EE 119c Functional Specification:
FPGA-Based Pitch Detector and Tuner

By: Dan Pipe-Mazo

Instructor: Glen George

California Institute of Technology

Spring 2013

I. DESCRIPTION

This project is a digital, real-time pitch detection and tuning system. The system is able to determine the dominant frequency of any audio signal between 50 and 5,000Hz with 2 cent general accuracy. The detected frequency will be displayed in floating-point form on a display with 0.001 Hz resolution. The device can wither be used in a general pitch detection mode, or a string tuning mode in which the performance is tailored to better detect the desired frequency.

The system uses a variable sampling rate in order to correctly identify frequencies. At the time at which the pitch detection algorithm converges, the sampling rate will be approximately 1024 times the detected frequency. In order to detect frequency, an autocorrelation algorithm is employed. The autocorrelation algorithm begins scanning for the lowest detectable frequency, and increases the sampling rate until the autocorrelation peak lands in the range of acceptable bins for which the accuracy specifications are met.
The system also has the ability to use a stepper motor to automatically tune a guitar string. The system will run its pitch detection algorithm and attempt to tune the string using the auxiliary stepper motor unit until the pitch detected is within the acceptable accuracy thresholds.

II. INPUTS

The LM4550 audio codec takes audio input from either a standard carded microphone or a line-level audio signal. This codec then supplies 18-bit PCM samples over a serial connection to the system.

The user is able to input commands through the eight on-board switches, five on-board buttons or reset button.

A 5V power supply is needed to run the FPGA and all auxiliary devices except for the stepper motor. The stepper motor requires a 30V power supply capable of sourcing at least 0.5A.

III. OUTPUTS

The system outputs important information to the 20-character x 4 line LCD dot matrix display.

If in automatic tuning mode, the system outputs step and direction commands to an A3967 stepper motor driver which in turn outputs controls to a 0.33A/phase, 200 step/rotation stepper motor which is used to turn the guitar tuning knob.

The system also outputs 18-bit PCM audio samples to the LM4550 codec which then outputs an analog audio signal to the headphone out jack on the board.

IV. USER INTERFACE

System Modes

The system has 7 basic modes of operation:

1. **Free tune mode** Capable of detecting any input frequency between 50 and 5,000 Hz to within 2 cent accuracy
2. **Tune E2 string mode** Capable of detecting any frequency between 50 and 5,000 Hz. Detects frequencies around 82.407 Hz to within 2 cent accuracy

3. **Tune A2 string mode** Capable of detecting any frequency between 60 and 5,000 Hz. Detects frequencies around 110.000 Hz to within 2 cent accuracy
4. **Tune D3 string mode** Capable of detecting any frequency between 80 and 5,000 Hz. Detects frequencies around 146.832 Hz to within 2 cent accuracy
5. **Tune G3 string mode** Capable of detecting any frequency between 108 and 5,000 Hz. Detects frequencies around 195.998 Hz to within 2 cent accuracy
6. **Tune B3 string mode** Capable of detecting any frequency between 135 and 5,000 Hz. Detects frequencies around 246.942 Hz to within 2 cent accuracy
7. **Tune E4 string mode** Capable of detecting any frequency between 182 and 5,000 Hz. Detects frequencies around 329.628 Hz to within 2 cent accuracy

The free tune mode should be used when attempting to identify an unknown frequency within the acceptable range. Each guitar string has its own mode which should be used when attempting to tune that particular string. The free tune mode guarantees accuracy across all frequencies, while the specific string tuning modes are most accurate around the respective string's frequency, though they can detect any frequency in the above range. The tradeoff being made between the free tune modes and the individual string modes is that of consistency. The free tune algorithm can occasionally misidentify overtones or undertones. This behavior is less tolerable when attempting to tune a guitar string, so the algorithm is slightly modified to improve consistency of identifying the correct frequency of interest.

Pushbuttons

When the system is reset, it will start up in free tune mode. The user may switch through the modes using the left and right buttons on the FPGA board. Pressing the right button will increase the mode as listed above and pressing the left button will decrease the mode. If the right button is pressed in tune E4 string mode, then the system will wrap back around to free tune mode. If the left button is pressed in free tune mode, then the system will wrap back around to tune E4 string mode.

