Exploiting Data Level Parallelism For Energy Efficient Implementation of LDPC Decoders and DCT on a FPGA

XIAOHENG CHEN and VENKATESH AKELLA
University of California, Davis

We explore the use of data level parallelism (DLP) as a way of improving the energy efficiency and power consumption involved in running applications on an FPGA. We show that static power consumption is a significant fraction of the overall power consumption in an FPGA and that it does not change significantly even as the area required by an architecture increases, because of the dominance of interconnect in an FPGA. We show that the degree of DLP can be used in conjunction with frequency scaling to reduce the overall power consumption.

Categories and Subject Descriptors: B.2.1 [Arithmetic and Logic Structures]: Design Styles—Parallel; C.4 [Computer Systems Organization]: Performance of Systems

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: FPGA, Power, LDPC Codes, DCT

1. INTRODUCTION

Increasingly, FPGAs are being used in a variety of embedded systems, such as base stations, flash-based storage devices, configurable radios in space and military applications, and are also beginning to appear in high volume battery-operated devices, such as Netbooks, with the advent of FPGAs from new companies such as Silicon Blue and new families of FPGAs such as Igloo (from Actel). In many of these emerging applications, FPGA power consumption is becoming an important concern. Clearly, as the density and clock frequency increase, dynamic power grows. But leakage (static) power is becoming a significant concern in FPGAs. For example, using the Xilinx power estimator (version 12.2) [Xilinx 2010], one can see that, for the low power XCV6VLX240-1 (a low end Virtex 6 FPGA), the static power, even before loading a design, is 1.383 Watts. The reason is as follows. Unlike an ASIC or an application-specific standard product (ASSP), FPGAs are designed to maximize flexibility through configurable interconnects, which require a large number of programmable switches, strong drivers to drive long (global/semi-global) interconnects, and memory to hold the configuration. It is estimated that almost 80% of the FPGA die is devoted to supporting the interconnect [Lin et al. 2006]. Whether a particular embedded application uses all the interconnect resources, a majority of the transistors in the interconnect leak. Unlike an ASIC designer, an FPGA application developer cannot shut down the unused transistors with Vdd gating or use aggressive clock gating to minimize the total power consumption.

We consider the problem of designing energy efficient solutions for embedded communications and signal processing applications, which are important market segments for FPGAs. Embedded applications are characterized by a throughput and an area constraint. For example, media and networking applications are dic-
tated by data rate, which establishes a throughput constraint for a module. Moreover, there are always other components in an embedded system, such as controllers, interfaces and memory, etc., which compete with the area and power budget for the given module. However, given the reconfigurability of the FPGA, the designer can choose the appropriate architecture for a given throughput, area and/or power constraint. We examine the following three typical scenarios for an FPGA developer who is targeting an application in an embedded system:

Scenario 1: Maximize throughput while minimizing power and area.

Scenario 2: Maximize throughput while reducing power given an area budget (constraint). Emulation is an example of this scenario, in which given a specific FPGA board, the goal is to maximize the emulation speed, while staying within the thermal limits of the FPGA board.

Scenario 3: Achieve a throughput target while reducing power and minimizing area. This is the problem faced by a designer of the discrete cosine transform (DCT) block in a FPGA based realization of an H.264 system, for example.

We propose the use of data level parallelism (DLP) as an additional knob in the architectural design space for the scenarios listed above. Generally the degree of data level parallelism is modeled as a parameter $K$, with $K = 1$ meaning that the architecture is scalar whereas $K = 4$ meaning four data items are processed simultaneously. For example, in the case of a low density parity check codes (LDPC) decoder, which is used as a case study in this paper, $K = 1$ implies one message at a time is read, processed, and written back. With $K = 4$, four messages are simultaneously read, processed and written back using four sets of functional units. So, in vector processing parlance, $K$ can be viewed as the number of vector lanes.

Exploiting DLP makes sense for an energy efficient FPGA based design for three reasons. First, the embedded memory blocks (called block RAMs in Xilinx) are wide (typically 72 bits), and multiple block RAMs can be cascaded to realize even wider words. Thus, we can store a short vector in one memory word, and access it without increasing the interconnect requirements or power consumption significantly. Also, we can emulate a fast vector memory with very little overhead, which is essential for exploiting DLP. Second, vector processing requires data alignment to meet the specific requirements of the underlying algorithm. In an FPGA, one can create a custom alignment unit, which will further improve performance and power consumption [Oliver and Akella 2003]. Finally, the reconfigurability of an SRAM based FPGA allows us to select the optimal degree of vectorization in conjunction with the clock frequency to meet the area and throughput constraints.

In this paper we will show the way that an architect can use DLP as an additional architectural knob in conjunction with frequency scaling to meet a given throughput constraint or the throughput and area constraints, which addresses Scenario 1 and Scenario 2 listed above. We propose a vector processing approach to exploiting DLP for these scenarios, which is illustrated with a detailed case study of the implementation of an LDPC decoder. LDPC decoding is an important emerging application in many of the embedded systems used in 802.11n, 802.16e, DVB-S2,
and flash-based storage systems [Tai 2010]. Also FPGAs are used to design LDPC codes for a specific application by emulating the bit-error performance, especially when performance at very low error rates is required. We will address the issues with Scenario 3 with DCT as an example. DCT is an important kernel in widely used video and image compression standards such as H.264, MPEG-4, JPEG, etc..

The paper is organized as follows. In Section 2 we summarize the work related to this paper in recent literature. In Section 3 we propose energy-efficient design methodology for LDPC decoders. In Section 4 we use the proposed methodology to explore the design space of DCT processors. Section 5 summarizes the key results, and we conclude in Section 6 with suggestions for future work.

2. RELATED WORK

There is a plethora of work in the area of energy and power optimization for FPGA based implementation. We organize them into the following five categories. First, general energy efficiency optimization techniques are proposed for FPGA based systems by numerous researchers. Algorithm-level techniques, such as pipelining and parallel processing, are presented to improve energy efficiency for FPGA-based design [Choi et al. 2003; Jang et al. 2005]. Also, power efficient DSP datapath configuration methodology is claimed to enhance the total power efficiency [McKeown et al. 2008]. Run-time partial reconfiguration technique is applied to reduce power [Liu et al. 2010].

Second, the growing importance of leakage in FPGAs has spawned research into techniques for reducing leakage power in FPGA based designs. Anderson and Najm present two approaches without any area or delay penalty for active leakage reduction [Anderson and Najm 2006]. The first approach is to interchange logic signals in an FPGA design towards their complemented form to spend the majority of time in low leakage states. The second scheme is to alter the routing steps of the FPGA CAD flow are to encourage more frequent use of routing resources that have low leakage power consumption. Also, Lodi et al. propose a new FPGA circuit implementation to reduce leakage based on multi-threshold and self reverse biasing techniques [Lodi et al. 2005]. Gayasen et al. divide the the FPGA fabric into small regions, and switches on/off the power supply to each region using a sleep transistor in order to conserve leakage energy [Gayasen et al. 2004].

