|  |
| --- |
| Thayer School of Engineering |
| DIGiT |
| Digital Integrated Guitar Tuner |
| **Brett Nicholas and Dylan Sewell** |
|  |
| **ENGS 31: Final Project Written Report**  **Professor Hansen** |

**

|  |
| --- |
|  |

**Abstract:**

The goal of this project was to design and implement an FPGA-based digital guitar tuner that enables a user to easily tune the six strings of an electric guitar to within half of a semitone from the correct pitch, without the need for Fourier analysis or frequency domain conversion. Ultimately, the final product was successfully implemented in hardware and performed satisfactorily under extensive product testing, all from within an easily navigable user interface.

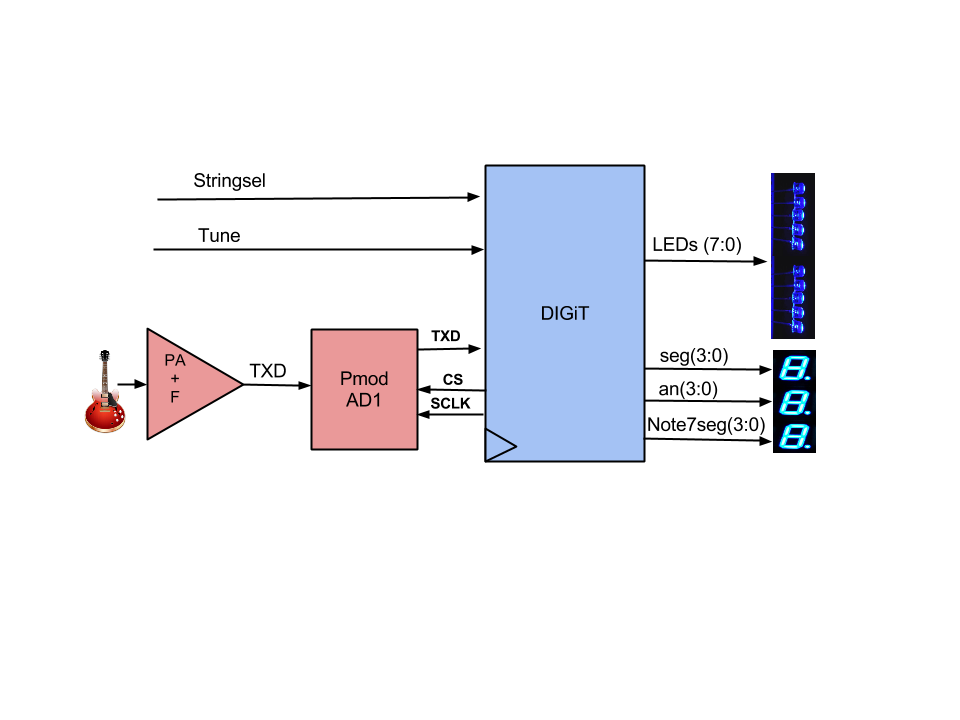
**Table of Contents**

1. **Project Report:** 
   1. **Introduction: Pg. 3**
   2. **Design Solution: Pg. 3**
      1. **Specifications: Pg. 3**
      2. **Operating Instructions: Pg. 4**
      3. **Theory of Operation: Pg. 4**
      4. **Construction and Debugging: Pg. 5**
   3. **Justification and Evaluation of design: Pg. 8**
   4. **Conclusion: Pg. 10**
   5. **Acknowledgements: Pg. 11**
2. **Appendix**
   1. **Appendix A: Control Panel**
   2. **Appendix B: Top Level Block Diagram**
   3. **Appendix C: Top Level Logic Block Diagram**
   4. **Appendix D: Serial Receiver Block Diagram**
   5. **Appendix E: Sample Rate Generator Block Diagram**
   6. **Appendix F: fSCLK\_GEN Block Diagram**
   7. **Appendix G: Comparator Block Diagram**
   8. **Appendix H: Freq\_Calc Block Diagram**
   9. **Appendix I: Package Map**
   10. **Appendix J: Top Level Schematic Diagram**
   11. **Appendix K: U2 Analog Conditioning Package Map**
   12. **Appendix L: Parts List**
   13. **Appendix M: Controller State Diagram**
   14. **Appendix N: Serial Receiver State Diagram**
   15. **Appendix O: VHDL Code** 
       1. **Top Level**
       2. **Controller**
       3. **Serial Receiver**
       4. **Frequency Calculator**
       5. **Comparator**
       6. **sCLK Generator**
       7. **Sample Rate Generator**
       8. **Sine Wave Testbench**
   16. **Appendix P: Simulation Waveforms**
   17. **Appendix Q:AD7476A Datasheet**

