Multigranular Simulation of Heterogeneous Embedded Systems

Aditya Agrawal
Institute for Software Integrated Systems
Vanderbilt University
Nashville, TN - 37235
1 615 343 7567
aditya.agrawal@vanderbilt.edu

Akos Ledeczi
Institute for Software Integrated Systems
Vanderbilt University
Nashville, TN - 37235
1 615 343 7440
akos.ledeczi@vanderbilt.edu

ABSTRACT

Heterogeneous embedded systems, where configurable or application specific hardware devices (FPGAs and ASICs) are used alongside traditional processors, are becoming more and more widely used. To facilitate rapid design and development of such heterogeneous hardware/software systems, it is essential to expand the software design cycle to integrate hardware modeling and simulation. Co-simulation and exploration of the joint design space are key problems. To design, develop and verify such systems, different kinds of simulations at various levels of granularity are required. The hardware modeling and simulation framework of the Model-Based Integrated Simulation Framework (MILAN) integrates these requirements into a single powerful design, development and simulation environment.

Categories and Subject Descriptors

E.3 [**HW/SW co-design**]: specification, modeling, co-simulation and performance analysis, system level partitioning and scheduling.

General Terms

Performance, Design, Standardization, Languages and Verification.

Keywords

Modeling, Orthogonalization, Design space and Simulation.

1. INTRODUCTION

Configurable FPGAs and fast ASICs are pushing embedded systems to implement more and more functionality directly in hardware. However, to facilitate the rapid design and development of such heterogeneous hardware/software systems, it is essential to expand the software design cycle to integrate hardware modeling and simulation. The Model-based Simulation Integration Framework (MILAN) [1][2] provides a unified environment for the design and simulation of these heterogeneous systems.

Such a unified environment poses unique requirements that need to be fulfilled. One important requirement of the heterogeneous design paradigm is the orthogonalization of concerns, that is to separate various aspects of design in order to effectively explore alternative solutions [9]. For example, system requirement specifications and implementation or computation and communication are good candidate concerns that should be separated.

In large and complex systems there is a need for modular design to mitigate complexity. Systems are typically designed in terms of components and component interactions. A component usually embodies some kind of computation and it has a standardized interface for communication. This helps to separate computation from communication and the developer can design and implement one without being concerned with the other.

Separation of system requirements and implementation is desirable because the former captures the intention of the system designer and provide a high level view, while the latter is specific and is done at a much finer level of granularity. By capturing the intention separate of the implementation, the high level abstraction is preserved, allowing the user to specify alternate implementations for the same intent. These alternatives may be in the form of different algorithms to solve the same problem, a choice between hardware and software implementation, or a selection of programming language. Furthermore, implementation is a refinement of the intent and needs to be captured at different levels of granularity. Initially a coarse grain implementation is used for prototyping. This

can be transformed in stages to a detailed low-level implementation later.

By capturing alternative implementations at different levels of granularity we gain the flexibility of choosing the implementation according to the exact needs of the system. The development cycle starts from a coarse grain implementation. This is tested for functional correctness and is then refined to different alternative implementations. The feasibility of these alternatives is explored by profiling them. This is followed by system simulation of a few feasible system wide implementations to validate the system with respect to the requirements. Simulation becomes more important as testing of these applications on actual hardware is expensive and time consuming, especially for applications implemented in hardware such as FPGAs or ASICs.

These design and development philosophies exist and are used in real world embedded systems and can be used for the expanded role of computer-based systems. In the absence of an integrated design and development environment, a variety of tools are used to achieve all the above needs. These tools more often than not are incapable of exchanging design, implementation and data between each other forcing developers to duplicate information manually between tools. This is a time consuming, inefficient and a error-prone. In order to speed up the design cycle, there is a need for an integrated design and development framework that facilitates all these requirements.

