2006

Design and Evaluation of a Discrete Wavelet Transform Based Multi-Signal Receiver

Tony Chiang
Wright State University

DESIGN AND PERFORMANCE EVALUATION OF A DISCRETE WAVELET TRANSFORM-BASED MULTI-SIGNAL RECEIVER

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science in Engineering

By

TONY CHIANG

B.S. ELECTRICAL & COMPUTER ENGINEERING, University of Rochester, 2003

2006

Wright State University

WRIGHT STATE UNIVERSITY

SCHOOL OF GRADUATE STUDIES

July 7, 2006

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Tony Chiang ENTITLED Design and Evaluation of a Discrete Wavelet Transform-Based Multi-Signal Receiver BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science in Engineering

_____
Chien-In Henry Chen, Ph.D.
Thesis Director

_____
Fred Garber, Ph.D.
Department Chair

Committee on
Final Examination

_____
Chien-In Henry Chen, Ph.D.

_____
Raymond Siferd, Ph.D.

_____
Marty Emmert, Ph.D.

_____
Dr. Joseph F. Thomas, Jr., Ph.D.
Dean, School of Graduate Studies

Abstract

General purpose receivers of today are designed with a broad bandwidth so that the receiver can accept a wide range of signal frequencies. These receivers usually accept one signal along with any interference that is included. To increase the signal detection capabilities of the wideband receiver, a design for a receiver that can detect two signals is needed. One of the requirements for this receiver is that the second weak signal needs to be processed in a timely manner so that the receiver can recognize it. To remedy the problem, a module was developed using wavelet-based techniques to remove spurs from the incoming signals to allow easier detection. The main basis for this concentration on wavelets comes from the way wavelets break down signals into portions (called resolutions) that allow easier determination of detail importance. Utilizing the multi-resolution attributes of the discrete wavelet transform, a way to remove signal spurs is made possible. When removing the signal noise from the signal, the two signal dynamic range of the system is increased, as this module is applied to multiple receiver systems for comparison of performance. Implementation of this system was originally done in C as well as MATLAB, but later is being implemented in VHDL with simulations done for verification of functionality.

Table of Contents

List of Figures

List of Tables

Acknowledgements

This work was supported in part by the program of Receiver and Processing Concepts Evaluation (RAPCEval), DoD, Air Force Research Lab, USA.

I would like to thank my advisor, Dr. Chien-In Henry Chen, for his dedication, his vision, and his support throughout the thesis process. It has proven to be an arduous, but worthy rite of passage that has conditioned me for the road ahead.

I would like to also thank the members of the thesis committee in taking time out of their busy schedules to read and evaluate my thesis. Their advise and help have proven a key instrument in the completion of this thesis work.

I would also like to thank numerous individuals from the Wright Patterson Air force Base for giving me the opportunity to work with them and to develop a technology they may consider for future use.

To the students in the lab, I am especially grateful for all the help and encouragement that you have given me throughout this process. You've shown me that research can be fun too!

Last but not least, I would like to thank my family, who has been providing me a way to worry only about the task on hand. I greatly treasure and appreciate the time and effort that has gone into giving me peace of mind while working on this thesis.

Dedication

This thesis is dedicated to my parents, Win and Lancy Chiang, who taught us that innovation and "thinking outside the box" is what sets one person apart from the crowd.

Chapter 1: Introduction

General purpose receivers of today are designed with a broad bandwidth so that the receiver can accept a wide range of signal frequencies. These receivers usually accept one signal along with any interference that is included. However, when such a receiver receives two simultaneous main signals compounded with the noise from both signals, the receiver chooses the signal closest to it so that only one signal is received while the other signal is simply not processed as a signal, but rather as interference or noise. To increase the signal capabilities of the wideband receiver, a design for a receiver that can detect two signals is needed. One of the requirements for this receiver is that the second main signal needs to be processed so that it can be recognized by the receiver. To make this two main signal receiver design for general purpose, the signal frequencies should be variable, and not known to the system previously. Also, this wideband receiver must be able to discern a signal from the compounded noise from two signals. To solve this problem, a wideband receiver requires signal processing in the forms of modulation or through the process of removal of noise and signal recognition to successfully detect two signals at the same time.

In digital signal processing, a common way of remedying this problem is to use modulation through convolution [1]. From this signal, it can be reconstructed by using the same pattern of impulses when used at the same frequency. This method allows the noise in the signal to be virtually ignored. In the case where the frequencies are known, a radio system using FM modulation

is the preferred method for achieving this ability. This solution, however, requires the prior knowledge of the frequencies that the signals are being received at, which cannot make this receiver a general-purpose receiver.

To deal with the noise and signal detection in a more direct fashion, there are many designs for filters that would shape the signal to the function that the filter is implemented [1]. However, this function is not acceptable for a general-purpose wideband receiver as the signals themselves would be distorted, and finding a very weak signal would be impossible. There is a need, then, for a more dynamic method to which the signal can be directly affected by the processing without resorting to a signal shaping method.

A method for removing noise from a signal is using two threshold values obtained from large volume of radar data simulation; the large threshold is used for detection of the first strong signal and the second threshold is used for detection of the second weak signal. This method has been used in a mono-bit receiver design [2], which precedes the general-purpose wideband receiver, and has moderate performance improvement. However, with the mono-bit receiver, a strong and a weaker signal can be only detected with very little difference, up to 5 dB direct range difference, between the strong and weak signal due to the lower resolution (2 bits) of the ADC in the system. Another method for removing noise from a signal is the compensation matrix, which has a matrix of stored coefficients that have been pre-calculated to match the signal behavior and uses those coefficients to delete the predicted noise from the actual signal,

thereby exposing any weak signals. A further improvement upon the design brings us to a 1 GHz input bandwidth receiver on a chip (ROC) design. The ROC design is a design that incorporates all the functions of a receiver onto one chip, and this particular design utilizes a super-resolution block, which based on the input signals, suppresses spurious signals and exposes the true signals [3].

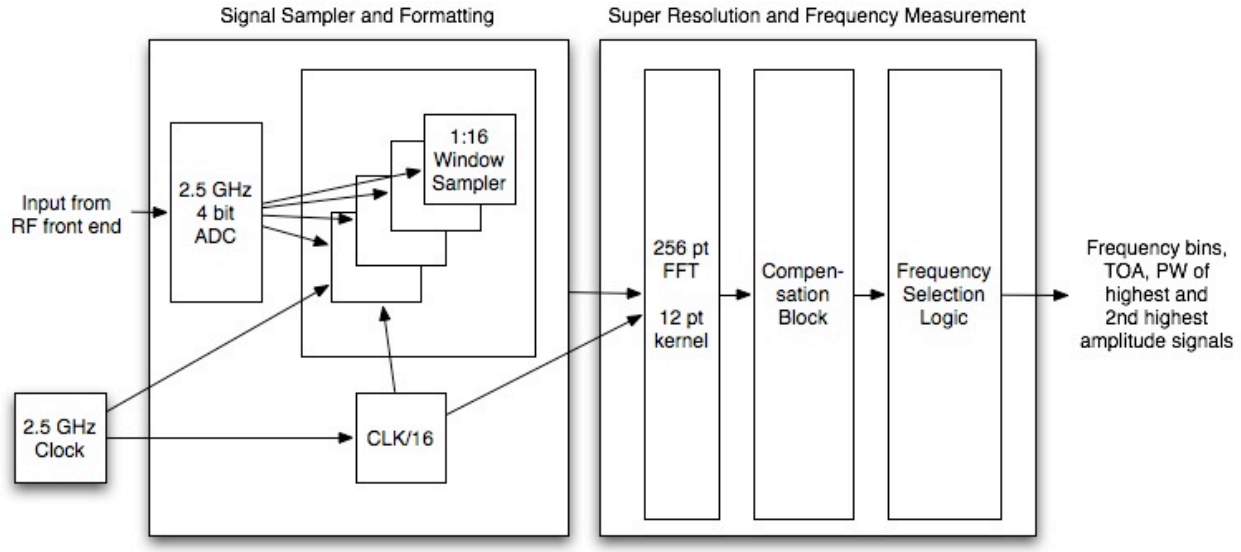

Figure 1.1: Block Diagram of Receiver Design

The receiver design can be separated into two areas: the signal sampler and formatting system, and the super resolution and frequency measurement section [3]. The signal sampler and formatting system portion comprises of an ADC that operates at 2.5 times the input bandwidth and samples the signal at 0.4 ns to produce 4 bit amplitude measurements. Each bit is then passed into a windowing circuitry, which converts the serial data stream to parallel and slows down the data rate by a factor of 16. This process is achieved by using a clock divider to slow the data stream. The slowing of the data rate is necessary to accommodate the speed at which the detection circuit can receive data.

The super-resolution and frequency measurement system relies on a 256 point FFT, a compensation block and a frequency selection logic block. The 256 point FFT is designed by using a 12 point kernel approximation which is fed the input data stream from the windows. Once all the data is collected after the FFT, it is input into the compensation block. The compensation process is essentially the comparison of a stored pre-calculated value that approximates the first detected frequency and the associated noise, then subtracts that value from the original signal response. This method assumes that if the pre-calculated signal response is the same as the actual signal response, the side lobes and spurs will be taken out while exposing any signal that is left behind to be detected as a secondary signal. In actual performance, the compensation must calculate the response close enough so that there won't be any kind of misinterpretation on the part of the detection module that would mistake an uncalculated spur as a second signal.

It is after the compensation block for which the frequency selection takes place to detect the signal. This takes place by the selection of a frequency bin that is closest to the frequency that the signal is detected at. This frequency bin is of a 10 MHz bandwidth [3].

The system performance with the compensation matrix was proven to receive a signal in acceptable specifications with a strong to weak signal dynamic range (also known as a two signal dynamic range) of 18 dB [3]. There is a need for the system performance to increase beyond that of 18 dB, however, to make the

receiver design more robust. This system performance, however, decreases exponentially when the two signal dynamic range reaches beyond 18 dB. To improve the performance of the receiver beyond 18 dB is impossible using only the current compensation matrix. Therefore, a different system approach may be more applicable to further improve the performance of the wideband receiver so that any signal, from a strong signal to a weak signal, can be detected.

The main basis for this concentration on wavelets comes from the way wavelets break down signals into portions (called resolutions) that allow easier determination of detail importance [4] [5] [6] [7] [8]. There are multiple versions contained within one wavelet transformation, so there can be a choice of how many to contain to maintain the details of each. In the example of a picture, there are many different resolutions that may contain very little information significant to the picture; therefore, the user can specify the amount of detail required for the picture by removing the less significant resolutions from the transform [5]. The same principle can be applied to a 1D signal such as the signals received by the wideband receiver. A signal can be broken down into different resolutions, where the signal can be discerned easily from the noise [6] [7] [8]. From these resolutions, a noise threshold can be specified so that only signals above the threshold will be kept while the rest of the signal will be removed or reduced in amplitude [9]. Variations in the threshold types as well as the variability of the threshold value can be analyzed and customized to fit the signal condition. When paired with the compensation matrix, an added

performance boost can be expected from this method. Since the application of this type of technology to a receiver design has not been documented before, the theory appears to be promising for the further performance boost of the receiver.

This thesis aims to solve the following problems:

1 Implement the discrete wavelet transform and thresholding technology into a module to be placed into the system to improve receiver system performance beyond 18 dB two signal dynamic range.

2 Investigate hardware implementations of the system by using simulations to further verify that the theoretical and experimental data from the software is viable in the hardware implementation.

3 Consider the hardware implementation of discrete wavelet transform based module for integration with a receiver on chip design.

Chapter 2: Theory and Background

Wavelets are small waves that are used to derive a signal depiction through stretching and shrinking the wavelet. Wavelets are localized functions in time with mean zero; the wavelet basis is derived from the wavelet (small wave) by its own dilations and translations. A general definition of a wavelet equation is:

$$w_{j,k}(t) = 2^{\frac{-j}{2}} w(2^{-j}t - k)$$

Let the original wavelet (also known as the mother wavelet) start at t = 0 and end at t = N. The shifted wavelet $w_{0,k}$, starts at t = k. and ends at t = k + N. The rescaled wavelet $w_{j,0}$ starts at t = 0, and ends at $t = N/2^j$.

The choice of the basis is related to the choice of the filters. The basis must present two main properties: linear independence and completeness. In the creation of the wavelet, there are two major components that define the wavelet transform: the details and the overall structure of the wavelet, based on the dilations and translations. The complexity of the wavelet transform theory utilized determines the closeness of the wavelet adhesion to the original signal depiction [8]. This depiction of a signal requires a criteria for the different resolutions of the signal in the transform domain to be formed so that the performance of the wideband receiver will be improved.

There are a few wavelet transform schemes that are available for usage in a system integration with varying degrees of complexity, making the choice of transforms a balance between hardware complexity and accuracy to the original signal, making a thresholding cut a more accurate removal of noise [9].

2.1: Haar Wavelet Transform

This wavelet transform relies on a rectangular function, which utilizes the basic approximating functions that would estimate the shape and detail of the waveform that is being transformed [6]. Simplified, the Haar wavelet transform uses an approximating function that is a variation of a step function.

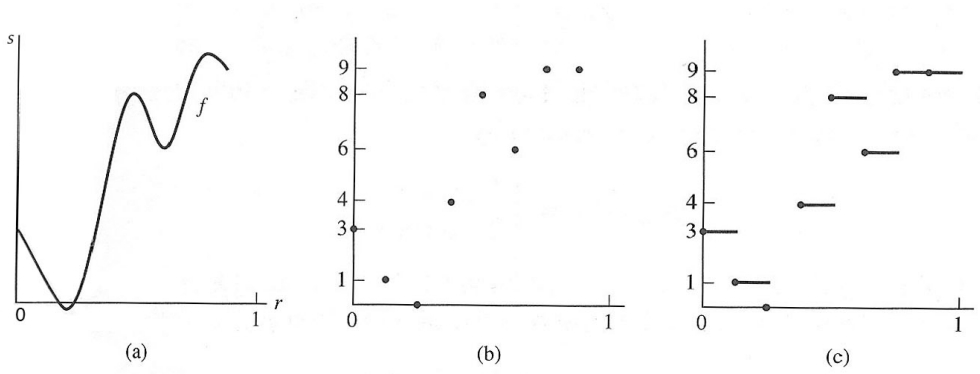

Figure 2.1: (a) Signal, (b) Sample (c) Approximation for Haar Wavelet [7].

Haar's basic transformation expresses the approximating function f with wavelets by replacing an adjacent pair of steps by one wider step and one wavelet. The wider step measures the average of the initial pair of steps, while the wavelet, formed by two alternating steps, measures the difference of the initial pair of steps [6][7].

This method is the basic wavelet transformation, and is not suitable for higher definition signals, as is expected in a wideband receiver design. While acceptable for lower (4 to 6 dB) two signal dynamic ranges, similar performance at 18+ dB cannot be expected as the resulting signal is depicted as step functions. This transform method, then, is not a viable candidate for improving system

performance based on a more accurate depiction, and therefore more accurate de-noising, of the signal spectrum.

2.2 : Daubechies Wavelet Transform Method

Ingrid Daubechies introduced a new series of algorithms that transformed using wavelets that contrast from Haar's wavelet transform in that the Daubechies wavelet transform is a continuous signal rather than Haar's discontinuous step design [6] [7] [8]. As a consequence of the continuity, continuous signals have a more accurate representation; however, this continuity has the cost of using complex equations and implementations, resulting in more complex hardware. Some of these functions are based on matrix calculations.

In order to design continuous wavelets that are time-efficient and implementable in a system use, Daubechies introduced a basic building block or scaling function as denoted by φ. An example of how Daubechies' wavelets are calculated is noted below for initial conditions:

$$\varphi(0) := 0,$$
$$\varphi(1) := \frac{1+\sqrt{3}}{2},$$
$$\varphi(2) := \frac{1-\sqrt{3}}{2},$$
$$\varphi(3) := 0.$$

These conditions satisfy the condition that:

$$\varphi(r) = \frac{1+\sqrt{3}}{4}\varphi(2r) + \frac{3+\sqrt{3}}{4}\varphi(2r-1) + \frac{3-\sqrt{3}}{4}\varphi(2r-2) + \frac{1-\sqrt{3}}{4}\varphi(2r-3).$$

It is also required that the initial values add up to 1 so that the values of φ may serve as averaging or weighting factors.

$$\varphi(0)+\varphi(1)+\varphi(2)+\varphi(3)=0+\frac{1+\sqrt{3}}{2}+\frac{1-\sqrt{3}}{2}+0=1$$

The function φ serves as the basic building block for its associated wavelet, denoted by ψ, and defined by the following notation:

$$\psi(r):=-\frac{1+\sqrt{3}}{4}\varphi(2r-1)+\frac{3+\sqrt{3}}{4}\varphi(2r)-\frac{3-\sqrt{3}}{4}\varphi(2r+1)+\frac{1-\sqrt{3}}{4}\varphi(2r+2)$$
$$=-h_0\varphi(2r-1)+h_1\varphi(2r)-h_2\varphi(2r+1)+h_3\varphi(2r+2)$$
$$=(-1)^{(1)}h_{1-1}\varphi(2r-1)+(-1)^{(0)}h_{1-0}\varphi(2r-0)+(-1)^{-1}h_{1-[-1]}\varphi(2r-[-1])+(-1)^{-2}h_{1-[-2]}\varphi(2r-[-2])$$

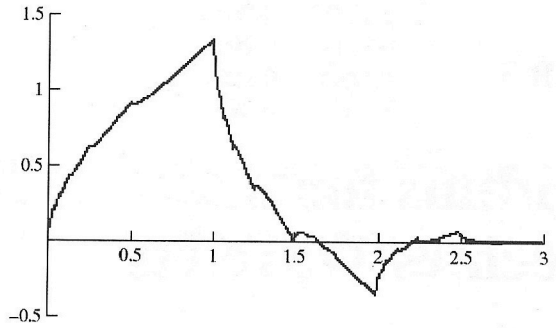

Figure 2.2: Daubechies Wavelet [7]

The Daubechies wavelet is calculated in this fashion. It can be seen that the Daubechies' wavelet method utilizes vanishing moments. Vanishing moments are described as a point in the function where it dips below the frequency axis. Vanishing moments contribute to the smoothness of the transformed wavelet because they allow the system to compensate for the differences in the transfer function without sacrificing expensive hardware implementation [8]. The signal adhesion is more apparent in the Daubechies wavelet transform than that of the Haar wavelet [10], making the Daubechies wavelet transform one of the more popular implementations of the discrete wavelet transform. This method for

transform and thresholding is much more promising for improving system performance of the receiver than the Haar wavelet transform.

2.3 : Thresholding and Noise Reduction

As discussed before, the wavelet transform separates the time-based signal into details and a general form for the signal. It is in the details that most of the noise creates a problem because the details are so much smaller in terms of frequency amplitude. Adding a zero where the noisy coefficients are is a possible solution to create a cleaner signal [9]. The main motivations of this method can be summarized with the following assumptions and observations:

1. The de-correlating property of the wavelet transform creates a sparse signal: most of the untouched coefficients are close to zero or at zero.
2. Noise is spread equally throughout all coefficients.
3. The noise level is not too high so that the signal cannot be differentiated from the noise.

Adding a threshold that adds a zero below a certain value allows the signal to be cleaned with a simple and efficient design. This simplicity is what makes thresholding a common solution to reducing noise in a signal.

Replacing small coefficients with a zero if under a certain threshold value is called hard thresholding. Another method, which coefficients above a coefficient are reduced by an absolute value, lends itself to more continuity in the coefficients. There is a clean transition between the noise coefficients and the signal coefficients. This method, called soft thresholding, is particularly effective

in maintaining mathematical controllability; for example, a discontinuous signal is not useable for use in a system that is made for continuous signals or may cause some un-desirable results. For these cases, a soft threshold, which maintains continuity, would be clearly a better choice than the discontinuous hard threshold method [9].

However, setting the type of threshold is only one parameter that needs to be set. An important parameter is the threshold selection. This threshold can be based on a few items: variance deviations, namely mean absolute deviation and numerical standard estimate; or threshold estimators using mean squared error. Mean squared error is primarily used as a threshold estimator for soft thresholding and applications include image processing as well as sound processing [9].

For this design, however, an absolute median of the signal is used as an estimator for the threshold. The basis for this design is because per input signal after the discrete wavelet transform, the majority of the important details (the main part of the true signal) will be contained in the first half of the transform while the second half will contain the majority of the details, which includes the noise in the signals. Following this logic, the second half of the data is sorted, then a median is found. From the median, the threshold can be estimated because all the small details should have amplitude with much less fluctuation than in the first half of the data. Another method that could be used in estimating the threshold is the use of a statistical standard deviation method.

This method is more complicated, and may require more hardware or computation time. In a system design that requires as little computation as possible to reduce power consumption as well as hardware real estate, a simplistic approach using the median would be more appropriate for this application.

2.4: Hardware Implementations

The major considerations for the design deal with the design of the discrete wavelet transform itself. In the transform, there are many recursive processes that occur; these processes rely heavily on memory to store previous values for recalculation. Another major consideration is the filters that would be used to implement the discrete wavelet transform. These filters differ based on different types of wavelet transforms; like in most cases, a more complex filter will produce better results, but also result in complex filter hardware implementation, requiring more space and power.

The discrete wavelet transform implemented in this design is a Daubechies Wavelet Filter, using a 4 –tap design, which is simple to implement because of its orthogonal properties as well as the simple shortness of the filter. It also satisfies the perfect reconstruction conditions [7][8]. In designs such as the ones used in [4] and [12], the filters were implemented directly in the system since they could vary based on the inputs of the chip. The most important characteristic of the FIR filters obtained from Daubechies functions is that they are Power Symmetric, which allows implementation of those filters using a Lattice Structure, as

described in chapter 6.4 of [10]. The Lattice Structure has many advantages, such as better coefficient quantization response as well as a reduction by a factor of two of the stages needed for a given filter order [11].

Another method of implementing the filters is described in [13], using a different method for implementing a DWT system, and compares the convolution-based, lifting-based, and b-spine based designs. For one dimensional DWT, the architectures are mainly convolution-based and lifting-based, while other designs, including the b-spline based architecture, is more suitable for two dimensional DWT. Traditionally, a convolution-based system is used for the filter system. This system provides an accurate transform and reconstruction of the signal when processed in a transform and an inverse transform. However, this method also requires space and some complex arithmetic to take place; therefore, some other methods are investigated [17].

When composing a DWT design, the input is divided into an even and odd signal, which is processed so that the output of the transform is a high pass sequence and a low pass sequence. This concept is illustrated below:

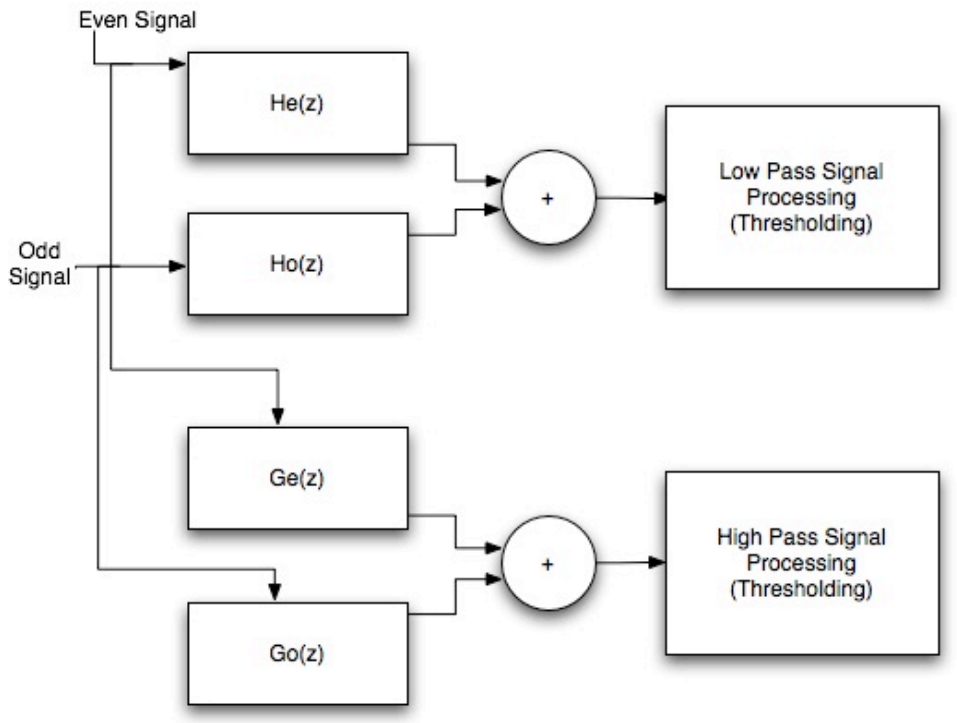

Figure 2.3: Discrete Wavelet Transform Hardware Block Diagram [16]

The filters here determine the complexity of the circuit. For the lifting-based concept, factorization of the filter coefficients using a lifting scheme allows for hardware cost to be reduced and for complexity to also be reduced. However, the cost for using a lifting-based scheme may result in a longer critical path [15]. This is because the factorization may result in different stages to connect differently from the convolution-based scheme. Optimized correctly, however, should maintain the critical path while reducing hardware.

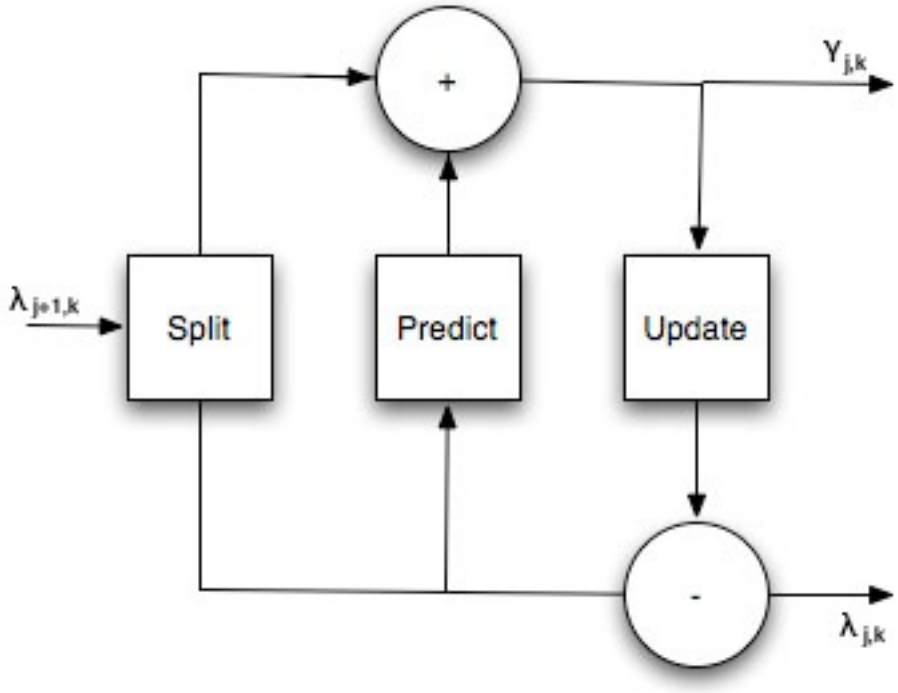

Figure 2.4: Lifting-Based DWT Architecture [12]

The lifting based scheme has three simple steps, applied repetitively on the samples: the split phase, predict phase, and the update phase. The split phase needs to split the incoming signal into two separate types, ideally, the odd and even signals. The first stage is called the lazy wavelet transform because it simply sub samples the signal into even and odd samples. During the Predict Phase, the even subset is used to predict the odd subset of the system. The purpose of this predict phase is to eliminate the need for the odd samples by predicting it using the even samples. To denote correlation differences between the odd and even channels, the predicted values are subtracted from the actual values. The more correlation between the even and the odd samples, the better the calculations. The Update Phase is where the samples are lifted with the help

of the neighboring wavelet coefficients so that the mean of the samples is preserved. The lifting wavelet transform requires less calculations than a convolution method because it eliminates the need to preserve the odd samples and uses lifting, which preserves the mean of the values in the transform [12].

There are other methods of implementing the discrete wavelet transform, including a design that would utilize look up tables (LUT) for storing the values of the coefficients or possibly the products of the coefficient and a number, which would be the input address. This method takes advantage of the space efficiency of the look up tables and is a viable alternative to multipliers, which take up a large amount of space.

The memory is also used to store the values for the calculation of the compensation table, so allocating some memory for the calculation of the DWT and the inverse DWT is easily accomplished. For the threshold estimation, the temporary storing and replacing of values will require some memory as well, but these operations may utilize the same memory area and will not require much more memory. Pipelining of the sorting algorithm blocks, however, may be needed to compensate for the large amount of data being input into the system and will also require hardware space, both in terms of the blocks as well as memory.

Implementations of the discrete wavelet transform are largely done on FPGAs because FPGAs have integrated multipliers and adders [11][14][15]. This attribute of FPGAs is attractive because multipliers and memory units are

expensive in terms of hardware real estate in ASIC design. Also, FPGAs provide a faster development cycle than ASIC. However, ASIC provides a faster performance speed and also takes up less space than the FPGA design. For the video applications of the DWT, ASIC is a preferable implementation scheme and provides the throughput that FPGA may not be able to offer [15]. For the receiver design, either path can be followed, but it remains to be seen which technology would be more practical for implementation.

Chapter 3: Methodology

The methodology for developing a receiver design improvement requires several different steps, and use of different types of equipment. The advantages and disadvantages of some of the steps are explained as well as the steps that were taken for the implementation of the receiver design. The steps are as follows:

1. Data Generation
2. ADC Background
3. FFT
4. Compensation Matrix
5. DWT De-noising
6. Frequency Selection (Signal Detection)

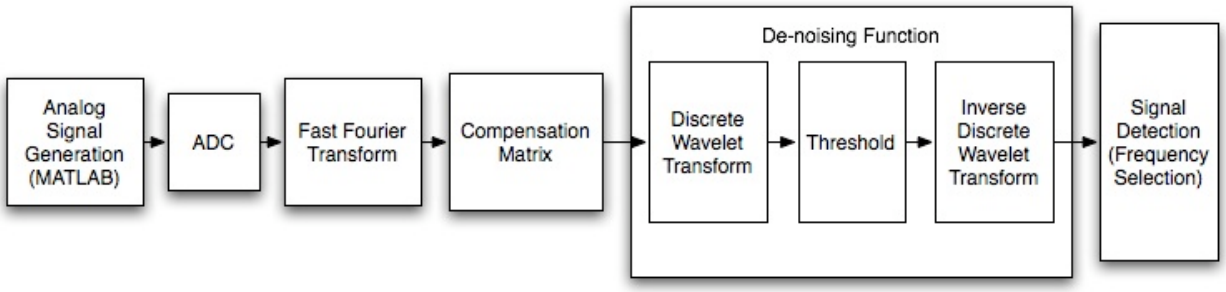

Figure 3.1: Block Diagram of Steps

The block diagram above depicts the flow of the system as it is being used for simulations. Each block is described in the next sections. The receiver is also designed and implemented in hardware. The detailed implementation of the hardware is described in Section 3.7.

3.1: Data Generation

The data sets for the simulation of the receiver design are generated frequencies using MATLAB, ranging from 125 MHz to 1125 MHz. These signals are generated with a minimum of 10 MHz difference between one another, and also have one signal generated as a strong signal and another signal generated as a weak signal. These signals are also generated with random noise added to the spectrum. This represents a typical signal input data for the receiver.

3.2: ADC Background

The ADC is used to convert the generated continuous time signal into a digital signal so that it can be processed easier. The highest frequency that can be received by the ADC is 1.125 GHz, so the 2.5 GHz sampling rate of the ADC corresponds with the Nyquist rate. In the MATLAB simulations, a simple quantizer is used for the function that the ADC would perform. However, a ADC design is required for the hardware implementation, which has the general design shown Figure 3.2.

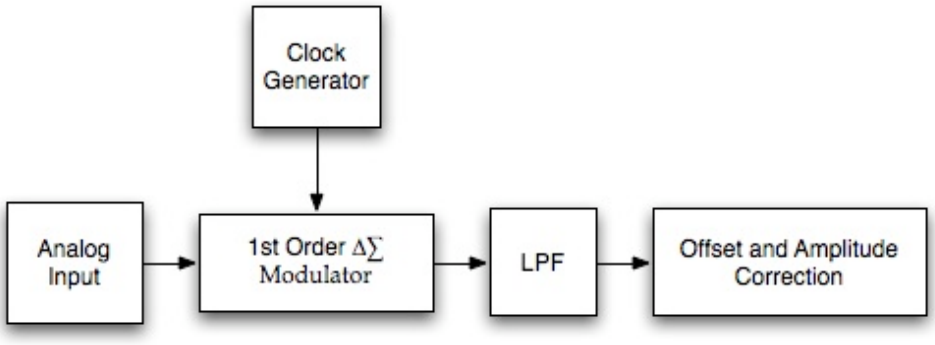

Figure 3.2: ADC Block Diagram

The ADC design is programmed in AMS Verilog and interfaces with the FFT in the receiver design.

3.3: Fourier Transform

The next step is to transform the digital data after the ADC into frequency domain for processing. The Fast Fourier Transform was chosen for the implementation of the Fourier Transform. The Fast Fourier Transform (FFT) is designed to generate 256 points. Each point generated by the FFT is approximated by a unit circle with 12 points on it and choosing the point closest to the point generated. This approximation is the 12 point kernel FFT. The 12 point kernel 256 point FFT is designed and written in MATLAB code for simulation use as well as for ease of numerical handling.

3.4: Compensation Matrix

The incoming 256 points from the FFT is a mirrored spectrum, so half of the spectrum (128 points) is used for processing the signal. To remove the noise from the incoming 128 points from the FFT, a compensation matrix, which contains pre-calculated noise in the matrix, is used to process the signal. This calculation comes from previous data patterns that were generated and stored in memory. An algorithm selects the compensation row. The compensation matrix is implemented in MATLAB code by using variables to denote the memory slots and implementing a separate algorithm to search and select a specific row for use in the signal processing. The information in the compensation row is used to subtract from the FFT signal, theoretically "exposing" any second signal that is

possibly overshadowed by the noise in the system. Theoretically, the noise should be completely removed from the signal, but in reality, this is not the case.

3.5: De-noising Function

The de-noising function is composed of three sections: the discrete wavelet transform (using the Daubechies implementation), thresholding, and an inverse wavelet transform. Further removal of signals requires the Daubechies wavelet transform, thresholding, and an inverse transform. The main inputs of this system are the filter coefficients, threshold value set by the user, and the signal from the compensation matrix. The filter coefficients are precalculated and stored in memory, then input into the de-noising function. The filter coefficients are calculated using MATLAB code and insert the values generated into the different components as matrixes.

3.5.1: Discrete Wavelet Transform

The discrete wavelet transform mainly utilizes the filtering coefficients to process the incoming signal from the compensation matrix. Based on these inputs, the discrete wavelet transform can calculate the Daubechies wavelet based on the coefficients that are provided by the user. The software implementation of this module is in C++ and is compiled in MATLAB for compatibility with the other modules. Mathematically, the Daubechies wavelet is calculated as explained in Chapter 2.2. From the mathematical calculation, the output signal is broken down into a signal like the one shown, using the coefficients [0.4830, 0.8365, 0.2241, -0.1294]:

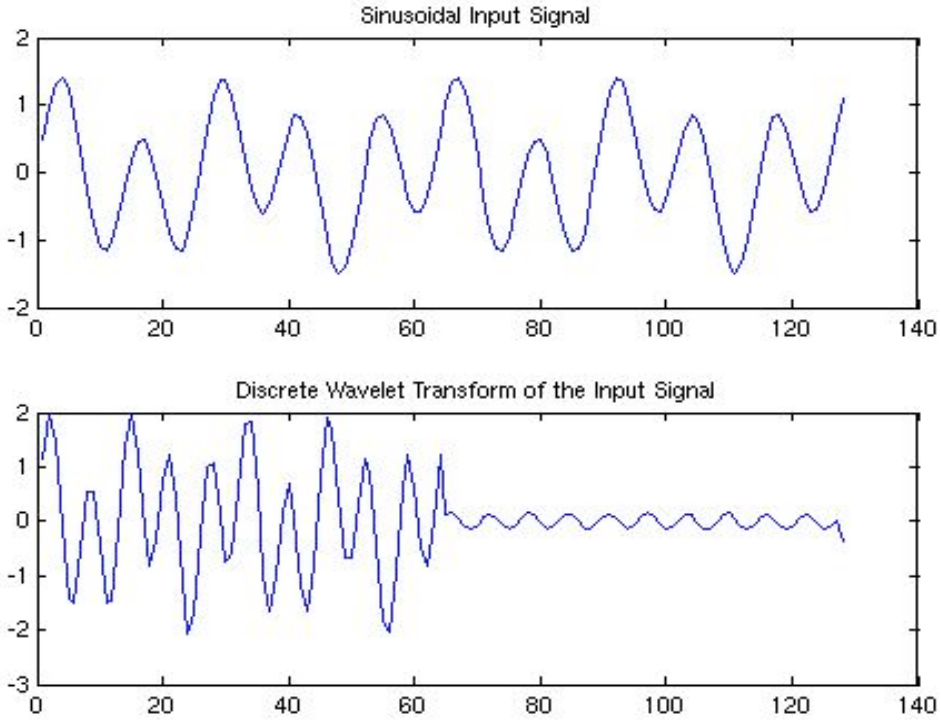

Figure 3.3: Discrete Wavelet Transform of a Sinusoidal Signal

The benefit of using the DWT comes from virtually the second half of the wavelet signal in that the small details can be removed as "noise" from the signal. To achieve that, thresholding is required.

