Work-In-Progress: Cyber-Physical Systems and Dynamic Partial Reconfiguration Scalability: opportunities and challenges

Gabriella D'Andrea
DISIM Department
Università degli Studi dell'Aquila (UNIVAQ)
L'Aquila, Italy
gabriella.dandrea@graduate.univaq.it

Giacomo Valente
DISIM Department
Università degli Studi dell'Aquila (UNIVAQ)
L'Aquila, Italy
giacomo.valente@univaq.it

***Abstract*—In the domain of Cyber-Physical Systems, the FPGA Dynamic Partial Reconfiguration (DPR) feature has been proved to be efficient to adapt the system hardware configuration to environment changes, also in the case of hard real-time constraints. Often, in this context, a single task can request multiple DPR to modify the platform configuration.**

However, also if in the hard real-time literature several works exploit the DPR process, to the best of our knowledge, no one deals with its scalability (i.e., the efficient management of multiple DPR requests).

Hence, in this work, focusing on hard real-time systems, we conduct both a theoretical and practical investigation about the DPR scalability, discussing the obtained outcomes and the objectives to be achieved in the near future.

***Index Terms*—Cyber-Physical Systems, Dynamic Partial Reconfiguration, FPGA, Hard Real-Time Systems.**

I. Introduction

In the Cyber-Physical Systems (CPSs) domain, embedded computers and networks monitor and control physical processes, usually with feedback loops where computation affects physical processes and vice versa [1]. Especially when safety is a concern, computation is often subject to hard real-time constraints. Moreover, such kind of systems should adapt themselves to possible changes in the environment. In such a context, a platform that offers the possibility to modify its configuration (e.g., a reconfigurable platform) at run-time is essential. Nowadays, a valid solution is provided by Field Programmable Gate Arrays (FPGAs) with Dynamic Partial Reconfiguration (DPR) features [2] [3]. The DPR, indeed, allows to modify at run-time the HW configuration adapting the CPSs architecture to the physical world changes in order to continue meeting the systems requirements.

In general, when a DPR is required (typically by a SW-task [4]), the reconfiguration occurs by moving the reconfiguration file of the task to be implemented in HW (HW-task) from a generic memory to an FPGA reconfiguration memory, along with an FPGA reconfiguration path, thanks to a reconfiguration controller and an FPGA Reconfiguration Interface (FRI) [4]. However, the variation that the DPR acts in the FPGA platforms can produce some kind of impact also on the system parts not involved in the HW modification [5]. This issue has been faced over the years by reducing the granularity of the HW-task reconfiguration file [6] (e.g., from Xilinx Virtex 2 [7] to Virtex 7 [8], and from Intel Stratix V [9] to Stratix 10 [10]). Besides, the satisfaction of different requirements often leads to the needs of multiple related DPR (see e.g., [5] and [11]). Consequently, both decreasing the reconfiguration file granularity and increasing the DPR frequency lead to the need for reconfigurable platforms able to efficiently handle multiple reconfiguration requests (*DPR scalability*, hereinafter).

Provider	Controller	DMA	Scalability
I	Xilinx PCAP-based - Zynq-7000 [12]	YES	N.D.
	Xilinx PCAP-based - Zynq-Ultrascale [13]	YES	N.D.
	Xilinx MCAP-based [13]	YES	YES
	Xilinx ICAP-based [14] [15]	NO	NO
	Intel PR [16]	NO	NO
A	Vipin et.al [6]	YES	NO
	Liu et.al [17]	YES	NO
	Claus et.al [18]	YES	NO
	Damschen et.al [19]	YES	NO
	Pezzarossa et.al [20]	NO	NO
	Xiao et.al [21]	NO	NO
	Hübner et.al [22]	NO	NO

TABLE I
RECONFIGURATION CONTROLLERS AVAILABLE AT INDUSTRY (I) AND ACADEMIC (A). THE THIRD COLUMN INDICATES WHETHER THEY USE OR NOT A DMA, WHILE THE FOURTH COLUMN REPORTS THE FLAG YES/NO/ND TO INDICATE IF A CONTROLLER SUPPORTS/DOES NOT SUPPORT THE DPR SCALABILITY OR IF THERE ARE NO INFORMATION ON EXISTING DOCUMENTATION.

