

A Model-Based Extensible Framework for Efficient Application Design Using FPGA

SUMIT MOHANTY and VIKTOR K. PRASANNA
University of Southern California

For an FPGA designer, several choices are available in terms of target FPGA devices, IP-cores, algorithms, synthesis options, runtime reconfiguration, degrees of parallelism, among others, while implementing a design. Evaluation of design alternatives in the early stages of the design cycle is important because the choices made can have a critical impact on the performance of the final design. However, a large number of alternatives not only results in a large number of designs, but also makes it a hard problem to efficiently manage, simulate, and evaluate them. In this article, we present a framework for FPGA-based application design that addresses the aforementioned issues. This framework supports a hierarchical modeling approach that integrates application and device modeling techniques and allows development of a library of models for design reuse. The framework integrates a high-level performance estimator for rapid estimation of the latency, area, and energy of the designs. In addition, a design space exploration tool allows efficient evaluation of candidate designs against the given performance requirements. The framework also supports extension through integration of widely used tools for FPGA-based design while presenting a unified environment for different target FPGAs. We demonstrate our framework through the modeling and performance estimation of a signal processing kernel and the design of end-to-end applications.

Categories and Subject Descriptors: I.6.3 [**Simulation and Modeling**]: Applications; J.6 [**Computer Applications**]: Computer-Aided Engineering—*Computer-aided-design (CAD)*

General Terms: Design, Performance

Additional Key Words and Phrases: Modeling, reuse, design tool, extensible

ACM Reference Format:

Mohanty, S. and Prasanna, V. K. 2007. A model-based extensible framework for efficient application design using FPGA. *ACM Trans. Des. Autom. Electron. Syst.* 12, 2, Article 13 (April 2007), 26 pages. DOI = 10.1145/1230800.1230805 <http://10.1145/1230800.1230805>

This work was supported by the DARPA Power-Aware Computing and Communication (PAC/C) Program under contract S7-6AX077X62A9. A preliminary version of this article appears in *the International Conference on Field-Programmable Logic and its Applications (FPL)*, 2003.

Authors' addresses: S. Mohanty, Microsoft, 1 Microsoft Way, Redmond, WA 98052-6399; email: sumit.mohanty@gmail.com; and V. K. Prasanna, University of Southern California, 3740 McClintock Ave., Los Angeles, CA 90089; email: prasanna@usc.edu.

Permission to make digital or hard copies part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from the Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org. © 2007 ACM 1084-4309/2007/04-ART13 \$5.00 DOI 10.1145/1230800.1230805 <http://doi.acm.org/10.1145/1230800.1230805>

ACM Transactions on Design Automation of Electronic Systems, Vol. 12, No. 2, Article 13, Publication date: April 2007.

1. INTRODUCTION

Continuous increase in the available reconfigurable logic, number of embedded multipliers, I/O bandwidth, and size of embedded memory make an FPGA an attractive device for implementing the complex and compute-intensive applications used in wired and wireless mobile devices [Actel 2007; Altera 2007; Xilinx 2007b]. In addition, several novel features, such as nonvolatile flash-based configuration memory [Actel 2007] which provide efficient startup and partial reconfiguration [Xilinx 2007b] enable low-power application design. Therefore, FPGAs are increasingly being considered as a high-performance low-cost alternative to general-purpose processors, DSPs, and ASICs, especially for the signal processing domain [Guo et al. 2004; Jang et al. 2002; Srivastava et al. 2003]. FPGAs offer several design choices such as operating frequency, precision, amount of memory, degrees of parallelism, clock gating, implementation alternatives (binding options), etc. In addition, other capabilities that play a significant role, especially for energy-efficient design, are dynamic voltage scaling, choice of low-power operating states, and device activation scheduling based on the duty cycle specification [Mohanty and Prasanna 2003]. Duty cycle is the proportion of time during which a system is operated. Such specification allows modeling a period of execution in alternating active and inactive phases. Energy dissipation (e.g., due to leakage current), especially for systems with low duty cycle, during inactive phases can contribute significantly to the overall energy dissipation of the system. Therefore, the tradeoff in performance cost between shutting down and starting-up a device and that of remaining idle needs to be considered during system design [Benini et al. 2000]. These choices define a large design space that must be efficiently explored to find suitable solutions. Adaptive beam forming, multirate and wavelet filters, software-defined radio, image processing, etc., are some of the applications that are suitable for implementation using reconfigurable devices [Choi et al. 2003; Ou et al. 2003]. These applications are compute-intensive and expected to satisfy hard latency constraints. Additionally, many of these applications are candidates for deployment in power-constrained environments.

Our focus is signal processing applications that are modeled as data flow graphs. An application model is an input to our design process and it is assumed that the user of our framework is able to define such a model. A data flow graph is a directed acyclic graph where the nodes represent tasks (or kernels) and the directed edges connecting the nodes represent dependency (order of execution) among the tasks (see Section 3). Thus we assume that an application consists of a set of kernels. In this article, a high-level model of an application refers to a data flow graph representation of the application. Efficient design of an application requires efficient implementation of constituent kernels. In this article, kernel refers to a signal processing kernel such as matrix multiplication, FFT, DFT, etc.

FPGA-based designs are usually evaluated using three performance metrics: latency, area, and energy. It has been observed that several implementations are possible for the same signal processing kernel. These provide a tradeoff between the three performance metrics, especially using parameterized IP (intellectual

property) cores [Choi et al. 2002; Jang et al. 2002]. Thus, our approach involves the use of multiple efficient designs for each kernel that provide a tradeoff between latency, area, and energy dissipation. Once candidate kernel designs are identified, application design involves selection of appropriate kernel implementation, as well as scheduling of the kernels on the target FPGAs, based on the performance requirements provided as input. An extension of the aforementioned problem is device selection, which involves evaluation of several candidate devices (FPGAs, DSPs, etc.) to identify the most efficient combination of devices. A novel contribution of our framework is enabling such device selection.

However, the major obstacle to widespread use of FPGAs is the lack of high-level design methodologies and tools [Doucet et al. 2002; McGregor et al. 1998; Mohanty and Prasanna 2004]. Most available FPGA design tools focus on compilation of a design specified in an HDL onto a target device. This process tends to be time-consuming and thus not suitable for efficient exploration of large design spaces. Therefore, it is necessary to support rapid and reasonably accurate performance estimation to evaluate candidate designs [Mamidipaka et al. 2003]. Towards this end, it has been shown that high-level design methodologies that are used to design efficient algorithms can be used to generate energy- and latency-efficient signal processing kernels [Jang et al. 2002; Jones et al. 2002; Mukherjee and Memik 2004].

While a number of implementation choices are available for a signal processing kernel, it is not easy to manage a large number alternatives in a manner suitable for efficient evaluation of the design choices [Kirk et al. 2002]. In addition, opportunities for runtime reconfiguration [Ghiasi et al. 2004; Maestre et al. 2001; Shirazi et al. 1998] and device shutdown and startup also need to be evaluated for lowpower application design. Therefore, a significant contribution of our framework is enabling duty-cycle-aware application design that maximizes battery life.

In this article, we discuss a model-based design framework that addresses the previous issues. For efficient kernel design, the framework enables modeling the data path and algorithm at an abstraction suitable for the representation of parameterized IP cores where the parameters can be varied to understand performance tradeoffs. Some sample parameters are the degree of parallelism, binding options, precision, etc. The designer uses the tools integrated in the framework to estimate performance, analyze the effect of parameter variation on performance, and identify optimization opportunities. The framework supports application modeling as a hierarchical data flow graph [Ledeczi et al. 2003] (explained in Section 3.2). Such representations allow us to define the application design problem as a mapping (task mapped to a device operating at a particular configuration) problem that can be solved in an efficient manner using, for example, dynamic programming or heuristics. In addition, the framework also supports a high-level performance estimator (HiPerE) to rapidly evaluate the performance of candidate application designs and a heuristic-based design space exploration tool, DESERT. Due to HiPerE and DESERT, our framework can perform rapid design space exploration, which is a significant advantage over traditional simulation-oriented design techniques. Our focus in designing

the framework is not to develop new techniques for compilation of high-level specifications onto FPGAs, but rather to provide efficient support for some of the design steps, such as modeling, design reuse, rapid coarse-grained estimation of various performance metrics, performance tradeoff analysis, and design space exploration.

