Energy-Efficient Signal Processing Using FPGAs

Seonil Choi, Ronald Scrofano, and Viktor K. Prasanna
Department of Electrical Engineering-Systems
University of Southern California
Los Angeles, USA
{seonilch, rscrofan, prasanna}@usc.edu

Ju-Wook Jang
Department of Electronic Engineering
Sogang University
Seoul, Korea
jjang@sogang.ac.kr

ABSTRACT

In this paper, we present techniques for energy-efficient design at the algorithm level using FPGAs. We then use these techniques to create energy-efficient designs for two signal processing kernel applications: fast Fourier transform (FFT) and matrix multiplication. We evaluate the performance, in terms of both latency and energy efficiency, of FPGAs in performing these tasks. Using a Xilinx Virtex-II as the target FPGA, we compare the performance of our designs to those from the Xilinx library as well as to conventional algorithms run on the PowerPC core embedded in the Virtex-II Pro and the Texas Instruments TMS320C6415. Our evaluations are done both through estimation based on energy and latency equations and through low-level simulation. For FFT, our designs dissipated an average of 60% less energy than the design from the Xilinx library and 56% less than the DSP. Our designs showed a factor of 10 improvement over the embedded processor. These results provide concrete evidence to substantiate the idea that FPGAs can outperform DSPs and embedded processors in signal processing. Further, they show that FPGAs can achieve this performance while still dissipating less energy than the other two types of devices.

Categories and Subject Descriptors

C.3 [**Special Purpose and Application Based Systems**]: Real-time and embedded systems; C.1.3 [**Processor Architectures**]: Other architecture styles—*adaptable architectures*

General Terms

Algorithms, Performance, Design

Keywords

Energy efficient design techniques, FPGA, FFT, matrix multiplicaiton, performance estimation

FPGA'03, February 23–25, 2003, Monterey, California, USA.

1. INTRODUCTION

FPGAs have become an attractive option for implementing signal processing applications because of their high processing power and customizability. The inclusion of new features in the FPGA fabric, such as a large number of embedded multipliers, adds to this attractiveness. FPGAs can now be considered for computationally demanding applications such as those in signal processing. Traditionally, the performance metrics for signal processing and indeed, most processing in general, have been latency and throughput. However, with the proliferation of portable, mobile devices, it has become increasingly important that systems are not only fast, but that they are also energy efficient. One such example is software-defined radio (SDR) [6]. Due to SDR's adaptivity and high computational requirement, an FPGA-based system is very viable solution.

There are currently no commercially available FPGAs that have both millions of gates and low-power features. Thus, instead of low-level optimization techniques, in this paper, we investigate and apply algorithmic techniques for minimizing the energy dissipated by FPGAs in signal processing applications. Our techniques can also be used for a next generation FPGA that has lower power dissipation feature as well as high computing power. We apply the techniques presented here to the design of architectures and algorithms for two well-known signal processing kernel applications: fast Fourier transform (FFT) and matrix multiplication. The FFT is the compute-intensive portion of broadband beamforming applications such as those generally used in SDR and sensor networks. Matrix multiplication is also a frequently used kernel operation in signal and image processing systems including mobile and SDR systems. We compare the latencies and energy dissipations of the energy-efficient designs to those of Xilinx IPcore, a DSP, and an embedded processor for the same signal processing kernel applications. We use both high-level estimation (based on latency and energy equations) and low-level simulation in our comparisons. These comparisons show that our proposed designs using FPGAs can provide significant reductions in not only latency but also energy dissipation.

The remainder of this paper is organized as follows. Section 2 characterizes the sources of energy dissipation in FPGAs and presents algorithmic techniques for minimizing this energy dissipation. The application of these techniques to two signal processing kernel applications is presented in Section 3. Section 4 explains our method for estimating the latency and energy dissipation for the algorithms and architectures presented in Section 3. It also details our method for comparing FPGA implementations to DSPs and embedded processors. Section 4.3 shows our high-level performance estimates. Section 5 presents the results of low-level simulations of selected implementations of the kernel applications. Finally, Section 6 presents areas for future work.

2. ENERGY-EFFICIENT DESIGN TECHNIQUES

We discuss techniques that can be applied to FPGA-based designs to obtain energy efficiency. Note that, though the terms are often used interchangeably, power and energy are not the same. Energy is the product of average power dissipation and latency. To understand energy dissipation, therefore, it is necessary to understand power dissipation and its effect on latency and vice versa.

In this section, we first briefly describe sources of power dissipation in FPGAs. We then present techniques for reducing energy dissipation, some of which do so by lowering power dissipation, others by lowering latency. Several low-level and algorithm-level techniques for energy-efficient design are discussed. The main focus is algorithm-level techniques. "Algorithm-level techniques" refers to those techniques in algorithm development that can be used to reduce energy dissipation.

2.1 Energy Dissipation in FPGAs

Several studies of FPGA power dissipation have recently appeared in literature [13, 18]. These works show that power dissipation in FPGA devices is due primarily to the programmable interconnects. In the Xilinx Virtex-II [17] family, for example, it is reported that between 50% and 70% of total power is dissipated in the interconnect, with the remainder being dissipated in the clocking, logic, and I/O blocks. This analysis differs from ASIC technology where clock distribution often dominates power dissipation [18]. The sources of power dissipation between these two technologies are different because their interconnect structures are composed disparately: FPGA interconnect consists of pre-fabricated wire segments of various lengths, with used and unused routing switches attached to each wire segment.

Another important factor affecting the power dissipation in FPGAs is resource utilization [13]. In typical FPGA designs, a majority of the resources are not used after the configuration and thus they will not dissipate any dynamic power. One more factor in determining power dissipation is the switching activity, which is defined as the number of signal transitions in a clock period. The switching activity for each resource depends not only on the type of design but also the input stimuli.

Understanding sources of power dissipation, we can now discuss energy-efficient low-level and algorithm-level design techniques for FPGA-based design.

2.2 Low-Level Design Techniques

In literature, there are many low-level power management techniques that lead to energy savings when applied to designing for FPGAs [13, 18]. Here, we discuss those low-level techniques that provide control knobs for algorithm-level design.

One such technique is clock gating, which is used to disable parts of the device that are not in use during the computation. In the Virtex-II family of FPGAs, clock gating can be realized by using primitives such as BUFGMUX to switch from a high frequency clock to a low frequency clock [17]. BUFGCE can be used for dynamically driving a clock tree only when the corresponding logic is used.

