AN ABSTRACT OF THE THESIS OF

Quanrong Li for the degree of Master of Science in Electrical and Computer Engineering presented on March 18, 1996. Title: Design and Performance Estimation of Two-dimensional Discrete Cosine Transform

Abstract approved:______________________________

Shih-Lien Lu

A VLSI system for image compression based on two-dimensional discrete cosine transform (2-D DCT) is designed and its performance is estimated. The focus is mainly on the reduction of power consumption and a reasonable speed. A 2-D DCT algorithm called row-column decomposition is chosen for the VLSI design of the system. Then a modified power-saving architecture is proposed based on the property and purpose of image compression. Several methods, including the use of low power library cells and low voltage (Vdd=1.5v), are used to achieve the goal of power reduction. Techniques that reduces power, such as ordering of input signals and common term sharing, are applied to the design of the system. These techniques and methods span from algorithm, architecture, logic style and circuit. In addition to using standard cells, some custom cells are also created. The control, timing and synchronization circuitry is detailed in the design of the system. HSPICE simulation shows that the designed 2-D DCT system can operate at more than 20MHz for 8 by 8 image blocks using 1.2u CMOS technology. Based on the effective switched capacitances provided by library cell data sheets, power consumption performance is estimated. The system consumes about 17mW at the maximum speed and the specified supply voltage. Comparisons to other implementations show that the designed system exceeds in power performance.
Design and Performance Estimation of Two-dimensional Discrete Cosine Transform

by

Quanrong Li

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirement for the
degree of

Master of Science

Completed March 18, 1996
Commencement June 1996
Master of Science thesis of Quanrong Li presented on March 18, 1996

APPROVED:

Redacted for Privacy

Major Professor, representing Electrical and Computer Engineering

Redacted for Privacy

Chair of Department of Electrical and Computer Engineering

Redacted for Privacy

Dean of Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Redacted for Privacy

Quanrong Li, Author
ACKNOWLEDGMENT

First, I would like to thank my research advisor professor Shih–Lien Lu for his support and guidance of this research. I found that advice is always available from him. This work could not have been done without his research fund. I would also like to thank professor David Allstot for his advice and kindness.

I would like to thank Chih–ming Chang for his friendly help with the editing of the thesis. He has made this work more efficient. I would not have been able to manipulate the word processor so skillfully without his assistance.

I would like to thank my fellow students Henry Chew , Che–jen Chang, and Wuucheng Huang. I got a lot of encouragement from them. Henry has been caring about my research progress since we become acquainted.

I would like to thank my wife sharon for her personal support, encouragement and consideration. As a researcher, an engineer, or a student, I need to put enormous amount of time working in Labs., companies, or libraries. She is always supportive whenever I need to go working. To her I dedicate this work.
# TABLE OF CONTENTS

1. INTRODUCTION ................................................................. 1

2. TWO-DIMENSIONAL DCT .................................................... 3
   2.1 Algorithm ................................................................. 3
   2.2 Architecture ............................................................. 4
   2.3 Summary ................................................................. 5

3. LOW-POWER DESIGN IN DCT ............................................... 7
   3.1 Architecture Level Power Reduction .................................. 7
   3.2 Circuit Level Power Saving ............................................ 9
   3.3 Conclusion ............................................................. 11

4. LOGIC/CIRCUIT DESIGN .................................................... 12
   4.1 Adder ................................................................. 12
   4.2 Multiplier ............................................................ 17
   4.3 Transposition RAM .................................................. 19
   4.4 System Synchronization .............................................. 22
   4.5 System Schematic Diagram ......................................... 26
   4.6 Summary ............................................................. 28

5. SIMULATION RESULT AND PERFORMANCE ESTIMATION .............. 29
   5.1 Functional Simulation ................................................ 29
   5.2 Critical Path Simulation ............................................. 31
   5.3 System Performance .................................................. 34
   5.4 Comparison to Existing Implementations ........................... 36

6. CONCLUSIONS ................................................................. 39
   5.1 Summary ............................................................. 39
   5.2 Lessons Learned ..................................................... 40
TABLE OF CONTENTS (Continued)

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.3 Future Work</td>
<td>40</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td>41</td>
</tr>
<tr>
<td>APPENDICES</td>
<td>43</td>
</tr>
<tr>
<td>Appendix A Video/Image Compression</td>
<td>44</td>
</tr>
<tr>
<td>Appendix B Two-Dimensional DCT Algorithms and Architectures</td>
<td>47</td>
</tr>
<tr>
<td>Appendix C Additional Schematic Diagrams of the System</td>
<td>49</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Architecture Based on the Algorithm</td>
<td>6</td>
</tr>
<tr>
<td>3.1 Energy Distribution Before and After DCT</td>
<td>8</td>
</tr>
<tr>
<td>3.2 Illustration of DCT and Quantization</td>
<td>8</td>
</tr>
<tr>
<td>3.3 Partitioning of Low Frequency Area</td>
<td>10</td>
</tr>
<tr>
<td>4.1 Block Diagram of the 20-Bit Carry-Select Adder</td>
<td>13</td>
</tr>
<tr>
<td>4.2 Schematic of 20-Bit Carry-Select Adder</td>
<td>14</td>
</tr>
<tr>
<td>4.3 Simulation Result of 20-Bit Carry-Select Adder</td>
<td>16</td>
</tr>
<tr>
<td>4.4 Schematic for Multiplier A (MULTA)</td>
<td>18</td>
</tr>
<tr>
<td>4.5 The Memory Cell</td>
<td>20</td>
</tr>
<tr>
<td>4.6 The Transposition Memory</td>
<td>21</td>
</tr>
<tr>
<td>4.7 The Read–Write Timing of Memory</td>
<td>22</td>
</tr>
<tr>
<td>4.8 The Memory Access Control Circuits</td>
<td>23</td>
</tr>
<tr>
<td>4.9 Circuits for System Timing</td>
<td>25</td>
</tr>
<tr>
<td>4.10 Schematic for the 2-D DCT</td>
<td>27</td>
</tr>
<tr>
<td>5.1 Schematic for Logic Simulation of a Data Path</td>
<td>30</td>
</tr>
<tr>
<td>5.2 Schematic for Critical Path Simulation</td>
<td>30</td>
</tr>
<tr>
<td>5.3 Schematic for Simplified Critical Path Simulation</td>
<td>32</td>
</tr>
<tr>
<td>5.4 Simulation Result of the Simplified Critical Path</td>
<td>33</td>
</tr>
<tr>
<td>5.5 Normalized Power Consumption</td>
<td>38</td>
</tr>
</tbody>
</table>
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 The 7 Constants in the Multiplication</td>
<td>10</td>
</tr>
<tr>
<td>4.1 Truth Table for Control and Synchronization Circuits</td>
<td>24</td>
</tr>
<tr>
<td>5.1 Power Dissipation Breakdown in the 2-D DCT System</td>
<td>35</td>
</tr>
<tr>
<td>5.2 Power Consumption of Different 2-D DCT Circuits</td>
<td>36</td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>C.1 Multiplier B</td>
<td>50</td>
</tr>
<tr>
<td>C.2 Multiplier C</td>
<td>51</td>
</tr>
<tr>
<td>C.3 Multiplier D</td>
<td>52</td>
</tr>
<tr>
<td>C.4 Multiplier E</td>
<td>53</td>
</tr>
<tr>
<td>C.5 Multiplier F</td>
<td>54</td>
</tr>
<tr>
<td>C.6 Multiplier G</td>
<td>55</td>
</tr>
<tr>
<td>C.7 Address Decoder Part 1</td>
<td>56</td>
</tr>
<tr>
<td>C.8 Address Decoder Part 2</td>
<td>57</td>
</tr>
<tr>
<td>C.9 Carry Select Adder Bitslice LSBEVEN</td>
<td>58</td>
</tr>
<tr>
<td>C.10 Carry Select Adder Bitslice LSBODD</td>
<td>59</td>
</tr>
<tr>
<td>C.11 Carry Select Adder Bitslice MSBEVEN</td>
<td>60</td>
</tr>
<tr>
<td>C.12 Carry Select Adder Bitslice MSBODD</td>
<td>61</td>
</tr>
</tbody>
</table>
DESIGN AND PERFORMANCE ESTIMATION OF TWO-DIMENSIONAL DISCRETE COSINE TRANSFORM

1. INTRODUCTION

Data compression, especially image/video compression, has found it's wide use in many applications such as teleconferencing, videophones, progressive image transmission, digital image storage, and videotex etc.. Discrete cosine transform (DCT), as one of the compression techniques, has been widely used in applications including the standards JPEG of ISO for still picture compression [1], H.261 of CCITT for audio–visual services at PX64 KBps, and MPEG of ISO for multimedia applications [2]. There are several advantages in using DCT as a compression method. First, DCT has energy packing capability and also approaches the statistically optimal transform (Karhunen–Loeve Transform or, KLT). Second, it can be effectively computed with fast algorithms because of its separable property. However, due to the intensive computation brought by the sheer numbers of DCTs calculation needed in many applications, dedicated VLSI implementation of DCT is necessary. A variety of VLSI implementations are reported in recent years [3–15], with most of them using algorithms based on matrix factorization and architectures based on distributed arithmetic (DA). The process technology used spans from 2um to 0.5um and the throughput reached as high as 200Mpels/sec which is sufficient for most applications [16].