Third, although static power dissipation has received much attention recently due to its sharp increase, dynamic power still dominates the power consumption of an FPGA. Thus, optimization methods are investigated to reduce the dynamic power for FPGA based design. Shang et al. study the dynamic power consumption in the Virtex-II FPGA family by identifying important resources in the FPGA architecture and determining their utilization [Shang et al. 2002]. Lamoureux et al. propose to reduce the dynamic power in FPGAs through edge alignment and glitch filtering [Lamoureux et al. 2008].

Fourth, high-level FPGA power models, power-aware architectures, and CAD tools are also studied for faster power estimation [Chen et al. 2007; Gupta et al. 2007]. Li et al. present a flexible FPGA architecture evaluation platform, which incorporates the switch-level models for interconnects and macro models for LUTs [Li et al. 2003]. Also, power-aware RAM mapping algorithm is proposed for FPGA
embedded memory blocks [Russell et al. 2006]. Again, binary decision diagrams are applied to build a power-aware synthesis tool [Tinmaung et al. 2007].

Fifth, device and circuits technology advances offer potential opportunities for power optimization in future FPGA architectures. Device and architecture are co-optimized for FPGA power reduction [Cheng et al. 2007]. In addition, device and circuit level techniques are explored for creating an FPGA that can be used in battery-powered applications [Tuan et al. 2006]. Monolithically stacked 3D-FPGA [Lin et al. 2006] and configurable dual-Vdd approach [Li et al. 2004; Lin et al. 2005] are claimed to have power advantages. Wang et al. propose two scheme to advance the power efficiency [Wang et al. 2009]. The first scheme is a placement-based technique, which reduces interconnect resource usage on the clock network. Another scheme applies the clock gating technique to reduce toggling on the clock interconnect for lower power.

Although the architecture and implementation for LDPC decoders [Sun et al. 2006; Yang et al. 2006; Wang and Cui 2007; Zhang and Parhi 2004; Chen et al. 2009; Chen et al. 2009; Bates et al. 2006] and DCT [Megalingam et al. 2009; Huang et al. 2009] are studied widely, we observe lack of research investigating DLP and its impact on power consumption and throughput of FPGA-based implementations for these two applications. We attempts to address this gap, and apply DLP to two efficient architecture: one is the partially parallel decoder [Chen and Parhi 2004], and another is the Xilinx two-dimensional (2D) DCT design [Pillai 2002].

3. DATA PARALLEL LDPC DECODERS

In this section, we firstly provide the background knowledge of LDPC codes and their decoding algorithms. Then we show how vector processing can be used to exploit DLP in the LDPC decoding algorithm. After that, we show the design space of implementing LDPC decoders with different DLP degrees on two different FPGAs—a Spartan family FPGA that is more geared towards low cost and power and a Virtex family FPGA that is geared towards high performance.

3.1 LDPC Codes and Decoding Algorithms

LDPC codes were first discovered by Gallager in 1962 [Gallager 1962], and were rediscovered to have Shannon capacity approaching error correction capability in the late 1990s. LDPC codes have been adopted as the forward error correction method for many emerging applications, such as high-density flash memory, satellite broadcasting (DVB-S2), WiFi (802.11n), and mobile WiMAX (802.16e). An LDPC code $C$ of length $n$ is given by the null space of an $J \times n$ sparse parity-check matrix $H = [h_{i,j}]$ over GF(2). If each column has constant weight $\gamma$ (the number of 1-entries in a column) and each row has constant weight $\rho$ (the number of 1-entries in a row), then $C$ is referred to as a $(\gamma, \rho)$-regular LDPC code. If the columns and/or rows of the parity-check matrix $H$ have multiple weights, then the null space of $H$ gives an irregular LDPC code. A binary $n$-tuple $v = (v_0, v_1, \ldots, v_{n-1})$ is a codeword in $C$ if and only if $vH^T = 0$. An LDPC code is also represented graphically by a bipartite graph (also known as a Tanner graph) which consists of two disjoint sets of nodes. Nodes in one set represent the code bits and are called variable nodes (VNs), and the nodes in the other set represent the check-sums that the code bits must satisfy and are called check nodes (CNs). There are $n$ VNs and
\( J \) CNs in a Tanner graph. The \( i \)-th CN is connected to the \( j \)-th VN by an edge if and only if \( h_{i,j} = 1 \).

Iterative message passing algorithms, such as the normalized min-sum algorithm (NMSA) [Fossorier et al. 1999], are widely used to decode LDPC codes. Let \( L = (L_0, L_1, \ldots, L_{n-1}) \) be the input soft-decision sequence. For \( 0 \leq i < J \) and \( 0 \leq j < n \), we define \( N_i = \{ j : 0 \leq j < n, h_{i,j} = 1 \} \), and \( J_j = \{ i : 0 \leq i < J, h_{i,j} = 1 \} \). Let \( K_{\text{max}} \) be the maximum number of iterations to be performed. For \( 0 \leq k \leq K_{\text{max}} \), let \( \mathbf{z}^{(k)} = (z_0^{(k)}, z_1^{(k)}, \ldots, z_{n-1}^{(k)}) \) be the hard decision vector generated in the \( k \)-th decoding iteration, \( L_{i \rightarrow j}^{(k)} \) be extrinsic message passed from the \( i \)-th CN to the \( j \)-th VN, \( L_{j \rightarrow i}^{(k)} \) be the extrinsic message passed from the \( j \)-th VN to the \( i \)-th CN, and \( L_j^{(k)} \) be the reliability value of the \( j \)-th code bit. The NMSA can be formulated as follows:

**Initialization:** Set \( k = 0 \), \( \mathbf{z}^{(0)} = \mathbf{z} \) and the maximum number of iterations to \( K_{\text{max}} \). For all \( j \), set \( L_j^{(0)} = L_j \), set \( L_{j \rightarrow i}^{(0)} = L_j \) when \( h_{i,j} = 1 \).

Step 1) Parity check: Compute the syndrome \( \mathbf{z}^{(k)} \mathbf{H}^T \) of \( \mathbf{z}^{(k)} \). If \( \mathbf{z}^{(k)} \mathbf{H}^T = \mathbf{0} \), stop decoding and output \( \mathbf{z}^{(k)} \) as the decoded codeword; otherwise go to Step 2.

Step 2) If \( k = K_{\text{max}} \), stop decoding and declare a decoding failure; otherwise, go to Step 3.

