

# Extending Schedulability Tests of Tree-Shaped Transactions for TDMA Radio Protocols

Shuai Li<sup>\*†</sup>, Frank Singhoff<sup>†</sup>, Stéphane Rubini<sup>†</sup>, Michel Bourdellès<sup>\*</sup>

<sup>\*</sup>Thales Communications & Security, 4 av. des Louvresses, 92622 Gennevilliers, France

<sup>\*</sup>Email: {first-name}.{last-name}@fr.thalesgroup.com

<sup>†</sup>Lab-STICC/UMR 6285, UBO, UEB, 20 av. Le Gorgeu, 29200 Brest, France

<sup>†</sup>Email: {last-name}@univ-brest.fr

**Abstract**—In this article, a schedulability test is proposed for tree-shaped transactions with non-immediate tasks. A tree-shaped transaction is a group of precedence dependent tasks, partitioned on different processors, which may release several other tasks upon completion. When there are non-immediate tasks, tasks are not necessarily released immediately upon their predecessor’s completion. The schedulability test we propose is based on an existing test that does not handle non-immediate tasks directly. Simulation results show that tighter response time upper-bounds can be accessed when effects of non-immediateness are considered. Our schedulability test is motivated by real industrial TDMA systems developed at Thales, and experimental results show it provides less pessimistic schedulability results compared to current methods used by Thales system engineers.

## I. INTRODUCTION

In this article we propose a schedulability test, called WCDOPS+\_NIM, for tree-shaped transactions with non-immediate tasks. A tree-shaped transaction is a group of tasks that are related by precedence dependencies. This kind of transaction considers that a task can release one or several other tasks, and that a task can only have one predecessor task. In this article, these transactions may include non-immediate tasks. We call non-immediate tasks, tasks that are not necessarily released immediately after their predecessor completion time. Non-immediateness in task releases changes how combination of tasks can interfere and how task Worst Case Response Times (WCRTs) should be computed. WCDOPS+\_NIM is an adaptation of WCDOPS+ [14], a holistic schedulability test for tree-shaped transactions that does not handle non-immediate tasks directly.

Like previous works on holistic schedulability tests applied to real systems [11], WCDOPS+\_NIM is motivated by industrial TDMA Software Radio Protocols (TDMA SRP) [7] developed at Thales. A SRP is a software that implements a communication protocol (like TDMA) embedded in a radio station. These radio stations communicate in a mobile ad-hoc wireless network.

A TDMA SRP is a time-triggered [5] system because it has tasks released in time due to the nature of the TDMA protocol. Furthermore, task parameters depend on the TDMA protocol. A TDMA SRP is also an event-triggered [5] system due to tasks handling data/control flow in the radio protocol.

Numerous works [9] have been done previously to analyze schedulability of TDMA systems, but they only handle the

time-triggered aspect of such systems, and they do not consider task dependencies (e.g. shared resources). In our approach, we specify a TDMA SRP with the transaction model [16] and consider schedulability of such systems as a fixed-priority schedulability analysis problem. Indeed, in a transaction model both tasks released by other tasks (event-triggered) [13], [14], and tasks released in time (time-triggered) [16], can be modeled.

The rest of the article is organized as follows. Section II compares our work to existing schedulability tests for transactions. We then expose our system model and basic concepts in Section III. Section IV presents a TDMA SRP and illustrates issues when applying WCDOPS+ to such a system. Our test is then explained in Section V. The test is evaluated in Section VI by simulation, complexity discussion, and application to a real TDMA SRP. Finally we conclude with future works.

## II. RELATED WORK

Since the seminal processor utilization based [8] and response time based [3] schedulability tests for periodic tasks was proposed, the periodic task model has been extended with offset and jitter and new tests have been proposed.

In [16] the authors introduce the transaction model and a response time based schedulability test. The test uses a holistic approach to compute an upper-bound of the end-to-end response time between communicating tasks. The authors in [16] apply their test to a TDMA system, where tasks are released at different times. Although precedence dependencies between tasks of a same transaction are implicitly modeled, offsets and jitters are static, thus precedence dependencies aren’t fully specified [14].

In [12] the authors generalize the work in [16] for systems where task offset, jitter and deadline may be higher than their period. In this case, several instances of tasks in a transaction may interfere. The authors also introduce dynamic offset and jitter to fully model precedence dependency between tasks. Their test, called WCDO, computes an upper-bound to the WCRT of a task.

In [13] the authors of WCDO compute tighter upper-bounds by exploiting precedence dependencies between tasks. They notice that there exists execution conflicts, (i.e. conflicts

between tasks that can interfere together) and then propose a new test called WCDOPS to encompass this issue.

All previous tests are only applicable to linear transactions where a task can release one, and only one, other task. They cannot be applied to our system. The WCDOPS+ test in [14] adapts the original test for tree-shaped transactions, where a task can immediately release several other tasks upon completion. It also reduces further the pessimism of WCDOPS by observing new execution conflicts between tasks.

WCDOPS+ is extended in [10] for time partitioned systems and in [4] for graph-shaped transactions. The authors in [2] propose relative offsets and jitters to compute lower response time upper-bounds for tree-shaped transactions.

Our WCDOPS+\_NIM test is an adaptation of WCDOPS+. Unlike existing extensions of WCDOPS+, our test focuses on non-immediate tasks. Non-immediate tasks introduce new execution conflicts and thus modifies the way jitters, interference, and thus response times should be computed.

### III. SYSTEM MODEL AND BASIC CONCEPTS

Let us first expose our system model and some basic concepts from schedulability tests.

#### A. System Model

The studied system has several tasks that are allocated on processors. Once allocated on a processor, a task does not migrate to other processors. Each processor is scheduled according to a preemptive fixed-priority (FP) policy. We assume that the processors are accurate enough to neglect the effects of jittery coarse-grain clocks. Tasks may communicate either directly or using communication buses. Messages on communication buses are scheduled according to a preemptive FP policy. When tasks use a shared resource, we assume it is protected by a protocol [15] that makes it possible to bound the time a task is blocked.

The system is modeled with tree-shaped transactions (denoted  $\Gamma_i$ ) that group several tasks (denoted  $\tau_{ij}$ ). A transaction is released by a periodic event that occurs every  $T_i$  units of time. A particular instance of a transaction is called a job. When a tree-shaped transaction is released, an unique root task is released. The root task will lead to the release of other tasks, upon completion time. The root task is denoted  $\tau_{i1}$ .

