THESIS REPORT

Master's Degree

Full Custom VLSI Implementation
of Time-Recursive 2-D DCT/IDCT Chip

by V. Srinivasan
Advisor: K.J.R. Liu

M.S. 93-23

Sponsored by
the National Science Foundation
Engineering Research Center Program,
the University of Maryland,
Harvard University,
and Industry

Abstract

Title of Thesis: Full Custom VLSI Implementation
of Time-Recursive 2-D DCT/IDCT Chip

Name of degree candidate: Vishnu Srinivasan

Degree and year: Master of Science, 1993

Thesis directed by: Professor K. J. Ray Liu
Department of Electrical Engineering

Discrete Cosine Transform (DCT) based compression techniques play an important role in today's digital applications such as high definition television (HDTV) and teleconferencing which require high speed transmission of digital video signals. In this thesis, a high-performance VLSI implementation of a DSP chip which computes the two-dimensional discrete cosine transform and its inverse (2-D DCT/IDCT) is presented. The chip is based on the fully-pipelined time recursive IIR structure and employs a highly modular and hierarchical design strategy. Architectural model simulations are performed for determining system parameters required to achieve a high-speed and high-performance implementation. Based on these simulations, ROM and internal bus precision are chosen to ensure a minimum PSNR of 40 dB which is required for most digital imaging applications. High speed design is obtained by using distributed arithmetic to achieve fast multiplication through table lookups. A two-phase nonoverlapping clock is employed to perform computations in both phases, resulting in twice the throughput. Various submodules like ROM lookup tables,

adders, half-latches, delay-units and multiplexors are implemented. Timing simulations of critical path modules indicate a clock frequency of 50 MHz corresponding to a data rate of 400 Mb/s. The chip dimensions are $24550\lambda \times 27094\lambda$ and its area is 240 mm^2. The chip has been submitted for fabrication in 1.2μ CMOS N-well double-metal single-poly technology.

Full Custom VLSI Implementation of Time-Recursive 2-D DCT/IDCT Chip

by

Vishnu Srinivasan

Thesis submitted to the Faculty of the Graduate School
of The University of Maryland in partial fulfillment
of the requirements for the degree of
Master of Science
1993

Advisory Committee:

Professor K. J. Ray Liu, Chairman/Advisor
Professor Kazuo Nakajima
Professor Linda Milor

Dedication

To my parents

Gomathi and Srinivasan.

Acknowledgements

I would like to begin by thanking my advisor Dr. K. J. Ray Liu for his guidance, support and sustained encouragement throughout my graduate study. I also wish to express my gratitude to Dr. Linda Milor with whom I had many a valuable discussion during the course of this project. I am grateful to Dr. Kazuo Nakajima for his insightful comments and suggestions for improving the thesis.

I learned many of my skills by interacting with several people at The University of Maryland. In particular, I would like to thank Ching-Te Chiu who guided me during the initial stages of this project. Special thanks go to Ravi Kolagotla who helped me get started on practical VLSI implementations and patiently answered all of my hardware and software questions.

I would like to thank the Institute for Systems Research and the Electrical Engineering Department for providing the excellent resources and a stimulating research environment. I also wish to thank the computer staff of ISR, in particular, Haisong Cai and Amar Vadlamudi, who helped me with the demanding computer needs of this project.

This research was supported in part by the Maryland Industrial Partnership MIPS/Micro Star grant and the National Science Foundation NSF ECD-8803012 grant.

Several students, both past and present, have contributed directly or indirectly to this thesis. I would like to thank An-Yeu Wu, Ut-Va Koc, Shyam Kuttikkad, Shu-Sun Yu, K. G. Satish, Hong-Yi Wang, Ying-Min Huang, Sridhar Krishnamurthy, Rajiv Madabhushi, Balu Rudra, Hung-Jen Lin, R. Idi, and V. Pammal, who were always available to discuss and clarify ideas on any topic,

even technical ones.

A special acknowledgment goes to my parents, who have made this possible with their unlimited confidence, support, and encouragement. I dedicate this thesis to them.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Recent advances in various aspects of digital technology have brought about many applications of digital imaging. These applications require vasts amounts of data to represent the digital images, resulting in high storage and transmission costs. Modern compression techniques play an important role in today's digital applications like high definition television (HDTV) which require high speed transmission of digital video signals [2, 3]. These techniques yield typical compression ratios ranging from 10–50, without introducing noticeable artifacts in the coded image [4].

The plethora of applications employing digital images have necessitated standardization efforts which have evolved into JPEG, MPEG, and MPEG-II digital image compression standards. The Joint Photographic Experts Group (JPEG) has developed the compression standard for still images, and the Moving Pictures Experts Group (MPEG) and MPEG-II have developed the standards for video (or motion pictures). After a rigorous selection process, it was found that Discrete Cosine Transform (DCT) based codecs give the best performance, with respect to compressibility, picture quality and implementation complexity. In

fact, several DCT based codecs were developed in the industry even while these standards were in the process of being approved. And today, the DCT has become the industry standard for image and video coding applications [5, 6].

1.1 DCT/IDCT Architectures

There has been considerable research in efficient mapping of DCT algorithms to practical and feasable VLSI implementations in the recent past [7, 8, 9]. Traditionally the architectures of the 2-D DCT chips developed in the industry are based on the row-column (separability) decomposition property [6, 10]. These are indirect methods. Here we take a look at the time-recursive DCT architectures developed by Liu and Chiu [1, 11]. These time-recursive lattice and IIR structures are ideally suited for handling serial data, as is often the case in digital video applications. These architectures lend themselves to a highly parallel, modular, regular and fully pipelined 2-D DCT/IDCT structures [1]. The lattice and IIR algorithms are direct 2D methods which do not require transposition, unlike the traditional row-column algorithms. Also, the architecture is very suitable for real-time applications as the time-recursive concept has been exploited to eliminate the waiting time for data to arrive. From the VLSI implementation point of view also, as these parallel lattice/IIR structures are decoupled into independent modules, the need for global communication is eliminated. The emphasis of this thesis is to present a novel VLSI implementation of a chip which computes the 2-D DCT and its inverse, based on one of the above architectures.

1.1.1 Lattice vs IIR

To identify the architecture (lattice or IIR) most appropriate for our high-performance VLSI implementation, extensive simulations of architectural models are performed. The primary objective is to satisfy the minimum SNR (peak or average) criterion requirement of 40 dB [7] which is required for most applications. Simulation studies suggested the IIR structure which, while satisfying the minimum SNR requirements, has a more compact layout than the lattice structure. The lattice structure, however, is inherently more accurate. Besides the area and accuracy criteria, the structure that will be implemented should also be able to compute the inverse DCT without requiring too much of additional hardware or area. Even in this aspect, the IIR architecture turns out to be superior. Once it was decided that IIR structure would be used for implementing the DCT/IDCT chip, further simulations are performed to determine various design parameters like internal bus precision, clocking scheme, and choice and design of various submodules required for a fast and efficient VLSI realization.

1.2 Chip Implementation

The DSP chip which computes the two-dimensional 8×8 block discrete cosine transform and its inverse (8×8 2-D DCT/IDCT) is implemented. A unique aspect of this implementation is the use of ROM look-up tables for effective multiplication [12, 13]. This is also known as the distributed arithmetic scheme. Some advantages of ROMs over multipliers are its high speed, good accuracy, and smaller area requirement for the same input word length (or internal bus precision). By employing an interdigitated structure for the ROMs, and taking

advantage of the IIR structure, the inverse 2-D DCT is also computed, with only a marginal increase in chip area requirements. Split phase clocking is used to achieve a much faster (almost double) throughput of 400 Mbits/sec, while operating at a clock frequency of 50 MHz. Magic has been used to implement the highly hierarchical and compact 2-D DCT/IDCT chip design. By employing a modular approach, we have saved considerably on the design time, and at the same time, also increased the reliability of the final chip implementation. IRSIM, a switch based logic-level simulator, is used to verify the functional specifications of the forward and inverse DCT. Crystal and Spice are used to perform detailed timing analysis and hence to estimate the clock speed. A prototype chip is being fabricated in 1.2μ CMOS N-well double metal single-poly technology. The chip measures $24550\lambda \times 27094\lambda$. Its implementation in 1.2μ technology results in a die-size of 240 mm^2.

1.3 Thesis Organization

The rest of the thesis is structured as follows. Chapter 2 describes the time recursive structures, IIR and lattice, which have been considered for implementation. Much of Chapter 2 has been paraphrased from [1] and [11]. For further information on these, please refer to the above mentioned papers. The description of the chip implementation is in a top-down fashion. In Chapter 3, we take a look at all the factors that influence the choice of architecture for the DCT chip. Based on such considerations, extensive simulations are performed to determine the most suitable structure for implementation. These simulations which are presented in Chapter 5, are also summarized in Chapter 3, to main-

tain continuity of the top-down description. The floor-plan of the final VLSI 2-D DCT/IDCT chip, along with a test-module chip which was also designed, are presented towards the end of this chapter. The various submodules like carry lookahead adders, compiled ROMs, dynamic half latches and multiplexers are documented in Chapter 4. This chapter also describes the design of the 1-D and 2-D modules which are assembled together with the CSA module for the computation of the 2-D DCT/IDCT. Chapter 5 presents the various simulation results. Conclusions are outlined in Chapter 6.

Chapter 2

Algorithm and Architecture

The Discrete Cosine Transform (DCT) is widely used for the compression of digital image and video signals. In any image, there is a high degree of correlation between adjacent pixel's intensity values. By striping away the redundant information, compression will be achieved. When the DCT of such a signal is taken, most of the energy is concentrated in the low frequency components. A simple way of visualizing the manner in which image compression is achieved is as follows. Throw away the high frequency components, and transmit only the transformed low frequency components, as most of the signal energy resides in these components. At the receiving end, compute the inverse transform and reconstruct the image. In the inverse transform, use zero for the high frequency components which are not transmitted. This simple technique yields good compression of the input image, without much compromise on the picture quality.

Actual systems which implement the JPEG or MPEG compression standards are understandably a lot more complicated. Besides the DCT, they have several other sub-systems like quantizers and entropy encoders which give additional compression in JPEG based codecs. MPEG codecs would also include motion

estimation modules. At the receiving end, we would have the corresponding modules like inverse DCT, dequantizers and entropy decoders. JPEG and MPEG do not spell out all details of the codec implementation. The user has the flexibility to choose, for example, which entropy coding scheme to use—Huffman coding or arithmetic coding [4]. The common denominator in all these is the computation of the DCT and its inverse. This, and the DCT's good energy compaction properties and near-optimal (least mean square error) performance with respect to the Karhunen-Loeve Transform have prompted extensive research in DCT algorithms and their efficient mapping to high-performance VLSI systems.

2.1 DCT Algorithm

In this section, we take a look at the various definitions—one dimensional DCT and its inverse, and the two dimensional form too.

2.1.1 1-D DCT and its Inverse

There are several ways of defining the DCT. Here, the DCT definition is chosen in such a manner that additional normalization will not be required when the IDCT is computed [5]. This is also known as the normalized DCT/IDCT form.

Forward 1-D DCT

The one-dimensional (1-D) DCT of a series of input data with N elements, starting from $x(t)$ and ending with $x(t+N-1)$ is defined as:

$$X_c(k,t) = C(k)\sqrt{\frac{2}{N}}\sum_{n=t}^{t+N-1} x(n)\cos\left[(n-t+\frac{1}{2})\frac{\pi k}{N}\right], \quad k = 0,1,\ldots,N-1, \tag{2.1}$$

where

$$C(k) = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } k = 0, \\ 1 & \text{otherwise.} \end{cases}$$

The time index t in $X_c(k,t)$ denotes that the transform starts from $x(t)$.

Inverse 1-D DCT

The inverse DCT is defined in a similar manner.

$$x_c(n,t) = \sqrt{\frac{2}{N}} \sum_{k=t}^{t+N-1} C(k-t) X_c(k) \cos\left[(n+\frac{1}{2})\frac{(k-t)\pi}{N}\right], \quad n = 0, 1, \ldots, N-1, \tag{2.2}$$

where $C(k)$ is as defined in equation (2.1).

2.1.2 2-D DCT/IDCT

The two dimensional DCT is just an extension of the 1D DCT. From the image compression point of view, the 2-D DCT is naturally more relevant. We will see in the subsequent sections, how the forward and inverse transforms are computed.

Forward 2-D DCT

The $N \times N$ 2-D DCT of a $N \times N$ 2-D image sequence $\{x(m,n) : m = t, t+1, \ldots, t+N-1; n = 0, 1, \ldots, N-1\}$ is defined as:

$$X_c(k,l,t) = \frac{2}{N} C(k) C(l) \sum_{m=t}^{t+N-1} \sum_{n=0}^{N-1} x(m,n) \cos\frac{\pi[2(m-t)+1]k}{2N} \cos\frac{\pi(2n+1)l}{2N} \tag{2.3}$$

where $C(k)$ is as defined in equation (2.1).

Inverse 2-D DCT

The $N \times N$ 2-D IDCT for inputs from the transform domain are given as:

$$x_c(m,n) = \frac{2}{N} \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} X_c(k,l)C(k)C(l) \cos[\frac{\pi[2k+1]m}{2N}] \cos[\frac{\pi(2l+1)n}{2N}] \quad (2.4)$$

where $C(k)$ is again, as defined in equation (2.1).

2.2 DCT Architecture

In this section the time-recursive DCT algorithms developed by Liu and Chiu [1, 11] are presented. The lattice and IIR structures for the computation of the 2-D DCT is described briefly. However, for a detailed description please refer to the original papers by Liu and Chiu [1, 11]. Also considered later on in this chapter, are some of the issues that are instrumental in determining which structure is more suitable for actual VLSI implementation.

2.2.1 Lattice Structure

The 2-D DCT can be generated by using the lattice structure. Figure 2.1 shows the basic kernel. Depending on the coefficients stored in the multiplier, the appropriate transform coefficient, $X_c(k)$ is computed. Every clock unit, a new data input enters and gets processed. At the end of N cycles, the computations for that set of N data elements are completed. By employing N such structures, all the transform coefficients are computed in parallel and made available at the end of N cycles. This parallel computation of N coefficients represents the 1-D DCT. See Section 2.3 and Fig. 2.3 for details on how the 2-D DCT is computed.

Each kernel requires six multipliers for the computation of the forward transform alone. Clearly, there are too many multipliers for efficient VLSI implemen-

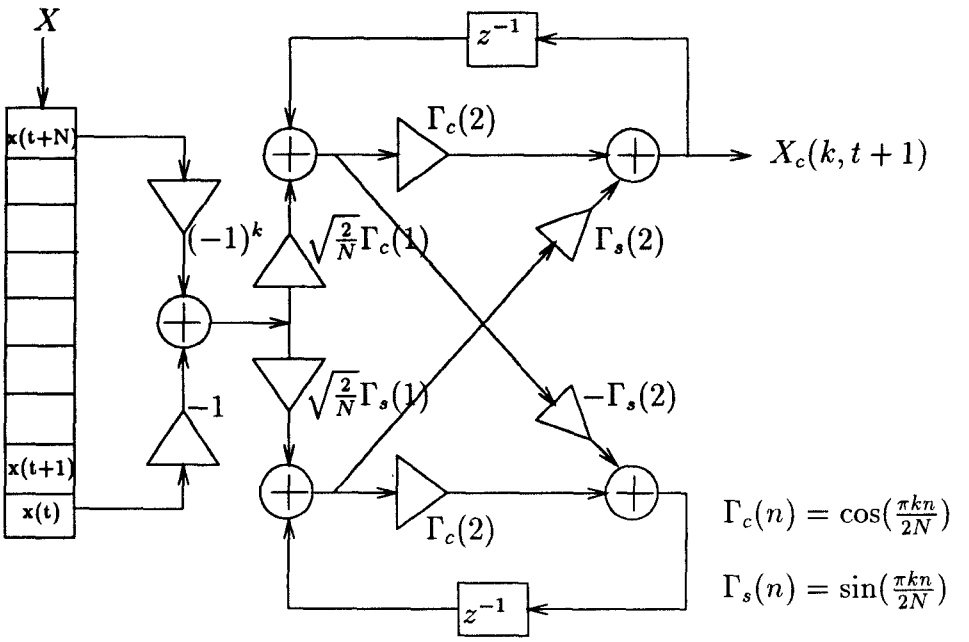


Figure 2.1: Lattice Structure for DCT computation [1]

tation. We should device some way of reducing the multiplier count. However, an advantage of using this structure is its better numerical properties, being a normal-form implementation as opposed to the IIR structure.

