Architectural Performance Analysis of FPGA Synthesized LEON Processors

Corentin Damman, Gregory Edison, Fabrice Guet, Eric Noulard, Luca Santinelli, Jérôme Hugues

Architectural Performance Analysis of FPGA Synthesized LEON Processors

Corentin Damman*, Gregory Edison*, Fabrice Guet†, Eric Noulard†, Luca Santinelli†, and Jerome Hugues*
* Institute for Space and Aeronautics Engineering (ISAE), Toulouse, [name.surname]@isae.fr
† ONERA (The French Aerospace Lab), Toulouse, [name.surname]@onera.fr

***Abstract*—Modern embedded processors have gone through multiple internal optimization to speed-up the average execution time e.g., caches, pipelines, branch prediction. Besides, internal communication mechanisms and shared resources like caches or buses have a significant impact on Worst-Case Execution Times (WCETs). Having an accurate estimate of a WCET is now a challenge. Probabilistic approaches provide a viable alternative to single WCET estimation. They consider WCET as a probabilistic distribution associated to uncertainty or risk.**

In this paper, we present synthetic benchmarks and associated analysis for several LEON3 configurations on FPGA targets. Benchmarking exposes key parameters to execution time variability allowing for accurate probabilistic modeling of system dynamics. We analyze the impact of architecture-level configurations on average and worst-case behaviors.

***Index Terms*—Performance Analysis, Embedded Systems, Benchmarking, Processor Synthesis, FPGA, LEON, Probabilistic Worst-Case Execution Time**

I. Introduction

Timing constraints contribute defining the correctness of most of embedded systems. For example, the processor that manages the injection of your engine or the electronic flight controls of an aircraft have to provide results within well-defined time windows. Such systems are called real-time systems. In particular, hard real-time systems if timing constraints misses are not tolerated and soft-real-time systems if some timing constraints misses can be tolerated.

Recent improvements and optimization of processors induce huge complexity onto real-time systems. We are now facing multi-core architectures with internal communications, cache memories, interruptions, multi-functionalities, etc. Each of which has to be properly analyzed in order to guarantee both system correctness and predictability. Classical single value Worst-Case Execution Time (WCET), based on modeling internal architectures behavior, may have reached its limit. Indeed, complexity in micro-architecture and/or costs in modeling system interactions, may lead to a large pessimism for the worst-case scenarios.

Probabilistic approaches now emerge as alternatives to single value WCETs. The probabilistic real-time modeling assumes the task WCET as worst-case distribution, the probabilistic WCET (pWCET), able to upper-bound any possible task execution behavior. The pWCETs allow accounting for the probability of occurrence of worst-case conditions which could be vanishingly small [1]. Hence, pWCETs may lead to important reduction of computing capability over-provisioning since they cope better with tasks actual behavior.

Contributions : In this paper we investigate the impacts of CPU architecture elements on tasks execution behaviors.
For that purpose we synthesize LEON3 processors with different architectures on two Xilinx FPGAs. We modify some core configuration parameters like cache policy, the branch prediction and the floating-point unit to determine their impact. We run some benchmarks selected from the Mälardalen WCET project [2] on the synthesized architectures. Finally, we analyze the results with the use of a measurement-based probabilistic timing analysis tool called DIAGXTRM [3], to perform average and worst-case performance analysis with probabilistic models. *We will publish a data pack with all the models for the final version of the paper, at the following URL: https://forge.onera.fr/projects/syntheticbench*

Organization of the paper: In Section II and Section III we present the architectural parameters applied for synthesizing LEON3 processors. Section IV describes the benchmarks and the task execution conditions applied. In Section V we introduce the DIAGXTRM framework applied for computing the pWCETs and for timing performance analysis. In Section VI we present the results obtained in terms of both average performance and worst-case probabilistic models. Section VII is for conclusions and future work.

A. Related Work

Performance analysis of multi-core platforms is a quite established research field, [4–5]. The approaches proposed so far refer to measurements for evaluating the impact of architectural elements on system average performance, [6–7].

The probabilistic timing analysis estimates pWCETs and can be either static-based or measurements-based. Static Probabilistic Timing Analysis (SPTA) [8–9] requires an exact model of the system in order to infer the probability law, thus computing the pWCETs. Measurements-Based Probabilistic Timing Analysis (MBPTA) relies on measurements and on the Extreme Value Theory (EVT) in order to provide pWCET distributions, [1–3–10]. The MBPTA does not require models of the system, but only measurements of its behavior.

II. Microprocessor Design Space

Recent processors architectures rely on multiple building blocks. In the following, we consider those with an impact on the pWCET and present their key parameters.

• *Cache memories* have a direct impact on performance through instruction/data prefetching. If there is no cache, the system is forced to load each instruction from the memory and to do multiple loads and stores for each variable directly. The consequent saturation of the memory buses will have an impact on the instruction latencies.

• *Cache replacement policies* control how cache misses are handled, and how new pages are loaded. Different policies may either speed up or slow down a software.

• *Branch prediction* controls how instructions are prefetched in the pipeline. Such policies usually rely on heuristics.

• *Math co-processor* enables faster mathematical operations compared to software-emulated ones.

For this study, we implement LEON3 processors, which general architecture is presented in Figure 1, [11]. The LEON3 is a 32-bit SPARC V8 processor, [12].

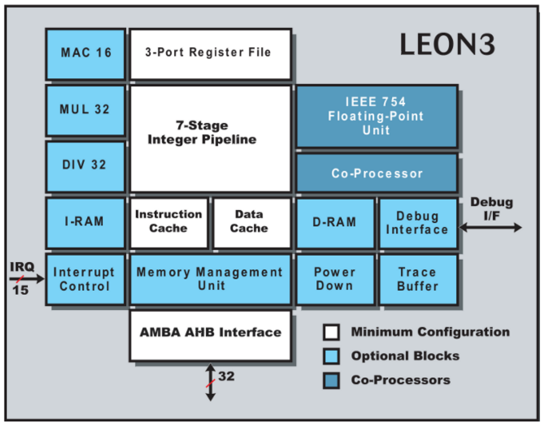


Fig. 1. LEON3 Architectural Blocks.