The framework facilitates both high-level estimation and low-level simulation. High-level refers to the level of abstraction where performance models of the algorithm and architecture can be defined in terms of parameters and cost functions. In contrast, low-level refers to the level of modeling suitable for cycle-accurate or RT-level simulation and analysis. Low-level simulation support is provided through the integration of available simulation tools. The choice of target FPGA is also an alternative available to FPGA designers [Actel 2007; Altera 2007; Xilinx 2007b]. Typically, each target FPGA is associated with its own set of design tools and design flow, in addition to the tools provided by other EDA vendors such as Mentor Graphics FPGA Advantage [Mentor Graphics 2007a] and Synopsis Design Compiler [Synopsis 2007]. Therefore, our design framework allows integration of multiple design flows to a single unified design environment. The design framework is developed using a Generic Modeling Environment, a graphical tool suite supporting model integrated computing (MIC) [Software Integrated Systems 2006].

The article is organized as follows. The next section discusses some related work. Our approach to model FPGA-based designs is discussed in Section 3. Section 4 discusses the design of the framework, supported design flow, and design space exploration techniques based on our modeling approach. Techniques for design reuse and extension of the framework are discussed in Section 5. Examples demonstrating the use of the framework are discussed in Section 6. We conclude in Section 7.

2. RELATED WORK

Several tools are available for efficient application design using FPGAs. Simulink and Xilinx system generators provide a high-level interface to design applications using precompiled libraries of signal processing kernels [Xilinx 2007a]. Jones et al. [2002] present a compiler, PACT HDL, which given a high-level algorithm in C, generates power- and performance-optimized design for FPGAs. Other tools such as [Mentor Graphics 2007a] and [Xilinx 2007b] provide integrated design environments that accept high-level specification as VHDL and Verilog scripts, schematics, finite state machines, etc., and provide simulation, testing, and debugging support for the designer to implement a design using FPGAs. However, these tools start with a single conceptual design. Design space exploration is performed as part of implementation or through local optimizations to address performance bottlenecks identified during synthesis.

Several C/C++ language-based approaches, such as SystemC, Handel-C, SpecC [Celoxica 2005], and are primarily aimed at making the C language usable for hardware and software design and to allow efficient and early simulation, synthesis, and/or verification [SystemC 2007; Celoxica 2005]. However, these approaches do not facilitate modeling of applications and kernels at a

level of abstraction that enables rapid performance analysis and design space exploration. Additionally, generating source code and going through the complete synthesis process to evaluate each algorithm decision is time-consuming.

Stammermann et al. presented ORINOCO, a software tool for power dissipation analysis and optimization based on C/C++ and VHDL description of kernels [2001]. A top-down compilation method that compiles C programs into VHDL was proposed by Bazargan et al. [2000]. However, C/C++ or VHDL descriptions do not capture parameters affecting system-wide energy and also require of the designer a time-consuming process of coding, compilation, and simulation for performance estimation. In contrast, if a kernel is described as an algorithm, then we can perform an analytical estimation (see Section 6.3) of performance estimates via mathematical equations.

In addition to the tools discussed earlier, many techniques for formal modeling and optimization of a single metric (latency or energy), while mapping applications onto reconfigurable computing systems, have been proposed [Bondalapati and Prasanna 2000; Ou et al. 2003]. Bondalapati and Prasanna [2000] proposed techniques for mapping loop computations onto high-performance pipelined configurations, which amortizes reconfiguration cost over multiple iterations of the execution of the loop to reduce total execution time. A dynamic programming-based mapping technique for energy-efficient mapping of applications onto reconfigurable SoC chips is proposed by Ou et al. [2003]. An algorithm that combines exact and heuristic techniques for the synthesis of large time-constrained heterogeneous adaptive systems was proposed by Shenoy et al. [2001]. While these techniques are faster than our approach, some cannot be easily extended to support additional performance metrics or design choices such as shutdown and startup of devices to reduce power. Shang and Jha [2001] 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 an 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 to design energy-efficient algorithms without generating low-level (C/VHDL/RT) implementation.

Our approach enables high-level parameterized modeling for rapid performance estimation and efficient tradeoff analysis. Using our approach, the designer need not synthesize the design to verify design decisions. Once a model has been defined and parameters have been estimated, design decisions are verified using the high-level performance estimator. Additionally, parameterized modeling enables exploration of a large design space in initial stages of the design process. Our design framework also allows integration of various design tools and thus provides a unified extensible environment for FPGA design.

3. MODELING FPGA-BASED DESIGNS

Our modeling technique integrates two complimentary techniques: a hierarchical data flow graph for application specification [Ledeczi et al. 2003] and, domain-specific modeling-based approach for modeling the kernels [Choi et al.

2002]. In addition, we discuss how we model the performance and design constraints and reconfiguration.

3.1 Kernel-Level Modeling

Our kernel-level modeling enables specification of a parameterized design for signal processing kernels for implementation using FPGAs. We exploit domain-specific modeling, a technique for high-level modeling of FPGAs developed by Choi et al. [2002]. This technique has been demonstrated successfully for designing energy-efficient signal processing kernels using FPGAs [Jang et al. 2002]. A domain refers to a class of architectures and the corresponding algorithms for a particular signal processing kernel. A class of architectures can be a uniprocessor, linear array of processors, 2D array of processors, or any other class of parameterized architecture. For example, matrix multiplication on a linear array of processors is a domain. A model defined using this technique consists of RModules, interconnects, component-specific parameters and power functions, component power state matrices, and a system-wide energy function. A *relocatable module (RModule)* is a high-level architecture abstraction of a computation or storage module. For hardware implementations on an FPGA, a register can be an RModule if the number of registers in the design can vary based on the algorithm used. *Interconnect* represents the resources used for data transfer between the RModules. A component (also referred to as a building block) can be an RModule or interconnect. *Component-specific parameters* depend on the characteristics of the component and its relationship to the algorithm. For example, the degree of parallelism, precision, size of internal memory (on the FPGA), binding options for RModules, and power states are possible component-specific parameters. *Component-specific power functions* capture the effect of component-specific parameters on the average power dissipation of the component. For this we assume an average switching activity. *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, \dots, 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 $T \times n_i$ (see Figure 1). Latency is derived manually based on the algorithm description (see Section 6.1) using any complexity analysis technique. An entry in a CPS matrix represents the power state (e.g., active or clock-gated) of a component during a specific cycle and is determined by the algorithm. A *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. Figure 1 summarizes the domain-specific modeling approach. Details on how to develop system-wide energy functions can be found in Choi et al. [2002]. Section 6.3 discusses a sample kernel-level modeling exercise.

Modeling based on the technique described before has the following advantages: (a) Various parameters get exposed at the algorithm level; (b) performance models for energy, area, and latency are generated in the form of parameterized functions; (c) it is possible to rapidly estimate different performance

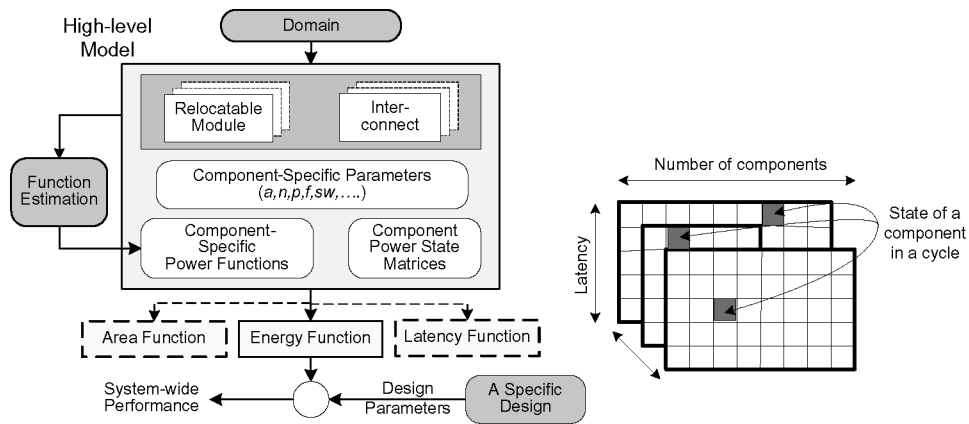


Fig. 1. Domain-specific modeling and performance estimation and component power state matrices.

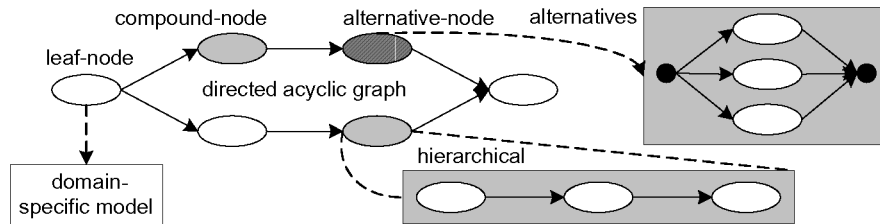


Fig. 2. Hierarchical data flow graph with alternatives.

metrics using only the information captured in the models; and (d) a parameterized model of a domain captures a set of designs (based on parameter values) that can be analyzed for various performance tradeoffs.

