



12-2015

A Hardware Based Audio Event Detection System

Jacob Daniel Tobin

University of Tennessee - Knoxville, jtobin@vols.utk.edu

Recommended Citation

Tobin, Jacob Daniel, "A Hardware Based Audio Event Detection System." Master's Thesis, University of Tennessee, 2015.
https://trace.tennessee.edu/utk_gradthes/3612

This Thesis is brought to you for free and open access by the Graduate School at Trace: Tennessee Research and Creative Exchange. It has been accepted for inclusion in Masters Theses by an authorized administrator of Trace: Tennessee Research and Creative Exchange. For more information, please contact trace@utk.edu.

To the Graduate Council:

I am submitting herewith a thesis written by Jacob Daniel Tobin entitled "A Hardware Based Audio Event Detection System." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Mark E Dean, Major Professor

We have read this thesis and recommend its acceptance:

Garrett S Rose, Jon M Hathaway

Accepted for the Council:

Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

A Hardware Based Audio Event Detection System

A Thesis Presented for the
Master of Science
Degree

The University of Tennessee, Knoxville

Jacob Daniel Tobin

December 2015

© by Jacob Daniel Tobin, 2015
All Rights Reserved.

Abstract

Audio event detection and analysis is an important tool in many fields, from entertainment to security. Recognition technologies are used daily for parsing voice commands, tagging songs, and real time detection of crimes or other undesirable events. The system described in this work is a hardware based application of an audio detection system, implemented on an FPGA. It allows for the detection and characterization of gunshots and other events, such as breaking glass, by comparing a recorded audio sample to 20+ stored fingerprints in real time. Additionally, it has the ability to record flagged events and supports integration with mesh networks to send alerts.

Table of Contents

1	Introduction	1
2	Related Work	3
3	Methodology	9
3.1	Cross Correlation Matching Algorithm	9
3.2	MATLAB Simulation and Early Testing	11
3.3	Hardware Implementation	19
4	Performance and Resource Utilization	24
4.1	Number of Fingerprints	24
4.2	FPGA Resource Utilization	28
4.3	Performance	30
4.4	Power	31
5	Results	32
6	Future Work	36
6.1	Further Tests	36
6.1.1	Fingerprint SNR Considerations	36
6.1.2	Threshold, Sampling Frequency, and NFFT	37
6.2	Optimizations	38
6.2.1	System Performance and Fingerprints	38
6.2.2	Power Consumption	39

7 Deliverables	40
Bibliography	41
Appendix	44
Vita	50

List of Tables

4.1	Required FFT Cycles & Processing Time	25
4.2	Sampling Period for Various NFFT Lengths	26
4.3	Fingerprint Memory Usage	26
4.4	FFT Memory Usage	27
4.5	Max Fingerprints for Given NFFT Size (Neglecting Other Memory Usage)	28
5.1	9mm Fingerprint vs LC9	33
5.2	9mm Fingerprint vs LCP	34
5.3	9mm Fingerprint vs .22 Revolver	34
5.4	.380 Fingerprint vs LC9	34
5.5	.380 Fingerprint vs LCP	34
5.6	.380 Fingerprint vs .22 Revolver	35
5.7	.22 Fingerprint vs .LC9	35
5.8	.22 Fingerprint vs .22 Revolver	35

List of Figures

2.1	Detection rate vs number of sensors	7
3.1	Computations required for fast vs normal cross correlation	11
3.2	Comparison of 32 bit double vs 16 bit int signals (t domain)	12
3.3	Comparison of 32 bit double vs 16 bit int signals (F domain)	13
3.4	Ruger .22 Pistol Fingerprint and Sample	15
3.5	Ruger .22 Pistol Fingerprint and Sample With Traffic Noise	15
3.6	Ruger .22 Pistol Fingerprint and Ruger 10/22 .22 Rifle Sample	16
3.7	Ruger .22 Pistol Fingerprint and Ruger 10/22 .22 Rifle Sample With Traffic Noise	16
3.8	Ruger .22 Pistol Fingerprint and Colt 1911 .45 Pistol Sample	17
3.9	Ruger .22 Pistol Fingerprint and Colt 1911 .45 Pistol Sample With Traffic Noise	17
3.10	Ruger .22 Pistol Fingerprint and Marlin 336 .30-30 Rifle Sample	18
3.11	Ruger .22 Pistol Fingerprint and Marlin 336 .30-30 Rifle Sample With Traffic Noise	18
3.12	Ruger .22 Pistol Fingerprint and Marlin 336 .30-30 Rifle Sample With Additive White Gaussian Noise	19
3.13	MATLAB and FPGA Generated Cross Correlations	20
3.14	Absolute Value of MATLAB and FPGA Generated Cross Correlations	20
3.15	Overlap Scheme for Audio Sampling	22
3.16	Block Diagram of FPGA System	23
4.1	Processing Time Required for Sample Comparison	25

4.2	BRAM Required for System	27
4.3	Resource Utilization of XC7A100T FPGA	29
4.4	DSP48 Slice	30
6.1	Training and Test SNR	37

Chapter 1

Introduction

The majority of audio event detection systems are software based, and, in many cases, rely on networking to send audio back to a central location for processing. The system described in this work is Field Programmable Gate Array (FPGA) based, and all processing occurs at the node. It is designed to interface with environmental sensor arrays which themselves will monitor air quality, temperature, light levels, etc. in urban environments. When possible, the arrays will use power supplied from light posts, buildings, or other sources, and will be mounted to existing infrastructure. The sensor arrays will communicate via a mesh network, although this is not required for the event detection system to function. Related works are discussed in chapter 2.

A fast cross correlation algorithm, discussed in chapter 3, is used for event detection. It takes advantage of an FPGA's ability to perform large operations in parallel and to pipeline the system to minimize delays. The detection system loads a predefined library of audio fingerprints onto an FPGA which then constantly monitors the nearby environment for occurrences of these events. The system is intended to capture impulsive events, such as gunshots, breaking glass, etc. When an event is detected, the system stores the buffered audio for further analysis and, depending on the configuration, sends an alert via a mesh network. This alert tells other sensor nodes to store the contents of their buffers. The recorded audio may be analyzed to

determine the approximate location that an event occurred via triangulation, based on relative amplitudes between different nodes with known locations. This latter application is not discussed in this paper, but could be implemented using known algorithms.

The FPGA used in this project is a Xilinx Artix 7 XC7A100T. The system is not limited to this specific board, and other FPGAs would also work. A large part of the design does depend on Xilinx IP cores, however, so utilizing a different Xilinx FPGA would be simplest and would not require any additional modules to be created. The only change necessary would be modifications of the constraint file to set the pins used for external interfacing. There could be a limitation on the number of fingerprints which could be stored on a different FPGA, depending on the amount of available memory on the FPGA. Resource utilization and requirements relating to this are discussed in [chapter 4](#). Results from a test of the system against real audio events are shown in [chapter 5](#), and further improvements and additional capabilities are examined in [chapter 6](#).

Chapter 2

Related Work

The number of audio event detection systems which both sample and perform data analysis at the node is fairly small. The number which implement this feature at a hardware level is even smaller. The majority are software based and only utilize the node as a means to record audio, transmit data, and display the results.

One of the most well known audio analysis systems is Shazam (An Industrial Strength Audio Search Algorithm [1]). This application uses one's mobile device to record a snippet of a song. The sample is uploaded to a server which creates a spectrogram of the audio. Large frequency intensity peaks are picked out, and the distance (time) between them is noted, along with the difference in frequency. These points are then placed in a hash table. The resulting values are then compared to fingerprints of other songs by looking for similar frequency-intensity vs time relationships. When enough corresponding points have been detected, the system sends the name and details of the detected song back to the user. The system performs well even in noisy environments, as long as the corresponding frequency peaks can be detected. While the application is limited by internet access, it maintains a high accuracy and has an extensive song catalog.

In the same vein as the method used by Shazam, but taking advantage of the parallel capabilities of an FPGA, is a music melody identification system (An FPGA

based parallel architecture for music melody matching [2]) which uses string matching. It allows for the matching of a sung or hummed melody, which is saved as a WAV file at a sampling frequency of 8kHz, to a database of MIDI files. A personal computer (PC) converts the user sung query into a regular expression, while the FPGA handles the back end processing and matching. The string formatting is fairly interesting; a MIDI file consists of a table of notes at different times. To create a more easily searchable format, the table is converted to a string of characters. Each character lasts a predetermined amount of time, and the pitch is its ASCII value (0-127). For example, "Happy Birthday" can be approximated by the following string: <<<<<<<>>>>>>>>><<<<<< AAAAAA@@@@@@@@@@@@@@@@. The pitch during a certain period of a recorded segment is estimated using a fundamental frequency estimator called YIN (YIN, a fundamental frequency estimator for speech and music [3]). To account for variances in recorded pitch versus stored pitch, the system shifts the recorded pitch over a range of nearby pitches before comparing each one to the database recording. A list of the 10 closest MIDI files are presented based on the minimum amount of change required to match the recorded sample to a file from the database (called edit distance), and a successful retrieval is defined as any list which contains the original song, regardless of position in the list. The system maintained 90% accuracy while checking a database of 5569 files in 19.4 seconds, which works out to an average comparison time of 54.6ms per query. While this is acceptable for matching nonrecurring events, it would not lend itself well to a real time monitoring system.

Another FPGA-based music retrieval algorithm is described in a paper titled "FPGA Implementation of Content-Based Music Retrieval Systems" [4]. Similarly to the implementation discussed in the music melody identification system [2], this one uses string matching of MIDI files and a scoring system based on the edit distance between the recorded sample and the stored library sample. Unlike the other implementation, however, this one uses a soft CPU, which is a CPU created from the reconfigurable logic of an FPGA as opposed to purpose built hardware, to interface

with the string matching hardware. The soft CPU runs at 50MHz while the matching hardware runs at 422MHz, which is the maximum operating frequency of the Altera Stratix EP1S40 FPGA used. A 2MB database contains strings representing 37,536 songs. While no mention is made of the accuracy, the total CPU time to check the 2MB database is 1,231 milliseconds. This does not give the total amount of time required for the hardware matching algorithm to run, so the total run time of the system cannot be determined.

An audio event detection system for screams and gunshots was presented in a 2007 paper (Scream and gunshot detection in noisy environments [5]). The described system uses a combination of 47 temporal, spectral, and perceptual features, such as zero crossing rate, spectral distribution, and Mel Frequency Cepstral Coefficients, respectively. Features which may be overly sensitive to the SNR, such as short time energy and loudness, are not used. The system starts by taking auto correlations of each frame, which are 23ms long and sampled at 22.05kHz, to find the energy distribution over different time lags. Much of the energy in a gunshot, which is highly impulsive, is contained in the first time lags. Screams, on the other hand, are spread over a larger number of time lags. While the number of features is large, only certain ones are used depending on their estimated usefulness. Past a certain point, the gains from including additional features begin to saturate while processing time increases. Training is performed for both gunshots and screams, although there is little explanation of the training process. As expected, the SNR of both the detected signals and the training signals affects the false positive and false negative rates. Resulting tests show that training with high SNR and testing with low SNR leads to a lower overall accuracy of about 15% vs a maximum accuracy of 90%. Best results were achieved using training samples with a SNR of 10dB and 15dB for screams and gunshots, respectively. Implications of this are discussed in 6.1.1.

The Vanderbilt based Institute for Software Integrated Systems has been successful in designing counter-sniper and weapon classification systems for urban environments since 2004, as described in "Sensor network-based countersniper

system” [6], ”Multiple simultaneous acoustic source localization in urban terrain” [7], ”Shooter localization and weapon classification with soldier-wearable networked sensors” [8], and ”Acoustic shooter localization with a minimal number of single-channel wireless sensor nodes” [9]. The first system that they developed is called PinPtr [6][7]. It utilizes distributed, time-synchronized, networked sensor nodes to detect both muzzle blasts and shock waves, send the events back to a base station, and then calculate the shooter location. Its average accuracy is roughly 1 meter with a 2 second turnaround time. The detection hardware is composed of a Mica mote based around an ATmega microcontroller to handle messaging, time management, etc. and a Xilinx Spartan II FPGA to process incoming audio data. The algorithm used for detection is based on zero crossing and signal amplitude over time. Muzzle blasts and shockwaves are detected using finite state machines (FSMs), based on the signal amplitude, length, and fall and rise times. Based on the difference in time between multiple motes detecting events, the location of the shot can be determined. The system requires a relatively high number of motes to accurately detect an event, as can be seen in figure 2.1. It should be noted, however, that a shot is considered undetected if less than six sensors detect an event. No mention is made of the ability of a single node to detect an event since the system is meant to be used with multiple networked motes to determine the shooter location.

