

A Methodology for Energy Efficient FPGA Designs Using Malleable Algorithms ^{*}

Jingzhao Ou and Viktor K. Prasanna

Department of Electrical Engineering, University of Southern California
Los Angeles, California, 90089-2560 USA
{ouj, prasanna}@usc.edu

Abstract. A recent trend towards integrating FPGAs with many heterogeneous components, such as memory systems, dedicated multipliers, etc., has made them an attractive option for implementing many embedded systems. Paradoxically, the integration that makes modern FPGAs an attractive computing substrate also makes the development of energy efficient FPGA designs very challenging in practice. This is due to the many alternatives available for implementing a desired functionality and a lack of high-level models of FPGA architectures that can accurately capture the energy dissipation behavior of alternatives. To address these issues, we propose a methodology for energy efficient FPGA designs using malleable algorithms. Malleable algorithms are used to expose the architecture-platform aware specifications of alternate implementations of the desired functionalities. Our methodology consists of three major design steps: domain-specific energy performance modeling, development of malleable algorithms, and system-level optimization. Energy efficient designs are realized through close interaction among these three design steps. To illustrate the proposed design methodology and demonstrate the benefits of designing using malleable algorithms, we present the development of a beamforming application through a high-level MATLAB/Simulink based FPGA design tool developed by us. By tuning the design knobs exposed by malleable algorithms, the design of the beamforming application identified through system-level optimization achieves up to 30% energy reduction compared with other designs considered in our experiments.

1 Introduction

Increasing density and integration of various hardware components, such as embedded multipliers, memory blocks, and RISC processors, etc., have made FPGAs an attractive option for implementing many embedded systems. Besides, with the proliferation of portable and mobile devices, energy efficiency has become increasingly important in the design of these embedded systems.

Paradoxically, the integration of heterogeneous components that makes modern FPGAs an attractive computing substrate makes energy efficient FPGA designs very difficult in practice. The main reason is that the progress in the design automation at VLSI level has outpaced the corresponding evolution of high-level abstractions that allow a designer to develop applications on such heterogeneous devices. Also, rapid and accurate energy estimation is challenging for FPGA designs. On the one hand, energy

^{*} This work is supported by the United States National Science Foundation (NSF) under award No. CCR-0311823.

estimation using RTL (Register Transfer Level) simulation (which can be accurate) is too time consuming and can be overwhelming considering the fact that there are usually many possible implementations of an application on FPGAs. On the other hand, the basic building blocks of FPGAs are look-up tables (LUTs), which are too low-level an entity to be considered for high-level modeling and rapid energy estimation. No single high-level model can capture the energy dissipation behavior of all possible implementations on FPGAs. Lacking such a rapid energy estimation technique would further prevent optimizing the energy performance of the complete applications.

In this paper, we address the following design problem. The target application is decomposed into a set of kernels. There are precedence or communication relations between the kernels. Various hardware implementations of these kernels on the target FPGA device are also given. Our goals are: (1) to find appropriate high-level abstractions of the given implementations which enable rapid and accurate energy performance estimation of a design instance; (2) to traverse the design space populated through the high-level abstractions and identify energy efficient design(s).

There are two main contributions of this paper. We propose the concept of *malleable algorithms*, which are architecture-platform aware specifications of alternate implementations of a given functionality. Details of malleable algorithms are further discussed in Section 3. The other contribution is a design methodology for energy efficient FPGA designs based on malleable algorithms. There are three main thrusts in our design methodology. First, *domain-specific modeling* is used to derive analytical models that capture the high-level energy dissipation behavior of the alternate implementations of the kernels. In [3], we have shown that the energy performance of these implementations can be obtained rapidly and fairly accurately using these analytical models. Second, *malleable algorithms* are developed to encapsulate the various implementations of the kernel as well as the analytical models obtained using domain-specific modeling. Finally, *system-level optimization* is performed by tuning the design knobs exposed by malleable algorithms to identify energy efficient designs of the target application.

