Comparing Ways of Producing an Audio Signal with Electronics

Sophia Dong

Abstract:

This study presents two comprehensive explorative efforts at the practical aspects involved in using basic principles of Fourier transform and Shannon sampling theorem to implement audio signals via digital synthesis and basic controlling on a MCU. This project aims to serve as a technical report for beginners, allowing individuals to get a head start on learning about circuits. While not centered on groundbreaking discoveries, the research focuses on providing a detailed, step-by-step account of the construction process. The project encompasses the selection and integration of electronic components, particularly of the PIC microcontroller series; using the C language in the MPLAB IDE XC8 and STM Cube IDE softwares to program working software for the PIC microcontroller to generate a sound. Emphasizing hands-on experience, the study offers valuable insights into the technical intricacies of electronic instrument assembly and production. Additionally, the research emphasizes the creative potential inherent in DIY instrument making, fostering a sense of innovation and exploration within the maker community, especially among those who are relatively new to the realm of electronics. By documenting the construction process, sharing the knowledge gained, and breaking down the work into easily digestible pieces for an unfamiliar audience, this research contributes to the collective expertise of hobbyists, electronics enthusiasts and aspiring instrument makers, encouraging further experimentation and innovation in the realm of electronic musical instruments.

Introduction

The technical end goal of this study aims to produce an audio signal using two different methods of DAC architecture using two different microcontrollers through analysis of the differences in architecture, audio output, and practicality. Moreover, along with meticulously documenting the needed materials, the used methods, and techniques employed, this research will provide myself a familiarity on the product cycle from hardware definition, construction, and software integration. It serves as a valuable resource for hobbyists, electronics enthusiasts, and those aspiring to create their own circuit applications in the future, especially those who are complete beginners or relatively new to the field of audio synthesis. The primary objective is to offer a step-by-step guide that not only showcases the technical aspects of circuit assembly and coding, but also highlights the creative and innovative possibilities involved in each step within this field.

Current research has seen individuals creating convoluted and confusing resources, often involving the knowledge of advanced mathematics [3, 12]. Simple, easy-to-understand, beginner-friendly articles are often hard to come by. Even then, authors often don’t explain the logic behind why they made the choices they did. This can be extremely frustrating, especially for those looking to teach themselves how to create a circuit or who might not have the resources to have a private tutor of sorts to help them on their journey. As a result, this study also aims to provide concrete reasoning behind each decision being made in hopes of outlining a clear logic related to getting a start. Additionally, this research also aims to expand the knowledge of the reader by providing an application note of a more professional-model MCU,

specifically related to using an STM32F407 discovery board to further enhance the created 6-bit to a 16-bit code.

Definitions

Below includes a list of industry-specific terms that will be often used throughout the text. As such, it is assumed that the reader understands the following terms as well as the definition/function of each term. It is strongly suggested for those unfamiliar with the terms to find the meanings online. Additionally, it is assumed that the reader has basic knowledge of basic math, circuitry concepts, and coding.

Here is the list of terms the reader is assumed to know before reading the paper: Sine/Sine Graphs. Harmonics/FFT, Coder/Decoder (CODEC), I2S, Pulse width modulation (PWM), Duty Cycle, Look-up Table, Bit resolution, Integrated circuit, Low-pass circuit, Digital to analog converter (DAC), Microcontroller (MCU), Sampling rate, Shannon's Sampling Theorem, Aliasing, Direct Memory Access (DMA), Ping-pong dual memory, dual bank ping pong buffers , Circular buffer, Logic analyzer, Sampling rate, USB/USB-OTG, Software Development Kit (SDK), Schematic, industry standard

Goals:

The technical goal is to use basic principles of Fourier transform and Shannon sampling theorem to implement audio signals via digital synthesis and basic controlling on a MCU. This project aims to serve as a technical report for beginners, allowing individuals to get a head start on learning about circuits.

