

An Algorithm Designer’s Workbench for Platform FPGAs^{*}

Sumit Mohanty and Viktor K. Prasanna

Electrical Engineering Systems
University of Southern California, CA, USA
{smohanty, prasanna}@usc.edu

Abstract. Growing gate density, availability of embedded multipliers and memory, and integration of traditional processors are some of the key advantages of Platform FPGAs. Such FPGAs are attractive for implementing compute intensive signal processing kernels used in wired as well as wireless mobile devices. However, algorithm design using Platform FPGAs, with energy dissipation as an additional performance metric for mobile devices, poses significant challenges. In this paper, we propose an algorithm designer’s workbench that addresses the above issues. The workbench supports formal modeling of the signal processing kernels, evaluation of latency, energy, and area of a design, and performance tradeoff analysis to facilitate optimization. The workbench includes a high-level estimator for rapid performance estimation and widely used low-level simulators for detailed simulation. Features include a confidence interval based technique for accurate power estimation and facility to store algorithm designs as library of models for reuse. We demonstrate the use of the workbench through design of matrix multiplication algorithm for Xilinx Virtex-II Pro.

1 Introduction

High processing power, programmability, and availability of a processor for control intensive tasks are some of the unique advantages for Platform FPGAs which integrate traditional processors into the FPGA fabric [13]. Therefore, Platform FPGAs are attractive for implementing complex and compute intensive applications used in wired and wireless devices [13]. Adaptive beam forming, multi-rate filters and wavelets, software defined radio, image processing, etc. are some of the applications that target reconfigurable devices [11]. Efficient design of an application requires efficient implementation of constituent kernels. In this paper, kernel refers to a signal processing kernel such as matrix multiplication, FFT, DFT, etc. Algorithm design for a kernel refers to the design of a datapath and the data and control flow that implements the kernel.

Even though FPGA based systems are not designed for low-power implementations, it has been shown that energy-efficient implementation of signal

^{*} This work is supported by the DARPA Power Aware Computing and Communication Program under contract F33615-C-00-1633.

processing kernels is feasible using algorithmic techniques [3], [5]. However, the major obstacle to the widespread use of Platform FPGAs is the lack of high-level design methodologies and tools [7]. Moreover with energy dissipation as an additional performance metric for the mobile devices, algorithm design using such FPGAs is difficult. In this paper, we discuss the design of a workbench for the algorithm designers based on the domain specific modeling technique [3] that addresses the above issues. A *workbench* in the context of algorithm design refers to a set of tools that aid an algorithm designer in designing energy-efficient algorithms. Although the workbench supports algorithm design for any kernel, we primarily target kernels that can be implemented using regular loop structures enabling maximum exploitation of hardware parallelism.

The workbench enables parameterized modeling of the datapath and the algorithm. Modeling using our workbench follows a hybrid approach, which starts with a top-down analysis and modeling of the datapath and the corresponding algorithm followed by analytical formulation of cost functions for various performance metrics such as energy, area, and latency. The constants in these functions are estimated through a bottom-up approach that involves profiling individual datapath components through low-level simulations. The model parameters can be varied to understand performance tradeoffs. The algorithm designer uses the tools integrated in the workbench to estimate performance, analyze the effect of parameter variation on performance, and identify optimization opportunities. Our focus in designing the workbench is not to develop new techniques for compilation of high-level specifications onto FPGAs, rather to formalize some of the design steps such as modeling, tradeoff analysis, estimation of various performance metrics using the available solutions for simulation and synthesis.

The workbench facilitates both high-level estimation and low-level simulation. High-level refers to the level of abstraction where performance models of the algorithm and the 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. The workbench is developed using Generic Modeling Environment, GME (GME 3), a graphical tool-suite, that enables development of a modeling language for a domain, provides graphical interface to model specific problem instances for the domain, and facilitates integration of tools that can be driven through the models [4].

The following section discusses some related work. Section 3 describes our algorithm design methodology. Section 4 discusses the design of the workbench. Section 5 demonstrates the use of the workbench. We conclude in Section 6.

2 Related Work

Several tools are available for efficient application design using FPGAs. Simulink and Xilinx System Generator provide a high-level interface to design applications using pre-compiled libraries of signal processing kernels [12]. Other tools such as [6] and [13] 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. For Platform FPGAs, the ongoing design automation efforts use available support for design synthesis onto the FPGA and processor and focus on communication synthesis through shared memory. However, a high-level programming model suitable for algorithmic analysis still remains to be addressed.

