An Overview of Computer Architecture and System Simulation

J. Manuel Colmenar
C.E.S. Felipe II
U. Complutense de Madrid
28300 Aranjuez, Spain

José L. Risco-Martín and Juan Lanchares
Dept. of Computer Architecture and Automation
U. Complutense de Madrid
28040 Madrid, Spain

Abstract

This work presents an overview of the modeling and simulation challenges that currently exist in the area of computer architecture design. First, we describe the requirements that a modeling and simulation framework must have in this particular application domain. Then, we review different state-of-the-art frameworks that we have classified taking into account the kind of source system they model.

1. Introduction

The framework for *Modeling and Simulation* (M&S) as described in [1], defines *entities* and their *relationships* that are central to the M&S enterprise. The entities of the framework are *source system*, *experimental frame*, *model*, and *simulator*. They are linked by the modeling and simulation relationships. Each entity is formally characterized as a system at an appropriate level of specification within a generic dynamic system. Figure 1 shows these relationships [1].

The source system is the real or virtual environment that we are interested in modeling. It can be viewed as a source of observable data in the form of time-indexed trajectories of variables. The data that has been gathered from observing or otherwise experimenting with a system is called the *system behavior database*. This data is viewed or acquired through experimental frames of interest to the modeler.

The experimental frame is a restricted set of the elements observed in the source system and the conditions under which they are observed.

The model is the system specification based on the data acquired, such as a set of instructions, rules, or mathematical equations. Models may be expressed in a variety of formalisms that may be understood as a means for specifying subclasses of dynamic systems.

Finally, a simulator is any computation system, like an algorithm, which is capable of executing a model to generate its behavior. A general purpose a simulator is able to execute a variety of models.

We can find many different source systems in the computer architecture (CA) field, not only processor or microarchitecture designs. Related topics like memory design, low power and efficient architectures, parallel architectures, multi-processor systems on chip (MPSoCs), networks on chip (NoC), graphics/gaming embedded processors, DSPs, ASIPs, etc. are usually modeled in the Computer Architecture field. Moreover, regardless of the actual source system, the early stages of all design projects in this field require simulations able to identify different problems related to functional, timing, thermal, energy, networking, performance or any other issues. In this way, we can find models of these source systems developed at different levels of description: transistor level, gate level, register-transfer level (RTL), components level, etc. Furthermore, models and simulators can be integrated in software, hardware, or both.

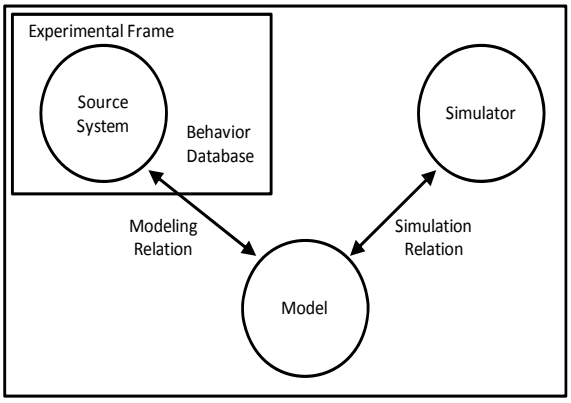

Figure 1. Basic entities in M&S and their relationships

We begin this article by reviewing some of the key issues relating to computer architecture simulation. Then, in Section 2 we describe the requirements that an M&S framework must fulfill in computer architecture simulation. Section 3 overviews different state-of-the-art M&S frameworks, and describes different simulation issues. Finally, Section 4 draws the conclusions.

2. M&S Requirements in CA

According to [2], the M&S process in computer architecture may be divided into five different steps: (1) validation and accuracy of the model; (2) selection of parameter values for components like processors or memories; (3) selection of benchmarks and input sets; (4) simulation run; and (5) performance analysis.

These steps, well described in the referenced work, are focused on the implementation phase of the M&S process. However, there exist some previous requirements that should be considered in a Computer Architecture M&S framework. We enumerate them below.

Ability of modeling at architectural level. We must start at the architectural level in order to analyze different design decisions and, after that, perform some detailed specifications. In this regard, the model allows us the description of components at high level and do not require low level descriptions.

