of phase.

This second property leads to efficient simplifications in the initial vector, which is reduced to:

$$y_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{2.17}$$

see [11], and in the output power calculation, useful for AGC, which is reduced to:

$$P = y^2[n] - \frac{cos(\theta) - 1}{cos(\theta) + 1} \cdot \hat{y}^2[n] \tag{2.18}$$

see [13].

As mentioned, AGC is implemented to ensure that the signal $y[n]$ does not degenerate after a long time in continuous operation. After some simplification [13], the AGC coefficient G is reduced to:

$$G \approx \frac{3}{2} - P \tag{2.19}$$

where the first-order Taylor expansion for the expression of G has been used [13], and P is the output power as calculated above.

Chapter 3

Implementation

The purpose of this chapter is to discuss the details relative to the implementation of the system described in the previous sections in an embedded system.

First of all is tackled the choice of the embedded system itself. Different candidate MCUs are analyzed, and their strengths and weaknesses relative to the project requirements are outlined.

Afterwards, technical details about the implementation in the chosen MCU of the subsystems detailed in the previous chapter are highlighted. Emphasis is placed on the technical challenges faced and the trade-offs between performance, bandwidth and system scalability.

3.1 MCU choice

One of the most critical choices in the design of an embedded system is the choice of the MCU. It shapes in an unique way the development of the software, and is often the component that is most expensive to replace in terms of engineering costs, therefore being critical to assess adequately the requirements and make the right choice from the beginning.

The ideal MCU for this project would have the following features:

- High core bandwidth: high core throughput is critical for audio applications, since the industry-standard audio sampling frequencies of 44.1KHz and 48KHz are still quite demanding for the capabilities of currently available MCUs. For instance, a MCU running at 48MHz only has available 1000 clock cycles per sample, and a

complex application can escalate to that figures quickly. Therefore, is necessary that the core is able to acommodate the required number of operations per sample with some safety margin.

- DSP capabilities: the availability of HW acceleration for operations such as integer multiplication, integer division, MAC, bitwise shifting and trigonometric operations is pivotal, since those are common operations in audio applications and a lot of the aforementioned core bandwidth is lost whenever this operations need to be performed through software algorithms.

  Other desirable features are registers with saturation instead of overflow for DF1 filter implementation to avoid overflows, circular access buffers for zero-overhead array accessing, and bit-reversed access buffers for FFT calculations.

- FPU availability: whenever there is not an obvious loss of performance from recirculation of round-off or truncation errors (for instance, in a system with no feedback), the presence of a FPU greatly simplifies code development while avoiding any noticeable loss of performance [15][16]. This is specially true in IIR filters, where great dynamic range and less precision is acceptable for the accumulators, while great precision and restricted dynamic range is preferable for the coefficients. The floating-point representation offers this kind of dynamism without the cumbersome programming that integer representation often carries within.

- Entry cost: the cost of tools and materials for developing a project in such a MCU should be within reach of an individual not aiming to make profit out of this development. An ideal candidate would be provided with a free IDE, a free compiler/linker/debugger toolchain, should be available within an inexpensive development board, and should have available a reasonably costed debugging tool.

- Scalability: it is desired that the chosen MCU provides an upgrade path in case that more features are selected for development in future stages of the project.

In addition, other requirements not directly related to the MCU itself may play an important role in the MCU choice. In this case, for instance, it is desired that the development board includes some method for easy assembling of an analog audio path, to avoid complex hardware design and modifications, which are outside the scope of the current project.

Taking a look at the available candidates, one of them stands out immediately. The Microchip dsPIC33FJ256GP506 is available in a development board [17] that conveniently includes 3.5mm TRS connectors for signal input and output, adequate analog signal paths, and a high-quality Wolfson WM8510 audio codec. The IDE is free, and the

toolchain is free too, with the caveat that code cannot be compiled with optimization levels higher than `-O0` with the free version. Nonetheless, a 1-month trial period for the full version is offered, which should be more than enough time for enabling optimizations, debug potential errors and compile the release version.

This MCU has available some DSP features [18] such as hardware acceleration of integer multiplications and divisions, MAC, a barrel shifter, two accumulators with saturation, and circular and bit-reversed buffer access. On the other hand, the bus width is only 16 bit, no FPU is available, and core throughput is fairly low at 40MHz. Nonetheless, upper-grade members of the family can reach up to 70MHz while maintaining code-compatibility, allowing for expansion for potential future upgrades, which would most likely imply a device change since this MCU is not recommended for new designs (a pin and code-compatible upgrade to this device which is in the mass production phase of its life cycle is the dsPIC33FJ256GP506A) [19].

Other MCUs considered have been:

- TI's C2000 [20] is a 32-bit family with FPU and HW acceleration for all common DSP operations that runs in the 100MHz-400MHz frequency range, and it may include up to one slave core and up to one additional CPU, with its respective slave core too. The F28377S MCU [21] is available in an inexpensive development board [22], along with a free IDE and a toolchain with free licenses, although code-size-restricted. Nevertheless, there are not available any audio-related add-ons for those development boards.

