ISSN No. 0976-5697

Volume 2, No. 3, May-June 2011

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

An Efficient Retargetable Simulator for ASIP Design Space Exploration

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

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

Abstract: The design of modern embedded systems requires automated modeling tools for faster design and for the study of various design tradeoffs. Such tools put together constitute an integrated environment where the designer can write the high level design specifications in a language and use these tools for automatic generation of system specific tools.

The major contribution of this paper lies in design and development of retargetable simulator and validation of the simulator. Proposed simulator measures cycle count for application executed on processor. Methodology for the Simulator is also discussed. The Operational aspect of retargetable simulator follows the simple and elegant steps and is easy to configure and understand. Optimized source code is generated by retargetable compiler. This optimized code is given as input to the Retargetable simulator. Along with the optimized code, the processor descriptions are required to enter and simulator is executed to get the desired result in the form of Cycle count.

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

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. In addition, low cost and superior performance increases the likelihood of broad consumer acceptance of new electronic products. Size constraints limit the amount of functionality that can be incorporated into product design. Finally low power consumption is necessary for portable electronic equipment that is battery operated.

An ASIP is a processor that is designed to efficiently execute the software for a specific application. Regardless of whether a newly designed ASIP or a preexisting processor core is used, the selected processor should be well suited for the given 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]

(a) Application Analysis: Application is normally written in High level language. Proper analysis of this application under consideration is done and the output of the information is stored in some suitable intermediate format. 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.

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

(c) Instruction Set Generation: Till this step we have identified application requirements and the suitable architecture. Based on this input instruction sets are generated in terms of required micro operation. This instruction set is used during the further steps for code synthesis and hardware synthesis.

(d) 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.

(e) Hardware Synthesis: In this step the hardware is generated using the ASIP architectural template and instruction set architecture using standard tools

B. Architecture Design Space Exploration

System on Chip designs has various goals and objectives. Design space consists of a set of parameters. The main focus of designers lies on minimal cost and maximal performance, low power, high reliability etc. 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 Retargetable Simulators approaches

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. REALTED 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. CHALLENGES IN ASIP DESIGN

The development of a processor is a complex task, involving several development phases, multiple design teams and different development languages. The key phase in processor design is architecture specifications since it serves as the basis for all remaining design phases. Although Hardware Description Languages (HDLs) are designed for architecture implementation, in a traditional design flow, these languages are also often used for the initial specification of the processor. In this design phase tasks, such as hardware/software partitioning, instruction-set and micro-architecture definition is performed. Based on the architecture specification, both the hardware implementation and development of software is triggered. Both tasks are basically independent and therefore performed by different experts and design methodologies.
Hardware designers use HDLs such as VHDL or Verilog, while software designers mostly utilize the C/C++ programming language. In addition, the target processor needs to be integrated into the SoC and the application software needs to be implemented. Communication between the design teams is obviously difficult because of the heterogeneous methodologies and languages.
Considering the traditional processor design flow, the strong dependencies between the design phases imply a unidirectional design flow and prevent even minor optimizations. Due to the different development languages, changes to the architecture are difficult to communicate and inconsistencies are very likely to appear.
The Complexity of processor design even increases in ASIP design, since optimizations targeted to particular applications are mandatory. Mapping an architecture to a given application means moving through a design space by axes such as flexibility, power consumption, clock speed, area and more.
Every design decision in one dimension constraints other decisions, for example

Architectural features	Vs	design time,
Design time	Vs	physical characteristics,
Physical characteristics	Vs	flexibility
Flexibility	Vs	verification effort
Verification effort	Vs	architectural features

It is obviously not possible to move through this design space by applying the traditional processor design methodology. A unified design methodology is required, which provides a common basis for all design phases. It must be suitable for all design engineers involved.

IV. 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 beween 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. Figure 4 shows the view of integrated environment. PowerPC 603 processor is specified in Sim-nML. 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 modeled 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.

