Timing-aware FPGA Partitioning for Real-Time Applications Under Dynamic Partial Reconfiguration

Alessandro Biondi and Giorgio Buttazzo
Scuola Superiore Sant'Anna, Pisa, Italy
Email: {alessandro.biondi, giorgio.buttazzo}@santannapisa.it

***Abstract*—Heterogeneous system-on-chips (SoC) that include both general-purpose processors and field programmable gate arrays (FPGAs) are emerging as very promising platforms to develop modern cyber-physical systems, combining the typical flexibility enabled by software with the speedup achievable by custom hardware accelerators. Furthermore, the dynamic partial reconfiguration (DPR) capabilities of modern FPGAs make such platforms even more attractive, offering the possibility of virtualizing the FPGA area to support several hardware accelerators in time sharing. However, heterogeneous platforms originate considerable challenges in the design and development process of applications, especially if timing and energy constraints are concerned.**

The FRED framework has been recently proposed to support the development of real-time applications upon such platforms, using a static slotted-based partitioning of the FPGA area to ensure predictable delays when managing custom hardware accelerators by DPR.

This paper addresses the problem of designing a suitable FPGA partitioning to support the execution of a real-time application within the FRED framework. The problem is formulated as a mixed-integer linear program that is in charge of (i) designing the size of the slots (in terms of FPGA resources), (ii) allocating hardware tasks to the slots, and (iii) selecting which hardware tasks must be statically allocated to the FPGA, while ensuring bounded worst-case response times on the tasks.

I. Introduction

Heterogeneous platforms including both general-purpose processors and *field programmable gate arrays* (FPGAs) are emerging as a solution for providing high computational performance with contained power dissipation. Properly distributing the computation between processors and FPGA, it is possible to combine the advantages of software flexibility and reusability with the speedup provided by the hardware acceleration. In addition, the *dynamic partial reconfiguration* (DPR) capability offered by modern FPGAs allows the application developer to reprogram a part of the FPGA area while the other parts continue to operate without interruption. During the years reconfiguration times have been progressively decreasing [1], thus enabling the possibility of virtualizing the FPGA area to support several hardware modules in time sharing, hence making these devices even more attractive.

In spite of these attractive features, several problems need to be solved to make such heterogeneous platforms suitable for being used in safety-critical real-time applications, such as space missions, automotive and robotic systems. Some of the major challenges are briefly listed below.

Predictability. The applications running on such platforms should be analyzable in terms of timing behavior, in order to guarantee bounded response times on all real-time computations carried out in the processors and on the FPGA.

Operating system support. A suitable software layer is needed in the operating system to simplify the programming of real-time applications, manage software and hardware tasks, and support their interaction by proper synchronization and communication mechanisms.

Resource allocation. Given an application consisting of a number of computations that have to be performed under a given set of constraints (e.g., periods, deadlines, energy consumptions, precedence relations), a method is needed to allocate the activities in such a way that all the constraints are satisfied and, possibly, some cost function is minimized.

Tools. To make these platforms practically usable by the industry, all the development phases (design, allocation, analysis, programming, and test) should be supported by proper tools that automatize and simplify the involved processes.

Recently, Biondi et al. [1] proposed a programming framework, called FRED, for supporting real-time applications on heterogeneous platforms consisting of a central processor unit (CPU) and an FPGA with DPR capability. The application consists of software tasks (SW-tasks) running on the CPU that can invoke the execution of hardware accelerators (HW-tasks) on the FPGA. The FPGA is virtualized so that the total number of HW-tasks that can run on the FPGA in time sharing is larger than the number of HW-tasks that could be accommodated in the same area according to a static allocation. Under the FRED framework, all the tasks experience bounded delays and a schedulability analysis has been provided to compute response times and verify the task set schedulability.

The authors also showed that there are applications guaranteed to be feasible when running in time sharing under the FRED framework, which could not be guaranteed otherwise, that is, neither when all the tasks are implemented in software (because response times would be too high), nor when all tasks are implemented in hardware (because the total FPGA area required by the application would exceed the available one).

To ensure predictability and a more robust inter-task communication, FRED relies on a static partitioning of the FPGA area, which is divided into a set of slots of fixed size. Each slot, however, can be shared by one or more HW-tasks whose size does not exceed the one of the slot. For this reason, each HW-task has an affinity value, specifying which slot it can be programmed into.

In that work, slots and affinities are assumed to be given and satisfy the area constraints. In practice, however, a design methodology is required to find a feasible partitioning, which

is not addressed by the authors. This paper fills this gap by focusing on such a design method.

Paper contributions. This work proposes an approach for computing a suitable FPGA partitioning to support the execution of a real-time application under the FRED framework. A *mixed-integer linear program* (MILP) formulation is presented to reach this objective. Given a modeling instance of a real-time application and a platform, the MILP produces as output (i) the design of the FPGA slots in terms of number of FPGA resources, (ii) the affinity of each HW-task, and (iii) the set of HW-tasks that must be statically allocated on the FPGA. The output solution ensures the feasibility of the application in terms of *timing constraints* and the feasibility of the FPGA partitioning in terms of resource availability. Both *preemptable* and *non-preemptable* FPGA reconfiguration interfaces are considered when accounting for timing constraints. The scalability and the performance of the proposed approach have been assessed with an experimental study.

Paper Organization. Section II presents the system model and summarizes some existing results derived for the FRED framework. Section III defines the problem addressed in the paper. Section IV presents the proposed solution for performing the FPGA partitioning. Section V reports some experimental results to evaluate the scalability and the performance of the proposed solution. Section VI discusses the related work. Section VII concludes the paper and illustrates possible future work.

II. Modeling and Background

This section briefly reviews the FRED framework proposed in [1], on which this work is based. It considers a heterogeneous computing system consisting of (at least) *one* processor and an FPGA with DPR capability, both sharing a common DRAM memory. The FPGA is composed of N^{RES} types of resources (e.g., configurable logic blocks, flip-flops, DSPs, etc.), each consisting of B^x units ($x = 1, \ldots, N^{\text{RES}}$). For a compact notation, $\mathbf{B}$ denotes the vector of FPGA resources, where B^x is its generic component.

Two types of computational activities are managed: **(i)** *software tasks* (SW-tasks): they are computational activities running on the processors; and **(ii)** *hardware tasks* (HW-tasks): they are hardware accelerators implemented in programmable logic and executed on the FPGA.