A job of a task in a transaction is released after the event that releases the job of the transaction; if the event that releases the  $p^{th}$  job of  $\Gamma_i$  occurs at  $t_0$ , then the  $p^{th}$  jobs of its tasks are released at or after  $t_0$ . Each task is defined by the following parameters:  $C_{ij}$  is the Worst Case Execution Time (WCET).  $C_{ij}^b$  is the Best Case Execution Time (BCET).  $O_{ij}$  is the offset, i.e. the  $p^{th}$  job of  $\tau_{ij}$  is released at earliest  $O_{ij}$  units of time after  $t_0$ .  $J_{ij}$  is the maximum jitter, i.e. the  $p^{th}$  job of  $\tau_{ij}$  is released in  $[t_0 + O_{ij}; t_0 + O_{ij} + J_{ij}]$ .  $D_{ij}$  is the global deadline (relative to  $t_0$ ) [12], i.e. the WCRT of the  $p^{th}$  job of  $\tau_{ij}$  must be lower or equal to  $D_{ij}$ .  $B_{ij}$  is the maximum shared resource blocking time [15].  $prio(\tau_{ij})$  is the fixed priority of  $\tau_{ij}$ . Finally  $proc(\tau_{ij})$  is the processor on which  $\tau_{ij}$  is allocated on. A transaction can have tasks allocated on different processors.

We want to compute an upper-bound of the WCRT of a task  $\tau_{ij}$ , denoted  $R_{ij}^w$ . Similarly, a task's best case response time is denoted  $R_{ij}^b$ . In this article,  $R_{ij}^w$  and  $R_{ij}^b$  are expressed as values relative to the release time of  $\Gamma_i$ .

Tasks in a transaction are related by precedence dependencies. A precedence dependency between two tasks, denoted  $\tau_{ip} \prec \tau_{ij}$ , is a constraint that means that  $\tau_{ip}$  must complete execution before  $\tau_{ij}$  can be released.

A task  $\tau_{ij}$ , of a tree-shaped transaction, is said to have one direct predecessor, denoted  $pred(\tau_{ij})$ , and a set of direct successors, denoted  $succ(\tau_{ij})$ . A task  $\tau_{ix}$  is  $pred(\tau_{ij})$  (resp. in  $succ(\tau_{ij})$ ) if there is no task  $\tau_{iy}$  such that  $\tau_{ix} \prec \tau_{iy} \prec \tau_{ij}$  (resp.  $\tau_{ij} \prec \tau_{iy} \prec \tau_{ix}$ ). For the root task,  $pred(\tau_{i1})$  is undefined.

Direct predecessors and successors may be non-immediate:

*Definition 1 (Non-Immediateness):* A task  $\tau_{ix}$  and its direct successor task  $\tau_{iy}$  are said to be non-immediate tasks if  $\tau_{ix} \prec \tau_{iy} \wedge O_{iy} > O_{ix} + C_{ix}^b$ . Task  $\tau_{ix}$  is called a *non-immediate predecessor* and  $\tau_{iy}$  a *non-immediate successor*.

When analyzing a particular task  $\tau_{ab}$ , the set  $hp_i(\tau_{ab})$  is the set of tasks in transaction  $\Gamma_i$  with a priority higher than or equal to  $prio(\tau_{ab})$  and allocated on the same processor as  $\tau_{ab}$ . A respective definition is given for lower priority tasks  $lp_i(\tau_{ab})$ .

For readability issues, when analyzing  $\tau_{ab}$  we will sometimes note  $hp_i(\tau_{ab})$  by  $hp_i$ , and similarly for any other notation that has the  $\tau_{ab}$  parameter.

#### B. Basic Concepts

**Busy period** [6]. A  $\tau_{ab}$  busy period is the time interval during which the processor is busy executing tasks in  $hp_i$  and  $\tau_{ab}$ . The length of a  $\tau_{ab}$  busy period is denoted  $w$ .

**Critical instant** [16]. The time at which a  $\tau_{ab}$  busy period starts, is called the critical instant, denoted  $t_c$ .

**Worst case scenario** [12]. In [12] the authors show that the maximum interference from a transaction  $\Gamma_i$  to a  $\tau_{ab}$  busy period occurs when the release of  $\Gamma_i$  is phased such that some task  $\tau_{ik} \in hp_i$  is released at  $t_c$  after having experienced its maximum release jitter  $J_{ik}$ . We say that  $\tau_{ik}$  starts the  $\tau_{ab}$  busy period and we created a (worst case) scenario (candidate). Jobs of tasks before/at  $t_c$  must experience enough jitter to be released at  $t_c$  and jobs of tasks after  $t_c$  must not experience jitter.

**Transaction phasing** [12]. When  $\tau_{ik}$  starts the  $\tau_{ab}$  busy period, the phasing of jobs of  $\Gamma_i$  can be determined. Fig. 1 shows parameters of the phasing of jobs of  $\Gamma_i$ .

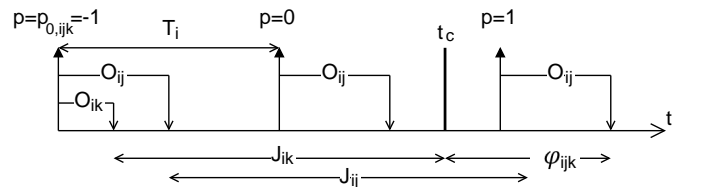


Fig. 1. Transaction Phasing

A job number  $p$  is assigned to a job of  $\Gamma_i$  according to the job's release time. Jobs  $p \leq 0$  are released before or at  $t_c$  and jobs  $p > 0$  are released after  $t_c$ . For a particular task  $\tau_{ij} \in \Gamma_i$ , the first job after  $t_c$  ( $p = 1$ ) is released at  $\varphi_{ijk}$  [12]:

$$\varphi_{ijk} = T_i + O_{ij} - (O_{ik} + J_{ik}) \bmod T_i \quad (1)$$

The first pending job of  $\tau_{ij}$  at  $t_c$  is numbered  $p_{0,ijk}$  [12]:

$$p_{0,ijk} = 1 - \left\lfloor \frac{J_{ij} + \varphi_{ijk}}{T_i} \right\rfloor \quad (2)$$

**Execution conflicts** [13]. When analyzing a task  $\tau_{ab}$ , not all tasks in  $hp_i$  are eligible to execute within the same  $\tau_{ab}$  busy period. This is called an execution conflict. Execution conflicts are due to priority schemes [13]. For example consider  $\tau_{ij} \prec \tau_{ij+1} \prec \tau_{ij+2}$  with  $\tau_{ij+2}, \tau_{ij} \in hp_i$  and  $\tau_{ij+1} \in lp_i$ . Tasks  $\tau_{ij}$  and  $\tau_{ij+2}$  cannot execute within the same  $\tau_{ab}$  busy period. To solve execution conflicts, some tasks are grouped into sets (e.g. tasks that must all execute within a same  $\tau_{ab}$  busy period).

