A Controller for Dynamic Partial Reconfiguration in FPGA-based Real-Time Systems

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 real-time systems, the use of hardware accelerators can lead to a worst-case execution-time speed-up, to a simplification of its analysis, and to a reduction of its pessimism. When using FPGA technology, dynamic partial reconfiguration (DPR) can be used to minimize the area, by only loading those accelerators that are needed at any given point in time. The DPR controllers provided by the FPGA vendors satisfy a wide range of requirements and rely on software to manage the reconfiguration. This approach may lead to slow reconfiguration and unpredictable timing. This paper presents an open-source DPR controller specially developed for hard real-time systems and prototyped in connection with the open-source multi-core platform for real-time applications T-CREST. The controller enables a processor to perform reconfiguration in a time-predictable manner and supports different operating modes. The paper also presents a software tool for bitstream conversion, compression, and for reconfiguration time analysis. The DPR controller is evaluated in terms of hardware cost, operating frequency, speed, and bitstream compression ratio vs. reconfiguration time trade-off. A simple application example is also presented with the scope of showing the reconfiguration features of the controller.**

I. Introduction

In general purpose systems, the main benefit gained from using reconfiguration is the speed-up of the average-case execution time obtained with the hardware acceleration of specific tasks of an application. In real-time systems, such average-case speed-up is not in itself relevant, since it is the worst-case execution time (WCET) of tasks that determines the ability of the system to respond in time. However, the use of hardware accelerators and co-processors may lead to a reduction of the WCET, simplification of the WCET analysis, and a reduction of its pessimism. In [1], we discussed and briefly explored the idea of using dynamic partial reconfiguration in the context of operational mode changes in hard real-time systems. This paper provides more details on the required hardware infrastructure and it includes a comprehensive evaluation of the cost and performance of the approach.

Dynamic partial reconfiguration (DPR) is typically supported by controllers provided by the FPGA vendors. These controllers target general-purpose architectures and are therefore developed to satisfy a wide range of requirements leading to large hardware implementations. Moreover, these controllers rely on software libraries and processor support to manage the reconfiguration, resulting in slow reconfiguration and to unpredictable timing. This is not suitable for hard real-time systems for embedded applications, which typically requires predictable hardware/software interfacing. In addition, a smaller hardware footprint is attractive.

This paper presents a lightweight real-time DPR controller, called RT-ICAP, to be used with the internal configuration access port (ICAP) of Xilinx FPGAs. The controller is time-predictable and has a low hardware cost. It enables a processor to write into the FPGA configuration memory through the ICAP interface in two different operating modes: (1) one driven by a host CPU and (2) one driven by the RT-ICAP controller itself. In addition, our RT-ICAP controller supports run-length encoding to compress the bitstreams, in addition to the default compression provided by the FPGA vendors' tools. This leads to interesting trade-off between compression and reconfiguration time, discussed in the evaluation section.

The paper also presents the software tool needed to translate the partial bitstreams produced by the FPGA vendors' tools into a format that is compatible with the controller and to perform the reconfiguration time analysis. The architecture is evaluated and compared with other controllers in terms of hardware cost, operating frequency, reconfiguration speed, bitstream compression ratio, and worst-case reconfiguration time. An application example is also presented aiming to show the reconfiguration features of the controller, how DPR can lead to a more efficient usage of the FPGA resource, also providing WCET numbers. This architecture is open-source and it is prototyped using the multi-core platform for real-time applications T-CREST [2]. However, it can be easily used by other time-predictable architectures (including FPGA-based acceleration devices) where a small hardware footprint of the controller and time-predictable reconfiguration are strong requirements.

This paper has two main contributions: (1) it presents the hardware architecture of a lightweight time-predictable reconfiguration controller and (2) it presents a software tool for bitstream manipulation and reconfiguration time analysis, to support the controller.