SW-tasks can speedup parts of their computation by requesting the execution of HW-tasks. To ensure predictability during FPGA reconfigurations and minimize the overhead related to the allocation of HW-tasks and the instantiation of communication channels, FRED relies on a *static partitioning* of the FPGA area into N^S slots. Each slot s_k ($k = 1, \ldots, N^S$) is composed of $\mathbf{b}_k$ resources, with $\sum_{k=1}^{N^S} \mathbf{b}_k \leq \mathbf{B}$. Resource units are not shared among the slots.

Since the goal of this work is to find a partitioning for the FPGA, in the following the parameters N_S and $\mathbf{b}_k$ ($k = 1, \ldots, N^S$) are assumed to be *unknown* and will be determined by the optimization procedure presented in Section IV as a function of a given real-time application.

Hardware Tasks. Each HW-task has a static *affinity* to a single slot and can execute only if it has been programmed into it. HW-tasks execute in a non-preemptive fashion. Each slot can be reconfigured at run-time by means of an *FPGA reconfiguration interface* (FRI). As for most platforms (e.g., [2]), the FRI **(i)** can reconfigure a slot without affecting the execution of the HW-tasks currently programmed in other slots; **(ii)** is a peripheral device external to the processor (e.g., like a DMA) and hence does not consume processor cycles to reconfigure slots; and **(iii)** can program at most one slot at a time. To program a given HW-task τ_i^H into a slot, the FRI has to program all its resources, independently of the number $\mathbf{b}_i$ of resource units required by τ_i^H, because unused resources have to be disabled to "clean" the previous slot configuration. The slot hosting a HW-task τ_i^H is denoted as $s(\tau_i^H)$ and also referred to as *affinity*. For all HW-tasks with affinity $s(\tau_i^H) = s_k$, it must be $\mathbf{b}_i \leq \mathbf{b}_k$.

The FRI is characterized by a *minimum throughput* ρ^x for each type of resource $x = 1, \ldots, N^{\text{RES}}$, meaning that at most $r_k^S = \sum_{x=1}^{N^{\text{RES}}} b_k^x / \rho^x$ units of time are needed to program a slot s_k.[1] Hence, the reconfiguration time r_a needed to program a HW-task τ_a^H is $r_a = r_k^S \ : \ s(\tau_a^H) = s_k$.

Software Tasks. Each SW-task can invoke multiple HW-tasks by alternating execution phases with *suspension* phases, where the SW-task is descheduled to wait for the completion of the corresponding HW-task. The same HW-task cannot be invoked by multiple SW-tasks. Every SW-task is activated periodically (or sporadically), with a period (or minimum interarrival time) T_i, thus producing a potentially infinite sequence of execution instances, denoted as *jobs*. Each SW-task is assigned a relative *deadline* D_i, meaning that each of its jobs must complete its execution within D_i units of time from its activation. The pseudo-code of a sample skeleton of a SW-task that uses two HW-tasks is shown in Listing 1. The *blocking* call `EXECUTE_HW_TASK` is in charge of (i) *requesting* the execution of a HW-task and (ii) *suspending* the execution of the SW-task until the completion of the requested HW-task. Figure 1 illustrates a typical schedule generated for the SW-task reported in Listing 1.

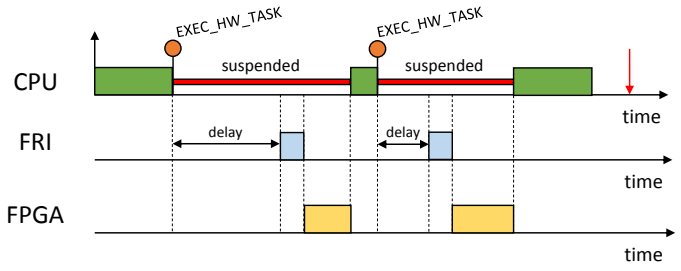

Figure 1. Typical schedule produced for the SW-task in Listing 1. The up-arrow and the down-arrow denote the release time and the deadline of the SW-task, respectively. Each HW-task request incurs in a variable delay, depending on the contention of the FRI and the FPGA slot availability.

A. Scheduling Infrastructure

To manage the contention of the FRI and the FPGA slots, FRED defines a scheduling mechanism based on a multi-level queueing structure represented in Figure 2. It includes n_p queues, one for each slot, needed to schedule the HW-tasks

[1] For the sake of simplicity, the overhead introduced by the header of the bitstreams is neglected.

```
1  SW_TASK(example)
2  {
3    << SW-task Initialization >>
4
5    // Initialization of the HW-tasks
6    HW_Task myHW_Task1 = hw_task_init(hw_task_1);
7    HW_Task myHW_Task2 = hw_task_init(hw_task_2);
8
9    // Task body
10   forever
11   {
12     << ... >>
13
14     << prepare input data for hw_task_1 >>
15     EXECUTE_HW_TASK(myHW_Task1);
16     << process output data of hw_task_1 >>
17
18     << ... >>
19
20     << prepare input data for hw_task_2 >>
21     EXECUTE_HW_TASK(myHW_Task2);
22     << process output data of hw_task_2 >>
23
24     << ... >>
25
26     // Wait for the next job
27     suspend_until(next_activation_time);
28   }
29 }
```

Listing 1. Pseudocode of a SW-task that invokes two HW-tasks.

having affinity with the same slot, and a *FRI queue* to schedule the reconfiguration requests. The slot queues are ordered using a first-in-first-out (FIFO) policy. Every time a SW-task issues a request $\mathcal{R}$ for a HW-task, $\mathcal{R}$ is assigned a *ticket* marked with the current absolute time. Then, $\mathcal{R}$ is inserted in the slot queue based on the affinity of the HW-task. Requests are inserted in a slot queue as long as the corresponding slot is used by another HW-task. The FRI queue is fed by the slot queues and ordered by *increasing ticket time*. Under such a scheduling mechanism, *the delays incurred by HW-task requests are bounded and can be computed using a response-time analysis*. Please refer to [1] for additional details.

