DIGITAL FILTER DESIGN AND SYNTHESIS USING HIGH-LEVEL MODELING TOOLS

by

Brian A. Jackson

Thesis submitted to the Faculty of
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Masters of Science
in
Electrical Engineering

Dr. James R. Armstrong, Thesis Advisor
Dr. F. Gail Gray
Dr. Dong S. Ha

Wednesday, December 1, 1999
Blacksburg, Virginia

Keywords: VHDL, COSSAP, MATLAB, digital filters

DIGITAL FILTER DESIGN AND SYNTHESIS USING HIGH-LEVEL MODELING TOOLS

by

Brian A. Jackson

Dr. James R. Armstrong, Thesis Advisor
Virginia Polytechnic Institute and State University
The Bradley Department of Electrical and Computer Engineering

ABSTRACT

The purpose of this thesis is to formulate a technically sound approach to designing Infinite Impulse Response (IIR) digital filters using high-level modeling tools. High-level modeling tools provide the ability to build and simulate ideal models. Once proper validation is complete on these ideal models, the user can then migrate to lower levels of abstraction until an actual real world model is designed. High-level modeling tools are the epitome of the top-down design concept in which design first takes place with the basic functional knowledge of a system. With each level of abstraction, validation is performed. High-level modeling tools are used throughout industry and their application is continually growing especially in the DSP area where many modes of communications are expanding. High-level modeling tools and validation significantly address this complex expansion by utilizing an ideal representation of a complicated network.

TABLE OF CONTENTS

CHAPTER 1: Introduction

The purpose of this thesis research is to formulate a technically sound approach to designing Infinite Impulse Response (IIR) digital filters using high-level modeling tools. The basic functional need for filtering is to pass a range of frequencies while rejecting others. This need for filtering has many technical uses in the digital signal processing (DSP) areas of data communications, imaging, digital video, and voice communications. Digital signal processing techniques are being used to handle these demanding challenges in digital communications system design.

Analog filters are continuous-time systems for which both the input and output are continuous-time signals. Digital filters are discrete-time systems whose input and output are discrete-time signals. Digital filters are implemented using electronic digital circuits that perform the operations of delay, multiplication, and addition. Analog filters are implemented using resistors, inductors, capacitors, and, possibly, amplifiers [Chirlian]. The values of these analog components can drift over time and their precision is limited. In addition, especially when filtering takes place at low frequencies, inductors are often large and heavy. The multiplier coefficients of digital filters are established by the circuitry and do not drift. The precision of the multiplier values can be made as large as desired by increasing the complexity of the circuitry. Digital filters can be implemented using integrated circuits so that the per unit cost of digital filter construction is less than a comparable analog filter [Chirlian]. Tolerances and accuracy considerations are important factors for both analog and digital signal processing. Digital signal processing provides better control of accuracy requirements. Wide tolerances in analog filters make it extremely difficult for a system designer to control the accuracy of an analog signal processing system. A system designer has much better control of accuracy of digital systems in terms of word length, floating-point versus fixed-point arithmetic, and other similar factors [Manolakis]. These are the major advantages of digital filters.

High-level modeling tools provide the ability to build and simulate ideal models. Once proper validation is complete on these ideal models, the user can then migrate to lower levels of abstraction until an actual real world model is designed. High-level modeling tools are the epitome of the top-down design concept in which design first takes place with the basic functional knowledge of a system. With each level of abstraction, validation is performed. High-level modeling tools are just beginning to be used throughout industry and their application is continually growing especially in the DSP area where many modes of communications are expanding. High-level modeling tools and validation significantly address this complex expansion by introducing an ideal representation of a complicated network.

A high-productivity environment is needed to support development from system definition and algorithm development to implementation and verification. An important key element deals with high-level modeling and analysis tool sets. Two prominent DSP tool sets of this nature are COSSAP and SPW. These tools allow fast, natural expression of single clock, multi-rate, and asynchronous systems; provide extremely fast high-level simulations; and provide full support for hardware and software implementation and verification at any abstract level. Another important DSP tool used for computation and visualization is the software package called MATLAB. This DSP tool is a prominent problem-solving application used in both universities and industry.

Another important key element in the aforementioned high-productivity environment deals with the already-established, industrial move to use hardware description languages to document, simulate, and synthesize an electronic system. The two prominent hardware description languages (HDLs) are VHDL and Verilog. Both languages have the necessary constructs to support the modeling, simulation, and synthesis of complicated digital systems. In addition, the benefits of making design specifications more technology-independent, automating low-level details, and improving design quality make VHDL and Verilog important tools for design.

Interfacing the two aforementioned key elements (HDLs and high-level tools) has significant technical rewards in industry. Specifically, two key elements that have established, feasible interface-capabilities are COSSAP and VHDL. Both these tools are heavily used in both universities and industry. COSSAP and VHDL are linked in that COSSAP contains an extensive DSP library written in VHDL. For high-level

design, VHDL makes verification at any abstract level significantly easier especially when it comes to synthesis.

Accuracy of results when comparing an ideal digital filter to an n-bit quantized digital filter is a major concern of validation. For quantized digital filters, binary multiplication is the main source of round-off errors. Therefore, quantized digital filters can never achieve 100% ideal accuracy due to fixed, binary word sizes. Effective round-off procedures are needed to produce accuracy results that are close to the 100% accuracy rating of an ideal digital filter

Broad background knowledge of VHDL and synthesis, DSP digital filter techniques, the high-level tool COSSAP, the software package MATLAB, and a UNIX workstation environment are paramount for a successful realization of a digital filter design. Understanding of Synopsys synthesis tools is also equally important. Synthesis tools provide the ability to map VHDL code to technology libraries at the structural gate level. Suitable training time on the part of the designer must be attained to ensure an efficient and optimized digital filter design.

This thesis will seek to address all of the aforementioned issues related to interfacing the high-level modeling tool COSSAP and VHDL. The following chapters will attempt to provide a broad understanding and methodology to IIR fixed-point digital filter design. Chapter 2 discusses quantization formats and quantization noise. This chapter also discusses finite-precision effects that are inherent in designing quantized digital filters. Chapter 3 discusses the different digital filter structures associated with Infinite Impulse Response (IIR) filter design. Special attention is paid to hardware considerations in terms of binary adders and multipliers. Chapter 4 describes the different types of IIR digital filter design. Characteristics of each filter design are discussed in terms of magnitude frequency response and other frequency-related attributes. Chapter 5 addresses the saturation modes associated with the n-bit adder sub-blocks when using the high-level modeling tool COSSAP. This chapter also discusses the reasoning behind choosing the best saturation mode for the adder sub-blocks to be used for all future quantized digital filter designs. Chapter 6 addresses the round-off modes associated with the n-bit multiplier sub-blocks when using the high-level modeling tool COSSAP. Chapter 7 investigates ways to compensate for the aforementioned tool deficiencies that are inherent in COSSAP when the tool produces VHDL-generated code. Chapter 8 outlines the flowchart of the digital filter design procedures utilizing COSSAP and VHDL. This extensive chapter is comprehensive and detailed to ensure a step-by-step approach to designing an optimum digital filter. Chapter 9 shows results of the outlined design methodology, in Chapter 8, for the various IIR digital filter structure types. Chapter 10 shows results of the design methodology for designing digital filters in the DSP areas of data communications, imaging, digital video, and voice communications. Nominal numbers for bandwidths are used. The same cut-off frequency and stopband frequency points used in Chapter 9 are also used in this chapter. Chapter 11 summarizes this research with remarks and analytical conclusions. Future work is also discussed in regards to upgrading the design process with other features.

CHAPTER 2: Quantization

2.1: Quantization

Quantization is the process in which a binary number with a finite number of bits represents a real number. In the case of this research, the numbers are multiplier coefficients and I/O signals of a specified digital filter. Because of quantization, any arbitrarily specified multiplier coefficient will not be realized 100% accurately [Chirlian]. This is due to finite-precision effects. In digital filters, arithmetic operations are performed with finite precision due to the use of fixed-size memory words or registers. Finite precision dictates that multiplier coefficients exceeding their bit length limit must be truncated or rounded within the number of significant bits allowed. An illustrative example would be trying to realize a multiplier coefficient using only two bits to represent the magnitude. The only possible binary numbers would be as follows:

$$0.00_2 = 0.00_{10}$$
$$0.01_2 = 0.25_{10}$$
$$0.10_2 = 0.50_{10}$$
$$0.11_2 = 0.75_{10}$$

Thus, many decimal numbers lying in the range 0 to 1 cannot be represented 100% accurately. The equation used to represent the quantization process is as follows:

$$x_q(n) = Q[x(n)] \qquad \text{(Eqn. 2.1)}$$

The function $Q[*]$ represents a b-bit quantizer, the variable $x(n)$ represents the n^{th} ideal sample of infinite precision, and the variable $x_q(n)$ represents the b-bit quantized result of the ideal sample $x(n)$.

2.2: Quantization Noise

Because all decimal numbers within a specified range cannot be represented 100% accurately, inherent errors occur. Even though increasing a word size directly increases the accuracy of the actual decimal-to-bit representation, that error will always exist. In short, the error introduced in representing a decimal number by a set of discrete value levels is called quantization error or quantization noise. The resulting quantization noise is represented as a sequence $E_q(n)$ defined as the difference between the quantized value and the actual decimal value. The equation for this description is as follows:

$$E_q(n) = x_q(n) - x(n) \qquad \text{(Eqn. 2.2)}$$

2.3: Two's Complement Representation

Because of its unique representation of all numbers including zero, its wide use in a majority of computer systems including digital filters, and its ease of implementation, the two's complement notation is the binary representation of choice for this thesis research. Using this form of binary representation, the most significant bit (MSB) is the designated sign bit with binary zero denoting a positive number and binary one denoting a negative number. Using a four-bit number, a brief illustrative example is as follows:

$$0011_2 = 3_{10}$$
$$1101_2 = -3_{10}$$

2.4: Two's Complement Truncation

Using Equation 2.2 as reference, quantization error when truncating a two's complement number yields the equations as follows:

$$E_t(n) = x_t(n) - x(n) \qquad \text{(Eqn. 2.3)}$$
$$-2^{-b} <= E_t <= 0 \qquad \text{(Eqn. 2.4)}$$

The variables E_t, b, and $x_t(n)$ represent the truncated error, the number of bits expressing the fractional value, and the truncated b-bit quantized result of the ideal sample $x(n)$, respectively. For binary truncation of a b-bit quantity, the resulting quantized magnitude will be smaller than the ideal sample. Figure 2.1 provides a statistical viewpoint of a probability density function (pdf) for truncation error. The x-axis, *e*, is the error or noise. The y-axis, *p(e)*, is the pdf of the error or noise [Oppenheim2].

2.5: Two's Complement Rounding

Considering the quantization noise due to the rounding of a number, the resulting error is noticeably smaller than truncation. Again using Equation 2.2 as reference, quantization error when rounding a two's complement number yields the equations as follows:

$$E_r(n) = x_r(n) - x(n) \qquad \text{(Eqn. 2.5)}$$
$$-0.5*2^{-b} <= E_r <= 0.5*2^{-b} \qquad \text{(Eqn. 2.6)}$$

The variables E_r, b, and $x_r(n)$ represent the rounded error, the number of bits expressing the fractional value, and the rounded b-bit quantized result of the ideal sample $x(n)$, respectively. Binary rounding requires that for a b-bit quantization, the b+1th bit is needed to mathematically decide whether or not to add binary one to the preceding b-bits. If the b+1th bit is binary zero, then binary one will not be added to the preceding b-bits. If the b+1th bit is binary one, then binary one will be added to the preceding b-bits. As a consequence, the quantization error range of rounding is mathematically smaller than its truncation counterpart. Figure 2.1 provides a statistical viewpoint of a probability density function (pdf) for rounded error. Notice the error mean, m_e, is different in both quantization methods. On the other hand, error variance, σ_e^2, of both quantization methods are identical. It is stated here again that the term *error* and *noise* are used interchangeably throughout this thesis. The x-axis, *e*, is the error or noise. The y-axis, *p(e)*, is the pdf of the error or noise [Oppenheim2].

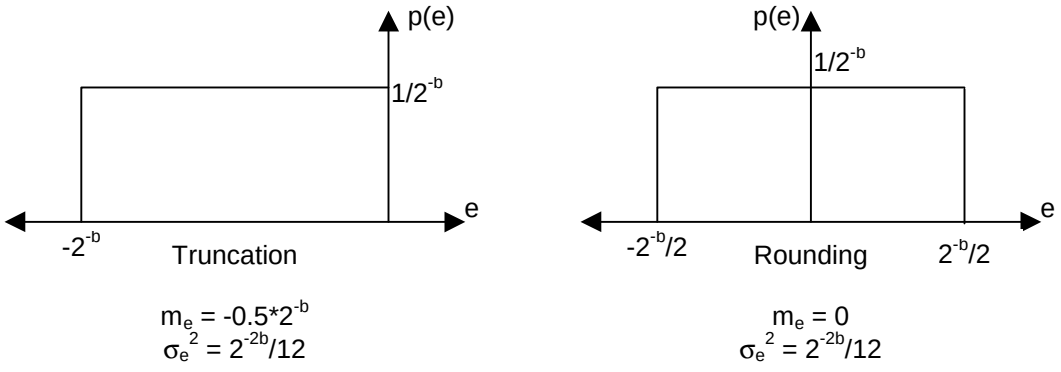


Fig. 2.1. Probability density function for quantization error.

2.6: Finite-Precision Effects

In DSP hardware, implementations are either in fixed-point or floating-point format. In fixed-point representation, a decimal number is represented as a string of digits with an implied decimal point. In this format, the digits to the left of the decimal point represent the integer part of the number, and the digits to the right of the decimal point represent the fractional part of the number. Fixed-point representation allows a user to cover a range of numbers x_{max} - x_{min} with a resolution as follows:

$$\text{ə} = (x_{max} - x_{min}) / (2^b - 1) \qquad \text{(Eqn. 2.7)}$$

In Equation 2.7, the term 2^b is the number of levels, b is the number of bits representing both the integer and fractional values, and ə is the resolution. The variables x_{max} and x_{min} can either be a positive or negative decimal value represented by b bits. Figure 2.1 is an illustrative example of a 3-bit two's complement implementation of a counting wheel. For this example, x_{max} and x_{min} are represented by the integers 3 and –4, respectively, and the number of levels is 8. The resolution, ə, is 1. To demonstrate the flexibility of Equation 2.7, Figure 2.2 is another example of a 3-bit two's complement implementation of a counting wheel with the exception that this is a fixed-point fractional example. For this example, x_{min} and x_{max} are represented by the decimals -1.00 and 0.75, respectively, and the number of levels is 8. The resolution, ə, is 0.25 for this case. A basic characteristic of fixed-point representation is that the resolution is fixed [Manolakis].

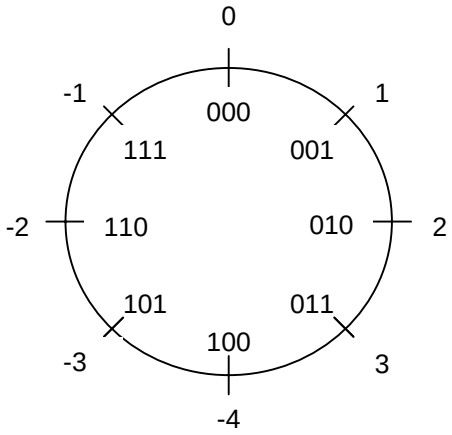


Fig. 2.1. Integer counting wheel for 3-bit two's complement numbers.

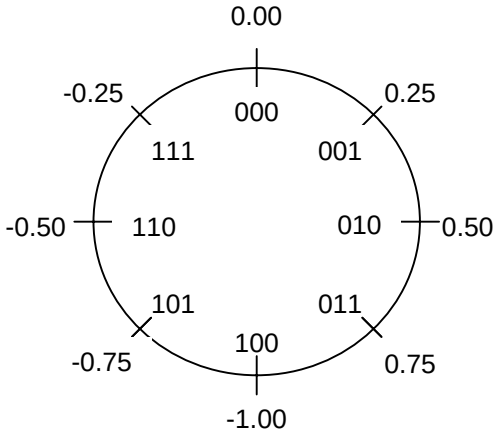


Fig. 2.2. Fractional counting wheel for 3-bit two's complement numbers.

Quantization stepsize is defined by the equation as follows:

$$q = 2^{-b} \qquad \text{(Eqn. 2.8)}$$

For this equation, the variables q and b represent the quantization stepsize and the number of bits expressing the fractional value, respectively. For Figure 2.1, the quantization stepsize is 1 because b is equal to zero. For Figure 2.2, the quantization stepsize is 0.25 because b is equal to 2.

In floating-point representation, in order to cover a larger dynamic range, the resolution varies across the range. For DSP hardware implementation, fixed-point representation requires less complicated circuitry and is more common in terms of use. Fixed-point representation will be investigated for this research.

2.7: Limit-Cycle Oscillations

Because of coefficient quantization and rounding (or truncation) of multiplier sub-block outputs, recursive systems like IIR digital filters experience nonlinear effects at the filter output in response to an input impulse. Ideally, a digital filter of infinite precision will exponentially decay to zero at the filter output when the input is an impulse. Because of the aforementioned quantization, the output of an n-bit quantized digital filter will either oscillate and be confined to a range of values or remain at a fixed value. In the case of the former, the range of values is called the dead band. An illustrative example of limit-cycle oscillations is the difference equation as follows:

$$y(n) = x(n) - 0.7y(n-1) \qquad \text{(Eqn. 2.9)}$$

Assuming an input impulse, x(0) = 15 and x(n) = 0 for all other values of n, Table 2-1 shows the filter response for an ideal and quantized output. The first column is the sample number, the second column is the ideal input impulse, the third column the ideal output, and the fourth column the quantized output rounded to the nearest integer. The ideal output continues to decay while the quantized output oscillates between –1 and 1. If the minus sign in Equation 2.9 were replaced with a plus sign, a similar result would occur with the exception that the quantized output would remain fixed at +1 and would not decay [Chirlian].

n	x(n)	y(n)	rounded y(n)
0	15	15	15
1	0	-10.5	-11
2	0	7.35	8
3	0	-5.145	-6
4	0	3.6015	4
5	0	-2.52105	-3
6	0	1.764735	2
7	0	-1.2353145	-1
8	0	0.86472015	1
9	0	-0.605304105	-1
10	0	0.4237128735	1

Table 2-1. Response to digital filter of Equation 2.9 for ideal and quantized outputs.

Limit-cycle oscillations are caused by quantization. This type of non-linear effect becomes more significant when designing filters of high order. The solution to this predicament is to decompose the high-order filter into 1st and 2nd order sub-blocks with the 2nd order sub-blocks having the higher design priority. Such a design format significantly reduces limit-cycle oscillations [Rabiner].

CHAPTER 3: IIR Digital Filter Structures

3.1: Linear Time-Invariant Systems

Digital filters are discrete-time systems. A discrete-time system is essentially an algorithm for converting an input sequence into an output sequence. The input signal *x(n)* is transformed by the system into a signal *y(n)*, and is expressed by the general relationship between *x(n)* and *y(n)* as follows:

$$y(n) \equiv H[x(n)] \qquad \text{(Eqn. 3.1)}$$

The symbol *H* denotes the transformation performed by the system on *x(n)* to produce *y(n)*. Figure 3.1 graphically illustrates the mathematical relationship of Equation 3.1.

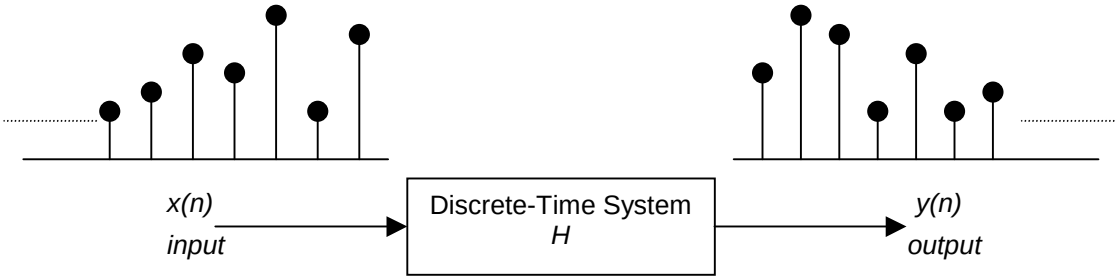


Fig. 3.1. Block diagram representation of a discrete-time system.

The type of discrete-time system focused upon by this research is linear, time-invariant (LTI). A linear system is defined in the following manner. If $x_1(n)$ and $x_2(n)$ are specific inputs to a linear system and $y_1(n)$ and $y_2(n)$ are the respective outputs, then if the sequence $ax_1(n)+bx_2(n)$ is applied to the input, the sequence $ay_1(n)+by_2(n)$ is obtained at the output, where *a* and *b* are arbitrary constants. In a time-invariant system, if the input sequence *x(n)* produces an output sequence *y(n)*, then the input sequence $x(n-n_0)$ produces the output sequence $y(n-n_0)$ for all n_0 [Gold].

3.2: Difference Equations and Recursive Systems

A linear, time-invariant system can have its input-output relationship mathematically described by a difference equation containing constant coefficients. Difference equations are a subset of the class of LTI systems. These equations are extremely important because they offer insights into efficient ways of designing LTI systems. Linear difference equations with constant coefficients are also known as recursive systems. A recursive system is defined as a system whose output *y(n)* at time *n* depends on any number of past output values [Manolakis]. To generally understand the input-output relationship, consider the simple difference equation (first-order recursive system) with the constant *a* as follows:

$$y(n) = ay(n-1) + x(n)$$

Figure 3.2 shows the block diagram and successive values of *y(n)* for all $n \geq 0$ beginning with *y(0)*. Assuming the initial condition *y(-1)* is zero, the computed steps of *y(n)* illustrates the recursive nature of difference equations in that present values are strongly dependent on past values. Another example of a recursive system is Equation 2.9.

$$y(0) = ay(-1) + x(0)$$
$$y(1) = ay(0) + x(1) = a^2y(-1) + ax(0) + x(1)$$
$$y(2) = ay(1) + x(2) = a^3y(-1) + a^2x(0) + ax(1) + x(2)$$

$$y(n) = ay(n-1) + x(n)$$
$$= a^{n+1}y(-1) + a^n x(0) + a^{n-1}x(1) + \ldots + ax(n-1) + x(n)$$

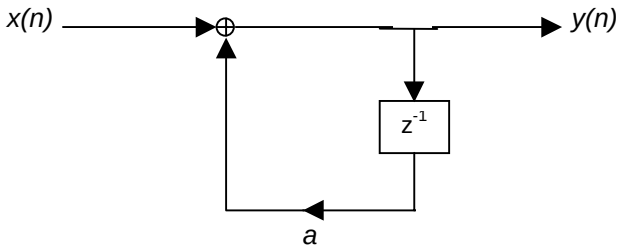


Fig. 3.2. Block diagram of a simple first-order recursive system.

3.3: Infinite Impulse Response (IIR) Digital Filters

IIR digital filters are recursive systems that involve fewer design parameters, less memory requirements, and lower computational complexity than finite impulse response (FIR) digital filters. These are primary advantages of implementing IIR digital filters. If there is no requirement for a linear-phase characteristic within the passband of a digital filter, the aforementioned advantages make IIR filters more attractive to a system designer [Manolakis]. This type of recursive system belongs to an important class of linear time-invariant discrete-time systems characterized by the general linear constant-coefficient difference equation as follows:

$$y(n) = -\sum_{k=1}^{N} a_k y(n-k) + \sum_{k=0}^{M} b_k x(n-k) \qquad \text{(Eqn. 3.2)}$$

Transforming this difference equation into the z-domain by means of the z-transform, such a class of linear time-invariant discrete-time systems is also characterized by the transfer function as follows:

$$H(z) = \frac{\sum_{k=0}^{M} b_k z^{-k}}{1 + \sum_{k=1}^{N} a_k z^{-k}} \qquad \text{(Eqn. 3.3)}$$

Different structures of IIR filters are described by the difference equation in Equation 3.2. These structures are referred to as direct-form realizations. It should be noted that although these structures are different from one another by design, they are all functionally equivalent. Three prominent direct-form realizations are the Direct-Form I, the Direct-Form II, and the Transposed Direct-Form II structures. In terms of hardware implementation, the Direct-Form I structure requires *M+N+1* multiplications, *M+N* additions, and *M+N+1* memory locations. Figure 3.3 depicts this structure as implemented from Equation 3.3.

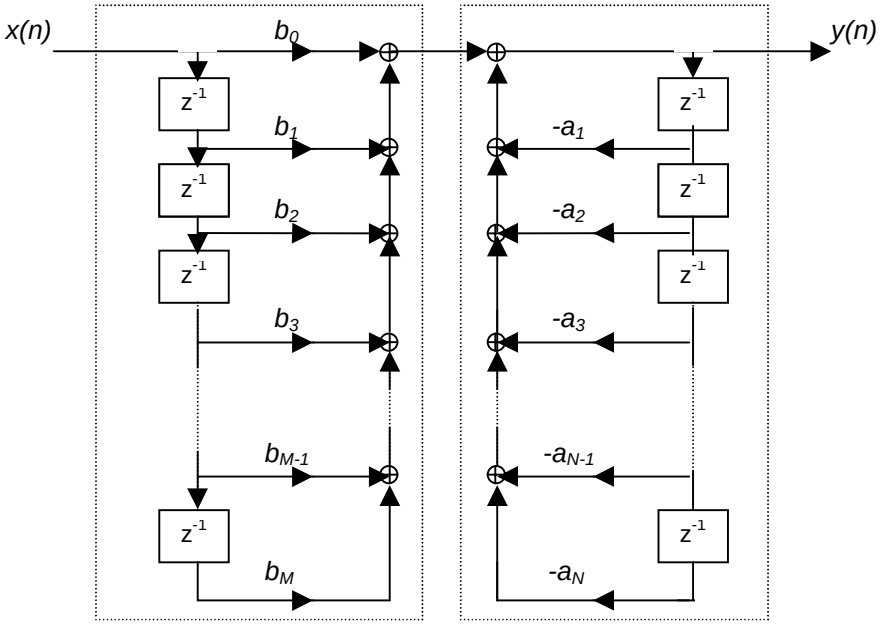


Fig. 3.3. Direct-Form I Realization.

The Direct-Form II structures require *M+N+1* multiplications, *M+N* additions, and the maximum of {*M,N}* memory locations. Because the Direct-Form II structure requires less memory locations than the Direct-Form I structure, it is referred to as being *canonic*. Figure 3.4 shows an IIR digital filter in Direct-Form II format.

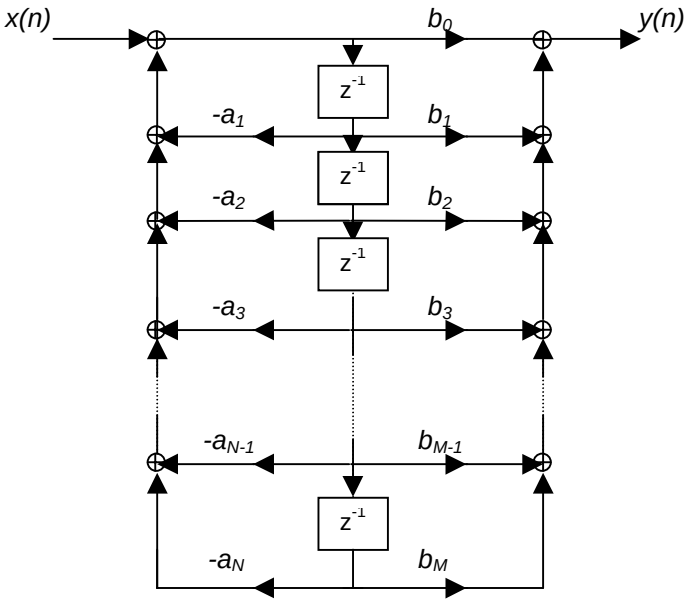
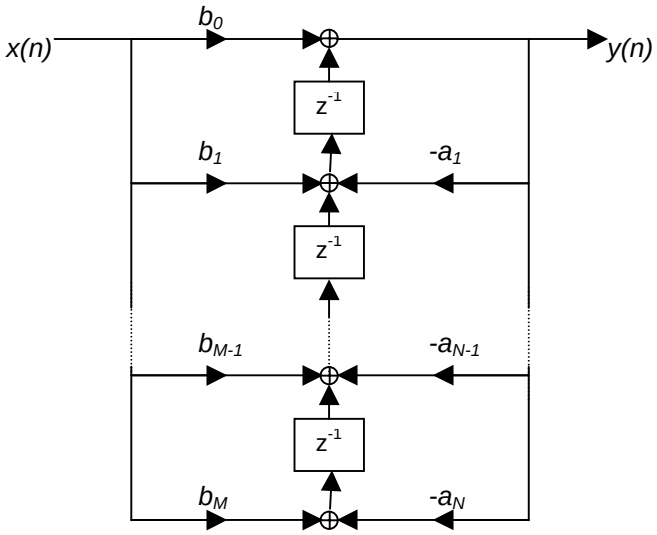


Fig. 3.4. Direct-Form II Realization.

Mathematical manipulation of Equation 3.2 based on Figure 3.4 yields the Transposed Direct-Form II structure. This structure requires the same number of multiplications, additions, and memory locations as the original Direct-Form II structure. Both Direct-Form II structures are more design-preferable compared to the Direct-Form I structure. This is because of the smaller number of memory locations required in their implementation. Figure 3.5 shows an example of the Transposed Direct-Form II structure. Because of this fact, for hardware considerations of this research, the Transposed Direct-Form II structure is the structure of choice for designing quantized, fixed-point IIR digital filters.



$$H(z) = \frac{b_0 + b_1 z^{-1} + \ldots + b_{M-1} z^{M-1} + b_M z^{-M}}{1 + a_1 z^{-1} + \ldots + a_{N-1} z^{N-1} + a_N z^{-N}}$$

Fig. 3.5. Transposed Direct-Form II Structure and Transfer Function *H(z)*.

3.4: Hardware Considerations

Due to finite-precision arithmetic in the realization of n-bit quantized digital filters, nonlinear effects make it extremely difficult to both analyze precisely and predict with 100% accuracy filter performance. Fixed-point realization of digital filters makes quantization effects very important. An example of an unwanted nonlinear effect is limit-cycle oscillations as described in section 2.7 of this thesis. Nonlinear effects at the filter output become a greater problem with high-order filters. As again stated in section 2.7, the solution to significantly minimize nonlinear effects is to decompose digital filters with orders greater than 2 into 2^{nd} order sub-blocks. There are two methods in which decomposing high-order digital filters into 2^{nd} order sub-blocks can achieve the goal of minimization of nonlinear effects. These methods are the parallel-form structure and cascade-form structure. All 2^{nd} order sub-blocks are in Transposed Direct-Form II format for this research.

3.4.1: Parallel-Form Structure

Parallel-form realization of an IIR digital filter can be obtained by performing a partial-fraction expansion on the transfer function *H(z)*. Performing this mathematical function produces the resulting transfer function in the form as follows:

$$H(z) = C + \sum_{k=1}^{K} H_k(z) \qquad \text{(Eqn. 3.4)}$$

The function $H_k(z)$ is in 2nd order form as follows:

$$H_k(z) = \frac{b_{k0} + b_{k1}z^{-1}}{1 + a_{k1}z^{-1} + a_{k2}z^{-2}} \qquad \text{(Eqn. 3.5)}$$

It should be noted that the transfer functions in Equation 3.4 and Equation 3.3 are functionally equivalent in that both are ideal representations of an infinite-precision filter. In Equation 3.4, the constant K is defined as the integer part of (N+1)/2. The constant N is the same constant N in Equation 3.3. Transfer function $H(z)$ is generally composed of poles and coefficients (residues) of the partial-fraction expansion. A more direct result of the partial-fraction expansion of $H(z)$ (Eqn. 3.4) yields the functional equivalent as follows:

$$H(z) = C + \sum_{k=1}^{N} \frac{A_k}{1 - p_k z^{-1}} \qquad \text{(Eqn. 3.6)}$$

The variables p_k and A_k stand for the poles and residues, respectively, in the partial-fraction expansion. The constant C is the same as the variable used in Equation 3.4. If N is odd then $C = 0$. If N is even then $C = b_N/a_N$. Figure 3.6 graphically illustrates a parallel-form structure of an IIR digital filter. The Transposed Direct-Form II realization of each 2nd order sub-block is illustrated in Figure 3.7 [Manolakis].

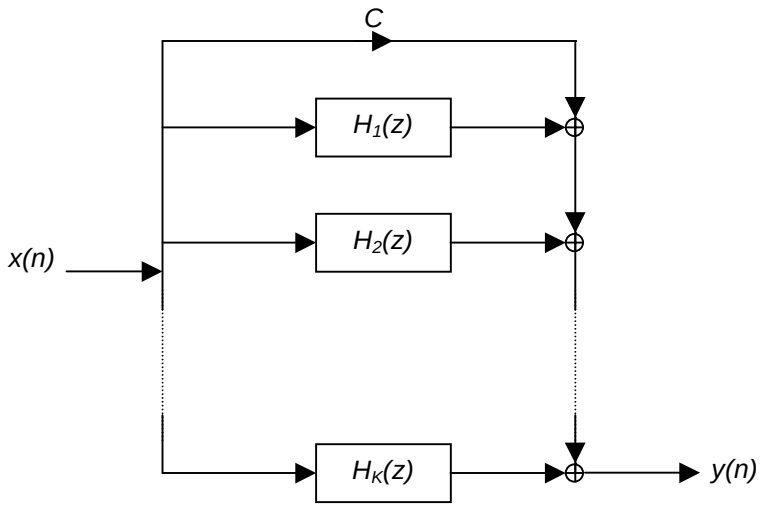


Fig. 3.6. Parallel-Form Structure of an IIR Digital Filter.

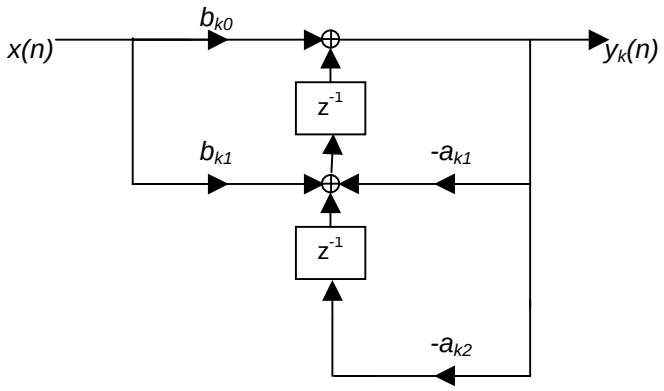


Fig. 3.7. Structure of 2nd order section of parallel-form structure.

3.4.2: Cascade-Form Structure

Cascade-form realization of an IIR digital filter can be obtained by performing mathematical factoring on the transfer function *H(z)* into a cascade of 2nd order sub-blocks. The resulting transfer function can then be expressed as

$$H(z) = \prod_{k=1}^{K} H_k(z) \qquad \text{(Eqn. 3.7)}$$

where *K* is the integer part of (N+1)/2 and $H_k(z)$ has the 2nd order form as follows:

$$H_k(z) = \frac{b_{k0} + b_{k1}z^{-1} + b_{k2}z^{-2}}{1 + a_{k1}z^{-1} + a_{k2}z^{-2}} \qquad \text{(Eqn. 3.8)}$$

Since this is a cascade system, the matter of grouping together a pair of complex-conjugate poles and a pair of complex-conjugate zeros becomes extremely critical. The output of a quantized digital filter is strongly dependent on the sequential ordering of the *K* sub-blocks as well as the exact way in which numerator and denominator sections are paired together. Arbitrarily grouping these terms can be performed on the part of the system designer but at the cost of a potentially high output noise variance. To produce an optimum quantized digital filter, a methodology must be applied in terms of grouping the terms. An established methodology used by this research is as follows:

i. Calculate the poles and zeros of the overall transfer function *H(z)*.
ii. Pick the pole with the largest magnitude and the nearest zero. Choose them and their complex conjugate values for the first 2nd order sub-block (section).
iii. Proceed similarly for the next sub-block (section) with the remaining poles and zeros. Repeat accordingly with the remaining sections.

The above 3-step methodology ensures minimization of quantized round-off noise produced by the n-bit multiplier sub-blocks [Gold]. Figure 3.8 portray the general form of the cascade structure. The Transposed Direct-Form II realization of each 2nd order sub-block is illustrated in Figure 3.9 [Manolakis].

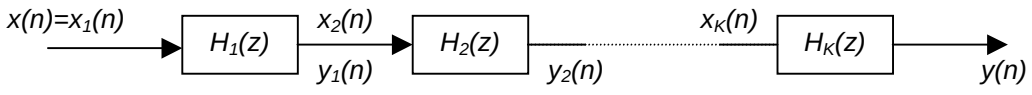


Fig. 3.8. Cascade-Form Structure of an IIR Digital Filter.

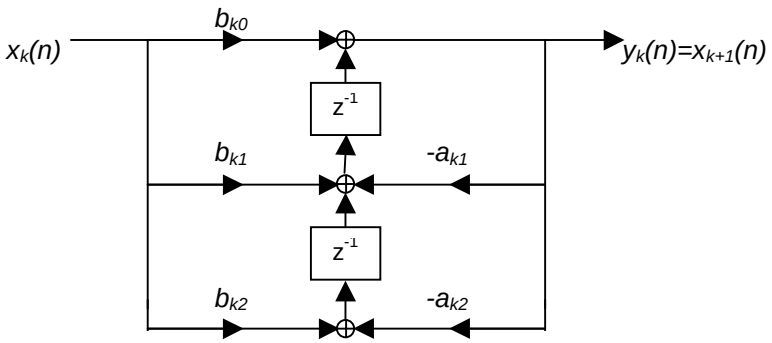


Fig. 3.9. Structure of 2nd order section of cascade-form structure.

CHAPTER 4: Digital Filter Designs

This thesis investigates four different digital designs that each have unique properties in terms of magnitude responses in the pass band, stop band, and transition band regions. These four designs are called Butterworth, Chebyshev I, Chebyshev II, and Elliptic filters [Jackson1]. For this research, the cutoff and stop band frequencies on the magnitude response for these filter designs are defined to be the -3dB point and the -40dB point, respectively.

Butterworth filters have magnitude responses that are “maximally flat” in both the pass and stop bands. The rolloff rate for these designs are typically low and not steep. For this research, rolloff rate is defined in the transition band region, and is described as the rate in which the transition is made from the cutoff frequency to the stop band frequency or vice-versa depending on the type of filter being designed (i.e.: lowpass, highpass, bandpass). If the type of filter being designed is lowpass, the rolloff is from the cutoff frequency to the stop band frequency. If the type of filter being designed is highpass, the rolloff is from the stop band frequency to the cutoff frequency. If the type of filter being designed is bandpass, the rolloff is both from the first stop band frequency to the first cutoff frequency and from the second cutoff frequency to the second stop band frequency. For an ideal filter, the rolloff rate is infinite and has an infinite slope. In addition, because there is an inverse relation between rolloff rate and width of transition band, a transition band does not exist for an ideal filter. In other words, the width of the transition band is zero.

Chebyshev filters have shorter transition bands than Butterworth filters. Hence, the rolloff rate is greater and the width of the transition band is significantly smaller. There are two different types of Chebyshev filters. Chebyshev Type I filters have ripple in the pass band region and “flatness” in the stop band region. Chebyshev Type II filters have ripple in the stop band region and “flatness” in the pass band region. Designing Type I provides better attenuation in the stop band region while designing Type II provides less signal distortion at the filter output. The system designer selects which type to use based on which band region is more important.

Elliptic filters have the narrowest transition band compared to both the Butterworth and Chebyshev designs. The cost of such a narrow band is ripple in both the pass and stop band regions.

Figure 4.1 illustrates the frequency magnitude response of three of the four digital filter designs. The results are based on 32768 samples. Each filter is 5^{th} order and Table 4-1 shows the relative frequency bandwidth of each design. A relative frequency of 0.5 is equal to half the digital filter sampling frequency.

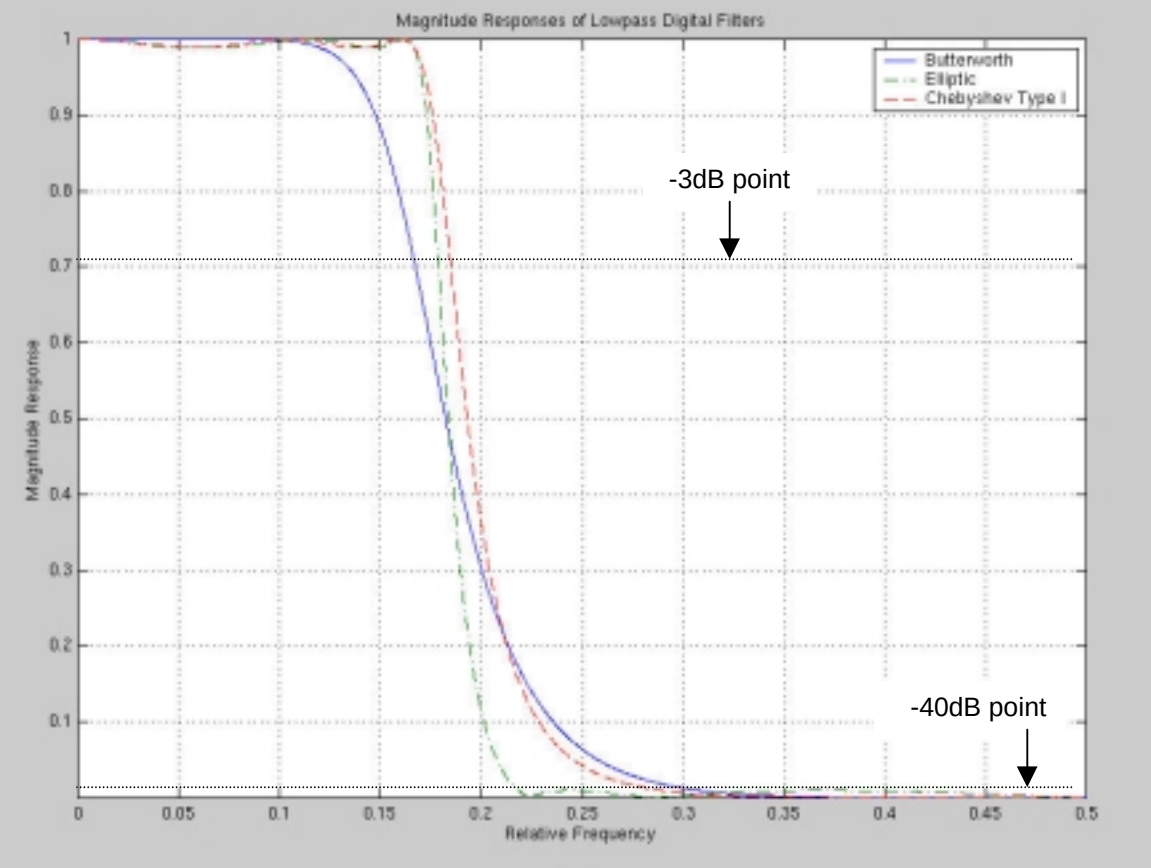


Fig. 4.1. Frequency magnitude response of Butterworth, Chebyshev Type I, and Elliptic Filters.

From the above figure, three important observations should be noted. First, in the stop band region, attenuation using the Chebyshev Type I filter is better than both the Elliptic and Butterworth filters. Second, the Butterworth filter introduces the least pass band distortion because the signal is flat in the pass band. Thirdly, the Butterworth filter has the largest transition band while the Elliptic filter has the smallest transition band.

Design Type	Cut-off (Rel. Frequency)	Stopband (Rel. Frequency)	Transition Width (Rel. Frequency)
Butterworth	0.16666666666667	0.30800000000000	0.14133333333333
Chebyshev Type I	0.18466666666667	0.28900000000000	0.10433333333333
Elliptic	0.17866666666667	0.21833333333333	0.03966666666667

Table 4-1. Relative frequency bandwidth results.

CHAPTER 5: COSSAP Saturation Modes for Fixed-point Binary Adders

5.1: Overflow in Binary Addition

When performing two's complement, fixed-point addition, there is always the possibility of binary overflow. Overflow is a consequence of using a finite number of bits. An illustrative example would be to use 4 bits in two's complement notation. Assuming integer calculations, the resulting decimal numbers are in the range of –8 through +7. When adding two 4-bit numbers, the chance of overflow exists. For example, adding $0001_2 = 1_{10}$ to $0111_2 = 7_{10}$ yields $1000_2 = -8_{10}$. This is clearly an error and a simple example of binary overflow using addition. Figure 5.1 illustrates a curve of output versus input of a 4-bit digital system using two's complement arithmetic. This figure shows the output for normal binary addition and binary overflow. The x-axis represents the ideal decimal value (infinite-bit value) while the y-axis represents the corresponding 4-bit two's complement value. Both axes are shown in base 10 for better illustration. Ideally, for an infinite-bit adder, there is no overflow and Figure 5.1 would have a one-to-one ratio (i.e.: a slope of 1) going to infinity in both directions from the origin.

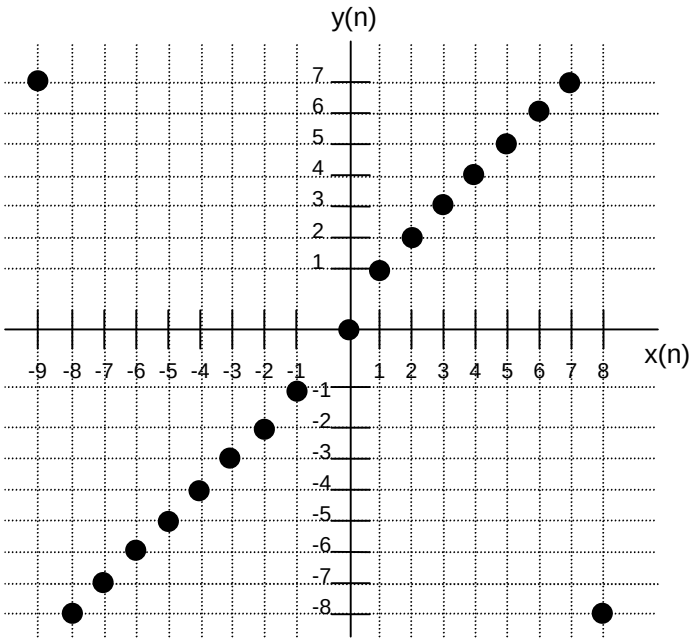


Fig. 5.1. Overflow in a 4-bit two's complement digital system.

5.2: COSSAP Saturation Modes

The DSP library in COSSAP contains overflow-correcting algorithms for n-bit fixed-point adders. These algorithms generally handle overflow by putting the output of an n-bit adder into a saturation mode. Depending on the values of the n-bit fixed-point inputs, the saturation mode will put the output of the adder to either its binary maximum or binary minimum two's complement n-bit representation. For the above case where $0001_2 = 1_{10}$ was added to $0111_2 = 7_{10}$, the resulting n-bit output, depending on the COSSAP saturation mode, would be $0111_2 = 7_{10}$. The output is clipped to the maximum two's complement 4-bit representation. The output is saturated. This is analogous to saturation in analog systems. COSSAP contains 3 saturation modes designed specifically for their n-bit fixed-point adders. The saturation modes available in the COSSAP DSP library are as follows:

SaturationMode = 0 : truncation of upper (most significant) bits

SaturationMode = 1 : truncation of upper (most significant) bits and clipping to minimum or maximum value

SaturationMode = 2 : truncation of upper (most significant) bits and symmetrical clipping (to minimum+1 or maximum value)

Figures 5.2 through 5.4 show curves of output versus input of 4-bit two's complement numbers in integer format. Each figure shows what the output would be for a specific saturation mode. The x-axis represents the ideal decimal value (infinite-bit value) in the range of –10 to +10 in unit increments. The y-axis represents the corresponding 4-bit two's complement value. Both axes are shown in base 10 for better illustration.

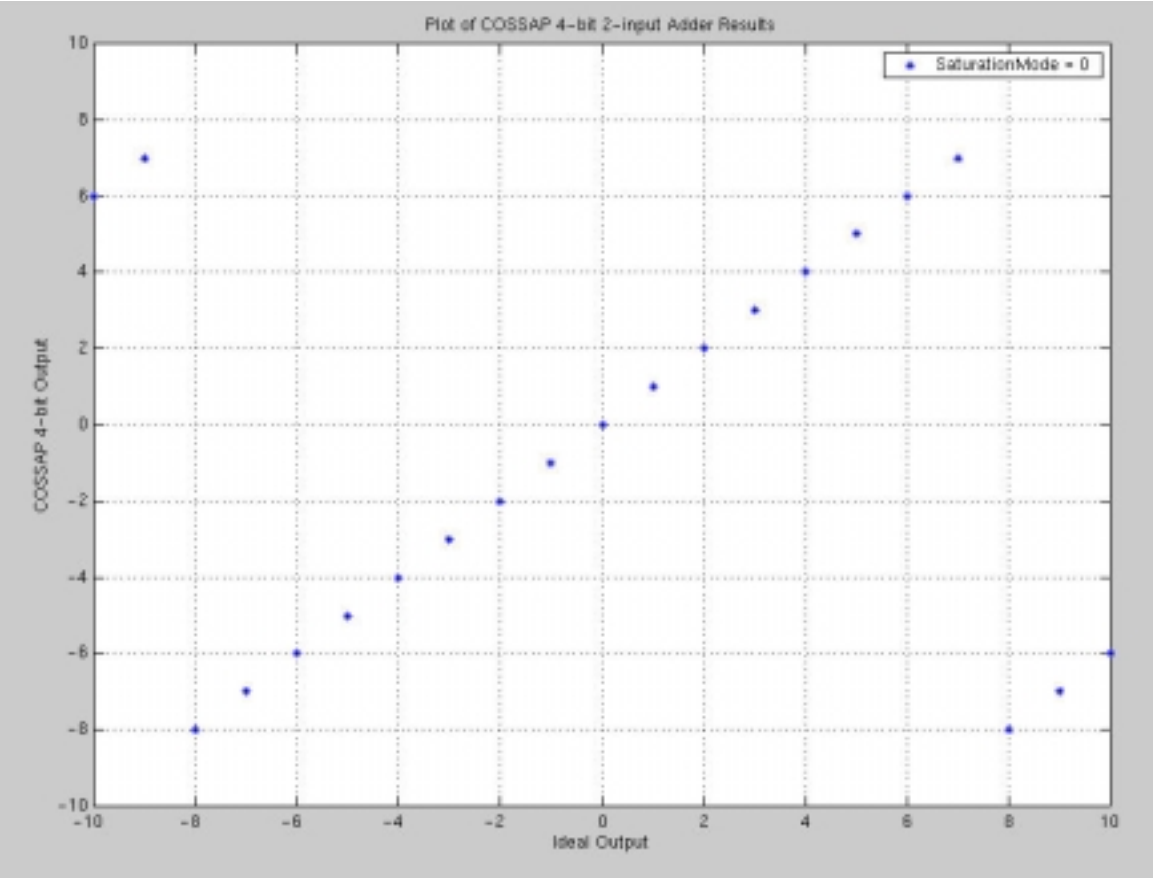


Fig. 5.2. Two's complement output of 4-bit fixed-point adder in the mode as follows: SaturationMode = 0

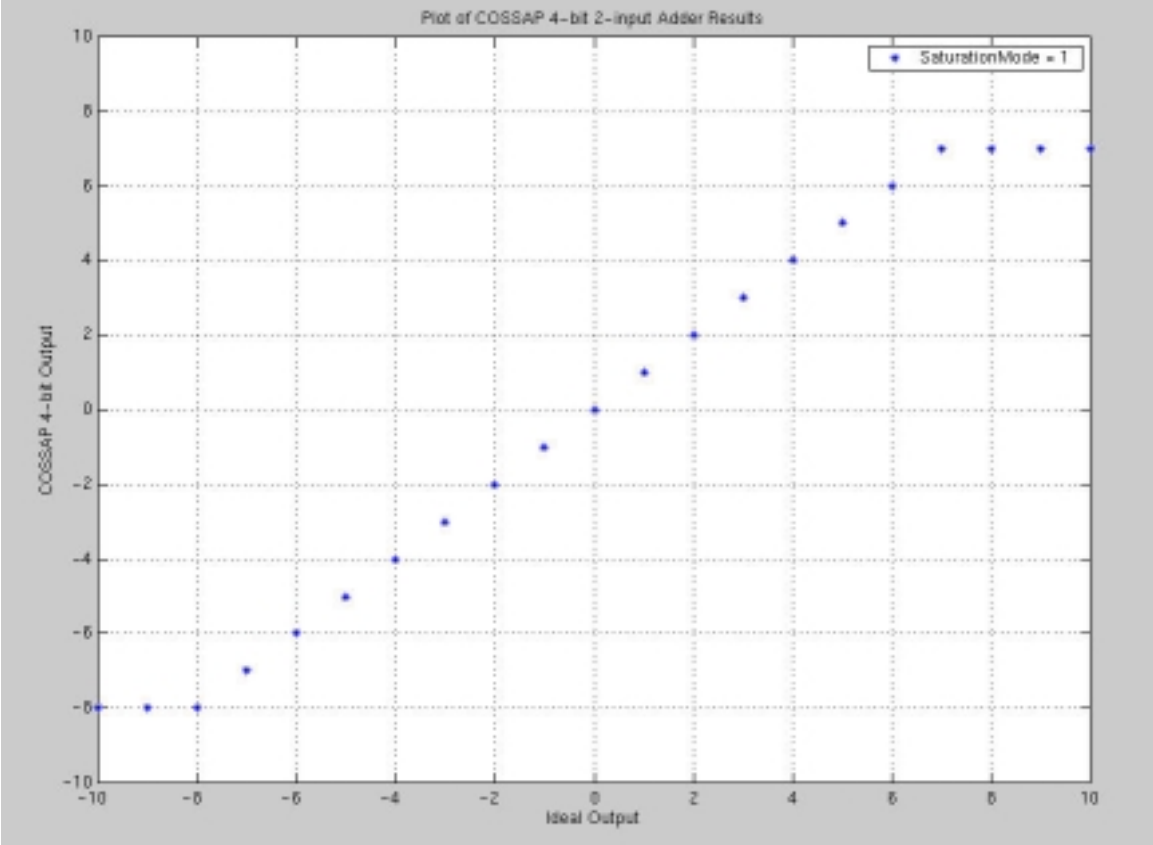


Fig. 5.3. Two's complement output of 4-bit fixed-point adder in the mode as follows: SaturationMode = 1

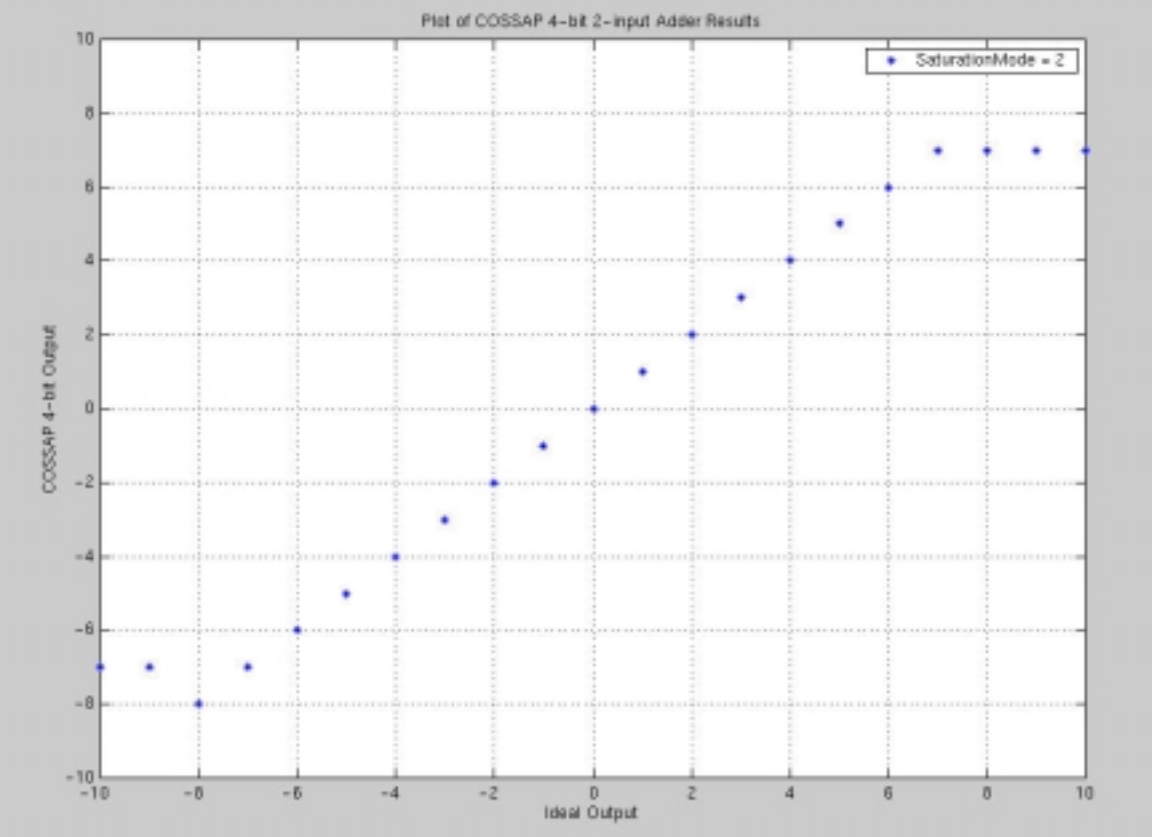


Fig. 5.4. Two's complement output of 4-bit fixed-point adder in the mode as follows: SaturationMode = 2

The performance criterion used to determine which of the 3 saturation modes is the best is the equation as follows:

$$P_e = \sigma_e^2 + m_e^2 \qquad \text{(Eqn. 5.1)}$$

This equation defines the error sequence between the ideal and quantized results. The variables P_e, σ_e^2, and m_e^2 stand for the round-off noise power, the error variance, and the error mean, respectively [Bailey]. Table 5-1 shows the round-off noise power results for each of the three COSSAP saturation modes. Saturation Mode 1 was chosen for this research to minimize the round-off noise power. This mode puts the n-bit fixed-point adder into either its binary minimum or binary maximum when an n-bit fixed-point overflow is computed.

Saturation Mode	Round-off Noise Power (P_e)
0	4.018932970521540e+03
1	0.88265306122449
2	1.81834467120181

Table 5-1. Round-off power results of COSSAP saturation modes.

CHAPTER 6: COSSAP Round-off Modes for Fixed-point Binary Multipliers

Quantized multiplier sub-blocks are the source of noise in digital filters [Chirlian]. The DSP library in COSSAP contains multiplier sub-blocks that can be programmed to one of four round-off modes. These modes specify the type of arithmetic n-bit rounding at the multiplier output. These round-off modes are listed as follows:

RoundMode = 0 : truncation of lower (least significant) bits
RoundMode = 1 : mathematical rounding
RoundMode = 2 : symmetrical rounding
RoundMode = 3 : truncation to zero (symmetrical truncation)
truncation of absolute value and correction of the sign

Figures 6.1 through 6.4 show an illustrative example of the output of an 8-bit fixed-point multiplier in the various COSSAP round-off modes. The 8-bit input into the four different multipliers are the same. The input is an arbitrary sinusoid within the magnitude range of the 8 bits. The implied binary point is between the second and third bits. Using two's complement representation, the range of the 8 bits is $[-2, 2\text{-}2^{-6}]$. The output of the multiplier is 8 bits wide and has the same fixed-point format as the input. The 8-bit constant in the multiplier is $00.101110_2 = 0.71875_{10}$. Each of the four multiplier sub-blocks is in stand-alone mode, and each figure shows the MATLAB output and the COSSAP output for the corresponding COSSAP round-off mode. Because of its floating-point precision, the software package MATLAB will be used as a source of ideal comparison. MATLAB multiplies the 8-bit input and the 8-bit constant to produce a floating-point number. COSSAP multiplies the 8-bit input and the 8-bit constant to produce an 8-bit output. The aforementioned arithmetic error sequence is in reference to the error between MATLAB's floating-point precision (ideal case) and COSSAP's 8-bit finite-precision (quantized case). Throughout this thesis, the terms *ideal* and *unquantized* will be used interchangeably. The DSP library in COSSAP provides the designer the unique ability to define the binary word length of the input and the output. This programming property of COSSAP is true for both n-bit adder and n-bit multiplier sub-blocks. This is extremely important when designing recursive systems like IIR digital filters where feedback loops play critical computational roles.

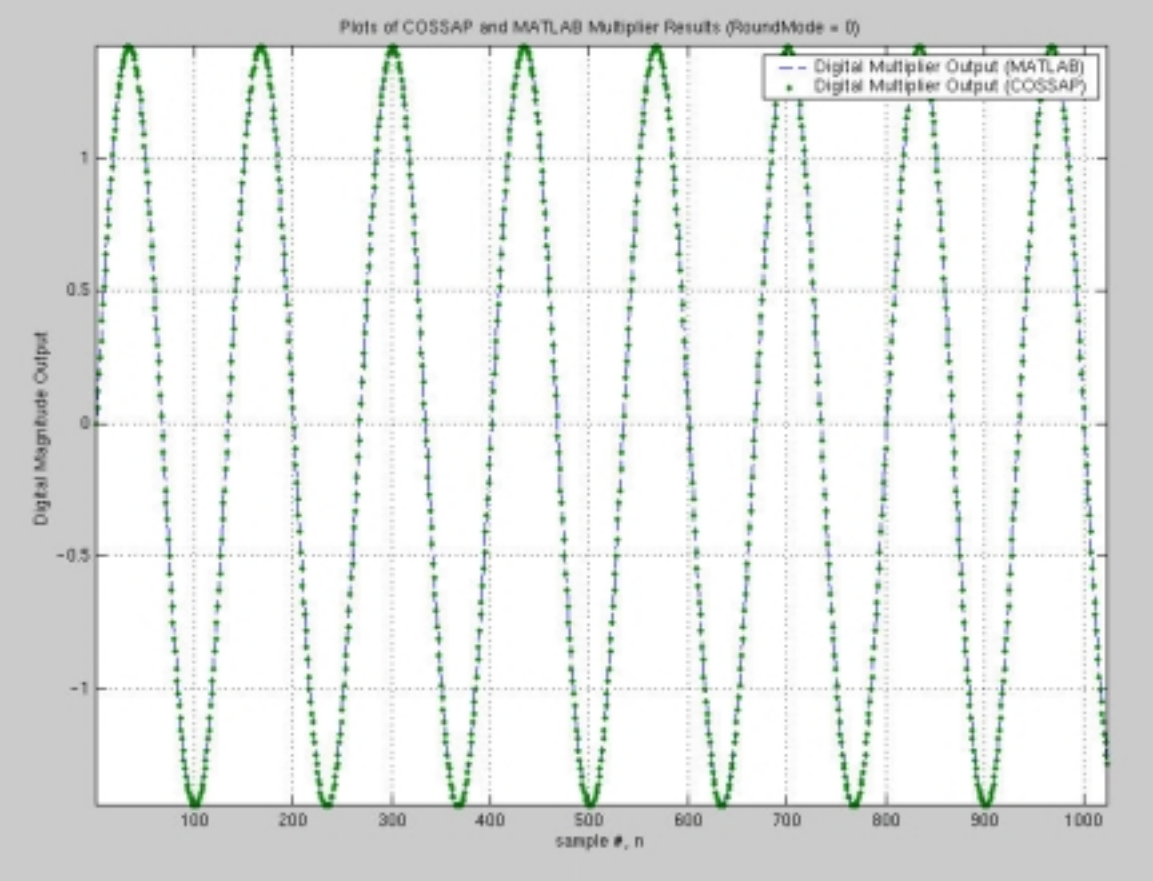


Fig. 6.1. Two's complement output of 8-bit fixed-point multiplier with RoundMode = 0.

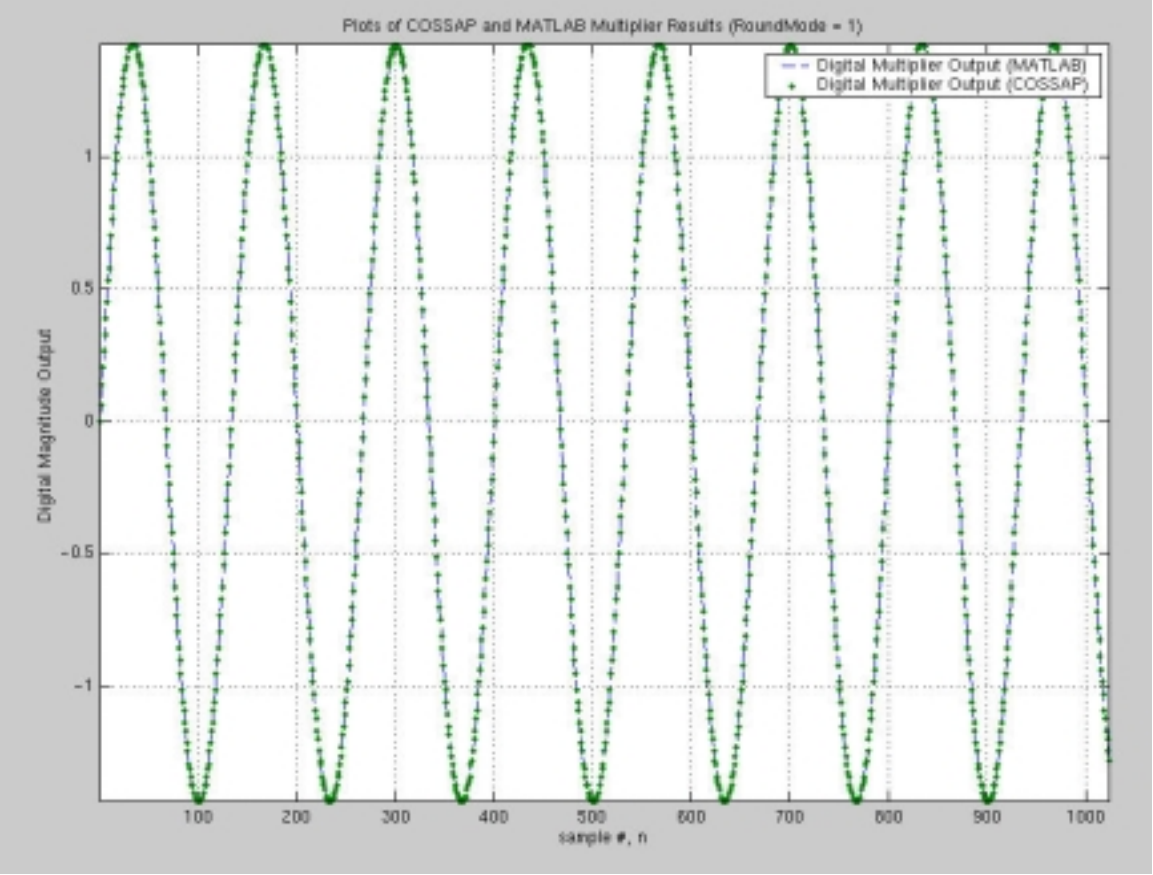


Fig. 6.2. Two's complement output of 8-bit fixed-point multiplier with RoundMode = 1.

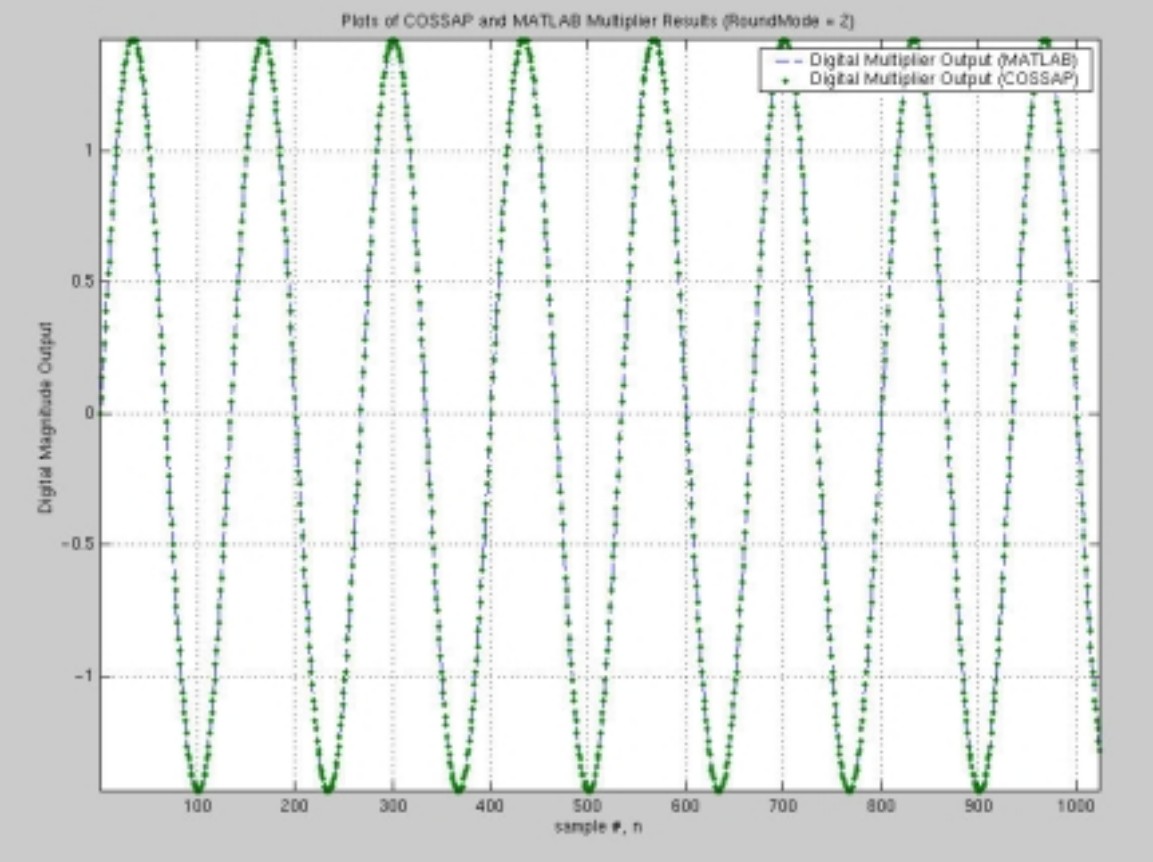


Fig. 6.3. Two's complement output of 8-bit fixed-point multiplier with RoundMode = 2.

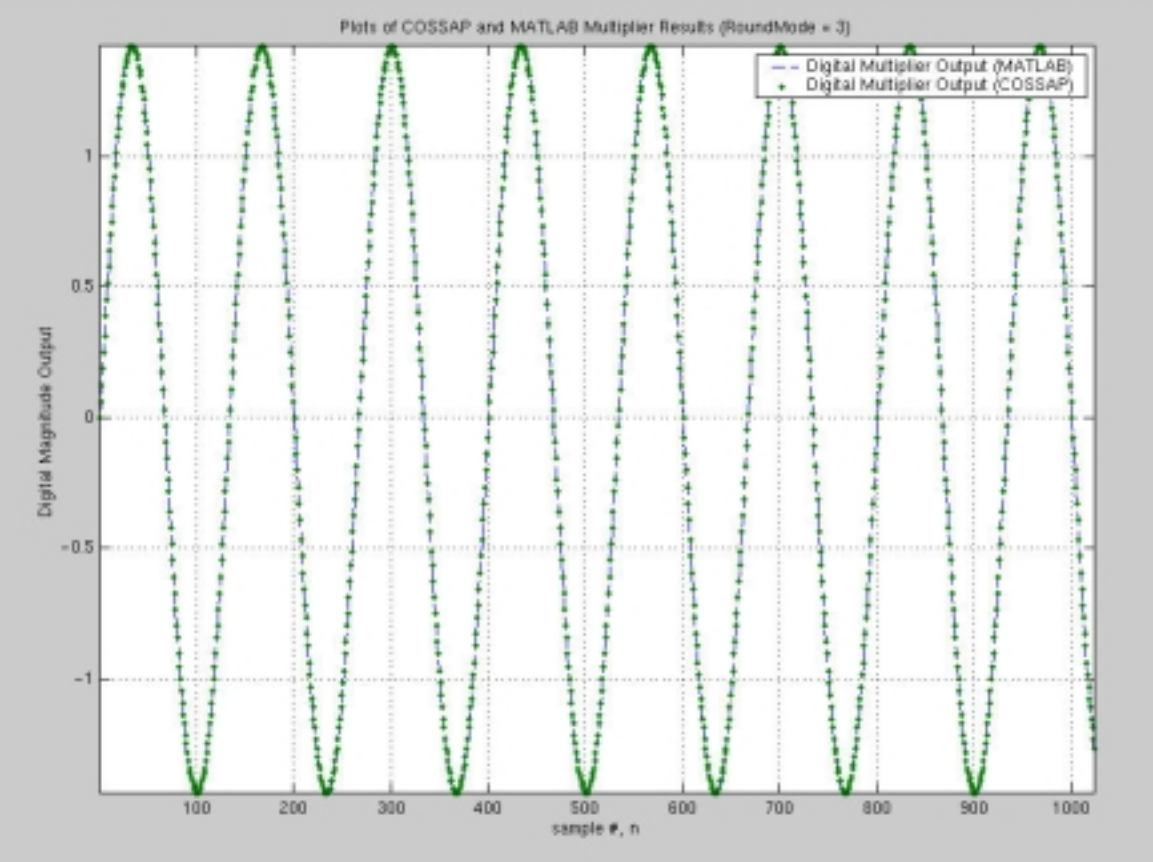


Fig. 6.4. Two's complement output of 8-bit fixed-point multiplier with RoundMode = 3.

Table 6-1 shows the round-off noise power results for the 4 different COSSAP modes. Results are based on 4096 samples. Based on Table 6-1, *RoundMode = 2* is the best mode for designing an n-bit IIR digital filter. However, in practice, this is not necessarily true. The above multiplier sub-blocks are set in stand-alone mode for the purpose of analyzing the four COSSAP round-off modes. Because this research deals with IIR digital filter design, which are recursive systems, nonlinear effects due to the multipliers make it extremely difficult to predict how the multiplier will behave. The level of difficulty substantially increases when designing high-order filters with many multiplier sub-blocks. The only guaranteed solution is to simulate the overall design and measure its performance in all 4 different round-off modes. This is an effective approach.

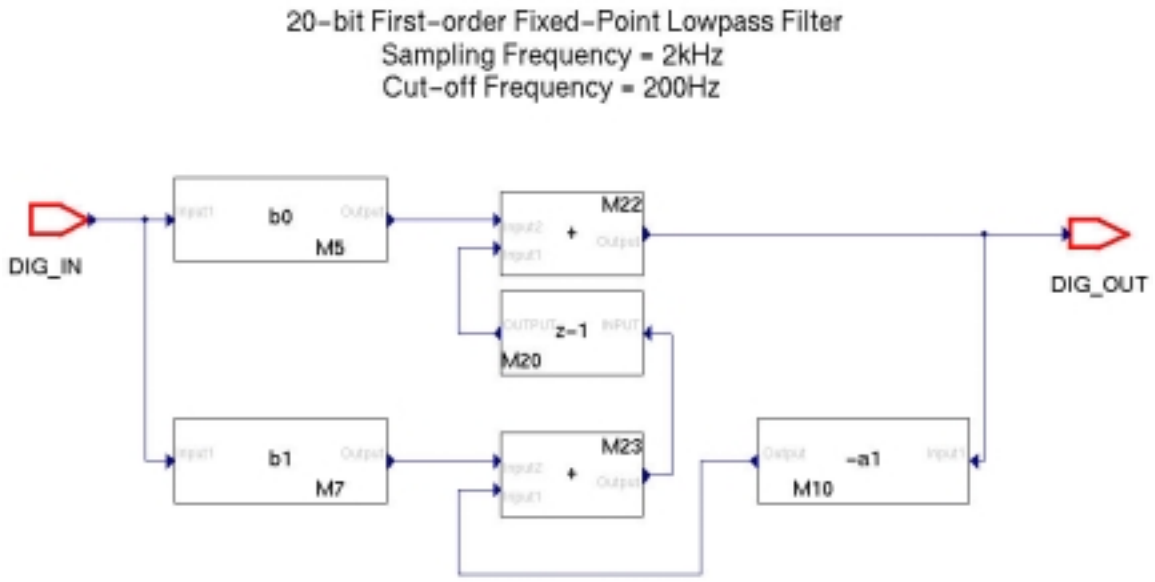
Round-off Mode	Round-off Noise Power (P_e)
0	6.113431873741523e-05
1	4.998328439872081e-08
2	4.827878486051140e-09
3	6.568585265585772e-07

Table 6-1. Round-off power results of COSSAP round-off modes.