Step 3) CNs update: Compute the message
\[
L_{i \rightarrow j}^{(k)} = \alpha \left( \prod_{j' \in N_i \setminus j} \text{sign}(L_{i \rightarrow j'}^{(k)})) \left( \min_{j' \in N_i \setminus j} |L_{i \rightarrow j'}^{(k)}| \right),
\]
where \( 0 < \alpha < 1 \) is the normalization factor. The optimal value of \( \alpha \) is chosen based on software simulation results for best decoding performance. Pass messages from CNs to VNs.

Step 4) VNs update: \( k \leftarrow k + 1 \). Compute the message
\[
L_{j \rightarrow i}^{(k)} = L_j + \sum_{i' \in J_j \setminus j} L_{i' \rightarrow j}^{(k-1)},
\]
and update the reliability of each received bit by
\[
L_j^{(k)} = L_j + \sum_{i' \in J_j} L_{i' \rightarrow j}^{(k-1)}.
\]
For \( 0 \leq j < n \), make the following hard-decision: 1) \( z_j^{(k)} = 0 \), if \( L_j^{(k)} \geq 0 \); 2) \( z_j^{(k)} = 1 \), if \( L_j^{(k)} < 0 \). Form a new received vector \( \mathbf{z}^{(k)} = (z_0^{(k)}, z_1^{(k)}, \ldots, z_{n-1}^{(k)}) \). Go to Step 1.

As Step 1) and Step 3) can be performed in parallel, they are often merged into one processing units for decoder implementation, which is named CN processing units (CNU). Step 4) is mapped to VN processing units (VNU).

### 3.2 Data Parallel Decoder for QC LDPC Codes

The most advantageous structure of an LDPC code is the quasi-cyclic (QC) structure in terms of encoding [Li et al. 2005] and decoding implementation [Sun et al. 2006; Yang et al. 2006; Wang and Cui 2007; Zhang and Parhi 2004; Chen et al. 2009; Chen et al. 2009; Bates et al. 2006]. The parity-check matrix \( \mathbf{H} \) of a QC-LDPC...
code is a $\gamma \times \rho$ array (or block) of circulants or circulant permutation matrices (CPMs) and/or zero matrices of the same size, say $m \times m$, of the following form:

$$
H = \begin{bmatrix}
A_{0,0} & A_{0,1} & \cdots & A_{0,\rho-1} \\
A_{1,0} & A_{1,1} & \cdots & A_{1,\rho-1} \\
\vdots & \vdots & \ddots & \vdots \\
A_{\gamma-1,0} & A_{\gamma-1,1} & \cdots & A_{\gamma-1,\rho-1}
\end{bmatrix}
$$

(4)

Then $H$ is a $\gamma m \times \rho m$ matrix over GF(2). The QC-LDPC code given by the null space of the $H$ matrix has length $\rho m$ and rate at least $1 - (\gamma/\rho)$.

![Figure 1](image)

Fig. 1. There are 18 circulant permutation matrices (or CPMs) in the parity check matrix for a (3,6)-regular QC-LDPC code, which are marked 1, 2, 4, 8, 16, 32, 6, 12, 24, 48, 96, 160. The number denotes the offset for the CPM. For example, the circulant marked labeled 3 is shown in more detail in (b). Each circulant is a $256 \times 256$ matrix. The offset is the position of the non-zero entry in the first row of the circulant.

Figure 1(a) shows the code structure of a (3, 6) regular QC-LDPC codes, which is denoted as $C_1$. The $H$ matrix has 3 block rows and 6 block columns for a total of 18 CPMs. Each CPM is $256 \times 256$ with a certain offset, which denotes the position of the non-zero entry in the first row of the matrix. Details of a CPM with offset equal to 3 is shown in Figure 1(b).

The partially-parallel decoder proposed by Chen and Parhi [Chen and Parhi 2004] groups the VNUs and CNUs by their CPM locality, reduces the routing complexity between memory and processing units, and thus is widely used for practical decoder implementation. We extend the architecture by packing more than one messages in the memory [Chen et al.]. For the NMSA, the intrinsic and the extrinsic messages are usually 6-8 bit wide, thus up to six 6-bit messages can be packed in one memory word for a $512 \times 36$ block RAM configuration. We define the number of messages packed into one memory word as $K$. For $(\gamma, \rho)$-regular QC-LDPC code, this approach uses $K\gamma \rho$-input CNUs, $K\rho \gamma$-input VNUs, $\rho$ intrinsic message memories (IMEM) with each storing $m$ intrinsic messages, and $\gamma \rho$ extrinsic message memories (EMEM) with each storing $m$ extrinsic messages and $m$ hard decision bits. Let $I_{ij}(0 \leq j < \rho)$ denote the IMEM of the $j$-th block column, which stores the received intrinsic message. Let $E_{ij}(0 \leq i < \gamma, 0 \leq j < \rho)$ denote the EMEM that stores the messages passed between the $i$-th CN update and VNU update. Figure 2 shows a decoder for the code $C_1$ when $K = 2$. 

In a data parallel decoder, each block RAM location holds multiple messages. Memory access conflicts could arise if the CNU and VNU try to write the same location simultaneously. Such conflicts are resolved by two methods. First, we use low-cost alignment units, which do not increase the complexity of the decoder and do not limit the scalability. Second, we use double buffering, i.e., the messages are replicated for CNU and VNU accessing, so that they are stored in different ways to match the access patterns of the CNU and VNU processing. Though it doubles the amount of required memory, it does not increase the number of block RAMs necessary, because we use the same block RAM to store both CNU and VNU memory. This approach works because typically the CPM sizes are much smaller than the depth of the block RAMs in an FPGA. The details for related alignment units and emulation of the existing two ports of block RAM to four ports is shown in [Chen et al.].

3.3 Results and Discussion

We have developed a tool that takes the generic architectural template of a partially parallel LDPC decoder, and automatically produces synthesizable Verilog code for data parallel LDPC decoders with different values of $K$. We implement the designs with two different FPGAs—a high performance Virtex 4 FPGA (XC4VLX160-FF1148) and a low cost Spartan 3 FPGA (XC3S4000-FG1156). We also present the results for two different LDPC codes. The decoders are implemented with Xilinx ISE 10.1 and simulated with Modelsim 6.5b. The area and timing results are reported after placement and routing. Power results are obtained from the Xilinx power estimator.

First, we show the results for the code $C_1$. We use an 8-bit quantization scheme and the NMSA ($\alpha = 1$). For CNU and VNU implementation, we adopted the
Table I. LDPC decoder implementation results for $C_1$ using XC4VLX160 with average iteration number as 3 and clock rate as 200 MHz.