3.5.2: Thresholding

The signal is sent to the thresholding section, which has inputs of the signal and the threshold value that is selected by the user. The user-defined threshold value, however, is only a component for the application of the threshold to the wavelet domain. This threshold value is used for estimating the threshold value that will be cut from the signal. This estimation is done by using the Mean Average Deviation, which utilizes an absolute median function. The median is based on the input signal, specifically, the small detail section of the discrete

wavelet transform. This used in the threshold by multiplying the value by the user-defined input to properly define and specify the wavelet domain value. Once the threshold is properly defined, the values below the threshold are removed from the signal (replaced by 0). The threshold element is implemented using MATLAB code that reads the output of the C++ DWT portion and also reads in the threshold set by the user from the header MATLAB file.

3.5.3: Inverse Wavelet Transform

The signal, now in the wavelet format, is inverse transformed into the original frequency domain signal. The inverse wavelet transform utilizes the same filter coefficients which are inversed for perfect reconstruction. The purpose of performing the inverse transform is to allow the system to recognize and properly detect peaks in the system. The inverse wavelet transform is implemented using C++ code and utilizes the same compiler for MATLAB compatibility. It also reads in the filter coefficients from the header MATLAB file. The coefficients for the inverse wavelet transform are the same as the filter coefficients, only reordered and also two coefficients are negated.

3.6: Signal Detection (Frequency Selection)

To detect the two originally transmitted signals, the two highest amplitude peaks in the compounded signal from the inverse wavelet transform are assumed to be the originally transmitted signals. However, since sorting the values in the signal require too much processing, a simpler method of using a threshold is utilized. The two signals from the compounded signal are detected

based on a threshold where frequencies above a certain threshold are assumed to be detected while the frequencies below the threshold are ignored. For the signals that are detected, they are still in the frequency domain. Signal detection is implemented in the software by using MATLAB code, which deals directly with the output of the C++ inverse transform function. An example of signal detection through thresholding is shown below:

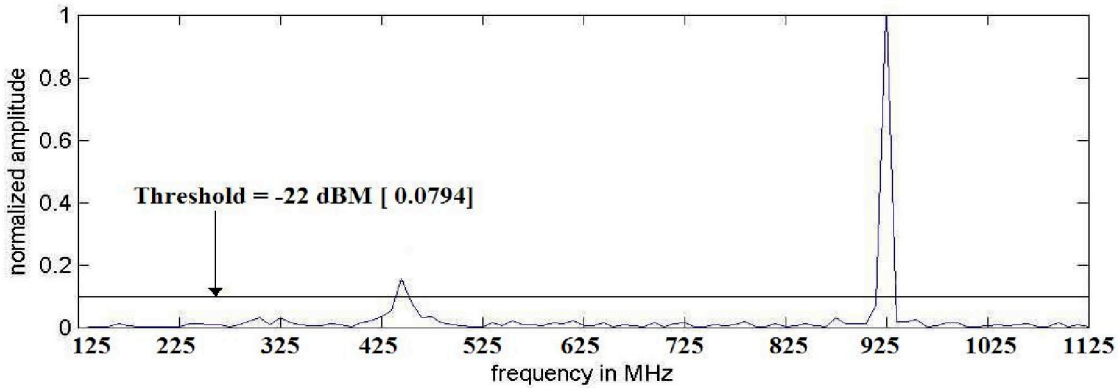

Figure 3.4: Signal Detection Through Thresholding

This figure is an example of signal detection through thresholding, using a threshold of -22 dBm. As shown above, the two highest peaks are detected as the two signals.

3.7: Hardware Implementation

Hardware implementation for this receiver is done in two languages, AMS Verilog and VHDL. The ADC is implemented in AMS Verilog, because AMS Verilog is able to deal with both the analog and digital portions of the ADC. The simulations for the hardware are done using Synopsis for VHDL and Cadence Custom IC tools for AMS Verilog. Cadence Custom IC Software used IBM's 0.13

micron technology for simulation of the receiver as a whole. This software runs on the Sun machines, using Solaris 8.0. Cadence can use Verilog, AMS-Verilog, and VHDL among other languages for hardware implementation. Usually, a receiver receives the signal as an analog signal, then an ADC is used to convert the analog signal into a digital signal. Both signal processing steps need simulation for complete system verification. As a result, for simulations with the whole system, AMS-Verilog was used for it's flexibility with both the analog and digital parts of the system. The Fast Fourier Transform, the compensation matrix, de-noising function, and the frequency selection are all implemented in VHDL as well, since they all deal with digital signals.

The denoising function is originally implemented in C, but to transfer the same functionality from C to VHDL requires directly translating the function of the C code to VHDL using a structural approach. Because the discrete wavelet transform requires multiplication, look-up tables were used to take the input value and search for the value in the look up table that would be equal to the input value multiplied times a coefficient, for example. This eliminated the requirement of using multipliers with a more hardware efficient result.

Chapter 4: Results and Discussion:

4.1: Simulation Environment

A large portion of the simulations required for the design of the receiver on chip was performed by using MATLAB, an industry and educational standard tool for math, applied science and engineering. MATLAB has the capability to work with numerical processing and has many tools and utilities that are highly suited for signal processing, as well as compatibility with different programming languages. This makes the MATLAB environment an attractive choice for prototyping different signal processing designs. The implementations of the software in the MATLAB environment can be pure MATLAB native code, or a mixture of MATLAB code and other language code. The native MATLAB code is a simple design, which each command calls a function that is in the MATLAB library. However, MATLAB code is an interpreted language, which performs like a script. For example, each command calls a function in the MATLAB library, and each value that is achieved is re-inserted into another function. The nested functions causes the run-time to be relatively slow and processor intensive, as the functions continuously call one another and swap inputs and output data. To overcome this deficiency in speed, MATLAB also has the capability to interpret and use other programming languages, which may be compiled languages. This provides a significant benefit in speed, as the program need not call another function in MATLAB, but have all the functionality available in the compiled program already. This capability is significantly

important to complicated computational programs, which does not need to be re-compiled every simulation run. MATLAB only requires that there be a header program that simply denotes the inputs and outputs of the function. This is why there is a mixture of MATLAB code and other code: to allow the interfacing of MATLAB's powerful mathematical capabilities with the practical functionality of other languages.

From the simulation perspective, any machine could be used for simulation. MATLAB is a program that is ported to all the major operating systems, so the simulations were run on three different machines, all using different configurations. The Sun machines, using a 64-bit architecture but with slower clock speed, processed simulations about the same speed as an Intel Pentium 3, which had a higher clock speed, but 32-bit architecture. Running the simulation on an IBM PowerPC G4 processor yielded similar performance. This performance similarity can be attributed to the way MATLAB was designed, and how the GUI interacts with the main core of the program. For the Solaris and Apple Mac OS X versions, both had a GUI that would access the main core program with every command. Through the console, both the Solaris and Mac OS X versions could be accessed through the UNIX prompt and without using the actual GUI. This yielded much faster simulation times than the Windows version, which requires the GUI to interact with the main core system. From this consideration perspective, using the UNIX based system, such as Mac OS X and

Solaris, produced faster simulation runtimes and therefore was the choice of operating system for MATLAB simulations.

Of the languages that MATLAB can use, the easiest to program with the best performance results would be C++. Because C++ is a compiled language, the performance is better than an interpreted language, such as Java. For the signal processing purposes, C++ was the choice of programming languages to implement the discrete wavelet transform (DWT) and it's inverse transform. To properly use the C++ programs with MATLAB, the programs needed to be compiled into MATLAB-compatible files (MEX files). Using MATLAB's C/C++ compiler and MEX file creator, the binaries for the DWT and the IDWT can be implemented for compatibility with MATLAB.

The choice of programming languages was based on the capabilities of the languages to deal with the simulation data and to do numerical processing. MATLAB's native language is able to deal with the numerical processing from the different functions that are implemented in the system. However, to access many of the DWT's architectural designs without re-compiling the DWT software each simulation, C++ was used to design the main computationally intensive modules. Once the modules are compiled, they don't need to be compiled again, thus alleviating extra processing.

In the design of the simulations, there are signals to be generated. To generate the signals to be received by the simulated receiver, two frequencies are designated at random as the two signals and are generated within a 1GHz

bandwidth (from 125 MHz to 1.125 GHz) and have at least 10 MHz separation between them. The strength of these frequencies are defined based on the scenario that is being investigated – for example, where the dynamic range between the highest signal and the second signal is 18 dB, 20 dB, etc. This dynamic range is called the "two-signal dynamic range."

Generated with these signals is random noise, which the receiver must be able to compensate for, especially for greater two signal dynamic ranges (i.e. >18 dB). As mentioned in Chapter 2, scenarios where the noise becomes similar in amplitude and power as the second signal will require some form of signal processing for noise reduction. For each simulation, a randomly generated signal set (1 signal at a "close" range, and 1 signal at a very "far" range, with a difference in 18+ dB between the close and far signal) is done 1000 times, with each time recording the statistical data (1st signal detected, missed, 2nd signal detected, missed, or false alarm). For further proof of concept, a run of 10,000 times is done to further investigate the system performance.

As previously mentioned in Chapter 2, the compensation matrix relies on a set of calculated outcomes of the FFT, and therefore, subtracts the value that is pre-calculated from the actual signal data, exposing the second signal and reducing the actual noise in the signal.

Figure 4.1: General Flow of Receiver Design without De-noising

The de-noising function utilizes a discrete wavelet transform, a threshold, and an inverse transform, as covered in Chapter 2. This method of removing noise from an input signal is based on the use of the inherent properties of the wavelet transform as well as the thresholding methods, which require the use of either a hard or soft threshold. Hard (all values below the threshold are set to 0) and soft (all values are reduced by a certain value) have their own applications for different cases; for the receiver design, both would be investigated to further improve results of the simulation.

4.2: Determination of Placement of De-noising Function

Purpose of experiments: To determine the proper placement of the discrete wavelet transform-based de-noising module for effective use in the receiver design.

The first experiments with the de-noising function that uses the discrete wavelet transform dealt with verification of the functionality of the module in a system performance level.

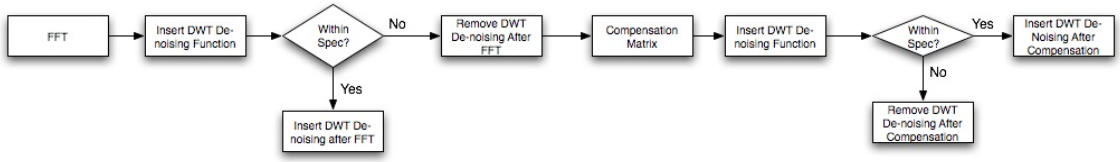

Figure 4.2: Flow Chart for Determining Placement of DWT De-noising

As can be observed from the diagram shown in figure 4.2, there are two methods for the application of this module in the system: placing the module before the compensation matrix or after the compensation matrix. The reason for placing the de-noising function in these two different areas is because the signal

coming from the fast-Fourier transform is a good signal for which the de-noising function can work with. Because the wavelet domain is unknown to the compensation matrix, and thus, renders it useless, there is a need for the FFT to remain in the design.

For 1,000 runs, a test of the different placements of the de-noising function is performed. The frequency of missing the second signal by using two alternatives, placing the de-noising module before or after the compensation matrix, in the system was counted as "2nd Missed" in the 1000 runs. Likewise, the frequency of mistakenly counting the noise as the second signal was counted as "2nd False Alarm" in the testing. The results can be seen in Table 4.1.

	Before Comp.	*After Comp*	*Difference (Δ)*
2nd Missed	53%	18%	35%
2nd False Alarm	30%	<1%	29%

Table 4.1: Comparison of Second Signal Missed Signal and False Alarm Based on Placement of DWT De-noising

Comparing the two configurations, there is a clear difference in performance between setting the de-noising function before the compensation matrix and setting the de-noising function after the compensation matrix. According to set design specifications, the system must have less than 20% second signal missed, and less than 1% second signal false alarms. As noted by the results above, the best method for using the de-noising function is to place it after the compensation matrix to act as an additional filter.

4.3: Determination of Threshold Type

Purpose of Experiments: To determine the best threshold type and threshold value for the best performance in the receiver design.

After observing that the hard thresholding produced more missed signals, but less false alarms from experimental results, it can be concluded that there is a balance to be struck, depending on the threshold value and the type of threshold. In the case of the receiver design, the least number of false alarms is desired, so the hard thresholding was chosen initially.

The following table shows the results by using the 5 different threshold values. Each of the threshold values were tested by using 3 sets of 1,000 random generations of two signal frequencies compounded with random noise using MATLAB. The various threshold values were tested for performance at a two signal dynamic range of 18 dB.

Threshold Value	False Alarm %
2.2	1.3 %
2.3	0.93 %
2.4	0.68 %
2.5	0.77 %
3.0	1.6 %

Table 4.2: Results of Simulations Testing Threshold Values

From Table 2, it is easy to see that the threshold value of 2.4 would be the best choice for the system to run on at the 18 dB two signal dynamic range. When running through further 10,000 random signal generations, the results showed the false alarm percentage at 0.68% and missed signal at 18.21 %. In comparison

with the compensation matrix alone (0.9 % false alarm percentage), the de-noising module is able to effectively aid the performance of the system.

After the threshold value being set, various different experiments were performed using the new threshold value with higher dynamic ranges and further exploring the design. However, at this point, an error was discovered with the compensation matrix, skewing the results so that it would register unacceptable results. This problem was solved by using a different set of rules for the compensation table. While the problem was being addressed, consideration of using the de-noising function with older receiver designs to further prove the effectiveness of the wavelet-based de-noising function in different applications.

4.4: Determining Performance Improvement in Mono-bit Receiver

Purpose of Experiments: To determine the performance improvement of the de-noising function when added to the mono-bit receiver design by examining the false alarm cases.

Previous designs of the receiver utilized either the compensation matrix, or did not have any additional technology to detect two signals. To further verify the functionality of the de-noising function in the system, the de-noising function was put into the previous designs, such as the monobit receiver. The main performance specification for the mono-bit receiver is the false alarm rate not to go beyond 1%. There is no specification for the second missed signal as the signal can always be sent again. The mono-bit receiver design has a 2 bit ADC,

and a 4 point kernel FFT. The results for the mono-bit receiver using the de-noising function are as follows in Table 4.3:

dB	Threshold	Hard/Soft	1st Detect	2nd Detect	2nd Miss	1st False	2nd False
4	2.4	Hard	99.96%	30.19%	68.96%	0.04%	0.85%
4	2.4	Soft	99.95%	27.24%	72.47%	0.05%	0.29%
4	1.6	Hard	99.98%	29.83%	69.29%	0.02%	0.88%
4	1.6	Soft	99.99%	27.90%	71.66%	0.01%	0.44%
4	3.0	Soft	99.95%	27.93%	71.82%	0.05%	0.25%
5	2.4	Hard	99.99%	24.44%	74.73%	0.01%	0.83%
5	2.4	Soft	99.95%	23.27%	76.50%	0.05%	0.23%
5	1.6	Hard	99.99%	23.47%	75.76%	0.01%	0.77%
5	1.6	Soft	99.99%	23.84%	75.81%	0.01%	0.35%
6	1.6	Hard	99.99%	19.80%	79.38%	0.01%	0.82%
6	1.6	Soft	99.99%	19.77%	79.98%	0.01%	0.25%
6	2.4	Hard	100%	19.68%	79.69%	0%	0.63%
6	2.4	Soft	99.98%	20.94%	78.71%	0.02%	0.25%

Table 4.3: Results of Simulations Using Random Generation of Signals in Mono-bit Receiver

From these results it can be seen that the soft threshold seems to create more missed signals, but less false alarms than hard threshold, which seems to decrease missed signals, but increase false alarms. The two threshold values were alternated between 2.4 and 1.6 to further test the validity of having a single threshold value.

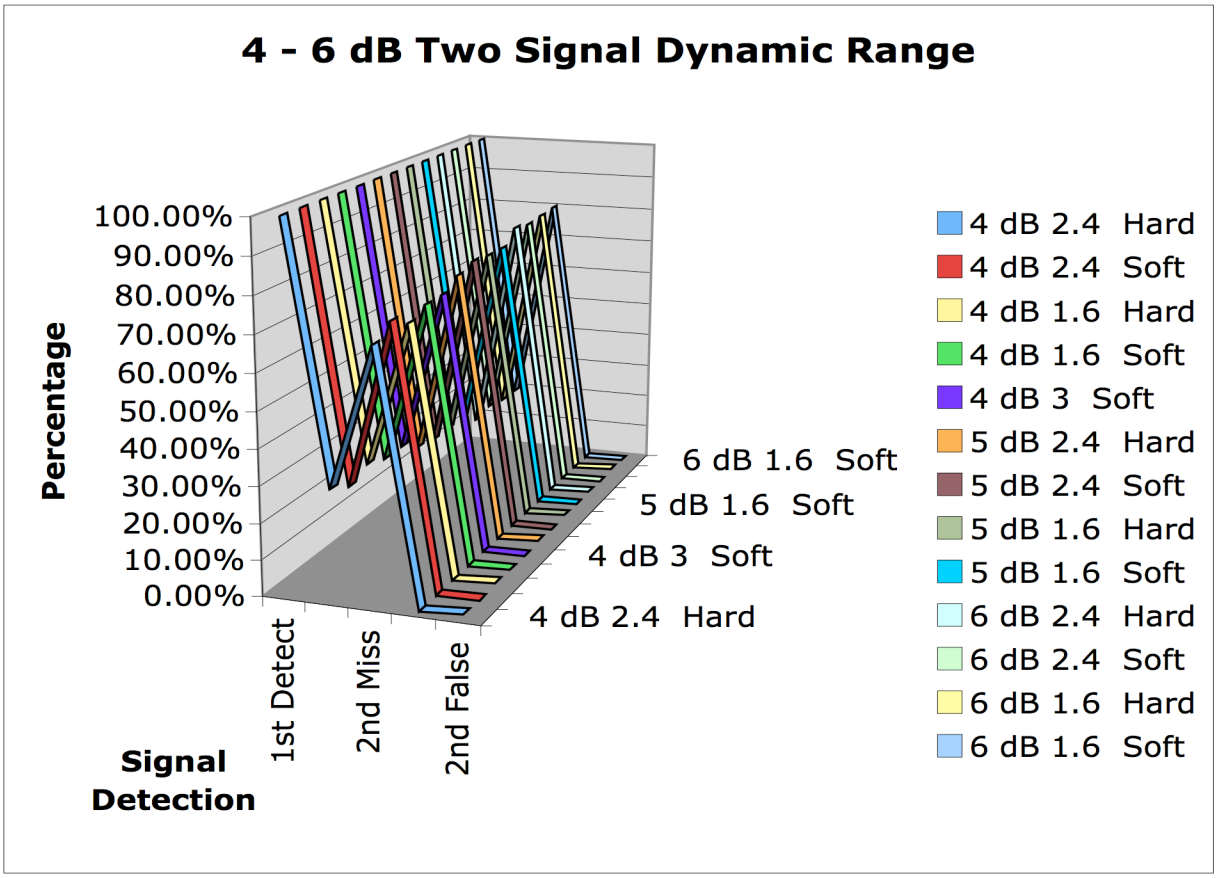

Figure 4.3: Graph of Signal Detection Types, Percentages and Types of Threshold

Figure 4.3 shows the performance of the system using the de-noising function in terms of first signal detection and second signal detection for different threshold values and types of threshold in the mono-bit receiver design. As can be determined from Figure 4.3, the second signal detection is around 30% while the second signal false alarms remains less than 1%. Looking closer at the different thresholds and types of thresholds, the general trend of the results shows that at 4 dB, the 2.4 hard threshold provides a better result than the 2.4 soft threshold or the 1.6 hard and soft thresholds, as the second missed signal is the lowest at that setting. But as the two signal dynamic range gets larger, the

different types of thresholds provide less and less of a difference. As can be seen with the 6 dB results in Figure 4.3, the second missed signal is virtually the same, no matter which threshold setting is used. This shows that as the two signal dynamic range increases, the difference in using different types of thresholds and threshold values significantly reduces the effect on simulation results.

A closer look at the behavior of the module in the system reveals a better comparison as shown in Table 4.4:

DR	Threshold	Hard/Soft	1st False	2nd False
4 dB	2.4	Hard	0.04%	0.85%
4 dB	1.6	Hard	0.02%	0.88%
4 dB	2.4	Soft	0.05%	0.29%
4 dB	1.6	Soft	0.01%	0.44%
4 dB	3	Soft	0.05%	0.25%
5 dB	2.4	Hard	0.01%	0.83%
5 dB	1.6	Hard	0.01%	0.77%
5 dB	2.4	Soft	0.05%	0.23%
5 dB	1.6	Soft	0.01%	0.35%
6 dB	2.4	Hard	0%	0.63%
6 dB	1.6	Hard	0.01%	0.82%
6 dB	2.4	Soft	0.02%	0.25%
6 dB	1.6	Soft	0.01%	0.25%

Table 4.4: Table of Results, Specifically False Alarms

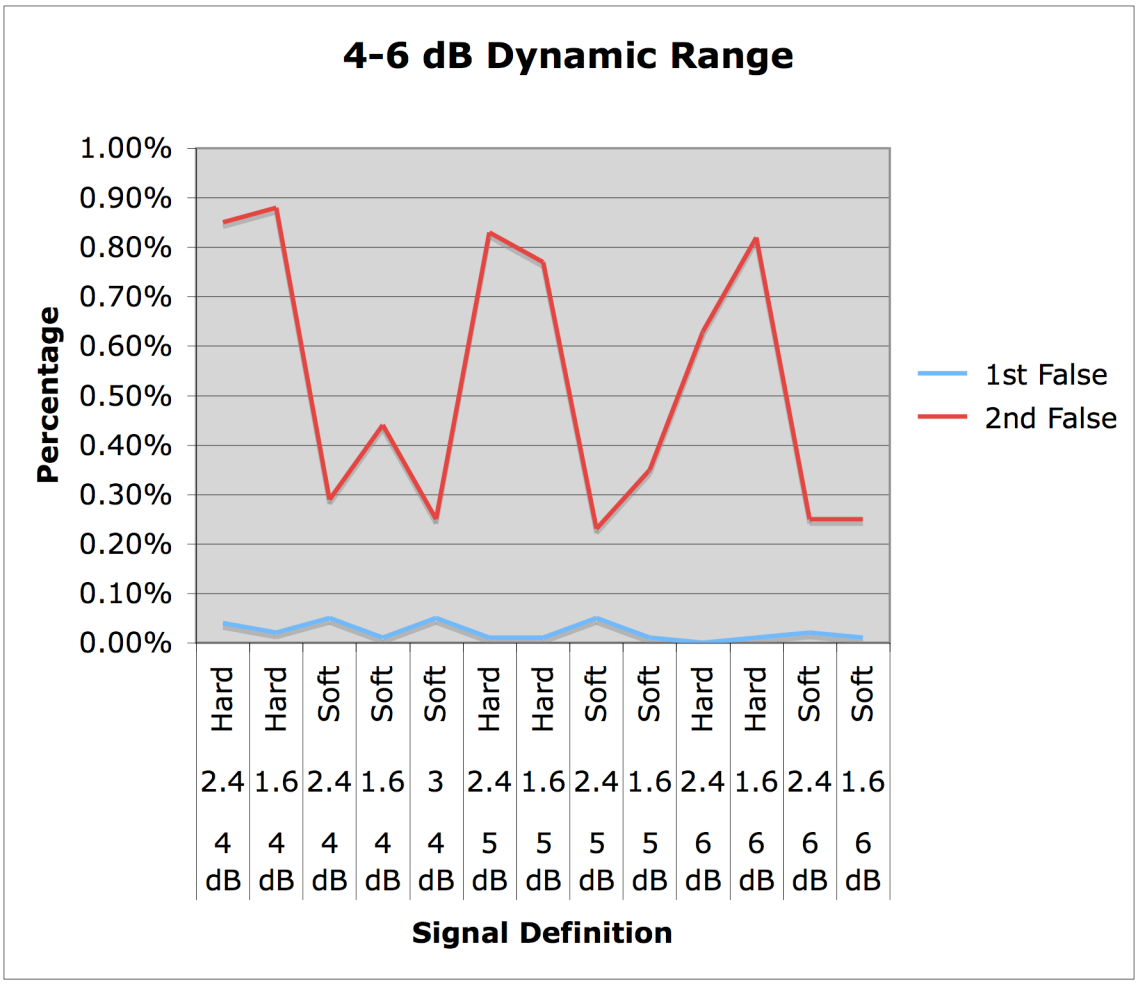

Figure 4.4: Graph of False Alarms in Mono-bit Receiver Simulations

The comparison table in Table 4.4 was graphed in Figure 4.4, which clearly shows a relation between the percentages of false alarm signals when using the de-noising function. This further narrows the threshold selection and type of threshold selection for this system, and possibly for the whole system in general. When the design performance is determined by the second signal false alarms, the general trend shown in the graph clearly denotes that the 1.6 soft threshold is the best selection for the monobit receiver design.

Purpose of experiments: To determine the performance improvement of the de-noising function in the mono-bit receiver design by examining the second missed signal.

To further investigate the usefulness of the de-noising function, a close look at the second signal missed signal rate is needed so that a deficit in performance would not be present as a result of using the module. Investigations of the second missed signal reveals the results in Table 4.5 below:

DR	Threshold	Hard/Soft	2nd Miss
4 dB	2.4	Hard	68.96%
4 dB	1.6	Hard	69.29%
4 dB	2.4	Soft	72.47%
4 dB	1.6	Soft	71.66%
4 dB	3	Soft	71.82%
5 dB	2.4	Hard	74.73%
5 dB	1.6	Hard	75.76%
5 dB	2.4	Soft	76.50%
5 dB	1.6	Soft	75.81%
6 dB	2.4	Hard	79.69%
6 dB	1.6	Hard	79.38%
6 dB	2.4	Soft	78.71%
6 dB	1.6	Soft	79.98%

Table 4.5: Table of Results, Specifically Missed Signals

While the second signal is not detected most of the time, the percentage missed seems to marginally increase with the use of the soft thresholding. Figure 4.5 shows the percentage of 2nd signal missed in various thresholds and thresholding values. It shows a trend of increasing percentages of 2nd signal missing in the monobit receiver with the increase of 2 signal dynamic range.

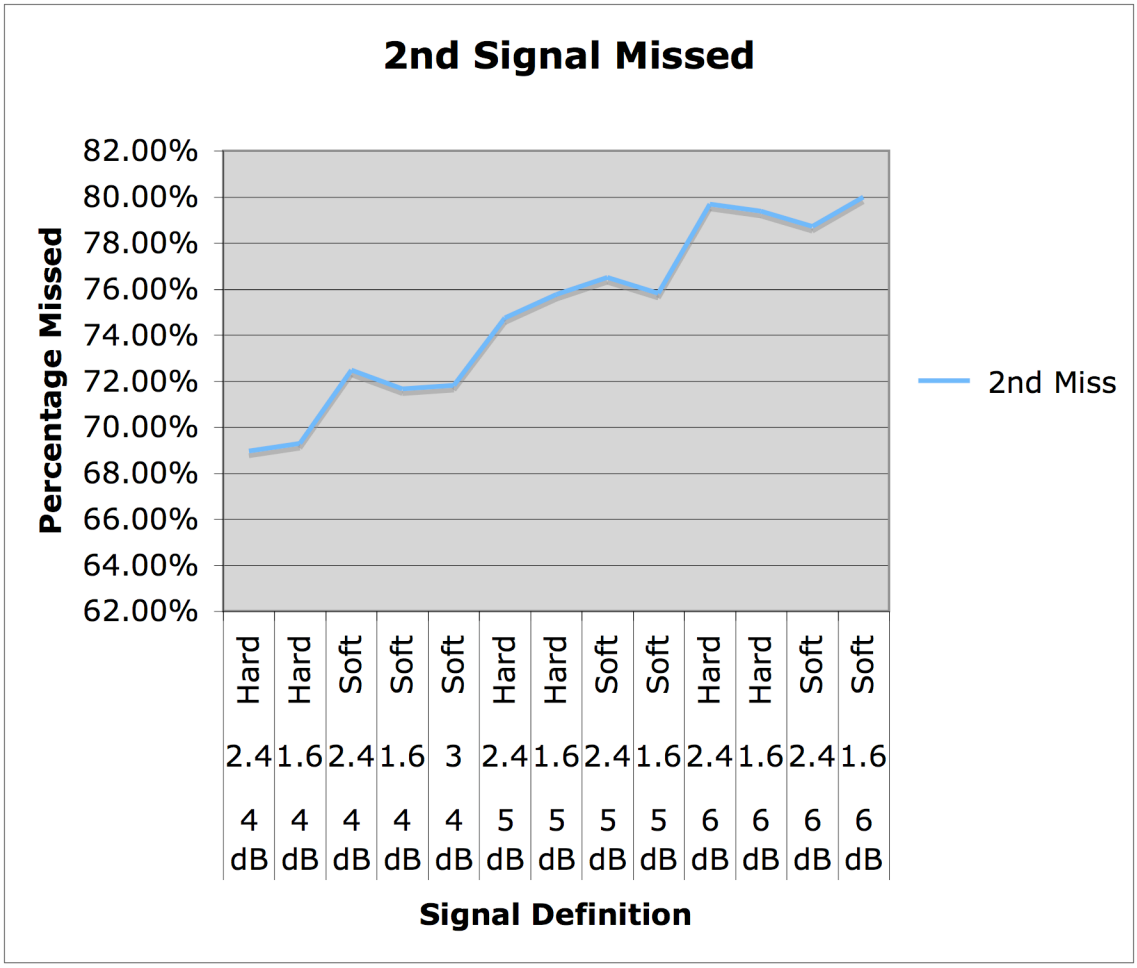

Figure 4.5: Graph of Percentage Missed vs. Threshold Type

These simulations were conducted using just one DWT de-noising function just before normalization (so there is no visible change to the signal). Some considerations would be to include a combination of hard and soft thresholding in the middle of the function. However, because the performance concerns and the space required would be effectively doubled for the including of both types of thresholds, this idea was abandoned.

4.5: Determining Performance Improvement in Receiver on a Chip

Purpose of Experiments: To determine performance improvement of integrating the de-noising function in the receiver on a chip design by applying isolated cases where the previous design fails.

After verification that the module works with the monobit receiver in improving performance, further research was done to see if previously unable to detect cases could be detected using the de-noising function integrated into the system. For this, oscilloscope-generated values were imported into the MATLAB simulation. Table 4.6 represents 9 different cases where the system is tested to see if the performance increased at 21 dB 2 signal dynamic range.

Case #	*Freq1 (MHz)*	*Freq2 (MHz)*	*Detect With Denoise?*
1	237.6334	863.8414	Yes, sometimes
2	637.8826	612.5074	No, not at all
3	149.7103	1037.336	Yes, everytime
4	1067.777	143.6644	Yes, everytime
5	129.7335	649.8135	No, not at all
6	852.8769	994.1029	Yes, everytime
7	744.9261	1089.383	Yes, sometimes
8	375.7071	863.9413	Yes, sometimes
9	247.1928	746.8173	Yes, everytime

Table 4.6: Table of Isolated Cases and Results from New Configuration

From Table 4.6, it can be seen that while all 9 cases failed for the receiver using the compensation table, sometimes or every time, the de-noising function could detect some of the cases. It further proves that the de-noising function is effective in detecting signals that the compensation matrix by itself could not.

One such case is displayed below in Figure 4.6:

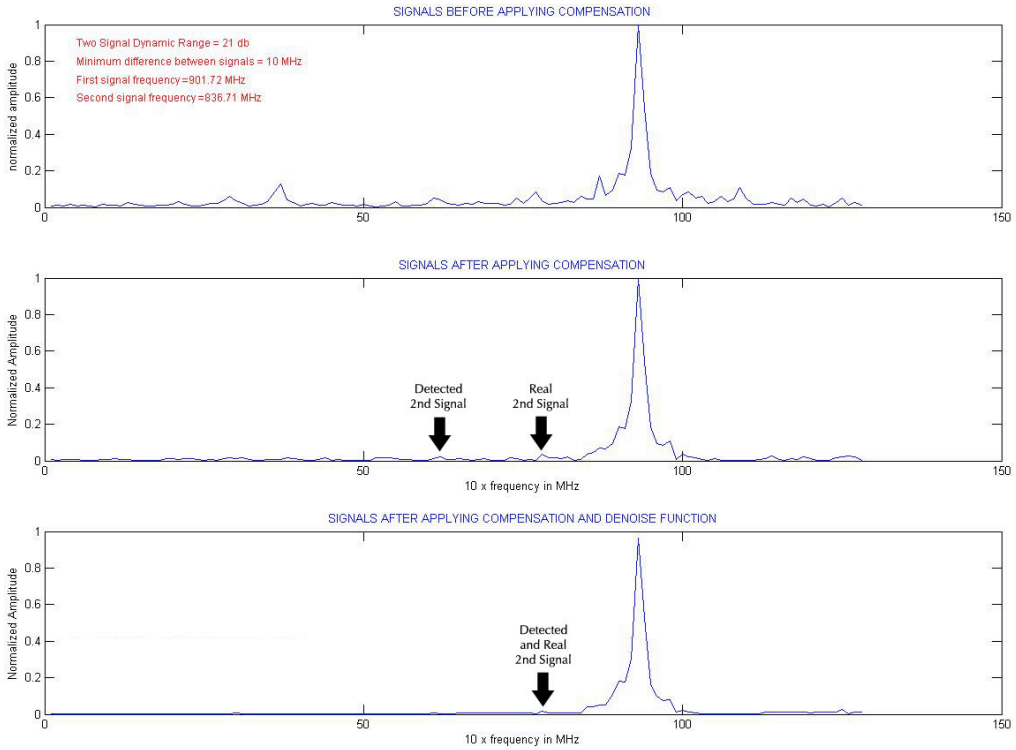

Figure 4.6: Signal Detected Using De-noising Function

In figure 4.6, the peak at the frequency bin 90 is the main signal and after compensation, the peak is removed along with its sidelobes to provide the processing for the second signal. To investigate the performance boost of the de-noising function, the second signal detection in general must be taken into consideration. For this purpose, Table 4.7 displays the results at various two-signal dynamic ranges.

Dynamic Range	Threshold	2nd Detect	2nd Missed	2nd False Alarm
18 dB	1.6 Soft	82.1%	17.1%	0.8%
18 dB	2.4 Hard	80.68%	18.92%	0.4%
19 dB	1.6 Soft	84%	14.4%	1.6%
19 dB	2.4 Hard	79%	19.5%	1.5%
20 dB	1.6 Soft	78.7%	18.2%	3.1%
21 dB	1.6 Soft	62.2%	33.4%	4.4%

Table 4.7: Simulations with De-noising Function, Specifically Second Signal

As can be seen from Table 4.7, the performance of the receiver can be pushed beyond the 18 dB range of the previous design and still meet the specifications of the design. However, once reaching beyond the 20 dB range, the number of missed signals increases beyond the specification. As can be seen from the data, however, the performance can be tweaked using either a hard or soft threshold.

In Table 4.8 the results are summed up for successful simulations of the receiver design using different components for investigation of the de-noising method as a viable source for reducing noise.

	3.5	4	4	4	5	6	7	8	9	11.5	18	19	20	20	21	21	22
2-bit ADC	✓	✓	✓	✓	✓	✓		✓	✓								
4-bit ADC							✓			✓	✓	✓	✓	✓	✓	✓	✓
12 pt. kernel 64 point FFT	✓	✓					✓	✓	✓	✓							
12 pt. kernel 256 point FFT			✓	✓	✓	✓					✓	✓	✓		✓		✓
16 pt. kernel 256 point FFT														✓		✓	
Compensation				✓			✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
DWT Denoise	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 4.8: Table of Acceptable Results with Different Configurations

4.6: Hardware Implementation

As an industry standard, VHDL was used to implement the module in a hardware description language. VHDL is a language that is used in industry for describing hardware designs. Therefore, since the signal that is being analyzed with the de-noising function is digital, VHDL is the logical choice. Synopsis tools were used to compile and analyze the VHDL code and also for synthesis. Because the industry standard is to use Synopsis for VHDL synthesis, these tools were used for implementing the system into hardware.

4.6.1: VHDL vs. AMS Verilog

AMS Verilog is a flexible language that allows top-down and down-up development techniques for both analog and mixed-signal designs. It is also flexible in that it allows different levels of abstraction for each block, which gives an optimized balance between accuracy and speed. Also, another part of AMS Verilog acts as a script, a command that creates netlists for an entire library. The netlists are created as AMS Verilog netlists, which means if there is code using other analog libraries, the analog libraries used are converted automatically and without destroying any information. Because of this ability, both analog and digital interfaces for different components in a system are accurately inserted into the design so that accuracy is maintained throughout simulation. AMS Verilog also allows for both text and schematic data entry, making either form of entering information readily available.

The environment for entering the AMS Verilog design allows for debugging the mixed signal, analog and digital signals in the system, while also allowing for stand-alone simulation. Because AMS Verilog has so many advantages, mostly based on flexibility, the AMS Verilog language and design environment is very attractive for implementation uses. AMS Verilog was used in the implementation of the ROC, and has direct interface with the de-noising function, which is described next.

4.6.2: Hardware Implementation of De-noising Module

For the hardware implementation of the module, a behavioural code, based on the functional C++ code and the MATLAB interface was investigated. Because VHDL implements vectors and arrays differently than C++ and is more rigid as a result (only 1 dimensional arrays can be declared and used), a more structural design was pursued, but still maintaining the behavioural design for flexibility for working with different types.