2.2.2 IIR Structure

The IIR structure is derived by considering the transform operation to be a linear shift invariant (LSI) system which transforms the serial input data into their transform coefficients [14]. It is viewed as a straightforward digital filter, for which the transfer function is derived. A kernel of this structure is illustrated in Fig 2.2. The second order IIR filter's transfer function which transforms the

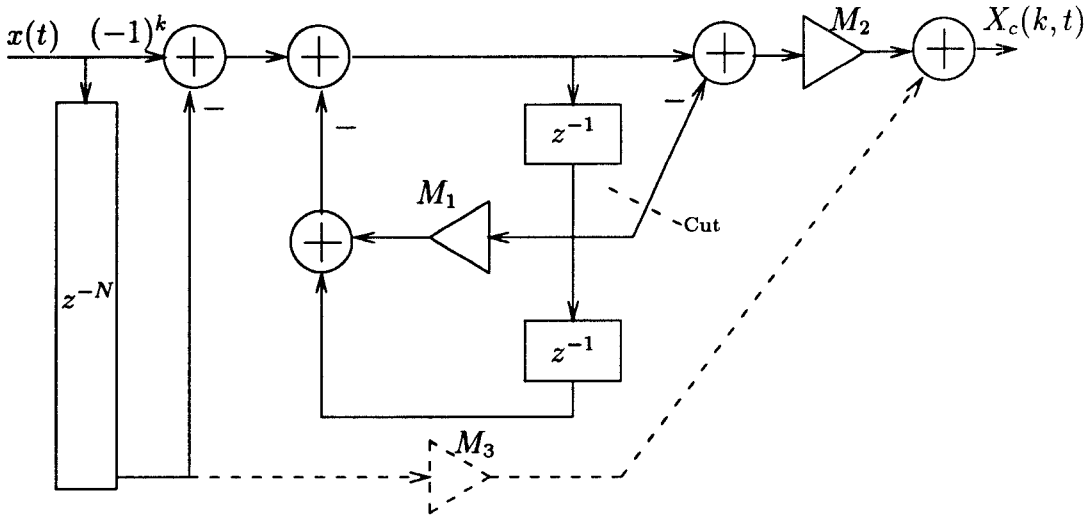


Figure 2.2: IIR Structure for DCT/IDCT computation

input data sequence into their DCT coefficients is:

$$H_c(z) = \sqrt{\frac{2}{N}} C(k)((-1)^k - z^{-N}) \cos\left(\frac{\pi k}{2N}\right) \frac{(1 - z^{-1})}{1 - 2\cos\left(\frac{\pi k}{N}\right) z^{-1} + z^{-2}} \tag{2.5}$$

The signal flow graph for equation (2.5) can be implemented as the direct form II and is shown in the figure. For the forward transform, the dashed portions of the figure are omitted.

The transfer function of the inverse transform is also derived from IDCT equation in a straightforward manner:

$$H_{ic}(z) = \frac{\sqrt{\frac{2}{N}} \cos\frac{(2n+1)(N-1)\pi}{2N}}{1 - 2\cos\left(\frac{(2n+1)\pi}{2N}\right) z^{-1} + z^{-2}} + \sqrt{\frac{2}{N}}\left(\frac{1}{\sqrt{2}} - 1\right) z^{-(N-1)} \tag{2.6}$$

Equation (2.6) for the inverse DCT computation is quite similar to equation (2.5) but for the additional delay term. By including the dashed portions as shown in Fig. 2.2, the inverse can be computed using much of the same hardware necessary to compute the forward transform. It is important to note that the derivation for the inverse transform implicitly assumes a block of size N, and that all registers

are cleared at the end of a block. Without this, the transfer function for the inverse becomes quite unmanageable, and it would not be possible to implement the forward and inverse transform together. Another important difference to note is that the multiplier coefficients are different in the forward and inverse transforms. If they are to be computed with little additional hardware requirements, then some scheme to implement a variable coefficient multiplier should be devised.

2.3 2-D DCT Architecture

So far, we have seen the kernel of IIR and lattice structure which generate the transform (or the inverse) for a single coefficient. In this section, we look at how the entire 2-D DCT is computed. As will be mentioned in Section 3.1, the image is divided into blocks of size $N \times N$ and the 2-D DCT is computed for each of the blocks individually.

The block 2-D DCT architecture is shown in Fig 2.3. The lattice/IIR modules in the second dimension of the 2-D DCT block architecture are similar to the modules in the first dimension. The serial input data is fed into the first dimension's lattice modules, which at the end of N cycles produce the N DCT coefficient in parallel. These data are latched into the circular shift array, and are fed to the second dimension serially as the next row is processed in the first dimension. At the end of $N^2 + N$ cycles, the 2-D DCT is available at the output of the second dimension.

This is a fully pipelined serial-in parallel-out (SIPO) system, with a throughput rate of N^2 cycles (or N cycles, if it is frame-recursive). In terms of hardware

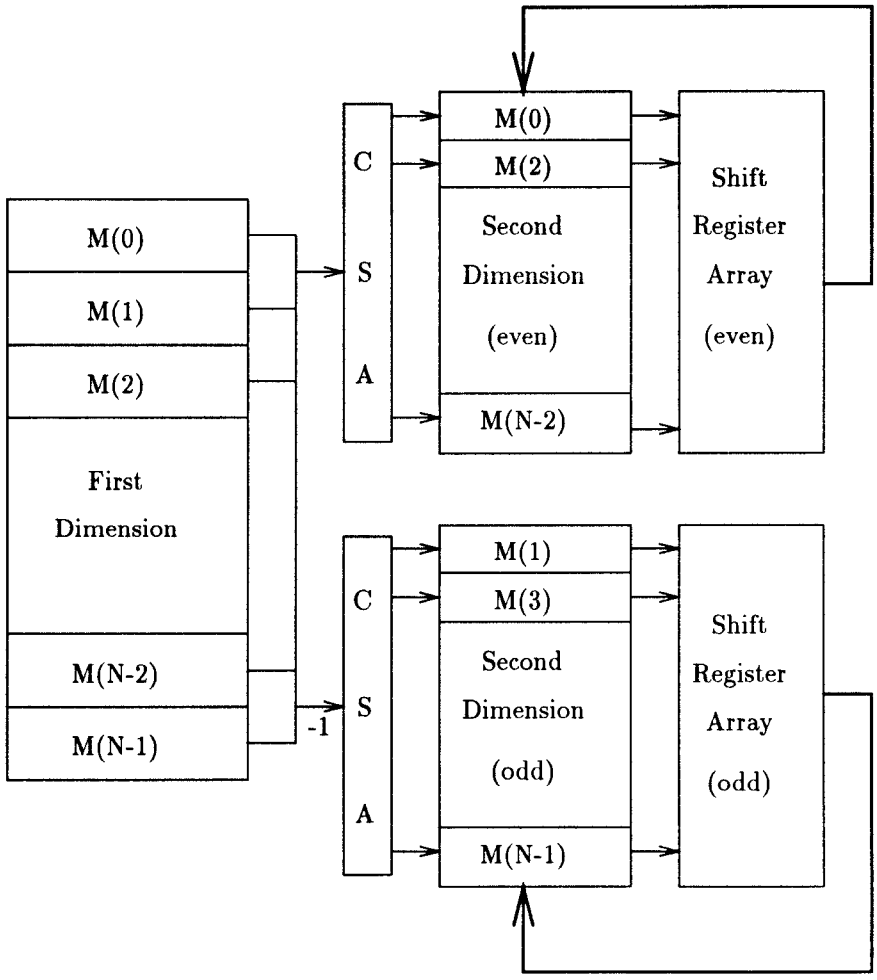


Figure 2.3: Block 2-D DCT Architecture

complexity, it requires only two 1-D DCT architectures. The primary advantages of this architecture over other contemporary systems are not only its efficiency in terms of system throughput, but also its hardware simplicity and regularity. Its modular structure proves to be quite amenable to VLSI implementation.

2.4 VLSI Implementation Considerations

The two-dimensional DCT algorithm developed by Liu and Chiu [1, 11] gives a broad outline and foundation for the actual VLSI implementation. There still are however, several issues and concerns which have to be decided before the DCT/IDCT chip can be implemented. Foremost of these concerns are the accuracy studies, which are conducted to ensure that the round-off errors do not cause too much noise in the signal output. Most digital image applications require a minimum peak SNR (PSNR) of $40dB$ [7]. There are other equally important implementation issues, such as how to make the choice between using distributed arithmetic schemes as opposed to multipliers. These have important repurcusions both from chip area and delay points of view. The third, and perhaps the most important concern for the architecture selection is its ability to compute the IDCT as well, with the least overhead in terms of hardware or area requirement.

2.5 Summary

In this chapter we have taken a look at the basics; the Discrete Cosine Transform, the time recursive structures that can be used to implement it, and some of the VLSI implementation considerations. In the following chapter we will see more

of these issues addressed. In Chapter 3 the VLSI chip that implements the two-dimensional DCT and its inverse is presented.

Chapter 3

DCT/IDCT Chip Overview

The VLSI implementation of the two-dimensional Discrete Cosine Transform (DCT) and its inverse (IDCT) is presented in this chapter. The 2-D DCT/IDCT chip employs a highly hierarchical and modular design. The actual design proceeded in a bottom-up manner, but for clarity of presentation, we have used the top-down method of description. In this chapter the floor plan of the two-dimensional DCT/IDCT chip and its main submodules are presented. A one-dimensional DCT/IDCT chip which implements a single channel (to compute DCT/IDCT coefficient 5) was designed as our test module. This chip is also briefly described here. The simulation criteria and other requirements which helped us decide the final structure (IIR or lattice) are also described, along with various other implementation aspects.

3.1 Input Format

A two-dimensional image, depending upon the application, will have a certain number of pixels in every row and columns. For the sake of simplicity, take

an image whose pixels are encoded in gray-scales as an example. Each pixel typically would have 256 gray levels represented by an 8-bit word. Now, the image size varies from one application to another, and moreover, computing the 2-D DCT for the entire image becomes prohibitively expensive. So, typically the image is divided into blocks of $N \times N$ pixels, and the 2-D DCT is computed for each of the blocks (see Fig 3.1). The source image data is fed in serially starting

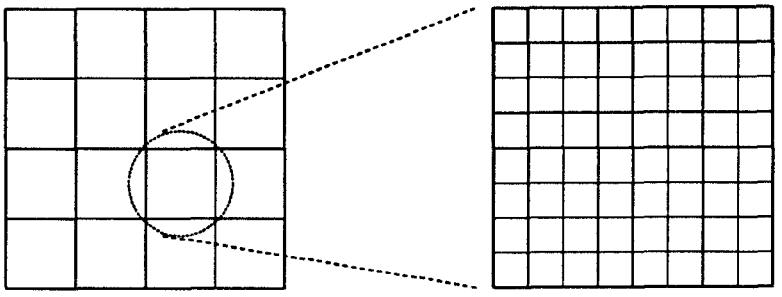


Figure 3.1: Source Image Format

from the top left pixel to the top right pixel, and then on to the next row, and so on for every block, until the entire image is coded.

3.2 Performance Criteria

Of the several criteria which help in determining the various chip parameters to be used, perhaps the most important one is the accuracy criterion. An input pixel represented as a fixed point finite precision number undergoes truncation at various stages of the DCT/IDCT computation. This introduces noise depending on the amount of truncation. We have considered the Peak Signal to Noise Ratio (PSNR) as employed in [7] and the Average SNR for estimation of the truncation noise in the DCT/IDCT architecture. Some of the other important criteria are

the chip area and speed. It is very important to implement a tight design as the fabrication cost for huge area-layout is prohibitively expensive. The applications of the DCT/IDCT chip are in real-time image compression systems like that of HDTV. To be able to achieve the required high throughput, a careful design to optimize the speed is required.

3.3 Architectural Aspects

There are several issues to be considered here. We take them one at a time and consider their pros-and-cons. Most of these implementation choices were made based on extensive simulations, which are outlined in Chapter 5. The details of various modules like half latches, adders, ROMs, multiplexers are to be found in the next chapter. We now highlight some of the major design choices made for the implementation.

3.3.1 Lattice vs IIR

As mentioned before, the two structures considered for this implementation are the lattice and IIR. Some features of these structures that we focus on are numerical accuracy, area, and ease of inverse DCT computation along with the forward DCT.

Numerical Properties

The lattice structure is inherently more accurate than the IIR structure which requires a much larger dynamic range for internal computations. This means that, to achieve the same accuracy as the lattice structure, the IIR structure

would have to allocate more bits. The internal bus precision would consequently be higher, resulting in more area.

Area

The largest module in the design is the multiplier. And the number of multipliers will determine to a large extent which structure will be more compact. At the first glance, we see that the IIR structure has only two multipliers as opposed to six for the lattice structure. So, even if we need to have a larger internal bus for the IIR (to have lower truncation noise), the area criterion might still favor the IIR structure, due to its fewer number of multipliers.

Combining forward/inverse structures

Even in this aspect, the IIR wins out. With just one extra multiplier for the first (and second) dimension, and additional multiplexers at appropriate signal paths in the channel-module, the inverse can also be computed. However, the multipler coefficients, M_1 and M_2 (See Fig. 2.2) are different in the forward and inverse transforms computation. This became a critical issue in the choice and design of the multiplier will be finally used.

3.3.2 Distributed Arithmetic vs Multiplier

There were primarily three aspects to be considered to decide whether to use a multiplier or a ROM lookup table.

Speed

ROM lookup tables were much better than multipliers. Preliminary simulation studies indicated that ROMs were about four times as fast as the best multiplier that we could find. Clearly, ROMs had an edge over multipliers for our fast DCT/IDCT chip implementation.

Area

Area-wise ROMs can become very huge, depending upon the number of lookup entries determined by the input precision. Aspects relating to this issue are discussed in Chapter 4 where the ROM design is described, and in Chapter 5 where simulations to determine the suitability of ROM over multipliers are discussed [15].

Forward/inverse transforms

As we have seen before, different multiplier coefficients are needed for the forward and inverse transform computation. With multipliers, it is fairly straightforward to have a small ROM encoded with different coefficients feeding into one of the multiplier inputs. With ROMs, it turns out to be a little more complex, but is still do-able (See Chapter 4). This was critical, as without this, it would not have been possible to use the ROMs in our implementation, even though they are much faster than multipliers.

3.3.3 Clocking Scheme

To keep our various modules simple, dynamic latches are used. Also, by the use of a split-phase clocking, we have achieved doubling of the clock speed which

would not have been possible otherwise [16].

3.4 Chip Architecture

In the previous section, we have seen the various architectural choices for the VLSI implementation. Based on architectural-level simulations to verify SNR criterion, preliminary timing analysis of various critical path modules and the area estimates, the component details were decided. The optimal VLSI implementation of the high-performance, high-speed chip for the computation of the two-dimensional DCT and its inverse are:

- Use of IIR structure as the basic kernel, with minimal hardware complexity, for both the forward and inverse transforms.
- Interdigitated ROM lookup tables for high-speed multiplication and reduced chip area.
- Non-overlapping split-phase clocking scheme for doubled throughput and higher clock speed.

The input format for the forward transform is an unsigned 8-bit word, representing 256 gray-levels for any individual pixel. Two's complement, fixed point numbers are used for representing the internal computations. The bus precision is 16 bits, with the binary point implicitly fixed by the calculation in progress. It needs to be so wide to accommodate the large dynamic range of the IIR structure. The inverse DCT is typically represented by 12 bits in 2's complement format. Here, the IDCT output coefficients are 16-bit words, again, in

2's complement form. For the inverse DCT computation, due to the hardware limitations, only the 8 MSB bits are used, resulting in a slight loss of accuracy.

3.5 Implementation Aspects

VLSI ASIC chips are implemented by full-custom or semi-custom designs. In full-custom designs, polygon editors are used to layout all the various layers that define not only the MOS transistors, but also the various metal layers including the interconnects. Full-custom design is usually employed in those situations where area and speed requirements are critical. By careful design, one can achieve a much tighter layout and higher speed of operation than can be achieved by the semi-custom approach. Semi-custom designs on the other hand are a lot quicker to implement. In this approach, basic cells are made available in standard cell libraries. Based on the circuit schematic, the various cells are placed and wires routed between them, using sophisticated placement and routing algorithms. This leads to very quick chip implementations, but usually the chip area and speed are much more than what can be achieved with a careful full-custom implementation. A distinct advantage that a semi-custom design has over the full-custom approach is higher reliability, which can also be thought of as the full-custom design requires more time to debug.