For the sake of illustration, we provide an implementation of our design methodology based on a start-of-the-art MATLAB/Simulink based high-level design tool. We design a beamforming application using this tool. By tuning the design knobs exposed by malleable algorithms, the design of the beamforming application identified through system-level optimization achieves up to 30% energy reduction compared with other designs considered in our experiments.

The paper is organized as follows. Section 2 discusses related work. Section 3 introduces the concept of *malleable algorithms*. Section 4 presents our design methodology based on malleable algorithms. Section 5 discusses an implementation of the design methodology using a high-level MATLAB/Simulink based FPGA design tool developed by us. In Section 6, we demonstrate the development of an adaptive beamforming application using malleable algorithms. We conclude in Section 7.

2 Related work

The Carte compiler [11] from SRC contains a library for FPGA implementations of various functionalities, which is used in the hardware and software compilation processes for their computers. These implementations are represented as RTL netlist files. Different implementations of the same functionality are not captured by their library. Also, energy performance of the various FPGA implementations is not captured.

High-level design tools are becoming popular for designing using FPGAs. One important kind of tools are those based on MATLAB/Simulink, such as *DSP Builder* [1] from Altera and *System Generator* [12] from Xilinx. These tools provide a high-level (arithmetic level) abstraction of the underlying hardware resources and allow the application designers to describe the data flow and its hardware realization directly through this high-level abstraction. While resource utilization can be obtained rapidly using these MATLAB/Simulink based tool, no energy performance information is associated with the high-level abstraction provided by these tools. Besides, systematic traversal of the design space and identification of energy efficient designs are also not supported by the current versions of the tools.

Other high-level design tools are such as DK2 [4] from Celoxica and Forge [12] from Xilinx which use high-level languages such as C and Java for designing using FPGAs. When using these tools, the application designers describe their applications using C or Java and rely on the compiler to infer the appropriate architecture for implementing the application and to perform optimizations such as loop unrolling, pipelining, etc. The output of these tools is either HDL code or EDIF netlist. While these system level design tools have proved to be capable of simplifying the design complexity and reducing the design time, the compilers used by these tools do not optimize the energy performance of the generated FPGA designs.

3 Malleable Algorithms

First of all, we assume that there are two roles, an algorithm designer and an end user, involved in the design process. An *algorithm designer* is concerned with providing the end user with efficient implementations of the set of kernel functionalities that constitute the complete application. For the adaptive beamforming application discussed in Section 6, Levinson Durbin recursion and FFT are examples of such kernels. An *end user* is concerned with designing and implementing an “efficient” hardware solution for an application, such as the beamforming application discussed in Section 6. Energy dissipation, latency, throughput, etc., are some of the considerations from the end user’s perspective. Also, we define a *platform* as a device provided by a vendor which consists of a set of hardware components for processing, storage, and interconnection. In this paper, we focus on platforms with FPGA as the major component on the device. The Xilinx Virtex-II Pro [12] is such an example platform we are targeting.

Based on the above assumptions and definitions, a *malleable algorithm* is defined as an architecture-platform aware specification of alternate implementations of a given functionality. The availability of such alternatives is due to the large amount of programmable resources and heterogeneous embedded components found on many modern FPGA devices (e.g. the platforms of interest), which provide a very high flexibility in the hardware implementation. Most importantly, these alternatives would result in different amounts of energy dissipation for the execution of the same functionality. For example, there are three possible hardware bindings for implementing storage elements on Virtex-II Pro, which are registers, slice based RAMs, and embedded Block RAMs (BRAMs). Our work in [2] shows that registers and slice based RAMs have better energy efficiency for implementing small amount of storage while BRAMs have better energy efficiency for implementing large amount of storage. Another example is that matrix multiplication can be implemented using many architectures including a linear

array or a 2-D array. A 2-D array implementation uses more interconnects and can result in more energy dissipation compared with a linear array implementation.

