Reconfiguration in FPGA-Based Multi-Core Platforms for Hard Real-Time Applications

Luca Pezzarossa, Martin Schoeberl, and Jens Sparsø
Department of Applied Mathematics and Computer Science
Technical University of Denmark, Kgs. Lyngby
Email: [lpez, masca, jspa]@dtu.dk

***Abstract*—In general-purpose computing multi-core platforms, hardware accelerators and reconfiguration are means to improve performance; i.e., the average-case execution time of a software application. In hard real-time systems, such average-case speed-up is not in itself relevant – it is the worst-case execution-time of tasks of an application that determines the systems ability to respond in time. To support this focus, the platform must provide service guarantees for both communication and computation resources. In addition, many hard real-time applications have multiple modes of operation, and each mode has specific requirements. An interesting perspective on reconfigurable computing is to exploit run-time reconfiguration to support mode changes. In this paper we explore approaches to reconfiguration of communication and computation resources in the T-CREST hard real-time multi-core platform. The reconfiguration of communication resources is supported by extending the message-passing network-on-chip with capabilities for setting up, tearing down, and modifying the bandwidth of virtual circuits. The reconfiguration of computation resources, such as hardware accelerators, is performed using the dynamic partial reconfiguration capabilities found in modern FPGAs.**

I. Introduction

Hard real-time systems are a class of systems characterized by strict constraints on the execution time of tasks. These systems are used for safety-critical applications, where a failure to respond in time may lead to catastrophic consequences. Some examples are: flight electronics, wind turbine control systems, medical devices, and factory automation systems.

In a multi-core platform supporting real-time applications, the requirement for time-predictable behavior goes beyond the individual processors; the network-on-chip (NoC) supporting communication among tasks mapped to different cores must also be time predictable. This is typically achieved using end-to-end virtual circuits for which bandwidth and latency can be guaranteed.

Field programmable gate array (FPGA) technology has advanced to a point where it is often used in end-products. For hard real-time systems that are mainly used in low-volume professional or high-end applications, this is particularly attractive since it is not possible to amortize the development costs of an application-specific integrated circuit over the production volume. In addition, FPGAs allow the integration of application-specific I/O-devices and hardware accelerators, thereby reducing the component count. Many FPGA devices allow portions of the chip to be reconfigured at run-time, while the rest of the device continues to operate without interruption [1]–[3]. The use of such dynamic partial reconfiguration (DPR) is currently an active area of research in general-purpose computing, where it can result in reduced hardware cost and/or higher performance [4].

978-1-5090-2520-6/16/$31.00 ©2016 IEEE

The T-CREST multi-core platform [5] is an example of an FPGA-based multi-core platform for hard real-time systems. The platform currently offers *boot-time configuration* of both its *communication resources* (message-passing NoC) and its *computation resources* (hardware accelerators).

Hard real-time systems often have multiple modes of operation, sometimes called use-cases, that they switch between during normal operation. Mode changes can occur as part of the normal and planned operation or in response to external events [6, p.340]. In a multi-core platform, a mode of operation can be defined as a set of tasks executing on a set of processors and communicating across a set of virtual circuits provided by the NoC.

In this paper we explore run-time reconfiguration in hard real-time systems to implement mode changes in the context of the T-CREST platform. Such a run-time reconfiguration relates to both communication resources (the virtual circuits provided by the NoC) and computation resources (hardware accelerators). The reconfiguration of communication resources is achieved by dynamically setting-up and tearing-down virtual circuits and by dynamically altering their bandwidth and latency guarantees. The reconfiguration of computation resources includes starting and stopping of tasks and DPR of co-processors and hardware accelerators. Since the reconfiguration is now part of the normal functionality of the system, it has to be time predictable as well. To our knowledge, the use of DPR of FPGAs to support mode changes in hard real-time systems represents a novel and unexplored field of research.

The paper has two main contributions: (1) it presents and extends the boot-time configuration capabilities of the T-CREST platform message-passing NoC Argo to implement run-time reconfiguration and thereby support the *communication aspects* of a mode change, and (2) it explores and evaluates how to use the DPR-capability of modern FPGAs to support the *computation aspects* of a mode change by dynamically reconfiguring computation resources in the platform.

The paper is organized as follows. Section II presents related work about reconfiguration of communication and computation resources. Section III gives a brief introduction to the T-CREST multi-core platform used as a basis for our work. Section IV provides background on the message-passing NoC Argo and its extension to implement run-time reconfiguration. Section V explores the usage of DPR to support the dynamic reconfiguration of computation resources in the platform. Section VI concludes the paper.

II. Related Work

A multi-core platform for hard real-time systems has to provide time-predictable inter-processor communication. This calls for a NoC that offers guaranteed bandwidth and latency for end-to-end communication flows. Such guaranteed service (GS) connections can be implemented by using non-blocking routers in combination with mechanisms that constrain packet injection rates, or by using some form of (virtual) circuit switching.

The NoC used in the Kalray MPPA-256 processor uses flow regulation [7], output-buffered routers with round-robin arbitration, and no flow control. Network calculus [8] is used to determine the flow regulation parameters that constrain the packet injection rates such that buffer overflows are avoided and GS requirements are fulfilled. In the perspective of this paper, the Kalray NoC is configured by initializing the routing tables and injection rate limits in the network interfaces (NIs).

The alternative, virtual circuit switching, is often implemented using static scheduling and time-division-multiplexing (TDM). Examples are the Æthereal family of NoCs [9], [10] and the Argo NoC [11]. In the perspective of this paper, these TDM-based NoCs are configured by initializing schedule tables and routing tables in the NIs and/or the routers.

