The Journal of Supercomputing, 26, 259–281, 2003

Domain-Specific Modeling for Rapid Energy Estimation of Reconfigurable Architectures

SEONIL CHOI seonilch@usc.edu
Department of Electrical Engineering, University of Southern California, Los Angeles, CA 90089, USA

JU-WOOK JANG jjang@sogang.ac.kr
Department of Electronic Engineering, Sogang University, Seoul, Korea

SUMIT MOHANTY smohanty@usc.edu
Department of Electrical Engineering, University of Southern California, Los Angeles, CA 90089, USA

VIKTOR K. PRASANNA prasanna@usc.edu
Department of Electrical Engineering, University of Southern California, Los Angeles, CA 90089, USA

Abstract. Reconfigurable architectures such as FPGAs are flexible alternatives to DSPs or ASICs used in mobile devices for which energy is a key performance metric. Reconfigurable architectures offer several design parameters such as operating frequency, precision, amount of memory, degree of parallelism, etc. These parameters define a large design space that must be explored to find energy-efficient solutions. It is also challenging to predict the energy variation at the early design phases when a design is modified at algorithm level. Efficient traversal of such a large design space requires high-level modeling to facilitate rapid estimation of system-wide energy. However, FPGAs do not exhibit a high-level structure like, for example, a RISC processor for which high-level as well as low-level energy models are available. To address this scenario, we propose a domain-specific modeling technique for energy-efficient kernel design that exploits the knowledge of the algorithm and the target architecture family for a given kernel to develop a high-level model. This model captures architecture and algorithm features, parameters affecting energy performance, and power estimation functions based on these parameters. A system-wide energy function is derived based on the power functions and cycle specific power state of each building block of the architecture. This model is used to understand the impact of various parameters on system-wide energy and can be a basis for the design of energy-efficient algorithms. Our high-level model is used to quickly obtain fairly accurate estimate of the system-wide energy dissipation of data paths configured using FPGAs. We demonstrate our modeling methodology by applying it to four domains.

Keywords: domain-specific modeling, energy estimation, energy optimization, FPGA

1. Introduction

Dramatic increase in the density and speed of FPGAs makes them attractive for complex applications. The state-of-the-art Virtex-II Pro FPGA from Xilinx has multi-million gates and delivers over 0.3 Tera MACs/sec. at an operating frequency of 300 MHz [22]. With such an available processing power, FPGAs are an attractive

fabric for implementing complex and compute intensive applications such as signal processing kernels for mobile devices [9, 17] (Styles and Luk, 2000). Mobile devices operate in power constrained environments. Therefore, in addition to time performance, energy performance is a key performance metric [12]. Studies show that optimization at the algorithmic level has a much higher impact on total energy dissipation of a system than RTL or gate level. It is reported that the impact (on energy optimization) ratio is 20:2.5:1 for algorithmic, register, and circuit level [16]. In this context, there is a need for a high-level energy model which not only enables algorithmic level optimizations but also provides rapid and reasonably accurate energy estimates.

Several issues must be addressed in developing a high-level energy model for FPGAs. There are numerous ways to map an algorithm onto an FPGA as opposed to mapping onto a traditional processor such as a RISC processor or a DSP, for which the architecture and the components such as ALU, data path, memory, etc. are well defined. For FPGAs, the basic element is the lookup table (LUT), which is too low-level an entity to be considered for high-level modeling. Besides, the architecture design depends heavily on the algorithm. Therefore, no single high-level model can capture the energy behavior of all feasible designs implemented on FPGAs. In addition, to elevate the level of abstraction, high-level models do not capture all the details of a system and consider only a small set of key parameters that affect energy. This lowers the accuracy of energy estimation.

In order to address the issues discussed above, we propose a *domain-specific modeling* technique (Figure 1). This technique facilitates high-level energy modeling for a specific domain. A domain corresponds to a family of architectures and algorithms that implements a given kernel. For example, a set of algorithms implementing matrix multiplication on a linear array is a domain. Detailed knowledge of the domain is exploited to identify the architecture parameters for the analysis of the energy dissipation of the resulting designs in the domain. By restricting our modeling to a specific domain, we reduce the number of architecture parameters and their ranges, thereby significantly reducing the design space. A limited number of architecture parameters also facilitate development of power functions that estimate the power dissipated by each component (a building block of a design). For a specific design, the component specific power functions, parameter values associated with the design, and the cycle specific power state of each component are combined to specify a system-wide energy function.

Our approach is a top-down approach in contrast with other approaches that exploit low-level simulations and estimations for each component and accumulate these results to estimate overall energy dissipation. The advantage of our approach is the ability to rapidly evaluate the system-wide energy using energy function for different designs within a domain. Our high-level energy model also facilitates algorithmic level energy optimization through identification of appropriate values for architecture parameters such as frequency, number of components, precision, etc., early in system design.

The organization of the paper is as follows. Related efforts are discussed in Section 2. Section 3 describes the domain-specific modeling technique and the methodology to estimate the power functions. A detailed description of modeling and energy

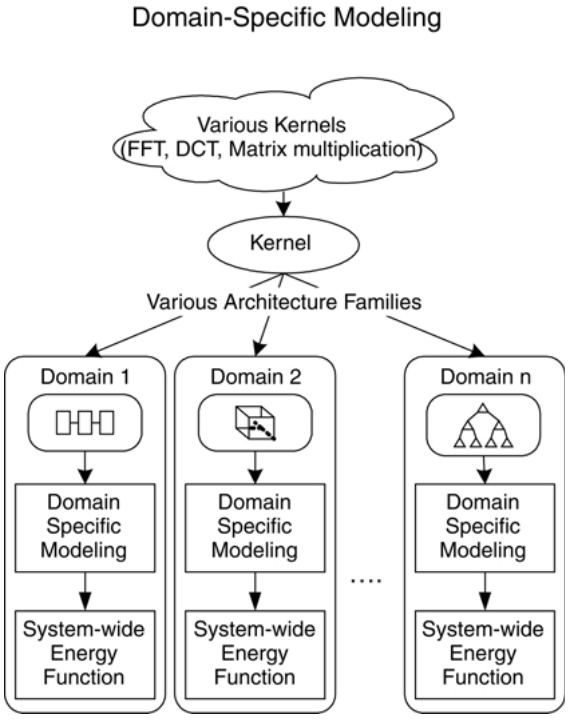

Figure 1. Domain-specific modeling.

estimation using domain-specific modeling for four different domains is presented in Section 4. We discuss some applications of the domain-specific modeling in Section 5. Section 6 concludes the paper.

2. Related work

Several research efforts have focused on rapid energy estimation of a design on FPGAs. Shang and Jha [18] proposed a black-box approach to estimate energy based on input and output signal statistics. This approach is suitable for estimation of average power dissipation of a RT-level component to be embedded into a system. However, it is not applicable for algorithm level power analysis. On the other hand, our model captures various architecture parameters that can be manipulated at algorithmic level for energy optimization.

XPower, the power estimation tool provided by Xilinx [22], estimates the energy dissipation of FPGAs based on low-level simulation. The input to the tool is LUT-level place-and-route information along with details of switching activity for LUT-level components. While its accuracy is comparable with the actual execution of the design, it does not support energy estimation early in the design phase when the complete system description in some HDL is not available. Stammermann et al. [20]

presented ORINOCO, a software tool for power dissipation analysis and optimization at the algorithmic level from C/C++ and VHDL description. However, C/C++ or VHDL descriptions do not capture parameters affecting system-wide energy and also a designer requires a complete knowledge of the final system before the code can be generated in these languages. Both ORINOCO and XPower are essentially estimation tools and can be used in our methodology to perform low-level sample simulations necessary for specifying our component specific power functions. We have compared our estimation accuracy against XPower.

