On the Determinism of Multi-core Processors

Vladimir-Alexandru Paun, Bruno Monsuez, Philippe Baufreton

On the Determinism of Multi-core Processors*

Vladimir-Alexandru Paun[1], Bruno Monsuez[1], and Philippe Baufreton[2]

1 **UIIS**
ENSTA ParisTech
828, Boulevard des Maréchaux, 91762 Palaiseau Cedex, France
surname@ensta-paristech.fr

2 **Sagem – SAFRAN Electronics**
Etablissement F. Hussenot – R&T
100 avenue de Paris – 91344 MASSY Cedex France

Abstract

Hard real time systems are evolving in order to respond to the increasing demand in complex functionalities while taking advantage of newer hardware. Software development for safety critical systems has to comply with strict requirements that will facilitate the certification process. During this process, each part of the system is evaluated, requiring a certain level of assurance in order to provide confidence in the product. In particular there must be a level of confidence that the system behaves deterministically that may be based on functionality, resources and time. The success of system verification depends greatly on the capacity to determine its exact behavior. Nonetheless, hardware evolved in order to maximize the average computation power throughput with little to no regard to the deterministic aspect. Therefore modern architectural features of processors, like pipelines, cache memories and co-processors, make it hard to verify that all the needed properties are respected. The multi-core is furthermore difficult to analyze as the architecture employs mechanisms that compromise strong spatial and temporal partitioning when using shared resources without rigorous access control like shared caches or shared input/outputs. In this paper we identify and analyze the main sources of nondeterminism of the multi-cores with regard to the timing estimation. Precise determination of the worst case execution time is a challenging task even in single-core architectures. The problems are accentuated in the multi-core context mainly due to the resource sharing that can lead to highly complex interactions or to nondeterminism. Most of the units that generate behaviors that are hard to take into account can be deactivated, but it is not always easy to predict the impact on the performance. Nevertheless some of the features cannot be disabled (such as the out of order execution or some nondeterministic crossbar access policies) which leads to the invalidation of the respective platform for applications with high criticality level. We will address the problematic units, propose configuration or architecture guidelines and estimate their impact on the performance and determinism of the system.

1998 ACM Subject Classification C.3 Special-Purpose and Application-Based Systems

Keywords and phrases multi-core, determinism, hard-real time systems

Digital Object Identifier 10.4230/OASIcs.FSFMA.2013.32

1 Introduction

The use of complex computers in safety-critical systems creates the need to ensure that the embedded systems act in the way they are supposed to and that consequences of a

* This work was supported by SAFRAN Sagem.

1st French Singaporean Workshop on Formal Methods and Applications 2013 (FSFMA'13).
Editors: Christine Choppy and Jun Sun; pp. 32–46
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

malfunction are completely handled in a safe manner. Different standards apply according to the danger level of the system failure. These standards are presented in a collection of guidelines to follow in order to empower the system with a necessary confidence level. The respect of these recommendations determine the success of the certification process necessary for the software approval.

In this paper we focus on the determinism issues related to the worst-case execution time (WCET) with regards to the avionics standards of software certification that will guide the study of potential difficulties of embedding multi-cores. For the assurance of commercial avionics systems a document called "Software Considerations in Airborne Systems and Equipment Certification" is used. Bearing the name DO-178B [21] in the US and ED-12B in Europe, it describes the objectives of software life-cycle processes, process activities and the evidence of compliance required at different software levels. Safety standards like DO-178B and IEC-61508 [13] explicitly call for the identification of functional and non-functional hazards and for software compliance with the relevant safety goals. In these standards three important non-functional software characteristics related to safety are mentioned: absence of run-time errors, execution time and memory consumption. The IEC-61508 has a great impact on the hardware selection as it requires the absence of unpredictable timing-related interferences which might affect real-time functions. This impacts directly the multi-cores as it must be ensured that no inherent timing interferences between cores take place. Nevertheless, this type of interferences are quite common and must be dealt with. Existing multi-core architectures employ mechanisms that compromise strong spatial and temporal partitioning when using shared resources without rigorous access control. Therefore we do not always dispose of precise information regarding the timing of some instructions in all circumstances due to their complex interactions with the memory and other dependencies.

The WCET estimation consists of two main steps, namely the control-flow analysis that determines the feasible paths in a program, and the processor-behavior analysis based on low-level analysis, hence the need to throughly determine the hardware behavior. The choice of a hardware platform is therefore greatly influenced by the visibility on the device internal structure as precise architectural implementation details are proprietary data often undisclosed.

The WCET estimation in multi-cores has different levels of difficulties. The first one is inherited from the single core world. Certain modern features in processors cannot be safely analyzed, nor disabled, making the certification of the processor impossible. Other features generate imprecision that will increase the safe margins needed to take in order to comply with the safety constraints, which impacts the feasibility. The second one is introduced by the multi-core architecture and can lead to the impossibility to determine the WCET. Problems from a core are not only translated, when integrated into the multi-cores, but amplified by the context of resource sharing.