Easy to develop and modify models. We must be able to quickly evaluate new designs and to develop elaborate models as needed. In both cases the model should generate traces or statistics as a replacement for detailed description of a resource.

Ability of modeling at different levels of detail. We must be able to develop models at various levels of detail. Note that it is counterproductive to create a very slow simulation run. Therefore, we should be able to connect a number of modules at a different level of detail. For example, to study the cache memory of a single processor, the memory can be modeled in the desired detail, while the IO components can be fairly simplified.

Mixed definition of models. The M&S framework must be able to define a mixed mode of models. This requirement allows some modules to be defied using graphical interface while others are defined by languages, such as C, C++ or Verilog.

User-friendly parameter specification. The model must allow the system designer to perform parametric specifications because much of the experimentation involves altering parameters and observing their impact on the performance.

Accurate timing modeling. The designer needs an accurate timing model which can accurately capture timing information within the architecture in as simple (high-level) manner as possible. The time required to perform a task must be accurately computed in the model; note however that this does not imply the modeling of cycle-by-cycle behavior . When we introduce metrics like bandwidth or latency, accurate timing results are critical.

The model must be easy to interrogate. The M&S framework must facilitate instrumentation for the purpose of gathering information in the form of statistics or traces of operations.

Within the context of these requirements, a computer engineer may develop the model using different tools.

One approach consists of using general programming languages such as C, C++, Java, etc., to develop both the model and the simulator. In a similar way, one can develop the model using a hardware description language like VHDL or Verilog, and integrate the model with a FPGA, for example. Using this latter approach, the model is able to interact with its environment without the necessity of a simulator.

A second approach consists of using specific M&S tools like SystemC, TLM, MatLab Simulink, Modelica, Verilog in ModelSim, etc. In this case, the system designer develops the model using an available modeling library. Then the model can be simulated with the simulator that is integrated in such tools.

Note also that there exist hybrid approaches in which a model developed in a specific M&S platform can be transformed to other one. For example, SystemC models can be compiled to VHDL.

We now examine several different state-of-the-art computer architecture M&S frameworks that accommodate the requirements outlined above.

3. Common M&S frameworks in CA

Many computer architecture M&S frameworks can be found in the literature. Many different designs and proposals are modeled and there exist a wide variety of source systems. The M&S frameworks that have been selected and analyzed have been chosen to show the heterogeneity of this field.

These frameworks are described using the classification presented in [2]. The simulators are classified into five categories, depending on the source system they model: single-processor, full-system, power consumption, multi-processor, and modular simulators. We have also added two extra categories to this classification; namely, system-on-chip and network-on-chip frameworks.

In the following discussion we describe each of those categories and include some example frameworks from the recent literature.

3. 1. Single-processor

This class of frameworks models the target microarchitecture with the aim of measuring the performance of a single processor. They are usually able to carry out different kinds of simulations from merely functional to detailed structural models.

SimpleScalar [3] is probably the *de facto* standard of past years in processor simulation. It offers a suite of compilers and simulation tools that allow the execution of benchmarks into a superscalar microarchitecture. The simulators belonging to this suite range from the simplest, namely `sim-fast`, which obtains an abbreviated set of statistics, to the more precise `sim-outorder`, which allows the configuration of elements like the type and size of the branch predictor, the number of load-store queue entries, or the cache configuration, thereby providing a detailed ensemble of statistics. These simulators are cycle-accurate because they model the behavior of the microarchitecture on a cycle-by-cycle basis. SimpleScalar simulators are also able to run benchmarks compiled for several instruction set architectures; e.g., the Alpha ISA. In addition, the model and the simulator in SimpleScalar are coded in the C language and the source is freely available. However, the model and the simulator are very coupled in the source code, especially in the detailed architectural simulator.

As an alternative to cycle-accurate simulators like SimpleScalar, in [4] the authors describe a microarchitecture simulator that characterizes the delay of each one of the stages of the processor through probability distribution functions. This approach allows the simulation of variable latency systems like asynchronous processors or globally-asynchronous locally-synchronous processors, which are not based on clock cycles, but rather on communication protocols. Based on SimpleScalar, this simulator is coded in C, and reads XML files for probability distribution function descriptions. Unlike SimpleScalar, this simulator does not accept trace files as input, but it does accept benchmarks compiled for the Alpha ISA.