While domain-specific modeling is primarily a manual process, our design framework provides a number of automation features, such as a library of models, performance estimation through simulator integration, and design space exploration, that use domain-specific models.

3.2 Application-Level Modeling

Application modeling involves application specification as a hierarchical data flow graph with alternatives (Figure 2). The functional specification of the target application specifies the structure of the data flow graph and the choice of algorithms/implementations specifies the alternatives. Additionally, if a C/C++/VHDL implementation is available for a task, the various optimization options [Almagor et al. 2004] during compilation can also result in different design choices which can also be modeled as alternatives. Data flow is a widely used, expressive language for specifying signal processing applications [Lee and Messerschmitt 1987]. Among the various flavors and variants of data flow, we chose synchronous data flow to model the application. Synchronous data flow specification is widely used for designing embedded software [Bhattacharyya et al. 1999; Johnston et al. 2004]. The data flow graph is represented as a

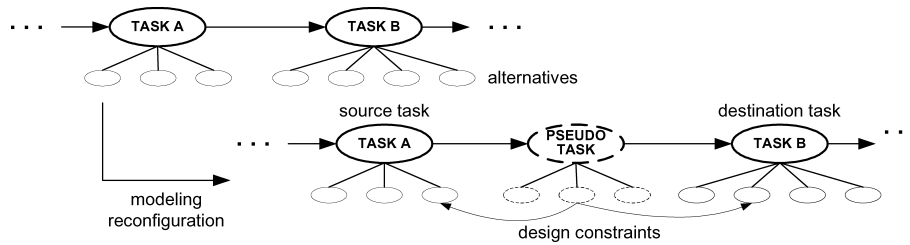


Fig. 3. Modeling reconfigurations.

directed acyclic graph (Figure 2) where each node refers to a kernel associated with the application. Nodes associated with the graph are classified into three types: leaf, compound, and alternative. The leaf node represents an application task (or kernel). A leaf node is associated with a model of kernels based on the domain-specific model (discussed in Section 3.1). Alternatively, a leaf node can be associated with a code (e.g., VHDL) that implements the task. This code is used to simulate the design of the kernel and to estimate latency, area, and energy dissipation. Alternative nodes capture alternatives associated with a task. An alternative can only be a leaf node or compound node. The alternatives implement the same task, but model different algorithms or implementations on different hardware devices. A compound node contains a hierarchical data flow graph with alternatives. Essentially, the compound nodes allow easy management of large application models. The directed edges connecting the nodes represent temporal dependencies among the nodes.

3.3 Modeling Constraints

We also model constraints that must be satisfied by the design(s). The constraints are divided into two types: *performance constraints* and *design constraints*. Performance constraints specify the latency and energy requirements for the complete application graph or any of its subgraphs. If the energy constraint is not specified, we minimize energy dissipation.

The design constraints specify valid combinations of task mappings. A mapping specifies the execution of a task on an FPGA operating in a specific configuration. A constraint on mapping specifies pairwise relations between two mappings. For example, if multiple FPGAs are available, a valid design constraint is “task A1 implemented on FPGA B1 operating in configuration C1 is not compatible with task A2 implemented on FPGA B2 operating in configuration C2”. This constraint will result in the selection of designs that do not include task A1 mapped on B1 using C1 and task A2 mapped on B2 using C2. Design constraints also capture requirements of the composition of components in case of device selection. Constraints on composition restrict the composition of alternate processing components. For example, given a set of choices that include two traditional processors, one such constraint can be “a valid system may contain one of the processors but not both”.

3.4 Modeling Reconfiguration

In our application model, each leaf node is mapped to an FPGA. We support designs that use multiple FPGAs. Therefore, each leaf node is associated with FPGAs that can implement the kernel represented by the leaf node. Each leaf node is also associated with a domain-specific model that is used to predict the area requirement of the kernel. Therefore, based on the capacity of the mapped FPGA and the area requirement of consecutive tasks, it is possible to determine whether reconfiguration is required between the execution of two tasks if both tasks are mapped onto the same FPGA.

The hierarchical data flow graph discussed earlier does not explicitly support reconfiguration. Therefore, we developed a technique that uses pseudotasks and design constraints to enable modeling of reconfiguration. This technique introduces a pseudotask (as an alternative node) between pairs of tasks, where a pair constitutes a *source* and a *destination* task. Given a set of tasks modeled as a data flow graph, two tasks are a source-destination pair if both are mapped onto the same FPGA and the execution of the destination task follows the execution of the source task on the mapped FPGA. The pseudotask models reconfiguration. The alternatives for this pseudotask are a set of possible reconfigurations determined based on the design choices available for the source and destination tasks. Each reconfiguration is modeled as a leaf node. As discussed before, a domain-specific model of a kernel specifies a set of designs. Therefore, based on the design selected for the source and destination tasks, multiple reconfigurations are possible: one for each unique pair of designs for the pair of source and destination tasks. Each reconfiguration is associated with a unique value of performance cost based on the latency and energy dissipation required per reconfiguration. Therefore, we specify a set of design constraints to ensure that appropriate reconfiguration is chosen based on the mappings chosen for the source and destination tasks. Thus, when we select the designs that meet these constraints, correct reconfigurations are also automatically chosen. In addition, reconfiguration costs also get automatically added to the overall performance of each design while evaluating the designs against the performance constraints. As overall performance includes both kernel execution cost and reconfiguration cost, it is possible to select suboptimal designs for kernels to avoid reconfiguration while minimizing overall latency or energy dissipation. This is useful, considering that popular FPGA devices such as the Virtex series [Xilinx 2007b] and Stratix series [Altera 2007] of devices have high energy and latency costs associated with reconfiguration.

If the target device supports dynamic voltage scaling, the performance cost for scaling voltage between kernel executions is also modeled as a reconfiguration.

3.5 Modeling Duty Cycle

Duty cycle is the proportion of time a system is active. Therefore, based on the duty cycle specification, system execution can be modeled as alternate active and inactive phases. The duration of each phase depends on the input rate and latency required to process a single input frame. Input rate can be variable. For

example, a target detection system can scan the environment and process data at a lower rate when no target is detected, and later increase the scan rate in case there is a detection. Our design framework needs the input rates to be statically defined. The variable input rate is specified as an ordered set of 2-tuples (Ir, Nf) where Ir refers to the input rate in Hz and Nf refers to the number of frames processed at the aforementioned rate. Given two consecutive 2-tuples (Ir_i, Nf_i) and (Ir_{i+1}, Nf_{i+1}) , application execution is modeled as “process Nf_i frames at the rate of Ir_i Hz followed by Nf_{i+1} frames at the rate of Ir_{i+1} Hz”. Our framework assumes that the given set of 2-tuples can repeat indefinitely to model processing of large number of input frames. Similarly, the maximum allowed latency to process an input frame can also be specified. If not specified, the framework derives the constraint based on the given maximum input rate.

During the inactive phases, devices that constitute the target hardware are idle. If the duration of inactive phases is significantly larger than that of active phases, the minimization of energy dissipation during inactive phases contributes significantly towards the minimization of the overall energy dissipation. Various options that can be evaluated to minimize energy dissipation include shutting-down devices, transition to a low-power mode, and leaving it as is (thereby saving the transition costs). Our framework evaluates these options to identify the one that is energy-optimal. In addition, based on the maximum allowed latency, our framework also evaluates the option of executing tasks in a low-performance mode (e.g., low operating frequency) that takes advantage of the available slack.

Duty-cycle-aware application design is considered nontrivial for the following reasons: (a) Both switching on and configuring of FPGAs involves energy dissipation which needs to be compared with energy dissipation while idling; (b) the latency required for switching on and configuration can be larger than the duration of inactive phases; and (c) using different optimization techniques results in different energy-efficient designs for the same duty cycle specification.

4. DESIGN FRAMEWORK

In this section, we discuss the design of the proposed framework. The framework provides a user-friendly modeling interface, supports creation of model libraries, and allows integration of analysis tools. Using the framework, a designer can specify and evaluate a target design space efficiently. Our design framework is based on the model integrated computing (MIC) approach [Sztipanovits and Karsai 1999]. In the following, we briefly discuss MIC and Generic Modeling Environment (GME) [Software Integrated Systems 2006] a tool supporting MIC. We will also discuss the design flow of the proposed framework.

4.1 Model-Integrated Computing and Generic Modeling Environment

The key idea of the MIC approach is the extension of the scope and usage of models such that they form the “backbone” of a model-integrated system development process [Sztipanovits and Karsai 1999]. Using MIC technology, the

designer captures information relevant to the system being designed in the form of high-level models. The high-level models can explicitly represent the target application, target hardware, and dependencies and constraints among different components of the models. Such models act as a repository of information that is needed for analyzing the system. Several tools exist that analyze different performance characteristics such as latency and energy of a system. Therefore, MIC allows the use of model interpreters to translate the information in the models to the input languages of analysis tools.