As reported in Table I, the Industry introduced new architectural elements inside the FPGAs to reduce the overhead associated with the whole DPR process management, (like Direct Memory Access (DMA) controller in Xilinx Zynq-7000 [12]), while the Academic proposed custom reconfiguration controllers (once more based on DMA and able to support the DPR scalability) to further improve the efficiency of industrial solutions [23] [24] [11]. However, even though the DPR scalability can introduce several benefits within the systems that adopt it, such a DPR aspect is not properly taken into account by the hard real-time analysis approaches existing in the literature (see e.g., [4], [25], [26], and [27]). These works do not consider the fact that existing reconfigurable platforms embedding an FPGA support the DPR scalability by allowing multiple DPR requests per time. They leave the constraint that each SW-task can require at most one DPR per time even if a HW-task needs multiple DPR requests to be implemented. Moreover, with this constraint is also overlooked the possibility of reducing the time that a SW-task has to wait before the required reconfigurations are well established.

Therefore, in this work, focusing on hard real-time systems, we

conduct both a theoretical and practical investigation on the DPR scalability reported in Section II. In more detail, in Section II.A, we report the theoretical investigation conducted using the only framework [4], presented in hard real-time literature, that does not explicitly exclude the DPR scalability adoption; in Section II.B, we report: (i) a practical investigation conducted to identify the existing DPR-compliant platforms that could support the DPR scalability, (ii) and a practical DPR-scalability experimentation carried out on one of these platforms. Each subsection reports the obtained outcomes, leaving the presentation of future objectives to Section III.

II. Investigation outcomes

A. Theoretical investigation

To theoretically investigate the DPR scalability we have chosen the FRED framework proposed in [4] since it allows a single SW-task to make multiple DPR requests when it has to execute a single HW-task. It is worth noting that the generic HW-task can be composed of multiple HW fragments here referred as *HW sub-tasks*. In particular, we test the DPR scalability in the FRED framework selecting an application constituted of a number of SW-tasks, where one of them requires multiple DPR to execute its HW-task; in this test, we evaluate if for the considered SW-tasks the delay bound is guaranteed even with a HW-task that needs multiple DPR requests to be executed. Before showing the test results, we report the concepts presented in [4] refining some of them to properly consider the fact that a single HW-task may need the satisfaction of multiple DPR requests before the execution start. In FRED a HW-task is a task running on FPGA, and the activities executed on the FPGA are modeled as a set Γ^H of n_H HW-tasks. In turn, for each HW-task $\tau_a^H \in \Gamma^H$, we define a set $\Gamma_a^{HS} = \{\tau_{a,1}^{HS}, ..., \tau_{a,n_{D_a}}^{HS}\}$ of *HW sub-tasks* that compose τ_a^H, where n_{D_a} is defined as the number of HW sub-tasks required to execute the HW-task τ_a^H. Using the same architecture of FRED, a HW-task τ_a^H can be executed only if all its HW sub-tasks have been programmed on FPGA slots: in particular, each HW sub-task $\tau_{a,j}^{HS}$, with $j = 1, ..., n_{D_a}$, needs to be programmed into a slot, since each slot can accommodate at most one HW sub-task. Each HW sub-task $\tau_{a,j}^{HS}$ requires $b_{a,j}$ logic blocks: to program $\tau_{a,j}^{HS}$ into a slot, the FRI has to program all its logic blocks, independently on the number $b_{a,j}$. Each $\tau_{a,j}^{HS}$ can be programmed in any of the slots belonging to an FPGA partition: the partition hosting it is denoted as $P(\tau_{a,j}^{HS}) = P_k$ and referred as *affinity*. For all $\tau_{a,j}^{HS}$, with $j = 1, ..., n_{D_a}$ and with $P(\tau_{a,j}^{HS}) = P_k$, it must be that $b_{a,j} \leq b_K^S$. The time $r_{a,j}^{HS}$ is the time needed to program the HW sub-task $\tau_{a,j}^{HS}$: the time r_a^H to program a generic HW-task is then given by $\sum_{j=1}^{n_{D_a}} r_{a,j}^{HS}$. When a generic SW-task τ_i requires the execution of a HW-task τ_a^H that needs multiple HW sub-tasks in $\Gamma_a^{HS} = \{\tau_{a,1}^{HS}, ..., \tau_{a,n_{D_a}}^{HS}\}$ to be executed, in this work we refer to that as $\tau_{i,a}^H$ and to the set of associated HW sub-tasks as $\Gamma_{i,a}^{HS} = \{\tau_{i,a,1}^{HS}, ..., \tau_{i,a,n_{D_a}}^{HS}\}$. The SW-task performs the request by means of a blocking system call named $EXECUTE\_HW\_TASK(\tau_{i,a,j}^{HS})$, with $j = 1, ..., n_{D_a}$ [4]. The FPGA is again modeled as done in [4] and reported above. However, we consider that a busy slot in addition to being *reserved* or *active* can also be *contributing*; a busy contributing slot is an FPGA slot that has been programmed with a HW sub-task but cannot start the execution until all the others HW sub-tasks have been programmed in other slots.