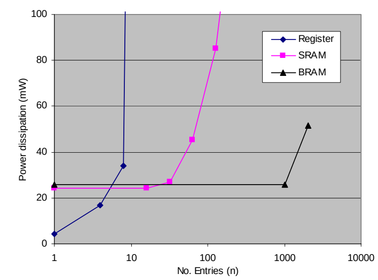


Figure 1: Power dissipation of various storage element implementations as the function of the number of entries (Virtex-II XC2V1500, 150MHz, 50% switching activity, 16 bits per entry)

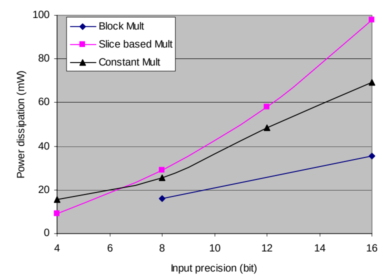


Figure 2: Power dissipation of various multipliers as the function of precision (Virtex-II XC2V1500, 150MHz, 50% switching activity)

Choosing energy-efficient *bindings* is another technique. A binding is a mapping of an operation to an FPGA component. The ability to choose the proper binding is due to the existence of several configurations for the same computation. Thus, different bindings affect FPGA energy dissipation. For example, in Figure 1, we show three possible bindings for storage in Virtex-II FPGAs based on the number of entries: registers, slice based RAM (SRAM), and embedded Block RAM (BRAM). For large storage elements (those with more than 48 entries) BRAM shows an advantage in power dissipation over other implementations. Another example is the choice between hard and soft IP. One such case is the choice of multipliers: block multipliers, such as those in the Xilinx Virtex-II and Altera Stratix, can be more efficient than CLB-based multipliers (See Figure 2). All results for Figure 1, Figure 2, and Figure 3 are obtained using the techniques proposed in [3]. In developing an algorithm, a designer can analyze the trade-offs that arise from various bindings based on the design requirements.

2.3 Algorithm-Level Design Techniques

It is known that energy performance can be improved significantly by optimizing a design at the algorithm level [11]. We summarize the algorithm-level techniques that can be used to improve the energy performance of designs implemented on FPGAs.

Architecture Selection: Since FPGAs provide the freedom to map various architectures, choosing the appropriate architecture affects the energy dissipation. It plays a large part in determining the amount of interconnect and logic to be used in the design. Since interconnect dissipates a large amount of power, minimizing the number of long wires between building blocks is beneficial [17]. Several past efforts have identified various architecture families, each having different characteristics in terms of I/O complexity, memory requirements, area, etc. [5, 9]. Based on the performance needs and the limitations of the target FPGA chip, it is possible to identify a suitable architecture. Identification of an appropriate architecture for an algorithm ensures that we begin with an efficient design most suitable for the performance requirements and that there are various architecture parameters that can be varied to explore trade-offs among energy, latency, and area. For example, matrix multiplication can be implemented using a 1-D array (linear array) or a 2-D array. A 2-D array dissipates more power from interconnect since more interconnects are required. Thus, it is possible that more energy would be dissipated, depending upon the resulting latency.

Module Disabling: In developing an algorithm, it is possible to design the algorithm such that it utilizes the clock gating technique to disable modules that are not in use during the computation. For example, FFT computation has many complex number multipliers to perform twiddle factor computations (multiplication and addition/subtraction). Because of the nature of the algorithm, some twiddle factors are 1, -1, j, or $-j$ and their computation can be bypassed. Thus, the implementation of twiddle factor computation can exploit clock gating to disable the unnecessary computation modules. Another example is that some RAM implementations have sleep states so that power dissipation is reduced when they are idle. Figure 3 shows the power dissipation of SRAM (16 bits per entry) of various sizes. The disabled memory dissipates less than 10% of the amount of power that the enabled memory does. The power dissipation of BRAM is even smaller than that of SRAM. Its power dissipation is less than 1% of the power dissipation of enabled memory. Because of these reduced power dissipations, energy dissipation is also reduced, provided that latency does not increase too much.

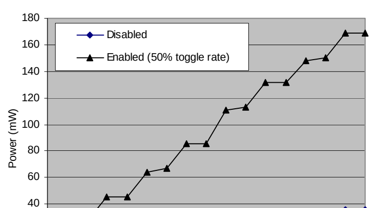


Figure 3: The effect on energy dissipation of the disabling of a SRAM as a function of the number of entries (Virtex-II XC2V1500, 150Mhz, 50% memory access rate, 8 bits per entry)

Pipelining: Pipelining is an efficient design practice for both time and energy performance. Many digital signal processing applications process streaming data. For these applications with regular data flow, pipelining increases throughput. Pipelining increases power dissipation, however, since all logic in the design is continuously active. In FPGA designs with streaming data, throughput is another important factor in energy dissipation. Thus, in the pipelined design, a modified version of the energy equation is $E_{pipe} = P_{avg}/Th$, where Th is the throughput of the design. Note that $\frac{1}{Th}$ can be considered the *effective latency* of the design. The effective latency accounts for the benefits of overlapping computations in pipelining. All designs in this paper adopt pipelining. Pipelining is one technique in which increasing the power dissipation may decrease the overall energy dissipation.

Parallel Processing: Parallel processing is an important technique for reducing energy dissipation in FPGA systems. In practice, the trade-off between pipelining and parallelism is not distinct: merely replicating functional units rather than using pipelining has the negative effect of increasing area and wiring, which in turn increases the energy dissipation. Instead, a more sophisticated approach to parallel processing is needed. In Section 5, we will discuss this effect in detail (See Figure 10).

Algorithm Selection: A given application is mapped onto FPGAs differently by selecting different algorithms. For example, using block matrix multiplication is the algorithm-level design choice for larger matrix multiplication. In Figure 10, the block matrix multiplication is energy-efficient choice for $n > 24$. In implementing the FFT, the choice of radices affects the energy performance. For example, a radix-4 based algorithm significantly reduces the number of complex multiplications that would otherwise be needed if a radix-2 based algorithm were used. All these algorithm selections affect the architectures and the energy dissipation of a design. The trade-offs between different algorithms should be analyzed to achieve energy-efficient designs.

3. ENERGY-EFFICIENT DESIGN FOR SIGNAL PROCESSING APPLICATIONS

Using the algorithm-level techniques, we map two applications onto FPGAs: FFT and matrix multiplication. Each of these is an important kernel applications in signal processing. For each application, we describe our energy-efficient design, including those of the aforementioned design techniques that have been used.

3.1 Fast Fourier Transform

For FFT designs, we use the well known Cooley-Tukey method. The calculation of an N-point FFT requires $O(N)$ operations for each of its $\log_2(N)$ stages, so the total computation required is $O(N \log_2 N)$ [10].