This paper is organized in six sections: Section II presents related work on controllers and infrastructures used for dynamically reconfigurable systems. Section III gives a brief background on T-CREST, on DPR, and on reconfiguration in real-time systems. Section IV presents the hardware/software infrastructure developed to support reconfiguration. Section V present the evaluation of this infrastructure. Finally, section VI concludes the paper.

II. Related Work

The XPS_HWICAP reconfiguration controller [3], provided by Xilinx, is an IP core designed to be connected to the processor local bus [4] and it provides the support for DPR using a set of software functions provided in processor-specific libraries. The AXI_HWICAP controller [5] by Xilinx delivers the same functionality as the XPS_HWICAP controller, but it can be interfaced to the AXI4-Lite bus [6]. The associated software library, (currently provided for the MicroBlaze [7] and the PowerPC [8] processors) allows an application programmer to write and read configuration bitstreams, and it enables the modification of single look-up tables and flip-flop properties.

Other reconfiguration controllers managed by the software running on a processor are presented in [9]. The work presents and investigates the performance of three controller architectures, named MST_HWICAP, DMA_HWICAP, and BRAM_HWICAP, strongly inspired by the Xilinx XPS_HWICAP controller, and therefore very similar in terms of functionality.

The ZyCAP controller [10] is a custom controller for processor/FPGA hybrid platforms, such as the Xilinx Zynq. The ZyCAP controller is connected to the hardcore processor through the AXI4-Lite bus and to the system memory through the AXI4 bus [6] (where the bitstreams are stored). A direct memory access controller loads the bitstream during reconfiguration using the ICAP interface. Software drivers are associated with the controller and allow the hardcore processor to manage the reconfiguration process.

For all the above mentioned controllers, most of the functionality is provided by software executing on a processor that reads from or writes to the controller interface. These frequent accesses to the controller through the system bus affect the reconfiguration speed and may increase the WCET pessimism, since I/O functions may be difficult to analyze. Moreover, for the Xilinx controllers, the source code is not provided and only netlists are available, making it impossible to perform WCET analysis. In our solution, we aim to minimize the interaction between the processor and the DPR controller, in order to increase time predictability.

Another class of controllers reconfigure in an autonomous fashion, somewhat similar to a direct memory access controller. The PRC controller [11], provided by Xilinx, is an IP core designed to independently manage DPR in reconfigurable designs targeting the Xilinx 7 series FPGAs. The controller is interfaced to a processor through the AXI4-Lite bus [6]. When it receives a software or hardware trigger, it can independently manage the reconfiguration of multiple regions by reading bitstreams from a memory connected to the AXI4-Lite bus and writing these into the ICAP interface. Also for this Xilinx controller, only the netlist is available.

The DPRM controller presented in [12] and the ICAP-I controller presented in [13] offer similar functionality for Xilinx Virtex-5 and Virtex-4 FPGAs, respectively. The DPRM controller supports only bitstream transfers from off-chip flash memories into the FPGA configuration memory, while the ICAP-I controller supports also transfer of bitstreams stored in on-chip BRAMs. The architecture of these two controllers and the BRAM_HWICAP are the ones that most resemble the architecture of our RT-ICAP controller. However, our architecture can support a tighter interaction with a processor, as explained in Subsection IV-B.

The D^{2}PR controller presented in [14] is an example of a minimal custom DPR controller connected to the ICAP interface. The controller can be configured to include circuitry for configurable error detection and correction, aiming to improve safety in DPR by monitoring for data errors in the partial bitstreams. Our controller relies on the default checksum-based bitstream checking already supported by Xilinx FPGAs.

III. Background

This section provides an overview of the T-CREST multi-core platform, background information regarding DPR of FPGAs, and it describes our approach to reconfiguration in real-time systems.

A. The T-CREST Multi-Core Platform

The T-CREST [2] multi-core platform has been developed specifically for use in hard real-time applications. All components have been designed with a focus on time-predictability and with a focus on reducing the complexity and pessimism of the WCET analysis. The platform consists of a number of processing nodes and two networks-on-chip (NoCs): (1) one NoC for message passing traffic between cores and (2) one NoC for traffic between cores and the shared memory.