At this point, we can report the result of the proposed test on FRED. We select an application (app_{test}, hereinafter) that consists of three SW-tasks: $\tau_1 = \langle \tau_{1,1}, \tau_{1,a}^H, \tau_{1,2} \rangle$, $\tau_2 = \langle \tau_{2,1}, \tau_{2,b}^H, \tau_{2,2} \rangle$, and $\tau_3 = \langle \tau_{3,1}, \tau_{3,c}^H, \tau_{3,2} \rangle$. The priority assignment is such that $\pi_1 > \pi_2 > \pi_3$. The HW-task $\tau_{1,a}^H$ requires $n_{D_{1,a}} = 4$ HW sub-tasks to be executed, i.e., $\Gamma_{1,a}^{HS} = \{\tau_{1,a,1}^{HS}, \tau_{1,a,2}^{HS}, \tau_{1,a,3}^{HS}, \tau_{1,a,4}^{HS}\}$. The HW-tasks $\tau_{2,b}^H$ and $\tau_{3,c}^H$, on the other hand, have one HW sub-task, i.e., $n_{D_{2,b}} = n_{D_{3,c}} = 1$ (therefore, we will refer to them with the HW-task notation). The FPGA module contains four partitions P_1, P_2, P_3, and P_4, each consisting of a single slot. The affinity is assigned as follows: $P(\tau_{1,a,1}^{HS}) = P_1$, $P(\tau_{1,a,2}^{HS}) = P(\tau_{2,b}^H) = P_2$, $P(\tau_{1,a,3}^{HS}) = P(\tau_{3,c}^H) = P_3$, and $P(\tau_{1,a,4}^{HS}) = P_4$. Fig. 1 shows the timing diagrams of the execution of app_{test} scheduled with FRED in non-preemptive way. It is worth noting that, for the τ_1 SW-task, we assume that when executing the $EXECUTE\_HW\_TASK(\tau_{1,a,j}^{HS})$ system call, with $j = 1, ..., n_{D_{1,a}}$, differently from FRED, it self-suspends only until the end of reconfiguration of the HW sub-task $\tau_{1,a,j}^{HS}$. τ_1 SW-task has to wait only the reconfiguration of $\tau_{1,a,j}^{HS}$ [4] since it cannot complete its execution until all the n_{D_a} HW sub-tasks have been programmed. Without this modification, τ_1 would remain self-suspended and never released (deadlock). The test result highlights that FRED treats a HW-task that requires multiple HW sub-tasks to be executed, as multiple requests to be enqueued. Indeed, being n_{DPR} the number of DPR requested, FRED performs n_{DPR} different HW-task requests. As shown in Fig. 1 introducing multiple DPR requests in the FRED framework, τ_1 is delayed in its execution more than τ_2 and τ_3; this is quantified by the worst-case bound computation provided in [4]. For τ_1, τ_2 and τ_3 the worst-case bounds can be computed as follows:

$$\overline{\Delta_{1,a}^P} = \overline{\Delta_{1,a,1}^P} + \overline{\Delta_{1,a,2}^P} + \overline{\Delta_{1,a,3}^P} + \overline{\Delta_{1,a,4}^P} = 22$$
$$\overline{\Delta_{2,b}^P} = 14 \quad \overline{\Delta_{3,c}^P} = 14 \tag{1}$$