- Any MCU based on the Cortex-M4 ARM core [23] implements a 32-bit wide data bus and HW acceleration for DSP operations, including SIMD arithmetic. The Cortex-M4F family members include a FPU too. Some popular MCU families implementing the Cortex-M4 and M4-F cores are Freescale/NXP Kinetis [24], ST STM32 [25] and Infineon XMC4000 [26].

  There is a rich set of tools for the ARM ecosystem and a wealth of development boards with a Cortex-M4 core, but again there are not available audio-related add-ons for those development boards with the required quality for the audio path to be used for this project.

- Analog Devices SHARC [27] and SigmaDSP [28] families are targeted directly to audio processing, and offer useful and innovative features such as 28-bit data buses targeting 24-bit audio samples, single-cycle execution for the entire instruction set, advanced buffer access modes, and SIMD for block-processing data, in addition to other standard features such as FPU and HW-accelerated DSP instructions. Moreover, most of their development boards target specifically audio applications.

Nonetheless, despite being ideal candidates when looking solely at the HW, the tools offered for development are not adequate for the purpose of the project. Development boards are quite expensive for both families. In addition, the price of the license for the SHARC IDE and compiler are prohibitive.

Furthermore, SigmaDSP MCUs do not even have an actual `C` compiler, instead generating assembly code from their graphical building tool directly, defeating the purpose of this project of dealing with the embedded software development part.

- OpenLabs offers an expansion board for Arduino with an integrated Wolfson WM8731 [29]. Nevertheless, compatible Arduino boards fall short in bus width and core bandwidth to be able to execute the required algorithms.
- Similarly, MikroElektronika offers an audio codec with the same Wolfson WM8731 [30]. Is connectable through pin headers to any device with enough pins to support it, although some HW modifications may be required depending on the interconnected device.

In the end, the dsPIC33FJ256GP506 has been chosen as the platform for development of this project, for hitting an acceptable middle-of-the-road trade-off between suitability, performance, ease of development and cost. Although some of the non-selected MCUs were far superior in performance, features and suitability for the current project than the dsPIC33FJ256GP506, the simplicity of avoiding completely HW design or device integration phases prior to the actual digital signal processing stage have tipped the scales in its favor.

3.2 CPU architecture and peripherals

The dsPIC33FJ256GP506 implements a modified Harvard architecture [18], which is suited for DSP applications, allowing to fetch instructions and data in the same clock cycle. In addition, dual data paths, allowing to fetch two memory locations at the same time, are also available, with some restrictions.

The CPU operates on fixed-point data. Although it offers native support to work in `Q1.15` format (see [31]), this feature is actually not usable in the context of our application, since most of the time a slightly larger range of $[-\pi, \pi)$ is required. Therefore, most of the application has been developed using integer data formats in the CPU, interpreted as fixed-point formats by the application.

Although the MCU offers a large array of DSP features implemented, some architectural details actually difficult taking significant advantage of them in the context of our application. For instance, the fact that the data bus is 16-bit wide carries along an avalanche of undesired consequences:

- Operations with 24-bit data (such as our input signal) need to be performed by dedicated SW algorithms, instead of being performed in the number of cycles specified by the machine instruction (1 in the case of multiplication, 18 in the case of integer division, see [32]).
- Since IEEE 754 single-precision floating point numbers are defined to be at least 32-bit wide [33], it is awkward to place a 32-bit FPU over this data bus. Therefore, the dsPIC33FJ256GP506 does not have FPU, relying instead on SW algorithms to perform floating point operations. This, in practical terms, means that floating point numbers can not be used unless they are only used as macros to be optimized at compile-time, used only during run-time initialization, or accessed during the application initialization phase: no floating point operations should be performed while the system is processing audio samples.
- It is not possible to take advantage of the dual fetch data bus because two reads need to be performed to completely fetch input data, turning a 1-cycle MAC in a 2-cycle MAC. Hence, most of the DSP features of this MCU are not exploitable.

In addition, other features can not really shine in this application. For instance, bit-reversed buffers are not required at all. Overhead-free circular buffers would have been really useful, but only a single circular buffer can be used at a time in this architecture, which is awkward in our application requiring 7 circular buffers, 5 of which are accessed at the same time. Therefore, it has been preferred to preserve symmetry between all buffers.

Nonetheless, other features have been absolute all-stars. HW-accelerated division and fast multiplication are ubiquitous in the application, and without them probably some features would had been cut out due to lack of core bandwidth. Since 32-bit multiplication has been avoided as much as possible, the barrel shifter has been very useful whenever multiplicative algorithms have been substituted by shift-and-add based algorithms.

In addition, some peripherals, such as timers, DCI and I2C have been really useful during development. Timers play a crucial role in non-sinusoidal oscillators, allowing to generate such signals with low overhead. The DCI allows the user to communicate with

the *codec* data interface in a seamless manner, as much as the I2C allows the user to communicate with the *codec* configuration interface without complications.

3.3 Module implementation & technical details

This section discusses in-depth technical details of the implementation of each of the system modules.

3.3.1 Audio path

The system input signal is received through a 3.5mm TRS stereo female connector. The stereo signal is converted to mono through physical interconnection, and afterwards is fed to an analog preamplifier, which allows for a selectable 3dB/23dB amplification range [34] through the use of an on-board potentiometer. The amplified signal is delivered to the audio *codec* input.

