Hardware Implementation of a Low-Power Two-Dimensional Discrete Cosine Transform

by

Rajul R. Shah

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering

at the Massachusetts Institute of Technology

May 10, 2002

Author _____
Department of Electrical Engineering
May 10, 2002

Certified by _____
Richard E. Anderson
VI-A Company Thesis Supervisor

Certified by _____
Anantha P. Chandrakasan
M.I.T. Thesis Supervisor

Accepted by _____
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Hardware Implementation of a Low-Power Two-Dimensional Discrete Cosine Transform

by
Rajul Shah

Submitted to the
Massachusetts Institute of Technology
Department of Electrical Engineering

May 10, 2002

In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering

Abstract

In this project, a JPEG compliant, low-power dedicated, two-dimensional, Discrete Cosine Transform (DCT) core meeting all IBM Softcore requirements is developed. Power is optimized completely at the algorithmic, architectural, and logic levels. The architecture uses row-column decomposition of a fast 1-D algorithm implemented with distributed arithmetic. It features clock gating schemes as well as power-aware schemes that utilize input correlations to dynamically scale down power consumption. This is done by eliminating glitching in the ROM Accumulate (RAC) units to effectively stop unnecessary computation. The core is approximately 180K transistors, runs at a maximum of 100MHz, is synthesized to a .18µm double-well CMOS technology with a 1.8V power supply, and consumes between 63 and 87 mW of power at 100MHz depending on the image data. The thesis explores the algorithmic evaluations, architectural design, development of the C and VHDL models, verification methods, synthesis operations, static timing analysis, design for test compliance, power analysis, and performance comparisons for the development of the core. The work has been completed in the ASIC Digital Cores I department of the IBM Microelectronics Division in Burlington, Vermont as part of the third assignment in the MIT VI-A program.

Thesis Supervisors: Richard E. Anderson (IBM)
Professor Anantha Chandrakasan (MIT)

Acknowledgements

This thesis project would not have been possible without the guidance of my mentor at IBM, Andy Anderson. I thank him for always being open to new ideas and always being accessible throughout all stages of development. I would also like to thank my manager at IBM, David Sobczak, for making sure I had all the resources I needed in a timely manner. Also at IBM, Peter Jenkins, Tony Kryzak, and Dhaval Sejpal were very helpful and always willing to answer questions. My thesis advisor on campus, Anantha Chandrakasan, deserves many thanks for pointing me in the right direction at key decision points. Furthermore, he managed to read through this mess of a thesis!

My journey through MIT has been filled with moments of stress, happiness and sorrow, stress, laughter and love, stress, excitement and relief, stress, inspiration and accomplishment, and did I mention stress? Without those moments of joy, love, laughter, excitement, and inspiration I don't think I would have succeeded to make it through this place. I want to thank all those individuals who have helped enhance and produce those moments of sanity for me. Without the constant support and love of my parents I would not be here today. In particular, I must thank my mom who forced me to follow my dreams. Without my sister, my life would not be nearly as entertaining nor would I have someone I could share such intimate experiences with. Without Amy, I would never have had the stamina to toil through the night on problem sets and projects, nor would I have had as much fun doing so. I must also thank Ramkumar, who has been so supportive with surprise tea breaks in 6.111 lab and simulation parties at IBM. I need to thank my 'brother' Kamal, who has had such a constant presence in my life at MIT and has never minded being rudely awaken for a mid-night talk. Without the presence of such amazingly loving and supporting friends and family, my experience at MIT would not have been as rich, fulfilling, or enjoyable. Thank you!

Table of Contents

List of Figures

List of Tables

1. Introduction

As multimedia applications on portable, low-power devices become more prominent, the need for efficient, low-power image encoding and decoding techniques increases. The Discrete Cosine Transform (DCT) and the Inverse Discrete Cosine Transform (IDCT) form the transform pair in the JPEG, MPEG, H.261, and H.263 image and video compression standards. Its widespread use can be attributed to the energy compaction quality of the transform. DCT transformation of a natural image from the spatial to the frequency domain results in a concentration of energy in low-order frequency coefficients.

Other applications of the DCT include DCT domain algorithms such as: translation, downscaling, filtering, masking, and blue screen editing. Image enhancements such as brightness adjustment and detail enhancement also become more efficient in the DCT domain. Although the DCT has many applications, the softcore developed in this project targeted the lossy, baseline sequential JPEG application to be developed by the Microelectronics Division of IBM in Burlington, Vermont.

The project, detailed in this thesis, outlines the development of a JPEG compliant 2-D DCT that met all IBM Softcore Requirements. The designed softcore was approximately 160K transistors, ran at a maximum of 100MHz, and targeted a 0.18um double-well CMOS technology with a 1.8V power supply. The design, which scaled power consumption down with increasing input correla-

tion, incorporated optimizations for low power at the algorithmic, architectural, and logical levels. It consumed between 63 and 87 mW of power at 100MHz depending on the image data.

This document investigates the algorithmic analysis, architectural design, implementation, verification, synthesis, static timing, and design for test compliancy phases of the project.

2. Project Requirements

The softcore design for the 2-D DCT was constrained by several sets of requirements. The design had to be compliant with IBM Softcore methodology to facilitate its fabrication and incorporation into standard offerings. Second, the design had to meet all requirements set by the JPEG standard for baseline sequential image compression in order to be used in a future JPEG system. In addition, the design had to be a compatible in the JPEG ASIC that would be developed at IBM. Finally, considerations were made for reusability and time to market.

2.1 IBM Softcore Requirements

The design of the 2-D DCT had to meet all IBM Methodology requirements for Digital Softcores. This requirement imposed constraints on many phases of the project including: algorithm selection, architectural design, synthesis, timing, and design for test compliance.

The design needed to contain operations that could easily be implemented in hardware. For instance, this meant that all computation was constrained to integer arithmetic. Simplicity was necessary since all operations would map to elements in the IBM SA27E Standard ASIC Library or the associated Bit-Stack Library. The standard library contained the building block, lower-level elements like basic gates and flip flops. Sample elements available in the Bit-Stack library include higher level elements such as: adders, multipliers, multiplexors, and comparators. Since ASIC design is intended to be extremely reliable, somewhat technology independent, and easily ported

to future IBM ASIC technologies, the elements in the standard libraries are static CMOS. These hardware limitations affected our choice of algorithms as well as the types of optimizations that were made. For instance, no custom circuitry could be incorporated into the design to help attain low power.

IBM methodology requirements also imposed constraints on the implementation of the core. For instance, it was mandated the use of certain wire load models, power level components, and capacitance values. These limitations are explored in greater detail in the synthesis section (7.0). Methodology also required larger than zero slack times to pass static timing analysis. Stricter timing requirements affected core implementation because it imposed design changes to reduce critical path length and to slow down fast paths. These timing requirements and their impacts are highlighted in the timing section (8.0). In addition, the core implementation was driven for manufacturing and needed to support IBM's design for test compliance methodology. This meant that the core was limited in the number of redundant paths it contained since it needed to attain a certain rate of testability. For more detailed information, refer to the section on design for test compliance (9.0).

2.2 JPEG Standard Requirements

JPEG standards imposed other sets of restrictions on the project. JPEG Baseline sequential operation set the DCT input pixel element (pel) precision to 8 bits/sample. The standard also required 2-D DCT input blocks be 8x8 pels in size. This requirement affected algorithmic selection for

hardware implementation as many algorithms were optimized for particular block sizes that would not be efficient for the mandated 8x8 block size.

While the desired accuracy in the datapath of the DCT is left up to the designer, the standard did impose requirements to limit the lossiness of the system. This margin of acceptable error allowed some degree of exploitation of the human visual system to reduce full accuracy requirements. This leniency allowed for better compression and performance. The limit on lossiness preserves image integrity by preventing the excessive trimming of datapath bit width as a means to achieve reduced power and area.

Therefore, the compliancy requirements outlined in the JPEG standard basically determined the internal bit width or computational precision necessary for each stage in the design. It also determined the degree of accuracy needed to emulate floating point arithmetic as integer arithmetic. These architectural necessities were determined in the verification phase and are described in detail in the compliancy testing section (6.1).

2.3 JPEG Core Design Specifications

The 2-D DCT was being developed for use in a proposed JPEG core that would be designed at IBM. For this reason, the DCT would need to interface properly with the other subsystems in the JPEG core. To meet JPEG core requirements, the design had to support a clock rate less than or equal to 100MHz. JPEG core specifications also indicated the DCT would receive one raw data

input per clock and would output one coefficient per clock. This requirement implied that throughput was not be a key parameter for optimization.

DCT integration into the JPEG core would require additional pipeline functionality for the core to properly communicate at the input and output interfaces. The design would have to handle pipeline stalls, resets, and flushes. Furthermore, it would have to provide information about incorrect operation to other systems. Section 5.0 addresses the developed interface protocol for the design.

2.4 Other Considerations

Aside from meeting all the required standards and specifications, it was desirable to develop a multi-purpose core. While it was desired that the DCT design be compliant with JPEG standards and the JPEG core, it was not desirable for the DCT implementation to be dependant on incorporation into a JPEG core. For example, the chosen implementation should not depend on the presence of a quantization table immediately following the DCT calculation (refer to section 3.3 for more information). Such a dependency would eliminate the modularity of the design and would not allow its use in non-compression applications such as image enhancement.

In addition, other factors such as cost and available time affected design decisions of this core. The project was only allocated funds for one staff member. Furthermore, thesis completion deadlines limited the time available for work on the project to eight months. These combined factors, which limited the number of man hours available for completion of the project, mandated a design with a fast time to market.

3. Background

3.1 Discrete Cosine Transform

The forward discrete cosine transform (DCT) processes 64 spatial samples, arranged as an 8x8 block, and converts them to 64 similarly arranged frequency coefficients. These 64 coefficients are the scale factors which correspond to the 64 respective cosine waveforms shown in figure 3.1 [1]. The cosine basis functions are orthogonal, and hence, independent. Figure 3.1 shows the 64

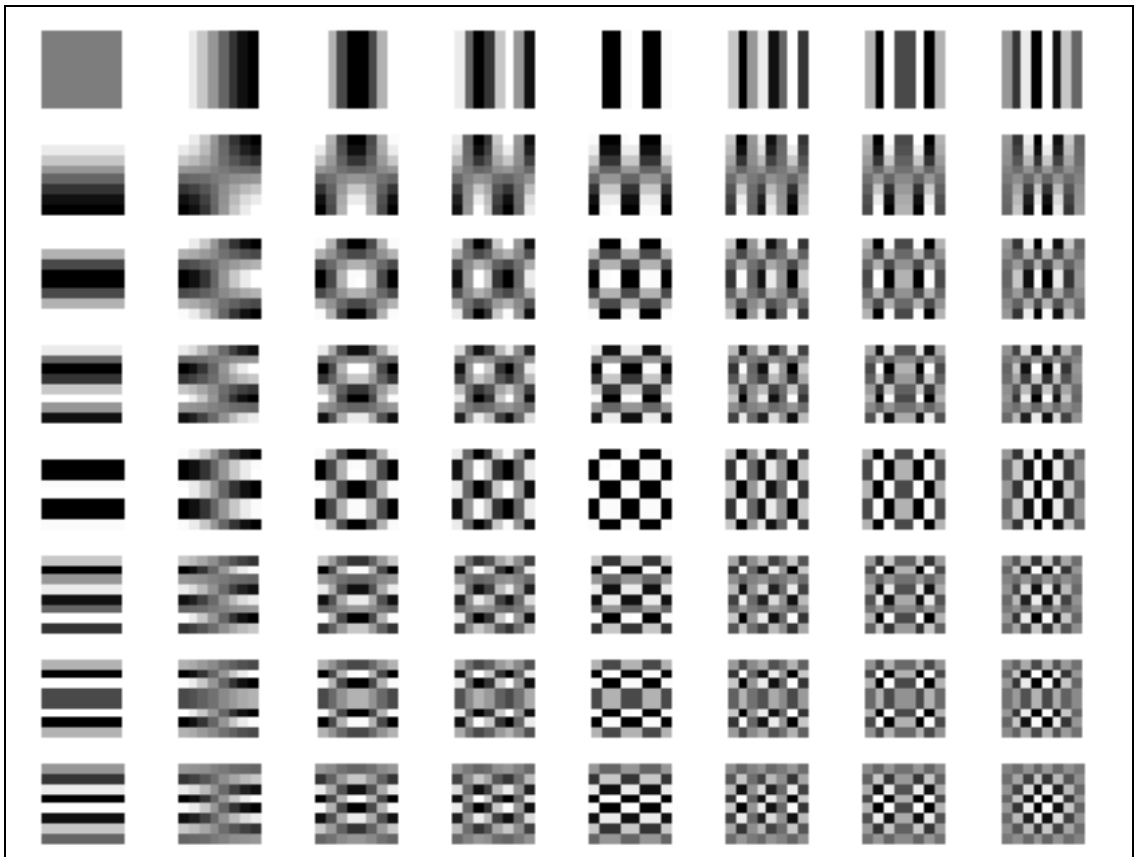


Figure 3.1: 2-D Cosine Basis Functions

cosine basis functions organized in a zigzag fashion of increasing spatial frequency. The basis function in the top left corner is the constant, DC basis function. The lower right corner contains

the cosine basis function with the highest spatial frequency. Horizontal frequencies increase in the basis functions from left to right and vertical frequencies increase from top to bottom. Hence, the top row of figure 1 corresponds to 1-D horizontal basis functions while the left column corresponds to the 1-D vertical basis functions. The 64 2-D basis functions are products of these two 1-D basis functions.

Any block of 64 samples can be represented by first scaling the 64 cosine basis functions by the corresponding 64 DCT-computed coefficients and then progressively summing the results. As a result of the energy compaction properties of the DCT on natural images, the most significant contributions to the reconstructed image are from the low-order frequency coefficients. Therefore, the majority of an image can be captured from just the summation of the lower order values. The DCT coefficients needed for the reconstruction process can be found mathematically from the following JPEG appropriate DCT formulas [2]:

Forward DCT in 1-dimension on 8 samples:

$$S(u) = \frac{C(u)}{2} \sum_{x=0}^{7} s(x) \cos\left[\frac{u\pi}{16}(2x+1)\right] \qquad \text{(Equation 3.1)}$$

$$C(u) = \frac{1}{\sqrt{2}} \qquad \text{for u=0, C(u) = 1 for u>0}$$

s(x) is a sample value
S(u) is a DCT Coefficient

The two dimensional DCT is merely a product of terms between a horizontal 1-D DCT and a vertical 1-D DCT.

Forward DCT in 2-dimensions on an input block of 8x8 samples:

$$S(v,u) = \frac{C(u)}{2}\frac{C(v)}{2}\sum_{y=0}^{7}\sum_{x=0}^{7} s(y,x)\cos\left[\frac{u\pi}{16}(2x+1)\right]\cos\left[\frac{v\pi}{16}(2y+1)\right]$$

(Equation 3.2)

$$C(u), C(v) = \frac{1}{\sqrt{2}}$$ for u,v=0; C(u),C(v) = 1 for u,v>0

s(y,x) is a sample value
S(v,u) is a DCT Coefficient

3.2 Algorithms

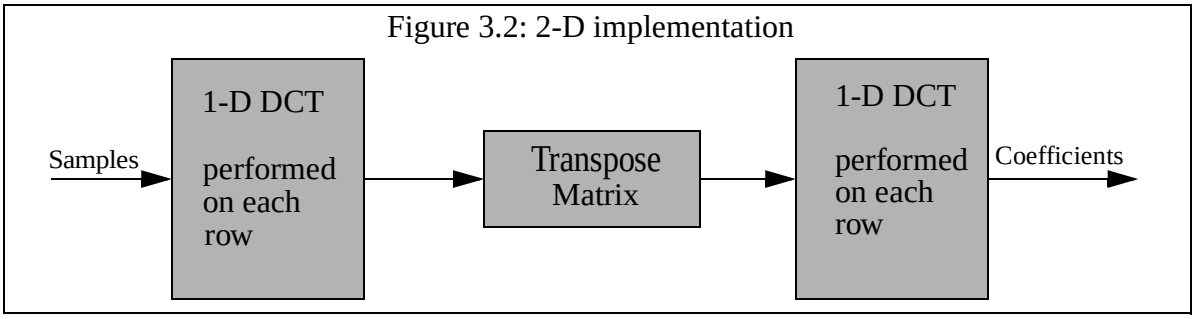
3.2.1 Two-Dimensional Approaches

Implementation of the 2-D DCT directly from the theoretical equation (equation 3.2), results in 1024 multiplications and 896 additions. Fast algorithms exploit the symmetry within the DCT to achieve dramatic computational savings.

There are three basic categories of approach for computation of the 2-D DCT [3]. The first category of 2-D DCT implementation is indirect computation through other transforms--most commonly, the Discrete Hartley Transform (DHT) and the Discrete Fourier Transform (DFT). The DHT-based algorithm of [4] shows increased performance in throughput, latency, and turnaround time. Optimization with respect to these parameters is not the focus of the proposed project. A DFT approach [5] calculates the odd-length DCT, which is not applicable to this project since the design must be compatible with JPEG standards.

The second style of algorithms computes the 2-D DCT by row-column decomposition. In this approach, the separability property of the DCT is exploited. An 8-point, 1-D DCT is applied to each of the 8 rows, and then again to each of the 8 columns. The 1-D algorithm that is applied to both the rows and columns is the same. Therefore, it could be possible to use identical pieces of hardware to do the row computation as well as the column computation. A transposition matrix would separate the two as the functional description in figure 3.2 shows. The bulk of the design

Figure 3.2: 2-D implementation



and computation is in the 8 point 1-D DCT block, which can potentially be reused 16 times--8 times for each row, and 8 times for each column. Therefore, the a fast algorithm for computing the 1-D DCT is usually selected (section 3.2.2). The high regularity of this approach is very attractive for reduced cell count and easy very large scale integration (VLSI) implementation.

The third approach to computation of the 2-D DCT is by a direct method using the results of a polynomial transform. Computational complexity is greatly reduced, but regularity is sacrificed. Instead of the 16 1-D DCTs used in the conventional row-column decomposition, [6] uses all real arithmetic including 8 1-D DCTs, and stages of pre-adds and post-adds (a total of 234 additions) to compute the 2-D DCT. Thus, the number of multiplications for most implementations should be halved as multiplication only appears within the 1-D DCT.

Although this direct method of extension into two dimensions creates an irregular relationship between inputs and outputs of the system, the savings in computational power may be significant with the use of certain 1-D DCT algorithms. With this direct approach, large chunks of the design cannot be reused to the same extent as in the conventional row-column decomposition approach. Thus, the direct approach will lead to more hardware, more complex control, and much more intensive debugging.

It is worth mentioning another direct algorithm [3], which tries to create more regularity than [6]. The result is a large increase in computational complexity. First, real numbers are mapped to the complex domain and then rotation techniques are applied. Other modifications follow. The result is that this direct algorithm requires more computation, but it doubles the throughput. The algorithm developed in [3] is fed with 16 inputs instead of 8. There is no need for this additional complexity when our system can only take 1 input (1 byte) per clock and it is not the intent to optimize with regards to throughput as long as the requirement is met. Thus, of the direct algorithms, the focus was on the algorithm presented in [6].

The computational requirements of both the row-column approach and the direct approach are compared with the use of different 1-D base algorithms in table 3.1. Detailed discussion of the 1-D algorithms can be found in section 3.2.2. From the computational requirements, the row-column and direct 2-D approaches of the DA Chen and Fralick 1-D algorithm appeared optimal since they required 0 multiplications. Although the direct approach used 278 less additions than the row-column approach, it had much greater complexity. Therefore, the number of computations alone could not determine which implementation would result in the lowest power design. Thus,

both 2-D approaches of the Chen and Fralick 1-D DCT with DA were selected for initial architectural design. The characteristics of both designs were accumulated and input to a spreadsheet to determine that the row-column approach was indeed optimal in terms of size and power (section 10.1).

Table 3.1: Comparison of 2-D Approaches with Different 1-D Basis Algorithms.

	2-D:	Row-Column	2-D:	Direct
1-D Base Algorithm	Multiplications	Additions	Multiplications	Additions
Theoretical Equations	1024	896	512	682
Ligtenberg and Vetterli	208	464	104	466
Arai, Agui, Nakajima	208	464	104	466
Arai, Agui, Nakajima with quantization table	144	464	not applicable	not applicable
Chen and Fralick	512	512	256	490
Chen and Fralick using DA	0	1024	0	746

3.2.2 One-Dimensional DCT Algorithms

This section describes some one-dimensional algorithms that were reviewed for use in the row-column or direct approaches to the 2-D DCT. Table 3.2 outlines the number of additions and multiplications required for each 8-point 1-D DCT algorithm that will be discussed in this section.

Table 3.2: Comparison of 1-D DCT Algorithms

1-D Algorithm	Multiplications	Additions
Theoretical Equation 3.1	64	56
Ligtenberg and Vetterli	13	29
Arai, Agui, Nakajima	13	29

1-D Algorithm	Multiplications	Additions
Arai, Agui, Nakajima optimized with quantization table	5	29
Chen and Fralick	32	32
Chen and Fralick using DA	0	64

As outlined in [2], Vetterli and Ligtenberg reduced computation of the 1-D DCT by refining the groupings of sums and differences in the basic 1-D DCT equations. To reduce the number of multiplication operations necessary, their algorithm grouped terms in such a way as to utilize rotation operations. A rotation is a shift about an angle, which recasts appropriately expressed equations containing four multiplications and three additions to equations containing three multiplications and three additions. A total of three rotations are utilized to compute a 1-DDCT. The computational requirements of this fast algorithm for 1-D and 2-D are given in tables 3.2 and 3.1 respectively.

Certain 1-D DCT algorithms become more optimal in the row-column approach when it is known that DCT calculation will be followed by quantization. In these cases, the number of multiplications are reduced by incorporating multiplications within the final stage of a 1-D DCT algorithm into the quantization table. When the Agui, Arai, and Nakajima 1-D DCT described in [2] is implemented in the row-column fashion, as few as 144 multiplications and 464 additions are needed (table 3.1). For this particular algorithm, a savings of 8 multiplications per 1-D DCT calculation on each column can be saved, for a total savings of 64 multiplications on the 2-D DCT computation. The reduction in multiplications is attained by incorporating the scale factors of the final step of the algorithm into a quantization table. The final scale factors of computation on each

row cannot be incorporated into the quantization table because the scale factors are distinct for each coefficient. When elements are summed in the next phase, where the 1-D DCT is applied to each column, those scale factors cannot be factored out. It is important to note that if one optimizes the 2-D DCT calculation by incorporating necessary multiplications into the quantization matrix, the design no longer computes the DCT. It computes a version in which each coefficient needs to be scaled appropriately and is dependant on the presence of a quantization table. Thus, this 1-D DCT algorithm is optimized for use only within a compression core, such as JPEG, where quantization follows DCT computation. Since it is the intent of the project to have a stand alone DCT core, this optimization is not feasible.

It is worth noting that while the direct method of 2-D DCT calculation [6] claims to reduce the number of multiplications in the row-column approach by a factor of two, that is not always true. For example, when the Agui, Arai, and Nakajima 1-D DCT algorithm, optimized for use with a quantization table, is used in row-column decomposition, the direct method does not have as great a comparative savings. This is because the direct method must do some post processing after the 1-D DCT calculation stage so the constant scale factors cannot be incorporated into the quantization matrix. Thus with 13 multiplications per 1-D DCT computation (instead of 8 multiplications in the optimized version), 104 multiplications would result in the direct approach (table 3.2). Although the direct extension of the Arai, Agui, and Nakajima's 1-D DCT to two dimensions did not exactly halve the number of multiplications, it did reduce the number of multiplications by 40 with only a mere increase of 2 additions (table 3.2). The cost, however, is less regularity, which translates to greater complexity of control hardware.

The second 6A assignment in the summer of 2000, focused on implementing the 1-D DCT with the intention of its use in the JPEG core, which was a funded project at the time. Therefore, the Agui, Arai, and Nakajima 1-D DCT, with optimizations for use with a quantization table, was chosen and implemented in C and VHDL. However, at this time, it is desirable for the 2-D DCT implementation to stand alone so it can be used in other image compression or processing functions.

The fast 1-D DCT algorithm that was selected for use in both the direct and row-column 2-D approaches was developed by Chen and Fralick[7]. The 8-point, 1-D DCT, written in matrix factorization, is given below. This algorithm was chosen because it lends itself well to distributed

$$\begin{bmatrix} X_0 \\ X_2 \\ X_4 \\ X_6 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} A & A & A & A \\ B & C & -C & -B \\ A & -A & -A & A \\ C & -B & B & -C \end{bmatrix} \begin{bmatrix} x_0 + x_7 \\ x_1 + x_6 \\ x_2 + x_5 \\ x_3 + x_4 \end{bmatrix} \qquad \text{(Equation 3.3)}$$

$$\begin{bmatrix} X_1 \\ X_3 \\ X_5 \\ X_7 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} D & E & F & G \\ E & -G & -D & -F \\ F & -D & G & E \\ G & -F & E & -D \end{bmatrix} \begin{bmatrix} x_0 - x_7 \\ x_1 - x_6 \\ x_2 - x_5 \\ x_3 - x_4 \end{bmatrix}$$

A=cos(pi/4), B=cos(pi/8), C=sin(pi/8), D=cos(pi/16)
E=cos(3*pi/16), F=sin(3*pi/16), G=sin(pi/16)

arithmetic (DA) implementation thereby eliminating all multiplications. DA remaps the high cost, high power multiplications as additions.

3.3 Distributed Arithmetic

Distributed Arithmetic (DA), as explained in [12], is a bit level rearrangement of a multiply accumulate to recast the multiplications as additions. The DA method is designed for inner (dot) products of a constant vector with a variable, or input, vector. It is the order of operations that distinguishes distributed arithmetic from conventional arithmetic. The DA technique forms partial products with one bit of data from the input vector at a time. The partial products are shifted according to weight and summed together.

Look-up tables (LUTs) are essential to the DA method. LUTs store all the possible sums of the elements in the constant vector. The LUT grows exponentially in size with the dimension of the input, but are optimal on four dimensions.

These characteristics defining DA are visualized through an example. Suppose the following four-dimensional dot product in equation 3.4 is being calculated. The horizontal vector consists of

$$\begin{bmatrix} A & B & C & D \end{bmatrix} \begin{bmatrix} W \\ X \\ Y \\ Z \end{bmatrix} \qquad \text{(Equation 3.4)}$$

constant elements, while the vertical vector contains the four variable elements. In this example, the two's complement variable elements are each eight bits. The most significant bit is the sign bit. In equation 3.5, the variable vector has been expanded to show the individual bits of each element. The various partial products mentioned earlier are attained computing the dot products for

$$\begin{bmatrix} A & B & C & D \end{bmatrix} \begin{bmatrix} W_7 & W_6 & W_5 & W_4 & W_3 & W_2 & W_1 & W_0 \\ X_7 & X_6 & X_5 & X_4 & X_3 & X_2 & X_1 & X_0 \\ Y_7 & Y_6 & Y_5 & Y_4 & Y_3 & Y_2 & Y_1 & Y_0 \\ Z_7 & Z_6 & Z_5 & Z_4 & Z_3 & Z_2 & Z_1 & Z_0 \end{bmatrix} \qquad \text{(Equation 3.5)}$$

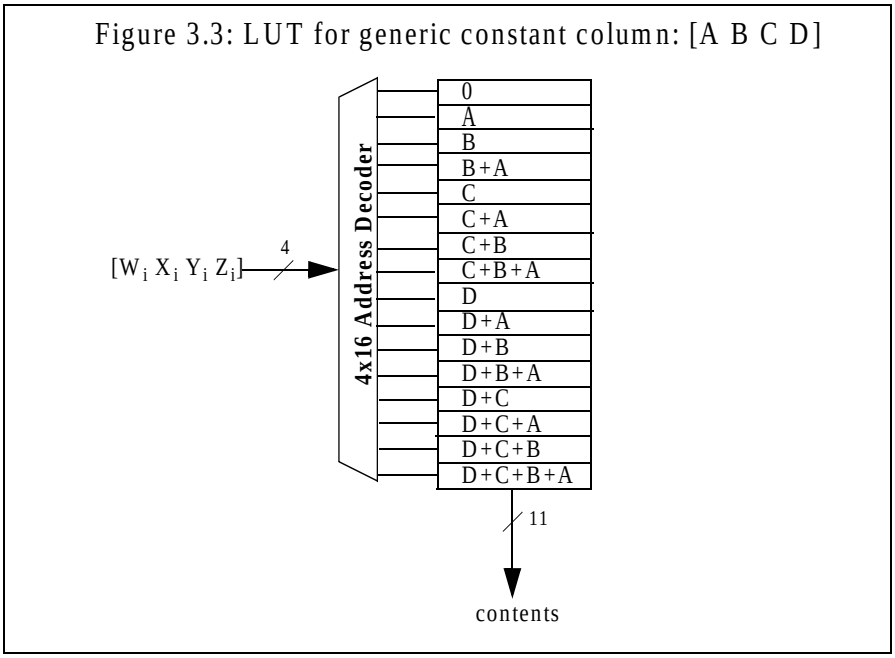
the constant vector with each individual column of bits from the input vector. Each partial product is weighted according to the position of the column of input bits that was used to compute that partial product. The weighted partial products are then summed together to produce the same result attained from conventional arithmetic. The weighted partial product resulting from the dot product with the most significant column of bits is the only partial product that is subtracted since it determines the sign of a two's complement number. Equation 3.6 shows how conventional arithmetic to compute a dot product can be rearranged in a DA manner.

$$
\begin{aligned}
& AW + BX + CY + DZ = \\
-\,& [AW_7 + BX_7 + CY_7 + DZ_7] \cdot 2^7 + \\
& [AW_6 + BX_6 + CY_6 + DZ_6] \cdot 2^6 + \\
& [AW_5 + BX_5 + CY_5 + DZ_5] \cdot 2^5 + \\
& [AW_4 + BX_4 + CY_4 + DZ_4] \cdot 2^4 + \\
& [AW_3 + BX_3 + CY_3 + DZ_3] \cdot 2^3 + \\
& [AW_2 + BX_2 + CY_2 + DZ_2] \cdot 2^2 + \\
& [AW_1 + BX_1 + CY_1 + DZ_1] \cdot 2^1 + \\
& [AW_0 + BX_0 + CY_0 + DZ_0] \cdot 2^0
\end{aligned}
\qquad \text{(Equation 3.6)}
$$

Each bit ofW, X,Y, and Z is either a zero or one. Therefore the terms in the brackets (equation 3.6) all reduce to combinations of sums of the four known constants: A, B, C, D. For each term in the brackets, constant A, B, C, and D are either included in the sum or not, depending on the values of W_i, X_i, Y_i, and Z_i for a particular bit i. With four constants, there are only 16 possible values that the terms in the brackets can reduce to. These values are precomputed and stored in a 16-

element LUT that is decoded with W_i, X_i, Y_i, and Z_i. The table decoding scheme for this example (figure 3.3) is small and fast since it only needs to store 16 elements.

