Automated Guitar Tuner

Jordan Pinson, Bryan Gonzalez

Abstract

This document outlines an automated guitar tuner that samples an amplified microphone signal with an STM32F446RE microcontroller, computes its frequency content, and drives a motor to adjust string tension of a guitar. The electret microphone's tiny AC output is first biased and amplified by an LM358 op-amp, centering the waveform at mid-supply and clamping it within the ADC's input range. An RC low-pass filter removes high-frequency harmonics and noise. The STM32 samples at 2048 Hz using the onboard ADC module, applies a Hann window to blocks of 2048 samples, and performs a radix-2 FFT (Cooley–Tukey) to convert the time-domain waveform into frequency-domain data. The dominant frequency bin (peak magnitude) is identified and converted to a pitch. This frequency is compared to the target string frequency; this delta then commands a step and direction signal to a stepper motor controller, which drives a NEMA-14 stepper motor, turning the guitar peg. In testing the 2048-point FFT yielded a complete reading in ~1 s, sufficient to reliably detect string pitch.

I. Introduction

Automatic guitar tuners use electronics to measure the vibration frequency of a string and adjust tuning pegs to achieve a desired note. Commercial systems exist (e.g. Gibson's robotic tuner or the TronicalTune), often using servo or stepper motors on each peg. Our approach uses a single-board STM32 microcontroller with an omnidirectional electret microphone and software pitch detection. Prior work has shown that careful analog front-end design (amplification with DC bias and filtering) improves ADC capture of audio, and that FFT-based pitch detection can be effective for discrete signals. The novelty here is implementing the entire chain (mic, preamp, ADC, FFT pitch detection, and motor control logic) on a resource-limited STM32F446RE. We describe the hardware (MW063002-2 Electret Microphone, LM358 OpAmp, RC Low Pass Filter, ADC Module, STM32F446RE, SN754410 H-Bridge motor driver) and software (Cooley–Tukey FFT, peak detection) design, and report that the system can correctly identify string pitches quickly and accurately enough to tune a guitar. Future work is to implement an LCD screen readout of the relevant data, a user button to cycle through which string is currently being tuned, and an

Uninterruptible Power Supply (UPS) to create a discrete portable package.

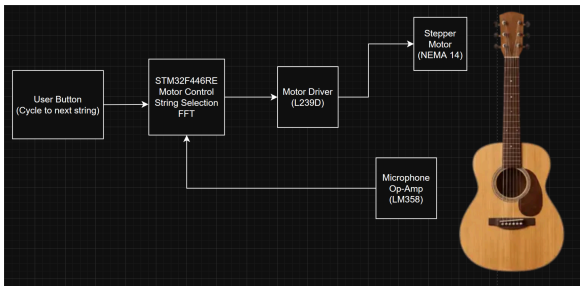

Fig 1.1 - Top Level Diagram

II. System Design

Microphone & Front End

An electret microphone (MW063002-2) capsule produces a small AC voltage (millivolts range) riding on a DC bias. We bias the mic with a pull-up resistor and couple its output through a capacitor into an LM358 op-amp. The LM358 is powered from 5 V and configured as an AC-coupled amplifier with a DC bias at mid-supply. Our design used R3=470 Ω and R4=100 kΩ set a 200× gain. The amplifier output anchors around +1.5 V so that the waveform is centered in the ADC range of [0V, +5V]. The LM358 cannot drive its output fully to the rails, so it effectively “clamps” the signal near 0 V and near VCC–1.5 V; on 5 V supply this means a maximum of ~3.5–4.0 V. In our circuit we observe about 1.58 V as the midpoint (the result of component tolerances and output swing limits), so the AC waveform is ~±1.5 V around that bias. This DC offset ensures that both positive and negative sound pressures are captured by the single-supply ADC.

After amplification, the signal passes through a simple RC low-pass filter (R=220 Ω, C=1 μF), yielding a –3 dB cutoff of about 723 Hz ($f_c = 1/(2\pi RC) \approx 723$ Hz). Such a filter “rolls off” higher-frequency harmonics. Guitar strings produce overtones, and higher harmonics (e.g. 2nd, 3rd) can sometimes be stronger than the fundamental; without filtering the pitch detection algorithm could latch on to these higher harmonics instead of the fundamental. By limiting the bandwidth to [0 Hz, 723 Hz], we keep the fundamental frequencies of all six strings (E2≈82 Hz up to high E4≈330 Hz) while attenuating most overtones and noise. The filtered, biased signal is then fed into the STM32’s ADC input (PA0).

