ISSN No. 0976-5697

Volume 2, No. 4, July-August 2011

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Validating SIM-A Simulator with ARM Based Keil Software

Dr Manoj Kumar Jain*
Associate Professor,
Department of Computer Science,
MLSU University, Udaipur
manoj@cse.iitd.ernet.in

Gajendra Kumar Ranka
Research Schlor,
Department of Computer Science
MLSU University, Udaipur
Gajendra_ranka@hotrmail.com

Abstract: While general purpose processors reach both high performance and high application flexibility, this comes at a high cost in terms of silicon area and power consumption. In systems where high application flexibility is not required, it is possible to trade off flexibility for lower cost by tailoring the processor to the application to create an Application Specific Instruction set Processor (ASIP) with high performance yet low silicon cost. If we look at the rapid rate at which mobile technology is developing and the constant need for miniaturization, ASIPs seem to be poised in a stronger position compared to ASICs. The major contribution of this paper lies in verifying or substantiating SIM-A with Keil Software. Simulator SIM-A measures cycle count for application executed on processor. This paper focuses on working with Keil Software and its configuration required to run any software on ARM based keil software.

Keywords: ASIP, Application Specific Instruction Processors, Retargetable Simulator, Embedded Systems, Processors, ASIP Simulators, Design Space Exploration, ARM , Keil software.

I. INTRODUCTION

Modern electronics are controlled by processors that must meet strict constraints in terms of performance, cost, size and power consumption. In a competitive market place, performance and cost are critical in differentiating one product from another.

An ASIP is a processor that is designed to efficiently execute the software for a specific application. Although incorporating a complete system on a single IC may improve performance, cost, and power consumption requirements, such a high level of integration constraints the size of the system components.

A. Steps in ASIP Synthesis

Various methodologies have been reported to meet these requirements. All these have been studied and five steps are suggested for synthesis of ASIPs [1]

Application Analysis: Application is normally written in High level language. Sometimes SUIF can be used as intermediate format. Analysis of the application is essential as it provides the essential requirement from the application that can guide for hardware synthesis as well as instruction set generation.

Architectural Design Space Exploration: Output of the Application analysis step along with the range of architecture for Possibility of suitable architecture is explored and the best architecture is selected that satisfy the different characteristics like minimum hardware cost, performance and power.

Instruction Set Generation: Till this step we have identified application requirements and the suitable architecture.

Code Synthesis: Till this step, architecture template, instruction set, and application are identified. This step generates the code. Generated code can be retargetable code generator or compiler generator.

Hardware Synthesis: In this step the hardware is generated using the ASIP architectural template and instruction set architecture using standard tools **[**1.2].

B. Architecture Design Space Exploration

System on Chip designs has various goals and objectives. Design space consists of a set of parameters. Architecture under consideration requires a range of good parameter to explore. These parameters may take up the different values.

Some of the parameter suggested can be functional unit of different type, Storage units, interconnect resources, number of memory units etc. Further the parameters can also be extended to size of instruction cache and size of data cache. This has been a very crucial step for ASIP design. Design Space exploration helps the SOC designers to make the trade-offs between these goals and arrive at the "optimal" design. Designers explore changes to the architecture or the instruction-set of the processor-memory system. Designers select a suitable architecture that satisfy the performance and power constraint and having minimum hardware cost. Architecture is defined using some suitable architecture description language (ADL).

C. Techniques for Performance Estimation

Two major techniques have been used for performance estimation. They are scheduler based and simulator based. In Scheduler based approach, application is scheduled to generate the output like cycle count. Architectural component is already identified at this stage. Target processor architecture can be given in the form of description file.

In Simulator based approach, application under consideration runs on a simulator. Depending upon the architecture selected in above steps, application is simulated to compute the performance.

Processor Models are extensively used in system design process. The system design process starts with an application and its implementation. Then the model is tested for its performance and other aspects. In such a scenario an integrated environment is required for the designer where several tools exist like simulator, assembler, compiler etc. Rewriting the tools after each design change is a tedious job. Hence automatic generation of these tools is more desirable according to the design changes.

