Energy-Efficient Discrete Cosine Transform on FPGAs

Ronald Scrofano
Department of Computer Science
University of Southern California
Los Angeles, CA

Ju-Wook Jang
Department of Electronic Engineering
Sogang University
Seoul, Korea

Viktor K. Prasanna
Department of Electrical Engineering-Systems
University of Southern California
Los Angeles, CA

Abstract— **The 2-D discrete cosine transform (DCT) is an integral part of video and image processing; it is used in both the JPEG and MPEG encoding standards. As streaming video is brought to mobile devices, it becomes important that it is possible to calculate the DCT in an energy-efficient manner. In this paper, we present a new algorithm and processing element (PE) architecture for computing the DCT with a linear array of PEs. This design is optimized for energy efficiency. We analyze the energy, area, and latency tradeoffs available with this design and then compare its energy dissipation, area, and latency to those of Xilinx's optimized IP core.**

Index Terms— **DCT, energy efficiency, performance modeling**

I. Introduction

The wireless industry is moving toward adding streaming video capabilities to mobile devices. Such capabilities require that the devices posses a great deal of processing power. Additionally, as their batteries only provide a limited amount of energy, mobile devices operate in an energy-constrained environment. It is evident, then, that high throughput and low latency are not the only concerns in this application area. It is also necessary that the image processing is energy efficient.

FPGAs are an option for streaming video as well as image processing in general. The regular structure of FPGAs is well-suited to the regular nature of the computations in image processing applications. Further benefits are derived from taking advantage of hard IP, such as multipliers or multiply-and-accumulate blocks, that FPGA vendors are now embedding into the FPGA [9], [10]. Thus, it is possible to use FPGAs for image processing tasks that require low latency and high throughput. However, as mentioned above, energy efficiency is also an important issue in image processing.

Because FPGAs can provide the computational power needed by many applications that operate in energy-constrained environments, energy-efficient design for FPGAs is an emerging field. However, there are currently no commercially available FPGAs that combine millions of gates with low-power features. As a result, we design our *algorithms and architectures* for energy efficiency. These designs will remain energy-efficient for the next generation of FPGAs that include low-power features; the algorithms can simply be augmented to utilize these new features.

In this paper, we present an energy-efficient algorithm and architecture for the 2-D DCT. The 2-D DCT is an important kernel in image processing as it is used in the MPEG and H.263 streaming video standards as well as the JPEG standard for still images[1]. We compare the performance estimates for the latency, energy, and area of our design to those of the Xilinx IP core for 8×8 DCT in order to demonstrate the efficiency of our design.

The remainder of this paper is organized as follows. The next section presents related work. Section III describes the new algorithm and architecture for 2-D DCT. Section IV presents optimization techniques used in this design. In Section V, we present the domain-specific model for the design. Section VI describes some of the energy, area, and latency tradeoffs available with this architecture and algorithm for computing the DCT. Section VII compares our design to the Xilinx IP core for 2-D DCT. Finally, Section VIII concludes the work and presents directions for future work.

II. Related work

There is much work related to both the DCT and energy-efficient mappings of algorithms onto FPGAs, though we are not aware of any that combine the two, as is done in this paper.

In [4], a systolic array architecture and algorithm for computing the 2-D DCT is described. This architecture uses a 2-D mesh structure, which differs from our design. In our design, we use a linear array rather than a 2-D mesh. Doing so decreases the amount of interconnect that is required by our design. We have chosen this approach because interconnect in FPGAs can dissipate much energy.

[1] and [8] each describe implementations of the DCT algorithm on FPGAs. Each method is different from the one employed in this paper. [1] describes DCT computation using polynomial transforms. It describes both the method for calculating DCTs using polynomial transform and an FPGA datapath for doing so. As background information, this paper also describes computing the DCT using the distributed arithmetic technique. The distributed arithmetic technique is featured in [8]. A distributed arithmetic scheme as described in [8] is used in the Xilinx IP Core to which we compare our results [9]. The distributed arithmetic approach performs a row-wise 1-D DCT using multiple FIR filters and then column-wise 1-D DCT on the row results, requiring a transpose memory in-between. Our design is significantly different from the approaches in [1] and [8]. Our design uses neither the bit-level input data arbitration circuit for distributed arithmetic nor

the transpose memory. We also do not follow the polynomial transform DCT calculation. Instead, we employ a modular design that uses $p \leq n$ copies of the PE in Figure 2 for the computation of a 2-D DCT of an $n \times n$ matrix. The method in which it does so is similar to performing two $n \times n$ matrix multiplications.