The *codec* allows for sampling rates of up to 48KHz, but a sampling rate of 44.1KHz has been chosen to have available more clock cycles per sample without degrading the signal. The signal has been quantized to 24 bits, which is the maximum resolution that the *codec* allows for. The quantized signal is transmitted to the MCU through the data converter interface in DSP mode [35], where is processed.

After completion of the required digital signal processing operations, the output data is sent back from the MCU to the *codec* through the DCI. The analog signal is reconstructed there, and afterwards is sent to a headphone amplifier [34][36], which can provide digitally-selectable amplification from +12dB to -33dB. Two output pins of the DSC in two-wire mode are used to control this amplifier, using an ad-hoc protocol [36].

The output of the headphone amplifier is, finally, delivered to the output of the system through a 3.5mm TRS stereo female connector. This output can carry up to 105mW per channel of continuous average power into a 16Ω load [36], which should be more than enough to drive standard headphones or the input of a further subsystem.

3.3.2 Trigonometric functions

HW acceleration for trigonometric functions is not present in the MCU, and both sinusoidal oscillators require the ability to compute cosines on run-time (see section 2.3.5). Therefore, it is required to implement this functionality in SW.

A LUT-based approach has been taken for the cosine calculation. Since no FPU is available and LUTs cannot operate on fractional inputs anyway, fixed-point `Q19.12` is used as the LUT input format. `QX.Y` numbers are fixed-point numbers with `X` integer digits and `Y` fractional digits (see [31] for further information). Therefore, a `Q19.12` can represent numbers in the range of 2^{19}, which is more than enough for a cosine input, which should be in the $[-\pi, \pi)$ range anyway, with precision of 2^{-12}, or 0.00024.

To avoid using floating point numbers without FPU support, LUT outputs are also in fixed-point format, in this case in `Q7.24` outputs. Precision on the cosine outputs is increased to allow for more precise stepping and lower minimum frequency in recursive sinusoidal oscillators, while maintaining performance, since those operations are implemented in 32-bit width data anyway. Using `Q7.24` format, output precision increases to 0.000000059, although input quantization is a larger source of error in the output than output quantization, due to the huge magnitude order difference between both.

To avoid increasing unnecessarily LUT size, only the input range of $[0, \frac{\pi}{4}]$ has been hard-coded. Cosines for the other three quadrants are obtained through arithmetic transforms on the angle input and the cosine output. Similarly, angles out of the $[-\pi, \pi)$ range are reduced to the $[-\pi, \pi)$ range before performing cosine calculation.

3.3.3 Filters

As described in section 3.3.3, three kinds of digital filters are implemented: low-pass, first-order IIR filters, high-pass, first-order IIR filters, and band-pass, second-order IIR filters. Both low-pass and high-pass filters share a common topology, changing only the sign of a coefficient. All of those filters were designed to require the minimum memory and bandwidth from the DSC, even sacrificing performance for a smaller footprint.

IIR filters could have been designed using the Remez exchange algorithm [16] as implemented in popular signal processing software suites such as Matlab [37] or Scilab [38], and implemented using multiplicative algorithms. Nevertheless, those filters require multiplications with carefully-computed coefficients, in which inadequate coefficient quantization can lead to unacceptable errors at the output.

Instead, some lightweight and resilient filters are implemented, which are proven and adequate for usage on a MCU, although unacceptable on a proper DSP.

For low-pass and high-pass filters, the leaky integrator topology depicted in Figure 3.1 is implemented, with the only difference between both being the sign in the $(N - 1)$ factor. As avid embedded systems designers may have observed, if $N = 2^m$, then those filters can be implemented without performing multiplications at all. For each sample:

1. Fetch $y[n-1]$.
2. Shift left m bits $y[n-1]$. Subtract $y[n-1]$ to the result.
3. Change sign, if required.
4. Fetch $x[n]$. Sum to the previous result.
5. Shift right m bits. Latch $y[n]$.

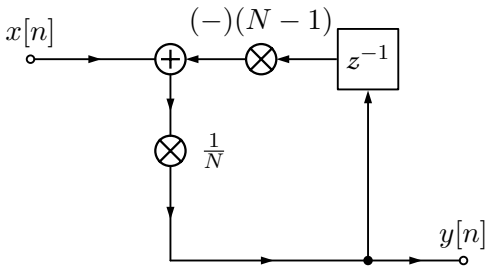

FIGURE 3.1: Low-pass and high-pass filter topology

Thus, complicated multiplications with 24-bit wide data is avoided, instead relying on much simpler operations that do not require complex algorithms such as Karatsuba multiplication [39] to be implemented on longer data widths.

Unfortunately, second-order band-pass IIR filters cannot be optimized that much, and 32-bit wide multiplications need to be used. Nonetheless, some lesser optimizations are still available. For instance, all of the filter coefficients are pre-computed and stored in memory during initialization, to avoid losing bandwith in execution time. With respect to this, the symmetric choice of coefficients has not been casual, allowing for most coefficients to be reused, saving memory. Due to coefficient symmetry, some intermediate results can be reused too. Finally, the feed-forward path only needs to be calculated once for all filters instead of once for each filter, further saving bandwidth.

