UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

RODRIGO DALTOÉ MADRUGA

Dual-microcontroller Direct Digital Synthesis system based on low-cost and off-the-shelf parts

Porto Alegre
2022

RODRIGO DALTOÉ MADRUGA

Dual-microcontroller Direct Digital Synthesis system based on low-cost and off-the-shelf parts

Work presented in partial fulfillment of the requirements for the degree of Bachelor in Computer Engineering

Advisor: Prof. Dr. Marcelo de Oliveira Johann

Porto Alegre
2022

CIP — CATALOGING-IN-PUBLICATION

Madruga, Rodrigo Daltoé

Dual-microcontroller Direct Digital Synthesis system based on low-cost and off-the-shelf parts / Rodrigo Daltoé Madruga. – Porto Alegre: 2022.

53 f.

Advisor: Marcelo de Oliveira Johann

Trabalho de conclusão de curso (Graduação) – Universidade Federal do Rio Grande do Sul, Escola de Engenharia. Curso de Engenharia de Computação, Porto Alegre, BR–RS, 2022.

1. DDS, microcontroller, sound synthesis, I2S, polyphonic. I. Johann, Marcelo de Oliveira, orient. II. Título.

Dedico este trabalho à Maria, ao Milton, à Martina e à minha paixão por música.

ACKNOWLEDGEMENTS

First of all, I would like to thank my advisor, Marcelo de Oliveira Johann, who not only helped me a lot with the many specific details of sound synthesis and music theory but also was always available to help and patiently answered the many questions I had about this art I love but still don't understand as much as I wanted.

To Francisco Knebel, Rodrigo Neves and Leonardo Costa, classmates and dear friends without whom I definitively wouldn't be able to achieve the title of engineer. To Alexandre Saccol, Lucas Flores and Cassio Fachinelli who helped me in the many times computers got me confused and lost. To Dirceu Bueno, Lucas Jantsch, Pedro Fetter, Luci Bongiorni and Yi Chen Wu who now I'll finally be able to call professional colleagues. To Matheus Perius, who made the task of working full time and studying less of a pain.

To Maria, Milton, Guilherme and all my family, who gave me unconditional support to study and focus in what was important. To Martina, who were my safe haven and gave me strength throughout the hardest part of my journey to become the one thing I wanted to become since childhood: An inventor who is capable of creating things that sing and fly.

Finally, to Bob Miller and the many anonymous in the many different communities and forums spread across the entire globe that helped me answer many questions and guide me to wisdom. You all are the reason I still believe in a better society through cooperation and kindness.

AGRADECIMENTOS

Em primeiro lugar, gostaria de agradecer ao meu orientador, Marcelo de Oliveira Johann, que não só me ajudou muito com os diferentes detalhes específicos à síntese de som e teoria musical como também sempre esteve disponível para me ajudar e responder as várias perguntas que eu fiz sobre essa arte que eu amo mas ainda não compreendo tanto quanto gostaria.

Gostaria de agradecer de coração à Francisco Knebel, Rodrigo Neves e Leonardo Costa, colegas e sem amigos os quais eu definitivamente não seria capaz de alcançar o título de engenheiro. À Alexandre Saccol, Lucas Flores e Cassio Fachinelli que me ajudaram nas várias vezes em que computadores me deixaram confuso e perdido. À Dirceu Bueno, Lucas Jantsch, Pedro Fetter, Luci Bongiorni e Yi Chen Wu, que agora eu vou poder finalmente chamar de colegas de profissão. À Matheus Perius, que tornou a tarefa de trabalhar turno integral e estudar menos sofrida.

À Maria, Milton, Guilherme e à toda minha família, que me deram suporte incondicional para estudar e focar no que era importante. À Martina, que se tornou meu porto seguro e me deu forças através da parte mais difícil da minha jornada em me tornar o que eu queria desde criança: Um inventor capaz de criar coisas que cantam e voam.

Finalmente À Bob Miller e tantos outros anônimos nas diferentes comunidades e fóruns espalhados no globo que me ajudaram a responder várias perguntas e me guiaram para a sabedoria. Vocês são a razão pela qual eu ainda acredito em uma sociedade melhor através da cooperação e gentileza.

“There’s only one thing that can save a man from madness and that’s uncertainty.”

— Dmitry Glukhovsky

ABSTRACT

Direct Digital Synthesis (DDS) is a signal synthesis method designed to generate arbitrary waveform from a single, fixed reference clock coupled with a memory device and a Digital-to-Analog Converter (DAC). This method can be used to synthesize basic sounds based on their recorded waveforms. Direct Digital Synthesis can be implemented not only in hardware but in software as well. The limitations being the memory and processing power of the processor involved in running the synthesis software. Later with the use of electronic devices, these systems became smaller and more powerful, with many possibilities of configurations. Microcontrollers are small computers that contain one or more Central Processing Units (CPU), memory and many different types of peripherals, all in the same package. There are many different types of microcontrollers architectures, for different applications and price ranges. Application-Specific Integrated Circuits (ASIC) designed to run a hardware DDS system are often expensive and not readily available. Microcontrollers on the other hand have become much more powerful, memory and performance wise, much more cheaper and available. This work proposes a sound synthesis system running a software DDS architecture utilizing a pair of off-the-shelf, low cost microcontrollers, in order to make the design easy to replicate, to reprogram the firmware and to change the overall interface design and functionality.

Keywords: DDS, microcontroller, sound synthesis, I2S, polyphonic.

Sistema de Síntese Digital Direta com dois microcontroladores baseado em partes de alta disponibilidade e baixo custo

RESUMO

Síntese Digital Direta (DDS) é um médoto de síntese de sinal criado para gerar formas de onda arbitrárias através de um único *clock* de referência fixo, um dispositivo de memória e um conversor digital-analógico (DAC). Esse método pode ser utilizado para sintetizar sons básicos, baseados nas gravações de suas formas de onda. DDS pode não só ser implementado em hardware mas também em software. As limitações da implementação em software são o limite de memória e o limite de poder de processamento envolvidos no processo de síntese. Mecanismos de síntese de som existem muito antes do campo da Eletrônica em si existir, começando com instrumentos musicais básicos conectados com diferentes mecanismos para no fim controlar as várias propriedades do som. Com o passar dos anos e com o uso de dispositivos eletrônicos, os vários mecanismos de geração de som se tornaram menores e mais poderosos, com várias possibilidades de configuração. Os sintetizadores dos anos 70 eram compostos por vários osciladores, filtros e amplificadores controlados por tensão (VCA), todos controlados pelas várias entradas de controle do usuário. Microcontroladores são pequenos computadores que contém uma ou mais unidades centrais de processamento (CPU), memória e diferentes tipos de periféricos, tudo no mesmo *package*. Existem vários tipos de arquiteturas de microcontroladores, para diferentes aplicações e faixas de preço. Circuitos integrados de aplicação específica (ASIC) desenvolvidos para executar síntese por via de sistema DDS são geralmente caros e difíceis de serem obtidos. Em contraste, microcontroladores se tornaram cada vez mais potentes com o passar dos anos em relação à memória e processamento, enquanto seus preços diminuem e sua disponibilidade aumenta. Esse trabalho propõem um sistema de síntese de som através de DDS em software utilizando um par de microcontroladores de alta disponibilidade e baixo custo, tendo como objetivo tornar o sistema fácil de replicar, de reprogramar o firmware e de mudar o design e funcionalidades da interface de usuário.

Palavras-chave: DDS, microcontrolador, síntese de som, I2S, polifonia.

LIST OF FIGURES

LIST OF ABBREVIATIONS AND ACRONYMS

ADC	Analog-to-Digital Converter
ADSR	Attack/Decay/Sustain/Release
ASIC	Application-Specific Integrated Circuits
AM	Amplitude Modulation
CPU	Central Processing Unit
DAC	Digital-to-Analog Converter
DDS	Direct Digital Synthesis
DSP	Digital Signal Processors
I^2S	Inter-IC Sound
IC	Integrated Circuit
I/O	Input/Output
FCR	Frequency Control Register
FM	Frequency Modulation
GPIO	General Purpose Input/Output
HAL	Hardware Abstraction Layer
LFO	Low Frequency Oscillator
LUT	Look-Up Table
MCU	Microcontroller Unit
MPU	Microprocessor Unit
NCO	Numerically-controlled oscillator
RLF	Reconstruction Low-pass Filter
RTOS	Real-Time Operational System
SoC	System-on-a-chip
SPI	Serial Peripheral Interface

UART Universal Asynchronous Receiver-Transmitter

UI User Interface

VCA Voltage Controlled Amplifier

VCF Voltage Controlled Filter

VCO Voltage Controlled Oscillator

CONTENTS

1 INTRODUCTION

This work is the culmination of years of personal interest in music, how to generate music using electronics systems and specifically the Deep Note (THX, 2022) and how it was created. The Deep Note is not only a very interesting sound, but a very famous and complex sound pattern created by the sound engineer James A. Moorer (MOORER, J. A., 2022) and it consists of 30 voices that change frequency and volume over time in order to form a specific pattern. In order to work with the generation of the Deep Note a system capable of working with many voices that change volume and frequency over time would have to be created. One device capable of these functions is a synthesizer. This work aims at evaluating the process of designing a synthesizer using microcontrollers and the results of such a design, as an instrument and as a computing system.

A synthesizer, or synth as it is also called, is an electronic device capable of generating audio signals of specific frequency, amplitude and timbre, according to user input (KLEIN, 1982). Not only that, it can be designed to modify those parameters and to apply many effects to the resulting signal. They are not the first kind of equipment or mechanism to generate sound in a more controlled and automated manner. Sound creating mechanisms exist prior to the existence of the field of Electronics itself, starting with basic musical instrument parts rigged with different mechanisms, first pneumatic then electric, in order to control the many properties of the sound generated. Electromechanical devices helped a lot in the creation of many sound devices in the first documented machines back in the 19th century. The invention of vacuum tubes allowed synthesizers to finally be free of mechanical parts for the generation of sound but the biggest leap in sound synthesis was the invention of the transistor and transistorized devices and integrated circuits. Transistors helped to miniaturize the synthesizers internal circuits in size and integrated circuits helped all inventors and engineers to achieve complicated signals and functions with less complex topologies.

Synthesizers don't necessarily have a fixed structure of architecture, but are comprised of different internal components which work on the generation and modification of the audio signal. These components can be controlled by the user or by another electronic component (PUCKETTE, 2007). The different components of a synthesizer are often called modules and they can be mounted together in what is called a modular synth. Modular synthesizers in the early 70's were of great importance for the music and culture of an entire era, changing how many music genres sounded forever. Together with ampli-

fiers, synthesizers also helped to turn music into a more popular and democratic media, with electronic components getting cheaper, smaller and more available over the years and knowledge about electronic circuits being more widespread in the music community.

The first synthesizers were of course purely analog. After the selection of the chosen frequency to be played all the circuitry that was used in the different parts of the synthesizers were designed with active components such as operational amplifiers, transistors and diodes and passive components as resistors, capacitors, inductors and many different kinds of buttons and switches (KLEIN, 1982). The operation of the components of the synthesizer could be described by differential and algebraic continuous equations, as summing, multiplication, integral and derivative.

Besides being capable of working in analog electronic circuits, vacuum tubes and transistors can operate in digital electronic circuits as well. They can be used as switches in order to implement a binary system, working with only two possible voltage levels. Instead of using the whole range of the power supply used to energize the circuit, basic digital circuits only operate with fully on and fully off levels. The operation of digital components can be described by discrete equations. Each variable that operates with binary levels is called a bit. Although analog voltage levels won't work with digital circuits, the circuits themselves can use analog values as inputs through the use of Analog-to-Digital Converters (ADC) and can output multi-bit values with the help of Digital-to-Analog Converters (DAC) (HOROWITZ; HILL, 2015).

Digital circuits, just as the analog counterparts, can be mounted on a board via discrete components or can be created in an integrated manner in the same silicon die, called Integrated Circuits (IC). It is common to integrate the core of an application-specific circuit into a Application-Specific Integrated Circuit (ASIC). ASICs are useful for they integrate all the essential functions of the circuit in a small form factor and at the same time allow for customization of some parameters. The downside of ASICs is mainly their higher price due to a more specific function, leading to a smaller number of produced parts. Another method to perform specific tasks with digital circuits is to use programmable devices. One largely widespread digital device designed to work with a programmable memory is a Microprocessor Unit (MPU).

