Bus-Aware Multicore WCET Analysis through TDMA Offset Bounds

Timon Kelter, Heiko Falk, Peter Marwedel
TU Dortmund University
{*timon.kelter,heiko.falk,peter.marwedel*}@*tu-dortmund.de*

Sudipta Chattopadhyay, Abhik Roychoudhury
National University of Singapore
{*sudiptac,abhik*}@*comp.nus.edu.sg*

***Abstract*—In the domain of real-time systems, the analysis of the timing behavior of programs is crucial for guaranteeing the schedulability and thus the safeness of a system. Static analyses of the *WCET* (Worst-Case Execution Time) have proven to be a key element for timing analysis, as they provide safe upper bounds on a program's execution time. For single-core systems, industrial-strength WCET analyzers are already available, but up to now, only first proposals have been made to analyze the WCET in multicore systems, where the different cores may interfere during the access to shared resources. An important example for this are shared buses which connect the cores to a shared main memory. The time to gain access to the shared bus may vary significantly, depending on the used bus arbitration protocol and the access timings. In this paper, we propose a new technique for analyzing the duration of accesses to shared buses. We implemented a prototype tool which uses the new analysis and tested it on a set of realworld benchmarks. Results demonstrate that our analysis achieves the same precision as the best existing approach while drastically outperforming it in matters of analysis time.**

***Keywords*-Multicores; WCET analysis; Shared bus;**

I. Introduction

With the rising importance of multicore systems in the processor market, including the embedded systems or cyber-physical domain, there is a growing need for tools to verify the timing behavior of such systems, and as such the WCET. For embedded systems, this may be the most important metric, because they often must work under real-time conditions where a response must be delivered in a predefined time. Therefore, fine-grained WCET analyses have been developed for single-core systems in the last decade [1], resulting in a variety of commercially available tools. In contrast, for multicores only first proposals exist. One of the major difficulties in analyzing the WCET for multicore platforms is that programs running on different cores may interfere with each other, for example during accesses to a common shared bus which connects the cores to a shared main memory. A possible approach to resolve these interferences is to implement a *Time Division Multiple Access* (TDMA) bus arbitration protocol which assigns a fixed-length time slot to each core in round robin fashion. In a scenario with n_c cores each having a TDMA time slot of length s_l cycles this leads to a maximum delay of $D^{max} = ((n_c - 1)s_l) + (z - 1)$ cycles for a bus access which occupies the bus for z cycles. This maximum delay is encountered when the access request is issued $z - 1$ cycles before the end of the executing core's slot. The bus cannot be granted then, since the access would span into the slot of the next core. On the other hand, the bus access is granted instantly, if the access request is issued when the bus is assigned to the executing core for at least z remaining cycles. Thus, an important problem is to determine tighter bounds on the durations of bus accesses. D^{max} cycles, as mentioned, is a valid but highly overestimated bound. In this paper, we present a new type of analysis which safely bounds the access time for TDMA-arbitrated resources with high precision and moderate analysis times thus enabling a tighter WCET estimation. The results can be used to avoid pessimistic hardware overdimensioning and to derive tighter system schedules.

The rest of this paper is organized as follows: In Section II, we will present related work, Section III introduces our system model used in the analyses and Section IV and V introduce the overall analysis framework as well as the general analysis concepts respectively. Section VI presents our new analyses which are evaluated in Section VII. Finally, we provide a summary of our results and give directions for future work in Section VIII.

II. Related Work

The first approaches to multicore WCET analysis only modeled the shared resources to some extent. Suhendra [2] and Zhang [3] analyzed the effects of a shared L2 cache without considering the interference on a shared bus that is used to access the shared cache. [3] provides a bound on the number of additional cache misses due to the inter-core interference, whereas [2] eliminates the interference altogether by exploring different scenarios of locking and partitioning the shared cache.

Gustavsson [4] investigates a totally different approach, where the whole multicore system is modeled as a set of timed automata. The WCET is obtained by proving special predicates through model checking. This approach allows for a detailed system modeling, but does not scale very well as all system states have to be explored in the course of the WCET analysis, leading to a state explosion.

For analyses that include the shared bus, the choice of the bus arbitration method is crucial. Pitter [5] compared the predominant arbitration methods and TDMA arbitration resulted as the most predictable method. Therefore, most

of the works which include a bus analysis are restricted to TDMA bus arbitration. To provide a better access time estimation than the mentioned D^{max} cycles, Andrei [6] tries to determine the precise time at which every single memory access takes place. The bus delay estimation is then performed separately for each access. The main problem is, that accesses in loops with an iteration count of i can potentially have i different access times associated to the same memory access. Therefore, the analysis has to unroll all loops virtually to determine the access times for each access individually, which makes the analysis runtime dependent on the loop iteration counts.

Chattopadhyay [8] circumvents this costly unrolling by aligning each loop head execution to the first TDMA slot during the analysis. However, this artificial alignment of each loop iteration results in an additional penalty term to be added in WCET estimation. Therefore, the analysis proposed in [8] is far more efficient but also less precise than [6]. The analysis which we propose in the following, will present a compromise between the two approaches, being almost as precise as [6] and only slightly less efficient than [8].

Finally, Pellizzoni [7] derives the worst-case bus delays in a multicore system analytically with the help of memory traffic arrival curves. This approach is different from ours since we do not require such curves.

A different direction in static timing analysis is the adaption of multicore hardware to exhibit better predictability properties. Paolieri [9] proposed a multicore architecture in which the WCET of basic blocks is measurable, whereas Mische [10] developed a superscalar SMT processor, which provides built-in real-time capabilities. These approaches are orthogonal to ours since we focus on estimating the WCET of tasks on existing hardware platforms.

III. System and Application Model

We assume a system architecture where $n_c \geq 2$ cores are present in a single processor. Each of the cores has an in-order pipeline and a private L1-Cache and all the cores are connected to a shared TDMA-arbitrated memory bus with a uniform TDMA slot size of s_l cycles per core. The bus is used to access a shared L2-Cache, which itself is linked to the main memory. The bus, the L2 cache and the main memory may be located on-chip or off-chip. We do not allow split transactions on the bus, therefore, for the maximum duration T^{max} of a bus transaction, $T^{max} \leq s_l$ must hold. An access to the TDMA bus may incur a variable delay, depending on when the access is performed, but the delay cannot exceed D^{max} cycles. As explained in the introduction, this bound is not tight in general. Due to $T^{max} \leq s_l$ and $D^{max} \geq ((n_c - 1)s_l)$, the maximum bus delay will at least be $(n_c - 1)$ times as big as the maximum memory latency. Thus, the bus access delay is the factor with the greatest variability and also with the greatest potential for overestimations during WCET analysis. This underlines the need for precise analyses of the bus access delays. In this paper we will provide such an analysis using a fixed TDMA schedule. The optimization of the TDMA schedule itself is out of the scope of the paper.