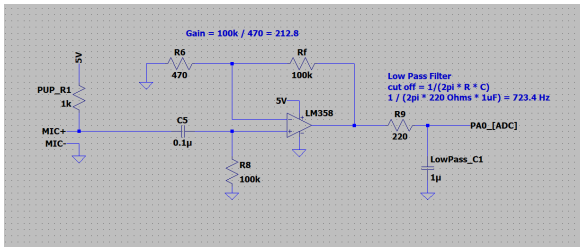

Fig 2.1 - Microphone & Op-Amp Circuit

ADC & Sampling

The STM32F446RE was configured with ADC1 on PA0 in single-ended 12-bit mode, triggered by TIM2 at a 2048 Hz rate. In the timer interrupt we start a conversion, and in the ADC end-of-conversion interrupt we read the 12-bit result. Each sample is scaled to a float in [–0.5, +0.5] (subtracting 0.5 after normalizing by 4095). We buffer 2048 samples per block. The choice of 2048 Hz sampling exceeds twice the highest string harmonic (~600–700 Hz) and provides a Nyquist of 1028 Hz, far above our low-pass. A 2048-sample block thus

represents 1 s of audio. Each block is windowed and transformed by FFT.

FFT & Pitch Detection

The discrete Fourier transform is produced via a Cooley–Tukey FFT (decimation-in-time, radix-2). FFTs are known to reduce the computation from $O(n^2)$ to $O(n \log n)$ compared to a naive DFT, making real-time processing feasible. Before the FFT we multiply the 2048 samples by a Hann window ($w[n]=0.5*(1–\cos(2\pi n/(N–1)))$ for n=0..2047) to taper the ends and reduce spectral leakage. This is a standard practice in audio analysis. The FFT outputs 2048 complex values; since the input is real-valued, the spectrum is symmetric and only the first 1024 bins (0..1023) contain unique information. Each bin k corresponds to frequency $k \cdot (fs/N)$ (bin size = 2048/2048 = 1Hz.

The magnitude of each bin is then computed and the “dominant” frequency is identified as the bin with maximum magnitude. This assumes the string’s fundamental dominates; if the second harmonic is stronger, it is possible to latch on to that instead, but our low-pass filter combined with typical string content usually makes the fundamental strongest. By confining tuning to a specific string at any given time, the software is able to account for when higher harmonics are latched onto, and adjust accordingly. The frequency is then estimated as (bin_index * (fs/N)). We convert that to a float and output it via UART to the terminal.

Code Explanation

High-level summary

The code implementation is a bare-metal STM32F446RE program that implements a guitar tuner + motor driver + microphone. It samples an analog microphone (ADC on PA0) at `SAMPLE_RATE` (2048 Hz), collects `SIZE` (2048) samples, computes an FFT to find the fundamental frequency, decides which musical note (from a note lookup table) the frequency corresponds to, and then drives a small stepper/actuator (via GPIOC) to tighten/loosen the string until it reaches the target frequency for the selected string. The active string can be changed by a pushbutton on PC13 (external interrupt). Results are printed over USART2 (PA2 TX).

Main blocks & flow

1. Initialization

- `button_LED_init()`
    - Enables GPIOA and GPIOC clocks, configures PA5 as LED output, configures PC13 as input with external interrupt (falling edge). EXTI15_10 IRQ enabled. Button press cycles `current_string` (0..5) in `EXTI15_10_IRQHandler`.
- `uart2_init()`
    - Enables GPIOA and USART2 clocks, configures PA2/PA3 to AF7, sets BRR to `UART_BAUD`, enables transmitter (UE & TE). Utility functions `uart_putc`, `uart_puts`, `utoa_dec`, `ftoa_2dec` are used for

printing.

- `adc_timer_adc_init()`
    - Enables GPIOA, ADC1, TIM2 clocks. Configures PA0 in analog mode, ADC1 single conversion on channel 0, enables ADC EOC interrupt, powers up ADC. Starts TIM2 configured to generate periodic update interrupts at `SAMPLE_RATE` (via `timer2_init_for_rate`), and enables NVIC IRQs for TIM2 and ADC.
- `motor_init()`
    - Enables GPIOC, sets several PC pins to output, and sets coil enable bits in `GPIOC->ODR` (using macros `A_EN`, `B_EN`).
- `systick_init()`
    - Configures SysTick with reload `ST_TIME` (defined as 50 * ST_ONE_MS, i.e. 50 ms based on the constants in the file). SysTick runs and calls `SysTick_Handler` periodically.
- `precompute_twiddles(SIZE)` and `prepare_frequency()`
    - `precompute_twiddles` fills `twiddle_cos`/`twiddle_sin` arrays.
    - `prepare_frequency()` fills `frequency[]` bins with increasing frequency values (0 .. fs/2 .. then negative frequencies for upper half).