The original Æthereal NoC supports both GS and best effort traffic. The scheduling tables are in the NIs and the routing tables are in the routers. They can be written by sending best effort packets, and this aspect compromises the time-predictability of a (re)configuration.

The aelite NoC [9] only supports GS traffic and it is based on source routing. The routers are simple pipelined switches and both schedule tables and routing tables are in the NIs. For aelite, reconfiguration involves sending messages across the NoC itself, and for dAElite [10] using a separate dedicated network.

In all cases, reconfiguration is done incrementally. From this follows that virtual circuits that persist across the reconfiguration cannot be re-mapped. This can lead to fragmentation of resources resulting in sub-optimal use of resources. If re-mapping of virtual circuits is needed, the entire application must be suspended during the reconfiguration.

A preliminary solution for the Argo NoC reconfiguration was studied and implemented in the work presented in [24]. This solution also uses a dedicated asynchronous tree network to broadcast the new TDM schedule and the commands to trigger the reconfiguration.

Essentially, the extension presented in this paper for our Argo NoC implements the same functionality, but using fewer hardware resources and supporting instantaneous reconfiguration, including re-mapping of virtual circuits.

The second topic of this paper relates to mode changes and DPR of FPGAs. A good survey of the most relevant hardware aspects of reconfigurable computing can be found in [4]. The work addresses both single-chip and multi-chip architectures and explores the challenges of run-time hardware reconfigurable architectures, with special focus on internal structures and coupling.

The ReCoBus-builder [12] is an FPGA-design-oriented framework for component-based, reconfigurable, non-real-time systems. It uses DPR to generate dynamically reconfigurable systems providing one or more run-time reconfigurable areas. For the communication between the reconfigurable resources and other parts of the system, it uses a fixed bus infrastructure or dedicated point-to-point links.

PaRA-Sched [13] is an automated design methodology that takes into account DPR in the scheduling infrastructure to improve overall performance by automatically masking reconfiguration time when possible. This allows a rapid exploration of the DPR impact during the early stages of the design process.

The work presented in [14] studies the dynamic behavior of reconfigurable architectures, especially focusing on the usage of dynamic partial reconfiguration. It proposes a simulation framework for reconfigurable architectures that comprises a generic application model and an architecture model, the combination of which captures the dynamic behavior of the reconfigurable architectures.

The work presented in [15] provides an overview of the hardware-software partitioning, scheduling, and placement issues and proposes an exact and a heuristic approach for hardware-software partitioning. This takes into account key factors such as placement implications and configuration pre-fetching for minimizing the schedule length.

III. The T-CREST Multi-Core Platform

This section provides a brief overview of the multi-core platform T-CREST [5]. The T-CREST platform has been developed specifically for use in hard real-time applications. All components have been designed with a focus on time-predictability and worst-case execution time (WCET) analysis and with a focus on reducing the complexity and pessimism of the WCET analysis. Figure 1 shows the overall architecture of the T-CREST platform. It consists of a number of processing nodes and two NoCs: a NoC, called Argo [11], that provides message-passing to support inter-processor communication, and a shared memory access NoC [16].

A processing node consists of a time predictable, dual-issue RISC processor, called Patmos, that is optimized for real-time systems [17], special instruction and data cache memories, and local private scratchpad memories (SPMs) for instructions and data.

The Argo NoC offers the possibility to set up virtual point-to-point circuits between processor nodes. Data is pushed across these circuits by direct memory access (DMA) controllers in the source end of the circuit.

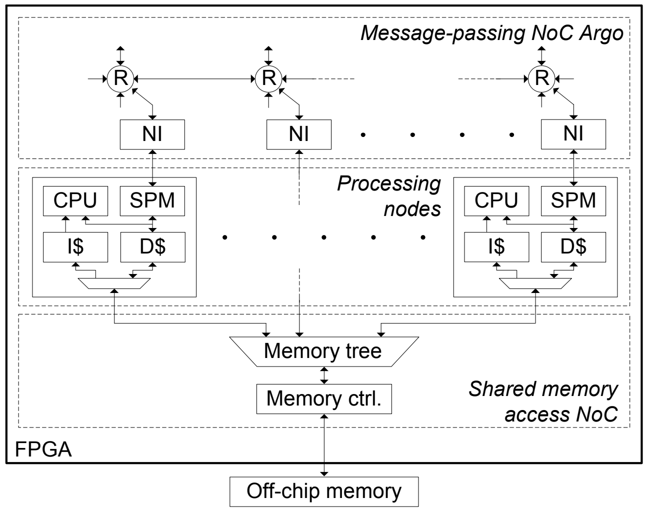

Fig. 1. The overall T-CREST architecture divided into three main parts: the message-passing NoC Argo, the processing nodes, and the shared memory access NoC. Each processing node contains the Patmos processor, a SPM, and the instruction and data caches. The shared memory is off-chip.

The shared memory access NoC [18] is a tree network that supports transfer of data between local caches and an external and shared main memory via a real-time memory controller [19]. It does not include any hardware support for cache coherency, as inter-processor communication mainly uses the Argo NoC. Communication via shared memory is possible, but coherency mechanisms have to be implemented in software.

The platform is supported by a compiler also developed with a focus on WCET [20]. Moreover, the WCET analysis tool aiT [21] from AbsInt, which allows statical derivation of tight WCET bounds, has been extended to support the Patmos processor.

The T-CREST platform was designed with particular attention on FPGA implementation, and in the base implementation, only the Argo message-passing NoC is configurable at boot time where the virtual circuits are set up.