CHAPTER 7: Editing VHDL-generated Code Produced from COSSAP

7.1: Designing IIR filters in COSSAP Block Diagram Editor

The designated DSP libraries used to produce VHDL code are the **bittrue** and **radix_fxp** libraries. As stated in the introduction to this thesis, a broad background knowledge of the high-level tool COSSAP is required. Figure 7.1 is an example of a 20-bit first-order lowpass digital filter designed using COSSAP's Block Diagram Editor. The design is a Transposed Direct-Form II Structure. The sampling frequency and cut-off frequency (-3dB point) are 2kHz and 200Hz, respectively. Based on this frequency information, the quantized transfer function is as follows:



$$H(z) = \frac{0.2452373504638671875 + 0.2452373504638671875\, z^{-1}}{1 - 0.509525299072265625\, z^{-1}}$$

Fig. 7.1. 20-bit first-order lowpass digital filter.

As a point of reference to the reader, each sub-block in Figure 7.1 has an instance name. The instance name is the letter M followed by a number. Instance names M5, M7, and M10 are 20-bit multiplier sub-blocks. Instance names M22 and M23 are 20-bit adder sub-blocks. Instance name M20 is a delay sub-block. This sub-block is basically an n-bit parallel shift register that stores past values to be used for future computations. The adder and multiplier sub-blocks are located in the **radix_fxp** library. The delay sub-block is located in the **bittrue** library. The adder and multiplier sub-blocks are combinational circuits while the delay sub-block is a synchronous sequential circuit.

To produce VHDL code, the utility program **xvcg** in COSSAP must be run after the appropriate compilation steps have been executed. The utility program **xvcg** generates code using the COSSAP HDL Code Generator (VCG). Figure 7.2 shows the command window for this utility program.

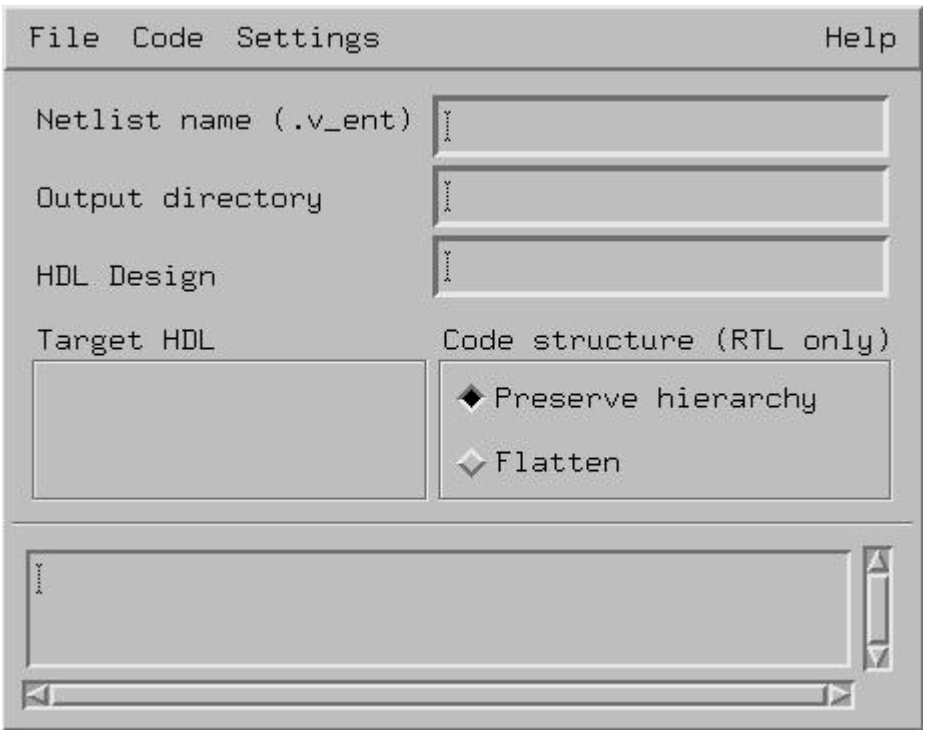


Fig. 7.2. Graphical window of the utility program **xvcg**.

The online tutorial of the utility program **xvcg** in COSSAP is available to the reader for a detailed understanding on how to use this program. It is strongly suggested that the reader read this COSSAP tutorial. Table 7-1 shows what options need to be followed to produce VHDL code.

Step #	Action
1	Settings -> Set target -> Synopsys VHDL BC
2	Settings -> Set file structure -> File structure -> single file (excl. COSSAP packages)
3	Settings -> Set BC options -> Handshake signals -> no handshake Settings -> Set BC options -> Pipeline options -> No pipeline
4	Code -> Create

Table 7-1. Steps to produce VHDL code for a digital filter.

These steps will generate a VHDL behavioral description of the digital filter in Figure 7.1. Step 1 produces a VHDL behavioral architecture of the digital filter. Even though the final design is to be a structural architecture, this initial behavioral architecture is extremely important. Step 2 excludes a copy of the COSSAP VHDL packages in the resulting VHDL-generated code. These packages are required for COSSAP binary arithmetic and should have already been analyzed using the current VHDL Synopsys compiler. The resulting VHDL-generated code has use clauses that reference these packages. The use clauses access these compiled packages from the designated COSSAP library and make their contents visible. At the time of this writing, the designated library in COSSAP is called BITTRUE_VHDLSNPS. The designer should know where the COSSAP packages are located for the **bittrue** and **radix_fxp** libraries. Step 3 eliminates both handshaking and pipeline hardware aspects of this digital filter in Figure 7.1. These two aspects were eliminated because the goal is to produce VHDL code that resembles the Transposed Direct-Form II Structure.

7.2: Deficiencies in the VHDL code generated by xvcg

The VHDL code generated by **xvcg** has two undesirable features. The first undesirable feature is that the generated code produces registered inputs. The second undesirable feature is that it produces a design that has an active-high reset. In the IIR digital filter shown in Figure 7.1, the only memory element is the delay unit (parallel shift register). The input *x(n)* is not registered. In actuality, the input *x(n)* is the output of an A/D converter. The outputs of A/D converters are typically registered (or latched). The values of *x(n)* are strongly dependent on the A/D converter's sampling frequency, f_s. The delay unit in Figure 7.1 must operate at the same sampling frequency as the A/D converter. The registered input produced by **xvcg** delays the computed output to *y(n)* by one clock cycle (period). To compensate for the first undesirable feature, the designer must convert the code to resemble the Transposed Direct-Form II Structure. The first step in this technical endeavor is to design the sub-blocks individually using both the COSSAP Block Diagram Editor (BDE) and **xvcg**. In VHDL, it is an established industry practice to start with behavioral architectures (behavioral domain) and end with a single structural architecture (structural domain). This practice employs the concept of modularity in which partitioning a top-level design allows the system designer the ability to minimize design complexity and ensure that low-level components are correctly functioning.

7.2.1: Fixed-point Multiplier Sub-blocks

Figure 7.3 illustrates a block diagram of instance M5 of Figure 7.1.

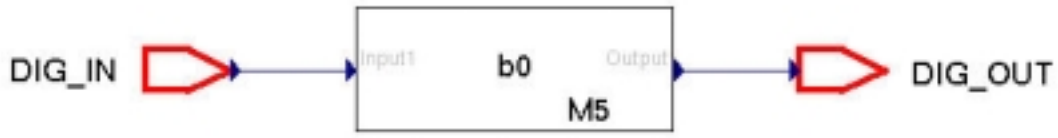


Fig. 7.3. COSSAP BDE of instance M5 of Figure 7.1.

Instance M5 is a 20-bit fixed-point multiplier sub-block. Executing **xvcg** produces VHDL code for both a combinational and sequential implementation. Notice that instance M5 in Figure 7.1 is a purely combinational circuit. The VHDL-generated code, excluding the comments, can be found in Appendix A.

The designer should convert this VHDL-generated code to resemble a combinational circuit by creating another copy of the code and performing specific editing steps. This new copy is to be converted to a VHDL code with generic parameters and be used as a VHDL library component. This conversion consists of 5 editing steps the results of which can be found in Appendix A of this research. These editing steps are a one-time procedure that covers future multiplier sub-block design and VHDL instantiation. For the example in Figure 7.1, instances M7 and M10 are covered. Lastly, these edits remove the two undesirable features stated earlier in section 7.2.

7.2.2: Fixed-point Adder Sub-blocks

Figure 7.4 illustrates a block diagram of instance M22 of Figure 7.1. Instance M22 is a 20-bit fixed-point adder sub-block.

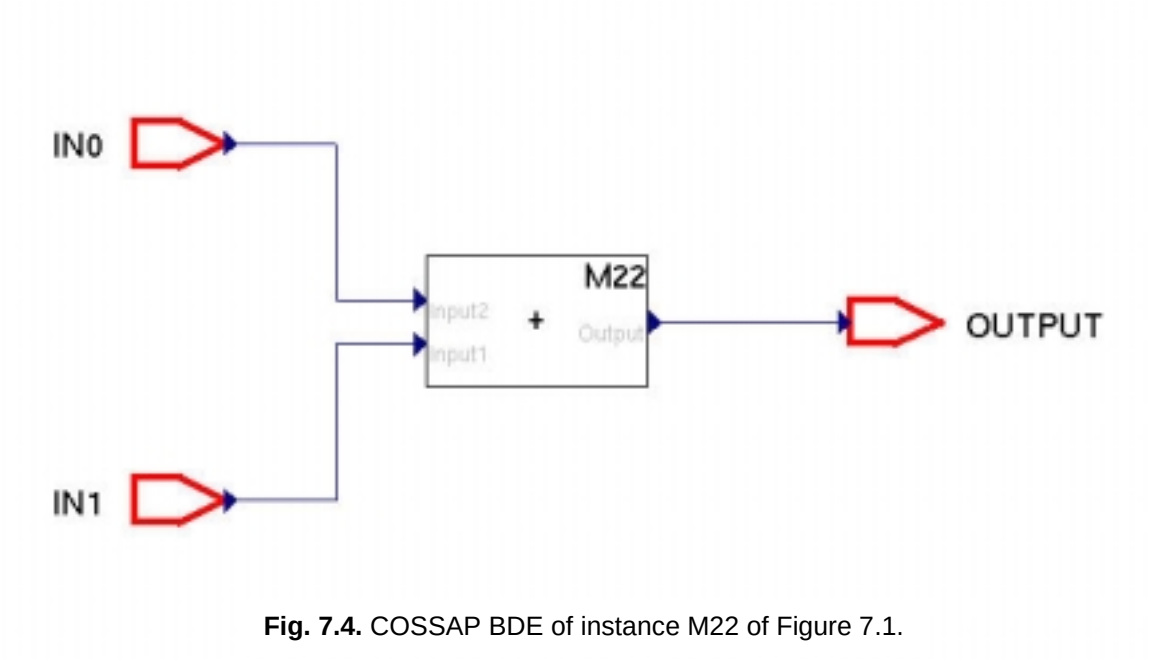


Fig. 7.4. COSSAP BDE of instance M22 of Figure 7.1.

The same 5-step editing methodology with minor changes is applied to this component. This generic VHDL code is used as a library component. Step by step results can be found in Appendix A. The generic code covers instance M22 as well. Functional verification is left to the designer. In Appendix A of the resulting generic multiplier and generic adder fixed-point conversions, the reader should notice the difference between the COSSAP function *fxp_round()*. For the adder, one of the parameters is the integer 0. Since the adder sub-blocks do not produce additional quantization noise [Chirlian], the default round-off mode of 0 is sufficient. Changing this parameter value to any other COSSAP round-off mode (1, 2, or 3) will not produce a difference at the adder output. It is noted here that this sub-section (7.2.2) deals with a 2-input fixed-point adder. For digital filters with orders greater than and equal to 2, adders with 3 inputs are required. Figure 7.5 is a block diagram of a 20-bit 3-input fixed-point adder.

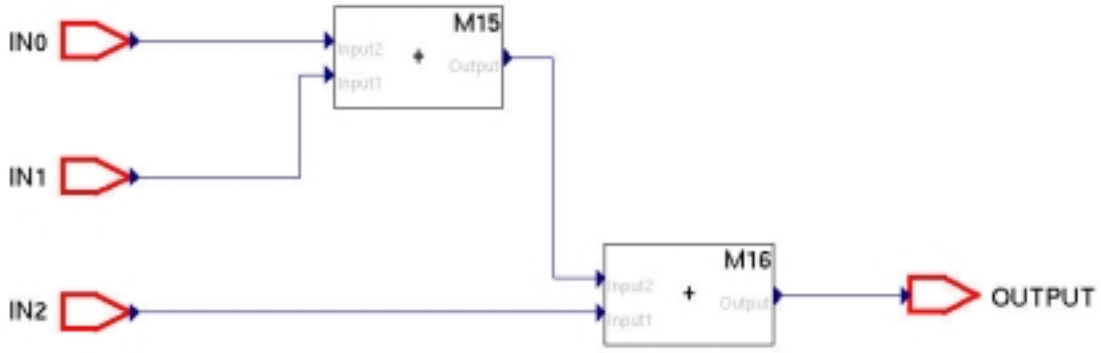


Fig. 7.5. COSSAP BDE of a 3-input fixed-point adder.

Following the aforementioned editing steps, the generic VHDL code can be found in Appendix A.

It is lastly stated here that the fixed-point adders used in this research (2 inputs and 3 inputs) have the same integer/fraction bit allocations when it comes to fixed-point representation at the adder input. In other words, the number of bits used to represent the integer value for one input is used for the second input for a 2-input adder. Likewise, for a 3-input adder, the number of bits used to represent the integer value for one input is used for the second and third inputs.

7.2.3: Fixed-point Delay Sub-blocks

Figure 7.6 illustrates a block diagram for instance M20 of Figure 7.1. This is a clock-dependent delay sub-block.

Fig. 7.6. COSSAP BDE of a 20-bit delay sub-block.

The VHDL-generated code after executing **xvcg** is left for the reader to produce. Through more efficient editing techniques, the resulting generic VHDL code used for this research can be found in Appendix A.

As for the second feature mentioned in section 7.2 (active-high reset), the designer has the ability to choose between designing either an active-high or active-low sub-block depending on design specifications. In short, the designer is not confined to designing filters with active-high resets. As with the multiplier and adder sub-blocks, functional verification of the delay sub-block is left to the designer. For this research, all digital filter designs have an active-high reset.

7.3: Designing IIR Filters in VHDL

As stated in section 7.1, the design goal is to build a digital filter in the Transposed Direct-Form II Structure. Section 7.2 dealt with how to convert *specific* VHDL-generated code for the sub-blocks into *general* (generic) VHDL code. The designer now has the basic building blocks needed to design the required digital filter as shown in Figure 7.1. This VHDL design is a generic structural architecture. An example of a generic VHDL structural code is found in Appendix A.

Functional verification is left to the designer to ensure that the generic VHDL structural architecture produces the same results as the initial VHDL-generated code. Because the multiplier sub-blocks are almost always different from one another in the same structure in terms of multiplier coefficients, different architectures of the same multiplier are required. Basically, the different architectures are based on the constants with *RoundProdWidth_* prefixes. These constants, which are produced by COSSAP's utility program **xvcg**, are crucial for the arithmetic functions used in the VHDL DSP library.

The designer must know which architecture is to be bound (configured) to which sub-block in the top-level structural architecture. The designer can find this information from the initial VHDL-generated code of the digital filter. Recall that this initial VHDL-generated code contains a behavioral architecture. The designer simply has to search for that part of the code where these constants are declared. The section that has

the *RoundProdWidth_* prefixes contains this information. These prefixes are directly linked to the schematic of the digital filter in COSSAP's Block Diagram Editor. For example, if a constant is named *RoundProdWidth_M_M12_1_3*, the designer knows that instance M12 is a multiplier sub-block and the constant value declared there is associated with that particular sub-block. The manner in which to read this information is simple. Sum the integers not including the n-bit length of the designed filter. For example, if the design is a 20-bit digital filter and the constant is *1 + 5 + 20 – 1*, the designer should sum together the integers 1, 5, and –1 to compute the value 5. The designer would then know which generic architecture needs to be bound (configured) to which multiplier sub-block in the top-level architecture. Specifically, the generic declaration of this constant in the architecture to be bound for this particular example is as follows:

```
constant RoundProdWidth_M_M12_1_3 : INTEGER := BIN_LENGTH + 5 ;
```

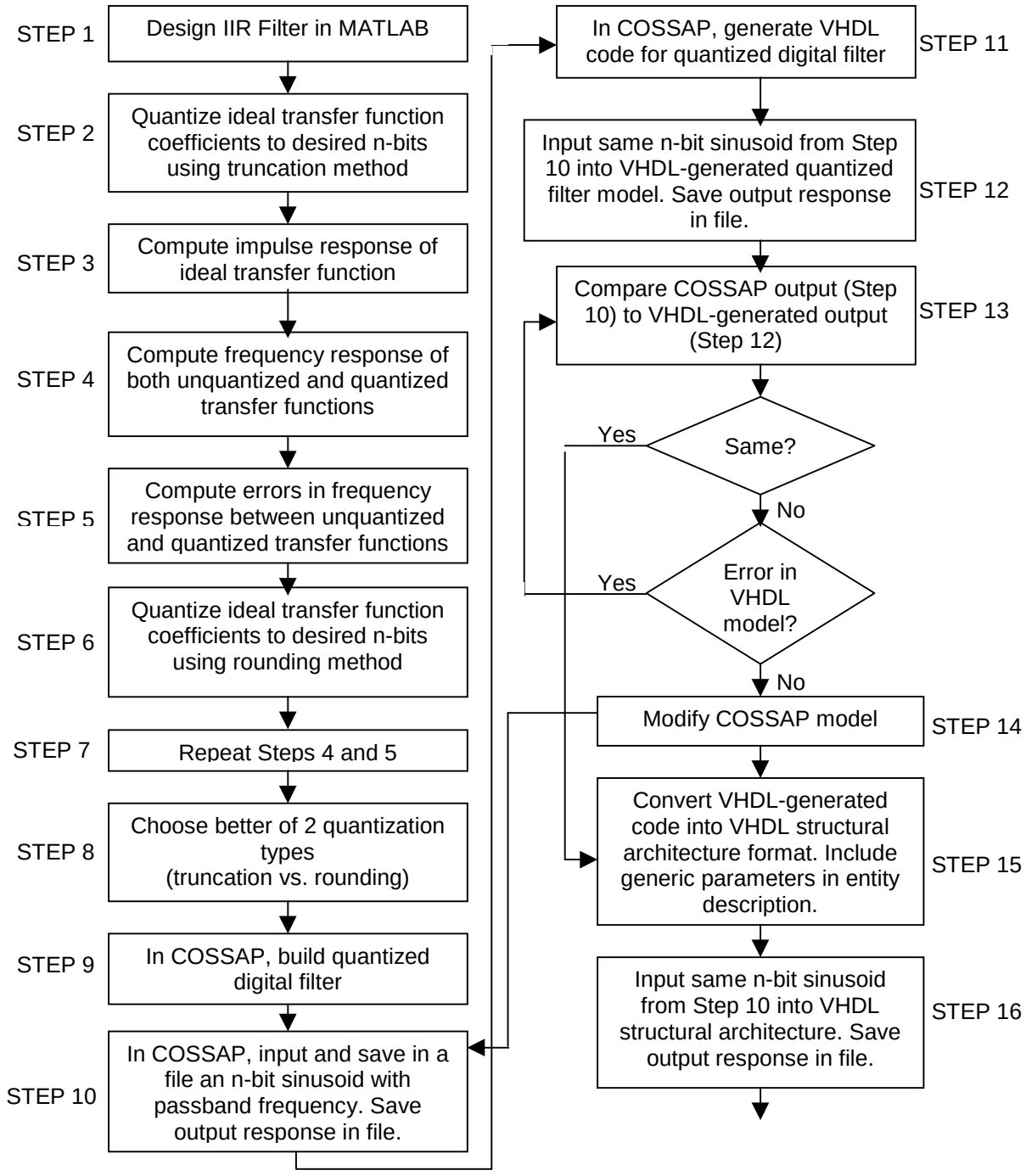
The variable *BIN_LENGTH* is obviously equal to 20 for this example. The same methodology is applied to the adder sub-blocks. In this case, the constant that has the required information contains the *RoundWidth_* prefix.

This section demonstrates the benefit of choosing the *Synopsys VHDL BC* option in Step 1 of Table 7-1. Although the other option, Synopsys VHDL RTL, contains the same necessary bounding information, the actual gathering of the information requires a painstaking search of the VHDL-generated code on the part of the designer.

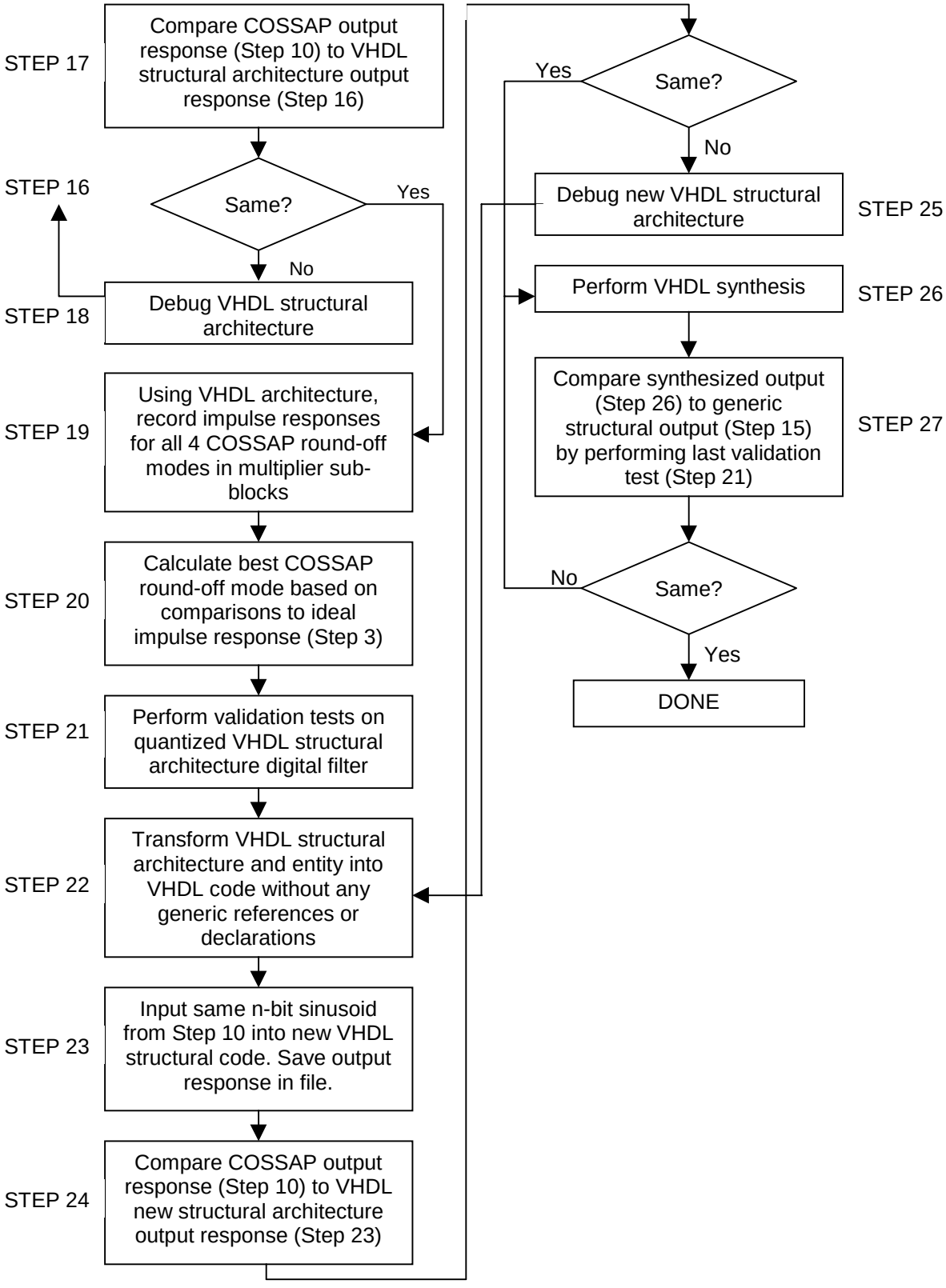
CHAPTER 8: Digital Filter Design Procedures Using High-level Tools

8.1: IIR Digital Filter Design Flowchart

The following flowchart shows how to design n-bit fixed-point IIR digital filters using the high-level modeling tools COSSAP and VHDL. As stated earlier in Chapter 6, the software package MATLAB is the DSP tool used to compute high-precision results. The version of MATLAB used in this research was version 5.3.0.10183 (R11). All computations in MATLAB were done in 64-bit double precision. Performance and analysis are also measured using MATLAB.



(continued)


STEP 17
Compare COSSAP output response (Step 10) to VHDL structural architecture output response (Step 16)
STEP 16
Same?
Yes
No
STEP 18
Debug VHDL structural architecture
STEP 19
Using VHDL architecture, record impulse responses for all 4 COSSAP round-off modes in multiplier sub-blocks
STEP 20
Calculate best COSSAP round-off mode based on comparisons to ideal impulse response (Step 3)
STEP 21
Perform validation tests on quantized VHDL structural architecture digital filter
STEP 22
Transform VHDL structural architecture and entity into VHDL code without any generic references or declarations
STEP 23
Input same n-bit sinusoid from Step 10 into new VHDL structural code. Save output response in file.
STEP 24
Compare COSSAP output response (Step 10) to VHDL new structural architecture output response (Step 23)
Same?
Yes
No
STEP 25
Debug new VHDL structural architecture
STEP 26
Perform VHDL synthesis
STEP 27
Compare synthesized output (Step 26) to generic structural output (Step 15) by performing last validation test (Step 21)
Same?
No
Yes
DONE

8.2: Description of Flowchart Steps

This section provides a full description of each step outlined in the flowchart of the previous section. The digital input signals used to test the performance of all the constructed digital filters throughout this research are quantized to the amplitude range [-1,1). Performance and analysis are done on a total of 4096 samples.

Step 1

As stated numerously throughout this research, for the purpose of having a point of reference for the ideal digital filter, the software package MATLAB is used as an ideal tool for comparison. The MATLAB digital filter will represent the unquantized digital filter while the COSSAP digital filter will represent the quantized digital filter. The former represents the ideal case while the latter is the two's complement, fixed-point case. From a digital filter specification, an IIR digital filter is ideally designed using MATLAB. This step produces the ideal digital filter transfer function, *H(z)*. The ideal filter is in the form of Transposed Direct-Form II Structure of order *n* [Manolakis]. Figure 3.5 provides a graphical illustration of such a network along with its transfer function. This step is not limited to the four following designs but the digital filter could be derived from one of the designs as follows:

- Butterworth
- Chebyshev Type I
- Chebyshev Type II
- Elliptic

The designer should verify if the desired specifications are met for parameters such as cut-off frequency and stopband frequency. For the purposes of analysis, the cut-off and stopband frequency points used throughout this research are -3dB and -40dB, respectively. Within the unit circle, pole/zero placement of the ideal transfer function must also be observed by the designer for the purposes of filter stability. Specifically, all poles must be inside the unit circle for the digital filter to be considered stable [Manolakis].

Step 2

With the availability of the ideal digital filter coefficients from the ideal transfer function, quantization is required to produce an n-bit digital filter. As stated earlier, the coefficients of this n-bit digital filter are represented in fixed-point two's complement format. For the purposes of optimization and round-off noise minimization, if the digital filter order is greater than 2, the ideal digital filter will be broken down into 2^{nd} Order Transposed Direct-Form II sub-blocks [Manolakis]. The coefficients of each respective sub-block will be quantized according to the required n-bit length. Initially, all coefficients will have a truncated quantized format. At the designer's discretion, if the digital filter order is greater than 2, either the parallel or cascade structural realization of the 2^{nd} Order Transposed Direct-Form II sub-blocks described in Chapter 3 is chosen.

To achieve optimum accuracy, the methodology in Table 8-1 is used to obtain an appropriate n-bit representation. Note that the optimum n-bit representation depends on the magnitude of the ideal transfer function coefficients.

Step #	Action
1	Calculate the absolute value of each coefficient (if designing a parallel structure, include the constant *C* if necessary)
2	Find the largest absolute value
3	Calculate the mathematical ceiling of that absolute value
4	Negate that value
5	Calculate the minimum number of bits required to represent that negative integer
6	Calculate the remaining number of bits to be used to represent the fraction

Table 8-1. Methodology to optimally represent n-bits in two's complement fixed-point format.

To illustrate the above methodology, assume that the four coefficients to be represented using 8 bits in two's complement fixed-point format are -0.31871, 2.17328, 1.7014, and -3.19353. Table 8-2 shows the results of each step in the methodology of Table 8-1.

Step #	Action
1	0.31871, 2.17328, 1.7014, 3.19353
2	3.19353
3	4
4	-4
5	3 bits (100_2)
6	8 – 3 = 5 bits

Table 8-2. Tabular results of methodology outlined in Table 8-1.

Since the quantized input into the digital filter is in the range of [-1,1), the methodology outlined in Table 8-1 ensures that the output of the multiplier sub-blocks containing fixed-point coefficients do not overflow (saturate). This is especially critical for the sub-block with the largest multiplier coefficient.

Step 3

Using MATLAB, record and save in a file the impulse response of the ideal transfer function *H(z)*. The impulse response is the time domain description of a filter. It provides information in regards to filter stability. This information is compared to the quantized impulse responses of Step 19.

Steps 4 and 5

Quantitative comparisons of frequency responses between ideal and quantized transfer functions, *H(z)* and $H_q(z)$, are computed. Basically, these are error calculations. Error calculations are computed for the parameters as follows:

- Magnitude
- Phase (in radians)
- Group Delay:
  - The time delay (in samples) that a signal component of frequency ω undergoes as it passes from the input to the output of the system.

The primary performance criterion is the magnitude error calculation.

Steps 6, 7, and 8

Steps 2 through 5 are repeated with the exception that rounded coefficients will be used instead of truncated coefficients. The type that produces the smaller error magnitudes is selected for the design.

Step 9

Using COSSAP's Block Diagram Editor, build the quantized, fixed-point digital filter. Set all adder and multiplier sub-blocks with the correct saturation mode and an initial round-off mode of zero. The optimum saturation mode was found in Chapter 5. The adder and multiplier sub-blocks are in COSSAP's **radix_fxp** library while the delay sub-block is found in the **bittrue** library. From the COSSAP tutorial on the properties of the adder and multiplier sub-blocks in the **radix_fxp** library, the designer knows that fractional coefficients are represented in decimal integer format. To briefly state the COSSAP algorithm for the unique representation, the following example will illustrate. The binary fixed-point representation of the quantized transfer function, $H_q(z)$, is converted to its decimal counterpart. This decimal counterpart consists of decimal coefficients. The fractional parts of these coefficients are then multiplied by 2^x. The variable x is the number of bits calculated in Step 6 in Table 8-1. Table 8-3 illustrates this procedure through example. This example assumes 8 bits are used for quantization; 6 of the 8 bits are used to represent the fractional part of the coefficient; the remaining 2 of the 8 bits are used to represent the integer part of the coefficient; and the quantization type is rounding. This procedure is performed using MATLAB.

Action	Result
Ideal (unquantized) coefficients	1.41704 -0.46091
Fixed-point (quantized) coefficients: binary fraction part	011011 100011
Fixed-point (quantized) coefficients: decimal fraction part	0.421875 -0.453125
COSSAP fractional representation	27 -29

Table 8-3. Tabular results of COSSAP's n-bit fixed-point representation of coefficients for multiplier sub-blocks.

Step 9.1: Digital filters of order 2 or less

If the digital filter being designed is no greater than order 2, the designer has to choose how many of the *n* bits must be allocated to represent the integer portion of the outputs of the multiplier sub-blocks. The rule of thumb used throughout this research stems from Steps 1 through 5 of Table 8-1. Using Figure 3.7 as an example, with the exception of the top-most adder sub-block, in which the output both is *y(n)* and serves as input to the right-hand multiplier sub-blocks, the number calculated in Step 5 is the number used for integer representation for the other adder sub-blocks. This number is used for both input and output. The aforementioned top-most adder sub-block contains the same number for integer representation but only for the inputs. The output is assigned 1 bit to represent the integer portion. This is because the input signal into the digital filter is quantized to the amplitude range of [-1,1). If a passband signal of amplitude range [-1,1) is input into the system, an output of amplitude range of [-1,1) is expected. This is the fundamental reason behind assigning 1 bit to the integer portion of the particular adder sub-block. Assigning 1 bit to the output of the top-most adder sub-block mathematically covers the quantized range of [-1,1). It is repeated here at this point that it is assumed that the designer has read the COSSAP tutorial on the properties of the adder and multiplier sub-blocks in COSSAP **radix_fxp** library.

Step 9.2: Parallel-structure for high-order filters

If the designer opts for a parallel structure for a digital filter greater than order 2, the designer must choose a suitable number of bits to optimally represent the integer portion of the adder sub-blocks. The rule of thumb in this case is to calculate the peak magnitude response of each transfer function. Include

the constant *C* if necessary. Next, calculate the mathematical ceiling of each peak magnitude number. Finally, the designer must find the largest mathematical ceiling, negate that value, and calculate the number of bits required to represent that number. This calculated number is now used for the adder sub-blocks of all the transfer function sections for both input and output. The final summation adder sub-block is assigned 1 bit to represent the integer portion at the output for reasons previously outlined in Step 9.1.

Step 9.3: Cascade-structure for high-order filters
If the designer opts for a cascade realization for a high-order filter greater than order 2, the designer must apply the same procedure outlined in Step 9.2 except that special consideration must be paid to the transfer function sections after the first stage. For the parallel structure in Step 9.2, the input into all the transfer function sections is in the amplitude range of [-1,1). This is true only for the first stage in a cascade structure, and not necessarily true for the succeeding stages. To remedy this situation, the designer could choose one of two methodologies as follows:

Step 9.3.1: Cascade-structure Methodology #1
The designer should follow Steps 1 through 5 of Table 8-1 for each transfer function section. The number calculated from Step 5 for each section is used for the integer representation for all the adder sub-blocks of that particular cascade section. For the top-most adder sub-block in the last cascade stage, the output is assigned 1 bit for reasons previously outlined in Step 9.1. Results of Chapter 10 will provide a better illustration of this concept in terms of application.

Step 9.3.2: Cascade-structure Methodology #2
The designer should input a passband signal into the overall digital filter transfer function*, H(z)*. Calculate the resulting minimum and maximum values out of each cascade stage. Of these values, calculate the absolute values; calculate the mathematical ceiling of each value; calculate the greatest of the mathematical ceiling computations; negate this value; and calculate the number of bits needed to represent this negated value. The first part of this methodology is complete. The next part entails all of the coefficients of each cascade section. For each cascade section, follow Steps 1 through 5 of Table 8-1. At this point, if there are *n* cascade stages, there should be *n* answers based on Step 5. Including the result from the first part of this methodology, there are now *n+1* answers. The designer should calculate the largest number of these answers. This number calculated is to be used for the integer representation for all the adder sub-blocks in the cascade structure. As always, for the top-most adder sub-block in the last cascade stage, the output is assigned 1 bit for reasons previously outlined in Step 9.1. Results of Chapter 10 will provide a better illustration of this concept in terms of application.

Step 10
An n-bit input sinusoid will serve as a performance test for the quantized digital filter designed in COSSAP. The sinusoid is required to have a passband frequency. This passband frequency serves as a means to verify whether or not the fixed-point adder and multiplier sub-blocks are functioning optimally. The passband frequency also tests to see if the filter can handle an input signal of quantized amplitude range [-1,1) without any clipping at the positive or negative output peaks. The designer should use the instance *SIN_GEN_TBL* in the *DSP* library in the Block Diagram Editor. Configuring the signal generator sub-block requires that the designer do the following:

NumberOfItems = *sampling frequency* (Eqn. 8.1)
NumberOfPeriods = *signal frequency* (Eqn. 8.2)
Amplitude = 1 (Eqn. 8.3)

The variables *NumberOfItems* and *NumberOfPeriods* must have values. These variables are equivalent to the sampling frequency and input signal frequency, respectively. Reading the COSSAP online documentation of the signal generator *SIN_GEN_TBL*, this sub-block produces an output according to the equation as follows:

$$\text{Output} = \sin(2\pi * \textit{NumberOfPeriods} / \textit{NumberOfItems} * k) \qquad \text{(Eqn. 8.4)}$$

The variable *k* is a non-negative integer starting from 0. Through careful mathematical manipulation, the term *2π*NumberOfPeriods/NumberOfItems*k* is mathematically equivalent to the term *2π*f/Fs*k*. This is the basis for equations 8.1 and 8.2. It is noted here that the value of π differs slightly in COSSAP and MATLAB. Calculating to 20 significant places, the values of π are as follows:

COSSAP: π = 3.14159265358979433846
MATLAB: π = 3.14159265358979311600

As stated earlier, analysis is performed on a total of 4096 samples.

On an extremely important note, it is stated here that, regardless if the passband signal input into the filter built in COSSAP's BDE meets the designer's requirements, digital signals in both the transition and stopband regions must be inputted into the filter. As always, MATLAB is used as the tool of comparison for its research. These signals test to see how nonlinear effects are handled within the adder and multiplier sub-blocks. If the filter output is not what is expected, the designer can apply the following remedies depending on whether a parallel or cascade structure was designed.

Step 10.1: Parallel-structure Remedy
The designer can try assigning more bits to represent the integers in the adder sub-blocks. These integer assignments should be done for both input and output sections of the fixed-point adders. The only integer fixed-point section that should *not* be changed is the output integer section of the final summation adder sub-block. This part of the adder sub-block keeps its 1 bit output integer assignment for reasons outlined in Step 9.1.

If this measure does not fix the problem, the designer should consider re-designing the filter using more bits. For example, the designer should consider going from a 20-bit design to a 24-bit design. If the filter still cannot be realized using the measure prescribed, the designer could try other ad-hoc methods or implementing the digital filter as a cascade structure. If nothing works, do not go to Step 11.

Step 10.2: Cascade-structure Remedy
Because of the ordering methodology of this cascade structure outlined in section 3.4.2, the designer should first attempt increasing the number of bits by 1 for integer representation in the adder sub-blocks of the last 2 cascade stages. If the output results still do not meet the designer's requirements, the designer is still allowed to increase the integer bit representation by another bit. At the designer's discretion, if the output results still do not meet design specifications, the designer should continue the incremental bit increase to a point where a loss of quantized fractional accuracy is not drastic.

As with the parallel structure, if the problem is not fixed, the designer should consider re-designing the filter using more bits. If the digital filter cannot be realized using the measures prescribed, the designer could try other ad-hoc methods. If nothing works, do not go to Step 11.

Step 11
Run the utility program **xvcg** to produce a VHDL-generated code for the n-bit fixed-point digital filter. This is found in COSSAP's Block Diagram Editor (BDE) under *Tools* -> ***xvcg*** option.
Steps 12 through 14
Input the same digital input signal from Step 10 into the VHDL-generated code of Step 11 and verify that both outputs are 100% identical. If outputs are dissimilar, the designer must perform debugging. Although both models should be rechecked (COSSAP BDE and VHDL-generated code), the designer should first look at the VHDL-generated code and verify that the VHDL testbench file used is correct.

Step 15
Convert the VHDL-generated code into a generic VHDL structural architecture model. This structural code must resemble the Transposed Direct-Form II structure. Include generic parameters for the purposes of providing the designer the ability to pass round-off mode settings from the VHDL testbench file. This style of coding using generics saves the designer considerable time from having to go into the VHDL-generated code itself and manually finding the multiplier sub-blocks and changing their round-off

mode settings. This is especially true when it comes to very high-order digital filters. Chapter seven provides a more detailed discussion.

Steps 16 through 18
The same input signal from Step 10 serves as input into the generic VHDL structural architecture. The output from this generic structural code is compared to the output of Step 10. As before, the designer should verify that both outputs are 100% identical. If dissimilar, debugging is in order. At this point, if debugging is necessary, the error is more likely to be in the generic VHDL structural architecture. The reason why debugging is not initially performed on the original VHDL-generated code is because it has already been debugged and rechecked in Step 14 of the flowchart in section 8.1. The designer would check to see if certain multiplier and adder sub-blocks were configured correctly.

Step 19
With the VHDL structural architecture with generics, four different impulse responses from four different COSSAP round-off modes available in the multiplier sub-blocks are recorded and observed. Using the ideal impulse response of the digital filter designed in MATLAB, each quantized impulse response will have its round-off noise power calculated using Equation 5.1 and the COSSAP round-off mode that produces the smallest round-off noise power will be performance criterion for the designed digital filter. Other factors to be observed by the designer are the dead-band range (due to limit-cycle oscillations) and the power spectral density of the error and filter output. Power spectral density is the distribution of signal power as a function of frequency. The signal is assumed to be periodic.

Step 20
Calculate the best COSSAP round-off mode based on comparisons to the ideal impulse response from Step 3. As stated in Step 19, the smallest round-off noise power is the performance criterion.

Step 21
Perform validation tests on the generic VHDL structural architecture using the ideal digital filter as a source of comparison. These tests consist of output responses to input signals in the passband, transition band, and stopband regions of the frequency spectrum of the digital filter. The performance criterion is the round-off noise power. These tests are performed to confirm the legitimacy of the best round-off mode calculated in Step 20. These tests are to be performed on all four COSSAP round-off modes for the multiplier sub-blocks. Depending on the round-off noise power results, the designer could choose to either use the established round-off mode calculated in Step 20 or another round-off mode that has potentially better round-off noise power results.

Step 21.1: Validation Test #1
- Input digital signal in the passband region.
- Input digital signal in the transition band region.
- Input digital signal in the stopband region.

Step 21.2: Validation Test #2
Input a digital signal containing 2 frequencies. This signal is composed as follows:
- One low-power frequency in the passband region.
- One high-power frequency in the stopband region.

The high-power digital signal has an amplitude strength 10 times that of the low-power digital signal. This test establishes how efficiently the quantized filter removes strong signals (noise) in the stopband region.

Step 22
Transform the generic VHDL structural architecture to make it synthesis-ready. This step requires the designer to remove all generic declarations in both the top-level and low-level components. Removing generics require the designer to “hardwire” necessary integer values into the multiplier and adder sub-blocks. Results from Chapter 9 will provide a better demonstration through example.

Steps 23 through 25

The same input signal from Step 10 will serve as input into the new generic-free VHDL structural architecture. The output of this structural code will be compared to the output of Step 10. As before, the designer should verify that both outputs are 100% identical. If dissimilar, debugging is in order. At this point, if debugging is necessary, the error is more likely to be in the generic-free VHDL structural architecture. The reason why debugging is not initially performed on the original VHDL-generated code is because it has already been debugged and rechecked in Step 14 of the flowchart.

Step 26

Perform VHDL synthesis. The designer should use script files to impose certain parameters and constraints where applicable. Timing constraints are the most important parameters.

Step 27

Test the synthesized digital filter using validation test #2 from Step 21.2. The synthesized output is then to be compared to the generic VHDL structural architecture result. These two results must have the same round-off mode established in Step 21. The two results are expected to be exactly the same. If there are any errors, the designer should first look at the script files used for synthesis as the primary source.

CHAPTER 9: Results of IIR Digital Filter Design Methodology

This chapter deals with the implementation of the flowchart methodology of Chapter 8. These results are based on a 16-bit 3rd order Butterworth fixed-point digital filter with a sampling frequency of 10kHz, a cut-off frequency of 1.5kHz (-3dB point), and a stopband frequency point of -40dB. Results are based on 4096 samples. Because the filter order is greater than 2, both parallel and cascade structures were investigated.

9.1: 16-bit Butterworth Lowpass Filter Design

The ideal overall transfer function of this filter is the following:

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3}}{1 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3}}$$

The MATLAB floating-point representation of the coefficients, taken to 14 significant places, are as follows:

b_0 = 0.04953299635725
b_1 = 0.14859898907176
b_2 = 0.14859898907176
b_3 = 0.04953299635725
a_1 = -1.16191748367173
a_2 = 0.69594275578965
a_3 = -0.13776130125989

Figures 9.1 through 9.4 show the frequency response of the above ideal transfer function. The frequency response data consists of magnitude, phase, and group delay. As stated in section 8.2, group delay is defined as the time delay (in samples) that a signal component of frequency ω undergoes as it passes from the input to the output of the system. It is the derivative of the phase with respect to the frequency. This is not to be confused with the phase delay which is defined as the phase divided by the frequency [Gold]. Group delay is a convenient measure of the linearity of the phase. Because this design is a Butterworth filter, the passband and stopband regions are maximally flat. Due to the frequency points mentioned in the beginning of this chapter, the resulting stopband frequency (-40dB point) is 3.726kHz. The figures show the frequency response from 0Hz to the Nyquist frequency. The Nyquist frequency is typically defined as half the sampling frequency, f_s. The Nyquist frequency, in this case, is 5kHz. The mentioning of Nyquist frequency introduces the concept of *aliasing*. If a bandlimited signal of frequency f_0 is sampled at less than half the sampling frequency, a sampled high-frequency component could take the identity of a low-frequency component [Oppenheim3]. All figures in this research are plotted from 0Hz to the Nyquist frequency.

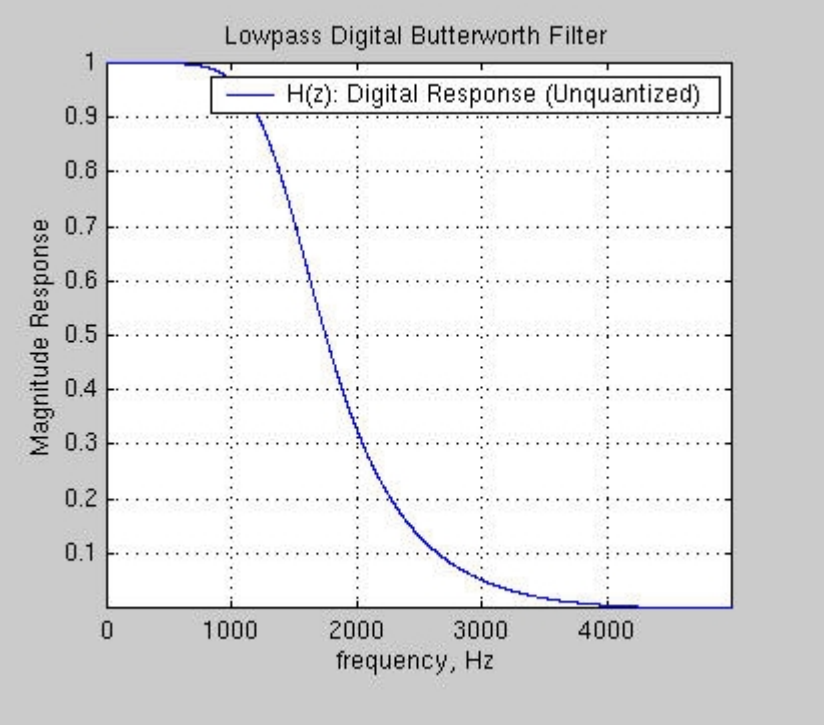


Fig. 9.1. MATLAB magnitude response of lowpass Butterworth filter.

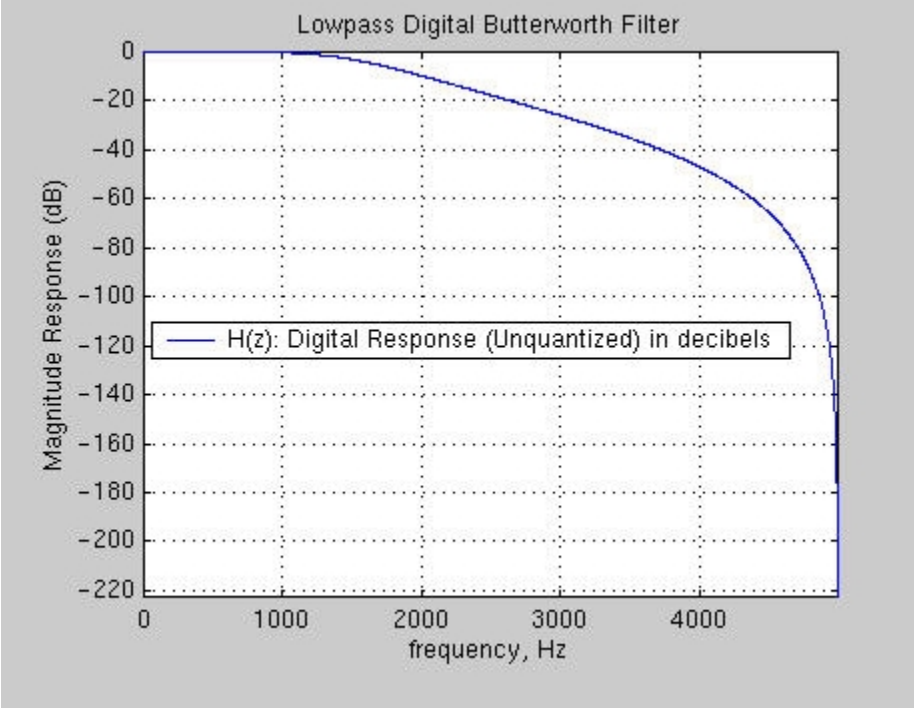


Fig. 9.2. MATLAB magnitude response of lowpass Butterworth filter (dB).

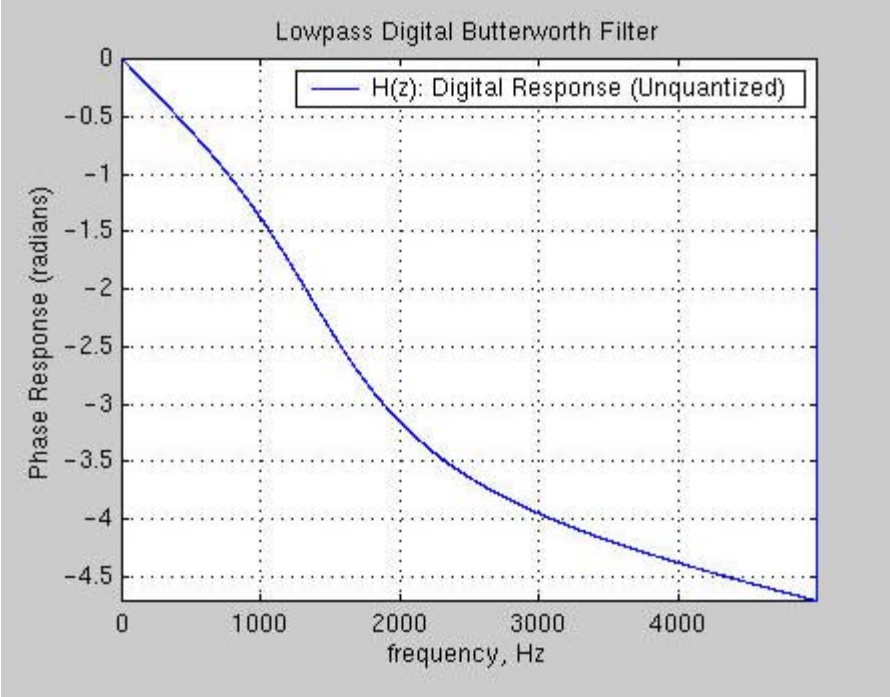


Fig. 9.3. MATLAB phase response of lowpass Butterworth filter (radians).

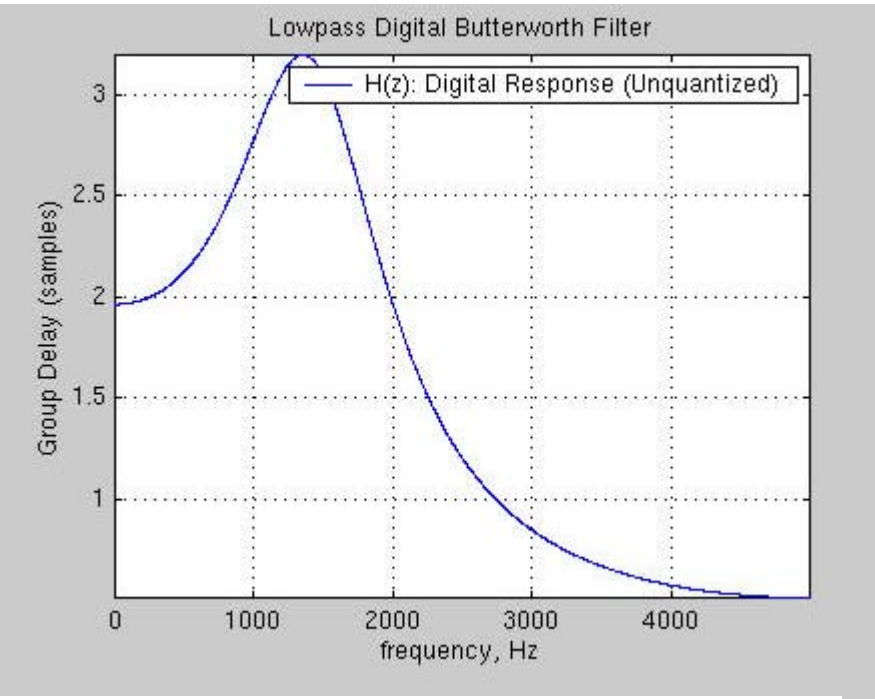


Fig. 9.4. MATLAB group delay response of lowpass Butterworth filter.

9.1.1: Parallel Structure Implementation of Butterworth Lowpass Filter

Based on Equation 3.4 from Chapter 3, the ideal transfer function sections of a parallel structure representation of the ideal overall transfer function are as follows:

$$H(z) = H_1(z) + H_2(z) \qquad \text{(Eqn. 9.1)}$$

$H_1(z)$: (Eqn. 9.2)

$b_0 = -0.96729224234444$
$b_1 = 0.56856011470698$
$b_2 = 0.00000000000000$
$a_1 = -0.83699778743883$
$a_2 = 0.42398568894741$

$H_2(z)$: (Eqn. 9.3)

$b_0 = 1.01682523870169$
$b_1 = 0.11682704781905$
$a_1 = -0.32491969623291$

The constant *C*, in this case, is zero. The above transfer function sections are represented in MATLAB's floating-point precision. These sections will next be transformed to fixed-point notation using the methodology outlined in Table 8-1 in section 8.2. Using the truncation method, quantizing these sections to the prescribed 16 bits with 2 bits for integer representation and 14 bits for fractional representation yields the following:

$H_{1qt}(z)$: (Eqn. 9.4)

$b_0 = -0.9672851562500000$
$b_1 = 0.5685424804687500$
$b_2 = 0.0000000000000000$
$a_1 = -0.8369750976562500$
$a_2 = 0.4239501953125000$

$H_{2qt}(z)$: (Eqn. 9.5)

$b_0 = 1.0167846679687500$
$b_1 = 0.1168212890625000$
$a_1 = -0.3248901367187500$

Figures 9.5 and 9.6 show the magnitude responses of both the unquantized and quantized transfer function sections using the truncation method. These figures graphically show no discernible difference of both transfer function sections.

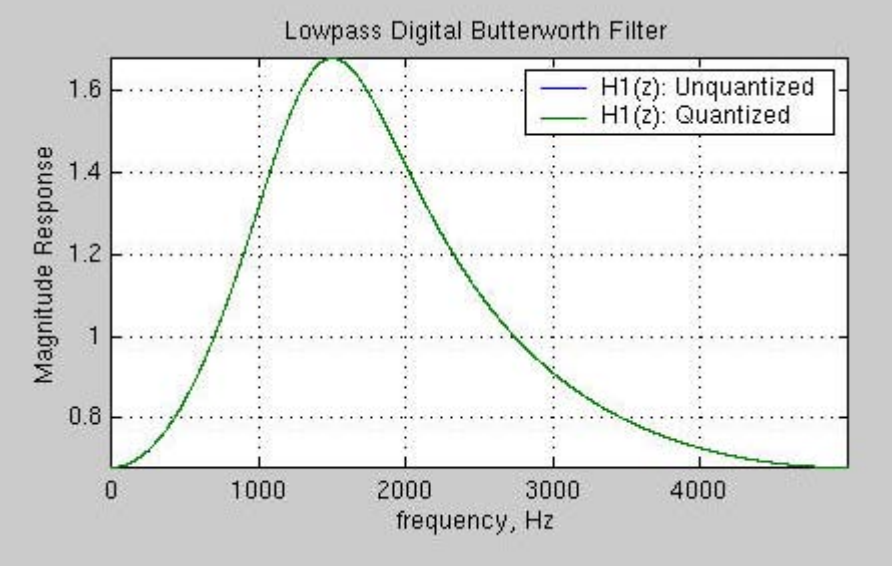


Fig. 9.5. MATLAB plot of magnitude response of transfer function section $H_1(z)$.

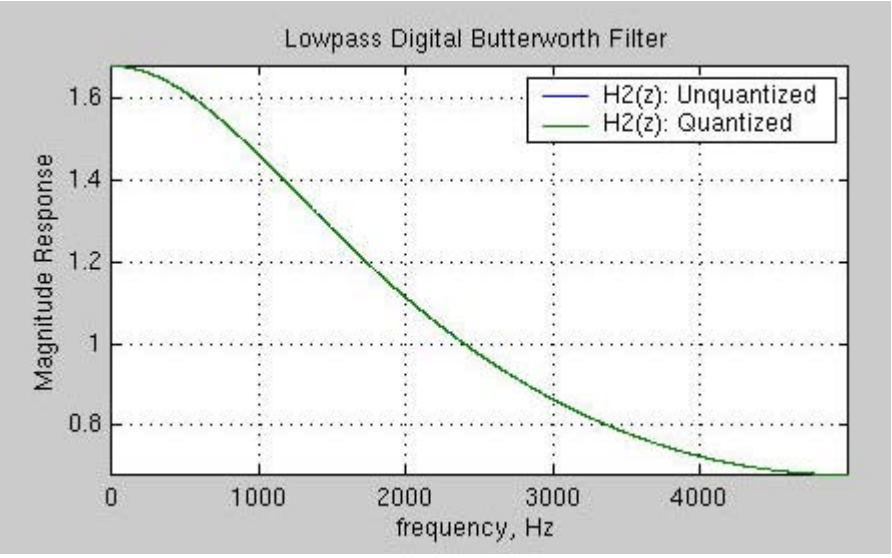


Fig. 9.6. MATLAB plot of magnitude response of transfer function section $H_2(z)$.

Equations 9.6 and 9.7 show the quantized transfer function sections using the rounding method.

$H_{1qr}(z)$: (Eqn. 9.6)

$b_0 = -0.9672851562500000$
$b_1 = 0.5685424804687500$
$b_2 = 0.0000000000000000$
$a_1 = -0.8369750976562500$
$a_2 = 0.4240112304687500$

$H_{2qr}(z)$: (Eqn. 9.7)

b_0 = 1.0168457031250000
b_1 = 0.1168212890625000
a_1 = -0.3248901367187500

Table 9-1 illustrates the magnitude response error between both the truncated and rounding methods when compared to the ideal. Equation 9.8 is the formula used for magnitude error calculation [Bailey]. Error calculations are performed up to the Nyquist frequency.

$$H_e(z) = |H(z)| - |H_q(z)| \qquad \text{(Eqn. 9.8)}$$

Quantization Type	$H_{1e}(z)$	$H_{2e}(z)$
Truncated	0.00011580424293	0.00014215218294
Rounded	0.00008192099318	0.00005174444403

Table 9-1. Magnitude error calculation.

Based on the results of Table 9-1, the rounding method is the better of the two quantization types. Using the rounded coefficients, the transformation to COSSAP coefficient representation is the next step in the design methodology cycle. These coefficients are the multiplier coefficients used in the multiplier sub-blocks. Table 9-2 illustrates the results of this transformation.

$H_{1qr}(z)$		$H_{2qr}(z)$	
b0_COSSAP = 0	-15848	b0_COSSAP = 1	276
b1_COSSAP = 0	9315	b1_COSSAP = 0	1914
b2_COSSAP = 0	0	a1_COSSAP = 0	5323
a1_COSSAP = 0	13713		
a2_COSSAP = 0	-6947		

Table 9-2. COSSAP results of multiplier coefficient transformation.

Figure 9.7 shows the schematic of the parallel structure using COSSAP's Block Diagram Editor (BDE). Instances M5, M7, M8, M9, M10, M11, and M12 are the multiplier sub-blocks. Instances M16, M17, M22, M23, M24, and M27 are the adder sub-blocks. Instances M14, M20, and M25 are the delay sub-blocks. Figure 9.8 shows that according to Step 9.2 in Chapter 8, the adder sub-blocks with the exception of the final summation sub-block, instance M23, are configured in COSSAP for a 16-bit fixed-point representation of 2 bits for integer representation and 14 bits for fractional representation.

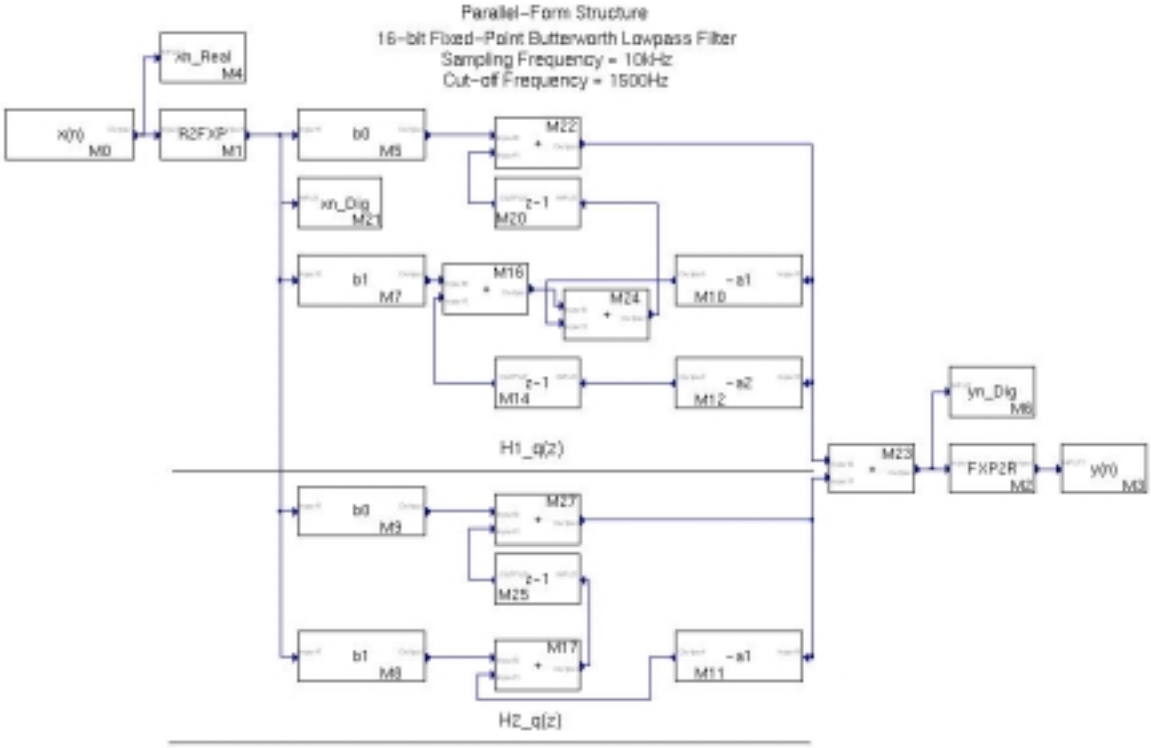


Fig. 9.7. COSSAP schematic of parallel structure.

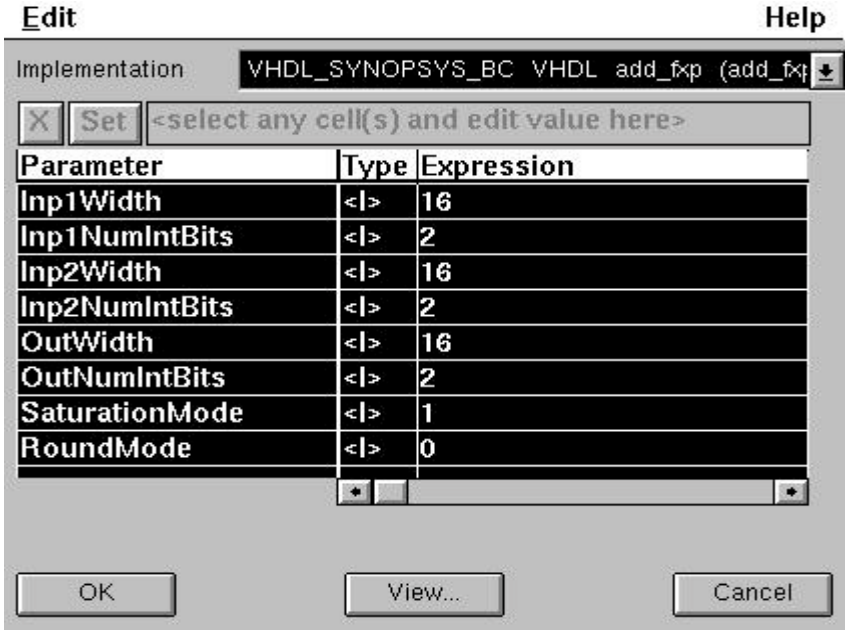


Fig. 9.8. COSSAP configuration of adder sub-blocks (excluding final summation sub-block)

The final summation adder sub-block, instance M23, is configured differently in terms of its 16-bit fixed-point output. The fixed-point output is assigned 1 bit for integer representation for reasons expressed in Step 9.1 in section 8.2. Figure 9.9 shows the COSSAP configured format of this particular adder sub-block.

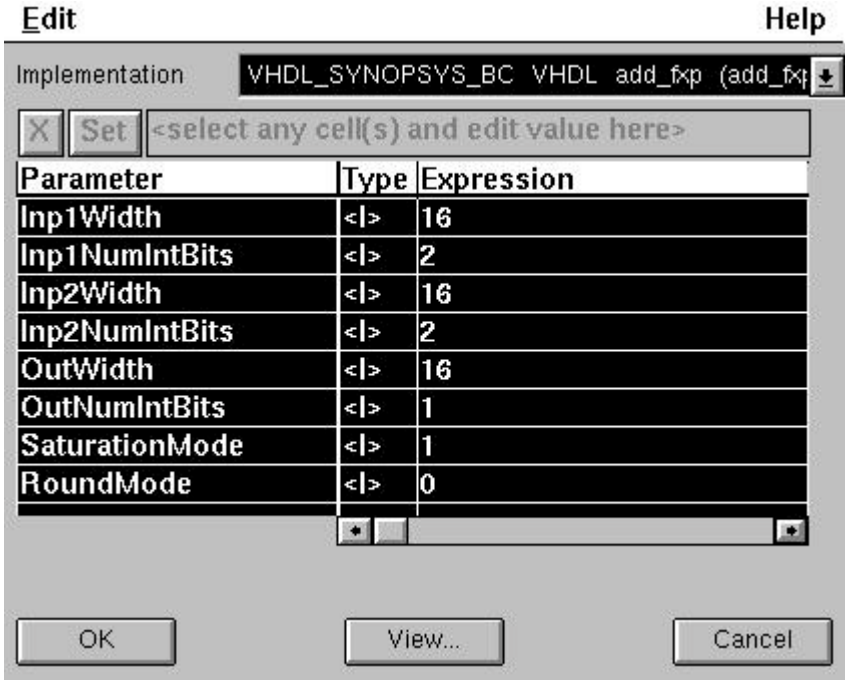


Fig. 9.9. COSSAP configuration of final summation adder sub-block.

Figure 9.10 shows an example of how the coefficients in the multiplier sub-blocks are configured. This figure shows the COSSAP configuration of the multiplier coefficient b_0 of the transfer function section $H_{1qr}(z)$ (instance M5).

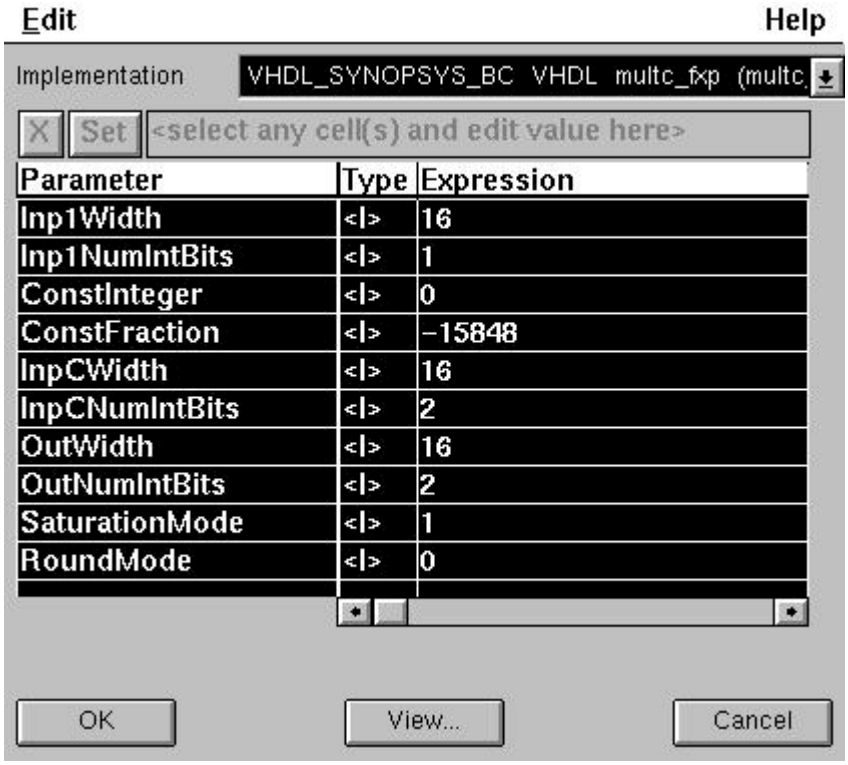


Fig. 9.10. COSSAP configuration of multiplier coefficient b_0.

Instances M0, M6, and M21 in Figure 9.7 are the required components for signal generator and storage to be used for future analysis. Instance M0 is the sinusoid generator. Instances M21 and M6 record the 16-bit fixed-point input and output, respectively. Instance M1 quantizes the real number input into a 16-bit fixed-point format. The amplitude range is [-1,1). Figure 9.11 shows how this instance is configured in COSSAP. The following statements are based on Equation 8.1 and 8.2. A passband signal of 100Hz is input into this fixed-point design. Using the prescribed sampling frequency of 10kHz, Figure 9.12 shows the COSSAP configuration of instance M0.

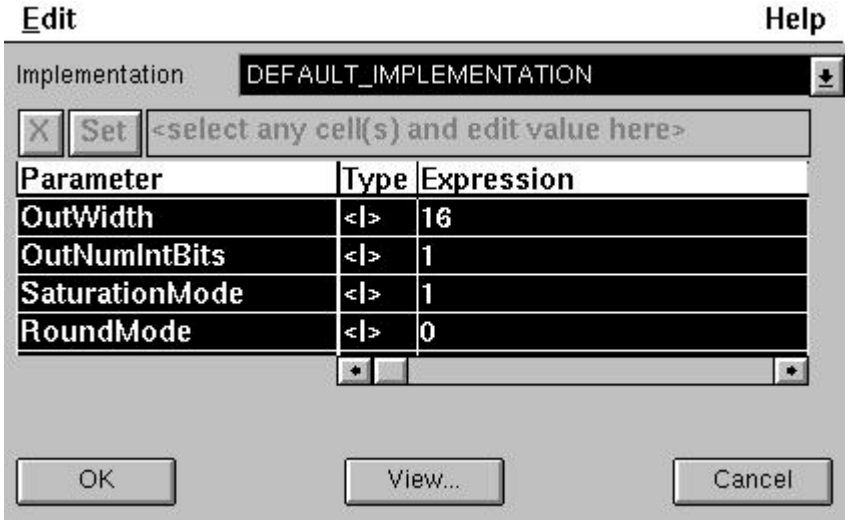


Fig. 9.11. COSSAP configuration of 16-bit quantizer.

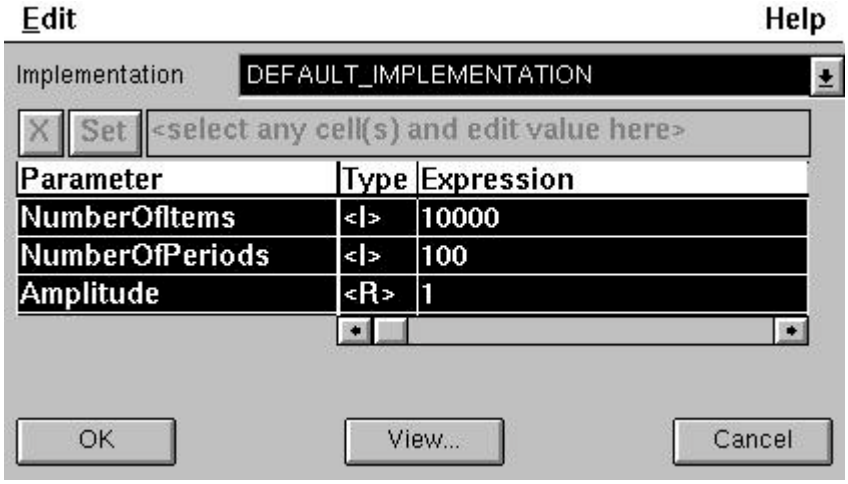


Fig. 9.12. COSSAP configuration of signal generator.

Figures 9.13 and 9.14 show the input/output waveforms of MATLAB and COSSAP, respectively. These results are as expected in that they are nearly identical considering an ideal-to-quantized comparison. The quantized input signal has the correct passband frequency of 100Hz. Using this input, the MATLAB waveform is the ideal output while the COSSAP waveform is the quantized output. Comparison is performed against the ideal transfer function *H(z)*. Transition band and stopband input signals are next input into the COSSAP filter and produce the expected results at the output.