Several C/C++ language based approaches such as SystemC, Handel-C, SpecC are primarily aimed at making the C language usable for hardware and software design and allow efficient and early simulation, synthesis, and/or verification [1], [2]. However, these approaches do not facilitate modeling of kernels at a level of abstraction that enables algorithmic analysis. Additionally, generating source code and going through the complete synthesis process to evaluate each algorithm decision is time consuming.

In contrast, our approach enables high-level parameterized modeling for rapid performance estimation and efficient tradeoff analysis. Using our approach, the algorithm designer does not have to 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 the initial stages of the design process and hence is more efficient when the final design needs to meet strict latency, energy, or area requirements.

3 Algorithm Design Methodology

Our algorithm design methodology is based on domain specific modeling, a technique for high-level modeling of FPGAs, developed by Choi et al. [3]. This technique has been demonstrated successfully for designing energy efficient signal processing kernels [3], [5]. A domain, in the context of domain specific modeling, refers to a class of architectures such as uniprocessor, linear array, etc. and the corresponding algorithm that implements a specific kernel [3].

In our methodology, the designer initially creates a model using the domain specific modeling technique for the kernel for which an algorithm is being designed. This model consists of RModules, Interconnects, component specific parameters and power functions, component power state matrices, and a system-wide energy function. We have extended the model to include functions for latency and area as well. A *Relocatable Module (RModule)* is a high-level architecture abstraction of a computation or storage module. *Interconnect* represents the resources used for data transfer between the RModules. A component (building block) can be a RModule or an Interconnect. *Component Power State (CPS) matrices* capture the power state for all the components in each cycle.

Parameter estimation refers to the estimation of area and power functions for the components. Power and area functions capture the effect of component specific parameters on the average power dissipation and area of the component respectively. Latency is implicit in the algorithm specification and is also specified as a function. Ultimately, component specific area and power functions and

the latency function are used to derive system-wide (for the kernel) energy, area, and latency functions.

Following parameter estimation, the designs are analyzed for performance tradeoffs as follows: a) each component specific parameter is varied to observe the effect on different performance metrics, b) if there exists alternatives for a building block then each alternate is evaluated for performance tradeoffs, and c) fraction of total energy dissipated by each type of component is evaluated to identify candidates for energy optimization. Based on the above analysis, the algorithm designer identifies the optimizations to be performed.

The workbench is used to assist designing using this methodology. Various supports include a graphical modeling environment based on domain specific modeling, integration of widely used simulators, curve-fitting for cost function estimation, and tradeoff analysis. In addition, the workbench integrates a high-level estimator, kernel performance estimator, that rapidly estimates latency, energy, and area for a kernel to enable efficient tradeoff analysis. The workbench also supports storage of various components as a library for reuse and a confidence interval based technique for statistically accurate power estimation.

4 Algorithm Designer’s Workbench

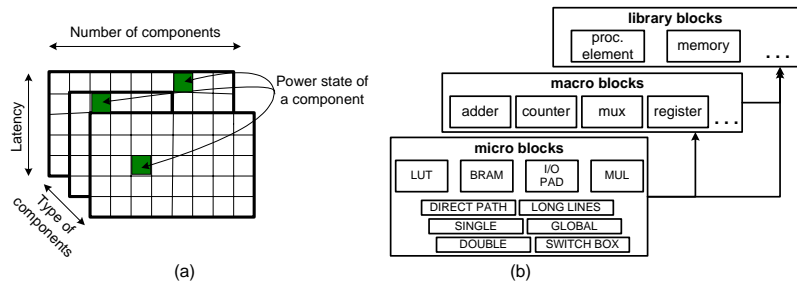


Fig. 1. Component Power State Matrices and Hierarchical Modeling

We have used GME to create the modeling environment for the workbench [4]. GME provides support to define a metamodel (modeling paradigm). A metamodel contains syntactic, semantic, and visualization specifications of the target domain. GME configures itself using the metamodel to provide a graphical interface to model specific instances of the domain. GME enables integration of various tools to the modeling environment. Integration, in this context refers, to being able to drive the tools from the modeling environment. In the following, we discuss various aspects of the workbench in detail.

4.1 Modeling

We provide a hierarchical modeling support to model the datapath. The hierarchy consists of three types of components; micro, macro, and library blocks.