The design framework is developed using the Generic Modeling Environment GME [Software Integrated Systems 2006], a graphical tool suite that enables development of a modeling language for a domain, provides a graphical interface to model specific problem instances for the domain, and facilitates integration of tools that can be driven through the models [Software Integrated Systems 2006]. A metamodel (modeling paradigm) is a formal description of model construction semantics. Once the metamodel is specified by the user, it can be used to configure GME itself to present a modeling environment specific to the problem domain. MIC enables design reuse through the models. Various design problems within a domain share resources. Model interpreters are software components that translate the information captured in the models to drive integrated tools that estimate the performance (latency, energy, throughput, etc.) of a system. They can also be configured to automatically translate the models into executable specifications. Feedback interpreters are software components that analyze the output generated by the integrated tools and update the models. These interpreters are based on the model construction semantics and thus are suitable for any model based on a given paradigm.

4.2 Developing the Framework

In order to implement our framework, we defined a metamodel based on the models discussed in Section 3. Details regarding the development of a metamodel suitable for GME based on a given model can be found in Software Integrated Systems [2006]. Due to space constraints, we do not provide a sample metamodel. However, to provide a context, a metamodel is similar to an XML schema and the models are equivalent to XML documents that are based on the schema. GME provides a graphical interface for both metamodeling and modeling tasks. The exercise of developing a metamodel is a one-time process and, once the metamodel is developed, it is used to configure GME to present a modeling environment which we have used to develop the various models discussed in this article. Based on our experience, once the reader has a full understanding of the problem domain, the development of a metamodel usually takes only a few days. Once the metamodel is developed, one only needs to make modifications if the assumptions regarding the problem domain change. In addition to supporting the models, our metamodel also includes semantics to describe mapping of application tasks onto target devices, linking various models (e.g., the application and kernel-level models), and integrating the modeling of the reconfiguration and application model. To allow efficient reuse, we added hierarchical modeling support to kernel-level modeling. The hierarchy consists

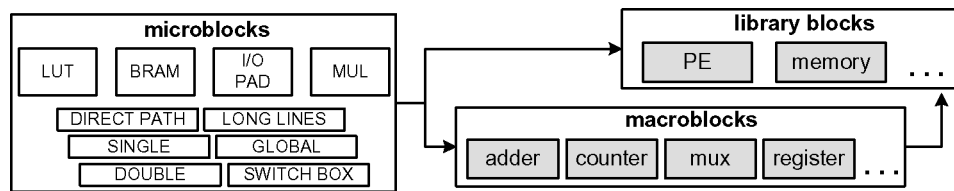


Fig. 4. Components for data path modeling.

of three types of components: micro-, macro-, and library blocks. A microblock is target FPGA-specific. For example, as shown in Figure 4, the microblocks specific to Xilinx Virtex II Pro are LUT, embedded memory cell, I/O pad, embedded multiplier, and interconnects [Xilinx 2007b]. In contrast, for the Actel ProASIC 500 series of devices, there will be no embedded multiplier. Macroblocks are basic architecture components such as adders, counters, multiplexers, etc., designed using the microblocks. A library block is an architecture component that is used by some instance of the target class of architectures associated with the domain. Both macro- and library blocks are also referred to as composite blocks. An RModule can be a micro-, macro-, or library block. We have developed a basic metamodel using GME that provides templates for different kind of building blocks and associated parameters. Once the target devices and kernels are identified, prior to using the design framework, the designer needs to modify the basic metamodel to support the specific instances of the building blocks. Being able to modify the modeling paradigm is a considerable advantage over many design environments where the description of a design is limited by the expressive power of the modeling language. While GME makes it quite easy to modify the metamodel, it is not a necessary step. Designers can instead develop a library of models using existing building blocks and reuse the models during the design process. Modification of the metamodel enables development of compact models. For example, a MAC can be modeled using a multiplier and an adder or a dedicated MAC building block. Thus a single building block would replace the two required for an adder and a multiplier. Each building block is associated with a set of component-specific parameters (Section 3.1). The state is one such parameter which refers to various operating states of each building block. Currently, we model only two states, *ON* and *OFF* for each microblock. In an *ON* state, the component is active and in *OFF* state it is clock gated. However, for composite blocks it is possible to have more than two states due to different combinations of states of the constituent microblocks. Power is always specified as a function with switching activity as the default parameter. The framework is available within MILAN, a model-based integrated simulation environment for embedded system design [Milan 2007].

We have developed a high-level estimation tool, the *kernel performance estimator*, to estimate different performance metrics associated with the complete kernel design. This estimation tool provides coarse-grained estimates for rapid design analysis during the initial design phases when the number of design choices is high. In our framework, we later use more accurate low-level simulators as the design space reduces. The input to the kernel performance estimator

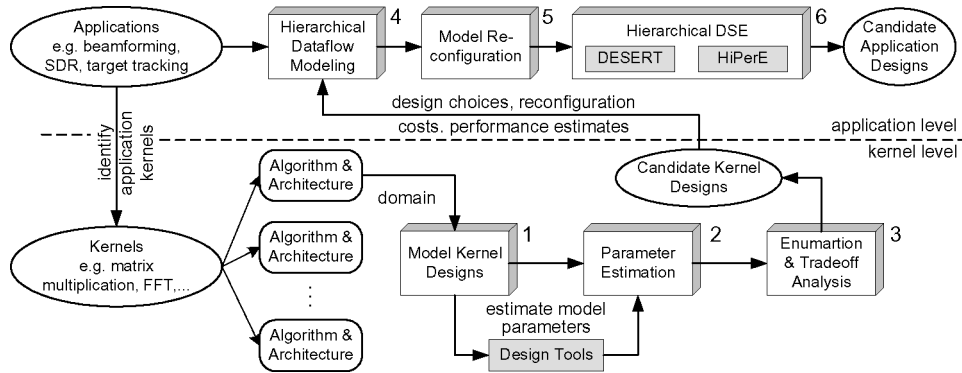


Fig. 5. Design flow using our framework.

is the model of the kernel data path and the component-specific power state matrices associated with the algorithm. The designer has the option of providing switching activity for each building block. The default assumption is a switching activity of 12.5% (the default value used by Xilinx XPower [Xilinx 2007b]). The area of the design is evaluated as the sum of the individual areas of the building blocks that constitute the data path. The latency estimate is implicit in the CPS matrices. Energy dissipation is estimated as a function of CPS matrices and the kernel data path model. Overall energy dissipation can be modeled as energy dissipated by each component during each cycle over the complete execution period. As energy can be evaluated as $power \times time$, for a given component, we use the power function for a specific power state (as specified by the CPS matrices) and the duration of each cycle to evaluate energy dissipated by the component in a specific cycle. Extending the preceding analysis for each component over the complete execution period, we can evaluate the overall energy dissipation by the design. The kernel performance estimator is based on the previous technique. The framework also includes a high-level performance estimator (HiPerE) to rapidly evaluate performance of candidate application designs and a heuristic-based design space exploration tool, DESERT.

4.3 Design Flow

As shown in Figure 5, the design flow using our framework consists of six steps. The first three steps deal with modeling the kernel designs based on domain-specific modeling and identifying design choices. The last three steps perform application-level modeling and design space exploration. In the following, we discuss each step in detail.

—(1). *Modeling kernel design*: In this step, the designer analyzes the kernels to define domain-specific models. The designer identifies the micro-, macro-, and library blocks and the associated component-specific parameters. The model of the data path is graphically constructed in this step using GME. The designer can also specify high-level scripts for the building blocks to be used in the next step. In addition, CPS matrices for the algorithm are also specified.

—(2). *Parameter estimation*: Estimation of the cost functions for power and area involves synthesis of a building block, low-level simulations, and in case of power, the use of confidence intervals to generate statistically significant power estimates [Choi et al. 2002]. The simulations are performed offline, or if the required simulator is integrated, using automatically specified high-level scripts. On the other hand, if a library of models is available, the stored performance estimates are used directly. Latency functions are estimated using the CPS matrices. System-wide energy is estimated using the latency function and component-specific power functions.

—(3). *Enumeration and tradeoff analysis*: In this step, the designer chooses the candidate kernel designs that would be evaluated while designing applications. Given a domain-specific model of a kernel, a set of designs is identified based on the parameter values and binding choices. The framework also generates comparison graphs to compare the performance of the designs.

—(4). *Hierarchical data flow modeling*: Once we have identified implementation choices for each kernel, we construct the application model as a hierarchical data flow with alternatives. Compound, alternative, and leaf nodes are used to specify the application model. Each leaf node models an implementation choice that is being considered for a kernel. In addition, each leaf node is associated with performance estimates obtained using the high-level performance estimator. Time required for modeling depends on the complexity of the task graph (number of nodes in the graph) and the number of design alternatives available for each task. For an application with six tasks and, on average, three alternatives for each task (see Section 6.2), based on our experience, it takes about two hours to generate the complete model.