3.5.1 Our Approach

We have used a mixed approach for our design of the DSP ASIC. All the critical modules like ROMs, adders, multiplexers and latches are designed using full-custom techniques. As these modules are repeatedly used in all channels of both

dimensions of the DCT chip, the savings in area becomes considerable. Besides the saving in area, another important advantage of reusing these basic modules is the gain in reliability. By designing a module once, and then exhaustively testing it before using it repeatedly brings us the reliability that we usually associate only with semi-custom (and standard cell library) approach. Here, we have, effectively built our own library of cells, albeit consisting of much larger individual cells like the ROM and implementing more complex functions than are usually found in standard cell libraries.

In the design of the chip, as will be illustrated later on, the idea of using the submodules as just another library cell is carried out a step further. As the various blocks which implement the computation of the DCT coefficients are designed, they are further modularized and arrayed to yield an entire dimension. This extremely hierarchical design is what gives this VLSI implementation a high reliability.

3.5.2 Design Tools

The 2-D DCT/IDCT chip has been entirely designed with Magic. Magic is an interactive VLSI circuit layout editor developed in Berkeley [17]. It is based on the Mead-Conway style of design, which permits simplified design rules and circuit structures. An advantage of using Magic is that it is more than just a regular layout editor. It is able to use information about the circuit being designed and provides additional functions. The most useful of which is the on-line incremental Design Rule Checker (DRC) which flags DRC violations, as the circuit is being laid out. This saves considerable design effort. Magic has a built-in hierarchical circuit extractor, which provides for logical verification and

timing analysis after proper file format conversions and by using appropriate tools. Some of the other design aids that Magic provides are 'plows' and routing tools.

3.5.3 Simulation Tools

After the design is completed using Magic, the circuit parameters are extracted into a 'ext' format. This format preserves all information about the module. By using appropriate conversion utilities like 'ext2sim' or 'ext2spice', the extracted file is massaged into a format suitable for either logic simulations or for detailed timing analysis.

Crystal and Spice

Crystal takes on the timing analysis front. It is a semi-interactive program for analyzing the timing characteristics of large circuits. Crystal is used to estimate the critical path in a module between a specified set of input and output vectors. Worst-case times and hence propagation delays are estimated for a given change in the input vector. As the effect of this change is propagated to the output vectors, critical paths are flagged. This provides a starting point for conducting a detailed analysis using Spice. This tool usually ends up being a good test confirming ones' intuition, regarding the critical paths. Crystal, I have noticed, usually overestimates all delays, and Spice simulations have to performed to obtain accurate results.

Spice is the obvious tool to perform any detailed circuit analysis. We have used Spice primarily to perform timing simulations of the various modules that have been designed. These simulations are naturally performed, only for small

critical path modules such as adders or ROMs or multiplexers, and never at the system level. Preliminary timing simulations of all our modules was performed for the 2.0μ technology. Certain bug fixes (MOS) had to be implemented to 'spice3e2' before simulations could be performed successfully using 1.2μ technology parameters.

IRSIM

So far, we have talked about the timing simulations which is performed mostly on the smaller modules. The logic level simulation, which tells us if the designed circuit performs its given task, is as important as timing analysis. While the timing analysis tells us how fast a chip may be expected to function, the logic-level simulations tell us if the design will work at all, in the first place, regardless, of what speed it operates at!

Logic level simulations are performed at a much higher level of circuit abstraction, treating the transistors as switches which are either ON or OFF. IRSIM is an event-driven logic-level simulator for MOS circuits. We have used IRSIM to perform functionality verification of all modules. It is important that logic-level simulations are performed at all hierarchy levels—starting from the bottommost, like that of the adder or ROM, to intermediate levels such as 1-D/2-D modules, to the topmost level, namely the entire 2-D DCT/IDCT chip.

3.6 Test Chip: Channel 5

As a preliminary design, this test-module chip computes the one-dimensional DCT/IDCT coefficient for channel 5. This design is along the lines of Section 3.4. Recapitulating, the IIR architecture is used for computation of both the forward

and inverse transform. Interdigitated ROM lookup tables provide multiplication with a choice of two coefficients. And lastly, the split-phase clocking scheme is used to achieve almost twice the throughput as opposed to a regular clocking scheme.

Function	Compute 1-D DCT/IDCT coefficient 5
Mode Selection	'MODE' = 1 for Forward DCT
Clock Pins	Phi 1: ϕ_1 & Phi 2: ϕ_2
Reset Pin	Active high 'Reset' pin
Input Pins	Inp0, Inp1,...,Inp8
Output Pins	Out0, Out1,...,Out11
Power Rails	'Vdd!' & 'GND!'

Table 3.1: 1-D DCT/IDCT Chip Description

This chip was to be our test-implementation, and was designed to be accommodated in a MOSIS standard frame of die-size $6900\lambda \times 6800\lambda$, packaged in 40 pin DIP. All timing simulations have been performed using the 2.0μ technology file supplied by MOSIS for use of their 2.0μ double metal, single poly, N well CMOS process. Use of the MOSIS standard frame imposes restrictions on various dimensions like pad-to-pad, pad-to-corner, internal circuit-to-pad, and pad-to-edge distances. These restrictions have been taken into consideration in the final assembly. Table 3.1 describes the various control and I/O pins of this chip.

The chip statistics are tabulated in Table 3.2. The timing analysis is performed using MOSIS supplied 2.0μ technology Spice parameter files. It was

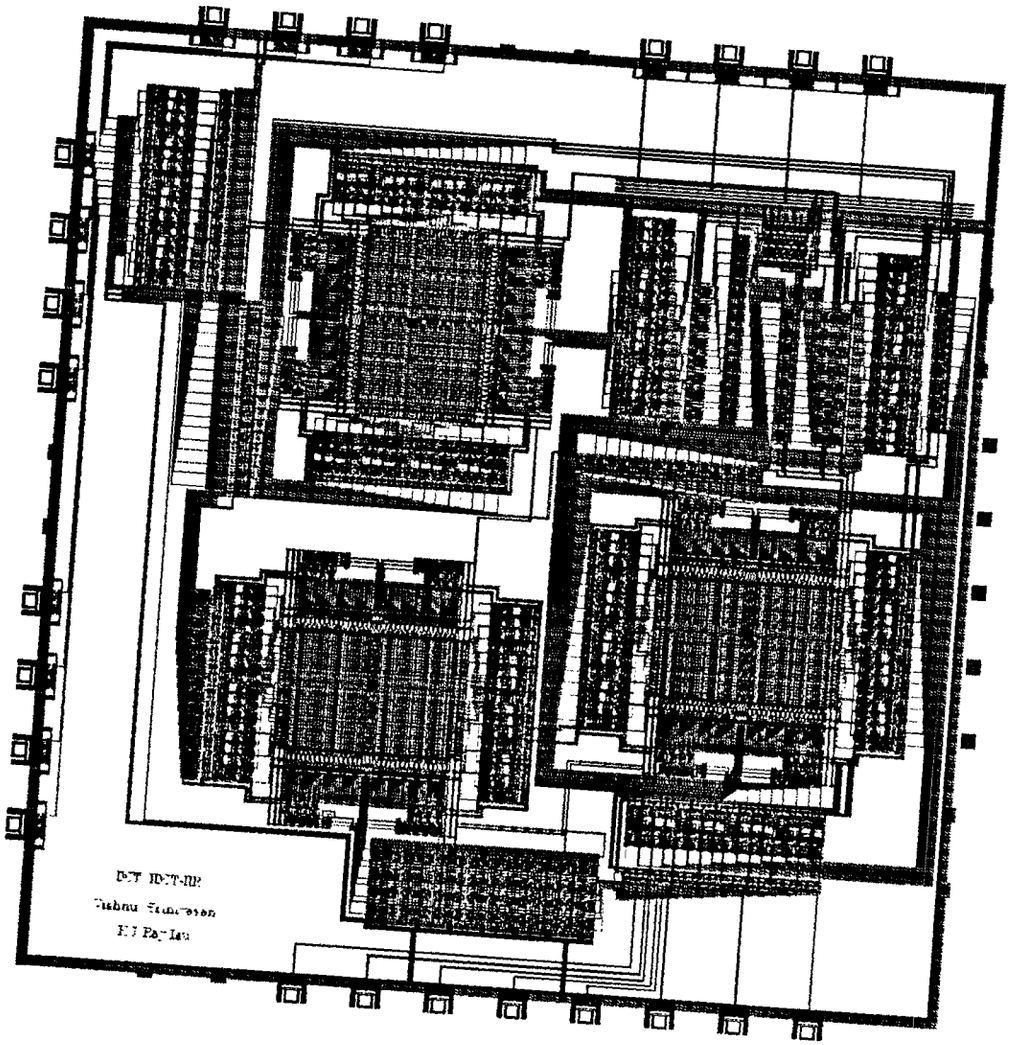


Figure 3.2: One dimensional DCT/IDCT Chip (Coefficient 5). Chip dimension is $6900\lambda \times 6800\lambda$. Area of 1-D DCT/IDCT chip is 46.9 mm^2.

Technology	2.0μ CMOS N-well
Die Dimensions	6900λ $\times$ 6800λ
Chip Area	46.9 mm^2
# of Transistors	25,000
Speed	25 MHz or 200 Mb/s

Table 3.2: 1D-DCT/IDCT Chip Statistics

decided later, not to fabricate this chip, but to proceed with the design of the complete two dimensional DCT chip. The data for this chip is based on our simulation data. The 1-D DCT/IDCT chip is shown in Fig. 3.2.

3.7 2-D DCT/IDCT Chip

The DSP ASIC which computes the Two-Dimensional Discrete Cosine Transform and its Inverse (2-D DCT/IDCT) is presented here. It is a fairly complex chip having many more control signals than the simple 1-D DCT/IDCT Test Module Chip described in the previous section. Much of the details of the control of this chip will be mentioned in the next chapter where the design of the various modules is addressed. Here, a basic overview of the various control and I/O pins are described (see Table 3.3).

The floorplan of the chip is shown in Fig. 3.3. The design of the chip proceeded in the following manner. A single 1-D module (like '1d0.mag') is designed first. This module makes use of previously designed and tested submodules like adders, latches, ROMs, delays, and multiplexers. After exhaustive logic functionality testing, this module is arrayed with a few modifications to the other

Function	Compute 2-D DCT/IDCT
Mode Selection	Active High 'Fwd' for Forward
Clock Pins	Phi 1: ϕ_1 & Phi 2: ϕ_2
Reset Pin (1D)	Active high 'Rst' pin
Reset Pin (2D)	Active high 'Rst_2d_C' pin
CSA Control	'Lat_shft_C' and 'Lat_inv_C'
2-D Inverse	'Inv_load_C' high for last block
Input Pins	ChipIn15, ..., ChipIn0 (16 pins)
Test Output	CSA: 1DOUT15, ..., 1DOUT0 (16 pins)
2-D Output	K0OUT16 — K0OUT1, ..., K7OUT16 — K7OUT1
Power Rails	'Vdd!' & 'GND!'

Table 3.3: 2D-DCT/IDCT Chip Description

modules. The 1-D module is now partially complete except for the inverse cell ('idct.mag' in Fig. 3.3) which is designed next. Now the whole 1-D is assembled and both forward and inverse transforms' logic-functionality are tested for all the 1-D channels put-together. This is the first phase of the implementation.

In a procedure similar to the one outlined above, we start the design of a single channel of the second dimension (say '2d1.mag'). When this is completed, logic-level simulations are performed to verify its functionality. After the simulate–debug–correct design cycle is completed, we proceed with the inverse module design for this stage. This concludes the second phase of our chip design.

In the third and perhaps most critical phase, the first and second dimension are linked together by the CSA module. This module contains not only a circular shift array, but also another circular shift register that stores data required for inverse computation at the second dimension. Even here, this module is designed separately, and then tested before combining it with the first dimension. This combined structure is again tested before the second dimension module is added. This is the final module that computes the 2-D DCT/IDCT. It is tested exhaustively before pads are placed. And once again, the complete chip is tested for its logic functionality.

It might seem a rather tedious task—logic-functionality verification while the module is designed, and then again, after it is assembled with another module, and so on. Not only is this the only way of ensuring a successful design, but it is the best way of minimizing any debug–design-change–verify cycle time. And at the low-level of design, as we have attempted here, any precaution taken can never be too cautious!

The 2-D DCT/IDCT chip statistics are presented in Table 3.4. Fig. 3.4 shows

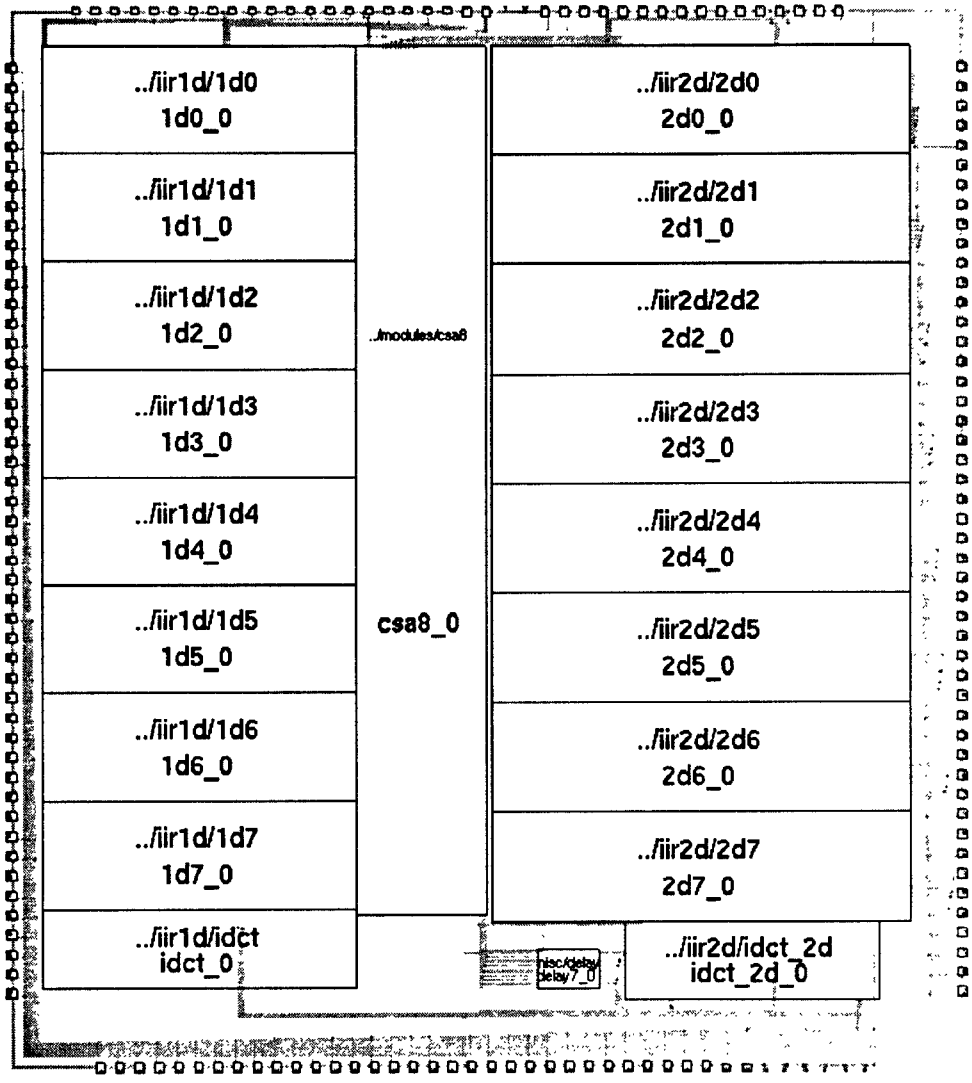


Figure 3.3: Floorplan of 2-D DCT/IDCT Chip.

