HÁSKÓLINN Í REYKJAVÍK
REYKJAVIK UNIVERSITY

Fall 2019

SE-T-535-MECH, Mechatronics II

Final Report

Þórður Örn Stefánsson
kt. 261096-2269

March 28, 2019

Instructor: Baldur Þorgilsson

Contents

1 Introduction **2**
1.1 Short introduction to music theory 2
1.2 Customer Needs 2
1.3 Functional requirements 2

2 Background **3**

3 Design **4**
3.1 Design Parameters 4
3.2 Design Process 4
3.2.1 Hardware 4
3.2.2 Software 5
3.3 Design matrix 6
3.4 System diagram 7
3.5 Bill of Materials 7

4 Testing **7**

5 Improvement possibilities **8**

6 Conclusion **8**

A Source Code **11**

B Circuit Schematic **14**

1 Introduction

This project is a final project in the course Mechatronics II. The aim is to make a device that detects the frequency of a sound, and displays which note is being played.

The motivation for doing this project is that I have plenty of interest in music and music theory, and like to play the piano in my freetime.

The axiomatic design methodology was used to develop customer needs and analyze how it can be transformed into functional requirements and design parameters. The axiomatic design is a way to break projects down in smaller parts to get a better overview of the project. The target customers are musicians, mainly guitarists.

The project was coded in *C* and the microcontroller that used was a RedBoard, made by SparkFun Electronics. On the next few pages, the software and hardware will be described, followed by some measurement and performance results and discussions.

1.1 Short introduction to music theory

In music, the different sounds that are played are called notes. Each note has its very own frequency. The frequencies are determined from the fundamental frequency of the middle A, which has a frequency of 440Hz. In each octave the frequency is doubled, so the A above the middle A has a frequency of 880Hz. Similarly, the A below the middle A has a frequency of 220Hz. An octave consists of 12 half-tones, so going n half-tones away from the middle A results in a note with a frequency of

$$440Hz \cdot 2^{n/12}$$

These frequency thoughts will be the main part in the algorithm to determine the pitch that is being played.

1.2 Customer Needs

A list of customer needs for the project was developed:

- $\mathbf{CN_0}$ A small, portable device that detects frequency and indicates which note is being played.
- $\mathbf{CN_1}$ A device that detects frequency.
- $\mathbf{CN_2}$ The device tells the user which note is being played.
- $\mathbf{CN_3}$ The device has a rechargeable battery to make it more portable.

1.3 Functional requirements

To satisfy the customer needs, the functional requirements can be constructed:

- $\mathbf{FR_0}$ Detect frequency and display the pitch, while being portable and easy to use.

- $\mathbf{FR_1}$ Detect frequency.
- $\mathbf{FR_2}$ Display pitch with LEDs
- $\mathbf{FR_3}$ Be portable.

2 Background

The first instrument to detect pitch was the tuning fork, invented by the British musician John Shore in 1711 [1]. Each pitch fork has its own specific pitch at which it resonates when struck against some surface, and generates a simple sine wave. Electronic tuners can be bought cheaply at the next musical instruments store. The cheapest ones can only detect a small number of pitches, often the six pitches on a guitar, but more complex and expensive ones can detect all the 12 notes in the octave. They usually use a microprocessor to calculate the average frequency of the wave. Some have a needle to indicate the note being played, others have a LED or an LCD screen to display the note. Some tuners are designed to be clipped on the guitar head and have a built-in vibration sensor. There also exist apps that can detect pitches and can be downloaded to smartphones.

Figure 1: A tuning fork. This one has the specific pitch A [2]

Figure 2: A clip on guitar tuner [3]

Figure 3: A needle guitar tuner [4]

The most expensive and accurate ones are strobe tuners. They have rotating discs and lamps or LED that flash at the same frequency as the input signal. A strobe tuner can be up to 30 times more accurate than a good electronic tuner. Cheap clip-on guitar tuners cost around 1500ISK, while the expensive probe tuners may cost up to 15-20.000ISK [5].

What would make this one better than those that currently exist is that it will be more accurate than the cheap ones, and cheaper than the expensive ones.