A micro block is target FPGA specific. For example, as shown in Figure 1 (b), the micro blocks specific to Xilinx Virtex II Pro are LUT, embedded memory cell, I/O Pad, embedded multiplier, and interconnects [13]. In contrast, for Actel ProASIC 500 series of devices, there will be no embedded multiplier. Macro blocks are basic architecture components such as adders, counters, multiplexers, etc. designed using the micro blocks. A library block is an architecture component that is used by some instance of the target class of architectures associated with the domain. For example, if linear array of processing elements (PE) is our target architecture, a PE is a library block. Both macro and library blocks are also referred to as composite blocks. We have developed a basic metamodel using GME that provides templates for different kind of building blocks and associated parameters. Once the target device and the kernel are identified, prior to using the workbench, 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. Each building block is associated with a set of component specific parameters (Section 3). States is one such parameter which refer to various operating states of each building block. Currently, we model only two states, *ON* and *OFF* for each micro block. In *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 2 states due to different combination of states of the constituent micro blocks. Power is always specified as a function with switching activity as the default parameter.

Given an algorithm, Component Power State (CPS) matrices capture the operating 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 two dimensional matrices are constructed where the i -th matrix is of size $T \times n_i$ (Figure 1(a)). An entry in a CPS matrix represents the operating state of a component during a specific cycle and is determined by the algorithm.

Modeling based on the technique described above has the following advantages: a) the model separates the low-level simulation and synthesis tools from high-level algorithmic analysis techniques, b) various parameters are exposed at high-level thus enabling algorithm-level tradeoff analysis, c) performance models for energy, area, and latency are generated in the form of parameterized functions which allows rapid performance estimation, and d) using GME's ability to store models in a database, models for various kernels are stored and reused during the design of other kernels that share the same building blocks.

4.2 Performance Estimation

Our workbench supports performance evaluation through rapid high-level estimation as well as low-level simulation using third-party simulators. We have developed a high-level estimation tool, a kernel performance estimator, to estimate different performance metrics associated with the algorithm. The input

to the kernel performance estimator is the model of the datapath and the CPS 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% (default value used by Xilinx XPower [13]).

Area of the design is evaluated as the sum of the individual areas of the building blocks that constitute the datapath. Latency estimate is implicit in the CPS matrices. Energy dissipation is estimated as a function of the CPS matrices and datapath. Overall energy dissipation is 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 in a specific cycle, we use the power function for the operating state specified in the CPS matrices and duration of each cycle to evaluate the energy dissipated. Extending the above analysis for each component over the complete execution period, we evaluate the overall energy dissipation for the design. The kernel performance estimator is based on the above technique and is integrated into the workbench. Once a model is defined, it is possible to automatically invoke the estimator to generate the performance estimates and update the model based on these estimates. The estimator operates at a level of abstraction where the domain specific modeling technique is defined. Therefore, the estimator is not affected by the modifications to the basic metamodel as described in Section 4.1.

Low-level simulation is used in our methodology to estimate various component specific parameters. Low-level simulation is supported in the workbench through widely used simulators available for the FPGAs such as ModelSim, Xilinx XPower, Xilinx Power Estimator, etc. [13]. Different simulators have different input requirements. For example, Power Estimator requires a list of different modules used in a design, expected switching activity, area in CLB slices, and frequency of operation to provide a rapid and coarse estimate of average power. In contrast, ModelSim and XPower accept placed and routed design and input waveforms to perform fairly accurate estimation of power and latency. Therefore, we have added capability in our modeling environment to specify VHDL source code and input vectors as files. Derivation of cost function involves simulation of the design at different instances where each instance refers to a combination of parameter values and curve-fitting to generate functions.

Energy dissipation depends on various factors such as voltage, frequency, capacitance, and switching activity. Therefore, simulating once using a random test vector may not produce statistically accurate results. We use confidence intervals to estimate average power and confidence in the estimate to generate statistically significant results. The approach uses results from multiple simulations performed using a set of randomly generated inputs (Gaussian distribution is assumed) and computes the mean of the performance values and the confidence associated with the mean. Given a synthesized design and set of input test vectors, the workbench automatically performs multiple simulations and evaluates the mean and the confidence interval and updates the model using the results.

4.3 Modeling and Simulating the Processors

Kernels that are control intensive are potential candidates for design using both the FPGA and the processor. We assume shared memory communication between the FPGA and the processor. We model computations on the processor as functions. A function is implemented using high-level languages such as C and is compiled and loaded into the instruction memory. It is assumed that the processor is aware of the memory location to read input and write output. Once a function is invoked the FPGA stalls until the results are available.

Functions are specified in the CPS matrices. A function specified in a cell (i, j) refers to invoking the function in the i th cycle. We model each function as time taken and energy dissipated during execution. Therefore, whenever a function is encountered in the CPS matrices, the kernel performance estimator includes the latency and energy values associated with the function to the overall energy or latency estimates. We use the available simulators for the processors to estimate latency and energy values for each function. For example, ModelSim, which uses the SWIFT model for PowerPC, can be used to simulate the processor and estimate latency [13]. We evaluate energy dissipation by scaling the latency value based on the vendor provided estimates. For example, PowerPC on Virtex-II Pro consumes 0.9 mW per MHz. Using this estimate, a program that executes over 9×10^8 cycles will have a latency of 3 seconds and power dissipation of approximately 810 mW when the processor is operating at 300 MHz.

