

# Energy-Efficient Design of Kernel Applications for FPGAs Through Domain-Specific Modeling

S. Choi<sup>1</sup>, R. Scrofano<sup>2</sup>, and V. K. Prasanna<sup>1</sup>

<sup>1</sup>Department of Electrical Engineering-Systems, University of Southern California, Los Angeles, CA

<sup>2</sup>Department of Computer Science, University of Southern California, Los Angeles, CA

{seonil, rscrofan, prasanna@halcyon.usc.edu}

## Abstract

Because of their high performance and flexibility, FPGAs are an attractive option for use in embedded systems, where both high performance and low energy consumption are important. Therefore, it is important to create FPGA designs that are not only high performance but also low energy. The flexibility of FPGAs facilitates their high performance, but also makes it difficult to design for them. The design space for mapping a kernel application onto an FPGA is very large. Consequently, a design methodology that allows the designer to analyze tradeoffs between architectures and algorithms and efficiently explore the design space is needed. In this paper, we present a methodology for efficiently exploring the design space and analyzing tradeoffs when mapping kernel applications onto FPGAs. This methodology is based on *domain-specific modeling*. Then, we present a design for matrix multiplication that shows that applying our design methodology can lead to designs that are energy-efficient. We compare our design to the Xilinx core for matrix multiplication using both high-level estimation and more accurate low-level simulations. For low-level simulations, we present our data generation method and statistical analysis of the results.

## I. INTRODUCTION

Technological advances have made multi-million gate FPGAs a reality. FPGAs are flexible, reprogrammable devices. Because of this flexibility, FPGA designs can be fine tuned to meet the needs of a given kernel application (*kernel applications* are the tasks that make up applications, e.g. matrix multiplication and fast Fourier transform are kernel applications). This customizability leads to very high performance.

Due to their flexibility and their high processing power, FPGAs have become a popular alternative to traditional DSPs and ASICs. As a result, many high-level algorithms are being mapped onto FPGAs. FPGAs are particularly well-suited to algorithms with inherent parallelism, such as signal processing algorithms that are part of battlefield and space-based applications. In these applications, high performance is important, but not the only concern. Energy efficiency is also a very important factor in determining the worth of an application's implementation. There is a limit to the amount of energy that a battery can provide. Further, current technology limits the amount of energy that can be scavenged

from the environment around a device such as a sensor. Yet, in military and space applications, changing a battery may not be feasible. Increasing the lifetime of battlefield and space devices is, therefore, an important goal. Clearly, then, when implementing military and space applications, energy efficiency is crucial.

However, finding energy-optimal designs for FPGAs is problematic because the design space for FPGAs is very large. There can be many candidate architectures for a given application and many algorithms that can be used with each architecture. Therefore, designing for FPGAs requires the traversal of a very large design space, making it difficult for the designer to follow the traditional design path of performing low-level simulations of all candidate designs. Consequently, a methodology by which energy consumption can be rapidly estimated is needed. Such a methodology will enable the focus of the design effort to be on a small part of the design space. In this paper, we present a methodology, based on *domain-specific modeling*, for rapid energy consumption estimation and analysis of energy, area, and latency tradeoffs and show the results of applying this methodology to matrix multiplication, an important signal processing kernel application. We compare a design created using our design methodology to the design from the Xilinx core library.

The remainder of this paper is organized as follows. In section 2, the design methodology is presented. Each of the four steps required to create a design using this methodology are described. In section 3, results for applying our design methodology to the problem of  $n \times n$  matrix multiplication are given. We use both high-level estimations and low-level simulations with the low-level simulations serving to validate the high-level estimates. We present our method for generating test input data for the low-level simulations and analyzing the resulting energy data. In section 4, the paper concludes with a summary of our work and areas for future research.

## II. METHODOLOGY

The goal of our design methodology is to design energy-efficient data paths for a given kernel application. The methodology generates a set of designs that provides tradeoffs among energy, area, and latency. Exploring these designs leads to the identification of the optimal design that meets the requirements. Further, through the use of domain-specific models, our methodology facilitates high-level evaluation of

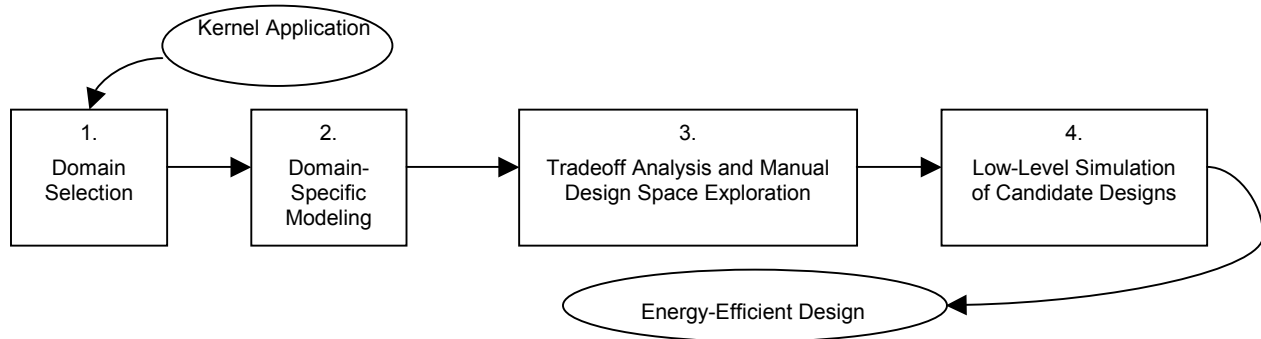


Figure 1: The design methodology.