Due to the fact that, in practice, FFTs often process streams of data, a pipelined architecture has been chosen. The N-point FFT design is based on the radix-4 algorithm. To generate different designs, we identify the parameters that determine the FFT architecture and eventually affect the energy dissipation. There are three design parameters that characterize an N-point FFT designs: 1) the problem size (N), 2) the degree of horizontal parallelism (H_p), and 3) the degree of vertical parallelism (V_p). The horizontal parallelism determines how many radix-4 stages are used in parallel ($1 \leq H_p \leq \log_4 N$). Vertical parallelism determines the number of inputs operated on in parallel. Using the radix-4 algorithm, it is possible to operate on up to 4 inputs in parallel. We have considered five basic modules in the architecture: radix-4 butterfly, data buffer, data path permutation, parallel-to-serial/serial-to-parallel mux, and twiddle factor computation modules. Each individual module is parameterized and, hence, scalable so that a complete design can be obtained from combinations of the basic modules (See Figure 4).

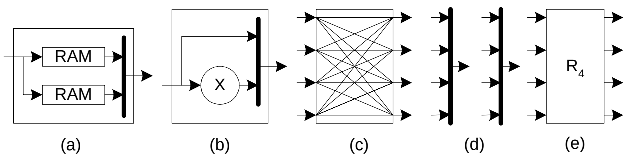


Figure 4: (a) Data buffer (Dbuf), (b) Twiddle factor computation (Twiddle), (c) Data path permutation, (d) parallel-to-serial/serial-to-parallel mux, and (e) Radix-4 computation ($\mathbf{R}_4$)

Radix-4 butterfly: This module performs a set of additions and subtractions with 16 adders/subtracters. It takes four inputs and produces four outputs in parallel. Each input data has real and imaginary values. The complex number multiplication for imaginary value j is removed since it is implemented by remapping the data path and using adders/subtracters.

Twiddle factor computation: This module performs the complex number multiplication of the data with twiddle factors. The twiddle factors are obtained from a sine/cosine lookup table. Bypassing the multiplication when the value of twiddle factors is 1, -1, j, or $-j$ can reduce computation and thus energy (by disabling the multipliers). This module contains 4 multipliers, 2 adders/subtracters and two sign inverters.

Data buffer: This module consists of two RAMs having N/V_p entries each. Data is written into one and read from the other RAM simultaneously. The read and write operations are switched after every N inputs. The data write and read addresses are at different strides determined by the architecture. For example in an N=16, single input case, writing is done sequentially and reading is at strides of four.

Data path permutation: In the parallel architecture ($V_p = 4$), data dependencies occur in accessing data in parallel from data buffers. This is due to stride accesses requiring data from either the same locations or the same RAMs. Thus before storing and after reading data in the data buffers of each stage, the data paths need to be permuted in a cyclic manner so that data can be accessed in parallel and in the correct order by the next stage. As an example, for $N = 16$, there are 16 data $(a_{[0]}, ..., a_{[15]})$ and 4 data are fed to four data buffers each cycle (See Figure 5 (b)). At the first cycle, $a_{[0]}, a_{[1]}, a_{[2]}, a_{[3]}$ are fed to the data buffer in parallel without permutation. At the second cycle, $a_{[4]}, a_{[5]}, a_{[6]}, a_{[7]}$ are fed with one permutation so that the data are stored such as $a_{[7]}, a_{[4]}, a_{[5]}, a_{[6]}$. Note that $a_{[4]}$ is stored in the second buffer not the first one. By doing these operations up to 4 cycles, the first buffer has $a_{[0]}, a_{[7]}, a_{[10]}, a_{[13]}$, the second buffer $a_{[1]}, a_{[4]}, a_{[11]}, a_{[14]}$, and so on. For the first radix-4 module, $a_{[0]}, a_{[4]}, a_{[8]}, a_{[12]}$ are read from each data buffer in parallel without any delay.

Parallel-to-serial/serial-to-parallel mux: This module is used when the data is fed into the Radix-4 module in parallel and fed out in serial for the serial or partially parallel architectures ($V_p < 4$). While the radix-4 module operates on four data in parallel, the rest of architecture is flexible. Thus, to match the data rate, a parallel-to-serial mux before the radix-4 module and a serial-to-parallel mux after the radix-4 module are required.

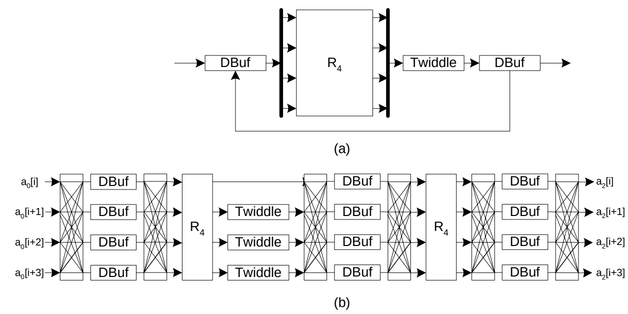


Figure 5: Architectures for 16-point FFT (a) $H_p = 1$, $V_p = 1$, **and (b)** $H_p = 2$, $V_p = 4$

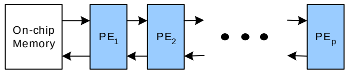
For example, a 16-point FFT algorithm has 2 radix-4 stages. In the design, we can use one or two radix-4 modules ($H_p = 1, 2$) depending on the sharing of the radix-4 module resource. If $H_p = 1$, one radix-4 module is used and is shared by the first and second stages. Thus a feedback datapath is necessary which decreases the throughput of the design (See Figure 5 (a)). Figure 5 (b) shows a fully parallel architecture for $N = 16$ when $V_p = 4, H_p = 2$. This design has 12 data buffers, two radix-4 modules, and 3 twiddle computation modules.

In the FFT designs, energy efficiency is achieved using parallel processing and pipelining along with module disabling. Pipelining and parallel processing increase the throughput and decrease the effective latency. Module disabling is used with the twiddle factor computation modules to reduce the required amount of energy by 50%. Other techniques used are 1) selecting a radix-4 algorithm as the basic module to reduce the number of complex multiplications in the design and 2) choosing bindings such that the data buffers and sine/cosine lookup tables are implemented using SRAM for $N < 64$ or BRAM for $N \geq 64$.