A malleable algorithm captures two types of knobs, one describing the high-level functionalities while the other describing the alternative implementations of the high-level functionalities on a specific platform. These knobs are identified by the algorithm designer and are made available to the end user. For example, when developing an FFT kernel, we identify two kinds of knobs as shown in Table 1 (see Section 6.2). Then, in the system-level optimization discussed in Section 6.3, we enumerate these knobs to evaluate the possible implementations of an beamforming application and identify an energy efficient design for this application. Therefore, malleable algorithms play a key role in the interaction between the algorithm designer and the end user. They encapsulate the algorithm designers' understanding of performance modeling and *potential optimization knobs* that may or may not be exploited by the end user. By handling the knobs exposed by malleable algorithms at high-level, the end user is free to use an available optimization technique to find out the final implementation.

Besides, malleable algorithms make it possible to model FPGA platforms at high-level and thus enable rapid and accurate energy estimation for various implementations on a specific FPGA platform. This is because malleable algorithms relate the performance of the various implementations on a specific platform to a set of knobs identified by the algorithm designer. By restricting the modeling to a well-defined algorithm and architecture, techniques such as domain-specific modeling (see Section 4) can be applied to provide high-level abstractions of the low-level RTL implementations. Our experiments have shown that such careful abstractions can lead to rapid and fairly accurate energy estimation. In this scenario, malleable algorithms can be used to encapsulate the domain-based knowledge of the kernels identified by the algorithm designer and expose them to the end user for optimizing the energy performance of the complete application.

Malleable algorithms are loosely analogous to parameterized soft-IP (Intellectual Property) cores provided by EDA vendors that are instantiated by the end user at logic synthesis time. However, one of the significant differences between a malleable algorithm and a soft-IP core is that the former is defined at a higher level of abstraction than the latter, which is typically designed and optimized at the register transfer level. Such high-level abstractions would bring two major advantages to the designer. One is that it would not be possible to encapsulate all of these optimization possibilities into a single IP core described at the register transfer level. However, by using architecture-level abstractions, a single malleable algorithm is able to express these optimization possibilities. This ability makes malleable algorithms much more flexible than state-of-the-art parameterized soft-IP cores. Another advantage is that the high-level abstractions developed using our techniques enable fast high-level simulation. For example, the arithmetic level simulation of the IP cores within MATLAB/Simulink using the high-level representations provided by *System Generator* are usually much faster than the behavioral and architectural simulations in traditional FPGA design flows [12]. Most importantly, we have shown in [5] that some important low-level parameters such as switching activities, etc. can be predicted through such high-level simulation. These predicted parameters are crucial for obtaining rapid and accurate energy estimation.

Platform based designs are proposed by Keutzer *et al.* [8] for application development on SoCs (System-on-Chips). In their approach, a *platform* is defined as a layer of abstraction with two views. The upper view allows an application to be developed without referring to the lower levels of abstraction. The lower view is a set of rules that clas-

sify a set of components belonging to the platform. Platform based designs distill the complex world of system-level design into a few core concepts that have the potential of making the design process more manageable and efficient. However, to the best of our knowledge, no concrete methodology has been proposed to facilitate *algorithm design*. Here, algorithm design is defined as the use of an “exposed” model of the architecture platform to specify alternate implementations of a specific functionality. In our design methodology, a malleable algorithm is used to represent such an “exposed” model. The alternatives of the given functionality are captured by the high-level and low-level design parameters associated with the malleable algorithm while their performance can be calculated rapidly using the performance models specified by the malleable algorithm. Given a target application consisting of a set of kernels, system-level optimization is performed based on the malleable algorithms for the kernels.

4 Design Methodology

The application is decomposed into a number of kernels. We consider an application graph as input to our design methodology, which describes the communication and precedence relationships between the kernels. Based on the application graph, the proposed design methodology for energy efficient FPGA designs is shown in Figure 2(a).

