Unraveling the Deep Note: THX's signature sound

RODRIGO DALTOÉ MADRUGA, Brazil, 2020.

This survey analyses the basic structure of the Deep Note, the widely known sound that is THX's signature trademark, and then suggests how it could be implemented in a simple, commercially available, low cost microcontroller board.

---

1 INTRODUCTION

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), a mainframe computer in Lucasfilm Computer Division. 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. [1]

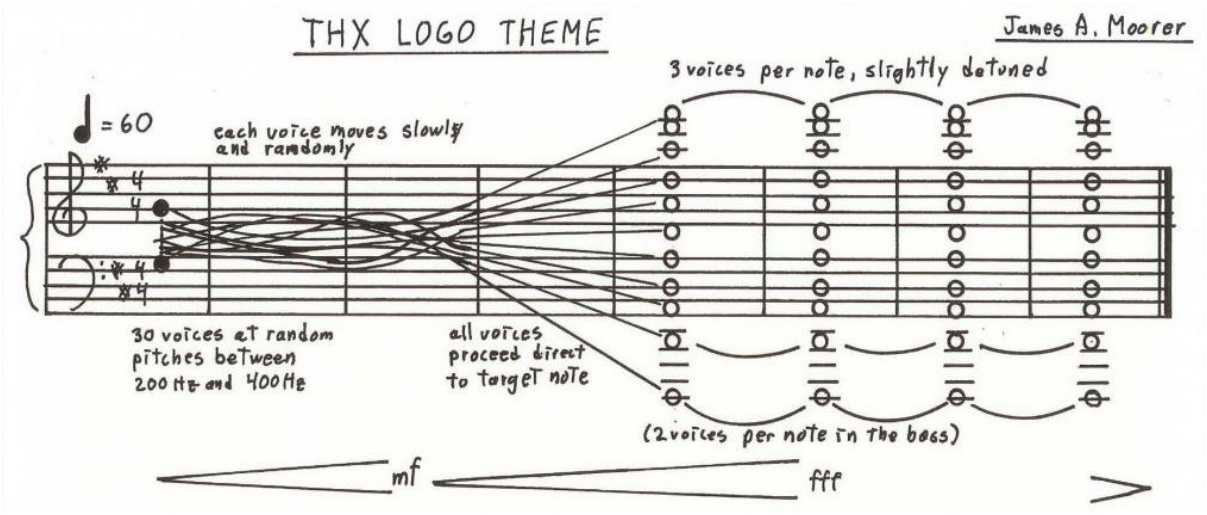
To understand the computer implementation, purely via software or via hardware, someone trying to recreate the Note needs to understand a few basic concepts about music itself, more specifically, music theory. One should be able to answer, "what is a note?", "how does a note translate to a sound frequency?" and "what is an octave?" with no problems, as these concepts will be fundamental in the process of creating sound. [2]

With music concepts in mind the process of digital sound synthesis comes next. Electric transducers will transform electric signals in sound, signals that were generated by a digital, analog or mixed source. Routines will be programmed in order to build a sequence of notes through time, and these notes will be summed and amplified so in the end the result is the 30 voices going on a glissando, which is known as the Deep Note [3].

