# ACCEPTOR: a model and a protocol for real-time multi-mode applications on reconfigurable heterogeneous platforms

Joël Goossens
joel.goossens@ulb.ac.be
Université libre de Bruxelles
Brussels, Belgium

Xavier Poczekajlo
xavier.poczekajlo@ulb.ac.be
Université libre de Bruxelles
Brussels, Belgium

Antonio Paolillo
antonio.paolillo@ulb.ac.be
Université libre de Bruxelles
Brussels, Belgium

Paul Rodriguez
paul.rodriguez.lobera@pm.me
Université libre de Bruxelles
Brussels, Belgium

## ABSTRACT

In this work, we consider hard real-time applications scheduled upon heterogeneous multiprocessor platforms. The originality of this study is to consider multi-mode real-time applications (software aspects) *and reconfigurable*-heterogeneous hardware platforms (composed of CPUs, GPUs, FPGAs…). Our approach is based on a multi-mode protocol, for mode-dependent tasks upon reconfigurable hardware. The goal is to handle predictable switches between different task sets *and* different hardware settings. The novelty here is the *dynamic* hardware *and* software reconfigurability. First, we propose a formal model of the applications and reconfigurable hardware platforms. We then propose and prove correct a mode change protocol. We propose in particular a validity test for the verification of the timing constraints of the application —including the time allowed to complete a mode change. We also perform a complete evaluation. We study the theoretical complexity of the protocol, use a simulation to evaluate the efficiency of our solution, and finally propose a competitive analysis of our protocol to prove that it is 2-competitive.

## CCS CONCEPTS

• **Computer systems organization → Real-time systems**; *Multicore architectures.*

## KEYWORDS

real-time scheduling, mode change protocol, multiprocessor, reconfigurable hardware

**ACM Reference Format:**
Joël Goossens, Xavier Poczekajlo, Antonio Paolillo, and Paul Rodriguez. 2019. ACCEPTOR: a model and a protocol for real-time multi-mode applications on reconfigurable heterogeneous platforms. In *27th International Conference on Real-Time Networks and Systems (RTNS 2019), November 6–8, 2019, Toulouse, France.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3356401.3356420


*RTNS 2019, November 6–8, 2019, Toulouse, France*

ACM ISBN 978-1-4503-7223-7/19/11…$15.00
https://doi.org/10.1145/3356401.3356420

## 1 INTRODUCTION

The performance requirements for embedded real-time applications are rising. Machine vision systems and intelligent vehicles are only a few examples taken from the wide range of those demanding applications. Several techniques are used to design those real-time applications. One of them is to model the different functionalities as a set of different recurrent tasks. Some applications have the particularity to go through several *states* (or modes) during their lifespans. Those modes may be for an example event-driven (the detection of a specific event by the system) or time-driven (day/night modes). Each mode may then be designed independently, with its own set of functionalities modelled as several task sets: one for each mode. Only one mode and its task set can be active at a time. Executing a real-time multi-mode application requires to be able to schedule each mode, but also to handle the transition from one mode to another. During a transition, the tasks from the old mode must be completed, and the tasks from the new mode must be enabled. The transition from one mode to another must be bounded by a real-time constraint, defined at design time.

Dividing an application into several distinct modes improves the schedulability analysis. Indeed, the analysis is then closer to what will happen during the life-span of the application which makes it more accurate. By dividing the tasks into several task subsets — one for each mode, the maximal instantaneous workload is reduced, leading to a potential reduction of the hardware requirements. This technique has been well studied on uni-processor and multi-processor systems (see [20–22, 26]).

Modern hardware offers to the designer of real-time applications the possibility to be reconfigured during run-time. Such hardware can be reconfigured in several ways: it can be turned on and off (to adjust the energy consumption to the current workload), or its speed and/or functionalities. Modifying the hardware functionalities at run-time allows to reduce the hardware requirements. Indeed, several specialised processors working at distinct time instants may be replaced by a single reconfigurable processor (any hardware that may be used to execute tasks, and may change its behaviour at run-time). For example, last generation FPGAs (field-programmable gate array) offer all those possibilities. This kind of hardware is included in new embedded processing platforms, which

makes them heterogeneous in nature. For example, the Xilinx Zynq UltraScale+™ system on chip [29] is a single die integrating several ARM processing cores from different architectures together with other heterogeneous components (64 bits quad-core Cortex® A53 cores, dual-core Cortex® R5 real-time cores, a Mali™-400 MP2 GPU and an FPGA programmable logic). This processor's block diagram is illustrated by Figure 1.

In this paper, we propose a new approach: combining both software and hardware reconfigurations. In this approach, each mode is defined by a set of functionalities and a specific hardware configuration adapted to those functionalities. The hardware reconfigurations are handled during the transition from one mode to another. To the best of our knowledge, combining both multi-mode software and hardware reconfiguration has never been studied before. However, fitting the hardware to the current functionalities of the application leads to a more efficient execution (in terms of speeds, energy consumption and/or hardware requirements).

**Contributions.** The key contributions of this paper are the following:

- the — to our knowledge — first model for multi-mode applications on reconfigurable heterogeneous platforms;
- an asynchronous aperiodic mode-change protocol to schedule such applications on reconfigurable multi-processor systems;
- a validity test for this protocol based on an upper-bound of the transition duration;
- a complete evaluation through simulations, complexity and competitive analysis of the contributions.

The remaining of the paper is organised as follows. Section 2 presents the related work. Section 3 introduces the model used in the paper. Section 4 describes the mode change protocol proposed as a first solution to handle the transition between the different modes. Section 5 proposes an upper-bound of the transitions duration, when using the introduced protocol. Section 6 provides a validity test and proves its correctness. Section 7 evaluates the protocol through a complete evaluation using simulations for the upper-bound efficiency, and both a complexity and competitive analysis of the protocol. It also discusses some of the limitations of the model. Section 8 concludes the paper.

## 2 RELATED WORK

***Multi-mode.*** A survey [26] proposes various solutions and defines the main vocabulary and concepts for multi-mode applications on *uni*-processor systems. Concerning *multi*-processors, the literature reports several multi-mode protocols which handle the transition from one mode to another. Regarding *global* scheduling, V. Nelis' works include several protocols for *homogeneous* or *heterogeneous related* multi-processors (see for instance [20–22]). Those protocols combine different paradigms: periodic/aperiodic and synchronous/asynchronous task systems. More recently, Shih et al. [28] provided a schedulability analysis for global scheduling of mode change for the imprecise computation model upon identical multi-processors.

Concerning *partitioned* scheduling a short contribution by Marinho et al. [19] formalises the scheduling problem and shows two

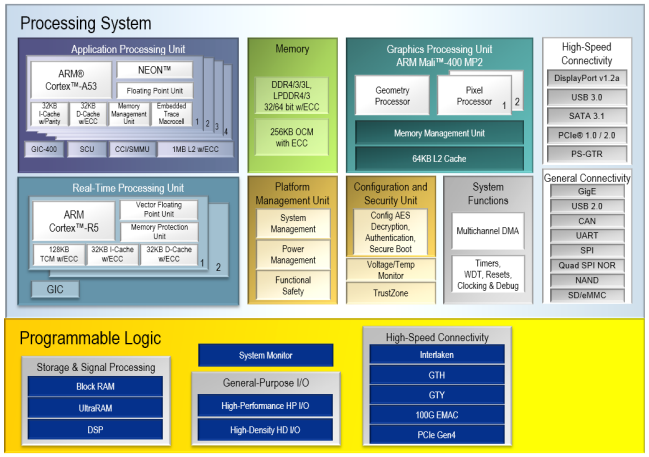


**Figure 1: The Zynq UltraScale+™ EG processor block diagram. EG devices feature a quad-core ARM® Cortex-A53 platform running up to 1.5GHz, combined with dual-core Cortex-R5 real-time processors, a Mali-400 MP2 graphics processing unit, and a 16nm FinFET+ programmable logic [29].**

counter-intuitive phenomena. Emberson et al. [9] proposed heuristics to handle the mode change. Lastly, Goossens et al. [13] consider the partitioned scheduling problem of multi-mode real-time systems upon *identical* multi-processors. The authors propose two methods for handling mode changes in partitioned scheduling.

In this work we extend our previous contributions where we introduced the study of multi-mode protocols on multi-processor platforms, with reconfigurable hardware [12].

***Scheduling upon heterogeneous platforms.*** Cluster-based scheduling —where tasks are assigned to a given set of processors called *cluster* and cannot migrate to a different one, has been well-studied for heterogeneous systems . Raravi et al. [25] propose, to the best of our knowledge, the most efficient approach to this problem.

***Reconfigurable hardware.*** Research concerning reconfigurable platforms combining CPUs and FPGA elements used for real-time systems is relatively new, as the platforms themselves are new to the market [15].

