

VHDL Modeling of a Parallel Architecture for Computing the Discrete Cosine Transform and its Inverse

Corey Graves, Clay Gloster, Christopher Doss
High Performance Digital Signal Processing Laboratory
URL: <http://www.ece.ncsu.edu/research/bdfa/>
ECE Department, Box 7911
North Carolina State University
Raleigh, NC 27695-7911

Abstract

The research described in this paper was motivated by the need for fast computations of the Discrete Cosine Transform (DCT) and the Inverse Discrete Cosine Transform (IDCT) in real-time video encoding and decoding, respectively. Using VHDL for functional verification and a commercially available logic synthesis tool for synthesis verification, we have successfully developed a highly parallel architecture which is capable of computing both the DCT and IDCT. This architecture's potential for high performance is mainly due to the overlapping of communication and computation. We assumed that we could receive only one element of an $8 \times n$ (where n is a multiple of 8) matrix per clock cycle, in row major fashion. This $8 \times n$ block-size results from a convention in many image compression/decompression standards. Because of its pipeline structure, the architecture that we have developed can achieve a throughput of one $8 \times n$ block being processed per $8n$ clock cycles. To say that an $8 \times n$ block has been processed, is to say that the DCT or IDCT has been computed for each of the 8×8 sub-images within the block. This architecture includes 16 multipliers and 16 adders, which is a small hardware complexity compared to that of other DCT/IDCT architectures with comparable throughput. Furthermore, we developed a test bench, for the VHDL model, that used the TEXTIO library functions in such a way that it was not necessary for it to be recompiled when different types of test input were applied.

I. Introduction

The 2-D Discrete Cosine Transform (2-D DCT) and its inverse (2-D IDCT) are very important in low bit-rate coding of image and video data. For this reason extensive research has been geared toward their VLSI implementation. Because of the similarities in the DCT and IDCT, an architecture designed to compute one is appropriate for the computation of the other. This is why the DCT and IDCT functions are often implemented together on a single chip [1], [2], [3], [4], [5]. While discussing the ideas behind the development of our architecture, we will refer only to the DCT.

The 2-D DCT is defined by the following equation:

$$Y = C^t A C \quad (1)$$

where C is a constant square matrix (coefficient matrix) with the same dimensionality as A . In the context of image processing, A is a square image (or square portion of a large image). Consequently, the transform matrix, Y , is a square matrix with the same dimensionality as C and A . The original image, A , can be retrieved from the transform, Y , via the 2-D IDCT, which is given by:

$$A = C Y C^t \quad (2)$$

For a $d \times d$ size matrix A , $2d^3$ multiply/accumulates are needed to perform the 2-D DCT (or 2-D IDCT). The number of computations can be cut in half (d^3 multiply/accumulates) by using the fast computational

algorithm described in [6], which exploits the symmetry of the C matrix. Other fast algorithms that require no multiplications have been presented [7]. All of the fast algorithms assume that all elements of the A matrix are accessible before the computations begin.

In both JPEG and MPEG standards all matrices are of dimensionality 8×8 [8], [9]. For this reason, the development of the architecture discussed in this paper is based on this dimensionality. It is important to note, however, that the ideas can be used for any size square matrices. The proposed architecture is an extension of the architecture presented in [10], which begins the DCT computation as soon as the first element of A is received, assuming that it is being read in row major fashion, one element at a time. This communication constraint, is more realistic for real time encoding than those implied by [6] and [7] because of the nature in which images are scanned from a camera. Additionally, the architecture of [10] does not contain an explicit transpose memory, which is a characteristic of most other high performance architectures for the DCT. The transpose memory is the memory which stores the intermediate calculation $C^t A$ (or AC). While the authors of [10] propose an architecture which calculates the DCT for a single 8×8 image block being scanned in row major, we propose a modification to the architecture which will allow an $8 \times n$ image block to be read in row major. Given that n is a multiple of 8, the architecture will calculate the DCTs for each of the $n/8$ sub-blocks (of size 8×8) in a time multiplexed fashion. This modification does not require that any computational units be added to the architecture described in [10].

This paper is organized as follows: Section II describes the base algorithm and architecture of [10], as well as the modifications we have made for $8 \times n$ operation; Section III describes how VHDL was used as a functional verification tool for the architecture; Section IV presents the results of the VHDL simulation in terms of the accuracy of the computed data as well as system parameters such as throughput and latency; Finally, Section V gives an assessment of the architecture's feasibility and discusses future research direction.

