SAFETY CONSIDERATIONS FOR WCET EVALUATION METHODS IN AVIONIC EQUIPMENT

Xavier Jean, Sylvain Girbal – Thales Research and Technology – Palaiseau, France
Anthony Roger, Thomas Megel– Thales Avionics – Vélizy-Villacoublay, France
Vincent Brindejonc – Thales Air System – Limours, France

Abstract

Most safety-critical avionics systems are defined as "hard real time". That means they must deliver their function within pre-defined deadlines. Missing a single deadline at system level is considered as a failure condition that may be catastrophic. At software level, this is a single failure that must be mitigated with appropriate means to prevent that failure condition.

Real-time requirements are addressed in software components by Worst Case Execution Time (WCET) evaluations. Several methods have been explored in the literature, for which classifications have been proposed according to their techniques and precision of their results. However, these classifications do not consider the contribution of WCET evaluation techniques to safety processes.

In this paper, we present a safety process that integrates WCET evaluation on embedded software. This process allows us to highlight the benefits and limits that WCET evaluation methods bring in industrial practices.

Introduction

Today's aircrafts embed several avionics systems that are critical for flight safety. These systems are usually qualified as "hard real-time". That means they must fulfill their service within pre-defined deadlines, while a single miss would be considered as a failure condition in the meaning of CS-25.1309 [1].

Hardware and software components belonging to a hard real-time system inherit from timing requirements, and are given local deadlines. In safety terminology, a deadline miss by one component is a "single failure". By design, the occurrence of a single failure cannot lead solely to a catastrophic failure condition [1]. However it may be directly linked to a hazardous failure condition, and/or combine with another failure to reach a catastrophic level. Hence it represents a non-negligible risk for flight safety, that shall be mitigated in order to achieve certification.

We focus in this paper on a class of analyses called "Worst Case Execution Time" (WCET) evaluation. This kind of analysis aims at assessing a time budget for a piece of software, with guarantees that it will terminate its execution if it is granted that budget. WCET evaluation actually refers to a large variety of concepts and techniques; many of them are summarized in a survey by Wilhelm et al. [2].

Several criteria can be proposed to classify WCET evaluation techniques. For instance, static methods are computational and simulate the behavior of a processor's model executing the task's software; dynamic methods rely on measurements of execution times of software on the final target. Some techniques claim true upper bounds on execution time, while others give probabilistic results [3]. Finally, some techniques apply on isolated tasks, while others require a global knowledge of all task's details. All methods share common principles, for instance they admit evaluated WCET is approximated more or less precisely, but will never be reached. However, there is no consensus on what kind of guarantees WCET evaluation is supposed to bring, and how it can be used to ensure that in the end, the system is safe.

That paper aims at proposing an overview of a safety process that includes a WCET evaluation method, and at illustrating how that method can contribute to avionics system's safety. This overview enables a qualitative comparison of WCET evaluation methods, with a focus on the way they can be combined to improve safety assurance.

That paper starts with two sections that sum up aspects relative to safety process and WCET evaluation. It is followed by one section that deals with WCET evaluation methods limitations and possible enhancement, and two sections that develop

Work for this paper was executed within the SAFURE project. The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 644080.

the following classes of analysis encountered in a safety process:

- Safe design. Top-down methods that aim at building a product that copes optimally with safety constraints.
- Safety assessment. Methods ensuring that safety objectives, as defined at design time, have been met.

Safety Process Overview

In industrial practices, design and development phases of safety-critical equipment are performed by engineers alongside with safety processes. In this section, we give an outline of the safety process WCET evaluation methods will have to comply with. Figure 1 provides a sketch view of a standard safety process in avionics as provided by ARP4754A [4] or ARP4761 [5].

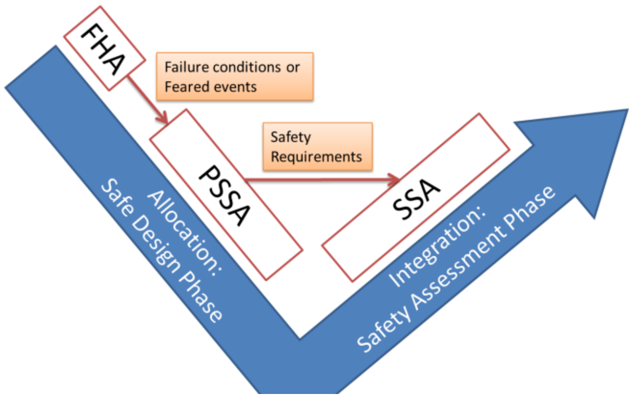
FHA
Failure conditions or Feared events
Safety Requirements
PSSA
SSA
Allocation: Safe Design Phase
Integration: Safety Assessment Phase

Figure 1: Sketch of Avionics Safety Process

This process is centered on a standard V cycle which descending branch is in general called allocation and an ascending branch called integration (see ARP4754 [4]). From a safety-centered point of view, the allocation phase can be named “safe design phase” and the integration phase, “safety assessment phase”. This denomination will be used all throughout this paper even though the allocation and integration principles are also applicable to safety process.

At the top of the safe design phase (and even prior to this phase, the Functional Hazard Analysis (FHA) determines Failure Conditions (or Feared Events) on which the complete analysis will be based. Preliminary System Safety Assessment (PSSA) is dedicated to architectural mitigation of these Feared Events. Safety requirements determined in PSSA are both qualitative via the definition of safety mechanism and quantitative via failure probabilities and performances of safety mechanisms. In the safety assessment phase a System Safety Analysis (SSA) compiles evidences that safety requirements are correctly implemented and realized on the final product.

Although this process is mainly defined at aircraft level, it is also applicable with a slight modification at lower levels, and it can become an iterative process from aircraft to basic components.

This process is aeronautics-oriented but is in fact very general and inspired by standard system engineering principles. It can be refined and implemented depending upon the application domain and the problem at hand.

Safe Design Phase

A safe design process aims at guiding the overall design phase so that the system copes optimally with high-level safety requirements. It is performed during the PSSA phase.

Figure 2 proposes a way of performing a PSSA in the safe design phase. On that figure, red boxes are PSSA related, orange boxes are inputs to PSSA process and green ones. Green arrows flow design data and red ones flow safety related data.

A safe design process presents the advantage to prepare the safety assessment phase very early by performing safety studies for allocation of safety requirements. This will be directly used in the safety assessment phase for the synthesis of evidences.

