Project Report for Touchscreen Piano

Tik Tsoi

***Abstract*—The implementation of a 9-key touchscreen piano on an 8-bit microprocessor is presented in this paper. Additional functions such as record and replay are available to facilitate user experience. Both software and hardware design are discussed in full. On the whole, the product works reliably and precisely. Alternative signal processing techniques were tested, and we propose using a digital-to-analog converter (DAC) to synthesize the true waveform from a piano for the best sound quality.**

I. Introduction

TRADITIONALLY, for beginners to learn how to play the piano, owning the actual instrument is necessary. Regardless, some families may not afford a piano because it is a costly investment. As an alternative, a variety of piano keyboard apps for tablets and other touchscreen devices are currently available on the market. However, even with this alternative, low-income families may still face difficulties.

With the touchscreen piano, we hope to provide a more economical solution to this problem. The product is highly scalable, capable to display all 88 keys of a genuine piano on a larger touchscreen. To enhance the learning experience, we also included a range of functions including record, replay and clear. This will allow the beginner to listen to their own music at any point.

II. High-level design

A. Software

The functionality of the touchscreen piano was constructed upon four core modules, including initialization, input reading, input processing and outputting.

1) Initialization: Once the program begins, a series of hardware components, including the graphical liquid crystal display (GLCD), analog-to-digital converter (ADC) and touch panel, are configured and turned on. A 9-key black-and-white piano keyboard is then displayed on the GLCD by loading a bitmap of our keyboard design, which is stored in four different tables. In addition, two timers TMR0 and TMR1, configured at different clock frequencies, are also enabled. When a timer finishes counting a cycle, an interrupt is called.

2) Input reading: When the touch panel is struck, two analog inputs, corresponding to the x and y coordinates of the location struck, enter the microprocessor and are read in its internal ADC. The ADC then converts the two analog signals into digital ones, which are now temporarily stored as two 12-bit numbers. The program runs in a loop to detect also any new command, such as record, replay and clear, alongside the touchscreen input.

3) Input processing: The touchscreen input requires two further conversions before it can be synthesized as a piano sound. The first step maps the 12-bit digits into one of the nine notes by comparing the number to the pre-determined coordinate ranges of these keys. If the number falls within a range, it indicates that the corresponding key of this range is pressed. To represent our result, the note is stored as a distinctive bit pattern. The second step converts the note (now a bit pattern) into its frequency by comparing the bit pattern to a list of all patterns. Similarly, if the pattern matches, we can infer which key is pressed. This assigns a rollover number, defined as the starting position of a timer's cycle before it rollovers, to TMR0. This changes the frequency of its interrupts, which in turn changes the frequency of the square wave output that they are responsible for generating. Although the conversion could have been done in one step, a 2-step choice was explicitly made for ease of debugging - a note is nicer to work with than a bit pattern.

4) Output: In the event of a record/ replay command being made, a rollover number is stored/ retrieved from the data memory of the microprocessor, which happens as a result of the interrupt operations called by TMR1. In the event of a clear command made, all the stored notes in data memory are removed. The actual synthesis of the piano note comes about by generating a square wave output, which relies on other audio components to make a sound.

B. Hardware

The hardware architecture constitutes three major blocks, including touchscreen, microprocessor and audio synthesizer.

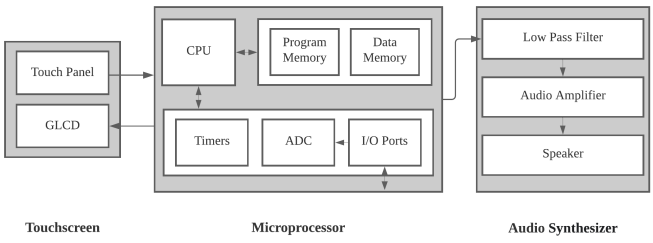

Figure 1. A block diagram that illustrates the hardware architecture of the touchscreen piano. It consists of three main blocks, namely the touchscreen, microprocessor and audio synthesizer. Arrows depict the transfer of data.