Designers of embedded processors also require microarchitecture simulators to increase their productivity. In this regard, [5] proposes a generic single-microarchitecture simulation framework built in Java, able to generate an RTL (Register Transfer Level) cycle-accurate simulator. The resulting simulator is able to process compiled code as input, modeling two instruction set architectures: ARM (32-bit) and THUMB (16-bit).

In brief, the single-processor performance simulators are able to detail the execution of every module of a microarchitecture at different abstraction levels in order to provide accuracy. The model is usually a module that runs compiled code either at high-level (functional simulator), or at low-level (by describing a set of interconnected components). This detail, added to the size of the input benchmarks, generally results in slow simulator execution.

To deal with the slow execution speed, these simulators often incorporate *sampling simulation*, which consists in running a reduced set of instructions instead of the complete input benchmark. Sampling simulation, however, should not avoid those instructions not selected for simulation. These instructions will probably change the state of components like caches or branch predictors, affecting the execution of the selected instructions. The process that fills in modeled structures in order to put them into the same state as if they were running the complete benchmark, is known as *warming up*.

3.2. Full-system

Single-processor simulators do not model or run an operating system (OS). Therefore, the influence of the OS is not measured, which may lead to errors or wrong performance results from the simulator [6]. To deal with that impact, many authors propose full-system simulators where the entire run-time stack is modeled.

PTLsim [7] is a cycle-accurate full-system superscalar x86-64 microprocessor simulator and virtual machine. Highly configurable, this full-system simulator is able to model modern instruction sets like AMD64. In fact, PTLsim decodes each one of the instructions of the input benchmark into a set of micro-operations similar to the classical RISC instructions. PTLsim also allows co-simulation, where the simulator runs directly on a reference machine supporting the instruction set being simulated. By using co-simulation, PTLsim can switch between native mode (directly running on the real hardware) and simulated mode, without affecting the user code. Moreover, PTLsim supports multi-threading and multi-processor simulation. Written primarily in C++, PTLsim is publicly available through GPLv2 license.

Simics [8] is a commercial full-system simulator. Simics supports a wide range of instruction sets (models), including Alpha, PowerPC, UltraSparc, and x86-64, and includes device models that are detailed enough to run the actual device drivers. Also, Simics can boot and run unmodified operating systems such as Linux, Solaris, and Windows XP. However, Simics does not include cycle-accurate simulation features below the x86 instruction level, as PTLsim does.

3.3. Single-processor power consumption

One of the main concerns of processor designers is energy consumption. Therefore, accurate specialized simulators are needed for this purpose. In the collection of such tools we have Wattch, which is the most widely used power consumption simulator [9].

Wattch [10] is a framework for analyzing and optimizing microprocessor power dissipation at the architecture-level. Wattch is based on SimpleScalar, and models the energy consumption of an Alpha microarchitecture. Coded in C, it allows the integration of its model with other architectural simulators in order to assign the energy consumption of any of the architectural simulator operations.

SimWattch [9] is a particularly noteworthy tool. It integrates the system-level support provided by the Simics full-system simulator with the cycle-level microarchitecture timing and power estimations provided by Wattch. As a result, SimWattch estimates the performance and power consumption of the superscalar microprocessor architectures modeled by Simics on a complete system environment.

Following the pattern of Wattch and SimWattch, the power consumption simulators usually complement the energy estimation feature of other simulators by providing a model that measures energy consumption.

3.4. Multi-processor

Simulation tools are even more relevant in the realm of simultaneous multi-threading (SMT) and chip multi-processors (CMP). Unfortunately the SMT and CMP simulators compound the time requirements noted previously for single processor architecture simulation. T main he goal of these simulators is therefore is to reach fast simulation times for such aggregated architectures. These simulators must also take into account that aggregated systems usually include shared resources which may require synchronization.

As in single-processor modeling, sampling simulation is a common technique. For instance,

[11] discusses the sampling for SMT simulations. The authors' perspective is that the creation of a sampling approach to SMT requires the determination of how far to fast-forward each individual thread between samples. The paper proposes an efficient SMT simulation methodology that estimates average performance over all starting points when running multiple programs on an SMT processor. Their simulation environment includes M5, which is based in SimpleScalar.

