DESIGN OF AN ARTIFICIAL COCHLEA USING DIGITAL FILTERS ON A FIELD-PROGRAMMABLE GATE ARRAY

NSF Summer Undergraduate Fellowship in Sensor Technologies
Aslan Ettehadieh (Electrical Engineering) – Morgan State University
Advisors: Dr. Jan Van der Spiegel and Dr. Paul Mueller

ABSTRACT

Traditional methods of speech recognition have very limited complexity and impose considerable grammar constraints. Today's systems have critical problems understanding different voices and do not have robust vocabularies. This paper describes research on a biological method for speech recognition that models an artificial cochlea using digital filters. The specific part of the cochlea of interest here, the basilar membrane acts as a collection of bandpass filters that will be mimicked to develop an artificial digital cochlea.

This digital cochlea will be implemented on a single Xilinx field programmable gate array (FPGA). The FPGA chip is on a board that contains both an analog to digital (A/D) converter and a digital to analog (D/A) converter. The A/D converter converts the speech signal into a digital representation; the D/A converter converts the digital signal back into its original analog form. The board also includes a lowpass anti-aliasing filter to reduce the noise and keep only the range of human speech frequencies approximately 100 to 3500 Hertz. The range of frequencies will be divided into 16 bands. The FPGA chip will contain 16 programmable bandpass filters to split up the speech signal into separate frequency components that can be used to determine phonemes, the simplest unit of speech. Phoneme-level recognition will improve the speed and accuracy of speech recognition.

The final product is expected to be cost efficient and to be implemented on a single chip. The digital cochlea will be used in conjunction with a neural network that will extract features and phonemes from speech signals.

1. INTRODUCTION

Speech recognition technology has become more popular and has advanced considerably over the past several years. Recognition software has been integrated into many applications. For example, Eloquently Stated, a software package for medical professionals, adds speech recognition to manage patient records and medical histories, and creates comprehensive referral source databases. Another application of speech recognition technology is language translation. Lernout & Hauspie has released a software package that can translate to and from English and: Spanish, German, French, Italian, Portuguese, and Japanese. [1]

Speech recognition in traditional methods has limited complexity and artificial grammar constraints. The importance of our speech recognition research is to have a design that is more robust in its ability to recognize speech. Drawing on a biological method for speech recognition, this research will model a design after the basilar membrane in the cochlea of a human ear.

There are three major divisions of the peripheral auditory system: the outer ear, middle ear, and inner ear. Without great detail the outer and middle parts of the ear are responsible for transforming and transporting sound to vibrations to the inner ear. The major component of the inner ear, the cochlea, cochlea converts mechanical vibrations caused by sound waves into electrical impulses. The cochlea is a coiled tube that looks like a snail and is filled with fluid. The basilar membrane is located about halfway down the cochlea's length and within the cochlear fluid. It is held to the cochlea by bone. A compressed sound wave is generated in the cochlear fluid by the vibrations of the eardrum, which result from movement where the middle ear is connected to the inner ear by way of the oval window. The compressed sound wave generated in the cochlear fluid causes an up-and-down vibration of the basilar membrane. Along the basilar membrane are thousands of inner hair cells that, when simulated by vibration "fire" short electrical pulses in the nerve fibers. The nerve fibers are bunched together to form the auditory nerve. When these electronic pulses travel along the auditory nerve, they find their way to the higher levels of the auditory processing in the brain. This is where the brain perceives what is heard as sound. [2]

When the ear is stimulated by sound, different regions of the basilar membrane respond to different frequencies that occur in a sort of "tuning" of frequencies. These different regions can be translated as a bank of cochlear filters along the basilar membrane. The cochlea consists of thousands of filters. Being practically for today's limitations, a design on a single field-programmable gate array (FPGA) could not hold enough memory to have a bank of thousands of filters working properly. The University of Pennsylvania built a neural network that can extract features and phonemes from speech signals. This neural network was designed to work in conjunction with a digital cochlea with a bank of 16 filters. The design described in this research is limited to the maximum memory capacity of the FPGA and leads to 16 separate filters.

The frequency range of human speech is approximately from 100 to 3500 Hertz. Since the design can have only 16filters, 16 center frequencies have been chosen to cover the range of human speech.

The digital cochlea will be downloaded into an FPGA and implemented into a neural network. This neural network will investigate phonemes in order to facilitate the later design of a system that can recognize these phonemes.

2. BACKGROUND

This project, among many other attempts since 1998 at the University of Pennsylvania, has been done primarily because of the doctoral research of Ahmed Ali. Ali's dissertation *Auditory-Based Acoustic-Phonetic Signal Processing for Robust Continuous Speech Recognition*, postulates a biological approach to speech recognition. He proposes to replicate cochlea-like filtering behaviors that will be able to recognize more spontaneous speech than can be recognized by current systems, which have very limited perplexity and artificial grammar constraints. [3]

Cheng and Edelman, attempted to implement 36 analog programmable cascading filters on a Xilinx FPGA chip. However, their design, which required roughly 80,000 gates, was too complex for the single proposed chip, which contained only 5,000 gates. [4]

Lee and Lee [5] attempted the first design of a 16-channel filter system. Because of time limitations, their designs were only simulated and not implemented on an FPGA chip.

Chen, Gaw, and Raskob continued the project based on Lee and Lee's work. They were able to download a 16-channel first-order bandpass filter system on a single FPGA. Their design showed only minimal promise in the eleventh filter. Time constraints left them unable to optimize their system design.

Some related work has been conducted outside the University of Pennsylvania. Hinck's unsuccessfully attempted to implement a design consisting of a 6 to 10 channel digital cochlea filter on an FPGA. [6] Watts built a functioning real-time, high-resolution, 240-tap, 10-octave, 44 kHz-sampling cochlear model on multiple FPGAs. [7]

3. STRATEGIC PLAN

This research will consist of four general stages. The first stage is to research current understanding of the biology of the human cochlea and to become familiar with software and hardware components that will be needed to complete the research. The second stage is to design models using software packages. Simulation of these designs will be necessary to ensure accuracies in the filter designs. The third stage is to implement the architectural design to VHDL code and download it to the FPGA. The last stage is to test the FPGA to compare simulations with actual real-time testing.

3.1 Proposed Approach

The main goal of this project is to create one working model of an FPGA with digital filters that mimic the basilar membrane of the cochlea. This FPGA chip must be optimized to a satisfactory performance level where the center frequency of a certain filter does not resonate in a neighboring filter. Filter designs will be constructed and tested on a software level. When desired results are reached, the filters will be downloaded and tested at a hardware level. Then, the filters will again be optimized further, downloaded again into the FPGA, and tested.

3.2 Hardware And Software Requirements

Much of the hardware and software was made available by past attempts at the project. New licensing was necessary to use software that had expired. Cables were previously made to connect to and from the board. The co-axel cables were connected to audio jacks had been attached with the wrong polarities. Polarities were corrected and were soldered to permanently fix the problem.

3.2.1 Hardware Requirements

The FPGA chip, XCV300, and the processing board, XSV300, were both already available. The board was tested with sample inputs to ensure proper working conditions. A sample test provided with the board, called GXStest, for unknown reasons gave the error that the board was not properly functioning. A zip file was obtained from XESS Corporation called loop-simple.zip. It contained a file named loopv300.bit that was used to test the input versus output of the board. This file proved that the board would function properly for the necessary tasks. [8]

3.2.1.1 FPGA Chip

The Xilinx Corporation manufactures a series of Virtex processing boards for FPGAs. The exact board and chip were predetermined because they were the only hardware components available with an audio compatibility and built-in A/D converter and D/A converter. The specifications are shown in Table 1. More specifications on this product can be found on the AK4520A datasheet provided by the Xilinx Corporation.

XCV300 Specifications	
System Gates:	322,970
CLB Array	32 x 48
Logic Cells:	6,912
Maximum Available I/O:	316

Table 1, XCV300 specifications

3.2.1.2 Virtex Processing Board

The board is described in Section 4.8.1.1. The board provides an environment for the XCV300 chip that makes it ideal for testing. The XCV300 chip is directly mounted onto the board, as shown in Figure 1.

Figure 1: XSV 300 prototyping board

3.2.2 Software Requirements

There are three main manufacturers of software needed for this research. The MathWorks Corporation provides Matlab R12.1, Simulink, and DSP Blockset. The Xilinx Corporation provides System Generator and ISE. XESS Corporation provides XSTools. Table 2 outlines the software packages that will be used in the project.

Software	**Purpose**
MathWorks Matlab	Holds the environment needed to run Simulink and the DSP Blockset.
MathWorks Simulink	Provides a library of blocks that represent commonly used functions for modeling, simulating, and analyzing dynamic systems.
MathWorks DSP Blockset	Provides the ability to simulate signals for testing. Provides digital design blocks for generating filter coefficients.
Xilinx System Generator	Translates the Simulink model into VHDL code.
Xilinx ISE	Compiles the VHDL code into a bitstream file.
XESS XSTools	Downloads the bitstream file to the FPGA.

Table 2, Software packages with brief descriptions.

3.2.2.1 Mathworks Matlab R12.1 (v6.1)

Matlab is a software package used for mathematical computation and visualization. It provides an environment for technical computing. Matlab's open architecture makes it

easy to explore data and create algorithms. Matlab holds the environment for Simulink and the DSP Blockset.

3.2.2.1.1 Mathworks Simulink (v4.1)

Simulink provides an interactive tool for modeling, simulating, and analyzing dynamic systems. Simulink is a library of blocks that represent many commonly used functions. It provides the platform for Xilinx System Generator blocks to design systems for FPGAs. The license file need for Simulink and DSP Blockset packages had expired, and an updated license was purchased. Several working days were lost in the interim.

3.2.2.1.2 Mathworks Dsp Blockset (v4.1)

The DSP Blockset is an extension to the Simulink package that works in the Simulink environment. The DSP Blockset has the ability to simulate signals to test models. Key features include fast Fourier transform (FFT) and its inverse, short time FFT; multi-rate signal processing; FIR and IIR Direct Form II Transpose filters; adaptive filters; and digital filter design blocks for generating filter coefficients.

The DSP Blockset was used for the actual filter modeling and generating filter coefficients. Figure 2 shows a graphical user interface (GUI) for the design of filters. This particular model is a second-order Butterworth bandpass filter with a sampling frequency of 16000 Hertz, band start of 2705 Hertz, and band stop of 3095 Hertz. This is a second order Butterworth filter.