A processing node consists of a time predictable, dual-issue RISC processor, called Patmos, that is optimized for real-time systems [15]. Patmos contains special instruction and data caches, and local private scratchpad memories (SPMs) for instructions and data. Patmos is supported by a compiler, also developed with a focus on WCET [16] and by the WCET analysis tools aiT [17] from AbsInt and platin [18], which allows static derivation of tight WCET bounds.

The Argo [19] NoC provides message-passing to support inter-processor communication and offers the possibility to set up virtual point-to-point circuits between processor nodes. Data is pushed across these circuits by direct memory access controllers in the source end of the circuit.

For code and larger data structures all processors are connected by a NoC in tree form to the memory controller and then to the shared external main memory. For the memory tree there exist two solutions: (1) the Bluetree [20] memory tree with the Predator memory controller [21] for DRAM memory and (2) a distributed memory arbiter [22] with a memory controller for SRAM memory.

The DPR controller presented in this paper is specifically designed for real-time systems and targets this platform, supplementing it with time predictable reconfiguration capabilities using DPR.

B. Dynamic Partial Reconfiguration

DPR is a feature of modern FPGAs that allows the modification of an operating FPGA [23]. Partial bitstreams can be loaded into the FPGA to reconfigure selected regions, without compromising the functionality of other parts of the device. A system that uses DPR can be conceptually divided into a static part and a dynamic part. The static part is configured only once at boot-time with a full bitstream. The dynamic part can be reconfigured multiple times during run-time with different partial bitstreams and it may consist of several independent reconfigurable regions.

When a bitstream is loaded into the FPGA, it is stored in a static RAM called configuration memory. The content of this memory defines the hardware configuration implemented in the FPGA. Therefore, by writing a partial bitstream into the reconfiguration memory, the hardware system implemented in the FPGA is dynamically modified.

For Xilinx FPGAs, DPR can be performed by loading a partial bitstream, at runtime, through one of the FPGA configuration interfaces. In this work, we use the ICAP interface [24]. The ICAP interface is a hardware primitive in Xilinx FPGAs that provides access to the FPGA configuration memory. It allows the user to access configuration registers, read-back configuration data, and to partially reconfigure the FPGA after its initial configuration.

The ICAP interface has separate read and write buses and it can be configured to support a data width of 8, 16, or 32 bits. It has a maximum operation frequency of 100 MHz. Therefore, it can stream at a maximum speed of 400 MB/s. During reconfiguration, the ICAP interface provides information about the current state of the reconfiguration and communicates when the reconfigurable region is successfully reconfigured through an output port. Specific ICAP timing diagrams and bitstream ordering information are provided in [24].

C. Reconfiguration in Real-Time Systems

In the T-CREST platform [2], reconfiguration is associated to an operation mode change. A mode change involves switching from executing one task graph to executing another task graph with different communication and computation guaranteed-services requirements.

The idea is to use DPR to dynamically adapt the hardware platform to the actual needs of a specific mode of operation. This minimizes the hardware resources utilization, by only loading those resources (such as co-processors, hardware accelerators, digital signal processors) that are needed at any given point in time. Moreover, execution time analysis of hardware used to implement software-equivalent tasks is often easier to perform than analysis of a pure software solution. Moving functionality from software into hardware can lead to a simplification of the WCET analysis and therefore to a reduction of its pessimism. This directly translates into a speed-up in the WCET of the tasks executed in hardware.

Figure 1 shows a block diagram of the FPGA implementation of the T-CREST platform with support for DPR [1]. 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. In this example, each slave processor is provided with a reconfigurable region. In addition, a shared reconfigurable region is also directly connected to the Argo NoC of the T-CREST platform [25]. Custom configurations are also possible. The reconfiguration master $\mathrm{M_{rec}}$ is connected to the ICAP interface through our RT-ICAP controller and it can reconfigure the hardware implemented in the reconfigurable regions of the slaves and in the shared reconfigurable region. In the following sections we focus on the architecture of this controller and the associated tool.