Another example of sampling simulation of multi-processor systems is provided in [12]. This work describes a software structure called the memory timestamp record (MTR), which behaves as a compressed snapshot of memory reference patterns. The MTR helps with fast-forwarded simulations allowing the reduction of simulation times in relation to the typical functional fast-forwarding technique. The full-system simulator that is tested is a blending of their own source code and an open source C++ system emulator called Bochs [13].

In [14] a pragmatic implementation is proposed. The idea is to take advantage of CMP systems both by running CMP simulations in parallel systems and also by replacing simulated processors by real processors. In order to reach that goal, the authors proposed a structural model consisting of a collection of components that execute concurrently and are connected by signals. The model is designed using the Liberty Simulation Environment (LSE) [15], which is a modular simulator that is described in Section 3.5. Their framework combines the instantiation and connection of components with the code representing each component's behavior. As a result, the framework is able to automatically generate a parallelizable simulator from a model specification. The integration of hardware components is feasible by replacing the behavior code by a hardware interface connection. The paper includes a description of a successful integration and simulation of eight PowerPC 405 cores implemented on a FPGA into a CMP model.

In [16] the authors study *statistical simulation* as a fast simulation technique for CMP design space exploration. This kind of simulation manages synthetic traces created from a set of statistics taken from real traces of a target benchmark. The major advantage is that the synthetic trace is much shorter compared to a real program trace and this leads to substantial simulation speedups.

3.5. Modular simulators

Complex simulators, either single-processor, full-system or multi-processor, are usually implemented as monolithic pieces of software where the model is very much coupled to the simulator. There is a requirement for all the software elements to run. Interface mechanisms that allow components to be reused or even concurrently executed are not provided. Therefore simulations are hard to parallelize and modeled components can rarely be reused.

Modular simulators strive to develop the idea of independent components that, running concurrently, can be instantiated and connected to any other system elements. The consequence of applying this basic concept of M&S is the generation of more easily parallelizable simulators, which obtain faster execution times on current CMP computers.

One example of modular simulator is the Liberty Simulation Environment (LSE) [15]. LSE maps each hardware component to a single software function and allows the designer to build complex processor components by instantiating components and specifying their connections hierarchically. Debugging is then a simpler task because each component can be tested individually and complex designs are manageable due to the hierarchical design feature.

A recent work, ([17]) presents a modular framework for stream-oriented multi-processor system-on-chip (MPSoC). The framework processes program traces instead of compiled binaries. Each trace generates a set of ordered events that are consumed by each component of the system, generating both results and new events. The components of the system, representing architectural elements, are modeled as virtual machines able to share resources, allocate dynamic memory and perform multi-

hop communications. The framework was developed using SystemC.

3.6. System-on-Chip

Earlier we noted that simulation of complex systems requires approaches different from detailed architectural simulation in order to achieve realistic simulation times. This is also the case for system-on-chip (SoC) simulation.

An SoC may contain components like processors, memories, analog devices and even radio-frequency elements in the same circuit. In brief, SoC's are typical examples of embedded systems. Some examples of SoC frameworks are provided in the following discussion.

In [18], the authors present a co-simulation platform for SoC design space exploration. The idea is to describe the hardware components using SystemC, and substitute the general-purpose processor cores with C and C++ Instruction Set Simulators (ISSs). The integration of all the components of the system is described in SystemC. The resulting platform allows a designer to explore different SoC alternatives by reusing components. However, the platform only provides statistics of execution speed and system bus communications; no other metrics are implemented.

SimSoC [19] is a full-system simulator for SoC. The hardware modeling is developed in SystemC, while the communications between components is described through Transaction Level Modeling (TLM). TLM refers both to a level of abstraction and to the SystemC-based library used to implement transactional models. SimSoC also integrates ISS's as SystemC modules with TLM interfaces to the other platform components. These ISS's perform the modeling of processor behavior. However, only the ARM5 ISS was tested at the time the work was presented.

Within the SoC domain there are even more complex kinds of SoC in terms of modeling and simulation. These systems are known as Multiple Configurable Processors System-on-Chip (MCPSoC). MCPSoC's have both performance and power advantages for embedded applications. However, once the processors are configured to perform a task, the operating system that will run the MCPSoC needs to be adapted to each particular configuration of the processors. In particular, the most hardware related parts of the operating system like the hardware abstraction layer, APIs and I/O device drivers have to be modified accordingly. In this regard, [20] proposes a hybrid simulation platform to explore the MCPSoC design space. Instead of realizing HAL, API's and driver API's with the assembly code of target processors, they implement those components on the host machine as TLM SystemC modules. This replacement allows the use of hybrid ISS instead of regular ISS. The hybrid ISS speeds up the simulation time by simplifying the communication with underlying modeled hardware.