State of the art works deal with this issue either by constraining some platforms, or by handling only a part of the issues and giving some new architectural workarounds that are custom tailored for some applications [30, 6, 5, 15, 31, 17, 11, 10, 9, 29]. Another approach is to gather best practices for future multi-core architectures [4] after acknowledging that analysing current multicore architectures is impossible in general. Nevertheless a unified and detailed approach is yet to be available.

Identifying which parts are impossible to analyze, or at what cost in precision, is key to even considering the choice of a certain multi-core processor. Our work is intended as a guideline in such choices, exposing the inherent problems of multi-cores and straitening the path towards a solution in the matter. The article is structured as follows: First we explain inherent problems to the use of microprocessors in hard real-time systems. In Sec. 3 we describe the major units of the processor, identify problematic execution scenarios and estimate the impact on the predictability and we conclude in Sec. 4.

2 Inherent problems to the use of microprocessors in hard real-time systems

The hardware platform is a central point when analyzing a system. Therefore it is essential to dispose of a precise model of the processor in order to determine its effective behavior. The available information comes mainly in reference manuals and application notes that present the processor's architecture, how to interface it with the environment and how to configure its different function modes. Nevertheless, information present in the user manual is not intended for testing or verification purposes. Furthermore information relevant to the design method and the verification methodology are only briefly discussed, if not at all in these documents, mainly from the integrator point of view. Another issue is the questionable validity of the information presented in the reference document altogether as contradictory information is sometimes provided in different document.

The behavior of a microprocessor is challenging or even impossible to characterize. This is either due to the uncertainty of the effectively calculated result or the uncertainty on the actual time of the effective calculation. De facto, these two aspects are directly related to the notion of data availability. The main consequence of the difficulty in architectural optimization is an interdependence between the data and the instructions of a same task. In the context of multitask applications, an interdependence between various tasks coming from the commutation of the environments during the passing from one task to another, is introduced. In the case of multi-cores additional difficulties appear in the estimation of the WCET, like the issue of two competing processes if they share common architecture elements, notably, memory access controllers. Futhermore challenges are given by the exchange of data between the two applications being executed in both processors. Let us consider two tasks, one critical task being executed in one core, and a second critical task monitoring the calculations of the first task on the second core, it is necessary to ensure the data handled by the two tasks is coherent. Without adding at the application level advanced synchronization operation, it is impossible for the majority of current multi-core microprocessors to guarantee that both applications handle the same data. In fact, due to induced latencies, nothing prevents one of the two applications from handling data d_{t-1} present at $t-1$ instant while the other application handles the data d modified at t instant. In fact, one of the rare means to remove these uncertainties would be to strongly pair the monitoring application with the command application, by implanting communications between the two applications via semaphores.

3 Hardware considerations

Most of the available processors were not especially designed for the hard real-time systems. The multi-cores are no exception as their goal is to maximize the shared resource average utilization. Data communication and synchronization between the different units are optimized for maximum throughput of executed instructions. Therefore multiple execution paths can be taken depending on the execution history, the current state of units or even local choices based on random decisions. A natural way of analyzing the hardware components that influence the determinism would be to first look at those who give a local impact and proceed to components that have a global impact. One can also start by looking into units already present in single-cores, and proceed with units unique to the multi-cores. However, as shaped by this section, there is a thin frontier between the two as even classical, predictable units have a different impact when integrated in the multi-cores. Therefore the analysis of this components can not be solely based on the analysis of the same component in the single-core context.

3.1 Pipeline

Present in all modern processors, the pipeline was introduced in order to increase the average performance by ensuring that, whenever possible, an available hardware resource will be occupied. Nevertheless, different events can introduce pipeline stalls such as structural hazards, data hazards and control hazards.

Out of order execution (OoOE), a feature introduced in order to avoid pipeline stalls by decoupling the issue/dispatch and the execution/completion stages, allows execution not following the instructions program order. A fetched instruction will be executed when the input operands and needed resources are available with no regard to whether it is the next in order instruction. The interaction between the cache memory and instruction scheduling influences the precision of the timing estimation.

Pipeline impact on the predictability

The impact of the pipeline varies from local influence with local monotonic optimizations to global influences with timing anomalies that cancel the monotonicity and compositionality. The size of the pipeline has an influence on the predictability of the WCET. A wrong branch prediction causes n cycles penalty, where n is the pipeline depth. The pipeline depth can further influence the predictability potentially generating more hazards as more instructions are being treated at the same time. Besides intrinsic impacts on the predictability, the pipeline, in conjunction with other units, can lead to precision loss or nondeterminism. For example, in case of a L2 cache miss, the number of pipeline stages influences the memory access time [8]. Furthermore, in the shared resources context of multi-core sharing a common bus, the pipeline can lead to nondeterminism.