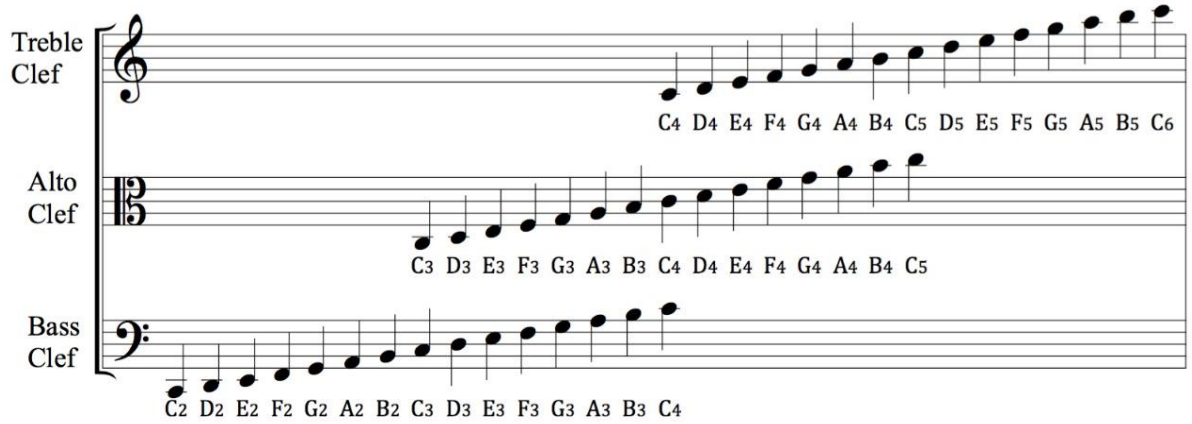
Special attention should be paid to the digital/analog aspect of the project. The easiest way to design a system that sequences musical notes is to use a programmable device. In the program, the sequence will be simply stored and edited. Every programmable device is inherently digital so, how can it generate sound, electric analog signals that generate vibrations when applied to a transducer? The answer is a digital-to-analog converter (DAC). Without those, our bits and bytes would not turn in the sound, image or every inherently analog media we know. With the use of the many methods available, and accordingly DAC parameters, it is possible to generate an infinite number of forms of electric signals. The quality of the sound generated is directly associated with the shape of the wave the synthesizer system generates.

2 MUSIC

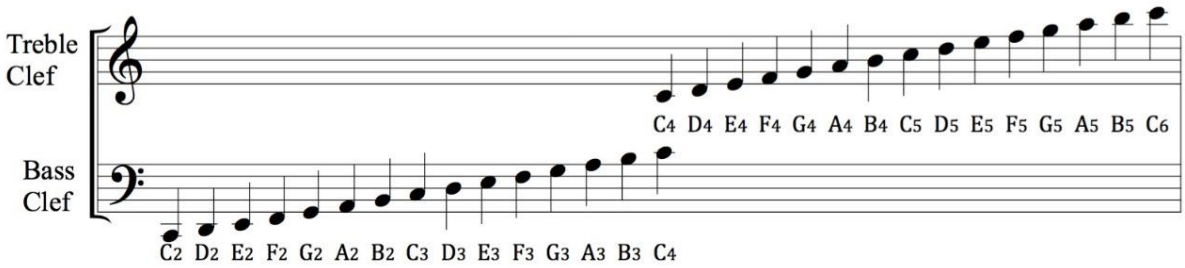
With the concepts of frequency of sound, note and octave in mind we can start to grasp what the structure of the Deep Note is. The Note begins with a mass of 30 voices varying randomly inside an envelope from 200Hz to 400Hz (G3 to G4). While they keep varying, they go from very silent to *mf* (mezzo forte). After a certain amount of time a second crescendo begins, that goes from very silent to fff (fortissimissimo), and a glissando of the voices also begins, transitioning from the random frequency they were executing to a final determined note. The three lower notes (in frequency) are made from two voices each slightly detuned from each other. The other notes are made from three voices, just as detuned as the bass ones. The lowest note is a D1 (36.7Hz) and the highest one is a F6 (1397Hz). THX released the original scheme drawn by A. Moorer when creating the sound and it is the main tool to help one to recreate the Note. [4]



To help us understand easily the music sheet we can make use of a standard clef table used by beginner piano players.