D. Existing Retagetable Simulators

Retargetable functional simulator (Fsimg) [2] focus on tools that deal with the machine language of processors, like assemblers, disassembler, instruction set simulator etc.Retargetable Function Simulator (Fsimg) was designed using Sim-nML language which is primarily an extension of the nML [3] language for processor modeling. Fsimg takes the specification of the processor in the intermediate representation [4] and an executable for the processor in ELF [5] format and generates a functional simulator (Fsim) which in turn gives the functional behaviour of the processor model for the given program.

II. RELATED WORK

Over the past several decades a considerable amount of research has been performed in the area of computer architecture simulation. These simulators can be broadly divided into several categories: full-system simulators, Instruction Set Architecture (ISA), and retargetable Simulators. Each category serves an entirely different purpose, but all have been used for the advancement of computer architecture research.

The purpose of full-system simulators is to model an entire computer system including the processor, memory system and any I/O. These simulators are capable of running real software completely unmodified just like a virtual machine. There are many simulation suites that take this approach, including PTLSim [6], M5 [7], Bochs [8], ASIM [9], GxEmul [10] and Simics [11]. Simics has several extensions that constitute their own full-system simulators such as VASA [12] and GEMS [13].

ISA simulators are less descriptive than full system simulators. Their objective is to model processor alone.ISA simulators performs the various functionalities.

It simulate and debug machine instructions of a processor type that differs from the simulation host, it also emphasis on investigating how the various instructions (or a series of instruction) affect the simulated processor. Hence modeling of the full computer system is unnecessary and would impose additional delay and complexity. Example of this type of simulator includes SimpleScalar [14], WWT-II [15], and RSIM [16]. Over the past decade, a few interesting ADLs have been introduced together with their supporting software tools. These ADL include MIMOLA, UDL/I, nML, ISDL, CSDL, Maril, HMDES, TDL, LISA, RADL, EXPRESSION and PRMDL.

III. EXISTING RETARGETABLE SIMULATORS

Anahita Processor Description Language (APDL), APDL [17] is one of the most recent contributions in the area of retargetable simulator. The language was introduced in 2007 by N. Honarmand et al. from the Shahid Beheshti University, IRAN. The Primary difference between APDL and other ADLs is the addition of Timed Register Transfer Level (T-RTL), which enables the simulation designer to define the latencies and hardware requirement of the processor operations. This separation of configuration data enables APDL to better integrate with external software for analysis as the T-RTL data is organized separately from the remainder of the processor description. Moreover, APDL can describe both instruction and structure descriptions of a target processor.

The Pascal-like syntax of APDL is clearly more intuitive than many other ADLs such as LISA and EXPRESSION. While the language is easier to read and understand, the researchers have not yet implemented a compiler to produce simulations. Furthermore, despite APDL's relative ease, users are still faced with the task of learning the details of the syntax.

ISDL [18] was introduced in 1997 by G.Hadjiyiannis, S.Hanono, and S. Devadas from Massachusetts Institute of Technology. The purpose of ISDL was to provide a language for describing instruction sets along with a limited amount of details of a processor structure for the automatic construction of compilers, assembler, and simulators. ISDL enables users to define their target processors in several ways. First, users can define operations, their format, and the associated assembly language instruction. Second users can define the storage resources available to the processor, including the register file and memory. Third users can define constraints in the processor such as instructions requesting the same data path, or restrictions regarding assembly syntax.

ReXSim [19] was introduced in 2003 by a computer architecture research team at Irvine. ReXSim is an extension of EXPRESSION language which sought to improve simulation speed by integrating a novel method of decoding instructions of the simulated program before execution of the simulation. As a result, the instruction decoding process was removed from the execution loop of the simulator, and thus improved the simulation speed significantly. Using this method, the team was able to produce retargetable simulations that showed performance in excess of major simulators like SimpleScalar, which is widely considered to be a simulation performance benchmark.