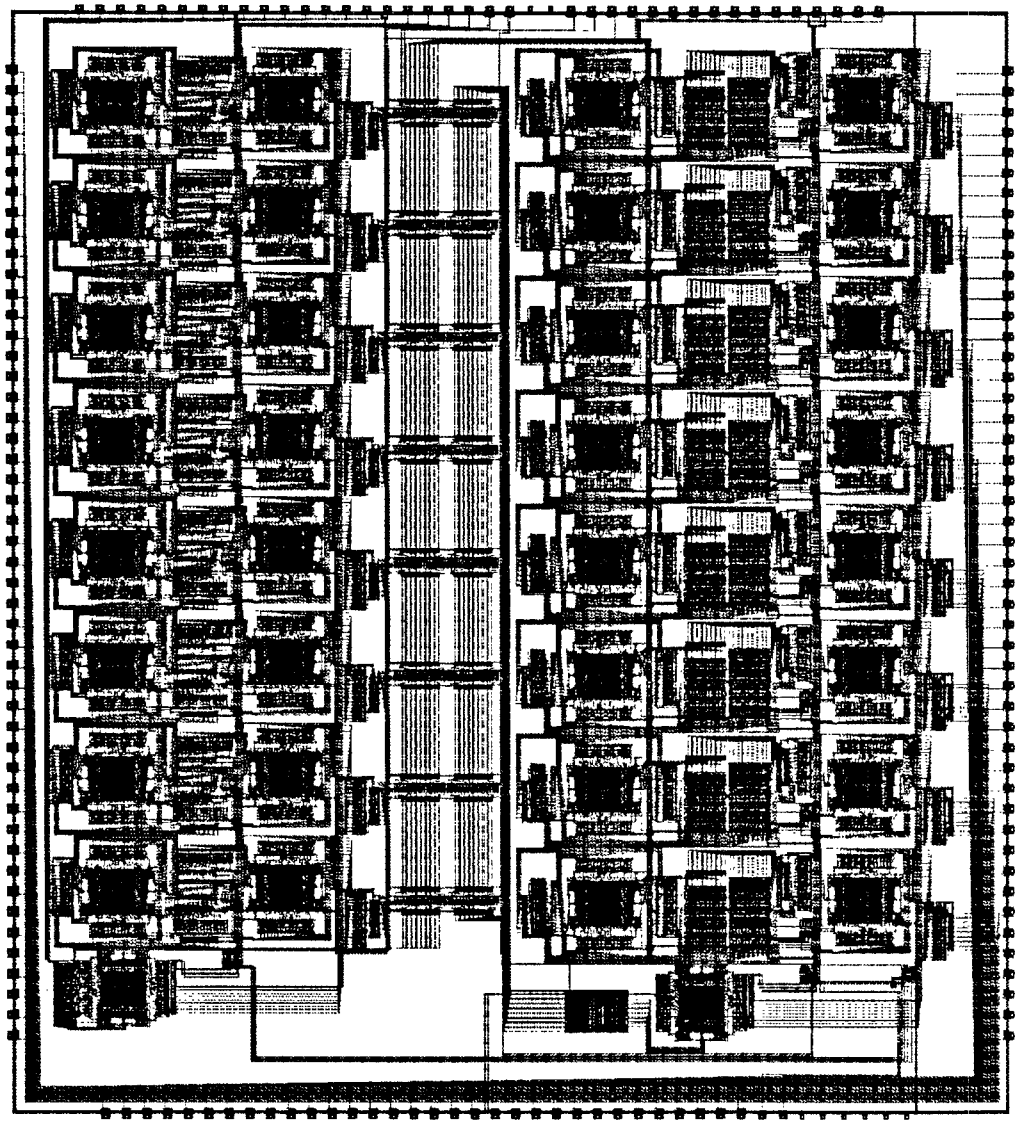


Figure 3.4: Two Dimensional DCT/IDCT Chip. Chip measures 24550λ $\times$ 27094λ. Area is 240 mm^2.

Technology	1.2μ CMOS N-well
Die Dimensions	$24550\lambda \times 27094\lambda$
Chip Area	240 mm^2
# of Transistors	320,000
Speed	50 MHz or 400 Mb/s

Table 3.4: 2D-DCT/IDCT Chip Statistics

the plot of the final 2-D DCT/IDCT chip. Considerable ingenuity had to be used to obtain this plot! The chip measures $24550\lambda \times 27094\lambda$ and has over 320,000 transistors. It has been submitted for fabrication at 1.2μ double metal, single poly CMOS N-well technology.

3.8 Summary

The various issues that have been addressed in this chapter range from consideration of the design choices that we had for our architectural choices, an overview of on what basis the choices for our implementation made, to the actual implementation outline. The floor-plan of the chip is shown and in the following chapter, we will look at the design of the various submodules.

Chapter 4

Sub-module Design

In this chapter design of various sub-modules, from simple modules such as half-latches, delay and clock-buffers to more complex modules such as multiplexers, adders, and ROMs are outlined. These submodules are used as building blocks for our higher-level (in the design hierarchy) modules like the '1d0.mag' (see Fig 3.3). In order to implement the chip in an efficient manner, we have tried to avoid reinventing the wheel—we have used parts of modules that have been designed before for other chips. In such cases, instead of designing from scratch, modifications were made and adapted to our needs.

Half-latches with and without reset pins, invertors, line and clock buffers, multiplexers, adders, and ROMs are some of the most basic modules that have been designed. These will be discussed first. Other higher-level modules like the '1d0.mag' and '2d0.mag' are discussed next. Also discussed in this chapter are how these modules are put-together to form the complete chip. Now, let us start with the more complex of the submodules, adders, and ROMs.

4.1 Adder

There are several possible designs for adders. They could be based on the simple majority logic function approach [18] implemented using straightforward combinational gates. Either static or dynamic gates can be used. Implementation can employ either the simple, but slow ripple carry adder or utilize the fast, but complex carry lookahead (CLA) method. There are pros and cons to every choice, and depending upon the application one should choose the appropriate design.

We need to choose the design which would yield the least propagation delay, and at the same time would not require elaborate clocking precharge schemes or take up too much area. Carry lookahead adders would give us the best performance in terms of delay, but it is not practical to implement CLA with more than 4-bits as they become too expensive area-wise. Ripple carry adders on the other hand, would yield compact layouts, but their speed would depend on the number of bits being added.

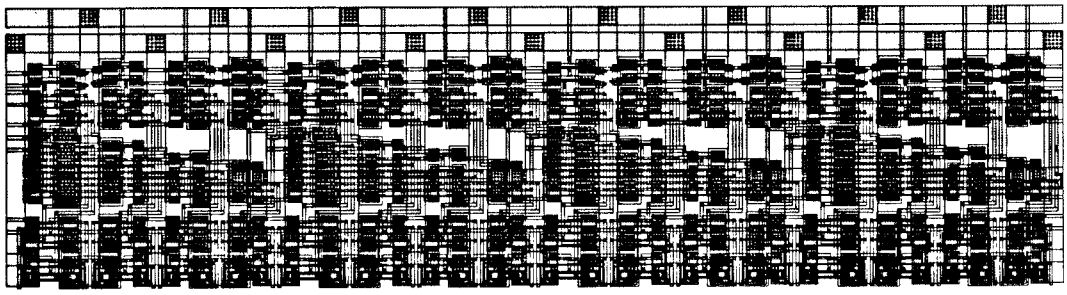


Figure 4.1: Physical layout of the 16-bit carry-lookahead–ripple Adder. Module dimension is $1348\lambda \times 355\lambda$.

In our case, accuracy simulations indicated the need for a 16-bit wide bus.

Clearly, a simple ripple carry adder would have been too slow for our requirements. At the same time, building a CLA scheme for 16 bits is impractical. A good compromise that we chose, was to use a pre-existing implementation of a 4-bit CLA, and connect up four of these units in a ripple-carry propagate fashion to obtain a 16-bit carry lookahead-cum-ripple adder. In other words, a high-speed bit-parallel implementation. As our timing analysis in Chapter 5 indicates, this provides adequate speedup.

The basic 4-bit carry lookahead adder implementation is based on [18]. The adder module is shown in Fig. 4.1. This module has 704 transistors and measures $1348\lambda \times 355\lambda$ units. Simulations performed using 1.2 μ technology parameters indicate a maximum propagation delay of 9 ns. The output signal is buffered by 2 large invertors to give sufficient drive capability.

Based on the basic adder implementation, there are three different versions. There are some cases (like in, 2's complement negation) where carry-in is hard-wired to logic 1. Instead of having to manually connecting the carry line to 1, we call the 'adder_h.mag' module. Similarly, when carry=0, the module 'adder_l.mag' is selected. The most general, 'adder.mag' is used in special cases.

4.2 Distributed Arithmetic

This is perhaps one of the most critical submodules designed in this project. The ROM is optimized both for speed and area. It is especially important, as the ROM is arrayed 34 times in the complete chip and occupies a significant percentage of the chip area. Even a small reduction in ROM area would affect our chip area statistics considerably. While speedwise, ROMs are much superior

to multipliers, their area increases exponentially with the input word size. Also, they can be used in only those applications which require multiplication of the input word by a predetermined constant.

4.2.1 ROM Design

ROM or Read Only Memory stores a predetermined sequence of 1s and 0s, with each bit being implemented by a transistor. By storing appropriate bit-sequences for every possible input (ROM lookup table entry selector) the multiplication is effected. A straightforward ROM implementation of a multiplier would require 2^x entries for an input which is x bits wide.

For our system, we conducted extensive simulations, based on those results, we found that we needed an internal bus precision of at least 14 bits to achieve our accuracy criterion. But from our initial area estimates, it would not have been feasable to implement a ROM of this size. Basically, if the size of the ROM becomes much larger than a multiplier, economics dictates that we use multipliers. But in doing so, it leads to a slower implementation. The tradeoff point comes at 12-13 bits for the ROM input. Keeping this in mind, we tried other alternatives, like using a 12 bit input ROM, but with 16-bit table-entry and internal bus precision. We performed simulations to verify that it would yield the required accuracy.

In all, we designed, three different ROM structures. All the ROMs that were designed are based on the ROM implementation described in [19] and [20]. The three parameters of concern are

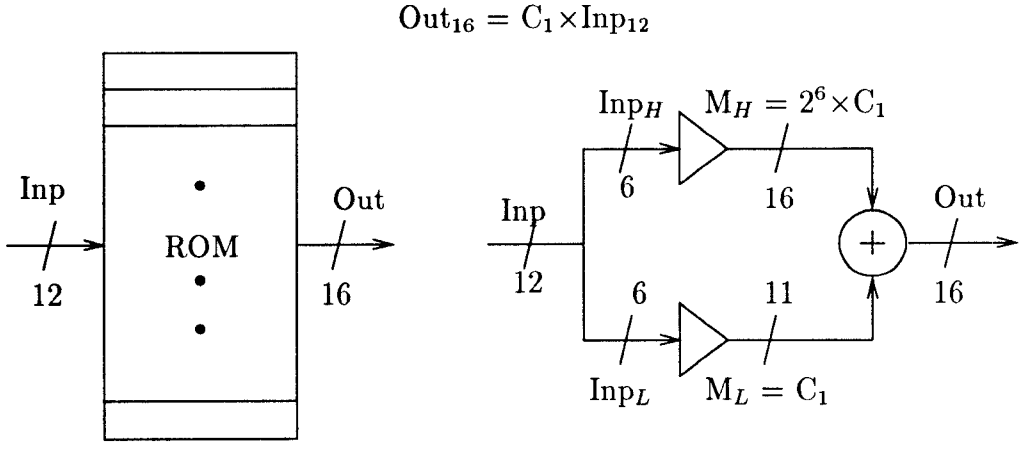
1. Input word precision (≤ 12).

2. Output word precision: which is the same as internal bus precision; choices of consideration are 12 and 16 bits.

3. Interdigitated ROM: whether two lookup tables are combined to utilize area more efficiently.

Hardware Structure

Now, lets take a look at the design of ROM which was finally used in the DCT/IDCT chip implementation. It has 12-bit input word and 16-bit output word precision. The ROM lookup tables are interdigitated and multiplication of the 12-bit input word by two different coefficients is simulated by performing simultaneous table-lookups corresponding to the two coefficients.



(4096 rows x 16 bits)

(a) Naive implementation

(64 rows x 27 bits)

(b) Our ROM Implementation

Figure 4.2: ROM Design Strategy

A naive implementation of the lookup table would consist of 2^{12} or 4096

rows of 16 bit words. But instead of such an inefficient implementation, we split-up the input word into two 6-bit words, say Inp_L and Inp_H, and effect the multiplication in the following manner

$$Inp = 2^6 \times Inp_H + Inp_L.$$

The output, $Out = C_1 \times Inp$, is computed as

$$Out = 2^6 \times C_1 \times Inp_H + C_1 \times Inp_L.$$

By precomputing the two sub-products with the required accuracy, the output if formed by storing the sub-products in the ROM lookup table and performing the addition. This is illustrated in Fig.4.2.

We now require two tables, each with 2^6 or 64 rows only. The lower and higher order ROM entries are to be added with the proper shift, taking into account the 2's complement representation for the numbers. It turns out that we need to store 16 bits for the upper precomputed product and 11 bits for the lower product. This is how the product is computed for one of the coefficients, C_1. It is the same for the other, and we will see how computation of two products is put together in an efficient manner in the next section.

The values which are stored in the ROM tables are precomputed using double-precision numbers on a Sun workstation. This actually increases the accuracy of ROMs over regular multipliers which actually perform multiplication with the given input word precisions.

4.2.2 VLSI aspects

Most existing ROMs are based on the precharged scheme, where the bit lines are precharged high and during the evaluate phase, according to the stored bit-

sequence, selected lines are discharged. Our ROM is based on a novel design [19], which reduces the access time. The main components of the ROM are tree-based row decoder, memory cells, and sense-amp. The ROM schematic is shown in Fig. 4.3.

As illustrated in Fig. 4.3, the two lookup tables (for high and low) are placed next to each other. There are 64 rows, and the row select lines for the low-lookup table are actually routed through the center.

6-64 Decoder

We need to decode the input word lines and select the appropriate ROM word. By using two 3-bit decoders and an array of 64 AND gates, instead of a straightforward 6-bit decoder implementation, the access time can be improved from 20.4ns to 8.55ns (Delay simulations are performed using MOSIS 2.0μ technology parameters [20]). The design of the 6-bit decoder is shown in Fig. 4.4.

Lookup Table

The ROM units cells which encode a 1 and 0 are shown in Fig. 4.5. These cells are identical and measure $13\lambda \times 16\lambda$. Depending on whether a logic high or low is to be stored, we have either a p-channel or a n-channel transistor connected between the output line and Vdd or Gnd, respectively. The transistor gate is connected to the row-select line of the decoder.

In our ROM table there are totally $(16+11) \times 2 \times 64$ or 3456 unit cells. Each cell corresponds to one bit of storage. As we shall see, the pitch of these cells is designed to be half that of the sense-amp. This facilitates in the interdigitation of coefficients, leading to a much higher density.

