.

Tim Edwards & Josh Johnston

Guitar Tuner

Digital Signal Processing

Table of Contents

***Abstract*- This paper details the design and implementation of a guitar tuner. The IDE Keil is used for coding and debugging an FM4 processor board. The algorithm used to detect a guitar note's fundamental frequency is YIN.**

Introduction

We created a guitar tuner which allows the user to pick a string, record a sound sample, and see if the sample's fundamental frequency matches the string's standard pitch. Otherwise, it indicates the string is in tune, too sharp, or too flat.

Theory

Samples are collected constantly by the device. When a sound input is recognized by the processor, the samples are converted to the frequency domain and analyzed. (Note: In order to do this, we assume the guitar is sufficiently close to the correct note already. If this assumption is invalid, the resulting output is inconclusive.) The peak frequency is compared to the target frequency and the appropriate output is given.

Simulation

Relation to Theory:

Our method has first proven to be reliable in MATLAB, but if we can't produce real-time results we will consider another method. One method we could try is to find the distance between subsequent peaks in frequency. This distance will be equal to the frequency.

Accuracy:

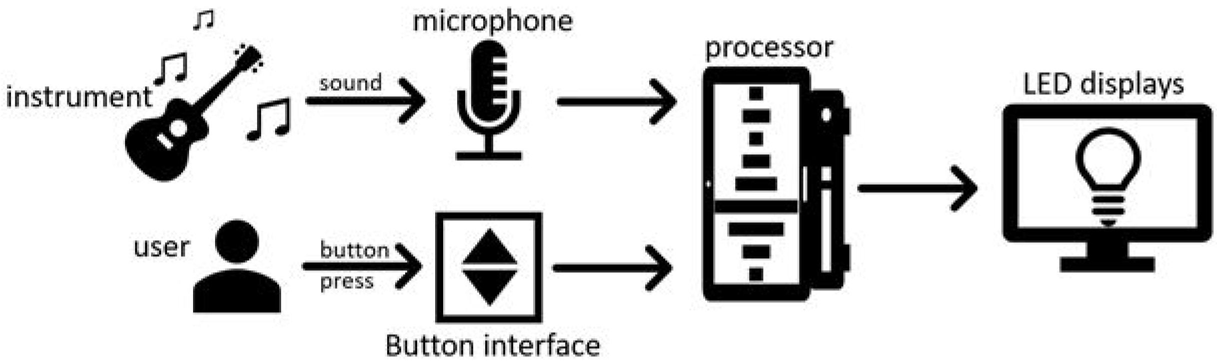
According to the Springer Handbook of Speech Processing [1], human ears can only discern a difference between two tones -both under 500Hz- if the difference is greater than 1Hz. Based on this information, we will make +-0.5 Hz our target accuracy.

Design

The tuner will consist of a microprocessor, a microphone, a series of LEDs, and a toggle button.

The system will take audio samples and the presses of a button interface as input and have output in the form of LEDs. The user input will indicate the note they wish to tune to by cycling through them. The series of LEDs will indicate the note the user selected. When a note is played, one of 3 LEDs will light up. These will indicate that the audio input was either too high, too low, or acceptably close to the desired frequency.

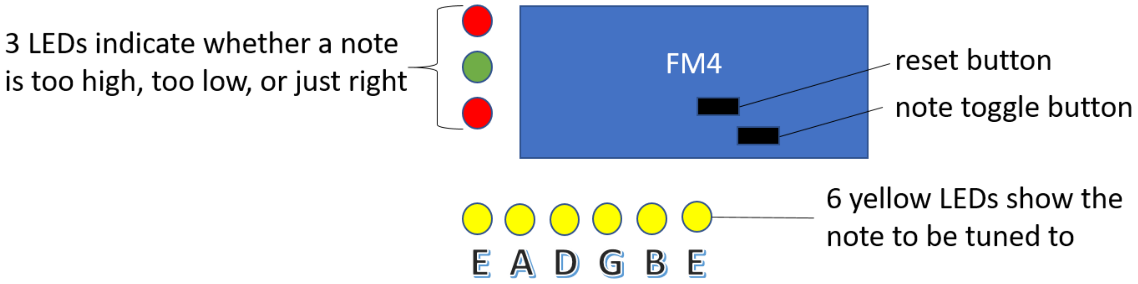
The general concept of the project is as follows:



The completed project:

- Can be toggled manually through E, B, G, D, A, and high e. (These are the notes required to tune a guitar to standard tuning.)
- Indicates whether a played note is too high, too low, or acceptably close relative to the selected note.