<table>
<thead>
<tr>
<th>$K$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slices</td>
<td>1988</td>
<td>3381</td>
<td>6569</td>
<td>8424</td>
</tr>
<tr>
<td>4 input LUTs</td>
<td>2298</td>
<td>4189</td>
<td>7614</td>
<td>9410</td>
</tr>
<tr>
<td>Flip Flops</td>
<td>2164</td>
<td>4249</td>
<td>8519</td>
<td>9431</td>
</tr>
<tr>
<td>Block RAMs</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>Throughput (Gbps)</td>
<td>0.283</td>
<td>0.560</td>
<td>0.830</td>
<td>1.109</td>
</tr>
<tr>
<td>Dynamic Power (W)</td>
<td>0.416</td>
<td>0.536</td>
<td>0.750</td>
<td>0.809</td>
</tr>
<tr>
<td>Clock (W)</td>
<td>0.099</td>
<td>0.154</td>
<td>0.259</td>
<td>0.281</td>
</tr>
<tr>
<td>Logic (W)</td>
<td>0.057</td>
<td>0.108</td>
<td>0.204</td>
<td>0.241</td>
</tr>
<tr>
<td>BRAM (W)</td>
<td>0.260</td>
<td>0.274</td>
<td>0.287</td>
<td>0.287</td>
</tr>
<tr>
<td>Static Power (W)</td>
<td>1.055</td>
<td>1.059</td>
<td>1.065</td>
<td>1.068</td>
</tr>
<tr>
<td>Total Power (W)</td>
<td>1.472</td>
<td>2.185</td>
<td>2.184</td>
<td>2.182</td>
</tr>
<tr>
<td>Energy/Sample (nJ)</td>
<td>5.201</td>
<td>5.284</td>
<td>5.284</td>
<td>5.284</td>
</tr>
</tbody>
</table>

Table II. LDPC decoder implementation results for $C_1$ using XC3S4000 with average iteration number as 3 and clock rate as 100 MHz.

<table>
<thead>
<tr>
<th>$K$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slices</td>
<td>2048</td>
<td>3514</td>
<td>6837</td>
<td>8677</td>
</tr>
<tr>
<td>4 input LUTs</td>
<td>2408</td>
<td>4518</td>
<td>7997</td>
<td>9721</td>
</tr>
<tr>
<td>Flip Flops</td>
<td>2216</td>
<td>4249</td>
<td>8519</td>
<td>9426</td>
</tr>
<tr>
<td>Block RAMs</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>Throughput (Gbps)</td>
<td>0.142</td>
<td>0.280</td>
<td>0.415</td>
<td>0.555</td>
</tr>
<tr>
<td>Dynamic Power (W)</td>
<td>0.260</td>
<td>0.353</td>
<td>0.485</td>
<td>0.51</td>
</tr>
<tr>
<td>Clock (W)</td>
<td>0.102</td>
<td>0.141</td>
<td>0.194</td>
<td>0.203</td>
</tr>
<tr>
<td>Logic (W)</td>
<td>0.026</td>
<td>0.050</td>
<td>0.093</td>
<td>0.109</td>
</tr>
<tr>
<td>BRAM (W)</td>
<td>0.132</td>
<td>0.149</td>
<td>0.198</td>
<td>0.198</td>
</tr>
<tr>
<td>Static Power (W)</td>
<td>0.278</td>
<td>0.280</td>
<td>0.284</td>
<td>0.285</td>
</tr>
<tr>
<td>Total Power (W)</td>
<td>0.538</td>
<td>0.62</td>
<td>0.769</td>
<td>0.795</td>
</tr>
<tr>
<td>Energy/Sample (nJ)</td>
<td>3.789</td>
<td>2.214</td>
<td>1.853</td>
<td>1.432</td>
</tr>
</tbody>
</table>

Table III. LDPC decoder implementation results for $C_1$ using XC4VLX160 with average number of iterations is 3 and the clock frequency is 200 Mhz. Block RAMs are not being shared in this experiment.

<table>
<thead>
<tr>
<th>$K$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slices</td>
<td>1988</td>
<td>3381</td>
<td>6569</td>
<td>8424</td>
</tr>
<tr>
<td>4 input LUTs</td>
<td>2298</td>
<td>4189</td>
<td>7614</td>
<td>9410</td>
</tr>
<tr>
<td>Flip Flops</td>
<td>2164</td>
<td>4249</td>
<td>8519</td>
<td>9431</td>
</tr>
<tr>
<td>Block RAMs</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>Throughput (Gbps)</td>
<td>0.283</td>
<td>0.560</td>
<td>0.830</td>
<td>1.109</td>
</tr>
<tr>
<td>Dynamic Power (W)</td>
<td>0.416</td>
<td>0.782</td>
<td>1.243</td>
<td>1.562</td>
</tr>
<tr>
<td>Clock (W)</td>
<td>0.099</td>
<td>0.154</td>
<td>0.259</td>
<td>0.281</td>
</tr>
<tr>
<td>Logic (W)</td>
<td>0.057</td>
<td>0.108</td>
<td>0.204</td>
<td>0.241</td>
</tr>
<tr>
<td>BRAM (W)</td>
<td>0.260</td>
<td>0.520</td>
<td>0.780</td>
<td>1.04</td>
</tr>
<tr>
<td>Static Power (W)</td>
<td>1.055</td>
<td>1.059</td>
<td>1.065</td>
<td>1.068</td>
</tr>
<tr>
<td>Total Power (W)</td>
<td>1.472</td>
<td>1.841</td>
<td>2.308</td>
<td>2.630</td>
</tr>
<tr>
<td>Energy/Sample (nJ)</td>
<td>5.201</td>
<td>3.287</td>
<td>2.757</td>
<td>2.372</td>
</tr>
</tbody>
</table>
Table IV. LDPC decoder implementation results for the (8176,7156) code using XC4VLX160 with average iteration number as 5 and clock rate as 200 MHz.