Microprocessors are comprised of one or more CPUs devices and work alongside one or more memory devices (HOROWITZ; HILL, 2015). They work by reading, executing and writing the result of computer instructions. These instructions are written by a programmer in order for the microprocessor to perform a determined task. Mi-

croprocessor too have been popularized and now are far more accessible and more customizable than in any part of history. There are many types of microprocessors in the electronics industry and one of them is a device designed to work directly with electronic components and circuits: The microcontroller unit (MCU). A modern microcontroller is a small computer comprised of a CPU, volatile memory, flash memory and many electronic peripherals such as timers, Analog-to-Digital Converters (ADC), General Purpose Inputs/Outputs (GPIO) and communication protocol drivers in a single silicon die and package. A microcontroller-based system with a DAC is capable of generating audio signals, given the system has enough processing power and memory capacity.

Something important to note in this work is that some engineering choices were made with the current electronics supply chain crisis that started in 2020. Availability is more important than ever to all embedded systems designs being worked on in the entire globe since a lot of microcontrollers and other active/integrated are not available anymore. Parts costs increased so drastically that's another specification that will be rigorously taken into account when it comes to choose a part to be used in the proposed design. All parts chosen for this work, from microcontrollers to resistors, were extremely available, theoretically in the entire planet via local shopping or in more extreme cases via international shipping and cheap as far as minimum desired performance allowed. Even though prices for electronics parts soared in the last 2 years still this entire projects circuit board would not cost more than US$ 70.00 in Brazil (R$ 330.00) including the expensive display chosen only because one was already available in the laboratory. A much cheaper version could be built using a smaller LCD display.

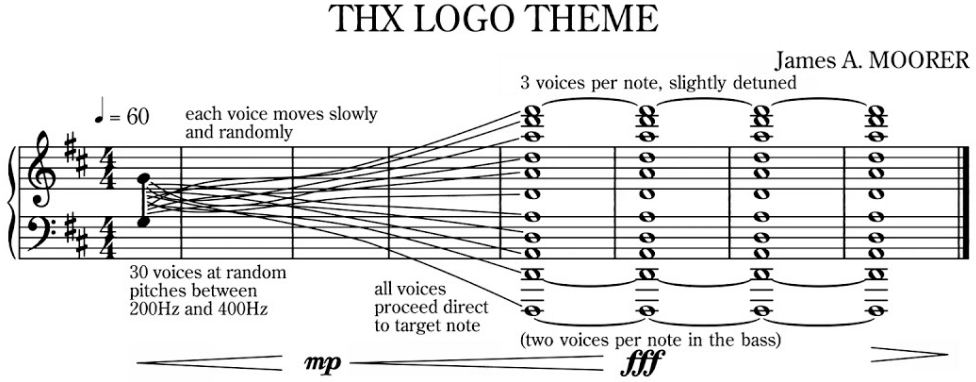
2 BACKGROUND

The following sections will describe and define some key concepts involved in generating audio signals and the basic functions of a synthesizer.

2.1 The Deep Note

In 1982, James 'Andy' Moorer created the Deep Note. It then became known worldwide as the eerie crescendo that announces the THX logo before every THX- certified movie. The distinct sound was created from 20,000 lines of C code that generated a 250,000 lines score to be played in the Audio Signal Processor (ASP) (MOORER, J., 1982), a sub-assembly of the Audio Signal Processing Station in Lucasfilm Computer Division. The Station was a semi modular self-contained unit that was composed of several major sub-assemblies, one which was the ASP. The ASP was composed of a controller and up to eight Digital Signal Processors (DSP), ASICs that are designed to process digital signals, in that case, audio signals. The Note then debuted in 1983 in the movie "The Return of the Jedi" and since then it is known for testing every single sound system that tries to play it. It consisted of 30 voices spanning over 3 octaves as it can be seen in Fig. 2.1.

Figure 2.1 – The Deep Note



Source: THX Deep Note

The Deep Note is a group of voices, or audio channels, that change in frequency and volume over time, in order to create the pattern shaped by Mr. Moorer. Any equipment or software that will run a version of the Deep Note needs to be capable of 3 different features: First it needs to be capable of synthesizing many different voices in parallel, to be mixed together. Then it needs to be able to control each voices' volume in order to

create the crescendos. And at last it needs to be able to shift the pitch (the frequency) of each voice over time in order to create the glissandos that characterize the Deep Note. Although this project won't be designed to specifically and only perform the Deep Note, it is desired that it will be able to perform all the three tasks needed for the Deep Note to be played, so the implementation of a "Deep Note function" would be made possible.

2.2 Audio Signals

An audio signal is an electronic signal that represents a certain sound wave, that is, it is intended for us humans to hear. Any mechanical wave between 20Hz and 20kHz can be categorized as sound (HARTMANN, 2013). An audio signal is an electric equivalent of a sound wave. Sound waves can be transformed in audio signals by a transducer, like a microphone. Audio signals can be transformed back into sound wave via another transducer as a loudspeaker. A sound wave has four basic parameters: Amplitude, frequency, phase and shape. The first three are the basic parameters to any sinusoidal wave and but only amplitude, frequency and shape are key to sound synthesis as phase is rarely used as a controlled parameter.

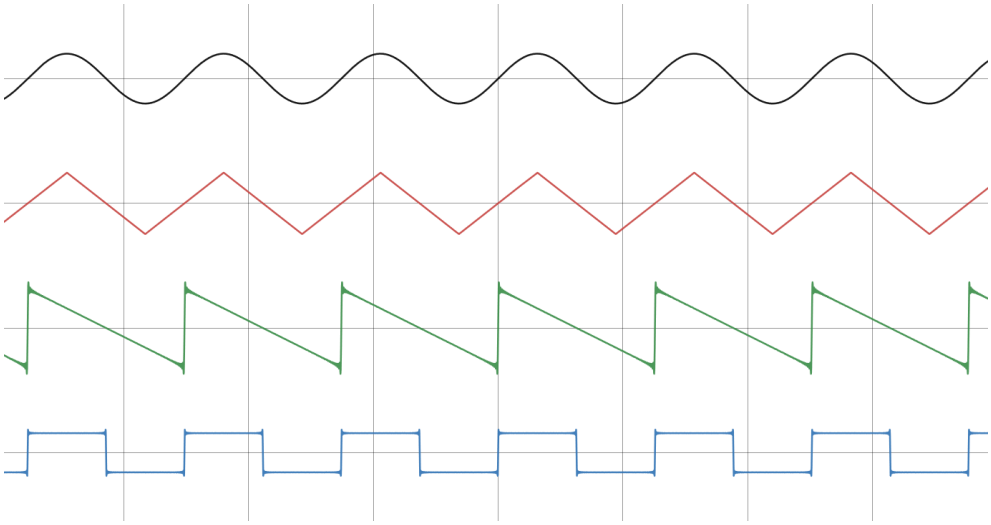
Amplitude determines how loud the audio signal will be perceived when it is heard through the use of some transducer, a device that allows the conversion of electric signals into sound. It's very easy to control this parameter for amplitude control is mainly just a basic operation of multiplication.

Frequency is the parameter that controls the pitch of the perceived sound. More basic synthesizers can be comprised of a single basic oscillator with a continuous frequency control. In order to make the task of using the synthesizer in a music easier for the player, synthesizers will use a discrete selection of frequencies to be generated which can be selected by the use of a keyboard. The most common way of defining what frequencies each key should generate is to design the internal oscillators according to the Diatonic Scale with equal temperament. The main advantage is that in electronic instruments, the reference key or octave can be configured easily, so the same key can generate different frequencies depending on what the player wants to play.

When describing a single audio signal as a graph over time one secondary characteristic is added to the three main parameters: Shape. Shape can be defined as the form of the wave of the sound recorded when plotted in a graph over time. An audio wave can be sinusoidal, triangular, squared, in the shape of a saw tooth or any other shape.

The shape of the wave will change it's spectrum, thus changing the way it sounds. The Fourier Theorem states that all periodic waveforms can be represented as a sum of sinusoidal waves (ALEXANDER; SADIKU, 2013). Depending on which harmonics of the wave (harmonics are waves with a frequency multiple of the frequency of the main wave) are added together a new wave can be formed. In the end all waveforms are a sum of sines and cosines. In the Fig. 2.2 all basic waveforms were formed by summing their respective harmonics to the 100th harmonic. Analyzing all waves in the frequency domain is not practical though. Shaping them in the time domain helps to identify how they sound and to construct their mathematical functions without having to resort to frequency domain analyzes. As synthesizers started as analog devices the first shapes used in audio synthesis were the ones with a simple mathematical model as the sinusoids and squares. Analyzing the shapes of the waveforms was relatively easy given oscilloscopes were a common technology.

Figure 2.2 – The four basic waveforms used in sound synthesis



Source: Image provided by author

Even though it is not always categorized as an audio signal, depending on the context in which this signal may appear, noise is also a kind of signal that can be used in audio synthesis and it plays an important role not only when added to other audio signals but also as a parameter control signal. Noise can also be shaped, but not as described previously. Instead as having a specific shape when plotted in a graph though time noise can be shaped differently in the frequency domain. How the noise is distributed throughout the frequency spectrum can change how it sounds. The most basic form of noise is the so called White Noise for its flat distribution. There are also other so called colors of noise such as Pink Noise, Brown noise and Grey Noise.

2.3 Basic Synthesizer Modules

The main component of a synthesizer would be an oscillator. It generates an electronic signal with a specific frequency, shape and amplitude, which can be turned into sound via loudspeakers or headphones. The synthesizer player can use a variety of electronic parts and sensors to determine the many properties of the signal. Potentiometers can be used to define the frequency of the signal but the main technique to choose what frequency will be played is to use a keyboard, just like a piano, but with electronic contacts underneath the keys to be used as inputs to the oscillator. The keys can have one electronic contact underneath so the oscillator can detect which frequency from a discrete selection of frequencies it should play or two electronic contacts in order to determine how fast the player pressed the key. The amplitude of the signal can be determined by some static parameter, like a potentiometer or switch, or by the key press velocity. The shape is usually chosen between different types, depending on what kind of oscillator is being used. A synth can be comprised of only one or many oscillators, in order to generate many sound signals simultaneously.

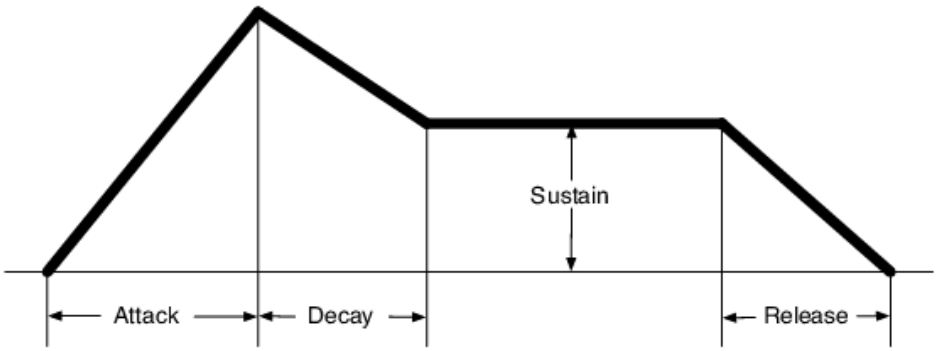
Although a synth can be designed only using oscillators, another important part of a basic synth is the filter. It's function is simple: To attenuate some frequencies of the generated signal while maintaining the voltage level of others. Filter can be designed to attenuate lower frequencies, higher frequencies or an specific range of frequencies. A synth capabilities are greatly improved with the addition of amplifiers. They can control a value which multiply the sound generated by the oscillators.

All the parameters of the oscillators, filters and amplifiers can be statically controlled by potentiometers or switches or dynamically controlled by another voltage. The resulting devices with the voltage inputs are called Voltage-Controlled Oscillators (VCO), Voltage-Controlled Filters (VCF) and Voltage-Controlled Amplifiers (VCA) respectively. Extra oscillators can be used to control the synth components' parameters. These oscillators are often called Low-Frequency Oscillators (LFO) (KLEIN, 1982).