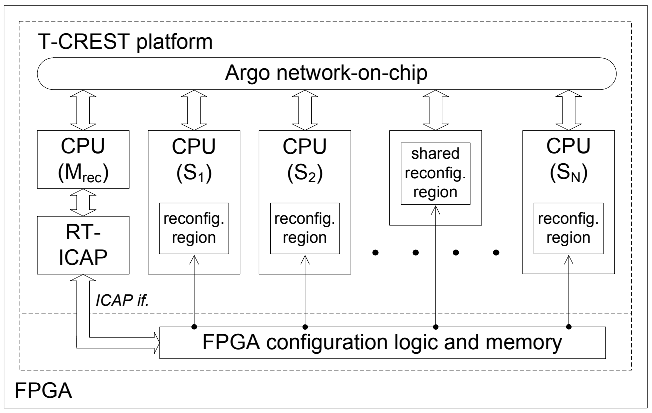

Fig. 1. Block diagram of the FPGA implementation of the T-CREST platform with DPR support. The reconfiguration master M_{rec} is connected to the ICAP interface through our RT-ICAP controller. Slave processors are provided with a reconfigurable region and a shared reconfigurable region is also directly connected to the Argo NoC.

IV. The Reconfiguration Controller

This section provides a description of the architecture and the functionality of our RT-ICAP reconfiguration controller. Moreover, it presents the bitstream compression technique, the software tool associated with the controller, and a hardware-level reconfiguration time analysis.

A. Overview

Some of the controllers presented in Section II offer a range of functionalities that are not strictly required by our approach to support reconfiguration in the T-CREST platform, where DPR is used to switch hardware accelerators and co-processor during an operational mode change [1]. Examples of these functions are the read-back, which is typically used for FPGA scrubbing purposes, or the LUT-based reconfiguration used to speed up DPR for specific applications (such as replacement of FIR-filter coefficients).

These additional functionalities increase the hardware complexity and therefore the complexity of the hardware-level reconfiguration time analysis. Moreover, for some of the architectures, the detailed hardware and software architecture of these controllers is not public, making it impossible to perform a precise and reliable WCET analysis.

Our controller is open-source and intended to be used in the T-CREST platform. It is specifically designed to assist processor-initiated partial reconfiguration of processing

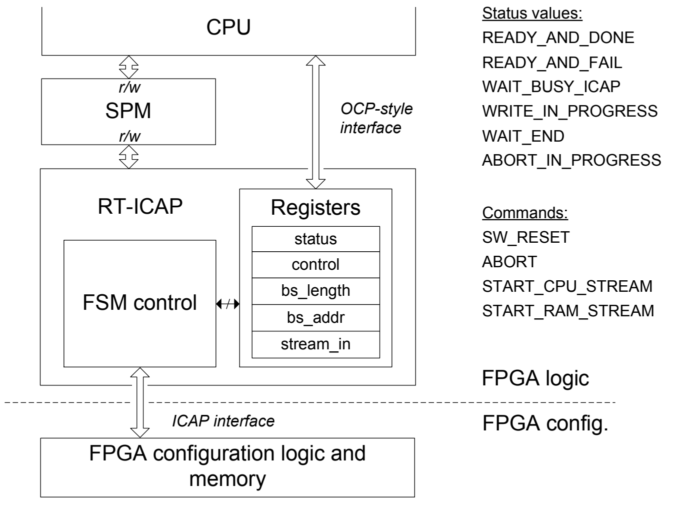

Fig. 2. A block diagram of the RT-ICAP reconfiguration controller and its interfaces, with the possible *status* register values and *control* register commands.

resources such as hardware accelerators. The controller uses run-length encoded bitstream decompression to minimize the size of the bitstream files and it does not support read-back. This lightweight and simple design makes DPR in T-CREST and the timing analysis straightforward and easy, and it results in a small hardware implementation. The controller is implemented in VHDL and it supports the Virtex-4, -5, -6, and the 7-series FPGAs from Xilinx.