Cornil et al. [5] assessed the research challenges to face with this kind of platforms. Ahmad et al. [27] provided tools to optimise the design of real-time applications running on reconfigurable devices (with regards to different metrics such as performance and energy consumption). Pagani et al. [24] proposed the integration of Dynamic Partial Reconfiguration (DPR, a technique to reconfigure an FPGA at run-time) as part of a provided service of operating systems. Biondi et al. [3, 4] provided several timing analyses and run-time framework works that make use of DPR, enabling reconfigurable heterogeneous platforms as target candidates for real-time systems. They also provided an extensive state of the art as part of their research paper [3]. Their approach is based on modelling the dependency between heterogeneous components with self-suspending tasks, waiting for resources to free on remote processing units. Later, Pagani et al. [23] provided an implementation of their DPR

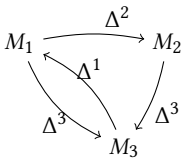


**Figure 2: Graph transition**

framework for the Linux operating system. Bini [2] presented the Adaptive Fair Scheduler technique, that considers resource allocation and provide guarantees to the application. His approach is general enough to be applied to heterogeneous and reconfigurable computing.

## 3 MODEL

In this section, we present the first contribution of our paper: the model. To the best of our knowledge, this model is the first to combine both multi-mode application model and reconfigurable platform model. A multi-mode application on a reconfigurable platform is composed of different *modes* $M^h =< \tilde{\Theta}^h, T^h, \Delta^h >$. Each *mode* will execute a different task set $T^h$ (see Section 3.2) on the platform configured in a specific way according to $\tilde{\Theta}^h$ (see Section 3.3) for this mode. For each mode, $\Delta^h$ is the real-time constraint of the mode change (see Section 3.1). The application is composed of $\mu$ modes $M \doteq \{M^1, M^2, \ldots, M^\mu\}$, executed on a platform containing $m$ processors.

Two use-cases of the model are shown in Section 3.6, as examples. All the notations introduced in this section are referenced in Table 1.

### 3.1 Mode transitions

The application executes at any instant a mode $M^h$. The active mode may only change during a *Mode Change Phase*. A *Mode Change Phase* is triggered when the system receives a *Mode Change Request*. It can only be received whenever the platform is not already performing a Mode Change Phase. When a Mode Change Request MCR($M^{\text{dst}}$) occurs at $t_{\text{MCR}}$, the current mode is immediately deactivated (and its task set is disabled), and new mode $M^{\text{dst}}$ must be activated (and its task set enabled) by $t_{\text{MCR}} + \Delta^{\text{dst}}$. $\Delta^{\text{dst}}$ is a real-time constraint specified at design time for each mode $M^h$. The Mode Change Phase ends when the new mode $M^{\text{dst}}$ is activated. Do note that this constraint depends only from the destination mode $M^{\text{dst}}$, independently from $M^{\text{src}}$.

*3.1.1 Mode Change Graph.* In an application, some transitions will never occur. The possible transitions are represented in the *Mode Change Graph* $\mathcal{G} \doteq \{\text{V}, E \subseteq \text{V}^2\}$. The *Mode Change Graph* is a directed graph, where V contains one and only one node for each mode $M^h \in M$, and $E$ represents all the possible transitions from one mode to another. A Mode Change Phase from a mode $M^{\text{src}}$ to a mode $M^{\text{dst}}$ is possible if and only if $(M^{\text{src}}, M^{\text{dst}}) \in E$. A graph example is given in Fig. 2. In this example, the mode after $M^2$ must be $M^3$ and not $M^1$ because the directed edge from $M^1$ to $M^2$ is unidirectional.

### 3.2 Task model

*3.2.1 Job.* The platform executes *jobs*. A *job* $J_i$ is defined by three parameters $(a_i, c_i, d_i)$. The arrival time in the system is denoted by $a_i$. The *worst case execution time* (WCET) $c_i$ is computed on an arbitrary platform running at a unitary rate (i.e. one computing unit per time-unit). Finally, $d_i$ is its *absolute deadline*. The platform must therefore execute the job $J_i$ for $c_i$ computing units between $a_i$ and $d_i$ to be completed. A job is said to be active if it has arrived in the system and is not completed yet. It is said to be inactive otherwise.

*3.2.2 Task.* In the application, *n tasks* are forming the task set. A *task* $\tau_i$ is defined by three parameters $(C_i, D_i, T_i)$. $C_i$ is the *WCET*, $D_i$ is the *relative deadline* and $T_i$ is the *minimum inter-arrival time*. In this work, we are considering implicit deadlines only: $\forall i,\ D_i = T_i$.

A task $\tau_i$ generates an infinite sequence of jobs. A task releases a job if and only if it has been enabled. When a task is disabled, its incomplete jobs are called rem-jobs. It is enabled (resp. disabled) when its mode becomes active (resp. inactive). A job $J_i$ released at $a_i$ has an absolute deadline of $d_i \doteq a_i + D_i$ and $c_i \doteq C_i$.

### 3.3 Platform model

From simple uniprocessor platforms, modern models are now also targeting complex platforms. Multiprocessors platforms can be either modelled as an *identical*, *uniform* or *unrelated* machine [10]. An *identical machine* is a platform on which all the processors are identical. A *uniform machine* is formed of similar processors with different performances. In this paper, we consider *unrelated machines*. This kind of platform is formed of processors having different instruction sets and/or different performance. More than unrelated processors, our model considers reconfigurable processors. Those processors may change their instruction sets or performances at run-time. For example, FPGA or GPU can be considered here as reconfigurable processors. We abstract here their specificity in this first model. To overcome this limitation, we may consider that we have some general purpose processors that are dedicated to launching jobs on the GPU.

The platform $P$ is composed of *m processors*. Each *processor* $p_q$ has a type $\pi^k$ with $k \in [1, \ldots, \phi]$, and $q \in [1, \ldots, m]$. There are $\phi$ different *processor types*. The vector $\Pi$ contains the type of each processor. The $q^{\text{th}}$ element $\Pi_q$ contains the type of the processor $p_q$. The type defines the possible configurations for the processor.

**Configurations.** A processor of type $\pi^k$ is configured at any time in a *configuration* $\theta_c$ from its set of configurations $\Theta_k$: $\theta_c \in \Theta_k$. A configuration defines several parameters like the instruction set of the processor or the speed. The set of sets $\Theta \doteq \{\Theta_1, \Theta_2, \ldots, \Theta_\phi\}$ contains the possible set of configurations for all types. A configuration is accessible to processors of one and only one type: i.e. $\forall k, k', k \neq k' \implies \Theta_k \cap \Theta_{k'} = \emptyset$. There are $o$ different configurations, with $o \doteq \sum_{k=1}^{\phi} |\Theta_k|$. Reconfiguring a processor is not instantaneous. It takes $\delta_c$ time-units (denoted as the *reconfiguration delay*) to reconfigure a processor of type $\pi^k$ to $\theta_c$ if $\theta_c \in \Theta_k$, otherwise it takes $+\infty$. We consider here parallel reconfigurations. That is not possible in the general case with existing hardwares. However, the platforms tend to allow more and more reconfigurations in parallel. The use of 3D-design may allow in a near future a parallel DPR for the whole platform.

**(Non-)Reconfigurable.** A processor $p_q$ of type $\pi^k$ may be *reconfigurable* or *non-reconfigurable*. It is said to be *reconfigurable* if and only if $|\Theta_k| > 1$.
**(In-)Active.** A processor may be turned on and off dynamically. This may be used to manage power consumption. A processor $p_q$ is inactive at time $t_1$ if and only if it is configured in an idle-configuration. An idle-configuration is a configuration in which a processor cannot execute any job, i.e. its speed is null for any job. It is said active otherwise.

## 3.4 Job progression rate

On heterogeneous machines, the *job progression rate* (speed at which a job is being completed) depends on both the job and the processor. Specifically, the job progression rate $r_{i,c}$ on the processor $p_q$ depends on both the task $\tau_i$ and the current configuration $\theta_c$ of $p_q$. Formally, Job $J_i$ completes $t \times r_{i,c}$ computing units when executed on a processor configured in $\theta_c$ for $t$ time-units. This amount may be null if the task cannot be executed on this configuration.

## 3.5 Mode model

A *mode* $M^h =< \tilde{\Theta}^h, T^h, \Delta^h >$ is a combination of a specific hardware configuration, a specific software and a real-time constraint. In our model, the software is represented by the task set to execute and the hardware by the configurations of the processors. The real-time constraint bounds the maximum delay to reconfigure the whole platform to the new mode. In the literature, previous works — as Nélis' work in [22]— were only considering software reconfiguration. Reconfiguring both *hardware and software* is the originality of our work.

Each mode has a cluster-based approach. It will consider several independent groups of processors — denoted as clusters, with a global approach inside each cluster. To do so, each mode splits its task set into several task subsets, and schedule each subset upon a distinct cluster.