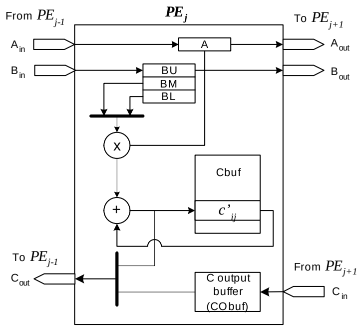
3.2 Matrix Multiplication

The matrix multiplication algorithm takes two $n \times n$ input matrices, A and B, and computes the product $C = A \times B$. An architecture for this kernel application is proposed in [8]. That architecture is optimized for low latency. When the matrix multiplication is performed on streams of data, throughput becomes the important performance metric. Our architecture, presented here, is *throughput-oriented* and utilizes parallelism and pipelining extensively. This architecture also includes logic for outputting the product matrix C while [8] lacks such logic. The output of one product matrix is overlapped with the computation of the next so no wait cycles are caused by the output. Thus, a throughput of one sample of data per clock cycle is achieved.

Figure 6 shows the architecture and the algorithm. A linear array architecture is employed. On-chip memory is used. That is, all matrices are stored in the on-chip memory of the



(a)



(b)

```
For t=1 to n do
For all j, 1≤j≤n, do in parallel
    PE_j shifts data in BU right to PE_j+1
    If (BU=b_kj), copy it into BM
For t=n+1 to n²+n do
For all j, 1≤j≤n, do in parallel
    PE_j shifts data in A, BU right to PE_j+1
    If (BU=b_kj), copy it into BL or BM (alternately)
    If (A=a_ik), c_ij' := c_ij' + a_ik x b_kj
        (b_kj is in either BM or BL, c_ij' is in Cbuf)
For t=n²+1 to 2n² do
For all j, 1≤j≤n, do in parallel
    PE_j store input C_in to CObuf
    for t=1 to n do
        PE_j output data c_ij' to PE_j-1
    for t=1 to n²-n do
        PE_j output data CObuf to PE_j-1
```

(c)

Figure 6: (a) Linear array architecture, (b) processing element, (c) algorithm for matrix multiplication

device (e.g. BRAM in the Virtex-II). The algorithm computes the product efficiently, both in terms of latency and energy, by cleverly moving the entries of the input matrices through the linear array. The entries from matrix A are fed into the linear array in column major order from the on-chip memory, while the entries from matrix B are fed into the linear array in row major order. Further, the entries from matrix A do not begin to enter the linear array until n cycles after the entries from matrix B. The PEs compute the sums of products that are the entries for matrix C. As an example, the algorithm is given in Figure 6 (c) for the case in which the number of PEs is equal to n.

In the algorithm, `A`, `BU`, `BM`, and `BL` are storage registers in a given PE; a_{ik} is the entry in the i-th row and k-th column of matrix A; and b_{kj} is the entry in the k-th row and the j-th column of matrix B. The output of PE_j is stored in PE_{j-1}. Starting from the (n^2+1)th cycle, n^2 results c'_{ij} are generated from each PE during the next $2n-1$ cycles. Output memory (`CObuf`) is necessary in each PE otherwise the final c'_{ij} from one matrix multiplication would be overwritten by the intermediate c'_{ij} of the next matrix. PE_{j-1} stores the output from PE_j via port `Cin` in `CObuf`. For n cycles, the final c'_{ij} at PE_j (for example, c_{12} at cycle 11) is output to PE_{j-1}. For the next n^2-n cycles, the stored c'_{ij} of PE_j in `CObuf` is outputted to PE_{j-1}. The outputs from PE_1 are stored in on-chip memory as a final results. The execution time becomes n^2 since there is one output result every clock cycle.

This linear array based design ensures that connections are only made between neighboring PEs and further ensures that only short interconnect resources are used. In a Virtex-II implementation of this design, most of the interconnect resources used are direct or double wires. All data from matrices A and B is fed in a pipelined fashion from left to right. All outputs flow from the right to the left. Each PE is always active which maximizes the throughput. Energy-efficient bindings are chosen for the multiplier, `CBuf`, and `CObuf`. If the number of entries is larger than 3, SRAM is used for energy efficiency. Block multipliers are used because they are energy-efficient, especially when both inputs are not constants. One more technique used in this design is block matrix multiplication. When $n < 24$, the fully parallel architecture that uses n PEs is a good candidate. When $n > 24$, block matrix multiplication with blocks of size (n/p) is used, where p is the number of PEs. This technique decreases throughput but saves area and energy.

It can be shown that this algorithm computes the product of two $n \times n$ matrices in $(n/p)^3(2p^2)$ cycles. However, pipelining can reduce the effective latency by a factor of 2.

4. HIGH-LEVEL PERFORMANCE AND EFFICIENCY COMPARISON

High-level estimation is a quick way for a designer to estimate performance of an application. Rather than implementation and simulation, the designer employs equations to provide reasonably accurate results that can be used to make decisions early in the design process (See Section 5 for a discussion of accuracy).

To compare FPGAs, DSPs, and embedded processors, we have picked a representative of each type of device and analyzed its performance in executing FFT and matrix multiplication. Our device selection is outlined below, followed by a description of our method for comparing these devices and the results of the comparisons.

4.1 Device Selection

The Virtex-II is a high-performance, platform FPGA from Xilinx [17]. We have chosen the XC2V1500/3000 devices for comparison. These FPGAs have 48 and 96 18×18-bit block multipliers, respectively, and run at a clock frequency of 100 MHz. The DSP that has been chosen for comparison is the Texas Instruments TMS320C6415. It has been chosen because it is Texas Instruments' highest performance fixed point DSP [16]. This device can perform four 16×16-bit or eight 8×8-bit multiplications per clock cycle. For an example embedded processor, we choose the IBM PowerPC 405 core embedded in the Xilinx Virtex-II Pro. This processor is a good example of the type of processor that can be found in a heterogenous, system-on-chip-type architecture. For signal processing tasks, the PowerPC core can perform one integer multiply and accumulate (MAC) operation per clock cycle.

4.2 Latency and Energy Estimation

Our first task is to estimate the latency required by each device in executing each of the kernel applications. Once we have this data, we can use it and information about the power dissipation of each device to calculate the energy dissipated in executing each kernel application. Each type of device requires a different method for performing the aforementioned two tasks.

4.2.1 FPGA

To estimate energy of FPGA based design at high level, we utilize the modeling technique proposed in [3]. Energy models specific to the designs are constructed at the module level by assuming that each module of a given type (register, multiplier, SRAM, BRAM, or I/O port) dissipates the same power independent of its location on the device. The high-level energy model has power functions associated with each module. To find each module's power function, each module was modeled in VHDL. Then, it was synthesized using Xilinx XST. After synthesis, the design was placed and routed. The output from the place and route tool was simulated using ModelSim 5.5e. The input stimuli for this simulation had an average switching activity of 50%. The output from the simulation and the output from the place and route tool were used as input to Xilinx XPower. XPower was used to estimate the power dissipation for each module. Curve fitting based on sample low-level simulations is used to determine the function. This function captures the effect on power dissipation of the variation of design parameters associated with the module such as precision, size, and frequency. Additional details of the model can be found in [3].