II. Algorithm and Architecture Development

The architecture is based on three basic assumptions:

1. The input matrix, A (an 8×8 matrix), is being read in row major fashion.
2. Data from A is read one element per clock cycle.
3. Any element of the constant matrix, C , is accessible on any clock cycle.

Given these constraints, the design problem is to overlap communication and computation such that the 2-D DCT is computed as efficiently as possible.

A. Stage 1

Since a full row of A is received every 8 clock cycles, and the first full column of A is not received until after 57 cycles, it is more reasonable to compute AC first, as opposed to $C^t A$. That is, AC should be calculated first because it requires rows of A to be multiplied by columns of C , whereas for the latter calculation, columns of A are needed. It can be easily seen how the simple multiply/accumulate unit (MAC) in Figure 1 can calculate a single column of AC at a rate of one element per 8 cycles. That is, every time eight elements of A (an entire row) are read, one element of the k th column of AC is generated by the MAC. In Figure 1 the variable j increments on every clock cycle and is reset to 1 on the cycle after $j = 8$, at which time the variable i is incremented. When $i = 8$, the calculation of the entire k th column of AC has completed. The other input of this MAC is the k th column of C .

The calculations of all columns of AC can be generated in parallel, due to the fact that they are independent of each other. A parallel structure that does this is shown in Figure 2. Here 8 of the MACs share the A matrix

as one of their inputs. However, the other input of each MAC receives the column of C that corresponds to

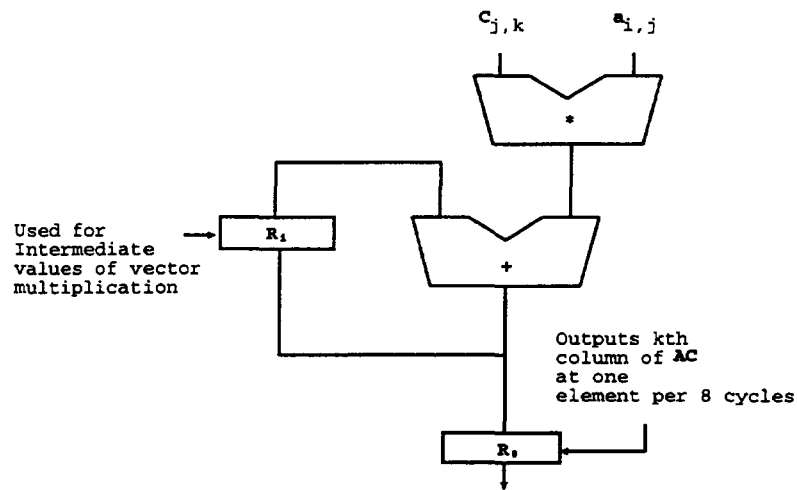


Fig. 1. Multiply/Accumulate Unit (MAC) for k th Column of AC

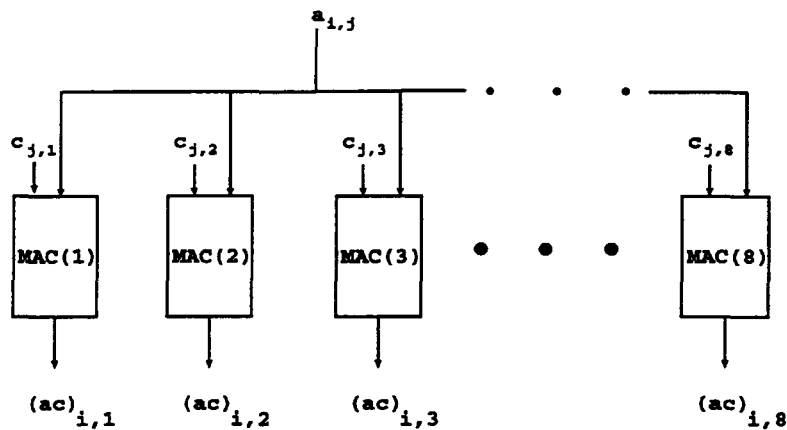


Fig. 2. Stage 1: Calculation of all Columns of AC in Parallel

the column of AC which that particular MAC is responsible for generating. That is, the MAC that generates the k th column of AC , has as one of its inputs all elements of A , and as its other input only the k th column of the C matrix. The variables j and i behave as described for Figure 1. Notice that the MACs in this figure produce an entire row of AC every 8 clock cycles.