Recently, portable multimedia and PCS (personal communication services) are becoming prevalent. Moreover, the integration level of VLSI is growing.

The switching power consumption of CMOS, the dominant technology in VLSI circuits, can be described as:

\[ P = \text{Prob.} \times C \times Vdd \times Vswing \times f \]

In this expression, Prob. is the probability of a transition of the capacitive load of the circuit C, Vdd is the supply voltage, Vswing is the voltage swing during the transition, and f is the clock speed of the circuit. The implementation reported in [15] achieved lower power consumption through voltage swing reduction while the design in [17] used a special algorithm to half the operating frequency.

The low–power design of a 2–D DCT developed in this thesis minimizes the power consumption through a reduced supply voltage of 1.5V. There are many problems, however, when the supply voltage is reduced to this level. We compare several designs at the circuit and logic level to counter the problems. We utilize the pipeline technique to compensate for long delays associated with low supply voltage to maintain system performance. At the
circuit level, we compared a hard–wired multiplier and a ROM–lookup multiplier in terms of power consumption. We also compared a few adders for power optimization. A two–dimensional DCT is design based on the analysis result. The technology used in this thesis is 1.2μ CMOS. The logic style chosen is static CMOS. The thesis contains 6 chapters. Chapter 2 describes the DCT algorithm and architecture adopted in this work. The low power design is presented in Chapter 3. Chapter 4 details the comparison and analysis of circuit blocks used to build the system. Chapter 5 presents the simulation result and the comparison to other implementations reported. At last, we draw some conclusion in Chapter 6. Appendix A provides some background knowledge in image/video compression and the role of DCT in the compression scheme. Appendix B gives a description of DCT algorithms and architectures. The last appendix (C) gives schematics that are similar but not so important to put in the main body of the thesis.
2. TWO-DIMENSIONAL DCT

Appendix B gives an overview of 2-D DCT algorithms and architectures reported in the literature. To briefly explain what DCT is and what it does, we can review the concept of discrete fourier transform (DFT). DFT transforms a sequence of time-domain signals into frequency-domain signals. By that transform, we can analyze the frequency components in a time-domain sequence. Similarly, DCT transforms time-domain signals into DCT-domain signals which are similar to frequency components. The equations used to do these 2 transforms are very similar. There are several reasons [18] that DCT is more popular than all the others including DFT. DCT achieves a similar performance to optimum KLT and involves only real numbers. This chapter will focus on the algorithm and architecture adopted in this thesis work.

2.1 Algorithm

For a given spatial data sequence \( \{ X(i,j) ; i, j = 0,1,2, ...,N-1 \} \), the definition of 2-D DCT is:

\[
Z(k,l) = \frac{2}{N} \times C(k) \times C(l) \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} X(i,j) \times \cos\left(\frac{(2i+1)k\pi}{2N}\right) \times \cos\left(\frac{(2j+1)l\pi}{2N}\right) \tag{eq.2.1}
\]

for \( k=0,1,2, ...,N-1; \ l=0,1,2,...,N-1 \)

where \( C(m) = \frac{1}{\sqrt{2}} \) for \( m=0 \), otherwise \( C(m)=1 \).

It should be stated that some people have not included a constant \( 2/N \). We include this constant in (eq 2.1) because it makes the overall data values smaller and thus minimizes the number of 1's in the coefficient matrix.

Because of DCT's property of separability, the transform can be computed by performing \( N \) successive one-dimensional DCT's on the rows and then another \( N \) one-dimensional DCT's on the resulting columns. An \( N \)-point 1-D DCT can be calculated as follows:

\[
Y(m) = \sqrt{\frac{2}{N}} \times C(m) \times \sum_{i=0}^{N-1} X(i) \times \cos\left(\frac{(2i+1)m\pi}{2N}\right) \tag{eq.2.2}
\]

for \( m=0,1,2, ..., N-1 \)

where \( C(m) = \frac{1}{\sqrt{2}} \) for \( m=0 \), otherwise \( C(m)=1 \). \( Y(i) \) and \( X(i) \) are row vectors.

For an 8 by 8 1-D DCT, the above equation (eq 2.2) can be written in the following matrix format [12]:

\[
Y(m) = \begin{bmatrix}
\sqrt{\frac{2}{8}} & \sqrt{\frac{2}{8}} & \sqrt{\frac{2}{8}} & \sqrt{\frac{2}{8}} & \sqrt{\frac{2}{8}} & \sqrt{\frac{2}{8}} & \sqrt{\frac{2}{8}} & \sqrt{\frac{2}{8}} \\
0 & -\frac{\sqrt{2}}{2} & 0 & -\frac{\sqrt{2}}{2} & 0 & -\frac{\sqrt{2}}{2} & 0 & -\frac{\sqrt{2}}{2} \\
-\frac{\sqrt{2}}{2} & 0 & -\frac{\sqrt{2}}{2} & 0 & -\frac{\sqrt{2}}{2} & 0 & -\frac{\sqrt{2}}{2} & 0 \\
0 & \frac{\sqrt{2}}{2} & 0 & -\frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 0 & -\frac{\sqrt{2}}{2} \\
-\frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 0 \\
0 & -\frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 0 & -\frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} \\
\frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 0 \\
0 & \frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2}
\end{bmatrix} \times \begin{bmatrix}
X(0) \\
X(1) \\
X(2) \\
X(3) \\
X(4) \\
X(5) \\
X(6) \\
X(7)
\end{bmatrix}
\]
\[ Y(i) = X(i) \times M \]

\[
= \begin{bmatrix}
X_{i,1}X_{i,2}X_{i,3}X_{i,4}X_{i,5}X_{i,6}X_{i,7}
\end{bmatrix}
\times
\begin{bmatrix}
a & b & c & d & a & e & f & g \\
a & d & f & -g & -a & -b & c & -e \\
a & e & -f & -b & a & g & c & d \\
a & g & -c & -e & a & d & -f & -b \\
a & -g & -c & e & a & -d & -f & b \\
a & -e & -f & b & -a & -g & c & d \\
a & -d & f & g & -a & b & -c & e \\
a & -b & c & -d & a & -e & f & -g
\end{bmatrix}
\]  (eq.2.3)

where \( a = \sqrt{2/N} \times \cos(\pi/4) \), \( b = \sqrt{2/N} \times \cos(\pi/16) \), \( c = \sqrt{2/N} \times \cos(\pi/8) \), \( d = \sqrt{2/N} \times \cos(3\pi/16) \), \( e = \sqrt{2/N} \times \cos(5\pi/16) \), \( f = \sqrt{2/N} \times \cos(3\pi/8) \), \( g = \sqrt{2/N} \times \cos(7\pi/16) \).

A permutation matrix \( P \) can be inserted which is capable of rearranging the order of columns in the matrix \( M \). The above equation can then be changed to the following:

\[ Y(i) = X(i) \times M' \times P \]

\[
= \begin{bmatrix}
X_{i,1}X_{i,2}X_{i,3}X_{i,4}X_{i,5}X_{i,6}X_{i,7}
\end{bmatrix}
\times
\begin{bmatrix}
a & a & c & f & b & e & d & g \\
a-a & f & c & d & -b & g & -e \\
a & a & -f & c & e & g & b & d \\
a & a & c & -f & g & d & -e & b \\
a & a & c & -f & g & -d & e & b \\
a & a & -f & c & -e & g & b & d \\
a & a & f & -c & -d & b & g & e \\
a & a & c & f & -b & -e & -d & -g
\end{bmatrix}
\times P \]  (eq.2.4)

With the above arrangement, the permutation matrix \( P \) is:

\[
P = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0
\end{bmatrix}
\]

After the rearrangement of the matrix \( M \), the resulting matrix \( M' \) contains every coefficient once and only once in each column. We will see that this rearrangement simplifies the architecture and enables the use of unique power reduction scheme in the architecture.

### 2.2 Architecture

To create an efficient architecture, the coefficient matrix is permuted, as described in section 2.1. The coefficient matrix after the permutation has the following characteristics.
First, column 0 and 1 are independent of each other. They are also independent of other columns since they are made of only $I$ and $a$, while columns 2 and 3 only have $c, f$ and columns 4 –7 only have $b, d, e, g$. This results in a modular architecture which has a simple wiring complexity. Fig.2.1 shows the architecture [12]. In Fig.2.1, a row vector $Y(i)$ is computed using multipliers, muxes, accumulators, and registers. The elements of input vector $X$ are fed into the circuit one at a time (or one at each clock). For $i=8$, the 8 elements of $Y$ are computed simultaneously, finished in 8 clock cycles, and shifted out serially. To describe it, we refer to eq.2.1, where a row vector $X$ is multiplied by the coefficient matrix $M'$ to get $Y$. If we ignore the permutation matrix $P$, then the first element of $Y$ is the sum of all the elements of $X$ since each element in the first column of matrix $M$ is “1”. The second element of $Y$ is computed through some additions and subtractions of the elements of $X$, and then multiplied by constant $a$. In a similar way, we can construct other elements of $Y$ from elements of $X$. Permutation is done before the result goes into the parallel_in/serial_out register. The simplicity of the circuit architecture is that wiring is regular and local. The most important property is that each computation of an element of $Y$ is decoupled and independent of some multipliers. In Fig.2.1, computation of $Y(1)$ is only dependent on input $X(1) \sim X(8)$; computation of $Y(1)$ is only dependent on multiplier $a$; computation of $Y(2)$ and $Y(3)$ is only dependent on multipliers $c$ and $f$; and the computation of $Y(4) \sim Y(8)$ is only dependent on multipliers $b, e, d$.