In a 2007 paper [8], improvements to the above system were described. The microphone array was turned from a static object into a helmet mounted version, with the addition of a 3 axis compass for orientation and bluetooth communications to send data to a PDA. At its core, the system still uses the zero crossing FSM to detect events based on rising and falling times over specific time periods. Additionally, since the period of a bullet’s shockwave is based on its length and diameter, the caliber can be estimated based on detected characteristics. The group also determined that the muzzle blast of a weapon is not unique enough to determine the type of weapon fired, since environmental reflections and interference have more of an influence on the acoustic signal than the actual weapon does. However, speed and caliber can still

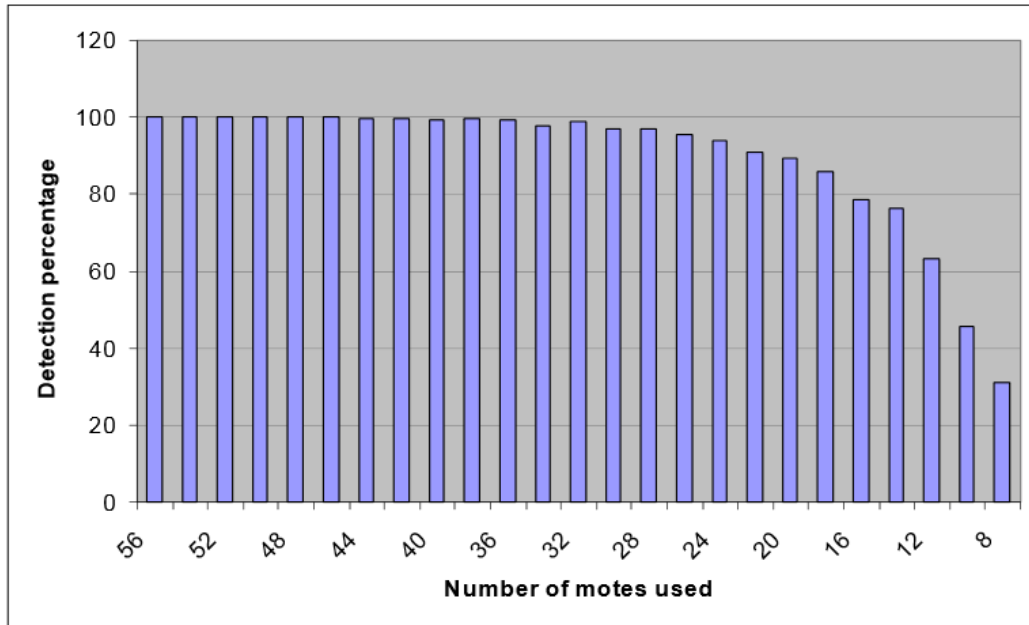


Figure 2.1: Detection rate vs number of sensors [6]

be used to estimate the weapon type, especially for common weapons. This paper also mentions the fact that a single sensor is able to detect and locate a shooter if at least 3 microphones on a node detect the muzzle blast and shockwave’s time of arrival. Another paper [9] was published in 2011 in which improvements in accuracy and weapon classification using fewer nodes were discussed. Overall, this seems to be one of the most interesting applications in addition to being similar to this project in that detection is performed using an FPGA.

There are two other gunshot detection systems which are currently being used by the military and police departments, although little information is available on the methods that they use for detection. The first is called ShotSpotter [10]. It is marketed towards police departments, college campuses, and commercial buildings. It claims to be able to detect a wide range of ”sharp acoustic events ... explosions, subsonic, supersonic gunfire.” The system uses ”spatial filtering,” which is the placement of sensing nodes at large enough distances that non gunshot or explosive sounds such as construction noise or traffic will not reach multiple sensors, and

therefore will not trigger a detection and cause false positives. It also allows for real time monitoring and triangulation to alert users to the location of event occurrences. A large amount of the information available is marketing related, however, and there is no information available regarding the accuracy, power, cost, etc. of the system.

The other detection system is named Boomerang and was developed by DARPA and BBN Technologies, which is a subsidiary of Raytheon, for usage in areas of military operation. It uses an array of microphones to detect gunfire in both urban and non-urban environments. Boomerang allows for detection to occur while being used by individual soldiers, ground based transport vehicles, and helicopters. The system designed for soldiers is referred to as Boomerang Warrior-X [11] and compensates for the individual's movement. Similarly, Boomerang III [12], which is the ground vehicle mounted system, and Boomerang Air [13], which is the helicopter mounted system, filter out events such as vehicle vibrations, outgoing weapons fire from that location, wind, doors closing, and other sounds. BBN claims a shot detection rate of 95% of all supersonic projectiles [14], with false positives occurring at a rate of less than one per every thousand hours of operation. It is able to detect and locate the origin of gunshots whose projectiles pass within about thirty meters of the microphone array. This indicates that the system depends on the bullet's supersonic shockwave for detection. From the available information on Boomerang, it seems to work similarly to the PinPtr system developed by Vanderbilt, although the hardware and detection methods are unknown.

Chapter 3

Methodology

3.1 Cross Correlation Matching Algorithm

The cross correlation of two functions provides an estimation of the similarity between the functions. Discrete time domain cross correlation is given as:

$$f[n] \star g[n] = \sum_{m=-\infty}^{\infty} f^*[m]g[m+n] \quad (3.1)$$

Time domain cross correlation requires $O(N^2)$ operations, where N is the number of samples, as each sample in f must be "slid" past each sample in g . This operation must be completed serially and is relatively slow. A quicker method of computing discrete cross correlation is to use fast cross correlation. This is done by transforming f and g to the frequency domain (equations 3.3 and 3.4) using a Discrete Fourier Transform (DFT), multiplying the cross conjugate of one of the frequency domain signals by the other signal (equation 3.5), then computing the Inverse Discrete Fourier Transform (IDFT) to yield the cross correlation of f and g (equation 3.6).

$$y[n] = f[n] \star g[n] \quad (3.2)$$

$$\mathcal{F}\{f[n]\} = F[k] = \sum_{n=0}^{N-1} f[n]e^{-j2\pi kn/N} \quad (3.3)$$

$$\mathcal{F}\{g[n]\} = G[k] = \sum_{n=0}^{N-1} g[n]e^{-j2\pi kn/N} \quad (3.4)$$

$$Y[k] = F[k]^*G[k]^{\text{I}} \quad (3.5)$$

$$y[n] = \mathcal{F}^{-1}\{Y[k]\} = \frac{1}{N} \sum_{k=0}^{N-1} Y[k]e^{j2\pi kn/N} \quad (3.6)$$

Once the cross correlation of $f[n]$ and $g[n]$ has been computed, the result is checked against a threshold. If the cross correlation contains values which are over that threshold, a match is made. If gunshot samples are being compared to recorded events and multiple gunshot samples are matched to a single sample, the system can rank the gunshots in order from that with the highest resulting cross correlation value to the lowest. While this can not tell the user that a detected gunshot definitely came from a specific type of gun, it gives an indication of the most likely type.

The benefit of using fast cross correlation instead of normal time domain cross correlation can be seen in figure 3.1. Utilizing a Fast Fourier Transform (FFT) algorithm to compute the DFT requires $O(N\log_2(N))$ operations. Multiplying the resulting frequency domain signals requires an additional N operations. Additionally, the N length signal must be zero padded to twice its original length. This is required as the fast cross correlation method uses circular cross correlation, where the signals wrap at the ends. Adding the zeros "removes" the overlap and simulates an infinite signal. This brings the total operations required to compute fast cross correlation to $O(2(2N)\log_2(2N) + 2N)$.

^INote: * refers to the complex conjugate, not multiplication

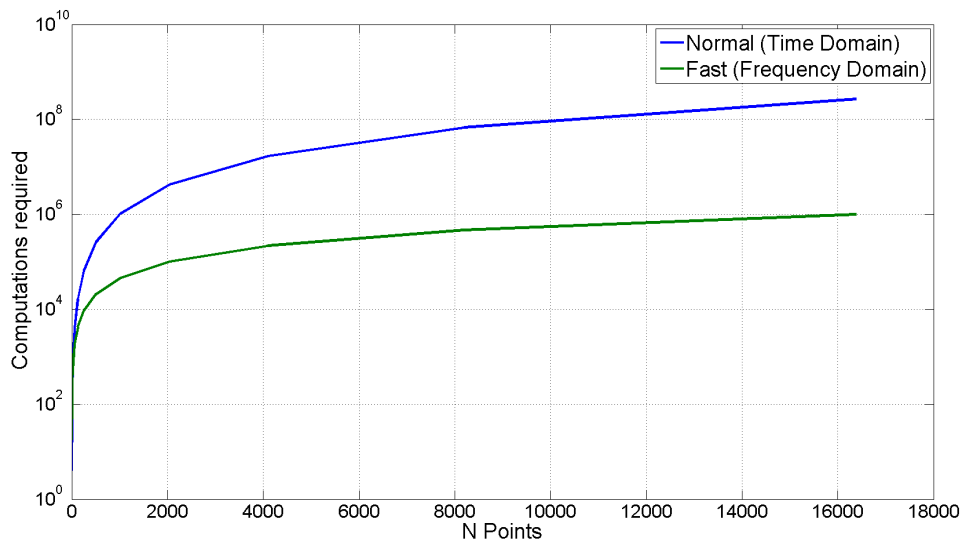


Figure 3.1: Computations required for fast vs normal cross correlation

3.2 MATLAB Simulation and Early Testing

Before work on the hardware implementation began, a number of tasks needed to be completed. The first was to develop a working simulation to verify that the cross correlation matching algorithm would perform as desired, in addition to determining parameters such as the number of points to collect during each sampling period and the sampling frequency. This was done in MATLAB, which provides a number of useful toolboxes for digital signal processing, serial communications, etc.

Recordings used to test the simulation were acquired from The Free Firearm Sound Effects Library [15], which contains roughly 280 raw gunshots from 24 different guns. The recordings are 32 bit doubles, sampled in stereo at 192kHz. Gunshots were chosen for the initial testing phase as they are consistent sounds with narrow spectrums. Recordings of broken glass, animal calls, etc. are not as consistent and would have made initial testing more difficult.

The audio signal is captured by a microphone and converted to a signed 16 bit number by an analog to digital converter (ADC), which is then fed to the FPGA

(this will be discussed further in section 3.3). When creating the fingerprints, the recordings from the Free Firearm Sound Effects Library are converted from 32 bit doubles with a range of -1 to 1 into 16 bit integers with a range of -32768 to 32767. This is done for two reasons: the first is to fully utilize the input range of the Xilinx FFT IP core on the FPGA. The second is to allow for the audio fingerprints to approximate the signal received from the ADC as accurately as possible. Figure 3.2 shows the original time domain 32 bit double signal overlaid on the 16 bit integer signal (both results are normalized to 1 in order to show the overlap). There is no visible difference, and the root mean squared error (RMSE) (given in equation 3.7) of the two signals is 8.6×10^{-6} .

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2} \quad (3.7)$$

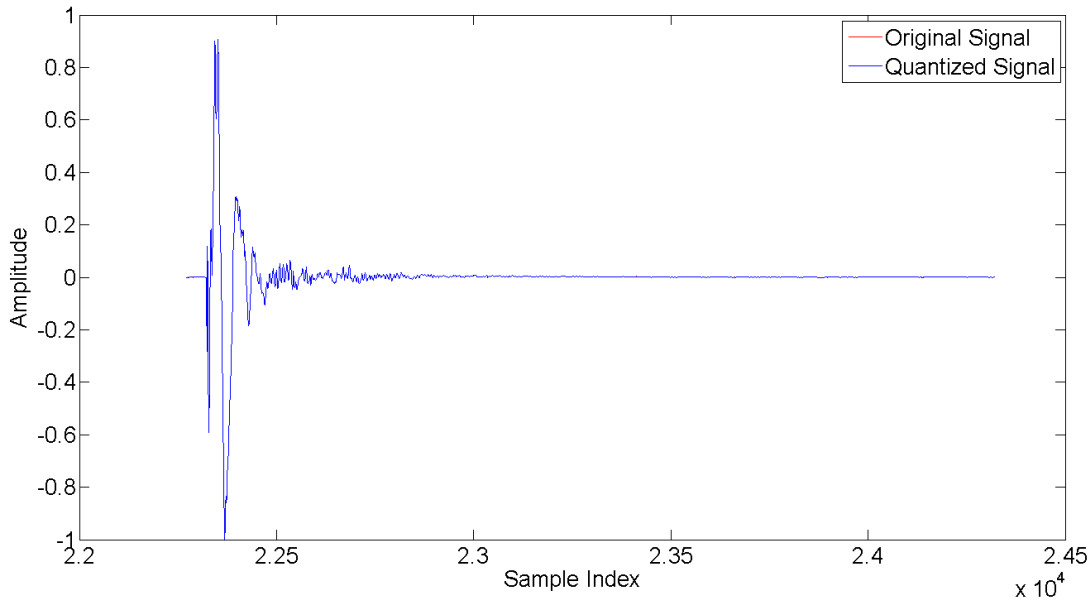


Figure 3.2: Comparison of 32 bit double vs 16 bit int signals (t domain)

Similarly, figure 3.3 shows the FFT of the original 32 bit double signal overlaid on the FFT of the scaled 16 bit integer signal. Again, there is no visible difference, and the root mean squared error of the two signals is 9.5×10^{-6} .