We synthesize each LEON3 platform configuration using the GAISLER GRLIB VHDL IP LIBRARY written in VHDL, [13], for the GAISLER GR-XC6S-LX75 and the XILINX ML605, which are two boards with XILINX FPGAs, DDR RAM, Flash, Ethernet, USB, UART, etc. The two FPGAs are flashed with various configurations, according to the four key parameters considered. The architecture and its configuration space are presented in the next subsections.

A. Architecture

The architecture we consider is based on two LEON3MP processors at 50MHz. We make the following design choices.

The integer unit is configured with SPARC V8 multiply and divide instructions. Emulated floating-point operations also benefit from this option. The integer multiplier is based on a 32x32 pipelined multiplier with a two-cycles latency.

For the cache system, LEON3 processors feature two separate instruction and data caches (Harvard architecture) with snooping and locking. The Instruction Cache (IC) is implemented as a multi-way cache with 4 ways, 4 kbytes/way and 32 bytes/line. Regarding the Data Cache (DC), it is implemented with 4 ways, 4 kbytes/way and 16 bytes/line. Four cache replacement algorithms are proposed: random, direct, Least-Recently-Replaced (LRR) and Least-Recently-Used (LRU). Local instruction and data RAM are disabled. We also choose to disable the Memory Management Unit (MMU) by default.

We implement a hardware module called LEON3 STATISTICAL UNIT (L3STAT). Eight counters are enabled, four per processor: the execution time counter (counted in CPU ticks at a frequency of 50MHz), the data and instruction cache miss counters and the branch prediction miss counter. In this work we make use of the execution time counter for evaluating each each configuration and its impact on the task executions.

1) Cache Memory: Cache memories are key elements of embedded systems since, depending on the state of the cache, task execution time could change. Both the worst-case timing analysis and the accuracy of the pWCET estimates would be impacted by the cache. We implement two cases: with caches (*caches*) and without caches (*nocaches*). In the first case the cache hierarchy is fully active, while in the second we disabled the twos caches for each processor. We expect a penalty on the execution time from the absence of cache.

2) Cache Replacement Policies: Cache replacement policies impact system performance i.e. task execution time. The way cache lines are replaced would affect next accesses; thus the latency of retrieving the information in cache changes. The replacement policies are critical to system variability especially with benchmarks which saturate the cache. The different replacement algorithms implemented are: the *Random* algorithm, which selects an item randomly and evicts it; the *Direct* algorithm, where the address of the new item is directly used to calculate its location in the cache; the *LRR* policy: it evicts the item least recently replaced; the *LRU* policy which evicts the item least recently accessed.

3) Branch Prediction (BP): The LEON3 is an advanced 7-stage pipelined processor, which implies the use of an efficient BP. According to the GRLIB Configuration Help, the BP option would improve performance with up to 20%, depending on application. In order to verify that gain of performance and measure the impact of this option on the worst-case behavior, we run the benchmarks on a platform with branch prediction (*BP*) and without branch prediction (*noBP*) for comparison.

4) Floating-Point Unit (FPU): The FPU is a high-performance, fully pipelined IEEE-754 FPU. The two FPUs provided are the GAISLER RESEARCH'S GRFPU and GRFPU-LITE. The GRFPU is a high-performance pipelined FPU with high area requirements. GRFPU-LITE provides a balanced option with high acceleration of floating-point computations combined with lower area requirements compared to GRFPU, [14]. They support all SPARC FPU instructions. If the FPU is disabled, a simulated, software FPU can be used. The option without FPU is identified *noFPU*, while that with FPU are identified as *GRFPU* or *GRFPU-lite*.

B. Architectural Configurations

A platform configuration is made of a set of architectural elements i.e. caches, cache replacement policies, BP and FPU.

We define the reference configuration as `Reference` = (*caches*, *Random* (IC), *Random* (DC), *BP*, *noFPU*). In the following, we make use of `Reference` for indicating also the choice on the element composing the reference configuration. For example, when talking about replacement policies, by referring to `Reference` we mean the random replacement policy. All the other configurations are compared with `Reference`.

Other configurations considered are:
`Direct` = (*caches*, *Direct* (IC), *Direct* (DC), *BP*, *noFPU*),
`LRR` = (*caches*, *LRR* (IC), *LRR* (DC), *BP*, *noFPU*),
`LRU` = (*caches*, *LRU* (IC), *LRU* (DC), *BP*, *noFPU*),
`noBP` = (*caches*, *Random* (IC), *Random* (DC), *noBP*, *noFPU*),
`nocaches` = (*nocaches*, N/A, N/A, *BP*, *noFPU*),
`GRFPU` = (*caches*, *Random* (IC), *Random* (DC), *BP*, *GRFPU*),
`GRFPU-lite` = (*caches*, *Random* (IC), *Random* (DC), *BP*, *GRFPU-lite*)

The configuration of a typical System on a Chip (SoC) architecture like the GAISLER GR712RC is `GR712RC` = (*caches*, *LRU* (IC), *LRU* (DC), *BP*, *GRFPU*), [15]. We will consider each choice of this architecture.

III. MICROPROCESSOR SYNTHESIS

The first step of this work is to synthesize the different architectural configurations presented in Section II. Figure 2 shows the entire methodology with the steps considered, respectively for synthesizing, benchmarking and analyzing.