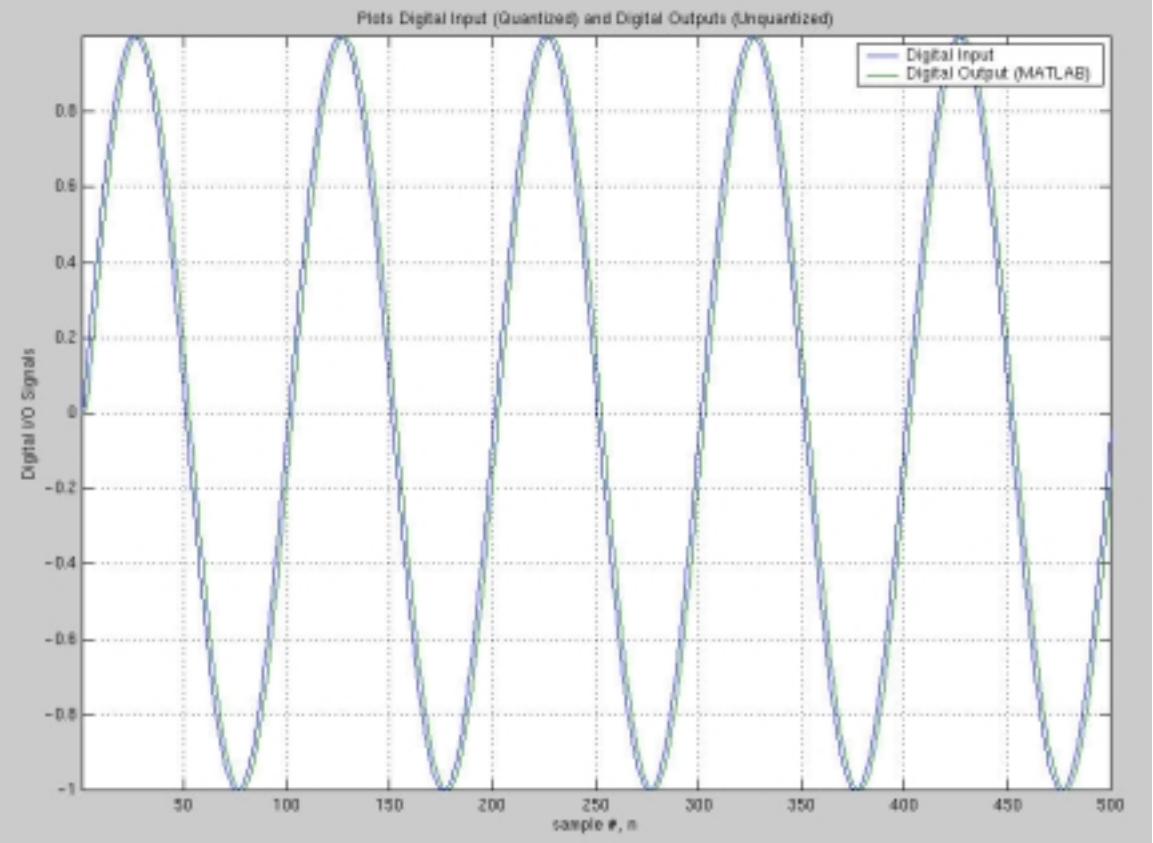


Fig. 9.13. MATLAB response of digital passband input signal.

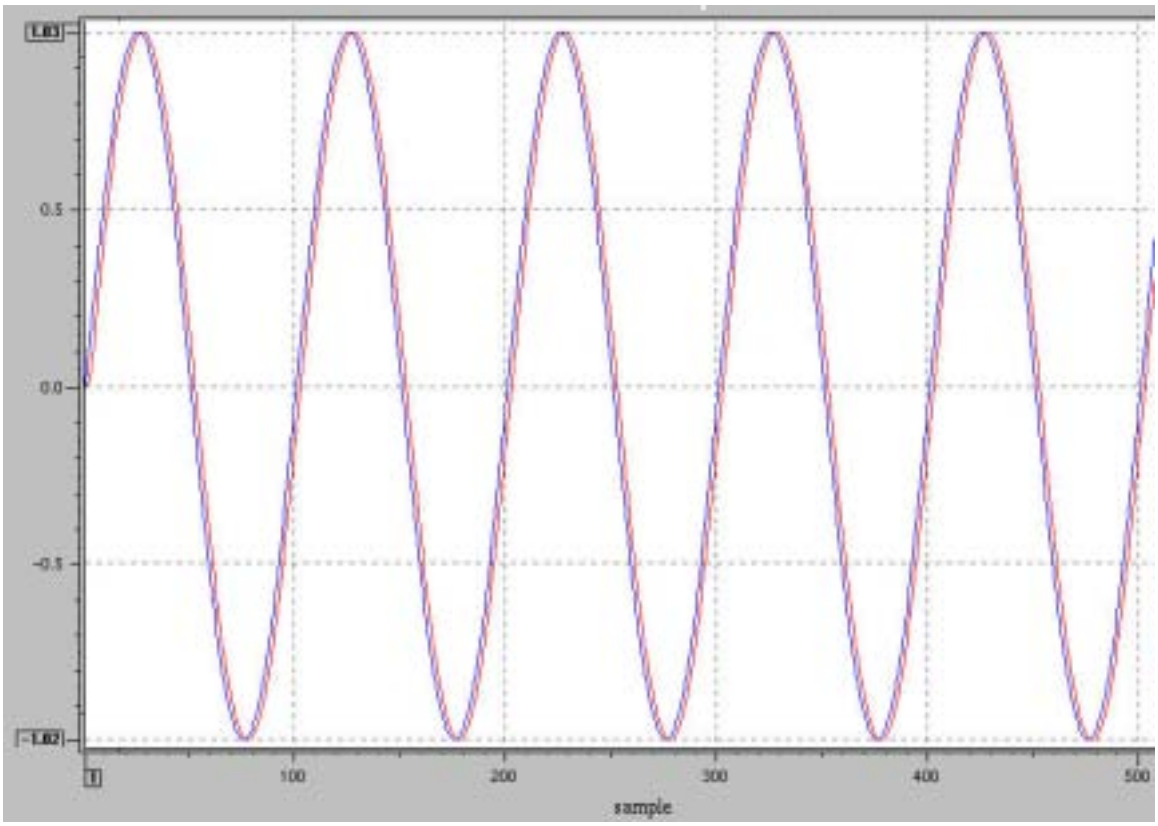


Fig. 9.14. COSSAP response of digital passband input signal.

Figure 9.15 shows the schematic conversion of Figure 9.7 required to generate VHDL code. The only instances remaining in this block diagram are multiplier, adder, and delay sub-blocks.

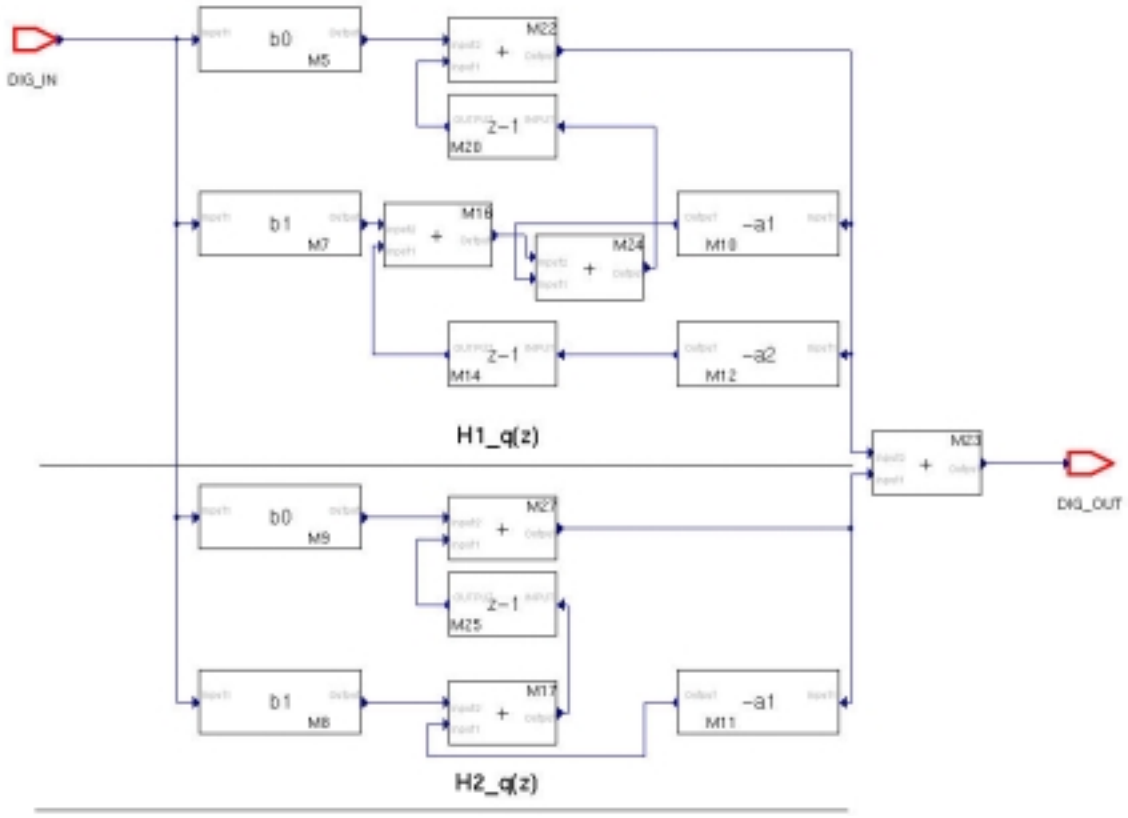


Fig. 9.15. COSSAP schematic setup to generate behavioral VHDL model.

Figure 9.16 shows the impulse response of the ideal overall transfer function *H(z)* and the quantized impulse response of the transfer function of the parallel structure. This result is based on COSSAP's *RoundMode = 0* for the multiplier sub-blocks. There is no exponential decay to zero in the COSSAP/VHDL model. The 16-bit fixed-point output of the COSSAP/VHDL digital filter remains at a fixed value. Table 9.3 shows the round-off noise power results for the four COSSAP round-off modes used for the multiplier sub-blocks. This table also shows the dead-band range (or resulting fixed value at the output), due to quantization, associated with each round-off mode as well as power spectral density information. This information provides the frequency at which the peak power spectral density is reached. In the MATLAB ideal model, the peak power spectral density is -130.405917dB at 1311.035156Hz.

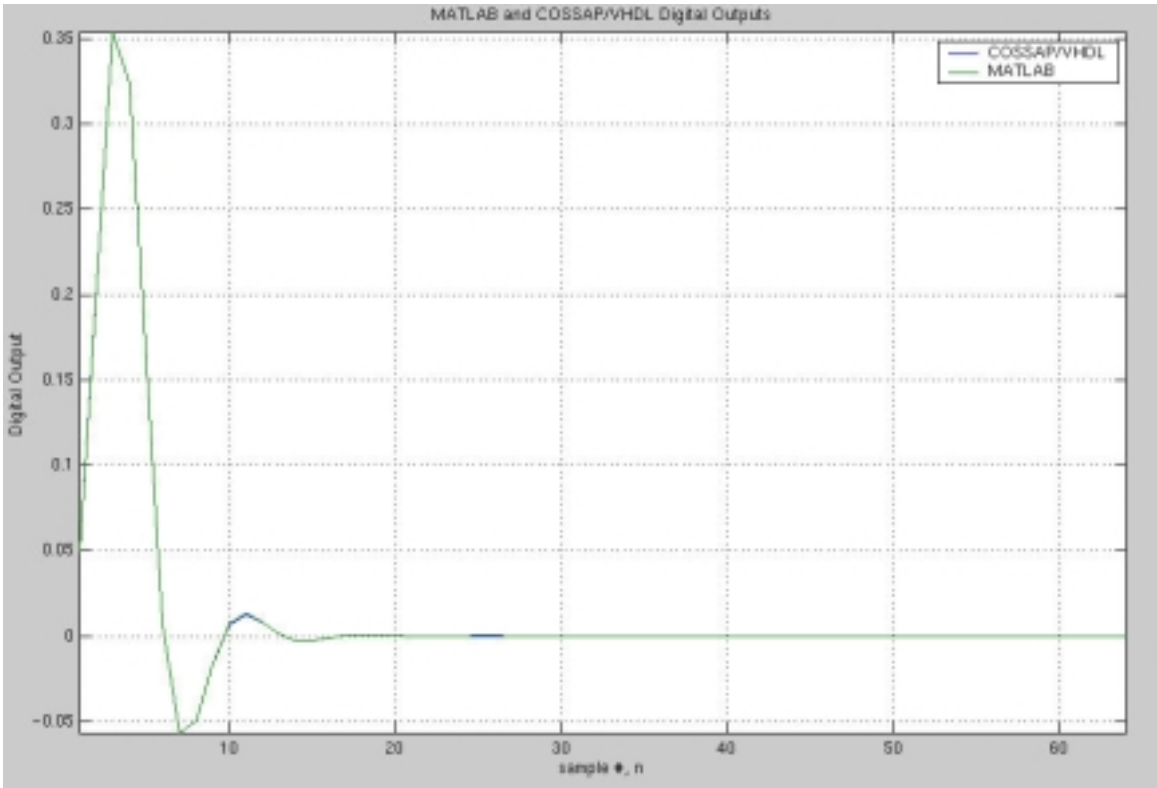


Fig. 9.16. Impulse response of ideal transfer function and quantized transfer function for *RoundMode = 0*.

COSSAP RoundMode	P_e	Dead Band Range (Fixed Value Output)	COSSAP/VHDL Peak PSD
0	$3.76450121076 \times 10^{-9}$	$-6.103515625 \times 10^{-5}$	-49.925002 dB at 0.000000 Hz
1	5.551×10^{-17}	0.00	-130.404258 dB at 1303.710938 Hz
2	5.551×10^{-17}	0.00	-130.404258 dB at 1303.710938 Hz
3	2.448042×10^{-14}	0.00	-130.479798 dB at 1298.828125 Hz

Table 9-3. Round-off noise power results for impulse response of COSSAP multiplier round-off modes.

Based on Table 9-3, *RoundMode = 1* and *RoundMode = 2* produce the best results in terms of smallest error. Figures 9.17 through 9.20 show the power spectral density of the impulse response of COSSAP round-off modes.

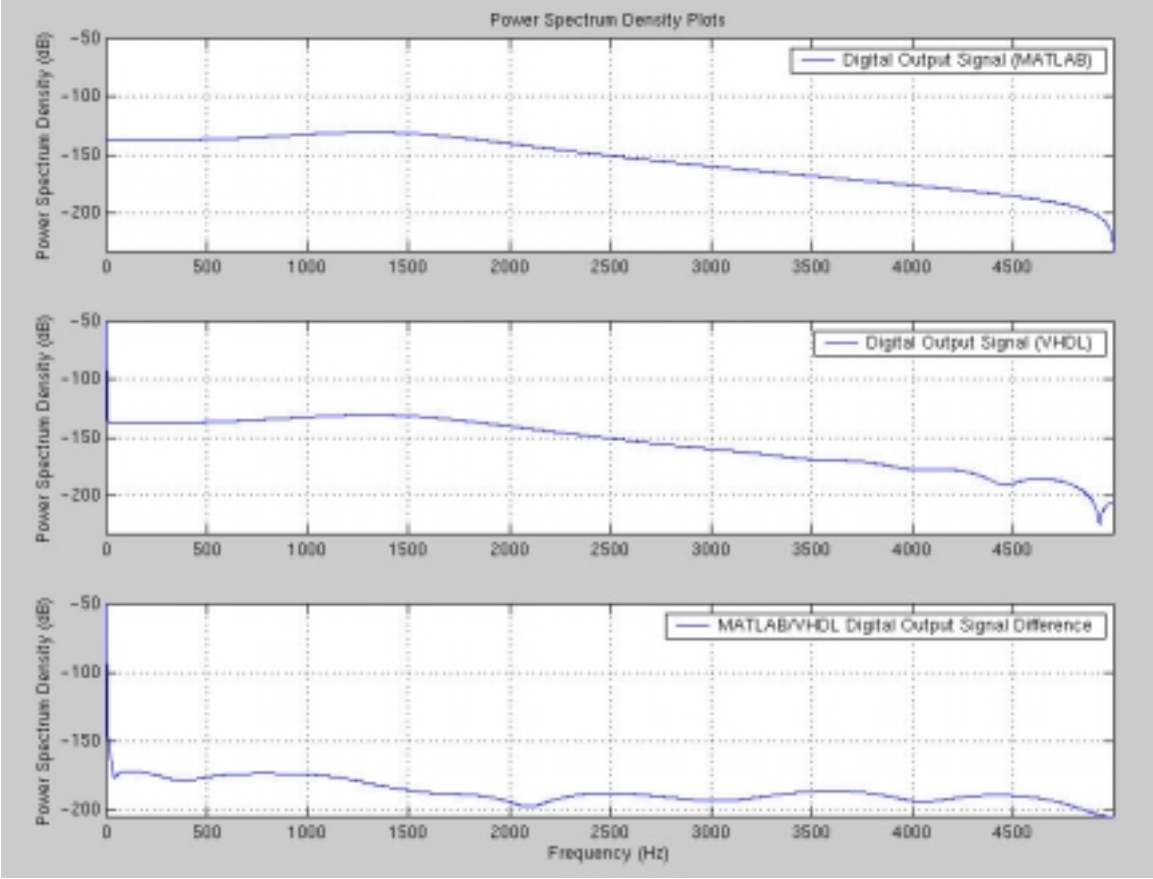


Fig. 9.17. Power spectral density results of impulse response of COSSAP *RoundMode = 0*.

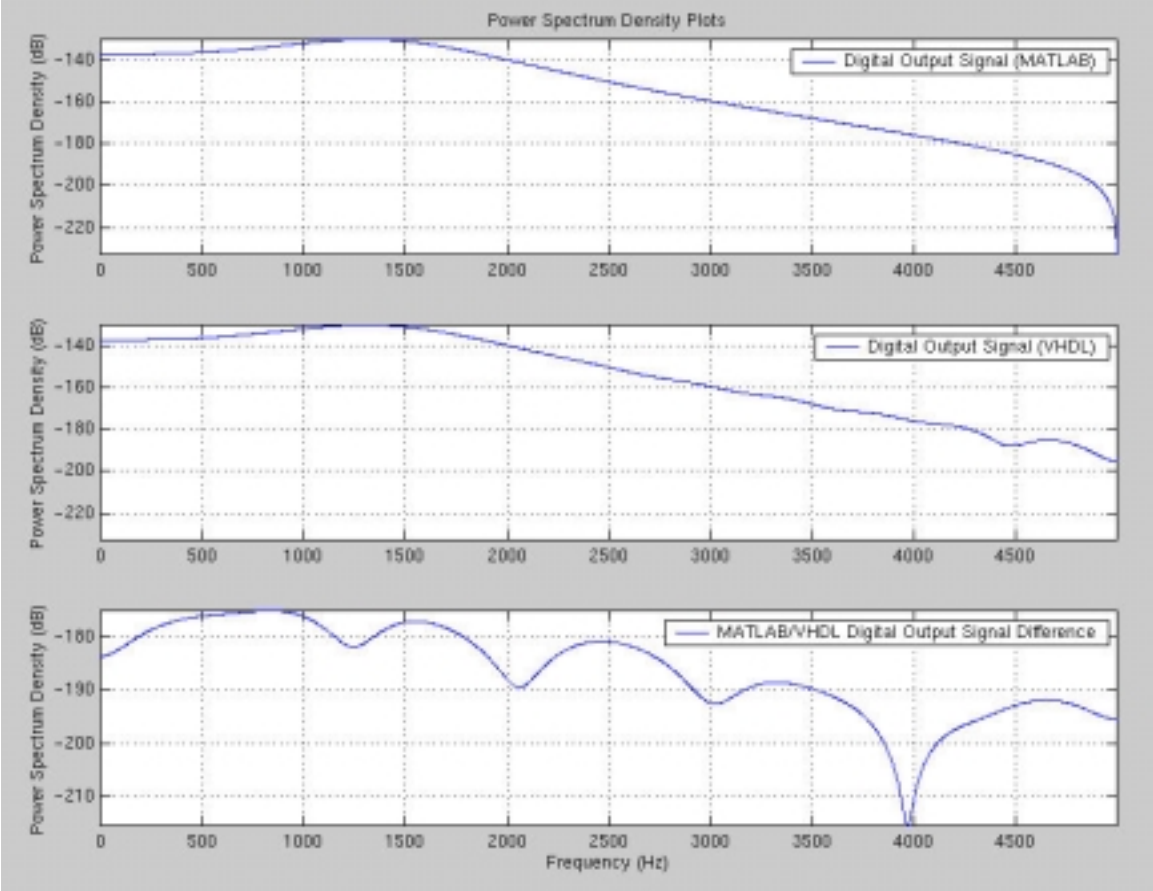


Fig. 9.18. Power spectral density results of impulse response of COSSAP *RoundMode = 1*.

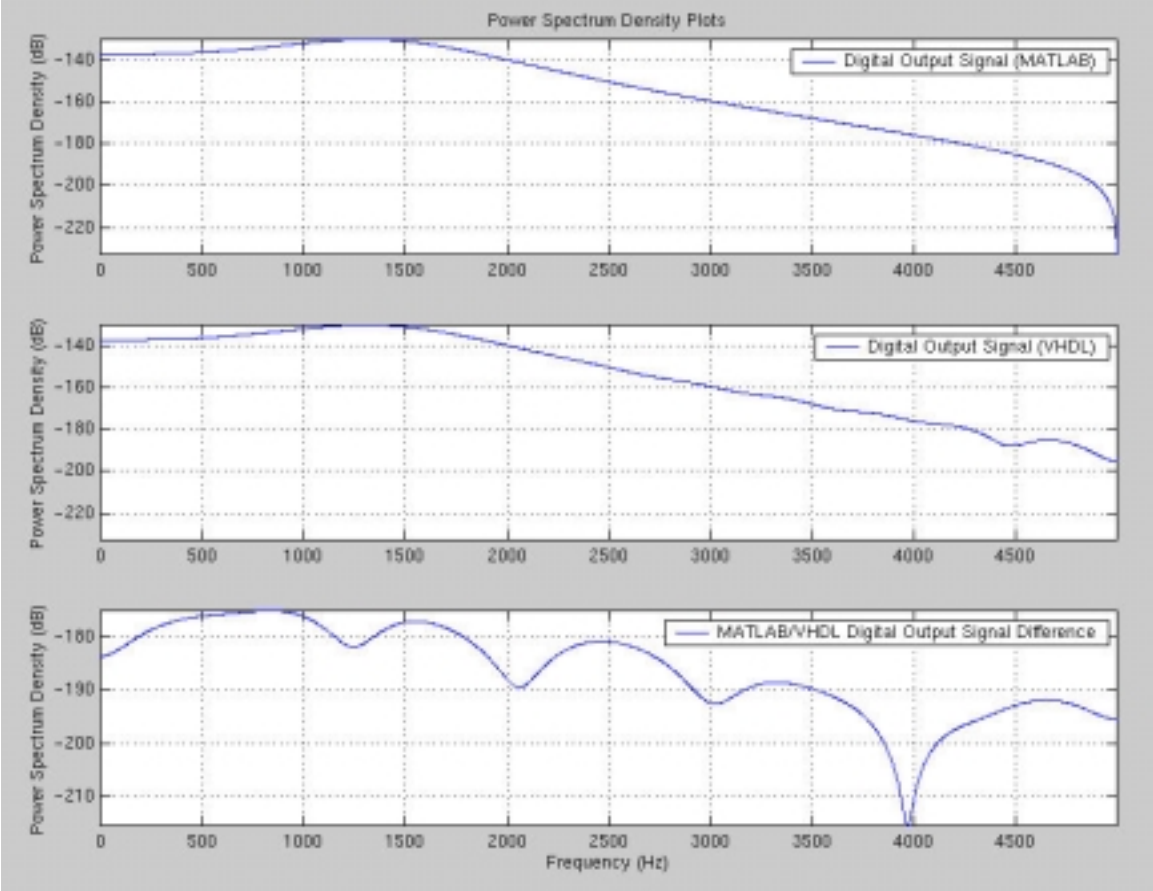


Fig. 9.19. Power spectral density results of impulse response of COSSAP *RoundMode = 2*.

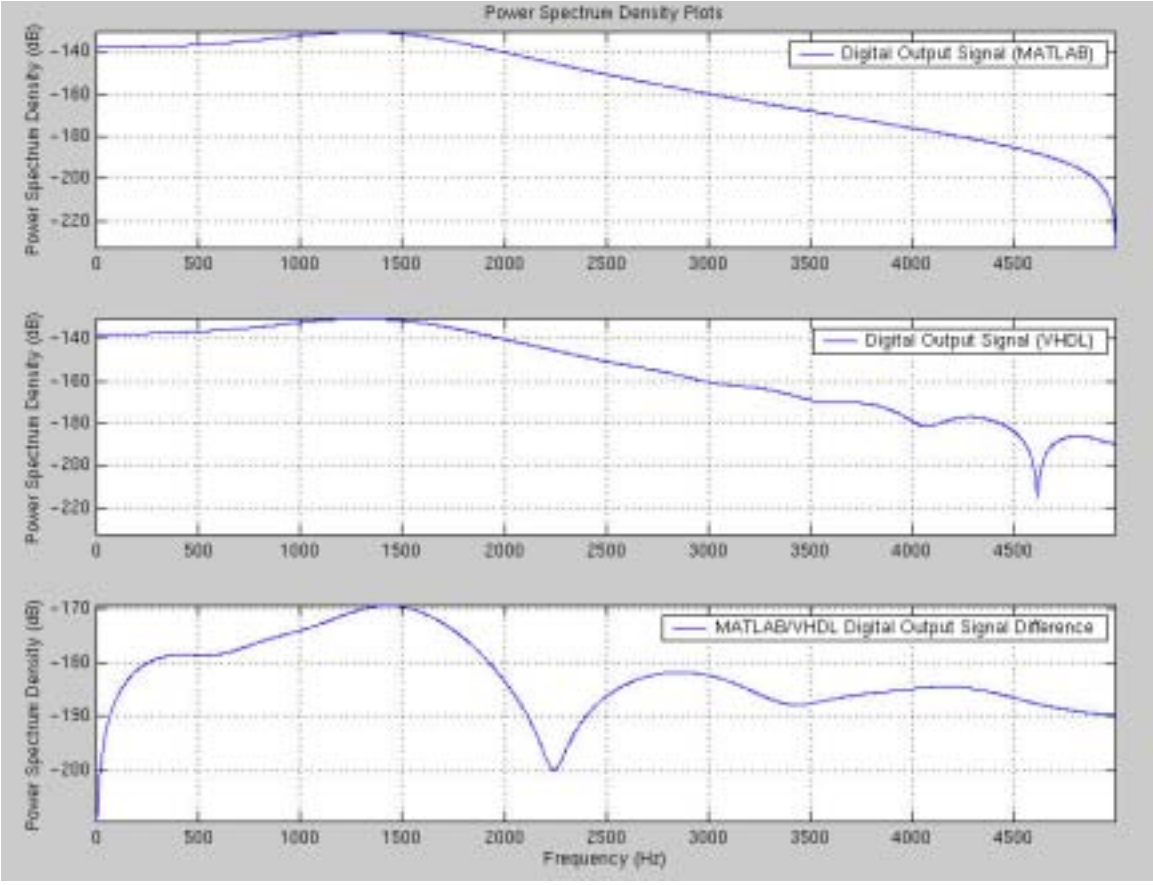


Fig. 9.20. Power spectral density results of impulse response of COSSAP *RoundMode = 3*.

9.1.1.1: Results of Validation Test #1

Figures 9.21 through 9.24 show the power spectral density of a passband digital signal (Step 21.1 of section 8.2) for the four COSSAP round-off modes. Results are based on the ideal 3rd order transfer function when compared to the fixed-point quantized parallel structure. For this validation test, the passband, transition band, and stopband input signals are 100Hz, 2.5kHz, and 4kHz, respectively. Table 9-4 illustrates the round-off noise power results for the four COSSAP multiplier round-off modes. Results are based on the error between the ideal (unquantized) and quantized outputs.

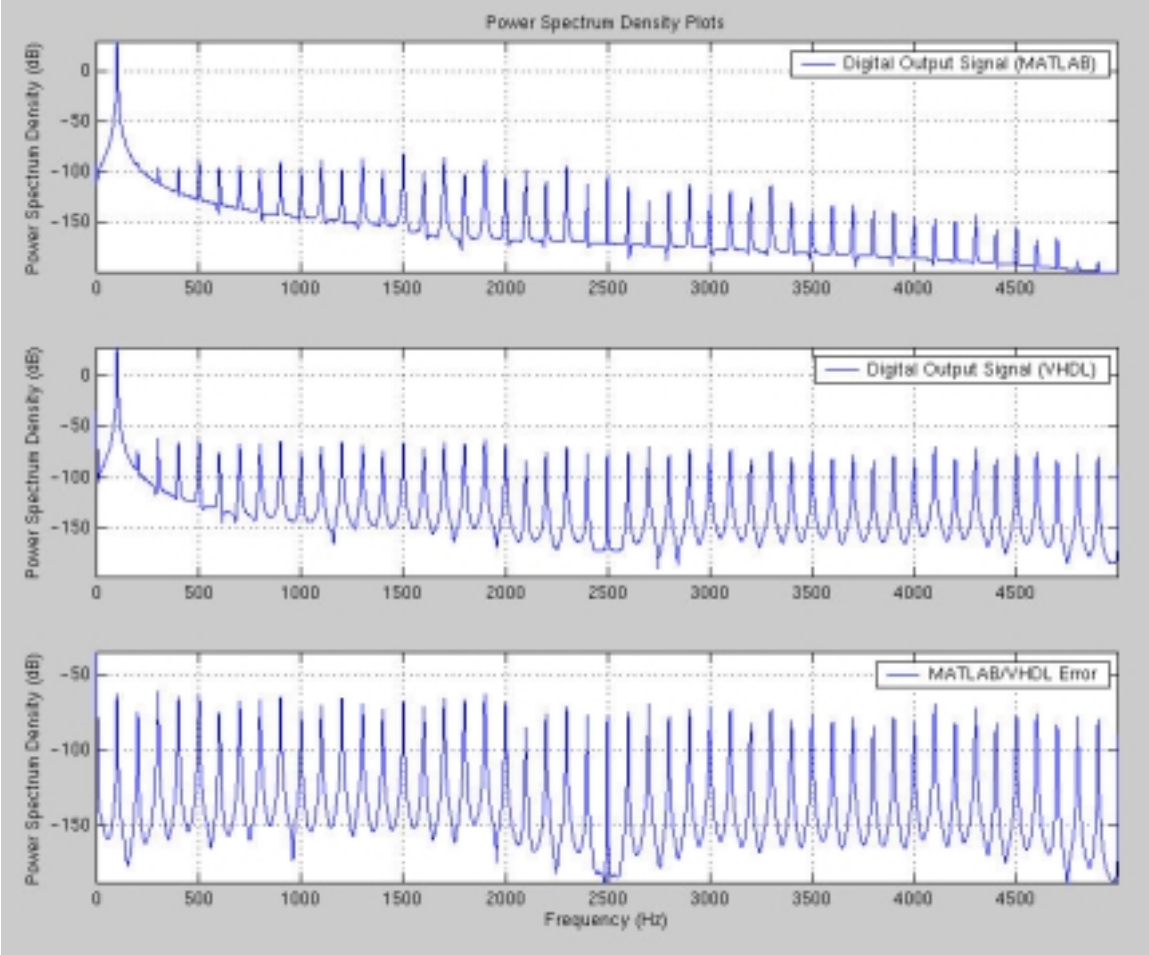


Fig. 9.21. Passband signal (100Hz) results of COSSAP *RoundMode = 0*.

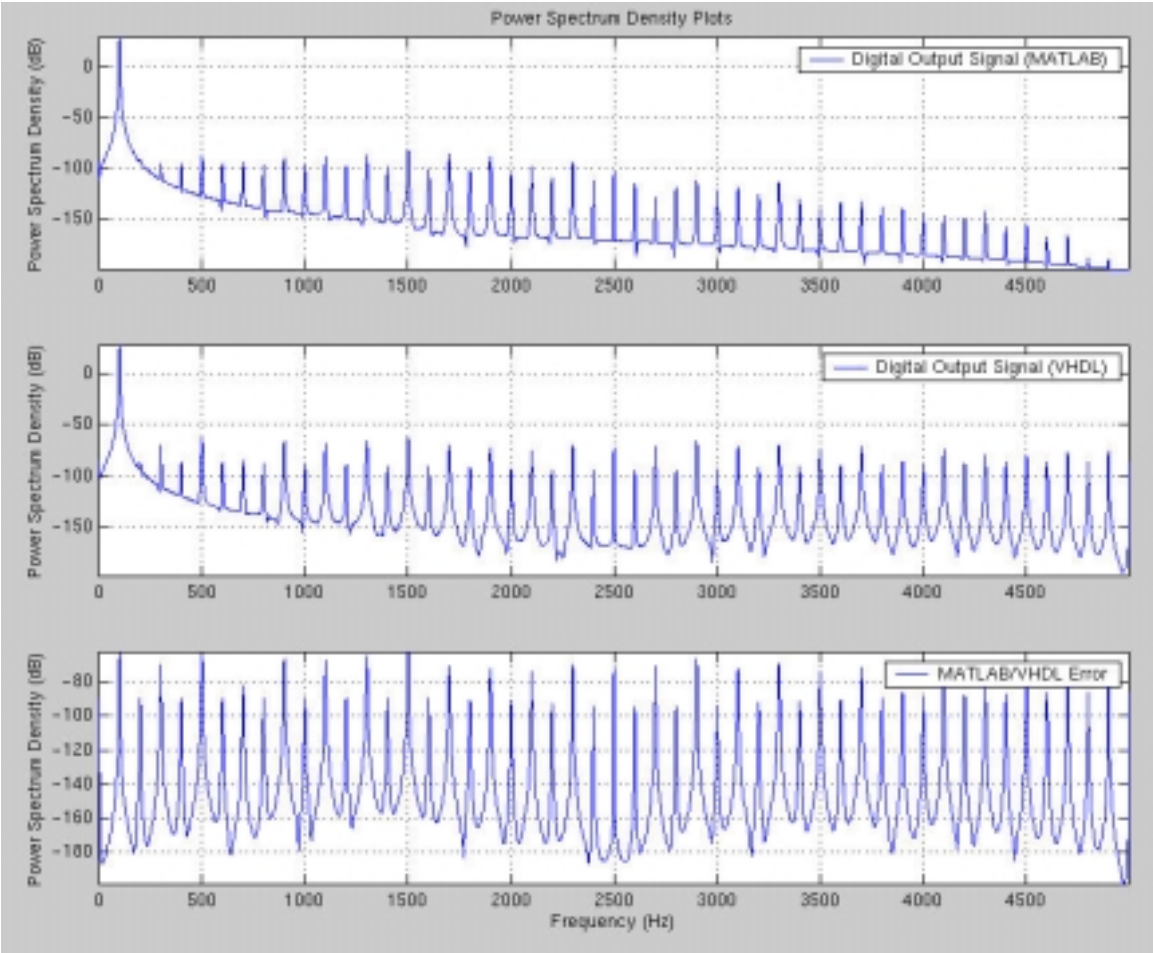


Fig. 9.22. Passband signal (100Hz) results of COSSAP *RoundMode = 1*.

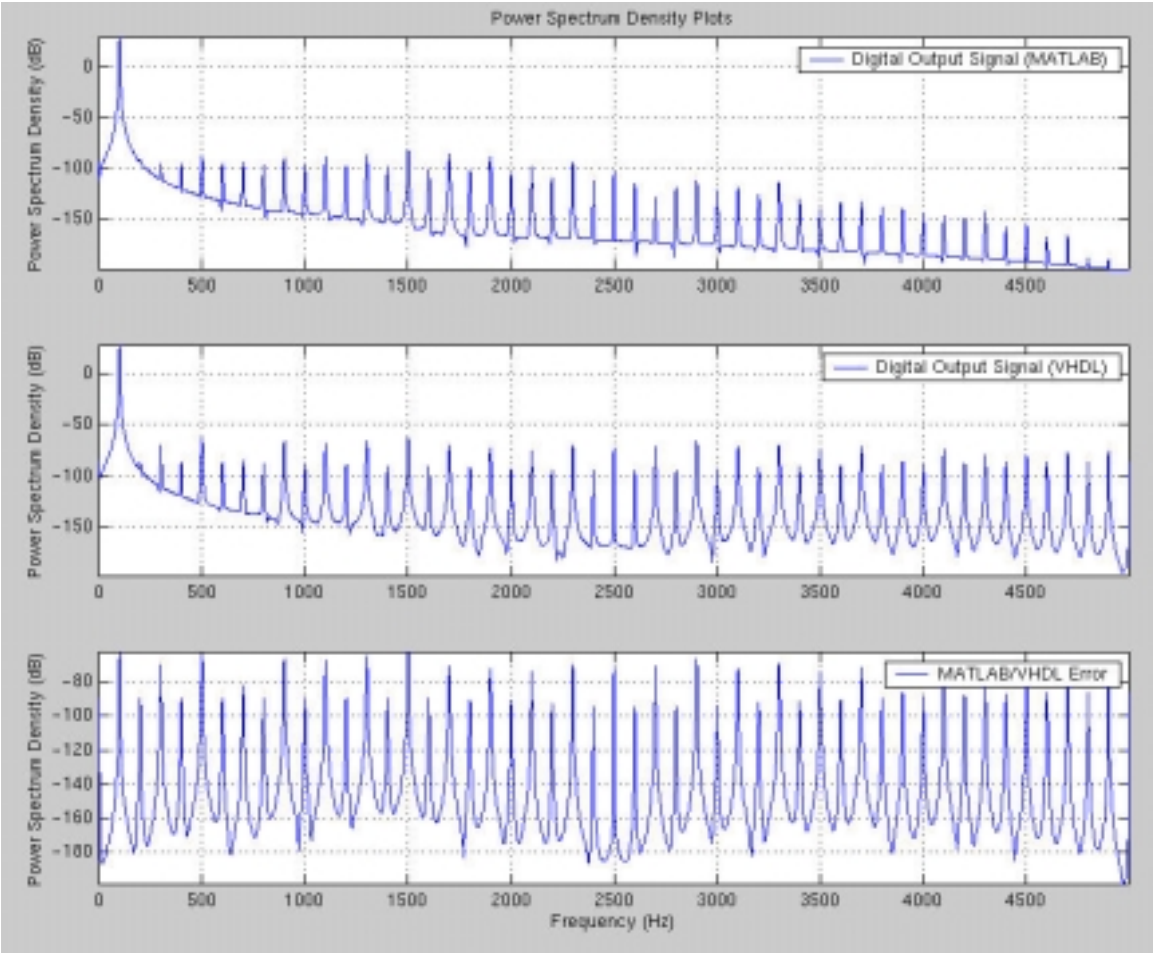


Fig. 9.23. Passband signal (100Hz) results of COSSAP *RoundMode* = *2*.

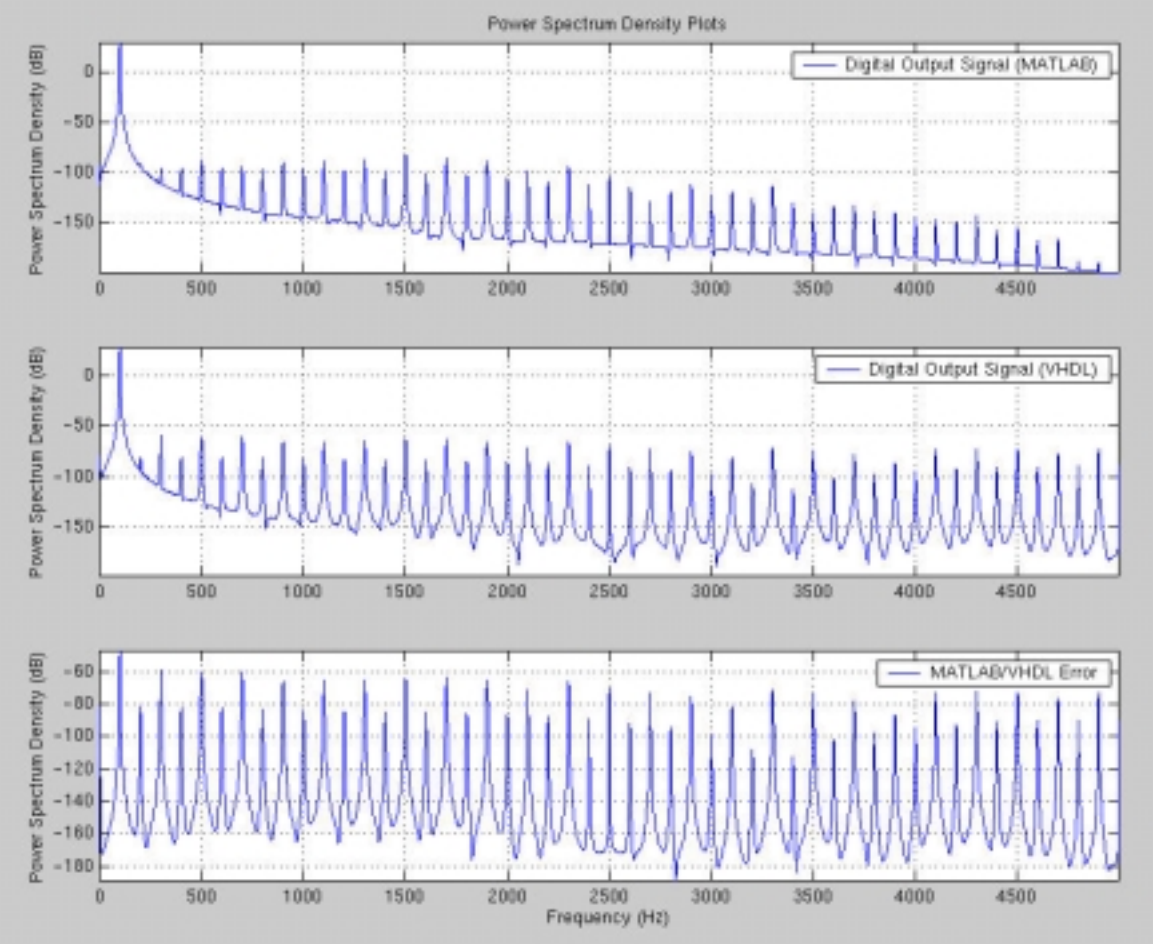


Fig. 9.24. Passband signal (100Hz) results of COSSAP *RoundMode = 3*.

	Round-off Noise Power (P_e)		
COSSAP RoundMode	**Passband Signal**	**Transition Band Signal**	**Stopband Signal**
0	$1.0882758913112 \times 10^{-7}$	$4.241848572125 \times 10^{-8}$	$8.577157750184 \times 10^{-8}$
1	$4.7110330 \times 10^{-13}$	$5.743103286 \times 10^{-11}$	$1.4820192510 \times 10^{-10}$
2	$4.7110330 \times 10^{-13}$	$5.743103286 \times 10^{-11}$	$1.4820192510 \times 10^{-10}$
3	$3.12153790 \times 10^{-12}$	$4.70576538624 \times 10^{-9}$	$1.4892842154 \times 10^{-10}$

Table 9-4. Tabulated results of validation test #1.

9.1.1.2: Results of Validation Test #2

Figure 9.25 shows the digital input/output response of the ideal lowpass Butterworth filter. The low-power passband input signal is at 100Hz. The high-power stopband input signal is at 4kHz. For the purposes of analysis, the high-power signal is 10 times greater in amplitude than the low-power signal. All figures of this validation test shown throughout this research will consist of 4 plots. The format of these plots is as follows:

- The top-most plot represents the low-power passband input frequency.
- The plot second from the top represents the high-power stopband input frequency.
- The plot third from the top represents the input signal of the two input frequencies added together.
- The bottom-most plot represents the ideal output of the MATLAB filter.

As can be seen from the figure, excluding the initial transient response at the output, the ideal Butterworth filter effectively removes the high-power stopband signal. Comparing the quantized filter, Figures 9.26 through 9.29 show the power spectral density plots for the four COSSAP round-off modes. Table 9-5 shows the tabulated results of the round-off noise power based on the COSSAP multiplier sub-blocks. Results are based on the error between the ideal (unquantized) and quantized outputs.

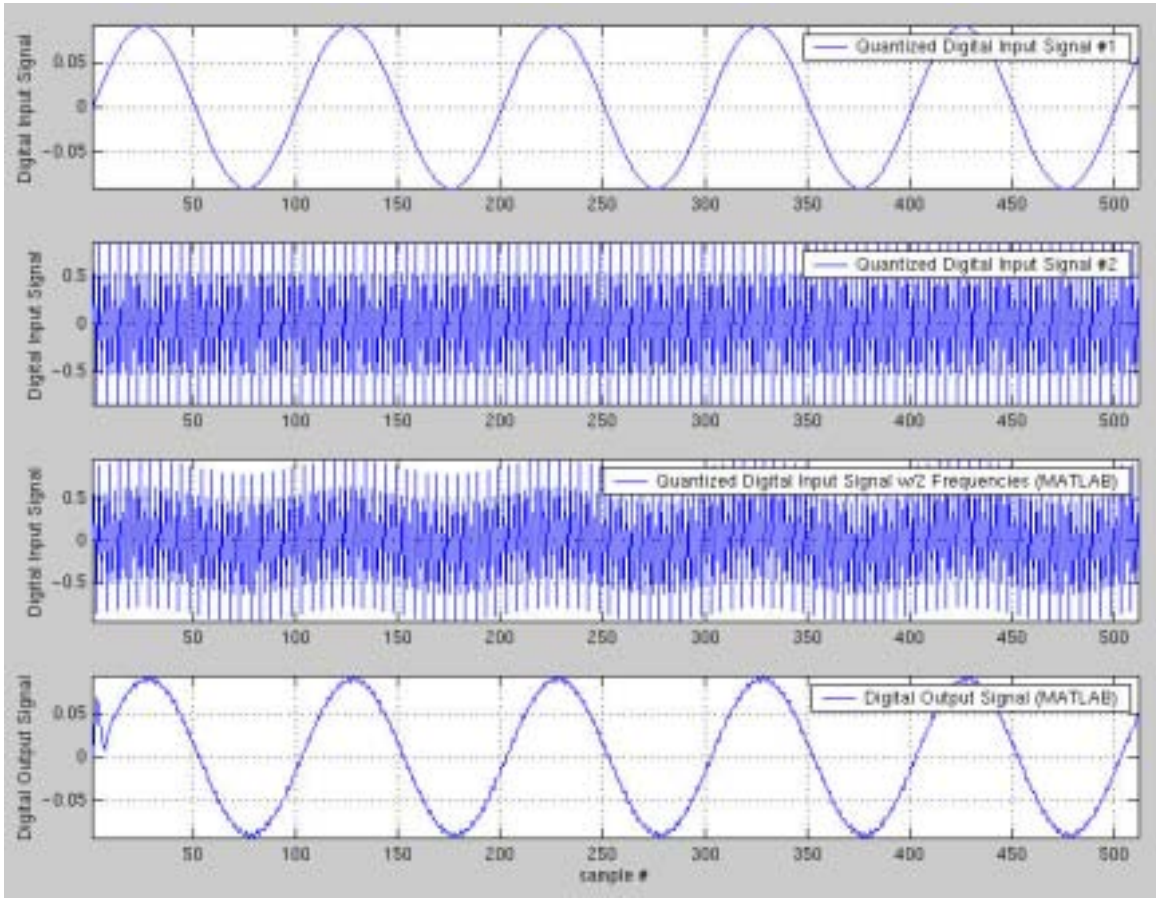


Fig. 9.25. Digital I/O response of validation test #2 for ideal Butterworth filter.

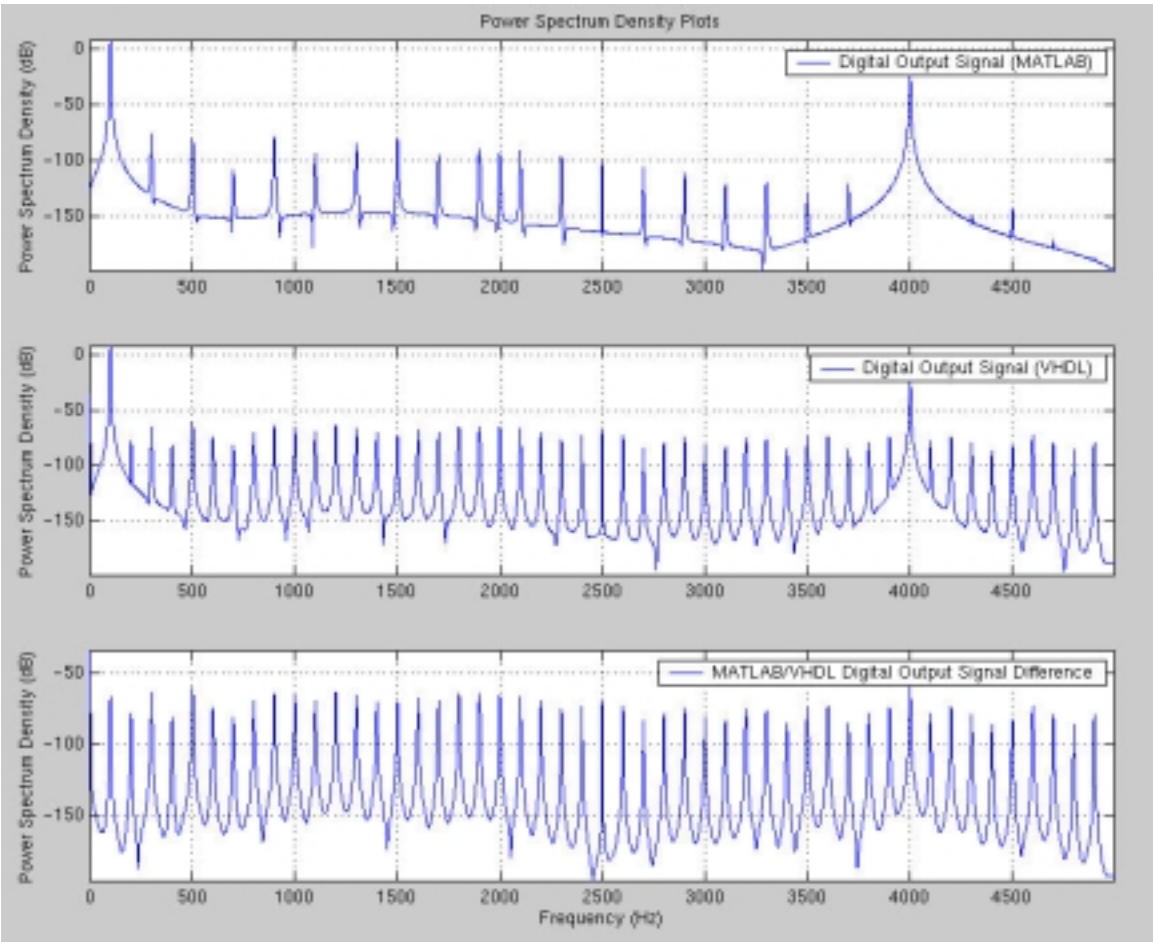


Fig. 9.26. Power spectral density plot of COSSAP *RoundMode = 0*.

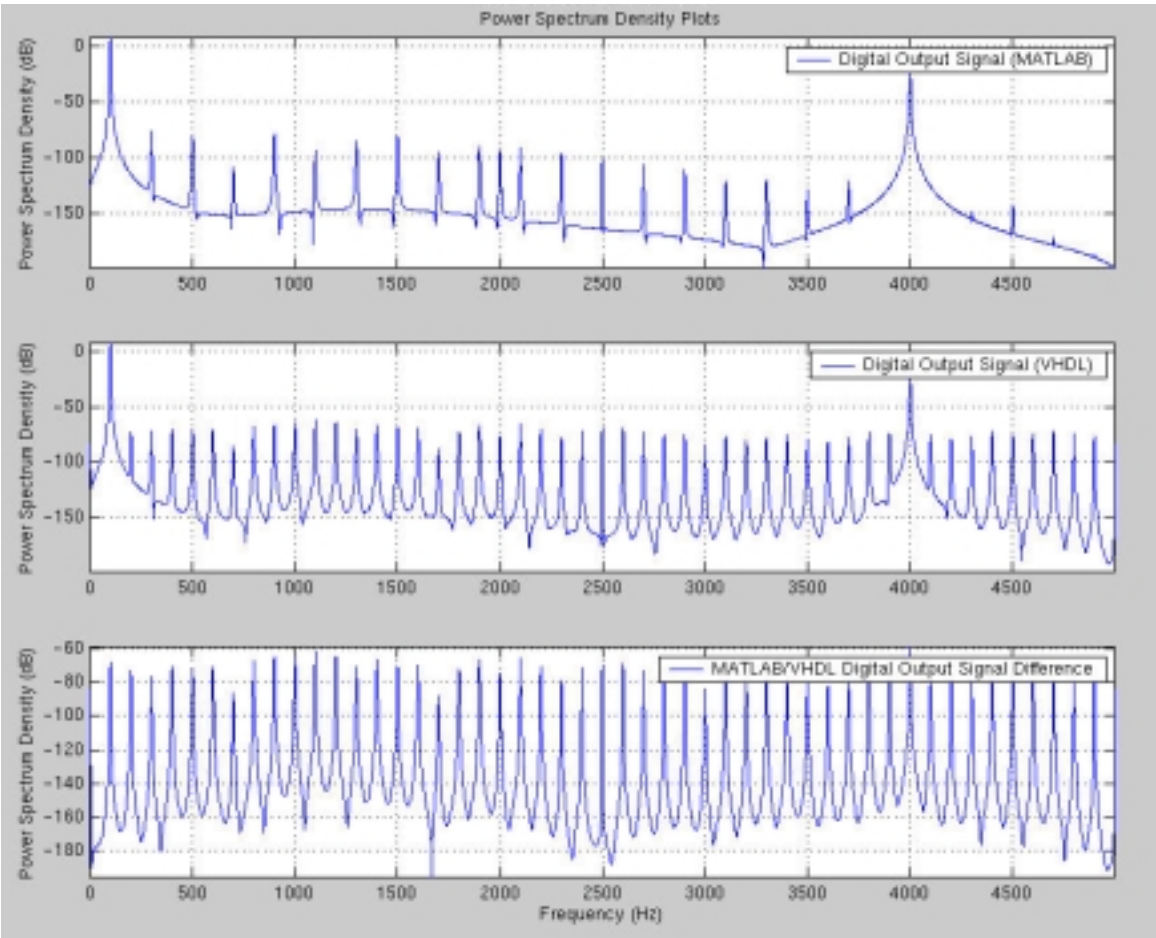


Fig. 9.27. Power spectral density plot of COSSAP *RoundMode* = *1*.

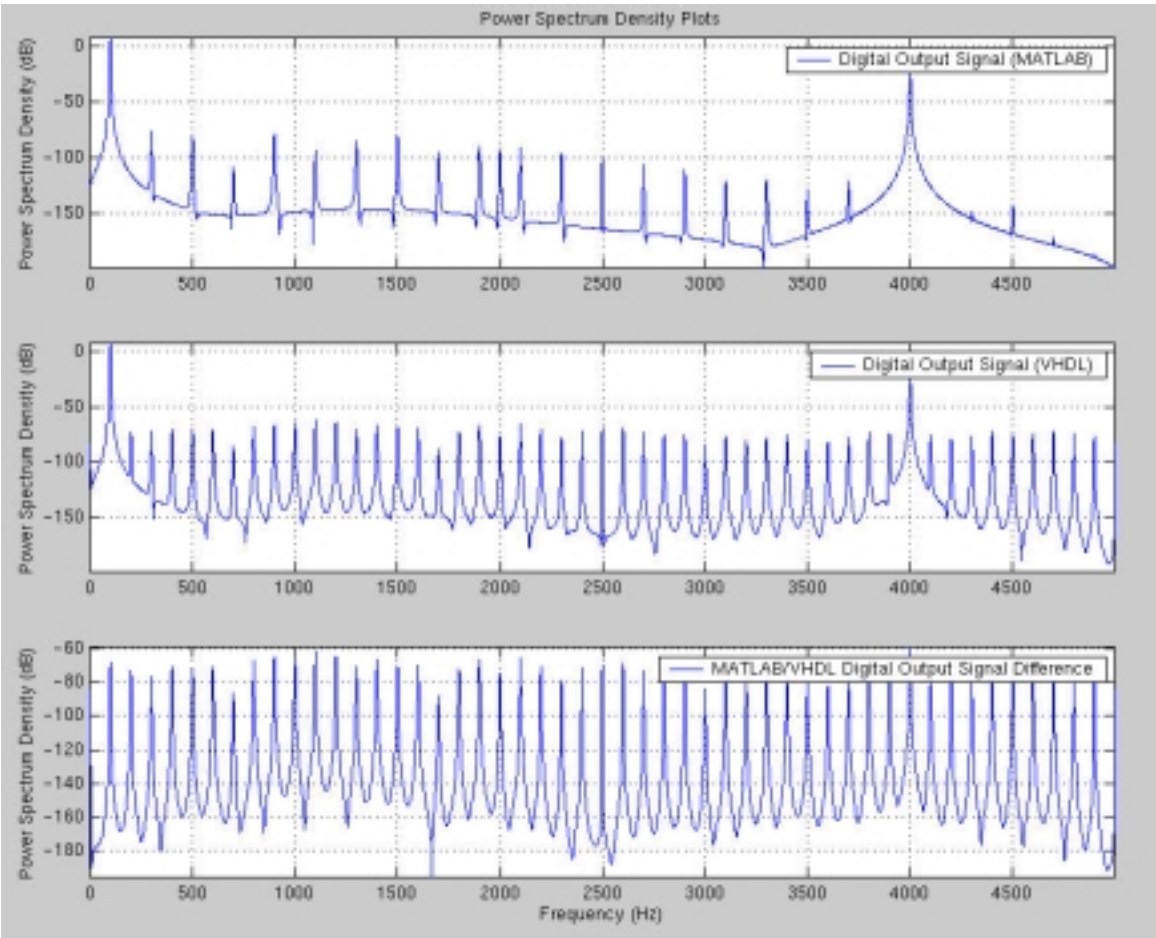


Fig. 9.28. Power spectral density plot of COSSAP *RoundMode* = *2*.

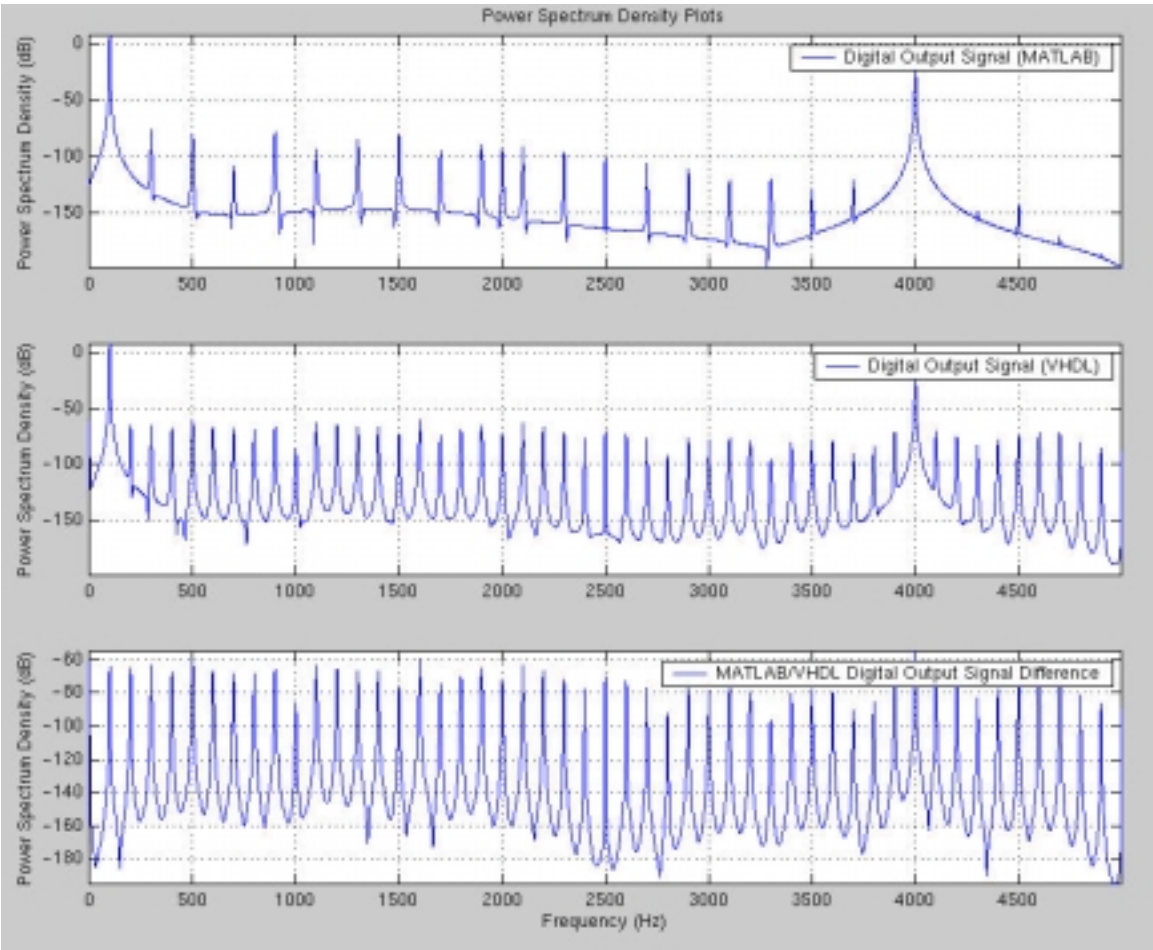


Fig. 9.29. Power spectral density plot of COSSAP *RoundMode* = *3*.

COSSAP RoundMode	P_e
0	$1.1922878511401 \times 10^{-7}$
1	$1.58058513 \times 10^{-12}$
2	$1.58058513 \times 10^{-12}$
3	$2.7487940146 \times 10^{-10}$

Table 9-5. Tabulated results of 4 COSSAP round-off modes.

Based on the results of Table 9-5, the best COSSAP round-off modes are *RoundMode* = *1* and *RoundMode* = *2*.

9.1.2: Cascade Structure Implementation of Butterworth Lowpass Filter

Based on Equation 3.7 from Chapter 3, the ideal transfer function sections of a cascade structure realization of the overall transfer function is the following:

$$H(z) = H_1(z)H_2(z) \qquad \text{(Eqn. 9.9)}$$

$H_1(z)$: (Eqn. 9.10)

$b_0 =$ 0.17002047747261
$b_1 =$ 0.34004011886191
$b_2 =$ 0.17001964139341
$a_1 =$ -0.83699778743883
$a_2 =$ 0.42398568894741

$H_2(z)$: (Eqn. 9.11)

$b_0 =$ 0.29133547378275
$b_1 =$ 0.29133690643799
$a_1 =$ -0.32491969623291

Based on the outlined methodology of Chapter 8, the following data is straightforward. Using Table 8-1 in section 8.2, the number of bits assigned for integer and fractional fixed-point representation are 1 and 15, respectively.

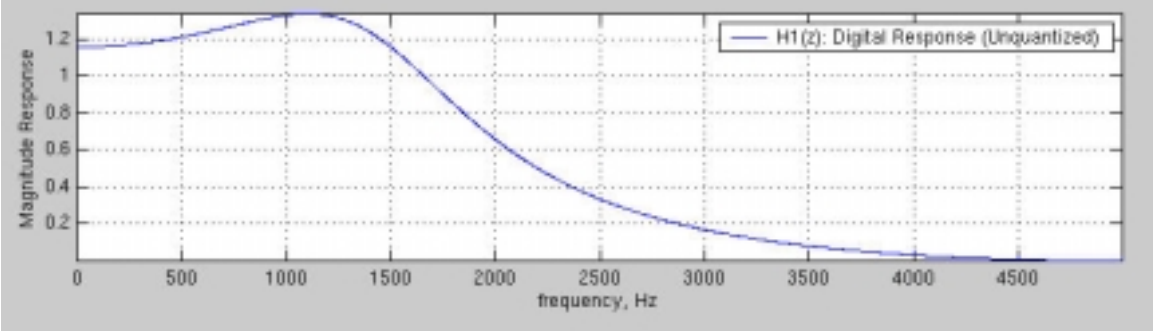


Fig. 9.30. MATLAB plot of magnitude response of ideal transfer function section *$H_1(z)$*.

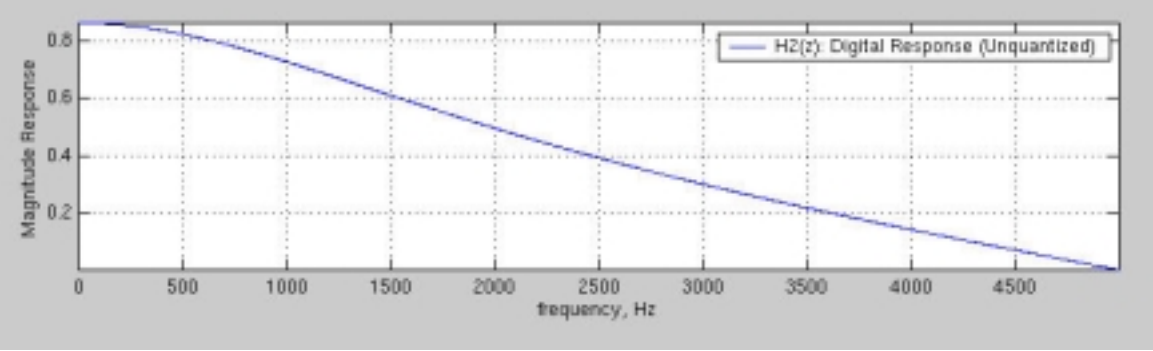


Fig. 9.31. MATLAB plot of magnitude response of ideal transfer function section *$H_2(z)$*.

$H_{1qt}(z)$: (Eqn. 9.12)

$b_0 = 0.1700134277343750$
$b_1 = 0.3400268554687500$
$b_2 = 0.1700134277343750$
$a_1 = -0.8369750976562500$
$a_2 = 0.4239807128906250$

$H_{2qt}(z)$: (Eqn. 9.13)

$b_0 = 0.2913208007812500$
$b_1 = 0.2913208007812500$
$a_1 = -0.3248901367187500$

$H_{1qr}(z)$: (Eqn. 9.14)

$b_0 = 0.1700134277343750$
$b_1 = 0.3400268554687500$
$b_2 = 0.1700134277343750$
$a_1 = -0.8370056152343750$
$a_2 = 0.4239807128906250$

$H_{2qr}(z)$: (Eqn. 9.15)

$b_0 = 0.2913208007812500$
$b_1 = 0.2913513183593750$
$a_1 = -0.3249206542968750$

Quantization Type	$H_{1e}(z)$	$H_{2e}(z)$
Truncated	0.00009483550988	0.00008338189205
Rounded	0.00005520336985	0.00000156157308

Table 9-6. Magnitude error calculation.

Based on the results of Table 9-6, the rounding method produces the better of the two quantization results in terms of smallest magnitude error. Based on Step 9.3.1 of section 8.2 the adder sub-blocks of cascade section $H_{1q}(z)$ have 2 bits assigned for both the input and output integer representation. Based on this same step, cascade section $H_{2q}(z)$ has 1 bit assigned for both the input and output integer representation.

$H_{1qr}(z)$		$H_{2qr}(z)$	
b0_COSSAP = 0	5571	b0_COSSAP = 0	9546
b1_COSSAP = 0	11142	b1_COSSAP = 0	9547
b2_COSSAP = 0	5571	a1_COSSAP = 0	10647
a1_COSSAP = 0	27427		
a2_COSSAP = 0	-13893		

Table 9-7. COSSAP results of multiplier coefficient transformation.

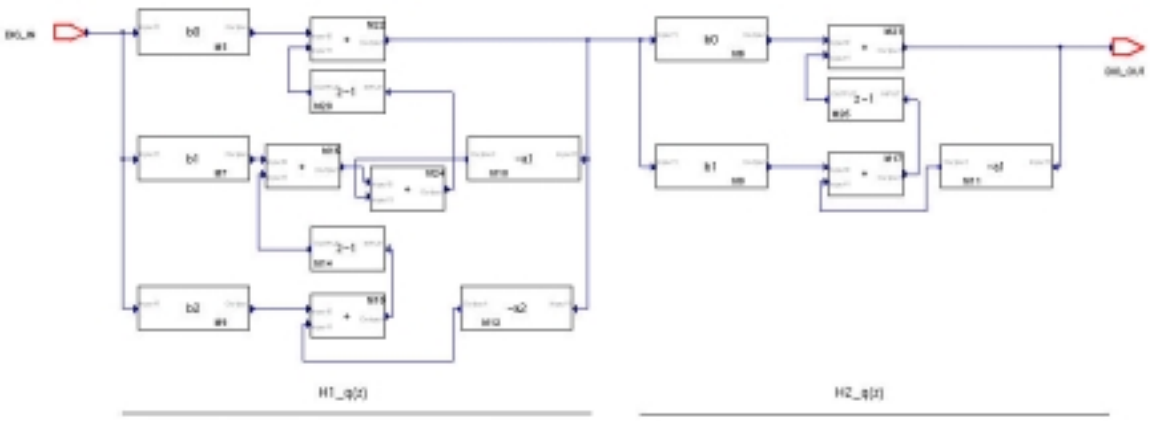


Fig. 9.32. COSSAP Block Diagram Editor of cascade structure.

COSSAP RoundMode	P_e	Dead Band Range (Fixed Value Output)	COSSAP/VHDL Peak PSD
0	$1.492845852491 \times 10^{-8}$	$-1.220703125 \times 10^{-4}$	-43.904228 dB at 0.000000 Hz
1	$8.28396551356 \times 10^{-9}$	$9.1552734375 \times 10^{-5}$	-46.402601 dB at 0.000000 Hz
2	$8.28396551356 \times 10^{-9}$	$9.1552734375 \times 10^{-5}$	-46.402601 dB at 0.000000 Hz
3	1.99840×10^{-15}	0.00	-130.496210 dB at 1301.269531 Hz

Table 9-8. Round-off noise power results of impulse response of COSSAP multiplier round-off modes.

In MATLAB, the peak power spectral density (PSD) is -130.405917 dB at 1311.035156 Hz for the ideal impulse response.

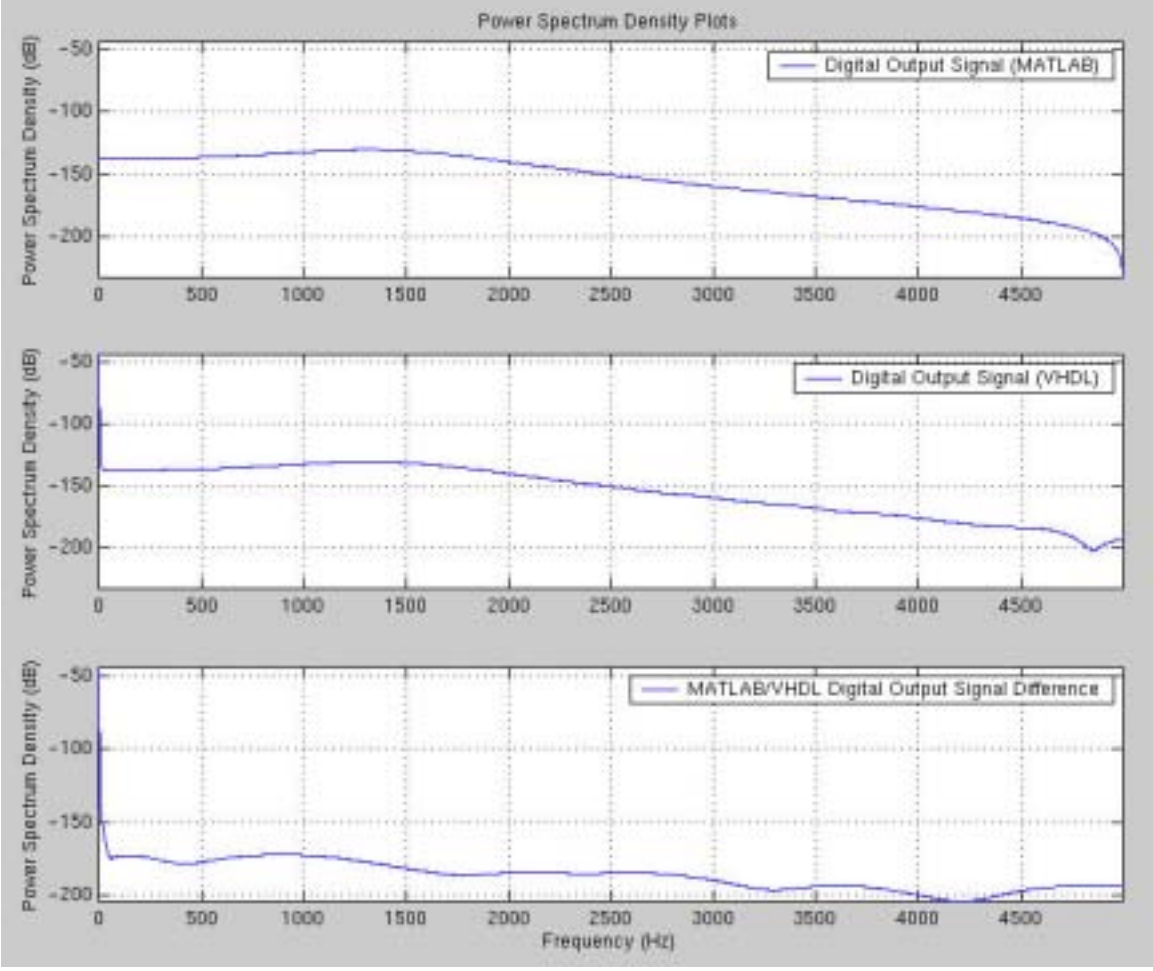


Fig. 9.33. Power spectral density of impulse response of COSSAP *RoundMode = 0*.

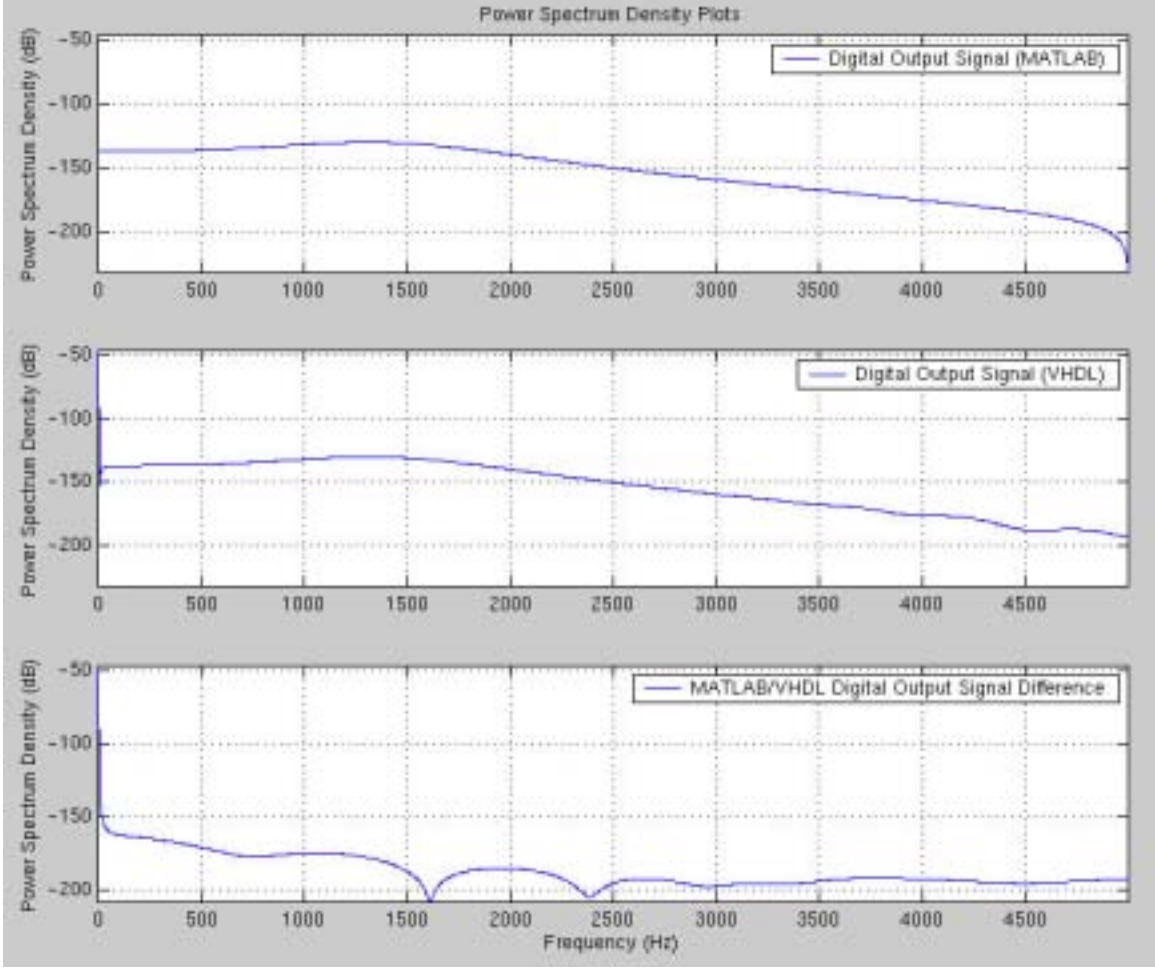


Fig. 9.34. Power spectral density of impulse response of COSSAP *RoundMode = 1*.