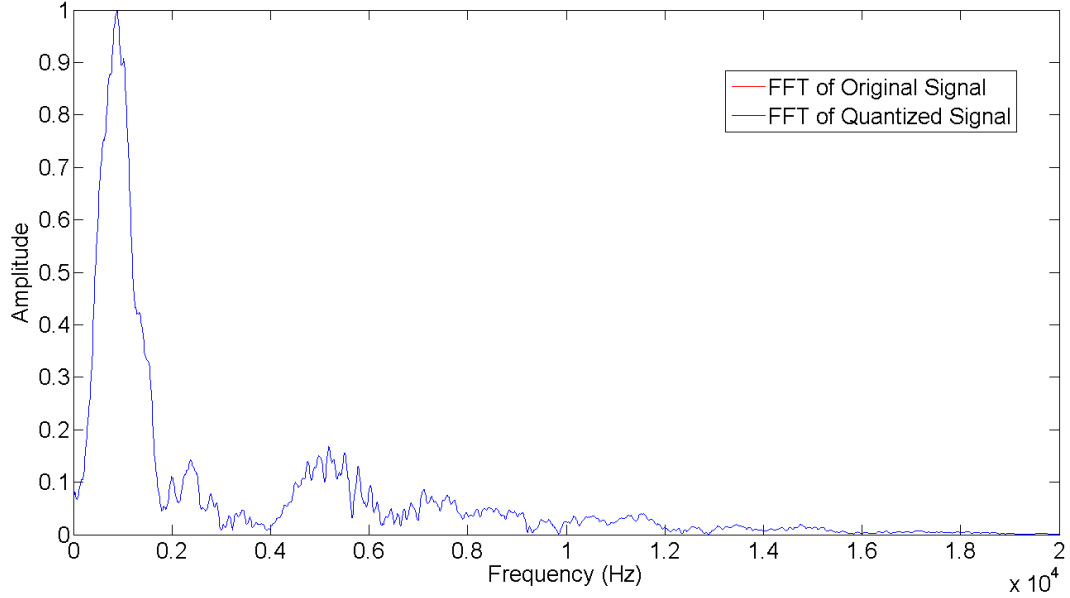


Figure 3.3: Comparison of 32 bit double vs 16 bit int signals (F domain)

In equation 3.5, $G[k]$ represents the FFT of the input signal, and $F[k]$ is the audio fingerprint. The audio fingerprints are created by zero padding a time domain sample, taking the complex conjugate of its FFT ($F[k]^*$), scaling it to fit into two 16 bit signed integers to represent both the real and imaginary parts, then converting the result to a binary representation to be stored in memory on the FPGA.

To test the matching simulation, both a fingerprint and raw gunshot sample are loaded, and, if desired, the raw gunshot recording is combined with other recordings, such as traffic noise or white noise, to simulate interference from the environment. Sampling frequency, FFT size ($NFFT$), and trigger threshold are set before running the simulation. The simulation loops through the audio track, grabs $NFFT/2$ samples, zero pads the samples so that the total length is equal to $NFFT$, takes the FFT of the padded sample, then takes the IFFT of product of the sample and the fingerprint (see 3.6). If the resulting cross correlation is greater than the threshold, then the index of the sample, FFT of the sample, and resulting cross correlation are saved. This repeats for both channels until the end of the audio file has been reached. The results are then plotted, as seen below.

Figures 3.4 and 3.5 show the results of a Ruger Mark III .22 pistol fingerprint compared against its original recording, in addition to the same recording with traffic noise [16] added. The top plot of each figure shows the original waveform, with the matching sections marked in red. The middle plot shows the resulting cross correlation at each matched section, while the bottom left plot shows the original fingerprint, and the bottom right shows the overlaid FFTs of both the fingerprint and the matching sections. Comparing the fingerprint against its original recording is more of a proof of concept than an actual test, as a match would be expected.

Figures 3.6 and 3.7 show the results of a Ruger Mark III .22 pistol fingerprint compared against a Ruger 10/22 .22 rifle. Even though the spectrum of the fingerprint differs from that of the sample, a match is still made. This suggests that a fingerprint for a specific caliber and type of gun will readily be able to detect a match against a different type of gun (e.g. handgun and rifle). Again, the algorithm is able to pick the gunshot out from the background noise.

Figures 3.8 and 3.9 show the results of a Ruger Mark III .22 pistol fingerprint compared against a Colt 1911 .45 pistol. Although they are different calibers, the spectrums of the two shots are very similar. This lends further support to the idea that a fingerprint from one type of gun may be used to detect the firing of a different type of gun.

Figures 3.10 and 3.11 show the results of a Ruger Mark III .22 pistol fingerprint compared against a Marlin 336 .30-30 rifle. Although this is both a different caliber and type of gun, the algorithm is still able to detect a match, as was suggested by the previous results.

From the collection of tests, it appears that the cross correlation based matching algorithm is robust and accurate, even in the presence of city noise or wind (modeled as white noise in 3.12). These results are promising and suggest the feasibility of a hardware implementation.

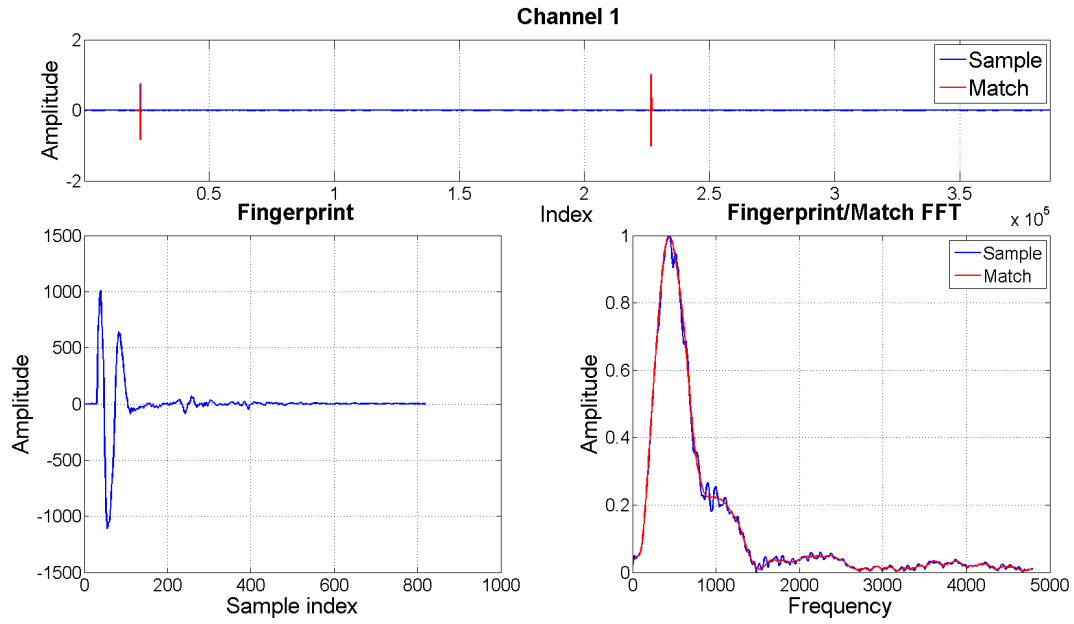


Figure 3.4: Ruger .22 Pistol Fingerprint and Sample

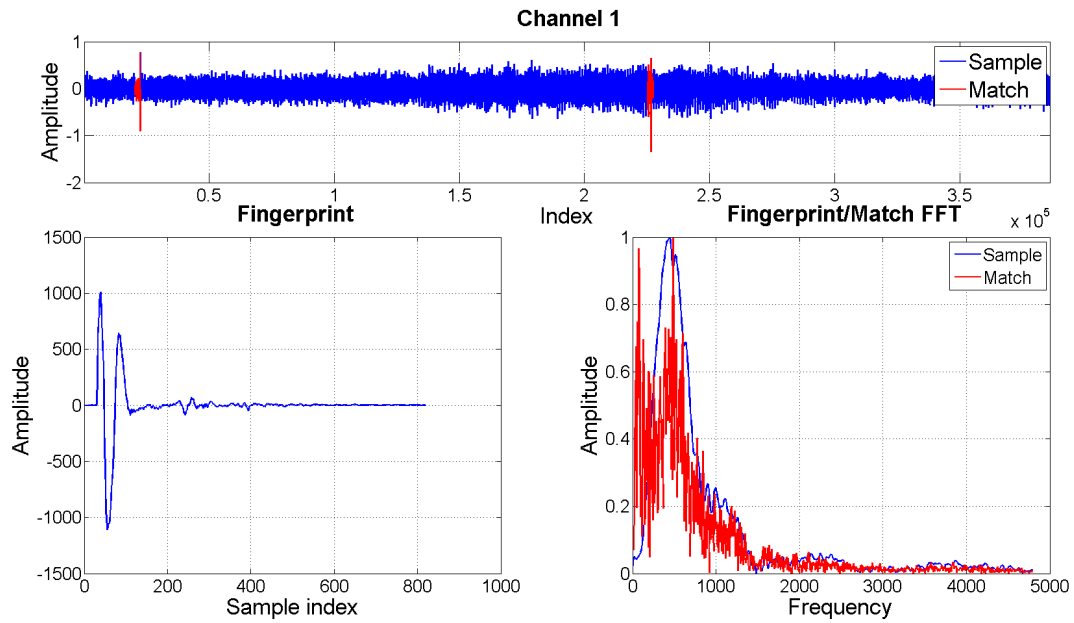


Figure 3.5: Ruger .22 Pistol Fingerprint and Sample With Traffic Noise

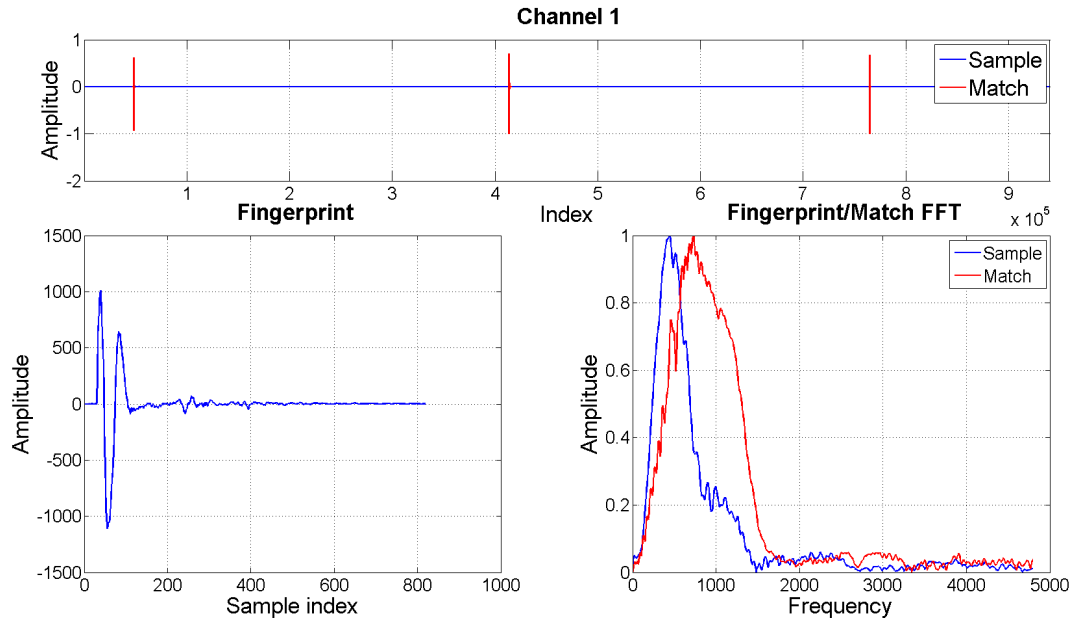


Figure 3.6: Ruger .22 Pistol Fingerprint and Ruger 10/22 .22 Rifle Sample

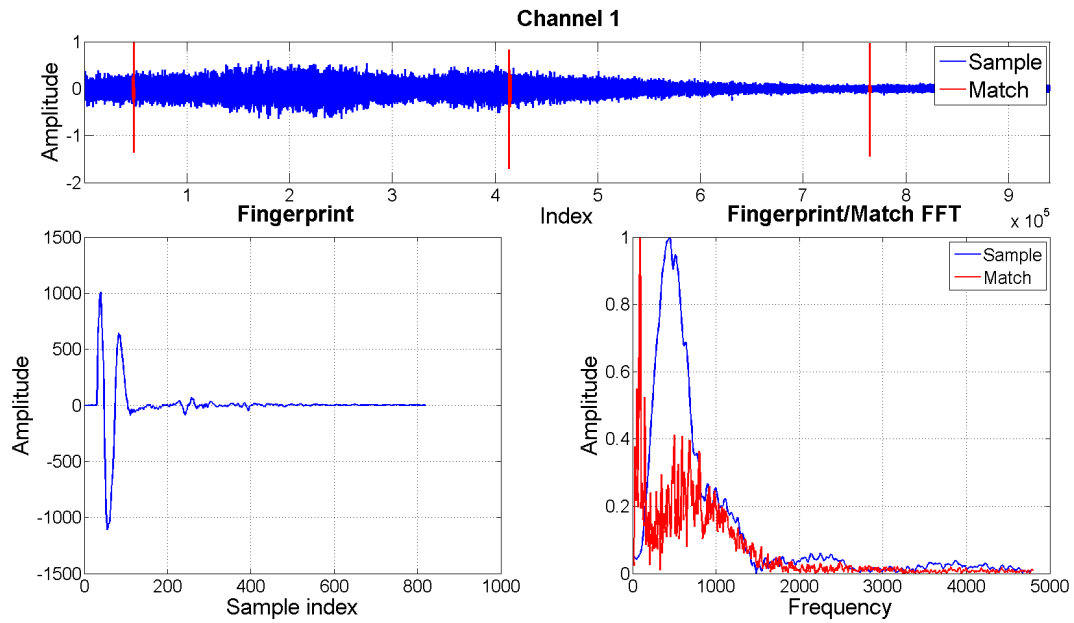


Figure 3.7: Ruger .22 Pistol Fingerprint and Ruger 10/22 .22 Rifle Sample With Traffic Noise