<table>
<thead>
<tr>
<th>K</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slices</td>
<td>4021</td>
<td>9085</td>
<td>14769</td>
<td>17857</td>
</tr>
<tr>
<td>4 input LUTs</td>
<td>7385</td>
<td>13720</td>
<td>21943</td>
<td>27046</td>
</tr>
<tr>
<td>Flip Flops</td>
<td>5907</td>
<td>13911</td>
<td>21786</td>
<td>27210</td>
</tr>
<tr>
<td>Block RAMs</td>
<td>80</td>
<td>80</td>
<td>80</td>
<td>80</td>
</tr>
<tr>
<td>Throughput (Gbps)</td>
<td>0.445</td>
<td>0.882</td>
<td>1.317</td>
<td>1.756</td>
</tr>
<tr>
<td>Dynamic Power (W)</td>
<td>1.238</td>
<td>1.65</td>
<td>2.07</td>
<td>2.312</td>
</tr>
<tr>
<td>Clock (W)</td>
<td>0.196</td>
<td>0.383</td>
<td>0.552</td>
<td>0.661</td>
</tr>
<tr>
<td>Logic (W)</td>
<td>0.174</td>
<td>0.353</td>
<td>0.560</td>
<td>0.693</td>
</tr>
<tr>
<td>BRAM (W)</td>
<td>0.868</td>
<td>0.914</td>
<td>0.958</td>
<td>0.958</td>
</tr>
<tr>
<td>Static Power (W)</td>
<td>1.080</td>
<td>1.094</td>
<td>1.108</td>
<td>1.117</td>
</tr>
<tr>
<td>Total Power (W)</td>
<td>2.318</td>
<td>2.744</td>
<td>3.176</td>
<td>3.429</td>
</tr>
<tr>
<td>Energy/Sample (nJ)</td>
<td>5.209</td>
<td>3.111</td>
<td>2.412</td>
<td>1.953</td>
</tr>
</tbody>
</table>

As $K$ increases from 1 to 4, the throughput grows from 283 Mbps to 1.109 Gbps (3.92 times of the $K = 1$ case), the total power steps up from 1472 mW to 1887 mW (1.3 times of the $K = 1$ case), and the energy efficiency (defined as energy/sample) improves by a factor of 3, decreasing from 5.201 nJ/sample to 1.692 nJ/sample. The static power and the block RAM power remains almost the same when $K$ changes, since DLP does not increase the number of block RAMs or the interconnect requirement significantly. The observation verifies the key insight of the paper: as FPGAs already start with a significant overhead, DLP can be exploited without significant additional costs. Table II shows the results for implementation of the same code on a Spartan FPGA; the energy efficiency improves by a factor of 2.6, decreasing from 3.789 nJ/sample to 1.432 nJ/sample. This improvement is less than that of the Virtex FPGA based design, since the static power for the Spartan FPGA is a smaller portion of the total power than the Virtex FPGA.

Next, we conduct an experiment to determine the sensitivity of these results with respect to the block RAM power only. For example, if an FPGA has smaller memory blocks, can data parallelism still help? Table III shows the results from this experiment. Note that the number of block RAMs increases in proportion to $K$, i.e., block RAMs are not shared. Clearly, even without block RAM sharing, an improvement in energy efficiency is observed, because the dynamic power grows sublinearly with $K$. In an architecture with finer grained embedded memory blocks, the improvement in energy efficiency will be more pronounced, because of the relative decrease in the percentage of memory power in the total power.

To verify that these results are not dependent on the specific code being used, we evaluate the proposed methodology on a large code ((8176,7156) (4,32)-regular QC-LDPC code [Chen et al. 2004]) used in a real embedded system, namely NASA’s...
Table V. LDPC decoder implementation results for (8176,7156) using XC3S4000 with average iteration number as 5 and clock rate as 100 MHz.

<table>
<thead>
<tr>
<th>K</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slices</td>
<td>5688</td>
<td>13914</td>
<td>22572</td>
<td>26894</td>
</tr>
<tr>
<td>4 input LUTs</td>
<td>7799</td>
<td>12829</td>
<td>20947</td>
<td>24740</td>
</tr>
<tr>
<td>Flip Flops</td>
<td>5867</td>
<td>13901</td>
<td>21536</td>
<td>27195</td>
</tr>
<tr>
<td>Block RAMs</td>
<td>80</td>
<td>80</td>
<td>80</td>
<td>80</td>
</tr>
<tr>
<td>Throughput (Gbps)</td>
<td>0.223</td>
<td>0.441</td>
<td>0.658</td>
<td>0.878</td>
</tr>
<tr>
<td>Dynamic Power (W)</td>
<td>0.682</td>
<td>0.895</td>
<td>1.209</td>
<td>1.304</td>
</tr>
<tr>
<td>Clock (W)</td>
<td>0.165</td>
<td>0.246</td>
<td>0.308</td>
<td>0.350</td>
</tr>
<tr>
<td>Logic (W)</td>
<td>0.077</td>
<td>0.151</td>
<td>0.241</td>
<td>0.294</td>
</tr>
<tr>
<td>BRAM (W)</td>
<td>0.44</td>
<td>0.498</td>
<td>0.660</td>
<td>0.66</td>
</tr>
<tr>
<td>Static Power (W)</td>
<td>0.289</td>
<td>0.295</td>
<td>0.304</td>
<td>0.307</td>
</tr>
<tr>
<td>Total Power (W)</td>
<td>0.971</td>
<td>1.190</td>
<td>1.513</td>
<td>1.611</td>
</tr>
<tr>
<td>Energy/Sample (nJ)</td>
<td>4.35</td>
<td>2.7</td>
<td>2.3</td>
<td>1.835</td>
</tr>
</tbody>
</table>

Table VI. Power consumption and clock rate combination (W, MHz) to reach the required throughput for LDPC decoder for C1. For example, for target throughput of 0.2 Gbps and K = 4, (1.264,36) means the power consumption of 1.264 W when the design runs at 36 MHz.

<table>
<thead>
<tr>
<th>Throughput (Gbps)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>XC4VLX160</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.2</td>
<td>1.361,141</td>
<td>1.282,71</td>
<td>1.272,48</td>
<td>1.264,36</td>
</tr>
<tr>
<td>0.3</td>
<td></td>
<td>1.371,107</td>
<td>1.385,72</td>
<td>1.342,54</td>
</tr>
<tr>
<td>0.4</td>
<td></td>
<td>1.46,143</td>
<td>1.468,96</td>
<td>1.409,72</td>
</tr>
<tr>
<td>0.5</td>
<td></td>
<td>1.55,179</td>
<td>1.55,120</td>
<td>1.478,90</td>
</tr>
<tr>
<td>XC3S4000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td>0.478,70</td>
<td>0.444,35</td>
<td>0.458,24</td>
<td>0.439,18</td>
</tr>
<tr>
<td>0.15</td>
<td></td>
<td>0.493,54</td>
<td>0.507,36</td>
<td>0.478,27</td>
</tr>
<tr>
<td>0.2</td>
<td></td>
<td>0.542,71</td>
<td>0.556,48</td>
<td>0.518,36</td>
</tr>
<tr>
<td>0.25</td>
<td></td>
<td>0.591,89</td>
<td>0.606,60</td>
<td>0.556,45</td>
</tr>
</tbody>
</table>

LANDSAT (near-earth high-speed satellite communication). We use NMSA with $\alpha = 0.75$ and 6-bit quantization scheme for the design. The implementation results using XC4VLX160 and XC3S4000 are presented in Table IV and Table V, respectively. The design using $K = 4$ results in a 3.95X improvement in throughput over $K = 1$. The energy efficiency metric (energy/sample) for XC4VLX160 decreases from 5.209 nJ/sample to 1.953 nJ/sample, which is a 62% saving. The energy efficiency metric for XC3S4000 decreases from 4.35 nJ/sample to 1.835 nJ/sample, which is a 57% saving. The block RAM power consumption increases from 0.868 W ($K = 1$) to 0.958 W ($K = 4$) for XC4VLX160. Also, the block RAM power consumption increases from 0.44 W ($K = 1$) to 0.66 W ($K = 4$) for XC3S4000, which is a significant energy saving. The static power change is within 5% of the $K = 1$ case.

