HARDWARE IMPLEMENTATION OF A VOICE STRESS DETECTION ALGORITHM USING EMPIRICAL MODE DECOMPOSITION

By

Joshua Schwartz

A Thesis
Submitted to the
Faculty of the Graduate School
of
Western Carolina University
in Partial Fulfillment of
the Requirements for the Degree
of
Master of Science in Technology

Committee:

_____ Director

_____

_____

_____ Dean of the Graduate School

Date: _____

Spring 2011
Western Carolina University
Cullowhee, North Carolina

HARDWARE IMPLEMENTATION OF A VOICE STRESS DETECTION ALGORITHM USING EMPIRICAL MODE DECOMPOSITION

A thesis presented to the faculty of the Graduate School of Western Carolina University in partial fulfillment of the requirements for the degree of Master of Science in Technology.

By

Joshua Schwartz

Director: Dr. Robert Adams
Associate Professor
Department of Engineering and Technology

April 2011

This thesis is dedicated to my parents who have always supported me through the best and worst of times. I have come a long way and it would not have been possible without them.

ACKNOWLEDGEMENTS

I would like to thank my committee, Dr. Adams, Dr. Zhang, and Dr. Howell for their guidance and assistance on this project. I would also like to thank the faculty and staff of the Kimmel school for supporting my research. Finally, I would like to thank my fellow graduate students for their support and making this a fun and entertaining experience.

TABLE OF CONTENTS

Acknowledgements iv
List of Figures vi
Abstract viii
CHAPTER 1. Introduction 1
1.1 Background and Need for Study 1
1.2 Significance of the Study 2
1.3 Structure of the Thesis 3
CHAPTER 2. Literature Review 4
2.1 Background Theory 4
2.1.1 Voice Stress Analysis using Empirical Mode Decomposition . . . 4
2.2 Hardware Specifications 8
2.2.1 TMS320C6416 DSP Board 9
2.2.2 Atmel AVR32 EVK1104 9
CHAPTER 3. Design Procedure 12
3.1 EMD Implementation in C 12
3.1.1 50 Hz Low-Pass Filter in C 12
3.1.2 Linear distribution in C 14
3.1.3 Dynamic Clamped Cubic Spline Interpolation 15
3.1.4 Standard MATH.h library functions 18
3.2 Hardware Implementation 18
3.2.1 TMS320C6416 DSP Design 19
3.2.2 Atmel AVR32 EVK1104 Design 21
CHAPTER 4. Results 24
4.1 Relavant Analysis of Previous Results 24
4.2 New Metric for Determining Voice Stress 25
4.3 TMS320C6416 DSK Results 26
4.4 Final Hardware Implementaton 32
CHAPTER 5. Conclusion and Future Work 35
Bibliography 37

LIST OF FIGURES

2.1 EMD process extracted from [2] 5
2.2 Mean of upper and lower envelopes extracted from [3] 5
2.3 EMD Example extrated from [2] 7
2.4 Detection of Deception results extracted from [1] 8
2.5 TMS320C6416 DSP Board from Spectrum Digital's website 10
2.6 Atmel AVR32 EVK1104 10

3.1 50 Hz LPF Impulse Response 13
3.2 Filter results of both Matlab and Embedded C 14
3.3 Filter Error between Matlab and Embedded C 15
3.4 Matlab's spline function versus C spline on DS1\Answer1 16
3.5 Cubic Spline Error between Matlab and Embedded C 17
3.6 Spectrum Digital TMS320C6416 DSP design 20
3.7 Atmel EVK1104 Design 22

4.1 DS1\ Answer1.wav Tremor Results 26
4.2 DS1\Answer1.wav Tremor IMF Matlab and Embedded C Error 27
4.3 DS1\ Answer2.wav Tremor Results 27
4.4 DS1\Answer2.wav Tremor IMF Matlab and Embedded C Error 28
4.5 DS1\ Answer3.wav Tremor Results 28
4.6 DS1\Answer3.wav Tremor IMF Matlab and Embedded C Error 29
4.7 DS1\ Answer4.wav Tremor Results 29
4.8 DS1\Answer4.wav Tremor IMF Matlab and Embedded C Error 30
4.9 DS1 Tremor Frequencies Matlab and Embedded C 31
4.10 DS2 Tremor Frequencies Matlab and Embedded C 31
4.11 EVK1104 Calculations Completed 32
4.12 EVK1104 DS1\Answer1 Tremor IMF 33
4.13 EVK1104 DS1\Answer2 Tremor IMF 33
4.14 EVK1104 DS1\Answer3 Tremor IMF 34

4.15 EVK1104 DS1\Answer4 Tremor IMF 34

ABSTRACT

HARDWARE IMPLEMENTATION OF A VOICE STRESS DETECTION ALGORITHM USING EMPIRICAL MODE DECOMPOSITION

Joshua Schwartz, M.S.T.

Western Carolina University (April 2011)

Director: Dr. Robert Adams

In previous research, using EMD for voice stress analysis showed promising results. The original algorithm was developed in Matlab, utilizing many of its functions. The algorithm decodes audio into a raw format sampling at 8k samples/sec. This signal array is then filtered and centered about the x-axis and the local maxima and minima are determined. Matlab's built-in cubic spline interpolation function was then used to create the upper and lower envelopes. The first Intrinsic Mode Function (IMF) is determined by calculating the difference between the original signal and the mean of the envelopes. The algorithm continually extracts IMFs starting with the highest frequency IMF until it extracts the last IMF. The last IMF represents the frequency of the stress induced tremor in the subjects voice. The successfulness of the algorithm was measured using a series of questions designed to invoke a stress response concatenated with irrelevant questions designed not to invoke a stress response. The difference between the tremor frequency related to the stressful and non-stressful questions determines if a person may be being untruthful.

CHAPTER 1: INTRODUCTION

Voice stress is a consequence of nervous behavior due to psychological and physiological stress. Empirical Mode Decomposition (EMD) has been shown [1, 2] to be an effective method for detecting stress in a persons voice. Prior to this research effort, an embedded hardware implementation had yet to be developed that employs the EMD method for the purpose of voice stress detection. The goal of this research effort was to develop such a device and implement it into a handheld package. The objectives of this effort were as follows:

- Understand the current EMD algorithm and its process.
- Define the necessary hardware components to complete the task.
- Develop a plan for development.
- Develop a hardware prototype.
- Test the prototype for repeatability and reproducibility of results.
- Validate the accuracy of the results by comparing with a MATLAB® simulation.

1.1 Background and Need for Study

Previous work by N. Mbitiru [1] and his associates [2], involved developing a MATLAB® algorithm to detect voice stress using the EMD method. Test results verified a

60-80% success rate in detecting voice stress. N. Mbitiru's test results have been verified and reproduced. The EMD voice stress analysis (VSA) was implemented into MATLAB® as a floating point algorithm that demonstrated that a 8-12Hz tremor frequency is indeed a normal stress response and that a frequency outside of this range was considered a stress response. It is because of this simple tremor frequency result, that makes for a much simpler yet accurate way to distinguish a person's stress level. This simplicity makes it so that anyone can read the results without any prior training or education. The current algorithm needed to be embedded into hardware because the current implementation is cumbersome and difficult to use since it requires a computer and MATLAB®. For the algorithm to be useful for multiple applications, it must be hand-held, easy to use, and fast.

1.2 Significance of the Study

Voice stress detection can be used for several applications, such as law enforcement, military, emergency services, and psychological evaluations. Law enforcement and military agencies may wish to use voice stress detection for the purpose of lie detection. The proposed handheld device will permit these agencies to employ secretive and discrete lie detection methods. If the device is not hand-held and/or portable, it makes it very difficult to use VSA for these applications.

There are many voice stress detection methods that have been developed over the years. The first models to be developed were big and bulky machines that did not work well enough to be used in useful and constructive ways. Today, computer VSA programs can be used to provide us with more accurate ways in which we can measure VSA.

Computer VSA methods employ several different algorithms and techniques to detect fundamental frequencies, and changes in pitch and/or tone in a subject's voice. Gener-

ally, all computer VSA results must be analyzed and interoperated by a trained examiner. Eliminating the need for a computer and/or expensive bulky equipment and having results that anyone could understand would be a big step for the future implementation of VSA in applications previously not possible.

The EMD method was chosen for implementation into embedded hardware because of its ability to display a simple result that doesnt have to be interpreted. The method was also chosen because of its ability to adapt to different voices [2]. These potential improvements will be explained and discussed later.

1.3 Structure of the Thesis

The rest of the thesis is organized by beginning with literature review on background research and necessary hardware specifications. Chapter 3 explains the design procedures for creating the C code and implementing it into hardware. Chapter 4 explains the results of the hardware implementation. Finally, the thesis will be concluded and possible future work will be explained.

CHAPTER 2: LITERATURE REVIEW

Two main areas of research were focused on. Research was conducted to gain an understanding of the EMD method and its applications related, and not related to VSA. Once a simple understanding of EMD was achieved, the Matlab EMD algorithm was deciphered. The other main area of research focused primarily on determining what hardware was needed to achieve the objective of developing an easy to use, accurate, fast, and handheld voice stress analyzer.