3.2 Branch Prediction Unit (BPU)

Through the BPU, processor attempts an early resolve of a branching instruction, before its time, by applying a strategy in order to anticipate the result. The BPU strategy can be either static or based on complex algorithms, un-deterministic in some cases. Based on this estimation, a speculative execution is initiated that will lead eventually to a significant time gain in the case the result is correct. The influence on the cache memory content is non negligible as a miss-prediction is not generally followed by a cache reorganization, therefore the cache configuration is polluted with information from the untaken path.

BPU impact on the predictability

The BPU can make incorrect branch predictions or incorrect branch target address lookup. It is systematically active and directly impacts the temporality of the instruction change. Furthermore, this unit relies on a set of data protection tables stored in tables. The impact is high because of the general unpredictable success rate of the early branching target resolution. It can be largely avoided in the case of statically resolved loops or branches that are not data dependent. Tailoring the condition of the jump taking into account the branching strategy in order to help it succeed in the majority of cases is also a solution as long as the WCET analyzer can take it into account. In this case, most techniques of adding watermarks in the code with information that help or enable the prediction can be useful.

3.3 Floating Point Unit (FPU)

Floating point computation timing can also be hard to accurately estimate because of their implementation. A micro-pipelined unit takes advantage of consecutive instructions that can be pipelined. Units can have either a part of the FPU instructions pipelined or all of them. Therefore consecutive pipelined and non-pipelined instructions can cause stalls, making the timing difficult to compute especially in the case of the out of order execution.

Floating-point data formats and instruction set generally conform to the IEEE Standard for Binary Floating-point Arithmetic, ANSI/IEEE Standard 754-1985. However, the SPARC V8 architecture, for example, does not require that all aspects of the standard, such as gradual underflow, be implemented in hardware [25]. This can be a problem if precise information about the implementation is not given. One of its implementations, the GR712RC/LEON3 does not provide sufficient information on this matter and precise timings in case of this exception could prove difficult to estimate. Similarly the ARM Cortex A9 architecture manual, does not provide precise timing of all instructions. This is mainly due to the unpredictable timing behavior at the instruction level generated by the unit's structure itself and memory system interactions [32].

FPU impact on the predictability

The impact of the FPU depends on its implementation. Instructions can take either a single cycle to execute or several cycles but they can also be pipelined. A combination of either way in parallel is also possible. In conjunction with the instruction rescheduling and thus with the change of data, cascade effects can occur and lead to pathological effects like it can be seen in some PowerPC architectures.

3.4 Level 1 Cache

Memories for instructions and data are implemented in order to make the most common case fast, benefiting from a program's spatial locality and temporal locality. Not taking this fact into account in the WCET estimation gives highly pessimistic timing estimations. Cache memory is usually organized in different levels, some local to the core and others situated outside the core. Different cache replacement strategies must be implemented in order to optimize the performance because a strategy that can fit all the possible cases is impossible to find. Therefore the average case performance is optimized. Commonly used strategies are LRU, pseudo LRU, FIFO or round robin and MRU each having a different impact on the predictability of the system. The analysis of the Level 1 cache must be made in conjunction with the other units and is discussed within the timing anomalies in the following.

Impact on the predictability

Worst-case analysis on cache memories is a challenging problem, mainly because they are conceived in order to maximize the average performance. Achieving good results for data cache analysis, for example, is still an open problem, as they are difficult to statically analyze. An approach that enables time-predictable caching, is to lock cache blocks. Combining cache locking with cache partitioning for multiple tasks in the case of task preemption can improve the predictability in some cases [26]. Unknown abstract cache states during the analysis generate loose WCET bounds. For example, unified cache for instruction and data can break down all the information on abstract cache states. After accessing n unknown addresses in an n-way set-associative cache all the cache lines will be unclassified in the analysis. Therefore, separation between the instruction and data cache memories should be chosen whenever possible (the problem still holds for shared caches and is discussed in the Level 2 cache section). For this reason Harvard architectures, with physically separate signal pathways for instructions and data, should be privileged in despite of the von Neumann architecture. Context switch or cache misses it can lead to a relatively high global impact due to timing anomalies or Translation Lookaside Buffer (TLB) strategies. In order to improve the performances of the cache memory, instruction and data locality could be increased using compiler techniques for example (code reposition, loop permutation, tiling [28] etc.). When performing the WCET analysis, the most problematic features to analyze are the replacement policies for set-associative caches [12]. Pseudo-round-robin and the 4-way associative cache is also a difficult combination in the Motorola ColdFire 5307 [23]. In order to ensure the time-predictability of processors, locally deterministic update strategies for caches should be used. According to [22] the LRU strategy performs best in terms of predictability, far ahead pseudo-LRU and FIFO.

