This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LES.2020.3004302, IEEE Embedded Systems Letters

Dynamic Partial Reconfiguration Profitability for Real-Time Systems

Giacomo Valente, Tania Di Mascio, Gabriella D'Andrea, and Luigi Pomante
Università degli Studi dell'Aquila - DEWS
Via Vetoio, L'Aquila (Italy)
name.surname@univaq.it

***Abstract*—Modern Field-Programmable Gate Arrays offer Dynamic Partial Reconfiguration (DPR) capabilities, a characteristic that opens new scheduling opportunities for real-time applications running on heterogeneous platforms. To evaluate when it is really useful to exploit a DPR, in this letter we present the characterization of its reconfiguration cost in terms of time and a definition of the "DPR Profitability" concept targeting real-time systems. To obtain such results, the components involved in a DPR process have been identified and an innovative approach to calculate the DPR time and its worst-case bound is provided. We validate our approach on a real DPR-compliant platform, showing that our proposal is general enough to be applied to modern DPR-compliant platforms.**

***Index Terms*—Dynamic Reconfiguration, FPGA, Real-Time Systems.**

I. Introduction

Modern *Field Programmable Gate Arrays* (FPGAs) offer *Dynamic Partial Reconfiguration* (DPR) capabilities, enabling the user to dynamically reconfigure a portion of the FPGA, while the remainder of the device continues to operate. This process occurs by moving a reconfiguration file from a memory to an FPGA reconfiguration area, along a *Reconfiguration Path* (RP) composed of buses, memories, reconfiguration controllers and reconfiguration interfaces. Since the DPR produces a change in the HW architecture, it can be used to improve task performances in terms of timing and/or power. These potentialities make the DPR process highly desirable in a wide range of systems, such as hard (e.g., [1], [2], [3], [4]) and soft real-time systems (e.g., [5], [6]). With respect to a full design-time approach, possible advantages can be, e.g., to allow schedulability of a given task set also in the case where the available FPGA resources are not enough to host all the HW tasks at the same time [2], or to keep schedulability also in the case of a run-time changing task set by moving some of the tasks in HW. However, in hard real-time systems, where it is mandatory to guarantee the *worst-case response time* (wcrt) of each process, the usage of DPR rarely appears, due to the non-negligible reconfiguration time impact [1] [5] and the lack of a general approach to calculate its worst-case. In fact, during the last years, the problem of calculating the DPR time has been addressed in literature by first considering it null or constant; then, it has been refined introducing a dependency on the reconfiguration file size and the reconfiguration interface throughput [2]. However, as highlighted in [1] in 2017, all the components of the RP need to be considered to guarantee a worst-case bound of the DPR time (i.e., *worst-case DPR time*, *wcdprt*). In fact, after an in-depth analysis of the literature, we discovered that only the works in [1], [3] and [4] propose an approach to calculate the wcdprt considering this dependency. However, such approaches are based on ad-hoc architectural elements, making them target-dependent. Conversely, in [5] and [6] no ad-hoc architectural elements are involved. However, the proposed measurement-based approaches are not applicable to hard real-time systems, since their DPR time is an average value among different tests that do not involve worst-case conditions. In this letter, we overcome all the limitations described above by providing a novel approach to calculate the DPR time and the wcdprt (i.e., suitable for hard real-time systems). It is based on a deterministic and target-independent static timing analysis that accounts for all the RP elements, and it is potentially applicable to all the actual DPR-compliant platforms [7] [8]. Furthermore, based on such approach, we also formalize the concept of *Profitability* of the DPR process associated to a generic task. This formalization has to be considered as a solid foundation of tools supporting the selection of when to perform the DPR. It is worth noting that this is beyond the scope of this paper, that directly focuses on DPR time calculation. The rest of the paper is organized as follows: in Section II, we present our approach to calculate the DPR time and the wcdprt; in Section III, we formalize the DPR Profitability concept for real-time systems. In Section IV, we apply the proposed approach to an actual DPR-compliant platform, and, in Section V, we conclude with a discussion and future works.

