Top-Down Design of Digital Signal Processing Systems

by

Amy M. Singer

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degrees of
Bachelor of Science in Electrical Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science
at the Massachusetts Institute of Technology

June 1996

Author ______________________________
Department of Electrical Engineering and Computer Science
May 10, 1996

Certified by ______________________________
Alan V. Oppenheim
Distinguished Professor of Electrical Engineering
Thesis Supervisor

Accepted by ______________________________
F. R. Morgenthaler
Chairman, Department Committee on Graduate Theses

Top-Down Design of Digital Signal Processing Systems

by

Amy M. Singer

Submitted to the
Department of Electrical Engineering and Computer Science

May 10, 1996

In Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Electrical Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

A methodology for top-down design of digital signal processing systems is described. The top-down design process can be viewed as a model which outlines the key steps necessary for successfully mapping an algorithm to implementation. A design exercise is detailed to illustrate how the top-down design process is applied. The discrete wavelet transform is mapped from the algorithm to an architectural description using top-down design. A new strategy for implementing the discrete wavelet transform is proposed.

Thesis Supervisor: Alan V. Oppenheim
Title: Distinguished Professor of Electrical Engineering

The work for this thesis was carried out in the summer and fall semester of 1995 at Motorola Inc., Scottsdale, Arizona as part of the VI-A internship program at MIT.

Acknowledgments

I wish to express my sincere thanks to my MIT advisor, Professor Alan Oppenheim, who not only helped me find the VI-A group to do this thesis, but also has been a mentor and positive influence on my education at MIT. He is truly one of the greatest teachers I have ever had, and I am grateful to have gotten the chance to work with him.

Next, I wish to thank my company supervisor, Dr. Lanny Mullens, for giving me the opportunity to work on this project. He provided excellent guidance, but also gave me freedom in developing my project and responsibility for providing results, so that I was able to learn on my own, in a very "hands-on" manner. I would also like to thank Susan Gilfeather for being my technical mentor throughout the completion of this project. She took time from her busy schedule to guide me through the different stages of my project, and supported my work despite my constant battles with the software, theory and time constraints. Also, I'd like to thank the many others at Motorola GSTG who from time to time gave me a hand. Without these people I would have not been able to complete my VI-A thesis in a year.

I would also like to thank a few people who enhanced my life as a student at MIT, and who were always willing to help out or give advice. First, my big brother Andy, who has been both a constant source of pressure and support. Without Andy I would have never been so successful at MIT. Also, my partner in crime, Agnieszka Reiss, without whom I could have never made it through Course Six. Also, Jeff Ludwig for helping me with everything from wavelet theory to being a surrogate big brother from time to time.

Finally, I wish to thank my parents, and my fiancé, John Guzek, for their loving support during the pursuit of my engineering studies.

Table Of Contents

List Of Figures

List Of Tables

Chapter 1

Introduction

1.1 Top-Down Design

Understanding the process required to transform an algorithm into a realizable architecture is essential for developing efficient digital signal processing systems. Top-down design is widely employed in the development of complex and costly systems because it allows system-level design and initial requirements to flow down to each subsequent design stage. Often system design focuses on requirements and not on the design implementation. Top-down design tries to provide a framework for taking an initial algorithm and mapping it seamlessly down to its final implementation.

In general, a top-down design methodology consists of system requirements definition, algorithm analysis, fixed-point mathematical analysis, resource analysis, and architecture implementation [1]. Defining the system requirements entails combining customer or application-specific constraints with general signal processing and other optimization criteria. Algorithm analysis is used to optimize and completely specify the functionality of the algorithm. Fixed-point analysis is performed to determine optimal signal widths while considering noise degradation and the cost of word-lengths for a fixed-point implementation. Resource analysis determines how to best partition the architecture in terms of control, storage, processing elements and data communication.

Finally, architecture implementation maps the original algorithm onto an architectural solution such that the functional mapping is preserved.

1.2 Scope of Thesis

The purpose of this thesis is to define and verify a top-down design process and demonstrate the process with a specific algorithm. The steps involved in implementing the specific algorithm described in this paper are to (1) define the unit requirements, (2) create and follow a simulation plan, (3) perform tradeoffs, algorithm analysis, and fixed-point analysis, (4) identify candidate architectures, and (5) generate and analyze a Very High Speed Integrated Circuit Hardware Description Language (VHDL) description of the architecture. The top-down design process generally proceeds beyond the VHDL level, but those steps of the process are not the focus of this thesis.

Cadence's Signal Processing WorkSystem (SPW) is used to demonstrate the top-down design process for a specific design exercise. SPW is chosen because it allows the user to quickly simulate high level designs, to easily create several design iterations, to graphically represent a signal data flow as a block diagram, to create and monitor signals relevant to the system design, to design and analyze filters, and to design hardware that automatically generates VHDL. In order to verify that the functionality of the initial algorithm is kept intact throughout the design process, it is important to be able to use the same input stimulus and generate the same output at each level of the design process. SPW allows the designer to use the same input and compare outputs from every design level including the floating-point algorithm, fixed-point model, architecture model and VHDL description of the function being mapped.

The process is applied to a specific algorithm in order to illuminate the intricacies and perhaps provide a better description of the design process. The discrete wavelet transform (DWT) is the test algorithm for the design process. The DWT is an excellent test case because it is easily interpreted in terms of basic signal processing elements and deals with multirate signal processing. The DWT is successfully implemented in an architecture using top down design and SPW.

The organization of this thesis is as follows. In Chapter 2, background on the DWT is provided. The DWT is described in the context of a filterbank, which lays the groundwork for its implementation. Chapter 3 describes the general top-down design process. In Chapter 4, the top-down DWT design exercise from algorithm to implementation is presented. The specific steps of the DWT design process illustrate how to apply the top-down design process model. Chapter 5 summarizes the accomplishments of this thesis, and gives some concluding remarks about the design process.

Chapter 2

The Discrete Wavelet Transform (DWT)

2.1 The Wavelet Transform

The Wavelet Transform (WT) is particularly useful for analyzing non-stationary signals because it provides better resolution at high and low frequencies than a classical Fourier transform or even a short-time Fourier Transform (STFT) [2]. Time resolution localizes a signal in time using windowing, and frequency resolution gathers detailed frequency information about a signal by focusing on a center frequency. The STFT computes the Fourier transform of the product of the input and a window, and then shifts the window in time and computes the Fourier transform again [3]. The input, x[n], and window v[n] are related by

$$X_{STFT}(e^{j\omega}, m) = \sum_{n=-\infty}^{\infty} x[n]v[n-m]e^{-j\omega n} \, . \tag{2.1}$$

The STFT introduces the notion of a time-frequency grid, as shown in figure 2.1a, where each intersection represents a sample of X_{STFT}. The same window is used at all frequencies in a STFT, so the window size creates a tradeoff between time localization and frequency resolution [3].

In contrast, the size of the window used at each frequency in a wavelet transform is chosen such that the relative bandwidth stays approximately the same. The bandwidth itself does not stay the same; it adjusts itself with frequency so that as the window in time gets larger, the bandwidth gets narrower. In other words, the ratio of bandwidth to center frequency stays approximately constant [3]. Or put more simply, the WT takes larger time steps at low frequencies, corresponding to smaller spacing between frequencies. The time-frequency grid for the wavelet transform is shown in figure 2.1b. Note from the figure that the wavelet transform has sharp time localization at high frequencies and sharp frequency resolution at low frequencies.

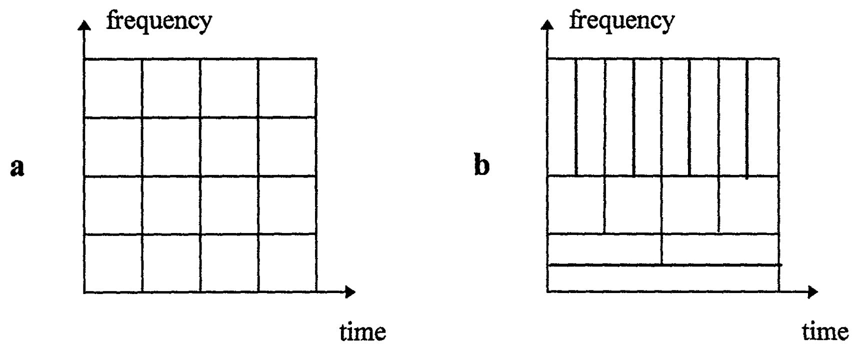


Figure 2.1 Time-frequency plane a) for the STFT b) for the WT

The WT is a decomposition of a signal onto a set of basis functions, or wavelets. All of the wavelets are contractions, dilations, or shifts of a single wavelet, w(t)

$$w_{a,s}(t) = |a|^{-1/2} w\left(\frac{t-s}{a}\right) \qquad (2.2)$$

where a is the scale and s is the shift [4]. The factor $|a|^{-1/2}$ ensures that the rescaled wavelets have unit energy, i.e. $\|w_{a,s}\| = \|w\|$.

Thus, the wavelet can be interpreted as a bandpass filter [2]. The wavelet transform is an inner product between the input signal x(t) and the wavelets, as seen by

$$X_{WT}(a,s) = \int x(t) w_{a,s}(t) dt . \qquad (2.3)$$

The inverse wavelet transform is given by

$$x(t) = c \int \int X_{WT}(a,s) w_{a,s}(t) \frac{da \cdot ds}{a^2} \qquad (2.4)$$

where c is a constant depending on the wavelet [2].

2.2 The Discrete Wavelet Transform

The discrete wavelet transform (DWT) computes sampled coefficients of the wavelet transform, where the time and scale parameters are discrete. Typically for the DWT a is chosen to be 2^j and s is usually $k2^j$, where j is the octave and k is a constant [5]. The DWT computes the wavelet coefficients on J octaves as

$$X_{DWT} = \sum_n x[n] h_j[n - 2^j k] \quad \text{for j= 1,...,J.} \qquad (2.5)$$

The inverse DWT computes the scaling coefficients as

$$X_{IDWT} = \sum_n x[n] g_j[n - 2^j k] \quad \text{for j=1,...,J.} \qquad (2.6)$$

The filters, h[n] and g[n] compute the analysis and synthesis wavelets respectively.