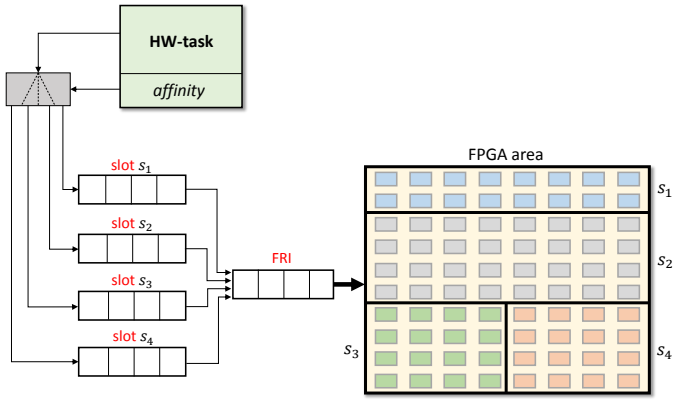

Figure 2. Queuing structure of the FRED framework for scheduling HW-task requests. The FPGA area is divided into four slots s_1, s_2, s_3 and s_4.

B. Communication between SW and HW tasks

Under the FRED framework, SW-tasks and HW-tasks exchange data using a *shared-memory* communication mechanism. In contrast to other approaches that store the shared data into private memory areas within the FPGA slots, the solution adopted in FRED allows decoupling the time a HW-task must hold a slot from the scheduling delays of SW-tasks. Moreover, it simplifies the timing analysis, allowing to upper-bound the suspension delay of a SW-task by the worst-case execution time of the HW-task plus the slot reconfiguration delay. Please refer to [1] for further details.

C. Timing Analysis

This section briefly reviews the timing analysis presented in [1] for preemptable and non-preemptable FPGA reconfiguration. Under the scheduling infrastructure described in Section II-A, the following theorem can be proved (as a special case of Theorem 1 in [1]) to ensure predictable worst-case delays when requesting the execution of a HW-task under preemptive FRI management.

Theorem 1 (From [1]): Consider an arbitrary HW-task request $\mathcal{R}_a$ for τ_a^H issued by a SW-task τ_i. Let $s_k = s(\tau_a^H)$ be the slot to which τ_a^H is allocated. Under *preemptive* FRI management, the maximum delay incurred by $\mathcal{R}_a$ is upper-bounded by

$$\overline{\Delta_a^P} = \sum_{\tau_j \neq \tau_i} \max_{\tau_b^H \in \mathcal{H}(\tau_j)} \left\{ \Delta_b^{slot} + r_b \right\} \tag{1}$$

where

$$\Delta_b^{slot} = \begin{cases} C_b^H & \text{if } s(\tau_b^H) = s_k \\ 0 & \text{otherwise.} \end{cases}$$

The term Δ_b^{slot} represents the interference experienced by τ_a^H due to slot contention originated by the execution of τ_b^H.

Under non-preemptive FRI management, a bound $\overline{\Delta_a^P}$ on the delay experienced by a HW-task is provided by Theorem 2 in [1].

Theorem 2 (From [1]): Consider an arbitrary HW-task request $\mathcal{R}_a$ for τ_a^H issued by a SW-task τ_i. Let $s_k = s(\tau_a^H)$ be the slot to which τ_a^H is allocated. Under *non-preemptive* FRI management, the maximum delay incurred by $\mathcal{R}_a$ is upper-bounded by

$$\overline{\Delta_a^{NP}} = \overline{\Delta_a^P} + NH_k^{max} \times r_k^{max} \tag{2}$$

where

$$NH_k^{max} = \left| \{ \tau_b^H \in \Gamma^H \ : \ s(\tau_b^H) = s_k \} \right|$$

and

$$r_k^{max} = \max_{\tau_b^H \in \Gamma^H} \{ r_b \ : \ s(\tau_b^H) \neq s_k \}.$$

The term NH_k^{max} represents the number of HW-tasks allocated to slot s_k: the FRED scheduling infrastructure ensures that each of them can be *directly* blocked due to non-preemptable FPGA reconfiguration by at most r_k^{max} units of time, representing the largest reconfiguration time of the HW-tasks allocated on the other slots.

III. Problem Definition

A design methodology for supporting real-time applications under the FRED framework must take into account three aspects simultaneously: (i) the *timing constraints* of the application, (ii) the constraints given by the *resource availability* in the FPGA fabric, and (iii) the *geometrical placement*

of the computational activities (synthetized as programmable logic) within the FPGA area. The third aspect originates very challenging problems due the non-uniform distribution of the FPGA resources and some limitations (e.g., rectangular areas only) imposed by commercial FPGA design tools. Given such difficulties, this work focuses on aspects (i) and (ii), thus providing a design solution that ensures the timing constraints of a real-time application while guaranteeing a *necessary* condition for a geometrical partitioning of the FPGA area.

To the best of our knowledge, no methodology has been proposed to date to optimally solve the geometrical placement under realistic assumptions. A notable exception is due to Rabozzi et al. [3], who recently proposed a floorplanning automation based on (i) the enumeration of all possible feasible placements of the reconfigurable regions and (ii) a collection of algorithms to identify a valid placement, which includes custom heuristics, genetic algorithms, and a MILP formulation.

In the present paper, the geometrical placement of the computational activities is left to the system designer, who can use the output of the proposed methodology to guide the actual geometrical partitioning of the FPGA area (usually performed manually using tools like Vivado by Xilinx) or as an input for the approach proposed in [3]. The design of a holistic approach that integrates this work with [3] is left for future work.

The objective of this work can now be precisely stated. It considers a *given* heterogeneous platform and a *given* real-time application, both modeled as reported in Section II. The goal is to identify an FPGA partitioning (if there exists one) such that the considered real-time application is *schedulable* (i.e., meets all its deadlines) by adopting the FRED scheduling infrastructure. More specifically, the identification of an FPGA partitioning consists in determining:

- **(i)** the number n_S of slots and their size in terms of resource units $\mathbf{b}_k$ (for each slot s_k);
- **(ii)** the affinity of each HW-task in the considered real-time application;
- **(iii)** the HW-tasks that can be statically allocated to the FPGA and those that are subject to DPR.

Besides the intrinsic complexity of the problem, which is determined by the underlying similarity with bin-packing, the timing constraints originate additional complications introducing several circular dependencies (e.g., note that reconfiguration times depend on the slot size).