In the remainder of this paper, we address how to extend this platform with capabilities for run-time reconfiguration in order to support mode changes. In Section IV, we first consider the *communication aspects* and extend the Argo NoC with functionality to dynamically alter the set of virtual circuits, in a way that is transparent to virtual circuits that persist across the mode change. Then, in Section V, we address the *computation aspects* by exploring DPR capabilities of the FPGA. Here the idea is to expand the T-CREST platform to use heterogeneous processing nodes such as co-processors, hardware accelerators, digital signal processors, etc., and to dynamically adapt the hardware to the different modes of operations.

IV. Reconfiguration of the Communication Resources

In this section, we first provide additional background about TDM scheduling and the micro-architecture of the Argo NoC network interface (NI). Then we describe how the NI can be

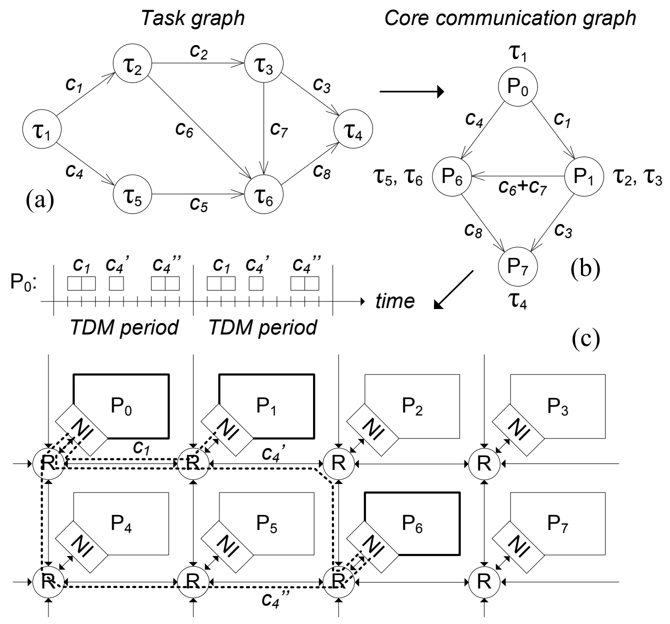

Fig. 2. Mapping of an application into a multi-core platform: (a) task graph for an application, (b) core communication graph, (c) section of multi-core platform with possible routing and schedule for processor P_0.

extended from being boot-time configurable to being run-time reconfigurable.

A. TDM Scheduling

The mapping of a multi-mode, real-time application into a multi-core platform and the generation of a TDM schedule in the T-CREST platform is performed through the steps shown in Figure 2.

Each mode of operation consists of a set of communicating tasks. A set can be modeled as a *task graph* (Figure 2(a)), where the nodes represent tasks and the edges represent the communication between the tasks.

By assigning the tasks to the processing nodes, it is possible to derive a *core communication graph* (Figure 2(b)). The assignment of tasks to processing nodes has to be performed in a way that minimizes the total number of hops for traffic. For this graph, the nodes represent the processing nodes, and the edges represent the communication channels between each pair of nodes.

In the T-CREST platform, the generation of the task graph and the core communication graph for each mode of operation is performed manually. The scheduler, described in [22], is an off-line procedure that uses the bandwidth requirements and a description of the NoC topology to generate a schedule that avoids deadlocks and collisions, and that ensures in-order arrival of packets.

The time is divided into periods, and a period is further divided in time-slots. Figure 2(c) shows two TDM periods for the traffic out of the processor P_0 (channel *c1* and *c4*) and the communication channel paths on a section of the multi-core platform. Channel *c1* has been assigned two time slots and channel *c4* has been assigned three time slots through two different paths in the NoC (*c4'* and *c4"*).

B. The Argo NoC

Argo is a TDM-based, packet-switched, source-routed NoC for hard real-time multi-processor platforms. Argo allows an application programmer to set up a set of virtual point-to-point circuits between processor nodes providing GS message-passing. The interface between the NoC and a processor node is shown in Figure 1. For each node, the processor is connected to one of the ports of a dual-ported SPM, while the NI is connected to the second port.

Through the virtual circuits, blocks of data can be transferred from the SPM of a node into the SPM of a remote node. The data transfer is managed by TDM-driven DMA controllers integrated in the NIs of the NoC. A DMA controller is dedicated to each virtual circuit in the sender, inserting the packets, according to the static TDM schedule, into the shared routing structure. The static TDM schedule avoids collisions by design and no flow control or buffering is needed.

Figure 3 shows a block diagram of the Argo NI [23], including our addition to support reconfiguration. The Argo NI contains a *slot counter* that is reset at the end of a schedule period and it indexes into a *slot table*. Each entry in the slot table points to an entry in the *DMA table* that stores the counters and pointers corresponding to a DMA controller. The DMA controller reads the data to be sent from the SPM, assembles the packet and sends it, in the time slot designated by the TDM counter. Each DMA controller implements the source end of a virtual circuit.

In order to transfer a block of data stored in its local SPM, a processor has to set up the TDM-driven DMAs in the NI, through the configuration interface. The static schedule, which has the same length for all nodes, is stored in the NIs of the NoC, and it specifies the route of each packet and the time slot in which each packet must leave the the NI. Incoming packets are directly written into the destination SPM at the target address carried by the packet.

The Argo NoC uses a 5-ported router based on a pipelined crossbar that routes incoming packets according to the routing information contained in the packet header. Argo supports both synchronous, mesochronous, and asynchronous router implementations. The topology of the NoC is configurable (mesh, bitorus, etc.).

C. Support for NoC Reconfiguration

The Argo NoC needs to be configured at boot-time. The configuration process is executed by each processor, and it consists of writing the scheduling information into the slot table of each NI. Once all the NIs are configured, the NoC can start to operate.

A simple way of reconfiguring the communication bandwidth and latency of the NoC is to reconfigure, at run-time, the scheduling information stored in the NIs of the NoC, in a way similar to the initial configuration at boot-time.