The Model-based Simulation Integration Framework (MILAN) [1] is a suite of tools developed to integrate the following design and development needs:

- Single design representation to use in different simulations and software synthesis,
- Separation of concerns,
- Capture different levels of hierarchy for refinement.
- Synthesize code to drive various simulation methodologies: isolated simulation, multi-granular simulation, full system simulation,
- Speed up the design and development cycle for rapid application development.

The focus of this paper is on the modeling paradigm for applications implemented in hardware and the associated tools integrated into MILAN. These tools consist of an integrated environment to specify modular system design with alternative implementations. At the lowest level, designers need to provide SystemC or VHDL implementations. The tools are capable to drive various kinds of simulations from these specifications to provide for verification of functionality and performance. The kinds of simulation supported are isolated simulation of components, multi-granular simulation of the system, complete system simulation and simulation of hardware in heterogeneous hardware/software systems.

We begin in Section 2 by discussing Model Integrated Computing (MIC) and an overview of MILAN. In Section 3 we discuss the modeling methodology of MILAN with focus on hardware modeling, followed by the interpretation of the models to drive various simulations in Section 4. We conclude in Section 5.

2. MILAN OVERVIEW

The software infrastructure of MILAN is based on Model Integrated Computing (MIC). MIC employs domain-specific modeling methodology to represent the system being designed. The system models are then used to automatically synthesize the applications and/or to generate inputs to analysis and/or simulation tools. This approach speeds up the design cycle, facilitates the evolution of the application, and helps system maintenance, dramatically reducing costs during the entire lifecycle of the system. MIC is implemented by the Generic Modeling Environment (GME), a metaprogrammable toolkit for creating domain-specific modeling environments [6].

MILAN is a typical MIC application. Its architecture is depicted in Figure 1. The domain-specific modeling paradigm developed specifically for MILAN enables the specification of the system in the form of multiple-aspect, hierarchical, primarily graphical models in GME. The three main categories of models specify the desired application functionality, available hardware resources and non-functional requirements in the form of explicit constraints. The application models capture the dataflow of the system. Both asynchronous and synchronous dataflow is supported, as well as their composition. By allowing the specification of explicit design and implementation alternatives as part of these models, MILAN captures the design space [10] of the application, not just a point solution. Size, weight, energy, performance and timing (SWEPT) requirements are also part of the models in the form of formal constraint.

Only a subset of the potentially exponentially large design-space satisfies all these constraints. A symbolic constraint satisfaction methodology is applied to explore and prune the design-space. Once a single design has been selected, generators translate the models into the input of the selected simulators. Simulators already integrated into MILAN include Matlab [8], SystemC [12], HiperE, a high-level performance estimator developed in parallel with

MILAN [13], SimpleScalar [4], Armulator [3]. Some simulation results need to be incorporated back in the models in the form of performance attributes of components, for example, to make them available for other simulators. For some simulators this will necessarily be a human-in-the-loop process, while for others the procedure can be automated.

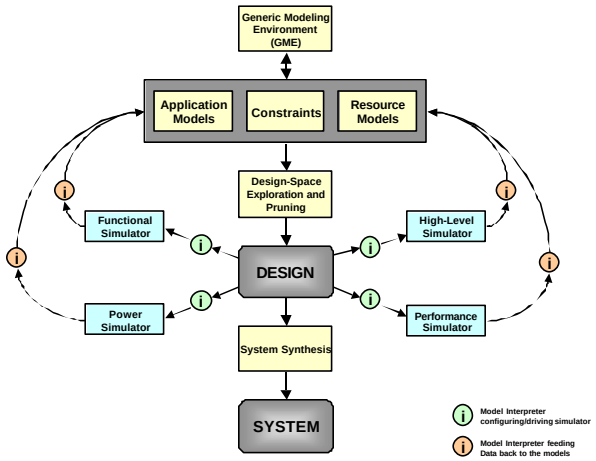


Figure 1. MILAN Architecture

The final component in the MILAN architecture is Software Synthesis. Notice that this step is similar to driving simulators. Instead of targeting the execution model of a simulation engine, the synthesis process needs to generate code for a given runtime system. Just like there is a need to support multiple simulators, MILAN supports multiple target runtime systems.