Through the use of a VCA, the audio signal can be shaped through time in an envelope. Instead of generating sound when the key is pressed and stopping the sound when the key is released in a binary manner, the volume of the audio signal will be determined by a pattern. This pattern can take many parameters and shapes but the most common is the ADSR envelope which can be seen in Fig. 2.3.

The ADSR stands for the 4 phases of the envelope: Attack, Decay, Sustain and

Figure 2.3 – ADSR envelope



Source: Image provided by author

Release. The Attack and Decay phases are responsible for emulating the impact of a player finger in acoustic instruments like a piano or guitar. The Sustain and Release phases are responsible for the natural fading of the sound. These four parameters are electronically controlled and can be manipulated by the user or another electronic system connected to the synthesizer.

2.4 Basic Analog Synthesis Methods

There are four basic sound synthesis techniques: Subtractive Synthesis, Additive Synthesis, Frequency Modulation Synthesis and Memory-based Synthesis. Many other synthesis techniques exist and some are derived from this main four techniques (PUCKETTE, 2007).

Subtractive Synthesis consists of generating any kind of audio signal and then filtering out some specific frequencies. The audio signal to be filtered is usually one of a very rich spectrum or even noise for its spectral distribution. This project will not operate with subtractive synthesis although it would be possible.

Additive Synthesis is the simplest form of synthesis. It consists of simply adding sinusoidal signals in order to compose a more complex and rich sound. It can be also implemented with other waveforms. The synthesis method will be used extensively in the proposed DDS engine.

Frequency Modulation Synthesis is a synthesis technique that consists in modulating the frequency of a main audio signal in order to obtain a very complex audio signal. One way to achieve this modulation is to use a voltage controlled oscillator although this method is not used with LFOs, for they are usually not stable enough when operating at

higher frequencies. The correct way of synthesizing audio signals with FM synthesis is to modulate a VCO using another VCO. Another oscillator's output is used as frequency control of the main oscillator. This process is the same used for FM radio, although in a much lower frequency range. This work will not try to operate FM synthesis in its DDS core.

Memory-based Synthesis can be divided in two main categories: Sample Synthesis and Wavetable Synthesis. Sample Synthesis consists in playing a sampled sound, going forward in a loop, back and forth or by utilizing small pieces of the sample. It can be used in both analog and digital designs. Wavetable Synthesis is strictly digital. The entire DDS engine can be analyzed as a wavetable engine for all sounds are generated using memory-stored waveforms. More on that subject on the next sections.

2.5 Analog To Digital

Analog circuits' behaviour can be modeled according to continuous equations, be them linear or non-linear, differential or just simply arithmetical. Digital circuits' behaviour can be modelled according to discrete mathematics. Digital circuits operate with discrete values, computers being no exception. Analog values when read by a digital pin of a MCU will be converted to digital values. Digital pins cannot output analog values either. The interface between analog and digital systems can be made possible via ADCs and DACs (HOROWITZ; HILL, 2015).

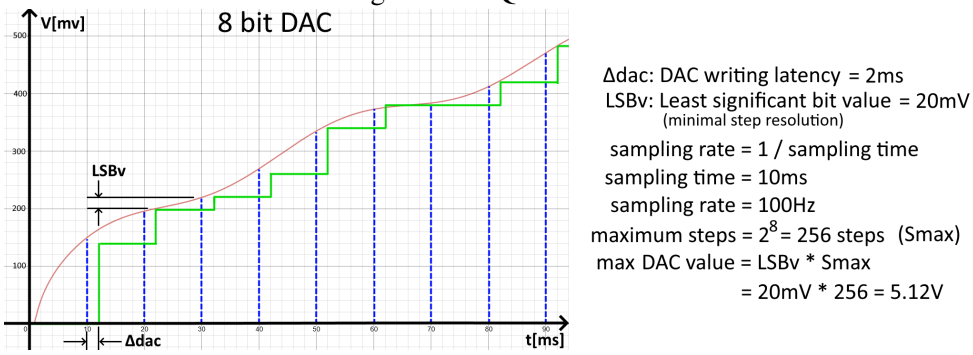
ADCs will convert an analog value to a digital one. Of course, this conversion will use more than just one bit, otherwise a single transistor would do the trick. Generally speaking ADCs outputs have 8 bits or more for the converted value. The process of converting an analog value into a digital one is called Quantization. DACs in the other hand do the opposite task: They convert a digital value into an analog one. As digital values are discrete, the converted analog value in the output of the DAC will have only a fixed number of possible values. Both ADCs and DACs have some parameters in common: Both need some time to execute the conversion therefore operating in a discrete timing, both will have a latency between the beginning and end of conversion process, both can be characterized by their Sampling Rate and Bit-Depth (the resolution of the conversion) and both can be evaluated by their linearity and noise-level.

The Sampling Rate is the frequency in which the ADC or DAC can execute the conversions. Each different converter will have a maximum Sampling Rate which in it's

turn will limit the maximum frequency the system will be able to read/write.

The Bit-Depth is the number of bits used in the digital part of the conversion and it directly corresponds to the resolution of each sample. The smaller the resolution, the better, for the system will be able to read/write signals with more detail. The resolution depends on the maximum voltage achievable by the analog value and the Bit-Depth. For example, picture a 8 bits DAC with a 2ms latency operating with a maximum range of 5.12V and a Sampling Rate of 100Hz. As the DAC is operating with 8 bits, it will have 256 possible values of output. With 5.12V as a maximum value, the smallest output possible is 5.12V / 256 steps which is 20mV. That is the resolution. A better visualization of the example can be seen in Fig. 2.4

Figure 2.4 – Quantization



Source: Image provided by author

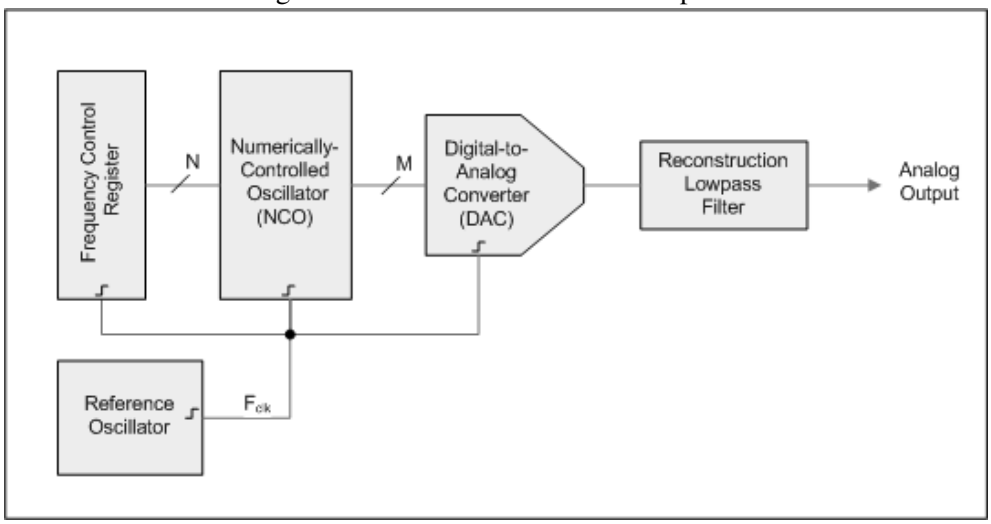
Although it's always good to have the most powerful converter regarding Sampling Rate and Bit-Depth it is important to note that there are some practical limits to both parameters. An increase in a converter's Sampling Rate won't necessarily increase the system's performance if the system cannot keep up with the data flow. The system's features and limitations will define what desirable Sampling Rate the chosen converter will need to have. The same happens with the Bit-Depth. A super small resolution won't be noticed in the final performance of a system if the system is too noisy.

2.6 Digital Synthesis

The digital synthesis technique discussed in this work will be the Direct Digital Synthesis (DDS). There are some other techniques to generate arbitrary digital signals but DDS is the typical architecture and by far the most used. A DDS system consists of a Frequency Reference in form of a clock, a Frequency Control Register (FCR) which controls the period of the desired signal to be synthesized, a memory device called Numerically-

controlled oscillator (NCO), a previously discussed DAC and a Reconstruction Low-pass Filter (RLF), as in the Fig. 2.5. In a DDS architecture the NCO stores one single period of the shape of the desired waveform to be generated (GENTILE; CUSHING, 1999).

Figure 2.5 – DDS architecture description



Source: Image provided by author

The Frequency Reference is the master clock that is responsible for synchronizing the entire system and it determines the frequency accuracy of the DDS. The NCO is the component that contains the digitized form of the waveform in a form of a table containing each value to be converted by the DAC. The NCO is sometimes called a Look-Up Table (LUT). The FCR can be used to determine the period of the generated signal by controlling how many positions of the NCO are skipped each time a sample is generated. In order to generate faster signals, the FCR increases its value and it takes less time for the NCO to complete a loop and go back to the initial position. The FCR usually uses a floating point variable but can be used with a fixed point variable to increase performance as a algorithm using floating point value takes more processing power depending on the CPU's architecture. The RLF is used to attenuate the undesired noise inherent of the analog continuous value to digital discrete value conversion. This noise is called Quantization Error. Important to note that the RLF is not capable of filtering aliasing error. According to Nyquist–Shannon sampling theorem aliasing can occur when a system tries to reproduce a signal that has higher frequency than 2 times the sampling frequency of the system itself (HOROWITZ; HILL, 2015).

It is of great importance to notice that as the DDS synthesis is digital in contrast to the more traditional methods of sound synthesis, error will be introduced. The error associated with the digital-to-analog conversion, previously discussed, is called quantization

error. It refers not only to the conversion from the analog signal to a digital signal, with a loss of information within the process, but also to the temporal characteristics of the conversion. The conversion only occurs in discrete times. If an analog signal has important information in between the sampling times, that information will be lost. In the process of synthesizing signals, the digital computer will try to achieve a determined frequency, in order to imitate the analog signal. With intermediate frequencies, that would need a higher sampling rate, aliasing (variation or error on the periodicity of a signal result of the quantization process) will occur. This aliasing will be noticeable by the human ear and there are many techniques to attenuate it, though they will not be discussed in this project.

2.7 Software Synthesis and The MCU

The DDS architecture can be easily implemented via software. The NCO will be just a Look-Up Table, an array with pre-determined values. The Frequency Control Register will be just a single variable responsible for storing the phase value that will be incremented in the reading of the LUT each sample period. As the NCO will be called a LUT, the FCR will be called a phase accumulator in the software version of a DDS system. As the increment of the LUTs index and the reading of the corresponding LUTs value is all done in software, the Reference Oscillator is simply the period in which the MCU sends the data to be converted to the DAC. The sampling rate in the software synthesis is limited by the capacity of the MCU to execute all the code necessary in time to send data to the DAC and be ready for the next cycle (FUNDAMENTALS..., 2009).

The DAC can be external or internal. Usually an internal DAC has Bit-Depth not bigger than 12 bits which is not enough for good sound synthesis. For a decent quality sound synthesis 16 bits is enough, but for a really good quality synthesis 24 bits is the standard. For that Bit-Depth there is a type of external DAC designed for sound applications, usually called Codec, as for coder-decoder. This kind of device communicates with the MCU using I^2S, a protocol similar to SPI but designed from the start to operate with sound data.

The easiest method to implement a software DDS architecture in a MCU is to use interrupts. In each interrupt the FCR is updated and the NCO outputs a value to the DAC. That has to be done before the next interrupt is triggered or these events will pile up and the core will lock processing only the interrupt routine or worse, a system error will occur. This limits the amount of processing that can be done unless the project is updated

to used a more powerful core or changes the sampling rate to a lower one. An alternative to this method is to use some sort of buffer to be filled by the CPU and loaded to the I^2S register. The main way of working with a intermediate buffer is to use Direct Memory Access (DMA). DMA is a feature computer systems have that allows certain hardware subsystems to access main system memory independently of the central processing unit (CPU) so the CPU can be busy executing other parts of the code while the DMA make the data transfers to the peripherals and back.