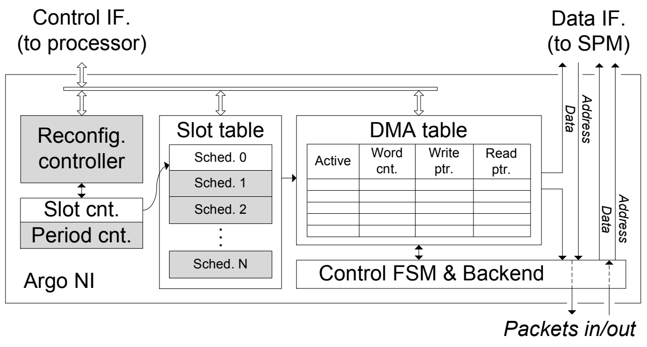

Fig. 3. Network interface of the Argo NoC. The blocks highlighted in gray are introduced to support reconfiguration.

We have investigated and presented an approach for reconfiguration of the Argo NoC in [25]. The blocks of Figure 3 highlighted in gray show the extension of the NI introduced to support reconfiguration.

The slot table is extended in order to store multiple schedules, each spanning a range of entries. Each schedule is represented by two pointers to the minimum and the maximum of the range. These pointers are stores in a small table in the *reconfiguration controller*. A reconfiguration simply requires that the TDM counter is set to the start entry of a new schedule at the end of a TDM period. A schedule period counter is also introduced to provide all the NIs with the same notion of the current period.

The schedule distribution and the NoC reconfiguration are performed by the individual processors, and these operation are coordinated through shared memory-based communication. A reconfiguration master invokes a reconfiguration of the NoC by sending a special packet to the reconfiguration controller of all the slave NIs, announcing a change to the new schedule. This packet contains the index of the reconfiguration table entry that holds the high and low pointers for the new schedule and the moment in time (TDM period) after which the new schedule should be applied. When that moment comes, according to the period counter, all the NIs start using the new schedule.

This approach to reconfiguration supports instant switching from one schedule to another, synchronously across all NIs. This allows virtual circuits that persist across a mode change to be mapped to different paths, without any interference on the data flow.

V. Reconfiguration of the Computation Resources

The reconfiguration of the computation resources in the T-CREST platform uses DPR to reconfigure, at run-time, the resources of the platform dedicated to computational tasks.

In this section, we provide the background information regarding DPR, we describe the hardware platform with DPR support and our reconfiguration approach, and we finally show an application example and evaluation.

A. DPR Background

DPR allows the modification of an operating FPGA. Partial bit-files can be loaded into the FPGA to reconfigure selected regions, without compromising the integrity and the functionality of those parts of the device not affected by the reconfiguration.

Therefore, a system that uses DPR can be conceptually considered as divided in two main parts: a static part and a dynamic part. The static part is configured only once at boot-time with a full bit-file. The dynamic part, which may consist of several independent reconfigurable regions, can be reconfigured multiple times during run-time with different partial bit-files.

For Xilinx FPGAs, the system implemented on the static part can perform the reconfiguration of the logic implemented on the dynamic part of the system through the internal configuration access port (ICAP) [26] . The ICAP provides access to the configuration memory. The ICAP interface is a streaming interface that receives partial bit-files as input. The address of the FPGA configuration memory and the control signals are not directly available on the interface, all the control information needed to manage the reconfiguration, such as commands, frame address, etc., are encoded into the bit-file, together with the data to be written into the configuration memory. The ICAP interface also provides information about the current state of the reconfiguration and communicates when the reconfigurable region is successfully reconfigured. An additional interface is also needed that connects the static part with the dynamic part in order to decouple the reconfigurable regions during reconfiguration.

The current FPGA technology introduces some limitations on the smallest size and on the shape of a reconfigurable region. The smallest reconfigurable region is called *base region* and it corresponds to the smallest addressable segments of the FPGA configuration memory space. Depending by the reconfigurable element type, the base region in Virtex-6 FPGAs is equivalent to 80 slices, or to 16 digital signal-processing elements (DSP), or to 8 blocks of RAM (BRAM). Moreover, the reconfigurable region must always be rectangular shaped. This topological constrain, combined with the fact that the base regions containing DSPs and BRAMs are uniformly distributed in the FPGA chip, may lead to an over-inclusion of resources into a reconfigurable area.

B. DPR Granularity and Reconfiguration Latency

When considering exploiting DPR to support mode changes, two issues must be considered: (1) What functionality is it relevant to implement in a DPR-region? (2) How long does it take to perform the DPR? We have identified three classes of DPR granularity: fine, medium, and coarse, as illustrated in Figure 4.

Fine-grain DPR involves reconfiguring a fraction of a processor node, for example by adding or removing hardware support for certain instructions. This involves several hundred slices in the FPGA. Medium-grain DPR involves reconfiguring a complete processor node. This is for example relevant when a processor node is a (stateless) hardware accelerator that is shared and accessed through the Argo NoC, and it involves several thousand slices. Coarse-grain DPR involves even larger areas, for example adding or removing stateful hardware accelerators for compute-intensive operations (e.g., fast Fourier transformation, encryption/decryption etc.). This involves tens of thousands of slices.

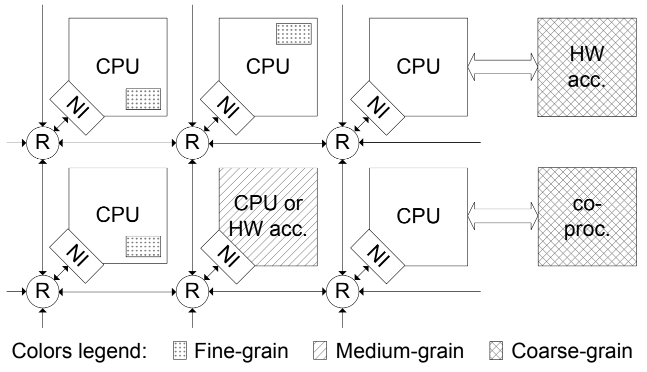

Fig. 4. Example of the three DPR granularities in a multi-core platform based on a network-on-chip. Fine-grain DPR corresponds to small modifications in the CPU architecture, medium-grain to the modification of an entire core, and coarse-grain to the reconfiguration of big accelerators or co-processors.