*3.5.1 Mode specifications.* The mode $M^h$ is defined by the following specifications:
**Hardware.** The processors must be configured in a specific way. *The configurations vector* $\tilde{\Theta}^h \doteq < m_1^h, m_2^h, \ldots, m_o^h >$ contains for a specific mode $M^h$ the required number $m_c^h$ of processors configured in $\theta_c$.

The required configurations for all the modes are specified as a vector of configuration vectors: $\tilde{\Theta} \doteq < \tilde{\Theta}^1, \tilde{\Theta}^2, \ldots, \tilde{\Theta}^\mu >$.
**Software.** The mode $M^h$ has to execute a specific task set $T^h$, containing $\nu^h$ tasks. To partition the task set among several clusters, $T^h$ itself is divided into task subsets: one per configuration present in the mode $M^h$. The processors configured in $\theta_c$ must run the task subset $T^{h,c}$. The set $\Lambda \doteq \{T^1, T^2, \ldots, T^\mu\}$ contains the task subsets for all the modes. The tasks are said to be *mode-dependent*: each task may appear in at most one mode, i.e. $\forall h, h', h \neq h' \implies (\cup T^{h,c}) \cap (\cup T^{h',c}) = \emptyset$. This assumption will be discussed in Section 7.4.
**Delay.** Switching from one mode to another is not instantaneous. Rem-jobs cannot be stopped before completion and reconfiguring a processor takes a delay. However, this delay must be bounded to take into account the real-time constraints of the application. $\Delta^h$ represents the maximum possible delay for reconfiguring the system after a Mode Change Request to $M^h$. The set $\Delta \doteq \{\Delta^1, \Delta^2, \ldots, \Delta^\mu\}$ contains the real-time constraint of each mode.

**Table 1: Summary of the notations**

| | |
|---|---|
| A mode | $M^h \in M$ |
| Mode change instant | $t_{\text{MCR}}$ |
| The mode change graph | $\mathcal{G}$ |
| A job | $J_i$ |
| A task | $\tau_i$ |
| A processor | $p_q \in P$ |
| A type | $\pi^k$ |
| Type of processor $p_q$ | $\Pi q$ |
| A configuration of type $\pi^k$ | $\theta_c \in \Theta_k$ |
| Reconfiguration delay to $\theta_c$ | $\delta_c$ |
| Progression rate for $J_i$ on $\theta_c$ | $r_{i,c}$ |
| Configurations requirements of Mode h | $\tilde{\Theta}^h \in \tilde{\Theta}$ |
| Task subset of a mode $M^h$ for proc. in $\theta_c$ | $T^{h,c}$ |
| Task set of a mode $M^h$ | $T^h \in \Lambda$ |
| Mode $M^h$'s real time constraint | $\Delta^h \in \Delta$ |

## 3.6 Model example

In this section, we provide instances of our model to match existing real-world platforms.

**Example 1.** As we explained, the platform is organised in several unrelated clusters of identical machines. When using DPR (Dynamic Partial Reconfiguration), the Zynq UltraScale+™, may be configured such that it is composed of the following identical platforms:

- a four cores identical platform meant to execute general purpose application software (the 4 Cortex-A53 cores);
- a dual core identical platform that may host highly critical and predictable software (the dual Cortex-R5 cores);
- a GPU dedicated to display information (the Mali™-400 GPU). We model it as a single processor;
- 4 independent hardware accelerators that are each dedicated to one specific task. These accelerators are hosted in the Programmable Logic (FPGA) which may arbitrarily be divided in 4 independent slots by the system designer (and managed by the DPR engine). Each one of these accelerators/slots corresponds to a single processor and the whole Programmable Logic is then modelled as an identical platform with 4 processors. The ability of the others processors to change some features—such as their voltage/frequency— will be ignored, to keep this example short.

In this example, we assumed that the system designer used Dynamic Partial Reconfiguration (DPR) capabilities offered by the Zynq chip family [30] (notice that earlier research allow the use of DPR in real-time systems as described in Section 2). We assume that the designer divided the FPGA into 4 processors that are reconfigurable as defined in Section 3.3.

In this example, the platform $P$ is composed of the following $m = 11$ processors:

- $p_1, p_2, p_3, p_4$: the application cores, forming the Application Processing Unit (APU). All these cores have the same architecture (A53) and belong to type $\pi^1$: $\Pi_1 = \Pi_2 = \Pi_3 = \Pi_4 = \pi^1$. As the APU is a static CPU, it has only one configuration, meaning $\Theta_1 = \{\theta_1\}$, where $\theta_1$ denotes the only possible configuration of the APU.
- $p_5, p_6$: the real-time cores, forming the Real-Time Processing Unit (RPU). These cores have the same architecture (R5) and then belong to type $\pi^2$. Similarly to the APU, we have $\Theta_2 = \{\theta_2\}$, where $\theta_2$ denotes the only possible configuration of the RPU.
- $p_7$: it represents the GPU, which has its own type $\pi^3$ and is also a static processor: $\Theta_3 = \{\theta_3\}$ where $\theta_3$ denotes the only possible configuration of the GPU.
- $p_8, p_9, p_{10}, p_{11}$: the different dynamically partially reconfigurable processors of the chip Programmable Logic (from the FPGA). These 4 processors are independent. We assume that the processors are specialised for some processing: we have two types of processors $\pi^4$ and $\pi^5$. We assume that in this example, $p_8$ and $p_9$ are image processing kernels (allowing to process an input image and apply a filter on it), and that $p_{10}, p_{11}$ implement cryptographic accelerators in order to implement several block cipher algorithms. The image processing kernels can be in three configurations: sepia, sobel and grayscale. The cryptographic accelerators can be configured either to run AES or 3DES block ciphers. Therefore:

$$\begin{cases} \Theta_4 = \{\theta_{\text{sepia}}, \theta_{\text{sobel}}, \theta_{\text{gray}}\} \\ \Theta_5 = \{\theta_{\text{aes}}, \theta_{\text{des}}\} \end{cases}$$

The rates of task $\tau_{\text{aes}}$ depend on the configuration of its processor. This task can be performed only on RPU, GPU or a specialised FPGA. Therefore, we could have : $r_{\text{aes,aes}} > r_{\text{aes},3} > r_{\text{aes},2} > r_{\text{aes},1} = r_{\text{aes,des}} = 0$. Obviously, this task is completed (way) faster on a specialised FPGA, and cannot be executed on an FPGA wrongly configured (hence $r_{\text{aes,des}} = 0$). This platform example is depicted in Fig. 3(a). The platform is represented with several divisions into *clusters*. Typical reconfiguration times of the Programmable Logic are of the order of a few milliseconds [24]. These timings could be used to set the values of $\delta_{\text{sepia}}, \delta_{\text{sobel}}, \delta_{\text{gray}}, \delta_{\text{aes}}$ and $\delta_{\text{des}}$.

**Example 2.** The ARM big.LITTLE architecture is a technique allowing to switch between high-performance cores and low-power cores. A typical design of this architecture is composed of 4 A57 cores (the high-performance cluster) and 4 A53 cores (the low-power cluster) used alternately. These kind of platforms suits our model as the multi-mode reconfiguration is implemented in the hardware. In this case, we model the platform $P$ as a set of $m = 8$ processors:

- $p_1, p_2, p_3, p_4$: the high-performance cluster. All cores are of type $\pi^1$ and $\Theta_1 = \{\theta_{A57}, \theta_{\text{off }57}\}$, these configurations meaning that the cores are either active or inactive.
- $p_5, p_6, p_7, p_8$: the low-power cluster. They are of type $\pi^2$ and, similarly to the other type, we have $\Theta_2 = \{\theta_{A53}, \theta_{\text{off }53}\}$

In some platform designs such as Samsung Exynos 5 Octa, only one kind of core can run simultaneously — the operating system must explicitly switch the whole platform from one kind of core to the other. This constraint can be enforced by defining the clusters and modes accordingly and by using the inactive configurations $\theta_{\text{off }53}$ and $\theta_{\text{off }57}$. For example, if we define two modes $M^{53}$ and $M^{57}$, $\tilde{\Theta}^{53} =< m^{53}_{A53}, m^{53}_{A57} >=< 4, 0 >$ and $\tilde{\Theta}^{57} =< m^{57}_{A53}, m^{57}_{A57} >=< 0, 4 >$. This means that during mode $M^{53}$, only the processors of type $\pi^2$ will be active.

These two examples illustrate how generic our model is and how it is able to capture the essence of real-world platforms.

## 4 PROTOCOL

In this section, we introduce the second contribution of this paper: the protocol to handle mode transitions on a reconfigurable platform. It is called ACCEPTOR, for : AsynChronous ClustEr-based ProTOcol for multi-mode applications on Reconfigurable platforms. ACCEPTOR conducts the reconfiguration phase: it schedules the rem-jobs, performs the requested reconfigurations and enables the new mode tasks. It also respects the real-time constraints such as the delay for new mode's activation and the rem-jobs hard deadlines.