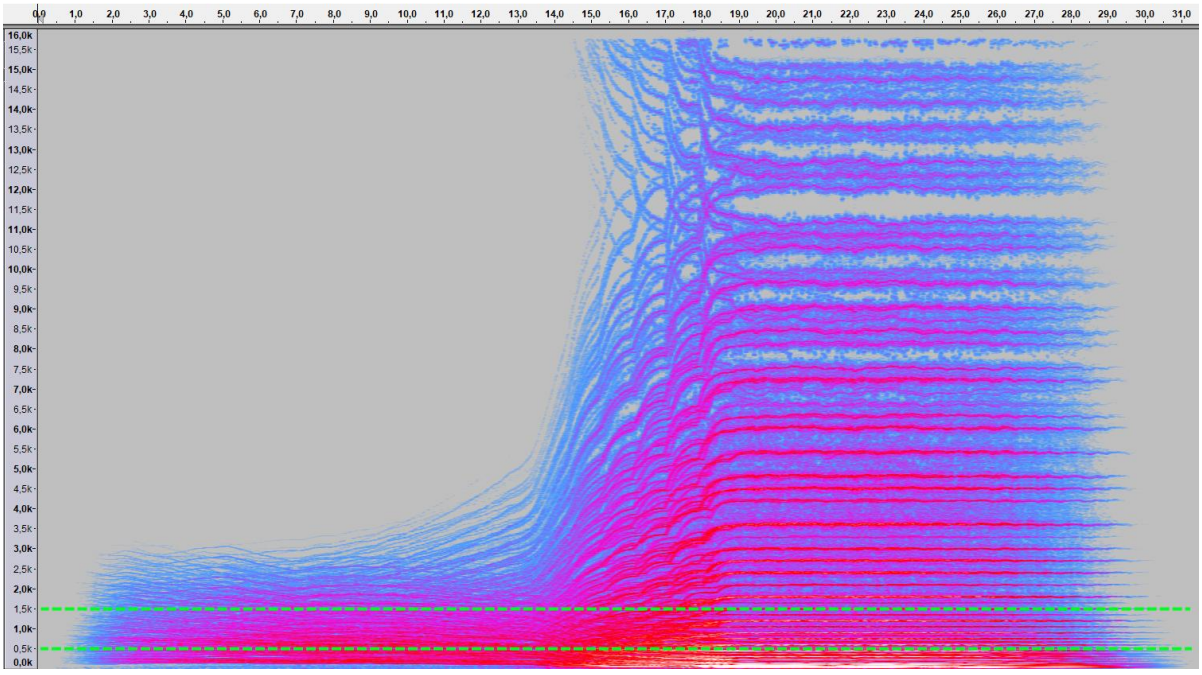


Now we are able to reconstruct via software the Deep Note by following its music sheet and, if you don't know all the notes by heart, consult the clef table to verify which note is which in the sheet. Notice that the Deep Note's music sheet has two clefs, Treble and Bass, so you can ignore the Alto clef. A smaller version of the table would look like this:



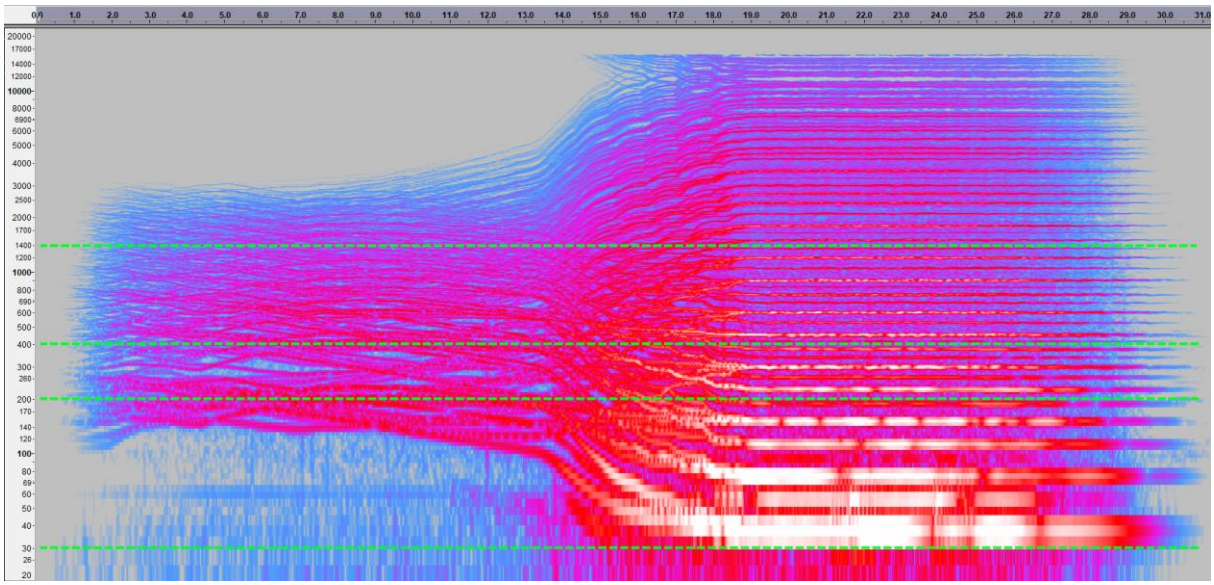
Notes and chords are a human notion, concepts created to help us construct music in a discrete and finite (and easier) way. In nature, sound has only two properties, amplitude and frequency (three if you count phase and consider by ‘sounds’ only pure sinusoids, but we are going to ignore that for now). The amplitude of each note in the Note is the same, with slight variations that for this project are not important. The frequency though varies a lot. The final arrangement goes like this, from top to bottom: F6 (1397Hz), D6 (1175Hz), A5 (880Hz), D5 (587Hz), A4 (440Hz), D4 (294Hz), A3 (220Hz), D3 (146.8Hz), A2 (110Hz), D2 (73.4Hz) and finally D1 (36.7Hz).