TABLE I
CALCULATED RECONFIGURATION LATENCIES FOR THREE DIFFERENT RECONFIGURABLE REGION SIZES IN A XILINX VIRTEX-6 FPGA.

DPR	Hardware resources			Bit-file	Recon.
granul.	Slices	DSP	BRAM	(Bytes)	latency
Fine	160	0	0	11 664	$\sim 29\,\mu$s
Medium	2 560	64	32	611 712	~ 1.5 ms
Coarse	12 840	256	140	3 074 112	~ 7.6 ms

The reconfiguration is performed by the ICAP controller through the ICAP interface. For the Xilinx Virtex-6 FPGA, the ICAP interface can be configured to have a data width of 1, 2, or 4 bytes and an operation frequency up to 100 MHz. The time to perform a DPR depends on the amount of data to be transferred over the ICAP interface and a constant time for set-up and activation of the DPR. Table I shows the bit-file size and the reconfiguration latency for the three levels of granularity, assuming the widest possible interface (4 bytes) and the fastest possible clock (100 MHz). The hardware resources for the medium-grain DPR shown in Table I are enough to implement a processing node of the T-CREST platform containing the Patmos processor.

The utilization of fine-grain reconfiguration may deliver significant theoretical speed-up, because small hardware modifications may allow the hardware to be adapted to the application needs very rapidly, which can be used at task level instead of at mode level.

C. Hardware Platform with DPR Support

In a multi-core platform with DPR support, multiple independent reconfigurable regions dedicated to different processors may coexist. For example, every processor node can have a dynamic section used to accelerate the execution of specific tasks. Or one or more dynamic sections can be connected to the message-passing NoC and be accessible (shared) by multiple processors. However, the current FPGA technology allows

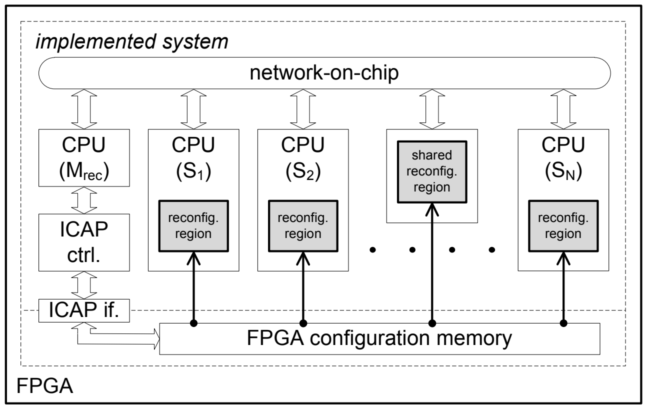

Fig. 5. Block diagram of the FPGA implementation of a multi-core platform with DPR support. The reconfiguration master processor M_{rec} can reconfigure the hardware implemented in the reconfigurable regions of the slaves $\mathrm{S_n}$ and on the shared reconfigurable region by modifying the content of the FPGA reconfiguration memory.

only the reconfiguration of one region at a time. Therefore, our approach to support DPR is to have a single processor of the multi-core platform, called reconfiguration master, which manages the reconfiguration during a mode change.

Figure 5 shows a block diagram of a multi-core platform based on this approach. The platform consists of a reconfiguration master processor $\mathrm{M_{rec}}$ and N slave processors ($\mathrm{S_1}$, $\mathrm{S_2}$, ..., $\mathrm{S_N}$) connected through a message-passing network-on-chip. Each slave processor is provided with a reconfigurable region and a shared reconfigurable region is also connected to the NoC; for example, to the Argo NoC in the T-CREST platform [**?**]. The reconfiguration master is connected to the ICAP interface through the ICAP controller and it can modify the content of the FPGA reconfiguration memory. The arrows in Figure 5 indicate the dependency. Therefore, by writing a partial bit-file into this memory, the hardware system implemented on the FPGA is dynamically modified.

The LogiCORE IP XPS HWICAP [27] is a widely used ICAP controller provided by Xilinx. The detailed architecture of this controller is unknown, and therefore it is not suitable for applications where time-predictability is a strong requirement. Moreover, it implements functionalities that are not required in our approach. For example, it provides a FIFO-based reading functionality that allows a processor to read the content of specific segment of the FPGA configuration memory. Therefore, we are currently developing an ICAP controller that allows loading partial bit-files in a time-predictable manner.

Our ICAP controller is a minimal hardware component that enables a processor to write into the FPGA configuration memory through the ICAP interface. When the master processor starts a reconfiguration, the controller loads a partial bit-file, pre-stored in a dedicated memory, into the FPGA configuration memory in a predictable time interval.

D. Reconfiguration Approach

As previously mentioned, in the T-CREST platform a reconfiguration is associated to an operation mode change.