Figure 3.3: LUT for generic constant column: [A B C D]



As shown in equation 3.6, DA reorders computation to recast multiplications with additions. For the example case, where the inputs are represented with 8 bits, the DA method recasts the 4 multiplications and 3 additions to 7 additions, 8 lookups to a table, and 8 shifts. The shifts are negligible since they are simply mapped to rewiring in hardware. However, the savings achieved in power and size from forgoing conventional multiplication is significant. To illustrate, an 8-bit multiplier from the IBM Bitstack library requires approximately 4 times as many cells as an 8-bit adder. Power consumption scales comparably as well.

DA is implemented with the least possible resources by computing it in a fully bit-serial manner. This means that one column of bits is fed into the system at a time. Equation 3.7 shows a factorization of equation 3.6 that elucidates how such a method can be implemented in a serial manner.

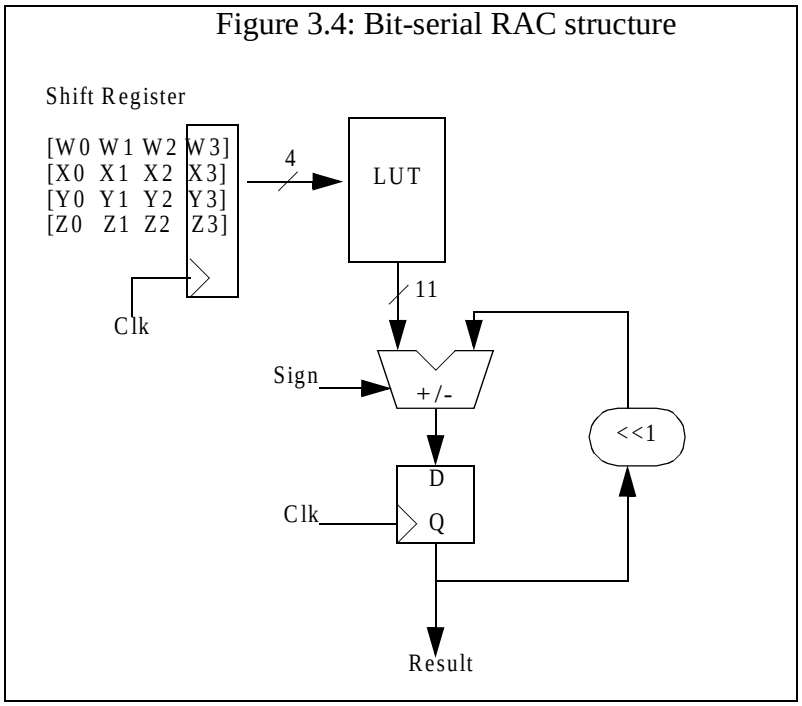
The terms in the brackets resulting from the look up tables are abbreviated to LUT_i, where i is the column of bits being input to the LUT.

(Equation 3.7)

$$AW+BX+CY+DZ=$$
$$(((((((-LUT_7)2+LUT_6)2+LUT_5)2+LUT_4)2+LUT_3)2+LUT_2)2+LUT_1)2+LUT_0$$

The key elements required to implement the DA are a 16-element LUT and decoder, an adder/ subtractor, and a shifter. These elements are grouped together in a ROM Accumulate (RAC) structure [10] as shown below (figure 3.4). A shift register inputs one column of input bits per

Figure 3.4: Bit-serial RAC structure

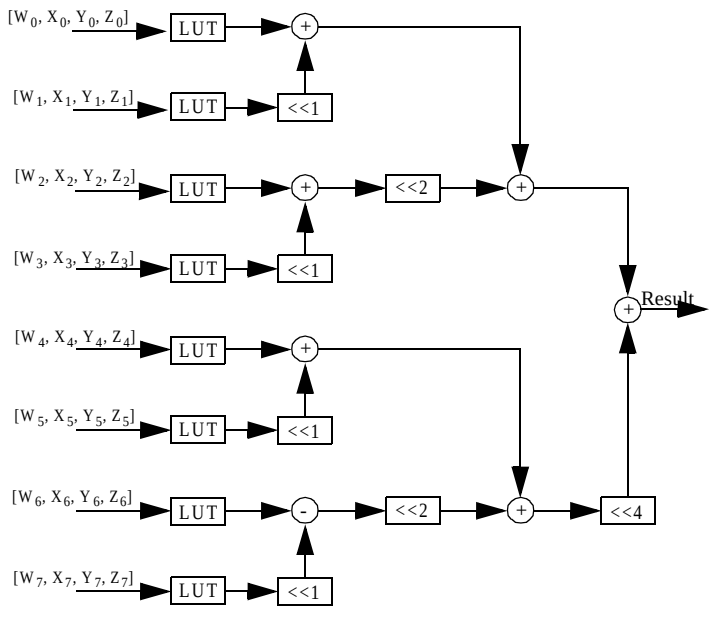


clock cycle to the LUT. It begins by inputting the most significant column of bits and rotates to finally input the least significant column of variable bits. The contents from the LUT get summed with the shifted contents of the previous look up. In this fully bit-serial approach, the answer converges in as many clock cycles as the bit length of the input elements. While the serial inputs limit

the performance of the RAC, it requires the least possible resources. Greater performance can be achieved with an increase in hardware.

With an increase in resources, the result of the RAC can converge quicker. The speed of the calculation is increased by replicating the LUT. In a fully parallel approach, the result of the DA converges at maximum speed--the clock rate [12]. In this case, the LUT must be replicated as many times as there are input bits. For the example in equation 3.6, where the inputs are 8 bits, the LUT must be replicated 8 times for the answer to converge in one clock cycle. This allows for each of the columns of input bits to be calculated in parallel. An adder tree can be used to weight and combine the results from the individual columns. The balanced tree increases regularity, reduces the number of adders, and the critical path delay. This fully parallel approach is highlighted below in figure 3.5.

Figure 3.5: Fully parallel DA implementation



Aside from the two extremes of the fully bit serial approach and the fully parallel approach, a partially paralleled structure can be adopted. The extent of parallelization can be varied to make the dot product converge between the number of clock cycles as input bits to as quickly as a single clock cycle.

4. Approach to Low Power

The development of the 2-D DCT was optimized for low power mainly at the algorithmic and architectural levels. Power considerations were also made whenever possible during the development stages.

Power could not be conserved through circuit or device optimizations. IBM digital ASIC softcore methodology did not support circuit or device level design since all elements within the design were instantiated from standard libraries. There were some choices available between power levels of elements, so wherever possible, the lowest power consuming elements were set to be selected by the synthesis tool.

The newest available technology was selected--SA27E. The technology selection included a conscious decision to reduce power. SA27E is a static CMOS 7SF technology with a 0.18 um lithography process, and an effective gate length of 0.11 um. The small gate length reduced capacitive loads and thus, proportionately reduced power consumption. Voltage islands and threshold scaling were not supported within the SA27E technology or within digital ASIC softcore methodology. With very little leeway for low power design at a custom circuit level, emphasis was placed on minimizing power consumption at the algorithmic and architectural levels.

Computational efficiency was one of the primary attributes optimized to reduce power. Specifically, an algorithm that minimized the number of multiplications without creating too dramatic an

increase in the number of additions was selected. With the use of DA, equation 3.3 eliminated all multiplications. Furthermore, the row-column decomposition method was selected for implementation after careful power comparisons against the direct method (10.1). The regularity of the row-column approach significantly reduced power consumption as well as area mainly due to the simplicity of the required control logic.

While physical size in terms of cell count was also minimized, it was mostly done as a by-product of power reduction methods. For example, the datapath bit width was sufficient enough to pass JPEG standard compliancy, but was kept at a minimum to reduce power. The minimum bit widths reduced resources since fewer flip flops were needed between stages, and that in turn reduced switching power. However, there were many instances in which logic had to be inserted to reduce power consumption. In these cases, power reduction took precedence over area reduction.

The architectural design kept extraneous switching of logic at a minimum. Key examples in the control stage include the choice of a Moore FSM over a Mealy FSM model and the design of one-round decrementers (5.2.1). Another example throughout the design was that a clear of the system did not reset all the flip flops in the system, but rather just a few key flip flops necessary for proper control. This led to the insertion of an additional mux in each RAC unit, but it was worth the power savings from not having to switch all the flop flops to 0 on a reset when they would be overwritten anyway. Another significant example was the insertion of flop flops before the LSB and MSB RAC unit partial products within both RAC stages were combined. The flop flops were inserted to eliminate the extraneous switching of the adders (5.2.5, 5.2.9).

All flip flops in the design were kept from switching during the assertion of the *stall* primary input. This signal was implemented as a clock gate wherever optimal (7.1). This reduced the power associated with the switching of clock nets.

The most significant power reduction can be attributed to the adopted “power awareness” scheme. ROM Accumulate (RAC) structures that implemented Most Significant Bit Rejection (MSBR) [10] were used to scale power consumption according to input correlations.

Most implementations of the DCT assume the worst case operating conditions. They have primarily focused on minimizing power through algorithmic studies to reduce the number of multiplications. Computation in these implementations are minimal, but constant. Xanthopoulos [10] implemented a 2-D DCT in the conventional row-column decomposition approach using the Chen 1-D algorithm [7]. The Chen 1-D algorithm, implemented with distributed arithmetic, lent itself well for scaling power consumption according to operating conditions. Utilizing the high degree of spatial correlation within image data, the work in [10] resulted in adaptive bit width computation through the introduction of MSBR. When successive spatial inputs were highly correlated with one another, the amount of computation scaled down.

4.1 Power Awareness

Distributed arithmetic implementations lend themselves well to power awareness schemes. Power awareness, as formulated in [13], pertains to the ability of a system to dynamically scale down power consumption based on operating conditions.While conventional designs operate under

worst-case operating scenarios, power-aware systems adapt to use the least possible resources in accordance with the current operating environment. In a temporal solution to power-awareness, such as the one implemented in the 2-D DCT, the entire system is used to handle worst case situations. When not operating under worst case states, parts of the system are shut off, or prevented from switching. In a spatial solution to power awareness, different point systems, each with different power consumption and cell count, are built to handle specific scenarios. There is always a point system to handle the worst case scenario, and up to as many point systems as operating scenarios. Each point system is optimized in size and power for that specific scenario. Usually, only those scenarios with high enough probability of occurring, make it worth the extra overhead to incorporate.

4.1.1 Most Significant Bit Rejection

Both the temporal and spatial realizations of power-aware systems require scenario determining units. The overhead of this unit should not be so great as to overshadow the reduction in power from making the data path power-aware. A MSBR unit functions as the scenario determining unit for the Chen and Fralick matrix factorization algorithm for computing the DCT (equation 3.3) when implemented with a RAC structure. The MSBR unit consists of two different scenario detections outlined by Xanthopolous [10]. The first detection scheme, termed MSBR123, deals with the scenarios that affect rows 1, 2, and 3 of the Chen and Fralick DCT constant matrix (equation 3.3). The second detection unit, referred to as MSBR04567, affects rows 0, 4, 5, 6, and 7 of the constant matrix in equation 3.3.

MSBR123 detects scenarios pertaining to the dot products of equation 3.3 shown again below in equation 4.1. The column vector, or variable vector, is one column of bits formed from the sum of

$$\begin{array}{l} row\ 1 \\ row\ 2 \\ row\ 3 \end{array} \begin{bmatrix} B & C & -C & -B \\ A & -A & -A & A \\ C & -B & B & -C \end{bmatrix} \begin{bmatrix} (x_0 + x_7)_i \\ (x_1 + x_6)_i \\ (x_2 + x_5)_i \\ (x_3 + x_4)_i \end{bmatrix} \qquad \text{(Equation 4.1)}$$

the inputs. If the variable vector is all zeros, then it is known that the partial products from rows 1, 2, and 3 of the constant matrix will be zero. Furthermore, if the variable vector is all ones, then the result of the partial product will the sum of the constants in each constant row. For these three rows, the sum of all the constant elements is zero. Therefore, when the variable column of bits is all ones, the partial product is known to be zero as well. For both of these cases, the MSBR123 will keep the RAC structure from computing, weighting, and summing that partial product. An example is formulated in figure 4.1 to illustrate how the MSBR123 works. In this example, all 8 columns of bits forming the variable vector are shown. Normally, each column of bits would be

Figure 4.1: MSBR123 Example

1	0	0	1	0	1	1	0
1	1	0	1	0	1	1	0
1	1	0	1	1	1	0	0
1	0	0	1	1	1	1	0

fed to the LUT of a RAC structure for decoding, summation, and weighting. However, when the MSBR123 is employed, those columns of bits which are boxed will not be fed to the LUT. This will keep the LUT from glitching, the adder from consuming power, and the flip flops within the RAC from switching. These columns of bits will not effect the result since they only add zero to

the progressively summed result. The process of identifying these cases is described in greater detail in section 5.2.4.

MSBR04567 is used to determine scenarios on the variable vectors that are multiplied with rows 0, 4, 5, 6, and 7 of the constant matrix from equation 3.3. For convenience, they are extracted and replicated in equation 4.2. Since the sum of the constants from these rows do not equal zero

$$\textit{row 0} \quad \begin{bmatrix} A & A & A & A \end{bmatrix} \begin{bmatrix} (x_0 + x_7)_i \\ (x_1 + x_6)_i \\ (x_2 + x_5)_i \\ (x_3 + x_4)_i \end{bmatrix} \qquad \text{(Equation 4.2)}$$

$$\begin{matrix} \textit{row 4} \\ \textit{row 5} \\ \textit{row 6} \\ \textit{row 7} \end{matrix} \begin{bmatrix} D & E & F & G \\ E & -G & -D & -F \\ F & -D & G & E \\ G & -F & E & -D \end{bmatrix} \begin{bmatrix} (x_0 - x_7)_i \\ (x_1 - x_6)_i \\ (x_2 - x_5)_i \\ (x_3 - x_4)_i \end{bmatrix}$$

another approach to bit rejection is taken. The detection for these rows is based on the concept that sign extension of a two's complement number does not provide any new information. Therefore, if the entire column of bits from the variable vector are part of the same sign extension for each respective variable element, then that column will not be passed on to the RAC structure for computation. Figure 4.2 shows an example variable vector that MSBR04567 operates on. The

Figure 4.2: MSBR04567 Example

1 1 1 1 0 1 1 1
1 1 1 1 1 0 1 1
1 1 1 0 1 0 1 1
1 1 1 0 0 0 1 1

first two columns are extraneous sign extension bits and are therefore rejected by MSBR04567. Hence, the first two columns will not be fed to the corresponding RAC structure. The third col-

umn is the last column indicating the sign of the elements, and thus, it is necessary for correct computation of the dot products. The last two columns are all ones, but, they are not part of the sign extension, so they must still be fed to the RAC structures. The method for MSBR04567 rejection is described in greater detail in section 5.2.4.

This example (figure 4.2) demonstrated the difference between MSBR04567 and MSBR123 detection schemes. MSBR123 detection would have rejected the first three columns as well as the last two columns from RAC calculation. Clearly, MSBR04567 is a more restricted form of MSBR123. MSBR123 already handles sign extension, since sign extension consists of columns of all ones or zeros. MSBR123 can also reject computation on columns of zeros or ones that are not part of the sign extension.

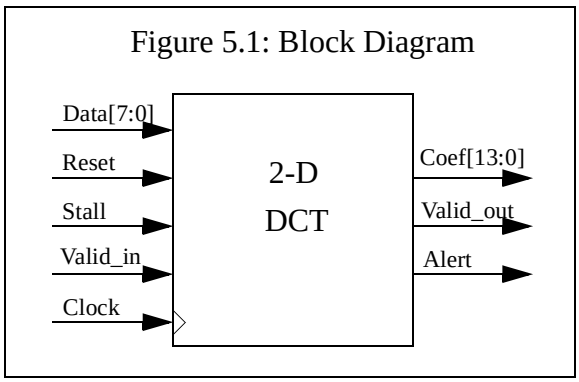
The MSBR04567 unit is especially useful for rows 4, 5, 6, and 7 since the variable vector is formed from the differences of the 8 input pels. Since the 8 inputs are neighboring pixel values from a natural image, there will be a high degree of correlation between their values. Therefore, differences between the pels will be centered around zero. This means that many of the most significant bits resulting from the differences will belong to the sign extension.

It is important to note that the MSBR stage and its effects on RAC computation do not create any error. MSBR only reduces unnecessary computation, thereby reducing power consumption by limiting the glitching of combinational logic, and the switching of flip flops.

5. Architectural Design

This section describes the architecture of the 2-D DCT core and the different ways in which data flows through the pipeline. The primary input and output signals to the 2-D DCT core are shown in the block diagram in figure 5.1. *Data* is an 8-bit pixel element (pel) as specified by the JPEG

Figure 5.1: Block Diagram



Standard and is provided serially in row-major order to the core.

The core has three control signals: *reset*, *stall*, and *valid_in*. Assertion of *reset* initializes the core. *Stall* functions as a clock gate and while it is asserted the state within the core stays the same along with all values stored throughout the pipeline.V*alid_in* reflects the validity of *data* with respect to each clock cycle. This signal would probably be passed as part of a handshaking agreement from a proceeding core. Only once the pipeline is filled with valid inputs can valid outputs begin arriving at the output of the core. The design is synchronous--all flip flops in the design are controlled with the rising edge of *Clock*. The precedence of the input control signals in as follows: *Reset*, *Stall*, and *Valid_in*. If there is a reset asserted at the same time as a stall, the system will reset instead of stall. Similarly, the system will handle stalls before it determines whether to grab valid data.

The output signal, *Coef*, is a 14-bit DCT coefficient. Although the coefficients are output serially from the core, they are output in a different order than samples are input. To limit the amount of storage elements within the core, the coefficients are output as they are computed in column-major order instead of row-major order. Under normal operation, *valid_out* instantaneously reflects the validity of *Coef*. The assertion of *alert* provides warning that improper use of the pipeline has occurred. Normal pipeline functionality is summarized in table 5.1 below.

Table 5.1: System Functionality

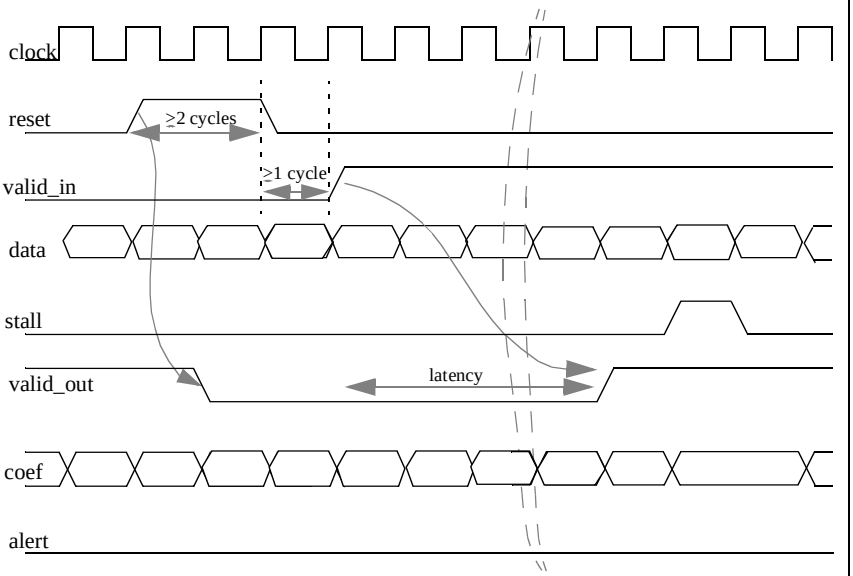
Reset	*Stall*	*Valid_in*	*Valid_out*	Function
1	X	X	X	system initialization
0	1	X	X	system hold (clock gate)
0	0	0	0	idle
0	0	0	1	pipeline flushing
0	0	1	0	pipeline filling
0	0	1	1	steady state computation

5.1 Interface Protocols

The timing of the signals at the core interfaces is crucial for correct functionality. The typical input and output behavior of the core is shown in the timing diagram below (figure 5.2).

Reset must be asserted for at least 2 clock cycles to correctly initialize the core. Once *reset*

Figure 5.2: General Behavior

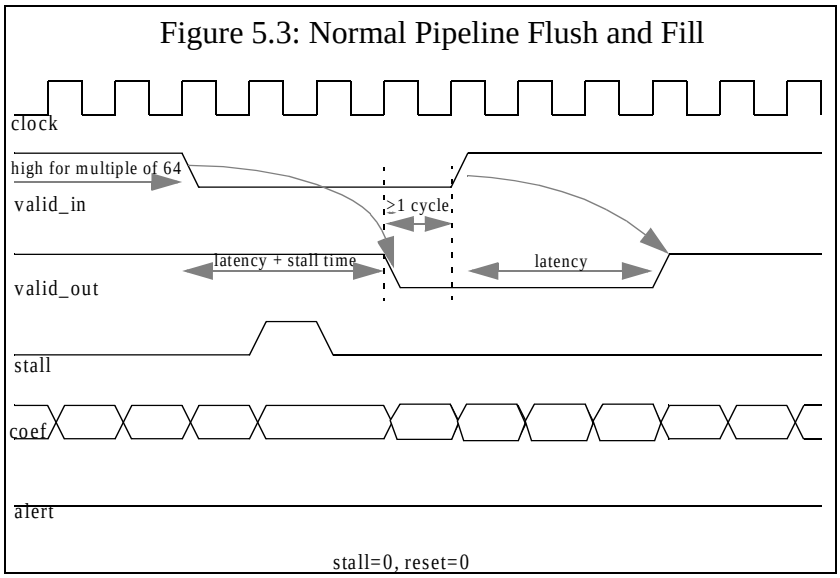


becomes deasserted, valid inputs cannot be accepted until at least one clock cycle later. After the onset of valid inputs, if there are no stall requests, it will take the latency of the system, which is 108 clock cycles, for valid coefficients to appear at the output of the core. At this point, if there continues to be valid inputs, the system will operate in steady state mode. The pipeline can be stalled at any time without impacting the DCT computation. In figure 5.2, the stall functionality is shown during the steady state phase.

A pipeline flush refers to the calculation and expulsion of those DCT coefficients which can be computed from the remaining valid data in the pipeline once the new inputs have ceased to be valid. The onset of pipeline flushing occurs when *valid_in* transitions from a high to a low. Since there may be more than one block of 64 input samples being processed within the pipeline at once (the latency through the pipeline 108 clock cycles), there may still some valid data in the pipeline

whose computation must be completed once *valid_in* falls. If a flush begins exactly after *valid_in* has been asserted for a multiple of 64 cycles, then a normal flush, demonstrated in figure 5.3, will occur. When *valid_in* has been high for a multiple of 64 cycles, it falls at a block boundary, indi-

Figure 5.3: Normal Pipeline Flush and Fill

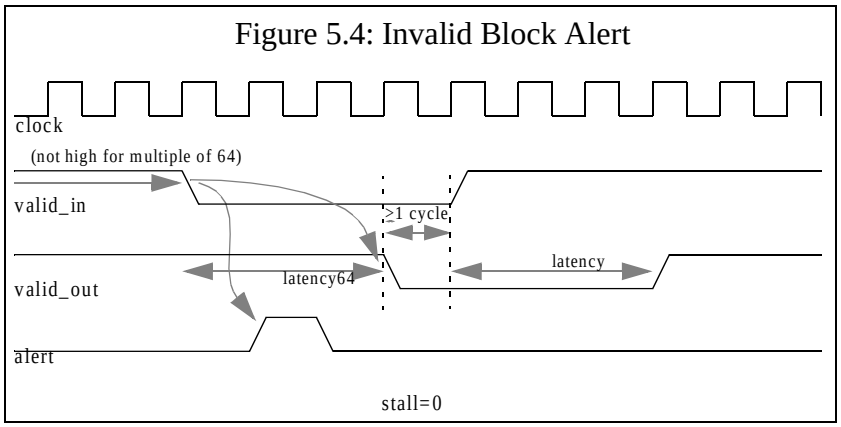


cating that the pipeline contains only valid, full blocks of 64 samples that are required for 2-D DCT computation. From the onset of this kind of flush, under normal operation, it will take the latency of the system for the flush to complete. Figure 5.3 also shows the affects of a stall during a flush. Of course, stalls will increase the time for flush completion.

Figure 5.3 also highlights how a the pipeline can correctly begin filling again after a flush. After at least one clock cycle from the completion of a flush, when *valid_out* transitions from 1 to 0, the system can begin accepting valid inputs, meaning *valid_in* can transition from 0 to 1. Again, it will take the latency of the system from the onset of pipeline filling before valid outputs are computed.

Flushing can create an alert if there is an incomplete data block in the pipeline. This alert should not be thought of as incorrect flushing, but rather, as an indication to the system that some valid data that was entered into the system will not be used. This would give the chance for the system to rerun this data if required. The data of a valid, but incomplete block is not used because the 2-D DCT requires a full input block of 64 samples to complete computation. The system will still correctly indicate valid outputs that correspond to coefficients calculated from the last full input block that is in the pipeline. Hence, in the case where an incomplete block has entered the pipeline, the completion of a pipeline flush will have variable latency, called latency64. Latency64 is calculated from the point where the last valid 64th sample is in the pipeline until when that last corresponding valid coefficient is computed. The maximum value of latency64 is the latency of the system. An example is diagramed in figure 5.4. Again, for this example, correct refilling of the

Figure 5.4: Invalid Block Alert

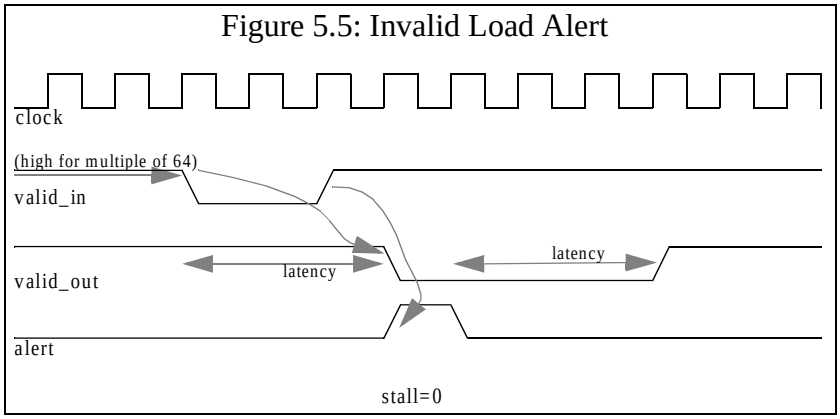


pipeline is shown.