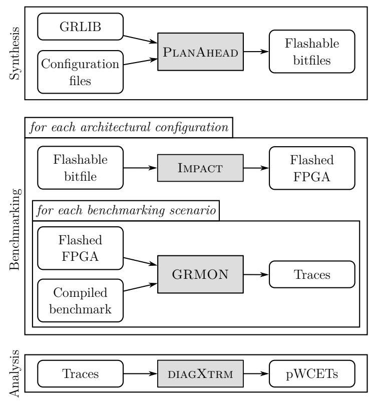


Fig. 2. Methodology applied.

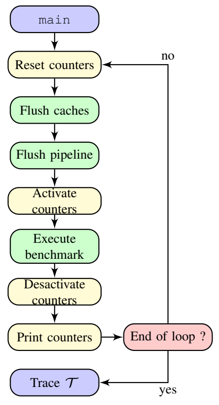


Fig. 3. Benchmarking execution diagram.

We use the GR-XC6S board for the architectural configurations `Reference`, `Direct`, `LRR`, `LRU`, `noBP` and `nocaches`; we use the ML605 board instead for the configurations `GRFPU` and `GRFPU-lite` to overcome limits in synthetizing FPUs. The GR-XC6S is a SPARTAN-6 FPGA, the ML605 board feature a Virtex-6 FPGA [16–17].

Figure 4 shows the occupations of the FPGAs in terms of slice registers (mostly Flip Flops), slice Look-Up Tables (LUTs, mostly used as logic) and the global numbers of occupied slices. The Table I presents detailed values.

We compare the different architectures related to `Reference`; as a reminder, `Reference` is the architecture with both data and instruction caches using random policy, without MMU, with branch prediction and without FPU.

• Removing the data and instruction caches reduce slices by 6%, but deteriorate run-time performances, see Section VI.

• We can see that the architectures using the Direct or the LRR replacement algorithm uses almost the same slices as the Random policy. However, the LRU policy occupies 15% more space. This area overhead is due to the addition of 5 flip-flops per line (for a 4-way LRU) to store the accesses.

• Disabling the branch prediction only saves 3% of the number of occupied slices, and reduces the performance.

• Regarding the implementation of the FPU on the ML605 board, we can see that the addition of the GRFPU uses almost twice the slices required without FPU. The GRFPU-LITE is, on the other hand, less demanding in terms of occupied slices.

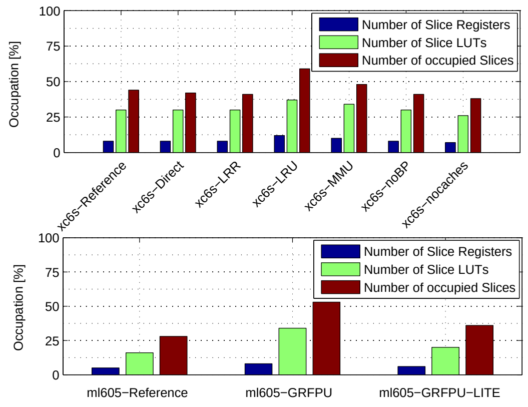


Fig. 4. Device utilization summary.

TABLE I
DEVICE UTILIZATION SUMMARY.

GR-XC6S architecture	Occupied slices out of 11662	ML605 architecture	Occupied slices out of 37680
`Reference`	5219 ⇒ 44%	`Reference`	10667 ⇒ 28%
`Direct`	4957 ⇒ 42%	`GRFPU`	20198 ⇒ 53%
`LRR`	4791 ⇒ 41%	`GRFPU-lite`	13724 ⇒ 36%
`LRU`	6895 ⇒ 59%		
`noBP`	4868 ⇒ 41%		
`nocaches`	4446 ⇒ 38%		

IV. EMBEDDED SYSTEM BENCHMARKING

The main objective of the benchmarking is to reveal the impact of each architectural change on both average and worst-case execution time of tasks. We chose the Mälardalen University WCET project benchmarks to implement tasks and to push the processor to its limits.

• *cnt* and *matmult* are used to fill entirely the data cache with a lot of matrix operations. *cnt* counts non-negative numbers in a matrix with a parametrized size of 1000×1000. *matmult* performs a matrix multiplication. As for *cnt*, the size is parameterizable and we used 200×200 matrix. Those benchmarks are applied to measure the performance of the cache replacement policy and the impact of the lack of caches if they were removed.

• *nsichneu* is similar to the first two benchmarks, but this time for the instruction cache. It uses a large amounts of if-statements (more than 250) to simulate an extended Petri Net. The benchmark is provided in three different versions, the *nsichneu_inner* is considered.

• *jfdctint* is a typical application used in embedded real-time systems. By typical application we mean an application that does not saturate neither the data cache nor the instruction cache. *jfdctint* performs a discrete-cosine transformation on a 8×8 pixel block. The code is made of long calculation sequences (i.e., long basic blocks) and single-nested loops.

- *lms* is used for FPU performances comparison. The benchmark realizes a Least Mean Squares (LMS) adaptive signal enhancement on a sine wave with added white noise.

All benchmarks are single path tasks, and so no functional variability (input vector) is considered. Also there is no need to look for the worst case path.

We use GRMON to run all benchmarks, [18]. This monitor allows a non-intrusive debugging of LEON3 systems. It also performs communication with the modules implemented in the architecture, like the L3STAT unit. Finally, it is used to download and run applications on FPGAs.

A. Execution Conditions

The benchmarks are executed on all architectural configurations, with medium optimization `-O2` and SPARC V8 multiply - divide instructions. We investigate two execution conditions:

The first execution condition is to run the task alone, without any other interaction. This execution condition is the baseline, as it is exempt of any interference tasks by construction. All benchmarks were compiled for this execution condition, and the flag `-msoft-float` was used to emulate floating-point operations when no FPU was present. This execution condition is named `ELF` after its compiler name.

