A Microchip PIC-based Ukulele Tuner

Son Phan*
Department of Electrical and Computer Engineering
Cleveland State University – EEC 517
Cleveland, Ohio 44115

May 2, 2014

*Email: s.phan82@vikes.csuohio.edu

Abstract

This paper describes in detail the design of an electronic ukulele tuner that uses the Microchip's PIC [1] microcontroller, an electret microphone, and a simple LED display interface. The filtering and tuning solution of the tuner is mostly done in software. The nature of the project is "learning by doing" and is mandatory at the graduate level of the Embedded Systems course, EEC 517, at Cleveland State University. The project is released open-source on GitHub [2].

1. Introduction

The tuner is an essential device for musicians to help their instruments sound correctly. This is a standard 4-string soprano ukulele tuner that tunes the following 4 notes at their respective frequencies: G (392 Hz), C (262 Hz), E (330 Hz), A (440 Hz). The electronic tuner includes a microphone to listen to sound inputs (ideally from a ukulele string), a LED segment display to show one of four notes (G, C, E, A), and a set of red/green LEDs to guide the pitch if it is low, high, or correct. At the core, the tuner uses a microcontroller, coupled with amplifying and filtering circuitries, to process the sound inputs, do the calculations, and to output the results.

Although there has already been plenty of tuners for musical instruments available on the market, it is always fun to build one yourself. The project starts with the core concepts learnt from the embedded systems course such as the PIC16F877 microcontroller, its timer modules, LED segment display, etc., and expands to utilizing advance peripheral such as the microphone and to high-level C programming language, thus helps reinforce the knowledge acquired.

The most important feature of the tuner, which also considered the most challenging, is indeed the tuning capability. Although there are many ways this can be done either purely by hardware, or software, or both, the goal of the project is to focus on simplicity, manageable timeframe, and minimum budget required. In the initial proposal, the tuner's hardware was to include a band-pass filter circuit to filter out the undesired frequencies. However, after assessing information found in the "Microchip Based Automatic Guitar Tuner [3]" poster, it is possible that the filtering and tuning process can be done in the PIC's software, as a result minimizes the amount of hardware and makes it easier to troubleshoot. The mentioned "Automatic Guitar Tuner", however, "used a 1/4" input jack, instead of a microphone. With a microphone, background sounds could cause a false frequency reading, resulting in an inaccurate rotation of the motor [3]".

In the continuing sections, this paper provides a detail look at the hardware of the Microchip PIC-based Ukulele Tuner in Section 2, its software implementation in Section 3, and conclusion with future work follows in Section 4. Finally, the paper ends with the Reference section and source code listing, which is placed in the Appendix.

2. Hardware

2.1. Overview

The tuner's primary components consist of a Microchip PIC16F877 (PIC) 8-bit microcontroller [1], a sound recording unit, an amplifier circuit, a signal processing unit, and a user interface with LED segment display and 2 red, 1 green LEDs (inspired by the GuitarToolkit iOS app [4]). Figure 1 illustrates how these components connect to each other and Figure 2 shows where they are located in the tuner. Table 1 lists the components utilized in the project.