The description of the protocol will be preceded by an introduction about clusters.

### 4.1 Clustered platform

A clustered platform is composed of several groups of processors called *clusters*. Formally, a cluster is formed by several processors, configured *identically*. At a given instant $t$, a processor of type $\pi^k$ configured in $\theta_c$ belongs to one and only one cluster. This cluster is composed of all processors of type $\pi^k$ configured in $\theta_c$. Jobs migrations are only allowed inside their cluster.

In this paper we will use the following popular concepts regarding multi-mode problem:

**Work-conserving** A scheduler is *work-conserving* iff the processors can be idle only when there are no job waiting to execute.

**The cluster's makespan** is the required time to complete all the rem-jobs generated by the task set of the cluster. [22] proposed an upper-bound on the worst case makespan $\overline{\text{MS}}$.

**Idle** A cluster is said to be *idle* iff all its processor are idle. A processor is idle if it is neither executing tasks nor being reconfigured.

**Idle$_k$ instant (from [20])** An *Idle$_k$* instant is the earliest instant such as at least $k$ processors are idle. The upper-bound on the Idle$_k$ instant denoted as $\overline{\text{Idle}}_k$.

**A $\pi^k$ - cluster** is composed of processors of type $\pi^k$, configured identically.

**A $\theta_c$ - cluster** is a cluster composed of processors configured in $\theta_c$.

Because the processors may be reconfigured, the clusters may (only) be modified during a mode change phase. Clustering is based on the configuration of the system, and may thus change from one mode to another. Fig. 3 represents the same platform with two different clustering. For example, $p_8$ is on a cluster with $p_9$ in the first mode (see (a)) but is then alone in the second mode (see (b)).

### 4.2 ACCEPTOR

The ACCEPTOR protocol is an aperiodic asynchronous protocol —a protocol is periodic iff it has mode-independent tasks, a protocol is synchronous iif dependent tasks of two different modes can never

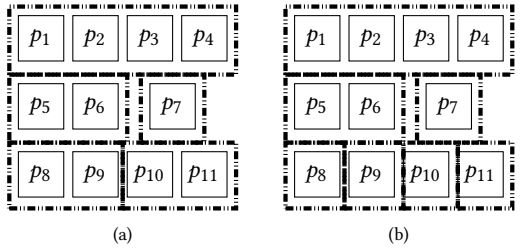


**Figure 3: Two possible divisions of the example platform (based on the Zynq UltraScale+™) into several clusters.**

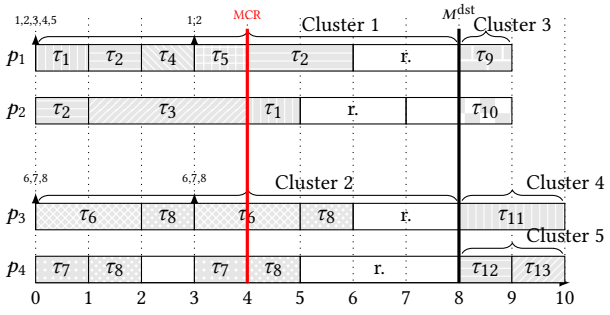


**Figure 4: Protocol illustration**

be active simultaneously. It can be used with sporadic task sets, scheduled by a clustered work-conserving scheduler.

The approach is to schedule the rem-jobs, and reconfigure each processor once it becomes idle. This protocol is composed of an offline phase and a run-time phase. The offline phase determines for each cluster the required reconfigurations for each possible Mode Change Transition (see Section 4.2.1). The run-time phase is dedicated to schedule the rem-jobs and reconfigure the processors (see Section 4.2.2). This run-time phase focuses on each cluster individually. An illustration is given in Fig. 4. Finally, an example of the protocol is given in Section 4.2.3.

*4.2.1 Offline: computing the required reconfigurations.* The offline phase simply computes which reconfigurations must be done for each mode change phase from $M^{\text{src}}$ to $M^{\text{dst}}$. This will be done iteratively for each type $\pi^k$. To do so, the protocol first computes the differences between the required reconfigurations of both modes.

Then, it computes the makespan of each $\pi^k$-cluster, using the bound provided by [22]. It then assigns the longest (resp. shortest) required reconfigurations (based on the reconfiguration delays) -to the $\pi^k$ - clusters having the shortest (resp. longest) makespans. Inside each cluster, the shortest (resp. longest) reconfigurations are assigned to the processors being idle the first (resp. last), based on the idle-instant upper-bound from [20].

This is mandatory since the reconfiguration delay is based on the new configuration and not on the current configuration. The assignment is denoted $\{\theta_c, \theta_{c'}\}$ where $\theta_c$ is the configuration of the processor in $M^{\text{src}}$ and $\theta_{c'}$ is the configuration of the processor in $M^{\text{dst}}$.

Formally,

- Compute for each $\theta_c$ - clusters in each mode $M^h$ the makespan upper-bound $\overline{\text{MS}}$ for the task subset $T^{h,c}$ on $m_c^h$ processors;

Then, $\forall (M^{\text{src}}, M^{\text{dst}}) \in \mathcal{G}, k \in [1, \ldots, \phi]$:

(1) The missing configurations for $\pi^k$ - clusters are all configurations $\theta_c \in \Theta_k$ where $m_c^+ > 0$, with $m_c^+ \doteq \max(0, m_c^{\text{dst}} - m_c^{\text{src}})$. Symmetrically, the excess configurations are all $\theta_c \in \Theta_k$ where $m_c^- > 0$, with $m_c^- \doteq \max(0, m_c^{\text{src}} - m_c^{\text{dst}})$.
(2) To perform the reconfiguration, we use a vector containing missing configurations. This vector will be used at run-time to track the required configurations to perform. Store $\tilde{\theta}^+ \doteq < \theta_c, \theta_{c'}, \ldots >$ a vector of missing configurations. $\tilde{\theta}^+$ contains $m_c^+$ times the configuration $\theta_c$. Order $\tilde{\theta}^+$ by reconfiguration delay, in decreasing order.
(3) Symmetrically, store $\tilde{\theta}^- \doteq < \theta_c, \theta_{c'}, \ldots >$ a vector of excess configurations. $\tilde{\theta}^-$ contains $m_c^-$ times the configuration $\theta_c$. Order $\tilde{\theta}^-$ by the cluster's makespan of the processors configured in $\theta_{c''} \in \tilde{\theta}^-$, in increasing order.
(4) Bind the $x^{\text{th}}$ longest reconfiguration to the $x^{\text{th}}$ idle processor. The $x^{\text{th}}$ longest reconfiguration $\theta_c$ is determined using its reconfiguration delay $\delta_c$. The $x^{\text{th}}$ idle processor is the $x^{\text{th}}$ processor to become idle on this cluster during this mode change phase. To do so, define the assignment $\{\theta_c, \theta_{c'}\}$, where $\theta_c$ is the $x^{\text{th}}$ element of $\tilde{\theta}^-$ and $\theta_{c'}$ is the $x^{\text{th}}$ of $\tilde{\theta}^+$.
(5) Store the assignment $\{\theta_c, \theta_{c'}\}$ in the multi-set $\text{RT}^{\text{src,dst}}$.

The results will be used at run-time. Because a cluster is formed by processors of the same type and configured identically the required reconfigurations for the $\theta_c$ - cluster are contained in the multiset $\{\{\theta_c, \theta_{c'}\} \mid \{\theta_c, \theta_{c'}\} \in \text{RT}^{\text{src,dst}}\}$.

*4.2.2 Run-time: scheduling and reconfiguring.* At run-time, the protocol must successfully schedule the rem-jobs, and reconfigure the processors as computed in the offline phase. This phase is straightforward: it will schedule the tasks using the same scheduler as before. The only difference concerns the reconfigurations: when a processor becomes idle, it may be reconfigured. For each $\theta_c$ - cluster it takes as input the list of remaining rem-jobs $J$ and the required reconfigurations to make $\{\{\theta_{c'}, \theta_{c''}\} \mid c = c' \wedge \{\theta_{c'}, \theta_{c''}\} \in \text{RT}^{\text{src,dst}}\}$.

Formally,

- Schedule $J$ using the same scheduler as before;
- When a processor becomes idle, reconfigure it to the longest required reconfiguration $\theta_{\text{dst}}$ that hasn't been done yet, where $\theta_{\text{dst}} \in \{\{\theta_{c'}, \theta_{\text{dst}}\} \mid c = c' \wedge \{\theta_{c'}, \theta_{\text{dst}}\} \in \text{RT}^{\text{src,dst}}\}$.