2. Sampling (TIM2 IRQ → ADC start → ADC IRQ)

- `TIM2_IRQHandler`
    - On each TIM2 update (configured to `SAMPLE_RATE`), it sets `ADC1->CR2` SWSTART to kick a conversion.
- `ADC_IRQHandler`
    - On EOC, reads `ADC1->DR` (12-bit), converts to float `v = 5 * raw / 4095.0f - 0.5f` and stores in `samples[adc_idx++]`.
    - When `adc_idx >= SIZE` it sets `adc_idx = 0` and `fft_ready = 1` to signal main loop that a full buffer is available.

3. Main processing loop (in `main`)

When `fft_ready` flag is observed:

1. Copy `samples[]` into `fft_input[]`.
2. `prepare_samples()` copies `fft_input` into `real_buf`, zeroes `imag_buf`, and subtracts the mean (DC removal).
3. `fft_float(real_buf, imag_buf, SIZE)` — a radix-2 Cooley-Tukey in-place DIT FFT (iterative).

4. `compute_magnitude()` computes magnitude of each complex bin into `mag_buf[]`.

5. A post-filter loop fills `output[]` by:

    - Zeroing any bin with `|frequency| < HP_CUTTOFF` (high-pass)

    - Zeroing any bin with `mag_buf[i] < SUPPRESSION`

    - Otherwise copying `mag_buf[i]`

6. Find the fundamental: iterate from `i=0` to `SIZE/2-1`; find the point with the highest frequency response and set that value as `fundamental`.

7. Map `fundamental` to the nearest note in `note_lut[]`, computing a `ratio` that shows how far between two semitone values it is.

8. Print status to UART depending on `#define` (currently `PRINT_FREQ_BASIC`): current string, frequency, note, and offset.
    - Set `fft_done = 1`.

4. Tuning (SysTick_Handler)

- When `fft_done` is set and `num_steps==0`, SysTick handler:
    - Check if we are getting the 2nd harmonic.

    - Checks if detected `fundamental` is within ~20 Hz of the selected string target (`target_freq[current_string]`); if not, it returns (ignore big outliers).

    - If already exactly in tune (difference 0) returns.

    - Sets `is_high` flag based on whether current freq is higher than target (tighten vs loosen).

    - Computes `num_steps = abs(delta)` × 3 + 1 (a heuristic mapping frequency error to number of motor steps).

- While `num_steps > 0` each SysTick tick calls either `motor_reverse_step()` or `motor_forward_step()` and decrements `num_steps`. Those functions toggle direction bits in `GPIOC->ODR` to step the motor

Motor Control Logic

The tuner maintains an array of target frequencies for the strings of a guitar (E2 = 82.41 Hz, A2 = 110.00 Hz, D3 = 146.83 Hz, G3 = 196.00 Hz, B3 = 246.94 Hz, e4 = 329.63 Hz). After detecting a string's current frequency, the system compares it to the desired pitch. If the pitch is flat (low), the string needs to be tightened (e.g. turn the peg clockwise); if sharp (high), it needs to be loosened. The STM32 then outputs a direction bit and a series of step pulses to an SN754410 H-bridge driver. The SN754410 is a dual H-bridge IC capable of driving up to 1A per channel, commonly used to drive stepper motors. Each end of the bipolar stepper (NEMA 14) would connect to one H-bridge. By energizing coils in sequence (and reversing as needed), the stepper rotates in small steps. The NEMA 14 has 200 steps/rev (1.8° per full step) and

can draw about 0.5 A per phase. In software we use the SN754410 inputs in the correct sequence for forward or reverse stepping. (eg, for our 4-wire bipolar stepper: energize A+ B+, A+ B–, A– B–, A– B+ in sequence for one direction, and reverse that sequence for the opposite direction).

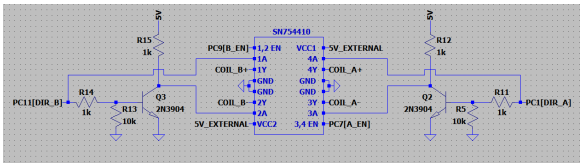

Fig 2.2 - Motor Driver Circuit

III. Results & Discussion

With a 2048Hz sample rate and 2048-sample FFT, we found that the STM32F446RE could complete an FFT-based frequency estimate roughly every second. Using larger FFT sizes or decreasing the sample rate would improve frequency resolution (bin size = fs/N) but at the cost of longer processing time. A sample size of 2048 was chosen as a compromise between speed, accuracy, and STM resources: it yields a frequency resolution of 1 Hz per bin, enough to distinguish adjacent guitar notes (semitones) since our goal is coarse tuning adjustments. The sample block updates every ~1 s, which is short enough to see pitch changes as the string is plucked and corrected.