possible designs. Time need not be spent needlessly performing time-consuming, low-level simulations on designs that will not meet the requirements or could not possibly be optimal. Instead, effort is focused on understanding and designing algorithms for the given kernel application. So that a choice can be made among the candidate designs, low-level simulations are performed, but only after the best possible architectures and algorithms for the kernel application are identified. Thus, low-level simulations are performed on a drastically reduced design space.

Our methodology consists of four steps (see Figure 1). These steps are summarized here. A more complete description can be found in [6].

### A. Domain Selection

A *domain* is a family of architectures and algorithms that implement a specific kernel application. For example, matrix multiplication on a linear array architecture is a domain. In FPGA design, there can be many possible domains for a given kernel application. Some of these domains may not yield designs that fit a given selection criteria. In the domain selection phase of the design methodology, the candidate domains are selected. The designer must use his knowledge of the application to choose only domains that could possibly yield promising results. Once domains are selected, effort only needs to be spent on designs within the domains. All other designs, which cannot yield the desired results, are weeded out right at the beginning of the design cycle.

### B. Domain-Specific Modeling

*Domain-specific modeling* is the process through which a high-level model of the domain is created. This high-level model can then be used to analyze power and energy consumption, as well as the latencies and areas of the designs in the domain.

In domain-specific modeling, an architecture is abstracted into *components*. Components are broken into two types: *Relocatable modules* (RModules) and *Interconnects*. Relocatable modules are parts of the architecture, such as multipliers and registers, that are assumed to consume the same amount of power wherever they are located on the FPGA. Interconnects are the connections between the RModules.

In addition to RModules and Interconnects, the model of the architecture contains several parameters, identified by the designer, that affect power consumption. Examples are frequency, precision, size of memory, and possible power states. The parameters are then used in component-specific power functions. These power functions capture the effect that varying the model parameters has on the power consumption of the components. In order to obtain the power functions, low-level simulations are performed for each component. More information on the low-level simulation of the components is given below. Curve fitting is done on the data from the low-level simulation to create a power function in terms of the model parameters.

The architecture model also contains *component power state (CPS) matrices*. These matrices tell in which power state each component is for each cycle of operation. In order to fill in the matrices, the designer uses his knowledge of the algorithm and how it moves data through the architecture. Combining the CPS matrices and the component-specific power functions yields the system-wide energy function. This function sums the power consumed by each component in each cycle, yielding the total energy used by a design in a domain with various parameter values [2]. Thus, domain-specific modeling allows the designer to estimate energy consumption of various designs without having to perform low-level simulations of each individual design in the domain.

#### 1) Low-Level Simulation of Components

Low-level simulation of components is an important part of the domain-specific modeling phase of our design methodology. Low-level simulations provide the data that is used to build the component-specific power functions, which are then used in computing the system-wide energy. The accuracy provided by low-level simulations of component power consumption is important in obtaining accurate high-level estimations.

As described in [2], low-level simulations are performed at various *design points*, where each design point is a unique combination of architectural parameters. Curve fitting is applied to the results of each of these low-level simulations in order to obtain a function for the power consumption of a given component. Once the curves are generated, they can be reused with any model using the same components and parameters.

Simulating a component at a low level requires several steps. The component, must be modeled in a high-level design language, simulated for various inputs, and the power consumption must be calculated from the simulation results and the placed and routed design. Below, we describe this procedure for designing for a Xilinx Virtex-II FPGA [14].

First, the component is modeled in VHDL. The design is then synthesized. With the Virtex-II as the target device, we use Xilinx XST<sup>1</sup> for synthesis. This tool takes the VHDL file as input and outputs a netlist description of the design, which describes the parts of the design and how they are connected. This netlist then must be placed and routed. For the Virtex-II, we use Xilinx PAR<sup>1</sup> to do the place and route. The output of this tool is in the form of a .ncd file. This file describes where on the FPGA each part of the design will be located and what interconnect resources, if any, will be used. The .ncd file can be converted to back-annotated VHDL and used for simulation. Simulation also requires test input waveforms as the stimuli for each component.

A test input waveform is generated for each component. This waveform tells what the input to each input port of the design will be for each clock cycle of the simulation. It can be randomly generated with a switching activity chosen by the designer. Or, if the designer has some representative data for his kernel application, he can provide this data as the test input waveform. In either case, the test input waveform will be used for simulation of the design.