IV. Design Strategy

This section presents a MILP formulation to solve the problem stated in the previous section: the design space is first limited (Sections IV-A and IV-B); then, the variables used to formulate the MILP are presented (Section IV-C); and, finally, the constraints of the MILP formulation are reported, distinguishing them between (i) *structural* (Section IV-D), (ii) *resource* (Section IV-E), and (iii) *timing* (Section IV-F) constraints. Structural constraints are related to the characteristics of the FRED model. Resource constraints describe the availability of the FPGA resources within each slot and across the overall FPGA area. Timing constraints encode the delay bounds guaranteed by the FRED scheduling infrastructure.

The proposed MILP formulation has no objective; that is, *any* solution of the optimization problem (corresponding to a FPGA partitioning) that satisfies the desired constraints is accepted as a *valid* solution. However, the formulation can be extended by including design objectives (e.g., minimizing the FPGA area used by the application, minimizing energy consumption metrics, etc.) by introducing additional modeling layers and matching objective functions.

The formulation is first presented for the case of preemptable FPGA reconfiguration, and the extensions needed to support non-preemptable reconfigurations are discussed in Section IV-G.

A. Slack bounds

To enforce timing constraints, the proposed MILP formulation leverages the knowledge of a *slack bound* $\mathcal{S}_i$ for each SW-task τ_i, i.e., a bound on the maximum time that each SW-task can wait for the completion of *all* its HW-tasks without missing its deadline. The worst-case response time of SW-task τ_i is given by the sum of three contributions: **(i)** the sum of the worst-case execution times (WCETs) of its code segments, **(ii)** the interference caused by high-priority SW-tasks, and **(iii)** the maximum time that it is suspended waiting for the completion of HW-tasks. Once obtained a safe upper-bound $\overline{R}_i$ on contributions (i) and (ii), the slack bound can be easily computed as $\mathcal{S}_i = D_i - \overline{R}_i$. The bound $\overline{R}_i$ can be obtained with the response-time analysis for real-time self-suspending tasks proposed by Nelissen et al. [4]. This analysis uses the response times of the high-priority tasks to derive their interference, which, however, is not known a priori as it depends on contribution (iii) that in turn depends on the FPGA partitioning. Such a circular dependency can be broken by considering the deadlines as upper bounds on the response times of the high-priority tasks: the approach is guaranteed to be safe by the sustainability [5] property of the adopted response-time analysis. If available, e.g., by forcing the allocation of some tasks or in the presence of SW-tasks that do not use HW-tasks, tighter bounds on the response times can be easily adopted at this stage.

B. Maximum Number of Slots

A trivial, but very useful information can be leveraged to construct the MILP formulation: in any possible valid FPGA partitioning, there cannot be more slots than the total number of HW-tasks in the considered real-time application, otherwise, there would be at least one slot that is not used. Hence, an implicit bound on the maximum number of slots can be computed as $N^S_{\text{max}} = \left|\Gamma^H\right|$.

C. Optimization Variables

To encode the problem as a MILP formulation, we introduce a mixed set of *binary* and *real* variables. The number of variables depends on the structure of the application (e.g., number of HW-tasks) and the constant N^S_{max} identified in the previous section. The size of the MILP can be improved by adopting iterative approaches, where N^S_{max} is progressively increased up to its maximum value or varied using a binary search.

For each slot s_k, with $k = 1, \ldots, N^S_{\text{max}}$ and for each FPGA resource-type with index $x = 1, \ldots, NR_x$, we define:

- $b_k^x \in \mathbb{R}_{\geq 0}$, a real variable that specifies the *number* of FPGA resources of type x in the k-th slot.

For each HW-task $\tau_a^H \in \Gamma^H$, for each slot s_k, with $k = 1, \ldots, N_{\text{max}}^S$, we define:

- $A_{a,k} \in \{0, 1\}$, a binary variable such that $A_{a,k} = 1$ iff τ_a^H is *allocated* to slot s_k.

For each HW-task $\tau_a^H \in \Gamma^H$ we define:

- $\gamma_a \in \{0, 1\}$, a binary variable such that $\gamma_a = 1$ iff τ_a^H is subject to DPR (i.e., it is *not* statically allocated to a slot).

For each possible couple of HW-tasks $(\tau_a^H, \tau_b^H) \in \Gamma^H \times \Gamma^H$, we define:

- $\Delta_{a,b}^{\text{slot}} \in \mathbb{R}_{\geq 0}$, a real variable that expresses a bound on the interference suffered by τ_a^H due to slot contention originated by τ_b^H.

For each HW-task $\tau_a^H \in \Gamma^H$, for each SW-task $\tau_i \in \Gamma$, we define

- $\Delta_{a,i} \in \mathbb{R}_{\geq 0}$, a real variable that expresses a bound on the interference suffered by τ_a^H due to HW-tasks used by τ_i.

For each HW-task $\tau_a^H \in \Gamma^H$, we define

- $r_a \in \mathbb{R}_{\geq 0}$, a real variable that expresses the reconfiguration time of τ_a^H.

Note that all such variables are positively-defined. Therefore, any constraint that enforces such variables to be greater than a negative number has no effect. More specifically, let x be one of the variables above and let y be a *negative* term: any constraint of the form $x \geq y$ degenerates to $x \geq 0$, thus imposing no bound on x. This simple observation will result crucial in understanding the following constraints.

D. Structural Constraints

First of all, we impose a basic constraint concerning the affinity of HW-tasks.

***Constraint** 1:* $\forall \tau_a^H \in \Gamma^H, \quad \sum_{k=1}^{N_{\text{max}}^S} A_{a,k} = 1$

Proof: By definition of the model presented in Section II, each HW-task must be allocated to one and only slot. ■

Another constraint is enforced to decide whether a HW-task is subject to DPR.

***Constraint** 2:*

$$\forall \tau_a^H \in \Gamma^H, \forall \tau_b^H \in \Gamma^H \ : \ \tau_a^H \neq \tau_b^H, \ \forall k = 1, \ldots, N_{\text{max}}^S, \\ \gamma_a \geq A_{b,k} - (1 - A_{a,k})$$

Proof: Consider an arbitrary HW-task τ_a^H and an arbitrary slot s_k. If $A_{a,k} = 0$, then τ_a^H is not allocated to slot s_k and the constraint degenerates to $\gamma_a \geq 0$ (independently of the value of $A_{b,k}$) thus enforcing no bound on γ_a. Otherwise, if $A_{a,k} = 1$, then τ_a^H is allocated to slot s_k: in this case, if there exists another HW-task $\tau_b^H \neq \tau_a^H$ that is also allocated to s_k, then $A_{b,k} = 1$ and the constraint becomes $\gamma_a \geq 1$, which correspondingly enforces that τ_a^H is subject to DPR. ■

E. Resource Constraints

We begin from a basic constraint to limit the resources used by all the slots to the total FPGA resources. That is, for each type of resource, any valid FPGA partitioning cannot include more resource units than those available in the platform.