2.3 Summary

Two-dimensional DCT can be achieved through 2 sets of one-dimensional DCT’s. By arranging the coefficient matrix in a particular form through a permutation matrix, the overall circuit architecture is greatly simplified. We adopt this particular architecture for our study in the low-power design approach. This particular architecture has 2 features: first, it enables us to explore some properties in compression for power reduction (as will be seen in the next chapter) and, second, it simplifies wiring complexity, which is good for manual layout.
Fig. 2.1 Architecture Based on the Algorithm
3. LOW-POWER DESIGN IN DCT

In recent years, much work has been done and reported in using VLSI to implement DCT [19]. These works concentrate mostly on optimizing the performance and silicon area. For example, the work reported in [4] has an estimated chip of 320MPe1/sec throughput, which is more than enough for today's applications such as MPEG-1 (about 3.5MPe1/sec) and HDTV (about 81Mpel/sec). However, only a few papers [15, 17] have addressed the power consumption optimization for DCT design. The work reported in [15] concentrated on low-swing differential logic macrocells to achieve lower power. The design reported in [17] achieved lower power by halving the operating frequency at the algorithm and architecture level. Both are for 1-D DCT.

This chapter presents the techniques that can be applied to the DCT architecture presented in Chapter 2. Several approaches to power reduction of DCT are discussed. In addition, a discussion on published papers related to power reduction of DCT is given. General low power design rules and techniques can be found in [20 – 27]. The work in this thesis concentrates on using low power cells and on exploiting the properties of DCT for power reduction. Moreover, the work concentrates on applying the techniques described in [22] to the system. We will review these innovative points individually in the following sections.

3.1 Architecture Level Power Reduction

The purpose of DCT is to compress data. In doing so, it actually packs the energy of a pixel–image. The energy packing property can be explained this way: suppose we have an image of 1000 by 1000 dots, and each dot is represented by a byte, which indicates the gray scale of the dot. The possibility is that each of these bytes can be as big as 255 in value (if unsigned), and that most often they are big numbers, meaning great energy when they are transmitted. Energy packing means that we could find a better way to represent the image, using only a few bytes of big values, and the rest of them could be bytes of very small values, which means less energy when they are transmitted. DCT is one way of serving this purpose. Figure 3.1 illustrates energy distribution before and after applying DCT. From this figure, we observe that energy is distributed all over the time-domain matrix before DCT is applied. It is packed in the upper left corner of the DCT-domain after the transform. Very little energy has been left on the rest of the area of the matrix in the DCT-domain. Quantization is the next processing step following DCT. During quantization, the elements in the upper left corner, which correspond to the low frequency signal or structural property of an image, are quantized with fine steps. The rest of the elements are quantized with coarse steps.
Fig. 3.1 Energy Distribution Before and After DCT

(a) samples before DCT

(b) the matrix after DCT

(d) after DCT and quantization

(c) quantization table

Fig. 3.2 Illustration of DCT and Quantization
In order to understand the principle and properties of DCT and quantization so that we can exploit these properties for our power reduction, we further present an example in the next section. Fig.3.2 shows an arbitrarily selected 8X8 block of 8-bit samples from a real image [1]. The little differences among the pixel values indicate that the low frequency signal is dominant in time domain. Fig.3.2(b) is the matrix after transform. It indicates that only several coefficients corresponding to low frequencies are of large values; the rest are quite small. Fig.3.2(c) is a table of quantization steps for each element in Fig.3.2(b). Those large values in the lower right indicate coarse steps in quantizing high frequency elements in Fig.3.2(b). Fig.3.2(d) is the final quantized matrix. Comparing Fig.3.2(b) to Fig.3.2(d), we found the following:

1. Most elements in Fig.3.2(d) are zeroes.
2. No matter how different the values in the lower right corner of the matrix in Fig.3.2(b), they are all quantized to the same value, i.e., zero.

For these 2 reasons, the following questions are raised: why would we make so much effort to compute the values while in the end they will all be quantized to zeroes? Can't we save this power-consuming process and directly output zeroes? In other words, can we turn off those computation circuits and blocks related to computing those zeroes to conserve power consumption? With these questions in mind, we revisit the DCT architecture depicted in Fig.2.1. We observe that the computation paths of the Ys are relatively independent of each other. This property in the architecture has enabled the realization of the power saving concept. Since the architecture is of multiplier-accumulator style, we can turn off some accumulators and muxes that compute high frequency coefficients. When they are turned off, their corresponding Ys will retain the initial values of the accumulators when each row is computed. There are many partitioning methods to determine which elements can be considered as high frequency elements. Typical methods select a triangular, a rectangular, or a fan-shaped area as low frequency area. (Fig.3.3) To simplify the control circuit complexity, the rectangular partitioning is chosen in this thesis.

3.2 Circuit Level Power Saving

The circuit technique used to reduce power consumption is using minimum-sized devices. In addition, sharing common terms is also a good power saving method [22]. This is more applicable especially in signal processing application where multiplication with a constant is common. In our DCT architecture, input is multiplied by 7 constants every clock cycle. There is quite a lot of common term sharing in the circuit to be exploited.
Ordering of operation is also considered here to reduce the switching activity. In our case, there are many ordering issues to be concerned about in a shift–and–add operation in multiplication.

First, let us take a look at the 7 constants given in Table 3.1. They are represented in radix–2 signed–digit format [11]. Every input is multiplied by these 7 constants. They have several common terms which may be shared with one another, if shift registers are to be used to obtain them.

Table 3.1 The 7 Constants in the Multiplication

<table>
<thead>
<tr>
<th>coefficient</th>
<th>decimal value</th>
<th>signed–digit representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0.35355</td>
<td>$2^{-2} + 2^{-4} + 2^{-5} + 2^{-7} + 2^{-9}$</td>
</tr>
<tr>
<td>b</td>
<td>0.49039</td>
<td>$2^{-1} - 2^{-7} - 2^{-9} + 2^{-13}$</td>
</tr>
<tr>
<td>c</td>
<td>0.46194</td>
<td>$2^{-1} - 2^{-5} - 2^{-7} + 2^{-10}$</td>
</tr>
<tr>
<td>d</td>
<td>0.41573</td>
<td>$2^{-2} + 2^{-3} + 2^{-5} + 2^{-7} + 2^{-9} - 2^{-12}$</td>
</tr>
<tr>
<td>e</td>
<td>0.27779</td>
<td>$2^{-2} + 2^{-5} - 2^{-8} + 2^{-11}$</td>
</tr>
<tr>
<td>f</td>
<td>0.19134</td>
<td>$2^{-3} + 2^{-4} + 2^{-8} - 2^{-14}$</td>
</tr>
<tr>
<td>g</td>
<td>0.09755</td>
<td>$2^{-4} + 2^{-5} + 2^{-8} - 2^{-13}$</td>
</tr>
</tbody>
</table>

From the table we see the following common terms: $a, b, c, d$ have common term \(x \times 2^{-7}\); $e, f, g$ all have \(x \times 2^{-8}\) as common term, etc. When we do the multiplication, we can use
shift register to obtain the common term, and then distribute it to all the multipliers that need it.

To further conserve power consumption, we may use a relatively large buffer and wiring scheme (Fig. 4.4) to obtain all the common terms, instead of using shift registers.

Ordering of operation is also applied and used throughout the design of 7 multipliers: the smaller values are first added together before the result is added to the larger value. This pattern is repeated throughout the shift-and-add process in the 3 pipeline stages (Fig. 4.4).

3.3 Conclusion

The unique property of compression and quantization is to represent an image with a few numbers of big values and many numbers of zero values. Because of this property, we can conserve power by turning off the circuit blocks that are related to computing those zeroes. This is considered as an architectural level power reduction. At circuit level, common term sharing and ordering of operations can be applied to the multipliers to conserve power. These techniques will be implemented in the design of the whole system and in the design of multipliers.
4. LOGIC/CIRCUIT DESIGN

We designed the complete 2-D DCT using Viewlogic's Powerview tool set. Within this tool set, there are some standard cell libraries (STD2_3 and DPPLP) [25] available. We took cells from these standard cell libraries for design capture. We also created some special full-custom cells which are integrated into the system. The particular cell libraries used are BLP (same as STD2_3) and DPPLP which are produced by Berkeley. These cells are optimized for low power operation with delay information simulated at Vdd=1.5V.

With the given architecture described in chapter 2, we need the following blocks such as multipliers, muxes, accumulators, shift registers, and RAM to build the DCT system. We implement the multipliers by using add-and-shift technique. Therefore, the main computation is addition. We then need to design the adder efficiently. For the adder, the width must be 22 bits in order to meet the stringent error requirement [11]. This is because error is caused by finite word length and error propagation [11]. The critical path is most likely going to be determined by the adder. It also determines the operating speed of this DCT design. We will discuss the design of the adder first, followed by multipliers and muxes.