Reduced Colored Petri Net (RCPN) [20] was introduced in 2005 by M.Reshadi and N. Dutta from University of California, Irvine. RCPN takes a vastly different approach to retargetable simulation, in which pipelines are modeled using a simplified version of Colored Petri Nets (CPN). Petri Nets are graph based mathematical method of describing a process. The nodes of the graph represent particular discrete events, states, or functions, and the graph edges represent the transitions of data between nodes. The transitions can be enabled or disabled based on conditions specified at the nodes.

The purpose of RCPN is to provide retargetable simulations for modeling of pipelined processors. RCPN reduces the functionality of a regular CPN by limiting the capabilities of the nodes in the graph for the purpose of increasing simulation speed and usability. Additionally, RCPN takes the advantage of some of the natural properties of CPNs to prevent structural and control hazards.

Retargetable functional simulator (Fsimg) [21] focus on tools that deal with the machine language of processors, like assemblers, disassembler, instruction set simulator etc. The objective was to have a single processor model for all the tools. Hence Retargetable Function Simulator (Fsimg) was designed using Sim-nML language which is primarily an extension of the nML language for processor modeling. Fsimg takes the specification of the processor in the *intermediate representation* and an executable for the processor in ELF.

Format and generates a *functional simulator (Fsim)* which in turn gives the functional behaviour of the processor model for the given program. Around 237 instructions have been specified with the resource usage model and pipeline. *Macro Preprocessor (nMP)* for processing Sim-nML macros is implemented.

It has some limitation. Fsimg is imposing a strong restriction on specification writing. Current bit-operator library supports only integer data types. The trace produced by Fsim is not compressed. It makes it difficult to handle and process trace files. It is very slow

The LISATek [22] processor design flow is based on LISA 2.0 processor models. Given a LISA model, the LISATek tool is able to generate instruction-set simulators for the processor under design. Typically, the debugger in form of a dynamic library directly uses the generated simulator. However, a compiled static simulator library is also generated, and specifications exist to integrate it into the system environment. The system environment would be the MPARM. All the core models generated by the LISATek suite, regardless of the nature of the ASIP at hand, have the same interface. The interaction is based upon four key pillars:

- The simulated core can be cycled by calling specific functions. If the processor is modelled in an instruction-accurate fashion, then the generated model can be stepped on an instruction basis. On the other hand, a model derived from a cycle-accurate LISA description can be stepped on both instruction and cycle basis.
- Core-initiated communication (e.g. reads, writes) is performed through a specific Application Programming Interface (API). It is the task of the external program to provide an implementation of said API.
- System-initiated communication (e.g. interrupts), if any, can be forwarded to the core when cycling it, and therefore on a fine-grain cycle-by-cycle basis, by proper flipping of extra pins. Of course the LISA core model must be made aware of the meaning of these extra pins to take proper action.
- An external LISATek Debugger tool can be interfaced to the core via the IPC (Inter-Process Communication) mechanism. The external program must simply invoke the Debugger with proper references; subsequently, the LISATek model and the Debugger interact autonomously.

The implementation of these function calls depends completely on the communication method used in the system. The implemented API will translate the requests into SystemC signals which can be understood by the MPARM [23] platform. The Assessment of the performance of alternative hardware communication is not addressed. Retargetability is poor.

All of these simulators use techniques to speed up the execution of application programs. This is achieved by minimizing the amount of details about the processor, needed for program execution on the simulator. Even though some of these previous approaches target ADL-based automatic toolkit generation and DSE, not much work has been done in bringing together these elements in an early DSE environment. Furthermore, previous approaches are restricted to certain classes of processor families and assume a fixed memory/cache organization. For a wide variety of such processor and memory IP library, the designer needs to be able to specify and analyze the interaction between the processor instruction set and architecture, and the application and explore the different points in design space.