2.1 Background Theory

The human voice contains a low frequency microtremor that can help determine the stress level of an individual. A typical person's neutral stress level is between 8-12Hz, but this neutral stress level can be offset by many psychological and physiological factors. These factors play a key role in determining a person's current neutral stress level.

2.1.1 Voice Stress Analysis using Empirical Mode Decomposition

The EMD process is a sifting process that extracts intrinsic mode functions (IMFs) until the microtremor frequency is extracted. The general EMD process is expressed in a flow chart as shown in figure 2.1. The EMD process starts by extracting the local maxima and minima along with their respectable time indexes and connecting each maxima and minima together using a clamped cubic spline interpolation creating the upper and lower envelopes. An example is expressed below in figure 2.2.

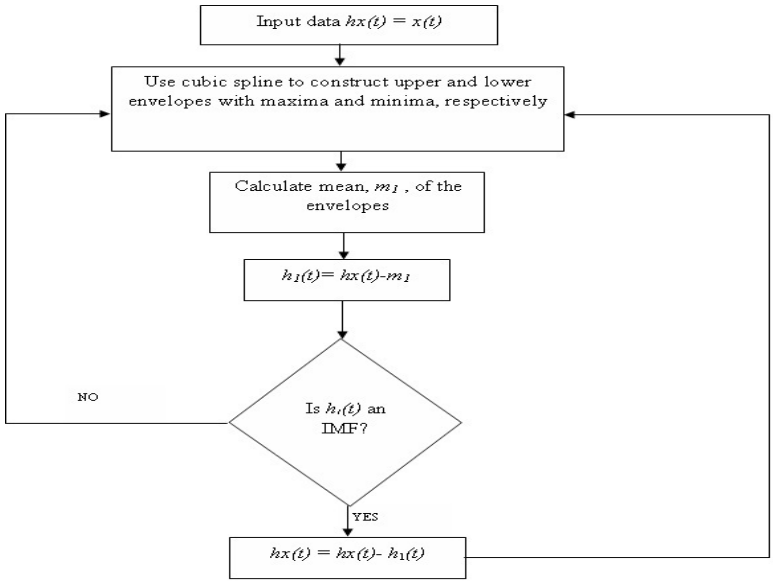

Figure 2.1: EMD process extracted from [2]

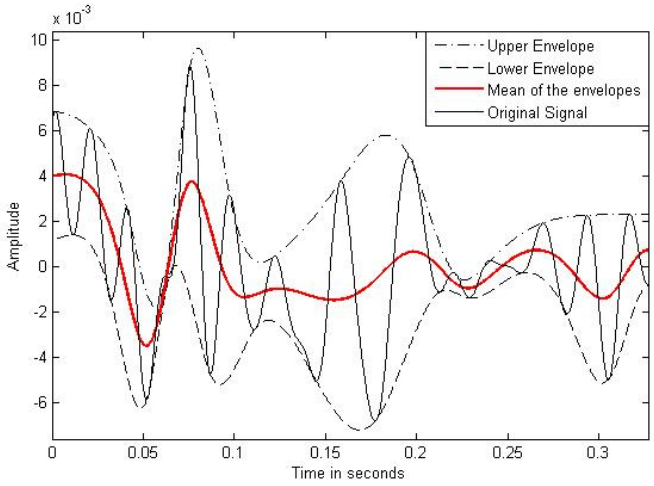

Figure 2.2: Mean of upper and lower envelopes extracted from [3]

The upper and lower envelopes are represented as the dotted lines in figure 2.2. Once the upper and lower envelopes have been constructed, the mean m_n of the envelopes is calculated. m_n is represented by the red line in figure 2.2. The difference between the original signal $hx(t)$ and the mean is given by:

$$h_n(t) = hx(t) - m_n \tag{2.1}$$

$h_n(t)$ is tested under specific criteria to determine if it is an IMF. The signal $h_n(t)$ is an IMF if the following are true:

1. The number of extrema and the number of zero crossings must either be equal or differ at most by one in the whole data set. [2]

2. At any point, the mean value of the envelope defined by local maxima and the envelope defined by the local minima is zero. [2]

If $h_n(t)$ is not an IMF, then the mean (m_n) of the envelopes of $h_n(t)$ is subtracted from $h_n(t)$. The resulting difference is then tested for the properties of an IMF. If $h_n(t)$ is an IMF then the algorithm subtracts the IMF from the original signal ($hx(t)$) to create a new original signal for the process to be repeated. This can expressed by equation (2.2):

$$hx(t)_{n+1} = hx(t) - h_n(t) \tag{2.2}$$

$hx(t)_{n+1}$ represents the new original signal that is further processed to determine the next IMF.

The algorithm sifts out IMF frequencies from highest to lowest until a specific stopping criteria is met and just the residue remains. This results of the EMD process can

be seen below in figure 2.3. Figure 2.3 shows how the EMD algorithm deals with a sum of sine waves example shown by equation (2.3).

$$x(t) = 0.3cos(2\pi 3t) + 0.5cos(2\pi 5t) + 0.7cos(2\pi 8t) + cos(2\pi 12t) \tag{2.3}$$

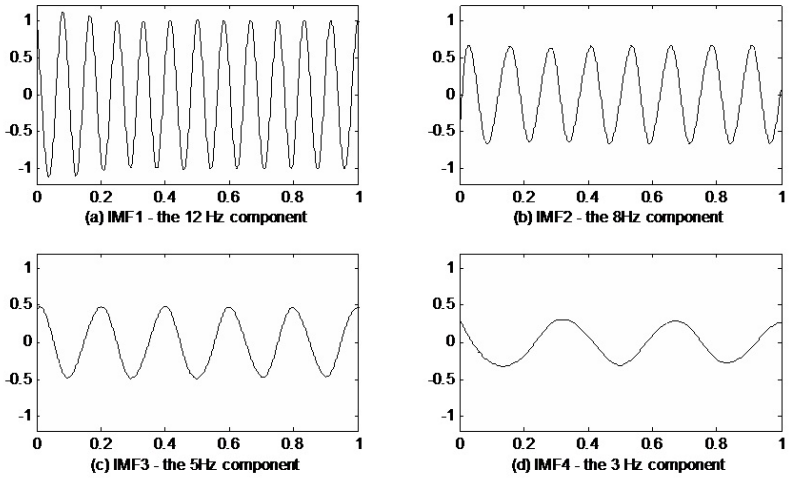

Figure 2.3: EMD Example extrated from [2]

In previous work it was thought that the last IMF to be pulled out of the test subject's voice happened to be the microtremor frequency. Further analysis have shown otherwise which will be explained later. The algorithm was initially tested by using a voice recording database. A control experiment was also conducted on 12 or so students. The students were asked irrelevant questions that were not designed to invoke a stress response. After each irrelevant question a relevant question specifically designed to invoke a stress response was asked. The results of analyzing the voice stress in these voice recordings are displayed in figure 2.4. The frequency of the voice mircrotremor is displayed as a function of a variety of responses to questions. The responses labeled A1, A3, A5, A7, A9, A11, A13, and A15 correspond to the irrelevant questions. The remaining responses correspond

to the relevant questions that were designed to invoke a stress response. It can be seen from figure 2.4 that although the data shows that the low stress microtremor threshold is not always between 8-12 Hz, it does show an increase in frequency on all questions designed to invoke a stress response in comparison with the irrelevant questions. This pattern of high-low microtremor frequency shows that the algorithm works, but it doesnt adhere to expected microtremor behavior. This problem has been solved and is explained in the results section.

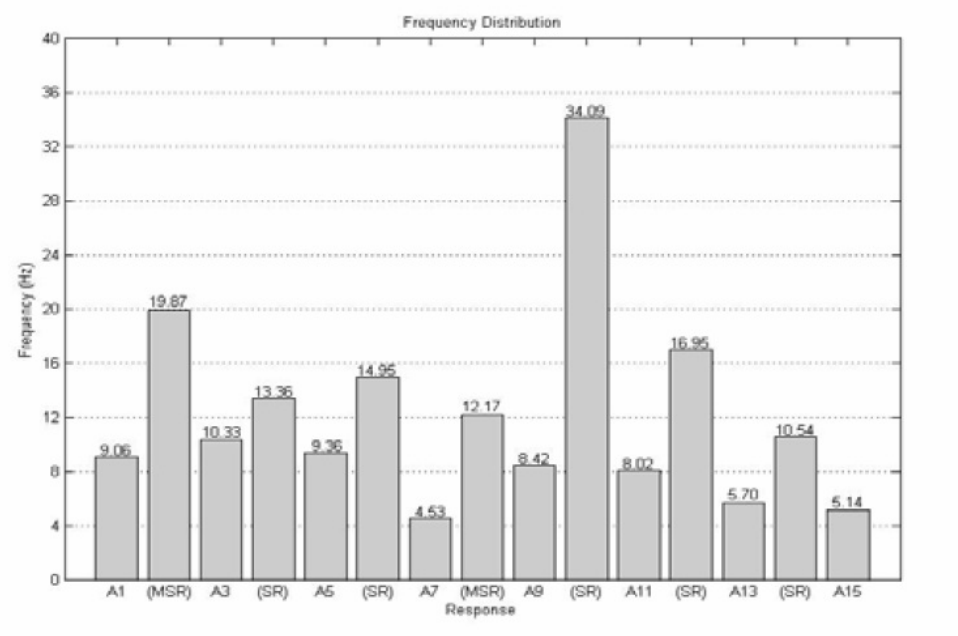

Figure 2.4: Detection of Deception results extracted from [1]

2.2 Hardware Specifications

Several development paths and plans had been discussed. Originally, the plan was to use TI's OMAP3 applications processor. At the time it seemed to meet all technical and customer requirements, but unfortunately using this chip would make it difficult to control the DSP chip and display results freely. It is difficult because the chip is designed

for a higher level of programming. In other words, the chip is designed for embedded operating systems, so programming the chip directly becomes unnecessarily taxing. The OMAP3 is basically an ARM MCU glued to a TI6000 series DSP chip. In order to solve this problem a separate display module and DSP chip is needed. Among others, the same problem would occur if the algorithm were implemented as a smart phone application.