FFT: Based on the architecture and algorithm in Section 3.1, it can be shown that the equation to calculate the effective latency, of computing an N-point, radix-4 FFT is

$$L = \frac{N \log_4 N}{V_p \times H_p} \tag{1}$$

where L is in cycles. Here and throughout the paper, we convert the latency in cycles to latency in seconds by dividing the latency in cycles by the clock frequency. We also know the types of FPGA components (multipliers, registers, etc.) and the amounts of each type of component that are used by the five modules. We obtain the power function for each of the modules using the techniques in [3]. We sum the average power dissipation of each module to estimate the total power dissipation. Energy dissipation is equal to the product of total power and latency. To estimate the energy dissipation, then, we simply multiply the total power dissipation by the estimated latency. We use this method throughout the paper. The power functions for the data buffer, the radix-4 module, the data path permutation, parallel-to-serial/serial-to-parallel mux, and the twiddle computation modules are P_D, P_{R4}, P_{map}, $P_{s2p/p2s}$ and P_{Tw}, respectively, where $P_D = 1.23N + 35.44$ (mW) using SRAM, $P_D = 0.0156N + 79.65$ (mW) using BRAM, $P_{R4} = 142.84$ (mW), $P_{map} = 54.09$ (mW), $P_{s2p/p2s} = 13.52$ (mW), $P_{Tw} = 0.0054N + 183.57$ (mW) using BRAM, $P_{Tw} = 0.4879N + 157.74$ (mW) using SRAM, and $P_{IO} = 44$ (mW). Thus, the energy can be estimated as

$$\begin{aligned} E \;\; = \;\; & L \cdot \{V_p(H_p+1)P_D + 2mH_pP_{map} + (H_p-1)P_{R4} \\ & +2s(Hp-1)P_{s2p/p2s} + tP_{Tw} + 2V_pP_{IO}\} \end{aligned} \tag{2}$$

where m is the number of data path permutation modules ($m = 1$ when $V_p = 4$, otherwise $m = 0$) and s is the number of parallel-to-serial/serial-to-parallel muxes ($s = 1$ when $H_p = 1$, otherwise $s = 0$). t is the number of twiddle computation modules and is calculated as follows:

$$t = \begin{cases} (V_p-1)(H_p-1) & \text{when} \quad (H_p = \log_4 N, \quad V_p = 4) \\ V_p(H_p-1) & \text{when} \quad (H_p = \log_4 N, \quad V_p < 4) \\ (V_p-1)H_p & \text{when} \quad (H_p < \log_4 N, \quad V_p = 4) \\ V_pH_p & \text{when} \quad (H_p < \log_4 N, \quad V_p < 4) \end{cases}$$

We apply the parameter values to Equation 2 to compare the energy dissipations of various problem sizes. The clock speed of FPGA-based designs varies based on the likely speed achievable after place and route results. Figure 7 shows the distribution of energy dissipations of the modules for various design points when computing a 256-point FFT. The precision of each data is 16 bits. We assume a clock frequency of 100MHz since we later achieve that frequency in synthesized designs. We choose the minimal energy dissipation by selecting appropriate parameter values ($H_p = 4$ and $V_p = \log_4 N$). Parallelism increases the energy efficiency of the FFT design despite increasing the area requirement. These designs use BRAMs for data buffer and phase lookup table since BRAMs are more energy-efficient for $N > 64$ (See Figure 1).

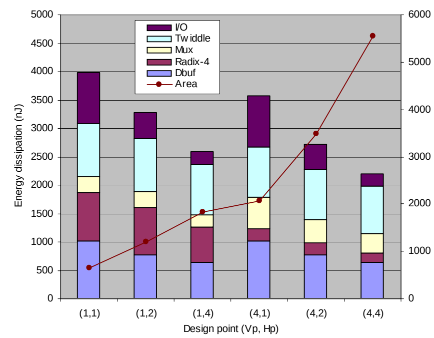


Figure 7: Energy distribution of modules in FFT architecture for various design points (N=256, BRAM based)

Matrix Multiplication: To calculate the latency and energy estimates for matrix multiplication, we follow the same strategy as for FFT. The effective latency, in cycles to multiply two $n \times n$ matrices for various values of n and different numbers of PEs (p) is

$$L = (n/p)^3 p^2 \tag{3}$$

The energy is given by

$$E = L \times (6pP_R + pP_{Acc} + pP_{Mult} + 2pP_{Buf} + 3P_{BRAM}) \tag{4}$$

where L is the latency for multiplying two $n \times n$ matrices using p processing elements, P_{Mult} is the power used by a multiplier, P_{Acc} is the power used by an accumulator, P_{BRAM} is the power used by an on-chip memory, P_R is the power used by a register, and P_{Buf} is the power used in `Cbuf` or `CObuf`. The values of the power variables are $P_{Mult} = 15.83$ (mW), $P_{BRAM} = 11.89$ (mW), $P_R = 2.12$(mW), $P_{ACC} = 5.54$ (mW), $P_{Buf} = 0.0048p^2 + 0.043p + 10.02$ (mW). The constant coefficients in Equation (4) are based on the number of each type of component used by a PE in the architecture of Section 3.2.

4.2.2 DSP

Texas Instruments provides optimized benchmark software algorithms for its TMS320C6415 DSP, among others [16]. In addition to providing the algorithms, Texas Instruments also provides equations for calculating the latency, in terms of number of cycles, for using the software benchmark algorithms. We utilize these equations in our estimations.

Table 1: Average power dissipation of the TMS320C6415

Clock frequency(MHz)	Operating voltage (V)	Power(W) 50%/50%	75%/25%
600	1.4	1.47	1.61
500	1.2	1.04	1.19

FFT: We use the latency equation associated with the FFT benchmark to estimate the latency for calculating an N-point FFT. The equation, which is for a radix-4, decimation-in-frequency decomposition is given below.

$$1.25N \log_4 N - 0.5N + 23 \log_4 N - 1 \quad (5)$$

Texas Instruments classifies two activity models for the DSP: the high activity model and the low activity model [16]. High activity represents the time the DSP spends performing compute-intensive, optimized algorithms, such as a FIR filter. Low activity represents the time spent performing less compute-intensive tasks, such as setting up registers or executing less optimized algorithms. Texas Instruments then presents data for two power activity categories: 75% high/25% low and 50% high/50% low. The former is for applications that spend 75% of their time in high activity and 25% of their time in low activity and the latter is for applications whose time is split into 50% high activity and 50% low activity. Table 1 shows the average power dissipation when the DSP is run at clock frequencies of 500 and 600 MHz. Note that these values do not include any external memory. We assume that all data can fit in the device's cache. We have determined that FFT more closely falls into the 75% high/25% low category because it is an optimized algorithm, spending more of its time in high activity.

