System Design and Re-Engineering Through Virtual Prototyping: A Temporal Model-Based Approach

M. H. Khan and V. K. Madisetti
VP Technologies, *A Georgia ATDC Company*,
P.O. Box 680190, Marietta, GA 30068-0004.
http://www.vptinc.com

Georgia Institute of Technology-ECE, Atlanta, GA 30332-0250

Abstract

During the design and re-engineering process, modeling the application's timing characteristics on the target architecture is necessary in order to evaluate the number of processors, communication fabric, and partitioning trade-offs required for an efficient design of a system. The over-all performance of most of COTS (Commercial-Off-The-Shelf) systems, ranging from supercomputers to multiprocessor DSP (Digital Signal Processing) systems, is uniquely affected by the behavior of the communication primitives supported by them. To evaluate the performance of such systems, it is essential that temporal models, commonly known as performance models, reflect the effects of the communication primitives on the system performance. In this paper, we propose a VHDL-based (VHSIC Hardware Description Language) temporal modeling environment where we model system's hardware as well as its accompanying software. We incorporated support for the Message Passing Interface (MPI) standard into the modeling domain, allowing an architecture independent abstraction of application and more accurate model for communication primitives. This tool provides an excellent platform for temporal evaluation of systems under design and re-engineering. Finally, when coupled with code-generation tools, we are able to generate control software that can directly run on the target platform, simplifying the task of code design for legacy system upgrades.

1 Introduction

In this paper, we propose the use of a simulation-based system design and re-engineering methodology, describe its implementation via tools, and its impact through case studies. In our modeling paradigm, we opted for an architecture-independent representation of the complex systems, while faithfully capturing the temporal related behavior of the desired implementation in an executable form. The proposed platform allows modeling of the underlying targeted HW (hardware) systems as well as the existing SW (software) or the SW under design. Modeling of both HW and SW allows to capture the timing characteristics of the system more accurately. Furthermore, by quickly changing the HW models and SW models we can find the performance of the application on various target architecture, and also the effect of the SW changes on systems performance.

The MPI (Message Passing Interface) standard provides the system's designer with a library of communication primitives which makes the underlying implementation of the message transaction transparent to the firmware engineers [10]. Modeling MPI allows us to represent the application in an architecture independent manner such that the models of HW and SW models can be changed independent of each other and analysis of trade-offs can be performed very easily. Furthermore, we can observe the effects of different MPI primitives on performance metrics of the system under design or re-engineering. We have chosen the RASSP (Rapid Prototyping for Application Specific Signal Processor) SAR (Synthetic Aperture Radar) benchmark as our case study to demonstrate the platform's usage for system design. In the second case study, we demonstrate how the temporal modeling tool can be useful for re-engineering of a typical avionics system.

2 Architectural Modeling Approaches

Evaluation of timing and performance (latency, throughput, and utilization) of digital computing systems has been a topic of research since the early days of computer science. Simulation-based or numerical techniques are preferred over analytical and statistical techniques typically due to their better control over the experimental conditions, ability to capture the transient scenarios and ease of use. Though simulation models can be time consuming to create, they can provide accurate performance of the system before the system HW and SW have been fabricated, thus reducing system design time [5].

In past few years, VHDL (VHSIC Hardware Design Language) has served as a powerful language for realization of many of the performance modeling techniques. Among them ATL (Advanced Technology Lab)[7] , PML (Performance Modeling Lab)[1], RASCAL (Rapid Architectural Synthesis of Advanced Computer Architecture)[6], MSCE[3]. The reasons behind VHDL's popularity as a platform are its inherent support for discrete event simulation, notion of explicit time, and ease of representation of actual underlying hardware.

3 Proposed Exploration Environment

Since a system's performance is dependent on software and hardware (which is not available at the time of design), it is essential to have early representations of both system components in the modeling environment. In earlier work at Georgia Tech, we had provided very realistic and accurate representations of interconnection components. Communication interconnects in RASCAL environment are modeled as virtual packet passing networks. These communication components were more detailed than those found in the PML library [1] and closer to actual protocol of the underlying interconnect since they capture the protocol in a similar fashion to Harel's state chart [8]. For representation of hardware we have adopted our earlier approach. This allowed us to reuse RASCAL's CPL (Conceptual Prototyping Library), a library of interconnect models and various packages (e.g. traffic generation package) as well as AIM (Automatic Interface Model Generator) and a hybrid modeling environment [5]. On the other hand, as part of DARPA's RASSP program, Lockheed Martin ATL proposed an efficient technique to model software [7] where the software was modeled as a set of generic instructions such as *compute, receive, send, loopback, and programdone*. The application composed by these generic instructions is read by a processor model. Due to the flexibility of this software representation, we have adopted an approach similar to that of ATL for modeling applications. We have extended the instruction set to account for the choice of processor, choice of algorithm implementation, and communication primitives (or MPI primitives).