A very important detail is that each note is generated by an oscillator that is not only generating a pure frequency but many harmonics. Each note is played by a sawtooth shaped wave generator. The shape of the wave changes its timbre, because it changes the harmonics of the note. It gets easier to visualize the concept that the notes are in fact multiple frequencies when you observe the spectrogram of the Deep Note.

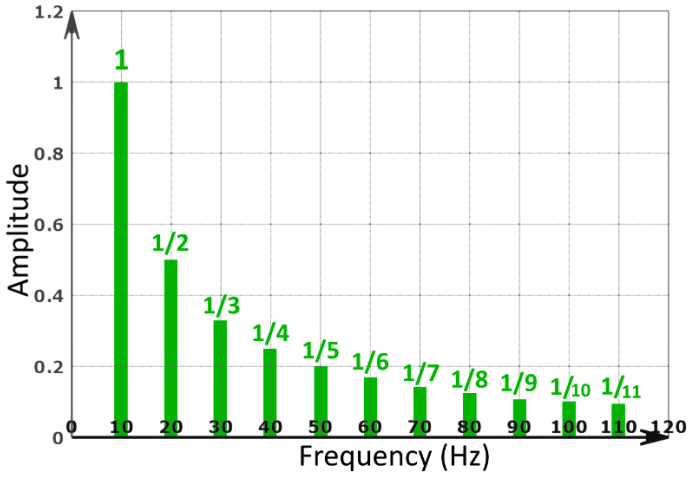


By observing the spectrogram, it is easy to visualize that the notes originally generated are not pure frequencies. The green lines mark 500Hz and 1500Hz. Remember that the highest frequency achieved by a Note is F6, ~1400HZ, and that before the glissando all notes were contained inside a 200Hz-400Hz envelope. By the spectrogram is easy to observe frequencies up to 3kHz before the glissando. After the

second crescendo is possible to observe frequencies as high as the 16kHz limit of the mp3 format of the wave analyzed. Let us observe a log spectrogram to better visualize the lower frequencies.



It is now even easier to notice that there are more frequencies being generated than the pure frequencies that we could erroneously assume are symbolized in the music chart (The notes in the charts are meant to be played by instruments that generally have many harmonics in their timbre). That is because of harmonics. The sawtooth wave is composed by its main frequency (fundamental) and a sum of many harmonics, waves that have a multiple of the fundamental frequency. Each wave shape has its own harmonics. We are going to analyze the sawtooth wave harmonics as they were in the original synth that generated the Deep Note.



Harmonic frequencies in a sawtooth wave
with a fundamental frequency of 10Hz

As seen by the spectrum analysis of the sawtooth wave, there is a fundamental frequency with amplitude of 1, and the subsequent harmonics with the gain 1/n, with n being the harmonics number. Now we know why the spectrogram looks so ‘polluted’ compared with the music sheet. But the ‘pollution’” is only visual. This harmonics are

what make the notes have a distinct sharp timbre, with audible characteristics resembling a violin.

With all these notions now, it is possible to design an algorithm and then write a code that would follow all these timings and notes. It is important to the resulting sound that the notes are generated as sawtooth waves, so the naturally occurring harmonics are going to fill the rest of the spectrogram and, after amplifying the output, the air.