The test input waveforms and the back-annotated VHDL code for the design are then input to the simulation tool. For the Virtex-II FPGA, we have used ModelSim 5.5e<sup>1</sup> for simulation. The output of ModelSim 5.5e is a .vcd file. This file details what happens in the circuit at each cycle.

The output file from place-and-route (.ncd file) and the output file from the simulation (.vcd file) are used as input to the power estimator. In the case of the Virtex-II, the power estimator is XPower<sup>1</sup> from Xilinx. The output of XPower is a report file that details the power consumption of the design. This power data is then used in the curve fitting to determine the component-specific power function of each component. RModules can be modeled in VHDL and simulated in isolation. Interconnects, however, require a different strategy. A cluster of  $k$  interconnected RModules is simulated. The power consumption of the Interconnect is found as the difference between the power consumption of the cluster and the sum of the power consumptions of each of the RModules [2].

Once the domain-specific modeling is complete, the designer can move to the next phase of the design methodology.

### *C. Tradeoff Analysis and Manual Design Space Exploration (DSE)*

In this phase of the design methodology, the design space of each domain under consideration is traversed in order to identify designs that meet the given criteria. This part of the methodology serves to weed out designs that cannot possibly meet the application specification. It would be a waste of resources to perform time-consuming, low-level simulations on such designs. Instead, the domain-specific model can be used to perform analysis at a high-level, using less time to find designs that are promising and weed out those that are not. Further, using the high-level equations, the designer can identify the tradeoffs that come from varying the inputs to the functions. Our design methodology does not endorse any particular DSE technique because the technique can be domain-dependent. The next phase of the design methodology gives the designer the opportunity to choose between promising candidate designs (those that meet the selection criteria).

### *D. Low-Level Simulation of Candidate Designs*

After DSE, there should be only a few candidate designs remaining that meet the selection criteria. Low-level simulation of those designs has two functions: validating the high-level model and distinguishing among candidate designs whose difference could be the result of experimental error.

The high-level model should provide good estimates of energy consumption. Our studies have shown that the error due to high-level estimation is usually no more than 10%. However, if the designer did not choose enough architectural parameters or missed choosing architectural parameters that have a profound impact on energy consumption, the high-level estimation error may be greater. If this is the case and the error is intolerably high, the designer needs to return to the domain-specific modeling stage and create a more accurate high-level model. Otherwise, he may have excluded designs that could indeed meet his selection criteria.

The high-level model may be accurate to about 10%, but this may not be good enough to distinguish between some designs. In this case, low-level simulation is needed to provide more accurate results that will allow a designer to make an informed choice between two or more similarly performing designs for a given kernel applications.

The low-level simulation of the candidate designs follows much the same flow as the low-level simulation of the components. The main difference is that, in this phase of the design methodology, complete designs are being simulated, not individual components.

Another difference occurs in the generation of the test input waveforms. As before, if the designer has some sort of benchmark data to use as input, he can use that to generate the test input waveforms. Or, if there is no knowledge of the data to be processed by the kernel application, input waveforms can be generated randomly from some random distribution. The designer must exercise care, though, when designing the

---

<sup>1</sup> Part of Xilinx ISE 4.1i [14]

waveforms for different designs of the same kernel application. These designs may have different numbers of input ports, which means that the same waveform cannot be used for both designs. Further, even if there are the same number of input ports for each design, the algorithm used by each design may access the input data in different patterns. Therefore, providing the same input waveform to each design would not provide a completely accurate comparison of the two designs.

As an example, consider the multiplication of two  $n \times n$  matrices,  $A$  and  $B$ , using two different candidate designs. Assume the two designs have the same inputs but access the data in a different order. The test input waveforms represent the entries from each matrix being given as input to the design. If the same waveforms are used for each design, then one design will multiply  $A$  and  $B$ , while the other design will wind up multiplying some permutation of  $A$  and some permutation of  $B$ . Therefore, it is important that test waveforms be generated carefully to allow for fair and accurate comparisons of multiple designs.

### III. RESULTS

We now present results from applying our design methodology to a particular domain for  $n \times n$  matrix multiplication. We do high-level analysis of the results and compare them to results from the use of an optimized core from the Xilinx library. Finally, we present low-level simulation results for the  $3 \times 3$  matrix multiplication problem.

A summary of the application of our design methodology to the problem of  $n \times n$  matrix multiplication is presented here. A more complete description of the domain can be found in [5] and a more complete example of applying our methodology for energy efficiency can be found in [6].

The first step in the design methodology is domain selection. The candidate architecture family that will be examined in this paper is that of a linear array of processing elements (PEs). In this architecture, data is passed from one PE to the next, in succession. The input goes to the first processing element and the output comes out of the last. Also note that each of the PEs is the same.

The algorithm chosen is that which is described in [5] as the algorithm based on that paper's Corollary 1. This algorithm cleverly moves data through the PEs such that the time complexity of  $n \times n$  matrix multiplication is

$$\left(\frac{n}{p}\right)^3 (p^2 + 2p + 1) \quad (1)$$

cycles, where  $p$  is the number of PEs.