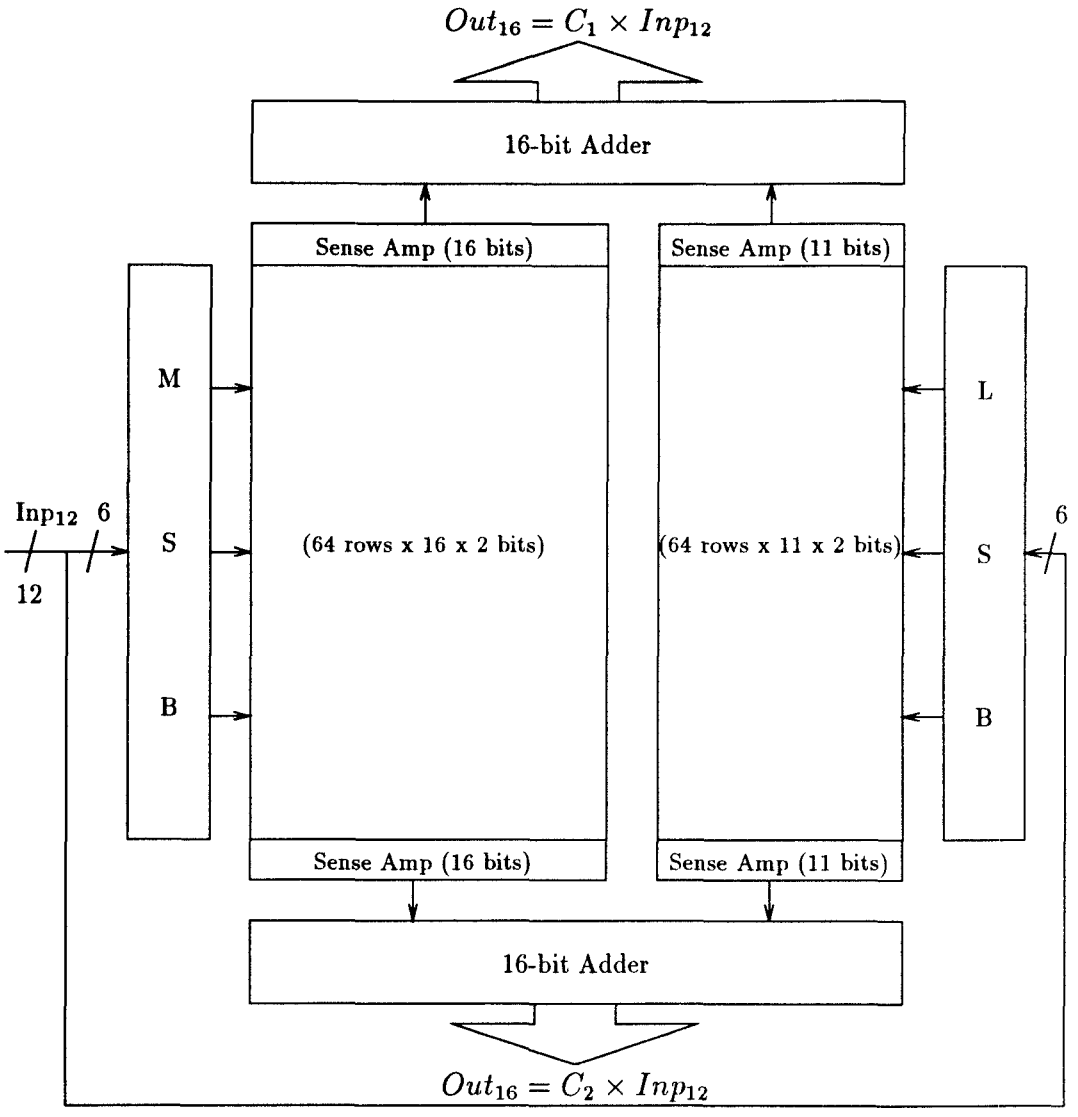


Figure 4.3: ROM Multiplier Schematic

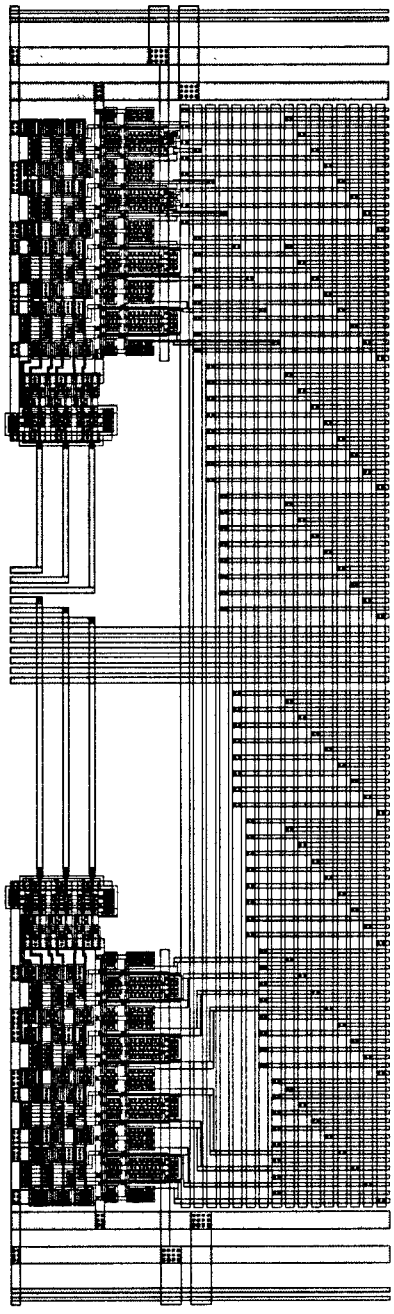


Figure 4.4: Physical layout 6-bit decoder ($377\lambda \times 1292\lambda$).

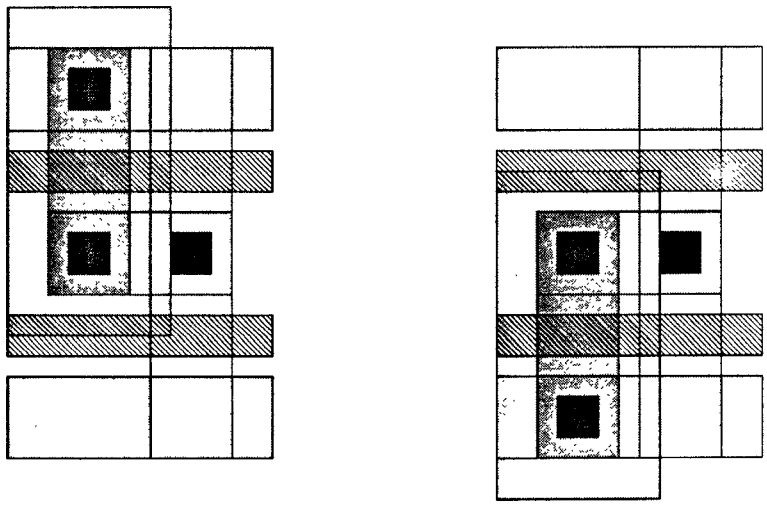


Figure 4.5: ROM unit-cells of size $13\lambda \times 16\lambda$. Encode bits 1 and 0 .

Sense-Amp

The function of this simple module is to speed up the word-lookup. It works on the simple principle of precharging the bit-lines to an intermediate voltage between Vdd and Gnd. In this way, regardless of whether the bit-lines are turned on or off, the delay time is reduced/optimized.

The schematic of the sense-amp is shown in Fig 4.6 (b). In the precharge phase of the clock ($\phi 1$ is low), the p-MOS shorts the input and output of the invertor, which forces the bit-lines to the invertor transition point at 2.5 volts (Simulations have however indicated that this voltage is closer to 3 volts). In the evaluate phase, the p-MOS transistor turns off and the bit-line signal is latched at the output through the simple butterfly-pass transistor. The SA measures $26\lambda \times 94\lambda$. The pitch of the SA is twice that of the unit-cell, allowing two such cell for every SA. By placing one set of SA at the bottom, and another at the top, the product for two coefficients is computed at the same time.

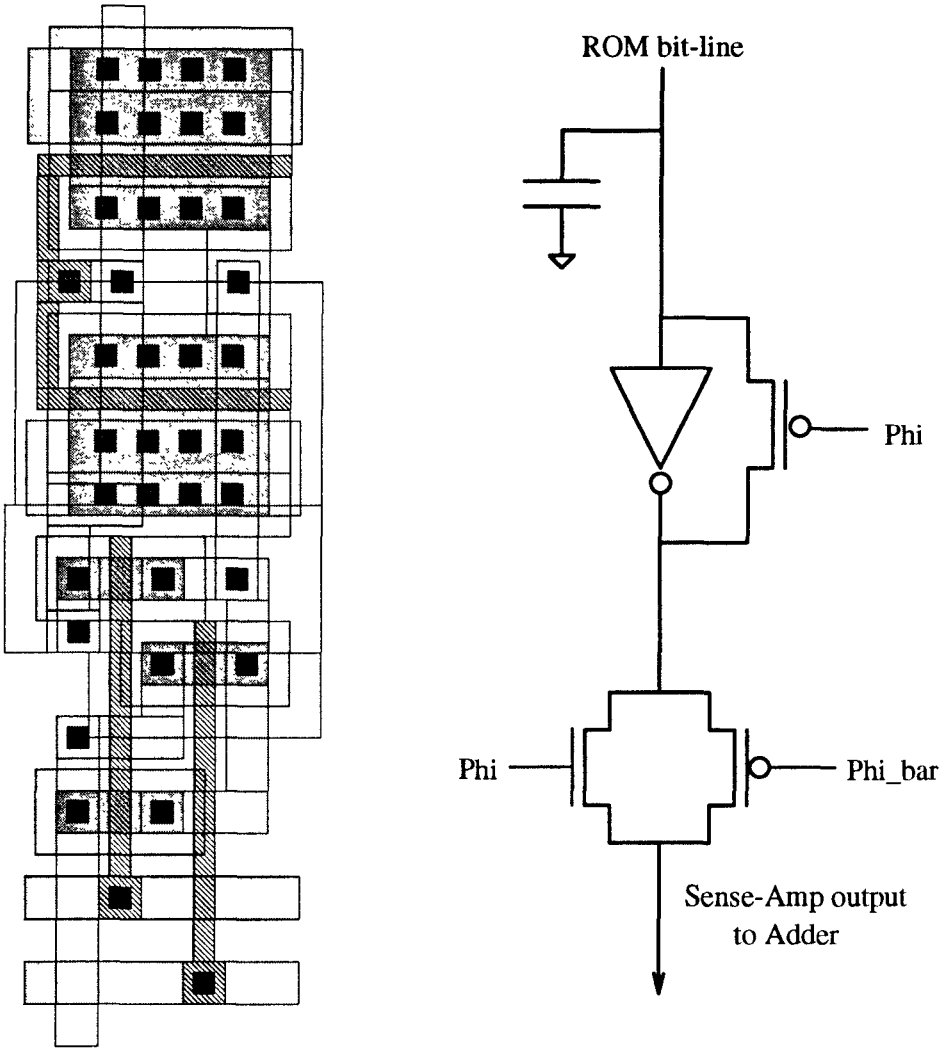


Figure 4.6: Sense-Amp: (a) Physical layout of SA ($26\lambda \times 94\lambda$), (b) Corresponding circuit schematic.

Combining H & L sums

The high order sum is combined with the sign-extended low-order sum using the adder module described in the previous section. The total time required for multiplication is now the combined propagation delay of ROM table-lookup and adder.

4.2.3 ROM Implementation

The first ROM is designed in the regular manner—design the individual cells like sense-amp, 3-bit decoders, 6-bit decoders, and 0/1 unit cells in their various orientation in different locations. In this project, we were able to reuse some of the ROM subunits used in a previous chip implementation. Once a complete ROM was assembled (with dummy coefficients), it was broken up into tiny subunits, and their locations and arraying information noted. Now, by using a perl script, new ROMs were assembled. The crucial part was the encoding of the interdigitated coefficients. The sin or cos coefficient was computed using double-precision floating point arithmetic on the Sun workstation, and these were inputted to the perl script. These numbers were converted to 2s complement binary, and routines were called to place the "1-unitcell" or "0-unitcell" depending on the bit to be stored in that location.

The perl script writes all the information about automatic placement of the ROM subunits in a temporary file. The script then invokes Magic as a child process which takes this temp file as its input and does the actual job of putting together all the modules to form the final ROM submodule. The automatically generated ROM is carefully tested for functionality verification, design rule checking and timing analysis. Now, this process is repeated for all the multipliers

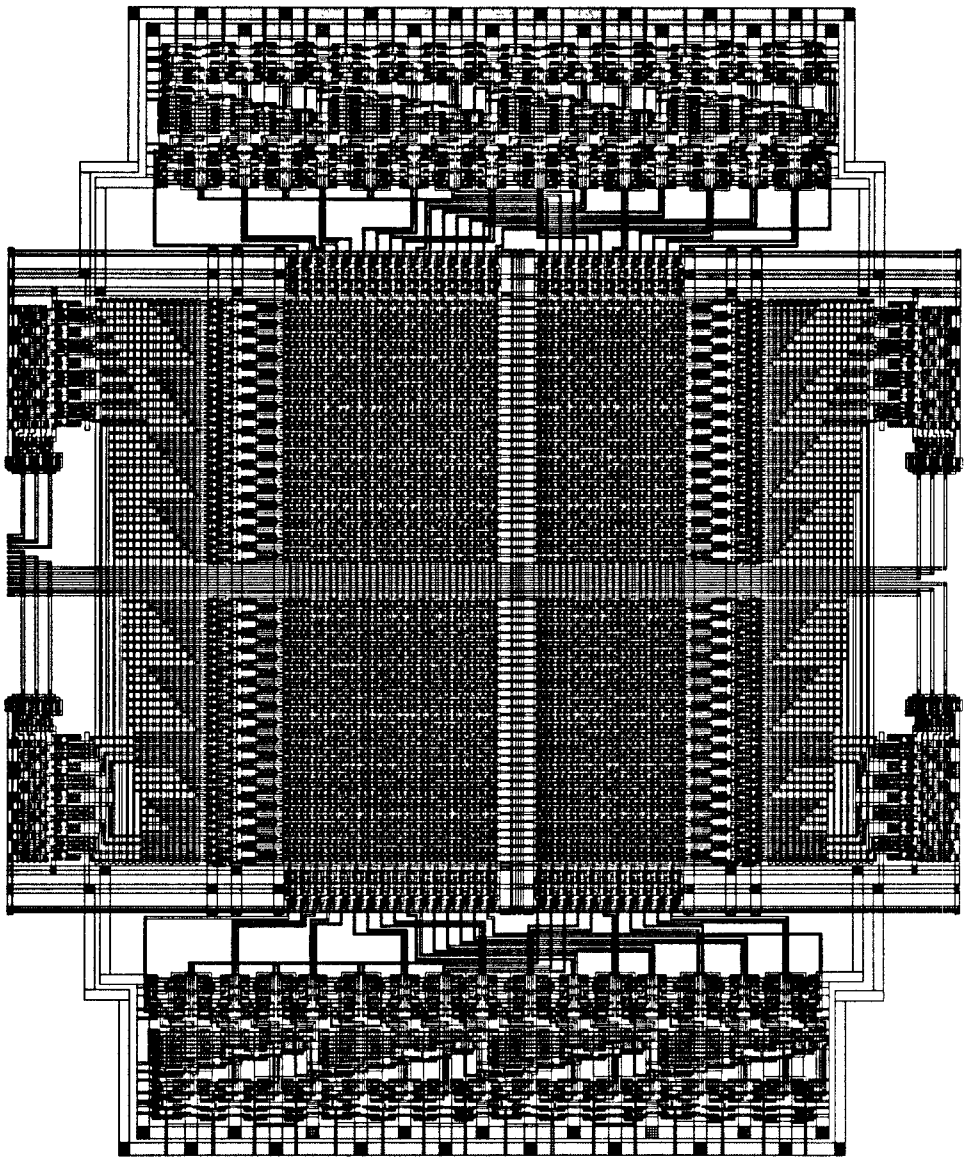


Figure 4.7: Physical layout of ROM lookup-table with associated adders. Dimension is $2226\lambda \times 1854\lambda$.

which are needed for the different channels. The coefficients are computed and the perl script is invoked for every one of these coefficient.

The size of the basic ROM structure is $1292\lambda \times 1854\lambda$, but with the adders to combine the high and low order products, the ROM/adder assembly measures $2226\lambda \times 1854\lambda$. The combined ROM lookup-table and adder to effect multiplication is shown in Fig. 4.7.

4.3 Shift-Registers

Several different kinds of latches and registers are used in our VLSI implementation. The latches are used to latch and store the data intermediate computations. Dynamic half latches, with and without reset circuitry are implemented. Use of dynamic latches and a non-overlapping split-phase clocking scheme (see Fig. 5.5) enables us to do away with more complex and area-consuming flip-flop based registers.

4.3.1 Half-Latch with Reset

The schematic for this latch is shown in Fig. 4.8, and the VLSI layout in Fig. 4.9. It is 16 bits wide corresponding to the internal bus precision. Depending on its location in signal flow graph (see Fig. 4.13), it latches on either Phase ϕ_1 or Phase ϕ_2. As shown in the schematic, provision to reset the latch is provided.

To design this latch, we first layout a single bit. This is tested with Irsim for functionality first, and then with Spice for timing analysis. The size of output invertor is made sufficiently large to allow adequate driving of expected output load in the module, where it would be used. This was done by connecting to the

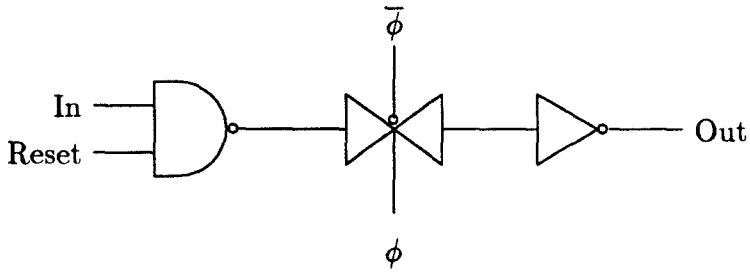


Figure 4.8: Schematic of Half-latch with Reset Control Circuitry

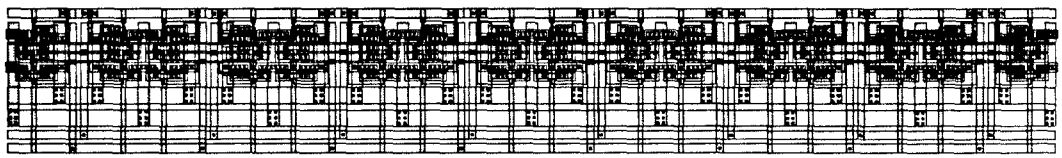


Figure 4.9: Physical layout of Half-latch with Reset. Module dimension is $938\lambda \times 127\lambda$.

output invertor, a representative M1 or M2 wire of typical length and carrying out the timing analysis. Once the testing is completed, the final module is assembled by arraying the one bit. Local distribution of all control signals are taken into account at the 1-bit design stage itself. So after the arraying, the module is ready with all clock/control signals and power rails already wired. Of course, the module is not fully ready until functional verification is done again for the 16-bit wide latch. At this level, either Spice or Irsim can be used for functional verification.