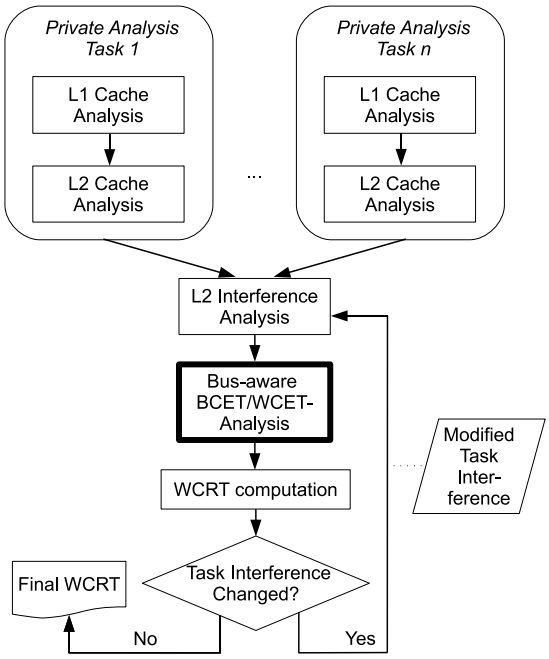


Figure 1. The analysis framework

All the caches in the considered system are non-inclusive and use the *least-recently-used* (*LRU*) replacement policy. The cache hierarchy can be easily extended e.g. with more private cache levels, because we apply the generic framework from [11] to determine which accesses from cache level $i-1$ hit cache level i. We only model instruction caches and thus assume that data accesses occur via a different bus and do not interfere with the instruction accesses in any other way. The integration of a data cache analysis into our analysis would remove these restrictions. We do not allow self-modifying code hereby removing the need to deal with cache-coherency in our model.

The input task dependencies are given as acyclic task graphs with a fixed mapping of tasks to cores. Each edge (x, y) in the task graph denotes that task y can start execution only after task x has finished. We use fixed-priority, non-preemptive[1] scheduling. For each loop L in the tasks, the minimum and maximum loop iteration counts B_L^{min} and B_L^{max} are given and the *control flow graphs* (*CFGs*) of the tasks are assumed to be well structured (reducible).

IV. Analysis Framework

We embed our new analyses into the Chronos timing analyzer framework from [8]. Figure 1 shows the analysis

[1]A preemptive scheduling would require the integration of a cache-related preemption-delay (*CRPD*) analysis which is out of the scope of this paper.

process. The framework first analyzes the cache behavior of each task in isolation and then computes the maximum possible cache interference in the shared L2 cache. This interference information is used to update the worst-case cache states of the individual tasks. The cache analysis assigns to each single access one of the following categories for each cache level: "Always Hit" (AH), "Always Miss" (AM), "First Miss" / "Persistent" (PS) or "Unknown Behavior" (UNKNOWN). PS means that the first execution of the instruction suffers a cache miss, but every following execution hits the cache, which is most useful for instructions inside of loops. For details on the cache analysis, the interested reader is referred to [8], since we are only using its results here. In the next analysis step, the cache information is used to compute BCET[2] and WCET values per task. This module (marked in bold in Figure 1) has been equipped with our new analysis technique, whereas all other modules have not been modified. After the tasks' BCETs and WCETs were computed, the overall system *worst-case response time* (WCRT) is determined. This process repeats as long as the task interference changes, e.g. due to altered task lifetimes. In the following, we will focus on the determination of single task WCETs with given cache states as this is our main contribution. Nevertheless, all of our analyses are applicable to the computation of BCETs as well.

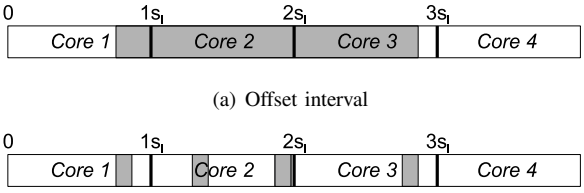
V. Static analysis of TDMA offsets

Our new analysis builds upon concepts which are heavily used in the analysis of other architectural features. To establish the link between those existing analyses and our new analysis, we first give a short overview of existing static analysis techniques. We also demonstrate why those techniques are not sufficient in our scenario.

A. Abstract Interpretation In Timing Analysis

A static timing analysis is usually composed of a microarchitectural analysis and a path analysis [1]. The microarchitectural analysis is responsible for determining abstract hardware states which describe the possible concrete hardware states at every basic block entry. This microarchitectural analysis is normally based on *abstract interpretation*, a technique for static program analysis, which can provide safe approximations of program or, in this case, hardware states. In the past it was successfully employed to analyze cache, branch prediction and pipeline behavior. With these hardware states, a basic block WCET can be computed, which in turn can be fed into the path analysis to compute the longest path through the program. The abstract hardware states which are used in our analysis model the state of the shared TDMA bus, i.e. at which points in time the execution of a block may start. Since the TDMA schedule is cyclic, we can revert to representing only *offsets* instead of absolute

[2]Best-Case Execution Time



(a) Offset interval

(b) Offset set

Figure 2. Different abstract representations for possible start offsets of a basic block

times. An offset o in our case can be computed from an absolute time t as $o = (t \mod n_c s_l)$. To model the fact that a block can be entered with more than one offset we devise two offset representations:

- An *offset interval* $I = [o_{min}, o_{max}]$
- An *offset set* $S = \{o_1, o_2, \ldots, o_n\}$

These offset representations are the abstract hardware states that will be used in the analyses. An example for the different representations can be found in Figure 2. While Figure 2(b) shows the offset set representation with the represented offsets marked in gray, Figure 2(a) presents the same offset information, again marked in gray, for the offset interval representation. Obviously, the set representation is more precise, but it also requires greater effort to maintain the sets during the analysis, thus leading to a typical tradeoff between analysis precision and analysis duration.