***Constraint** 3:* $\forall x = 1, \ldots, N^{\text{RES}}, \ \sum_{k=1}^{N_{\text{max}}^S} b_k^x \leq B^x$

Proof: It follows from the preceding discussion. ■

As stated in Section II, each slot s_k must be able to host any HW-task with affinity to s_k. The following constraint enforces this requirement.

***Constraint** 4:*

$$\forall x = 1, \ldots, N^{\text{RES}}, \ \forall k = 1, \ldots, N_{\text{max}}^S, \forall \tau_a^H \in \Gamma^H, \\ b_k^x \geq b_a^x \cdot A_{a,k}$$

Proof: Consider an arbitrary slot s_k and an arbitrary HW-task τ_a^H. If $A_{a,k} = 1$, then τ_a^H has affinity to s_k and the constraint becomes $b_k^x \geq b_a^x$, which imposes that the slot must be able to include a sufficient number of resources to host τ_a^H. Otherwise, if $A_{a,k} = 0$, τ_a^H is not allocated to P_k and the constraint degenerates to $b_k^x \geq 0$, thus imposing no constraint on the number of resources included in s_k. ■

F. Timing Constraints

The objective of the following constraints is to encode the delay bound of Theorem 1 as part of the MILP formulation.

The first constraint enforces a bound on the reconfiguration time of HW-tasks, which depends on the size of the slots to which they are allocated to. The constraint makes use of a numerical constant $\mathcal{M}$ to represent infinity, which can be formally defined as $\mathcal{M} = \sum_{x=1}^{N^{\text{RES}}} \{B^x / \rho^x\} + 1$ (i.e., the time needed to reconfigure the entire FPGA area plus one).

***Constraint** 5:* $\forall k = 1, \ldots, N_{\text{max}}^S, \ \forall \tau_a^H \in \Gamma^H,$

$$r_a \geq \sum_{x=1}^{N^{\text{RES}}} \frac{1}{\rho^x} \cdot b_k^x - (1 - A_{a,k}) \cdot \mathcal{M} - (1 - \gamma_a) \cdot \mathcal{M}$$

Proof: Consider an arbitrary HW-task τ_a^H and an arbitrary slot s_k. First of all, note that if τ_a^H is not subject to DPR (i.e., it is statically allocated to one slot), then $\gamma_a = 0$ and the constraint degenerates to $r_a \geq 0$, thus enforcing zero reconfiguration time. Otherwise, τ_a^H incurs in a reconfiguration delay and the term $(1-\gamma_a) \cdot \mathcal{M}$ degenerates to zero. If τ_a^H is not allocated to s_k, then $A_{a,k} = 0$ and the constraint degenerates to $r_a \geq 0$, thus imposing no constraint to the reconfiguration time of τ_a^H. Conversely, if τ_a^H is allocated to P_k, then $A_{a,k} = 1$. In this case, the constraint $r_a \geq \sum_{x=1}^{N^{\text{RES}}} \frac{1}{\rho^x} \cdot b_k^x$ is enforced for each slot of P_k, thus correctly matching the definition of reconfiguration time reported in Section II. ■

The following constraint enforces a bound on the delay caused by the execution of a HW-task τ_b^H to another HW-task τ_a^H when they both contend the same slot by means of DPR.

***Constraint** 6:*

$$\forall k = 1, \ldots, N_{\text{max}}^S, \ \forall \tau_a^H \in \Gamma^H, \ \forall \tau_b^H \in \Gamma^H \ : \ \tau_a^H \neq \tau_b^H, \\ \Delta_{a,b}^{\text{slot}} \geq C_b^H - C_b^H \cdot (1 - A_{a,k}) - C_b^H \cdot (1 - A_{b,k})$$

Proof: Consider two different and arbitrary HW-tasks τ_a^H and τ_b^H. If there exists a slot s_k to which τ_a^H and τ_b^H are allocated to, then $A_{a,k} = A_{b,k} = 1$. In this case, both the terms $-C_b^H \cdot (1 - A_{a,k})$ and $-C_b^H \cdot (1 - A_{b,k})$ degenerates to zero and the constraint $\Delta_{a,b}^{\text{slot}} \geq C_b^H$ is enforced. By looking at the term Δ_b^{slot} in Theorem 1, the latter is a valid bound on the delay caused by τ_b^H to τ_a^H.

If there *not* exists a slot s_k to which τ_a^H and τ_b^H are allocated to, then they cannot interfere each other during their execution. In this case, for each slot s_k, one of the two terms $-C_b^H \cdot (1 - A_{a,k})$ and $-C_b^H \cdot (1 - A_{b,k})$ is equal to $-C_b^H$ and the constraint degenerates to $\Delta_{a,b}^{\text{slot}} \geq 0$ accordingly. ■

With the above constraints in place, it is now possible to impose a bound on the delay suffered by a HW-task. Likewise Constraint 5, the following constraint makes use of a numerical constant $\mathcal{M}$ to represent infinity, which can be formally defined as $\mathcal{M} = \max_{\tau_b^H \in \Gamma^H}\{C_b^H\} + \sum_{x=1}^{N^{\text{RES}}}\{B^x/\rho^x\} + 1$.

***Constraint** 7:*

$$\forall \tau_a^H \in \Gamma^H,\ \forall \tau_i \in \Gamma^S\ :\ \tau_i \neq \tau(\tau_a^H),\ \forall \tau_b^H \in \mathcal{H}(\tau_i),$$
$$\Delta_{a,i} \geq \Delta_{a,b}^{\text{slot}} + r_b - (1 - \gamma_a) \cdot \mathcal{M}$$

Proof: Consider an arbitrary HW-task τ_a^H. If τ_a^H is not subject to DPR, then $\gamma_a = 0$ and the constraint degenerates to $\Delta_{a,i} \geq 0$, thus correctly enforcing no delay. Otherwise, the constraint reduces to $\Delta_{a,i} \geq \Delta_{a,b}^{\text{slot}} + r_b$: following the delay bound provided Theorem 1 (specifically, the contribution related to the maximum operator), the constraint leads to a valid bound on the delay incurred by τ_a^H due to the HW-tasks used by τ_i. ■

Finally, for each SW-task τ_i, we impose that the total delay suffered by its HW-tasks (i.e., those in the set $\mathcal{H}(\tau_i)$) plus their worst-case execution times and reconfiguration times, cannot be larger than the slack bound $\mathcal{S}_i$. This condition is sufficient to ensure the system schedulability.