1) Touchscreen: We are using a touchscreen that is assembled by a GLCD and a touch panel on top. The GLCD has a 128x64 resolution and itself is further composed of two 64x64 selectors. The touch panel is 4-wire analog-resistive [1], which is put under a 5V voltage across the ends of the vertical and horizontal directions. When a touch event occurs, an analog voltage is measured in each axial direction. The touchscreen is interfaced on the microprocessor development board.

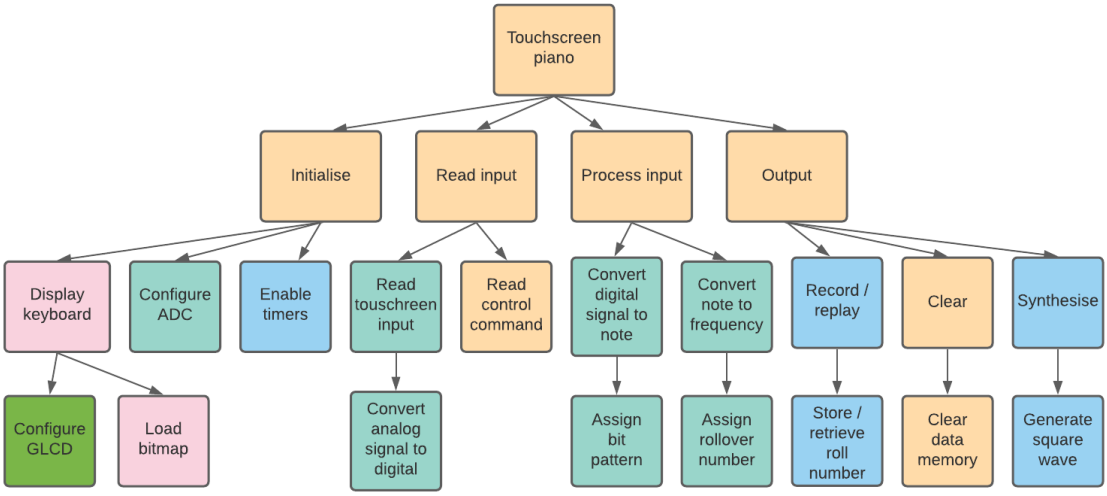

Figure 2. A top-down modular diagram that illustrates the high-level software design of the touchscreen piano. Modules are colored according to the files they are in.

2) Microprocessor: Our program runs on an 8-bit 64MHz microprocessor. The central processing unit (CPU) transceives control and data signals amongst modules internally, performing logical operations when needed. Apart from our program code, the program memory also stores tables, which enables the display of the piano keyboard. The data memory is used to store recordings as a sequence of rollover numbers which is retrieved upon a playback request. In contrast to the program memory, the data memory only stores values temporarily, meaning all the recordings will be lost if the microprocessor is powered off or the program is rerun. External devices such as the touchscreen and low pass filter are interfaced to the microprocessor via the I/O ports, allowing the exchange of input and output between the three major blocks.

3) Audio Synthesizer: The square wave output from the microprocessor is converted into a sinusoid at a second order active low pass filter. A sinusoid is favourable, as it more resembles the actual analog sound wave a piano produces, which is believed to improve the overall sound quality. The circuit consists of two RC networks (cascaded with an operational-amplifier), which removes the high-frequency components from the square wave beyond our desired cutoff frequency. Previously, a third order passive low pass filter was implemented. However, this alternative suffers from the disturbance of white noise. Finally, the attenuated electric signals are amplified by an audio amplifier and passed on to the speaker to synthesise music.

III. Software & Hardware Design

A. Software

The low-level implementation of the software was designed on the main program and two high-priority interrupt service routines, ISR0 and ISR1. Briefly, the main program initializes the system, after that, it polls perpetually to look for any new touchscreen input or control command. ISR0 generates the square wave output and ISR1 samples/ recreates the touchscreen input regularly if a record/ replay command is requested.