[2] and [3] describe energy-efficient design and performance modeling, respectively, for FPGAs. [2] characterizes sources of energy dissipation in FPGAs and presents techniques that can be used during the design of an algorithm and architecture in order to develop energy-efficient designs. We employ these techniques in the design of our algorithm and architecture for the 2-D DCT (see Section IV). [3] details the domain-specific modeling technique for the performance modeling of algorithms and architectures mapped onto FPGAs. We use this technique in the derivation of performance estimates for our 2-D DCT design (see Section V).

III. Energy-Efficient Algorithm and Architecture for the 2-D DCT

The 1-D DCT of an n-element vector B is computed as

$$Y_k = a_k \sum_{i=0}^{n-1} b_i \cos\left(\frac{2\pi}{4n}(2i+1)k\right), k = 0, \ldots, n-1 \quad (1)$$

where the b_i are the n elements of B and $a_k = \frac{1}{\sqrt{n}}$ for $k = 0$ and $\sqrt{\frac{2}{n}}$ otherwise.

Let X be an input $n \times n$ matrix. Conventionally, the 2-D DCT is computed by the row-column approach in which a 1-D DCT of each of the rows of X is computed and a 1-D DCT is then computed on the each of the columns of the result. This approach is equivalent to finding the result of the DCT to be Z in Equation 2 where C denotes the coefficient matrix, shown below for an 8×8 DCT. In the coefficient matrix, $C_k = \cos\left(\frac{2k\pi}{4n}\right), k = 0, \ldots, n-1$. It is important to note that there are only eight distinct magnitudes in C. That is, ignoring the sign of the entry, there are only eight distinct values stored in the coefficient matrix. Further, in a given row, the entry in the first column has the same magnitude as that in the last, the entry in the second column has the same magnitude as that in the second-to-last, and so on in toward the middle columns. We exploit this property in our algorithm, as will become clear below.

$$Z = CXC^T \quad (2)$$

$$C = a_k \begin{bmatrix} C_0 & C_0 & C_0 & C_0 & C_0 & C_0 & C_0 & C_0 \\ C_1 & C_3 & C_5 & C_7 & -C_7 & -C_5 & -C_3 & -C_1 \\ C_2 & C_6 & -C_6 & -C_2 & -C_2 & -C_6 & C_6 & C_2 \\ C_3 & -C_7 & -C_1 & -C_5 & -C_5 & C_1 & C_7 & C_3 \\ C_4 & -C_4 & -C_4 & C_4 & C_4 & -C_4 & -C_4 & C_4 \\ C_5 & -C_1 & C_7 & C_3 & -C_3 & -C_7 & C_1 & -C_5 \\ C_6 & -C_2 & C_2 & -C_2 & -C_2 & C_2 & -C_2 & C_6 \\ C_7 & -C_5 & C_3 & -C_1 & C_1 & -C_3 & C_5 & -C_7 \end{bmatrix}$$

In our design, we use a linear array of PEs to compute Z. The input data for the linear array can be stored in on-chip memory or off-chip memory. In this paper, we do not consider the memory cost of storing the input or the output,

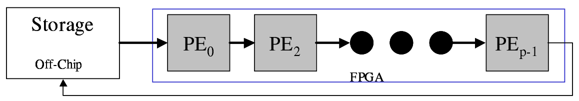

Fig. 1. Linear array of processing elements with data stored in off-chip memory

though we do consider the cost of storing and reading the intermediate results. A diagram of an off-chip linear array is shown in Figure 1. The PEs that constitute the linear array are shown in Figure 2. The DCT is calculated in two phases: the first phase computes $D = XC^T$ and the second computes $Z = CD$.

A. Phase 1

In Phase 1 of the algorithm, the entries from X are input in a modified row-major order. The first input is from column 0, the next from column $n-1$, then column 1, column $n-2$, and so on, towards the middle of the row. For example, for an 8×8 DCT, the order of the input of the second row of X will be x_{10}, x_{17}, x_{11}, x_{16}, x_{12}, x_{15}, x_{13}, x_{14}. This input pattern facilitates the exploitation of the pattern in the coefficient matrix that is described above. For example, when the second row of X is multiplied with the second column of C^T in PE$_1$, x_{13} and x_{14} are both multiplied by C_7, though C_7 is negated when multiplied with x_{14}. In our algorithm, rather than storing both positive and negative C_7 and performing two multiplications (one with x_{13} and one with x_{14}), x_{13} is added to $-x_{14}$ and the result is multiplied by positive C_7 ($x_{13}C_7 + x_{14}(-C_7) = (x_{13} - x_{14})C_7$). This method reduces the number of coefficient memory accesses and the number of multiplies by a factor of 2. A linear of array of the PEs shown in Figure 2 computes the product $D = XC^T$ in this fashion.