Matrix Multiplication: Matrix multiplication is also one of the software benchmark algorithms. For $n \times n$ matrix multiplication, the latency equation is

$$L = \left[(2n + 18)\frac{n}{4} + 7\right]\frac{n}{2} + 5 \quad (6)$$

We use Equation 6 in our calculation of the latency required for matrix multiplication.

Matrix multiplication is a highly optimized benchmark, so its power dissipation likely falls into the 75% high/25% low activity category.

4.2.3 Embedded Processor

For each kernel application, we find a *lower bound* on the latency by considering the number of arithmetical operations that must be performed. The actual latencies and energies will likely be much higher than the lower bounds due to the control and memory instructions in the algorithms.

FFT: It is shown in [4] that the number of operations to compute an N-point, radix-4, decimation-in-time FFT is

$$\left(\frac{17}{4}\right) N \log_2 N - \frac{43}{6}N + \frac{32}{3} \quad (7)$$

The power dissipation of the PowerPC core is 0.9 mW/MHz [17]. At a clock frequency of 300 MHz, then, 270 mW of power are dissipated.

Matrix Multiplication: In the multiplication of two $n \times n$ matrices using the standard matrix multiplication algorithm, there are n^3 MAC operations. To find a lower bound, we assume that these MAC operations will dominate the computation time and we ignore other instructions. The PowerPC executes one MAC per clock cycle, so the lower bound of the latency is estimated to be

$$L = n^3 \quad (8)$$

4.3 High-Level Estimation Results

We apply the parameter values to the equations obtained in Section 4.2 to compare the energy dissipations of the three devices. The clock speed of FPGA-based designs varies based on the likely achievable speed after place and route results. The results presented for the TMS320C6415 are based on a clock rate of 500 MHz. The results presented for the PowerPC core on Virtex-II Pro are based on a clock rate of 300 MHz.

4.3.1 FFT

Figure 8 shows the energy dissipations of each device when computing an N-point FFT. The precision of each data is 16 bits. For FPGA-based designs, we assume a clock frequency of 100 MHz since we later achieve a 100 MHz design in low-level simulation. We choose the minimal energy dissipation by selecting different parameter values ($H_p = 4$ and $V_p = \log_4 N$). Despite only estimating a lower bound for the PowerPC, it is still the highest energy device for the FFT. It is reasonable to conclude, then, that the embedded processors are not good options for FFT if low energy is the highest priority. We see that the FPGA-based designs require the least amount of energy. For example, the FPGA-based design at $N = 1024$ has about 40% better energy performance compared with the DSP solution.

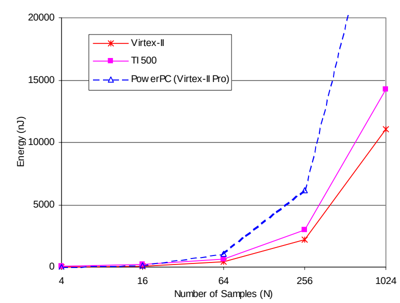


Figure 8: Energy vs. problem size for FFT

4.3.2 Matrix Multiplication

Figure 9 shows the energy dissipated by each device when multiplying two $n \times n$ matrices. For FPGA-based designs, we assume a clock frequency of 150 MHz since we later achieve 150 MHz in low-level simulation. The precision of each data is 8 bits. We use a fully parallel architecture when $n < 15$ and use the block matrix multiplication when $n > 24$. When $15 \leq n \leq 24$, the energy estimation values of fully parallel architecture and block based architecture are very close. Thus we can choose one depending on other constraints such as area and latency. When $n < 15$, $p = n$. That is, the number of multipliers equals the number of rows (or columns) in the matrices being multiplied. When $n > 24$, we choose the value of p where n/p is the block size and n is divisible by p. We have chosen here to set the number of processing elements to 16 because it leads to one of the most optimal configurations of the Virtex-II in terms of energy dissipation.

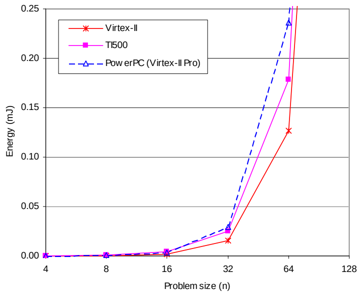


Figure 9: Energy vs. problem size for matrix multiplication

One note is that we have not studied the low-power modes of the TMS320C6415 or the PowerPC core.

5. LOW-LEVEL SIMULATION RESULTS

To test the accuracy of the high-level energy estimation in Section 4, we compared estimates from the functions against actual values based on synthesized or programmed designs and low-level simulation. We present low-level simulation results for FFT and matrix multiplication using the Virtex-II FPGA running at 100 and 150 MHz, Texas Instruments TMS320C6415 DSP running at 500 MHz, and PowerPC core running at 300 MHz. To compare our proposed designs on the same FPGA, we also synthesized FFT and matrix multiplication from Xilinx library.

The results for Virtex-II are based on the low-level simulation from VHDL codings of our designs. These designs were synthesized using XST (Xilinx Synthesis Technology) in Xilinx ISE 4.1i. The place-and-route file (.ncd file) was obtained for Virtex-II XC2V1500 (package bg575, speed grade -5) and XC2V3000 (package bg728, speed grade -5). The input test vectors for the simulation were randomly generated and had an average switching activity is 50%. Mentor Graphics ModelSim 5.5e was used to simulate the designs and generate simulation results (.vcd file). These two files were then provided to the Xilinx XPower tool to evaluate the average power dissipation. Energy dissipation was obtained by multiplying the average power by the latency. We observed that the estimation error using our functions (see Table 3) is below 20% for energy dissipation. We have also adopted a technique based on statistical analysis. We utilize confidence intervals about the sample mean energy dissipation for a design. Confidence intervals allow us to address dependency upon input stimuli because they describe the likelihood that the true mean over an entire population is within a certain range of the experimental mean. The equation $\overline{x} \pm z_{\alpha/2}(s/\sqrt{M})$ is employed to estimate the confidence interval for our simulations where $\overline{x}$ is the sample mean (the mean found by experiment), α is a number between 0 and 1, $z_{\alpha/2}$ is a constant as explained in [7], s is the population standard deviation, and M is the number of samples. To statistically analyze energy dissipation for the matrix multiplication, we performed 50 different $n \times n$ matrix multiplication trials for our designs. Each trial consists of performing the low-level simulation procedure, as described above, with the uniformly distributed, randomly generated matrices as input. For example, for n=15, we have a mean energy dissipation of 1509.72 nJ with a 95% confidence interval of ± 1.85.