With this approach, we have that $\tau_{1,a}^H$ requests can be delayed by the execution of HW-tasks required by other SW-tasks interference caused by τ_2 and τ_3 in the example), suffering a worst-case delay that is much greater than other HW-tasks requests. Moreover, using priority scheduling, as done in the example above, can conduce to a delay for the highest priority task.

These results confirm that FRED supports multiple DPR requests but, at the same time, that some revisions have to be made to adequately consider the efficiency that can be introduced by exploiting the DPR scalability provided by modern reconfigurable platforms.

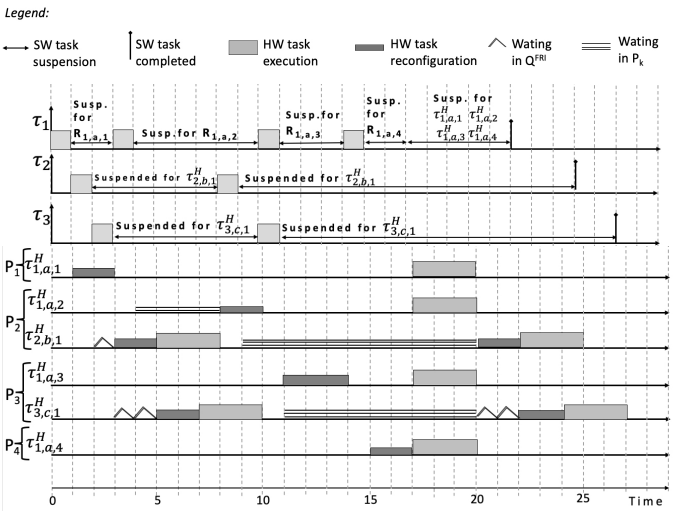

Fig. 1. Timing diagrams of the execution of app_{TEST} on FRED. The x-axis reports the temporal quantum [4], while the y-axis reports the SW-tasks and the HW-tasks with related partitions. The first three x-axes from the top represent the SW-task behavior; each SW-task self-suspends after requiring the HW-task reconfiguration (indicated with the letter R and the HW-task name as subscript). The last six X-axes report the evolution of the states of each HW-task in each partition P_k (e.g., τ_1, after requiring the reconfiguration of $\tau_{1,a,1}^H$ in P_1, self-suspends. Then τ_2 requires the $\tau_{2,b}^H = \tau_{2,b,1}^{HS}$ reconfiguration in P_2; in turn $\tau_{2,b}^H = \tau_{2,b,1}^{HS}$ waits in Q^{FRI} until $\tau_{1,a,1}^{HS}$ completes its reconfiguration and so forth.)

B. Practical investigation

To practically validate the DPR scalability, reconfigurable platforms with a reconfiguration controller capable of managing multiple DPR requests per single HW-task would be needed. However, since no works in literature dealt with the DPR scalability, we have conducted an in-depth analysis of the Academic and Industrial reconfiguration controllers, in order to identify a platform that supports the DPR scalability and that it is suitable for edge-computing applications; in particular, hard real-time edge-computing applications constitute the main target of this work. From Table I, which reports the results of such analysis, we notice that the only controller whose documentation asserts the DPR scalability support is the Xilinx MCAP-based [13]. In such a controller the DPR is based on the PCI Express bus [28] that, however, is not present in DPR-compliant platforms for edge-computing applications with limited HW resources. Conversely, the Xilinx PCAP-based controllers are used in the platforms (e.g., Xilinx Zynq7000 SoC [12] and the Xilinx Zynq Ultrascale+ SoC [13]) involved in the development of edge-computing applications. Moreover, the documents related to the Xilinx PCAP-based controllers [13] [12] do not deny the ability of these controllers to manage multiple DPR requests per single HW-task (i.e., DPR scalability support). Hence, to investigate the DPR scalability in platforms that are also suitable to develop edge-computing applications we analyze both the PCAP controller involved in the Xilinx Zynq Ultrascale+ SoC and Xilinx Zynq7000 SoC. The Zynq7000 has the DevC controller, while the Zynq Ultrascale+ has the Configuration and Security Unit (CSU). Since these two controllers have a similar structure in the part that deals with the DPR in this work we focus only on the DevC of Xilinx Zynq7000 SoC.