Approaches:

Difference	**Approach A**	**Approach B**
Microcontroller	PIC16F88, older and more hobby-ist level but involved in more basic operations	STM32, industry standard ARM Cortex M4F MCU
DAC Architecture	Filtering a digitally-produced PWM signal	Generating an analog voltage via a dedicated audio DAC through an I2S bus
Goal waveform	Pure sine wave controlled via interrupt and look-up table (tone table)	Recorded waveform stored on a USB
Audio Output Resolution	Up to 10-bit, only implemented 6-bit	Default stereo 16-bit

The approaches taken differ in four ways. The first difference is the complexity of the microcontroller. Approach A's MCU was constructed on a circuit built from scratch. Due to the nature of this MCU and its software development kit (SDK), it synthesized a relatively lower bit

resolution audio output and a limited variety of tones [7]. On the other hand, Approach B's MCU was used on a pre-made STM32F407 discovery board. The powerful 16-bit MCU and its more advanced SDK allow for more versatility in applications [11]. Additionally, any arbitrary audio waveform stored on a flash drive using the FAT32 register can be played [1, 9]. The STM32 circuit also uses the implementation of a DMA pass to DAC and ping-pong dual memory bank in a circular buffer to process the data flow [1].

The second way is in their *DAC Architecture* and their consequent audio output. Approach A synthesized the tone by pure MCU programming and created a purely-digital PWM signal based on a lookup tone table. Following circuitry filters the PWM signal to generate a sine tone which drives the speaker via a power amplifier. While Approach B uses an I2S audio CODEC [1].

The third difference is in their *goal waveform*. While both approaches aim to output a sine wave, Approach A digitally-produces a sine wave to approximate the goal waveform [7]. On the other hand, Approach B approximates a WAV file in a USB to transcribe a recorded audio WAV file that is stored in a USB thumb drive [1].

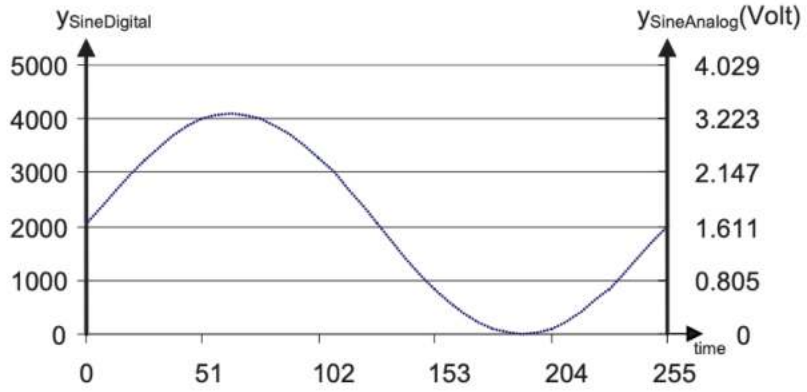


Figure 1. The output of the goal waveform for Approach A is supposed to emulate a perfect sine wave. Each sampling value corresponds to outputting a different voltage as shown. Credits: [1]

Finally, the fourth difference is their *audio output resolution*. Approach A uses 6 bits of resolution to define the duty cycle of its generated PWM signal. Approach B uses 16 bits resolution to define the output voltage of the DAC built into the STM microcontroller. As a result, more accurate audio signals will be preserved, producing a cleaner sound [5]. **In this method, the voltage determines which DAC channel processes and plays the resulting analog signal through the speaker which then gets heard by the individual [1].**

Approach A

The basis of tone generation for our PIC microcontroller was related to fundamental audio concepts. Any arbitrary audio waveforms can be decomposed into sine wave components according to Fourier analysis [6].

In Approach A's circuit schematic shown in Figure 2, we included eight push buttons to represent piano keys. Depending on which button is pressed, a different note will be played, generating the frequency according to corresponding piano note values in Figure 7.

The form of DAC architecture uses pulse-width modulation (PWM), a technique which involves varying the duty cycle of a digital output pin [7]. Sources [7], [12], and my basic knowledge were used to generate a code which produced a sine tone.

A second order low-pass filter was created to filter any unwanted harmonics, producing a clearer sine tone with less distortion. The basic low pass filter consisted of a tandem configuration made of two resistors and two capacitors. The filtered output then drives a speaker power amplifier to push the low resistance speaker to generate the sound.