Table 3 and Table 5 compare the performance of our designs against the Xilinx design and the DSP solution from TI [16]. Xilinx provides the various sizes of FFT in CoreGen library. We run Xilinx FFT at 100 MHz. Xilinx also provides an optimized design for 3×3 matrix multiplication only. The Xilinx matrix multiplication design executes at 150 MHz. For $n > 3$, we use block matrix multiplication. The TMS320C6415 results come from using Texas Instruments Code Composer Studio 2.1. For FFT and matrix multiplication, we ran the *DSP_fft* and *DSP_mat_mult* algorithms from the Texas Instruments DSP library [16]. The latency is based directly on profiling. To compute energy dissipation, we again assume the 75% high/25% low activity category, as described above. We have observed that the estimation error using the high-level estimation (see Table 3) is below 11% for energy dissipation.

Table 2: Energy distribution for matrix multiplication

Problem size(n)	Logic E (nJ)	Interconnect E (nJ)
3	8.99	6.99
6	58.82	47.09
12	421.73	353.24
15	785.55	724.17

These comparisons are based on individual metrics as well as more comprehensive metrics of energy×area×latency (EAT) product [2]. When comparing multiple designs with the EAT metric, the one with the smallest EAT value is the best. Table 4 shows the improvement of the proposed designs compared with those from Xilinx and TI DSP.

Energy efficiency for FFT is achieved using pipelining, appropriate disabling of the modules and choosing efficient memory bindings. Pipelining increases the throughput and decreases the effective latency. Disabling twiddle factor computation modules reduces the amount of energy used by more than 30%. Also, selecting a radix-4 algorithm as the basic module reduces the number of complex multiplications in the design. Choosing bindings such that the data buffers and sine/cosine lookup tables are implemented using SRAM for $N < 64$ or BRAM for $N \geq 64$ increases the energy efficiency. We also observed that while parallelism increases the throughput and eventually the energy efficiency, the energy used by the interconnect in FPGAs significantly increases. For example, the design of $(V_p, H_p) = (4, 1)$ dissipates 20% more energy than the design of $(V_p, H_p) = (1, 4)$ for $N = 256$ while the former has 2 times higher throughput. The increased energy dissipation is due to the former's use more interconnects and memory elements.

Energy efficiency for matrix multiplication is due to reduction in latency, pipelining, parallelism, and energy-efficient bindings. Table 2 shows the energy distribution over logic and interconnect (XPower provides this information, including for I/O). While the energy distribution of a typical design is dominated by the interconnect [13], the logic is dominant in our proposed designs. This is due to the architecture selection of the linear array. Figure 10 shows the energy dissipation of matrix multiplication on a linear array architecture. For problem sizes up to 24, the full parallel architecture gives good energy performance. However, for problem sizes greater than 24, block matrix multiplication

Table 3: FFT performance of Xilinx library based design, TI DSP, and our designs

Problem	Xilinx (100MHz)				TI DSP (500MHz)				Our designs (100MHz)								
size(n)	A	T	Em	EAT	T	Eest	Em	E err	Vp	Hp	Binding	A	T	Eest	Em	E err	EAT
16	136	0.16	179.6	0.04	0.17	183.2	199.9	9%	1	2	SRAM	1171	0.16	65.4	77.0	15%	0.014
									4	2	SRAM	2390	0.04	63.5	75.2	15%	0.007
64	1079	1.92	1785.6	3.70	0.60	656.8	716.4	9%	1	3	SRAM	2266	0.64	552.4	493.3	12%	0.72
									1	3	BRAM	1613	0.64	464.2	390.4	19%	0.40
									4	3	SRAM	5690	0.16	393.9	418.7	6%	0.38
									4	3	BRAM	4193	0.16	403.2	400.4	1%	0.27
256	1303	7.68	6927.3	69.32	2.53	2958.3	3008.3	2%	1	4	BRAM	2050	2.56	2582.2	2223.1	16%	11.67
									4	4	BRAM	5624	0.64	2203.2	1971.3	12%	7.10
1024	1557	30.72	34283.5	1639.82	12.07	14284.7	14365.7	1%	1	5	BRAM	2744	10.24	14963.5	13739.4	9%	386.06
									4	5	BRAM	6673	2.56	11424.7	9204.2	20%	157.23

Eest is the estimated energy (nJ). Em is the measured energy(nJ) from the synthesized designs.

The unit of EAT is 1E-9. The unit of Area (A) is slice, The unit of time (T) is usec.

Table 4: FFT performance comparison with Xilinx library based designs and TI DSP

Problem	Our designs (100MHz)			Our designs vs. Xilinx designs				Our designs vs. DSP	
size(n)	Vp	Hp	Binding	E (decrease)	A (increase)	T (decrease)	EAT (decrease)	E (decrease)	T (decrease)
16	1	2	SRAM	57%	0.86x	1.0x	2.71x	61%	1.06x
	4	2	SRAM	58%	1.75x	4.0x	5.45x	62%	4.25x
64	1	3	SRAM	72%	2.10x	3.0x	5.17x	31%	0.94x
	1	3	BRAM	78%	1.49x	3.0x	9.18x	46%	0.94x
	4	3	SRAM	77%	5.27x	12.0x	9.70x	42%	3.75x
	4	3	BRAM	78%	3.89x	12.0x	13.77x	44%	3.75x
256	1	4	BRAM	68%	1.57x	3.0x	5.94x	68%	0.99x
	4	4	BRAM	72%	4.32x	12.0x	9.77x	72%	3.95x
1024	1	5	BRAM	60%	1.76x	3.0x	4.25x	60%	1.18x
	4	5	BRAM	73%	4.29x	12.0x	10.43x	73%	4.71x

Table 5: Energy dissipation for matrix multiplication

Problem	Xilinx (150MHz)				TI DSP (500MHz)				Our designs (150MHz)					
size(n)	A	T	Em	EAT	T	Eest	Em	E err	A	T	Eest	Em	E err	EAT
3	299	0.18	23.12	1.24	0.14	149.94	166.6	11%	434	0.06	13.37	15.98	16%	0.42
6	299	1.44	184.98	79.65	0.38	454.58	456.96	1%	861	0.24	98.95	105.91	7%	21.89
12	299	11.52	1479.84	5097.29	1.54	1825.46	1827.84	0%	1699	0.96	755.95	774.97	2%	1264.00
15	299	22.50	2890.32	19444.63	3.22	3834.18	3836.56	0%	2083	1.50	1463.47	1509.72	3%	4717.12

Eest is the estimated energy (nJ). Em is the measured energy(nJ) from the synthesized designs.

