Multimode application on a reconfigurable platform

Introducing a new model and a first protocol

Joël Goossens
Faculté des Sciences of the Université libre de Bruxelles Brussels, Belgium
Joel.Goossens@ulb.ac.be

Xavier Poczekajlo[*]
Faculté des Sciences of the Université libre de Bruxelles Brussels, Belgium
Xavier.Poczekajlo@ulb.ac.be

ABSTRACT

We consider the new problem of multimode applications for reconfigurable platforms in the context of hard real-time scheduling. In this problem, the taskset **and the hardware** may change over the time whenever the current mode of the system changes. Ensuring schedulability here requires to prove *(i)* The system schedulability of every mode, *(ii)* That any allowed mode change can take place with respect to the given timing constraints. Solving *(i)* is a schedulability problem upon heterogeneous systems. We propose a model to formalise the whole problem, and a first protocol to run any allowed mode change in the system with respect to the timing constraints. Finally, we propose a validity test to ensure that the property *(ii)* is respected.

Keywords

Multimode application; Reconfigurable system; Hard real-time; Heterogeneous system

1. INTRODUCTION

Hard real-time systems become more and more complex. Their correctness must be proven to ensure the safety of the system. When using a classic mono-mode application, proving the system correctness often leads to over-approximation of the workload. On certain systems, where some functions are executed only in certain situations, it is useful to use a more realistic and advanced model. The multimode application model fulfils this role. For an example, the application of an airplaine system will have very different *modes* depending on whether the airplane is on the ground before the take-off, or in cruise. Splitting the whole taskset into several sub-tasksets allows more precise bounding on the overall load at any instant. This is crucial when defining the system requirements, and may lead to huge gains in term of system capacity.

On a multimode application, the running taskset changes over the system lifespan. It may be interesting to adapt the system as well to fit each taskset the best. Today's FPGAs allow run-time hardware reconfiguration at high rate. Even more interesting, FPGAs allow dynamic partial reconfiguration (DPR): an FPGA may be divided into several partitions each having different configurations, and a single partition may be reconfigured without jeopardising the whole system. [1] describes, in a low-level approach, how DPR may be used and also gives more details about the current level of performance of the existing FPGAs. It is also possible to use processors which can change their speed at run-time.

Related work. The survey [5] proposes various solutions for a multimode application on a *uni*processor system. More important, it unifies a vocabulary for mode change applications. For an example, the notions of *periodicity* and *synchronous* or *asynchronous* protocols are as useful in uniprocessor systems as in multiprocessor systems. Based on those notions, [3] proposes the first multiprocessors protocols: a synchronous protocol *SM-MSO* and an asynchronous protocol *AM-MSO*. It also computes an upper-bound for the *makespan* of a taskset on a given system, which represents the required time to scheduled the pending jobs during the mode change (at most one job per task). This upper bound is then refined in [2].

Multimode protocols, as they currently exist, only handle the transition phase. During each mode execution, a scheduler must be used to handle the taskset. Multiprocessor systems may be scheduled by partitioned, global, semi-partitioned, or clustered algorithms. The latest has been well studied for heterogeneous system, and [4] proposes LPG_{IM}. It is, to the best of our knowledge, one of the most efficient approach for this problem. LPG_{IM} considers the system as clusters of identical processors and assigns a sub-taskset to each cluster. Inside the cluster, a global scheduler may be used. The problem of heterogeneous system is hence reduced to identical multiprocessor systems. This approach may be used in the context of multimode application on reconfigurable systems.

To the best of our knowledge, no such model nor mode change protocol exist, where both the hardware and the software can change during the lifespan of the system.

Contributions. In this paper, we introduce the first model where the taskset *and* the hardware may change over the time. We also propose a first synchronous protocol for such application and its feasibility test.

2. MODEL

A reconfigurable multiprocessor system is a system composed of *partitions*. A partition can be any computing unit, as an FPGA partition or a general purpose processor. Each partition can implement a *configured processor* through reconfiguration. The *configured processor* behaviour is defined by its *configuration*. In the case of a processor with a fixed behaviour, it is modelised as a partition with only two configurations: *on* and *off*. From now on, *configured processors*

[*]Corresponding author.

will be denoted as *processors*.