Hardware


3 LEDs indicate whether a note is too high, too low, or just right
FM4
reset button
note toggle button
6 yellow LEDs show the note to be tuned to
E A D G B E

Note: In our design, one of the buttons "toggles" through tunable notes and the other (reset) will restart the program, initializing the note to low E.

Ports/Pins:

A0: light for E-String (when it's selected)

A1: light for A-string (when it's selected)

A2: light for D-string (when it's selected)

A3: light for G-string (when it's selected)

A4: light for B-string (when it's selected)

A5: light for e-string (when it's selected)

P7C: light for saying note's too low

P15: light for saying note's in tune

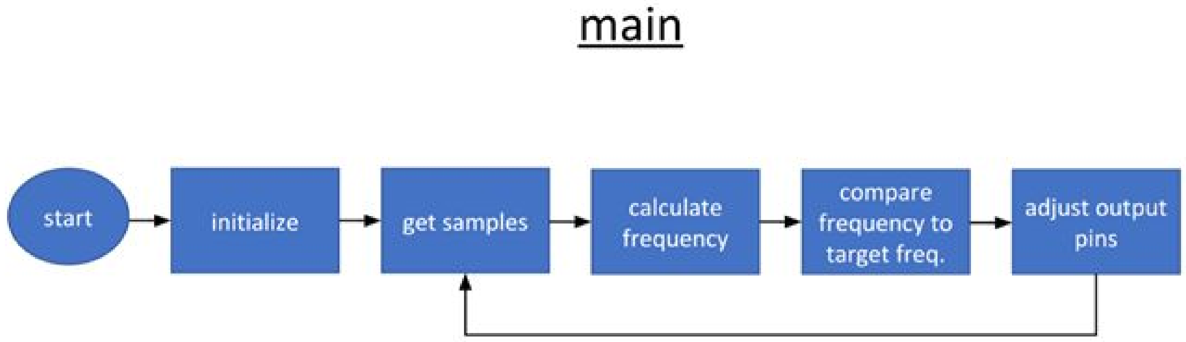
PB3: light for saying note's too high

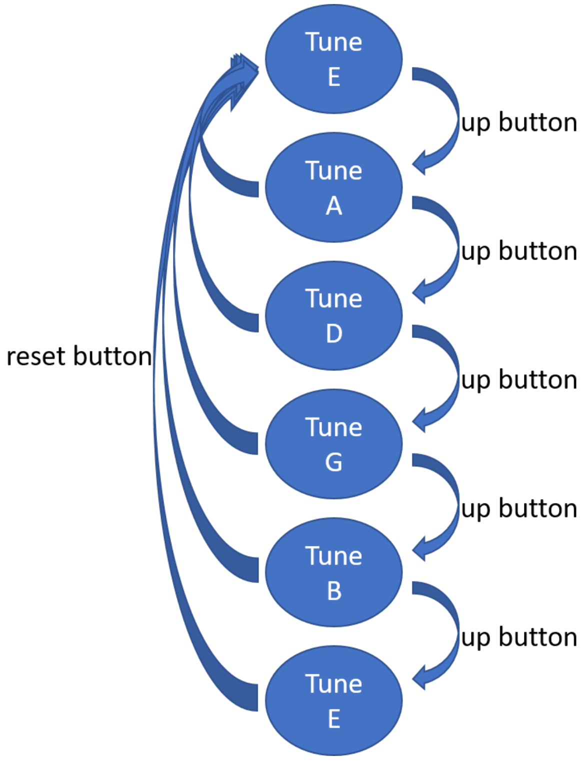
User Interface:

Reset: button to restart program

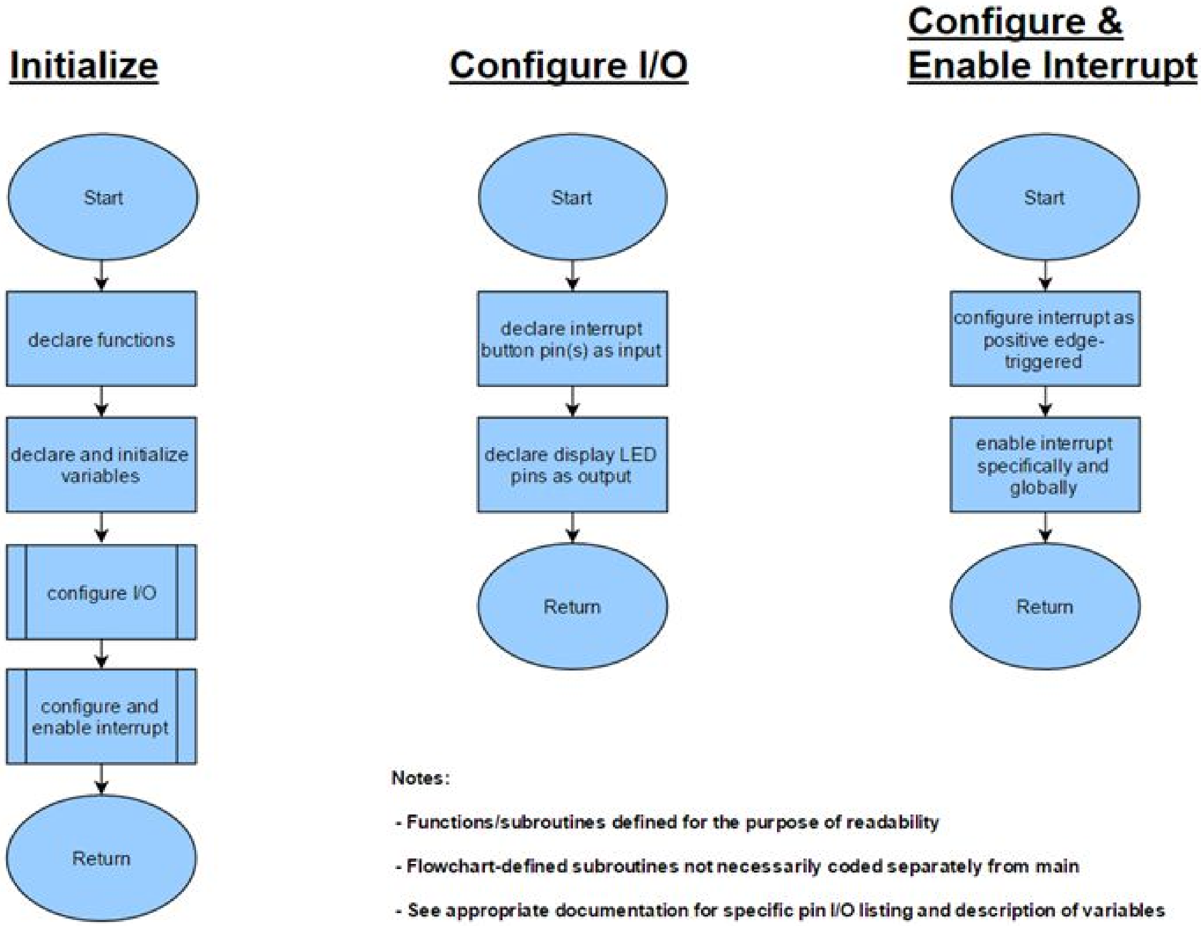
Toggle button: increments note to be tuned - SW2