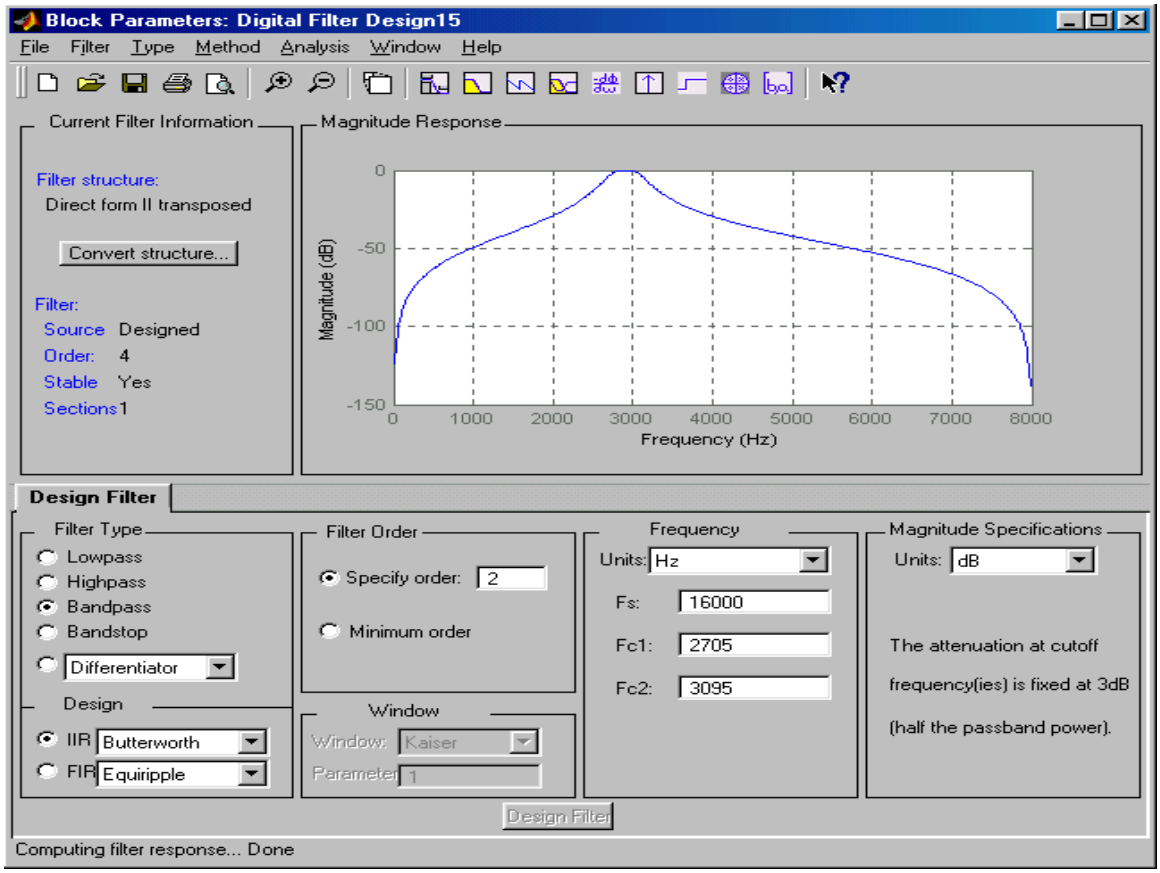


Figure 2: Digital filter design block.

The 'Analysis' option, located in the menu bar of the window, allows the user to view different items in the area presently marked 'Filter Specifications'. Figure 2 shows a diagram of the specifications of the filter to be designed. The magnitude response can also be viewed, as shown in Figure 3. Figure 4 shows the filter coefficients, from which we extract the coefficients to be used later.

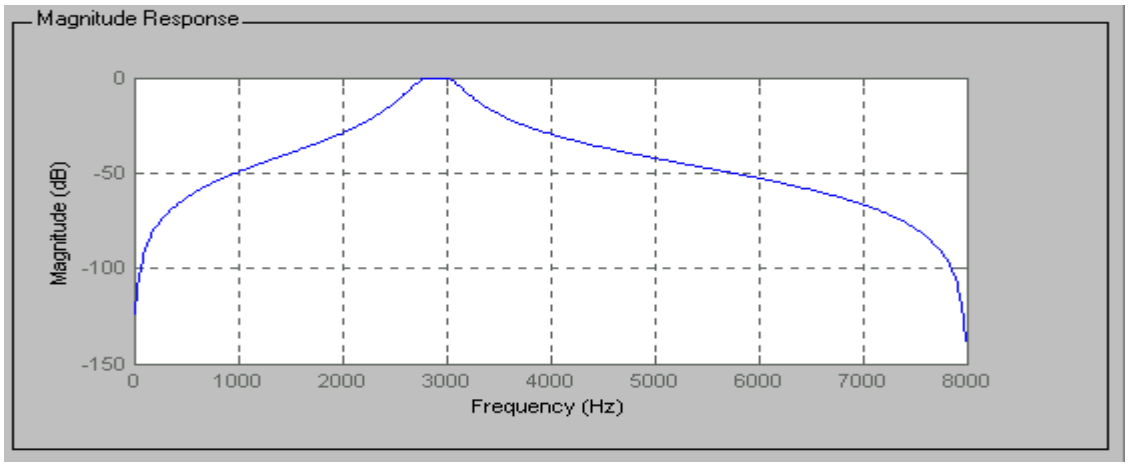


Figure 3: Digital filter design block: magnitude response.

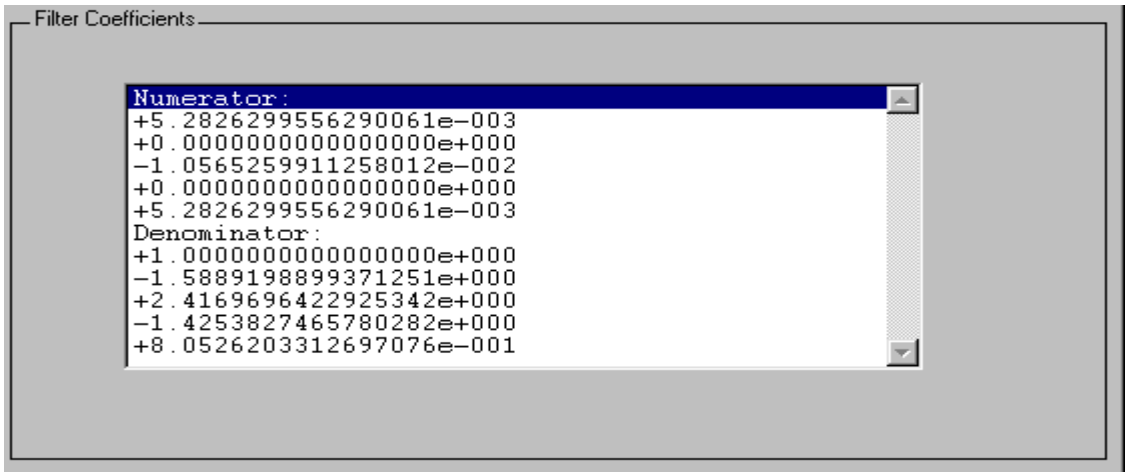


Figure 4: Digital filter design block: filter coefficients

3.2.2.2 XILINX

The Xilinx Corporation provides software and hardware that enable the programming of logic to devices like FPGAs.

3.2.2.2.1 XILINX System Generator (v2.1)

The System Generator allows a conceptual architectural design to be created with the MathWorks software packages to an actual Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL) code.

3.2.2.2.2 XILINX ISE (v4.2i)

The VHDL code created by the System Generator can be imported into the ISE environment. The ISE package makes an actual implementation of a bitstream file that can be downloaded to the FPGA.

3.2.2.3 XESS XSTools

With the use of the board's PC parallel port, XSTools downloads the bitstream file to the FPGA. XSTools provides features to test the functionality of the prototyping board.

3.3 Filter Design

The neural network that will be used was designed for 16 inputs. Bandpass filters are used because they tell us if a signal contains a frequency component with a specific frequency range. Using a bank of 16 such bandpass filters provides information about the frequency components in the signal. These 16 bandpass filters are all infinite impulse

response (IIR) filters. IIR filters are preferred to finite impulse response (FIR) filters because IIR filters can achieve a given filtering characteristic using less memory and calculations than similar FIR filters. The main drawback of using an IIR filter is that it is slower than an FIR filter to implement using fixed-point arithmetic.

A design of a single Xilinx chip has been implemented to have the 16 bandpass filters. Lowpass and highpass filters are to be optimized. Table 3 gives an example of the center frequency and a range of frequencies for the bandpass filters

Center Frequency (Hertz)	Frequency Range of Bandpass Filter (Hertz) (3dB-points)
150	125 – 175
250	225 – 275
350	325 – 375
450	420 – 480
570	530 – 605
700	655 – 745
840	790 – 890
1000	940 – 1060
1170	1105 – 1235
1370	1285 – 1455
1600	1505 – 1695
1850	1745 – 1955
2150	2005 – 2295
2500	2345 – 2655
2900	2705 – 3095
3400	3145 – 3655

Table 3, Center frequencies and their corresponding frequency ranges.

Fixed-point arithmetic will be used and manipulated to optimize the number of bits used while trying to keep the performance level close to full precision.

Matlab allows users to select the filter design method. All of the filters used in this project are Butterworth. In Butterworth filters the magnitude response is maximally flat in the passband and monotonic overall.

3.3.1 Filter Structure

Matlab is used to generate the necessary filter. Instead of the traditional direct form II structure, Matlab generates coefficients for the direct form II transpose structure using the general equation shown below.

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_1 + b_2 z^{-1} + \ldots + b_{m+1} z^{-(m-1)}}{a_1 + a_2 z^{-1} + \ldots + a_{n+1} z^{-(n-1)}}$$

The general diagram for the direct form II transpose general equation follows in Figure 5.

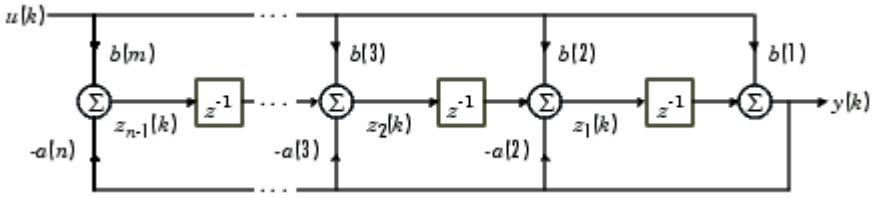


Figure 5: Direct form II transpose block diagram.

Different structures are to be considered over the simple direct form structure because even thought they are theoretically equivalent they may behave differently when implemented with finite numerical precision. [9] The direct form II transpose equation is preferred here because it cannot overflow "internally" in two's complement fixed-point arithmetic. The filter models are all designed in two's complement fixed-point arithmetic that implies the need for the direct form II transpose structure.

Fixed-point numbers are not as precise as floating-point numbers, but fixed-point hardware is much more cost effective, allowing a significant saving in hardware.

3.3.2 Filter Type

Of the many different types of filters available, only the lowpass, highpass, and bandpass types are needed for this research. Lowpass filters pass low frequencies and attenuate higher frequencies. Highpass filters do the opposite. Bandpass filters pass a limited range or band of frequencies, and attenuate frequencies above and below this range.

3.3.3 Filter Order

The order of a filter indicates the sharpness of the filtering or the slope of the rolloff curve. The higher-order filter designed, the sharper the filtering. A second-order filter provides much greater precision because of the use of more coefficients. The downside is that it uses more bits and therefore requires more memory in the system. Filters can be designed with the use of Simulink and DSP Blocksets. Figure 6 shows the block diagram of a first-order bandpass filter with center frequency of 2900 Hertz. Figure 7 shows a block diagram of a second-order bandpass filter with the same center frequency. It is apparent that the second-order filter has more blocks that would require more bits of memory.