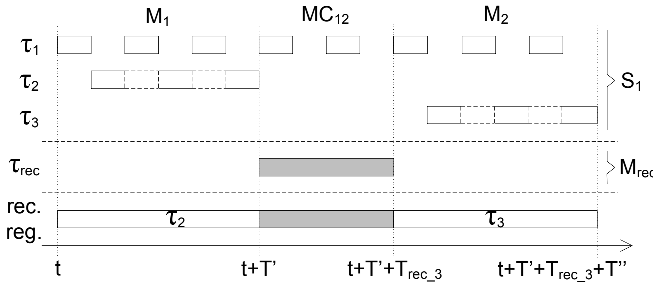

Fig. 6. Time diagram for an example of the mode-level DPR model. The tasks τ_1, τ_2, and τ_3 run on the slave processor $\mathrm{S_1}$. The reconfiguration is performed by the reconfiguration master Mrec. The last row shows how the reconfigurable region is shared.

Every mode consists of a set of tasks and a set of resources, such as co-processors, hardware accelerators, digital signal processors, implemented on the dynamic part of the FPGA.

In hard real-time systems, the usage of hardware accelerators and co-processors leads to a simplification of the WCET analysis and reduces its pessimism (overestimation of the WCET), because analysis of hardware used to implement software-equivalent tasks is often easier to perform than analysis of a pure software solution.

If the reconfiguration is triggered by a mode change that is part of the normal operation of the system, this can be statically scheduled at compile time. If the reconfiguration is in response to external events, the processors that need the reconfiguration can issue a request to the reconfiguration master $\mathrm{M_{rec}}$. In any case, a reconfiguration associated with a mode change can be modeled as a task that belongs to a mode change scenario executed by the reconfiguration master $\mathrm{M_{rec}}$.

As an example, Figure 6 shows a time diagram of the tasks execution during a mode change between the two modes $\mathrm{M_1}$ and $\mathrm{M_2}$. $\mathrm{M_1}$ consists of the tasks τ_1 and τ_2, while $\mathrm{M_2}$ consists of the tasks τ_1 and τ_3. Tasks τ_1, τ_2, and τ_3 run on the slave processor $\mathrm{S_1}$ of the architecture presented in Figure 5 and share its reconfigurable region. We assume that τ_2 and τ_3 need different resources to be implemented on the shared reconfigurable region and that the periodic execution of τ_1 cannot be suspended during the mode change.

In the time diagram of Figure 6, we can observe that during the mode change scenario $\mathrm{MC_{12}}$, task τ_1 continues to run and task τ_{rec}, which performs the reconfiguration process, is executed by the reconfiguration master $\mathrm{M_{rec}}$. The value $\mathrm{T_{rec\_3}}$ is the time that the task τ_{rec} needs to reconfigure the region with the hardware needed by task τ_3. The last row of the diagram shows which task is the user of the reconfigurable region. The solid color represents when the reconfigurable region is under reconfiguration.

The reconfiguration time $\mathrm{T_{rec\_nm}}$ of a generic mode change $\tau_{\mathrm{nm}}, \mathrm{n}, \mathrm{m} \in \mathbb{N}_1$, is the time needed to perform a reconfiguration from a system point of view. Since the reconfiguration takes place in the transition between two modes, which must happen in bounded time, the reconfiguration time must be known to provide execution-time guarantees.

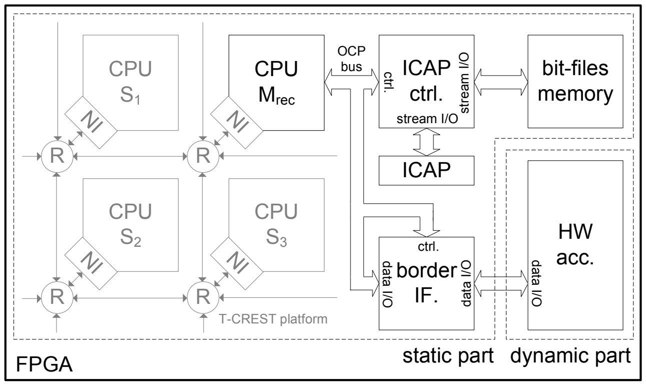

Fig. 7. The 2-by-2 T-CREST platform configuration for the evaluation with medium-grain DPR. The master processor $\mathrm{M_{rec}}$ uses the ICAP controller to manage the reconfigurable region.

In our approach, the reconfiguration is associated to a mode change; this implies that the set of tasks and their computational requirement are known at compile time. Therefore, the exact value of $\mathrm{T_{rec\_nm}}$ can also be calculated at compile time for every allowed mode change.

E. Application Example and Evaluation

The architecture used for this application example is a 2-by-2 T-CREST platform section implemented on the Xilinx Virtex-6 FPGA (ML605 development board). The platform has four processing nodes and a hardware accelerator accessible by the master core $\mathrm{M_{rec}}$, as shown in Figure 7. The ICAP controller is connected to $\mathrm{M_{rec}}$ and to a bit-file memory. The border interface, which is also driven by $\mathrm{M_{rec}}$, consists of registers with enable. The hardware accelerator is a double-precision floating-point unit, generated with FloPoCo [28], that performs addition and multiplication.

For this example, we focus only on the non-gray elements in Figure 7, since the test application using the hardware accelerator is also executed by the reconfiguration master $\mathrm{M_{rec}}$.

The test application has two modes of operation, $\mathrm{M_1}$ and $\mathrm{M_2}$. $\mathrm{M_1}$ consists of a series of N double-precision floating-point additions, and $\mathrm{M_2}$ consists of a series of M double-precision floating-point multiplications. These operations are performed by the hardware accelerator. The application starts in the mode $\mathrm{M_1}$ and executes the additions series. When finished, it changes to mode $\mathrm{M_2}$ for the multiplications; then it repeats to mode $\mathrm{M_1}$.

This simple application emulates a behavior that can be found in real-case applications. For example, a matrix product consists of a series of additions, followed by a series of multiplications; the same applies to the calculation of fast Fourier transforms, where the transformed array is calculated with a series of additions and multiplications, followed by a series of divisions for normalization.