In Bogliolo et al. [1] regression tree [2] is used to improve the power estimation of a RT-level component. Starting with candidate variables (I/O bits), the variable v_i, which has the maximum impact on the power dissipation is identified. Then the sample power dissipation results of power measurement is split in two subsets based on this variable. The splitting is recursively performed to build a regression tree which ranks variables in their significance with respect to the power. It is a bottom-up approach starting from low-level implementation and ends in identifying significant variables affecting the power.

In contrast, our model starts with candidate parameters chosen from a high-level view of the architecture and algorithm. The effect of the parameters on the system-wide energy is captured in the component specific power functions. The component specific power functions are used to obtain parameter values for optimal power performance by traversing the design space at an algorithmic level.

3. Domain-specific energy modeling

Since FPGAs provide the freedom to map various architectures, choosing an appropriate architecture plays a significant role in determining the amount of interconnect and logic to be used in the design which also affects energy dissipation, latency, and area. Therefore, based on the performance needs and the characteristic of the target FPGA chip, it is possible to identify a set of suitable architectures, each having different characteristics in terms of I/O complexity, memory requirements, area, etc. Defining a domain which consists 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 1D array (linear array) or a 2D array. A 2D array would dissipate more power from interconnect since more interconnects are required. Thus, it is possible that more energy would be dissipated, depending upon the resulting latency. The parameters representing algorithm level and architecture level choices for a specific application form a multi-dimensional space. For example, the number of multipliers, registers and the I/O channels can be changed from algorithm level choices for matrix multiplication.

In the course of high-level modeling, we consider many power management techniques that provide control knobs when applied to designing for FPGAs [19, 23]. 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 [22]. BUFGCE can be used for dynamically driving a clock tree only when the corresponding logic is used. 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 blocks. Choosing *bindings* is another technique. A binding is a mapping of a computation 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, there are three possible bindings for storage elements in Virtex-II devices based on the number of entries: registers, slice based RAM (SRAM), and embedded Block RAM (BRAM). 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. In high-level modeling, we can analyze the trade-offs that arise from various bindings based on the design requirements.

Exploiting the domain knowledge and the power management techniques, the goal of domain-specific modeling (Figure 2(a)) is to represent energy dissipation of the designs specific to a domain in terms of parameters associated with this domain. For a given domain, only those parameters which can significantly affect system-wide energy dissipation and can be varied at algorithmic level are chosen for the high-level energy model. As a result, our model (a) facilitates algorithmic

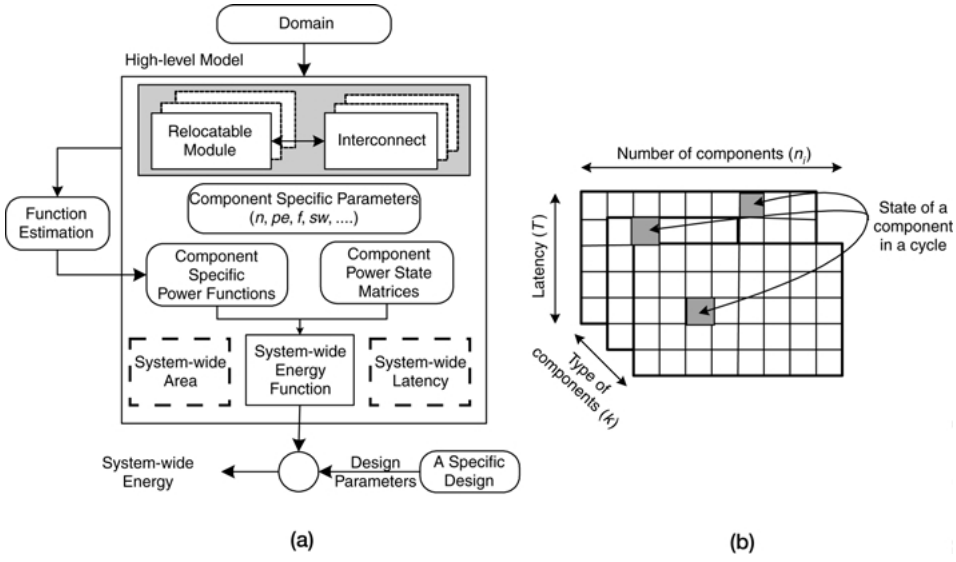

Figure 2. (a) Domain-specific modeling and system-wide energy estimation and (b) component power state matrices.

level optimization of energy performance, (b) provides rapid and fairly accurate estimates of the energy performance, and (c) provides energy distribution profile for individual components to identify candidates for further optimization. First, we define the high-level energy model. Then we provide details of energy estimation method using this model.

3.1. *High-level energy model*

Our high-level energy model consists of *RModules, Interconnects, component specific parameters and power functions, component power state matrices, and a system-wide energy function*.

Relocatable module (RModule) is a high-level architecture abstraction of a computation or storage module. It is either a CLB (configurable logic block)-based logic or a ''larger'' module composed of multiple RModules and Interconnects. We define RModule whose power dissipation can be individually characterized once their input stimuli are known, regardless of their location. For example, a register can be a RModule if the number of registers varies in the design depending on algorithmic level choices. One important assumption about RModule is that energy performance of an instance of a RModule is independent of its location on the device. While this assumption can introduce small error in energy estimation, it greatly simplifies the model. We regard RModules as building blocks which are used to construct the energy model. The granularity of RModules for a specific domain are influenced by the domain. For example, the adders or registers inside the multiplier can be RModules. But there is no sense choosing them since there are no corresponding parameters for them in the domain. They are not targets for energy optimization at algorithm level. Interconnect represents the connection resources used for data transfer between the RModules. The power dissipated in a given Interconnect depends on its length, width, and switching activity. Interconnect can be of various types. For example, in Virtex-II FPGAs, there are several Interconnect types such as long lines, hex lines, double lines, and single connections which differ in their lengths [22]. In the rest of the paper, we use component to refer to both RModule and Interconnect.

Component specific parameters depend on the characteristics of the component and its relationship to the algorithm. We choose those parameters which may significantly affect the total energy using knowledge of application, algorithm and architecture and model the domain using the chosen parameters. For example, we model the domain for the matrix multiplication using the number of multipliers and registers since power dissipation in these components significantly affects the total energy (Section 4.2). From our knowledge in the algorithm, we find that there exists frequent systolic movement of intermediate results among them. Another examples are operating frequency and precision of a multiplier RModule if they are varied by the algorithm. Possible candidates parameters include operating frequency (f), input switching activity (sw), word precision (w), power states (ps), number of RModule type i (n_i), etc. Component specific power functions capture the effect of component specific parameters on the average power dissipation of the component. The power

functions are obtained by implementing sample designs of individual components and simulating them using low-level simulators (described later in this section).

Component power state (CPS) matrices capture the power state for all the components in each cycle. For example, consider a design that contains k different types of components $(C_1, \ldots, C_k)$ with n_i components of type i. If the design has the latency of T cycles, then k 2D matrices are constructed where the i-th matrix is of size $T \times n_i$ (Figure 2(b)). An entry in a CPS matrix represents the power state of a component during a specific cycle and is determined by the algorithm.

System-wide energy function represents the energy dissipation of the designs belonging to a specific domain as a function of the parameters associated with the domain.

The domain-specific nature of our energy modeling is exploited when the designer identifies the level of architecture abstraction (RModules and Interconnects) appropriate to the domain and/or chooses the parameters to be used in the component-specific power functions. This is a human-in-the-loop process and exploits the designer's expertize in the algorithm and the architecture family that constitutes the domain. Well-known power models based on capacitance, voltage, and switching activity can be more accurate and are generic to be applicable across many domains. However, they do not provide a designer a clear understanding of the impact of his/her algorithmic level design choices on the energy performance. Our modeling enables the designer to rapidly explore a large design space based on the understanding of the effect of the design choices on the overall energy performance.