In our design methodology, various alternative implementations of the kernels are first developed, which provide design trade-offs regarding energy, area and time. Using domain-specific modeling, analytical energy models are derived, which can be used to quickly estimate the performance of these implementations. By combining the implementations with corresponding performance models, malleable algorithms are developed for the kernels. Then, the end user describes the target application using the developed malleable algorithms. Finally, system-level optimization is performed based on this high-level description and identifies energy efficient design(s). The three major design steps (the shaded boxes in Figure 2(a)) are further explained in the following.

- **Domain-specific energy performance**

modeling: Domain-specific modeling is a hybrid approach (top-down followed by bottom up) toward performance modeling for kernels. While more details about this technique can be found in [3], we summarize it here for the sake of completeness.

As shown in Figure 1, we group implementations of a kernel based on their architectures and algorithm families. By doing so, we impose a high-level architecture onto the FPGA implementations within each domain. For a given kernel within each domain, performance models are first defined analytically as energy functions of the design parameters of the kernel. Low level simulations are then performed to estimate the constants in the analytical models for a specific hardware platform. The performance of these implementations, such as energy dissipation, latency, etc., can be calculated rapidly using these

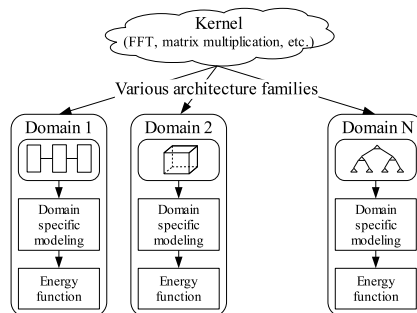


Fig. 1. Domain-specific modeling

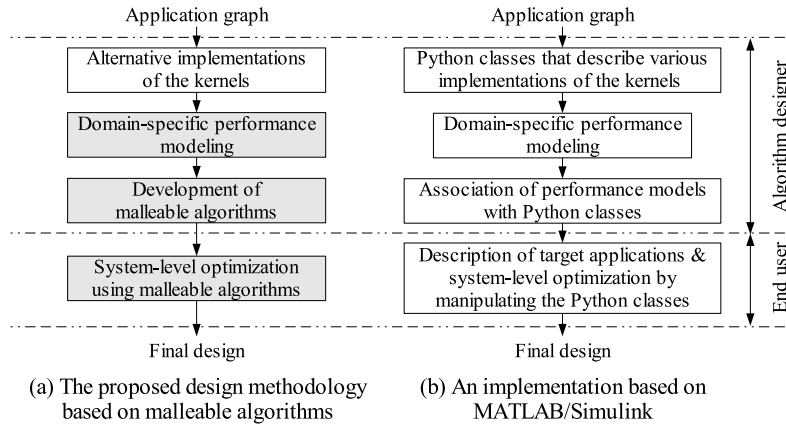


Fig. 2. Design flows

(analytical) performance models. We have developed several kernels such as matrix multiplication, matrix factorization, etc. and have shown that domain-specific modeling can lead to an average estimation error around 10% for these kernels [5].

- **Development of malleable algorithms:** The first step in developing malleable algorithms is to associate the various implementations of the kernels with the analytical models developed using domain-specific modeling. In addition, malleable algorithms expose the knobs that describe the high-level functionalities of the kernels as well as those that control the low level realization. These knobs are identified through domain-specific modeling and based on the possible application requirements. They are accessible to the end user during system-level optimization. One way of developing malleable algorithms is to use the object-oriented mechanism to perform such encapsulation and exposition as illustrated in Section 5.

- **System-level optimization using malleable algorithms:** Given an end-to-end application graph and a set of malleable kernel algorithms for the kernels, the application is described using the malleable algorithms. Basically, this includes describing the interface between the kernels and the design constraints. Then, system-level optimization consists of exploring the design space exposed by the malleable algorithms and identifying the settings of the algorithms which lead to energy efficient designs. Various combinatorial algorithms can be applied in the optimization process. For example, considering the beamforming algorithm discussed in Section 6.3, the application graph can be described as a linear pipeline. A dynamic programming based algorithm proposed in [6] can be used to find a design with minimum energy dissipation for executing one instance of the application.