4.1 Adder

In the 2-D DCT system, the inputs to the adders are 15–22 bits wide, thus ripple-carry adders will render unacceptably long delay. Tree adders, on the other hand, require a relatively larger area, and are only suitable for addition with more than 32 bits. Other feasible fast addition techniques are carry-lookahead, carry-skip, carry-save and carry-select. The standard cells in [25] include general ripple-carry and carry-select adder cells. Since the adder in our system is a significant component and it determines the operating speed, it is custom designed based on the carry-select adder in the standard cell library.

There are 4 different adders: 22-bit, 20-bit, 17-bit and 15-bit in our design. We illustrate the design and analysis of a 20-bit adder as an example. The rest of the adders can be built in a similar way. The 20-bit adder shown in Fig. 4.1 is a carry-select adder which consists of 4 adder blocks of 3, 4, 6, 7 bits in each block. The number of blocks and the number of bits in each block are determined through delay estimation in the carry and sum paths. The partition of groups and bits determines the optimal delays of the carry and sum paths. The DPPLP library documents give the delay estimates for muxes as well as for bit slices of the adder. We base our design on the delay information provided. The schematic of our design of a 20-bit adder is shown in Fig.4.2. It has 4 adder blocks in...
Fig. 4.1 Block Diagram of the 20-bit Carry-Select Adder
There are 4 different types of bit slices in the cell library which can be used to construct each adder block. These bit slices are: CSALSBEVEN, CSALSBODD, CSAMSBEVEN and CSAMSBODD. For each adder block in Fig.4.2, from bottom to top, the order is LSB—>MSB, and, therefore, the LSB slice (CSALSBEVEN or CSAMSBODD depending on parity of the bit position) is used for the bottom bit slice. The rest of the bits in each block are CSAMSBEDVEN or CSAMSBODD depending on the parity of the respective bit position. For each block, there are block—carry—in and block—carry—out signals. Block—carry—out of a block serves as block—carry—in of the next block. The schematic diagrams for the 4 types of slices are included in Appendix C. Readers interested in the detail are also referred to the DPPLP document [25].

The adder appears small in size, but its HSPICE file is rather big. The HSPICE simulation file of the adder is generated from the Viewlogic tool called “spicelink”. Fig.4.3 is the result obtained from HSPICE simulation. It shows the corresponding node names in HSPICE simulation of MSB sum terms of each block, the sum term of the MSB of the 20—bit adder, the carry—out term and the inputs. They are:

- V(a0) input A0, representative of the A[0:19] input
- V(SUM2) sum of bit 2, the MSB of the first adder block
- V(SUM6) sum of bit 6, the MSB of the second adder block
- V(SUM12) sum of bit 12, the MSB of the third adder block
- V(SUM19) sum of bit 19, the MSB of the fourth adder block
- V(N6) the carry—out of the 20—bit adder

From our analysis, the possible critical paths are:

Path 1: bit0 —> bit1 —> bit2 —> 4—bit adder Cblkout delay —> 6—bit adder Cblkout delay —> 7—bit adder Cblkout delay —> xor (sum of 7—bit adder).
Variable: Cout0, 1 for bit0—2; Cblk propagation for 3 blocks, sum.

Variable: Carry—out of each bit position rippling from bit13 —> bit18, then sum19 (through XOR).

We used the following inputs for the critical path simulation: A[19—0] = FFFF, B[19—0]=00001, Carry—in=0, together with proper load capacitance on SUM nodes and the CARRY—OUT node. This set of inputs manifests both paths mentioned. The critical path turned out to be path 2. From Fig.4.3, we see that SUM18 is stabilized last, giving a worst case delay of 35ns.
SIMULATION RESULT OF 20-BIT CARRY-SELECT ADDER

Fig. 4.3 Simulation Result of the 20-bit Carry-Select Adder
A work could be done later to regroup bits into different number of groups so that the 2 paths have the same delay.

4.2 Multiplier

Although there exists many kinds of multiplier designs, simple and energy-efficient ones are not easily identified. In DSP applications, multiplication usually operates on an input variable and a constant. This property makes the shift-and-add type multiplier fit in such a situation since the constant can be represented in binary form and can be hard-coded in the circuit to implement the multiplication. Madisetti and Willson [11] discussed the trade-offs between a hard-coded multiplier and a ROM-based multiplier, and adopted hard-coded multiplier.

From Table 3.1, we can see that each multiplication involves three to six additions. If we do not pipeline these additions, the delay in the multiplier will be intolerable and the throughput will be low. Hence, we designed all seven multipliers with 3-stage pipelines.

For convenience, we call the multiplier that has the constant “a” as one of the inputs “MULTA”, we call the multiplier that has the constant “b” as one of the inputs “MULTB”, and so on. Fig.4.4 is the schematic for MULTA; schematics for MULTB—MULTG are included in Appendix C. We will use MULTA as an example in discussing the design of these multipliers.

Fig.4.4 has 3 stages. The computations in each stage are as follows:

Stage 1:  
\[ X4 + X5 \]  
\[ X2 + X3 \]  
\[ X1 \text{ is latched} \]

Stage 2:  
\[ (X4 + X5) + (X2 + X3) \]  
\[ X1 \text{ is latched} \]

Stage 3:  
\[ ( (X4 + X5) + (X2 + X3) ) + X1 \]

where \( X1 - X5 \) are the shifted version of input \( X \).

In order to conserve power, shifters are further replaced by wiring permutation directly from the input. When fan-out capability is considered, the input \( X \) can be buffered before being connected to the adders. In our design, the input goes through a buffer driver (a large buffer with a capacitive load of about 250fF).
Fig 4.4 Schematic for Multiplier A (MULTA)
Order of operation is carefully arranged in the multiplier. The dynamic ranges of \( X1 \), \( X2 \), \( X3 \), \( X4 \) and \( X5 \) are very different. \( X5 \) has the least range. The principle of ordering of operation is: small variables (which have small dynamic ranges) should add with other small variables in order to decrease switching activity. The dynamic ranges of these results are as follows:

\[
\begin{align*}
X1 & : \quad X \times 2^{-2} \\
X2 & : \quad X \times 2^{-4} \\
X3 & : \quad X \times 2^{-5} \\
X4 & : \quad X \times 2^{-7} \\
X5 & : \quad X \times 2^{-9} \\
X2 + X3 & : \quad X \times 2^{-3} \\
X4 + X5 & : \quad X \times 2^{-6}
\end{align*}
\]

In Fig.4.4, \( X4 \) and \( X5 \), the smallest two, are summed together. At the same time, \( X3 \) and \( X2 \) are summed. In the second stage, these two results are further summed together. In the last stage, the result from the second stage is summed with \( X1 \), the largest variable. This order of operation is the best arrangement because it minimizes the switching activity of these operations.

### 4.3 Transposition RAM

In the architecture adopted for this work, a 64-word transposition memory is needed. It is placed between the first dimensional DCT and the second dimensional DCT, as shown in Fig.2.2.

The memory used in [9] is a 3-transistor DRAM type cell. In our system, this 3-transistor cell cannot be used because of the lower supply voltage. The write transistor (the N type transistor) is supposed to pass both 0 and 1. This is usually not a big problem in systems with \( Vdd = 5V \) or \( 3V \), \( Vth \) of about 1V. But with a \( Vdd = 1.5V \) and \( Vth = 1v \) or so, it will not write “1” to cell correctly. As a remedy, we added a P-transistor in parallel to the N-transistor to make a transmission gate. The schematic of a single cell is shown in Fig.4.5. The addition of the P-tran makes the cell larger. Luckily, however, the memory is fairly small. As to refreshing of DRAM, since all the cells are read and written very frequently, no refresh circuitry is needed.

The overall architecture of the memory is as follows: the whole memory is organized in an array of 64 rows and one column. Each row in the array is a word, therefore each word has a length of 17 bits. The address decoder decodes the 6-bit address, and selects a row (a
word) out of 64 rows. The read–bit lines are precharged during clock low, and are read during clock high. There is no column decoder or column mux.

Fig. 4.6 shows the memory system. The upper left corner is the circuitry for precharge and logic inversion for read–lines and write–lines. The rest of the circuit in the figure are memory words. The array accepts read and write addresses from the address generator, and accepts data to be written from the data processing blocks. The only output is the data being read out in each cycle. It is important to remember that data are being written in and read out at the same time in each cycle. There are four different inputs and outputs for this memory block: read address, read data, write address, write data. The timing needs to be properly arranged. Fig. 4.7 illustrates the address timing for the write and the read address. The figure indicates that each address uses 1.5 clock periods for both read and write, which means that the address generation circuit needs to retain the address for an extra half cycle before it is
overwritten. Therefore, it is possible that in some clock cycles there are only one address, while in others there are 2.

Synchronization of the address and the data which come from the accumulators must be carefully taken care of. Since data will not be available until after 13 clock cycles (3 clock cycles delay in multiplexer), after 8 in an accumulator, 1 in parallel shifting-out, and 1 in the serial shifting-out registers, the first data is actually written into

---

**Fig. 4.6 The Transposition Memory**
the memory at the 14th clock cycle. This is the latency. The transposition of the data array is implemented by writing-in and reading-out with different address sequences. Assume sequence 1 and sequence 2 are:

Sequence 1: 0, 1, 2, 3, 4, 5, 6, 7, 8, ....
Sequences 2: 0,8,16,24,32,40,48,56,1,9,....