**Holistic approach** [16], [12]. Tests in [12], [13], [14] are based on the holistic approach proposed in [16]. In this approach, the system starts in an initial state where task parameters are set according to precedence dependencies [12]  $\tau_{ip} \prec \tau_{ij}$ :

$$O_{ij} = R_{ip}^b \quad (3)$$

$$R_{ij}^b = O_{ij} + C_{ij}^b \quad (4)$$

$$J_{ij} = R_{ip}^w - O_{ij} \quad (5)$$

$$R_{ip}^w = O_{ip} + C_{ip}^w \quad (6)$$

WCRTs ( $R_{ij}^w$ ) are then updated and they may increase jitters. Since jitters and WCRTs are dependent, the holistic approach algorithm is iterative: response times and jitters are updated until the system comes to a stable state where no values are modified.

#### IV. APPLICABILITY OF WCDOPS+ ON A TDMA SRP

In this section we introduce TDMA SRPs, our motivational system. We show how an example of such system is modeled with a tree-shaped transaction, with non-immediate tasks. Then we try to apply directly the original WCDOPS+ test to the example and discuss the resulting analysis.

##### A. TDMA Software Radio Protocol

From a system point of view, a SRP is divided into several layers, according to the OSI model for communication systems. In the layers, tasks implement the radio protocol.

Tasks implementing a TDMA protocol are constrained by a *TDMA Frame*. A *TDMA Frame* is divided into several time slots of different types, durations, and modes. The duration of a *TDMA Frame* is the sum of durations of its slots. A *TDMA Configuration* defines the combination of slots (type and mode) in a *TDMA Frame*. We assume to analyze a particular *TDMA Configuration*.

Fig. 2 shows an example of tasks in two layers  $L1$  and  $L2$ , and a *TDMA Frame* with two slots.

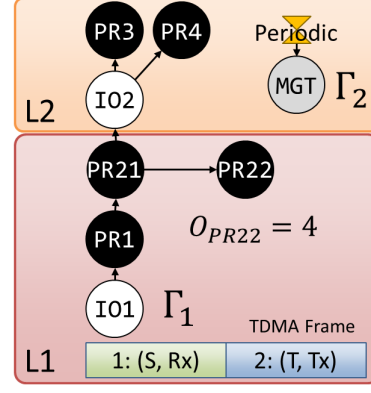


Fig. 2. TDMA SRP Example: Black circles are high-priority tasks; Gray circles are mid-priority tasks; White are low-priority tasks; "IO" are input-output tasks; "PR" are processing tasks; "MGT" is a management task; Slots have a duration of 4; Slot "1: (S, Rx)" is of type Service (S), mode Reception (Rx); Slot "2: (T, Tx)" is of type Traffic (T), mode Transmission (Tx)

In Fig. 2, transaction  $\Gamma_1$  is used to model tasks constrained by the *TDMA Frame*. Clearly  $\Gamma_1$  is a tree-shaped transaction. Task  $IO1$  is released at slot 1.  $IO1$  then leads to the releases of other tasks. Task  $PR22$  cannot be released earlier than slot 2. It can be released only if  $PR21$  has finished execution, because of some data dependency.  $IO$  tasks have lower priorities than processing tasks so input-output operations won't preempt processing operations. Transaction  $\Gamma_2$  has a single  $MGT$  task that is released periodically. Generally this kind of task has a period greater or equal to the *TDMA Frame* duration, and it must not be delayed by more than a *TDMA Frame* duration. For this reason,  $MGT$  has a priority higher than  $IO$  tasks, to ensure that  $MGT$  won't be preempted too long after its release.

##### B. Applying WCDOPS+ Directly

Let us apply WCDOPS+ directly to the system in Fig. 2. We assume that each task of  $\Gamma_i$  has a WCET of 1. Then we have  $PR22$  a non-immediate successor of  $PR21$ , because  $PR22$  is released at earliest at  $t = 4$  and  $PR21$  can complete execution as early as  $t = 3$ .

If WCDOPS+ is applied directly for the analysis of  $MGT$ , since non-immediateness is not handled by WCDOPS+, the test considers that  $PR22$  can only execute with, at most,  $PR1$  and  $PR21$  in the same  $MGT$  busy period. This is not true as shown by one possible schedule in Fig. 3. If, for example,  $PR3$  has a WCET equal to 2 instead, the interference of  $\Gamma_1$  to  $MGT$  is underestimated.

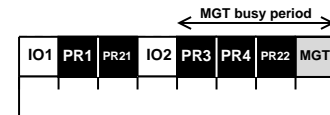


Fig. 3.  $PR3$ ,  $PR4$ , and  $PR22$  in same  $MGT$  busy period

A possible solution to better identify combinations of tasks that can interfere, is to model the non-immediateness between

two tasks by *ghost intermediate tasks*. The concept of *ghost tasks* is introduced in [14], while the concept of *intermediate tasks* is proposed in [1]. A *ghost intermediate task* can be defined as:

*Definition 2:* A *ghost intermediate task*  $\tau_{ixy}$  between a task  $\tau_{ix}$  and its non-immediate direct successor  $\tau_{iy}$  ( $\tau_{ix} \prec \tau_{iy}$ ) is one that is allocated alone on a unique processor. It is defined as follows:  $C_{ixy}^a = C_{ixy}^b = O_{iy} - (O_{ix} + C_{ix}^b)$ ;  $O_{ixy} = O_{ix} + C_{ix}^b$ ;  $J_{ixy} = R_{ix}^w - O_{ixy}$ ;  $D_{ixy} = \infty$ ;  $B_{ixy} = 0$ ;  $prio(\tau_{ixy}) = 1$ ;  $proc(\tau_{ixy})$  is unique. Precedence dependency  $\tau_{ix} \prec \tau_{iy}$  is replaced by  $\tau_{ix} \prec \tau_{ixy} \prec \tau_{iy}$ .

Adding *ghost intermediate tasks* introduces pessimism to WCRTs computation. For example if a *ghost intermediate task* is added between *PR21* and *PR22*, any increase in  $R_{PR21}^w$  will increase  $J_{PR22}$  and thus  $R_{PR22}^w$ . This is not always the case, as shown by a possible schedule in Fig. 4, so  $R_{PR22}^w$  can be overestimated.

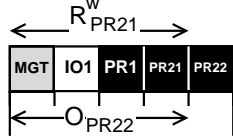


Fig. 4.  $R_{PR21}^w$  increases due to *MGT* preempting *IO1* but  $J_{PR22}$  does not increase. ( $C_{MGT} = 1$ )

Furthermore, with a *ghost intermediate task*, the test considers that *PR3* and *PR4* can interfere *PR22* even if *PR22* experiences jitter, since *IO2* is allowed to execute during the execution of the *ghost intermediate task* on another processor. Then both  $J_{PR22}$  and interference from *PR3* and *PR4*, contribute to  $R_{PR22}^w$ . This is not possible as shown by the schedule in Fig 5, so  $R_{PR22}^w$  can again be overestimated.