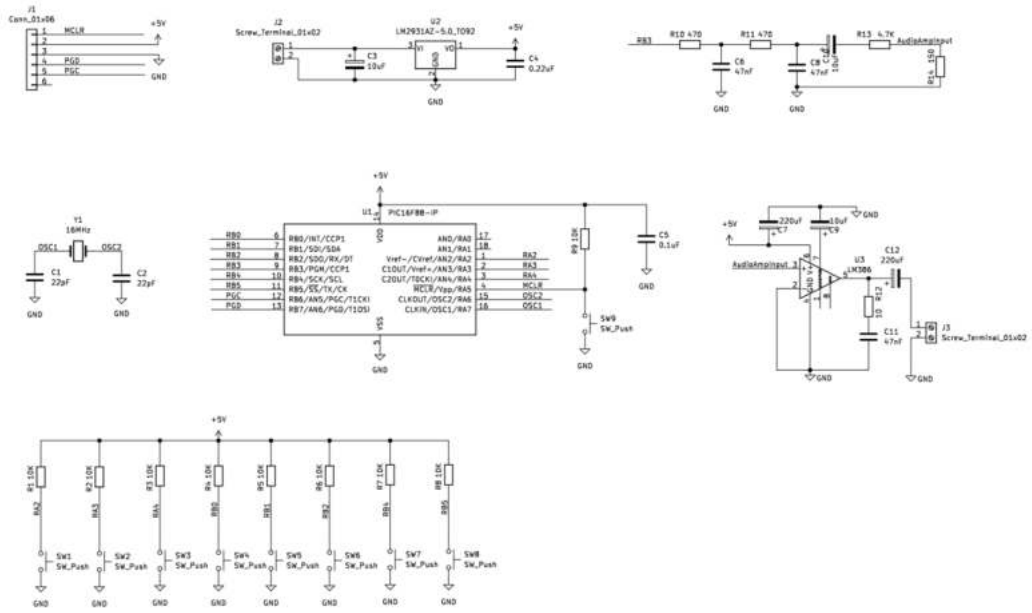


Figure 2.
KiCAD schematic of the piano in approach A. The diagram includes a PIC16F88 microcontroller, a simple power management integrated circuit (PMIC), a bank of eight buttons, second order low pass filter and a speaker power amplifier.

Construction of the circuitry

Making a circuit board starts from the schematic. As such, we have created the schematic using KiCAD, a free, easy-to-use software that is ideal for hobbyists. Based on the schematic shown in Figure 2, we then generate a PCB file which is shown below. Additionally, we made the circuit board using the approach shown in this source [9].

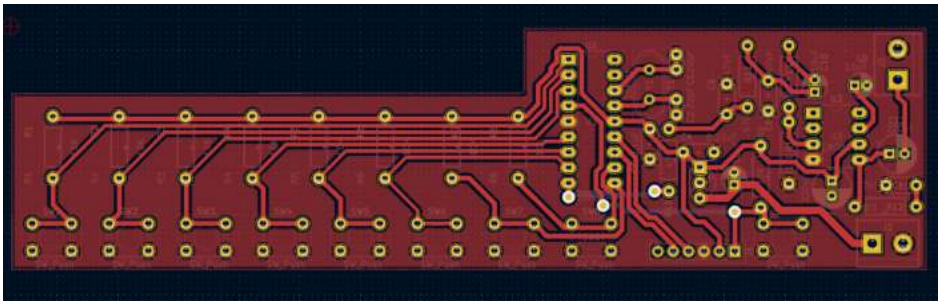


Figure 3. The board layout of the PIC16F88 piano. This PCB layout converts the schematic to the real components and uses real traces to connect all these components.

Based on the board layout, we generated gerber files which were used to fabricate the physical circuit board. Using a table top CNC machine, we machined the circuit board, defining the copper trace connections to construct the circuitry in the process.

After careful inspection to avoid short and open circuits on the fabricated PCB, we then soldered all the components on the PCB to construct the hardware of the piano.

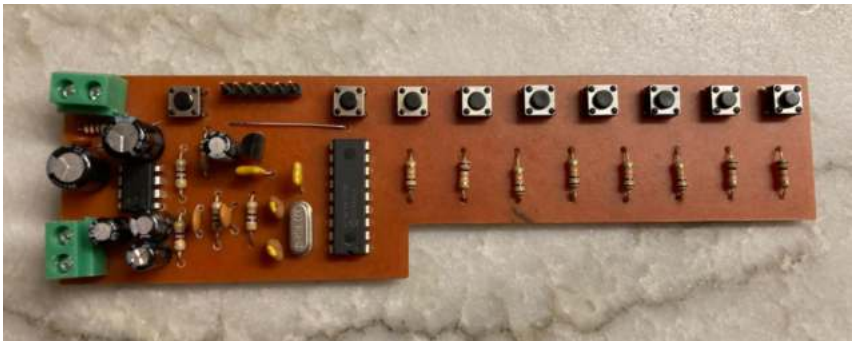
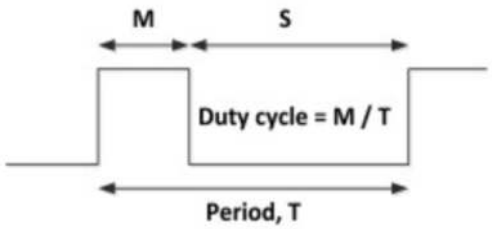


Figure 4. The actual piano was constructed after CNC milling, component soldering and inspection.