3.5 Scratchpad

Scratchpad memories (SPMs) are used to guarantee a unit can work without main memory contention in a system employing multiple processors. As the memory access latencies are predictable, scratchpad memories have become popular for real-time embedded systems. However, the difficulty of allocating code/data to scratchpad memory lies now with the compiler. Scratchpad memory works like a local store and act like "software caches" therefore the strategy is implemented in software and the interactions in the global hardware must be analysed. Timing anomalies with regard to the replacement strategy should be integrated into the hardware model. The most convenient approach to manage the SPM is using static allocation [18] but dynamic SPM allocation is more efficient (it can use profile-based optimization but multiple strategies exist). Analyzing dynamic strategies is challenging, especially the software implemented ones that give optimal allocation for the average execution time. Some WCET-centric techniques exist but they do not handle all architectures.

3.6 Memory Management Unit (MMU) and Translation Lookaside Buffer

The TLB is a cache that MMU use to improve virtual addresses translation speeds. The time needed to determine the physical address depends on the number of performed operations. TLB time access is variable. In order to enforce the predictability, the MMU can be deactivated (however the performance loss is significant) or by reducing the size and thus complexity of the TLB (the TLB main entries can be blocked in order to ensure their persistence). A solution is to increase the TLB size so that we only have hits but we still have the problem of an error in the translation that is detected late and takes an undefined (even if still reasonable) time to be corrected. Typical user manuals [1] give upper bounds in the TLB miss case. Timing anomalies invalidate the monotonicity assumption in the general case [27], which means that we cannot directly use the upper-bound information as the worst-case scenario. Therefore, without precise information on the exact behavior all possible cases must analyzed, leading to a potential state space explosion. In order to reduce the potential temporal variability, the MMU should be disabled. Nevertheless due to the consequences on the global performances it is not recommendable.

In general, virtual memory raises predictability issues at two levels. First at the level of address translation that provides mapping between virtual to physical pages requires a TLB lookup. If the mapping is absent from the TLB a page table lookup is performed. The duration of address translation is hard-to-predict, because not all mappings can be stored in the limited capacity of the TLB or because the TLB might be shared between concurrent processes. Second at the level of paging activity as knowing whether or not a reference to a virtual page will result in a page fault. This is hard to predict because physical memory is shared between concurrent processes.

Impact on the predictability

The virtual addressing and tasks using the shared cache the MMU has a global impact. In the case of multi-cores it is problematic to ensure the micro-TLB coherency. Choosing to handle the TLBs separately introduces new problems of guaranty.

3.7 BUS

As competition for resources grows, the natural solution was to use techniques that enable the access from master to slave, and utilization of shared resources in general. Through the use of switching mechanism, permission is granted to one master or the other, which introduces the need of a bus arbiter. Therefore a controller is usually implemented following different strategies that are more or less straightforward. The first difficulty comes from the implementation of the aforementioned strategies. In the case of multi-cores, the resolve of access conflicts is not always deterministic. Therefore at a given processor execution step, a strongly dataflow dependent transition can be made with no way of determine which of the competing units will have gain access to the shared resources. This behavior, otherwise seen as random, at possibly every program point makes it impossible for the analysis to converge to a useful result.

In the following, we will refer to the AMBA AHB bus protocol, an open standard widely used that give a good case study enabling us to pin-point more general advantages and disadvantages of interconnection protocols. Some of the features it provides, are the following: split transactions, several bus masters, burst transfers, pipelined operations and single-cycle bus master handover. The bus arbiter ensures that only one bus master at a time is allowed to initiate data transfers. The arbiter also receives requests from the slaves that wish to complete SPLIT transfers [3].

Preventing starvation

The arbitration algorithm between the channels can ensure that if the current owner requests the interface again it will always acquire it. The starvation problems are avoided in the LEON 3FT implementation of the AMBA AHB since the DMA engines always deassert their requests between accesses [2].

Preventing deadlocks

The SPLIT and RETRY transfer responses can both produce deadlocks. The deadlock can occur when different masters try to access a slave that issues SPLIT and RETRY responses in a way that the slave is unable to deal with. If a slave issues a RETRY response only one master must access it at a time. More importantly, this constraint is not enforced by the AMBA AHB protocol and should be ensured by the system architecture. According to the GR712RC manual, *cache snooping should always be enabled in SMP systems to maintain data cache coherency between the processors* [2].