The goal of the paper is to describe how a careful composition of a variety of modeling concepts can result in a highly domain-specific modeling methodology that supports the unique needs of the complex application domain of simulation of computer-based systems. We shall focus on modeling of applications implemented in hardware and how the models are used to drive the various types of simulation.

3. MODELING METHODOLOGY

Modeling is an abstraction of the system and captures specific details required to best represent, understand, implement and modify the system. Traditional hardware design approaches and the need for integrating with the expanding domain of computer-based systems motivated us to incorporate various modeling methodologies to mitigate complexity, separate concerns and to provide for effective management and maintenance of these systems.

The wide variety of domain specific modeling concepts that has been incorporated are as follows.

1. Modeling of hardware applications using domain specific concepts and to separate the concerns.
2. Strong data typing of communication ports for accurate simulation of data exchange and to catch modeling errors at design time.
3. Parameterization of components to develop generic modules for reuse, as well as to design a set of solutions instead of a single solution.
4. Data abstraction and information hiding to better manage complexity using multiple aspects of the same module.
5. Explicit designs of alternative implementations to capture design choices in order to better explore different solutions.
6. A paradigm to compose hardware and software components together to facilitate the design of heterogeneous systems.

3.1 Hardware Modeling

The goals while developing the hardware application modeling paradigm were separation of concerns, flexibility to capture the implementation in different languages and at different levels of granularity. Another goal was to mitigate complexity to help design more manageable systems.

The model of computation in hardware is unbuffered hierarchical dataflow. Hierarchical dataflow implies that a dataflow node can contain a dataflow subgraph. Unbuffered dataflow means that the receiver and sender need to be synchronized. This model of computation can alternatively be looked at as the structural description.

The hardware modeling paradigm consists of a set of modules implementing behavior and directed links connecting modules specifying the dataflow graph of the system. The modules are hierarchical, that is they can contain other modules and module associations forming a dataflow subgraph. Figure 2 shows the basic meta model of MILAN's hardware-modeling paradigm.

hwModule is the basic building block. It is a hierarchical module as it can contain subgraphs. Ports define the input and output interface of the module, while *hwSignalConn* is an association between ports representing a data path. These ports can also be connected to and from a *hwBus*. Modules also contain *hwDataStore* which represent memory elements.

A module that doesn't contain a subgraph has processes associated with it. Processes specify the behavior of a module. This is captured as functions implemented in a hardware description language such as VHDL or SystemC. Notice that, *hwModule* contains *hwFunctionBase*, an

abstract base class. This class has been specialized for SystemC and VHDL. It can also be specialized to supported other languages later. The functions can be event driven or sequential. Events are specified using the *hwTrigger* connection between processes and ports.

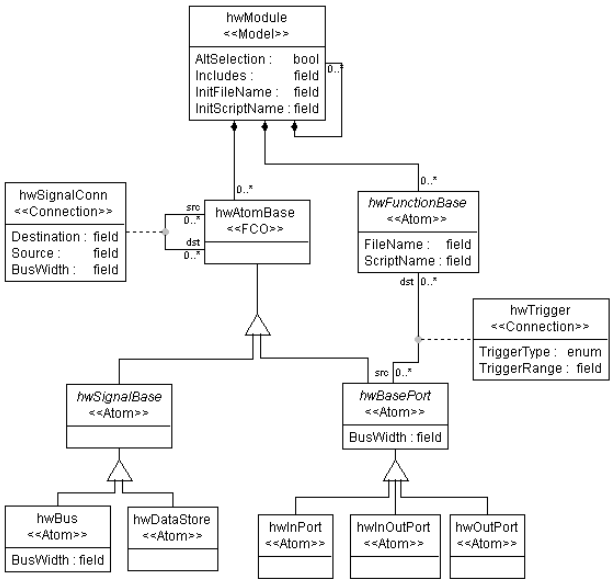


Figure 2. Meta model of the hardware application paradigm.