Each PE$_j$ computes the jth column of the intermediate matrix D. The coefficient memory in each PE stores the magnitudes of the unique coefficients in C. During Phase 1, multiplexers M1 and M2 both select input line 0. Input from X enters the PE through IOL1 and is stored in register R1 every clock cycle. Data in R1 is passed to the next PE through IOR1. Additionally, after being stored into R1, data is alternately written into registers R2 and R3 (first R2, then R3, then R2, and so). Every other clock cycle, the data in R2 and R3 are negated if necessary, added together, and stored in R4. On the clock cycle following its storage into R4, the data now in R4 is multiplied with the appropriate coefficient from the coefficient memory (this coefficient is stored in R7 on the same clock cycle that data is stored in R4). The product of this multiplication is stored in R5. R6 is used to accumulate successive products until a complete intermediate value is stored in R6. At this point, the data in R6 is written into the data memory and R6 is cleared. When an entire column of the intermediate matrix D has been computed in this fashion, the algorithm switches to Phase 2.

B. Phase 2

Phase 2 computes the matrix product $Z = CD$ to finish the 2-D DCT computation. This phase performs almost identically to Phase 1. PE$_j$ computes column j of the Z matrix.

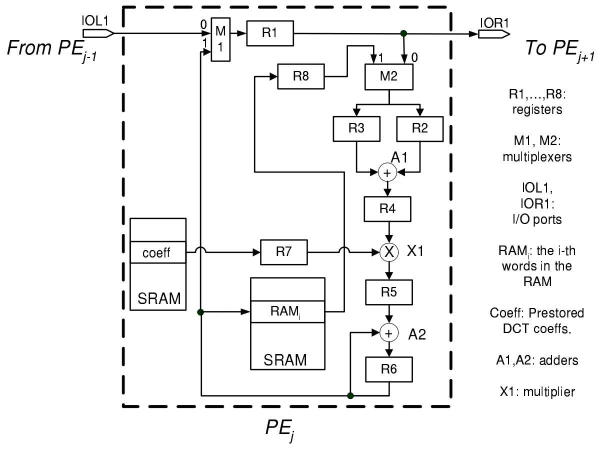

Fig. 2. PE$_j$ of the array for DCT computation

In this phase, rather than taking input from outside, each PE uses the intermediate data stored in the data RAM for the computation. Thus, M2 selects input line 1. A further difference is that the data accumulated in R6 is not stored in the data RAM. Instead, it is written into R1. On the cycle of writing to R1, M1 selects input line 1. During the rest of the cycles, M1 selects input line 0. This is so that there is path through the linear array that allows the output of completed entries of the result matrix Z to be output from the array through the final PE. The outputting of finished entries is sequenced so that no entries are overwritten somewhere in the array. To achieve this, the data from PE$_{j-1}$ is written into R1 of PE$_j$ one cycle before the data in R6 of PE$_j$ is written into R1 of PE$_j$. Additionally, the first data to leave the array is that of the last PE in the array. Thus, the data leaves the array in a sort of reverse row-major order. For example, in an 8×8 DCT, the first output of the linear array is z_{07}, followed by z_{06}, and so on, until z_{70} is finally output. Once the first entry leaves the array, a new entry leaves the array on every clock cycle until all n^2 entries of Z have left the array. At this point, the PE can go back to Phase 1 and compute another DCT.

IV. Optimizing Energy Performance

To optimize the energy performance of our design, we employ several energy-efficient design techniques [2]. One such technique is *architecture selection*. FPGAs give the designer the freedom to map almost any architecture he chooses onto hardware. Different architectures have varying energy performances as well as latencies, throughputs, and areas. In our DCT design, we have chosen a linear array of processing elements. In FPGAs, long interconnects dissipate a great deal of power[9]. Therefore, it is beneficial in energy-efficient designs to minimize the number of long interconnects. A linear array of PEs accomplishes this goal. Each processing element only communicates with its nearest neighbors, minimizing the use of long wires.

Additionally, the linear array architecture facilitates the use of two more techniques: *parallel processing* and *pipelining*. Both parallel processing and pipelining decrease the effective latency of a design. Parallel processing does so by increasing the amount of resources while pipelining does so by increasing the resource utilization. Energy is the product of latency and average power dissipation. So, by decreasing effective latency, both techniques can lead to lower energy dissipation. Further, the reduced latency reduces the amount of energy dissipated due to quiescent power dissipation during the computation. Such a reduction is important in SRAM-based FPGAs, such as the Xilinx Virtex-II, which have a high quiescent power dissipation. Parallel processing and pipelining do, however, have a drawback: they can also lead to increased power dissipation because more computation and storage units are active at a given time. This increased power dissipation can have an effect detrimental to energy efficiency. The designer must strike a balance between low latency and high power in order to achieve low energy. In our design, each of the PEs in the linear array computes one column of the Z matrix in Equation 2. Thus, minus some latency for moving data into the array, each of the PEs is working in parallel to compute the intermediate and result matrices. Within each PE, the calculations are pipelined. Each phase of the algorithm has several stages like adding, multiplication, accumulation. The data is pipelined from one stage to the next.

