MILAN: A Model Based Integrated Simulation Framework for Design of Embedded Systems*

A. Bakshi
Dept. of EE-Systems
University of Southern California
Los Angeles, CA
abakshi@usc.edu

V. K. Prasanna
Dept. of EE-Systems
University of Southern California
Los Angeles, CA
prasanna@usc.edu

A. Ledeczi
Institute for Software Integrated Systems
Vanderbilt University
Nashville, TN
akos.ledeczi@vanderbilt.edu

ABSTRACT

We present MILAN, a model based extensible framework that facilitates rapid, multigranular performance evaluation of a large class of embedded systems, by seamlessly integrating different widely used simulators into a unified environment. MILAN provides a formal paradigm for specification of structural and behavioral aspects of embedded systems, an integrated model-based approach, and a unified software environment for system design and simulation. This paper provides an overview of MILAN, discusses the Model Integrated Computing philosophy, and illustrates the high-level modeling concepts being developed in the MILAN project for embedded systems design and evaluation.

1. INTRODUCTION

Design of an embedded system typically involves a balance between high performance requirements such as latency and throughput, and constraints on power dissipation, area, design cost etc. System-on-Chip (SoC) architectures combine high performance with low power consumption, and are thus suitable candidates for development of these systems. A SoC architecture integrates multiple functional units (RISC processors, configurable logic, customized functional units, on-chip memories, etc.) of a complete end product onto a single chip. However, the heterogeneous nature of the components makes efficient mapping of applications onto these architectures a challenging task. While the availability of these heterogeneous components on a single platform provides the designer with increased flexibility in mapping applications, efficient design requires an in-depth understanding of their functionality, interactions, and possible tradeoffs in co-design [8, 15].

*This work is supported by the DARPA Power Aware Computing and Communication Program under contract F33615-C-00-1633 monitored by Wright Patterson Air Force Base.

State-of-the-art methodologies adopt an ad hoc approach towards system design. Programming models and design tools for each hardware component are utilized independently to map an application task onto that resource, and integration is performed much later in the design cycle. Lack of system-wide performance evaluation and optimization tools increases the complexity of the design process. System-wide performance analysis is typically a manual process involving the use of component specific simulators in isolation. This is tedious, since each simulator requires a different input/output interface and execution platform. Moreover, separate simulators are utilized for estimating multiple performance metrics even for the same component. This results in sub-optimal solutions because global, multi-objective optimizations are difficult, and traversing the entire design space is too time-consuming to be practical.

This paper introduces MILAN[1], a **M**odel based **I**ntegrated simu**LA**tio**N** framework to facilitate embedded system design and optimization. MILAN is a collaborative project between the University of Southern California (USC) and the Vanderbilt University (VU).

The MILAN modeling paradigms facilitate seamless integration of a variety of simulators at multiple levels of granularity, into the framework. A single graphical user interface allows designers to specify different aspects of embedded system hardware and software, and performance requirements. This information is automatically translated into the various simulator-specific input formats. The results of individual simulations are interpreted in the global context to provide system-wide estimates of different performance metrics. A design space exploration tool, coupled with a high-level, coarse-grained performance estimator facilitates evaluation of a large number of possible solutions and eliminates those that do not meet system requirements. The high-level estimator is designed to provide rapid, coarse-grained performance statistics, possibly at the expense of accuracy. MILAN will provide a feedback mechanism for refining high-level model parameters based on results of low-level simulation, for greater accuracy in future simulations. This framework incorporates power as an important design metric. Power estimation and optimization is supported through integration of existing component specific power simulators, based on system-wide power models.

[1]milan (hindi): meeting, unification. *http://milan.usc.edu/*

The focus of the project is on developing formal modeling paradigms that will enable simulator integration and efficient application-to-architecture mapping through automatic design space exploration. The architectures modeled in MILAN consist of tightly-coupled, heterogeneous, digital components, specifically, emerging SoC architectures such as [7, 20, 21]. Initially, applications will be modeled as hierarchical signal flow graphs with no control dependences between tasks. Even with this restriction, the application model is capable of representing most streaming applications including signal and image processing applications.