Hierarchy in the modeling paradigm serves two purposes. First, it helps separate intention from implementation. Using hierarchy the system is designed according to the intention, that is the high-level dataflow of the system. Then the design in refined by designing the modules in details until it is low-level enough to provide an implementation. Second, it helps to mitigate complexity. The dataflow graph of large systems can be very complex; hierarchy hides data at different levels to make the systems more manageable. The functions can be specified at any level of granularity and can be in either VHDL or SystemC. This provides the user with the flexibility to choose between different HDLs. Moreover implementations in different languages can coexist providing the user with design choices.

3.2 Data Type

Data type models in MILAN are used for several purposes. First of all, to accurately simulate communication performance, the amount of data exchanged needs to be captured. Furthermore, as data type models are attached to hardware modules, or more precisely to their input and output ports, they define the interface of those components. When the components are attached using signal connections, their interfaces are checked to ensure that only compatible objects are connected. Finally, the data type models can also be used to generate the corresponding definitions in the target hardware description language ensuring consistency.

The MILAN data type modeling paradigm allows the specification of both simple and composite types. Simple types, such as floats and integers, specify their representation size, i.e. the number of bits used. Composite types can contain simple types and other composite types. Attributes of the fields specify extra information such as array size or signed/unsigned type. Data types supported by the C programming language can be modeled in MILAN. Preexisting data types, specified in a DSP library for example, can also be modeled. Their name and size in bytes are the only information MILAN requires.

The hardware application and the data type modeling paradigms are composed together according to precise rules (not shown). OCL constraints ensure that every port has exactly one type specification and that dataflow connections are only allowed between ports having compatible data types.

3.3 Parameters

In order to support parametric hardware modules, such as an FFT block with configurable data points, MILAN allows for the specification of such parameters.

Components contain *ParameterPorts* capturing their parameter interface. A *Parameter* can be connected to a ParameterPort supplying a value to it. Each port has a default value that is used if no Parameter is attached to it. Both the ParameterPort and the Parameter are data typed, using the same modeling technique as for ports. Typing information is used to verify that the supplied parameter is compatible with the parameter interface of the component.

Parametric modeling plays an important role in representing design spaces. A parametric component encapsulates multiple implementations that can be selected by supplying an appropriate value for the parameter. For example, an N-point FFT model encapsulates a number of FFT implementations spanning the valid range of N. Thus, a number of options can be represented in the models instead of an implementation. Furthermore parameterization helps to design and develop generic components that can be reused.

3.4 Multiple Aspects

The MILAN application modeling paradigm is quite complex. However, the hardware description, data type specification and parameter modeling are largely orthogonal concepts. Therefore, they can be separated into different aspects to allow the user to better manage and understand the system. In the *Hardware aspect*, only module, ports, buses, data stores and the true implementation scripts are shown. In *Type aspect*, Ports,

Parameters, ParameterPorts and data type references are displayed. Finally, Components, Parameters, ParameterPorts and their corresponding connections are visible in the *Parameter aspect*. Multiple-aspect modeling is a natural way to implement separation of concerns.

3.5 Software application modeling

The software application modeling paradigm is based on a dataflow representation. A dataflow graph consists of a set of compute nodes and directed links connecting them representing the flow of data. A flat graph representation does not scale well for human consumption, so we extended the basic methodology with hierarchy.

There is extensive literature on various dataflow representations. At the two ends of the spectrum are synchronous [7] and asynchronous dataflow. Both these models of computation are supported by MILAN and have been discussed in greater details in [6].

3.6 Alternatives

Till now we have discussed the modeling environment and a lot of its feature, however we haven't discussed how alternative designs are represented in the models. Parameterized components are one way of representing design alternatives. Being able to use multiple languages of implementation provides for alternative implementation.