When the system is in one of the string tuning modes (any mode other than free tune mode), pressing the middle button will activate automatic tuning mode. This will not engage the motors, but simply put the system into a state where it is ready to run the automatic tuning algorithm using the stepper motor. When the system is in automatic tuning mode, pressing the top button will engage the stepper motor and will run the automatic tuning algorithm. Once this button is pressed, the stepper motor will attempt to turn the tuning knob to tune to the frequency corresponding the the string mode which the system is in. The stepper motor can be disengaged by pressing this button again. Changing between any string mode or turning off automatic tuning mode will also disengage the stepper motor.

If the string being tuned reaches the desired frequency, the system will alert the user that the correct tune has been achieved. The bottom pushbutton can be used to dismiss the tuned message and either begin tuning another string or continue to tune the current string.

Display

The display is split into 4 lines, with 20 characters on each line.

The first line of the display shows which mode the system is currently in.

The second line of the display shows the target frequency for tuning if the system is in string tuning mode, else it is blank.

The third line of the display shows the most recent frequency which the system detected.

The fourth line of the display shows status information regarding the state of the tuning algorithm if the system is in string tuning mode, else it is blank. If the frequency detected falls within the accuracy bounds for the current string which the system is attempting to detect, then this line will show the word "tuned". Else, if the system is not in automatic tuning mode, this line will be blank. If the string is not in tune and the system is in automatic tuning mode, then this line will display whether or not the automatic tuning algorithm is running. The state of the automatic tuning algorithm is toggled by pressing the top pushbutton.

Switches

The switches are numbered from 7 to 0 from left to right on the FPGA board with the pushbutton array in the bottom right corner of the board as you look at it. All switches are considered to be a '1' when in the up position and a '0' in the down position.

Switches 7 to 3 (MSB to LSB) make up a 5-bit unsigned word which controls the volume of the output audio at the headphone jack. Volume increases with higher word values.

Switch 2 controls the input to the LM4550 audio codec. If this switch is up, then the line in input will be fed into the codec's ADC and will serve as the analog signal for the samples which the system receives. If this switch is down, then the microphone will serve as the input to the codec.