Multiresolution analysis is easily accomplished using these filters to decompose and reconstruct discrete-time signals. The DWT algorithm can be represented in a tree-structured filter bank [3] by splitting the signal into its high and low frequency components, downsampling by a factor of two, and iterating this same procedure on the low frequency branch as shown in Figure 2.2. The filter bank structure makes the DWT particularly efficient for discrete-time signal processing.

The two filters h[n] and g[n] are halfband highpass and lowpass filters respectively. Each stage of band-splitting and downsampling is referred to as an octave. The number of octaves, J, is related to the length of the input signal, N, by

$$J \leq \log_2 N \qquad (2.7)$$

Depending on the application, it may be desirable to choose the number of octaves for a finer time-frequency resolution than for others. Practical implementations rarely exceed four octaves. Some applications for which the DWT and its inverse have been applied include signal and image coding and compression; speech, signal and system analysis; and stochastic, acoustic, and seismic signal processing [6].

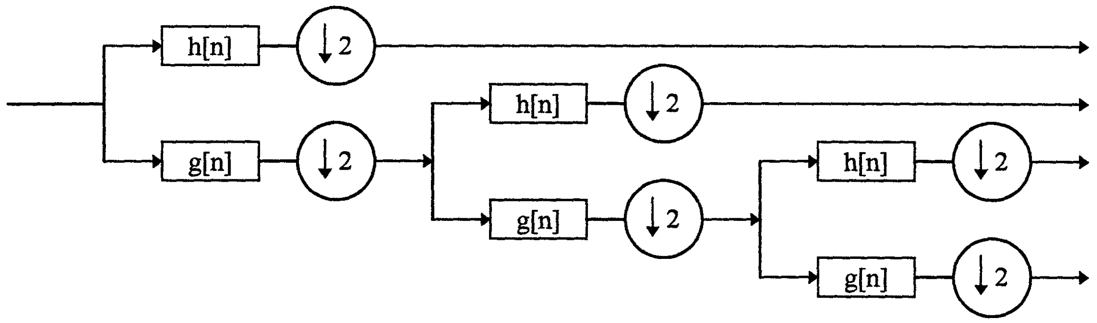


Figure 2.2: Computation of the DWT using a tree-structured filter bank.

2.3 The DWT as part of a Perfect Reconstruction (PR) System

There exists a class of wavelet analysis and synthesis filters for which the input can be perfectly reconstructed from its wavelet coefficients. The inverse discrete wavelet transform (IDWT) is computed using upsampling by a factor of two followed by reconstruction highpass and lowpass filters as shown in Figure 2.3.

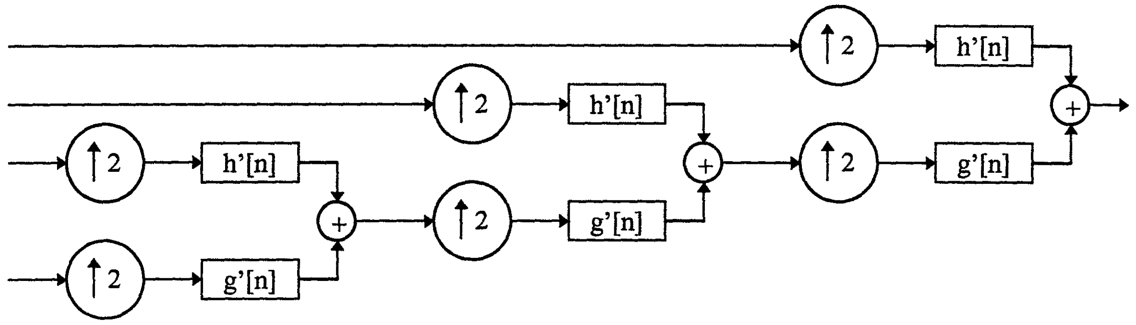


Figure 2.3: Computation of the IDWT using synthesis wavelet filters.

Biorthogonal wavelets can be implemented as linear phase FIR filters that exhibit perfect reconstruction [7], making them particularly desirable for speech and image processing applications. For biorthogonal wavelets, the relationship between the analysis and synthesis wavelet filters is given by

$$\begin{aligned} h[n] &= (-1)^n g'[n] \\ h'[n] &= (-1)^{n+1} g[n] \end{aligned} \tag{2.8}$$

where h'[n] and g'[n] are the highpass and lowpass reconstruction filters respectively.

Chapter 3

Top-Down Design Process

The design process from algorithm to VHDL has not been well documented. The front end of any design process needs careful planning and execution to ensure a successful finished product. Since DSP systems are often complex and difficult to design, there is a great deal of interest in using a structured methodology to plan, develop, and design them efficiently. Top-down design is significant because decisions are made no sooner than is necessary, meaning that at the highest levels of abstraction, the focus is on the algorithm functionality as opposed to the implementation.

The steps involved in the top-down design process are: defining the system requirements; performing the algorithm analysis; creating and verifying the fixed-point model; identifying and designing candidate architectures; and generating and verifying the VHDL model. The top-down design process defined in this chapter is one of the main results of this thesis, and is the model that the author found best representative of what takes place when mapping an algorithm to a hardware description.

These steps are not entirely sequential; many of them are iterative and feed back into one another. For example, the system requirements are defined at the beginning of the process, but as the design progresses to lower levels of abstraction, the requirements need to be updated. Updating the requirements is a refining or an expansion of the previous set of requirements, such that there is a functional decomposition for every design level. The simulation plan generated at the beginning of the process verifies that each design level is complete. The plan calls for comparison between the initial algorithm output and subsequent design stage outputs. Also, tradeoffs occur at virtually every level of the design process and often times several tradeoffs are made within a particular design stage. The general design process flow is shown in Figure 3.1.

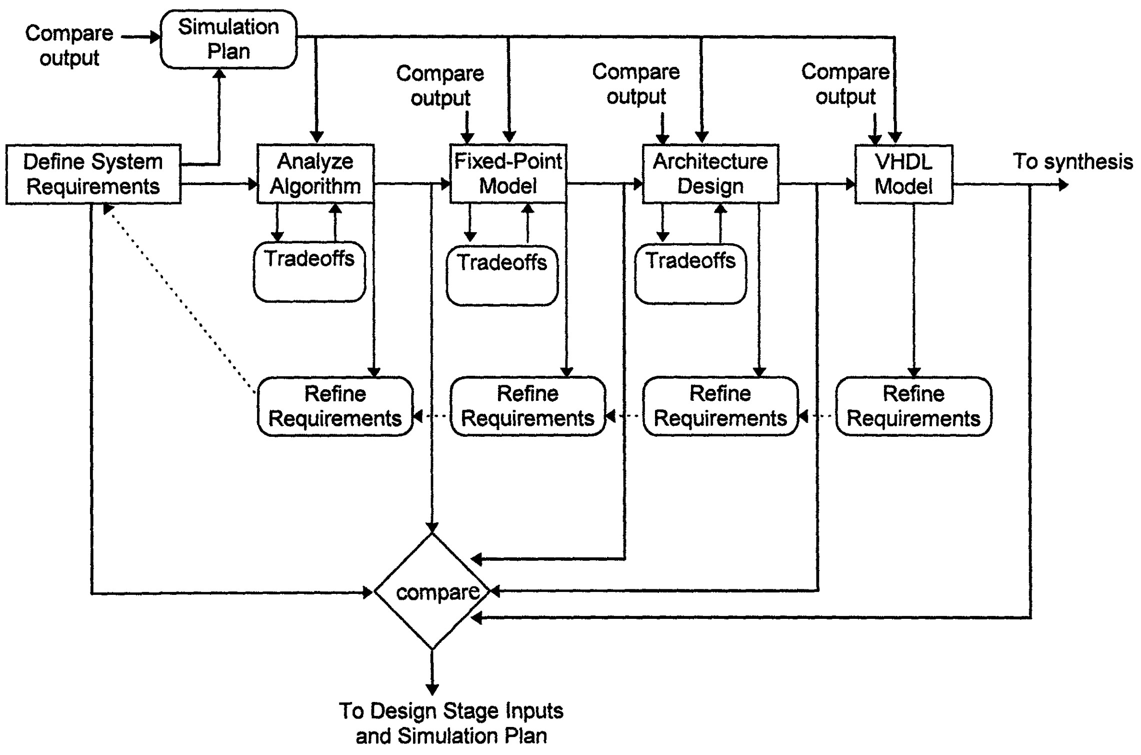


Figure 3.1: General top-down design process.

3.1 Defining System Requirements

The first step taken once an algorithm has been chosen is to define the system requirements. Initially this step entails defining the algorithm and the high level signal processing blocks needed to compute the algorithm correctly. Tradeoffs may be necessary to determine the specific algorithm to be performed. The intended applications for the algorithm will often illuminate what sorts of tradeoffs are needed, if any.

The requirements in a top-down design process will necessarily get narrower as the design proceeds into lower levels of abstraction. Thus the initial unit requirements need not try to encompass every possibility that may arise in the future; they only need to be specific enough to map the mathematical algorithm to a data flow. The system requirements are usually refined and allocated to the next level after each design stage has been completed. In many cases, the sampling rate(s) will drive the speed of the finished product. However, in a floating-point data flow, the processing speed is not relevant to understanding how the algorithm works. The sampling rate is an example of a specification that may be given earlier than needed. It is important in top-down design to recognize the difference between the requirements that drive the current process stage and specifications to be incorporated in a later stage.