MILAN also allows the user to have an explicit choice between synchronous, asynchronous data flow and hardware implementation. This is achieved by using alternatives. Alternatives are models that can contain synchronous, asynchronous software dataflow and hardware modules and the containment implies an or condition. That is one and only one of the given implementations will be used. Furthermore choice between different algorithms to solve the same problem can also be captured using explicit alternatives.

3.7 Composition of hardware and software

MILAN supports the composition of hardware and software models.

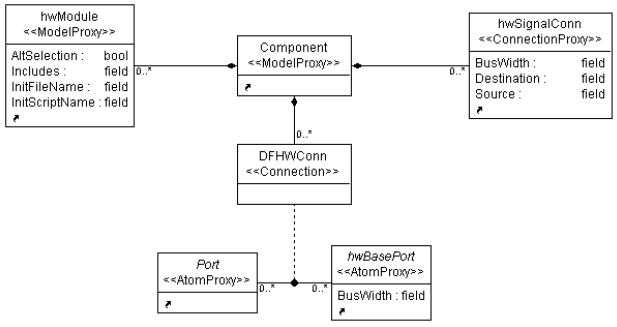


Figure 3. Composition of hardware and software

The metamodel in Figure 3 specifies that a dataflow component can contain hardware modules and signal connections. Furthermore, hardware and dataflow can be associated using the connection *DFHWConn*. This represents a data path between software and hardware components. Thus, a hardware implementation of a sub-SystemCan reside in any dataflow component.

4. SUPPORT FOR SIMULATION

After creating a design environment that allows us to model applications and is able to capture hardware and software designs, parameterized components and design alternatives, there is a need to drive various simulations from these models.

A typical development cycle starts from a coarse grain implementation, which is tested for functional correctness. With respect to hardware applications, SystemC is a good language to provide a coarse grain implementation. Simulation of the SystemC implementation can be used to verify functional correctness. The user will normally refine one block at a time and so he/she may need to simulate the refined module with coarse grain implementations of the other. MILAN's support of simulation at different levels of hierarchy is called multi-granular simulation. After refining the design to various alternative implementations, the feasibility of the implementations is explored by profiling the modules. Profiling of modules requires simulation of each component in isolation to come up with performance numbers such as throughput, latency, power consumption and memory requirements. This kind of simulation in MILAN is referred to as isolated simulation. After profiling the modules a few feasible designs are chosen for system wide simulation to validate the system requirements. This is referred to as full system simulation.

In heterogeneous systems, that is systems having hardware and software components interacting with each other, verification of design becomes a more challenging task. Simulation of the hardware having communication with software components can be achieved by providing a communication bridge between the hardware and software components. This helps to simulate hardware with the software implementation providing more accurate results.

To drive the various simulations mentioned above and to automate this process the models captured in the design environment need to be interpreted to generate code for simulation.

4.1 Model Interpretation

The interpreter of the hardware-modeling paradigm in MILAN generates code for simulation. Currently SystemC code generation is supported, that is if the behavior of the system is described in SystemC, the interpreter can

generate the code for isolated, multi-granular and full simulation. The generated code can be compiled and run to get simulation results. Furthermore simulation of hardware in a heterogeneous system is also supported.

The interpreter traverses through the graph and gathers the required information, like ports, signals, data elements, event driven functions and their dependencies. These are then used to generate SystemC glue code.

4.2 Multi-granular Simulation

In the hierarchical graph representing the system, typically the lowest level modules contain the behavioral information. Using multi-granular simulation the user can choose to provide behavioral information for any module at any level of hierarchy. Thus the user can simulate a system with a mixture of coarse grain and fine grain implementations. The high level behavioral information is captured in the *Coarse Grain Aspect* of the module.

The interpreter generates the code for the system and whenever it finds a module marked for using the coarse grain implementation it uses that and doesn't traverse deeper in that module.

