Journal of Engineering Science and Technology
Vol. 7, No. 5 (2012) 574 - 588
© School of Engineering, Taylor's University

EMBEDDED MICROPROCESSOR PERFORMANCE EVALUATION CASE STUDY OF THE LEON3 PROCESSOR

NABIL LITAYEM[1,2,*], BOCHRA JAAFAR[2], SLIM BEN SAOUD[1]

[1]LECAP, EPT-INSAT, Centre urbain nord, BP 676 Tunis cedex, Tunisia
[2]Computer Science and Information Department, King Salman Bin Abdulaziz University, Wadi Addwaser, Saudia Arabia
*Corresponding Author: nabil.litayem@instructor.net

Abstract

In this paper we propose a performance evaluation methodology based on three complementary benchmarks. This work is motivated by the fact that embedded systems are based on very specific hardware platforms. Measuring the performance of such systems becomes a very important task for any embedded system design process. In a classic case hardware performance is a basic result reported by the hardware manufacturer. The personalization of hardware configuration is one of the fundamental task of FPGA based embedded systems designer. They must measure the hardware performance himself. This paper will focus on hardware performance analysis of FPGA based embedded system using freely available benchmarks to reflect various performance aspects. We used in our study two embedded systems (Mono-processor and Bi-Processor) based on LEON3MMU processor and eCos RTOS.

Keywords: Embedded Systems, Performance, Benchmark.

1. Introduction

The human activity becomes more and more reliant to embedded systems that are actually present in many products like PDA, camera, telephones etc. Designing this kind of systems can take many approaches depending on the used platform. In a classic approach, General Purpose processor (GPP), Application Specific Processor (ASIP) or Application Specific Integrated Circuit (ASIC) can be used as a heart of the embedded system. Each one of precedent solutions has its advantages and weaknesses.

Actually we assist to the Field Programmable Gate Array (FPGA) based embedded systems emergence. This kind of solution can allow rapid embedded

Nomenclatures

CPI	Cycles per instruction
MFLOPS	Million Floating Point Operations Per Second
MIPS	Million Instructions Per Second
MOPS	Millions of Operations Per Second

Abbreviations

AMBA	Advanced Microcontroller Bus Architecture
ASIC	Application-Specific Integrated Circuit
ASIP	Application-Specific Instruction-set Processor
EEMBC	EDN Embedded Microprocessor Benchmark Consortium
FPGA	Field-Programmable Gate Array
GNU	Gnu's Not Unix
GPL	General Public License
GPP	General Purpose Processor
HAL	Hardware Abstraction Layer
HDL	Hardware Description Language
MMU	Memory Management Unit
PDA	Personal Digital Assistant
POSIX	Portable Operating System Interface for Unix
RISC	Reduced Instruction Set Computer
RTEMS	Real-Time Executive for Multiprocessor Systems
RTOS	Real-Time Operating System
SPEC	Standard Performance Evaluation Corporation
VAX	Virtual Address eXtension

systems’ generation [1], easy personalization of hardware configuration using pre-designed Hardware IPs [2], future evolution of embedded system and cost reduction. To design an FPGA based embedded system we have to choose an embedded processor and embedded operating system.

Embedded processor can be Hard-Core (built in silicon level) or Soft-Core (netlist or as HDL source). Hardcore embedded processor has the advantage of computing performance but limit the system in terms of portability and scalability. Soft-Core embedded processor offer less computing possibility if the final platform is an FPGA, but is greatly enhanced in term of configurability, portability, customization and scalability [3, 4].

Embedded operating system can be shared time or real-time, proprietary or open source. The adoption of an embedded operating system depends on the available memory, developers' strategy, real-time requirements, operation fields’ certification etc. On the other hand, the hardware resources limitation for embedded software requires that the developer must have a clear idea about the hardware computing performance. Actually, many studies focus on hardware performance evaluation or estimation of customized architecture [5].

In this paper we present an approach to measure hardware performance of FPGA based embedded system using freely available benchmark solutions. This work is divided into six parts. The first part is a survey about the performance evaluation. The second part presents an overview about the used platform. In the third part we will present the adopted benchmarks. This is followed by the experimental condition and environment preparation presentation. In the fifth part experimental results are presented. The last section summarizes our position and future works.

2. Overview of Performance Evaluation Tools And Techniques

Evaluating performance in computer system [6] will always be a true challenge for designer of this kind of systems due to the constant evolution of such systems, especially for embedded system field where the architecture tend to be more and more complex [7]. This kind of activity is actually related to the performance engineering field which tries to propose tools and methods to quantify non-functional requirements of computer systems. The performance analysis of an embedded system may have various aspects depending on the application for which the embedded system is designed. Several design decisions are a direct result of performance evaluation.