3.2 Simulation Plan

As a result of defining the unit requirements, a simulation plan is generated for the design process. The simulation plan is a guideline for verifying that the mapping is intact at each level in the design process. Verification ensures that the design meets the system requirements, and that the design does not exhibit unexpected or undesirable behavior. With the existence of commercial tools to model not only the signal processing flows but also actual hardware designs, the tool simulation capabilities can often significantly speed up the cycle time for much of the design process.

The simulation plan is designed to verify at every level of the design process that the mapping of the original algorithm functionality is preserved. Examining the floating-point model of the algorithm is usually the first step in defining a simulation plan. The output from the floating-point model can serve as a test signal to compare with subsequent design stage outputs.

The fixed-point model is created to determine the bit representation of the data flowing through the algorithm such that the error due to quantization and fixed-point computation does not significantly degrade the output. Different systems will have different tolerance margins for output error. Using an objective or subjective measure of the degraded output, the system requirements and simulation plan can eliminate solutions that are not within the error margin. For instance, speech signals can be played on a commercial tool to find the threshold for audible distortion.

Once the algorithm has been simulated as a fixed-point model and the error margin is within the requirements, the simulation plan then calls for a representation at the hardware description level. The process of going from algorithm to an architecture description is not automatic, it involves moving from unlimited computational resources to limited ones and requires design by hand. Numerous trade-studies and design iterations are required to bridge the gap to an architecture.

Broadly stated, the issues to consider along with the functional mapping of the algorithm include designing control hardware, input/output capabilities, memory and storage capacity, and computational complexity. The design process will always need some sort of user input because it takes the design from a sample driven data flow to a sequential, instruction driven system. The design steps will follow more automatically from fixed-point models to architecture when technology can accommodate implementing data flows directly. As the verification, the output of the hardware description should match the floating point output.

The simulation plan finally calls for a VHDL description of the architecture design. The VHDL must be simulated to match the output of the floating-point model before the design process is complete. In the complete top-down process, the simulation plan continues down to the implementation in hardware, even though this thesis discussion is limited to simulation at the VHDL level.

3.3 Tradeoffs

A good design process will consider all feasible tradeoffs within a trade space. Top-down design enhances tradeoffs because it allows different constraints to be considered at different times. Tradeoffs are made within each process level, and are performed by weighing all of the possible options while considering the design constraints. The result of a trade study is determined by evaluating the different options against the system requirements and determining which solution best fits within those constraints. For example, even at the architecture level, where the hardware can be described in VHDL, the designer has the flexibility of synthesizing the real hardware a number of different ways. However, in top-down design, there is no need to be concerned with physical implementation until the process is actually at that level.

3.4 Algorithm Analysis

Algorithm analysis results in a floating-point model of the algorithm, which includes determining the functional signal processing blocks which comprise the algorithm. Tradeoffs at this design stage are performed to decide which functional blocks can be used. The tradeoffs compare performance characteristics such as simulation time, latency, and overall design to choose the functional blocks to best represent the algorithm. The results of simulating this algorithm serve as a testbench from which lower design stage outputs are compared. Initially, the resources are unlimited. Then, the system requirements are refined to reflect information gained from the algorithm analysis such as resource limits on the maximum number of computation elements needed.

3.5 Fixed-Point Model

The fixed-point model is simply a representation of the algorithm in terms of fixed-point functional blocks. The precision of the algorithm is now hindered by the computational accuracy, which can introduce errors due to word-length, rounding modes and overflow/underflow. Quantization error is introduced in the conversion from floating to fixed-point. Tradeoffs are performed to determine the level of fixed-point quantization error that is low enough to accurately represent the algorithm, while trying to keep the bus width at a reasonable size. Once the representation in bits is determined for the fixed-point algorithm, the requirements are updated to include specifications on bus width, data word size and signal characteristics for the input and output signals. The simulations at the fixed-point functional level verify that the algorithm has been mapped correctly. If the algorithm is not preserved, the results of the simulation and the specifications are fed back into the fixed-point design stage again. The difference signal between the floating and fixed-point outputs serves as a specification for the architecture output.

3.6 Architecture Design

The architecture design can be thought of as mapping a set of functional tasks onto limited resources using a set of simple instructions. Thus, all of the functions of the algorithm are broken down into subtasks or partitions of the architecture. Each subtask is further broken down into a set of instructions. At this stage, tradeoffs identify and compare different implementations of the functional tasks. Tradeoffs of possible control structures to drive the architecture are determined, along with different processing elements, data storage capacities, and input/output capabilities. Performance and functional issues addressed by the tradeoffs include algorithm accuracy, system performance, ability to upgrade the design, portability, and input/output bandwidth.

Simulations are run to ensure that the architecture has correctly interpreted the algorithm in terms of an instruction driven system implemented in hardware and software. The difference signal between the floating-point output and architecture output should be within the error margin set in the system requirements. The final architecture should be chosen such that all areas within the architecture have been optimized or altered to improve performance, power, size, cost or other specific criteria set by the requirements. Finally, the system requirements are updated with data rates, input/output specifications, and any pre- or post-processing requirements.

3.7 VHDL Model

The final step in this top-down design process is the creation of a VHDL description of the architecture. The VHDL can be automatically generated from the Hardware Design System (HDS) model if the architecture has been designed in SPW. There are other tools which can also generate VHDL automatically, and though this is not part of the general top-down design process, it definitely contributes to ease of going from the architecture to the VHDL description.

The simulation plan verifies that the behavior of the algorithm is accurately implemented by comparing the results of the VHDL simulation with the floating-point output. The difference signal should be within the limit set in the specifications, and it should be no different than that of the architecture level simulation. The system requirements should then be updated to reflect VHDL timing and design considerations to guarantee that the synthesized hardware accurately represents the algorithm.

Chapter 4

Discrete Wavelet Transform Design Exercise

4.1 Top-Down Design of a DWT

In Chapter 3, a top-down design process was outlined. The discrete wavelet transform is now implemented using the top-down design methodology presented. The DWT is mapped from the algorithm description to an architecture design which can be implemented in hardware through its VHDL description.

The algorithm is first modeled in floating-point signal processing blocks to represent all of the signal processing functions. Next, the algorithm is converted into fixed-point signal processing blocks. Conversion from floating-point to fixed-point representation causes quantization, and the system requirements dictate how much degradation the system can tolerate. Use of a commercial tool expedites this step because it is easy to view and compare results of different fixed-point representations. After verifying that the algorithm is still executing correctly with an acceptable amount of quantization error, the algorithm is rebuilt as a description of hardware and software, by means of an architecture.

The architecture development is the biggest leap in the process flow from algorithm to VHDL. Not only do the signal processing functions need to be represented, but also the memory, control and resource sharing need to be determined. The transformation from the high level data flow of the DWT algorithm is achieved by modeling a sequenced, instruction driven system to perform the computation. SPW's Hardware Design System (HDS) allows the user to rapidly test different implementations of the same function. The same analyses done by hand would not generate results of tradeoffs as readily as a commercial tool. Finally, VHDL code is automatically generated and verified for architecture. The rest of this chapter deals with the specific design process steps as they are applied to the DWT.

4.2 DWT System Requirements

The system requirements for implementing the DWT start off very broadly, and get updated as each design stage is completed. The initial specifications are to design a four octave DWT, and to use a perfect reconstruction system as a test bench for functional verification. The signal processing functions which comprise the algorithm are recognized as wavelet filters and downsamplers by two.

The filters are specified to be halfband lowpass and highpass filters that implement biorthogonal wavelets. Biorthogonal wavelet filters are chosen because they exhibit perfect reconstruction. Since biorthogonal wavelets produce finite impulse response (FIR) linear phase filters, they are well-suited to image and speech processing applications. The coefficients for the analysis and synthesis filters are given in Appendix A, Table 1.

At the top level of the design process, an example of an overall system which includes the DWT is considered. The top level perfect reconstruction system is modeled to verify that the DWT coefficients are being computed correctly. In a perfect reconstruction system, the output is exactly the input signal, within a delay. Figure 4.1 shows an example a simple speech coding system. Note that once encoding and fixed-point representations are used, the system no longer exactly reconstructs the input.

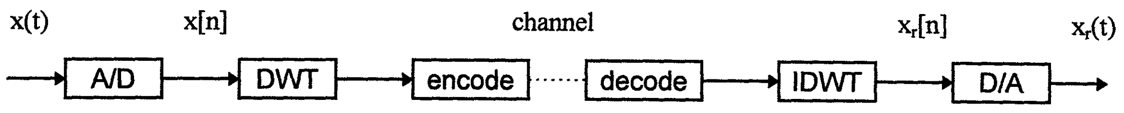


Figure 4.1: A top level speech coding system which includes the DWT.

4.3 DWT Design Simulation Plan

Verification of the functionality of the DWT is accomplished with a simulation plan. Using SPW expedites the simulation plan for the DWT design exercise. The same input stimulus, a sample speech signal, is used for each stage of the DWT design. The simulation plan calls for each DWT design stage to be tested with the speech input, and for the outputs to be compared with the top level output of the perfect reconstruction system. The difference in outputs is computed along with an audio playback of the reconstructed output to determine whether the output degradation is significant. The maximum tolerable error margin is updated into the system requirements.

The floating-point model of the algorithm is first created and simulated to determine the nature of the outputs. The IDWT is used to reconstruct the signal from the DWT coefficients to verify the correct algorithm behavior. The floating-point DWT does not experience any quantization and is fed directly into the IDWT, so there is no channel degradation. Thus, at the floating-point level, perfect reconstruction is observed.

From that point on, the outputs of the floating-point DWT are used to verify that at each level of the design process, the current description accurately represents the DWT within some margin of error set by the requirements. The error characteristics for the different design stages are given in Table 4.1 below. The floating-point DWT outputs are encoded to examine the effects of a coding system on perfect reconstruction. The fixed-point model is compared against its floating-point predecessor. The fixed-point model is also simulated with the same coding system as before and compared with the uncorrupted floating-point output. Finally, the architecture outputs are compared against the floating-point results.