The unit of EAT is 1E-12. The unit of Area (A) is slice, The unit of time (T) is usec.

Table 6: Energy dissipation for FFT and matrix multiplication on PowerPC (300MHz)

FFT (300MHz)					Matrix Multiplication (300MHz)				
Problem size (n)	Tℓ (usec)	Tm (usec)	Eℓ (nJ)	Em (nJ)	Problem size (n)	Tℓ (usec)	Tm (usec)	Eℓ (nJ)	Em (nJ)
16	2.24	59.40	605	16038	3	0.09	34.98	24	9444
64	23.68	353.10	6394	95337	6	0.72	310.29	194	83778
256	183.47	n/a	49536	n/a	12	5.76	2311.35	1555	624065
1024	1206.40	n/a	325728	n/a	15	11.25	4436.64	3038	1197893

Tℓ is a lower bound of time (T). Tm is the measured time (T). Eℓ is a lower bound of energy (E). Em is the measured energy (E).

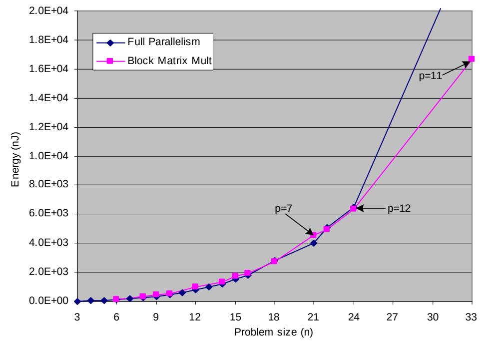


Figure 10: Energy dissipation of matrix multiplication using fully parallel architecture and block matrix multiplication (150Mhz, 8 bits per data)

leads to better energy performance. Since the internal storage required for large problem sizes increases dramatically, parallel processing has a negative effect on the total energy dissipation. This result implies that a designer must carefully investigate the trade-offs among the algorithm and the degree of parallelism.

For the PowerPC core on Virtex-II Pro, we coded FFT and matrix multiplication programs in C. We then compiled using PowerPC-based gcc compiler and simulated them using the tools from the Xilinx Virtex-II Pro Developers Kit. We performed a SWIFT model simulation of the PowerPC. The SWIFT model simulation permits the execution of actual PPC405 code. The data to be computed are stored in the BRAM of the Virtex-II Pro. The results for latency come directly from the simulation while the energy results come from assuming power dissipation of 0.9 mW/MHz and a clock frequency of 300 MHz, as described above. The results are shown in Table 6 with the lower bounds obtained from Section 4. The constant factor is relatively high in PowerPC designs due to the fact that very rough estimates that ignore all types of instructions but arithmetic instructions are used. Note that as problem size increases and arithmetic instructions do begin to dominate, the constant factor goes down, albeit slowly. Additionally, the enegy dissipated on the BRAM is not included; this would add more energy dissipation. In the future work, more accurate high-level estimation technique for PowerPC core is required.

6. FUTURE WORK

There are many areas for future work. For instance, we can develop a technique to more accurately measure the energy dissipated by the DSPs and embedded processors. One area of particular interest is analyzing the influence that duty cycle can have on the choice of device on. For example, one device may be more energy-efficient for a given kernel application, but it may dissipate more energy during idle time. Choosing the right device in this situation involves not only the type of analysis presented in this paper, but also analysis of specific usage scenarios and the availability of low-power modes in each type of device.

7. ACKNOWLEDGEMENTS

The authors wish to thank Jingzhao Ou and Gokul Govindu for their contributions to the low-level simulation results.

This work is part of the DARPA PARIS project, supported by the DARPA PAC/C program under contract S7-6AX077X 62A9 and in part by the National Science Foundation under award No. 99000613.

8. REFERENCES

[1] Altera Corporation, http://www.altera.com. *Apex 20K data sheet*, 2002.

[2] B. Bass, “A Low-Power, High-Performance, 1024-Point FFT Processor,” *IEEE Journal of Solid-State Circuits*, Vol. 34, No. 3 (1999) 380-38.

[3] S. Choi, J.-W. Jang, S. Mohanty, and V. K. Prasanna, “Domain-Specific Modeling for Rapid System-Wide Energy Estimation of Reconfigurable Architectures,” *Engineering of Reconfigurable Systems and Algorithms*, 2002.

[4] E. Chu and A. George, *Inside the FFT Black Box*, CRC Press, 2000.

[5] J. A. B. Fortes, K. S. Fu, and B. Wah, “Systematic Approaches to the Design of Algorithmically Specified Systolic Arrays,” *International Conference on Acoustics, Signal, and Speech Processing*, 1985.

[6] C. Dick, “The Platform FPGA: Enabling the Software Radio,” *Software Defined Radio Technical Conference and Product Exposition (SDR)*, November 2002.

[7] R. Hogg and E. Tanis, *Probability and Statistical Inference,* 6th Eds., Prentice Hall, pp656-657, 2001.

[8] J.-w. Jang, S. Choi, and V. K. Prasanna, “Energy Efficient Matrix Multiplication on FPGAs,” *Field-Programmable Logic and Applications*, 2002.

[9] S. Lei and K. Yao, “Efficient Systolic Array Implementations of Digital Filtering,” *IEEE International Symposium on Circuits and Systems*, 1989.

[10] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, Prentice Hall, 1989.

[11] A. Raghunathan, N. K. Jha, and S. Dey, *High-level Power Analysis and Optimization*, Kluwer Academic Publishers, 1998.

[12] R. Scrofano, S. Choi, and V. K. Prasanna, “Energy Efficiency of FPGAs and Programmable Processors for Matrix Multiplication,” *IEEE International Conference on Field-Programmable Technology*, 2002.

[13] L. Shang, A. Kaviani, and K. Bathala, “Dynamic Power Consumption in Virtex-II FPGA Family,” *International Symposium on Field Programmable Gate Arrays*, 2002.

[14] H. Styles and W. Luk, “Customising Graphics Application : Techniques and Programming Interface,” *IEEE Symposium on Field Programmable Custom Computing Machines*, 2000.

[15] R. Tessier and W. Burleson, “Reconfigurable Computing and Digital Signal Processing: A Survey,” *Journal of VLSI Signal Processing*, May/June 2001.

[16] Texas Instruments, http://www.ti.com.

[17] Xilinx Incorporated, http://www.xilinx.com.

[18] G. Yeap, *Practical Low Power Digital VLSI Design*, Kluwer Academic Publishers, 1998.