B. Architecture and Functionality

Figure 2 shows a block diagram of our reconfiguration controller. The controller is connected to a processor through an OCP interface [26], to a scratchpad memory (SPM), and to the ICAP interface. A SPM is an on-chip memory that is private to a processor. To use the SPM, the programmer/compiler must allocate data structures in the SPM. In our architecture, the SPM is used to store the reconfiguration bitstreams and also acts as a local general purpose memory for the processor. In comparison to a data cache, the access time for an SPM is guaranteed to be a single cycle. This is one further element that distinguishes our RT-ICAP controller from the ones discussed in Section II.

Our RT-ICAP controller is interfaced with the processor through a set of 32-bit registers. The *status* register can be read by the processor to monitor the controller/reconfiguration status. The *control* register can be written by the processor to manage the controller. The possible *status* register values and the *control* register commands are listed in Figure 2. The functionality of the other registers is explained below.

Our RT-ICAP controller can operate in two different modes: *SPM-stream* and *CPU-stream* mode. When the controller operates in *SPM-stream* mode, it autonomously fetches the bitstream from the SPM. In order to start a reconfiguration, the bitstream is stored in the SPM and the processor configures the *bs_addr* register with the SPM address that points at the beginning of the bitstream and the *bs_length* register with the length (in bytes) of the bitstream. By writing the *START_SPM_STREAM* command into the *control* register, the processor starts the transfer from the SPM to the ICAP. The *status* register reports the controller status, including the end of the reconfiguration process. With this operating mode, it is possible to achieve the maximum transfer speed of the ICAP interface, but it requires that the bitstream fits into the SPM. When possible, for example if a reconfiguration is scheduled to happen at a certain point in time, bitstream pre-fetching from main memory (on-chip or off-chip) to the SPM can be performed by the reconfiguration master to minimize the reconfiguration time during a mode change. Therefore, it should be used for small bitstreams associated to reconfigurable regions that require a fast reconfiguration.

When the controller operates in *CPU-stream* mode, it receives the bitstream from the processor as a sequence of writes. In order to start a reconfiguration, the processor must first configure the *bs_length* register and write the *START_CPU_STREAM* command into the *control* register. Then, the bitstream is written into the *stream_in* register of the controller. The CPU reads the bitstream from some on-chip or off-chip memory. This operating mode is slower than the maximum speed of the ICAP controller and the bottleneck is typically determined by the bandwidth to the (off-chip) memory from where the processor reads the bitstream. This mode of operation should be used when it is not possible to store the bitstream into the SPM (e.g., in case of an unexpected mode change triggered by an aperiodic event). The advantage is that it avoids potentially large SPMs.

C. Bitstream Compression

The size of the SPM is a limiting factor of our approach in *SPM-stream* operating mode. To overcome this limitation, lossless compression techniques can be used to decrease the size of a partial bitstream and therefore the memory needed for storage. The application of the most common compression techniques (e.g. run-length encoding, Huffman, Arithmetic, Lempel-Ziv, etc.) on Xilinx bitstreams is a well-known topic and it is explored in the work presented in [27], [28] and [29].