The rest of this paper is organized as follows. Section 2 describes related work in embedded system design. Section 3 has an overview of the MILAN architecture. Sections 4, 5, and 6 provide technical details on system representation, design space exploration and simulator integration in MILAN. Section 7 has the concluding remarks.

2. RELATED WORK

Several research efforts have focused on the study of system-level tradeoffs and performance optimization techniques. Notably, various hardware/software co-design groups have developed formal modeling techniques for the design of embedded systems. The POLIS project [3] provides a hardware-software co-synthesis tool for design and synthesis of embedded micro-controllers. A single abstract representation, the co-design finite state machine model [9], was proposed in POLIS for specification, partitioning, and implementation of such systems. Chinook [10] is a hardware-software co-synthesis CAD tool for control dominated, reactive embedded systems, with emphasis on IP integration. This effort has studied the issues of IP composition, communication synthesis, and rapid evaluation. The POLIS and Chinook frameworks enable detailed low-level simulation, evaluation, and synthesis of an embedded system. However, they model point solutions in the sense that the user is required to specify a particular hardware-software partitioning of the application. The Ptolemy project [16] provides a formalism to express various computational models related to embedded systems and supports heterogeneous concurrent modeling. Ptolemy also supports modeling of interactions between sets of components that are represented by different computational models. Simulink 4 [18] is a commercial tool that provides an extensive graphical interface to MATLAB for interactive modeling and functional simulation.

MILAN leverages some of the concepts related to modeling, co-simulation, and system level analysis techniques, from the above projects. The primary focus of MILAN is to facilitate integration of heterogeneous simulators, develop/leverage models to formally represent the system structure and behavior, and provide a high-level abstraction to the system designer. MILAN enables evaluation of several performance metrics including power, with a high-level estimation tool for rapid design space exploration. MILAN also provides a single graphical user interface for all aspects of system development.

3. MILAN ARCHITECTURE OVERVIEW

MILAN adopts Model Integrated Computing (MIC) [19] as the core design philosophy. Model Integrated Computing is especially valuable for the design of computer-based systems with strong interdependence between the hardware and software components. By formally modeling all aspects (application, resource, behavior, constraints, etc.) of a system and using well-defined rules to generate new systems or manage existing ones, it is possible to avoid the errors that arise when requirements change and the system has to be redesigned or reimplemented. While the initial modeling effort might be costly compared to ad hoc approaches, the benefits are clearly visible for a system that evolves over time.

Models are essentially abstractions that allow the representation and manipulation of various aspects of the underlying system. The level of abstraction at which a system is modeled depends entirely on the intended usage of the model information. For example, the RAM model of computation assumes an infinite main memory with unit access cost. This model is used in arriving at rough estimates of time complexity of sequential applications. Any analysis that requires a more accurate estimate of application performance needs to model memory in more detail, such as row access cost, column access cost, number of memory banks, access latencies, etc. An environment that supports MIC allows designers to create domain-specific models of systems at the required level of abstraction, validate these models, and perform various computational transformations on them. Model interpreters are the software components that translate these models for use in the MILAN execution environment. A model database stores the translated information, which is used in driving various simulators. The intermediate results are also stored in the model database.

The Generic Modeling Environment (GME) is a configurable graphical tool suite supporting MIC [11]. The configuration of the environment to support domain-specific modeling is done in a formal manner through the use of metamodels. The metamodeling language is the UML class diagram notation [6]. Well-formedness rules that are also part of the metamodels are specified using the Object Constraint Language (OCL). These constraints, along with the syntactical rules of the domain language, are enforced by the automatically generated target environment. MILAN exploits the MIC technology to present an environment tailored for embedded system design, evaluation, and optimization.

Figure 1 shows the architecture of MILAN and also depicts the system design flow from the users' perspective. The graphical interface is provided by GME configured to support the modeling paradigms developed for MILAN. These modeling paradigms can be categorized into two broad classes. The first consists of resource, application and constraint models that primarily capture the system structure (layout, interconnections, and parameters). The second consists of performance and communication models that characterize those aspects of the hardware/software that enable analysis and performance estimation. These two models are not visible to the designer through GME, and therefore are not part of the design flow implied in Figure 1.