4.3.2 Half-latch Invertor with Reset

This module was designed by simply adding an invertor stage at the output of the half-latch. The circuit schematic is illustrated in the Fig. 4.10. It was designed to aid in implementation of the subtraction in the SFG (Fig.2.2) in conjunction with the module 'adder1.mag'. Two's complement subtraction is implemented by adding the complement of the number being subtracted, to the other number, plus one.

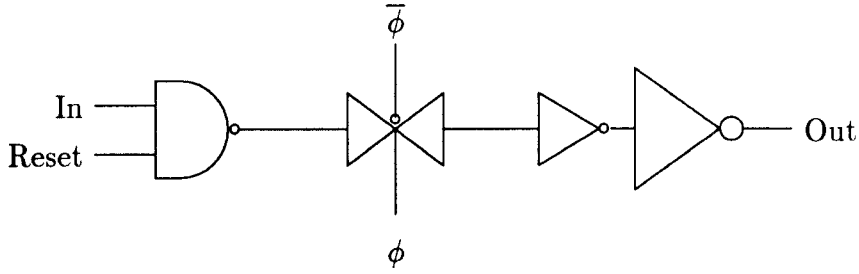


Figure 4.10: Half-latch Invertor with Reset Control

4.3.3 Delay, Delay7 and Delay8

These are shift-registers which act as clock delays for 1, 7 and 8 cycles. As before they are implemented by modifying the basic 1-bit unit to latch for an entire clock period. The 1-bit 1-clock period delay unit is arrayed 16 times to obtain a 16-bit wide bus. And this is further arrayed 7 or 8 times to get the final modules. The schematic for the basic unit is shown in Fig. 4.11. The control/clock signals required by this module are 'Reset', ϕ_1 and ϕ_2. The 'delay.mag' module measures $938\lambda \times 217\lambda$. The 'delay7.mag' measures $1028\lambda \times 1549\lambda$, and the 'delay8.mag', $1028\lambda \times 1771\lambda$.

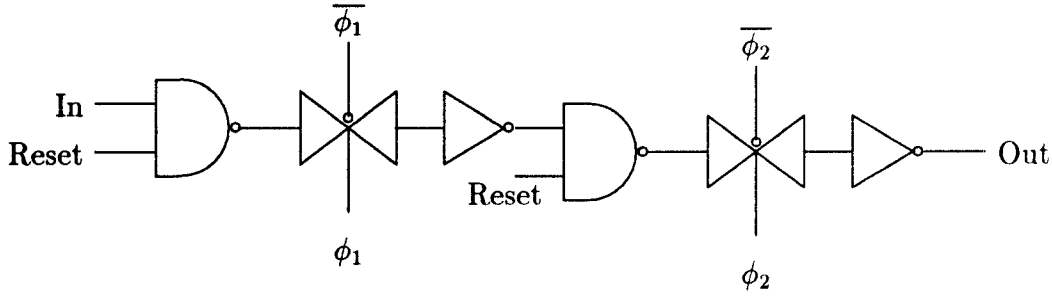


Figure 4.11: Schematic of Delay module

4.4 Multiplexers

These are needed to choose between two sets of signals. It is used after the ROM (see Fig. 4.13) and other locations in the signal flow graph affected by a switch from computing the forward to the inverse transform. The circuit schematic is illustrated in Fig. 4.12. The size of the multiplexer is $1271\lambda \times 119\lambda$.

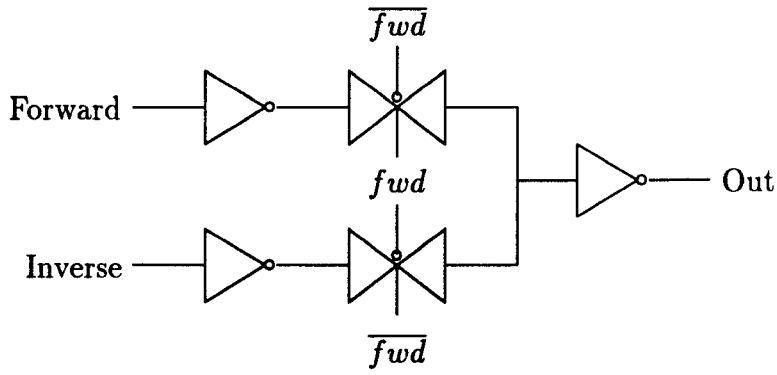


Figure 4.12: Schematic of Multiplexer

4.5 Control Signal Distribution

This is a fairly important issue as there are quite a few control signals that need to be distributed to the various clocked modules like latches, delays, and multiplexers. The basic idea is to distribute the master control signal to each module (such as 1-D/2-D channel or the 1-D/2-D inverse module) and use a local buffer to generate the local control signals, which are then distributed to all modules within that particular submodule. This is essentially a multi-level tree distribution of the control signals. By employing such a scheme, we are not only able to minimize skew, but also have improved rise and fall times. This scheme is particularly relevant to the clock signal distribution.

The 'rst' control signal is to be distributed to all those modules that are clocked as they need to be reset between blocks. The 'fwd' signal which determines whether the forward or inverse DCT is computed is routed to all the multiplexers. The modules which require these control signals also need the

complement (which is generated at the local buffer). It is not necessary to route the complement of the control signals on a global chip scale. Routing of these control signals to each bit slice is built into the sub-module design.

4.6 2-D DCT/IDCT Chip

The entire chip can itself be thought of as a giant module whose submodules are the 1-D, CSA, and the 2-D modules. These are now described.

4.6.1 1-D channel module

In the previous sections we have seen all the basic submodules such as adders, multiplexers, ROMs, and latches. Proceeding up the design hierarchy, we come to the module which computes the one-dimensional DCT and its inverse. This module is fairly complex and uses all of the previous submodules as library cells. Magic 'instantiates' [17] every occurrence of these modules, resulting in much faster layout, extraction, and DRC of the module.

The signal flow graph implemented by this module is shown in Fig. 4.13. All but multiplier M_3 and the associated multiplexers and latches are included in this module. Only one set of M_3 and its associated circuitry is needed for the entire 1-D module and that is designed as a separate module.

The physical layout of the 1-D channel is shown in Fig. 4.14. This corresponds to the "1dmod1.mag" module shown in Fig. 3.4. All of the eight channel modules are identical except that they instantiate different ROM tables. Channel 4 had to be handled somewhat differently as one of the ROM coefficients was zero.

The signal flow graph for the 1-D channel is a little different from what

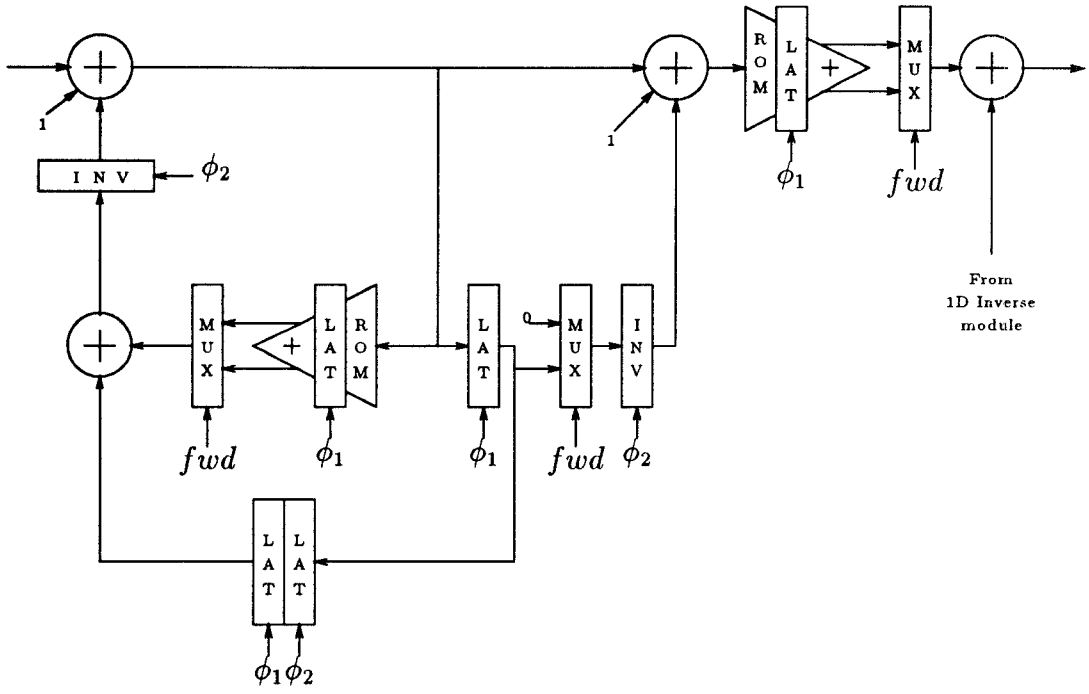


Figure 4.13: 1-D IIR SFG with clocks and latches

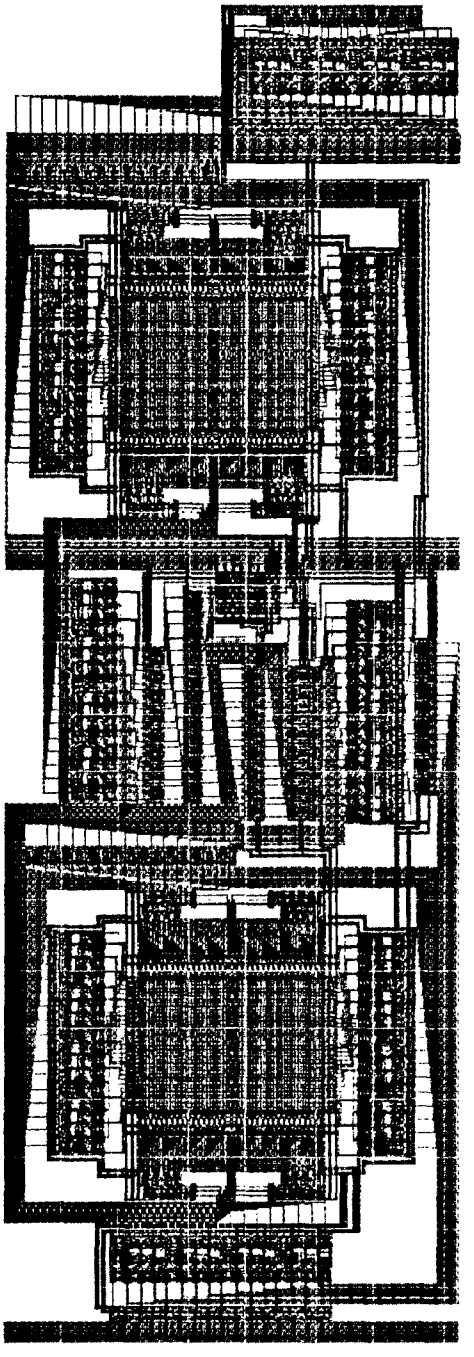


Figure 4.14: VLSI layout of 1-D channel. The module measures $2742\lambda \times 8029\lambda$.

is shown in Fig 2.2. The first difference is that the 8-unit delay in front is not required as only block transforms are computed. The second difference, is minor, but relevant from the VLSI implementation point of view. The negation of input for odd channels has been moved to M_2. This permits, the same 1-D channel design to be used for all the 8 channels of the 1-D module. The channel is replicated, but different ROM lookup tables are instantiated.

The circuit has been laid out in such a manner that it facilitates easy modular development. Inter-module connections are brought to the edges of the blocks where they get connected with the other modules wire-segments when these modules are tiled. By adopting such a methodology, we save considerably in design time and effort, and at the same time, if the modules are designed properly (matching pitch), we save in interconnection area requirement also.

The power rails, input/output, and other important control signals are routed from top to bottom in each module. As they are tiled vertically the routing of all signals is done automatically. We only have to concern ourselves with feeding these signals to the entire 1-D module, either from the top or the bottom, as local distribution of these signals is already taken care of in the design of the channel-module.

Extensive functionality verification was performed using Irsim. In fact it took a significant time to complete the verify–debug–modify-design iteration cycle. A reason for this is that all routing of wires between modules was done manually with the layout editor, and this process is error-prone.

Regarding the control signals, as discussed before, this module requires phase ϕ_1 and phase ϕ_2. The control signal 'Fwd', is active high and controls whether this module computes the forward or the inverse transform. 'Rst' is another

important control line. It is active low reset. All latches and delay units need to be reset between consecutive 1-D blocks of data.

1-D Inverse Module

The layout of the module which computes the inverse term is shown in Fig. 4.15. This is placed below Channel 7 in the 1D module. Even for this, the design is such that all external routing is unnecessary.

4.6.2 2-D Channel Module

This module is designed along similar lines as the previous 1-D channel-module. The SFG (see Fig. 4.16) is almost the same, except that 8-block delay units are present in both loops. This facilitates computation of 8 blocks of data in a time-displaced parallel fashion. The same concept can also viewed from systolic point of view.

The VLSI layout of this module is shown in Fig. 4.17. But, designing and debugging the module were much quicker than the 1-D channel, primarily due to the experience gained. Even here extensive Irsim simulations are performed to verify the functionality. The control signals for the second dimension are somewhat more complex. They will be explained along with the control signals for the CSA module which will be described next.

Inverse

The inverse module for the second dimension is quite similar to the 1-D inverse module. It however has an additional control signal Inv_load_C. This is active high and should be high for the last 8 cycles when the inverse coefficients are