The first evaluation method was based on the number of operations per second. This approach quickly became deprecated and traditional benchmark was developed and adopted to measure special performance aspects. This aspect can be one of the various computer system criteria. Actually we have too many solutions to measure hardware performance. The most part of solutions are based on standard algorithms that are executed and used to report a number which reflects the performance of the hardware speed in a special field.

In generally the MIPS unit in its literal meaning millions of instructions per second is used to measure the processor hardware performance. This unit became insignificant when RISC computer architectures appeared since the performed instruction by one CISC computer cycle requires several RISC instructions. Which lead to the MIPS redefinition as VAX MIPS which is the factor for a given machine relative to the performance of a VAX 11/780. Other redefinitions are later proposed such as Peak MIPS, and EDN MIPS.

Due to the fuzziness of performance unit definition, several benchmarks are proposed that can be executed under various architectures to report their execution speed. The most recognized solutions are Dhrystone which report the performance of the architecture in Dhrystone MIPS, Stanford which computes different algorithms and report the performance of the architecture in every computation field covered by the benchmark, and Paranoia who is able to report the characteristics of the floating point unit.

Nowadays, there are several benchmarks that include the previously presented ones combined with other widely known benchmarks such as Whestone and Linpac. Each one of these benchmarks tries to reflect the most of the hardware performance aspects. Mibench [8] is the most popular implementation of this combination since it can measure the performance of a studied architecture in several application fields.

We can also find other commercial benchmarking solutions more efficient and more specialized like SPEC (Standard Performance Evaluation Corporation) which cover different computing field or EEMBC (Embedded Microprocessor Benchmark Consortium) designed especially for embedded systems.

In this paper we will present a hardware performance analysis of mono-processor and bi-processor embedded systems using three benchmarks, each one cover one side of computing system. Our platform is based on two open source components (LEON3 Processor and eCos RTOS) allowing us to be independent to any FPGA or RTOS vendor.

3. Presentation of the Used Platform

3.1. Overview of LEON3 microprocessor

LEON3 [9] presented in Fig. 1 is a synthesizable VHDL model of a 32-bit processor compliant with the SPARC V8 [10] architecture. The model is highly configurable, and particularly suitable for system-on-a-chip (SOC) designs. The full source code is available under the GNU GPL (General Public License), allowing free and unlimited use for research and education. LEON3 is also available under a low-cost commercial license, allowing it to be used in any commercial application for a fraction of the cost of comparable IP cores. On the other hand Gaisler research offers a fault tolerant version of the LEON3 for a very competitive cost.

The LEON3 processor is distributed as a part of the GRLIB IP library, allowing simple integration into complex SoC designs. GRLIB also includes a configurable LEON multi-processor design, with up to 16 CPU's attached to AHB bus, and a large range of on-chip peripheral blocks. The GRLIB library contains template designs and bitfiles for various FPGA boards from Actel, Altera, Latice and Xilinx.

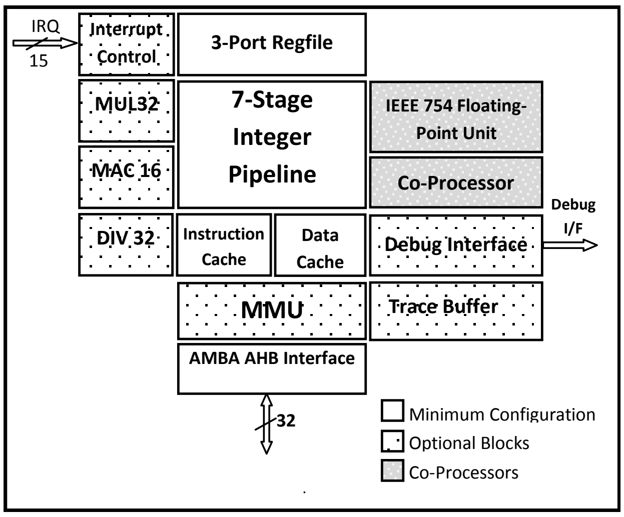


Fig. 1. Overview of LEON3 Architecture [9].

3.2. Overview of eCos RTOS

eCos is an abbreviation of Embedded Configurable Operating System [11]. It is an open source, royalty-free, real-time operating system intended for deeply embedded applications. The highly configurable nature of eCos allows the operating system to be customized to precise application requirements, delivering the best possible run-time performance and an optimized hardware resource footprint. A thriving net community has grown up around the operating system ensuring on-going technical innovation and wide platform support. Figure 2 shows the layered architecture of eCos.