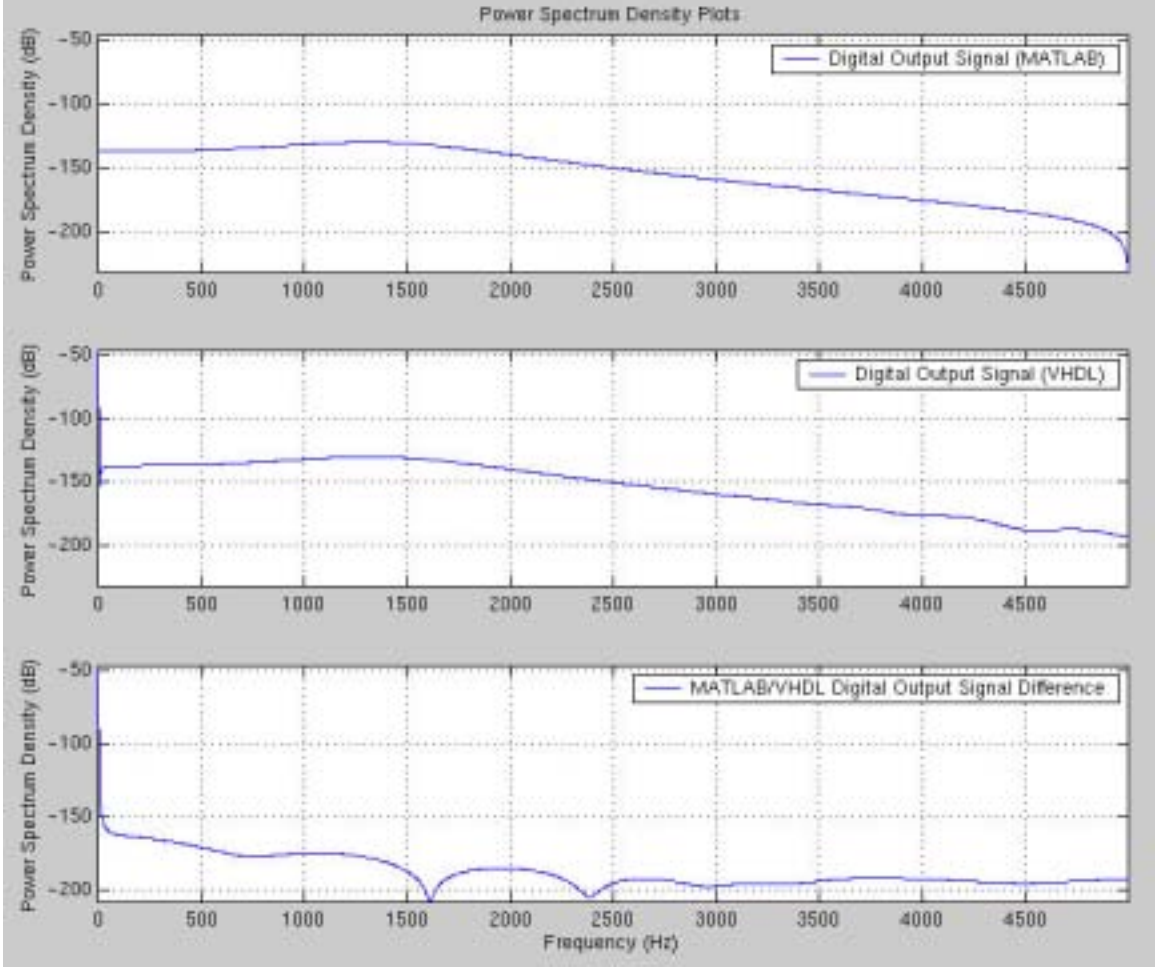


Fig. 9.35. Power spectral density of impulse response of COSSAP
RoundMode = 2.

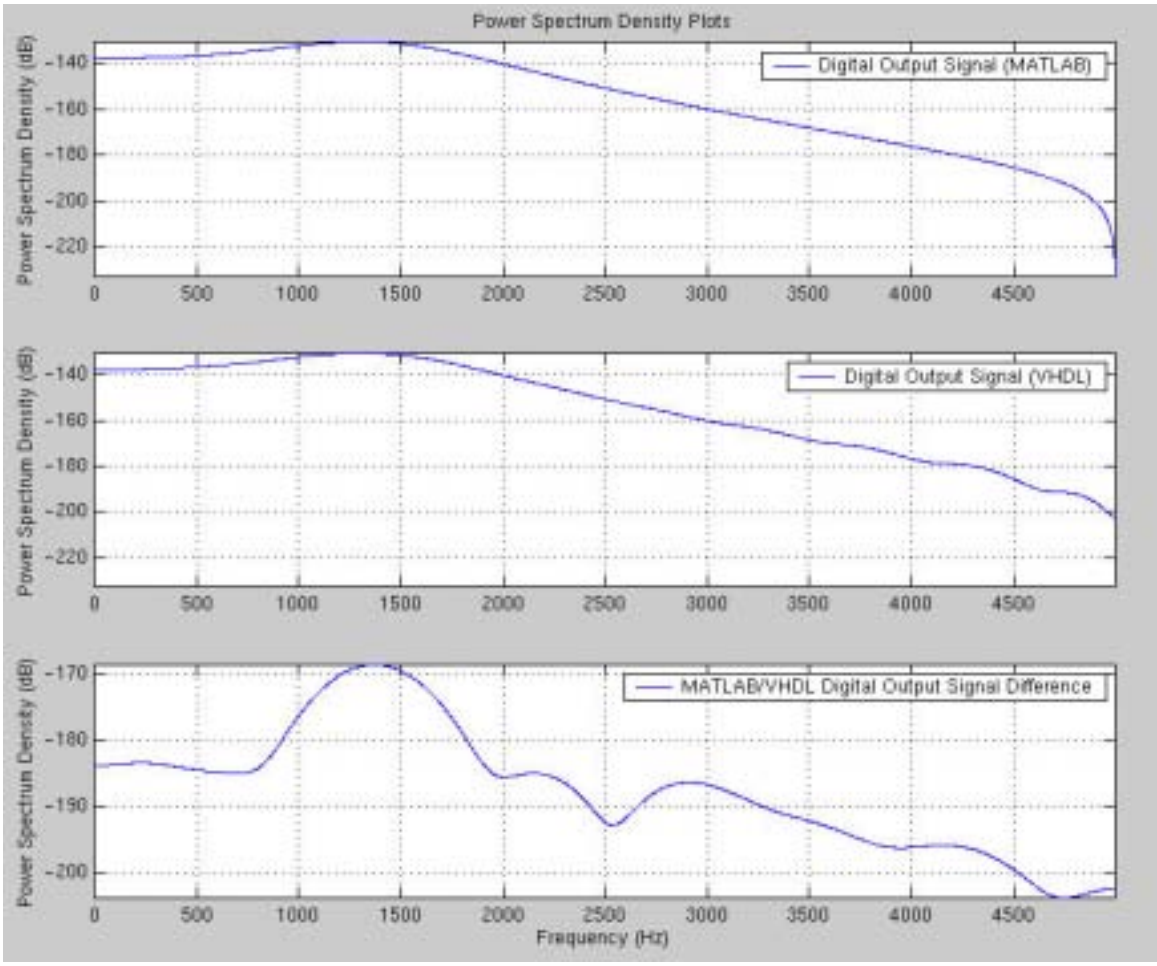


Fig. 9.36. Power spectral density of impulse response of COSSAP *RoundMode = 3*.

9.1.2.1: Results of Validation Test #1

The passband, transition band, and stopband signals used in section 9.1.1.1 are the same signals used here for comparison. Table 9-9 shows these results.

	Round-off Noise Power (P_e)		
COSSAP RoundMode	**Passband Signal**	**Transition Band Signal**	**Stopband Signal**
0	$8.134425152775 \times 10^{-8}$	$4.558753381259 \times 10^{-8}$	$7.705438106169 \times 10^{-8}$
1	7.145560×10^{-14}	$2.06750340680 \times 10^{-9}$	$9.2630103108 \times 10^{-10}$
2	$2.21773234 \times 10^{-12}$	$1.65072390827 \times 10^{-9}$	$9.2630103108 \times 10^{-10}$
3	$3.31123023 \times 10^{-12}$	$1.3150422857 \times 10^{-10}$	$1.4765820674 \times 10^{-10}$

Table 9-9. Tabulated results of validation test #1.

9.1.2.2: Results of Validation Test #2

The low-power passband input signal and high-power stopband input signal used in section 9.1.1.2 are used here for the purposes of comparison.

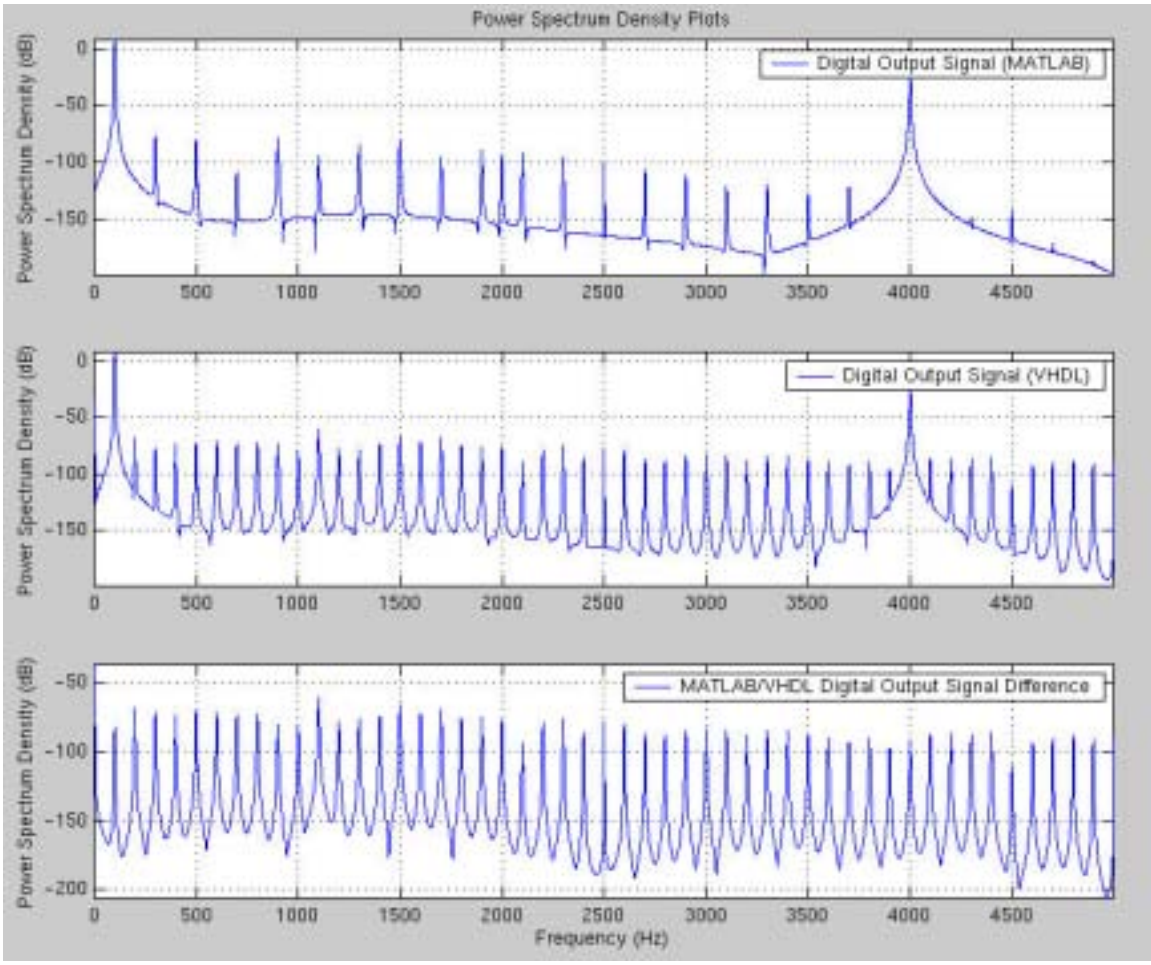


Fig. 9.37. Power spectral density of COSSAP *RoundMode* = *0*.

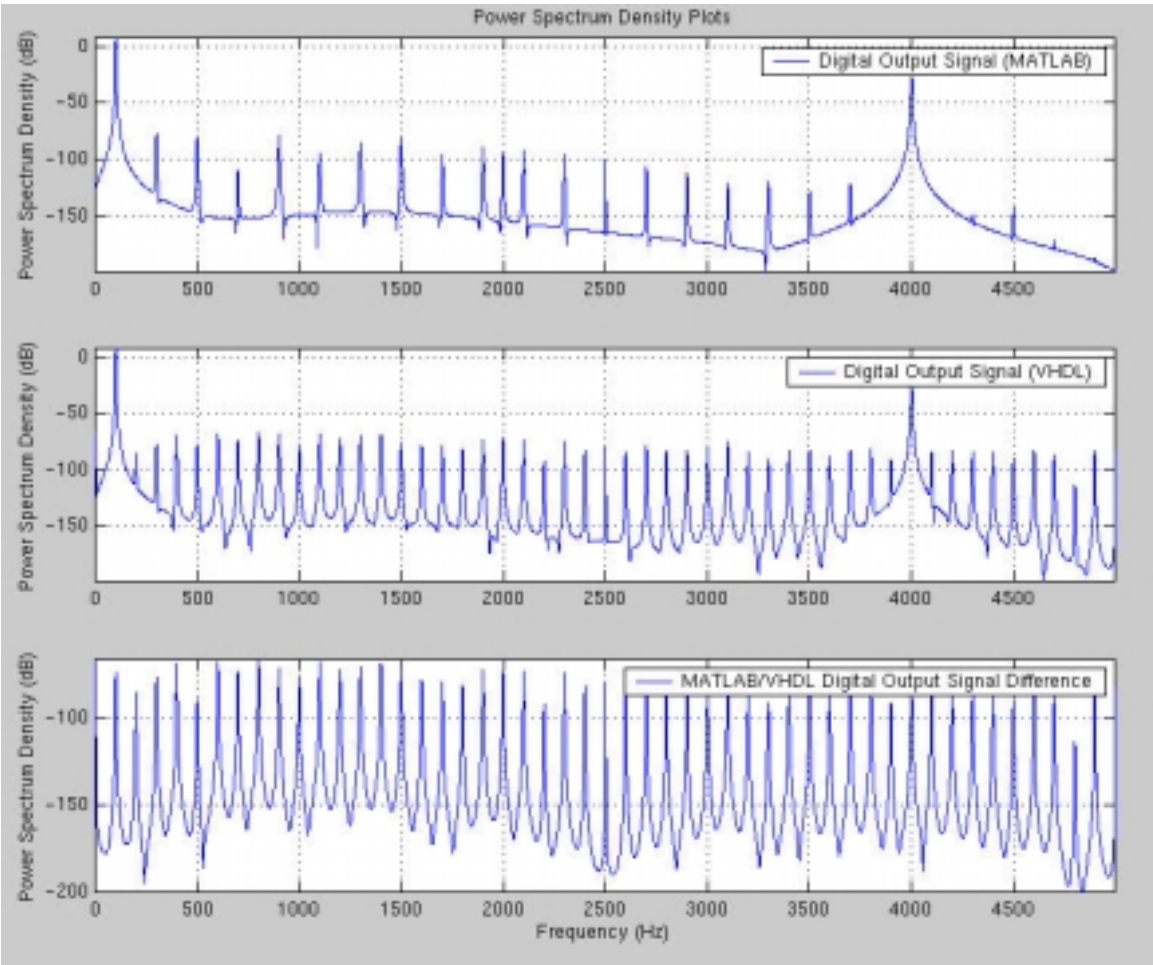


Fig. 9.38. Power spectral density of COSSAP *RoundMode* = *1*.

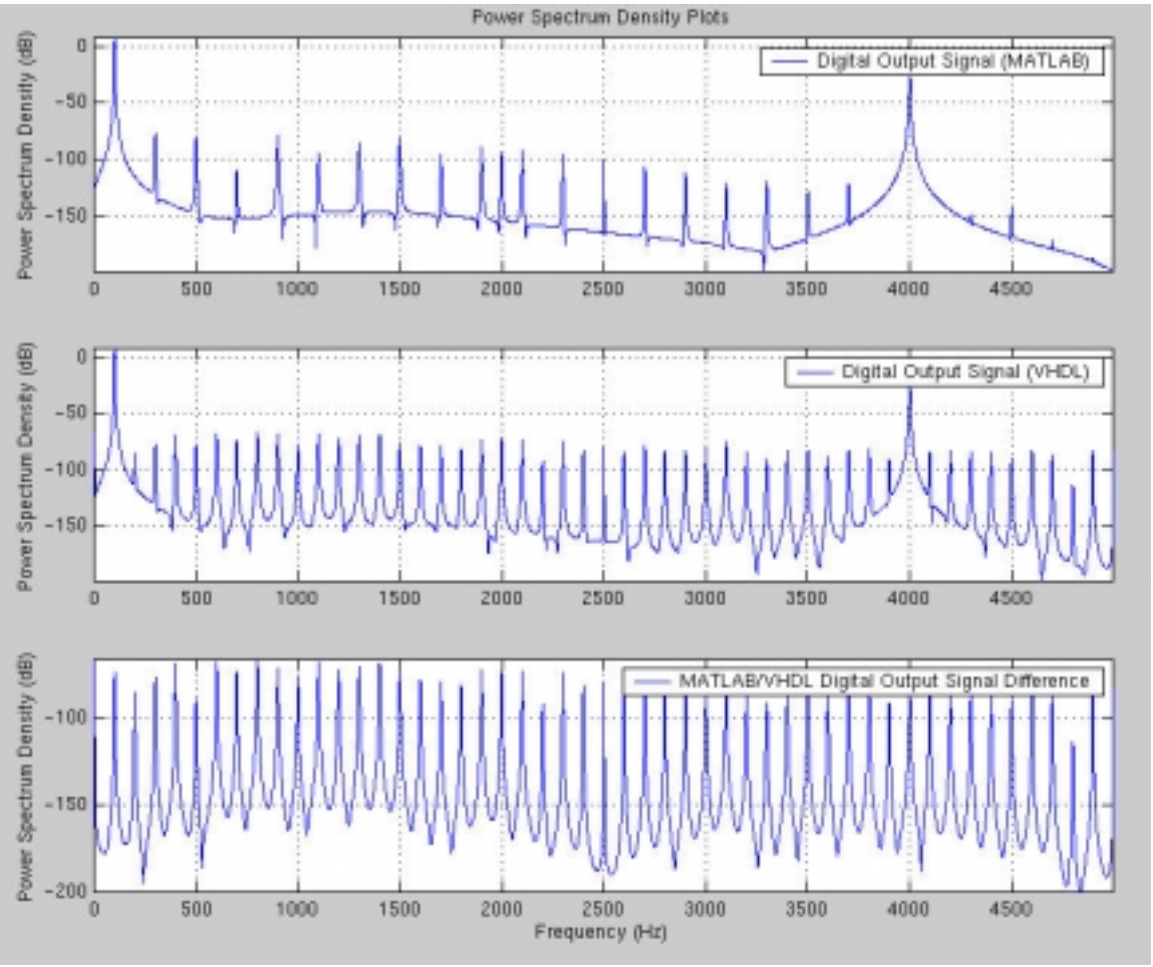


Fig. 9.39. Power spectral density of COSSAP *RoundMode* = *2*.

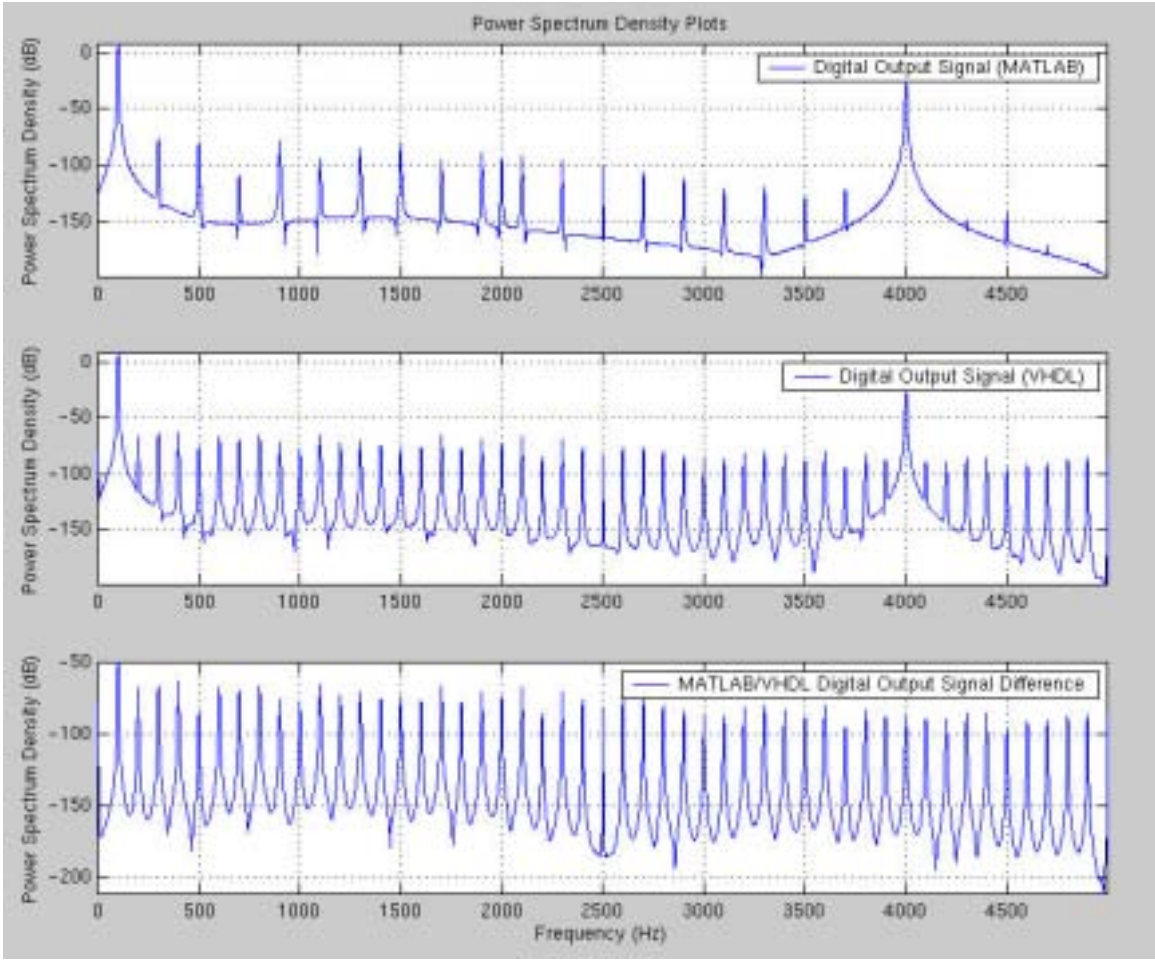


Fig. 9.40. Power spectral density of COSSAP *RoundMode* = *3*.

COSSAP RoundMode	Round-off Noise Power (P_e)
0	$8.322332829021 \times 10^{-8}$
1	$6.193306856 \times 10^{-11}$
2	$6.193306856 \times 10^{-11}$
3	$3.96516653 \times 10^{-12}$

Table 9-10. Tabulated results of 4 COSSAP round-off modes.

Based on the results of Table 9-10, the best COSSAP multiplier sub-block round-off mode is *RoundMode* = *3*.

9.1.3: VHDL Synthesis of Parallel Structure

The first step in synthesizing the parallel structure requires the designer to start with the multiplier, adder, and delay sub-blocks. Figure 9.15 is used as a visual reference for the designer. Instances M17, M22, M23, and M27 are considered 2-input adders. The script file that was used in this research for synthesis can be found in Appendix G. As stated in the introduction in Chapter 1, a broad background of VHDL synthesis is required on the part of the reader. The script file sets constraints on which technology library to use, defines timing parameters in terms of maximum allowable combinational propagation delay, and records synthesis results via report files. It is important to note that, after synthesis is complete, the synthesized circuit is saved in database format (.db extension). The reason for this brief mentioning will later on be explained. All sub-blocks will be saved in database format as well as in VHDL format. Instances M16 and M24 are considered to be a 3-input adder. The script file used by this research for synthesis can also be found Appendix G. The numerical values used in the script files for this research are all in nanoseconds units.

Instances M5, M7, M8, M9, M10, M11, and M12 are the multiplier sub-blocks with coefficients. An example script file for the multiplier sub-blocks for this structure is found in Appendix G. The only difference between synthesizing these sub-blocks and the adder sub-blocks is that the multipliers are unique. They are unique because they each are configured to a specific coefficient. In the top-level VHDL file, the adder and delay sub-blocks can have a single component declaration and be instantiated multiple times in the structural architecture body. For the multiplier sub-blocks, there is usually a one-to-one component declaration/instantiation ratio due to the specificity of each sub-block. Although there may be occasions where a multiplier sub-block could be instantiated more than once depending on the transfer function section, multiple instantiations are less likely to occur. The same goes for cascade structure realizations.

Instances M14, M20, and M25 are delay sub-blocks. Since these are clock-dependent components, timing requirements are particularly important. The script file used for these sub-blocks have timing constraints configured for a clock frequency (sampling frequency) of 10kHz. The script file can be found in Appendix G.

After synthesis is complete for the multiplier, adder, and delay sub-blocks, a top-level script file is the next and final step required for synthesizing this parallel structure. The top-level script file for this parallel structure can be found in Appendix B. The script file reads in the saved database files of the sub-blocks and connects them accordingly. The use of the *set_dont_touch* option is a useful and time-saving command. This style of efficient synthesis drastically reduces synthesis time. For example, if re-synthesis is required on one of the multiplier sub-blocks, the designer need only re-synthesize that particular sub-block. Next, the designer re-runs the top-level script file and re-synthesizes the top-level structure. The *set_dont_touch* option guarantees that re-synthesis does not occur on *all* the sub-blocks. This is the main importance of using the database formats of the saved sub-blocks. The script file also takes into account the propagation delay of the assumed A/D converter connected to the input of this digital filter, *x(n)*. For this research, a 10-nanosecond assumption is the propagation delay. The synthesis command for this is *set_input_delay*.

Lastly, it is noted here that for the synthesized model, parallel registers are incorporated at the filter output. These component additions ensure that the filter output is virtually glitch-free when clocked. The tradeoff is that the output is delayed by 1 clock cycle. Parallel registers are included for all the synthesized parallel structures in this research. The VHDL synthesis-ready top-level code can be found in Appendix B as well as the synthesis timing report.

Performing Step 27 of section 8.2 shows that the synthesized circuit and generic structural circuit produce the same output results.

9.1.4: VHDL Synthesis of Cascade Structure

The steps stated in the previous section (9.1.3) are also applied to designing the cascade structure of the lowpass Butterworth filter. The top-level script file can be found in Appendix B. Because the cascade structure by design incurs large propagation delay from input to output, at high sampling frequencies, the expected output data may not have ample processing time to be correctly computed. This is due to the effective combinational path from one cascade section to the next. This research remedies this situation by incorporating 16-bit parallel registers between *each* section and at the final output, *y(n)*. Specifically, because there are 2 cascade sections, there are 2 parallel shift registers incorporated into the cascade structure. Generally, if there are *n* cascade sections, there will be *n* parallel registers incorporated into the cascade structure. The tradeoff is that it will take an additional *n* clock cycles to compute the expected quantized data at the output *y(n)*. This method for incorporating parallel registers into the synthesized cascade structures is followed throughout this research. The VHDL synthesis-ready top-level code for this structure can be found in Appendix B. The top-level script file and timing report produced by synthesis can also be found there.

CHAPTER 10: Results of Design Methodology for DSP Applications

This chapter deals with the implementation of the outlined design methodology of Chapter 8 for DSP areas of data communications, imaging, digital video, and voice communication. Both parallel and cascade structures were explored. All digital filter designs in this chapter are 12^{th} order. Table 10.1 illustrates the bandwidths of each DSP application. Nominal numbers are used for each bandwidth. All the results of this chapter were based on 4096 samples. All input signals used for analysis were in the range of [-1,1).

DSP Application	Nominal Bandwidth
Data Communications and Imaging	30kHz – 5MHz
Digital Video	0Hz – 2MHz
Voice Communications	4kHz – 32kHz

Table 10-1. Table of DSP bandwidths for digital communications.

10.1: Voice Communication Bandwidth Results

The sampling frequency used for this design was 200kHz. This design is a 20-bit bandpass digital Butterworth filter. The first and second cut-off frequencies (-3dB points) are 4005Hz and 31990Hz, respectively. The first and second stopband frequencies (-40dB points) are 2051Hz and 52454Hz, respectively. The ideal transfer function is in the format as follows:

$$H(z) = \frac{b_0 + b_1z^{-1} + b_2z^{-2} + b_3z^{-3} + b_4z^{-4} + b_5z^{-5} + b_6z^{-6} + b_7z^{-7} + b_8z^{-8} + b_9z^{-9} + b_{10}z^{-10} + b_{11}z^{-11} + b_{12}z^{-12}}{1 + a_1z^{-1} + a_2z^{-2} + a_3z^{-3} + a_4z^{-4} + a_5z^{-5} + a_6z^{-6} + a_7z^{-7} + a_8z^{-8} + a_9z^{-9} + a_{10}z^{-10} + a_{11}z^{-11} + a_{12}z^{-12}}$$

The MATLAB floating-point representations of the coefficients, taken to 14 significant places, are as follows:

b_0 = 0.00184349359688
b_1 = 0.00000000000000
b_2 = -0.01106096158128
b_3 = 0.00000000000000
b_4 = 0.02765240395320
b_5 = 0.00000000000000
b_6 = -0.03686987193761
b_7 = 0.00000000000000
b_8 = 0.02765240395320
b_9 = 0.00000000000000
b_{10} = -0.01106096158128
b_{11} = 0.00000000000000
b_{12} = 0.00184349359688
a_1 = -8.04270971990645
a_2 = 30.01337319862856
a_3 = -68.99003418531834
a_4 = 109.13542229944890
a_5 = -125.41290790524731
a_6 = 107.44019947612287
a_7 = -69.14327117309267
a_8 = 33.16794617528626
a_9 = -11.56329960387100
a_{10} = 2.78021661742503

a_{11} = -0.41376834540838
a_{12} = 0.02883413690976

Following the outlined design methodology, the results are straightforward. As stated earlier in its research, all frequency plots are from 0Hz to the Nyquist frequency. For this section, the Nyquist frequency is 100kHz.

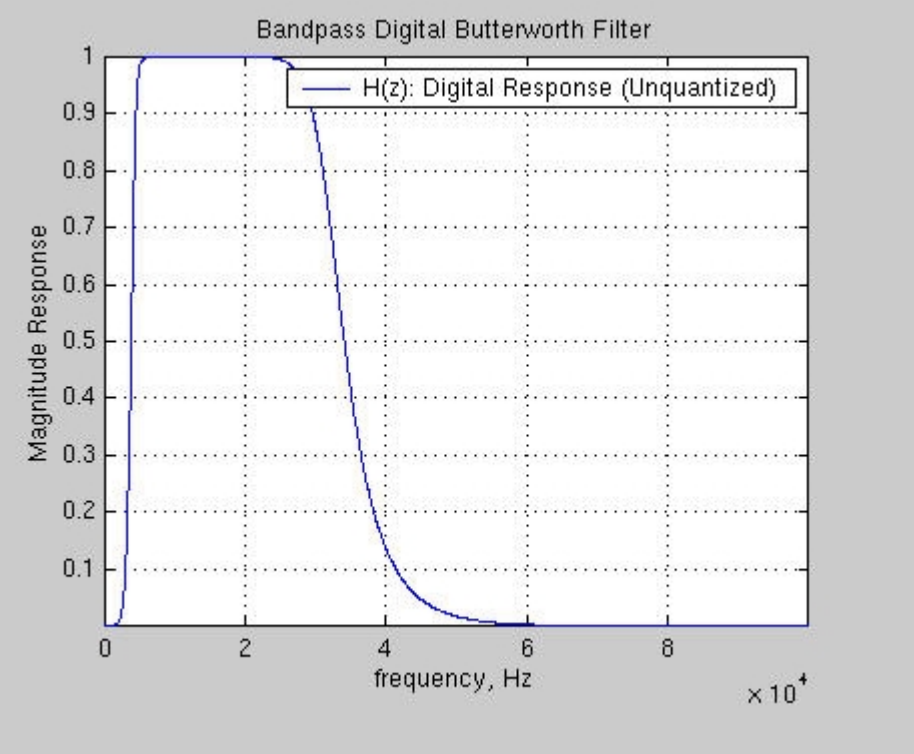


Fig. 10.1. MATLAB magnitude response of bandpass Butterworth filter.

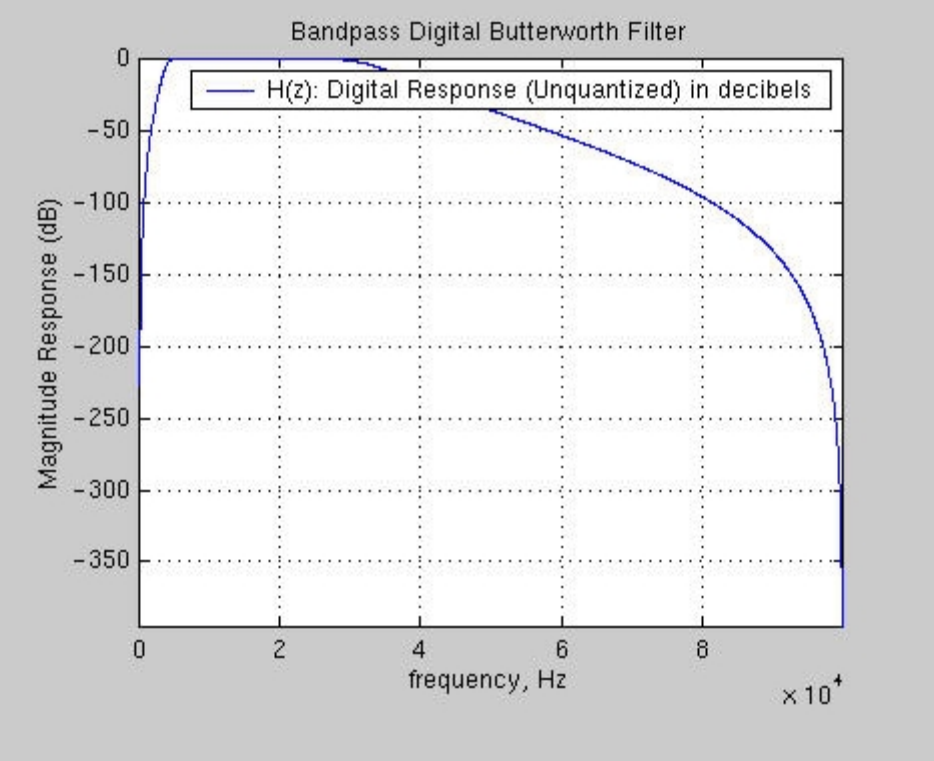


Fig. 10.2. MATLAB magnitude response of bandpass Butterworth filter (dB).

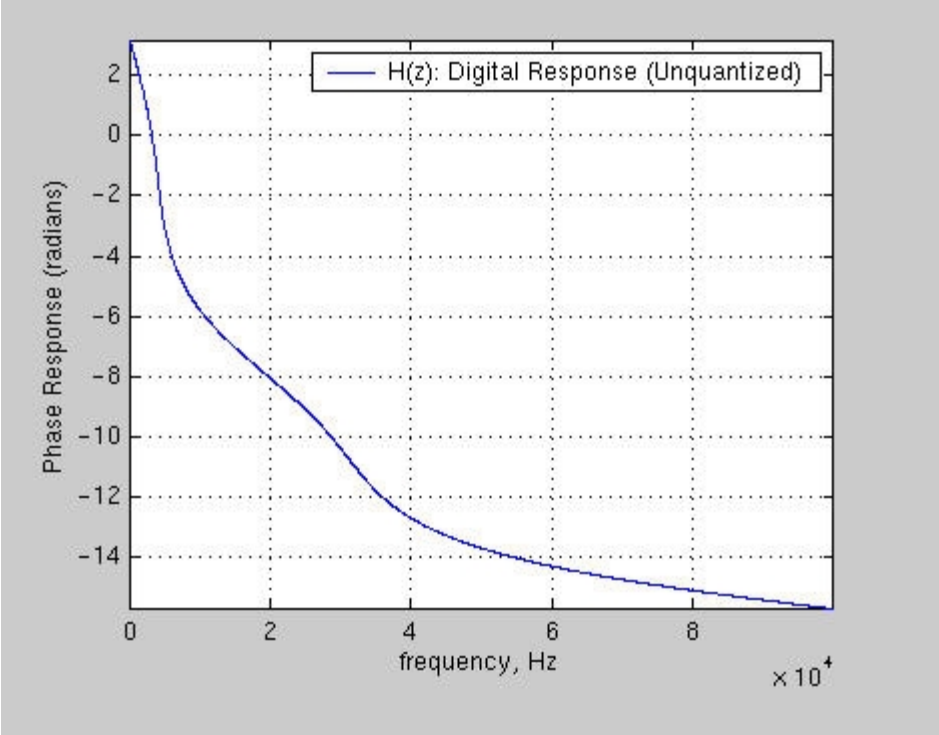


Fig. 10.3. MATLAB phase response of bandpass Butterworth filter (radians).

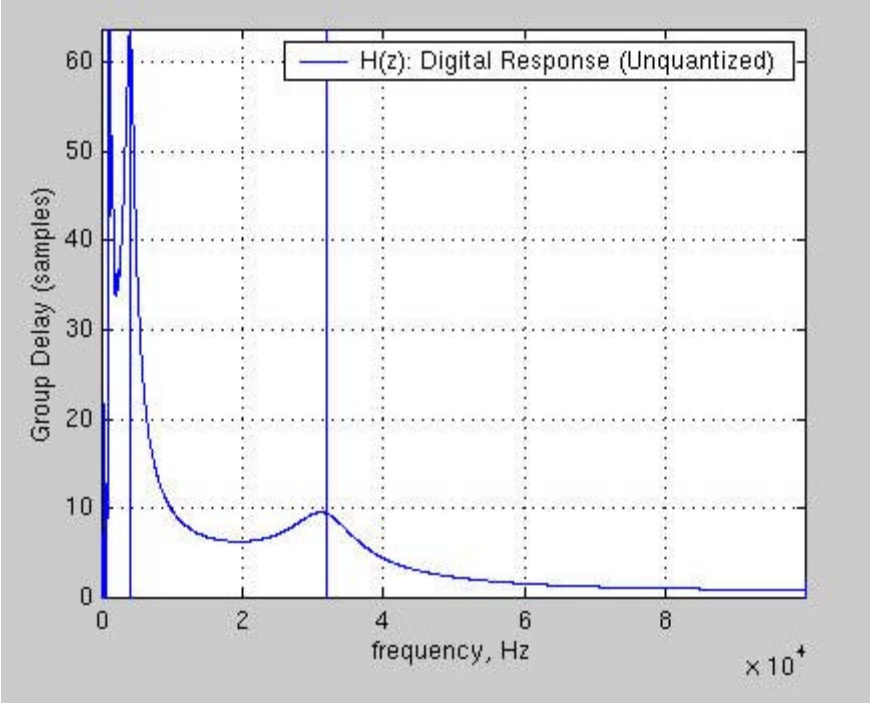


Fig. 10.4. MATLAB group delay response of bandpass Butterworth filter.

For Figure 10.4, vertical markers were placed at the cut-off frequencies. This was done to graphically illustrate the bandwidth region.

10.1.1: Parallel Structure of Butterworth Bandpass Filter Results

The unquantized transfer function sections are as follows:

$$H(z) = H_1(z)+H_2(z)+H_3(z)+H_4(z)+H_5(z)+H_6(z) \quad \text{(Eqn. 10.1)}$$

$H_1(z)$: (Eqn. 10.2)

$b_0 = 0.06523998068243$
$b_1 = -0.05677217409622$
$b_2 = 0.00000000000000$
$a_1 = -1.93317570078060$
$a_2 = 0.94885440559193$

$H_2(z)$: (Eqn. 10.3)

$b_0 = -0.15413571172704$
$b_1 = 0.17726428571135$
$b_2 = 0.00000000000000$
$a_1 = -1.82919846105757$
$a_2 = 0.84672134723855$

$H_3(z)$: (Eqn. 10.4)

$b_0 = -1.13070864967498$
$b_1 = 0.97778688899106$
$b_2 = 0.00000000000000$
$a_1 = -1.71517028232064$
$a_2 = 0.73813390470860$

$H_4(z)$: (Eqn. 10.5)

$b_0 = 0.30582385316763$
$b_1 = -0.47156149221871$
$b_2 = 0.00000000000000$
$a_1 = -0.92491572696505$
$a_2 = 0.70344589982355$

$H_5(z)$: (Eqn. 10.6)

$b_0 = -2.57963641586232$
$b_1 = 0.93499040211112$
$b_2 = 0.00000000000000$
$a_1 = -0.80713143885592$
$a_2 = 0.34797343307832$

$H_6(z)$: (Eqn. 10.7)

$b_0 = 3.43132602728197$
$b_1 = -0.51115438289390$
$b_2 = 0.00000000000000$
$a_1 = -0.83311810992667$
$a_2 = 0.19863461087104$

$C = 0.06393441227839$ (Eqn. 10.8)

The quantized transfer function sections are as follows:

$H_{1qt}(z)$: (Eqn. 10.9)

$b_0 = 0.06523895263671875000$
$b_1 = -0.05677032470703125000$
$b_2 = 0.00000000000000000000$
$a_1 = -1.93317413330078125000$
$a_2 = 0.94885253906250000000$

$H_{2qt}(z)$: (Eqn. 10.10)

$b_0 = -0.15412902832031250000$
$b_1 = 0.17726135253906250000$
$b_2 = 0.00000000000000000000$
$a_1 = -1.82919311523437500000$
$a_2 = 0.84671783447265625000$

$H_{3qt}(z)$: (Eqn. 10.11)

$b_0 = -1.13070678710937500000$
$b_1 = 0.97778320312500000000$
$b_2 = 0.00000000000000000000$
$a_1 = -1.71516418457031250000$
$a_2 = 0.73812866210937500000$

$H_{4qt}(z)$: (Eqn. 10.12)

$b_0 = 0.30581665039062500000$
$b_1 = -0.47155761718750000000$
$b_2 = 0.00000000000000000000$
$a_1 = -0.92491149902343750000$
$a_2 = 0.70344543457031250000$

$H_{5qt}(z)$: (Eqn. 10.13)

$b_0 = -2.57963562011718750000$
$b_1 = 0.93498992919921875000$
$b_2 = 0.00000000000000000000$
$a_1 = -0.80712890625000000000$
$a_2 = 0.34796905517578125000$

$H_{6qt}(z)$: (Eqn. 10.14)

$b_0 = 3.43132019042968750000$
$b_1 = -0.51115417480468750000$
$b_2 = 0.00000000000000000000$
$a_1 = -0.83311462402343750000$
$a_2 = 0.19863128662109375000$

$C_{qt} = 0.06393432617187500000$ (Eqn. 10.15)

$H_{1qr}(z)$: (Eqn. 10.16)

$b_0 = 0.06523895263671875000$
$b_1 = -0.05677032470703125000$
$b_2 = 0.00000000000000000000$
$a_1 = -1.93317413330078125000$
$a_2 = 0.94885253906250000000$

$H_{2qr}(z)$: (Eqn. 10.17)

b_0 = -0.15413665771484375000
b_1 = 0.17726135253906250000
b_2 = 0.00000000000000000000
a_1 = -1.82920074462890625000
a_2 = 0.84671783447265625000

$H_{3qr}(z)$: (Eqn. 10.18)

b_0 = -1.13070678710937500000
b_1 = 0.97778320312500000000
b_2 = 0.00000000000000000000
a_1 = -1.71517181396484375000
a_2 = 0.73813629150390625000

$H_{4qr}(z)$: (Eqn. 10.19)

b_0 = 0.30582427978515625000
b_1 = -0.47156524658203125000
b_2 = 0.00000000000000000000
a_1 = -0.92491912841796875000
a_2 = 0.70344543457031250000

$H_{5qr}(z)$: (Eqn. 10.20)

b_0 = -2.57963562011718750000
b_1 = 0.93489892919921875000
b_2 = 0.00000000000000000000
a_1 = -0.80712890625000000000
a_2 = 0.34797668457031250000

$H_{6qr}(z)$: (Eqn. 10.21)

b_0 = 3.43132781982421875000
b_1 = -0.51115417480468750000
b_2 = 0.00000000000000000000
a_1 = -0.83311462402343750000
a_2 = 0.19863128662109375000

C_{qr} = 0.06393432617187500000 (Eqn. 10.22)

	Quantization Type	
	Truncated	**Rounded**
$H_{1e}(z)$	0.00009114382776	0.00009114382776
$H_{2e}(z)$	0.00007593673320	0.00021975926777
$H_{3e}(z)$	0.00016858260082	0.00016858260082
$H_{4e}(z)$	0.00002390748391	0.00001879255585
$H_{5e}(z)$	0.00002893581776	0.00003311780680
$H_{6e}(z)$	0.00003868085915	0.00002255275345
C_q	0.00000008610651	0.00000008610651

Table 10-2. Magnitude error calculation.

From the results of Table 10-2 in terms of majority count, both quantization types are even. The following results of this section are based on the rounding quantization type.

	COSSAP Coefficients	
$H_{1qr}(z)$	b0_COSSAP = 0	8551
	b1_COSSAP = 0	-7441
	b2_COSSAP = 0	0
	a1_COSSAP = 1	122313
	a2_COSSAP = 0	-124368
$H_{2qr}(z)$	b0_COSSAP = 0	-20203
	b1_COSSAP = 0	23234
	b2_COSSAP = 0	0
	a1_COSSAP = 1	108685
	a2_COSSAP = 0	-110981
$H_{3qr}(z)$	b0_COSSAP = -1	-17132
	b1_COSSAP = 0	128160
	b2_COSSAP = 0	0
	a1_COSSAP = 1	93739
	a2_COSSAP = 0	-96749
$H_{4qr}(z)$	b0_COSSAP = 0	40085
	b1_COSSAP = 0	-61809
	b2_COSSAP = 0	0
	a1_COSSAP = 0	121231
	a2_COSSAP = 0	-92202
$H_{5qr}(z)$	b0_COSSAP = -2	-75974
	b1_COSSAP = 0	122551
	b2_COSSAP = 0	0
	a1_COSSAP = 0	105792
	a2_COSSAP = 0	-45610
$H_{6qr}(z)$	b0_COSSAP = 3	56535
	b1_COSSAP = 0	-66998
	b2_COSSAP = 0	0
	a1_COSSAP = 0	109198
	a2_COSSAP = 0	-26035
C_{qr}	C = 0 33520	

Table 10-3. COSSAP results of multiplier coefficient transformation.

It is noted here that based on Step 9.2 of section 8.2, the number of bits assigned for integer representation was 3. But when inputting passband, transition band, and stopband signals into the COSSAP model, the respective output responses were not as expected when compared to MATLAB results. Using Step 10.1 (section 8.2) remedied this predicament. The number of bits assigned for integer representation is now 5.

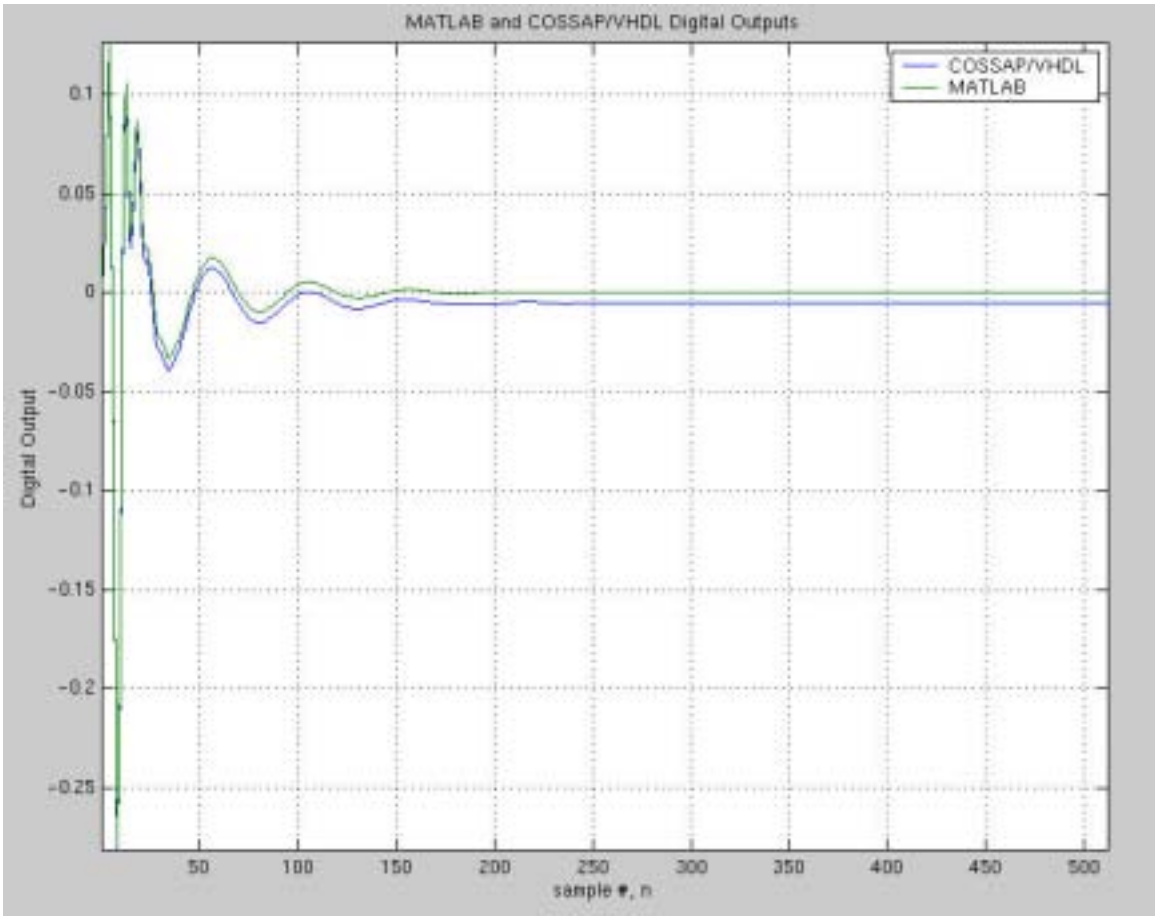


Fig. 10.5. Impulse response of ideal transfer function and quantized transfer function for *RoundMode = 0*.

COSSAP RoundMode	P_e	Dead Band Range (Fixed Value Output)	COSSAP/VHDL Peak PSD
0	$2.629023240369581 \times 10^{-5}$	$-5.126953125 \times 10^{-3}$	-11.438693 dB at 0.000000 Hz
1	$8.6072701675673 \times 10^{-7}$	$6.40869140625 \times 10^{-4}$ to $1.251220703125 \times 10^{-3}$	-26.119458 dB at 0.000000 Hz
2	$8.6072701675673 \times 10^{-7}$	$6.40869140625 \times 10^{-4}$ to $1.251220703125 \times 10^{-3}$	-26.119458 dB at 0.000000 Hz
3	$5.93578718 \times 10^{-12}$	0.00	-85.582945 dB at 4150.390625 Hz

Table 10-4. Round-off noise power results of impulse response of COSSAP multiplier round-off modes.

Based on Table 10-4, *RoundMode = 3* produces the best results in terms of smallest error. Figures 10.6 through 10.9 show the power spectral density (PSD) of the impulse response of the COSSAP round-off modes. The power spectral density of an impulse response gives the frequency response on a power

scale. It is primarily a concept issue that links frequency and time domain responses. In MATLAB, the peak PSD of the ideal impulse response is -83.507698 dB at 4003.906250 Hz.

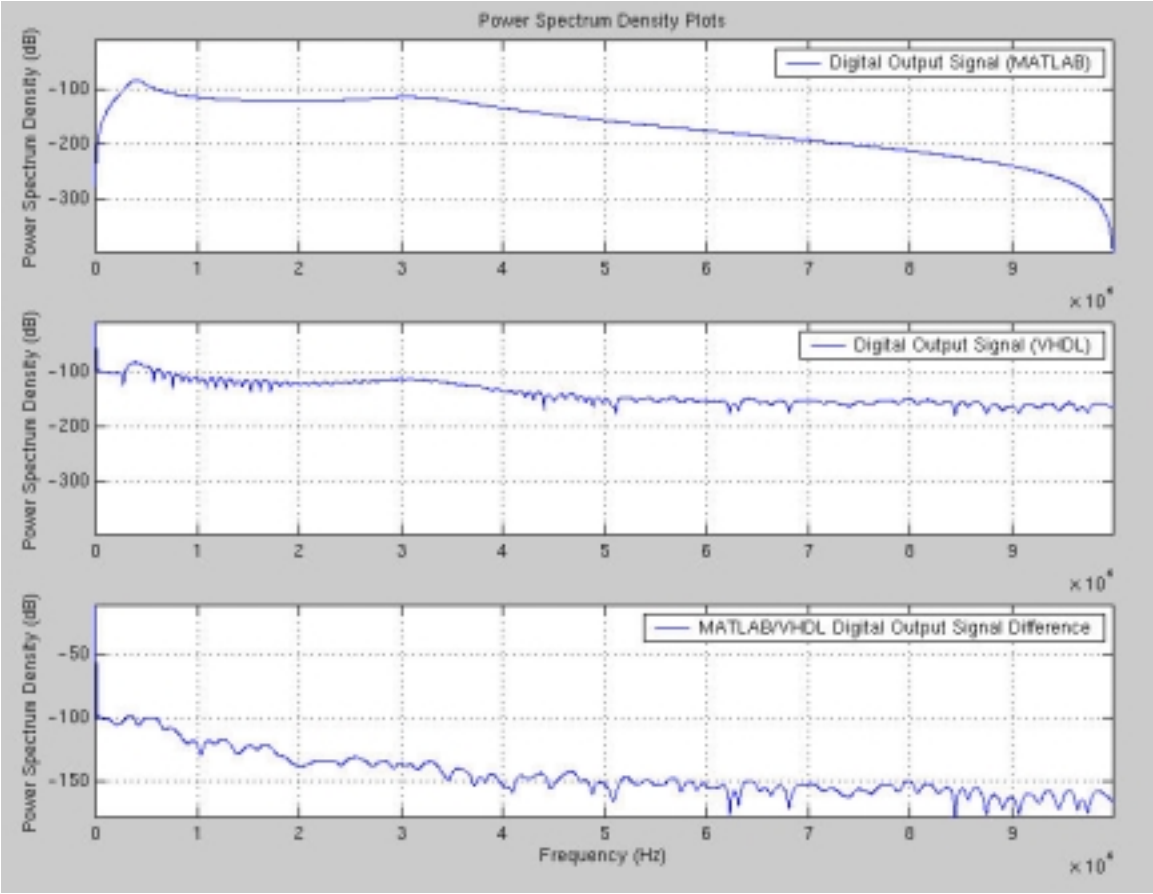


Fig. 10.6. Power spectral density results of impulse response of COSSAP *RoundMode = 0*.

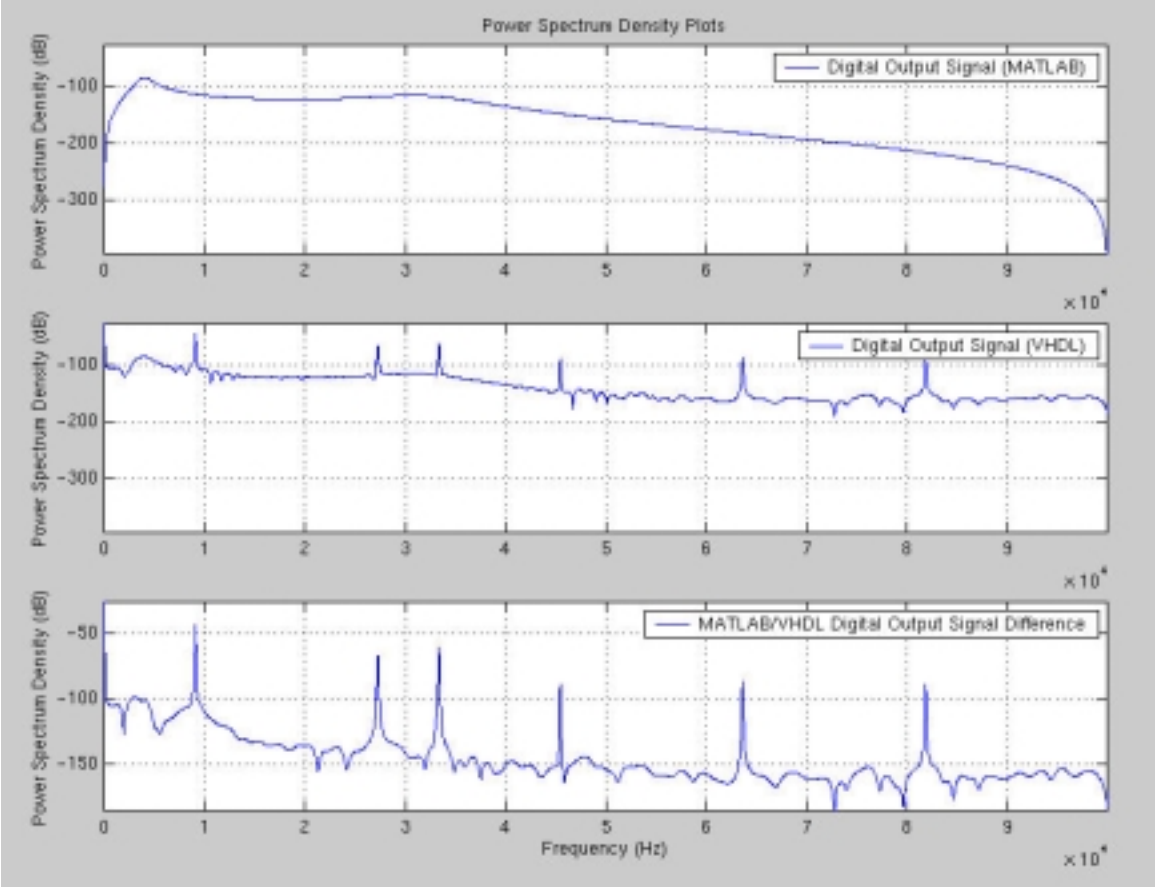


Fig. 10.7. Power spectral density results of impulse response of COSSAP *RoundMode = 1*.

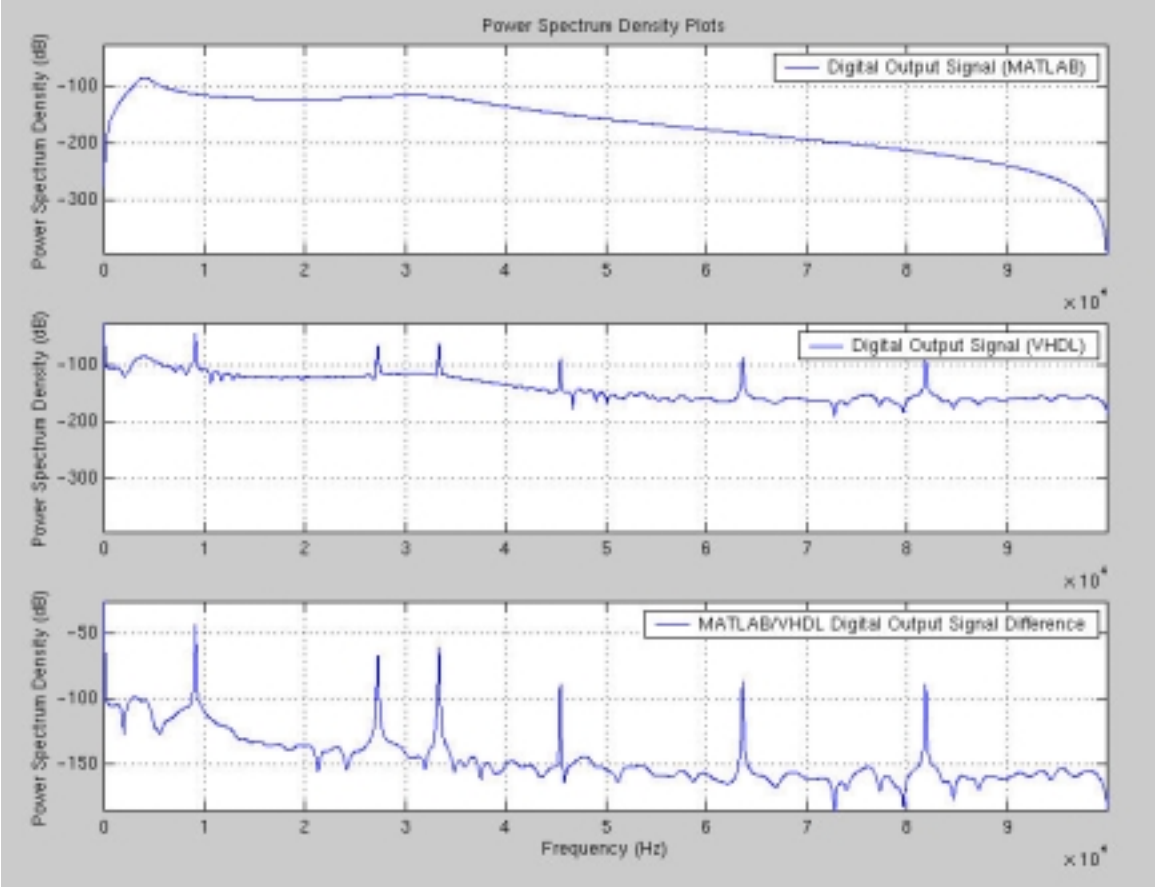


Fig. 10.8. Power spectral density results of impulse response of COSSAP *RoundMode = 2*.

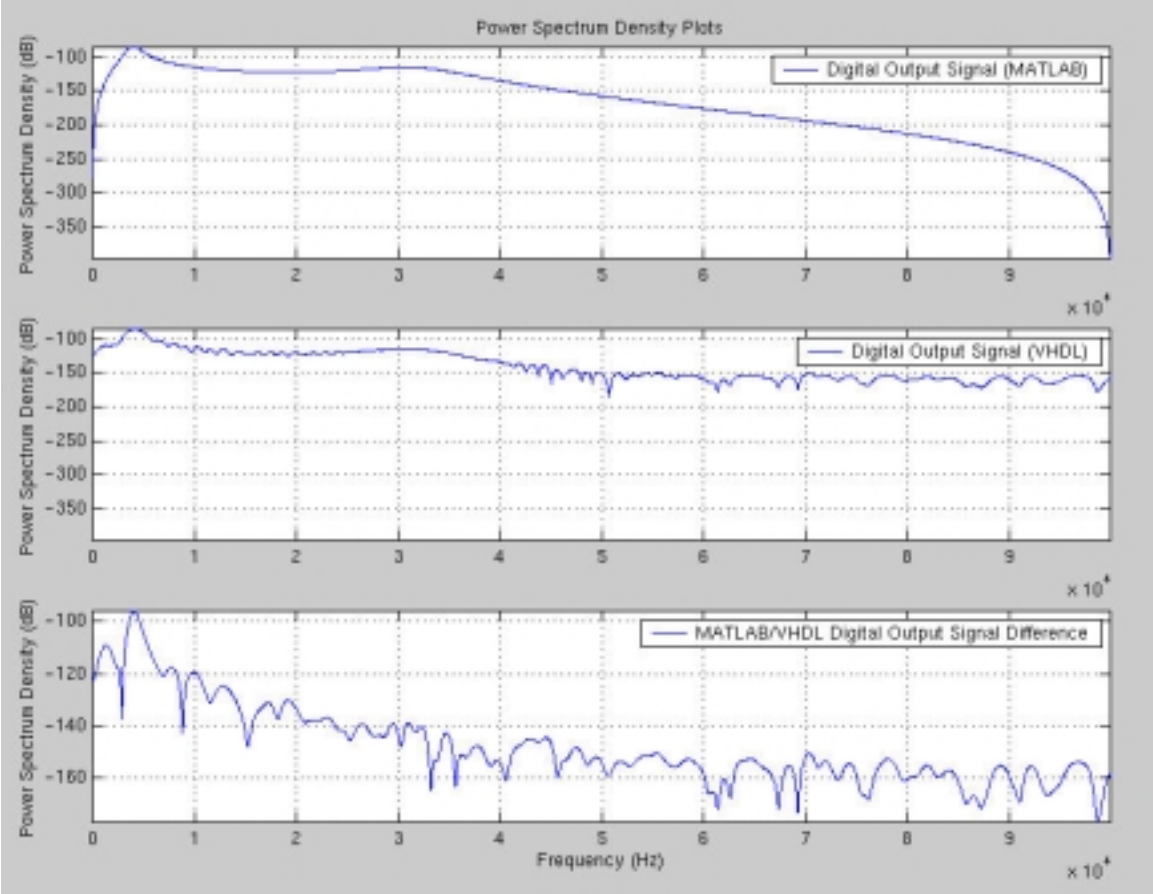


Fig. 10.9. Power spectral density results of impulse response of COSSAP *RoundMode = 3*.

10.1.1.1: Results of Validation Test #1

The passband digital input signal used is at 18kHz. The transition band input signal used is at 3kHz. The stopband digital input signal used is at 60kHz. Table 10-5 shows the tabulated results of these 3 signals.

	Round-off Noise Power (P_e)		
COSSAP RoundMode	**Passband Signal**	**Transition Band Signal**	**Stopband Signal**
0	$1.0063049989508636 \times 10^{-4}$	$1.0458186318219129 \times 10^{-4}$	$8.5058544051341 \times 10^{-5}$
1	$8.5544061745 \times 10^{-10}$	$2.51160144764 \times 10^{-9}$	$1.455644011608 \times 10^{-8}$
2	$6.24598906837 \times 10^{-9}$	$1.58560072563 \times 10^{-9}$	$1.455644011608 \times 10^{-8}$
3	$1.990957065873 \times 10^{-8}$	$4.76757901169 \times 10^{-9}$	$2.26142987468 \times 10^{-9}$

Table 10-5. Tabulated results of validation test #1.

10.1.1.2: Results of Validation Test #2

Figure 10.10 shows the digital input/output response of the unquantized bandpass Butterworth filter. The low-power passband input signal is at 18kHz. The high-power stopband input signal is at 60kHz. For the purposes of analysis, the high-power signal is 10 times greater in amplitude than the low-power signal. As can be seen from the figure, excluding the initial transient response at the output, the ideal Butterworth filter effectively removes the high-power stopband signal. Comparing the quantized filter, Figures 10.11 through 10.14 show the power spectral density plots for the four COSSAP round-off modes. Table 10-6 shows the tabulated results of the round-off noise power based on the COSSAP multiplier sub-blocks.

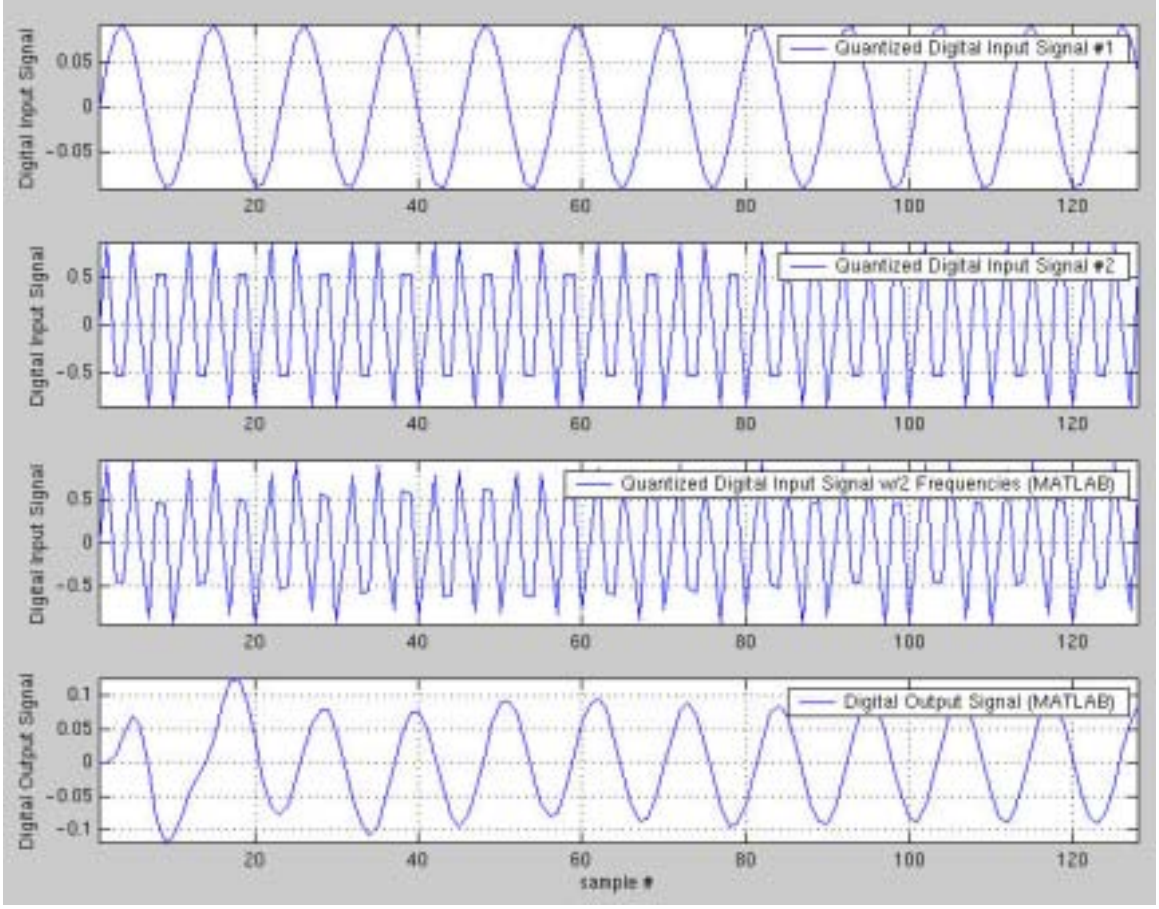


Fig. 10.10. Digital I/O response of validation test #2 for ideal Butterworth filter.

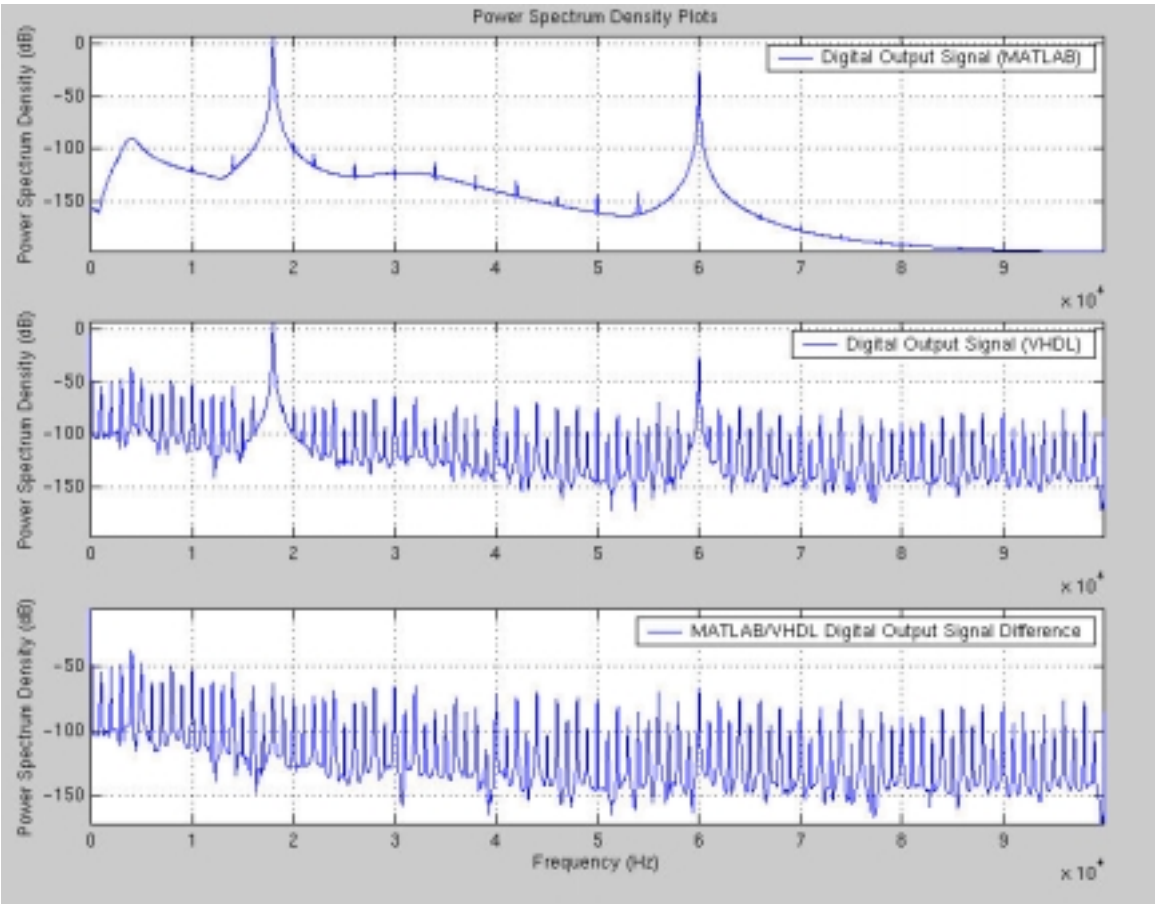


Fig. 10.11. Power spectral density plot of COSSAP *RoundMode = 0*.