Resource models describe available hardware components and their interconnectivity in a hierarchical block diagram-like notation. *Application* models are based on a hierarchical signal flow representation with important extensions.

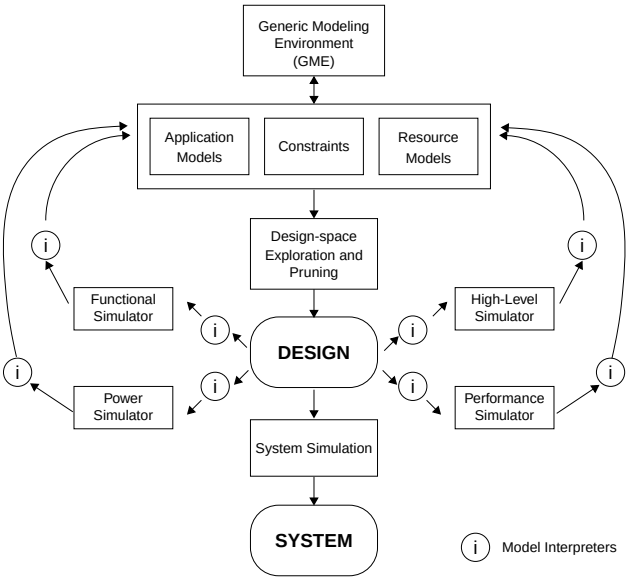

Figure 1: MILAN Architecture

Most notably, the modeling language allows for the specification of explicit design or implementation alternatives of any component. This enables modeling of the entire design space of the application as opposed to a point solution. To manage this design space, application requirements, resource constraints and other specifications are captured explicitly through OCL. *Performance* modeling of SoC architectures involves characterizing desired performance metrics of a given mapping in terms of architecture parameters. The *communication* model provides a common formalism to enable interoperability of simulators that represent the same information in different formats. Section 4 discusses these models in detail.

The design-space exploration and pruning tool takes the potentially very large design space[2] and applies the constraints using a symbolic constraint satisfaction technique to find the set of solutions that satisfy all the constraints. The goal of design space exploration is to identify a small number of valid candidate designs. To find the balance between an under-constrained and an over-constrained model is a highly iterative, human-in-the-loop process. One of the design goals of the modeling environment and the design-space exploration tools is to support this activity. The next step in the design process is to utilize the integrated simulators to simulate candidate designs one-by-one. Each supported simulator has a corresponding model translator (or model interpreter, to use MIC terminology) that configures the simulation from the system models.

MILAN supports different classes of simulators. Functional simulators, such as MATLAB or SystemC, verify the functionality of the application. The integrated high-level estimation tool provides a rapid, reasonably accurate estimate of different performance criteria of the system. The performance model is utilized to evaluate system design for a quick estimation of critical performance parameters without the use of any cycle accurate simulation. This quick estimation is enabled by use of functions or heuristics evaluating various performance attributes based on architecture parameters. For example, rapid prediction of performance and power consumption of various cache configuration is possible through a fast estimation equation generated using linear approximation [12].

Lower-level simulators such as SimpleScalar [17] are also supported. While they can be very accurate, their slow speed may prevent the simulation of all possible design choices. MILAN allows multi-level simulation, which exploits the trade-off between accuracy of results and simulation speed. Simulator integration is described in detail in Section 6.

4. MODELING PARADIGMS

As discussed earlier, various modeling paradigms are being developed for MILAN to capture the structural and behavioral aspects of an embedded system. A modeling paradigm (*metamodel* in GME nomenclature) captures the syntactic, semantic, and presentation information necessary to create models of systems within a particular domain [11]. Modeling paradigms are developed based on an exhaustive characterization of the underlying domain, and are used by the designer to instantiate domain-specific system architectures in terms of models.

4.1 Resource Model

Resource models represent the hardware components of the system. Hardware components of SoC architectures typically consist of general-purpose processor cores, application specific accelerators, interconnect, memory, and configurable logic among others. Recent advances in hardware synthesis have enabled design of hardware with various *malleable* parameters (e.g. frequency, voltage, memory size). These parameters are critical for system optimizations and need to be identified and their effect evaluated early in the design cycle.