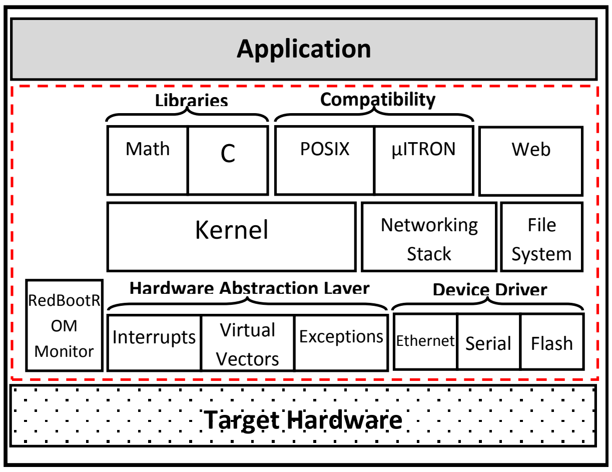


Fig. 2. Overview of eCos Architecture.

The main components in eCos architecture are the HAL (Hardware Abstraction Layer) and eCos Kernel.

The purpose of eCos HAL is to allow the application to be independent of hardware target. It can manipulate the hardware layer using the HAL API. This HAL is also used by others upper OS layer which make porting eCos to a new hardware target a simple task consisting of developing the HAL of the new target. eCos kernel is the core of eCos system, it includes the most part of modern operating system components: scheduling, synchronization, interrupt, exception handling, counters, clocks, alarms, timers, etc. It is written in C++ language allowing application written in this language to interface directly to the kernel resources. The eCos kernel also supports interfacing to standard µITRON and POSIX compatibility layers.

3.3. Combination eCos – LEON3

The choice of these components allows us to be independent from any FPGA constructor or RTOS vendor since eCos is available for a wide range of embedded processors and LEON is ported for XILINX, ALTERA, ACTEL and LATICE FPGA. In the other hand eCos allows a smooth migration to embedded Linux. We can also use OpenRISC [12] as processor. RTEMS [13] or embedded Linux [14] can be adopted as OS.

The performance measure will be presented using the three benchmarks that we will present, and the hardware platform will be simulated using tsim-leon3 for a mono-processor architecture and grsim-leon3 for the bi-processor platform in SMP (synchronous multi-processing) configuration. These two simulation tools are able to represent very closely the LEON3 architecture with many other very important enhanced features for system prototyping.

4. Used Benchmarks

In our work we have chosen to adopt three complementary benchmarks that can mirror a complementary performance side of the studied embedded system. Each one of the obtained results of these benchmarks can be used to evaluate a special performance aspect of the hardware platform.

4.1. Dhrystone

Dhrystone [15] is a synthetic benchmark developed in 1984 by Reinhold P. Weicker in ADA, Pascal and C languages. It is intended to be representative of integer system performances. Dhrystone is constituted from 12 procedures included in one measuring loop with 94 statements. The Dhrystone grew to become representative of general processor (CPU) performance until it was outdated by the CPU89 benchmark suite from the Standard Performance Evaluation Corporation [16], today known as the "SPECint" suite.

4.2. Stanford

The Stanford Benchmark Suite [17] is a small benchmark suite that was assembled by John Hennessy and Peter Nye around the same time period of the MIPS R3000 processors. The benchmark suite contains ten applications, eight integer benchmarks and two floating-point benchmarks. The original suite measured the execution time in milliseconds for each benchmark in the suite. The Stanford Small Benchmark Suite includes the following programs:

- **Perm:** A tightly recursive permutation program.
- **Towers:** the canonical Towers of Hanoi problem.
- **Queens:** The eight Queens Chess problem solved 50 times.
- **Integer MM:** Two 2-D integer matrices multiplied together.
- **FP MM:** Two 2-D floating-point matrices multiplied together.
- **Puzzle:** a compute bound program.
- **Quicksort:** An array sorted using the quicksort algorithm.
- **Bubblesort:** An array sorted using the bubblesort algorithm.
- **Treesort:** An array sorted using the Treesort algorithm.
- **FFT:** A floating-point Fast Fourier Transform program.

This kind of benchmark is very interesting in terms of exploration of various architecture behaviours.

4.3. Paranoia

The Paranoia benchmark [18] is designed by William Kahan as a C programable to characterize floating-point behaviour of computer system.

Paranoia does the following test:

- Small integer operations.
- Search for radix and precision.
- Check if rounding is done correctly.
- Check for sticky bit.

- Test if $\sqrt{X^2} = X$ for a number of integers.
- If it will pass monotonicity.
- If it is correctly rounded or chopped.
- Testing power Z^i, for small Integers Z and i.
- Searching for underflow threshold and smallest positive number.
- Testing power Z^Q at four nearly extreme values.
- Searching for overflow threshold and saturation.
- Tries to compute 1/0 and 00.

5. Experimental Set-Up

Gaisler research offer all the required tools to begin developing application using Leon processor combined with a wide range of supported RTOS such eCos. In this section, we present the eCos configuration phases, the benchmarks compilation and their respective execution environment.