1) Initialization: The ADC module has three control registers: ADCON0, ADCON1 and ADCON2. ADCON0 controls the operation of the module and is first set to high to power on. ADCON1 configures the ADC's voltage reference which should be set to match that of the touch panel ideally. Otherwise, we risk losing the higher voltage levels and overall resolution. As opposed to the 5V reference stated in the datasheet [2], the experimental value was found to be around 2V, hence an internal voltage reference of 2.048V was chosen. A pair of registers, ADRESH and ADRESL, stores a 16-bit result of the AD conversion initially. ADCON2 allows us to choose the left justification of our result, meaning only the most significant 12 bits are selected, with 8 bits from ADRESH and 4 bits from ADRESL. Thus, the final ADC reading covers the full range of the touch panel's resolution.

Each selector of the GLCD contains eight pages (big rows) which individually contains eight rows of pixels. We converted a black-and-white image of the 9-key keyboard design into a 128x64 monochrome bitmap which consists of decimals of either 0 or 255. The bitmap was broken up into four parts, corresponding to the top left, bottom left, top right and bottom right sections of the keyboard. A vertical division is necessary as the left and right selectors have to be programmed independently. A further horizontal division marks the vertical extent of the black keys. To minimise the use of memory space, only the first row (containing 64 numbers) of each section is stored in a table. When a table is loaded, the decimals stored are read in as an 8-bit binary. For instance, 0 (black) translates to 00000000B and 255 (white) translate to 11111111B. This binary number then sets a pixel column on a page. This step begins from the leftmost column, continuing to the right until the page is fully set. The same table is overall looped over four times to display a section completely.

The configuration of the two timers, TMR0 and TMR1, determines the output frequency f_{output} of their respective

interrupt service routine. Exactly, their relationship is the following

$$f_{output} = \frac{f_{clock}}{N_{count} \times N_{data}} \quad (1)$$

where f_{clock} is the clock frequency of the timer, N_{count} is the number of counts left before a rollover and N_{data} is the number of data points in a cycle. In ISR0, a square wave output is generated by decrementing and incrementing port J every cycle, meaning N_{data} is 2. Hence, to cover our 9 notes between 261 Hz (C) and 440 Hz (G♯), TMR0 was chosen to be 16-bit with the lowest f_{clock} of 62.5 kHz. To sufficiently capture all touchscreen inputs, TMR1 was configured to be 16-bit with a higher f_{clock} of 2 MHz. Without a rollover number explicitly set, the timer counts from 0x0000 to 0xFFFF, which results in an output time period of ≈ 0.033 s.

2) Input Reading: The x and y coordinates of the touchscreen input arrive in the microprocessor as two analog signals. To read in x, DRIVEA of the touch panel has to be powered and the input is then moved to an AD input channel (AN10) for conversion into a 12-bit number. Similarly, for y, DRIVEB is powered and the input is moved to another channel (AN5). Only the most significant eight bits stored in ADRESH are retained because the remaining bits are offering extra precision which is not helpful in our case, given the comparatively larger size of a pixel. Other control commands, including start recording, stop recording, clear recording, start replay and stop replay, can be enabled on port C of the microprocessor development board. The main program checks in the above order if their pin is set. If so, their corresponding global variables are set to the right state. The main program remains in polling to endlessly loop over reading in new touchscreen input and detecting new control commands. This procedure has the advantage of making sure a spontaneous response to any incoming input. In between each loop, a delay of ≈ 0.03 s is added to reduce the effect of switch bouncing. The implementation of the main program and the specific task of each control command are presented in detail in Figure 3.

3) Input Processing: The aim of the first conversion is to identify the key pressed. To calibrate the touch panel, ADC readings of certain keyboard boundaries (see Figure 4) were determined, which are used as thresholds to help categorise our touchscreen input to the key that it belongs to. We found that when no key is pressed, the ADC still returns a y reading which is below that of the minimum bound. Hence, a value above the minimum implies there is an incoming input. The input is then classified into the top or the bottom half, where the top contains black and white keys and the bottom is solely composed of white keys. The x coordinate is further compared with the vertical boundaries of its half, from the leftmost and to the right. If the coordinate lies within the limits of a key, a distinctive bit pattern will be assigned.