On master data concurrency

The bus arbiter of the AMBA AHA can manage up to 16 bus masters. It grants bus access according to the master's priority. The signals used are: HBUSREQx, HLOCKx, HGRANTx, HMASTER[3:0], HMASTLOCK and HSPLIT[15:0] as described in [3]. When a master is granted access, the HGRANTx signal is generated by the arbiter that indicates the appropriate master is currently the highest priority master requesting the bus. After the current transfer completes, the HREADY signal is HIGH and the arbiter will change the HMASTER[0:3] signal to indicate the bus master number. The ownership of the data bus is delayed from the ownership of the address bus. When the HREADY signal is HIGH, the master that owns the address bus will continue to own the address bus until its transfer will be completed. Several problems can occur from this behavior.

(a) When the master is in burst mode, performing bursts of undefined length, it should continue to assert the request until it has started the last transfer. A problem occurs if the arbiter cannot predict when to change the arbitration at the end of an undefined length burst, leading to the impossibility to accurately determine the timing of the transfer. This is what happens in our case study also.

(b) A different behavior can lead to a data inconsistency. Using a central multiplexer, each potential master present on the bus can drive out the address of the transfer immediately without having to wait to be granted the bus.
Let $\text{HADDR}_{M_1} = addr_1$ at clk_1 and $\text{HADDR}_{M_1} = addr_2$ at clk_2 and the first master, M_1 be granted master at clk_1. If $\text{HADDR}_{M_2} = addr_1$ at clk_2, the data at $addr_1$ is still unmodified but it will be as M_1 still owns the data bus which leads to data access consistency conflict at $addr_1$. The time needed to resolving such a conflict is hard to estimate. A typical example is the case of sharing un-partitioned memory. In order to avoid this case, a strong coupling is recommended between the monitoring application and the command application, by implementing the communication though semaphores.

(c) A case where the timing is difficult to compute is when both M_1 and M_2 drive out the same $addr_1$ on the same slave, M_1 is granted ownership of the bus and then it is stalled by the arbiter that grants the bus ownership to M_2. If M_2 needs to access $addr_1$ we cannot precisely determine the time when M_2 will finish its action.
For example, this can occur in shared un-partitioned or partitioned memory.

Impact on the predictability

The bus and its arbitration strategy are at the core of the predictability and determinism issues. Because the main role is to grant access to different participants to the shared resources, certain properties must be ensured (fairness, deadlocks prevention) while still being able to ensure good average performances and timing predictability. Therefore when choosing a particular architecture for the hard real-time systems, certain bus architectures and arbitration algorithms should be privileged. Most of the multi-core architectures implement round-robin-like arbiters which allows considering an upper bound on the latency of the access to the shared resources. In [19] a round-robin-like bus arbiter to the shared memory hierarchy in a multi-core architecture is proposed that facilitates the systems predictability. The round robin arbiter can avoid the bus starvation. Nevertheless, the maximum length of a burst for each peripheral connected to the bus influences the maximum delay induced by bus contention. This may lead to high maximum delay bounds and may not be enough to provide firm real time guarantees for heavily loaded systems. Furthermore, having variable burst lengths, combined with the ability to pause them (split transfers), influences the predictability of the WCET. Bus contention can be avoided by using the TDMA bus arbitration with the cost of wasting bus band when the bus load is low. By manipulating the TDMA time slots, the maximum delay bound on transactions is controllable by the designer.

3.8 Direct Memory Access (DMA)

DMA allows access to the system memory independently from the CPU. Therefore the processor can proceed with its computations while waiting for relatively slow input/output data transfer. In the multi-core case, DMA is also used for intra-chip data transfer.

Error handling

The DMA controller does not generally detect deadlocks in its communication channels, so it is up to the system to manually abort the DMA transfer. The DMA unit can be disabled not without a strong impact on the performances of certain class of applications.

3.9 Level 2 cache

Level 2 cache memory can be either private to each core or shared among cores. Data hazard is one of the factors that become even more prominent in the case of multi-cores because of memory sharing, even though already present in the non-shared L1 memory. Moreover, timing anomalies render the result hard to predict like in the case when a cache miss from a core can reconfigure the memory in a state that is, timing wise, beneficial to the other. This also applies in other cache related scenarios. The problems that can occur in the analysis of the interactions with the pipeline are detailed in the timing anomalies part.

Level 2 cache impact on the predictability

The impact of the shared cache is high and global and gets amplified in the context switch case. Modeling the behavior of shared caches between cores is practically impossible because of the possible interactions between concurrent threads running on different cores [23]. When using a shared cache with parallel programs running on the multi-core processor, a cache-coherency mechanism must be implemented. The WCET analysis of such systems must calculate the worst-case delay caused by maintaining the cache coherence between different cores. Furthermore, resource contention and inter-thread conflicts among the program threads should be considered. Under the assumption that the bus strategy can be statically analyzed, the second level of cache can be made predictable by partitioning the L2 cache for each core [24].

3.10 Timing Anomalies

Timing anomalies inside a given core influence the WCET estimation of integrating multi-cores in the case of shard memory because of the tight coupling of its internal units and the shared resources. Therefore we cannot ignore the WCET estimation problems of single cores as they are translated into the multi-core case also.