5.1. eCos configuration

The eCos RTOS installation and configuration can be divided into four steps. In our case we have chosen to use Windows host but similar approach can be followed with Linux.

5.1.1. Environment installation

Since eCos and their associated tools require Linux-like environment, we must begin by installing Linux emulation environment. For this purpose there are two candidates which are “Cygwin” and “MingW”. We chose to use “MingW” due to its lightweight. It must be noted that “make”, “mpfr”, “sharutils”, “tcltk”, “wget”, “automake” and “gcc” packages must be installed with the emulation environment.

5.1.2. Cross compiler installation

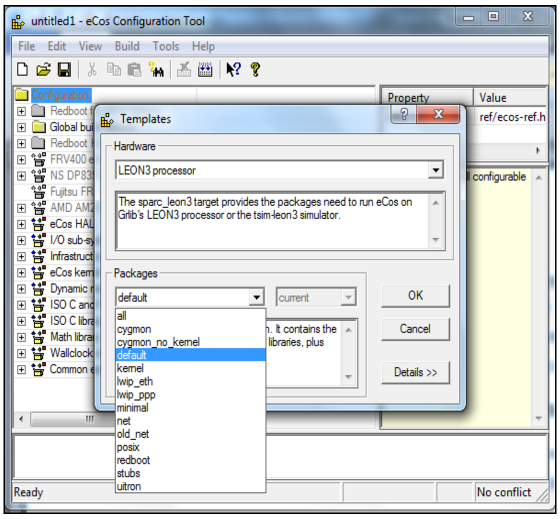
In order to produce Leon3 executable we must have a Leon cross compiler installed in our host. The recommended cross-compiler for eCos is sparc-elf-gcc. This one is available in the Gaisler web site [19] and must be downloaded, decompressed in the `“/opt”` directory and installed using `“export PATH=/opt/sparc-elf-4.4.2-mingw/bin:$PATH”` command.

5.1.3. Source code and configuration tool

In this stage we must install the eCos source code and their configuration tool. These resources are also available in the Gaisler web site [19]. Source code must be downloaded and decompressed in a chosen directory that can be used in the configuration phase. The configuration tool is available in different versions that differ according to their software dependencies. For our case we used the native Windows version that does not need additional library.

5.1.4. eCos configuration

eCos is one of the most architecture free RTOS. The choice of a specific target is done using the configuration tool GUI that offer a wide range of hardware platform and software configurations. In this step we must run the eCos configuration tool and select the target platform "`LEON3 processor`" as shown in Fig. 3. In the same configuration window we can chose a predefined setting that can be customised later by selecting specific networking stack, debugging interface or any other specific software component. In our case default packages can be enough.


untitled1 - eCos Configuration Tool

Templates

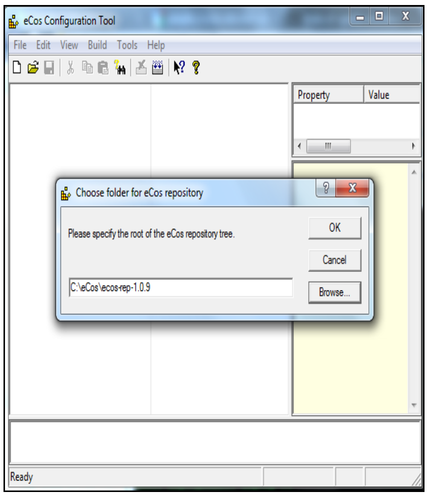
Hardware: LEON3 processor

The sparc_leon3 target provides the packages need to run eCos on Gaisler's LEON3 processor or the tsim-leon3 simulator.

Packages: default

Fig. 3. Hardware Platform and Used Package.

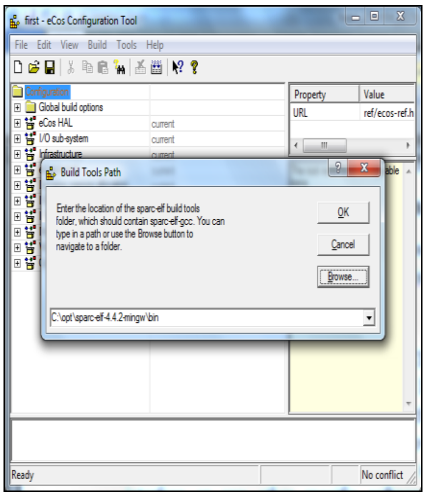
The build tool and Cygwin binary tools directories must be selected in the configuration tool as shown in Figs. 4 and 5. In the case of using MingW, these ones are located at "`C:\MinGW\msys\1.0\bin`".


eCos Configuration Tool

Choose folder for eCos repository

Please specify the root of the eCos repository tree.

C:\eCos\ecos-rep-1.0.9


first - eCos Configuration Tool

Build Tools Path

Enter the location of the sparc-elf build tools folder, which should contain sparc-elf-gcc. You can type in a path or use the Browse button to navigate to a folder.