In the following, we use a special definition of basic blocks. A basic block $b = (i_1, \ldots, i_k)$ in our definition is a sequence of instructions which may only be entered at i_1 and only be exited at i_k and which, in addition, must also either not contain any instruction which potentially accesses the shared bus, or the block contains only a single instruction. The information whether an instruction potentially accesses the shared bus can be extracted from the cache information. In our case it may access the bus when it may access the L2 cache. This splits up a standard basic block which contains l potential bus accesses into at most $2l + 1$ basic blocks whose WCET is either fixed (no bus access) or variable (bus access). The basic blocks execute in-order, since we required an in-order pipeline. A generalization of our concepts to out-of-order execution is possible, but it is omitted due to size constraints. With this notion of basic blocks and the results from the other microarchitectural analyses which yield WCET values for the blocks without bus accesses, we can formulate the offset analysis as a data flow problem. The data flow analysis requires a function u which *updates* the computed state after the execution of a basic block b and a function m which *merges* the states at control flow joins in the control flow graph. Given the set $ET_b \subseteq \mathbb{N}$ of possible execution times of b and either an offset set S_b or an offset

interval I_b, we have

$$u(S_b) = \begin{cases} \text{off}_{\text{execute}} & b \text{ never accesses bus} \\ \text{off}_{\text{execute}} \cup \text{off}_{\text{access}} & b \text{ may access bus} \\ \text{off}_{\text{access}} & b \text{ always accesses bus} \end{cases} \quad (1)$$

$$u(I_b) = [\min(u(set(I_b))), \max(u(set(I_b)))] \quad (2)$$

with $set([o_{min}, o_{max}]) = \{o_{min}, \ldots, o_{max}\}$ and

$$\text{off}_{\text{execute}} = \{(o+d) \bmod s_l n_c | o \in S_b, d \in ET_b\}$$
$$\text{off}_{\text{access}} = \{(o+\Phi_p(o,d)) \bmod s_l n_c | o \in S_b, d \in ET_b\}$$

The $\Phi_p(o,d)$ function returns the time needed to finish the bus access (including the bus delay), when the bus request is issued by core $p \in \{0, \ldots, n_c - 1\}$, begins at offset $o \in \{0, \ldots, n_c s_l - 1\}$ and needs $d \in \{1, \ldots, T^{max}\}$ cycles to complete after the bus access was granted. In the TDMA arbitration we can define $\Phi_p(o,d)$ as:

$$\Phi_p(o,d) = d + \begin{cases} s_l p - o & \text{if } o < s_l p \\ 0 & \text{if } s_l p \le o \le s_l(p+1) - d \\ s_l n_c - o + s_l p & \text{else} \end{cases}$$

Note that ET_b may for example model the fact that we have a block with variable-latency instructions or a block whose L2 instruction memory access was classified as UNKNOWN. The merge functions for the two offset representations are:

$$m(S_1, \ldots, S_j) = \bigcup_{i \in \{1,\ldots,j\}} S_i \quad (3)$$

$$m(I_1, \ldots, I_j) = [\min(S^m), \max(S^m)] \quad (4)$$

where $S^m = m(set(I_1), \ldots, set(I_j))$. With these functions, we can establish a standard data flow analysis on the *interprocedural control flow graph* of each task (with given starting offsets for the task start block) which terminates after all offset data has stabilized. Unfortunately, this analysis will not be very precise, because *branches* and *loops* in the control flow force us to repeatedly merge the offset information, which quickly leads to results where a block can be reached with arbitrary offsets. In this situation, we cannot provide a better estimation than the pessimistic assumption that each bus access is delayed by D^{max} cycles. The imprecision that stems from branches can be reduced through the offset set representation which allows to track the offset development in more detail. Loops pose a bigger problem. They can only be handled effectively with the concept of *contexts* in the analysis.

Since the functions u and m are defined for both offset sets S and offset intervals I, we will formulate our analyses based on an abstract offset data structure O in the following which may be either an offset set or an offset interval.

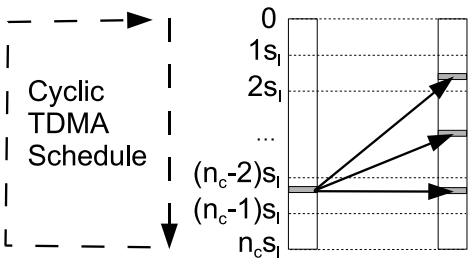


Figure 3. Mapping of a start offset to possible end offsets for a single loop iteration. The iteration takes $3s_l$ to $n_c s_l$ cycles in this example, depending on which path through the loop body is taken

B. Abstract hardware states and contexts

Usually, the hardware states presented in Section V-A are computed in a context-insensitive way, meaning that the abstract interpretation computes states which are valid for all *execution contexts* of a basic block, where an execution context denotes a certain loop iteration or calling context. This behavior is insufficient for some analyses like e.g. the cache analysis, where the first loop iteration may have a significantly different cache behavior than the following ones. For this purpose, *analysis contexts* were introduced, which describe the hardware states for a certain execution context. The known methods for dealing with contexts during bus access duration analysis are the following:

- The loop is virtually unrolled by a factor equal to its loop bound and thus, each loop iteration is explicitly analyzed [6]. This method, called *full virtual unrolling* is very precise but also very inefficient for larger loop bounds. It results in the analysis of B_L^{max} analysis contexts, which each represents exactly one execution context.
- The analysis is performed for a fixed offset o, and a delay is added that represents the maximum additional delay that can occur due to execution with offsets $s \neq o$. This is the approach from [8], and we will refer to it under the name *fixed-alignment approach*. It results in a single analysis context which represents all B_L^{max} execution contexts.

In the next section, we will present a third, novel approach to context handling in bus access duration analysis, which will analyze $1 \le x \le B_L^{max}$ contexts to provide a compromise between analysis duration and analysis precision. Our approach is based on an analysis of TDMA offsets as presented above.

VI. Computing Loop Offset Bounds

Our approach is based upon the observation that for each loop iteration which starts from a given set of offsets, we can compute the set of offsets in which the iteration may terminate. Therefore, our goal is to track the development of the TDMA offsets of the loop header block and thus to

provide more precise offset bounds than by using the data flow analysis from Section V. This requires:

- A structural analysis to find loops in the CFG, and to build a *directed acyclic graph* (DAG) from each loop or procedure body. Nested loops are represented as single nodes in the surrounding DAG. Due to this, we required our input tasks to be reducible in Section III.
- An analysis that computes the set of offsets that may be reached when a loop body is executed once with given starting offsets.