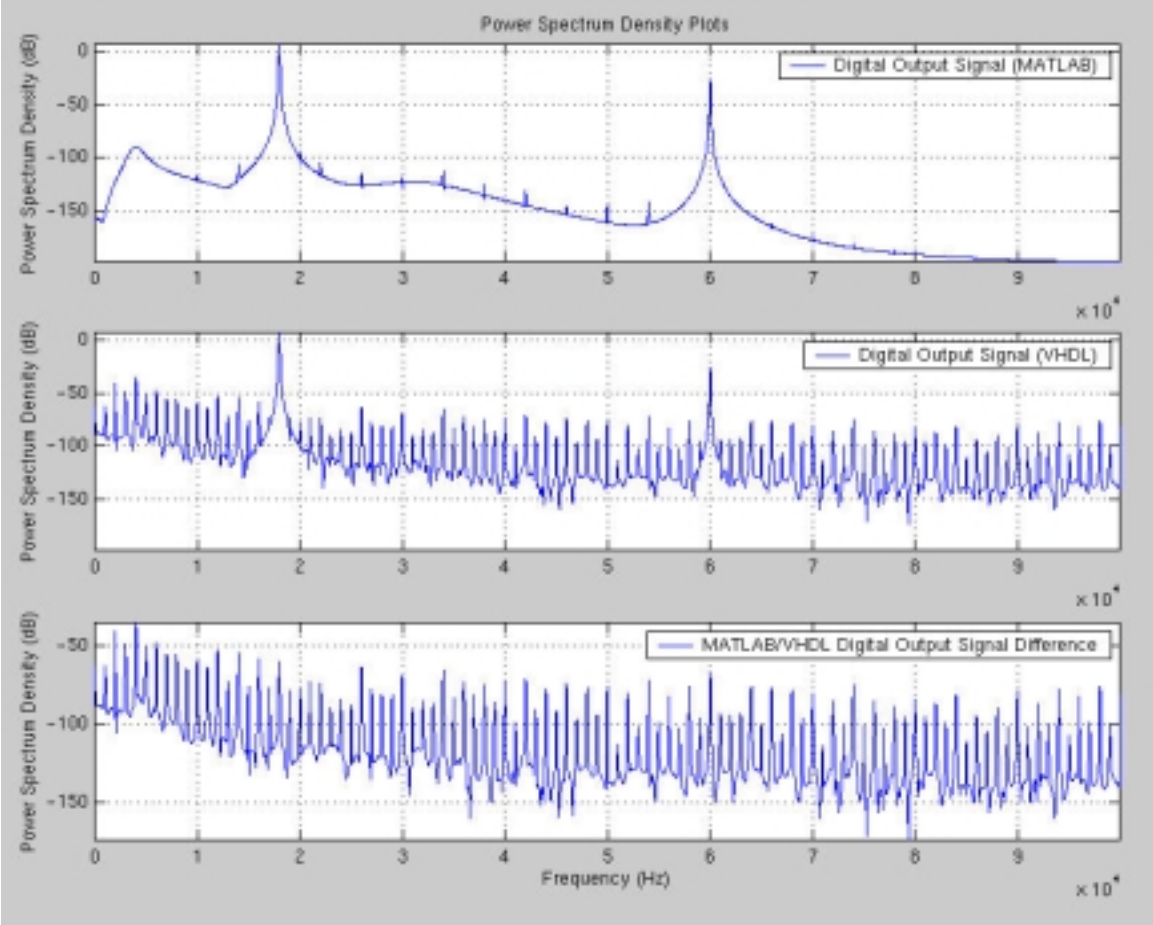


Fig. 10.12. Power spectral density plot of COSSAP *RoundMode = 1*.

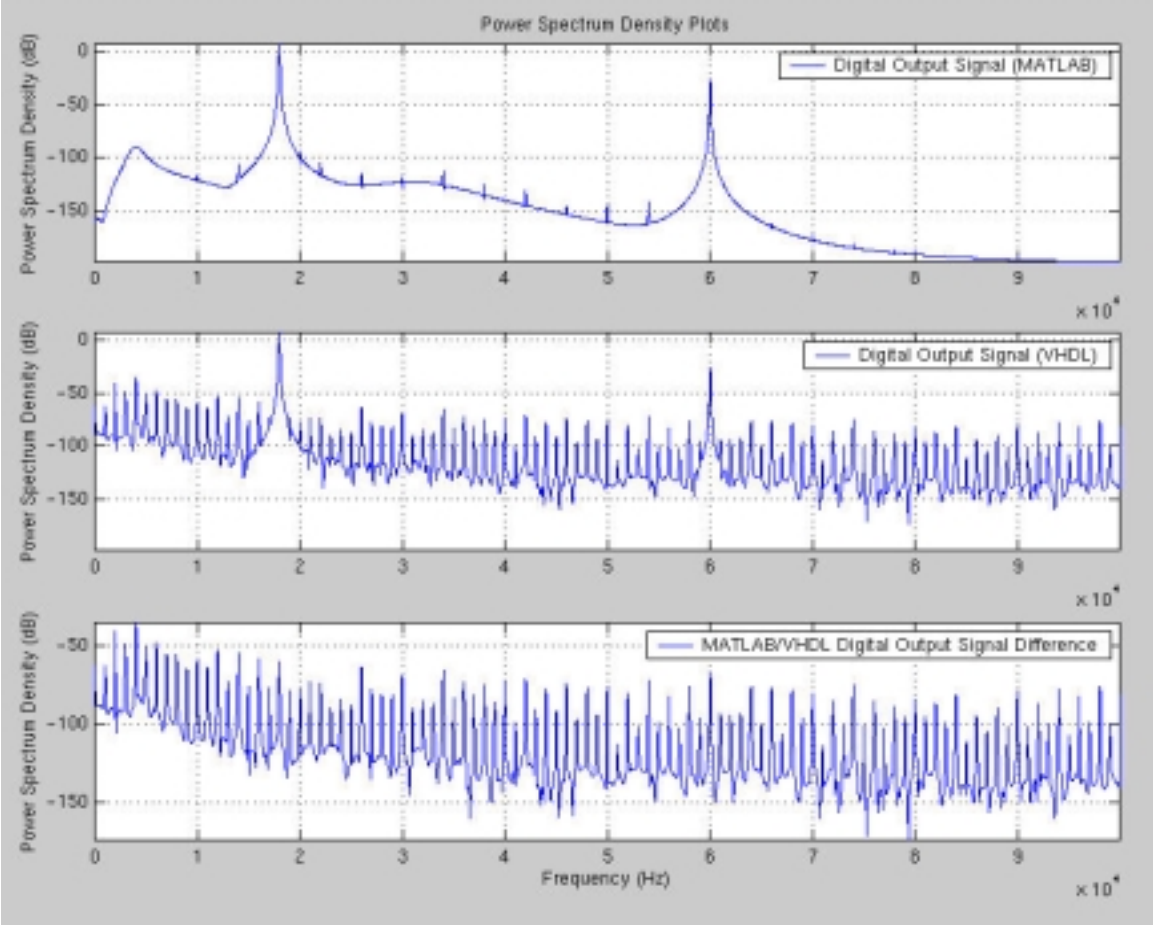


Fig. 10.13. Power spectral density plot of COSSAP *RoundMode* = *2*.

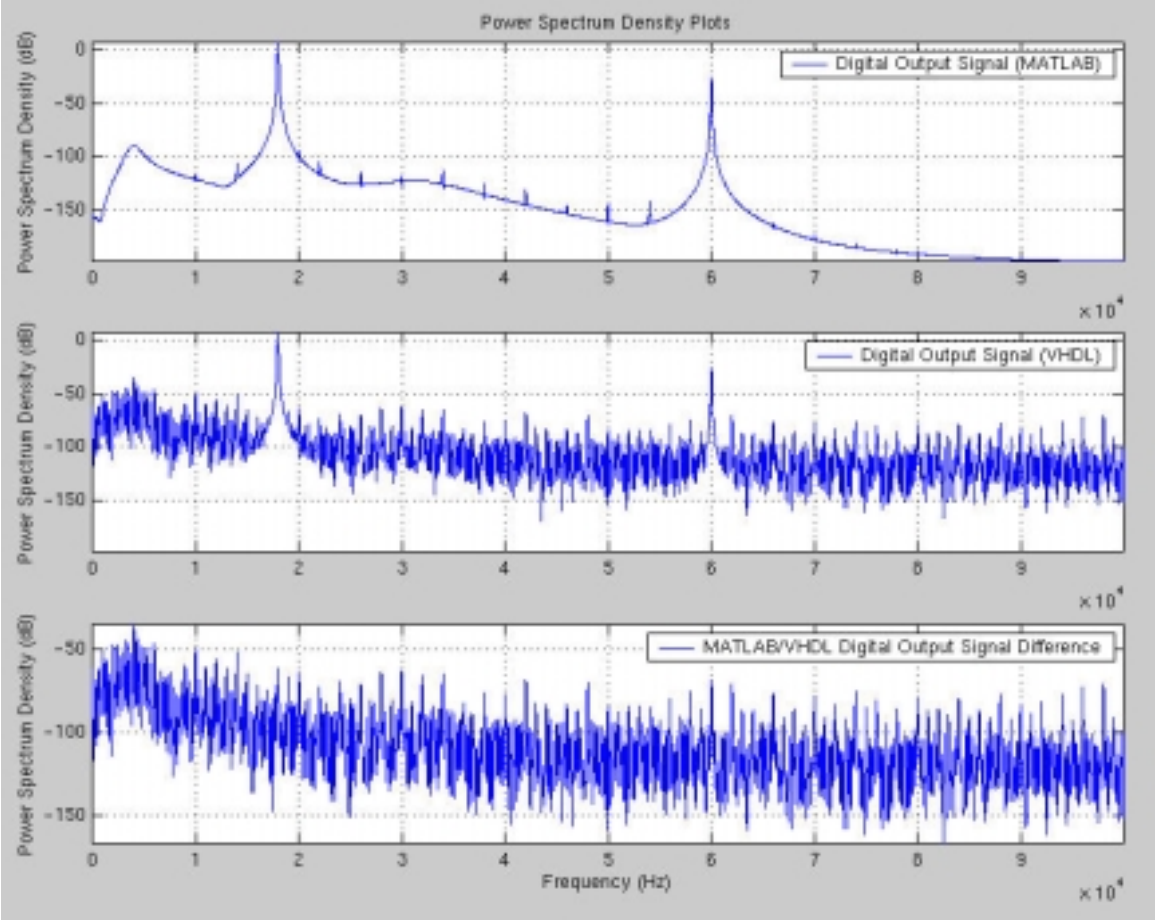


Fig. 10.14. Power spectral density plot of COSSAP *RoundMode* = *3*.

COSSAP RoundMode	P_e
0	$1.0947356406282245 \times 10^{-4}$
1	$3.7505982301 \times 10^{-10}$
2	$3.7505982301 \times 10^{-10}$
3	$1.2669903207 \times 10^{-10}$

Table 10-6. Tabulated results of 4 COSSAP round-off modes.

Based on Table 10-6, *RoundMode* = *3* produces the best results in terms of smallest error.

10.1.2: Cascade Structure of Butterworth Bandpass Filter Results

The unquantized transfer function sections are as follows:

$$H(z) = H_1(z)H_2(z)H_3(z)H_4(z)H_5(z)H_6(z) \qquad \text{(Eqn. 10.23)}$$

$H_1(z)$: (Eqn. 10.24)

$b_0 = 0.05553049275961$
$b_1 = -0.11106109747729$
$b_2 = 0.05553078292203$
$a_1 = -1.93317570078060$
$a_2 = 0.94885440559193$

$H_2(z)$: (Eqn. 10.25)

$b_0 = 1.81457724155634$
$b_1 = -3.62352244923983$
$b_2 = 1.80895103635644$
$a_1 = -1.82919846105757$
$a_2 = 0.84672134723855$

$H_3(z)$: (Eqn. 10.26)

$b_0 = 4.61316718780215$
$b_1 = -9.24064329348397$
$b_2 = 4.62749089495188$
$a_1 = -1.71517028232064$
$a_2 = 0.73813390470860$

$H_4(z)$: (Eqn. 10.27)

$b_0 = 0.17090301137699$
$b_1 = 0.34127557971965$
$b_2 = 0.17037311730489$
$a_1 = -0.92491572696505$
$a_2 = 0.70344589982355$

$H_5(z)$: (Eqn. 10.28)

$b_0 = 0.15679633173001$
$b_1 = 0.31359297968004$
$b_2 = 0.15679715112736$
$a_1 = -0.80713143885592$
$a_2 = 0.34797343307832$

$H_6(z)$: (Eqn. 10.29)

$b_0 = 0.14799595274850$
$b_1 = 0.29645095189930$
$b_2 = 0.14845547360649$
$a_1 = -0.83311810992667$
$a_2 = 0.19863461087104$

The quantized transfer function sections are as follows:

$H_{1qt}(z)$: (Eqn. 10.30)

$b_0 = 0.0555114746093750000$
$b_1 = -0.1110534667968750000$
$b_2 = 0.0555114746093750000$
$a_1 = -1.9331665039062500000$
$a_2 = 0.9488525390625000000$

$H_{2qt}(z)$: (Eqn. 10.31)

$b_0 = 1.81457519531250000000$
$b_1 = -3.62350463867187500000$
$b_2 = 1.80892944335937500000$
$a_1 = -1.82919311523437500000$
$a_2 = 0.84671020507812500000$

$H_{3qt}(z)$: (Eqn. 10.32)

$b_0 = 4.61315917968750000000$
$b_1 = -9.24063110351562500000$
$b_2 = 4.62747192382812500000$
$a_1 = -1.71514892578125000000$
$a_2 = 0.73812866210937500000$

$H_{4qt}(z)$: (Eqn. 10.33)

$b_0 = 0.17089843750000000000$
$b_1 = 0.34124755859375000000$
$b_2 = 0.17034912109375000000$
$a_1 = -0.92489624023437500000$
$a_2 = 0.70343017578125000000$

$H_{5qt}(z)$: (Eqn. 10.34)

$b_0 = 0.15676879882812500000$
$b_1 = 0.31356811523437500000$
$b_2 = 0.15676879882812500000$
$a_1 = -0.80712890625000000000$
$a_2 = 0.34796142578125000000$

$H_{6qt}(z)$: (Eqn. 10.35)

$b_0 = 0.14797973632812500000$
$b_1 = 0.29644775390625000000$
$b_2 = 0.14843750000000000000$
$a_1 = -0.83309936523437500000$
$a_2 = 0.19860839843750000000$

$H_{1qr}(z)$: (Eqn. 10.36)

$b_0 = 0.05554199218750000000$
$b_1 = -0.11105346679687500000$
$b_2 = 0.05554199218750000000$
$a_1 = -1.93316650390625000000$
$a_2 = 0.94885253906250000000$

$H_{2qr}(z)$: (Eqn. 10.37)

$b_0 = 1.81457519531250000000$
$b_1 = -3.62353515625000000000$
$b_2 = 1.80895996093750000000$
$a_1 = -1.82919311523437500000$
$a_2 = 0.84671020507812500000$

$H_{3qr}(z)$: (Eqn. 10.38)

b_0 = 4.61315917968750000000
b_1 = -9.24063110351562500000
b_2 = 4.62750244140625000000
a_1 = -1.71517944335937500000
a_2 = 0.73812866210937500000

$H_{4qr}(z)$: (Eqn. 10.39)

b_0 = 0.17089843750000000000
b_1 = 0.34127807617187500000
b_2 = 0.17037963867187500000
a_1 = -0.92492675781250000000
a_2 = 0.70346069335937500000

$H_{5qr}(z)$: (Eqn. 10.40)

b_0 = 0.15679931640625000000
b_1 = 0.31359863281250000000
b_2 = 0.15679931640625000000
a_1 = -0.80712890625000000000
a_2 = 0.34796142578125000000

$H_{6qr}(z)$: (Eqn. 10.41)

b_0 = 0.14801025390625000000
b_1 = 0.29644775390625000000
b_2 = 0.14846801757812500000
a_1 = -0.83312988281250000000
a_2 = 0.19863891601562500000

	Quantization Type	
	Truncated	**Rounded**
$H_{1e}(z)$	0.00476283586785	0.00479749621261
$H_{2e}(z)$	0.00047914253420	0.00047950739909
$H_{3e}(z)$	0.00064403036298	0.00068575687106
$H_{4e}(z)$	0.00030847232951	0.00012322898054
$H_{5e}(z)$	0.00015119586304	0.00004029047005
$H_{6e}(z)$	0.00010023502175	0.00009783758146

Table 10-7. Magnitude error calculation.

From the results of Table 10-7 in terms of majority count, both quantization types are even. The following results of this section are based on the rounding quantization type.

	COSSAP Coefficients
$H_{1qr}(z)$	b0_COSSAP = 0 1820 b1_COSSAP = 0 -3639 b2_COSSAP = 0 1820 a1_COSSAP = 1 30578 a2_COSSAP = 0 -31092
$H_{2qr}(z)$	b0_COSSAP = 1 26692 b1_COSSAP = -3 -20432 b2_COSSAP = 1 26508 a1_COSSAP = 1 27171 a2_COSSAP = 0 -27745
$H_{3qr}(z)$	b0_COSSAP = 4 20092 b1_COSSAP = -9 -7885 b2_COSSAP = 4 20562 a1_COSSAP = 1 23435 a2_COSSAP = 0 -24187
$H_{4qr}(z)$	b0_COSSAP = 0 5600 b1_COSSAP = 0 11183 b2_COSSAP = 0 5583 a1_COSSAP = 0 30308 a2_COSSAP = 0 -23051
$H_{5qr}(z)$	b0_COSSAP = 0 5138 b1_COSSAP = 0 10276 b2_COSSAP = 0 5138 a1_COSSAP = 0 26448 a2_COSSAP = 0 -11402
$H_{6qr}(z)$	b0_COSSAP = 0 4850 b1_COSSAP = 0 9714 b2_COSSAP = 0 4865 a1_COSSAP = 0 27300 a2_COSSAP = 0 -6509

Table 10-8. COSSAP results of multiplier coefficient transformation.

Step 9.3.1 of section 8.2 was used for the number of integer bits configured to the adder sub-blocks of each cascade stage. Because there are 6 cascade stages, the number of integer bits configured to the adder sub-blocks of each stage was 1, 3, 4, 3, 2, and 1.

COSSAP RoundMode	P_e	Dead Band Range (Fixed Value Output)	COSSAP/VHDL Peak PSD
0	$9.87823501268909 \times 10^{-6}$	$-3.1795501708984 \times 10^{-3}$ to $-3.171920776367 \times 10^{-3}$	-15.603663 dB at 0.000000 Hz
1	$1.167917120788 \times 10^{-8}$	$-4.8446655273438 \times 10^{-4}$ to $1.2016296386719 \times 10^{-4}$	-44.072141 dB at 0.000000 Hz
2	$1.167917120788 \times 10^{-8}$	$-4.8446655273438 \times 10^{-4}$ to $1.2016296386719 \times 10^{-4}$	-44.072141 dB at 0.000000 Hz
3	$1.82561562047 \times 10^{-9}$	$4.3869018554688 \times 10^{-5}$ to $4.57763671875 \times 10^{-5}$	-52.518877 dB at 0.000000 Hz

Table 10-9. Round-off noise power results of impulse response of COSSAP multiplier round-off modes.

Based on Table 10-9, *RoundMode = 3* produces the best results in terms of smallest error. Figures 10.15 through 10.18 show the power spectral density of the impulse response of the COSSAP round-off modes. Using MATLAB as the ideal model, the peak PSD of the impulse response is -83.507698 dB at 4003.906250 Hz.

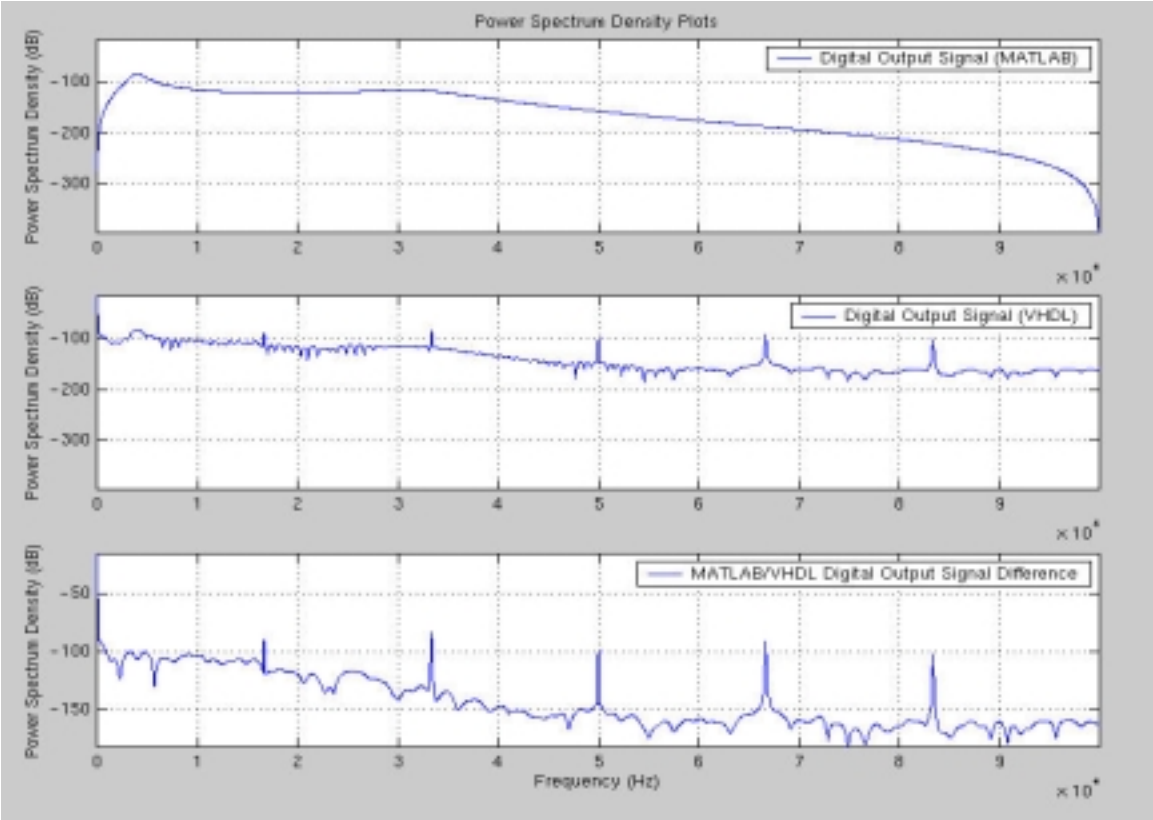


Fig. 10.15. Power spectral density results of impulse response of COSSAP *RoundMode = 0*.

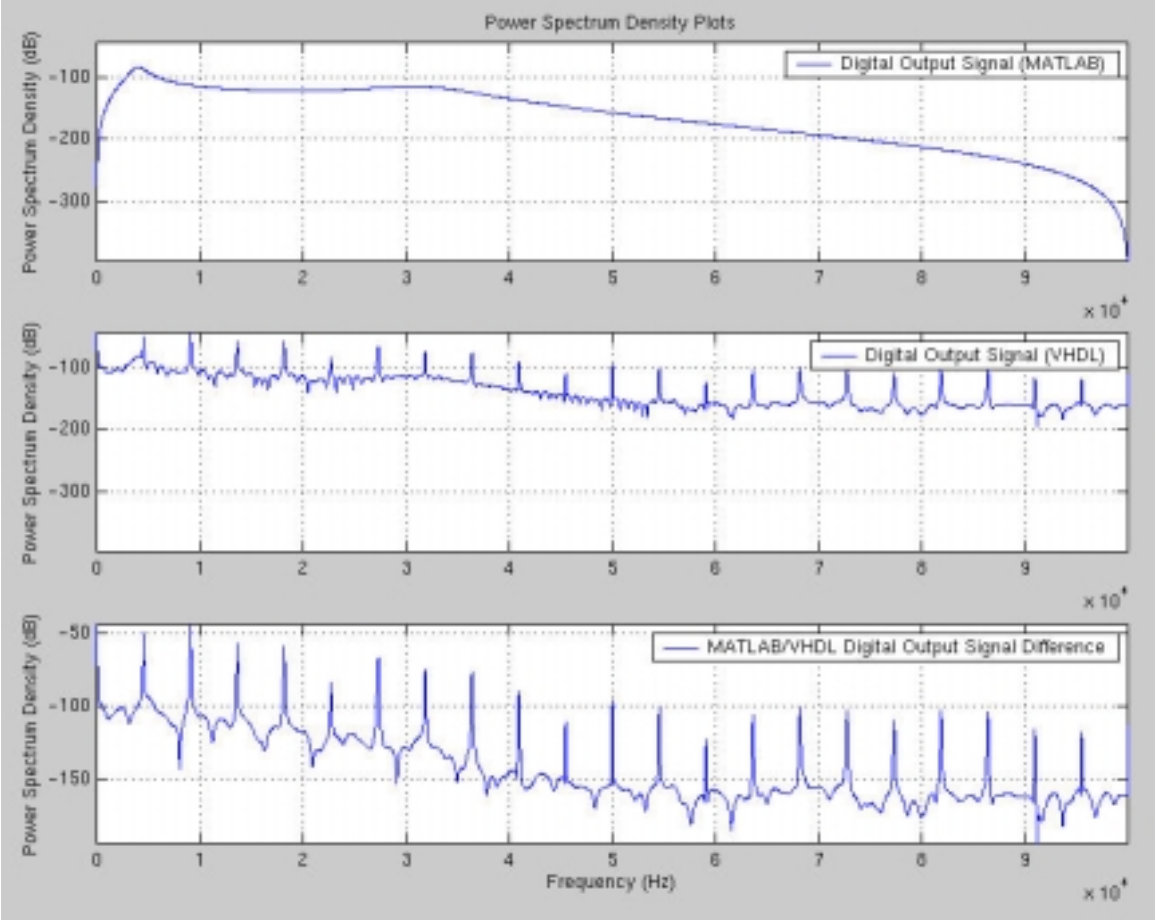


Fig. 10.16. Power spectral density results of impulse response of COSSAP *RoundMode = 1*.

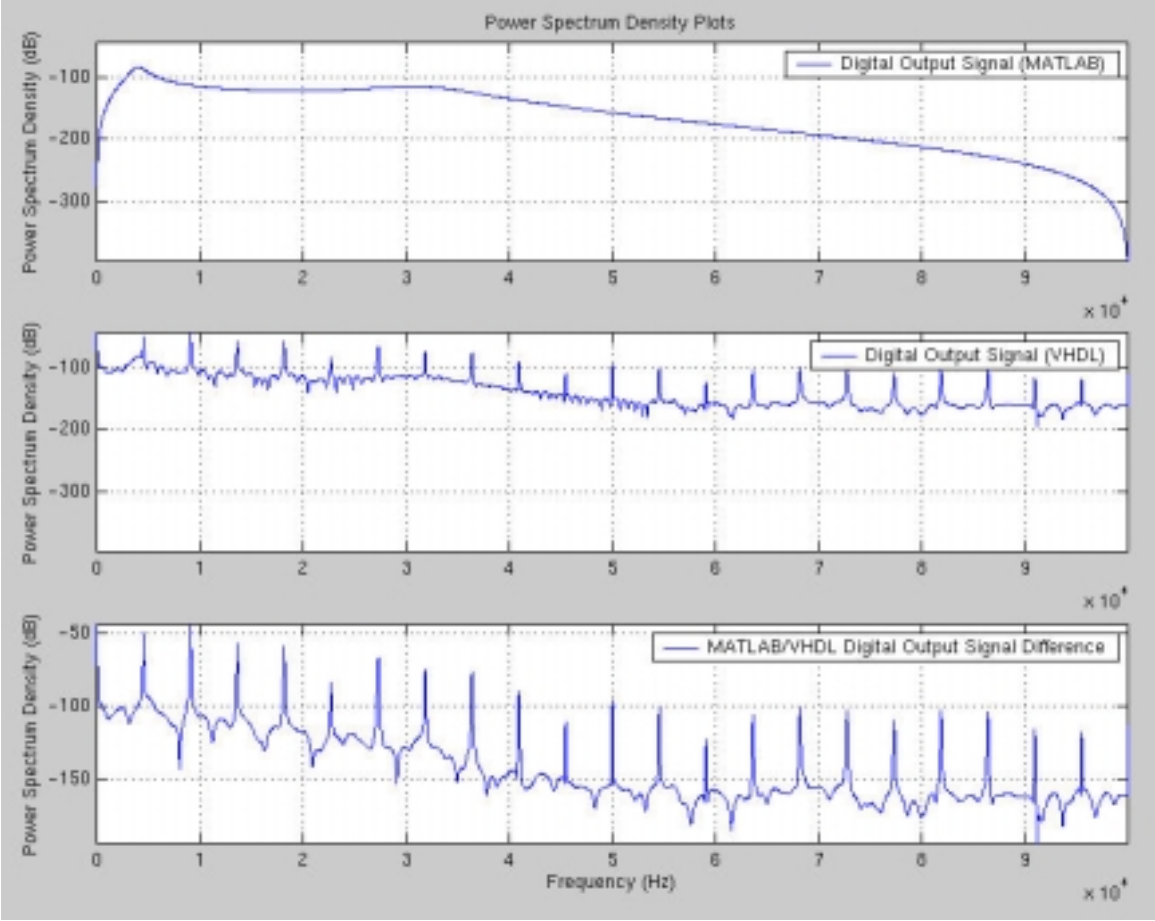


Fig. 10.17. Power spectral density results of impulse response of COSSAP *RoundMode = 2*.

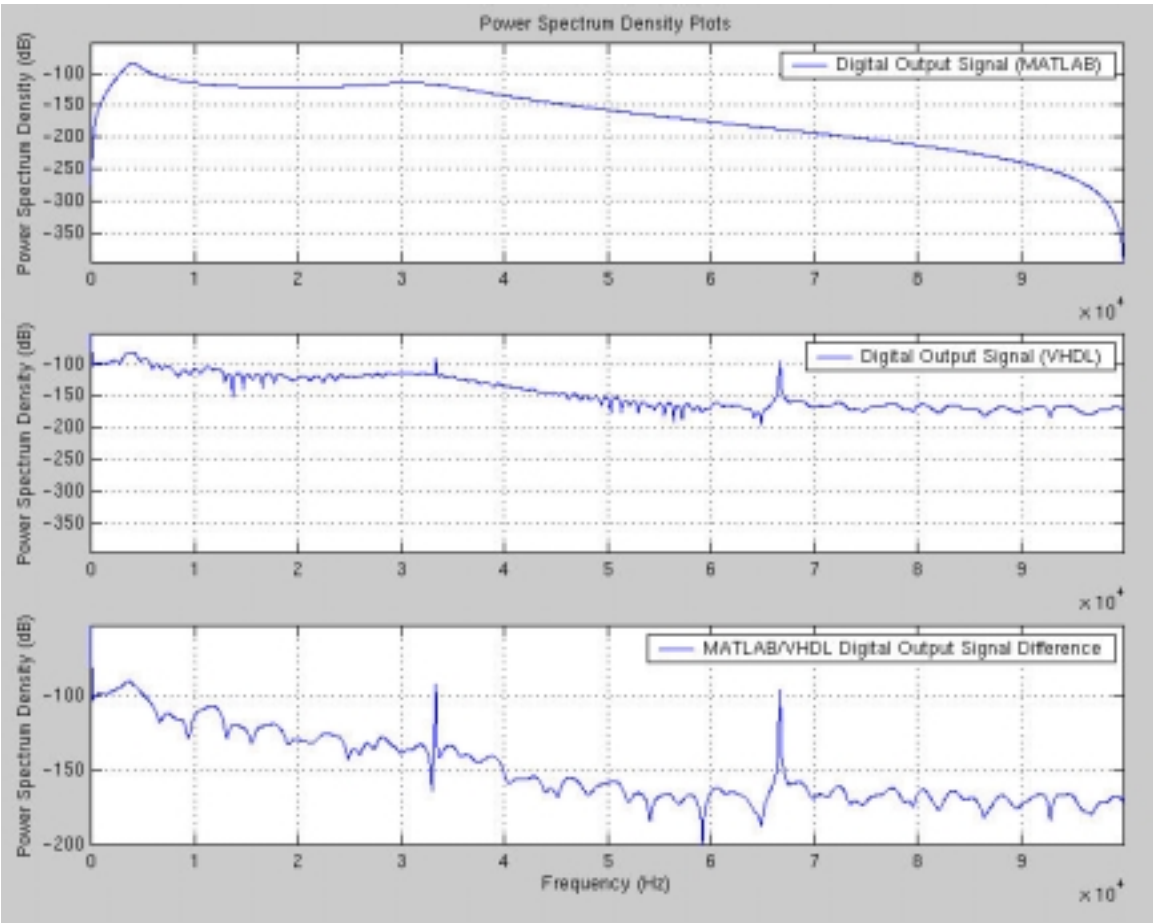


Fig. 10.18. Power spectral density results of impulse response of COSSAP *RoundMode = 3*.

10.1.2.1: Results of Validation Test #1

The digital input signals used in section 10.1.1.1 were also used in this section. Table 10-10 shows these results.

	Round-off Noise Power (P_e)		
COSSAP RoundMode	**Passband Signal**	**Transition Band Signal**	**Stopband Signal**
0	$8.55073884480357 \times 10^{-6}$	$8.26422700098962 \times 10^{-6}$	$8.286744428225 \times 10^{-6}$
1	$3.93673678660 \times 10^{-9}$	$4.26600835706 \times 10^{-9}$	$2.3708123255 \times 10^{-10}$
2	$3.93713730783 \times 10^{-9}$	$4.26842773179 \times 10^{-9}$	$2.3708123255 \times 10^{-10}$
3	$2.98706502689 \times 10^{-9}$	$3.76628415968 \times 10^{-9}$	$2.88406919 \times 10^{-12}$

Table 10-10. Tabulated results of validation test #1.

10.1.2.2: Results of Validation Test #2

The digital input signals used in section 10.1.1.2 were also used in this section.

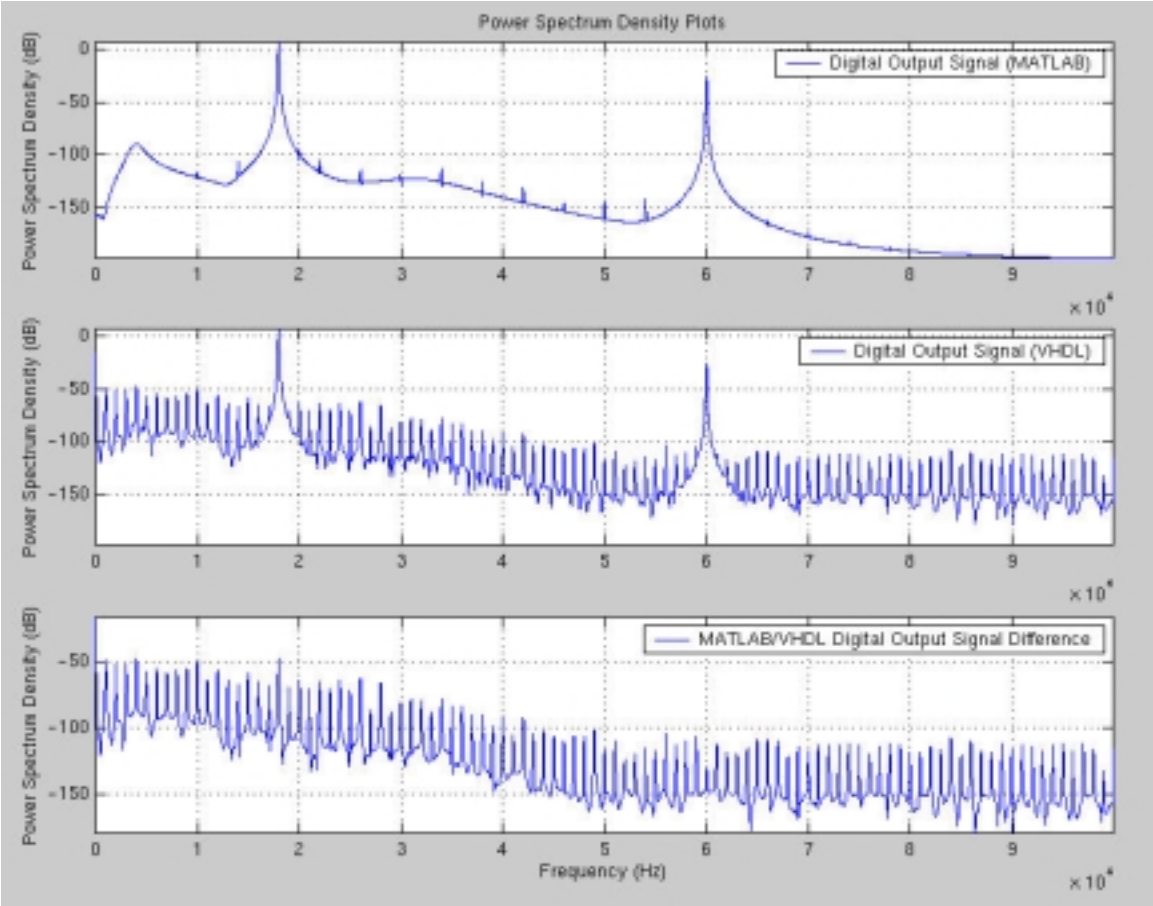


Fig. 10.19. Power spectral density of COSSAP *RoundMode* = *0*.

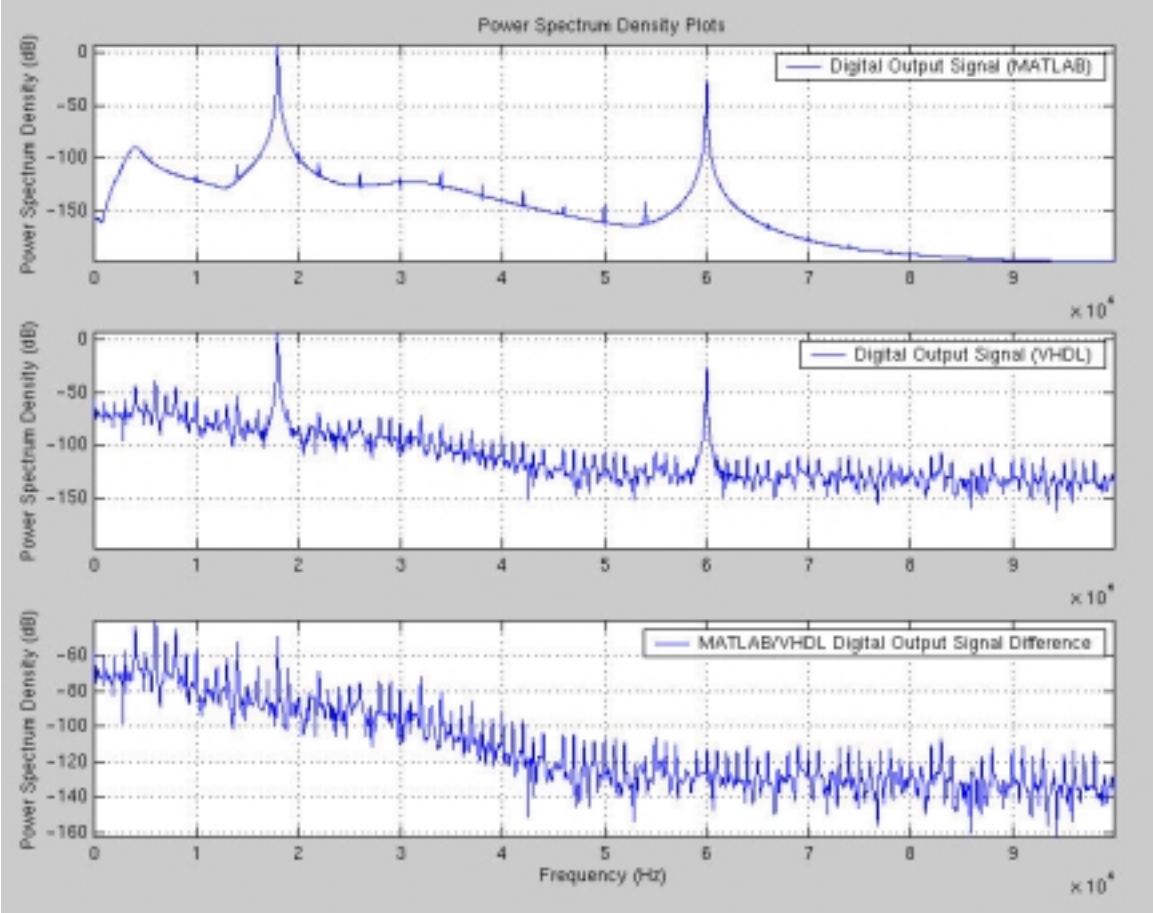


Fig. 10.20. Power spectral density of COSSAP *RoundMode* = *1*.

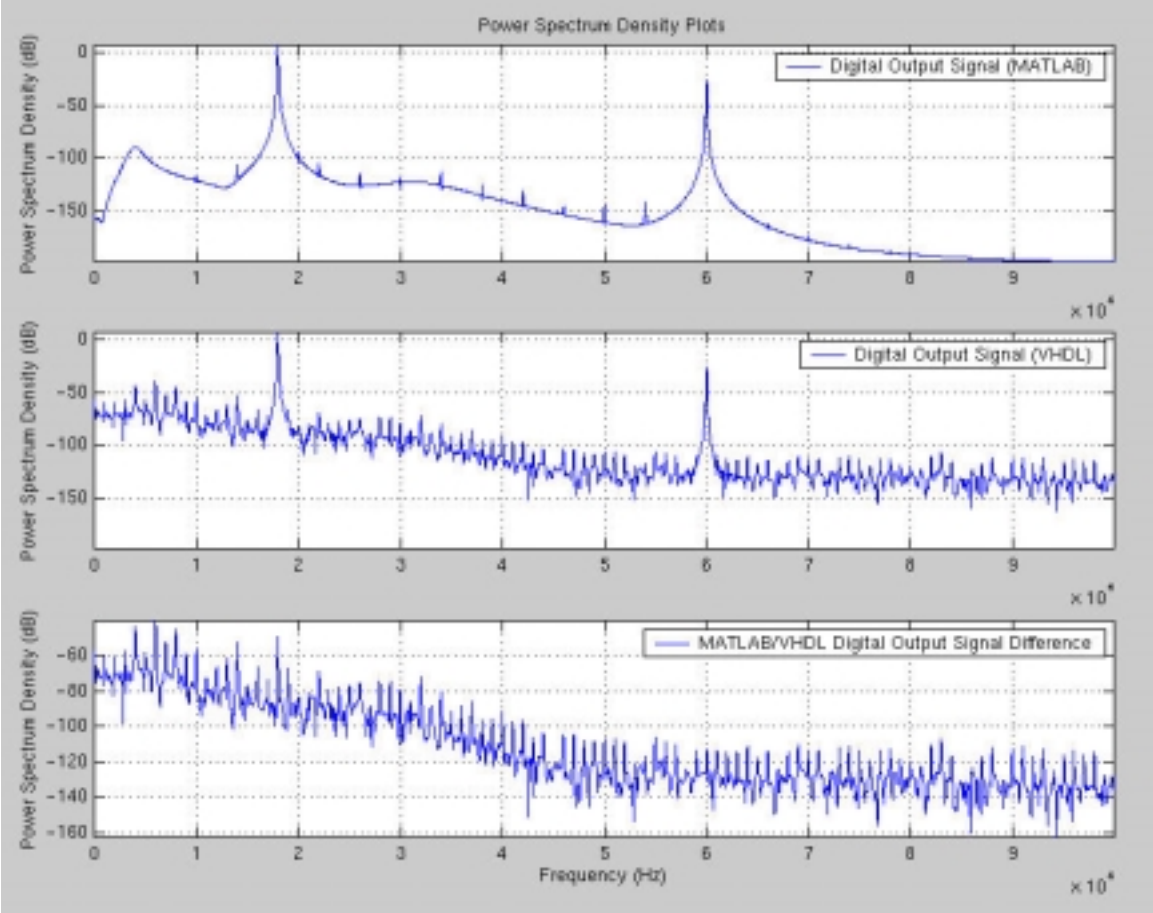


Fig. 10.21. Power spectral density of COSSAP *RoundMode* = *2*.

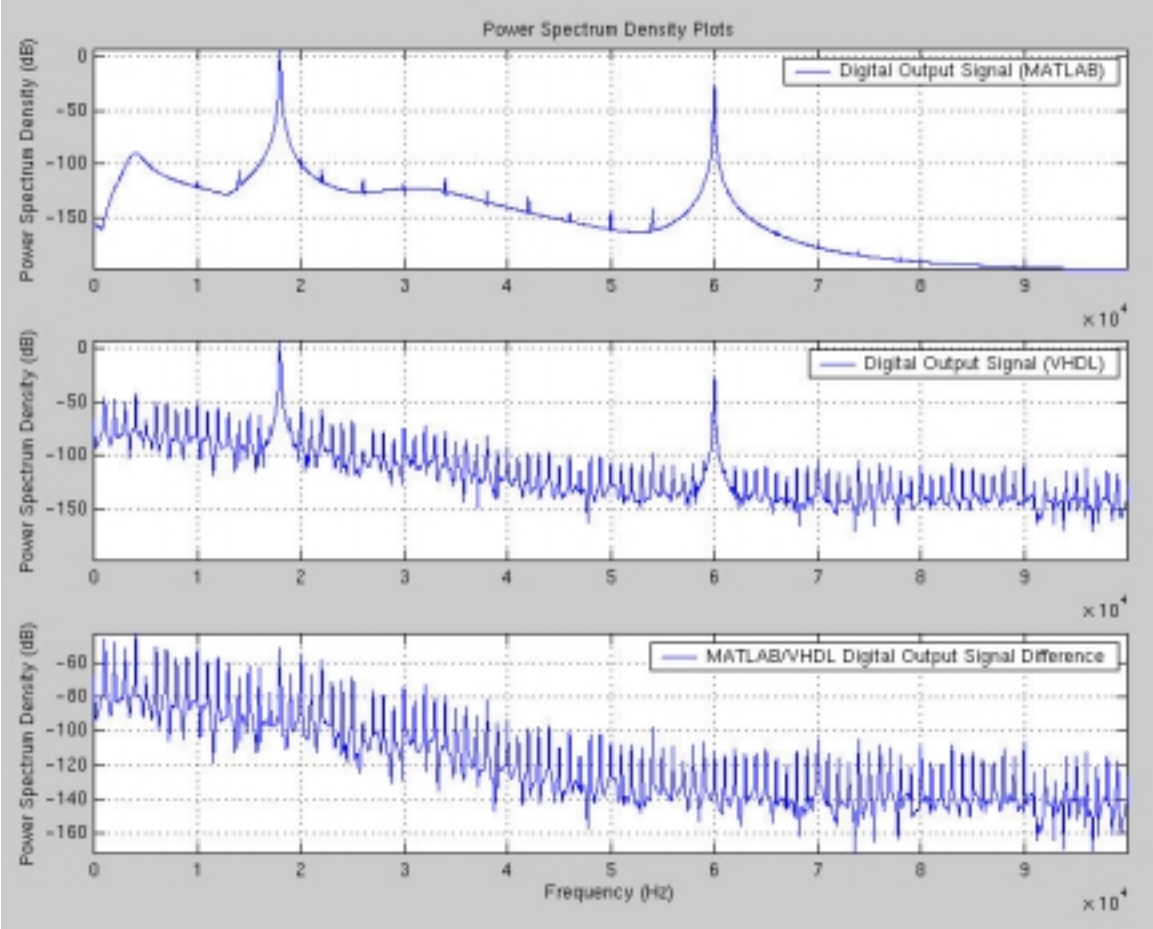


Fig. 10.22. Power spectral density of COSSAP *RoundMode = 3*.

COSSAP RoundMode	P_e
0	$8.49324543927961 \times 10^{-6}$
1	$2.8491824904 \times 10^{-10}$
2	$2.8491824904 \times 10^{-10}$
3	$2.692591290 \times 10^{-12}$

Table 10-11. Tabulated results of 4 COSSAP round-off modes.

Based on the results of Table 10-11, the best COSSAP multiplier sub-block round-off mode is *RoundMode = 3*.

10.2: Digital Video Bandwidth Results

The sampling frequency used for this design was 10MHz. The cut-off frequency (-3dB point) is 2MHz. The stopband frequency (-40dB point) is 2.144078MHz. This design is a 20-bit bandpass digital filter design and the design type is Chebyshev Type II. The MATLAB floating-point representations of the multiplier coefficients, taken to 14 significant places, are as follows:

b_0 = 0.04588208205408
b_1 = 0.13656264939956
b_2 = 0.34751449611781
b_3 = 0.62954494594627
b_4 = 0.95796852137520
b_5 = 1.21270173589814
b_6 = 1.31276566337727
b_7 = 1.21270173589814
b_8 = 0.95796852137519
b_9 = 0.62954494594627
b_{10} = 0.34751449611781
b_{11} = 0.13656264939956
b_{12} = 0.04588208205408
a_1 =-0.28455481395673
a_2 = 2.35933880696702
a_3 = 0.10701461849930
a_4 = 2.11227062100930
a_5 = 0.65506585946252
a_6 = 1.04098525446599
a_7 = 0.47035521092297
a_8 = 0.31587427927210
a_9 = 0.13173314200725
a_{10} = 0.04984113806814
a_{11} = 0.01308324636888
a_{12} = 0.00210716187263

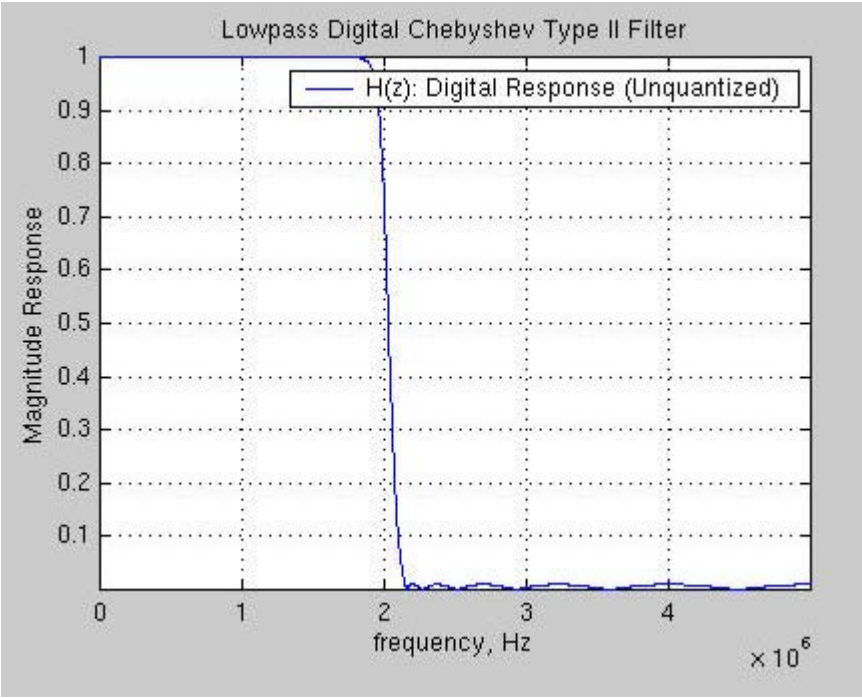


Fig. 10.23. MATLAB magnitude response of lowpass Chebyshev Type II filter.

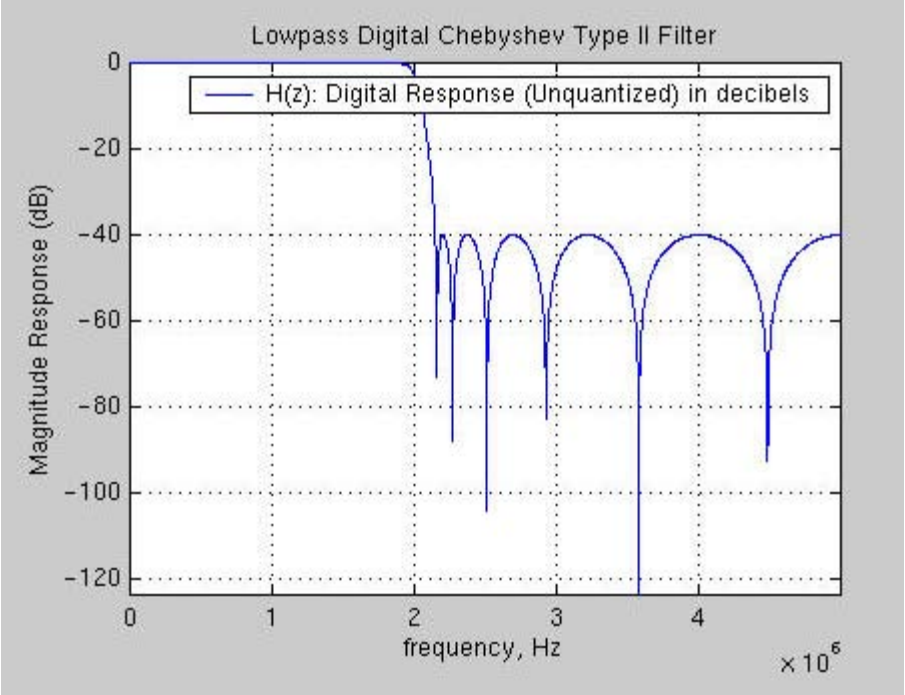


Fig. 10.24. MATLAB magnitude response of lowpass Chebyshev Type II filter (dB).

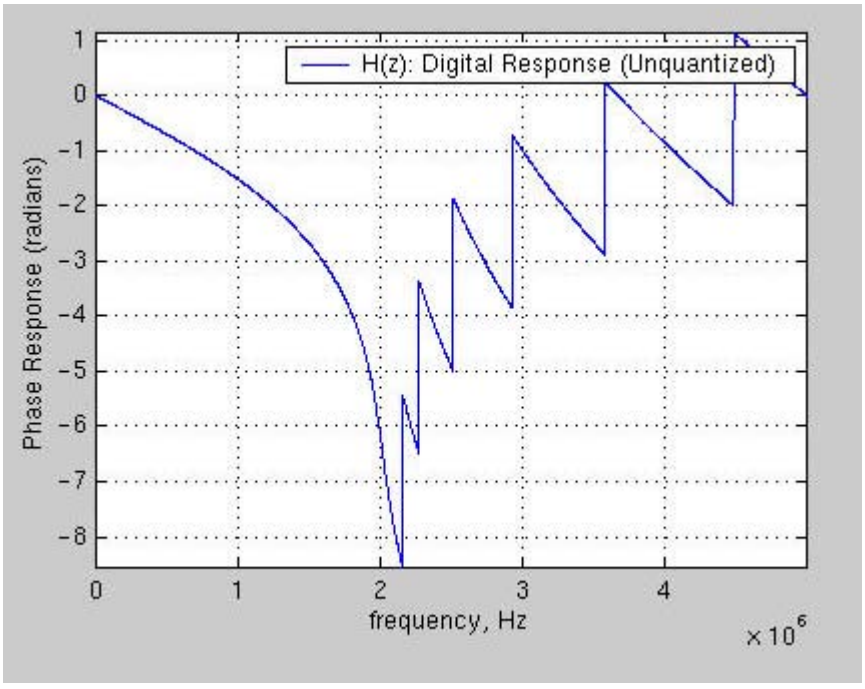


Fig. 10.25. MATLAB phase response of lowpass Chebyshev Type II filter (radians).

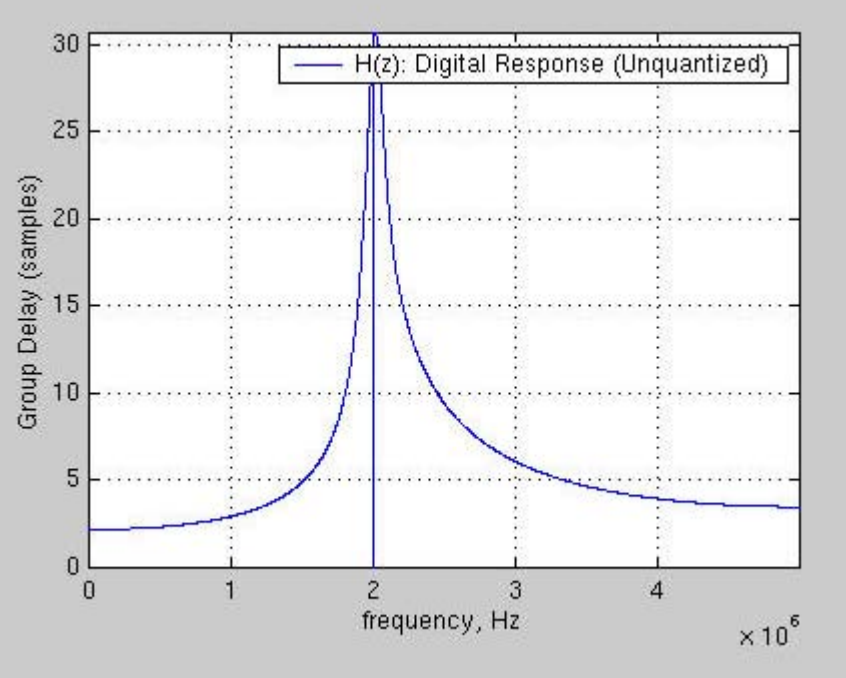


Fig. 10.26. MATLAB group delay response of lowpass Chebyshev Type II filter.

10.2.1: Parallel Structure of Chebyshev Type II Lowpass Filter Results

The unquantized transfer function sections are as follows:

$H_1(z)$: (Eqn. 10.42)

b_0 = 0.08824283594937
b_1 = 0.08910645252623
b_2 = 0.00000000000000
a_1 = -0.57512824697051
a_2 = 0.90116474888622

$H_2(z)$: (Eqn. 10.43)

b_0 = 0.69626630834828
b_1 = -0.16110151137575
b_2 = 0.00000000000000
a_1 = -0.42850343365581
a_2 = 0.71817606931620

$H_3(z)$: (Eqn. 10.44)

b_0 = 1.08718759187966
b_1 = -1.28141857040708
b_2 = 0.00000000000000
a_1 = -0.20811850838971
a_2 = 0.53778141515093

$H_4(z)$: (Eqn. 10.45)

b_0 = -1.23732501568752
b_1 = -2.48414125820985
b_2 = 0.00000000000000
a_1 = 0.06598033080641
a_2 = 0.35586414751661

$H_5(z)$: (Eqn. 10.46)

b_0 = -7.36231407277814
b_1 = -2.95116120788696
b_2 = 0.00000000000000
a_1 = 0.34033872061873
a_2 = 0.19173505082426

$H_6(z)$: (Eqn. 10.47)

b_0 =-15.00052648289581
b_1 = -4.03777135245981
b_2 = 0.00000000000000
a_1 = 0.52087632363417
a_2 = 0.08873015419028

C = 21.77435091723825 (Eqn. 10.48)

The quantized transfer function sections are as follows:

$H_{1qt}(z)$: (Eqn. 10.49)

b_0 = 0.08822631835937500000
b_1 = 0.08908081054687500000
b_2 = 0.00000000000000000000
a_1 = -0.57510375976562500000
a_2 = 0.90115356445312500000

$H_{2qt}(z)$: (Eqn. 10.50)

b_0 = 0.69625854492187500000
b_1 = -0.16107177734375000000
b_2 = 0.00000000000000000000
a_1 = -0.42849731445312500000
a_2 = 0.71817016601562500000

$H_{3qt}(z)$: (Eqn. 10.51)

b_0 = 1.08715820312500000000
b_1 = -1.28140258789062500000
b_2 = 0.00000000000000000000
a_1 = -0.20809936523437500000
a_2 = 0.53778076171875000000

$H_{4qt}(z)$: (Eqn. 10.52)

b_0 = -1.23730468750000000000
b_1 = -2.48413085937500000000
b_2 = 0.00000000000000000000
a_1 = 0.06579900390625000000
a_2 = 0.35583496093750000000

$H_{5qt}(z)$: (Eqn. 10.53)

b_0 = -7.36230468750000000000
b_1 = -2.95114135742187500000
b_2 = 0.00000000000000000000
a_1 = 0.34033203125000000000
a_2 = 0.19171142578125000000

$H_{6qt}(z)$: (Eqn. 10.54)

b_0 =-15.00051879882812500000
b_1 = -4.03775024414062500000
b_2 = 0.00000000000000000000
a_1 = 0.52087402343750000000
a_2 = 0.08871459960937500000

C_{qt} = 21.77429199218750000000 (Eqn. 10.55)

$H_{1qr}(z)$: (Eqn. 10.56)

b_0 = 0.08825683593750000000
b_1 = 0.08911132812500000000
b_2 = 0.00000000000000000000
a_1 = -0.57513427734375000000
a_2 = 0.90115356445312500000

$H_{2qr}(z)$: (Eqn. 10.57)

b_0 = 0.69625854492187500000
b_1 = -0.16110229492187500000
b_2 = 0.00000000000000000000
a_1 = -0.42849731445312500000
a_2 = 0.71817016601562500000

$H_{3qr}(z)$: (Eqn. 10.58)

b_0 = 1.08718872070312500000
b_1 = -1.28143310546875000000
b_2 = 0.00000000000000000000
a_1 = -0.20812988281250000000
a_2 = 0.53778076171875000000

$H_{4qr}(z)$: (Eqn. 10.59)

b_0 = -1.23733520507812500000
b_1 = -2.48413085937500000000
b_2 = 0.00000000000000000000
a_1 = 0.06597900390625000000
a_2 = 0.35586547851562500000

$H_{5qr}(z)$: (Eqn. 10.60)

b_0 = -7.36230468750000000000
b_1 = -2.95117187500000000000
b_2 = 0.00000000000000000000
a_1 = 0.34033203125000000000
a_2 = 0.19174194335937500000

$H_{6qr}(z)$: (Eqn. 10.61)

b_0 =-15.00051879882812500000
b_1 = -4.03778076171875000000
b_2 = 0.00000000000000000000
a_1 = 0.52087402343750000000
a_2 = 0.08874511718750000000

C_{qr} = 21.77435302734375000000 (Eqn. 10.62)

	Quantization Type	
	Truncated	**Rounded**
$H_{1e}(z)$	0.00059309268750	0.00011597182782
$H_{2e}(z)$	0.00007898914417	0.00008408559463
$H_{3e}(z)$	0.00010478704520	0.00004926607068
$H_{4e}(z)$	0.00022032804086	0.00001795303147
$H_{5e}(z)$	0.00028290654429	0.00010620185850
$H_{6e}(z)$	0.00047426579551	0.00061698655317
C_q	0.00005892505075	-0.00000211010550

Table 10-12. Magnitude error calculation.

Based on the results of Table 10-12, the rounding quantization method produces the best results in terms of smallest error.

COSSAP RoundMode	P_e	Dead Band Range (Fixed Value Output)	COSSAP/VHDL Peak PSD
0	$1.488649920033 \times 10^{-8}$	$-1.220703125 \times 10^{-4}$	-43.904695 dB at 0.000000 Hz
1	2.051274×10^{-14}	$-3.0517578125 \times 10^{-4}$	-41.814991 dB at 1999511.718750 Hz
2	2.051274×10^{-14}	$-3.0517578125 \times 10^{-4}$	-41.814991 dB at 1999511.718750 Hz
3	5.62515×10^{-15}	0.00	-97.203554 dB at 1999511.718750 Hz

Table 10-13. Round-off noise power results of impulse response of COSSAP multiplier round-off modes.

Based on Table 10-13, *RoundMode = 3* produces the best results in terms of smallest error. Figures 10.27 through 10.30 show the power spectral density of the impulse response of the COSSAP round-off modes. Using MATLAB as the ideal model, the peak PSD of the impulse response is -95.847452 dB at 2004394.531250 Hz. The number of bits assigned for fixed-point integer representation in the adder sub-blocks is 6 based on Steps 1 through 5 of section 8.2.

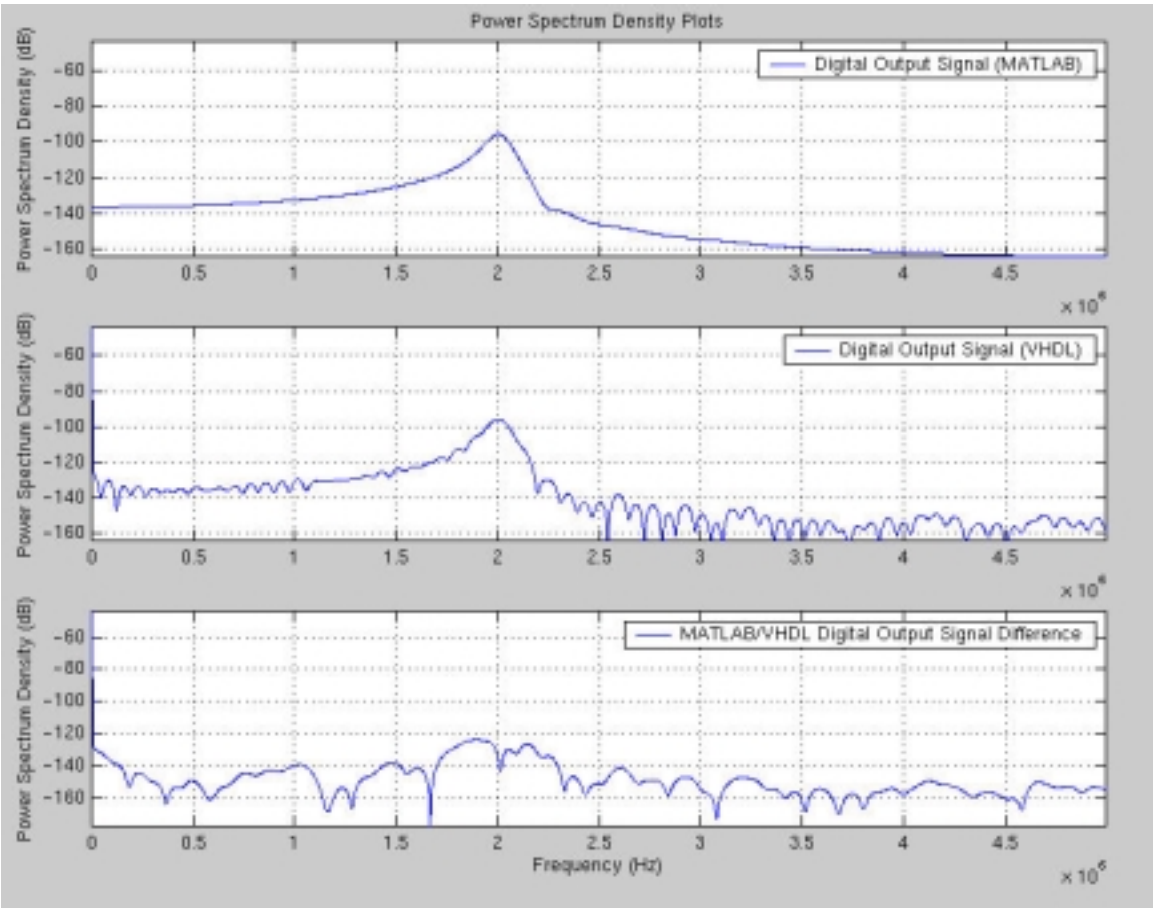


Fig. 10.27. Power spectral density results of impulse response of COSSAP *RoundMode = 0*.

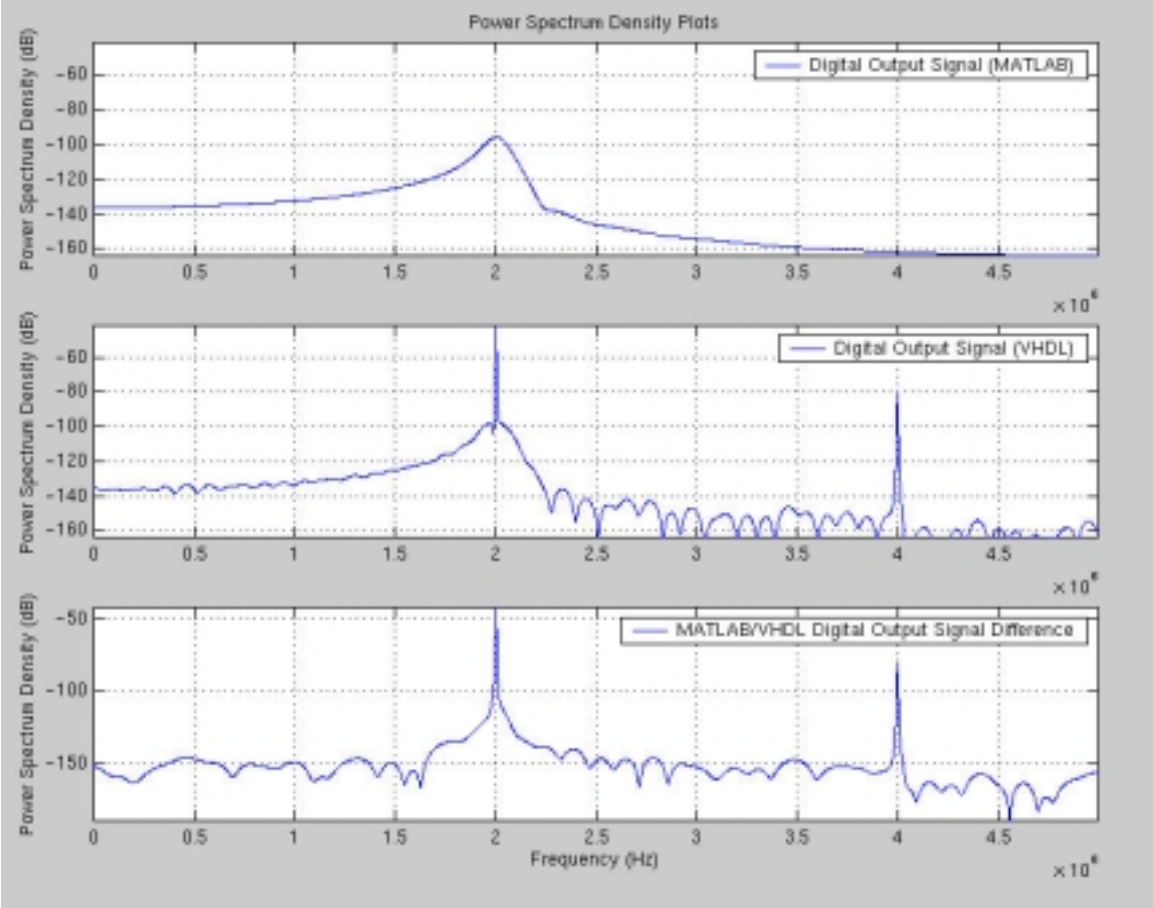


Fig. 10.28. Power spectral density results of impulse response of COSSAP *RoundMode = 1*.

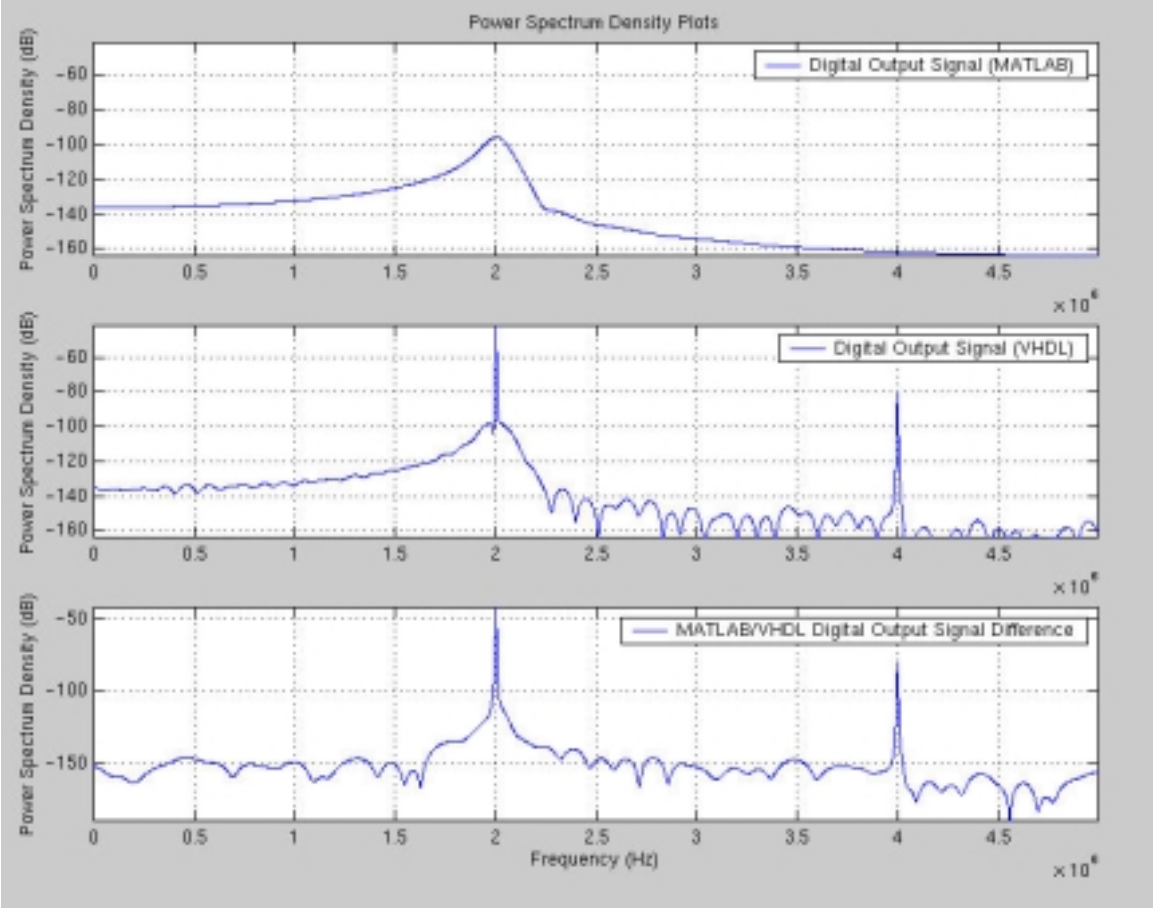


Fig. 10.29. Power spectral density results of impulse response of COSSAP *RoundMode = 2*.

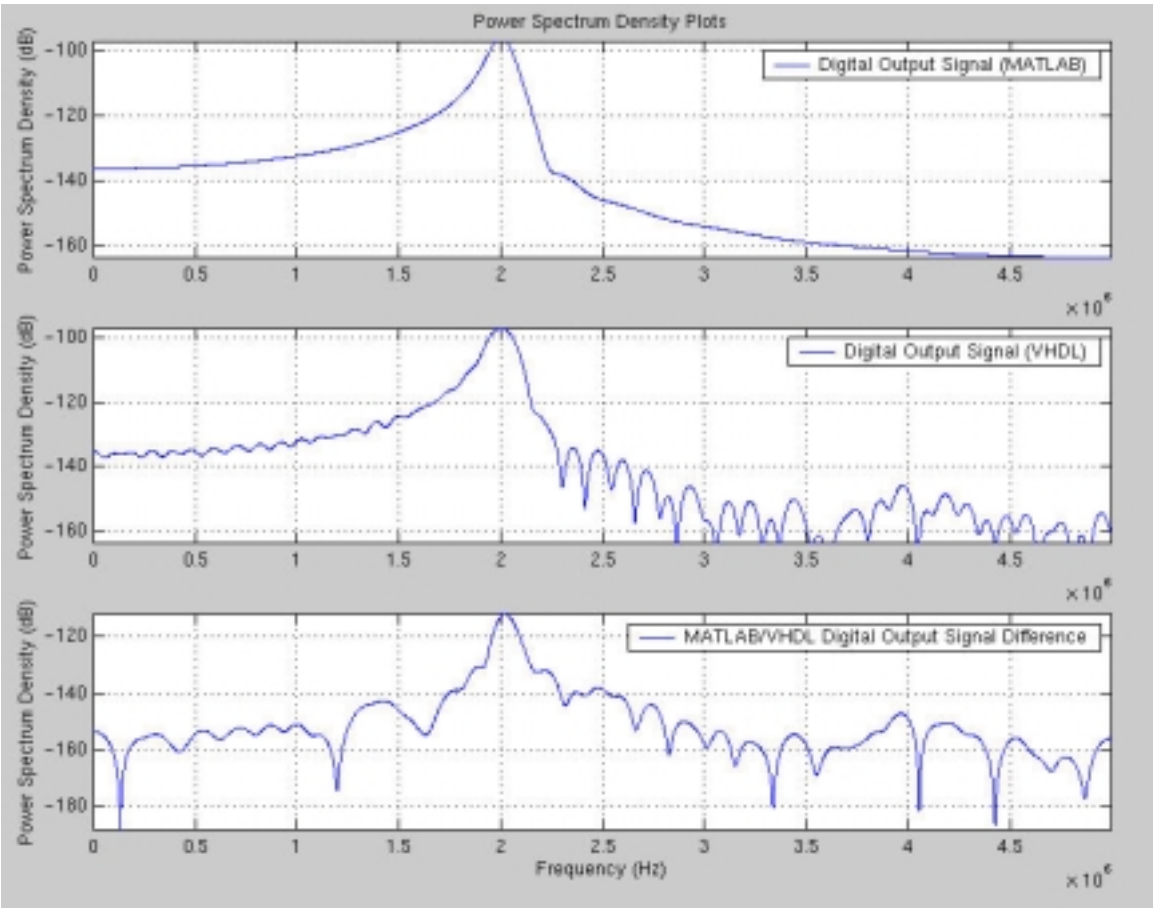


Fig. 10.30. Power spectral density results of impulse response of COSSAP *RoundMode = 3*.