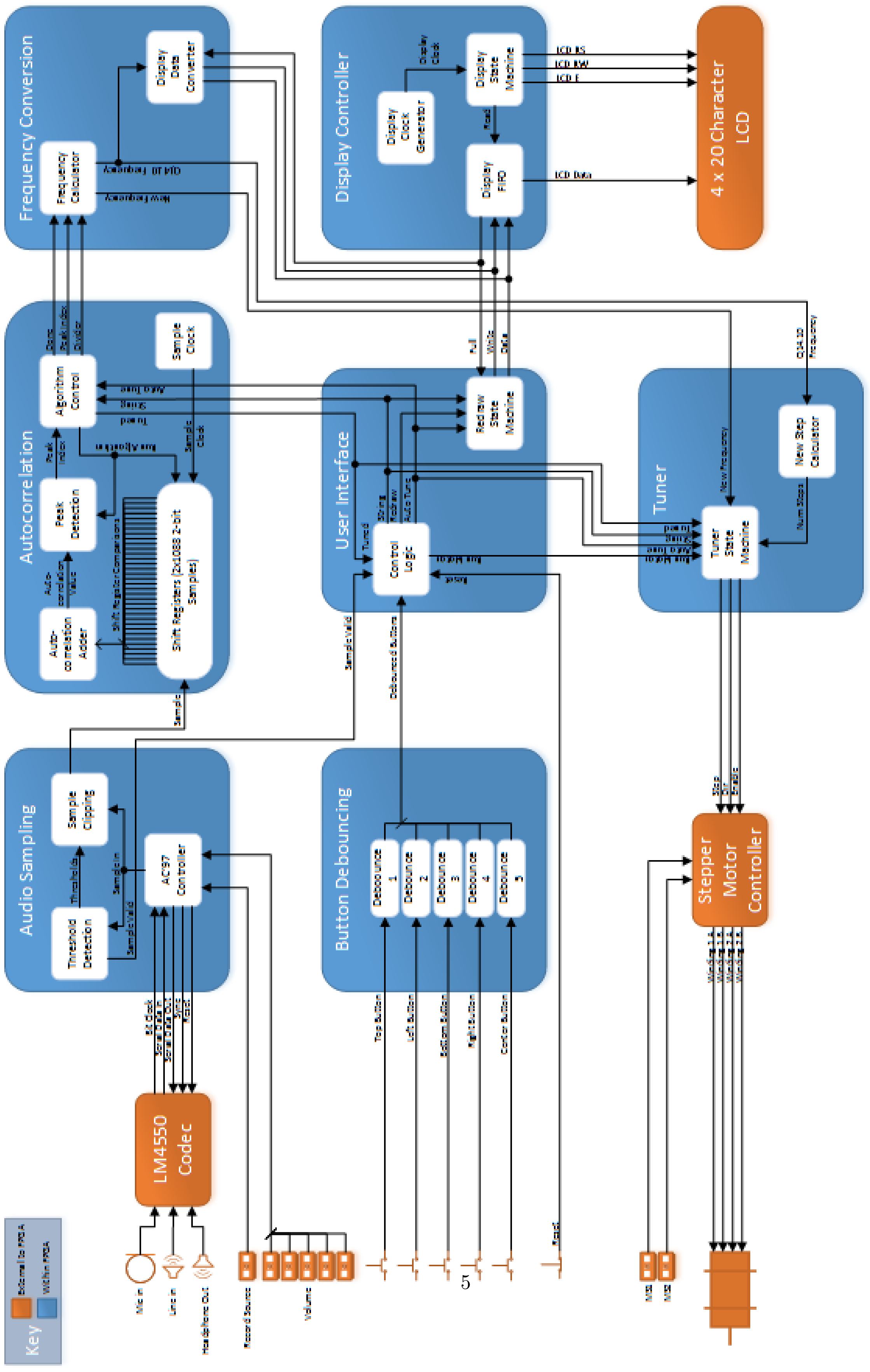
Switches 1 and 0 make up the 2-bit codeword which controls the stepping resolution of the stepper motor driver. This switches are fed into the MS2 and MS1 inputs, respectively, of the A3967 stepper motor driver. The values of MS2 and MS1 and corresponding step resolutions can be found in the table below. Best performance is obtained with these switches both in the 'up' position, since this allows the finest stepping control. Quicker convergence can be obtained using full stepping mode, since less steps need to be taken.

MS2	MS1	Step Resolution
0	0	Full Step
0	1	1/2 Step
1	0	1/4 Step
1	1	1/8 Step

Motor

When using the system to automatically tune a guitar string, it is necessary for the user to attach the motor and supplied tuning peg bracket to the guitar for use. To do this, simply hold the motor by the casing and insert the guitar tuning peg into the tuning peg bracket attached to the motor drive shaft. When ready to tune, engage the motor by pressing the top pushbutton. At this point, the system is ready to attempt an automatic tune, so simply strum string until the word "tuned" appears on the bottom line of the display.

V. BLOCK DIAGRAM



VI. BLOCK DESCRIPTIONS

VII. ALGORITHM DESCRIPTION

Audio samples are input through a standard cardiod microphone to an LM4550 audio codec. This codec uses the AC'97 standard to transmit 18-bit signed PCM samples over serial to the FPGA at a rate of 48 KHz. Once within the FPGA, the audio samples are decimated from 18 bits to 2 bits. In order to do this, a "thresholding" mechanism is employed. After every 1024 samples, a sample maximum and minimum are identified. The sample amplitude is then calculated as the difference between the maximum and the minimum. Thresholds are then set at

$$threshold_{high} = maximum - \frac{amplitude}{8}$$

and

$$threshold_{low} = mimumum + \frac{amplitude}{8}$$

If the incoming audio sample is greater then the high threshold, it is assigned a value of 1. If the sample is below the minimum threshold, it is assigned a value of -1. If the sample is between the two thresholds, it is assigned a value of 0. This allows for samples to be defined in 2 bits as opposed to 18, and also serves to filter out high frequency noise.