Table 4.1 The outputs of each design level are compared against the uncorrupted floating-point output.

Difference signal	Mean	Variance
floating-point system with coding	2.33 e-8	7.78e-11
fixed-point system	9.39 e-7	1.7 e-13
fixed-point system with coding	9.71e-7	7.9e-11
architecture	1.4e-7	2e-15

The VHDL portion of the DWT design exercise is easily verified since SPW has automatic VHDL generation capabilities from its HDS blocks. Since the architecture is modeled entirely in HDS blocks, the VHDL and architectural simulations are identical. This is an advantage of using a tool with automatic VHDL generation capability; the hardware is only designed at the architecture level, which lends itself to the top-down design process.

4.4 DWT Floating-Point Analysis

The DWT is first modeled using floating-point blocks. The filters and downsamplers by two are connected in a simple two band system as shown in Figure 4.2. The outputs of the DWT are verified using the synthesis filters to perfectly reconstruct the input signal. Once perfect reconstruction is achieved in the two band case, octaves are added one at a time to the lowpass branch, resulting in a tree-structured filterbank. Figure 4.3 shows the perfect reconstruction system for a four octave filterbank.

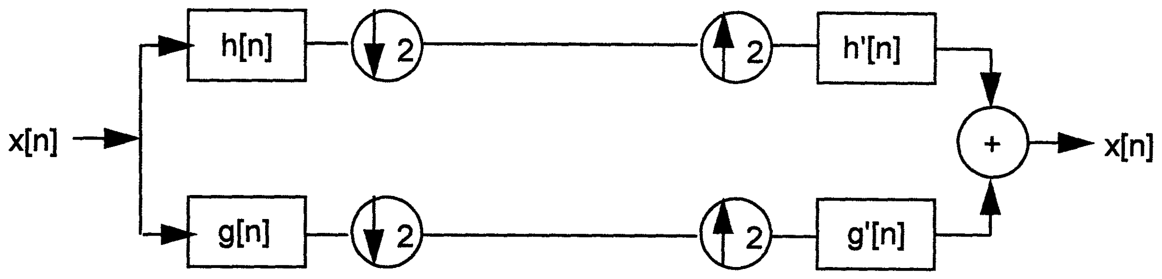


Figure 4.2: A simple two-band perfect reconstruction system is verified using the wavelet filters given in Appendix A.

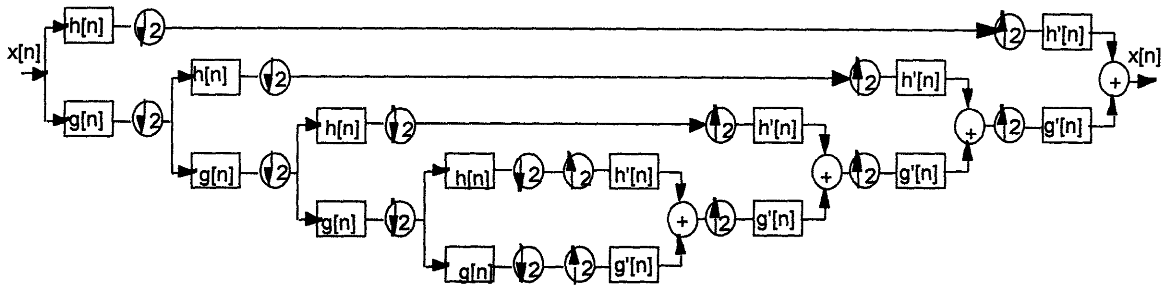


Figure 4.3: A four octave DWT-IDWT perfect reconstruction filter bank.

Initially, perfect reconstruction is not exhibited by the systems, and this is due to the fact that there is a different amount of delay accrued while traversing different "branches" of the tree-structured filterbank, such that the samples are misaligned at the adders on the reconstruction side. By adding a compensatory bulk delay before the adder at each level on the reconstruction side, the system behaved as expected- perfect reconstruction within a delay. Depending on the tool or tools used to do the algorithm analysis, block diagrams and data flows may be computed differently than the user intends. SPW has a sample driven simulator, so the bulk delay is an appropriate fix to the problem. Other fixes may be necessary using other tools.

Once a working four stage DWT simulates accurately, the top level system as depicted in Figure 4.1 is built around the perfect reconstruction system. The purpose of modeling all of the components of the overall system is to ensure that the transform performs in a real-world application as well as it performs in test bench simulations.

Although the only part of the top level system to be exercised through the top-down design process is the DWT, the overall system is modeled to ensure the efficacy of the DWT implementation. Once the transform coefficients are encoded, transmitted across the channel, and decoded, the reconstruction is no longer exact. The difference signal between the uncorrupted wavelet coefficients and the encoded coefficients has a mean of 2.33 e-8 and a variance of 7.78 e-11. The error due to this quantization is determined to be within the requirements, as the reconstructed output does not produce audible distortion.

In a typical speech subband coding application, the quantization is a significant parameter since different frequency bands might be coded with different word-lengths to exploit the time-frequency information contained in each octave [8]. The design of the DWT as well as the coding blocks needs to ensure that the performance is not severely degraded for the target application.

4.5 Fixed-Point DWT Model

Once the floating-point DWT algorithm is modeled, simulated, and verified to meet the system requirements, it is converted to a fixed-point representation of the algorithm. In SPW, this task is not automatic, though it is easily accomplished by replacing each floating-point block with an equivalent fixed-point block. Each filter is rebuilt as a combination of registers, scaling factors, and adders, as shown in Figure 4.4. The downsampling by a factor of two is achieved by running each octave at half the clock rate of the previous octave. The fixed-point four octave DWT is shown in Figure 4.5.

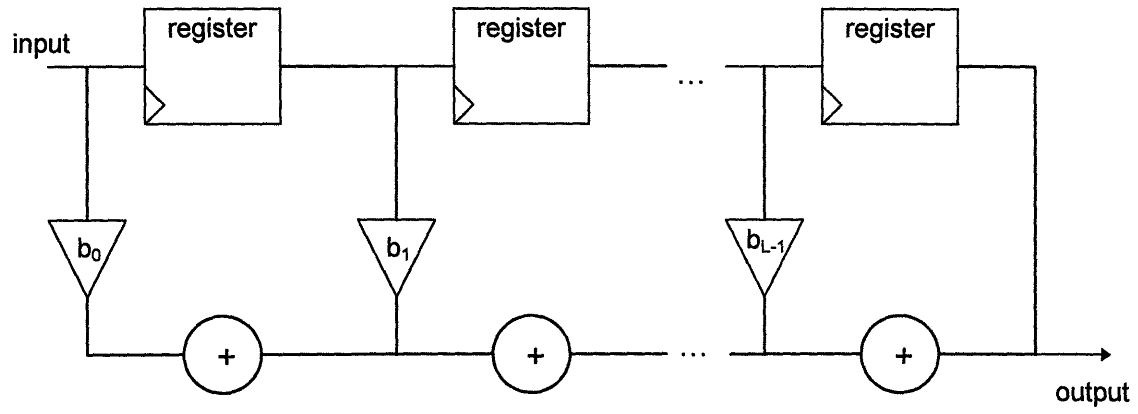
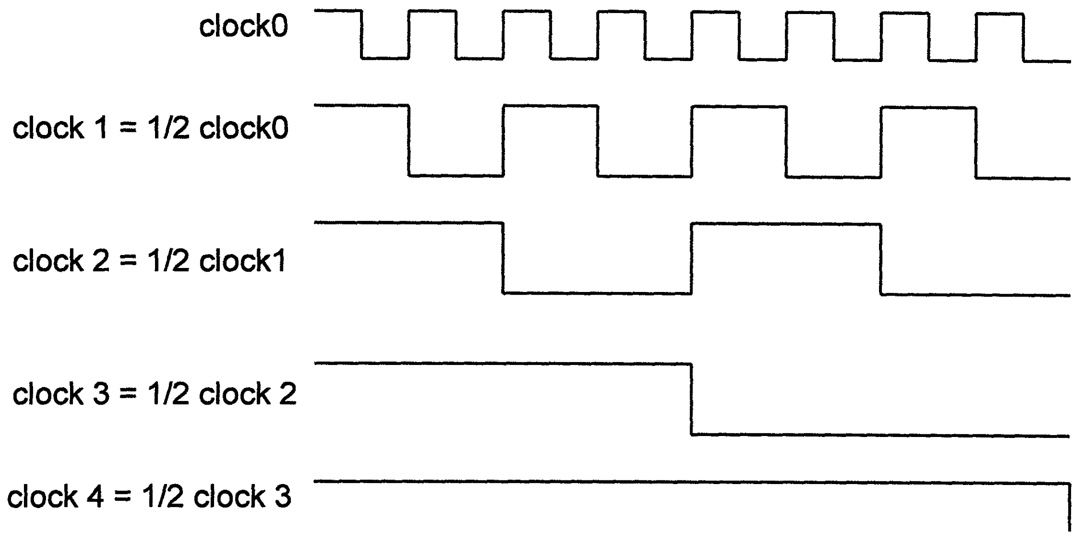


Figure 4.4: A fixed-point version of the filter block. L is the number of filter taps.