An MPI representation of a program consists of a set of autonomous processes, executing their own codes in either MIMD (Multiple Instruction and Multiple Data) or SPMD (Single Program and Multiple Data) programming paradigms [10]. The basic communication mechanism within MPI is point-to-point communication between pairs of processes. Apart from point to point communication primitives MPI offers group communication primitives. These operations can be performed in different modes, blocking and non blocking. Each communication mode can be either synchronous or asynchronous. Due to the variety of communications primitives offered by MPI, almost any application can be adequately modeled with a set of computation commands and MPI communication commands. Therefore, once an application under design or a currently existing is abstracted in terms of MPI primitives, we can observe the execution of the application on various platforms to evaluate trade offs.

For most avionics systems, communication overhead appears to play a major role in determining systems performance. Communication overhead is dependent not only on the underlying network characteristics and topology, but also the communication primitives. Not only communication overhead but also peak buffer length and even overall execution time of an algorithm may change because of different communication modes. Execution time of a task on a system has two dominant components *computation time* and *communication time* or *communication overhead*. The communication overhead (t_{OH}) for any communication primitive is comprised of the following components.

$$\mathbf{t_{OH} = t_{Setup} + t_{Lat} + t_{Trans} - t_{Overlap}} \qquad (1)$$

Modeling message passing in the temporal domain, essentially, means capturing these delays in the model. One can represent $\mathbf{t_{Lat}}$ as the time a process/task waits to start communicating, and the queuing delay at various nodes as the message is transferred, $\mathbf{t_{Trans}}$ as the transmission time of the message, $\mathbf{t_{Setup}}$ as the time taken by the process for internal copying etc., and $\mathbf{t_{Overlap}}$ as the amount of time communication and computation can be overlapped. For blocking communication there is no overlap of computation and communications. The value of $\mathbf{t_{Setup}}$ is usually provided by vendors or measured from an existing system. The

network delay cannot be predetermined since they are dependent on the dynamic behavior of a system and often is the dominating contributor to the equation. Thus these delay parameters are to be extracted by simulation.

3.1 Model of Computation

The processor model comprises of a set of configurable generic VHDL processes: these are communicating processes, control processes, and computation processes. The communication process handles the network communications, while the computation processes handle the computation, and perform communication between nodes through the communicating process. Control processes control the scheduling of the computing processes. This modular approach allows us to quickly map the communication, computation, and control related design decisions (for instance, the control process can reflect the scheduling scheme, whereas the communication process can model the communication modes etc.).

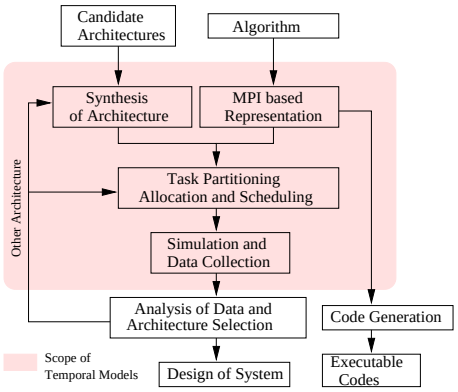

Figure 1. Architecture exploration environments.

4 Case Study I: Architecture Selection for System Design

Our design exploration environment appears very useful to evaluate different system architectures in terms of their performance. The Figure. 1 shows the typical methodology for systematic architectural trade-off analysis in early system design. This case study evaluates several design decisions for a SAR system.

SAR is an important tool for the collection of high-resolution, all-weather image data for military and remote sensing applications. The SAR algorithm has been accepted as a RASSP evaluation benchmark (Figure. 2) [11]. At the maximum PRF (Pulse Repetition Frequency) 556 Hz, the radar delivers 512 pulses of 4064 data samples for each of the four transmit/receive polarization (HH, HV, VH, VV) in 0.92 seconds. The system under design must be able to form a 512 pulse image for each of three(3 out of 4 are used) polarizations in real-time. Given a maximum PRF of 556 Hz for the radar, the real time constraint on the range processing task is 1.8 ms, the real time constraint on the azimuth processing is approximately 1.1 Gflop/sec, and the memory requirement is approximately 77 MBytes. Though the SAR algorithm is composed of a number of FFT kernels, the size of the problem, substantial computational throughput and memory requirements and real time constraints have made a successful design of the system a challenging task.