A multimode application for a reconfigurable system is defined by a set of x different modes $M \overset{\text{def}}{=} \{M^1, M^2, ..., M^x\}$. Each mode $M^h \overset{\text{def}}{=} \langle \tau^h, S_h, \widetilde{\Theta}_h, \Delta_h \rangle$ has to execute a taskset τ^h with the given scheduler S_h on a system composed of a set of processors (through partitions reconfiguration) with respect to the configuration multiset $\widetilde{\Theta}_h \in \Theta^m$ (see section 2.2.2). When the system requires a mode change to M^h, it must be done within a delay of Δ_h unit of time.

The system activates and deactivates the mode with the following constraint: at any time, one and only one mode can be active in the system. When a mode is activated, its taskset is *enabled*.

2.1 Taskset model

The taskset $\tau \overset{\text{def}}{=} \cup_{h=1}^{x} \tau^h$ is composed of several sub-tasksets τ^h, each one being the specific taskset of the h^{th} mode.

Each taskset is composed of n_h tasks: $\tau^h \overset{\text{def}}{=} \{\tau_1^h, \ldots, \tau_{n_h}^h\}$.

Each task τ_i^h is a sporadic task, defined with three parameters $\langle C_i^h, D_i^h, T_i^h \rangle$ — a worst-case execution time, a relative deadline, and a minimum inter-arrival time.

When a mode's taskset is *enabled*, all its tasks are enabled. At most one taskset can be enabled at any instant in the system. When a task τ_i^h is enabled, it may release a job with respect to the minimum inter-arrival time T_i^h. When a taskset is disabled, all its tasks are disabled and may not release new jobs. However, if a job of a disabled taskset is not complete, it must be completed with respect to its absolute deadline. Those jobs are called *rem-jobs* in the following.

2.2 System model

2.2.1 Processors and partitions

The system is composed of m reconfigurable partitions denoted $P \overset{\text{def}}{=} \{p_1, p_2, \ldots, p_m\}$. The m partitions can dynamically be configured as m processors with a specific configuration. This operation is denoted as *reconfiguration*.

A partition has a type, depending on whether the partition represents a specific FPGA partition, or an other type of computing unit. A partition of type ρ must always be configured in a specific configuration. If the partition is not used, it must be configured in the configuration $\theta_{\rho,0}$ later defined in Section 2.2.2.

The configured partitions form a system of m *configured processors* $\Pi \overset{\text{def}}{=} \{\pi_1, \pi_2, \ldots, \pi_m\}$, denoted as processors. Hence, they form an unrelated system of several *clusters*. A *cluster* is a set of identical processors, i.e., partitions from the same type sharing the same configuration.

P_h represents all the used partitions of a mode M^h and $P_{h,\text{cl}}$ represents the used partitions of a given cluster cl for the mode M^h.

2.2.2 Partitions configuration

Reconfiguration can occur during a transition phase. $\Theta \overset{\text{def}}{=} \cup_\rho \Theta_\rho$ represents the set of all available configurations set in the system. A configuration set Θ_ρ represents all the configurations available for partitions of type ρ (e.g. a partition of FPGA with a given number of logic blocks).

$\Theta_\rho \overset{\text{def}}{=} \{\theta_{\rho,0}, \ldots, \theta_{\rho,\ell_\rho}\}$ where:

- $\theta_{\rho,0}$ is the configuration where the partition is off, on which no task can be scheduled. A partition configured with $\theta_{\rho,0}$ is *free*.
- $\theta_{\rho,1}, \ldots, \theta_{\rho,\ell_\rho}$ represent ℓ_ρ different configurations. Each configuration executes the different tasks at an *unrelated* speed, i.e., the speed of the processor configured as such depends on the job being executed. Its partition is *used*.

δ_z represents the delay to change to a specific configuration $z \in \Theta$. During this reconfiguration time, the processor cannot execute any task. It is important to note that the reconfiguration time does not depend on the previous configuration of the partition. The reconfiguration time varies because the partitions may be from different hardwares like different FPGA models.

For any partition p, $\theta(p)$ represents the current configuration of p. $\widetilde{\Theta}_{h,\rho}$ represents the configurations of mode h for the partitions of type ρ.

2.2.3 Tasks progression rate

Because the system is heterogeneous, the job speed depends on the type of the processor which executes the job. Those data are an input of the schedulers which are black boxes. Hence, there are omitted because of space constraints.