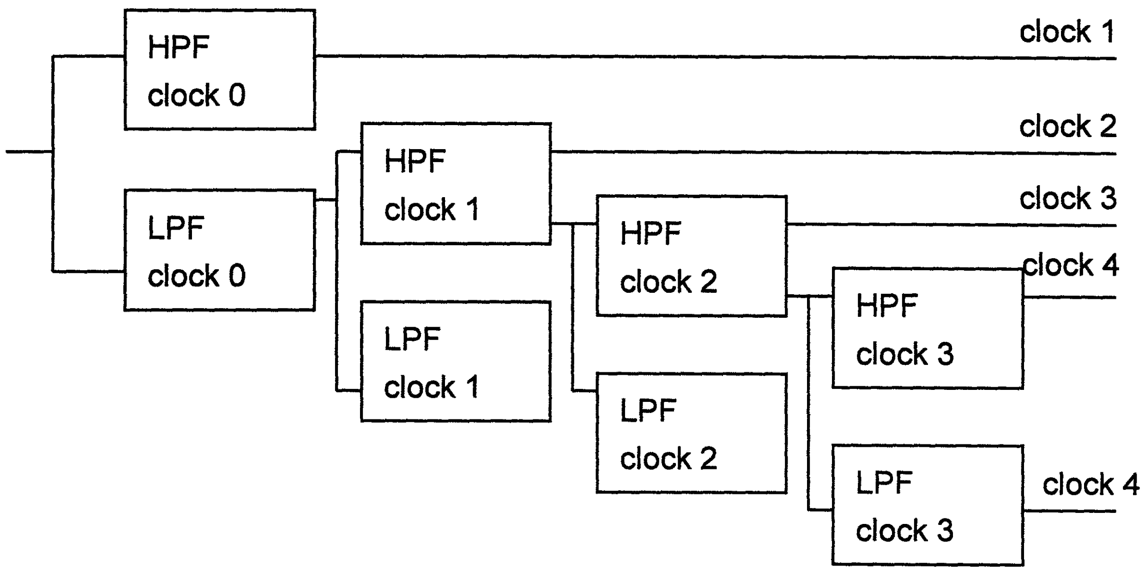
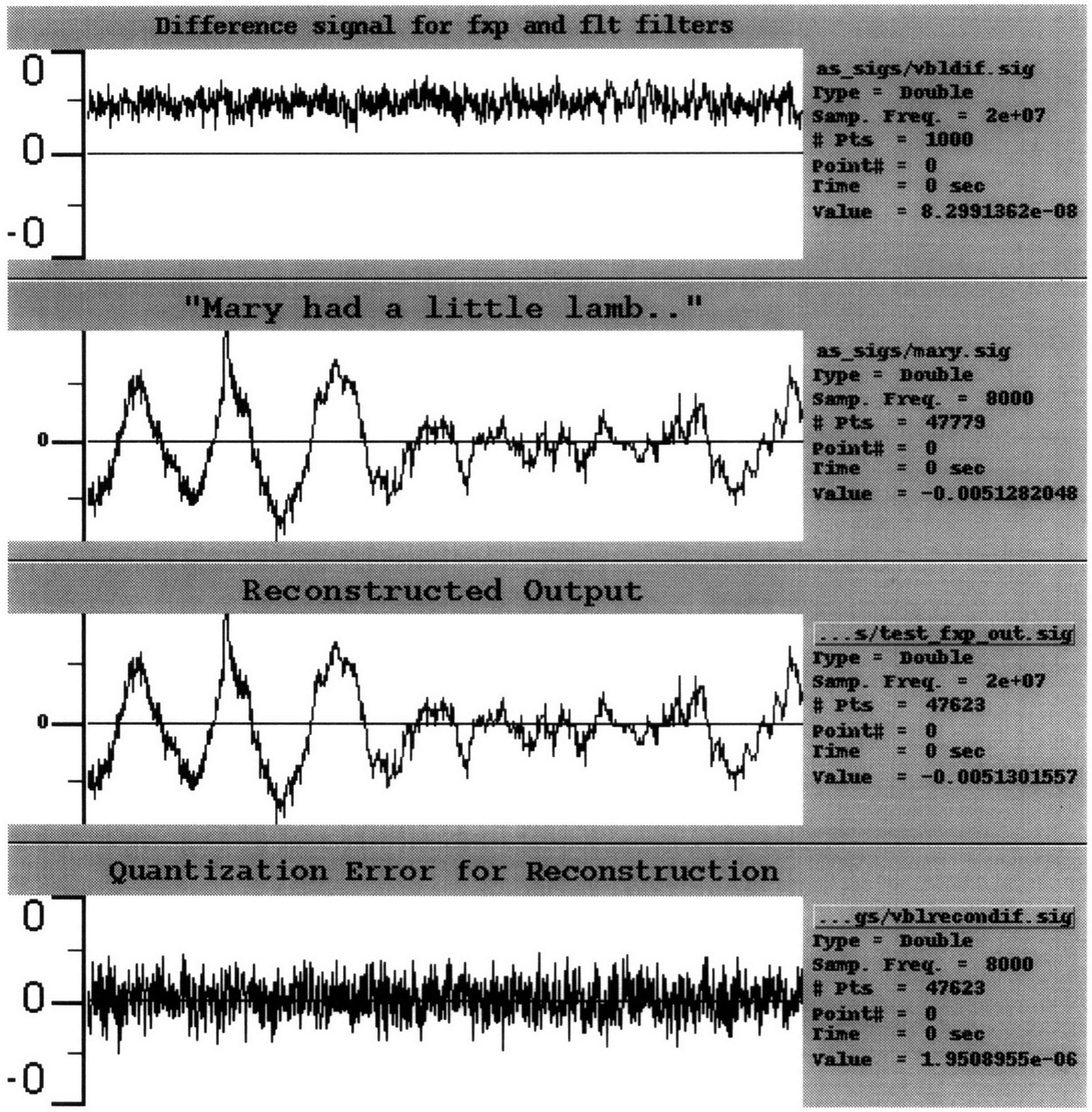


Figure 4.5: The fixed-point DWT uses filters made up as blocks shown in figure 4.4 and runs at 5 different clock speeds. Note that the outputs need to be run at half of the speed of the terminating filter. This design requires a multirate simulation capability.

When the DWT is expressed in terms of fixed-point blocks, the choice of word-length to represent the data is the most critical change from floating-point analysis. The word-length is chosen so that the data is accurate to the desired decimal place, and is determined by setting two raised to the number of bits equal to the desired accuracy. The word length size is chosen based on a trade off between bus size and error such that the output can still be reconstructed without audible distortion. The bus width size is minimized to save cost and area on the target architecture. Using SPW, different word-lengths are examined for the fixed-point DWT. The difference between the floating and fixed-point representation of the filter coefficients is on the order of 10^{-8} with a word-length of twenty-four bits. Once fixed-point computation is introduced, the reconstruction is no longer exact.

The fixed-point DWT is reconstructed and the difference signal with the floating-point output is computed. Figure 4.6 shows the reconstructed output and difference signal along with a comparison between floating and fixed-point outputs. The overall error for a four stage fixed-point DWT has a mean of 9.71e-7 and a variance of 7.9e-11, which is acceptable for reconstructing the output without audible distortion. The difference signal shows that the reconstructed output using a fixed-point model is within the specifications, and thus the top-down design process proceeds to the next level.


Difference signal for fxp and flt filters
0
0
-0
"Mary had a little lamb.."
0
Reconstructed Output
0
Quantization Error for Reconstruction
0
0
-0

```
as_sigs/vbldif.sig
Type = Double
Samp. Freq. = 2e+07
# Pts  = 1000
Point# = 0
Time   = 0 sec
Value  = 8.2991362e-08
```

```
as_sigs/mary.sig
Type = Double
Samp. Freq. = 8000
# Pts  = 47779
Point# = 0
Time   = 0 sec
Value  = -0.0051282048
```

```
...s/test_fxp_out.sig
Type = Double
Samp. Freq. = 2e+07
# Pts  = 47623
Point# = 0
Time   = 0 sec
Value  = -0.0051301557
```

```
...gs/vblrecondif.sig
Type = Double
Samp. Freq. = 8000
# Pts  = 47623
Point# = 0
Time   = 0 sec
Value  = 1.9508955e-06
```

Figure 4.6: The difference signal for the floating and fixed-point filter outputs is displayed on top, followed by the test speech signal and the reconstructed output. The last difference signal shows the quantization in overall reconstruction including coding.

4.6 DWT Architecture Design

Once the algorithm has been successfully converted to fixed-point and the quantization effects are found to be consistent with the system requirements, candidate architectures for implementing the DWT are identified and examined. The main aspects of the DWT architecture design are now discussed.

4.6.1 Filtering

Initially, a strategy needs to be developed for performing the filtering operations of the DWT. There are several methods for implementing FIR filters in hardware. The multiply-accumulate (MAC) is the basic operation in current digital signal processing (DSP) technology, and it is chosen to perform the convolution for each of the two filters in a DWT octave. Building filters out of MAC's saves power and chip area because the number of multipliers and adders per L-tap filter shrinks from L and L-1 to one multiplier and adder.

The basic structure of a MAC FIR filter is shown in Figure 4.7. The filter coefficients are stored in a read-only memory (ROM) and are read out sequentially by a counter into a clocked register. The input data is read into the other clocked register in sync with the filter coefficients so that the convolution is correctly computed. The pipeline diagram in Figure 4.8 shows the contents of the two input registers and the accumulate register at each clock cycle. Another ROM is used as a local microcontroller to sequence the data flow, clear the register, and write the output.

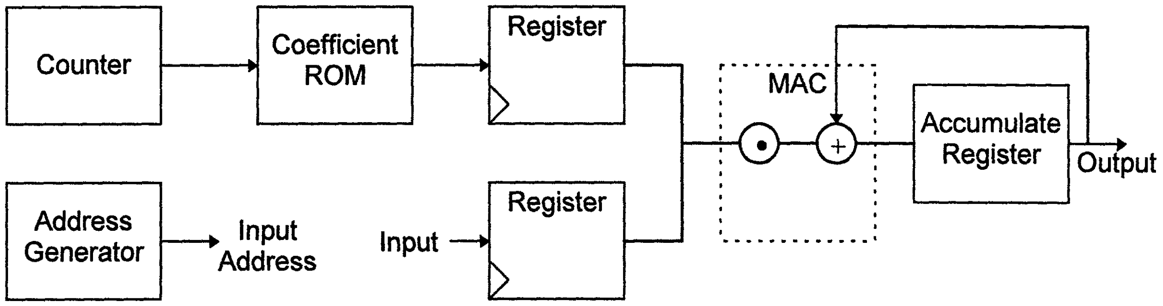


Figure 4.7: The MAC filter structure consists of two input registers, a coefficient ROM, an accumulate register, the multiply-accumulator, and an address generator to sequence

the input values. The control signals actually come from outside the filter, but there is a local microcontroller for each filter.