10.2.1.1: Results of Validation Test #1

The passband digital input signal used is at 500kHz. The transition band input signal used is at 2.1MHz. The stopband digital input signal used is at 3MHz. Table 10-14 shows the tabulated results of these 3 signals.

	Round-off Noise Power (P_e)		
COSSAP RoundMode	**Passband Signal**	**Transition Band Signal**	**Stopband Signal**
0	$2.7593046651912 \times 10^{-7}$	$2.9615807101575 \times 10^{-7}$	$2.269853469814 \times 10^{-7}$
1	$4.792357585 \times 10^{-11}$	$1.25242013 \times 10^{-12}$	$3.541640570 \times 10^{-11}$
2	$9.976336315 \times 10^{-11}$	$2.7389096 \times 10^{-13}$	$3.541640570 \times 10^{-11}$
3	$1.6735581939 \times 10^{-10}$	$3.64642492 \times 10^{-12}$	$3.744875743 \times 10^{-10}$

Table 10-14. Tabulated results of validation test #1.

10.2.1.2: Results of Validation Test #2

Figure 10.31 shows the digital input/output response of the unquantized lowpass Chebyshev Type II filter. The low-power passband input signal is at 500kHz. The high-power stopband input signal is at 2.1MHz. For the purposes of analysis, the high-power signal is 10 times greater in amplitude than the low-power signal. As can be seen from the figure, excluding the initial transient response at the output, the ideal Chebyshev Type II filter effectively removes the high-power stopband signal. Comparing the quantized filter, Figures 10.32 through 10.35 show the power spectral density plots of the four COSSAP round-off modes. Table 10-15 shows the tabulated results of the power spectral density and round-off noise power based on the COSSAP multiplier sub-blocks. As can be seen from this table, *RoundMode = 1* and *RoundMode = 2* produce the smallest round-off noise power.

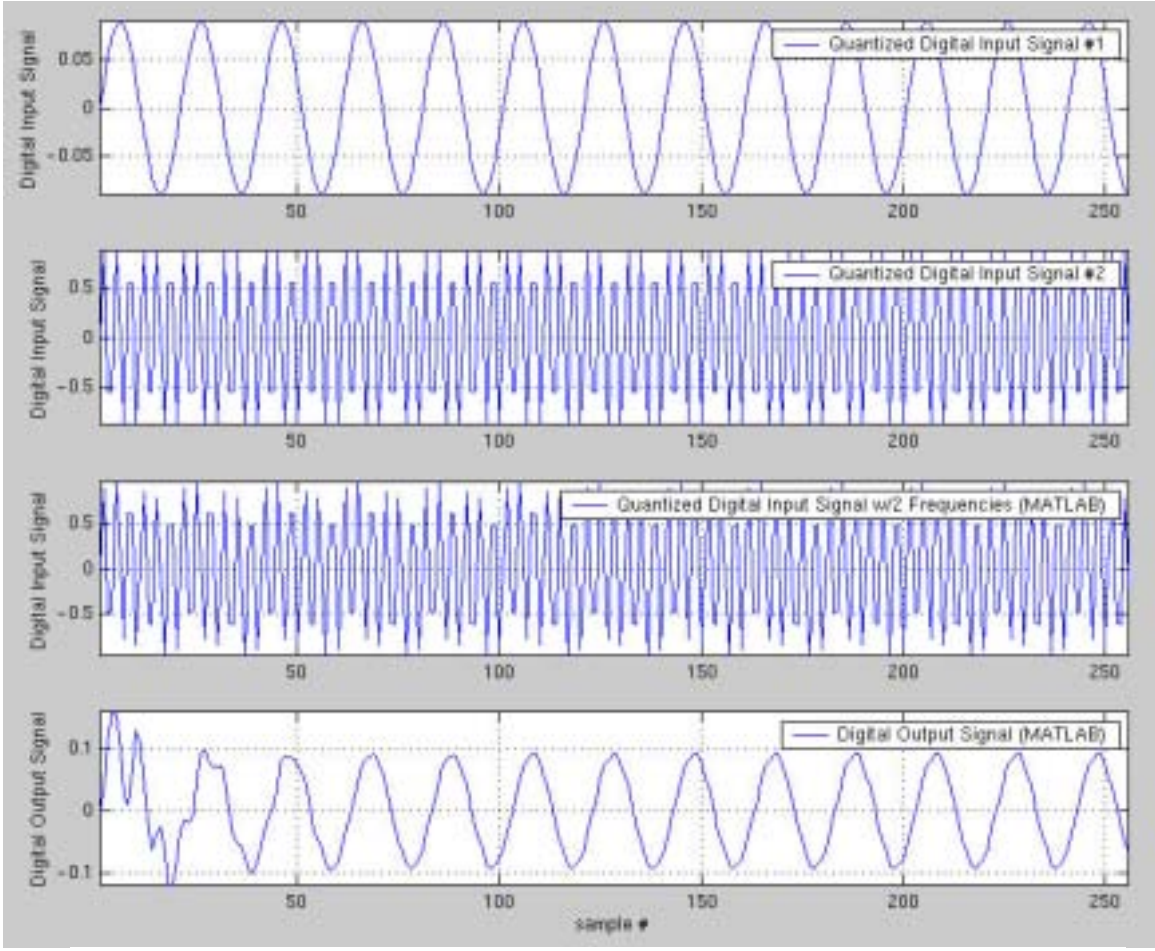


Fig. 10.31. Digital I/O response of validation test #2 for ideal Chebyshev Type II filter.

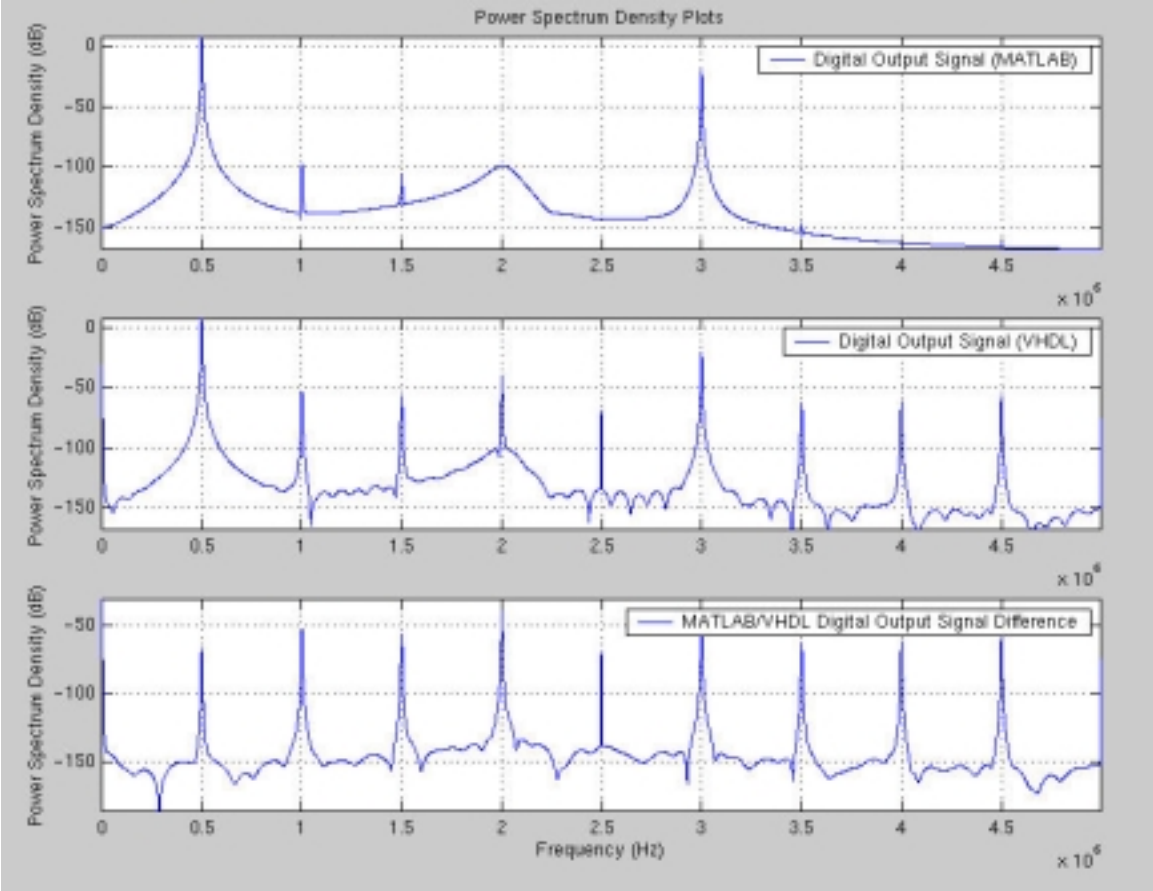


Fig. 10.32. Power spectral density plot of COSSAP *RoundMode = 0*.

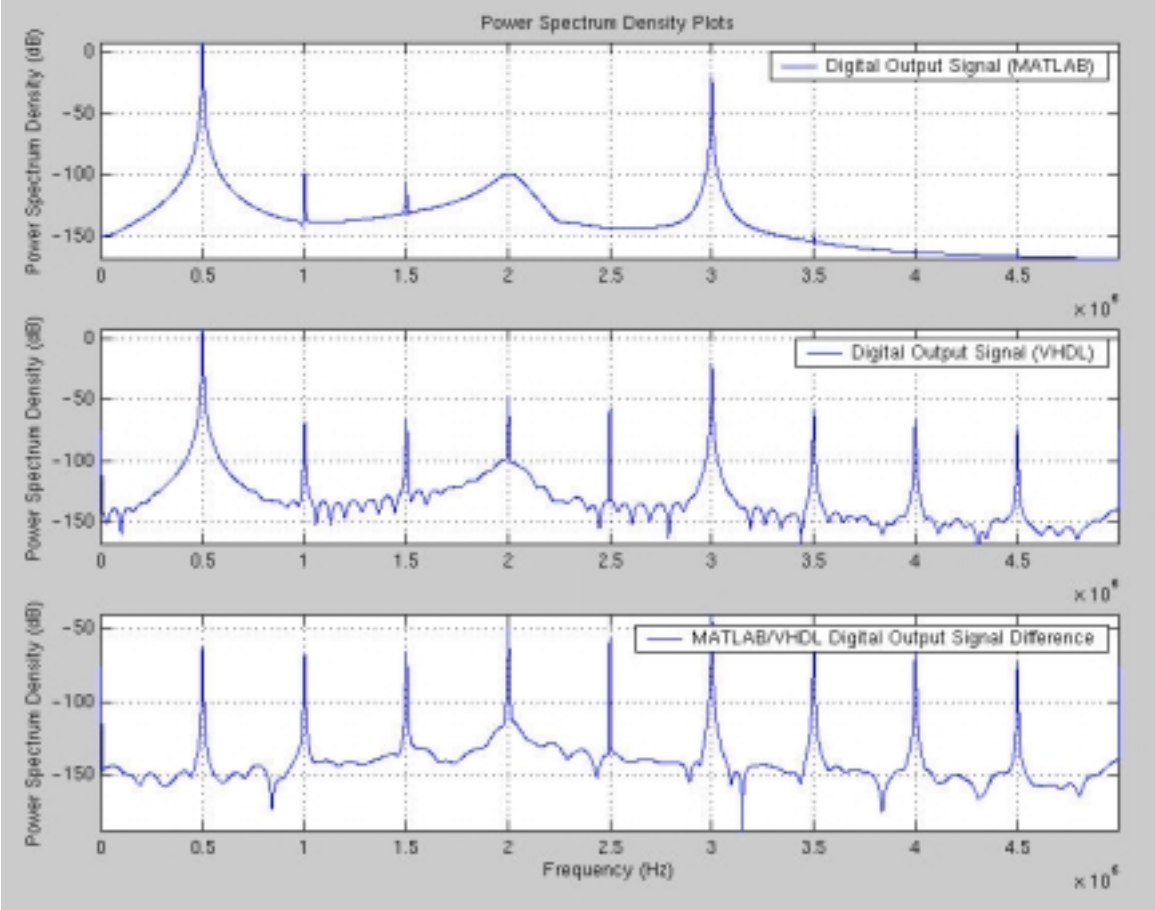


Fig. 10.33. Power spectral density plot of COSSAP *RoundMode* = *1*.

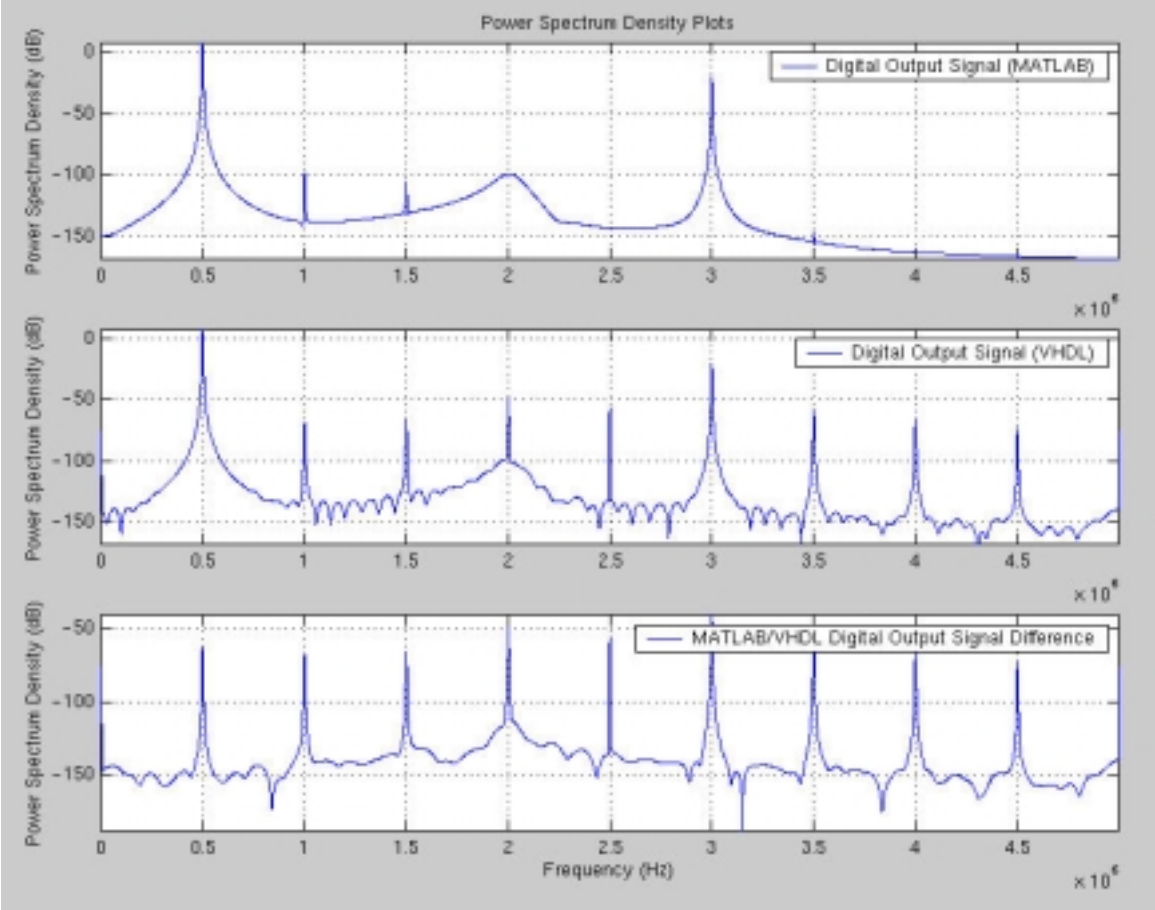


Fig. 10.34. Power spectral density plot of COSSAP *RoundMode* = *2*.

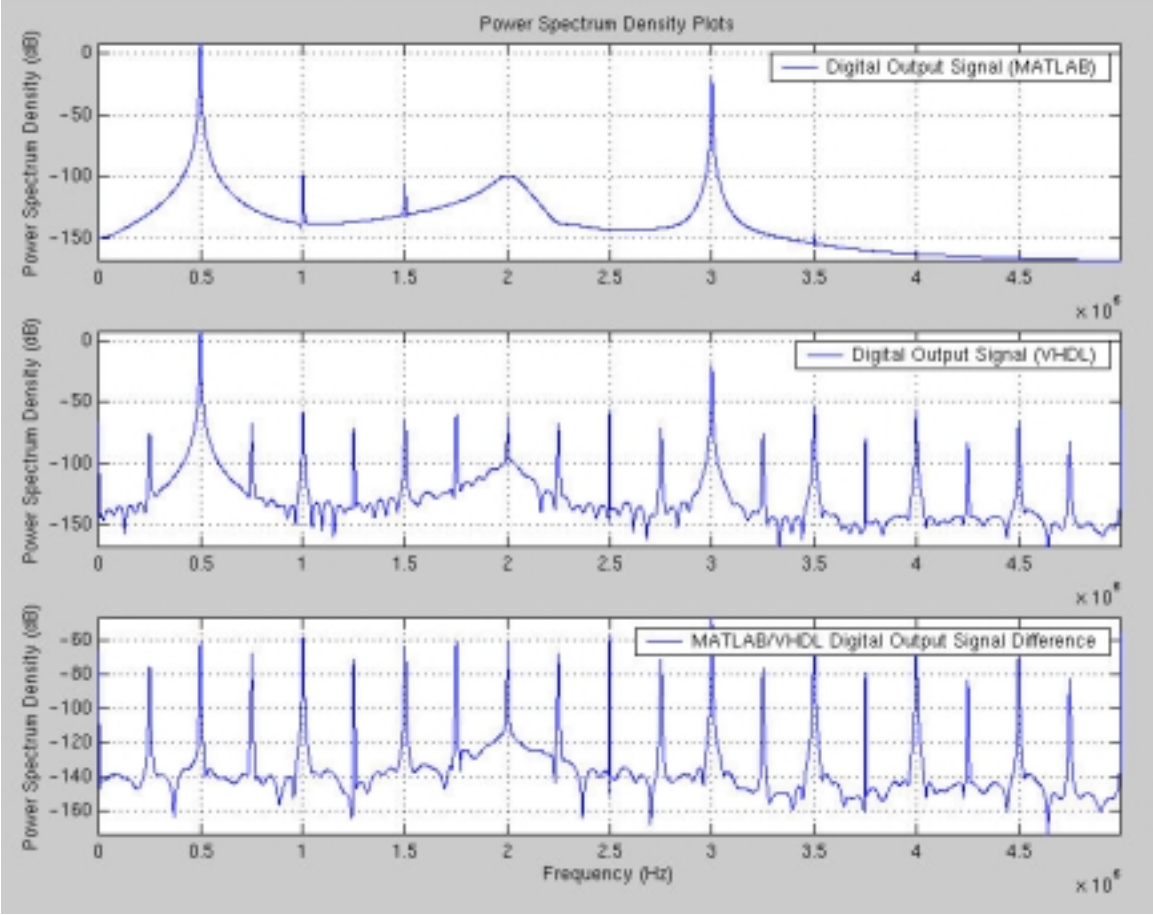


Fig. 10.35. Power spectral density plot of COSSAP *RoundMode* = *3*.

COSSAP RoundMode	P_e
0	$2.6607173257207 \times 10^{-7}$
1	$9.90852667 \times 10^{-12}$
2	$9.90852667 \times 10^{-12}$
3	$8.294560197 \times 10^{-11}$

Table 10-15. Tabulated results of 4 COSSAP round-off modes.

10.2.2: Cascade Structure of Chebyshev Type II Lowpass Filter Results

The unquantized transfer function sections are as follows:

$H_1(z)$: (Eqn. 10.63)

$b_0 = 0.54299498930682$
$b_1 = -0.23133891024656$
$b_2 = 0.54299498930680$
$a_1 = -0.57512824697051$
$a_2 = 0.90116474888622$

$H_2(z)$: (Eqn. 10.64)

$b_0 = 1.12793419825532$
$b_1 = -0.32648956929317$
$b_2 = 1.12793419825543$
$a_1 = -0.42850343365581$
$a_2 = 0.71817606931620$

$H_3(z)$: (Eqn. 10.65)

$b_0 = 0.91891728465592$
$b_1 = 0.01205708702461$
$b_2 = 0.91891728465583$
$a_1 = -0.20811850838971$
$a_2 = 0.53778141515093$

$H_4(z)$: (Eqn. 10.66)

$b_0 = 0.73208063339366$
$b_1 = 0.38789978416300$
$b_2 = 0.73208063339369$
$a_1 = 0.06598033080641$
$a_2 = 0.35586414751661$

$H_5(z)$: (Eqn. 10.67)

$b_0 = 0.57606567135222$
$b_1 = 0.72180140098196$
$b_2 = 0.57606567135222$
$a_1 = 0.34033872061873$
$a_2 = 0.19173505082426$

$H_6(z)$: (Eqn. 10.68)

$b_0 = 0.19331083487109$
$b_1 = 0.36650159251432$
$b_2 = 0.19331083487109$
$a_1 = 0.52087632363417$
$a_2 = 0.08873015419028$

The quantized transfer function sections are as follows:

$H_{1qt}(z)$: (Eqn. 10.69)

$b_0 = 0.54299163818359375000$
$b_1 = -0.23133850097656250000$
$b_2 = 0.54299163818359375000$
$a_1 = -0.57512664794921875000$
$a_2 = 0.90116119384765625000$

$H_{2qt}(z)$: (Eqn. 10.70)

$b_0 = 1.12793350219726562500$
$b_1 = -0.32648849487304687500$
$b_2 = 1.12793350219726562500$
$a_1 = -0.42850112915039062500$
$a_2 = 0.71817398071289062500$

$H_{3qt}(z)$: (Eqn. 10.71)

$b_0 = 0.91891479492187500000$
$b_1 = 0.01205444335937500000$
$b_2 = 0.91891479492187500000$
$a_1 = -0.20811843872070312500$
$a_2 = 0.53778076171875000000$

$H_{4qt}(z)$: (Eqn. 10.72)

$b_0 = 0.73207855224609375000$
$b_1 = 0.38789749145507812500$
$b_2 = 0.73207855224609375000$
$a_1 = 0.06597900390625000000$
$a_2 = 0.35586166381835937500$

$H_{5qt}(z)$: (Eqn. 10.73)

$b_0 = 0.57606506347656250000$
$b_1 = 0.72179794311523437500$
$b_2 = 0.57606506347656250000$
$a_1 = 0.34033584594726562500$
$a_2 = 0.19173431396484375000$

$H_{6qt}(z)$: (Eqn. 10.74)

$b_0 = 0.19330978393554687500$
$b_1 = 0.36650085449218750000$
$b_2 = 0.19330978393554687500$
$a_1 = 0.52087402343750000000$
$a_2 = 0.08872985839843750000$

$H_{1qr}(z)$: (Eqn. 10.75)

$b_0 = 0.54299545288085937500$
$b_1 = -0.23133850097656250000$
$b_2 = 0.54299545288085937500$
$a_1 = -0.57512664794921875000$
$a_2 = 0.90116500854492187500$

$H_{2qr}(z)$: (Eqn. 10.76)

$b_0 = 1.12793350219726562500$
$b_1 = -0.32648849487304687500$
$b_2 = 1.12793350219726562500$
$a_1 = -0.42850494384765625000$
$a_2 = 0.71817779541015625000$

$H_{3qr}(z)$: (Eqn. 10.77)

$b_0 = 0.91891860961914062500$
$b_1 = 0.01205825805664062500$
$b_2 = 0.91891860961914062500$
$a_1 = -0.20811843872070312500$
$a_2 = 0.53778076171875000000$

$H_{4qr}(z)$: (Eqn. 10.78)

$b_0 = 0.73208236694335937500$
$b_1 = 0.38790130615234375000$
$b_2 = 0.73208236694335937500$
$a_1 = 0.06597900390625000000$
$a_2 = 0.35586547851562500000$

$H_{5qr}(z)$: (Eqn. 10.79)

$b_0 = 0.57606506347656250000$
$b_1 = 0.72180175781250000000$
$b_2 = 0.57606506347656250000$
$a_1 = 0.34033966064453125000$
$a_2 = 0.19173431396484375000$

$H_{6qr}(z)$: (Eqn. 10.80)

$b_0 = 0.19330978393554687500$
$b_1 = 0.36650085449218750000$
$b_2 = 0.19330978393554687500$
$a_1 = 0.52087783813476562500$
$a_2 = 0.08872985839843750000$

	Quantization Type	
	Truncated	**Rounded**
$H_{1e}(z)$	0.00005773164965	0.00001283204844
$H_{2e}(z)$	0.00000975928717	0.00001081330912
$H_{3e}(z)$	0.00000923307641	0.00000355582764
$H_{4e}(z)$	0.00000627982856	0.00000546820036
$H_{5e}(z)$	0.00000450898430	0.00000151385948
$H_{6e}(z)$	0.00000252681606	0.00000228881216

Table 10-16. Magnitude error calculation.

Based on Table 10-16, the rounding method produces the best results in terms of smallest error.

	COSSAP Coefficients	
$H_{1qr}(z)$	b0_COSSAP = 0	142343
	b1_COSSAP = 0	-60644
	b2_COSSAP = 0	142343
	a1_COSSAP = 0	150766
	a2_COSSAP = 0	-236235
$H_{2qr}(z)$	b0_COSSAP = 1	33537
	b1_COSSAP = 0	-85587
	b2_COSSAP = 1	33537
	a1_COSSAP = 0	112330
	a2_COSSAP = 0	-188266
$H_{3qr}(z)$	b0_COSSAP = 0	240889
	b1_COSSAP = 0	3161
	b2_COSSAP = 0	240889
	a1_COSSAP = 0	54557
	a2_COSSAP = 0	-140976
$H_{4qr}(z)$	b0_COSSAP = 0	191911
	b1_COSSAP = 0	101686
	b2_COSSAP = 0	191911
	a1_COSSAP = 0	-17296
	a2_COSSAP = 0	-93288
$H_{5qr}(z)$	b0_COSSAP = 0	151012
	b1_COSSAP = 0	189216
	b2_COSSAP = 0	151012
	a1_COSSAP = 0	-89218
	a2_COSSAP = 0	-50262
$H_{6qr}(z)$	b0_COSSAP = 0	50675
	b1_COSSAP = 0	96076
	b2_COSSAP = 0	50675
	a1_COSSAP = 0	-136545
	a2_COSSAP = 0	-23260

Table 10-17. COSSAP results of multiplier coefficient transformation.

It is noted here that based on Step 9.3.1 of section 8.2, the number of integer bits configured to the adder sub-blocks of each cascade stage was 2, 2, 2, 2, 2, and 1. But when inputting passband, transition band, and stopband signals into the COSSAP model, the respective output responses were not as expected when compared to MATLAB results. Using Step 10.2 (section 8.2) remedied this predicament. The number of bits now used for integer bit representation is 2, 2, 2, 2, 3, and 2.

COSSAP RoundMode	P_e	Dead Band Range (Fixed Value Output)	COSSAP/VHDL Peak PSD
0	$2.8665804127 \times 10^{-10}$	$-1.9073486328125 \times 10^{-5}$ to $-1.52587890625 \times 10^{-5}$	-61.435597 dB at 0.000000 Hz
1	$2.12300686 \times 10^{-12}$	$-1.52587890625 \times 10^{-5}$ to $1.1444091796875 \times 10^{-5}$	-69.016742 dB at 1999511.718750 Hz
2	$2.12300686 \times 10^{-12}$	$-1.52587890625 \times 10^{-5}$ to $1.1444091796875 \times 10^{-5}$	-69.016742 dB at 1999511.718750 Hz
3	1.756×10^{-17}	0.00	-96.119385 dB at 2001953.125000 Hz

Table 10-18. Round-off noise power results of impulse response of COSSAP multiplier round-off modes.

Based on Table 10-18, *RoundMode = 3* produces the best results in terms of smallest error. Figures 10.36 through 10.39 show the power spectral density of the impulse response of the COSSAP round-off modes. Using MATLAB as the ideal model, the peak PSD of the impulse response is -95.847452 dB at 2004394.531250 Hz.

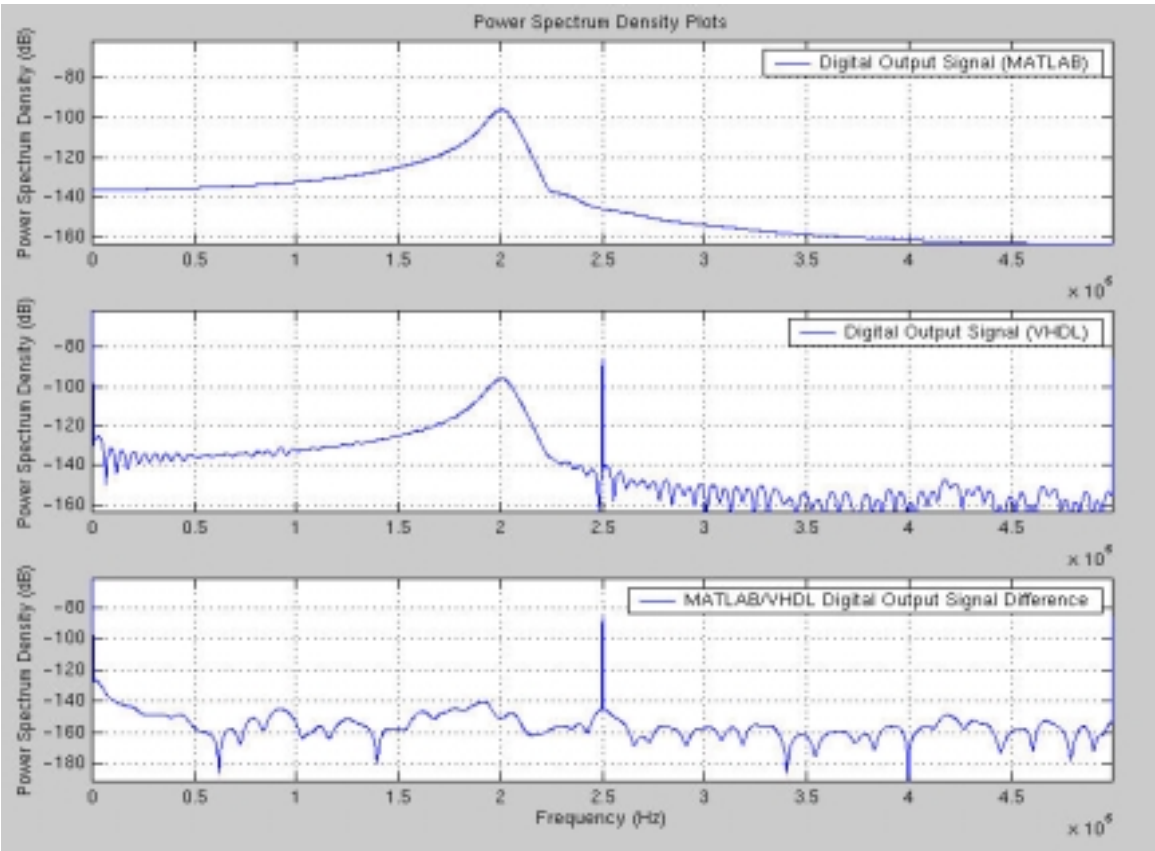


Fig. 10.36. Power spectral density results of impulse response of COSSAP *RoundMode = 0*.

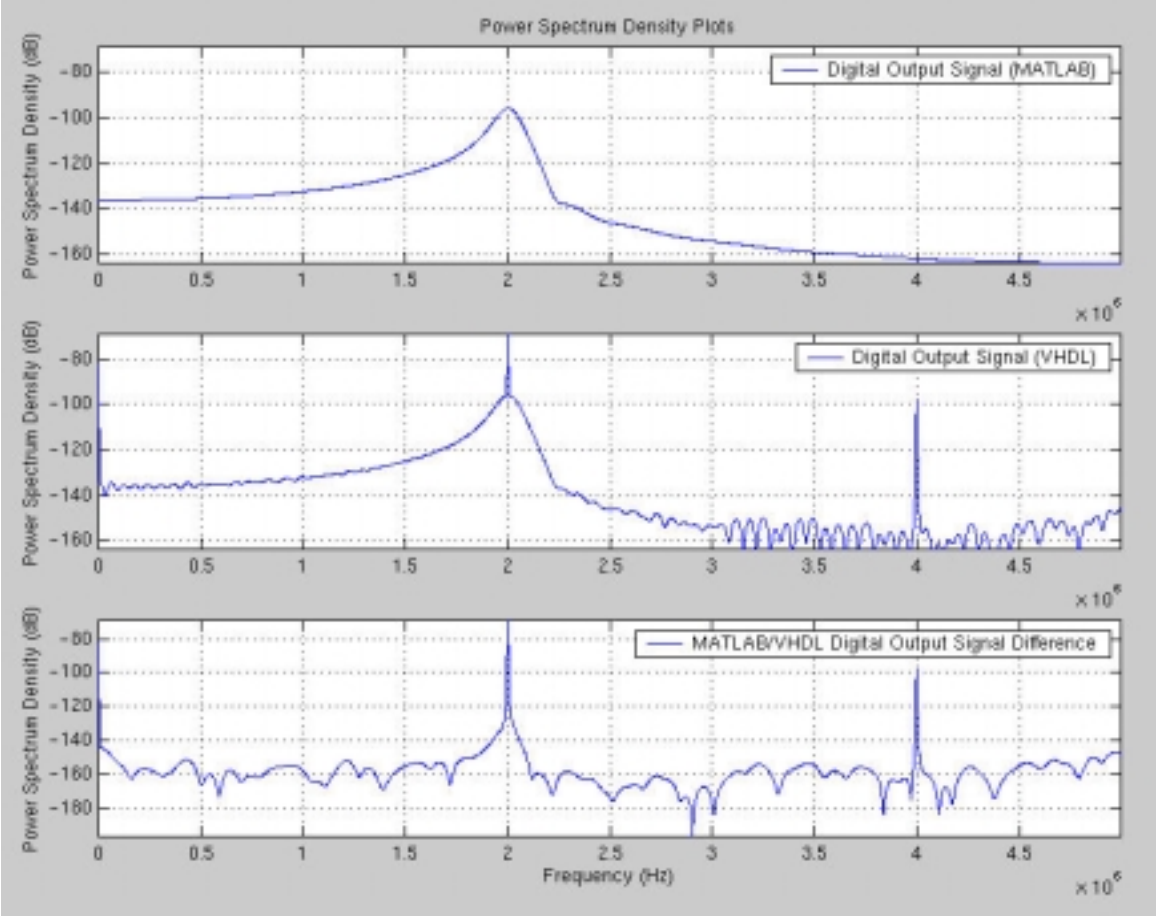


Fig. 10.37. Power spectral density results of impulse response of COSSAP *RoundMode = 1*.

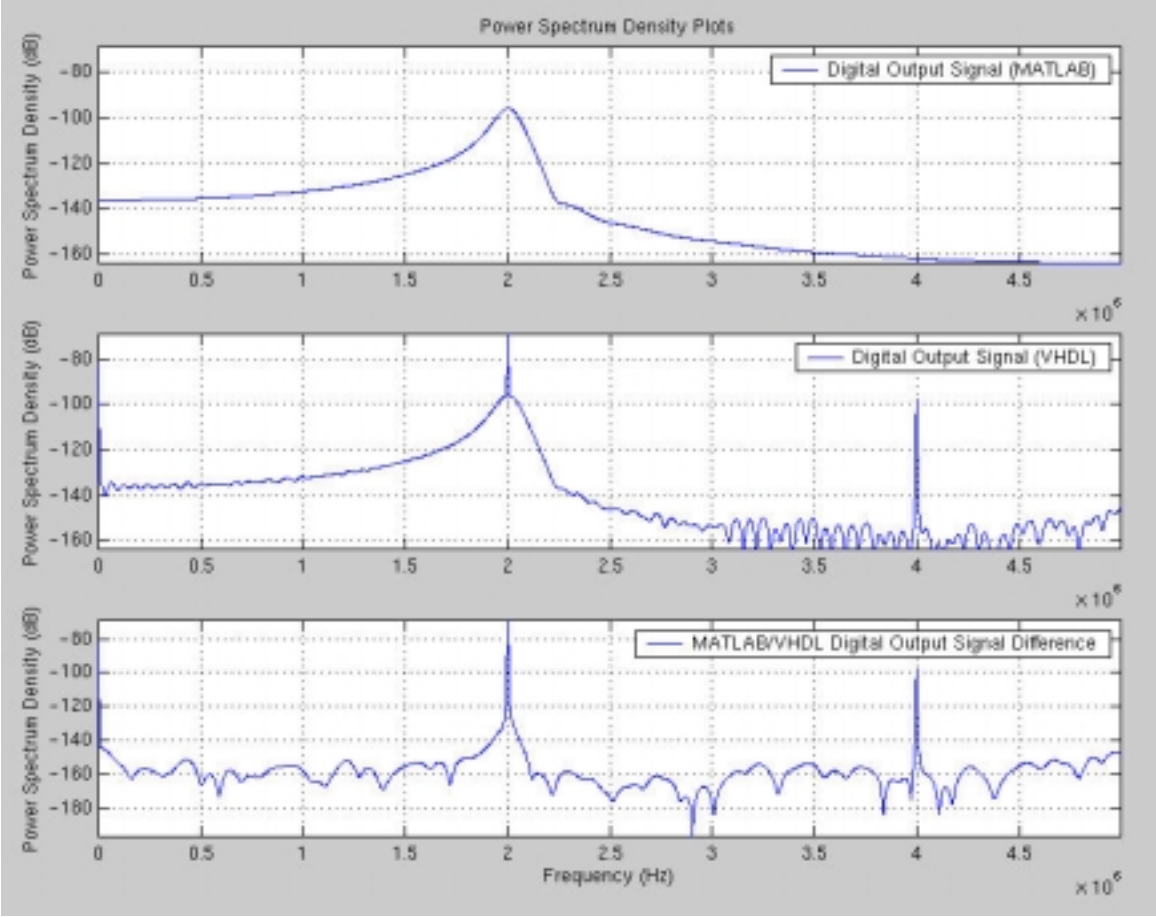


Fig. 10.38. Power spectral density results of impulse response of COSSAP *RoundMode = 2*.

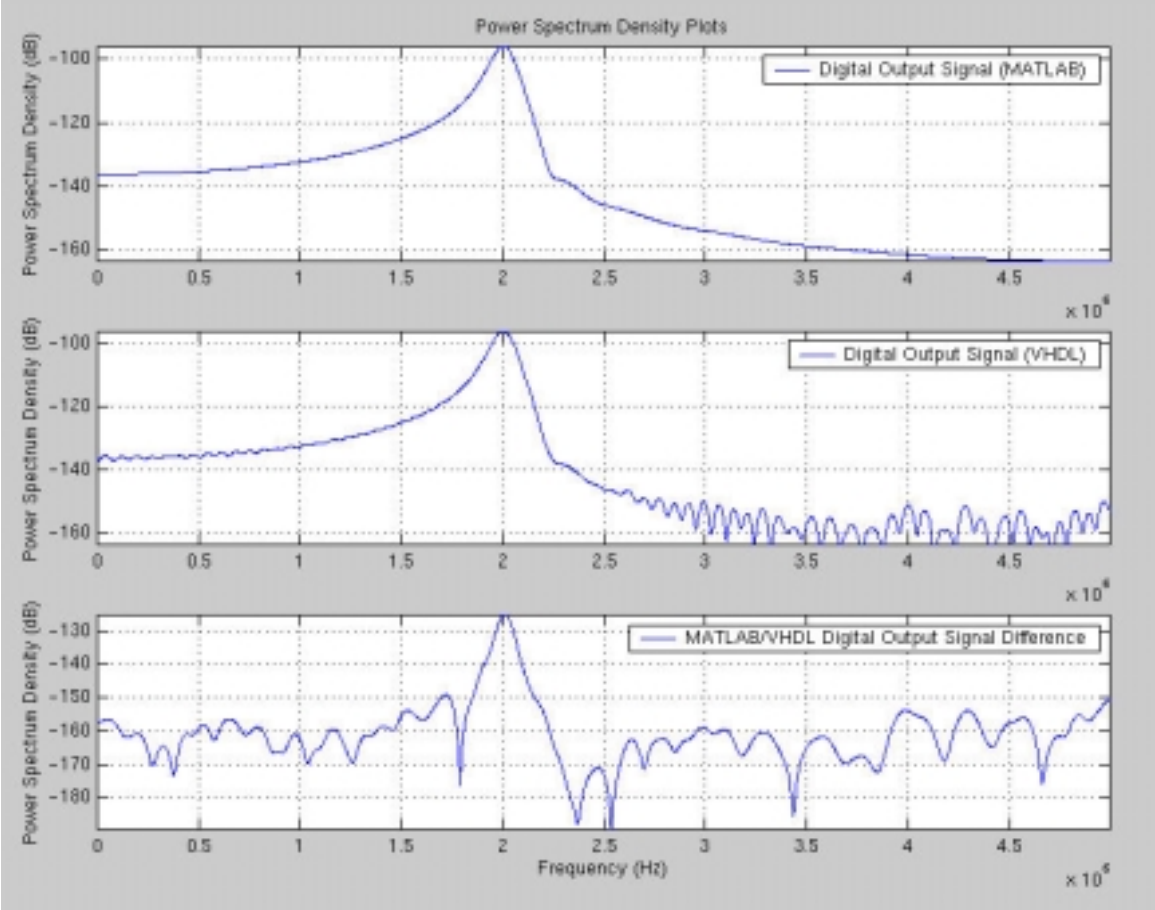


Fig. 10.39. Power spectral density results of impulse response of COSSAP *RoundMode = 3*.

10.2.2.1: Results of Validation Test #1

The digital input signals used in section 10.2.1.1 were also used in this section. Table 10-19 shows these results.

	P_e		
COSSAP RoundMode	**Passband Signal**	**Transition Band Signal**	**Stopband Signal**
0	$2.27916303782 \times 10^{-9}$	$1.51829447952 \times 10^{-9}$	$1.48027757148 \times 10^{-9}$
1	$5.482129324 \times 10^{-11}$	1.549787×10^{-14}	1.3453×10^{-16}
2	$5.482129324 \times 10^{-11}$	1.549787×10^{-14}	1.3453×10^{-16}
3	$2.736200975 \times 10^{-11}$	1.94296×10^{-15}	$8.0178357 \times 10^{-13}$

Table 10-19. Tabulated results of validation test #1.

10.2.2.2: Results of Validation Test #2

The digital input signals used in section 10.2.1.2 were also used in this section.

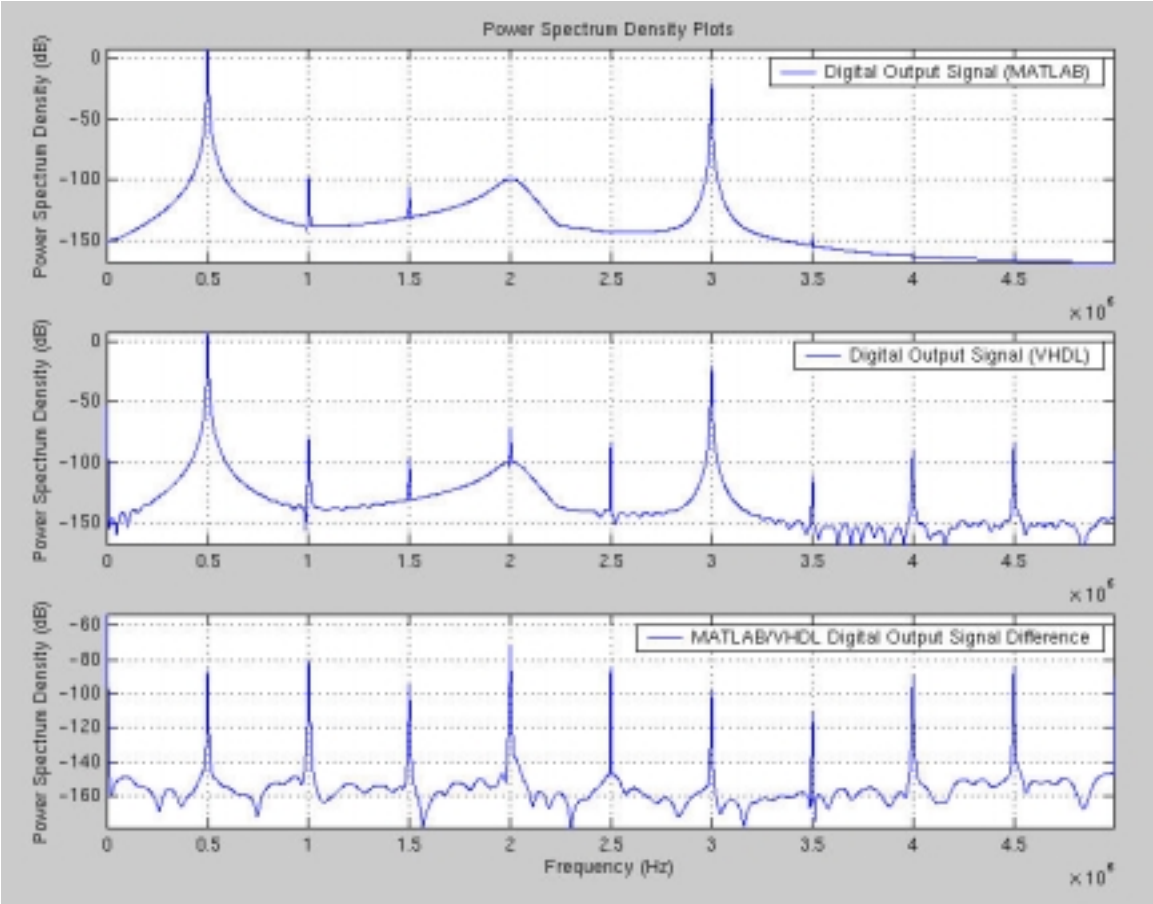


Fig. 10.40. Power spectral density of COSSAP *RoundMode* = *0*.

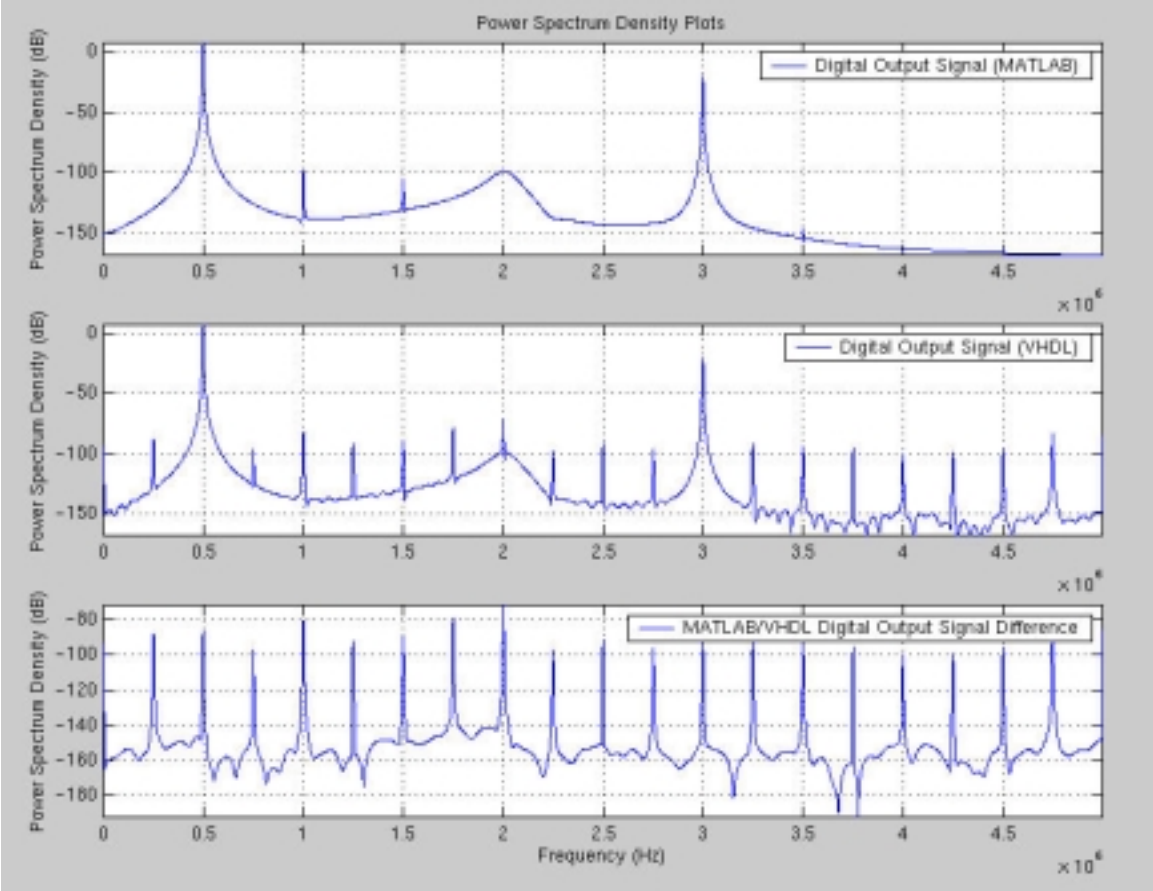


Fig. 10.41. Power spectral density of COSSAP *RoundMode* = *1*.

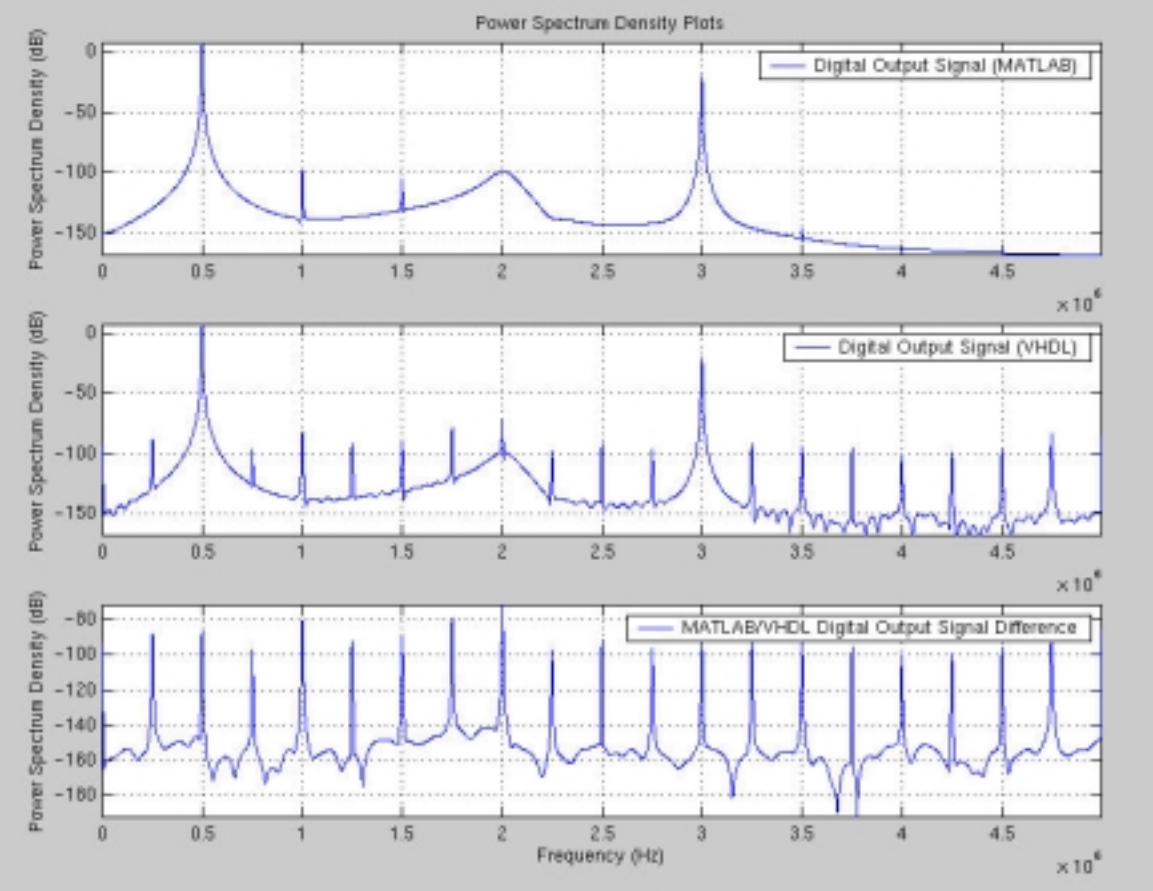


Fig. 10.42. Power spectral density of COSSAP *RoundMode* = *2*.

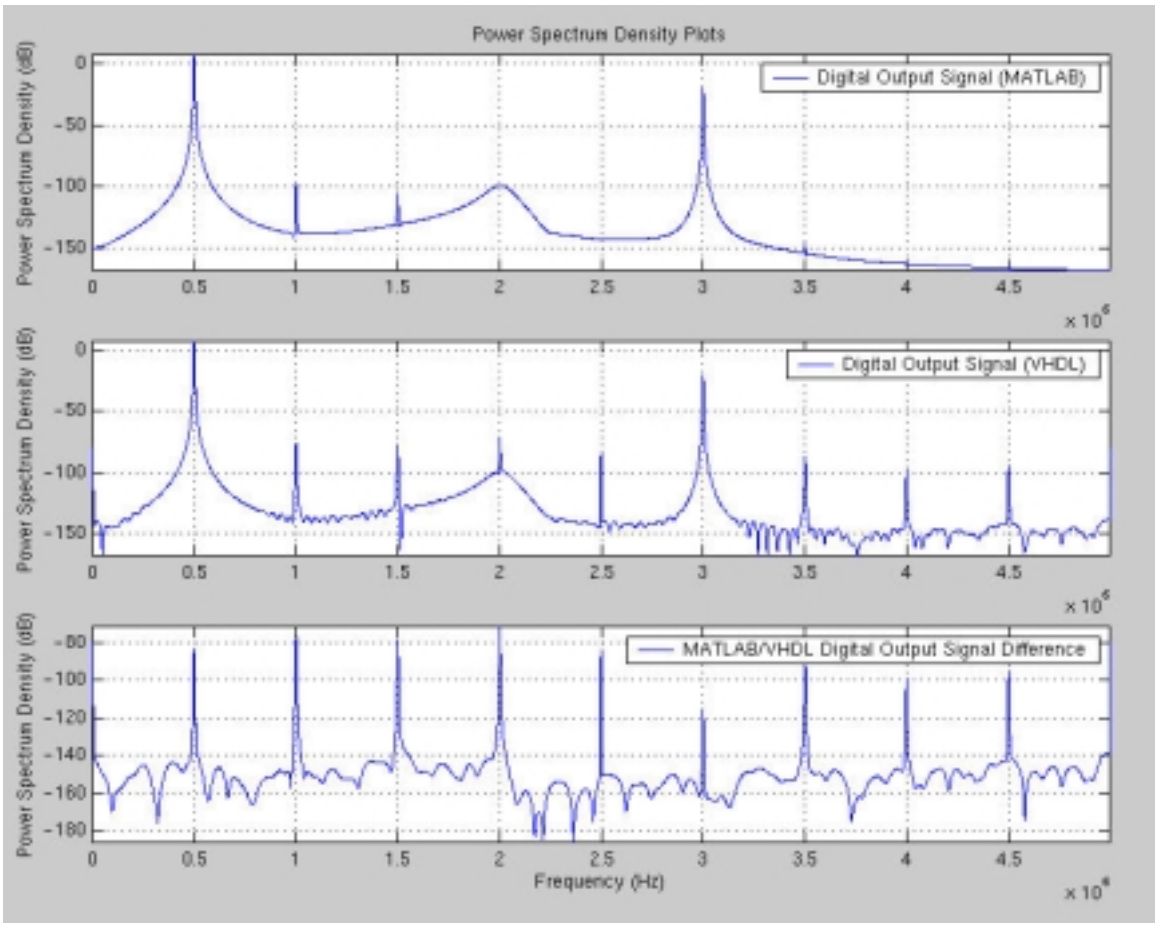


Fig. 10.43. Power spectral density of COSSAP *RoundMode = 3*.

COSSAP RoundMode	P_e
0	$1.52536741464 \times 10^{-9}$
1	$1.4156064 \times 10^{-13}$
2	$1.4156064 \times 10^{-13}$
3	$3.33889286 \times 10^{-12}$

Table 10-20. Tabulated results of 4 COSSAP round-off modes.

Based on the results of Table 10-20, the best COSSAP multiplier sub-block round-off modes are *RoundMode = 1* and *RoundMode = 2*.

10.3: Data Communications and Imaging Bandwidth Results

The sampling frequency used for this design was 20MHz. This design is a 24-bit bandpass digital filter design and the design type is Elliptic. The first and second cut-off frequencies (-3dB points) are 29.304kHz and 5.001221MHz, respectively. The first and second stopband frequencies (-40dB points) are 24.42kHz and 5.445665MHz, respectively. The MATLAB floating-point representations of the coefficients, taken to 14 significant places, are as follows:

$b_0 = 0.07987329621409$
$b_1 = -0.26396911739870$
$b_2 = 0.29915964301465$
$b_3 = -0.26454500340110$
$b_4 = 0.41604640068847$
$b_5 = -0.34604558265014$
$b_6 = 0.15896072706548$
$b_7 = -0.34604558265014$
$b_8 = 0.41604640068847$
$b_9 = -0.26454500340110$
$b_{10} = 0.29915964301465$
$b_{11} = -0.26396911739870$
$b_{12} = 0.07987329621409$
$a_1 = -6.62643192221140$
$a_2 = 20.52814661024491$
$a_3 = -40.84768268409439$
$a_4 = 59.97920058345937$
$a_5 = -68.93513855046317$
$a_6 = 62.97666301624375$
$a_7 = -45.87184600775393$
$a_8 = 26.51594785904635$
$a_9 = -11.79494300412995$
$a_{10} = 3.78168492675603$
$a_{11} = -0.79228595105469$
$a_{12} = 0.08668512396083$

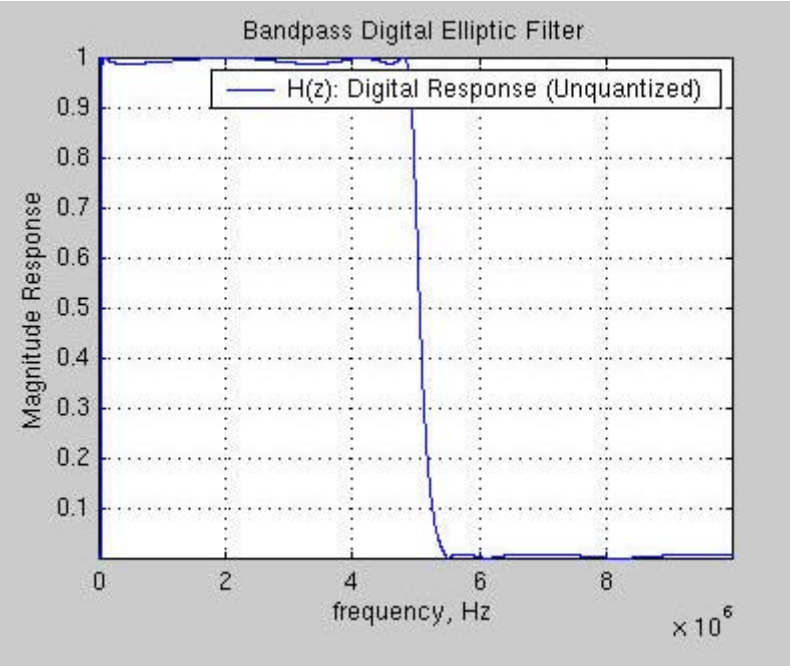


Fig. 10.44. MATLAB magnitude response of bandpass Elliptic filter.

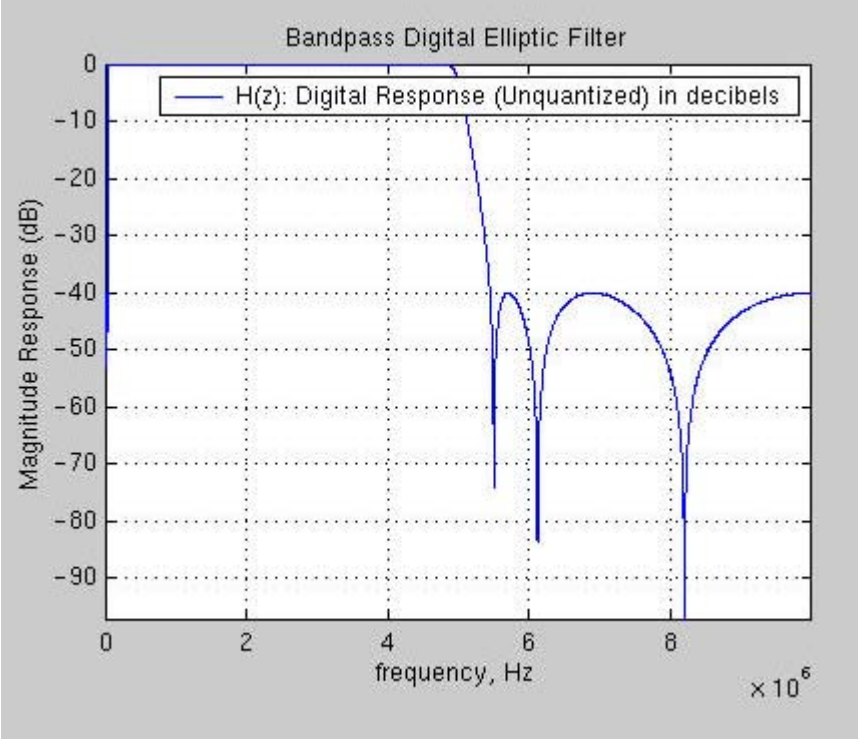


Fig. 10.45. MATLAB magnitude response of bandpass Elliptic filter (dB).

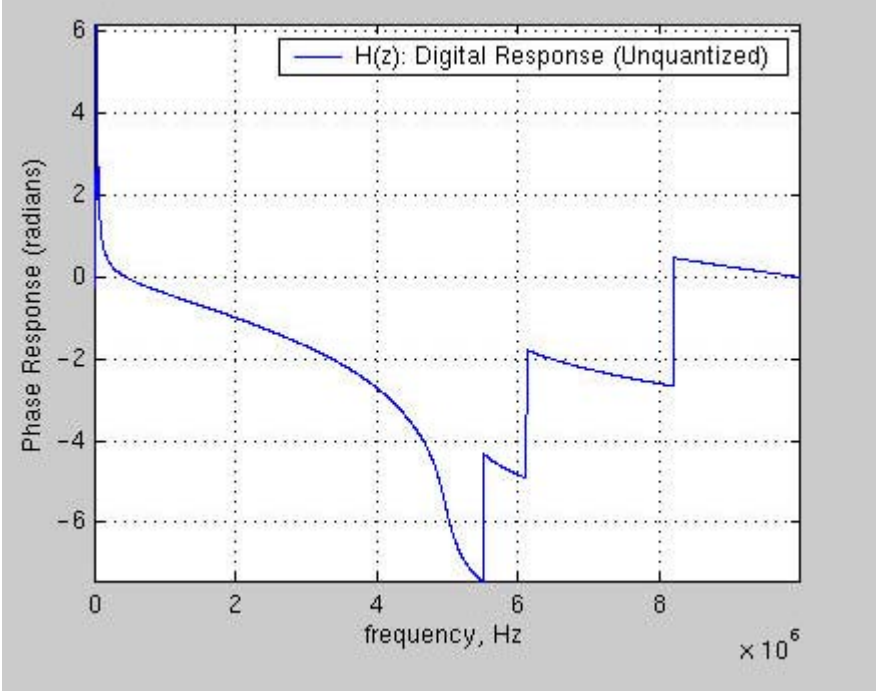


Fig. 10.46. MATLAB phase response of bandpass Elliptic filter (radians).

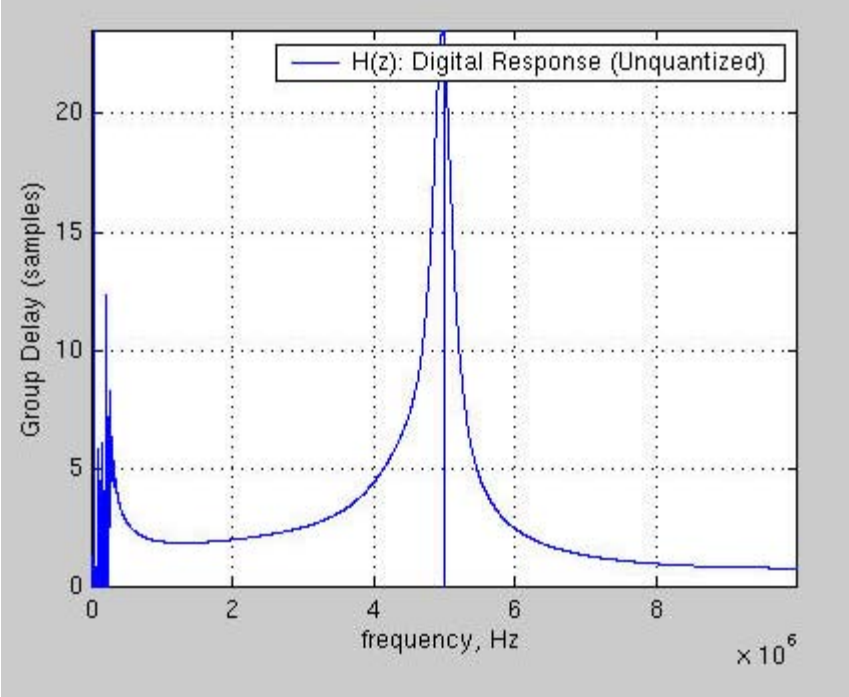


Fig. 10.47. MATLAB group delay response of bandpass Elliptic filter.

10.3.1: Parallel Structure of Elliptic Bandpass Filter Results

The unquantized transfer function sections are as follows:

$H_1(z)$: (Eqn. 10.81)

b_0 = 0.00041754780789
b_1 = -0.00041853200266
b_2 = 0.00000000000000
a_1 = -1.99928810579703
a_2 = 0.99937493222654

$H_2(z)$: (Eqn. 10.82)

b_0 = -0.00153637238075
b_1 = 0.00159484444159
b_2 = 0.00000000000000
a_1 = -1.99442029179693
a_2 = 0.99453510743410

$H_3(z)$: (Eqn. 10.83)

b_0 = -0.02753390048704
b_1 = 0.02722912721656
b_2 = 0.00000000000000
a_1 = -1.97767024097127
a_2 = 0.97787185728449

$H_4(z)$: (Eqn. 10.84)

$b_0 = 0.11620735597411$
$b_1 = -0.00877830810331$
$b_2 = 0.00000000000000$
$a_1 = -0.01831558526104$
$a_2 = 0.89972679483240$

$H_5(z)$: (Eqn. 10.85)

$b_0 = -0.33117998124798$
$b_1 = -0.34143788679057$
$b_2 = 0.00000000000000$
$a_1 = -0.17586277586059$
$a_2 = 0.60203802158022$

$H_6(z)$: (Eqn. 10.86)

$b_0 = -0.59792241472711$
$b_1 = 0.97547798514305$
$b_2 = 0.00000000000000$
$a_1 = -0.46087492252454$
$a_2 = 0.16465672860857$

$C = 0.92141872289623$ (Eqn. 10.87)

The quantized transfer function sections are as follows:

$H_{1qt}(z)$: (Eqn. 10.88)

$b_0 = 0.00041747093200683593750 0$
$b_1 = -0.00041842460632424218750 0$
$b_2 = 0.00000000000000000000000$
$a_1 = -1.99928808212280273437500 0$
$a_2 = 0.99937486485595703125000$

$H_{2qt}(z)$: (Eqn. 10.89)

$b_0 = -0.00153636932373046875000 0$
$b_1 = 0.00159478187561035156250 0$
$b_2 = 0.00000000000000000000000$
$a_1 = -1.99442028999328613281250 0$
$a_2 = 0.99453496932983398437500 0$

$H_{3qt}(z)$: (Eqn. 10.90)

$b_0 = -0.02753376960754394531250 0$
$b_1 = 0.02722907066345214843750 0$
$b_2 = 0.00000000000000000000000$
$a_1 = -1.97767019271850585937500 0$
$a_2 = 0.97787165641784667968750 0$

$H_{4qt}(z)$: (Eqn. 10.91)

$b_0 = 0.11620712280273437500000 0$
$b_1 = -0.00877809524536132812500 0$
$b_2 = 0.00000000000000000000000$
$a_1 = -0.01831555366516113281250 0$
$a_2 = 0.89972662925720214843750 0$

$H_{5qt}(z)$: (Eqn. 10.92)

b_0 = -0.331179857254028320312500
b_1 = -0.341437816619873046875000
b_2 = 0.000000000000000000000000
a_1 = -0.175862550735473632812500
a_2 = 0.602037906646728515625000

$H_{6qt}(z)$: (Eqn. 10.93)

b_0 = -0.597922325134277343750000
b_1 = 0.975477933883666992187500
b_2 = 0.000000000000000000000000
a_1 = -0.460874795913696289062500
a_2 = 0.164656639099121093750000

C_{qt} = 0.921418666839599609375000 (Eqn. 10.94)

$H_{1qr}(z)$: (Eqn. 10.95)

b_0 = 0.000417470932006835937500
b_1 = -0.000418424606323242187500
b_2 = 0.000000000000000000000000
a_1 = -1.999288082122802734375000
a_2 = 0.999374866485595703125000

$H_{2qr}(z)$: (Eqn. 10.96)

b_0 = -0.001536369323730468750000
b_1 = 0.001594781875610351562500
b_2 = 0.000000000000000000000000
a_1 = -1.994420289993286132812500
a_2 = 0.994535207748413085937500

$H_{3qr}(z)$: (Eqn. 10.97)

b_0 = -0.027534008026123046875000
b_1 = 0.027229070663452148437500
b_2 = 0.000000000000000000000000
a_1 = -1.977670192718505859375000
a_2 = 0.977871894836425781250000

$H_{4qr}(z)$: (Eqn. 10.98)

b_0 = 0.116207361221313476562500
b_1 = -0.008778333663940429687500
b_2 = 0.000000000000000000000000
a_1 = -0.018315553665161132812500
a_2 = 0.899726867675781250000000

$H_{5qr}(z)$: (Eqn. 10.99)

b_0 = -0.331180095672607421875000
b_1 = -0.341437816619873046875000
b_2 = 0.000000000000000000000000
a_1 = -0.175862789154052734375000
a_2 = 0.602037906646728515625000

$H_{6qr}(z)$: (Eqn. 10.100)

b_0 = -0.597922325134277343750000
b_1 = 0.975477933883666992187500
b_2 = 0.000000000000000000000000
a_1 = -0.460875034332275390625000
a_2 = 0.164656639099121093750000

C_{qr} = 0.921418666839599609375000 (Eqn. 10.101)

	Quantization Type	
	Truncated	**Rounded**
$H_{1e}(z)$	0.00047278394141	0.00047278394141
$H_{2e}(z)$	0.00144342084271	0.00189052058450
$H_{3e}(z)$	0.00077618133661	0.00032127790090
$H_{4e}(z)$	0.00000437858494	0.00000094000653
$H_{5e}(z)$	0.00000089114714	0.00000028762238
$H_{6e}(z)$	0.00000021411323	0.00000026682198
C_q	0.00000005605663	0.00000005605663

Table 10-21. Magnitude error calculation.

Based on the results of Table 10-21, the rounding quantization method produces the best results in terms of smallest error.

Designing this quantized filter structure proved to be extremely difficult. Attempts were made to implement a working filter through Steps 9.2 and 10.1 of section 8.2. The cascade structure is attempted in the next section to realize this bandpass filter.

10.3.2: Cascade Structure of Elliptic Bandpass Filter Results

The unquantized transfer function sections are as follows:

$H_1(z)$: (Eqn. 10.102)

b_0 = 0.07987329621409
b_1 = -0.15974142191627
b_2 = 0.07987324230420
a_1 = -1.99928810579703
a_2 = 0.99937493222654

$H_2(z)$: (Eqn. 10.103)

b_0 = 1.00000000000000
b_1 = -1.99995845045300
b_2 = 1.00000108572615
a_1 = -1.99442029179693
a_2 = 0.99453510743410

$H_3(z)$: (Eqn. 10.104)

$b_0 = 1.00000000000000$
$b_1 = -1.99999215766739$
$b_2 = 0.99999958921734$
$a_1 = -1.97767024097127$
$a_2 = 0.97787185728449$

$H_4(z)$: (Eqn. 10.105)

$b_0 = 1.00000000000000$
$b_1 = 0.31457928609158$
$b_2 = 1.00000000000000$
$a_1 = -0.01831558526104$
$a_2 = 0.89972679483240$

$H_5(z)$: (Eqn. 10.106)

$b_0 = 1.00000000000000$
$b_1 = 0.69054347943834$
$b_2 = 1.00000000000000$
$a_1 = -0.17586277586059$
$a_2 = 0.60203802158022$

$H_6(z)$: (Eqn. 10.107)

$b_0 = 1.00000000000000$
$b_1 = 1.68991493148360$
$b_2 = 1.00000000000000$
$a_1 = -0.46087492252454$
$a_2 = 0.16465672860857$

The quantized transfer function sections are as follows:

$H_{1qt}(z)$: (Eqn. 10.108)

$b_0 = 0.079873085021972656250000$
$b_1 = -0.159741401672363281250000$
$b_2 = 0.079873085021972656250000$
$a_1 = -1.999288082122802734375000$
$a_2 = 0.999374866485595703125000$

$H_{2qt}(z)$: (Eqn. 10.109)

$b_0 = 1.000000000000000000000000$
$b_1 = -1.999958276748657226562500$
$b_2 = 1.000009536743164062500000$
$a_1 = -1.994420289993286132812500$
$a_2 = 0.994534969329833984375000$

$H_{3qt}(z)$: (Eqn. 10.110)

$b_0 = 1.000000000000000000000000$
$b_1 = -1.999992132186889648437500$
$b_2 = 0.999999523162841796875000$
$a_1 = -1.977670192718505859375000$
$a_2 = 0.977871656417846679687500$

$H_{4qt}(z)$: (Eqn. 10.111)

b_0 = 1.000000000000000000000000
b_1 = 0.31457924842834472656250 0
b_2 = 0.99999976158142089843750 0
a_1 = -0.01831555366516113281250 0
a_2 = 0.89972662925720214843750 0

$H_{5qt}(z)$: (Eqn. 10.112)

b_0 = 1.000000000000000000000000
b_1 = 0.69054341316223144531250 0
b_2 = 0.99999976158142089843750 0
a_1 = -0.17586255073547363281250 0
a_2 = 0.60203790664672851562500 0

$H_{6qt}(z)$: (Eqn. 10.113)

b_0 = 1.000000000000000000000000
b_1 = 1.68991470336914062500000 0
b_2 = 0.99999976158142089843750 0
a_1 = -0.46087479591369628906250 0
a_2 = 0.16465663909912109375000 0

$H_{1qr}(z)$: (Eqn. 10.114)

b_0 = 0.07987332344055175781250 0
b_1 = -0.15974140167236328125000 0
b_2 = 0.07987332344055175781250 0
a_1 = -1.99928808212280273437500 0
a_2 = 0.99937486648559570312500 0

$H_{2qr}(z)$: (Eqn. 10.115)

b_0 = 1.000000000000000000000000
b_1 = -1.99995851516723632812500 0
b_2 = 1.00000119209289550781250 0
a_1 = -1.99442028999328613281250 0
a_2 = 0.99453520774841308593750 0

$H_{3qr}(z)$: (Eqn. 10.116)

b_0 = 1.000000000000000000000000
b_1 = -1.99999213218688964843750 0
b_2 = 0.99999952316284179687500 0
a_1 = -1.97767019271850585937500 0
a_2 = 0.97787189483642578125000 0

$H_{4qr}(z)$: (Eqn. 10.117)

b_0 = 1.000000000000000000000000
b_1 = 0.31457924842834472656250 0
b_2 = 1.000000000000000000000000
a_1 = -0.01831555366516113281250 0
a_2 = 0.89972686767578125000000 0

$H_{5qr}(z)$: (Eqn. 10.118)

$b_0 = 1.00000000000000000000000$
$b_1 = 0.69054341316223144531250$
$b_2 = 1.00000000000000000000000$
$a_1 = -0.17586278915405273437500$
$a_2 = 0.60203790664672851562500$

$H_{6qr}(z)$: (Eqn. 10.119)

$b_0 = 1.00000000000000000000000$
$b_1 = 1.68991494178771972656250$
$b_2 = 1.00000000000000000000000$
$a_1 = -0.46087503433227539062500$
$a_2 = 0.16465663909912109375000$

	Quantization Type	
	Truncated	**Rounded**
$H_{1e}(z)$	0.05757750813714	0.02043468379527
$H_{2e}(z)$	0.00177261661019	0.00134240831291
$H_{3e}(z)$	0.00028398412879	0.00021683830328
$H_{4e}(z)$	0.00000612905963	0.00000204911174
$H_{5e}(z)$	0.00000144949500	0.00000084215994
$H_{6e}(z)$	0.00000114799856	0.00000151439930

Table 10-22. Magnitude error calculation.