2.3 Mode transition

The system may receive a *Mode Change Request* MCR(h) to the destination mode h, whenever the system is not handling a transition phase. This instant is denoted $t_{\text{MCR}(h)}$.

The transition graph represents all the possible configuration transitions. A transition between M^{src} and M^{dst} is possible if and only if there is an edge from M^{src} to M^{dst} in the transition graph, denoted by $(M^{\text{src}}, M^{\text{dst}})$.

The system enters a reconfiguration phase at $t_{\text{MCR(dst)}}$. It must then reconfigure itself according to the new mode within the given delay Δ_{dst}. After this delay, the destination mode M^{dst} is activated and the transition phase ends.

For a given mode M^h, $\widetilde{\Theta}_h$ contains the required configurations for the execution of τ^h.

After a *Mode Change Request* to mode M^h which occurred at $t_{\text{MCR}(h)}$, the system must be reconfigured before the instant equal to $t_{\text{MCR}(h)} + \Delta_h$ to be feasible.

3. PROTOCOL

3.1 Protocol

The mode-change protocol must ensure that the rem-jobs are correctly scheduled and that the system is able to activate the new mode within the given delay. For that purpose, it is composed of an offline computation phase, and of two run-time phases. The offline phase computes for each couple $(M^{\text{src}}, M^{\text{dst}})$ of the source mode M^{src} the necessary reconfigurations.

The run-time phase 1 begins at $t_{\text{MCR(dst)}}$ and is completed for each cluster, when all of their processors are idle.

The run-time phase 2 is the reconfiguration of the required partitions.

3.1.1 Hypothesis on the schedulers

We consider in this protocol only clustered schedulers. Unlike global schedulers, clustered schedulers allow tasks to migrate *only* between the processors of the cluster where the task is statically assigned. In addition, we consider only schedulers that are preemptive, fixed-job priority and work-conservative. We use the notions defined in [3].

The taskset of a mode h is divided into several sub-tasksets, one per cluster: $\tau^h = \cup_{\text{cl}}\ \tau^{h,\text{cl}}$. The scheduler S_h may use a specific scheduling policy for each cluster cl which respects the three assumptions aforementioned. Each scheduling policy must schedule the sub-taskset $\tau^{h,\text{cl}}$ of each cluster cl feasibly. Every $\tau^{h,\text{cl}}$ is determined at design time.

3.1.2 *Offline computation*

The offline computation creates two tables per couple $(M^{\text{src}}, M^{\text{dst}})$ from the transition graph: the Empty Partitions Reconfigurations table (EPRT) and the Used Partitions Reconfigurations table (UPRT). Those tables are necessary to reconfigure the system after a MCR(dst) when the mode M^{src} is active.

Each table depends on both the source mode M^{src} and the destination mode M^{dst}. Their computations use the makespan upper bound of each cluster of M^{src} (see [3]).

The Empty Partitions Reconfigurations table contains a list of configurations required by M^{dst} not used by M^{src}. Those reconfigurations will be launched at $t_{\text{MCR(dst)}}$. Its computation is described by the Algorithm 1.

The Used Partitions Reconfigurations table contains a list of couple of configurations. A couple $(\theta_{z_1}, \theta_{z_2})$ in the UPRT indicates that a partition configured in θ_{z_1} will be reconfigured in θ_{z_2}. Its computation is described by the Algorithm 2.

Algorithm 1: Creation of the EPRT for $(M^{\text{src}}, M^{\text{dst}})$

```
Input: $M^{\text{src}}$: the source Mode
       $M^{\text{dst}}$: the destination Mode
Output: EPRT: the EPRT for ($M^{\text{src}}$, $M^{\text{dst}}$)

begin
  let EPRT be an empty multiset

  For each type of partition $\rho$
    let $\overrightarrow{\Theta_{\text{dst},\rho}} \overset{\text{def}}{=} \widetilde{\Theta}_{\text{dst},\rho} \setminus \widetilde{\Theta}_{\text{src},\rho} \cap \widetilde{\Theta}_{\text{dst},\rho}$: a vector
    Order $\overrightarrow{\Theta_{\text{dst},\rho}}$ by reconfiguration time,
        in decreasing order
    let free = the number of free partitions
        of type $\rho$ for the mode $M^{\text{src}}$
    Add the free$^{\text{th}}$ first elements of $\overrightarrow{\Theta_{\text{dst},\rho}}$
        in EPRT
end
```