Clock	R1 in	R1 out	R2 in	R2 out	Accumulate register in	Accumulate register out	comments
disable	clear	junk	clear	junk	clear	junk	
0	h8	0	x0	0	0	0	
1	h7	h8	x1	x0	x0h8	0	
2	h6	h7	x2	x1	x0h8+x1h7	x0h8	
3	h5	h6	x3	x2	x0h8+x1h7+x2h6	x0h8+x1h7	
4	h4	h5	x4	x3	x0h8+x1h7+x2h6+x3h5	x0h8+x1h7+x2h6	
5	h3	h4	x5	x4	x0h8+x1h7+x2h6+x3h5+x4h4	x0h8+x1h7+x2h6+x3h5	
6	h2	h3	x6	x5	x0h8+x1h7+x2h6+x3h5+x4h4+x5h3	x0h8+x1h7+x2h6+x3h5+x4h4	
7	h1	h2	x7	x6	x0h8+x1h7+x2h6+x3h5+x4h4+x5h3+x6h2	x0h8+x1h7+x2h6+x3h5+x4h4+x5h3	
8	h0	h1	x8	x7	x0h8+x1h7+x2h6+x3h5+x4h4+x5h3+x6h2+x7h1	x0h8+x1h7+x2h6+x3h5+x4h4+x5h3+x6h2	
disable	clear	h0	clear	x8	x0h8+x1h7+x2h6+x3h5+x4h4+x5h3+x6h2+x7h1+x8h0	x0h8+x1h7+x2h6+x3h5+x4h4+x5h3+x6h2+x7h1	
0	h8	0	x1	0	clear	x0h8+x1h7+x2h6+x3h5+x4h4+x5h3+x6h2+x7h1+x8h0	write out A
1	h7	h8	x2	x1	x1h8	0	
2	h6	h7	x3	x2	x1h8+x2h7	x1h8	
3	h5	h6	x4	x3	x1h8+x2h7+x3h6	x1h8+x2h7	
4	h4	h5	x5	x4	x1h8+x2h7+x3h6+x4h5	x1h8+x2h7+x3h6	
5	h3	h4	x6	x5	x1h8+x2h7+x3h6+x4h5+x5h4	x1h8+x2h7+x3h6+x4h5	
6	h2	h3	x7	x6	x1h8+x2h7+x3h6+x4h5+x5h4+x6h3	x1h8+x2h7+x3h6+x4h5+x5h4	
7	h1	h2	x8	x7	x1h8+x2h7+x3h6+x4h5+x5h4+x6h3+x7h2	x1h8+x2h7+x3h6+x4h5+x5h4+x6h3	
8	h0	h1	x9	x8	x1h8+x2h7+x3h6+x4h5+x5h4+x6h3+x7h2+x8h1	x1h8+x2h7+x3h6+x4h5+x5h4+x6h3+x7h2	
disable	clear	h0	clear	x9	x1h8+x2h7+x3h6+x4h5+x5h4+x6h3+x7h2+x8h1+x9h0	x1h8+x2h7+x3h6+x4h5+x5h4+x6h3+x7h2+x8h1	
...	..	...	...	...	..	x1h8+x2h7+x3h6+x4h5+x5h4+x6h3+x7h2+x8h1+x9h0	write out A

Figure 4.8: The pipeline diagram above shows the convolution being computed for y_8 and y_9, where h_i is the ith filter coefficient, x_i is the ith input and y_i is the ith output. The registers in this example all have a delay of one clock cycle from input to output.

The first octave of the DWT is built using a MAC filter for each of the lowpass and highpass filters. Controllers local to each filter are built to handle the data flow and bookkeeping for the outputs. The downsample by two is initially accomplished by using different clock speeds as in the fixed-point model. A later architecture design iteration incorporates a better downsampling scheme, and is described in section 4.6.3. The outputs from the MAC filters are compared with the floating-point filter outputs, and the difference signal is verified to be on the order of 10^{-7}. The MAC filters perform the DWT within the error constraints, so the architecture design process continues.

4.6.2 Implementing the Four Octaves

The choice is made to separately implement all four octaves for the DWT. By inspecting the block diagram for the DWT in Figure 2.2, it is clear that the basic octave computation of splitting the data into its high and lowpass components followed by downsampling by a factor of two is an iterative computation. There have been proposed implementations which exploit the fact that there are only two distinct filters used in the computation of the DWT. These implementations build only one highpass and one lowpass filter and control the scheduling of data through these filters [9,10,11,12,13]. However, since the filters built in this thesis use a MAC structure and are not costly in terms of multipliers and adders, all four stages are built separately. Parallel computation is chosen over minimizing the number of filters.

The data is computed from stage to stage and flows through much like the algorithm block diagram. The alternative implementation must wait for each filter to finish computing its result and intersperse higher octave computations in between each set of computations. The result is arrived at more quickly with all of the stages built and running concurrently, and requires less complicated control structures than a shared filter implementation would require.

The architecture is designed in a similar fashion to the algorithm models, by adding subsequent octaves after the first one has been verified to map the algorithm correctly. The design of the second, third, and fourth stage is exactly the same as the first, with a few exceptions. The clock speed of every subsequent stage is half that of the previous stage. Also, subsequent stages cannot compute valid output data until after the previous stage has computed its first output value. For the first L-1 computations of any MAC convolution , the incoming data will consist of at most L-1 zeroes and one input point. Figure 4.9 shows the first nine convolution points for a nine tap filter and the result out of the accumulate register. Because the first convolution of the MAC filter needs only one data point and L-1 zeroes, the second filtering stage only needs to wait until one output of the first stage has been computed before beginning its own computation. The MAC filter implementation is computationally efficient because once the first output value is calculated, the next stage can begin its computation.

convolution	Accumulate result
0	x0h0+0h1+0h2+0h3+0h4+0h5+0h6+0h7+0h8
1	x1h0+x0h1+0h2+0h3+0h4+0h5+0h6+0h7+0h8
2	x2h0+x1h1+x0h2+0h3+0h4+0h5+0h6+0h7+0h8
3	x3h0+x2h1+x1h2+x0h3+0h4+0h5+0h6+0h7+0h8
4	x4h0+x3h1+x2h2+x1h3+x0h4+0h5+0h6+0h7+0h8
5	x5h0+x4h1+x3h2+x2h3+x1h4+x0h5+0h6+0h7+0h8
6	x6h0+x5h1+x4h2+x3h3+x2h4+x1h5+x0h6+0h7+0h8
7	x7h0+x6h1+x5h2+x4h3+x3h4+x2h5+x1h6+x0h7+0h8
8	x8h0+x7h1+x6h2+x5h3+x4h4+x3h5+x2h6+x1h7+x0h8
9	x9h0+x8h1+x7h2+x6h3+x5h4+x4h5+x3h6+x2h7+x1h8

Figure 4.9: The first ten convolutions for a nine tap filter are shown. For the first eight convolutions, note that some of the filter coefficients are actually being convolved with zeroes.

4.6.3 Efficient Downsampling

The initial DWT design uses an inefficient downsample by two method by first filtering all of the data and then subsequently discarding every other point. Therefore, it is more economical to only compute the values which will not be discarded. The reduced computation saves power because the number of MAC's has been cut in half. The new downsampling method is achieved using register files in between each stage, in order to read to and write from storage locations simultaneously. The address pointers to the memory prior to the first filter stage simply need to increment by two, so that if the first convolution requires the input values x_0 through x_8, the second computation requires inputs x_2 through x_{10}, and so on.

The subsequent stages also index the incoming addresses in a similar manner, but they still run at half the clock speed of the previous stage. Since the subsequent stage asks for samples twice as fast as before using the new downsampling method, the clock speed is cut in half so that stage j will always be writing to a location in memory a fixed distance ahead of the read location for stage j+1.

4.6.4 Data Storage

Register files are chosen to store the intermediate data between octaves. They allow simultaneous read and write activities on the same clock cycle, making them ideal for inter-octave storage. The size of the register files is determined by looking at the input storage lifetime, i.e. the amount of clock cycles an input needs to be stored before it is not needed for further computations. For the nine tap lowpass filter, the data is read in sequentially for nine values, then the starting input address jumps by two for the next convolution and continues. The register file size for the nine tap lowpass filter is chosen to be seventeen so that the necessary nine input locations plus the initial offset of eight zeroes can be stored. This size is nearly double what is needed, but makes the startup convolutions and addressing a little simpler. A similar analysis for the seven tap highpass filter yields a register file with thirteen locations.

4.6.5 Control Processing

Each octave has a microcontroller local to each filter to generate the write addresses and control the pipeline. The write pointer is incremented by one after each write until the end of the register file is reached, at which time the addresses wrap around. Also, each filter contains an address generator to determine the read pointer location for the input data. These controls handle the bookkeeping to ensure that the data is stored long enough to be read before being overwritten, and is read and written in the correct order. The choice to create controls local to each filter and to generate circular addressing from within the filters is appropriate for a parallel processor. The only global dependence for the control is the time an octave can begin computation, which involves the current octave itself and the immediately preceding one.

A shared filter architecture would have several disadvantages. For instance, with shared filters, the control circuitry must all be generated from one main control structure. The controls would also have to be much more complex. The outputs from intermediate octaves would have to be stored longer, since there would only be two filters doing the convolutions for four octaves. Therefore, memory size would need to increase and its addressing would have to be more sophisticated to remember where each set of data resides, and when the data can be sent as output. Not only would the control logic and address generators be extremely complex in the shared filter case, but also the latency for output data from the lowest frequency bands would be very high. Furthermore, if the DWT processor is going to be used in a reconstruction system, the lowest frequency bands (or latest octaves) need to be reconstructed first, and therefore need to be computed quickly.