In our implementation, we have selected a simple run-length encoding (RLE) compression technique in order not to affect the reconfiguration speed and not to increase the hardware cost of performing decompression. Moreover, implementing the RLE decompression in hardware instead of a software task executed by the reconfiguration master processor contributes to the reduction of the complexity of the WCET analysis. The data element size used for the compression is the same as the data size of the ICAP interface. In the following, we refer to data element of this size as ‘character’. The idea is to store sequences of repeated characters (data run) as a single character and a count identified by an escape character. In the bitstream, when a data run longer than three elements gets compressed, it appears as an escape value to signal the beginning of a compressed sequence, followed by the count and the data itself. A single escape value is represented in the compressed bitstream as replicated character to distinguish it from a compressed

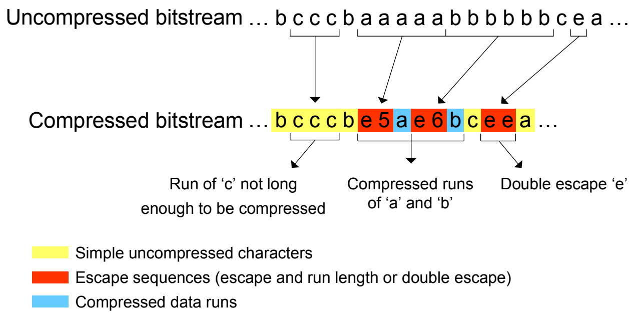
Uncompressed bitstream … b c c c b a a a a a b b b b b c e a …
Compressed bitstream … b c c c b e 5 a e 6 b c e e a …
Run of 'c' not long enough to be compressed
Compressed runs of 'a' and 'b'
Double escape 'e'
Simple uncompressed characters
Escape sequences (escape and run length or double escape)
Compressed data runs

Fig. 3. An example of the RLE compression on a sequence of characters. In this example, the escape character is 'e'.

sequence. Figure 3 shows an example of RLE compression on a sequence of characters. The escape character is 'e'. The bitstream is compressed by the software tool presented in the following subsection, and it is decompressed in hardware by the controller.

The tools provided by the FPGA vendors offer a bitstream compression functionality based on writing identical configuration frames once, instead of writing each frame individually. A frame is the smallest addressable segment of the FPGA configuration memory space and its size is in the order of KB, depending on the FPGA model (e.g., 5.7 KB for the Virtex-6 FPGA). If more than one frame has identical data, the frame is loaded into the configuration logic and written to multiple address locations in parallel. In this case, the decompression is executed by the FPGA logic that manages the reconfiguration (not accessible by the user). The RLE technique can be used stand-alone or in addition to the frame-based compression offered by the Xilinx tools. The trade-off between compression ratio and reconfiguration time is discussed and evaluated in Subsection V-B.

D. Tool Support and Timing Analysis

The hardware architecture is supported by a software tool, named *convbitstream*. This tool performs two tasks: (1) it compresses the partial bitstreams produced by the Xilinx tools and converts them to the format required by our RT-ICAP controller, and (2) for each compressed bitstream it computes the time that the RT-ICAP controller takes to perform the reconfiguration. The time is computed for both the *SPM-stream* and *CPU-stream* mode and explained in detail below.

Similarly to the other tools of the T-CREST platform [2], *convbitstream* receives as input an XML file specifying parameters that control the compression and format conversion as well as a path to a directory containing the bitstream files to be processed. For each bitstream provided as input, the tool produces the converted bitstream in two formats: as a binary file, to be used if the bitstreams are stored in an off-chip memory, and as an array declaration in a C file containing all the bitstreams in the form of arrays, to be used to embed a bitstream into a C program. The operations that are performed on the bitstream are bit-swapping [24], RLE compression, and translation into an RT-ICAP compatible format.

For each compressed bitstream the *convbitstream* tool computes the time it takes to perform the corresponding partial reconfiguration. The reconfiguration time T_{rec} is from the moment when the processor (or a master device) starts the reconfiguration and until the partial bitstream is completely written into the FPGAs configuration memory. This reconfiguration time is needed during WCET analysis of an application that uses the reconfiguration feature, and it depends on properties of the RT-ICAP controller and on properties of the compressed bitstream.