—(5). *Modeling reconfiguration*: Based on the mapping and area estimates of the task implementations, pseudonodes are introduced to model reconfiguration. This step is automatic within our framework. The application model is analyzed using a topological sort and for each consecutive task (both source and destination) executing on a single FPGA, the application model introduces a pseudotask. Each pseudotask is automatically associated with a set of alternatives, and design constraints are introduced to ensure that correct reconfiguration is chosen based on the choices selected for the source and destination tasks.

—(6). *Hierarchical DSE*: This step uses DESERT and HiPerE to explore the design space using the application model. DESERT applies all the performance and design constraints and selects a set of designs that meets the constraints. HiPerE evaluates the selected designs based on their performance estimates and allows the designer to choose the final design based on the given performance requirements. A detailed discussion of this step is provided in Section 4.5.

4.4 Advantages of Our Methodology

There exist many design approaches to address the issue of energy-efficient system design, both in general and specific to FPGAs [Stammermann et al. 2001; Bondalapati and Prasanna 2000; Ou et al. 2003; Shenoy et al. 2001]. Different design approaches can be classified into two major categories. The first category is *optimization heuristics*. Approaches that fall under this category are based on a high-level abstraction (model) of the underlying system design problem and allow development of (provably) optimal solutions. Such high-level models are a mathematical abstraction of the application behavior and hardware characteristics, without the underlying implementation details (and therefore runtime complexities). While such approaches can quickly identify an optimal solution, they are sensitive to the errors introduced due to approximations during high-level modeling. The second category is *simulation-based* design space exploration. Approaches that fall under this category use simulators that are based on execution models suitable for cycle-accurate or register-transfer-level simulation. While simulations provide reliable results in terms of accuracy, design space exploration using such an approach consumes a significant amount of time due to low simulation speeds.

In contrast, our approach uses a hierarchical methodology for design space exploration that integrates the best of both worlds. Our methodology consists of two steps. The first step uses an optimization heuristic that operates on the initial design space and prunes it to a smaller set of designs based on the performance requirements. The second step uses a high-level estimation tool that operates on the designs identified in the first step. A high-level estimation tool operates at a higher level of abstraction than a typical simulator such as a cycle-accurate, register-transfer-level, or even an instruction-set simulator. For example, the high-level estimation tool discussed in this article requires application input as a data flow graph, which is at a much higher level of abstraction than, say, “C”, as required by SimpleScalar [SimpleScalar 2007]. The advantages of hierarchical design space exploration are that it:

- is robust against approximation errors due to high-level abstractions (models) used by the optimization heuristics;
- reduces the number of simulations necessary when compared against simulation-based design space exploration;
- combines the speed of optimization heuristic-based DSE and the higher accuracy of simulation-based DSE; and
- designers can potentially combine different optimization heuristics and high-level estimators to suit the needs of a target application design problem.

Low-level detailed simulators such as SimpleScalar [2007] and ModelSim [Mentor Graphics 2007b] provide accurate performance estimates, but are time-consuming. Evaluation of a large design space, even of the order of 10’s or 100’s, can take days. Our hierarchical design methodology does not require simulation to evaluate the complete design space. Rather, the simulators are used just to estimate the performance of different mappings that are used by DESERT and HiPerE to perform DSE. Therefore, our methodology is

significantly faster than simulation-based DSE. Section 6.2 compares the overhead due to our methodology and a simulation-based DSE.

4.5 Design Space Exploration for FPGA-Based Designs

Design space exploration refers to evaluation of the specified designs to identify candidates that meet the specified design and performance constraints.

—*Enumerating kernel designs*: Given a kernel, several domains can be identified, and within each domain several designs can be identified [Choi et al. 2002]. All possible designs within a domain are analyzed as follows: (a) For a given input size, the designs are analyzed to evaluate energy dissipated by each type of component as a fraction of total energy dissipation. This analysis helps to identify candidates for energy optimization. Similar analysis is also performed for area; (b) for a given input size, a single design parameter is varied while keeping the others constant, and different performance metrics are plotted as a graph to observe the effect of each parameter on different performance metrics. Depending on the number of parameters, several such graphs are possible; (c) if there exist alternatives for a building block (e.g., the embedded multiplier and configured multiplier in Virtex-II Pro), each option is considered so as to study the tradeoffs among the performance metrics. The design framework automatically generates a set of designs and estimates the latency, energy, and area associated with each design. We discuss a uniprocessor (single data path) implementing the “usual” block matrix multiplication as an example domain to demonstrate the exploration of kernel designs.

This domain uses a single multiplier and results in area- and energy-efficient designs. We consider an off-chip design where all matrix data is stored in an external memory outside the FPGA. In this design, the uniprocessor has one MAC (multiplier and accumulator), a cache (local buffer) of size c , and I/O ports. The data matrices are stored in an external memory. Block matrix multiplication is performed with block size $\sqrt{c} \times \sqrt{c}$. We identified four components: MAC, cache, the memory banks as RModules, and the I/O as an interconnect. The RModules have w bit precision. Therefore, the cache size (c) and precision (w) constitute the parameter that can be varied at design time. 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 cache size is small, the cache can be realized using CLBs configured as register modules. A larger cache can be realized using CLBs configured as SRAM [Xilinx 2007b]. However, a SRAM-based cache can only be configured to be a multiple of 16 bytes. Therefore, the aforementioned model provides several design choices based on the choice of multiplier, cache, cache size, and precision.

—*Exploring application design*: Exploring application design involves the use of DESERT and HiPerE to evaluate the design space and identify those

designs that meet the given performance constraints. The application design space is defined by the choice of implementations associated with the application tasks. Therefore, an application with n tasks $T_1 \dots T_n$, each task T_i with A_i implementation alternatives, will have a design space of size $\prod_{i=1}^n A_i$. Therefore, the initial design space can be large. Our experience with DESERT shows that we can prune a design space with approximately $10^{20} \sim 10^{40}$ designs in order of minutes [Mohanty et al. 2002]. DESERT uses symbolic methods based on ordered binary decision diagrams (OBDDs) for constraint satisfaction [Ledeczi et al. 2003]. Therefore, using DESERT we can quickly prune a large design space to identify designs that meet the specified design and performance constraints. However, DESERT is not an optimization tool; it selects a set of designs. Therefore, we use HiPerE to evaluate the designs selected by DESERT. HiPerE evaluates system-level energy dissipation and latency. In order to provide a rapid estimate, HiPerE operates at task-level abstraction of the application. In addition to the task execution cost, various other aspects considered by HiPerE for accurate performance estimation are data access cost, energy dissipation when a component is idle, and state transition cost (reconfiguration and voltage scaling). Additionally, both DESERT and HiPerE consider the available parallelism in the data flow graph representation while estimating latency. Our results for signal processing applications show that HiPerE estimates are within 8% of the estimates using low-level simulations [Mohanty et al. 2002]. Selection of the final design is a manual process based on the estimates provided by HiPerE. The framework allows sorting the designs based on performance estimates or generation of comparison graphs (e.g., Figure 7). While a manual exploration can be time-consuming, the number of designs to be evaluated can be controlled by modifying the constraints. As DESERT is extremely efficient, a user of our framework can continue modifying constraints to reduce the overall design space to a manageable size. Low-level simulation can be used to further evaluate designs selected manually based on the estimates generated by HiPerE. Performance estimates as identified by the low-level simulators can be fed back into the models, and design space exploration can be performed again. Model interpreters can be developed to automate the feedback process.

5. DESIGN REUSE AND EXTENSIBILITY

In this section, we discuss how the design framework supports design reuse and extensibility.

5.1 Design Reuse

The use of domain-specific modeling and the model-integrated computing approach allow us to create and reuse models across designs. The micro-, macro-, and library building blocks used to model a kernel design are stored using the model library supported by GME. Each of these models is associated with performance estimates (average power and area) for a target FPGA. Our design framework supports specification of a set of performance estimates where each estimate corresponds to a unique target FPGA. When possible, designs can use preexisting building blocks along with performance estimates, alleviating the

designer of the need to perform time-consuming low-level simulations. Such reuse is possible, as: (a) The micro- and macroblocks are selected such that they are basic design components supported by the target FPGA, independent of the functionality of the kernels being designed; and (b) it is common to use the same class of FPGA devices to design several different applications.

On the other hand, if the same application needs to be designed using a different FPGA, the designer can reuse the complete application and kernel model. Only the building blocks need be reevaluated using low-level simulators to estimate their performance for the new target FPGA. While the simulation is relatively time-consuming, our design framework can completely automate the simulation and performance estimation if a suitable low-level simulator for the new target FPGA is integrated.