Whenever $m_c^{\text{dst}}$ processors are reconfigured to $\theta_c$, the $\theta_c$ - cluster of the new mode $M^{\text{dst}}$ is formed and its task set is activated. This phase ends once every cluster is idle and reconfigured. The new mode $M^{\text{dst}}$ is then enabled.

*4.2.3 Example.* Figure 4 depicts an example of the protocol's execution. The platform is composed of 4 processors ($m = 4$). Processors $p_1$ and $p_2$ share the same type (i.e. $\Pi_1 = \Pi_2 = \pi^1$), as $p_3$ and $p_4$ (i.e. $\Pi_3 = \Pi_4 = \pi^2$), Two processors must be configured in $\theta_1$ for mode $M^{\text{src}}$: $\tilde{\theta}^{\text{src}} = \{2, 2, 0, 0, 0\}$. For example, $m_1^{\text{src}} = 2$. For mode $M^{\text{dst}}$: $\tilde{\theta}^{\text{dst}} = \{0, 0, 2, 1, 1\}$. This means that there will be three clusters in

this mode, because there are three different types of required configurations. Here are some of the different reconfigurations delay: $\delta_3 = 2$, $\delta_4 = 3$, $\delta_5 = 2$. The real-time constraint is $\Delta^{\text{dst}} = 10$.

For $\pi^2$ processors, $\tilde{\theta}^- =< \theta_2, \theta_2 >$ and $\tilde{\theta}^+ =< \theta_5, \theta_4 >$. Computing also $\tilde{\theta}^-$ and $\tilde{\theta}^+$ for $\pi^1$, we obtain all the required reconfigurations. Using the clusters' makespans, we can assign those reconfigurations. The multi-set $RT^{\text{src,dst}} = \{(\theta_1, \theta_3), (\theta_1, \theta_3), (\theta_2, \theta_4), (\theta_2, \theta_5)\}$.

At $t = 0$, $p_1$ and $p_2$ form the $\theta_1$ - cluster denoted as cluster 1. $p_3$ and $p_4$ form the $\theta_2$ - cluster denoted as cluster 2. The tasks $\tau_1$ and $\tau_2$ release a job at $t = 3$, which become rem-jobs. The same goes for $\tau_6, \tau_7$ and $\tau_8$

Processor $p_4$ becomes idle at $t = 5$. There are no other job to execute which leads the protocol to reconfigure it. The $\theta_2$ - cluster has the following reconfigurations to make: $(\theta_2, \theta_4), (\theta_2, \theta_5)$. The reconfigurations to $\theta_5$ are made first because $\delta_5 > \delta_4$. These reconfigurations end at $t = 8$. Because $m_5^{\text{dst}} = 1$, a cluster of the new mode is formed and its task set $\{\tau_{12}, \tau_{13}\}$ is activated. On the other hand, even if $p_2$ has been reconfigured at $t = 8$, because $m_3^{\text{dst}} = 2$: the cluster is not formed yet. At $t = 9$, all the new mode's clusters have been formed. The mode $M^{\text{dst}}$ can then be enabled. The phase last for $9 - 4 < \Delta^{\text{dst}}$ units of time, and all the rem-jobs were successfully scheduled so the mode change phase succeeds.

## 5 MAKESPAN UPPER-BOUND

To assert the feasibility of a given system using the ACCEPTOR protocol, we now introduce an upper-bound of the instant in which the cluster's rem-jobs are completed, and the required reconfigurations are done. This instant is denoted as $\overline{\text{reconfigured}}$, and can be immediately derived to obtain an upper-bound on any mode change phase. Then, the validity of the protocol will be demonstrated in Section 6.

The new upper-bound denoted as $\overline{\text{reconfigured}}$ will be used to verify the feasibility of an application running on a given platform.

LEMMA 5.1 (LEMMA 2.11 IN [20]). *Suppose that J is ordered by non-decreasing job processing times, i.e., $c_1 \leq c_2 \leq \cdots \leq c_n$, all starting at $t = 0$. Then, whatever the job priority assignment, an upper-bound on the $\text{Idle}_k$ instant, $1 \leq k \leq m$, when scheduled by a FJP work-conserving global scheduler upon m identical processors, with no reconfiguration is given by:*

$$\overline{\text{Idle}}_k = \begin{cases} c_k & \text{if } n = m \\ \frac{\sum_{i=1}^{n} c_i + (k-1) \times c_{n-m+k}}{m} & \text{otherwise} \end{cases} \quad (1)$$

The new upper-bound $\overline{\text{reconfigured}}$ is based on $\overline{\text{Idle}}_k$. It simply searches for the processor with the maximum sum of idle time + reconfiguration delay.

COROLLARY 5.2 (UPPER BOUND ON THE CLUSTER'S MAKESPAN). *Suppose that J is ordered by non-decreasing job processing times, i.e., $c_1 \leq c_2 \leq \cdots \leq c_n$, all starting at $t = 0$.*

*Suppose that the required reconfigurations' delays are in the vector $\Delta$, ordered by reconfiguration time in decreasing order, and that $\delta_q$ is the $q^{\text{th}}$ element of the vector.*

*Then, whatever the job priority assignment, an upper-bound on the cluster's makespan $\overline{\text{reconfigured}}$, when scheduled by a FJP work-conserving clustered scheduler is*

$$\overline{\text{reconfigured}} = \max_{q=1}^{m} \{\overline{\text{Idle}}_q + \delta_q\} \quad (2)$$

PROOF. The proof is trivially obtained by construction. □

## 6 VALIDITY

A validity test is used to determine whether the mode change phases of an application will be successfully managed by a multi-mode protocol, on a specific system. To ensure this, we need to verify that every deadline will be met during the transition phase, and that the new mode will always be enabled on time.

We now introduce the sustainability. This notion will be useful to prove that scheduling rem-jobs is at most as difficult as scheduling the recurrent tasks.

**Sustainability.** A scheduler $S$ is sustainable iff for any set of jobs $J$ schedulable by $S$, any jobset J′ derived from $J$ where all jobs are identical or smaller (equal or smaller C) will be feasibly and non-strictly faster scheduled by $S$.

Lemma 6.1 (Corollary 8 from [6]) will be used to show that any scheduler used in the protocol will be sustainable, by hypothesis. This lemma applies to any uniform multiprocessor platform, and thus to identical multiprocessor platform. Therefore, we consider in the following that the jobset J is composed of one job of each task, as a worst-case.

LEMMA 6.1 ([6]). *Any work-conserving and Fixed Job Priority (FJP) algorithm is sustainable on uniform multiprocessor platforms.*

The schedulability of the rem-jobs during mode change phase is ensured by the Lemma 6.1.

LEMMA 6.2 (REM-JOB'S SCHEDULABILITY). *Every rem-job's deadline of every cluster will be respected during every possible mode change phase, iff (i) the scheduler is a clustered, FJP, preemptive, work-conserving scheduler and (ii) $T^{\text{src},c}$ is schedulable by the scheduler upon $m_c^{\text{src}}$ processors configured in $\theta_c$.*

PROOF. Because of (i) and Lemma 6.1, we know that the scheduler is sustainable. Hence, the scheduler is sustainable and (ii): the rem-jobs of the task set $T^{\text{src},c}$ are schedulable on their cluster. Thus, the schedulability is ensured. □

Lemma 6.3 ensures that the real-time constraint of each mode will be respected during every possible transition.

LEMMA 6.3 (TRANSITION'S TIME CONSTRAINT). *The protocol will respect the time constraint $\Delta_{\text{dst}}$ iff $\forall$ src, dst, $(M^{\text{src}}, M^{\text{dst}}) \in \mathcal{G}$, $\forall\, \theta_c$ - cluster, with $\Delta_c^{\text{src,dst}}$ being the required reconfigurations's delays ordered in decreasing order:*

$$\Delta_{\text{dst}} \geq \max_{\theta_c \text{ - cluster}} \overline{\text{reconfigured}}(T^{\text{src},c}, m_c^{\text{src}}, \Delta_c^{\text{src,dst}})$$

PROOF. The right part of the inequation computes the maximal upper-bound $\overline{\text{reconfigured}}$. By construction, if the hypothesis is respected, every cluster of each possible transition will be reconfigured when required. □

Theorem 6.4 proves the validity of our protocol, under its hypothesis.