Software

To program the PIC16F88 MCU, we used MPLAB IDE and XC8 compiler to generate our C code. As shown in Figure 2, we implemented a PWM output via the RB3 pin.

The PWM generates the analog voltage by varying the duty cycle of the RB3 pin staying on logic high which is 5V in this case [4]. The duty cycle is controlled by three registers, Timer2, PR2, and the CCP1L registers [data sheet]. By changing the value stored in the CCP1L register according to the lookup table, we can change the duty cycle of the PWM pin voltage and



therefore, the analog voltage driving the speaker.

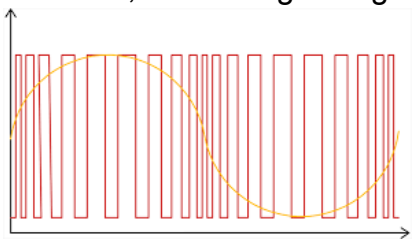


Figure 5. Left: The duty cycle is a percentage showing how much of the period is spent on the voltage high setting. Right: Sine waves can be implemented via PWM with varied duty cycles. The resolution of PWM output depends on the duty cycle resolution [4, 7].

Although the PIC16F88 can output up to 10-bit resolution, we wanted a simpler implementation in our C code and only implemented a 6-bit resolution PWM paradigm. In our current case, there are 64 (2^6) steps in the voltage domain meaning the smallest step in the voltage domain would be 5/64 which is about 78.125 mV.

As shown in Figure 5, to implement a proper sine tone, we need to output the correct analog voltage via PWM at a correct given time [7]. The time interval to change the PWM output is determined by the period of the sine tone we want to play according to a tone table. We divided each period of a full sine tone into sixteen intervals. Therefore our sampling rate is determined by the tone frequency times 16. The varied sampling frequency is not a typical way of implementing a DAC but is very simple to implement in Approach A.

```
unsigned char sin_table[16] = {0, 2, 9, 19, 32, 44, 54, 61, 63, 61, 54, 44, 32, 19, 9, 2};
```

Figure 6. This table is from the resulting calculated sine tables from [8].

After a full cycle of 16 voltage points according to the sine look-up table, we need to restart from the beginning of this look-up table. How frequently we go through the full sine cycle is determined by the note frequency we want to play. We used a second timer in PIC16F88, Timer1 to precisely control how frequently we go through a full sine wave cycle.

```
unsigned int tone_table[8] = {65058, 65110, 65156, 65177, 65216, 65251, 65282, 65296};
```

Figure 7. We have eight different tone table values corresponding to eight different buttons. Because we were using a counter to define the time, the actual interrupt interval is the count number from the values in this table count up to 65536. To find the frequency

of each tone played, we subtract the table value from 65535 and divide it by the clock frequency/4 (single instruction cycle time)[7].

The PIC microcontroller series requires the user to program in C. Through the MPLAB IDE XC8 software as well as a Pick-it 3 device, the user can flash code into the microcontroller. The program contains a section where certain elements of the microcontroller such as the internal timer or brown-out mechanism could be enabled. Another section initializes the required timers and pins as well as the register compatible with enabling PWM, an interrupt code which implements the PWM mechanism. The final section contains a main code that detects the button pressed and varies the Timer1 according to the button position and corresponding tone frequency.

```
void __interrupt() isr(void) {
    TMR1=tone_table[x];
    if (PIR1bits.TMR1IF==1) {
            n++;

            if (n>=16)
            {n=0;
            CCPR1L=0;
    PR2=tone_table[n]
            }
    }
    if (button ==1)
            {
            CCPR1L= sin_table[n];
            }
    else
        {CCPR1L=0;}
    PIR1bits.TMR1IF=0;

    }
```

Figure 8. This is the interrupt code setting the proper period according to the tone frequency we want to play.