In testing, we were required to find the balance between sample rate and the number of samples. If the number of samples is greater than our sample rate, then our FFT output is very accurate and we reduce spectral leakage. However to even get the accuracy down to 0.5 Hz bins brings our compute time up to 2 seconds to reach a completed FFT; in effect, increasing the time to compute in exchange for greater accuracy. Increasing the sample rate to be more than the sample size speeds up compute time, but resulted in a loss of accuracy. A 2 Hz difference is significant for a musician so we opted for a 1 Hz bin size. When working with our DFTs, we found that equal sample rate and number of samples gives us 1Hz of accuracy every second is sufficient to achieve our goal.

Overall, our prototype meets the requirements outlined at the start: it reliably identifies the string's pitch using the microphone input and FFT. One deficiency that remains to be addressed is the motor itself. We devoted the majority of our time to getting the microphone output into the correct voltage range for the ADC and the FFT to reliably latch on to the correct frequency, leaving motor testing for later in the design process. In an effort to limit our out of pocket cost, we selected a reliable and readily available stepper motor with purported specifications that met our needs. During testing, including powering both coils, we only get enough torque to loosen the strings but not enough to tighten the string. On the hardware side, the SN754410 can work at higher voltages but has a current limit of 1A. Most higher torque motors require at least 2A. Given the time and financial constraints, we were unable to acquire a motor that met our design requirements in real world testing. Because of this, our current system is only able to tune down high frequencies but cannot tune up. In future, SN754410 and NEMA 14 for a system that produces more torque without needing to change the logic or code. This means that a drop-in solution could be

identified and tested quickly and easily due to the portability of our design.

Reliable Detection

For string D3 (146.83 Hz), we can see that we are clipping at the top of the ADC but the Op-Amp 5V rail keeps it from increasing (protecting the board). Despite the clipping, the FFT is able to consistently latch on to the correct fundamental frequency.

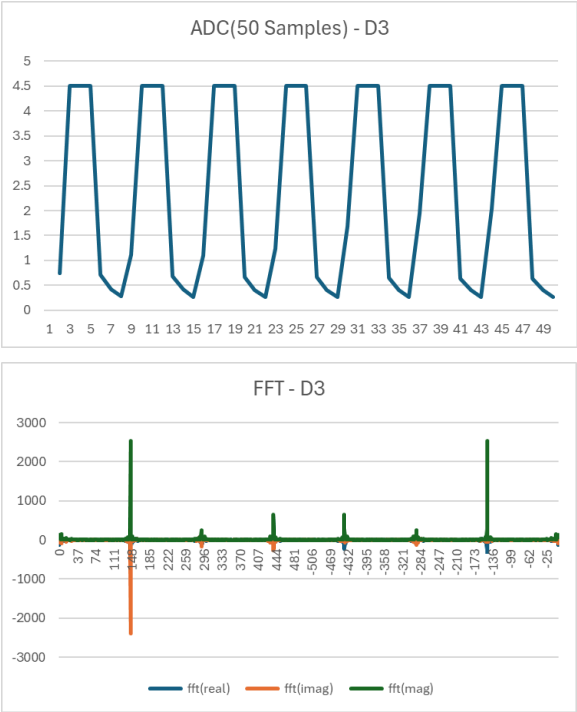

Fig 3.2 - Voltage Output and Frequency Binning for D3 (146.83 Hz)

For strings A2 (110.00 Hz) and E2 (82.41), the fundamental frequency is closer to the noise floor, but the second harmonic becomes more dominant. The system is able to identify this, latch on, and use it for accurate tuning.