Next, we show DLP used in combination with the frequency scaling to reduce the total power consumption while meeting a throughput constraint, a typical requirement in embedded system design. For example, we can take a decoder with a higher degree of DLP (i.e., larger $K$) with a throughput of $B_1$ and a clock frequency of $f_1$, and reduce its throughput to the target throughput of $B_2 < B_1$,
Table VII. Power consumption and clock rate combination (W, MHz) to reach the required throughput for LDPC decoder for (8176,7156) code. For example, for target throughput of 0.4 Gbps and $K = 4$, (1.725,46) means the power consumption of 1.725 W when the design runs at 46 MHz.

<table>
<thead>
<tr>
<th>Throughput (Gbps)</th>
<th>$K=1$</th>
<th>$K=2$</th>
<th>$K=3$</th>
<th>$K=4$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>XC4VLX160</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td>2.199,180</td>
<td>1.881,91</td>
<td>1.804,61</td>
<td><strong>1.725,46</strong></td>
</tr>
<tr>
<td>0.5</td>
<td></td>
<td>2.057,113</td>
<td>1.941,76</td>
<td><strong>1.851,57</strong></td>
</tr>
<tr>
<td>0.6</td>
<td></td>
<td>2.239,136</td>
<td>2.1,91</td>
<td><strong>1.976,68</strong></td>
</tr>
<tr>
<td>0.7</td>
<td></td>
<td>2.422,159</td>
<td>2.249,106</td>
<td><strong>2.105,80</strong></td>
</tr>
<tr>
<td>0.8</td>
<td></td>
<td>2.598,181</td>
<td>2.403,121</td>
<td><strong>2.226,91</strong></td>
</tr>
<tr>
<td><strong>XC3S4000</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.2</td>
<td>0.909,90</td>
<td>0.75,45</td>
<td>0.73,30</td>
<td><strong>0.676,23</strong></td>
</tr>
<tr>
<td>0.3</td>
<td>0.933,68</td>
<td>0.904,46</td>
<td></td>
<td><strong>0.811,34</strong></td>
</tr>
<tr>
<td>0.4</td>
<td>1.062,91</td>
<td>1.074,61</td>
<td></td>
<td><strong>0.951,46</strong></td>
</tr>
</tbody>
</table>

by reducing the clock frequency to $f_2$, such that $f_2 = B_2 \times f_1/B_1$, to reduce the overall power consumption. Table VI and Table VII show the design space of this optimization for the $C_1$ code and the (8176,7156) code. Interestingly, for the wide range of throughput targets, using a decoder with $K = 4$, and then scaling down the frequency to the $1/4$ of the $K = 1$ case, is the best approach for reducing the overall power consumption. Based on this observation, we could trade off dynamic power for static power (which is quite high in FPGAs) in order to reduce the overall power consumption. For the (8176,7156) code, the power saving ranges between 10% (a throughput of 0.5 Gbps, XC4VLX160) and 21% (a throughput of 0.4 Gbps, XC4VLX160). The power saving for the (8176,7156) code is higher than that of the $C_1$ code, since the block RAM power consumption occupies a larger portion of the total power.

4. DATA PARALLEL TWO DIMENSIONAL DISCRETE COSINE TRANSFORM

We will continue our design space exploration with a different example to show the effectiveness of the DLP technique. We choose the two dimensional discrete cosine transform (2D DCT) as another example for two reasons. First, it is a very important kernel in almost all video and image compression standards. Second, we can vary $K$ to a much larger value, as opposed to just 4, as in the case of the LDPC decoder. (Note that we are restricted to $K = 4$ in the LDPC decoder example because, beyond that, the design would not fit in a single Virtex 4 LX160 device.)

We start with a brief overview of the 2D DCT algorithm and present results that show the impact of increasing $K$ on the static power and the throughput. We also illustrate the optimization of finding the lowest power solution by jointly optimizing $K$ and the clock frequency.

4.1 Two Dimensional DCT

Two dimensional DCT is a lossy compression scheme in which an $N \times N$ image block is transformed from the spatial domain to the frequency domain. DCT decomposes the signal into spatial frequency components called DCT coefficients. The low frequency DCT coefficients appear toward the upper left-hand corner of the DCT...
Fig. 3. The diagram for 8 × 8 DCT, the architecture is derived directly from a Xilinx reference design [Pillai 2002]. The 2D DCT module composes of two independent 1-D DCT submodules, which exchange messages via the transpose memory. For eliminating the idle time, double buffering technology is employed for the transpose memory. The independent 1-D DCT submodules are computing messages for two consecutive 8 × 8 blocks. S/P denotes the serial to parallel converter, every clock cycle a input message is appended to the serial to parallel converter, the eight-tuple of messages are hold valid for eight clock cycles, during the period, the messages are computed by PE (processing element, the detail is shown in (b)). The ADD denotes the adder. The Coeff. Mem stores the constant coefficients for multiplication. The Toggle Flop output a sequence of the form 010101… continuously, following the symmetry property of the coefficients.

Fig. 4. Data parallel DCT with parallelism factor $K = 4$

matrix, and the high frequency coefficients are in the lower right-hand corner. DCT is image independent and can be performed with fast parallel algorithms, which can be efficiently implemented with parallel architecture. DCT has been widely adopted by multiple standards, e.g., JPEG, MPEG, and H.264. For most image compression standards, $N = 8$. An 8 × 8 block size does not have significant memory requirements, and furthermore, a block size greater than 8 × 8 does not offer significantly better compression.