To handle modeling complexity we follow a hierarchical approach. Each RModule can be recursively divided into RModules and Interconnects. This hierarchical nature allows the designer to capture the details of architecture in the design at various levels of abstraction to identify parameters affecting performance.

3.2. *Component specific power function estimation*

Power dissipation by a RModule or Interconnect in a particular state is captured as a power function of a set of parameters. These functions are typically constructed through curve fitting based on some sample low-level simulations. We demonstrate our function estimation technique in detail by deriving the power function for a register-based memory implemented on the Xilinx Virtex-II device. Figure 3(a) summarizes the technique. This technique was applied for power function estimation during the modeling of the various domains described in Section 4.

Let $C_{\_p}(p_1, \ldots, p_n)$ be the component power function and $p_1, \ldots, p_n$ be the parameters associated with the component. Estimation of the component-specific power function involves estimation of power dissipation through low-level simulation of the component at different design points. A design point is a unique combination of parameter values. For our chosen component, a register based memory, we used the basic register design provided by the Xilinx library. The component specific parameters are frequency of operation, number of registers in a memory, and the precision. We decided not to vary the precision and assumed it to

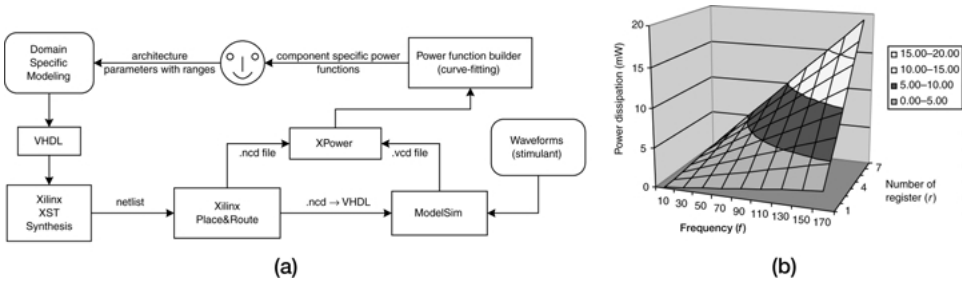

Figure 3. (a) Power function estimation and (b) register power function as a function of the number of registers (r) and frequency (f).

be 8-bit. Therefore the parameters that affect energy dissipation of the memory are number of registers (r) and frequency (f). Let (r, f) denote a design point. We identified the candidate designs randomly (for low-level simulation) to be the combinations of $r = 1, 4, 8$ and $f = 10, 50, 150$ MHz.

The designer associates a VHDL implementation with each RModule. These VHDL implementations are parameterized based on the parameters supported by the associated RModule. Low-level simulation is performed at each of the chosen design points to estimate the power dissipation at that design point. We use random input vectors since there is no general purpose technique to predict exactly what data is available as input to a component. However, we have developed a technique based on statistical analysis to obtain reasonably accurate average estimate of power dissipation of a design [8]. 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 mean found from a sample out of the population. 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 Hogg and Tanis [6], s is the population standard deviation, and M is the number of samples. We assume that the distribution from which the results come is not too badly skewed or discrete.

For example, to statistically analyze energy dissipation for the matrix multiplication in Section 4.2, we performed 50 different $n \times n$ matrix multiplication trials for our linear architecture. Each trial consists of performing the low-level simulation procedure, as described above, with the uniformly distributed, randomly generated matrices as input.

These power estimates and the design points are provided as inputs to the power function builder. For components with a single parameter, the power function are obtained using curve-fitting on sample simulation results. In case of more number of the parameters, surface fitting can be used. Currently, we only focus on building component power functions with at most two parameters. The resulting power functions are provided to the designer. We have used Microsoft Excel for power

function estimation. Figure 3(b) shows the graph based on sample simulation results of different design of the register based memory. The power function, based on the graph is $R._p(r,f) = 0.0142 \cdot r \cdot f + 0.0011$.

The component power function of an interconnect depends on its length, operating frequency, and the switching activity. Unfortunately, estimating the interconnect length requires the knowledge of the placement of the physical implementation of the components. In Xilinx Virtex-II device, various routing resources can be identified based on long, hex, double and direct wires. After performing synthesis and simulation, we can obtain the number of different wires used from a XDL file which is the text version of place and routed circuit description (.ncd file) [22]. The power function of an Interconnect component is $I._p(L,w) = (1/2)V^2 \cdot f \cdot sw \cdot (C_l \cdot l + C_h \cdot h + C_{db} \cdot db + C_{dr} \cdot dr)$ where V is voltage, f is the operating frequency, sw is the average switching activity, L is the length of an interconnect, w is the precision (or width), and C_l, C_h, C_{db}, C_{dr} and l, h, db, dr are the average capacitance and number of long, hex, double, and direct wires respectively [19].

However, since the architectures we are currently consider has neighboring connections, we use a simplified approach. We use Equation (1) to estimate power dissipation in an interconnect. $\Phi._p$ denotes the power dissipation of a cluster of k RModules connected through the candidate interconnect and $M \cdot p_i$ represents power dissipation of the i-th RModule. The power dissipated by the cluster and RModules are obtained by low-level simulation:

$$IC._p = \Phi._p - \sum_{i=1}^{k} M._{p_i}. \tag{1}$$

The low-level simulation is performed as follows. The sample VHDL design is synthesized using XST (Xilinx Synthesis Technology) on Xilinx ISE 4.1i. The place-and-route file (.ncd file) is obtained for the target FPGA device using PAR. Mentor ModelSim 5.5e is used to simulate the module and generate simulation results (.vcd file). These two files are then provided to the Xilinx XPower tool to estimate the energy dissipation. Above simulation technique was also applied to the candidate designs to estimate power which was multiplied with latency to obtain the measured energy estimates shown in Tables 3 and 4.

While the initial effort to build the component power function might be expensive, the benefits are noticeable when the same components are re-used in different designs within and (possibly) across domains.

3.3. *Deriving system-wide energy function*

The CPS matrices capture the operating state of each component for every cycle and the power functions provide the power estimate for each state. Therefore, the total energy of the complete system is obtained by summing the energy dissipation of

individual components in each cycle. The system-wide energy function SE is obtained as:

$$SE = \sum_{i=1}^{k} \frac{1}{f} \left(\sum_{j=1}^{n_l} \sum_{t=1}^{T} C_{i \cdot p \cdot ps} \right) \quad \text{where } ps = CPS(i,t,j), \tag{2}$$

$C_{i \cdot p \cdot ps}$ is the power dissipated in the j-th component ($j = 1, \ldots, n_l$) of type i during cycle t ($t = 1, \ldots, T$) and f is the operating frequency. $CPS(i,t,j)$ is the power state of the j-th component of the i-th type during the t-th cycle. Many state-of-the-art FPGAs feature multiple clock domains. However, we focus on the design of signal processing kernels which typically perform an atomic task such as FFT, DCT, filters, etc. Therefore, we consider a single clock frequency for the complete kernel design.

Since the system-wide energy function is derived using component specific power functions, the energy distribution among various components (the fraction of the total energy dissipated by each component) can be obtained. This information is used to identify candidate components to be considered by the designer for energy optimization. Details can be found in Section 5.

Due to the high-level nature of the model, we can rapidly estimate the system-wide energy. In the worst case, the complexity of energy estimation is $O\left(T \times \sum_{l=1}^{k} n_i\right)$ (see Equation (2)) which corresponds to iterating over the elements of the CPS matrices and adding the energy dissipation by each component in each cycle. However, typically, there is a repeating pattern of state changes for a component (for example, due to loop structures within the algorithms). Also, different components of the same type dissipate the same amount of energy during each cycle. Therefore, based on these observations the time to compute the energy is better than the worst case complexity of energy estimation stated above. Further, even if we compute the system-wide energy based on each cycle we do not analyze the activities at the level of individual gates. Typically, there are only a few distinct components within a domain that affect energy dissipation of the designs in that domain. Indeed, for the illustrative examples considered in this paper, the time for energy estimation does not depend on the problem size.