3.3.4 Oscillators

Six different oscillator topologies have been implemented in this system. In this section, their implementation details are going to be briefly outlined:

- The recursive sinusoidal oscillator $o_{s1}[n]$ is implemented as described in section 2.3.5.
- The sinusoidal oscillator $o_{s0}[n]$ is implemented using a simple mechanism of phase accumulation. As such, the internal state of the oscillator is updated each sample

through adding $\frac{f_o}{f_s}$, this is, the increment in phase that is expected to happen for the oscillator during a sample period T_s, and then reduced to the interval $[-\pi, \pi)$.

Storing the phase accumulated and calculating the cosine of the phase during oscillator read instead of storing the cosine output directly has both advantages and inconveniences. If the same sample is to be accessed more than once, the cosine needs to be recalculated, which is certainly inefficient. On the other hand, storing the accumulated phase allows to generate multiple out-of-phase sinusoidal signals using the same oscillator engine and a single delay line instead of requiring multiple oscillator engines and multiple delay lines, which allows for huge resources saving. Since it has been deemed interesting to be able to have multiple equidistant-phase oscillator outputs (for emulating the popular tri-chorus effect, for instance), this is the mechanism that has actually been implemented.

- Square waves can be generated directly by the HW available in the dsPIC33FJ256GP506 using timers, setting the period to half the square wave period, and changing the output level each time the timer overflows.

- Pulse waves can be generated in a very similar manner to square waves, only that two timers are required: one for the base period, and another for the duty period.

- Sawtooth waves can be easily generated using the same timer mechanisms than for square waves. Nonetheless, instead of just switching the output level between 1 and -1, the sawtooth oscillator requires a division of the timer value by the period value. Although 32-bit by 16-bit division can be accelerated by HW [32] and executed in just 18 clock cycles, performing it for each sample is still demanding. Therefore, the division is not performed whenever a sample is stored, but only whenever a sample is read.

- Triangle waves can be implemented combining the mechanisms used in square wave generation and sawtooth wave generation. Again, the division of the timer by the period is only performed during read operations, and not during write operations.

Careful manipulation needs to be taken whenever updating the period of timer-based oscillators, to avoid changing the period to a value smaller than the current timer. To avoid this, after updating the period, the timer should be checked: if is greater than the period, the output level should be changed manually and the timer should be manually reset, incurring in a small jitter.

In addition, interpolation of fractional delay values as suggested in [1] has not been used. It has been found out that the benefit obtained is not worth the increase in computational costs, due to the fact that other elements of the signal chain such as the

analog audio path actually degrade the signal to the point that such an improvement is not actually audible. Nonetheless, if implemented using a proper DSP with audio-grade components, sample interpolation should be a must.

3.3.5 RNG

The MCU chosen does not include support for random number generation. Since it would be desirable to have available random or pseudo-random numbers for certain algorithms, a RNG needs to be implemented in SW.

Assuming the following model for the received audio signal:

$$x_m[n] = x[n] + w[n] \tag{3.1}$$

where $x[n]$ is the original signal, $x_m[n]$ is the measured signal and $w[n]$ is a generic noise component that results from the aggregation of all the noises present in the signal chain (transduction non-linearities, thermal noise in transmission, sampling and quantization noise...), it is feasible to assume that due to the CLT [40] the $w[n]$ noise component can be approximated as Gaussian noise.

Using this approximation and taking into account that the input signal is quantized to 24 bits, it can be safely assumed that as long as the following equation holds true:

$$|w[n]| > \frac{2^N}{2^{24}} \tag{3.2}$$

where N is the selected number of LSb of the input sample, then variations on those LSb are essentially due to $w[n]$ and not to $x[n]$, and therefore that the following signal:

$$RNG[n] = x[n] \mod 2^N \tag{3.3}$$

which is the result of truncating $x[n]$ to the last N LSb, can also be considered to behave as Gaussian noise, therefore being suitable for random number generation in the interval $[0, 2^{N-1}]$.

In our system, $N = 4$, and $RNG[n]$ is stored in a pool of random numbers, using $RNG[n]$ to process sample $x[n]$. Instead, $RNG[n-k]$ can be used to process sample $x[n]$, where k is a small delay, to further decrease correlation between the input sample and the random sample.

3.3.6 Envelope follower

The envelope follower implementation is straightforward, taking into account the architecture described in section 2.3.3 and the details about filter implementation described in section 3.3.3.

3.3.7 Pitch change detector

The pitch change detector implementation should be straightforward, as the architecture described in section 2.3.4 along with the filter implementation described in section 3.3.3 do not leave too much of a choice. Nevertheless, the reason why the input $x[n]$ is decimated by $D_R = 26$ should be explained, since it caused by implementation constraints.

As it is to be expected, to compute the output of 24 second-order IIR filters in the time allocated for processing a single sample and, in addition, to be able to perform additional operations, it is too much for a 40MHz MCU operating at 44.1KHz data rates, which only has available roughly 900 clock cycles per sample. In addition, IIR coefficients are 32-bit wide to avoid signal degradation, resulting in having to store and even sum 64-bit results. Since 64-bit wide data is not supported natively by neither the data bus nor the CPU, additional bandwidth is required for the SW library to handle this.