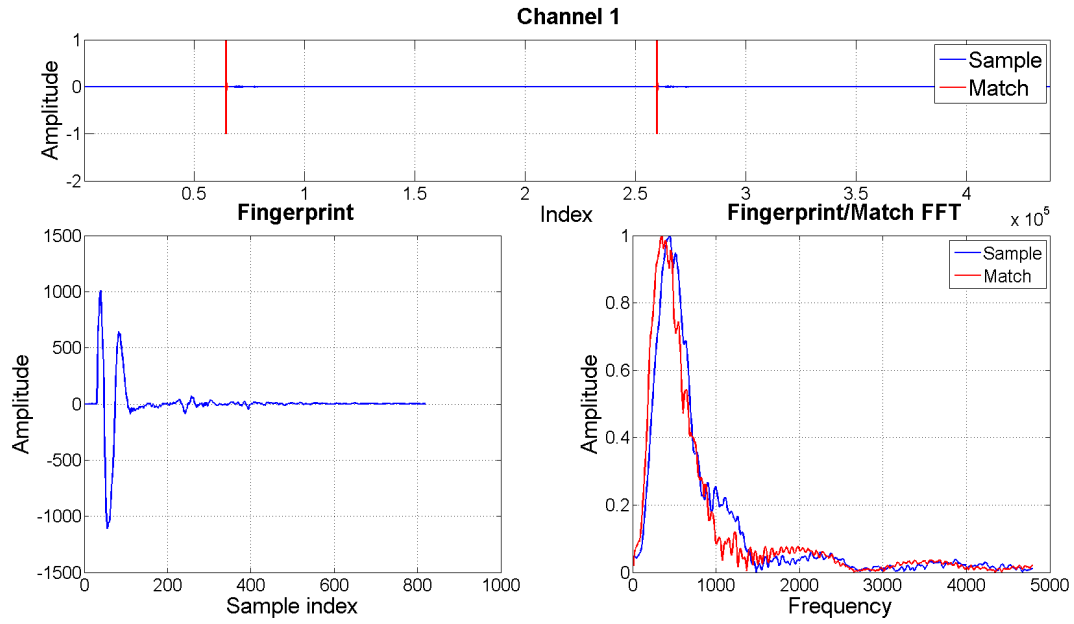


Figure 3.8: Ruger .22 Pistol Fingerprint and Colt 1911 .45 Pistol Sample

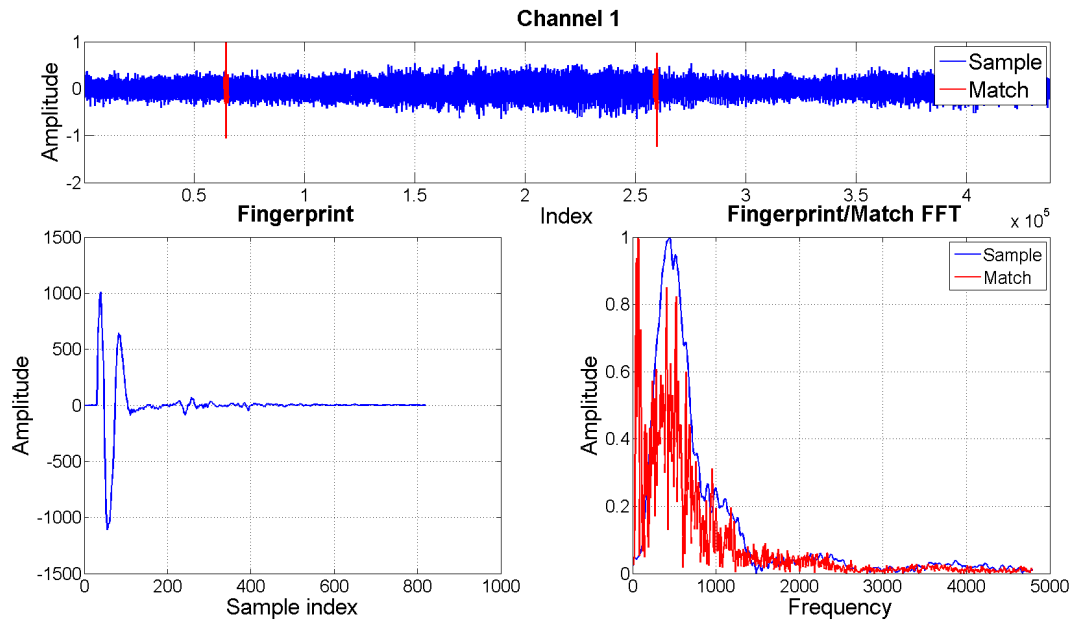


Figure 3.9: Ruger .22 Pistol Fingerprint and Colt 1911 .45 Pistol Sample With Traffic Noise

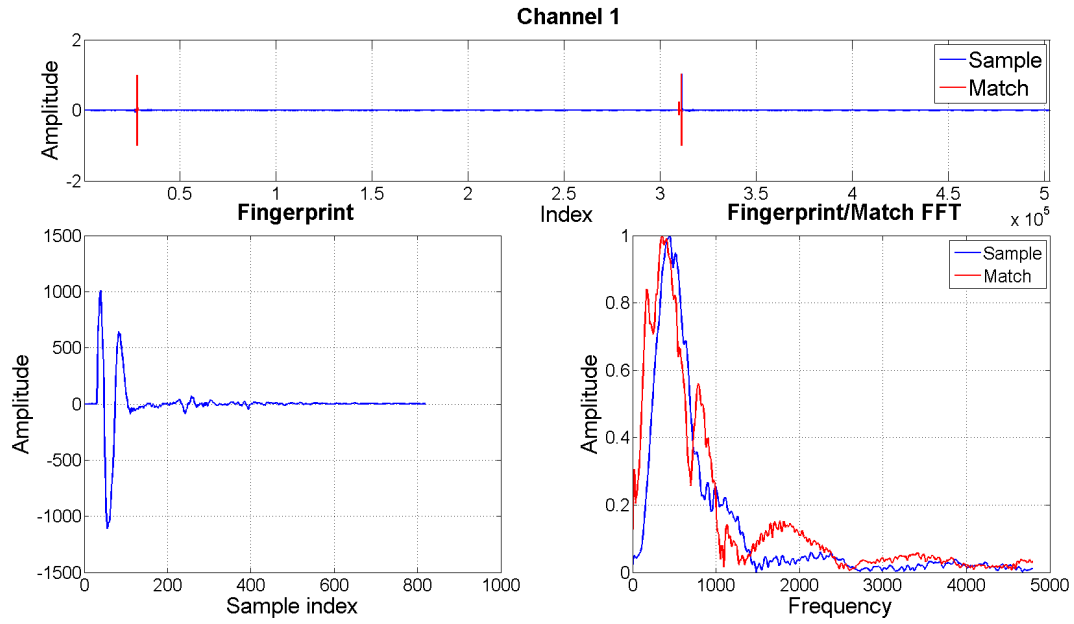


Figure 3.10: Ruger .22 Pistol Fingerprint and Marlin 336 .30-30 Rifle Sample

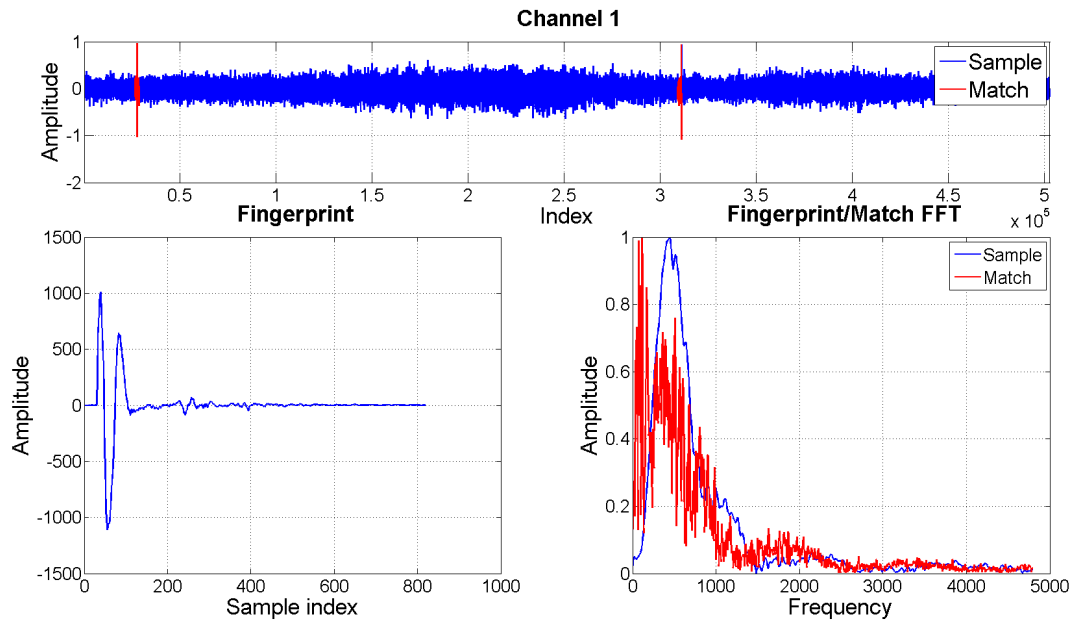


Figure 3.11: Ruger .22 Pistol Fingerprint and Marlin 336 .30-30 Rifle Sample With Traffic Noise

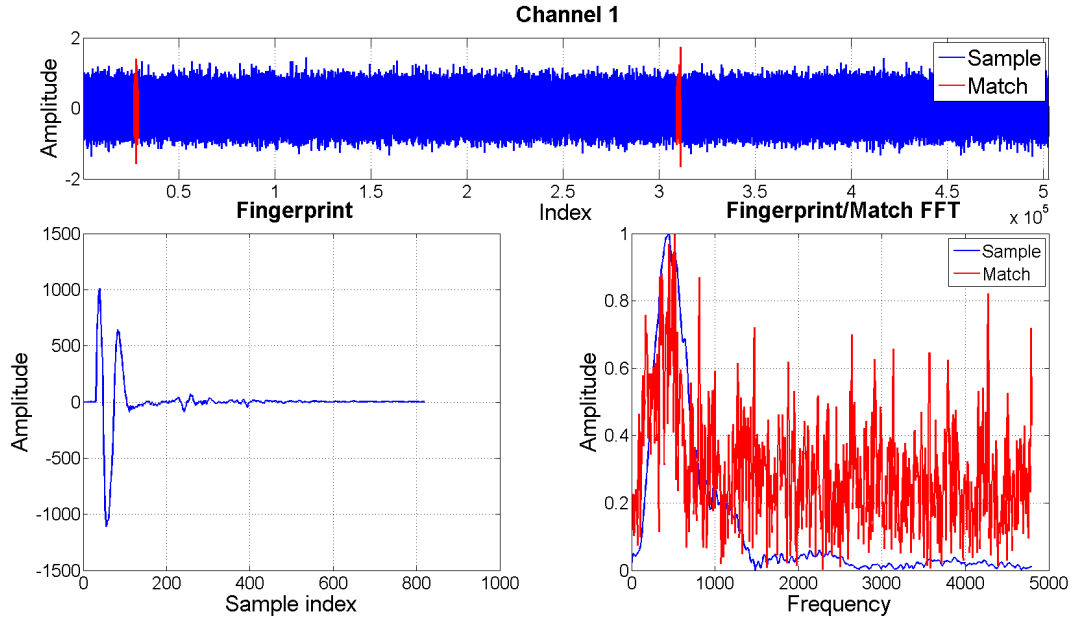


Figure 3.12: Ruger .22 Pistol Fingerprint and Marlin 336 .30-30 Rifle Sample With Additive White Gaussian Noise

3.3 Hardware Implementation

To ensure that the resulting hardware cross correlation matched the MATLAB simulation, a simple test was performed on the FPGA. Figure 3.13 shows the normalized cross correlation between two channels of the same recording, while figure 3.14 shows the absolute value of the normalized cross correlation. The MATLAB and FPGA cross correlations had a RMSE of 0.025. There is an inversion in the two correlations in figure 3.13 from indices 1000 to 2000, but the remainder of the signals are very close to each other. Some variance is expected, as the FPGA version has multiple sources of error which accumulate throughout the process. These include truncation or rounding of the lower bits at the outputs of the FFT and IFFT in addition to truncation of the lower bit of the imaginary result of the complex multiplication due to bit width growth. While small, these errors collect and affect the output of the IFFT. Without the accumulated errors, the result of the IFFT

would be a completely real number as the original inputs to the cross correlation process were both real numbers. However, due to these inaccuracies, the result has some small imaginary components. This is unavoidable but easy to work around, as the magnitude of the cross correlation is nearly the same.