The 2-bit samples are then fed into two 1088-sample shift registers. Each round of the pitch detection algorithm takes 2176 samples. For the first half of the samples (samples 0 - 1087), the input audio signal is fed into both shift registers. For the second half of the samples (samples 1088 - 2175), the incoming audio sample is fed into only one of the shift registers. Between each entry in the two shift registers, there is a simple comparator. If the samples at the i^{th} index in each shift register match, then a '1' is outputted, else a '0'. The 1088 comparator outputs are then fed into a 1088-bit adder tree which computes the bit sum. Using this setup, autocorrelation values are computed. The output of the adder tree for each sample, i, in the second half (samples 1088 - 2175) corresponds to the discrete autocorrelation value for bin $(i-1088)$.

The system performs peak detection on the autocorrelation results, and remembers the value and bin number of the maximum overall peak across the entire sampling round. At the end of the round, after 2176 samples have been taken, the identified frequency is equal to the sampling frequency divided by the bin number of the result. However, the algorithm does not stop here. Sampling frequencies are generated using a clock divider to divide down the 100MHz system clock. The algorithm converges when the sampling frequency is as close to 1024 times the identified frequency as possible. In oder to accommodate this, after each round of sampling, a new clock divider is calculated as

$$new\ divider = \frac{old\ divider \times (bin + 1)}{1024}$$

When the new divider is equal to the old divider, then the algorithm has converged and the frequency is reported as a clock divider and a bin number.

Once a frequency has been found, it is converted from the divider and bin number result into a Q14.10 fixed-point result by performing

$$detected\ frequency = \frac{system\ clock}{divider \times bin} = \frac{100MHz}{divider \times bin}$$

This frequency is then converted to its ASCII equivalent and is output to the display.

If the system is in automatic tuning mode, the Q14.10 frequency is also sent to the tuning unit. This unit takes the input frequency and calculates the number of steps which are required to tune the string to the desired frequency. The number of steps needed to be taken is calculated as

$$steps = \frac{(previous\ steps) \times (expected\ freq - actual\ freq)}{previous\ freq - actual\ freq}$$

where "expected freq" is the expected frequency for the string to be in tune, "actual freq" is the new reported frequency from the autocorrelation unit, "previous freq" is the lacteal frequency from the previous tuning attempt and "previous steps" are the number of steps taken in the previous tuning attempt. Using this algorithm, the system dynamically learns the spring constant of the string and converges to the correct frequency. If the tuning algorithm is being run for the first time, a predetermined number of steps are taken.

VIII. FPGA SIZE

The final design utilizes approximately 2,500 Xilinx Spartan 6 standard SLICEs and 30 Xilinx Spartan 6 DSP48A1 slices. Each standard SLICE contains 8 DFFs and four six-input LUTS. Each DSP48A1 slice contains an 18-bit multiplier, an adder and an accumulator. The XTREME-DSP slices are mainly used for the two dividers required by the design. The design was created so that the dividers are not in time-critical sections of the code. They can easily be replaced with subtraction loops, allowing the entire design to be compiled into simple logic and freeing the design from using any FPGA-specific resources. The design also uses 2 8-bit block RAM blocks which are used in a FIFO to communicate between the main control loop and the display controller.

IX. ALGORITHM RUNTIME

The algorithm begins by taking 2176 samples at 24.42 KHz, which takes 89.1ms. Each subsequent sample attempt will run significantly faster than 89.1 ms, since this is the slowest possible sampling rate. While the algorithm takes a variable amount of time to converge, it will typically converge between 100 - 200 ms.

If the system is in automatic tuning mode, steps are sent at a rate of 700Hz, with a 200ms wait period after each string of steps before enabling the autocorrelation algorithm to run again. This causes the automatic tuning algorithm to converge on the order of 1 - 5 seconds, depending on how far off the string was from being tuned.

X. FUNCTIONAL SPECIFICATION

The FPGA-Based Pitch Detector and Tuner will display the frequency of any audible tone between 50 and 5,000Hz to 2 cent accuracy.

