WCET Analysis Methods: Pitfalls and Challenges on their Trustworthiness

Jaume Abella[†], Carles Hernandez[†], Eduardo Quiñones[†], Francisco J. Cazorla[†,‡],
Philippa Ryan Conmy[*], Mikel Azkarate-askasua[⋆], Jon Perez[⋆], Enrico Mezzetti[ϕ], Tullio Vardanega[ϕ]

[†]Barcelona Supercomputing Center [‡]Spanish National Research Council (IIIA-CSIC)
[*]Rapita Systems Ltd. [⋆]Embedded Systems Group, IK4-Ikerlan [ϕ]University of Padua

***Abstract*—In the last three decades a number of methods have been devised to find upper-bounds for the execution time of critical tasks in time-critical systems. Most of such methods aim to compute Worst-Case Execution Time (WCET) estimates, which can be used as trustworthy upper-bounds for the execution time that the analysed programs will ever take during operation. The range of analysis approaches used include static, measurement-based and probabilistic methods, as well as hybrid combinations of them. Each of those approaches delivers its results on the assumption that certain hypotheses hold on the timing behaviour of the system as well that the user is able to provide the needed input information.**

Often enough the trustworthiness of those methods is only adjudged on the basis of the soundness of the method itself. However, trustworthiness rests a great deal also on the viability of the assumptions that the method makes on the system and on the user's ability, and on the extent to which those assumptions hold in practice. This paper discusses the hypotheses on which the major state-of-the-art timing analyses methods rely, identifying pitfalls and challenges that cause uncertainty and reduce confidence on the computed WCET estimates. While identifying weaknesses, this paper does not wish to discredit any method but rather to increase awareness on their limitations and enable an informed selection of the technique that best fits the user needs.

I. INTRODUCTION

Determining the Worst-Case Execution Time (WCET) of software programs running in time-critical systems (with criticality related to safety, availability, security, mission or business needs) is a known challenge. A host of methods have been devised to date to that end, each asserting trustworthiness on the basis of certain assumptions on the timing behaviour of the processor hardware (e.g. known placement policies in cache), the software programs (e.g. known flow facts), and their run-time environment. However, for each distinct application domain and for the particular system of interest, the preservation and observance of those assumptions may be overly difficult to attain, if at all possible. Industrial users are therefore confronted with the daunting challenge of arriving at the computation of WCET estimates with the domain-specific degree of trustworthiness and within possibly strict cost and effort constraints.

In each domain, a certain degree of rigour and depth for analysis is imposed by safety-related standards such as ISO26262 (automotive) [18], DO-178B/C (avionics software guidance) [41], IEC61508 (generic electronic systems) [17], EN50128 (Railway) [7], etc.; however, in many cases industry adopts additional safety measures beyond those in the standards so as to have additional *confidence* on the safety of the systems delivered. How to quantify such confidence is a hard – if at all doable – step. Interestingly, WCET analysis is not directly related to safety: how a timing failure (i.e. overrunning a WCET) affects safety is determined by complex reasoning at system level. WCET estimates have to be *trustworthy* for the conclusions of that reasoning to be relied upon as well for sustaining the cost of the relevant mitigation measures. Industry-viable timing validation of safety-related systems is, therefore, a complex process with a number of constraints, all of which need to be balanced when performing WCET analysis and further to help claim that the timing of the overall system meets the system safety requirements.

In this paper we focus on the process followed to obtain WCET estimates once the hardware and software to be analysed are available, leaving Worst-Case Response Time analysis out of this discussion. We focus on a number of methods to timing analysis including static, measurement-based and hybrid techniques following a deterministic and a probabilistic approach [48], [5], [9]. For each method we identify the main elements and steps which determine the confidence that can be placed on it. Some steps can be taken in full trust whereas others are challenging. It is often the case that the user neglects to ascertain the trustworthiness of some of those steps, therefore lapsing into undue claims. For instance, steps like the estimation of loop bounds and relevant execution paths may be performed manually or, in the best case, semi-automatically, which fails on the side of trustworthiness. Indeed, the trustworthiness of the whole process cannot be higher than that of those steps unless suitable countermeasures are taken. In summary, our contributions are as follows:

1) For static timing analysis methods we carefully review the steps in the design and development of a hardware device (e.g. a processor) and how its associated documentation is contributed in each step. Documentation on processor-related timing feeds the models built on top of that information, therefore becoming a central factor to confidence and trust.
2) We review how measurements are collected and the inputs/conditions exercised in the measurement runs, and then how WCET estimates are derived based on that information. We relate those aspects to confidence, cost and quality of the WCET estimates (hence of their trustworthiness).
3) For hybrid analysis, we examine how and what it takes from static and measurement-based approaches in the intent of drawing the best from both worlds.

While some of these findings have already been identified for some of the timing analysis techniques in the past [30], to the best of our knowledge a comprehensive analysis of the limitations challenging the different timing analyses has not been provided so far. This paper aims at covering this gap, also considering the new challenges brought forward by multicore processor architectures.

TABLE I
WCET TECHNIQUES TAXONOMY

	Deterministic	Probabilistic
Static	SDTA [8], [26], [32], [48]	SPTA [5], [2]
Measurement-based	MBDTA [48], [31]	MBPTA [15], [9], [6]
Hybrid	HYDTA [38], [47]	HYPTA [25]

The conclusion we arrive at is that none of the surveyed methods is fully trustworthy on all accounts. Still, we identify scenarios where their application is intended to provide higher confidence on the WCET estimates obtained.

The rest of the paper is organised as follows. Section II reviews the set of timing analyses considered and the criteria against which they are assessed. We discuss the case of the deterministic and probabilistic flavors of static, measurement-based and hybrid timing analysis in Sections III-VIII. A summary is provided in Section IX. Finally, Section X concludes this paper.

II. TECHNIQUES AND ASSESSMENT CRITERIA

In this section we review the particular timing analysis techniques we evaluate and the criteria we use.

A. WCET Techniques

In our taxonomy we split WCET analysis techniques into two categories: deterministic and probabilistic. The former category applies to systems whose hardware and software resources, for a given input and initial state, provide an output in a defined time on those systems; deterministic timing analysis (DTA) is used to provide a single WCET estimate. Probabilistic timing analysis (PTA) applies to resources with either time-randomised or time-deterministic behaviour. PTA techniques compute a distribution function called probabilistic WCET (pWCET) whose tail can be cut at a probability of exceedance sufficiently low to make a negligible contribution to the overall probability of failure of the system according to the applicable safety standard.

For each approach we have three variants: static, measurement-based, and hybrid, for a total of 6 variants (cf. Table I). The specific characteristics of static, measurement-based and hybrid approaches are described later.

B. Criteria

We use the following set of criteria to assess each of the methods for practicality and tightness:

Trustworthiness and tightness - we must have some defined confidence that the WCET estimate produced is above and sufficiently close to that which would be observed during the operational life of the system under test[1].