2.2.1 TMS320C6416 DSP Board

The Spectrum Digital TMS320C6416 DSK utilizes a TI 6000 series DSP processor with a 1 GHz clock. This processor was ideal for proper hardware implementation due to its speed, 64-bit floating-point precision and its large Flash and SDRAM resources [4]. The shortcomings of the Spectrum Digital development kit and compiler made it difficult to adhere to the design spec goals. There was no clear cut way to display the embedded results. There was not a way to pull audio files into the memory without additional hardware. Math.h library functions did not work properly with the compiler with very small 64-bit numbers. The compiler was very picky about how you wrote your C code. Despite the DSK's short comings, it proved to be very useful for debugging as well as providing a way to manually read memory locations and copy them over to the PC for error analysis. A picture of the DSP board can be seen below in figure 2.5.

2.2.2 Atmel AVR32 EVK1104

The AVR32 EVK1104 employs an Atmel 32UC3A3256 micro-processor operating at 12 MHz. Despite the slow speed, the EVK1104 development board made it possible to meet the design goals. The board employed a 240x360 LCD, SD memory card ports, and buttons. Figure 2.6 shows the EVK1104. The EVK1104 was used for the final product.

Figure 2.5: TMS320C6416 DSP Board from Spectrum Digital's website

Figure 2.6: Atmel AVR32 EVK1104

The EVK1104 utilizes a 64-bit compiler and is programmed via JTAG. The development tools include: LCD drivers, DSP functions, FAT file system, and standard data communication protocols.

CHAPTER 3: DESIGN PROCEDURE

To successfully develop a hardware implementation of the EMD algorithm, the MATLAB® version of the EMD algorithm was carefully dissected to make sure that the complete EMD algorithm was understood mathematically. Once all previous results were repeated and understood, the long process of converting the EMD algorithm to C had began.

3.1 EMD Implementation in C

The original Matlab EMD algorithm utilized several Matlab functions and internal operations. These functions and internal operations had to be replicated in C. This proved to be the biggest challenge.

3.1.1 50 Hz Low-Pass Filter in C

The embedded algorithm begins by importing a stored audio file into an audio buffer array memory location as a 64-bit double precision floating point waveform. The audio buffer is filtered with a 50 Hz low-pass filter. The filter response is shown in figure 3.1.

This filter is used to remove the higher frequency voice content to speed up the EMD process. The MATLAB® implementation employed the use of the built in convolution function to convolve the audio waveform with the predetermined filter coefficients.

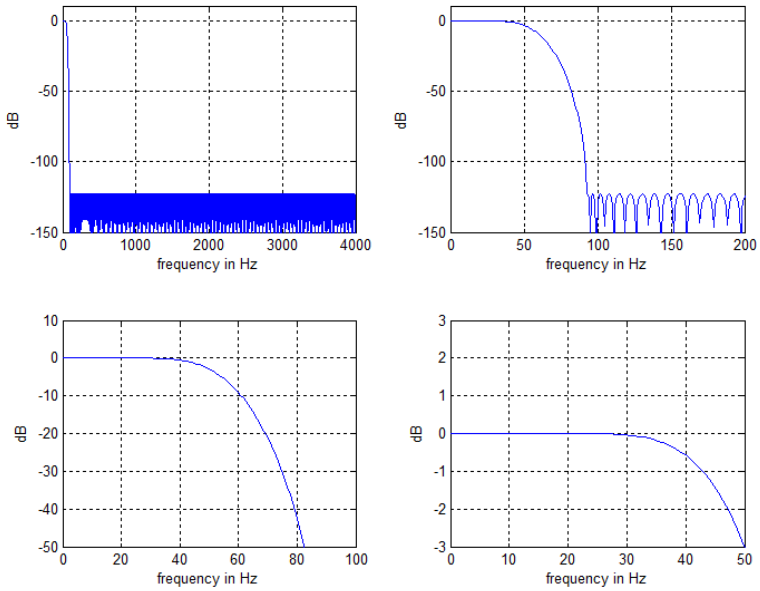

Figure 3.1: 50 Hz LPF Impulse Response

In order to implement the 50 Hz filter into C a convolution function was written in C. This convolution function dynamically adjusts to convolve the filter coefficients with an audio signal that varies in size, which is stored on the audio buffer with a max size of 8000 samples. The number of audio samples stored on the audio buffer is determined by a delimiter put on the end of the audio waveform. The algorithm looks for the delimiter in order to determine the length of the audio. To dynamically adjust the convolution process based upon the length, three key calculation points had to be monitored. The convolution function looked for the point in which the signal was completely contained within the filter coefficients. Next, it looks for the point in which the signal starts to exit the filter. Finally, it looks for the point in which the signal has completely left the filter. Based upon the length of the filter versus the length of the signal, the extra samples are removed and so we are left with the filtered signal. Figure 3.2 shows a comparison between the MATLAB®

filter and the embedded filter applied to a voice audio file.

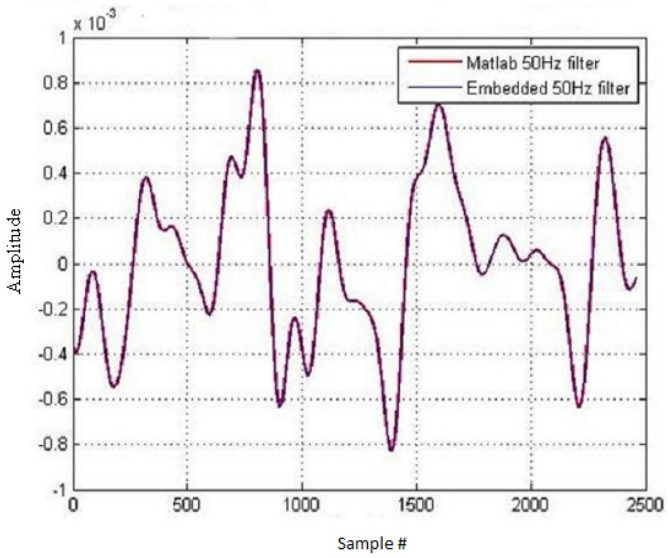

Figure 3.2: Filter results of both Matlab and Embedded C

The results in figure 3.2 show an almost exact match between the MATLAB® filter and the embedded C filter. The error between the results is minimal and only shows a difference starting roughly at the 17th decimal place. The error results can be seen in figure 3.3.

3.1.2 Linear distribution in C

MATLAB® makes use of a linear distribution function used to not only speed up code execution when compared to for loop logic, but to make it easier on the programmer by eliminating the problems that may occur when distributing a number that can't be distributed evenly. For example: if using MATLAB®, $0:1/3:1 = [0, 3.3333333333e-001, 6.66666666667e-001, 1]$. MATLAB® does not simply start at 0.0 then add 0.3333 repeatedly; if it did the final value would never reach 1. Dealing with this concept was especially difficult for more complex linear distributions. A method was developed and

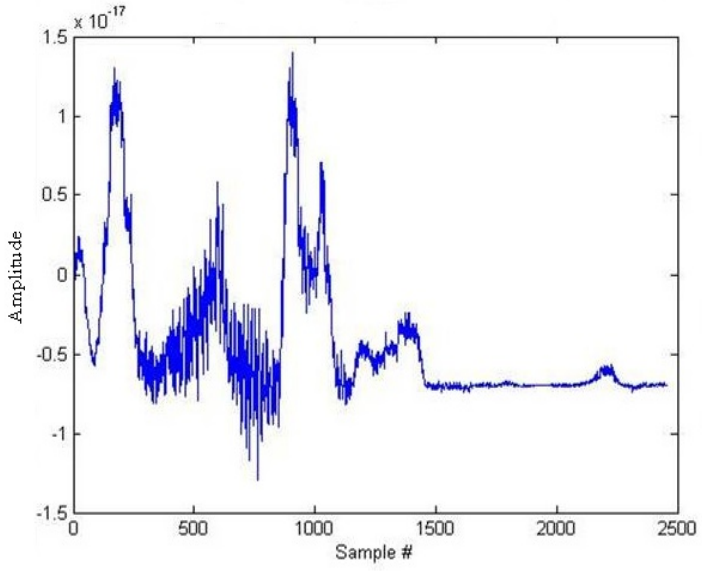

Figure 3.3: Filter Error between Matlab and Embedded C

utilized in C to deal with this problem. This linear distribution method is used to define a time index for the audio file. It is also used to refer to specific locations in the audio so that a specific sample can be pulled by looking it up by its respective time index.