The design of the de-noising function is shown as programmed in C++ and MATLAB in Figure 4.7:

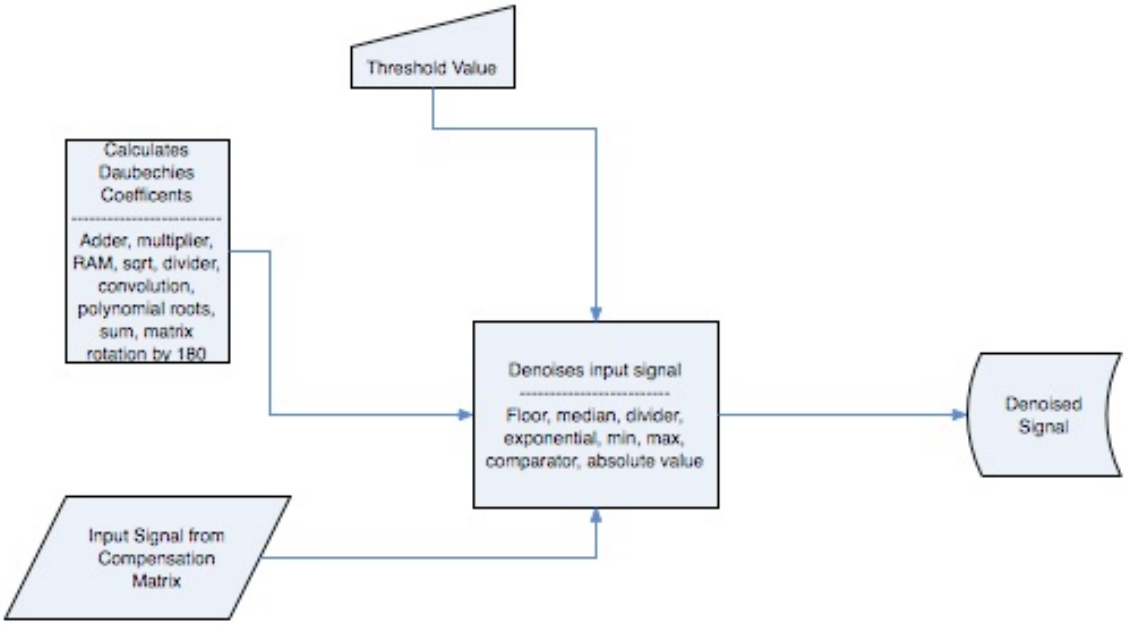

Figure 4.7: De-noising Function Interfaces

The diagram in Figure 4.7 shows the different functions that are implemented outside of the de-noising function, as well as some of the internal functions of each of the different modules. Most notable are the Daubechie's coefficient calculations of the filter coefficients, which includes adders, multipliers, memory, and some other mathematical operations. These operations, however, are redundant in this design as only one set of coefficients are generated for this application, so instead of using the mathematical operations, the coefficients are stored in memory as an alternative. Further examination of the de-noising function code shows a design flow as shown below:

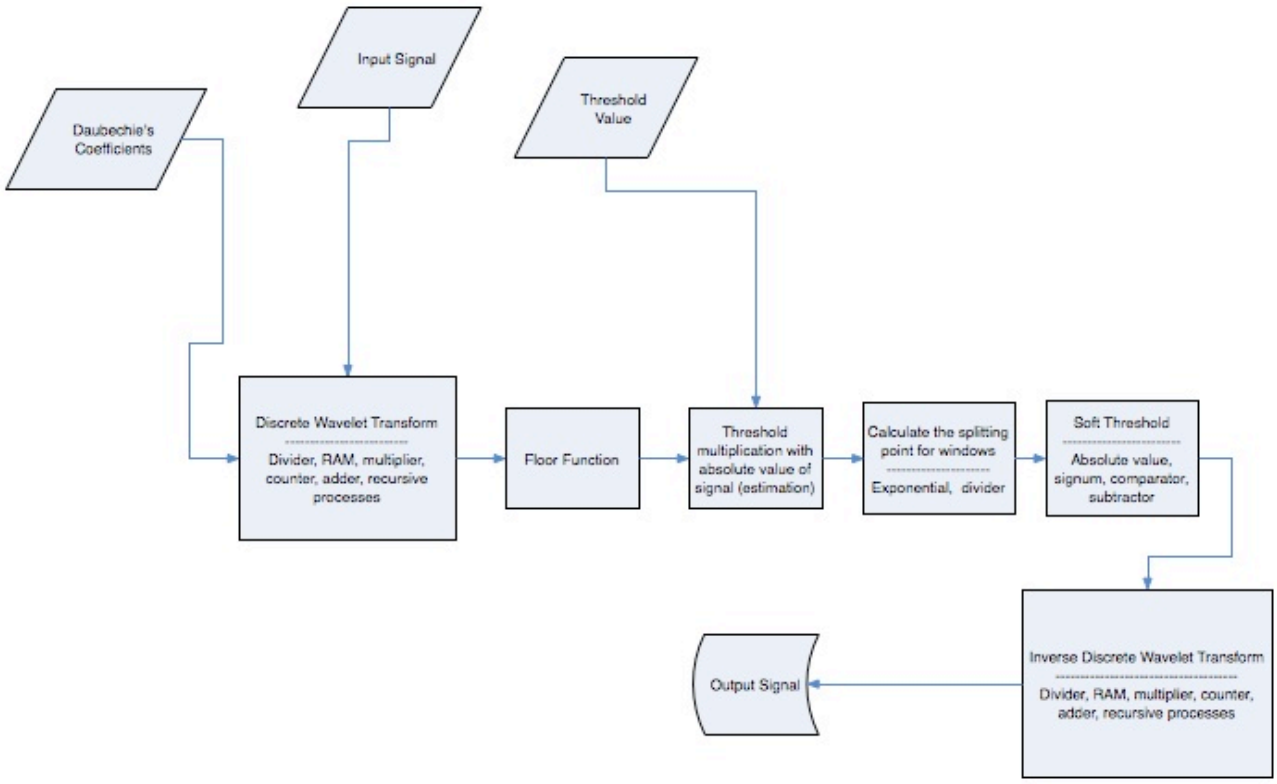

Figure 4.8: De-noising Function Details

The de-noising function is broken down into several blocks, each containing different elements that would require multiplication, memory and have some recursive processes. These processes will require many resources, especially space in terms of multiplication and memory. To design the de-noising function effectively, considerations of processing intensity and space requirement are important factors in the design. The de-noising function, then, needs to be considered from the basic building blocks of the system and to consider different alternative implementations of each of the different blocks.

The de-noising function is broken up in to 4 separate blocks, as shown below:

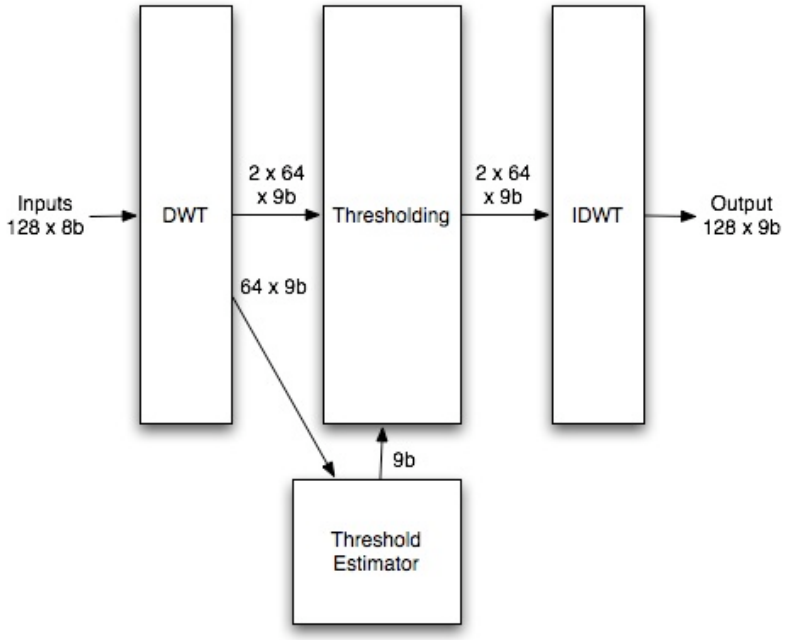

Figure 4.9: De-noising Function Block Diagram

The inputs to the discrete wavelet transform is 128 element set of 8 bit data from the compensation table. Each element of the data is assumed to be a positive number because of normalization from the compensation block. However, after a discrete wavelet transform, there are some coefficients that will be negative because of the filter coefficient values used. For negative numbers, the numbers are represented by 2's complement scheme, so the output of the discrete wavelet transform requires 9 bits.

After the discrete wavelet transform, there is a high-pass and a low-pass breakdown of the signal. Both the high-pass and the low-pass signals are transmitted to the thresholding block, where selection of signal removal takes place. In addition, the low-pass component also passes into the threshold estimator unit, which finds the median of the low-pass signal elements and

multiplies that value with a separate coefficient. This allows the thresholding value to be determined for the best possible cut. This value is then passed into the thresholding unit.

Once the signal has been processed through the thresholding unit, the high-pass and low-pass signals are passed into the inverse discrete wavelet transform to bring the signal back from the wavelet domain to the frequency domain. From the inverse discrete wavelet transform, the signal is then sent to the frequency selection, which can be easily changed to accommodate the 9 bit outcome of the de-noising function.

4.6.2.1: Look Up Table Implementation

There are several methods of hardware implementation of the discrete wavelet transform and it's inverse, as discussed in Chapter 2; however, all the implementations require repeated multiplications in the process. To alleviate any use of multipliers, look up tables can be used to accept the 8 bit value as an address, then output the 9 bit value at the location as the result. This 9 bit value is the calculated coefficient multiplied times every possible input combination, each stored in the address that corresponds with the 8 bit combination. Because look up tables are simpler than multipliers and take up less space, the use of look up tables for the discrete wavelet transform and it's inverse is used in this design.

For the implementation of the inverse discrete transform, look up tables are also used, but in order to access the negative numbers (which are represented with 2's complement) in the table, the order of the table has to be changed to

match the address that is coming into the look up table, as the look up table only accesses addresses as unsigned numbers. For example, the number -5 is "111111011" in 2's complement. When this value is read by the look up table, it will not consider the number as a 2's complement, but rather as an unsigned value, which means the actual value that the look up table interprets the address as is 507. This means that the value that is at the address 507 must have the value -5 multiplied times the coefficient stored in that location. Since the value "100000000" represents the value of negative 0 in 2's complement and is considered 256 in the look up table, this value is stored at position 256. The other values, however, appear in reverse order, from -255 down to -1, at look up table addresses 257 to 511. A diagram of this setup is shown in Figure 4.10.

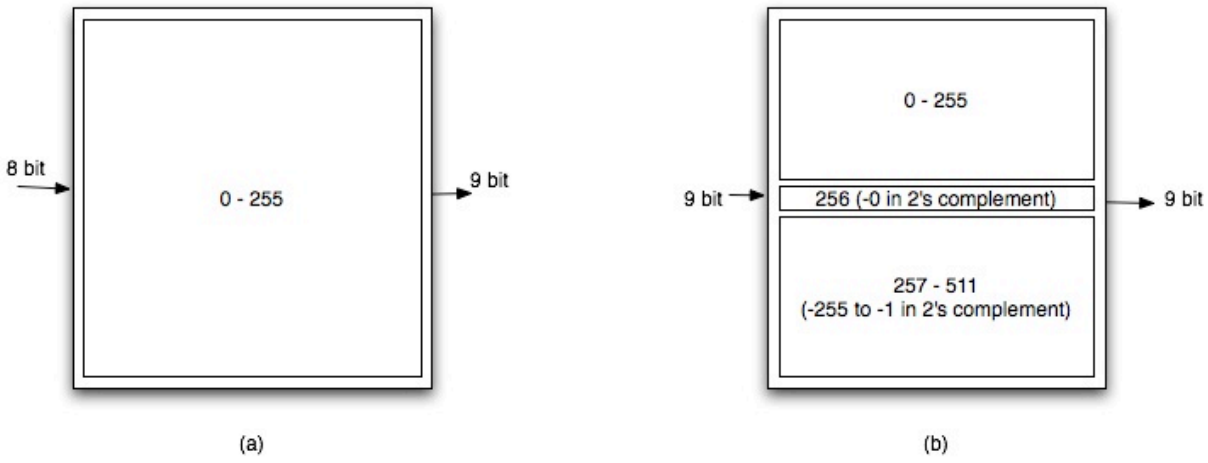

Figure 4.10: Look Up Table Configuration a) DWT LUT, b) IDWT LUT

4.6.2.2: Decimal Representation in Binary

The numbers represented in the system are normalized numbers, meaning the highest number is 1, being the first signal peak. All the other values are below 1 in amplitude: there is a requirement, then, for a scheme for representing

the values in the 8 bit binary for the signal data. The scheme used for the compensation matrix number representation is the biased number representation. For example, the number 0.056202 can be represented with the binary number 00001110 with a bias of 8. The number 00001110 represents the value ($2^1+2^2+2^3$), but with a bias of 8, the formula becomes :

$$
\begin{aligned}
&00001110 = \\
&(2^1 + 2^2 + 2^3) \times 2^{-8} = \\
&(2^{-7} + 2^{-6} + 2^{-5}) = 0.0546875 \\
&0.056202 \approx 0.0546875
\end{aligned}
$$

With this representation scheme, any bias of any number can be used to represent small numbers of this type. The greater the number of biases, the more likely the number will match the intended number. However, the cost of using more bits in a signal is directly proportional to this representation. For the use of this denoising function, a bias of 8 is chosen for the inputs, as the inputs to the denoising function are the outputs from the compensation matrix, which have a bias of 8. For within the look up tables, the values are generated using C code, which utilizes the values from 0 to 255 (all the possible values from the 8 bit input) and multiplies those values with the coefficient required. However, to get the binary representation of the coefficient without hand calculation, the following process was used:

$$
\begin{aligned}
&h_3 = 0.2241 \\
&0.2241 \times 2^8 = 57.3696 \\
&57.3696 \approx 57 = (00111001)_{binary}
\end{aligned}
$$

This process takes the coefficient and multiplies it with 256 or 2^8 (which is the bias) and rounds off the result to the nearest integer. This is the approximate binary value of the number using a bias of 8. Throughout the denoising process, the bias of 8 was used to maintain simplicity in the processing circuitry.

4.6.2.3: Look Up Table Configuration

There are several coefficients required for the transform process. Four coefficients are from the discrete wavelet transform based on the Daubechie's wavelet theory. These coefficients are 0.483, 0.8365, 0.2241, and -0.1294, and are referred to as h_1, h_2, h_3, and h_4, respectively. As previously mentioned, the representation of the numbers uses a bias of 8. However, since h_4 is a negative coefficient, the system requires the use of 2's complement, and thus requires 9 bits for processing within the denoising function. For the inverse transform, there are 6 tables required to be created. Four look up tables are represent the same coefficients as the discrete wavelet transform, but have an addition of 2's complement values. Another two look up tables contain the negative coefficients of h_1 and h_3 for reconstruction, as well as the addition of positive values for any positive value inputs. There is also a look up table for thresholding because there is a coefficient of 1.6/0.67 or 2.388 that is multiplied with the median of the incoming 64 samples from the low-pass signal of the DWT. This provides a viable value to compare the samples from the transform to for cutting. The VHDL code for the different blocks are included in Appendix B.

4.6.2.4: Simulation

To test the functionality of the VHDL, the same inputs for the MATLAB are put in to the system to see the differentiation from the MATLAB due to bit truncation. This bit truncation is due to the default bias of 8 in the numbers on the output of the compensation matrix. Representing the numbers in binary is similar to how the coefficients are represented in binary. The numbers can be quickly generated to provide a speedier way to provide a relatively accurate representation of the number through the same process as how coefficients are represented in binary. VHDL code can be quickly generated, especially if it is redundant code, through the use of a program written in C. This program uses file input and output commands to generate the code, and is included in Appendix C.

When the data is input into the de-noising function, the peak of the data is noted first. For this simulation, the second signal that is generated is at 700 MHz, which corresponds with the 72nd frequency bin. The peak in the data, therefore appears at the 72nd sample. From Figure 4.11 shows below, it is shown that the peak value from the VHDL implementation of the DWT based de-noising function is at frequency bin 72, which matches the input data. However, the spurs in the signal are reduced to very low values due to the thresholding unit in the de-noising function implemented in the VHDL. The threshold value that was implemented in the LUT was 1.6, which was observed to be the best overall threshold value for this application.

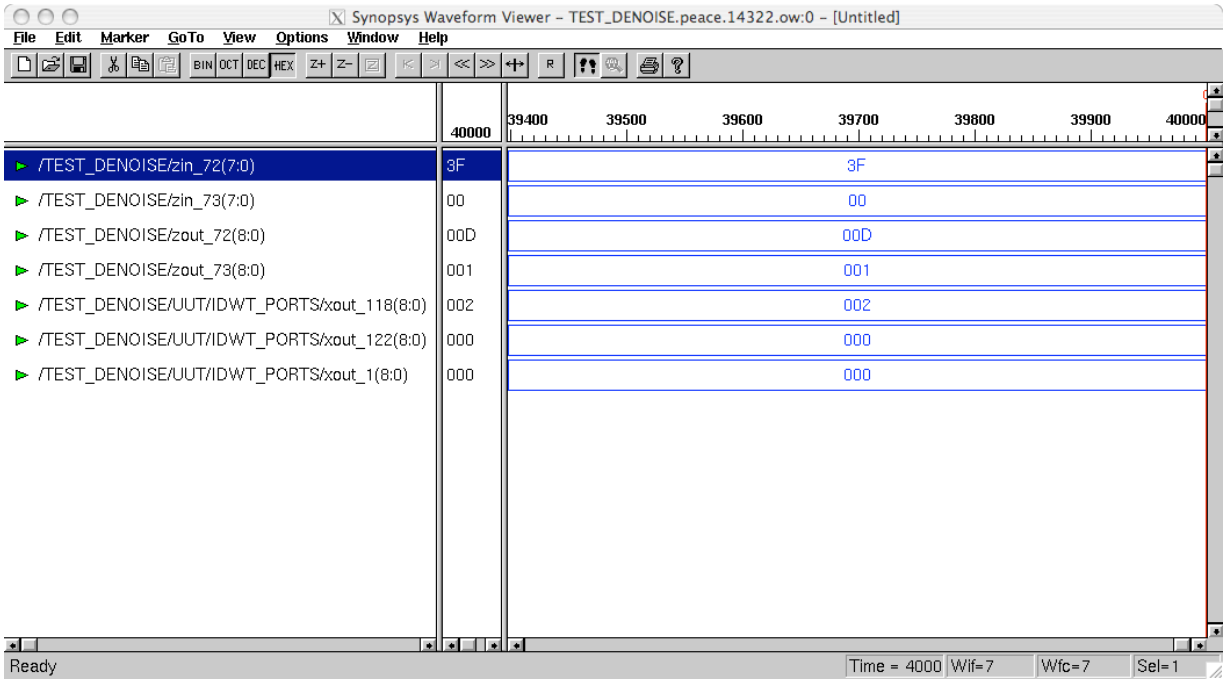

Figure 4.11: De-noising Function in VHDL Simulation

From this simulation, it can be seen that the de-noising module is effectively implemented in VHDL, which can be used for hardware simulation and implementation on a chip or an FPGA. In terms of hardware cost, the number of LUTs along with the sizes of each, the size and number of adders, and the size and number of muxes are the important features to consider. The component breakdown can be seen in Table 4.9:

Module/Component	*Number*	*Size*
DWT:		
LUT (8-b in/ 9-b out)	4	256 x 9-b
Adder (8-b)	256	
Adder (9-b)	128	
Threshold Estimator:		
Mux (9-b 2x1)	1	
Comparators	128	
Thresholding:		
LUT (9-b in/ 9-b out)	1	512 x 9-b
Mux (9-b 2x1)	128	
IDWT:		
LUT (9-b in/ 9-b out)	6	512 x 9-b
Adder (9-b)	256	
Adder (10-b)	128	
Total Component Costs:		
LUT (8-b in/ 9-b out)	4	256 x 9-b
Adder (8-b)	256	
Adder (9-b)	384	
Adder (10-b)	128	
LUT (9-b in/ 9-b out)	7	512 x 9-b
Mux (9-b 2x1)	129	
Comparators	128	

Table 4.9 : Component Requirements for De-noising Function

Storage for the values while processing the different elements are also required, but can be allocated dynamically with the memory that is allocated for the compensation matrix operations. The hardware requirements for the de-noising function can easily be implemented using pipeline structures in ASIC. Furthermore, the LUTs can be implemented efficiently in an FPGA configuration, which would further reduce the hardware implementation requirements so that the actual hardware cost of the de-noising function in relation to the receiver on chip design is relatively small, especially when considering the size of the FFT and compensation matrix.

Chapter 5: Conclusion and Recommendations

5.1: Conclusion

For effective signal processing that performs beyond a receiver on chip design with a compensation table, the discrete wavelet transform-based de-noising function is a valid solution. The discrete wavelet transform breaks down the incoming signal into high pass and low pass components. Low pass components are considered small details while high pass components are considered large details. Because spurious signals are considered small details in a discrete wavelet transform domain, thresholding is required to remove the spurious signals. When the thresholding is completed, the inverse transform reassembles the signal to the original signal domain. For the receiver on chip design, the de-noising function is effective for improving performance of the receiver for detecting the second signal from two-signal dynamic ranges of 18 dB to 22 dB. This thesis describes the investigation and implementation of a de-noising module that utilizes a configuration that is effective for increasing the two signal dynamic range beyond 18 dB. By using look up tables, the de-noising function is implemented in VHDL in an efficient design that eliminates the need for multipliers and thus, reduces the hardware cost of the module.

5.2: Recommendations

I recommend an implementation of the discrete wavelet transform with greater number of bits for better precision and for the design to further improve

performance of the receiver on chip design. Because the maximum number of bits for the coefficients to have perfect precision is 19 bits, I would recommend that a 19 bit design be pursued to have a complete design. The only drawback to this design would be that the increase in bits will require a larger space for the LUTs because the values required will be stored as 256 values, each being 19 bits. For negative numbers, there will be 512 values, each being 19 bits.

I also recommend implementation of this design in ASIC to investigate the speed benefits of ASIC design and to also investigate new designs for different registers, such as double-edge trigger D-flip flops, new ADC designs, and possibly new compensation matrix calculations. It would be interesting to see what effect the newer components in the receiver design would have on the de-noising process.

References:

[1] J. Proakis and D. Manolakis. "Digital Signal Processing: Principles, Algorithms, and Applications, Third Edition" New York: Prentice Hall, 1998.

[2] D. Pok, C.-I. H. Chen, J. Schamus, C. Montgomery, and J. B. Y. Tsui, "Chip design for monobit receiver," IEEE Trans. Microwave Theory Tech., Vol. 45, No. 12, pp 2283-2295, December 1997.

[3] C. H. Chen, and K. George. "Design and Performance Evaluation of a 2.5-GSPS Digital Receiver." IEEE Transactions on Instrumentation and Measurement. Vol. 54, No. 3, pp 1089-1099, June 2005.

[4] J. Särelä and H. Valpola. "De-noising Source Separation." Journal of Machine Learning Research 6. (March 2005): 233-272.

[5] G. Dimitroulakos, M. D. Galanis, A. Milidonis, and C. E. Goutis, "A high-throughput, memory efficient architecture for computing the tile-based 2D discrete wavelet transform for the JPEG2000." Integration, Journal 39, pp 1-11, 2005.

[6] A. Jensen, and A. la Cour-Harbo. Ripples in Mathematics: The Discrete Wavelet Transform. New York: Springer-Verlag, 2001.

[7] Y. Nievergelt. Wavelets Made Easy. Boston: Birkhäuser, 2001.

[8] D. Walnut. An Introduction to Wavelet Analysis. Boston: Birkhäuser, 2002.

[9] M. Jansen. Noise Reduction by Wavelet Thresholding. New York: Springer-Verlag, 2001.

[10] P. Vaidyanathan. "Multirate Systems and Filter Banks." New York: Prentice Hall, 1993.

[11] V. Herrero, J. Cerdà, R. Gadea, M. Martinez, and A. Sebastià. "Implementation of 1-D Daubechies Wavelet Transform on FPGA."

[12] G. Kuzmanov and B. Zafarifar. "Reconfigurable DWT Unit Based on Lifting."

[13] C. Huang, P. Tseng, and L. Chen. "Analysis and VLSI Architecture for 1-D and 2-D Discrete Wavelet Transform." IEEE Transactions on Signal Processing, Vol. 53, No. 4, pp. 1575 – 1586, April 2005.

[14] M. Nibourche, A. Bouridane, F. Murtagh, and O. Nibouche. "FPGA-Based Discrete Wavelet Transforms System."

[15] B.-F. Wu and Y.-Q. Hu. "An Efficient VLSI Implementation of the Discrete Wavelet Transform Using Embedded Instruction Codes for Symmetric Filters." IEEE Transactions on Circuits and Systems for Video Technology, Vol. 13, No. 9, pp. 936 – 943, September 2003.

[16] F. Maire. "Low Power Implementation for a Class of Orthogonal Wavelet Transform Using Synthesizable VHDL." Master of Science Thesis in Electronic System Design, University of Stockholm, Stockholm, Norway, September 1999.

[17] C.-T. Huang, P.-C. Tseng and L.-G. Chen. "Flipping Structure: An Efficient VLSI Architecture for Lifting-Based Discrete Wavelet Transform." IEEE Transactions on Signal Processing, Vol. 52, No. 4, pp. 1080 – 1089, April 2004.

Appendix A:

Denoising Function MATLAB Code:

```
function [xd,xn,option] = denoise(x,h,type,option)
%   [xd,xn,option] = denoise(x,h,type,option);
%
%   DENOISE is a generic program for wavelet based denoising.
%   The program will denoise the signal x using the 2-band wavelet
%   system described by the filter h using either the traditional
%   discrete wavelet transform (DWT) or the linear shift invariant
%   discrete wavelet transform (also known as the undecimated DWT
%   (UDWT)).
%
%   Input:
%     x       : 1D or 2D signal to be denoised
%     h       : Scaling filter to be applied
%     type     : Type of transform (Default: type = 0)
%               0 --> Discrete wavelet transform (DWT)
%               1 --> Undecimated DWT (UDWT)
%     option    : Default settings is marked with '*':
%               *type = 0 --> option = [0 3.0 0 0 0 0]
%               type = 1 --> option = [0 3.6 0 1 0 0]
%     option(1) : Whether to threshold low-pass part
%               0 --> Don't threshold low pass component
%               1 --> Threshold low pass component
%     option(2) : Threshold multiplier, c. The threshold is
%               computed as:
%                 thld = c*MAD(noise_estimate)).
%               The default values are:
%                 c = 3.0 for the DWT based denoising
%                 c = 3.6 for the UDWT based denoising
%     option(3) : Type of variance estimator
%               0 --> MAD (mean absolute deviation)
%               1 --> STD (classical numerical std estimate)
%     option(4) : Type of thresholding
%               0 --> Soft thresholding
%               1 --> Hard thresholding
%     option(5) : Number of levels, L, in wavelet decomposition. By
%               setting this to the default value '0' a maximal
%               decomposition is used.
%     option(6) : Actual threshold to use (setting this to
%               anything but 0 will mean that option(3)
%               is ignored)
%
%   Output:
%     xd    : Estimate of noise free signal
%     xn    : The estimated noise signal (x-xd)
%     option : A vector of actual parameters used by the
%            program. The vector is configured the same way as
%            the input option vector with one added element
%            option(7) = type.

%File Name: denoise.m
```

```
%Last Modification Date: 04/15/97        10:44:28
%Current Version: denoise.m     2.4
%File Creation Date: Mon Feb 20 08:33:15 1995
%Author: Jan Erik Odegard  <odegard@ece.rice.edu>
%
%Copyright (c) 2000 RICE UNIVERSITY. All rights reserved.
%Created by Jan Erik Odegard, Department of ECE, Rice University.
%
%This software is distributed and licensed to you on a non-exclusive
%basis, free-of-charge. Redistribution and use in source and binary forms,
%with or without modification, are permitted.

if(nargin < 2)
  error('You need to provide at least 2 inputs: x and h');
end;
if(nargin < 3),
  type = 0;
  option = [];
elseif(nargin < 4)
  option = [];
end;
if(isempty(type)),
  type = 0;
end;
if(type == 0),
  default_opt = [0 3.0 0 0 0 0];
elseif(type == 1),
  default_opt = [0 3.6 0 1 0 0];
else,
  error(['Unknown denoising method',10,...
              'If it is any good we need to have a serious talk :-)']);
end;
option = setopt(option,default_opt);
[mx,nx] = size(x);
dim = min(mx,nx);
if(dim == 1),
  n = max(mx,nx);
else,
  n = dim;
end;
if(option(5) == 0),
  L = floor(log2(n));
else
  L = option(5);
end;
if(type == 0),                          % Denoising by DWT
  xd = mdwt(x,h,L);
  if (option(6) == 0),
    tmp = xd(floor(mx/2)+1:mx,floor(nx/2)+1:nx);
    if(option(3) == 0),
      thld = option(2)*median(abs(tmp(:)))/.67;
    elseif(option(3) == 1),
      thld = option(2)*std(tmp(:));
```

```
  else
    error('Unknown threshold estimator, Use either MAD or STD');
  end;
 else,
  thld = option(6);
 end;
 if(dim == 1)
  ix = 1:n/(2^L);
  ykeep = xd(ix);
 else
  ix = 1:mx/(2^L);
  jx = 1:nx/(2^L);
  ykeep = xd(ix,jx);
 end;
 if(option(4) == 0),
  xd = SoftTh(xd,thld);
 elseif(option(4) == 1),
  xd = HardTh(xd,thld);
 else,
  error('Unknown threshold rule. Use either Soft (0) or Hard (1)');
 end;
 if (option(1) == 0),
  if(dim == 1),
    xd(ix) = ykeep;
  else,
    xd(ix,jx) = ykeep;
  end;
 end;
 xd = midwt(xd,h,L);
elseif(type == 1),                          % Denoising by UDWT
 [xl,xh] = mrdwt(x,h,L);
 if(dim == 1),
  c_offset = 1;
 else,
  c_offset = 2*nx + 1;
 end;
 if (option(6) == 0),
  tmp = xh(:,c_offset:c_offset+nx-1);
  if(option(3) == 0),
    thld = option(2)*median(abs(tmp(:)))/.67;
  elseif(option(3) == 1),
    thld = option(2)*std(tmp(:));
  else
    error('Unknown threshold estimator, Use either MAD or STD');
  end;
 else,
  thld = option(6);
 end;
 if(option(4) == 0),
  xh = SoftTh(xh,thld);
  if(option(1) == 1),
    xl = SoftTh(xl,thld);
  end;
```

```
 elseif(option(4) == 1),
  xh = HardTh(xh,thld);
  if(option(1) == 1),
   xl = HardTh(xl,thld);
  end;
 else,
  error('Unknown threshold rule. Use either Soft (0) or Hard (1)');
 end;
 xd = mirdwt(xl,xh,h,L);
else,                                          % Denoising by unknown method
 error(['Unknown denoising method',10,...
       'If it is any good we need to have a serious talk :-)']);
end;
option(6) = thld;
option(7) = type;
xn = x - xd;
```

Daubechie's Wavelet Coefficient Calculations:

```
function [h_0,h_1] = daubcqf(N,TYPE)
%    [h_0,h_1] = daubcqf(N,TYPE);
%
%    Function computes the Daubechies' scaling and wavelet filters
%    (normalized to sqrt(2)).
%
%    Input:
%       N    : Length of filter (must be even)
%       TYPE : Optional parameter that distinguishes the minimum phase,
%              maximum phase and mid-phase solutions ('min', 'max', or
%              'mid'). If no argument is specified, the minimum phase
%              solution is used.
%
%    Output:
%       h_0 : Minimal phase Daubechies' scaling filter
%       h_1 : Minimal phase Daubechies' wavelet filter

%    Reference: "Orthonormal Bases of Compactly Supported Wavelets",
%               CPAM, Oct.89
%
%File Name: daubcqf.m
%Last Modification Date: 01/02/96        15:12:57
%Current Version: daubcqf.m    2.4
%File Creation Date: 10/10/88
%Author: Ramesh Gopinath  <ramesh@dsp.rice.edu>
%
%Copyright (c) 2000 RICE UNIVERSITY. All rights reserved.
%Created by Ramesh Gopinath, Department of ECE, Rice University.
%
if(nargin < 2),
  TYPE = 'min';
end;
if(rem(N,2) ~= 0),
  error('No Daubechies filter exists for ODD length');
end;
```

```
K = N/2;
a = 1;
p = 1;
q = 1;
h_0 = [1 1];
for j  = 1:K-1,
  a = -a * 0.25 * (j + K - 1)/j;
  h_0 = [0 h_0] + [h_0 0];
  p = [0 -p] + [p 0];
  p = [0 -p] + [p 0];
  q = [0 q 0] + a*p;
end;
q = sort(roots(q));
qt = q(1:K-1);
if TYPE=='mid',
  if rem(K,2)==1,
    qt = q([1:4:N-2 2:4:N-2]);
  else
    qt = q([1 4:4:K-1 5:4:K-1 N-3:-4:K N-4:-4:K]);
  end;
end;
h_0 = conv(h_0,real(poly(qt)));
h_0 = sqrt(2)*h_0/sum(h_0);       %Normalize to sqrt(2);
if(TYPE=='max'),
  h_0 = fliplr(h_0);
end;
if(abs(sum(h_0 .^ 2))-1 > 1e-4)
  error('Numerically unstable for this value of "N".');
end;
h_1 = rot90(h_0,2);
h_1(1:2:N)=-h_1(1:2:N);
```

Discrete Wavelet Transform C++ Code:

```
/*
File Name: MDWT.c
Last Modification Date:  06/14/95          13:15:44
Current Version: MDWT.c          2.4
File Creation Date: Wed Oct 19 10:51:58 1994
Author: Markus Lang  <lang@jazz.rice.edu>

Copyright (c) 2000 RICE UNIVERSITY. All rights reserved.
Created by Markus Lang, Department of ECE, Rice University.

%y = mdwt(x,h,L);
%
% function computes the discrete wavelet transform y for a 1D or 2D input
% signal x.
%
%    Input:
%          x    : finite length 1D or 2D signal (implicitely periodized)
%       h    : scaling filter
%       L    : number of levels. in case of a 1D signal length(x) must be
%              divisible by 2^L; in case of a 2D signal the row and the
```

```
%            column dimension must be divisible by 2^L.
%
% see also: midwt, mrdwt, mirdwt
*/

#include <math.h>
#include <stdio.h>

#define max(A,B) (A > B ? A : B)
#define mat(a, i, j) (*(a + (m*(j)+i)))  /* macro for matrix indices */

#ifdef __STDC__
MDWT(double *x, int m, int n, double *h, int lh, int L, double *y)
#else
MDWT(x, m, n, h, lh, L, y)
double *x, *h, *y;
int m, n, lh, L;
#endif
{
 double  *h0, *h1, *ydummyl, *ydummyh, *xdummy;
 long i, j;
 int actual_L, actual_m, actual_n, r_o_a, c_o_a, ir, ic, lhm1;
 xdummy = (double *)mxCalloc(max(m,n)+lh-1,sizeof(double));
 ydummyl = (double *)mxCalloc(max(m,n),sizeof(double));
 ydummyh = (double *)mxCalloc(max(m,n),sizeof(double));
 h0 = (double *)mxCalloc(lh,sizeof(double));
 h1 = (double *)mxCalloc(lh,sizeof(double));


 /* analysis lowpass and highpass */
 if (n==1){
  n = m;
  m = 1;
 }
 for (i=0; i<lh; i++){
  h0[i] = h[lh-i-1];
  h1[i] =h[i];
 }
 for (i=0; i<lh; i+=2)
  h1[i] = -h1[i];

 lhm1 = lh - 1;
 actual_m = 2*m;
 actual_n = 2*n;

 /* main loop */
 for (actual_L=1; actual_L <= L; actual_L++){
  if (m==1)
   actual_m = 1;
  else{
   actual_m = actual_m/2;
   r_o_a = actual_m/2;
  }
```

```
   actual_n = actual_n/2;
   c_o_a = actual_n/2;

   /* go by rows */
   for (ir=0; ir<actual_m; ir++){          /* loop over rows */
    /* store in dummy variable */
    for (i=0; i<actual_n; i++)
          if (actual_L==1)
            xdummy[i] = mat(x, ir, i);
          else
            xdummy[i] = mat(y, ir, i);
    /* perform filtering lowpass and highpass*/
    fpsconv(xdummy, actual_n, h0, h1, lhm1, ydummyl, ydummyh);
    /* restore dummy variables in matrices */
    ic = c_o_a;
    for  (i=0; i<c_o_a; i++){
          mat(y, ir, i) = ydummyl[i];
          mat(y, ir, ic++) = ydummyh[i];
    }
   }

   /* go by columns in case of a 2D signal*/
   if (m>1){
    for (ic=0; ic<actual_n; ic++){          /* loop over column */
          /* store in dummy variables */
          for (i=0; i<actual_m; i++)
            xdummy[i] = mat(y, i, ic);
          /* perform filtering lowpass and highpass*/
          fpsconv(xdummy, actual_m, h0, h1, lhm1, ydummyl, ydummyh);
          /* restore dummy variables in matrix */
          ir = r_o_a;
          for (i=0; i<r_o_a; i++){
            mat(y, i, ic) = ydummyl[i];
            mat(y, ir++, ic) = ydummyh[i];
          }
    }
   }
  }
}

#ifdef __STDC__
fpsconv(double *x_in, int lx, double *h0, double *h1, int lhm1,
          double *x_outl, double *x_outh)
#else
fpsconv(x_in, lx, h0, h1, lhm1, x_outl, x_outh)
double *x_in, *h0, *h1, *x_outl, *x_outh;
int lx, lhm1;
#endif

{
  int i, j, ind;
  double x0, x1;
```

```
 for (i=lx; i < lx+lhm1; i++)
   x_in[i] = *(x_in+(i-lx));
 ind = 0;
 for (i=0; i<(lx); i+=2){
   x0 = 0;
   x1 = 0;
   for (j=0; j<=lhm1; j++){
     x0 = x0 + x_in[i+j]*h0[lhm1-j];
     x1 = x1 + x_in[i+j]*h1[lhm1-j];
   }
   x_outl[ind] = x0;
   x_outh[ind++] = x1;
 }
}
```