In the domain-specific model for this domain, the PEs are considered the RModules and the buses that connect the PEs

are considered the Interconnects. Each PE has 4 registers, 1 MAC, and one SRAM. The number of words in the SRAM depends on the number of PEs being used. If there are  $p$  PEs used in the architecture, then the number of words in each PE's SRAM is  $p$ . Additionally, the design has 2 I/O ports that bring in data. This design is termed the "off-chip" design because the input data (the matrices) are stored in memory external to the FPGA. An "on-chip" design is also possible. The data is stored in memory internal to the FPGA, such as the Block SelectRAMs in the Xilinx Virtex-II FPGA.  $n \times n$  matrix multiplication requires the use of  $2n^2$  words of on-chip memory. The on-chip design does not, however, require the 2 I/O ports that bring in data while the matrices are being multiplied. In this paper, we will focus on the implementation of the off-chip design.

Table 1: Power usage for parts of a PE (8-bit precision)

Variable	Power (mW)
$P_{Mult}$	17.00
$P_{SRAM}$	8.39
$P_R$	2.34
$P_{IO}$	11.31
$P_{IC}$	10.00

Having isolated the pieces that make up a PE and their quantities, it is possible to create a component-specific power function for a PE. The power function is

$$P_{Mult} + \left\lceil \frac{p}{16} \right\rceil P_{SRAM} + 4P_R \quad (2)$$

where  $p$  is the number of PEs,  $P_{Mult}$  is the power used by a multiplier,  $P_{SRAM}$  is the power used by SRAM memory, and  $P_R$  is the power used by a register. To get the total power consumed by the system, Equation 2 needs to be multiplied by the number of PEs. Further, the power used by the I/O modules,  $P_{IO}$ , needs to be added in, as does the power from the Interconnects,  $P_{IC}$ . These values, as well as the other power values, can be found through low-level simulation as described above. They are given in Table 1. Thus, the total power used by the system is

$$p \left( P_{Mult} + \left\lceil \frac{p}{16} \right\rceil P_{SRAM} + 4P_R \right) + 2P_{IO} + (p-1)P_{IC} \quad (3)$$

This is the estimate of power used by the system. The energy used by the system can be estimated as the product of the latency, in seconds, and Equation 3. The energy used by the system is, then

Table 2: Comparison of the linear array design with the Xilinx matrix multiplication core for various problem sizes. E is for energy, L for latency, and A for area. % Reduction, Speedup, and Area Increase are in terms of the linear array design compared to the Xilinx design (e.g. in the  $3 \times 3$  case, our design led to a 32% energy reduction compared to the Xilinx design) [5].

Matrix Size	$E_{\text{LinearArray}}$ (nJ)	$E_{\text{Xilinx}}$ (nJ)	% Reduction	$L_{\text{LinearArray}}$ (cycles)	$L_{\text{Xilinx}}$ (cycles)	Speedup (x)	$A_{\text{LinearArray}}$ (slices)	$A_{\text{Xilinx}}$ (slices)	Area Increase (x)
$3 \times 3$	16	33	52	16	45	2.8	393	179	2.2
$6 \times 6$	92	261	65	49	360	7.3	786	179	4.4
$9 \times 9$	276	879	69	100	1215	12.2	1179	179	6.6
$12 \times 12$	614	2084	71	169	2280	13.5	1572	179	8.8
$15 \times 15$	1153	4070	72	256	5625	22.0	1965	179	11.0
$24 \times 24$	5213	16671	69	625	23040	36.7	3912	179	21.9
$48 \times 48$	45555	133364	66	2401	184320	76.7	9360	179	52.3

$$\frac{\left[ \left( \frac{n}{p} \right)^3 (p^2 + 2p + 1) \right] \left[ p \left( P_{\text{Multi}} + \left[ \frac{p}{16} \right] P_{\text{SRAM}} + 4P_R \right) + 2P_{\text{IO}} + (p-1)P_{\text{IC}} \right]}{f} \quad (4)$$

where  $f$  is the frequency at which the design is run [5].

It is also possible to construct a similar equation for area, and the latency equation of the linear array architecture, in cycles, is given above in Equation 1 [5]. Using such equations, it is possible to analyze the tradeoffs among energy, area, and latency. For example, varying  $p$ , the number of PEs, will lead to different latencies, area usage, and energy consumption. Based on the requirements for a given application, the right number of PEs can be selected. If other RModules had been identified, the number of these could vary also, leading to more tradeoff analysis.

We now, using high-level estimation, compare the implementation of the linear array design on a Xilinx Virtex-II FPGA running at 150 MHz with that of the Xilinx optimized matrix multiplication core for various values of  $n$ . Table 2 shows the results of the comparison. It is assumed that the matrix entries have 8-bit precision. For the linear array architecture, the number of PEs is allowed to increase with  $n$ .