3.7. Network-on-Chip

Networks-on-chip (NoC's) go one step further than SoC's. An NoC is a design where many SoC's and/or intellectual property (IP) cores from possibly different clock domains get communicated in order to perform a task. Then, NoC's can span synchronous and asynchronous clock domains or use asynchronous logic to perform communications. NoC's usually improve network techniques like bus access or crossbar connection obtaining reliable and fast communications between the components of the system.

Thus, one of the challenges of NoC design is testing of the networking policy to be implemented. This requires simulating the nodes, connections and constraints, and the traffic of the network.

Type of Source System	Model	Simulator	Inputs
Single-processor / full-system	RTL and component level C, C++	Coupled to the model C, C++	Binary code
SMT / CMP	Functional descriptions C++ , LSE	Decoupled, some HW support	Binary / Trace files
SoC / NoC	Independent of simulator SystemC	Totally decoupled	Trace Files

Table 1: Summary of common M&S features for different kinds of source systems.

Recently an NoC simulation environment which models on chip components and traffic generators able to behave following different patterns was presented in [21]. The simulator was built using SystemC and TLM and can also be used to model SoC's.

A similar NoC simulator developed using SystemC is described in [22]. This is a discrete event and cycle accurate simulation environment that allows simulation with various options available for topology, switching technique, virtual channels, buffer parameters, routing mechanism, and applications.

4. Conclusions

Computer architecture simulation tries to achieve accurate results with realistic simulation times. Many factors influence these issues: modeling detail of the source system, language employed to code both the model and the simulator, benchmarks and input sets for simulations, etc.

In this article we have reviewed the most important problems that arise and have described the requirements that an M&S framework for computer architecture systems should fulfill. In addition, we have given an overview of several simulators for different computer architectures from single-processor, through SMP, CMP, SoC and NoC systems. Table 1 provides a summary of this overview.

References

1. B. P. Zeigler, T. Kim and H. Praehofer, "Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems", Academic Press, 2000.
2. J. J. Yi and D. J. Lilja, "Simulation of computer architectures: simulators, benchmarks, methodologies, and recommendations0", IEEE Trans. on Computers, 55 (3), pp. 268- 280, 2006.
3. T. Austin, E. Larson, and D. Ernst, "SimpleScalar: An Infrastructure for Computer System Modeling," Computer, vol. 35, no. 2, pp. 59-67, 2002.
4. J. M. Colmenar, O. Garnica, J. Lanchares and J. I. Hidalgo, "Characterizing asynchronous variable latencies through probability distribution functions" Microprocessors and Microsystems, Volume 33, Issues 7-8, pp. 483-497, 2009.
5. A. K. Ghanem, A. H. El-Mahdy and I. A. El-Salam, "A Cycle-Accurate Micro-Architecture Simulation Framework for Embedded Processors," Computer Engineering and Systems, The 2006 Int'l Conf. on, pp. 71-76, 2006.
6. H. Cain, K. Lepak, B. Schwartz, and M. Lipasti, "Precise and Accurate Processor Simulation," Proc. Workshop Computer Architecture Evaluation Using Commercial Workloads, 2002.
7. M. T. Yourst, "PTLsim: A Cycle Accurate Full System x86-64 Microarchitectural Simulator," Performance Analysis of Systems & Software, 2007. ISPASS 2007. IEEE Int'l Symp., pp. 23-34, 2007.
8. P. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Halberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A Full System Simulation Platform," Computer, vol. 35, no. 2, pp. 50-58, Feb. 2002.
9. J. Chen; M. Dubois, and P. Stenstrom, "Integrating complete-system and user-level performance/power simulators: the SimWattch approach," Proc. IEEE Int'l Symp. Performance Analysis of Systems and Software, pp. 1- 10, 2003.