Theorem 6.4 (Validity of acceptor). *The protocol* acceptor *is valid iff (i) the scheduler is a clustered, FJP, preemptive, work-conserving scheduler , (ii)* $T^{\text{src},c}$ *is schedulable by the scheduler upon* $m_c^{\text{src}}$ *processors configured in* $\theta_c$ *and (iii)* $\forall$ src, dst, $(M^{\text{src}}, M^{\text{dst}}) \in \mathcal{G}$, $\forall\, \theta_c$ - cluster, *with* $\Delta_c^{\text{src,dst}}$ *being the required reconfigurations's delays ordered in decreasing order:*

$$\Delta_{\text{dst}} \geq \max_{\theta_c \text{ - cluster}} \overline{\text{reconfigured}}(T^{\text{src},c}, m_c^{\text{src}}, \Delta_c^{\text{src,dst}})$$

Proof. To prove the validity of the protocol, we must prove that every rem-job's deadline will be respected during the mode change phase and that the real-time constraint $\Delta_{\text{dst}}$ will be respected during each possible transition from every mode $M^{\text{src}}$ to mode $M^{\text{dst}}$.

- Because of the hypotheses (*i–ii*), Lemma 6.2 can be applied an thus prove that every job's deadline will be respected during the mode change phase.
- Because of the hypotheses (*i–iii*), Lemma 6.3 can be applied. Every real-time constraint will be therefore respected.

Because the job's deadlines and the real-time constraint will be respected, the protocol is valid when the hypotheses (*i–iii*) are respected. □

## 7 EVALUATION

In this section, we present a complete evaluation of the contribution. We first evaluate in Section 7.1 the theoretical worst case time complexity. Then, we use simulations in Section 7.2 to evaluate the pessimism of the bound introduced in Section 5. Section 7.3 performs a competitive analysis of the protocol itself, and finally Section 7.4 discuss a limitation of the model.

### 7.1 Complexity

We now evaluate the theoretical worst case time complexity of both the offline and run-time phases of the protocol.

Concerning the offline phase: the computation of $\overline{\text{Idle}}_k$ has a worst case time complexity of $O(n)$, as $\overline{\text{MS}}$. First, the protocol computes $C$ makespan upper-bounds, where $C \leq m$ is the number of clusters and $n$ the largest number of tasks that a cluster has to schedule, hence a complexity of $O(C \times n) = O(m \times n)$. Steps 2 and 3 performed on clusters contain a sort on at most $m$ elements, and Step 3 also computes $\overline{\text{Idle}}_k$ for at most $m$ processors; Thus, complexity of Step 2 is $O(m \times \log(m))$ and complexity Step 3 is $O(m \times \log(m) + m) = O(m \times \log(m))$. The total complexity is thus $O(m \times \log(m) + m \times n)$.

Concerning the run-time phase, it has the same complexity as the scheduler. Choosing the reconfiguration to perform can be made in $O(1)$.

Complexities for both offline and run-time phases are very low, and make the protocol scalable.

### 7.2 Empirical pessimism of $\overline{\text{reconfigured}}$

To the best of our knowledge, the upper-bound $\overline{\text{Idle}}_k$ pessimism has not been evaluated, neither theoretically nor empirically. We provided here an empirical evaluation of $\overline{\text{reconfigured}}$, which is based on $\overline{\text{Idle}}_k$. We measure the pessimism ratio $\frac{\text{upper-bound}}{\text{mode change time}}$, where the *mode change time* is a measurement of the simulated mode change phase, and where the *upper-bound* corresponds to the computation of $\overline{\text{reconfigured}}$ for a given cluster.

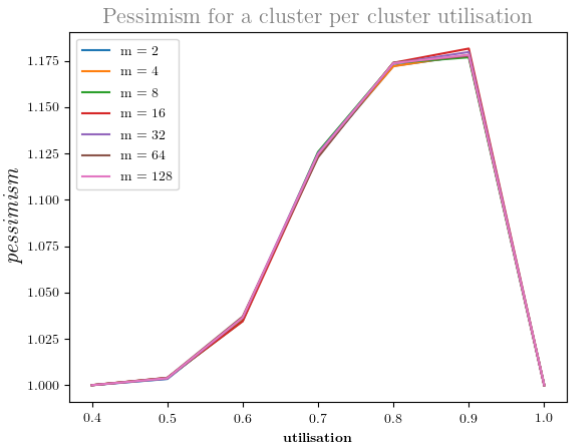


**Figure 5: Pessimism of $\overline{\text{reconfigured}}$**

The experiment has been conducted for clusters with $m = 2, 4, 8, 16, 32, 64, 128$ processors. The reconfiguration times are uniformly chosen in the range $[0, 10]$, where 0 indicates that no reconfiguration is required. We generate approximately 1 000 feasible task sets with utilisations $\in (p - 0.1, p]$, where we increase $p$ from 0.6 to 1.0 in steps of 0.1. The deadlines are uniformly chosen in the range $[1, 20]$. As a scheduler, we use Global-RM and remove any non-schedulable task set from our experiments. For those experiments, time is discrete.

The results are displayed in Fig. 5 and can be read this way: the average pessimism ratio is 1.125 for a cluster with $m = 4$ and a task set utilisation of 0.7. This means that the upper-bound is (on average) 12.5% larger than the actual duration in this configuration. One can observe two major trends: first, the pessimism does not depend on the number of processor. Secondly, a higher utilisation leads to a higher (but bounded) pessimism until $u \in (0.9, 1.0]$ for most cases. The pessimism is empirically bounded by 1.2. At $u = 1.0$, the pessimism ratio drops to 1.0. This phenomena is only due to our task set generator which produces task sets with a different shape for such a high utilisation.

This experimental results shows that the upper-bound $\overline{\text{reconfigured}}$ is accurate and can be used for clusters having a large amount of processors or a high utilisation.

### 7.3 Competitive analysis of acceptor

After the empirical evaluation of the upper-bound $\overline{\text{reconfigured}}$'s pessimism, we now evaluate the competitive analysis of the protocol itself. A protocol is said to be $\lambda$-competitive if it takes at most $\lambda \times O$ time to complete a mode change phase, where $O$ is the optimal mode change phase duration. To perform the competitive analysis of the protocol, we search for a worst case scenario, i.e. a system for which the relative difference between our protocol's performances and an optimal one is the highest. We first define what a *squeezable* system is, and then prove several properties on those systems, which leads

to prove that they correspond to the worst case. Once we have this worst case, we compute the competitiveness of the protocol. An example of a *squeezable* system is depicted in Fig. 6. The top graph shows a specific mode change phase handled by ACCEPTOR, and below is the same mode change phase handled by an optimal one.

Lemma 7.2 and Lemmas 7.4–7.6 introduce several properties that stand for any system. Lemma 7.1, Corollary 7.3 and Lemma 7.7 use both properties on *squeezable* systems and the global properties to prove that squeezable systems are a worst case in Corollary 7.8. Using the worst case, Theorem 7.9 states that ACCEPTOR is $\frac{2m-1}{m}$-competitive on a cluster composed of $m$ processors. Then, Corollary 7.10 states that ACCEPTOR is 2-competitive.

Several useful definitions follow. As defined in Section 5, an $\text{Idle}_k$ instant is the earliest instant such as at least $k$ processors are idle. Formally, $\text{Idle}_k(J, m)$ is the earliest instant such as at least $k$ processors may be idle when the job set $J$ is scheduled upon $m$ processors, by any work-conserving scheduler. Unlike $\overline{\text{Idle}}_k$, this value is the exact instant. We also introduce the following new notations: $\text{Idle}_{\max}(S)$ is the length of the longest period during which a processor is idle when a mode change phase of a system $S$ is handled by ACCEPTOR. The required time to handle the mode change phase of $S$ with an optimal protocol (resp. ACCEPTOR) is denoted $|\,\text{OPT}(S)|$ (resp. denoted $|\,\text{US}(S)|$). An optimal protocol doesn't have the same limitation as ACCEPTOR. For example, it doesn't have to schedule the rem-jobs using a work-conserving scheduler. We also introduce the notion of *excess idle*, denoted $\text{Idle}_+$. This is the extra duration where processors are idle during a mode change with our protocol, in comparison to an optimal. For example, in Fig. 6 $\text{Idle}_+ = 2 \times 6 = 12$. Finally, we define the notion of *squeezable* system, in the context of a mode change transition.

***Squeezable*** A system $S$ is said to be *squeezable* if and only if $S$ has only one required reconfiguration $\theta_c$ —with a reconfiguration delay of $\delta_c$— to perform during the mode change transition and a set of rem-jobs $J$ on a cluster having $m$ processors such that:

i $\sum C_i = \delta_c \times (m-1)$,
ii $\text{Idle}_1(J,m) = \text{Idle}_2(J,m) = \cdots = \text{Idle}_m(J,m) = \delta_c \times \frac{m-1}{m}$,
iii $\text{Idle}_1(J,m-1) = \text{Idle}_2(J,m-1) = \cdots = \text{Idle}_{m-1}(J,m-1) = \delta_c$,
iv $\forall i, d_i \geq \delta_c$.

This notions will be heavily used in the following lemmas, corollaries and theorems, see Fig. 6 for an illustration.