Another case in which *alert* could become asserted is with the incorrect refilling of the pipeline. This occurs when the pipeline begins to fill before a pipeline flush is complete, or basically while *valid_out* is still high. In this case, the computation of coefficients will proceed correctly, but *valid_out* will not properly track the valid outputs corresponding to inputs that arrived during the

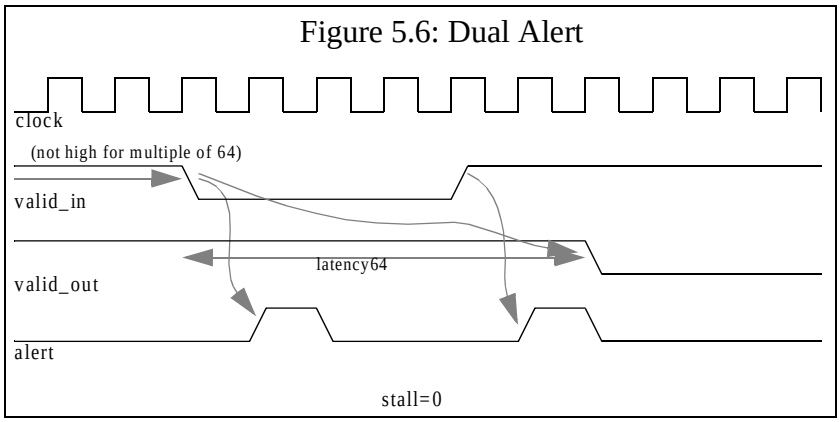
flush. *Valid_out* will only rise after the latency of the system from one cycle after the flush is complete. Figure 5.5 helps illustrate this example of an invalid load alert.

Figure 5.5: Invalid Load Alert



Alert can react to both invalid loads and invalid blocks. *Alert* is first fired when the system begins a pipeline flush at a non-block boundary. *Alert* becomes asserted again if the pipeline begins to refill before all the valid coefficients have been flushed. This example is illustrated in figure 5.6.

Figure 5.6: Dual Alert

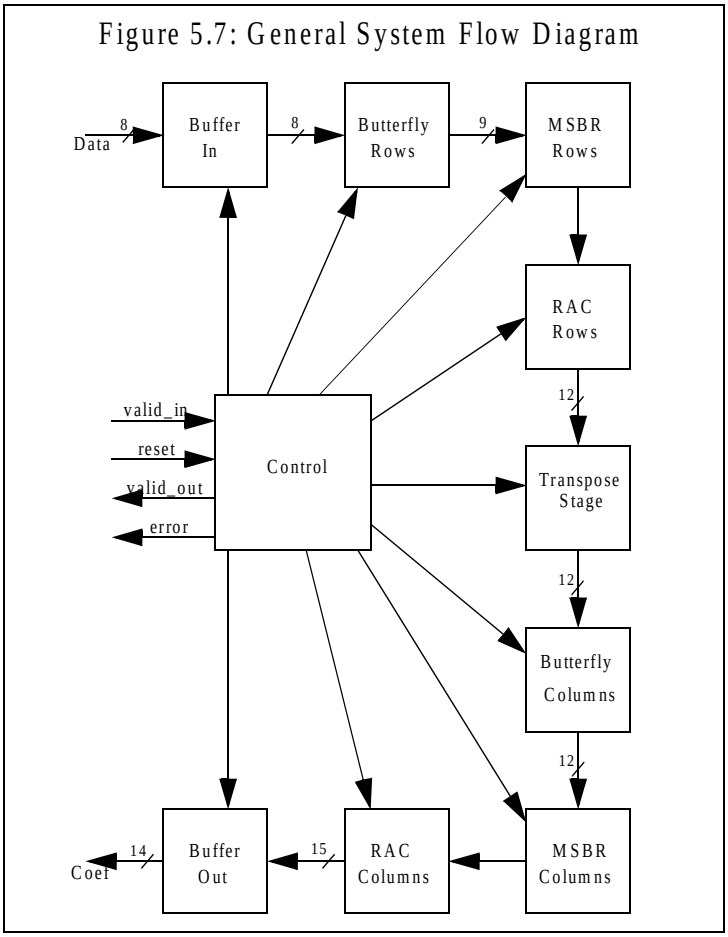


5.2 Implementation

The implementation of the 2-D DCT follows the row-column technique on the Chen and Fralick fast 1-D DCT (equation 3.3). The design consists of 10 units: Control, Buffer in, Butterfly Rows, MSBR Rows, RAC Rows, Transpose, Butterfly Columns, MSBR Columns, RAC Columns, and

Buffer out. These units are arranged as shown in figure 5.7 to provide the general pipeline flow through the core. Butterfly Rows, MSBR Rows, and RAC Rows compose the calculation of the 1-

Figure 5.7: General System Flow Diagram



D DCT on each row. Similarly, Butterfly Columns, MSBR Columns, and RAC Columns compose the calculation of the 1-D DCT on each column. The row 1-D DCT perform the same function as the column 1-D DCT, but they operate on different bit widths. MSBR Rows and MSBR Columns are two different instantiations of identical hardware. All other units are unique. Although it is not be explicitly stated in each section, all flip flops in the design are triggered off the rising edge of the same system clock. Also, all flip flops are not enabled when *stall* is asserted. Signal *stall* was set as a clock gate in the synthesis tool for all the flip flops with large enough bit width. Please refer to section 7.1 for more details.

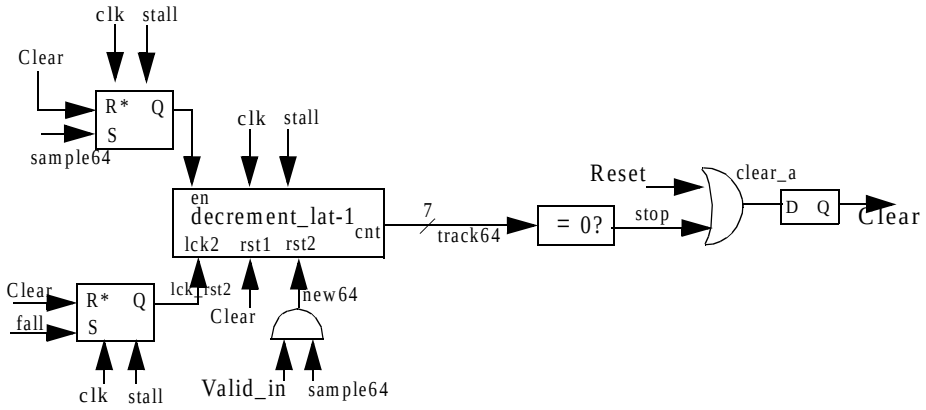
5.2.1 Control

The control stage has been divided into two interacting subunits: the control logic and the control finite state machine (FSM). The Control Logic subunit is responsible for the generation of three signals: *clear, alert,* and *valid_out.*

5.2.1.1 *Clear* Generation

Clear functions as the internal reset to the control FSM, MSBR units, and RAC units. The logic to generate this signal is shown in figure 5.8. Clear becomes asserted by either a hard reset of the

Figure 5.8: Generation of Clear Signal



system or by the completion of a pipeline flush. The end of a pipeline flush is marked with the assertion of *stop*. The logic to generate *stop* tracks the progress through the pipeline of the last valid 64th pel latched into the system. The main unit responsible for this tracking is a special, one-round, 7-bit decrementer. The decrementer tracks the 64th sample of the last valid input block by

only enabling countdown after the first 64th sample has been latched into the system. This is achieved with a Set-Reset flip flop (SR-ff). The SR-ff is set with the *sample64* signal from the control FSM and is reset with the *clear* signal.

The tracking decrementer, when reset, is initialized to begin the countdown from the latency of the system. Although the latency of the system is 108 cycles the decrementer is set to countdown for 107 cycles. This is because the stop signal must be generated one clock cycle before the pipeline flush is actually completed. The stop signal asserts *clear_a* which must go through a D-ff, causing the clear to be delayed by one cycle. In this manner, the system clears exactly at the completion of a pipeline flush. When the decrementer is enabled, and not stalled or reset, it counts down by one every rising clock edge until it hits zero. The value of the decrementer will remain at zero until reset.

This tracking decrementer has been specially designed to reset only under certain conditions. First, it will reset whenever the clear signal is asserted. Given the condition that a lock is not in place on the tracking decrementer, then it will also be reset whenever the 64th pel of a valid block is latched into the system. Under steady state operation, this decrementer is reset to the latency value every 64 clock cycles with every 64th valid input. While the decrementer is not locked, meaning *lck_rst2* is low, the decrementer will continue to be reset in this cyclic manner. Therefore, decrementer cannot hit zero to assert the stop signal until a lock is set on the decrementer preventing it to reset. This lock is set via another SR-ff that outputs *lck_rst2*. It is set when *valid_in* transitions from a 1 to a 0 to indicate the onset of a pipeline flush. This implies that only

during a pipeline flush will the decrementer be allowed to count down to zero. The SR-ff controlling the decrementer lock is reset with the *clear* signal.

When the count of the 64th sample tracking decrementer hits zero, *stop* is asserted. When the *stop* or the *reset* are asserted, *clear_a*, will become asserted after the propagation time associated with an OR gate. As mentioned before, *clear* is derived from sending the *clear_a* through a D-ff. This was worth the cost of the extra cells associated with the flip-flop for several reasons. First, after synthesis, the name of *clear* would be preserved if it were the output of a flip flop. This would aid in debugging the netlist. Second, the flip flop would buffer the *clear* signal, which drives quite a few other gates. Furthermore, adding the flip flop may have broken what could have been critical timing paths.

From the implementation in figure 5.8, it is clear why *Reset* must be asserted for two clock cycles to initialize the core. One clock cycle is necessary for *Reset* to generate *Clear*, and the second clock cycle is needed for *Clear* to restart the decrementer. Only once the decrementer does not output an undefined value will *stop* become defined. *Reset* must be asserted until *stop* becomes defined otherwise *clear_a* will be undefined since the OR of 0 with X is X. Clear will then become undefined as well and the core will not function correctly.

The design for the implementation of *clear* contains some notable optimizations. The decrementer was specially designed to count down only one round instead of using a Bit Stack library element to save computational power. There was no need for the extra power consumption associated with

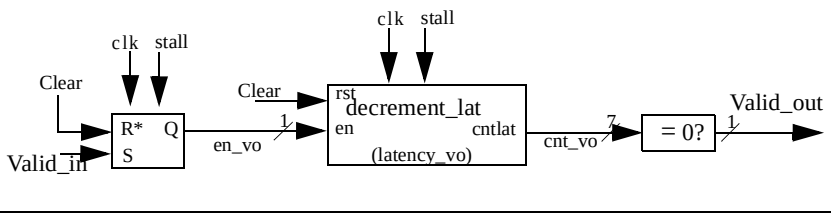
letting the decrementer wrap around and count down again after it hit zero. Using a regular decrementer, the output would keep transitioning. In that case, the output of the zero check would have to set another SR-ff to achieve the desired level output for stop. Using the special design saves this extra latch, which not only reduces cell count but also clock load.

Furthermore, the decrementer is used instead of an incrementer to save cells. The check for zero at the output can be synthesized to a simple 4-input nand gate. The use of an incrementer would create a check for the latency value of 106, which could require more logic.

5.2.1.2 *Valid_out* Generation

The control logic also generates the *valid_out* primary output (PO) signal. The implementation is shown in figure 5.9. The bulk of this design contains a similar one round decrementer to that used

Figure 5.9: Generation of Valid_out Signal



to generate *Clear*(figure 5.8). Again, this type of design was used to reduce power consumption. This decrementer differs from the one in figure 5.8 because it only resets with *Clear* and does not contain the lock functionality on the reset. Also, this decrementer counts down the full latency of the system, or 108 clock cycles, instead of one cycle less than the latency. The full count down is possible because *Valid_out* does not have an additional one flip flop delay. *Valid_out* is merely the

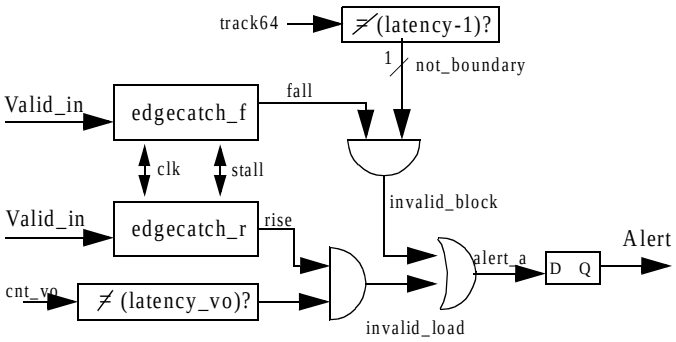
output of the zero check on the count from the decrementer.

The decrementer is enabled to countdown via a SR-ff. The SR-ff is set once the system receives valid inputs and is reset with *clear*. Once *valid_in* has gone high initially, the decrementer can begin countdown. From that point, *valid_out* becomes asserted after the latency of the system to indicate the arrival of valid DCT coefficients at the output. If valid_in were to become deasserted, *clear* would become asserted after some time to reset this decrementer and the SR-ff. This in turn would cause *valid_out* to go low again.

5.2.1.3 *Alert* Generation

The final function of the control logic is to generate the PO *Alert*. The *alert* signal reflects invalid blocks or invalid loads during a pipeline flush. More details can be found in section 5.1. *Alert* is generated with the logic shown in figure 5.10.

Figure 5.10: Generation of Alert Signal



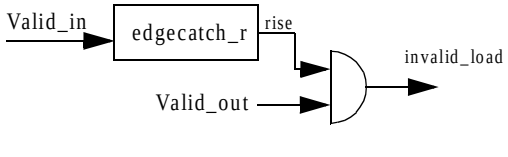
Track64 is the output count from the tracking decrementer (figure 5.8). When *track64* does not

equal the value 106, it indicates that the 64th sample was not the last sample latched into the system, and asserts *not_boundary. Track64* can only equal 106 when the tracking decrementer is reset, and that occurs when the 64th sample of a valid block is input. If *not_boundary* is high at the time that *Valid_in* transitions from high to low, it indicates that *Valid_in* has not been high for a multiple of 64 cycles at the time a pipeline flush begins. In this case, *invalid_block* becomes asserted to highlight an invalid block alert.

An invalid load alert arises when *valid_in* goes high when there are still valid coefficients in the pipeline. This would mean valid inputs would be loaded before the pipeline could complete flushing. This case is detected with the assertion of *invalid_load* when two conditions are simultaneously met: *cnt_vo* does not equal the latency value and *valid_in* transitions from low to high. Signal c*nt_vo* is the output from the valid out decrementer (figure 5.9). If *cnt_vo* does not equal the latency stored in the valid out decrementer, then it indicates that valid data is still in the pipeline awaiting expulsion. The value of *cnt_vo* can equal the latency only when clear has been asserted, meaning there has been a hard reset of the system or a pipeline flush has completed (figure 5.9).

Another method to produce *invalid_load* was investigated. The design for this approach is shown in figure 5.11. This approach simply indicates the case where valid inputs begin to be loaded dur-

Figure 5.11: Incorrect Generation of Invalid_load



ing a flush that was initiated after the system has reached steady state, meaning *valid_out* is already high. This means that valid_in falls and rises again while valid_out is still high. However, *alert* was not generated for the case where valid inputs begin to load before the completion of a flush that was onset during the filling of the pipeline. For this case,Valid_out would not yet be high, but there would still be an invalid load problem that would need to be indicated.

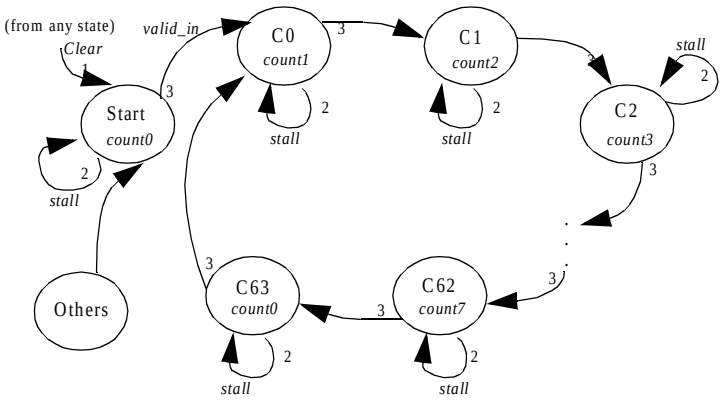
Invalid_block or *invalid_flush* create *alert_a*. This signal is fed to a D-ff to create a latch bounded, synchronous *alert* PO. This D-ff is essential to eliminate the timing dependency between primary inputs (PIs) and POs. In this case, we needed to isolate the PI, *valid_in* from the PO, *alert.* This issue is investigated further in the static timing section (8.0).

5.2.1.4 Finite State Machine

The second unit comprising the control stage is the finite state machine (FSM). The FSM is responsible for generating most the internal control signals in the design. It is a fully synchronous design following the Moore model with 65 defined states. The control FSM has been optimized for power by using the Moore model, in which all outputs are only a function of the current state instead of the Meally model, where outputs are a function of the state as well as the inputs. The Moore design eliminated glitching of outputs on asynchronous input transitions.

Figure 5.12 highlights enough of the FSM design to provide intuition on its functionality. With the

Figure 5.12: Control FSM

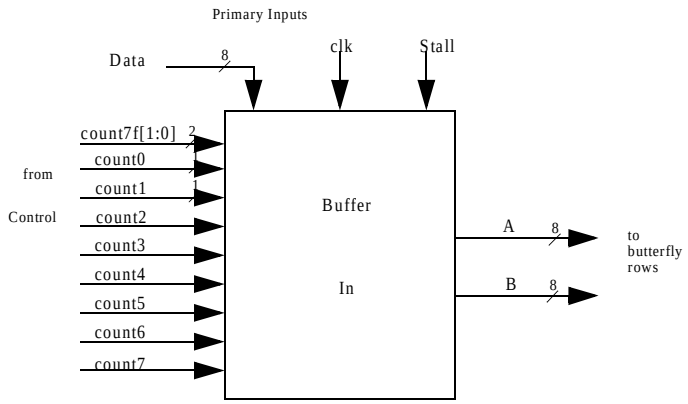


assertion of *Clear* from any state, the FSM will transition to the Start state. If *Clear* is not on, and *stall* is, then the FSM will hold its state. *Stall* is implemented as a clock gate. The FSM remains in Start until *valid_in* is asserted, at which point it can transition to the first of the "counting" states, C0. There are 64 "counting" states, C0-C63, that will transition in a sequential manner unless *clear* becomes asserted. Although there are many output signals defined by the states, only the *count0* through *count7* are shown in figure 5.12. These signals are asserted in a period 8 cyclic manner. Signals such as these, provide the correct enabling signals to the registers in all the other stages. Other signals such as *full64*, are only asserted in one state and therefore have a periodicity of 64 cycles. Two signals, *clear_RAC* and *cclear_RAC*, are only asserted one time after the system enters the "counting states." SR-ffs have been inserted to assert *clear_RAC* and *cclear_RAC* once the FSM has traversed through a particular state a certain number of times. After the FSM has reached state C18, *clear_RAC* will become high until *clear* is asserted. The second time through state C32, *cclear_RAC* will be asserted until the assertion of *clear.* These two aperiodic signals are needed for the RAC Rows and RAC Columns stages respectively.

5.2.2 Buffer In Stage

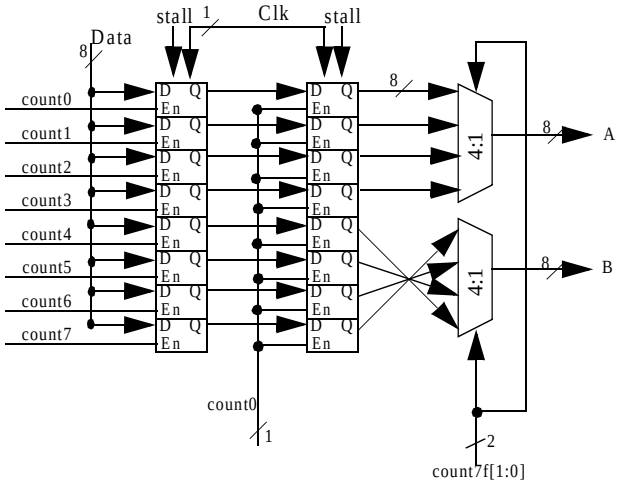
The first stage in the DCT computation datapath is the buffer in stage. A block diagram of the stage is provided in figure 5.13. The purpose of this stage is to convert the 1 byte per clock input

Figure 5.13: Block Diagram of Buffer In Stage



to 8 bytes every 8 clocks. This is necessary since all 8 pels are required at the butterfly stage to begin computation of the 1-D DCT. The implementation of this stage is illustrated in figure 5.14. There are two stages of eight, 1-byte D-ffs. The first stage is referred to as the sampling registers.

Figure 5.14: Buffer In Implementation



Every clock, only one sampling register is enabled at a time. They are enabled in a cyclic manner once every 8 clock cycles, as controlled by the outputs of the control FSM. Every 8 clock cycles, after the assertion of count7, the last of the 8 sampling registers has latched a new correct value. Then with the next clock, coubnt0 is asserted and all the values from the sampling registers are copied into 8 corresponding 1-byte computation registers. The outputs of the first 4 registers, corresponding to the first 4 sampled inputs, are fed in ascending order to a 1 of 4 multiplexor. The output of the multiplexor is the signal A. Signal B, is the output of a second 1 of 4 multiplexor, whose inputs are the last 4 sampled inputs in descending order. The period 4, cyclic select signal to the two multiplexors (muxes) is provided by outputs from the control FSM. The outputs of the muxes, A and B, provide the correct inputs to the next stage to either be added or subtracted in the butterfly computation. For the input pels, x_i, the output of the Buffer In stage over an 8 clock cycle period would be:

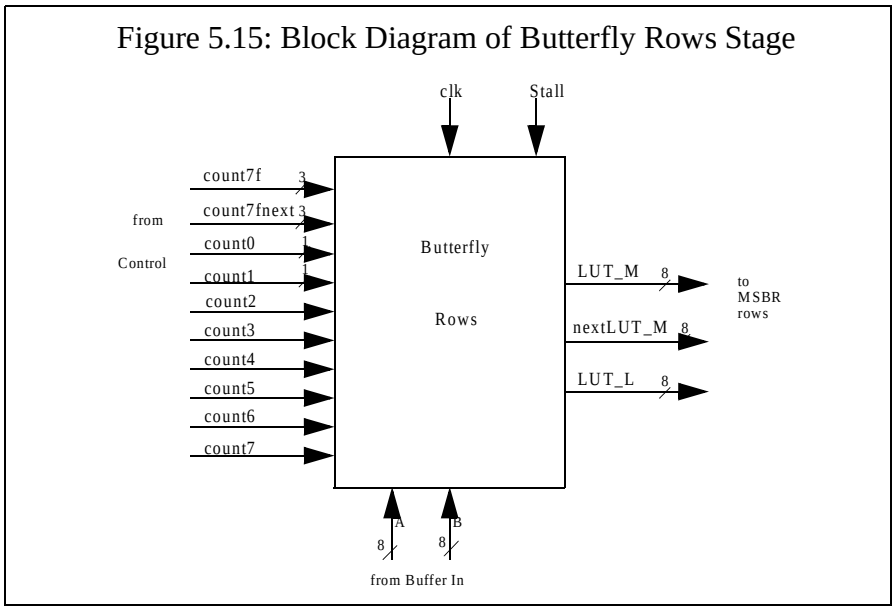
clock	A	B
1	x_0	x_7
2	x_1	x_6
3	x_2	x_5
4	x_3	x_4
5	x_0	x_7
5	x_1	x_6
7	x_2	x_5
8	x_3	x_4

5.2.3 Butterfly Rows Stage

The butterfly rows stage is responsible for computing the correct sums and differences of the inputs, A and B, from the buffer in stage. A block diagram of the unit's inputs and outputs is

shown in figure 5.15. The design the butterfly rows stage is shown in figure 5.16. For the 1-D

Figure 5.15: Block Diagram of Butterfly Rows Stage



DCT, the 8-bit values for A and B repeat with a period of 4. On the first period of A and B, the sum of the two is calculated. On the second period, the difference between A and B is calculated. The adder/subtracter subtracts whenever the sign input is asserted, otherwise it adds. The sign input is a control signal from the FSM.

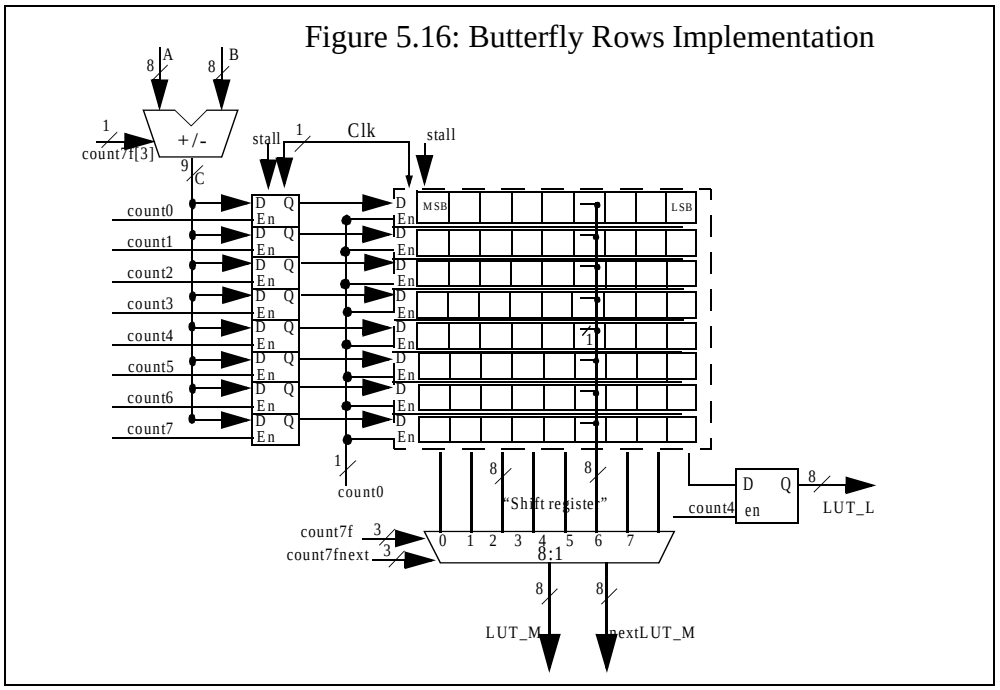
To maintain full precision, the result from the adder/subtracter is 9-bits. The result is stored in one of the 8, 9-bit registers that make up the butterfly stage sampling register stage. The sampling registers are enabled one at a time in a cyclic pattern that repeats every 8 clock cycles. The enabling control signals are provided by the FSM. It is in this manner that the following butterfly matrix is stored in the 8 sampling registers in the butterfly stage:

Reg0 x_0+x_7
Reg1 x_1+x_6
Reg2 x_2+x_5
Reg3 x_3+x_4
Reg4 x_0-x_7

Reg5 x_1-x_6
Reg6 x_2-x_5
Reg7 x_3-x_4

Every 8 clock cycles, a control signal from the FSM, allows all the values in the sampling registers to be copied into 8 corresponding, 9-bit computation registers. The computation registers are expanded to show all 9-bits in figure 5.16. In order to perform DA, a new combination of bits

Figure 5.16: Butterfly Rows Implementation



must be formed. One bit from the same magnitude position of each computation register is selected and adjoined. This wiring scheme creates the nine columns of bits that will be serially used to compute the correct dot products in the RAC stage. The most significant eight columns of bits are fed first to MSBR units to utilize the correlations between them.

However, the least significant column of bits is not fed through the MSBR units since there is the least amount of correlation between the LSB bits. It would not be worth the extra overhead to per-

form MSBR on this last column of bits. Because the LSB column of bits does not go through the MSBR unit, it must be delayed by the equivalent amount of time so that all inputs to the RAC stages arrive simultaneously. This is necessary to prevent erroneous results. This is done by feeding the column into a D-ff that is enabled to latch new data at the exact time that the other columns have arrived at the outputs of the MSBR stage. The enable signal is generated in the control FSM.

The serial input to the MSBR stage is generated in the butterfly stage. The most significant eight columns of bits from the computation registers are fed to two 8-to-1 multiplexors. The two multiplexors, which each output one column of bits every clock, give the effect of a shift register. Since they each have the same input busses, they have been drawn as one in figure 5.16. The two multiplexors have different selectors which are generated in the control FSM. The value of one selector is advanced by 1 from the other selector. Thus, one multiplexor will produce the current column of bits, *LUT_M*, and the other multiplexor will create the next column of bits, *nextLUT_M*. Both are needed in order to perform most significant bit rejection in the following stage.