3 MICROCONTROLLER

A microcontroller is a really small and humble computer with many peripherals inside. Generally, it has a less powerful CPU, less memory and less electric power available than a consumer computer. The main advantage of this kind of system is that it not only allows the programmer to access the hardware directly but also to control the timing of each access.

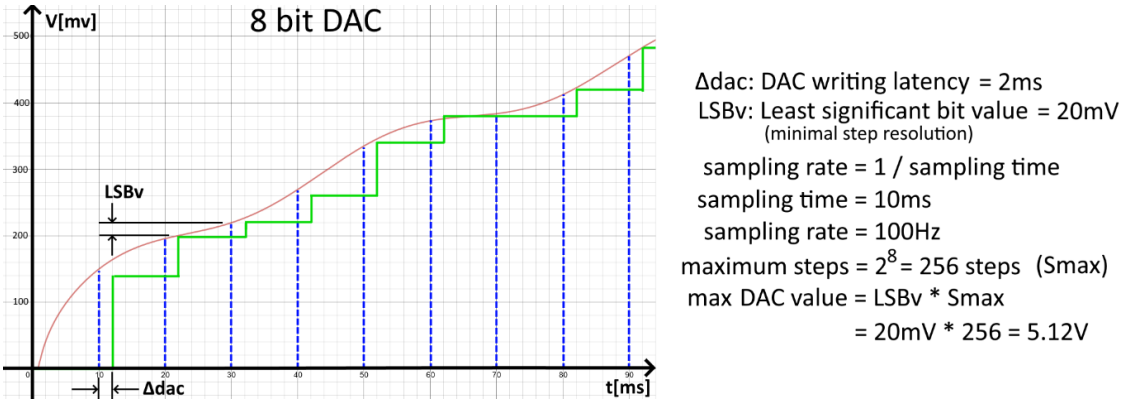
Writing a code for a microcontroller is very different from writing a code for a general-purpose computer. The key here is the hardware power and access. Instead of sending asynchronous bytes to the operational system (OS) that lies beneath the code in a PC, here it is the responsibility of the programmer to make sure every command (it can be note or chord) reaches the output (could be a sound amplifier) in the right time.

Interestingly enough, when programming a more demanding task on a microcontroller, it is common to code (or to import) an embedded operational system. An embedded OS is a specific type of OS that is made for embedded devices. Embedded devices are small electronic circuits, usually programmable, that are meant to be more power efficient, mobile and designed for specific task rather than being general purpose.

4 DIGITAL TO ANALOG

The most important part of a synthesizer is what it does in the output. In this case the synth should play the Deep Note so it must be capable of playing the 30 voices with strict timing in order to not change their frequency and execute the frequency shifts correctly so to execute the many glissandos simultaneously. All that could be done with a DAC. Keep in mind that there are other ways of generating multiple voices and mixing them to an output but a most simplistic approach would be to use a DAC. A DAC would transform an integer value into an analog value. It should be noted that the signal that comes out of it is shaped by the succession of values that are written in the DAC. So in order to output a sine wave, the DAC should receive, by the program that is written in the microcontroller, all the values of a sine throughout the whole operation. Each time the DAC receives a new value, it updates the output. The frequency in which each sample is written in the DAC is called sampling rate (SR). The higher the SR, the higher is the frequency that a generated signal can have. The integer value that is written in the DAC is stored in one or more bytes. The amount of bits used to store the value is called bit-depth. The more you increase the bit-depth the bigger the resolution of your signal is, the more complex you can get in your DAC

output, the less aliasing you observe in the wave generated. A good example of how DAC sampling works is found in the picture:



The curve in red is the ideal analog signal the DAC tries to mimic through the algorithm inside the microcontroller. Each blue timestamp an interruption occurs and calculate a discrete value that approximates the red ideal curve and then it goes to write the final value in the DAC. The ∆dac is the latency it takes from calculating the value to actually writing it in the output. The green curve is the actual output from the DAC. It differs from the ideal curve because of the two main factors discussed before, the bit-depth and the sampling frequency, plus the DAC writing latency. Improving the bit-depth consists of increasing the amount of bits of the DAC's output, and therefore reducing the vertical aliasing. Improving the sampling resolution thought is not that easy. The minimal sampling time is the DAC latency itself. To reduce the horizontal aliasing means not only reducing the time you decide to wait for the next sampling but also reducing the DAC latency which involves not only hardware choice but code efficiency. Reducing both vertical and horizontal aliasing means bringing the output of our synthesizer (green curve) closer and closer to the ideal theoretical signal (red curve) as it is going to be discussed next.