Approach B

Approach B, in [2]'s approach uses an STM32F407 microcontroller to read the wav files stored on an external storage media and parse the data into dual bank ping pong buffers and then pass the data directly to a Cirrus Logic CS43L22 audio DAC to generate a 16-bit audio output. Since the CS43L22 includes a power amplifier, it can drive an 8 ohm speaker directly. Based on [10]'s pin schematic, [3] creates a functioning audio player which can skip, play, and stop. Like [3], we are generating audio tones. Sources [10] and [3] were used to configure the code of my board accordingly.

Because Approach B uses an industry standard board, the sampling rate is not varied like Approach A's but fixed at either 44,100 or 48,000 Hz [6].

Board

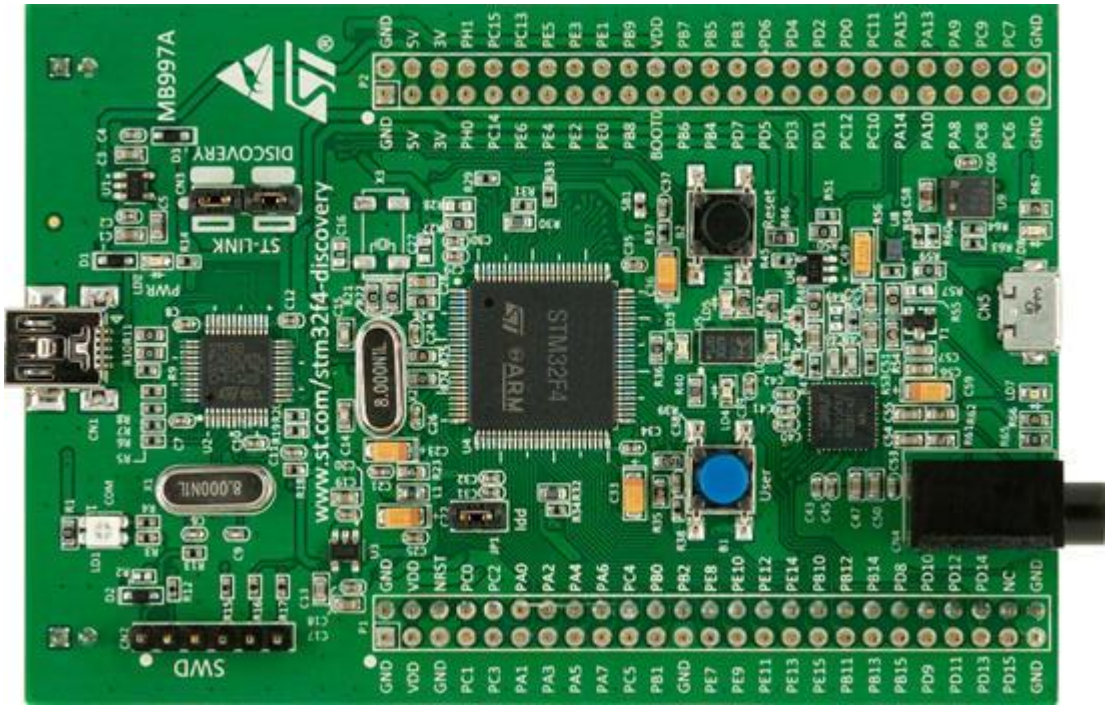


Figure 9. STM32F407 discovery board from STMicro. The center chip is the STM32F407 MCU. The audio jack is the big black figure on the bottom left. Near the audio jack is the CS43L22 audio DAC. The USB port at the bottom is used to connect the USB flash drive storing the audio WAV files.

Software

As stated before, the DAC in STM32 discovery board has 16 bit resolution, therefore, there are 65536 steps in the 5V reference voltage which means the actual step would be 5/66536 which is about 76.3 uV.

The program automatically adjusts to the changes in the frequency from the pre-recorded track. As such, a variable is not required to address the issue unlike Approach A.

Similar to the PIC microcontroller series, the STM32 series requires the user to program in C. Through the STM32CubeIDE software and a STLink converter, the user is able to quickly flash code into the microcontroller.

Other source files and header files downloaded from [2] were used in the program to be used throughout the program. Some of these include File_Handling.h to handle USB input and cs43l22.c to manage the external CL43L22 audio DAC [3].