Inverse DWT C++ Code:

```
/*
File Name: MIDWT.c
Last Modification Date:  06/14/95          13:01:15
Current Version: MIDWT.c         2.4
File Creation Date: Wed Oct 12 08:44:43 1994
Author: Markus Lang  <lang@jazz.rice.edu>

Copyright (c) 2000 RICE UNIVERSITY. All rights reserved.
Created by Markus Lang, Department of ECE, Rice University.

description of the matlab call:
%y = midwt(x,h,L);
%
% function computes the inverse discrete wavelet transform y for a 1D or 2D
% input signal x.
%
%    Input:
%          x    : finite length 1D or 2D input signal (implicitely periodized)
%       h    : scaling filter
%       L    : number of levels. in case of a 1D signal length(x) must be
%              divisible by 2^L; in case of a 2D signal the row and the
%              column dimension must be divisible by 2^L.
%
% see also: mdwt, mrdwt, mirdwt*/

#include <math.h>
#include <stdio.h>

#define max(A,B) (A > B ? A : B)
#define mat(a, i, j) (*(a + (m*(j)+i)))  /* macro for matrix indices */


#ifdef __STDC__
MIDWT(double *x, int m, int n, double *h, int lh, int L, double *y)
#else
MIDWT(x, m, n, h, lh, L, y)
double *x, *h, *y;
```

```
int m, n, lh, L;
#endif
{
 double  *g0, *g1, *ydummyl, *ydummyh, *xdummy;
 long i, j;
 int actual_L, actual_m, actual_n, r_o_a, c_o_a, ir, ic, lhm1, lhhm1, sample_f;
 xdummy = (double *)mxCalloc(max(m,n),sizeof(double));
 ydummyl = (double *)mxCalloc(max(m,n)+lh/2-1,sizeof(double));
 ydummyh = (double *)mxCalloc(max(m,n)+lh/2-1,sizeof(double));
 g0 = (double *)mxCalloc(lh,sizeof(double));
 g1 = (double *)mxCalloc(lh,sizeof(double));

 if (n==1){
  n = m;
  m = 1;
 }
 /* synthesis lowpass and highpass */
 for (i=0; i<lh; i++){
  g0[i] = h[i];
  g1[i] = h[lh-i-1];
 }
 for (i=1; i<=lh; i+=2)
  g1[i] = -g1[i];

 lhm1 = lh - 1;
 lhhm1 = lh/2 - 1;
 /* 2^L */
 sample_f = 1;
 for (i=1; i<L; i++)
  sample_f = sample_f*2;

 if (m>1)
  actual_m = m/sample_f;
 else
  actual_m = 1;
 actual_n = n/sample_f;

 for (i=0; i<(m*n); i++)
  x[i] = y[i];

 /* main loop */
 for (actual_L=L; actual_L >= 1; actual_L--){
  r_o_a = actual_m/2;
  c_o_a = actual_n/2;

  /* go by columns in case of a 2D signal*/
  if (m>1){
   for (ic=0; ic<actual_n; ic++){          /* loop over column */
          /* store in dummy variables */
          ir = r_o_a;
          for (i=0; i<r_o_a; i++){
           ydummyl[i+lhhm1] = mat(x, i, ic);
           ydummyh[i+lhhm1] = mat(x, ir++, ic);
```

```
            }
            /* perform filtering lowpass and highpass*/
            bpsconv(xdummy, r_o_a, g0, g1, lhm1, lhhm1, ydummyl, ydummyh);
            /* restore dummy variables in matrix */
            for (i=0; i<actual_m; i++)
              mat(x, i, ic) = xdummy[i];
     }
    }
    /* go by rows */
    for (ir=0; ir<actual_m; ir++){          /* loop over rows */
      /* store in dummy variable */
      ic = c_o_a;
      for  (i=0; i<c_o_a; i++){
            ydummyl[i+lhhm1] = mat(x, ir, i);
            ydummyh[i+lhhm1] = mat(x, ir, ic++);
      }
      /* perform filtering lowpass and highpass*/
      bpsconv(xdummy, c_o_a, g0, g1, lhm1, lhhm1, ydummyl, ydummyh);
      /* restore dummy variables in matrices */
      for (i=0; i<actual_n; i++)
        mat(x, ir, i) = xdummy[i];
    }
    if (m==1)
      actual_m = 1;
    else
      actual_m = actual_m*2;
    actual_n = actual_n*2;
  }
}

#ifdef __STDC__
bpsconv(double *x_out, int lx, double *g0, double *g1, int lhm1,
            int lhhm1, double *x_inl, double *x_inh)
#else
bpsconv(x_out, lx, g0, g1, lhm1, lhhm1, x_inl, x_inh)
double *x_inl, *x_inh, *g0, *g1, *x_out;
int lx, lhm1, lhhm1;
#endif
{
  int i, j, ind, tj;
  double x0, x1;

  for (i=lhhm1-1; i > -1; i--){
    x_inl[i] = x_inl[lx+i];
    x_inh[i] = x_inh[lx+i];
  }
  ind = 0;
  for (i=0; i<(lx); i++){
    x0 = 0;
    x1 = 0;
    tj = -2;
    for (j=0; j<=lhhm1; j++){
      tj+=2;
```

```
    x0 = x0 + x_inl[i+j]*g0[lhm1-1-tj] + x_inh[i+j]*g1[lhm1-1-tj] ;
    x1 = x1 + x_inl[i+j]*g0[lhm1-tj] + x_inh[i+j]*g1[lhm1-tj] ;
   }
   x_out[ind++] = x0;
   x_out[ind++] = x1;
  }
}
```

Hard Thresholding MATLAB Code:

```
function  x = HardTh(y,thld)
%    x = HardTh(y,thld);
%
%    HARDTH hard thresholds the input signal y with the threshold value
%    thld.
%
%    Input:
%      y    : 1D or 2D signal to be thresholded
%      thld : threshold value
%
%    Output:
%      x : Hard thresholded output (x = (abs(y)>thld).*y)
%File Name: HardTh.m
%Last Modification Date: 8/15/95          17:49:37
%Current Version: HardTh.m     2.4
%File Creation Date: Mon Jan 31 09:42:50 1994
%Author: Haitao Guo  <harry@jazz.rice.edu>
%
%Copyright (c) 2000 RICE UNIVERSITY. All rights reserved.
%Created by Haitao Guo, Department of ECE, Rice University.

x = (abs(y) > thld).*y;
```

Soft Thresholding MATLAB Code:

```
function  x = SoftTh(y,thld)
%    x = SoftTh(y,thld);
%
%    SOFTTH soft thresholds the input signal y with the threshold value
%    thld.
%
%    Input:
%      y    : 1D or 2D signal to be thresholded
%      thld : Threshold value
%
%    Output:
%      x : Soft thresholded output (x = sign(y)(|y|-thld)_+)

%    Reference:
%      "De-noising via Soft-Thresholding" Tech. Rept. Statistics,
%      Stanford, 1992. D.L. Donoho.
%
%File Name: SoftTh.m
%Last Modification Date: 8/15/95          17:49:48
%Current Version: SoftTh.m       2.4
```

```
%File Creation Date: Mon Mar  7 10:38:45 1994
%Author: Haitao Guo  <harry@jazz.rice.edu>

x = abs(y);
x = sign(y).*(x >= thld).*(x - thld);
```