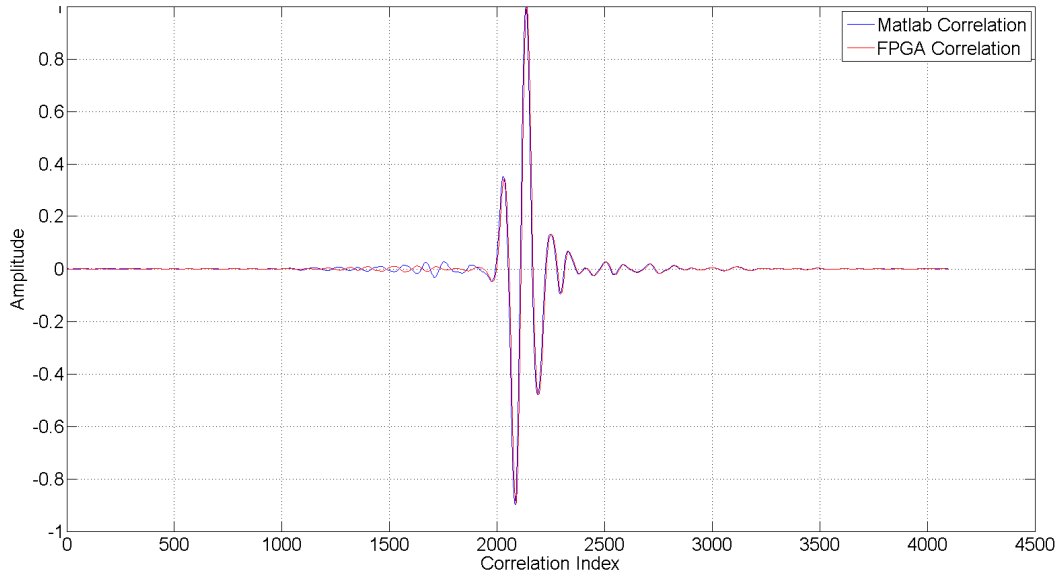


Figure 3.13: MATLAB and FPGA Generated Cross Correlations

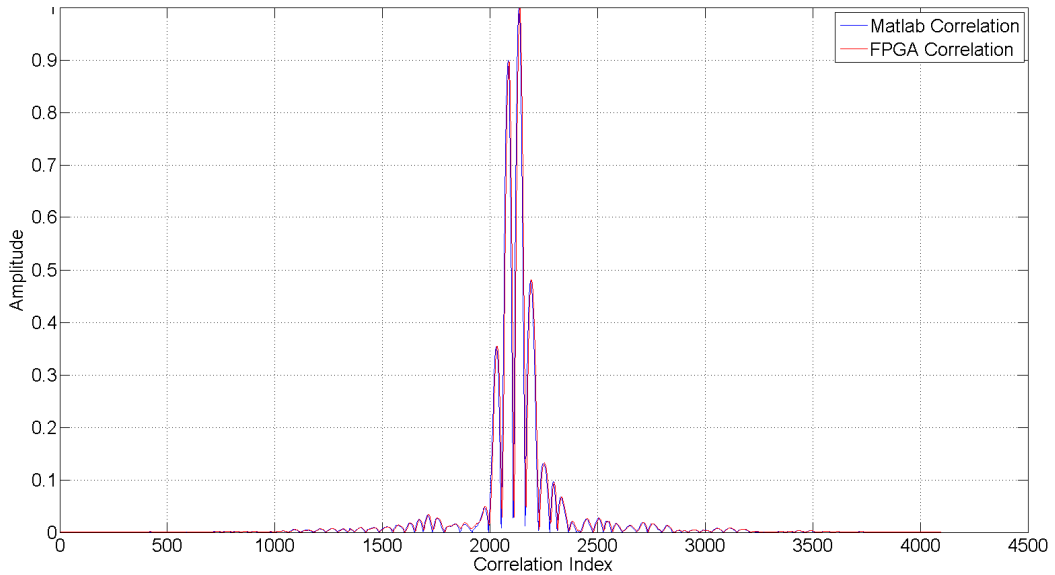


Figure 3.14: Absolute Value of MATLAB and FPGA Generated Cross Correlations

All data transfer operations, flags, run commands, etc. are handled by a finite state machine (FSM) control unit which manages the operation of the entire system. It receives an 8 bit command which contains the following settings: run, reset, ADC enable, run continuously/once, $NFFT$, and the option to send serial results. Run and reset cannot be enabled at the same time, as reset will take precedence. The options to use the ADC and to run the system continuously/once are used for debugging and testing. In normal use, the ADC would always be enabled, the system would run continuously, and the results would not be sent via serial. The value of $NFFT$ also determines the number of samples to record, equal to $NFFT/2$. Each time a run command is received, the FFT and IFFT IP cores are configured with the current value of $NFFT$. Sending serial results can only be done if the system is connected to a PC via UART, and the send serial data flag is set high. Assuming it is, then once a sample has been processed and checked, the number of detections and the indices of the fingerprint(s) that caused the detections are sent to MATLAB, assuming an event was detected and triggered a flag to send the data.

Before processing is performed on the FPGA, audio samples are captured using a supporting board which contains a MEMS microphone, adjustable gain amplifier (LM741), and an ADC (AD9851), along with the necessary voltage regulators and biasing circuits. The ADC allows for a sampling rate of up to 100kHz at 16 bits of resolution. Control is handled by the FPGA; when it is ready to capture a sample, it sends a 'ready' signal to the ADC. When the conversion is complete, the ADC sends a 'finished' signal and presents the converted data on its 16 bit parallel output. From here, the FPGA stores the sample in memory and repeats the process until the desired number of samples have been collected. Additionally, samples can be loaded from a PC to allow for repeatable and consistent testing.

To ensure that events are not missed or split during the transition from one sampling period to the next, two separate memories are used to save the sampled audio. The recorded samples are staggered and alternate between the two memories. The first memory will fill until a set overlap point, after which both the first and

second memories will store data. Once the first memory is full, an alert is triggered, a flag which denotes which memory is currently busy flips, and the second memory continues to fill. Once the overlap point is reached again, the first memory will begin to store data, and the cycle repeats. The overlap size is adjustable; if it is too large, then events will be detected multiple times and the amount of available processing time will be limited. Conversely, if the overlap range is too small, short events which occur during the transitional period may be cut off and not detected at all. Figure 3.15 shows this process, with blue representing one memory block and red representing the other.

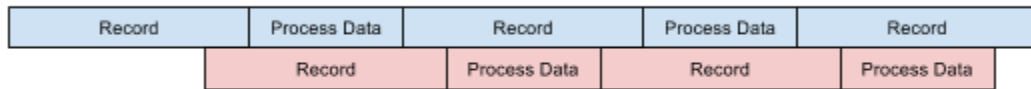


Figure 3.15: Overlap Scheme for Audio Sampling

Once samples have been stored in memory, they are zero padded and sent to a Xilinx FFT IP core via a multiplexer. The multiplexer passes $NFFT/2$ samples to the input of the FFT before its control bit switches, causing the value of the remaining $NFFT/2$ samples fed to the FFT to be zero. The FFT has a configurable transform length and scaling factor. Changes to the transform length can be made depending on the length of the events that the system is looking for. The majority of the energy in a gunshot, for example, occurs in a few ms or less, while a window breaking may last tens or hundreds of ms. The scaling factor is required due to the nature of the FFT algorithm. The width of the data grows by $\log_2(NFFT)$ bits, which can potentially cause its value to overflow and wrap around to a lower value. To prevent this, each stage scales the value of the data down by $\frac{1}{2}$ to $\frac{1}{4}$.

Once the FFT core has completed the transform of the sampled audio, the data is saved to memory so that it can be used for comparisons with multiple fingerprints. The control logic then handshakes with a Xilinx complex multiplier IP core and begins

incrementing the addresses of both the current fingerprint and the stored frequency domain sample until all data has been fed into the multiplier.

As the complex multiplication process finishes, the results are fed into an IFFT of $NFFT$ length. The IFFT sends the resulting cross correlation (eq. 3.6) to a threshold comparison block. If the threshold is surpassed, a flag is triggered and the index of the fingerprint that caused the detection is saved, along with the time domain sample that caused the detection. With this, the cross correlation operation and checking are finished. If more fingerprints remain, the process will repeat, and each fingerprint will be cross correlated with the stored transformed sample and checked. Once the remaining fingerprints have been checked, the process will wait for new data from the ADC.

A block diagram of the system can be seen in figure 3.16. The control logic interfaces with each component, but the connections are not shown for simplicity's sake.

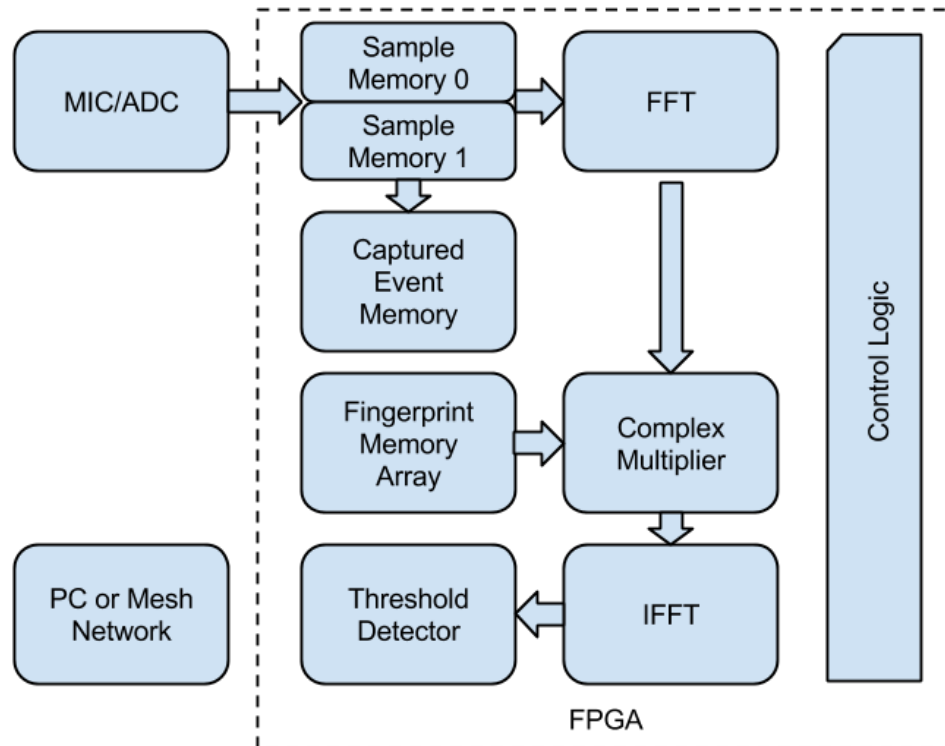


Figure 3.16: Block Diagram of FPGA System

Chapter 4

Performance and Resource Utilization

4.1 Number of Fingerprints

The first limitation on the number of fingerprints that the system can handle is processing time. To give an example, a 8192 point FFT takes 30896 clock cycles to complete, after taking $NFFT$ cycles to read in the data. Storing the results of the FFT for comparisons against the fingerprint library takes an additional $NFFT$ cycles. Each complex multiplication takes 6 cycles, although by pipelining this operation, it only takes $NFFT + 6$ cycles, and the output is fed directly into the IFFT. Comparing the resulting cross correlation to the threshold takes 1 cycle per comparison. An estimate of the resulting clock cycle count is presented in equation 4.1. The total processing time from reading a stored sample to checking the result against the threshold is roughly equal to equation 4.2. Table 4.1 shows the processing time for various values of $NFFT$, while figure 4.1 shows the total processing time required to compare a sample against a range of fingerprints for various values of $NFFT$. Note that the number of samples recorded are equal to $NFFT/2$.

$$ClockCycles = 2NFFT + FFTCycles + NFingerprints(2NFFT + FFTCycles) \quad (4.1)$$

Table 4.1: Required FFT Cycles & Processing Time

NFFT	Cycles	Time(μS)
1024	3458	34.58
2048	7321	73.21
4096	14489	144.89
8192	30896	308.96

$$ProcessingTime(s) = \frac{ClockCycles}{ClockFrequency} \quad (4.2)$$

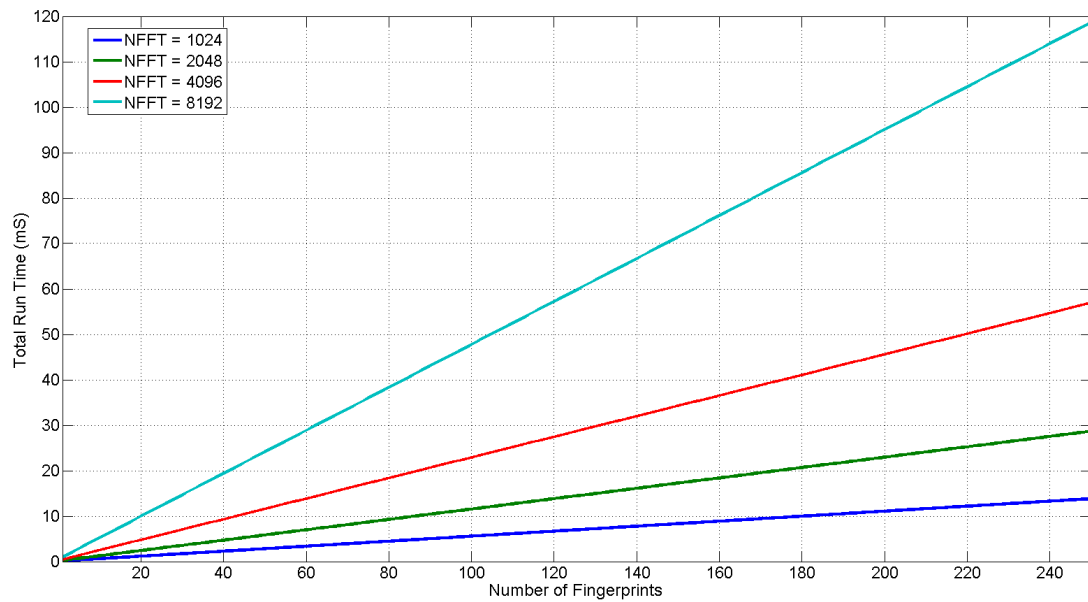


Figure 4.1: Processing Time Required for Sample Comparison

Table 4.2: Sampling Period for Various NFFT Lengths

NFFT	Sample Length (mS) at 48kHz	Sample Length (mS) at 100kHz
1024	11	5
2048	21	10
4096	43	20
8192	85	41

From the results shown in figure 4.1 and table 4.2, it is apparent that processing time is not much of a limiting factor regarding the number of fingerprints which can be examined in a single sampling period. Even at a 100kHz sampling frequency, roughly 85 to 100 fingerprints could be analyzed in a single cycle, depending on the length of $NFFT$.