Example of timing anomalies in multi-cores A timing anomaly occurs when a local worst case contributes to the global favorable case. In the case of multi-core, such an example is a cache miss on one core that generates a series of cache hits on the other and vice versa. An example of each of these timing anomalies is given. The architectural setup is configured of two cores with private L1 cache memory but shared L2 cache memory (instruction and data).

a A cache miss on one core can generate an overall improvement of the global WCET. Let l be the cache line that will replace the obsolete line in the cache according to the implemented strategy. If l contains information that will benefit the second core then it can generate several cache hits that were not predicted. Furthermore, this line will be promoted in the priority hierarchy and could furthermore avoid future misses of the first core.

b A cache hit on the first core generates the persistence of a cache line in the disadvantage of another that will be replaced after a future cache miss. If the replaced cache line would have generated several cache hits on the second core, the overall timing performance is affected. A first core's cache hit followed by a cache miss is worse than a first core's cache miss followed by a cache miss.

c Timing amplification example as a generalization of b) A series of cache hits on the first core with a higher frequency, then the cache accesses of the second core make that every cache miss of the first core lead to the elimination of the second core's cached lines and a great amount of cache misses.

Timing anomalies remarks

As previously stated, several types of timing anomalies exist. Some are inherent to instruction execution order and are generally caused by greedy scheduler that will change the instruction execution order causing inversion or amplification of the execution time difference. Others are caused by parallel decomposition and divide et impera approaches to WCET estimations. As the first ones cannot be avoided, the others may prove essential for the possibility to construct an efficient processor behaviour analysis that does not need to search the whole state space for the whole program at once. The timing anomalies determine three infeasibility criterions in the following.

Criterion a) Let p be the processor architecture model, we say that the estimation of the WCET or more generally the processor behaviour analysis cannot be completed on behalf of the parallel decomposition ($PD(p)$) if there is no other scalable method that can do the analysis of p without the $PD(p)$. In other words, not applying parallel decomposition can affect the scalability and applicability of the estimation method. We proceed by questioning the safeness and efficiency on dealing with parallel timing anomalies. [16] formalizes the different types of timing anomalies and presents cases when the parallel timing anomalies can lead to the underestimation of the WCET with parallel composition.

Criterion b) The use of parallel decomposition in the processor behaviour analysis leads to the underestimation of the WCET in the case of coupled parallel timing anomalies. This point can be referred to in order to decide the use of parallel decomposition. In [14] a solution to take into account timing anomalies in general is described. The method uses compilation techniques and modifies the binary by instruction injection in order to avoid timing anomalies. The main idea is to interfere with the prefetch stage and ensure that we start with an empty or flushed prefetch window hence there is never an excess instruction waiting to be executed. It can be seen as a compile time method to disable the instruction prefetch in the case where all the slots of the instruction prefetch queue are filled with NOPs. The reference also provides estimation of the overhead when using this method as ranging between 33% and 300%. This leads us to another infeasibility criterion that is related to the maximum overhead allowed for the target platform.

Criterion c) Let $Ot(p)$ be the upper-bound of the overhead on a target platform and $Op(p)$ be the overhead of filling prefetch slots with NOPs. If $Op(p) > Ot(p)$ then the analysis does not pass the feasibility test.

Table 1 Architectural impact on the determinism.

Unit	**Problems / Failure mode**	**Problem frequency**	**Solutions**	**Impact Level**
TLB	TLB misses times are hard to predict	M/ L	Increase the TLB size	M
TLB	TLB error	L	**None => disable**	M/H
MMU	The time to access the tables can take several cycles	M	Depends on the availability of the behavioural model and corresponding timings. Disabling the MMU will only affect performances if we use its features. This means, flat address mapping, no memory protection in the case a process reads/writes the address space of another process and not least, when performing a context switch there is no longer possible to identify the cached lines of a certain process.	**H**
Scratch-pad	Application controlled -> hard to estimate the timings		Analysing dynamic strategies is not an easy task, especially when being software implemented that give optimal allocation for the average execution time. Some WCET-centric techniques exist but they do not handle all architectures.	M
L1 cache	Timing anomalies	H	Construct complex, accurate processor model	**H**
L2 cache	Timing anomalies	H	**Deactivate**	**H**