The overall analysis will then proceed in a hierarchical way, starting at the beginning of the task entry procedure and descending into called procedures or loops only when they are discovered in the CFG. The structural analysis is already present in the CHRONOS framework, whereas a single-iteration offset analysis is presented in Section VI-A. Section VI-B then introduces the core analysis which combines the single-iteration results into a complete loop WCET.

A. Determination of offset results for single iterations

As mentioned, we are interested in determining the offsets that can be reached after a single execution of the loop body finishes. This will be called a *loop iteration* in the following, in contrast to a *loop execution* which denotes the (possibly) repeated execution of the loop body until the loop condition is false. Figure 3 shows a scenario where a loop iteration, starting from a single, given offset may end at various different offsets, e.g. due to different paths through the loop. These single-iteration offset results can be determined by iterating over the loop's basic block DAG in topological order, as sketched in the following.

In our analysis of a single loop iteration, each basic block is seen as a transformation function which maps input offsets O_{in} (either an offset set or an offset interval as explained in Section V-A) to resulting offsets O_{out} and produces WCET values which are valid for the given O_{in}. Algorithm 1 shows the analysis of single basic blocks. Function calls (lines 11 - 14) or blocks which represent inner loops (line 2) are handled by specialized analysis functions. Note that function calls terminate basic blocks in our model. The WCET and offsets which result from bus accesses (lines 4 - 8) or simple instructions (line 10) are computed with the known ET_b values and Φ_p and u functions from Section V-A, where p is the core which executes the currently analyzed task. Each DAG analysis, on either a procedure or a loop body, then composes the single-block results in topological order and forms its own WCET and offset result out of them. Algorithm 2 shows this for the case of a single loop iteration, where b_{sink} and b_{header} are the sink and header node of loop l, respectively and $pred\,(b_i)$ returns the set of predecessor blocks for block b_i. By supplying the starting offsets to the loop iteration analysis (lines 3 - 4), this information becomes part of the analysis context, as explained in Section V-B. The iteration analysis then analyzes the behavior of each single block (lines 9 - 11) and propagates the results to the successor blocks (lines 6 - 7). Finally the results per loop iteration are summarized (line 13). The analysis of procedures in "AnalyzeProcedure" works analogously as "AnalyzeLoopIteration". This implies that recursive calls must be converted to standard loops before our analysis can handle them. For the analysis of complete loop executions (all iterations) in "AnalyzeLoop", we need to combine the context-sensitive single iteration results to form an overall loop WCET and offset result. This will be discussed in the next section.

Algorithm 1 AnalyzeBlock

Require: block b, offsets O_{in}
1: **if** b is head of inner loop l_{inner} **then**
2: **return** AnalyzeLoop (l_{inner}, O_{in})
3: **else if** b consists of bus access instruction **then**
4: $wcet = 0$
5: **for all** offset $o \in O_{in}, d \in ET_b$ **do**
6: $wcet = \max\left(wcet, \Phi_p\left(o, d\right)\right)$
7: **end for**
8: **return** $\langle wcet, u\left(O_{in}\right)\rangle$
9: **else**
10: $result = \langle \max\left(ET_b\right), u\left(O_{in}\right)\rangle$
11: **if** b is terminated by call to function f **then**
12: tmp = AnalyzeProcedure $(f, result.offsets)$
13: $result = \langle tmp.wcet + result.wcet, tmp.offsets\rangle$
14: **end if**
15: **return** $result$
16: **end if**

Algorithm 2 AnalyzeLoopIteration

Require: loop l, offsets O_{in}
1: **for all** blocks b_i of loop l in topological order **do**
2: **if** $b_i = b_{header}$ **then**
3: $wStart = 0$
4: $oStart = O_{in}$
5: **else**
6: $wStart = \max_{b_d \in pred(b_i)}\left(wFinish\left[b_d\right]\right)$
7: $oStart = m\left(\{oFinish\left[b_d\right] \mid b_d \in pred\left(b_i\right)\}\right)$
8: **end if**
9: $\langle wcet_{b_i}, O_{b_i}\rangle$ = AnalyzeBlock $(b_i, oStart)$
10: $wFinish\left[b_i\right] = wStart + wcet_{b_i}$
11: $oFinish\left[b_i\right] = O_{b_i}$
12: **end for**
13: **return** $\langle wFinish\left[b_{sink}\right], oFinish\left[b_{sink}\right]\rangle$

B. Deriving full loop WCETs

To implement "AnalyzeLoop" for a given loop l and starting offsets $O_{in,l}$, full unrolling could be performed by analyzing all iterations and supplying the offset results from

one iteration as inputs to the next one. Alternatively, only a single iteration can be analyzed, with a forced alignment at the TDMA schedule border and an added alignment penalty as suggested in [8]. Section V-B already mentioned that our goal is to avoid these two approaches, because they are computationally too expensive or lose precision, respectively. In this section we devise two new methods which present a compromise between those two extremes.

Global Convergence Analysis: Starting with the initial offset information $O_{in}^1 = O_{in,l}$ we iteratively analyze single loop iterations $i \in \{1, 2, \dots\}$ and record the WCET $wcet_i$ and offset result O_{out}^i. With the merge function m from Section V-A the offset inputs O_{in}^i for iteration i are then computed as $m\left(O_{in}^{i-1}, O_{out}^{i-1}\right)$ instead of simply setting $O_{in}^i = O_{out}^{i-1}$ as it would be the case in the fully unrolling approach. $m\left(O_{in}^{i-1}, O_{out}^{i-1}\right)$ is a superset of O_{out}^{i-1}, therefore the O_{in}^i increase monotonically and converge towards the set of all offsets with which the loop can possibly be entered. The analysis stops after iteration j when either $j = B_l^{max}$ or $O_{in}^j = O_{in}^{j+1}$ is true. In the first case we have hit the loop bound and thus have performed full unrolling implicitly, therefore this is the undesired case. In the second case we have reached a fixpoint of the starting offsets and thus the result from iteration j stays valid for all following iterations. In total there can't be more than $n_c s_l$ iterations, which is the number of possible offset values. The final loop WCET can then be easily computed as:

$$wcet_l\left(O_{in,l}\right) = \left(\sum_{i=1}^{j} wcet_i\right) + \left(B_l^{max} - j\right) \cdot wcet_j \quad (5)$$

The offset result for the loop is equal to the offset result from iteration j, because this result stays valid for all following iterations.

Graph Tracking Analysis: The global convergence analysis is superior to the static unrolling insofar, that it implicitly unrolls the loops selectively, as long as new information can be obtained. This is more suitable than a static unrolling, but it still relies on the idea of unrolling the first j iterations and handling the rest of the iterations under a single analysis context.