Another technique that we use to our advantage is *choosing the appropriate bindings*. In an FPGA, there can be many possible mappings of computation and storage elements to the actual hardware. For example, in the Xilinx Virtex-II, storage can be implemented as registers, distributed memory, or embedded block RAM. Each of these types of storage dissipates a different amount of energy and can lead to implemented designs with widely varied energy dissipations. Choosing the appropriate storage binding, therefore, is crucial to energy-efficient design. Similar decisions can be made for other elements of the design, such as choosing between embedded multipliers or configured multipliers. In our design, we choose distributed memory for the coefficient and data memories and we choose embedded multipliers.

Finally, and possibly most importantly, *the algorithm itself can be designed for energy efficiency*. The energy dissipated by a given component is proportional to the switching frequency of that component. In the algorithm design, it is possible to reduce the switching frequency of the *hot components*. The hot components are those which dissipate a large portion of the total energy. For our design, the hot components are the multipliers, memory, and registers. As a result, our design, as described in Section III, has been designed to reduce the number of accesses to each of these components. The modified row-major order in which the inputs enter the PE is an example of this. By changing the input order, we are able to add two inputs together before multiplication, rather than multiplying with each input and then adding. In this manner, we reduce the amount of multiplications by 50%. We also reduce the number of accesses to the coefficient memory and the number of writes to register R7. Additionally, storing only the absolute values of the coefficients reduces the algorithm's storage requirement and thus its power dissipation. This reduction in power dissipation does not increase the latency, so it results directly in energy savings. Further study of the characteristics of the coefficient matrix shows that all entries of row 0 are identical, as are all the entries of row 4. This fact can be used to further reduce the number of accesses to the coefficient memory. In fact, it is possible to reduce the number of read accesses to the coefficient memory to $\frac{1}{3}$ of those that are necessary in a direct implementation of the 2-D DCT based on the defining equation (Equation 1).

In order to evaluate whether or not these techniques have led to an efficient design in this case, we employ domain-specific modeling. This technique allows us to do this evaluation before investing time in implementing the design and performing time-consuming, low-level simulations.

V. Domain-Specific Energy Model

To create this design, we have employed *domain-specific modeling* (for a thorough description of domain-specific modeling, see [3]). Domain-specific modeling is a hybrid (top-down plus bottom-up) approach to performance modeling that allows the designer to rapidly evaluate candidate algorithms and architectures in order to determine the design that best meets his criteria. Using this technique, we developed and evaluated several designs, finally settling on that which is the most energy-efficient.

In domain-specific modeling, an architecture is divided into *relocatable modules* (RModules) and *Interconnects*. RModules are hardware elements that are assumed to dissipate the same amount of power no matter where they are instantiated on the chip and Interconnects are the wires connecting the RModules. From the algorithm, the designer knows when and for how long each RModule is active. With this knowledge, the designer can calculate the latency of the design. Additionally, with estimates for the power dissipated by each RModule and the Interconnect, the designer can estimate the energy dissipated by the design. Finally, if the designer knows the area required by each RModule, he can estimate the area that will be occupied by the design. Deriving the equations to estimate energy, area, and latency is the top-down portion of domain-specific modeling because it requires that the designer analyze the algorithm and architecture. Finding the constants for the energy and area equations is the bottom-up portion because it requires that the designer perform low-level simulations of the RModules and Interconnects to obtain constants for the model. These constants give the power dissipation of each component for a given operating frequency and switching activity as well as the area of each component. Note that the same simulation data for an RModule can be used in the evaluation of many different designs and that simulating an RModule is much less time-consuming that simulating an entire design.

In our design, the RModules are multipliers, adders, multiplexers, RAM, and registers. The majority of the Interconnect costs are captured in the simulation of the RModules so we do not consider any Interconnect separately..

In the model described below, the $n \times n$ DCT is computed using $p \leq n$ PEs. We assume that n is divisible by p. Each PE calculates $\frac{n}{p}$ columns of the intermediate and final matrices.