**D.I.Gi.T**

1. **Introduction**: Quickly and accurately tuning a guitar by ear is a difficult and tedious task, especially on stage or in a noisy environment, making automated tuners an important tool for any guitarist. Tuning a string to a desired pitch involves detecting the frequency of the input signal, comparing this to the frequency of the desired pitch, and then providing the user with corrective tuning feedback. However, a guitar signal is not perfectly sinusoidal; the fundamental frequency is masked by the presence of harmonics and overtones that result from the construction of the guitar and the physics of a vibrating string. Thus to effectively provide tuning feedback, a tuning device must be able to receive the raw analog signal, convert it do the digital domain, and detect the fundamental frequency amidst the numerous other frequencies present.
2. **Design Solution**:
   1. **Specifications:** *DIGiT* takes a conditioned analog guitar signal, calculates the fundamental frequency to determine the pitch of the note, and then compares this value to the known “in-tune” frequency of the string specified by the user. When the tuner is enabled, the user first selects a string to be tuned by flipping the corresponding switch on the FPGA and then plucking the string. Once a string is selected, instantaneous tuning feedback is displayed on the board’s eight LEDs, telling the user if the note being played is flat, sharp, or in tune. DIGiT’s overarching functionality can be lumped into three concurrent processes: *analog conditioning;* *conversion, calculation, comparison (CCC)*; and *user interface*. *Analog conditioning* consists of passing the analog input signal through an OP-Amp circuit and low-pass filter to amplify the voltage to a readable level and eliminate noise. *CCC* involvesconverting the analog amplitude to a 12-bit resolution digital sample via Pmod AD1 analog-to-digital converter, calculating the fundamental frequency of the signal through an algorithm that utilizes the principle of hysteresis to distinguish harmonics and measure the period between a known number of voltage threshold crossings, and finally comparing this period to that of the desired string. Finally, the *user interface* consists of six string selection switches, a tune enable switch, eight LED’s for tuning feedback, and three 7-segment LED displays that display the state of the system, all of which are present on the Basys 2 board.
   2. **Operating Instructions:** (For Control Panel see Appendix A)
      * **Setup**
        1. Connect ¼ inch guitar cable to breadboard wired with Op-Amp and Low pass filter via adapters: 1/4inch guitar cable 🡪 1/4inch to RCA 🡪 RCA to 6-pin 🡪 Breadboard
        2. Connect 6-pin output of Breadboard to input of Pmod AD2, such that the signal is wired to the A0 input pin on the AD2
        3. Connect 6-pin AD2 output to JA Pmod socket on Digilent Basys 2 FPGA, and program the board
      * **Tuning**
        1. Flip enable switch (SW0) to enable string selection. 7-segment display will change from *OFF* to *SEL*.
        2. Flip one single string select switches (SW7-SW2) to begin tuning, and then pluck string. 7-segment displays will show selected string and LEDs will provide tuning feedback (LD7-6: flat, LD5-2: in tune, LD1-0: sharp). **Optimal performance occurs when string is plucked softly with thumb to eliminate pick attack.** If more than one string is selected, 7-segment displays will show *000* error message.
        3. Bringing tune enable switch (SW0) low at any point in the process will disable the tuner.
   3. **Theory of Operation**: As previously stated, functionality can be lumped into three concurrent processes: *analog conditioning* –encompasses analog signal processing before digital conversion*;* *conversion, calculation, comparison (CCC)* – encompasses digital conversion and execution of original frequency detection and comparison algorithms utilizing the FPGA; and the *user interface* – consisting of input feedback and the FPGA 7-segment displays.