The time needed to perform high-level estimation (assuming the power functions are pre-computed) is on the order of minutes on a Pentium III Xeon running at 700 MHz, whereas the time needed for low-level simulation and power estimation was 3–24 hours per design on the same machine. For the domains discussed in this paper, we typically need 4–8 low-level simulations (one for each design point) for each power function. Once all power functions are computed and the system-wide energy function is derived, they are applied to the complete design space. For Domain 2 (Section 4.2), the number of low-level simulations performed to define the domain-specific models were approximately 30. As these simulations are for a component not the complete design each low-level simulation takes approximately 30 to 60 minutes. The model is applicable to all the design of $n \times n$ matrix multiplication where $1 \leq n \leq 48$ (we chose 20 designs). Therefore, our effort approximately takes 10–12 hours of simulation and computation which is very small when compared with approximately 2 weeks needed to simulate 20 designs.

4. Illustrative examples of domain-specific modeling

To illustrate our domain-specific modeling methodology, we apply the techniques discussed in the previous section to define high-level models for four different domains implementing matrix multiplication and fast Fourier transforms (FFT), two frequently used kernel operations in wide variety of signal processing algorithms (ElGindy and Shue, 2002). For each domain, we identify the components and the component specific parameters, evaluate the power functions for each component, and finally derive a system-wide energy function. Three architecture families, a uniprocessor architecture, a homogeneous linear array architecture, and a heterogeneous linear pipelined architecture are chosen to demonstrate our approach. We have chosen the Xilinx Virtex-II FPGA (XC2V1500, speed grade-5) as our target device.

4.1. Domain 1: Uniprocessor architecture

We define a uniprocessor (PE) implementing the "usual" block matrix multiplication (BMM) as the first domain. This domain uses a single multiplier and results in compact energy-efficient designs. There are two possible scenarios: on-chip design and off-chip design. If the matrix multiplication kernel is a stand-alone application, all matrix data are stored in an external memory outside the FPGA. We refer to such a design as off-chip design. If the matrix multiplication kernel is one of many kernels in an application, it is desirable to have the matrix data reside (in a Block SRAM) on the device. We refer to such a design as on-chip design. The Block SRAM is a dedicated on-chip memory in Virtex-II and is usually used for storing intermediate data between (kernel) computations. Figure 4 shows our target architectures.

In the off-chip design, the PE has one MAC (multiplier and accumulator), a cache (local buffer) of size c, and I/O ports (see Figure 4(a)). Each word of cache is three 8-bit registers. The data matrices are stored in an external memory. For $n \times n$ matrix multiplication, the computational complexity of the algorithm is $O(n^3)$ [7]. Block matrix multiplication is performed with block size $\sqrt{c} \times \sqrt{c}$. The I/O complexity (amount of traffic between the PE and external memory) is $O(n^3/\sqrt{c})$. It can be

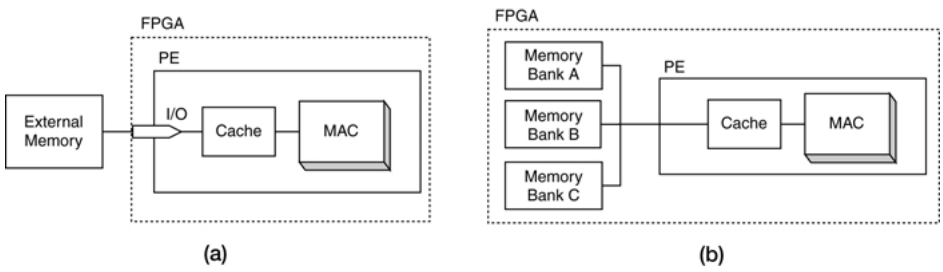

Figure 4. Uniprocessor architecture: (a) off-chip design and (b) on-chip design.

observed that a large cache decreases the I/O traffic and as a result improves the energy dissipation in performing I/O.

In the on-chip design, the PE has one MAC, a cache of size c, and three memory banks for storing three matrices (see Figure 4(b)). BMM is performed with block size $\sqrt{c} \times \sqrt{c}$. The energy for I/O (outside the device) is not included, but the energy dissipated in the three memory banks is considered. The read/write access frequency of the memory banks depends on the traffic between the memory banks and the PE. It can be observed that as the cache size increases, the number of memory bank accesses decreases and as a result the energy dissipated in the memory banks reduces.

4.1.1. Defining components and parameters. We identified four components: MAC, cache, and the memory banks as RModules, the I/O as an Interconnect. The RModules have w bit precision. We assumed the precision of input data to be 8 bits and the precision of the intermediate and the output data to be 16 bits. Therefore, the cache size (c) is the only parameter that can be varied at design time.

The component specific power functions for MAC ($MAC_{.p}$), cache ($R_{.p}$), I/O ($IO_{.p}$), and memory bank ($MEM_{.p}$) were obtained through low-level simulation using the method described in Section 3.

To implement the MAC in Virtex-II, there are two design choices: a CLB-based multiplier and a dedicated multiplier. A dedicated multiplier is a stand-alone ASIC-based multiplier. A CLB-based multiplier is built using CLBs and it was observed that it dissipates more power than a dedicated multiplier. Similarly, there are two design choices for implementing the cache using CLBs. If the cache size is small, the cache can be realized using CLBs configured as register modules. Larger cache can be realized using CLBs configured as SRAM [22]. However, a SRAM-based cache can only be configured to be a multiple of 16 bytes. We noticed that for $c > 6$, the SRAM-based cache dissipates less power than the register-based cache of the same size. The caches for matrix A and B has 8 bit precision and for matrix C, 16 bit precision. $MAC_{.p}$ and $IO_{.p}$ are constants. The power function for the register-based cache is: $R_{.p}(c) = 2.12c$ (mW) for 8 bit precision. The power function for 16 bit register is $2R_{.p}(c)$. The power function for the SRAM-based cache is: $R_{.p}(c) = 0.12(\lceil c/16 \rceil)^2 + 4.52\lceil c/16 \rceil + 7.81$ (mW). The memory bank is implemented using Block SRAM in Virtex-II. The power state of Block SRAM can be controled by gated clocking. However, there is not much difference ($< 2\%$) in the power dissipation between the on and off states. Instead, the power dissipated in Block SRAM depends mainly on the access frequency (fm) of the memory bank. A Block SRAM can be configured to be a multiple of 2 K bytes with 8 bit precision. The power function for 2 K byte Block SRAM is: $MEM_{.p}(fm) = 2.89fm^2 + 25.79fm + 0.29$ (mW).