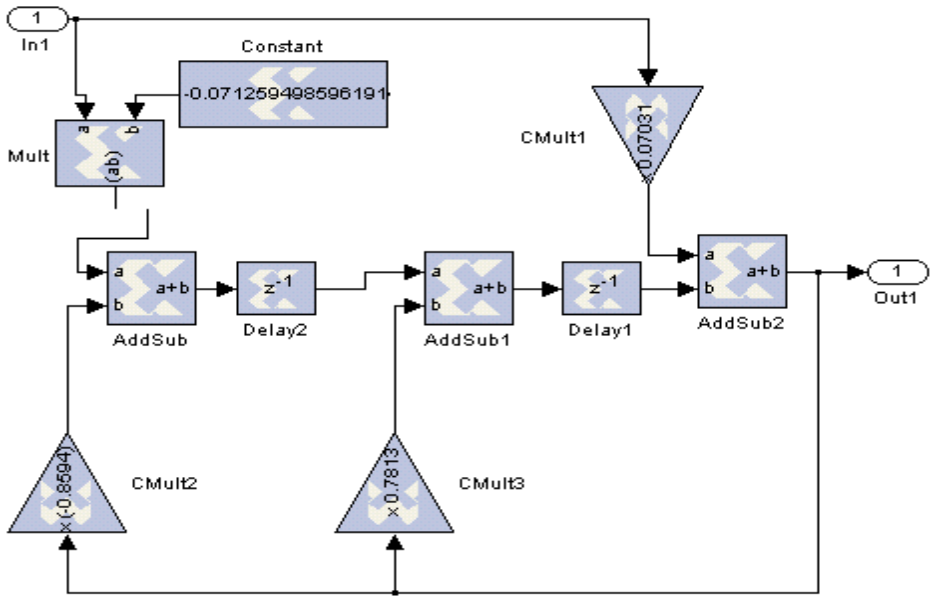


Figure 6: First order bandpass filter block diagram

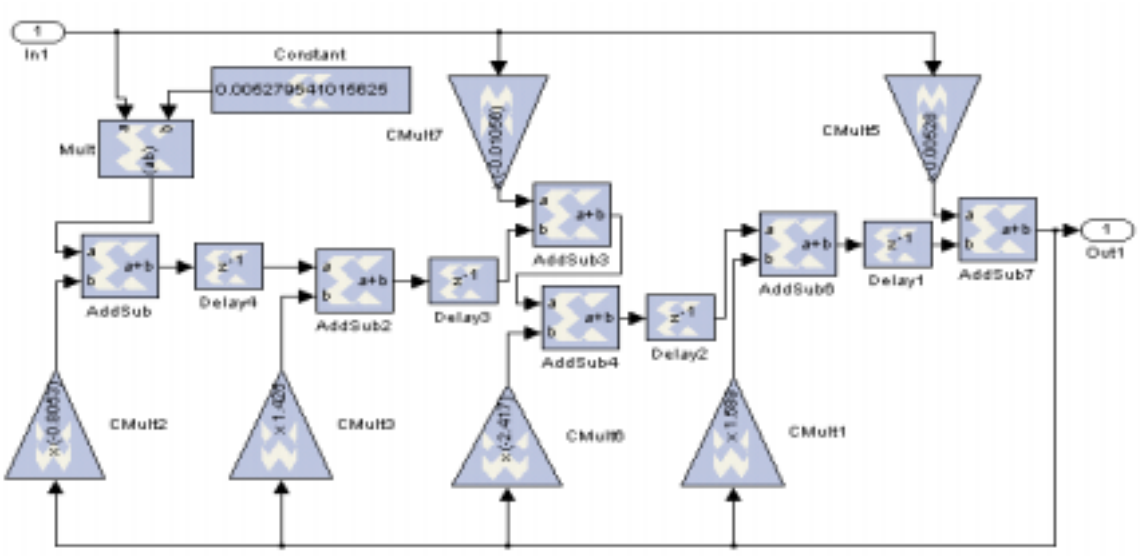


Figure 7: Second order bandpass filter block diagram

Adder (AddSub) blocks have only two inputs and one output. In the second-order filter, two adders must be joined together to allow three inputs to be added. To date, the software package's adders are able to add only two inputs.

3.3.4 Filter Diagrams

A few major designs proposed for comparison. A standard 16 bandpass filter was first drafted, the simplest of the designs. A simple series of four bandpass filters is shown in Figure 8.

A modified version of the 16 bandpass filters was also designed. This model is a chain of 16 lowpass filters that the bandpass filter taps off of. The cut-off frequencies of the lowpass filters are the same as those of the bandpass filters (see Table 3). This model decomposes a broadband signal into a collection of successively more band-limited components by repeatedly dividing the frequency range. In comparison to the basic 16 filters, this model gives an increase of one order higher to one side of the bandpass filters, resulting in a steeper roll-off. Therefore, if a bandpass filter was of the first-order, the bandpass would be theoretically equivalent to a first-order lowpass and first-order highpass. If lowpass filters were added before the bandpass, the design could be interpreted as a second-order lowpass and a first-order highpass. The left column of filters represents the lowpass filters and the right column represents the bandpass filters. A design of four lowpass to four bandpass filters is shown in Figure 9.

The last major design is the symmetric basic tree structure design. The tree structure decomposes both the high and low frequency sub-bands with lowpass and highpass filters at each level until the range of frequencies fit the desired range of the 16 bandpass filters. A standard four bandpass tree structure is shown in Figure 10.

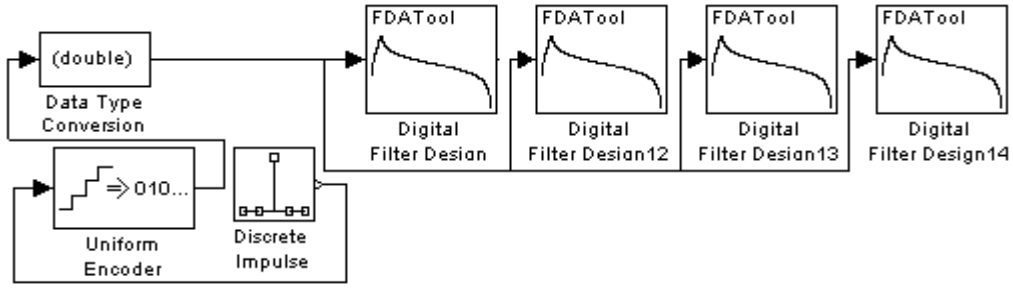


Figure 8: Basic filter diagram

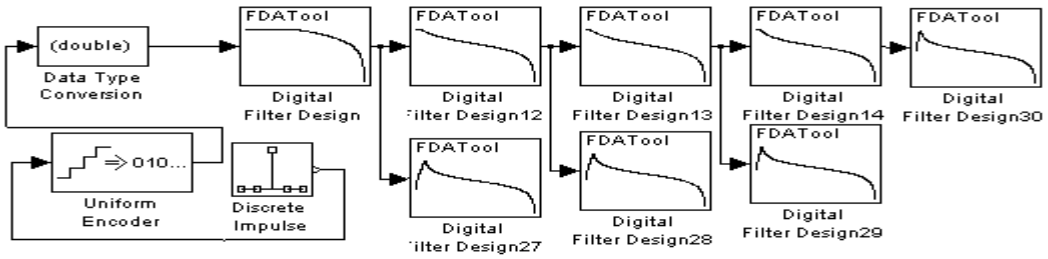


Figure 9: Lowpass to bandpass filter diagram

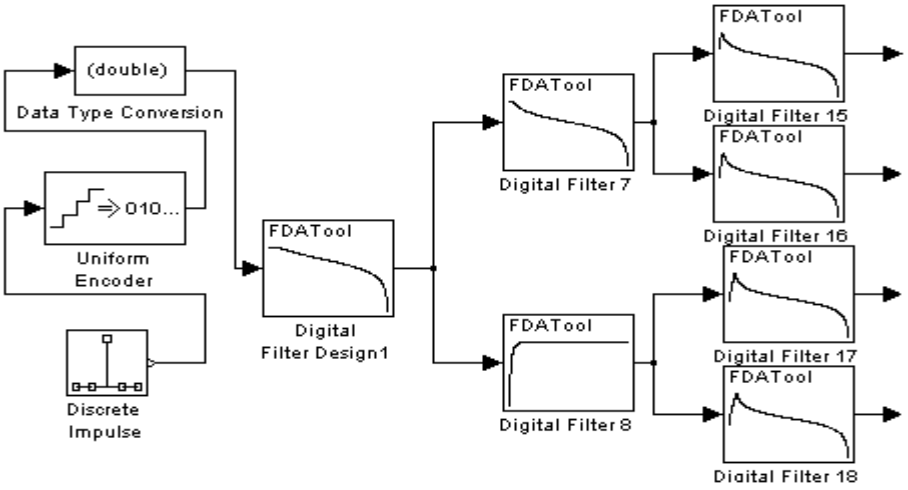


Figure 10: Tree structure filter diagram

Because of time constraints, most of the research has focused on the basic 16 filter design. Some results were simulated from the tree structure and the lowpass to highpass structure, but it was too early in the research and many of the software "bugs" had not been fixed. No results from this research are reported here.

3.3.5 Filter Frequency Range

Because of the limit on the use of bits, the system design will still have to be optimized. In order to keep a certain frequency from resonating in more than one filter, filters will have to be optimized. If memory does not permit the system to increase the order of the filter, a sacrifice can be made by decreasing the range of frequencies around the center frequency of each filter. This will reduce the magnitude of the intersection between neighboring filters, which in turn produces a decrease of overlap between neighboring filters.

3.4 SUB-SYSTEM DESIGN AND IMPLEMENTATION

The original goal was to have a working model of a 16-channel bandpass filter model of a digital cochlea. The goal was hindered by the discovering of many errors in designs, both in the prior research methods and current research methods. Errors were found in the clocking design, in adder blocks of the filter design, in binary point arithmetic calculations, and in generating VHDL code with the use of a multiplexer. The filter design is complicated by the need to use the correct number of bits to arrive at a level of precision able to accurately filter the desired range of frequencies while maintaining the memory capacity of the FPGA. The filters with lower frequencies need more bits because without high accuracy in the coefficients, primary the numerator coefficients, the coefficients would seem to be the same. An example of the difference between coefficients is described in Table 4. If two coefficients, 0.009722345450232430 and 0.009722345450233991, both use insufficient number of bits to retain the precision, both numbers could represented as 0.009722345450230000 and there would be no difference seen. Otherwise if two coefficients, 0.0574428650970612 and 0.071258915652282401 use the same number of bits for precision the numbers could be seen as 0.057442865097000 and 0.071258915652280000; and the differentiation still can be made between the two coefficients.

Center freq.	Bandpass range	Numerator 1 (b1) coefficient
250	225 – 275	0.00972234545023**2430**
350	325 – 375	0.00972234545023**3991**
2500	2345 – 2655	0.0574428650970612
2900	2705 – 3095	0.071258915652282401

Table 4, Coefficient precision comparison (sample frequency of 16 kHz).

The second and third filter coefficients are very similar and it is crucial to use enough bits to differentiate the two. The similarity between the first and second filter coefficients is even greater. The numerator 1 (b1) and numerator 3 (b3) are equal except one is negative and the other positive. Through trial and error, b3 was shown to have the greatest need for bits than b1. Coefficient b1 needs more bits than the denominator coefficients. The denominator coefficients seem to follow the same trend, with a3 needing more bits than a2, and so forth. See Section 3.3.1 to visualize the numerator and denominator coefficients.