Certifiability - the approach must be compatible with certification practice in the different domains. This leads to two sub-criteria: Tool qualification - the tools used to support the WCET analysis must be of sufficient quality for their output to be trusted. Safety justifications for any additional manual effort used in the process, e.g. to exclude some execution paths.

Cost effectiveness - the cost and effort used to apply the WCET method must be within practical bounds, e.g. for As Low As Reasonably Practicable (ALARP).

Good implementation practices - The soundness of each of the methods depends on whether their implementation is correct. While this is a critically important concern, it affects all methods in a similar manner. Further, since their implementations have been successfully assessed against safety standards in the past, we omit this concern in the rest of the paper.

Target application scalability - All methods rely on the use of good programming practices for the applications to be analysed, since this is already needed for functional verification. Analogously, the *size* of the application under analysis is not deemed as a challenge per se since its parts can be analysed separately if too large. Instead, we dig into the challenges related to the collection of flow facts and test input generation, which are concerns orders of magnitude above the application size (e.g., number of lines of code).

III. STATIC DETERMINISTIC TIMING ANALYSIS

Within the class of deterministic approaches, static WCET analysis techniques (SDTA) derive WCET bounds for a given program on a target platform without resorting to program execution, by combining the results from two distinct models: an abstract model of the hardware and a structural representation of the program under analysis. Each of such models typically corresponds to a different analysis scope in SDTA methods:

- *Low-level analysis:* where a timing model of the target architecture is constructed, by attaching a precise functional and temporal behaviour to each hardware component (e.g., cache hierarchy, pipelines, buses, memory controllers, etc.).
- *High-level data flow or path analysis*: where a representation of the program under analysis is used as a structure on which the information from the low-level timing model is combined with path information to derive a WCET estimate.

Strong emphasis in SDTA approaches is put on the *safeness* of results [48], as determined by the application of provably sound static analysis steps. SDTA approaches consider all possible inputs (values and states) for a program: the search space is kept within a tractable dimension only by using safe abstractions for both hardware (e.g., abstract hardware states) and software (e.g., contexts of execution). Tightness of the produced WCET estimates is also important, but it is degraded to a subsidiary objective that cannot always be had, due to the conservative approach inherent to the analysis steps.

Arguing on the safeness of SDTA analysis results might seem, in the common acceptation of the term, a stronger claim than trustworthiness, in that a safe bound is by definition infallible. However, the safeness guaranteed by static analysis methods is only a product of the results of their analysis steps. Despite the precision of SDTA methods, if the assumptions or inputs to these analysis steps are flawed then the results are inaccurate. Unfortunately, both low- and high-level analysis steps are somehow exposed to weaknesses that may jeopardise the accuracy of WCET results.

In the following we highlight the weaknesses of SDTA assumptions, thus showing that full confidence cannot be always attained, even in case of precise SDTA techniques.

A. Accuracy of Low-level Analysis

An accurate characterisation of the timing behaviour of the hardware platform is fundamental in the determination of the WCET behaviour of each single instruction. Needless to say that novel hardware architectures and increasingly advanced

---

[1] Although trustworthiness and tightness are different concepts, we consider them together because typically timing analysis techniques trade them off, and WCET estimates need to attain both (to some extent) simultaneously.

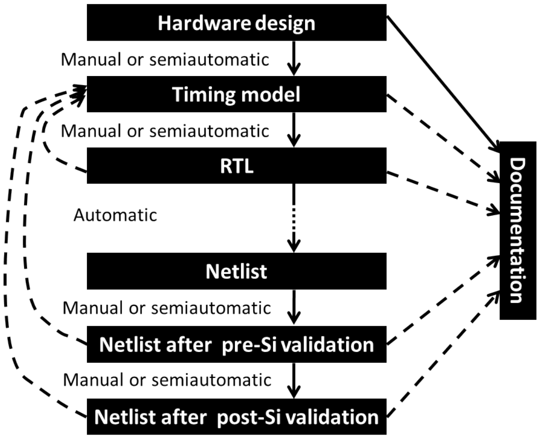


Fig. 1. Steps in hardware design and their impact on documentation and TM.

hardware speed-up features complicate this low-level analysis step. Timing models (TM) for a specific hardware can be, in the most favourable cases, provided by the hardware vendor itself or derived a-posteriori from the available technical specification. In both cases, the TM may not always report the actual timing behaviour of the hardware (or an upper-bound) due to possible inaccuracies introduced along the typical hardware development process as well as due to intentionally limited disclosure.

A.1. Vendor's TMs and technical specifications

The typical hardware design flow is briefly summarised in Figure 1. It moves from the definition of a highly accurate TM and its (only in part automatically generated) representation as a Register Transfer Level (RTL) description. Then the RTL code can be automatically translated to a gate level design and a netlist, which already accounts for the electrical effects of the physical implementation of the circuit such as resistances and capacitances. Additionally, a detailed documentation is produced and updated within each development phase. Some inaccuracies and inconsistencies may be injected at certain critical points in the process:

- *TM definition and RTL description:* some inconsistencies may arise between the TM, the RTL and the documentation. Hardware designers in charge of developing the TM are often in charge of producing the documentation describing the timing behaviour of the design. Based on our first-hand experience on hardware design, we know that those engineers, who are highly skilled in hardware design, tend to consider documentation as a painful process, to be done only once the design is complete. This may lead to incomplete or inaccurate documentation that does not reflect the actual timing behaviour of the hardware. Moreover, as RTL must be fully unambiguous, some design choices affecting timing may be taken during the TM-to-RTL translation and never transferred back to the documentation, especially when the TM-to-RTL translation is not performed by the same hardware designers.
- *Pre-Silicon Validation:* the RTL hardware description is typically translated into a netlist, corresponding to the actual layout of the circuit to be created. Before actually fabricating circuits, a pre-Silicon validation step is performed to check the actual implementation of the circuit against the timing, power, temperature and area constraints placed on the product. Whenever some constraints are not met – and it is often so – modifications are performed directly into the RTL, gate implementation or netlist iteratively, until every circuit constraint is satisfied. Eventually, this may lead to some changes into the timing behaviour of the hardware, such as pipelining some circuits to reduce the cycle time (and increase operating frequency) or adding some constraints to save power (e.g., preventing the issue of some instructions in particular cycles to allow for the sharing of a register file or cache port). Again, this leads to an increased risk of undocumented changes and mismatches with the TM.
- *Post-Silicon Validation:* once validated, the netlist is regarded as final and it is fabricated and tested. Such test may expose further issues and unmet constraints. At this point, for example, conflictive layout patterns are typically detected. Those patterns may be fully correct, but lead to high process variations due to fabrication limitations, which translate into frequent failures and thus, low yield – many chips are faulty. Similarly to pre-Silicon validation, modifications to the design may not be captured in the documentation or in the TM.

An additional source of complexity is to be identified in timing interactions among different hardware components, typically produced by different vendors. Determining the timing interaction of different components (e.g., multiple CPUs, memory devices, I/O devices, etc.) with sufficient accuracy at system integration is a cumbersome and challenging task. Furthermore, inaccuracies on a component documentation are inadvertently passed on into the documentation of the final integrated hardware platform.