The proposed design methodology is accomplished through the interactions between algorithm designers and end users. According to their roles discussed in Section 3, for the design flow shown in Figure 2(a), the algorithm designer is responsible for developing the various implementations of the kernels, using domain-specific modeling to analyze their energy performance, developing malleable algorithms by combining these information while the end user is responsible for system-level optimization using the malleable algorithms developed by the algorithm designer.

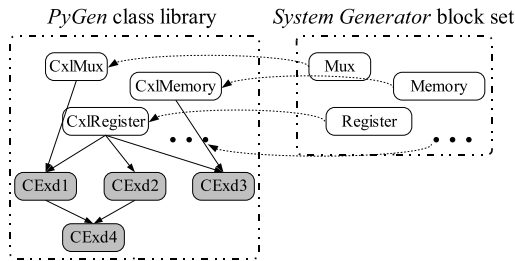


Fig. 3. Python class library within *PyGen*

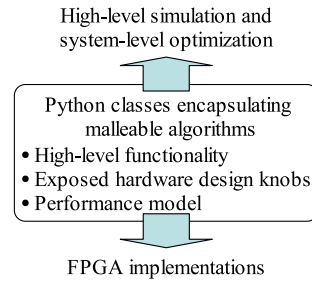


Fig. 4. Development of malleable algorithms in *PyGen*

5 An Implementation of the Design Methodology based on MATLAB/Simulink

To illustrate the development of malleable algorithms and our design methodology, we enhance *System Generator* [12], a MATLAB/Simulink based high-level FPGA design tool. This results in an add-on tool called *PyGen*, which provides additional functionalities. By creating an interface between Python and the MATLAB/Simulink based system level design tools, our tool allows the use of Python language [10] for describing FPGA designs within the high-level modeling environment provided by MATLAB/Simulink.

As shown in Figure 3, each block in the *System Generator* block set is mapped to the corresponding basic Python classes within the *PyGen* class library. By manipulating the basic Python classes, the designer can derive their own extended classes (the shaded blocks in Figure 3) and develop parameterized designs for the kernels.

We use *PyGen* to support the development of malleable algorithms. This is realized using the object-oriented mechanism to encapsulate the various implementations of the kernels as well as the energy performance models obtained through domain-specific modeling. As shown in Figure 4, the Python classes encapsulate (1) high-level functionalities of the kernels which can be for high-level simulation in MATLAB/Simulink; (2) exposed hardware design knobs which control the generated low-level implementations on the target device; (3) performance models for rapid energy estimation. See [5] for more details on the software architecture of *PyGen*.

The design flow of the proposed design methodology realized using *PyGen* is shown in Figure 2(b). Various implementations of the kernels are described as Python classes. After organizing the class hierarchy and classifying the classes into different domains, domain-specific modeling is performed within each domain to derive high-level performance models. Malleable algorithms are then developed by associating these performance models with the Python classes. The identified design knobs are exposed as data attributes of the Python classes. Finally, description of the target applications and system-level optimization are performed by manipulating these Python classes.

6 An Illustrative Example

To illustrate the proposed design methodology and demonstrate its effectiveness in developing energy efficient FPGA designs, we show the process of developing a beam-

forming application. Adaptive beamforming is used by many telecommunication systems such as software defined radio systems for better utilization of the limited radio spectrum. Energy efficiency is an important metric when implementing this application as these systems are usually battery operated.

6.1 MVDR Adaptive Beamforming

We consider a fast MVDR (Minimum Variance Distortionless Response) spectrum calculation application, which is described in [7]. This application uses Levinson Durbin recursion and saves much computation that is otherwise required by the direct calculation of the spectrum. It is part of the MVDR adaptive beamforming process. The application graph of the beamforming application is shown in Figure 5(a), which consists of three kernels: Levinson Durbin recursion, correlation of the predictor coefficients, and spectrum calculation using FFT.