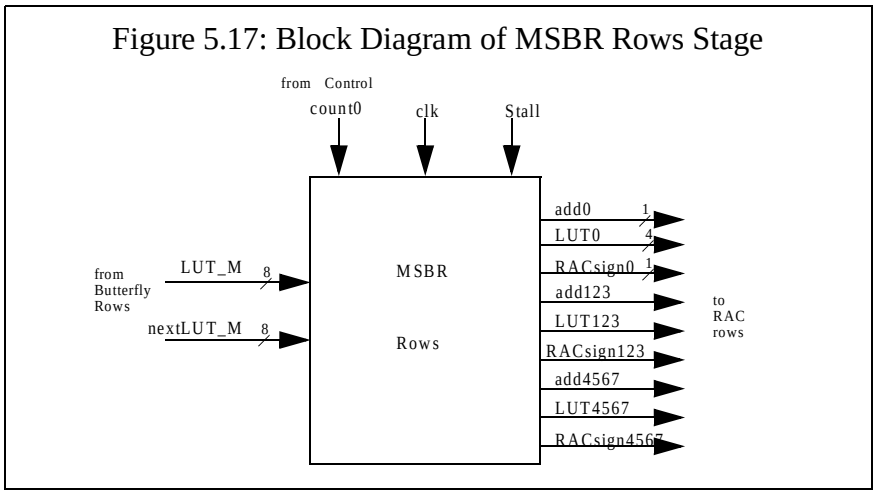
5.2.4 MSBR Rows Stage

The MSBR stages do not add any mathematical functionality, but rather, have been designed and inserted to make the system power scalable. It is the control unit that prevents needless switching in the look up table decoding, adder/subtracter, and 19-bit result registers in the RAC units. The MSBR unit can do this by generating signals that control the functionality of the RAC units. They

provide the bit columns of inputs to feed the LUTs (*LUTout*). They also determine whether the RAC structures should perform computation at all (*add*), and if so, when that computation should be a subtraction instead of an addition (*RACsign*). A detailed description of how the MSBR algorithmically works can be found in section 4.1.1.

MSBR Rows and MSBR Columns are two instantiations of the same design. Therefore, only a block diagram for the inputs and outputs of the MSBR Rows stage is shown in figure 5.17.

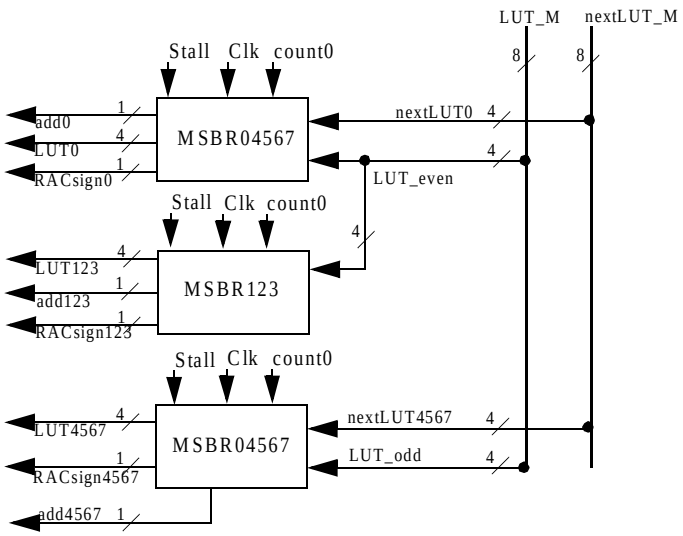
Figure 5.17: Block Diagram of MSBR Rows Stage



Parallel data paths are first encountered in the MSBR stage. Processing must be done in parallel to feed the 8 RAC structures with columns of bits at the same time. An inside look at the architecture

(figure 5.18) helps illustrate this. There are two MSBR04567 units, which are two different

Figure 5.18: Parallel Architecture of the MSBR Stages



instantiations of the same hardware. The two structures are identical since they screen for the same thing—unnecessary sign extension. This type of sign extension based bit rejection is performed on for column inputs multiplying the DCT matrix rows 0, 4, 5, 6, and 7. These rows, replicated from equation 3.3 are shown below:

$$\begin{bmatrix} A & A & A & A \\ D & E & F & G \\ E & -G & -D & -F \\ F & -D & G & E \\ G & -F & E & -D \end{bmatrix} \begin{matrix} \text{Row 0} \\ \text{Row 4} \\ \text{Row 5} \\ \text{Row6} \\ \text{Row7} \end{matrix}$$

One MSBR04567 unit receives the data for row 0, while the other unit receives the data for rows 4,5,6, and 7. The multiplying column vector input for row 0 corresponds to the summed pels, while the multiplying column vector for rows 4, 5, 6, and 7 corresponds to the differenced pels.

MSBR123 performs an all zeros or all ones based bit rejection for column inputs multiplying rows 1, 2, and 3 of the DCT computation matrix. These rows, replicated from equation 3.3 are

shown again below:

$$\begin{bmatrix} B & C & -C & -B \\ A & -A & -A & A \\ C & -B & B & -C \end{bmatrix} \quad \begin{matrix} \text{Row 1} \\ \text{Row 2} \\ \text{Row 3} \end{matrix}$$

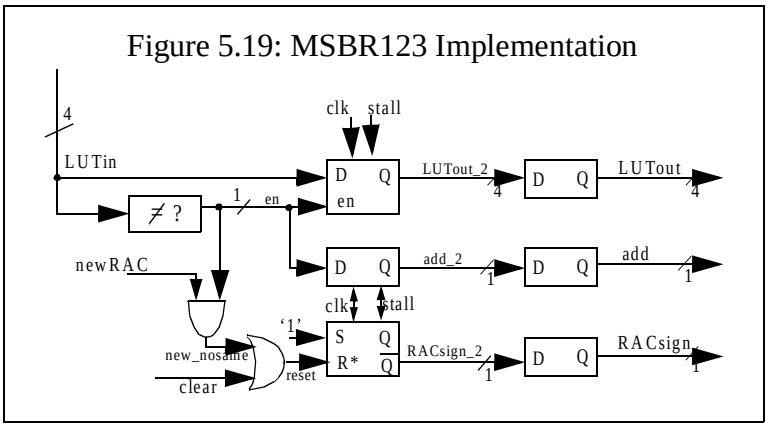
The column inputs that multiply these rows correspond to summed pels.

The 8-bit column busses, *LUT_M* and *nextLUT_M* that are inputs from the butterfly stage, are each split into two 4-bit busses. The 4-bit busses, *LUT_even* and *nextLUT0,* are formed from the most significant bits from *LUT_M* and *nextLUT_M* respectively. They contain the bits from the summed pels(x_0+x_7, x_1+x_6, x_2+x_5, x_3+x_4), which are used for the computation of the even coefficients. Hence, these busses are fed to the MSBR04567 unit that computes the dot product corresponding to the first row. Since the type of bit rejection performed by MSBR123 does not require the next column of bits, only *LUT_even* is fed to it. The other 4-bit busses, *LUT_odd* and *nextLUT4567,* are formed from the least significant bits from *LUT_M* and *nextLUT_M* respectively, and contain the bits from the differenced pels(x_0-x_7, x_1-x_6, x_2-x_5, x_3-x_4). These 4-bit busses are used to compute the odd DCT coefficients and are fed to the other instantiation of MSBR04567.

As described in section 4.1.1, MSBR123 is a simple function, that basically stops needless computation for columns of input bits (*LUTin*) that are all ones or all zeros. When this occurs *add* will be zero signalling no computation for the adder/subtracter (ALU) in the RAC units. Also, *LUTout* will be held steady to eliminate glitching in the LUTs and ALU. When *RACsign* is asserted it indicates the ALU in the RAC unit should subtract instead of add. The MSB column needs is the only

one that subtracting, and that too, only if the column of bits are not all zeros or all ones. This functionality was achieved with the implementation shown in figure 5.19.

Figure 5.19: MSBR123 Implementation



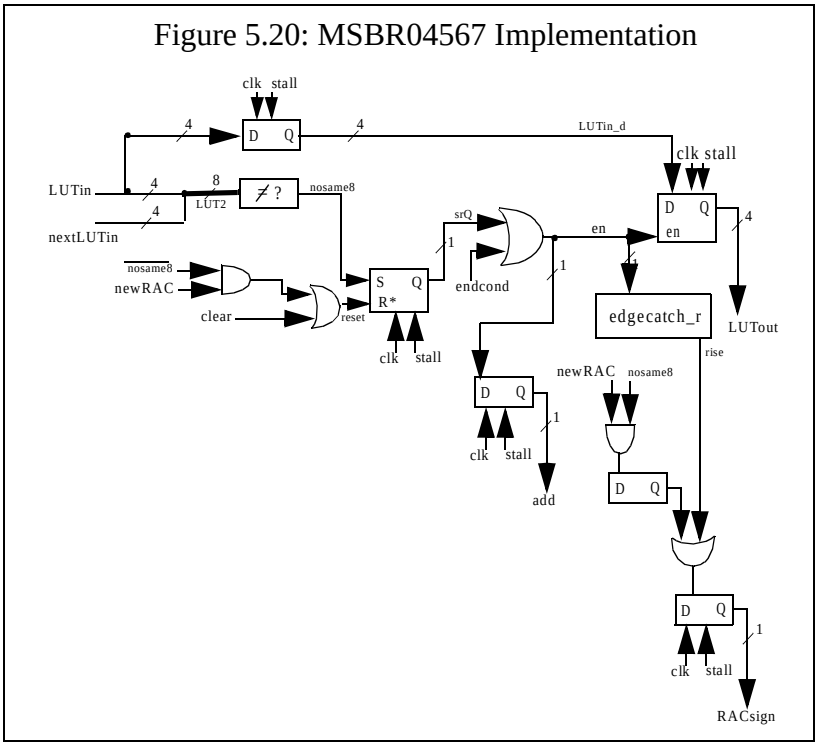
LUTin is only latched if all its bits are not the same. If the input bits are the same, the unit continues to output its previously held value. *RACsign* is generated through the negative output of a SR-ff. With the setup in figure 5.19, if the SR-ff is not explicitly being reset, RACsign will be low, signalling an addition function in the RAC. RACsign will be high whenever a new RAC begins and if that first column of inputs of most significant bits is being calculated (the input bits are not all equal). A new RAC computation is signalled by *newRAC,* which occurs in a cyclic manner every 8 clock cycles by the control FSM. *RACsign* will also be asserted when the SR-ff is reset with the clear signal. The second stage of D-ffs to convert *LUTout_2*, *add_2,* and *RACsign_2* to *LUTout*, *add*, and *RACsign* was inserted to match the latency for all outputs of the MSBR123 stage with that of all outputs from the MSBR04567 stages. This is necessary since MSBR123 and MSBR04567 units are in a parallel datapath.

The MSBR unit for the DCT matrix rows 0,4,5,6,and 7 is more intricate. The bit rejection is based on detecting extraneous sign extension (4.1.1) and stopping its associated computation. However,

the bit rejection for these rows cannot occur anywhere during the computation of that particular dot product. Once the end of sign extension has been detected, no more bit rejection can occur for the rest of that particular dot product calculation.

Figure 5.20 illustrates the design of the MSBR04567 component The detection of the sign exten-

Figure 5.20: MSBR04567 Implementation



sion is implemented by comparing the current column of inputs, *LUTin*, with the next column of input bits, *nextLUTin*. When new inputs arrive from the butterfly stage, if all bits of both inputs are either all zeros or all ones, it indicates that current column, *LUTin*, is part of the sign extension. Similar to a shift register, on the next cycle, *LUTin* will get what *nextLUTin* was and so on. Only when both columns are the same does the SR-ff become set. Once this is set, the rest of the columns of bits for that dot product cannot be part of the sign extension. Thus, once the input columns are no longer part of the sign extension, the D-ff will always be enabled to let the current column of bits, *LUTin* proceed to the RAC structure as *LUTout*. During the rejected cycles, the

output for *LUTout* will remain steady to keep the RAC units from switching. Also, after the SR-ff has been set, *add* becomes asserted to allow computation in the RAC for these columns of inputs. Furthermore, just when the SR-ff becomes set, component edgecatch_r will find the rising edge of *en*, and *rise* will become asserted to allow the most significant computed column of bits to be subtracted in the RAC instead of added. This subtraction is necessary for the MSB column since all values are 2's complement.

The boundaries between different sets of inputs had to be given careful attention for the MSBR04567 unit. The first column of inputs from a new data set arrives at each stage in the pipeline every 8 clock cycles as indicated by the assertion of *newRAC* from the control FSM. Similarly, the last column of inputs for each data set is indicated by the 8-cycle periodic assertion of *endcond* from the control FSM. These inputs were necessary to ensure that the sign extension detection did not carry through from one input group to the next. For example, if the end of sign extension had been detected in the first set of inputs, it should not assume that the second set of inputs has no sign extension. This problem is highlighted with the following example:

Input Set 1	Input Set 2
11110100	11111110
11110111	11111100
11110000	11111001
11110110	11111111

The MSBR04567 would detect the end of sign extension on the fourth column of inputs from data set one. Without *newRAC,* it would appear as if the inputs from data set two were part of the first set, and therefore columns 1-4 of the second data set would not be rejected even though they should be. This would mean that the MSBR would only reduce computation until the first sign

extension ended and never work again. Also, if the columns of bits from the two sets were viewed as one continuous stream, then the system would not assert *RACsign* correctly either.

To handle this case, the system begins checking for sign extension again from the beginning of the next set of inputs. This is accomplished by resetting the SR-ff whenever there is a newRAC and the first two columns of inputs are the same. This means that sign extension has been detected and to hold the previous outputs of the unit stable so no switching occurs in the RAC. If both columns are not the same, then there is no sign extension and computation can continue as before. For this case, there will be no rising edge of *en*, so *rise* will be 0. Therefore, additional logic was inserted to make *RACsign* assert under the condition when there is *newRAC* and the column of bits is not part of the sign extension, indicated by the assertion of *nosame8*. With this additional "OR" condition based on *newRAC*, there is no risk of RACsign asserting twice for the same input set. The SR-ff cannot be reset during that input set if the first MSB column is not part of the sign extension. Therefore, *en* will not transition to zero during that set.

The second boundary condition case that the design handled used *endcond* to make sure that an entire data set could not be rejected. This problem could occur if the design interpreted all columns of the data set as part of the sign extension. The following example illustrates this case:

Input Set:
11111111
11111111
11111111
11111111

If all columns were interpreted as the sign extension, then all four data inputs would be viewed as

the value 0 instead of -1. Measures were taken to ensure the design could not interpret all columns as part of a sign extension. *Endcond*, from the control FSM, is inserted to end the sign extension on the LSB column, when the SR-ff was not already set earlier for that input stream. This inhibited sign extension to be infinite and to stopped sign extension from carrying through to the next input set.

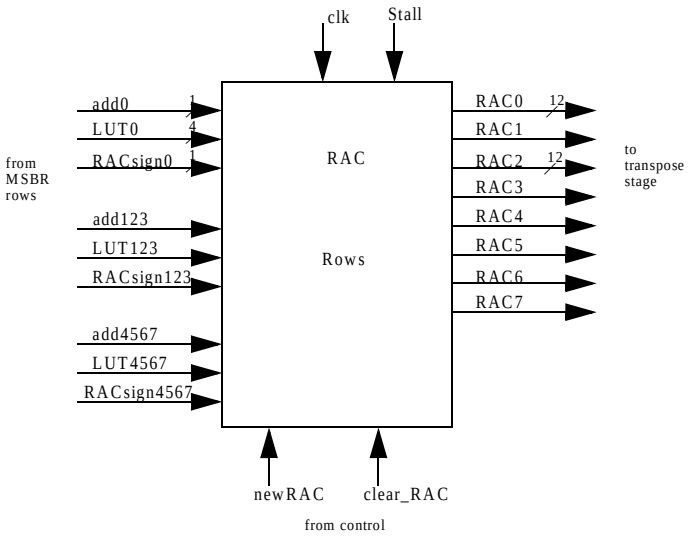
D-ffs have been inserted in the MSBR04567 stage to create equivalent latency paths of two cycles for all outputs of the unit--*LUTout, add, RACsign*. This provides the additional benefit that all outputs from the unit are latch bounded, thereby reducing the critical path in the following RAC stages.

5.2.5 RAC Rows Stage

The RAC Rows stage calculates the 8 dot products in the 1-D DCT algorithm (equation 3.3). It differs from the RAC Columns stage, which performs the same function, because it requires less bit width since it is earlier in the pipeline. The unit performs 8, 4-dimensional, dot products of distinct constant row vectors with 9-bit wide variable input column vectors. It uses DA for the computation method and is implemented with an adaptation of a RAC structure (3.3). Control signals as well as bit-columns of inputs are provided from the MSBR stage. Additional control signals are supplied by the control FSM. The the 8 dot product calculations result in 8 corresponding 12-bit values that are input to the transpose stage. The block diagram of the RAC Rows stage, highlight-

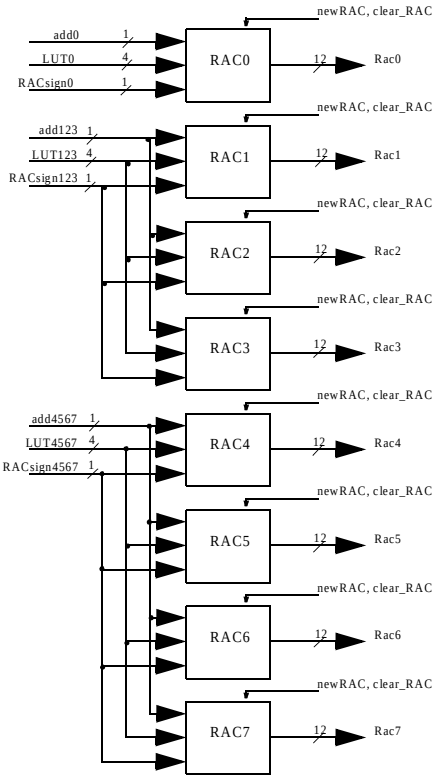
ing all the inputs and outputs, is given in figure 5.21.

Figure 5.21: Block Diagram of RAC Rows Stage



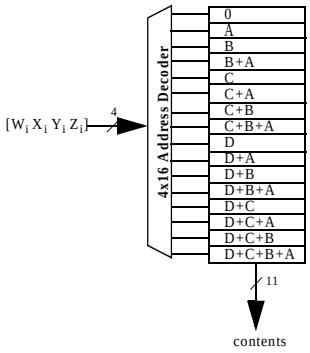
All results from the RAC Rows stage must be ready every 8 clock cycles. This requires parallel

Figure 5.22: Parallel Architecture of the MSBR Stages



datapaths with eight RAC units as shown in figure 5.22. The RAC units are identical in structure and design. However, they differ in that each has a unique LUT that stores different combinations of constants corresponding to a particular row of the DCT matrix. For example, if the constant row in the dot product were [A B C D], and one column of variable input bits were $[W_i\ X_i\ Y_i\ Z_i]$', figure 5.23 shows the configuration for the corresponding LUT in the RAC unit. The assertion of

Figure 5.23: LUT Design for Constant Row [A B C D]

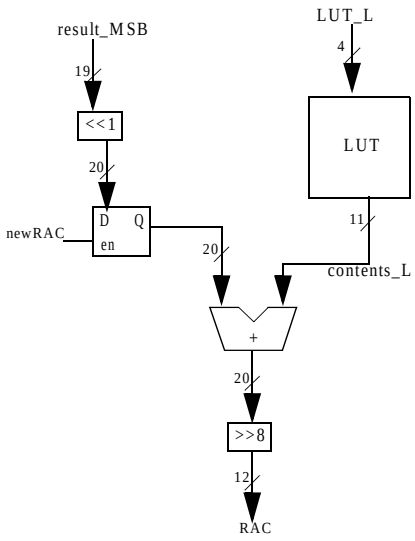


W_i indicates that A is included in the sum. Similar correlations exist between X, Y, Z and B,C, D. The four basic constants comprising each LUT were given in equation 3.3. In order to represent the irrational constant values as integers, the constants were individually multiplied by 256, which corresponded to a left shift by 8. Following the DA computation, dot product results were shifted back right by 8 to compensate for the constants. The range of sums, resulting from the combinations of the constant row elements from the DCT matrix, was -443 through +724. To represent this range, a width of 11-bits for the LUTs in the RAC structures was used.

While there are 8 RAC structures computing in parallel, there is further parallelization within each individual RAC structure. With a fully serial RAC structure, it would take 9 clock cycles the answer for a 9-bit input to converge. Due to the pipelined structure of the system, the RAC struc-

tures must finish computation within 8 cycles so data corresponding to a new set can begin every 8 cycles. To utilize the least number of resources possible, while still meeting the performance requirement, the RAC units are partially parallelized into two units as shown in figure 5.24. The

Figure 5.24: Partially Parellelized RAC structures for Rows



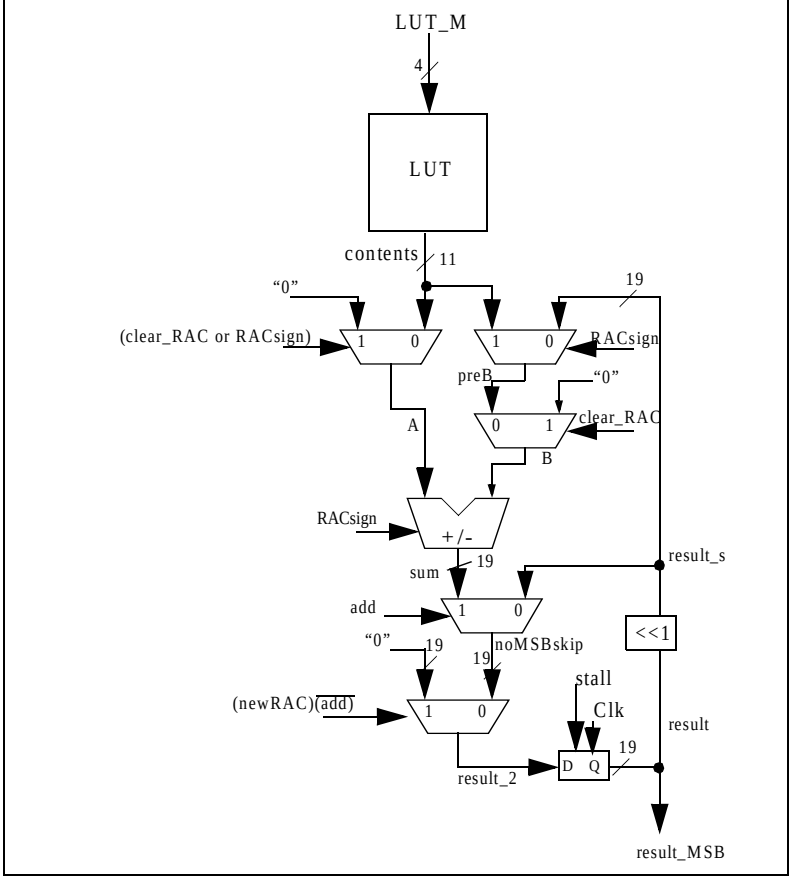
first unit, operating on the 8 MSB columns converges in 8 clock cycles. The second unit, which operates on the LSB column of bits will converge concurrently, but in only 1 clock cycle. The MSB result must be shifted left by 1 before it can be added to the result of the LSB calculation. Since the answers converge at different times, the result of the MSB calculation is latched to ensure no glitching in the adder until the computation of the MSB calculation is complete. Results from the adder are shifted right by 8 following all computation on the 1-D DCT to compensate for the constants that were multiplied by 2^8 in the LUT.

The LSB unit is simple. Since it only calculates on one column, there are no sums that need to be made. As is shown in figure 5.24, the unit consists only of a LUT. This is possible because there is no bit rejection on the LSB column. The inputs to this subunit arrive directly from the Butterfly

Rows stage instead of the MSB unit. Inserting extra logic in this small RAC structure to implement bit rejection did not seem worth it for the LSB column since there is the least amount of correlation at the LSB position of input pels.

The MSB unit adopts a RAC structure (3.3) to compute on the first 8 columns of inputs. Five multiplexors have been inserted into the basic RAC structure to include the additional paths necessary to implement MSBR and initialization. There are 5 basic functions that require different paths

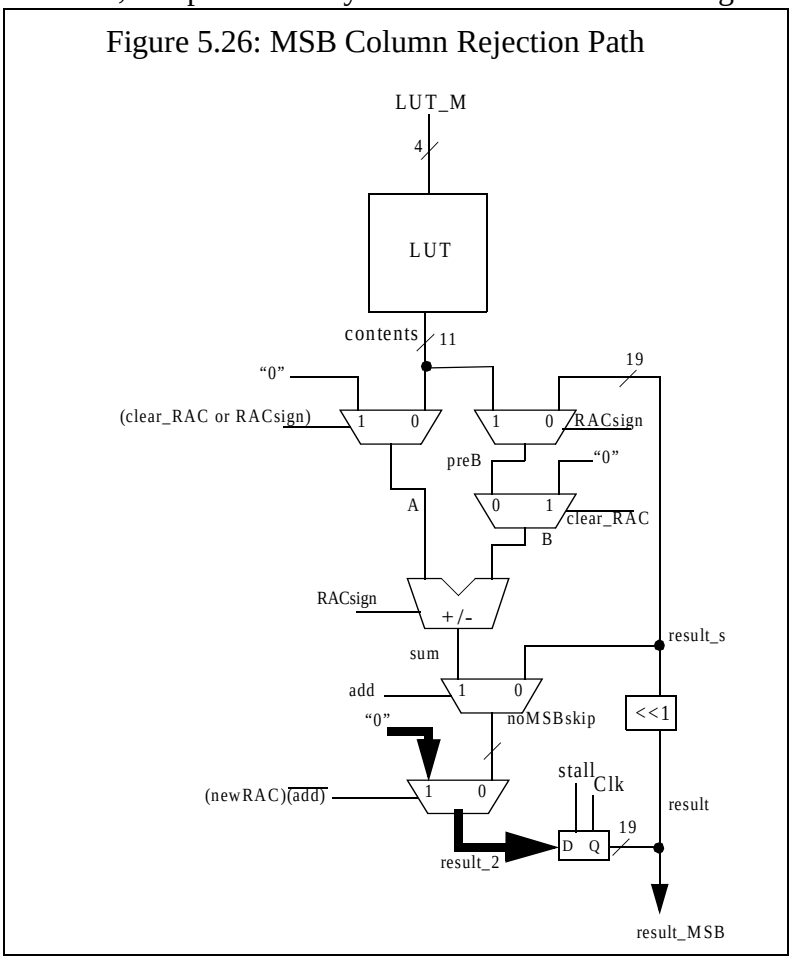
Figure 5.25: MSB RAC Rows Implementation



through the MSB RAC structure: calculation on the MSB column, rejection of the MSB column, calculation of non-MSB rows, rejection of non-MSB rows, and initiation of the unit.The design for the 8 MSB RAC Rows units is given in figure 5.25.

Inputs arrive to the unit serially from the MSBR Rows stage with the MSB column arriving first. The most significant column of bits is signalled with *newRAC*. It may be part of the extraneous sign extension and may be skipped. In this case, *add* will not be asserted from the MSBR unit and the result register will be cleared to zero. It is important that the previous data set result is cleared from the feedback to the ALU so it cannot contaminate the current data set calculation. This path through the unit in which calculation on the most significant column of bits is rejected is highlighted in figure 5.26. Note, this path can only be taken at most once during the 8-cycle calcula-

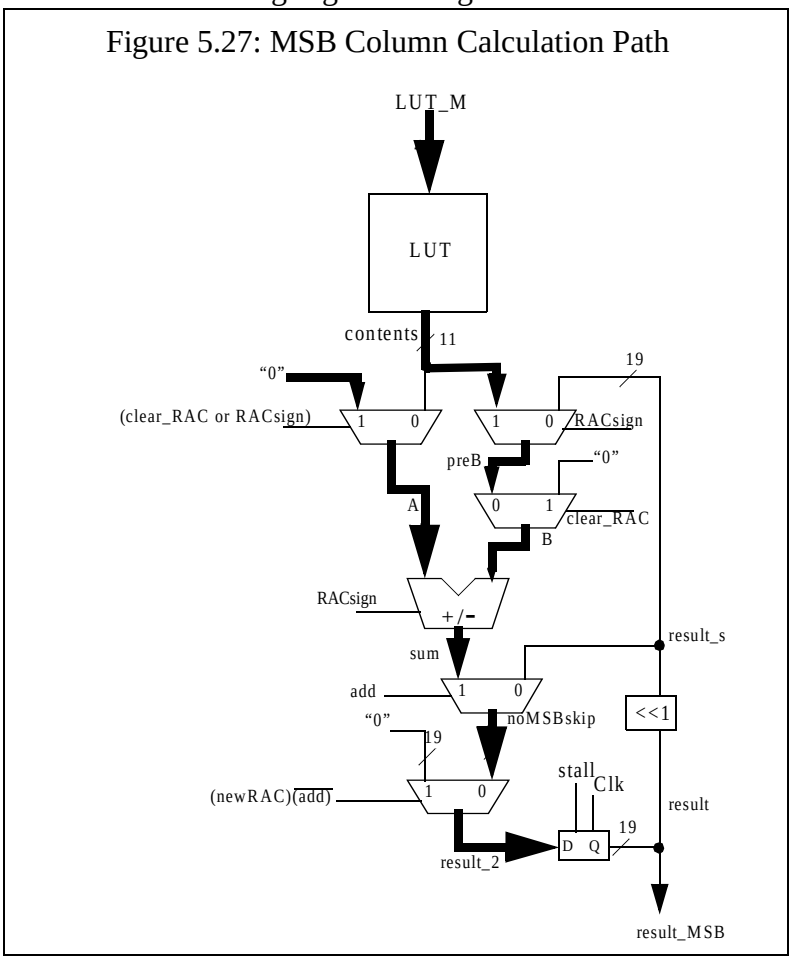
Figure 5.26: MSB Column Rejection Path



tion of one dot product.