The algorithm used for the computation of the 2D DCT is based on the following
where the equation may be written in the matrix form of

\[ Y_{uv} = \frac{2}{\sqrt{N}} C_u C_v \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} X_{ij} \cos \left[ \frac{(2i + 1)u\pi}{2N} \right] \cos \left[ \frac{(2j + 1)v\pi}{2N} \right], \quad (5) \]

where \( C_u = C_v = \sqrt{2} \) for \( u, v = 0 \), and \( C_u = C_v = 1 \) otherwise. Alternatively, the equation may be written in the matrix form of \( Y = CX^T \), where \( C \) is the cosine coefficients and \( C^T \) are the transpose coefficients. This equation can also be written as \( Y = CZ \), where \( Z = X^T \). As the coefficient (organized by \( C = [c_{i,j}] \)) has the property that \( c_{i,j} = -c_{N-i-1,j} \), half of the storage amount for coefficients and multiplication is used to compute the messages.

The intermediate results produced from the first one-dimensional transform are stored in the transpose memory. The transpose memory is a dual-port RAM storing an entire 8 × 8 block, which is the result of the row decomposition in the first stage. While the transpose memory is written in the row-major order, the second stage of processing reads data from the transpose memory in the column-major order, effectively performing a transposition of the intermediate results. The design of an 8 × 8 DCT is shown in Figure 3(a).

For the baseline design, one message is read into and generated by the processing core per clock cycle. For a vector processor, \( K \) independent 8 × 8 blocks are processed simultaneously, as shown in Figure 4. The coefficient memory is shared among \( K \) processing cores; the transpose memory is widened by \( K \) times. There are no alignment units. Other parts in the design, e.g., processing elements and the serial to parallel converter, are replicated \( K \) times.

### 4.2 Results and Discussion

We use the XC4VLX160 to implement the DCT (varying from \( K = 1 \) to \( K = 12 \)), as shown in Table VIII. Table IX shows the results for a Spartan FPGA (varying from \( K = 1 \) to \( K = 11 \)). (We stop at \( K = 11 \) for the Spartan FPGA because Spartan has fewer resources than a Virtex FPGA.) The number of bits for input messages, the intermediate message stored in transpose memory, and the output
messages are 8, 11, and 12, respectively. First we show the advantages of DLP by comparing our implementation with Xilinx's own implementation [Pillai 2002]. Our implementation with $K = 12$ demonstrates a 15X improvement in energy efficiency (0.723 nJ/sample versus Xilinx's 11.12 nJ/sample). Note that even with $K = 1$, which is scalar, our implementation outperform the Xilinx design because of the use of block RAMs as opposed to distributed memory in the case of Xilinx's implementation.

For the $K$-parallel design, $K$ 8-bit inputs and $K$ 12-bit outputs are used. On average, the bandwidth of the block RAM supports up to three intermediate messages per access. Thus, $\lceil K/3 \rceil$ block RAMs are used. The throughput of the DCT scales with $K$. As some resources are shared among DCT steams, e.g., the coefficient memory, the toggle flop, and the control signal generator, the area increases sub-linearly. For instance, for XC4VLX160, when $K$ increases from 1 to 12, the throughput increases by 12X, but the area increases by only 10.2X. As far as energy efficiency goes, for the Virtex implementation, as $K$ goes from 1 to 12, the energy per sample changes from 5.755 nJ to 0.723 nJ for a 8X improvement. As expected,


---

### Table IX. 8 × 8 DCT results on a XC3S4000, 100 MHz

<table>
<thead>
<tr>
<th>Throughput (Gsps)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slices</td>
<td>760</td>
<td>815</td>
<td>1632</td>
<td>2259</td>
<td>3125</td>
<td>3588</td>
<td>4482</td>
<td>5103</td>
<td>5562</td>
<td>6284</td>
<td>7447</td>
<td>8085</td>
</tr>
<tr>
<td>4-input LUTs</td>
<td>1206</td>
<td>862</td>
<td>1651</td>
<td>2445</td>
<td>3236</td>
<td>4027</td>
<td>4815</td>
<td>5606</td>
<td>6392</td>
<td>7182</td>
<td>7963</td>
<td>8752</td>
</tr>
<tr>
<td>Flip Flops</td>
<td>830</td>
<td>926</td>
<td>1787</td>
<td>2647</td>
<td>3128</td>
<td>3588</td>
<td>5240</td>
<td>6101</td>
<td>6936</td>
<td>7838</td>
<td>8666</td>
<td>9526</td>
</tr>
<tr>
<td>Block RAMs</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Multipliers</td>
<td>8</td>
<td>8</td>
<td>16</td>
<td>24</td>
<td>32</td>
<td>40</td>
<td>48</td>
<td>56</td>
<td>64</td>
<td>72</td>
<td>80</td>
<td>88</td>
</tr>
<tr>
<td>Dynamic Power (W)</td>
<td>0.072</td>
<td>0.093</td>
<td>0.152</td>
<td>0.205</td>
<td>0.238</td>
<td>0.274</td>
<td>0.329</td>
<td>0.372</td>
<td>0.409</td>
<td>0.448</td>
<td>0.487</td>
<td>0.523</td>
</tr>
<tr>
<td>Clock (W)</td>
<td>0.049</td>
<td>0.059</td>
<td>0.091</td>
<td>0.112</td>
<td>0.122</td>
<td>0.130</td>
<td>0.155</td>
<td>0.166</td>
<td>0.177</td>
<td>0.187</td>
<td>0.196</td>
<td>0.205</td>
</tr>
<tr>
<td>Logic (W)</td>
<td>0.009</td>
<td>0.010</td>
<td>0.019</td>
<td>0.029</td>
<td>0.036</td>
<td>0.043</td>
<td>0.057</td>
<td>0.066</td>
<td>0.075</td>
<td>0.085</td>
<td>0.094</td>
<td>0.103</td>
</tr>
<tr>
<td>BRAM (W)</td>
<td>0</td>
<td>0.006</td>
<td>0.008</td>
<td>0.008</td>
<td>0.013</td>
<td>0.017</td>
<td>0.017</td>
<td>0.022</td>
<td>0.023</td>
<td>0.025</td>
<td>0.030</td>
<td>0.031</td>
</tr>
<tr>
<td>MULT (W)</td>
<td>0.013</td>
<td>0.017</td>
<td>0.033</td>
<td>0.050</td>
<td>0.067</td>
<td>0.084</td>
<td>0.100</td>
<td>0.117</td>
<td>0.134</td>
<td>0.151</td>
<td>0.167</td>
<td>0.184</td>
</tr>
<tr>
<td>Static Power (W)</td>
<td>0.274</td>
<td>0.274</td>
<td>0.276</td>
<td>0.272</td>
<td>0.278</td>
<td>0.280</td>
<td>0.281</td>
<td>0.282</td>
<td>0.284</td>
<td>0.285</td>
<td>0.286</td>
<td>0.285</td>
</tr>
<tr>
<td>Total Power (W)</td>
<td>0.346</td>
<td>0.367</td>
<td>0.428</td>
<td>0.477</td>
<td>0.516</td>
<td>0.554</td>
<td>0.616</td>
<td>0.654</td>
<td>0.693</td>
<td>0.733</td>
<td>0.773</td>
<td>0.808</td>
</tr>
<tr>
<td>Energy/Sample (nJ)</td>
<td>4.325</td>
<td>3.67</td>
<td>2.14</td>
<td>1.59</td>
<td>1.29</td>
<td>1.108</td>
<td>1.017</td>
<td>0.934</td>
<td>0.866</td>
<td>0.814</td>
<td>0.773</td>
<td>0.735</td>
</tr>
</tbody>
</table>

### Table X. 8 × 8 DCT: Power Consumption (W) to reach the required throughput, XC4VLX160. For simplicity, the frequency is not listed.