* ***Analog conditioning***: refers to the conditioning of the analog guitar signal to prepare it for digital processing. This phase is critical to DIGiT's functionality because passive single-coil guitar pickups must be amplified to a certain level to interface correctly with the ADC. Amplification is achieved through an Op-amp circuit wired on a Pmod Bread Board (PBB) with adjustable gain. The signal is then run through a single-pole low pass filter wired in series with Op-Amp on the PBB to help filter out noise, overtones, and higher order harmonics. The conditioned signal is then fed into the AD2. A more detailed description of the construction of these components is provided in Section 2.4.
* ***CCC***: This represents the bulk of the project, and encompasses the ADC and all digital signal processing elements including the serial receiver, controller, and frequency calculation and comparison, implemented as VHDL modules under a top level entity within the FPGA.
  + *Top Level:* The top level module is the overarching entity that houses each VHDL module. Figure 1 show the DIGiTs top level interface with the outside world. Inputs are a 25MHz clock (*CLK*), serial data stream from the ADC (*TXD*), and string selection and enable signals from the user. Outputs are the SPI clock (*sCLK*), chip select (*CS*) 7-segment display controls (*an, seg, note7seg*), and LED signals for tuning feedback. See Appendix O for VHDL code and diagrams.



**Top Level I/O VHDL Instantiation**

Port ( CLK : in STD\_LOGIC;

TXD : in STD\_LOGIC;

stringsel : in STD\_LOGIC\_VECTOR(5 downto 0);

tune : in STD\_LOGIC;

sCLK : out STD\_LOGIC;

CS : out STD\_LOGIC;

an : out STD\_LOGIC\_VECTOR (3 downto 0);

seg : out STD\_LOGIC\_VECTOR (0 to 6);

note7seg : out STD\_LOGIC\_VECTOR(3 downto 0);

LEDs : out STD\_LOGIC\_VECTOR(7 downto 0) );