This problem is addressed in SIMPRESS simulators. The EXPRESSION ADL captures both the instruction set and architecture information for a design draw from an IP library. The library contains a variety of parameterizable processor cores and customizable memory / cache organizations. Simpress produces a structural simulator capable of providing detailed structural feedback in terms of utilization, bottle-necks in the processor architecture. The processor-system description is input using a graphical schematic capture tool, called V-SAT, that outputs an Expression Description which is fed into the toolkit generators to produce DSE tools. The SIMPRESS generated simulator provides feedback information which is back-annotated to the same V-SAT graphical description.

Though SIMPRESS Simulators addresses many issues, it has certain limitation. The application having function calls are not supported. Compilation steps exist in three passes: PcProGUI, Expression console, acesMIPS console. Basically it is very complex to understand the process of compilation and simulator. The Application needs .proc and .def file. The .c program generates these files. There is no clear cut method as how .c is converted to .proc and .def, especially in case of windows environment. This is strong limitation as we can not simulate our own program written in .c. this has to be first converting to .procs and .defs and for that we need to depend on their servers to provide for the same, which is not functional right now.

In order to overcome all these complexities, we suggest a simple and elegant solution. Just there is a need to provide the standard application program in the form of scheduled and optimized code along with the processor description to our Simulator and you will get the cycle count as an output of the simulation.

IV. OVERALL APPROACH

Application or a set of application in the form of High Level Language is taken as input and it given as input to retargetable compiler. Architecture description is also given input to retargetable compiler. Retargetable compiler generates the schedule and optimized code. This code is given as input to Simulator. None of the existing simulator provides and easy GUI to enter the processor components and simulate the code for target host.

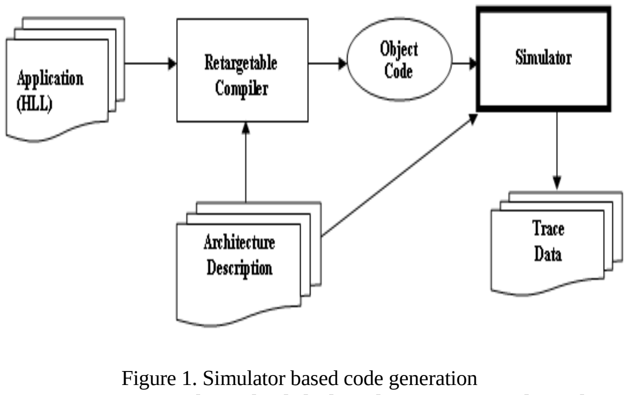
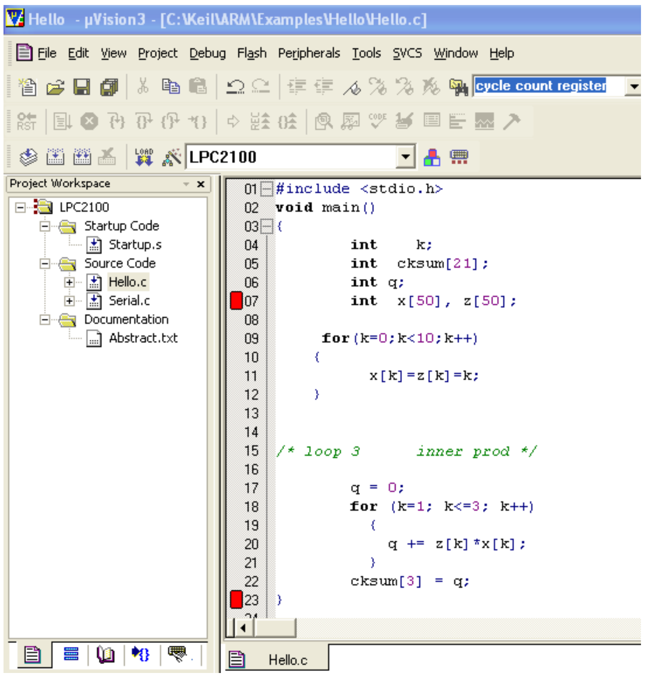


Figure 1. Simulator based code generation

We are assuming the scheduled and optimize code to be generated from retargetable compiler and this code along with the Processor description or Architecture description is given as input to the Simulator. The Simulator generates the data in the form of cycle count.