Clearly, from high-level estimation, the linear array design seems to be superior to the design provided by Xilinx. To validate this fact, we perform low-level simulations.

### 1) Low-Level Simulation

In order to validate our high-level estimations, we perform low-level simulations for the linear array architecture and the Xilinx core architecture. For each architecture, we simulate the case where  $n = 3$ . That is, we simulate  $3 \times 3$  matrix multiplication. As before, we use 8 bit precision for the matrix entries. In order to do so, we follow the steps for low-level simulation of a candidate design, as outlined in section II.D. The first step is to code each design in a high-level design language. We do so by coding each design in VHDL. The

designs are synthesized and placed and routed using the appropriate software tools, as described above.

Before simulation, we need to create test input waveforms for each design. We have written a C++ program and a Perl script to automate the test input waveform generation. The C++ program is responsible for

1. generating the requisite number of matrices,
2. outputting the matrix entries in the proper format such that the Xilinx architecture and the linear array architecture are both performing the multiplication of the same two matrices, and
3. keeping track of the switching activity of each matrix entry input port of the design.

The Perl script is responsible for

1. running the C++ program multiple times and
2. splicing files such that the output from the C++ program can be used in the simulations.

The program flow for the C++ program and the Perl script is shown in Figure 2. One input to the C++ program is the number of rows (equal to the number of columns) in the matrices to be multiplied (i.e.,  $n$  for  $n \times n$  matrix multiplication). Additionally, the number of matrices to generate is an input parameter ( $m$ ). In a real world application, it is unlikely that a matrix multiplication will be done in isolation. Because of this, we take the pipelining effect into account in our simulations. One set of simulations is performed for a pair of matrices (one matrix for each input of the design). At the beginning of the computation, the pipeline of the design is empty and is filled up before the computation starts. During this period, less energy is consumed. However, in general, it is likely that multiple matrices are fed to the design as a stream of data. So, most of the time, the pipeline is unlikely to be empty, and the computation and data feed are overlapped. We consider the initial operation to be a special case. Thus in generating the test input waveforms, we repeat the pair of the same matrices  $m$  times so that we can measure the energy consumption of the

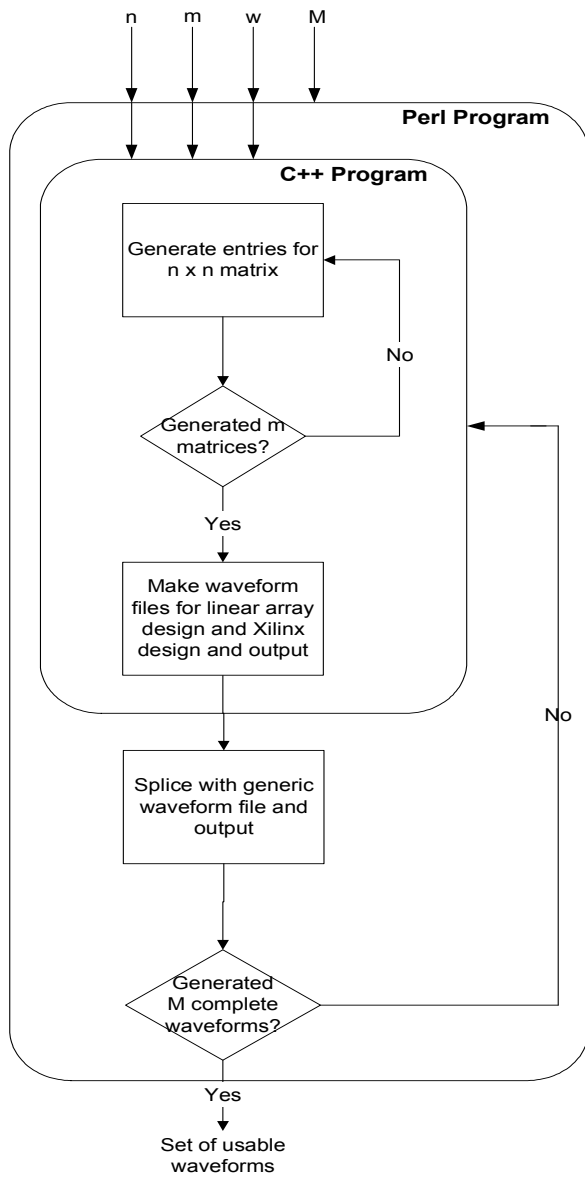


Figure 2: Flowchart for the Perl script and C++ program used in generating test input waveforms.

normal operation in which all data feeding and computation are pipelined and overlapped.

The first step in the C++ program is to randomly generate the  $n \times n$  matrices. This is done matrix by matrix, entry by entry, bit by bit. For example, for ten  $3 \times 3$  matrices with 8-bit precision, the program will start by generating the first bit of the first entry in the first matrix, then the second bit of the first entry, and so on until the first entry is complete. Then it will do the same for the second entry, and so on, until the first matrix is complete. Then it will repeat the process for the second matrix, and so on, until the tenth matrix is complete.