V. OVERALL APPROACH

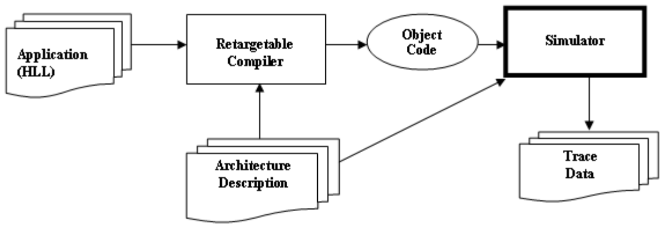


Figure1: Simulator based code generation

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. 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.

IV. METHODOLOGY

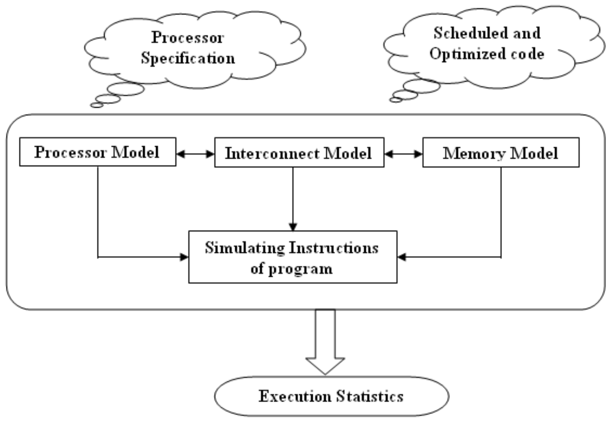


Figure 2 : Overall Methodology adopted

Our architecture model consists of a number of architecture simulation components that simulate the three main parts of the system: Processing elements, memory and interconnection medium. Figure 2 shows the overall methodology adopted for our simulator

A. Processor Model

This method assumes that each instruction completes in a fixed number of cycle. In architecture terms, we can say that our processor has fast private memory where code and local data can be stored. Hence each processing element is equipped with large instruction and local data cache that guarantee a very high hit ratio.

Architectural Parameters of the processor component are:

- Local instruction costs in cycle.
- Number of available interrupts types.
- Context switch latency (cycles to save and restore processor state).
- Interrupt Latency (Cycles needed to save state and branch to interrupt handler).

B. Memory Model

We are considering two-level memory. The levels are main memory and cache(s). We are assuming Shared memory as Main Memory, though we are considering logical partitions with in that memory. Such partitions may be code segment, data segment etc.
Cache memory is assumed to be local to each processing element. However any change in this model can be easily incorporated in our model as it can be specified in input description and that is taken care by our simulator.
Architectural parameters of the memory components are:

- Number of memory modules.
- Memory module size.
- Cache line size.
- Cache set size.
- Number of sets per cache.
- Cache access latency.
- Memory module access latency.

C. Interconnection Model

In a real-time system architecture, the notion of a bus component play an important role as it forms the backbone of communication among all the devices of the system. For this purpose, we need a precise specification of buses for applications that will run on top of them.
Our simulator supports the interconnection of bus. All processing elements and all memory modules are connected through a common bus. Uniform shared memory access is assumed, that is, access of any memory module from any processor takes the same amount of time (ignoring delays due to bus contention).
The simplest interconnection strategy is to use a single bus which is being shared by every other component for communication. Though this strategy is easy to implement, as the number of processor go up, the bus becomes the bottleneck. All the components connected to this bus should tune their interfaces to use the bus protocol. Apart from this, designers have to implement some arbitration mechanism to resolve the conflicts.

VIII.DEVELOPMENT OF RETARGETABLE SIMULATOR

Electronic devices built nowadays are often built with a single IC composed of multitude of hardware blocks that implement the device functionality. In most cases such circuit contains one or more processors that enable to implement a part of the circuit functionality as software that runs on that processor rather than as a specific hardware component. Such IC is commonly referred to as a system-on-a-chip (SoC).

The main CPU features are:

- 101 instructions with possible addressing modes
- CPU with independent stack pointer registers
- Eight 32-bit data, eight 32-bit address and 32-bit status registers
- 16-bit external memory interface

Main assumptions for the ISS were:

- Developed in pure Visual basic 6.0 language for high performance.
- Crystal Report is used as a reporting tool to display the different status.
- MS Access is used to Store the different schedules and optimized code.
- Single-instruction accuracy, without taking internal architecture under consideration.
- Fully static design with the support of loop / wait statements.
- Usage of native VB types to gain high simulation speed.
- Communication interfaces separated from functionality.