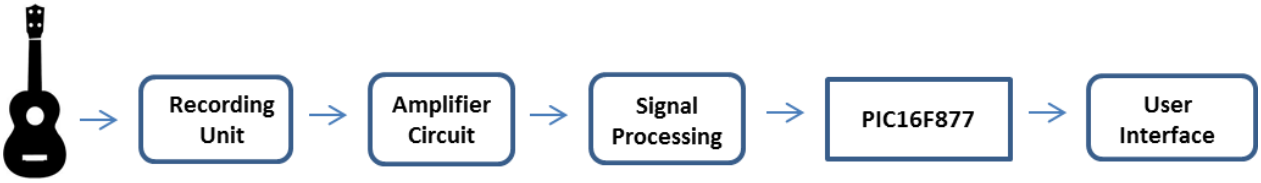

Figure 1 – Tuner Design Flowchart

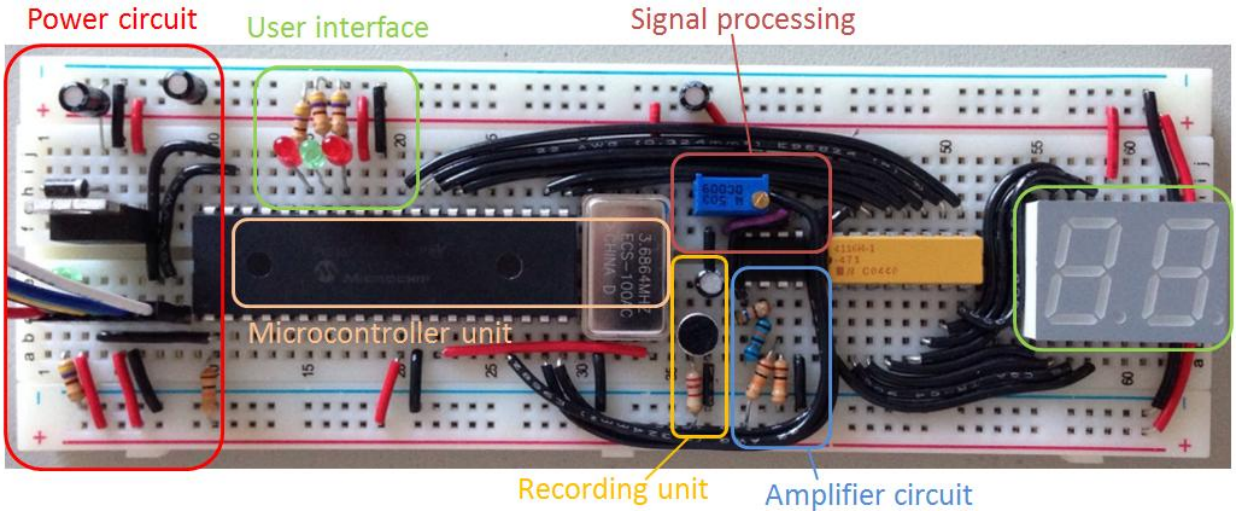

Figure 2 – Tuner's Hardware Setup

Part	Size / Part No.	Qty.
Microchip microcontroller	PIC16F877	1
Crystal oscillator	3.6864 MHz	1
Voltage regulator	7805	1
Panasonic Microphone	WM-64PNT	1
Operational amplifier	LM358	1
7-segment LED display	HDSP-521E	1
LED	Red	2

LED	Green	1
Capacitor	47 µF	1
Capacitor	1 µF	1
Capacitor	0.1 µF	2
Resistor	680 KΩ	1
Resistor	10 KΩ	2
Resistor	560 Ω	1
Resistor	470 Ω	2
Resistor	2.2 KΩ	1
Array of 8 resistors (DIP 16)	470 Ω	1

Table 1 – Parts List

2.2. Recording unit

Most typical electret microphones should be able to accommodate the frequencies of the musical notes. The Panasonic's microphone – model WM-64PNT [5] was chosen due to its small form factor as specified in Table 1. To meet these specifications, the microphone was wired exactly as illustrated in the schematic diagram in Figure 3.

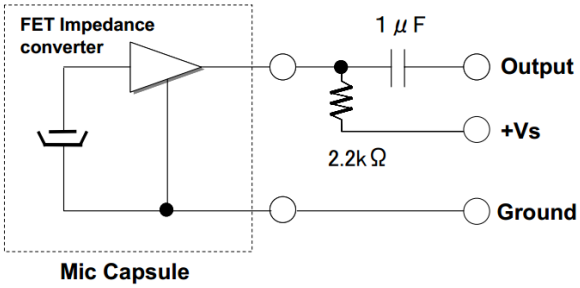

Figure 3 – Panasonic Microphone Schematic Diagram [5]

Specifications

Dimensions	6 x 2.2 mm
Impedance	Less than 2.2k Ω
Directivity	Omnidirectional
Frequency	20–16,000Hz
Max operation voltage	10V
Standard operation voltage	2V
Current consumption	Max 0.5mA
Sensitivity reduction	Within −3dB at 1.5V
S/N ratio	More than 58dB

Table 2 – Panasonic Microphone Specifications [5]

2.3. Amplifier circuit

The LM358 is a standard low-power dual operational amplifier (op-amp) that is commonly used and affordable. The circuit was implemented as in Figure 4, which used one op-amp of the LM358 (originally built and tested with a gain of "50-time setup" that yielded "the best noise vs. output swing results [6]"). In this circuit, the sound signal was amplified with a gain ratio of 1213 (680KΩ / 560Ω).