3.4.1 Audio Codec And Clocking

The neural network was designed for a sampling frequency of 16000 Hertz. The system has to be designed using the same sampling frequency if the signal processing is to be done accurately. The filter coefficients were generated with the 16 kHz sampling frequency. The undivided XSV300 board's clock runs at 100 MHz. The XSV300 has a clock divider that allows the clock to be divided by 2, 3, 4, etc. A clock frequency of 33.3 MHz was selected.

The Stereo codec on this board requires three control clocks: LRCK, SCLK, and MCLK.

LRCK is the signal that selects the left (0) or right (1) channel. This signal should be the same as the sampling frequency. Defined in the codec datasheet (Appendix ###) for the codec, the minimum frequency allowed for the LRCK is 16 kHz.

SCLK is used to synchronize the serial bit stream. There must be 32 cycles for each of the two channels in the sample, which means the SCLK must be set to 64 times the sampling frequency.

MCLK is the master clock, which is used to synchronize the internal operations of the codec. This board requires that the MCLK be set to 256 times the sampling frequency.

The following are the settings desired:

Divisor	3
Clock frequency provided	33.33 MHz
MCLK (CLK divide by 8)	4.167 MHz
SCLK (CLK divide by 32)	1.042 MHz
LCLK (CLK divide by 2048)	16.28 KHz

A clock division macro was developed using T-flipflops in sequence. This clock divider was designed by Lee and Lee and is shown in Appendix C. Their codec design is shown in Appendix D. [5]

The codec specifies that the rising edge of SCLK should not occur at the same time as the LRCK clock edge. This required a slight modification in the design of the clock divider to invert the SCLK signal.

When the bitstream of filters was downloaded into the FPGA the data shown on the oscilloscope was not stable and did not correspond to the discrete output. The clocking devised by Lee and Lee was investigated and found to be done incorrectly. The LCLK that had match the sampling frequency of 16 kHz was producing a signal of 8 kHz. The clocking structure created by Lee and Lee is shown in Figure 11. The corrected clock with LCRK at 16 kHz is shown in Figure 12.

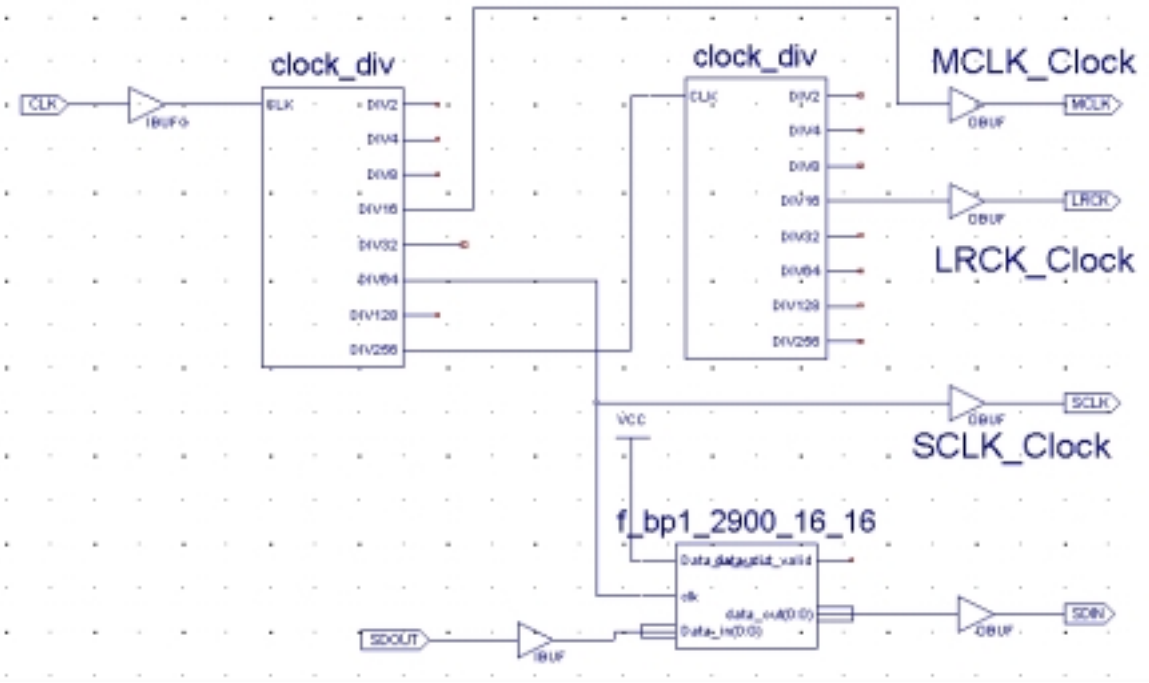


Figure 11: Clock created by Lee and Lee.

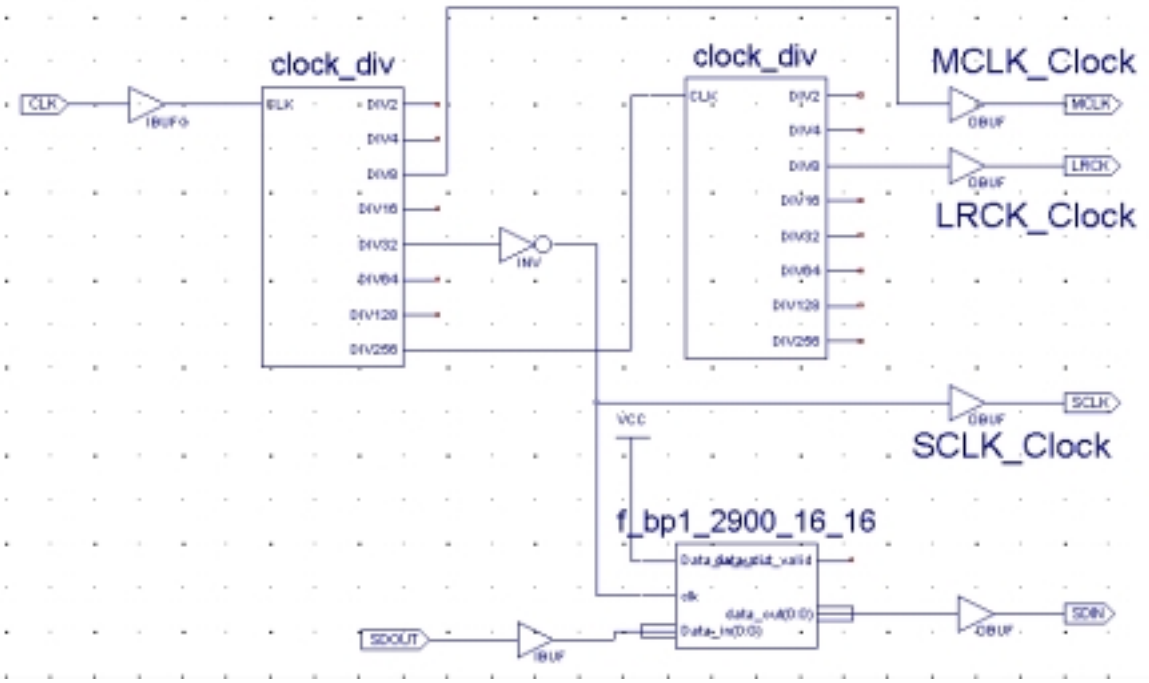


Figure 12: Corrected clocking structure with LCLK at 16 kHz.

3.4.2 Shift Register Operations

Only two operations involved the shift register: a serial-to-parallel conversion and a parallel-to-series conversion. (These were designed by Chen, Gaw, and Raskob [4].) The serial-to-parallel conversion takes the single bitstream and outputs a slice of bits to the filter. The macro designed is a 32-bit register made of 32 D-flipflops in sequence. While the macro outputs a 32-bit bus, only the selected slice of bits is passed to the filter and the remaining bits are simply dropped. The slice of bits was first done with 8 bits and then with 16 bits. The 16-bit slice proved to be a more desirable option. The difference between an 8-bit slice and a 16-bit slice will be shown in Section 3.4.5.

3.4.3 System Errors And Their Solutions

There were many errors in simulating models using the shift registers. Lee and Lee simulated results mainly with an 8-bit slice with an output of 8 bits in the filter that did not give errors. They used 8 bits across the system in order to keep the number of bits minimal. When more bits had to be added in the system, there was no documentation of the resulting errors that resulted. Matching the number of bits that were sliced to the last output bits of the subsystem solved the errors. Figure 6 and Figure 7 in Section 3.3.3 show a model of a subsystem. In this subsystem the last AddSub would have to output the same number of bits as the slice. Also, the last AddSub needed to carry a sample period of –1, which means it inherits the first known input period.

Another error (ERROR: NgdBuild:604) was encountered when generating the VHDL file. When generating more than one of the 16 outputs, a multiplexer is needed. This multiplexer gave a puzzling error message that was stored in a file called coregen.log. This file gave an error in the cores of different blocks, usually multipliers. This error was solved by generating with the option of “Everywhere Available” under the Xilinx Core Generator. This problem was not encountered again after that change was made.

3.4.4 Binary Point Arithmetic

The accuracy of the filter is determined by three main options. The first is the number of bits that will be selected to slice the data that goes into the system. The second is the precision of the bits selected for certain blocks. Blocks that allow the user to select the number of bits used to determine the precision are AddSub and multipliers. The last option is the selection of the right binary point position.

In a design, delay and adder blocks are usually simple, but multipliers may be cause for concern. If the option of “full precision” is not selected then the user must select the number of bits of the coefficient and its binary point position. Xilinx defines full precision in the block as having sufficient precision to represent the result without error. When selecting the binary point, the user is selecting how many bits are to the right of the binary point (the size of the fraction). The binary point position must be between zero and the number of bits that the user selected for the coefficient. If the correct binary

point position is not selected then the coefficient could be read as zero or some number not even close to the actual value.

When designing the multipliers, the block is labeled with the value of the coefficient. If the binary point is not calculated correctly, the value of the coefficient will not be a good representation of the value on the block.

3.4.5 Data Input Bits

Data input bits or slice, refers to the system containing 32 bits and the amount of bits that is used or sliced from the most significant bits in a number. This slicing is important because memory is saved with the use of a lesser bit slice.

4. RESULTS AND CONCLUSION

A system of 16 digital bandpass filters was successfully designed. The entire design was constructed with 18 data input bits (slice). The proposed plan was to use a 12-bit slice to maintain a satisfactory level of memory usage. After numerous tests, the use of a 16-bit slice simulated satisfactory results. The entire design was constructed to be implemented with an 18-bit slice, which showed more accurate results than the 16-bit slice. Filters 8 – 16 are second-order bandpass filters. Filters 1 – 7 would not produce satisfactory results using second-order bandpass filters because of the need for a high number of bits due to the high precision in the coefficients. Also, in the lower frequency filters, the coefficients are much more similar than the higher frequencies, and the need for a higher precision is necessary. The use of first-order filters was designed instead. While, first-order was satisfactory, there was a significant amount of overlap in neighboring filters. Second-order lowpass filters were added before the first-order bandpass filters that minimized the overlap to satisfactory levels. The final diagram is shown in Figure 13 and its simulation in Figure 14.