Software Overview





Lower Level Flowcharts



Initialize

Start
declare functions
declare and initialize variables
configure I/O
configure and enable interrupt
Return

Configure I/O

Start
declare interrupt button pin(s) as input
declare display LED pins as output
Return

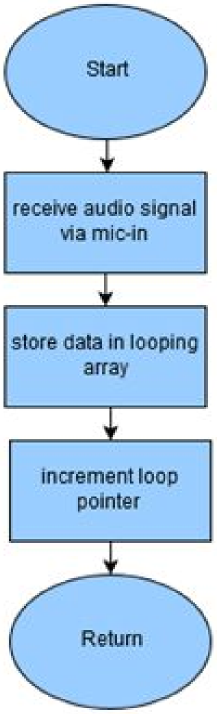
Configure & Enable Interrupt

Start
configure interrupt as positive edge-triggered
enable interrupt specifically and globally
Return

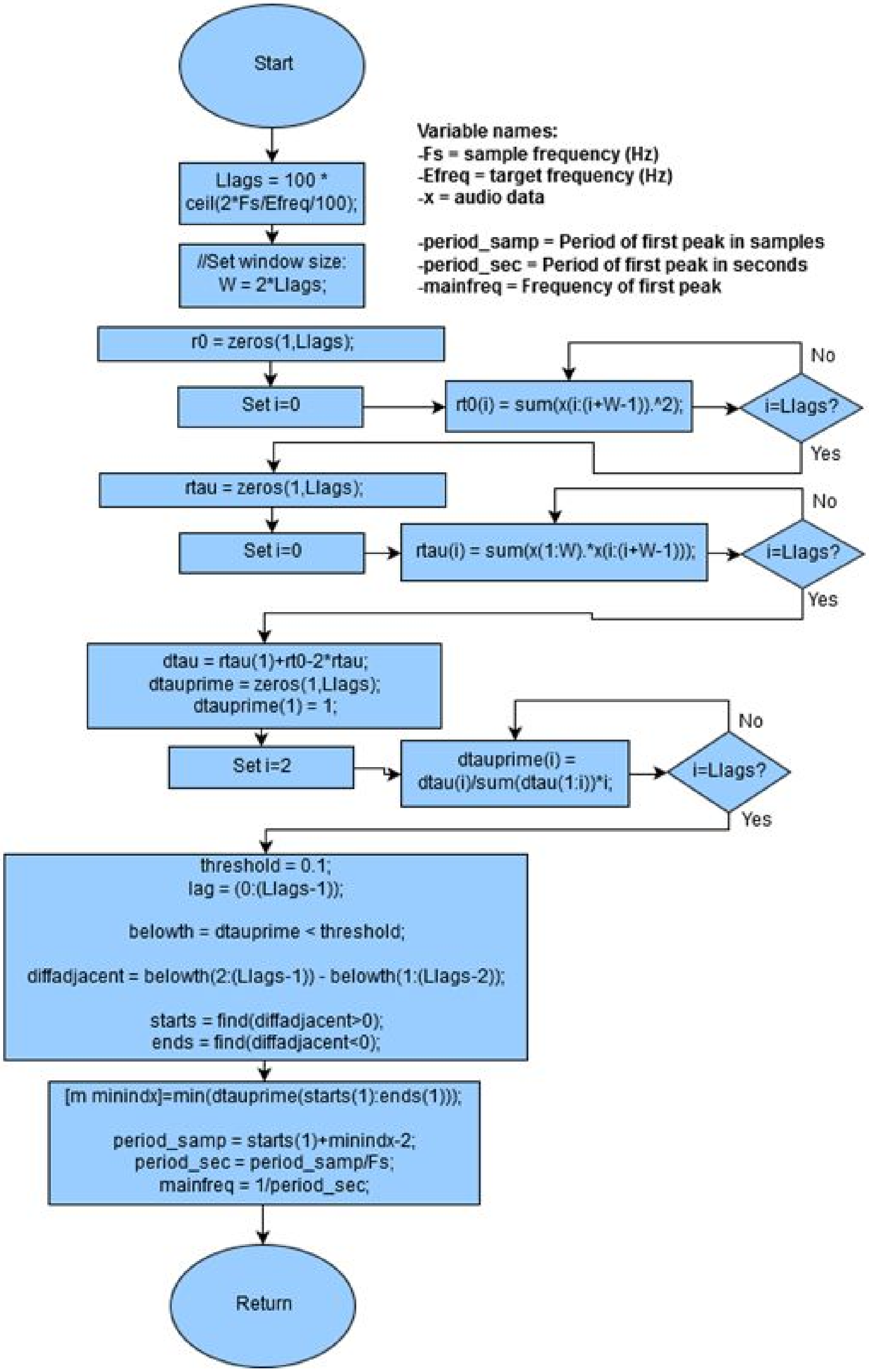
Notes:

- Functions/subroutines defined for the purpose of readability
- Flowchart-defined subroutines not necessarily coded separately from main
- See appropriate documentation for specific pin I/O listing and description of variables

Get Samples



Calculate Frequency

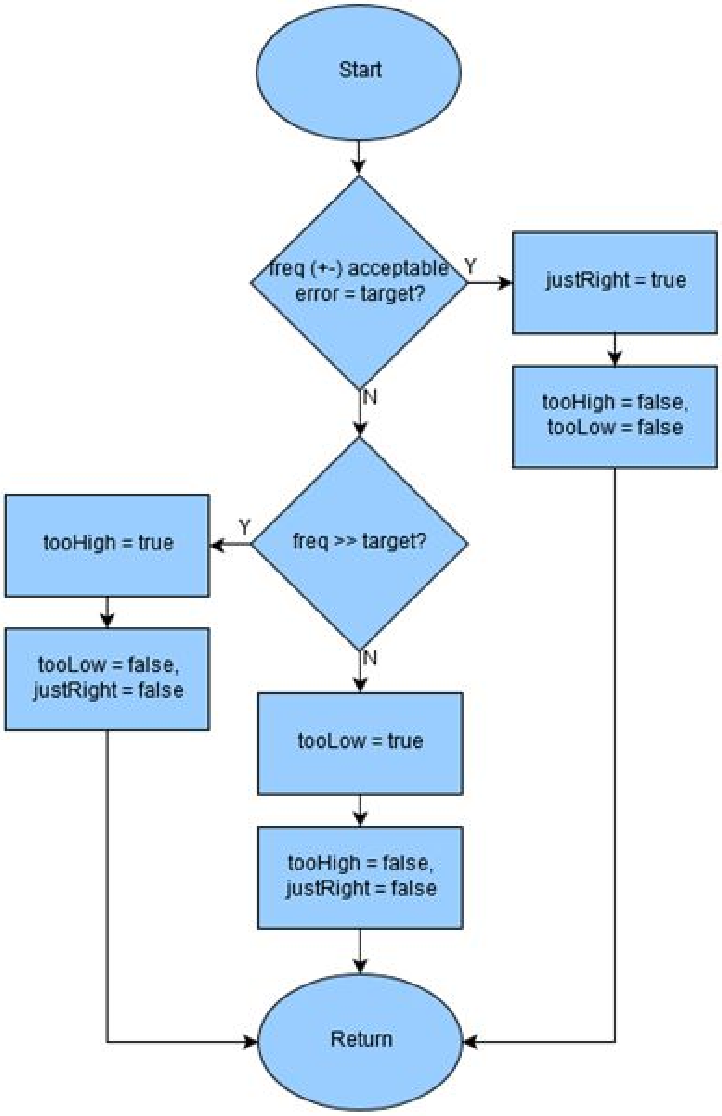


```
Llags = 100 *
ceil(2*Fs/Efreq/100);

//Set window size:
W = 2*Llags;

r0 = zeros(1,Llags);
rt0(i) = sum(x(i:(i+W-1)).^2);

rtau = zeros(1,Llags);
rtau(i) = sum(x(1:W).*x(i:(i+W-1)));

dtau = rtau(1)+rt0-2*rtau;
dtauprime = zeros(1,Llags);
dtauprime(1) = 1;
dtauprime(i) =
dtau(i)/sum(dtau(1:i))*i;

threshold = 0.1;
lag = (0:(Llags-1));

belowth = dtauprime < threshold;

diffadjacent = belowth(2:(Llags-1)) - belowth(1:(Llags-2));

starts = find(diffadjacent>0);
ends = find(diffadjacent<0);

[m minindx]=min(dtauprime(starts(1):ends(1)));

period_samp = starts(1)+minindx-2;
period_sec = period_samp/Fs;
mainfreq = 1/period_sec;
```