Features

- LCD Display shows detected frequency, closest musical note and margin of error from closest musical note.
- FPGA-based sigma-delta converter sampling at speeds up to 50Msps.
- Frequency identification accuracy within 1Hz and 0.1% for all values between 15Hz and 5000Hz (C_0 to $D_8^{\#}$).
- Stepper motor with 200 steps per revolution and micro stepping capability allows for precision tuning while also having enough torque to crank stiff tuning knobs.

- Precise, digitally-controlled amplification and background-noise cancellation prevents erroneous output from displaying on the screen and increases the accuracy of the system.

Inputs

- Pushbutton rotary encoder and four input keys (up, down, left, right) for navigating through the menus and features of the user interface
- Microphone to capture audio

Outputs

- LCD screen displays exact frequency of tone played, closest note, and margin of error from closest note
- If automatic tuning is activated, stepper motor will rotate to tune instrument

Major Hardware

The FPGA-Based pitch detector and Tuner will run on a Xilinx FPGA. The FPGA will need to be between 2 million and 4 million gates, as discussed in the FPGA Size Estimate section above.

In addition to the FPGA, the pitch detector will utilize:

- A sigma-delta converter with $\geq$ 20 bits of resolution capable of sampling at speeds greater than 50Msps.
- An analog low-pass filter (10KHz corner frequency), in order to allow for faster pitch calculation times.
- Analog voltage subtractor and amplifying circuitry controlled by digital potentiometers in order to subtract out background noise and perform clipping.
- A LCD or similar display.
- A stepper motor with $\geq$ 200 steps per revolution, accompanied by stepper motor driver chips.

User Interface

The FPGA-Based Pitch Detector and tuner will feature a LCD which the act to both display information to the user and take input from the user. The LCD and accompanying switches and rotary encoder will allow the user to select a mode to operate the device in from one of the following modes:

1. **General Pitch Detection**: Have the pitch detector continuously run and display the current pitch of audio input. Pitch shown in to an appropriate number of decimal places.
2. **Tuning**: Choose a note in the range of C_0 to $D_8^{\#}$ and view how close to in tune the audio input is to the desired note. Real-time feedback allows for easy and quick manual tuning. Pitch shown in Hz to an appropriate number of decimal places. Visual feedback as to how far out of tune from the desired note is provided on-screen.
3. **Automatic Tuning**: For any open-string guitar tuning, choose a note, attach the stepper motor unit to the tuning knob, pluck the note once, and by the time time note finishes playing, it will be perfectly in tune.