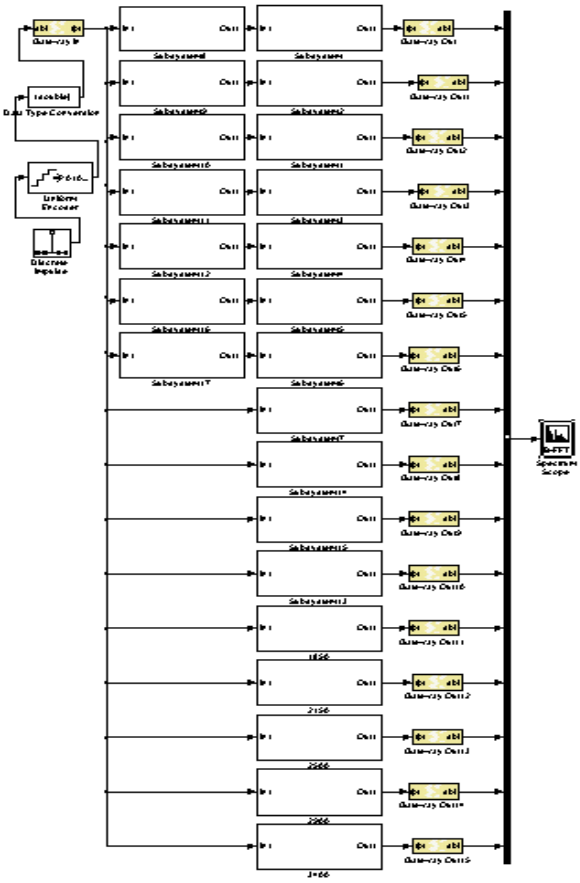


Figure 13: Final diagram of filter design.

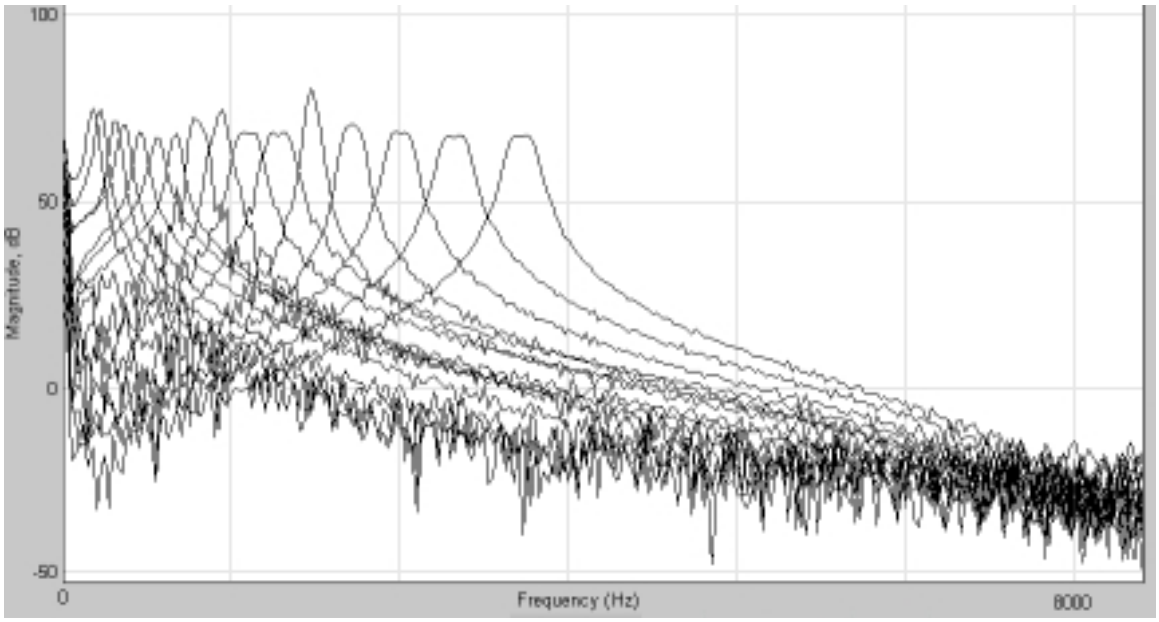


Figure 14: Final simulation of filter design.

The final design would need four of the proposed XCV300 chips to implement the entirety of the design. Table 5 shows the memory usage per chip of each of the 16 filter banks. Filters 8 – 16 were implemented individually but filters 1 – 7 were implemented together with the lowpass and bandpass filters together and the memory usage was accounted for together.

Filter bank	Total memory usage per chip (%)
1	24
2	24
3	26
4	26
5	26
6	27
7	27
8	19
9	19
10	20
11	20
12	19
13	20
14	19
15	19
16	19

Table 5, Total memory usage per chip for each filter bank.

This design can be adjusted by manipulating the bits in order to use only three XCV300 chips but the performance will decrease. With the use of four XCV300 chips there is some memory that is not being used. The design can be optimized to fit the entirety of the four chips. Another option for this design is to use a chip with a larger memory capacity. The XCV1000 increases the system gates from the XCV300's 322970 system gates to 1124022 that would be enough to implement all 16-filter banks on a single FPGA chip.

5. ACKNOWLEDGMENTS

I would like to thank the National Science Foundation (NSF) for their sponsorship of the 2002 Summer Undergraduate Fellowships In Sensor Technologies (SUNFEST) Research Program through the NSF-REU grant. Special thanks to Dr. Jan Van der Spiegel, who provided his time, resources, and facilities, and Dr. Paul Mueller who also provided his time and resources. Without the support of Dr. Van der Spiegel and Dr. Mueller, the progression of my research would not have been possible. Ms. Lois Clearfield, Dr. Dwight Jagard, Dr. Nathan Ensmenger, Dr. Jorge Santiago-Aviles, Mr. Sid Deliwala, and Ms. Janice Fisher were responsible for the program's organization, support, and educational instruction of the program. Thanks also to Fran Olivieri and the Xilinx support staff for their help in the research.

6. REFERENCES:

1. 21st Century Eloquence. *Industry Trends*. http://voicerecognition.com/2000/trends/
2. Quatieri, Thomas. *Discrete-Time Speech Signal Processing, Principals and Practice*. Upper Saddle River, NJ: Prentice-Hall, 2002.
3. Ali, Ahmed. *Auditory-Based Acoustic-Phonetic Signal Processing for Robust Continuous Speech Recognition*. Ph.D. Dissertation, Department of Electrical Engineering, University of Pennsylvania. December 1999.
4. Chen, Miranda and Gaw, Samantha and Raskob, Benjamin. *Auditory Based Bandpass Filters Implemented on a single FPGA*. Senior Project, Department of Electrical Engineering, University of Pennsylvania. April 2002.
5. Lee, Leonard and Lee, Tjenchew. *FPGA Implementation of Front-End Auditory Speech Processing*. Senior Project, Department of Electrical Engineering, University of Pennsylvania. May 2001.
6. Hinck, Todd. *"Experiments with Digital Cochleas" in Workshop on Neuromorphic Engineering*. http://www.isr.umd.edu/~djklein/tride99/cochlea.html
7. Watts, Loyd. *"Reverse Engineering the Brain" in Seminar on Computer Systems.* http://murl.microsoft.com/LectureDetails.asp?640 May 2000.
8. The XESS Corporation. loop-simple.zip. http://www.xess.com/
9. Oppenhein, Alan and Schafer, Ronald. *Discrete-Time Signal Processing.* Upper Saddle River, NJ: Prentice-Hall, 1999.

7. APPENDICES A - E

APPENDIX A

FACILITIES/EQUIPMENT

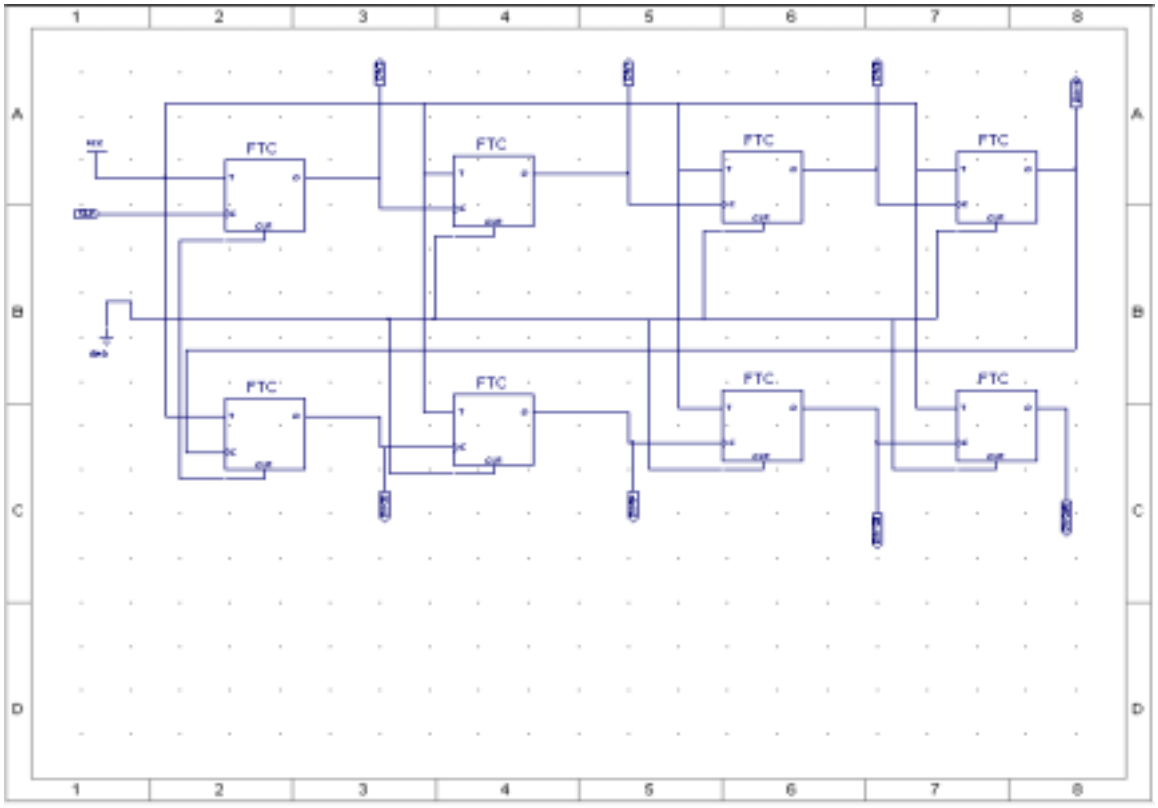
XSV300
XCV300
XESS XSTools
Matlab Release 12 with Simulink's DSP Blockset license
Xilinx with System Generator (ISE 4)
ATX power supply (needed to power board and chip)
Oscilloscope
Function Generator
Cables (various)