On Figure 2, preliminary design elements are used to imagine a safety concept for each Feared Events. Such safety concepts can be informal or not, but simplicity of exposition and communication should be favored. In its basic principle, it is based on scheme and drawing that only focuses on the design elements (systems, parts, information flows, etc.) contributing to the considered feared events (see for instance the Functional Failure Path Analysis of DO254 [6]) and the mitigation principle that avoid propagation of the various associated failures up to a the Feared Event. These mitigation measures are summarized in requirements, that are further refined to particular timing performances (see [7] for instance). These requirements are thus qualitative with quantitative performance characteristics.

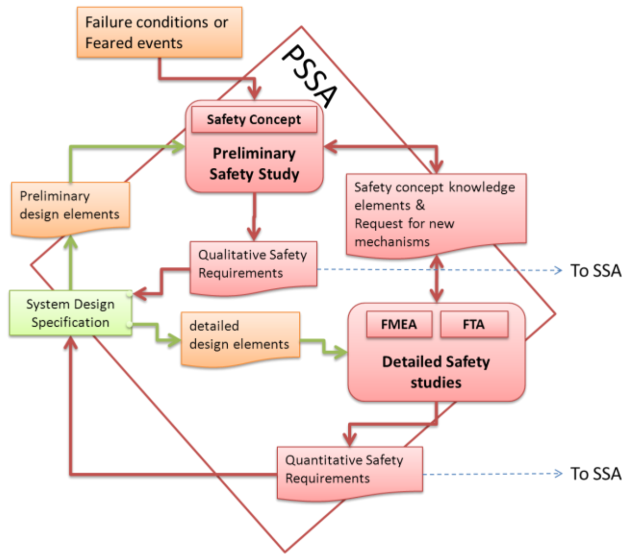
Failure conditions or Feared events
PSSA
Safety Concept
Preliminary Safety Study
Preliminary design elements
Safety concept knowledge elements & Request for new mechanisms
Qualitative Safety Requirements
To SSA
System Design Specification
detailed design elements
FMEA
FTA
Detailed Safety studies
Quantitative Safety Requirements
To SSA

Figure 2: Overview Of Safe Design Process

It is a general assumption that on safety concept elaboration, safety engineers and design engineers work in a close collaboration. This is even more crucial when technology aspects are key.

Feared Events are characterized by probability objectives that are allocated to requirements on design elements failure modes through detailed safety studies. These studies are basically of two complementary types:

- Failure Mode Effect Analysis (FMEA),
- Fault Tree Analysis (FTA).

The requirements on design elements failure modes, derived through these studies, are focused on:

- The safety level to be reached on the control of systematic failures (e.g. DAL),
- The probability allocated to the failure mode.

Safety Assessment Process

The goals of the Safety Assessment phase is to confirm that the designed and tested product complies with the safety objectives of the FHA and does not introduce other risks that have not been foreseen. These general goals are realized through three main activities (see Figure 3):

- Verification of qualitative safety requirements, in particular on safety mechanisms. It corresponds to the verification of presence and correct implementation of the mechanisms.
- Verification of quantitative safety performances of safety mechanisms: lapse time to safe state, coverage rate, etc.
- Verification of qualitative requirements on development level objectives (e.g. DAL).
- Verification of quantitative requirements on failure mode probabilities. This will be developed in sections dealing with WCET evaluation.
- Analysis of technical events occurring during development or on similar systems already in operation. This analysis can influence safe design phase analysis up to Feared Event definition in the FHA (hopefully rarely). It has to be analyzed through Detailed Safety Analysis that it, in turn, influences. It has not been represented on Figure 3 for readability reasons and because it is out of the scope of this paper.

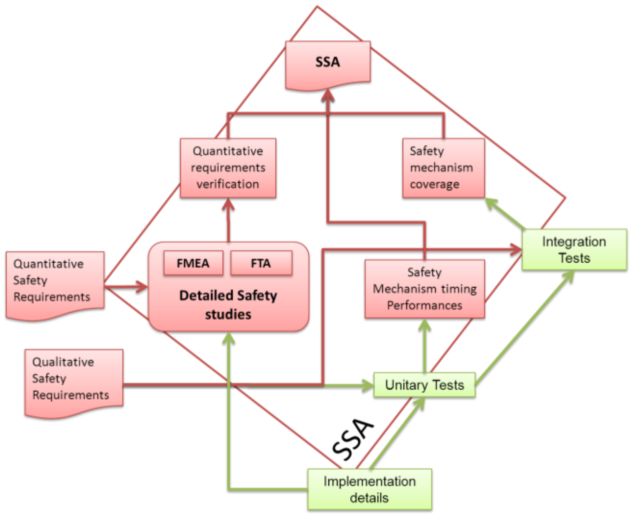
SSA
Quantitative requirements verification
Safety mechanism coverage
Integration Tests
Quantitative Safety Requirements
FMEA
FTA
Detailed Safety studies
Safety Mechanism timing Performances
Qualitative Safety Requirements
Unitary Tests
SSA
Implementation details

Figure 3: Overview Of Safety Assessment Process

When the safe design phase has been performed, the integration or safety assessment process is simplified for all preceding aspects. For instance

- Safety concepts allow to identify directly the impact of an incorrect realization of a safety mechanism on the Feared Events. Of particular interest for WCET is the correct timing realization of safety mechanisms that rely on software.

- Quantitative requirements on failure modes are directly checked unitarily. In case of discrepancy, a safety analysis performed during safe design phase is reused in order to verify that the safety requirement is still enforced despite this particular requirement.

We presented in this section a safety process that can be applied on avionic equipment. This process is generic and can be refined according to specific topic of interests, WCET evaluation in our case. The following sections present an overview of WCET evaluation methods, their limitations, and the way they can be integrated in our safety process.

WCET Evaluation Methods Overview

Motivations

In avionics equipment, embedded software is usually a set of tasks that are executed under the control of a scheduler, and that interact with third-party libraries through pre-defined API, e.g. libc, mathematical library.

Each task is associated to local deadlines, that come up periodically or are triggered by external events. Ensuring that each task will meet its deadline requires the following analyses:

- A task-per-task evaluation of individual needs for computation time. That stage is called "Worst Case Execution Time Evaluation" (WCET). The leading idea consists in exploring jointly hardware and software's most unfavorable behaviors.
- A global verification that scheduling will grant all tasks their needed amount of computation time before they reach their deadlines. That can be performed through simulation or offline tests. Scheduling analysis is a field that has both a wide community and has a rich literature, e.g. Davis and Burns survey for multi-core scheduling algorithms [8].

Worst Case Execution Time is a metric defined for each task, even if it may depend on the overall tasks set. We define it as the duration for which it is considered, with an acceptable level of confidence, that the task will have fulfilled its execution, whatever the processor's initial state and the events it will face, as long as their occurrences are compatible with a predefined usage domain.