The DAC's bit-depth is usually defined by the manufacturer of the hardware but there is the option of creating a discrete DAC, external to the microcontroller. Usually though the internal DAC does the job well. For example, the ESP-32 [5] microcontroller board has an 8 bit DAC. The Teensy 3.5 [6] board has a 12 bit DAC. We want as many bits as possible in our DAC, but if it happened that we chose to work with the ESP32, designing our own discrete DAC circuit should be simple and efficient. The sampling rate in contrast to the bit-depth is totally under the programmer's control. The only limitation is the maximum sampling rate (SRmax) that could be achieved with each board. The SRmax is influenced by the performance of the CPU of the microcontroller and how fast the CPU can write in the DAC. The performance of the CPU consists usually of the clock with which the CPU runs, the internal architecture, which defines how the CPU actually uses the clock to process data and, just as important, the size of the block of code that runs every time a sample is written in the DAC. This time between sampling is defined by the programmer and in the end an interruption is triggered. The interruption will execute the smallest piece of code possible, just in order to calculate and update the output through the DAC. In modern microcontroller architectures, it is already standard to execute one assembly instruction each clock cycle. Therefore, in the end, the peak performance a modern microcontroller board

based synthesizer will have is defined by the core base clock, the size of the block of code that runs each interruption and the output latency. This performance analysis is crucial to the project, as choosing a weaker board will render the synthesizer unable to execute the necessary code in time.

5 THE OUTPUT TOPOLOGY

Going back to the Deep Note now that the main concepts of music, hardware and software were covered, let us recapitulate. We need to get 30 sawtooth voices, ranging from 1400Hz to 36Hz, to the DAC, as fast as possible. Not only that, but we need to shift the frequency of these 30 voices constantly, and make them sound undistorted in the process. Due to hardware constraints, there is the possibility of reducing the scale of the project. Making each note a single voice instead of two or three. That would reduce the number of voices from 30 to 11. Alternatively, the project designer could use multiple PWM pins instead of one DAC. There is even the possibility to use external devices, peripheral boards, to generate the individual voices or all of them. As microcontroller boards of higher performance are getting more and more common in the last five years, the ‘Occam’s Razor’ approach is usually the best. With many low cost boards going beyond the 100MHz clock, with multiple internal peripherals as ADC’s, DAC’s, PWM capable pins and GPIOs, this survey will aim at a more sophisticated software approach. [7]

The definition of the output stage should be done together with the definition of the output algorithm for obvious reasons. The algorithm will be closely tied to the operation of the stage itself. The output stage defines the number of control pins, data pins, and control cycle. The interruption code and sampling frequency are directly connected to the complexity of output’s calculation and control of the output stage. If there are signal generators outside of the microcontroller, the CPU is free from calculating digital integer values and the sampling rate can be drastically increased. There is a downside though: More control pins. With a simple DAC approach, the number of control pins should be minimal, with the drawback of a much more complex interruption block of code since everything will be done by software.