APPENDIX B

NOMENCLATURE

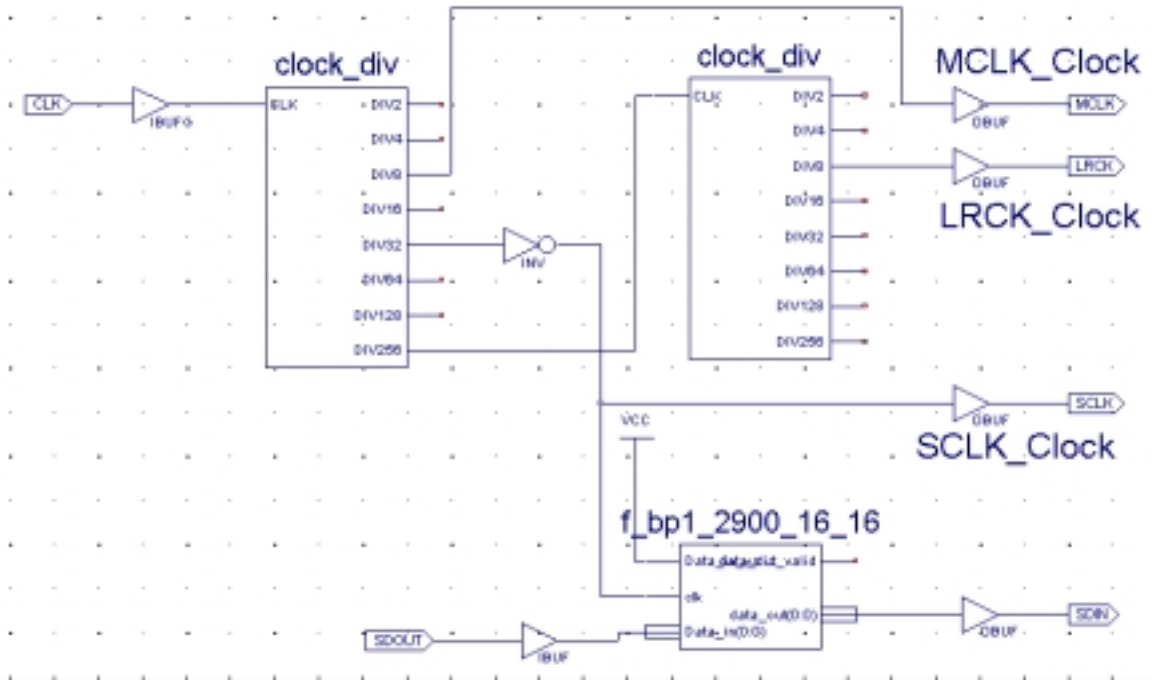
A/D	Analog to Digital
D/A	Digital to Analog
DSP	Digital Signal Processing
FFT	Fast Fourier transform
FIR	Finite Impulse Response
FPGA	Field programmable gate array
I/O	Input/Output
IIR	Infinite Impulse Response
ISE	Integrated Service Environment
VHDL	Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

APPENDIX C



The clock divider designed by Lee and Lee.

APPENDIX D

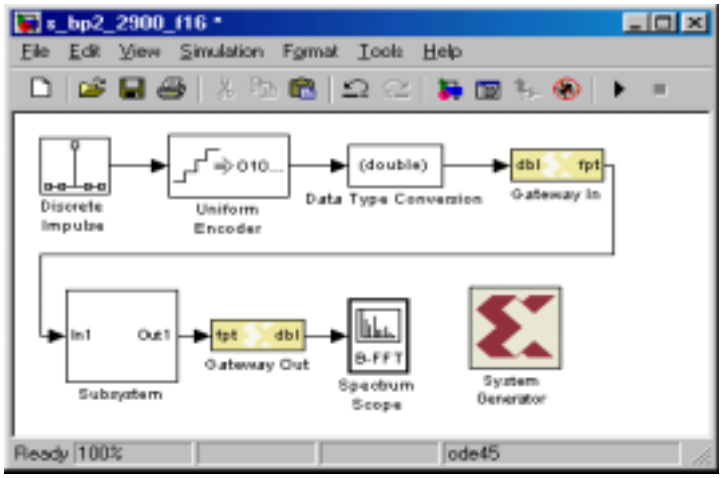


The corrected codec design used in the final design.

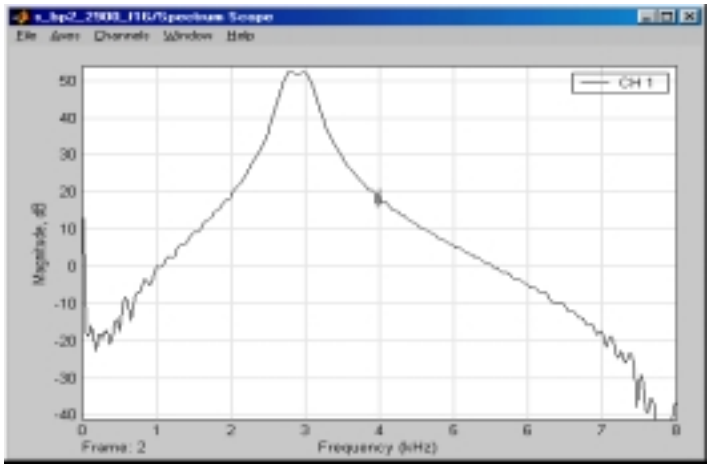
APPENDIX E

This appendices will define a few graphs that would give a better understanding of a comparison of what is expected from the results due to the parameters* selected for this project. All graphs are simulated with Matlab's Spectrum Scope.

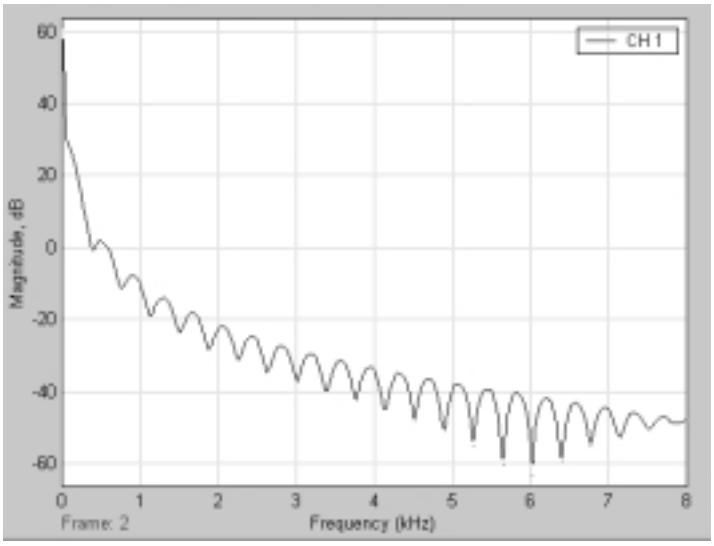
*All parameters are not shown or discussed in these appendices. Only the major factors are discussed for a better understanding of the project.



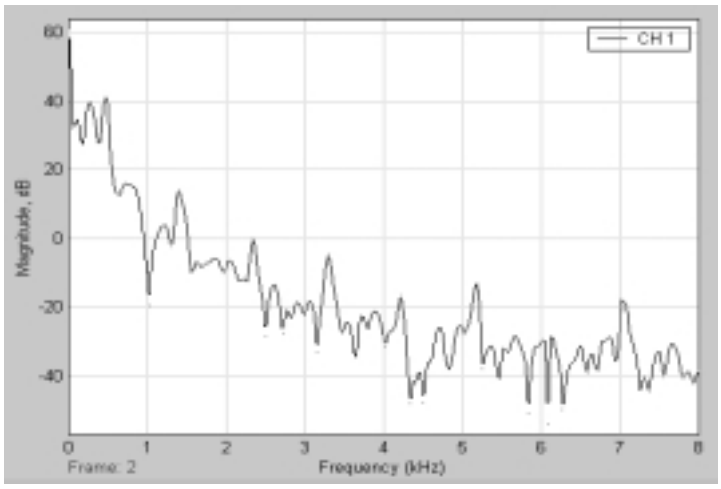
This window shows how a single filter (subsystem) is setup for viewing with a spectrum scope.



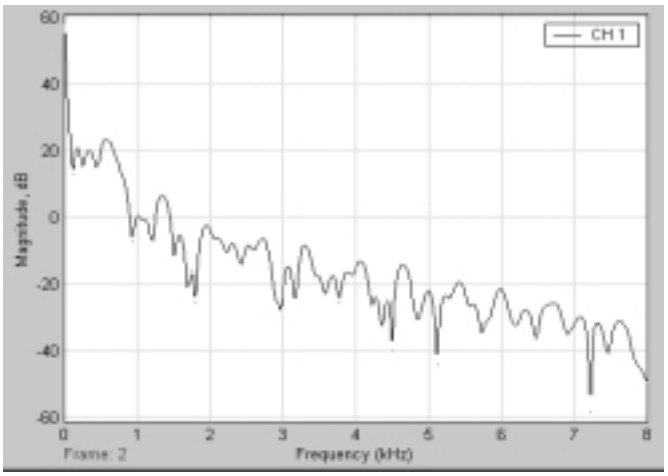
This is the output of the previous window. The subsystem is a second-order bandpass filter with center frequency of 2900 Hz. Full precision was used in the coefficient bits and a 16-bit slice for the data input bits.



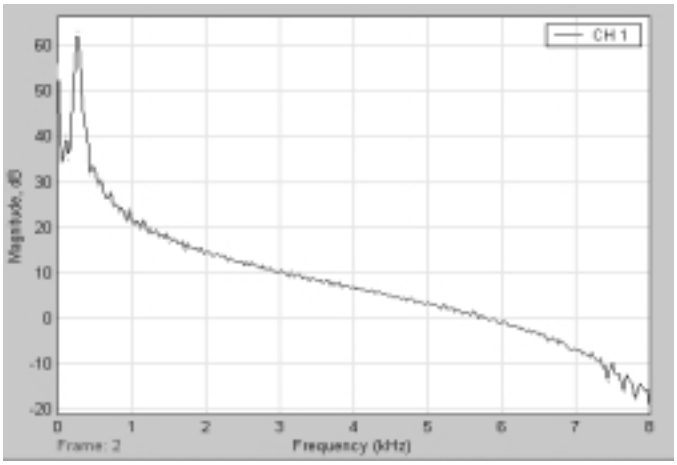
Type:	bandpass
Order:	1st
Center Frequency (Hz):	250
Coefficient bits:	full
Data input bits:	8



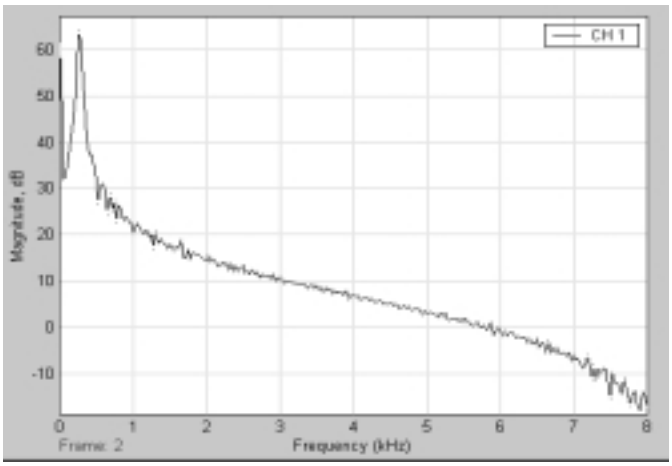
Type:	bandpass
Order:	1st
Center Frequency (Hz):	250
Coefficient bits:	16
Data input bits:	8