The solution encountered for being able to implement the pitch change detector in this MCU is to operate on a decimated signal $x[n \cdot D_R]$. To accomplish this, the following steps are executed:

- When the module is started, the input signal $x[n_0]$ is latched, and the feed-forward path of all IIR filters, which is common to all filters, is prepared. No further operations are performed during this sample.
- On the next sample $x[n_0+1]$, a single filter is processed, for the input sample $x[n_0]$. The feedback path of the current filter is updated, and no further operations are performed during this sample.
- The previous step is repeated until all filters have been processed for input $x[n_0]$.
- When all filters have finished being processed, the previous output of all filters is subtracted from the current output of all filters, and the absolute value of the result accumulated to $u_p[n]$, which is the output of the pitch change detector module.

On the next sample, $x[n_0 + D_R]$ is latched, and the cycle starts again. Since there are 24 filters, one pre-processing step, and one post-processing step, D_R becomes 26.

In addition, for saving additional bandwidth, whenever the pitch change coefficient $u_p[n]$ is not used (this is, when S_{fp}, S_{fT} and $S_{f\tau}$ are 16, and therefore $2^{-Sp} = 0$ in 16-bit data), no operations related to pitch change detection are performed. Operation is resumed again whenever one of those values changes to a value strictly less than 16.

Chapter 4

Results

4.1 Results obtained

This section summarizes the results obtained after testing the unit, both in an engineering test-bench and in a real-world environment with actual musicians.

4.1.1 Objective quality

Testing with a musical demo signal [41], certain coloration can be appreciated. To debug the cause of this unexpected behaviour, the oscillator square wave output $o_{sq}[n]$ has been output and observed with the help of an oscilloscope, providing the results depicted in Figure 4.1.

It can be observed the charge and discharge process of capacitors affecting the signal output. If a pure sinusoidal signal is observed instead, the results in Figure 4.2 are obtained instead.

Since the charge-discharge cycle of the capacitors is more relaxed with the sinusoidal signal input, no deformation is observed. Therefore, it is concluded that capacitors with lower time constant should had been used in the analog audio path instead of the capacitors currently used, and that this capacitor choice by the designers of the embedded development kit was most probably related to cost in scale economies than to actual audio engineering.

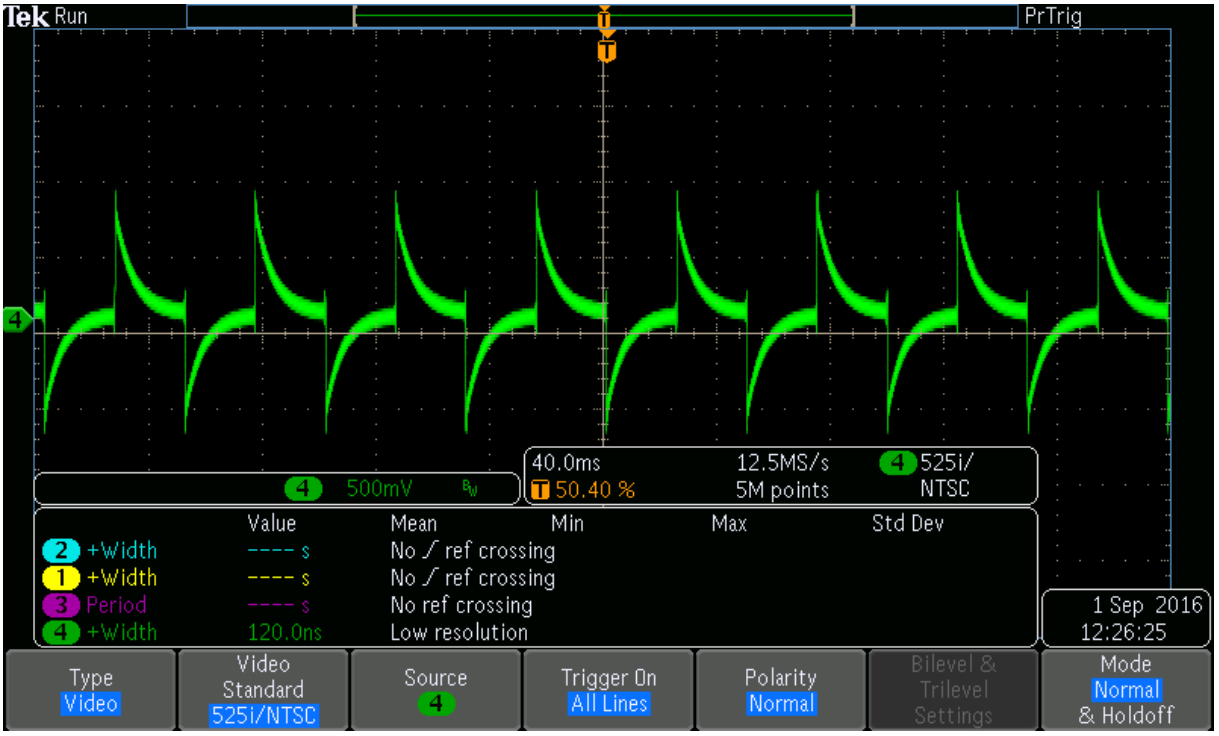

FIGURE 4.1: Square wave output signal as received by the oscilloscope (mV)