A. Latency Estimation

The equation for the latency can be determined by analyzing each phase of our algorithm and the PE. In Phase 1, 6 cycles are required for the first data to be stored in R6. Per intermediate result, there are $\frac{n}{2}$ writes into R6. After a write, R6 is idle for one cycle and is written into on the cycle after that. Thus, it takes $6 + \frac{n}{2}(2) - 1 = n + 5$ cycles for the first entry to be completed in R6 and written into the data SRAM. Each PE calculates n such entries for each column of the intermediate matrix that it computes. However, for all but the first entry calculated, the pipeline of registers is full and thus only $\frac{n}{2}(2)$ cycles are required to compute an entry and write it into the RAM. Thus, one entry for the first intermediate column calculated by a PE in Phase 1 requires $n + 5$ cycles to be computed while the other $n - 1$ entries in that column require $\frac{n}{2}(2)$ cycles to be computed. Therefore, the latency for computing one intermediate column in Phase 1 is $n^2 + 5$

When $p < n$, there are two modes in which the algorithm can operate. In the first, it computes all the intermediate results in Phase 1 and stores them. This mode will be referred to as the nonalternating mode and its latency will be denoted as L_{nonalt}. In the second mode, each PE computes one intermediate column in Phase 1, then computes a result column in Phase 2, then computes another intermediate column in Phase 1, and so on. This will be referred to as the alternating mode and its latency will be denoted as L_{alt}. In the nonalternating mode, the pipeline inside the linear array remains full, so the latency for Phase 1 is $\frac{n}{p}\left(n^2\right) + 5$. In the alternating mode, the pipeline must be refilled every time Phase 1 begins, so the total combined latency for all the Phases 1 is $\frac{n}{p}\left(n^2 + 5\right)$.

Phase 2 requires the same steps as Phase 1, only the inputs are the intermediate results rather than the matrix to be transformed. Thus, the Phase 2 latency equations are the same as those for Phase 1.

Regardless of mode, computation in Phase 1 does not overlap with computation in Phase 2. Consequently the sum of L_{p1} and L_{p2} for each mode is the total latency for one PE in the calculation of the DCT. The last PE in the linear array is $p-1$ cycles behind the first PE, so a delay of $p-1$ must be added to the total latency. Beyond this delay, there is another $p-1$ cycle delay for the output from the first PE to be output at the end of the array. In nonalternating mode, these delays only happen once. In alternating mode, these delays happen for every Phase 1-Phase 2 pair. Combining this information and the latencies for each phase into an equation for each mode and simplifying, we find that the total latencies are

$$L_{\text{nonalt}} = 2\left(\frac{n^3}{p}\right) + 2p + 8 \tag{3}$$

$$L_{\text{alt}} = \frac{n}{p}\left(2n^2 + 2p + 8\right) \tag{4}$$

B. Energy Estimation

To estimate the energy dissipation of our design, we analyze the algorithm to determine the activity of each of the components in each PE. This analysis will give us the duration of

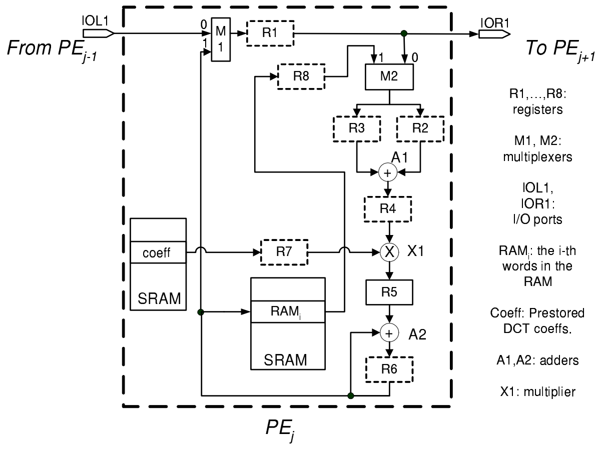

Fig. 3. PE$_j$ of the array for DCT computation. Registers that are part of other components are shown with dashed lines for borders.

time that each component is active. Multiplying this time by the power dissipation of each component gives the estimate for the energy dissipation. The power dissipation of each type of component has been determined through low-level simulation. The SRAM, multiplexers, and adders were simulated as having nonregistered inputs and registered outputs. Thus, some of the registers shown in Figure 2 do not need to be included in the estimation: their values are encompassed by other components. Figure 3 shows those registers that will not be included separately in the energy dissipation computations as having dashed, rather than solid, lines for their borders.

The same amount of computation is done by each component whether the alternating mode or nonalternating mode is used. The only difference is the power dissipation of the memory: in the nonalternating method a larger amount of memory is required in each PE than in the alternating method because all the intermediate results from Phase 1 must be stored before any are operated on in Phase 2; larger memories have higher power dissipations.

1) Phase 1: In Phase 1 of our algorithm, the adders, multiplier, register R5, and coefficient RAM are accessed once every two cycles. A total of n^2 inputs are operated on per intermediate column produced by each PE. So each of these components is active $\frac{n^2}{2}$ times per intermediate column. Regardless of the number of PEs, the total number of intermediate columns produced must be n because there are n columns in the complete intermediate matrix. So the adders, multiplier, register R5, and coefficient RAM are each active for a total over all PEs over all Phases 1 of $\frac{n^3}{2}$ cycles.