The second execution condition consider using the RTEMS Real-Time Operating System (RTOS), [19]. This execution condition is more representative of an actual real-time embedded system. It is compiled in Asymmetric Multi Processor mode, to exercise perturbations between cores. This execution condition is identified as `RTEMS`.

The RTEMS Multi-Processor application is running in the pre-emptive mode on two cores. In the first core, two tasks are launched: the benchmarking task with priority 2 and a second interference task, doing infinite loops, with priority 1 (higher priority). The second core contains another task made of an infinite loop. This minimal intra- and inter-core interference is used to determine the effect of interactions and is representative of a not heavily loaded system. The methodology could apply to any kind of interference, and it will be considered in future work.

B. Initial State Assumptions

The benchmarking execution diagram is presented in Figure 3. The trace of measurements is made of 5000 consecutive executions of the same benchmark. There exists a trace per benchmark, per configuration and per execution condition.

We have to ensure that the processor is in the same initial state at each benchmark execution. Before each run, the hardware counters from the L3STAT unit are reset, the caches are flushed and the pipeline is filled with `nop()` instructions. Then, the counters are activated, the benchmark is executed and finally the eight countersare disabled and printed for that execution. The resulting trace $\mathcal{T}$ is a matrix of size 8×5000 (eight counters, measured for each of the 5000 executions). The execution time is measured in CPU ticks and the trace duration spans from 4 minutes to 30 hours, depending on the benchmark to run as well as the architectural configuration. This allows including multiple systemic effects in the traces which could happen late in the executions.

V. Probabilistic Performance Evaluation

The traces obtained from the benchmarking traces are applied to estimate the pWCETs, Figure 2.

Measurements-Based Probabilistic Time Analysis (MBPTA) allows defining tasks execution behaviors from runtime measurements. We use the MBPTA approach called diagXtrm to evaluate the impacts of architectural elements on both average and worst-case behavior of tasks. Only the execution time counter from the L3STAT traces is used.

The Extreme Value Theory (EVT) is the statistical tool composing any MBPTA that produces continuous distributions which are safe estimation of the task worst-case behavior[1]: the pWCETs. For this specific paper, diagXtrm applies the EVT in its peak over threshold version identifying the pWCETs from the Generalized Pareto Distribution (GPD) family. The distribution shape parameter ξ identifies the distribution that better cope with the measurements within this family. In particular, $\xi < 0$ is for a Weibull distribution, $\xi = 0$ is for a Gumbel distribution and $\xi > 0$ is for a Frechet distribution.

Hence, the pWCET is a worst-case thresholds with a probability associated to the risk to go past this value.

The EVT, thus the MBPTA, is applicable if the measurements are 1. stationary, 2. independent or 3. extremal independent and 4. they match a specified theoretical model, Figure 5. These situations correspond to four hypothesis that can be asserted from measurements. diagXtrm verifies all those hypotheses with the corresponding statistical tests. The confidence on the hypotheses validation is related to the quality of the pWCET model produced, named reliability of the pWCET. Only if all the tests succeed, the pWCET obtained is reliable and safe. To note that the *independence* and the *extremal independence* are partially overlapping hypotheses; if one is verified, there is no need to verify the other, Figure 5.

The robust statistics [3–20] is applied by diagXtrm for quantifying the uncertainties from statistical tests. It defines the confidence cl on the hypothesis verified.

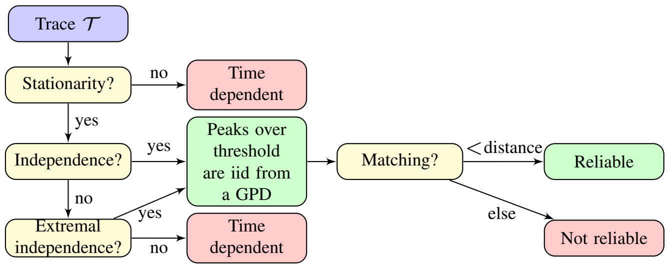


Fig. 5. Decision diagram of diagXtrm with tests and action applied.

[1]By safe estimation we mean *a pWCET which is larger than or equal to any possible task execution time*, [8].

a) Stationarity Hypothesis: The analysis of trace can show the relationship between consecutive measurements and evaluate the impact that previous (in time) measurements on future ones, i.e. the independence between measurements. The EVT applicability relates to stationary traces. An established test for verifying this hypothesis is the Kwiatowski Phillips Schmidt Shin test, [21]. The confidence associated is cl_1.

b) Independence Hypothesis: The statistical dependence is checked through the existence of correlated patterns of measurements, using the Brock Dechert Scheinkman statistical test [22]. The confidence on the independence is $cl_{2.1}$.

c) Extremal Independence Hypothesis: When overall independence does not hold, another way is to look for the independence of extreme measurements[2]. The extremal index $\theta \in [0, 1]$ is an indicator of dependence degree between extreme measurements, [23]. We denote this parameter $cl_{2.2}$.

d) Matching Hypothesis: The matching test is based on a quadratic statistic which measures the square distance between the pWCET model estimated and an hypothetical exact pWCET model following the GPD. The Cramer Von Mises criterion is applied for verifying the degree of matching between the two distributions, leading to parameter cl_3.

Finally, DIAGXTRM computes pWCET parameters such as the threshold u and the pWCET distribution shape ξ that guarantee the best pWCET model among the possible EVT pWCETs, see [3] for details.

We stress the fact that with MBPTA approaches, the resulting pWCET model represents the worst-case for the execution conditions accounted for by the measurements. It is not an absolute worst-case. Thus safety, confidence and reliability relates to the specific measurements considered.

VI. Trace Analysis

Given a trace of measurements, DIAGXTRM profiles both the average and the worst-case behavior of tasks. We use that tool for each architectural configuration and present the results in the following subsections.

A. Average Performance

At first we focus on the average execution time to compare the architectural configuration effects.