While Loop Pseudocode

```
while (1)
{
  /* USER CODE END WHILE */
  MX_USB_HOST_Process();

  /* USER CODE BEGIN 3 */
  if (Appli_state==APPLICATION_READY)
  {
      Mount_USB();
      AUDIO_PLAYER_Start(0);

      while (!IsFinished){
          AUDIO_PLAYER_Process(TRUE);

          if (AudioState == AUDIO_STATE_STOP){
              IsFinished = 1;
          }
          }
      }
  }
}
/* USER CODE END 3 */
```

Figure 10. In the main while loop of the code, the program checks if the USB is mounted. Then, it starts the audio-playing process. When there are no more tracks to be played, the program is terminated [3].

Results:

Approach A

In Approach A, the PIC-based circuit produced a monotone, plain sounding via a PWM signal. The logic analyzer reading showed a pure sine wave while the FFT showed all of the frequencies generally concentrated at one point.

To ensure accurate results, a Saleae brand logic analyzer was attached to several testing points in the circuit— a terminal to ground it and a terminal before the amplification— to ensure its PWM was working and that the eventual output took the shape of a sine wave.

Logic Analyzer Reading

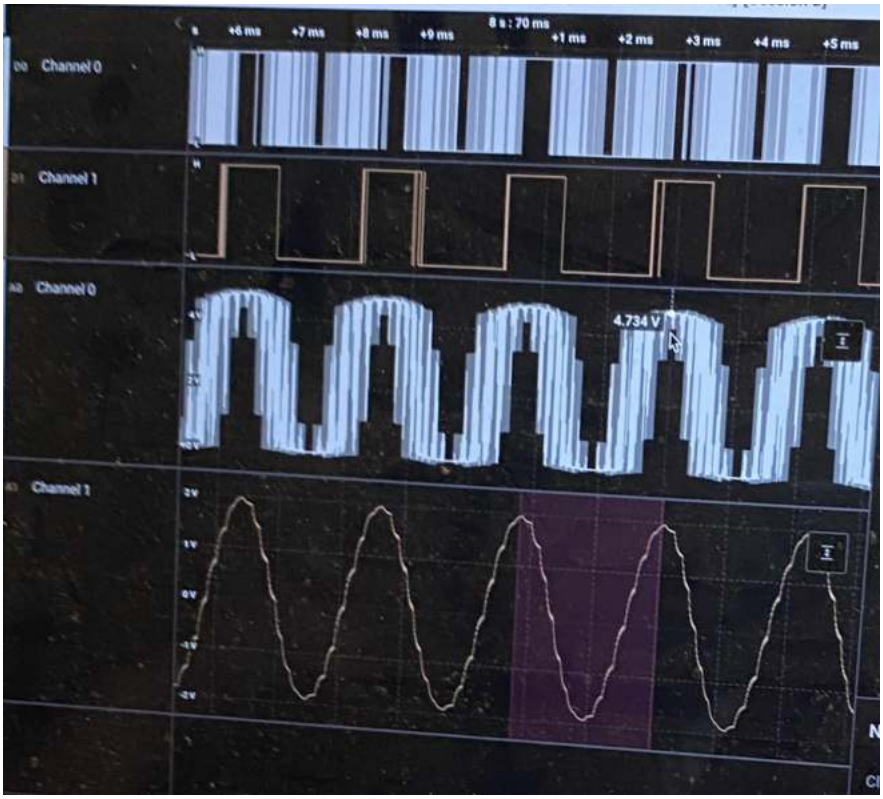


Figure 11. PWM output of PIC16F88 board. Top row is the raw PWM output, third row is the analog representation of PWM output and fourth row is filtered PWM output which is very close to a sine wave.