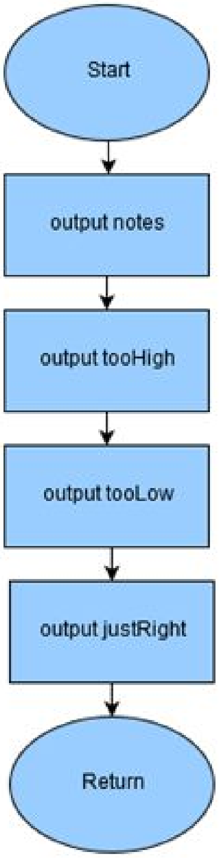
Variable names:
-Fs = sample frequency (Hz)
-Efreq = target frequency (Hz)
-x = audio data

-period_samp = Period of first peak in samples
-period_sec = Period of first peak in seconds
-mainfreq = Frequency of first peak

Compare Frequency


Start
freq (+-) acceptable error = target?
Y
justRight = true
tooHigh = false, tooLow = false
N
freq >> target?
Y
tooHigh = true
tooLow = false, justRight = false
N
tooLow = true
tooHigh = false, justRight = false
Return

Adjust Output



Note: all outputs made to pins providing binary values to LEDs. This function simply updates all outputs to match internal variables.

Note Frequencies (contents of notes array):

-**Low E**: 82.41Hz = 517.80 rad/sec
-**A**: 110.00Hz = 691.15 rad/sec
-**D**: 146.83Hz = 922.56 rad/sec
-**G**: 196.00Hz = 1,231.50 rad/sec
-**B**: 246.94Hz = 1,551.57 rad/sec
-**High e**: 329.63Hz = 2,071.13 rad/sec

Test Methods

We ensured that the program was responding to the button which changed the strings. We ensured that when each string was selected by the user, the variable properly kept track of which string was selected and the corresponding LED lit up. We inputted guitar note samples to the board and made sure that the board properly reported if a string was in tune, too sharp, or too flat. The right LED to indicate that should have also lit up.

To ensure complete accuracy, we tested our device against another guitar tuner and generated tone. We adjusted expected string frequencies as needed.

Results and Discussion

The user was able to successfully select whatever string he or she wanted to tune. Sampling a real guitar, the tuner successfully indicated if a string was sharp, flat, or in tune. The guitar was able to be tuned within acceptable bounds of error.

Conclusion

This project was a guitar tuner where the user selected a string with a button and was told if the string was too sharp, too flat, or in tune. To estimate the guitar's fundamental frequency, YIN (a fundamental frequency estimator) was used. See references for details on YIN [2]. For final testing, we tuned an acoustic guitar and played that guitar. It produced a pleasing sound. Therefore, testing was successful.

References

- [1] B. Kollmeier; T. Brand; B. Meyer (2008). "Perception of Speech and Sound". In Jacob Benesty; M. Mohan Sondhi; Yiteng Huang. *Springer handbook of speech processing*. Springer. p. 65. ISBN 978-3-540-49125-5.

- [2] Alain de Cheveigne; Hideki Kawahara. "YIN, a fundamental frequency estimator for speech and music"

Appendices

-A Keil YIN Implementation (*see references for more info*)