***Constraint** 8:*

$$\forall \tau_i \in \Gamma^S,\ \sum_{\tau_a^H \in \mathcal{H}(\tau_i)} \left(C_a^H + r_a + \sum_{\tau_j \in \Gamma^S\ :\ \tau_j \neq \tau_i} \Delta_{a,j} \right) \leq \mathcal{S}_i$$

Proof: Follows from the preceding discussion. ■

G. Handling Non-preemptable Reconfiguration

To account for the additional delay introduced by non-preemptable reconfiguration, it is necessary to define further variables with respect to those defined in Section IV-C.

For each possible couple of HW-tasks $(\tau_a^H, \tau_b^H) \in \Gamma^H \times \Gamma^H$, we define:

- $\Delta_{a,b}^{\text{NP}} \in \mathbb{R}_{\geq 0}$, a real variable that expresses the delay directly suffered by τ_b^H due to non-preemptable reconfigurations during a request for τ_a^H.

Such variables are provided to encode the term $NH_k^{max} \times r_k^{max}$ of the delay bound in Theorem 2. This goal is achieved with the following constraint, which makes use of the same numerical constant $\mathcal{M}$ defined for Constraint 5.

***Constraint** 9:*

$$\forall \tau_a^H \in \Gamma^H,\ \forall \tau_b^H \in \Gamma^H,\ \forall \tau_c^H \in \Gamma^H, \forall k = 1, \ldots, N_{\text{max}}^S$$

$$\begin{aligned} \Delta_{a,b}^{\text{NP}} \quad \geq \quad & r_c - (2 - A_{a,k} - A_{b,k}) \cdot \mathcal{M} - A_{c,k} \cdot \mathcal{M} \\ & -(1 - \gamma_a) \cdot \mathcal{M} - (1 - \gamma_c) \cdot \mathcal{M} \end{aligned}$$

Proof: Consider an arbitrary HW-task τ_a^H. First of all, note that if τ_a^H is not subject to DPR, then $\gamma_a = 0$ and the constraint degenerates to $\Delta_{a,b}^{\text{NP}} \geq 0$, thus correctly enforcing no delay. Now, suppose that τ_a^H is subject to DPR ($\gamma_a = 1$). Following Theorem 2, each HW-task τ_b^H allocated to the same slot of τ_a^H (τ_a^H included) contributes to the delay due to non-preemptive reconfiguration when requesting τ_a^H. Also, the contribution of each of such HW-tasks is bounded by the the largest reconfiguration time of HW-tasks τ_c^H that do not have the same affinity of τ_a^H and τ_b^H.

If there exists a slot s_k to which both τ_a^H and τ_b^H are allocated, then $(2 - A_{a,k} - A_{b,k}) = 0$ and the constraint becomes $\Delta_{a,b}^{\text{NP}} \geq r_c - A_{c,k} \cdot \mathcal{M} - (1 - \gamma_c) \cdot \mathcal{M}$. If τ_c^H is allocated to slot s_k ($A_{c,k} = 1$), the constraint enforces no bound, while it becomes $\Delta_{a,b}^{\text{NP}} \geq r_c - (1 - \gamma_c) \cdot \mathcal{M}$ if τ_c^H is allocated to a slot different than s_k. If τ_c^H is subject to DPR ($\gamma_c = 1$), hence it can generate reconfiguration delay and the constraint correctly enforces $\Delta_{a,b}^{\text{NP}} \geq r_c$.

Finally, if the slot s_k does not exist, then the term $(2 - A_{a,k} - A_{b,k})$ is always positive and the constraint correspondingly degenerates to $\Delta_{a,b}^{\text{NP}} \geq 0$. ■

***Constraint** 10 (In place of Constraint 8):*

$$\forall \tau_i \in \Gamma^S,$$
$$\sum_{\tau_a^H \in \mathcal{H}(\tau_i)} \left(C_a^H + r_a + \sum_{\substack{\tau_j \in \Gamma^S \\ \tau_j \neq \tau_i}} \Delta_{a,j} + \sum_{\tau_b^H \in \Gamma^H} \Delta_{a,b}^{\text{NP}} \right) \leq \mathcal{S}_i$$

Proof: The constraint is analogous to Constraint 8. The delay bound follows from Theorem 2 given that $\sum_{\tau_b^H \in \Gamma^H} \Delta_{a,b}^{\text{NP}}$ is a bound on the delay due to non-preemptive reconfiguration when requesting τ_a^H. ■

The above constraints can be further improved by exploiting the serialization of the requests for HW-tasks issued by the same SW-task: please refer to [1] (Lemma 2) for further details.

V. Experimental results

This section reports on the results of an experimental study that has been conducted to assess the performance and the scalability of the proposed design methodology. The experimental study considered the Xilinx Zynq 7020 as a reference platform. The FPGA fabric available in such a platform includes *four* resources: 53200 *look-up tables* (LUTs), 106400 *flip-flops* (FFs), 140 *block RAMs* (BRAMs), and 220 *digital signal processor* slices (DSPs) (see [6], page 3). The proposed design strategy has been evaluated under synthetic workload, which has been generated as described below.