4.6.6 Input-Output (I/O) Processing

The input-output (I/O) processing is the final issue in the DWT implementation. In any design process, the I/O characteristics determine the utility and portability of the finished product. Particular attention should be paid to I/O to avoid creating serious bottlenecks into or out of the processor, which could seriously degrade the overall system performance. The processor is targeted for a single chip implementation. Therefore, the data needs to be brought in at a rate such that the first stage can begin computation as soon as possible.

Register files are not costly in size, so they are employed as input buffers to the first stage. The data must be written into the input register files more slowly than it is read out. Lifetime analysis on the input buffers determines that the input data needed for a computation increments by two input locations for every new output, and each computation takes ten clock cycles in either filter. Thus, from one computation to the next, two locations are freed. For the input buffer, it is then apparent that two register file locations are available to be overwritten once every ten clock cycles. Therefore the data can be written in no faster than one-fifth the clock rate of the first octave, or else the computations will not be able to keep up with the incoming data. This sets the maximum rate at which data can be read into the DWT relative to the fastest clock within the DWT.

For the output processing, the data is again buffered into register files. However, the register file the size needs to be able to hold all of the outputs written from each band long enough to be read out before being overwritten. The data must be sent out in a regular fashion so that it can easily be used by the outside system. To minimize the overall pin count on the chip, the output is placed on a single bus line. A single bus line reduces the overall pin count from five times the word-length to one times the word-length. The buffer sizes for each octave are chosen to hold a fixed amount of outputs to be bursted onto the single output bus in succession.

By examining how often the data is written for each output band, the output buffer sizes are determined. The outputs are written every ten clock cycles for the highest octave band, every twenty for the next band, every forty for the next, and every eighty clock cycles for the last two bands, because each octave runs at half the speed of the previous one. Figure 4.10 shows the write times for each octave band relative to the fastest clock. The first band corresponds to the highest frequency band, the second band corresponds to the next highest frequency band, and so on until the last band, which corresponds to the lowest frequency band. Note that as the frequency bands get smaller, the write signals are asserted less often by a factor of two.

Using the timing of the write signals from each octave as a guide, the output processing can be determined. The control circuitry is set up so that the first buffer reads out all values in the buffer, at which time the second buffer is full and reads out all of its values and so on. Register file sizes are chosen such that the outputs read out from each buffer do not overlap in time, and thus can easily be separated outside of the processor. By putting out the data in a regular order, it is fairly simple to demultiplex the output data. Further processing in an overall system can readily take advantage of this regular pattern of output data, or address generators can be designed to store the outputs. The output burst pattern for the DWT is shown in Figure 4.11. The top signal shows the outputs being burst on a single bus line. The following four signals show the individual burst patterns for each octave output.

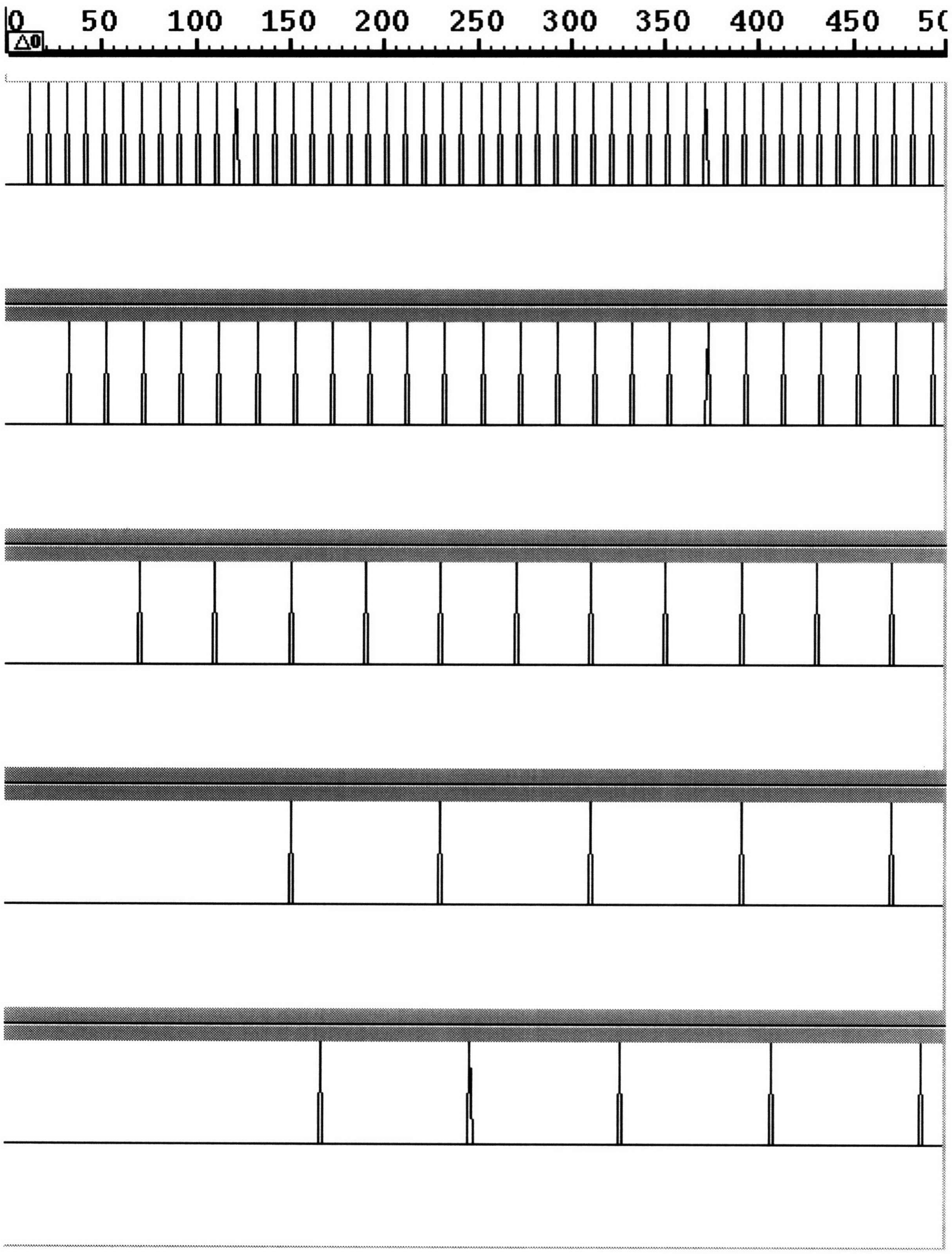


Figure 4.10: The write signals for each band are shown.

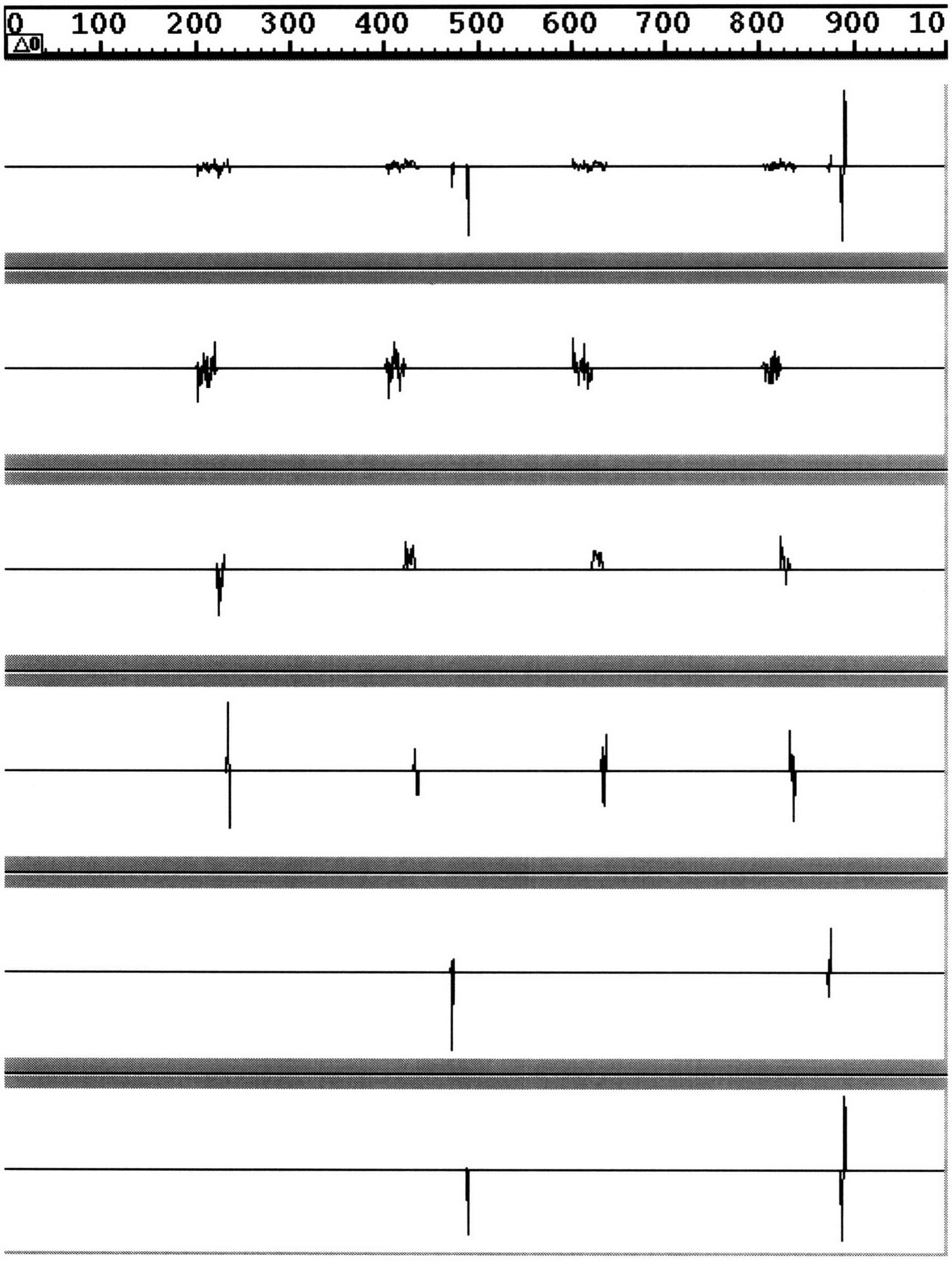


Figure 4.11: The output burst pattern for the speech test signal is shown.