B. Stage 2

Stage 2 performs the $C^t(AC)$ (complete 2-D DCT) calculation. Again, examine the k th column of AC . As Figure 1 illustrates, the k th column of AC is generated one element per 8 cycles. Since the entire k th column of AC is needed to calculate the k th column of $C^t(AC)$, the complete 2-D DCT cannot be computed until the final row of AC is output from Stage 1, which is 64 clock cycles after the first element of A is read. A column of $C^t(AC)$ can be computed in the same manner as a row of AC is computed in Stage 1. The MACs that compute the k th column of the complete 2-D DCT are shown in Figure 3. Here, i and j are incremented the same as before. Each output of Stage 1 ($k = 1$ to 8) goes into a structure similar to that in Figure 2. Thus, Stage 2 contains a total of 64 MACs. As the architecture has been described so far, all 64

elements of the 2-D DCT of A are output simultaneously 64 cycles (when $i = 8$) after the initial value of A is read.

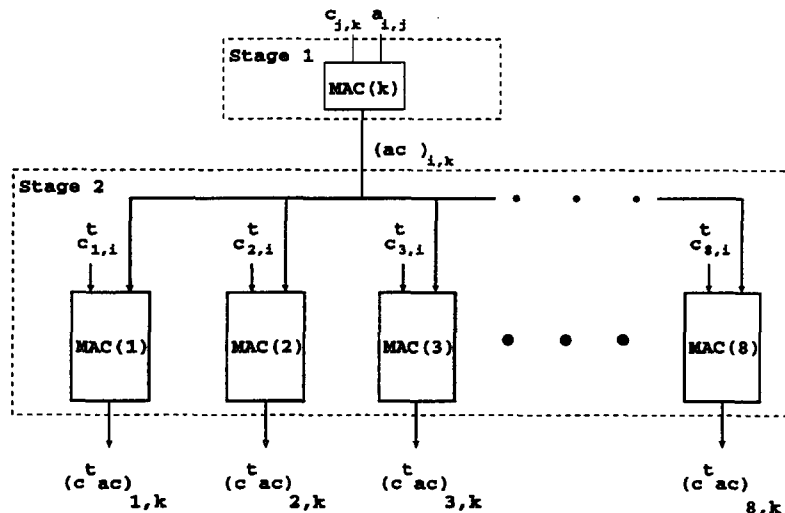


Fig. 3. Calculation of the k th Column of $C^t AC$

C. Handling System Inefficiency

The architecture described above requires 72 MACs: 8 in Stage 1 and 64 in Stage 2. Notice that the MACs in Stage 1 are fully utilized. Every time an element is read from A (every clock cycle) a multiply/accumulate operation is performed by each MAC in Stage 1. On the other hand, each MAC in Stage 2 has a utilization of $1/8$. Each one is required to perform only 1 multiply/accumulate operation per 8 clock cycles, because new data is coming from Stage 1 every 8 cycles. The authors in [10] introduce the idea of the Time-Multiplexed Multiply/Accumulate Unit (TMAC) in order to get rid of this inefficiency. Look at only one of the outputs of Stage 1 again (the k th column of AC) in Figure 3. Assume that any of the multiply/accumulate operations of Stage 2 can be done in any one of the eight available cycles. Now assume that the 1st MAC chooses to operate on its data in cycle 1, the 2nd operates in cycle 2, etc. If this is the protocol, only one of the MACs in Stage 2 of Figure 3 is active within any given cycle. For this reason the 8 MACs can be merged into a single TMAC, which is shown in Figure 4. The TMAC in Figure 4 differs from one of the MACs of Figure 3 in that it uses the entire C matrix as input instead of just one row, and also in that it outputs intermediate data to a bank of shift registers instead of a single register. Note that the MAC, in Figure 1, is simply a special case of the TMAC where the number of shift registers is 1.

The result is that only 16 fully utilized multiply accumulate units are needed, instead of 72. However, there is an increase in both memory (registers) and system latency. Before the TMACs were put into Stage 2, in place of the multiple MACs, the entire DCT was computed 64 cycles after the initial input of A . With the TMAC implementation, though the first row of the 2-D DCT will be computed 64 cycles after the first element of A is read, the last row will not be computed until another 8 cycles. This is a small increase in latency, considering the substantial decrease in computational hardware.