The amount of memory available on the FPGA is a limiting factor, however. The XC7A100T FPGA has 4,860kb available in 135×36 kb block RAMs (BRAM). A single fingerprint is composed of 16b of real and 16b of imaginary data, for a total of $32 \times NFFT$ bits. Table 4.3 shows the total block ram usage for a number of $NFFT$ sized fingerprints, while table 4.4 shows the amount of BRAM required for a 16b input FFT IP core for varying values of $NFFT$ (the IFFT takes 32b of real and 32b of imaginary data, due to bit growth in the multiplication stage). This assumes a Radix-4 FFT, which is the fastest available non-streaming FFT IP core. All values were taken directly from the resource utilization pane of the IP core configuration screen in Vivado. Vivado is the Xilinx tool used to synthesize hardware description language (HDL) modules, determine proper placement of components on the FPGA, and to generate a bitstream, which is used to program the FPGA.

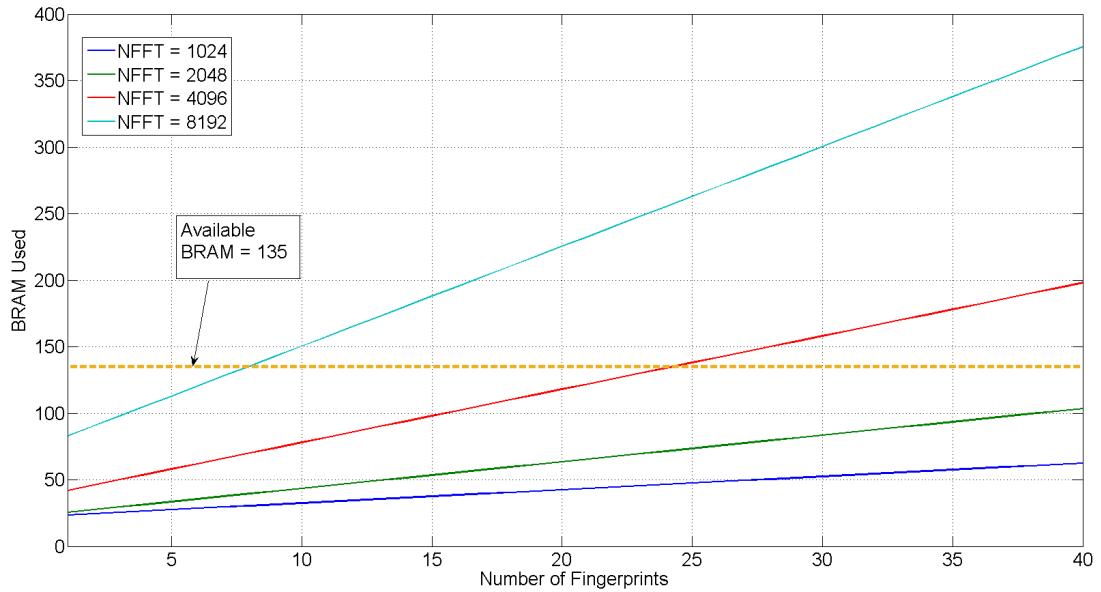
Table 4.3: Fingerprint Memory Usage

NFFT	Total Mem. (kb)	36kb BRAM Req.
1024	32.768	1
2048	65.536	2
4096	131.072	4
8192	262.144	7.5

Table 4.4: FFT Memory Usage

NFFT	36kb BRAM Req.
1024	7
2048	7
4096	11
8192	22

Additionally, $\frac{NFFT}{2} \times 16$ bits of memory are needed to store the acquired sample before transferring it to the FFT (the FFT expects to receive data at 100MHz while the ADC can only supply data at a maximum of 100kHz). An additional $NFFT \times 32$ bits of memory are needed to store the transformed sample. The total amount of required memory for the audio matching system can be seen in figure 4.2, where the yellow line represents the total available memory.

**Figure 4.2:** BRAM Required for System

The actual number of fingerprints which can be stored on the FPGA can be increased by exploiting the symmetry of a frequency domain signal. The first point of a transformed signal is equal to its DC offset, while the remaining samples from 1 : $(NFFT/2 - 1)$ are mirrored copies of $(NFFT/2) : (NFFT - 1)$. By only saving half

Table 4.5: Max Fingerprints for Given NFFT Size (Neglecting Other Memory Usage)

NFFT	Max Fingerprints
1024	112
2048	55
4096	24
8192	7

of a transformed signal as a fingerprint, incrementing its address until $NFFT/2 - 1$ is reached, then decrementing the address, the total number of fingerprints can be doubled. This would lead to processing time becoming more of an issue for smaller sized FFTs, but for anything larger than 2048, memory utilization would be remain the primary bottleneck

4.2 FPGA Resource Utilization

Figure 4.3 shows the resource utilization of the detection system in the most recent revision. This includes twenty 4096 point fingerprints. Flip flop (FF), look up table (LUT), and clock buffer (BUFG) usages are all very low. The logic and state machines which control the system are responsible for a moderate amount of the FF and LUT utilization, while the FFT IP cores are responsible for the remaining amount. Each FFT IP core relies on a table of sine values to reduce the computational complexity and time of the FFT operation. The 16 bit parallel ADC, its control signaling, and the LEDs and switches which are used to control the system make up the Input/Output (I/O) usage.

Nearly one third of the available digital signal processing units (DSP48) are used to transform the input sample, perform the multiplication during the cross correlation phase, and then transform the resulting frequency domain cross correlation signal back into the time domain. Figure 4.4 shows the internal structure of the DSP slice; it is composed of registers for each input, a multiplier, an accumulator, a pre-adder, an arithmetic unit, an optional logic unit, pattern detector, and other features such

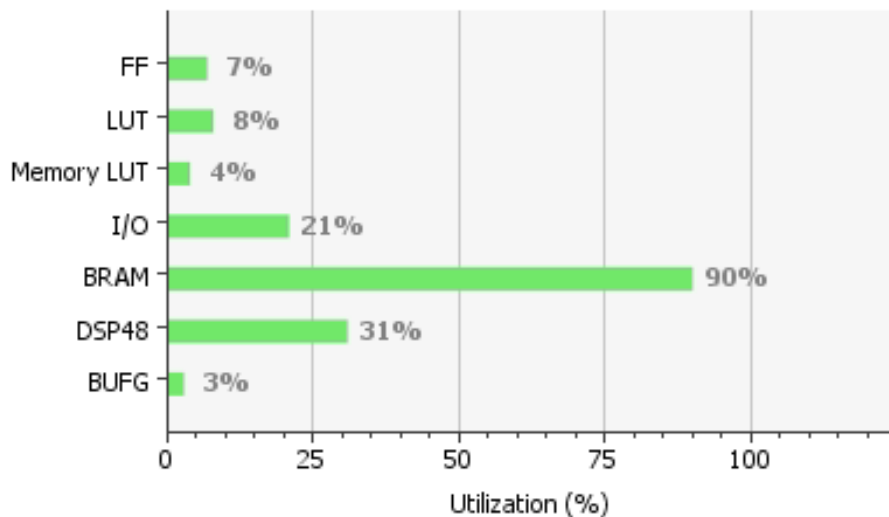


Figure 4.3: Resource Utilization of XC7A100T FPGA

as buses for pipelining. The FFT and IFFT cores mainly rely on the multipliers, accumulators, and pipelining capabilities. The forward FFT used to transform the incoming sample uses 17 DSP slices, while the inverse FFT used to obtain the cross correlation uses 52 DSP slices, regardless of the size of $NFFT$. The reason for the lack of change in the number of multipliers used for different values of $NFFT$ is that the FFT butterfly structure remains the same while the amount of data which must be processed increases; therefore, resource utilization stays the same while processing time increases. The large increase in the number of slices used by each FFT is due to the inverse FFT taking in two 32 bit numbers as the real and imaginary parts, versus the forward FFT only taking 16 bit inputs. Finally, the complex multiplier requires 4 DSP slices.

As discussed in the previous section (4.1), the number of fingerprints are responsible for the high memory utilization. Twenty 4096 point fingerprints require $73 \times 36kb$ BRAM, of which the system has 135 available. The FFT and IFFT use an additional 22 and 11 BRAM, respectively, while the sample memories consume an additional 16 altogether, bringing the total memory utilization to 90%. Some methods of improving on the system utilization are discussed in chapter 6.

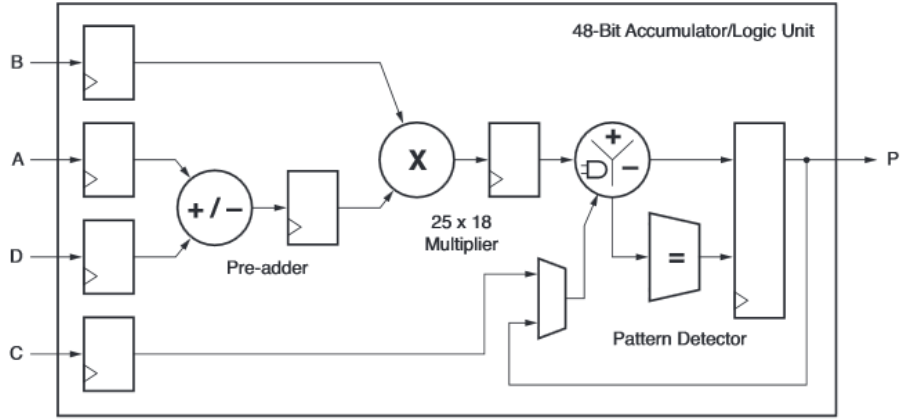


Figure 4.4: DSP48 Slice [17]

4.3 Performance

The system is able to process twenty-plus fingerprints during a sampling period. Tests were run to get an accurate count of the number of clock cycles that the system takes to perform cross correlations. In these tests, 4096 point fingerprints and samples were used, while the number of fingerprints was varied to measure the per-fingerprint processing time for a small range of values. The sampling frequency was 48kHz, and a 100MHz clock with a period of 10ns was used. Clock counts were tallied from the time that a run command was issued to the time that the ADC had gathered 2048 samples, and then from that point to the completion of the threshold check of the cross correlated signals. The clock counting periods will be referred to as the ADC period and cross correlation period.

Running the above test on a single fingerprint yielded 4,270,082 counts during the ADC period, and 29,033 in the cross correlation period. A sampling period of 42.7ms is consistent with the expected value of $\frac{2048 \text{ samples}}{48000 \text{ Hz}} = 42.6 \text{ ms}$. The cross correlation process took 290.33 μs to complete. This is quicker than the estimated value of 453 μs from figure 4.1. A ten fingerprint test yielded a total cross correlation processing time of 1.599ms, or 159 μs per sample, while a twenty fingerprint test yielded a total cross correlation processing time of 3.053ms, or 152 μs per sample.

These values are well within the range of time available, and show that the system is more than capable of processing a large number of fingerprints each sampling period. The difference between the estimated cycle counts from equation 4.1 and the measured values is due to pipelining in the FFT IP cores. As they do not require all $NFFT$ values to be read in before they can begin processing data, a large amount of the data transfer delay can be hidden. A more accurate estimate of the delay would be $NFFT + FFTcycles + NFingerprints(NFFT + FFTCycles)$.

4.4 Power

The estimated on chip power given by Vivado for the implemented design is 259mW. This estimate is likely lower than the actual value, as it does not account for the I/O communications with the ADC. The supporting board containing the ADC and the amplifiers will also use additional power. The datasheet for the ADC lists its max power dissipation at 100mW. The LM741 op-amp driving the ADC has a max output of 10V, and it sees a 9.5k Ω load, which means it is sourcing roughly 1mA of current, for up to 100mW of additional power. The op-amp's input bias currents are typically 80nA. Using +/-10V rails, this means that the static power consumption of each op-amp is 160nW. This leads to a total estimated power of roughly 460mW. Using a power meter at the supply would lead to a more accurate number; however, one was not readily available.

Chapter 5

Results

To verify the detection capabilities of the system, a small sampling of fingerprints were compared against three different handguns. There were some issues interfacing the FPGA with a laptop which was set up to send commands and receive detection data, so modifications were made. Mainly that run, reset, and the threshold were set by switches on the Nexys 4 board, and that the number of detections was displayed on the LEDs instead of being sent to MATLAB. This meant that it would not be possible to determine the indices of matching fingerprints, so only one fingerprint at a time was tested.

Three handguns were used for the test: a Ruger LC9 9mm, a Ruger LCP .380, and a .22 revolver. The system was placed on a box to elevate it a couple feet from the ground and slightly angled down. It was located directly next to a house on top of a hill, which itself is located on a gradual downhill slope. Shots were fired over a wooded valley, which caused somewhat lengthy echoes each time. A fingerprint would be loaded and each gun would be fired multiple times at distances of twenty and fifty feet from the sensor. Adjustments were made to the threshold during the test as necessary. The fingerprints used were 4096 points long, the sampling frequency of the ADC was 48kHz, and the overlap between samples was set to one quarter, or 512 samples.

The first fingerprint tested was that of a 9mm Walter PPQ pistol. Once the threshold was successfully dialed in, it detected 76.9% of all shots fired by the 9mm LC9 (table 5.1) at 20 feet, and 60% of shots at 50 feet. The "*" next to the numbers in the "% Detected" field means that the system detected more shots than were actually fired, due to a low threshold. This could be caused by multiple factors. The overlap between the alternating sample memory could have captured the gunshot at the end of one period and the beginning of another, meaning the event would appear twice. It could also be due to echoes off of the house or from the valley which was being fired over causing events to appear multiple times. Regardless, raising the threshold improved the results, although some more intermediate values should be tested in the future. The system performed better when the shots were being fired closer to it. Some improvements and further tests are discussed in chapter 6.