A.2. Deriving a TM from the technical specifications

Besides the risk of potential mismatches between the hardware and its TM, we need to consider that a TM typically contains highly-confidential information that vendors may not want to fully disclose, in order not to give away competitive advantage. In this case, to meet SDTA prerequisites, a TM is to be derived from the available documentation. Unfortunately, such documentation is not guaranteed to be either accurate or complete. Moreover, the construction of a new TM from scratch is a complex and error-prone process, especially given that documentation for many embedded processors is in the order of some thousands of pages. For instance, some parts of the Freescale P4080 specification have 2,000 pages [12]. Similarly, the Infineon XMC4500 microcontroller documentation has more than 2,500 pages [16].

A.3. Trustworthiness of TMs and technical specifications

A number of steps in the hardware design process expose to a large number of errors and inaccuracies in the hardware TM as well as in its technical documentation, which instils some doubts about the trustworthiness of any analysis relying on them. The massive presence of *errata* documents for modern processors confirms that relying on the accuracy of product specifications may be a hazard. For instance, the FreeScale e500mc core documentation has already reached the third revision. Details about the non-negligible changes across revisions can be found in [13]. Similarly, processors such as the ARM Cortex R5 – specifically devised for real-time systems – have abundant errata in their documentation [3] despite being relatively simple (i.e. its technical reference manual has around 450 pages). These errata affect also timing. For instance, one can read in revision r1p2 that stall cycles caused by the divider and the latencies of some operations (VDIV.F64 and VSQRT.F64) needed to be updated in the documentation.

B. Accuracy of High-level Analysis

SDTA derives WCET bounds on a program without executing it. Among several possible program representations, the control flow graph (CFG) is the most popular one in current state-of-the-art approaches. CFG-based approaches allow to translate the program structure into a set of flow constraints according to the implicit path enumeration technique (IPET) [27] WCET computation method.

Several static analysis techniques are applied to the structural representation of a program to derive information on the program flow, typically referred to as *flow facts* [22]. Whereas some flow facts can be statically determined or automatically extracted, for example, by the compiler, not all program properties are decidable, owing to the halting problem. In these cases, the user is required to provide flow facts in the form of manual annotations. Exemplifying properties, in this sense, are the maximum number of iterations for a loop, or the targets of dynamic jumps and calls. Although some simple occurrences of these problems can be automatically resolved by state-of-the-art static analysis methods (typically relying on value analysis or pattern-based techniques) user annotations are generally required to reconstruct the program control flow. Flow facts, either automatically determined or manually annotated, are also required to shave pessimism off the computed WCET (e.g., by excluding infeasible paths).

In any case, high-level path analyses strongly rely on the availability and correctness of such flow facts. In case of manual annotations, providing accurate flow facts is an error-prone task that requires a deep understanding of the program behaviour. Similarly to the low-level analysis step, inaccurate flow facts could definitely compromise the quality of the analysis results. The way flow-fact information is collected and handled within the development process may introduce some inaccuracies, which in turn may affect the correctness of the provided annotations.

B.1. Collection of flow-fact information

Making effective manual annotations requires specialised skills and deep knowledge of the program under analysis. This knowledge, obviously owned by the system designers and software developers, may be unavailable when timing analysis is performed and difficult to retrieve, either because SDTA is not performed by the same people or because the program includes black-box external components or libraries. Lost flow information can be reconstructed by (a combination of) different methods, none of which is completely flawless.

- *Code inspection*: flow facts can be derived by manual inspection of the program code. Whereas this could be reasonable for simple flow facts, it may result to be extremely complex or even unfeasible for non-trivial programs [30]. In general, code inspection alone cannot be considered fully reliable as its effectiveness is a function of the skills and effort put in it.
- *Empirical observations*: flow facts may be also derived from program observations, thus with actual runs of the program. Under this approach, however, SDTA inherits the same limitations as measurement-based methods, which we discuss in the next sections.

B.2. Scope of flow-fact information

Flow information and the respective manual annotations are typically defined at the source-code level where the program logic is much more readable. However, SDTA does not operate directly at the source level since the timing behaviour is actually determined by the low-level object code, as produced by the compilation process. The gap between source- and object-level is determined by the code generation engine of compiler back-ends which, depending on the optimisation policy, reorganises the control flow, and introduces loops and branches not traceable back to the source code [26]. Besides the fact that some flow information may be only available at the object-code level, a translation mechanism is required to map source-level annotations into the object-level code constructs. An automatic translation may not always be possible and the user may be required to reformulate or express new annotations directly at the object-code level. Flow facts also need to conform syntactically to the static analysis tool, which may introduce further inaccuracies.

B.3 Fragility of flow-fact information

The circumstance that flow facts are effective at object-code level makes annotations inherently fragile: they are only valid under the exact and specific conditions they were defined for. Both automatically produced or manually defined annotations use addresses and offsets that are very specific to the executable under analysis. For example, the correctness of an annotation on the target of a dynamic call depends on the fact that the addresses identified at analysis time stay exactly the same in operation.

C. Certification

Static analysis techniques are advocated in the software verification stage of industrial safety standards [37]. When WCET analysis is recommended [37], SDTA can in principle meet all the requirements set by the standards, particularly for the highest integrity levels [41], [17]. The mathematical foundations of SDTA, in fact, provide a strong basis for building solid certification arguments.

However, besides the discussed issues in term of trustworthiness, SDTA cannot always yield satisfactory results in terms of cost-efficiency and value tightness. Thus, the cost of applying SDTA can only be afforded by the average user for comparatively small programs running on comparatively simple hardware [30].

D. Multicore Considerations

Timing analysis for multicore systems becomes much more complicated in consideration of the effects of sharing hardware resources among tasks running on different cores: complex and detailed information on the functional and timing behaviour of the processor hardware is required to precisely account for the contention incurred by accessing shared resources. SDTA approaches could then be extended to include the interference from shared caches and buses, for example, by jointly computing a safe approximation of the incurred delay [49], [8] (i.e. computing the WCET for all tasks in all cores simultaneously). The assumptions of joint static WCET analysis approaches, however, are questionable as explained in [31] due to the complexity of task sets and timing variations (e.g., tasks in other cores may easily change or align differently in time depending on whether previous tasks execute faster or slower).

Although in principle a worst-case impact of contention, or Upper Bound Delay (*UBD*), could be defined for each shared hardware resource, its actual computation is hampered by the increasingly complex multicore processors adopted in safety-critical real-time domains, and the limited information available on their internal functioning. This is somehow confirmed, for example, by a collaborative study between an avionics end-user and a SDTA tool provider [32], where the maximum memory contention on a P4080 processor is determined on the basis of *empirical* UBD values, instead of analytic bounds.

SDTA has only been proven practical on top of specific hardware designs with simple cores where time and space partitioning are strictly applied for resource sharing [40], [21], thus leading to low average and guaranteed performance [19]. These design choices are against the adoption of multicores by creating almost-federated architectures inside the chip.