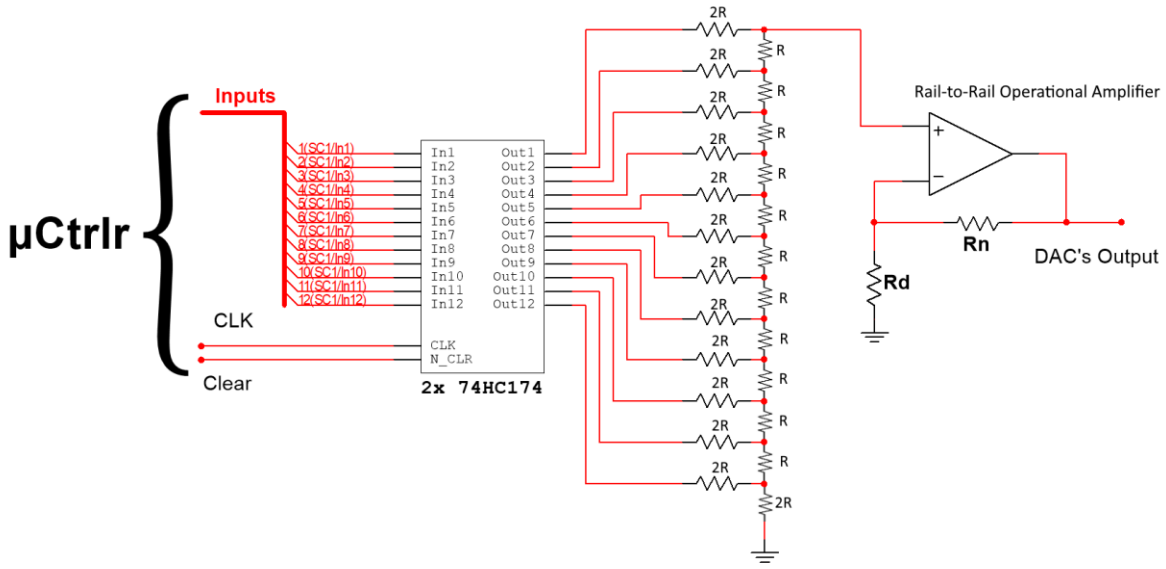
Taking the ESP32 as a good candidate for the core of the project, it runs at 240MHz, really fast if compared with common microcontroller boards from 10 years ago. In addition, usually does not cost more than US$10. The only problem with this board is that, besides having two DAC’s, they are both 8 bits. Therefore, the best solution would be to design our own DAC, using the GPIO pins of the microcontroller as the input. An Operational Amplifier would be used in a R2R configuration. This topology let us design a DAC with a custom number of bits but also brings a problem: How can we deliver the bits synchronously? All bits should be written together otherwise wrong output values would occur in the process of writing the pins without synchronicity. The solution is a simple, multiple inputs, D-type flip-flop. Finally, the DAC’s OpAmp should be rail-to-rail (RRO), in order to make the most out of the power supply that will power the board. Microcontroller boards usually run on 5V or 3.3V. Alternatively, and preferably, the circuit could be powered by two power supplies. One for the microcontroller and another to the output stage’s operational amplifier. This way more common OpAmps can be used, making the project more accessible to build.

The idea is never to rely on rare, expensive components. One big advantage of having this custom discrete topology is the capability to adjust the gain of this DAC through Rn and Rd. The total gain is:

$$A_v = 1 + \frac{R_n}{R_d} \Big| V_{output} = \frac{Input}{2^{n\,bits}} \cdot A_v \cdot V_{DD} \Big| V_{DD} = Power\ supply's\ voltage$$

This DAC amplifier topology is known as R2R non-inverting amplifier digital to analog converter. For a simple unitary gain amplifier, it only takes a 0Ω Rn and a ∞Ω Rd, and Av will be 1. The operational amplifier part of the circuit consist of the resistor ladder connected to the '+' pin of the OpAmp and the '-' pin connected to the output. It is necessary to take care of what value is designed for Av for the output of the amplifier never to exceed the maximum value possible, limit imposed in the operational amplifier by its power supply.

In a 12 bit DAC case our output circuit would look something like this:



The buffer, made from two 74HC174 D-type flip-flops controls the OpAmp's input, and can be controlled by the Clock and Clear pins. The first dictate if data should cross the buffer and the seconds clears it. Finally, the DAC's output would connect to an audio amplifier so the circuit could connect to a speaker or other audio inputs.

6 THE INTERRUPTION CODE

In the end, the core factor of a programmable synthesizer is the interruption function and the code inside it. It is so important because independent from the way you should generate the signal, calculate each value and operate other peripherals, the resolution of your output in time is solely defined by the sampling time, and as we want the maximum definition as possible, it tends to be minimized. The problem of making the sampling time smaller is that once it equals the time our interruption takes to be completed, there is no more room for improvement. Making the interruption code as fast as possible is key to a high sampling rate.