The first column of bits that is not rejected indicates the sign of the 2's complement inputs. Therefore, only for this first computed column, the ALU in the MSB RAC unit is set to subtract instead of add. This is signalled by the MSBR Rows stage through the assertion of *RACsign*. Care is taken such that the first computed column for the data set is not subtracted from what is stored in the result register. This would contaminate the calculations of the current dot product with the previous dot product calculation. Instead, at the end of the clock cycle, the result register contain the negative contents of the LUT that was addressed with the first computed input column of bits. The data path associated with this case is highlighted in figure 5.27.

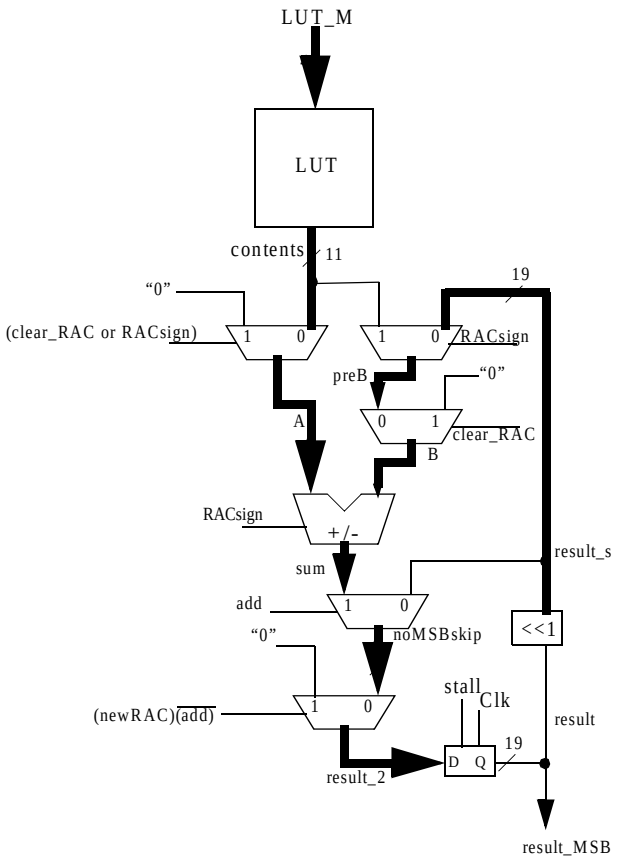
Figure 5.27: MSB Column Calculation Path



Once the end of sign extension has been determined, computation begins. The first column com-

puted is the most significant and will be subtracted as was shown in figure 5.27. For the remaining columns of bits in the data set, a different path is followed through the unit. The current contents of the LUT are summed with the left-shifted results from the previous look up. This path through the unit is illustrated in figure 5.28. We know the column is not rejected since *add* will be set high

Figure 5.28: Computation of Non-MSB Column

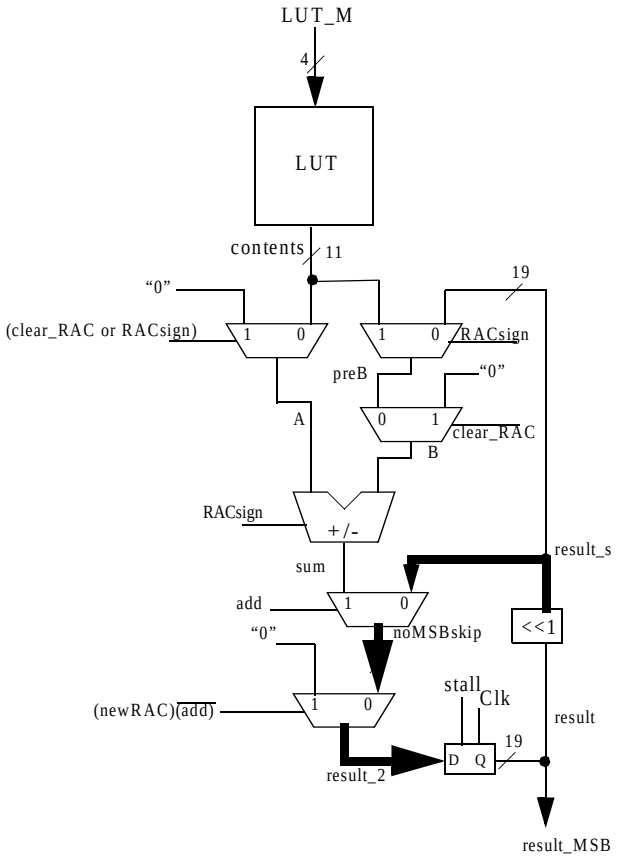


in the MSBR Rows stage. Furthermore, *RACsign* will be low, to detect that the most significant column is not being input for computation.

Another path is necessary to reject computation on columns that do not correspond to the MSB. In this case, the inputs to the LUT and adder are held constant by the MSBR Rows stage so glitching will not occur. Since computation is being rejected, the ALU is bypassed. The result register still

grabs new data that corresponds to the previous result multiplied by two. This is necessary since each column of inputs must be shifted by the correct amount corresponding to its position in the input vector. The path for this case, where computation is rejected for columns other than the most significant column, is highlighted in figure 5.29.

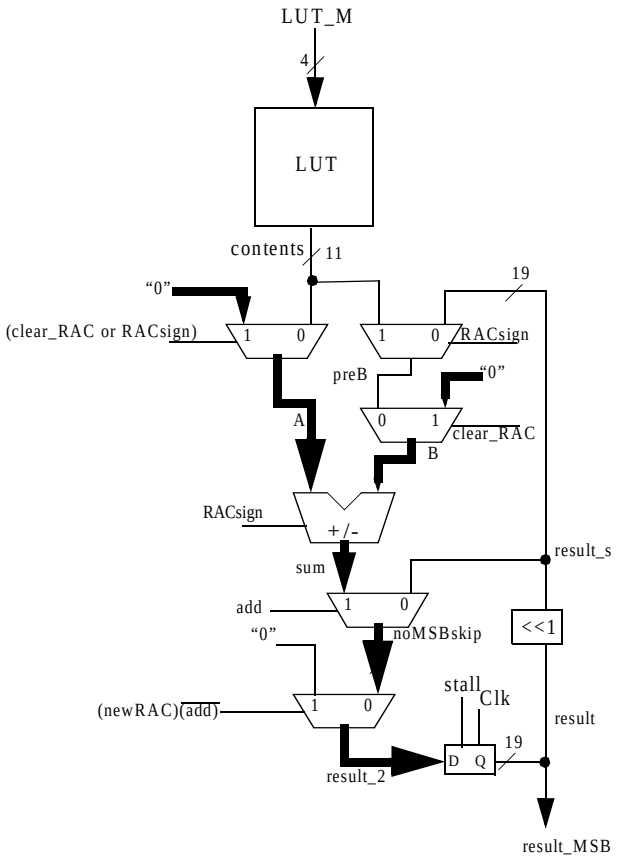
Figure 5.29: Rejection of Non-MSB Column



The designed structure required initialization before valid computation could begin. This was necessary to flush X's out of the feedback loop to the adder. This was achieved through the path illus-

trated in figure 5.30. Initialization occurs once during the filling of the pipeline. The unit cannot

Figure 5.30: Initialization of the RAC

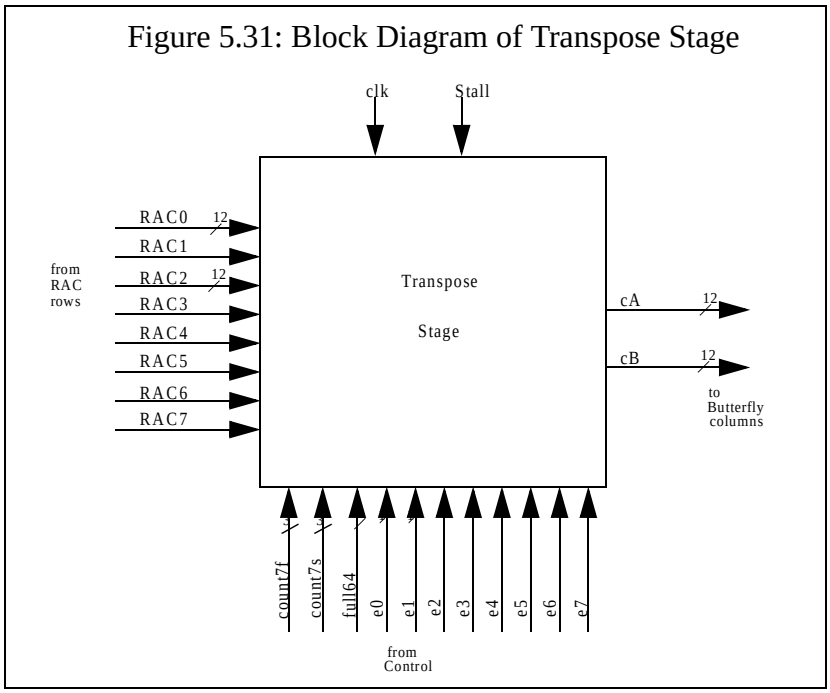


be initialized with system Clear, because X values still flow through the unit until all previous stages are overwritten with valid inputs. It would have been an option to reset all flip flops in the design at zero, but that would have created too much unnecessary logic. Initialization of this unit occurs some latency after the assertion of *Clear*, exactly at the time when valid inputs arrive to the unit. When valid inputs begin to arrive, the control FSM asserts *clear_RAC*. This way, A and B will be defined signals at the same time so the ALU result will also be defined at zero right before it is used initially.

5.2.6 Transpose Stage

The transpose stage collects the results of 8 sequential 1-D DCT calculations. Each 1-D calculation has 8 results. All results from 1-D calculations arrive from the RAC Rows unit periodically every 8 clock cycles. Since all 8, 12-bit results from a 1-D calculation arrive at the same time, the transpose unit writes all 8 input values at once. The transpose stage is filled with correct data every 64 clock cycles, as indicated with the control FSM signal *full64*. The inputs and outputs of this stage are shown in figure 5.31.

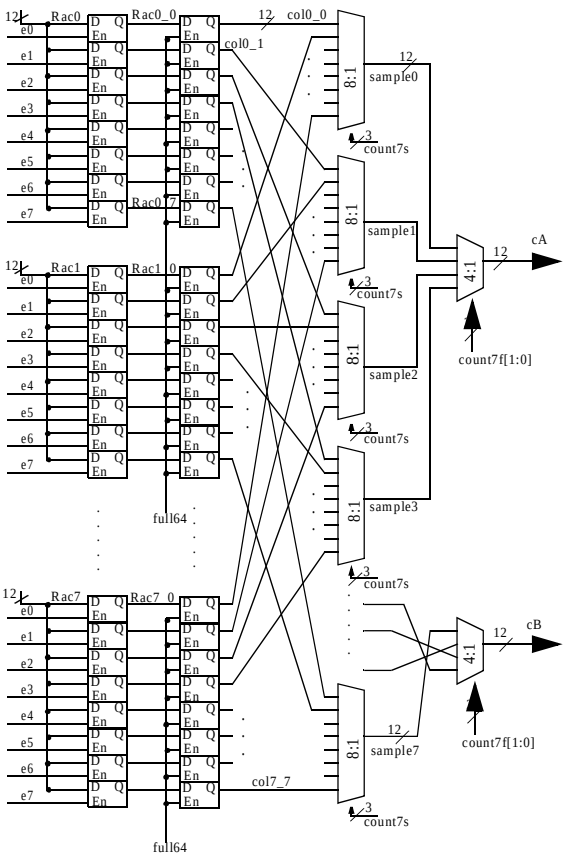
Figure 5.31: Block Diagram of Transpose Stage



The transpose stage consists mainly of two sets of registers: sampling registers and computation registers. There are 64, enabled, 12-bit D-ffs composing each set. Data is copied to the computation registers from the sampling registers when all the data in the sampling buffers have been overwritten with results from 8 new 1-D DCT computations. Each of the 8 inputs to the stage are hardwired to 8 possible D-ffs in the sampling register set, only one of which will be enabled one

at a time. The enabling of the registers is handled with signals *e0, e1, e2, e3, e4, e5, e6, e7* from the control FSM which each have a period of 64, and are offset from one another by 8 cycles. The architecture of this stage is illustrated in figure 5.32.

Figure 5.32: Transpose Stage Implementation



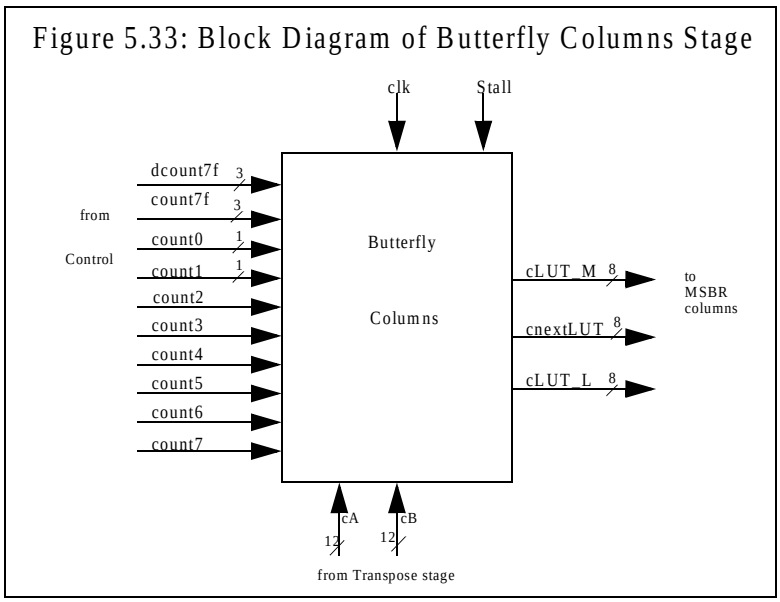
The transpose matrix also serves the same function as the buffer in stage had. From the 64 stored elements in the computation registers, 8 must be selected at a time for 1-D DCT computation. The 8 samples are chosen via 8, 8-to-1 muxes that select the same element from each of the row computations. For example, the first column computation would correspond to the RAC0 results from the 8 different computations of the 1-D DCT on each row. The outputs of the first stage of multiplexors, *sample0, sample1, sample2,* up to *sample7,* correspond to one column. A new column is

selected every 8 clock cycles with the *count7s* signal from the control FSM. The period of this signal is 64 cycles. The second stage of multiplexors are controlled with the bottom two bits of count7f, which has a period 4 cycles. These two 4-to-1 multiplexors extract the elements to be added or subtracted in the butterfly computation on the columns. They follow the same order as was discussed in the buffer in stage (5.2.2).

5.2.7 Butterfly Columns Stage

The function of the butterfly columns stage is the same as the butterfly rows stage(5.2.3). However, because it is later in the pipeline, it must function on increased bit-width inputs and therefore required some design modifications. The inputs and outputs from the stage are given in figure 5.33.

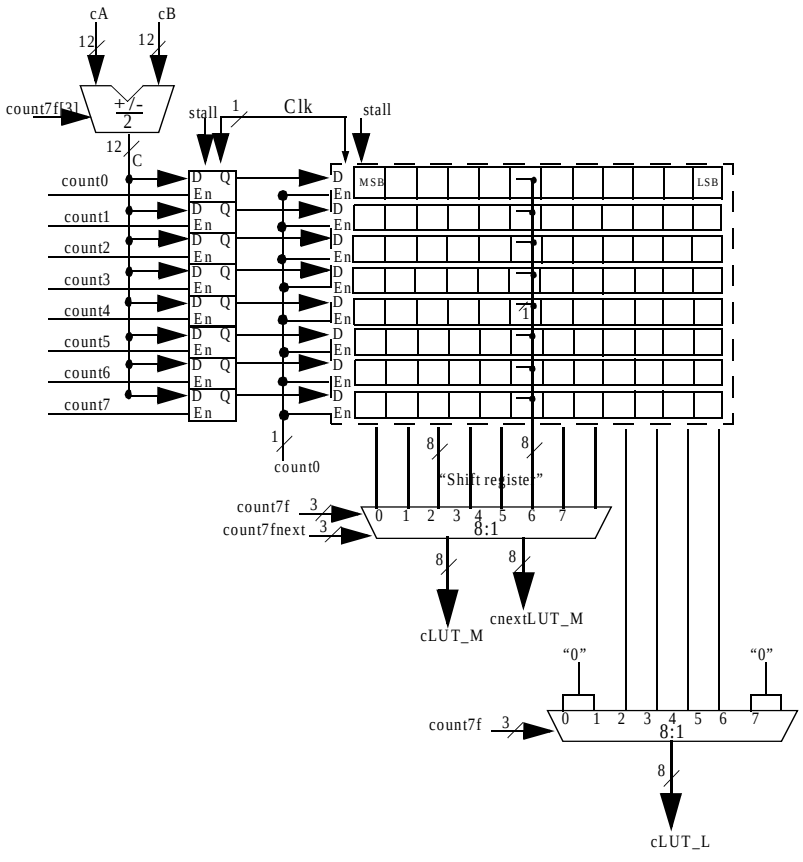
Figure 5.33: Block Diagram of Butterfly Columns Stage



Although the inputs are 12 bits, the result of the addition or subtraction in the unit is left at 12 bits. This adder/subtracter design accounts for one of the divisions by two mandated by the algorithm.

Compliancy testing (6.1) determined that dropping the one bit of accuracy at this point in the pipeline would not sacrifice image integrity. Therefore, the unit only needed to extract 12 columns of bits to pass on for further computation. Only the 8 most significant columns are subject to bit rejection and will pass through the MSBR Columns unit. The 4 least significant columns of bits have less correlation with one another and are not worth inserting the extra logic in the LSB RAC structure or MSBR Columns unit necessary to perform bit rejection on those columns. The design of this stage is illustrated in figure 5.34. The first 8 columns of bits are extracted and sent

Figure 5.34: Butterfly Columns Implementation



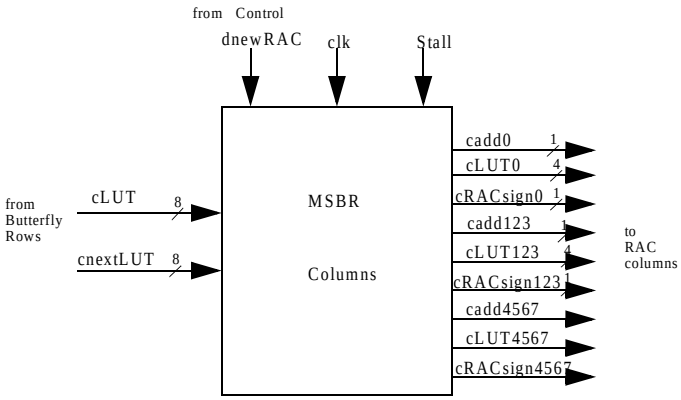
to the following MSBR unit in the same way as described in section 5.2.3. The least significant 4 columns are passed along to the RAC columns stage in a slightly different way. Since they are not subject to bit rejection, only the current column of bits is needed at a time. An 8-to-1 multiplexor

selects the columns most significant first with *count7f* from the control FSM. The most significant of the LSB columns is output after 2 cycles to account for the latency of MSBR Columns unit.

5.2.8 MSBR Columns Stage

The next stage in the pipeline has the inputs and outputs shown in figure 5.35. The MSBR Columns stage has an identical design as the MSBR Rows stage (5.2.4) and performs the same function.

Figure 5.35: Block Diagram of MSBR Columns Stage

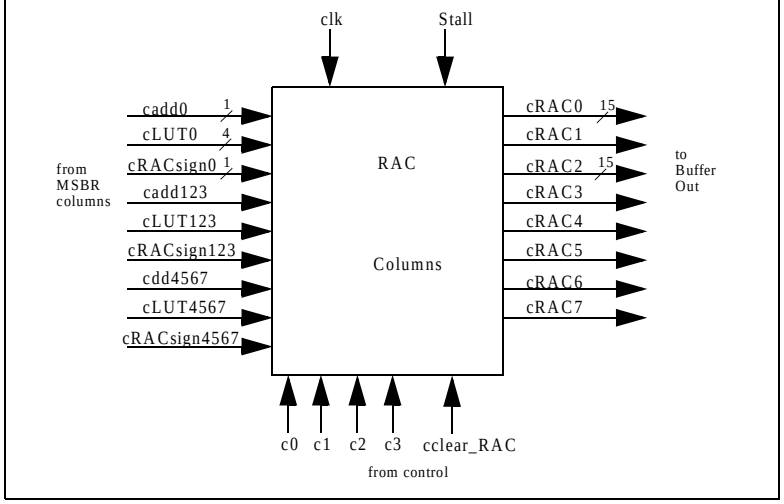


5.2.9 RAC Columns Stage

The RAC Columns Stage is very similar to the RAC Rows stage, except it computes on 12 columns of bits instead of 9. Since the dot product still needs to converge every 8 clock cycles, greater parallelization is adopted. This requires additional control signals from the FSM. The

inputs and outputs of this stage are shown in figure 5.36. Similar to the RAC Rows stage, there are

Figure 5.36: Block Diagram of RAC Columns Stage

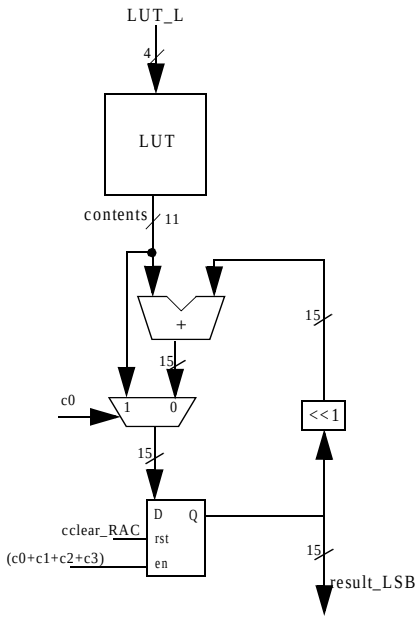


8 parallel RAC structures, each with a distinct LUT, to compute the 8 dot products (figure 5.22). Again, each of the RAC units is composed of two parallel units: the MSB RAC unit and the LSB RAC unit. The MSB RAC unit, which gets its inputs and control signals from the MSBR Columns stage, computes partial product from the 8 MSB columns of bits and implements bit rejection on them. The design is identical to the MSB RAC unit from the RAC Rows stage illustrated in figure 5.25. The other unit, LSB RAC, computes the partial product from the 4 least significant columns of bits that arrive directly from the Butterfly Columns stage.

The LSB RAC unit in the columns stage is different from the row stage because it computes the partial product of 4 columns of bits instead of just 1. Therefore, the design is more complex than just a simple LUT as it must contain registers and an adder. There are no subtractions necessary since the most significant column, which provides the sign information, is calculated in the MSB

RAC Columns unit. The design of the LSB RAC Columns is given in figure 5.37. Note that it

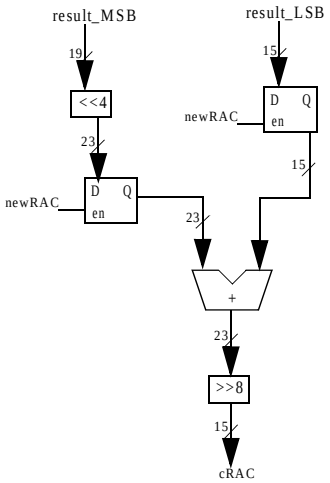
Figure 5.37: LSB RAC Columns Implementation



does not implement MSBR. Following a clear of the system, the values within the feedback loop must be initialized at the correct time just as valid inputs arrive at the unit. Assertion of *clear_RAC*, from the control FSM, handles the initialization by loading zeros into the register and feedback path. At the beginning of each new dot product computation, the unit does not add to the previous result, but directly loads the contents of the LUT for the first column into the result register. The 2-to-1 mux, with select signal c0 from the control FSM enables this behavior. Since the latency of the LSB RAC unit is less than the latency of the MSB RAC unit, there are 4 extra clock cycles after the partial product has converged in the LSB RAC unit. To keep *result_LSB* from changing during the remaining 4 clock cycles, the register is only enabled 4 times until the answer from the 4 columns has been computed. Since *result_LSB* cannot change after 4 cycles, and inputs from the Butterfly Columns stage will be zero, the adder is kept from glitching.

The final partial products from the MSB and LSB RAC Columns units are combined in the manner illustrated in figure 5.38. The MSB partial product, *result_MSB*, must be shifted by four posi-

Figure 5.38: Combining the Column MSB and LSB RACs

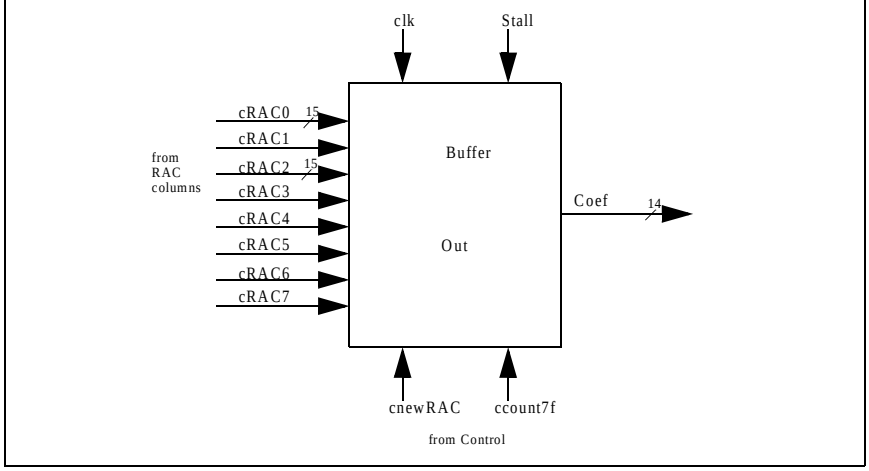


tions to left to get the right magnitude before it can be added with the least significant partial product. The D-ffs are inserted to keep the adder from glitching before both partial products have converged. The result from the MSB RAC unit, *result_MSB*, changes 8 times until it converges and similarly, *result_LSB* changes 4 times. Once *result_MSB* has converged, both D-ffs are enabled. The result of the addition is shifted to the right by 8 to compensate for the integer representation of the constants in the LUT.

5.2.10 Buffer Out Stage

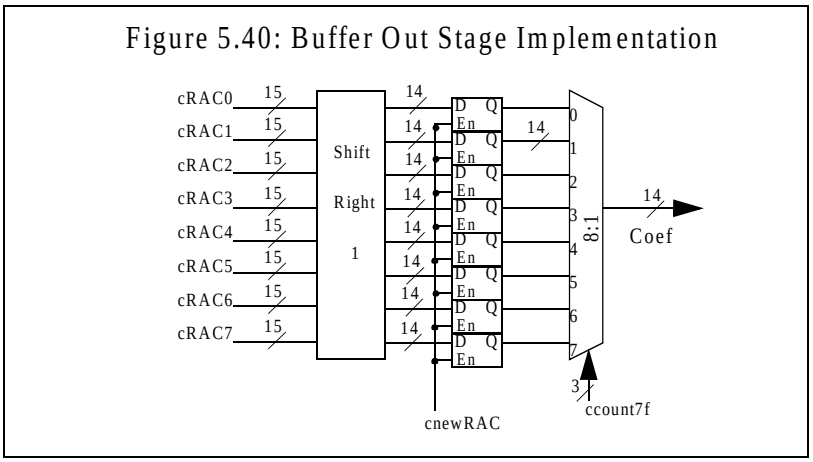
The Buffer Out stage is the final stage in the pipeline. A block diagram of the stage is given in figure 5.39. Buffer Out receives 8 new dot product results from the RAC Columns stage every 8 clock cycles as is indicated with *cnewRAC* from the control FSM. It outputs one final 2-D DCT coefficient every clock in column-major order.

Figure 5.39: Block Diagram of Buffer Out Stage



The design for the stage is illustrated in figure 5.40. Every 8 clock cycles, the results from the

Figure 5.40: Buffer Out Stage Implementation



RAC Columns stage are shifted right by 1 to satisfy the second division by 2 mandated by the algorithm. Then, the 8 shifted results are latched at once into 8 corresponding D-ffs. The shifting was intentionally done before the results were stored to reduce the cell count as well as the power associated with switching the flip flops. The coefficients are read out from the flip flops one at a time with every rising clock edge. The correct coefficient is selected for output with an 8-to-1 multiplexor. The output coefficient, *Coef*, from the 2-D DCT is a 14 bit primary output of the system.