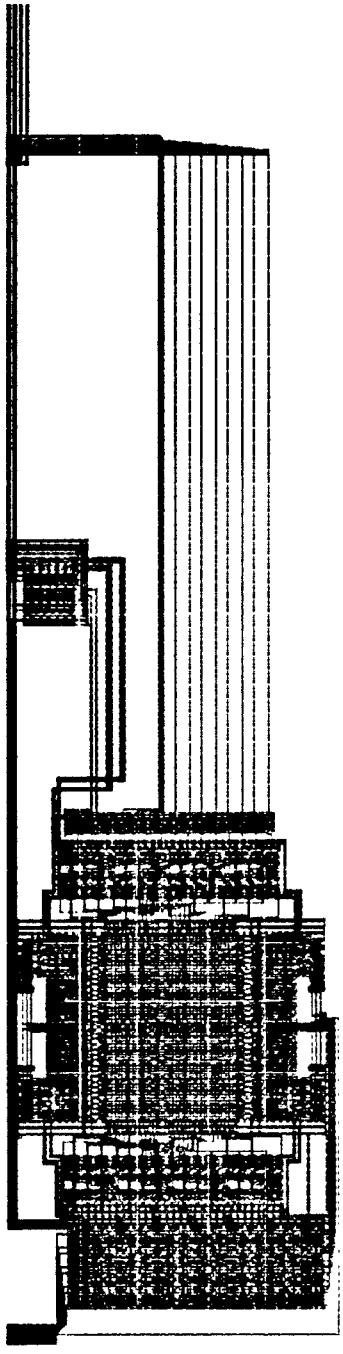


Figure 4.15: 1-D Inverse Module Layout

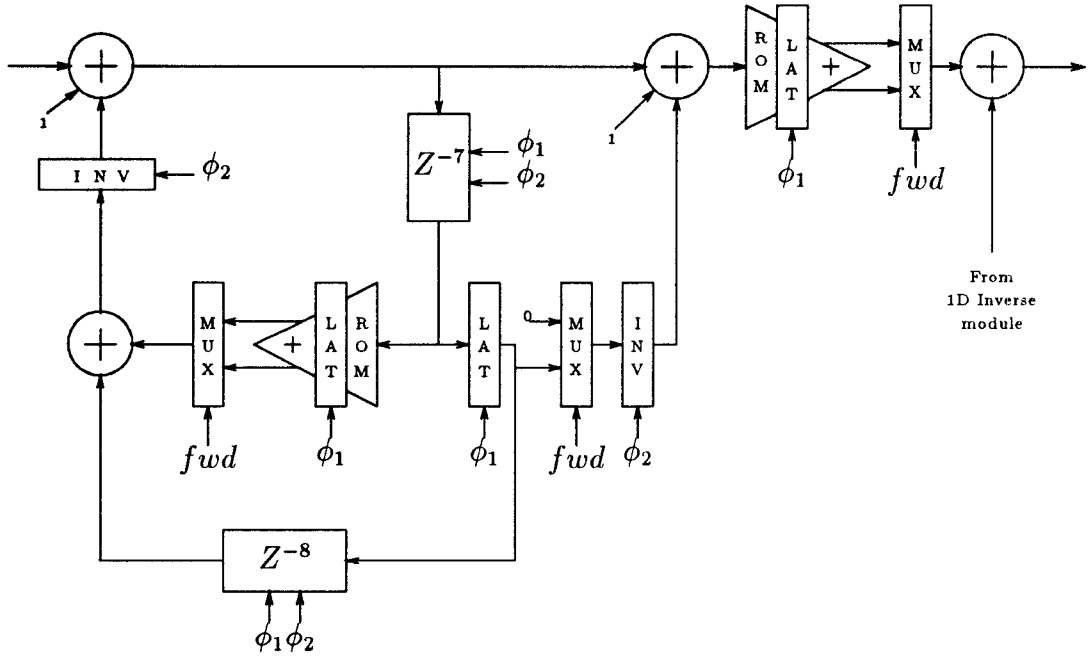


Figure 4.16: 2-D IIR SFG with latches and clock details

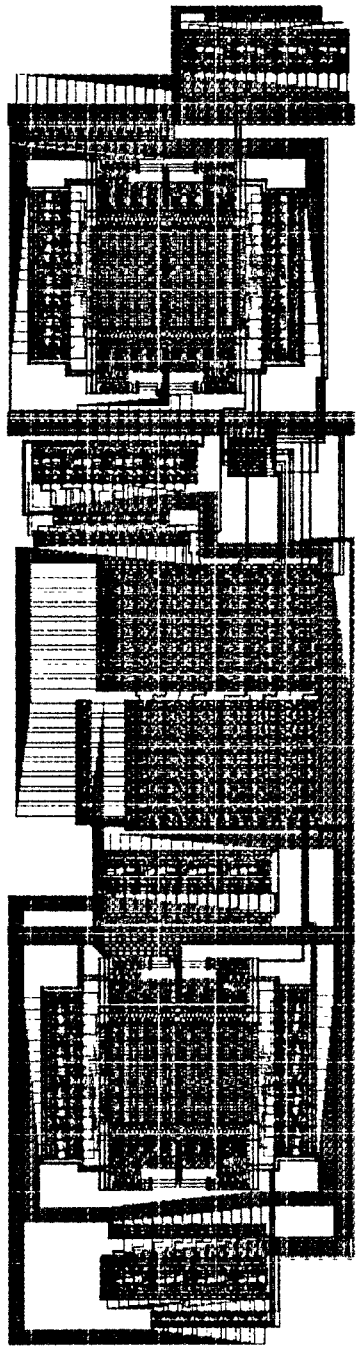


Figure 4.17: 2-D Module Layout ($2784\lambda \times 10672\lambda$)

being generated by the 2-D channels.

4.6.3 CSA

This module was the last designed. It takes the 8 parallel 16-bit words generated by the 1-D module and feed it serially to the 2-D module. The CSA module serves another important function—of storing the 1-D IDCT coefficients of the first row required for computation of the inverse at the second stage. But these 8 16-bit numbers have to be stored for 64 clock cycles. As dynamic-latches are used, to avoid the charge leakage problem, the data is circulated in this group of registers until it needs to be outputted to the 2-D channel modules. The CSA module can be identified as the central non-dense set of modules between the 1-D and 2-D in the complete chip plot in Fig. 3.4.

There are several control signals for this module—to read data from the 1-D module, to latch it in, and shift it out serially to the 2-D module, to latch in data to help in computing of the inverse, and to hold it until required. These control signals are Lat_shft_C and Lat_inv_C. It is during the time the 1-D module is being reset that the 1-D channel outputs are latched into the CSA. At the same time the 2-D DCT is also being computed. In the clock period when the 1-D channel modules are being reset, the 2-D channel modules are not clocked. This is done to ensure synchronicity. For the same reason, the CSA's second set of shift registers which circulate the first row of 1-D DCT coefficients, is also not clocked in that period.

4.7 Summary

This chapter describes the design and implementation details of all the submodules that have been used to build the DSP chip. The implementation of ROM and adder submodules are described in detail. The schematic and layouts of the various other modules like half-latches, latched-invertors, delays, and multiplexers are given. The design procedure of the 2-D DCT/IDCT chip is described next. The 1-D channel, 2-D channel, and their inverse modules along with how they are put together with the CSA to form the complete chip is explained.

Chapter 5

Simulation Results

The various simulations which have been performed at different stages of the project are outlined in this chapter. These simulations are presented mostly in the order they were performed in. Architectural models for IIR and Lattice structure are constructed in C. Simulations are performed using these models to determine the system specifications such as internal bus precision and clocking scheme which satisfy not only the accuracy requirements, but also optimize chip area and delay. Timing analysis is performed on the critical path sub-modules like adders and ROMs. Spice is primarily used to perform the timing simulations. Important clocking issues are considered next. The functionality verification is performed at every level of the hierarchy. Irsim is the tool which is used for this purpose. It also helps in estimating the power consumption of the chip.

5.1 Architectural Simulations

Even though the lattice and IIR structures that we have considered for our DCT/IDCT VLSI implementation are theoretically equivalent, the lattice struc-

ture is superior to IIR structure when implemented with finite precision arithmetic. After preliminary considerations when IIR structure was chosen for VLSI implementation, we focused the architectural level simulations to determine system parameters. These models are developed in C, and they reflect the truncation effects which take place due to finite wordlength in the DCT/IDCT computations. The simulation results are summarized and presented in Table 5.2

The DCT/IDCT chip finds its application in image processing/compression modules of teleconferencing/HDTV which require signal-to-noise ratio of more than 40 dB. Itemizing the three criteria which have been taken into consideration in the architecture selection are

- Chip area,
- Accuracy,
- Propagation delay of critical path modules.

Choosing a structure which minimizes the chip area is a fairly important consideration, and along with this, being able to compute the forward and inverse transform with minimum additional area was also taken into account. The choice between distributed arithmetic and conventional parallel multipliers affected all of our criteria. Miscellaneous implementation issues such as clocking scheme and distribution of control signal were decided based on optimization of these criteria.

ROM vs Multiplier

Table 5.1 gives the key performance measures—the area and the propagation delay of different ROMs and a multiplier, that we could have used in our VLSI

implementation. The 12-bit input, 12-bit output two-coefficient interdigitated ROM is from an earlier implementation [20]. This design was completely rebuilt, using some of the existing cells to layout the first and third ROM listed in the table. The first ROM is for a single coefficient while the others have an interdigitated structure and implement multiplication by two coefficients. The fourth entry is a two's complement parallel multiplier which had been used in a different chip. As a fast and efficient parallel multiplier, it gives us a good comparison reference.

	Precision	Size	Delay
ROM (one coeff.)	12 x 12	1217λ x 1532λ	20 nS
ROM	12 x 12	1292λ x 1646λ	20 nS
ROM	12 x 16	1292λ x 1854λ	20 nS
ROM	16 x 16	Not feasable	20 nS
Multiplier	16 x 16	3513λ x 1568λ	80 nS

Table 5.1: ROM vs Multiplier Comparisons

From the area point of view, the ROMs with 12-bit input word length are clearly better than the multiplier. And table look-up is about four times faster than the parallel multiplier. These propagation delays are based on 2.0 μ technology simulations. Unless architectural simulations demand a ROM with more than 12 bit input word-length to satisfy accuracy requirements, the ROM is the clear choice for high-speed and reduced area requirements.

Simulation Outline

For our simulations, image data is used instead of random numbers. Thus the simulation results reflect real-life situations. For this purpose, the images—Lena, Airplane and Sailboat are used. These images are 512×512 pixels with a depth of 256 gray-scales. As we consider block size of 8, the image is segmented into blocks of 8×8 and fed to the DCT block as shown in Fig. 5.1.

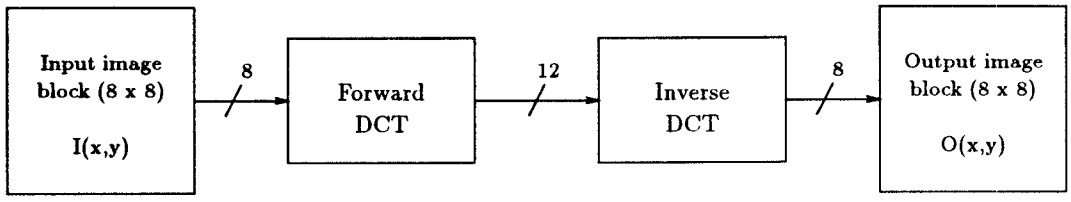


Figure 5.1: Accuracy simulation block-outline

The image data is read in block-by-block by the C program which serves as the architectural model for IIR structure. In these simulations, implementation details like internal bus precision and ROM input word length to achieve the required accuracy criterion of PSNR of 40 dB. The feasability of using ROM lookup tables is studied here. Table 5.2 highlights the simulation results (accuracy criterion) for the IIR structure for different choices of bus precision and ROM input word length.

The implementation issues that this simulation focused on are the internal bus precision and the ROM input word-length. Average SNR and PSNR as defined in equation 5.1 are used to quantify the truncation error. The table shows the simulation results obtained with Lena image. Statistics collected are

# of ROM input bits	Internal bus precision	Average SNR	Peak SNR	Comments
12	12	21.1 dB	27.3 dB	Low ($\sim$ 200)
12	16	37.8 dB	44.0 dB	Medium ($\sim$ 800)
16	16	38.0 dB	44.2 dB	High ($\sim$ 1100)

Table 5.2: Finite precision arithmetic simulation of IIR Architecture (Input Data: 4096 8×8 blocks of LENA).

averaged over 4096 blocks of size 8 × 8. We notice that the first case (ROM with 12-bit input wordlength and 12-bit bus precision) does not have sufficient accuracy. The 'Comments' column indicates the number of blocks for which the output image pixel value matched the input image pixel. For such blocks the noise is zero and SNR is not computed. As these blocks are not included in the SNR figure, we should also look at the block-count, and this will give a rough indication of how much better the actual SNR would be. The third case in which the ROM has a 16-bit input word-length is not feasable, but we have included it to illustrate the accuracy improvements at that precision.

Accuracy Criterion

A measure of the truncation can be obtained by computing the signal to noise (SNR) in the following way

$$SNR = 20 \log \frac{I(x,y)}{|O(x,y) - I(x,y)|} \tag{5.1}$$

where $I(x,y)$ and $O(x,y)$ are the input and output pixel intensity values at position (x,y). The SNR is obtained for x,y between 1 and 8. To study the

truncation noise at any particular pixel position (corresponding to a certain channel), one may aggregate the SNR statistics over the 4096 block for each pixel position separately. We have computed the averages of the pixel position SNR and this is tabulated in Table 5.2 as the 'Average SNR'.

If the input and output pixel values are identical, then SNR as defined cannot be computed. We keep track of the number of such occurrences and this is indicated in the 'Comments' column. If this number is high (a sizable fraction of 4096), then it means that the true SNR is considerably better then what is shown in the SNR column. This is because, the 'Average SNR' is computed only for the differing pixel values and gives a pessimistic estimate.

Peak SNR [7] is defined as in equation 5.1 with I(x,y) in the numerator, being replaced by the maximum value of the input data. The justification being that it is the truncation errors due to the architecture which is being estimated. PSNR usually yields higher values than the 'Average SNR'.

5.2 Timing Analysis

The timing analysis of the critical path modules like ROM and adder is discussed here. Line delays are also estimated. Spice is the work-horse for the timing simulations. Crystal is also used for preliminary propagation delay estimates in the case of the adder submodule.

The Spice source for the critical path modules is obtained by converting the extracted circuit into Spice format. The extraction style (1.2μ or 2.0μ) determines the parameters for parasitic resistance and capacitance. For the MOS transistor models, we have used MOSIS level 2 and level 3 parameters.

5.2.1 Adder

We can expect the longest delay in the adder to be the signal propagated from carry-in to the MSB. In our case, as carry-in is always preset to 0/1, the longest delay is from the LSB to the MSB. Spice simulations are performed and the output graphs are plotted in Fig. 5.2 and Fig. 5.3, corresponding to 1.2μ and 2.0μ technology simulations.

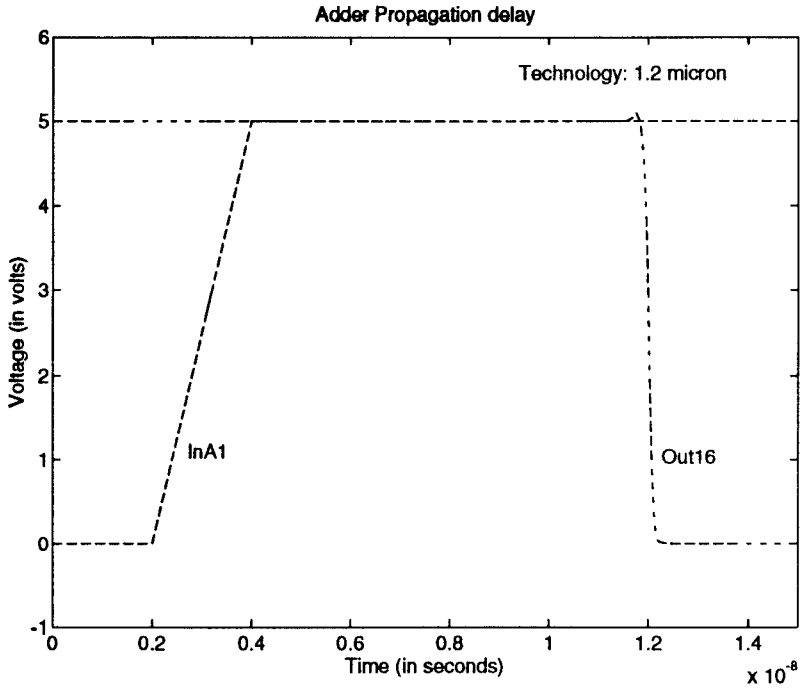


Figure 5.2: 1.2μ Adder

The propagation delays are also tabulated in Table 5.3. The adder inputs are $FFFF_{hex}$ and 0000_{hex}. 'b1' is pulsed from 0 to 5 volts in 2 nano-seconds. 's16' drops to 0 after the propagation delay.