We executed on this SoC a simple bare-metal application that allows selecting between two mathematical operations (i.e., addition and multiplication) to be implemented in HW by exploiting the DPR [29] with the related reconfiguration files (called partial-bitstreams *pBS*). The pBS called add.bin is used to implement in HW the addition, while the pBS called mult.bin is used to implement in HW the multiplication. For the sake of simplicity, we have chosen two operations independent of each other whose *pBS* preparation procedure is well explained in the Xilinx Partial Reconfiguration User Guide [29]. To test the DPR scalability, we require one DPR associating to it more than one *pBS*s with the different destination address of the FPGA reconfiguration memory (e.g., we require one DPR associating to it the *pBS* of the adder and the *pBS* of multiplier).

To verify the correctness of the DPR: (i) we measured, through a custom HW monitoring system [30]–[32], the time spent to fulfill all the DPR requested (called reconfiguration time (RT) hereinafter) and we compared this time with the RT calculated in advance with the model proposed in [33]; (ii) we compared the outputs obtained by the reconfigured applications with the expected ones.

Based on this scenario, the tests are carried out requiring to the application to launch (i) the DPR with only add.bin, (ii) the DPR with two pBSs (add.bin and mult.bin), (iii) the DPR with three pBSs (add.bin, mult.bin, add.bin), and (iv) the DPR with four pBSs (add.bin, mult.bin, add.bin, and mult.bin). In these tests, each DPR request can be considered well established only when is acquired the interrupt used to notify that all pBSs involved in the request are properly placed inside the FPGA reconfiguration memory. The obtained results are shown in the last three columns of Table II. In particular, we observed that the application outputs are correct in the case of 1 and 2 DPR requests; we also noted that in these cases the expected and the detected RTs are quite similar in the trend. Conversely, in the cases of 3 and 4 DPR requests, the application outputs are not correct, and the detected RTs are not only far to be the expected ones but are also mistakenly equal to the RT detected in the case where 2 DPR are requested.

These results highlight that in the Xilinx Zynq7000 SoC a DPR scalability of more than 2 DPR requests cannot be efficiently managed.

pBS Number	pBS Name	Total pBS size (KB)	Application Output	Measured RT (ms)	Expected RT (ms)
1	add.bin	110.984	✓	0.69469	0.85341
2	add.bin mult.bin	221.968	✓	1.38938	1.70682
3	add.bin mult.bin add.bin	332.952	✗	1.38938	2.56023
4	add.bin mult.bin add.bin mult.bin	442.952	✗	1.38938	3.41364

TABLE II

DPR SCALABILITY EXPERIMENTATION RESULTS: THE FIRST, SECOND, AND THIRD COLUMNS REPORT THE NUMBER OF *pBS*S INVOLVED, THEIR NAME, AND THEIR SIZE, RESPECTIVELY. THE *Application output* COLUMN REPORTS IF THE OUTPUT OF THE EXECUTED APPLICATION IS CORRECT OR NOT, USING THE SYMBOLS ✓INSTEAD OF YES AND ✗INSTEAD OF NO, RESPECTIVELY. THE *Measured RT* COLUMN REPORTS THE MEASURED RECONFIGURATION TIME DURING EACH DPR, WHILE THE *Expected RT* COLUMN REPORTS THE EXPECTED RECONFIGURATION TIME COMPUTED WITH THE EXPRESSION IN [33].

Therefore, an implementation that takes into account that limitation could be useful to fully exploit the DPR scalability potential.