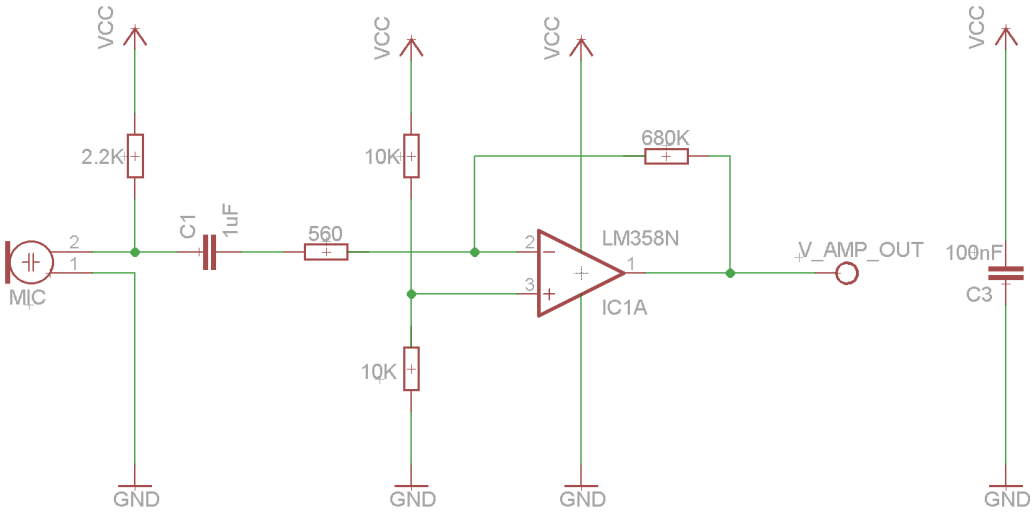

Figure 4 – LM358 Amplifier Circuit Diagram [6]

2.4. Signal processing

The second op-amp of the LM358 was used as a comparator to convert the amplified analog sound signal into the preferable square-wave output signals. Figure 5 illustrates how the input amplified voltage, which was between 0 – 3.4V, was designed to be clipped at a reference voltage (chosen to be 1V) to eliminate all harmonic created by the ukulele's acoustic string. A potentiometer was used to create the 1V reference voltage.

An example of musical note E signal conversion at 330Hz is shown in Figure 6.