4.1.2. System-wide energy function. We now consider the system-wide energy dissipated by the design. In both the on-chip and off-chip designs, the amount of computation performed by the MACs is the same and the MACs dissipate the same amount of energy. For the off-chip design, we do not consider the energy dissipated in the external memory. The system-wide energy function (SE) for performing $n \times n$

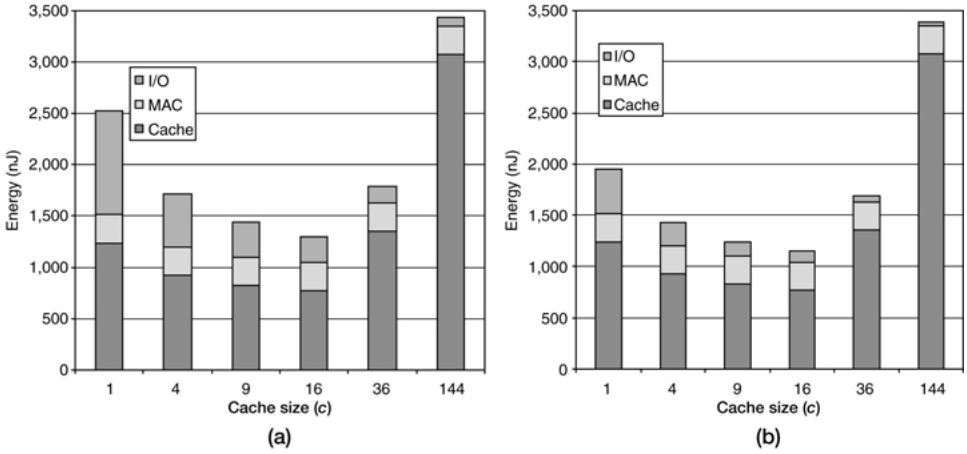

Figure 5. System-wide energy dissipation and energy distribution for 12×12 matrix multiplication as a function of cache size: (a) off-chip design and (b) on-chip design.

matrix multiplication is:

$$SE(n,c) = \frac{1}{f}\left(n^3 MAC_{.p} + 4\left(n^3 + \frac{n^3}{\sqrt{c}}\right)R_{.p}(c) + 3\left(\frac{n^3}{\sqrt{c}}\right)IO_{.p}\right). \quad (3)$$

Note that as c varies, we obtain a family of architectures each implementing matrix multiplication using BMM with different block sizes. The operating frequency of our design was set to 166 MHz. Figure 5 shows how different values of c affect the system-wide energy dissipation and the energy distribution among the components of the design for 12×12 matrix multiplication. As c increases, the energy for performing I/O decreases but the energy dissipated in the cache increases. Initially, the system-wide energy decreases as c increases but for large values of c, the system-wide energy increases.

For the on-chip design, the energy dissipated in the memory banks is considered instead of the energy dissipated in the I/O. The system-wide energy function is:

$$SE(n,c) = \frac{1}{f}\left(n^3 MAC_{.p} + 4\left(n^3 + \frac{n^3}{\sqrt{c}}\right)R_{.p}(c) + 3\left(\frac{n^3}{\sqrt{c}}\right)MEM_{.p}\right). \quad (4)$$

Note that as c increases, the traffic between the memory banks and the PE decreases and as a result the energy dissipated in the memory banks decreases.

4.1.3. Design trade-offs and performance analysis. As the system-wide energy function is a well-behaved function with easily determinable minima, we were able to identify the most energy-efficient designs from the trade-off graphs (see Figure 5). For both designs, the cache size $c = 16$ gives the minimum system-wide energy. Since I/O operations are expensive, using larger cache helps to reduce the energy for off-

chip design. However, as the cache size increases, its energy dissipation becomes dominant and the system-wide energy increases. For the on-chip design, the energy for Block SRAM is not as significant as the energy for I/O. All designs use a single dedicated multiplier.

4.2. *Domain 2: Linear array architecture*

For the second domain, we consider a linear array of processing elements (PEs) as the candidate architecture (see Figure 6(a)). Each PE has one multiplier and storage. We start with an algorithm for optimal latency on linear array [14]. PE_j in Figure 6(a) computes $c_{ij} = \sum_{k=1}^{n} a_{ik} \times b_{kj}$ for all $i, 1 \leq i \leq n$ where a_{ik}, b_{kj}, and c_{ij} represent an element of $n \times n$ matrices A, B, and C. In iteration $(i,k), 1 \leq i,k \leq n, c_{ij} = c_{ij} + a_{ik} \times b_{kj}$ is computed in PE_j. Elements of matrix A and B are fed to the array via two input ports of PE_1 in column major and row major order, respectively. It is critical to ensure that a_{ik} "meets" b_{kj} in a cycle in PE_j. For this, a_{ik} and b_{kj} pass through two and one delay(s), respectively in each PE. The resulting architecture for each PE is shown in Figure 6(b). a_{ik} enters input port AS and goes through two delays ($AS.LR$ and $AS.RR$), while b_{kj} enters BS and goes through one delay. Details of the algorithm, its analysis, and proof of correctness can be found in Prasanna and Tsai [14].

Compared with the uniprocessor design in Section 4.1, we use more multipliers to reduce the latency. The above family of architectures offers several advantages compared to other architecture families. These architectures have a low I/O-bandwidth requirement and they scale as the problem size grows. To achieve the minimal I/O complexity $(O(n^2))$, the total amount of storage across all the PEs should be n^2. As shown in Prasanna and Tsai [14], this architecture can perform $n \times n$ matrix multiplication in $O(n^2)$ time using $n\lceil n/s \rceil$ PEs. For the sake of illustration, we consider the on-chip design for this domain.

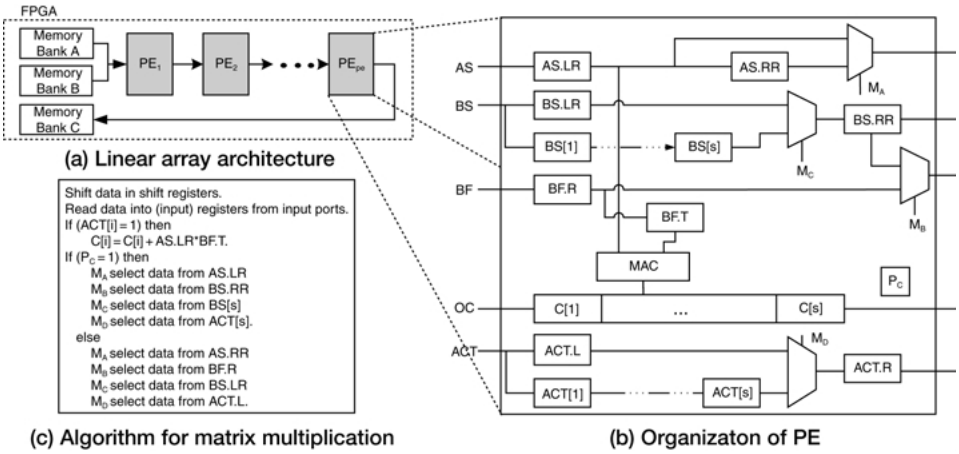

Figure 6. Architecture for matrix multiplication, PE organization, and corresponding algorithm.

4.2.1. Defining components and parameters. The structure of the linear array is shown in Figure 6(a). It consists of three components: processing elements (PEs), buses connecting adjacent PEs, and memory banks. For the purpose of high-level modeling, we identified the PE and the memory bank as RModules, and the bus between two adjacent PEs as an Interconnect. The PE has a MAC of precision w and storage of size s (see Figure 6(b)). The MAC is implemented using a dedicated multiplier. The PE has two power states *on* and *off*. In the *on* state, the multiplier is on and thus the PE dissipates more power than in the *off* state when the multiplier is off. The power state of the multiplier is controlled by clock gating. The PE also includes six registers and three multiplexers of w bits. The key parameters affecting energy are the number of PEs (pe), the amount of storage within a PE (s), and power states (ps).

We implemented the PE using a Virtex-II FPGA operating at $f = 166\,\text{MHz}$ and performed simulations to obtain the power functions for the PE and the bus. The power function for the PE is:

$$PE_{.p\cdot ps} = \begin{cases} 7.01s + 31.04\,\text{mW} & (ps = on) \\ 7.01s + 14.04\,\text{mW} & (ps = off) \end{cases}. \tag{5}$$

The interconnect power function is constant. It is estimated using Equation (1) since the interconnect between the PEs is localized in the design and is regular. We implemented two PEs and the interconnect, and measured the power dissipation while both PEs are in ON state. The power dissipated in the interconnect is $IC_{.p} = 39.74\,\text{mW}$. The power function for the memory bank is the same as in the uniprocessor architecture (see Section 4.1.1).