The main part which contains implementation of main processor's logic (ALU, instruction fetch, decoding and execution routines) together with fields corresponding to the internal resources (all registers). Sub-module features:

- Fetching and decoding instructions
- Instruction processing routines
- Handling interrupts and exceptions
- Register implementation and registers read/write access
- Instruction counter

Simulators are critical components of the exploration toolkit for the system designer. Simulators can be used to perform a variety of tasks such as verifying the functionality and / or timing behavior of the system, and generates quantitative measurement, for e.g. Cycle count etc. As per our design Methodology and hypothetical assumption of the Architecture we have taken MIPS Architecture as a base to develop our Retargetable simulator. We have given a Nomenclature to our Simulator as SIM-A {Simulators for Architectures}.
We will be using Expression Language for Architecture Description. We have developed the GUI Interface for the same. We have also provided the GUI for easy evaluation and analysis.

A. SIM-A- Basics

SIM-A is a 32-bit datapath, every instruction is 32 bits wide, and data comes in “words” which are also 32 bits wide. Memory in SIM-A, however, is addressed in bytes.
SIM-A is load-store architecture that is, the only instructions that access memory are **LW** and **SW**.

B. Memory Organization

We are considering Two-level memory. The levels are main memory and cache(s). We are assuming main memory as Shared Memory. Cache memory is assumed to be local to each processing element. We are also considering logical partitions with in that memory. Such Partition may be code segment, extra segment etc

Data begins at virtual address 0x10000000 and grows in the direction of increasing virtual addresses (this data is called dynamic data because the machine doesn't know how much of it will be used at runtime). In SIM-A, there is also a concept of stack – that is, data that starts just below virtual address 0x80000000 and grows in the direction of increasing virtual addresses.

C. SIM-A Register Set

Registers are a small set of fast memory that the datapath has available at its disposal for most immediate operations. All registers are 32-bit wide. SIM-A contains thirty two user registers (that is, registers that the user can access/use in the assembly program) and four special-purpose registers that are hidden from the user.

D. SIM-A Instruction Set

This section describes in detail all the SIM-A instructions.

Rs and Rt are source registers – the datapath should fetch their values whenever they are used. Source registers are usually treated as twos-complement signed 32-bit numbers. In some special cases they are treated as unsigned numbers (the note that follows explains such circumstances)

Rd is the destination register – the datapath will write the result to that register number.

Immediate values may either be treated as signed or unsigned values, and may either be zero-extended (in which case the padding bits are all zero), or sign extended (in which case the padding bits are all equal to the most significant bit of the immediate value).

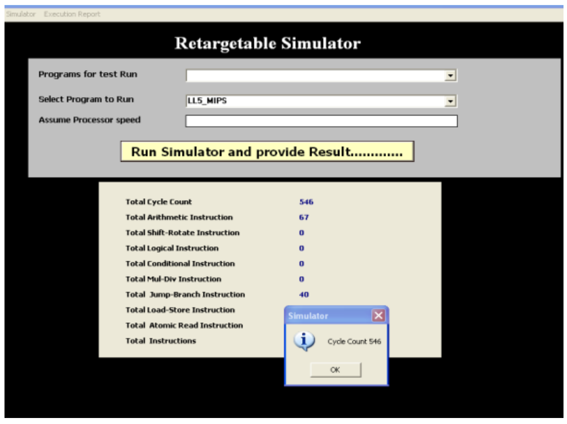
SOC designs have various design goals. These goals include minimal cost, maximal performance, low power, high reliability, etc. Design Space Exploration allows the SOC designer to make trade-offs between these goals and arrive at an "optimal" design.