Some similar projects have been done with a microcontroller and can easily be found on the internet. In [6] is an Arduino Guitar tuner. It has an audio jack to connect to a guitar for picking up the sound and utilizes a low-pass filter to filter out undesirable frequencies caused by noise. The microcontroller lights up an LED strip depending on the frequency.

3 Design

3.1 Design Parameters

The design parameters are meant to

- **DP_0** A device with a frequency detection and indication controlled by a simple microcontroller.
- **DP_1** A frequency detector
- **DP_2** LED bulbs
- **DP_3** A rechargeable battery

3.2 Design Process

3.2.1 Hardware

In the beginning of the project, the project hardware was planned. How was it going to work? How could a sound signal be processed in order to find its frequency?

Firstly, something was needed to pick up the sound and generate electric signal depending on the sound characteristics. To do this, an electret microphone is usually used. The signal from them

is very small and thus needs to be amplified. Some electronics retailers offer electret microphone breakout boards with an on-board amplifier. One from SparkFun was eventually ordered and used. Since the signals from these boards can be hard to work with, a signal generator was used at first to emulate a signal from an electret microphone.

The options to process the signal were for example to recreate the signal as a digital signal, use a frequency to voltage converter, or try to convert the signal to high and low box signal and use the external interrupt pin on the microcontroller. The first option was quickly crossed out because it was more work than the other options and wouldn't give any better results. The choice was then between a frequency to voltage converter or converting the signal to a binary signal and counting the clock cycles of the period. In the end, it was decided to go with the latter because it is easier to implement and the components needed did exist in the lab.

The RedBoard does have an external interrupt pin on it that generates an interrupt request when its state changes. To utilize this, the sound signal needed to be converted to a digital binary signal. A Schmitt trigger is ideal for this task. It has an upper and a lower threshold, and when the input voltage rises above the upper threshold, the trigger returns a high, and once it goes below the lower threshold it returns a low.

The Schmitt trigger chosen was an MM74C14N. It typically has upper threshold of 3.6V and lower threshold of 1.4V [7]. The output voltage acquired from the electret microphone, however, is in the region of millivolts, typically around 200mV. This means it needs to be amplified for it to be high enough for the Schmitt trigger. An LM324 operational amplifier was picked for this part. Since only the frequency of the signal and not the amplitude is of interest here, the op-amp was driven to saturation, and the output signal fed to the Schmitt trigger. At last the output from the Schmitt trigger was connected to the external interrupt pin on the RedBoard (pin 3). A signal of 473mV, 1kHz from the signal generator was fed to the system. The input signal and the output from the Schmitt trigger can be seen in figure 4.
The final circuit schematic can be seen in appendix B.

3.2.2 Software

The frequency of a signal tells how many cycles happen over one second. The reciprocal of a signals frequency is its period, that is how long does one cycle take. As mentioned in the hardware section, the external interrupt pin prompts an interrupt when a logical change is detected. The pin can be set to request interrupt on a rising edge, falling edge, any change or when it is low or high. Here it is suitable to get an interrupt on a rising edge because that is the time of one period.

The main idea with the code is simply to count the time of one period. The 8 bit timer/counter register (TCCR0B) is set up in the beginning of the code with the lines

```
TCCR0B |= (1 << CS00);          //no prescale
TCNT0 = 0;                      //set up timer/counter
TIMSK0 |= (1 << TOIE0);
```

Because the TCCR0B register is an 8-bit register, it goes to 255 before it overflows. The clock speed of the processor is 16MHz, so the time of one overflow is:

$$t = 255 \cdot \frac{1}{16MHz} = 15.94\mu s$$

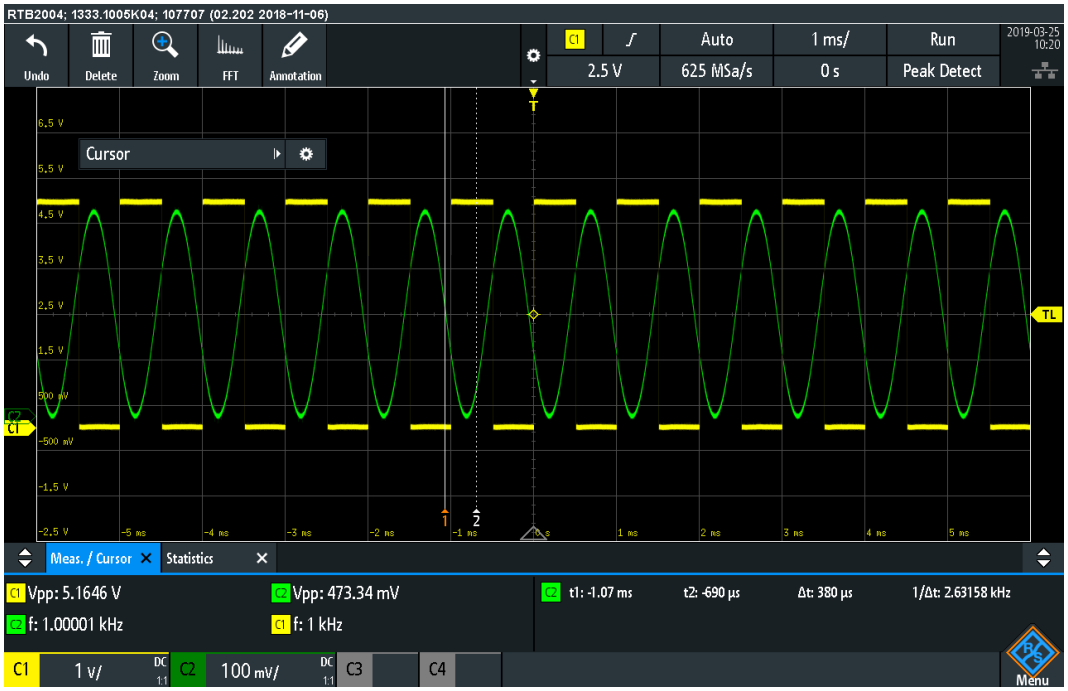

Figure 4: The input signal (green) and the output from the Schmitt trigger (yellow). The signals have the same frequency but the amplitude of the yellow signal is higher.

When the timer/counter register overflows, it generates an interrupt where the unsigned long variable *ovfCount* is incremented. This value is then checked in the pin change interrupt. Let's take an example of the middle A note. It has a frequency of 440Hz, so its period is:

$$T = \frac{1}{440Hz} = 2.27ms$$

So the count of the overflows needed is:

$$n = \frac{2.27ms}{15.94\mu s} = 142$$

When it is shifted by 8 bits, the value ends up being:

$$142 \cdot 2^8 = 36456$$

When the INT0 external interrupt pin goes from low to high, an ISR is entered. Within this ISR, the timer value shifted by 8 bits is stored and some if-statements check in which range it is. Depending on that, the program can decide which note is being picked up by the microphone and lights up the corresponding LED.

The final edition of the source code in whole can be seen in appendix A.

3.3 Design matrix

The mapping from the functional requirements to the design parameters is shown on table 1

Table 1: FR-DP mapping

ID	Functional Requirement	Design Parameter
1	Detect Frequency	A frequency detector
2	Display pitch	LEDs
3	Be portable	A rechargeable battery

The design matrix in equation 1 shows how the functional requirements are related to the design parameters.

$$\begin{Bmatrix} FR_1 \\ FR_2 \\ FR_3 \end{Bmatrix} = \begin{bmatrix} X & O & O \\ X & X & O \\ O & O & X \end{bmatrix} \begin{Bmatrix} DP_1 \\ DP_2 \\ DP_3 \end{Bmatrix} \tag{1}$$

The X below the diagonal line means that if DP_1, a frequency detector, fails, it will cause FR_2, to display on an LCD, to fail also. This means that the design is coupled, which is not ideal.

3.4 System diagram

Figure 5 shows how the system is supposed to work.