1) Execution Time Variability: Figure 6 illustrate the execution time variability of any system. More precisely, the trace is the result of 5000 executions of the *cnt* benchmark on the reference architecture without any RTOS. That proves that even when an application is run in a standalone mode, variability is present due to systemic and functional changes happening at runtime.

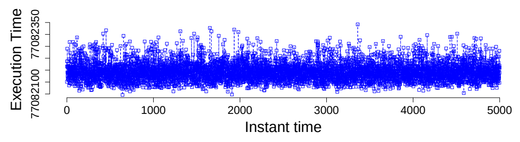
2) Impact of the Cache Replacement Policy: We analyze the traces of the two benchmarks *matmult* and *nsichneu* which saturate respectively the data and the instruction caches.

Figure 7a presents the box plots of execution times for the *matmult* benchmark. We can see that in order to improve the average performance of the data cache, the best cache replacement policy is the LRU algorithm. Then comes the random, the direct and finally the LRR algorithm.

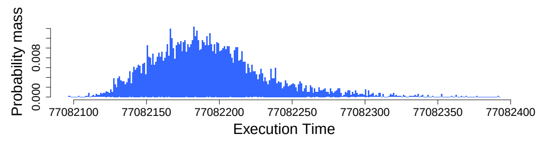
As a matter of fact, programs usually compute results thanks to a restricted set of variables. It is then natural that the LRU algorithm is the best choice for the data cache replacement policy. This result is consistent with the fact that multiple SoC architectures implement the LRU algorithm for their data cache replacement policy, like the `GR712RC` configuration.

Figure 7b illustrates the execution times as box plots for the benchmark *nsichneu*. We can see that the LRU algorithm is not the best replacement policy for the instruction cache. The random algorithm is the first, then the direct, the LRU and finally the worst is the LRR.

For instruction cache the random replacement policy has better performance on average than the other policies, [9]. Unlike data variables accesses, sequences of instructions of any program are not really regular. That explains why the random algorithm is the best instruction cache replacement policy. However, the choice of another algorithm is not critical as the drawback induced on the execution time is not significant, comparatively to the bigger differences of execution times related to the data cache replacement policies, Figure 7a (to note that the scales are not the same in the two figures).

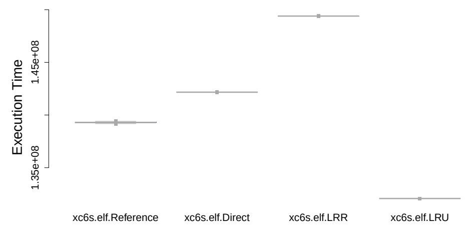


(a) Time trace of execution times.

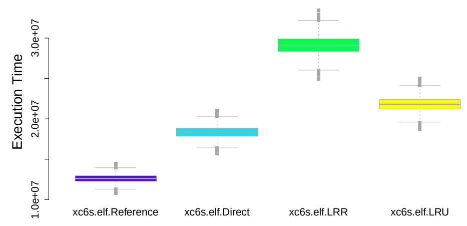


(b) Histogram of execution times.

Fig. 6. Time trace and histogram of execution times (benchmark *cnt* on the GR-XC6S reference architecture without RTEMS).



(a) Bench. *matmult* on the GR-XC6S without RTEMS.



(b) Bench. *nsichneu* on the GR-XC6S without RTEMS.

Fig. 7. Box plots of execution times for different cache replacement policies.

[2]By extremal measurements we allude to observations relatively far from the average values as well as observations well separated in time.

3) Disabling Key Architectural Elements: In order to evaluate the gain of performance provided by the branch prediction or the cache memories, we disabled them from reference architecture. The Table II summarizes the execution time distributions and the performance drawbacks for different benchmarks. We chose *matmult* and *nsichneu* because they overuse the cache memories when present. Thus, those benchmarks are prone to reveal the catastrophic effects of disabling the instruction and the data caches. The third benchmark analyzed is *jfdctint* which is applied for investigating the impact on the execution time for an application that does not saturate caches.

TABLE II
EXECUTION TIME DISTRIBUTIONS AND PERFORMANCE DRAWBACKS FOR DIFFERENT BENCHMARKS AND ARCHITECTURES.

(a) Benchmark *matmult* on the GR-XC6S without RTEMS.

GR-XC6S architecture	**Execution Time Distribution**			**Perf. drawback**
	Min	**Mean**	**Max**	
`Reference`	139147148	139292184	139447778	–
`noBP`	154722839	154869205	155006999	11.19%
`nocaches`	1095326983	1095468302	1095562181	687.17%

(b) Benchmark *nsichneu* on the GR-XC6S without RTEMS.

GR-XC6S architecture	**Execution Time Distribution**			**Perf. drawback**
	Min	**Mean**	**Max**	
`Reference`	10828017	12624121	14445302	–
`noBP`	11597909	13516958	15485162	7.20%
`nocaches`	94652781	110600474	126897285	778.47%

(c) Benchmark *jfdctint* on the GR-XC6S without RTEMS.

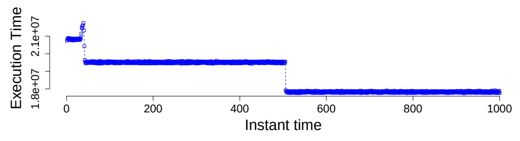
GR-XC6S architecture	**Execution Time Distribution**			**Perf. drawback**
	Min	**Mean**	**Max**	
`Reference`	2468750	2468762	2468788	–
`noBP`	2546749	2546760	2546791	3.16%
`nocaches`	16326625	16328010	16329192	561.43%

As described in Section II, the branch prediction deteriorates performances up to 20% if disabled. A maximal drawback of 11.19% is measured, Table IIa.