When a WCET evaluation method is applied on a complex processor, the resulting WCET is always an approximation. Therefore, methods are usually ranked according to their precision, i.e. the difference between the computed WCET and the empirical bound observed on the execution times distribution. Finally, many WCET evaluation methods claim to produce "safe" WCET, i.e. sound overestimations of the real one. Other methods [9] [3] provide WCET associated with a probability.

WCET Evaluation Flow

WCET evaluation is a process that takes as input raw binaries. The goal is to ease certification by focusing on the final binary, and making this analysis independent from the compilation chain and associated software optimization flow. Hence WCET soundness does not depend on external tools.

WCET evaluation is composed of the following steps, represented on Figure 4. The first three steps are described in [2], and consider the task as uninterrupted. The fourth one is more empirical and comes from industrial practices:

- Flow Analysis. The binary will be explored to find out reachable execution paths. It abstracts the task as a "Control Flow Graph" (CFG), whose nodes, called "Basic Blocs", usually refer to functions and loops. Some methods formalize relations between basic blocs in the form of "execution contexts".
- Timing Analysis. Each basic bloc's execution time is assessed. At this level, we distinguish dynamic methods from static ones. The former gather basic bloc's execution time by test campaigns. The latter evaluates them with a simulation of software execution on a processor timing model (step 2 of Figure 4). Timing analysis results are often formalized as annotations on the CFG.

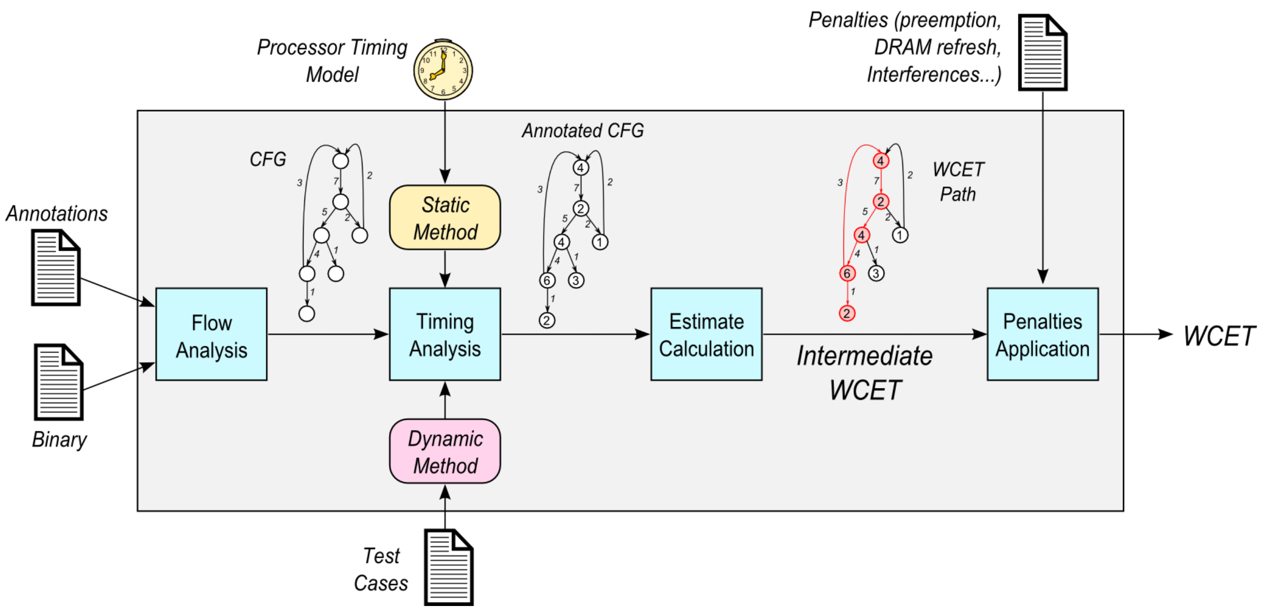

Figure 4: Overview Of WCET Evaluation Flow

- Estimate Calculation. Information from flow and timing analyses are correlated to find out the longest path in the program. An intermediate WCET is computed.
- Penalties Application on the intermediate WCET. Penalties mitigate the impact of phenomena that were ignored in previous stages, because of their complexity and/or the incapacity to anticipate them accurately. These phenomena may refer to Operating System ticks, external interrupts, DRAM refresh cycles…In most cases, penalties are evaluated empirically.

These steps are encountered under more or less formalized and automatized forms in industrial processes. They may be performed by WCET analyzers, which are available in commercial or open source solutions, described thereafter.

Static vs. Dynamic Methods

As described in step 2 of Figure 4, WCET evaluation methods are classified as “static” and “dynamic”.

Static methods rely on the analysis of software execution over a nearby cycle-accurate model of the processor. This enables fine grain analyses that identify worst case behaviors in components like pipeline and caches. These analyses are capable to cover non trivial behaviors on the processor, such as timing anomalies [10], i.e. worst case behaviors on the processor that result from local non worst-case behaviors. These situations cannot be easily reproduced by tests. However, soundness of static methods relies on the possibility to build correct models of processors. In practice, processor cores are described by manufacturers with a good level of details. Therefore, models' soundness can be assessed with a correct level of confidence. On other resources of the processor, timing models are coarser. They just associate timings for each operations over each device. This timing information refers to worst case situations, which are not systematic. For instance, they are strongly linked to banks and page states in DRAM controllers [11].

On the other hand, **dynamic methods** aim at gathering execution times from test campaigns. Thus, such a method will guide test scenarios to ensure that reachable execution paths have been explored, and unfavorable hardware behaviors have been covered. A dynamic method may rely on local measurements of portions of code, or end-to-end measurements over the whole task. The objective of a dynamic method is rather to speed up test campaigns without changing the test procedures.

Finally, static and dynamic methods only differ on a small part of the evaluation flow. Both require a deep exploration of the task's binary. That exploration is not trivial and often requires additional information provided by the user, and called “annotations”.

We introduced the general analysis flow encountered in existing solutions, under more or less automatized forms. The next section is about WCET analysis tools available in the state of the art.

WCET Analysis Tools

The WCET evaluation process described earlier is implemented in several tools, commercial or open-source. Census of existing tools have been performed in several publications [2] [12]. We focus here on tools that competed during the latest WCET Tool Challenge in 2011 [13], with a focus on a tool we consider as representative.

aiT [14] [15], developed by AbsInt, is a mature tool that implements a WCET computation flow with a static method. It supports several COTS processors cores, including PowerPC and ARM series, and has been used successfully in industrial environment [16]. Moreover, aiT has been designed to meet stability and traceability requirements for certification. Finally, it has been observed that aiT could compute WCET with a high precision [17], often under 25%.