The decision of the output stage and the signal generation method is vital to the interruption code because the more complex it is to generate the control signals and to calculate the output value, the longer is the time that take to complete all those tasks, and we want an interruption cycle as small as possible. With our DAC topology chosen before, all the stage needs from the microcontroller is to periodically write the data and pulse the clock pin. The complexity of this topology lies in calculating the value to be written in the data pins.

The Deep Note is composed of 30 voices and our DAC has 12 bits, resulting in 4096 possible values. If we use 32 voices, each voice will have $2^{12}/2^5 = 2^7 = 128$ possible values. The idea of the microcontroller synthesizer is to generate ramp (saw tooth) waves just like the original one. With this setup, each voice can be generated as a 7 bits ramp that after a slight filtering would sound really close to an analog ramp. This filter should be of a minimal effect and it should be added after the DAC stage and before the sound amplifier, to be adjusted mainly by how the resulting output sounds. A 7 bits ramp should be close enough to original Deep Note. Maybe the additional harmonics (from the steps of the digitalized ramp) will not ruin the original sound but give it more personality. This is where creativity can run free in sound synthesis.

In order to generate all these ramps and sum them to deliver the result to the DAC's input, the project have some constraints. Each ramp will have 128 values possible. Let us suppose that we want a maximum possible frequency of 1500Hz so we can generate all the fundamentals, our sampling frequency should be 1500Hz * 128 = 192,000 interruptions per second. That means the maximum time the interruption will have available to execute all instructions will be less than 5.2µs. In order to minimize the size of the code inside the interruption, all non-essential calculations should be done outside of it asynchronously.

The next challenge is to create all the different frequencies. With a fixed sampling rate, it is possible only to change de output in small window over time. It is a discrete output of a discrete device after all. Horizontal resolution now should be observed as we try to generate frequencies from 30 to 1500Hz. To generate frequencies lower than the maximum a software prescaler could be used. The prescaler should divide the sampling frequency in order to generate slower signals. Additionally another technique to generate signals which the frequency does not match a multiple of the sampling frequency is not to count to 128, but stop the counting before that. This way is possible to generate many more frequencies.

After all this independent ramps are generated, calculated inside the interruption, the only thing left is to sum all of their values into a single (multi byte) variable, set the external pins accordingly to the bit associated with them and update the latch using the clock pin.

7 FUTURE WORK

As there is a microcontroller already in use, there is room for more features. A digital Variable Gain Amplifier (VGA, often called Voltage Controlled Amplifier, VCA) could be implemented so it would let the synthesizer control the output volume. An

algorithm could be developed to differ the volume of each voice independently. This could be very useful, as seen in the spectrogram, not all voices sound with the same loudness as others. The original Deep note tends to generate louder voices in the low frequency range. A Voltage Controlled Filter (VCF) could be implemented after the VCA stage in order to change the timbre of the resulting sound. An input could be used to shift the Deep Note up and down in the frequency domain in order to make it a 'truer' synthesizer, a machine that not only makes sounds, but also let you change them. This could be done with simple buttons or with a soft potentiometer (membrane potentiometer) as well. A soft potentiometer, in contrast to a turning knob connected to a slider, is a slider sensitive to touch. With enough GPIO pins, a display and extra controls could be connected in order to configure extra functions. An SPI SD card reader would be useful to read samples or instructions from a file to generate other voices. An interesting thing to do would be to use the latest Teensy 4.1, released in 2020. It's the fastest microcontroller in the market, clocked at 600MHz, under US$30.

Acknowledgments

Special thanks to Cassio Fachinelli, a fellow college classmate, for the tremendous help with the basic music concepts and the patience to explain stuff more than once, and to Bob Miller (KernlBob), the author of the reference article number 7, for helping me through email with all my questions and doubts about his article, hardware choices and software techniques.

REFERENCE

[1] http://earslap.com/article/recreating-the-thx-deep-note.html

[2] https://en.wikipedia.org/wiki/Musical_note

[3] https://en.wikipedia.org/wiki/Deep_Note

[4] https://www.thx.com/deepnote/

[5] https://www.espressif.com/en/products/socs/esp32

[6] https://www.pjrc.com/store/teensy35.html

[7] https://kbob.github.io/2018/09/27/deep-synth-introduction