Alternatively, a designer can also create a library of widely used kernel designs. For example, matrix multiply, FFT, and DFT are some of the most widely used signal processing kernels. Therefore, a designer can create domain-specific models for the aforementioned signal processing kernels and store them in the model library. These models can be reused for application specification and subsequent design space exploration.

5.2 Extending the Design Framework

The design framework supports two types of extensions. The first extension involves integration of additional simulators and design tools. Integration of a tool (or a simulator) involves generation of the required input from the models specified in the framework to drive the tool. In addition, once the tool generates output, integration optionally involves parsing the output to extract required information and storing them in the models. Towards integration of the tools, GME supports creation of model interpreters through the support of a set of well-defined C++ APIs that provides bidirectional access to the models [Software Integrated Systems 2006]. Thus, the model interpreters can extract data required by the integrated tools from the models and also update the models based on the data generated by the tools. For example, integration of ModelSim [Mentor Graphics 2007b] would involve development of a model interpreter that extracts required inputs from the models, such as the high-level implementation using VHDL and the associated test bench. Then the interpreter will invoke the compiler, and compile the design and simulator to generate performance output. The interpreter then analyzes the output to extract latency estimates and updates the model. The complexity of integrating a simulator depends on the input and output formats. For example, integration of SimpleScalar and HiPerE involved writing about 1,000 lines of C++ code.

In addition to integration of tools, the modeling paradigm associated with the design framework can also be extended. Through extension of the modeling paradigm, newer building blocks can be added. For example, currently some FPGAs provide embedded high-performance DSP blocks [Altera 2007], high-performance I/O such as Rocket I/O [Xilinx 2007b], or flash-based configuration memory [Actel 2007]. These components can be added as microblocks in the modeling paradigm to support kernel design using the previously described

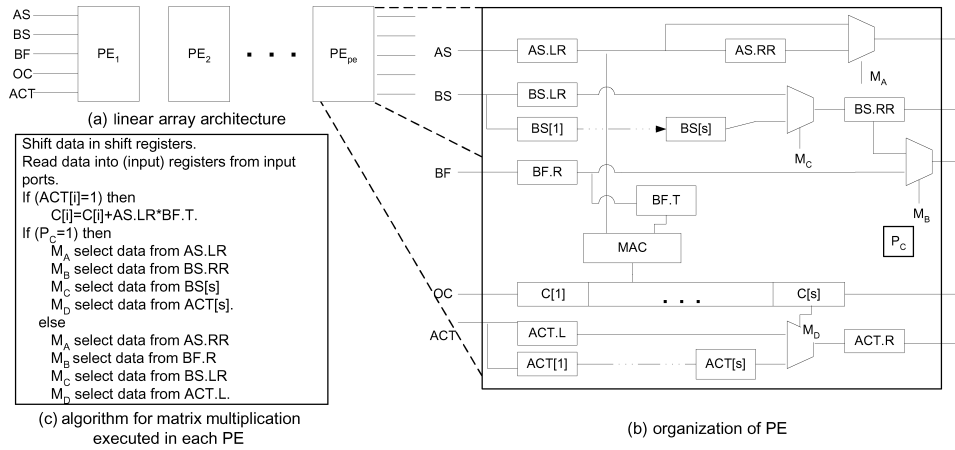


Fig. 6. Architecture and algorithm for matrix multiplication [Prasanna and Tsai 1991].

FPGAs. Extension of a modeling paradigm may require modification of existing model interpreters.

6. ILLUSTRATIVE EXAMPLES

In this section, we discuss three related examples demonstrating various capabilities of the design framework. The first example discusses the kernel-level modeling using the framework. In this example, we model a matrix multiplication algorithm using domain-specific modeling and generate a set of energy-efficient designs. In the second example, we use designs for various kernels for an adaptive beam-forming application and identify an energy-efficient design for the application. The third example demonstrates the use of the framework for evaluation of hardware devices to select the most energy-efficient choice based on duty cycle.

6.1 Energy-Efficient Designs of Matrix Multiplication Algorithm

A matrix multiplication algorithm for linear array architectures is proposed in Prasanna and Tsai [1991]. We use this algorithm to demonstrate the modeling, high-level performance estimation, and performance tradeoff analysis capabilities of the design framework. Thus it uses only Steps 1, 2, and 3 of the design flow. The focus is to generate a set of energy-efficient designs for matrix multiplication using Xilinx Virtex-II Pro.

In Step 1, the architecture and algorithm were analyzed to define the domain-specific model. Various building blocks that were identified are register, multiplexer, multiplier, adder, processing element (PE), and interconnects between the PEs. The composition of PE is described in Figure 6. Among these building blocks only PE is a library block and the rest of the components are microblocks. Component-specific parameters for the PE include the number of registers (s) and power states *ON* and *OFF*. Specifically, *ON* refers to when the multiplier (within the PE) is in *ON* state and *OFF* refers to when it is in *OFF* state. Additionally, for the complete kernel design, the number of PEs (pe) is also

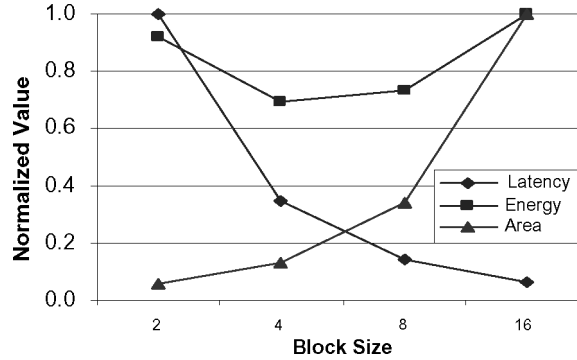


Fig. 7. Analysis of matrix multiplication algorithm.

a parameter. For $N \times N$ matrix multiplication, the range of values for s is $1 \leq s \leq N$ and for pe it is $1 \leq pe \leq N \times \lceil N/s \rceil$. For matrix multiplication with larger-sized matrices (large values of N), it is not possible to synthesize the required number of PEs due to area constraint. In such cases, block matrix multiplication is used. Therefore, block size (bs) is also a parameter.

Once the data path was modeled, we generated the cost function for power and area for the different components. Switching activity was the only parameter for power functions. To define the CPS matrices, we analyzed the algorithm to identify the operating state of each component in six different cycles. As per the algorithm [Jang et al. 2002], in each PE, the multiplier is in *ON* state for $T/\lceil n/s \rceil$ cycles and in *OFF* state for $T \times (1 - 1/\lceil n/s \rceil)$ cycles. All other components are active for the complete duration.

In Step 2, we performed simulations to estimate the power dissipated and area occupied by the building blocks. The latency (T) of this design using $N \lceil N/s \rceil$ PEs and s storage per PE [Prasanna and Tsai 1991] is $T = (N^2 + 2N \lceil N/s \rceil - \lceil N/s \rceil + 1)$. Using the latency function, component-specific power functions, and CPS matrices, we derived the system-wide energy function.

Finally, we analyzed the model to identify a set of designs that provides a tradeoff between different performance metrics. Figure 7 shows the variation of energy, latency, and area for different block sizes for 16×16 matrix multiplication. It can be observed that energy dissipation is minimum at a block size of 4 and area and latency are minimum at block sizes 2 and 16, respectively. This information is used to identify a suitable design (block size) based on latency, energy, or area requirements.

6.2 Energy-Efficient Design of an Adaptive Beam-Forming Application

In this example, we demonstrate the use of our framework for energy-efficient application design for LMS (least mean square)-based MVDR (minimum variance distortionless response) adaptive beam-forming algorithm [Ou et al. 2003; Devlin 2003; Dick 2003]. The LMS-based MVDR algorithm consists of three steps. The first step is the calculation of filter output. The second is the calculation of input signal power, correlations between signals, normalization, and

Table I. Candidate Designs and Max Update Rates Supported

No.	designs	total energy dissipated (mJ)		max weight coeff. update rate supported
		Scenario 1	Scenario 2	
1	two xc2vp2	921.6	1113.1	8
2	xc2vp2 + PowerPC	816.6	755.6	12×10^3
3	xc2vp2 + PXA 255	682.6	839.6	11×10^3
4	xc2vp2 + TI DSP	490.6	1960	1.5×10^3
5	xc2v1500	792.3	852.2	8
6	xc2vp20	968.6	1019.6	8

calculation of error signals. The third step is the update of the weight coefficients of the adaptive filter. The incoming data rate is approximately 105×10^6 samples per second [Devlin 2003]. This provides us with the latency constraint of ≤ 9.5 ns. In addition, the base station can be deployed in a power-constrained environment [Dick 2003], which makes energy dissipation an important performance metric.