E. Takeaway Message

The assumptions made by SDTA are not guaranteed to hold in all circumstances. The risk of inaccurate TM or technical specification, and unreliable manual annotations cannot be ignored when reasoning on the trustworthiness of SDTA. Although SDTA relies on sound mathematical abstractions, unattended assumptions may threaten the correctness of the whole approach: for SDTA this might question even the safeness of WCET bounds.

This notwithstanding, SDTA remains an industrially-viable option for timing analysis [44], especially when the aforementioned implicit assumptions are guaranteed to hold:

- *Simple hardware design*: simple, consolidated (and debugged) hardware designs are expected to exhibit high-quality documentation and accurate TMs. Arguably however, such hardware may be incapable of providing high average and guaranteed performance. This might require the deployment of a larger number of hardware systems, with higher costs, power consumption and payload.
- *Simple software*: for simple-enough program fragments, the cost and risk incurred in the collection of flow information is very low, where not zeroed by full automation.

Whereas those favourable conditions are still realistic, the current industrial scenario is witnessing a substantial grow in the number and complexity of safety-related functions to meet the increasing expectations on higher value and increased reliability in different domains. Unmanned vehicles in the avionics domain or x-by-wire technology in the automotive domain critically rely on complex software running, with exponentially growing code size [10], [42], on top of high performance hardware platforms. In practice, under these challenging conditions, SDTA might not be able to deal with the totality of time-critical functions.

IV. MEASUREMENT-BASED DETERMINISTIC TIMING ANALYSIS

Measurement-based deterministic timing analyses (MBDTA) derive WCET estimates by collecting measurements (mostly execution time measurements) on top of the actual hardware platform and operate on those measurements. While measurements themselves can be deemed trustworthy as they provide real data (thus are exempt from TM or documentation flaws), a number of issues challenge the trustworthiness of MBDTA. Among those we identify the following ones:

1) Test conditions. Measurements need to be collected in a platform *identical* to those used at deployment.
2) Test inputs. Inputs used to collect measurements must – ideally – include the one leading to the WCET.
3) Measurement collection. Accurate measurements of the execution time, among other metrics, are needed.
4) Operation on the measurements. Given a set of measurements, how to determine the WCET estimate requires some guidance.

In the rest of this section we analyse those issues and how they challenge timing analysis trustworthiness.

A. Test Conditions

Reproducing during the platform-test phase the execution conditions that will occur during deployment is a complex task. First, very likely this cannot be done until all hardware and software components have been developed, which may occur too late to react if WCET estimates fail to satisfy the required deadlines. If those measurements are collected with an incomplete platform (e.g., while other software components are being developed), it may be complex guaranteeing that missing components will not alter the timing of the software under analysis. For instance, memory placement of objects has been deemed as a critical factor affecting execution time in the presence of cache memories as it determines how different addresses compete for cache space [29], [36]. Even if those addresses can be fixed so that side-effects across tasks can be avoided, this is not the case with stateful OS services, whose execution time may depend on the execution history of the whole system. Depending on how the OS is designed, OS services may use a different set of addresses depending on past history, thus leading to different cache patterns and so, different execution times when running the task with and without other software components of the system.

Test conditions also include the modifications made to the system for collecting measurements. This practice may be intrusive if extra instructions need to be executed or if tracing devices alter the hardware timing in any significant way.

B. Test Inputs

Typically, tasks have an inordinately large input space so that all potential inputs cannot be tested to derive a WCET estimate. Therefore, a method is required to determine a set of inputs so that their execution can be performed in an affordable time while providing useful data for the estimation of the WCET. Ideally, one might want to execute the task with the input leading to the highest execution time. However, determining such input analytically is often beyond the user means, especially if the program under analysis is legacy code developed by someone else. In fact, even if one can determine the execution path leading to the highest execution time, producing an input exercising that path can be regarded as a very complex task.

In this scenario, users face a complex challenge. Typically, the only inputs available are those used for functional testing, which may not produce the highest execution time. Therefore, users may end up using those inputs and, in the best case, devise some *stressful* inputs with the hope they will deliver execution time measurements close enough to the real WCET. Unfortunately, the existence of hardware features devised for

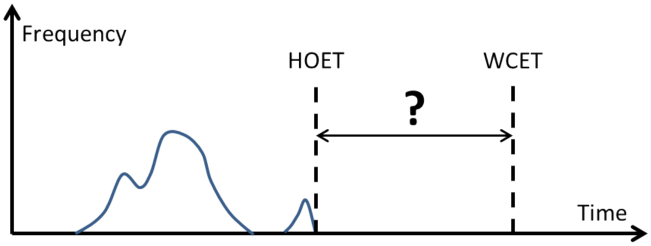


Fig. 2. Determination of the WCET estimate in MBDTA.

high average performance typically leads to execution times that may vary significantly under minor modifications in the inputs, as cache-related effects can introduce execution time variations in the order of one or more orders of magnitude [36].

Overall, input data used to collect measurements can produce *high* execution times, but it is hard to determine how close are those to the actual WCET. In this scenario the degree of trustworthiness attained strongly depends on the skills of the user to understand the timing behaviour of the software when running on the particular hardware platform.

C. Measurement Collection

MBDTA needs to collect some execution time measurements. However, accurately counting the number of cycles a program run takes is a complex problem. Hardware performance monitoring counters (PMCs) can be used to that end, as they allow counting execution time at the right granularity. However, PMCs are typically accessed through some specialised API, whose inherent latency makes it very unlikely to start counting cycles right when the program under analysis starts executing and stop counting right when it completes. Thus, intrinsic inaccuracy pollutes those measurements.

The degree of such inaccuracy as well as whether it may lead to under-measurements or only to over-measurements will depend on the characteristics of the API and the hardware, and how they are used. Further, it is to be considered how such inaccuracy may affect the trustworthiness of MBDTA. For instance, if inaccuracy can only inflate WCET estimates, then it may affect somehow tightness but not trustworthiness. Therefore, it is of utmost importance to guarantee that measurements are taken so that the WCET is never underestimated.

If other PMCs are used, analogous problems can be expected if counters are not shared across cores. If some degree of sharing occurs, for instance by counting misses in a shared cache, then it is unclear how to accurately identify the program responsible for each particular event.

D. Operation on the Measurements

Once a set of execution time measurements has been obtained, the only trustworthy claim is that the WCET is higher or equal than the highest observed execution time (HOET). However, the distance between the HOET and the WCET is typically unknown as shown in Figure 2. Again, several approaches have been considered to estimate the WCET [46], [28]. Typically, the WCET is estimated by increasing the HOET by a given factor (e.g., by 20%). While the scientific justification behind such an approach is roughly null, it works in practice in many cases where the user has non-negligible knowledge of both hardware and software being analysed.

E. Certification

MBDTA has been successfully used in the certification of many single-core and few dual-core safety-critical systems where SDTA cannot be afforded. Those systems typically include fail-safe systems[2], which account for the vast majority of systems in the automotive and railway domains among others, and a significant fraction in avionics [14].