6.2 Development of Malleable Algorithms for the Kernels

First of all, the algorithm designer develops malleable algorithms for the kernels that compose the beamforming application. A malleable algorithm of the spectrum calculation kernel using FFT is developed as a Python class *CxlFFT*. It contains two types of parameters. (1) *High-level functional parameters* describe the functionalities of the Python class. They are identified by considering the application requirements (see Section 6.3). (2) *Low-level architecture parameters* describe the hardware design knobs identified through domain-specific modeling on

Virtex-II Pro, our target FPGA platform. All the parameters are associated with the corresponding data attributes of the Python class, which are shown in Table 1. When this Python class is instantiated with the specific design parameters, *PyGen* will generate the corresponding hardware implementation. A performance model is obtained using the domain-specific modeling technique as described in Section 4. This model is associated with the corresponding data attributes of the Python class. Rapid power estimation can be performed using this performance model and the switching activity information estimated through high-level simulation within MATLAB/Simulink. An average estimation error of 6% is observed by comparing with the results from low-level RTL simulation. See [5] for more details on the various implementations of the FFT kernel, the derivation of the performance models and the experimental results.

Using a similar approach, we develop malleable algorithms for the other two kernels, which are represented by Python classes *CxlLevDur* and *CxlCorr*. The design parameters captured by these two classes are: *Frq* (operating frequency), *M* (number of antenna elements in the system), *degPar* (degree of parallelism), and *Precision* (precision of the data). Details of the development process of these malleable algorithms are not shown due to space limitation.

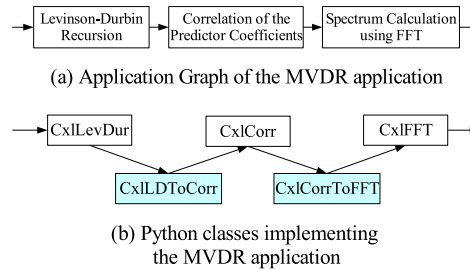


Fig. 5. MVDR beamforming

Table 1. Data attributes of Python class *CxIFFT* for an FFT kernel

| High-level functional parameters | Low-level architecture parameters |
|---|--|
| <i>Frq</i> (operating frequency) | <i>Arch</i> (architecture: unfolded or folded) |
| <i>nPnt</i> (number of frequency points) | <i>Sto</i> (hardware binding of storage elements: registers, slice-based RAM or Block RAM) |
| <i>Precision</i> (data precision, number of bits) | <i>degPar</i> (degree of parallelism) |

6.3 System-level Optimization Using Malleable Algorithms

Using the malleable algorithms developed in Section 6.2, the end user describes the target application and performs system-level optimization. As shown in Figure 5(b), description of the beamforming application uses the Python classes that implement the three kernels. In addition, two interfacing Python classes, *CxLDTtoCurr* and *CxLCorrToFFT*, are developed to describe the data communication between the kernels, such as the input/output data patterns and the buffering requirements, etc. Description of design constraints are performed by writing Python code to manipulate these Python classes. Then, the correctness of the designs described in Python classes can be verified by instantiating them with appropriate design parameters, generating corresponding Simulink models, and performing arithmetic level simulation in MATLAB/Simulink.

Since the beamforming application can be described as a linear pipeline of kernels, the dynamic programming algorithm proposed in [6] can be applied to find a design for this application so that the energy dissipation for executing one data sample is minimized. First, we create a trellis. The nodes on the trellis represent the various implementations of the kernels. They are obtained by enumerating the design knobs exposed by the corresponding Python classes implementing these kernels while satisfying the application requirements. Taking the FFT kernel as an example, while *Frq*, *nPnt* and *Precision* are set according to application requirements, we enumerate the various possible combinations of *Arch*, *Sto* and *degPar* and obtain five nodes on the trellis representing different possible implementations of this kernel. The weights of these nodes are the costs of executing the kernels using the corresponding implementations represented by them. The weights can be obtained by querying the performance models encapsulated by the Python classes. Besides, there are edges between the nodes which describe the communication between the kernels. Similarly, the weights of the edges can be obtained from the Python classes that implement the communication channels between the kernels. Thus, the identification of the energy efficient design is