The MILAN resource metamodel is based on a hierarchical classification of hardware components related to embedded system design, within the scope of the project. The resource metamodel will not represent analog/RF components, peripheral devices, wireless networks, etc. The top level of the hierarchy consists of the generic components - processing element, storage element, interconnect, sensor, and actuator. Although we do not model I/O components, the sensors and actuators are included to represent the source and destination of data. All possible hardware components for the target embedded system architectures in the MILAN project can be classified under one of these general categories.

To facilitate a logical structure for the resource representation, our metamodeling process follows a design technique similar to object-oriented programming. The models consist of specialized components, which inherit the parameters of the base component and are associated with additional capabilities. For example, an FPGA and a RISC core are both processing elements, but an FPGA has additional features, such as the capability to be configured to a specific task. Therefore, in the resource model hierarchy, the FPGA

[2] *Design space*, as used in this and subsequent sections, refers to the space of application-to-SoC mappings from an overall system design point of view, and not just the set of options at the hardware synthesis level.

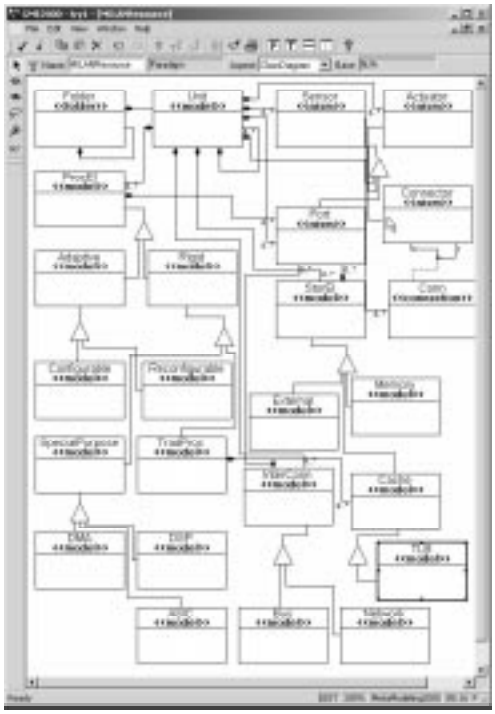

Figure 2: Screenshot of Resource Modeling Paradigm on GME

forms a separate category - configurable processing elements - inherited from the more generic "processing element" category. While frequency and voltage are common parameters of all processing elements, reconfiguration cost, hardware area usage, etc. are some parameters applicable specifically to configurable processing elements. Figure 2 is a screenshot of the resource metamodel created using GME [11].

Another important characteristic being captured by the resource model is the architecture parameters, which can be varied to optimize system performance. For example, voltage is a constant parameter for many components, whereas for devices such as the Intel StrongARM processor, it can be varied by application programs. Variable parameters such as these characterize a design space. The choice of parameters depends on several factors such as (a) desired detail of simulation, (b) capability of the resource, (c) parameters explicitly manipulated by the application, (d) availability of simulators with desired capabilities, and (e) range of design choices to explore.

4.2 Application Model

Application models are used to represent the algorithm to be implemented by the resulting system. Many different types of application representations may be used to represent embedded systems. We have chosen to initially focus on an enhanced hierarchical signal flow notation, that models a system as distinct components with well-defined interfaces. The "ports" that form these interfaces allow data to be exchanged between components. The signal flow defines the (partial) order of processing for an application. Each component in the signal flow graph receives data from other components, performs some transformation on the data, and then outputs new data to other system components. Figure 3 is a screenshot of the Automatic Target Recognition (ATR) application using a signal flow graph created in GME.