V. KEIL SOFTWARE

Keil Software development tools for the ARM microcontroller family support every level of developer from the professional applications engineer to the student just learning embedded software development. µVision3 ensures easy and consistent Project Management. A single project file stores source file names and saves configuration information for Compiler, Assembler, Linker Debugger, Flash Loader, and other utilities. The Project menu provides access to project files and dialogs for project management. When microvision 3 Project started target device needs to be selected from the device database. It displays only those option that are relevant to the selected device.

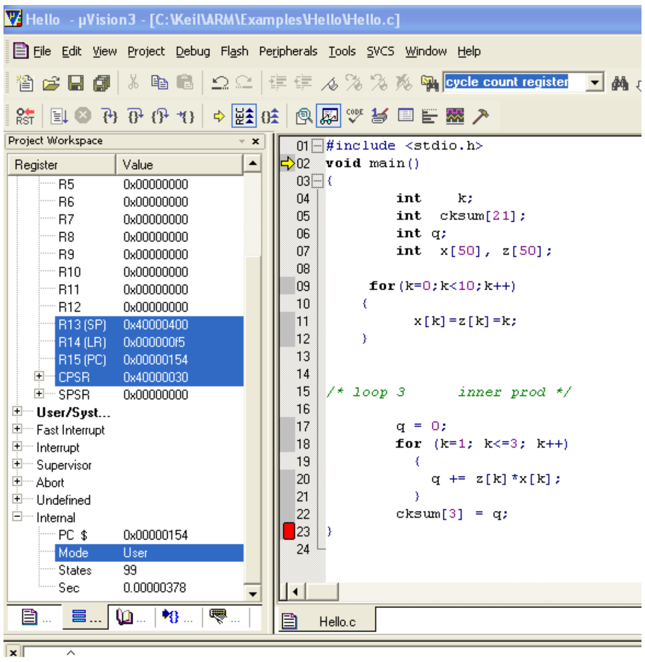


```
#include <stdio.h>
void main()
{
        int    k;
        int    cksum[21];
        int  q;
        int    x[50], z[50];

    for(k=0;k<10;k++)
      {
            x[k]=z[k]=k;
      }

/* loop 3       inner prod */

        q = 0;
        for (k=1; k<=3; k++)
          {
             q += z[k]*x[k];
          }
        cksum[3] = q;
}
```

Figure 2. Keil editor to write any program

The instructions about how to write Keil ARM can be listed as follows.

- Open Keil uVision Program which is Text Editor of Keil, that ARM uses for writing C Language Source Code Program as shown in figure2
- Set default value to translate uVision3 Code to use with Keil uVision3 Program and Keil ARM. Click Project Components, Environment, Books… then select default value to use Complier titled Select ARM Development Tools.
- Open the project and in turn open the .c program and start debugging session. Left hand pane shows the details of the Register details and the states as shown in the figure 3.


Hello - µVision3 - [C:\Keil\ARM\Examples\Hello\Hello.c]
File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
cycle count register
Project Workspace
Hello.c

Register	Value
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x40000400
R14 (LR)	0x000000f5
R15 (PC)	0x00000154
CPSR	0x40000030
SPSR	0x00000000
User/Syst...	
Fast Interrupt	
Interrupt	
Supervisor	
Abort	
Undefined	
Internal	
PC $	0x00000154
Mode	User
States	99
Sec	0.00000378

```
#include <stdio.h>
void main()
{
        int    k;
        int    cksum[21];
        int  q;
        int    x[50], z[50];

    for(k=0;k<10;k++)
      {
            x[k]=z[k]=k;
      }

/* loop 3       inner prod */

        q = 0;
        for (k=1; k<=3; k++)
          {
             q += z[k]*x[k];
          }
        cksum[3] = q;
}
```

Figure 3 Editor showing the register status and timing information

All instructions are 32 bits long. Most instructions execute in a single cycle. Every instruction can be conditionally executed. Data processing instructions act only on registers. Three operand format Combined ALU and shifter for high speed bit manipulation Specific memory access instructions with powerful auto-indexing addressing modes.