Table 5.1: 9mm Fingerprint vs LC9

% Detected	Distance	Threshold
76.9	20	23000
233.3*	20	22000
240*	20	20000
60	50	23000
40	50	20000

Table 5.2 contains the results of testing the LCP against the 9mm fingerprint. The system was able to detect its shots at a rate similar to that of the LC9 when nearby, but the detection accuracy decreased quickly at distance. It should be noted that only 5 shots were fired for the 50 foot test, as the remaining ammo needed to be saved for the comparison of the LCP against the .380 fingerprint. With a lower threshold, it likely would have performed better. Also note that the 80% detection rate listed contained one set of shots where 6 were detected when only 5 were fired, which means the percent detected is listed as higher than it actually was.

The .22 revolver test is shown in table 5.3. The system had a large deal of trouble detecting the .22 revolver in all subsequent tests. The ammo used was labeled as "quiet" and contained a smaller amount of powder than a .22 would typically use.

This, combined with the fact that it was being fired from a revolver while the other two guns used were semi automatic pistols, likely led to this low detection rate.

Table 5.2: 9mm Fingerprint vs LCP

% Detected	Distance	Threshold
80*	20	23000
120*	20	22000
20	50	23000

Table 5.3: 9mm Fingerprint vs .22 Revolver

% Detected	Distance	Threshold
0	20	23000
20	20	22000
120*	20	20000

Tables 5.4, 5.5, and 5.6 show the results of the .380 fingerprint test. The system appears to detect the LC9 better than the LCP; however, note that in both the 20 and 50 foot tests in table 5.4, an extra event was detected. Still, it supports the idea that a fingerprint of one caliber can be used to detect a shot from a gun of a different caliber. The system had some issues detecting the LCP- it would have had a 70% detection rate, but on the last test, it only detected 1 out of 5 shots, which skewed the results downward. Again, detection of the .22 was not satisfactory.

Table 5.4: .380 Fingerprint vs LC9

% Detected	Distance	Threshold
93.3*	20	23000
86.7*	50	23000

Table 5.5: .380 Fingerprint vs LCP

% Detected	Distance	Threshold
53.3	20	23000
60	20	22000

Table 5.6: .380 Fingerprint vs .22 Revolver

% Detected	Distance	Threshold
0	20	23000
0	20	22000
50	20	20000
60	50	20000

By the time that the .22 fingerprint was being tested, the .380 ammunition had been depleted, leaving only the LC9 and the .22 revolver to test. Again, the system performed well and detected 80% of the LC9's shots. However, it still had issues with the .22 revolver.

Table 5.7: .22 Fingerprint vs .LC9

% Detected	Distance	Threshold
80	20	23000

Table 5.8: .22 Fingerprint vs .22 Revolver

% Detected	Distance	Threshold
0	20	23000
40	20	20000
20	50	20000

The above results show that the fast cross correlation algorithm used for audio event detection is a viable method. While the system still needs tweaking, it does perform well in the right situations, achieving accuracies of up to 80% (discounting those which contained false positives). Based off of the above results and the amount of overlap in detections between different types of guns, it would likely be possible to store only a handful of gunshot fingerprints on the system and use those to detect a large variety of gun types. Improvements and ideas for further tests are presented in the next chapter. A complete table of results is presented in the appendix.

Chapter 6

Future Work

6.1 Further Tests

6.1.1 Fingerprint SNR Considerations

The scream and gunshot detection system described in [5] mentioned a change in detection accuracy depending on the difference between the SNR of training samples and the SNR of the test samples. Figure 6.1 shows these negative effects as both false negatives and false positives. It can be seen that the best results are obtained when the test and training sequences have the same SNR, while the worst results occur when the disparity in SNR between the test and training sequences are at their largest. This will be something to keep in mind when creating fingerprints and running the system in the future. For example, when the sensors are placed in areas with lots of potential interference, possibly due to echoes off of buildings or background noise, a lower SNR may be preferable due to changes in the received waveform. Conversely, in large open areas with little background noise, a higher SNR may be preferable as the received samples will more closely match those of the original fingerprints. More testing will need to be done to determine the optimal fingerprint SNR for different circumstances.

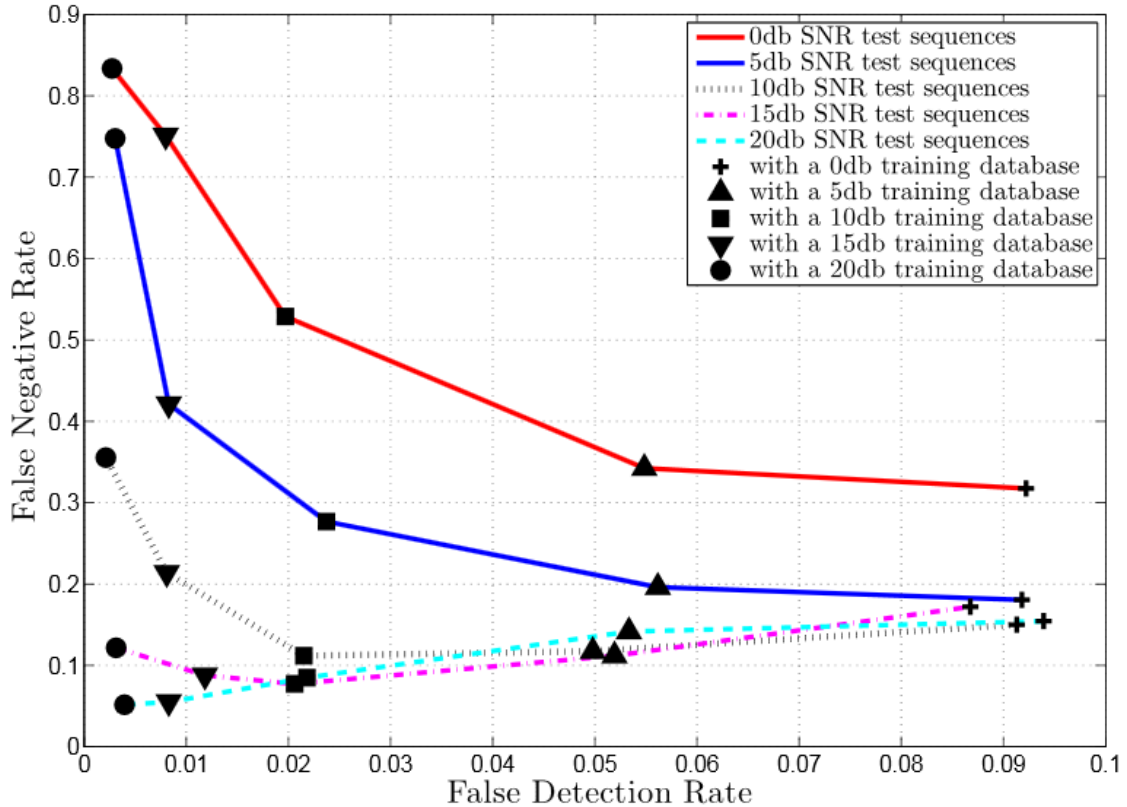


Figure 6.1: Training and Test SNR [5]

6.1.2 Threshold, Sampling Frequency, and NFFT

More testing needs to be performed to find an optimal threshold to trigger an event detection. Too low, and non-events will trigger the system; too high, and the system won't capture any events. A sliding threshold that changes based of the background noise level over a certain time period may be beneficial. Narrowing the threshold down to a certain range is a simple matter of trial and error.

The gain of the microphone needs to be tested over a range of values. At short distances, a high gain would lead to the signal clipping and introducing high frequency components. However, too low of a gain would mean that shots would be missed at greater distances. Each unit would likely have a different practical level of gain depending on its deployment location.

The sampling frequency and $NFFT$ size is also important. High and low (100kHz vs 48kHz) sampling frequencies should be tested across a range of $NFFT$ sizes. The adjustment of these variables will have two effects; the first will be that the sampling period changes. For certain events such as gunshots, a shorter sampling period would probably be preferable. For others, such as breaking glass or snapping branches, a longer period would be better as it would allow for more of the event to be captured. The second effect would be a higher or lower resolution of the captured event. Again, this would need to be tested to see if a higher sampling frequency, and therefore a higher resolution, is preferable. Likewise, the amount of overlap between the recorded sample memory needs to be tested to find a compromise between the ability of the system to capture events without splitting them between samples, and having such a long overlap that events are detected multiple times.

6.2 Optimizations

6.2.1 System Performance and Fingerprints

The results in section 4.3 showed that the processing time of a single fingerprint is lower than expected, at roughly $150\mu s$. Assuming $NFFT = 4096$, a sampling frequency of 48kHz, and an overlap between samples of $\frac{1}{4}$, the system would have 32.25ms to process the current data before it needed to begin processing the next sample. This allows for a maximum of $\frac{32.25ms}{150\mu s} = 215$ fingerprints to be processed. To get around the issue of memory limitations on the FPGA chip itself, external memory could be used. The Nexys 4 board that the system was tested on has an additional 16MB of RAM. Reading from this would increase the complexity and delay as the interface requires some additional clock cycles to access the memory's contents. However, the number of fingerprints available to compare would greatly increase.

There is also some room for pipelining and improving the cross correlation process. Before the last point of a sample is saved to memory, the preceding points could be

sent to the forward FFT so that the last sample arrives just in time to be sent out and complete the process. As data is being read out of the forward FFT into memory for re-use with later fingerprints, it could be sent to the complex multiplier at the same time. While the IFFT is dumping its data out to the threshold comparison, the next frequency domain cross correlation could be read in to the IFFT. Additionally, if off chip memory is used to store fingerprints, the on chip memory could be used to increase the number of channels of the IFFT, which would allow for multiple fingerprints to be compared in parallel. Integrating all of these improvements would push the number of fingerprints that can be compared in a single sampling period into the high hundreds, or even low thousands. This number could further be increased by raising the clock speed.

6.2.2 Power Consumption

As the system will be part of a sensing platform, it may be placed in areas without readily available power. In these locations, power consumption will be an important consideration as it could greatly affect the run time of the sensing platform. Assuming that the FPGA can process the desired number of fingerprints before the next sampling period begins, the system clock could be gated, leaving only the ADC and sample input memory running during this down time. Removing the linear voltage regulators from the supporting board and replacing them with fixed supplies would be beneficial. Other optimization methods will need to be explored to improve on this aspect.

Chapter 7

Deliverables

The final product of this project is an FPGA based audio event detection system. It supports the option of a simple integration with a mesh network of environmental sensors. The onboard library can hold fingerprints of more than 20 unique sounds, including gunshots, breaking glass, and other impulsive noises. On detection of an event, the system saves the sample for later analysis and drives an alert signal high. If interfaced with a mesh network, this alert will propagate through the network and tell all other units to save the last period of audio for further analysis. A supporting board containing a microphone and ADC is also included, along with the board schematics, a parts list, and a MATLAB program to create and save audio fingerprints.

Bibliography

- [1] A. Wang *et al.*, “An Industrial Strength Audio Search Algorithm.,” in *ISMIR*, pp. 7–13, 2003. [3](#)
- [2] H. Wang and J.-C. Liu, “An FPGA based parallel architecture for music melody matching,” in *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*, pp. 235–244, ACM, 2013. [4](#)
- [3] A. De Cheveigné and H. Kawahara, “YIN, a fundamental frequency estimator for speech and music,” *The Journal of the Acoustical Society of America*, vol. 111, no. 4, pp. 1917–1930, 2002. [4](#)
- [4] Chien-Min Ou and Che-Yu Yeh and Yong-Long Su and Wen-Jyi Hwang and Jing-Fung Chen, “FPGA Implementation of Content-Based Music Retrieval Systems,” in *Embedded Software and Systems Symposia, 2008. ICESSE Symposia '08. International Conference on*, pp. 96–103, July 2008. [4](#)
- [5] L. Gerosa, G. Valenzise, M. Tagliasacchi, F. Antonacci, and A. Sarti, “Scream and gunshot detection in noisy environments,” in *15th European Signal Processing Conference (EUSIPCO-07), Sep. 3-7, Poznan, Poland, 2007*. [5](#), [36](#), [37](#)
- [6] G. Simon, M. Maróti, Á. Lédeczi, G. Balogh, B. Kusy, A. Nádas, G. Pap, J. Sallai, and K. Frampton, “Sensor network-based countersniper system,” in *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pp. 1–12, ACM, 2004. [6](#), [7](#)

- [7] P. Volgyesi, M. Maroti, G. Simon, G. Balogh, A. Nádas, B. Kusy, S. Dora, G. Pap, *et al.*, “Multiple simultaneous acoustic source localization in urban terrain,” in *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pp. 491–496, IEEE, 2005. [6](#)
- [8] P. Volgyesi, G. Balogh, A. Nadas, C. B. Nash, and A. Ledeczi, “Shooter localization and weapon classification with soldier-wearable networked sensors,” in *Proceedings of the 5th international conference on Mobile systems, applications and services*, pp. 113–126, ACM, 2007. [6](#)
- [9] J. Sallai, A. Lédeczi, and P. Völgyesi, “Acoustic shooter localization with a minimal number of single-channel wireless sensor nodes,” in *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, pp. 96–107, ACM, 2011. [6](#), [7](#)
- [10] ShotSpotter, “Technology,” 2015. [7](#)
- [11] BBN, “Boomerang warrior-x,” 2015. [8](#)
- [12] BBN, “Boomerang iii,” 2015. [8](#)
- [13] BBN, “Boomerang air,” 2015. [8](#)
- [14] BBN, “Boomerang,” 2015. [8](#)
- [15] Still North Media, “The Free Firearm Sound Effects Library,” 2014. [11](#)
- [16] tagigobaso, “Flowing traffic near a traffic light in the inner ring of milan ,” 2010. [14](#)
- [17] Xilinx, “7 Series DSP48E1 Slice User Guide ,” 2014. [30](#)