Once the matrices have been generated, two output files are created: one consisting of the waveforms for the linear array design and one consisting of the waveforms for the Xilinx design. As described above, care must be taken to ensure that the waveforms provide an accurate comparison of

the two designs. As a result, the linear array architecture waveform file is generated by traversing each matrix in the order that the input to the linear array architecture would be needed if this design were implemented and used to multiply two matrices. Likewise, for the Xilinx architecture, each matrix is again traversed, this time in the order that the input would be needed if the Xilinx design were implemented.

As the matrices are being traversed, the program keeps track of the switching activity of the inputs to each matrix entry input port of the design. The switching activity is, therefore, determined by the entries in the matrix. We have not used any data specific to a given application for our comparisons. Rather, we have assumed that the data will be uniformly distributed. So, for each bit in each entry in each matrix, the probability that it is a 1 is 0.5 and the probability that it is a 0 is 0.5. Consequently, for any matrix entry input port in the waveform file that has a 0 on one cycle, the probability that it will have a 1 on the next cycle is 0.5 and vice versa. This fact should lead to a switching activity of 50% for the linear array design, which takes one matrix entry per clock cycle as input. The average switching activities for each bit of one input to the linear array design over fifty sets of ten matrices are shown in Table 3. Note that in each case the switching activity is slightly less than 50%. This is due to the fact that outputs are still being calculated after all the inputs have been fed into the array. The other input is omitted for brevity. The switching activity is completely dependent on the entries in the matrix, so other switching activities can be achieved by using different matrix data.

The average switching activity for the Xilinx design, with the same 50 sets of ten input matrices, is also shown in Table 3. Note that it is much less than for the linear array design. The main reason for this difference is that the Xilinx design accesses data differently. While the linear array design accesses data entry by entry, one entry per clock cycle, the Xilinx design accesses data row by row (hence the three inputs in Table 3), one row per sixteen clock cycles. This less frequent accessing of new data by the Xilinx design leads to lower switching activity. Despite this lower switching activity, the Xilinx design uses more energy than the linear array design, as evidenced by the high-level estimates presented above and the low-level simulation results presented below.

The Perl script serves to run the C++ waveform generation program multiple times ( $M$ ) so that many test input waveform files can be generated at once. The C++ program also only generates the waveform data for the input ports that are for matrix entries. The Perl script takes the outputs from the C++ program and splices them with generic matrix multiplication waveform files for each type of design to create complete and correctly formatted test input waveform files.

With test input waveform files and back-annotated VHDL, the two matrix multiplication designs can be simulated using ModelSim 5.5e. Finally, the output files from the simulation and the place and route are used as input for Xilinx XPower, the power estimator. The output from XPower is used to calculate the energy consumed by each matrix multiplication

design. We have statistically analyzed the outputs from 50 simulations of multiplying  $n \times n$  matrices in order to have confidence in the validity of using the results for comparison.

## 2) Statistical Analysis

For statistical analysis, we utilize *confidence intervals* about the sample mean energy consumptions for our design

Table 3: Switching activity for each input bit of the  $A$  input for both our design and the Xilinx design.

Input Bit	Linear Array (%)	Xilinx (%)		
		Input 0	Input 1	Input 2
0	46	3	3	3
1	46	4	3	4
2	49	3	3	3
3	47	3	3	3
4	47	3	3	3
5	45	3	3	3
6	45	3	3	3
7	45	3	3	3
Overall	46	3	3	3

and the Xilinx design and about the difference between the two means. Confidence intervals allow us to address dependency upon input data because they describe the likelihood that the true mean over an entire population is within a certain range of the mean found from a sample out of the population. More specifically, assuming the data is normally distributed, if  $\mu$  is the unknown population mean and  $\bar{x}$  is the sample mean (the mean found by experiment), then with a probability of  $(1 - \alpha)$ ,  $\mu$  is within  $\pm z_{\alpha/2} \frac{\sigma}{\sqrt{M}}$

of  $\bar{x}$ , where  $\alpha$  is a number between 0 and 1,  $z_{\alpha/2}$  is a constant as explained in [4],  $\sigma$  is the population standard deviation, and  $M$  is the number of samples. The derivation of the confidence interval equation can be found in [4]. In order to use this equation, two criteria must be met: the data are from a normal distribution and the population standard deviation is known to be  $\sigma$ . That is, it assumes that the results are from  $N(\mu, \sigma)$ . If it cannot be assumed that the data are from a normal distribution, an approximate confidence interval can be obtained using the same equation, provided that the sample size is large enough (greater than 30) and the underlying distribution is not badly skewed or discrete [4]. Also, when  $\sigma$  is unknown,  $s$ , the sample standard deviation, can be used in its place, provided the sample size is large enough (50 or larger) and the underlying distribution is not badly skewed [4]. Thus, the confidence interval for  $\mu$  becomes