A. Workload generation

A set of n SW-tasks has been generated, each using a corresponding HW-task. The n HW-tasks have been configured for requiring a given amount of FPGA resources, obtained as $\lfloor U \cdot B^x \rfloor$ (for each resource $x = 1, \ldots, 4$). The term $U \in \mathbb{R}_{\geq 1}$ is a parameter that controls the percentage of additional resources that are needed by the HW-tasks with respect to the ones that are *statically* available in the platform. Note that $U > 1$ in order to generate scenarios that require DPR to ensure the feasibility of the application. For each resource, the available units have been randomly distributed among the HW-tasks following a uniform distribution using the UUnifast algorithm [7]). The number of resources used by each HW-task has been limited to be in the range $[B^x_{\text{min}}, \lfloor u_{\text{max}} \cdot B^x \rfloor]$. The parameters controlling this range have been set to $B^1_{\text{min}} = B^2_{\text{min}} = 10$ (LUTs and FFs), $B^3_{\text{min}} = B^4_{\text{min}} = 0$ (BRAMs and DSPs), and $u^{\text{max}} = 0.7$. The WCETs of the HW-tasks have been randomly generated in the range $[5, 500]$ ms with uniform distribution. The slack bounds $\mathcal{S}_i$ have been randomly generated with uniform distribution in the range $[C^H_i + \alpha \cdot \Delta^{\text{MAX}}_i, C^H_i + \Delta^{\text{MAX}}_i]$, where Δ^{MAX}_i is the *maximum* delay that can τ^H_i can incur (with respect to a trivial solution of the partitioning problem) and $\alpha \in [0, 1]$ is a parameter that controls the tightness of the timing requirements. The term Δ^{MAX}_i can be computed by applying Theorem 1 to the case in which *all* the HW-tasks are allocated to the same slot. Higher values of α correspond to scenarios with relaxed timing constraints, while lower values correspond to scenarios that have tight timing constraints, which hence determine more difficult instances for the design methodology.

B. Experiments

The design strategy has been evaluated with a multidimensional exploration of the parameters that control the workload generation. The parameter α (controlling the slacks) has been varied from 0.1 to 1.0 with step 0.1. The number n of tasks has been varied from 3 to 20 with step 1. The parameter U has been set to 2. For each combination of such parameters, 500 problem instances have been generated and each of them has been tested under both preemptable and non-preemptable reconfiguration, for a total of 180,000 instances. The experiments have been performed on a machine equipped with 24 cores Intel Xeon E3-12 @ 2.7 GHz and 16 GB of RAM. The GUROBI solver has been used to solve the MILP formulations.

Figure 3 reports the maximum and the average running times needed to solve the MILP formulations, for three representative configurations ($\alpha \in \{0.2, 0.3, 0.5\}$), as a function of the number n of tasks. As it can be noted from the graphs, the running times decrease as α increases, which is compatible with the rationale that under relaxed timing constraints the problem is easier to be solved. For $\alpha = 0.2$, the maximum running time under non-preemptive reconfiguration achieves large values for $n \in \{14, 15, 16\}$ tasks, while the average running times are in the order of one minute. Scenarios with non-preemptive reconfiguration tend to require larger solving times due to the additional variables introduced in Section IV-G. Note that the results are not monotonic with n. For a small number of tasks, the size (in terms of variables) of the MILP formulation is small, leading to short running times. When the number of tasks is high, since the total demand of FPGA area is fixed, the area consumption of each HW-task tends to decrease, so facilitating their partitioning. Overall, the results show that the proposed approach is affordable in most of the tested cases, resulting compatible with the time-frame of design activities.

Figure 4 reports the ratio of successfully solved instances, under other three representative configurations, as a function of the parameter α. As expected, the success ratio increases as α increases (corresponding to scenarios with more relaxed timing constraints). Being the overall area consumption of the HW-tasks fixed ($U = 2$), instances with a few tasks resulted difficult to partition; this is also due to a disruptive contention for the limited number of available slots (with very large area). For larger number of tasks ($n > 12$), the approach is able to find a valid partitioning for more than 50% of the tested instances, even in the presence of extremely stringent timing constraints ($\alpha = 0.1$). As also observed in [1], the difference between preemptable and non-preemptable reconfiguration is limited, since the contention for the slots tends to be the dominant contribution to the overall delay.

VI. Related Work

The problem of partitioning the FPGA area has been extensively studied from different perspectives and under multiple assumptions. A detailed discussion of all the works proposed in the literature is too vast to fit in the space available in this paper, therefore, the discussion is limited to the works that are more closely related to the present paper.

In the context of real-time systems, Di Natale and Bini [8] proposed an optimization method to partition the area of an FPGA into slots to be allocated to HW-tasks and softcores to execute SW-tasks. Given the high computational complexity of the method, their approach can only be used off-line to obtain a *static task allocation*. Furthermore, their approach did not consider dynamic partial reconfiguration, nor synchronization issues between software and hardware tasks, and it was limited to homogeneous FPGA areas. One of the first attempts to support DPR-enabled FPGAs in the context of real-time systems is due to Pellizzoni, and Caccamo [9]. Their work includes an allocation scheme and an admission test to provide real-time guarantees for applications that consists of both SW- and HW-tasks. However, reconfiguration is performed only when the task set is modified, (i.e., in the presence of *mode-changes*) and not for virtualizing the FPGA area, as done in the FRED framework. In addition, their analysis does not consider any delay introduced by the reconfiguration interface and assumes a homogeneous type of FPGA resource.

Outside the context of real-time systems, several approaches have been proposed for allocating HW-tasks within an FPGA area, both considering static and dynamic allocation by means of DPR. The work more related to the present paper is the one by Rabozzi et al. [3] (already discussed in Section III) and Beckhoff et al. [10], which proposed a technique based on optimization methods to perform the floorplanning of reconfigurable areas starting from an initial solution provided by tools distributed by Xilinx. A thorough state-of-the-art analysis of this research area is available in [3]. Deiana et al. [11] proposed a MILP formulation for computing a *static* schedule for a set of computational activities with precedence constraints. Their work is mostly focused on average-case performance and does not cope with recurrent activities in multitasking. Later, Purgato et al. [12] proposed a

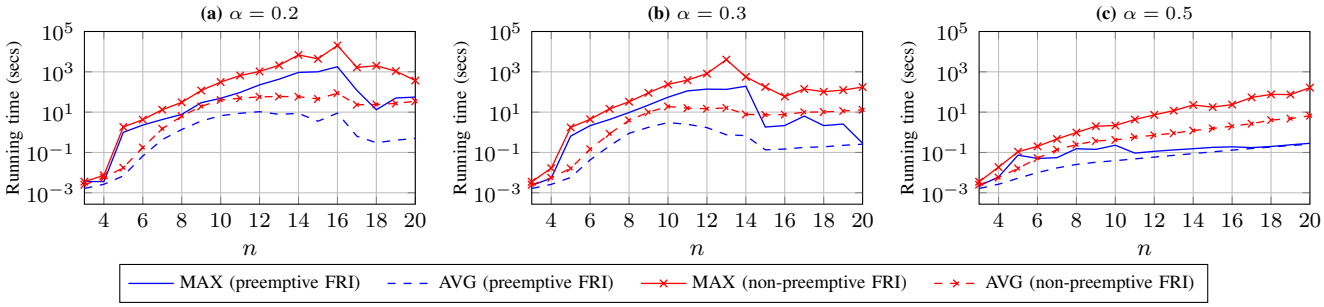

Figure 3. Running times needed to solve the MILP formulation for three representative configurations as a function of the number n of tasks.