We consider the problems of size $1 \leq n \leq 16$. For the sake of illustration, we fixed w at 8. The parameters and their ranges are shown in Table 1. Note that the parameters of interest are pe, ps, and s. The system-wide energy function is specified using these three parameters.

4.2.2. System-wide energy function. There are several constraints imposed by the algorithm which is exploited to identify component specific parameters and their ranges. The value of s determines the total number of PEs (pe). The latency (T) of this design using $n\lceil n/s\rceil$ PEs and s storage per PE [14] is: $T = (n^2 + 2n\lceil n/s\rceil - \lceil n/s\rceil + 1)$.

We consider problems in the range $1 \leq n \leq 16$. Precision (w) is set to 8. In each PE, the multiplier is on for $T/(\lceil n/s\rceil)$ cycles and is off for $T \times (1 - 1/\lceil n/s\rceil)$ cycles. $PE_{.p\cdot ps}$ refers to the power dissipation of PE when its multiplier is in state ps (see Equation (5)). Note that the I/O traffic between the PEs and the memory banks is

Table 1. Model parameters

Parameters	s	pe	w	ps
Values or ranges	$1 \leq s \leq n$	$1 \leq pe \leq n\lceil n/s\rceil$	8	*on*, *off*

$O(n^2)$. The system-wide energy function is:

$$SE(n,s) = \frac{1}{f}\Big(n \cdot T \cdot PE_{.p \cdot ps=on} + T \cdot (n\left\lceil\frac{n}{s}\right\rceil - n) \cdot PE_{.p \cdot ps=off} + T \cdot (n\left\lceil\frac{n}{s}\right\rceil - 1) \cdot IC_{.p} + 3n^2 \cdot MEM_{.p}\Big) \quad (6)$$

4.2.3. Design trade-offs and performance analysis. Figure 7(a) shows the effect of varying the amount of storage (s) on the power dissipation of a PE. Figure 7(b) shows the effect of varying the amount of storage (s) on the system-wide energy for three problem sizes ($n = 4, 8, 16$). Based on these plots, to obtain energy-efficient designs we choose $pe = s = n$, where n is the problem size.

Table 2 shows the energy, latency, and area of the designs for various problem sizes. We compared the performance of our design with a design for 3×3 matrix multiplication provided by Xilinx [22]. Since Xilinx library does not provide on-chip design, we added Block SRAMs to the Xilinx design. All Xilinx designs execute at 150 MHz. For $n > 3$, we used block matrix multiplication using the 3×3 design. The improvement in energy dissipation and latency in our designs compared with the Xilinx designs are also shown. On the average our designs dissipate 32% less energy compared with the Xilinx design. The latency improvement varies from $5.8\times$ to $17.3\times$. However, our designs occupy more area.

The energy dissipation of the designs discussed in this section is based on high-level estimation using the system-wide energy function for the domain. In order to validate these energy estimations, we performed the following experiment. For a particular design, we used the corresponding system-wide energy function to estimate the total energy dissipation. We compared this result with a complete VHDL simulation of the design using Xilinx tools described in Section 3.2. In the simulations, the same input data used to obtain the component specific power functions were used. As noted earlier, the average switching activity was observed to

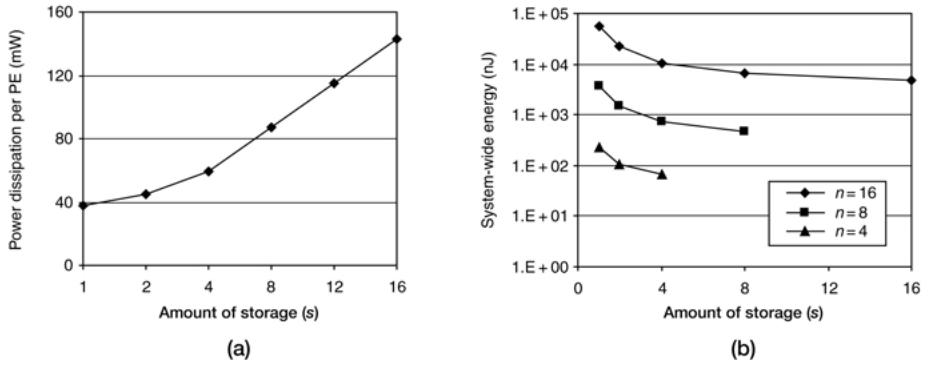

Figure 7. (a) Power dissipation for a single PE and (b) the system-wide energy as a function of the amount of storage (s) for $n = 4, 8, 16$.

Table 2. Comparison of our designs on linear array architecture with Xilinx design

Matrix size	Design based on Xilinx library			Design based on linear array architecture			Performance improvement	
$n \times n$	T (usec)	A (slices)	E (nJ)	T (usec)	A (slices)	E (nJ)	E (%)	T (times)
6×6	1.71	251	292.40	0.30	1,074	213.2	37	5.8
9×9	5.76	251	986.86	0.30	1,935	715.5	38	9.6
15×15	26.67	251	4,568.80	1.54	4,305	3,812.5	20	17.3

Table 3. Accuracy of the high-level energy estimation of our designs

Matrix size ($n \times n$)	3×3	6×6	8×8	9×9	12×12	16×16
Estimated energy (nJ)	34.0	213.2	497.8	715.5	1,801.2	4,759.7
Measured energy (nJ)	37.4	228.6	536.9	768.4	1,913.6	5,078.6
Error	9.0%	6.7%	7.3%	6.9%	5.9%	6.3%

be 50%. We performed this experiment for various problem sizes using designs in Section 4.2.2. Table 3 also shows the error percentage of our high-level estimation method when compared with energy estimation values obtained through low-level simulation. The error percentages are below 9.0%.

4.3. Domain 3: Block matrix multiplication on linear array architecture

The third domain targets large size ($n > 12$) matrix multiplications. It consists of block matrix multiplication (BMM) and the linear array architecture presented in Domain 2. The BMM algorithm for $N \times N$ matrices repeatedly uses the design (hardware) for sub-matrix multiplication of size $n \times n$, where N is a multiple of n. In this domain, we have considered the off-chip design.

4.3.1. Defining components, parameters, and system-wide energy function. All components defined in Domain 2 are also applicable to this domain. An additional parameter is the block size (n). For a blocksize of n, we chose the designs with $pe = s = n$ and $ps = on$ to implement $n \times n$ matrix multiplication. Based on our performance trade-off analysis of Domain 2 (Figure 7) these designs are the most efficient ones in terms of latency and energy dissipation for $n \times n$ matrix multiplication. Since $N \times N$ matrix is divided into $n \times n$ sub-matrices, $(N/n)^3$ block matrix multiplications are performed. Therefore, the latency is: $T = (N/n)^3 \times (n^2 + 2n)/f$. Therefore, the system-wide energy function is:

$$SE(N, n) = (nT \cdot PE_{.p \cdot ps=on} + (n-1)T \cdot IC_{.p} + 3T \cdot IO_{.p}) \quad (7)$$

Table 4. Performance comparison and accuracy of various designs in Domain 3

		Estimated			Measured		
Matrix size	Block size	T (cycles)	A (Slice)	E (nJ)	T (cycles)	A (Slice)	E (nJ)
24×24	8×8	2,187	1,048	5,271	2,187	1,101	5,491
	12×12	1,352	1,572	4,757	1,352	1,667	4,983
48×48	8×8	17,496	1,048	42,164	17,496	1,101	43,929
	12×12	10,816	1,572	38,053	10,816	1,667	39,867
	16×16	7,803	2,096	36,100	7,803	2,186	37,679

4.3.2. Design trade-offs and performance analysis. We vary the block size (n) to evaluate various trade-offs. Table 4 shows the area, latency, and energy of 24×24 and 48×48 matrix multiplication using various block sizes. Results show that the matrix multiplication for $N = 24$ dissipates least energy when $n = 12$. To verify our result we simulated all the designs that are within 10% of the optimal design in terms of energy dissipation. Through low-level simulation (Table 4) design with $n = 12$ is verified as the most energy-efficient design for 24×24 matrix multiplication. For 48×48 matrix multiplication, the design using 16×16 block matrix multiplication is the most energy-efficient design. Note that, our estimations based on the system-wide energy function are within 10% of the estimation using low-level simulations.