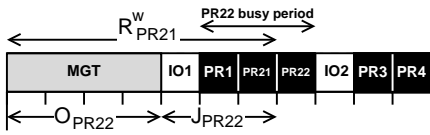


Fig. 5. *PR22* experiences jitter, due to increase in  $R_{PR21}^w$ , but *PR3* and *PR4* cannot interfere *PR22*. ( $C_{MGT} = 4$ )

### C. Conclusion on Applicability

To conclude this section, two problems are observed when applying WCDOPS+ to our TDMA SRP modeled with transactions with non-immediate tasks. First, if applied directly, tasks interference may be underestimated. Second, by modeling non-immediate tasks with *ghost intermediate tasks*, jitter and task interference can both be overestimated. In conclusion both problems lead to WCRTs that may be over or underestimated. In the following section we show how our test proposes to solve these problems by considering the effects of non-immediateness directly.

## V. WCDOPS+\_NIM

WCDOPS+\_NIM adapts the original feasibility test by considering the effects of non-immediateness.

### A. Overview of the Analysis

Fig. 6 gives an overview of the WCDOPS+\_NIM algorithm for the analysis of  $\tau_{ab}$  during an iteration of the holistic algorithm. The approach is inherited from [16], [12], [14].

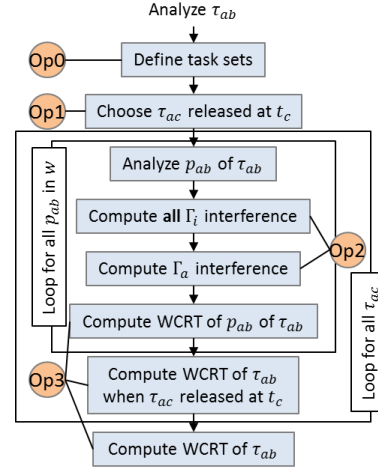


Fig. 6. WCDOPS+\_NIM Overview: Circles indicate key operations

Some task sets are first defined to help the analysis of  $\tau_{ab}$  (**Op0**). The idea is to compute the WCRT of  $\tau_{ab}$  for each scenario where a  $\tau_{ac}$  starts the  $\tau_{ab}$  busy period (**Op1**). Within a scenario, the WCRT of each job  $p_{ab}$  of  $\tau_{ab}$  in the  $\tau_{ab}$  busy period is computed. The length of the busy period  $w$  can be estimated [12]. To compute the WCRT of  $p_{ab}$  of  $\tau_{ab}$ , interference from transactions in the system (**Op2**) is computed. The WCRT of  $\tau_{ab}$  is then the maximum of WCRTs of each job  $p_{ab}$  of each scenario (**Op3**).

Once the WCRT of each task  $\tau_{ab}$ , in the system, is computed, jitters are updated. Convergence is then checked and if any values are modified, we go on to the next iteration of the holistic analysis.

The algorithm has several key operations. In the following sections each key operation in Fig. 6 is explained.

### B. (Op0) Task Sets and Execution Conflicts

**Op0** consists in defining some task sets to help the analysis of  $\tau_{ab}$  when resolving execution conflicts.

Two sets of tasks are defined in [14]. An H segment is a set of tasks that must all execute within a same  $\tau_{ab}$  busy period. Two tasks in  $hp_i$  belong to the same H segment if there is no other task that is not in  $hp_i$  that precedes one but not the other. An H section is a set of tasks that may execute in the same  $\tau_{ab}$  busy period. Two tasks in  $hp_i$  belong to the same H section if there is no other task in  $lp_i$  that precedes one but not the other.

We now adapt these sets. Let us first integrate the definition of non-immediateness (Definition 1) in the  $\text{IM}$  function in

Algorithm 1. This function checks if the successor  $\tau_{ij}$  of a task  $\tau_{ip}$  is immediate.

---

**Algorithm 1** Immediate Function

---

```

1: function IM( $\tau_{ip}, \tau_{ij}$ )
2:   return  $\tau_{ip} = \text{undefined} \vee O_{ij} > O_{ip} + C_{ip} \vee \text{IS\_IMMEDIATE}(\tau_{ij})$ 
3: end function

```

---

$\text{IS\_IMMEDIATE}(\tau_{ij})$  returns true if we decide to make  $\text{IM}(\tau_{ip}, \tau_{ij})$  always return true. Otherwise it returns false. Without loose of generality, this will simplify explanations of algorithms later in this article.

Let us re-define an H segment by modifying its conditions: there is also no non-immediate predecessor that precedes one of the task but not the other, and there is no non-immediate predecessor that is a direct predecessor of both tasks. To formally re-define an H segment, we use some notations from [14].  $\Gamma_{ij}$  is a sub-transaction and is the set of all tasks, in  $\Gamma_{ij}$ , preceding  $\tau_{ij}$  and  $\tau_{ij}$  itself.  $\Gamma_{ij} \Delta \Gamma_{ik}$  is the symmetric difference between two sub-transactions. Formally, the new definition of an H segment is then:

$$H_{ij}^{seg}(\tau_{ab}) = \{ \tau_{ik} \mid \tau_{ik} \in hp_i(\tau_{ab}) \wedge (\neg \exists \tau_{il} \in \Gamma_{ij} \Delta \Gamma_{ik} \mid \tau_{il} \notin hp_i(\tau_{ab}) \vee \neg \text{IM}(\text{pred}(\tau_{il}), \tau_{il})) \} \quad (7)$$

The definition of an H section does not need any modification since a non-immediate successor may belong to the same busy period as its predecessor (both in  $hp_i$ ).

Let us consider again the system in Fig. 2, assuming *MGT* is under analysis. Sets (*PR1*, *PR21*), (*PR3*, *PR4*), and (*PR22*) are H segments. Set (*PR1*, *PR21*, *PR22*) is an H section.

**C. (Op1) Worst Case Scenario**

**Op1** consists in creating a scenario where  $\tau_{ik} \in \Gamma_i$  (resp.  $\tau_{ac} \in \Gamma_a$ ) starts the  $\tau_{ab}$  busy period at  $t_c$ .

In [14], tasks in  $\Gamma_i$  that may start the busy period are in a set  $XP_i(\tau_{ab})$ , which is the set of tasks that come first in their respective H segments.

We re-define  $XP_i$ : a set that contains tasks in  $hp_i$  whose predecessors are not in  $hp_i$  but also tasks in  $hp_i$  that are non-immediate successors.

$$XP_i(\tau_{ab}) = \{ \tau_{if} \in hp_i(\tau_{ab}) \mid \text{pred}(\tau_{if}) \notin hp_i(\tau_{ab}) \vee \neg \text{IM}(\text{pred}(\tau_{if}), \tau_{if}) \} \quad (8)$$

For example in Fig. 2, *PR1*, *PR22*, *PR3* and *PR4* are in  $XP_i(\text{MGT})$ .