The second conversion is aimed to establish the frequency for our square wave output. Our bit pattern is compared to a sequence of reference patterns. If the bit pattern matches, the program then branches to a function of that note which will assign its corresponding rollover number to the ***freq_rollover*** variable.

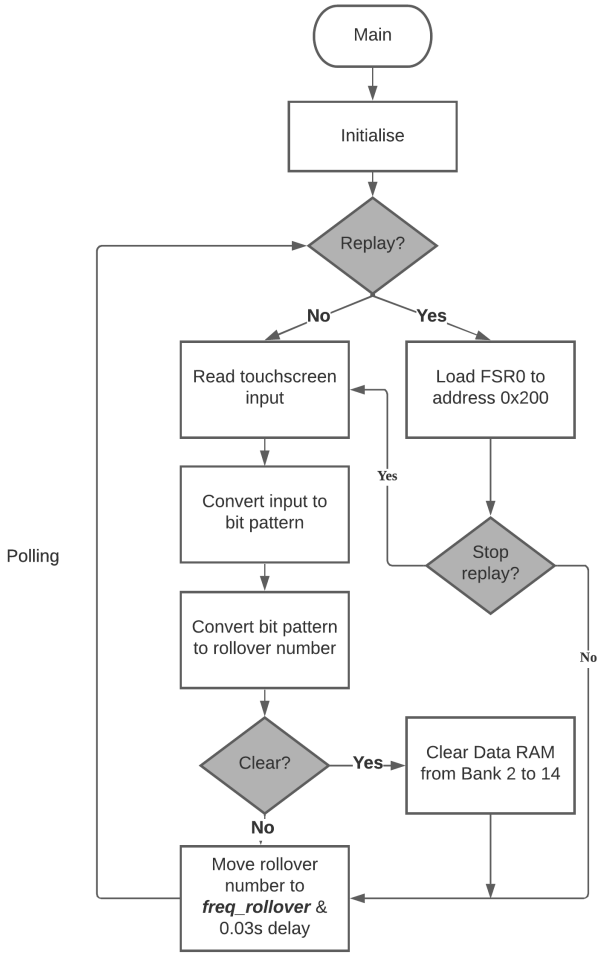

Figure 3. A flowchart that illustrates the low-level software design of the main program. Once started, the main initializes the system and remains in an endless loop to detect new input or control commands.

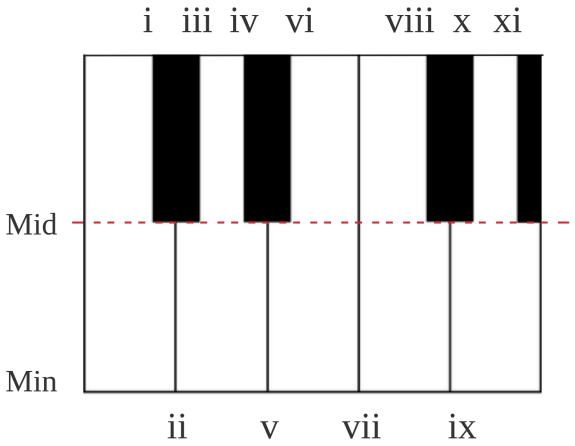

Figure 4. The display of the touchscreen piano shown on the GLCD. Each roman numeral is a label for a vertical boundary, whereas Min and Mid represent the horizontal boundaries of the minimum bound and mid-level of the keyboard. The ADC readings of these boundaries were determined by manually poking individual borderline with a sharp instrument.