The evaluation compares two test-cases, $\mathrm{TC_{std}}$ and $\mathrm{TC_{dpr}}$, in terms of hardware resource utilization and execution time. The $\mathrm{TC_{std}}$ test-case executes the test application on a platform that does not support partial reconfiguration, while the $\mathrm{TC_{dpr}}$ test-case executes it on a platform that supports reconfiguration. This means that for the first test-case, the adder and the multiplier need to be both simultaneously implemented on the FPGA. For the second test-case, the processor needs to reconfigure the dynamic region during a mode change, alternatively implementing the adder or the multiplier.

TABLE II
HARDWARE RESOURCE UTILIZATION FOR THE XILINX VIRTEX-6 FPGA IMPLEMENTATION OF THE TWO TEST CASES $\mathrm{TC_{std}}$ AND $\mathrm{TC_{dpr}}$.

Entity	Slices	DSP	BRAM
Patmos processor	2459	4	24
FP Adder	523	0	0
FP Multiplier	449	12	0
OCP/FPU adapter	197	0	0
$\mathrm{TC_{std}}$ total	1 169	12	0
ICAP ctrl.	32	0	0
Border logic	20	0	0
Reconfig. region	532	12	0
$\mathrm{TC_{dpr}}$ total	584	12	0

TABLE III
EXECUTION TIMES, EXPRESSED IN CLOCK CYCLES, USED FOR THE EVALUATION FOR THE TWO TEST-CASES AND THEIR RATIO FOR DIFFERENT VALUES OF N.

N	$\mathrm{T_{exe\_std}}$	$\mathrm{T_{exe\_dpr}}$	$\mathrm{T_{exe\_std}}$ / $\mathrm{T_{exe\_dpr}}$
100	4 209	49 909	0.084
1 000	42 009	87 709	0.479
10 000	4 200 009	4 245 709	0.989

Table II shows the FPGA hardware resource utilization in terms of Virtex-6 slices, DSP, BRAM for a Patmos processor and for the entities of the two test cases. The bit-file memory is not included in the comparison, since it is possible to store the bit-files in off-chip memory, without affecting the hardware resource utilization.

The results in Table II show that the total resources needed to implement the reconfigurable region and the reconfiguration infrastructure for the $\mathrm{TC_{dpr}}$ are roughly the 50% of the resources needed to implement the static hardware accelerator for the case $\mathrm{TC_{std}}$. This hardware reduction comes at the cost of having to perform the reconfiguration between modes, which slows down the program execution. The size of the partial bit-files for the $\mathrm{TC_{dpr}}$ test-case are 182 764 bytes each.

In terms of execution times, we have measured the execution times of a section of the test application, for varying numbers of iterations of N. The application section considered for the execution time measurement includes the mode $\mathrm{M_1}$ and the mode-change scenario. For the case $\mathrm{TC_{std}}$, we have measured the execution time $\mathrm{T_{exe\_dpr}}$ of the mode $\mathrm{M_1}$ for the $\mathrm{TC_{dpr}}$ test-case. Analogously, for the case $\mathrm{TC_{dpr}}$, we have measured the execution time $\mathrm{T_{exe\_std}}$ of the mode $\mathrm{M_1}$ that takes into account also the reconfiguration time overhead, by including the duration of the mode change scenario.

Table III shows the execution times, expressed in clock cycles, of a section of the test application, for different values of N. It is possible to observe that for N greater than 10^5, the ratio becomes very close to 1 (0.989). This means that if the execution time of a mode is long enough, the reconfiguration time overhead in a mode change is negligible with respect to the duration of the mode itself.

VI. Conclusion

In this paper we have provided an overview of run-time reconfiguration in hard real-time systems, especially focusing on the T-CREST platform. We have explored the usage of run-time reconfiguration associated to mode changes of real-time multi-mode applications that have different computation and communications requirements for different modes. Our approach to reconfiguration allows the hardware platform and the software infrastructure to reconfigure the provided resources at run-time in order to support mode changes.

The reconfiguration of the inter-process communication is achieved through the reconfiguration of the message-passing NoC Argo by modifying the bandwidth and latency of the communication channels between the cores of the platform. The reconfiguration of computation resources, such as co-processors and hardware accelerators, is achieved using DPR. We have discussed the challenges related to the use of DPR, and we have shown its usage with an application example of the reconfigurable features using the T-CREST multi-core platform on the Xilinx Virtex-6 FPGA. We have shown that if the reconfiguration time overhead in a mode change is negligible with respect to the duration of the mode itself, the usage of DPR can lead to a more efficient usage of the FPGA resource, while maintaining comparable computational performance.

References

[1] L. Wang and F. Y. Wu, "Dynamic partial reconfiguration in FPGAs," in *Proc. of IEEE Third International Symposium on Intelligent Information Technology Application*, vol. 2, 2009, pp. 445–448.

[2] XILINX, "UG702: Partial reconfiguration user guide." Tech. Rep., 2012, online.

[3] ALTERA Corporation, "QII51026: Design planning for partial reconfiguration." Tech. Rep., 2013, online.

[4] K. Compton and S. Hauck, "Reconfigurable computing: A survey of systems and software," *Acm Computing Surveys, Acm Comput Surv, Acm C Surv, Acm Computing Surveys the Survey and Tutorial Journal of the Acm, Acm Comput. Surv*, vol. 34, no. 2, pp. 171–210, 2002.

[5] M. Schoeberl, S. Abbaspour, B. Akesson, N. Audsley, R. Capasso, J. Garside, K. Goossens, S. Goossens, S. Hansen, R. Heckmann, S. Hepp, B. Huber, A. Jordan, E. Kasapaki, J. Knoop, Y. Li, D. Prokesch, W. Puffitsch, P. Puschner, A. Rocha, C. Silva, J. Sparsø, and A. Tocchi, "T-CREST: Time-predictable multi-core architecture for embedded systems," *Journal of Systems Architecture*, vol. 61, no. 9, pp. 449–471, 2015.