4.3 Isolated Simulation

In order to simulate a module in isolation the module needs to be driven by sourcing functions and the output of the module needs to be sent to sinking functions. In MILAN we allow the user to capture the exact sourcing and sinking function associated with each communication port.

Hence to synthesize for an isolated simulation of a module, the true implementation of the module in question is used along with the sourcing and sinking functions from adjacent modules. The interpreter generates code of the module in question and creates sourcing and sinking modules for it. The sourcing and sinking modules contain simulation scripts specified by the user in the *Substitute aspect* of the adjacent modules. Isolated simulation can be performed not only on a single module but on a subgraph also, that is a connected subgraph can be chosen for simulation and the modules adjacent to this graph will be used to supply and consume data.

4.4 Full Simulation

For a complete simulation of the entire system a single design needs to be chosen. The user can choose between alternative implementations by marking one of various alternative implementations to use. Alternatively, the design-space exploration tool can identify the point designs that satisfy the all constraints [10] and mark the selected alternatives automatically. The interpreter then traverses through the models and picks up the chosen alternative implementations to form a single design. The true implementations of the design are then used to generate SystemC code for a full simulation.

4.5 Simulation in Heterogeneous Systems

To simulate hardware in a heterogeneous system, it is necessary to facilitate communication between hardware and software components. In a real-world system, hardware software interactions are facilitated using device drivers. However, we do not require device drivers to simulate the system. The communication is achieved by using entities called proxies. At a hardware software interface proxies are generated on both sides of the interface. For example, a hardware proxy will read data from the hardware module at the interface and pipe it to its software counterpart using TCP. Similarly it will read data from the pipe and provide it to the hardware module. The software proxy does the same at the other end.

The interpreter breaks the heterogeneous graph into hardware and software graphs. It then generates the proxies and connects the respective graphs at the interface. Finally, the two graphs are sent through their respective interpreters. The hardware and software code can finally be compiled independently and then run together to simulate the hardware.

5. EXAMPLE APPLICATION

Image processing systems and specifically, missile Automatic Target Recognition (ATR) systems face many challenges due to extremely large computational requirements and physical, power, and environmental constraints [11]. Thus, it is a good example to demonstrate some of the capabilities of MILAN.

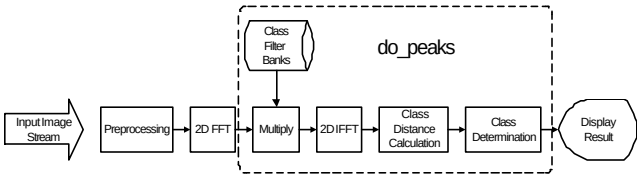


Figure 4. ATR application block diagram

The design and simulation of a system in the MILAN framework starts with application modeling. Given the size of the ATR application and the large number of design choices, both hierarchy and alternatives are used extensively. Figure 5 shows a model of the do-peaks block of the ATR in the MILAN framework. This model captures one of the core computations in the ATR application. When designing the application it was determined that the functionality of the peak to surface ratio (PSR) can be realized in hardware or software. Instead of making the selection upfront, the alternative realizations are captured in the models and the selection postponed till a later phase of design. Notice that in Figure 5 PSR is modeled as an

AsyncAlternative (colored differently) Figure 6 shows the alternative realizations of PSR.