Appendix B:

```
--Daubechies Wavelet Transform using LUT design
--This design does not incorporate any safety nets for overflow.
-- h1 values are in lut_h1, lut_h2, lut_h3, lut_h4
-- h0 values are in lut_h4, lut_h3, lut_h2, lut_h1, respectively for each of
-- the coefficients.

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;


entity dwt is
	port(
		xin_0 : in std_logic_vector(7 downto 0);
		xin_1 : in std_logic_vector(7 downto 0);
		xin_2 : in std_logic_vector(7 downto 0);
		xin_3 : in std_logic_vector(7 downto 0);
		xin_4 : in std_logic_vector(7 downto 0);
		xin_5 : in std_logic_vector(7 downto 0);
		xin_6 : in std_logic_vector(7 downto 0);
		xin_7 : in std_logic_vector(7 downto 0);
		xin_8 : in std_logic_vector(7 downto 0);
		xin_9 : in std_logic_vector(7 downto 0);
		xin_10 : in std_logic_vector(7 downto 0);
		xin_11 : in std_logic_vector(7 downto 0);
		xin_12 : in std_logic_vector(7 downto 0);
		xin_13 : in std_logic_vector(7 downto 0);
		xin_14 : in std_logic_vector(7 downto 0);
		xin_15 : in std_logic_vector(7 downto 0);
		xin_16 : in std_logic_vector(7 downto 0);
		xin_17 : in std_logic_vector(7 downto 0);
		xin_18 : in std_logic_vector(7 downto 0);
		xin_19 : in std_logic_vector(7 downto 0);
		xin_20 : in std_logic_vector(7 downto 0);
		xin_21 : in std_logic_vector(7 downto 0);
		xin_22 : in std_logic_vector(7 downto 0);
		xin_23 : in std_logic_vector(7 downto 0);
		xin_24 : in std_logic_vector(7 downto 0);
		xin_25 : in std_logic_vector(7 downto 0);
		xin_26 : in std_logic_vector(7 downto 0);
		xin_27 : in std_logic_vector(7 downto 0);
		xin_28 : in std_logic_vector(7 downto 0);
		xin_29 : in std_logic_vector(7 downto 0);
		xin_30 : in std_logic_vector(7 downto 0);
		xin_31 : in std_logic_vector(7 downto 0);
		xin_32 : in std_logic_vector(7 downto 0);
		xin_33 : in std_logic_vector(7 downto 0);
		xin_34 : in std_logic_vector(7 downto 0);
		xin_35 : in std_logic_vector(7 downto 0);
		xin_36 : in std_logic_vector(7 downto 0);
		xin_37 : in std_logic_vector(7 downto 0);
		xin_38 : in std_logic_vector(7 downto 0);
		xin_39 : in std_logic_vector(7 downto 0);
		xin_40 : in std_logic_vector(7 downto 0);
		xin_41 : in std_logic_vector(7 downto 0);
		xin_42 : in std_logic_vector(7 downto 0);
		xin_43 : in std_logic_vector(7 downto 0);
		xin_44 : in std_logic_vector(7 downto 0);
		xin_45 : in std_logic_vector(7 downto 0);
		xin_46 : in std_logic_vector(7 downto 0);
		xin_47 : in std_logic_vector(7 downto 0);
```

```
xin_48 : in std_logic_vector(7 downto 0);
xin_49 : in std_logic_vector(7 downto 0);
xin_50 : in std_logic_vector(7 downto 0);
xin_51 : in std_logic_vector(7 downto 0);
xin_52 : in std_logic_vector(7 downto 0);
xin_53 : in std_logic_vector(7 downto 0);
xin_54 : in std_logic_vector(7 downto 0);
xin_55 : in std_logic_vector(7 downto 0);
xin_56 : in std_logic_vector(7 downto 0);
xin_57 : in std_logic_vector(7 downto 0);
xin_58 : in std_logic_vector(7 downto 0);
xin_59 : in std_logic_vector(7 downto 0);
xin_60 : in std_logic_vector(7 downto 0);
xin_61 : in std_logic_vector(7 downto 0);
xin_62 : in std_logic_vector(7 downto 0);
xin_63 : in std_logic_vector(7 downto 0);
xin_64 : in std_logic_vector(7 downto 0);
xin_65 : in std_logic_vector(7 downto 0);
xin_66 : in std_logic_vector(7 downto 0);
xin_67 : in std_logic_vector(7 downto 0);
xin_68 : in std_logic_vector(7 downto 0);
xin_69 : in std_logic_vector(7 downto 0);
xin_70 : in std_logic_vector(7 downto 0);
xin_71 : in std_logic_vector(7 downto 0);
xin_72 : in std_logic_vector(7 downto 0);
xin_73 : in std_logic_vector(7 downto 0);
xin_74 : in std_logic_vector(7 downto 0);
xin_75 : in std_logic_vector(7 downto 0);
xin_76 : in std_logic_vector(7 downto 0);
xin_77 : in std_logic_vector(7 downto 0);
xin_78 : in std_logic_vector(7 downto 0);
xin_79 : in std_logic_vector(7 downto 0);
xin_80 : in std_logic_vector(7 downto 0);
xin_81 : in std_logic_vector(7 downto 0);
xin_82 : in std_logic_vector(7 downto 0);
xin_83 : in std_logic_vector(7 downto 0);
xin_84 : in std_logic_vector(7 downto 0);
xin_85 : in std_logic_vector(7 downto 0);
xin_86 : in std_logic_vector(7 downto 0);
xin_87 : in std_logic_vector(7 downto 0);
xin_88 : in std_logic_vector(7 downto 0);
xin_89 : in std_logic_vector(7 downto 0);
xin_90 : in std_logic_vector(7 downto 0);
xin_91 : in std_logic_vector(7 downto 0);
xin_92 : in std_logic_vector(7 downto 0);
xin_93 : in std_logic_vector(7 downto 0);
xin_94 : in std_logic_vector(7 downto 0);
xin_95 : in std_logic_vector(7 downto 0);
xin_96 : in std_logic_vector(7 downto 0);
xin_97 : in std_logic_vector(7 downto 0);
xin_98 : in std_logic_vector(7 downto 0);
xin_99 : in std_logic_vector(7 downto 0);
xin_100 : in std_logic_vector(7 downto 0);
xin_101 : in std_logic_vector(7 downto 0);
xin_102 : in std_logic_vector(7 downto 0);
xin_103 : in std_logic_vector(7 downto 0);
xin_104 : in std_logic_vector(7 downto 0);
xin_105 : in std_logic_vector(7 downto 0);
xin_106 : in std_logic_vector(7 downto 0);
xin_107 : in std_logic_vector(7 downto 0);
xin_108 : in std_logic_vector(7 downto 0);
xin_109 : in std_logic_vector(7 downto 0);
xin_110 : in std_logic_vector(7 downto 0);
xin_111 : in std_logic_vector(7 downto 0);
```

```
xin_112 : in std_logic_vector(7 downto 0);
xin_113 : in std_logic_vector(7 downto 0);
xin_114 : in std_logic_vector(7 downto 0);
xin_115 : in std_logic_vector(7 downto 0);
xin_116 : in std_logic_vector(7 downto 0);
xin_117 : in std_logic_vector(7 downto 0);
xin_118 : in std_logic_vector(7 downto 0);
xin_119 : in std_logic_vector(7 downto 0);
xin_120 : in std_logic_vector(7 downto 0);
xin_121 : in std_logic_vector(7 downto 0);
xin_122 : in std_logic_vector(7 downto 0);
xin_123 : in std_logic_vector(7 downto 0);
xin_124 : in std_logic_vector(7 downto 0);
xin_125 : in std_logic_vector(7 downto 0);
xin_126 : in std_logic_vector(7 downto 0);
xin_127 : in std_logic_vector(7 downto 0);
yl_0 : out std_logic_vector(8 downto 0);
yl_1 : out std_logic_vector(8 downto 0);
yl_2 : out std_logic_vector(8 downto 0);
yl_3 : out std_logic_vector(8 downto 0);
yl_4 : out std_logic_vector(8 downto 0);
yl_5 : out std_logic_vector(8 downto 0);
yl_6 : out std_logic_vector(8 downto 0);
yl_7 : out std_logic_vector(8 downto 0);
yl_8 : out std_logic_vector(8 downto 0);
yl_9 : out std_logic_vector(8 downto 0);
yl_10 : out std_logic_vector(8 downto 0);
yl_11 : out std_logic_vector(8 downto 0);
yl_12 : out std_logic_vector(8 downto 0);
yl_13 : out std_logic_vector(8 downto 0);
yl_14 : out std_logic_vector(8 downto 0);
yl_15 : out std_logic_vector(8 downto 0);
yl_16 : out std_logic_vector(8 downto 0);
yl_17 : out std_logic_vector(8 downto 0);
yl_18 : out std_logic_vector(8 downto 0);
yl_19 : out std_logic_vector(8 downto 0);
yl_20 : out std_logic_vector(8 downto 0);
yl_21 : out std_logic_vector(8 downto 0);
yl_22 : out std_logic_vector(8 downto 0);
yl_23 : out std_logic_vector(8 downto 0);
yl_24 : out std_logic_vector(8 downto 0);
yl_25 : out std_logic_vector(8 downto 0);
yl_26 : out std_logic_vector(8 downto 0);
yl_27 : out std_logic_vector(8 downto 0);
yl_28 : out std_logic_vector(8 downto 0);
yl_29 : out std_logic_vector(8 downto 0);
yl_30 : out std_logic_vector(8 downto 0);
yl_31 : out std_logic_vector(8 downto 0);
yl_32 : out std_logic_vector(8 downto 0);
yl_33 : out std_logic_vector(8 downto 0);
yl_34 : out std_logic_vector(8 downto 0);
yl_35 : out std_logic_vector(8 downto 0);
yl_36 : out std_logic_vector(8 downto 0);
yl_37 : out std_logic_vector(8 downto 0);
yl_38 : out std_logic_vector(8 downto 0);
yl_39 : out std_logic_vector(8 downto 0);
yl_40 : out std_logic_vector(8 downto 0);
yl_41 : out std_logic_vector(8 downto 0);
yl_42 : out std_logic_vector(8 downto 0);
yl_43 : out std_logic_vector(8 downto 0);
yl_44 : out std_logic_vector(8 downto 0);
yl_45 : out std_logic_vector(8 downto 0);
yl_46 : out std_logic_vector(8 downto 0);
yl_47 : out std_logic_vector(8 downto 0);
```

```
yl_48 : out std_logic_vector(8 downto 0);
yl_49 : out std_logic_vector(8 downto 0);
yl_50 : out std_logic_vector(8 downto 0);
yl_51 : out std_logic_vector(8 downto 0);
yl_52 : out std_logic_vector(8 downto 0);
yl_53 : out std_logic_vector(8 downto 0);
yl_54 : out std_logic_vector(8 downto 0);
yl_55 : out std_logic_vector(8 downto 0);
yl_56 : out std_logic_vector(8 downto 0);
yl_57 : out std_logic_vector(8 downto 0);
yl_58 : out std_logic_vector(8 downto 0);
yl_59 : out std_logic_vector(8 downto 0);
yl_60 : out std_logic_vector(8 downto 0);
yl_61 : out std_logic_vector(8 downto 0);
yl_62 : out std_logic_vector(8 downto 0);
yl_63 : out std_logic_vector(8 downto 0);
yh_0 : out std_logic_vector(8 downto 0);
yh_1 : out std_logic_vector(8 downto 0);
yh_2 : out std_logic_vector(8 downto 0);
yh_3 : out std_logic_vector(8 downto 0);
yh_4 : out std_logic_vector(8 downto 0);
yh_5 : out std_logic_vector(8 downto 0);
yh_6 : out std_logic_vector(8 downto 0);
yh_7 : out std_logic_vector(8 downto 0);
yh_8 : out std_logic_vector(8 downto 0);
yh_9 : out std_logic_vector(8 downto 0);
yh_10 : out std_logic_vector(8 downto 0);
yh_11 : out std_logic_vector(8 downto 0);
yh_12 : out std_logic_vector(8 downto 0);
yh_13 : out std_logic_vector(8 downto 0);
yh_14 : out std_logic_vector(8 downto 0);
yh_15 : out std_logic_vector(8 downto 0);
yh_16 : out std_logic_vector(8 downto 0);
yh_17 : out std_logic_vector(8 downto 0);
yh_18 : out std_logic_vector(8 downto 0);
yh_19 : out std_logic_vector(8 downto 0);
yh_20 : out std_logic_vector(8 downto 0);
yh_21 : out std_logic_vector(8 downto 0);
yh_22 : out std_logic_vector(8 downto 0);
yh_23 : out std_logic_vector(8 downto 0);
yh_24 : out std_logic_vector(8 downto 0);
yh_25 : out std_logic_vector(8 downto 0);
yh_26 : out std_logic_vector(8 downto 0);
yh_27 : out std_logic_vector(8 downto 0);
yh_28 : out std_logic_vector(8 downto 0);
yh_29 : out std_logic_vector(8 downto 0);
yh_30 : out std_logic_vector(8 downto 0);
yh_31 : out std_logic_vector(8 downto 0);
yh_32 : out std_logic_vector(8 downto 0);
yh_33 : out std_logic_vector(8 downto 0);
yh_34 : out std_logic_vector(8 downto 0);
yh_35 : out std_logic_vector(8 downto 0);
yh_36 : out std_logic_vector(8 downto 0);
yh_37 : out std_logic_vector(8 downto 0);
yh_38 : out std_logic_vector(8 downto 0);
yh_39 : out std_logic_vector(8 downto 0);
yh_40 : out std_logic_vector(8 downto 0);
yh_41 : out std_logic_vector(8 downto 0);
yh_42 : out std_logic_vector(8 downto 0);
yh_43 : out std_logic_vector(8 downto 0);
yh_44 : out std_logic_vector(8 downto 0);
yh_45 : out std_logic_vector(8 downto 0);
yh_46 : out std_logic_vector(8 downto 0);
yh_47 : out std_logic_vector(8 downto 0);
```

```
		yh_48 : out std_logic_vector(8 downto 0);
		yh_49 : out std_logic_vector(8 downto 0);
		yh_50 : out std_logic_vector(8 downto 0);
		yh_51 : out std_logic_vector(8 downto 0);
		yh_52 : out std_logic_vector(8 downto 0);
		yh_53 : out std_logic_vector(8 downto 0);
		yh_54 : out std_logic_vector(8 downto 0);
		yh_55 : out std_logic_vector(8 downto 0);
		yh_56 : out std_logic_vector(8 downto 0);
		yh_57 : out std_logic_vector(8 downto 0);
		yh_58 : out std_logic_vector(8 downto 0);
		yh_59 : out std_logic_vector(8 downto 0);
		yh_60 : out std_logic_vector(8 downto 0);
		yh_61 : out std_logic_vector(8 downto 0);
		yh_62 : out std_logic_vector(8 downto 0);
		yh_63 : out std_logic_vector(8 downto 0)
		);
	end entity;

	architecture behavioral of dwt is
	component lut_h1
		port(
			address : in std_logic_vector(7 downto 0);
			result : out std_logic_vector(8 downto 0)
		);
	end component;

	component lut_h2
		port(
			address : in std_logic_vector(7 downto 0);
			result : out std_logic_vector(8 downto 0)
		);
	end component;

	component lut_h3
		port(
			address : in std_logic_vector(7 downto 0);
			result : out std_logic_vector(8 downto 0)
		);
	end component;

	component lut_h4
		port(
			address : in std_logic_vector(7 downto 0);
			result : out std_logic_vector(8 downto 0)
		);
	end component;
	signal x0_0, x0_1, x0_2, x0_3, x0_4, x0_5, x0_6, x0_7, x0_8, x0_9, x0_10,
x0_11,
			x0_12, x0_13, x0_14, x0_15, x0_16, x0_17, x0_18, x0_19, x0_20,
x0_21,
			x0_22, x0_23, x0_24, x0_25, x0_26, x0_27, x0_28, x0_29, x0_30,
x0_31,
			x0_32, x0_33, x0_34, x0_35, x0_36, x0_37, x0_38, x0_39, x0_40,
x0_41,
			x0_42, x0_43, x0_44, x0_45, x0_46, x0_47, x0_48, x0_49, x0_50,
x0_51,
			x0_52, x0_53, x0_54, x0_55, x0_56, x0_57, x0_58, x0_59, x0_60,
x0_61,
			x0_62, x0_63, x0_64, x0_65, x0_66, x0_67, x0_68, x0_69, x0_70,
x0_71,
			x0_72, x0_73, x0_74, x0_75, x0_76, x0_77, x0_78, x0_79, x0_80,
x0_81,
			x0_82, x0_83, x0_84, x0_85, x0_86, x0_87, x0_88, x0_89, x0_90,
```

```
x0_91,
                x0_92, x0_93, x0_94, x0_95, x0_96, x0_97, x0_98, x0_99, x0_100,
x0_101,
                x0_102, x0_103, x0_104, x0_105, x0_106, x0_107, x0_108, x0_109,
x0_110,
                x0_111, x0_112, x0_113, x0_114, x0_115, x0_116, x0_117, x0_118,
x0_119,
                x0_120, x0_121, x0_122, x0_123, x0_124, x0_125, x0_126, x0_127,
x0_128,
                x0_129, x0_130, x0_131, x0_132, x0_133, x0_134, x0_135, x0_136,
x0_137,
                x0_138, x0_139, x0_140, x0_141, x0_142, x0_143, x0_144, x0_145,
x0_146,
                x0_147, x0_148, x0_149, x0_150, x0_151, x0_152, x0_153, x0_154,
x0_155,
                x0_156, x0_157, x0_158, x0_159, x0_160, x0_161, x0_162, x0_163,
x0_164,
                x0_165, x0_166, x0_167, x0_168, x0_169, x0_170, x0_171, x0_172,
x0_173,
                x0_174, x0_175, x0_176, x0_177, x0_178, x0_179, x0_180, x0_181,
x0_182,
                x0_183, x0_184, x0_185, x0_186, x0_187, x0_188, x0_189, x0_190,
x0_191,
                x0_192, x0_193, x0_194, x0_195, x0_196, x0_197, x0_198, x0_199,
x0_200,
                x0_201, x0_202, x0_203, x0_204, x0_205, x0_206, x0_207, x0_208,
x0_209,
                x0_210, x0_211, x0_212, x0_213, x0_214, x0_215, x0_216, x0_217,
x0_218,
                x0_219, x0_220, x0_221, x0_222, x0_223, x0_224, x0_225, x0_226,
x0_227,
                x0_228, x0_229, x0_230, x0_231, x0_232, x0_233, x0_234, x0_235,
x0_236,
                x0_237, x0_238, x0_239, x0_240, x0_241, x0_242, x0_243, x0_244,
x0_245,
                x0_246, x0_247, x0_248, x0_249, x0_250, x0_251, x0_252, x0_253,
x0_254,
                x0_255, x1_0, x1_1, x1_2, x1_3, x1_4, x1_5, x1_6, x1_7, x1_8,
x1_9, x1_10,
                x1_11, x1_12, x1_13, x1_14, x1_15, x1_16, x1_17, x1_18, x1_19,
x1_20, x1_21,
                x1_22, x1_23, x1_24, x1_25, x1_26, x1_27, x1_28, x1_29, x1_30,
x1_31, x1_32,
                x1_33, x1_34, x1_35, x1_36, x1_37, x1_38, x1_39, x1_40, x1_41,
x1_42, x1_43,
                x1_44, x1_45, x1_46, x1_47, x1_48, x1_49, x1_50, x1_51, x1_52,
x1_53, x1_54,
                x1_55, x1_56, x1_57, x1_58, x1_59, x1_60, x1_61, x1_62, x1_63,
x1_64, x1_65,
                x1_66, x1_67, x1_68, x1_69, x1_70, x1_71, x1_72, x1_73, x1_74,
x1_75, x1_76,
                x1_77, x1_78, x1_79, x1_80, x1_81, x1_82, x1_83, x1_84, x1_85,
x1_86, x1_87,
                x1_88, x1_89, x1_90, x1_91, x1_92, x1_93, x1_94, x1_95, x1_96,
x1_97, x1_98,
                x1_99, x1_100, x1_101, x1_102, x1_103, x1_104, x1_105, x1_106,
x1_107, x1_108,
                x1_109, x1_110, x1_111, x1_112, x1_113, x1_114, x1_115, x1_116,
x1_117, x1_118,
                x1_119, x1_120, x1_121, x1_122, x1_123, x1_124, x1_125, x1_126,
x1_127, x1_128,
                x1_129, x1_130, x1_131, x1_132, x1_133, x1_134, x1_135, x1_136,
x1_137, x1_138,
                x1_139, x1_140, x1_141, x1_142, x1_143, x1_144, x1_145, x1_146,
```

```
x1_147, x1_148,
                        x1_149, x1_150, x1_151, x1_152, x1_153, x1_154, x1_155, x1_156,
x1_157, x1_158,
                        x1_159, x1_160, x1_161, x1_162, x1_163, x1_164, x1_165, x1_166,
x1_167, x1_168,
                        x1_169, x1_170, x1_171, x1_172, x1_173, x1_174, x1_175, x1_176,
x1_177, x1_178,
                        x1_179, x1_180, x1_181, x1_182, x1_183, x1_184, x1_185, x1_186,
x1_187, x1_188,
                        x1_189, x1_190, x1_191, x1_192, x1_193, x1_194, x1_195, x1_196,
x1_197, x1_198,
                        x1_199, x1_200, x1_201, x1_202, x1_203, x1_204, x1_205, x1_206,
x1_207, x1_208,
                        x1_209, x1_210, x1_211, x1_212, x1_213, x1_214, x1_215, x1_216,
x1_217, x1_218,
                        x1_219, x1_220, x1_221, x1_222, x1_223, x1_224, x1_225, x1_226,
x1_227, x1_228,
                        x1_229, x1_230, x1_231, x1_232, x1_233, x1_234, x1_235, x1_236,
x1_237, x1_238,
                        x1_239, x1_240, x1_241, x1_242, x1_243, x1_244, x1_245, x1_246,
x1_247, x1_248,
                        x1_249, x1_250, x1_251, x1_252, x1_253, x1_254, x1_255 :
std_logic_vector(8 downto 0);
       begin

i0_0: lut_h1
       port map(
       address => xin_0, result => x0_0
);
i0_1: lut_h2
       port map(
       address => xin_1, result => x0_1
);
i0_2: lut_h3
       port map(
       address => xin_2, result => x0_2
);
i0_3: lut_h4
       port map(
       address => xin_3, result => x0_3
);
yl_0 <= x0_0 + x0_1 + x0_2 + x0_3;

i0_4: lut_h1
       port map(
       address => xin_2, result => x0_4
);
i0_5: lut_h2
       port map(
       address => xin_3, result => x0_5
);
i0_6: lut_h3
       port map(
       address => xin_4, result => x0_6
);
i0_7: lut_h4
       port map(
       address => xin_5, result => x0_7
);
yl_1 <= x0_4 + x0_5 + x0_6 + x0_7;

i0_8: lut_h1
       port map(
       address => xin_4, result => x0_8
```

```
);
i0_9: lut_h2
       port map(
       address => xin_5, result => x0_9
);
i0_10: lut_h3
       port map(
       address => xin_6, result => x0_10
);
i0_11: lut_h4
       port map(
       address => xin_7, result => x0_11
);
yl_2 <= x0_8 + x0_9 + x0_10 + x0_11;

i0_12: lut_h1
       port map(
       address => xin_6, result => x0_12
);
i0_13: lut_h2
       port map(
       address => xin_7, result => x0_13
);
i0_14: lut_h3
       port map(
       address => xin_8, result => x0_14
);
i0_15: lut_h4
       port map(
       address => xin_9, result => x0_15
);
yl_3 <= x0_12 + x0_13 + x0_14 + x0_15;

i0_16: lut_h1
       port map(
       address => xin_8, result => x0_16
);
i0_17: lut_h2
       port map(
       address => xin_9, result => x0_17
);
i0_18: lut_h3
       port map(
       address => xin_10, result => x0_18
);
i0_19: lut_h4
       port map(
       address => xin_11, result => x0_19
);
yl_4 <= x0_16 + x0_17 + x0_18 + x0_19;

i0_20: lut_h1
       port map(
       address => xin_10, result => x0_20
);
i0_21: lut_h2
       port map(
       address => xin_11, result => x0_21
);
i0_22: lut_h3
       port map(
       address => xin_12, result => x0_22
);
i0_23: lut_h4
```

```
        port map(
        address => xin_13, result => x0_23
);
yl_5 <= x0_20 + x0_21 + x0_22 + x0_23;

i0_24: lut_h1
        port map(
        address => xin_12, result => x0_24
);
i0_25: lut_h2
        port map(
        address => xin_13, result => x0_25
);
i0_26: lut_h3
        port map(
        address => xin_14, result => x0_26
);
i0_27: lut_h4
        port map(
        address => xin_15, result => x0_27
);
yl_6 <= x0_24 + x0_25 + x0_26 + x0_27;

i0_28: lut_h1
        port map(
        address => xin_14, result => x0_28
);
i0_29: lut_h2
        port map(
        address => xin_15, result => x0_29
);
i0_30: lut_h3
        port map(
        address => xin_16, result => x0_30
);
i0_31: lut_h4
        port map(
        address => xin_17, result => x0_31
);
yl_7 <= x0_28 + x0_29 + x0_30 + x0_31;

i0_32: lut_h1
        port map(
        address => xin_16, result => x0_32
);
i0_33: lut_h2
        port map(
        address => xin_17, result => x0_33
);
i0_34: lut_h3
        port map(
        address => xin_18, result => x0_34
);
i0_35: lut_h4
        port map(
        address => xin_19, result => x0_35
);
yl_8 <= x0_32 + x0_33 + x0_34 + x0_35;

i0_36: lut_h1
        port map(
        address => xin_18, result => x0_36
);
i0_37: lut_h2
```

```
        port map(
        address => xin_19, result => x0_37
);
i0_38: lut_h3
        port map(
        address => xin_20, result => x0_38
);
i0_39: lut_h4
        port map(
        address => xin_21, result => x0_39
);
yl_9 <= x0_36 + x0_37 + x0_38 + x0_39;

i0_40: lut_h1
        port map(
        address => xin_20, result => x0_40
);
i0_41: lut_h2
        port map(
        address => xin_21, result => x0_41
);
i0_42: lut_h3
        port map(
        address => xin_22, result => x0_42
);
i0_43: lut_h4
        port map(
        address => xin_23, result => x0_43
);
yl_10 <= x0_40 + x0_41 + x0_42 + x0_43;

i0_44: lut_h1
        port map(
        address => xin_22, result => x0_44
);
i0_45: lut_h2
        port map(
        address => xin_23, result => x0_45
);
i0_46: lut_h3
        port map(
        address => xin_24, result => x0_46
);
i0_47: lut_h4
        port map(
        address => xin_25, result => x0_47
);
yl_11 <= x0_44 + x0_45 + x0_46 + x0_47;

i0_48: lut_h1
        port map(
        address => xin_24, result => x0_48
);
i0_49: lut_h2
        port map(
        address => xin_25, result => x0_49
);
i0_50: lut_h3
        port map(
        address => xin_26, result => x0_50
);
i0_51: lut_h4
        port map(
        address => xin_27, result => x0_51
```

```
);
yl_12 <= x0_48 + x0_49 + x0_50 + x0_51;

i0_52: lut_h1
       port map(
       address => xin_26, result => x0_52
);
i0_53: lut_h2
       port map(
       address => xin_27, result => x0_53
);
i0_54: lut_h3
       port map(
       address => xin_28, result => x0_54
);
i0_55: lut_h4
       port map(
       address => xin_29, result => x0_55
);
yl_13 <= x0_52 + x0_53 + x0_54 + x0_55;

i0_56: lut_h1
       port map(
       address => xin_28, result => x0_56
);
i0_57: lut_h2
       port map(
       address => xin_29, result => x0_57
);
i0_58: lut_h3
       port map(
       address => xin_30, result => x0_58
);
i0_59: lut_h4
       port map(
       address => xin_31, result => x0_59
);
yl_14 <= x0_56 + x0_57 + x0_58 + x0_59;

i0_60: lut_h1
       port map(
       address => xin_30, result => x0_60
);
i0_61: lut_h2
       port map(
       address => xin_31, result => x0_61
);
i0_62: lut_h3
       port map(
       address => xin_32, result => x0_62
);
i0_63: lut_h4
       port map(
       address => xin_33, result => x0_63
);
yl_15 <= x0_60 + x0_61 + x0_62 + x0_63;

i0_64: lut_h1
       port map(
       address => xin_32, result => x0_64
);
i0_65: lut_h2
       port map(
       address => xin_33, result => x0_65
```

```
);
i0_66: lut_h3
       port map(
       address => xin_34, result => x0_66
);
i0_67: lut_h4
       port map(
       address => xin_35, result => x0_67
);
yl_16 <= x0_64 + x0_65 + x0_66 + x0_67;

i0_68: lut_h1
       port map(
       address => xin_34, result => x0_68
);
i0_69: lut_h2
       port map(
       address => xin_35, result => x0_69
);
i0_70: lut_h3
       port map(
       address => xin_36, result => x0_70
);
i0_71: lut_h4
       port map(
       address => xin_37, result => x0_71
);
yl_17 <= x0_68 + x0_69 + x0_70 + x0_71;

i0_72: lut_h1
       port map(
       address => xin_36, result => x0_72
);
i0_73: lut_h2
       port map(
       address => xin_37, result => x0_73
);
i0_74: lut_h3
       port map(
       address => xin_38, result => x0_74
);
i0_75: lut_h4
       port map(
       address => xin_39, result => x0_75
);
yl_18 <= x0_72 + x0_73 + x0_74 + x0_75;

i0_76: lut_h1
       port map(
       address => xin_38, result => x0_76
);
i0_77: lut_h2
       port map(
       address => xin_39, result => x0_77
);
i0_78: lut_h3
       port map(
       address => xin_40, result => x0_78
);
i0_79: lut_h4
       port map(
       address => xin_41, result => x0_79
);
yl_19 <= x0_76 + x0_77 + x0_78 + x0_79;
```

```
i0_80: lut_h1
       port map(
       address => xin_40, result => x0_80
);
i0_81: lut_h2
       port map(
       address => xin_41, result => x0_81
);
i0_82: lut_h3
       port map(
       address => xin_42, result => x0_82
);
i0_83: lut_h4
       port map(
       address => xin_43, result => x0_83
);
yl_20 <= x0_80 + x0_81 + x0_82 + x0_83;

i0_84: lut_h1
       port map(
       address => xin_42, result => x0_84
);
i0_85: lut_h2
       port map(
       address => xin_43, result => x0_85
);
i0_86: lut_h3
       port map(
       address => xin_44, result => x0_86
);
i0_87: lut_h4
       port map(
       address => xin_45, result => x0_87
);
yl_21 <= x0_84 + x0_85 + x0_86 + x0_87;

i0_88: lut_h1
       port map(
       address => xin_44, result => x0_88
);
i0_89: lut_h2
       port map(
       address => xin_45, result => x0_89
);
i0_90: lut_h3
       port map(
       address => xin_46, result => x0_90
);
i0_91: lut_h4
       port map(
       address => xin_47, result => x0_91
);
yl_22 <= x0_88 + x0_89 + x0_90 + x0_91;

i0_92: lut_h1
       port map(
       address => xin_46, result => x0_92
);
i0_93: lut_h2
       port map(
       address => xin_47, result => x0_93
);
i0_94: lut_h3
```

```
        port map(
        address => xin_48, result => x0_94
);
i0_95: lut_h4
        port map(
        address => xin_49, result => x0_95
);
yl_23 <= x0_92 + x0_93 + x0_94 + x0_95;

i0_96: lut_h1
        port map(
        address => xin_48, result => x0_96
);
i0_97: lut_h2
        port map(
        address => xin_49, result => x0_97
);
i0_98: lut_h3
        port map(
        address => xin_50, result => x0_98
);
i0_99: lut_h4
        port map(
        address => xin_51, result => x0_99
);
yl_24 <= x0_96 + x0_97 + x0_98 + x0_99;

i0_100: lut_h1
        port map(
        address => xin_50, result => x0_100
);
i0_101: lut_h2
        port map(
        address => xin_51, result => x0_101
);
i0_102: lut_h3
        port map(
        address => xin_52, result => x0_102
);
i0_103: lut_h4
        port map(
        address => xin_53, result => x0_103
);
yl_25 <= x0_100 + x0_101 + x0_102 + x0_103;

i0_104: lut_h1
        port map(
        address => xin_52, result => x0_104
);
i0_105: lut_h2
        port map(
        address => xin_53, result => x0_105
);
i0_106: lut_h3
        port map(
        address => xin_54, result => x0_106
);
i0_107: lut_h4
        port map(
        address => xin_55, result => x0_107
);
yl_26 <= x0_104 + x0_105 + x0_106 + x0_107;

i0_108: lut_h1
```

```
        port map(
        address => xin_54, result => x0_108
);
i0_109: lut_h2
        port map(
        address => xin_55, result => x0_109
);
i0_110: lut_h3
        port map(
        address => xin_56, result => x0_110
);
i0_111: lut_h4
        port map(
        address => xin_57, result => x0_111
);
yl_27 <= x0_108 + x0_109 + x0_110 + x0_111;

i0_112: lut_h1
        port map(
        address => xin_56, result => x0_112
);
i0_113: lut_h2
        port map(
        address => xin_57, result => x0_113
);
i0_114: lut_h3
        port map(
        address => xin_58, result => x0_114
);
i0_115: lut_h4
        port map(
        address => xin_59, result => x0_115
);
yl_28 <= x0_112 + x0_113 + x0_114 + x0_115;

i0_116: lut_h1
        port map(
        address => xin_58, result => x0_116
);
i0_117: lut_h2
        port map(
        address => xin_59, result => x0_117
);
i0_118: lut_h3
        port map(
        address => xin_60, result => x0_118
);
i0_119: lut_h4
        port map(
        address => xin_61, result => x0_119
);
yl_29 <= x0_116 + x0_117 + x0_118 + x0_119;

i0_120: lut_h1
        port map(
        address => xin_60, result => x0_120
);
i0_121: lut_h2
        port map(
        address => xin_61, result => x0_121
);
i0_122: lut_h3
        port map(
        address => xin_62, result => x0_122
```

```
);
i0_123: lut_h4
       port map(
       address => xin_63, result => x0_123
);
yl_30 <= x0_120 + x0_121 + x0_122 + x0_123;

i0_124: lut_h1
       port map(
       address => xin_62, result => x0_124
);
i0_125: lut_h2
       port map(
       address => xin_63, result => x0_125
);
i0_126: lut_h3
       port map(
       address => xin_64, result => x0_126
);
i0_127: lut_h4
       port map(
       address => xin_65, result => x0_127
);
yl_31 <= x0_124 + x0_125 + x0_126 + x0_127;

i0_128: lut_h1
       port map(
       address => xin_64, result => x0_128
);
i0_129: lut_h2
       port map(
       address => xin_65, result => x0_129
);
i0_130: lut_h3
       port map(
       address => xin_66, result => x0_130
);
i0_131: lut_h4
       port map(
       address => xin_67, result => x0_131
);
yl_32 <= x0_128 + x0_129 + x0_130 + x0_131;

i0_132: lut_h1
       port map(
       address => xin_66, result => x0_132
);
i0_133: lut_h2
       port map(
       address => xin_67, result => x0_133
);
i0_134: lut_h3
       port map(
       address => xin_68, result => x0_134
);
i0_135: lut_h4
       port map(
       address => xin_69, result => x0_135
);
yl_33 <= x0_132 + x0_133 + x0_134 + x0_135;

i0_136: lut_h1
       port map(
       address => xin_68, result => x0_136
```

```
);
i0_137: lut_h2
       port map(
       address => xin_69, result => x0_137
);
i0_138: lut_h3
       port map(
       address => xin_70, result => x0_138
);
i0_139: lut_h4
       port map(
       address => xin_71, result => x0_139
);
yl_34 <= x0_136 + x0_137 + x0_138 + x0_139;

i0_140: lut_h1
       port map(
       address => xin_70, result => x0_140
);
i0_141: lut_h2
       port map(
       address => xin_71, result => x0_141
);
i0_142: lut_h3
       port map(
       address => xin_72, result => x0_142
);
i0_143: lut_h4
       port map(
       address => xin_73, result => x0_143
);
yl_35 <= x0_140 + x0_141 + x0_142 + x0_143;

i0_144: lut_h1
       port map(
       address => xin_72, result => x0_144
);
i0_145: lut_h2
       port map(
       address => xin_73, result => x0_145
);
i0_146: lut_h3
       port map(
       address => xin_74, result => x0_146
);
i0_147: lut_h4
       port map(
       address => xin_75, result => x0_147
);
yl_36 <= x0_144 + x0_145 + x0_146 + x0_147;

i0_148: lut_h1
       port map(
       address => xin_74, result => x0_148
);
i0_149: lut_h2
       port map(
       address => xin_75, result => x0_149
);
i0_150: lut_h3
       port map(
       address => xin_76, result => x0_150
);
i0_151: lut_h4
```

```
        port map(
        address => xin_77, result => x0_151
);
yl_37 <= x0_148 + x0_149 + x0_150 + x0_151;

i0_152: lut_h1
        port map(
        address => xin_76, result => x0_152
);
i0_153: lut_h2
        port map(
        address => xin_77, result => x0_153
);
i0_154: lut_h3
        port map(
        address => xin_78, result => x0_154
);
i0_155: lut_h4
        port map(
        address => xin_79, result => x0_155
);
yl_38 <= x0_152 + x0_153 + x0_154 + x0_155;

i0_156: lut_h1
        port map(
        address => xin_78, result => x0_156
);
i0_157: lut_h2
        port map(
        address => xin_79, result => x0_157
);
i0_158: lut_h3
        port map(
        address => xin_80, result => x0_158
);
i0_159: lut_h4
        port map(
        address => xin_81, result => x0_159
);
yl_39 <= x0_156 + x0_157 + x0_158 + x0_159;

i0_160: lut_h1
        port map(
        address => xin_80, result => x0_160
);
i0_161: lut_h2
        port map(
        address => xin_81, result => x0_161
);
i0_162: lut_h3
        port map(
        address => xin_82, result => x0_162
);
i0_163: lut_h4
        port map(
        address => xin_83, result => x0_163
);
yl_40 <= x0_160 + x0_161 + x0_162 + x0_163;

i0_164: lut_h1
        port map(
        address => xin_82, result => x0_164
);
i0_165: lut_h2
```

```
        port map(
        address => xin_83, result => x0_165
);
i0_166: lut_h3
        port map(
        address => xin_84, result => x0_166
);
i0_167: lut_h4
        port map(
        address => xin_85, result => x0_167
);
yl_41 <= x0_164 + x0_165 + x0_166 + x0_167;

i0_168: lut_h1
        port map(
        address => xin_84, result => x0_168
);
i0_169: lut_h2
        port map(
        address => xin_85, result => x0_169
);
i0_170: lut_h3
        port map(
        address => xin_86, result => x0_170
);
i0_171: lut_h4
        port map(
        address => xin_87, result => x0_171
);
yl_42 <= x0_168 + x0_169 + x0_170 + x0_171;

i0_172: lut_h1
        port map(
        address => xin_86, result => x0_172
);
i0_173: lut_h2
        port map(
        address => xin_87, result => x0_173
);
i0_174: lut_h3
        port map(
        address => xin_88, result => x0_174
);
i0_175: lut_h4
        port map(
        address => xin_89, result => x0_175
);
yl_43 <= x0_172 + x0_173 + x0_174 + x0_175;

i0_176: lut_h1
        port map(
        address => xin_88, result => x0_176
);
i0_177: lut_h2
        port map(
        address => xin_89, result => x0_177
);
i0_178: lut_h3
        port map(
        address => xin_90, result => x0_178
);
i0_179: lut_h4
        port map(
        address => xin_91, result => x0_179
```

```
);
yl_44 <= x0_176 + x0_177 + x0_178 + x0_179;

i0_180: lut_h1
       port map(
       address => xin_90, result => x0_180
);
i0_181: lut_h2
       port map(
       address => xin_91, result => x0_181
);
i0_182: lut_h3
       port map(
       address => xin_92, result => x0_182
);
i0_183: lut_h4
       port map(
       address => xin_93, result => x0_183
);
yl_45 <= x0_180 + x0_181 + x0_182 + x0_183;

i0_184: lut_h1
       port map(
       address => xin_92, result => x0_184
);
i0_185: lut_h2
       port map(
       address => xin_93, result => x0_185
);
i0_186: lut_h3
       port map(
       address => xin_94, result => x0_186
);
i0_187: lut_h4
       port map(
       address => xin_95, result => x0_187
);
yl_46 <= x0_184 + x0_185 + x0_186 + x0_187;

i0_188: lut_h1
       port map(
       address => xin_94, result => x0_188
);
i0_189: lut_h2
       port map(
       address => xin_95, result => x0_189
);
i0_190: lut_h3
       port map(
       address => xin_96, result => x0_190
);
i0_191: lut_h4
       port map(
       address => xin_97, result => x0_191
);
yl_47 <= x0_188 + x0_189 + x0_190 + x0_191;

i0_192: lut_h1
       port map(
       address => xin_96, result => x0_192
);
i0_193: lut_h2
       port map(
       address => xin_97, result => x0_193
```

```
);
i0_194: lut_h3
       port map(
       address => xin_98, result => x0_194
);
i0_195: lut_h4
       port map(
       address => xin_99, result => x0_195
);
yl_48 <= x0_192 + x0_193 + x0_194 + x0_195;

i0_196: lut_h1
       port map(
       address => xin_98, result => x0_196
);
i0_197: lut_h2
       port map(
       address => xin_99, result => x0_197
);
i0_198: lut_h3
       port map(
       address => xin_100, result => x0_198
);
i0_199: lut_h4
       port map(
       address => xin_101, result => x0_199
);
yl_49 <= x0_196 + x0_197 + x0_198 + x0_199;

i0_200: lut_h1
       port map(
       address => xin_100, result => x0_200
);
i0_201: lut_h2
       port map(
       address => xin_101, result => x0_201
);
i0_202: lut_h3
       port map(
       address => xin_102, result => x0_202
);
i0_203: lut_h4
       port map(
       address => xin_103, result => x0_203
);
yl_50 <= x0_200 + x0_201 + x0_202 + x0_203;

i0_204: lut_h1
       port map(
       address => xin_102, result => x0_204
);
i0_205: lut_h2
       port map(
       address => xin_103, result => x0_205
);
i0_206: lut_h3
       port map(
       address => xin_104, result => x0_206
);
i0_207: lut_h4
       port map(
       address => xin_105, result => x0_207
);
yl_51 <= x0_204 + x0_205 + x0_206 + x0_207;
```

```
i0_208: lut_h1
       port map(
       address => xin_104, result => x0_208
);
i0_209: lut_h2
       port map(
       address => xin_105, result => x0_209
);
i0_210: lut_h3
       port map(
       address => xin_106, result => x0_210
);
i0_211: lut_h4
       port map(
       address => xin_107, result => x0_211
);
yl_52 <= x0_208 + x0_209 + x0_210 + x0_211;

i0_212: lut_h1
       port map(
       address => xin_106, result => x0_212
);
i0_213: lut_h2
       port map(
       address => xin_107, result => x0_213
);
i0_214: lut_h3
       port map(
       address => xin_108, result => x0_214
);
i0_215: lut_h4
       port map(
       address => xin_109, result => x0_215
);
yl_53 <= x0_212 + x0_213 + x0_214 + x0_215;

i0_216: lut_h1
       port map(
       address => xin_108, result => x0_216
);
i0_217: lut_h2
       port map(
       address => xin_109, result => x0_217
);
i0_218: lut_h3
       port map(
       address => xin_110, result => x0_218
);
i0_219: lut_h4
       port map(
       address => xin_111, result => x0_219
);
yl_54 <= x0_216 + x0_217 + x0_218 + x0_219;

i0_220: lut_h1
       port map(
       address => xin_110, result => x0_220
);
i0_221: lut_h2
       port map(
       address => xin_111, result => x0_221
);
i0_222: lut_h3
```

```
        port map(
        address => xin_112, result => x0_222
);
i0_223: lut_h4
        port map(
        address => xin_113, result => x0_223
);
yl_55 <= x0_220 + x0_221 + x0_222 + x0_223;

i0_224: lut_h1
        port map(
        address => xin_112, result => x0_224
);
i0_225: lut_h2
        port map(
        address => xin_113, result => x0_225
);
i0_226: lut_h3
        port map(
        address => xin_114, result => x0_226
);
i0_227: lut_h4
        port map(
        address => xin_115, result => x0_227
);
yl_56 <= x0_224 + x0_225 + x0_226 + x0_227;

i0_228: lut_h1
        port map(
        address => xin_114, result => x0_228
);
i0_229: lut_h2
        port map(
        address => xin_115, result => x0_229
);
i0_230: lut_h3
        port map(
        address => xin_116, result => x0_230
);
i0_231: lut_h4
        port map(
        address => xin_117, result => x0_231
);
yl_57 <= x0_228 + x0_229 + x0_230 + x0_231;

i0_232: lut_h1
        port map(
        address => xin_116, result => x0_232
);
i0_233: lut_h2
        port map(
        address => xin_117, result => x0_233
);
i0_234: lut_h3
        port map(
        address => xin_118, result => x0_234
);
i0_235: lut_h4
        port map(
        address => xin_119, result => x0_235
);
yl_58 <= x0_232 + x0_233 + x0_234 + x0_235;

i0_236: lut_h1
```

```
        port map(
        address => xin_118, result => x0_236
);
i0_237: lut_h2
        port map(
        address => xin_119, result => x0_237
);
i0_238: lut_h3
        port map(
        address => xin_120, result => x0_238
);
i0_239: lut_h4
        port map(
        address => xin_121, result => x0_239
);
yl_59 <= x0_236 + x0_237 + x0_238 + x0_239;

i0_240: lut_h1
        port map(
        address => xin_120, result => x0_240
);
i0_241: lut_h2
        port map(
        address => xin_121, result => x0_241
);
i0_242: lut_h3
        port map(
        address => xin_122, result => x0_242
);
i0_243: lut_h4
        port map(
        address => xin_123, result => x0_243
);
yl_60 <= x0_240 + x0_241 + x0_242 + x0_243;

i0_244: lut_h1
        port map(
        address => xin_122, result => x0_244
);
i0_245: lut_h2
        port map(
        address => xin_123, result => x0_245
);
i0_246: lut_h3
        port map(
        address => xin_124, result => x0_246
);
i0_247: lut_h4
        port map(
        address => xin_125, result => x0_247
);
yl_61 <= x0_244 + x0_245 + x0_246 + x0_247;

i0_248: lut_h1
        port map(
        address => xin_124, result => x0_248
);
i0_249: lut_h2
        port map(
        address => xin_125, result => x0_249
);
i0_250: lut_h3
        port map(
        address => xin_126, result => x0_250
```

```
);
i0_251: lut_h4
       port map(
       address => xin_127, result => x0_251
);
yl_62 <= x0_248 + x0_249 + x0_250 + x0_251;

i0_252: lut_h1
       port map(
       address => xin_126, result => x0_252
);
i0_253: lut_h2
       port map(
       address => xin_127, result => x0_253
);
i0_254: lut_h3
       port map(
       address => xin_0, result => x0_254
);
i0_255: lut_h4
       port map(
       address => xin_1, result => x0_255
);
yl_63 <= x0_252 + x0_253 + x0_254 + x0_255;

i1_0: lut_h4
       port map(
       address => xin_0, result => x1_0
);
i1_1: lut_h3
       port map(
       address => xin_1, result => x1_1
);
i1_2: lut_h2
       port map(
       address => xin_2, result => x1_2
);
i1_3: lut_h1
       port map(
       address => xin_3, result => x1_3
);
yh_0 <= x1_2 + x1_1 + x1_0 + x1_3;

i1_4: lut_h4
       port map(
       address => xin_2, result => x1_4
);
i1_5: lut_h3
       port map(
       address => xin_3, result => x1_5
);
i1_6: lut_h2
       port map(
       address => xin_4, result => x1_6
);
i1_7: lut_h1
       port map(
       address => xin_5, result => x1_7
);
yh_1 <= x1_6 + x1_5 + x1_4 + x1_7;

i1_8: lut_h4
       port map(
       address => xin_4, result => x1_8
```

```
);
i1_9: lut_h3
       port map(
       address => xin_5, result => x1_9
);
i1_10: lut_h2
       port map(
       address => xin_6, result => x1_10
);
i1_11: lut_h1
       port map(
       address => xin_7, result => x1_11
);
yh_2 <= x1_10 + x1_9 + x1_8 + x1_11;

i1_12: lut_h4
       port map(
       address => xin_6, result => x1_12
);
i1_13: lut_h3
       port map(
       address => xin_7, result => x1_13
);
i1_14: lut_h2
       port map(
       address => xin_8, result => x1_14
);
i1_15: lut_h1
       port map(
       address => xin_9, result => x1_15
);
yh_3 <= x1_14 + x1_13 + x1_12 + x1_15;

i1_16: lut_h4
       port map(
       address => xin_8, result => x1_16
);
i1_17: lut_h3
       port map(
       address => xin_9, result => x1_17
);
i1_18: lut_h2
       port map(
       address => xin_10, result => x1_18
);
i1_19: lut_h1
       port map(
       address => xin_11, result => x1_19
);
yh_4 <= x1_18 + x1_17 + x1_16 + x1_19;

i1_20: lut_h4
       port map(
       address => xin_10, result => x1_20
);
i1_21: lut_h3
       port map(
       address => xin_11, result => x1_21
);
i1_22: lut_h2
       port map(
       address => xin_12, result => x1_22
);
i1_23: lut_h1
```

```
        port map(
        address => xin_13, result => x1_23
);
yh_5 <= x1_22 + x1_21 + x1_20 + x1_23;

i1_24: lut_h4
        port map(
        address => xin_12, result => x1_24
);
i1_25: lut_h3
        port map(
        address => xin_13, result => x1_25
);
i1_26: lut_h2
        port map(
        address => xin_14, result => x1_26
);
i1_27: lut_h1
        port map(
        address => xin_15, result => x1_27
);
yh_6 <= x1_26 + x1_25 + x1_24 + x1_27;

i1_28: lut_h4
        port map(
        address => xin_14, result => x1_28
);
i1_29: lut_h3
        port map(
        address => xin_15, result => x1_29
);
i1_30: lut_h2
        port map(
        address => xin_16, result => x1_30
);
i1_31: lut_h1
        port map(
        address => xin_17, result => x1_31
);
yh_7 <= x1_30 + x1_29 + x1_28 + x1_31;

i1_32: lut_h4
        port map(
        address => xin_16, result => x1_32
);
i1_33: lut_h3
        port map(
        address => xin_17, result => x1_33
);
i1_34: lut_h2
        port map(
        address => xin_18, result => x1_34
);
i1_35: lut_h1
        port map(
        address => xin_19, result => x1_35
);
yh_8 <= x1_34 + x1_33 + x1_32 + x1_35;

i1_36: lut_h4
        port map(
        address => xin_18, result => x1_36
);
i1_37: lut_h3
```

```
        port map(
        address => xin_19, result => x1_37
);
i1_38: lut_h2
        port map(
        address => xin_20, result => x1_38
);
i1_39: lut_h1
        port map(
        address => xin_21, result => x1_39
);
yh_9 <= x1_38 + x1_37 + x1_36 + x1_39;

i1_40: lut_h4
        port map(
        address => xin_20, result => x1_40
);
i1_41: lut_h3
        port map(
        address => xin_21, result => x1_41
);
i1_42: lut_h2
        port map(
        address => xin_22, result => x1_42
);
i1_43: lut_h1
        port map(
        address => xin_23, result => x1_43
);
yh_10 <= x1_42 + x1_41 + x1_40 + x1_43;

i1_44: lut_h4
        port map(
        address => xin_22, result => x1_44
);
i1_45: lut_h3
        port map(
        address => xin_23, result => x1_45
);
i1_46: lut_h2
        port map(
        address => xin_24, result => x1_46
);
i1_47: lut_h1
        port map(
        address => xin_25, result => x1_47
);
yh_11 <= x1_46 + x1_45 + x1_44 + x1_47;

i1_48: lut_h4
        port map(
        address => xin_24, result => x1_48
);
i1_49: lut_h3
        port map(
        address => xin_25, result => x1_49
);
i1_50: lut_h2
        port map(
        address => xin_26, result => x1_50
);
i1_51: lut_h1
        port map(
        address => xin_27, result => x1_51
```

```
);
yh_12 <= x1_50 + x1_49 + x1_48 + x1_51;

i1_52: lut_h4
       port map(
       address => xin_26, result => x1_52
);
i1_53: lut_h3
       port map(
       address => xin_27, result => x1_53
);
i1_54: lut_h2
       port map(
       address => xin_28, result => x1_54
);
i1_55: lut_h1
       port map(
       address => xin_29, result => x1_55
);
yh_13 <= x1_54 + x1_53 + x1_52 + x1_55;

i1_56: lut_h4
       port map(
       address => xin_28, result => x1_56
);
i1_57: lut_h3
       port map(
       address => xin_29, result => x1_57
);
i1_58: lut_h2
       port map(
       address => xin_30, result => x1_58
);
i1_59: lut_h1
       port map(
       address => xin_31, result => x1_59
);
yh_14 <= x1_58 + x1_57 + x1_56 + x1_59;

i1_60: lut_h4
       port map(
       address => xin_30, result => x1_60
);
i1_61: lut_h3
       port map(
       address => xin_31, result => x1_61
);
i1_62: lut_h2
       port map(
       address => xin_32, result => x1_62
);
i1_63: lut_h1
       port map(
       address => xin_33, result => x1_63
);
yh_15 <= x1_62 + x1_61 + x1_60 + x1_63;

i1_64: lut_h4
       port map(
       address => xin_32, result => x1_64
);
i1_65: lut_h3
       port map(
       address => xin_33, result => x1_65
```

```
);
i1_66: lut_h2
       port map(
       address => xin_34, result => x1_66
);
i1_67: lut_h1
       port map(
       address => xin_35, result => x1_67
);
yh_16 <= x1_66 + x1_65 + x1_64 + x1_67;

i1_68: lut_h4
       port map(
       address => xin_34, result => x1_68
);
i1_69: lut_h3
       port map(
       address => xin_35, result => x1_69
);
i1_70: lut_h2
       port map(
       address => xin_36, result => x1_70
);
i1_71: lut_h1
       port map(
       address => xin_37, result => x1_71
);
yh_17 <= x1_70 + x1_69 + x1_68 + x1_71;

i1_72: lut_h4
       port map(
       address => xin_36, result => x1_72
);
i1_73: lut_h3
       port map(
       address => xin_37, result => x1_73
);
i1_74: lut_h2
       port map(
       address => xin_38, result => x1_74
);
i1_75: lut_h1
       port map(
       address => xin_39, result => x1_75
);
yh_18 <= x1_74 + x1_73 + x1_72 + x1_75;

i1_76: lut_h4
       port map(
       address => xin_38, result => x1_76
);
i1_77: lut_h3
       port map(
       address => xin_39, result => x1_77
);
i1_78: lut_h2
       port map(
       address => xin_40, result => x1_78
);
i1_79: lut_h1
       port map(
       address => xin_41, result => x1_79
);
yh_19 <= x1_78 + x1_77 + x1_76 + x1_79;
```

```
i1_80: lut_h4
       port map(
       address => xin_40, result => x1_80
);
i1_81: lut_h3
       port map(
       address => xin_41, result => x1_81
);
i1_82: lut_h2
       port map(
       address => xin_42, result => x1_82
);
i1_83: lut_h1
       port map(
       address => xin_43, result => x1_83
);
yh_20 <= x1_82 + x1_81 + x1_80 + x1_83;

i1_84: lut_h4
       port map(
       address => xin_42, result => x1_84
);
i1_85: lut_h3
       port map(
       address => xin_43, result => x1_85
);
i1_86: lut_h2
       port map(
       address => xin_44, result => x1_86
);
i1_87: lut_h1
       port map(
       address => xin_45, result => x1_87
);
yh_21 <= x1_86 + x1_85 + x1_84 + x1_87;

i1_88: lut_h4
       port map(
       address => xin_44, result => x1_88
);
i1_89: lut_h3
       port map(
       address => xin_45, result => x1_89
);
i1_90: lut_h2
       port map(
       address => xin_46, result => x1_90
);
i1_91: lut_h1
       port map(
       address => xin_47, result => x1_91
);
yh_22 <= x1_90 + x1_89 + x1_88 + x1_91;

i1_92: lut_h4
       port map(
       address => xin_46, result => x1_92
);
i1_93: lut_h3
       port map(
       address => xin_47, result => x1_93
);
i1_94: lut_h2
```

```
        port map(
        address => xin_48, result => x1_94
);
i1_95: lut_h1
        port map(
        address => xin_49, result => x1_95
);
yh_23 <= x1_94 + x1_93 + x1_92 + x1_95;

i1_96: lut_h4
        port map(
        address => xin_48, result => x1_96
);
i1_97: lut_h3
        port map(
        address => xin_49, result => x1_97
);
i1_98: lut_h2
        port map(
        address => xin_50, result => x1_98
);
i1_99: lut_h1
        port map(
        address => xin_51, result => x1_99
);