Figure 1: Top Level Block Diagram

* + *Pmod AD2 ADC* (top level I/O *– CS, sCLK,* *TXD*): The ADC converts an analog value from 0-3.3V to a 12-bit binary value ranging from 0-4095. When *CS* is asserted low-true, bits are serially transmitted on the falling edge of input *sCLK* (output by the serial clock generator within the top level entity) to be read on a rising edge. Serial data is a 16-bit packet comprised of 4 leading zeroes and 12 data bits transmitted at a sample rate determined by the frequency of *CS*. For detailed timing specifications, refer to Appendix Q.
  + *Serial Receiver:* (top level I/O – *TXD*): The Serial Receiver receives serial data stream *TXD* from the ADC and converts it to a parallel data signal *rx\_data* within the entity, which is then output to the frequency calculator component. The receiver is ready to receive data when CS is asserted low, and emits a done signal *rx\_done\_tick* upon completion of data reception. See Appendix O.3 for VHDL code and diagrams.
  + *HLSM Controller* (top level I/O *stringsel, tune,* *CS*): The controller manages high level state machine (HLSM) operations and processes user input. When the tuner is enabled and a string selected, the controller asserts *CS* low, signaling the ADC to begin serial data transmission. If at any point *CS* goes high, the data stream will be aborted and data is considered erroneous. When the controller receives *rx\_done\_tick* upon completion of SPI transmission, the magnitude of the sample is compared with a pre-determined amplitude threshold value, and the controller outputs a pulse to the frequency calculator on the rising edge, establishing a time reference so the period can be measured. The controller also implements a hysteresis filter to help eliminate any contribution from harmonic frequencies, detailed in Section 3. See Appendix O.2 for VHDL code and diagrams.
  + *Frequency Calculator*: (top level I/O – N/A): This module receives the pulse signal (*freq\_pulse\_sig)* generated by the controller on the rising edge amplitude threshold crossing and compares it to the 25 MHz clock in order to calculate the period of the signal in nanoseconds. The process utilizes two running count signals: *curr\_count –* an up-counter that increments each 25 MHz clock tick, and *prev\_count –* a signal that stores the value of *curr\_count* on each pulse. Each time the module receives a threshold crossing pulse, the value of prev\_count is subtracted from *curr\_count,* the present value of curr\_count is stored in *prev\_count*, and then *curr\_count* continues incrementing. The difference between the two count signals represents the number of clock ticks that elapsed between pulses, providing a time reference to calculate the period/frequency. The period in nanoseconds (*PULSE\_PER*)is then output to the comparator module. See Appendix O.4 for VHDL code and diagrams.
  + *Comparator:* This module receives the pulse period in nanoseconds from the frequency calculator module and compares it to the “in tune” period of the selected string. For a string to be considered “in tune”, the frequency/period must be less than a half semi-tone above or below the “in tune” value of the note, whose upper and lower bounds are stored in a lookup table (LUT) within the module. The upper and lower boundaries are concurrently assigned values from the LUT based on user input and are used to calculate the relative pitch of the input inside the comparison process. The comparison process is executed on the rising edge of the 25 MHz clock, each time checking whether the input period is greater than, less than, or within the upper and lower boundaries for the given note stored in the LUT, and then updating the LEDs to display whether the note is sharp, flat, or in tune. See Appendix O.5 for VHDL code and diagrams.
  + *sCLK Generator* (top level I/O – *CLK, sCLK*): This module generates the serial clock utilized in SPI operations. It consists of a simple clock divider, taking an input 25 MHz clock tick and dividing it in half to 12.5 MHz This frequency was chosen due to its simple implementation and that it facilitates the calculation and comparison of binary period values since binary division is executed with a simple left shift. See Appendix O.6 for VHDL code and diagrams.
  + *Sample Rate Generator:* (top level I/O – N/A): This module generates the sample rate start signal, emitting a pulse one clock cycle in duration for every 500 ticks of the 25 MHz clock. See Appendix O.7 for VHDL code and diagrams.
* ***User Interface:*** The user interface utilizes three 7-segment displays to inform the user of the current mode of the tuner. When *tune enable* is off, the tuning LEDs are all lit and the 7-segment displays show *“OFF”* (Fig. 1.1).When *tune enable* is switched on, the tuning LEDs remain lit and the 7-segment displays show *“SEL”* (Fig. 1.2).When a single string is selected, the 7-segment displays show the selected string, and tuning LEDs display feedback (Fig. 1.4-6). If more than one string is selected or an error occurs, the LEDs are all lit and the 7-segment displays show “*000”* (Fig. 1.3).