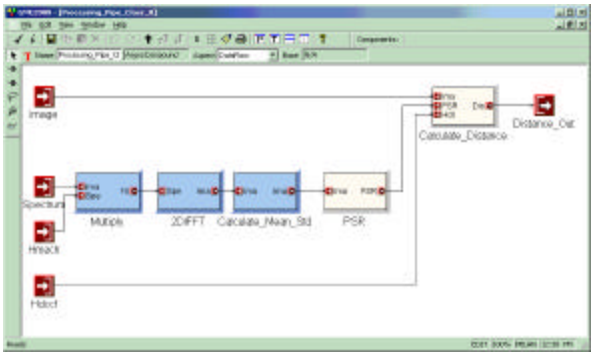


Figure 5. do_peaks model of the ATR application

The software implementation of PSR is an asynchronous dataflow node and uses regular C code. The hardware implementation, on the other hand, is a high-level node representing a dataflow subgraph (Figure 7b). In the early phases of the design it is not necessarily clear which one is a better implementation. The suitability of one or the other depends upon the actual resources that are available, the runtime execution environment that is employed, and other factors. The various simulation tools assists the designer in making these selection decisions based on the requirements of the system.

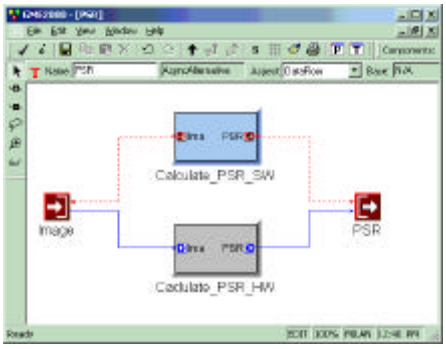
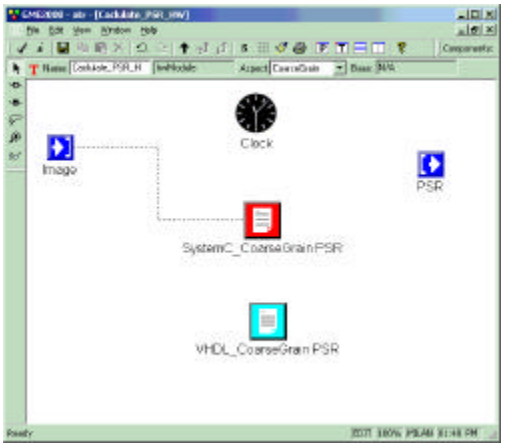


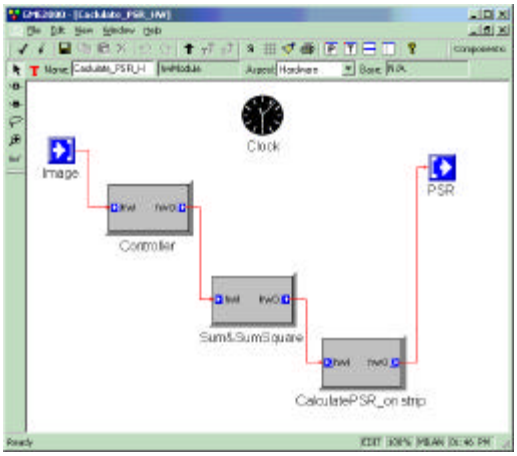
Figure 6. Alternatives in ATR application

Figure 7 shows two aspects of the hardware model of the PSR. In addition to the input and output ports, the *Hardware* aspect contains a data flow sub-graph of the model showing the refined realization of the module. The coarse grain aspect shows the high level implementation of PSR. It contains a SystemC-script, and a VHDL-script. The SystemC-script is a placeholder for the high level SystemC code implementing the PSR, while the VHDL-script is a placeholder for VHDL code. An appropriate script is selected based on the target simulator.

A typical design cycle will begin with the designer wanting to verify the functional correctness of the alternative hardware application with a coarse grain implementation. So the designer will choose the hardware version of the PSR and mark it to specify the use of a course grain implementation. Then the user will simulate the system with the coarse grain implementation. The next step will require the user to profile the hardware alternative in order to allow him to choose between the hardware and software at a later stage. This will require an isolated simulation of the detailed hardware implementation of the PSR. The designer will select the hardware version in the alternative model and run isolated simulation on it. The interpreter will then use the hierarchical implementation of the PSR and use sourcing and sinking functions from Calculate_Mean_Std and Calculate_Distance to generate SystemC code for isolated simulation. After deciding on the implementation to use the designer will want to run a full simulation of the system with the right design choice. In this example, let's say that the hardware implementation is chosen. Then the user selects that alternative and runs the full system simulation. In this case the interpreter uses the hierarchical implementation of the PRS implemented in hardware as well as the complete implementation of the rest of the system. The full simulation allows the user to verify the design with respect to system requirements.