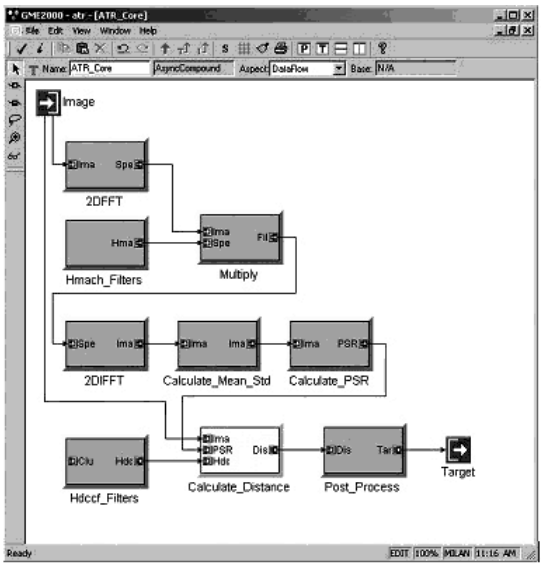

Figure 3: Screenshot of ATR Application Model

The following features are included in the application model:

- Both asynchronous and synchronous dataflow (ASDF, SDF) semantics are supported.
- Signals are strongly typed. A separate metamodel (developed for data typing) is used in conjunction with the application model, to specify and associate data types with ports. The environment enforces the type consistency of connections.
- Component level functionality implemented in hardware (i.e. configurable logic) can also be represented, thereby supporting SystemC and VHDL simulations.
- The modeling paradigm allows the specification of explicit implementation alternatives at any level of the hierarchy. For example, a filter may be implemented in the time or the spectral domain, it may be implemented on a DSP chip in assembly language, a RISC processor in C, an FPGA or an ASIC, etc. These alternatives, each with different performance characteristics and resource constraints, can be captured in the models. Alternatives allow the environment to support the modeling of the design-space of the application, as opposed to a single-point solution.
- The environment supports multi-granular simulations by allowing the user to specify implementation scripts at any level in the hierarchy. Implementation scripts can be in C, Java, MATLAB, SystemC, or VHDL. Specifying these is mandatory at the leaf level of the signal flow graph; this information is utilized during system synthesis. However, the user may choose to provide a C implementation of a high-level component directly. This provides fine control over simulation granularity.

4.3 Constraint Model

Constraints in MILAN are broadly of two types, semantic and design constraints. *Semantic constraints* are described

as part of the MILAN metamodels. These constraints define composability rules that are enforced by the domain-specific model building environment used by the designer to instantiate a particular system architecture. For example, a semantic constraint in a traditional uniprocessor system metamodel can specify that individual components can only be connected to the bus, and not directly to each other.

Design constraints, on the other hand, are specified by the user as part of the model building process, and formally specify the requirements (latency, throughput, power dissipation, etc.) of the final system. They also restrict mapping of tasks onto hardware components. For example, an FFT algorithm specified as a C program, should be mapped only to an Instruction Set Architecture (ISA)-based component such as a RISC core.

4.4 Performance and Communication Models

Performance and communication models are different from the models described above in two respects. Firstly, the primary purpose of application, resource, and constraint models is to enable the user to instantiate a system and provide the information required to drive simulation, synthesis, etc. These models define the designer's visualization of the system. Performance and communication models, on the other hand, determine the structure of MILAN itself since they form the basis for high-level performance estimation and simulator integration respectively. Secondly, application, resource, and constraint meta-modeling can be expressed in the GME tool as UML-like class diagrams that are used to synthesize the domain-specific modeling environment. Performance and communication models are not defined in MILAN through such meta-modeling.

A performance model is essentially a set of functions that relate the functionality and parameter values of a particular hardware component to the desired performance metrics of the task that is mapped onto that resource. This model will be used in a high-level estimation tool for rapid, coarse-grained performance evaluation of a given mapping. MILAN will leverage prior USC work in high-level modeling of both traditional and advanced architectures [1, 14]. Performance can also be modeled at a very low architecture level, which greatly improves the accuracy of statistics. However, accurate performance estimation typically results in low-level (cycle-accurate) simulation, which is too time intensive for evaluating a large set of possible mappings.

The motivation behind a communication model is the need for a common data-exchange layer/methodology between simulators. Most widely-used simulators are designed to be used as stand-alone tools, and making such simulators interact with each other is a non-trivial task. The communication model will form the basis for a canonical representation for information exchanged between simulators. Using this representation, a translation mechanism will convert one simulator-specific I/O format into another.