Algorithm 2: Creation of the UPRT for $(M^{\text{src}}, M^{\text{dst}})$

```
Input: $M^{\text{src}}$: the source Mode
       $M^{\text{dst}}$: the destination Mode
Output: UPRT: the UPRT for ($M^{\text{src}}$, $M^{\text{dst}}$)

begin
  let UPRT be an empty multiset

  For each cluster cl of $M^{\text{src}}$
    For each $p \in P_{src,\text{cl}}$
      makespan($p$) $\overset{\text{def}}{=}$ makespan(cl)

  For each type of partition $\rho$
    let $\overrightarrow{\Theta_{\text{src},\rho}}$ the vector of config.
      for used partitions of type $\rho$ in $M^{\text{src}}$
    Order $\overrightarrow{\Theta_{\text{src},\rho}}$ by makespan
    let $\overrightarrow{\Theta_{\text{dst},\rho}} \overset{\text{def}}{=} \widetilde{\Theta}_{\text{dst},\rho} \setminus \widetilde{\Theta}_{\text{src},\rho} \cap \widetilde{\Theta}_{\text{dst},\rho}$: a vector
    Order $\overrightarrow{\Theta_{\text{dst},\rho}}$ by reconfiguration time,
        in decreasing order
    let free = the number of free partitions
       of type $\rho$ for the mode $M^{\text{src}}$
    Remove from $\overrightarrow{\Theta_{\text{dst},\rho}}$ the free$^{\text{th}}$ first el.
    if $|\overrightarrow{\Theta_{\text{dst},\rho}}| > 0$
      For each $n^{th}$ el. of $\overrightarrow{\Theta_{\text{dst},\rho}}$
        let $\theta_{\text{dst},\rho} = \overrightarrow{\Theta_{\text{dst},\rho}}(n)$
        add to UPRT: $(\overrightarrow{\Theta_{\text{src},\rho}}(n), \theta_{\text{dst},\rho})$
end
```

3.1.3 *Run-time phase 1: Schedule rem-jobs*

This phase relies heavily on the SM-MSO protocol from [3].

At the MCR(M^{dst}), the protocol deactivates the mode M^{src} and disables all the current enabled tasks. Then, it keeps the scheduler S^{src} to schedule the rem-jobs until idle-time is reached for every processor.

Because the scheduler S^{src} can schedule τ^{src} with no deadline miss: the rem-jobs will be feasibly scheduled by using the same scheduler.

3.1.4 *Run-time phase 2: Reconfiguration*

At the MCR(M^{dst}), the unused partitions of M^{src} are reconfigured. For each element θ_{z_1} in UPRT, the protocol reconfigures a free partition to the configuration θ_{z_1}.

When all the processors of a cluster are idle, the protocol launches its required reconfigurations. Each partition p of the cluster picks, if possible, a couple $(\theta_{\text{src}}, \theta_{\text{dst}})$ in the EPRT where the source configuration θ_{src} is the current configuration of p, i.e., $\theta_{\text{src}} = \theta(p)$. An entry of the EPRT can be picked only once per transition phase. Then, the partition p is reconfigured to the destination configuration θ_{dst}.

When all the cluster are idle, the system can safely activate the mode M^{dst}, and enables all the tasks of τ^{dst}.

3.1.5 *Example*

To illustrate our protocol, we show an example of a multimode application with two modes M^1, M^2 (see Figure 1). The system is allowed to change from M^1 to M^2, and we show how works the transition phase. For that purpose, here are the partial specifications of the system:

- For mode M^1: $\widetilde{\Theta}_1 = \{\theta_0, \theta_1, \theta_1, \theta_2\}$, $\tau^1 = \{\tau^{1,1,}, \tau^{1,2}\}$, where $\tau^{1,1} = \{\tau_1, \tau_2, \tau_3\}, \tau^{1,2} = \{\tau_4, \tau_5\}$;
- For mode M^2: $\Delta_2 = 5.5$, $\widetilde{\Theta}_2 = \{\theta_2, \theta_4, \theta_4, \theta_5\}$, $\tau^2 = \{\tau^{2,1}, \tau^{2,2}, \tau^{2,3}\}$, where $\tau^{2,1} = \{\tau_6, \tau_7, \tau_8\}$, $\tau^{2,2} = \{\tau_9\}$, $\tau^{2,3} = \{\tau_{10}\}$;
- Reconfiguration time for θ_4, θ_5: $\delta_{\theta_4} = 2$, $\delta_{\theta_5} = 1$;
- Makespan upper-bound for mode M^1 clusters: $\mathit{makespan}(\tau^{1,1}) = 3.5$, $\mathit{makespan}(\tau^{1,2}) = 4.5$;
- EPRT$(M^1, M^2) = \{\theta_4\}$,
  UPRT$(M^1, M^2) = \{(\theta_1, \theta_4), (\theta_1, \theta_5)\}$;
- Tasks specification are omitted because of space constraint and relevance;
- All the partitions have the same type and hence the same available configurations.