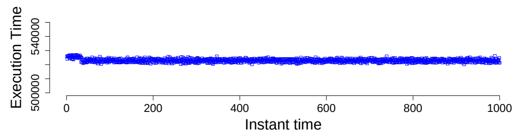
The cache memories impact on the execution time distribution is huge. For the same application, a factor of 8.78 on the execution time can be observed (Table IIb), where a factor of at least 2-3 was expected according to the GRLIB Configuration Help. For a typical application, the performance drawbacks are not negligible, Table IIc.

The results demonstrate that branch prediction and cache memories are key elements for embedded systems. Although they clearly improve average performances, they increase execution time variability. The system predictability or analyzability are affected by those elements.

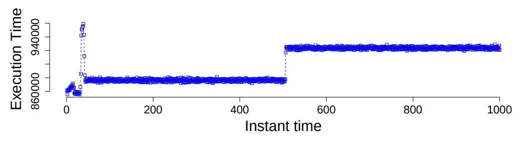
4) Impact of The FPU Architecture: We use the benchmark *lms* that computes a lot of floating-point operations. Figure 8 presents the first 1000 executions of the resulting traces. We note that there are different execution modes, each with some variability, depending on the FPU architecture. The only element which is not reset between each benchmark execution is the FPU branch prediction, as the integer unit branch prediction and the cache memories are flushed each time. The multiple modes are then probably due to a non-zero prediction distance as detailed in [24].



(a) Reference architecture with software FPU (emulated).



(b) Architecture with hardware FPU: GRFPU.



(c) Architecture with hardware FPU: GRFPU-LITE.

Fig. 8. Time traces of execution times for different Floating-Point architectures (benchmark *lms* on the ML605 without RTEMS).

In the remaining 4000 measurements only one execution mode exists, revealing a convergence to a mode after some executions. Its mean value is used for comparison between the 3 FPUs. The emulated FPU has a mean level of $17'826'717$ CPU ticks, the GRFPU converges around $522'888$ CPU ticks and the GRFPU-LITE at $923'719$ CPU ticks. This corresponds to improvements of respectively $33\times$ and $18\times$ faster.

The advantages of using a hardware FPU instead of an emulated FPU for floating-point applications are comforted. The GRFPU is up to 34 times faster than a software FPU. The GRFPU-LITE is a good compromise between speed improvement (19 times faster) and occupied slices (see Section III).

A side comment regards the capability that DIAGXTRM has of modeling architectural elements without the need for an exact element model. For example, with *lms* and the FPU architecture, although not knowing the internal behavior of the FPUs applied, DIAGXTRM is able to characterize the different modes just from measurements.

Table III summarizes the average execution time of each architectural change. This is done for the five benchmarks of interest and for each execution condition.

Two additional configurations were synthesized for each FPGA: theyare identified as `Best`. For the GR-XC6S it is `Best1` = (*caches*, *Random* (IC), *LRU* (DC), *BP*, *noFPU*) and for the ML605 it is `Best2` = (*caches*, *Random* (IC), *LRU* (DC), *BP*, *GRFPU*). They are the best configuration since they collect the choices which improves the most the average performances, respectively for each FPGA architectures.

On a global scale, the `Best` configurations improve the performance more than they deteriorate it, Table III. The architectures without BP or without caches are clearly the worst. The LRU algorithm is not always the faster, as we can see with the *nsichneu* benchmark, but its choice rather than the random replacement policy for the data cache is profitable. As we can see, `Best` configurations allows an additional gain of 1.01% on the GR-XC6S and of 8.36% on the ML605

TABLE III

MEAN EXECUTION TIME IN THOUSANDS OF CPU TICKS OF EACH BENCHMARK FOR DIFFERENT ARCHITECTURAL CONFIGURATIONS.

Embedded system benchmarking		GR-XC6S Architectures							ML605 Architectures			
		Cache Replacement Policies				Key Elements		Best	FPU			Best
Env.	**Bench.**	`Reference`	`Direct`	`LRR`	`LRU`	`noBP`	`nocaches`	`Best1`	`Reference`	`GRFPU`	`GRFPU-lite`	`Best2`
`ELF`	*cnt*	77.082	77.082	77.082	77.082	81.132	436.601	77.082	77.276	77.276	77.276	77.276
	matmult	139.292	142.168	149.393	132.077	154.869	1095.468	132.077	129.721	129.721	129.721	129.721
	nsichneu	12.624	18.337	29.122	21.806	13.517	110.600	12.624	21.638	21.791	21.785	12.627
	jfdctint	2.469	2.469	2.469	2.469	2.547	16.328	2.469	2.501	2.485	2.485	2.501
	lms	17.738	17.737	18.312	17.737	19.108	178.854	17.738	17.827	0.523	0.924	0.524
`RTEMS`	*jfdctint*	2.496	2.498	2.503	2.495	2.575	18.268	2.496	2.530	2.532	2.532	2.532
	lms	17.980	18.032	18.304	17.977	19.387	207.048	17.979	18.078	0.535	0.939	0.535
Global time scores[3]		–	+8.32%	+24.58%	+11.67%	+6.62%	+663.63%	-1.01%	–	-19.35%	-18.91%	-27.71%

in comparison with the second best architectures, respectively `Reference` and `GRFPU`.

Finally, the effect of the RTOS can be observed: the mean execution times undergo a penalty of 2.52% on average with `RTEMS` as compared to standalone `ELF`.

B. Worst-Case Performance

This section focuses on the worst-case performance and the model used for estimating the pWCETs. The methodology applied as well as the quality of the pWCET models (reliability and hypothesis confidence) are detailed. Two cases are investigated: the impact of different cache replacement policies and the impact of the RTOS on worst-case behaviors.