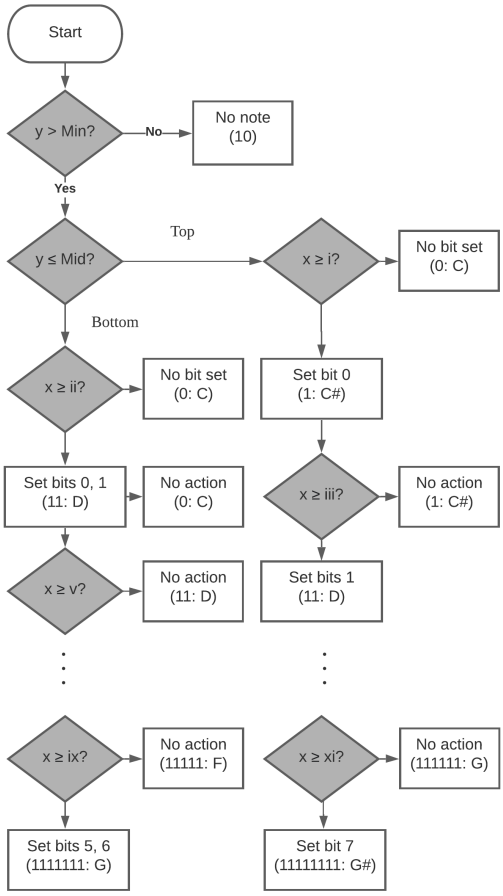

Figure 5. A flowchart that shows the first conversion step of the touchscreen input. At any stage, a coordinate is always compared to a boundary, which will result in a final bit pattern indicative of the key pressed.

4) Output: The output of the touchscreen piano is entirely controlled by the two high-priority interrupt service routines. Since the two occur at different frequencies (ISR0 happens at least ten times more often), it is very unlikely that they will interfere with the other when one is on. Henceforth, a priority hierarchy between them is not imposed. When a timer rollovers to the next cycle, an interrupt flag for its associated ISR is automatically set. In such an event, the program leaves the main and jumps to an interrupt. The program first checks if it is in ISR0 or ISR1. If the program is in ISR1, it will branch off and check if either the record or replay variable is set. If the variable for the record is set, this suggests a record command has been requested and the program will execute the function, likewise for the replay command.

As banks 0 and 15 in internal SRAM are partially occupied, banks 2 to 14 were chosen for storing a recording. To achieve this, a file select register FSR0 was used for indirect addressing. Similar to the ADC case, FSR0 is composed of a pair of registers FSR0H:FSR0L, which stores a 12-bit number. FSR0H holds four bits and indicates which bank we are in, and FSR0L specifies the address within the bank. Thus, FSR0 is initially loaded to 0x200, equivalently the start of bank 2. To move through the block of memory, POSTINC0 is used, which points FSR0 to its next file register.

In the record function, the rollover number, now stored in the ***freq_rollover*** variable, is moved to POSTINC0. In the limit of exceeding bank 14, FSR0 is loaded back to the start of bank 2, hence rewriting the stored data. Conversely, in the replay function, the rollover number in POSTINC0 is moved to ***freq_rollover*** for synthesis, starting from bank 2. If neither of these commands is requested, the interrupt flag is immediately cleared, which returns the program back to the main loop. When an ISR0 takes place, 0xFF is moved to TMR0H and the rollover number in ***freq_rollover*** is moved to TMR0L, meaning the timer will be counting from 0xFFxx (xx is an 8-bit hexadecimal value of the rollover number) to 0xFFFF. Essentially, this changes the frequency of which the output state is alternated and hence the frequency of our final square wave output.

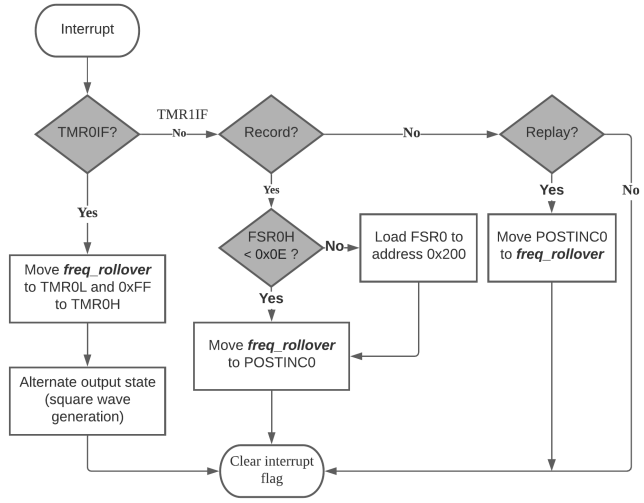

Figure 6. A flowchart that illustrates the low-level software design of the interrupt service routines ISR0 and ISR1. ***freq_rollover*** is a variable that holds the rollover number of a note.