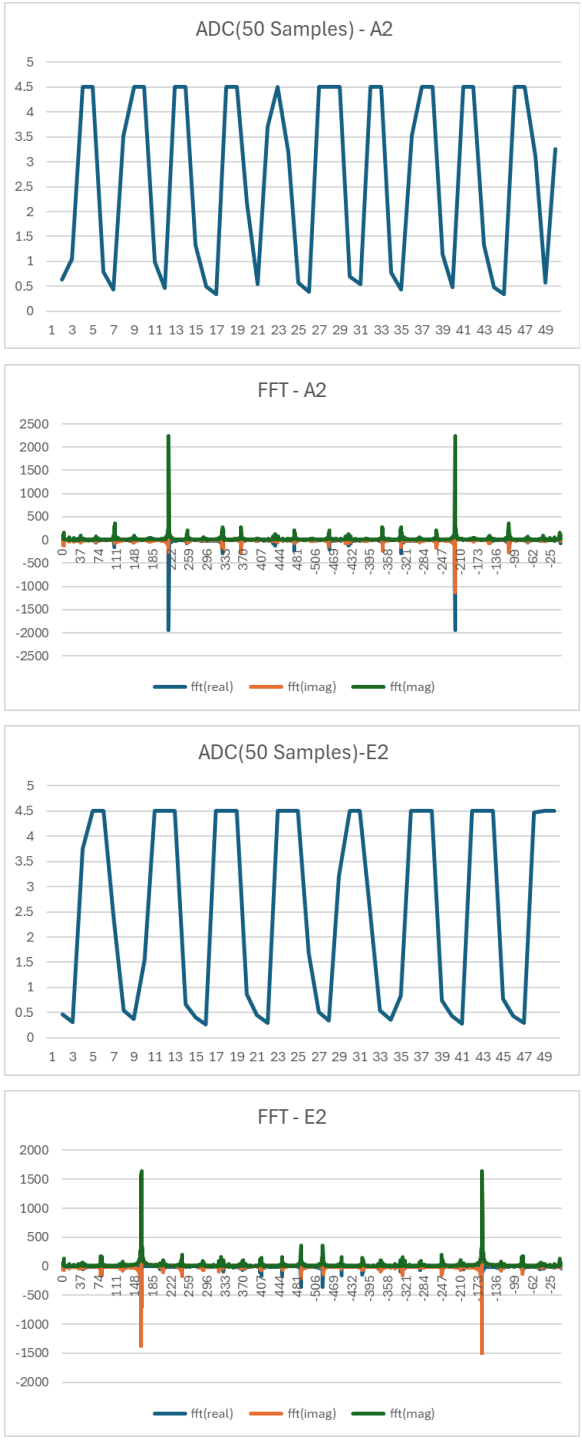

Fig 3.2 - Voltage Output and Frequency Binning for A2 (110.00 Hz) and E2 (82.41)

Conclusion

We have successfully designed an automated guitar tuner using an STM32F446RE microcontroller. The system amplifies an electret microphone signal with an LM358 (DC-biased at mid-supply) and filters it at ~723 Hz to focus on guitar fundamentals. A 2048-point Cooley–Tukey FFT is used to convert the sampled waveform into a frequency spectrum, from which the dominant frequency (string pitch) is extracted. In testing the STM32 delivered a new frequency estimate roughly every 1s, and it correctly identified string pitches. The control loop compares the detected frequency to the desired target and sends direction/step pulses to an SN754410 H-bridge to drive a NEMA-14 stepper motor on the tuning peg. Our approach follows known techniques (DC offset bias, Hann window, peak-picking FFT) shown to be effective in similar tuners. Motor control implementation allows the tuner to automatically adjust each peg until the string is in tune.

References

Kovachev, D. (2013, February 4). *LM358 microphone amplifier*. Low Voltage. Mostly Harmless. . . https://lowvoltage.wordpress.com/2011/05/21/lm358-mic-amp/

List, J. (n.d.). *cuttlefish – Hackaday*. Hackaday. https://hackaday.com/tag/cuttlefish/#:~:text=guitar%20tuner%20www,was%20confusing%20the%20Arduino%E2%80%99s%20algorithm

Roche, B. (2024, November 17). *Frequency detection using the FFT (aka pitch tracking) With Source Code*. Bjorg. https://bjorg.bjornroche.com/audio/frequency-tracking-fft/#:~:text=To%20reduce%20the%20sidelobes%2C%20we,I%20used%20the%20Hann%20window

Gentleman, W., & Sande, G. (1966). *Fast Fourier Transforms: for fun and profit*. https://www.semanticscholar.org/paper/Fast-Fourier-Transforms%3A-for-fun-and-profit-Gentleman-Sande/e4a80900817cbaddee9a81c67a44da0dcfae7c4

L293D data sheet, product information and support | TI.com. (n.d.). https://www.ti.com/product/L293D#:~:text=The%20L293%20and%20L293D%20devices,supply%20applications

Pololu - Stepper Motor: Bipolar, 200 Steps/Rev, 35×28mm, 10V, 0.5 A/Phase. (n.d.). https://www.pololu.com/product/1208/specs#:~:text=Save