Table 1 – continued from previous page

Unit	**Problems / Failure mode**	**Problem frequency**	**Solutions**	**Impact Level**
L2 cache	Data conflicts	M	Partition (core access separation) Partial solution by considering inter-thread instruction conflicts [30] Only solutions for instruction caches in some configurations are presented in [11]. [6] Addresses only instruction caches with no code sharing, LRU strategy, no data –instruction memory interference, without timing anomalies, so a very restricted environment. [7] Deals with inter-thread interferences but in a restricted architecture. No details are given concerning the shared resources granting policy or about the BUS context.	H
L2 cache	Unknown behaviour induced by the arbiter	L	**Deactivate**	**H**
L3 cache	Timing anomalies	H	Partition	M/H
L3 cache	Data conflicts	M/L	**Deactivate**	M/H
BUS	Arbitration, timing anomalies, memory interference	H	None for the general case In [6] a very restricted "BUS" is analysed with fully separated code and data accesses and no inter-process communication through shared memory and TDMA based static scheduling where a fixed length bus slot is allocated to each core in a round-robin fashion.	**H**
Arbiter	Nondeterministic		None	**H**
Arbiter	Starvation		Software supervision, but the risk of using it is even higher. DMA engines should always deassert their requests between accesses in order to prevent starvation.	**H**
Arbiter	Deadlocks		Prevent by careful hardware integration of the bus arbiter protocol. Can rarely be disregarded by construction.	**H**
Pipeline	Timing anomalies	H	Construct complex, accurate processor model. A solution to take into account all the timing anomalies (that might prove efficient in the multi-core case) is presented in [20].	**H**
FPU	High complexity		Construct complex, accurate processor model	M/L

Table 1 – continued from previous page

Unit	Problems / Failure mode	Problem frequency	Solutions	Impact Level
FPU	Mixes multi-cycle instructions (for which timing estimation is not always possible) and single cycle instructions		None	M/H
FPU	Can generate computational errors. Timing of exception catching is hard to precisely determine		Must analyse the fault tolerant mechanism. If the behaviour is taken into account at each step, might prove very costly.	M/H
ALU	Can generate computational errors.	L	Construct complex, accurate processor model taking into account the fault tolerance.	M/L
BPU	In the strategy is fixed and the prediction is wrong, the time penalty is very important.	M	Construct complex, accurate processor model	**H**
BPU	If the prediction strategy is not fixed, it can be very difficult to model or even impossible if randomness is used.	L	None	**H**

4 Conclusion

Software verification and quality assurance process of hard real-time systems in general are of great importance. Non-functional properties, such as timing, are highly dependent on the underlying hardware platform. Nevertheless, there is a rising demand to integrate more complex processors, such as the multi-cores, even though many problems are yet to be solved in single-cores. Powerful industrial WCET estimation tools available today can do nothing against the lack of information regarding the exact behavior of the platform or the nondeterministic behavior of certain units. Therefore the choice of the processor is crucial in ensuring the success of the system verification.

We have presented the behavior of several units that pose problems concerning the WCET estimation, found either in multi-cores or single-cores. Each unit description is followed by the problematic behavior and the remarks regarding its impact on the predictability. The results can be used to invalidate certain units or architectures and also as a guideline for further analysis.

References

1. Aeroflex. *UT699 LEON 3FT/SPARCTM V8 MicroProcessor, Functional Manual*, 2012.
2. Aeroflex Gaisler AB. *GR712RC - Dual-Core LEON3FT SPARC V8 Processor, User's Manual*, 2011.
3. ARM. *AMBA Specification (Rev 2)*, 1999.
4. Christoph C., Christian F., Gernot G., Grund D., Maiza C., Reineke J., Triquet B., and Wilhelm R. Predictability considerations in the design of multi-core embedded systems. In *Proceedings of Embedded Real Time Software and Systems*, pages 36–42, May 2010.
5. S. Chattopadhyay and A. Roychoudhury. Scalable and precise refinement of cache timing analysis via model checking. In *Proceedings of the 2011 IEEE 32nd RTSS*, RTSS'11, 2011.
6. S. Chattopadhyay, A. Roychoudhury, and T. Mitra. Modeling shared cache and bus in multi-cores for timing analysis. In *Proceedings of the 13th International Workshop on Software 38; Compilers for Embedded Systems*, SCOPES'10, pages 6:1–6:10, New York, NY, USA, 2010. ACM.
7. F. Chen, D. Zhang, and Z. Wang. Characterizing the inter-thread interference of multi-core architectures for accurate wcet estimations of real-time applications. In *Przeglad Elektrotechniczny*, 2012.
8. N. Drach, A. Seznec, and D. Windheiser. Direct-mapped versus set-associative pipelined caches. In *Proceedings of the IFIP WG10.3 working conference on Parallel architectures and compilation techniques*, PACT 95, 1995.
9. D. Hardy, T. Piquet, and I. Puaut. Using bypass to tighten WCET estimates for multi-core processors with shared instruction caches. In *Proceedings of the 2009 30th IEEE Real-Time Systems Symposium*, RTSS'09, pages 68–77, Washington, DC, USA, 2009. IEEE Computer Society.
10. D. Hardy and I. Puaut. WCET analysis of multi-level non-inclusive set-associative instruction caches. In *Proceedings of the 2008 Real-Time Systems Symposium*, RTSS'08, pages 456–466, Washington, DC, USA, 2008. IEEE Computer Society.
11. D. Hardy and I. Puaut. Estimation of cache related migration delays for multi-core processors with shared instruction caches. In Laurent George and Maryline Chetto andMikael Sjodin, editors, *17th International Conference on RTNS*, pages 45–54, Paris, France, 2009.
12. R. Heckmann, M. Langenbach, S. Thesing, and R. Wilhelm. The influence of processor architecture on the design and the results of WCET tools. *Proceedings of the IEEE*, 91(7):1038 – 1054, july 2003.
13. International Electrotechnical Commission. *IEC 61508 Functional safety of electrical/electronic/programmable electronic safety-related systems*, 2010.
14. A. Kadlec, R. Kirner, and P. Puschner. Avoiding timing anomalies using code transformations. In *Proc. 13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, pages 123–132, May. 2010.
15. T. Kelter, H. Falk, P. Marwedel, S. Chattopadhyay, and A. Roychoudhury. Bus-aware multicore WCET analysis through TDMA offset bounds. In *Proceedings of the 2011 23rd Euromicro Conference on Real-Time Systems*, ECRTS'11, pages 3–12, Washington, DC, USA, 2011. IEEE Computer Society.
16. R. Kirner, A. Kadlec, and P. Puschner. Precise worst-case execution time analysis for processors with timing anomalies. In *Real-Time Systems, 2009. ECRTS'09. 21st Euromicro Conference on*, pages 119–128, July.
17. Y. Liang, H. Ding, T. Mitra, A. Roychoudhury, Y. Li, and V. Suhendra. Timing analysis of concurrent programs running on shared cache multi-cores. *Real-Time Syst.*, 48(6):638–680, November 2012.