10. D. Brooks, V. Tiwari, and M. Martonosi. “Wattch: a framework for architectural-level power analysis and optimizations”. SIGARCH Comput. Archit. News 28, 2, 83-94. 2000.
11. M. Van Biesbrouck, L. Eeckhout, and B. Calder, “Considering All Starting Points for Simultaneous Multithreading Simulation,” Proc. IEEE Int’l Symp. Performance Analysis of Systems and Software, pp. 143-153, 2006.
12. K.C. Barr, H. Pan, M. Zhang, and K. Asanovic, “Accelerating Multiprocessor Simulation with a Memory Timestamp Record,” Proc. IEEE Int’l Symp. Performance Analysis of Systems and Software, pp. 66-77, 2005.
13. Bochs: http://bochs.sourceforge.net/ 2011
14. D. A. Penry, D. Fay, D. Hodgdon, R. Wells, G. Schelle, D.I. August, and D. Connors, “Exploiting Parallelism and Structure to Accelerate the Simulation of Chip Multi-Processors”, Proc. 12th Int’l Symp. High-Performance Computer Architecture (HPCA), pp. 27-38, 2006.
15. M. Vachharajani, N. Vachharajani, D. A. Penry, J. A. Blome and D. I. August, "Microarchitectural exploration with Liberty," Microarchitecture, 2002. (MICRO-35). Proc. 35th Annual IEEE / ACM Int'l Symp. on, pp. 271-282, 2002.
16. D. Genbrugge and L. Eeckhout, "Chip Multiprocessor Design Space Exploration through Statistical Simulation," Computers, IEEE Trans. on , vol. 58, no. 12, pp. 1668-1681, 2009.
17. Kai Huang, I. Bacivarov, Jun Liu and W. Haid, "A modular fast simulation framework for stream-oriented MPSoC,". IEEE Int'l Symp. on Industrial Embedded Systems, pp. 74-81, 8-10, 2009.
18. Y.W. Hau and M. Khalil-Hani, "SystemC-based HW/SW co-simulation platform for system-on-chip (SoC) design space exploration", Int'l Conf. on Electronic Design, pp.1-6, 2008.
19. C. Helmstetter, V. Joloboff, and Hui Xiao, "SimSoC: A full system simulation software for embedded systems", IEEE Int'l Workshop on Open-source Software for Scientific Computation (OSSC), pp. 49-55, 2009.
20. Hao Shen, F. Petrot, "A flexible hybrid simulation platform targeting multiple configurable processors SoC," Design Automation Conf. 15th Asia and South Pacific, pp.155-160, 2010.
21. G. N. Khan and V. Dumitriu, "Simulation environment for design and verification of Network-on-Chip and multi-core systems", Modeling, Analysis & Simulation of Computer and Telecommunication Systems. IEEE Int'l Symp. on , pp.1-9, 21-23 Sept. 2009.
22. Wang Zhang, Ligang Hou, Da Chang, Zhenyu Peng and Wuchen Wu, "A simulation environment for Network-on-Chip based on SystemC," Int'l Conf on Computer Application and System Modeling., vol.9, no., pp.V9-79-V9-83, 22-24 Oct., 2010.

Author Biographies

J. Manuel Colmenar obtained a M.S. degree in Computer Engineering in 2001, and received a Ph.D. degree in 2008, both from the Complutense University of Madrid (UCM). He is currently an Assistant Professor of Computer Science at the Aranjuez campus of the UCM. His current research interests include evolutionary algorithms, DEVS, SoC and MPSoC architectures, and asynchronous systems and microprocessors.

José L. Risco-Martín is Assistant Professor at the Computer Architecture and Automation Department of Complutense University of Madrid (UCM), Spain. His research interests focus on computational theory of modeling and simulation, with emphasis on Discrete Event Systems Specification (DEVS), methodologies for integrated systems and high-performance embedded systems, including new modelling frameworks to explore thermal management techniques for Multi-Processor System-on-Chip, dynamic memory management and memory hierarchy optimizations for embedded systems, and low-power design of embedded systems.

Juan Lanchares is Associate Professor at the Computer Architecture and Automation Department of Complutense University of Madrid (UCM), Spain. He received the MS degree in Physics from the UCM in 1990 and he received his Ph.D degree in 1995. His research interests are SMT processors, Asynchronous Techniques for System Design and the Genetic CAD tools for HW design.