Based on Table 10-22, the rounding method produces the best results in terms of smallest error.

	COSSAP Coefficients	
$H_{1qr}(z)$	b0_COSSAP = 0	335013
	b1_COSSAP = 0	-670004
	b2_COSSAP = 0	335013
	a1_COSSAP = 1	4191318
	a2_COSSAP = 0	-4191682
$H_{2qr}(z)$	b0_COSSAP = 1	0
	b1_COSSAP = -1	-4194130
	b2_COSSAP = 1	5
	a1_COSSAP = 1	4170901
	a2_COSSAP = 0	-4171383
$H_{3qr}(z)$	b0_COSSAP = 1	0
	b1_COSSAP = -1	-4194271
	b2_COSSAP = 0	4194302
	a1_COSSAP = 1	4100646
	a2_COSSAP = 0	-4101492
$H_{4qr}(z)$	b0_COSSAP = 1	0
	b1_COSSAP = 0	1319441
	b2_COSSAP = 1	0
	a1_COSSAP = 0	76821
	a2_COSSAP = 0	-3773728
$H_{5qr}(z)$	b0_COSSAP = 1	0
	b1_COSSAP = 0	2896349
	b2_COSSAP = 1	0
	a1_COSSAP = 0	737622
	a2_COSSAP = 0	-2525130
$H_{6qr}(z)$	b0_COSSAP = 1	0
	b1_COSSAP = 1	2893713
	b2_COSSAP = 1	0
	a1_COSSAP = 0	1933050
	a2_COSSAP = 0	-690620

Table 10-23. COSSAP results of multiplier coefficient transformation.

Using Step 9.3.2 of section 8.2, the number of integer bits configured to the adder sub-blocks of each cascade stage was 2, 2, 2, 2, 2, and 2.

COSSAP RoundMode	P_e	Dead Band Range (Fixed Value Output)	COSSAP/VHDL Peak PSD
0	$7.822413942097593 \times 10^{-4}$	$-4.1267871856689 \times 10^{-2}$ to $-1.7266273498535 \times 10^{-2}$	3.490241 dB at 0.000000 Hz
1	$1.782562338712 \times 10^{-8}$	$-2.9435157775879 \times 10^{-3}$ to $3.1774044036865 \times 10^{-3}$	-24.723011 dB at 53710.937500 Hz
2	$1.782562338712 \times 10^{-8}$	$-2.9435157775879 \times 10^{-3}$ to $3.1774044036865 \times 10^{-3}$	-24.723011 dB at 53710.937500 Hz
3	$1.649846435411 \times 10^{-8}$	$-1.3280391693115 \times 10^{-2}$ to $1.0949373245239 \times 10^{-2}$	-16.812936 dB at 29296.875000 Hz

Table 10-24. Round-off noise power results of impulse response of COSSAP multiplier round-off modes.

Based on Table 10-24, *RoundMode = 3* produces the best results in terms of smallest error. Figures 10.48 through 10.51 show the power spectral density of the impulse response of the COSSAP round-off modes. Using MATLAB as the ideal model, the peak PSD of the impulse response is -38.896013 dB at 29296.875000 Hz.

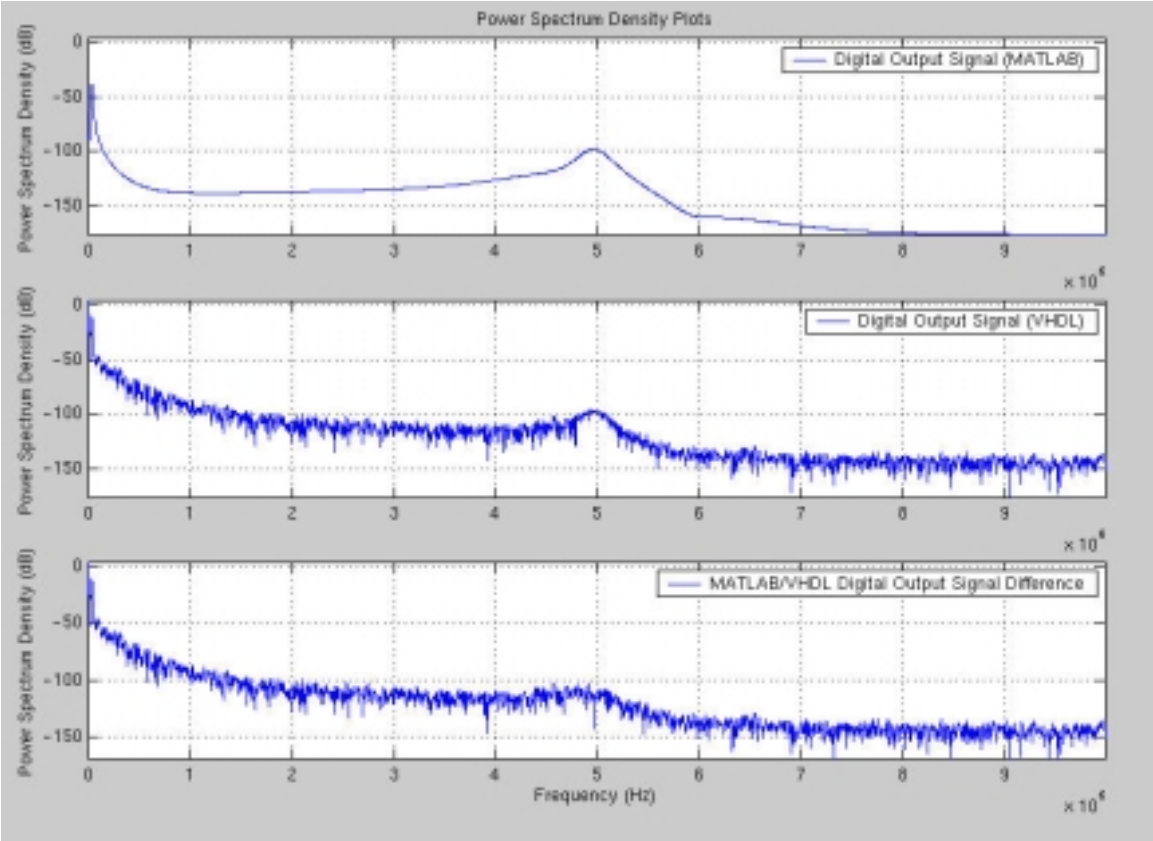


Fig. 10.48. Power spectral density results of impulse response of COSSAP *RoundMode = 0*.

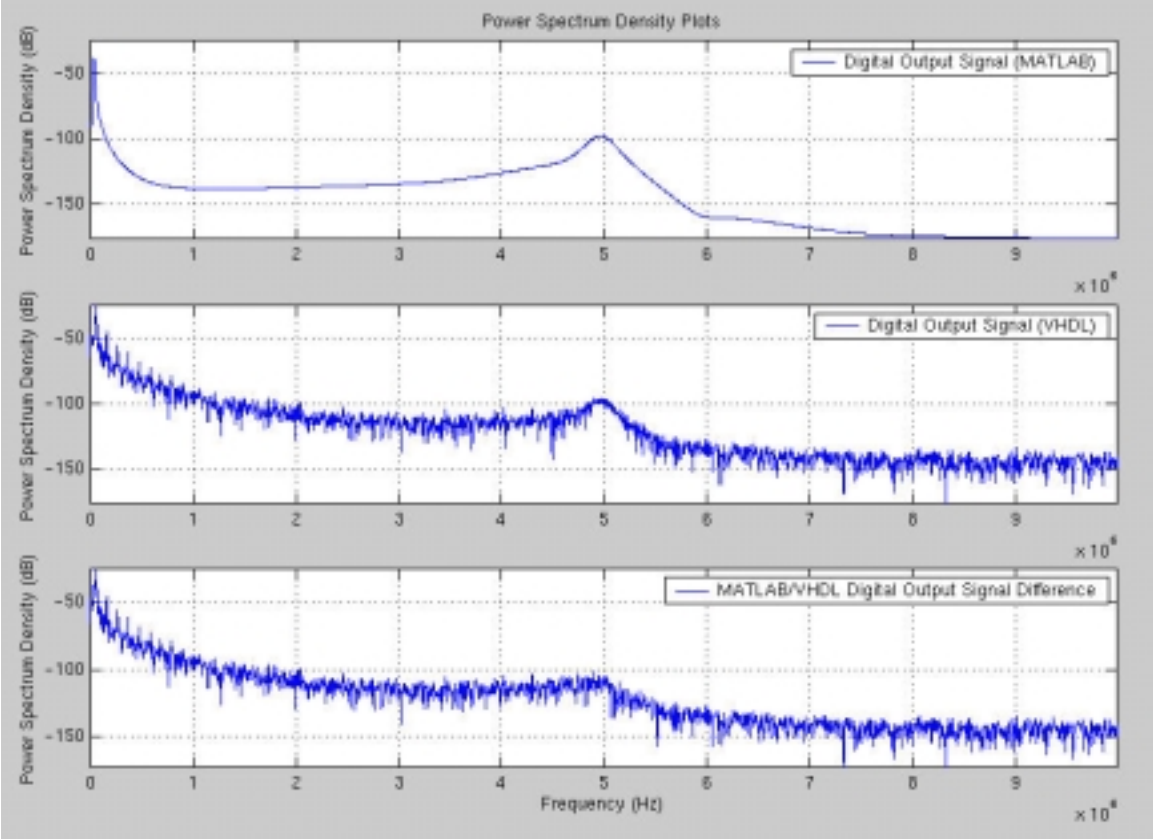


Fig. 10.49. Power spectral density results of impulse response of COSSAP *RoundMode = 1*.

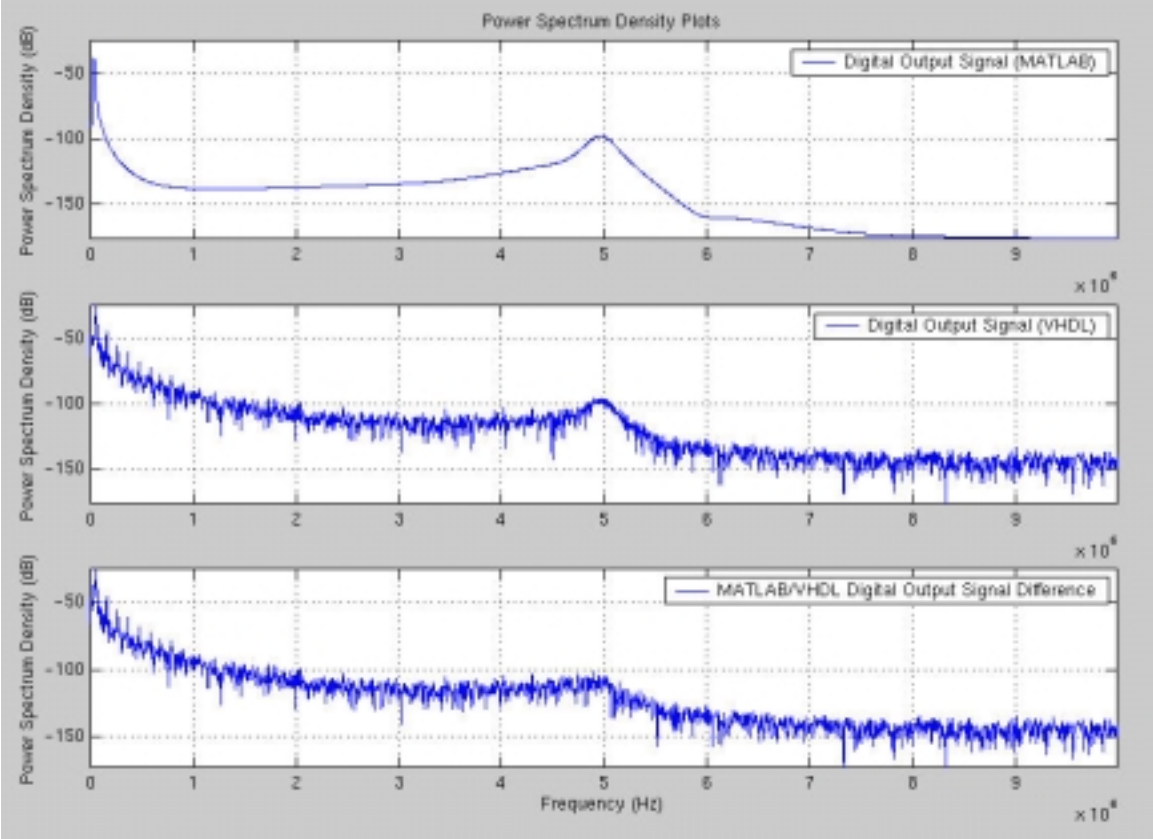


Fig. 10.50. Power spectral density results of impulse response of COSSAP *RoundMode = 2*.

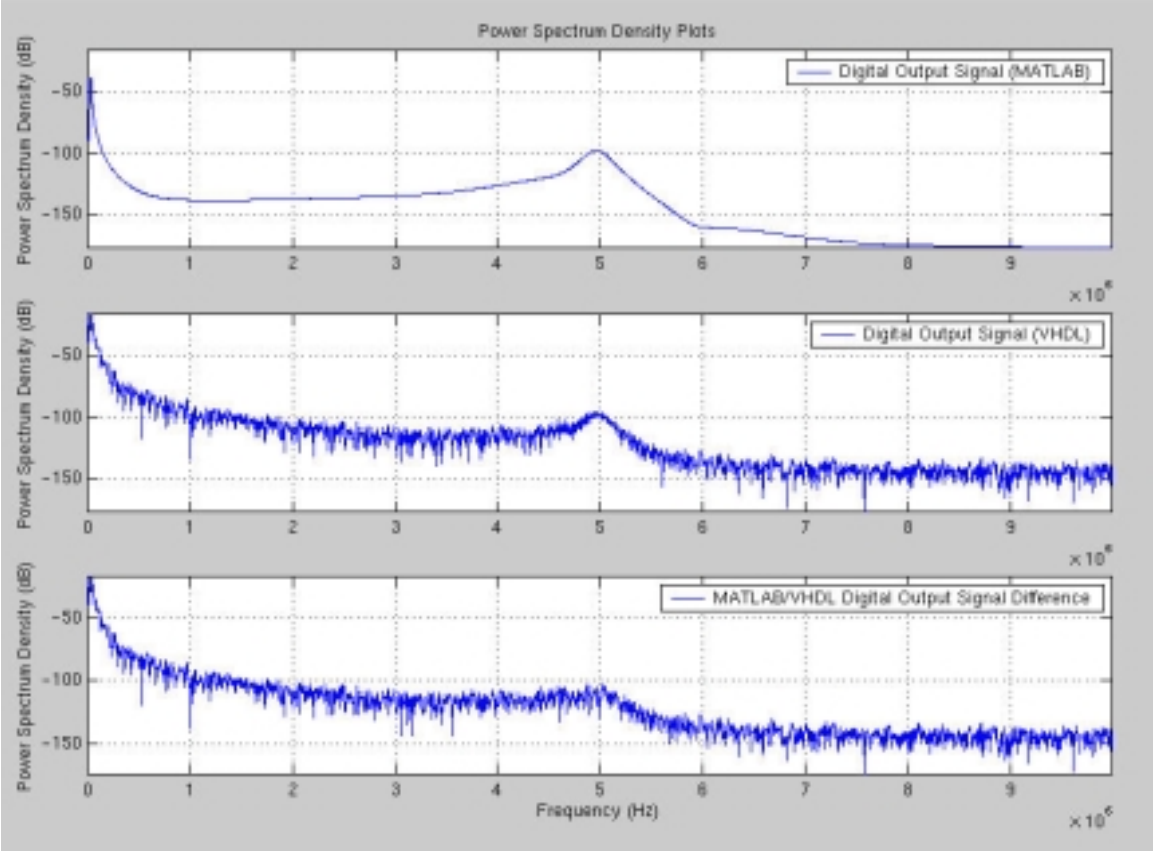


Fig. 10.51. Power spectral density results of impulse response of COSSAP *RoundMode = 3*.

10.3.2.1: Results of Validation Test #1

The passband digital input signal used is at 500kHz. The transition band input signal used is at 25kHz. The stopband digital input signal used is at 6MHz. Table 10-25 shows the tabulated results of these 3 signals.

	Round-off Noise Power (P_e)		
COSSAP RoundMode	**Passband Signal**	**Transition Band Signal**	**Stopband Signal**
0	$8.40375276549102 \times 10^{-4}$	$8.52927357129575 \times 10^{-4}$	$7.843499903049 \times 10^{-4}$
1	$3.786735796544 \times 10^{-8}$	$1.7805977991143 \times 10^{-7}$	$5.377076997478 \times 10^{-8}$
2	$3.786735796544 \times 10^{-8}$	$1.7805977991143 \times 10^{-7}$	$5.377076997478 \times 10^{-8}$
3	$8.7527318462 \times 10^{-9}$	$2.3349065241627 \times 10^{-7}$	$1.53199655723 \times 10^{-9}$

Table 10-25. Tabulated results of validation test #1.

10.3.2.2: Results of Validation Test #2

Figure 10.52 shows the digital input/output response of the unquantized bandpass Elliptic filter. The low-power passband input signal is at 500kHz. The high-power stopband input signal is at 6MHz. For the purposes of analysis, the high-power signal is 10 times greater in amplitude than the low-power signal. As can be seen from the figure, excluding the initial transient response at the output, the ideal Elliptic filter effectively removes the high-power stopband signal. Comparing the quantized filter, Figures 10.53 through 10.56 show the power spectral density plots of the four COSSAP round-off modes. Table 10-26 shows the tabulated results of the round-off noise power based on the COSSAP multiplier sub-blocks.

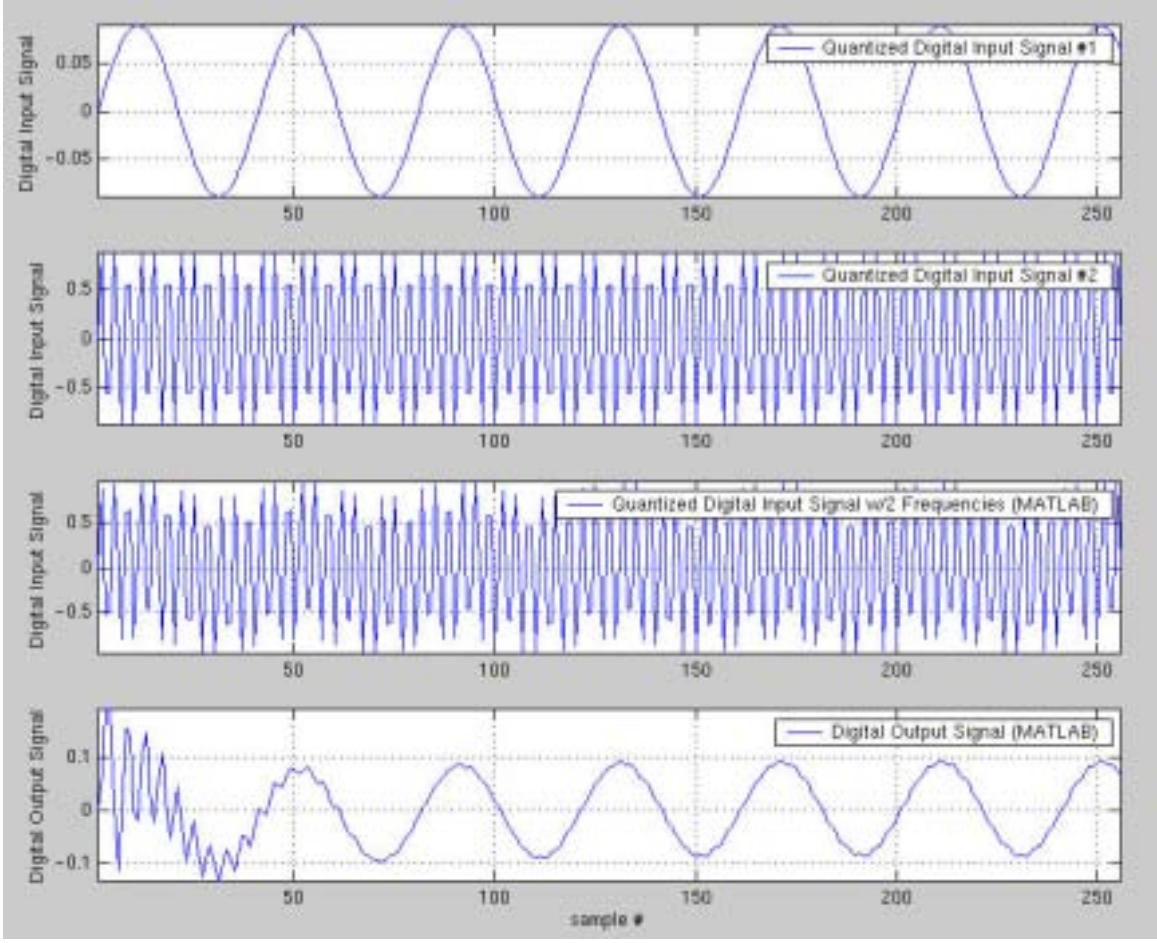


Fig. 10.52. Digital I/O response of validation test #2 for ideal Elliptic filter.

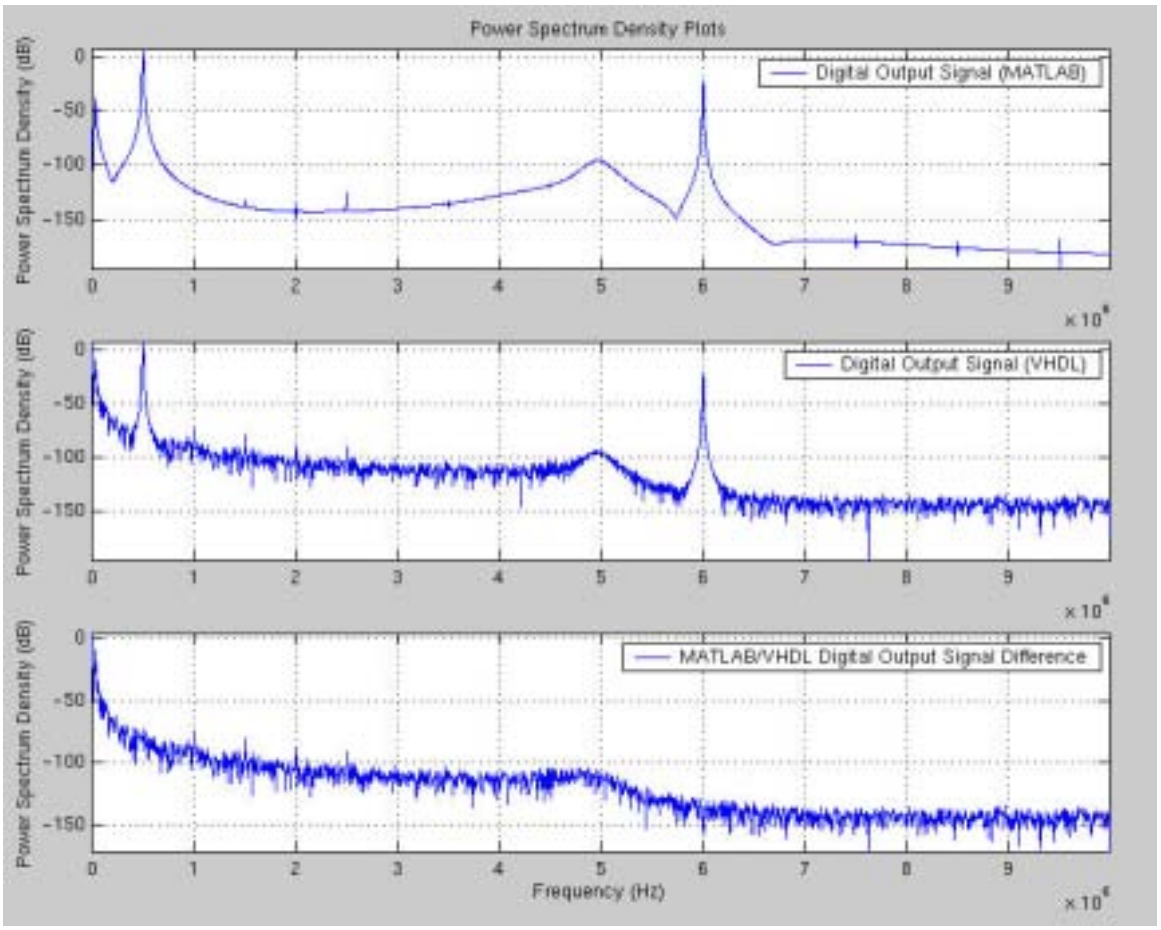


Fig. 10.53. Power spectral density plot of COSSAP *RoundMode* = *0*.

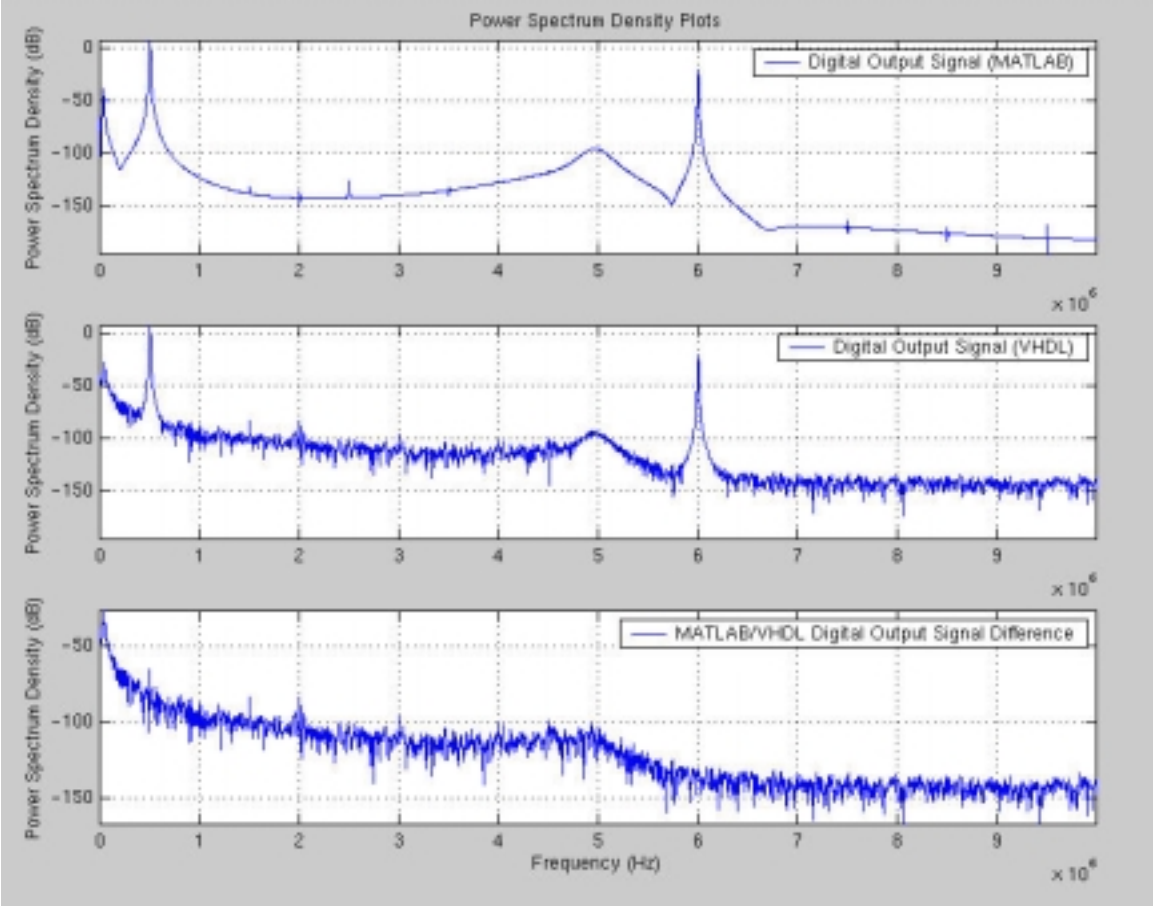


Fig. 10.54. Power spectral density plot of COSSAP *RoundMode* = *1*.

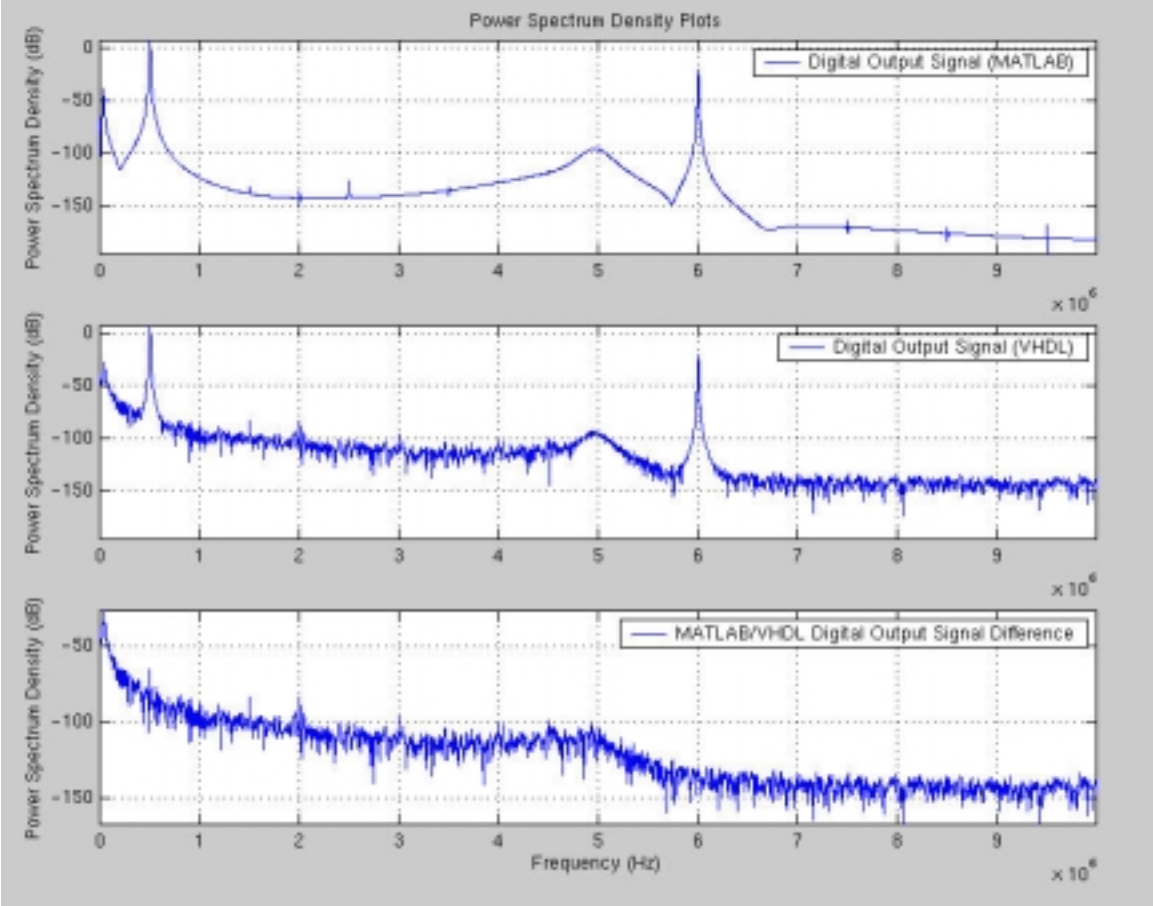


Fig. 10.55. Power spectral density plot of COSSAP *RoundMode* = *2*.

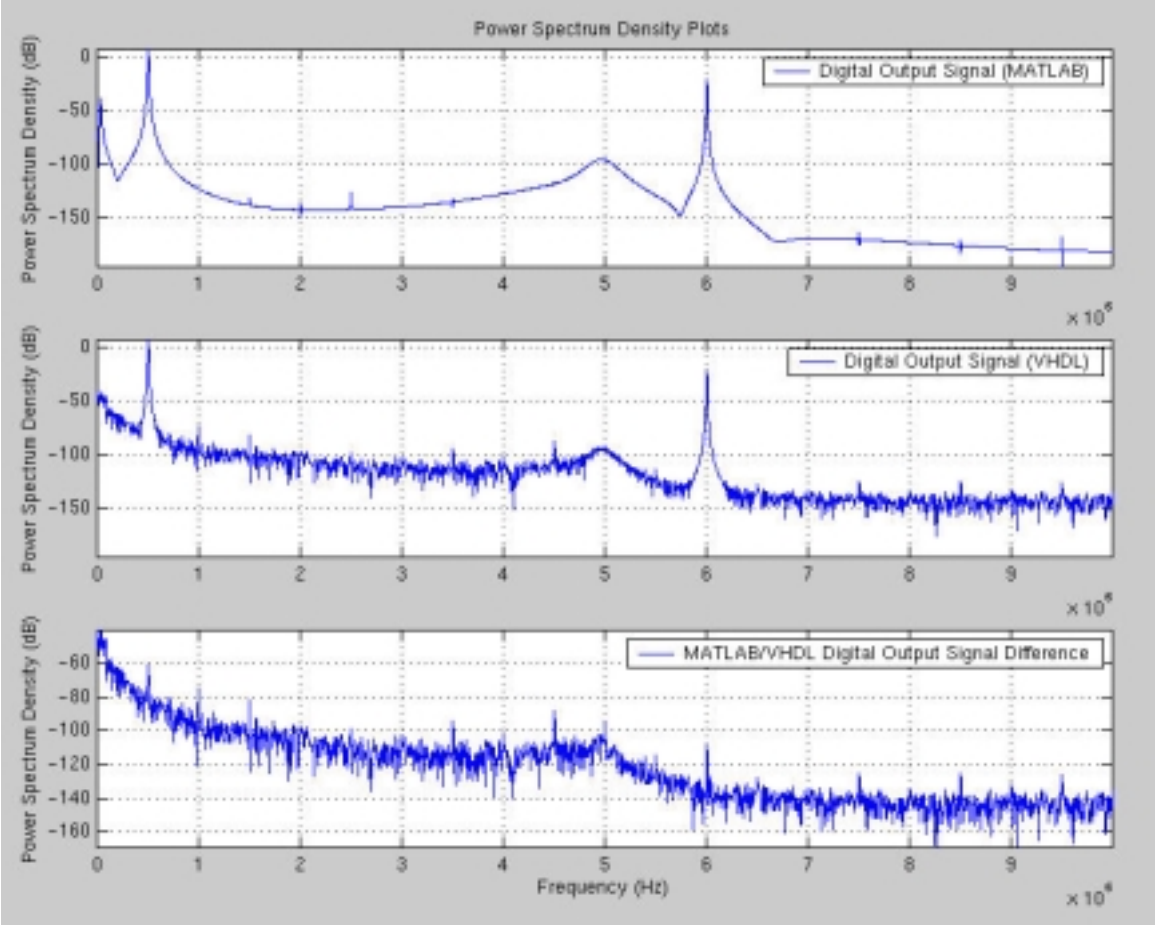


Fig. 10.56. Power spectral density plot of COSSAP *RoundMode = 3*.

COSSAP RoundMode	P_e
0	$8.9156947029847685 \times 10^{-4}$
1	$1.519423402377 \times 10^{-8}$
2	$1.519423402377 \times 10^{-8}$
3	$2.238765273714 \times 10^{-8}$

Table 10-26. Tabulated results of 4 COSSAP round-off modes.

Based on Table 10-26, *RoundMode = 1* and *RoundMode = 2* produce the best results in terms of smallest error.

CHAPTER 11: Research Summary and Future Work

The objective of this research was to define a methodology for designing fixed-point IIR digital filters using modeling tools. A significant observation realized during the course of this research dealt with the parallel structure implementation. Designing more than two transfer functions sections introduces the problem on how to go about summing the sections together. Because of the fixed-point representation, the non-linear aspect of summing could potentially be a setback as was the case with trying to implement the data communications/imaging bandwidth. Deciding which sections to add together requires future work in terms of analysis. The cascade structure implementation provides the designer better control in terms of handling the interface from section to section.

An important question is which COSSAP round-off mode should a designer use for synthesis. No general conclusion could be drawn from the research examples. In fact, for a given filter, the best round-off mode depends on which frequency band is analyzed. Therefore, a designer must try all four modes in each region to determine the best tradeoff. As can be seen from the round-off noise power results throughout this research, depending on which frequency region is most important to the designer (passband, transition band, stopband), a specific round-off mode could be chosen for synthesis. This is left to the designer's discretion. There was one noticeably consistent result when performing analysis on the 4 round-off modes. *RoundMode = 0* produced the greatest round-off noise power when compared to the other 3 round-off modes.

This research provided a technical bridge between DSP design techniques and digital design. Bridging the two to go from an ideal representation to real world model required trial-and-error approaches on the part of this researcher. The trial-and-error approaches produced technical remedies to quantization problems coming from fixed-point multipliers and adders. For synthesis timing constraints, going from an ideal, delay-less model to a technology library-based model required the researcher to incorporate memory elements in the final filter design. Combinational logic delay is a factor a digital designer must deal with when moving to lower levels of design abstraction. Incorporating a parallel register methodology significantly reduces the effect of this problem. It reduces this timing problem by eliminating potentially timing violations at the filter output.

Overall, the methodology outlined in this research is technically sound because it provides an interface between DSP design techniques and digital design.

BIBLIOGRAPHY

[Aggarwal] J.K. Aggarwal, “Digital Signal Processing”, Western Periodicals Company, 1979.

[Armstrong] James R. Armstrong, F. Gail Gray, “Structured Logic Design with VHDL”, Prentice-Hall, Inc., 1993.

[Bailey] Daniel A. Bailey, “Simulation and implementation of Fixed-Point Digital Filter Structures”, M.S. Thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 1995.

[Chirlian] Paul M. Chirlian, “Signals and Filters”, Van Nostrand Reinhold, 1994.

[Fettweis] Alfred Fettweis, “Roundoff Noise and Attenuation Sensitivity in Digital Filters with Fixed-Point Arithmetic,” IEEE Transactions on Circuit Theory, Vol. 20, March 20, 1973, pp. 174-175.

[Gold] Bernard Gold, Lawrence R. Rabiner, “Theory and Application of Digital Signal Processing”, Prentice-Hall, Inc., 1975.

[Higgins] Richard J. Higgins, “Digital Signal Processing in VLSI”, Prentice-Hall, Inc., 1990.

[Jackson1] Leland B. Jackson, “Digital Filters and Signal Processing”, Kluwer Academic Publishers, 1996.

[Jackson2] Leland B. Jackson, “Roundoff Noise Bounds Derived from Coefficient Sensitivities for Digital Filters,” IEEE Transactions on Circuit and Systems, Vol. 23, August 1976, pp. 481-484.

[Karl] John H. Karl, “An Introduction to Digital Signal Processing”, Academic Press, 1989.

[Manolakis] Dimitris G. Manolakis, John G. Proakis, “Digital Signal Processing: Principles, Algorithms, and Applications”, Macmillan Publishing Company, 1992.

[Mullis] Clifford T. Mullis, Richard A. Roberts, “Digital Signal Processing”, Addison-Wesley Publishing Company, Inc., 1987.

[Oppenheim1] Alan V. Oppenheim, Ronald W. Schafer, “Digital Signal Processing”, Prentice-Hall, Inc., 1975.

[Oppenheim2] Alan V. Oppenheim, Ronald W. Schafer, “Digital Signal Processing”, Prentice-Hall, Inc., 1989.

[Oppenheim3] Alan V. Oppenheim, Ronald W. Schafer, “Discrete-Time Signal Processing”, Prentice-Hall, Inc., 1989.

[Rabiner] Lawrence R. Rabiner, Charles M. Rader, “Digital Signal Processing”, IEEE Press, 1972.

APPENDIX A: Generic VHDL Library

Fixed-point Multipliers

Figure 7.3 is used as reference for this appendix. The initial VHDL-generated code, excluding comments, is as follows:

```
**************************************************************************************************
library IEEE ;
library BITTRUE_VHDLSNPS ;

use IEEE.std_logic_1164.all ;
use IEEE.std_logic_arith.all ;
use BITTRUE_VHDLSNPS.COSSAP_PACKAGE_SYNOPSYS.all ;
Use BITTRUE_VHDLSNPS.fxp_arith.all ;

entity mult is
   generic ( SCHEDULE_LENGTH : in INTEGER := 1 ) ;
   port    ( DIG_IN : in STD_LOGIC_VECTOR((20-1) downto 0) ;
             clock : in STD_LOGIC ;
             reset : in STD_LOGIC ;
             DIG_OUT : out STD_LOGIC_VECTOR((20-1) downto 0) ) ;
end mult ;

architecture behavior of mult is
  constant RoundProdWidth_M_M5_1_1 : INTEGER := 1 + 1 + 20 - 1 ;
  constant Min_Sched_Length : integer := 1;
  constant Sched_Mpy : integer := Schedule_length / Min_Sched_Length;
begin

 main: process
       variable DIG_IN_temp : STD_LOGIC_VECTOR((20-1) downto 0);
       variable DIG_OUT_temp : STD_LOGIC_VECTOR((20-1) downto 0);
       variable Input2_M_M5_1_1 : SIGNED(20 - 1 DOWNTO 0) ;
       variable RoundProd_M_M5_1_1: SIGNED(ROUNDPRODWIDTH_M_M5_1_1-1 DOWNTO
0);
       begin
         reset_loop: loop
                       DIG_IN_temp := (others => '0');
                       DIG_OUT_temp := (others => '0');
                       DIG_OUT <=  (others => '0');
                       wait until (clock'event and clock = '1');
                       exit reset_loop when reset = '1';
         main_loop: loop
                DIG_IN_temp := DIG_IN ;
                Input2_M_M5_1_1 := const2fxp(0, 128575, 1, 20);
                RoundProd_M_M5_1_1 := fxp_round(SIGNED(DIG_IN_temp) *
                       Input2_M_M5_1_1, 0, RoundProdWidth_M_M5_1_1);
                DIG_OUT_temp :=
                STD_LOGIC_VECTOR(fxp_saturate(RoundProd_M_M5_1_1, 1, 20));
                 if (Sched_Mpy > 1) then
                     read_wloop_1: for i in 1 to  (Sched_Mpy-1) loop
                       wait until (clock'event and clock = '1');
                       exit reset_loop when reset = '1';
                     end loop;
                 end if;
                DIG_OUT <= DIG_OUT_temp;
```

```
                wait until (clock'event and clock = '1');
                exit reset_loop when reset = '1';
                     end loop main_loop;
         end loop reset_loop;
 end process main;

end behavior ;
```

****

The first editing step is removing the generic statement, the ‘clock’ input port, and the ‘reset’ input port. This first edit should be performed in the entity declaration. The second edit entails removing all reset and edge-triggered clock references and unwanted variables and possible constants in the architecture. The resulting manually edited VHDL-generated code, at this point, is as follows:

****

```
library IEEE ;
library BITTRUE_VHDLSNPS ;

use IEEE.std_logic_1164.all ;
use IEEE.std_logic_arith.all ;
use BITTRUE_VHDLSNPS.COSSAP_PACKAGE_SYNOPSYS.all ;
Use BITTRUE_VHDLSNPS.fxp_arith.all ;

entity mult is

   port    ( DIG_IN : in STD_LOGIC_VECTOR((20-1) downto 0) ;
             DIG_OUT : out STD_LOGIC_VECTOR((20-1) downto 0) ) ;
end mult ;

architecture behavior of mult is
  constant RoundProdWidth_M_M5_1_1 : INTEGER := 1 + 1 + 20 - 1 ;
begin

 main: process(DIG_IN)
       variable Input2_M_M5_1_1 : SIGNED(20 - 1 DOWNTO 0) ;
       variable RoundProd_M_M5_1_1: SIGNED(ROUNDPRODWIDTH_M_M5_1_1-1 DOWNTO
0);
       begin
        Input2_M_M5_1_1 := const2fxp(0, 128575, 1, 20);
        RoundProd_M_M5_1_1 := fxp_round(SIGNED(DIG_IN) *
                        Input2_M_M5_1_1, 0, RoundProdWidth_M_M5_1_1);
        DIG_OUT <= STD_LOGIC_VECTOR(fxp_saturate(RoundProd_M_M5_1_1, 1, 20));
       end process main;
end behavior ;
```

****

There should only be 2 VHDL variables remaining. These variables should have the *Input2_* and *RoundProd_* prefixes. The only VHDL constant remaining should have the *RoundProdWidth_* prefix. It is noted here that this constant is obviously equal to 1. Briefly stated, this number was calculated as the n-bit length plus the summation of integer bits allocated to represent the input, the constant, and the output. The third editing step requires the designer to enter the generic statement in the entity as follows:

generic (RND_MODE, INT, DEC, INT_LENGTH, BIN_LENGTH: in INTEGER);

The fourth editing step requires the designer to replace the number 20 with the generic variable *BIN_LENGTH* throughout the code. Sum the remaining numbers in the constant declaration to get the number 1. The fifth and final editing step requires the designer to replace the reference *const2fxp()* with the reference *const2fxp(INT, DEC, INT_LENGTH, BIN_LENGTH)*. The final VHDL code is as follows:

```
**************************************************************************************************
library IEEE ;
library BITTRUE_VHDLSNPS ;

use IEEE.std_logic_1164.all ;
use IEEE.std_logic_arith.all ;
use BITTRUE_VHDLSNPS.COSSAP_PACKAGE_SYNOPSYS.all ;
Use BITTRUE_VHDLSNPS.fxp_arith.all ;

entity mult is

   generic ( RND_MODE, INT, DEC, INT_LENGTH, BIN_LENGTH : in INTEGER ) ;
   port    ( DIG_IN : in STD_LOGIC_VECTOR((BIN_LENGTH-1) downto 0) ;
             DIG_OUT : out STD_LOGIC_VECTOR((BIN_LENGTH-1) downto 0) ) ;
end mult ;

architecture behavior of mult is
  constant RoundProdWidth_M_M5_1_1 : INTEGER := BIN_LENGTH + 1 ;
begin

 main: process(DIG_IN)
       variable Input2_M_M5_1_1 : SIGNED(BIN_LENGTH - 1 DOWNTO 0) ;
       variable RoundProd_M_M5_1_1: SIGNED(ROUNDPRODWIDTH_M_M5_1_1-1 DOWNTO
0);
       begin
        Input2_M_M5_1_1 := const2fxp(INT, DEC, INT_LENGTH, BIN_LENGTH);
        RoundProd_M_M5_1_1 := fxp_round(SIGNED(DIG_IN) * Input2_M_M5_1_1,
                              RND_MODE, RoundProdWidth_M_M5_1_1);
        DIG_OUT <= STD_LOGIC_VECTOR(fxp_saturate(RoundProd_M_M5_1_1, 1,
                              BIN_LENGTH));
       end process main;
end behavior ;

**************************************************************************************************
```

The VHDL code edit is now purely combinational and is used as a library reference. To ensure that functionality did not change, it is left to the designer to compare the initial VHDL-generated code to the final edited code via a VHDL testbench. Using the COSSAP function *const2fxp()*, an n-bit fixed-point representation for a multiplier coefficient is realized. The generic variables *INT*, *DEC*, *INT_LENGTH*, and *BIN_LENGTH* are parameters passed to this function to represent this multiplier coefficient. The generic variable *RND_MODE*, in the COSSAP function *fxp_round()* in the above code, represent one of four round-off modes explained in Chapter 6. Notice that in the COSSAP function *fxp_saturate()* one of the parameters is the integer 1. This is the set saturation mode that was found to produce the best result of the three available COSSAP saturation modes. The method by which this mode was found to be the best is found in Chapter 5. The reason for these 5 editing steps is to design a generic combinational multiplier sub-block to be used solely for design verification.

2-input Fixed-point Adders

Figure 7.4 is used as visual reference for this appendix. The initial VHDL-generated code after executing **xvcg**, excluding comments, is as follows:

```
**********************************************************************************************************
library IEEE ;
library BITTRUE_VHDLSNPS ;

use IEEE.std_logic_1164.all ;
use IEEE.std_logic_arith.all ;
use BITTRUE_VHDLSNPS.COSSAP_PACKAGE_SYNOPSYS.all ;
Use BITTRUE_VHDLSNPS.fxp_arith.all ;

entity adder2 is

   generic ( SCHEDULE_LENGTH : in INTEGER := 1 ) ;
   port    ( IN0 : in STD_LOGIC_VECTOR((20-1) downto 0) ;
             IN1 : in STD_LOGIC_VECTOR((20-1) downto 0) ;
             clock : in STD_LOGIC ;
             reset : in STD_LOGIC ;
             OUTPUT : out STD_LOGIC_VECTOR((20-1) downto 0) ) ;

end adder2 ;


architecture behavior of adder2 is
  constant RoundWidth_M_M22_1_1 :  INTEGER := MAXOF2INT(1, 1) + 20 - 1 + 1 ;
  constant Min_Sched_Length : integer := 1;
  constant Sched_Mpy : integer := Schedule_length / Min_Sched_Length;
begin

 main: process
       variable IN0_temp : STD_LOGIC_VECTOR((20-1) downto 0);
       variable IN1_temp : STD_LOGIC_VECTOR((20-1) downto 0);
       variable OUTPUT_temp : STD_LOGIC_VECTOR((20-1) downto 0);
       variable RoundSum_M_M22_1_1: SIGNED(ROUNDWIDTH_M_M22_1_1 - 1 DOWNTO 0)
;
       begin
      reset_loop: loop
                  IN0_temp := (others => '0');
                  IN1_temp := (others => '0');
                  OUTPUT_temp := (others => '0');
                  OUTPUT <=  (others => '0');
            wait until (clock'event and clock = '1');
              exit reset_loop when reset = '1';
            main_loop : loop
                        IN0_temp := IN0 ;
                        IN1_temp := IN1 ;
                  if (20 > 20) then
                          RoundSum_M_M22_1_1 :=
      fxp_round(SIGNED(IN1_temp(IN1_temp'high) & IN1_temp) +
SIGNED(IN0_temp),
                    0, RoundWidth_M_M22_1_1);
                  else
                          RoundSum_M_M22_1_1 :=
```

```
        fxp_round(SIGNED(IN0_temp(IN0_temp'high) & IN0_temp) +
SIGNED(IN1_temp),
                  0, RoundWidth_M_M22_1_1);
                  end if;
                  OUTPUT_temp :=
      STD_LOGIC_VECTOR(fxp_saturate(RoundSum_M_M22_1_1, 1, 20));
                  if (Sched_Mpy > 1) then
                    read_wloop_1: for i in 1 to  (Sched_Mpy-1) loop
                        wait until (clock'event and clock = '1');
                        exit reset_loop when reset = '1';
                    end loop;
                  end if;
                  OUTPUT <= OUTPUT_temp;
                  wait until (clock'event and clock = '1');
                  exit reset_loop when reset = '1';
            end loop main_loop;
      end loop reset_loop;
       end process main;

end behavior ;

**************************************************************************************************
```

The designer should enter the generic statement in the entity as follows:

generic (BIN_LENGTH: in INTEGER);

The only VHDL constant remaining should have the *RoundWidth_* prefix. The only VHDL variable remaining should have the *RoundSum_* prefix. The final VHDL generic code is as follows:

```
**************************************************************************************************
library IEEE ;
library BITTRUE_VHDLSNPS ;

use IEEE.std_logic_1164.all ;
use IEEE.std_logic_arith.all ;
use BITTRUE_VHDLSNPS.COSSAP_PACKAGE_SYNOPSYS.all ;
Use BITTRUE_VHDLSNPS.fxp_arith.all ;

entity adder2 is

   generic ( BIN_LENGTH : in INTEGER ) ;
   port    ( IN0 : in STD_LOGIC_VECTOR((BIN_LENGTH-1) downto 0) ;
             IN1 : in STD_LOGIC_VECTOR((BIN_LENGTH-1) downto 0) ;
             OUTPUT : out STD_LOGIC_VECTOR((BIN_LENGTH-1) downto 0) ) ;

end adder2 ;


architecture behavior of adder2 is
  constant RoundWidth_M_M22_1_1 :  INTEGER := BIN_LENGTH + 1 ;
begin

 main: process(IN0,IN1)
       variable RoundSum_M_M22_1_1: SIGNED(ROUNDWIDTH_M_M22_1_1 - 1 DOWNTO 0)
;
       begin
```

```
      RoundSum_M_M22_1_1 := fxp_round(SIGNED(IN0(IN0'high) & IN0) +
                        SIGNED(IN1), 0, RoundWidth_M_M22_1_1);
      OUTPUT <= STD_LOGIC_VECTOR(fxp_saturate(RoundSum_M_M22_1_1, 1,
BIN_LENGTH));
       end process main;

end behavior ;

*********************************************************************************************
```

3-input Fixed-point Adders

Figure 7.5 is used as visual reference for this appendix. The final VHDL generic code is as follows:

```
*********************************************************************************************
library IEEE ;
library BITTRUE_VHDLSNPS ;

use IEEE.std_logic_1164.all ;
use IEEE.std_logic_arith.all ;
use BITTRUE_VHDLSNPS.COSSAP_PACKAGE_SYNOPSYS.all ;
Use BITTRUE_VHDLSNPS.fxp_arith.all ;

entity adder3 is

   generic ( BIN_LENGTH : in INTEGER ) ;
   port    ( IN0 : in STD_LOGIC_VECTOR((BIN_LENGTH-1) downto 0) ;
             IN1 : in STD_LOGIC_VECTOR((BIN_LENGTH-1) downto 0) ;
             IN2 : in STD_LOGIC_VECTOR((BIN_LENGTH-1) downto 0) ;
             OUTPUT : out STD_LOGIC_VECTOR((BIN_LENGTH-1) downto 0) ) ;

end adder3 ;


architecture behavior1 of adder3 is
 constant RoundWidth_M_M15_1_1: INTEGER:= BIN_LENGTH + 1 ;
 constant RoundWidth_M_M16_1_2: INTEGER:= BIN_LENGTH + 1 ;
begin

  main: process(IN0,IN1,IN2)
        variable RoundSum_M_M15_1_1:  SIGNED(ROUNDWIDTH_M_M15_1_1-1 DOWNTO 0)
;
        variable RoundSum_M_M16_1_2:  SIGNED(ROUNDWIDTH_M_M16_1_2-1 DOWNTO 0)
;
        variable SIG_4M_M15_1_1 : STD_LOGIC_VECTOR((BIN_LENGTH-1) downto  0)
;
      begin
          RoundSum_M_M15_1_1 := fxp_round(SIGNED(IN0(IN0'high) & IN0) +
                        SIGNED(IN1), 0, RoundWidth_M_M15_1_1);
          SIG_4M_M15_1_1 := STD_LOGIC_VECTOR(fxp_saturate(RoundSum_M_M15_1_1,
1, BIN_LENGTH));
          RoundSum_M_M16_1_2 :=
fxp_round(SIGNED(SIG_4M_M15_1_1(SIG_4M_M15_1_1'high) &
                        SIG_4M_M15_1_1) + SIGNED(IN2), 0,
RoundWidth_M_M16_1_2);
```

```
          OUTPUT <= STD_LOGIC_VECTOR(fxp_saturate(RoundSum_M_M16_1_2, 1,
BIN_LENGTH));
      end process main;

end behavior1 ;

*********************************************************************************************
```

Fixed-point Delay Sub-blocks

Figure 7.6 is used as visual reference for this appendix. The final VHDL generic code is as follows:

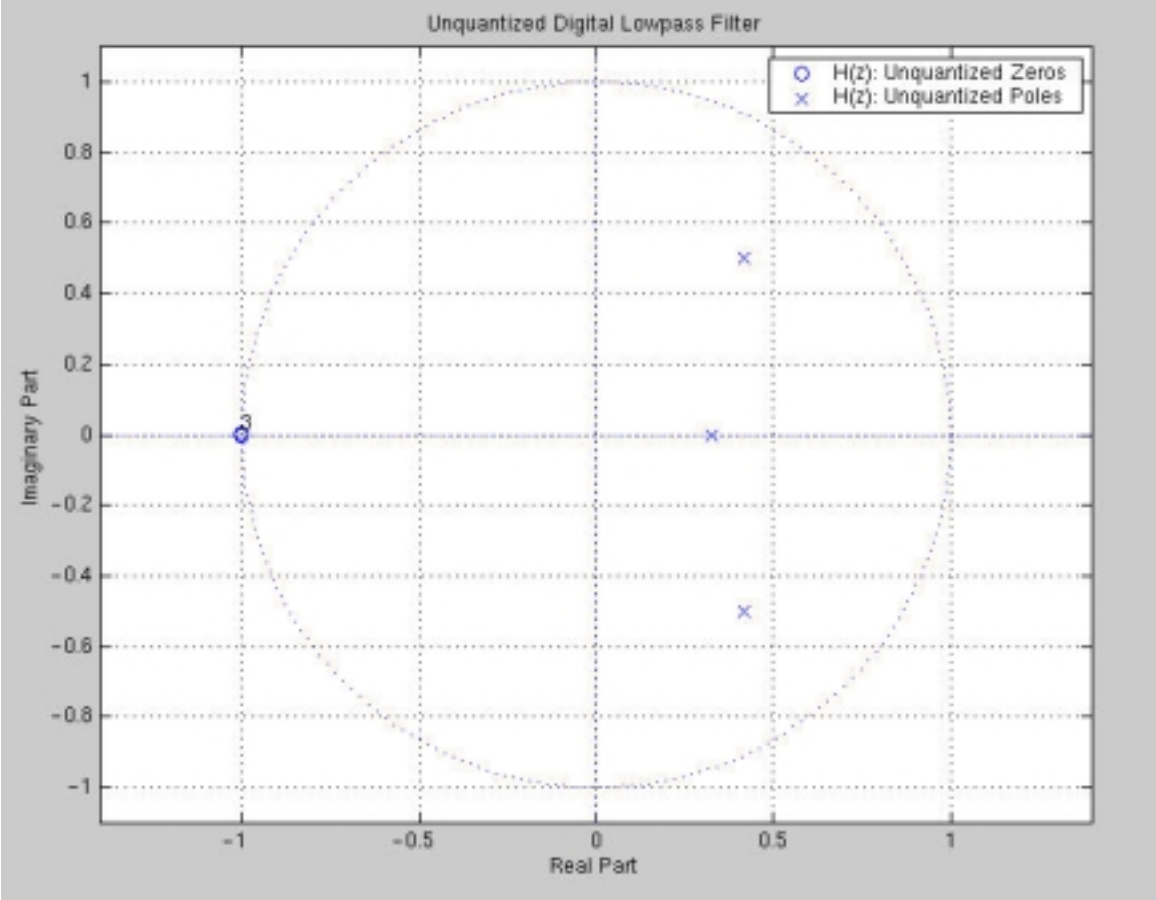
```
*********************************************************************************************
library IEEE ;
use IEEE.std_logic_1164.all ;

entity delay is

   generic ( BIN_LENGTH: in INTEGER);
   port    ( INPUT : in STD_LOGIC_VECTOR((BIN_LENGTH-1) downto 0) ;
             OUTPUT : out STD_LOGIC_VECTOR((BIN_LENGTH-1) downto 0);
               CLK, RESET : in STD_LOGIC ) ;

end delay;

architecture behavior of delay is
begin

  main: process(CLK,RESET)
      begin
       if (RESET = '1') then
             for i in 0 to BIN_LENGTH-1 loop
                  OUTPUT(i) <= '0';
           end loop;
       elsif (CLK'EVENT and CLK = '1') then
            OUTPUT <= INPUT;
       end if;
      end process main;

end behavior ;

*********************************************************************************************
```

Generic VHDL Structural Code of Figure 7.1.

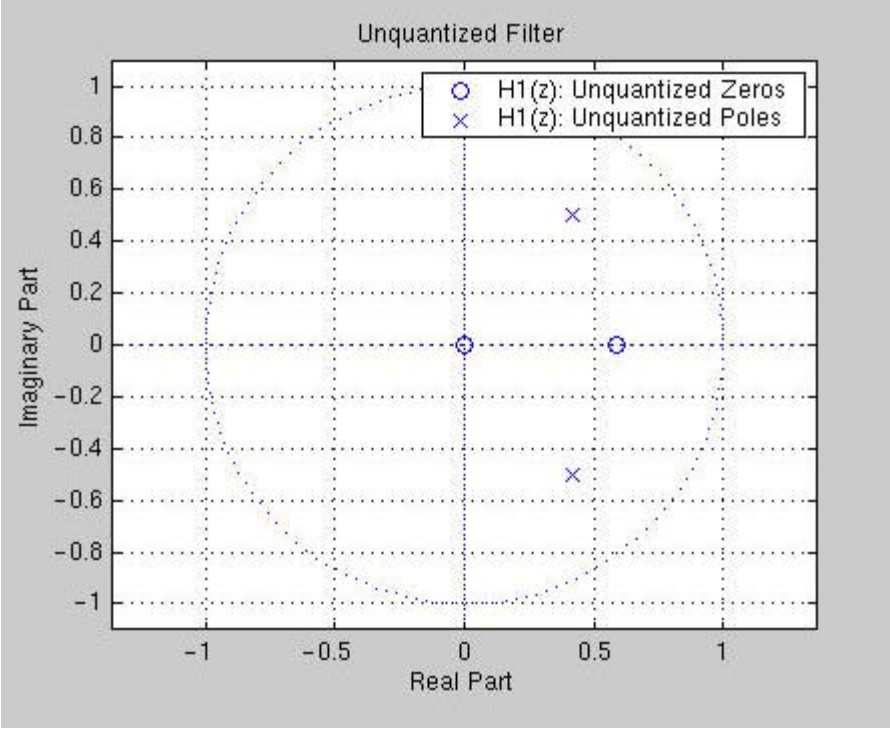
```
library IEEE ;
library BITTRUE_VHDLSNPS ;

use IEEE.std_logic_1164.all ;
use IEEE.std_logic_arith.all ;
use BITTRUE_VHDLSNPS.COSSAP_PACKAGE_SYNOPSYS.all ;
Use BITTRUE_VHDLSNPS.fxp_arith.all ;

entity LOWPASS_FILTER is
```

```
   generic ( RND_MODE, INT_LENGTH, BIN_LENGTH : INTEGER ) ;
   port    ( DIG_IN : in STD_LOGIC_VECTOR((BIN_LENGTH-1) downto 0) ;
             DIG_OUT : out STD_LOGIC_VECTOR((BIN_LENGTH-1) downto 0) ;
             CLK, RESET : in STD_LOGIC ) ;
end LOWPASS_FILTER ;

architecture STRUCTURE of LOWPASS_FILTER is
component ADDER2
   generic ( BIN_LENGTH : INTEGER ) ;
   port    ( IN0 : in STD_LOGIC_VECTOR((BIN_LENGTH-1) downto 0) ;
             IN1 : in STD_LOGIC_VECTOR((BIN_LENGTH-1) downto 0) ;
             OUTPUT : out STD_LOGIC_VECTOR((BIN_LENGTH-1) downto 0) ) ;
end component;
component MULT
   generic ( RND_MODE,INT,DEC,INT_LENGTH,BIN_LENGTH : INTEGER ) ;
   port    ( IN0 : in STD_LOGIC_VECTOR((BIN_LENGTH-1) downto 0) ;
             OUTPUT : out STD_LOGIC_VECTOR((BIN_LENGTH-1) downto 0) ) ;
end component;
component DELAY
   generic ( BIN_LENGTH : INTEGER ) ;
   port    ( INPUT : in STD_LOGIC_VECTOR((BIN_LENGTH-1) downto 0) ;
             OUTPUT : out STD_LOGIC_VECTOR((BIN_LENGTH-1) downto 0 ) ;
             CLK, RESET : in STD_LOGIC ) ;
end component;

signal DLY_IN_0, DLY_OUT_0    : STD_LOGIC_VECTOR((BIN_LENGTH-1) downto 0) ;

signal B0_OUT, B1_OUT, A1_OUT : STD_LOGIC_VECTOR((BIN_LENGTH-1) downto 0) ;

signal DIG_OUT_INT            : STD_LOGIC_VECTOR((BIN_LENGTH-1) downto 0) ;

begin
      DIG_OUTPUT: DIG_OUT <= DIG_OUT_INT;

      DELAY_DELAY_0: DELAY generic map ( BIN_LENGTH )
                     port map ( DLY_IN_0, DLY_OUT_0, CLK, RESET );

      B0: MULT generic map ( RND_MODE, 0, 128575, INT_LENGTH, BIN_LENGTH )
                port map ( DIG_IN, B0_OUT );
      B1: MULT generic map ( RND_MODE, 0, 128575, INT_LENGTH, BIN_LENGTH )
                port map ( DIG_IN, B1_OUT );
      A1: MULT generic map ( RND_MODE, 0, 267138, INT_LENGTH, BIN_LENGTH )
                port map ( DIG_OUT_INT, A1_OUT );

      ADDER_0: ADDER2 generic map ( BIN_LENGTH )
                    port map ( B0_OUT, DLY_OUT_0, DIG_OUT_INT );
      ADDER_1: ADDER2 generic map ( BIN_LENGTH )
                    port map ( B1_OUT, A1_OUT, DLY_IN_0 );
end STRUCTURE ;
```

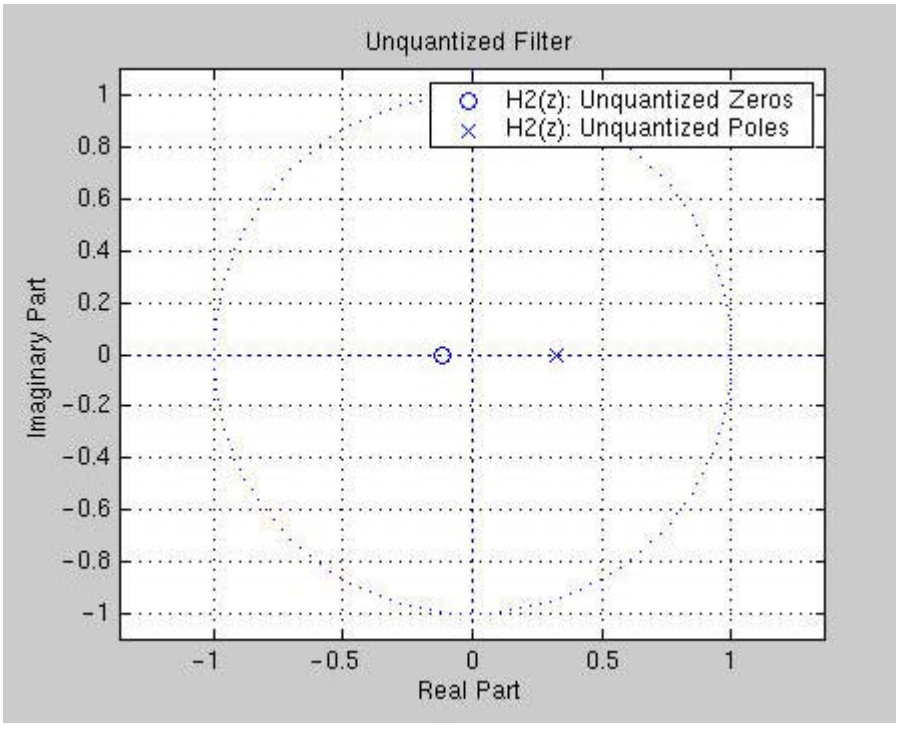
APPENDIX B: Additional Results of 3rd order 16-bit Lowpass Butterworth Filter



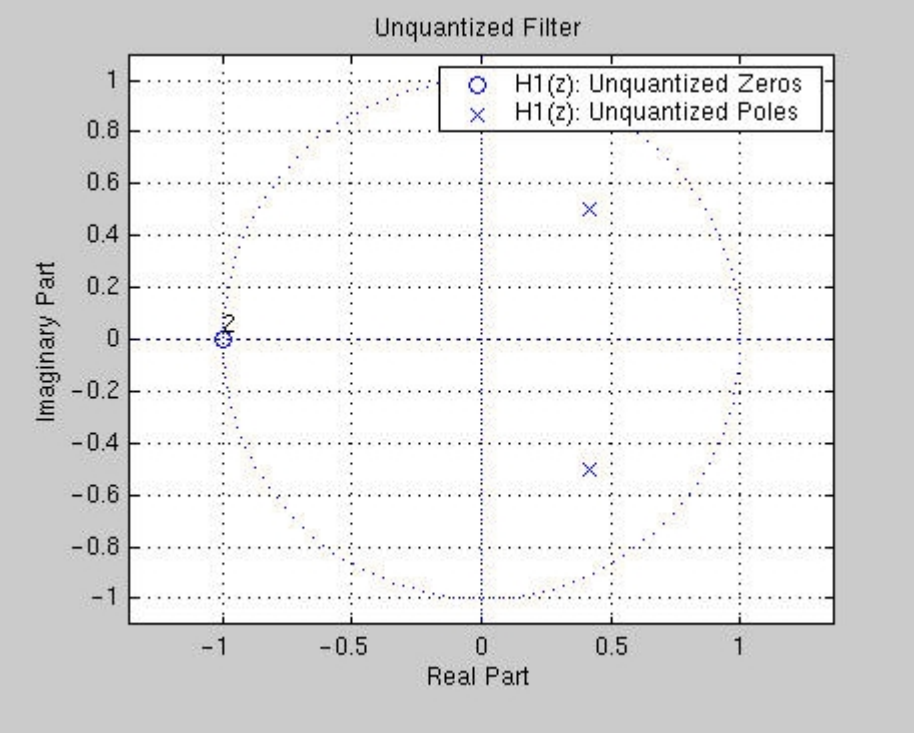
Pole/Zero Plot of Overall Transfer Function in Section 9.1.



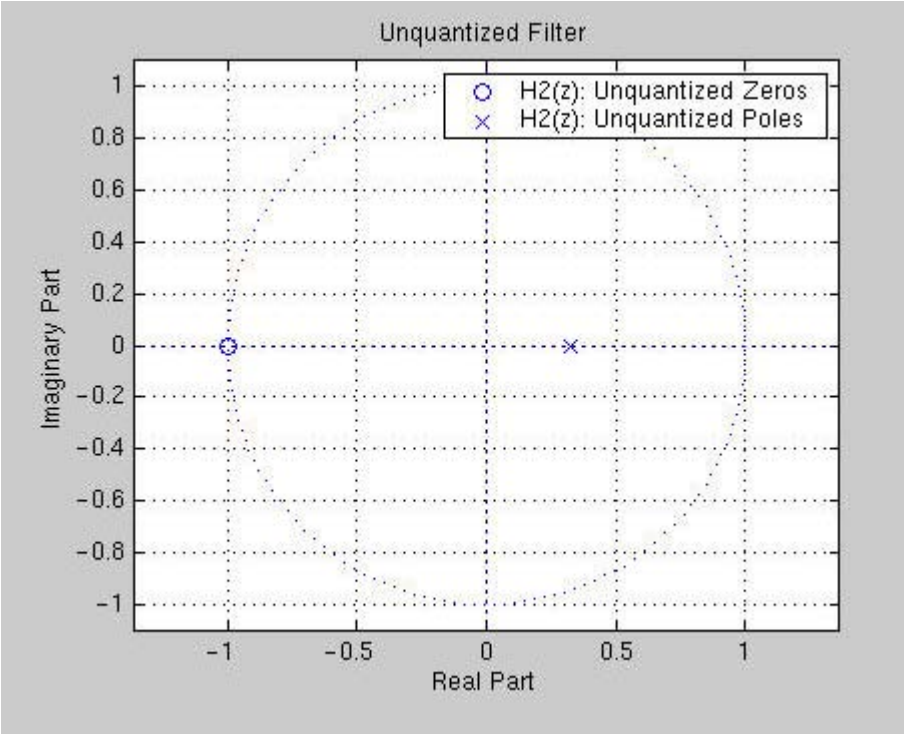
Pole/Zero Plot of Parallel Transfer Function Section $H_1(z)$ of Equations 9.2.



Pole/Zero Plot of Parallel Transfer Function Section $H_2(z)$ of Equations 9.3.



Pole/Zero Plot of Cascade Transfer Function Section $H_1(z)$ of Equations 9.10.



Pole/Zero Plot of Cascade Transfer Function Section $H_2(z)$ of Equations 9.11.