SOC designer would like to explore changes to the architecture or the instruction-set of the processor-memory system. Common examples of such changes include, but not limited to:

- *Changing the pipeline structure.* e.g., increasing (or decreasing) the number of stages to increase (or decrease) the clock frequency, adding forwarding paths to reduce pipeline stalls.
- *Changing the data path structure.* e.g., changing slow units to fast units in order to increase performance, changing connectivity between units and storage elements (like register files) in order to decrease power consumption.
- *Increasing parallelism.* e.g Adding more functional units that can execute in parallel in order to increase performance.
- *Changing the instruction-set.* e.g Adding new operations which can be exploited by particular applications.
- *Changing the memory component.* e.g Changing the size of register file, changing the associativity of the cache, etc.
- *Changing the memory hierarchy.* e.g Adding a cache between the processor and off-chip memory, changing the on-chip memory hierarchy etc.

E. SIM-A Look and Feel

This is the first and main form which helps us to calculate the cycle count of any program as shown in Figure3.First section allows us to select the different programs that we are required to simulate.



Programs for test Run:
Select Program to Run: LL5_MIPS
Assume Processor speed:

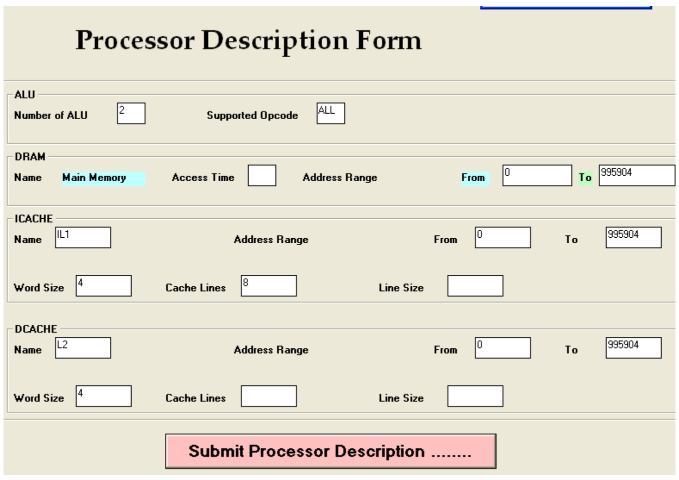
```
Total Cycle Count                 546
Total Arithmetic Instruction      67
Total Shift-Rotate Instruction    0
Total Logical Instruction         0
Total Conditional Instruction     0
Total Mul-Div Instruction         0
Total Jump-Branch Instruction     40
Total Load-Store Instruction
Total Atomic Read Instruction
Total Instructions
```

Simulator: Cycle Count 546

Figure 3: GUI for SIM-A Simulator

This is the interface through which user will enter the processor description and will mimic the behaviour of the processor. If you click on the option "Select Program to Run" , it contains all the list of programs. Just select the program that we need to simulate and Click the button "Run Simulator and Provide Result ...." Others buttons are not used right now. Second Section provides the output of the program. It contains information like total Arithmetic instruction, Shift rotate, Logical, Jump Branch etc. It also gives the pop up when the program finishes by providing the cycle count.

User will first enter the processor description details as shown in figure 4. Submit the form to update the processor description file. Then it browses to the GUI form where he can select the programs that he/She needs to simulate.


Processor Description Form

ALU

Number of ALU	2	Supported Opcode	ALL

DRAM

Name	Main Memory	Access Time		Address Range	From 0	To	995904

ICACHE

Name	IL1	Address Range	From 0	To	995904
Word Size	4	Cache Lines	8	Line Size	

DCACHE

Name	L2	Address Range	From 0	To	995904
Word Size	4	Cache Lines		Line Size	

Figure 4: GUI for Processor Description

The above is the brief description of the SIM-A Simulator that has been developed in our Embedded System Lab.

[8] PERFORMANCE ESTIMATES AND VALIDATION OF SIMULATOR

The Framework is based on MIPS 4K like processor architecture. The architecture contains five pipeline stages – fetch, decode, operand read, execute and writeback. There are five parallel issue paths corresponding to two ALU Units, one for floating point unit, a branch unit and a Load/store unit. The memory hierarchy consists of two L1 data caches for instructions and data, a unified L2 cache and a DRAM main memory. There is a 32-bit wide general purpose register file and a 32-bit wide floating point register file, each containing 32 registers.