B. Hardware

The low-level hardware design of the touchscreen piano is depicted by its circuit diagram (see Figure 7).

1) Touchscreen: The model of our touchscreen is WDG0151-TMI-V#N00. In particular, the 128x64 GLCD and touch panel are separately powered and have separate data buses which are interfaced to different ports of the microprocessor.

2) Microprocessor: We are using an 8-bit 64 MHz PIC18F87K22 microprocessor. The overall length of our program code is 1.494 kB, which is permanently stored in the program memory. Amongst it, ≈ 0.26 kB is responsible for the GLCD tables. Having 13 banks in the internal SRAM available for temporary storage, we can store a maximum amount of 3328 rollover numbers (each an 8-bit). This means the longest recording can last up to 109.82 s, which is sufficient for the purpose of our prototype. As a note, RJ0 is used to output the square wave signal and is interfaced with the second order active low pass filter. The five control commands, discussed in the Input Reading subsections, can be enabled via RC0:4 with a push of the port switch.

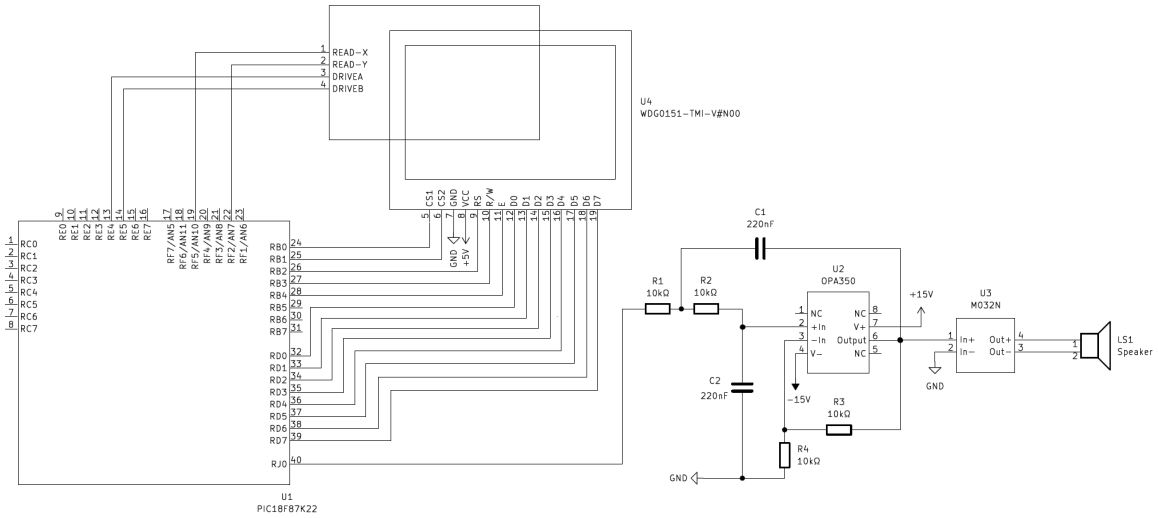

Figure 7. A circuit diagram of the touchscreen piano. Note the touch panel and GLCD were depicted as two separate compartments in the touchscreen module.

3) Audio Synthesizer: The operational amplifier we used in the second order active low pass filter is an OPA350. The two RC circuits in the filter were chosen with equal values of resistance and capacitance. This results in a cutoff frequency f_c of ≈ 72 Hz since [3]

$$f_c = \frac{1}{2\pi\sqrt{R_1 R_2 C_1 C_2}}, \tag{2}$$

which was found effective at reducing the square wave to a sinusoid. The choice of using this filter will be justified in the Results & Performance section. Finally, a 12 W Kemo M032N audio amplifier was used to drive up the weak electric signals to an audible level.