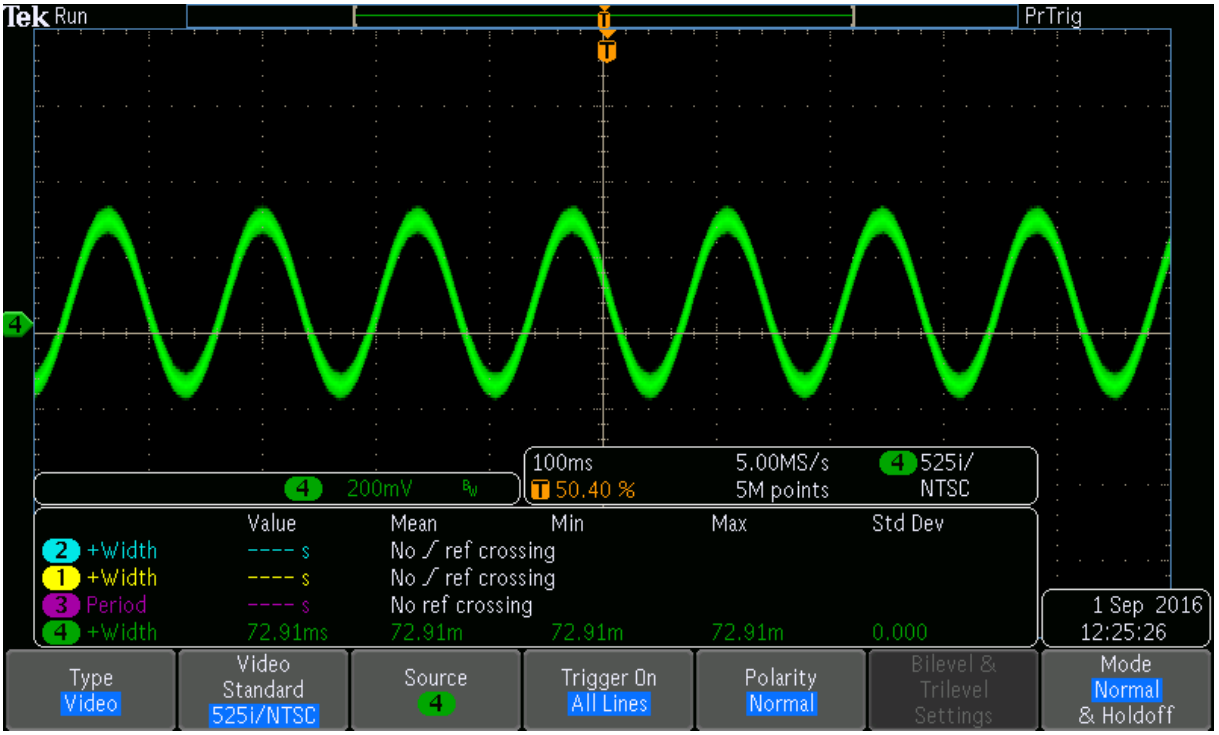

FIGURE 4.2: Sinusoidal wave output signal as received by the oscilloscope (mV)

4.1.2 Subjective quality

The unit has been brought to be tested by actual musicians, songwriters and arrangers of diverse musical backgrounds, diverse musical interests, diverse instruments of choice and varying degrees of skill. Three record producers and one audio engineer have also participated. A total of 21 subjects, excluding myself, have participated. Their demographics are summarized on tables B.1, B.2, B.3 and B.4 on Appendix B.

I have been present during the evaluation of the unit due to the lack of availability of multiple prototypes and the difficulty to send it back and forth to places where subjects could evaluate them, and also to make it easier for the subject to just concentrate on playing their instrument and toying with the unit while I was taking notes on their evaluations. In addition, although the subjects were explicitly instructed to be very honest on their evaluations, most of the subjects that tested the unit were either first or second degree acquaintances of mine, due to the difficulty of finding random test subjects without offering economic compensation. To make matters worse, due to schedule compatibilities, it has been frequent that some subjects evaluated the unit in the presence of another subject, and as a consequence some subjects were present during the evaluation of another subject. Therefore, subjective evaluations may be contaminated in an unknown degree due the aforementioned. No blind tests have been performed.

71% of the subjects consider the unit to provide an interesting effect. In particular, 57% of the subjects express interest in the additional features of the unit, and 52% of the subjects express satisfaction with the unit as it is, while 71% have explicitly expressed that they would prefer it to have improved sound quality. Only 9% of the subjects express complete discomfort with the unit and with what is trying to achieve. One of the discomforted test subjects claimed that is a “horrible effect”, while the other mentioned lack of interest due to lack of applicability in a real context. On the other hand, 28% of the users requested to have more time to experiment in depth with the unit, while 52% of the users expressed interested in being updated about future improvements on this unit, and 28% of the users claimed that they would use the unit in a professional situation right now.