Typically, the rate of weight coefficient update is once every second [Devlin 2003]. However, the rate of update depends on the stability of the environment and the application requirement. For example, an environment with high interference and path loss might require a higher rate of update to maintain the desired accuracy. Therefore, in addition to device selection, we also use the framework to identify the maximum rate of weight coefficient update that can be supported without affecting latency. The set of devices considered includes two Xilinx Virtex-II Pro devices (xc2vp2 and xc2vp20), one Xilinx Virtex-II (xc2v1500), Intel PXA 255, PowerPC 405, and TI C6711 DSP. Input specifications also include design constraints to ensure that a target hardware can consist only of either an FPGA or an FPGA with a DSP or a traditional processor. The size of the design space was 243.

For our target design problem, while there are five different target devices available, not all combinations are valid. The three valid combinations are only-DSP, only-FPGA, and a composition of an FPGA and a traditional processor or DSP. Hence, if a design is identified that contains both processors (Intel PXA 250 and PowerPC) it is not a valid design. We specified a set of constraints using the OCL (object constraint language) support provided in MILAN to ensure that only valid designs are chosen [Ledeczi et al. 2003].

HiPerE was configured to power-down components when idle if it reduces the overall energy dissipation. Table I shows the designs selected by our framework. Scenario 1 refers to coefficient update rate of once for every 105×10^6 samples. In Scenario 2, the coefficient update rate is the maximum that can be sustained by a design. We evaluated all the designs based on their performances when the system executes for a period of one second. For each design, Table I shows the energy dissipation per scenario. The last column shows the maximum update rate that can be supported.

An additional advantage of using our framework is the savings in design time due to reduction in simulation time. Cycle-accurate simulation of the processors and RT-level simulation of the FPGAs are time-consuming. For example,

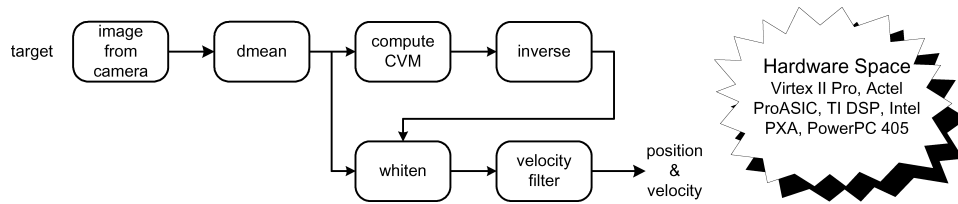


Fig. 8. Personnel detection application and the hardware design space.

simulation of the task filter using the DSP simulator takes approximately 15 minutes and using ModelSim and XPower takes about 30 minutes. Therefore, simulating all 243 designs would take 100+ hours. Using our approach, we could identify the results in minutes. There are some overheads associated with computing the performance estimations of different mappings while modeling. However, in the worst case, the overhead will be five simulations (one simulation of the three-stage application per device) as opposed to 243 simulations.

6.3 Energy-Efficient Architecture Selection for a Personnel Detection Application

We consider a personnel detection (video surveillance) application for our third example [Singer 2002]. The personnel detection algorithm is required to process input in real time and hence there is a hard latency requirement. In addition, as the system needs to be deployed in a power-constrained environment, energy dissipation is also an important metric. The application consists of five tasks, as shown in Figure 8 (image from camera is an input). The input comes from a sensor. The application design problem is to select the most energy-efficient hardware and the corresponding mapping of tasks onto the hardware components. The hardware needs to be selected from a set that consists of Xilinx Virtex-II Pro, Actel ProASIC^{PLUS}, Intel PXA 255, PowerPC 405, and TI C6711 DSP [Actel 2007; Intel 2007; IBM 2007; Xilinx 2007b]. Micron Mobile SDRAM was chosen as the memory. In order to identify the most energy-efficient solution, it is necessary to evaluate the designs based on a duty cycle which was provided as an input. For our target mapping problem, the various duty-cycle parameters are: input rate of the camera, rate of computation of the covariance matrix (CVM) and inverse, and shut-down or leave-on the components when idle.

Before applying our hierarchical design space exploration technique, we modeled the application design problem using MILAN. Modeling involved specification of the application as a hierarchical data flow graph. Modeling the target devices involved modeling individual processing components and memory. Once the application and hardware devices are modeled, possible mapping choices are indicated in the model for mapping. For example, inverse and whiten need to be computed using floating point arithmetic for which Actel ProASIC^{PLUS} is not a suitable choice (due to the large area requirement). Such constraints are indicated by not allowing inverse and whiten to be mapped onto ProASIC^{PLUS} while modeling. Additionally, while we have specified five distinct devices in the design space, the four valid combinations are a stand-alone Virtex-II Pro

Table II. Results for Personnel Detection Application

Designs	Scenario 1		Scenario 2	
	Latency (ms)	Energy (mJ)	Latency (ms)	Energy (mJ)
Virtex-II Pro only	247.33	114.06	17895	13938.1
TI DSP only	614.57	670.11	18286	9692.7
ProASIC + TI DSP	496.50	538.18	17166	8652.9

and ProASIC^{PLUS} combined with either the DSP or one of the two processors. Such constraints are specified in the model using the object constraint language supported by MILAN [Ledeczi et al. 2003].

The duty cycle was specified as follows: rate of CVM computation is 2, input rate of the camera is 0.5 Hz, and 10 instances of continuous execution, with devices to be shut-down when idle. We used DESERT and HiPerE to perform hierarchical design space exploration. As DESERT cannot be used to evaluate designs based on duty cycle requirements, we used DESERT to prune the design space, using the performance of each design for a single end-to-end execution. Therefore, we selected a latency constraint of ≤ 1 second to identify the designs that can support an input rate of 1 Hz while staying idle for at least one second (note that we want to select a set of designs). Through trial and error, an energy constraint of ≤ 860 milli Joules was chosen so as to have DESERT select 16 designs as output of the first step. The size of the initial design space was approximately 73,000. Once 16 designs were identified by DESERT, we used HiPerE to perform duty-cycle-based design space exploration to identify the best design that meets the duty cycle requirements and dissipates the minimum energy. As Intel PXA 255 and PowerPC 405 do not support floating point arithmetic (due to high latency for floating point emulation), designs that included these two processors were eliminated by DESERT. The 16 designs identified by DESERT included those that use only Virtex-II Pro or a combination of DSP and ProASIC^{PLUS}. Designs that use the same device (or device combination) differed in mapping. For Virtex-II Pro, the availability of different configurations for each task resulted in different designs. Among the designs we chose three: the most energy-efficient using only Virtex-II Pro, only TI DSP, and both TI DSP and ProASIC^{PLUS}. Results of our design space exploration are provided in Table II.

In Scenario 1, the application is executed only once (no start-up or shut-down cost included) and in Scenario 2, the application was executed based on the duty cycle requirement specified to us (Table II). Note that though the Virtex-II Pro-based design is the most energy-efficient for Scenario 1, it is the least energy-efficient for Scenario 2. This fact demonstrates the advantage of using hierarchical design space exploration and also the usefulness of higher parameter coverage (number of parameters considered while performing performance estimation or design evaluation) of HiPerE when compared with DESERT. On the other hand, the use of an optimization heuristic in the first step allows us to evaluate a design space of size 73,000 in less than a minute. Using HiPerE, it takes approximately 10 hours to estimate performance of all the designs and a tedious manual comparison of all the estimates to identify the most suitable design.

7. CONCLUSION

We discussed a design framework suitable for application design using FPGAs. The framework allows modeling of application designs, creation of model libraries, efficient performance estimation, and design space exploration. Integration of HiPerE and DESERT enables hierarchical design space exploration, which demonstrates a significant advantage over simulation- or optimization heuristics-based design techniques. In addition, the framework is extensible and presents a unified environment for application design using FPGAs. The framework relies on the performance estimates associated with the building blocks to evaluate overall performance of the designs. Therefore, the accuracy of the result is dependent on that of the estimates. However, the framework also allows integration of low-level simulators which, while time-consuming, can generate accurate performance estimates. Thus the framework provides a tradeoff between design time and accuracy. Additionally, duty-cycle-aware application design and device selection are two novel design techniques useful for the design of energy-efficient embedded systems.

ACKNOWLEDGMENT

We acknowledge Akos Ledeczki, James R. Davis, and Sandeep Neema for helpful discussions regarding GME and MIC in implementing the design framework and Seonil Choi and Jingzhao Ou for input regarding the models.