At $t = 0$, each active task releases a job.

At $t = 3$, τ_3 releases a new job.

At $t = 4$, τ_1, τ_2 and τ_4 release a new job.

At $t = 4.5$ a MCR(2) occurs: the mode M^2 must be activated by $t_{\text{MCR(2)}} + \Delta_2 = 4.5 + 5.5$. However, $\tau_1, \tau_2, \tau_3, \tau_4$ have active rem-jobs. For an example, the second job of τ_1 was released at 4 and must be completed. The scheduler used by M^1 is kept to schedule the rem-jobs. p_1 is free in the mode M^1: it is immediately reconfigured to a configuration in the EPRT: θ_4.

At $t = 5.5$, the cluster $\tau^{1,2}$ is idle, but no couple (τ_2, X) exists in the UPRT: p_4 is not reconfigured. At $t = 7$, the

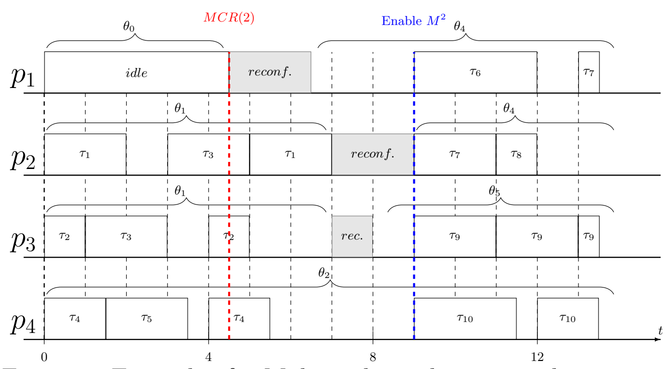

Figure 1: Example of a Multimode application with reconfigurations

cluster $\tau^{1,1}$ is idle, so each configuration picks a couple in the EPRT: (θ_1, θ_4) and (θ_1, θ_5).

At $t = 9$, all the partitions are idle. The mode M^2 is activated, and its taskset τ^2 enabled. Because $t_{\mathrm{MCR}(2)} + \Delta_2 \geqslant 9$, the transition delay time constraint was respected.

3.2 Validity test

We provide a validity test for our protocol. The validity test is a sufficient condition which indicates whether a given multimode application is feasible or not, i.e., if all the deadlines of the rem-jobs will be met and if the transition phase delay will be respected, for any mode change allowed by the transition graph.

By construction, all the deadlines will be met during a transition phase. However, we must provide an upper bound of the transition phase for every mode M^{dst}. This upper bound depends on the upper-bound of the makespan of a cluster ms(src, cl), defined by [2] (see Corollary 2.2). Without loss of generality, we assume the tasks to be ordered by processing time.

$$\mathrm{ms}(\mathrm{src}, \mathrm{cl}) \overset{\mathrm{def}}{=} \begin{cases} c_{|\tau^{\mathrm{src,cl}}|}, \text{ if } |\tau^{\mathrm{src,cl}}| = |P_{\mathrm{cl}}| \\ \frac{\sum_{i=1}^{|\tau^{\mathrm{src,cl}}|-1} c_i}{|P_{\mathrm{cl}}|} + c_{|\tau^{\mathrm{src,cl}}|}, \text{ otherwise} \end{cases}$$

For any couple $(M^{\mathrm{src}}, M^{\mathrm{dst}})$, the upper bound for an empty partition reconfiguration (EPR-UB) is:

$$\mathrm{EPR\text{-}UB}(M^{\mathrm{src}}, M^{\mathrm{dst}}) \overset{\mathrm{def}}{=} \max(\{\forall z \in \widetilde{\Theta}_{\mathrm{src}} \ \delta_z\})$$