$$\bar{x} \pm z_{\alpha/2} \frac{s}{\sqrt{M}} \quad (5)$$

To statistically analyze the difference in energy consumption between our design and the Xilinx design, we perform 50 different  $3 \times 3$  matrix multiplication trials for the linear architecture and Xilinx's matrix multiplication core. All results described in this section are for  $3 \times 3$  matrix multiplication. Each trial consists of performing the low-level simulation procedure, as described above, with the uniformly distributed, randomly generated matrices as input and the Xilinx Virtex-II FPGA running at 150 MHz as the target platform. The test input waveforms are created by running the Perl script described above, telling it to create 50 test input waveforms and to pass to the C++ program that the matrices should be  $3 \times 3$  and there should be 10 matrices in the pipeline. Thus, each trial involves the multiplication of the same matrices using the linear architecture and the Xilinx architecture. Because we perform 50 trials, we are able to use the confidence interval formula as in Equation 5, regardless of the fact that we do not know if our data is normally distributed [4]. We assume that the distribution from which the results come is not too badly skewed or discrete.

The energy consumption values from each trial are used to calculate the sample mean and sample standard deviation of the energy consumption of each design. The mean and standard deviation for the linear array design are 16.79 nJ and 0.96, respectively. For the Xilinx design, these values are 24.70 nJ and 1.29, respectively. It appears, then, that, on the average, our design consumes less energy than the Xilinx design. To determine the validity of using the sample means as estimators of the true mean energy consumption, we calculate the 95% confidence intervals about each mean using Equation 5. In each calculation,  $\alpha = 0.05$ ,  $z_{\alpha/2} = z_{0.025} = 1.96$  [4], and  $M = 50$ . The confidence interval about the mean energy consumption of the linear array design is

$$16.79 \pm 1.96 \frac{0.96}{\sqrt{50}} = 16.79 \pm 0.26 \quad (6)$$

The confidence interval about the mean energy consumption of the Xilinx design is

$$24.70 \pm 1.96 \frac{1.29}{\sqrt{50}} = 24.70 \pm 0.36 \quad (7)$$

With 95% probability, the mean energy consumption over all combinations of ten-matrix pipelines for each design lie in the intervals given above. Notice that these intervals never cross. This means that, with 95% probability, the mean energy consumption of the linear array design is better than that of the Xilinx design. In fact, [5] uses the confidence intervals to claim that the Xilinx design will use, on average, at least 7.28 nJ more than the linear array design. Using that same logic, the Xilinx design will consume at most 8.53 nJ more energy than the linear array design. These bounds can be tightened, though, using the confidence interval about the difference of the two means.

The confidence interval about the difference of two means is very similar to that which is about a single mean. Here, the confidence interval gives a range around the difference of two sample means into which the difference between the two population means should fall. The equation for calculating the confidence interval about the difference of two means is

$$\bar{x} - \bar{y} \pm z_{\alpha/2} \sqrt{\frac{s_x^2}{M_x} + \frac{s_y^2}{M_y}} \quad (8)$$

where  $\bar{x}$  and  $\bar{y}$  are the two samples means,  $z_{\alpha/2}$  is the same constant as for the confidence interval about a single mean,  $s_x$  and  $s_y$  are the sample standard deviations corresponding to the sample means, and  $M_x$  and  $M_y$  are the number of trials conducted to obtain the corresponding sample means and standard deviations [4].

Using equation 8, it is possible to calculate the 95% confidence interval about the difference between the mean energy consumption of the Xilinx design and that of our design. It is

$$24.70 - 16.79 \pm 1.96 \sqrt{\frac{1.29^2}{50} + \frac{.96^2}{50}} = 7.91 \pm .45 \quad (9)$$

Therefore, the Xilinx design will, on the average, consume between 7.46 nJ and 8.36 nJ (44% and 50%) more energy than the design based on the linear array architecture, regardless of the input matrices (provided the matrix entries are uniformly distributed). Figure 3 shows the sample means and their confidence intervals, as well as the difference between the two sample means and the confidence interval about it.

#### IV. CONCLUSION

In this paper, a design methodology and a design generated by using that methodology have been presented. The design methodology utilizes both high-level modeling and low-level simulation. The matrix multiplication design that results from the use of the methodology shows that the design methodology can lead to designs that are superior to other optimized cores, in terms of both energy consumption and latency. Further, if area is also a constraint, that can be taken into account as well. In fact, the usefulness of the high-level model is limited only by the amount of detail that the designer puts into it.

For the design described, results for latency are shown using high-level estimations while results for energy consumption are given using both high-level estimation and low-level simulation. High-level estimation and low-level simulation show that the linear array design is better than the Xilinx design in terms of both energy consumption and latency. However, the linear array design requires more area. Additionally, we have presented our method for generating input data for low-level simulation and analyzing the energy consumption data that results from this input data.