yh_24 <= x1_98 + x1_97 + x1_96 + x1_99;

i1_100: lut_h4
        port map(
        address => xin_50, result => x1_100
);
i1_101: lut_h3
        port map(
        address => xin_51, result => x1_101
);
i1_102: lut_h2
        port map(
        address => xin_52, result => x1_102
);
i1_103: lut_h1
        port map(
        address => xin_53, result => x1_103
);
yh_25 <= x1_102 + x1_101 + x1_100 + x1_103;

i1_104: lut_h4
        port map(
        address => xin_52, result => x1_104
);
i1_105: lut_h3
        port map(
        address => xin_53, result => x1_105
);
i1_106: lut_h2
        port map(
        address => xin_54, result => x1_106
);
i1_107: lut_h1
        port map(
        address => xin_55, result => x1_107
);
yh_26 <= x1_106 + x1_105 + x1_104 + x1_107;

i1_108: lut_h4
```

```
        port map(
        address => xin_54, result => x1_108
);
i1_109: lut_h3
        port map(
        address => xin_55, result => x1_109
);
i1_110: lut_h2
        port map(
        address => xin_56, result => x1_110
);
i1_111: lut_h1
        port map(
        address => xin_57, result => x1_111
);
yh_27 <= x1_110 + x1_109 + x1_108 + x1_111;

i1_112: lut_h4
        port map(
        address => xin_56, result => x1_112
);
i1_113: lut_h3
        port map(
        address => xin_57, result => x1_113
);
i1_114: lut_h2
        port map(
        address => xin_58, result => x1_114
);
i1_115: lut_h1
        port map(
        address => xin_59, result => x1_115
);
yh_28 <= x1_114 + x1_113 + x1_112 + x1_115;

i1_116: lut_h4
        port map(
        address => xin_58, result => x1_116
);
i1_117: lut_h3
        port map(
        address => xin_59, result => x1_117
);
i1_118: lut_h2
        port map(
        address => xin_60, result => x1_118
);
i1_119: lut_h1
        port map(
        address => xin_61, result => x1_119
);
yh_29 <= x1_118 + x1_117 + x1_116 + x1_119;

i1_120: lut_h4
        port map(
        address => xin_60, result => x1_120
);
i1_121: lut_h3
        port map(
        address => xin_61, result => x1_121
);
i1_122: lut_h2
        port map(
        address => xin_62, result => x1_122
```

```
);
i1_123: lut_h1
       port map(
       address => xin_63, result => x1_123
);
yh_30 <= x1_122 + x1_121 + x1_120 + x1_123;

i1_124: lut_h4
       port map(
       address => xin_62, result => x1_124
);
i1_125: lut_h3
       port map(
       address => xin_63, result => x1_125
);
i1_126: lut_h2
       port map(
       address => xin_64, result => x1_126
);
i1_127: lut_h1
       port map(
       address => xin_65, result => x1_127
);
yh_31 <= x1_126 + x1_125 + x1_124 + x1_127;

i1_128: lut_h4
       port map(
       address => xin_64, result => x1_128
);
i1_129: lut_h3
       port map(
       address => xin_65, result => x1_129
);
i1_130: lut_h2
       port map(
       address => xin_66, result => x1_130
);
i1_131: lut_h1
       port map(
       address => xin_67, result => x1_131
);
yh_32 <= x1_130 + x1_129 + x1_128 + x1_131;

i1_132: lut_h4
       port map(
       address => xin_66, result => x1_132
);
i1_133: lut_h3
       port map(
       address => xin_67, result => x1_133
);
i1_134: lut_h2
       port map(
       address => xin_68, result => x1_134
);
i1_135: lut_h1
       port map(
       address => xin_69, result => x1_135
);
yh_33 <= x1_134 + x1_133 + x1_132 + x1_135;

i1_136: lut_h4
       port map(
       address => xin_68, result => x1_136
```

```
);
i1_137: lut_h3
       port map(
       address => xin_69, result => x1_137
);
i1_138: lut_h2
       port map(
       address => xin_70, result => x1_138
);
i1_139: lut_h1
       port map(
       address => xin_71, result => x1_139
);
yh_34 <= x1_138 + x1_137 + x1_136 + x1_139;

i1_140: lut_h4
       port map(
       address => xin_70, result => x1_140
);
i1_141: lut_h3
       port map(
       address => xin_71, result => x1_141
);
i1_142: lut_h2
       port map(
       address => xin_72, result => x1_142
);
i1_143: lut_h1
       port map(
       address => xin_73, result => x1_143
);
yh_35 <= x1_142 + x1_141 + x1_140 + x1_143;

i1_144: lut_h4
       port map(
       address => xin_72, result => x1_144
);
i1_145: lut_h3
       port map(
       address => xin_73, result => x1_145
);
i1_146: lut_h2
       port map(
       address => xin_74, result => x1_146
);
i1_147: lut_h1
       port map(
       address => xin_75, result => x1_147
);
yh_36 <= x1_146 + x1_145 + x1_144 + x1_147;

i1_148: lut_h4
       port map(
       address => xin_74, result => x1_148
);
i1_149: lut_h3
       port map(
       address => xin_75, result => x1_149
);
i1_150: lut_h2
       port map(
       address => xin_76, result => x1_150
);
i1_151: lut_h1
```

```
        port map(
        address => xin_77, result => x1_151
);
yh_37 <= x1_150 + x1_149 + x1_148 + x1_151;

i1_152: lut_h4
        port map(
        address => xin_76, result => x1_152
);
i1_153: lut_h3
        port map(
        address => xin_77, result => x1_153
);
i1_154: lut_h2
        port map(
        address => xin_78, result => x1_154
);
i1_155: lut_h1
        port map(
        address => xin_79, result => x1_155
);
yh_38 <= x1_154 + x1_153 + x1_152 + x1_155;

i1_156: lut_h4
        port map(
        address => xin_78, result => x1_156
);
i1_157: lut_h3
        port map(
        address => xin_79, result => x1_157
);
i1_158: lut_h2
        port map(
        address => xin_80, result => x1_158
);
i1_159: lut_h1
        port map(
        address => xin_81, result => x1_159
);
yh_39 <= x1_158 + x1_157 + x1_156 + x1_159;

i1_160: lut_h4
        port map(
        address => xin_80, result => x1_160
);
i1_161: lut_h3
        port map(
        address => xin_81, result => x1_161
);
i1_162: lut_h2
        port map(
        address => xin_82, result => x1_162
);
i1_163: lut_h1
        port map(
        address => xin_83, result => x1_163
);
yh_40 <= x1_162 + x1_161 + x1_160 + x1_163;

i1_164: lut_h4
        port map(
        address => xin_82, result => x1_164
);
i1_165: lut_h3
```

```
        port map(
        address => xin_83, result => x1_165
);
i1_166: lut_h2
        port map(
        address => xin_84, result => x1_166
);
i1_167: lut_h1
        port map(
        address => xin_85, result => x1_167
);
yh_41 <= x1_166 + x1_165 + x1_164 + x1_167;

i1_168: lut_h4
        port map(
        address => xin_84, result => x1_168
);
i1_169: lut_h3
        port map(
        address => xin_85, result => x1_169
);
i1_170: lut_h2
        port map(
        address => xin_86, result => x1_170
);
i1_171: lut_h1
        port map(
        address => xin_87, result => x1_171
);
yh_42 <= x1_170 + x1_169 + x1_168 + x1_171;

i1_172: lut_h4
        port map(
        address => xin_86, result => x1_172
);
i1_173: lut_h3
        port map(
        address => xin_87, result => x1_173
);
i1_174: lut_h2
        port map(
        address => xin_88, result => x1_174
);
i1_175: lut_h1
        port map(
        address => xin_89, result => x1_175
);
yh_43 <= x1_174 + x1_173 + x1_172 + x1_175;

i1_176: lut_h4
        port map(
        address => xin_88, result => x1_176
);
i1_177: lut_h3
        port map(
        address => xin_89, result => x1_177
);
i1_178: lut_h2
        port map(
        address => xin_90, result => x1_178
);
i1_179: lut_h1
        port map(
        address => xin_91, result => x1_179
```

```
);
yh_44 <= x1_178 + x1_177 + x1_176 + x1_179;

i1_180: lut_h4
       port map(
       address => xin_90, result => x1_180
);
i1_181: lut_h3
       port map(
       address => xin_91, result => x1_181
);
i1_182: lut_h2
       port map(
       address => xin_92, result => x1_182
);
i1_183: lut_h1
       port map(
       address => xin_93, result => x1_183
);
yh_45 <= x1_182 + x1_181 + x1_180 + x1_183;

i1_184: lut_h4
       port map(
       address => xin_92, result => x1_184
);
i1_185: lut_h3
       port map(
       address => xin_93, result => x1_185
);
i1_186: lut_h2
       port map(
       address => xin_94, result => x1_186
);
i1_187: lut_h1
       port map(
       address => xin_95, result => x1_187
);
yh_46 <= x1_186 + x1_185 + x1_184 + x1_187;

i1_188: lut_h4
       port map(
       address => xin_94, result => x1_188
);
i1_189: lut_h3
       port map(
       address => xin_95, result => x1_189
);
i1_190: lut_h2
       port map(
       address => xin_96, result => x1_190
);
i1_191: lut_h1
       port map(
       address => xin_97, result => x1_191
);
yh_47 <= x1_190 + x1_189 + x1_188 + x1_191;

i1_192: lut_h4
       port map(
       address => xin_96, result => x1_192
);
i1_193: lut_h3
       port map(
       address => xin_97, result => x1_193
```

```
);
i1_194: lut_h2
       port map(
       address => xin_98, result => x1_194
);
i1_195: lut_h1
       port map(
       address => xin_99, result => x1_195
);
yh_48 <= x1_194 + x1_193 + x1_192 + x1_195;

i1_196: lut_h4
       port map(
       address => xin_98, result => x1_196
);
i1_197: lut_h3
       port map(
       address => xin_99, result => x1_197
);
i1_198: lut_h2
       port map(
       address => xin_100, result => x1_198
);
i1_199: lut_h1
       port map(
       address => xin_101, result => x1_199
);
yh_49 <= x1_198 + x1_197 + x1_196 + x1_199;

i1_200: lut_h4
       port map(
       address => xin_100, result => x1_200
);
i1_201: lut_h3
       port map(
       address => xin_101, result => x1_201
);
i1_202: lut_h2
       port map(
       address => xin_102, result => x1_202
);
i1_203: lut_h1
       port map(
       address => xin_103, result => x1_203
);
yh_50 <= x1_202 + x1_201 + x1_200 + x1_203;

i1_204: lut_h4
       port map(
       address => xin_102, result => x1_204
);
i1_205: lut_h3
       port map(
       address => xin_103, result => x1_205
);
i1_206: lut_h2
       port map(
       address => xin_104, result => x1_206
);
i1_207: lut_h1
       port map(
       address => xin_105, result => x1_207
);
yh_51 <= x1_206 + x1_205 + x1_204 + x1_207;
```

```
i1_208: lut_h4
       port map(
       address => xin_104, result => x1_208
);
i1_209: lut_h3
       port map(
       address => xin_105, result => x1_209
);
i1_210: lut_h2
       port map(
       address => xin_106, result => x1_210
);
i1_211: lut_h1
       port map(
       address => xin_107, result => x1_211
);
yh_52 <= x1_210 + x1_209 + x1_208 + x1_211;

i1_212: lut_h4
       port map(
       address => xin_106, result => x1_212
);
i1_213: lut_h3
       port map(
       address => xin_107, result => x1_213
);
i1_214: lut_h2
       port map(
       address => xin_108, result => x1_214
);
i1_215: lut_h1
       port map(
       address => xin_109, result => x1_215
);
yh_53 <= x1_214 + x1_213 + x1_212 + x1_215;

i1_216: lut_h4
       port map(
       address => xin_108, result => x1_216
);
i1_217: lut_h3
       port map(
       address => xin_109, result => x1_217
);
i1_218: lut_h2
       port map(
       address => xin_110, result => x1_218
);
i1_219: lut_h1
       port map(
       address => xin_111, result => x1_219
);
yh_54 <= x1_218 + x1_217 + x1_216 + x1_219;

i1_220: lut_h4
       port map(
       address => xin_110, result => x1_220
);
i1_221: lut_h3
       port map(
       address => xin_111, result => x1_221
);
i1_222: lut_h2
```

```
        port map(
        address => xin_112, result => x1_222
);
i1_223: lut_h1
        port map(
        address => xin_113, result => x1_223
);
yh_55 <= x1_222 + x1_221 + x1_220 + x1_223;

i1_224: lut_h4
        port map(
        address => xin_112, result => x1_224
);
i1_225: lut_h3
        port map(
        address => xin_113, result => x1_225
);
i1_226: lut_h2
        port map(
        address => xin_114, result => x1_226
);
i1_227: lut_h1
        port map(
        address => xin_115, result => x1_227
);
yh_56 <= x1_226 + x1_225 + x1_224 + x1_227;

i1_228: lut_h4
        port map(
        address => xin_114, result => x1_228
);
i1_229: lut_h3
        port map(
        address => xin_115, result => x1_229
);
i1_230: lut_h2
        port map(
        address => xin_116, result => x1_230
);
i1_231: lut_h1
        port map(
        address => xin_117, result => x1_231
);
yh_57 <= x1_230 + x1_229 + x1_228 + x1_231;

i1_232: lut_h4
        port map(
        address => xin_116, result => x1_232
);
i1_233: lut_h3
        port map(
        address => xin_117, result => x1_233
);
i1_234: lut_h2
        port map(
        address => xin_118, result => x1_234
);
i1_235: lut_h1
        port map(
        address => xin_119, result => x1_235
);
yh_58 <= x1_234 + x1_233 + x1_232 + x1_235;

i1_236: lut_h4
```

```
        port map(
        address => xin_118, result => x1_236
);
i1_237: lut_h3
        port map(
        address => xin_119, result => x1_237
);
i1_238: lut_h2
        port map(
        address => xin_120, result => x1_238
);
i1_239: lut_h1
        port map(
        address => xin_121, result => x1_239
);
yh_59 <= x1_238 + x1_237 + x1_236 + x1_239;

i1_240: lut_h4
        port map(
        address => xin_120, result => x1_240
);
i1_241: lut_h3
        port map(
        address => xin_121, result => x1_241
);
i1_242: lut_h2
        port map(
        address => xin_122, result => x1_242
);
i1_243: lut_h1
        port map(
        address => xin_123, result => x1_243
);
yh_60 <= x1_242 + x1_241 + x1_240 + x1_243;

i1_244: lut_h4
        port map(
        address => xin_122, result => x1_244
);
i1_245: lut_h3
        port map(
        address => xin_123, result => x1_245
);
i1_246: lut_h2
        port map(
        address => xin_124, result => x1_246
);
i1_247: lut_h1
        port map(
        address => xin_125, result => x1_247
);
yh_61 <= x1_246 + x1_245 + x1_244 + x1_247;

i1_248: lut_h4
        port map(
        address => xin_124, result => x1_248
);
i1_249: lut_h3
        port map(
        address => xin_125, result => x1_249
);
i1_250: lut_h2
        port map(
        address => xin_126, result => x1_250
```

```
);
i1_251: lut_h1
	port map(
	address => xin_127, result => x1_251
);
yh_62 <= x1_250 + x1_249 + x1_248 + x1_251;

i1_252: lut_h4
	port map(
	address => xin_126, result => x1_252
);
i1_253: lut_h3
	port map(
	address => xin_127, result => x1_253
);
i1_254: lut_h2
	port map(
	address => xin_0, result => x1_254
);
i1_255: lut_h1
	port map(
	address => xin_1, result => x1_255
);
yh_63 <= x1_254 + x1_253 + x1_252 + x1_255;

end architecture;

--Look up table for H1 coefficient calculations
-------------------------------------------------
-- This LUT is used to get the results of a bitwise multiplication of a number and the
coefficient
-- h1, which is the first coefficient of the DWT Daubechies wavelet transform.
-- h1 has the value of 0.483.  This is encoded by using 0.483 * 256, which will give a
number
-- which is rounded off so that it will give an integer.  This integer is the 8 bias
number for 0.483.

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity LUT_h1 is
	port(
		address : in std_logic_vector(7 downto 0);
		result : out std_logic_vector(8 downto 0)
		);
end entity;

architecture behavioral of LUT_h1 is
type LUT is array(0 to 255) of std_logic_vector(8 downto 0);
constant lut_h1calc : LUT := (
"000000000",
"000000000",
"000000000",
"000000001",
"000000001",
"000000010",
"000000010",
"000000011",
"000000011",
"000000100",
"000000100",
"000000101",
```

```
"000000101",
"000000110",
"000000110",
"000000111",
"000000111",
"000001000",
"000001000",
"000001001",
"000001001",
"000001010",
"000001010",
"000001011",
"000001011",
"000001100",
"000001100",
"000001101",
"000001101",
"000001110",
"000001110",
"000001111",
"000001111",
"000001111",
"000010000",
"000010000",
"000010001",
"000010001",
"000010010",
"000010010",
"000010011",
"000010011",
"000010100",
"000010100",
"000010101",
"000010101",
"000010110",
"000010110",
"000010111",
"000010111",
"000011000",
"000011000",
"000011001",
"000011001",
"000011010",
"000011010",
"000011011",
"000011011",
"000011100",
"000011100",
"000011101",
"000011101",
"000011110",
"000011110",
"000011111",
"000011111",
"000011111",
"000100000",
"000100000",
"000100001",
"000100001",
"000100010",
"000100010",
"000100011",
"000100011",
"000100100",
```

```
"000100100",
"000100101",
"000100101",
"000100110",
"000100110",
"000100111",
"000100111",
"000101000",
"000101000",
"000101001",
"000101001",
"000101010",
"000101010",
"000101011",
"000101011",
"000101100",
"000101100",
"000101101",
"000101101",
"000101110",
"000101110",
"000101110",
"000101111",
"000101111",
"000110000",
"000110000",
"000110001",
"000110001",
"000110010",
"000110010",
"000110011",
"000110011",
"000110100",
"000110100",
"000110101",
"000110101",
"000110110",
"000110110",
"000110111",
"000110111",
"000111000",
"000111000",
"000111001",
"000111001",
"000111010",
"000111010",
"000111011",
"000111011",
"000111100",
"000111100",
"000111101",
"000111101",
"000111110",
"000111110",
"000111110",
"000111111",
"000111111",
"001000000",
"001000000",
"001000001",
"001000001",
"001000010",
"001000010",
"001000011",
```

```
"001000011",
"001000100",
"001000100",
"001000101",
"001000101",
"001000110",
"001000110",
"001000111",
"001000111",
"001001000",
"001001000",
"001001001",
"001001001",
"001001010",
"001001010",
"001001011",
"001001011",
"001001100",
"001001100",
"001001101",
"001001101",
"001001101",
"001001110",
"001001110",
"001001111",
"001001111",
"001010000",
"001010000",
"001010001",
"001010001",
"001010010",
"001010010",
"001010011",
"001010011",
"001010100",
"001010100",
"001010101",
"001010101",
"001010110",
"001010110",
"001010111",
"001010111",
"001011000",
"001011000",
"001011001",
"001011001",
"001011010",
"001011010",
"001011011",
"001011011",
"001011100",
"001011100",
"001011101",
"001011101",
"001011101",
"001011110",
"001011110",
"001011111",
"001011111",
"001100000",
"001100000",
"001100001",
"001100001",
"001100010",
```

```
"001100010",
"001100011",
"001100011",
"001100100",
"001100100",
"001100101",
"001100101",
"001100110",
"001100110",
"001100111",
"001100111",
"001101000",
"001101000",
"001101001",
"001101001",
"001101010",
"001101010",
"001101011",
"001101011",
"001101100",
"001101100",
"001101100",
"001101101",
"001101101",
"001101110",
"001101110",
"001101111",
"001101111",
"001110000",
"001110000",
"001110001",
"001110001",
"001110010",
"001110010",
"001110011",
"001110011",
"001110100",
"001110100",
"001110101",
"001110101",
"001110110",
"001110110",
"001110111",
"001110111",
"001111000",
"001111000",
"001111001",
"001111001",
"001111010",
"001111010",
"001111011",
"001111011");
--signal address_int : std_logic_vector(7 downto 0);
begin
	process(address)
	variable address_int : std_logic_vector (7 downto 0);
	begin
		address_int := address;
		result <= lut_h1calc(conv_integer(address_int));
	end process;
end architecture;

configuration config_lut_h1 of lut_h1 is
	for behavioral
```

```
	end for;
end configuration;

--Look up table for H1 coefficient calculations(idwt)
-------------------------------------------------
-- This LUT is used to get the results of a bitwise multiplication of a number and the
coefficient
-- h1, which is the first coefficient of the DWT Daubechies wavelet transform.
-- h1 has the value of 0.483

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity lut_h1_idwt is
	port(
		address : in std_logic_vector(8 downto 0);
		result : out std_logic_vector(8 downto 0)
		);
end entity;


architecture behavioral of LUT_h1_idwt is
type lut is array(0 to 511) of std_logic_vector(8 downto 0);
constant lut_h1calc : lut := (
"000000000",
"000000000",
"000000000",
"000000001",
"000000001",
"000000010",
"000000010",
"000000011",
"000000011",
"000000100",
"000000100",
"000000101",
"000000101",
"000000110",
"000000110",
"000000111",
"000000111",
"000001000",
"000001000",
"000001001",
"000001001",
"000001010",
"000001010",
"000001011",
"000001011",
"000001100",
"000001100",
"000001101",
"000001101",
"000001110",
"000001110",
"000001110",
"000001111",
"000001111",
"000010000",
"000010000",
"000010001",
"000010001",
```

```
"000010010",
"000010010",
"000010011",
"000010011",
"000010100",
"000010100",
"000010101",
"000010101",
"000010110",
"000010110",
"000010111",
"000010111",
"000011000",
"000011000",
"000011001",
"000011001",
"000011010",
"000011010",
"000011011",
"000011011",
"000011100",
"000011100",
"000011100",
"000011101",
"000011101",
"000011110",
"000011110",
"000011111",
"000011111",
"000100000",
"000100000",
"000100001",
"000100001",
"000100010",
"000100010",
"000100011",
"000100011",
"000100100",
"000100100",
"000100101",
"000100101",
"000100110",
"000100110",
"000100111",
"000100111",
"000101000",
"000101000",
"000101001",
"000101001",
"000101010",
"000101010",
"000101010",
"000101011",
"000101011",
"000101100",
"000101100",
"000101101",
"000101101",
"000101110",
"000101110",
"000101111",
"000101111",
"000110000",
"000110000",
```

```
"000110001",
"000110001",
"000110010",
"000110010",
"000110011",
"000110011",
"000110100",
"000110100",
"000110101",
"000110101",
"000110110",
"000110110",
"000110111",
"000110111",
"000111000",
"000111000",
"000111000",
"000111001",
"000111001",
"000111010",
"000111010",
"000111011",
"000111011",
"000111100",
"000111100",
"000111101",
"000111101",
"000111110",
"000111110",
"000111111",
"000111111",
"001000000",
"001000000",
"001000001",
"001000001",
"001000010",
"001000010",
"001000011",
"001000011",
"001000100",
"001000100",
"001000101",
"001000101",
"001000110",
"001000110",
"001000111",
"001000111",
"001000111",
"001001000",
"001001000",
"001001001",
"001001001",
"001001010",
"001001010",
"001001011",
"001001011",
"001001100",
"001001100",
"001001101",
"001001101",
"001001110",
"001001110",
"001001111",
"001001111",
```

```
"001010000",
"001010000",
"001010001",
"001010001",
"001010010",
"001010010",
"001010011",
"001010011",
"001010100",
"001010100",
"001010101",
"001010101",
"001010101",
"001010110",
"001010110",
"001010111",
"001010111",
"001011000",
"001011000",
"001011001",
"001011001",
"001011010",
"001011010",
"001011011",
"001011011",
"001011100",
"001011100",
"001011101",
"001011101",
"001011110",
"001011110",
"001011111",
"001011111",
"001100000",
"001100000",
"001100001",
"001100001",
"001100010",
"001100010",
"001100011",
"001100011",
"001100011",
"001100100",
"001100100",
"001100101",
"001100101",
"001100110",
"001100110",
"001100111",
"001100111",
"001101000",
"001101000",
"001101001",
"001101001",
"001101010",
"001101010",
"001101011",
"001101011",
"001101100",
"001101100",
"001101101",
"001101101",
"001101110",
"001101110",
```

```
"001101111",
"001101111",
"001110000",
"001110000",
"001110001",
"001110001",
"001110001",
"001110010",
"001110010",
"001110011",
"001110011",
"001110100",
"001110100",
"001110101",
"001110101",
"001110110",
"001110110",
"001110111",
"001110111",
"001111000",
"001111000",
"001111001",
"001111001",
"001111010",
"001111010",
"001111011",--
"000000000",
"110000100",
"110000101",
"110000101",
"110000110",
"110000110",
"110000111",
"110000111",
"110001000",
"110001000",
"110001001",
"110001001",
"110001010",
"110001010",
"110001011",
"110001011",
"110001100",
"110001100",
"110001101",
"110001101",
"110001110",
"110001110",
"110001110",
"110001111",
"110001111",
"110010000",
"110010000",
"110010001",
"110010001",
"110010010",
"110010010",
"110010011",
"110010011",
"110010100",
"110010100",
"110010101",
"110010101",
"110010110",
```

```
"110010110",
"110010111",
"110010111",
"110011000",
"110011000",
"110011001",
"110011001",
"110011010",
"110011010",
"110011011",
"110011011",
"110011100",
"110011100",
"110011100",
"110011101",
"110011101",
"110011110",
"110011110",
"110011111",
"110011111",
"110100000",
"110100000",
"110100001",
"110100001",
"110100010",
"110100010",
"110100011",
"110100011",
"110100100",
"110100100",
"110100101",
"110100101",
"110100110",
"110100110",
"110100111",
"110100111",
"110101000",
"110101000",
"110101001",
"110101001",
"110101010",
"110101010",
"110101010",
"110101011",
"110101011",
"110101100",
"110101100",
"110101101",
"110101101",
"110101110",
"110101110",
"110101111",
"110101111",
"110110000",
"110110000",
"110110001",
"110110001",
"110110010",
"110110010",
"110110011",
"110110011",
"110110100",
"110110100",
"110110101",
```

```
"110110101",
"110110110",
"110110110",
"110110111",
"110110111",
"110111000",
"110111000",
"110111000",
"110111001",
"110111001",
"110111010",
"110111010",
"110111011",
"110111011",
"110111100",
"110111100",
"110111101",
"110111101",
"110111110",
"110111110",
"110111111",
"110111111",
"111000000",
"111000000",
"111000001",
"111000001",
"111000010",
"111000010",
"111000011",
"111000011",
"111000100",
"111000100",
"111000101",
"111000101",
"111000110",
"111000110",
"111000111",
"111000111",
"111000111",
"111001000",
"111001000",
"111001001",
"111001001",
"111001010",
"111001010",
"111001011",
"111001011",
"111001100",
"111001100",
"111001101",
"111001101",
"111001110",
"111001110",
"111001111",
"111001111",
"111010000",
"111010000",
"111010001",
"111010001",
"111010010",
"111010010",
"111010011",
"111010011",
"111010100",
```

```
"111010100",
"111010101",
"111010101",
"111010101",
"111010110",
"111010110",
"111010110",
"111010111",
"111010111",
"111010111",
"111011000",
"111011000",
"111011000",
"111011001",
"111011001",
"111011010",
"111011010",
"111011011",
"111011011",
"111011100",
"111011100",
"111011101",
"111011101",
"111011110",
"111011110",
"111011111",
"111011111",
"111100000",
"111100000",
"111100000",
"111100001",
"111100001",
"111100010",
"111100010",
"111100011",
"111100011",
"111100011",
"111100100",
"111100100",
"111100101",
"111100101",
"111100110",
"111100110",
"111100111",
"111100111",
"111101000",
"111101000",
"111101001",
"111101001",
"111101010",
"111101010",
"111101011",
"111101011",
"111101100",
"111101100",
"111101101",
"111101101",
"111101110",
"111101110",
"111101111",
"111101111",
"111110000",
"111110000",
"111110001",
"111110001",
"111110001",
"111110010",
"111110010",
```

```
"111110011",
"111110011",
"111110100",
"111110100",
"111110101",
"111110101",
"111110110",
"111110110",
"111110111",
"111110111",
"111111000",
"111111000",
"111111001",
"111111001",
"111111010",
"111111010",
"111111011",
"111111011",
"111111100",
"111111100",
"111111101",
"111111101",
"111111110",
"111111110",
"111111111",
"111111111");
--signal address_int : std_logic_vector(7 downto 0);
begin
	process(address)
	variable address_int : std_logic_vector (8 downto 0);
	begin
		address_int := address;
		if (conv_integer(address_int) > 256) then
			result <= lut_h1calc(conv_integer(address_int)) + "000000001";
		else
			result <= lut_h1calc(conv_integer(address_int));
		end if;
	end process;
end architecture;

configuration config_lut_h1 of lut_h1_idwt is
	for behavioral
	end for;
end configuration;

--threshold estimator using LUT design
-- has an implied threshold of 1.6
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity threshold_estimator is
	port(
		x0_in : in std_logic_vector(8 downto 0);
		x1_in : in std_logic_vector(8 downto 0);
		x2_in : in std_logic_vector(8 downto 0);
		x3_in : in std_logic_vector(8 downto 0);
		x4_in : in std_logic_vector(8 downto 0);
		x5_in : in std_logic_vector(8 downto 0);
		x6_in : in std_logic_vector(8 downto 0);
		x7_in : in std_logic_vector(8 downto 0);
		x8_in : in std_logic_vector(8 downto 0);
		x9_in : in std_logic_vector(8 downto 0);
```

```
        x10_in : in std_logic_vector(8 downto 0);
        x11_in : in std_logic_vector(8 downto 0);
        x12_in : in std_logic_vector(8 downto 0);
        x13_in : in std_logic_vector(8 downto 0);
        x14_in : in std_logic_vector(8 downto 0);
        x15_in : in std_logic_vector(8 downto 0);
        x16_in : in std_logic_vector(8 downto 0);
        x17_in : in std_logic_vector(8 downto 0);
        x18_in : in std_logic_vector(8 downto 0);
        x19_in : in std_logic_vector(8 downto 0);
        x20_in : in std_logic_vector(8 downto 0);
        x21_in : in std_logic_vector(8 downto 0);
        x22_in : in std_logic_vector(8 downto 0);
        x23_in : in std_logic_vector(8 downto 0);
        x24_in : in std_logic_vector(8 downto 0);
        x25_in : in std_logic_vector(8 downto 0);
        x26_in : in std_logic_vector(8 downto 0);
        x27_in : in std_logic_vector(8 downto 0);
        x28_in : in std_logic_vector(8 downto 0);
        x29_in : in std_logic_vector(8 downto 0);
        x30_in : in std_logic_vector(8 downto 0);
        x31_in : in std_logic_vector(8 downto 0);
        x32_in : in std_logic_vector(8 downto 0);
        x33_in : in std_logic_vector(8 downto 0);
        x34_in : in std_logic_vector(8 downto 0);
        x35_in : in std_logic_vector(8 downto 0);
        x36_in : in std_logic_vector(8 downto 0);
        x37_in : in std_logic_vector(8 downto 0);
        x38_in : in std_logic_vector(8 downto 0);
        x39_in : in std_logic_vector(8 downto 0);
        x40_in : in std_logic_vector(8 downto 0);
        x41_in : in std_logic_vector(8 downto 0);
        x42_in : in std_logic_vector(8 downto 0);
        x43_in : in std_logic_vector(8 downto 0);
        x44_in : in std_logic_vector(8 downto 0);
        x45_in : in std_logic_vector(8 downto 0);
        x46_in : in std_logic_vector(8 downto 0);
        x47_in : in std_logic_vector(8 downto 0);
        x48_in : in std_logic_vector(8 downto 0);
        x49_in : in std_logic_vector(8 downto 0);
        x50_in : in std_logic_vector(8 downto 0);
        x51_in : in std_logic_vector(8 downto 0);
        x52_in : in std_logic_vector(8 downto 0);
        x53_in : in std_logic_vector(8 downto 0);
        x54_in : in std_logic_vector(8 downto 0);
        x55_in : in std_logic_vector(8 downto 0);
        x56_in : in std_logic_vector(8 downto 0);
        x57_in : in std_logic_vector(8 downto 0);
        x58_in : in std_logic_vector(8 downto 0);
        x59_in : in std_logic_vector(8 downto 0);
        x60_in : in std_logic_vector(8 downto 0);
        x61_in : in std_logic_vector(8 downto 0);
        x62_in : in std_logic_vector(8 downto 0);
        x63_in : in std_logic_vector(8 downto 0);
        thld_est : out std_logic_vector(8 downto 0)
        );
end entity;

architecture behavioral of threshold_estimator is

component lut_threshold
        port(
                address : in std_logic_vector(7 downto 0);
                result : out std_logic_vector(8 downto 0)
```

```
                );
        end component;
        signal x_median : std_logic_vector(7 downto 0);
        type xd_array is array (0 to 63) of integer;
        begin
        process(
                x0_in, x1_in, x2_in, x3_in, x4_in, x5_in, x6_in, x7_in,
                x8_in, x9_in, x10_in, x11_in, x12_in, x13_in, x14_in,
                x15_in, x16_in, x17_in, x18_in, x19_in, x20_in, x21_in,
                x22_in, x23_in, x24_in, x25_in, x26_in, x27_in, x28_in,
                x29_in, x30_in, x31_in, x32_in, x33_in, x34_in, x35_in,
                x36_in, x37_in, x38_in, x39_in, x40_in, x41_in, x42_in,
                x43_in, x44_in, x45_in, x46_in, x47_in, x48_in, x49_in,
                x50_in, x51_in, x52_in, x53_in, x54_in, x55_in, x56_in,
                x57_in, x58_in, x59_in, x60_in, x61_in, x62_in, x63_in
                )
variable temp: integer;
variable xd : xd_array;
begin
        xd(0) := abs(conv_integer(x0_in));
        xd(1) := abs(conv_integer(x1_in));
        xd(2) := abs(conv_integer(x2_in));
        xd(3) := abs(conv_integer(x3_in));
        xd(4) := abs(conv_integer(x4_in));
        xd(5) := abs(conv_integer(x5_in));
        xd(6) := abs(conv_integer(x6_in));
        xd(7) := abs(conv_integer(x7_in));
        xd(8) := abs(conv_integer(x8_in));
        xd(9) := abs(conv_integer(x9_in));
        xd(10) := abs(conv_integer(x10_in));
        xd(11) := abs(conv_integer(x11_in));
        xd(12) := abs(conv_integer(x12_in));
        xd(13) := abs(conv_integer(x13_in));
        xd(14) := abs(conv_integer(x14_in));
        xd(15) := abs(conv_integer(x15_in));
        xd(16) := abs(conv_integer(x16_in));
        xd(17) := abs(conv_integer(x17_in));
        xd(18) := abs(conv_integer(x18_in));
        xd(19) := abs(conv_integer(x19_in));
        xd(20) := abs(conv_integer(x20_in));
        xd(21) := abs(conv_integer(x21_in));
        xd(22) := abs(conv_integer(x22_in));
        xd(23) := abs(conv_integer(x23_in));
        xd(24) := abs(conv_integer(x24_in));
        xd(25) := abs(conv_integer(x25_in));
        xd(26) := abs(conv_integer(x26_in));
        xd(27) := abs(conv_integer(x27_in));
        xd(28) := abs(conv_integer(x28_in));
        xd(29) := abs(conv_integer(x29_in));
        xd(30) := abs(conv_integer(x30_in));
        xd(31) := abs(conv_integer(x31_in));
        xd(32) := abs(conv_integer(x32_in));
        xd(33) := abs(conv_integer(x33_in));
        xd(34) := abs(conv_integer(x34_in));
        xd(35) := abs(conv_integer(x35_in));
        xd(36) := abs(conv_integer(x36_in));
        xd(37) := abs(conv_integer(x37_in));
        xd(38) := abs(conv_integer(x38_in));
        xd(39) := abs(conv_integer(x39_in));
        xd(40) := abs(conv_integer(x40_in));
        xd(41) := abs(conv_integer(x41_in));
        xd(42) := abs(conv_integer(x42_in));
        xd(43) := abs(conv_integer(x43_in));
        xd(44) := abs(conv_integer(x44_in));
```

```
        xd(45) := abs(conv_integer(x45_in));
        xd(46) := abs(conv_integer(x46_in));
        xd(47) := abs(conv_integer(x47_in));
        xd(48) := abs(conv_integer(x48_in));
        xd(49) := abs(conv_integer(x49_in));
        xd(50) := abs(conv_integer(x50_in));
        xd(51) := abs(conv_integer(x51_in));
        xd(52) := abs(conv_integer(x52_in));
        xd(53) := abs(conv_integer(x53_in));
        xd(54) := abs(conv_integer(x54_in));
        xd(55) := abs(conv_integer(x55_in));
        xd(56) := abs(conv_integer(x56_in));
        xd(57) := abs(conv_integer(x57_in));
        xd(58) := abs(conv_integer(x58_in));
        xd(59) := abs(conv_integer(x59_in));
        xd(60) := abs(conv_integer(x60_in));
        xd(61) := abs(conv_integer(x61_in));
        xd(62) := abs(conv_integer(x62_in));
        xd(63) := abs(conv_integer(x63_in));

        -- Sort the array
        for i in 63 downto 0 loop
                for j in 1 to i loop
                        if xd(j-1) > xd(j) then
                                temp := xd(j-1);
                                xd(j-1) := xd(j);
                                xd(j) := temp;
                        else
                                --for eliminating other hardware.
                        end if;
                end loop;
        end loop;
        x_median <= conv_std_logic_vector(xd(32),8);
        end process;
        instance_threshold: lut_threshold
                port map(
                        address => x_median, result => thld_est
                        );
end architecture;

--Daubechies Inverse Discrete Wavelet Transform
-- Code is generated using C.
-- This design is based on the Look up tables for multiplication.

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity idwt is
        port(
                xout_0 : out std_logic_vector(8 downto 0);
                xout_1 : out std_logic_vector(8 downto 0);
                xout_2 : out std_logic_vector(8 downto 0);
                xout_3 : out std_logic_vector(8 downto 0);
                xout_4 : out std_logic_vector(8 downto 0);
                xout_5 : out std_logic_vector(8 downto 0);
                xout_6 : out std_logic_vector(8 downto 0);
                xout_7 : out std_logic_vector(8 downto 0);
                xout_8 : out std_logic_vector(8 downto 0);
                xout_9 : out std_logic_vector(8 downto 0);
                xout_10 : out std_logic_vector(8 downto 0);
                xout_11 : out std_logic_vector(8 downto 0);
                xout_12 : out std_logic_vector(8 downto 0);
```

```
xout_13 : out std_logic_vector(8 downto 0);
xout_14 : out std_logic_vector(8 downto 0);
xout_15 : out std_logic_vector(8 downto 0);
xout_16 : out std_logic_vector(8 downto 0);
xout_17 : out std_logic_vector(8 downto 0);
xout_18 : out std_logic_vector(8 downto 0);
xout_19 : out std_logic_vector(8 downto 0);
xout_20 : out std_logic_vector(8 downto 0);
xout_21 : out std_logic_vector(8 downto 0);
xout_22 : out std_logic_vector(8 downto 0);
xout_23 : out std_logic_vector(8 downto 0);
xout_24 : out std_logic_vector(8 downto 0);
xout_25 : out std_logic_vector(8 downto 0);
xout_26 : out std_logic_vector(8 downto 0);
xout_27 : out std_logic_vector(8 downto 0);
xout_28 : out std_logic_vector(8 downto 0);
xout_29 : out std_logic_vector(8 downto 0);
xout_30 : out std_logic_vector(8 downto 0);
xout_31 : out std_logic_vector(8 downto 0);
xout_32 : out std_logic_vector(8 downto 0);
xout_33 : out std_logic_vector(8 downto 0);
xout_34 : out std_logic_vector(8 downto 0);
xout_35 : out std_logic_vector(8 downto 0);
xout_36 : out std_logic_vector(8 downto 0);
xout_37 : out std_logic_vector(8 downto 0);
xout_38 : out std_logic_vector(8 downto 0);
xout_39 : out std_logic_vector(8 downto 0);
xout_40 : out std_logic_vector(8 downto 0);
xout_41 : out std_logic_vector(8 downto 0);
xout_42 : out std_logic_vector(8 downto 0);
xout_43 : out std_logic_vector(8 downto 0);
xout_44 : out std_logic_vector(8 downto 0);
xout_45 : out std_logic_vector(8 downto 0);
xout_46 : out std_logic_vector(8 downto 0);
xout_47 : out std_logic_vector(8 downto 0);
xout_48 : out std_logic_vector(8 downto 0);
xout_49 : out std_logic_vector(8 downto 0);
xout_50 : out std_logic_vector(8 downto 0);
xout_51 : out std_logic_vector(8 downto 0);
xout_52 : out std_logic_vector(8 downto 0);
xout_53 : out std_logic_vector(8 downto 0);
xout_54 : out std_logic_vector(8 downto 0);
xout_55 : out std_logic_vector(8 downto 0);
xout_56 : out std_logic_vector(8 downto 0);
xout_57 : out std_logic_vector(8 downto 0);
xout_58 : out std_logic_vector(8 downto 0);
xout_59 : out std_logic_vector(8 downto 0);
xout_60 : out std_logic_vector(8 downto 0);
xout_61 : out std_logic_vector(8 downto 0);
xout_62 : out std_logic_vector(8 downto 0);
xout_63 : out std_logic_vector(8 downto 0);
xout_64 : out std_logic_vector(8 downto 0);
xout_65 : out std_logic_vector(8 downto 0);
xout_66 : out std_logic_vector(8 downto 0);
xout_67 : out std_logic_vector(8 downto 0);
xout_68 : out std_logic_vector(8 downto 0);
xout_69 : out std_logic_vector(8 downto 0);
xout_70 : out std_logic_vector(8 downto 0);
xout_71 : out std_logic_vector(8 downto 0);
xout_72 : out std_logic_vector(8 downto 0);
xout_73 : out std_logic_vector(8 downto 0);
xout_74 : out std_logic_vector(8 downto 0);
xout_75 : out std_logic_vector(8 downto 0);
xout_76 : out std_logic_vector(8 downto 0);
```

```
xout_77 : out std_logic_vector(8 downto 0);
xout_78 : out std_logic_vector(8 downto 0);
xout_79 : out std_logic_vector(8 downto 0);
xout_80 : out std_logic_vector(8 downto 0);
xout_81 : out std_logic_vector(8 downto 0);
xout_82 : out std_logic_vector(8 downto 0);
xout_83 : out std_logic_vector(8 downto 0);
xout_84 : out std_logic_vector(8 downto 0);
xout_85 : out std_logic_vector(8 downto 0);
xout_86 : out std_logic_vector(8 downto 0);
xout_87 : out std_logic_vector(8 downto 0);
xout_88 : out std_logic_vector(8 downto 0);
xout_89 : out std_logic_vector(8 downto 0);
xout_90 : out std_logic_vector(8 downto 0);
xout_91 : out std_logic_vector(8 downto 0);
xout_92 : out std_logic_vector(8 downto 0);
xout_93 : out std_logic_vector(8 downto 0);
xout_94 : out std_logic_vector(8 downto 0);
xout_95 : out std_logic_vector(8 downto 0);
xout_96 : out std_logic_vector(8 downto 0);
xout_97 : out std_logic_vector(8 downto 0);
xout_98 : out std_logic_vector(8 downto 0);
xout_99 : out std_logic_vector(8 downto 0);
xout_100 : out std_logic_vector(8 downto 0);
xout_101 : out std_logic_vector(8 downto 0);
xout_102 : out std_logic_vector(8 downto 0);
xout_103 : out std_logic_vector(8 downto 0);
xout_104 : out std_logic_vector(8 downto 0);
xout_105 : out std_logic_vector(8 downto 0);
xout_106 : out std_logic_vector(8 downto 0);
xout_107 : out std_logic_vector(8 downto 0);
xout_108 : out std_logic_vector(8 downto 0);
xout_109 : out std_logic_vector(8 downto 0);
xout_110 : out std_logic_vector(8 downto 0);
xout_111 : out std_logic_vector(8 downto 0);
xout_112 : out std_logic_vector(8 downto 0);
xout_113 : out std_logic_vector(8 downto 0);
xout_114 : out std_logic_vector(8 downto 0);
xout_115 : out std_logic_vector(8 downto 0);
xout_116 : out std_logic_vector(8 downto 0);
xout_117 : out std_logic_vector(8 downto 0);
xout_118 : out std_logic_vector(8 downto 0);
xout_119 : out std_logic_vector(8 downto 0);
xout_120 : out std_logic_vector(8 downto 0);
xout_121 : out std_logic_vector(8 downto 0);
xout_122 : out std_logic_vector(8 downto 0);
xout_123 : out std_logic_vector(8 downto 0);
xout_124 : out std_logic_vector(8 downto 0);
xout_125 : out std_logic_vector(8 downto 0);
xout_126 : out std_logic_vector(8 downto 0);
xout_127 : out std_logic_vector(8 downto 0);
yl_0 : in std_logic_vector(8 downto 0);
yl_1 : in std_logic_vector(8 downto 0);
yl_2 : in std_logic_vector(8 downto 0);
yl_3 : in std_logic_vector(8 downto 0);
yl_4 : in std_logic_vector(8 downto 0);
yl_5 : in std_logic_vector(8 downto 0);
yl_6 : in std_logic_vector(8 downto 0);
yl_7 : in std_logic_vector(8 downto 0);
yl_8 : in std_logic_vector(8 downto 0);
yl_9 : in std_logic_vector(8 downto 0);
yl_10 : in std_logic_vector(8 downto 0);
yl_11 : in std_logic_vector(8 downto 0);
yl_12 : in std_logic_vector(8 downto 0);
```

```
yl_13 : in std_logic_vector(8 downto 0);
yl_14 : in std_logic_vector(8 downto 0);
yl_15 : in std_logic_vector(8 downto 0);
yl_16 : in std_logic_vector(8 downto 0);
yl_17 : in std_logic_vector(8 downto 0);
yl_18 : in std_logic_vector(8 downto 0);
yl_19 : in std_logic_vector(8 downto 0);
yl_20 : in std_logic_vector(8 downto 0);
yl_21 : in std_logic_vector(8 downto 0);
yl_22 : in std_logic_vector(8 downto 0);
yl_23 : in std_logic_vector(8 downto 0);
yl_24 : in std_logic_vector(8 downto 0);
yl_25 : in std_logic_vector(8 downto 0);
yl_26 : in std_logic_vector(8 downto 0);
yl_27 : in std_logic_vector(8 downto 0);
yl_28 : in std_logic_vector(8 downto 0);
yl_29 : in std_logic_vector(8 downto 0);
yl_30 : in std_logic_vector(8 downto 0);
yl_31 : in std_logic_vector(8 downto 0);
yl_32 : in std_logic_vector(8 downto 0);
yl_33 : in std_logic_vector(8 downto 0);
yl_34 : in std_logic_vector(8 downto 0);
yl_35 : in std_logic_vector(8 downto 0);
yl_36 : in std_logic_vector(8 downto 0);
yl_37 : in std_logic_vector(8 downto 0);
yl_38 : in std_logic_vector(8 downto 0);
yl_39 : in std_logic_vector(8 downto 0);
yl_40 : in std_logic_vector(8 downto 0);
yl_41 : in std_logic_vector(8 downto 0);
yl_42 : in std_logic_vector(8 downto 0);
yl_43 : in std_logic_vector(8 downto 0);
yl_44 : in std_logic_vector(8 downto 0);
yl_45 : in std_logic_vector(8 downto 0);
yl_46 : in std_logic_vector(8 downto 0);
yl_47 : in std_logic_vector(8 downto 0);
yl_48 : in std_logic_vector(8 downto 0);
yl_49 : in std_logic_vector(8 downto 0);
yl_50 : in std_logic_vector(8 downto 0);
yl_51 : in std_logic_vector(8 downto 0);
yl_52 : in std_logic_vector(8 downto 0);
yl_53 : in std_logic_vector(8 downto 0);
yl_54 : in std_logic_vector(8 downto 0);
yl_55 : in std_logic_vector(8 downto 0);
yl_56 : in std_logic_vector(8 downto 0);
yl_57 : in std_logic_vector(8 downto 0);
yl_58 : in std_logic_vector(8 downto 0);
yl_59 : in std_logic_vector(8 downto 0);
yl_60 : in std_logic_vector(8 downto 0);
yl_61 : in std_logic_vector(8 downto 0);
yl_62 : in std_logic_vector(8 downto 0);
yl_63 : in std_logic_vector(8 downto 0);
yh_0 : in std_logic_vector(8 downto 0);
yh_1 : in std_logic_vector(8 downto 0);
yh_2 : in std_logic_vector(8 downto 0);
yh_3 : in std_logic_vector(8 downto 0);
yh_4 : in std_logic_vector(8 downto 0);
yh_5 : in std_logic_vector(8 downto 0);
yh_6 : in std_logic_vector(8 downto 0);
yh_7 : in std_logic_vector(8 downto 0);
yh_8 : in std_logic_vector(8 downto 0);
yh_9 : in std_logic_vector(8 downto 0);
yh_10 : in std_logic_vector(8 downto 0);
yh_11 : in std_logic_vector(8 downto 0);
yh_12 : in std_logic_vector(8 downto 0);
```

```
		yh_13 : in std_logic_vector(8 downto 0);
		yh_14 : in std_logic_vector(8 downto 0);
		yh_15 : in std_logic_vector(8 downto 0);
		yh_16 : in std_logic_vector(8 downto 0);
		yh_17 : in std_logic_vector(8 downto 0);
		yh_18 : in std_logic_vector(8 downto 0);
		yh_19 : in std_logic_vector(8 downto 0);
		yh_20 : in std_logic_vector(8 downto 0);
		yh_21 : in std_logic_vector(8 downto 0);
		yh_22 : in std_logic_vector(8 downto 0);
		yh_23 : in std_logic_vector(8 downto 0);
		yh_24 : in std_logic_vector(8 downto 0);
		yh_25 : in std_logic_vector(8 downto 0);
		yh_26 : in std_logic_vector(8 downto 0);
		yh_27 : in std_logic_vector(8 downto 0);
		yh_28 : in std_logic_vector(8 downto 0);
		yh_29 : in std_logic_vector(8 downto 0);
		yh_30 : in std_logic_vector(8 downto 0);
		yh_31 : in std_logic_vector(8 downto 0);
		yh_32 : in std_logic_vector(8 downto 0);
		yh_33 : in std_logic_vector(8 downto 0);
		yh_34 : in std_logic_vector(8 downto 0);
		yh_35 : in std_logic_vector(8 downto 0);
		yh_36 : in std_logic_vector(8 downto 0);
		yh_37 : in std_logic_vector(8 downto 0);
		yh_38 : in std_logic_vector(8 downto 0);
		yh_39 : in std_logic_vector(8 downto 0);
		yh_40 : in std_logic_vector(8 downto 0);
		yh_41 : in std_logic_vector(8 downto 0);
		yh_42 : in std_logic_vector(8 downto 0);
		yh_43 : in std_logic_vector(8 downto 0);
		yh_44 : in std_logic_vector(8 downto 0);
		yh_45 : in std_logic_vector(8 downto 0);
		yh_46 : in std_logic_vector(8 downto 0);
		yh_47 : in std_logic_vector(8 downto 0);
		yh_48 : in std_logic_vector(8 downto 0);
		yh_49 : in std_logic_vector(8 downto 0);
		yh_50 : in std_logic_vector(8 downto 0);
		yh_51 : in std_logic_vector(8 downto 0);
		yh_52 : in std_logic_vector(8 downto 0);
		yh_53 : in std_logic_vector(8 downto 0);
		yh_54 : in std_logic_vector(8 downto 0);
		yh_55 : in std_logic_vector(8 downto 0);
		yh_56 : in std_logic_vector(8 downto 0);
		yh_57 : in std_logic_vector(8 downto 0);
		yh_58 : in std_logic_vector(8 downto 0);
		yh_59 : in std_logic_vector(8 downto 0);
		yh_60 : in std_logic_vector(8 downto 0);
		yh_61 : in std_logic_vector(8 downto 0);
		yh_62 : in std_logic_vector(8 downto 0);
		yh_63 : in std_logic_vector(8 downto 0)
		);
end entity;

architecture behavioral of idwt is

	component lut_h1_idwt
		port(
			address : in std_logic_vector(8 downto 0);
			result : out std_logic_vector(8 downto 0)
		);
	end component;

	component lut_h2_idwt
```

```
		port(
			address : in std_logic_vector(8 downto 0);
			result : out std_logic_vector(8 downto 0)
		);
	end component;

	component lut_h3_idwt
		port(
			address : in std_logic_vector(8 downto 0);
			result : out std_logic_vector(8 downto 0)
		);
	end component;

	component lut_h4_idwt
		port(
			address : in std_logic_vector(8 downto 0);
			result : out std_logic_vector(8 downto 0)
		);
	end component;

	component lut_h1_idwt_neg
		port(
			address : in std_logic_vector(8 downto 0);
			result : out std_logic_vector(8 downto 0)
			);
	end component;

	component lut_h3_idwt_neg
		port(
			address : in std_logic_vector(8 downto 0);
			result : out std_logic_vector(8 downto 0)
			);
	end component;
	signal x0_0, x0_1, x0_2, x0_3, x0_4, x0_5, x0_6, x0_7, x0_8, x0_9, x0_10,
x0_11,
			x0_12, x0_13, x0_14, x0_15, x0_16, x0_17, x0_18, x0_19, x0_20,
x0_21,
			x0_22, x0_23, x0_24, x0_25, x0_26, x0_27, x0_28, x0_29, x0_30,
x0_31,
			x0_32, x0_33, x0_34, x0_35, x0_36, x0_37, x0_38, x0_39, x0_40,
x0_41,
			x0_42, x0_43, x0_44, x0_45, x0_46, x0_47, x0_48, x0_49, x0_50,
x0_51,
			x0_52, x0_53, x0_54, x0_55, x0_56, x0_57, x0_58, x0_59, x0_60,
x0_61,
			x0_62, x0_63, x0_64, x0_65, x0_66, x0_67, x0_68, x0_69, x0_70,
x0_71,
			x0_72, x0_73, x0_74, x0_75, x0_76, x0_77, x0_78, x0_79, x0_80,
x0_81,
			x0_82, x0_83, x0_84, x0_85, x0_86, x0_87, x0_88, x0_89, x0_90,
x0_91,
			x0_92, x0_93, x0_94, x0_95, x0_96, x0_97, x0_98, x0_99, x0_100,
x0_101,
			x0_102, x0_103, x0_104, x0_105, x0_106, x0_107, x0_108, x0_109,
x0_110,
			x0_111, x0_112, x0_113, x0_114, x0_115, x0_116, x0_117, x0_118,
x0_119,
			x0_120, x0_121, x0_122, x0_123, x0_124, x0_125, x0_126, x0_127,
x0_128,
			x0_129, x0_130, x0_131, x0_132, x0_133, x0_134, x0_135, x0_136,
x0_137,
			x0_138, x0_139, x0_140, x0_141, x0_142, x0_143, x0_144, x0_145,
x0_146,
			x0_147, x0_148, x0_149, x0_150, x0_151, x0_152, x0_153, x0_154,
```