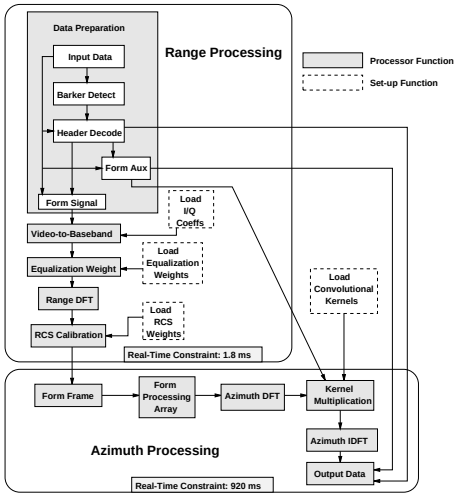

Figure 2. SAR algorithm block diagram.

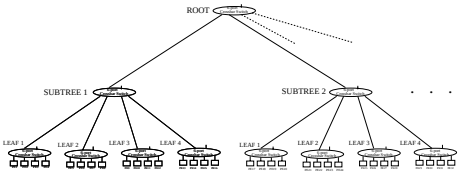

Figure 3. Target architecture for SAR.

In order to implement the SAR algorithm, we decomposed the functional block diagram into three parallel data flow graphs for each polarization. Each functional block was decomposed further into parallel blocks to exploit the maximum parallelism in the algorithm.

For a given candidate architecture, one can schedule and allocate tasks using one of many different well known techniques [9]. In this particular case, we used

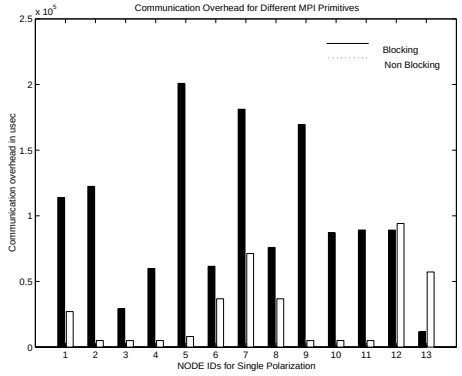

Figure 4. Communication overhead (SAR algorithm) for different MPI schemes.

CADE (Cost Driven Architecture Design Engine) to decide on allocation and schedule for a candidate architectures [4]. The target architecture in this case is a tree-network built using the Mercury Raceway crossbar fabric with 36 processors (Figure. 3). In conventional design, task mapping and scheduling are done with certain static assumptions about the underlying system. Thus, verification needs to be performed on these design decisions. Temporal models, in our environment, validate and evaluate the performance of the target architecture, task allocation and scheduling, and the effect of different communication primitives.

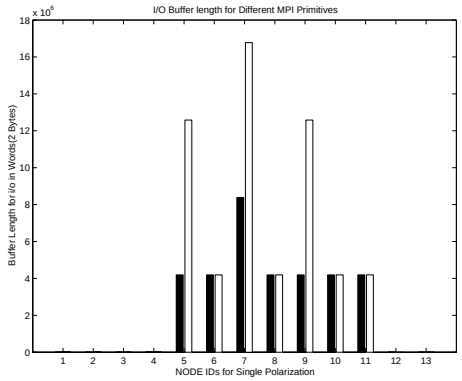

Figure 5. Effect of MPI on I/O buffer length.

While choosing the communications primitives, we ensured the precedence relation in the algorithm and group communications calls were maintained. In Figure. 4, it is observed that the modeling tool could capture the variation of communication overhead caused by different communications primitives. In addition to the communication overhead, the communication primitives also affect the I/O buffer requirements and peak I/O buffer length (Figure. 5) have been captured for different communication modes.

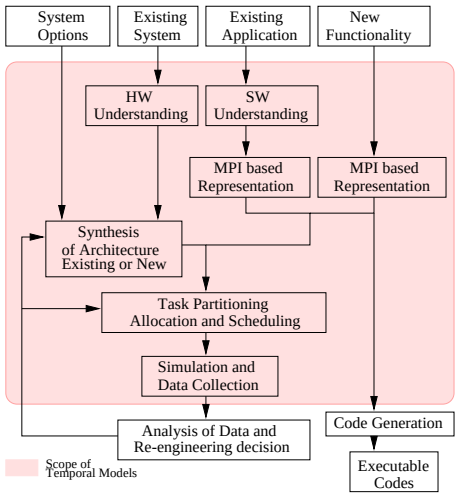

Figure 6. Re-engineering approach.

5 Case Study II: Legacy System Re-Engineering

Like system design, system re-engineering is also a challenging task since it deals with two sets of constraints — one defined by the existing legacy system and other from the upgrade requirements. However, there is no systematic approach to perform re-engineering and more often it is done in an ad hoc fashion. Using temporal modeling, we can formulate a systematic approach for system re-engineering. Re-engineering typically comprises of addition of more functionality to the system, addition or replacement of older (possibly unavailable) components with better ones, or upgrading the existing system architecture to a newer COTS system architecture. The following steps can be followed to evaluate the re-engineering trade offs while re-engineering an existing system (Figure 6).