Competitors of aiT in the field of static methods are Bound-T [18], TuBound [19], SWEET [20], Otawa [21]. All these tools have some maturity either in the academic or industrial communities. We can also encounter more recent tools that have interesting properties, such as formal validation of the absence of error in the analysis flow [22].

In the field of dynamic methods, RapiTime [23], developed by Rapita Systems, is considered as a mature tool. It aims at speeding up existing industrial processes by automating time measurement operations. Hence it automatically collects execution traces produced at runtime and computes statistics to obtain a distribution of execution times. A WCET can be derived from such measurements with an associated probability. Even if all reachable states at hardware level are not covered, RapiTime relies on randomization to ensure that pathological situations are highly improbable.

Competitors of RapiTime are TimeWeaver, developed by AbsInt, and GameTime [24] which is open-source.

WCET Analysis Flow Limitations and Possible Enhancements

In the previous section, we presented the analysis flow involved in existing WCET evaluation methods. To reduce certification costs, this flow is directly applied to the application binary so that it does not involve the compilation chain. It however introduces a set of limitations due to the lack of semantic of the binary format.

Limitations Due To CFG Complexity

Performing a retro-engineering construction of the control flow graph from the binary format is a tremendous task. Furthermore, the complexity of this generated CFG is very high, as it covers every possible execution paths, including control path corresponding to unrecoverable errors. Such errors usually correspond to sanity checks that might be part of the program itself, or part of the application libraries such as libmath sanity checks.

For safety critical software, it is part of the design process to decide which paths will be analyzed and which paths will not. Uncovered paths will require guarantee that they are not taken at runtime. For instance it may be possible to ensure that none of sensitive sanity checks can fail, relying on proof or testing, keeping sure that each of the program value is used within its usage domain.

Nevertheless, sanity checks do exist in the binary and the WCET analysis tool cannot differentiate them from the regular nominal control flow. This is illustrated on Figure 5 and Figure 6.

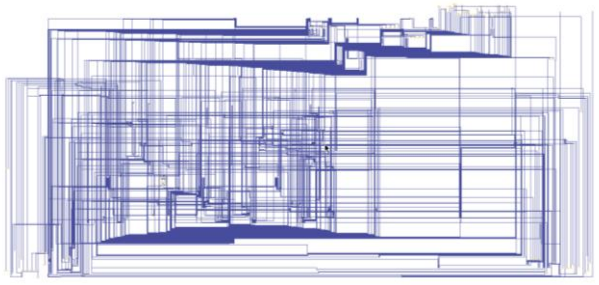

Figure 5: Raw CFG of Fibonacci Computation

Figure 5 provides an overview of CFG computed from a binary code that performs a Fibonacci computation. We used a non-optimized recursive Fibonacci implementation provided as an example with aiT [14]. CFG were also computed by aiT with a version for MPPA many-core processor developed by Kalray. While it is not appearing in the original source code, for each recursion, the program checks for a possible overflow, responsible for most of the Fibonacci function control, and the complexity of the CFG depicted on Figure 5.

Assuming that the domain usage analysis has already being performed, these sanity checks are useless and will never be triggered. We can therefore eliminate the corresponding CFG branches for the analysis, providing some annotations for the WCET estimation tool. Doing so leads to a simplified CFG appearing in Figure 6.

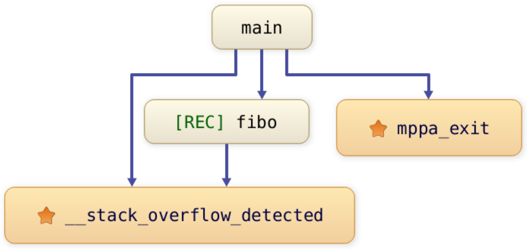

Figure 6: Simplified CFG, Ready For Analysis

Performing time analysis on the first version of the CFG will result in the best case a much more pessimistic WCET bound than performing it on the second simplified version, and in a worse case will not let the WCET analysis converge in an acceptable time.

Therefore, user-provided annotations are critical for the estimation tool to produce an acceptable, not overestimated WCET value.

Limitations Due To Complex Loop Bounds

Beyond not being able to spot unrecoverable-error related paths, performing the analysis on the binary format of the application also impacts the ability to successfully identify loop nests and their associated loop bounds.

The only kind of control appearing in the binary format are branches and conditional branches. It is therefore not trivial to distinguish the control flow associated with a loop to the one associated with a conditional.

Let's assume we analyze the program appearing in Figure 7. While being minimalistic, it involves a computation within a triangular loop with conditionals. The associated WCET will clearly depend on the number of time the computation is executed.

```
loop with complex control
    for(i=1; i<=10; i++)
    | for(j=1; j<i; j++)
    | | if((i+j) mod 2==1)
(I) | | | computation(i,j);
```

Figure 7: Source Code of Triangular Loop

Once compiled the corresponding assembly code appears in Figure 8. It is already very hard to identify the triangular nested loops, and WCET evaluation tools will usually fail to accurately compute the loop bounds, overestimating the number of iterations, and therefore the number of time the computation is executed and thus, the worst case execution time.

Again, user-provided annotations can solve the issue by indicating to the evaluation tool the exact number of iterations for most loops, avoiding some over bounding.

```
assembly version
04007b0 <triangle>:
 4007b0:  li     $16,1
 4007b4:  li     $17,1
 4007b8:  li     $18,1
 4007bc:  b      4007f0 -----------+
 4007c4:  addu   $2,$17,$16   <--- | --+
 4007c8:  srl    $3,$2,0x1f        |   |
 4007cc:  addu   $2,$2,$3          |   |
 4007d0:  andi   $2,$2,0x1         |   |
 4007d4:  subu   $2,$2,$3          |   |
 4007d8:  bne    $2,$18,4007ec-+   |   |
 4007e0:  jal    computation   |   |   |
 4007e8:  lw     $28,16($29)   |   |   |
 4007ec:  addiu  $16,$16,1 <---+   |   |
 4007f0:  slt    $2,$16,$17 <------+   |
 4007f4:  bnez   $2,4007c4   ----- | --+
 4007fc:  addiu  $17,$17,1         |
 400800:  slti   $2,$17,11         |
 400804:  beqz   $2,400818 ---+    |
 40080c:  move   $16,$18      |    |
 400810:  b      4007f0 ----- |  --+
 400818:  ret <---------------+
```

Figure 8: PowerPC Assembly of Triangular Loop

Limitations Due To Dependency Analysis

Through instruction level parallelism (ILP), pipeline architectures are able to execute several instructions concurrently. However, dependency between instructions may prevent such a parallel execution, impacting the overall execution time. For instance, successive instructions, each using the result of the previous instruction cannot be run in parallel.