x0_155,
x0_156, x0_157, x0_158, x0_159, x0_160, x0_161, x0_162, x0_163,
x0_164,
x0_165, x0_166, x0_167, x0_168, x0_169, x0_170, x0_171, x0_172,
x0_173,
x0_174, x0_175, x0_176, x0_177, x0_178, x0_179, x0_180, x0_181,
x0_182,
x0_183, x0_184, x0_185, x0_186, x0_187, x0_188, x0_189, x0_190,
x0_191,
x0_192, x0_193, x0_194, x0_195, x0_196, x0_197, x0_198, x0_199,
x0_200,
x0_201, x0_202, x0_203, x0_204, x0_205, x0_206, x0_207, x0_208,
x0_209,
x0_210, x0_211, x0_212, x0_213, x0_214, x0_215, x0_216, x0_217,
x0_218,
x0_219, x0_220, x0_221, x0_222, x0_223, x0_224, x0_225, x0_226,
x0_227,
x0_228, x0_229, x0_230, x0_231, x0_232, x0_233, x0_234, x0_235,
x0_236,
x0_237, x0_238, x0_239, x0_240, x0_241, x0_242, x0_243, x0_244,
x0_245,
x0_246, x0_247, x0_248, x0_249, x0_250, x0_251, x0_252, x0_253,
x0_254,
x0_255, x1_0, x1_1, x1_2, x1_3, x1_4, x1_5, x1_6, x1_7, x1_8,
x1_9, x1_10,
x1_11, x1_12, x1_13, x1_14, x1_15, x1_16, x1_17, x1_18, x1_19,
x1_20, x1_21,
x1_22, x1_23, x1_24, x1_25, x1_26, x1_27, x1_28, x1_29, x1_30,
x1_31, x1_32,
x1_33, x1_34, x1_35, x1_36, x1_37, x1_38, x1_39, x1_40, x1_41,
x1_42, x1_43,
x1_44, x1_45, x1_46, x1_47, x1_48, x1_49, x1_50, x1_51, x1_52,
x1_53, x1_54,
x1_55, x1_56, x1_57, x1_58, x1_59, x1_60, x1_61, x1_62, x1_63,
x1_64, x1_65,
x1_66, x1_67, x1_68, x1_69, x1_70, x1_71, x1_72, x1_73, x1_74,
x1_75, x1_76,
x1_77, x1_78, x1_79, x1_80, x1_81, x1_82, x1_83, x1_84, x1_85,
x1_86, x1_87,
x1_88, x1_89, x1_90, x1_91, x1_92, x1_93, x1_94, x1_95, x1_96,
x1_97, x1_98,
x1_99, x1_100, x1_101, x1_102, x1_103, x1_104, x1_105, x1_106,
x1_107, x1_108,
x1_109, x1_110, x1_111, x1_112, x1_113, x1_114, x1_115, x1_116,
x1_117, x1_118,
x1_119, x1_120, x1_121, x1_122, x1_123, x1_124, x1_125, x1_126,
x1_127, x1_128,
x1_129, x1_130, x1_131, x1_132, x1_133, x1_134, x1_135, x1_136,
x1_137, x1_138,
x1_139, x1_140, x1_141, x1_142, x1_143, x1_144, x1_145, x1_146,
x1_147, x1_148,
x1_149, x1_150, x1_151, x1_152, x1_153, x1_154, x1_155, x1_156,
x1_157, x1_158,
x1_159, x1_160, x1_161, x1_162, x1_163, x1_164, x1_165, x1_166,
x1_167, x1_168,
x1_169, x1_170, x1_171, x1_172, x1_173, x1_174, x1_175, x1_176,
x1_177, x1_178,
x1_179, x1_180, x1_181, x1_182, x1_183, x1_184, x1_185, x1_186,
x1_187, x1_188,
x1_189, x1_190, x1_191, x1_192, x1_193, x1_194, x1_195, x1_196,
x1_197, x1_198,
x1_199, x1_200, x1_201, x1_202, x1_203, x1_204, x1_205, x1_206,
x1_207, x1_208,
x1_209, x1_210, x1_211, x1_212, x1_213, x1_214, x1_215, x1_216,

```
x1_217, x1_218,
                        x1_219, x1_220, x1_221, x1_222, x1_223, x1_224, x1_225, x1_226,
x1_227, x1_228,
                        x1_229, x1_230, x1_231, x1_232, x1_233, x1_234, x1_235, x1_236,
x1_237, x1_238,
                        x1_239, x1_240, x1_241, x1_242, x1_243, x1_244, x1_245, x1_246,
x1_247, x1_248,
                        x1_249, x1_250, x1_251, x1_252, x1_253, x1_254, x1_255 :
std_logic_vector(8 downto 0);
                        begin
                        -- even signal
        i0_0: lut_h3_idwt
                port map(
                        address => yl_0, result => x0_0
        );
        i0_1: lut_h2_idwt
                port map(
                        address => yh_0, result => x0_1
        );
        i0_2: lut_h1_idwt
                port map(
                        address => yl_1, result => x0_2
        );
        i0_3: lut_h4_idwt
                port map(
                        address => yh_1, result => x0_3
                );
        xout_0 <= x0_0 + x0_1 + x0_2 + x0_3;
        i0_4: lut_h3_idwt
                port map(
                        address => yl_1, result => x0_4
        );
        i0_5: lut_h2_idwt
                port map(
                        address => yh_1, result => x0_5
        );
        i0_6: lut_h1_idwt
                port map(
                        address => yl_2, result => x0_6
        );
        i0_7: lut_h4_idwt
                port map(
                        address => yh_2, result => x0_7
                );
        xout_2 <= x0_4 + x0_5 + x0_6 + x0_7;
        i0_8: lut_h3_idwt
                port map(
                        address => yl_2, result => x0_8
        );
        i0_9: lut_h2_idwt
                port map(
                        address => yh_2, result => x0_9
        );
        i0_10: lut_h1_idwt
                port map(
                        address => yl_3, result => x0_10
        );
        i0_11: lut_h4_idwt
                port map(
                        address => yh_3, result => x0_11
                );
        xout_4 <= x0_8 + x0_9 + x0_10 + x0_11;
        i0_12: lut_h3_idwt
                port map(
```

```
                address => yl_3, result => x0_12
);
i0_13: lut_h2_idwt
        port map(
                address => yh_3, result => x0_13
);
i0_14: lut_h1_idwt
        port map(
                address => yl_4, result => x0_14
);
i0_15: lut_h4_idwt
        port map(
                address => yh_4, result => x0_15
        );
xout_6 <= x0_12 + x0_13 + x0_14 + x0_15;
i0_16: lut_h3_idwt
        port map(
                address => yl_4, result => x0_16
);
i0_17: lut_h2_idwt
        port map(
                address => yh_4, result => x0_17
);
i0_18: lut_h1_idwt
        port map(
                address => yl_5, result => x0_18
);
i0_19: lut_h4_idwt
        port map(
                address => yh_5, result => x0_19
        );
xout_8 <= x0_16 + x0_17 + x0_18 + x0_19;
i0_20: lut_h3_idwt
        port map(
                address => yl_5, result => x0_20
);
i0_21: lut_h2_idwt
        port map(
                address => yh_5, result => x0_21
);
i0_22: lut_h1_idwt
        port map(
                address => yl_6, result => x0_22
);
i0_23: lut_h4_idwt
        port map(
                address => yh_6, result => x0_23
        );
xout_10 <= x0_20 + x0_21 + x0_22 + x0_23;
i0_24: lut_h3_idwt
        port map(
                address => yl_6, result => x0_24
);
i0_25: lut_h2_idwt
        port map(
                address => yh_6, result => x0_25
);
i0_26: lut_h1_idwt
        port map(
                address => yl_7, result => x0_26
);
i0_27: lut_h4_idwt
        port map(
                address => yh_7, result => x0_27
```

```
	);
xout_12 <= x0_24 + x0_25 + x0_26 + x0_27;
i0_28: lut_h3_idwt
	port map(
		address => yl_7, result => x0_28
);
i0_29: lut_h2_idwt
	port map(
		address => yh_7, result => x0_29
);
i0_30: lut_h1_idwt
	port map(
		address => yl_8, result => x0_30
);
i0_31: lut_h4_idwt
	port map(
		address => yh_8, result => x0_31
	);
xout_14 <= x0_28 + x0_29 + x0_30 + x0_31;
i0_32: lut_h3_idwt
	port map(
		address => yl_8, result => x0_32
);
i0_33: lut_h2_idwt
	port map(
		address => yh_8, result => x0_33
);
i0_34: lut_h1_idwt
	port map(
		address => yl_9, result => x0_34
);
i0_35: lut_h4_idwt
	port map(
		address => yh_9, result => x0_35
	);
xout_16 <= x0_32 + x0_33 + x0_34 + x0_35;
i0_36: lut_h3_idwt
	port map(
		address => yl_9, result => x0_36
);
i0_37: lut_h2_idwt
	port map(
		address => yh_9, result => x0_37
);
i0_38: lut_h1_idwt
	port map(
		address => yl_10, result => x0_38
);
i0_39: lut_h4_idwt
	port map(
		address => yh_10, result => x0_39
	);
xout_18 <= x0_36 + x0_37 + x0_38 + x0_39;
i0_40: lut_h3_idwt
	port map(
		address => yl_10, result => x0_40
);
i0_41: lut_h2_idwt
	port map(
		address => yh_10, result => x0_41
);
i0_42: lut_h1_idwt
	port map(
		address => yl_11, result => x0_42
```

```
);
i0_43: lut_h4_idwt
       port map(
              address => yh_11, result => x0_43
       );
xout_20 <= x0_40 + x0_41 + x0_42 + x0_43;
i0_44: lut_h3_idwt
       port map(
              address => yl_11, result => x0_44
);
i0_45: lut_h2_idwt
       port map(
              address => yh_11, result => x0_45
);
i0_46: lut_h1_idwt
       port map(
              address => yl_12, result => x0_46
);
i0_47: lut_h4_idwt
       port map(
              address => yh_12, result => x0_47
       );
xout_22 <= x0_44 + x0_45 + x0_46 + x0_47;
i0_48: lut_h3_idwt
       port map(
              address => yl_12, result => x0_48
);
i0_49: lut_h2_idwt
       port map(
              address => yh_12, result => x0_49
);
i0_50: lut_h1_idwt
       port map(
              address => yl_13, result => x0_50
);
i0_51: lut_h4_idwt
       port map(
              address => yh_13, result => x0_51
       );
xout_24 <= x0_48 + x0_49 + x0_50 + x0_51;
i0_52: lut_h3_idwt
       port map(
              address => yl_13, result => x0_52
);
i0_53: lut_h2_idwt
       port map(
              address => yh_13, result => x0_53
);
i0_54: lut_h1_idwt
       port map(
              address => yl_14, result => x0_54
);
i0_55: lut_h4_idwt
       port map(
              address => yh_14, result => x0_55
       );
xout_26 <= x0_52 + x0_53 + x0_54 + x0_55;
i0_56: lut_h3_idwt
       port map(
              address => yl_14, result => x0_56
);
i0_57: lut_h2_idwt
       port map(
              address => yh_14, result => x0_57
```

```
);
i0_58: lut_h1_idwt
       port map(
              address => yl_15, result => x0_58
);
i0_59: lut_h4_idwt
       port map(
              address => yh_15, result => x0_59
       );
xout_28 <= x0_56 + x0_57 + x0_58 + x0_59;
i0_60: lut_h3_idwt
       port map(
              address => yl_15, result => x0_60
);
i0_61: lut_h2_idwt
       port map(
              address => yh_15, result => x0_61
);
i0_62: lut_h1_idwt
       port map(
              address => yl_16, result => x0_62
);
i0_63: lut_h4_idwt
       port map(
              address => yh_16, result => x0_63
       );
xout_30 <= x0_60 + x0_61 + x0_62 + x0_63;
i0_64: lut_h3_idwt
       port map(
              address => yl_16, result => x0_64
);
i0_65: lut_h2_idwt
       port map(
              address => yh_16, result => x0_65
);
i0_66: lut_h1_idwt
       port map(
              address => yl_17, result => x0_66
);
i0_67: lut_h4_idwt
       port map(
              address => yh_17, result => x0_67
       );
xout_32 <= x0_64 + x0_65 + x0_66 + x0_67;
i0_68: lut_h3_idwt
       port map(
              address => yl_17, result => x0_68
);
i0_69: lut_h2_idwt
       port map(
              address => yh_17, result => x0_69
);
i0_70: lut_h1_idwt
       port map(
              address => yl_18, result => x0_70
);
i0_71: lut_h4_idwt
       port map(
              address => yh_18, result => x0_71
       );
xout_34 <= x0_68 + x0_69 + x0_70 + x0_71;
i0_72: lut_h3_idwt
       port map(
              address => yl_18, result => x0_72
```

```
);
i0_73: lut_h2_idwt
       port map(
              address => yh_18, result => x0_73
);
i0_74: lut_h1_idwt
       port map(
              address => yl_19, result => x0_74
);
i0_75: lut_h4_idwt
       port map(
              address => yh_19, result => x0_75
       );
xout_36 <= x0_72 + x0_73 + x0_74 + x0_75;
i0_76: lut_h3_idwt
       port map(
              address => yl_19, result => x0_76
);
i0_77: lut_h2_idwt
       port map(
              address => yh_19, result => x0_77
);
i0_78: lut_h1_idwt
       port map(
              address => yl_20, result => x0_78
);
i0_79: lut_h4_idwt
       port map(
              address => yh_20, result => x0_79
       );
xout_38 <= x0_76 + x0_77 + x0_78 + x0_79;
i0_80: lut_h3_idwt
       port map(
              address => yl_20, result => x0_80
);
i0_81: lut_h2_idwt
       port map(
              address => yh_20, result => x0_81
);
i0_82: lut_h1_idwt
       port map(
              address => yl_21, result => x0_82
);
i0_83: lut_h4_idwt
       port map(
              address => yh_21, result => x0_83
       );
xout_40 <= x0_80 + x0_81 + x0_82 + x0_83;
i0_84: lut_h3_idwt
       port map(
              address => yl_21, result => x0_84
);
i0_85: lut_h2_idwt
       port map(
              address => yh_21, result => x0_85
);
i0_86: lut_h1_idwt
       port map(
              address => yl_22, result => x0_86
);
i0_87: lut_h4_idwt
       port map(
              address => yh_22, result => x0_87
       );
```

```
xout_42 <= x0_84 + x0_85 + x0_86 + x0_87;
i0_88: lut_h3_idwt
       port map(
              address => yl_22, result => x0_88
);
i0_89: lut_h2_idwt
       port map(
              address => yh_22, result => x0_89
);
i0_90: lut_h1_idwt
       port map(
              address => yl_23, result => x0_90
);
i0_91: lut_h4_idwt
       port map(
              address => yh_23, result => x0_91
       );
xout_44 <= x0_88 + x0_89 + x0_90 + x0_91;
i0_92: lut_h3_idwt
       port map(
              address => yl_23, result => x0_92
);
i0_93: lut_h2_idwt
       port map(
              address => yh_23, result => x0_93
);
i0_94: lut_h1_idwt
       port map(
              address => yl_24, result => x0_94
);
i0_95: lut_h4_idwt
       port map(
              address => yh_24, result => x0_95
       );
xout_46 <= x0_92 + x0_93 + x0_94 + x0_95;
i0_96: lut_h3_idwt
       port map(
              address => yl_24, result => x0_96
);
i0_97: lut_h2_idwt
       port map(
              address => yh_24, result => x0_97
);
i0_98: lut_h1_idwt
       port map(
              address => yl_25, result => x0_98
);
i0_99: lut_h4_idwt
       port map(
              address => yh_25, result => x0_99
       );
xout_48 <= x0_96 + x0_97 + x0_98 + x0_99;
i0_100: lut_h3_idwt
       port map(
              address => yl_25, result => x0_100
);
i0_101: lut_h2_idwt
       port map(
              address => yh_25, result => x0_101
);
i0_102: lut_h1_idwt
       port map(
              address => yl_26, result => x0_102
);
```

```
i0_103: lut_h4_idwt
        port map(
                address => yh_26, result => x0_103
        );
xout_50 <= x0_100 + x0_101 + x0_102 + x0_103;
i0_104: lut_h3_idwt
        port map(
                address => yl_26, result => x0_104
);
i0_105: lut_h2_idwt
        port map(
                address => yh_26, result => x0_105
);
i0_106: lut_h1_idwt
        port map(
                address => yl_27, result => x0_106
);
i0_107: lut_h4_idwt
        port map(
                address => yh_27, result => x0_107
        );
xout_52 <= x0_104 + x0_105 + x0_106 + x0_107;
i0_108: lut_h3_idwt
        port map(
                address => yl_27, result => x0_108
);
i0_109: lut_h2_idwt
        port map(
                address => yh_27, result => x0_109
);
i0_110: lut_h1_idwt
        port map(
                address => yl_28, result => x0_110
);
i0_111: lut_h4_idwt
        port map(
                address => yh_28, result => x0_111
        );
xout_54 <= x0_108 + x0_109 + x0_110 + x0_111;
i0_112: lut_h3_idwt
        port map(
                address => yl_28, result => x0_112
);
i0_113: lut_h2_idwt
        port map(
                address => yh_28, result => x0_113
);
i0_114: lut_h1_idwt
        port map(
                address => yl_29, result => x0_114
);
i0_115: lut_h4_idwt
        port map(
                address => yh_29, result => x0_115
        );
xout_56 <= x0_112 + x0_113 + x0_114 + x0_115;
i0_116: lut_h3_idwt
        port map(
                address => yl_29, result => x0_116
);
i0_117: lut_h2_idwt
        port map(
                address => yh_29, result => x0_117
);
```

```
i0_118: lut_h1_idwt
	port map(
		address => yl_30, result => x0_118
);
i0_119: lut_h4_idwt
	port map(
		address => yh_30, result => x0_119
	);
xout_58 <= x0_116 + x0_117 + x0_118 + x0_119;
i0_120: lut_h3_idwt
	port map(
		address => yl_30, result => x0_120
);
i0_121: lut_h2_idwt
	port map(
		address => yh_30, result => x0_121
);
i0_122: lut_h1_idwt
	port map(
		address => yl_31, result => x0_122
);
i0_123: lut_h4_idwt
	port map(
		address => yh_31, result => x0_123
	);
xout_60 <= x0_120 + x0_121 + x0_122 + x0_123;
i0_124: lut_h3_idwt
	port map(
		address => yl_31, result => x0_124
);
i0_125: lut_h2_idwt
	port map(
		address => yh_31, result => x0_125
);
i0_126: lut_h1_idwt
	port map(
		address => yl_32, result => x0_126
);
i0_127: lut_h4_idwt
	port map(
		address => yh_32, result => x0_127
	);
xout_62 <= x0_124 + x0_125 + x0_126 + x0_127;
i0_128: lut_h3_idwt
	port map(
		address => yl_32, result => x0_128
);
i0_129: lut_h2_idwt
	port map(
		address => yh_32, result => x0_129
);
i0_130: lut_h1_idwt
	port map(
		address => yl_33, result => x0_130
);
i0_131: lut_h4_idwt
	port map(
		address => yh_33, result => x0_131
	);
xout_64 <= x0_128 + x0_129 + x0_130 + x0_131;
i0_132: lut_h3_idwt
	port map(
		address => yl_33, result => x0_132
);
```

```
i0_133: lut_h2_idwt
	port map(
		address => yh_33, result => x0_133
);
i0_134: lut_h1_idwt
	port map(
		address => yl_34, result => x0_134
);
i0_135: lut_h4_idwt
	port map(
		address => yh_34, result => x0_135
	);
xout_66 <= x0_132 + x0_133 + x0_134 + x0_135;
i0_136: lut_h3_idwt
	port map(
		address => yl_34, result => x0_136
);
i0_137: lut_h2_idwt
	port map(
		address => yh_34, result => x0_137
);
i0_138: lut_h1_idwt
	port map(
		address => yl_35, result => x0_138
);
i0_139: lut_h4_idwt
	port map(
		address => yh_35, result => x0_139
	);
xout_68 <= x0_136 + x0_137 + x0_138 + x0_139;
i0_140: lut_h3_idwt
	port map(
		address => yl_35, result => x0_140
);
i0_141: lut_h2_idwt
	port map(
		address => yh_35, result => x0_141
);
i0_142: lut_h1_idwt
	port map(
		address => yl_36, result => x0_142
);
i0_143: lut_h4_idwt
	port map(
		address => yh_36, result => x0_143
	);
xout_70 <= x0_140 + x0_141 + x0_142 + x0_143;
i0_144: lut_h3_idwt
	port map(
		address => yl_36, result => x0_144
);
i0_145: lut_h2_idwt
	port map(
		address => yh_36, result => x0_145
);
i0_146: lut_h1_idwt
	port map(
		address => yl_37, result => x0_146
);
i0_147: lut_h4_idwt
	port map(
		address => yh_37, result => x0_147
	);
xout_72 <= x0_144 + x0_145 + x0_146 + x0_147;
```

```
i0_148: lut_h3_idwt
	port map(
		address => yl_37, result => x0_148
);
i0_149: lut_h2_idwt
	port map(
		address => yh_37, result => x0_149
);
i0_150: lut_h1_idwt
	port map(
		address => yl_38, result => x0_150
);
i0_151: lut_h4_idwt
	port map(
		address => yh_38, result => x0_151
	);
xout_74 <= x0_148 + x0_149 + x0_150 + x0_151;
i0_152: lut_h3_idwt
	port map(
		address => yl_38, result => x0_152
);
i0_153: lut_h2_idwt
	port map(
		address => yh_38, result => x0_153
);
i0_154: lut_h1_idwt
	port map(
		address => yl_39, result => x0_154
);
i0_155: lut_h4_idwt
	port map(
		address => yh_39, result => x0_155
	);
xout_76 <= x0_152 + x0_153 + x0_154 + x0_155;
i0_156: lut_h3_idwt
	port map(
		address => yl_39, result => x0_156
);
i0_157: lut_h2_idwt
	port map(
		address => yh_39, result => x0_157
);
i0_158: lut_h1_idwt
	port map(
		address => yl_40, result => x0_158
);
i0_159: lut_h4_idwt
	port map(
		address => yh_40, result => x0_159
	);
xout_78 <= x0_156 + x0_157 + x0_158 + x0_159;
i0_160: lut_h3_idwt
	port map(
		address => yl_40, result => x0_160
);
i0_161: lut_h2_idwt
	port map(
		address => yh_40, result => x0_161
);
i0_162: lut_h1_idwt
	port map(
		address => yl_41, result => x0_162
);
i0_163: lut_h4_idwt
```

```
        port map(
                address => yh_41, result => x0_163
        );
xout_80 <= x0_160 + x0_161 + x0_162 + x0_163;
i0_164: lut_h3_idwt
        port map(
                address => yl_41, result => x0_164
);
i0_165: lut_h2_idwt
        port map(
                address => yh_41, result => x0_165
);
i0_166: lut_h1_idwt
        port map(
                address => yl_42, result => x0_166
);
i0_167: lut_h4_idwt
        port map(
                address => yh_42, result => x0_167
        );
xout_82 <= x0_164 + x0_165 + x0_166 + x0_167;
i0_168: lut_h3_idwt
        port map(
                address => yl_42, result => x0_168
);
i0_169: lut_h2_idwt
        port map(
                address => yh_42, result => x0_169
);
i0_170: lut_h1_idwt
        port map(
                address => yl_43, result => x0_170
);
i0_171: lut_h4_idwt
        port map(
                address => yh_43, result => x0_171
        );
xout_84 <= x0_168 + x0_169 + x0_170 + x0_171;
i0_172: lut_h3_idwt
        port map(
                address => yl_43, result => x0_172
);
i0_173: lut_h2_idwt
        port map(
                address => yh_43, result => x0_173
);
i0_174: lut_h1_idwt
        port map(
                address => yl_44, result => x0_174
);
i0_175: lut_h4_idwt
        port map(
                address => yh_44, result => x0_175
        );
xout_86 <= x0_172 + x0_173 + x0_174 + x0_175;
i0_176: lut_h3_idwt
        port map(
                address => yl_44, result => x0_176
);
i0_177: lut_h2_idwt
        port map(
                address => yh_44, result => x0_177
);
i0_178: lut_h1_idwt
```

```
        port map(
                address => yl_45, result => x0_178
);
i0_179: lut_h4_idwt
        port map(
                address => yh_45, result => x0_179
        );
xout_88 <= x0_176 + x0_177 + x0_178 + x0_179;
i0_180: lut_h3_idwt
        port map(
                address => yl_45, result => x0_180
);
i0_181: lut_h2_idwt
        port map(
                address => yh_45, result => x0_181
);
i0_182: lut_h1_idwt
        port map(
                address => yl_46, result => x0_182
);
i0_183: lut_h4_idwt
        port map(
                address => yh_46, result => x0_183
        );
xout_90 <= x0_180 + x0_181 + x0_182 + x0_183;
i0_184: lut_h3_idwt
        port map(
                address => yl_46, result => x0_184
);
i0_185: lut_h2_idwt
        port map(
                address => yh_46, result => x0_185
);
i0_186: lut_h1_idwt
        port map(
                address => yl_47, result => x0_186
);
i0_187: lut_h4_idwt
        port map(
                address => yh_47, result => x0_187
        );
xout_92 <= x0_184 + x0_185 + x0_186 + x0_187;
i0_188: lut_h3_idwt
        port map(
                address => yl_47, result => x0_188
);
i0_189: lut_h2_idwt
        port map(
                address => yh_47, result => x0_189
);
i0_190: lut_h1_idwt
        port map(
                address => yl_48, result => x0_190
);
i0_191: lut_h4_idwt
        port map(
                address => yh_48, result => x0_191
        );
xout_94 <= x0_188 + x0_189 + x0_190 + x0_191;
i0_192: lut_h3_idwt
        port map(
                address => yl_48, result => x0_192
);
i0_193: lut_h2_idwt
```

```
        port map(
                address => yh_48, result => x0_193
);
i0_194: lut_h1_idwt
        port map(
                address => yl_49, result => x0_194
);
i0_195: lut_h4_idwt
        port map(
                address => yh_49, result => x0_195
        );
xout_96 <= x0_192 + x0_193 + x0_194 + x0_195;
i0_196: lut_h3_idwt
        port map(
                address => yl_49, result => x0_196
);
i0_197: lut_h2_idwt
        port map(
                address => yh_49, result => x0_197
);
i0_198: lut_h1_idwt
        port map(
                address => yl_50, result => x0_198
);
i0_199: lut_h4_idwt
        port map(
                address => yh_50, result => x0_199
        );
xout_98 <= x0_196 + x0_197 + x0_198 + x0_199;
i0_200: lut_h3_idwt
        port map(
                address => yl_50, result => x0_200
);
i0_201: lut_h2_idwt
        port map(
                address => yh_50, result => x0_201
);
i0_202: lut_h1_idwt
        port map(
                address => yl_51, result => x0_202
);
i0_203: lut_h4_idwt
        port map(
                address => yh_51, result => x0_203
        );
xout_100 <= x0_200 + x0_201 + x0_202 + x0_203;
i0_204: lut_h3_idwt
        port map(
                address => yl_51, result => x0_204
);
i0_205: lut_h2_idwt
        port map(
                address => yh_51, result => x0_205
);
i0_206: lut_h1_idwt
        port map(
                address => yl_52, result => x0_206
);
i0_207: lut_h4_idwt
        port map(
                address => yh_52, result => x0_207
        );
xout_102 <= x0_204 + x0_205 + x0_206 + x0_207;
i0_208: lut_h3_idwt
```

```
        port map(
                address => yl_52, result => x0_208
);
i0_209: lut_h2_idwt
        port map(
                address => yh_52, result => x0_209
);
i0_210: lut_h1_idwt
        port map(
                address => yl_53, result => x0_210
);
i0_211: lut_h4_idwt
        port map(
                address => yh_53, result => x0_211
        );
xout_104 <= x0_208 + x0_209 + x0_210 + x0_211;
i0_212: lut_h3_idwt
        port map(
                address => yl_53, result => x0_212
);
i0_213: lut_h2_idwt
        port map(
                address => yh_53, result => x0_213
);
i0_214: lut_h1_idwt
        port map(
                address => yl_54, result => x0_214
);
i0_215: lut_h4_idwt
        port map(
                address => yh_54, result => x0_215
        );
xout_106 <= x0_212 + x0_213 + x0_214 + x0_215;
i0_216: lut_h3_idwt
        port map(
                address => yl_54, result => x0_216
);
i0_217: lut_h2_idwt
        port map(
                address => yh_54, result => x0_217
);
i0_218: lut_h1_idwt
        port map(
                address => yl_55, result => x0_218
);
i0_219: lut_h4_idwt
        port map(
                address => yh_55, result => x0_219
        );
xout_108 <= x0_216 + x0_217 + x0_218 + x0_219;
i0_220: lut_h3_idwt
        port map(
                address => yl_55, result => x0_220
);
i0_221: lut_h2_idwt
        port map(
                address => yh_55, result => x0_221
);
i0_222: lut_h1_idwt
        port map(
                address => yl_56, result => x0_222
);
i0_223: lut_h4_idwt
        port map(
```

```
                address => yh_56, result => x0_223
        );
xout_110 <= x0_220 + x0_221 + x0_222 + x0_223;
i0_224: lut_h3_idwt
        port map(
                address => yl_56, result => x0_224
);
i0_225: lut_h2_idwt
        port map(
                address => yh_56, result => x0_225
);
i0_226: lut_h1_idwt
        port map(
                address => yl_57, result => x0_226
);
i0_227: lut_h4_idwt
        port map(
                address => yh_57, result => x0_227
        );
xout_112 <= x0_224 + x0_225 + x0_226 + x0_227;
i0_228: lut_h3_idwt
        port map(
                address => yl_57, result => x0_228
);
i0_229: lut_h2_idwt
        port map(
                address => yh_57, result => x0_229
);
i0_230: lut_h1_idwt
        port map(
                address => yl_58, result => x0_230
);
i0_231: lut_h4_idwt
        port map(
                address => yh_58, result => x0_231
        );
xout_114 <= x0_228 + x0_229 + x0_230 + x0_231;
i0_232: lut_h3_idwt
        port map(
                address => yl_58, result => x0_232
);
i0_233: lut_h2_idwt
        port map(
                address => yh_58, result => x0_233
);
i0_234: lut_h1_idwt
        port map(
                address => yl_59, result => x0_234
);
i0_235: lut_h4_idwt
        port map(
                address => yh_59, result => x0_235
        );
xout_116 <= x0_232 + x0_233 + x0_234 + x0_235;
i0_236: lut_h3_idwt
        port map(
                address => yl_59, result => x0_236
);
i0_237: lut_h2_idwt
        port map(
                address => yh_59, result => x0_237
);
i0_238: lut_h1_idwt
        port map(
```

```
                address => yl_60, result => x0_238
);
i0_239: lut_h4_idwt
        port map(
                address => yh_60, result => x0_239
        );
xout_118 <= x0_236 + x0_237 + x0_238 + x0_239;
i0_240: lut_h3_idwt
        port map(
                address => yl_60, result => x0_240
);
i0_241: lut_h2_idwt
        port map(
                address => yh_60, result => x0_241
);
i0_242: lut_h1_idwt
        port map(
                address => yl_61, result => x0_242
);
i0_243: lut_h4_idwt
        port map(
                address => yh_61, result => x0_243
        );
xout_120 <= x0_240 + x0_241 + x0_242 + x0_243;
i0_244: lut_h3_idwt
        port map(
                address => yl_61, result => x0_244
);
i0_245: lut_h2_idwt
        port map(
                address => yh_61, result => x0_245
);
i0_246: lut_h1_idwt
        port map(
                address => yl_62, result => x0_246
);
i0_247: lut_h4_idwt
        port map(
                address => yh_62, result => x0_247
        );
xout_122 <= x0_244 + x0_245 + x0_246 + x0_247;
i0_248: lut_h3_idwt
        port map(
                address => yl_62, result => x0_248
);
i0_249: lut_h2_idwt
        port map(
                address => yh_62, result => x0_249
);
i0_250: lut_h1_idwt
        port map(
                address => yl_63, result => x0_250
);
i0_251: lut_h4_idwt
        port map(
                address => yh_63, result => x0_251
        );
xout_124 <= x0_248 + x0_249 + x0_250 + x0_251;
i0_252: lut_h3_idwt
        port map(
                address => yl_63, result => x0_252
);
i0_253: lut_h2_idwt
        port map(
```

```
                address => yh_63, result => x0_253
        );
        i0_254: lut_h1_idwt
                port map(
                address => yl_0, result => x0_254
        );
        i0_255: lut_h4_idwt
                port map(
                address => yh_0, result => x0_255
                );
        xout_126 <= x0_252 + x0_253 + x0_254 + x0_255;

--odd signal
        i1_0: lut_h2_idwt
                port map(
                address => yl_1, result => x1_0
        );
        i1_1: lut_h4_idwt
                port map(
                address => yl_0, result => x1_1
        );
        i1_2: lut_h1_idwt_neg
                port map(
                address => yh_0, result => x1_2
        );
        i1_3: lut_h3_idwt_neg
                port map(
                address => yh_1, result => x1_3
                );
        xout_1 <= x1_0 + x1_1 + x1_2 + x1_3;
        i1_4: lut_h2_idwt
                port map(
                address => yl_2, result => x1_4
        );
        i1_5: lut_h4_idwt
                port map(
                address => yl_1, result => x1_5
        );
        i1_6: lut_h1_idwt_neg
                port map(
                address => yh_1, result => x1_6
        );
        i1_7: lut_h3_idwt_neg
                port map(
                address => yh_2, result => x1_7
                );
        xout_3 <= x1_4 + x1_5 + x1_6 + x1_7;
        i1_8: lut_h2_idwt
                port map(
                address => yl_3, result => x1_8
        );
        i1_9: lut_h4_idwt
                port map(
                address => yl_2, result => x1_9
        );
        i1_10: lut_h1_idwt_neg
                port map(
                address => yh_2, result => x1_10
        );
        i1_11: lut_h3_idwt_neg
                port map(
                address => yh_3, result => x1_11
                );
        xout_5 <= x1_8 + x1_9 + x1_10 + x1_11;
```

```
i1_12: lut_h2_idwt
       port map(
              address => yl_4, result => x1_12
);
i1_13: lut_h4_idwt
       port map(
              address => yl_3, result => x1_13
);
i1_14: lut_h1_idwt_neg
       port map(
              address => yh_3, result => x1_14
);
i1_15: lut_h3_idwt_neg
       port map(
              address => yh_4, result => x1_15
       );
xout_7 <= x1_12 + x1_13 + x1_14 + x1_15;
i1_16: lut_h2_idwt
       port map(
              address => yl_5, result => x1_16
);
i1_17: lut_h4_idwt
       port map(
              address => yl_4, result => x1_17
);
i1_18: lut_h1_idwt_neg
       port map(
              address => yh_4, result => x1_18
);
i1_19: lut_h3_idwt_neg
       port map(
              address => yh_5, result => x1_19
       );
xout_9 <= x1_16 + x1_17 + x1_18 + x1_19;
i1_20: lut_h2_idwt
       port map(
              address => yl_6, result => x1_20
);
i1_21: lut_h4_idwt
       port map(
              address => yl_5, result => x1_21
);
i1_22: lut_h1_idwt_neg
       port map(
              address => yh_5, result => x1_22
);
i1_23: lut_h3_idwt_neg
       port map(
              address => yh_6, result => x1_23
       );
xout_11 <= x1_20 + x1_21 + x1_22 + x1_23;
i1_24: lut_h2_idwt
       port map(
              address => yl_7, result => x1_24
);
i1_25: lut_h4_idwt
       port map(
              address => yl_6, result => x1_25
);
i1_26: lut_h1_idwt_neg
       port map(
              address => yh_6, result => x1_26
);
i1_27: lut_h3_idwt_neg
```

```
        port map(
                address => yh_7, result => x1_27
        );
xout_13 <= x1_24 + x1_25 + x1_26 + x1_27;
i1_28: lut_h2_idwt
        port map(
                address => yl_8, result => x1_28
);
i1_29: lut_h4_idwt
        port map(
                address => yl_7, result => x1_29
);
i1_30: lut_h1_idwt_neg
        port map(
                address => yh_7, result => x1_30
);
i1_31: lut_h3_idwt_neg
        port map(
                address => yh_8, result => x1_31
        );
xout_15 <= x1_28 + x1_29 + x1_30 + x1_31;
i1_32: lut_h2_idwt
        port map(
                address => yl_9, result => x1_32
);
i1_33: lut_h4_idwt
        port map(
                address => yl_8, result => x1_33
);
i1_34: lut_h1_idwt_neg
        port map(
                address => yh_8, result => x1_34
);
i1_35: lut_h3_idwt_neg
        port map(
                address => yh_9, result => x1_35
        );
xout_17 <= x1_32 + x1_33 + x1_34 + x1_35;
i1_36: lut_h2_idwt
        port map(
                address => yl_10, result => x1_36
);
i1_37: lut_h4_idwt
        port map(
                address => yl_9, result => x1_37
);
i1_38: lut_h1_idwt_neg
        port map(
                address => yh_9, result => x1_38
);
i1_39: lut_h3_idwt_neg
        port map(
                address => yh_10, result => x1_39
        );
xout_19 <= x1_36 + x1_37 + x1_38 + x1_39;
i1_40: lut_h2_idwt
        port map(
                address => yl_11, result => x1_40
);
i1_41: lut_h4_idwt
        port map(
                address => yl_10, result => x1_41
);
i1_42: lut_h1_idwt_neg
```

```
        port map(
                address => yh_10, result => x1_42
);
i1_43: lut_h3_idwt_neg
        port map(
                address => yh_11, result => x1_43
        );
xout_21 <= x1_40 + x1_41 + x1_42 + x1_43;
i1_44: lut_h2_idwt
        port map(
                address => yl_12, result => x1_44
);
i1_45: lut_h4_idwt
        port map(
                address => yl_11, result => x1_45
);
i1_46: lut_h1_idwt_neg
        port map(
                address => yh_11, result => x1_46
);
i1_47: lut_h3_idwt_neg
        port map(
                address => yh_12, result => x1_47
        );
xout_23 <= x1_44 + x1_45 + x1_46 + x1_47;
i1_48: lut_h2_idwt
        port map(
                address => yl_13, result => x1_48
);
i1_49: lut_h4_idwt
        port map(
                address => yl_12, result => x1_49
);
i1_50: lut_h1_idwt_neg
        port map(
                address => yh_12, result => x1_50
);
i1_51: lut_h3_idwt_neg
        port map(
                address => yh_13, result => x1_51
        );
xout_25 <= x1_48 + x1_49 + x1_50 + x1_51;
i1_52: lut_h2_idwt
        port map(
                address => yl_14, result => x1_52
);
i1_53: lut_h4_idwt
        port map(
                address => yl_13, result => x1_53
);
i1_54: lut_h1_idwt_neg
        port map(
                address => yh_13, result => x1_54
);
i1_55: lut_h3_idwt_neg
        port map(
                address => yh_14, result => x1_55
        );
xout_27 <= x1_52 + x1_53 + x1_54 + x1_55;
i1_56: lut_h2_idwt
        port map(
                address => yl_15, result => x1_56
);
i1_57: lut_h4_idwt
```

```
        port map(
                address => yl_14, result => x1_57
);
i1_58: lut_h1_idwt_neg
        port map(
                address => yh_14, result => x1_58
);
i1_59: lut_h3_idwt_neg
        port map(
                address => yh_15, result => x1_59
        );
xout_29 <= x1_56 + x1_57 + x1_58 + x1_59;
i1_60: lut_h2_idwt
        port map(
                address => yl_16, result => x1_60
);
i1_61: lut_h4_idwt
        port map(
                address => yl_15, result => x1_61
);
i1_62: lut_h1_idwt_neg
        port map(
                address => yh_15, result => x1_62
);
i1_63: lut_h3_idwt_neg
        port map(
                address => yh_16, result => x1_63
        );
xout_31 <= x1_60 + x1_61 + x1_62 + x1_63;
i1_64: lut_h2_idwt
        port map(
                address => yl_17, result => x1_64
);
i1_65: lut_h4_idwt
        port map(
                address => yl_16, result => x1_65
);
i1_66: lut_h1_idwt_neg
        port map(
                address => yh_16, result => x1_66
);
i1_67: lut_h3_idwt_neg
        port map(
                address => yh_17, result => x1_67
        );
xout_33 <= x1_64 + x1_65 + x1_66 + x1_67;
i1_68: lut_h2_idwt
        port map(
                address => yl_18, result => x1_68
);
i1_69: lut_h4_idwt
        port map(
                address => yl_17, result => x1_69
);
i1_70: lut_h1_idwt_neg
        port map(
                address => yh_17, result => x1_70
);
i1_71: lut_h3_idwt_neg
        port map(
                address => yh_18, result => x1_71
        );
xout_35 <= x1_68 + x1_69 + x1_70 + x1_71;
i1_72: lut_h2_idwt
```

```
        port map(
                address => yl_19, result => x1_72
);
i1_73: lut_h4_idwt
        port map(
                address => yl_18, result => x1_73
);
i1_74: lut_h1_idwt_neg
        port map(
                address => yh_18, result => x1_74
);
i1_75: lut_h3_idwt_neg
        port map(
                address => yh_19, result => x1_75
        );
xout_37 <= x1_72 + x1_73 + x1_74 + x1_75;
i1_76: lut_h2_idwt
        port map(
                address => yl_20, result => x1_76
);
i1_77: lut_h4_idwt
        port map(
                address => yl_19, result => x1_77
);
i1_78: lut_h1_idwt_neg
        port map(
                address => yh_19, result => x1_78
);
i1_79: lut_h3_idwt_neg
        port map(
                address => yh_20, result => x1_79
        );
xout_39 <= x1_76 + x1_77 + x1_78 + x1_79;
i1_80: lut_h2_idwt
        port map(
                address => yl_21, result => x1_80
);
i1_81: lut_h4_idwt
        port map(
                address => yl_20, result => x1_81
);
i1_82: lut_h1_idwt_neg
        port map(
                address => yh_20, result => x1_82
);
i1_83: lut_h3_idwt_neg
        port map(
                address => yh_21, result => x1_83
        );
xout_41 <= x1_80 + x1_81 + x1_82 + x1_83;
i1_84: lut_h2_idwt
        port map(
                address => yl_22, result => x1_84
);
i1_85: lut_h4_idwt
        port map(
                address => yl_21, result => x1_85
);
i1_86: lut_h1_idwt_neg
        port map(
                address => yh_21, result => x1_86
);
i1_87: lut_h3_idwt_neg
        port map(
```

```
                address => yh_22, result => x1_87
        );
xout_43 <= x1_84 + x1_85 + x1_86 + x1_87;
i1_88: lut_h2_idwt
        port map(
                address => yl_23, result => x1_88
);
i1_89: lut_h4_idwt
        port map(
                address => yl_22, result => x1_89
);
i1_90: lut_h1_idwt_neg
        port map(
                address => yh_22, result => x1_90
);
i1_91: lut_h3_idwt_neg
        port map(
                address => yh_23, result => x1_91
        );
xout_45 <= x1_88 + x1_89 + x1_90 + x1_91;
i1_92: lut_h2_idwt
        port map(
                address => yl_24, result => x1_92
);
i1_93: lut_h4_idwt
        port map(
                address => yl_23, result => x1_93
);
i1_94: lut_h1_idwt_neg
        port map(
                address => yh_23, result => x1_94
);
i1_95: lut_h3_idwt_neg
        port map(
                address => yh_24, result => x1_95
        );
xout_47 <= x1_92 + x1_93 + x1_94 + x1_95;
i1_96: lut_h2_idwt
        port map(
                address => yl_25, result => x1_96
);
i1_97: lut_h4_idwt
        port map(
                address => yl_24, result => x1_97
);
i1_98: lut_h1_idwt_neg
        port map(
                address => yh_24, result => x1_98
);
i1_99: lut_h3_idwt_neg
        port map(
                address => yh_25, result => x1_99
        );
xout_49 <= x1_96 + x1_97 + x1_98 + x1_99;
i1_100: lut_h2_idwt
        port map(
                address => yl_26, result => x1_100
);
i1_101: lut_h4_idwt
        port map(
                address => yl_25, result => x1_101
);
i1_102: lut_h1_idwt_neg
        port map(
```

```
                address => yh_25, result => x1_102
);
i1_103: lut_h3_idwt_neg
        port map(
                address => yh_26, result => x1_103
        );
xout_51 <= x1_100 + x1_101 + x1_102 + x1_103;
i1_104: lut_h2_idwt
        port map(
                address => yl_27, result => x1_104
);
i1_105: lut_h4_idwt
        port map(
                address => yl_26, result => x1_105
);
i1_106: lut_h1_idwt_neg
        port map(
                address => yh_26, result => x1_106
);
i1_107: lut_h3_idwt_neg
        port map(
                address => yh_27, result => x1_107
        );
xout_53 <= x1_104 + x1_105 + x1_106 + x1_107;
i1_108: lut_h2_idwt
        port map(
                address => yl_28, result => x1_108
);
i1_109: lut_h4_idwt
        port map(
                address => yl_27, result => x1_109
);
i1_110: lut_h1_idwt_neg
        port map(
                address => yh_27, result => x1_110
);
i1_111: lut_h3_idwt_neg
        port map(
                address => yh_28, result => x1_111
        );
xout_55 <= x1_108 + x1_109 + x1_110 + x1_111;
i1_112: lut_h2_idwt
        port map(
                address => yl_29, result => x1_112
);
i1_113: lut_h4_idwt
        port map(
                address => yl_28, result => x1_113
);
i1_114: lut_h1_idwt_neg
        port map(
                address => yh_28, result => x1_114
);
i1_115: lut_h3_idwt_neg
        port map(
                address => yh_29, result => x1_115
        );
xout_57 <= x1_112 + x1_113 + x1_114 + x1_115;
i1_116: lut_h2_idwt
        port map(
                address => yl_30, result => x1_116
);
i1_117: lut_h4_idwt
        port map(
```

```
                address => yl_29, result => x1_117
);
i1_118: lut_h1_idwt_neg
        port map(
                address => yh_29, result => x1_118
);
i1_119: lut_h3_idwt_neg
        port map(
                address => yh_30, result => x1_119
        );
xout_59 <= x1_116 + x1_117 + x1_118 + x1_119;
i1_120: lut_h2_idwt
        port map(
                address => yl_31, result => x1_120
);
i1_121: lut_h4_idwt
        port map(
                address => yl_30, result => x1_121
);
i1_122: lut_h1_idwt_neg
        port map(
                address => yh_30, result => x1_122
);
i1_123: lut_h3_idwt_neg
        port map(
                address => yh_31, result => x1_123
        );
xout_61 <= x1_120 + x1_121 + x1_122 + x1_123;
i1_124: lut_h2_idwt
        port map(
                address => yl_32, result => x1_124
);
i1_125: lut_h4_idwt
        port map(
                address => yl_31, result => x1_125
);
i1_126: lut_h1_idwt_neg
        port map(
                address => yh_31, result => x1_126
);
i1_127: lut_h3_idwt_neg
        port map(
                address => yh_32, result => x1_127
        );
xout_63 <= x1_124 + x1_125 + x1_126 + x1_127;
i1_128: lut_h2_idwt
        port map(
                address => yl_33, result => x1_128
);
i1_129: lut_h4_idwt
        port map(
                address => yl_32, result => x1_129
);
i1_130: lut_h1_idwt_neg
        port map(
                address => yh_32, result => x1_130
);
i1_131: lut_h3_idwt_neg
        port map(
                address => yh_33, result => x1_131
        );
xout_65 <= x1_128 + x1_129 + x1_130 + x1_131;
i1_132: lut_h2_idwt
        port map(
```

```
                address => yl_34, result => x1_132
);
i1_133: lut_h4_idwt
        port map(
                address => yl_33, result => x1_133
);
i1_134: lut_h1_idwt_neg
        port map(
                address => yh_33, result => x1_134
);
i1_135: lut_h3_idwt_neg
        port map(
                address => yh_34, result => x1_135
        );
xout_67 <= x1_132 + x1_133 + x1_134 + x1_135;
i1_136: lut_h2_idwt
        port map(
                address => yl_35, result => x1_136
);
i1_137: lut_h4_idwt
        port map(
                address => yl_34, result => x1_137
);
i1_138: lut_h1_idwt_neg
        port map(
                address => yh_34, result => x1_138
);
i1_139: lut_h3_idwt_neg
        port map(
                address => yh_35, result => x1_139
        );
xout_69 <= x1_136 + x1_137 + x1_138 + x1_139;
i1_140: lut_h2_idwt
        port map(
                address => yl_36, result => x1_140
);
i1_141: lut_h4_idwt
        port map(
                address => yl_35, result => x1_141
);
i1_142: lut_h1_idwt_neg
        port map(
                address => yh_35, result => x1_142
);
i1_143: lut_h3_idwt_neg
        port map(
                address => yh_36, result => x1_143
        );
xout_71 <= x1_140 + x1_141 + x1_142 + x1_143;
i1_144: lut_h2_idwt
        port map(
                address => yl_37, result => x1_144
);
i1_145: lut_h4_idwt
        port map(
                address => yl_36, result => x1_145
);
i1_146: lut_h1_idwt_neg
        port map(
                address => yh_36, result => x1_146
);
i1_147: lut_h3_idwt_neg
        port map(
                address => yh_37, result => x1_147
```

```
	);
xout_73 <= x1_144 + x1_145 + x1_146 + x1_147;
i1_148: lut_h2_idwt
	port map(
		address => yl_38, result => x1_148
);
i1_149: lut_h4_idwt
	port map(
		address => yl_37, result => x1_149
);
i1_150: lut_h1_idwt_neg
	port map(
		address => yh_37, result => x1_150
);
i1_151: lut_h3_idwt_neg
	port map(
		address => yh_38, result => x1_151
	);
xout_75 <= x1_148 + x1_149 + x1_150 + x1_151;
i1_152: lut_h2_idwt
	port map(
		address => yl_39, result => x1_152
);
i1_153: lut_h4_idwt
	port map(
		address => yl_38, result => x1_153
);
i1_154: lut_h1_idwt_neg
	port map(
		address => yh_38, result => x1_154
);
i1_155: lut_h3_idwt_neg
	port map(
		address => yh_39, result => x1_155
	);
xout_77 <= x1_152 + x1_153 + x1_154 + x1_155;
i1_156: lut_h2_idwt
	port map(
		address => yl_40, result => x1_156
);
i1_157: lut_h4_idwt
	port map(
		address => yl_39, result => x1_157
);
i1_158: lut_h1_idwt_neg
	port map(
		address => yh_39, result => x1_158
);
i1_159: lut_h3_idwt_neg
	port map(
		address => yh_40, result => x1_159
	);
xout_79 <= x1_156 + x1_157 + x1_158 + x1_159;
i1_160: lut_h2_idwt
	port map(
		address => yl_41, result => x1_160
);
i1_161: lut_h4_idwt
	port map(
		address => yl_40, result => x1_161
);
i1_162: lut_h1_idwt_neg
	port map(
		address => yh_40, result => x1_162
```

```
);
i1_163: lut_h3_idwt_neg
        port map(
                address => yh_41, result => x1_163
        );
xout_81 <= x1_160 + x1_161 + x1_162 + x1_163;
i1_164: lut_h2_idwt
        port map(
                address => yl_42, result => x1_164
);
i1_165: lut_h4_idwt
        port map(
                address => yl_41, result => x1_165
);
i1_166: lut_h1_idwt_neg
        port map(
                address => yh_41, result => x1_166
);
i1_167: lut_h3_idwt_neg
        port map(
                address => yh_42, result => x1_167
        );
xout_83 <= x1_164 + x1_165 + x1_166 + x1_167;
i1_168: lut_h2_idwt
        port map(
                address => yl_43, result => x1_168
);
i1_169: lut_h4_idwt
        port map(
                address => yl_42, result => x1_169
);
i1_170: lut_h1_idwt_neg
        port map(
                address => yh_42, result => x1_170
);
i1_171: lut_h3_idwt_neg
        port map(
                address => yh_43, result => x1_171
        );
xout_85 <= x1_168 + x1_169 + x1_170 + x1_171;
i1_172: lut_h2_idwt
        port map(
                address => yl_44, result => x1_172
);
i1_173: lut_h4_idwt
        port map(
                address => yl_43, result => x1_173
);
i1_174: lut_h1_idwt_neg
        port map(
                address => yh_43, result => x1_174
);
i1_175: lut_h3_idwt_neg
        port map(
                address => yh_44, result => x1_175
        );
xout_87 <= x1_172 + x1_173 + x1_174 + x1_175;
i1_176: lut_h2_idwt
        port map(
                address => yl_45, result => x1_176
);
i1_177: lut_h4_idwt
        port map(
                address => yl_44, result => x1_177
```

```
);
i1_178: lut_h1_idwt_neg
        port map(
                address => yh_44, result => x1_178
);
i1_179: lut_h3_idwt_neg
        port map(
                address => yh_45, result => x1_179
        );
xout_89 <= x1_176 + x1_177 + x1_178 + x1_179;
i1_180: lut_h2_idwt
        port map(
                address => yl_46, result => x1_180
);
i1_181: lut_h4_idwt
        port map(
                address => yl_45, result => x1_181
);
i1_182: lut_h1_idwt_neg
        port map(
                address => yh_45, result => x1_182
);
i1_183: lut_h3_idwt_neg
        port map(
                address => yh_46, result => x1_183
        );
xout_91 <= x1_180 + x1_181 + x1_182 + x1_183;
i1_184: lut_h2_idwt
        port map(
                address => yl_47, result => x1_184
);
i1_185: lut_h4_idwt
        port map(
                address => yl_46, result => x1_185
);
i1_186: lut_h1_idwt_neg
        port map(
                address => yh_46, result => x1_186
);
i1_187: lut_h3_idwt_neg
        port map(
                address => yh_47, result => x1_187
        );
xout_93 <= x1_184 + x1_185 + x1_186 + x1_187;
i1_188: lut_h2_idwt
        port map(
                address => yl_48, result => x1_188
);
i1_189: lut_h4_idwt
        port map(
                address => yl_47, result => x1_189
);
i1_190: lut_h1_idwt_neg
        port map(
                address => yh_47, result => x1_190
);
i1_191: lut_h3_idwt_neg
        port map(
                address => yh_48, result => x1_191
        );
xout_95 <= x1_188 + x1_189 + x1_190 + x1_191;
i1_192: lut_h2_idwt
        port map(
                address => yl_49, result => x1_192
```

```
);
i1_193: lut_h4_idwt
	port map(
		address => yl_48, result => x1_193
);
i1_194: lut_h1_idwt_neg
	port map(
		address => yh_48, result => x1_194
);
i1_195: lut_h3_idwt_neg
	port map(
		address => yh_49, result => x1_195
	);
xout_97 <= x1_192 + x1_193 + x1_194 + x1_195;
i1_196: lut_h2_idwt
	port map(
		address => yl_50, result => x1_196
);
i1_197: lut_h4_idwt
	port map(
		address => yl_49, result => x1_197
);
i1_198: lut_h1_idwt_neg
	port map(
		address => yh_49, result => x1_198
);
i1_199: lut_h3_idwt_neg
	port map(
		address => yh_50, result => x1_199
	);
xout_99 <= x1_196 + x1_197 + x1_198 + x1_199;
i1_200: lut_h2_idwt
	port map(
		address => yl_51, result => x1_200
);
i1_201: lut_h4_idwt
	port map(
		address => yl_50, result => x1_201
);
i1_202: lut_h1_idwt_neg
	port map(
		address => yh_50, result => x1_202
);
i1_203: lut_h3_idwt_neg
	port map(
		address => yh_51, result => x1_203
	);
xout_101 <= x1_200 + x1_201 + x1_202 + x1_203;
i1_204: lut_h2_idwt
	port map(
		address => yl_52, result => x1_204
);
i1_205: lut_h4_idwt
	port map(
		address => yl_51, result => x1_205
);
i1_206: lut_h1_idwt_neg
	port map(
		address => yh_51, result => x1_206
);
i1_207: lut_h3_idwt_neg
	port map(
		address => yh_52, result => x1_207
	);
```

```
xout_103 <= x1_204 + x1_205 + x1_206 + x1_207;
i1_208: lut_h2_idwt
	port map(
		address => yl_53, result => x1_208
);
i1_209: lut_h4_idwt
	port map(
		address => yl_52, result => x1_209
);
i1_210: lut_h1_idwt_neg
	port map(
		address => yh_52, result => x1_210
);
i1_211: lut_h3_idwt_neg
	port map(
		address => yh_53, result => x1_211
	);
xout_105 <= x1_208 + x1_209 + x1_210 + x1_211;
i1_212: lut_h2_idwt
	port map(
		address => yl_54, result => x1_212
);
i1_213: lut_h4_idwt
	port map(
		address => yl_53, result => x1_213
);
i1_214: lut_h1_idwt_neg
	port map(
		address => yh_53, result => x1_214
);
i1_215: lut_h3_idwt_neg
	port map(
		address => yh_54, result => x1_215
	);
xout_107 <= x1_212 + x1_213 + x1_214 + x1_215;
i1_216: lut_h2_idwt
	port map(
		address => yl_55, result => x1_216
);
i1_217: lut_h4_idwt
	port map(
		address => yl_54, result => x1_217
);
i1_218: lut_h1_idwt_neg
	port map(
		address => yh_54, result => x1_218
);
i1_219: lut_h3_idwt_neg
	port map(
		address => yh_55, result => x1_219
	);
xout_109 <= x1_216 + x1_217 + x1_218 + x1_219;
i1_220: lut_h2_idwt
	port map(
		address => yl_56, result => x1_220
);
i1_221: lut_h4_idwt
	port map(
		address => yl_55, result => x1_221
);
i1_222: lut_h1_idwt_neg
	port map(
		address => yh_55, result => x1_222
);
```

```
i1_223: lut_h3_idwt_neg
	port map(
		address => yh_56, result => x1_223
	);
xout_111 <= x1_220 + x1_221 + x1_222 + x1_223;
i1_224: lut_h2_idwt
	port map(
		address => yl_57, result => x1_224
);
i1_225: lut_h4_idwt
	port map(
		address => yl_56, result => x1_225
);
i1_226: lut_h1_idwt_neg
	port map(
		address => yh_56, result => x1_226
);
i1_227: lut_h3_idwt_neg
	port map(
		address => yh_57, result => x1_227
	);
xout_113 <= x1_224 + x1_225 + x1_226 + x1_227;
i1_228: lut_h2_idwt
	port map(
		address => yl_58, result => x1_228
);
i1_229: lut_h4_idwt
	port map(
		address => yl_57, result => x1_229
);
i1_230: lut_h1_idwt_neg
	port map(
		address => yh_57, result => x1_230
);
i1_231: lut_h3_idwt_neg
	port map(
		address => yh_58, result => x1_231
	);
xout_115 <= x1_228 + x1_229 + x1_230 + x1_231;
i1_232: lut_h2_idwt
	port map(
		address => yl_59, result => x1_232
);
i1_233: lut_h4_idwt
	port map(
		address => yl_58, result => x1_233
);
i1_234: lut_h1_idwt_neg
	port map(
		address => yh_58, result => x1_234
);
i1_235: lut_h3_idwt_neg
	port map(
		address => yh_59, result => x1_235
	);
xout_117 <= x1_232 + x1_233 + x1_234 + x1_235;
i1_236: lut_h2_idwt
	port map(
		address => yl_60, result => x1_236
);
i1_237: lut_h4_idwt
	port map(
		address => yl_59, result => x1_237
);
```

```
i1_238: lut_h1_idwt_neg
	port map(
		address => yh_59, result => x1_238
);
i1_239: lut_h3_idwt_neg
	port map(
		address => yh_60, result => x1_239
	);
xout_119 <= x1_236 + x1_237 + x1_238 + x1_239;
i1_240: lut_h2_idwt
	port map(
		address => yl_61, result => x1_240
);
i1_241: lut_h4_idwt
	port map(
		address => yl_60, result => x1_241
);
i1_242: lut_h1_idwt_neg
	port map(
		address => yh_60, result => x1_242
);
i1_243: lut_h3_idwt_neg
	port map(
		address => yh_61, result => x1_243
	);
xout_121 <= x1_240 + x1_241 + x1_242 + x1_243;
i1_244: lut_h2_idwt
	port map(
		address => yl_62, result => x1_244
);
i1_245: lut_h4_idwt
	port map(
		address => yl_61, result => x1_245
);
i1_246: lut_h1_idwt_neg
	port map(
		address => yh_61, result => x1_246
);
i1_247: lut_h3_idwt_neg
	port map(
		address => yh_62, result => x1_247
	);
xout_123 <= x1_244 + x1_245 + x1_246 + x1_247;
i1_248: lut_h2_idwt
	port map(
		address => yl_63, result => x1_248
);
i1_249: lut_h4_idwt
	port map(
		address => yl_62, result => x1_249
);
i1_250: lut_h1_idwt_neg
	port map(
		address => yh_62, result => x1_250
);
i1_251: lut_h3_idwt_neg
	port map(
		address => yh_63, result => x1_251
	);
xout_125 <= x1_248 + x1_249 + x1_250 + x1_251;
i1_252: lut_h2_idwt
	port map(
		address => yl_0, result => x1_252
);
```

```
        i1_253: lut_h4_idwt
                port map(
                        address => yl_63, result => x1_253
        );
        i1_254: lut_h1_idwt_neg
                port map(
                        address => yh_63, result => x1_254
        );
        i1_255: lut_h3_idwt_neg
                port map(
                        address => yh_0, result => x1_255
                );
        xout_127 <= x1_252 + x1_253 + x1_254 + x1_255;
end architecture;
```