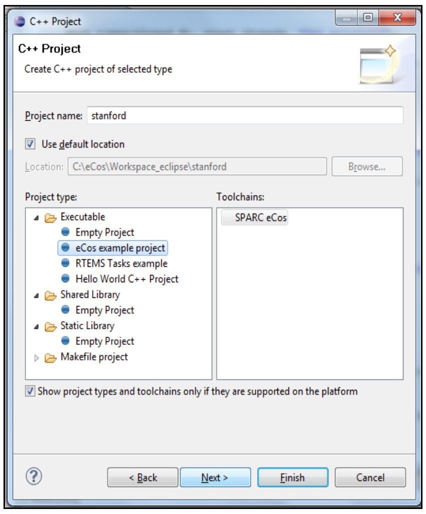
C:\opt\sparc-elf-4.4.2-mingw\bin

Fig. 4. eCos Repository. **Fig. 5. eCos Build Tools.**

Finally we must save the configuration file and starting the building process using build item in the configuration tool GUI.

5.2. Benchmarks compilation

One of the most valuable tools offered by Gaisler research is the eclipse based IDE. This one offers various wizards for project creation that generates project template for eCos shared library, static library or executable. We used this wizard as showed in Figs. 6 and 7 to generate eCos project template.



C++ Project

C++ Project

Create C++ project of selected type

Project name: stanford

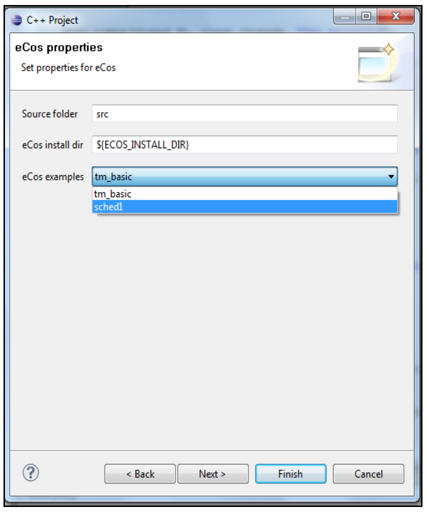
☑ Use default location

Location: C:\eCos\Workspace_eclipse\stanford

Project type:
- Executable
  - Empty Project
  - eCos example project
  - RTEMS Tasks example
  - Hello World C++ Project
- Shared Library
  - Empty Project
- Static Library
  - Empty Project
- Makefile project

Toolchains: SPARC eCos

☑ Show project types and toolchains only if they are supported on the platform



C++ Project

eCos properties

Set properties for eCos

Source folder: src

eCos install dir: ${ECOS_INSTALL_DIR}

eCos examples: tm_basic

tm_basic

sched1

Fig. 6. eCos Project Generation. **Fig. 7. eCos Example Code Generation.**

The eCos "`ECOS_INSTALL_DIR`" environment variable must then be configured in the project setting according the previously chosen eCos install directory.

To build adequately the three previously presented benchmarks, we created three projects in which we substitute the generated code with the benchmark source code. In the build configuration panel math library and eCos must be selected to be used during the link process.

5.3. Benchmarks execution

Gaisler research offers two simulation platforms. The first one is Tsim-leon3 designed to simulate mono-processor Leon3 based architecture, the second one is grsim-leon3 who is designed to simulate bi-processor Leon3 based architecture. These two simulators have nearly the same usage syntax. We can then load and execute the appropriate benchmark in their respective architecture.

6. Performance Measurement and Analysis

In the following section we present the performance evaluation and their interpretation according the flowchart showed in Fig. 8.

After preparing the environment in terms of configuring, building eCos for LEON3MMU architecture and testing some applications examples running under eCos and executing their benchmarks from SDRAM. We build the three benchmarks for our two hardware configurations described in Table 1.

Table 1. Hardware Configuration.

	Mono-processor Configuration	**Bi-processor Configuration**
SDRAM	16 Mbyte in 1 bank	16 Mbyte 1 bank
ROM	2048 Kbyte	2048 Kbyte
Instruction Cache	1*4 Kbytes, 16 bytes/line	1*4 Kbytes, 16 bytes/line
Data cache	1*4 Kbytes, 16 bytes/line	1*4 Kbytes, 16 bytes/line