The drawback is that cyclic progressions of offsets cannot be captured by the analysis. Consider e.g. a loop in which all even iterations start with offset x and all odd iterations start with offset $y \geq x$, because only the even iterations have to wait for the TDMA bus access, whereas the odd iterations can then proceed with direct bus access. The global convergence analysis will analyze the first two iterations ($j = 2$), compute $O_{in}^j = \{x, y\}$ and use this offset information for all following iterations. This is clearly valid, but still imprecise. The example shows the need to handle *cyclic contexts* which do not distinguish the first j execution

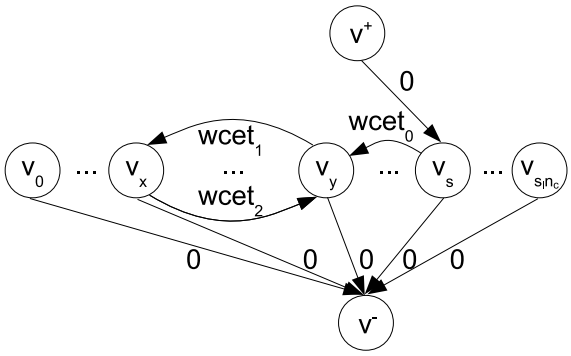


Figure 4. An example offset graph

contexts from the remaining ones, but which distinguish *groups* of execution contexts which repeat cyclically. In our case, a cyclic context consists of all iterations starting with offset o, which leads to $s_l n_c$ contexts. Thus, we can identify a cyclic context via the offset which it represents.

To obtain the final timing results using cyclic contexts we construct a weighted, directed graph from the contexts and compute the loop WCET by solving a flow problem on that graph. This graph $G = (V, E, c)$, also called *offset graph* in the following, has $V = \{v^+, v^-\} \cup V_{off}$ with *source* v^+, *sink* v^-, *context nodes* $V_{off} = \{v_0, ..., v_{s_l n_c}\}$, $E = E_{enter} \cup E_{exit} \cup E_{transition}$ and weight function $c : E \to \mathbb{N}$. We have

$$E_{enter} = \left\{(v^+, v_o) \mid o \in O_{in,l}\right\} \quad (6)$$

$$E_{exit} = \left\{(v_o, v^-) \mid o \in [0, s_l n_c]\right\} \quad (7)$$

For all edges $e \in (E_{enter} \cup E_{exit})$ we set the weight $c(e)$ to 0. $E_{transition}$ is then constructed by iteratively analyzing single iterations. For each iteration i, we compute $wcet_i$ and O_{out}^i from O_{in}^i using Algorithm 2. O_{in}^1 is set to $O_{in,l}$ and for the other iterations $O_{in}^i = O_{out}^{i-1}$ applies. After the analysis of each iteration we extend $E_{transition}$ by all edges $e = (v_i, v_o)$ with $i \in O_{in}^i, o \in O_{out}^i$ and $c(e) = wcet_i$. If any of these edges already exists, we update its weight by setting it to $\max\left(c(e), wcet_i\right)$. We stop the iteration analyses when we reach an iteration where no edge is added or updated. An example for such a graph is given in Figure 4 where the first iteration starts with offset s and the succeeding iterations alternate between starting offset x and y as sketched in the example at the beginning of this section.

The offset graph can then be used to obtain the final loop WCET by solving a dynamic flow problem [12]. In contrast to standard flow problems, dynamic flow problems have an explicit notion of time built into the problem formulation. Based on the offset graph we can derive two different dynamic flow problems: one for determining the WCET and one for the resulting offsets. The basis of the problem formulation is a flow function $x : E \times T \to \mathbb{N}$, which specifies for each edge $e = (u, v)$ the amount of flow $x(e, t)$

which leaves u at the discrete time instant t. This flow arrives at v at time $t + \tau(e)$ where $\tau(e)$ is the constant runtime of the edge. Conceptually, in our graph, a single time step of the flow problem corresponds to a single iteration of the loop, which implies $T = \{0, ..., B_l^{max}\}$. Thus a flow of $x(e,t) = 1$ through an edge $e = (v,w) \in E_{transition}$ represents the loop iteration t which starts at offset v and ends at offset w and has a maximum runtime of $c(e)$. Therefore we set $\tau(e) = 1$ for all $e \in E_{transition}$, since these edges model single loop iterations, and we set $\tau(e) = 0$ for all $e \in E_{enter} \cup E_{exit}$, modeling entry into and exit from the loop. Both dynamic flow problems share a common constraint that ensures that all flow which enters a node at a time step must leave it in the same step (i.e. there must be one loop iteration per time step):

$$\forall t \in T : \forall v \in V_{off} : \sum_{e \in \delta^-(v)} x(e, t - \tau(e)) = \sum_{e \in \delta^+(v)} x(e,t)$$

Here, $\delta^-(v)$ and $\delta^+(v)$ denote the sets of incoming and outgoing edges at node $v \in V$. For the start node v^+ and the sink node v^- we need to provide explicit bounds on the flow. We want F units of flow to leave v^+ at time 0 and to arrive at v^- at time B_l^{max} (i.e. we can model F full loop executions in a single flow problem). Therefore we have:

$$\sum_{e \in \delta^+(v^+)} x(e,0) = F \tag{8}$$

$$\forall_{e \in \delta^+(v^+)} : \forall t \in T \setminus \{0\} : x(e,t) = 0 \tag{9}$$

$$\sum_{e \in \delta^-(v^-)} x(e, B_l^{max}) = F \tag{10}$$

$$\forall_{e \in \delta^-(v^-)} : \forall t \in T \setminus \{B_l^{max}\} : x(e,t) = 0 \tag{11}$$

For the WCET analysis we only model the single worst-case loop execution scenario by setting $F = 1$ and by maximizing the objective function

$$\max \sum_{e \in E} \sum_{t \in T} c(e)\, x(e,t) \tag{12}$$

The loop WCET is then given by the value of the objective function.