Synthesis-ready VHDL Code of Parallel Structure

```
library IEEE ;
library BITTRUE_VHDLSNPS ;

use IEEE.std_logic_1164.all ;
use IEEE.std_logic_arith.all ;
use BITTRUE_VHDLSNPS.COSSAP_PACKAGE_SYNOPSYS.all ;
Use BITTRUE_VHDLSNPS.fxp_arith.all ;

entity LOW1_P1_STRUCT is
   port    ( DIG_IN : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             DIG_OUT : out STD_LOGIC_VECTOR((16-1) downto 0) ;
             CLK, RESET : in STD_LOGIC ) ;
end LOW1_P1_STRUCT ;

architecture STRUCTURE of LOW1_P1_STRUCT is
component ADDER2_16
   port    ( IN0 : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             IN1 : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             OUTPUT : out STD_LOGIC_VECTOR((16-1) downto 0) ) ;
end component;
component ADDER3_16
   port    ( IN0 : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             IN1 : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             IN2 : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             OUTPUT : out STD_LOGIC_VECTOR((16-1) downto 0) ) ;
end component;
component MULT_B0_H1
   port    ( IN0 : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             OUTPUT : out STD_LOGIC_VECTOR((16-1) downto 0) ) ;
end component;
component MULT_B1_H1
   port    ( IN0 : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             OUTPUT : out STD_LOGIC_VECTOR((16-1) downto 0) ) ;
end component;
component MULT_A1_H1
   port    ( IN0 : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             OUTPUT : out STD_LOGIC_VECTOR((16-1) downto 0) ) ;
end component;
component MULT_A2_H1
   port    ( IN0 : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             OUTPUT : out STD_LOGIC_VECTOR((16-1) downto 0) ) ;
end component;
component MULT_B0_H2
   port    ( IN0 : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             OUTPUT : out STD_LOGIC_VECTOR((16-1) downto 0) ) ;
end component;
component MULT_B1_H2
   port    ( IN0 : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             OUTPUT : out STD_LOGIC_VECTOR((16-1) downto 0) ) ;
end component;
component MULT_A1_H2
   port    ( IN0 : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             OUTPUT : out STD_LOGIC_VECTOR((16-1) downto 0) ) ;
end component;
```

```
component DELAY_16
   port    ( INPUT : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             OUTPUT : out STD_LOGIC_VECTOR((16-1) downto 0 ) ;
             CLK, RESET : in STD_LOGIC ) ;
end component;

signal DLY_IN_0_H1, DLY_OUT_0_H1: STD_LOGIC_VECTOR((16-1) downto 0) ;
signal DLY_OUT_1_H1           : STD_LOGIC_VECTOR((16-1) downto 0) ;
signal DLY_IN_0_H2, DLY_OUT_0_H2: STD_LOGIC_VECTOR((16-1) downto 0) ;

signal DIG_OUT_INT            : STD_LOGIC_VECTOR((16-1) downto 0) ;

signal B0_OUT_H1, B1_OUT_H1   : STD_LOGIC_VECTOR((16-1) downto 0) ;
signal DIG_OUT_H1, A1_OUT_H1  : STD_LOGIC_VECTOR((16-1) downto 0) ;
signal A2_OUT_H1        : STD_LOGIC_VECTOR((16-1) downto 0) ;

signal B0_OUT_H2, B1_OUT_H2   : STD_LOGIC_VECTOR((16-1) downto 0) ;
signal DIG_OUT_H2, A1_OUT_H2  : STD_LOGIC_VECTOR((16-1) downto 0) ;

begin
      DIG_OUTPUT2: DELAY_16
                  port map ( DIG_OUT_INT, DIG_OUT, CLK, RESET );

      DIG_OUTPUT: ADDER2_16 port map ( DIG_OUT_H1, DIG_OUT_H2, DIG_OUT_INT );

      DELAY_16_DELAY_0_H1: DELAY_16
                  port map ( DLY_IN_0_H1, DLY_OUT_0_H1, CLK, RESET );
      DELAY_16_DELAY_1_H1: DELAY_16
                  port map ( A2_OUT_H1, DLY_OUT_1_H1, CLK, RESET );
      DELAY_16_DELAY_0_H2: DELAY_16
                  port map ( DLY_IN_0_H2, DLY_OUT_0_H2, CLK, RESET );

      B0_H1: MULT_B0_H1 port map ( DIG_IN, B0_OUT_H1 );
      B1_H1: MULT_B1_H1 port map ( DIG_IN, B1_OUT_H1 );
      A1_H1: MULT_A1_H1 port map ( DIG_OUT_H1, A1_OUT_H1 );
      A2_H1: MULT_A2_H1 port map ( DIG_OUT_H1, A2_OUT_H1 );

      B0_H2: MULT_B0_H2 port map ( DIG_IN, B0_OUT_H2 );
      B1_H2: MULT_B1_H2 port map ( DIG_IN, B1_OUT_H2 );
      A1_H2: MULT_A1_H2 port map ( DIG_OUT_H2, A1_OUT_H2 );

      ADDER_0_H1: ADDER2_16 port map ( B0_OUT_H1, DLY_OUT_0_H1, DIG_OUT_H1 );
      ADDER_1_H1: ADDER3_16
            port map ( B1_OUT_H1, DLY_OUT_1_H1, A1_OUT_H1, DLY_IN_0_H1 );
      ADDER_0_H2: ADDER2_16 port map ( B0_OUT_H2, DLY_OUT_0_H2, DIG_OUT_H2 );
      ADDER_1_H2: ADDER2_16 port map ( B1_OUT_H2, A1_OUT_H2, DLY_IN_0_H2 );
end STRUCTURE ;
```

Synthesis-ready Top-level VHDL Code with A/D Converter (Parallel Structure)

```
library IEEE ;
library BITTRUE_VHDLSNPS ;

use IEEE.std_logic_1164.all ;
use IEEE.std_logic_arith.all ;
use BITTRUE_VHDLSNPS.COSSAP_PACKAGE_SYNOPSYS.all ;
Use BITTRUE_VHDLSNPS.fxp_arith.all ;

entity LOW1_P1_TOP is
   generic ( BIN_LENGTH: in INTEGER);
   port    ( DIG_IN : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             DIG_OUT : out STD_LOGIC_VECTOR((16-1) downto 0) ;
             CLK, RESET : in STD_LOGIC ) ;
end LOW1_P1_TOP ;

architecture STRUCTURE of LOW1_P1_TOP is
component LOW1_P1_STRUCT
   port    ( DIG_IN : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             DIG_OUT : out STD_LOGIC_VECTOR((16-1) downto 0) ;
             CLK, RESET : in STD_LOGIC ) ;
end component;
component AD_CONVRTR
   generic ( BIN_LENGTH: in INTEGER);
   port    ( INPUT : in STD_LOGIC_VECTOR((BIN_LENGTH-1) downto 0) ;
            OUTPUT : out STD_LOGIC_VECTOR((BIN_LENGTH-1) downto 0);
             CLK, RESET : in STD_LOGIC ) ;
end component;

signal DIG_INT: STD_LOGIC_VECTOR((BIN_LENGTH-1) downto 0) ;

begin

      AD_16BITS: AD_CONVRTR generic map ( BIN_LENGTH )
                        port map ( DIG_IN, DIG_INT, CLK, RESET );
      LOWPASS: LOW1_P1_STRUCT port map ( DIG_INT, DIG_OUT, CLK, RESET );


end STRUCTURE ;
```

Top-level Script File for Parallel Structure

```
read -f db ./db/DFF_16.db
read -f db ./db/ADDER2_16.db
read -f db ./db/ADDER3_16.db
read -f db ./db/MULT_B0_H1.db
read -f db ./db/MULT_B1_H1.db
read -f db ./db/MULT_A1_H1.db
read -f db ./db/MULT_A2_H1.db
read -f db ./db/MULT_B0_H2.db
read -f db ./db/MULT_B1_H2.db
read -f db ./db/MULT_A1_H2.db

set_dont_touch DFF_16
```

```
set_dont_touch ADDER2_16
set_dont_touch ADDER3_16
set_dont_touch mult_b0_h1
set_dont_touch mult_b1_h1
set_dont_touch mult_a1_h1
set_dont_touch mult_a2_h1
set_dont_touch mult_b0_h2
set_dont_touch mult_b1_h2
set_dont_touch mult_a1_h2

analyze -f vhdl ../src/Butterworth16/Parallel/low1_p1_struct.vhd

elaborate LOW1_P1_STRUCT -arch "STRUCTURE" -lib BITTRUE_VHDLSNPS -update

include "./scripts/low1_p1_clk.scr"
include "./scripts/low1_p1_reset.scr"

set_input_delay  -clock CLK 10.0 all_inputs()
set_output_delay -clock CLK 10.0 all_outputs()

set_prefer { hcells/* }
set_dont_use { lsi_10k/* }

uniquify

compile -map_effort high

write -f db -hierarchy -out ./db/LOW1_P1_STRUCT.db
write -f vhdl          -out ./LOW1_P1_STRUCT.vhd

check_design                                  > ./low1_p1_check.rpt
report_timing                                 > ./low1_p1_timing.rpt
report_constraints -max_delay -all_violators -verbose > ./low1_p1_viol.rpt
report_cell                                   > ./low1_p1_cell.rpt
report_area                                   > ./low1_p1_area.rpt
report_cell all_registers()                   > ./low1_p1_registers.rpt
report_net                                    > ./low1_p1_net.rpt
report_clock                                  > ./low1_p1_clock.rpt
report_timing -path end -delay max -max_paths 600 -nworst 1 -to all_outputs()
>> ./low1_p1_timing.rpt

quit
```

Timing Report of Parallel Structure

```
Information: Updating design information... (UID-85)

****************************************
Report : timing
        -path short
        -delay max
        -max_paths 1
Design : LOW1_P1_STRUCT
Version: 1999.10
Date   : Thu Nov 25 21:32:51 1999
```

```
****************************************

Operating Conditions:
Wire Load Model Mode: top

  Startpoint: DIG_IN<14> (input port clocked by CLK)
  Endpoint: DFF_16_DELAY_0_H2/OUTPUT_reg<0>
            (rising edge-triggered flip-flop clocked by CLK)
  Path Group: CLK
  Path Type: max

  Point                                                    Incr       Path
  -----------------------------------------------------------------------------
  clock CLK (rise edge)                                    0.00       0.00
  clock network delay (ideal)                              0.00       0.00
  input external delay                                    10.00      10.00 f
  DIG_IN<14> (in)                                          0.00      10.00 f
  ...
  DFF_16_DELAY_0_H2/OUTPUT_reg<0>/d0 (hdrpq)              42.42      52.42 f
  data arrival time                                                  52.42

  clock CLK (rise edge)                                 100000.00  100000.00
  clock network delay (ideal)                              0.00   100000.00
  clock uncertainty                                      -10.00   99990.00
  DFF_16_DELAY_0_H2/OUTPUT_reg<0>/ck (hdrpq)               0.00   99990.00 r
  library setup time                                      -0.45   99989.55
  data required time                                              99989.55
  -----------------------------------------------------------------------------
  data required time                                              99989.55
  data arrival time                                                 -52.42
  -----------------------------------------------------------------------------
  slack (MET)                                                     99937.12


Performing report_timing on port 'DIG_OUT<15>'.
Performing report_timing on port 'DIG_OUT<14>'.
Performing report_timing on port 'DIG_OUT<13>'.
Performing report_timing on port 'DIG_OUT<12>'.
Performing report_timing on port 'DIG_OUT<11>'.
Performing report_timing on port 'DIG_OUT<10>'.
Performing report_timing on port 'DIG_OUT<9>'.
Performing report_timing on port 'DIG_OUT<8>'.
Performing report_timing on port 'DIG_OUT<7>'.
Performing report_timing on port 'DIG_OUT<6>'.
Performing report_timing on port 'DIG_OUT<5>'.
Performing report_timing on port 'DIG_OUT<4>'.
Performing report_timing on port 'DIG_OUT<3>'.
Performing report_timing on port 'DIG_OUT<2>'.
Performing report_timing on port 'DIG_OUT<1>'.
Performing report_timing on port 'DIG_OUT<0>'.

****************************************
Report : timing
        -path end
        -delay max
        -max_paths 600
Design : LOW1_P1_STRUCT
```

```
Version: 1999.10
Date   : Thu Nov 25 21:32:53 1999
****************************************

Operating Conditions:
Wire Load Model Mode: top

Endpoint                          Path Delay     Path Required     Slack
-------------------------------------------------------------------------
DIG_OUT<0> (out)                     0.42 f        99980.00     99979.58
DIG_OUT<1> (out)                     0.42 f        99980.00     99979.58
DIG_OUT<2> (out)                     0.42 f        99980.00     99979.58
DIG_OUT<3> (out)                     0.42 f        99980.00     99979.58
DIG_OUT<4> (out)                     0.42 f        99980.00     99979.58
DIG_OUT<5> (out)                     0.42 f        99980.00     99979.58
DIG_OUT<6> (out)                     0.42 f        99980.00     99979.58
DIG_OUT<7> (out)                     0.42 f        99980.00     99979.58
DIG_OUT<8> (out)                     0.42 f        99980.00     99979.58
DIG_OUT<9> (out)                     0.42 f        99980.00     99979.58
DIG_OUT<10> (out)                    0.42 f        99980.00     99979.58
DIG_OUT<11> (out)                    0.42 f        99980.00     99979.58
DIG_OUT<12> (out)                    0.42 f        99980.00     99979.58
DIG_OUT<13> (out)                    0.42 f        99980.00     99979.58
DIG_OUT<14> (out)                    0.42 f        99980.00     99979.58
DIG_OUT<15> (out)                    0.42 f        99980.00     99979.58
```

Synthesis-ready VHDL Code of Cascade Structure

```
library IEEE ;
library BITTRUE_VHDLSNPS ;

use IEEE.std_logic_1164.all ;
use IEEE.std_logic_arith.all ;
use BITTRUE_VHDLSNPS.COSSAP_PACKAGE_SYNOPSYS.all ;
Use BITTRUE_VHDLSNPS.fxp_arith.all ;

entity LOW1_C1_STRUCT is
   port    ( DIG_IN : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             DIG_OUT : out STD_LOGIC_VECTOR((16-1) downto 0) ;
             CLK, RESET : in STD_LOGIC ) ;
end LOW1_C1_STRUCT ;

architecture STRUCTURE of LOW1_C1_STRUCT is
component ADDER2_16
   port    ( IN0 : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             IN1 : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             OUTPUT : out STD_LOGIC_VECTOR((16-1) downto 0) ) ;
end component;
component ADDER3_16
   port    ( IN0 : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             IN1 : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             IN2 : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             OUTPUT : out STD_LOGIC_VECTOR((16-1) downto 0) ) ;
end component;
component MULT_B0_H1
```

```
   port    ( IN0 : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             OUTPUT : out STD_LOGIC_VECTOR((16-1) downto 0) ) ;
end component;
component MULT_B1_H1
   port    ( IN0 : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             OUTPUT : out STD_LOGIC_VECTOR((16-1) downto 0) ) ;
end component;
component MULT_B2_H1
   port    ( IN0 : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             OUTPUT : out STD_LOGIC_VECTOR((16-1) downto 0) ) ;
end component;
component MULT_A1_H1
   port    ( IN0 : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             OUTPUT : out STD_LOGIC_VECTOR((16-1) downto 0) ) ;
end component;
component MULT_A2_H1
   port    ( IN0 : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             OUTPUT : out STD_LOGIC_VECTOR((16-1) downto 0) ) ;
end component;
component MULT_B0_H2
   port    ( IN0 : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             OUTPUT : out STD_LOGIC_VECTOR((16-1) downto 0) ) ;
end component;
component MULT_B1_H2
   port    ( IN0 : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             OUTPUT : out STD_LOGIC_VECTOR((16-1) downto 0) ) ;
end component;
component MULT_A1_H2
   port    ( IN0 : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             OUTPUT : out STD_LOGIC_VECTOR((16-1) downto 0) ) ;
end component;
component DELAY_16
   port    ( INPUT : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             OUTPUT : out STD_LOGIC_VECTOR((16-1) downto 0 ) ;
             CLK, RESET : in STD_LOGIC ) ;
end component;

signal B0_OUT_H1, B1_OUT_H1, B2_OUT_H1    : STD_LOGIC_VECTOR((16-1) downto 0)
;
signal A1_OUT_H1, A2_OUT_H1         : STD_LOGIC_VECTOR((16-1) downto 0) ;

signal B0_OUT_H2, B1_OUT_H2         : STD_LOGIC_VECTOR((16-1) downto 0) ;
signal A1_OUT_H2              : STD_LOGIC_VECTOR((16-1) downto 0) ;

signal DLY_IN_0_H1, DLY_OUT_0_H1    : STD_LOGIC_VECTOR((16-1) downto 0) ;
signal DLY_IN_1_H1, DLY_OUT_1_H1    : STD_LOGIC_VECTOR((16-1) downto 0) ;

signal DLY_IN_0_H2, DLY_OUT_0_H2    : STD_LOGIC_VECTOR((16-1) downto 0) ;

signal DIG_OUT_H1             : STD_LOGIC_VECTOR((16-1) downto 0) ;

signal DIG_OUT_INT                  : STD_LOGIC_VECTOR((16-1) downto 0) ;
signal DIG_OUT_H1_INT               : STD_LOGIC_VECTOR((16-1) downto 0) ;

begin

      B0_H1: MULT_B0_H1 port map ( DIG_IN, B0_OUT_H1 );
```

```
      B1_H1: MULT_B1_H1 port map ( DIG_IN, B1_OUT_H1 );
      B2_H1: MULT_B2_H1 port map ( DIG_IN, B2_OUT_H1 );
      A1_H1: MULT_A1_H1 port map ( DIG_OUT_H1, A1_OUT_H1 );
      A2_H1: MULT_A2_H1 port map ( DIG_OUT_H1, A2_OUT_H1 );

      B0_H2: MULT_B0_H2 port map ( DIG_OUT_H1_INT, B0_OUT_H2 );
      B1_H2: MULT_B1_H2 port map ( DIG_OUT_H1_INT, B1_OUT_H2 );
      A1_H2: MULT_A1_H2 port map ( DIG_OUT_INT, A1_OUT_H2 );

      ADDER_0_H1: ADDER2_16 port map ( B0_OUT_H1, DLY_OUT_0_H1, DIG_OUT_H1 );
      ADDER_1_H1: ADDER3_16
                port map ( B1_OUT_H1, DLY_OUT_1_H1, A1_OUT_H1, DLY_IN_0_H1);
      ADDER_2_H1: ADDER2_16 port map ( B2_OUT_H1, A2_OUT_H1, DLY_IN_1_H1 );

      ADDER_0_H2: ADDER2_16 port map ( B0_OUT_H2, DLY_OUT_0_H2, DIG_OUT_INT
);
      ADDER_1_H2: ADDER2_16 port map ( B1_OUT_H2, A1_OUT_H2, DLY_IN_0_H2 );

      DFF_DELAY_0_H1: DELAY_16
                  port map ( DLY_IN_0_H1, DLY_OUT_0_H1, CLK, RESET );
      DFF_DELAY_1_H1: DELAY_16
                  port map ( DLY_IN_1_H1, DLY_OUT_1_H1, CLK, RESET );

      DFF_DELAY_0_H2: DELAY_16
                  port map ( DLY_IN_0_H2, DLY_OUT_0_H2, CLK, RESET );

      STAGE_1: DELAY_16 port map ( DIG_OUT_H1, DIG_OUT_H1_INT, CLK, RESET );
      STAGE_2: DELAY_16 port map ( DIG_OUT_INT, DIG_OUT, CLK, RESET );

end STRUCTURE ;
```

Synthesis-ready Top-level VHDL Code with A/D Converter (Cascade Structure)

```
library IEEE ;
library BITTRUE_VHDLSNPS ;

use IEEE.std_logic_1164.all ;
use IEEE.std_logic_arith.all ;
use BITTRUE_VHDLSNPS.COSSAP_PACKAGE_SYNOPSYS.all ;
Use BITTRUE_VHDLSNPS.fxp_arith.all ;

entity LOW1_C1_TOP is
   generic ( BIN_LENGTH: in INTEGER);
   port    ( DIG_IN : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             DIG_OUT : out STD_LOGIC_VECTOR((16-1) downto 0) ;
             CLK, RESET : in STD_LOGIC ) ;
end LOW1_C1_TOP ;

architecture STRUCTURE of LOW1_C1_TOP is
component LOW1_C1_STRUCT
   port    ( DIG_IN : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             DIG_OUT : out STD_LOGIC_VECTOR((16-1) downto 0) ;
             CLK, RESET : in STD_LOGIC ) ;
end component;
component AD_CONVRTR
```

```
   generic ( BIN_LENGTH: in INTEGER);
   port    ( INPUT : in STD_LOGIC_VECTOR((BIN_LENGTH-1) downto 0) ;
            OUTPUT : out STD_LOGIC_VECTOR((BIN_LENGTH-1) downto 0);
              CLK, RESET : in STD_LOGIC ) ;
end component;

signal DIG_INT: STD_LOGIC_VECTOR((BIN_LENGTH-1) downto 0) ;

begin

      AD_16BITS: AD_CONVRTR generic map ( BIN_LENGTH )
                        port map ( DIG_IN, DIG_INT, CLK, RESET );
      LOWPASS: LOW1_C1_STRUCT port map ( DIG_INT, DIG_OUT, CLK, RESET );


end STRUCTURE ;
```

Top-level Script File for Cascade Structure

```
read -f db ./db/DFF_16.db
read -f db ./db/ADDER2_16.db
read -f db ./db/ADDER3_16.db
read -f db ./db/MULT_B0_H1.db
read -f db ./db/MULT_B1_H1.db
read -f db ./db/MULT_B2_H1.db
read -f db ./db/MULT_A1_H1.db
read -f db ./db/MULT_A2_H1.db
read -f db ./db/MULT_B0_H2.db
read -f db ./db/MULT_B1_H2.db
read -f db ./db/MULT_A1_H2.db

set_dont_touch DFF_16
set_dont_touch ADDER2_16
set_dont_touch ADDER3_16
set_dont_touch MULT_B0_H1
set_dont_touch MULT_B1_H1
set_dont_touch MULT_B2_H1
set_dont_touch MULT_A1_H1
set_dont_touch MULT_A2_H1
set_dont_touch MULT_B0_H2
set_dont_touch MULT_B1_H2
set_dont_touch MULT_A1_H2

analyze -f vhdl ../src/Butterworth16/Cascade/low1_c1_struct.vhd

elaborate LOW1_C1_STRUCT -arch "STRUCTURE" -lib BITTRUE_VHDLSNPS -update

include "./scripts/low1_clk.scr"
include "./scripts/low1_reset.scr"

set_input_delay  -clock CLK 10.0 all_inputs()
set_output_delay -clock CLK 10.0 all_outputs()
```

```
set_prefer { hcells/* }
set_dont_use { lsi_10k/* }

uniquify

compile -map_effort high

write -f db -hierarchy -out ./db/LOW1_C1_STRUCT.db
write -f vhdl          -out ./LOW1_C1_STRUCT.vhd

check_design                                  > ./low1_c1_check.rpt
report_timing                                 > ./low1_c1_timing.rpt
report_constraints -max_delay -all_violators -verbose > ./low1_c1_viol.rpt
report_cell                                   > ./low1_c1_cell.rpt
report_area                                   > ./low1_c1_area.rpt
report_cell all_registers()                   > ./low1_c1_registers.rpt
report_net                                    > ./low1_c1_net.rpt
report_clock                                  > ./low1_c1_clock.rpt
report_timing -path end -delay max -max_paths 600 -nworst 1 -to all_outputs()
>> ./low1_c1_timing.rpt

quit
```

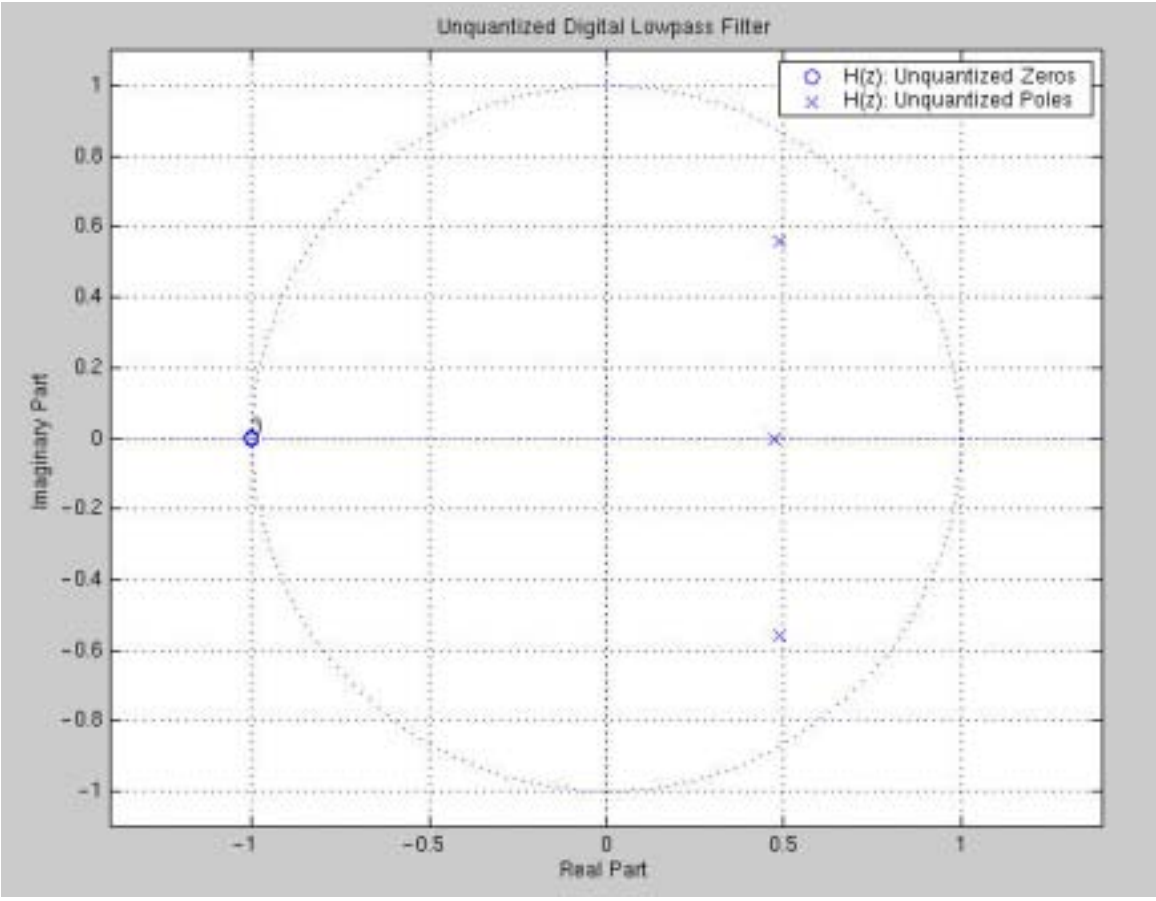
Timing Report of Cascade Structure

```
Information: Updating design information... (UID-85)

****************************************
Report : timing
        -path short
        -delay max
        -max_paths 1
Design : LOW1_C1_STRUCT
Version: 1999.10
Date   : Thu Nov 25 23:06:51 1999
****************************************

Operating Conditions:
Wire Load Model Mode: top

  Startpoint: DIG_IN<13> (input port clocked by CLK)
  Endpoint: DFF_DELAY_0_H1/OUTPUT_reg<0>
            (rising edge-triggered flip-flop clocked by CLK)
  Path Group: CLK
  Path Type: max

  Point                                                   Incr       Path
  -----------------------------------------------------------------------
  clock CLK (rise edge)                                   0.00       0.00
  clock network delay (ideal)                             0.00       0.00
  input external delay                                   10.00      10.00 f
  DIG_IN<13> (in)                                         0.00      10.00 f
  ...
  DFF_DELAY_0_H1/OUTPUT_reg<0>/d0 (hdrpq)                42.47      52.47 r
  data arrival time                                                 52.47
```

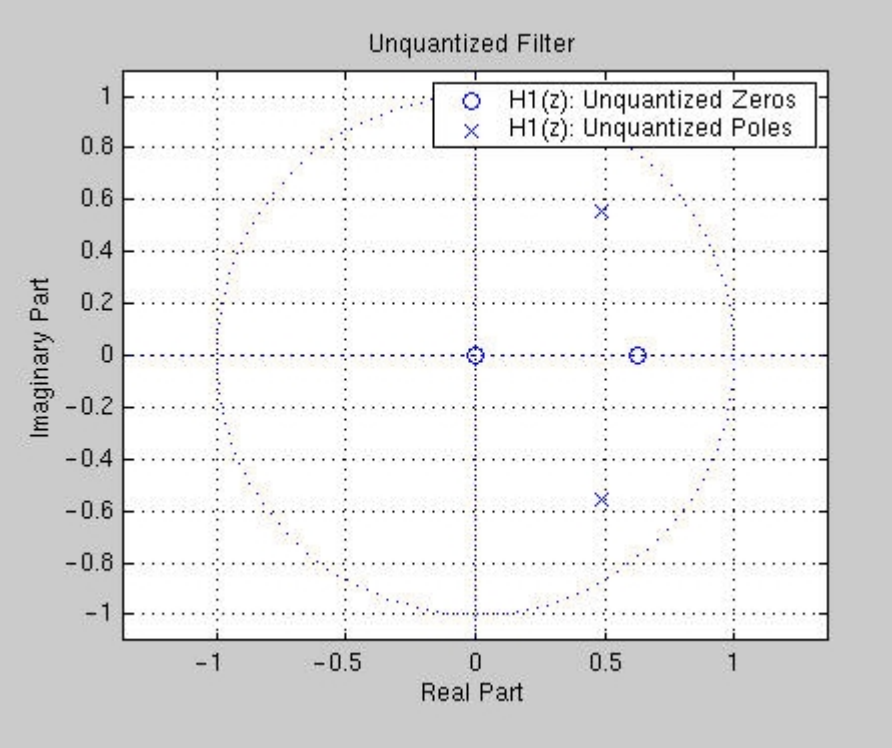
```
  clock CLK (rise edge)                                  100000.00  100000.00
  clock network delay (ideal)                                0.00   100000.00
  clock uncertainty                                        -10.00   99990.00
  DFF_DELAY_0_H1/OUTPUT_reg<0>/ck (hdrpq)                    0.00   99990.00 r
  library setup time                                        -0.45   99989.55
  data required time                                                99989.55
  ---------------------------------------------------------------------------
  data required time                                                99989.55
  data arrival time                                                   -52.47
  ---------------------------------------------------------------------------
  slack (MET)                                                       99937.08


Performing report_timing on port 'DIG_OUT<15>'.
Performing report_timing on port 'DIG_OUT<14>'.
Performing report_timing on port 'DIG_OUT<13>'.
Performing report_timing on port 'DIG_OUT<12>'.
Performing report_timing on port 'DIG_OUT<11>'.
Performing report_timing on port 'DIG_OUT<10>'.
Performing report_timing on port 'DIG_OUT<9>'.
Performing report_timing on port 'DIG_OUT<8>'.
Performing report_timing on port 'DIG_OUT<7>'.
Performing report_timing on port 'DIG_OUT<6>'.
Performing report_timing on port 'DIG_OUT<5>'.
Performing report_timing on port 'DIG_OUT<4>'.
Performing report_timing on port 'DIG_OUT<3>'.
Performing report_timing on port 'DIG_OUT<2>'.
Performing report_timing on port 'DIG_OUT<1>'.
Performing report_timing on port 'DIG_OUT<0>'.

****************************************
Report : timing
        -path end
        -delay max
        -max_paths 600
Design : LOW1_C1_STRUCT
Version: 1999.10
Date   : Thu Nov 25 23:06:54 1999
****************************************

Operating Conditions:
Wire Load Model Mode: top

Endpoint                          Path Delay     Path Required     Slack
---------------------------------------------------------------------------
DIG_OUT<0> (out)                     0.42 f        99980.00     99979.58
DIG_OUT<1> (out)                     0.42 f        99980.00     99979.58
DIG_OUT<2> (out)                     0.42 f        99980.00     99979.58
DIG_OUT<3> (out)                     0.42 f        99980.00     99979.58
DIG_OUT<4> (out)                     0.42 f        99980.00     99979.58
DIG_OUT<5> (out)                     0.42 f        99980.00     99979.58
DIG_OUT<6> (out)                     0.42 f        99980.00     99979.58
DIG_OUT<7> (out)                     0.42 f        99980.00     99979.58
DIG_OUT<8> (out)                     0.42 f        99980.00     99979.58
DIG_OUT<9> (out)                     0.42 f        99980.00     99979.58
DIG_OUT<10> (out)                    0.42 f        99980.00     99979.58
```

DIG_OUT<11> (out)	0.42 f	99980.00	99979.58
DIG_OUT<12> (out)	0.42 f	99980.00	99979.58
DIG_OUT<13> (out)	0.42 f	99980.00	99979.58
DIG_OUT<14> (out)	0.42 f	99980.00	99979.58
DIG_OUT<15> (out)	0.42 f	99980.00	99979.58

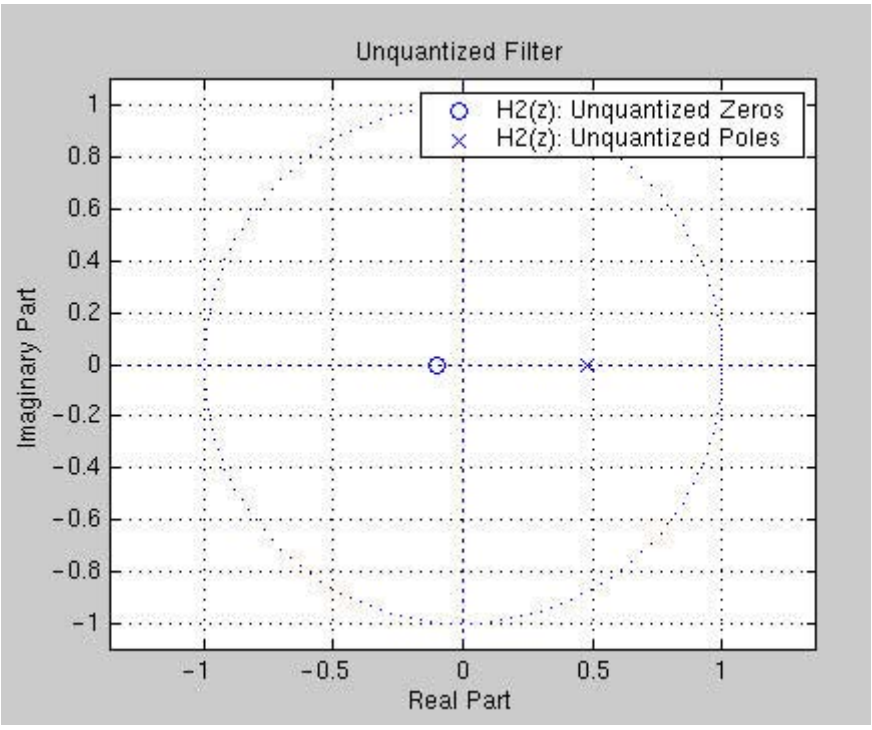
APPENDIX C: Pole/Zero Plots of 3rd order 16-bit Lowpass Chebyshev Type I Filter



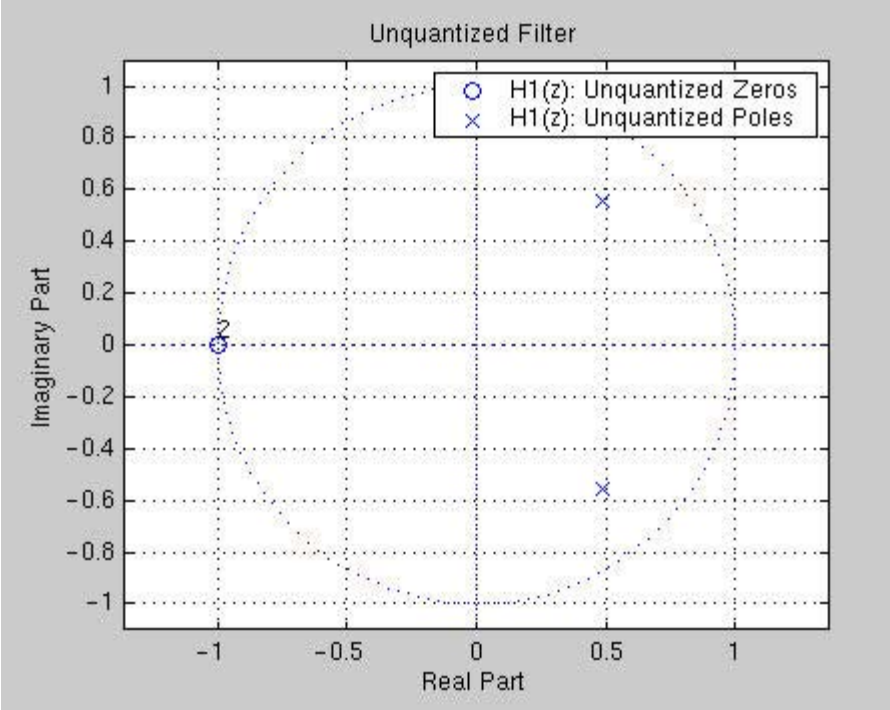
Pole/Zero Plot of 3rd Order Lowpass Chebyshev Type I Lowpass Filter.



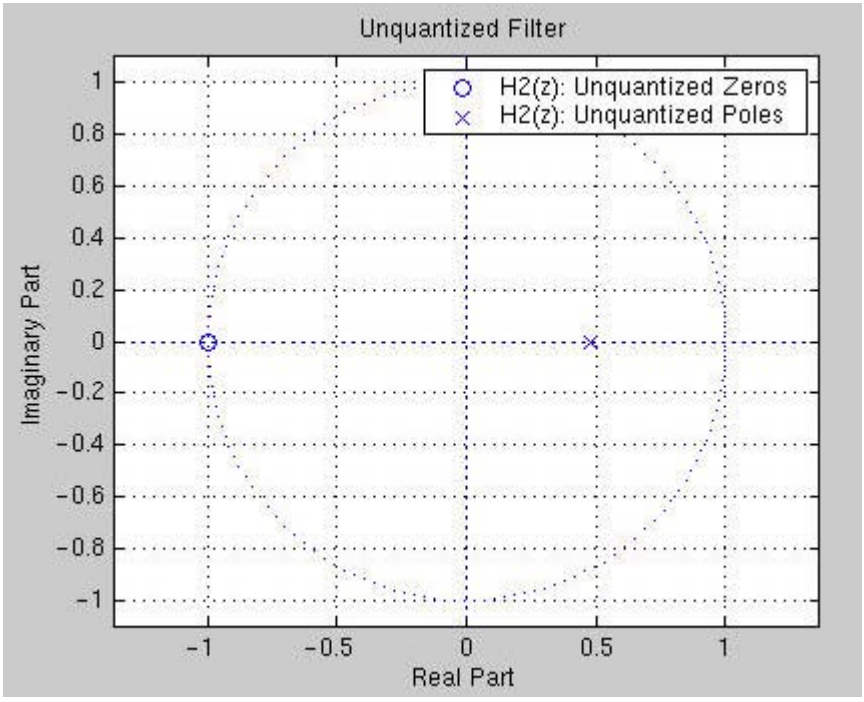
Pole/Zero Plot of Transfer Function $H_1(z)$ of Parallel Structure of Chebyshev Type I Lowpass Filter.



Pole/Zero Plot of Transfer Function $H_2(z)$ of Parallel Structure of Chebyshev Type I Lowpass Filter.

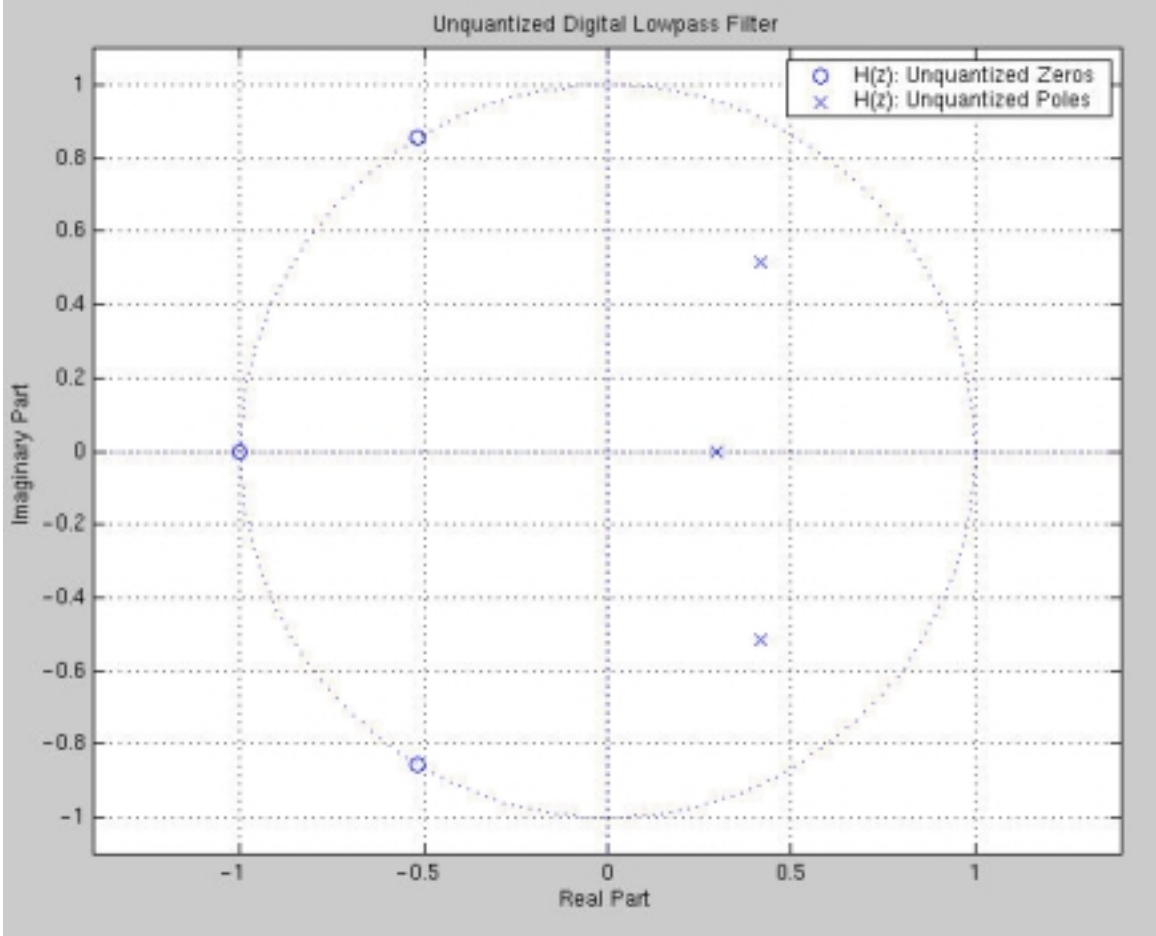


Pole/Zero Plot of Transfer Function $H_1(z)$ of Cascade Structure of Chebyshev Type I Lowpass Filter.

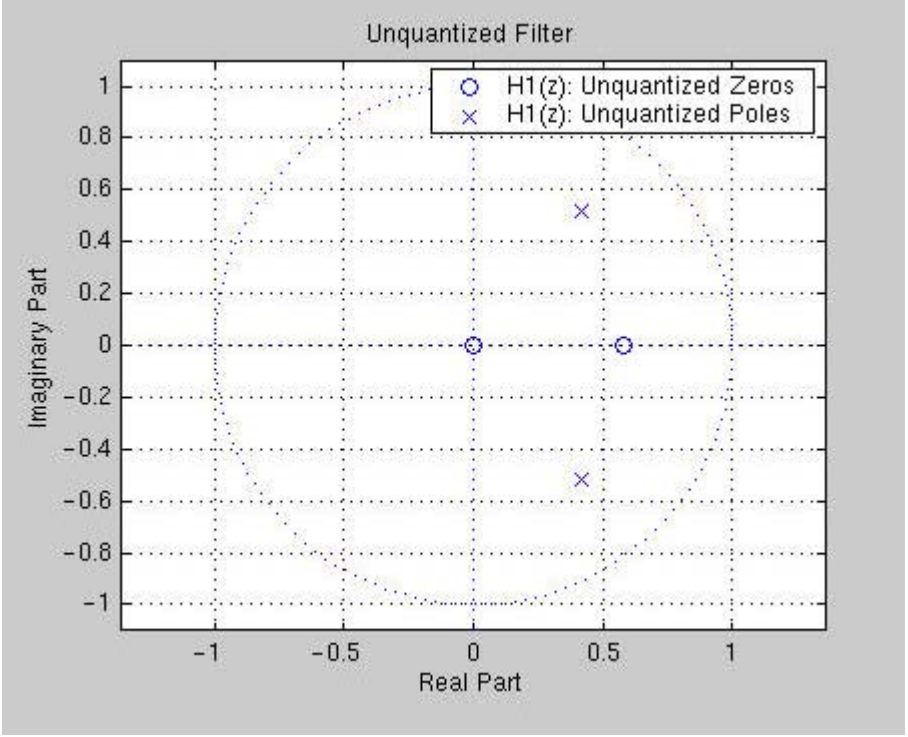


Pole/Zero Plot of Transfer Function $H_2(z)$ of Cascade Structure of Chebyshev Type I Lowpass Filter.

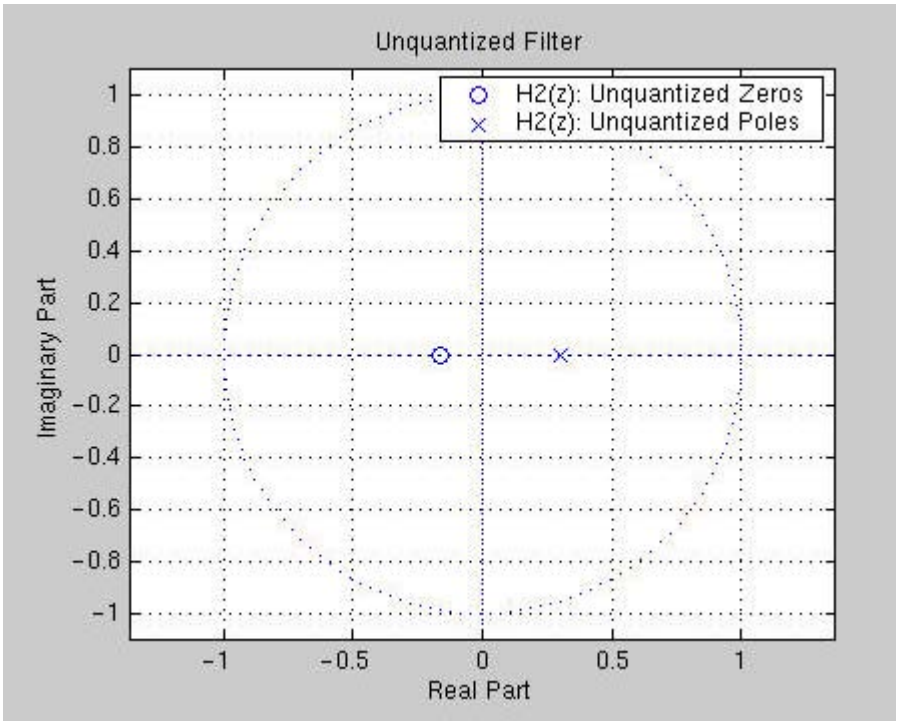
APPENDIX D: Pole/Zero Plots of 3rd order 16-bit Lowpass Chebyshev Type II Filter



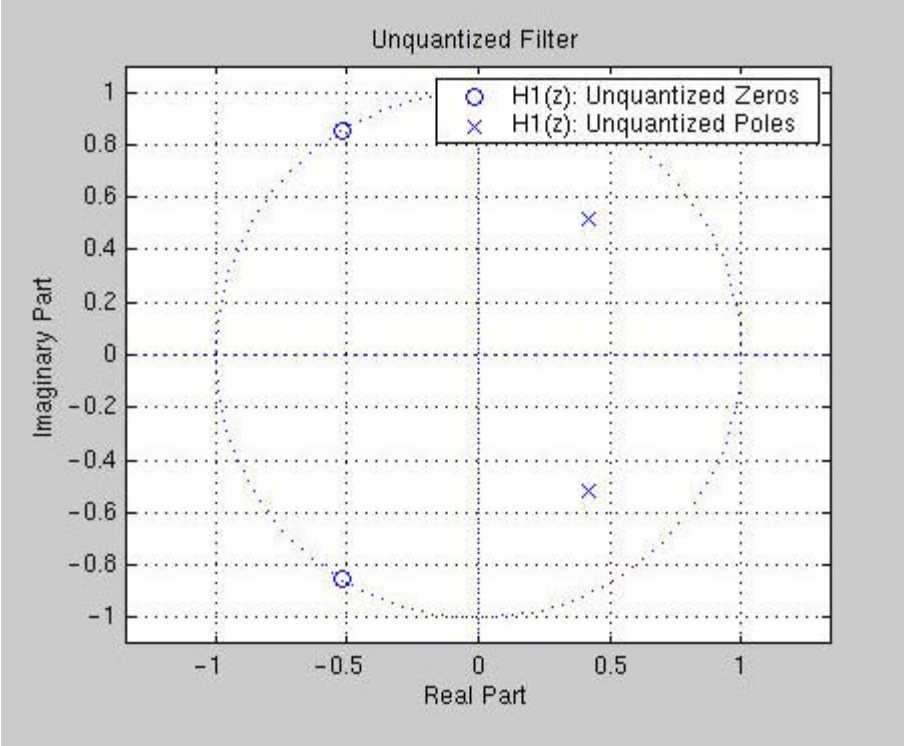
Pole/Zero Plot of 3rd Order Lowpass Chebyshev Type II Lowpass Filter.



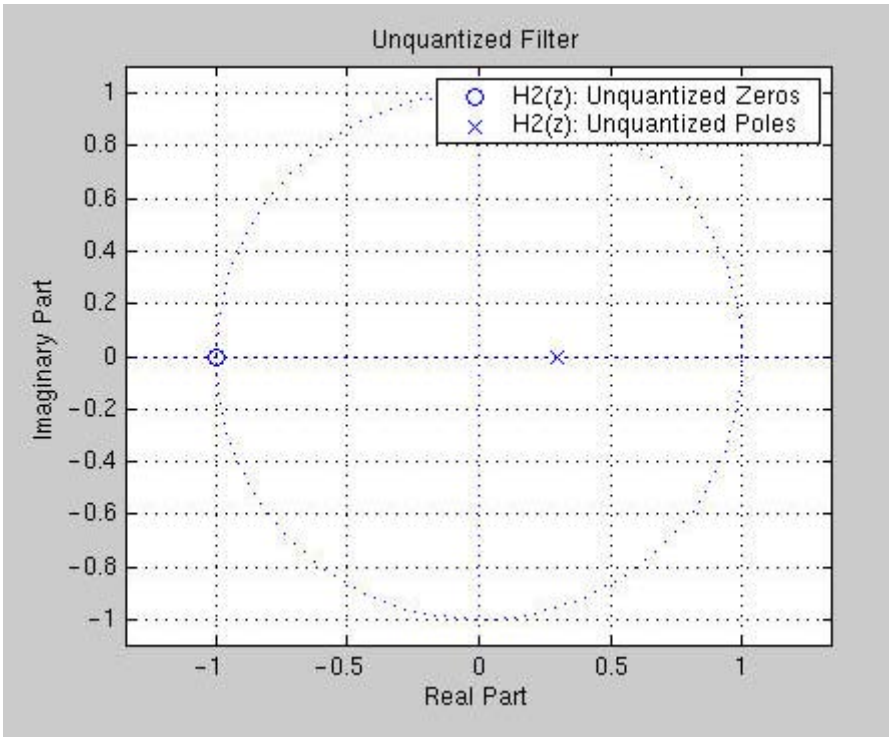
Pole/Zero Plot of Transfer Function $H_1(z)$ of Parallel Structure of Chebyshev Type II Lowpass Filter.



Pole/Zero Plot of Transfer Function $H_2(z)$ of Parallel Structure of Chebyshev Type II Lowpass Filter.

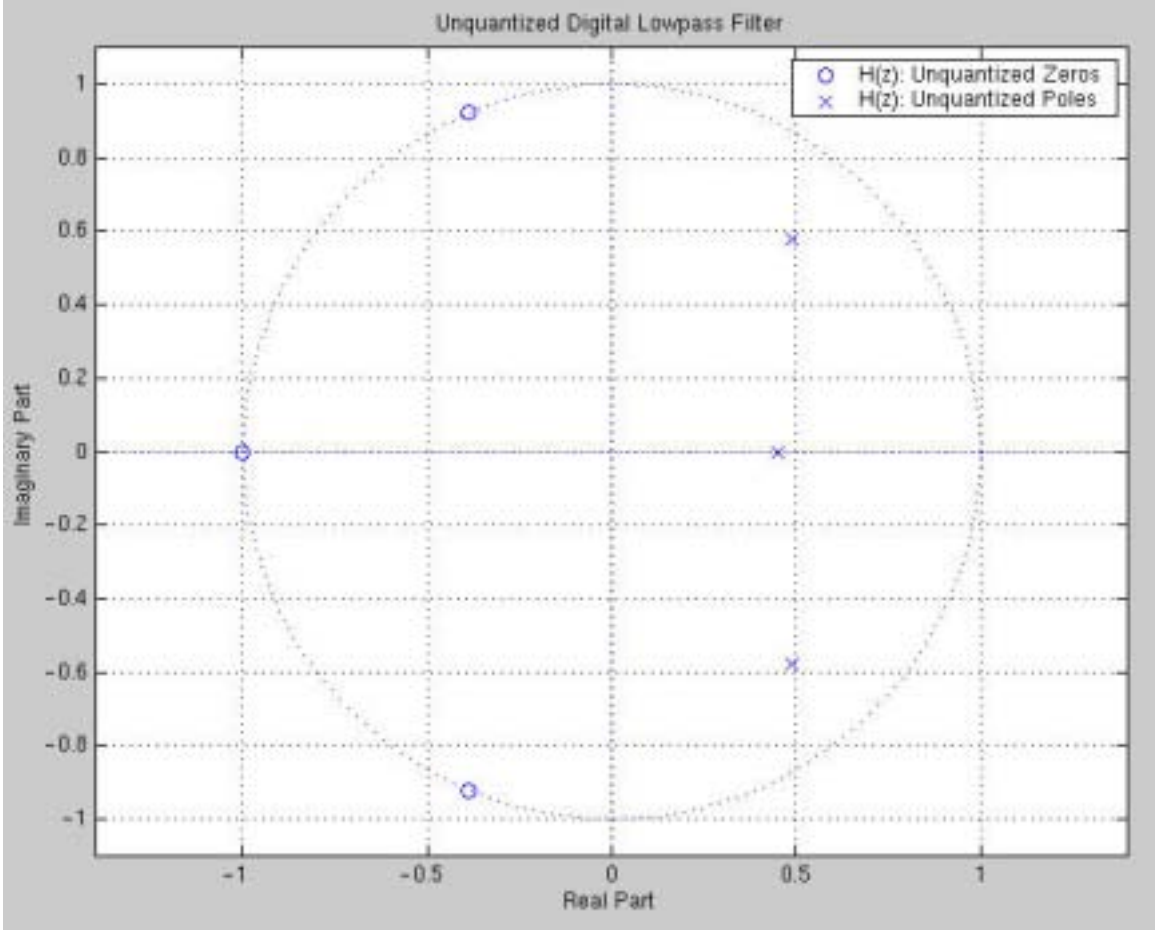


Pole/Zero Plot of Transfer Function $H_1(z)$ of Cascade Structure of Chebyshev Type II Lowpass Filter.

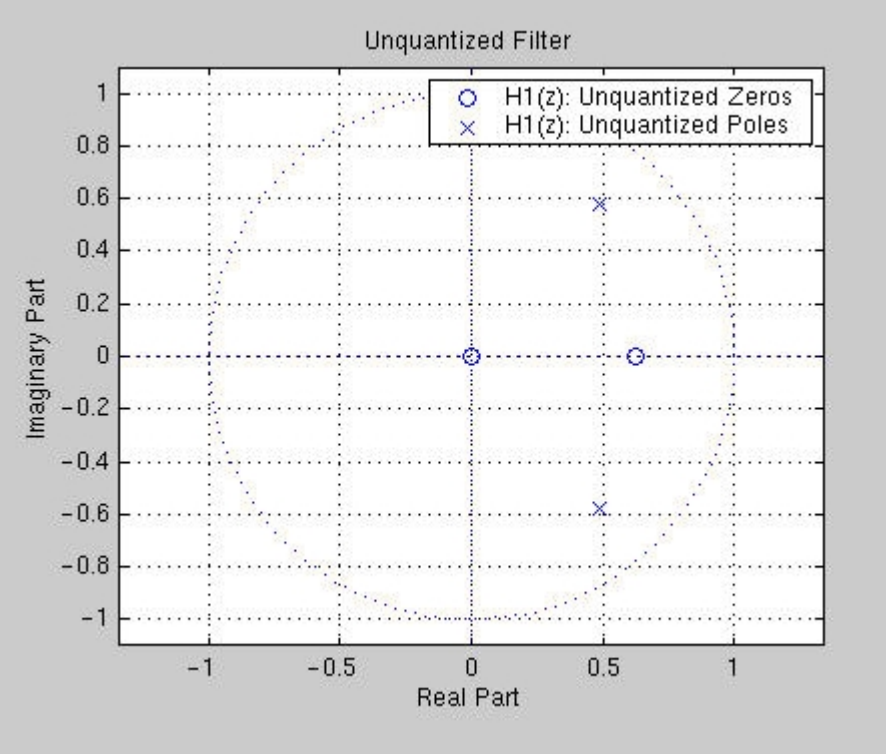


Pole/Zero Plot of Transfer Function $H_2(z)$ of Cascade Structure of Chebyshev Type II Lowpass Filter.

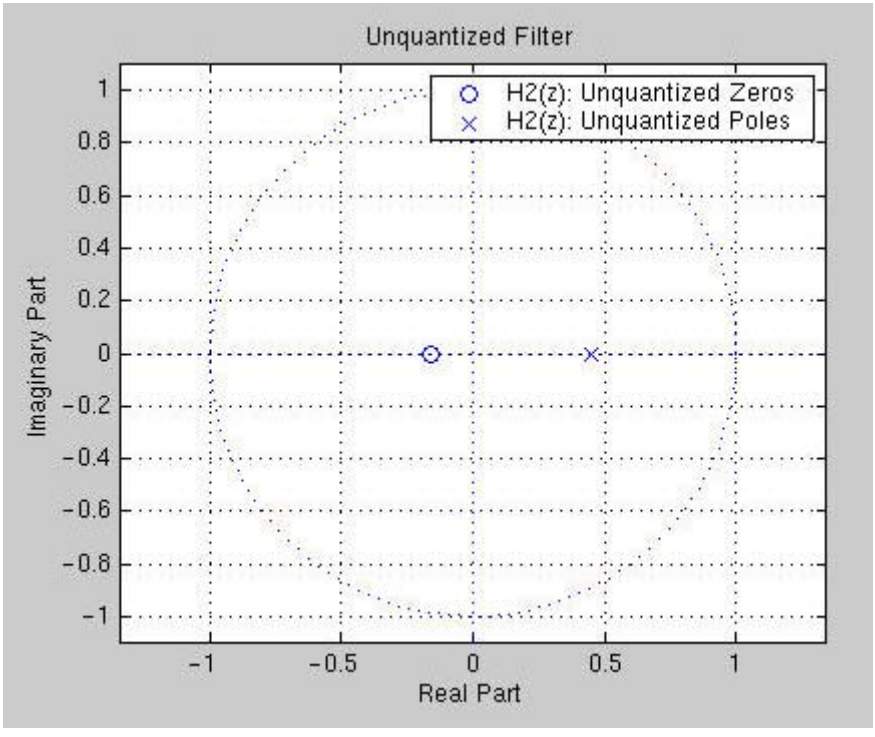
APPENDIX E: Pole/Zero Plots of 3rd order 16-bit Lowpass Elliptic Filter



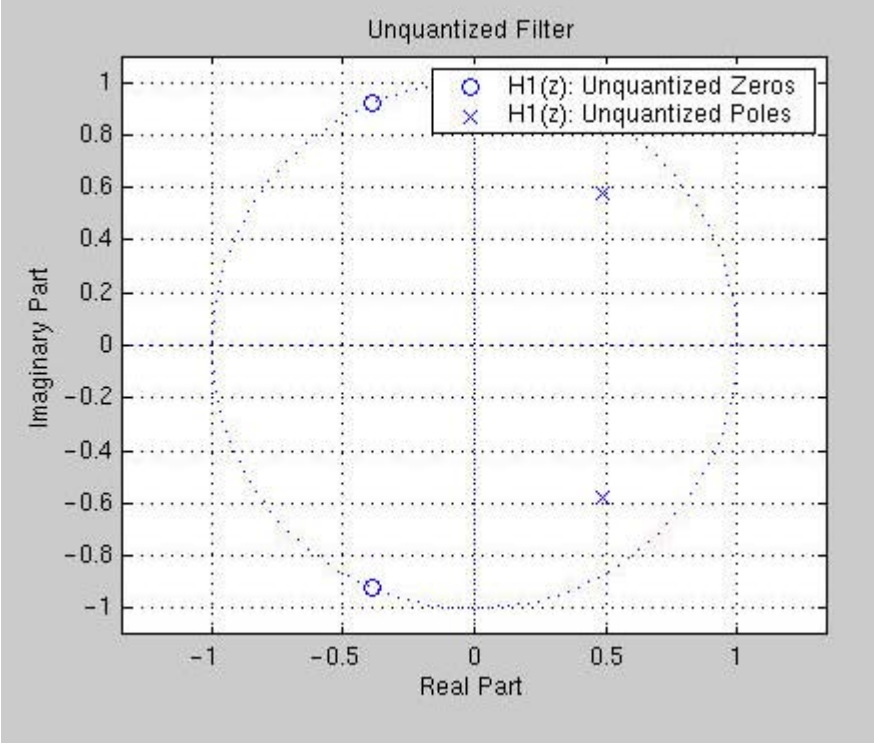
Pole/Zero Plot of 3rd Order Lowpass Elliptic Lowpass Filter.



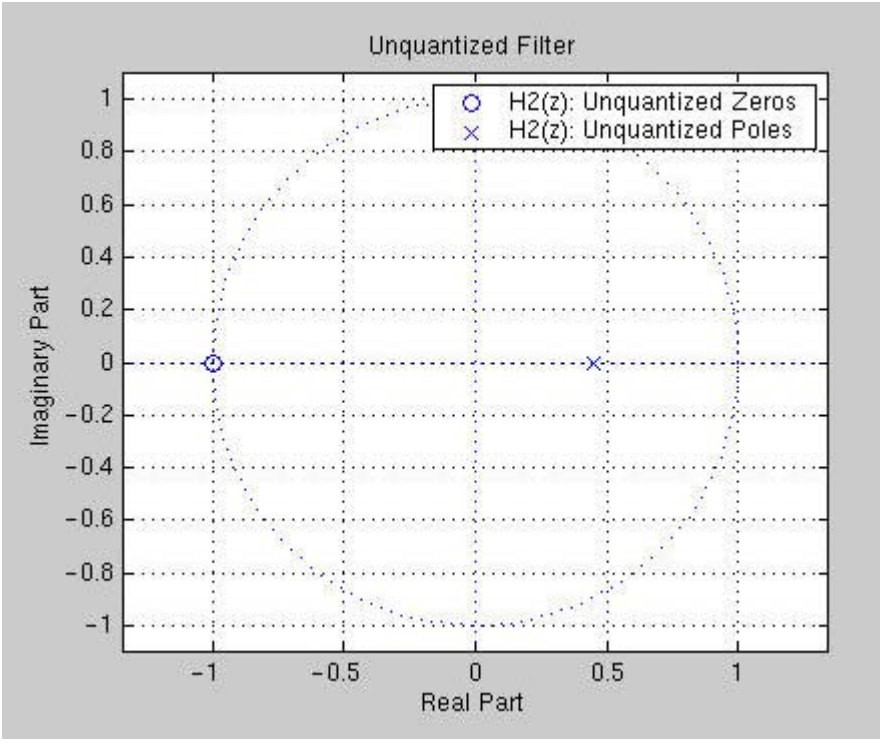
Pole/Zero Plot of Transfer Function $H_1(z)$ of Parallel Structure of Elliptic Lowpass Filter.



Pole/Zero Plot of Transfer Function $H_2(z)$ of Parallel Structure of Elliptic Lowpass Filter.



Pole/Zero Plot of Transfer Function $H_1(z)$ of Cascade Structure of Elliptic Lowpass Filter.



Pole/Zero Plot of Transfer Function $H_2(z)$ of Cascade Structure of Elliptic Lowpass Filter.

APPENDIX F: Synthesis-ready VHDL Library (8-bit and 16-bit examples)

Synthesis-ready VHDL Code of 16-bit 2-input Fixed-point Adder

```
library IEEE ;
library BITTRUE_VHDLSNPS ;

use IEEE.std_logic_1164.all ;
use IEEE.std_logic_arith.all ;
use BITTRUE_VHDLSNPS.COSSAP_PACKAGE_SYNOPSYS.all ;
Use BITTRUE_VHDLSNPS.fxp_arith.all ;

entity ADDER2_16 is

   port    ( IN0 : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             IN1 : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             OUTPUT : out STD_LOGIC_VECTOR((16-1) downto 0) ) ;

end ADDER2_16 ;

architecture behavior of ADDER2_16 is
 constant RoundWidth_M_M15_1_1: INTEGER:= 16 + 1 ;
begin
  main: process(IN0,IN1)
        variable RoundSum_M_M15_1_1: SIGNED(ROUNDWIDTH_M_M15_1_1-1 DOWNTO 0)
;
      begin
              RoundSum_M_M15_1_1 := fxp_round(SIGNED(IN0(IN0'high) & IN0) +
                        SIGNED(IN1), 0, RoundWidth_M_M15_1_1);
       OUTPUT <= STD_LOGIC_VECTOR(fxp_saturate(RoundSum_M_M15_1_1, 1, 16));
      end process main;
end behavior ;
```

Synthesis-ready VHDL Code of 16-bit 3-input Fixed-point Adder

```
library IEEE ;
library BITTRUE_VHDLSNPS ;

use IEEE.std_logic_1164.all ;
use IEEE.std_logic_arith.all ;
use BITTRUE_VHDLSNPS.COSSAP_PACKAGE_SYNOPSYS.all ;
Use BITTRUE_VHDLSNPS.fxp_arith.all ;

entity ADDER3_16 is

   port    ( IN0 : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             IN1 : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             IN2 : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             OUTPUT : out STD_LOGIC_VECTOR((16-1) downto 0) ) ;

end ADDER3_16 ;


architecture behavior of ADDER3_16 is
 constant RoundWidth_M_M15_1_1: INTEGER:= 16 + 1 ;
```

```
 constant RoundWidth_M_M16_1_2: INTEGER:= 16 + 1 ;
begin

  main: process(IN0,IN1,IN2)
        variable RoundSum_M_M15_1_1:  SIGNED(ROUNDWIDTH_M_M15_1_1-1 DOWNTO 0)
;
        variable RoundSum_M_M16_1_2:  SIGNED(ROUNDWIDTH_M_M16_1_2-1 DOWNTO 0)
;
        variable SIG_4M_M15_1_1 : STD_LOGIC_VECTOR((16-1) downto  0) ;
      begin
          RoundSum_M_M15_1_1 := fxp_round(SIGNED(IN0(IN0'high) & IN0) +
                        SIGNED(IN1), 0, RoundWidth_M_M15_1_1);
          SIG_4M_M15_1_1 := STD_LOGIC_VECTOR(fxp_saturate(RoundSum_M_M15_1_1,
1, 16));
          RoundSum_M_M16_1_2 :=
fxp_round(SIGNED(SIG_4M_M15_1_1(SIG_4M_M15_1_1'high) &
                        SIG_4M_M15_1_1) + SIGNED(IN2), 0,
RoundWidth_M_M16_1_2);
          OUTPUT <= STD_LOGIC_VECTOR(fxp_saturate(RoundSum_M_M16_1_2, 1,
16));
      end process main;

end behavior ;
```

Synthesis-ready VHDL Code of 8-bit Fixed-point Multiplier w/coefficient

This VHDL code contains the decimal coefficient 0.6093750. One bit is assigned for integer representation. The remaining 7 bits are assigned for fractional representation. The COSSAP representation for this fractional coefficient is 78. The resulting synthesis-ready code is as follows:

```
library IEEE ;
library BITTRUE_VHDLSNPS ;

use IEEE.std_logic_1164.all ;
use IEEE.std_logic_arith.all ;
use BITTRUE_VHDLSNPS.COSSAP_PACKAGE_SYNOPSYS.all ;
Use BITTRUE_VHDLSNPS.fxp_arith.all ;

entity MULT_SYN is

   port    ( IN0 : in STD_LOGIC_VECTOR((8-1) downto 0) ;
             OUTPUT : out STD_LOGIC_VECTOR((8-1) downto 0) ) ;

end MULT_SYN ;


architecture behavior of MULT_SYN is
 constant RoundProdWidth_M_M1_1_1: INTEGER := 8 ;
begin
  main: process(IN0)
        variable Input2_M_M1_1_1: SIGNED(8 - 1 DOWNTO 0) ;
        variable RoundProd_M_M1_1_1: SIGNED(ROUNDPRODWIDTH_M_M1_1_1-1 DOWNTO
0);
      begin
        Input2_M_M1_1_1 := const2fxp(0, 78, 1, 8);
```

```
        RoundProd_M_M1_1_1 := fxp_round(SIGNED(IN0) * Input2_M_M1_1_1,
                        0, RoundProdWidth_M_M1_1_1);
        OUTPUT <= STD_LOGIC_VECTOR(fxp_saturate(RoundProd_M_M1_1_1, 1, 8));
      end process main;
end behavior ;
```

Synthesis-ready VHDL Code of 16-bit Fixed-point Delay Sub-block (Active-high reset)

```
library IEEE ;
use IEEE.std_logic_1164.all ;

entity delay is

   port    ( INPUT : in STD_LOGIC_VECTOR((16-1) downto 0) ;
             OUTPUT : out STD_LOGIC_VECTOR((16-1) downto 0);
             CLK, RESET : in STD_LOGIC ) ;

end delay;

architecture behavior of delay is
begin

  main: process(CLK,RESET)
      begin
       if (RESET = '1') then
             for i in 0 to 16-1 loop
                  OUTPUT(i) <= '0';
           end loop;
       elsif (CLK'EVENT and CLK = '1') then
            OUTPUT <= INPUT;
       end if;
      end process main;

end behavior ;
```

APPENDIX G: :Synthesis Script Files for Sub-blocks (16-bit example)

Synthesis Script File for 16-bit 2-input Fixed-point Adder

```
analyze -format vhdl ../src/adder2.vhd

elaborate ADDER2 -arch "behavior" -lib BITTRUE_VHDLSNPS -update

set_prefer { hcells/* }
set_dont_use { lsi_10k/* }

set_max_delay 20.0 -from all_inputs() -to all_outputs()

compile -incremental_mapping -ungroup_all -map_effort high

write -f db   -out ./db/ADDER2.db
write -f vhdl -out ./ADDER2.vhd

check_design                                > ./adder2_check.rpt
report_timing                               > ./adder2_timing.rpt
report_constraints -all_violators -verbose > ./adder2_viol.rpt
report_cell                                 > ./adder2_cell.rpt
report_area                                 > ./report/adder2_area.rpt
report_cell all_registers()                 > ./adder2_registers.rpt
report_net                                  > ./adder2_net.rpt
report_clock                                > ./adder2_clock.rpt
report_timing -path end -delay max -max_paths 600 -nworst 1 -to all_outputs()
>> ./adder2_timing.rpt

quit
```

Synthesis Script File for 16-bit 3-input Fixed-point Adder

```
analyze -format vhdl ../src/adder3.vhd

elaborate ADDER3 -arch "behavior" -lib BITTRUE_VHDLSNPS -update

set_prefer { hcells/* }
set_dont_use { lsi_10k/* }

set_max_delay 20.0 -from all_inputs() -to all_outputs()

compile -ungroup_all -map_effort high

write -f db   -out ./db/ADDER3.db
write -f vhdl -out ./ADDER3.vhd

check_design                                > ./adder3_check.rpt
report_timing                               > ./adder3_timing.rpt
report_constraints -all_violators -verbose > ./adder3_viol.rpt
report_cell                                 > ./adder3_cell.rpt
report_area                                 > ./adder3_area.rpt
report_cell all_registers()                 > ./adder3_registers.rpt
report_net                                  > ./adder3_net.rpt
report_clock                                > ./adder3_clock.rpt
```

```
report_timing -path end -delay max -max_paths 600 -nworst 1 -to all_outputs()
>> ./adder3_timing.rpt

quit
```

Synthesis Script File for 16-bit Fixed-point Multiplier w/coefficient

```
analyze -f vhdl ../src/mult.vhd

elaborate MULT -arch "behavior" -lib BITTRUE_VHDLSNPS -update

set_max_delay 20.0 -from all_inputs() -to all_outputs()

set_prefer { hcells/* }
set_dont_use { lsi_10k/* }

compile -map_effort high -ungroup_all

write -f db   -out ./db/MULT.db
write -f vhdl -out ./MULT.vhd

check_design                                > ./mult_check.rpt
report_timing                               > ./mult_timing.rpt
report_constraints -all_violators -verbose > ./mult_viol.rpt
report_cell                                 > ./mult_cell.rpt
report_area                                 > ./mult_area.rpt
report_cell all_registers()                 > ./mult_registers.rpt
report_net                                  > ./mult_net.rpt
report_clock                                > ./mult_clock.rpt
report_timing -path end -delay max -max_paths 600 -nworst 1 -to all_outputs()
>> ./mult_timing.rpt

quit
```

Synthesis Script File for 16-bit Delay Sub-block (16-bit Butterworth Lowpass Filter)

```
analyze -f vhdl ../src/delay.vhd

elaborate DELAY -arch "behavior" -lib BITTRUE_VHDLSNPS -update

create_clock CLK -name "CLK" -period 500000

set_clock_skew -uncertainty 10.0 { CLK }

/* set_fix_hold { CLK } */
set_dont_touch_network { CLK }
set_drive 0 { CLK }

set_drive 0 { RESET }
set_dont_touch find(net,"RESET")

set_input_delay  -clock CLK 10.0 all_inputs()
```

```
set_output_delay -clock CLK 10.0 all_outputs()

set_prefer { hcells/* }
set_dont_use { lsi_10k/* }

compile -map_effort low

write -f db   -out ./db/DELAY.db
write -f vhdl -out ./DELAY.vhd

check_design                                  > ./delay_check.rpt
report_timing                                 > ./delay_timing.rpt
report_constraints -max_delay -all_violators -verbose > ./delay_viol.rpt
report_cell                                   > ./delay_cell.rpt
report_area                                   > ./delay_area.rpt
report_cell all_registers()                   > ./delay_registers.rpt
report_net                                    > ./delay_net.rpt
report_clock                                  > ./delay_clock.rpt
report_timing -path end -delay max -max_paths 600 -nworst 1 -to all_outputs()
>> ./delay_timing.rpt

quit
```

Acknowledgments

First, I would like to thank God for maintaining my personal perseverance of seeing this hard work through. Next, I would like to thank a plethora of people who helped me along the way in different senses. In the academic sense, I thank my advisor, Dr. Armstrong, who presented a real challenge to me in terms of learning, applying my own intuition, and guiding me through the particulars of this research. I thank my committee members, especially Dr. Gray, for their patience and their valuable input into my research. I thank my unofficial "committee members", Dr. Amy E. Bell and Dr. Jeffrey H. Reed, for sparing time to observe and critique my work. I thank friends at Lucent Technologies for sharing concepts on how to realize my design methodology. In the non-academic sense, I thank my advisor again, Dr. Armstrong, for the short random chats we had during office and non-office hours. I thank the friends I made here for their confidence and support when I felt I was "spinning my wheels" at times. I thank my fraternity brothers who checked my "sanity" level every now and then as well as provided support. Lastly, I thank my parents, Aubrey and Annette Jackson, who offered words of encouragement and whose support mentally and emotionally helped me in ways I'm unable to convey in words. Their being there for me has put me where I am today. I dedicate this research to them. I love you. Thank you all.

Vita

Brian Aliston Jackson was born on March 28, 1970, in Kingston, Jamaica, West Indies. He is a graduate of Brooklyn Technical High School located in Brooklyn, New York. He entered the engineering dual degree program of Johnson C. Smith University and The University of North Carolina at Charlotte and graduated with baccalaureate degrees in both mathematics and electrical engineering. He was a GEM Masters of Science Engineering Fellowship Recipient and entered the masters of electrical engineering at Virginia Polytechnic Institute and State University in the fall semester of 1996. He graduated in the fall semester of 1999 with a Masters of Science degree in electrical engineering. He is continuing his academic education by entering the doctorate program of electrical and computer engineering at Virginia Polytechnic Institute and State University.