A DDS system can be used to generate an arbitrary waveform but what about more than one in a single channel? All it takes to achieve polyphony (more than one audio signal being played simultaneously) is more than one LUT and phase accumulator running in parallel, having their outputs mixed together and sent to the DAC. In parallel here means they would be processed via different variables but in a MCU core they would be processed one after another. This of course increases the overhead of the code. The more channels that need to be run in parallel the more processing the CPU needs to do before sending the resulting signal to the DAC. The limit of how many DDS channels an MCU can run is proportional to its processing power and inversely proportional to the sampling rate.

Sound effects such as reverb or low-pass filters can be applied to the output of the DDS before sending it to the DAC. This will also add an overhead and it needs to be taken into account before implementation. An emulation of a VCF can be done by simply running the filter function with the output of the DDS system and then sending it to the DAC.

Having an interrupt routine running at every sample period simulates the hardware DDS reference clock. The LUT and the phase accumulator are the software version of the FCR and NCO. The software DDS system described so far can emulate the behaviour of an analog oscillator: It can generate different waveforms at different frequencies. In order to change its frequency all it takes is to modify the phase accumulator variable. With this possibility it behaves as an VCO. Together with the VCF implementation the only module of a traditional analog synthesizer left is the VCA, a controlled amplifier. This can be achieved easily using a simple gain variable multiplying the output of the DDS. This variable can be changed dynamically, in order to make the gain controllable.

According to the Nyquist–Shannon sampling theorem, in order to reproduce a signal of frequency F it is needed a sampling rate of at least 2F. So in order to synthesize sounds we need a sampling rate of at least 40kHz. This leaves the CPU with a sample

time of 25μs. This 25μs time period is all the CPU has to process all the parallel DDS channels and run the main function of the program. In order to keep the interrupt from using the whole core or even worse, generating a general error and rebooting the MCU, the interrupt routine needs the be as small as the programmer can make it. Everything not time critical should be processed in the main function or on a background task.

The control functions such as the ADSR envelope and the Low Frequency Oscillator (LFO) need to run in a time controlled manner for they are time dependable. The LFO is an oscillator and the ADSR parameters are measured in fractions of seconds. The control system can run in the main loop with the use of unblocking polling mechanisms or a second interrupt, of a lower priority than the DDS interrupt. This secondary interrupt can be triggered in a much slower rate, 1kHz for example, as the ADSR envelope phases are usually defined in milliseconds and the LFO maximum frequency is usually 20Hz.

2.8 Related Work

There are many projects involving digital synthesis using microcontrollers. Even when microcontrollers were devices reserved for industry application only digital and analog sound synthesis were already popular with makers and musicians. Sound, or music more precisely, is a very popular topic with a lot of electronics enthusiasts and hobbyists.

Among the many works that came before this one, one is the main inspiration: The work of Bob Miller on his sound synthesis system called the DeepSynth (MILLER, 2018). The DeepSynth consists of a audio synthesis software running on a 1Bitsy, a prototyping board with a STM32F4 MCU in it. His project had 32 oscillators being processed at 44.1kHz, delivering an output of 12 bits at the internal DAC of the STM32F4. STM32 MCUs are know for being more than capable of being deployed in any embedded systems project but they are not as available nor cheap as the ESP32. Bob Miller's design is certainly not the first project aimed at synthesizing the Deep Note but it is the most impressive. It is capable of using Sheppard's tone distribution, a distribution of notes that emulate an endless scale, filtering, random audio pan and polyphony using all 32 oscillators.

Another source that was fairly used was the work of Jouko Vankka on "Direct Digital Synthesizers: Theory, Design and Applications" although Vankka's work is way above this one complexity wise (VANKKA; HALONEN; HALONEN, 2001).

3 PROPOSAL

The following sections will describe all the engineering choices, trade-offs and motives regarding the design of the proposed MCU software synthesizer.

3.1 Design Topology

The most usual way of designing a system using a MCU is to place the MCU in the center of the entire circuit and connect all the peripherals to it. Given that the number of pins of the entire User Interface (UI) is going to be high, a way of expanding the GPIO capabilities of the main MCU is going to be needed. The chosen solution was to use a low processing power, basic architecture, high pin-count MCU for the UI and a high processing power microcontroller for the sound generation, always keeping in mind that the aim of the project is not only to develop a synthesizer but also that the parts used are widely available, off-the-shelf and on the low side of the price list. A consequence of having a MCU for UI and another MCU for the sound engine is modularity: Each part of the synthesizer system can be changed without affecting the other, allowing for a more customizable design, given the protocol between the two devices stays the same. The communication between the two MCUs will be implemented using UART, for it is a simple device to configure and use.

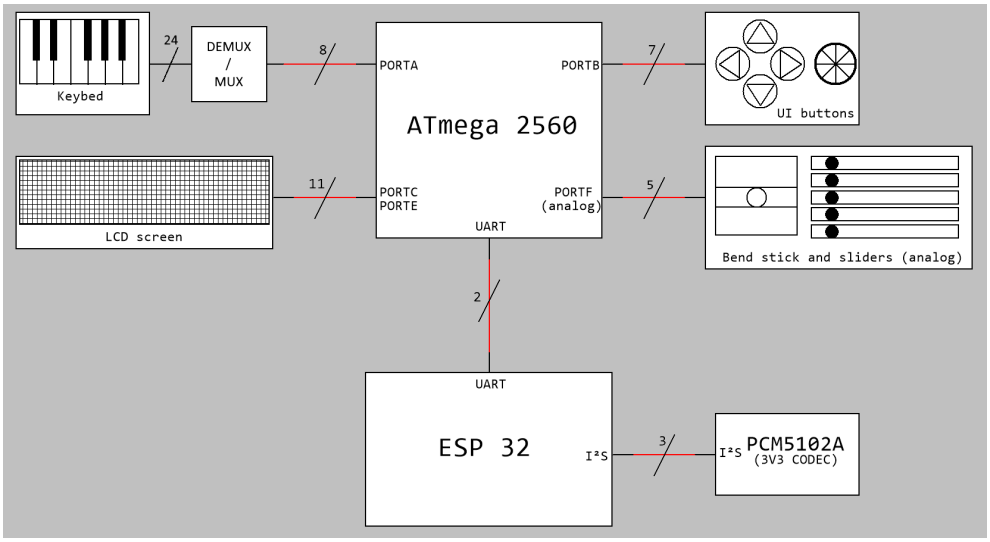
The basic blocks of the proposed solution are an UI, consisting of a display, buttons, one encoder and some potentiometers, a main input for the musical notes, a MCU to read all the inputs and write to the display, a MCU to work as a sound engine, processing all the inputs of the user and generating all the audio signals and a DAC to convert the digital data into analog signals. A diagram of the chosen topology can be seen in Fig. 3.1

The chosen voltage to run the whole synthesizer is 5V for the wide range of parts compatible with 5V supply and higher resistance of the inputs to noise compared to supplying the MCU with 3.3V.

3.2 Main Input

In order to design a musical instrument, the best input for an electronic device is a musical keyboard. It has all the 12 notes of the octaves disposed in line and is a music

Figure 3.1 – Block diagram of the proposed solution with all components already defined



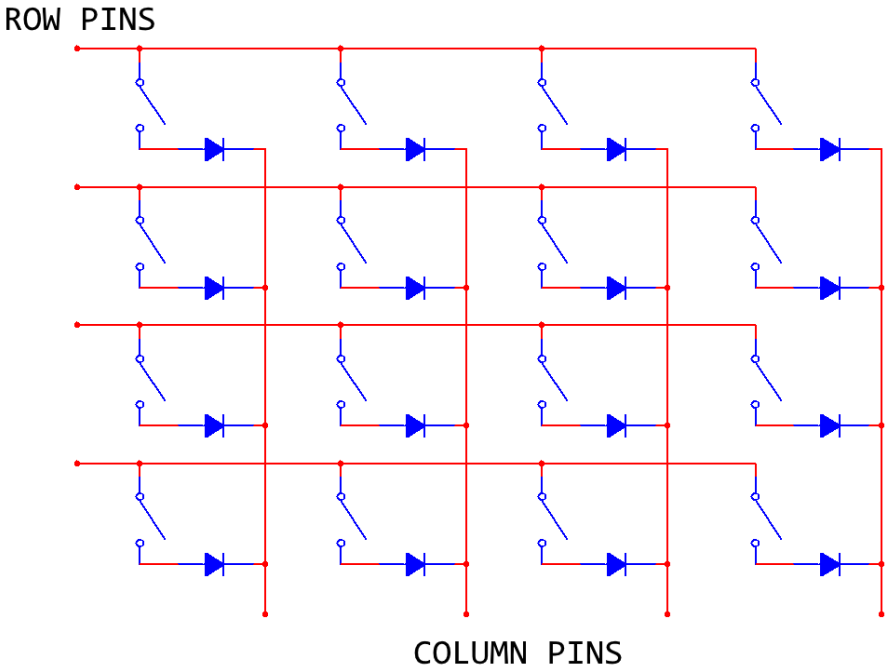
Source: Image provided by author

standard since electronic music keyboards drawn inspiration directly from the piano. As previously described musical keyboards can have one or two electrical contacts below the key in order to detect the key press and key press speed, in the two contacts case. As it is usual for keyboards to contain 4 octaves or more, with 12 keys per octave, this leads to a huge amount of electrical contacts. In order to make the reading of this big quantities of input less troubling the keyboard keys are already arranged in a board using a Key Matrix (or Scan Matrix) topology. In this way instead of using 2N inputs in the MCU if the chosen design would be direct reading of each contact, only $\sqrt{N}$ inputs are necessary. A 4x4 key matrix can be seen as example in Fig. 3.2

The connections of the Key Matrix are arranged in rows and columns. In order to detect a key press, the MCU will scan all the rows and read the column pins in order to detect which key was pressed. The diodes are inserted in the circuit to prevent what is called "ghosting", when on key press generates more than one output pin to be turned on. By scanning row and column pins the MCU can detect any key press in the matrix. This method of assembly and scanning is also used in computer keyboards.

The chosen keyboard for the experimental phase of this work had 5 octaves, resulting in 61 keys (60 keys plus an extra one). Managing to read all the electrical contacts separately would result in 122 inputs. The use of a Key Matrix reduces the number of pins required from 122 to 8 plus 16, 8 for the columns and 16 for the 2 groups of rows (double electrical contact per key), totalling 24 pins. This reduced the number of pins needed to read the keyboard significantly but it is still a lot of pins to operate considering the system will also have to operate a User Interface (UI) as well. There are many ways

Figure 3.2 – 4x4 Key Matrix



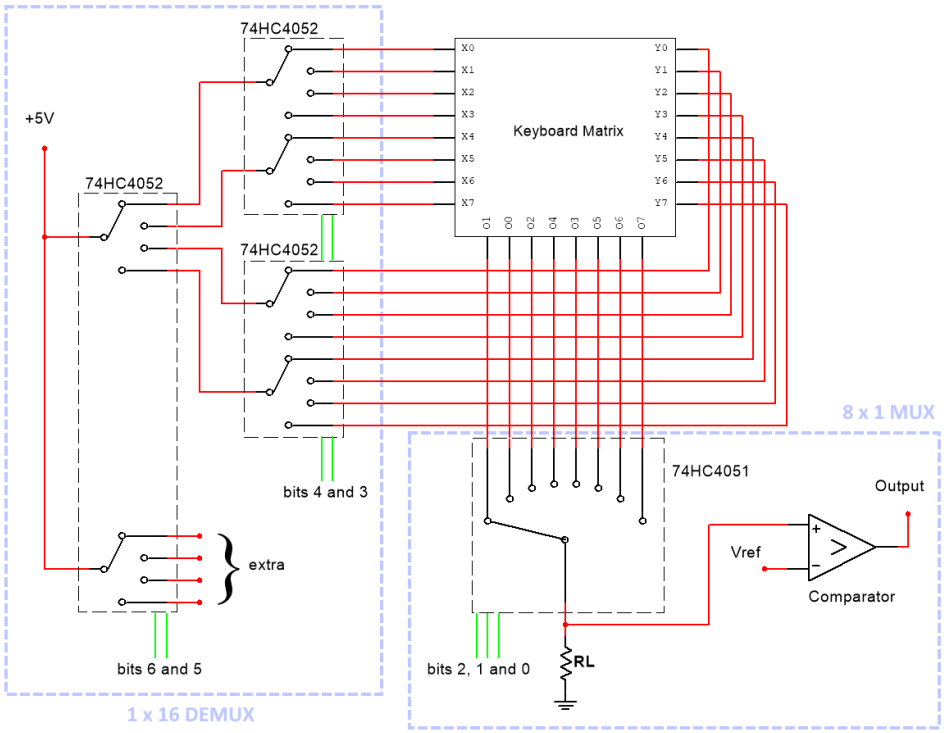
Source: Image provided by author

to reduce the number of pins even more including using serial do parallel converters and shift registers but the proposed solution to reduce the number of pins even further is to design a demultiplexer/multiplexer (DEMUX/MUX) circuit. The proposed circuit can be seen in Fig. 3.3.