Each time a PE calculates an intermediate column, it needs to read in all n^2 entries in the original matrix. Thus, the source matrix must be input to PE$_0$ $\frac{n}{p}$ times. M1 and M2 are, therefore, each active for n^3 cycles over all PEs over all Phases 2.

All the entries of the intermediate matrix must be written into data SRAM. The total number of SRAM writes, regardless of the number of PEs, is, therefore, n^2.

Combining all this information into one energy equation, we find that

$$E_1 = n^3\left[\frac{1}{2}\left(2P_{add} + P_{mult} + P_{reg} + P_{cRAM1p}\right) + 2P_{mux} + \frac{1}{p}\left(P_I\right)\right] + n^2 P_{dRAM1p} \tag{5}$$

where P_{add}, P_{mult}, P_{mux}, P_{reg}, P_{cRAM1p}, P_{dRAM1p}, and P_I are the power constants for an adder/subtracter, a multiplier, a multiplexer, a register, a 1 port coefficient RAM, a 1 port data RAM, and inputting 8-bit data, respectively.

2) Phase 2: In Phase 2, the activity of the adders, multiplier, register R5, the coefficient SRAM, and multiplexer M2 is the same as in Phase 1. There is no writing to the data SRAM but instead one element is read from it every cycle. Regardless of the number of PEs, a total of n^3 reads from data SRAM must occur: each result entry requires n reads and there are a total of n^2 entries in the result.

In this phase, multiplexer M1 in PE$_j$ is active when the output from PE$_{j-1}$ is passed into PE$_j$ as it is being output from the array and when the data in R6 in PE$_j$ is to be written to PE$_{j+1}$. Thus, M1 in PE$_j$ is active $n(j+1)$ cycles for each column of the output that it computes. So, the total over all PEs over all Phases 2 that multiplexer M1 is active is $n\left(\frac{n}{p}\right)\left(\frac{p(p+1)}{2}\right)$ cycles.

Additionally, we must account for the fact that a total of n^2 elements are output from the array. We thus estimate that the total energy dissipation in Phase 2 is

$$E_2 = n^3\left[\frac{1}{2}\left(2P_{add} + P_{mult} + P_{reg} + P_{cRAM1p}\right) + P_{mux} + P_{dRAM1p}\right] + n^2 P_O + \left(\frac{n^2\left(p+1\right)}{2}\right) P_{mux} \tag{6}$$

where P_O is the power for outputting 8-bit data and the other constants are as described above.

3) Total: Summing E_1 and E_2 gives the calculation energy dissipation. To get the total energy dissipation, the energy dissipation due to quiescent power dissipation must be added to that sum. Thus, the total energy is

$$E = E_1 + E_2 + LP_Q \tag{7}$$

where L is the latency calculated by Equation 3 or 4 and P_Q is the quiescent power dissipation of the device.

C. Area Estimation

The area of those components with dashed lines for borders in Figure 3 are encompassed within the areas of other components. Thus, the area of the PE can be estimated from the area of one register, two single port RAMs of appropriate size, two 2-to-1 multiplexers, two adders, and one multiplier. It follows that the estimated area for p PEs is

$$A = p\left(A_{reg} + A_{cRAM1p} + A_{dRAM1p} + 2A_{mux} + 2A_{add} + A_{mult}\right) \tag{8}$$

where A_{reg}, A_{dRAM1p}, A_{cRAM1p}, A_{mux}, A_{add}, and A_{mult} are the areas of a register, a single port RAM to hold the data, a single port RAM to hold the coefficients, a multiplexer, an adder/subtracter, and a multiplier, respectively.

TABLE I

VALUES FOR THE CONSTANTS IN THE ENERGY ESTIMATION EQUATION

Constant	Size	Power(mW)
P_{reg}	8 bits	1.41
$P_{RAM1p-16}$	width=8 bits, depth=16	1.85
$P_{RAM1p-32}$	width=8 bits, depth=32	5.58
$P_{RAM1p-64}$	width=8 bits, depth=64	6.45
P_{mux}	8 bits	1.85
P_{add}	8 bits	1.85
P_{mult}	8 bits	10.56
P_I	8 bits	0.63
P_O	8 bits	12.50
P_Q	N/A	150

TABLE II

VALUES FOR THE CONSTANTS IN THE AREA ESTIMATION EQUATION

Constant	Size	Area (slices)
A_{reg}	8 bits	4
$A_{RAM1p-16}$	width=8 bits, depth=16	4
$A_{RAM1p-32}$	width=8 bits, depth=32	16
$A_{RAM1p-64}$	width=8 bits, depth=64	16
A_{mux}	8 bits	4
A_{add}	8 bits	4
A_{mult}	8 bits	16

VI. TRADEOFFS

In this section, we analyze the tradeoffs among energy, area, and latency in this design. We will vary the number of processing elements as well as the algorithm mode. We make our comparisons for the 8×8 DCT because this is the most common size of DCT: it is used in both JPEG and MPEG as well as other standards. For the 8×8 DCT, the possible numbers of PEs in the linear array are 1, 2, 4, and 8.