There are several areas for future work. One area is the investigation how variations in switching activity affect the

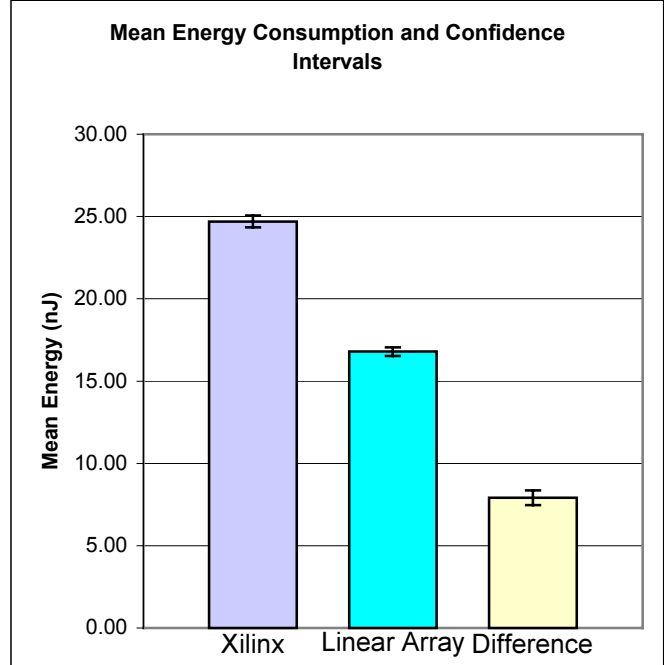


Figure 3: The sample means for each design and the difference between the two means. The error bars at the top of each bar in the graph show the confidence intervals.

performance of our designs compared to the Xilinx design cited. We used random data from the uniform distribution in our simulations, which, as mentioned above, led to a switching activity of ~50%. However, in real-world applications, the data may not be so uniformly distributed, leading to higher or lower switching activities.

We also plan to apply our design methodology to other kernels. We have already begun work on a fast Fourier transform (FFT) design and also plan to tackle matrix inversion and LU decomposition.

Finally, one long term challenge is to develop methodologies for putting together kernel applications to optimally make a complete application. This will require analysis of tradeoffs among kernel application implementations, rather than among particular designs in a domain. Clearly, many exciting challenges lie ahead.

#### ACKNOWLEDGMENTS

This work is part of the Model based Integrated SimuLatioN (MILAN) project, supported by the US DARPA Power Aware Computing and Communication (PAC/C) program under contract F33615-C-00-1633, monitored by Wright Patterson Air Force Base.

#### REFERENCES

1. A. Bogliolo, L. Benini, and G. Micheli. Regression-based RTL Power Modeling. In *ACM Transaction*

- on *Design Automation of Electronic Systems*, Vol. 5, No. 3. 2000.
2. S. Choi, J. Jang, S. Mohanty, and V. K. Prasanna. Domain-Specific Modeling for Rapid System-Wide Energy Estimation of Reconfigurable Architectures, in *Engineering of Reconfigurable Systems and Algorithms*, 2002.
  3. A. Garcia, W. Burlison, and J. L. Danger. Power Modeling in FPGAs. In *Proceedings of the 9<sup>th</sup> International Conference on Field Programmable Logic and Applications*. 1999.
  4. R. V. Hogg and E. A. Tanis, *Probability and Statistical Inference*, Prentice Hall, 2001.
  5. J. Jang, S. Choi, and V. K. Prasanna. Energy-Efficient Matrix Multiplication on FPGAs, in *Field-Programmable Logic and Applications*, volume 2483 of *Lecture Notes in Computer Science (LCNS)*. Springer Verlag, 2002.
  6. S. Mohanty, S. Choi, J. Jang, and V. K. Prasanna. A Model-Based Methodology for Application Specific Energy Efficient Data Path Design Using FPGAs, *The IEEE International Conference on Application-Specific Systems, Architectures, and Processors*. 2002.
  7. T. Mudge. Power: A First-Class Architectural Design Constraint. In *IEEE Computer*, Vol. 34. 2001.
  8. M. Nemani and F. N. Najm. High-level Area and Power Estimation for VLSI Circuits. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*. 1997.
  9. A. Raghunathan, N. K. Jha, and S. Dey. *High-level Power Analysis and Optimization*. Kluwer Academic Publishers, 1988.
  10. L. Shang, A. Kaviani, and K. Bathala. Dynamic Power Consumption in Virtex-II FPGA Family. In *International Symposium on Field Programmable Gate Arrays*, 2002.
  11. A. Stammerman, et. al. System Level Optimization and Design Space Exploration for Low Power. In *Proceedings of ISSS*. 2001.
  12. R. Tessier and W. Burlison. Reconfigurable Computing and Digital Signal Processing: A Survey. *Journal of VLSI Signal Processing*, May/June 2001.
  13. F. G. Wolff, et. al. High-level Low Power FPGA Design Methodology. National Aerospace and Electronics Conference, 2000.
  14. The Virtex-II FPGA and the Xilinx ISE are from Xilinx Corporation, 2100 Logic Drive, San Jose, CA.
  15. G. Yeap. *Practical Low Power Digital VLSI Design*. Kluwer Academic Publishers, 1998.