The memory is accessed in the following way:
1) write with sequence 1, and read with sequence 2;
2) after 64 data, write with sequence 2, and read with sequence 1;
3) after another 64 data, write with sequence 1, and read with sequence 2.

The pattern repeats after 128 cycles. Fig.4.8 is the schematic diagram showing the address generation, mode selection, synchronization, decoding, and write-read address overlapping. The mode selection signal can be obtained from the MSB of the 7-bit counter. In the figure, the counter is a 6-bit counter, and the mode selection signal can be generated from the carry-out of the counter.

The decoder is implemented in the following way: the 6-bit address goes through two 3-8 decoders, and the outputs are ANDed in an AND plane having 64 AND gates. The circuits for the 3-8 decoder and the AND plane are included in Appendix C.

4.4 System Synchronization

In the DCT computation using the architecture in Fig.2.2, there are several synchronization issues: (1) the outputs of the constant multipliers are multiplexed to different accumulators; therefore, mux control signals are needed; (2) the accumulators do add/subtract
Fig 4.8 The Memory Access Control Circuits
according to the sign in the DCT matrix, so a control signal for each accumulator is needed; (3) all accumulators need reset once every 8 clock cycles; (4) the results in the 8 accumulators are shifted out every 8 clock cycles; (5) because of pipelines in the memory address decoder and the multipliers, synchronization among them and the accumulator is needed; (6) in the second one-dimensional DCT, the same signals are needed. These synchronization and control signals are given in Fig. 4.9. In this figure, the 3-bit counter gives all the signals needed to generate the mux control signals and the add/subtract signals for the accumulators. The three TSPC (true single phase clock) registers give a delay of three clock cycles to keep the signal synchronized with the multipliers. Truth tables and logic equations for mux control and accumulator control are given in Table 4.1. The logic equation for the control signals are as follows:

Table 4.1 Truth Table for Control and Synchronization Circuits

<table>
<thead>
<tr>
<th>clock</th>
<th>0</th>
<th>000</th>
<th>001</th>
<th>010</th>
<th>011</th>
<th>100</th>
<th>101</th>
<th>110</th>
<th>111</th>
</tr>
</thead>
<tbody>
<tr>
<td>counter</td>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>clock</th>
<th>0</th>
<th>000</th>
<th>001</th>
<th>010</th>
<th>011</th>
<th>100</th>
<th>101</th>
<th>110</th>
<th>111</th>
</tr>
</thead>
<tbody>
<tr>
<td>mct0</td>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>mct1</td>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>mct2</td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>mct3</td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mct4</td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mct5</td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>mct6</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>mct7</td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>mct8</td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>mct9</td>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>add0ctrl</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>add1ctrl</td>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>add2ctrl</td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>add3ctrl</td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>add4ctrl</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>add5ctrl</td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>add6ctrl</td>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>add7ctrl</td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
loads of some signals:
q2: 4x; q1: 2x; q0: 5x; q1-bar: 1x.
mux c sel0: 3x; mux b sel0: 5x; mux b sel1: 4x
mux e sel1: 2x; mux d sel0: 2x

Fig. 4.9 Circuits for System Timing
Notes: * By exchanging the 2 inputs of mux F, the sel0 signal becomes the same as mux C's sel0.
**for mux control SEL0=MSB, SEL1=LSB.
MUX C SEL0 = Q1 ⊕ Q0
MUX F SEL0 = Q1 ⊕ Q0 (input IN0 and IN1 must be exchanged)
MUX B SEL0 = Q2 XOR Q0
MUX B SEL1 = Q2 XOR Q1
MUX E SEL0 = MUX B SEL1
MUX E SEL1 = NOT(MUX B SEL0)
MUX D SEL0 = NOT(MUX B SEL1)
MUX D SEL1 = MUX B SEL0
MUX G SEL0 = MUX E SEL1
MUX G SEL1 = MUX D SEL0
ADD0CTRL = 0
ADD1CTRL = MUX C SEL0
ADD2CTRL = MUX B SEL1
ADD3CTRL = MUX B SEL0
ADD4CTRL = Q2
ADD5CTRL = Q1−BAR * Q0 + Q2*Q1−BAR + Q2*Q0
ADD6CTRL = MUX B SEL0
ADD7CTRL = Q0

4.5 System Schematic Diagram

In previous sections, the adder, multiplier, memory, timing circuit and synchronization circuit are described. With all these components, together with TSPC (true single phase clock) registers, buffers, and generic 2–1 muxes, we constructed the 2–D DCT schematic diagram shown in Fig.4.10. The diagram shows the first dimension DCT and the second dimension DCT separately. Each of the DCTs contains the following circuit blocks: multipliers, XOR gates, muxes, adders, registers, shift registers and memory array. The circuits which generate control signals and synchronization signals are not included, owing to the size of the figure. They are described in Fig.4.8 and Fig.4.9. Buffers are added in the following locations: (1) before data enter multipliers and (2) before data enter memory array. Note
Fig. 4.10 Schematic for the 2-D DCT
that both of the one-dimensional DCTs use the same control and synchronization signals. To enable this and therefore simplify the control circuitry, 5 TSPC registers are added before the intermediate results enter the second one-dimensional DCT.

The second one-dimensional DCT is more simple than the first one. The paths which compute $Y(4)$, $Y(5)$, $Y(6)$ and $Y(7)$ are removed to reduce power consumption and to provide simpler circuits. The DRAM is on the lower left corner of the first one-dimensional DCT. Its address inputs are from the outputs of the memory access control circuits shown in Fig.4.8.

4.6 Summary

This chapter discusses the design of adders, multipliers, memory, and synchronization circuits of the 2-D DCT system. A 20-bit carry-select adder is implemented and simulation shows a delay of 35ns. Three-stage pipelined multiplier is designed and order of input signals are done to conserve power consumption. Because of low supply voltage, we modified traditional three-transistor DRAM. We designed a pipelined address decoder for the memory array to ensure proper access speed. Memory address generation and memory access timing circuits are also designed. We also designed a simple synchronization circuit for the 2-D DCT system. This synchronization circuit eliminates the need for two separate synchronization circuits for the two one-dimensional DCTs. The schematic of the whole 2-D DCT system is designed with these building blocks. The system schematic diagram has removed the unnecessary computation blocks to conserve power, as was proposed in Chapter 3.
5. SIMULATION RESULT AND PERFORMANCE ESTIMATION

This chapter gives reports on the simulation of the whole 2-D DCT system. As is known, functional simulation and timing simulation are usually performed after the completion of a schematic. Because of the simulation tools available, functional simulation is only partially performed. Timing simulation is done using HSPICE. Since HSPICE couldn’t handle the entire circuit design file, we simulate only the critical paths. Nevertheless, much simulation is performed on each of the components such as registers, muxes, adders, and XOR gates. The following sections give functional simulation results, timing simulation results, and the results on performance estimation. In the last section, a comparison between our implementation and other implementations is given.

5.1 Functional Simulation

At first, functional simulation is done on individual blocks to ensure logic correctness. The custom-designed components are all simulated at switch-level. We started with Viewlogic’s Viewsim tool. Since this tool does not support the transistor level switch-level simulation and our custom cells are built with transistors, we end up with a lot of unknown logic state. We then reduce the coverage of the functional simulation to only cover the critical path. Even on this critical path, there are still custom cells. We then solve this problem by simulating the custom cells with HSPICE and simulating the rest with Viewsim. Functional simulation verifies that the accumulator, which is a widely used block in the 2-D DCT system, works correctly. The schematic used is given in Fig.5.1.

The following data patterns are used in the logic simulation of the critical path:

Pattern 1:

\[ A = 01(H) \]
\[ ZIN = 0(H) \]
\[ CLK = \text{waveform } 1 0 1 0 1 0 \]

Pattern 2:

\[ A = 1f1f1(H) \]
\[ ZIN = 0(H) \]
\[ CLK = \text{waveform } 1 0 1 0 1 0 \]

These data are chosen at random. The clock is set to run 3 cycles. The outputs at the circuit blocks are as follows:
Pattern 1:

<table>
<thead>
<tr>
<th>4:1 mux</th>
<th>XOR</th>
<th>add</th>
<th>Reg1</th>
<th>2:1 mux</th>
<th>Reg2(OUT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycle 1: 00001</td>
<td>00001</td>
<td>00001</td>
<td>xxxxx</td>
<td>xxxxx</td>
<td>xxxxx</td>
</tr>
<tr>
<td>Cycle 2: 00001</td>
<td>00001</td>
<td>00001</td>
<td>00001</td>
<td>00001</td>
<td>xxxxx</td>
</tr>
<tr>
<td>Cycle 3: 00001</td>
<td>00001</td>
<td>00001</td>
<td>00001</td>
<td>00001</td>
<td>00001</td>
</tr>
</tbody>
</table>

Pattern 2:

<table>
<thead>
<tr>
<th>4:1 mux</th>
<th>XOR</th>
<th>add</th>
<th>Reg1</th>
<th>2:1 mux</th>
<th>Reg2 (OUT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycle 1: lflfl</td>
<td>lflfl</td>
<td>lflfl</td>
<td>xxxxx</td>
<td>xxxxx</td>
<td>xxxxx</td>
</tr>
<tr>
<td>Cycle 2: lflfl</td>
<td>lflfl</td>
<td>lflfl</td>
<td>lflfl</td>
<td>lflfl</td>
<td>xxxxx</td>
</tr>
<tr>
<td>Cycle 3: lflfl</td>
<td>lflfl</td>
<td>lflfl</td>
<td>lflfl</td>
<td>lflfl</td>
<td>lflfl</td>
</tr>
</tbody>
</table>

The logic function could be simulated more extensively within the system. However, since the CAD tool used could not support it, further simulation is expected to be done either with more advanced tools or with FPGA [28].