Limitations

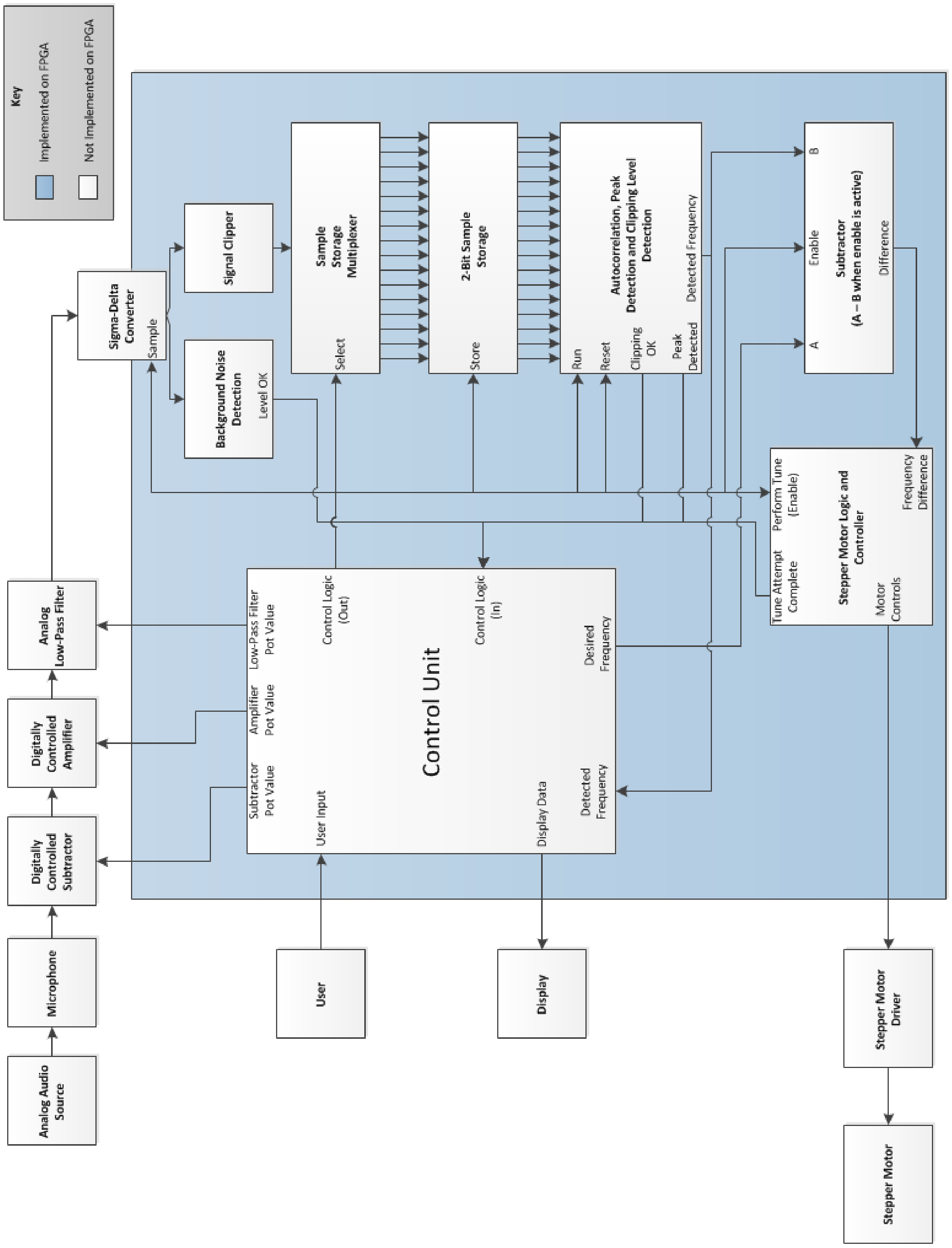
Current limitations for the device are:

- Can only detect frequencies from 15Hz to 5KHz within 0.1% and 1Hz error margins

- Automatic tuning only works for guitars
- Not able to detect multiple notes being played at the same time

All of the above limitations can potentially be solved through further development, but are not included in the base design of the project.

XI. BLOCK DIAGRAM



XII. DEMONSTRATION

The project will be demonstrated with the following tests:

1. Use a function generator to generate tones in the range of 15 to 5000 Hz, and view the display of the pitch detector to verify functionality. Ensure that 1Hz or 0.1% error margins are obeyed, whichever margin is tighter.

2. Use a function generator to generate tones close to but not exactly that of E2, A2, D3, G3, B3 and E4, the 6 open strings of the guitar, and use the information on the LCD of the pitch detector to manually tune the function generator to precise frequencies.

3. Untune a guitar string and use the pitch detector's automatic tuning motor to automatically tune the string in a single pluck.

XIII. SCHEDULE

Weekly milestones are proposed as follows:

1. Choose all hardware and order/sample parts. Completely flesh out block diagrams, detailing all signals and control logic.

2. Write control unit VHDL code

3. Write sampling and sigma-delta code, breadboard analog circuitry

4. Write autocorrelation and peak detection code

5. Write motor control code, breadboard entire design

6. Integrate all code together, design any PCBs needed

7. Write VHDL test bench

8. Test using test bench and fix bugs

9. Assemble PCBs and final hardware design, more testing and debugging

10. Final testing, presentation and documentation

Notes

[1]**Real-Time Digital Hardware Pitch Detector**, Dubnowski, Shafer and Rabiner. Published February 1976 in Volume ASSP-24, No. 1 of *IEEE Transactions on Acoustics, Speech and Signal Processing*. Currently available at http://cronos.rutgers.edu/~lrr/Reprints/095_hardware%20pitch%20detector.pdf