[6] A. Burns and A. Wellings, *Real-Time Systems and Programming Languages: Ada 95, Real-Time Java and Real-Time POSIX*. Addison-Wesley, 2001.

[7] B. Dupont de Dinechin, Y. Durand, D. van Amstel, and A. Ghiti, "Guaranteed services of the NoC of a manycore processor," in *Proc. Intl. Workshop on Network on Chip Architectures (NoCArc)*. New York, NY, USA: ACM, Dec. 2014, pp. 11–16.

[8] R. L. Cruz, "A calculus for network delay. I. Network elements in isolation," *IEEE Transactions on Information Theory*, vol. 37, no. 1, pp. 114–131, Jan 1991.

[9] K. Goossens and A. Hansson, "The aethereal network on chip after ten years: Goals, evolution, lessons, and future," in *Proc. ACM/IEEE Design Automation Conference (DAC)*, Jun. 2010, pp. 306 –311.

[10] R. A. Stefan, A. Molnos, and K. Goossens, "dAElite: A TDM NoC Supporting QoS, Multicast, and Fast Connection Set-Up," *IEEE Transactions on Computers*, vol. 63, no. 3, pp. 583–594, 2014.

[11] E. Kasapaki, M. Schoeberl, R. B. Sørensen, C. T. Müller, K. Goossens, and J. Sparsø, "Argo: A Real-Time Network-on-Chip Architecture with an Efficient GALS Implementation," *IEEE Transactions on VLSI Systems*, vol. 24, no. 2, pp. 479–492, 2016.

[12] D. Koch, C. Beckhoff, and J. Teich, "ReCoBus-Builder - a novel tool and technique to build statically and dynamically reconfigurable systems for FPGAs," in *Proc. of International Conference on Field Programmable Logic and Applications*. IEEE, 2008, pp. 4 629 918, 119–124.

[13] R. Cattaneo, R. Bellini, G. Durelli, C. Pilato, M. D. Santambrogio, and D. Sciuto, "Para-sched: A reconfiguration-aware scheduler for reconfigurable architectures," in *Parallel Distributed Processing Symposium Workshops, 2014 IEEE International*, May 2014, pp. 243–250.

[14] K. Wu and J. Madsen, "Reconfigurable architectures: From physical implementation to dynamic behavoir modelling," Ph.D. dissertation, 1 2008.

[15] S. Banerjee, E. Bozorgzadeh, and N. D. Dutt, "Integrating physical constraints in hw-sw partitioning for architectures with partial dynamic reconfiguration," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 14, no. 11, pp. 1189–1202, Nov 2006.

[16] J. Garside and N. C. Audsley, "Investigating shared memory tree prefetching within multimedia noc architectures," in *Memory Architecture and Organisation Workshop*, 2013.

[17] M. Schoeberl, P. Schleuniger, W. Puffitsch, F. Brandner, C. W. Probst, S. Karlsson, and T. Thorn, "Towards a time-predictable dual-issue microprocessor: The Patmos approach," in *First Workshop on Bringing Theory to Practice: Predictability and Performance in Embedded Systems (PPES 2011)*, Grenoble, France, March 2011, pp. 11–20.

[18] M. Schoeberl, D. V. Chong, W. Puffitsch, and J. Sparsø, "A time-predictable memory network-on-chip," in *Proceedings of the 14th International Workshop on Worst-Case Execution Time Analysis (WCET 2014)*, Madrid, Spain, July 2014, pp. 53–62.

[19] M. D. Gomony, B. Akesson, and K. Goossens, "Architecture and optimal configuration of a real-time multi-channel memory controller," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, 2013, pp. 1307–1312.

[20] P. Puschner, R. Kirner, B. Huber, and D. Prokesch, "Compiling for time predictability," in *Computer Safety, Reliability, and Security*, ser. Lecture Notes in Computer Science, F. Ortmeier and P. Daniel, Eds. Springer Berlin / Heidelberg, 2012, vol. 7613, pp. 382–391.

[21] R. Heckmann and C. Ferdinand, "Worst-case execution time prediction by static program analysis," AbsInt Angewandte Informatik GmbH, Tech. Rep., (last accessed: April 2016).

[22] R. B. Sørensen, J. Sparsø, M. R. Pedersen, and J. Højgaard, "A Metaheuristic Scheduler for Time Division Multiplexed Networks-on-Chip," in *Proc. IEEE/IFIP Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (SEUS)*, 2014, pp. 309–316.

[23] J. Sparsø, E. Kasapaki, and M. Schoeberl, "An Area-efficient Network Interface for a TDM-based Network-on-Chip," in *Proc. Design, Automation and Test in Europe (DATE)*, 2013, pp. 1044–1047.

[24] I. Kotleas, "Mode changes in network-on-chip based multiprocessor platforms," Master's thesis, 2014.

[25] R. B. Sørensen, L. Pezzarossa, and J. Sparsø, "An area-efficient tdm noc supporting reconfiguration for mode changes," in *Accepted at the 10th IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, 2016.

[26] XILINX, "UG360: Virtex-6 FPGA configuration user guide." Tech. Rep., 2015, online.

[27] ——, "DS586: LogiCORE IP XPS HWICAP (v5.00a) product specifications." Tech. Rep., 2010, online.

[28] F. De Dinechin and B. Pasca, "Designing custom arithmetic data paths with FloPoCo," *IEEE Design and Test of Computers*, vol. 28, no. 4, pp. 18–27, 2011.