6. Verification

The verification strategy consisted of development of VHDL testbenches as well as a higher-level, bit-exact C model of the datapath. The C model was necessary for several reasons. First, it enabled the tracking of key signals within the datapath to highlight the sources of several errors in the VHDL implementation. Simple and quick modifications of the C model helped determine the changes to the architecture necessary to meet the JPEG Standard accuracy requirements. For this reason, it was important that the C model was bit-exact. Secondly, other JPEG components will be modelled in C in the future. When the DCT is included in the JPEG design, the C model will be useful for quickly verifying the entire system together. Furthermore, when the JPEG core is tested with other cores, a C model is even more essential to reduce the lengthy simulation times.

Although implemented differently, the signals generated at key interfaces are designed to be bit exact between the C and VHDL models. This was desired for testability and debugging purposes. Although bit-exact at the algorithmic level, the C model is implemented differently. The C model lacks bit rejection units as well as the control logic. The C model does not handle the reset, stall, or flushing ability involved for pipeline functionality.

Initially, the C and VHDL models were individually debugged for general functionality. The two models were tested pipeline stage against pipeline stage with simple, customized tests that had predictable results. The results for the VHDL models were viewed through MTI simulation. Mathematical subunits for both models were debugged by comparing test case results from the

design under test (DUT) against MATLAB results. General algorithmic functionality was established by running hand-coded input blocks through both the C and VHDL models. The results were qualitatively compared to results generated from driving the DCT function in MATLAB with the same input vectors.

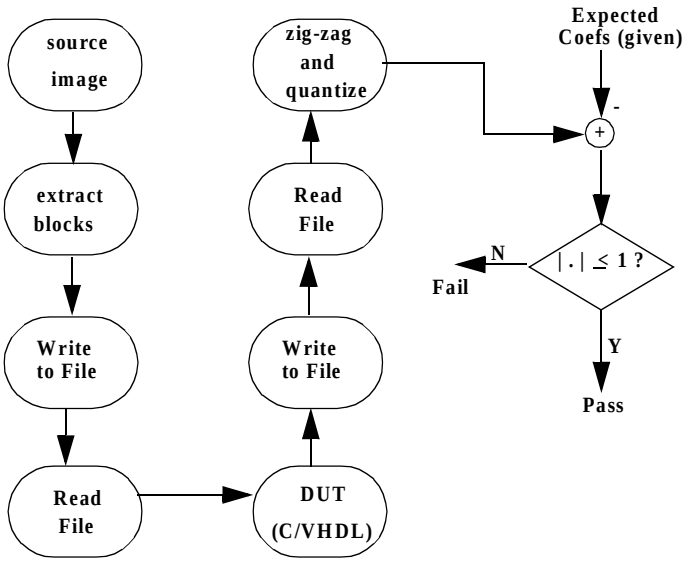
Furthermore, the C model and VHDL models were debugged against each other. This allowed the discrepancies between the two models to be debugged, ensuring the two models would be bit-exact.

6.1 Compliancy Verification

Next, compliancy testing on the debugged C model ensued. Compliancy testing followed the guidelines outlined in the JPEG still image compression standard ISO DIS 10918-1 for baseline sequential operation. The standard provided files of four different baseline source images: A, B, C, D. For each of the 4 source images, 4 corresponding quantization tables and 4 files containing reference, quantized DCT coefficients were provided. The test coefficients, generated from running the provided source images through the DUT, were quantized with the corresponding quantization table. To be compliant with JPEG standards, the result had to be within 1 from the provided quantized reference DCT coefficients. Combined, the four images contained a total of 937 blocks or 59,968 coefficients, all of which were checked through compliancy testing.

The compliancy testing is pictured in figure 6.1, The processing environment was automated in C.

Figure 6.1: Structure of Compliancy Tests



C programs read in the corresponding compliancy image sequentially in row major order from the source file, arranged the pels in block sequential order, and wrote the new ordering to an intermediate file. The file was then read by the DUT, which was implemented either in C or VHDL, depending on which model was being verified with compliancy testing. The testbench for the DUT wrote the coefficients to another intermediate file. Other C programs were responsible for the rest of the compliancy testing. The programs read in the DUT coefficients from the respective file, arranged them in zig zag order, quantized the DUT coefficients, and then determined whether they quantized coefficients were within the acceptable range of error. The standard provided four files for coefficients corresponding to baseline images A, B, C, and D for comparison. It also gave the unique quantization matrix that was associated with each image. Compliancy checking was not done at run-time; results from the DUT were written to a file and read in afterwards for checking. This alleviated the need of a foreign language interface between C and VHDL.

It is important to emphasize that assessment of compliancy testing, as defined in the JPEG standards, was on the quantized coefficients. Furthermore, the provided reference coefficients also assumed that input pels had been zero-shifted before being fed into the DCT DUT. The zero shifting centered the data around zero, thereby decreasing the necessary bit-width of the design. Quantization had a similar effect. It gave the DUT some slack in accuracy, or reduced the necessary bit-width of certain stages in the pipeline, by accounting for the insensitivity of the human visual system towards certain frequencies.

The defined method for quantization in the JPEG standard is also worth mentioning, as it affects compliancy results, and hence, precision requirements for the DUT. All the coefficients from the DUT are integer values as are all the elements in the quantization tables. The quantized coefficients are also mandated to be integer values. The result from the division of the coefficient with the quantizor is initially a floating point number that will be rounded to the nearest integer. This is done in C by first adding one-half to the floating point number, and then chopping off the decimal portion of the floating point number. Effectively, this integer quantization essentially loosens the requirements for compliancy from a margin of error of 1 up to 1.5.

The edge conditions of the image also affected compliancy determination. The four input images were sized: 85x65, 85x129, 255x65, and 85x257, which could not be divided into sequential, non-overlapping 8x8 input blocks. Using JPEG recommendations, the edges of each of these four images were extended to make full 8x8 blocks for input to the DUT. The extension was done by replicating the last column or row as many times as necessary to make the smallest image possible

that could be divided into 8x8 sequential blocks. The sizes of the four images were extended as part of the C testbench to 88x72, 88x136, 256x72, and 88x264 respectively.

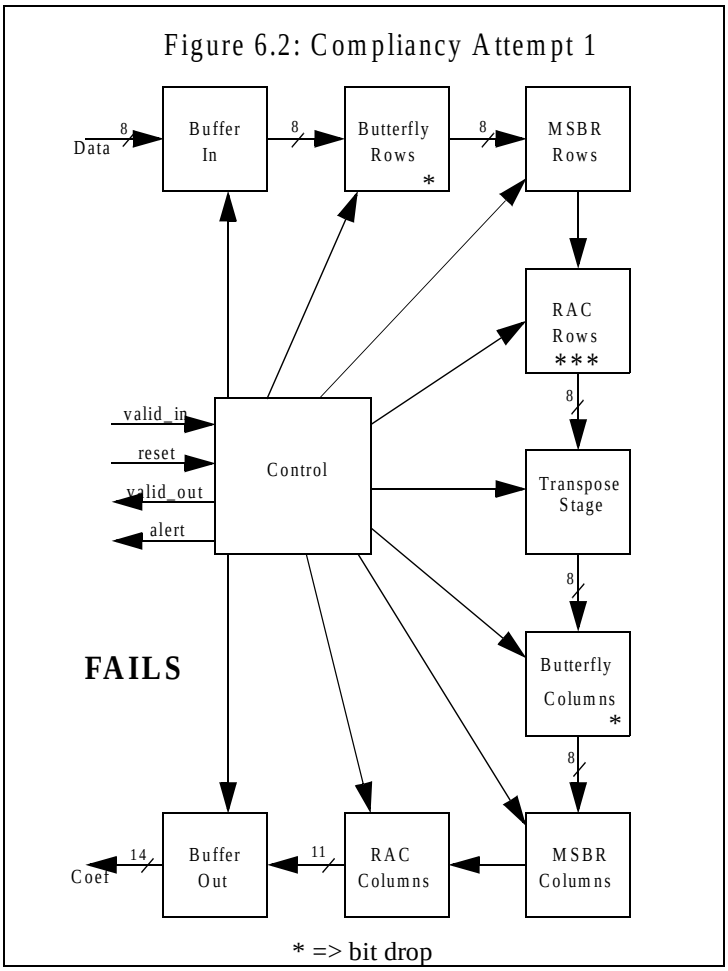
6.1.1 Architectural Attempts

Several stages of modifications were made to the C model of the DUT to find the correct balance between resources and the necessary accuracy to pass compliancy testing. The C model was used for this experimentation since it was easier to modify and update. The accuracy of the design was most significantly affected by the positioning of the bit drops within the datapath. The chosen Chen and Fralick algorithm (equation 3.3) required a division by 2 for each dimension. Hence, two divisions by two, implemented as right shifts, or bit drops, needed to be performed within the design. One bit drop accounted for the division by 2 on all the rows, and the second shift accounted for the division by 2 on all the columns. Since it was not specified in which order the divisions by two needed to occur, the different modifications examined the accuracy impact of performing these bit drops at different positions in the datapath.

The modifications were made progressively, to fine tune the accuracy while creating the least possible increases in size and power. Since initial spreadsheet estimates predicted the transpose stage was responsible for 38% of the total area, an attempt was made to keep bit width growth minimum before that stage. The size of the transpose stage linearly depends on the bit width of the input to that stage. This is because the stage essentially consists of 128 registers, each the width of the input to the stage. The initial design, with 8 bit inputs to the transpose (figure 6.2), contained 1024 1-bit registers in the transpose stage.

In the initial design (figure 6.2), the least possible resources were used. Symmetric bit width

Figure 6.2: Compliancy Attempt 1



selection allowed identical hardware for both the row and column Butterfly, MSBR, and RAC units. The first bit drop occurred early on in the design at the Butterfly Rows stage. To keep full accuracy, the sum/difference of two eight-bit 2's complement numbers would be 9 bits. However, in this case, the least significant bit is dropped taking care of one of the divisions by two. The other division by 2 occurred in the identical Butterfly Columns unit, which similarly results in an 8 bit number. The 8-bit outputs of the Butterfly stages allow for the row and column RAC structures to be fully serial, since the answer can still converge in 8 clock cycles. For proper pipeline functionality, the RAC structures must finish computation by the end of 8 clock cycles.

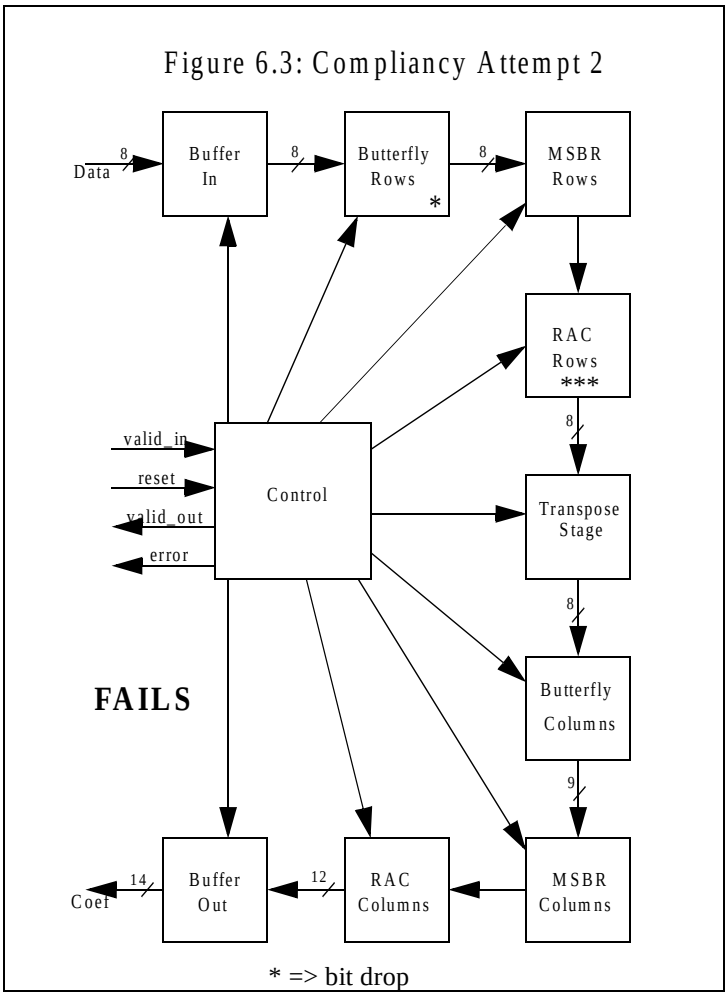
For full accuracy in the RAC, with 8 bit inputs and 11-bit width for the LUT, the output must be 11 bits wide. The intermediates in the RAC are 19 bits, and then a right shift of 8 at the output compensates for the multiplications by 256. The floating point constants in the LUT were multiplied by 256 for 11-bit integer representation. To minimize the size of the transpose stage, and to have symmetric hardware for the row and column calculations, the three least significant bits from the pre-transpose RAC calculations were dropped. This meant that the 11-bit RAC Rows outputs were reduced to 8-bit accuracy. This division by 8 was accounted for at the end of the pipeline by shifting the 11 bit data left 3 positions. Hence, the output coefficients were 14 bits.

This first architectural attempt had large margins of error, and hence, did not pass compliancy testing on even the first block of image A.

The second implementation (figure 6.3) increased the accuracy after the transpose stage and at the very latest stage possible. The calculations on the rows stayed the same as was described in the first attempt. However, in the column calculation, the butterfly stage was increased in accuracy such that a bit was not dropped. 9-bit outputs from the Butterfly Columns resulted and were input to the RAC Columns. Wider inputs to the RAC implied architectural changes would need to be made at the RTL levels that would increase the size of the stage. This is because greater than 8 bit inputs to the RAC mandate levels of parallelization in the structure for the answer to converge in 8 cycles. Outputs from the RAC columns were increased to 12 bits to account for the bit increase at the input. The output coefficients were shifted left by 2 to account for the extra bits dropped before the transpose. Though small improvements were found, compliancy testing still did not

proceed past the first block of image A. As the margins of error slightly decreased with the change, a small reduction in the number of failing coefficients resulted.

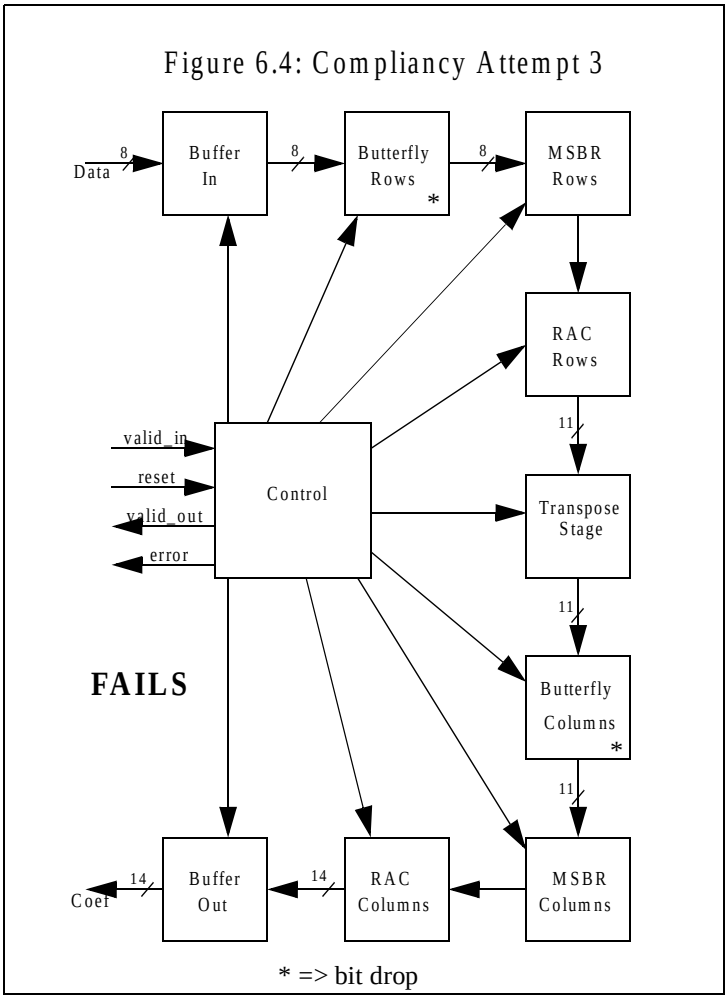
Figure 6.3: Compliancy Attempt 2



Results from the second attempt indicated a much more significant change had to be made. Since the second attempt increased the bit width in the stage directly following the transpose, the third attempt (figure 6.4) increased the bit width of the transpose stage, but decreased the bit width following it. Again, the 1-D calculation on the rows was left the same, where a bit was dropped in the Butterfly Rows stage. Instead of dropping the 11 bit outputs of the RAC Rows down to 8-bits

for the transpose, the size of the transpose stage is increased to allow 11-bit inputs. This signifi-

Figure 6.4: Compliancy Attempt 3



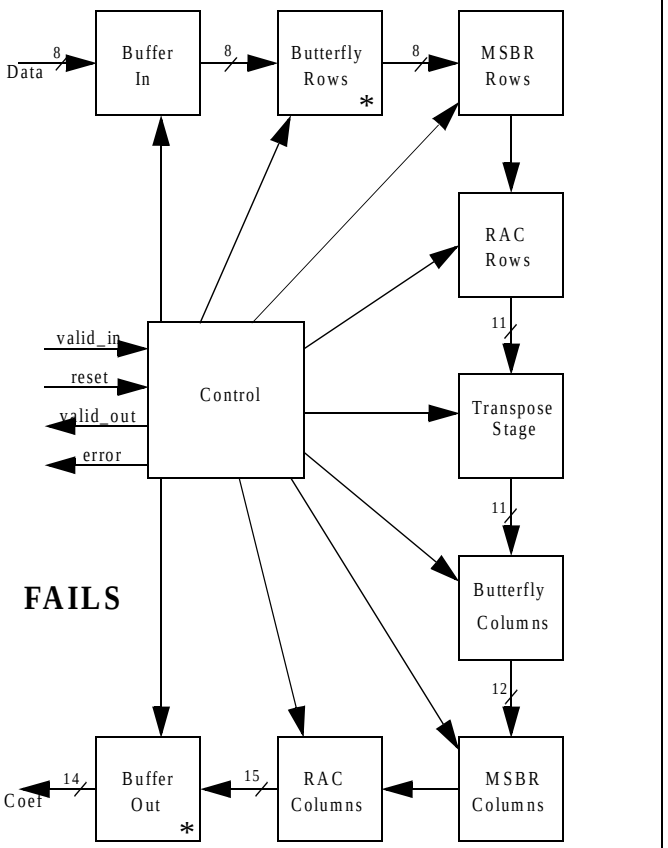
cant change implied an increase in the transpose stage from 1024 1-bit registers to 1408 1-bit registers. As with the first attempt, the second bit was dropped at the Butterfly Columns stage, meaning that the 11-bit inputs to the stage resulted in 11-bit outputs. The 11-bit inputs to the RAC Columns stage would mean further parallelization to that stage and larger bit-widths within it. Since only 2 bit drops occurred in this attempt, no left shifting had to occur at the end. The precision of the outputs was still 14 bits.

The changes implemented in the third attempt had a huge impact on the accuracy of the system. The DUT passed compliancy testing on images A and B. However, it failed on 4 different coeffi-

cients from 4 different blocks in image D. These results implied that the major source of error had been identified and fixed. The following attempts would have to fine tune the accuracy.

In the fourth attempt (figure 6.5), the design stayed very similar to the previous attempt. However,

Figure 6.5: Compliancy Attempt 4

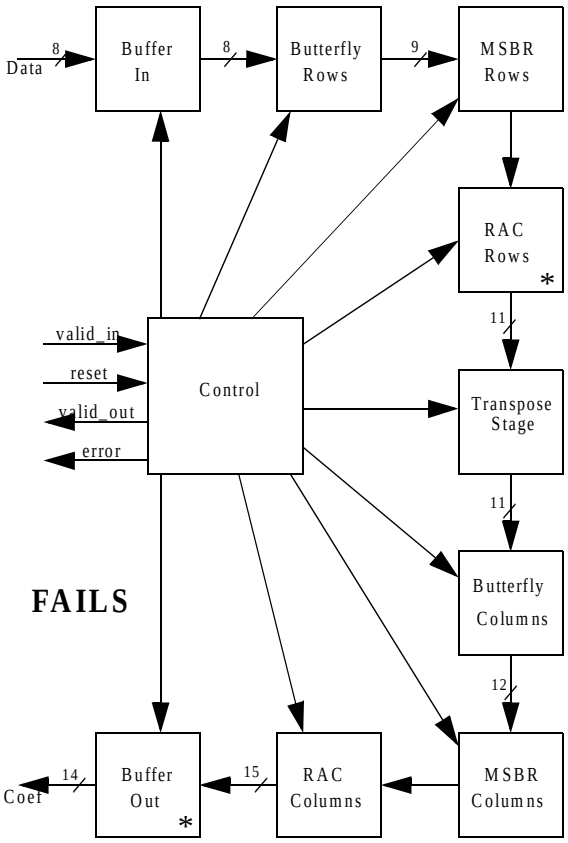


the bit drop in the Butterfly Columns stage was eliminated. This resulted in 12-bit inputs and 15-bit outputs for the RAC Columns. One bit drop remained, as the first drop occurred early on in the Butterfly Rows stage. The final bit drop was implemented at the end of the pipeline in the Buffer Out stage resulting in the 14-bit coefficient accuracy. This implementation fixed the failing coeffi-

cients in image D such that images A, B, and D passed compliancy testing. However, this implementation failed on 2 different coefficients in 2 different blocks in image C.

To attain better accuracy, the fifth attempt (figure 6.6) modified the pre-transpose stages from the fourth implementation. The bit drop in the Butterfly Rows stage was eliminated. 9-bit inputs to

Figure 6.6: Compliancy Attempt 5



the RAC Rows mandated partial parallelization of that stage in the row computations. This is in contrast to prior attempts, which had maintained the RAC Rows stage as fully serial. For full accuracy, 12 bit outputs would have resulted from the RAC Rows stage. However, to try and keep the Transpose stage at minimum size, the first bit drop occurs on the LSB to provide the transpose

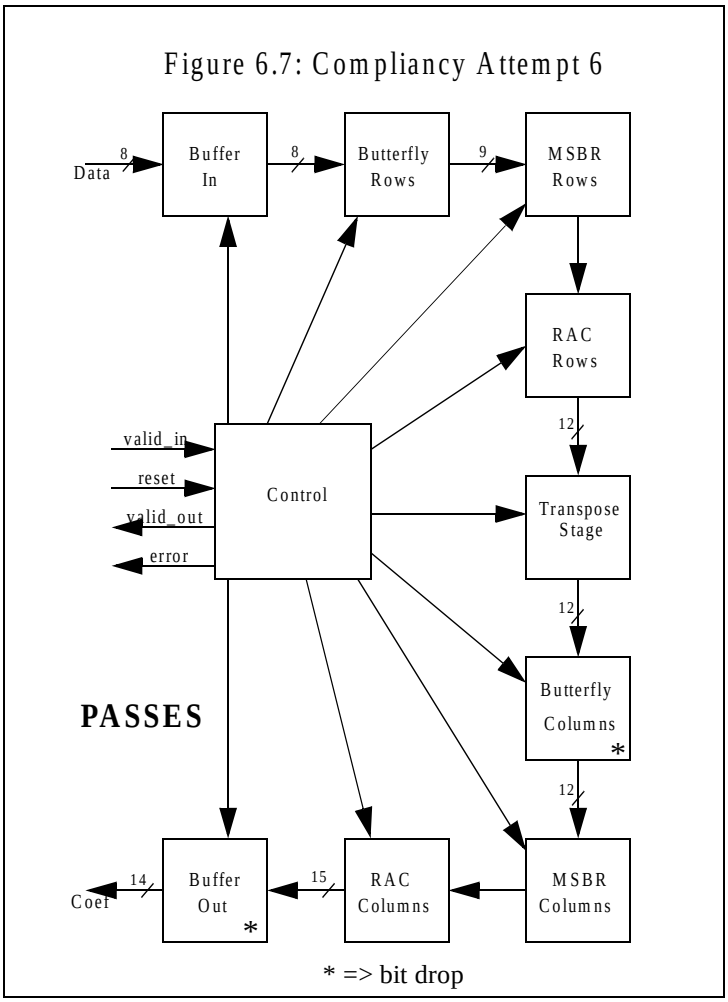
stage with 11-bit inputs. The implementation of the 1-D DCT on the columns stayed the same as the previous attempt.

While results from compliancy testing did not get worse, the fifth implementation still failed on a couple of coefficients in image C. Since only fine tuning to the accuracy was required in attempts 4 and 5, the constant representation in the LUTs underwent experimentation. Multiplication of the constants by anything less than 256 increased the error in the system while multiplying the constants by up to 2^12, or 4096, had no affect.

Hence, experimentation on the placement of the 2 bit drops resumed. A sixth implementation was created by minor changes to the fifth attempt. The sixth attempt (figure 6.7) rearranged the 2 bit drops to be in the post transpose stages. Since no bit drops occurred in the Row calculations, the inputs to the transpose stage were 12 bits. Of the attempts, this led to the maximum size of the transpose stage--1536, 1-bit registers. This is a 50% increase in the number of registers in the transpose stage from the original attempt which used 1024 1-bit registers.

The first bit drop occurred immediately after the transpose stage in the Butterfly columns. The addition/subtraction of 2 12-bit numbers is restricted to 12-bits. The outputs of the RAC Columns is still 15 bits, making the final bit drop occur at the end of the pipeline in the Buffer Out stage.

Figure 6.7: Compliancy Attempt 6



Finally, the sixth attempt passed compliancy testing on all four images. The C implementation of the sixth attempt revealed the architectural changes that would need to be made to the initial RTL design. No longer could the same hardware, instantiated twice, be used to compute the row and column 1-D DCT. The RAC stages for the rows and columns had to be partially parallelized, but to different extents. The adder/subtractor in the Butterfly Rows stage would not divide by two, while the one in the Butterfly Columns stage would. Of course, other bit widths throughout the initial design would be increased as well.

After the correct accuracy was determined for the C model to pass compliancy, it became evident what architectural changes would have to be made in the DUT. Data paths would need widening, the control logic would be modified and extra control logic would be inserted, and the RAC structures would have to be parallelized. The Butterfly and RAC stages underwent the most design changes, as their designs were highly dependant on the bit width of the inputs.

The updated VHDL model was again verified by subunit against corresponding interfaces in the compliant, bit-exact C model. Compliancy testing ensued on the updated, debugged, RTL model. The updated, debugged RTL model passed compliancy testing on all four images.

6.2 Pipeline Functionality Verification

Pipeline functionality of the RTL model was debugged and verified. This included testing reset, stall, flush, and error generation functions. Each of these functions were tested during the three phases of pipeline operation: startup or filling, steady state, and flushing or shutdown. The tests also checked different durations of assertions. For example, the stall tests were verified for the assertion of the stall signal for 1 clock cycle, 2 clock cycles, and a few other randomly chosen values.

Specific hand-coded tests were made by modifying the existing VHDL testbench. The results for the resets and flushes were evaluated via hand analysis. However, for the stall tests, bad data was inserted into compliance image A at the points where stalls were set to occur. Since a stall neglects input data for the clock cycle while it is asserted, the invalid inserted data should not

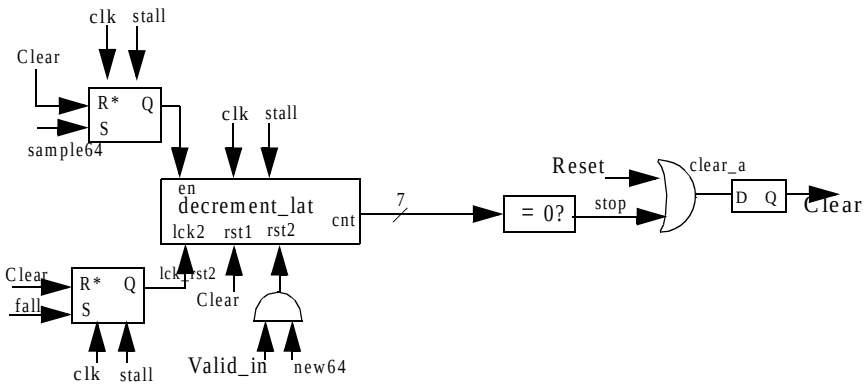
affect any calculations of coefficients. The resulting outputs under the stall should remain the same as under normal operation, except that during the stall, the previous coefficient output is repeated. Hence, the automated compliancy testing suite was used for the stall cases to ensure that the effects of the stall did not affect the other coefficients in the pipeline or future computations.