F. Multicore Considerations

The adoption of multicore processors – either wanted or imposed – leads to new scenarios where tasks interact in non-obvious ways in a number of hardware resources *simultaneously*, thus leading to huge execution time variations. For instance, it has been shown that execution time may experience a 5X growth in 4-core processors used in the space domain [11] and above 4X when 4 cores are used in a Freescale P4080 considered for an avionics domain case study [31], which is much beyond the margin industry could afford. Hence, the complexity to derive those margins grows with multicores and so trustworthiness can only decrease if an inordinate price is not to be paid to budget for the unknown. Note that safety standards regard any approach as acceptable if it has been used sufficiently in the past and it can be proven that use conditions are analogous for the system being verified, which holds for MBDTA and single-core processors but not for multicores.

G. Takeaway Message

The trustworthiness of MBDTA depends on factors critically contingent on the user knowledge of the hardware and software under analysis. Inter-task software and hardware conflict effects, which are complex enough to analyse for single-core processors, do severely aggravate in multicores. On the other hand, industrial practice often relies on measurements as users may be simply unable to produce the information needed to apply SDTA, which is regarded as more trustworthy than MBDTA when systems and applications are sufficiently simple. Those users may therefore have to conceive increasingly stressful tests not only to certify their products against safety-related standards, but also to reach sufficient coverage-based confidence to make final product release.

It is unclear how this process will scale with the advent of multicores, especially if hardware platforms need to be used efficiently to reduce costs and keep competitive edge.

V. HYBRID DETERMINISTIC TIMING ANALYSIS

Some hybrid approaches have arised by increasing confidence of measurements with static information while still keeping those analyses friendly for industry in an attempt for bringing the best from SDTA and MBDTA into a hybrid timing analysis (HYDTA). Those hybrid approaches augment MBDTA with some static information from the control flow of the program to provide WCET estimates for unobserved execution paths [47], [38].

Next we review the effect of hybrid approaches on two aspects that challenge trustworthiness of MBDTA: test inputs and operation on the measurements.

---

[2] A system is fail-safe if there is a safe state in the environment that can be reached in case of a system failure either by the safety function or diagnostics, e.g., a train can be stopped.

A. Test Inputs

Hybrid approaches typically use only those test inputs provided by the user. However, as it is the case of RapiTime and other approaches [47], [38], they may collect execution time measurements at a finer granularity than the whole program (e.g., at function level, at basic block level, etc.). Then, some type of control flow analysis is performed to identify potential execution paths that have not been observed and measurements are operated to derive WCET estimates considering those new paths. How to operate on those measurements is described in the next subsection.

While considering new paths increases confidence on the WCET estimates, it is unclear to which extent those estimates are trustworthy:

- All paths may have not been considered for complex programs (e.g. with indirect function calls). If so, it is hard to determine whether the path leading to the WCET estimate has been included in the new paths that the hybrid approach considers.
- Path coverage is not enough. Apart from path coverage, other input-dependent effects may alter the execution time. While the user has some degree of control of those effects for the particular test inputs used, there is no guarantee on whether those effects are upper-bounded in the new paths considered.
- Excessive path coverage. Although it is not a matter of trustworthiness, building paths analytically may produce some infeasible paths that may increase the WCET estimate unnecessarily.

Overall, it is hard, if at all possible, to quantify the extent to which trustworthiness grows with increased path coverage.

B. Operation on the Measurements

Hybrid approaches add some degree of intelligence in the computation of the margin that needs to be considered over the HOET. For instance, instead of using the HOET, one can use the highest execution time across all those paths considered feasible, thus gaining some confidence. Still, even if those effects indicated in Section V-A are properly upper-bounded, measurements from different parts (e.g., basic blocks) have been collected in different runs and thus, their cache interaction has not been truly observed. As explained before, cache memories are particularly sensitive to small variations in the address patterns, as they can lead to large execution time variations. Thus, although hybrid techniques increase trustworthiness, it is unclear to which extent.

C. Certification

Since HYDTA provides no less confidence than MBDTA, it can be used in the certification of the same type of systems. However, HYDTA is subject to roughly the same limitations as MBDTA on the complexity of the underlying platform.

D. Multicore Considerations

Inter-task effects can occur in multicores, and identifying those scenarios leading to the WCET is a hopeless task. If this is the case, confidence on the WCET estimates grows very little by using hybrid approaches as the impact in execution time of inter-task effects in shared resources is much less evident than that of intra-task effects in single-core processors.

E. Takeaway Message

Hybrid approaches increase trustworthiness w.r.t. MBDTA while keeping industrial viability of the approach. This is so because some degree of intelligence and analysis is added into the computation of the WCET estimate. However, such trustworthiness gain is hard to quantify. Furthermore, WCET estimates obtained on multicores still are subject to the same issues as in the case of MBDTA. Therefore, hybrid approaches are better than MBDTA in terms of trustworthiness and so they may be used reducing the pressure on the user to produce tests. In fact, they may indicate which execution paths lead to the highest execution times, thus guiding the user in the process of producing stressful tests. However, in order to have *sufficient* confidence, skilled users are needed, as for MBDTA, so that they are able to understand how the program under analysis interacts with hardware features.

VI. STATIC PROBABILISTIC TIMING ANALYSIS (SPTA)

SPTA [5], [2] has been devised for very simple processor models limited to fully-associative or set-associative caches for which a deterministic placement is used. The latter however, resorts to keeping *exactly* the same cache set alignment for all memory addresses at analysis and at deployment, much like SDTA, thereby renouncing the competitive benefit of SPTA against SDTA in terms of more lenient requirements.

The applicability of SPTA to more realistic processor designs has not been proven yet, which impedes considering current SPTA as a viable alternative for industrial use. Nevertheless, the trustworthiness of current SPTA techniques is challenged analogously to SDTA techniques: its trustworthiness strongly depends on the accuracy of the timing model of the hardware and the information provided by the user.

VII. MEASUREMENT-BASED PROBABILISTIC TIMING ANALYSIS (MBPTA)

Some works have focused on applying statistical or mathematical techniques, such as extreme value theory (EVT) and convolution, to the execution time observations taken from real platforms [15], [4]. It has been shown however that the trustworthiness of the resulting pWCET estimates is challenged by the same threats noted for MBDTA [6]: the test inputs provided by the user must provide sufficient coverage for a number of sources of execution time variation, such as the alignment of objects in memory – which determine their location in cache – that are hard to control accurately. Hence, although the input data passed to EVT or convolution meet their requirements (e.g. independence and identical distribution for EVT), the strong dependence of those values on the particular conditions incurred in the test cases, degrades the trustworthiness of the probability distribution function obtained in that manner beyond the particular conditions exercised during analysis.

Other MBPTA approaches set explicit constraints to mitigate the impact of the sources of execution time variation [9]. Considerable aid to that end may come from using hardware that actively breaks dependence on those factors (for example via time randomisation) so that the coverage needs on the user are much lower [6]. In principle, those MBPTA approaches only need the user to ensure sufficient path coverage, but not memory address or value range coverage of any kind. Those MBPTA techniques have been evaluated in complex processor designs including pipelined cores, complex cache hierarchies and shared resources [23], [20]. However, they rely on hardware features that are not in production yet, although they are being explored in domain-specific processors [35]. It is also worth noting that software-only randomisation alternatives have been shown to work for single-core processors [24].