3.1.3 Dynamic Clamped Cubic Spline Interpolation

The MATLAB® implementation of the EMD algorithm utilized the built-in clamped cubic spline function. The interpolation of the maxima and minima points to create the upper and lower envelopes is perhaps the most important aspect of the algorithm. The speed and accuracy of the algorithm is mostly dependent upon the speed and accuracy of the clamped cubic spline interpolation. The taxing and numerous calculations per interpolation multiplied by the 1000s of loop iterations to determine each IMF is why the interpolation speed and accuracy is crucial.

The C implementation of a dynamic clamped cubic spline interpolation was not only crucial, it was by far the most difficult. The core foundation of the cubic spline

interpolation was pulled from [5]. This core foundation was further expanded to clamp the end points to a first derivative of 0 and to dynamically determine the number of points between each maxima and minima point while using the audio signals total number of samples as a reference to make sure the time index remains the same for each sample.

MATLAB® compiles and executes code line by line. This meant that MATLAB® itself would be the best tool for designing, testing, and debugging a C version of a clamped cubic spline interpolation algorithm. Strictly using only C logic, an almost exact replica of Matlab's spline function was developed. Once the C version of the interpolation was developed in MATLAB®, it was used to replace the MATLAB® spline function used in the original EMD algorithm. Figure 3.4 shows the results of processing DS1\Answer1.wav on its first cubic spline calculation of the first upper envelope.

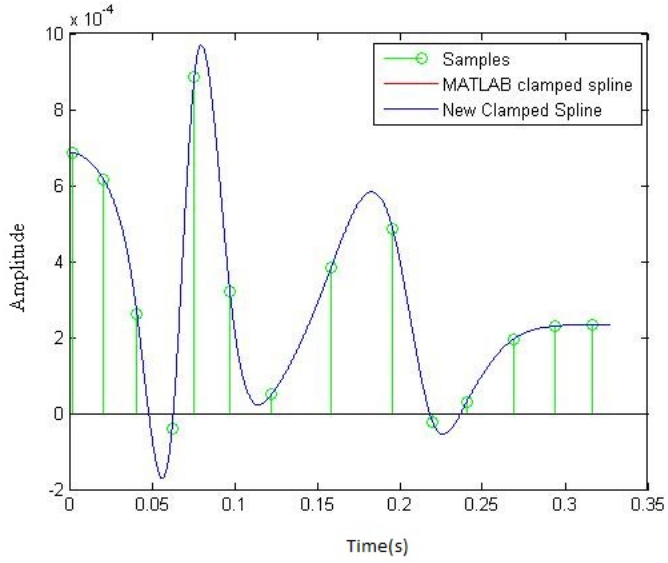

Figure 3.4: Matlab's spline function versus C spline on DS1\Answer1

The results in figure 3.4 show the comparative results of both clamped cubic splines with a dynamic number of samples to solve for between each given point. Since these

comparative results were so close to exact, the absolute error was calculated and is shown below in figure 3.5.

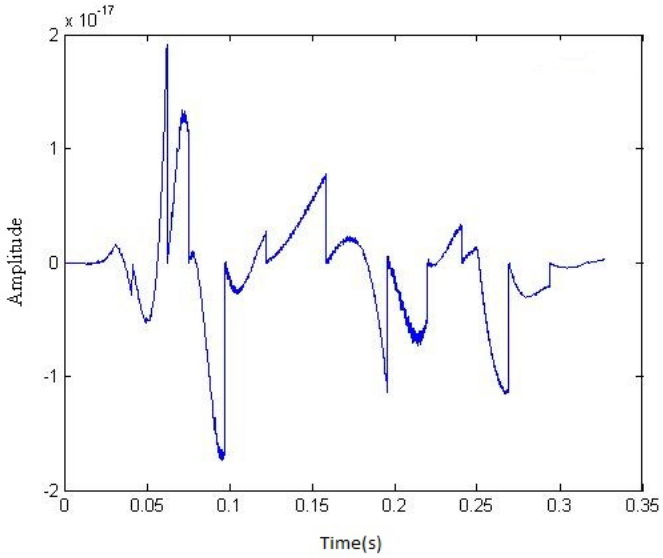

Figure 3.5: Cubic Spline Error between Matlab and Embedded C

The C implementation of the dynamic clamped cubic spline was tested on several iterations of the EMD process for several previously tested audio files. To completely confirm its accuracy, all of the previously tested audio files were tested to the completion of the EMD process. All resulting IMFs and residues showed minimal to no error.

The new dynamic clamped cubic spline interpolation algorithm did have one major flaw. During the MATLAB® testing of the new C spline, it quickly became evident that it is much slower than the built in MATLAB® spline function. The EMD process on roughly a $1/4$ sec of audio which results in about 5 IMFs, has a 10 to 12 hour completion time. This was an extremely detrimental find, since the original algorithm only took roughly 12 sec under the same circumstances.

The cause of such a drastic change in completion time depended on several factors.

The time was affected by the processing speed of MATLAB®'s line-by-line compilation and execution of the C code. With that said, Matlab is a program made to compile and execute code on a higher level of the computer's architecture in which processing priority is given to other functions, thus hindering the speed. Finally, the simple presence of several for-loops will simply cause a reduction in speed, in any case.

3.1.4 Standard MATH.h library functions

The MATLAB® algorithm employed several other functions that didn't work the way they were intended within the C implementation. More than likely this was due to the fact that the C compilers are 64-bit compilers and the processors are 32-bit. All the functions contained within the standard MATH.h library did not seem to work with certain numbers and/or very small numbers. The following functions had to be re-written to ensure the proper mathematical execution of the algorithm using numbers with a 64-bit resolution on the embedded device: absolute value, exponential powers, square root, and sum of products.

3.2 Hardware Implementation

The embedded C algorithm was first implemented on to the Spectrum Digital TMS320C6416 DSP board due to its high speed 1 GHz processor and ability to read values stored in various memory locations at any given moment. During the development, MATLAB® was used as a cross reference, and to help with the break down of the original EMD algorithm piece by piece. The embedded C was later moved over to the Atmel EVK1104 strictly for its interfacing features and drivers.

3.2.1 TMS320C6416 DSP Design

The EMD algorithm was successfully implemented onto Spectrum Digital's TMS320C6416 DSK DSP board. The original algorithm was slowly broken down line by line and slowly implemented and debugged using Matlab as a comparison. In order to eliminate the possibility of future issues that the algorithm may experience with certain types of data, several specific situational test were ran at key strategic points in the development process. Some of the tests ran can be seen below:

1. Signals with odd and even number of total samples, or resulting with odd and even numbers of zeros between each max or min point.
2. Signals that result with an upper or lower envelope with both positive and negative maxima and minima present.
3. Signals that result in a maxima or minima point at the last sample of the signal.
4. Signals that would cause specific mathematical result to be too small for the 64-bit resolution to store and display, and how these zeros are handled.
5. Really short signals were tested.

Figure 3.6 shows the step-by-step flow of the DSP implementation. Halfway into the development, a lack of memory resources was observed. With some work, the external SDRAM was employed as additional memory resources that would contain the memory address of all 64-bit double floating point arrays. With more resources available the base EMD algorithm was implemented onto the hardware.

Once the base EMD algorithm was implemented, it was decided that the algorithm would be limited to only process a max of 1 sec of audio sampled at 8 ksamples per sec.