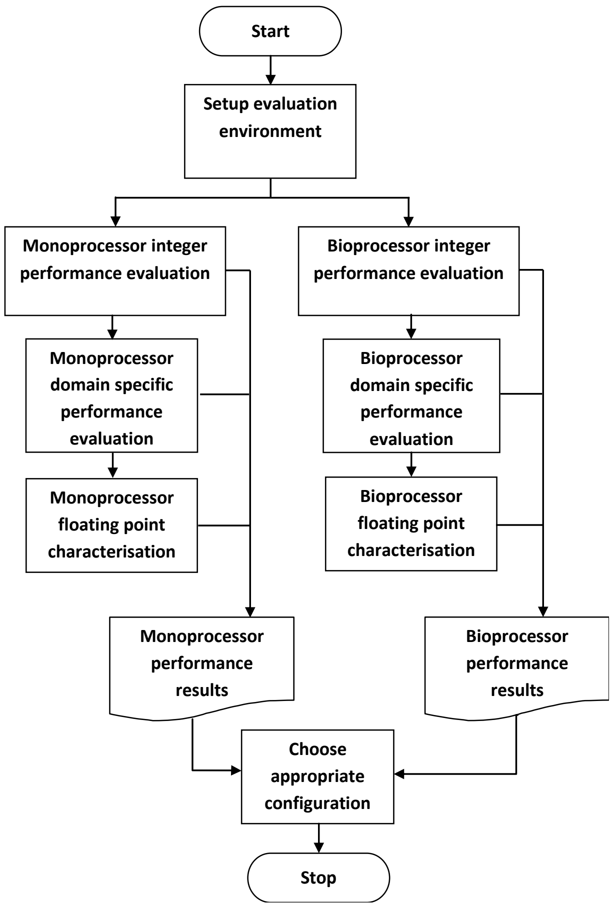

Start
Setup evaluation environment
Monoprocessor integer performance evaluation
Bioprocessor integer performance evaluation
Monoprocessor domain specific performance evaluation
Bioprocessor domain specific performance evaluation
Monoprocessor floating point characterisation
Bioprocessor floating point characterisation
Monoprocessor performance results
Bioprocessor performance results
Choose appropriate configuration
Stop

Fig. 8. Performance Evaluation Flowchart.

6.1. Obtained results using Dhrystone benchmark

After executing Dhrystone benchmark under our platforms simulators we have the reported values in Fig. 9 and summarized in Table 2. These results show the performance of studied architecture in term of Dhrystone MIPS. The gain in performance is about 33% with the bi-processor configuration.

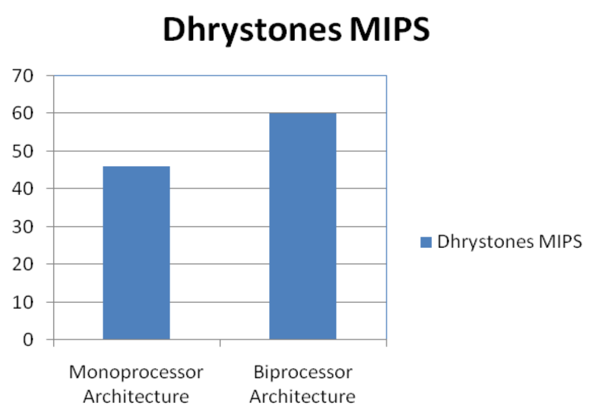


Fig. 9. Performance in Dhrystones MIPS of the Two Architectures.

Table 2. Results Obtained with the Two Platform Simulators for Dhrystone Benchmark.

	Mono-processor Architecture	Bi-processor Architecture
Cycles	247650694	189292275
Instructions	156822447	156860966
Overall CPI	1.58	1.21
CPU performance (50.0 MHz)	31.66 MOPS (31.66 MIPS, 0.00 MFLOPS)	41.43 MOPS (41.43 MIPS, 0.00 MFLOPS)

6.2. Obtained results using Stanford benchmark

After executing Stanford in our platform simulator we collected the performance report plotted in Figs. 10 and 11 and summarised in Table 3.

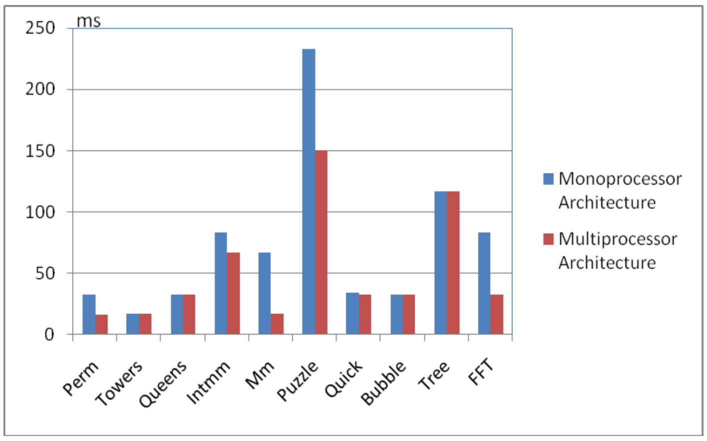


Fig. 10. Execution Time in Millisecond of the Ten Algorithms Included in Stanford Benchmark.