ARM has 37 registers in total, all of which are 32-bits long.

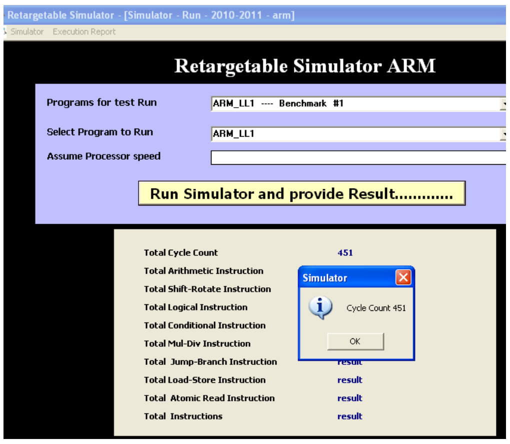
- 1 dedicated program counter
- 1 dedicated current program status register
- 5 dedicated saved program status registers
- 30 general purpose registers

However these are arranged into several banks, with the accessible bank being governed by the processor mode. Each mode can access.

- a particular set of r0-r12 registers
- a particular r13 (the stack pointer) and r14 (link register)
- r15 (the program counter)
- cpsr (the current program status register) and privileged modes can also access
- a particular spsr (saved program status register)

VI. PERFORMANCE ESTIMATES AND VALIDATION OF SIMULATOR

Benchmark programs are selected and run on SIM-A Simulator as shown in figure 4. The Framework is based ARM like processor architecture. There is a 32-bit wide general purpose register file and a 32-bit wide floating point register file, each containing 32 registers.



Retargetable Simulator - [Simulator - Run - 2010-2011 - arm]

Simulator Execution Report

Retargetable Simulator ARM

Programs for test Run	ARM_LL1 ---- Benchmark #1
Select Program to Run	ARM_LL1
Assume Processor speed	

Run Simulator and provide Result............

Total Cycle Count	451
Total Arithmetic Instruction	
Total Shift-Rotate Instruction	
Total Logical Instruction	
Total Conditional Instruction	
Total Mul-Div Instruction	
Total Jump-Branch Instruction	result
Total Load-Store Instruction	result
Total Atomic Read Instruction	result
Total Instructions	result

Simulator

Cycle Count 451

OK

Figure 4: SIM-A showing cycle count of ARM based LL1 Benchmark Program

Table 1: Benchmark Programs along with Descriptions

SNo	Name	Description
1	Benchmark#1	Excerpt from a hydrodynamic code
2	Benchmark#2	Standard Inner product function of Linear Algebra
3	Benchmark#3	Excerpt from a Tridiagonal Elimination routine
4	Benchmark#4	First Sum
5	Benchmark#5	First Difference

Table 1 lists all the benchmarks programs that have been used to validate the simulators. Table 2 shows the actual data collected while running ARM Based keil software and SIM-A Simulator. Graphical representation is shown in figure 4. After running this benchmark program on the SIM-A as well as SimpleScalar Simulator, following results were obtained.

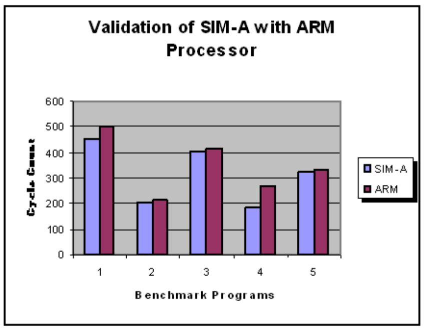


Figure 5: Comparative analysis of SIM-A and ARM based Keil

VII. CONCLUSION AND FUTURE DIRECTION

In this paper we have verified SIM-A Simulator for ARM Based Keil simulator. The paper discusses the working of ARM based keil software. The different customization needed to run the application program has been discussed in detail.

SIM-A Simulator developed at our MLSU embedded Lab generates the performance estimates for the application under consideration. The cycle accurate, structural simulator generated using SIM-A allows the user to collect statistics called cycle count. It definitely helps the designer to analyze the design and modify the critical portions.