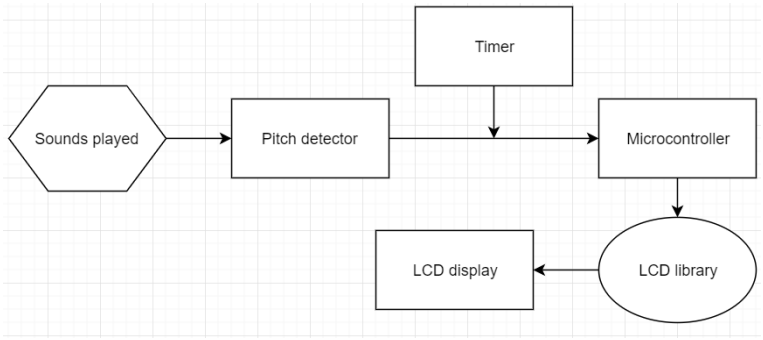

Figure 5: System diagram

3.5 Bill of Materials

The following table contains the bill of materials:

4 Testing

Once the device had been assembled on a breadboard, it was time for the testing to begin. The input was first connected to a signal generator and sine waves with different frequencies were fed to the system. The device was able to correctly detect every frequency of the waves that were made. Then it was time to test the system with a microphone. Again, the device displayed the pitch correctly, but the main problem is that the source of the sound had to be right next to the microphone for it to detect the correct pitch.

Table 2: Bill of Materials (*MBR = Miðbæjarradíó)

Vendor	**Model**	**Description**	**Price each (ISK)**	**Quantity**	**Total Price (ISK)**
SparkFun	RedBoard	A microcontroller	2.440	1	2.440
Adafruit	LCD display	16x2 LCD display	1.700	1	1.700
ebay	MM74C14N	Schmitt trigger	260	1	260
Digi-Key	LM324N	Op-amp	67	1	67
MBR*	LED	LED bulbs	444	2×10	888
MBR	Resistor	10kΩ	48	1	48
MBR	Resistor	220Ω	30	13	390
Grand Total					**5.739**

5 Improvement possibilities

Although the project does work, there is always plenty of room for improvement.

For example, the sound detection could be better. Although the electret microphone breakout board does work, it only detects the sound at a good quality when the sound source is within centimeters from the microphone. It is preferable to be able to place the device a bit from the instrument that is played on.

Another field to improve is how the pitch is displayed. Currently it is done with 12 LEDs, one for each pitch in the octave. At the beginning of the project, the intention was to use an LCD display to do this. It finally proved to be too time-consuming and the focus had to be put on other tasks. That being said, it is not easy to say if it would be a straight improvement. The current way with the LEDs is quite informative and the LCD is more expensive and can flicker a bit, so maybe using the LEDs is simply a good enough solution.

Another obvious improvement is in terms of powering the device. In the current state it is powered via a USB cable or a 9-12V DC adapter to the onboard barrel jack. The dream was to have a rechargeable battery to power the system, but due to time pressure it couldn't be done. A rechargeable battery would make the device much more portable and greatly improve its usability.

In this version of the code, only the middle A and the octave above it can be detected. This is simply because only their periods have been installed. It would be very simple to add more octaves to the code, but for simplicity it is now only with the single octave.

6 Conclusion

To determine if the project was a success or a failure, let's take a look at the functional requirements proposed earlier:

- **FR_0** Detect frequency and display the pitch, while being portable and easy to use.

- **FR_1** Detect frequency.
- **FR_2** Display pitch with LEDs.
- **FR_3** Be portable.

At a glance, FR_1 and FR_2 have been fulfilled, but FR_3 hasn't. The device is able to detect frequency and display the pitch, and is easy to use, although it is not as portable as originally intended, so FR_0 is almost satisfied. Despite the lack of portability, the project will have to be counted to having been a success.

References

[1] H. Feldmann, *History of the tuning fork. I: Invention of the tuning fork, its course in music and natural sciences. Pictures from the history of otorhinolaryngology, presented by instruments from the collection of the Ingolstadt German Medical History Museum.* Howard Piano Industries, 1997.