In the following, we assume that any system S has the required reconfigurations $\tilde{\Theta}$ to perform, and a set of rem-jobs $J$ to execute on a cluster having $m$ processors during any mode change. For such a system, we denote the longest atomic duration —either the WCET of a job or a reconfiguration delay— the value $C \doteq \max\{\max_{i=1}^{n}\{C_i\}, \max_{\theta_c \in \tilde{\Theta}}\{\delta_c\}\}$.

LEMMA 7.1. *Any optimal mode change of any* squeezable *system $S$ may be handled in $\delta_c$ time-units. Formally, $|\,\text{OPT}(S)| = \delta_c = C$.*

PROOF. The rem-jobs can all be scheduled on $m-1$ processors, and one processor can be reconfigured at $t = 0$. The rem-jobs will be complete at $t = \text{Idle}_{m-1}(J, m-1) = \delta_c$ and the reconfiguration will be done at $t = \delta_c$. Because of iv), the deadlines will be respected. By construction, $\delta_c = C$. The lemma follows. This lemma is illustrated in Fig. 6. □

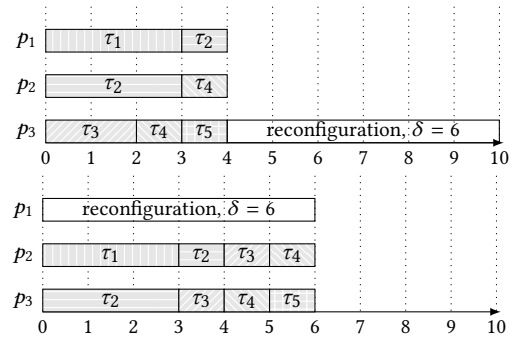


**Figure 6: A *squeezable* system with $m = 3$ and $n = 5$**

LEMMA 7.2. *Any optimal mode change of any system $S$ takes more than $C$ time-units. Formally, $|\,\text{OPT}(S)| \geq C$.*

PROOF. No intra-parallelism is allowed, so the system $S$ may not last less than the longest element to schedule, whether it is a job or a reconfiguration. □

COROLLARY 7.3. *For any mode change of any system $S$, the ratio $\frac{C}{|\,\text{OPT}(S)|}$ is maximised when $|\,\text{OPT}(S)| = C$.*

PROOF. This is a direct implication of Lemma 7.2. □

LEMMA 7.4. *During any mode change handled by ACCEPTOR of any system $S$, a processor cannot be idle longer than $C$ time-units. Formally, $\text{Idle}_{\max}(S) \leq C$.*

PROOF. ACCEPTOR uses a work-conserving scheduler. Thus, a processor may be idle only if the number of jobs and reconfigurations waiting to be executed or performed is lower than the number of processors. It immediately follows that all the reconfigurations and jobs will be completed at most $C$ time-units later. The lemma follows. □

LEMMA 7.5. *For any mode change of any system $S$, $|\,\text{US}(S)| = |\,\text{OPT}(S)| + \frac{\text{Idle}_+}{m}$.*

PROOF. We introduce here the notion of $\text{Idle}_{\text{opt}}$. The value $\text{Idle}_{\text{opt}}$ is the amount of idle time with an optimal protocol. It stands that:

$$m \times |\,\text{OPT}(S)| = \sum_{i=1}^{n} C_i + \sum_{\theta_c \in \tilde{\Theta}} \delta_c + \text{Idle}_{\text{opt}}$$

Informally, this equation represents the sum of the work to perform plus the idle time. By definition of the excess idle, it stands that:

$$m \times |\,\text{US}(S)| = \sum_{i=1}^{n} C_i + \sum_{\theta_c \in \tilde{\Theta}} \delta_c + \text{Idle}_{\text{opt}} + \text{Idle}_+$$

With the two previous equations, we trivially have that:

$$|\,\text{US}(S)| = \frac{\sum_{i=1}^{n} C_i + \sum_{\theta_c \in \tilde{\Theta}} \delta_c + \text{Idle}_{\text{opt}} + \text{Idle}_+}{m} = |\,\text{OPT}(S)| + \frac{\text{Idle}_+}{m}$$

. □

Lemma 7.6. *For any mode change of any system $S$ handled by* acceptor*,*

$$\text{Idle}_+ \leq (m-1) \times C$$

Proof. By definition, the excess idle is smaller than the sum of the length of all the periods where a processor is idle. At most $m-1$ processors will be idle, because at least one processor will never be idle. Therefore:

$$\begin{aligned} \text{Idle}_+ &\leq (m-1) \times \text{Idle}_{\max}(S) \\ &\leq (m-1) \times C \qquad \text{(Lemma 7.4)} \end{aligned}$$

The lemma follows. □

Lemma 7.7. *For any mode change of any system $S$ handled by* acceptor*, when $\frac{C}{|\text{OPT}(S)|}$ is maximised, $\frac{|\text{US}(S)|}{|\text{OPT}(S)|}$ is maximised as well.*

Proof.

$$\begin{aligned} &|\text{US}(S)| = |\text{OPT}(S)| + \frac{\text{Idle}_+}{m} && \text{(Lemma 7.5)} \\ \Rightarrow &\frac{|\text{US}(S)|}{|\text{OPT}(S)|} = \frac{|\text{OPT}(S)| + \frac{\text{Idle}_+}{m}}{|\text{OPT}(S)|} \\ \Rightarrow &\frac{|\text{US}(S)|}{|\text{OPT}(S)|} \leq \frac{|\text{OPT}(S)| + C \times \frac{m-1}{m}}{|\text{OPT}(S)|} && \text{(Lemma 7.6)} \\ \Rightarrow &\frac{|\text{US}(S)|}{|\text{OPT}(S)|} \leq 1 + \frac{C}{|\text{OPT}(S)|} \times \frac{m-1}{m} \end{aligned}$$

□

Corollary 7.8. *When handling any mode change of any* squeezable *system $S$ with* acceptor*, the ratio $\frac{|\text{US}(S)|}{|\text{OPT}(S)|}$ is maximised. Thus, any* squeezable *system is a worst-case of* acceptor*.*

Proof. Lemma 7.1 and Corollary 7.3 states that this corollary's hypotheses leads to a maximal ratio $\frac{C}{|\text{OPT}(S)|}$ and Lemma 7.7 shows that maximising the ratio $\frac{C}{|\text{OPT}(S)|}$ is equivalent to maximising the ratio $\frac{|\text{US}(S)|}{|\text{OPT}(S)|}$. Thus, this corollary follows. □

Theorem 7.9. *For any system composed of $m$ processors, the protocol* acceptor *is $\frac{2m-1}{m}$-competitive.*

Proof. To prove that acceptor is $\frac{2m-1}{m}$-competitive, we have to prove that for any system $S$, the ratio $\frac{|\text{US}(S)|}{|\text{OPT}(S)|} \leq \frac{2m-1}{m}$. The Corollary 7.8 states that the maximal ratio will be obtained on any *squeezable* system $S$. On such a system, the ratio

$$\begin{aligned} \frac{|\text{US}(S)|}{|\text{OPT}(S)|} &= \frac{\text{Idle}_1(J,m) + \delta_c}{\delta_c} \\ &= \frac{\delta_c \times \frac{m-1}{m} + \delta_c}{\delta_c} \\ &= \frac{2m-1}{m} \end{aligned}$$

Thus, acceptor takes at most $\frac{2m-1}{m}$ of the required time than an optimal protocol. The theorem follows.

Figure 6 shows an example of such a system with $m = 3$, $n = 5$ and $\delta_c = 6$. □

Corollary 7.10. *For any system composed of $m$ processors, the protocol* acceptor *is 2-competitive. This upper-bound is tight.*

Proof. Theorem 7.9 states that acceptor is $\frac{2m-1}{m}$-competitive. It stands that

$$\forall m > 0, \frac{2m-1}{m} < 2$$

Thus, acceptor is 2-competitive. This upper-bound is tight, because of the following limit:

$$\lim_{m \to \infty} \frac{2m-1}{m} = 2$$

□

## 7.4 Handling mode independent tasks

In our model, all tasks are mode dependent. Real-time applications may have mode independent tasks: i.e. tasks that run during the whole lifespan of the application such as the OS. Our model 'as is' does not permit such tasks to run. However, a trivial extension with no effect on the validity nor the complexity would be to restrain those tasks to a specific cluster with no reconfiguration allowed. Still, mode dependent tasks could run on such a cluster and because of Lemma 6.1, their rem-jobs would be correctly scheduled.

Doing so removes the limitation in a very easy way and makes the model usable for real-world applications.

# 8 CONCLUSIONS AND FUTURE WORK

In this research we propose a new model combining *dynamic* hardware *with* software reconfigurability. This is the *first* model for multi-mode real-time applications on reconfigurable heterogeneous platforms. We also propose a protocol to handle the mode change transition of such application and prove its validity. An upper-bound on the mode change transition is also introduced. Then, a complete evaluation of the contributions is done by evaluating the usability of the protocol (by evaluating its worst-case time complexity), its performance (by evaluating the competitiveness of the protocol, and the empirical pessimism of the bound) and discussing the constraint of the model on mode-independent tasks and showing that this is not a limitation.