<table>
<thead>
<tr>
<th>Throughput (Gsps)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>1.151</td>
<td>1.143</td>
<td>1.145</td>
<td>1.151</td>
<td>1.155</td>
<td>1.16</td>
<td>1.166</td>
<td>1.17</td>
<td>1.175</td>
<td>1.181</td>
<td>1.184</td>
<td>1.191</td>
</tr>
<tr>
<td>0.4</td>
<td>1.206</td>
<td>1.202</td>
<td>1.209</td>
<td>1.211</td>
<td>1.214</td>
<td>1.220</td>
<td>1.224</td>
<td>1.228</td>
<td>1.234</td>
<td>1.236</td>
<td>1.24</td>
<td></td>
</tr>
<tr>
<td>0.6</td>
<td>1.259</td>
<td>1.266</td>
<td>1.267</td>
<td>1.268</td>
<td>1.275</td>
<td>1.278</td>
<td>1.281</td>
<td>1.287</td>
<td>1.292</td>
<td>1.294</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td>1.324</td>
<td>1.322</td>
<td>1.322</td>
<td>1.339</td>
<td>1.331</td>
<td>1.334</td>
<td>1.341</td>
<td>1.344</td>
<td>1.346</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.0</td>
<td>1.375</td>
<td>1.384</td>
<td>1.384</td>
<td>1.385</td>
<td>1.393</td>
<td>1.395</td>
<td>1.397</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.2</td>
<td>1.438</td>
<td>1.438</td>
<td>1.439</td>
<td>1.438</td>
<td>1.447</td>
<td>1.448</td>
<td>1.45</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.4</td>
<td>1.492</td>
<td>1.492</td>
<td>1.492</td>
<td>1.492</td>
<td>1.501</td>
<td>1.5</td>
<td>1.504</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.6</td>
<td>1.546</td>
<td>1.546</td>
<td>1.546</td>
<td>1.546</td>
<td>1.554</td>
<td>1.554</td>
<td>1.554</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.8</td>
<td>1.599</td>
<td>1.607</td>
<td>1.608</td>
<td>1.607</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0</td>
<td>1.661</td>
<td>1.661</td>
<td>1.661</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.2</td>
<td>1.711</td>
<td>1.709</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

the benefits for Spartan are slightly lower, because the static power is a smaller fraction of the total power in a Spartan FPGA.

Next we explore the impact of frequency scaling in conjunction with increased DLP, as shown in Table X. The results are interesting because the trend is different for the DCT, compared to the LDPC decoder. Recall that, for the LDPC decoder, it is beneficial to go with the highest value of $K$ and then reduce the frequency to meet the target constraint in order to reduce the overall power consumption. The case of the DCT shows that that higher degree of DLP might not be the best strategy. Choosing a smaller value of $K$ seems to yield the best overall power consumption while meeting a given throughput constraint. (The bold entries in Table X show the instance with the lowest power consumption.) Also, the choice of $K$ varies with the target throughput in a unapparent way. The reason for this discrepancy is the relatively smaller contribution of the clock power and block RAM power to the overall power consumption. In the case of the DCT, the block RAM power is almost negligible, so clock power becomes a determining factor in the overall power consumption, and clock power is dictated by the clock frequency and not $K$.

5. SUMMARY OF THE RESULTS

A careful reader will have noticed that the improvement in energy efficiency with DLP is accompanied by an increase in area. So two questions emerge: 1) What do the trade-off in area versus throughput and energy efficiency look like? 2)
How does DLP impact that design space? To investigate this, we plot the energy efficiency and the area requirement as a function of the degree of DLP (namely, $K$) in Figure 5 for the LDPC decoder and in Figure 6 for the DCT example. The trends are remarkably consistent: as $K$ increases from 1 to 2, the energy per sample drops significantly and the area increases modestly. As $K$ increases beyond a threshold, the improvement in energy efficiency is almost negligible, though the area increases significantly. Clearly, some DLP is good, but too much DLP is not always good, especially when there is an area constraint, as in most embedded systems.

Finally, we consider the following question—Will DLP always improve the energy efficiency of an FPGA based design? Are there other factors in play which can explain why DLP yields good results on the LDPC and DCT applications? Let us consider the total power $P_{\text{total}} = P_{\text{fix}} + P_{\text{var}}$, where $P_{\text{var}}$ is a function of the degree of DLP ($K$) and clock frequency and $P_{\text{fix}}$ represents the power that does not change as $K$ increases beyond 1. Any application for which $P_{\text{fix}}$ is large will benefit from the proposed approach, because $P_{\text{fix}}$ is not just static power; it also includes the large amount of interconnect power and the power consumed by any resources that are shared when $K$ increases—such as block RAMs, coefficient ROMs, address generation logic, etc. Alternately, if $P_{\text{fix}}$ is small, or if $P_{\text{var}}$ were to increase significantly with an increase in $K$, the proposed technique will not yield much improvement in energy efficiency. In the case of both DCT and LDPC decoder, the so called cost of exploiting DLP is very low (because of simple memory access patterns without any need for complex data alignment and synchronization)—and $P_{\text{fix}}$ is high, so these applications yields great improvements.
6. CONCLUSIONS

We explore the use of data level parallelism (DLP) to improve the energy efficiency and power consumption of applications on an FPGA. We show that static power consumption is a significant fraction of the overall power consumption in an FPGA and it does not change significantly even as area increases. The reason is the high fixed costs in an FPGA based system (the dominance of interconnect). We show that the degree of DLP (called $K$ in this paper) can be used in conjunction with frequency scaling to reduce the overall power consumption, but the trade-offs between block RAM power and clock power can dictate the optimal value of $K$. Finally, we show that, for area conscious designs, a small value of $K$ results in the best return on investment in terms of energy efficiency and throughput.

Acknowledgement

We are grateful to Prof. Shu Lin for help with the code construction and other suggestions. We are also grateful to Palma Lower for her assistance in proofreading of the paper.

REFERENCES