To compute an efficient execution time, it is therefore necessary to perform a dependency analysis. Such kind of analysis already exist in compilers [25] [26] to check for the applicability of code transformations.

Dependency analysis are already very complex when applied to an high level compiler intermediate representation of the source code. When applied to binaries, they are even more complex, as the semantic behind variables has disappeared, with several variables being successively stored in the same register.

For this particular case, user-provided annotation cannot really help. The efficiency of the evaluated WCET will strictly depend on the accuracy of the pipeline model in the estimation tool.

Limitations Due To Data Alignment

The efficiency of many data-processing algorithms depends on data alignment: computation on well-aligned data can usually directly be performed by the ISA, while misaligned data require a set of shifting and logic operation to be performed prior and after the computation.

For global variables appearing in the final binary, the developer can enforce a suitable alignment. However, all the local variables appearing in the functions composing the application are allocated at runtime on the stack. As a consequence, they do not exist in the binary, and therefore, no assumption can be made by the evaluation tool on their existence not their alignment. Some functions like “memcpy” are very sensitive to this behavior: Let’s assume the source code of Figure 9 that is only copying one structure into another.

```
usage of memcpy
void myfunc(...)
  struct my_struct_t a,b;
  ...
  memcpy(&a,&b,sizeof(my_struct_t));
  // could also be written a = b;
  ...
```

Figure 9: Piece of Code Calling Memcpy

Both the source and the destination of the copy are local variables that will be allocated on the stack with unknown addresses at the time of the analysis. The memcpy algorithm appears in Figure 10.

```
memcpy
void memcpy(void *dst, const void *src, int len)
  if ((len==0) || (src==dst)) return;
  if(dst<src || no overlap)  // forward copy
    if(alignment of src and dst are not indentical)
      byte-per-byte copy for the whole data
    else
      if(src and dst are not word aligned)
        byte-per-byte copy few bytes until word aligned
      word-per-word copy for all complete words
      byte-per-byte copy few remaining bytes
  else // backward copy
    src+=len; dst+=len;
    as above but perform all copies backward
```

Figure 10: Overview of Memcpy Algorithm

The control path of the worst execution corresponds to mis-aligned source and destination, with an offset for their alignment and some overlapping regions. All these conditions depend on the source and destination addresses, so the evaluation tools have to systematically consider the worst case. In a code with a lot of recopy of data structures, the impact on WCET over bounding can be significant.

However, as the source and the destination are allocated on the stack, 1) there will be no overlap, and 2) both struct will be 32-bit aligned with a null alignment offset. As a consequence, only the best possible control path will be used at runtime for this copy, with no possibility for the evaluation tool to discover it.

Such information on data alignment can be provided by the user as annotations. However, to identify the issue, the user needs a deep knowledge, not only of his own source code, but also of the standard libraries such as libc which is providing memcpy.

Enhancements Due To Compiler Information

Many of the limitations described in this section are due to the lack of semantic associated with the binary form, and can be complemented by the user with annotations to provide the lost semantic. Rather than relying on the user, it would be possible to rely on the compiler that produced this binary to get the required information, as proposed in [27].

A compiler tool suite such as GCC [28] is organized in successive analysis and optimization passes. The compiler itself is manipulating a CFG structure, which is annotated during analysis phases and transformed during optimization phases down to producing the CFG corresponding to the binary.

If the annotations and analysis performed by the compiler are not related to timing, these annotations still carries some useful information like originating loop nests, loop bounds values, or variable-to-register mapping.

For a more efficient WCET evaluation, a first option would be to replace the generated CFG with the one used internally by GCC. If it would avoid the flow graph generation part of the analysis, it will make the analysis tool compiler-dependent. As the internal representations of compilers may evolve without notice, it might not be viable for a multi-target approach running different versions of various compilers.

A more suitable second approach would be to make the compiler systematically dump the information related to the currently user-provided annotations. In GCC, modularity allows us to develop a new pass that would be dedicated to extracting the required information from the internal CFG representation.

A first advantage would be to benefits from the existing loop bound computation pass of the compiler. Such a pass is very complex and relies on Z-polyhedral representation techniques to accurately compute loop bounds, as long as they are defined as affine functions of external loop iterators [29].

In this model, each instruction is clearly associated with an execution domain corresponding to the possible iterations during which this instruction will be executed. For instance the domain corresponding to the computation of Figure 7 is defined in Figure 11.

This domain information is later used by the compiler to effectively compute the number of iteration at instruction level [30] Extracting such information for the WCET evaluation tools would solve the issue related with over bounding of such loop bounds.

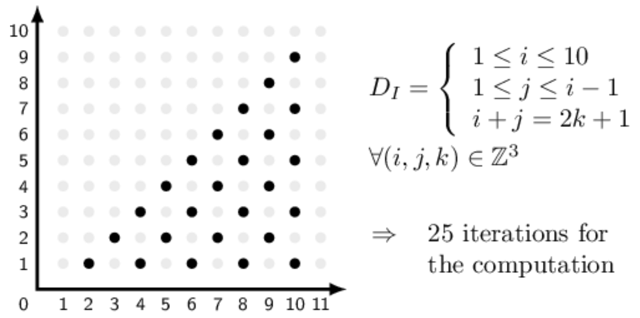

Figure 11: Execution Domain of Triangular Loop

The compiler has also more knowledge about the application data. While only the global variables are appearing in the application binary, the compiler also keeps information related to local variables. This would allow us to extract more information about alignment and data size, that could be used to statistically resolve some checks as those appearing for the memcpy example depicted in Figure 10. While the source and destination addresses are still unknown, the compiler knows that these addresses are 32-bit aligned, and that there is no overlap for the memcpy. Again, extracting such information can be useful to automatically build some annotations instead of forcing the user to provide them.

Several studies have already considered a coupled compiler / WCET estimation tool approach:

- In [19] Prantl et al. are coupling together a set of standalone tools including a Fortran compiler, an interval analyzer, a loop bound analyzer and a WCET estimation tool. From a source code, annotated with both real time constraints and loop bound information, the tool-suite generates an annotated binary that is later analyzed by the CALCWCET$_{167}$ estimation tool.
- In [31] the authors push the concept forward with a tight integration of the aiT engine with an ad-hoc C compiler. Beyond the ability to pass relevant information to the WCET estimation engine, this tight coupling also enable the compiler to perform some optimization passes dedicated to the worst-case optimization by considering estimated execution times while evaluating the pertinence of some loop transformations.