Even if the appropriate hardware support is in place, the trustworthiness of MBPTA is challenged by a number of aspects that we cover next.

A. *Test Inputs*

As explained, MBPTA relies on the user providing path coverage [9]. This is regarded as a simpler task than in MBDTA as the number of sources of execution time variation is reduced to only path coverage, as opposed to the case of MBDTA, which needs coverage for all valid combinations of the different sources of execution time variation [6]. For instance, given an application with 1,000 different execution paths and 1,000 different placements of objects in memory (e.g. code, data, libraries, OS objects), MBPTA would require the user (ideally) to produce test inputs for those 1,000 paths, whereas MBDTA would require 1,000,000 test inputs for each combination of memory placement and path.

B. *Measurement Collection*

Challenges on accurate measurement collection for MBPTA are analogous to those for MBDTA. Still, if inaccuracies only inflate WCET estimates – which we assume to be also the case for MBDTA – and given that MBPTA only relies on execution time measurements, this challenge should not affect the trustworthiness of the method.

C. *MBPTA Method*

MBPTA has been shown to use a number of statistical tests for its correct application [9]. Those statistical hypothesis tests use particular threshold values (e.g. significance value) which, although being set to typical values for use in criticality-concerned domains, may lead to some false positive/negative outputs. Interestingly, the event of tests not passing due to some random effects does not lead per se to non-trustworthy pWCET estimates. Instead, the analysis process can just be repeated as needed.

MBPTA also relies on a convergence criterion to decide when the data contained in the set of execution time measurements is enough to obtain trustworthy and tight pWCET estimates. In particular, this method stops collecting data when a number of steps lead to similar-enough pWCET distributions. Unfortunately, convergence may be reached too early, before execution time measurements include enough data. This risk is discussed in [1], [39]. In such (extreme) cases, MBPTA could deliver optimistic pWCET estimates, thus being non-trustworthy. Recent work [1] deeply analyses the scenarios in which these anomalies may arise and provides methods to detect them for some types of programs with even distribution of accesses to most addresses, so that the convergence criterion does not challenge trustworthiness anymore. This method needs to be extended to other program types.

D. *Certification*

MBPTA is a new method not used before for certification against safety standards. Thus, arguments for its certification are needed so that pWCET estimates can go through the certification process. Initial steps in this direction have been already taken for the single core case [45] and for the mixed-criticality multicore case [34], but further work is needed to make a safety case valid in different domains.

E. *Multicore Considerations*

MBPTA relies on hardware platforms with specific characteristics for the arbitration policies on access to shared resources (i.e. buses, memory controllers) and management of shared cache memories. Simple – yet effective – solutions have been devised to deal with these issues. In particular, randomised policies [20] or the use of the *worst-case mode* [33] have been shown to be effective for arbitrating the access to shared resources. Solutions for cache memories rely on, for instance, cache partitioning [33] or controlling the eviction frequency of the different cores [43]. While other methods also enable the use of multicores, the combination of these designs and MBPTA has been proven to attain high average and guaranteed performance – which SDTA fails to provide – while still providing time composability – which MBDTA fails to provide – as needed by end users.

F. *Takeaway Message*

MBPTA on top of MBPTA-friendly hardware keeps the industrial viability of MBDTA – it is a cost effective approach – while gaining trustworthiness in most aspects where MBDTA lacks of it. Also tightness has been proven in industrial avionics case studies [46]. Still some further steps need to be taken so that appropriate hardware, software and safety arguments can be made available before MBPTA can be used in real products. On the other hand, once those conditions are met – and no impediment has been found so far –, the number of issues challenging the trustworthiness of the method has been shown to be low and they are often attainable for end users. Still path coverage and the convergence criterion of MBPTA for any type of program are important challenges to be tackled.

VIII. HYBRID PROBABILISTIC TIMING ANALYSIS

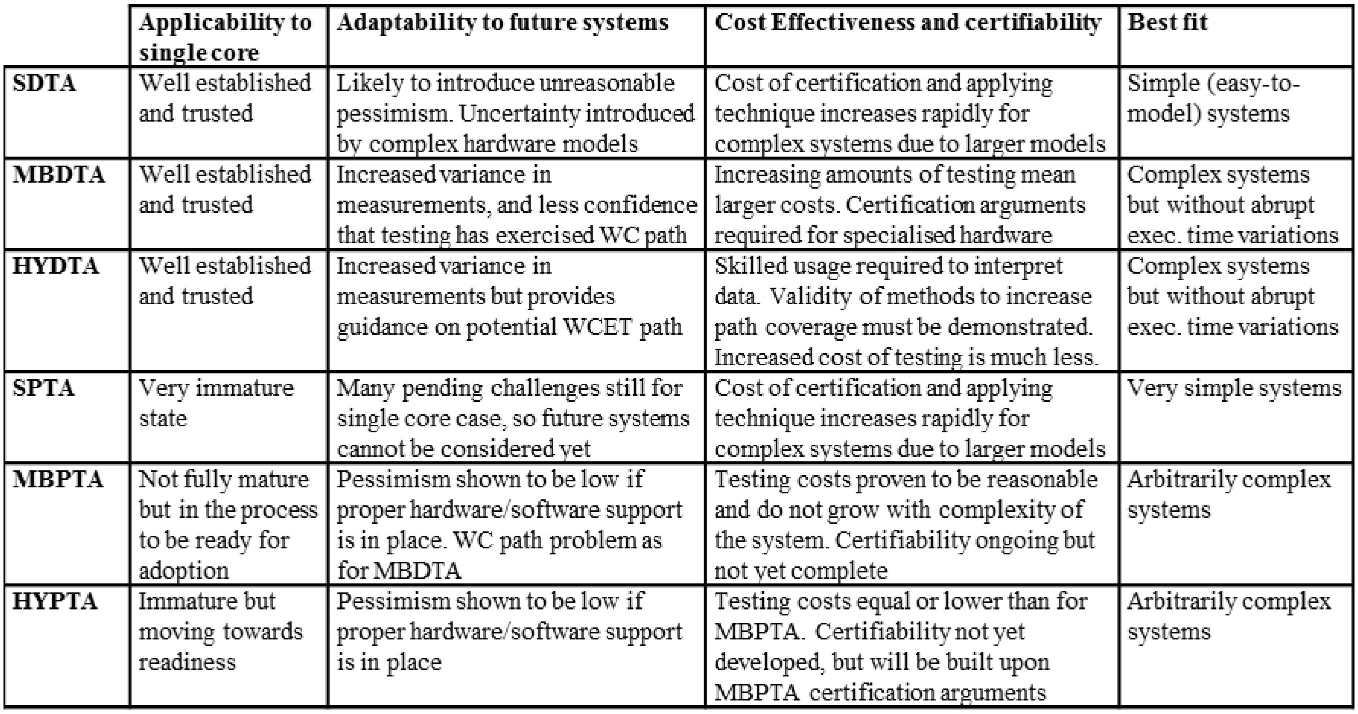
As for SPTA, hybrid PTA (HYPTA) is still in its infancy and thus, no viable alternative for industry has been deployed yet. To the best of our knowledge, *Path Upper-Bounding (PUB)* [25] is the main HYPTA technique so far. PUB increases path coverage w.r.t. MBPTA, thus releasing the user to some significant extent from providing such coverage. However, PUB relies on (automatic) code modifications to derive pWCET estimates for the original program. It is unclear how pWCET tightness is challenged by PUB and, more importantly, how those pWCET estimates can be proven valid for the original program for different safety standards given that they have been obtained for a modified program.