The reconfiguration time is computed using Eq. (1) and it is the period of the clock signal, T_{clk} times the number of clock cycles as a sum of three contributions: the number of cycles needed to invoke (initialize and finalize) a reconfiguration, the number of cycles needed to transfer the compressed bitstream into the RT-ICAP controller and the number of additional cycles required to expand compressed data runs and write these into the reconfigurable area of the FPGA.

$$T_{rec} = T_{clk}\{n_{oh} + n_{if}n_s + \sum_{i=1}^{n_r}(R_{i\_len} - 1)\} \tag{1}$$

- n_{oh} is a small overhead required for starting and finishing a reconfiguration; for our RT-ICAP controller this is 3 cycles.
- n_{if} is the number of cycles to write a character into the RT-ICAP controller; for *SPM-stream* mode this is 1 cycle and for *CPU-stream* mode it is 2 clock cycles (the time for an OCP-transaction [26]). In parallel, a character is written into the reconfigurable area of the FPGA in one clock cycle.
- n_s is the length of the compressed bitstream, which includes the number of simple (uncompressed) characters, the number of escape sequences (each comprising two characters), and the number of compressed data runs. In the example shown in Figure 3, these elements are marked yellow, red and blue respectively.
- The third contribution is the number of additional clock cycles required for writing repeating characters into the reconfigurable area of the FPGA: n_r is the number of compressed data runs and R_{i_len} is the number of times a compressed character repeats in the i-th data run.

The parameters n_{oh} and n_{if} characterize the RT-ICAP controller and n_s, n_r, and R_{i_len} characterize the bitstream.

When the controller operates in *CPU-stream* mode, the bitstream is loaded from main memory or other form of external storage. The time needed for this is independent of the RT-ICAP controller and needs to be analyzed separately with a relevant tool. In T-CREST, the WCET analysis tools aiT [17] and platin [18] can be used to perform the WCET analysis. This information, together with the reconfiguration time provided by our analysis, is needed for the application-level WCET analysis that uses the reconfiguration feature.

V. Evaluation

This section evaluates the proposed architecture in terms of hardware cost, operating frequency, reconfiguration speed,

TABLE I
CHARACTERIZATION IN TERMS OF HARDWARE RESOURCES, MAXIMUM CLOCK FREQUENCY, AND RECONFIGURATION SPEED OF OUR RECONFIGURATION CONTROLLER AND COMPARISON WITH RELATED PUBLISHED DESIGNS.

Controller	Target FPGA	Hardware resources			f_{max} (MHz)	Recon. speed (MB/s)
		FF	LUT	BRAM		
RT-ICAP	Kintex-7	101	245	0	>300	382.2[1]
PRC [11]	Kintex-7	1270	1075	0	>100	n/a
ZyCAP [10]	Zynq-7000	806	620	0	>100	382
RT-ICAP	Virtex-6	88	190	0	323	382.2[1]
DPRM [12]	Virtex-6	77	109	0	379	6.6[2]
D^{2}PR [14][3]	Virtex-6	112	249	0	>100	400
XPS_HWICAP [3], [30]	Virtex-5	745	741	3	>100	1.3[2]
AC_ICAP [30]	Virtex-5	1667	1161	7	>100	381.0
ICAP-I [13]	Virtex-4	303	177	0	90	180.0
DMA_HWICAP [9]	Virtex-4	4277	977	0	121	82.1[2]
MST_HWICAP [9]	Virtex-4	1083	918	2	200	234.5
BRAM_HWICAP [9]	Virtex-4	963	469	32[4]	121	332.1

[1] *SPM-stream* mode. [2] Using off-chip memory. [3] Synchronous version - no error check. [4] Including storage.

bitstream compression ratio, and worst-case reconfiguration time. Finally, it presents a simple application example of the reconfiguration features offered by the presented architecture, including WCET estimations, hardware resources, and configuration speed results. All the results of our architecture, presented in this section, were produced using Xilinx Vivado (v16.4) targeting the Xilinx Kintex-7 FPGA (model XC7K325T-2FFG900C) and using Xilinx ISE and PlanAhead (v14.7) targeting the Xilinx Virtex-6 FPGA (model XC6VLX240T-1FFG1156). All the synthesis properties were set to their defaults. The data size of the ICAP interface is 32 bits.