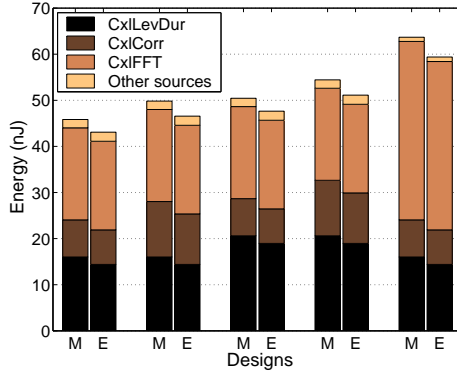


Fig. 6. Energy performance of various designs of the MVDR application (*M* denotes measured data; *E* denotes estimated data obtained by querying the performance models associated with the Python classes)

formulated as an optimization problem, which is to find a traversing path on the trellis with minimum energy cost. The dynamic programming algorithm identifies such a path in an iterative manner. See [6] for more details on the problem formulation and the optimization algorithms.

To show the effectiveness of the proposed design methodology, we perform exhaustive search and identify five designs with lowest energy dissipation according to their energy performance obtained through low-level RTL simulation. We also traverse the MATLAB/Simulink design space using our tool and identify five designs with lowest energy dissipation using the performance models associated with the Python classes. They are the same as the five designs identified through low-level simulation. Figure 6 shows the measured (through low-level simulation) and estimated energy performance of these designs. For these five designs, the identified design (e.g. the left most design shown in Figure 6) achieves an energy reduction of up to 30% compared with the other designs considered. For this identified design, the FFT kernel is implemented with *Arch = unfolded*. The FFT implementation using an unfolded architecture occupies more slices than one using a folded architecture. The implementation using an unfolded architecture has less control and storage overheads and thus improves the energy efficiency of the complete application.

7 Conclusion

The concept of malleable algorithms was proposed. We presented a design methodology based on this concept for developing energy efficient applications on FPGAs that contain heterogeneous components. We are currently exploring the potentials of malleable algorithms for application synthesis using FPGAs that integrate embedded processors.

References

1. Altera, Inc., <http://www.altera.com>.
2. S. Choi, R. Scrofano, V. K. Prasanna, and J.-W. Jang, "Energy Efficient Signal Processing Using FPGAs," *ACM Int. Symp. on Field-Programmable Gate Arrays (FPGA)*, 2003.
3. S. Choi, J.-W. Jang, S. Mohanty, V. K. Prasanna, "Domain-Specific Modeling for Rapid System-Wide Energy Estimation of Reconfigurable Architectures," *Engineering of Reconfigurable Systems and Algorithms (ERSA)*, 2002.
4. DK2, Celoxica, Inc., <http://www.celoxica.com/products/tools/dk.asp>.
5. J. Ou and V. K. Prasanna, "PyGen: A MATLAB/Simulink based Tool for Synthesizing Parameterized and Energy Efficient Designs Using FPGAs," *Field-Programmable Custom Computing Machines (FCCM)*, 2004.
6. J. Ou, S. Choi, and V. K. Prasanna, "Energy-Efficient Hardware/Software Co-Synthesis for a Class of Applications on Reconfigurable SoCs," *Int. Journal of Embedded Systems*, 2004.
7. S. Haykin, "Adaptive Filter Theory," 3rd Edition, *Prentice Hall*, 1991.
8. K. Keutzer, S. Malik, A. R. Newton, J. M. Rabaey, and A. Sangiovanni-Vincentelli, "System Level Design: Orthogonalization of Concerns and Platform-Based Design," *IEEE Trans. on CAD*, 19(12), Dec. 2000.
9. Mentor Graphics, Inc., www.mentor.com.
10. Python, <http://www.python.org>.
11. "Carte Programming Environment," online available at <http://www.srccomp.com/>.
12. Xilinx, Inc., <http://www.xilinx.com>.