The proposed solutions have not yet reached a sufficient maturity level to be integrated into an industrial process. Considering the wide variety of hardware targets and of compilation languages, the integration of dedicated WCET-related passes in GCC seems promising.

The efficiency of the WCET estimation tools are usually tightly coupled with the effort the user provide to annotate the source code. Being able to automatically extract most of this information from the compiler will considerably reduce this effort and the applicability of WCET estimation tools.

Enhancements Toward Multi-Cores Support

Multi-core processors are expected to be embedded in the next generation of avionics equipment. On such processors, various sequential tasks will run at the same time on different cores. Evaluation of WCET is hardened because of interferences [32] [33] [34] [35].

Regarding WCET evaluation flow, the impact of interferences must be represented as a penalty, that is applied either in the second step (see Figure 4) that deals with timing analysis, or in the last step that deals with offline penalties applications.

On COTS multi-core processors, bounding interferences on a set of tasks is a complex problem. The largest research effort focuses on efficient and smart ways to constrain embedded tasks to bound interferences in order to provide static interferences penalties. That means they consider WCET evaluation method as fixed and alter the hardware and/or software environment to make them applicable.

Several approaches have extended existing WCET evaluation methods to take interferences into account on a closed set of tasks. Nowotsch [36] proposes the notion of Interference Sensitive WCET (isWCET), which extends the analysis flow implemented in aiT. It alters the processor's model with an interference penalty that is obtained by gathering a profile of resource usage for each task. That operation is also performed by aiT.

In [37], Bin proposes a set of benchmarks that stress selected components on a processor. Then she defines the notion of signature over a component of the processor, that roughly corresponds to the level of interferences inside the component with regard to the amount of incoming traffic. The signature also applies on software. Hence, by combining signatures at hardware and software level, it becomes possible to compute an interferences penalty over a set of tasks, and apply this penalty at the last step of evaluation flow.

An alternative approach based on budgets allocation for resource usage has been proposed in [38]. The authors allocate to each embedded task a budget for the number of operations they are allowed to perform on shared resources. An interference penalty is computed according to this budget. At runtime, a monitoring mechanism is in charge of stopping a task that exceeds its budget.

In this section, we pointed out some limitations of WCET evaluation methods due to the lack of information retrievable from a binary code of a task. This lack of precision is not by itself due to imprecise hardware models. It comes from the assertion that computed WCET will correspond to an execution path that cannot be actually reached.

These limitations, and others that are more dependent on methods, impact the way WCET evaluation methods contribute to safety processes. This is developed in the following sections.

WCET Considerations In Safe Design

In the preceding sections a general safety process has been depicted and articulation of the different WCET methods has been presented. This section shows how to integrate WCET evaluation in safe design process.

Probabilistic Objectives On Failures

WCET problematic arises in FHA at Failure Condition (or Feared Event) level or at refined levels. Consider at aircraft level that a deadline miss leads to a Feared Event (FE). Consider for instance an elevator raise time characterized by a random variable T then $T > T_{obj} \Leftrightarrow \{FE\}$, while T_{obj} corresponds to a timing budget.

Consequently, we have the following equations:

$$P\ \ FE\ \ = P\ \ T > T_{obj}\ \ ,$$

$$P\ \ T > T_{obj}\ \ \leq P_{obj} \qquad (1)$$

If a timing constraint occurs only on some system participating elements failure mode then similar equation applies.

For any refinement level, as soon as time constraints appear, the process in Figure 2 applies at system and electronic device levels with the following customization.

At **system level**, design specification provide a preliminary basic design. Then, designer and safety engineer collaborate to propose an architecture that can reach objectives on failures probabilities including deadline miss induced failures. This is the safety concept step in Figure 2. At this level the safety mechanism will consist in redundancies and surveillance of elapsed time. With this enriched architecture, the designer can allocate timing constraints and the safety engineer can allocate probabilities of failure including probabilities of missed deadlines using the previous equation. At this level, probabilities are allocated to mechatronics elements such as actuators, wires, computers…

The same process can be applied to electronic devices where the safety mechanisms are mainly timing surveillance. Probabilities will be allocated to electronic basic processes (IO transmissions, PCIe communication, PLD treatments and microprocessor treatments).

At **microprocessor level**, the same process can be implemented with significant differences if the microprocessor is a COTS. In this case, safety mechanisms can be either external to the microprocessor, either software. For the same reason, the electronic timings and associated probabilities cannot be considered as simple allocations but as constrained allocations. The only possible adjustment is at software level, as hardware is a COTS and remains as it is. In the case of multicore use or even if DMA are fully used in a single-core context, timing allocation should apply a safety margin for possible interferences. In the case of a multi-core processor, in failure probability computation, a probability could be allocated to the interferences: the deadline is not met at processor level if the processes do not meet their deadlines or if processes interfere.

A safe design process is thus applicable to WCET problems through the assimilation of a deadline miss to a particular failure mode. Several aspects may be explored regarding WCET evaluation, some of them have been detailed in [39]:

- It is valuable to be capable to perform WCET evaluation in early design for an equipment. In this case static methods offer a clear advantage: they do not require a final hardware to be applied. A model of this hardware, even if it is approximated, may be used to guide timing budgets allocation and failure probabilities.
- It is recommended to leverage qualitative risks linked with WCET evaluation methods as soon as possible. For instance, limitations described in the previous section may weaken budgets allocation, or rely on compiler information to get exploitable results.
- A WCET evaluation method will have to be integrated in an industrial process, which may integrate several actors. It is the case for Integrated Modular Avionics (IMA) systems [32]. A WCET evaluation method may require specific information that usually are not exchanged within existing processes. That is the case for details on an Operating System implementation.

This approach allows for considering deadline misses in a system failure from a safety analysis point of view. Consequently it allows for considering safety mechanisms that can mitigate the deadline misses. In an aeronautics context, as requested by CS25-1309 [1] a single failure cannot lead alone to a catastrophic event.

WCET Aspects In Safety Assessment

As presented in Safety Process Overview, the Safe Design Phase can enlighten the Safety Assessment Phase.

The safety concepts settled for the mitigation of deadline misses have to be tested, including the estimation of coverage rate through modeling and computation. Latency time, when it relies on software processing, have to be carefully checked by WCET evaluation methods. Even if the method does not returns probabilistic WCET, it is recommended to associate it with a failure probability, so that its impact can be directly integrated in safety analyses, e.g. FTA.