5.2 Critical Path Simulation

As we know, a system’s maximum clock frequency is determined by the longest delay path in that system. For small systems, a thorough simulation can be performed on the whole system to quickly find out the longest delay path (or critical path) with enough accuracy. For large systems, however, it is more practical to run simulations on only the possible longest paths called critical paths. These critical paths are extracted from all the paths of the system based on the delay information of each component on these possible critical paths. The delay information of the components are obtained either from library cell documents or from results of simulations performed on that component. In our work, we thoroughly simulated the custom cells that we have created using HSPICE, and obtained the delay information. We refer to the DPPLP and STD2_3 library documents [25] to get the delay information of the cells we used in our system. Based on the delay information, we analyze the paths in the system and pinpoint the possible paths that could have longest delay. After careful observation and estimation of our DCT system, it becomes obvious that one path is much longer than any other ones. This path is shown in Fig.5.2. It consists of 4:1 muxes, XOR gates, adders, and registers. Care must be given in determining the critical path to avoid human error. We made a mistake when we were first extracting the critical path. We included a 2:1 mux and a register, just like the one shown in Fig.5.1, and we discovered it during HSPICE simulation. In Fig.5.2, the delay of each individual component is as follows:
The minimum critical path delay, based on these data, is 45.4ns. This is obtained by summing all of the block delays in the path.

The HSPICE file for Fig. 5.2 is large, resulting in difficulties in simulation. We degraded the simulation by making just a bit-slice path as shown in Fig. 5.3. The Carry–Select adder slice is not included in the path because there are several slices, each with different delay. Furthermore, the carry–propagation can not be simulated in a bit–slice fashion. Our method is to add the delay of the adder, which we obtained in chapter 4, to the simulation result of Fig. 5.3. Fig. 5.4 is the simulation result for Fig. 5.3. It shows that the simplified path has the following delay characteristics:

- Rising–edge delay: 10ns
- Falling–edge delay: 11ns
- Average delay in the path: 10.5ns

The rising–edge delay is defined as the time interval between rising edge of input V(in) and rising edge of output V(out). The falling–edge delay is defined as the interval between the
Fig. 5.4: Simulation Results of the Simplified Critical Path
falling edge of input signal $V_{\text{in}}$ and the falling edge of output signal $V_{\text{out}}$. The average delay is the average of the rising–edge delay and the falling–edge delay. Now we can calculate the critical path delay:

$$\text{Delay of simplified path} + \text{Delay of CSA (adder)} = 10.5 + 35 = 45.5\text{ns}$$

This result is very close to the predicted result: $45.4\text{ns}$. The partitioning of the value, however, is very different. In the predicted one, the adder has a delay of $24\text{ns}$; the rest of them have a combined delay of $21.4\text{ns}$. Whereas in the simulation result, the adder has a delay of $35\text{ns}$, and the rest have a combined delay of only $10.5\text{ns}$. The reason may be that: for XOR, mux and TSPC register, the predicted delay is based on the delay data evaluated with heavy load ($32.5\text{fF}$), while in our system, the real load is smaller ($10-15\text{fF}$). As to the adder, the predicted delay is based on an average of a $32$–bit cell, while ours is a $20$–bit cell with a different grouping of bits.

### 5.3 System Performance

The main goal for our design of this 2-D DCT system has been power consumption. In this section, we will estimate the power of the system, starting from the components of the system. These estimates are based on the maximum clocking speed obtained in the above section and a power supply of $1.5\text{V}$.

Table 5.1 gives component names, number of each kind of component used, average switched capacitance of each component, and power dissipation of the components as well as the whole system. The number of transistors in each component and the number of transistors in the system are also included. From the table we made the following conclusions:

1. Multipliers are the main consumers of power. Each multiplier consumes $1\text{mW}$ to $2.2\text{mW}$, depending on the number of bits of the constant in the multiplier. (For example, Multiplier D has a constant of 6 bits: it consumes $2.2\text{mW}$).

2. Memory’s power is only a small portion of the overall power consumption ($5\sim10\%$). This means that our main focus for power reduction should be on other parts of the system.

The maximum lock frequency of the system is already estimated in the previous section. As to the silicon area of the system, more advanced CAD tools are needed in order to generate a layout and the area can then be predicted.
Table 5.1 Power Dissipation Breakdown in the 2-D DCT System

<table>
<thead>
<tr>
<th>Name</th>
<th>Property</th>
<th>quantity</th>
<th>Switched cap (fF, avg.)</th>
<th>Power *</th>
<th>Transistor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer</td>
<td>8-bit, large</td>
<td>1</td>
<td>43x8</td>
<td>15uW</td>
<td>8x4</td>
</tr>
<tr>
<td>Buffer</td>
<td>17-bit, large</td>
<td>1</td>
<td>43x17</td>
<td>33uW</td>
<td>17x4</td>
</tr>
<tr>
<td>Buffer20bit</td>
<td>driving ratio=64</td>
<td>1</td>
<td>20x244</td>
<td>220uW</td>
<td>20x12</td>
</tr>
<tr>
<td>Multa (2)</td>
<td>Adder</td>
<td>(4)</td>
<td>4x20x165</td>
<td>594uW</td>
<td>20x736</td>
</tr>
<tr>
<td></td>
<td>Reg.</td>
<td>(6)</td>
<td>6x20x67</td>
<td>362uW</td>
<td>6x220</td>
</tr>
<tr>
<td>Multb (2)</td>
<td>Adder</td>
<td>(3)</td>
<td>2x3x20x165</td>
<td>0.9mW</td>
<td>2x3x736</td>
</tr>
<tr>
<td></td>
<td>Reg.</td>
<td>(6)</td>
<td>2x6x20x67</td>
<td>723uW</td>
<td>2x6x220</td>
</tr>
<tr>
<td>Multc (2)</td>
<td>Adder</td>
<td>(3)</td>
<td>2x3x20x165</td>
<td>0.9mW</td>
<td>2x3x736</td>
</tr>
<tr>
<td></td>
<td>Reg.</td>
<td>(6)</td>
<td>2x6x20x67</td>
<td>723uW</td>
<td>2x6x220</td>
</tr>
<tr>
<td>Multd (2)</td>
<td>Adder</td>
<td>(5)</td>
<td>2x5x20x165</td>
<td>1.5mW</td>
<td>2x5x736</td>
</tr>
<tr>
<td></td>
<td>Reg.</td>
<td>(6)</td>
<td>2x6x20x67</td>
<td>723uW</td>
<td>2x6x220</td>
</tr>
<tr>
<td>Multe (2)</td>
<td>Adder</td>
<td>(3)</td>
<td>2x3x20x165</td>
<td>0.9mW</td>
<td>2x3x736</td>
</tr>
<tr>
<td></td>
<td>Reg.</td>
<td>(6)</td>
<td>2x6x20x67</td>
<td>723uW</td>
<td>2x6x220</td>
</tr>
<tr>
<td>Multf (2)</td>
<td>Adder</td>
<td>(3)</td>
<td>2x3x20x165</td>
<td>0.9mW</td>
<td>2x3x736</td>
</tr>
<tr>
<td></td>
<td>Reg.</td>
<td>(6)</td>
<td>2x6x20x67</td>
<td>723uW</td>
<td>2x6x220</td>
</tr>
<tr>
<td>Multg (2)</td>
<td>Adder</td>
<td>(3)</td>
<td>2x3x20x165</td>
<td>0.9mW</td>
<td>2x3x736</td>
</tr>
<tr>
<td></td>
<td>Reg.</td>
<td>(6)</td>
<td>2x3x60x67</td>
<td>723uW</td>
<td>2x6x220</td>
</tr>
<tr>
<td>Muxf201</td>
<td>2:1, 16bit (avg.)</td>
<td>15</td>
<td>15x16x67</td>
<td>0.9mW</td>
<td>15x16x14</td>
</tr>
<tr>
<td>Mux</td>
<td>4:1, 18-bit (avg.)</td>
<td>6</td>
<td>6x18x108</td>
<td>525uW</td>
<td>6x18x26</td>
</tr>
<tr>
<td>Adder</td>
<td>20-bit</td>
<td>8</td>
<td>8x20x165</td>
<td>0.8mW</td>
<td>8x736</td>
</tr>
<tr>
<td>Adder</td>
<td>15-bit</td>
<td>4</td>
<td>4x15x165</td>
<td>446uW</td>
<td>4x576</td>
</tr>
<tr>
<td>Register</td>
<td>20-bit w/ asyn rst</td>
<td>8</td>
<td>8x20x109</td>
<td>0.8mW</td>
<td>8x20x21</td>
</tr>
<tr>
<td>Register</td>
<td>17-bit, TSPCR</td>
<td>13</td>
<td>13x17x67</td>
<td>666uW</td>
<td>13x17x11</td>
</tr>
<tr>
<td>Register</td>
<td>15-bit w/ asyn rst</td>
<td>4</td>
<td>4x15x67</td>
<td>180uW</td>
<td>4x15x21</td>
</tr>
<tr>
<td>Register</td>
<td>12-bit, TSPCR</td>
<td>4</td>
<td>4x12x67</td>
<td>141uW</td>
<td>4x12x11</td>
</tr>
<tr>
<td>DRAM</td>
<td>Drive</td>
<td>(1)</td>
<td>714</td>
<td>32uW</td>
<td>85</td>
</tr>
<tr>
<td></td>
<td>Word</td>
<td>(64)</td>
<td>64x17x18</td>
<td>0.9mW</td>
<td>64x17x4</td>
</tr>
</tbody>
</table>
Note: Total Power: 16.95mW (calculation error is 5%). Total No. of transistors: 76K (Multa–Multg: 50,444, the rest: 25,582) (of the 50,444 transistors, 33,284 are from adders, 17,160 are from registers). Estimation condition: (1) The power is calculated at 1.5V, 20MHz. (2) Input capacitance is not included. (3) Clocking buffer capacitance is not included.