Appendix

A.1

X_39 (Walter PPQ 9mm)	Vs Lc9 (9mm)	Distance	Threshold	Vs LPC (.380)	Distance	Threshold	Vs Revolver (.22)	Distance	Threshold
	3/5	20	23000	5/5	20	23000	0/5	20	23000
	4/5	20	23000	3/5	20	23000	6/5	20	20000
	3/3	20	23000	6/5	20	22000	1/5	20	22000
	12/5	20	20000	1/5	50	23000			
	7/3	20	22000						
	3/5	50	23000						
	3/5	50	23000						
	2/5	50	22000						
F_46(Bersa .380 Pistol)	4/5	20	23000	4/5	20	23000	0/5	20	23000
	6/5	20	23000	3/5	20	23000	0/5	20	22000
	4/5	20	23000	1/5	20	23000	0/5	20	22000
	3/5	50	23000	3/5	20	22000	1/5	20	20000
	4/5	50	23000				4/5	20	20000
	6/5	50	23000				3/5	50	20000
Q_30(Ruger 10/22 .22 rifle)	3/5	20	23000				0/5	20	23000
	5/5	20	23000				3/5	20	20000
							1/5	20	20000
							2/5	20	20000
							2/5	20	20000
							2/10	50	20000

Figure A 1: Testing Results

A.2

	Shazam	FPGA MIDI Search	Scream / Gunshot Detector	Vanderbilt Countersniper	ShotSpotter	Boomerang	This system
Mobile	Yes	No	No	Yes	No	Yes	Possible
Network connection required	Yes	No	Yes	Yes	Yes	No	No (but possible)
Query time	5-500ms for 20k songs	54ms per query	Not given	Not given	Not given	<1s (including time to determine shooter location)	155us per sample
Accuracy	No numbers given, but very high	90%	90%	Up to 100%	Not given	>95%	Varies
Types of detectable events	Songs	MIDI files	Screams and gunshots	Gunshots	“Sharp acoustic events”- gunshots, explosions	Supersonic gunshots	Impulsive sounds (Gunshots, breaking glass, etc.)

Figure A 2: System Comparisons

A.3

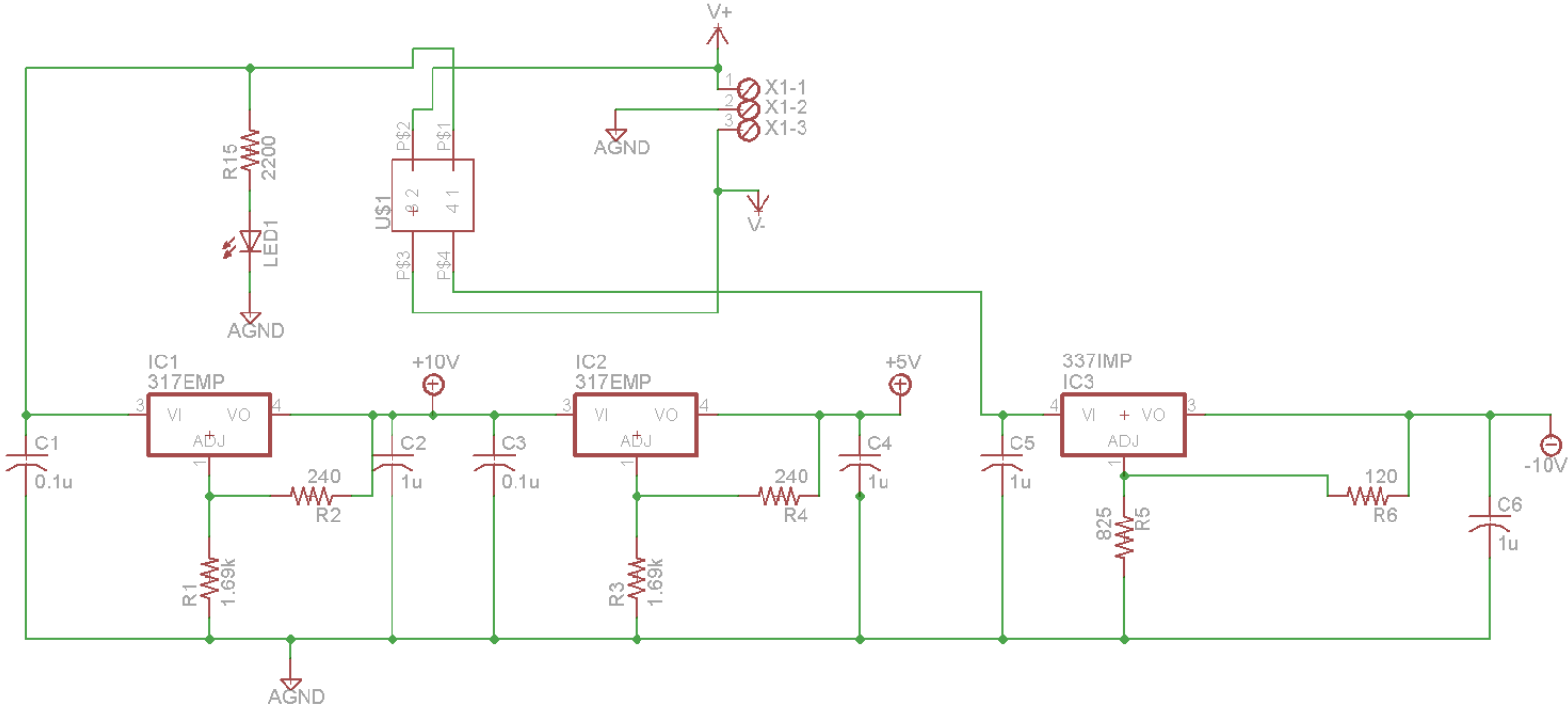


Figure A 3: ADC/Mic board power supply schematic

A.4

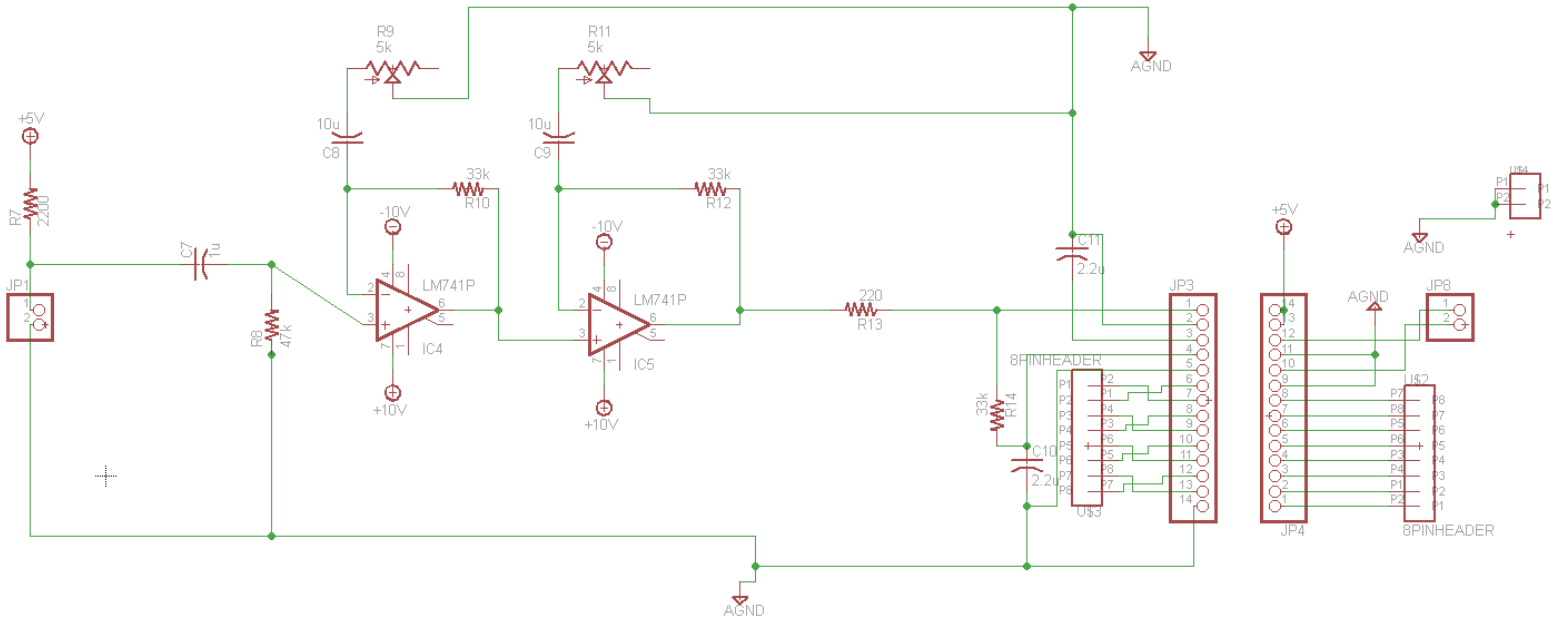


Figure A 4: ADC/Mic board amplifier schematic

A.5

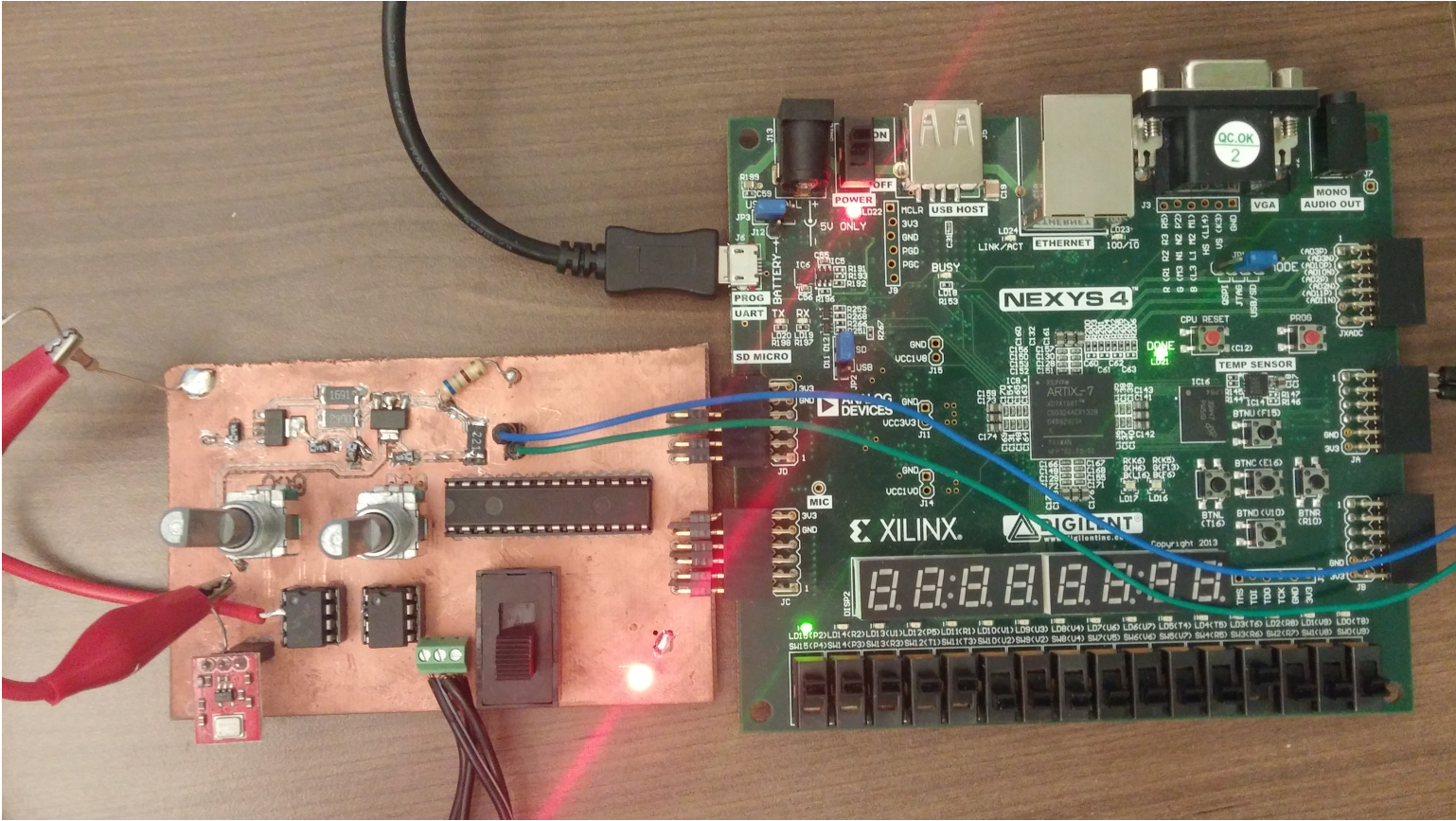


Figure A 5: ADC/Mic board connected to FPGA. Note that board was modified to accept a MEMS mic instead of an electret mic

Vita

Jacob Tobin was born in Oak Ridge, TN to the parents of Ken and Cathy Tobin. He is the first of three children: Emma and Zoe. He attended Roane County High School in Kingston, TN. After graduating in 2010, he moved to Knoxville and, in 2014, graduated from the University of Tennessee with a B.S. in Electrical Engineering. Following the B.S., he was accepted into the M.S. program. He will graduate with a M.S. in Electrical Engineering in December 2015, and he plans to hike the Appalachian Trail the following June.