The DEMUX/MUX circuit is connected to the MCU via 8 pins. 7 pins for control (bits 6 through 0) and one output pin. In order to read the 16 row pins (8 for the first and 8 for the second electrical contact on each key) a 1:16 DEMUX is used. It is built using two 1:4 DEMUX in series with a ON voltage as the input. This further reduces the need of 16 pins to only 4 pins. With a binary code written to the 4 control pins of the 1:16 DEMUX the equivalent row of the matrix will be turn on. The DEMUX can scan the entire matrix with the configuration of only those 4 bits (bits 6, 5, 4, 3). With a row selected, the 8:1 MUX is used to scan the columns in order to check the status of each electrical contact, using the bits 2 through 0. A load resistor is used to give a good reference to the current coming out of the matrix.

The value of the resistor RL was defined in order to optimize the output voltage's swing. Firs the internal resistances of the matrix board need to be modelled, for measuring it would be more complicated due to non-linearity of the present diodes. Using know values for the RL, 1kΩ and 10kΩ, the output voltage was measured for many keys, when pressed and not pressed. The simulation used can be seen in Fig. 3.4

Figure 3.3 – Keyboard DEMUX/MUX circuit



Source: Image provided by author

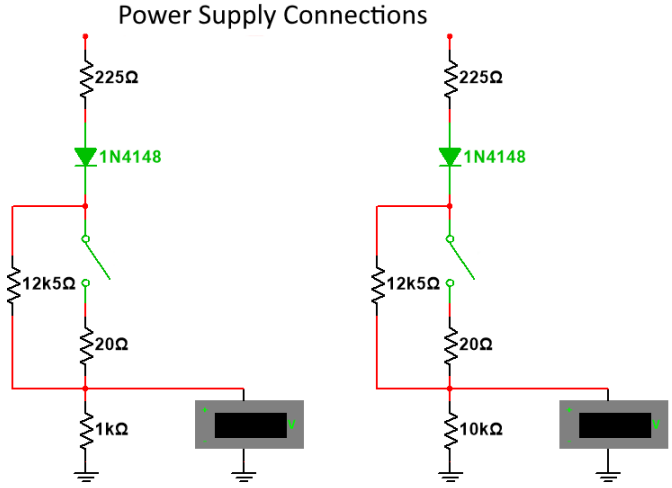
Knowing that the DEMUX internal resistance is approximately 75Ω, all keys will have 3 DEMUXes in the path to the power supply, the equivalent DEMUX resistance is 225Ω. Then there is a 1N4148 fast-switching diode. As both pressed and not pressed tests had a voltage in the RL, it can be deduced that the electrical contacts can be modelled as a switch (or button) with series and parallel equivalent resistances. Using the output measured values it is possible to calculate the approximate value of both resistors (considering the ideal model of the diode) and then using the simulation to determine more precise results.

The approximation was calculated with the following results: For RL = 1kΩ, Von = 3.5V and Voff = 0.3V. For RL = 10kΩ, Von = 4.32V and Voff = 1.94V. Considering an ideal diode, there is only a simple voltage divider to solve considering an ideal diode voltage of 0.55V and the series current being equal to the output voltage divided by RL:

$$R = \frac{5V - 0.6V}{\frac{Vo}{RL}} - (RL + 225\Omega)$$

Taking the average of the results, the resulting resistances can be assumed as a series resistance of 20Ω and a parallel resistance of 12500Ω. Now with this model determined it is possible to proceed to choose an optimal RL value. As the membrane of the

Figure 3.4 – Modelled internal resistances



Source: Image provided by author

keyboard that touches the electrical contact is flexible, the resistance change is not abrupt and ideal but continuous and smooth. In order to make the detection of a key press more of a binary process rather than an analog one. The proposed circuit uses a comparator to help turn the value of the output in more of a binary output. A comparator is a simple component that compares the voltage on the '+' pin to the voltage of the '-' pin. If the voltage on the '+' pin is higher, the output is turned on. If the voltage of the '-' pin is higher then the output is turned off.

The best value for RL is a value that would result in the bigger difference between the output voltage when the key is pressed and the output voltage when the key is released in order to make the comparator output as clean as possible.

The voltage on the output with a released key can be described by the equation:

$$Voff = \frac{5V - 0.6V}{12500 + 225 + RL} * RL$$

The voltage on the output with a pressed key can be described by the equation:

$$Von = \frac{5V - 0.6V}{20 + 225 + RL} * RL$$

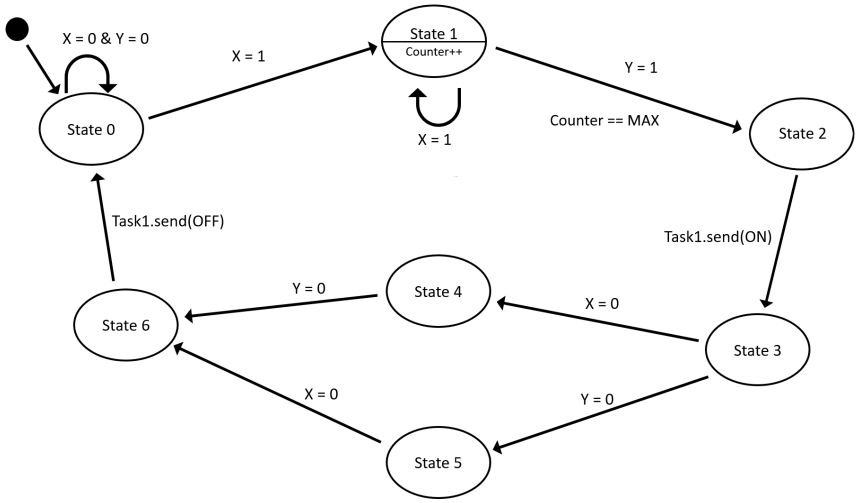
In order to achieve the widest windows between Von and Voff all is needed is to calculate at what value of Rl the derivative of the difference between the two voltages is zero. That resistance is 1767Ω. The chosen RL is 1800Ω for it's a common value in the E12 series.

With the value of RL chosen, all this Key Matrix driver circuit needs is a model of

a comparator. Common operational amplifiers like the LM741 and the LF357 were tested and didn't work in the 5V range of the power supply. The Operational Amplifiers TL072 worked with the supplied voltage but couldn't respond in time with the scan's frequency. The keyboard is being read every 1ms in a very short time frame so general purpose operational amplifiers wouldn't be able to switch on and off so fast. The comparator TL712 was tested and it's performance was flawless, being able to response to every key press in time for detection at the output. This will be the comparator used to read the Key Matrix's output.

Every 1ms the MCU will scan the entire matrix and register each contact status. If the first contact of a key is on, a corresponding timer will start. When the second contact is on, the timer will stop and its value will be sent to the audio processing MCU. When the contacts are off again, it means the player released the key, so an off message will be sent to the audio MCU. The state-machine can be seen in Fig. 3.5

Figure 3.5 – Key press state machine



Source: Image provided by author

3.3 User Interface

In order to keep the interface simple and easy to read and write the project is going to use a simple LCD display to show all the synthesizer parameters, up, down, left and right buttons to navigate, an encoder to set numerical parameters and some sliding potentiometers to control some parameters dynamically. All the buttons will have capacitors in parallel to work as a simple deboucing circuit. The LCD chosen is a 40 by 4 characters in order not only to have a lot of visual space to display the variables and controls but

also to look very nice. The encoder and all four buttons are read via simple GPIO but the encoders use interrupt routines in order to register every small rotation.

As the analog inputs of the UI MCU are susceptible to noise they need some filtering. The method used in this design aims to sacrifice resolution in order to mitigate noise. The first filtering step is to calculate an average between multiple values consecutively. Then this averaged value passes through a hysteresis window. This helps in order to eliminate undesired changes since every change in the value of the analog variables will be sent to the audio MCU and the update of every analog variable, every single cycle can clog the communication bus. After the filtering and the hysteresis window the value of the analog input is then divided in order to be only 8 bits. This helps further improve its resistance against noise but also makes it fit inside the communication packet for packets are usually 8, 16 or 32 bits long.

3.4 UI MCU

As stated before, the UI MCU needs to have a lot of GPIO pins in order to deal with the main keyboard, lots of buttons, encoders, potentiometers and the display. Not only it needs to be capable of reading and writing to all those GPIOs, it needs flexibility in order to be able to have its features changed or expanded. In contrast to the audio MCU, the UI MCU doesn't need to be powerful processing wise, just widely available, cheap and easy to work with. Although the AVR ATMega2560 is not the cheapest microcontroller board available, it was chosen as the UI MCU for it is extremely capable pin wise, extremely available everywhere thanks to the maker and students community and extremely easy to be programmed. The AVR ATMega2560 runs with a AVR RISC 8 bits core at 16MHz clock, 256KB of Flash memory and 8KB of RAM. The microcontroller pays itself for not only expanding the pin capability of the dual microcontroller design but also removing a lot of overhead processing necessary to deal with all the UI functions. For prototyping reasons the project will use a breakout board of the microcontroller called Arduino Mega Pro, which by the way has nothing to do with the Arduino Institution.

3.4.1 Pin count

The AVR ATMega2560, let's call it Mega for now on, has 11 ports with I/O capabilities. 4 ports with all 8 pins accessible: Port A, port B, port C and port L. 2 ports with all 8 pins accessible but in addition to the GPIO capabilities these two ports have analog inputs in their pins: Port F and port K. Port D has 5 pins available to use, two of them are used as UART1 pins. Port E has also 5 pins available but 2 pins are already used by the main UART, UART0. Port G has 4 digital pins available, port H has 6 pins available and port J has 2.

Port A will be used to control and read the main keyboard. Port B will be used to read the UI buttons and encoder. Port C will be used for the LCD data writing. The two UART pins of the port D will be used as a communication bus with the audio MCU. Port E shares its pins with two tasks: The two pins of the UART0 bus used for writing code to the microcontroller and serial debugging and the three control pins of the LCD. Port F will be used to read the 5 potentiometers in the synthesizer panel and also the two inputs of the mod wheel, a joystick like double potentiometer that is located really close to the keyboard in order to be used by the player to control some parameter of the synthesizer while it plays the keyboard. The remaining ports, G, H, J, K and L are not used and could be allocated in the future for expanded capabilities.

3.4.2 Firmware

The code expected to be loaded in the Mega can be divided in two: The main background task and the interrupt routine. The main background routine, called task 1, will be running indefinitely and asynchronously. Task 1 will control the UI inputs reading, the display writing, the UI state machine, all the control variables associated with the DDS system and both the serial communications. The interrupt, called task 0, runs every 1ms and is responsible for two things: A software timer count and the main keyboard readings.

Task 1 will run in an eternal loop with no real-time constraints. In every loop it will first check if any key was completely pressed. According to the keyboard state machine, the task will send to the audio MCU a message containing which key was pressed and with what intensity or which key was released. Then it will read all the inputs. In case the user pressed any valid key (combos are not allowed) it will change any variables if necessary depending on the UI state machine. If the user is navigating the menu then the menu

shown will change accordingly. If the user is changing a control variable's value then the display will show the new value and it will have the variable updated. Task 1 keeps in store all the variables of the DDS machine, all the controls, amplitudes and frequencies. Every time the user changes a parameter task 1 will send its updated version to the audio MCU. When needed, a software delay function will be called using a software timer. The software delay sets the delay variable and waits for it to reach zero. The increment control of the delay variable is done by task 0.