(a) Coarse grain aspect



(b) Hardware aspect

Figure 7. Two aspects of the hardware model of PSR

Subsequent to application modeling, the next step in the ATR design is resource modeling. In this step the target resources are modeled as per the resource-modeling paradigm.

6. CONCLUSION

Needs of embedded applications have expanded with the advent of FPGA's and ASIC's. The increasing complexity and heterogeneity of such systems drives the need to have an integrated framework to support design and development to speed up the design cycle and to explore various alternative solutions.

MILAN is a framework that provides an integrated environment to design embedded system applications using domain specific concepts. It allows for the design of alternative solutions and abstracts the implementation from design. Separation of concerns and modular design are the pillars of MILAN.

Modeling of hardware using domain specific concepts in a heterogeneous system allows for better representation of the design and helps developers to design systems in an intuitive manner. Support for various simulation needs by a composite environment helps to speed up the design cycle and allows for better exploration of alternative solutions.

The framework, specifically the hardware paradigm has been applied to various small and medium sized projects with a great deal of success in terms of increasing efficiency and reducing the design time.

7. ACKNOWLEDGMENTS

The research described in this paper is sponsored by the DARPA IPTO Power Aware Computing and Communications program. The MILAN project is a joint effort of Prof. Viktor Prasanna's group at the University of Southern California and the Institute for Software Integrated Systems at Vanderbilt University.

8. REFERENCES

[1] Agrawal, A. et al. MILAN: A Model Based Integrated Simulation Framework for Design of Embedded Systems, Workshop on Languages, Compilers, and Tools for Embedded Systems (LCTES 2001), Snowbird, Utah, June 2001.

[2] MILAN. www.isis.vanderbilt.edu/projects/milan/index.htm

[3] ARMulator. http://www.arm.com/sitearchitek/support.ns4/html/sdt_debug

[4] Burger, D. and Austin, M., The SimpleScalar Tool Set, Version 2.0, Computer Architecture News, 25 (3), pp. 13-25, June, 1997.

[5] J.T. Buck, S. Ha, E.A. Lee and D.G. Messerschmitt, Ptolemy: A Framework for Simulation and Prototyping Heterogeneous Systems: Int. Journal of Computer Simulation, special issue on "Simulation Software Development", vol.4, pp. 155-182, April, 1994

[6] Ledeczi, A., et al. Composing Domain-Specific Design Environments, Computer, pp. 44-51, November, 2001.

[7] Lee, E. A. And Messerschmidt, D. G., Static scheduling of synchronous data flow programs for digital signal processing. Transactions on Computers, C36 (1):24 --35, January 1987.

[8] MATLAB. http://www.mathworks.com/products/matlab/

[9] Marco Sgroi, Luciano Lavagno, Alberto Sangiovanni-Vincentelli. Formal Models for Embedded System Design. IEEE Design & Test of Computers, 17(2):14-27, June 2000.

[10] Neema, S., System Level Synthesis of Adaptive Computing Systems, Ph. D. Dissertation, Vanderbilt University, Department of Electrical and Computer Engineering, May 2001.

[11] Nichols, K. And Neema, S., Dynamically Reconfigurable Embedded Image Processing System, Proceedings of the International Conference on Signal Processing Applications and Technology, Orlando, FL, November, 1999.

[12] SystemC. http://www.systemc.org/

[13] Mohanty, S. And Prasanna, Viktor K., Rapid System-Level Performance Evaluation and Optimization for Application Mapping onto SoC Architectures, 15th IEEE International ASIC/SOC Conference, Rochester, New York.