For instance, in [39], the authors pointed out several risks for WCET evaluation methods. These risks deal with:

- Uncertainty over processor models soundness in the case of static methods.
- Uncertainty over path analysis, especially if compiler information is used.
- Uncertainty about processor's state coverage, in case of dynamic methods.
- Uncertainty about annotations that are written by the end-user.

Quantitative requirements on timing failure modes are expressed in probability through Equation (1). These requirements can be checked unitarily using probabilistic WCET estimation methods [3], or jointly over the whole set of tasks. In case of discrepancy between a requirement and a probabilistic evaluation, this probabilistic evaluation can be directly inserted in the safety detailed study (for instance a fault tree), issued from Safe Design studies, in order to verify that the objective of safety goal is reached despite this particular requirement.

Static methods' results are locally very accurate and exhaustive on the hardware and software behavior. Additionally, they are expressive enough to provide feedback on why a WCET has been reached, even if the incriminated path or hardware configuration is not reachable. That makes them valuable bug-finders for WCET evaluation.

Finally, in the framework of safety assessment, timing global properties should be assessed using probabilistic arguments, whatever the evaluation method.

It is interesting to note that the probabilities of missed deadlines do not model WCET as exponential probability laws (see for example [40]). This leads to quantitative treatments in fault trees that are mathematically more complicated than the examples of ARP-4761 [5].

Conclusion

Satisfying hard real-time constraints with COTS hardware is a complex problem, especially on today's multi-core processors. This problem has been addressed in the academic and industrial communities as a matter of Worst Case Execution Time computation performed on each sequential piece of software. Literature dealing with WCET evaluation is abundant. Several WCET analyzers are available, under commercial license or open-source, and are organized in a structured ecosystem.

As explained in this paper, one of the main limitations of WCET analyzers is their difficulty to accurately retrieve and explore all binary's execution paths. Historically, that was seen as the cost of being independent from compilers. However, several approaches coupling a compiler and a WCET analyzer have recently emerged.

We proposed in this paper an outline of a safety process that integrates WCET evaluation. This safety process starts from system-level requirements, and refines these requirements among processing chains, that may include computation tasks, but also network communications and occasionally electro-mechanical elements, e.g. actuators. This safety process shows that :

- A safety-critical system at the highest level of criticality must tolerate at least one deadline miss.
- The "safe design" part of the process allocates time budgets and failure probabilities to each tasks.
- Time budgets and failure probabilities are checked in the "safety assessment" process. Iterations may be performed to adjust budgets.

During the safe design process, it seems relevant to apply static WCET evaluation methods in an early-design phases. Static methods do not require final hardware to be available, which is often the case in certification projects. Hence their use on hardware models, even if they are approximated, helps allocating time budgets and check that the platform is not over neither undersized. Among industrial practices, the main effort is focused on validation rather than pre-analysis. This approach would benefit from early WCET evaluation.

Later, in the development process corresponding to safety assessment phases, both static and dynamic methods can be applied. What is important is to keep in mind that all methods, even safe ones, bring their own sources of risks, either in WCET itself, or the way these methods are used. The risk has to be as easy as possible to assess. Hence it would be valuable to distribute alongside with WCET evaluation tools some skeletons or pre-filled information that could be used by safety engineers in analyses such as FMEA or FTA.

Combining dynamic and static methods also seems valuable in a safety assessment process. While the former does not rely on a hardware model, and considers the final platform as a whole, the latter can act as a pathfinder, by highlighting non-trivial execution paths and/or hardware situations that should be covered.

One interesting extension of this work would be the application of this safety process with a specific refinement for a multi-core processor. That would entail the introduction of a random variable that represents interferences impact over a set of tasks. That random variable, like for execution times, would be gathered in order to size a budget of interferences, while exceeding this budget might lead to a deadline miss.

References

[1] *CS-25, Certification Specifications for Large Aeroplanes, Amendment 6,* 2009.

[2] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat and P. Stenström, "The Worst-Case Execution-Time problem overview of methods and survey of tools," *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 3, pp. 36:1--36:53, may 2008.

[3] F. J. Cazorla, E. Quiñones, T. Vardanega, L. Cucu-Grosjean, B. Triquet, G. Bernat, E. Berger, J. Abella, F. Wartel, M. Houston, L. Santinelli, L. Kosmidis, C. Lo and D. Maxim, "PROARTIS: Probabilistically Analyzable Real-Time Systems," *ACM Trans. Embed. Comput. Syst.*, vol. 12, no. 2s, pp. 94:1--94:26, may 2013.

[4] SAE, *ARP-4754 : Certification Considerations for Highly-Integrated or Complex Aircraft Systems,* 1996.

[5] SAE, *ARP-4761 : Guidelines And Methods For Conducting The Safety Assessment Process On Civil Airborne Systems And Equipment,* 1996.

[6] RTCA/EUROCAE, *DO-254/ED-80: Design Assurance Guidance For Airborne Electronic Hardware,* 2000.

[7] V. Brindejonc and N. Plaze, "Rédaction, vérification et gestion des exigences de Sûreté de Fonctionnement," in *Lambda-Mu 18*, Tours, 2012.

[8] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Comput. Surv.*, vol. 43, no. 4, pp. 35:1--35:44, oct 2011.

[9] G. Bernat, A. Colin and S. Petters, "WCET analysis of probabilistic hard real-time systems," in *23rd IEEE Real-Time Systems Symposium (RTSS)*, 2002.

[10] J. Reineke, B. Wachter, S. Thesing, R. Wilhelm, I. Polian, J. Eisinger and B. Becker, "A Definition and Classification of Timing Anomalies," in *6th Intl. Workshop on Worst-Case Execution Time (WCET) Analysis*, Dagstuhl, Germany, 2006.

[11] U. Drepper, *What Every Programmer Should Know About Memory,* 2007.

[12] J. Gustafsson, A. Betts, A. Ermedahl and B. Lisper, "The Mälardalen WCET Benchmarks: Past, Present And Future.," *WCET,* vol. 15, pp. 136-146, 2010.

[13] R. von Hanxleden, N. Holsti, B. Lisper, E. Ploedereder, A. Bonenfant, H. Cassé, S. Bünte, W. Fellger, S. Gepperth, J. Gustafsson, B. Huber, N. M. Islam, D. Kästner, R. Kirner, L. Kovacs, F. Krause, M. de Michiel, M. C. Olesen, A. Prantl, W. Puffitsch, C. Rochange, M. Schoeberl, S. Wegener, M. Zolda and J. Zwirchmayr, "WCET Tool Challenge 2011: Report," in *Proc. 11th International Workshop on Worst-Case Execution Time (WCET) Analysis (WCET 2011)*, 2011.