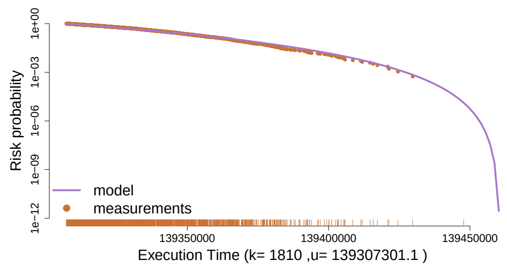


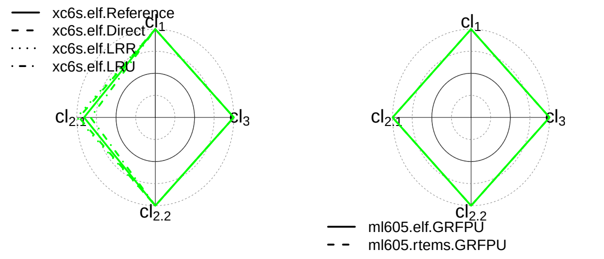
Fig. 9. Measurements and model for pWCET estimation (benchmark *matmult* on the GR-XC6S without RTEMS).

1) Probabilistic Distribution Fitting: Figure 9 shows the measurements and the pWCET derived from the EVT. The inverse cumulative distributions are represented. u is the threshold parameter that DIAGXTRM computes for obtaining the best pWCET model; k is the number of measurements above the threshold selected. We see how the pWCET model is able to perfectly cope with the input measurements and then safely infers the rare events (risk probability $\leq 10^{-9}$).

2) Quality of the Probabilistic WCET Model: As described in Section V, the reliability of the pWCET model is based on the confidence levels of the four hypotheses $\{cl_1, cl_{2,1}, cl_{2,2}, cl_3\}$. Each hypothesis is scored out of 4.0, and a minimum of $1.0/4.0$ is required in order to apply the method. Hypotheses $cl_{2.1}$ and $cl_{2.2}$ are covering each other, so that a score of $1.0/4.0$ in one of the two is sufficient. The spider net in Figure 10 represents the reliability of the pWCET model from these confidence levels.

[3]For each benchmark, a score is determined as $\frac{executionTime}{\max(executionTimes)}$. Then, the sum of the scores are compared relatively to those of the `Reference` configurations.

Hence, all the assumptions are satisfied, the model is reliable. Stationarity, independence, extremal independence and matching are verified with almost the maximal confidence. The EVT is applicable with realistic architectures as there is enough variability from the measurements.



(a) Different cache replacement policies (bench. *matmult* on the GR-XC6S without RTEMS).

(b) Impact of the RTOS: without and with RTEMS (benchmark *lms* on the ML605).

Fig. 10. Reliability of the pWCET estimations (scores out of 4.0 for each hypothesis).

3) Comparisons of Worst-Case Behaviors: From the pWCET distributions we can extract the WCET thresholds with a 10^{-9} risk probability i.e. confidence. This allows us to identify the impact of each architectural change on the worst-case behaviors with single values. Table IV indicates the pWCET distribution shape parameter ξ for each pWCET model as well as the theoretical estimations i.e. the WCET thresholds at 10^{-9}. DIAGXTRM automatically selects ξ to best fit the input measurements. As it can be seen, the measurements have small variance hence they are better approximated with a Weibull GPD distribution, $\xi < 0$.

The accuracy of the pWCET is: accuracy $\overset{def}{=}$ $\frac{\text{Theoretical estimation} - \text{Max measured}}{\text{Max measured}}$ and defines how close the pWCET (Theoretical estimation) is to the maximum measured execution time (Max measured). As the models are all Weibull distributions, thus bounded to the right, they are quite close to the maximum measured values, Table IV. The small accuracy indicates that the measured values embed already the worst-case conditions. Thus, the pWCET as Weibull distribution is able to perfectly cope with the measurements and the task average behavior. The case with accuracy equal to 1.32E-01, Table IV `ELF`, indicates that the measurements have large variability. This means that the worst-case conditions could have a significant impact on the task execution times. The resulting Weibull, in order to account for that and foreseen the worst-cases, has to differ more from the measurements. Nonetheless, the pWCET is equally reliable due to the maximum confidences achieved.

The impact of the cache replacement policy on the worst-case behavior is presented in Table IVa. The results are consistent with those presented in Section VI-A2.

The benchmark *lms* was used to characterize the impact of the RTOS. We noted that the behavior of the FPU is not stable in the first 1000 runs of the benchmark, thus we use the last 4000 iterations to determine the pWCET distribution (i.e. achieve the hypotheses scores presented before in Figure 10b). The Table IVb gives the results. They confirm that the use of a RTOS like RTEMS induces a small penalty of 2.85% on the task performance: minimal interference were added and then accounted by a more pessimistic pWCET.

TABLE IV
pWCET DISTRIBUTION PARAMETERS AND ESTIMATIONS FOR DIFFERENT BENCHMARKS AND ARCHITECTURES

(a) Impact of the cache replacement policy (benchmark *matmult* on the GR-XC6S without RTEMS).

GR-XC6S	**Distribution**	**diagXtrm results**		
architecture	**shape ξ**	**Theoretical**	**Measured**	**Accuracy**
Reference	-0.249	139447252.58	139433822	9.63E-03
Direct	-0.219	142170799.12	142170390	2.88E-04
LRR	-0.095	149397501.05	149396595	6.06E-04
LRU	-0.317	132076829.57	132076819	8.00E-06

(b) Impact of the RTOS (benchmark *lms* on the ML605).

ML605	**Distribution**	**diagXtrm results**		
with GRFPU	**shape ξ**	**Theoretical**	**Measured**	**Accuracy**
ELF	-0.149	527244.18	526547	1.32E-01
RTEMS	-0.596	542279.08	542228	9.42E-03

VII. Conclusion and Future Work

In this work, we review i) configuration space of LEON processors on FPGA and ii) benchmarking LEON3 processors elements. Some architectural choices have been made, described and then synthesized on two FPGA platforms. Tasks benchmarks have been selected for their ability of revealing the impact of each architectural configuration on average and worst-case behaviors. Some execution conditions have been chosen in order to expose key parameters in execution time variability in both standalone and RTOS environments.