Let us now assume that a task  $\tau_{ik} \in XP_i$  starts a  $\tau_{ab}$  busy period. If  $\tau_{ik}$  is not an immediate successor then the following theorem applies if  $\tau_{ik}$  starts the  $\tau_{ab}$  busy period.

*Theorem 1:* Let  $\tau_{ik}$  be a task in  $XP_i$  that starts a  $\tau_{ab}$  busy period. If  $\tau_{ik}$  is a non-immediate successor, and  $\text{pred}(\tau_{ik}) \in hp_i$ , then  $\tau_{ik}$  must not experience jitter to be released at  $t_c$ . If  $\text{pred}(\tau_{ik}) \notin hp_i$ , the scenario where  $\tau_{ik}$  experiences maximum jitter to be released at  $t_c$  is also analyzed.

*Proof:* We assume  $\tau_{ik} \in XP_i$  is a non-immediate successor that starts the  $\tau_{ab}$  busy period. If  $\tau_{ik}$  experiences jitter in a scenario, then it means that the response time of  $\text{pred}(\tau_{ik})$  is greater than the offset of  $\tau_{ik}$ .  $\tau_{ik}$  is then released immediately after  $\text{pred}(\tau_{ik})$  completes execution. If  $\text{pred}(\tau_{ik}) \in hp_i(\tau_{ab})$  then, according to Lemma 5-1 in [13],  $\tau_{ik}$  cannot start the busy period, which contradicts our assumption. The case where  $\tau_{ik}$  experiences its maximum jitter will be analyzed in the scenario where the H segment of  $\text{pred}(\tau_{ik})$  starts the  $\tau_{ab}$  busy period. If  $\text{pred}(\tau_{ik}) \notin hp_i(\tau_{ab})$  then there is no predecessor H segment that may start the busy period with  $\tau_{ik}$  experiencing jitter. This is why, in this case, the scenario where  $\tau_{ik}$  starts the busy period, after having experienced  $J_{ik}$ , is also analyzed. ■

To create a scenario where  $\tau_{ij}$  is released at  $t_c$  without experiencing jitter,  $J_{ij}$  is set to 0. This is what we call *jitter canceling*. In this case, we say that  $J_{ij}$  is canceled. To integrate *jitter canceling* in the schedulability test, whenever we loop through  $\tau_{ik}$  tasks in  $XP_i$  to create scenarios, we memorize the original value of  $J_{ik}$ , set  $J_{ik}$  to 0, compute interference for the scenario, and afterwards reset  $J_{ik}$  to the memorized value. We also create the scenario where  $J_{ik}$  is not canceled if  $\text{pred}(\tau_{ik}) \notin hp_i$ , so interference for both scenarios can be compared.

**D. (Op2) Worst Case Interference**

**Op2** consists in computing the interference from transactions to job  $p_{ab}$  of  $\tau_{ab}$ .

Like WCDOPS+, our test computes two kinds of interference for a transaction: blocking and non-blocking [14]. The existence of blocking interference is due to execution conflicts. Only one blocking interference from any transaction in the system, can contribute to the  $\tau_{ab}$  busy period. If a transaction's blocking interference is not chosen to contribute, then its non-blocking interference contributes to the  $\tau_{ab}$  busy period.

In the following first two sections, we show how to compute interference of jobs of a single transaction  $\Gamma_i$  before/at  $t_c$ , then jobs after  $t_c$ . In these sections, it is assumed that task  $\tau_{ik}$  from  $XP_i$  starts the  $\tau_{ab}$  busy period, of length  $w$ , at  $t_c = 0$ . Afterwards we show how to compute the total interference: interference from  $\Gamma_a$  and from all other transactions  $\Gamma_i \neq \Gamma_a$ .

1) *Jobs before and at  $t_c$  ( $p \leq 0$ ):* The interference of jobs  $p \leq 0$  are computed with three functions in [14]: **(F1)** Compare/sum interference of each job  $p \leq 0$  of  $\Gamma_i$ ; **(F2)** Compute interference of a particular job  $p$  of  $\Gamma_i$ ; **(F3)** Compute interference of a particular task of job  $p$  of  $\Gamma_i$ . In the following paragraphs we modify these three functions for non-immediate tasks.

a) **(F1) TransactionInterference:** Interference from jobs before/at  $t_c$  is computed by the `TransactionInterference` [14] function. This function returns a transaction's blocking and non-blocking interference. It loops through each pending job  $p$  of  $\Gamma_i$  before/at  $t_c$  that may interfere. Assuming tasks are ordered by increasing offsets in  $\Gamma_i$ , the first pending job of  $\Gamma_i$  that may interfere is the first pending job of its last task's H segment:  $p_{0,iNk}^{seg}(\tau_{ab})$  [14] computed by Equation 2 applied to the first

task of  $H_{iN}^{seg}$ , with  $\tau_{iN}$  the last task of  $\Gamma_i$ . Algorithm 2 shows our modification of TransactionInterference.

### Algorithm 2 TransactionInterference Function

---

```

1: function TRANSACTIONINTERFERENCE( $\tau_{ab}, \tau_{ik}, w$ )
2:   Add ghost root task  $\tau_{i0}$  as predecessor to  $\tau_{i1}$ 
3:
4:   for  $p$  in  $p_{0,i}^{seg}(\tau_{ab})..0$  do
5:     for  $\tau_{ij} \in \Gamma_i$  do
6:       if  $\tau_{ij} \in hp_i(\tau_{ab}) \wedge \neg \text{IM}(\text{pred}(\tau_{ij}), \tau_{ij}) \wedge (\varphi_{ijk} + (p-1) \times$ 
 $T_i) < 0$  then
7:         Make IS_IMMEDIATE( $\tau_{ij}$ ) return true
8:       end if
9:     end for
10:
11:   [jobI, jobDelta]  $\leftarrow$  BranchInterference( $\tau_{ab}, \tau_{ik}, \tau_{i0}, w, p$ )
12:   trans_NoB  $\leftarrow$  trans_NoB + jobI
13:   transDelta  $\leftarrow$  max(transDelta, jobDelta)
14:
15:   for  $\tau_{ij} \in \Gamma_i$  do
16:     Make IS_IMMEDIATE( $\tau_{ij}$ ) return false
17:   end for
18: end for
19:
20: transI_B  $\leftarrow$  transI_NoB + transDelta
21: return [transI_NoB, transI_B]
22: end function

```

---

Before computing the interference of job  $p$  of  $\Gamma_i$ , we check which non-immediate successors  $\tau_{ij}$ , at job  $p$ , are released immediately (lines 5 to 9) in the scenario where  $\tau_{ik}$  starts the  $\tau_{ab}$  busy period. Checking if a non-immediate successor  $\tau_{ij} \in hp_i$  is to be considered immediately released by  $\text{pred}(\tau_{ij}) \in hp_i$  also determines to which H segment  $\tau_{ij}$  belongs to in the given scenario: either the H segment of  $\tau_{ij}$  or the H segment of  $\text{pred}(\tau_{ij})$ . This has an effect on blocking interference computation.