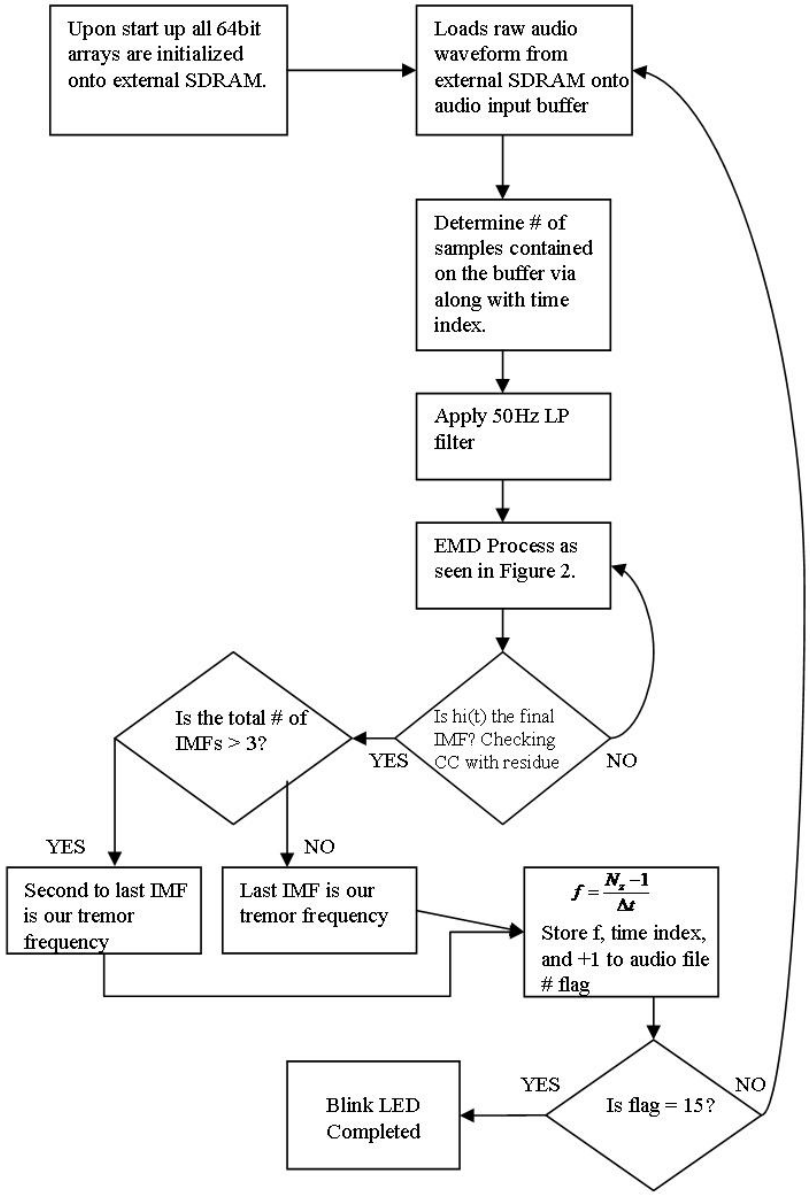

Figure 3.6: Spectrum Digital TMS320C6416 DSP design

With a max buffer size of 8000 samples, the EMD algorithm was modified to dynamically accept and process any audio file that is 1 sec or shorter. Once the audio size limitation was in place, the rest of the VSA algorithm could be added to the base EMD algorithm.

Up to this point in the design, a post-filtered signal is used as the input signal to eliminate error that could have been created by the C implementation of the 50 Hz low-pass filter. With the confirmed accuracy of the base EMD implementation the 50 Hz low-pass filter was implemented.

Once the algorithm finishes processing the first audio signal data, the number of zero crossings contained within the tremor IMF was determined. Equation (3.1) was applied to determine the average frequency.

$$f = \frac{N_z - 1}{\Delta t} \tag{3.1}$$

The algorithm is repeated once all variables are returned to there original values. The EMD process repeats for up to 15 audio signals. A blinking LED signifies that all audio signals have finish processing.

3.2.2 Atmel AVR32 EVK1104 Design

The Spectrum Digital DSP board may have been fast but it did not have the proper features to provide for an easy implementation of an interface. The Atmel AVR32 EVK1104 development board was chosen to bridge the gap. Sacrificing speed and memory resources, the EVK1104 turned out to be a good platform for the final product. Figure 3.7 shows the flow of the EMD algorithm embedded on the EVK1104 board.

The embedded C source code used in the Spectrum Digital DSP board implementation was copied directly onto the EVK1104 board, with minimal difficulties. The

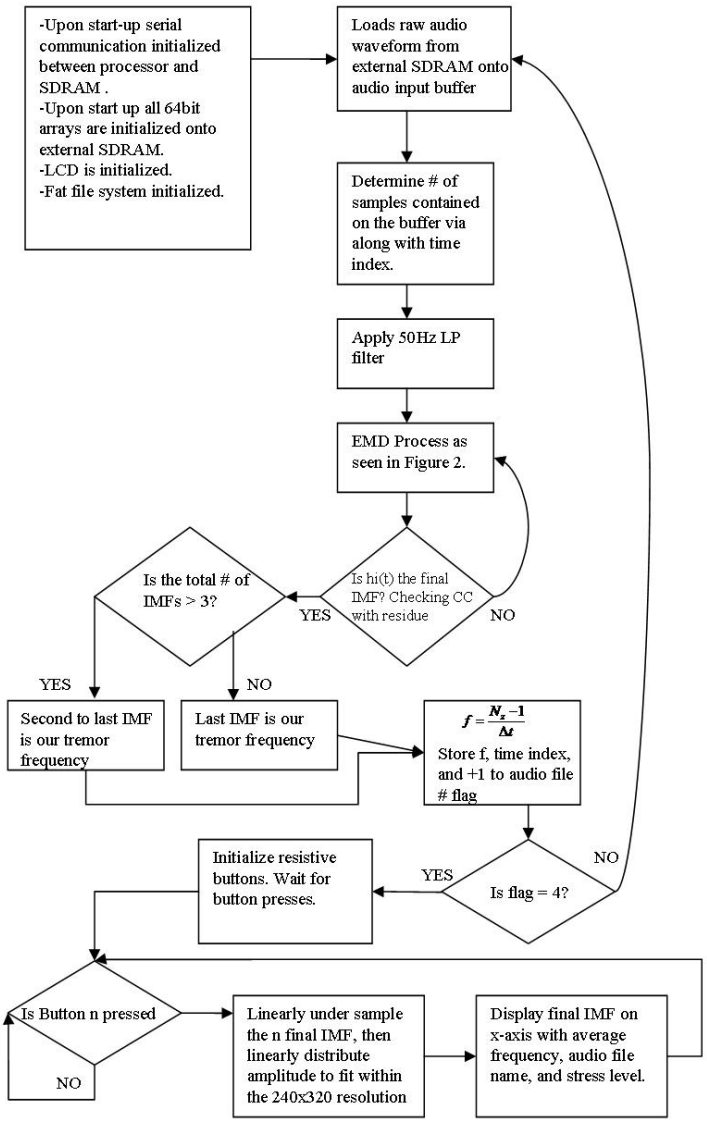

Figure 3.7: Atmel EVK1104 Design

EVK1104 was not only slower, it also had less memory resources despite the fact that more was needed for this implementation than that of the DSP board implementation. The additional resources needed were for mainly the LCD display. It behaves the same way the DSP board implementation does, except that it can only process only 4 audio files at a time. It can only process 4 audio files due to the lack of memory resources.

Once the four audio files have been processed, a button maybe pressed to show the subsequent audio file results and display them to the screen. When a button is pressed all data must be linearly distributed to fit within the 320x240 resolution LCD screen. When a button is pressed the tremor IMF is under sampled by a linear distribution that results in a re-sampled 320 sample IMF. Using pixel row 120 as my reference to an amplitude of 0, the amplitude was linearly scaled to fit within a pixel row height of 240. A multiplier was applied to the equation to shrink the signal's amplitude on screen so that it would be much clearer and so the LCD could also display other information. The name of the audio file, average frequency, and stress level is displayed on screen along with the IMF signal with its associated time index.

CHAPTER 4: RESULTS

Understanding the MATLAB® EMD algorithm and reproducing N. Mbitiru's results [1] was the first step to successfully completing a hardware implementation. Once the hardware implementation was complete, the results were compared once again with N. Mbitiru's results. Most of the test comparisons were conducted using 30 specific voice audio files collected from interviews from 2 students. These 30 voice audio files were labeled Deceptive Subject 1 and 2 (DS1 and DS2) [1].

4.1 Relavant Analysis of Previous Results

In previous research [1] there were about 10 more Deceptive Subject tests conducted other than DS1 and DS2 that were thrown out of the previous research results for an undisclosed reason. Further inspection of this thrown out data revealed that the signal to noise ratio was extremely low, making it very difficult for the EMD algorithm to consistently process the audio properly based upon the predefined stopping criteria. It was determined that even in the DS1 and DS2 answers, the audio contained a large amount of values that were near zero. These near-zero values represent silence in the audio where voice is not present. The presence of this near-silence can possibly have an adverse affect on the algorithm's behavior when trying to determine valid IMFs. Depending on the amount of near-silence and signal-to-noise ratio, the results can vary drastically. This is a fact that can be easily proven and seen in the other Deceptive Subject tests. The thrown out Deceptive Subject audio files show the noise power of the voice signal to be relatively

on the same power level as the near-silence. Even if the algorithm's stopping criteria were adjusted to compensate for this, it would not work well because it is trying to extract IMFs out of a signal with a lot of additive noise with extra non-voice samples that contain high noise power. The only way to ensure that this doesn't happen is by processing only good and loud voice recordings and making sure that all non-voice elements are removed. Almost all the recordings in DS1 and DS2 showed a desired signal to noise ratio that was not present in the other DS tests. In future work, new audio recordings will be needed to conduct controlled experiments that can properly produce statistical evidence of this specific relationship. Proving this relationship was out of the scope of my research, but this discovered concept needs to be brought to attention.

4.2 New Metric for Determining Voice Stress

It became apparent that reproducing the results recorded in the previous research was not enough to prove that the algorithm worked properly. It only demonstrated that it worked by displaying a noticeable shift in the final IMF with respect to the specific question's purpose of either not invoking or invoking stress. This high low pattern seen from previous results was either way over or under what a valid microtremor frequency should be [1].