Although current HYPTA solutions cannot be used in real industrial problems yet, we are aware of some ongoing work to increase path coverage of MBPTA by means of HYPTA approaches without needing to modify the application under analysis. Thus, one of the limitations of PUB is expected to be removed in the near future. How to link new methods to certification processes is still an issue to be tackled, but no showstopper is currently known to impede this to be achieved.

IX. SUMMARY

We have surveyed a number of WCET analysis techniques. Table II summarises their main traits for applicability,

TABLE II
SUMMARY OF WCET TECHNIQUES



	Applicability to single core	**Adaptability to future systems**	**Cost Effectiveness and certifiability**	**Best fit**
SDTA	Well established and trusted	Likely to introduce unreasonable pessimism. Uncertainty introduced by complex hardware models	Cost of certification and applying technique increases rapidly for complex systems due to larger models	Simple (easy-to-model) systems
MBDTA	Well established and trusted	Increased variance in measurements, and less confidence that testing has exercised WC path	Increasing amounts of testing mean larger costs. Certification arguments required for specialised hardware	Complex systems but without abrupt exec. time variations
HYDTA	Well established and trusted	Increased variance in measurements but provides guidance on potential WCET path	Skilled usage required to interpret data. Validity of methods to increase path coverage must be demonstrated. Increased cost of testing is much less.	Complex systems but without abrupt exec. time variations
SPTA	Very immature state	Many pending challenges still for single core case, so future systems cannot be considered yet	Cost of certification and applying technique increases rapidly for complex systems due to larger models	Very simple systems
MBPTA	Not fully mature but in the process to be ready for adoption	Pessimism shown to be low if proper hardware/software support is in place. WC path problem as for MBDTA	Testing costs proven to be reasonable and do not grow with complexity of the system. Certifiability ongoing but not yet complete	Arbitrarily complex systems
HYPTA	Immature but moving towards readiness	Pessimism shown to be low if proper hardware/software support is in place	Testing costs equal or lower than for MBPTA. Certifiability not yet developed, but will be built upon MBPTA certification arguments	Arbitrarily complex systems

adaptability, cost and certifiability. The conclusion we can come to is that many of the existing techniques will become increasingly difficult to apply in mixed-criticality and multi-core systems, which are increasingly becoming commonplace even in criticality-related domains. Those techniques will also progressively become less cost-effective due to the entailed level of complexity and extent of testing required. We believe there is not a single timing analysis technique that dominates all others in all respects, that is providing higher confidence, requiring less user-provided data – hence increasing trustworthiness – and scaling to complex architectures. In general each technique is better equipped to analyse systems with specific characteristics, as detailed in Table II.

SDTA can hardly keep pace with modern hardware designs and it is therefore bound to be used with comparatively simple processor architectures. This trait makes it fit for the most stringent safety-related functions, e.g. DAL-A in commercial avionics, running on single-core processor architectures or federated systems (i.e. more stringently isolated than with classic Integrated Modular Avionics, IMA). However, it is worth nothing that, for instance, in current avionics designs, DAL-B to DAL-E functions can easily account for more than 70% of all on-board functions [14]. Thus, we recommend using SDTA only for fail-operational systems in the highest safety integrity levels, which can be run on isolated and simple systems where all information required for accurate timing modelling can be obtained with affordable costs. However, only a small fraction of systems falls within this category.

MBDTA is affected by the fact that on new platforms old margins (e.g. 20%) are questionable. Deriving new margins is complex if at all possible. HYDTA is better than MBDTA in terms of trustworthiness but in order to have sufficient confidence, skilled users are needed able to understand how the program under analysis interacts with hardware features. MBDTA and HYDTA are the only choice when high performance is required or multiple applications need to be consolidated onto the same system for cost, power and weight reasons. While skilled users have proven that these methods can be used in the context of single-core processors, scaling these techniques towards multicores becomes extremely painful due to the lack of understanding of application interactions in shared resources. Thus, only few systems have been certified on top of multicores (in particular dual-cores) with MBDTA and HYDTA.

Although MBPTA and HYPTA have not been deployed in any real system yet, they have been shown to tackle the limitations of DTA by allowing end users resort on easy-to-carry-on measurements and use mixed-criticality multicore systems while increasing the automation of the process and thus, reducing the degree of control that users need to exercise to obtain WCET estimates. All in all, MBPTA/HYPTA allows using multicores efficiently for safety-critical systems (SDTA does not attain such efficiency) attaining the level of confidence required for certification (MBDTA/HYDTA cannot reach the same level of confidence). However, MBPTA/HYPTA industrialisation and certification arguments are not yet complete. Thus, these techniques need to become mature enough to be a real choice for end users. To that end they are being integrated in commercial HYDTA tools (RapiTime [38]) to be used on industrial hardware/software stacks in the avionics, railway, space and automotive domains in the context of PROXIMA [35]. While results so far are promising, this technology is not yet ready for its use in commercial products.

In summary, SDTA is the best choice for (few) systems in the highest integrity levels if simple hardware is used and the effort (time and cost) to produce the information needed by SDTA can be afforded. Otherwise, MBDTA and HYDTA are the only choice. While their guarantees are poor from a scientific point of view, skilled users have proven them to be of practical use for single-core and some dual-core systems. Scalability to multicores is, however, a roadblock. MBPTA and HYPTA tackle this challenge and offer promising results, but they have not reached the maturity required for commercial products yet.

X. CONCLUSIONS

Deriving WCET estimates for safety-critical applications is a complex task and many issues jeopardise the trustworthiness of the whole process. In this paper we have reviewed the main timing analyses and identified a number of issues challenging each of the approaches. In particular, input data and models challenge SDTA trustworthiness. MBDTA lacks of sound methods to derive trustworthy WCET estimates from measurements. Hybrid approaches reduce – but not remove – those issues challenging MBDTA. Finally, some variants of PTA show promising results and are exposed to few issues challenging their trustworthiness in comparison with other timing analyses; however, PTA still misses real industrial hardware platforms and complete certification arguments to be used in real products.

ACKNOWLEDGMENTS

The research leading to these results has received funding from the European Community's FP7 [FP7/2007-2013] under the PROXIMA Project (www.proxima-project.eu), grant agreement no 611085. This work has also been partially supported by the Spanish Ministry of Science and Innovation under grant TIN2012-34557 and the HiPEAC Network of Excellence. Jaume Abella has been partially supported by the Ministry of Economy and Competitiveness under Ramon y Cajal postdoctoral fellowship number RYC-2013-14717.