A throughput of 1 DCT operation per 64 clock cycles can still be achieved through pipelining. In the 8 cycles that Stage 2 is outputting the rows of the DCT, Stage 1 can be reading in the first row of a new input matrix, A . So now, even though the DCT for a given matrix is completed 72 cycles after its first element is read, it is completed only 64 cycles after the DCT for the previous input matrix is completed. So, in general, the time to process b consecutive 8×8 blocks is given by:

$$T_p(b) = (72) + (b - 1)(64)(cycs) \quad (3)$$

The first term in the equation takes into account the fact that the first block is processed in 72 cycles. The second term considers that the remaining $b - 1$ blocks are processed at a rate of one block per 64 cycles, because of pipelining.

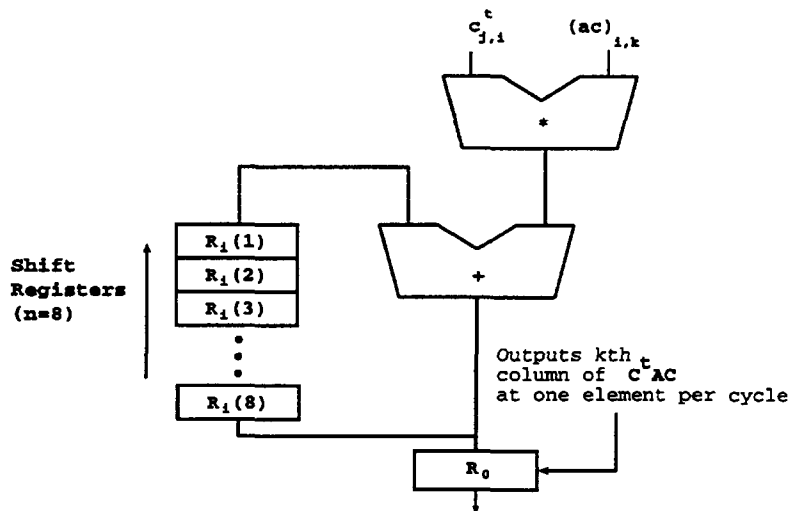


Fig. 4. Time-Multiplexed Multiply/Accumulate Unit (TMAC) Used in Stage 2

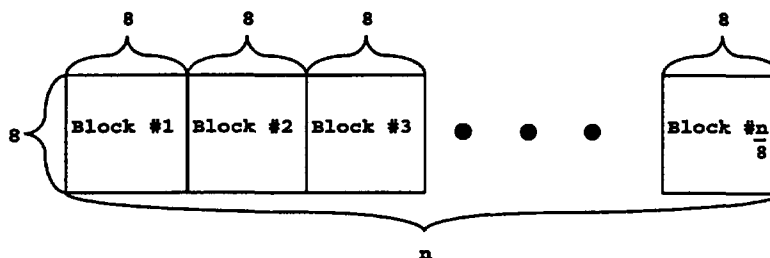


Fig. 5. An $8 \times n$ Block of an Image

D. Extension to $8 \times n$ Block

The architecture proposed by the authors in [10] can be extended to operate on 8×8 sub-blocks of an $8 \times n$ (shown in Figure 5) block being read in row major. The new architecture would operate such that when it receives the first row of the 2nd sub-block it starts a new DCT calculation, and does the same when the first row of the 3rd sub-block is received, etc. Computations continue in this manner such that 8 cycles after the last element of the $8 \times n$ block is read, the DCTs for all sub-blocks within the block are completed. This architecture modification involves an increase to n (instead of 8) shift registers in each TMAC of Stage 2, as well as some simple changes in the control of coefficient data flow. This new architecture can be pipelined as before. One can see the usefulness of the architecture, if an $m \times n$ (both m and n are multiples of 8) image is viewed as several $8 \times n$ blocks being read one after the other. That is, the $m \times n$ image is being read in row major fashion. This is typically the way an image is scanned from a camera.

In general, the algorithm operates as follows:

- An $8 \times n$ matrix is read one element at time in row major fashion.

- The DCTs for all $n/8$ sub-blocks (of size 8×8) are completed $8n + 8$ cycles after the initial element of the block is read.
- A system throughput of $n/8$ DCT calculations per $8n$ cycles can be achieved through pipelining.

Note that the initial architecture, described for a single 8×8 matrix, is a special case of this extended architecture, where $n = 8$. A general equation for the time that it takes to process an entire $m \times n$ image is:

$$T_p(m, n) = (8n + 8) + (m/8 - 1)(8n)(cycs) \quad (4)$$

where m and n are multiples of 8. The first term in the equation takes into account the fact that the first block is processed in $8n + 8$ cycles. The second term considers that the remaining $m/8 - 1$ blocks are processed at a rate of one block per $8n$ cycles, because of pipelining.