IV. Results & Performance

Qualitatively, the touchscreen piano runs smoothly without any noticeable delay. It is able to synthesize frequencies accurately within an acceptable level of deviation from the actual note. Unlike our expectations, the quality of sound is fairly mechanical and muffled. Given some distance away from a keyboard boundary, we are able to consistently resolve the correct key pressed. Yet approaching closer to the boundary can lead to a shivering sound effect.

A. Frequency synthesis

Our original attempt with a third order passive filter resulted in a smooth sine wave yet with highly attenuated amplitude. This significantly diminishes the sound quality because the signal is now prone to white noise from the protoboard. To tackle this issue, we instead used an active filter that is externally driven and does not consume power from the circuit, unlike the passive one.

Mathematically, a square wave is a Fourier series. To obtain a single-frequency, our goal is to keep the first harmonic and minimize all higher frequency components of our synthetic output. The frequency response graph (see Figure 8) of our circuit shows that the idea has been successfully implemented from its increasing attenuation of higher frequencies. However, our choice of a small f_c also resulted in a large attenuation of the fundamental frequencies between C note and G♯ note at the same time. Immediately, our implementation can be improved by shifting f_c to the G♯ note (440 Hz), such that the signal within our synthesis range is unaffected and decreases rapidly beyond. The final waveform of the output after passing

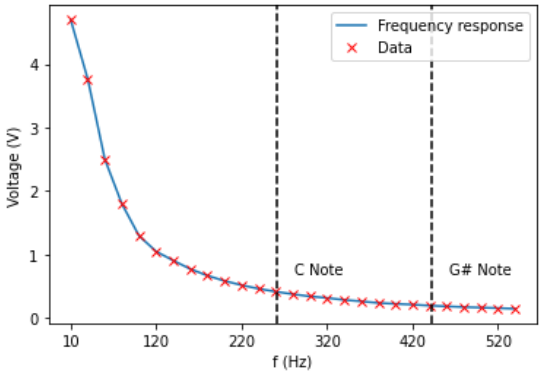

Figure 8. A frequency response graph of the second order active low pass filter. The frequency range of our output is between 261 Hz and 440 Hz, which is indicated by the two dashed lines. Data points were sampled by measuring the voltage of different sinusoids generated by the oscilloscope.

through the filter (see Figure 9) resembled a sinusoidal shape and can be concluded to be almost single-frequency by eye. A Fourier transform on the low pass filter signal and square wave signal was performed to allow quantitative analysis. We measured an 85% attenuation of the second harmonic, thereby confirming higher order components are sufficiently reduced. Overall, the output frequencies were measured to have an average deviation of ± 1.16 Hz from their true value, after discounting the contribution from one of the notes, F♯, which was believed to be an outlier due to an incorrectly assigned rollover number.

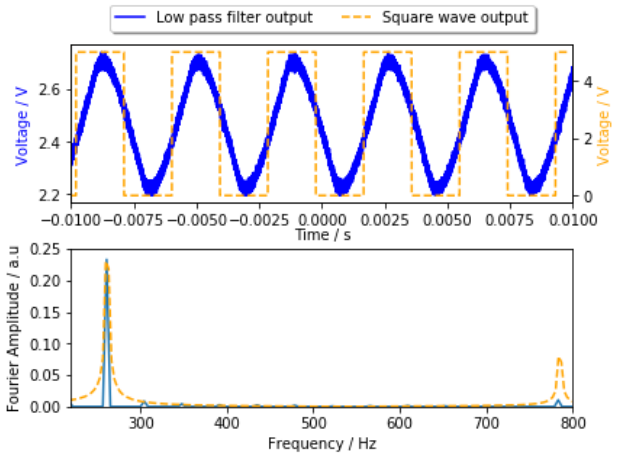

Figure 9. A graph that compares the square wave output of C note and that after passing through the second order active low pass filter. Top: the square wave signal (orange) and low pass filter signal (blue) in the time domain. Bottom: the associated frequency spectra after Fourier transform. The fast Fourier transform algorithm was implemented on the datasets for this purpose. Note also appropriate scaling of the Fourier coefficients were done to allow for a comparison.