For the offset analysis, we use $F = s_l n_c$ flow units which must arrive at the sink between time step B_L^{min} and B_L^{max}. We therefore need different sink flow constraints which replace Equations 10 and 11:

$$\sum_{e \in \delta^-(v^-)} \sum_{t \in T_{leave}} x(e,t) = F \tag{13}$$

$$\forall_{e \in \delta^-(v^-)} : \forall t \in T \setminus T_{leave} : x(e,t) = 0 \tag{14}$$

$$\text{with } T_{leave} = \left\{ t \mid B_l^{min} \leq t \leq B_l^{max} \right\} \tag{15}$$

The flow of each of the flow units through the graph models a possible loop execution scenario. If K is the (unknown) set of offsets with which the loop can be left, then we have $|K| \leq s_l n_c$ since this is the total number of possible offsets. With $F = s_l n_c$ flow units we can thus model at least one loop execution scenario which terminates with offset k for each offset $k \in K$. Therefore we can compute an overapproximation of K by maximizing the objective function

$$\max |\left\{ o \mid \exists t \in T_{leave} : x\left(\left(v_o, v^-\right), t\right) > 0 \right\}| \tag{16}$$

The offsets $O_{out,l}$ which result after the loop execution are then given as the elements of the set from Equation 16 with $K \subseteq O_{out,l}$. A formal proof of correctness is omitted due to space constraints but can be found in [18].

Using either the global convergence or the graph tracking analysis, the analysis of tasks as a whole now only requires the offset information at the entry point of the task, which is provided by the overall analysis framework through the known processor mapping and task dependencies. All internal offset information, and with this, the WCET of the task, can then be computed through the presented framework.

C. Offset analysis in architectures without timing anomalies

Timing anomalies are a phenomenon which complicates WCET analysis. According to the definition from [14] a system shows timing anomalies whenever local worst-case behaviour does not forcedly lead to global worst-case behaviour, thus for example whenever a cache hit instead of a cache miss does trigger the global worst-case behaviour. This may be the case e.g. on systems with instruction prefetching and speculative execution [13]. In the static analysis of systems with timing anomalies it is not feasible to prune the search space of the analysis [14]. Therefore in a cache analysis for a system exposing timing anomalies we may not assume an UNKNOWN access to be a cache miss (AM), but instead we must then consider both possibilities, a hit and a miss, in the analysis. On systems without timing anomalies we can safely assume the local worst-case (AM) to increase the analysis performance and precision.

In our offset analysis we did not prune the search space (the set of reachable offsets) at any point up to now. To increase the analysis precision for timing-anomaly-free architectures we can thus reduce the offset result of any merge or update operation to the offset o which is reached by the local worst-case path. Therefore, the differentiation between offset sets and offset intervals is of no importance for the analysis any longer, because we are only tracking single offsets after this reduction. The graph-based analysis is then ideally suited to track the development of the worst-case offsets inside of loops using the known ILPs from Section VI-B to compute the total loop WCET. This reduction to the local worst-case makes the analysis highly precise, because the main source of imprecision, the divergence of offset information, is eliminated.

D. Extensions for further micro-architectural analyses

In an analysis that includes the analysis of more microarchitectural features like pipeline and branch prediction, the computed overapproximations of the hardware states must become part of the analysis context, in addition to the offset information. For the global convergence analysis, this means that a global overapproximation of the hardware states at the loop header is built and used in the analyses. For the offset graph, every context node must be annotated with an overapproximation of the hardware states with which the node may be entered, including cache, pipeline and branch prediction states. In such a scenario, the graph must be iteratively refined until

1) No more edges are added or updated
2) The hardware states on all nodes have converged

Alternatively it is also possible to construct only a single, global overapproximation of the hardware states, depending on which degree of precision is required.

VII. Experimental Results

In the following, the different approaches to bus-aware WCET analysis are compared. As mentioned, we have implemented our approaches based upon the code from [8] which enables a precise comparison. The prototype tool analyzes executables compiled for the SimpleScalar platform and includes a thorough cache analysis. Unfortunately, no pipeline or branch prediction analysis is integrated yet, so all instruction latencies are set to 1 cycle. Section VI-D nevertheless introduced the general concept of how to perform such an integration. It can be expected that the classification of the approaches with respect to precision and analysis time stays the same even after additional microarchitectural analyses were integrated, since the number of analysis contexts is directly dependent on the analysis type as explained in V-B. The number of contexts in turn has the biggest influence on the analysis precision and duration. All experiments were run on an Intel Xeon 2.13GHz machine with 4GB of main memory under Debian Linux. Concerning the solution of the dynamic flow problems during the graph-tracking analysis, we used the CPLEX ILP solver in the experiments.

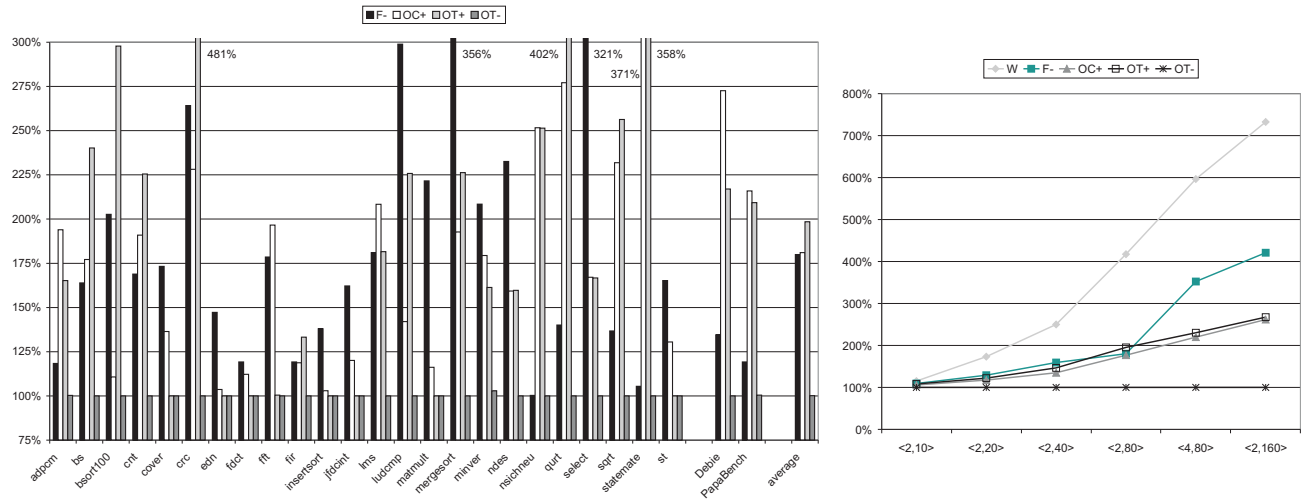
The experiments were performed on a subset of the `MRTC` test bench [15] where the tasks are independent from one another. Thus we map each `MRTC` test case $i \in [0, 23]$ from Table I to core $(i \mod n_c)$ with priority i, where 0 is the highest priority. We also tested the presented algorithms with the publicly available `PapaBench` [16] and `Debie` [17] benchmarks which are an unmanned aerial vehicle control software and a space debris monitoring software, respectively. The mapping of tasks to cores was done manually for these two benchmarks. The default system configuration is a 2-core system with 1KB L1 cache (direct-mapped, block size 32 byte) and 2KB L2 cache (4-way associative, block size 64 byte). Only for `Debie`, the cache configuration was changed to 2KB L1 cache (2-way associative) and 8KB L2 cache to account for the bigger program sizes of `Debie`. In any case, the L1 hit penalty is 0 cycles, the L2 hit penalty is 1 cycle and the main memory access time is 5 cycles modeling a Flash-based main memory. The default TDMA schedule assigns a slot of 80 cycles to each core. A more detailed overview of the used benchmarks is provided in Table I, including the byte size s_{byte} of the "text" section of the executable (excluding startup code), the lines of code LOC (excluding comments and empty lines), the number of loops L, the maximum loop nesting level D and the average loop bound $\varnothing_B$. The `Debie` and `PapaBench` benchmarks consist of 6 resp. 13 tasks which have a relatively simple structure, especially since they have almost no nested loops.