5. DESIGN SPACE EXPLORATION

Emerging SoC architectures provide a large number of hardware parameters - *knobs* - which can be manipulated by a designer to optimize application performance. In addition, diverse memory and I/O organizations, and alternative implementations of application tasks contribute to the exponentially large design space that needs to be explored to arrive at an optimal solution.

The traditional, combinatorial view of mapping defines a design space as the (large) set of all possible hardware-software partitions that are implied by the task implementations specified in the application model. For instance, if the resource model includes a RISC processor and an FPGA, and one of the application tasks is specified in both VHDL and C code, the design space doubles. MILAN will use a tool developed at VU [4], that uses Ordered Binary Decision Diagrams for exploration and pruning of this design space.

The characteristics of architectures described above, necessitates a new definition of design space. In this new scenario, DSE also includes exploring tradeoffs in algorithm design that are enabled by exploiting hardware knobs. Structured languages like C are not expressive enough to capture such information. Part of the MILAN effort will focus on designing a new application representation to capture the effect of architecture knobs on system performance. The design space exploration and pruning tool will use this representation to explore the multitude of design possibilities unique to such SoC architectures.

6. SIMULATOR INTEGRATION

Integrating simulators poses a multitude of challenges. Lack of standard interface among the simulators hinders integration of component specific simulators which is desirable for simulation of SoC type of architectures. Such integration involves the issues of uniform interpretation of simulation results, use of results to modify the high-level estimation tool or drive other simulations, and in the long run, development of tools and techniques for semi-automatic integration of third-party simulators into MILAN. MILAN addresses these issues through a model-based approach.

In MILAN, *model interpreters* are employed to address the issue of interpretation of results of dissimilar simulators. Model interpreters specific to each simulator provide a bridge between the MILAN models and the simulator. For example, the SimpleScalar model interpreter will extract the required architecture parameters from the resource model of the uniprocessor component and configure the simulator. The C files specified as part of the application model, also have to be compiled with the SimpleScalar compiler. After running SimpleScalar, the simulation results can be used for verifying satisfiability of the performance constraints. Parametric refinement of the high-level estimator for a system-wide performance evaluation can also be done using the results of this low-level simulation. Note that the interpretation of the results will be a human-in-the-loop process.

Providing the functionality of a single system-wide SoC simulator, by suitably integrating both the execution and the results of component-specific simulation, involves vertical and horizontal simulation.

Vertical simulation addresses the issue of providing multi-granular simulation, and refers to interpretation of low-level, component-specific simulation results in a global context [13].

The assumption here is that all the simulations are run independent of each other, i.e. there is no data exchange or any other communication between simulators at any granularity level. Component-specific low-level simulators might not be capable of representing system-wide effects that affect overall system performance (like communication delays between components). Vertical simulation is partially enabled by model interpreters that use the results of low-level simulations to compute parameters required by tools at higher levels of granularity. For instance, while SimpleScalar provides detailed simulation results (e.g. number of instructions executed, cache access, TLB access) a high level estimator might need just the energy consumed (as a function of number of instructions executed), and total execution time.

Horizontal simulation refers to the concurrent execution of component-specific simulators for system-wide performance evaluation. Unfortunately, none of the existing widely-used simulators lend themselves to easy modification that will enable them to trap selected events and efficiently interact with similar simulators in real time. Also, there is no common input/output format used by different simulators to represent information such as architecture parameters, application description, desired level of simulation, output statistics, etc. A possible approach towards solving this problem is defining a standard API for simulators to synchronize on events, and either modifying existing simulators or requiring new simulators to confirm to the standard.