Once the results of the embedded algorithm confirmed a match with the MATLAB® results, the microtremor frequency offset issue was looked into. After considering and searching for different causes, it was determined that the second to last IMF would be the tremor frequency. If the total number of IMFs is less than or equal to 3, then the last IMF is considered the tremor frequency.

The effectiveness of this new metric for determining the final IMF was proven by using a series of sum-of-sine-waves as the inputs to the algorithm. In several cases, the

EMD algorithm would produce results that contained all the proper IMFs, as well as an extra IMF containing a frequency that does not exist within the input signal. Throwing out this phantom last IMF provides for a good, yet temporary solution until more controlled statistical tests can be conducted. Running initial tests for the reason why this happens expands beyond just one independent variable.

4.3 TMS320C6416 DSK Results

One of the biggest challenges of this project was maintaining a 64-bit resolution throughout all numeric calculations done during the EMD process. This process had to be completed for 15 audio files in a row while maintaining accuracy. The resulting IMFs were accurate to roughly the 17th decimal place. The next 8 figures show the tremor frequency IMF comparisons for the first 4 answers of DS1.

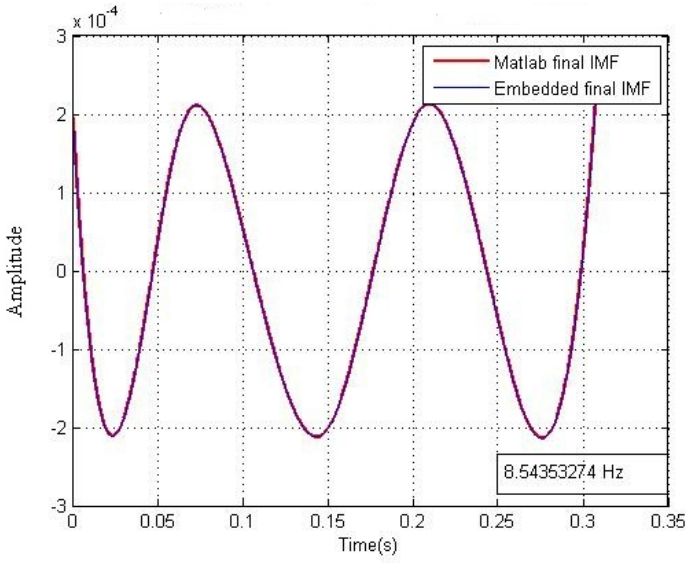

Figure 4.1: DS1\ Answer1.wav Tremor Results

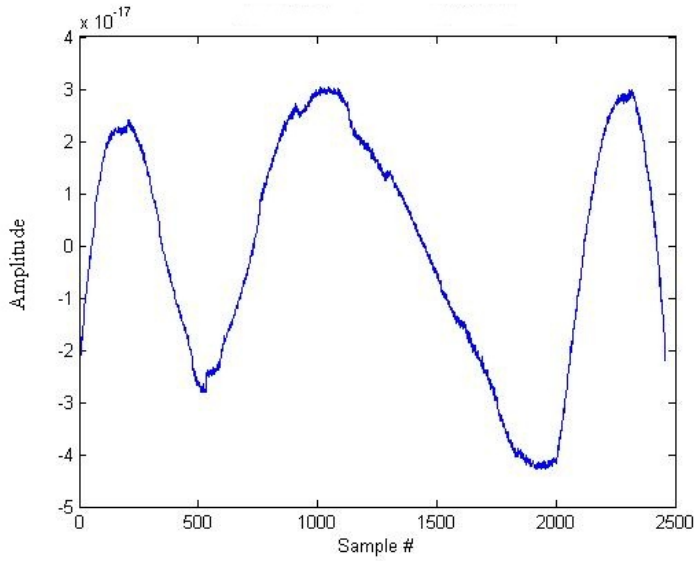

Figure 4.2: DS1\Answer1.wav Tremor IMF Matlab and Embedded C Error

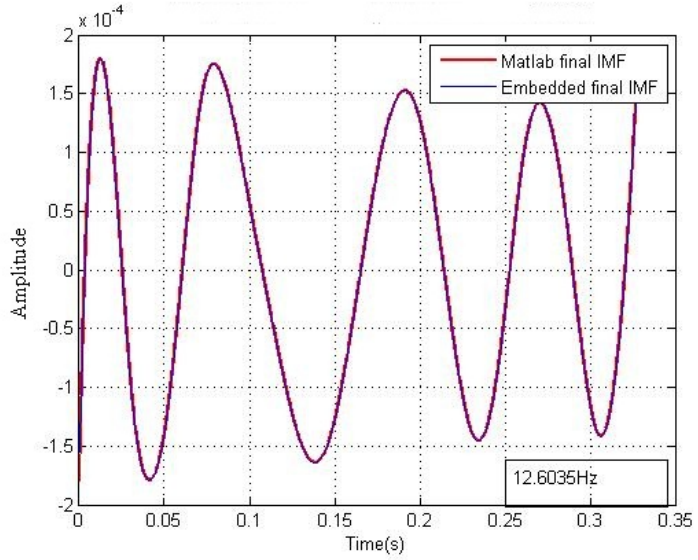

Figure 4.3: DS1\ Answer2.wav Tremor Results

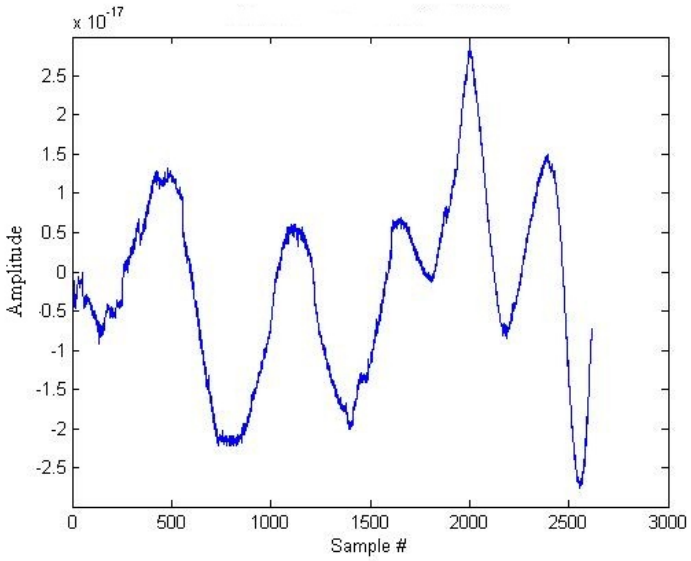

Figure 4.4: DS1\Answer2.wav Tremor IMF Matlab and Embedded C Error

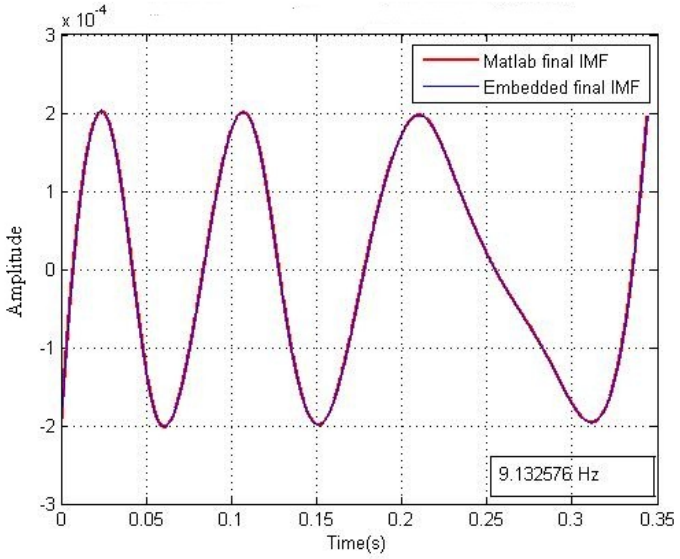

Figure 4.5: DS1\ Answer3.wav Tremor Results

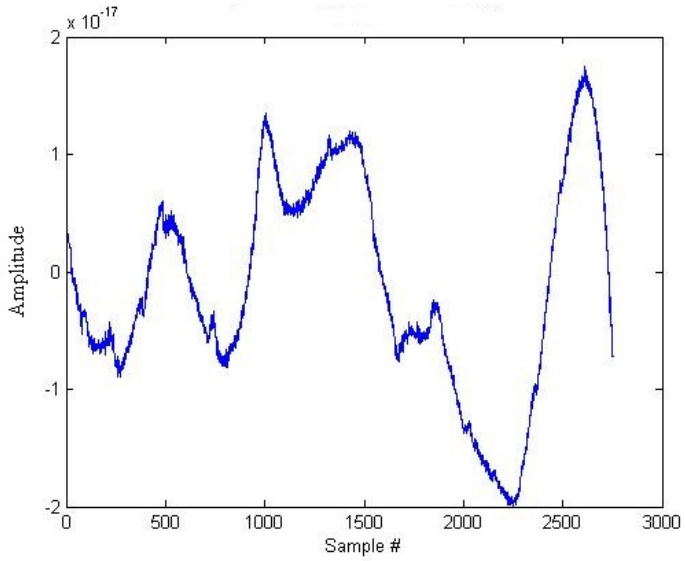

Figure 4.6: DS1\Answer3.wav Tremor IMF Matlab and Embedded C Error

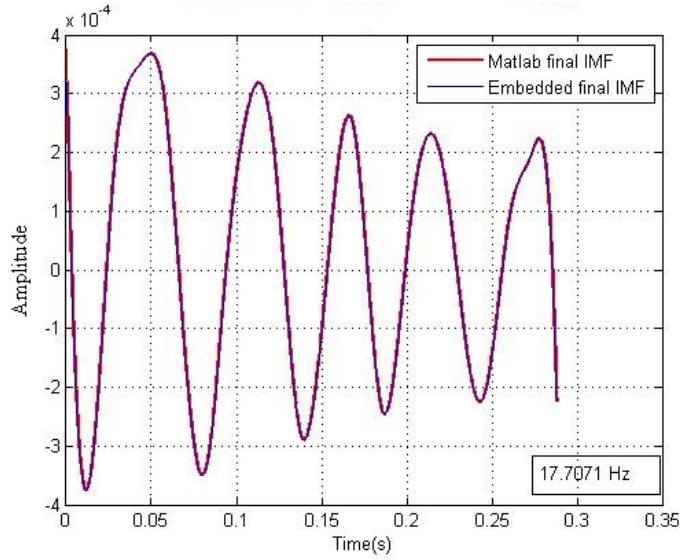

Figure 4.7: DS1\ Answer4.wav Tremor Results

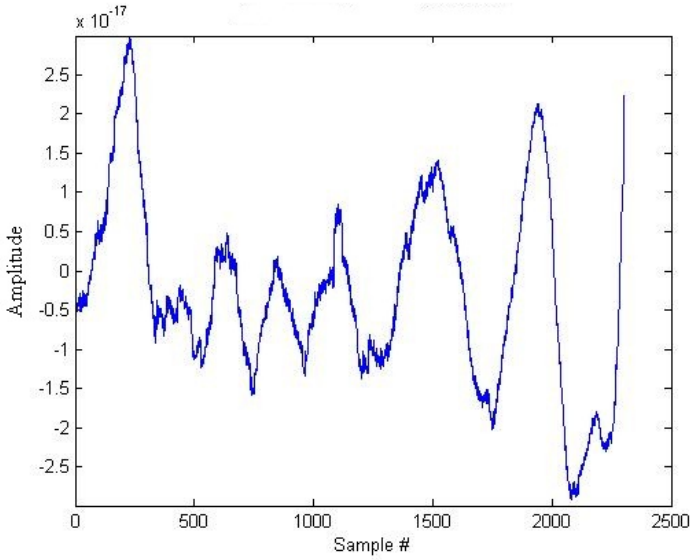

Figure 4.8: DS1\Answer4.wav Tremor IMF Matlab and Embedded C Error

In each case it can be seen that even after several iterations of calculations done to produce each tremor IMF, the associated absolute error between the MATLAB® and the embedded tremor IMF roughly never exceeds the 17th decimal place. A completely accurate and consistent hardware implementation has been achieved.

In order to make the comparisons seen in figures 4.1-4.8, the embedded data was copied from the Code Composer Studio® debugger to a .txt file. The .txt file was loaded into MATLAB® so that all the tremor IMF sample values could be sampled out and counted.