Benchmark	LOC	s_{byte}	L	D	$\varnothing_B$
`adpcm`	468	12480	18	0	69
`bs`	79	480	1	0	4
`bsort100`	74	1024	3	1	100
`cnt`	72	1552	4	1	40
`cover`	228	9312	3	0	60
`crc`	66	1936	3	0	102
`edn`	196	8000	11	2	61
`fdct`	148	5088	2	0	8
`fft`	97	8368	7	2	88
`fir`	188	1056	2	1	375
`insertsort`	20	752	2	1	10
`jfdcint`	165	5424	3	0	26
`lms`	146	4576	10	0	50
`ludcmp`	71	4544	11	2	4
`matmult`	81	1536	5	2	20
`mergesort`	266	9152	23	3	126
`minver`	135	6160	17	2	2
`ndes`	201	6256	12	0	19
`nsichneu`	2362	63632	1	0	2
`qurt`	87	1952	1	0	19
`select`	55	3120	4	2	8
`sqrt`	42	912	2	0	12
`st`	98	60528	1	0	1000
`statemate`	1128	11728	4	0	20
`Debie`	24528	1622912	39	1	157
`PapaBench`	4663	200256	10	0	3

Table I
Benchmark properties

A. Precision gain

In this section, we will distinguish between the approaches that assume no timing anomalies on the target hardware and those which do not make such an assumption. The fully unrolling and fixed-alignment analyses that are built into Chronos do make this assumption. Therefore, strictly speaking, only the comparison to our approaches with the extension from Section VI-C is feasible. In Figure 5(a), we have listed the WCET results for the different approaches on the `MRTC` test bench subset with the mentioned default machine configuration. In Figure 5 as well as in the following, all WCET results are relative to the WCET result of the fully unrolling analysis which does *not* consider timing anomalies . We use the following shorthands for the different approaches:



(a) WCET results per benchmark and average WCET results for $\langle n_c, s_l \rangle = \langle 2, 80 \rangle$

(b) Average WCET results for varying $\langle n_c, s_l \rangle$

Figure 5. WCET result summary for `MRTC`, `Debie` and `PapaBench`. All results are relative to the WCET computed with **U-**. The **W** approach is not shown in Figure 5(a) since its results are greater than 300% for 21 out of the 26 benchmarks (average performance of **W**: 414%)

W	Assume worst-case bus delay of D^{max} cycles for each bus access
F-	Fixed alignment (No timing anomalies) [8]
OC+	Offset analysis (Global convergence, allow timing anomalies)
OT+	Offset analysis (Graph-tracking, allow timing anomalies)
OT-	Offset analysis (Graph-tracking, no timing anomalies - with extensions from Sec. VI-C)
U-	Full virtual unrolling (No timing anomalies)

The results for **OC-** are not displayed here, because the graph-tracking is the most suitable method for the case without timing anomalies. As can be seen in Figure 5(a) **OT-** almost always (except for `minver`) reaches the same precision as **U-** ($100\% =$ **U-**). It also outperforms **F-** which does not analyze cyclic contexts (compare Section VI-B), but instead analyzes all the loop iterations with a fixed alignment and finally adds a penalty term to the result which accounts for the ignored actual alignment of the loop iterations. This leads to imprecision because the actual blocking time due to bus accesses may be much lower than the blocking time for the fixed-alignment situation plus the penalty.

In contrast to **OT-**, our general analyses **OC+** and **OT+** are less precise, which was expected, but still they outperform **F-** on benchmarks which show deeply nested loops or loops with high loop bounds, like for example `mergesort`, `edn`, `ludcmp` or `select`. On benchmarks which have a flat structure with many branches, like `statemate`, **OC+** and **OT+** are outperformed by **F-**, because they lose track of the offsets and must revert to worst-case assumptions. Nevertheless, even in those cases, they are still much more precise than the pessimistic assumption (**W**) that all bus accesses incur maximum delay, which results in an average WCET ratio of 414%. A surprising result is, that **OT+** is worse than **OC+** on average for the `MRTC` test bench subset. This is possible, because the global convergence analysis implicitly unrolls the first iterations as discussed in Section VI-B, whereas the graph-tracking analysis summarizes the iteration behavior in the offset graph. Therefore, once the offset information gets highly imprecise, the graph will be imprecise for all iterations, whereas the global convergence may achieve a better precision during its implicit unrolling. For loops with few iterations, this can have a strong impact on the precision of the WCET estimations. The graph tracking only shows its strength on the rather sparse graphs of **OT-**.

On `Debie` and `PapaBench` **F-** performs much better than on the `MRTC` test bench, because there are almost no nested loops and the loop bounds are rather small. Nevertheless, **F-** is still outperformed by **OT-**, and also **OT+** performs consistently better than **OC+** which emphasizes its applicability for realworld programs.

All presented results of the offset analyses use the offset interval representation, from Section V-A. Using the offset set representation the WCET estimation is further reduced by a maximum of 89% for `bsort100` (avg. 1.3%) when combined with graph-tracking, or by a maximum of 0.3% for `bs` (avg. 0.0%) when combined with the global convergence. This underlines the suitability of the combination of offset sets with the graph tracking analysis.