Crossing results with subject demographic data, it can be observed that most of the subjects interested and satisfied with the unit are either guitarists or keyboardists, interested in either rock or jazz. On the other hand, piano and brass players, and people with classical formation, tend to dislike this product the most. People with a professional relationship with music tended to be more interested in possible improvements on this product, while amateurs and hobbyist either liked it or disliked it categorically.

4.1.3 Comparison with similar products

Comparison with similar products can be done with respect to several metrics. The two metrics that have been chosen to evaluate our product have been features available and sound quality. The unit has been compared to the following devices, in increasing degree of overall quality:

- Mooer Ensemble King [42].
- Boss CE-5 [43].
- TC Electronics Corona Chorus [44].
- Strymon Mobius [45].
- Eventide ModFactor [46].

Comparison to rack-mounted units and VSTs has been avoided, for having different purpose and price range, although some embedded units like the Strymon Mobius and the Eventide ModFactor fall also in the highest end of the quality spectre.

With regards to features, our product packs much more features than both the Mooer Ensemble King and the Boss CE-5. Those are analog units dedicated to producing chorus effects only, and as such they only include a minimal set of configurable parameters: level, rate (equivalent to oscillator frequency) and depth (equivalent to maximum delay) for the Mooer, and the same parameters plus additional high and low pass filters before the input for the Boss.

Moving to digital units, the Corona chorus is an affordable chorus unit that can be configured using a desktop application and a PC [47]. Our unit still provides more features, such as additional waveforms, the capability to modulate oscillator parameters using input data metrics, and the possibility to output oscillator data directly. Nevertheless, the Corona chorus offers also a great deal of reconfigurability, being able to work as either a delay, a chorus with up to 3 voices, possibly with equidistant phases, includes pre-modulation filters, and is a very flexible unit overall. The Corona chorus is the kind of affordable unit that our system expected to be able to hold its own with, so it is a good thing that we managed to pack more features in our unit than on it.

Our unit starts falling short in features when compared to high-end units like the Strymon Mobius or the Eventide ModFactor, which would be like the reference benchmark for embedded modulation units. In both of them, the following modulation effects are provided, plus others: chorus, flanger, Leslie rotary speaker emulation, vibrato, phaser

. . . Still, the possibility to use non-sinusoidal oscillators, the capacity to modulate oscillator parameters with input data metrics and the possibility to output oscillator data directly seem to be original features of our unit.

With regards to sound quality, the results are not as stellar. The few subjects that have been able to compare our unit with either one of the aforementioned units, or another unit, including myself, either place at the same sound quality level than their reference chorus unit, or at a lower level. The only units that have placed worse than this unit in a compared test have been very cost-sensitive units. Therefore, some improvement should be placed on this aspect to obtain at least average reviews.

Chapter 5

Conclusions and future work

5.1 Conclusions

The research reported in this thesis reveals that additional features and enhancements to industry-standard chorus effects are generally appealing to the music community, and that it is more than possible to pack those additional features and enhancements in an embedded device. In particular, the usage of non-sinusoidal waveforms for delay modulation, including linear combinations of them, the increased number of delayed copies of the input signal, and the possibility to modulate oscillator parameters with metrics from the input signal, are interesting novelties that have been generally well received within the test subjects. Therefore, further efforts on this area could be directed by either the academic world or the music electronics industry.

5.2 Future work

Future work in this area should start by choosing upper quality elements for development. Choosing a daughter board with high-quality, stereo analog audio path with a decent *codec* should be a top priority -the *codec* used for this project is fine, though-, and if not available, designing one from scratch should be seriously considered.

Some features have been implemented in a non-optimal way due to the fact that the DSC chosen is barely up to the task. Redesigning this unit to work in a DSC with additional bandwidth and actually usable HW acceleration features could help this product to reach industry-standard sound quality, for instance through implementation of inter-sample interpolation, as suggested in [1], or through the usage of first-order noise shaping in

DF1 IIR filters [48][49]. Noise-floor decreasing techniques such as suggested in [4] should also be considered.

Additional features such as stereo inputs, outputs and processing path, and output feedback could also be desirable improvements.

I think that with the aforementioned improvements, this unit could be candidate to industrialization and commercialization.

Appendix A

Project source code

For the sake of brevity, the source code developed for this project has not been included. The latest version of this code can always be found at `https://bitbucket.org/gerard_amposta/pfc/`, while the version considered at the time of writing this thesis is tagged as `1.2.0`. Read access can be requested at gerard.amposta@gmail.com.

Appendix B

Test subjects demographics

Subject primary relationship with music	Number
Producer	3
Arranger	0
Audio engineer	1
Touring/studio musician	4
Amateur musician	10
Hobbyist	3

Table B.1: Subject relationship with music industry

Subject primary instrument	Number
Piano	2
Keyboard	1
Guitar	5
Classical guitar	1
Bass guitar	2
Vocals	4
Brass	1
Drums	5

Table B.2: Subject primary instrument

Subject musical education	Number
Classical degree	4
Modern degree	3
Individual classes	3
Self-taught	11

Table B.3: Subject musical education

Subject musical interests	Number
Classical	3
Rock	10
Pop	3
Flamenco	1
Jazz	4

Table B.4: Subject musical interests

Bibliography

[1] **Datorro, J.**, *Effects Design*, Journal of Audio Engineering Society, Vol. 45, No. 10, 1997.