Table 1: Benchmark Programs along with Description

No	Name	Description
1	SIM-A-BENCH#1(SIM1)	Excerpt from a hydrodynamic code
2	SIM-A-BENCH#2(SIM2)	Standard Inner product function of Linear Algebra
3	SIM-A-BENCH#3(SIM3)	Excerpt from a Tridiagonal Elimination routine
4	SIM-A-BENCH#4(SIM4)	First Sum
5	SIM-A-BENCH#5(SIM5)	First Difference

Table 1 lists all the benchmarks programs that have been used to validate the simulators. After running this benchmark program on the SIMPRESS as well as SIM-A Simulator, following results are obtained.

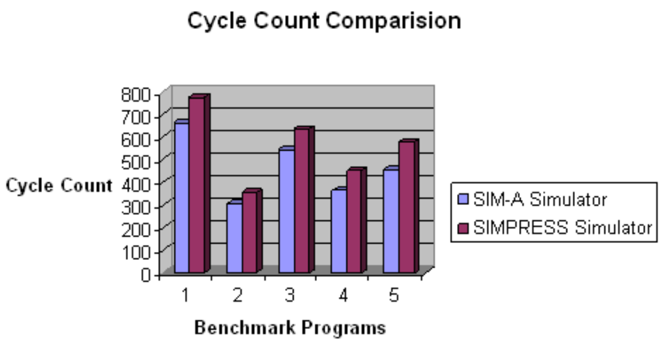


Figure 5: Comparative analysis of SIM-A and SIMPRESS Simulator of Cycle Count

Figure 5 show the graphical analysis of the SIM-A and SIMPRESS Simulator.
At 1% level of significance, the critical value of *'t'* for (5+5-2) 8 degree of freedom is 3.36 and calculated value is 0.368329. Since the calculated value of *'t'* is 0.368329 which is less than the critical value, which is 3.36, it falls in the acceptance region.
Hence it may be concluded that both the results are equally acceptable at 1% level of Significance.

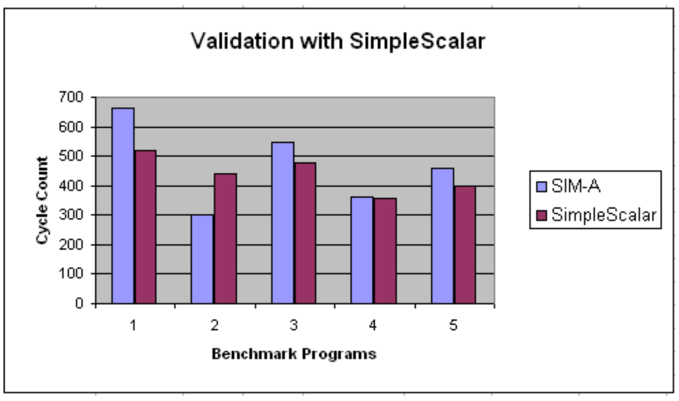


Figure 6: Comparative analysis of SIM-A and SimpleScalar Simulator of Cycle Count

The SimpleScalar tool set is a system software infrastructure used to build modeling applications for program performance analysis, detailed microarchitectural modeling, and hardware-software co-verification. Using the SimpleScalar tools, users can build modeling applications that simulate real programs running on a range of modern processors and systems.
Figure 6 show the graphical analysis of the SIM-A and SimpleScalar Simulator.

IX. CONCLUSION AND FUTURE DIRECTION

In this paper we presented a SIM-A Simulator entirely developed at our Lab that generates the performance estimates for the application under consideration. Processor description is captured in the form of GUI, which allows the user to specify the architecture in visual form. 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 goal of this project is to allow modeling of a wide variety of processors and memory systems. In order to achieve this goal, the simulator generator includes very general mechanisms for capturing processor architectures. But its usage has to be extended for other class of processors.

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.

Although the speed is acceptable, another direction that could be pursued is that of improving the speed of the simulators, which is a very important factor in design space exploration.

X. REFERENCES

[1] Manoj Kumar 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 June 1999.

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

[4] JAIN, N.C. 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] Mehrdad Reshadi, Prabhat Mishra, Nikhil Bansal, Nikhil 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] Fedrico 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.