REFERENCES

- ACTEL. 2007. Actel ProASIC^{PLUS}—The nonvolatile reprogrammable gate array. <http://www.actel.com/products/proasicplus/default.aspx>.
- ALMAGOR, L., COOPER, K., GROSUL, A., HARVEY, T., REEVES, S., SUBRAMANIAN, D., TORCZON, L., AND WATERMAN, T. 2004. Finding effective compilation sequences. In *Proceedings of the Conference on Language Compilers and Tools for Embedded Systems*.
- ALTERA. 2007. Altera Stratix, Stratix-II, hardcopy. <http://www.altera.com/products/devices/>.
- BAZARGAN, K., KASTNER, R., OGRENCI, S., AND SARRAFZADEH, M. 2000. A C to hardware/software compiler. In *Proceedings of the Conference on Field-Programmable Custom Computing Machines*.
- BENINI, L., BOGLIOLO, A., AND MICHELI, G. 2000. A survey of design techniques for system-level dynamic power management. In *IEEE Trans. Very Large Scale Integration Syst.* 8, 3.
- BHATTACHARYYA, S., MURTHY, P., AND LEE, E. A. 1999. Synthesis of embedded software from synchronous dataflow specifications. *J. Very Large Scale Integration Signal Proc. Syst.* 21, 2.
- BONDALAPATI, K. AND PRASANNA, V. K. 2000. Loop pipelining and optimization for reconfigurable architectures. In *Proceedings of the Reconfigurable Architectures Workshop*.
- CELOXICA LIMITED. 2005. The Handel-C language. <http://www.celoxica.com/>.
- CHOI, S., GOVINDU, G., JANG, J., AND PRASANNA, V. K. 2003. Energy-Efficient and parameterized designs for fast Fourier transform on FPGAs. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*.
- CHOI, S., JANG, J., MOHANTY, S., AND PRASANNA, V. K. 2002. Domain-Specific modeling for rapid system-wide energy estimation of reconfigurable architectures. *J. Supercomput.* 26, 3 (Nov.), 259–261.
- DEVLIN, M. 2003. Product focus DSP: How to make smart antenna arrays. *Xcell J. Q1*.
- DICK, C. 2003. FPGA: Enabling the software/reconfigurable radio. In *Proceedings of the 14th International Conference on Field-Programmable Logic and Applications* (Antwerp, Belgium, Aug.–Sept.). Lecture Notes in Computer Science, vol. 3203. Springer Verlag.

- DOUCET, F., OTSUKA, M., SHUKLA, S., AND GUPTA, R. 2002. An environment for dynamic component composition for efficient co-design. In *Proceedings of the Conference on Design Automation and Test in Europe*.
- GHLIASI, S., NAHAPETIAN, A., AND SARRAFZADEH, M. 2004. An optimal algorithm for minimizing runtime reconfiguration delay. *ACM Trans. Embedded Comput. Syst.* 3, 2 (May), 237–256.
- GUO, Z., NAJJAR, W., VAHID, F., AND VISSERS, K. 2004. A quantitative analysis of the speedup factors of FPGAs over processors. In *Proceedings of the Symposium on Field-Programmable Gate Arrays*.
- IBM. 2007. PowerPC 405 embedded cores. http://www-3.ibm.com/chips/techlib/techlib.nsf/products/PowerPC_405_Emb%eddeds_Cores.
- INTEL. 2007. PXA 255 processor. <http://www.intel.com/design/intelxscale/>.
- JANG, J., CHOI, S., AND PRASANNA, V. K. 2002. Energy-Efficient matrix multiplication on FPGAs. In *Proceedings of the Field Programmable Logic and Applications*.
- JOHNSTON, W., HANNA, J., AND MILLAR, R. J. 2004. Advances in dataflow programming languages. *ACM Comput. Surv.* 36, 1.
- JONES, A., BAGCHI, D., PAL, S., TANG, X., CHOUDHARY, A., AND BANERJEE, P. 2002. PACT HDL: A C compiler with power and performance optimizations. In *Proceedings of the Conference on Compilers, Architectures and Synthesis for Embedded Systems*.
- KIRK, D., ROPER, M., AND WOOD, M. 2002. Defining the problems of framework reuse. In *Proceedings of the Computer Software and Applications Conference*.
- LEDECCI, A., DAVIS, J., NEEMA, S., AND AGRAWAL, A. 2003. Modeling methodology for integrated simulation of embedded systems. *ACM Trans. Model. Comput. Simul.* 13, 1 (Jan.), 82–103.
- LEE, E. A. AND MESSERSCHMITT, D. G. 1987. Synchronous data flow. *Proc. IEEE* 75.
- MAESTRE, R., KURDAHI, F., FERNANDEZ, M., HERMIDA, R., BAGHERZADEH, N., AND SINGH, H. 2001. A framework for reconfigurable computing: Task scheduling and context management. *IEEE Trans. Very Large Scale Integration Des.* 9, 6, 858–873.
- MAMIDIPIKA, M., KHOURI, M., DUTT, N., AND ABADIR, M. 2003. IDAP: A tool for high level power estimation of custom array structures. In *Proceedings of the Conference on Computer Aided Design*.
- MCGREGOR, G., ROBINSON, D., AND LYSAGHT, P. 1998. Hardware/Software co-design environment for reconfigurable logic systems. In *Proceedings of the Conference on Field-Programmable Logic and Applications*.
- MENTOR GRAPHICS. 2007a. Seamless hardware/software co-verification and FPGA advantage. <http://www.mentor.com/fpga-advantage/>.
- MENTOR GRAPHICS. 2007b. ModelSim. <http://www.model.com/>.
- MILAN. 2007. Model-Based integrated simulation. <http://milan.usc.edu/>.
- MOHANTY, S. AND PRASANNA, V. 2003. A hierarchical approach for energy efficient application design using heterogeneous embedded systems. In *Proceedings of the Conference on Compilers, Architectures and Synthesis for Embedded Systems*. 243–254.
- MOHANTY, S. AND PRASANNA, V. 2004. An algorithm designer’s workbench for platform FPGAs. In *Proceedings of the Conference on Field Programmable Logic and its Application*.
- MOHANTY, S., PRASANNA, V. K., NEEMA, S., AND DAVIS, J. 2002. Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation. In *Proceedings of the Conference on Language Compilers and Tools for Embedded Systems*.
- MUKHERJEE, R. AND MEMIK, S. 2004. Power-Driven design partitioning. In *Proceedings of the Conference on Field Programmable Logic and its Application*.
- OU, J., CHOI, S., AND PRASANNA, V. K. 2003. Performance modeling of reconfigurable SoC architectures and energy-efficient mapping of a class of applications. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*.
- PRASANNA, V. K. AND TSAI, Y. 1991. On synthesizing optimal family of linear systolic arrays for matrix multiplication. *IEEE Trans. Comput.* 40, 6, 770–774.
- SHANG, L. AND JHA, N. K. 2001. High-Level power modeling of CPLDs and FPGAs. In *Proceedings of the International Conference on Computer Design*.
- SHENOY, N., CHOUDHARY, A., AND BANERJEE, P. 2001. An algorithm for synthesis of large time-constrained heterogeneous adaptive systems. *ACM Trans. Des. Autom. Electron. Syst.* 2, 6, 207–225.

- SHIRAZI, N., LUK, W., AND CHEUNG, P. 1998. Automating production of run-time reconfigurable designs. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*.
- SIMPLESCALAR. 2007. The SimpleScalar tool set. <http://www.simplescalar.com/>.
- SINGER, P. 2002. The optimal detector. In *SPIE Conference: Signal and Data Processing for Small Targets*.
- SOFTWARE INTEGRATED SYSTEMS. 2006. GME-4. Generic Modeling Environment. <http://www.isis.vanderbilt.edu/Projects/gme/>.
- SRIVASTAVA, N., TRAHAN, J., VAIDYANATHAN, R., AND RAI, S. 2003. Adaptive image filtering using run-time reconfiguration. In *Proceedings of the Reconfigurable Architectures Workshop*.
- STAMMERMANN, A., KRUSE, L., NEBEL, W., PRATSCH, A., SCHMIDT, E., SCHULTE, M., AND SCHULZ, A. 2001. System level optimization and design space exploration for low power. In *Proceedings of the International Symposium on System Synthesis*.
- SYNOPSIS. 2007. Synopsis design compiler FPGA. <http://www.synopsys.com/fpga/>.
- SYSTEMC. 2007. SystemC HDL. <http://www.systemc.org/>.
- SZTIPANOVITS, J. AND KARSAI, G. 1999. Model-Integrated computing. *IEEE Comput. Mag.* 30, 4 (Apr.), 110–111.
- XILINX. 2007a. Xilinx system generator for Simulink (Matlab). http://www.xilinx.com/products/software/sysgen/product_details.htm.
- XILINX. 2007b. Xilinx Virtex-II and Virtex-II Pro. <http://www.xilinx.com/>.

Received October 2004; revised August 2006; accepted December 2006