[2] **Zölzer, U.**, DAFX - Digital Audio Effects, John Wiley & Sons Ltd., 2002.

[3] **Rocchesso, D.**, *Fractionally-addressed delay lines*, 2000.

[4] **Fernndez-Cid, P.** and **Casajs-Quirs, F.**, *Enhanced Quality and Variety for Chorus/Flange Units*.

[5] https://en.wikipedia.org/wiki/Delay_(audio_effect)#Digital_delay.

[6] https://ccrma.stanford.edu/~jos/cfdn/Feedback_Delay_Networks.html.

[7] https://en.wikipedia.org/wiki/Vibrato.

[8] https://en.wikipedia.org/wiki/Tremolo_(electronic_effect).

[9] https://www.music.mcgill.ca/~gary/307/week2/filters.html.

[10] http://www.gaussianwaves.com/2010/11/moving-average-filter-ma-filter-2/.

[11] **J. Smith, P. C.**, *The second order digital waveguide oscillator*, International Computer Music Conference, pp. 150-153, 1992.

[12] **et at., J. S. I.**, *System and method for real time sinusoidal signal generation using waveguide resonance oscillators*, 1997.

[13] **Turner, C. S.**, *Recursive discrete-time sinusoidal oscillators*, IEEE Signal Processing Magazine, May 2003.

[14] **Lyons, R. G.**, Streamlining digital signal processing, IEEE Press, 2012.

[15] **Goldberg, D.**, *What Every Computer Scientist Should Know About Floating-Point Arithmetic*, https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html, 1991.

[16] **John G. Proakis, D. G. M.**, Digital Signal Processing, Prentice-Hall Intl. Inc., 1996.

[17] http://www.microchip.com/Developmenttools/ProductDetails.aspx?PartNO=DM330011.

[18] http://ww1.microchip.com/downloads/en/DeviceDoc/70286C.pdf.

[19] http://www.microchip.com/wwwproducts/en/dsPIC33FJ256GP506A.

[20] http://www.ti.com/lsds/ti/microcontrollers_16-bit_32-bit/c2000_performance/overview.page.

[21] http://www.ti.com/product/tms320F28377S.

[22] http://www.ti.com/tool/LAUNCHXL-F28377S.

[23] https://www.arm.com/products/processors/cortex-m/cortex-m4-processor.php.

[24] http://www.nxp.com/products/microcontrollers-and-processors/arm-processors/kinetis-cortex-m-mcus/k-series-performance-m4:KINETIS_K_SERIES.

[25] http://www.st.com/content/st_com/en/products/microcontrollers/stm32-32-bit-arm-cortex-mcus/stm32f4-series.html?querycriteria=productId=SS1577.

[26] http://www.infineon.com/cms/en/product/microcontroller/32-bit-industrial-microcontroller-based-on-arm-registered-cortex-registered-m/32-bit-xmc4000-industrial-microcontroller-arm-registered-cortex-registered-m4/channel.html?channel=db3a30433580b3710135a03abaf9385e.

[27] http://www.analog.com/en/products/processors-dsp/sharc.html.

[28] http://www.analog.com/en/products/processors-dsp/sigmadsp-audio-processors.html.

[29] http://www.openmusiclabs.com/projects/audio-codec-shield/arduino-audio-codec-shield/.

[30] http://www.mikroe.com/add-on-boards/audio-voice/audio-codec-proto/.

[31] https://en.wikipedia.org/wiki/Q_(number_format).

[32] *MPLAB C Compiler for PIC24 MCUs and dsPIC DSCs*, http://ww1.microchip.com/downloads/en/DeviceDoc/51284H.pdf.

[33] https://en.wikipedia.org/wiki/IEEE_floating_point.

[34] *MPLAB Starter Kit for dsPIC Digital Signal Controllers*, http://ww1.microchip.com/downloads/en/DeviceDoc/51700B.pdf.

[35] https://www.cirrus.com/en/pubs/proDatasheet/WM8510_v4.5.pdf.

[36] http://www.ti.com/lit/ds/symlink/lm4811.pdf.

[37] http://es.mathworks.com/help/signal/ug/iir-filter-design.html.

[38] https://help.scilab.org/docs/6.0.0/en_US/iir.html.

[39] https://en.wikipedia.org/wiki/Karatsuba_algorithm.

[40] https://en.wikipedia.org/wiki/Central_limit_theorem.

[41] https://www.freesound.org/people/Sub-d/sounds/47464/.

[42] http://www.mooeraudio.co.uk/p/mooer-ensemble-king-analog-chorus-pedal?pp=24.

[43] https://www.boss.info/us/products/ce-5/.

[44] http://www.tcelectronic.com/corona-chorus/.

[45] http://www.strymon.net/products/mobius/.

[46] https://www.eventideaudio.com/products/stompboxes/chorus-flanger/modfactor.

[47] http://www.tcelectronic.com/toneprint/.

[48] http://dspguru.com/dsp/tricks/fixed-point-dc-blocking-filter-with-noise-shaping.

[49] https://en.wikipedia.org/wiki/Noise_shaping.