B. *Touchscreen precision*

Previously during calibration, we measured the voltage output of the touchscreen systematically along its x-axis at every eight pixels, which showed an approximately linear dependence with distance. However, the touchscreen can give non-uniform results, for example, when probing coordinates at the same x but different y. Therefore, a linear calibration is not ideal. Nonetheless, the linear proportionality was measured to be 0.82 ± 0.05 pixels per voltage level which is equivalent to one level occupying 1.22 pixels.

We investigated the coverage of the shivering effect by sampling the distance from the individual boundary to its furthest pixel in which the effect still persisted, and found an average error of ± 4.32 pixels. This result explains why a narrower key is more likely to synthesize a wrong output, as the error on either side of its boundary combines to essentially cover the size of the entire key. This effect arises due to the fluctuation in the ADC reading as the touchscreen is very sensitive to the pressure applied and touch.

V. Updates, Modification & Improvements

A. *Touchscreen calibration*

To minimise the shivering effect on the software end, we can impose dead zones along each keyboard boundary so that no sound is made when we touch near these regions.

B. *Signal processing*

Instead of a low pass filter, it is better to use a band pass filter with a bandwidth between our output frequency range (261 Hz - 440 Hz). With this, we can retain the amplitude of a range of desired frequencies, unlike the previous case where only frequencies before f_c are kept. An even better alternative is using a digital-to-analog converter (DAC). A DAC allows us to synthesize any analog waveform by referencing values directly from a lookup table. This technique was trialled but not fully implemented. Again, we attempted to produce a C note with an 8-bit DAC this time. The waveform appears the most sinusoidal looking amongst others and contains only the fundamental frequency component in its Fourier transform.

Additionally, in reality, sound waves produced by all musical instruments, including the piano, are composed of infinitely many harmonic modes. This means that the sound quality can be improved by generating an analog output of the actual waveform from a piano [4]. We can do this by summing the function values of various harmonic modes on a computer and then loading those values onto a lookup table for synthesis.

VI. Conclusions

In this paper, we demonstrated the implementation of a 9-key touchscreen piano on an 8-bit microprocessor. Useful functions such as record and replay are added to facilitate user experience. Both the software and hardware design are discussed in detail. The device performs consistently and accurately on the whole. Alternative methods of signal processing were compared, in which we propose continuing with a DAC in order to generate the true waveform from a piano for maximum sound quality.

VII. Product Specifications

Table I

	Description
Keyboard	9 keys (C4 - G#4)
Pitch Range	(261.63 - 440.00) Hz $\pm$ 1.16 Hz
Features	Note synthesis, record, replay, clear
Sampling Period	0.033 s
Program Space	1.494 kB
Recording Space	3.315 kB (maximum 109.82 s)
Processor Type	PIC18F87K22
Display	128x64 pixels
External Devices	• Touchscreen: WDG0151-TMI-V#N00 • Operational-amplifier: OPA350 • Audio amplifier: M032N • Unbranded speaker

References

[1] n.d. How it works: 4-Wire Analog-Resistive Touch Screens. HantouchUSA.

[2] 2018. PIC18F87K22 Family Datasheet. Microchip Technology Inc., p.343.

[3] n.d. Second Order Low Pass Filter. [online] Available at: https://www.electronics-tutorials.ws/filter/second-order-filters.html [Accessed 28 December 2021].

[4] People.carleton.edu. 2012. MUSC 101 Unit 1 Sound Basics. [online] Available at: https://people.carleton.edu/~jellinge/m101s12/Pages/01/01SoundBasics.html [Accessed 29 December 2021].

APPENDIX

A. Cycle 1 feedback

Excellent data analysis technique, from data processing to uncertainties Data could be presented in better format to give an overall final result with more impact to the audience. Physics could be better presented, with more clarity. Overall well done. Good slides, well formatted. Could be more confident in the discussion and have a better understanding of the fundamentals.

B. Cycle 2 feedback

The proposal lacks some technical details. It is not clear how output sample rate is calculated and how many samples per cycles will be used. Organization of look-up tables and their content could be described in more detail. E.g. Do you store Sin values in one of the look up table? Consider storing key sequences instead of actual generated samples when using 'record' function, since samples are generated by your own program.