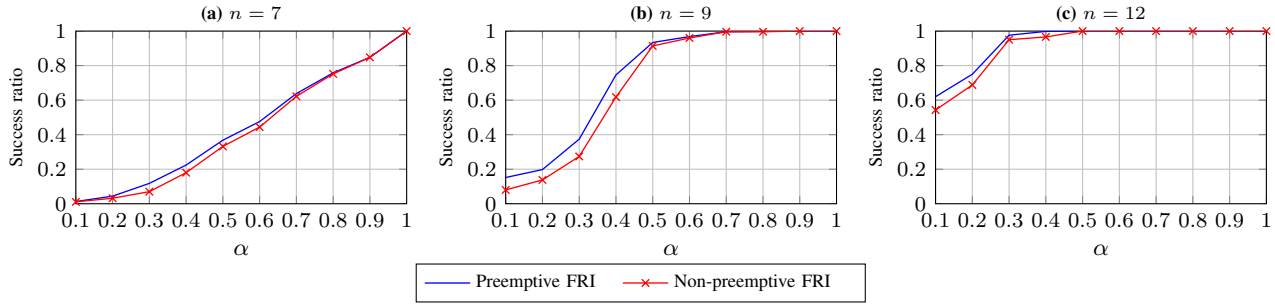

Figure 4. Ratio of successfully solved instances under three representative configurations.

set of heuristics that improved the approach of [11] in terms of running time and maximum makespan of the static schedule.

Other authors addressed the problem of supporting DPR-enabled in real-time operating systems. Iturbe et al. [13] implemented R3TOS, an operating system that allows a fully-dynamic HW-tasks allocation, which is achieved by avoiding the use of slots and static communication channels. This flexibility comes at the cost of using the reconfiguration interface for implementing communication channels between tasks. Their work is not supported by a worst-case timing analysis and only considers HW-tasks. Lubbers and Platzner [14] developed ReconOS, an operating system that extends the multi-thread paradigm to HW-tasks. ReconOS has been extended to support reconfigurable FPGAs by using cooperative multitasking to handle the contention of predetermined FPGA slots [15]. To the best of our records, no solutions have been proposed for partitioning the FPGA area under ReconOS while matching real-time constraints.

VII. Conclusion and Future Work

This paper presented a design methodology for partitioning the FPGA area under FRED, a programming framework that has been proposed to support the development of real-time applications upon DPR-enabled heterogeneous system-on-chips. The design is accomplished by means of a mixed-integer linear program that is in charge of (i) sizing a set of slots realized into the FPGA area, (ii) allocating hardware tasks to such slots, and (iii) selecting which hardware tasks can be statically allocated to the FPGA. All such objectives are reached while ensuring predictable worst-case response times of the tasks. The running time of the proposed technique has been shown to be compatible with the time-frame of off-line design activities. Future work includes the development of a holistic methodology that accounts for the geometrical placement of the FPGA slots and the inclusion of the synthesis of configurations for communication infrastructures.

References

[1] A. Biondi, A. Balsini, M. Pagani, E. Rossi, M. Marinoni, and G. Buttazzo, "A framework for supporting real-time applications on dynamic reconfigurable FPGAs," in *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, December 2016.

[2] *Vivado Design Suite User Guide: Partial Reconfiguration*, Xilinx, November 2015, v2015.4.

[3] M. Rabozzi, G. C. Durelli, A. Miele, J. Lillis, and M. D. Santanbrogio, "Floorplanning automation for partial-reconfigurable FPGAs via feasible placements generation," *IEEE Transactions on Very large Scale Integration (VLSI) Systems*, vol. 25, no. 1, January 2017.

[4] G. Nelissen, J. Fonseca, G. Raravi, and V. Nelis, "Timing analysis of fixed priority self-suspending sporadic tasks," in *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, July 2015.

[5] A. Burns and S. Baruah, "Sustainability in real-time scheduling," *Journal of Computing Science and Engineering*, vol. 2, no. 1, pp. 74–97, 2008.

[6] Xilinx zynq-7000 all programmable SoC overview. [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf

[7] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1-2, pp. 129–154, 2005.

[8] M. D. Natale and E. Bini, "Optimizing the FPGA implementation of HRT systems," in *Proceedings of the 13th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS)*, April 2007.

[9] R. Pellizzoni and M. Caccamo, "Real-time management of hardware and software tasks for FPGA-based embedded systems," *IEEE Transactions on Computers*, vol. 56, no. 12, pp. 1666–1680, December 2007.

[10] C. Beckhoff, D. Koch, and J. Torres, "Automatic floorplanning and interface synthesis of island style reconfigurable systems with goahead," in *Proc. of the 26th International Conference on Architecture of Computing Systems (ARCS)*, June 2013.

[11] E. A. Deiana, M. Rabozzi, R. Cattaneo, and M. D. Santambrogio, "A multiobjective reconfiguration-aware scheduler for FPGA-based heterogeneous architectures," in *Proc. of the International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, December 2015.

[12] A. Purgato, D. Tantillo, M. Rabozzi, D. Sciuto, and M. D. Santambrogio, "Resource-efficient scheduling for partially-reconfigurable FPGA-based systems," in *Proc. of the IEEE International Parallel and Distributed Processing Symposium Workshops*, May 2016.

[13] X. Iturbe, K. Benkrid, C. Hong, A. Ebrahim, R. Torrego, and T. Arslan, "Microkernel architecture and hardware abstraction layer of a reliable reconfigurable real-time operating system (R3TOS)," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 8, no. 1, pp. 5:1–5:35, March 2015.

[14] E. Lübbers and M. Platzner, "ReconOS: Multithreaded programming for reconfigurable computers," *ACM Transactions on Embedded Computing Systems*, vol. 9, no. 1, pp. 8:1–8:33, October 2009.

[15] ——, "Cooperative multithreading in dynamically reconfigurable systems," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, August 2009.