"Very little work has been done in mapping the DWT into [hardware]" [13]. Using a MAC structure for each of the highpass and lowpass filters in an octave, a new architecture for implementing the DWT is developed. Using simple control structures and a practical data storage size, the implementation is not complicated to build. A direct implementation of each of the filters is typically costly, but with MAC filters, only eight multipliers and eight adders are needed to implement a four octave DWT. Direct implementation of each filter would require too many multipliers and adders, and the shared filter implementation has a higher input to output latency.

4.7 DWT VHDL Generation

Once the architecture is completely designed and a full DWT processor is described in HDS, the final step is to generate VHDL and verify that the design simulates correctly at this level. SPW has the ability to automatically generate VHDL from an HDS design so the transformation amounts to telling the tool to generate the VHDL code. The VHDL simulates exactly the same as the HDS design.

Simulation times are significant to top-down design because verification requires running test cases against the floating-point outputs. The VHDL simulated for over twenty-four hours in a VHDL simulator whereas the HDS design simulated in less than three hours using the SPW simulator. Therefore, using the SPW tool greatly aided the top-down design process because the simulations at the architecture level could be completed relatively quickly, and the VHDL generation ensured that the next level would behave the same.

The output of the HDS and VHDL designs are simulated and compared against the floating-point output. The difference signal is still within the range of 10^{-7}, indicating near perfect reconstruction. The DWT architecture design exercise is successfully completed.

Chapter 5

Conclusions

5.1 Accomplishments

The goal of this work was to describe and verify a methodology for top-down design. Specifically, the portion of the design process which takes a specified algorithm to the hardware description level was examined. The steps involved in this process included defining the system requirements, performing floating-point algorithm analysis, creating and verifying a fixed-point model, designing an architecture, and finally, generating a VHDL model. Each design process step included iterative revision, tradeoffs, and simulations. The VHDL description should actually be a hardware description of the algorithm to be implemented, so that the bottom of the design process is readily completed from the VHDL.

The DWT design exercise was successful in both testing the top-down design process and also in creating an architecture which could be implemented in hardware. The algorithm, fixed-point model, and architecture were all analyzed and designed in SPW. The VHDL generated from SPW's HDS can be targeted to a commercial VHDL simulator and synthesis tool to verify the continuity of the design process and synthesize the VHDL into hardware.

5.2 Comments on the Top-Down Design Process

In any application there may be more or less time and focus devoted to different activities within the design process. The purpose of describing the design process in a general manner is to give guidelines for how to approach the complex task of designing a DSP system. The value of system requirements, algorithm analysis, fixed-point modeling, candidate architecture identification and design, and VHDL modeling lies in the user's ability to easily apply these steps to any of the tasks for design. The simulation plan is helpful in creating an executable specification for several of the design levels. The simulation plan can also reduce the amount of time spent updating designs and determining the impacts of changes made. The system requirements along with tradeoffs serve as the glue that ties all of the design steps together, because the decisions at every design level could not be made without tradeoffs and system requirements to base the tradeoffs upon. It is also important to note that choices made early on in top-down design do constrain lower level requirements.

The usefulness of a commercial tool for top-down design lies in the user's ability to make revisions, compare results, and perform tradeoffs in a single environment in an efficient, rapid manner. For many designers, the use of a single graphical interface to control block diagrams, simulations, and signal flows is very helpful for quickly seeing the results of different decisions made during the design process. In a top-down design process where the goal is to get to a VHDL description of the hardware, a tool with automatic VHDL generation can speed up the design process. However, the steps from a fixed-point model to an architectural implementation are not automatic, because there are no tools that can factor in all of the unit requirements, tradeoffs, and resource sharing to generate the architecture automatically. Thus, the hardware is still designed by the user. However, the tool requires the design to be revised only once at the architecture level and then generates the VHDL automatically. Tools with automatic VHDL generation capabilities therefore speed up the top-down design process because the VHDL is not rewritten once the architecture is designed, so there is less chance for error in the final implementation.

Commercial tools can be helpful to the top-down design process, but the user must pay careful attention to the tool's limitations or constructs. A side-effect of using a commercial tool for top-down design is that the models in their libraries may not be representing the same function or component that the user has intended. With SPW's HDS, certain common architecture components such as a register file or flip-flop may not behave exactly like the real components. Also, the user has to be aware of the tool's simulation techniques. The difference between a sample driven simulator and clock driven simulator can affect the way an algorithm or design works within the tool, and care has to be taken to work with the simulator. Then the issue of designing for implementation versus designing around the tool's shortcomings might become significant. Therefore, if much of what the tool models has to be redesigned later, or if the tool is not accurately representing the environment of the final implementation, then the tool is not furthering the top-down design process.

The design exercise of mapping the DWT from algorithm to VHDL is meant to illustrate the issues encountered in top-down design. The exercise is one instance of the design process, and with a different set of requirements the final result would have been different. To some extent, the application will specify where the emphasis of the process lies; the top-down process is only a model.

The implementation exercise of the DWT is fairly low level in an overall system design. However, the same process steps would apply to even higher and lower levels of abstraction, because the process is general enough to allow for different levels of system design. Also, the exercise was helpful for understanding what steps comprise the top-down design process because the actual process definition evolved as it was being applied to the DWT.

Appendix A

Biorthogonal Wavelet Filters

g[n]	h[n]	g'[n]	h'[n]
0.037828	-0.064539	-0.064539	-0.037828
-0.23849	0.040689	-0.040689	-0.23849
-0.110624	0.418092	0.418092	0.110624
0.377402	-0.788486	0.788486	0.377402
0.852699	0.418092	0.418092	-0.852699
0.377402	0.040689	-0.040689	0.377402
-0.110624	-0.064539	-0.064539	0.110624
-0.23849			-0.23849
0.037828			-0.037828

Table A-1: Wavelet filters from Cohen, Daubechies, and Feauveau, [12].

BIBLIOGRAPHY

[1] Lauwereins, R., Engels, M., Ade, M. Peperstraete, J. A. "Rapid Prototyping of Digital Signal Processing Systems with GRAPE-II," *DSP & Multimedia Technology,* (September 1994): 22-31.

[3] Strang, G. & Nguyen, T. *Wavelets and Filter Banks*, Wellesley, MA: Wellesley-Cambridge Press, 1996.

[4] Rioul, O. & Vetterli, M. "Wavelets and Signal Processing," *IEEE Signal Processing Magazine,* (October 1991): 14-38.

[5] Rioul, O. "A Discrete-Time Multiresolution Theory," *IEEE Transactions on Signal Processing,* (August 1993): 2591-2606.

[6] Vetterli, M. & Herley C. "Wavelets and Filter Banks: Theory and Design," *IEEE Transactions on Signal Processing,* (September 1992): 2207-2232.

[7] Smith, M. J. T. & Barnwell, T. P. III, "Exact Reconstruction Techniques for Tree-Structured Subband Coders," *IEEE Transactions on Acoustics, Speech, and Signal Processing,* (June 1986): 434-441.

[8] Villasenor, J. D., Velzer, B. & Liao, J. "Wavelet Filter Evaluation for Image Compression," *IEEE Transactions on Image Processing,* (August 1995): 1053-1060.

[9] Hlawatsch, F. & Boudreaux-Bartels, G. F. "Linear and Quadratic Time-Frequency Signal Representations," *IEEE Signal Processing Magazine,* (April 1992): 21-67.

[10] Oppenheim, A. V. & Schafer, R. W. *Discrete-Time Signal Processing*, New Jersey: Prentice Hall, 1989.

[11] Vaidyanathan, P. P., *Multirate Systems and Filter Banks*, New Jersey: Prentice Hall, 1993.

[12] Cohen, A., Daubechies, I., and Feauveau, J. C., "Biorthogonal Bases of Compactly Supported Wavelets," *Communications on Pure and Applied Mathematics*, (1992): Vol. XLV, 485-560.

[13] G. Knowles, "VLSI Architecture for the Discrete Wavelet Transform," *Electronics Letters*, July 1990, Vol. 26, 1184-1185.

[14] O. Rioul and P. Duhamel, "Fast Algorithms for Discrete and Continuous Wavelet Transforms," *IEEE Transactions on Information Theory*, Mar. 1992, Vol.38, No.2, 569-586.

[15] K. Parhi and T. Nishitani, “VLSI Architectures for Discrete Wavelet Transforms,” *IEEE Transactions on VLSI Systems*, June 1993, Vol. 1, No. 2, 191-202.

[16] M. Vishwanath, “The Recursive Pyramid Algorithm for the Discrete Wavelet Transform,” *IEEE Transactions on Signal Processing*, Mar. 1994, Vol. 42, No. 3, 673-676.

[17] C. Chakrabarti and M. Vishwanath, “Efficient Realizations of the Discrete and Continuous Wavelet Transforms: From Single Chip Implementations to Mappings on SIMD Array Computers,” *IEEE Transactions on Signal Processing*, Mar. 1995, Vol. 43, No. 3, 759-771.

[18] M. Vishwanath, R. M. Owens, and M. J. Irwin, “VLSI Architectures for the Discrete Wavelet Transform,” *IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing*, May 1995, Vol. 42, No. 5, 305-316.

[19] Aware, Inc., Wavelet Transform Processor Chip User’s Guide, 1994.