Task 0 will run every 1ms for real-time purposes. It increments the software delay functions in order to save hardware timer resources. It also scans the key matrix in order to detect any change in the status of the electrical contacts. In case of any change, the corresponding key's state machine's state will be changed. This has to occur very fast, and the scan function was calibrated to last an exact amount of time. Too fast and parasitic capacitances of the key matrix wouldn't allow for a clear reading. Too slow and the interrupt would last for too long.

The serial function implemented in the code works as a communication bridge between the two MCUs of the design. It will relay any user command to the audio MCU in order to change the many parameters of the DDS engine. The communication will be unilateral, only the Mega will send messages. The communication will be comprised of two transmitted bytes: One to specify what command it is sending and the other to send the value of the command. In case of the rare 16 bits values, two messages will be sent.

3.5 DDS MCU

The second MCU of the project will work as an audio board: It will receive commands to alter parameters in the DDS and it will alter this parameters while keeping the DDS software running. It will need to be very powerful processing wise, be capable of dealing with good I^2S and UART communication and just as any other parts on this project, widely available and low cost. Although not technically a MCU, the device chosen to be the audio MCU is the ESP32. The ESP32 is a System on a Chip (SoC), which is very similar to a MCU. A SoC can not only integrate processing core, memory and peripherals, but those peripherals are more specialized in certain tasks, as graphics or connectivity. As SoC's are more specialized MCU's, the ESP32 will be still considered a MCU in this work. One of the cheapest MCUs in the market it comes with a staggering 240MHz 32-bit LX6 clock dual core processor, 320 KB RAM and 448 KB ROM.

The best MCU in the market concerning performance by price, the ESP32 can be found literally everywhere, just as the Mega.

It is important to notice that while the AVR MCU runs the code loaded without any layer in between, giving the programmer full access to the hardware, the ESP32 does not. As the ESP32 has a much more complex structure, including two cores, a Real-Time Operational System (RTOS) is used between the programmer's code and the hardware. This extra layer between the code and the hardware is used for two purposes: It allows the processing of many threads in the system and it abstracts much of the hardware via the HAL libraries of the ESP32's framework. The usage of the RTOS, which in this case is the FreeRTOS, brings some benefits as well as some setbacks. As the access to the hardware is not direct anymore, some timings are not as precise. There will be some variation on the average time to trigger the interrupt and this variation can be analyzed as a jitter on the signal from the sync pin. This oscillation has to be taken into account when developing code for the MCU as well when evaluating its performance.

The ESP32 will be responsible for using just two UARTS, one for the program loading and debugging and the other to receive messages from the Mega, and one I^2S bus, to write the audio data to the Codec. The I^2S runs through a DMA as a standard but our design will still process all the audio in a fixed clock interrupt and deliver one data packet at a time.

3.5.1 Dual core operation

As the MCU will only be used as an audio device, our code will "hijack" both cores all the time. Hijack between quotation marks because the ESP uses both cores for many of the standard features as Bluetooth or the Wi-Fi. All the extra functionalities of the MCU will not be turned off on the start up process as the design only wants audio synthesis to be executed. One core, specifically core 0, will run the main function. The main function will start up peripherals, such as the I^2S and UART drivers, and variables, allocate the DDS interrupt in the core 0 and allocate the the control interrupt in the core 1.

3.5.2 Core 1 control task

The core 1 of the MCU will be responsible for two tasks: Receiving and processing messages from the UI MCU and processing the control variables of the DDS sound engine, as the ADSR envelope and the LFO functionalities. Both tasks will be executed in an interrupt triggered every 1ms.

The serial task will process all the commands and change variables and execute DDS commands accordingly. All the control made in the DDS engine is done here: Allocating a voice (audio channel), free a voice, change voice's frequency (pitch) or modulate any of the voices parameters including ADSR, amplitude modulation, frequency modulation, or direct control. Direct control can be made possible by the UI MCU sending the current control value to the audio MCU every time the value changes. This can make possible for the player to control a parameter such as volume or pitch shift with the mod wheel or key press speed.

3.5.3 Core 0 DDS engine

As core 1 will run the control task, core 0 will run the software DDS and nothing more. In every interrupt cycle that triggers the interrupt routine will advance the indexes of the multiple software NCOs (each audio channel will access the LUT independently), modulate the audio channels, mix all of them together, apply the effects that are turned on and send the resulting data to the I^2S DMA. The DMA Hardware Abstraction Layer (HAL) function will work on the transfer of the data to the buffer and from the buffer to the I^2S bus. All changes to the LUTs or any other parameters of the DDS are done in the core 1 task.

3.6 Dual MCU topology

The dual MCU topology was the result of some analysis about how to expand the UI managing capabilities of the ESP32. The ESP32 doesn't have a big amount of pins but it would be capable to run the code that now runs on the UI MCU. The only problem or running the UI code in the ESP32 would be that I/O expanders would be necessary. That would add less available parts to the design, which is a big plus when it comes to

read/write to GPIOs and add latency for I/O expanders usually communicate with the MCU via some serial communication and that takes time proportional to the amount of pins that the system needs expanded. For simpler designs the ESP32 can be used in a single MCU design.

The dual MCU design led to the realization that the synthesizer could be split in two: An interface with keyboard and control commands and a sound synthesizer itself. By this realization it was made very clear that the MIDI standard is a nice and elegant solution in the audio industry, it being a protocol between user interface and sound generators and controls.

The MIDI standard is a technical standard that describes a communications protocol, digital interface, and electrical connectors that connect a wide variety of electronic musical instruments, computers, and related audio devices for playing, editing, and recording music. Although the proposed system could implement MIDI, a custom designed communication protocol will be used, given that not all inputs are standard and fast prototyping is key in order to finish such a big project in time. The custom communication protocol is simple enough to be adapted to MIDI in the future though.

3.7 Interrupt Performance For Both MCUs

Both the Mega and the ESP32 will be programmed with two extra functionalities. A LED will be used in both, for debugging reasons at first and to signal special conditions once the code is done. The second functionality is the performance pins called sync pins. As the interrupts' times are critical to the working of the system, common GPIOs will be used to directly measure the interrupt duration. One pin is going to be used in the Mega and two pins will be used in the ESP32, one for each core. The pin turning on will be the first thing to happen when the interrupt start and the last thing to occur when the interrupt reaches its end. By measuring the length of the pulse out of the sync Pins in contrast to the period of the pulse is easy to get a percentage of core use for the time of the pulse turned on divided by the period of the pulse will result in a value from 0 to 1 proportional to the use of the core. Both the pulses from the Mega's only sync Pin and the ESP32's Core 1 sync Pin will have a period of 1ms. The period of the pulse from the ESP32 core 0 sync Pin will be determined by the audio performance desired in the system.

3.8 Audio Format And Codec

The sampling rate will determine how much time the ESP32's Core 0 will have to execute all the DDS process. In order to give the core as much time as possible, the sampling rate chosen was the lowest one capable of generating 20Hz to 20kHz: 40kHz, according to Nyquist–Shannon sampling theorem. With that sampling rate, the sampling period is 25μs, as is the period of the pulse from the ESP32 core 0 sync Pin.

The chosen codec is the PCM5102A. It is not only cheap but plenty available and supports a lot of different configurations. It supports 16, 24 and 32 bit audio, stereo audio, hardware soft mute, I^2s and left-justified serial communication and the possibility to use 1.8V or 3.3V as its power supply. As the designed system will run on 5V, the codec will have its own voltage converter to generate 3.3V from from the main 5V supply. One really important feature of the PCM5102A is that it performs not only an internal reconstruction low-pass filter but also a anti-aliasing filtering automatically, so filtering the signal using the MCU is not needed.

3.9 Tools

To develop this project Visual Code will be used in conjunction with the PlatformIo extension (PLATFORMIO..., 2014). PlatformIo is an extension that allows Visual Code to not only have all the frameworks and API for embedded systems development but also drivers to communicate and load to the microcontrollers. Both MCUs will be programmed using this IDE/extension combination. All codes related to this project will be written in C. The AVR and LUT generation codes will not require the use of any library. The code for the ESP32 will use its proprietary framework, the ESP-IDF, that uses a version of a Real-Time Operational System called FreeRTOS.

In order to test and evaluate the proper working and output signal of the synthesizer a Rigol DS1054Z oscilloscope will be used alongside the traditional serial port.

4 EXPERIMENTS

This chapter will present the results of the benchmark testings regarding the technical limitations of the proposed design regarding the musical aspects as well the computing aspects.

4.1 Synth As A Music Instrument

The first way to analyze the design is to evaluate if the microcontroller based digital synthesizer is capable of performing the basic functions of its analog counterpart, the original synthesizer.

4.1.1 Basic Synthesizer Functions

As stated before, there are three main basic blocks of a synthesizer: A VCO, a VCF and a VCA. The VCO is just the main DDS engine itself, it generates arbitrary waves at frequencies, with a dynamic control over the frequency generated. The allocation and control of the generated voices is done via functions executed in the core 1 task of the audio MCU. The VCF in the digital synthesizer can be a digital filter applied to the output of the mixer, as any other digital audio effect. For testing, a low-pass filter with a controlled cutoff frequency will be used. Lastly the VCA is basically a multiplier. Together with the mixer in the end (that sums all the values) , they both operate a sum and multiply mixer. All voices are accumulated in a sum variable. Then this sum variable is multiplied by the gain variable. With this simple code we can mix all the voices and control the gain over the master channel, the output of the DDS. In order to fulfil its role as a music instrument the digital synth should be able to perform the functions of a VCO, a VCF (or any other effect) and a VCA, with dynamic control over their parameters.

4.1.2 Basic Synthesizer Controls

After the basic functionalities have been established, the controls of each basic synthesizer block should be tested dynamically using a user control input as control variable. The mod wheel installed in basically all synthesizers is the perfect tool for that: A

two-dimension potentiometer, which the value can be sent to the audio MCU in real-time in order to test if the control of any parameter is working properly.

4.1.3 Synthesizing Methods

By definition, a DDS system will always be performing Wavetable Synthesis, for all waves start in a LUT, which is a wavetable. Although the memory based synthesis is the method by which a DDS system works, the system can also emulate some analog functionalities. Although not implemented in this system, sample-based synthesis could be implemented easily for all the structure would be the same, save from the LUT, would store a much longer audio signal. Additive synthesis is actually implemented in the system as the DDS system implemented allows the user to generate multiple voices. For the user to generate sound using additive synthesis the only thing that is needed is to configure the many extra voices to be multiple of the first, with their amplitudes configured as well. In this way additive synthesis is used, although digitally.

Frequency modulation synthesis is the most complicated method of the four mentioned. Although basic frequency modulation can be done with this design, controlling the frequency generated by a audio channel with the amplitude of a LFO, this design will not be capable of generating FM Synthesis.

4.1.4 The music it makes

The last aspect that will be analyzed from the musical perspective is the sound it generates. Although the audio signals can be and were analyzed in the oscilloscope, listening to the resulting sound is of great importance. First there is the function of the project that is to generate sound for humans to listen to. Second there is the need of evaluation of the transient aspect of the audio signals. It is not easy to analyze audio signals' transients in the oscilloscope for the events can be hard to be triggered. The human ear can pick on very small details of the sound generated that the oscilloscope analysis would have missed.

4.2 Synth As A Computer

The Second way to analyze the design is to evaluate the project as a computer, more specifically, an embedded system. The technical tests will aim to analyze the processing performance of the DDS engine, how many different channels can it run before it overloads the core and how many effects can be applied to the output audio signal.

4.2.1 LUT generation and access

The first piece of code to be written is not any code to be loaded into any MCU. The generation of the look-up tables is the first step in order to build a basic DDS engine. A small code was written to generate the base LUTs' data, the 2000 points in the array that will be used to generate the audio signals. A different base LUT will be used for each different wave shape. When the user selects a shape to be synthesized, the data from the base LUT selected will be transferred to the LUT used by the DDS system. Each channel (right and left) will have its own LUT.