The results verify that time execution variability is present in any embedded processor. Average performance analyses show that the best data cache replacement policy is the LRU algorithm, while the random algorithm is the best choice for the instruction cache. The results confirm that cache memories, branch predictions and high-performance math co-processor are key elements of modern processors. Both contribute in improving average performance at the cost of increased variability in the system. We highlight configurations that are best compromise: they improve performance in almost every case, and provide guidelines for predictable embedded systems.

Probabilistic models have been estimated in order to see the impact of different cache replacement policies and the impact of the RTOS on the worst-case task behaviors. We validated the reliability and the accuracy of the resulting pWCETs.

Our approach could be applied with other benchmarks or other architectural configurations. Future work will study interference among tasks in RTOS, and evaluate the impact of scheduler or resource sharing at software and hardware levels (cache, MMU, etc.). This will improve our understanding of modern processors so as to guarantee time constraints.

References

[1] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzetti, E. Quiñones, and F. J. Cazorla, "Measurement-based probabilistic timing analysis for multi-path programs," in *24th Euromicro Conference on Real-Time Systems, ECRTS 2012, Pisa, Italy, July 11-13, 2012*, pp. 91–101, 2012.

[2] Mälardalen University, "Mälardalen WCET benchmark programs." http://www.mrtc.mdh.se/projects/wcet/benchmarks.html, 2013.

[3] F. Guet, L. Santinelli, and G. Morio, "On the reliability of the probabilistic worst-case execution time estimates," in *Proceedings of the 8th European Congress on Embedded Real-Time Software and Systems (ERTSS)*, 2016.

[4] I. MHADHBI, N. Rejeb, S. B. OTHMEN, and S. B. SAOUD, "Performance Evaluation of FPGA Soft Cores Configurations Case of Xilinx MicroBlaze," *International Journal of Computer Science, Communication & Information Technology (CSCIT)*, vol. 1, pp. 14–19, 2014.

[5] D. Mattsson and M. Christensson, "Evaluation of synthesizable cpu cores," *Master's thesis, Department of Computer Engineering, Chalmers University of Technology*, 2004.

[6] W. Jiang, Z. Guo, Y. Ma, and N. Sang, "Measurement-based research on cryptographic algorithms for embedded real-time systems," *Journal of Systems Architecture*, vol. 59, no. 10, pp. 1394–1404, 2013.

[7] R. Njuguna, "A survey of fpga benchmarks," *Project Report, November*, vol. 24, 2008.

[8] R. I. Davis, L. Santinelli, S. Altmeyer, C. Maiza, and L. Cucu-Grosjean, "Analysis of probabilistic cache related pre-emption delays," *Proceedings of the 25th IEEE Euromicro Conference on Real-Time Systems (ECRTS)*, 2013.

[9] S. Altmeyer, L. Cucu-Grosjean, and R. I. Davis, "Static probabilistic timing analysis for real-time systems using random replacement caches," *Real-Time Systems*, vol. 51, no. 1, pp. 77–123, 2015.

[10] J. Hansen, S. Hissam, and G. A. Moreno, "Statistical-Based WCET Estimation and Validation," in *9th International Workshop on Worst-Case Execution Time Analysis WCET*, pp. 1–11, 2009.

[11] RTE - Realtime Embedded, "LEON3 32-bit processor core." http://www.rte.se/blog/blogg-modesty-corex/leon3-32-bit-processor-core/1.5, 2012.

[12] Aeroflex Gaisler, "LEON3 Product Sheet." http://gaisler.com/doc/leon3_product_sheet.pdf, 2010.

[13] Aeroflex Gaisler, "GRLIB VHDL IP Core Library." http://www.gaisler.com/products/grlib/grip.pdf, 2016.

[14] Aeroflex Gaisler, "GRFPU - High Performance Floating Point Unit." http://www.gaisler.com/doc/grfpu_product_sheet.pdf, 2010.

[15] Aeroflex Gaisler, "GR712RC User's Manual." http://www.gaisler.com/doc/gr712rc-usermanual.pdf, pp. 43–44, 2016.

[16] Xilinx, "Spartan-6 Family Overview." http://www.xilinx.com/support/documentation/data_sheets/ds160.pdf, 2011.

[17] Xilinx, "Virtex-6 Family Overview." http://www.xilinx.com/support/documentation/data_sheets/ds150.pdf, 2015.

[18] Aeroflex Gaisler, "GRMON2 User's Manual." http://www.gaisler.com/doc/grmon2.pdf, 2016.

[19] OAR Corporation, "RTEMS, An Open Real-Time Operating System." http://www.rtems.com, 2016.

[20] J. J. Buckley, "Fuzzy statistics: hypothesis testing," *Soft Comput.*, vol. 9, no. 7, pp. 512–518, 2005.

[21] D. Kwiatkowski, P. C. B. Phillips, P. Schmidt, and Y. Shin, "Testing the null hypothesis of stationarity against the alternative of a unit root : How sure are we that economic time series have a unit root?," *Journal of Econometrics*, vol. 54, pp. 159–178, 00 1992.

[22] W. A. Brock, J. A. Scheinkman, W. D. Dechert, and B. LeBaron, "A Test for Independence based on the Correlation Dimension," *Econometric Reviews*, vol. 15, no. 3, pp. 197–235, 1996.

[23] P. J. Northrop, "An efficient semiparametric maxima estimator of the extremal index," *Extremes*, vol. 18, no. 4, pp. 585–603, 2015.

[24] G. Tyson, "The effects of predicated execution on branch prediction," in *Proceedings of the 27th annual international symposium on Microarchitecture*, pp. 196–206, ACM, 1994.