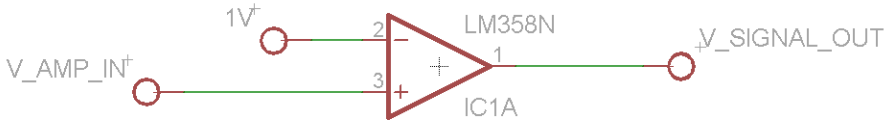

Figure 5 – LM358 Comparator Diagram

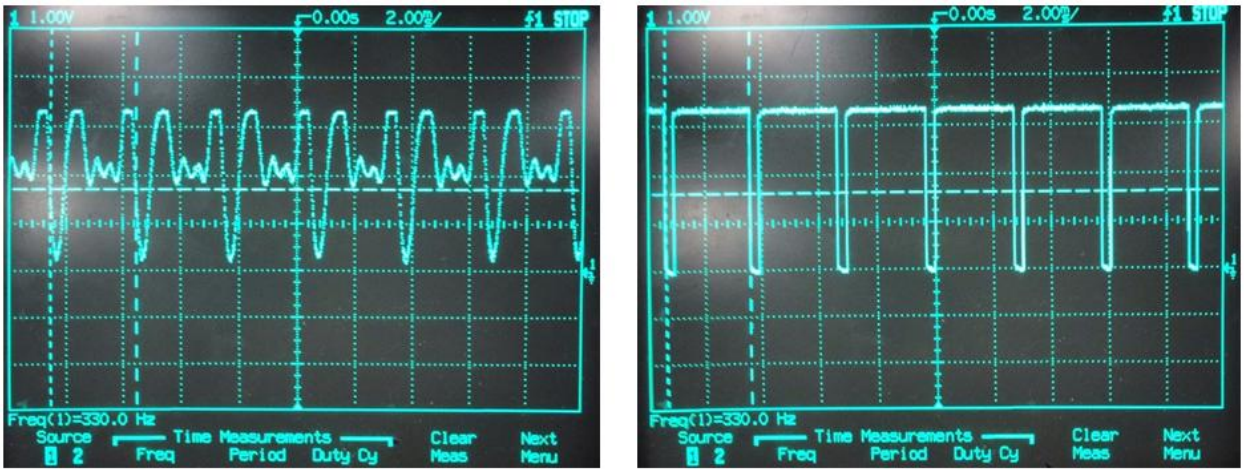

Figure 6 – LM358 Comparator Signal Input vs. Output

2.5. PIC

To compute the sound frequency using the PIC, the square-wave signal was fed into the RC2/CCP1 pin (pin 17) that generates interrupts on edge detection. CCP1CON register was configured for capture mode, in which "CCPR1H:CCPR1L captures the 16-bit value of the [Timer1] (TMR1) register when an event occurs on pin RC2/CCP1 [1, page 59];" for the project, the event was defined as every rising edge. The PIC program then calculated the sound signal frequency directly to figure out the associated pitch, and display the LED segments representing the letters G, C, E, A with the pitch LED.

Figure 7 illustrates how all components were connected to the PIC.