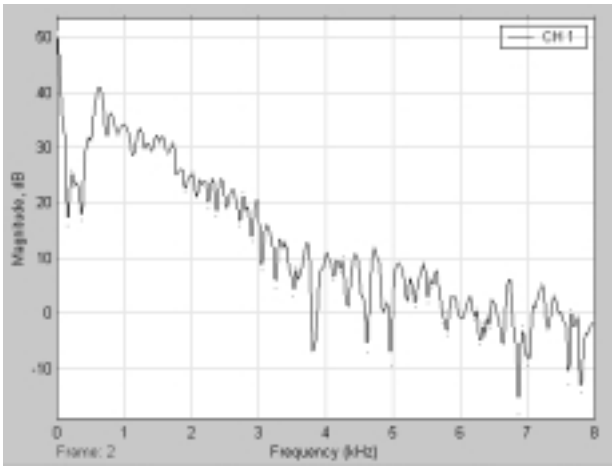
Type:	bandpass
Order:	1st
Center Frequency (Hz):	250
Coefficient bits:	16
Data input bits:	8



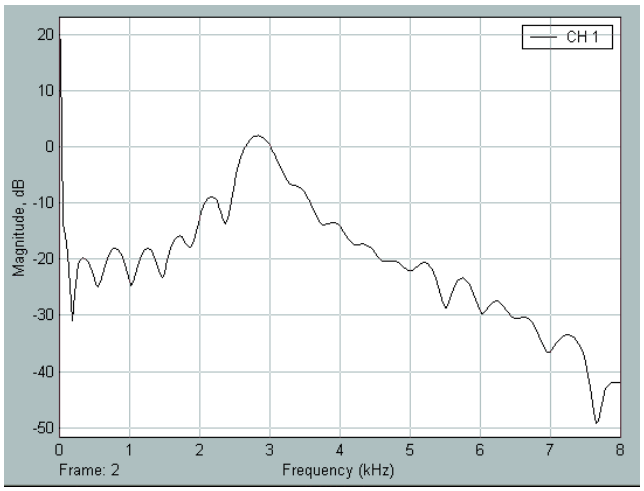
Type:	bandpass
Order:	1st
Center Frequency (Hz):	250
Coefficient bits:	full
Data input bits:	16



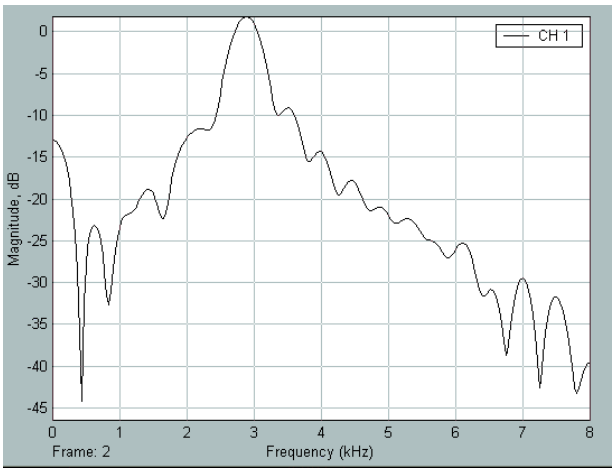
Type:	bandpass
Order:	1st
Center Frequency (Hz):	250
Coefficient bits:	16
Data input bits:	16



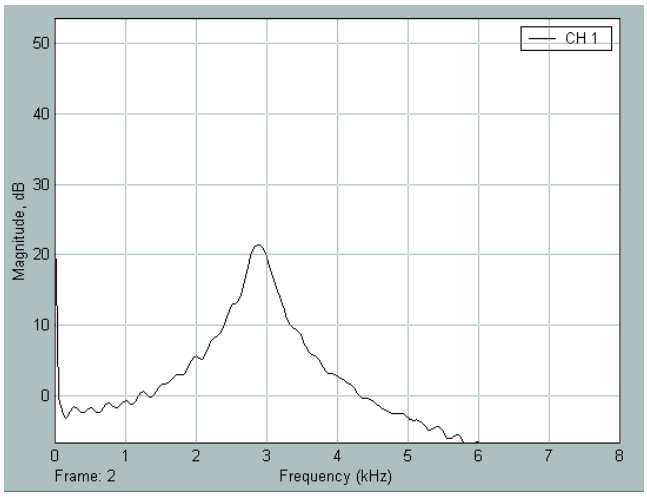
Type:	bandpass
Order:	1st
Center Frequency (Hz):	250
Coefficient bits:	8
Data input bits:	16



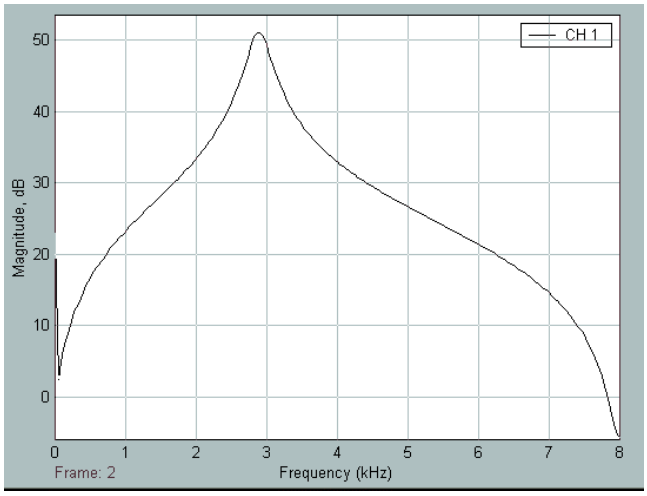
Type:	bandpass
Order:	1st
Center Frequency (Hz):	2900
Coefficient bits:	16
Data input bits:	8



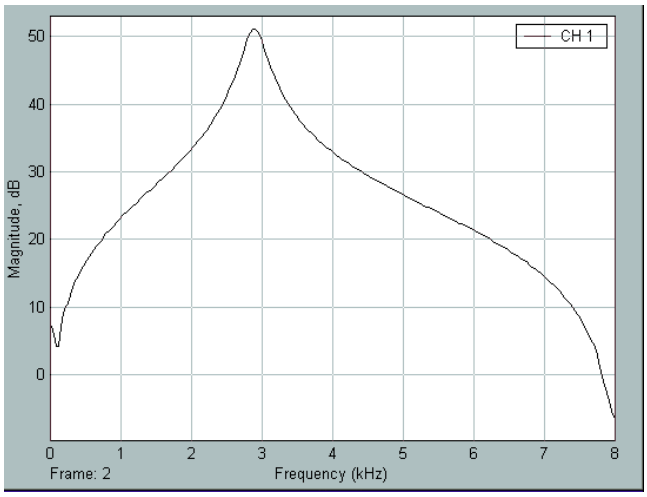
Type:	bandpass
Order:	1st
Center Frequency (Hz):	2900
Coefficient bits:	full
Data input bits:	8



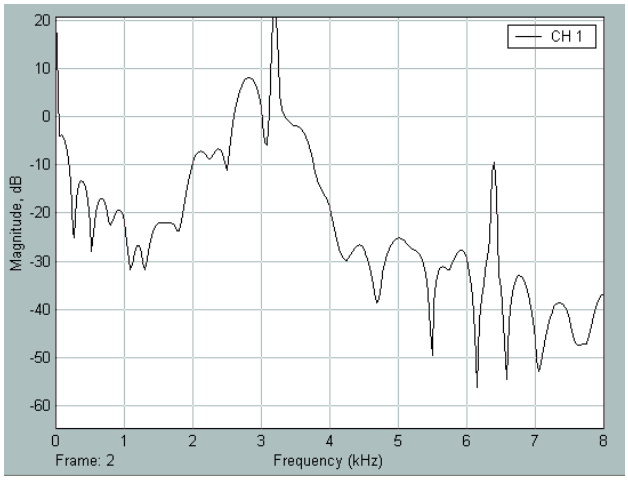
Type:	bandpass
Order:	1st
Center Frequency (Hz):	2900
Coefficient bits:	8
Data input bits:	16



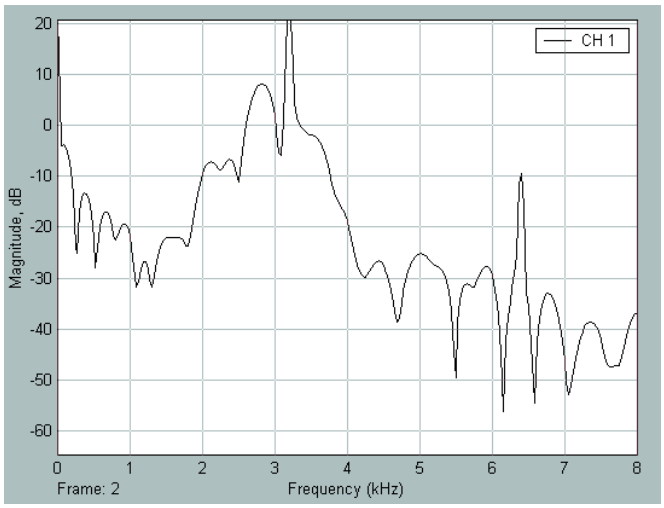
Type:	bandpass
Order:	1st
Center Frequency (Hz):	2900
Coefficient bits:	16
Data input bits:	16



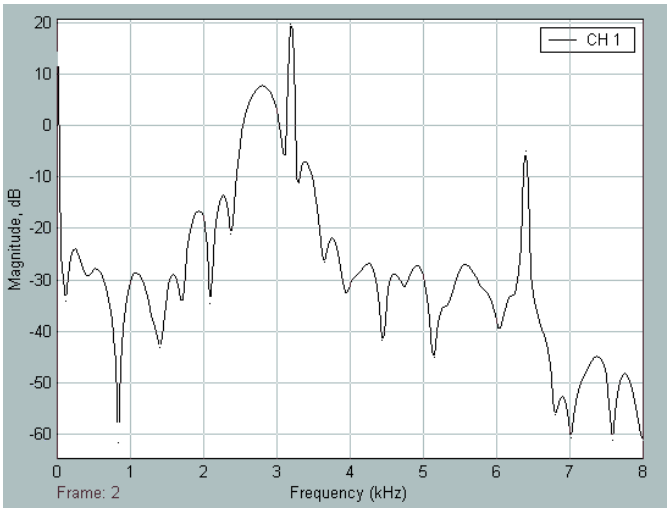
Type:	bandpass
Order:	1st
Center Frequency (Hz):	2900
Coefficient bits:	full
Data input bits:	16



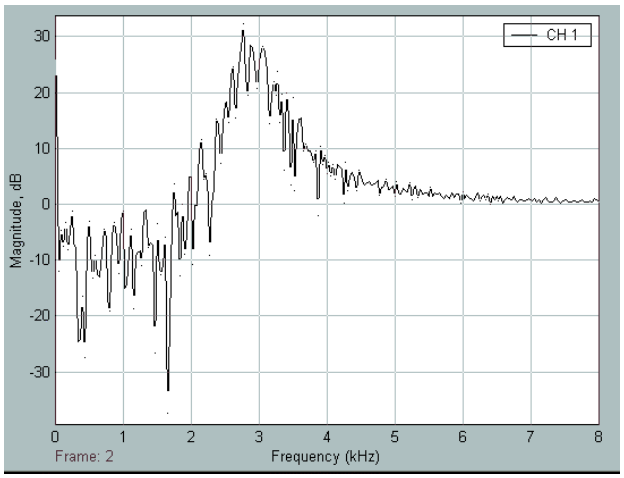
Type:	bandpass
Order:	2nd
Center Frequency (Hz):	2900
Coefficient bits:	8
Data input bits:	8