For any couple $(M^{\mathrm{src}}, M^{\mathrm{dst}})$, the upper bound for an used partition reconfiguration (UPR-UB) is:

$$\mathrm{UPR\text{-}UB}(M^{\mathrm{src}}, M^{\mathrm{dst}}) \overset{\mathrm{def}}{=} \max(\{\forall \rho \ \ \mathrm{UPR\text{-}UB}(M^{\mathrm{src}}, M^{\mathrm{dst}}, \rho)\})$$

with

$$\mathrm{UPR\text{-}UB}(M^{\mathrm{src}}, M^{\mathrm{dst}}, \rho) \overset{\mathrm{def}}{=} \max(\{\forall i = 0..|\widetilde{\Theta}_{\mathrm{dst},\rho}| \ \delta_{z_{i,\rho}} + \mathrm{UPM}(\rho, i)\})$$

where

- z_{i_t} is the i^{th} element of the table of configurations of M^{dst} for partitions of type ρ, ordered by reconfiguration time in decreasing order and
- $\mathrm{UPM}(\rho, i)$ is the i^{th} element of the vector of the upper bound makespan of each partition of type ρ of P_{src}, ordered by duration increasing.

Therefore, for any couple $(M^{\mathrm{src}}, M^{\mathrm{dst}})$, the upper bound of a transition phase is:

$$\mathrm{UB}(M^{\mathrm{src}}, M^{\mathrm{dst}}) \overset{\mathrm{def}}{=} \max(\mathrm{EPR\text{-}UB}(M^{\mathrm{src}}, M^{\mathrm{dst}}), \mathrm{UPR\text{-}UB}(M^{\mathrm{src}}, M^{\mathrm{dst}}))$$

Hence, the upper bound for any transition allowed to M^{dst} is:

$$\max(\{\mathrm{UB}(M^{\mathrm{src}}, M^{\mathrm{dst}}) | (M^{\mathrm{src}}, M^{\mathrm{dst}}) \in \mathrm{TransitionGraph}\})$$

4. CONCLUSION AND FUTURE WORK

The technical advance for FPGAs leads us to consider a new paradigm for multimode application, with a system that is reconfigured in an efficient way for each mode. To the best of our knowledge, no such model currently exists. In this paper, we introduce the first model which fills that gap. This model can be seen as an extension of the multimode application model for multiprocessor systems. We also propose a synchronous protocol, associated with a validity test for any application and system on the new model. This protocol is more than a simple generalisation of existing protocols, using the makespan of each cluster to tighten the transition phase delay, and thus allowing more applications to pass the validity test provided by this article.

Future work. A first contribution would be to tighten the validity test by computing a makespan for each processor rather than each cluster. Furthermore, we intend to propose a more generic model. Our first model can be extended via several parameters. We want to be able to handle schedulers which allow inter-cluster migrations. We also want to introduce mode independent tasks to describe tasks that need to release jobs at any instant in the system, even during a transition phase (see periodicity in [5]). To fully take advantage of the mode independent tasks, we intend to propose an asynchronous protocol with periodicity.

In the current model, the different partition types do not share any configuration. A configuration for an FPGA partition can be used on an other FPGA partition with more logic blocks. It may be interesting to have configuration that are usable by different partition types to gain more flexibility.

5. REFERENCES

[1] A. Biondi, A. Balsini, M. Pagani, E. Rossi, M. Marinoni, and G. C. Buttazzo. A framework for supporting real-time applications on dynamic reconfigurable fpgas. In *Real-Time Systems Symposium*, pages 1–12, 2016.

[2] V. Nélis. *Energy-Aware Real-Time Scheduling in Embedded Multiprocessor Systems*. PhD thesis, Université libre de Bruxelles, 2010.

[3] V. Nélis, J. Goossens, and B. Andersson. Two protocols for scheduling multi-mode real-time systems upon identical multiprocessor platforms. In *Euromicro Conference on Real-Time Systems*, pages 151–160, 2009.

[4] G. Raravi, B. Andersson, V. Nélis, and K. Bletsas. Task assignment algorithms for two-type heterogeneous multiprocessors. *Real-Time Systems*, 50(1):87–141, 2014.

[5] J. Real and A. Crespo. Mode change protocols for real-time systems: A survey and a new proposal. *Real-Time Systems*, 26(2):161–197, 2004.