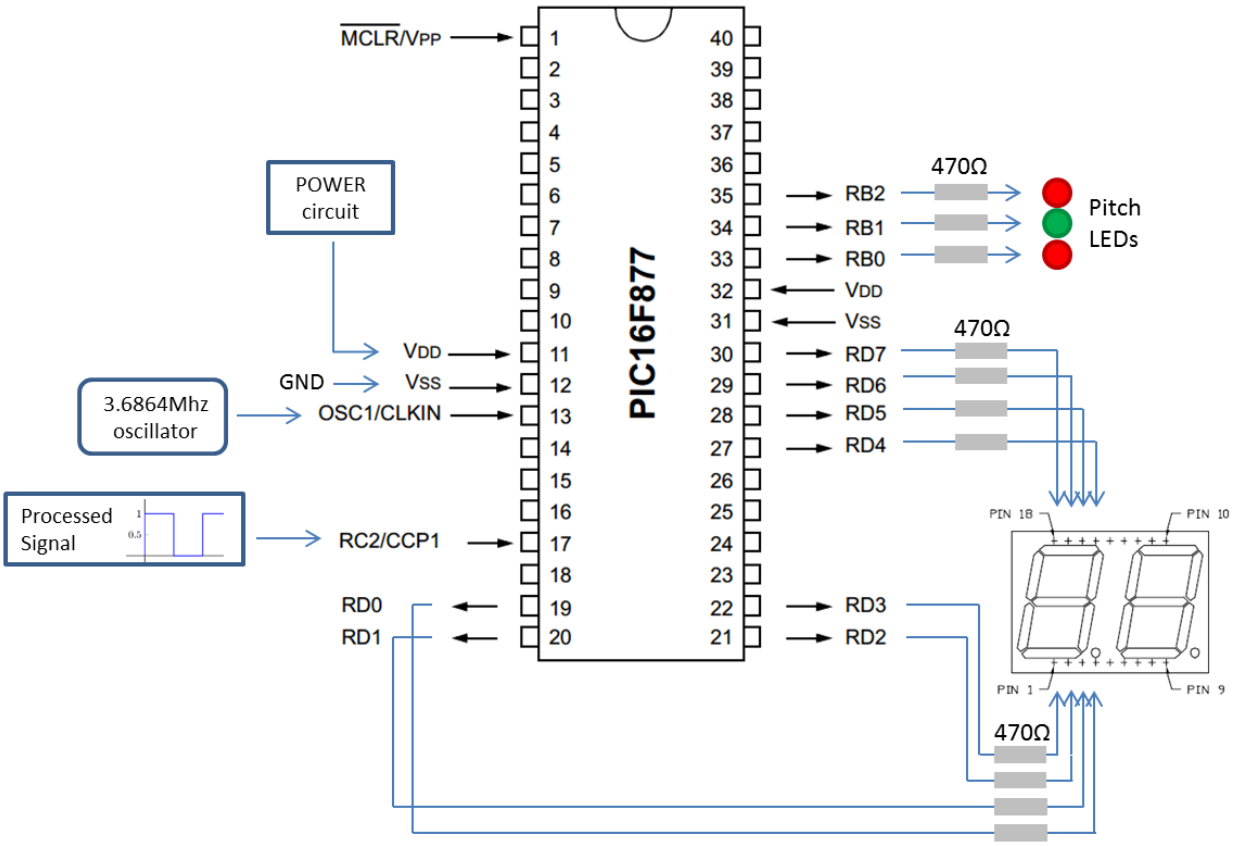

Figure 7 – Microchip PIC Ukulele Tuner Diagram

2.6. User Interface

The electronic ukulele tuner when powered on will listen to sounds from its surrounding and display on the 7-*segment LED display* one of the letters G, C, E, or A, which is the closest note according to the pitch of the sound, and also turn on one of the LEDs. The *red* LED on the *left* indicates the pitch being lower than the displaying note, *right red* LED indicates pitch being higher, and *middle green* LED indicates tuning correctly (Figure 8). It will be required to keep the noise level at the minimum while the ukulele is being strummed, one string at a time, to achieve the expected results.

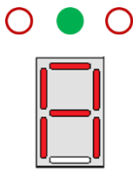

Figure 8 – Ukulele Tuner Interface

3. Software

The tuner's software was developed in C language with MPLAB IDE version 8.92. HI-TECH C Compiler [7] was also used to help alleviate the complexity of the floating point calculation of the frequency.

There are three core processes: initialization, frequency calculation, and filtering/tuning, that are executed in such order. For initialization, a certain constant values had to be defined, such as the musical notes' frequencies, the operating clock speed, timer value, etc. Also the PIC's registers and its ports/pins that being used as inputs/outputs had to be configured, such as the INTCON register was configured to enable Global (GIE) and Peripheral (PEIE) interrupts, CCP1CON was set to capture mode at every rising edge, TRISC <RC2> pin was set to be input for processed sound signal, etc.

For frequency calculation, the CCP1 interrupt flag (CCP1IF) is constantly checked, when set, the Timer1 16-bit integer value, which stored in CCPR1H:CCPR1L, is captured and calculated. After verifying the captured value is valid, in which the new timer value "capnew" has to be larger than the old value "capold", the difference between "capnew" and "capold" yields the period (the amount of time ran) between captures, as Timer1's continuously counting. For more than every rising edge, this value would have to be divided by the number of edges that occurred, configured in CCP1CON, such as every 4^{th} or 16^{th} rising edge. So to calculate the final frequency, the period value is then divided by the instruction clock per prescale value.

When the frequency of the input signal is found, the filtering/tuning process simply compares that frequency value to the constants that were previously defined, and set the user interface outputs accordingly, which are PORTD (7-segment LED display) and [RB2-RB0] (pitch LEDs). The "light_pitch_led" function's conditional if-statements use a tolerance value set at 3.5Hz, which was found by using the oscilloscope to measure the signal's frequency.

Figure 9 outlines the operation and the Appendix provides the full code listing of the software.