III. VHDL Modeling

The main intent of this research was to verify the architecture for DCT/IDCT processing of a series of $8 \times n$ blocks as described in the previous section. Since functionality of the architecture was the main concern, the structures in the previous section were described, in VHDL, at more of a behavioral level than a structural one.

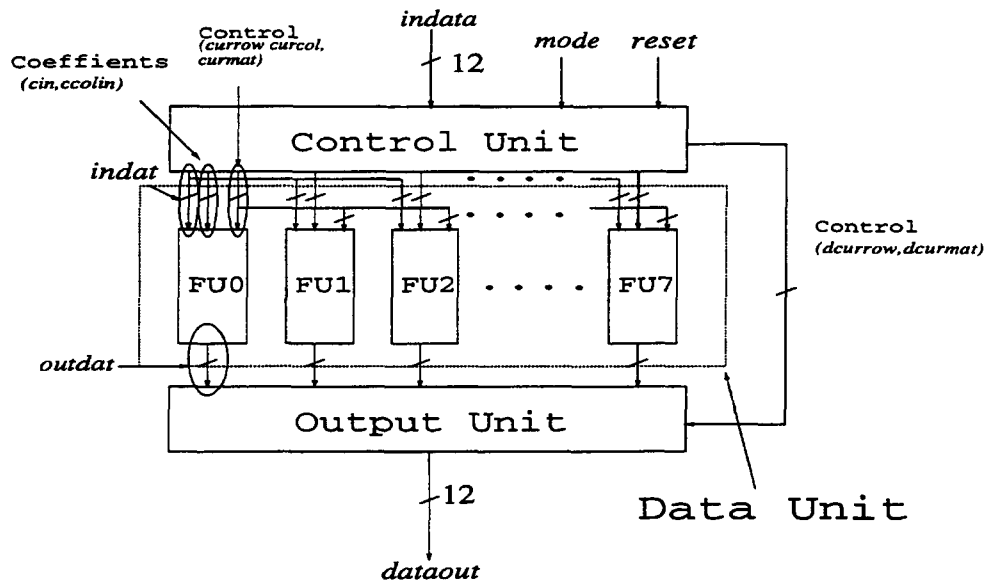


Fig. 6. Final System Architecture for DCT/IDCT

A. High Level Architecture

At its highest level, the architecture was described structurally. As shown in Figure 6, the architecture consists of three major modules: a Control Unit, a Data Unit, and an Output Unit. The function of the Control Unit is to pass the correct input and coefficient data to the Data Unit at the appropriate times, as well as to provide the Data Unit and the Output Unit with control signals. The way in which coefficient data is passed into the Data Unit is dictated by what mode is specified by the *mode* input bit. The Data Unit consists of 8 identical Functional Units (FUs) that operate in parallel. Each FU computes a different column of the DCT/IDCT matrix, and is divided into two sub-modules, one contains a MAC, as described in Stage 1 of the architecture, and the other contains a TMAC, as described in stage 2. The DCT/IDCT is output

from the FUs an entire row per clock cycle. Taking practical communication constraints into consideration, we described an Output Unit which buffers this high bandwidth output to cause data to be output from the system one element per cycle. This increased the system latency from $8n + 8$ to $16n$, thus changing the first term of eq. 4. We define the system latency as the time from the first system input to the time of the last output.