7. CONCLUDING REMARKS

MILAN leverages previous ISIS [4, 11] and USC research [1, 14] in adaptive and data-intensive computing systems. The performance modeling paradigm is based on an extension of the hybrid system architecture models developed at USC [5]. The structural modeling paradigms are based on the extensive experience of the VU team in modeling a wide variety of embedded systems using MIC [19]. Currently, the framework supports SDF and ASDF application models, other models such as Finite State Machine and Discrete Event will be included in the future. A preliminary performance model for representing uniprocessor and memory hierarchy has been developed. A high-level model for power dissipation in such a system has also been defined. The current version of resource model captures a majority of structural details of DSPs, FPGAs, memories, interconnects, etc. A power/performance estimation tool (with limited capabilities) is also available. SimpleScalar, MATLAB, and SystemC have been integrated into MILAN. Other simulators (DSP, VHDL, etc.), and also simulator synthesis frameworks [2] will be integrated in due course. Defining interfaces between these simulators to enable horizontal and vertical simulations will be a challenging task. Therefore, we are also developing generic guidelines and an *extensibility toolkit* for integrating third-party simulators. We expect a preliminary version of the framework to be available for public use by the end of the year.

8. ADDITIONAL AUTHORS

V. Mathur, S. Mohanty, C. S. Raghavendra, M. Singh, A. Agrawal, J. Davis, B. Eames, S. Neema, G. Nordstrom, email: {vaibhav, smohanty, raghu, mitalisi}@usc.edu, {aditya.agrawal, james.r.davis, brandon.eames, sandeep.k.neema, greg.nordstrom}@vanderbilt.edu.

9. REFERENCES

[1] Algorithms for Data Intensive Applications on Intelligent and Smart Memories (ADVISOR), Univ. of Southern California. http://advisor.usc.edu.

[2] A. Bakshi and V. K. Prasanna, "Abstract Resource Representations for Custom Design of System-on-Chip Architectures," submitted to IFIP VLSI-SOC 2001, Montpellier, France, December 2001.

[3] F. Balarin et al., "Hardware-Software Co-Design of Embedded Systems: The POLIS Approach," Kluwer Academic Publisher, Massachusetts, 1997.

[4] T. Bapty et al., "Model-Integrated Tools for the Design of Dynamically Reconfigurable Systems," ISIS Technical Report/Vanderbilt University, 2000.

[5] K. Bondalapati and V. K. Prasanna, "Mapping Loops onto Reconfigurable Architectures," International Workshop on Field Programmable Logic and Applications, Tallinn, Estonia, August 1998.

[6] G. Booch et al., "The Unified Modeling Language User Guide," Addison-Wesley Pub Co., 1999.

[7] Chameleon Systems Reconfigurable Communications Processor, http://www.chameleonsystems.com/.

[8] H. Chang et al., "Surviving the SOC revolution - A guide to Platform-Based Design," Kluwer Academic Publisher, Boston, November 1999.

[9] M. Chiodo et al., "A Formal Specification Model for Hardware/Software Codesign," Proc. of the International Workshop on Hardware-Software Codesign, October 1993.

[10] P. Chou et al., "IPCHINOOK: An Integrated IP-based Design Framework for Distributed Embedded Systems," Design Automation Conference, June 1999.

[11] Generic Modeling Environment, http://www.isis.vanderbilt.edu/projects/gme/default.html.

[12] T. D. Givargis et al., "Fast Cache and Bus Estimation for Parameterized System-on-a-Chip Design," Design, Automation and Test in Europe, March 2000.

[13] V. Mathur and V. K. Prasanna, "A Hierarchical Simulation Framework for Application Development on System-on-Chip Architectures," submitted to the 14th IEEE Intl. ASIC/SOC Conference, Washington DC, September 2001.

[14] Models, Algorithms and Architectures for Reconfigurable Computing (MAARC), Univ. of Southern California, http://maarc.usc.edu.

[15] T. Mudge, "Power: A First Class Design Constraint for Future Architectures," 7th Intl. Conference on High Performance Computing, Bangalore, India, December 2000.

[16] The Ptolemy Project, http://ptolemy.eecs.berkeley.edu.

[17] SimpleScalar Tool Set, http://www.simplescalar.org/.

[18] Simulink 4, http://www.mathworks.com/.

[19] J. Sztipanovits and G. Karsai, "Model-Integrated Computing," IEEE Computer, April 1997.

[20] Triscend Configurable System-on-Chip Family, http://www.triscend.com

[21] D. C. Wyland, "The Universal Micro System: Hardware Performance with Software Convenience," Cradle Technologies White Paper, http://www.cradle.com/literature/tech_papers.html.