REFERENCES

[1] J. Abella et al. Heart of gold: Making the improbable happen to extend coverage in probabilistic timing analysis. In *ECRTS*, 2014.

[2] S. Altmeyer and R. I. Davis. On the correctness, optimality and precision of static probabilistic timing analysis. In *DATE*, 2014.

[3] ARM. *Cortex-R5. Technical Reference Manual. Revision: r1p2*, 2011.

[4] G. Bernat, A. Colin, and S.M. Petters. WCET analysis of probabilistic hard real-time systems. In *RTSS*, 2002.

[5] F.J. Cazorla et al. Proartis: Probabilistically analysable real-time systems. *ACM Trans. on Embedded Computing Systems*, Dec. 2013.

[6] F.J. Cazorla et al. Upper-bounding program execution time with extreme value theory. In *WCET Workshop*, 2013.

[7] CENELEC. *EN50128 Railway Applications: Software for Railway Control and Protection*, 2001.

[8] S. Chattopadhyay et al. A unified WCET analysis framework for multi-core platforms. In *RTAS*, 2012.

[9] L. Cucu-Grosjean et al. Measurement-based probabilistic timing analysis for multi-path programs. In *ECRTS*, 2012.

[10] G. Edelin. Embedded systems at THALES: the artemis challenges for an industrial group. In *presentation at ARTIST Summer School*, 2009.

[11] M. Fernández et al. Assessing the suitability of the NGMP multi-core processor in the space domain. In *EMSOFT*, 2012.

[12] FreeScale. *P4080 QorIQ Integrated Multicore Communication Processor Family Reference Manual. Rev 1*, 2012.

[13] FreeScale. *e500mc Core Reference Manual. Rev 3*, 2013.

[14] D. Geiger. Personal communication from Airbus. Workshop on Integration of mixed-criticality subsystems on multi-core and manycore processors, 2015.

[15] J. Hansen, S. Hissam, and G. A. Moreno. Statistical-based wcet estimation and validation. In *WCET Workshop*, 2009.

[16] Infineon. *XMC4500 Microcontroller Series for Industrial Applications Reference Manual. Rev 1*, 2012.

[17] International Electrotechnical Comission. *IEC 61508, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems, Edition 2.0*, 2009.

[18] International Organization for Standardization. *ISO/DIS 26262. Road Vehicles – Functional Safety*, 2009.

[19] J. Jalle et al. Deconstructing bus access control policies for real-time multicores. In *SIES*, 2013.

[20] J. Jalle et al. Bus designs for time-probabilistic multicore processors. In *DATE*, 2014.

[21] T. Kelter et al. Static analysis of multi-core TDMA resource arbitration delays. *Real-Time Systems*, 50(2):185–229, 2014.

[22] R. Kirner and P. Puschner. Classification of WCET analysis techniques. In *ISORC*, 2005.

[23] L. Kosmidis et al. Multi-level unified caches for probabilistically time analysable real-time systems. In *RTSS*, 2013.

[24] L. Kosmidis et al. Probabilistic timing analysis on conventional cache designs. In *DATE*, 2013.

[25] L. Kosmidis et al. PUB: Path upper-bounding for measurement-based probabilistic timing analysis. In *ECRTS*, 2014.

[26] H. Li, I. Puaut, and E. Rohou. Traceability of flow information: Reconciling compiler optimizations and WCET estimation. In *RTNS*, 2014.

[27] Y.-T. S. Li and S. Malik. Performance analysis of embedded software using implicit path enumeration. In *DAC*, 1995.

[28] T. Lundqvist and P. Sandin. Towards a practical WCET analysis approach based on testing. In *ECRTS Work-in-progress*, 2008.

[29] E. Mezzetti and T. Vardanega. Cache Optimisations for LEON Analyses (COLA) Final Report. Technical Report COLA-FR-001-i1r1, ESA/ESTEC, 2011.

[30] E. Mezzetti and T. Vardanega. On the industrial fitness of WCET analysis. *WCET Workshop*, 2011.

[31] J. Nowotsch and M. Paulitsch. Leveraging multi-core computing architectures in avionics. In *EDCC*, 2012.

[32] J. Nowotsch et al. Multi-core interference-sensitive WCET analysis leveraging runtime resource capacity enforcement. In *ECRTS*, 2014.

[33] M. Paolieri et al. Hardware support for WCET analysis of hard real-time multicore systems. In *ISCA*, 2009.

[34] J. Perez. Towards modular certification of mixed-criticality product lines based on multicore and virtualization technology (IEC-61508) wind power and railway case studies. In *Workshop on Integration of mixed-criticality subsystems on multi-core and manycore processors*, 2015.

[35] PROXIMA. Probabilistic real-time control of mixed-criticality multicore and manycore systems. oct 2014. http://www.proxima-project.eu/.

[36] E. Quinones et al. Using Randomized Caches in Probabilistic Real-Time Systems. In *ECRTS*, 2009.

[37] Rapita Systems Ltd. Automating WCET analysis for DO-178B/C. White Paper. Accessed Jan 2015.

[38] Rapita Systems Ltd. Rapita verification suite. http://www.rapitasystems.com/products/rvs. Accessed Jan 2015.

[39] J. Reineke. Randomized caches considered harmful in hard real-time systems. *Leibniz Transactions on Embedded Systems*, 1(1), 2014.

[40] Jakob Rosen, Alexandru Andrei, Petru Eles, and Zebo Peng. Bus access optimization for predictable implementation of real-time applications on multiprocessor systems-on-chip. In *RTSS*, 2007.

[41] RTCA and EUROCAE. *DO-178C / ED-12C, Software Considerations in Airborne Systems and Equipment Certification*, 2011.

[42] D. Siewiorek and P. Narasimhan. Fault tolerant architectures for space and avionics. In *Workshop on Dependability in Robotics and Autonomous Systems*, 2006.

[43] M. Slijepcevic et al. Time-analysable non-partitioned shared caches for real-time multicore systems. In *DAC*, 2014.

[44] J. Souyris et al. Computing the worst case execution time of an avionics program by abstract interpretation. In *WCET Workshop*, 2005.

[45] Z. Stephenson, J. Abella, and T. Vardanega. Supporting industrial use of probabilistic timing analysis with explicit argumentation. In *INDIN*, 2013.

[46] F. Wartel et al. Measurement-based probabilistic timing analysis: Lessons from an integrated-modular avionics case study. In *SIES*, 2013.

[47] I. Wenzel. *Measurement-Based Timing Analysis of Superscalar Processors*. PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 2006.

[48] R. Wilhelm et al. The worst-case execution time problem: overview of methods and survey of tools. *Trans. on Embedded Computing Systems*, 7(3):1–53, 2008.

[49] J. Yan and W. Zhang. WCET analysis for multi-core processors with shared L2 instruction caches. In *RTAS*, 2008.