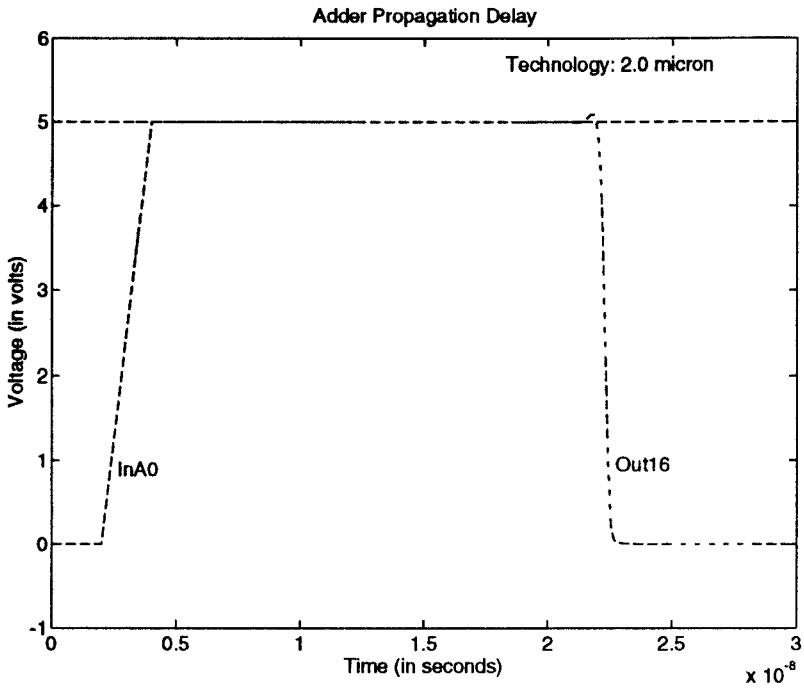


Figure 5.3: 2.0μ Adder

Technology	Propagation Delay
1.2μ	9 ns
2.0μ	19.3 ns

Table 5.3: Adder Propagation Delay

5.2.2 ROM

The timing analysis for ROM lookup-table extracted using the 2.0μ technology scaling parameters (of Magic) is shown in Fig. 5.4. The ROM input is pulsed to 5V at 3ns and the clock, at 5ns. The 'Row select' line is the output of the

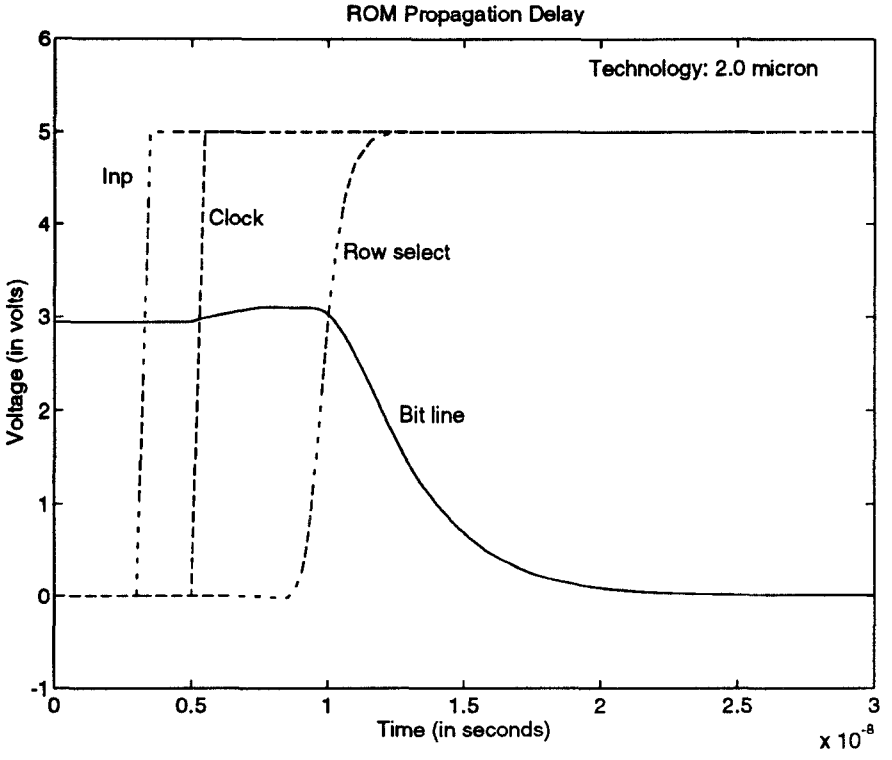


Figure 5.4: 2.0μ ROM

6-bit decoder which selects one of the words in the ROM table. The propagation delay is defined to as the time difference between the 50% points of the input and the 'Row select' lines. The bit line is at 2.9 volts, and it discharges to 0.5 volts at 16.5 ns. The propagation delay for the ROM lookup is 13.5ns.

Timing simulations for 1.2μ have failed for this module. This could be be-

cause the 'MOS fix for Level 3' which was patched into 'spice3e2' does not function for such a large number of transistors (about 6000). The adder module for which we were able to perform simulations had 700 transistors only.

To solve this problem, I reduced the transistor count in the ROM by discarding all but the essential table entries, reducing it to 1100 transistors. 'Spice3e2' however, still did not work. To solve this, we need to use either 'spice3f3' or 'Hspice'. We can be sure that the lack of this simulation will not cause any timing problems as its propagation delay is less than the adder-delay.

5.3 Clocking

A two-phase clocking scheme is used as it permits simple submodule design. In a traditional two-phase clocking scheme, one phase would be used for evaluation and in the next phase, the results would be latched in (see Fig 5.5).

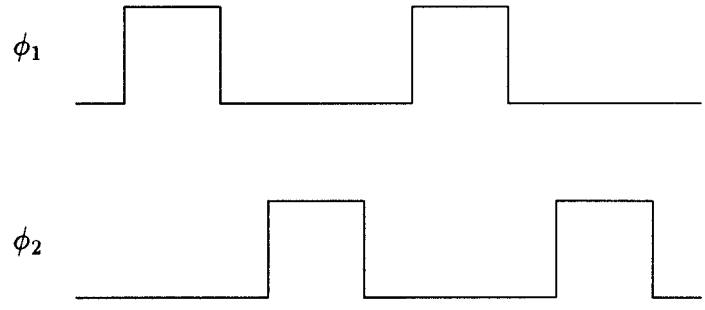
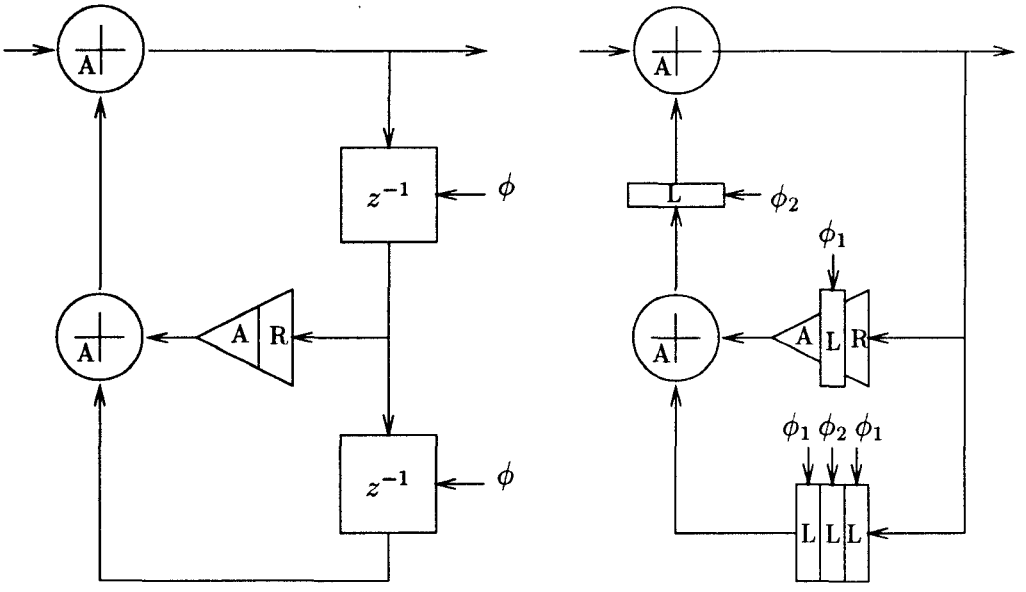


Figure 5.5: Two Phase Clock

Speeding up

By careful placement of the half-latches which latch intermediate data, the advantages of using a 2-phase scheme over a single phase becomes clear—we achieve clocking at twice the speed that was possible before. This is illustrated in Fig. 5.6. In the placement of our latches, we have taken advantage of the fact that propagation delays of ROM lookup and the adder (the critical-path modules) are about the same.



(a) Single phase clock

(b) Two-phase clocking scheme

Figure 5.6: Clock Speedup

With such fully pipelined and latched circuits, the clock speed is determined by the longest delay between any two subsequent latches. In Fig. 5.6 (a), the

propagation delay between two subsequent latches are (3A+R, 2A), where A and R stand for the Adder and ROM propagation delay. The maximum delay is (3A+R) and this determines the clock rate. Now, the same signal flow graph is retimed as shown in Fig. 5.6 (b). The maximum delay now is either 2A or (A+R), which is half of what we had before. Thus by careful placement of half-latches, a 100% speedup is obtained. The details of the half-latch placement in the 1-D IIR signal flow graph is shown in Fig. 4.13

5.4 Functionality Verification

IRSIM is used to verify the functionality of all modules. It is an event-driven switch-level logic simulator. It does not perform a detailed timing analysis, but by the same token, it is much faster and is useful for testing huge and complex module.

After the design of a module, the circuit is extracted and massaged into a format suitable for logic verification. Proper clock phases are set up within the simulation environment before testing with a suitable set of test-vectors. An example of this is shown below:

```
============= ROM logic simulation, IRSIM input ===============
clock phi 1 0
clock phib 0 1
stepsize 100ns
vector in in11 in10 in9 in8 in7 in6 in5 in4 in3 in2 in1 in0
vector am am15 am14 am13 am12 am11 am10 am9 am8 am7 am6 am5 am4
am3 am2 am1 am0
vector al al10 al9 al8 al7 al6 al5 al4 al3 al2 al1 al0
vector bm bm15 bm14 bm13 bm12 bm11 bm10 bm9 bm8 bm7 bm6 bm5 bm4
bm3 bm2 bm1 bm0
vector bl bl10 bl9 bl8 bl7 bl6 bl5 bl4 bl3 bl2 bl1 bl0
vector saout sa16 sa15 sa14 sa13 sa12 sa11 sa10 sa9 sa8 sa7 sa6
```

```
sa5 sa4 sa3 sa2 sa1
vector sbout sb16 sb15 sb14 sb13 sb12 sb11 sb10 sb9 sb8 sb7 sb6
sb5 sb4 sb3 sb2 sb1

w sbout
w saout
w in

set in 000000000000
c
set in 000000000001
c
set in 111111111111
c
set in 000000000010
c
set in 111111111101
c
set in 000100000000
c
set in 000000000100
c
set in 000010000000
c
set in 000011111111
c
set in 000000010000
c
========================= IRSIM output =====================
*** IRSIM version 8.6 ***
1450 nodes; transistors: n-channel=4932 p-channel=1338
parallel txtors:none
in=000000000000 saout=0000000000000000 sbout=0000000000000000
time = 200.0ns
in=000000000001 saout=0000000000001110 sbout=0000000000010000
time = 400.0ns
in=111111111111 saout=1111111111110010 sbout=1111111111110000
time = 600.0ns
in=000000000010 saout=0000000000011100 sbout=0000000000100000
time = 800.0ns
in=111111111101 saout=1111111111010110 sbout=1111111111010000
time = 1000.0ns
```

```
in=000100000000 saout=0000111000000110 sbout=0001000000000000
time = 1200.0ns
in=000000000100 saout=0000000000111000 sbout=0000000001000000
time = 1400.0ns
in=000010000000 saout=0000011100000011 sbout=0000100000000000
time = 1600.0ns
in=000011111111 saout=0000110111110111 sbout=0000111111110000
time = 1800.0ns
in=000000010000 saout=0000000011100000 sbout=0000000100000000
time = 2000.0ns
```

Such simulations are performed in a hierarchical fashion—first the submodules like ROMs, adders, latches, and multiplexers are designed and the logic is verified. Then the next level of modules such as 1-D channel are designed and tested. All these 1-D channels are put together to form the 1-D module, which is tested again. A similar logic-test procedure is used for testing the 2-D module, and lastly the whole chip is tested.

5.5 Summary

Simulations are an important phase of any design and development activity. In the VLSI chip design, different kinds of simulations are performed to test and verify various aspects of the system being designed. Architectural-level simulations are conducted to determine the chip parameters like bus precision based on accuracy criteria—speed and area considerations. Irsim, used for logic-level simulations verify the functionality of all modules and the complete chip. At lower level, timing analysis is performed on critical-path modules to ensure a high-speed design. A summary of the various simulations that were conducted during this project has been presented in this chapter.

Chapter 6

Conclusion

In this thesis, we have presented the VLSI implementation of a high-performance high-speed DSP chip which computes the two-dimensional discrete cosine transform and its inverse (2-D DCT/IDCT). It is a full-custom implementation employing a highly modular and hierarchical design strategy. The chip is based on the fully-pipelined time-recursive IIR structure which computes the forward and the inverse transform with minimum additional hardware.

Distributed arithmetic is used to implement fast and compact multipliers in the form of ROM lookup tables. Fast adders are implemented by cascading four 4-bit carry-lookahead adders in a ripple fashion. A non-overlapping two-phase clocking scheme is used to perform computations on both phases resulting in doubled throughput. This clock scheme also permits us to use simple dynamic latches for our implementation. Half-latches, delays, latched invertors, multiplexers, and buffers are some of the other modules which have been used in the 1-D and 2-D DCT/IDCT chip implementations.

Architectural simulations of the IIR structure are conducted to determine the system parameters. These simulations ensured a minimum PSNR of 40

dB which is required for most image processing applications such as video-conferencing, teleconferencing and high definition television (HDTV). Timing simulations which were performed using Spice indicate a clock frequency of 50 MHz corresponding to a data rate of 400 Mb/s. The 2-D DCT/IDCT chip dimensions are $24550\lambda \times 27094\lambda$ and its area is 240 mm^2 based on 1.2 μ technology. The pin-count is 176, and the chip has over 320,000 transistors. This chip has been submitted for fabrication in 1.2μ CMOS N-well double-metal single-poly technology.

We have implemented the chip using Magic. Though it is a very powerful tool, in this chip design, we have pushed it to its limits. In retrospect, if I were to do such a chip implementation, though I would use Magic to design all the low-level submodules, I would prefer some sort of high level synthesis tool for the automatic placement and routing, and which would also provide access to integrated debugging tools.

Another issue which came up while performing architectural simulations was variable bit allocation for different channels. In this implementation, we have arrayed our 1-D channel modules to compute all of the 8 DCT coefficients with the same precision. At the cost of increased design time, one could perform simulations to determine required channel bit precision for each coefficient and custom-design all the channels. This would lead to a smaller chip implementation without any loss in accuracy.

Bibliography

[1] C. T. Chiu and K. J. R. Liu, "Real-time parallel and fully pipelined two-dimensional DCT lattice structures with applications to HDTV systems," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 2, pp. 25–37, March 1992.

[2] F. Kretz and D. Nasse, "Digital television: Transmission and coding," *Proceedings of the IEEE*, vol. 73, pp. 575–591, April 1985.

[3] G. Tonge, "Image processing for higher definition television," *IEEE Trans. Circuits and Systems*, vol. CAS-34, pp. 1385–1398, November 1987.

[4] G. K. Wallace, "The JPEG still picture compression standard," *Communications of the ACM*, vol. 34, pp. 31–44, April 1991.

[5] K. R. Rao and P. Yip, *Discrete Cosine Transform : Algorithms, Advantages, and Applications.* Academic Press, Inc., 1990.

[6] H. Fujiwara, M. Liou, and M. Sun, "An all-ASIC implementation of a low bit-rate video codec," *IEEE Trans. Circuits and Systems for Video Techn.*, vol. 2, pp. 123–134, June 1992.

[7] M.-T. Sun, T.-C. Chen, and A. M. Gottlieb, "VLSI implementation of a 16x16 Discrete Cosine Tranform," *IEEE Trans. Circuits Syst.*, vol. 36, pp. 610–617, April 1989.

[8] D. Slawecki and W. Li, "DCT/IDCT processor design for high-data rate image coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 2, pp. 135–146, June 1992.

[9] L. B. Jackson, J. F. Kaiser, and H. S. McDonald, "An approach to the implementation of digital filters," *IEEE Trans. Audio, Electroacoustics*, vol. AU-16, pp. 413–421, September 1968.

[10] M. Vetterli and A. Ligtenberg, "A discrete fourier-cosine transform chip," *IEEE Selected Areas in Communications*, vol. SAC-4, pp. 49–61, January 1986.

[11] K. J. R. Liu and C. T. Chiu, "Unified parallel lattice structures for time-recursive Discrete Cosine/Sine/Hartley transforms," *IEEE Trans. Signal Processing*, vol. 41, pp. 1357–1377, March 1993.

[12] S. A. White, "Application of Distributed Arithmetic to Digital Signal Processing: A Tutorial Review," *IEEE ASSSP Magazine*, pp. 4–19, July 1989.

[13] G. Ma and F. J. Taylor, "Multiplier policies for digital signal proessing," *IEEE ASSP Magazine*, pp. 6–20, January 1990.

[14] K. J. R. Liu, C. T. Chiu, R. K. Kologotla, and J. F. JaJa, "Optimal Unified Architectures for the Real-Time Computation of Time-Recursive Discrete Sinusoidal Transforms," Technical Research Report, ISR, University of Maryland at College Park, no. 66, 1992.

[15] C. Stearns and P. Ang, "Yet another mulitplier architecture," *IEEE Custom Integrated Circuits Conference*, no. 24, pp. 6.1–6.4, 1990.

[16] H. T. Kung, "Why systolic architectures," *Computer*, pp. 37–46, January 1982.

[17] R. N. Mayo, M. H. Arnold, W. S. Scott, D. Stark, and G. T. Hamachi, *DECWRL/Livermore Magic Release.* DEC and WRL, Research Report 90/7, 1990.

[18] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design, A Systems Perspective.* Addison-Wesley, 1988.

[19] L. A. Glasser and D. W. Dobberpuhl, *The Design and Analysis of VLSI Circuits.* Addison-Wesley, 1985.

[20] C. T. Chiu, R. K. Kolagotla, K. J. R. Liu, and J. F. JaJa, "VLSI Implementation of Real-Time Parallel DCT/DST Lattice Structures for Video Communications," Technical Research Report, ISR, University of Maryland at College Park, vol. TR92-34, 1992.