Several extensions to this work can be done. First, the protocol could be improved in terms of competitiveness. We could improve the model to have a dynamic number of processors at run-time, which is allowed by FPGA reconfiguration. We could also try to introduce in a more advance way mode-independent tasks. Finally, other kind of possible future work is to put this model and protocol (and future extensions) into practice by implementing those in a Real-Time Operating System supporting a heterogeneous reconfigurable platform such as the Zynq UltraScale+™. Such experiments would validate the approach of this theoretical work in a close-to-industry, real-world scenario.

# ACKNOWLEDGEMENT

This work is supported in part by the H2020-ICT-2015 Innovation Action 688403 Tulipp and by the *Fédération Wallonie-Bruxelles*, Concerted Research Action named Sofist. Lastly, the authors would like to thank Dragomir Milojevic for his helpful collaboration, Juan

M. Rivas for his careful review of our manuscript and the reviewers for their insightful comments.

## REFERENCES


[1] T. Amert, N. Otterness, M. Yang, J. H. Anderson, and F. D. Smith. 2017. GPU Scheduling on the NVIDIA TX2: Hidden Details Revealed. In *2017 IEEE Real-Time Systems Symposium (RTSS)*. 104–115. https://doi.org/10.1109/RTSS.2017.00017

[2] Enrico Bini. 2016. Adaptive Fair Scheduler: Fairness in Presence of Disturbances. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems (RTNS '16)*. ACM, New York, NY, USA, 129–138. https://doi.org/10.1145/2997465.2997468

[3] A. Biondi, A. Balsini, M. Pagani, E. Rossi, M. Marinoni, and G. Buttazzo. 2016. A Framework for Supporting Real-Time Applications on Dynamic Reconfigurable FPGAs. In *2016 IEEE Real-Time Systems Symposium (RTSS)*. 1–12. https://doi.org/10.1109/RTSS.2016.010

[4] A. Biondi and G. Buttazzo. 2017. Timing-aware FPGA partitioning for real-time applications under dynamic partial reconfiguration. In *2017 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. 172–179. https://doi.org/10.1109/AHS.2017.8046375

[5] Martin Cornil, Antonio Paolillo, Joël Goossens, and Ben Rodriguez. 2017. Research and implementation challenges of RTOS support for heterogeneous computing platforms. In *Heterogeneous Architectures and Real-Time Systems Seminar, Brussels*.

[6] Liliana Cucu-Grosjean and Joël Goossens. 2010. Predictability of Fixed-Job Priority schedulers on heterogeneous multiprocessor real-time systems. *Inf. Process. Lett.* 110, 10 (2010), 399–402. https://doi.org/10.1016/j.ipl.2010.03.009

[7] Matthias Kalle Dalheimer and Matt Welsh. 2005. *Running Linux* (5th ed.). O'Reilly, Chapter 10, 382–383.

[8] Edsger W. Dijkstra. 1968. Letters to the editor: go to statement considered harmful. *Commun. ACM* 11, 3 (1968), 147–148. https://doi.org/10.1145/362929.362947

[9] Paul Emberson and Iain Bate. 2007. Minimising task migration and priority changes in mode transitions. In *13th IEEE Real Time and Embedded Technology and Applications Symposium*. IEEE, 158–167.

[10] S. Funk, J. Goossens, and S. Baruah. 2001. On-line scheduling on uniform multiprocessors. In *Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001) (Cat. No.01PR1420)*. 183–192. https://doi.org/10.1109/REAL.2001.990609

[11] Joël Goossens, Shelby Funk, and Sanjoy K. Baruah. 2003. Priority-Driven Scheduling of Periodic Task Systems on Multiprocessors. *Real-Time Systems* 25, 2-3 (2003), 187–205. https://doi.org/10.1023/A:1025120124771

[12] Joël Goossens and Xavier Poczekajlo. 2017. Multimode application on a reconfigurable platform: Introducing a new model and a first protocol. In *11th Junior Researcher Workshop on Real-Time Computing*. 1–4.

[13] Joël Goossens and Pascal Richard. 2013. Partitioned scheduling of multimode multiprocessor real-time systems with temporal isolation. In *Proceedings of the 21st International Conference on Real-Time Networks and Systems*. ACM, 297–305.

[14] Rhan Ha and Jane W.-S. Liu. 1994. Validating Timing Constraints in Multiprocessor and Distributed Real-Time Systems. In *Proceedings of the 14th International Conference on Distributed Computing Systems, Poznan, Poland, June 21-24, 1994*. 162–171. https://doi.org/10.1109/ICDCS.1994.302407

[15] Laura Hopperton. 2011. embedded world: Xilinx introduces 'industry's first' extensible processing platform. (2011). https://goo.gl/EuNhir

[16] Engin Ipek, Meyrem Kirman, Nevin Kirman, and Jose F. Martinez. 2007. Core Fusion: Accommodating Software Diversity in Chip Multiprocessors. In *Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA '07)*. ACM, 186–197. https://doi.org/10.1145/1250662.1250686

[17] J. Liedtke. 1995. On Micro-kernel Construction. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles (SOSP '95)*. ACM, New York, NY, USA, 237–250. https://doi.org/10.1145/224056.224075

[18] Chung Laung Liu and James W Layland. 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)* 20, 1 (1973), 46–61.

[19] José Marinho, Gurulingesh Raravi, Vincent Nélis, and Stefan M Petters. 2011. Partitioned Scheduling of Multimode Systems on Multiprocessor Platforms: when to do the Mode Transition? *RTSOPS* (2011).

[20] Vincent Nélis. 2010. *Energy-Aware Real-Time Scheduling in Embedded Multiprocessor Systems.* Ph.D. Dissertation. Université libre de Bruxelles.

[21] Vincent Nélis, Björn Andersson, José Marinho, and Stefan M. Petters. 2011. Global-EDF Scheduling of Multimode Real-Time Systems Considering Mode Independent Tasks. In *23rd Euromicro Conference on Real-Time Systems, ECRTS 2011, Porto, Portugal, 5-8 July, 2011*. 205–214. https://doi.org/10.1109/ECRTS.2011.27

[22] Vincent Nélis, Joël Goossens, and Björn Andersson. 2009. Two Protocols for Scheduling Multi-mode Real-Time Systems upon Identical Multiprocessor Platforms. In *Euromicro Conference on Real-Time Systems*. 151–160.

[23] M. Pagani, A. Balsini, A. Biondi, M. Marinoni, and G. Buttazzo. 2017. A Linux-based support for developing real-time applications on heterogeneous platforms with dynamic FPGA reconfiguration. In *2017 30th IEEE International System-on-Chip Conference (SOCC)*. 96–101. https://doi.org/10.1109/SOCC.2017.8226015

[24] Marco Pagani, Mauro Marinoni, Alessandro Biondi, Alessio Balsini, and Giorgio Buttazzo. 2016. Towards real-time operating systems for heterogeneous reconfigurable platforms. *OSPERT 2016* (2016), 49.

[25] Gurulingesh Raravi, Björn Andersson, Vincent Nélis, and Konstantinos Bletsas. 2014. Task assignment algorithms for two-type heterogeneous multiprocessors. *Real-Time Systems* 50, 1 (2014), 87–141. https://doi.org/10.1007/s11241-013-9191-3

[26] Jorge Real and Alfons Crespo. 2004. Mode Change Protocols for Real-Time Systems: A Survey and a New Proposal. *Real-Time Systems* 26, 2 (2004), 161–197. https://doi.org/10.1023/B:TIME.0000016129.97430.c6

[27] Ahmad Sadek, Ananya Muddukrishna, Lester Kalms, Asbjørn Djupdal, Ariel Podlubne, Antonio Paolillo, Diana Goehringer, and Magnus Jahre. 2018. Supporting Utilities for Heterogeneous Embedded Image Processing Platforms (STHEM): An Overview. In *Applied Reconfigurable Computing. Architectures, Tools, and Applications*, Nikolaos Voros, Michael Huebner, Georgios Keramidas, Diana Goehringer, Christos Antonopoulos, and Pedro C. Diniz (Eds.). Springer International Publishing, Cham, 737–749.

[28] Chi-Sheng Shih and Chang-Min Yang. 2017. Schedulability Analysis of Mode Change for Imprecise Computation on Multi-Core Platforms. In *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*. ACM, 261–268.

[29] Xilinx. 2018. Zynq UltraScale+ MPSoC. (2018). https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html

[30] Christian Kohn (Xilinx). 2013. Partial Reconfiguration of a Hardware Accelerator on Zynq-7000 All Programmable SoC Devices (XAPP1159). (2013).