Approach A's FFT

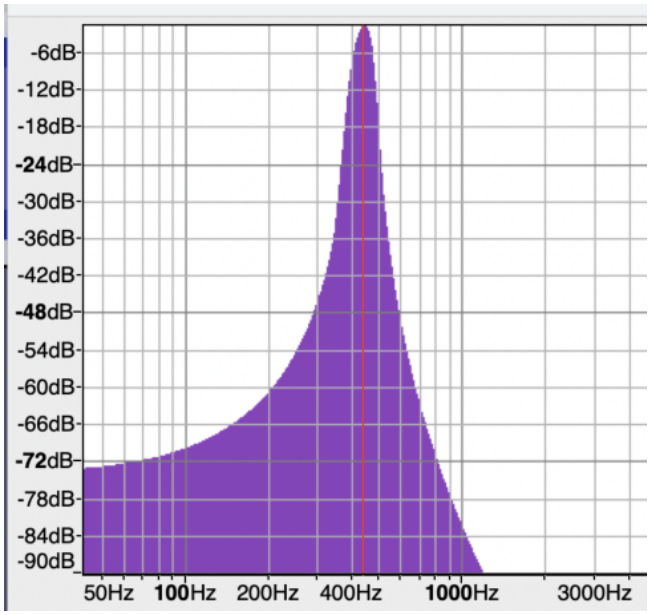


Figure 12. FFT drawing of the output in the fourth row from Figure 11. In this diagram, the note being played is A4 which has a frequency of 440Hz.

Approach B

The program flashed onto the pre-made STM32F407 discovery board used in Approach B played recorded piano notes stored audio on a USB. Visually, Approach B FFT has higher harmonics, but it also has a greater noise floor than Approach A's FFT. Based on these differences, it's likely that Approach A would have a purer-sounding note. However, given the visual similarities between Approach B's FFT and an actual piano note's FFT, we can reasonably infer that Approach B will produce a sound similar to that of a real-life piano note.

Approach B's FFT

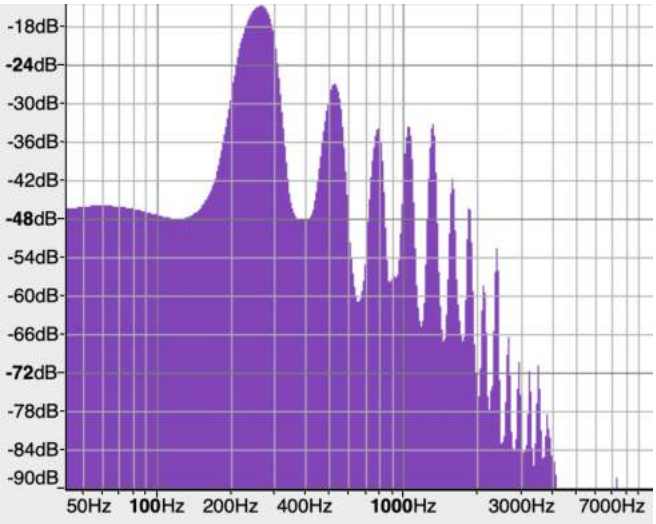


Figure 13. FFT of one section of audio output from Approach B. In this diagram, the note being played is C4 which has a frequency of 262Hz. Note the various harmonics denoted by peaks in locations that are multiples of 262Hz (i.e. 524Hz, 786Hz).

Piano Sound FFT

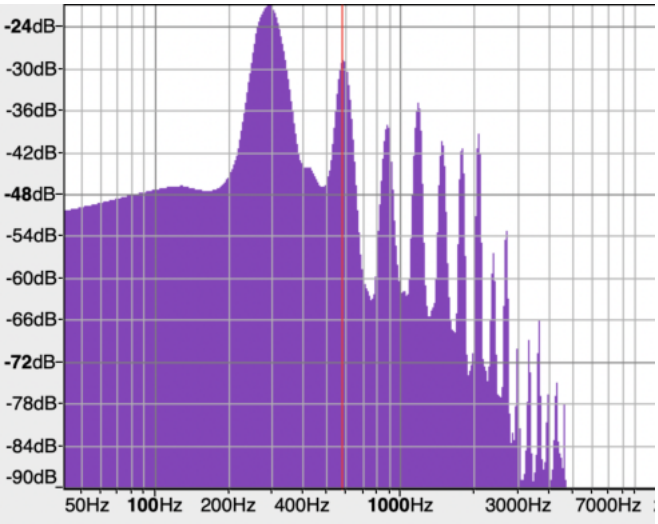


Figure 14. FFT of an actual piano note. The note being played here is a D4 which has a base frequency of 294Hz. Note the differences and similarities between this graph and the ones shown in Figures 12 and 13.

Conclusion:

This project gives me first hand experiences in many concepts that are beyond a high school student's scope. From the hardware development cycle, I learned about the basic building blocks of a microcontroller board, the steps of making a circuit board, and how to

program a microcontroller board. I also grasped the basic concepts of hardware-related C coding.

Additionally, I learnt basic concepts of digital audio such as Fourier Transform and Shannon's sampling theorem, both of which are pillars supporting the current digital audio on smartphones, TVs, internet music and social media.