4.2.2 Interrupt routine time limit

The core of the project is the audio MCU performing the processing intensive task of running the DDS engine. The core 1 task will always be underloaded for the ESP32 is a very powerful MCU and 1ms is many times the time needed for it to run its code. The core 0 task is where the processing bottleneck will manifest as system errors when the interrupt lasts more than it could.

This limit is a fixed time measure, of only 25μs, that will be shared between running voices and applying effects. Both of those functions need to be executed inside the interrupt in order for the DDS application to work. Observe that the less voices are allocated, more time is left for the effects to be processed. The opposite is also true. The less effects are turned on, the more time is left for the DDS engine to run all the voice channels and to mix them to the output value.

Special attention was paid to the ESP core 0 task for it was the piece of code that needed optimization the most. In contrast to the other functions and tasks, every change in the core 0 task is done with performance in mind. The ESP32 is not capable of processing

float-type variables in an interrupt, as they are not very efficient. Fixed-point variables will be used and operated manually. As fixed point operations are done in the DDS engine in order to calculate the phase accumulator next value, multiplication and division would be normally used. In order to save mere microseconds the fixed point math is going to use shift right for division and shift left for the multiplications. This way the operations will use less processing power. The only multiplication is done after all the channels are mixed, in order to control the volume. This reduces the possibility of controlling each channel's volume independently but increases a lot the overall performance by reducing the amount of instructions to be executed each cycle. Every task of changing a control value of the DDS engine is done outside the interrupt, including the allocation and freeing of each audio channel. Finally GCC, the compiler, is called with level 2 optimization enabled. Without it this prototype would not be possible.

The performance of the prototype will be tested based on how many independent audio channels can be processed by the audio MCU without any errors or system failures. First the number of audio channels will be increased until its maximum value. The desired value of active audio channels is in this project the number of keys in the main input, in this case 61 keys. If the maximum number of active audio channels is bigger than 61, it will be set to 61 and then one effect will be turned on and its effect in the Core 0 performance will be measured, the extra time in the interrupt being proportional to the processing needed for the effects to be calculated and saved.

Both tests can be analyzed using the sync pin associated with the core to be tested. It will generate a pulse with the exact duration of the time the interrupt routine was executing. Keeping in mind that the pulse can never reach a duty-cycle (time on ON position) of 100%, the length of the pulse will be monitored on the oscilloscope and serial ports will be read for error logs.

As the interrupt routine time to execute can oscillate due to many conditional events within and to the RTOS-induced relative inaccuracy, a safety margin of 10% will be applied, making the maximum time of execution for the interrupt routine to be 22.5μs in the worst case.

4.2.3 Voice limit

The first parameter to analyze is the maximum number of voices that the system is capable of generating properly. This can be determined by the sync pin associated with

the core to be tested. The test will consist on gradually increasing the number of voices allocated on the DDS and registering the duty-cycle of the pulse on the corresponding sync pin. The increase will stop as the MCU starts reporting errors due to overflow.

4.2.4 Post-mixer effects

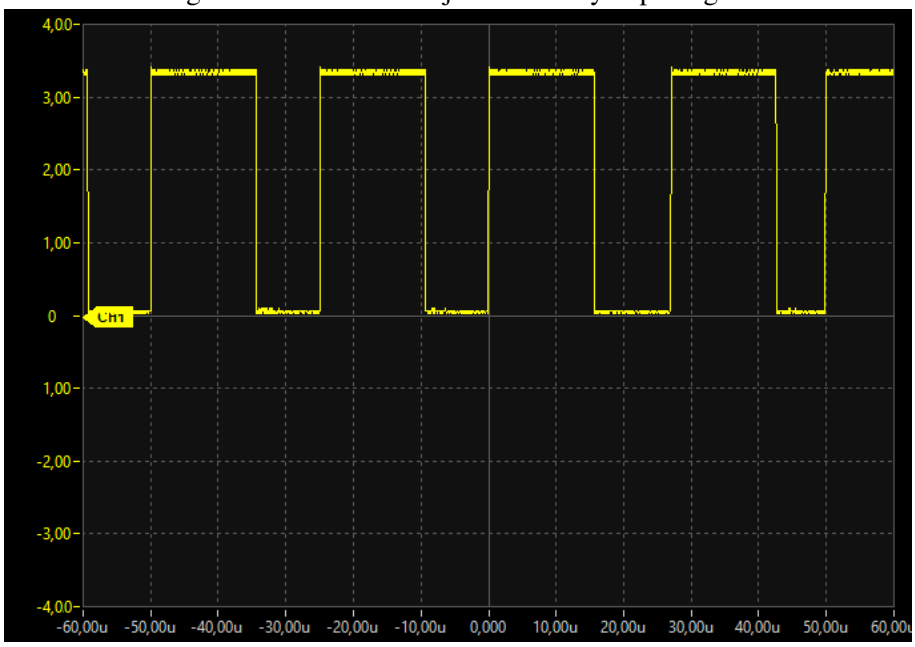
The second parameter to analyze is the amount and quality of the effects applied on the output of the DDS mixer. As effects are more of a case of creativity, a basic low-pass filter will be applied and the impact of it's execution will be analysed by the time impact in the duty-cycle of the pulse on the corresponding sync pin.

5 RESULTS

5.1 System Performance

It's important to notice that the already mentioned RTOS-resulting variation of interrupt trigger time will be noticeable in the analysis of the sync pin signal but not noticeable in the sound frequency range. It is so because this oscillation in the interrupt event clock is of only 2.12μs as it can be seen in Fig. 5.1. That is a variation too small for the human ear to notice, but big enough to be a problem when developing the interrupt routine. The only variation that will be perceived by the player is the aliasing associated with the DDS architecture (previously discussed).

Figure 5.1 – Noticeable jitter in the sync pin signal

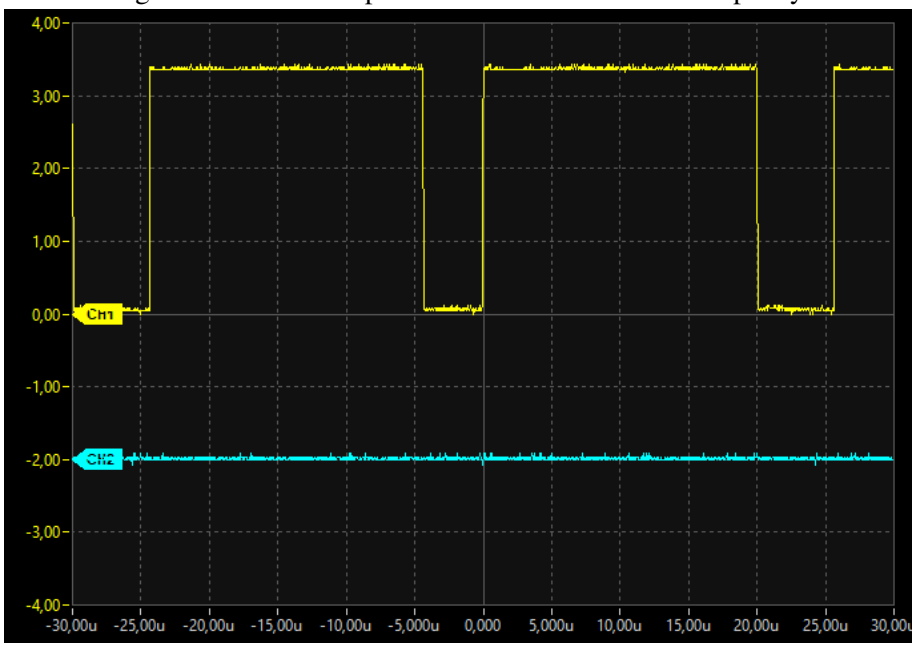


Source: Image provided by author

As for the performance, the proposed design performed really well. The maximum number of audio channels processable in the DDS engine was of 80 voices. Far more than the number of keys in the main input. With this number of audio channels the Core 0 interrupt achieved the limit 22.5μs of processing time (worst case), as it can be seen in Fig. 5.2.

After the capability of the design to have one audio channel per key, it was time to evaluate the performance with 61 audio channels and the effects of running a simple low-pass filter algorithm in the main output before sending its value to the codec. The period of processing time of the Core 0 interrupt was of approximately 16μs (Fig. 5.3),

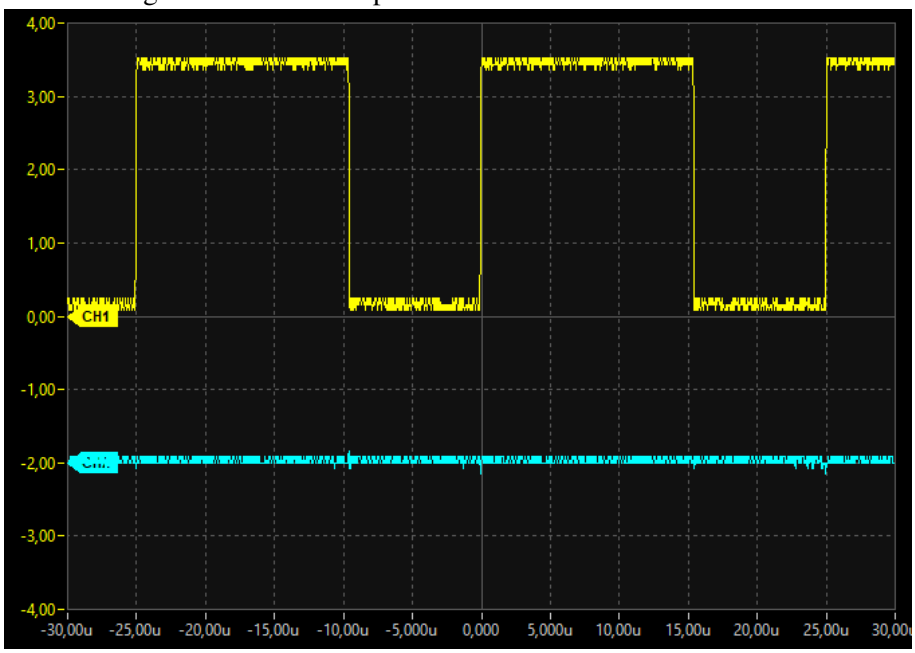
Figure 5.2 – Core 0's performance at its maximum capacity



Source: Image provided by author

which left a lot of room to work with before reaching the 22.5μs safety limit.

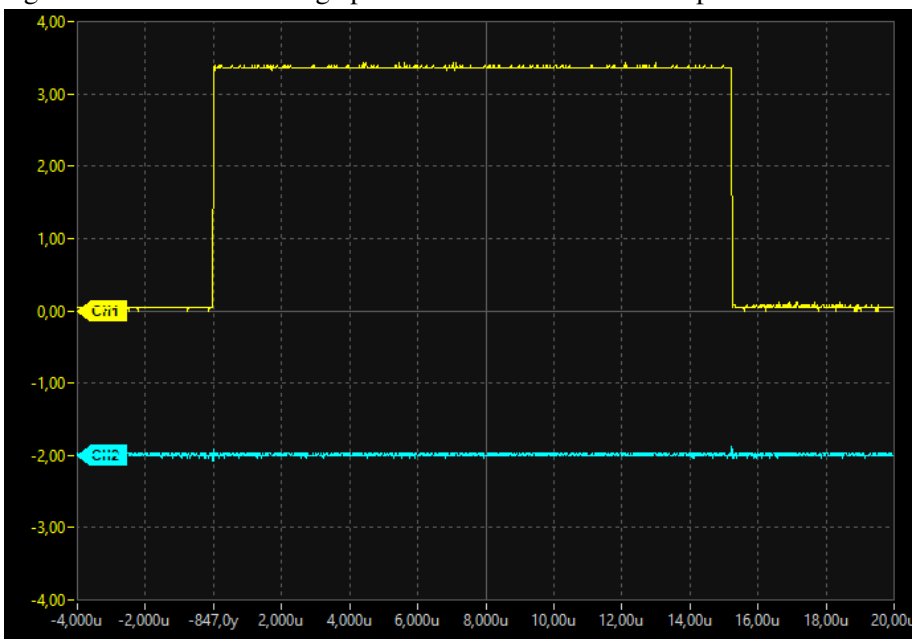
Figure 5.3 – Core 0's performance with 61 audio channels



Source: Image provided by author

In order to measure the performance of the DDS engine with and without the filter applied a more precise measure would be needed. This measurements were done using the oscilloscope functions but unfortunately were not visible via the oscilloscope PC software that generates the images used in this work. The performance of the DDS system without the filter was of 15.24μs (Fig. 5.4) on average with a worst case peak of 17.36μs.