II. DPR time characterization

To provide a general approach to calculate the DPR time, we consider the general reference platform shown in Fig. 1. The DPR follows a RP (see the solid curve line in Fig. 1): a *processor* (PS) transfers a *reconfiguration file* (*bitstream*, BS) from a shared *external memory* to a *local memory*; then, PS transfers the BS from local memory to *Dynamic Reconfiguration Memory* (DRM) through a *Dynamic Reconfiguration Controller* (DRC) and a *Dynamic Reconfiguration Interface* (DRI). The DRM content directly acts on the FPGA array. The DPR time is dependent on the BS size (BS_{size}) and the throughput of the transfer along the whole RP (TP_{RP}):

$$t_{DPR} = \frac{BS_{size}}{TP_{RP}} \tag{1}$$

With the purpose of obtaining an expression of t_{DPR}, we have modeled the BS transfer along the RP as a transfer along a

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LES.2020.3004302, IEEE Embedded Systems Letters

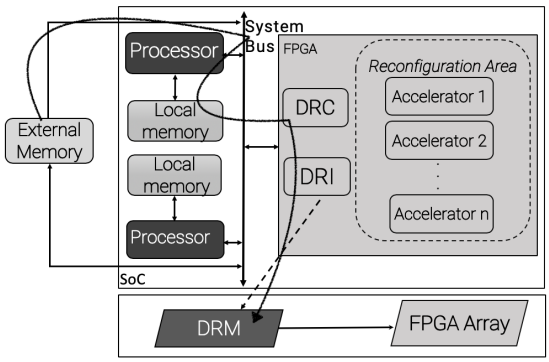

Fig. 1. A general reference platform for DPR, defined by analyzing Xilinx and Intel products [7] [8] and some works from literature [2] [5].

chain of N *elementary paths* (EPs, see Fig. 2a), where the BS can be transferred in chunks of different sizes ($size_{chunk}$). In general, an *i-th elementary path* (EP_i), shown in Fig. 2b, is defined as a master (e.g., processor, DMA) that transfers data between two memories (e.g., RAM, queue) through an interconnection (e.g., bus). For instance, with reference to Fig. 1, an EP is from the shared external memory to the local memory. In our approach, we assume that, by using datasheets, it is possible to clearly identify: (i) the EPs that compose a RP, and (ii) the master, the two memories and the interconnection of each EP. Considering typical DPR-compliant platforms (e.g., [9] [8]), this is not a strong assumption, since their datasheets report information at this level of detail.

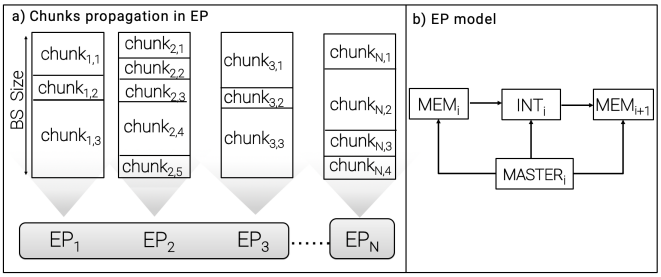

Fig. 2. Reconfiguration path as a chain of N EPs. a) shows the chunks propagation in elementary paths. b) shows the EP model.

With the proposed modeling, t_{DPR} can be expressed as:

$$t_{DPR} = f_M(BS_{size}, TP_{i,j}, RP_{M_{i,j}})$$
$$i = 1, ..., N \quad j = 1, ..., F_i \tag{2}$$

where N is the number of EPs that compose the RP and F_i is the number of chunks to be transferred within the i-th EP (EP_i). In Eq. (2), t_{DPR} is a function (f_M) of: (i) the BS_{size}, (ii) the $TP_{i,j}$, that represents, $\forall\ EP_i$, the throughput associated to the transfer of each $chunk_{i,j}$ (of size $size_{chunk_{i,j}}$) within EP_i, and (iii) $RP_{M_{i,j}}$, that represents the RP policy adopted to manage the transfer of chunks among adjacent EPs. Then, we make the following hypothesis: (hp_1) concurrent transfers (i.e., pipelined) among EPs are not allowed (this represents a worst-case); (hp_2) $size_{chunk_{i,j}}$ is constant within each EP (the worst-case is given by assuming the chunk as the smallest possible). Consequently, considering hp_1, the f_M can be written as a sum, while considering hp_2 we have that $F_i = BS_{size}/size_{chunk_i}$, valid $\forall\ EP_i$. Eq. (2) becomes:

$$t_{DPR} = \sum_{i=1}^{N}\left[\sum_{j=1}^{F_i}\left(\frac{size_{chunk_{i,j}}}{TP_{i,j}}\right)\right] =$$
$$\sum_{i=1}^{N}\left[\left(\frac{BS_{size}}{size_{chunk_i}} \cdot \frac{size_{chunk_i}}{TP_i}\right)\right] = \sum_{i=1}^{N}\frac{BS_{size}}{size_{chunk_i}} \cdot t_{M_i} \tag{3}$$

where TP_i has been expressed as $TP_i = size_{chunk_i}/t_{M_i}$, with t_{M_i} being the time required to transfer the $chunk_i$ along the EP_i. Considering the EP model, t_{M_i} depends on:

$$t_{M_i} = f_t(t_{MEM_i}, t_{MEM_{i+1}}, t_{INT_i}, t_{MST_i}) \quad i = 1, ..., N \tag{4}$$

In turn, t_{MEM_i}, t_{INT_i} and t_{MST_i} depend, $\forall\ EP_i$, on:

$$t_{MEM_i} = f_{MEM_i}(t^a_{MEM_i}, t^s_{MEM_i}) \tag{5}$$
$$t_{INT_i} = f_{INT_i}(t^t_{INT_i}, t^s_{INT_i}) \tag{6}$$
$$t_{MST_i} = f_{MST_i}(t^e_{MST_i}, t^s_{MST_i}) \tag{7}$$

where, in Eq. (5), $t^a_{MEM_i}$ is the time to perform an exclusive access to MEM_i and $t^s_{MEM_i}$ is the waiting time caused by MEM_i sharing. In particular:

$$t^a_{MEM_i} = f^a_{MEM_i}(freq_{in}, freq_{out}, size_{in}, size_{out}, size_{tot}, latency) \tag{8}$$
$$t^s_{MEM_i} = f^s_{MEM_i}(sh_{policy}) \tag{9}$$

where, $\forall\ MEM_i$, starting from the datasheet of the selected DPR-compliant platform, it is possible to identify the values of $t^a_{MEM_i}$ and $t^s_{MEM_i}$: $freq_{in}$ and $freq_{out}$ are the input and output frequencies of MEM_i, $size_{in}$ and $size_{out}$ are the input and output ports sizes, $size_{tot}$ is the total memory capacity, sh_{policy} is a factor to be kept into account for memory sharing, and, finally, $latency$ is the time to access to a generic data stored in memory; $latency$, expressed in number of clock cycles, is considered constant for the access to every memory address, except the time to wait for the memory sharing. In Eq. (6), $t^t_{INT_i}$ is the time to perform an exclusive access to INT_i and $t^s_{INT_i}$ is the waiting time caused by INT_i sharing. Since INT_i could model more than one HW interconnection (e.g., a cascade of buses, with different data sizes [9]), within each EP we refer to the interconnection with the lowest bandwidth. In particular:

$$t^t_{INT_i} = f^t_{INT_i}(freq, data_{size}, beat_{size,lat,num,WT}) \tag{10}$$
$$t^s_{INT_i} = f^s_{INT_i}(sh_{policy}) \tag{11}$$

where, $\forall\ INT_i$, $freq$ is the working frequency, $data_{size}$ is the size of the bus data, sh_{policy} is a factor to be taken into account for interconnection sharing. Moreover, $beat_{size,lat,num,WT}$ are four parameters related to the bus transfer policy. Without loss of generality, we consider a policy based on bursts, in which each burst is composed of $beat_{num}$ number of beats, and each beat has a size, namely the $beat_{size}$. Each beat requires some time to be transmitted, indicated as $beat_{lat}$, expressed in clock cycles, and there can be a waiting time, $beat_{WT}$, between different beats. $t^s_{INT_i}$ is calculated starting from datasheets, while $t^t_{INT_i}$ can be further expanded as follows:

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LES.2020.3004302, IEEE Embedded Systems Letters

$$t^t_{INT_i} = \sum_{k=0}^{G} burst_{ctb_k} = \sum_{k=0}^{G} \left[\left(beat_{lat_k} + beat_{WT_k} \right) \cdot beat_{num_k} \right] \cdot \frac{1}{freq} \quad (12)$$

where G are the total bursts required to transfer a $chunk_i$ and $burst_{ctb_k}$ is the contribution of the $k-th$ burst to $t^t_{INT_i}$. In Eq. (7), $t^e_{MST_i}$ is the time to execute commands by the MST_i and $t^s_{MST_i}$ is a waiting time caused by MST_i sharing. In particular:

$$t^e_{MST_i} = f^e_{MST_i}(num_{cmd}, master_{time}) \quad (13)$$

$$t^s_{MST_i} = f^s_{MST_i}(sh_{policy}) \quad (14)$$

where, $\forall\ MST_i$, num_{cmd} is the number of commands to be executed to manage the transfer, $master_{time}$ is the time to execute a command, and sh_{policy} is a factor to be taken into account for master sharing. $t^s_{MST_i}$ and $t^e_{MST_i}$ are calculated starting from datasheets. The function f_t in Eq. (4) can be written, without loss of generality, as follows:

$$t_{M_i} = G \cdot (t_{MEM_i} + t_{MEM_{i+1}} + t_{MST_i}) + t_{INT_i} \quad (15)$$

where we consider that the memory accesses and the master time are spent per-burst. Expanding t_{INT_i} and bringing the contribution of MEM and MST in the sum in Eq. (15), we get:

$$t_{M_i} = \sum_{k=0}^{G} \big(t^a_{MEM_{i,k}} + t^s_{MEM_{i,k}} + t^a_{MEM_{i+1,k}} + t^s_{MEM_{i+1,k}} + burst_{ctb_k} + t^s_{INT_{i.k}} + t^s_{MST_{i,k}} + t^e_{MST_{i,k}} \big) \quad (16)$$

In conclusion, using Eq. (16) in Eq. (3), we can write the expression of t_{DPR}, that can be used also to get the wcdprt.

III. DPR Profitability formalization

To exploit the proposed DPR time characterization for the evaluation of the profitability in accelerating a task execution (*DPR Profitability*), we propose the following definition:

Definition 1. *The DPR Profitability P is a Boolean variable to evaluate whether doing a DPR to accelerate a task is useful or not. P can take the following values :*

$$P = \begin{cases} 1 & \text{if } t^A_{TK} + t_{DPR} \leq D_{TK} \\ 0 & \text{otherwise.} \end{cases} \quad (17)$$

where D_{TK} is the deadline of the task, t^A_{TK} is the execution time of the accelerated task and t_{DPR} is the time to perform the DPR.

D_{TK} and t^A_{TK} are given at design time, while t_{DPR} is calculated by means of Eq. (3). In the case of hard real-time systems, t^A_{TK} is replaced by its wcrt, and t_{DPR} by its wcdprt.

IV. Experimental activities

In order to validate the proposed approach on a real DPR-compliant platform, we apply it on the Xilinx Zynq7000 PCAP-associated RP [9]. The considered case study is a task executed on a single-core ARM Cortex A9 on a *Zedboard* [10]. The task can be accelerated using an IP-core for matrix multiplication, introduced with the DPR through the *Processor Configuration Access Port* (PCAP) [9] [11]. The BS related to the accelerator is stored within the external DRAM [10] and its size is $BS_{size} = 857740$ bytes. For this validation, we refer to the t_{DPR} calculated by means of Eq.(3) as t^{CAL}_{DPR}. Using the Xilinx datasheet [9], we first modelled the PCAP-associated RP of Zynq-7000, which results to be composed of three elementary paths. Fig. 3 shows the RP in *a)* and its model in *b)*. Then, from Eq. (3):

$$t^{CAL}_{DPR} = \sum_{i=1}^{3} \frac{BS_{size}}{TP_i} = \sum_{i=1}^{3} \left(\frac{BS_{size}}{chunk_i} \cdot t_{M_i} \right) \quad (18)$$

In the considered application, MEM_1 (i.e., the external DRAM) is not shared: this causes a negligible contribution of EP_1 ($\approx$ ns), while DPR times are in the order of ms [2]. Moreover, for the EP_3 the datasheet directly provides the transfer bandwidth TP_3, that is equal to the PCAP bandwidth BW_{PCAP} [9]. Then:

$$t^{CAL}_{DPR} \approx \left(\frac{BS_{size}}{chunk_2} \cdot t_{M_2} + \frac{BS_{size}}{BW_{PCAP}} \right) \quad (19)$$