Figures 4.9 and 4.10 show all tremor frequencies and IMF# 's for all DS1 and DS2 answers. Both subjects displayed no measurable difference in calculated frequency. It can be observed in the DS1 results that 7 of the 8 questions designed to not invoke a stress response were within the 8 to 12 Hz range. Similarly, 6 of the 7 questions designed to invoke a stress response showed a shift in frequency outside the 8 to 12 Hz range. The DS2 results appear to share a similar amount of success, but it appears as though the subject

being interviewed had a stress level that generally increased from question to question but still showed proper stress patterns. It can be observed that 4 of the 8 questions designed to not invoke stress were actually within the 8 to 12 Hz range. Also, 7 of 7 questions designed to invoke stress were outside the 8 to 12 Hz range.

	MATLAB		Embedded C	
DS1 Response	IMF Frequency (Hz)	IMF # used	IMF Frequency (Hz)	IMF # used
Answer 1	8.54353274	5/6	8.54353274	5/6
Answer 2 MSR	12.60353148	4/5	12.60353148	4/5
Answer 3	9.13257673	5/6	9.13257673	5/6
Answer 4 SR	17.70710314	3/4	17.70710314	3/4
Answer 5	9.049926242	5/6	9.049926242	5/6
Answer 6 SR	12.50015979	4/5	12.50015979	4/5
Answer 7	12.75454609	5/6	12.75454609	5/6
Answer 8 MSR	13.23973086	4/5	13.23973086	4/5
Answer 9	10.0915464	5/6	10.0915464	5/6
Answer 10 SR	13.68119233	5/6	13.68119233	5/6
Answer 11	10.39893457	4/5	10.39893457	4/5
Answer 12 SR	8.609721332	5/6	8.609721332	5/6
Answer 13	10.84388468	4/5	10.84388468	4/5
Answer 14 SR	33.30617026	2/2	33.30617026	2/2
Answer 15	9.929863899	4/5	9.929863899	4/5

Figure 4.9: DS1 Tremor Frequencies Matlab and Embedded C

	MATLAB		Embedded C	
DS2 Response	IMF Frequency (Hz)	IMF # used	IMF Frequency (Hz)	IMF # used
Answer 1	11.05864479	4/5	11.05864479	4/5
Answer 2 MSR	19.22280948	3/4	19.22280948	3/4
Answer 3	8.940695057	6/7	8.940695057	6/7
Answer 4 SR	17.14731524	3/4	17.14731524	3/4
Answer 5	9.704952781	5/6	9.704952781	5/6
Answer 6 SR	18.21780893	3/4	18.21780893	3/4
Answer 7	12.69121813	3/4	12.69121813	3/4
Answer 8 MSR	15.60077678	4/5	15.60077678	4/5
Answer 9	10.12302784	4/5	10.12302784	4/5
Answer 10 SR	31.10169513	2/2	31.10169513	2/2
Answer 11	16.51211561	5/6	16.51211561	5/6
Answer 12 SR	21.85829063	3/4	21.85829063	3/4
Answer 13	13.16283083	4/5	13.16283083	4/5
Answer 14 SR	14.51927035	5/6	14.51927035	5/6
Answer 15	13.57823015	3/4	13.57823015	3/4

Figure 4.10: DS2 Tremor Frequencies Matlab and Embedded C

4.4 Final Hardware Implementaton

There was not any varying differences between the DSP implementation results and the EVK1104 results. Once the EVK1104 finished processing the audio files, the LCD would display the following image that can be seen in figure 4.11. Figures 4.12-4.15 show proof of proper operation. Each figure displays the under sampled IMF with the time index. They also display the average frequency and stress level.