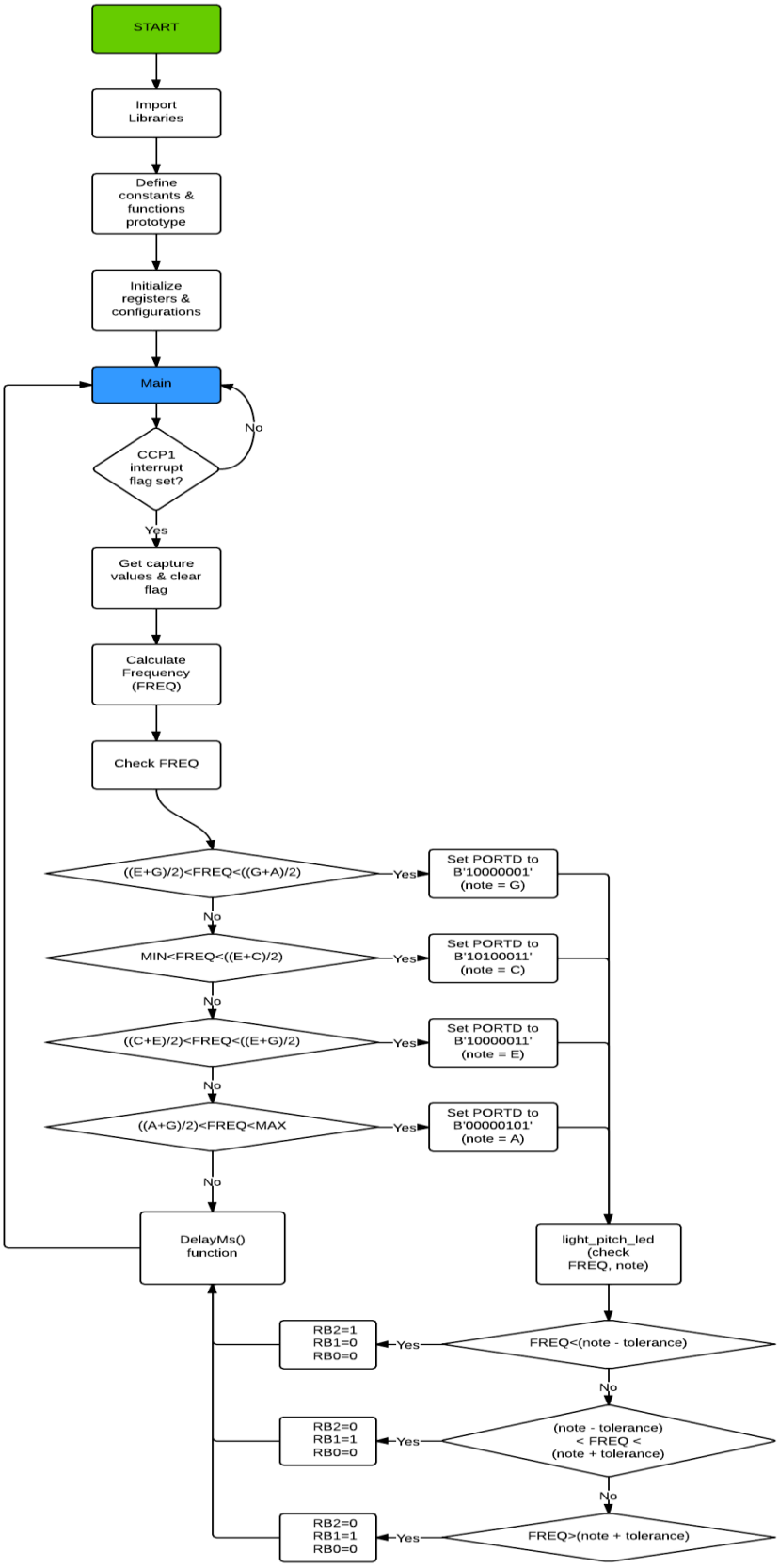

Figure 9 – Software Flowchart

4. Conclusion

This paper has discussed the details of an electronic tuner that tunes a standard 4-string soprano ukulele. The tuner used an embedded Microchip PIC microcontroller that was a primary subject of the Embedded Systems course. The extra peripherals that were not used in the course lectures are the electret microphone and the operational amplifiers. By completing this project, the core concepts of embedded systems have been reinforced and strengthen in the aspects of practicality and applied technology in everyday use. Also by programming in C language, its connection to the lower level assembly language is better understood and appreciated; on the other hand, one who understands assembly language has the valuable background to troubleshoot and modify compiled source code by the C compiler as needed. Although, the drawback of the project might be the usage requirement of the Microchip PIC family of microcontroller, which is hardware restricted and feature limited, in comparison to existing electronic microcontroller kit such as the Arduino, etc.