III. Future Investigations

In this work, we dealt with the DPR scalability investigation, both theoretically and practically. In a theoretical investigation, we focus on the hard real-time systems conducting scalability experimentation on an existing approach [4], highlighting its weaknesses in managing multiple DPR requests. In the practical investigation, using a reconfigurable platform that supports the DPR scalability and is suitable for the development of edge-computing applications, we have found a limit of two concurrently manageable DPR requests.

We are now investigating the DPR scalability pointing out its opportunities and challenges. In particular, we are working on: (i) a revision of FRED [4] to verify if it is possible to adequately consider the DPR scalability feature, (ii) an implementation to overcome the discovered DPR scalability limit enabling the possibility to fully exploiting the benefits introduced by the usage of multiple DPR requests.

As future activities, we proceed on the DPR scalability investigation with more complex applications on other DPR compliant platforms (both Xilinx and Intel), also improving other existing hard real-time analysis frameworks in order to efficiently consider the DPR scalability.

IV. Acknowledgment

This work has been partially supported by the ECSEL RIA 2017 FITOPTIVIS project.

References

[1] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems, A Cyber-Physical Systems Approach*, 2nd ed. MIT Press, 2017, *ISBN* 978-0-262-53381-2.

[2] G. D′Andrea, T. D. Mascio, and G. Valente, "Self-adaptive loop for CPSs: is the dynamic partial reconfiguration profitable?" in *8th Mediterranean Conference on Embedded Computing (MECO)*, June 2019, pp. 1–5.

[3] A. Rodriguez, J. Valverde, J. Portilla, A. Otero, T. Riesgo, and E. de la Torre, "Fpga-based high-performance embedded systems for adaptive edge computing in cyber-physical systems: The artico3 framework," *Sensors*, vol. 18, p. 1877, 06 2018.

[4] 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 *IEEE Real-Time Systems Symposium (RTSS)*, Nov. 2016, pp. 1–12.

[5] M. Gokhale, P. Graham, E. Johnson, N. Rollins, and M. Wirthlin, "Dynamic reconfiguration for management of radiation-induced faults in fpgas," in *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.*, 2004, pp. 145–.

[6] K. Vipin and S. A. Fahmy, "Fpga dynamic and partial reconfiguration: A survey of architectures, methods, and applications," *ACM Comput. Surv.*, vol. 51, no. 4, Jul. 2018. [Online]. Available: https://doi.org/10.1145/3193827

[7] Xilinx, *Virtex-II Platform FPGAs: Complete Data Sheet*, 2014, Accessed: Apr. 16,2020. [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds031.pdf

[8] ——, *Virtex-7 T and XT FPGAs (DS183)*, 2019, Accessed: Apr. 16,2020. [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds183_Virtex_7_Data_Sheet.pdf

[9] Intel, *Stratix V Device Datasheet*, 2019, Accessed: Apr. 16,2020. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/stratix-v/stx5_53001.pdf

[10] ——, *Intel Stratix 10 Device Datasheet*, 2020, Accessed: Apr. 16,2020. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/stratix-10/s10_datasheet.pdf

[11] K. Vipin and S. A. Fahmy, "Zycap: Efficient partial reconfiguration management on the xilinx zynq," *IEEE Embedded Systems Letters*, vol. 6, no. 3, pp. 41–44, 2014.

[12] Xilinx, *Zynq-7000 SoC Technical Reference Manual (UG585)*, 2018, Accessed: Oct. 3,2019. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf

[13] ——, *UltraScale Architecture Configuration (UG570)*, 2020, Accessed: Apr. 16,2020. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug570-ultrascale-configuration.pdf

[14] ——, *OPB HWICAP*, 2004, Accessed: Apr. 16,2020. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/opb_hwicap.pdf

[15] ——, *LogiCORE IP XPS HWICAP (v5.01a)*, 2011, Accessed: Apr. 16,2020. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/xps_hwicap/v5_01_a/xps_hwicap.pdf

[16] Intel, *Intel Quartus Prime Pro Edition User Guide: Partial Reconfiguration*, 2020, Accessed: Sep. 20,2020. [Online]. Available: https://www.intel.com/content/www/us/en/programmable/documentation/tnc1513987819990.html

[17] S. Liu, R. N. Pittman, and A. Forin, "Minimizing partial reconfiguration overhead with fully streaming dma engines and intelligent icap controller (abstract only)," in *Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 292. [Online]. Available: https://doi.org/10.1145/1723112.1723190

[18] C. Claus, B. Zhang, W. Stechele, L. Braun, M. Hubner, and J. Becker, "A multi-platform controller allowing for maximum dynamic partial reconfiguration throughput," in *International Conference on Field Programmable Logic and Applications*, Sept. 2008, pp. 535–538.