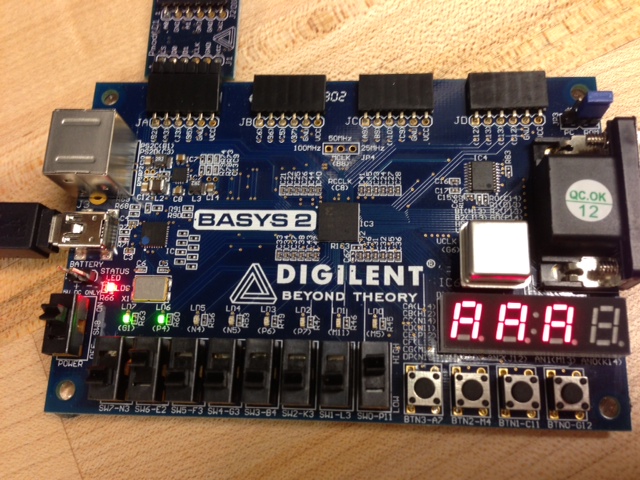
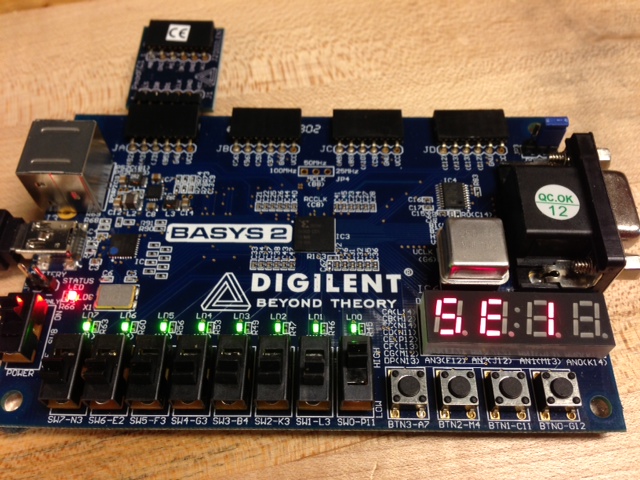
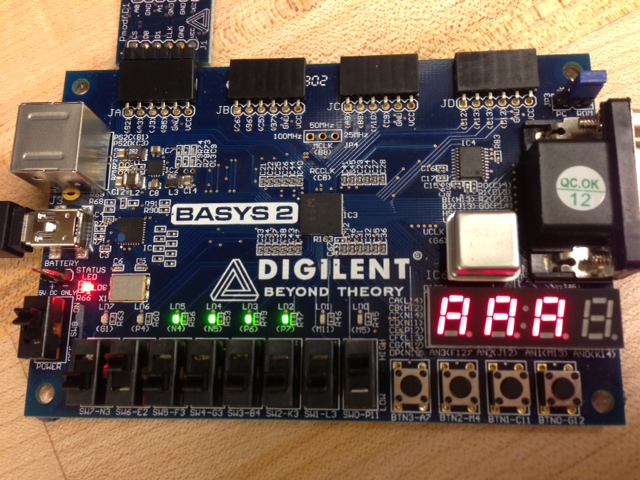
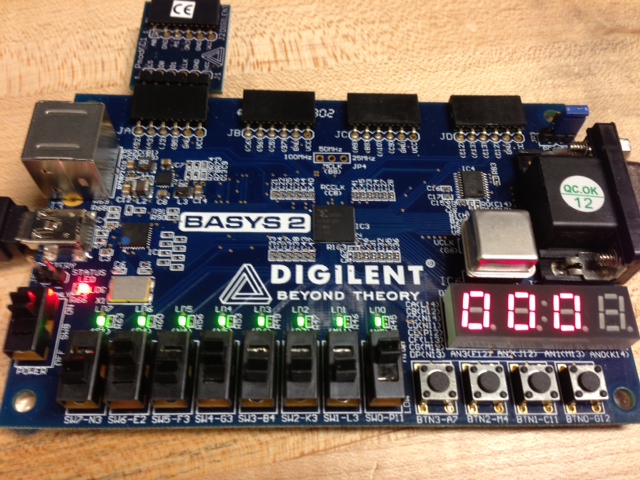
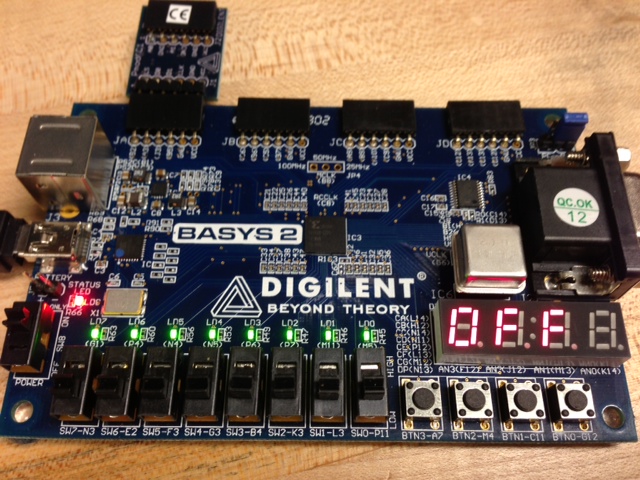
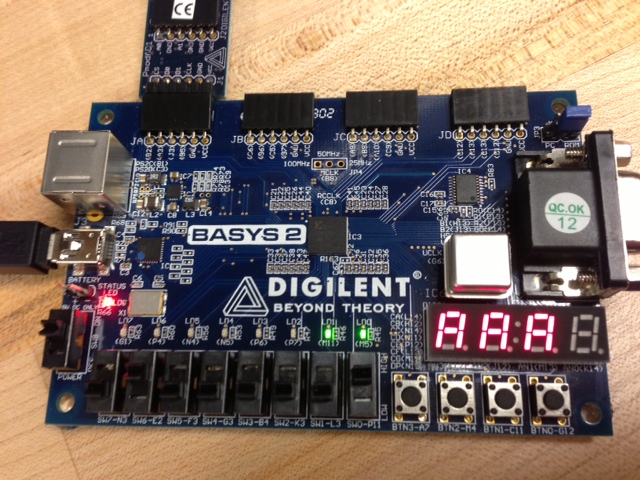