Through this project, I understood certain tradeoffs between sound quality, power consumption, hardware and software complexity.

Limitations: Because the microcontroller brand was only compared in one section, it is hard to determine which of the listed differences are actually responsible for the results from this paper. Moreover, we were only able to speculate how Approach B would actually sound by comparing it to a known graph of a piano sound. Finally, not everyone has the resources to replicate the comparisons done in this technical report.

Future Work: This project will likely be enhanced in the future by incorporating user interfaces to create a satisfactory experience for the user; and addressing challenges related to acoustic emulation to generate more realistic-sounding piano sounds, combining the components used in Approach A and Approach B to create a digital piano sound.

Acknowledgements:

Thanks to Bibit for guiding me throughout the entire project. I would like to also thank Courtney Smith for reviewing my paper.

Works Cited:

[1] *Audio and waveform generation using the DAC in STM32 products* [Brochure]. (2020). STMicroelectronics. https://www.st.com/resource/en/application_note/an3126-audio-and-waveform-generation-using-the-dac-in-stm32-products-stmicroelectronics.pdf

[2] ControllersTech. (n.d.). *WavePlayer using STM32 Discovery*. ControllersTech. Retrieved December 9, 2023, from https://controllerstech.com/waveplayer-using-stm32-discovery/

[3] ControllersTech. (2021, January 17). *WAVEPLAYER using STM32 || I2S AUDIO || CS43L22 || F4 DISCOVERY* [Video]. YouTube. https://www.youtube.com/watch?v=_Pm0L1ropJs

[4] Dogan, D. (2014). Generating Pulse-Width Modulation Waveform. In *Designing Embedded Systems with 32-Bit PIC Microcontrollers and MikroC: Chapter 8 - Advanced PIC32 Projects* (pp. 359-442). Elsevier Ltd. https://www.sciencedirect.com/topics/engineering/pulse-width-modulation#:~:text=Pulse%2Dwidth%20modulation%20(PWM),to%20power%20control%20and%20conversion.

[5] J. J. Wikner. (2014, February 9). Note on the Power-Speed-Resolution trade-off for a DAC. https://mixedsignal.wordpress.com/2014/02/09/power-speed-resolution-tradeoff/

[6] McAllister, M. (2023, June 1). *Digital Audio Basics: Aliasing Explained*. https://producelikeapro.com/blog/digital-audio-basics-aliasing-explained/#:~:text=Max%20McAllister,other%20artifacts%20into%20the%20recording

[7] *PIC16F87/88 Data Sheet 18/20/28-Pin Enhanced Flash Microcontrollers with nanoWatt Technology* [Pamphlet]. (2005). Microchip Technology Inc. https://ww1.microchip.com/downloads/en/devicedoc/30487c.pdf

[8] *PIC16F88_organ_tone* [Table]. (n.d.). https://docs.google.com/spreadsheets/d/1k-E058Yxvg-VUPHUuZCpRiVFsv-ybpGm0FHm6Kd1CHQ/edit#gid=0

[9] Wach, Matt. (2022). *Milling Printed Circuit Boards (PCBs) on a Cheap CNC Machine.* https://www.instructables.com/Milling-Printed-Circuit-Boards-PCBs-on-a-Cheap-CNC/

[10] *STM32F405/415, STM32F407/417, STM32F427/437 and STM32F429/439 advanced Arm®-based 32-bit MCUs* [Brochure]. (2021). STMicroelectronics. https://www.st.com/resource/en/reference_manual/dm00031020-stm32f405-415-stm32f407-417-stm32f427-437-and-stm32f429-439-advanced-arm-based-32-bit-mcus-stmicroelectronics.pdf

[11] *STM32F4DISCOVERY* [Pamphlet]. (2012). chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://www.st.com/content/ccc/resource/technical/layouts_and_diagrams/schematic_pack/group1/0f/91/8b/39/b3/78/4d/c4/MB997-F407VGT6-B02_Schematic/files/MB997-F407VGT6-B02_Schematic.pdf/jcr:content/translations/en.MB997-F407VGT6-B02_Schematic.pdf

[13] MECHATRONICS. (2015, December 9). *Tutorial 8 Interrupt Programming and its simulation using Proteus for PIC microcontroller* [Video]. YouTube. https://www.youtube.com/watch?v=UjoDGipCIeg