*Theorem 2:* A non-immediate successor task  $\tau_{ij} \in hp_i$  is released immediately by  $\text{pred}(\tau_{ij})$ , at job  $p$ , when  $\tau_{ik}$  starts the  $\tau_{ab}$  busy period, if  $\tau_{ij}$  is released before  $t_c = 0$ :

$$(\varphi_{ijk} + (p-1) \times T_i) < 0$$

*Proof:* Let  $\tau_{ij} \in hp_i$  be a non-immediate successor. Value  $\varphi_{ijk} + (p-1) \times T_i$  is the release time of  $\tau_{ij}$  at job  $p$ , when  $\tau_{ik}$  starts the  $\tau_{ab}$  busy period. If  $\tau_{ij}$  is released before  $t_c = 0$ ,  $\tau_{ij}$  needs to have experienced enough jitter to be released at  $t_c$  [13]. If  $\tau_{ij}$  experiences jitter, then  $\tau_{ij}$  is immediately released by  $\text{pred}(\tau_{ij})$ . ■

For example, in Fig. 5, if  $PR1$  starts a busy period after having experienced  $J_{PR1} = 4$ ,  $PR22$  is released at  $t = -1$  and experiences a jitter of 1 to be released at  $t_c$ .  $PR22$  is thus released immediately by  $PR21$  and belongs to the same H segment as  $PR21$ .

*b) (F2) BranchInterference:* To compute the interference of a particular job  $p \leq 0$  of  $\Gamma_i$ , since the transaction is tree-shaped, the tree is explored by a depth-first search algorithm in the BranchInterference [14] function. The tree is explored by branches defined by tasks denoted  $\tau_{iB}$  (our modified formal definition below). The general idea is to compute interference of a branch and compare/sum it with interference from branches that arrive after it in the tree (called sub-branches  $SB$  in the algorithm). Algorithm 3 shows our modification of BranchInterference.

### Algorithm 3 BranchInterference Function

---

```

1: function BRANCHINTERFERENCE( $\tau_{ab}, \tau_{ik}, \tau_{iB}, w, p$ )
2:    $SB \leftarrow \text{succ}(\tau_{iB})$ 
3:   if  $\exists \tau_{im} \in SB \mid \tau_{im} \in hp_i(\tau_{ab}) \wedge \text{IM}(\tau_{iB}, \tau_{im})$  then
4:      $S \leftarrow \{\tau_{il} \in H_{im}(\tau_{ab}) \mid \tau_{iB} < \tau_{il}\}$ 
5:     sectionI  $\leftarrow \sum_{\tau_{ij} \in S} \text{TaskInterference}(\tau_{ab}, \tau_{ik}, \tau_{ij}, w, p)$ 
6:      $SB \leftarrow \{SB \cup \text{succ}(H_{im}^{seg}(\tau_{ab}))\} \setminus \{\text{succ}(\tau_{iB}) \cap H_{im}^{seg}(\tau_{ab})\}$ 
7:   end if
8:
9:   if  $\tau_{iB} \in hp_i(\tau_{ab})$  then
10:    sectionI  $\leftarrow$  sectionI + TaskInterference( $\tau_{ab}, \tau_{ik}, \tau_{iB}, w, p$ )
11:   end if
12:
13:   for  $\tau_{iS} \in SB$  do
14:     [bI, bD]  $\leftarrow$  BranchInterference( $\tau_{ab}, \tau_{ik}, \tau_{iS}, w, p$ )
15:     subBranchesI  $\leftarrow$  subBranchesI + bI
16:     subBDelta  $\leftarrow$  max(subBDelta, bD)
17:   end for
18:
19:   if  $\tau_{iB} \in lp_i(\tau_{ab})$  then
20:     branchI  $\leftarrow$  subBranchesI
21:     branchDelta  $\leftarrow$  max(sectionI - subBranchesI, subBDelta)
22:     if  $\neg \text{IM}(\text{pred}(\tau_{iB}), \tau_{iB})$  then
23:       branchDelta  $\leftarrow$  max(branchDelta, 0)
24:     end if;
25:   else
26:     branchI  $\leftarrow$  max(sectionI, subBranchesI)
27:     branchDelta  $\leftarrow$  max(subBranchesI + subBDelta - branchI, 0)
28:   end if
29:
30:   return [branchI, branchDelta]
31: end function

```

---

In Algorithm 3, the modified definition of a branch-defining task  $\tau_{iB}$  is:

$$\tau_{iB} \notin hp_i \vee \neg \text{IM}(\text{pred}(\tau_{iB}), \tau_{iB})$$

For example, in Fig. 2,  $IO1$ ,  $IO2$ , and  $PR22$  define branches, when analyzing  $MGT$ .

Compared to [14], sub-branches of  $\tau_{iB}$  ( $SB$ ) can now contain  $hp_i$  tasks. For example, in Fig. 2,  $IO2$  and  $PR22$  are in  $SB$  of the branch defined by  $IO1$ .

Due to the existence of non-immediate tasks, the exploration and computation of interference need some modifications on lines 3, 10, and 22. These modifications model the existence *ghost intermediate task* between a task  $\tau_{ij}$  and its non-immediate successor  $\text{succ}(\tau_{ij})$ , so correct values are returned by the function.

*c) (F3) TaskInterference:* When computing interference of a particular job  $p \leq 0$  of  $\Gamma_i$ , each task's interference is computed by the TaskInterference function [14]. This function returns the task's WCET if it can interfere. A task can interfere if it is released in  $[0, w)$  [12] and if it passes a number of *reduction rules* [13] that eliminate execution conflicts.

We add a new *reduction rule* to the original ones in [14], according to the following theorem:

*Theorem 3:* Let  $\tau_{ik}$  be a non-immediate successor that starts the  $\tau_{ab}$  busy period at  $t_c$ , with  $J_{ik}$  canceled. A job  $p$  of a task  $\tau_{ij} \in hp_i$ , that precedes  $\tau_{ik}$ , does not interfere the  $\tau_{ab}$  busy period if  $p \leq p_{0,ikk}^{seg}$ .

*Proof:* We assume that  $\tau_{ik} \in XP_i$  is a non-immediate successor that starts the  $\tau_{ab}$  busy period, and  $J_{ik}$  is canceled. Task  $\tau_{ik}$  does not experience jitter and is released at  $t_c$ . If



any job, earlier or same as  $p_{0,ikk}^{seg}$ , of a task, that precedes  $\tau_{ik}$ , executes in the  $\tau_{ab}$  busy period, then the task executes after  $t_c$ . If a preceding task executes after  $t_c$ , then  $\tau_{ik}$  is not released at  $t_c$ . This contradicts our assumption that  $\tau_{ik}$  is released at  $t_c$ . ■

Our new *reduction rule* is formally defined as:

$$\tau_{ij} \prec \tau_{ik} \wedge p < p_{0,ikk}^{seg} \wedge \neg \text{IM}(\text{pred}(\tau_{ik}), \tau_{ik}) \wedge J_{ik} = 0$$

For example, in Fig. 3, let us assume  $PR22$  starts a busy period with  $J_{PR22}$  canceled.  $PR22$  is released at  $t_c$  so tasks  $PR1$  and  $PR21$  must have completed execution before  $t_c$  or  $PR22$  is not released at  $t_c$ .