[14] AbsInt. http://www.absint.com/ait/

[15] C. Ferdinand and R. Heckmann, "aiT: Worst-Case Execution Time Prediction by Static Program Analysis," in *Building the Information Society*, vol. 156, R. Jacquart, Ed., Springer US, 2004, pp. 377-383.

[16] J. Souyris, E. L. Pavec, G. Himbert, V. Jégu and G. Borios, "Computing the worst case execution time of an avionics program by abstract interpretation," in *In Proceedings of the 5th Intl Workshop on Worst-Case Execution Time (WCET) Analysis*, 2005.

[17] J. Gustafsson, "The Worst Case Execution Time Tool Challenge 2006," in *Leveraging Applications of Formal Methods, Verification and Validation, 2006. ISoLA 2006. Second International Symposium on*, 2006.

[18] Tidorum Ltd, http://www.bound-t.com/

[19] A. Prantl, M. Schordan and J. Knoop, "TuBound - A Conceptually New Tool for Worst-Case Execution Time Analysis," in *8th International Workshop on Worst-Case Execution Time Analysis (WCET'08)*, Dagstuhl, Germany, 2008.

[20] B. Lisper, "SWEET – a Tool for WCET Flow Analysis," in *6th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, 2014.

[21] C. Ballabriga, H. Cassé, C. Rochange and P. Sainrat, "OTAWA: an open toolbox for adaptive WCET analysis," in *Proceedings of the 8th IFIP WG 10.2 international conference on Software technologies for embedded and ubiquitous systems*, Berlin, Heidelberg, 2010.

[22] A. Oliveira Maroneze, S. Blazy, D. Pichardie and I. Puaut, "A Formally Verified WCET Estimation Tool," in *14th International Workshop on Worst-Case Execution Time Analysis*, Madrid, Spain, 2014.

[23] Rapita Systems,

www.rapitasystems.com/products/rapitime/

[24] S. A. Seshia and J. Kotker, "Game Time: A Toolkit for Timing Analysis of Software," in *Proceedings of the 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems: Part of the Joint European Conferences on Theory and Practice of Software*, Berlin, Heidelberg, 2011.

[25] M. Wolfe and U. Banerjee, "Data dependence and its application to parallel processing," *International Journal of Parallel Programming*, vol. 16, no. 2, pp. 137-178, 1987.

[26] S. Pop, A. Cohen, C. Bastoul, S. Girbal, G. andré Silber and N. Vasilache, "GRAPHITE: Polyhedral Analyses and Optimizations for GCC," in *In Proceedings of the 2006 GCC Developers Summit*, 2006.

[27] R. Kirner, P. P. Puschner and others, "Classification of Code Annotations and Discussion of Compiler-Support for Worst-Case Execution Time Analysis.," *WCET*, vol. 1, 2005.

[28] R. M. Stallman and G. DeveloperCommunity, Using The Gnu Compiler Collection: A Gnu Manual For Gcc Version 4.3.3, Paramount, CA: CreateSpace, 2009.

[29] C. Bastoul, A. Cohen, S. Girbal, S. Sharma and O. Temam, "Putting Polyhedral Loop Transformations to Work," in *LCPC'16 International Workshop on Languages and Compilers for Parallel Computers, LNCS 2958*, College Station, Texas, 2003.

[30] C. Bastoul, "Code Generation in the Polyhedral Model Is Easier Than You Think," in *PACT'13 IEEE International Conference on Parallel Architecture and Compilation Techniques*, Juan-les-Pins, France, 2004.

[31] H. Falk, P. Lokuciejewski and H. Theiling, "Design of a WCET-Aware C Compiler," in *Proceedings of the 2006 IEEE/ACM/IFIP Workshop on Embedded Systems for Real Time Multimedia*, Washington, DC, USA, 2006.

[32] G. Fernandez, J. Abella, E. Quiñones, C. Rochange, T. Vardanega and F. J. Cazorla, "Contention in Multicore Hardware Shared Resources: Understanding of the State of the Art," in *14th International Workshop on Worst-Case Execution Time Analysis*, Dagstuhl, Germany, 2014.

[33] T. Moscibroda and O. Mutlu, "Memory performance attacks: denial of memory service in multi-core systems," in *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, Berkeley, CA, USA, 2007.

[34] J. Nowotsch and M. Paulitsch, "Leveraging Multi-core Computing Architectures in Avionics," *European Dependable Computing Conference*, vol. 0, pp. 132-143, 2012.

[35] P. Radojkovic, S. Girbal, A. Grasset, E. Quiñones, S. Yehia and F. J. Cazorla, "On the evaluation of the impact of shared resources in multithreaded COTS processors in time-critical environments," *ACM Trans. Archit. Code Optim.*, vol. 8, no. 4, pp. 34:1--34:25, jan 2012.

[36] J. Nowotsch, M. Paulitsch, D. Buhler, H. Theiling, S. Wegener and M. Schmidt, "Multi-core Interference-Sensitive WCET Analysis Leveraging Runtime Resource Capacity Enforcement," in *Real-Time Systems (ECRTS), 2014 26th Euromicro Conference on*, 2014.

[37] J. Bin, S. Girbal, G. P. Daniel, A. Grasset and A. Merigot, "Studying co-running avionic real-time applications on multi-core COTS architectures," in *7th European Congress On Embedded Real Time Software And Systems (ERTS)*, 2014.

[38] A. Schranzhofer, R. Pellizzoni, J.-J. Chen, L. Thiele and M. Caccamo, "Worst-case response time analysis of resource access models in multi-core systems," in *Proceedings of the 47th Design Automation Conference*, New York, NY, USA, 2010.

[39] S. Altmeyer, B. Lisper, C. Maiza, J. Reineke and C. Rochange, "WCET and Mixed-Criticality: What does Confidence in WCET Estimations Depend Upon?," in *15th International Workshop on Worst-Case Execution Time Analysis*, 2015.

[40] W. Talaboulma, C. Maxim, A. Gogonel, Y. Sorel and L. Cucu-Grosjean, "Estimation of probabilistic worst case execution time while accounting OS costs," *21st IEEE Real-Time and Embedded Technology and Applications Symposium*, p. 21, 2015.

Acknowledgements

This research has been performed under FAA Aviation Research Division's Software and Electronics Section contract SDS-TO05. The technical monitor for this research was Srini Mandalapu of ANG-E271 Software and Electronics Section.

The authors acknowledge the contribution of Ms. Laurence Mutuel from CGH for reviewing the paper.