4.4. Domain 4: Fast Fourier transform

Fast Fourier transform (FFT) on a heterogeneous linear pipelined architecture is chosen as the fourth domain. 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 is $O(n \log_2 n)$ [13]. Due to the fact that, in practice, FFTs often process a stream of data, a pipelined architecture has been chosen.

4.4.1. Defining components, parameters, and system-wide energy function. The n-point FFT design is based on the radix-4 algorithm and has three components in the architecture: radix-4, data buffer, and twiddle factor computation (see Figure 8):

- The *radix-4 butterfly block* performs a set of additions and subtractions with 24 adders/subtracters. It takes four inputs and produces four outputs in parallel.

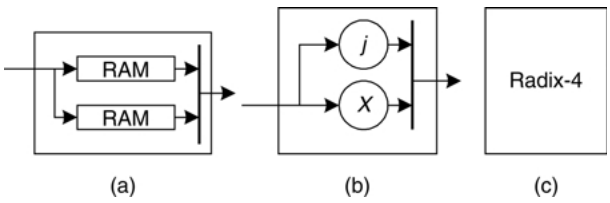

Figure 8. (a) Data buffer (Dbuf), (b) Twiddle factor computation (Twiddle), and (c) Radix-4 computation.

- The *twiddle factor computation block* performs the 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 block contains four multipliers, two adders/subtracters and two sign inverters.
- The *data buffer* consists of two RAMs having n 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 is written into sequential locations and the data is read out from locations at strides of 4. The RAM is implemented using SRAM for $n < 64$ or Block SRAM for $n \geq 64$.

The architecture uses a combination of the above three components. There are three parameters that produce 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 $(1 \leq H_p \leq \log_4 n)$. For example, a 16 point FFT algorithm has two radix-4 stages. In the design, we can use one or two radix-4 blocks $(H_p = 1, 2)$ depending on the sharing of radix-4 block resource. If $H_p = 1$, one radix-4 block 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 9(a)). Vertical parallelism determines how many inputs are computed in parallel. Using the radix-4 block, up to four inputs can be operated on in parallel. Figure 9(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 blocks, and three twiddle computation blocks.

The power functions for the data buffer, the radix-4 block, and the twiddle computation block are $P_{Dbuf}, P_{radix-4}$, and P_{Tw}, respectively, where $P_{Dbuf} = 1.23n + 35.44\,(\text{mW})$ using SRAM, $P_{Dbuf} = 0.0156n + 79.65\,(\text{mW})$ using BRAM, $P_{Radix-4} = 142.84$ (mW), $P_{Tw} = 0.0054n + 183.57$ (mW) using Block SRAM, $P_{Tw} = 0.4879x + 157.74$ (mW) using SRAM, and $P_{IO} = 11.31V_P$ (mW). The latency becomes: $T = (n \log_4 n)/(V_p \times H_p)/f$. The system-wide energy function is

$$E = T(V_p(H_p + 1)P_{Dbuf} + H_p P_{radix-4} + tP_{Tw}), \tag{8}$$

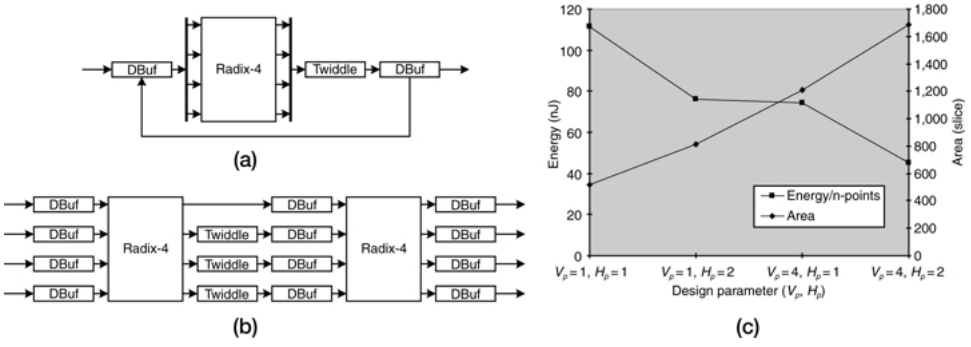

Figure 9. (a), (b) Architectures of 16-point FFT and (c) energy dissipation and area of 16-point FFT for various values of H_p, V_p (100 MHz, 16 bits per data).

Table 5. Energy dissipation for FFT (from low-level simulation)

Problem size (n)	H_p	V_p	T (usec)	Estimated E (nJ)	Measured E (nJ)	Error
16	2	1	0.16	75.98	88.54	14%
	2	4	0.04	45.41	52.59	14%
64	3	1	0.64	600.461	578.57	4%
	3	4	0.16	402.371	350.92	15%

t is the number of twiddle computation blocks and is calculated as follows:

$$t = \begin{cases} (V_p - 1)(H_p - 1) & \text{when } (H_p = \log_4 n \ \& \ V_p = 4) \\ V_p(H_p - 1) & \text{when } (H_p = \log_4 n \ \& \ V_p < 4) \\ (V_p - 1)H_p & \text{when } (H_p < \log_4 n \ \& \ V_p = 4) \\ V_p H_p & \text{when } (H_p < \log_4 n \ \& \ V_p < 4). \end{cases} \tag{9}$$

4.4.2. Design trade-offs and performance analysis. We vary the parameter values to analyze the energy variation based on the architectural changes. Figure 9(b) shows the energy dissipation and area of various design points for $n = 16$. It shows that more parallelism increases the energy efficiency of the FFT design despite increasing the area requirement.

We compared estimates from the functions against actual values based on synthesized designs and low-level simulation as described in Section 3. The input test vectors for the simulation are randomly generated and its average switching activity is 50%. We observed that the estimation error using our functions (see Table 5) is below 15% for energy dissipation.

5. Design methodology using the model and energy optimization

Our domain-specific modeling provides an energy estimation methodology to facilitate design decisions in the early phases of the design cycle. The system-wide energy function captures the impact of the architecture parameters on the system-wide energy at the algorithmic level. Using this, the designer identifies trade-offs among area, latency, and energy and explores a domain and identifies an appropriate design based on a selection criteria [4, 11]. Also, using our model and design methodology, the designer can further optimize a chosen design by improving the performance of the components that dissipate the significant energy. Initially, the system-wide energy function is analyzed to identify the distribution of energy dissipation among various types of components. Components with higher percentage of energy dissipation are chosen as possible candidates for design modification.

Design 1 in Figure 10(a) and (b) shows the distribution of energy dissipation among various components for the best design for $n = 3, 12$ identified in Section 4.2. 47% and 76% of the energy is dissipated in registers for problems of size 3×3 and 12×12, respectively. Note that bulk of energy is dissipated in registers. Also, it is

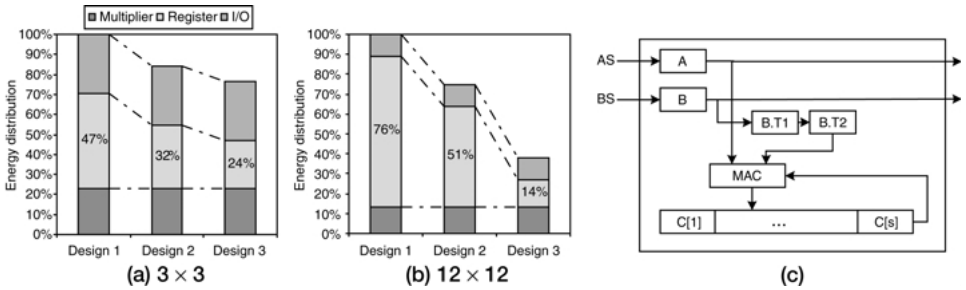

Figure 10. Change in distribution of energy among components during optimization for problem sizes (a) 3×3 (b) 12×12, and (c) architecture of the *PE* used in Design 2, 3.

hard to reduce energy dissipation in multipliers and I/O ports without increasing latency (hence energy dissipation). Thus, effort at algorithm and architecture level must be directed at reduction of the number of registers.