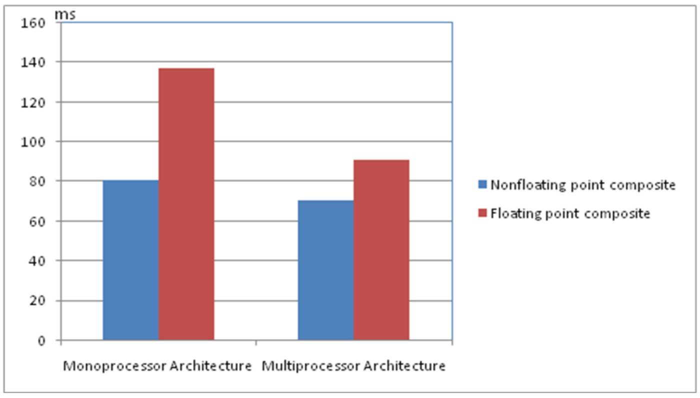


Fig. 11. Composite Performance of the Two Architectures for Nonfloating and Floating Point Applications.

These results show a little enhancement made by the multiprocessor architecture, especially for complex algorithms such as Puzzle and Intmm. These results can be explained by the fact that Stanford benchmark was not written to exploit parallel architectures.

The composite performance report shows that the Nonfloating point enhancement by multiprocessor adoption is about 14%. On the other hand the enhancement made by multiprocessor adoption fore floating point application is about 55%. These results can be justified by the complexity of floating point algorithms that can exploit the multiprocessor architecture.

Table 3. Results Obtained with the Two Platform Simulators for Stanford Benchmark.

	Mono-processor Architecture	**Bi-Processor Architecture**
Cycles	36873110	26063953
Instructions	22181699	22218991
Overall CPI	1.66	1.17
CPU performance (50.0 MHz)	30.08 MOPS (29.49 MIPS, 0.59 MFLOPS)	42.62 MOPS (41.79 MIPS, 0.83 MFLOPS)
Cache hit rate	96.5% (99.8 / 75.1)	93.5% (99.8 / 60.2)
Simulated time	737.46 ms	521.28 ms

After examining the simulator report we conclude that we have a 12% performance gain for integer operation, and 32% for floating point operations while using bi-processor configuration.

This gain of performance is not equally distributed between the ten algorithms included in Stanford benchmark. The adoption of one of these two architectures will depend on the final application.

6.3. Results obtained using paranoia benchmark

After executing paranoia benchmark under the two platform simulators we conclude that the FPU operation is correctly executed for the two architectures. But the benchmark reports that we have:

```
Addition/Subtraction neither rounds nor chops.
Sticky bit used incorrectly or not at all.
FLAW:  lack(s) of guard digits or failure(s) to correctly round
or chop (noted above) count as one flaw in the final tally below.
```

This type of failure is not so dangerous for the system functionality but can cause some precision loss. The source of this failure is certainly caused by the code generation in the soft-float parameters of GCC compiler.

6.4. Analysis of the obtained results

Performance results show a performance improvement from 0 to 55% using multicore architectures. This is especially due to the non-optimization of both benchmarks and compiler for multicore architectures. Better improvement is shown using complex algorithm such as *puzzle* or *Mm.* This fact must be considered during the algorithm development to adequately exploit the multicore aspects.

7. Conclusion And Perspectives

The reported benchmarks results cover three computing system performance fields. Dhrystone is used to compare integer unit performance. Stanford is able to compare different standard algorithm performance execution both in integer and floating point computing. Paranoia is able to characterize floating point operations.

We observed that the performance gain for multicore architecture is not so considerable since the used benchmarks were not designed to exploit the multicore architecture. The same approach can be used to compare performance of other architectures, but this kind of work can be done carefully since a few studies report some fragility's of SPEC CPU95 and CPU2000 [20] which is a superset of our used benchmarks.

In our work we focused on hardware execution speed evaluation in the embedded system design flow. Modern embedded system has other performance aspects and factors that must be considered such as multiprocessor impact and RTOS overhead [21]. This work can be extended by multithreading these benchmark or use multiprocessor benchmark to compare the studied architectures [22].

In our future work we will focus on multiprocessor and operating system overhead. Multiprocessor performance evaluation can be done using a specific multicore and parallel benchmark such as SPLASH2 and NPB or by adapting standard benchmarks [22, 23]. In the other hand real-time overhead and RTOS comparison can be evaluated using standard benchmarks that evaluate the overall performance after the adoption of a specific operating system or by adopting dedicated benchmarks such as thread-metric [24].

For this reason we attempt to extend our study by measuring the performance of eCos and comparing it with other RTOS such as RTEMS, uC/OS and VxWorks using the same platform.

Acknowledgment

I would like to thank the National Institute of Applied Science and Technologies (INSAT) for the continued and positive support for scientific researches, and also thank the anonymous reviewers for their valuable comments.