6.3 Gate-Level Verification

Following the functional and compliancy verification effort on the RTL, the synthesized netlist was verified. Similar to the RTL verification, netlist verification began with a general debug. In this stage, several discrepancies between the RTL and the gate-level netlist were discovered and resolved.

The first discrepancy between the RTL and the gate-level models dealt with reset issues. Correct operation of the RTL model only required the assertion of the Reset signal for one clock cycle. However, it was identified that initialization of the netlist required Reset to be asserted for at least 2 clock cycles. The generation of the clear signal in the control logic is shown again in figure 6.8

Figure 6.8: Generation of the Clear Signal



to clarify this. The clear signal is a combination of a hard reset or a soft reset when the pipeline has completed a flush. The following clock cycle after Reset is asserted, the clear signal becomes asserted. However, not until the next clock cycle (2 cycles after the assertion of Reset) will the clear initialize the decrementer, which will in turn change the stop signal from undefined to 0. If the reset were only asserted for 1 cycle, then the stop signal would still be undefined when the reset became zero again, and the resulting clear signal would be undefined as well. In this case, correct initialization of the core does not occur in hardware. Hence, for correct operation, Reset must be asserted long enough for a valid stop signal to be generated, which for this design is 2 cycles. This observation did not merit changes to the design, but rather to the testbench and the interface protocol.

This requirement on the Reset was not found for the RTL model because the adopted coding style of some elements did not propagate X's. However, when mapped to actual logic in the standard libraries, the X's did flow through the pipeline and highlighted this constraint. This RTL coding style of the multiplexors in the RAC stages also hid another error in the design that appeared only in the netlist.

The RAC units, and in particular, the feedback path input to the adders within the units, were not correctly reset. Thus, undefined signals became trapped in the feedback path to the adder, which kept generating undefined outputs. For most cases, the clear signal generated from the Reset was used to reset control circuitry, which needed to occur prior to computation. However, for the RAC units, the reset was necessary to load zeros into the feedback path to the adders. This was necessary so that new valid data from the LUT would not be added with an undefined value in the feed-

back path, which would result in an undefined value in hardware. Because of the coding style of the RTL model, it was possible to use the clear signal to reset the value held in the RAC. However, for the netlist, previous stages were flushing X's from their system and overwriting them with new valid inputs. In this startup process, inputs to the RAC structures were undefined, resulting in an undefined operand to the adder even though the other operand from the feedback loop had started off reset to 0. The addition with an undefined value resulted in an undefined value which then became trapped in the feedback path.

To eliminate this problem, two choices seemed possible. The first option was to initialize every flip flop in the design to 0 at the reset of the system. However, this option would create a large logical overhead since there are over 3,000 flip flops in the design that would need to be reset. Furthermore, inserting resets on all the flip flops seemed like a waste since, with the exception of the RAC structure, the rest of the units in the datapath would overwrite their stored X values with valid data as it became available.

The second option was to generate signals that would clear the RAC result feedback loops at the exact time that valid data would be entering those stages in the pipeline. This option was selected for implementation as it only created a small change to the control FSM and resulted in very little logical overhead. In fact, only three, 1-bit SR flip-flops were inserted. A clear of the system resets the flip flops to a high value and otherwise they were set to a low value when the FSM flagged that data was at a particular location in the pipeline. Two of the outputs from the SR flip flops corresponded to the reset for the RAC Rows and RAC columns respectively. These flops kept the RAC stages in constant reset mode, continuously selecting zeros in the feedback path to the adder,

until valid data entered the stage. Only when the SR flip flops were set by the FSM, indicating the data had reached the correct position in the pipeline, did they cease to reset the RAC structures.

This design change for appropriately resetting the RAC structures was implemented in the RTL model. Following these changes, the RTL model was verified again and then resynthesized. The resulting netlist could be correctly initialized. However, other errors still remained.

Simulation of the netlist on a hand coded input block revealed timing errors that could be traced back to the FSM. By tracing through instantiated components in the netlist, it was identified that unwanted latches had been instantiated. These latches were inferred by the FSM coding style which created combinational logic within a process statement. Slight stylistic modifications were made to the RTL. This bug was resolved in the sythesized netlist of this modified RTL.

The final discrepancy between the RTL and netlist simulations was not a result of the design, but rather of the simulation tool. The ModelTech simulator, used in zero-delay mode, read delta delays from the instantiated logic elements from the standard library. These delta delays helped the tool determine on which time slice to evaluate the signals. In certain fast paths, the simulator erroneously interpreted the data and the clock to be on the same time slice. This caused a race condition between the clock and the data, where the next cycle's data was sampled instead of the current cycle's data. This error in simulation was observed because the tool ignored the 100 ps delta delays after each library element. The 100 ps delays were neglected since the resolution in the setup file for the tool was set to a ns step time. Modification of the tool step time to ps allowed

the 100 ps delta delays to be read, thereby placing the data and clock on separate time slices and correcting the race condition.

Once the netlist passed the general debug stage, where the RTL and gate-level models did not have any discrepancies between results for hand-coded tests, quantitative compliancy testing ensued on the netlist. Similar to the RTL model, the netlist was tested on the four compliancy images provided from JPEG. With a few modifications, the same compliancy testing suite was used on the netlist. The minor changes to the testbench for the netlist accounted for the glitchy Valid_out signal. On the RTL, since valid_out did not glitch, data was acquired once it was detected to go high. However, on the netlist, prior to valid outputs, the valid_out signal spiked, and it was not desired to begin capturing the data at that point. It is important to note that the observed glitches on the valid_out primary output were on the order of ps, so they did not adversely affect the functionality of the design.

The netlist passed compliancy testing on all four images without failures. Furthermore, the same suite of pipeline functionality tests that were used on the RTL model (described above) were applied to the netlist. The pipeline operated correctly for the netlist. Results from the verification testing suite conclude that the C, RTL, and gate-level models of the DUT operate correctly and with enough precision for JPEG applications and for use in the IBM JPEG core.

7. Synthesis

The Synopsys Design Compiler tool was used in the synthesis step to map the higher-level VHDL RTL code to actual elements from standard cell libraries targeting the IBM SA27E technology. The result of synthesis was a gate-level netlist. Scripts were made and read by the tool to set up the correct operating conditions, limitations on instantiated books, and input/output characteristics to make the netlist compliant with IBM Softcore requirements.

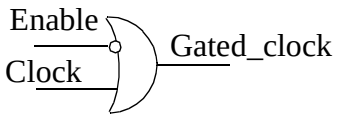
The VHDL subunits were analyzed from the leaves up. The top level design was then flattened and compiled without hierarchy. Subunits, with generic parameters, were elaborated to specify those constant values. Also, those units in which clock gating was intended, were elaborated with the clock gate switch enabled. The replace synthetic switch allowed the correct clock gating logic structures to be inserted into the clock tree.

7.1 Clock Gating

Clock gating is instantiated during synthesis to reduce power consumption. With small logical overhead, clock gating turned off the clock trees to keep them from switching. Two different methods for achieving the clock gating were investigated: Clock-OR (CG-OR) and Clock-AND

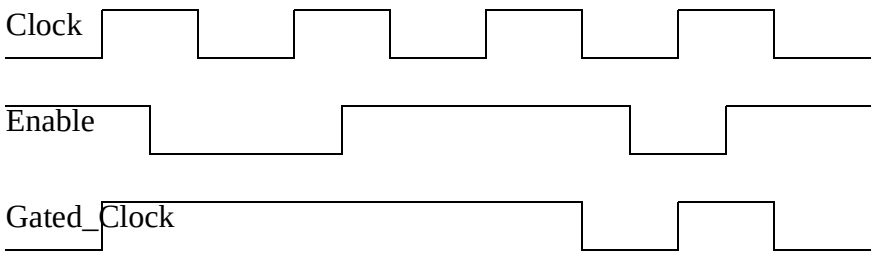
(CG-AND). The implementation of the CG-OR style, as shown in figure 7.1, requires the nega-

Figure 7.1: CG-OR Implementation



tive clock enable signal to arrive before the falling edge of the clock for the valid stall to occur. This means that the valid enable signal must be derived in less than half the clock cycle for correct gating to occur. If the negative enable becomes a valid low during the low part of the clock, then the intended clock gating will not occur. Furthermore, the clock can only be gated to the high state. The timing restrictions for the CG-OR are shown in figure 7.2. These restrictions imply that

Figure 7.2: CG-OR Timing Diagram

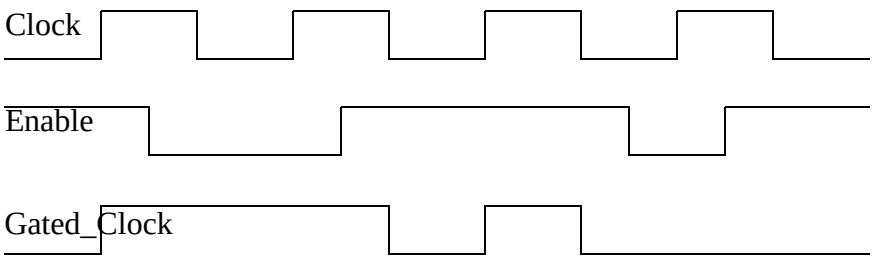


this style is preferable when the enable signal is generated from rising-edge triggered logic and when there is minimal amount of logic in the clock gate enable path. In the DUT, the clock gate enable signal is the primary input (PI) *stall*. This Clock-OR style is not preferred for the DUT because it will restrict the PI stall to be the output of a flip flop. It is desirable to put as few restrictions as possible on the primary inputs to the DUT.

In contrast to the CG-OR style, the CG-AND implementation of clock gating presents fewer restrictions. Therefore, the CG-AND style was chosen for instantiation into the DUT. The nega-

tive enable can gate the clock at either the high or low state. The correct negative enable signal must be available by a small setup time before the rising edge of the clock, giving it approximately a full clock cycle to become valid. These characteristics are pictured in the timing diagram in figure 7.3. This less restrictive timing constraint may lead to reduced power consumption since

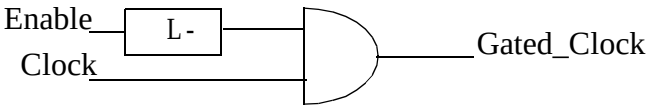
Figure 7.3: CG-AND Timing Diagram



logic cells may not need to be powered up to meet strict timing.

The implementation of the more robust CG-AND style, shown in figure 7.4 is not without its costs. For approximately every 128 gated latches this style requires the insertion of one extra level

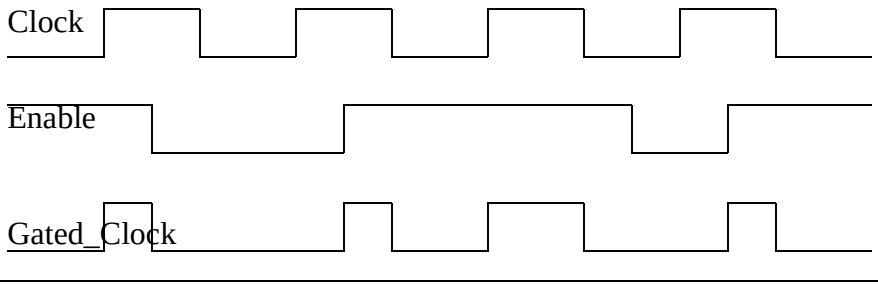
Figure 7.4: CG-AND Implementation



sensitive latch before the AND. The latch is transparent to the enable signal during the low phase of the clock and holds its value during the high phase. The negative latch is necessary to ensure that a transition of the enable signal does not propagate to a unintended transition on the gated clock during the high phase of the clock. Without the presence of the latch, if the enable signal changed during the high phase of the clock, the gated clock would asynchronously change. This

example is illustrated in the timing diagram in below (figure 7.5). The insertion of a latch is not

Figure 7.5: CG-AND Timing Diagram Implemented Without a Latch



necessary in the CG-OR style.

7.2 Setup of Library Specific Operating Conditions

An IBM internal setup script was modified to provide the synthesis tool with information specific to the IBM ASIC library for the SA27E technology. The script set up the worst case operating model of 100 degrees Celsius and 1.65 Volts.These values differ from the worst case operating model for the static timing tool. This discrepancy will be addressed further in the static timing section (8).

The setup script indicated the synthesis tool should perform the most aggressive area optimization. This was achieved by setting the maximum area goal to zero. Initial synthesis results of the DUT yielded a size estimate of 150K cells. This area count was too large for the 90K cell wire-load model, so the next largest size wire-load model of 180K cells would have been optimal. However, since no placement or routing is done for a softcore, it is possible that the customer may spread the logic across the chip. For this reason, synthesis using a 5.3mm chip wire-load model was required. The dramatic effects of this requirement on size, timing, and power will be dis-

cussed in detail later. Along with setting the 5.3mm chip wire-load model, the standard ceramic perimeter (C4P) package with 5 layers of metal (5MZ) was also targeted.

Experimentation was done on the incorporation of elements with varying Block Hardware Codes (BHCs) in the netlist. Increasing BHCs of a book indicate higher power levels, performance, and sizes for that book. The lowest power levels, were not allowed in IBM Softcore methodology for use during the synthesis step as they can potentially create wiring congestion in the physical design. However, after the initial place and route, these cells can be incorporated during the final placement. Therefore, a “don’t touch” was set for certain BHC levels across all elements. Wire-ability and timing performance were further improved by setting the maximum fanout of 10 for all the books in the standard library.

With the lower 180K Cell wire-load model, it was feasible to reduce size and power by setting preferences on the lowest allowable BHC codes. However, with the increase to the 5.3mm chip wire-load model, these lower power cells did not have enough drive strength or performance to meet the capacitive and timing requirements. After experimentation with the static timing of the DUT, the preference statement for the lowest power visible cells was removed in order to reduce the capacitive violations and the setup time violations in the netlist.

7.3 Input and Output Assertions

The netlist generated by the synthesis tool was highly dependant on the input and output assertions of the core formulated in a script file. All the input driving gates, with the exception of the

clock, were modeled with medium strength BHC buffers. This was to ensure that the DUT would function even with weak input drive. A resulting netlist indicated the input ports had fan outs as large as 12. To further ensure any signal could drive the inputs to the DUT, the core was resynthesized, adding a maximum fan out of 1 requirement for the input ports. Of course, the clock port was excluded from this requirement.

The synthesis assertions were extended to include assertions made in static timing analysis. Primary input minimum and maximum slew rates were set with the knowledge of typical customer input driving gates. Similarly, capacitive loads were set on all the primary outputs of the core with values that modelled typical customer gates.

The primary input and output arrival times were defined as well. Primary inputs are given a maximum of 5ns from the rising edge of the clock, or half the clock cycle, to arrive at the core. Feedback from the static timing analysis phase led to an over constraint in synthesis for the output arrival time. The outputs were set to arrive by 1ns after the rising edge of the clock, giving 90% of the clock cycle time to logic external to the core. This is in contrast to the 2ns output arrival time that was allocated when timing the core.

An ideal clock input was selected for the netlist. This was done by setting an ideal clock skew, indicating synchronous clock arrival at all the latches in the design. Second, a “don’t touch” switch was selected on the clock network. This was intended to keep the tool from building and repowering the clock network. By Softcore guidelines, the clock tree is designed once the DUT is incorporated into a customer chip.

The slew rate for the clock primary input underwent slight modifications in the different synthesized versions of the DUT. Initial estimates for the minimum and maximum clock slew rates were chosen from softcore methodology recommendations. Feedback from static timing analysis indicated very small setup time violations in a limited number of long paths.To slightly increase the efforts of synthesis in optimizing these problematic paths, the clock slew rate was adjusted to minimum and maximum values of 100ps and 500ps respectively. These values were chosen with experimentation. With static timing closure, however, all the timing constraints of Softcore guidelines were adhered to.

Effective clock cycle time was one of the major parameters of experimentation for timing driven synthesis optimization. The Synopsys Design Compiler synthesis tool optimized with different algorithms and to different levels of effectiveness depending on how the system was over constrained. Over constraining the synthesis tool was necessary to account for the differences in the synthesis environment and the static timing environment. For instance, Synopsys Design Compiler performed its optimizations based on models analyzed at 100 degrees, while the static timing tool used the same models analyzed at 125 degrees. Furthermore, the synthesis tool optimized until there was zero slack in the timing paths. However, in the static timing tool, delta adjusts were made to make the netlist meet more rigorous, Softcore mandated, timing with a certain margin of positive slack for the setup and hold times.

Initially, a netlist was synthesized to an ideal clock with a 10ns period, and 0ns clock uncertainty. Although reports from the synthesis tool indicated the netlist passed timing, static timing analysis

indicated differently. The netlist failed setup tests on the long paths in the RAC structures by a worst case of -.98ns. To optimize those paths, the effective clock cycle time was incrementally reduced by increasing the clock uncertainty. The uncertainty accounted for clock jitter and made the synthesis tool work harder to optimize timing. As the clock uncertainty was increased from 0ps to 500ps, the worst case setup time violations decreased to -.38ns. The clock uncertainty was increased again to 1ns, the setup time violations became smaller. However, the long paths still needed to be sped up by about 100ps. The clock uncertainty for the synthesis tool was again increased, this time to 1.5ns. Static timing analysis found that the timing actually got worse under these extreme conditions. It is hypothesized that the tool gave up optimization since it could not meet the higher clock uncertainty specifications.

Another approach was taken to increase timing optimization in synthesis. The clock period was reduced to 9ns, a decrease of 10%, and the clock uncertainty was reset to 0ns. These netlist created from these selections of constraints eliminated all setup time violations in static timing. It is hypothesized that the creation of a higher frequency clock triggered different algorithms within Synopsys Design Compiler that made the netlist pass setup time tests.

The selected wire-load model for synthesis also had a large impact on the types of optimizations necessary. For instance, when the core was synthesized for the optimal 180K cell wire-load model, it was not necessary to over constrain the synthesis tool by increasing the clock frequency specification. Instead, the netlist met late mode timing analysis just by increasing the clock uncertainty to 1ns.

Aside from timing optimizations, the wire load model had a large effect on the size of the core. Synthesis of the design on the small, 180K cell wire-load model, with preferences set for lowest visible BHC elements, resulted in a netlist with a cell count of 112,046. Of that, the combinational area accounted for 46,914 cells and the noncombinational area, 65,132 cells. For this netlist, the combinational area was about 42% of the total area.

However, the IBM softcore compliant netlist, synthesized to the 5.3mm chip wire load model, resulted in 120,767 cells. The combinational area, making up 46% of the total area, was 55,300 cells, and the noncombinational area was 65,467 cells. The total area increase from the noncompliant netlist (180K wire-load) to the compliant netlist (5.3mm chip wire-load) was approximately 8%. Combinational logic accounted for 96% of the increase in cell count between the two netlists. This can be attributed to the powering up of combinational cells to drive higher capacitances and resistances associated with logic spread across a chip.

8. Static Timing Analysis

8.1 Background

Static timing analysis, performed by the IBM internal Einstimer tool, ensured that the designed hardware functioned properly with the timing and electrical constraints of the system. Four different path types were analyzed by the timing tool: primary inputs to primary outputs, primary inputs to a register, register to register, and register to primary outputs. For each of these path types, the tool checked that data arrived at its destination in time (setup time) and that it stayed steady for the required time (hold time). This was determined by slack measurements, or relations between the Required Arrival Times (RAT) and the Actual Arrival Time (AT). Both the RAT and AT values differ for early mode and late mode tests. Negative slacks indicated static timing failures, while positive slacks indicated the hardware would function properly for that path. Of course the results of these tests were dependant on the assertions provided.

The setup tests were checked in late mode or long path analysis. The slack, in late mode analysis, is calculated as RAT-AT. In this mode, the latest arrival times are propagated to find the longest path delays. If this slowest path is too long, then theAT will be larger than the required time of arrival. In this case, data may not reach its destination in time, thereby inhibiting the hardware from running at the specified clock frequency.

Early mode, or short, fast path analysis, identified hold time violations. Slack times, in early mode analysis, were equal to AT-RAT. The AT was calculated by propagating the earliest cumulative arrival times for a path. In a fast path, the new signal may arrive too quickly, or before the RAT. The RAT for the early mode case is earliest time that a signal can change after the clock edge. These problematic paths, create negative slack time, and could cause incorrect hardware operation. In these cases, a race condition could occur, where data would be stored from the next clock cycle rather than from the current clock cycle.

The static timing tool also conducted electrical violation tests. For each element instantiated from the standard library, the tool compared its minimum and maximum specified load capacitances with its load capacitance in the design. The tool did the same comparisons for minimum and maximum slew values as well.

8.2 Assertions

This section describes the timing assertions and constraints for the netlist, all of which were compliant with IBM Softcore requirements. These inputs to the tool were provided in the Tool Command Language (TCL) timing assertion files described in detail below. The values were modified from sample files provided by Peter Jenkins for IBM Softcore development.

A phase file was created for the core that defined the clock period at 10ns with a 50% duty cycle. In contrast to the significance of the clock period, the duty cycle of the clock was not a significant factor that affected the performance of the netlist. Both leading and trailing edges of the clock

were set symmetrically with a worst case slew of 500ps and a best case slew of 50ps using clock overrides. User delta adjusts were set to test the netlist under more stringent timing. These delta adjusts required the setup tests to have at least a positive slack of 600ps and the hold tests to have greater than a 100ps positive slack.

A file describing the primary input arrival times was also created. The clock input for Softcores is ideal, so there is no delay in its arrival. The clock slew was defined the same as it was in the phase file. All other inputs signals to the core (stall, reset, valid_in, sample) are given a maximum of half the clock cycle to arrive at the DUT input interface, while their minimum time for arrival was set at 500ps. A fast late slew rate of 700ps was chosen for the PI data. The faster choice was intended to reduce short circuit power. The early slew rate, used for hold time checks, was set to 20ps. These requirements are made to reduce electrical violations after integration of the core into a chip. Furthermore, they attempt to model the bounds of a typical System On Chip (SOC) timing environment, where either low power gates or very high power gates can drive the inputs to the core.

Another file was created for primary input maximum capacitance declarations. Defining resistance and maximum capacitive load values for the driving gates allowed the Einstimer tool to test that typical customer gates could drive the core. During the synthesis phase, these same constraints were used. The clock was idealized, since clock trees are designed and inserted at a later phase when the core is integrated into a chip. The clock idealization was achieved by setting a virtually infinite capacitive limit of 999pF on the clock input port, thereby allowing it to drive a large number of gates without repowering. Repowering of the clock net was avoided so buffers would

not be inserted to create unwanted delays on the clock lines. The resistance was set to 0 KΩ so that the RC time delay on the clock lines would be zero. All other inputs to the core were not idealized. They were set to have maximum capacitances of 0.2pF and resistances of 1KΩ.

A file describing the expected time of arrival for all the primary outputs was created. It provided the late setup and early hold time requirements for the outputs of the core. At the latest, outputs were specified to be valid at 20% of the clock cycle or at 2ns. At the earliest, the outputs could become valid immediately at 0ns. These conditions allow up to 80% of the clock cycle time to be used by a following unit.

Requirements are set for the smallest and largest loads that the output must be able to drive in a primary output loading file. Again, these assertions are made to reduce risks of electrical violations after integration of the core into a chip. The worst case and best case capacitances are set to 0.3pF and 0.01pF respectively. The fanout is set to zero in order to keep the capacitance at a constant value.

Finally, interconnect effects on timing are accounted for by assertions in the wire-load file. Even though the size of the core is about 135K cells, which would optimally utilize a 180K cell wire load model, applications of a softcore require the use of the 5.3mm chip wire-load model. This is because no placement is done for Softcores, so it must be ensured that the core will pass timing even if the logic is spread across the entire chip.

8.3 Timing Driven Modifications

Results from static timing analysis led to modifications at several stages of the project. Changes were made in the design, in the RTL coding style, in the synthesis environment, and at the gate-level netlist.

Static timing setup time violations, found in late mode analysis, made it apparent that there was a considerable design flaw in the control logic. The PO *Alert* was not latch bounded and was logically dependant on the PI *valid_in*. An error was created because the PI arrived at 5ns after the rising edge of the clock, but outputs were expected to be valid no later than 2ns after the clock. The setup time violation of greater than 3ns obviously flagged this design error. It was corrected by inserting a flip flop to make *Alert* latch bounded.

The RTL was further modified to hand optimize some of the elements in the longest delay paths that created smaller setup violations. For example, the two, two-input multiplexors in the RAC units were recoded as one, three-input multiplexor. Unfortunately, these changes yielded very little to no gain in timing.

To fix the setup violations, the long delay paths essentially had to be sped up. This was done by over constraining the Synopsys Design Compiler synthesis tool. By over constraining the synthesis tool, the discrepancy in operating conditions between the synthesis and timing tools was accounted for. Specifically, Synopsys Design Compiler performed its optimizations using models at 100 degrees Celsius while the Einstimer static timing tool used models analyzed at 125 degrees Celsius. The synthesis tool was over constrained by effectively reducing the clock period (see

synthesis section for more details). The effect was that gates with large delays or large slews were powered up to run faster.

Once the proper constraints for synthesis were determined, and the new netlist passed all the setup tests, attention was focused on passing the 1,619 hold time violations found in the early mode analysis. These violations meant that there were fast paths in the design, where data needed to be slowed down. The Booledozer tool was used to insert delay books from the standard library in all the fast paths to eliminate the hold time violations.

In the final electrical tests, the Booledozer tool found no violations. However, if books were found to drive too large a capacitance, their power level could have been incrementally increased by hand until the violation was resolved.

The new netlist, altered to meet all the timing requirements, increased in size to 135,407 cells. The addition of the delay elements in the fast paths accounted for all 14,640 cells of increase. An approximate transistor count for the core can be obtained using the provided technology specific IBM SA27E ratio of 2.96 cells per gate. Referencing a two-way NAND gate, an estimation of approximately 4 transistors per gate can be used. This leads to a conversion factor of 1.35 transistors per cell. Hence, the total number of transistors in the design is estimated at 182,800.

Other features of the netlist include 12 bits of inputs, 16 bits of outputs, 12,956 instantiated gates, and 3,169 register bits. The average fanout for each book is approximately 2.025, which is well below the 2.7 value recommended to avoid wiring congestion in the physical design.

9. Design For Test Compliance (DFTC)

The final front-end stage required by IBM Softcore development methodology is the design for test compliance (DFTC). This step is driven for manufacturing testability purposes.

The process is mostly automated with the use of the DFTC Cortex tool. Three pre-physical design checks are made: Electrical Rule Checks (ERC), Pin Usage Checks (PUC), and Name Checks (NAC). The ERC collects all the pin type information for all the books being instantiated in the netlist of the core. It makes sure that all the correct pins types are being driven or are driving other structures. The PUC makes sure all the synthesis rules (SRULES) are being abided. For instance, it makes sure that pins mandating a connection have one. For example, all latch outputs must be connected. The NAC checks the net names and ensures that there are no illegal characters or formats.

The Cortex tool invoked the IBM internal TestBench tool to perform testbench checking. It makes sure all instantiated books can be located and compiles all their associated faults. Although the DUT contained 187,900 faults, they were collapsed into 139,890 faults once the books were connected. Test patterns for the transition and stuck faults are automatically generated.

Next, the Cortex tool launched the IBM internal Booledozer tool to do the design for test synthesis (DFTS) process. This process replaced all pseudo flip-flops from the standard cell library with real master-slave pairs of level sensitive scan design (LSSD) latches. The LSSD latches have

extra clock and control pins for testability purposes. A, B, and C clock pins are inserted as well as scan in and scan out pins [14].

Test structure verification (TSV) ensues to make sure all the latches or flip flops are accessible through a scan chain. This is necessary for observability and testability. Furthermore, it eliminates race conditions in the test patterns. For a softcore, only one scan chain is created that includes all 1-bit registers in the design. The scan chain length for the DUT was 3169 flip-flops, indicating all the registers in the design where included. This scan chain was used in the final steps to determine the test coverage of the system.

The TestBench tool first flushed all the latches in the scan chain, successfully testing 75,437 or 53.93% of all the collapsed faults. Cortex tested the remaining faults by setting up the latches with specific patterns, launching the scan out, and analyzing the results. Fourteen redundant logic paths were found, and were therefore rendered untestable. However, the DUT still met IBM Softcore requirements and passed DFTC since the final test coverage was 99.99%.

10. Power Analysis

The major component of power dissipation in the design is attributed to dynamic CMOS switching, as calculated by the formula in equation 10.1. The targeted IBM SA27E technology determined the power supply voltage (V_{dd}). Capacitive loads (C_L) scaled proportionately with the technology constants as well. The JPEG core specifications set the clock frequency (f_{CLK}) at 100 MHz. Few parameters in the power calculation were left for optimization by the designer. Attaining a design with reduced cell count optimized the capacitive load factor in the power calculation. The primary objective of the "power aware" architectural design of the DUT was to reduce the probability of switching (p_s) factor within the core to attain low power.

$$P = \frac{1}{2} \bullet p_s \bullet f_{CLK} \bullet C_L \bullet V_{DD}^2 \qquad \text{(Equation 10.1)}$$

Power analysis of the DUT was conducted during the beginning and ending phases of the project development. An early power calculation of the architectural design, using spreadsheet analysis, provided a rough estimate of power consumption. An accurate measurement of power consumption was obtained during the final phase of the project by using the Sente WattWatcher tool on the synthesized netlist that had passed static timing.