The SIM-A environment has been designed to allow modeling of diverse range of processors. This has been demonstrated to an extent through the modeling of RISC processor with traditional memory hierarchies.

In future, it should be used to model novel memory hierarchy and other classes of processors such as DSP's.

VIII. REFERENCES

[1] M.K. Jain, M. Balakrishnan, Anshul Kumar. "ASIP Design Methodologies: Survey and Issues "In proceedings of the IEEE/ACM International Conference on VLSI Design. (VLSI 2001)", pages 76-81, January 2001.

[2] Y Subhash Chandra. Retargetable functional simulator – M.Tech Thesis, Department of Computer Science, IIT Kanpur, June 1999.

[3] M. FREERICK, The nML Machine Description Formalism, July 1993.

[4] N.C. JAIN, Disassemble using High level Processor Models. Master's thesis, Department of Computer Science and Engg, IIT Kanpur, Jan 1999.

[5] UNIX System V Rel 4, Programmers Guide : ANSI C and Programming Support Tools. PHI, New Delhi 1992. Executable and Linkable format (ELF), Tools Interface Standards (TIS), Portable Formats Specification, Version 1.1.

[6] M. Yourst, "Ptlsim." http://www.ptlsim.org/. Jan. 2010.

[7] "M5." http://www.m5sim.org. Jan2010.

[8] "bochs: The open source IA-32 emulation project." http://bochs.sourceforge.net/. Jan. 2010.

[9] J. Emer, P.Ahuja, and E.Borch, "Asim: A performance model framework" pp.68-76, 2002.

[10] "Gxemul" http://gxemul.sourceforge.net/ Jan 2010.

[11] P.M et al. , "Simics : A Full system simulation platform, " Computer, Vol.35, pp. 50-58, 2002.

[12] D. Wallin, H.Zeffer, M.Karlsson, and E.Hagersten, "Vasa: A Simulator infrastructure with adjustable fidelity," Parallel and Distributed Computing, 2005.

[13] M.M. et al., "Multifacets general execution-driven multiprocessor simulator (gems) toolset," SIGARCH Computer Architecture News, pp. 92-99, 2005.

[14] "SimpleScalar LLC." http://www.simplescalar.com/, August 2010

[15] S.M. et al., "Wisconsin wind tunnel ii: A fast and portable parallel architecture simulator," Workshop on performance Analysis and Its Impact on Design, June 1997.

[16] V. Pai, P. Ranganathan, and S.Adve, "Rsim : An execution-driven simulator for ilp-based shared memory multiprocessor and uniprocessors," Third Workshop on Computer Architecture Education, Feb 1997.

[17] N. Honarmand, H.Sohofi, M. Abbaspour, and Z.Navabi, " Processor description in APDL for design space exploration of embedded processors," Proc. EWDTS, 2007.

[18] G.H. et al . ,"ISDL : An Instruction set description language for retargetability," In proc Design Automation Conference , pp.299-302,,1997.

[19] M. Reshadi, P. Mishra, N. Bansal, N. Dutt. "Rexsim : A Retargetable framework for instruction-set architecture simulation" CECS Technical Report #03-05 ,Feb,2003

[20] M. Reshadi and N.Dutt, "Generic pipedlined processor modelling and high performance cycle-accurate simulator generation," Vol.2, pp. 786-791, 2005.

[21] Y. Subhash Chandra. Retargetable functional simulator – M.Tech Thesis June 1999.

[22] F. Angiolini,;Jianjiang Ceng; Rainer Leuper ;Cesare Ferri;Luca Benini; "An Integrated Open Framework for Heterogeneous MPSoc Design Space Exploration",page3 , Date06,2006 EDAA.

[23] M. Loghi; F. Angioni; D. Bertozzi; L. Benini. "Analyzing on-chip communication in a MPSoC environment" In proceeding of the 2004, Design, Automation and test in Europe Conference (DATE'04), IEEE, 2004.