The equations derived above are used to analyze the design. Based on the latency equations (Equations 3 and 4), it is easy to see that for the same number of PEs, the nonalternating mode of the algorithm will lead to a lower latency. However, the nonalternating mode requires more per PE storage than the alternating mode. The larger memory in the PEs in alternating mode leads to increased power dissipation for memory accesses as well as increased area.

Table I shows the power dissipation values for the components in our design when mapped onto the Xilinx Virtex-II, running at 100 MHz. Note that in the Xilinx Virtex-II, the minimum-sized memory holds 16 entries, so, even though some designs require a memory with only 8 entries, they must use one with 16. Table II shows the areas of the components of our design when implemented on the Virtex-II [2].

Table III presents the energy, area, and latency estimates for designs with 1, 2, 4, and 8 PEs in both operating modes when implemented on a Xilinx Virtex-II operating at 100 MHz. Additionally, it shows the product of energy, area, and time (EAT) where the energies, areas, and times have been normalized to those of the 8 PE, nonalternating mode case. For EAT, a lower value is better.

Table III shows that the 8 PE design is the fastest, most energy-efficient, and has the lowest EAT despite using the most area. This fact is largely due to the low latency of the 8 PE design and the high quiescent power of the Virtex-II. Faster computation of the DCT reduces the amount of energy dissipated due to quiescent power dissipation. The quiescent power dissipation is very high compared to the power dissipations of the components in the design. As a result, a fast design has significantly lower energy dissipation than the slower designs. The area savings of the slower designs cannot overcome the dual benefits of low latency and low energy dissipation in the 8 PE design. If implemented on an FPGA with low quiescent power, such as the Actel ProASIC, the result may be different. On such an FPGA, the area may play a greater role in the EAT computation.

Also of interest is that for designs with 8 PEs and 4 PEs, the nonalternating mode design has a lower EAT while for designs with 2 PEs or 1 PE, the alternating mode has a lower EAT. In all cases, the nonalternating mode has lower energy and latency. But, for the 1 and 2 PE cases, the size of the memory in each of the PEs in nonalternating mode increases. This increase in memory size reduces the comparative energy savings of nonalternating mode and increases the area. As a result, the alternating mode designs perform better in terms of EAT.

VII. PERFORMANCE COMPARISON

A. Experimental Setup

We now compare the high-level estimates for the latency, energy dissipation, and area of our design with the latency, area, and energy dissipation of the highly optimized Xilinx IP Core for the 2-D DCT. For each design, we assume 8-bit precision and a 100 MHz clock frequency. For our design, we use the equations and tables from Sections V and VI.

For the Xilinx IP Core, we implemented and simulated the design to obtain results for comparison. First, we used the Xilinx CoreGenerator (unless stated otherwise, all tools are from Xilinx ISE 4.2i). We then synthesized and placed and routed the design using Xilinx XST and PAR, respectively. The result was a .ncd file. The .ncd file was converted to back-annotated VHDL and simulated using Mentor Graphics ModelSim 5.6b. The testbench input to the simulation consisted of randomly generated data from a uniform distribution. The input had a switching activity of 50%. The simulation outputs a .vcd file. The .vcd and .ncd files were input to Xilinx XPower where the power dissipated by the design was estimated.

B. Results

For comparison, we again use the 8×8 DCT. Since the 8 PE version of our design is the most efficient, we choose that one for comparison with the Xilinx design.

Figures 4 and 5 show the latency and energy, respectively, for our design and the Xilinx IP Core. In each figure, we make two comparisons. We first compare the performance for one 8×8 DCT. In this case, our design has a lower latency and dissipates less energy than the Xilinx design. Our design gives a modest speed-up of 1.1 times but lowers energy dissipation by an estimated *320 nJ*.

The second comparison is of the average latency per DCT and average energy per DCT for each design in computing ten 8×8 DCTs in a row. Notice that the latency and energy

TABLE III

ENERGY, AREA, LATENCY, AND EAT

	Nonalternating Mode				Alternating Mode			
No. PEs	E (nJ)	A (slices)	T (us)	EAT	E (nJ)	A (slices)	T (us)	EAT
8	370.51	352	1.52	1.00	370.51	352	1.52	1.00
4	548.54	176	2.72	1.32	572.54	176	2.88	1.46
2	947.65	112	5.24	2.81	980.16	88	5.60	2.44
1	1718.68	56	10.34	5.02	1797.18	44	11.04	4.40