5.4 Comparison to Existing Implementations

There are many reported 2-D DCT circuits; they differ in algorithm, architecture, and logic style. Among these reported designs, only a few gave data on power consumption of their implementations. Among these, they further differ in technology, supply voltage, operating frequency, and block size (the block in a image–pixel matrix, on which DCT is performed). Table 5.2 lists the implementations that have reported power consumption. But none of the implementations gives a report on how the power consumption data is obtained, This also results in difficulty in comparison.

<table>
<thead>
<tr>
<th>Implementation</th>
<th>block size</th>
<th>frequency (MHz)</th>
<th>Vdd (V)</th>
<th>Technology (micron)</th>
<th>Power (W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[9]</td>
<td>16x16</td>
<td>14.3</td>
<td>5</td>
<td>2</td>
<td>0.36</td>
</tr>
<tr>
<td>[15]–1</td>
<td>8x8</td>
<td>200</td>
<td>3.3</td>
<td>0.5</td>
<td>0.35</td>
</tr>
<tr>
<td>[15]–2</td>
<td>8x8</td>
<td>100</td>
<td>2</td>
<td>0.5</td>
<td>0.15</td>
</tr>
<tr>
<td>our circuit</td>
<td>8x8</td>
<td>20</td>
<td>1.5</td>
<td>1.2</td>
<td>0.017</td>
</tr>
</tbody>
</table>

To make a comparison among them, we tried to normalize these results. This normalization process is very coarse because there is no accurate formula, and it is most often empirical. When we apply the scaling formula to normalize them, there are the following problems: (1) for voltage scaling, a circuit that operates at Vdd=5V may not function at Vdd=1.5V unless the circuit is modified or even redesigned, which usually means adding more circuits. Which may result in a higher switching capacitance; (2) a circuit that is immigrated to a different technology usually needs modification to the original circuit; and (3) circuits that operate at one frequency may not operate at the same frequency after it’s Vdd is
lowered. All these may cause inaccuracy in our scaling model. Even though our model may not give a perfect comparison, we believe it does show some merit in predicting approximate power consumption. The formulas we use to normalize these power consumption results are:

\[ S = \frac{W}{W'} \]
\[ U = \frac{V}{V'} \]
\[ P' = P \times \frac{S}{U^3} \]  \hspace{1cm} (eq 5.1)
\[ f' = f \times \frac{S^2}{U} \]
\[ PP' = P' \times \frac{f_{\text{ref}}}{f'} \]

where \( W' \), \( V' \) are the reference process and supply voltage. \( W \) and \( V \) represent the implementation’s process and supply voltage. \( P' \) and \( f' \) represent the power and frequency of an implementation after scaling. \( PP' \) represents the power of an implementation if throughput is further kept fixed, which means that comparison is done with the same throughput. The above equations mean that when process scales by \( 1/S \), voltage scales by \( 1/U \), then power will scale by \( S/U^3 \), and frequency will scale will scale by \( S^2/U \). The power will further scale by \( f_{\text{ref}}/f' \) if throughput is scaled to a reference throughput \( f_{\text{ref}} \). In our comparison, reference parameters are: \( W=0.5 \), \( V=1.5V \). \( f_{\text{ref}} \) is 117.65MHz, which is the frequency of our system after \( W \)–and–\( V \) scaling. With the above equation, we plug the raw data that are presented in Table 5.1 into the equation (eq.5.1). We obtained a normalized power consumption chart, as shown in Fig.5.5. This chart shows that our system has a very good power performance. It indicates that the power performance of our system is better than that in [9], almost the same as that in [15]–1, and better than that in [15]–2. We have the following explanation for the result of comparison: because there are several ways to calculate or measure power consumption, different methods will give different results even for the same circuit. So this comparison only gives us a rough idea of the approximate power consumption. The “list of order” is not significant. On the other hand, our system is much better than others in absolute power consumption. It is very useful in applications where absolute power is critical.
Fig. 5.5 Normalized Power Consumption
6. CONCLUSIONS

There are many different architectures, logics, and circuits that are used to implement two-dimensional discrete cosine transform (2-D DCT). In this thesis work, a two-dimensional discrete cosine transform circuit is developed. Our focus is on efficient power consumption with proper speed. Performance estimation shows that the scheme employed in the circuits can achieve a small power dissipation of 16.95mW and an operating speed of 20MHz.

5.1 Summary

Analysis and comparison are performed on the architectures of DCTs, then on the circuit components, in terms of their power consumption properties. Based on the result, a simple architecture is chosen. This architecture enables the exploitation of the unique property of DCT, which is energy packing for power reduction. The new architecture based on the data compression property eliminates the computation on coefficients which will be dropped by quantization that follows the DCT. Circuit level optimization includes using standard library cells which are optimized for low power at circuit and logic levels and sizing transistors of self-created components. Architectural level power reduction techniques such as ordering of input signals, minimizing glitching activity and pipelining are applied to the system. At the algorithm level, common term sharing, which minimizes the number of operations, is applied to the design of multipliers.

Circuits for timing and control are designed to perform certain operations specified in the DCT algorithm. These include mux control signals, accumulator control signals, memory access control signals, and the timing signals between memory address decoder, multiplier, adder and memory.

The resulting circuit is partially simulated for its functional correctness. Each component is individually simulated at both logic level and circuit level. Critical path simulation shows that the circuit can be operated at a maximum of more than 20MHz.

Estimation of power consumption is based on the effective capacitance of each component, which is given in cell library documents. The result shows that the system consumes only 16.95mW. Comparison between 3 other implementations is made which indicates that our implementation gives a pretty good result in power consumption.

The 2-D DCT system we have designed can find its application in laptop computers where power is a critical issue. Our design helps reduce power consumption in the image
processing portion. Personal communication devices with visual communication capability can also benefit from our system. In other applications such as multimedia, JPEG and MPEG, where DCT is a standard, our design can be used to meet the power consumption budget of the overall system.

5.2 Lessons Learned

The CAD tools available for use in this work include HSPICE and Powerview. Both tools have very obvious limitations. To handle a very large simulation file, a different tool becomes very necessary for efficient functional and circuit level simulation. We hope this can be done later on. To do a complete logic level simulation for functional verification, a better tool is essential to support the work.

5.3 Future Work

FPGA, as an approximate alternative to simulation and fabrication of VLSI design, can help verify and prototype our design in a very timely fashion. We are looking forward to transferring our design to the Xilinx chip [29] using the Xilinx library cells. Although speed can not be expected to be verified, it is still a good direction to go.

The design in this thesis work mainly focuses on DCT. In the commercial market, more flexibility in block size such as 4 by 4, 12 by 12, 16 by 16, and 32 by 32, and ability in color processing are demanded. In some situations, better error performance is also required, in which case, adaptive circuit architecture can be designed. The system could adapt to error requirements while turning off certain computing blocks to conserve power. The ability to perform IDCT (inverse DCT) and to process MPEG sequence is also important. There is a lot of work to be done in this direction, to make our chip more commercial. The design we have, up to now, is featured in power consumption. It can find its use in portable computing and in other applications in which power is critical.
BIBLIOGRAPHY


APPENDICES
Appendix A Video/Image Compression

For both storage and transmission purposes, image/video compression is so necessary. Many techniques are developed in the past, DCT is one of them. To help the readers understand more about compression, the techniques and criteria are described in this appendix, they are not very exact, but rather explanatory.

A.1 Simple Techniques

Run–length—when there are consecutive repeated values in a sequence of pixel values, these repeated values can be represented by the single value and a count of how many times to repeat that value (run length). It is lossless.

Vector quantization (VQ)—Instead of quantizing pixel values one at a time using corresponding quantization step size, the VQ refers to quantizing a sequence of pixel values at a time using a lookup table, that is, search in the lookup table and find a vector which has the least difference in elements from the vector being quantized. It is a many–to–one mapping. The compression results from the fact that only the index (small data) of the selected vector is transmitted provided that both sender and receiver has that lookup table. Obviously it is lossy since the selected vector in the table is only an approximation of the vector being quantized.

A.2 Interpolative Method

Subsampling—Especially in color components I,Q of YIQ format, the color components are subsampled to achieve a reduction of the bits needed. The values that were dropped are obtained through interpolation. This is so because human eyes are less perceptive to distortion of color signals. Subsampling can not be applied to luminance component and, it is lossy.