An excellent future for the project to be contributed by the open-source community is to make the project fully available via GitHub [2]. The project, bundled with the class lectures, will provide a core foundation for new learners who share similar interests in embedded systems. An example for future contribution can be to add extra features to the tuner: to be able to tune better in noisy environment, tune a different variety of ukuleles, etc. Moreover, the tuner can certainly be ported to other microcontrollers and devices such as the Arduino, the Raspberry Pi, etc.

References

[1] Microchip Inc., "PIC16F873/4/6/7 Datasheet," 2013.
ww1.microchip.com/downloads/en/DeviceDoc/30292D.pdf.

[2] GitHub Inc., "A Microchip PIC-based Ukulele Tuner," 2014.
www.github.com/sonphanusa/

[3] Ryan Monteleone, "Microchip Based Automatic Guitar Tuner," 2012.
www.csuohio.edu/academic/success_in_math/posters/CSU/CSU_2012/Monteleon.pdf.

[4] Agile Partners, "GuitarToolkit," 2013.
www.agilepartners.com/apps/guitartoolkit.

[5] Panasonic Electronic, "WM-64PN Datasheet," 2014.
industrial.panasonic.com/www-data/pdf/ABA5000/ABA5000CE11.pdf.

[6] Dimitar Kovachev, "LM358 microphone amplifier," 2011.
lowvoltage.wordpress.com/2011/05/21/lm358-mic-amp/.

[7] Microchip Inc., "DevToolsParts," 2013.
www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2115.