Fig 2 - User Interface

**Fig 1.1**

**Fig 1.2**

**Fig 1.3**

**Fig 1.4**

**Fig 1.5**

**Fig 1.6**

* 1. **Construction and Debugging:**
     + *Amplifier Construction:* A typical passive electric guitar signal has an output amplitude range of 0-250mV, however the Pmod AD2 ADC has a dynamic input range of 0-3.3V. To properly interface with the ADC and provide adequate data resolution at a 50 kHz sampling rate, the signal is amplified by an Op-Amp circuit wired on a Pmod Bread Board (PBB). Gain is precisely controlled by adjusting the value of the parallel resistor, chosen to be 0.56 MΩ. For detailed schematics, reference Appendix K.
     + *Filter Construction:* The signal is then run through a single-pole low pass filter in series with Op-Amp on the PBB to help filter out noise, overtones, and higher order harmonics. The cutoff frequency of the filter is controlled by adjusting the value of the parallel resistor – set to 15 kΩ as to yield a cutoff frequency of 482 Hz, which is somewhat above the frequency of the high E string (329.6Hz). This conditioned signal is then fed into the ADC as seen in the top level state diagram in Appendix B, and the analog conditioning package map in Appendix K.
     + *Analog Debugging:* Analog component parameters were set based on respective formulas to yield the necessary gain and bandwidth values for each component. Initially, the preamplifier was constructed with a variable resistor value of 100 kΩ, but testing demonstrated severe amplitude clipping and minimum filter impact. Through experimentation, it was discovered that using a 0.56 MΩ resistor in order to increase gain beyond what was initially predicted yielded the optimum waveform.
     + *VHDL Debugging:* A significant challenge in implementing the tuner was managing the SPI interface with the ADC. Despite simulating correctly in software, the results of initial hardware implementation were highly erroneous. Diagnosing the problem was difficult due to the layer of abstraction added by hardware implementation despite successful simulation. A debugging strategy used involved the input of a function generator sine wave, and observing a series of HLSM flagging signals mapped to the output ports on the FPGA on the oscilloscope to determine in what state errors occurred. Results pointed to the serial receiver (RX) as likely source of error, and when the serial data stream was compared to the parallel output of RX on the oscilloscope it was found that the most significant bit of the RX output would oscillate once it crossed the threshold value instead of remaining high, indicating a SPI timing issue. The PmodAD1 employs two AD7476A chips for analog to digital conversion, requiring adherence to a strict SPI timing protocol for correctly data transmission (Fig. 3). It was found that the state logic of the tuner’s serial receiver (RX) was programmed such that when controller output *CS* was asserted low ‘0’ (triggered by a start tick from a clock divider pulse at the sample rate of 50 kHz) the first four zero bits would be shifted on the rising edge of the SPI clock (*sCLK*) into a shift register. Errors occurred because *CS* was not synchronous with the rising edge of *sCLK*, and so could go low any point in between rising edges. Thus when *CS* was asserted low, the bit shifted on the first rising edge of *sCLK* was actually the second of four zero bits but incorrectly assumed to be the first, causing a series of asynchronous errors that resulted in improper shifting and erroneous data. This was corrected by adding two extra states in the controller state machine to synchronize *CS* with the rising edge of *sCLK*, ultimately shifting a15 bit packet comprised of 3 zero start bits and 12 data bits into the parallel load register on the rising edge of the *sCLK* clock. For the AD7476A datasheet and detailed timing diagrams, see Appendix Q.