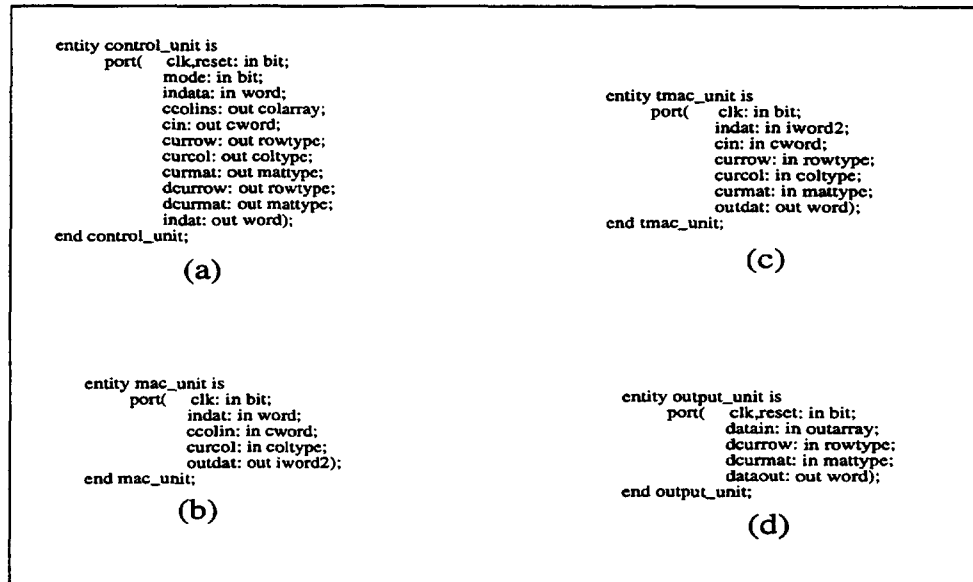


Fig. 7. Primary VHDL Entities: a) Control Unit b) MAC c) TMAC and d) Output Unit.

B. Primary VHDL Entities

The VHDL entity declarations for all of the primary modules are shown in Figure 7. Figure 7a shows the declaration for the **control_unit**. The *mode* input is a bit that determines whether the architecture performs the DCT (*mode* = '1') or the IDCT (*mode* = '0'). The system input goes into the port called *indata*, which accepts a 12 bit, 2's complement number. When in DCT mode, only 9 of the 12 bits of *indata* are used, but when in IDCT mode all 12 bits are used. This input is routed directly to the *indat* output port. Both *ccolins* and *cin* are ports by which the matrix *C* is output from the **control_unit**. The *ccolins* port repeatedly outputs the entire *C* matrix at a rate of one column per clock cycle, while the *cin* port repeatedly outputs *C* in column major fashion, at a rate of one element per clock cycle. The ports *currow*, *curcol*, and *curmat* are control signals used by the Data Unit, which indicate the row, column, and sub-image, respectively, associated with the current input element. The *reset* bit is used to synchronize these controls signals with the input data. The ports *dcurrow* and *dcurmat* are control signals used by the Output Unit, which are the one clock cycle delays of *currow* and *curcol*, respectively.

Two entities, **mac_unit** and **tmac_unit**, were used to model the Data Unit. The entity **mac_unit**, shown in Figure 7b, models the simple MAC shown in Figure 1. The input ports *indat* and *curcol* of this entity correspond directly to the *indat* and *curcol* output ports of the **control_unit** entity, while *ccolin* corresponds to only one element of the *ccolins* array of **control_unit**. Finally, the *outdat* port outputs one column of the *AC* matrix at a rate of one element per 8 clock cycles. This port passes values that range from -16384 to 16383 (15 bits), as specified by type *iword2*.

The *outdat* port of **mac_unit** corresponds directly to the *indat* port of the **tmac_unit** entity (Figure 7c), which models the TMAC shown in Figure 4. The ports *cin*, *currow*, *curcol* and *curmat* of this entity correspond to the ports with the same names in the **control_unit** entity. Finally, the *outdat* port outputs

one column of the final C^tAC matrices at a rate of one element per cycle (after an initial latency of $8(n-1)+8$ from the time of the first element of the input matrix). This port passes values that range from -2048 to 2047 (12 bits), as specified by type *word*. The two entities, *mac_unit* and *tmac_unit*, constitute one of the FUs shown in the Data Unit of Figure 6.

The entity declaration of *output_unit* is in Figure 7d. The *datain* port is an array of data of the type *word* (outputs of all of the FUs). The *dcurrow* and *dcurmat* ports are control signal from *control_unit*. The *dataout* port, the final system output, is of type *word* and is the resultant transform (or inverse transform) block being output one element at a time in row major order.

The separate VHDL code for each module and sub-module described above was combined to form the VHDL model for the diagram shown in figure 6. The VHDL code for this architecture is flexible because it allows crucial system parameters to be changed with relatively little effort. For example, the number of sub-blocks within a block, the dimension of the sub-blocks, and the bit-widths of internal busses can be modified by simply changing constants within the package that we associate with the code.