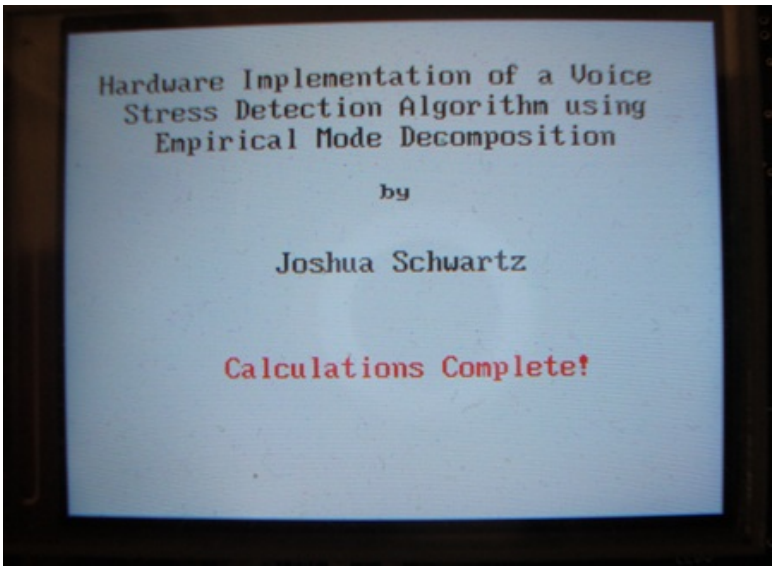

Figure 4.11: EVK1104 Calculations Completed

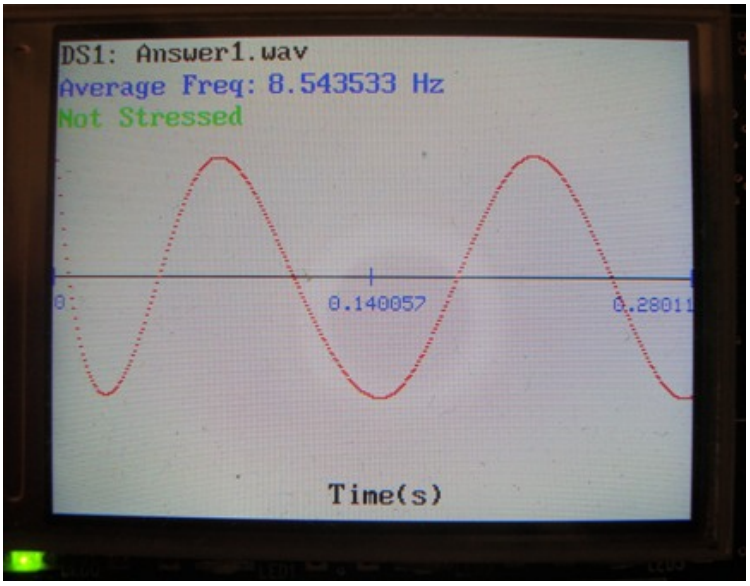

Figure 4.12: EVK1104 DS1\Answer1 Tremor IMF

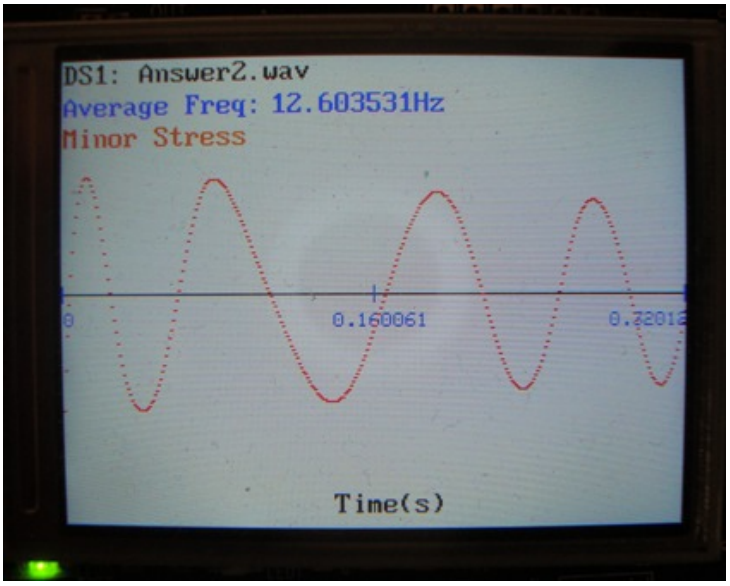

Figure 4.13: EVK1104 DS1\Answer2 Tremor IMF

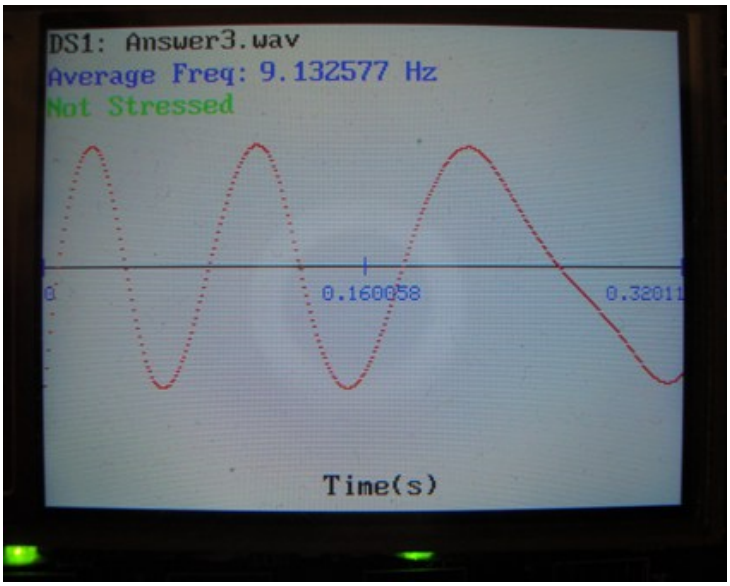

Figure 4.14: EVK1104 DS1\Answer3 Tremor IMF

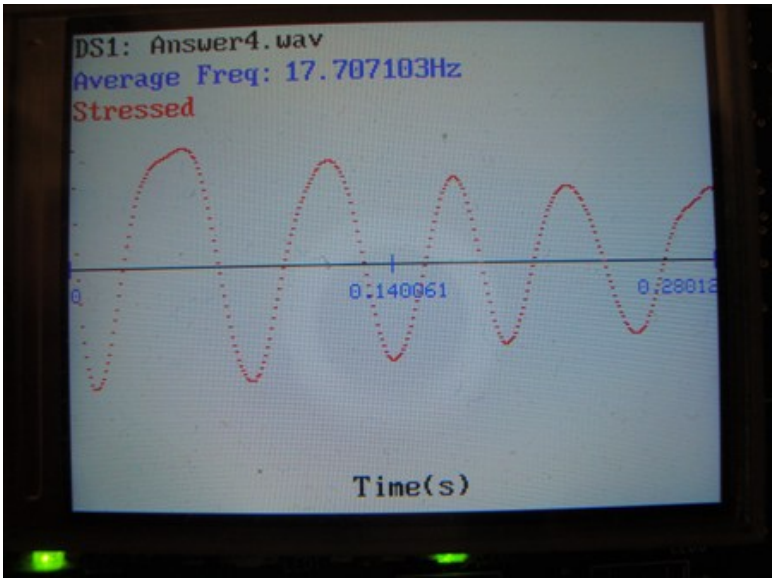

Figure 4.15: EVK1104 DS1\Answer4 Tremor IMF

CHAPTER 5: CONCLUSION AND FUTURE WORK

In this thesis, a C algorithm for detecting voice stress, based upon the EMD method was implemented onto two different hardware platforms. The initial implementation onto the TMS320C6416 DSP board showed near perfect results when compared to the results presented in MATLAB®. This accuracy was seen consistently throughout all testing for both implementations. If an audio file $\frac{1}{3}$ of a second long produced roughly 5 IMFs, the processing time for completion is roughly 30 seconds. Given the same situation on the Atmel AVR32 EVK1104, the processing time would roughly be two hours. This is due to the slow clock speed. A GUI has been implemented to display the results graphically and numerically on a 240x320 LCD display.

The next step in the development process would be to get a more resourceful and faster processor for final product implementation. A higher resolution LCD would be beneficial as well. Before this should happen several other factors must be dealt with and/or considered.

To improve the speed dramatically, the C code can be rewritten to include several calculations within subsequent loops. The most significant time waster is the cubic spline calculation that is applied to create each envelope. This cubic spline algorithm is applied thousands of times. Currently, for each iteration of the EMD process, the cubic spline algorithm is applied to create the upper envelope. It then starts over and creates the lower envelope. The two envelopes could instead be generated at the same time within the same iteration. Applying similar techniques to the rest of the EMD algorithm accompanied with

a high speed clock, could make real-time stress detection possible.

There is still much work to be done to improve the algorithm itself. Since everyone's tremor threshold can differ, it would make sense to apply an adaptive EMD method [2] that controls the stopping criteria of the IMF detection. The metric for determining voice stress previously explained in the results section, is a temporary method that needs further controlled testing to either prove or disprove its validity. This controlled testing should involve several interviews similar to DS1 and DS2. The signal-to-noise ratio of the audio recordings need to be controlled in such a way to provide statistical evidence of its influence on the EMD process. Similarly, the impact of removing quite samples or silence from the beginning and end of the audio should be statistically measured. These statistical measurements should then be weighted by the length of audio file. This statistical data could then be used to help determine ideal audio input requirements versus stopping criteria adaptation and limitations.

The algorithm could be implemented as a smart phone application in the future. This will especially be a challenge. The development tools generally limit the programmer's control over certain aspects of the device. This is done to prevent application developers from writing viruses that effect the device's initial functionality. It is also done this way to protect the company's proprietary hardware and software from abuse. To implement this resource hungry algorithm on and around all these controls could be a challenge. If this challenge were met, the question is how fast will it process the audio if the algorithm is on top of an OS? The algorithm will be much slower when implemented as a high level program. Further research will be needed to find out if this is a practical implementation with current smart phone technology.

BIBLIOGRAPHY

[1] N. Mbitiru, *Analysis of stress in speech using empirical mode decomposition*, M.S. Thesis, Western Carolina University, 2009.

[2] James Z. Zhang, Nyaga Mbitiru, Peter C. Tay, and Robert D. Adams, *Analysis of stress in speech using adaptive empirical mode decomposition*, 43rd Asilomar Conference on Signals, Systems, and Computers, Pacific Grove, CA, 2009.

[3] N. E. Huang, M. C. Wu, H. H. Shih, Q. Zheng, N. C. Yen, C. C. Tung, S. R. Long and H. H. Liu, *The empirical mode decomposition and the Hilbert spectrum for nonlinear and non-stationary time series analysis*, The Royal Society, London, 1998.

[4] Spectrum Digital, INC., Stafford, TX, *TMS320C6416T DSK Technical Reference*, 2004.

[5] J. Douglas Faires and Richard L. Burden, *Numerical Analysis*, Bob Pirtle, Belmont, CA, 2002.