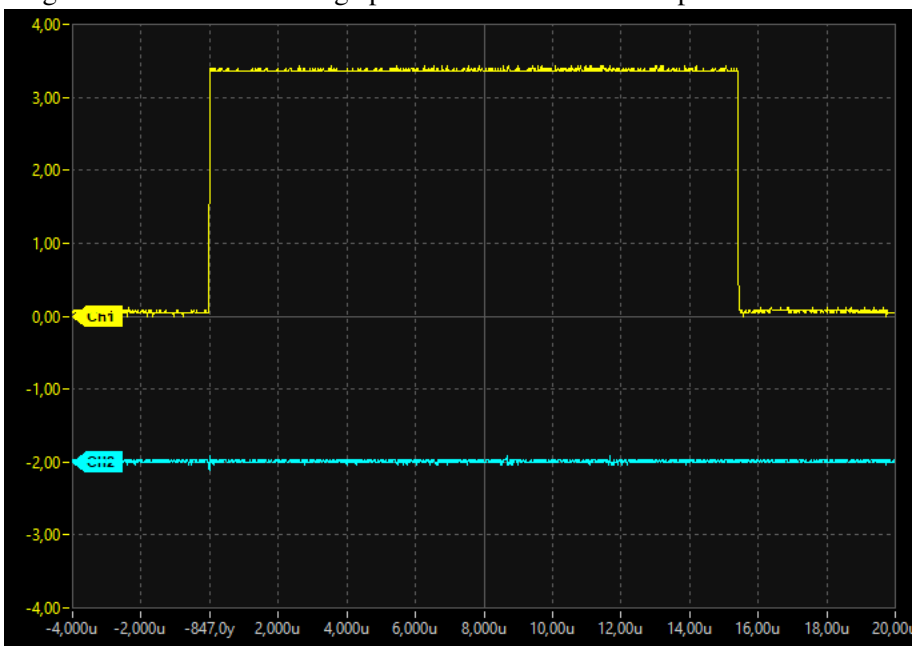
Figure 5.4 – Core 0's average performance without the low-pass software filter



Source: Image provided by author

With the filter on, the average length of the interrupt was of 15.40μs (Fig. 5.5) and the worst case registered was of 17.52μs. This results show a difference of 160ns of extra processing time resulted of the filter algorithm.

Figure 5.5 – Core 0's average performance with the low-pass software filter

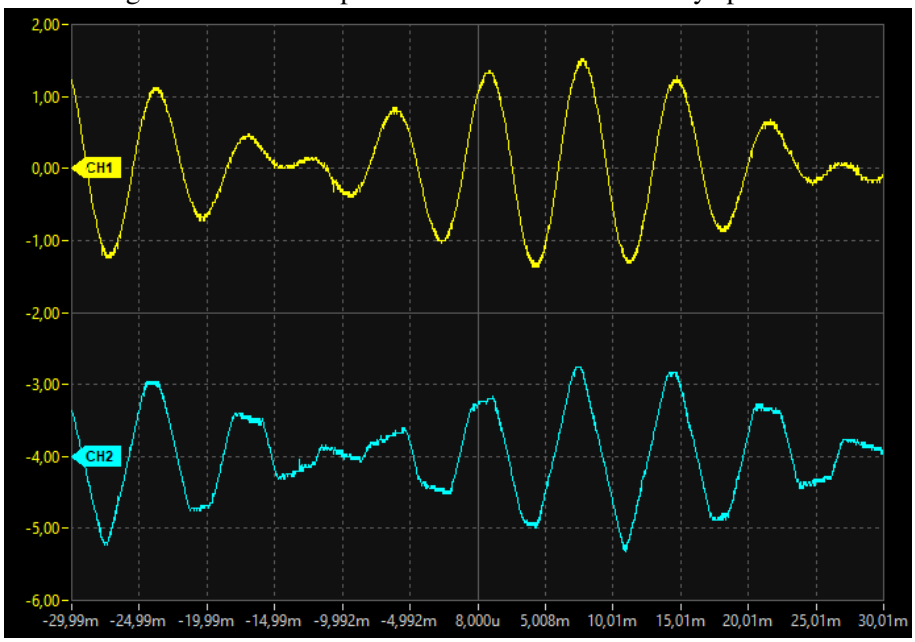


Source: Image provided by author

5.2 Instrument Performance

The Synthesizer sounds exactly as expected: It sounds like a discrete frequency digital signal generator. In the lower frequencies the difference between the design and the analog counterpart is negligible. In the higher frequencies the quantization error (aliasing) begins to be more noticeable. In the image below (Fig. 5.6) the signal generated when the player presses the C3 and D3# (130.8Hz and 155.6Hz) in the right (above) and left (below) channel. The right channel is using a sinusoidal wave LUT and the left channel is using a triangular wave LUT.

Figure 5.6 – The output of the channel with two keys pressed



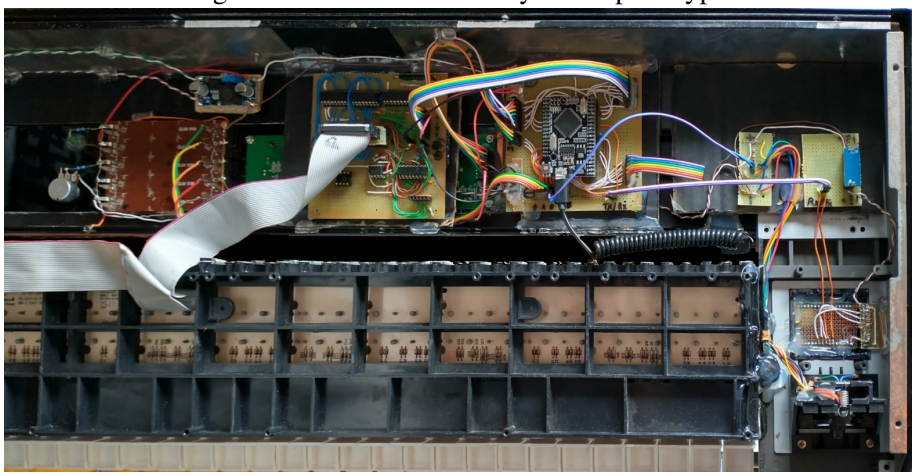
Source: Image provided by author

Below in the Fig. 5.7 a picture of the finished prototype developed along this work.

Figure 5.7 – The final version of the prototype

Source: Image provided by author

Figure 5.8 – Internal circuitry of the prototype



Source: Image provided by author

6 CONCLUSION

Microcontrollers came a long way since their origin, back in the 1970's. Not only a vital part in many electronic circuits nowadays but also a device capable of high processing capabilities for a very low price. With modern microcontrollers it is possible to execute very CPU and memory intensive tasks without increasing too much the total cost of a project.

With the popularization of electronics and embedded systems in general alongside the huge increase in microcontrollers' processing power it was never so easy to design and build an electronic music instrument that generates good quality sounds with a low budget and very accessible parts and tools.

The results of the performance tests of the proposed design confirms that modern and cheap microcontrollers are more than capable of handling a huge amount of work compared with the devices used in the first commercial synthesizers. Able to process a total of 80 audio channels, or 61 audio channels with a lot of room to implement many more audio effects, the proposed synth can clearly perform as an entry level musical instrument. Thanks to the popularization of entry-level microcontrollers it is extremely accessible to anyone with the required basic knowledge to design, develop and build a microcontroller based digital synthesizer with affordable and easy to acquire parts.

7 FUTURE WORK

Of course the first thing to be done in order to expand the current code is to add more waveforms and effects. As this is just a work of creativity and knowledge in music it will take some research time to discover what would this waveforms be and what would the new effects do to the audio signal. The second thing to do would be to optimize the Core 0's task even further. Maybe optimizing the LUT's size, mixing or even the effects algorithms, in order to use the core more efficiently and to allow the allocation of many more audio channels.

Something that would be easily implemented in the current code is sample-based synthesis. The only thing needed is to extend the LUT length and adjust the code to deal with single cycle or sample reading mode. The first is the current mode of the DDS, that reads a waveform and sends it to the DAC. The later would read multiple periods of the recorded wave.

As it is right now, the prototype uses a female TRS (P2 3mm) connector directly connected to the output of the codec. Adding loudspeakers alongside with a more robust power supply and amplifiers would make for a really nice upgrade as for now the system needs an external amplifier or just headphones, that were actually used in the project.

As described before, the development of the project led to the finding that communicating the UI with the sound engine via a protocol was a nice design choice and made clear why things like the MIDI standard exists. Maybe breaking the prototype into two MIDI adapted devices would be an interesting project in itself.

One improvement that would not actually fit in this project would be to fully use the advantages of a DMA system. As specified before, The ESP32 has a DMA and the DMA is mandatory when using the I^2S HAL but the audio signal was still generated only each sampling period. One way to make a more powerful and versatile design was to not process the LUT and effects in a fixed sample rate but to fill the DMA buffer all at once, read the users inputs, and wait to fill the buffer again. Then you have a bigger windows to process extra voices and effects without the constrains of the 25μs (or other sampling period) keeping the system limited processing wise.

A more powerful MCU could be used as an audio MCU instead of an ESP32. The ARM Cortex-M family of MCU's is well know for being extremely capable architecture, with many subsets of devices, from low-power/small form factor to some very powerful and capable chips. The Teensy boards are one of the most recognized boards in the maker

community. The Teensy 4.1 carries an ARM Cortex-M7 running at amazing 600MHz of clock. This board surely is the future of this project, as a more deluxe, not so low-cost version of the current design.

Finally something that would be considered "advanced" is phase-dithering. Sampling and quantization can lead to quantization noise. This noise will be distributed at the whole frequency domain. There are many techniques that could be used in order to reduce the quantization noise but one of the simplest techniques is phase-dithering. I consists of adding a small noise factor to the phase accumulator. It was not considered into the project for quantization-noise reducing techniques are used when the system is aimed at high fidelity systems, usually with much higher frequencies.

REFERENCES

ALEXANDER, Charles K.; SADIKU, Matthew N. O. **Fundamentos de Circuitos Elétricos**. Fifth. [*S. l.*]: McGraw-Hill, 2013.

ANALOG DEVICES. **Fundamentals of Direct Digital Synthesis (DDS)**. [*S. l.*], 2009.

GENTILE, Ken; CUSHING, Rick. **A Technical Tutorial on Digital Signal Synthesis**. [*S. l.*]: Analog Devices, 1999.

HARTMANN, William M. **Principles of Musical Acoustics**. [*S. l.*]: Springer, 2013.

HOROWITZ, Paul; HILL, Windield. **The Art Of Electronics**. Third. [*S. l.*]: Cambridge University Press, 2015.

KLEIN, Barry. **Electronic Music Circuits**. [*S. l.*]: Sams Technical Publishing, 1982. v. 21833.

MILLER, Bob. **DeepSynth**. [*S. l.: s. n.*], 2018. https://hackaday.io/project/161481-deep-synth. [Online; acessed in April 28th, 2022].

MOORER, James. The Lucasfilm Audio Signal Processor. *In*. v. 6, p. 85–88. DOI: 10.1109/ICASSP.1982.1171526.

MOORER, James A. **Deep Synth**. [*S. l.: s. n.*], 2022. http://www.jamminpower.org/index.html. [Online; acessed in April 28th, 2022].

PLATFORMIO. [*S. l.: s. n.*], 2014. https://platformio.org/. [Online; acessed in April 28th, 2022].

PUCKETTE, Miller. **The Theory and Techniques of Electronic Music**. [*S. l.*]: World Scientific Publishing Company, 2007.

THX. **THX Deep Note**. [*S. l.: s. n.*], 2022. https://www.thx.com/deepnote. [Online; acessed in April 28th, 2022].

VANKKA, Jouko; HALONEN, Kari AI; HALONEN, Kari. **Direct Digital Synthesizers: Theory, Design and Applications**. [*S. l.*]: Springer Science & Business Media, 2001. v. 614.