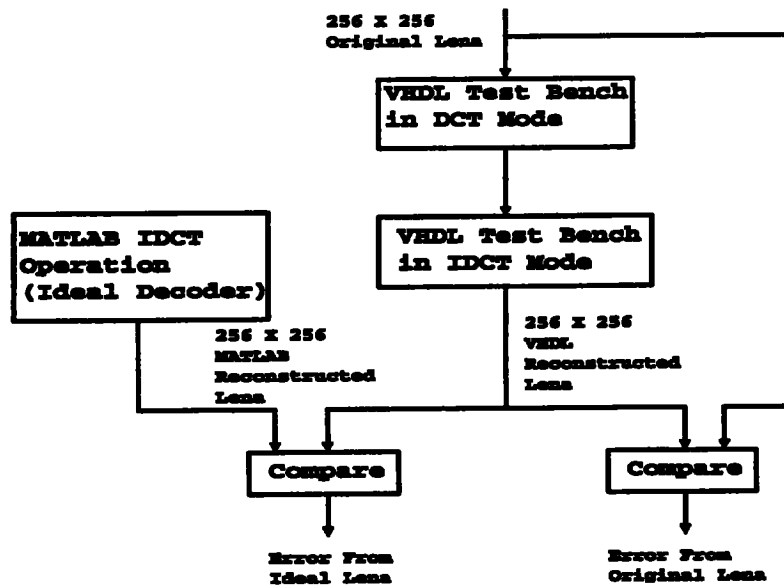


Fig. 8. Experimental Procedure for VHDL Model

IV. Experiment and Results

Once we verified that each module was able to be synthesized using a commercially available synthesis tool [11], the entire system was verified using a VHDL test bench and the 256×256 version of the test image “Lena” as input data. Given the number of columns of this image, we needed to simulate an architecture that processed a series of 8×256 ($n = 256$) blocks. The use of the TEXTIO VHDL library to read and write data from and to text files proved to be very beneficial for efficient functional verification. The experimental set up, illustrated graphically in Figure 8, was as follows:

1. The input image is stored in a file called “test.in” (in row major fashion).
2. While in DCT mode the test bench read the data from “test.in” into the architecture model, and wrote the results to a file called “test.out” (in row major fashion).
3. “Test.in” was copied to a file called “original”, and “test.out” was copied to “test.in”.

4. While in IDCT mode the test bench read the new "test.in", and wrote the results to a new "test.out". This was an approximate representation of the original image. MATLAB performed an Ideal IDCT on "test.in", and output it to a file called "idealOut".
5. "Test.out" was compared to both "original" and "idealOut".

The three files in step 5 should be virtually identical. Table I shows some error measurements that we gathered from the experiment. The three versions of "Lena" corresponding to the three files is shown in Figure 9. The simulation proved that the throughput of the system is indeed $n/8$ DCT/IDCT calculations

TABLE I
COMPARISON OF RECONSTRUCTED DATA TO ORIGINAL AND IDEAL DECODER DATA

Measurement	Compared to Original Data	Compared to Ideal Reconstruction
Peak pixel error	6.0	2.8661
mean error	1.7516	1.0954
mean square error	0.5935	0.6101

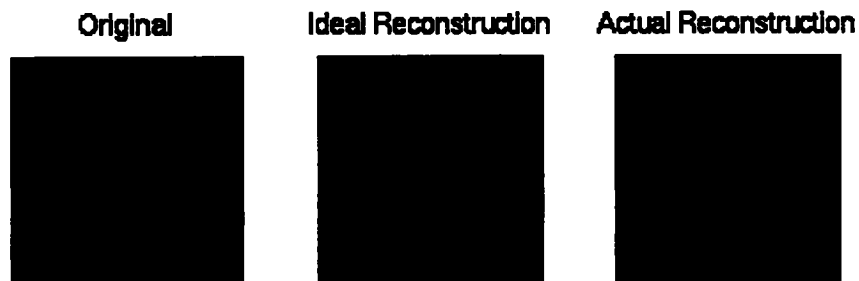


Fig. 9. Different Versions of Lena

per $8n$ cycles. We found that the last element of a transformed block was output from the Output Unit $16n + 2$ cycles after the first element of the non-transformed block was read. This is the overall system latency. The total number of cycles needed to process the 256×256 (65536 pixel) image was 67586. In general the processing time, T_p , for an $m \times n$ image is given by:

$$T_p(m, n) = (16n + 2) + (m/8 - 1)(8n)(cycs) \quad (5)$$

where both m and n are multiples of 8. The first term in the equation takes into account the fact that the first block is processed in $16n + 2$ cycles. The second term considers that the remaining $m/8 - 1$ blocks are processed at a rate of one block per $8n$ cycles, because of pipelining.