Based on above observations, we modify the algorithm or architecture in such a way that the number of registers are minimized. The result is denoted as Design 2 shown in Figure 10(c). Various mapping techniques are developed to reduce the number of registers from $2n^2 + 6n$ to $n^2 + 4n$. For example, two registers ($AS.LR$ and $AS.RR$) in Figure 6(b) are replaced by one register (A) by feeding elements of matrix A n cycles after those of matrix B.

Careful analysis of data movement reveals that only two registers ($B.T1$ and $B.T2$) are enough to store the elements of matrix B. Figure 10(a) and (b) shows that, in Design 2, energy dissipation is reduced by 16% and 25% for problems of size 3×3 and 12×12, respectively. Amount of energy dissipation in registers is reduced from 47% and 76% to 32% and 51%, respectively.

Besides reduction of energy at architecture and algorithm level, further reduction is possible at implementation level. For example, the registers to store intermediate results (c in Figure 10(c)) can be replaced by CLB-based SRAMs to reduce power per element. However, the minimum number of words for the SRAM in the target FPGA is 16.

The result is shown as Design 3 in Figure 10. Compared with Design 1, energy dissipation is reduced by 23% and 62% for problems of size 3×3 and 12×12, respectively. Proportion of energy dissipation in registers is further reduced to 24% and 14%, respectively.

6. Conclusion

This paper introduced domain-specific energy modeling for rapid system-level energy estimation and algorithmic level optimization for reconfigurable architectures. The modeling captures the details of the architecture and the algorithm to identify parameters affecting the power performance, hence facilitating derivation of a system-wide energy function. Matrix multiplication on a uniprocessor and a linear array architecture and FFT on a heterogeneous linear pipelined architecture were

chosen as example domains to illustrate construction of a high-level model, derivation of power functions for individual components, and combine them to obtain an system-wide energy function. The reference low-level energy evaluations were obtained on a Virtex-II device. For specific domains, the system-wide energy function was tested on several sample designs for its accuracy against time-consuming low-level energy estimation. The error in the system-wide energy estimation using the high-level model ranged from 7% to 15%.

Our modeling technique provides a virtual malleable data path. It is virtual because at the time of performance estimation we do not implement the design on target FPGAs. It is malleable because there are several parameters that can be varied to understand the trade-offs between different performance metrics such as energy, area, and latency. Such a characteristic makes our proposed model suitable to be considered during large application synthesis where several different kernels are integrated to implement a system such as MPEG encoding or Software Defined Radio. In such scenarios, once we have models for various kernel implementations, we can exploit the multi-level design space exploration technique provided in the MILAN framework [11]. Model-based Integrated SimuLAtioN (MILAN) is a model based extensible framework that facilitates rapid, multi-granular performance evaluation of a large class of embedded systems, by seamless integration of different widely used simulators and design tools into a unified environment. MILAN can be used to provide graphical interface for domain-specific modeling, automate power function estimation, and generation of energy profile based on the system-wide energy function.

Acknowledgments

This work is supported by the DARPA Power Aware Computing and Communication Program under contract F33615-C-00-1633 monitored by Wright Patterson Air Force Base, in part by the National Science Foundation under award No. 99000613, and in part by the DARPA/DSO OPAL Program through the Carnegie Mellon University under subcontract number 1-541704-50296. A preliminary version of this paper appears in *Engineering of Reconfigurable Systems and Algorithms*, 2002. Ju-wook Jang's work is supported by LG Yonam Foundation.

References

1. A. Bogliolo, L. Benini, and G. Micheli. Regression-based RTL power modeling. *ACM Transactions on Design Automation of Electronic Systems*, 5(3), 2000.
2. B. L. Bowerman and R. T. O'Connell. *Linear Statistical Models-An Applied Approach*, 2nd ed. Brooks/Cole Pub Co., 1990.
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. S. Choi, R. Scrofano, and V. K. Prasanna. Energy-efficient design of kernel applications for FPGAs through domain-specific modeling. *5th annual Military and Aerospace Programmable Logic Devices*, 2002.
5. H. ElGindy and Y.-L. Shue. On sparse matrix-vector multiplication with FPGA-based systems. *IEEE Symposium on Field Programmable Custom Computing Machines (FCCM)*, April 2002.
6. R. Hogg and E. Tanis. *Probability and Statistical Inference*, 6th ed. Prentice Hall, Upper Saddle River, NJ, pp. 656–657, 2001.
7. J. W. Hong and H. T. Kung. I/O complexity: The red-blue pebbling game. *ACM Symposium on Theory of Computing (STOC)*, 1981.
8. J. Jang, S. Choi, and V. K. Prasanna. Energy-efficient matrix multiplication on FPGAs. *International Conference on Field Programmable Logic and Applications*, 2002.
9. P. Master and P. M. Athanas. Reconfigurable computing offers options for 3G. *Wireless Systems Design*, 1999.
10. S. Mohanty, V. K. Prasanna, S. Neema, and J. Davis. Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation. *Language Compilers and Tools for Embedded Systems*, 2002.
11. S. Mohanty, S. Choi, J. Jang, and V. K. Prasanna. A model-based methodology for application specific energy efficient data path design using FPGAs. *IEEE International Conference on Application-Specific Systems, Architectures and Processors*, 2002.
12. T. Mudge. Power: A first-class architectural design constraint. *IEEE Computer*, 34(4), April 2001.
13. A. V. Oppenheim and R. W. Schafer. *Discrete-Time Signal Processing*, Prentice Hall, Englewood Cliffs, NJ, 1989.
14. V. K. Prasanna Kumar and Y. Tsai. On synthesizing optimal family of linear systolic arrays for matrix multiplication. *IEEE Transactions on Computers*, 40(6), 1991.
15. H. Quinn, M. Leeser, and L. S. King. Implementing image processing pipelines in a hardware/software environment. *High Performance Embedded Computing Workshop*, 2002.
16. A. Ragunathan, N. K. Jha, and S. Dey. *High-level Power Analysis and Optimization*, Kluwer Academic Publishers, Norwell, MA, 1998.
17. R. Scrofano, S. Choi, and V. K. Prasanna. Energy efficiency of FPGAs and programmable processors for matrix multiplication. *International Conference on Field-Programmable Technology*, 2002.
18. L. Shang and N. K. Jha. High-level power modeling of CPLDs and FPGAs. *International Conference on Computer Design*, 2001.
19. L. Shang, A. Kaviani, and K. Bathala. Dynamic power consumption in Virtex-II FPGA family. *International Symposium on Field Programmable Gate Arrays*, 2002.
20. A. Stammermann, L. Kruse, W. Nebel, and A. Prastsch. System level optimization and design space exploration for low power. *International Symposium on System Simulation*, 2001.
21. H. Styles and W. Luk. Customising graphics application: Techniques and programming interface. *IEEE Symposium on Field Programmable Custom Computing Machines* (*FCCM*), 2000.
22. Xilinx Inc. Xilinx Application Note: Virtex-II Series and Xilinx ISE 4.1i Design Environment. http://www.xilinx.com.
23. G. Yeap, *Practical Low Power Digital VLSI Design*, Kluwer Academic Publishers, Norwell, MA, 1998.