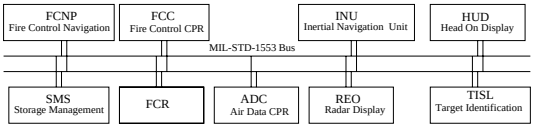

Figure 7. A legacy avionics system architecture.

The first step would be SW understanding and abstraction of the currently existing legacy application to the MPI-based abstract instruction set. Since this representation is architecture independent, we can easily perform architectural trade-offs if needed. The next

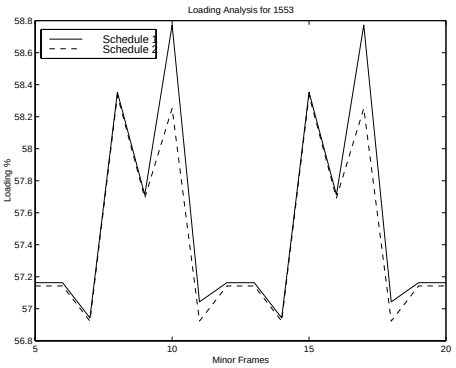

Figure 8. MIL-STD-1553 bus loading analysis.

step would be to understand the existing HW, its architecture, configuration and resources. The underlying HW can then be captured in terms of a COTS model from the reuse library. Now the designer can include the desired changes in the architecture, application SW model and co-simulate to evaluate the effects of the changes on the system.

A typical example for re-engineering would be re-engineering of a legacy avionics system for enhancing its functionality. Figure. 7 shows a commonly used avionics architecture [2]. This system uses MIL-STD-1553 serial multiplex data bus for its communication between various subsystems. The system consists of a number of sub-systems — fire controller, fire control navigation, air data unit, target identification set, radar display, head-up display, etc., [2]. Each of the subsystems perform their tasks under given real time constraints. Most of the computational and communications tasks are periodic, for example, the navigation unit updates the display unit in real time. In order to cater to all the periodic and aperiodic real time communication needs, the bus controller follows a predefined static or cyclo-static schedule to arbitrate access to the bus. Bus transactions take place in a command-response fashion. The communication schedule affects the communication latency and communication overhead of each of the nodes, thus may potentially affect the real time performance of the subsystems specially when errors and retransmission are taken into consideration. Typically, rate monotonic schedules can be designed to ensure guaranteed real time performance of all the tasks. Now, even if some simple features are added to such an existing system, the communication schedule of the controller may have to change to accommodate additional need which may affect the RT performance of the existing system. Thus a systematic approach towards evaluation of the re-engineering alternatives is very crucial. In this example, we considered the case of increasing the image formation window for the RADAR unit. This change caused additional periodic message transfers between head up display, target detector and radar unit. Figure 8 shows the loading of a MIL-STD-1553 bus for two different communication schedules, one for the existing system, other for the re-engineered schedule showing increased loading due to additional message transfers.

6 Conclusion

In this paper, we have presented a temporal/performance modeling environment/toolset that is suitable for system design and legacy system re-engineering. This virtual prototyping environment affords the capability for architecture selection and design evaluation for a variety of complex applications design and re-engineering tasks. When coupled with code-generation, this environment promises much in terms of productivity, design integrity, and efficiency.

References

[1] Url: rassp.scra.org/public/tb/honeywell/honeywell-docs.html. 1994.

[2] E. C. deLong. *Databus Systes Integration Handbook*. The SAE Press, Warrendale, PA, first edition, 1991.

[3] J.-P. Calvez. System Level Performance Model and Method. In *Meta Modeling*, pages 57–102, 1996.

[4] J. Debardelaben and V. K. Madisetti. The Economics of Design and Test of COTS-based Embedded Micro Electronics Systems. *IEEE Design and Test of Computers*, Fall 1997.

[5] L. Dung and V. K. Madisetti. Conceptual Prototyping of Scalable Embedded DSP System . *IEEE Design and Test of Computers*, pages 54 – 65, Fall 1996.

[6] L. Dung and V. K. Madisetti. RASCAL: A Library-based Environment for Rapid Architectural Synthesis of Advanced Computing Architectures . *SCIzzl-6 Workshop*, pages 97 – 103, Sept 1996.

[7] J. S. Fred Rose and C. Hein. Performance Modeling of System Architecture. *Journal of VLSI Signal Processing*, 15:97 – 109, 1997.

[8] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.

[9] V. K. Madisetti. *VLSI Signal Processing*. The IEEE Press, New Jersey, first edition, 1995.

[10] M. Snir. *MPI: The Complete Reference* . The MIT Press, Cambridge, Massachusetts, first edition, 1996.

[11] B. Zuerndorfer and G. Shaw. SAR Processing for RASSP Application. *Proceedings of 1st Annual RASSP Conference*, pages 253 – 268, Aug 1994.