V. Conclusion

The error between the reconstructed image created by our VHDL model and the ideal reconstruction of the image is virtually non-perceptible with the human eye. However, the quantitative errors between the actual and ideal reconstruction, recorded in Table I, are significantly greater than those specified for the IDCT by the JPEG and MPEG standards [8], [9]. These errors are influenced by the choice of several bit-widths within our architecture. Namely the choice of the bit precision of the elements of the C matrix and the bit-widths of the registers within the MACs and TMACs. All of these values can be changed by modifying constants that are defined in the package associated with the VHDL model. A first step in future

research should be to alter these bit-widths until acceptable error values can be obtained for a wide range of test data.

After satisfactory system accuracy is achieved, the second phase of future research should be geared toward optimizing the hardware components of the architecture. Our VHDL simulations have proven that the architecture is capable of achieving high throughput. However, performance is based on system speed, as well as throughput. The circuit obtained from the synthesis tools [11] that we used was far from optimal in terms of system speed. For example, fully functioning multipliers were used in the synthesis of the MAC and TMAC structures. However, a closer look at the functions of the multipliers of these structures suggests that fully functioning multipliers are not needed. Notice that one input of each of the multipliers has a limited number of values that are fed into it from the constant C matrix. This implies that it would be inefficient to use fully functioning multipliers in this architecture. The authors of [5] researched the use of both hardwired or ROM based multipliers as an alternative, and found that the hardwired solution was more reasonable in their DCT/IDCT architecture, in which they were able to achieve a clock rate of 100Mhz. This and other solutions should be investigated for the architecture proposed in this paper.

Another hardware optimization that needs to be performed involves choosing the type of memory structure that should be used for the TMAC. The shift register implementation makes sense when the value of n is small (i.e. $n = 8$, as in Figure 4). When n is a fairly large number (i.e. $n = 256$, as in our test case), one can see how the system clock rate is adversely affected by overloading the clock signal with too many registers. Some alternatives would be to use either a dual ported memory or multiple banks of single ported memory in the place of shift registers. These two solutions may offer a higher system clock rate at the expense of slightly more complicated control.

A final hardware consideration would be the optimal placement of the C matrix values. That is, it should be determined if it is more efficient to broadcast these values to the Functional Units from the Control Unit, or to include a separate copy of C in each Functional Unit.

Once the refinements described in this section have been completed, our architecture can be properly leveraged against other high performance DCT/IDCT architectures. The authors of this paper believe that the realistic communication constraints used in our development lead to an architecture that is better suited for real-time video encoding/decoding than other architectures which assume input data can be randomly accessed.

References

- [1] L. Kim T. Kim and J. Kim. 'High-data-rate DCT/IDCT architecture by parallel processing'. In *Proc. SPIE*, pages vol. 2308, 895–905, 1994.
- [2] C. L. Wang and Y. T. Chang. 'Highly parallel VLSI architectures for the 2-D DCT and IDCT Computations'. In *Proc. Reg. 10's Ann. Int. Conf.*, pages vol. 1, 295–299, Aug. 1994.
- [3] T. Miyazaki et al. 'DCT/IDCT processor for HDTV developed with DSP silicon compiler'. *J. VLSI Signal Process.*, 5:39–46, June 1993.
- [4] D. Slawewski and W. Li. 'DCT/IDCT processor for design for high data rate image coding'. *IEEE Trans. on Circuits Syst. Video Tech.*, 2:135–146, June 1992.
- [5] Avanindra Madiseti and Alan N. Wilson. 'A 100 MHz 2-D 8×8 DCT/IDCT processor for HDTV applications'. *IEEE Trans. on Circuits Syst. Video Tech.*, 5(2):158–165, April 1995.
- [6] W. Chen et al. 'A fast computational algorithm for the discrete cosine transform'. *IEEE Trans. on Commun.*, COM-25:1004–1009, Sept. 1977.
- [7] J. A. Hallas E. P. Mariatos, D. E. Metafas and C. E. Goutis. 'A fast DCT processor, based on special purpose CORDIC rotators'. In *IEEE Int. Symp. Cir. Syst.*, pages 271–274, 1991.
- [8] ISO/IEC JTC1/SC29/WG10. JPEG Committee Draft CD10918. Technical report, 1991.
- [9] ISO/IEC JTC1/SC29/WG11. MPEG Committee Draft CD11172. Technical report, 1991.
- [10] S. P. Kim and D. K. Pan. 'Highly modular and concurrent 2-D DCT chip'. In *Proc. IEEE Int. Symp. On Circuits and Systems*, pages 1081–1083, May 1992.
- [11] Synopsys. 'Synopsys Online Documentation v3.4b 2.1.0/sunos5'. Interleaf, Inc., Waltham, Mass., 1995.