Appendix C:

C Code to generate LUT Values

```
#include <iostream>
#include <stdio.h>
#include <math.h>

short decimal2binary(unsigned long decimal_value, char
binary_value[32])
{
    short index,significant_digits=0;
    unsigned long temp_value;
    for(index=31;index>=0;index--)
    {
        // temp_value=decimal_value/pow(2,index)
        temp_value=decimal_value/(1<<index);
        if(temp_value>0)
        {
            binary_value[index]=(char)('0'+temp_value);
            // decimal_value=decimal_value%pow(2,index)
            decimal_value=decimal_value%(1<<index);

            if(!significant_digits)
                significant_digits=index;
        }
        else
        {
            binary_value[index]='0';
        }
    }
    return significant_digits;
}

int main (int argc, char * const argv[]) {
    FILE *fp;
    //char *mode = "w";
    char outputFilename[] = "out.txt";
    int i,j;
    short significant_digits,index;
    char binary_value[32];

    fp = fopen(outputFilename, "w");
    for (i = 0; i <= 255; i++){
        j = int(round(0.2241*256*i));
        significant_digits=decimal2binary(j,binary_value);
        fprintf(fp,"'");
        /*if (significant_digits >= 8)
        {*/
                for ( index = 16; index >= (8); index--)
```

```
                    fprintf(fp,"%c",binary_value[index]);
        /*}
        else
        {
                for(index = 7; index >= 0; index--)
                    fprintf(fp,"%c",binary_value[index]);
        }*/
        fprintf(fp,"',\n");
    }
    fclose(fp);
}
```

C Code to Generate VHDL Code for IDWT (highpass)

```
#include <iostream>
#include <stdio.h>
#include <math.h>

int main (int argc, char * const argv[]) {
    FILE *fp;
    char outputFilename[] = "out.txt";
    int i, k;
    int counter_in, counter_out;
    int counterh;
    fp = fopen(outputFilename, "w");
    counterh = 4;
    counter_in = 0;
    k = 0;
    for (i = 0; i <= 255; i++)
    {
        fprintf(fp,"i1_");
        fprintf(fp,"%u",i);
        fprintf(fp,": lut_h");
        fprintf(fp,"%u",counterh);
        counterh = counterh - 1;
        fprintf(fp,"\n");
        fprintf(fp,"\t");
        fprintf(fp,"port map(");
        fprintf(fp,"\n");
        fprintf(fp,"\t");
        fprintf(fp,"address => xin_");
        fprintf(fp,"%u",counter_in);
        counter_in = counter_in + 1;
        fprintf(fp,", result => x1_");
        fprintf(fp,"%u",i);
        fprintf(fp,"\n");
        fprintf(fp,");");
        fprintf(fp,"\n");
        if (counterh == 0)
```

```
        {
            counterh = 4;
            fprintf(fp,"yh_");
            fprintf(fp,"%u",counter_out);
            counter_out = counter_out + 1;
            fprintf(fp," <= x1_");
            fprintf(fp,"%u",i-1);
            fprintf(fp," + x1_");
            fprintf(fp,"%u",i-2);
            fprintf(fp," + x1_");
            fprintf(fp,"%u",i-3);
            fprintf(fp," + x1_");
            fprintf(fp,"%u",i);
            fprintf(fp,";\n");
            fprintf(fp,"\n");
            k = k + 2;
            counter_in = k;
        }
    }
    fclose(fp);
}
```

C Code to Generate Thresholding VHDL code

```
#include <iostream>
#include <stdio.h>
#include <math.h>

int main (int argc, char * const argv[]) {
    FILE *fp;
    char outputFilename[] = "out.txt";
    fp = fopen(outputFilename, "w");
    for (int i = 0; i <= 63; i++)
    {
        fprintf(fp,"if yh_");
        fprintf(fp,"%u",i);
        fprintf(fp," <= thld then");
        fprintf(fp,"\n");
        fprintf(fp,"\t");
        fprintf(fp,"yhout_");
        fprintf(fp,"%u",i);
        /*fprintf(fp," <= yl_");
        fprintf(fp,"%u",i);
        fprintf(fp," - thld;");*/
        fprintf(fp," <= '000000000';");
        fprintf(fp,"\n");
        fprintf(fp,"else\n");
        fprintf(fp,"\t");
        fprintf(fp,"yhout_");
```

```
            fprintf(fp,"%u",i);
            fprintf(fp," <= yh_");
            fprintf(fp,"%u",i);
            fprintf(fp,";\n");
            fprintf(fp,"end if;\n");
            fprintf(fp,"\n");
        }
        fclose(fp);
    }
```