In the EP_2, $\forall k$, since the memory access contributions are negligible ($\approx$ ns, since they are on-chip memories and not shared [9]), we assume $t^a_{MEM_{2,k}} = 0$, $t^s_{MEM_{2,k}} = 0$, $t^a_{MEM_{3,k}} = 0$ and $t^s_{MEM_{3,k}} = 0$. INT_2 refers to the interconnection with the lowest bandwidth within EP_2 that, in the case of Fig. 3, is the *SI* interconnect (INT_2 hereinafter). In the proposed application, INT_2 is not shared ($t^s_{INT_{2,k}} = 0\ \forall k$). We know from documentation [9] that INT_2 is based on NIC301 ARM interconnect [12]. With reference to the bus model (AMBA [13]), which is not shared, we assume that INT_2, $\forall k$, has a constant $beat_{num_k}$ (hp_3), a constant $beat_{size_k}$ (hp_4) and a $beat_{WT_k} = 0$ (hp_5). Moreover, for INT_2 we have $beat_{lat_k} = 1$ clock cycle. hp_3, hp_4 and hp_5 represent the worst-case in the considered application, since we have paths composed of one master and one slave, with a one clock cycle arbitration and a full pipelining among transfers, providing a continuous beats transmission [12] [13]. Finally, the master of the EP_2, represented by a DMA, has $num_{cmd} = 1$, for which the $master_{time}$ is negligible ($\approx$ ns, since it is a dedicated finite state-machine [9])). Moreover, it is not shared, so $t^e_{MST_{2,k}} = 0$ and $t^s_{MST_{2,k}} = 0\ \forall k$. For the EP_2 we then have:

$$t_{M_2} = \sum_{k=0}^{G} burst_{ctb_k} \approx \sum_{k=0}^{\left(\frac{chunk_2}{beat_{num_2} \cdot beat_{size_2}}\right)} burst_{ctb_k} \approx \left(\frac{chunk_2}{beat_{num_2} \cdot beat_{size_2}} \cdot beat_{num_2} \cdot \frac{1}{freq_2} \right) \quad (20)$$

The final expression of t^{CAL}_{DPR} becomes:

$$t^{CAL}_{DPR} \approx BS_{size} \left(\frac{1}{freq_2 \cdot beat_{size_2}} + \frac{1}{BW_{PCAP}} \right) \quad (21)$$

By means of Eq. (21), it is now possible to evaluate the DPR time and the wcdprt for the considered BS. This is done by considering all the possible values for the unknown $beat_{size_2}$ parameter (the datasheet does not provide enough

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LES.2020.3004302, IEEE Embedded Systems Letters

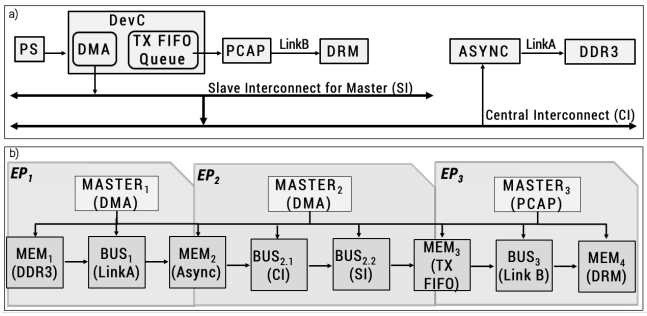

Fig. 3. Model of the PCAP-associated RP of Zynq7000. a) shows the RP, where a *Processor System* (PS) commands the start of DPR: a *Direct Memory Access* (DMA) transfers the BS from a *DDR3* DRAM external memory (where the BS is initially stored) to a *TX FIFO* queue, through two AMBA [13] buses (*Central Interconnect* (CI) and *Slave Interconnect for Master* (SI)). Due to different clock domains between the involved HW elements, an *ASYNC* queue is present, connected with a custom bus *LinkA* with the DDR3 controller. From TX FIFO, a PCAP starts transferring BS from TX FIFO to DRM, through a custom bus *LinkB*. When the transfer is complete, an interrupt is signaled to PS. b) shows the model using three EPs: below each component, there is the correspondent indication of the modeled HW element.

details and it is normally not observable, since it is a hardwired implementation). Since the bus related to EP_2 has a width of 32 bit, $beat_{size_2}$ can be only of 1, 2, 3 or 4 bytes [13]. So, recalling the the $BS_{size} = 857740$ bytes for the considered application, t_{DPR}^{CAL} can be 8.6 ms, 5.4 ms, 4.3 ms or 3.7 ms, where 8.6 ms represents the wcdprt. This is fully reasonable since, with a $beat_{size_2}$ equal to 1, we are exploiting the available bus in the worst possible way. By introducing a custom HW monitoring system [14], we measured the real value of the DPR time for the same BS, indicated as t_{DPR}^{ACT}, that results in 6.6 ms. Such a value, as expected, belongs to the interval [3.7, 8.6] ms.