[19] M. Damschen, L. Bauer, and J. Henkel, "Corq: Enabling runtime reconfiguration under wcet guarantees for real-time systems," *IEEE Embedded Systems Letters*, vol. 9, no. 3, pp. 77–80, Sept., 2017.

[24] M. Liu, W. Kuehn, Z. Lu, and A. Jantsch, "Run-time partial reconfiguration speed investigation and architectural design space exploration,"

[20] L. Pezzarossa, M. Schoeberl, and J. Sparsø, "A controller for dynamic partial reconfiguration in fpga-based real-time systems," in *2017 IEEE 20th International Symposium on Real-Time Distributed Computing (ISORC)*, May 2017, pp. 92–100.

[21] Z. Xiao, D. Koch, and M. Lujan, "A partial reconfiguration controller for altera stratix v fpgas," in *26th International Conference on Field Programmable Logic and Applications (FPL)*. United States: IEEE Computer Society, 8 2016, p. 1.

[22] M. Hubner, D. Gohringer, J. Noguera, and J. Becker, "Fast dynamic and partial reconfiguration data path with low hardware overhead on xilinx fpgas," in *2010 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW)*, April 2010, pp. 1–8.

[23] C. Claus, F. H. Muller, J. Zeppenfeld, and W. Stechele, "A new framework to accelerate virtex-ii pro dynamic partial self-reconfiguration," in *2007 IEEE International Parallel and Distributed Processing Symposium*, March 2007, pp. 1–7.
in *2009 International Conference on Field Programmable Logic and Applications*, Aug 2009, pp. 498–502.

[25] K. Danne and M. Platzner, "Periodic real-time scheduling for fpga computers," in *Third International Workshop on Intelligent Solutions in Embedded Systems, 2005.*, 2005, pp. 117–127.

[26] 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 Trans. Reconfigurable Technol. Syst.*, vol. 8, no. 1, Mar. 2015. [Online]. Available: https://doi.org/10.1145/2629639

[27] E. Lübbers and M. Platzner, "Reconos: Multithreaded programming for reconfigurable computers," *ACM Trans. Embed. Comput. Syst.*, vol. 9, no. 1, Oct. 2009. [Online]. Available: https://doi.org/10.1145/1596532.1596540

[28] Xilinx, *Vivado Design Suite User Guide. Dynamic Function eXchange*, 2020, Accessed: Sep. 1,2020. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2020_1/ug909-vivado-partial-reconfiguration.pdf

[29] ——, *Xilinx Partial Reconfiguration User Guide*, 2018, Accessed: Oct. 3,2019. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_3/ug909-vivado-partial-reconfiguration.pdf

[30] A. Moro, F. Federici, G. Valente, L. Pomante, M. Faccio, and V. Muttillo, "Hardware performance sniffers for embedded systems profiling," in *2015 12th International Workshop on Intelligent Solutions in Embedded Systems (WISES)*, 2015, pp. 29–34.

[31] G. Valente, V. Muttillo, L. Pomante, F. Federici, M. Faccio, A. Moro, S. Ferri, and C. Tieri, "A flexible profiling sub-system for reconfigurable logic architectures," in *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, Feb 2016, pp. 373–376.

[32] G. Valente, T. Fanni, C. Sau, and F. Di Battista, "Layering the monitoring action for improved flexibility and overhead control: work-in-progress," in *Proceedings of the 2020 Embedded Systems Week*, ser. ESWEEK '20, 2020.

[33] G. Valente, T. Di Mascio, G. D'Andrea, and L. Pomante, "Dynamic partial reconfiguration profitability for real-time systems," *IEEE Embedded Systems Letters*, pp. 1–1, 2020.