[2] Wittner, *Wittner Piano Tuning Fork - A440.* Howard Piano Industries. [Online]. Available: https://www.howardpianoindustries.com/wittner-piano-tuning-fork-a440-cps/

[3] T. Music, *Tiger Chromatic Guitar Tuner.* Tiger Music. [Online]. Available: https://www.amazon.co.uk/Tiger-Chromatic-Guitar-Tuner-Clip/dp/B002IRTZWM

[4] *Boss TU-6 Guitar Tuner.* audiofanzine. [Online]. Available: https://en.audiofanzine.com/guitar-tuner/boss/tu-6/user_reviews/r.39635.html

[5] Hljóðfærahúsið, *Stillitæki og taktmælar.* Hljóðfærahúsið. [Online]. Available: https://www.hljodfaerahusid.is/is/vefverslun/fylgihlutir-hljodfaera/stillitaeki-og-taktmaelar?sort=price-asc&page=2

[6] WyeItHertz. (2017) Arduino guitar pitch detection. [Online]. Available: https://www.instructables.com/id/Arduino-Guitar-Pitch-Detection/

[7] *Hex Schmitt Trigger*, Fairchild Semiconductors, 10 1987, rev. 2002.

Appendices

A Source Code

```
#include <avr/io.h>
#include <avr/interrupt.h>

volatile unsigned long ovfCount;

int main()
{
        ovfCount = 0;

        TCCR0B |= (1 << CS00);          //no prescale
        TCNT0 = 0;                      //set up timer/counter
        TIMSK0 |= (1 << TOIE0);

        EICRA &= ~3;
        EICRA |= (1 << ISC00) | (1 << ISC01);   // set INT0 to trigger on
            rising edge
        EIMSK |= (1 << INT0);   // enable external interrupt 0

        DDRB = 0x20;    // set PORTB pins as output
        DDRC = 0x20;    // set PORTC pins as output

        sei();  // enable external interrupts

        while(1);

        return 0;
}

ISR(INT0_vect)
{
        unsigned long l = (ovfCount<<8)+TCNT0;

        if((l > 35720 && l < 37000))            // A
        {
                PORTB = 0x00;
                PORTC = 0x00;
                PORTB |= (1<<PB0);
        }
        else if((l > 33368 && l < 35320))       // A#
        {
                PORTB = 0x00;
                PORTC = 0x00;
                PORTB |= (1<<PB1);
        }
        else if((l > 31496 && l < 33368))       // B
        {
                PORTB = 0x00;
                PORTC = 0x00;
                PORTB |= (1 << PB2);
        }
        else if((l > 29739 && l < 31496))       // C
        {
```

```
                PORTB = 0x00;
                PORTC = 0x00;
                PORTB |= (1 << PB3);
        }
        else if((l > 28068 && l < 29739))       // C#
        {
                PORTB = 0x00;
                PORTC = 0x00;
                PORTB |= (1 << PB4);
        }
        else if((l > 26490 && l < 28068))       // D
        {
                PORTB = 0x00;
                PORTC = 0x00;
                PORTB |= (1 << PB5);
        }
        else if((l > 25001 && l < 26490))       // D#
        {
                PORTB = 0x00;
                PORTC = 0x00;
                PORTC |= (1 << PC5);
        }
        else if((l > 23600 && l < 25001))       // E
        {
                PORTB = 0x00;
                PORTC = 0x00;
                PORTC |= (1 << PC4);
        }
        else if((l > 22271 && l < 23600))       // F
        {
                PORTB = 0x00;
                PORTC = 0x00;
                PORTC |= (1 << PC3);
        }
        else if((l > 21014 && l < 22071))       // F#
        {
                PORTB = 0x00;
                PORTC = 0x00;
                PORTC |= (1 << PC2);
        }
        else if((l > 19842 && l < 21014))       // G
        {
                PORTB = 0x00;
                PORTC = 0x00;
                PORTC |= (1 << PC1);
        }
        else if((l > 18650 && l < 19842))       // G#
        {
                PORTB = 0x00;
                PORTC = 0x00;
                PORTC |= (1 << PC0);
        }

        ovfCount = 0;   // reset overflow counter
}

ISR(TIMER0_OVF_vect)
```

```
{
        ovfCount++;      //increment overflow counter
}
```

B Circuit Schematic