4.4 Design Flow

We briefly describe the design flow for algorithm design using our workbench. We assume that designer has already identified a suitable domain for the kernel to be designed and the target Platform FPGA.

Workbench Configuration (1): In this step, the designer analyzes the kernel to define a domain specific model. However, the designer does not derive any of the functions associated with the model. The designer only identifies the RModules and Interconnects and classifies them as micro, macro, and library blocks and also identifies the associated component specific parameters. Following this, the designer modifies the basic metamodel to configure GME for modeling using the workbench. The building blocks identified for one kernel can also be used for other kernels implemented using the same target FPGA. Therefore, modifications to the metamodel are automatically saved for future usage.

Modeling (2): The model of the datapath is graphically constructed in this step using GME. GME provides the ability to drag and drop modeling constructs and connect them appropriately to specify the structure of the datapath. If appropriate simulators are integrated, the designer can 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. The workbench facilitates building a library of components to save models of the building blocks and associated performance estimates (see *Step 3*) for reuse during the design of other algorithms.

Parameter Estimation (3): 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. The simulations are performed off-line or, if required simulator is integrated, automatically using specified high-level scripts. Latency functions are estimated using the CPS matrices. System-wide energy and area functions are estimated using the latency function and component specific power and area functions.

Tradeoff Analysis and Optimization (4): In this step, the designer uses the workbench to understand various performance tradeoffs as described in Section 3. While the workbench facilitates generation of the comparison graphs, the designer is responsible for specific design decisions based on the graphs. Once the design is modified based on the decisions, kernel performance estimator is used for rapid verification. Tradeoff analysis and optimization is performed iteratively till the designer meets the performance goals.

5 Illustrative Example: Energy-Efficient Design of Matrix Multiplication Algorithm

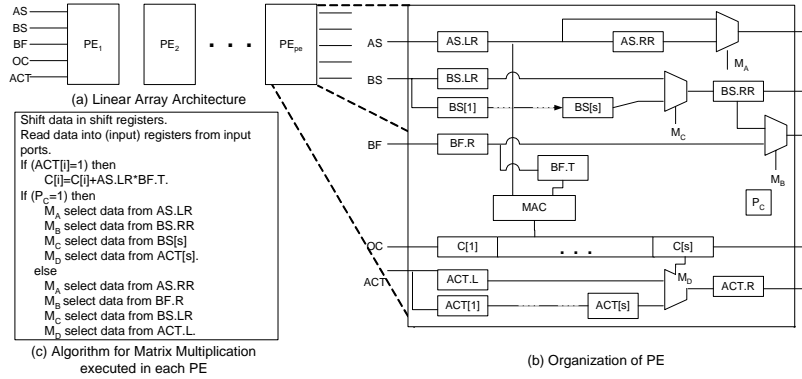


Fig. 2. Architecture and Algorithm for Matrix Multiplication [10]

A matrix multiplication algorithm for linear array architectures is proposed in [10]. We use this algorithm to demonstrate modeling, high-level performance estimation, and performance tradeoff analysis capabilities of the workbench. The focus is to generate an energy efficient design for matrix multiply using Xilinx Virtex-II Pro. The matrix multiplication algorithm and the architecture are shown in Figure 2. In this experiment, only FPGA is considered while designing the algorithm.

In *step 1*, the architecture and the 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. Among these building blocks only the PE is a library block

and the rest of the components are micro blocks. Component specific parameters for the PE include number of register (s) and power states ON and OFF . ON refers to the state when the multiplier (within the PE) is in ON state and OFF refers to the state when the multiplier is in OFF state. Additionally, for the complete kernel design number of PEs (pe) is also 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 $1 \leq pe \leq N(\lceil N/s \rceil)$. For matrix multiplication with larger size 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. Based on the above model, the basic meta-model was enhanced to support modeling of linear array architecture for matrix multiply.

Step 2 involved modeling using the configured GME. Once the datapath 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 different cycles. As per the algorithm shown in Figure 2 (c), in each PE, the multiplier is in ON state for $T/(\lceil n/s \rceil)$ cycles and is in OFF state for $T \times (1 - 1/\lceil n/s \rceil)$ cycles [10]. All other components are active for the complete duration.