Appendix: Software Listing

```
/*
 * A Microchip PIC-based Ukulele Tuner
 * Son Phan
 * http://github.com/sonphanusa
 *
 * This is the software for a standard 4-string soprano ukulele tuner that tunes the
 * following 4 notes at their respective frequencies: G (392 Hz), C (262 Hz), E (330 Hz), A
 * (440 Hz). The electronic tuner includes a microphone to listen to sound inputs (ideally
 * from a ukulele string), a LED segment display to show one of four notes (G, C, E, A), and a
 * set of red/green LEDs to guide the pitch if it is low, high, or correct.
 *
 * For frequency calculation, the CCP1 interrupt flag (CCP1IF) is constantly checked, when
 * set, Timer1 16-bit integer value, which stored in CCPR1H:CCPR1L,  is captured and
 * calculated. After verifying the captured value is valid, in which the new timer value
 * "capnew" has to be larger than the old value "capold", the difference between "capnew" and
 * "capold" yields the period (the amount of time ran) between captures, as Timer1's
 * continuously counting. For more than every rising edge, this value would have to be divided
 * by the number of edges that occurred, configured in CCP1CON, such as every 4th or 16th
 * rising edge. So to calculate the final frequency, the period value is then divided by the
 * instruction clock per prescale value.
 *
 * This program is developed and built in MPLAB IDE 8.92 with HITECH-C compiler.
 *
 * Reference:
 * Microchip Based Automatic Guitar Tuner - Ryan Monteleone
 * http://www.csuohio.edu/academic/success_in_math/posters/CSU/CSU_2012/Monteleon.pdf
 */

#include <htc.h>            //HI_TECH C header file
#include "delay.h"          //Microchip delay library

__CONFIG (CP_OFF & CPD_OFF & LVP_OFF & WDTE_OFF & BOREN_OFF & PWRTE_OFF & WRT_OFF & FOSC_XT);

/* DEFINE CONSTANTS */
#define  C         261.6    // Notes' frequencies
#define  E         329.6    // +68 Hz
#define  G         392.0    // +62.4
#define  A         440.0    // +48

#define  OSC_CLK   3686400.0         // external/oscillator clock
#define  INT_CLK   (OSC_CLK/4)       // instruction clock

#define  TMR1_PRESC        4        // Timer1 prescale 1:4
#define  TMR1_EDGES        1        // Timer1 capture event (every rising edge)

#define  FREQ_TOLE         3.5      // Frequency tolerance
#define  LOWER_BOUND       100.0    // Set freq. lower bound to 100Hz
#define  UPPER_BOUND       600.0    // Set freq. upper bound to 600Hz
#define  DELAY_TIME        20       // Delay timer in ms
```

```
/* FUNCTIONS PROTOTYPE */
// Init registers and ports values
void core_init(void);
// Light up LEDs according to pitch
// uses the frequency and the note reference
// as passed in parameters.
void light_pitch_led(float freq, float note);

/* FUNCTIONS IMPLEMENT */
void core_init(void)
{
	INTCON = 0b11000000;	//Enable Global (GIE) and Peripheral (PEIE) interrupts

	TRISB =  0b11000000;	//Set PORTB pins to outputs for pitch LEDs
	TRISC =  0b00000100;	//Set RC2 to input for sound signal
	TRISD =  0b00000000;	//Set PORTD pins to outputs for 7-segment LED display
	T1CON =  0b00100001;	//Set TMR1 prescale (TMR1_PRESC)

	CCP1CON= 0b00000101;	//Set CCP1 to capture mode at rising TMR1_EDGES
	TMR1H =  0b00000000;	//Clear Timer1 register
	TMR1L =  0b00000000;	//
	CCPR1H = 0b00000000;	//Clear CCPR1 register
	CCPR1L = 0b00000000;	//

	CCP1IE = 0;		// Disable CCP1 interrupt
	CCP1IF = 0;		// Clear CCP1 interrupt flag
	TMR0IE = 0;		// Disable interrupt on TMR0 rollover

	PORTD =  0b11111111;	// Turn off all 7 segments of the LED display

	RB0 =    0;		// Turn off all pitch LEDs
	RB1 =    0;		//
	RB2 =    0;		//
}

// Main
void main(void)
{
	unsigned int capold, capnew, cap;    //Timer1 capture values
	float FREQ;	//Frequency calculated value

	//Initialize registers and ports
	core_init();

	//Calculation and display UI
	while(1)
	{
		if (CCP1IF) //TMR1 capture occurred (interrupt flag set)
		{
			//Store old capture value
			capold = capnew;
```

```
                        //Get new 16-bit CCPR1 capture value (CCPR1H:CCPR1L)
                        capnew = 256*CCPR1H + CCPR1L;
                        //Clear the flag
                        CCP1IF = 0;

                        if (capnew > capold) //Verify both captures are independent
                        {
                                //Because TMR1 is set to run free without resetting to 0,
                                //we capture its new value at every rising TMR1_EDGES
                                //and subtract the old value to get the actual new value
                                //(the difference)
                                cap = capnew-capold;
                                //Calculate the frequency based on Timer1,
                                //and number of rising TMR1_EDGES
                                //TMR1 Frequency   = INT_CLK / TMR1_PRESC;
                                //Capture Period   = cap / TMR1_EDGES;
                                FREQ = (INT_CLK/TMR1_PRESC) / (cap/TMR1_EDGES);

                                //Tuning and setting User Interface
                                //in the order of G,C,E,A
                                if( ((G+E)/2)<FREQ && FREQ<((A+G)/2) )
                                {
                                        //7-segment LED display
                                        PORTD=0b10000001; //G note
                                        light_pitch_led(FREQ, G);
                                }

                                if( LOWER_BOUND<FREQ && FREQ<((E+C)/2) )
                                {
                                        PORTD=0b10100011; //C note
                                        light_pitch_led(FREQ, C);
                                }

                                if( ((E+C)/2)<FREQ && FREQ<((G+E)/2) )
                                {
                                        PORTD=0b10000011; //E note
                                        light_pitch_led(FREQ, E);
                                }

                                if( ((A+G)/2)<FREQ && FREQ<UPPER_BOUND )
                                {
                                        PORTD=0b00000101; //A note
                                        light_pitch_led(FREQ, A);
                                }

                                //Delay to 'debounce' harmonic
                                DelayMs(DELAY_TIME);
                        }
                }
        } //end while(1)
}
```

```
void light_pitch_led(float freq, float note)
{
        if( freq<(note-FREQ_TOLE) )
        {
                RB2=1;    //low pitch
                RB1=0;
                RB0=0;
                return;
        }
        if( (note-FREQ_TOLE)<freq && freq<(note+FREQ_TOLE) )
        {
                RB2=0;    //correct pitch
                RB1=1;
                RB0=0;
                return;
        }
        if( freq>(note+FREQ_TOLE) )
        {
                RB2=0;    //high pitch
                RB1=0;
                RB0=1;
                return;
        }
}
```