10.1 Spreadsheet Analysis

The spreadsheet analysis technique of estimating power consumption was done in the initial phase of the project to determine the best of two different algorithms, both of which resulted in a different architectural implementation. Prior to results from spreadsheet analysis, it was unclear whether the Chen and Fralick 1-D DCT algorithm was best extended to two-dimensions via a direct approach[6] or through conventional row-column decomposition methods.

Initial architectural designs were formulated for both 2-D extensions of the Chen and Fralick 1-D DCT algorithm. The architectures provided estimates of the combinational logic area, number of flip flops, and switching factors for each implementation. Estimated cell counts of combinational logic in the architectures were found by determining which logic elements would be needed from the library. Then, the IBM ASIC SA27E Databook was used to find the cell count of each anticipated logic element in the design. The initial designs assumed minimum computational accuracy, where all bus widths were 8 bits. The reduced accuracy limited the number of flip flops in the design that stored intermediate calculations between pipeline stages. The number of flip flops in the designs were manually counted and included in the respective spreadsheets. From the architecture, it was clear that flip flops dividing pipeline stages were only being enabled only once every 8 clock cycles. This knowledge helped set the register switching factor estimate in the spreadsheet.

The initial design of the row-column approach contained 1,777 flip flops and had a total cell count of 79,686 cells. A 0.1 activity factor was estimated for the combinational and flip flop switch factors. The design for the direct approach used the same switching factor estimates. However, the

total number of flip flops had increased to 7,967 and the total number of estimated cells rose to 157,975.

The IBM Methodology team provided an SA27E technology specific spreadsheet template to estimate power consumption. Design parameters including operating voltage, clock frequency, combinational logic area, flip flop count, and switching factors were input to the spreadsheet. For the direct approach design, the anticipated register arrays were also incorporated into the spreadsheet analysis. The spreadsheet accounted for technology constants such as gate length and data and clock capacitances. It created estimates of the clock tree and included it in the total power consumption. Initial results of spreadsheet analysis produced a 41mW power estimate for the row column design, while the direct approach design was estimated to consume 127mW of power.

Initial spreadsheet analysis indicated the direct approach required approximately twice as much silicon area as the row column method and consumed about three times as much power. This dramatic difference can be attributed to the irregularity of the direct approach algorithm that created significant control logic overhead in the design. These results confirmed that the best algorithm for hardware implementation was based on row column decomposition. Therefore all the following development stages of the project were done only on the optimal, fast, Chen and Fralick 1-D DCT algorithm extended to two dimensions via row-column decomposition.

10.2 Sente WattWatcher Analysis of the Netlist

The final product, which was a verified, synthesized, and timed netlist, underwent refined power analysis using the Sente WattWatcher tool. The tool was set to run in full simulation mode, in which switching factors were utilized for all nets in the gate-level design. These switching factors are obtained by writing out activities files from the MTI simulation environment of the netlist. The WattWatcher power calculations incorporated the SA27E technology and library parameters. Furthermore, it estimated a clock tree network for the design and included that power dissipation in the total.

Two types of image data were tested on this design: unnatural image data and natural image data. The unnatural image set consisted of JPEG compliancy images A and B (refer to section 6.1). The natural image data set was obtained from the Barbara image, shown below in figure 10.1, which

Figure 10.1: Barbara Image

was represented in the luminance-chrominance (YUV) coordinate system[2]. The luminance component (Y), provided the information of the gray scale version of the image while the U and V chrominance components provided the extra information to convert the gray scale image to a color image. Each of the luminance-chrominance components were processed separately by the

DCT. The resulting coefficients corresponding to each component are normally interleaved in post DCT processing.

Table 10.1 provides the results of the power estimation for the different input sets. Compliance images A and B, were fully characterized to derive the unnatural power estimate. The total number of unnatural blocks tested was 286, which corresponded to the computation of 18,304 coefficients. The natural Barbara image (figure 10.1), was tested over the computation of 69,120 coefficients, corresponding to the first 1080 blocks, or first 24 rows of pixels.

Table 10.1: Power Consumption

Image	Wireload Model	Power (mW)
Compliance A	5.3mm Chip	79.7
Compliance B	5.3mm Chip	87.1
IBarb, U component	5.3mm Chip	65.0
IBarb, V component	5.3mm Chip	63.9
IBarb, Y component	5.3mm Chip	73.2

As table 10.1 highlights, the power consumption for the calculation of the luminance component is approximately 14% greater than the average power consumed from calculation on the chrominance components. This significant increase is not surprising since the luminance component contains the majority of the edge information in the image.

Across all three natural image components, an average power of 67.4mW is estimated. Average power is also computed across the unnatural compliancy images A and B. The results of average power calculations across the natural and unnatural data sets are given in table 10.2 below. On

average, processing of the natural data consumed 23.7% less power than the processing of unnatural data.

Table 10.2: Average Power

Image Type	Avg Power (mW)
Unnatural	83.4
Natural	67.4

Results show the design did dynamically scale down power consumption based on increasing input correlation. From the unnatural image data set, the worst case power is estimated at 87.1mW. The lowest power, estimated from the natural image sets, was 63.9mW. The data dependant values for best and worst case power consumption indicate a maximum savings of 36.3% in power consumption can be achieved with correlated inputs.

Aside from data dependant power reduction, a clock gating scheme (detailed in section 7.1) was implemented in the DUT to reduce power in the clock net. The power savings achieved from clock gating is yet to be measured.

All power analysis was done on the DUT that was synthesized to the 5.3mm chip wire-load model. To study the first order affects that the required larger wire-load model had on power, a comparison was made to when the more optimal, 180K cell wire-load model was selected in the tool setup. Although the netlist of the DUT did not change under the two comparisons, WattWatcher used different resistance and capacitive values which corresponded to each wire-load model.

Power comparisons between the 5.3mm chip and 180K cell wire load models were performed over 1366 blocks. The same data, consisting of both unnatural images and the YUV components of a natural image, were tested on both models. The results are shown in table 10.3. A 29.6% increase in power results from the use of the 5.3mm chip wire load model as opposed to the 180K cell model. Of course, power comparisons would have been even more dramatic if the DUT were synthesized to the 180K cell wire-load model instead of the 5.3mm chip wire load model. If the core were synthesized with the lower wire-load model, lower power elements could have been selected by the tool to further reduce overall power consumption.

Table 10.3: Affects of Wire-load on Power

Wire-load Model	power (mW)
180 K Cells	57.0
5.3mm Chip	73.86

The initial power estimates obtained from spreadsheet analysis were along the same order of magnitude to those obtained from the Sente Wattwatcher tool. The lower estimate from spreadsheet analysis can be attributed to the minimum accuracy architecture which had to be significantly revamped following compliancy testing. Furthermore, the initial hand estimates for combinational cell count did not account for the large, 5.3mm chip wire-load model. Following synthesis and timing, exact combinational cell counts and flip flop counts of the DUT were acquired; the combinational area of the DUT was 69,940 cells and it contained 3,169 flip flops. With all other parameters set the same as described in section 10.1, spreadsheet analysis of the synthesized, timed DUT resulted in a power estimate of 76mW. Since the spreadsheet did not take into account

data dependant power scaling, it is not surprising that the estimate is approximately the same as the average between the Sente Wattwatcher power estimates for natural and unnatural images -- 75.4mW. Table 10.4 provides the breakdown for power dissipation within the DUT as calculated through final spreadsheet analysis.

Table 10.4: Breakdown of Power Dissipation within the DUT

Component	Percent of Power Consumption
Clock Tree	45%
Flip Flops	22%
Combinational Logic	33%

11. Performance Comparisons

The 2-D DCT DUT was dedicated to low-power on the algorithmic and architectural levels. The core was optimized for power through reduced capacitive loads and switching factors. More details on adopted power reduction methods are outlined in section 4. On natural images, the DUT consumed an average of 67.4 mW across all components.

The features of the DUT are compared to three other similar 2-D DCT products on the market in the table 11.1 below. The DUT is listed as chip D. While all four designs are dedicated to low power, they achieve this goal by focusing on different design parameters. There are similarities between the implementations though. For instance, all the implementations use a fast 1-D DCT algorithm as a basis for extension to 2-D. Furthermore, all incorporate DA methods for calculating products. The comparison across the four chips will help determine the limitations of the design and provide insight to future improvements that can be made. Furthermore, it confirms the competitive nature of the developed product.

Table 11.1: Feature Comparison

	Chip A	Chip B	Chip C	Chip D
Transistor Count	152,017	120,000	120,000	180,000
Clock Rate	100 MHz	150 MHz	14 MHz	100 MHz
Latency	198 Cycles	112 Cycles		108 Cycles
Block Size	8x8	8x8	8x8	8x8
Supply Voltage	2.0 V	0.9 V	1.56 V	1.8 V

	Chip A	Chip B	Chip C	Chip D
Throughput		64 clk/block		64 clk/block
Power	138 mV	10 mW	5 mW	67.4 mW
Accuracy	HDTV	H.261		JPEG
Technology	0.6um CMOS, SPDM	0.3um CMOS, triple well, DM	0.6um CMOS TM	0.18um CMOS, dual well

In order to effectively compare the implementations presented in table 11.1, one must reflect on how the different design metrics affect dynamic power consumption. For this purpose, equation 10.1 is replicated below. Power increases proportionately with the clock frequency as well as with

$$P = \frac{1}{2} \bullet p_s \bullet f_{CLK} \bullet C_L \bullet V_{DD}^2 \qquad \text{(Equation 10.1)}$$

the square of the supply voltage.

Chip A [9], designed at the DSP/IC Design Lab of National Taiwan University, optimized for power at the algorithmic, architectural, and circuit levels. Special adder, memory, and register power saving circuits were designed and used extensively in this chip. The adders had a hybrid-architecture that combined the carry select structure for high speed and the Manchester structure for low power. A power saving ROM was designed using precharged logic. The dual ported SRAM used reduced voltages to achieve low power. A TSPC D-FF register design was employed, which enabled combinational logic to be embedded within the flip flops in a pipelined datapath. Compared to the static CMOS D-FFs, the TSPC D-FFs were sited to be better in terms of power, speed, and transistor count.

At the algorithmic level, chip A adopted the direct approach [9] to the 2-D DCT to attempt to reduce computational complexity. The design needed several 64-word SRAMs with varying word sizes. Aside from the ALUs used in the 1-D DCT to calculate the DA, 64 other 13-bit adders were necessary. Other substantial chunks of logic were designed as routing modules to handle the irregularity of the algorithm. This same algorithm was investigated for use in the DUT, but after initial architectural design and spreadsheet power analysis, it was not selected due to very high cell counts and power consumption.

Chip B [11], designed for the Toshiba Corporation by members of the ULSI Device and System Engineering Laboratories, used the conventional row-column approach for the 2-D DCT. The low-power design relied heavily on device level optimizations. Multi threshold voltage CMOS (MT-CMOS) and self-adjusting threshold voltage schemes were developed and applied to logic gates and memory elements throughout the design. This enabled a low supply voltage of 0.9V without negatively impacting performance. As can be seen from equation 10.1, the reduced operating voltage had dramatic effects on reducing power. Furthermore, the targeted technology had a triple well process that was useful for isolating the sensitive circuits from noise sources.

Custom circuits, utilizing the MT-CMOS, were also an integral part of reducing power in chip B. Low voltage SRAMs were designed with six transistors per cell, and a latched sense-amplifier. A carry skip structure was selected for the adder and was optimized using small-swing differential pass-transistor logic (SAPL) and sense amplifying pipeline flip flops (SA-F/F).

The third design, chip C, was developed at the Massachusetts Institute of Technology [10] to reduce power consumption at the algorithmic, architectural, and circuit levels. Similar to the second design, chip C used row-column decomposition to extend the 1-D DCT to 2-D. The DA version of the Chen Fast 1-D DCT [7] was chosen in chip C. This is the same algorithmic combination used in the DUT. Similar to chips A and B, chip C [10] also used custom circuitry to achieve low power. Adders were all designed with a 4-stage carry-bypass structure. Bit rejection units were also designed and optimized at a circuit level.

Chip C adopted a unique approach to low power at the architectural and algorithmic interface. It utilized data-dependencies to scale down power consumption by introducing MSBR. Power could also be scaled down via dynamic selection of computational accuracy. Note that the very low power estimate was achieved by testing the chip at a very slow clock frequency 14 MHz.

Chip D in table 11.1 provides the characteristics of the DUT. It was developed at the IBM Microelectronics Division in conjunction with the Massachusetts Institute of Technology. Design D used the same algorithmic selections as chip C. It also implemented MSBR to dynamically scale down power consumption by reducing the switching of nets.

Design D was unique in that it was a softcore while the other chips had gone through physical design. This constrained Chip D from using circuit or device level optimizations since all logical elements were selected from a standard library. Physical design would allow chip D to optimize size and power to greater degrees. For instance, components with reduced transistor count could be chosen. Furthermore, the extension of design D into a hardcore would allow for the optimiza-

tion of clock nets as well. This could potentially be a significant factor as chip A and C estimated 26% and 40% of total power dissipation was in the clock buffers respectively.

In essence, when the differing design metrics are accounted for, each of the dedicated low power designs presented in table 11.1 consume around the same amount of power. The circuit optimizations presented in designs A, B, and C can always be incorporated into the DUT if a hardcore were desired. Given approximately the same amount of power dissipation, the DUT may be optimal given the faster time to market and reusability associated with a softcore.

12. Future Work

Enhancements to the DUT or the design's environment can be made at all stages in the development cycle. General areas of improvement include power optimization, additional core functionality, and development of more extensive automated verification schemes.

12.1 Architecture

At the architectural level, investigation should be made into redesigning the Transpose Stage. This stage contained 48% of the total number of flip flops in the design and accounted for approximately half the total size of the core. Since power increases proportionately with area (flip flop count and combinational cells), the transpose stage is a critical unit to optimize in the future. One should investigate changing the design to allow the insertion of register arrays in the transpose stage. These arrays could optimize the size and power of this stage.

Power consumption is proportional to switching factors. To reduce power consumption in the transpose stage, one could ping-pong the sampling and computation register stages to reduce the the switching factors of the flip flops in the stage by 1/2. Instead of writing to the sampling registers, and then copying the data to a set of computation registers, the sets of registers could just be swapped once the sampling stage is full. With the design change in the transpose stage, each flip flop would at most switch every 128 cycles instead of every 64 cycles as currently designed. This

would significantly reduce power consumption of the stage to make it worth the logical overhead that would control the ping-ponging of the registers.

Other optimizations at the RTL or architectural level could be to instantiate components from the IBM Bitstack library. While the IBM Bitstack Library calls the SA27E standard library for basic building blocks, it contains higher-level logic elements with balanced or repeated structures and wide data paths. The Bitstack elements have been optimized with respect to performance, density, power, and size. These elements have been designed with 90-95% density, short interconnects, and small wire capacitance and RC effects. The bits of the datapath are kept physically aligned and logic is placed in an orderly fashion to avoid the random structuring that can occur from synthesis tools. Physical design tools have the option of identifying and taking advantage of these elements which have a specific, optimized layout. Examples of elements in the Bitstack library that may improve the design include adders, multiplexors, and demultiplexers.

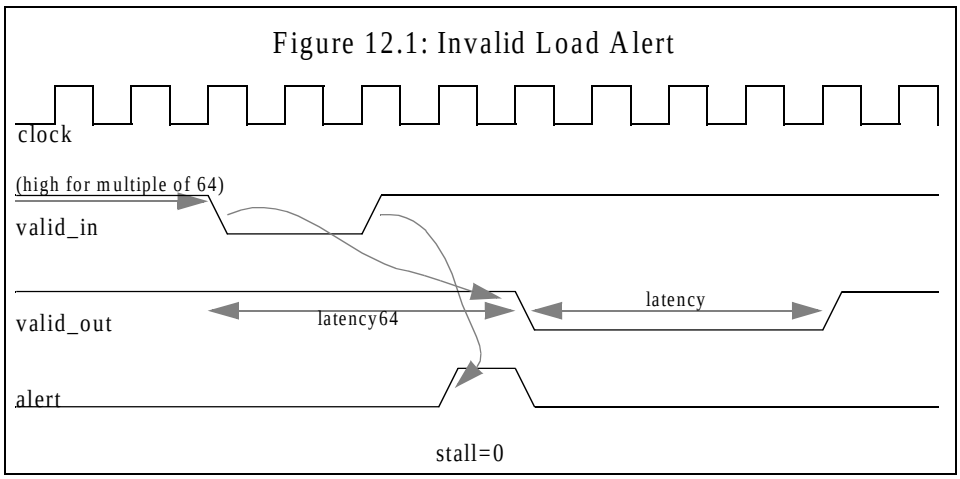
Aside from inclusion of Bitstack elements, further analysis and evaluation of the design could identify other means of optimizing logic. RTL coding styles could be modified to more accurately reflect the hardware operation. For example, alteration of the coding style could reflect the movement of X's within the design. The ability to track X's in the design would enable more effective verification in the RTL design rather than finding these problems after the netlist is generated.

Another simple logic optimization would be to eliminate unnecessary registers. For example, within the RAC units, the results from the MSB partial products were shifted left appropriately before being summed with the results from the LSB partial product (figures 5.24 and 5.38). The

design could be changed to do the left shifting after the D-ff. This simple change would save 32 1-bit registers in the RAC Columns stage and 8 1-bit registers in the RAC Rows stage.

The architectural design could be modified to provide more functionality for the Alert signal. In particular, the Alert signal should indicate all types of loading errors. As currently designed, an invalid load is signaled by the Alert if the pipeline is in the process of flushing when valid inputs begin (valid out = 1 and valid_in transitions from 0 to 1). The timing diagram in figure 12.1 illustrates this scenario. Correct loading of the pipeline, to enable correct tracking of valid outputs,

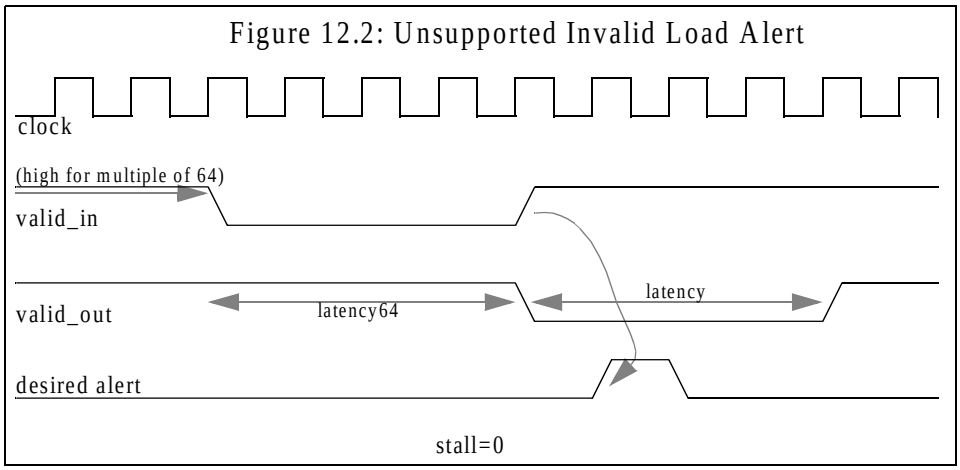
Figure 12.1: Invalid Load Alert



requires the valid_in signal transition from 0 to 1 no earlier than 1 clock cycle after valid_out becomes 0. For completeness, logic generating the Alert signal should be modified to also indicate the boundary case when the valid_in signal transitions high on the same clock edge as the

valid_out signal falls. The timing diagram in figure 10.2 highlights this case, in which the current design does not indicate an invalid loading alert, but for which a future design should.

Figure 12.2: Unsupported Invalid Load Alert



12.2 Verification

While it would be worth implementing some design enhancements in the future, it is critical to invest in a more thorough verification effort of the design. In particular, independent verification of the design would be most valuable. This additional verification effort should focus on pipeline functionality since the current verification suite comprehensively tested for accuracy errors via automated compliancy testing. The resetting, stalling, and flushing were only tested from four or five randomly selected states from the control FSM. The testing of these selected states did occur during the three different phases of the pipeline: filling, steady state, and emptying. Also, specific sequences of these control signals were tested to confirm the core specifications.

The verification environment should be extended to reset, stall, and flush the system from each of the 65 states in the FSM during all three phases of the pipeline. In addition, more thorough inves-

tigation should be made into the affects on the system from interactions between these control signals. With the expansion of the verification effort, greater automation will become necessary. Extensive testing would create too much data to analyze manually, and could therefore result in erroneous conclusions. Furthermore, it is not feasible to maintain a design with manual verification steps. For the current pipeline verification environment, it was more time-efficient to manually analyze results instead of developing complicated testbenches that themselves would require debugging.

12.3 Synthesis

The suggested efforts in verification and architectural design are predicted to have the most significant impact on the DUT. Second order modifications to fine tune the design can be made in the later stages of the design flow. In the synthesis stage, most improvements rely on optimally controlling the synthesis tool.

Dramatic changes to the netlist, created in the synthesis stage, are difficult to make. There is little freedom for experimentation since most of the crucial input parameters to the synthesis tool are dictated by either the IBM Softcore Methodology, the SA27E technology, or the architectural design. One area of experimentation that significantly impacts the performance of the core, is the means and extent to which the synthesis tool is overconstrained to meet all timing requirements while keeping power consumption and area at a minimum. Some of the values that can be varied to include slew rates, capacitive loads, clock uncertainty, and clock period.

Another aspect that remains to be analyzed is the power, size, and timing trade-offs between incorporation of the CG-OR and CG-AND clock gating styles (7.1). Currently the DUT invokes the CG-AND style of clock gating for conservative timing. However, the CG-OR style could be incorporated into the netlist instead. The CG-OR style would be optimal if it created less logic than the CG-AND style, and passed static timing analysis.

12.4 Static Timing

There are very few changes that can be made late in the design flow. However, one could use the static timing information to determine the critical paths that are limiting the clock rate for the design. For this design, the slow paths are isolated to the RAC stages. To run the core at a faster frequency than 100MHz, one could investigate deeper pipelining of this stage.

To fix hold time violations, specific delay elements from the standard library were inserted. There were several choices of delay elements, but a conservative element was chosen for instantiation into all the fast paths to maintain a moderate positive slack of a few picoseconds. Perhaps, insertion of a smaller delay element would have been sufficient to pass static timing. Investigating this suggestion is straight forward since inserting delay elements to the netlist is automated and repeatable with the Booledozer tool. Therefore, it should not pose as a barrier to synthesizing the RTL again in the future.

12.5 Power Analysis

Work can be done in the future to develop more accurate power specifications of the core as well as to ease the power calculation effort. The power specification for natural images was generated from testing portions of one natural image. To get a more characteristic result, it is recommended to run power estimates on a variety of natural images. In order to do this, in a limited amount of time, better methods of acquiring switching factors from simulation is necessary. Currently, VCD files are written out from the simulator, but they double the simulation time, which for a netlist is already lengthy. Even worse, the power analysis tool would take days to generate power estimates across just one natural image. In addition, the generated VCD files, which list every change in every net at every picosecond, would be too large for storage.

To cope with issues related to VCD files, the Sente Wattwatcher tool supports IAF format activity files. IAF files are supposed to be much faster for the power tool to analyze, and are supposed to be magnitudes smaller in size than VCD files. Sente provided code for insertion into the design netlist to enable the MTI simulator to generate the IAF files. The IAF capabilities were not used because the provided code created segmentation faults in the simulator at runtime. In the future, it may be worth the time to debug this issue so the benefits of IAF files can be exploited.

12.6 Core Utilization or Expansion

The 2-D DCT softcore design could be extended to offer a hardcore product through an additional physical design effort. A hardcore version would result in significant benefits with regards to area, power, and clock cycle time of the design. Furthermore, a hardcore version of the low-power 2-D

DCT would provide a competitive edge in the market since most the other low-power options have completed physical design (11).

At a much broader scope, efforts in the future should be directed to utilize the designed softcore implementation of the 2-D DCT. For instance, a JPEG core could be developed. It is estimated that the availability of this softcore would reduce the JPEG core development effort by 1/3.

Furthermore, development of this core has provided insight into algorithms and design techniques to reduce power for DCT-based transforms. One could use this expertise to develop a low-power inverse DCT core.

13. Conclusions

A low-power 2-D DCT core was designed according to IBM ASIC Softcore Methodology. Algorithmic selection, architectural design, verification, synthesis, static timing, design for test compliance, and power analysis development stages were outlined in this thesis. The core was targeted to the IBM SA27E or 7SF technology which featured at 0.18um CMOS technology and 1.8V supply voltage. The core operated at 100 MHz, at which clock rate it consumed approximately 63mW of power on natural images. The resulting area of the core was 135,407 cells or approximately 182,800 transistors.

While the DUT was implemented with enough accuracy to be JPEG compliant, it was designed with minimal resources to reduce cell count and power consumption. The 2-D DCT was dedicated to low-power on the algorithmic and architectural levels. The design utilized the matrix factorization version of the Chen and Fralick fast 1-D DCT algorithm which could be implemented with distributed arithmetic to remap the high-power, high-cost multiplications to simple additions and shifts. Results from spreadsheet power analysis identified row-column decomposition as the optimal 2-D extension of the 1-D algorithm. This 2-D approach kept the control logic overhead at a minimum and increased the regularity of the design.

At the architectural level, most significant bit rejection units were designed to produce a temporally “power-aware” system. Power consumption dynamically scaled down with increasing input correlation. This resulted in a power savings of 23.7% for natural images compared to unnatural

images. The 2-D DCT core also made use of clock gating methods to reduce power consumption in the clock tree.

The 2-D DCT project was completed at the IBM Microelectronics Division under the ASIC Digital Cores I department in Burlington,Vermont as part of the MIT VI-A program.

14. References

[1]MIT Information and Entropy Spring 2000 website.
http://www-mtl.mit.edu/Courses/6.095/spring-00/unit3/dct.html

[2]Mitchell, Joan and Pennebaker, William. JPEG: Still Image Data Compression Standard. Van Nostrand Reinhold, New York. 1993. pp. 1-96, 335-626.

[3]Y.P. Lee, T.H. Chen, L.G. Chen, M.J. Chen, C.W. Ku, "A Cost-Effective architecture for 8x8 2-D DCT/IDCT Using Direct Method", *IEEE Trans. Circuits Syst. Video Technol.*, vol. 7, no. 3, pp. 459-467, June 1997.

[4]J.H. Hsiao, L.G. Chen, T.D. Chiueh, C.T. Chen, "High Throughput CORDIC-Based Systolic Array Design for the Discrete Cosine Transform", *IEEE Trans. Circuits Syst. Video Technol.*, vol. 5, no. 3, pp. 218-225, June 1995.

[5]M.T. Heideman, "Computation of an odd-length DCT from a real-valuedDFT of the same length", *IEEE Trans. Signal Process.*, vol. 40, no. 1, pp. 54-61, Jan. 1992.

[6]N.I. Cho, I.D. Yun, S.U. Lee, "A Fast Algorithm for 2-D DCT", *ICASSP,* vol. 3, pp. 2197-2200, 1991.

[7]W.H. Chen, C.H. Smith, S.C. Fralick, "A Fast Computational Algorithm for the Discrete Cosine Transform", I*EEE Trans. on Comm.*, Sept. 1977.

[8]Mentor Graphics Inventra Intellectual Property Products: 8x8 Discrete Cosine Transform.
http://www.mentorg.com/inventra/cores/catalog/prod_desc/dct_8x8_pd.pdf

[9]L.G. Chen, J.Y. Jiu, H.C. Chang, Y.P. Lee, C.W. Ku, "A Low Power 2-D DCT Chip Design Using Direct 2-D Algorithm", *Design Automation Conference,* pp. 145-150, 1998.

[10]T. Xanthopoulos, "Low Power Data-Dependent Transform Video and Still Image Coding", Ph.D. thesis, Massachusetts Insititute of Technology, Feb. 1999.

[11]T. Kuroda *et al.*, "A 0.9V, 150MHz, 10-mV, 4*mm*^{2}, 2-D Discrete Cosine Transform Core Processor with Variable Threshold-Voltage (VT) Scheme.", *IEEE J. Soid-State Circuits,* vol. 32, no. 11, pp. 1770-1779, Nov. 1996.

[12]Andraka Consulting Group, Inc. website on distributed arithmetic.
http://www.andraka.com/distribu.html

[13]M. Bhardwaj, R. Min, A.P. Chandrakasan, "Quantifying and Enhancing Power Awareness of VLSI Systems."

[14]IBM Microelectronics. ASIC SA-27E Databook. International Business Machines Corporation. 2000.