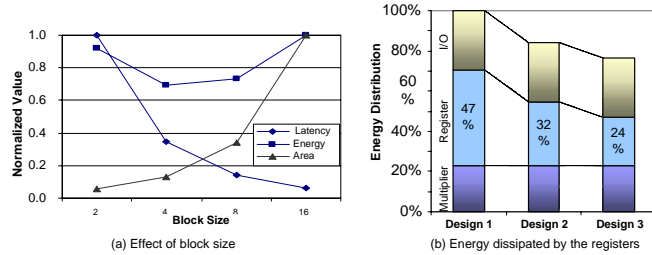


Fig. 3. Analysis of Matrix Multiplication Algorithm

In *Step 3*, we performed simulations to estimate the power dissipated and area occupied by the building blocks. Currently, we have integrated ModelSim, Xilinx XPower, and Xilinx Power Estimator to the workbench [13]. The latency (T) of this design using $N \lceil N/s \rceil$ PEs and s storage per PE [10] 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 performed a set of tradeoff analyses to identify suitable optimizations. Figure 3 (a) shows the variation of energy, latency, and area for different block sizes for 16×16 matrix multiplication. It can be observed that energy is minimum at a block size of 4 and area and latency are minimum at block size 2 and 16 respectively. This information is used to identify a suitable design (block size) based on latency, energy, or area requirements. Figure 3 (b) shows energy distribution among multipliers, registers, and I/O pads for three different

designs. Design 1 corresponds to the original design described in [10] and Design 2 and 3 are low energy variants discussed in [5]. Using Figure 3, we identify that the registers dissipate the maximum energy and select them as candidates for optimization. Optimizations considered include reduction of number of registers through analysis of data movements (Design 2) and use of CLB based SRAMs instead of registers to reduce energy dissipation (Design 3). Details of the optimized algorithm are available in [5], [9].

Using the workbench, an optimized matrix multiplication algorithm was designed and compared against the Xilinx IP core for matrix multiplication. The best design obtained using our approach achieved 68% reduction energy dissipation, 35% reduction in latency while occupying $2.3 \times$ area when compared with the design provided by Xilinx [9].

6 Conclusion

We discussed an algorithm designer's work bench suitable for a general class of FPGA and Platform FPGA devices. The work discussed in this paper is part of the MILAN project. MILAN addresses the issues related to the design of end-to-end applications [8]. In contrast, the workbench addresses the issues pertaining to the design of application kernels only. The workbench is developed as a stand-alone tool with plans for integration into the MILAN environment.

References

1. Cai, L., Olivarez, M., Kritzinger, P., and Gajski, D: C/C++ Based System Design Flow Using SpecC, VCC, and SystemC. Tech. Report 02-30, UC, Irvine, June 2002.
2. The Handel-C language. <http://www.celoxica.com/>
3. Choi, S., Jang, J., Mohanty, S., and Prasanna, V.K.: Domain-Specific Modeling for Rapid System-Wide Energy Estimation of Reconfigurable Architectures. Engineering of Reconfigurable Systems and Algorithms, 2002.
4. Generic Modeling Environment. <http://www.isis.vanderbilt.edu/Projects/gme/>
5. Jang, J., Choi, S., and Prasanna, V.K.: Energy-Efficient Matrix Multiplication on FPGAs. Field Programmable Logic and Applications, 2002.
6. Mentor Graphics FPGA Advantage. <http://www.mentor.com/fpga-advantage/>
7. McGregor, G., Robinson, D., and Lysaght, P.: A Hardware/Software Co-design Environment for Reconfigurable Logic Systems. Field-Programmable Logic and Applications, 1998.
8. Model-based Integrated Simulation. <http://milan.usc.edu/>
9. Mohanty, S., Choi, S., Jang, J., and Prasanna, V.K.: A Model-based Methodology for Application Specific Energy Efficient Data Path Design using FPGAs. Conference on Application-Specific Systems, Architectures and Processors, 2002.
10. Prasanna, V.K. and Tsai, Y.: On Synthesizing Optimal Family of Linear Systolic Arrays for Matrix Multiplication. IEEE Tran. on Computers, Vol. 40, No. 6, 1991.
11. Srivastava, N., Trahan, J., Vaidyanathan, R., and Rai, S.: Adaptive Image Filtering using Run-Time Reconfiguration. Reconfigurable Architectures Workshop, 2003.
12. System Generator for Simulink. http://www.xilinx.com/products/software/sys-gen/product_details.htm.
13. Xilinx Virtex-II Pro and Xilinx Embedded Development Kit (EDK). <http://www.xilinx.com/>