2) *Jobs after  $t_c$  ( $p > 0$ ):* For jobs  $p > 0$ , the original test does not need any modification. Indeed, jobs  $p > 0$  of  $\tau_{ij}$  can only interfere  $\tau_{ab}$  if  $\tau_{ij}$  belongs to the first H section of  $\Gamma_i$ , and the first H segment is not preceded by a task in  $lp_i$  [14].

If a non-immediate successor  $\tau_{ij} \in hp_i$ , of  $\text{pred}(\tau_{ij}) \in hp_i$ , is in the first H section, we do not need to check if  $\tau_{ij}$  belongs its own H segment or the H segment of  $\text{pred}(\tau_{ij})$ , because both H segments belong to the first H section and the rule in [14], for jobs  $p > 0$ , applies.

3) *Total interference:* In [14] the interference from  $\Gamma_a$  on a  $\tau_{ab}$  busy period is computed the same way as for  $\Gamma_i$ , only with more *reduction rules*. Thus computation of  $\Gamma_a$  interference needs the same kind of modifications as those for  $\Gamma_i$  (replacing respectively  $\tau_{aj}$  for  $\tau_{ij}$  and  $\tau_{ac}$  for  $\tau_{ik}$  when necessary).

When computing interference from  $\Gamma_i \neq \Gamma_a$  to job  $p_{ab}$  of  $\tau_{ab}$  the upper-bound of the interference is computed as the maximum interference computed for each scenario  $\tau_{ik}$ . The upper-bound of the non-blocking interference is denoted  $W_i^*$  [14], and the upper-bound of the blocking interference is expressed as an interference increase, denoted  $\Delta W_i^*$  [14].

When computing interference from  $\Gamma_a$  to job  $p_{ab}$  of  $\tau_{ab}$  the non-blocking interference of  $\Gamma_a$  (denoted  $W_{ac}$ ) and the interference increase (denoted  $\Delta W_{ac}$ ) are computed for a given  $\tau_{ac}$ , and not as upper-bounds [14].

### E. (Op3) Worst Case Response Time

**Op3** consists in computing the WCRT of  $\tau_{ab}$  from the WCRTs computed for each of its jobs  $p_{ab}$  in each of the scenarios  $\tau_{ac}$ .

The WCRT of  $p_{ab}$  of  $\tau_{ab}$  [14] is:

$$R_{abc}^w(p_{ab}) = w_{abc}(p_{ab}) - (\varphi_{abc} + (p_{ab} - 1)T_a) + O_{ab} \quad (9)$$

$$w_{abc}(p_{ab}) = B_{ab} + W_{ac} + \sum_{\forall i \neq a} W_i^* + \max(\Delta W_{ac}, \Delta W_i^*) \quad (10)$$

The WCRT of  $\tau_{ab}$  is then  $R_{ab}^w$ : maximum  $R_{abc}^w(p_{ab})$  for all  $\tau_{ac}$  that start the  $\tau_{ab}$  busy period.

Our modifications introduced to the general algorithm, has a consequence on the contribution of jitter to a WCRT computed for a scenario. When we create the scenario  $\tau_{ac} = \tau_{ab}$ , if  $\tau_{ab}$

is a non-immediate successor,  $J_{ab}$  is canceled.  $J_{ab}$  will then not contribute to the WCRT of  $\tau_{ab}$  computed for the scenario  $\tau_{ac} = \tau_{ab}$ . Otherwise the WCRT computed for  $\tau_{ac} = \tau_{ab}$  is overestimated.

## VI. EXPERIMENTS AND EVALUATION

WCDOPS+\_NIM and WCDOPS+ have been implemented in the Cheddar scheduling analysis tool<sup>1</sup> for our experiments. In this section, WCDOPS+\_NIM is compared to WCDOPS+ by simulation. The complexity of our test is also discussed. Finally our test is applied to a real TDMA SRP.

### A. Comparison to WCDOPS+ by Simulation

1) *Simulation Parameters:* In order to compare WCDOPS+\_NIM with WCDOPS+, the tests are applied to randomly generated systems composed of 4 processors and 10 transactions with 10 tasks in each. The systems are generated according to the same parameters as the WCDOPS+ simulations in [14] so both tests can be compared. Initially tasks have the same priorities and allocated processors. Both parameters can vary with a probability of 0.25 to choose a random priority/processor. Tasks are immediate initially (offsets set by precedence dependency [12]). When its offset is set, a task has a probability of *nim\_prob* to become non-immediate (offset increased by a value between 1 and 1000). WCDOPS+\_NIM is applied on systems without any modification. WCDOPS+ is applied with *ghost intermediate tasks* to model non-immediateness.

2) *Simulation Results:* Fig. 7 shows results of two simulations where WCDOPS+\_NIM is compared to WCDOPS+. The CPU utilization varies between 10% and 70%. Like in [14], due to the nature of the experiment, the simulation becomes unfeasible for high CPU utilizations (higher than 70% in our case).

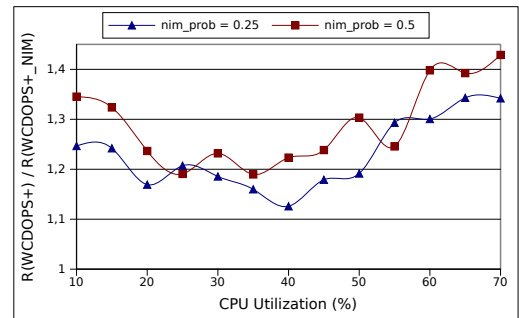


Fig. 7. Comparison between WCDOPS+ and WCDOPS+\_NIM by CPU Utilization and Offset Increase Probability (*nim\_prob*)

From the simulation, we see that results are mainly dependent on the CPU utilization and WCDOPS+\_NIM gives more significant tighter upper-bounds for lower and higher CPU utilizations. For the highest CPU utilization, WCDOPS+ gives an upper-bound 1.43 higher than WCDOPS+\_NIM.

<sup>1</sup><http://beru.univ-brest.fr/svn/CHEDDAR/>

## B. Complexity

Although WCDOPS+\_NIM gives tighter upper-bounds, its time complexity needs to be discussed. The general algorithm is pseudo-polynomial, a complexity inherited from [12]. In our feasibility test, two statements increase the complexity of the general algorithm: jitter canceling and checking immediateness. Each of these statements is a loop.