18 P. Panda, N. Dutt, and A. Nicolau. Efficient utilization of scratch-pad memory in embedded processor applications. In *Proceedings of the 1997 European conference on Design and Test*, EDTC'97, pages 7–, Washington, DC, USA, 1997. IEEE Computer Society.

19 M. Paolieri, E. Quiñones, F. Cazorla, G. Bernat, and M. Valero. Hardware support for WCET analysis of hard real-time multicore systems. *SIGARCH Comput. Archit. News*, 37(3):57–68, June 2009.

20 V. A. Paun and B. Monsuez. Adaptable and precise worst case execution time estimation tool. In *LCTES 2012 Work-in-Progress Proceedings*, LCTES'12, 2012.

21 Radio Technical Commission for Aeronautics. *DO-178B Software Considerations in Airborne Systems and Equipment Certification.*

22 J. Reineke, D. Grund, C. Berg, and R. Wilhelm. Timing predictability of cache replacement policies. *Real-Time Syst.*, 37(2):99–122, November 2007.

23 M. Schoeberl. Time-predictable cache organization. In *Proceedings of the First International Workshop on Software Technologies for Future Dependable Distributed Systems (STFSSD 2009)*, pages 11–16. IEEE Computer Society, 2009.

24 M. Schoeberl, B. Huber, and W. Puffitsch. Data cache organization for accurate timing analysis. *Real-Time Systems*, DOI: 10.1007/s11241-012-9159-8:1–28, 2012.

25 SPARC International Inc. *SPARC V8 architecture manual, Revision SAV080SI9308*, 1992.

26 Xavier Vera, Björn Lisper, and Jingling Xue. Data caches in multitasking hard real-time systems. In *Proceedings of the 24th IEEE International Real-Time Systems Symposium*, RTSS'03, pages 154–, Washington, DC, USA, 2003. IEEE Computer Society.

27 I. Wenzel, R. Kirner, P. Puschner, and B. Rieder. Principles of timing anomalies in superscalar processors. In *Quality Software, 2005. (QSIC 2005). Fifth International Conference on*, pages 295 – 303, sept. 2005.

28 M. Wolf and M. Lam. A data locality optimizing algorithm. In *Proceedings of the ACM SIGPLAN 1991 conference on Programming language design and implementation*, PLDI'91, pages 30–44, New York, NY, USA, 1991. ACM.

29 J. Yan and W. Zhang. Hybrid multi-core architecture for boosting single-threaded performance. *SIGARCH Comput. Archit. News*, 35(1):141–148, March 2007.

30 J. Yan and W. Zhang. WCET analysis for multi-core processors with shared L2 instruction caches. In *Real-Time and Embedded Technology and Applications Symposium, 2008. RTAS'08. IEEE*, pages 80 –89, april 2008.

31 J. Yan and W. Zhang. Accurately estimating worst-case execution time for multi-core processors with shared direct-mapped instruction caches. In *15th IEEE International Conference RTCSA'09*, 2009.