To evaluate the impact of different TDMA slot sizes or processor configurations on the precision of the WCET estimations, the analyses were performed for a varied number of cores n_c (with manually adapted task mapping) and varied TDMA slot lengths s_l. The average WCET results of these experiments are shown in Figure 5(b) where each

	F-	OC+	OT+	OT-	U-
MRTC	$0.4s$	$15.5s$	$927.3s$	$52.4s$	$1770.6s$
Debie	$0.6s$	$5.8s$	$400.6s$	$198.5s$	$1458.7s$
PapaBench	$0.05s$	$1.7s$	$1.7s$	$0.4s$	$0.05s$
Sum	$1.05s$	$23.0s$	$1329.6s$	$251.3s$	$3229.4s$

Table II
ANALYSIS TIME COMPARISON

configuration is described as a tuple $\langle n_c, s_l \rangle$. The experiment shows that **OT-** is able to compute results which are almost equal to those of **U-**, whereas the other analyses suffer from the increased maximum bus delay, **F-** even more so than **OC+** and **OT+**.

B. Analysis time

Table II summarizes the analysis duration in seconds for the WCET analyses that generated Figure 5(a). Here it becomes visible that all analyses are much faster in total than **U-**, which takes 53.8 minutes. **OT-** only requires 7.7% of that time and delivers WCET results which deviate by less than 1% from those of **U-**. Therefore **OT-** is the best choice when high analysis precision with moderate runtimes is required. For applications where an extremely short analysis time is required, **F-** can be better suited. It delivers results with 79% overestimation compared to **U-** in only 0.4% of the analysis time of **OT-**.

The unrolling is quick for benchmarks with few loops and low loop bounds like e.g. `PapaBench`. Since its analysis time is directly dependent on the loop structures and the loop bound values, it performs much worse for `Debie` and `MRTC` where nested loops and higher loop bounds are found (see Table I). This indicates that the unrolling is unsuitable for bigger realworld applications.

VIII. CONCLUSIONS

We have presented a new approach to the WCET analysis of TDMA-arbitrated shared resources, and applied it to a multicore system with shared bus. Our new analysis type is based on a static analysis of the TDMA offsets with which basic blocks may be entered and uses the key concept of cyclic contexts to improve the analysis precision. Concerning precision and analysis time, our solutions provide a good compromise between the fastest and the most precise approaches. The best variant (**OT-**) reduces the WCET overestimation by 79% compared to the quickest preexisting approach (**F-**) and achieves a speedup of 12.9 compared to the most precise preexisting approach (**U-**). Possible improvements to our methods are

- The integration of further microarchitectural analyses
- Specialized algorithms for the graph-based approach, possibly based on the polynomially solvable *Maximum Dynamic Flow* problem [12]
- A tailored graph clustering or graph expansion to fine-tune the precision of the graph-tracking analysis
- Heuristics which combine the presented analysis techniques to optimize runtime and precision

ACKNOWLEDGMENTS

This work was partially funded by the European Community's ArtistDesign Network of Excellence and by the European Community's 7^{th} Framework Program FP7/2007-2013 under grant agreement n^{o} 216008.

REFERENCES

[1] R. Wilhelm, J. Engblom, A. Ermedahl and others, "The worst-case execution-time problem - overview of methods and survey of tools," *ACM Transactions on Embedded Computing Systems*, vol. 7, no. 3, 2008.

[2] V. Suhendra and T. Mitra, "Exploring locking & partitioning for predictable shared caches on multi-cores," in *Proc. of DAC*, 2008.

[3] W. Zhang and J. Yan, "Accurately Estimating Worst-Case Execution Time for Multi-core Processors with Shared Direct-Mapped Instruction Caches," in *Proc. of RTCSA*, 2009.

[4] A. Gustavsson, A. Ermedahl, B. Lisper, and P. Pettersson, "Towards WCET Analysis of Multicore Architectures using UPPAAL," in *Proc. of WCET*, 2010.

[5] C. Pitter and M. Schoeberl, "A real-time Java chip-multiprocessor," *ACM Transactions on Embedded Computing Systems*, 2009.

[6] A. Andrei, P. Eles, Z. Peng, and J. Rosen, "Predictable Implementation of Real-Time Applications on Multiprocessor Systems-on-Chip," in *Proc. of VLSID*, 2008.

[7] R. Pellizzoni, A. Schranzhofer, J. Chen and others, "Worst case delay analysis for memory interference in multicore systems," in *Proc. of DATE*, 2010

[8] S. Chattopadhyay, A. Roychoudhury, and T. Mitra, "Modeling Shared Cache and Bus in Multi-cores for Timing Analysis," in *Proc. of SCOPES*, 2010.

[9] M. Paolieri, E. Quinones, F. J. Cazorla and others, "Hardware support for WCET analysis of hard real-time multicore systems," *SIGARCH Computer Architecture News*, vol. 37, no. 3, 2009.

[10] J. Mische, I. Guliashvili, S. Uhrig, and T. Ungerer, "How to Enhance a Superscalar Processor to Provide Hard Real-Time Capable In-Order SMT," *Architecture of Computing Systems*, vol. 5974/2010, 2010.

[11] D. Hardy and I. Puaut, "WCET Analysis of Multi-level Non-inclusive Set-Associative Instruction Caches," in *Proc. of RTSS*, 2008.

[12] M. Skutella, "An Introduction to Network Flows Over Time," *Research Trends in Combinatorial Optimization*, 2009.

[13] J. Reineke, B. Wachter, S. Thesing and others, "A definition and classification of timing anomalies," in *Proc. of WCET Workshop*, 2006.

[14] J. Reineke and Rathijit Sen, "Sound and Efficient WCET Analysis in the Presence of Timing Anomalies," in *Proc. of WCET Workshop*, 2009.

[15] Mälardalen WCET Research Group, "Mälardalen WCET Benchmark Suite," http://www.mrtc.mdh.se/projects/wcet.

[16] F. Nemer, H. Cassé, P. Sainrat and others, "PapaBench: A Free Real-Time Benchmark," in *Proc. of WCET Workshop*, 2006.

[17] ESA, "DEBIE - First Standard Space Debris Monitoring Instrument," http://gate.etamax.de/edid/publicaccess/debie1.php, 2008.

[18] T. Kelter, S. Chattopadhyay and others, "Bus-Aware Multicore WCET Analysis through TDMA Offset Bounds," Technical Report #837, Department of Computer Science, TU Dortmund University, 2011.