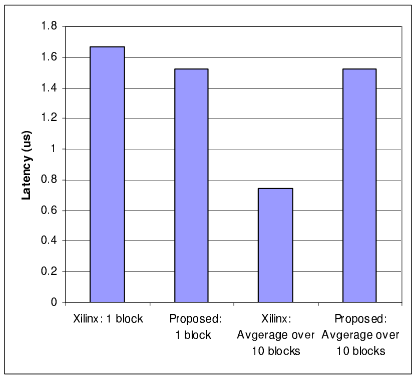

Fig. 4. Latency and Average Latency

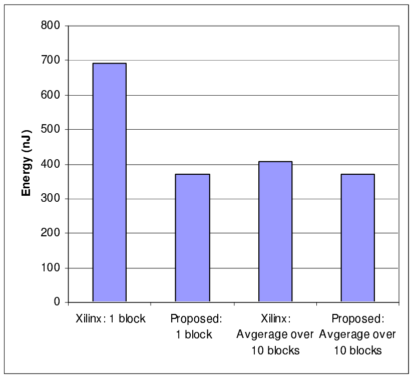

Fig. 5. Energy and Average Energy

dissipation of the Xilinx design both drop dramatically. In fact, the latency of the Xilinx design drops below that of our design, though ours still dissipates less energy. The reason the Xilinx IP Core's values drop is that the IP Core is a pipelined design. So, the initial costs are amortized as it computes successive DCTs. Due to the pipelining, the IP Core has a throughput of one 8×8 DCT per 64 clock cycles. Notice that our design's values do not change from the single DCT case. This lack of change is due to the fact that our design is not pipelined for multiple DCTs. Thus, it takes the same amount of time and energy to compute each DCT of 10 in a row as it would to compute them each individually.

Using Equation 8, we estimate that the area of our design is 352 slices. The place and route report for the Xilinx design tells us that the area of that design is 1001 slices. Our design is a clear improvement in terms of area.

VIII. CONCLUSION AND FUTURE WORK

Using domain-specific modeling and energy-efficient design techniques, we have developed an algorithm and architecture for the 2-D DCT using FPGAs. Our estimates show that our design is more energy-efficient than the highly optimized Xilinx IP core.

There are many directions for future work. We are developing a high-throughput version of our design. In this design, the computation of Phase 2 of DCT i is overlapped with Phase 1 of DCT $i+1$. The computation can be overlapped by using the cycles in which the adders and multiplier are idle in the current design. This design appears promising, but we must ensure that the added resources and resource utilization do not drastically increase its power dissipation and correspondingly decrease its energy efficiency.

Another future direction is to study the effects of precision on the design. We can study both accuracy of the results and the effect of precision on energy dissipation.

ACKNOWLEDGEMENTS

The authors wish to thank Seonil Choi for his invaluable assistance in the energy-efficient design process.

This work is supported in part by the DARPA Adaptive Computing Systems Program under contract DABT63-96-C-0049 monitored by Fort Hauchuca and in part by the National Science Foundation under award No. 99000613. Ju-wook Jang's work has been supported by Sogang University Research Grants in 2003.

REFERENCES

[1] C. Dick, "Minimum multiplicative complexity implementation of the 2-D DCT using Xilinx FPGAs," in *Proceedings of SPIE's Photonics East*, November 1998.
[2] S. Choi, R. Scrofano, V. K. Prasanna, and J.-W. Jang, "Energy-efficient signal processing using FPGAs," in *Proceedings of the Eleventh ACM International Symposium on Field-Programmable Gate Arrays*, 2003.
[3] S. Choi, J.-W. Jang, S. Mohanty, and V. K. Prasanna, "Domain-specific modeling for rapid system-wide energy estimation of reconfigurable architectures," in *Engineering of Reconfigurable Systems and Algorithms*, 2002.
[4] H. Lim, V. Piuri, and E. E. Swartzlander, Jr., "A serial-parallel architecture for two-dimensional discrete cosine and inverse discrete cosine transforms," *IEEE Transactions on Computers*, vol. 49, no. 12, pp. 1297–1309, December 2000.
[5] D. Slawecki and W. Li, "DCT/IDCT processor design for high data rate image and coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 2, no. 2, pp. 135–146, June 1992.
[6] L. Shang, A. Kaviani, and K. Bathala, "Dynamic power consumption in Virtex-II FPGA family," in *International Symposium on Field Programmable Gate Arrays*, 2002.
[7] A. Raghunathan, N. K. Jha, and S. Dey, *High-level Power Analysis and Optimization.* Kluwer Academic Publishers, 1998.
[8] L. Mintzer, *The Role of Distributed Arithmetic to Digital Signal Processing in FPGAs*, Xilinx, Corp.
[9] Xilinx Inc., http://www.xilinx.com.
[10] Altera Corp., http://www.altera.com.