Jitter canceling adds a scenario to create, if  $\tau_{ik}$  is non-immediate and  $pred(\tau_{ik}) \notin hp_i$ . Let  $n_\tau$  be the number of tasks,  $n_\Gamma$  the number of transactions. The maximum number of extra scenarios created is  $(n_\tau - n_\Gamma)/2$  and the complexity is about  $O(n_\tau - n_\Gamma)$ , thus stays linear.

Checking immediateness is done by looping through tasks of a transaction. It depends on the number of tasks in the system and so it also stays linear.

In conclusion these statements do not add a significant increase in time complexity.

## C. Experimentation on TDMA SRP

To assess the gain of applying WCDOPS+\_NIM on a TDMA SRP, the test is compared to current practices at Thales, where the classic test in [3] is applied.

Our case-study is a real TDMA SRP, implemented with 9 POSIX threads, running on a platform with 2 processors (GPP1 and GPP2), scheduled by the SCHED\_FIFO scheduler of Linux (patched RT-Preempt) on both processors. For a TDMA Frame of 14 slots, the time parameters of pthreads instances are illustrated in Fig. 8.

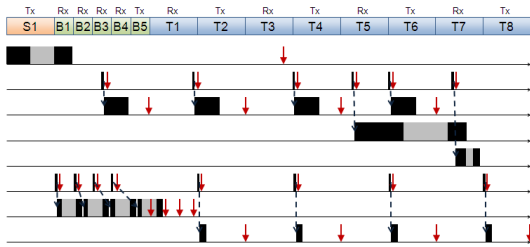


Fig. 8. TDMA Frame and Pthreads: Line = Instances of a pthread; Down arrow = Deadline; Dashed arrow = Precedence; Black = Exec on GPP1; Gray = Exec on GPP2; 9 pthreads (36 instances) in total; For readability, sizes are not proportional to time values.

The case-study is modeled with a transaction of 44 tasks, illustrated in Fig. 9<sup>2</sup>.

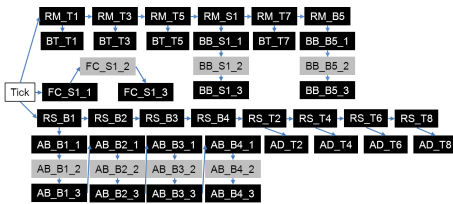


Fig. 9. Transaction of TDMA SRP: Processors differentiation by colors

<sup>2</sup>The model and analysis results can be found at: [http://beru.univ-brest.fr/svn/CHEDDAR/trunk/project\\_examples/wcdops+\\_nimp/tdma\\_mac/](http://beru.univ-brest.fr/svn/CHEDDAR/trunk/project_examples/wcdops+_nimp/tdma_mac/)

In average the test in [3] gives a WCRT 5 times higher than WCDOPS+\_NIM. Thus using transactions to model a TDMA SRP and applying a holistic schedulability test, like WCDOPS+\_NIM, increases schedulability of such systems.

## VII. CONCLUSION

In this article we proposed a schedulability test for tree-shaped transactions with non-immediate tasks. The test, called WCDOPS+\_NIM, is based on WCDOPS+ and is motivated by TDMA SRPs. Simulation results show that WCDOPS+ gives upto 1.43 higher WCRT upper-bounds than WCDOPS+\_NIM. Both tests have the same complexity. This work shows that the effects of non-immediateness must be taken into consideration when computing the WCRTs of tasks. Applying our test to real TDMA SRP also shows that our test gives upto 5 times tighter WCRT upper-bounds compared to WCRTs given by the classic test [3] used at Thales. In the future, WCDOPS+\_NIM will be integrated in a design process at Thales.

## REFERENCES

- [1] J. Garcia, J. Gutierrez, and M. Harbour. Schedulability analysis of distributed hard real-time systems with multiple-event synchronization. In *Proceedings of the 12th Euromicro Conference on Real-Time Systems*, pages 15–24. IEEE Comput. Soc, 2000.
- [2] R. Henia and R. Ernst. Improved offset-analysis using multiple timing-references. In *Proc. Conf. Design Automation and Test in Europe 2006*, pages 450–455, Munich, Germany, 2006.
- [3] M. Joseph and P. Pandya. Finding response times in a real-time system. *Comput. J.*, 29(5):390–395, 1986.
- [4] J. Kany and S. Madsen. Design optimisation of fault-tolerant event-triggered embedded systems. Master’s thesis, Tech. Univ. of Denmark, Lyngby, Denmark, 2007.
- [5] H. Kopetz. Event-triggered versus time-triggered real-time systems. In *Operating Systems of the 90s and Beyond*, volume 563 of *Lecture Notes in Computer Science*, pages 86–101. Springer Berlin Heidelberg, 1991.
- [6] J. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proc. 11th Real-Time Syst. Symp.*, pages 201–209, Lake Buena Vista, USA, 1990.
- [7] S. Li, F. Singhoff, S. Rubini, and M. Bourdellès. Applicability of real-time schedulability analysis on a software radio protocol. *ACM SIGAda Ada Lett.*, 32(3):81–94, Dec, 2012.
- [8] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, Jan, 1973.
- [9] N. Malcolm and W. Zhao. The timed-token protocol for real-time communications. *Comput.*, 27(1):35–41, Jan. 1994.
- [10] S. O. Marinescu, D. Tamas-Selicean, V. Acretoaie, and P. Pop. Timing analysis of mixed-criticality hard real-time applications implemented on distributed partitioned architectures. In *Proc. 17th Int. Conf. Emerging Technologies & Factory Automation*, pages 1–4, Krakow, Poland, 2012.
- [11] S. Mubeen, J. Maki-Turja, and M. Sjodin. Extending offset-based response-time analysis for mixed messages in controller area network. In *Proceedings of the 18th Conf. on Emerging Technologies & Factory Automation*, Cagliari, Italy, 2013.
- [12] J. Palencia and M. Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *Proc. 19th IEEE Real-Time Syst. Symp.*, pages 26–37, Madrid, Spain, 1998.
- [13] J. Palencia and M. Harbour. Exploiting precedence relations in the schedulability analysis of distributed real-time systems. In *Proc. 20th IEEE Real-Time Syst. Symp.*, pages 328–339, Phoenix, USA, 1999.
- [14] O. Redell. Analysis of tree-shaped transactions in distributed real time systems. In *Proc. 16th Euromicro Conf. Real-Time Syst.*, pages 239–248, Catania, Italy, 2004.
- [15] L. Sha, R. Rajkumar, and J. Lehoczky. Priority inheritance protocols: an approach to real-time synchronization. *IEEE Trans. Comput.*, 39(9):1175–1185, Sep, 1990.
- [16] K. Tindell and J. Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and Microprogramming*, 40(2-3):117–134, Apr, 1994.