TABLE I
RESULTS FOR $BS_{size} = 857740$ BYTES, WHERE $acc = |t_{DPR}^{CAL} - t_{DPR}^{ACT}|/t_{DPR}^{ACT}$

$beat_{size_2}$ (bytes)	**1 (worst case)**	2	3	4
$t_{DPR}^{CAL} - t_{DPR}^{ACT}(ms)$	**2**	-1.2	-2.3	-2.9
$acc_{\%}$	**30.3**	18.1	34.8	43.9

V. DISCUSSION AND FUTURE WORKS

In this letter, we have dealt with the exploitation of the DPR process in the real-time systems domain. With respect to the literature, three main innovative results have been obtained: (i) the definition of a general approach to calculate the DPR time and the wcdprt in all DPR-compliant platforms, (ii) the formalization of the DPR Profitability, and (iii) the calculation of the DPR time and the wcdprt associated to the PCAP RP of Zynq7000 SoC [10]. In particular, we have discovered that the DPR time and the wcdprt for that RP are mainly dependent on the AHB-lite internal bus. Table 1 reports the calculated results and their accuracy, using the four possible values of the $beat_{size_2}$ parameter [13], and it highlights a wcdprt accuracy of 30.3%. Differently from other works in literature (i.e, [1], [3], [4]), our approach is target-independent and, to the best of our knowledge, our wcdprt represents the first worst-case bound (i.e., suitable for hard real-time systems) related to Zynq7000 PCAP RP. At the same time, the other values in the table, that present an accuracy comparable with other works in literature [5], can be used for more speculative scheduling approaches in soft real-time systems. As a final result, the achieved DPR time and wcdprt are exploited to evaluate the DPR Profitability. As future works, the accuracy of our calculation will be enhanced by overcoming the lack of observability inside Zynq7000 [10] with the insertion of custom monitoring systems. Moreover, we will continue validating the proposed t_{DPR} calculation with other platforms and applications, together with the exploitation of the proposed DPR Profitability concept to improve existing tools for schedulability analysis of real-time systems (e.g., [2]). With respect to applications, we will improve their complexity by considering scenarios with multiple shared resources.

ACKNOWLEDGMENT

This work is part of the FitOptiVis project funded by the ECSEL Joint Undertaking under grant number H2020-ECSEL-2017-2-783162.

REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] M. Kirchhoff, P. Kerling, D. Streitferdt, and W. Fengler, "A real-time capable dynamic partial reconfiguration system for an application-specific soft-core processor," *Int. J. Reconfig. Comp.*, vol. 2019, pp. 4 723 838:1–4 723 838:14, 2019.

[5] K. Papadimitriou, A. Dollas, and S. Hauck, "Performance of partial reconfiguration in fpga systems: A survey and a cost model," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 4, no. 4, pp. 36:1–36:24, Dec., 2011.

[6] 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.

[7] Xilinx - Adaptable. Intelligent. https://www.xilinx.com. (Accessed: Oct. 3,2019).

[8] Intel FPGAs and Programmable Devices. https://www.intel.it/content/www/it/it/products/programmable.html. (Accessed: Oct. 3,2019).

[9] 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

[10] AVNET, *ZedBoard Hardware User's Guide*, 2014, Accessed: Oct. 3,2019. [Online]. Available: http://zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf

[11] 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.

[12] ARM, *CoreLink™ Network Interconnect NIC-301. Revision: r2p2. Technical Reference Manual*, 2006-2010, Accessed: Feb. 11,2020. [Online]. Available: https://static.docs.arm.com/ddi0397/h/DDI0397H_corelink_network_interconnect_nic301_r2p2_trm.pdf

[13] ——, *AMBA:The Standard for On-Chip Communication*, 2019, Accessed: Oct. 3,2019. [Online]. Available: https://www.arm.com/products/silicon-ip-system/embedded-system-design/amba-specifications

[14] 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.