A.3 Predictive Methods

DPCM (differential pulse code modulation)—Based on the prediction that adjacent pixels or lines or even frames are highly correlated in general, one doesn’t need to send
the whole value of a pixel each time, but only need to send the difference of the value from the previous one. Since the difference is really small, less bits are needed, and the compression is therefore achieved. This is widely used in audio compression, rather than in video compression.

Motion compensation——The simplest way to predict the next frame in a video sequence is to say that it is the same as the last frame, then the pixel-by-pixel differences are zeroes which means no transmission needed, but this is not generally true except for still images. The reality is that, the objects in the image is usually in motion, and the pixel-by-pixel differences are not zero and might be statistically very large, which means more bits would be generated and have to be transmitted. However, the motion in most natural scenes is organized and can be represented as a local translation. The estimate of next frame, after taking into account the translation, is more accurate, and therefore the pixel-by-pixel difference will be smaller, which means less bits need to be transmitted. The compression is thus gained.

A.4 Compression via Transforms

Discrete cosine transform (DCT)——Similar to discrete fourier transform, DCT transforms a sequence of signals in time domain into a sequence of signals in DCT domain which is similar to well-known frequency domain. The beauty is that after DCT, only a few elements in the sequence are large values, most of them are nearly zero. This creates a chance for the run-length encoding to be used (the close-to-zero ones may be treated as zeroes and can then be efficiently run-length encoded).

Discrete wavelet transform (DWT)——Suppose $X(t)$ is a continuous-time signal whose frequency range is 10Hz–10KHz. During A/D conversion, the sampling frequency has to be 20KHz or above (Shannon’s sampling theory). One would ask such questions: is 20KHz too high for the 10Hz portion of the signal because it generates too many data that are more than necessary? can we use lower sampling frequency for the low-frequency portion? or in general, can we use multiple sampling frequencies for corresponding portion of the signal in order to minimize the data generated? Wavelet transform is just for answering these questions. The main idea is, for low-frequency part of a signal, the sampling rate is low (less samples), meaning that time resolution is low frequency resolution is made high (next sampling rate is very close to the previous one), while for the high-frequency part, more samples are obtained but the intervals between adjacent sampling rate is large (frequency resolution is low). The detailed formula can be find in [2]. In deed, DWT is a discrete representation of a continuous-time signal, and is therefore similar to the fourier series, but with different
resolution in both time-domain and frequency domain. The compression is achieved by using DWT filter bank and allocating less bits (even no nits) to the high-frequency part of the filter bank output when quantizing them since there is relatively little energy in the higher frequency subbands and most energy is packed in the lowest frequency subband. At this point, it is similar to DCT.

A.5 Statistical Techniques

Huffman encoding———This coding technique exploits the property of probability differences of event occurrence. Events which are more likely to occur are assigned shorter length codewords while those which are less likely to occur are assigned longer length codewords. A codebook is generated according to the assignment and, is referred to by both encoder and decoder. It effectively minimizes the average description length (bits) of events.

A.6 Model-Based Coding

Most of existing coding methods such as vector quantization, transform coding are based on information-theory in which image signals are considered as random signals. Compression in these methods are achieved by exploiting stochastic properties (such as correlation). Contrary to conventional coding methods, model-based coding represent image signals using structural image models, that is, it assumes a meaningful object, instead of just any signal combination, in the scene. One of the models is to consider an image as having structural features such as contours and textures. For coding, the encoder and the decoder both have the same set of models; the encoder analyzes input images and transmits mainly the indexes of the corresponding models, the decoder generates output images using the models. By using object models, less bits are needed to store and transmit images, and compression is achieved. This type of compression is only applicable to certain applications.
Appendix B Two-Dimensional DCT Algorithms and Architectures

B.1 Algorithms

The N-point 1-D DCT is:
\[
Z(k) = C(k) \sum_{i=0}^{N-1} X(k) \cos((2i+1)k \pi/2N)
\]
The 2-D DCT can be written as
\[
Z(k,l) = C(k)C(l) \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} X(i,j) \cos((2i+1)k \pi/2N) \cos((2j+1)l \pi/2N)
\]
for \( k=0,1,2, ..., N-1; \ l=0,1,2,...,N-1 \)
where \( C(m) = 1/\sqrt{2} \) for \( m=0 \) and \( C(m)=1 \) otherwise.

There are mainly 2 ways in implementing 2-D DCT. The first one, and the most famous one is to decompose it into 1-D DCT’s and then use the algorithms and architectures developed for the 1-D DCT [6-15], [17]. The second one is to implement the 2-D DCT directly via systolic array [3-5], via recursive decomposition into smaller 2-D DCT’s, or via other transforms such as 2-D DFT and 2-D DWT [18]. In practice, the decomposition into 1-D DCT is most frequently used, and we will concentrate on this approach.

Like DFT, there are many fast 1-D DCT algorithms. Most of them reduce the number of multiplications by exploiting the DCT matrix symmetries, resulting in flowgraphs that has several butterfly stages [6-8]. In VLSI implementation, the butterflies mean complex wiring. Because of the complexity incurred, most recent implementations use one stage of butterfly, and then use matrix–vector multiplication thereafter [6], [12].

B.2 Architectures

Programmable architectures——[16] listed 11 implementations of programmable and dedicated VLSI architectures for video coding applications. All of those has tailored the general–purpose microprocessor to video processing where intensive computation and dedicated processing paths are required. 3 methods, i.e.: increase of clock frequency, parallel data paths, coprocessor concept have been used. For particular example of increased clock
rate, the VSP3 in [16] has a Pipelined ALU and Pipelined Convolution Unit, and can run up to 300MHz. The chip consists of 1.27 million transistors using 0.5u BiCMOS with a die size of 16.5mm X 17mm. As an example of coprocessor concept, the AxPe640 V has a RISC core as a controller, and a SIMD module as a slave performing low level tasks. In this architecture, the RISC supports higher level processing, i.e., DCT, quantization and VLC/VLD, while the inner–product like tasks can be handed over to SIMD which is dedicated to arithmetic processing.

Systolic array ——- Architectures based on systolic array are reported in [3–5]. In [4] the system is constructed with only 4 types of cells. These cells are simple combination of adders, multipliers and some control circuits. The advantage of this architecture is the elimination of transposition memory which can be a big power and real estate consumer.

Architectures based on the row–column decomposition approach are most frequently adopted. The process with N by N input matrix can be described as follows (table B.1):

Table B.1 Steps to take in Row–Column Decomposition Approach

<table>
<thead>
<tr>
<th>Step</th>
<th>Input</th>
<th>Process</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Input matrix X</td>
<td>1–D DCT</td>
<td>N–point 1–D DCT along the rows</td>
</tr>
<tr>
<td>2</td>
<td>Intermediate result matrix Y</td>
<td>Transposition</td>
<td>Transposition of Y (Y')</td>
</tr>
<tr>
<td>3</td>
<td>Matrix Y'</td>
<td>1–D DCT</td>
<td>N–point 2–D DCT along the rows</td>
</tr>
</tbody>
</table>

After the decomposition, The problem becomes solving inner product. To implement an inner product, one can use:

* ROM – accumulator
* Hardwired multiplier – accumulator

approaches. The first one is based on Distributed Arithmetic [8], the second one directly maps formula to hardware. Since the inputs are multiplied by constants, a hardwired multiplier gives better performance than a general–purpose one.
Appendix C Additional Schematic Diagrams of the System

This appendix gives some additional schematic diagrams. They can be classified into 3 categories:

- Multb—multg
- Address decoder part 1, part 2
- 4 types of bit slices of CSA

These would help reader understand more about the details of the system.
X : 9-bit 2's complement
X1: 22-bit; X1=2^(-1)*X
X2: 22-bit; X2=2^(-7)*X
X3: 22-bit; X3=2^(-9)*X
X4: 22-bit; X4=2^(-13)*X
X: 9-bit 2's complement
X1: 22-bit; $X_1 = 2^{(-1)} \cdot X$
X2: 22-bit; $X_2 = 2^{(-7)} \cdot X$
X3: 22-bit; $X_3 = 2^{(-9)} \cdot X$
X4: 22-bit; $X_4 = 2^{(-13)} \cdot X$
X : 9-bit 2's complement

X1: 22-bit; X1=2^(-2)X
X2: 22-bit; X2=2^(-5)X
X3: 22-bit; X3=2^(-8)X
X4: 22-bit; X4=2^(-11)X

Fig C.4 Multiplier E
X : 9-bit 2's complement
X1: 22-bit; X1=2^(-3)*X
X2: 22-bit; X2=2^(-4)*X
X3: 22-bit; X3=2^(-8)*X
X4: 22-bit; X4=2^(-14)*X
X : 9-bit 2's complement
X1: 22-bit; X1 = 2^(-4) * X
X2: 22-bit; X2 = 2^(-5) * X
X3: 22-bit; X3 = 2^(-6) * X
X4: 22-bit; X4 = 2^(-13) * X
Fig.C.7 Address Decoder Part 1
Fig. C.8 Address Decoder Part 2
Fig.C.9 Carry Select Adder Bitslice LSBEVEN
Fig.C.10 Carry Select Adder Bitslice LSBODD
Fig.C.11 Carry Select Adder Bitslice MSBEVEN
Fig.C.12 Carry Select Adder Bitslice MSBODD