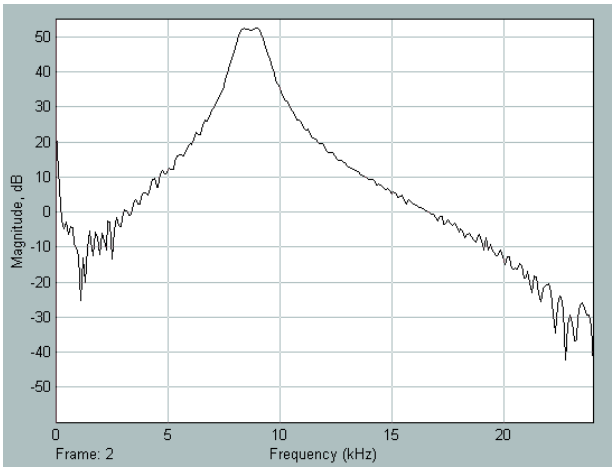
Type:	bandpass
Order:	2nd
Center Frequency (Hz):	2900
Coefficient bits:	16
Data input bits:	8



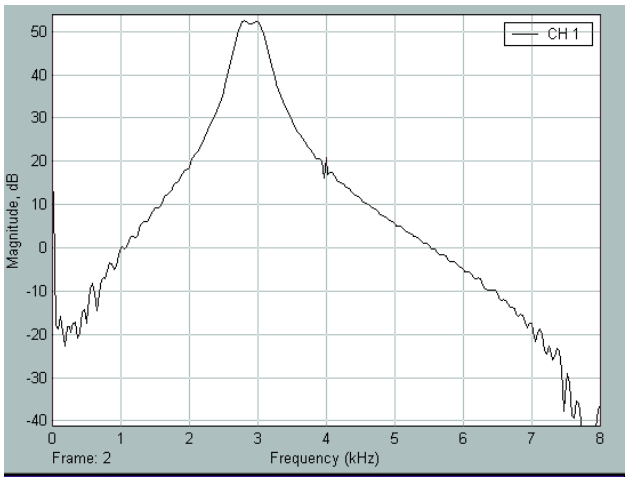
Type:	bandpass
Order:	2nd
Center Frequency (Hz):	2900
Coefficient bits:	full
Data input bits:	8



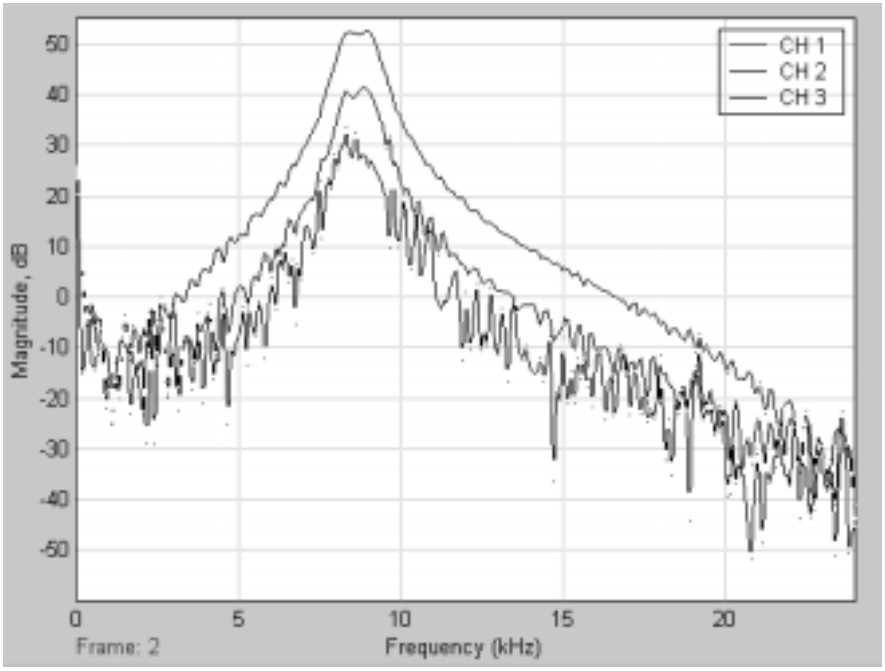
Type:	bandpass
Order:	2nd
Center Frequency (Hz):	2900
Coefficient bits:	8
Data input bits:	16



Type:	bandpass
Order:	2nd
Center Frequency (Hz):	2900
Coefficient bits:	16
Data input bits:	16



Type:	bandpass
Order:	2nd
Center Frequency (Hz):	2900
Coefficient bits:	full
Data input bits:	16



Type:	bandpass
Order:	2nd
Center Frequency (Hz):	2900

This is a comparison of:

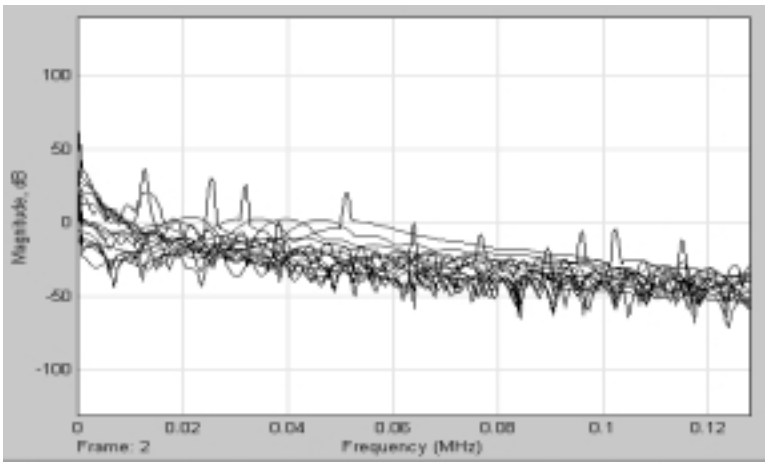
Coefficient bits:	12
Data input bits:	12
(lowest magnitude)	

versus

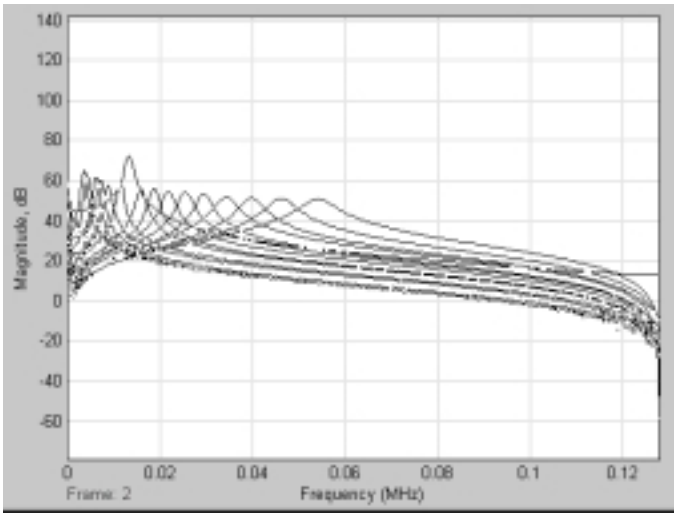
Coefficient bits:	14
Data input bits:	14
(middle one)	

versus

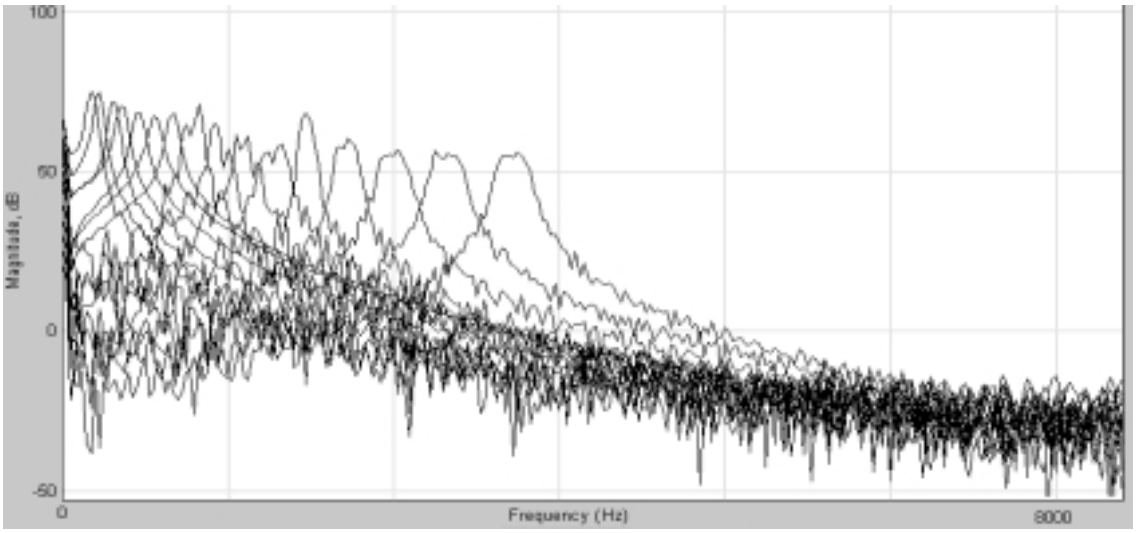
Coefficient bits:	16
Data input bits:	16
(highest magnitude)	



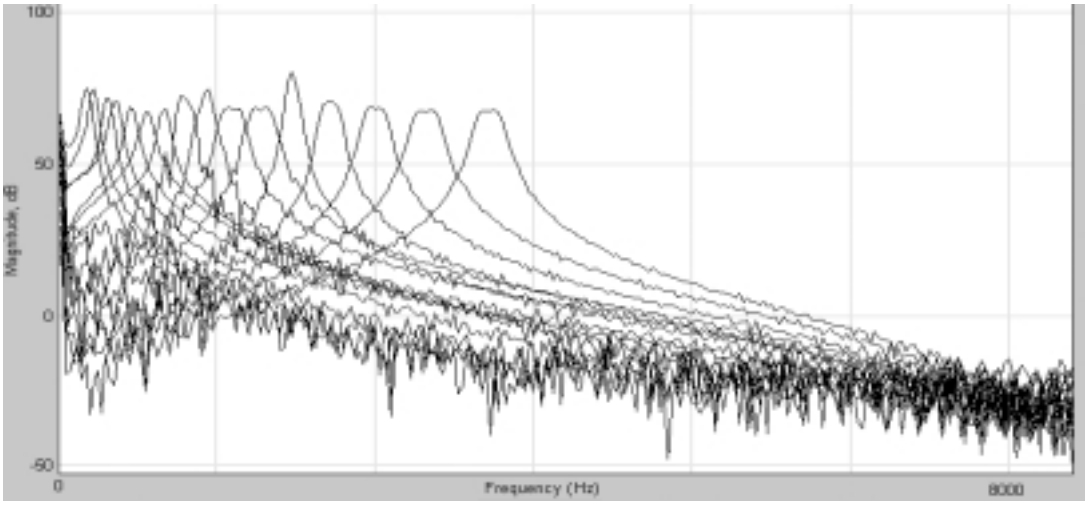
Type:	bandpass
Center Frequency (Hz):	all
Coefficient bits:	full
Data input bits:	8



Type:	bandpass
Center Frequency (Hz):	all
Coefficient bits:	full
Data input bits:	16



Type:	bandpass
Filters:	1 – 7
Coefficient bits:	18
Data input bits:	18
Filters:	8 – 16
Coefficient bits:	16
Data input bits:	16



Type:	bandpass
Filters:	1 – 16
Coefficient bits:	18
Data input bits:	18