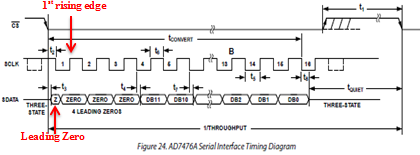


Figure 3

1. **Justification and evaluation of design**: The key design aspect of DIGiT is the implementation of a hysteresis threshold filter in conjunction with the amplifier/low-pass filter combination to detect a rising edge threshold crossing. Each time the amplitude sample crosses the established hysteresis threshold, the controller outputs a pulse used by the frequency calculator component to determine whether the note is in tune. DIGiT’s unique design is effective because it capitalizes on certain inherent aspects of the electric guitar signal that allows for the entire process to operate in the time domain. Due to its construction, the harmonic frequencies of an electric guitar provide dramatically less amplitude contribution to the signal than in many other instruments, and so with the right filtering algorithm, the frequency can be analyzed without the use of any Fourier analysis. Because the next harmonic is the next octave up (2x fundamental frequency) and higher order harmonics and overtones are eliminated by the analog filter (also reduces the amplitude of the harmonics), only a small frequency bandwidth needs to be analyzed for each string (Fig. 4). Thus the fundamental is determined by setting the hysteresis threshold at a level above the peak amplitude contribution of the second harmonic, but below that of the maximum amplitude of the signal.

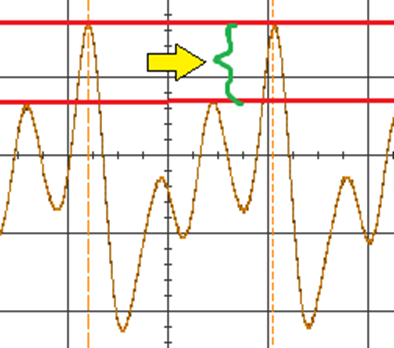


Figure 4: Hysteresis Threshold

1. **Conclusion:** The DIGiT tuner satisfactorily met its goal as a functioning electric guitar tuner. Testing demonstrated that the device could precisely tune the six strings of an electric guitar to within half of a semitone of the correct pitch. Despite its effectiveness, one possible drawback of the design is that it may not be universally compatible with every guitar. It is plausible that certain types of hum-bucking pickups or a guitar that employs active electronics could generate a signal with too heavy of a harmonic presence for DIGit to isolate the fundamental frequency. Unfortunately, due to the time and resource constraints, more than one guitar was not able to be tested. However, though it may not be universally compatible, DIGiT has the capability to rapidly and precisely tune most guitars and provide the user with real-time feedback and an elegant user interface.
2. **Acknowledgments:**

We would like to extend a huge thanks to Professor Hansen for his guidance and teaching skills and to Dave Pickard for his seemingly unlimited patience and interest in his students. Also a massive thank you to all of the T.A.’s for helping us when we needed it most , we really could not have done it without your hard work. A special mention should goes to Sarah Pasternak, who went above and beyond her requirements as a T.A. when she stayed in the lab past 11:30PM for two nights in a row to help us resolve an unexplained Xilinx issue that had been halting our progress. We owe you one.

**Workload Breakdown:**

Brett Nicholas

* Primary Programmer and HLSM Designer
* Wrote Theory of Operation and Operating Instructions procedure
* Contributed to design schematics in appendix

Dylan Sewell

* Designed and Constructed all analog components
* Assembled final design
* Secondary Programmer
* Created state diagrams
* Compiled written report and appendices
* Designed visuals