References

1. Peddersen, J.; Shee, S.L.; Janapsatya, A.; and Parameswaran, S. (2005). Rapid embedded HW/SW system generation. *Proceedings of the* 18*th International Conference on VLSI Design*, 111-116.
2. Sheldon, D.; Kumar, R.; Vahid, F.; Tullsen, D.; and Lysecky, R. (2006). Conjoining soft-core FPGA processors. *IEEE/ACM International Conference on Computer-Aided Design, 2006. ICCAD* '06, 694-701.
3. L'Hours, L. (2005). Generating efficient custom FPGA soft-cores for control-dominated applications. *Proceedings of the* 16th *International Conference on Application-Specific Systems, Architecture and Processors* (*IEEE ASAP'05*), 127-133.
4. Huerta, P.; Castillo, J.; Jgnacio Martinez, J.I.; and Pedraza, C. (2007). Exploring FPGA capabilities for building symmetric multiprocessor systems. *SPL* '07. 2007, 3rd *Southern Conference on Programmable Logic*, 113-118.
5. Groβschädl, J.; Tillich, S.; and Szekely, A. (2007). Performance evaluation of instruction set extensions for long integer modular arithmetic on a SPARC V8 processor. *DSD 2007*. 10th *Euromicro Conference on Digital System Design Architectures, Methods and Tools*, 690-689.
6. John, K.L.; and Eeckhout, L. (2006). Performance *evaluation and benchmarking*. CRC Press.
7. Wolf, W. (2007). *High-performance embedded computing. Architecture, Applications, and Methodologies*. Elsevier Inc.
8. Guthaus, M.R.; Ringenberg, J.S.; Ernst, D.; Austin, T.M.; Mudge, T.; and Brown, R.B. (2001). MiBench: A free, commercially representative embedded benchmark suite. *WWC*-4. 2001 *IEEE International Workshop on Workload Characterization,* 3-14.
9. Ahmed, S.Z.; Eydoux, J.; Rougé, L.; Cuelle, J.-B.; Sassatelli, G.; and Torres, L. (2009). Exploration of power reduction and performance enhancement in LEON3 processor with ESL reprogrammable eFPGA in processor pipeline and as a co-processor. *Design, Automation & Test in Europe Conference & Exhibition*, 184-189.
10. Gaisler, J. (2002). A portable and fault-tolerant microprocessor based on the SPARC V8 architecture. *Proceedings of International Conference on Dependable Systems and Networks*, 409-415.
11. Massa, A.J. (2003). *Embedded software development with eCos*. Prentice Hall.
12. OpenCores. www.opencores.org/projects.cgi/web/or1k/overview.

13. RTEMS Real Time Operating System. www.rtems.org.
14. Yaghmour, K.; Masters, J.; Ben-Yossef, G.; and Gerum. P. (2003). Building embedded Linux systems. (2nd Ed.) O'Reilly Media.
15. Stitt, G.; Lysecky, R.; and Vahid, F. (2003). Dynamic hardware/software partitioning: A first approach. *Proceedings of the* 40*th Design Automation Conference* (*DAC*), 250-255.
16. Henning. J.L. (2000). SPEC CPU2000: measuring CPU performance in the new millennium. *Computer*, 33(7), 28-35.
17. Assmann, U. (1996). How to uniformly specify program analysis and transformations with graph rewrite systems. *In Proceedings of the CC'96.* 6*th International Conference on Compiler Construction*, 121-135.
18. Hillesland, K.E.; and Lastra, A. (2004). GPU floating point paranoia. *In: Proceedings of* 1*st ACM Workshop General-Purpose Computing on Graphics Processors* (*GP2'*04).
19. Cross-Compiler for eCos: http://www.gaisler.com/doc/libio/bcc.html#Installation.
20. Vandierendonck, H.; and De Bosschere, K. (2004). Eccentric and fragile benchmarks. *International Symposium on ISPASS*, *Performance Analysis of Systems*, 2-11.
21. Proctor, F.M.; and Shackleford, W.P. (2001). Real-time operating system timing jitter and its impact on motor control. *Proceedings of the SPIE Conference on Sensors and Controls for Intelligent Manufacturing II*, 4563, 10-16.
22. Joshi, A.M.; Eeckhout, L.; and John, L.K. (2007). Exploring the application behavior space using parameterized synthetic benchmarks. *16th International Conference on Parallel Architecture and Compilation Techniques, PACT* 2007, 412.
23. Jerraya, A.A.; Yoo, S.; Wehn, N.; and Verkest, D. (2003). *Embedded software for SOC*. Springer
24. Fletcher, B.H. (2005). FPGA embedded processors revealing true system performance. *Embedded Systems Conference, San Francisco*, 1-18.