A. Hardware and Performance

Table I presents hardware resources, maximum clock frequency, and reconfiguration speed. Our architecture is compared against some of the controllers presented in the related work and listed in the first column of the table. For these controllers, the results are taken from the respective publication. The second column of the table reports the target FPGA used to produce the results.

Table I shows the FPGA hardware resource usage in terms of flip-flops (FFs), look-up tables (LUTs), and block-RAMs (BRAMs) for our controller and for the other designs. The BRAM used to store the bitstreams needed to produce the hardware results of Table I is not taken into account (except for the BRAM_HWICAP). The hardware results for 7-series, Virtex-6 and -5 FPGAs can be quantitatively compared, since these FPGAs use 6-input LUTs. Virtex-4 FPGAs use 4-input LUTs, therefore the LUT results are only reported for qualitative comparison. We can observe that our controller is comparable in size with the controllers DPRM [12], D^{2}PR [14] (synchronous version without error check), and ICAP-I [13], even if our controller offers more functionality than these, such as support of two operating modes, bitstream compression, and a *status/control* register based interface. Our controller is considerably smaller than the other controllers.

Table I also presents the maximum operating frequency of our controller and of the other designs. In practical applications, the controller typically runs at the same operation frequency as the ICAP interface (max. 100 MHz) in order to avoid clock domain crossing. We can observe that all the controllers are able to meet this constraint, except for the ICAP-I.

Table I shows, in the last column, the reconfiguration speed computed as a ratio between the bitstream size and the reconfiguration time as defined in Subsection IV-D. The operating frequency is assumed to be 100 MHz for all the controllers, except for the ICAP-I (90 MHz). The reconfiguration speed is computed assuming the bitstream stored in an on-chip memory. In the cases where an on-chip memory is not available, we report for qualitative comparison the reconfiguration speed when the bitstreams are stored in an off-chip flash memory (marked with the superscript [2]).

For our controller, the table shows the speed for the *SPM-stream* mode calculated as average of the reconfiguration speed of 6 different RLE-compressed sample bitstreams of size between 13.1 KB and 129.7 KB. We can observe that, for our controller, reconfiguration speed is comparable or faster than the one of the other controllers, except for the D^{2}PR [14] (synchronous version, no error check) which is faster, since it does not offer any further functionality, such as compression and register based interface.

B. Bitstream Compression and Reconfiguration Time

This subsection evaluates the compression capability of the *convbitstream* tool and shows the trade-off between compression and reconfiguration time. Table II presents the compression ratios of three sample bitstreams. The compression ratio is

TABLE II
BITSTREAM COMPRESSION RATIOS USING OUR RLE COMPRESSION, THE XILINX COMPRESSION ONLY, AND OUR RLE COMPRESSION ON TOP OF THE XILINX ONE FOR THREE BITSTREAM WITH DIFFERENT UTILIZATION RATIOS. THE RECONFIGURABLE REGION SIZE IS 880 SLICES AND THE UNCOMPRESSED BITSTREAM SIZE IS 184.8 KB.

Bitstream	Recon. region utilization	Compression ratio				RLE+Xilinx size (KB)
		Ideal	RLE	Xilinx.	RLE+Xilinx	
Mult. & Add.	91 %	4.1	1.4	1.3	1.4	132.5
Adder	55 %	6.4	2.0	1.4	1.9	95.1
Blank	0 %	38.9	14.1	2.2	8.7	21.2

TABLE III
COMPUTED AND MEASURED RECONFIGURATION TIME, EXPRESSED IN CLOCK CYCLES (CC), FOR THE UNCOMPRESSED AND COMPRESSED SAMPLE BITSTREAMS FOR THE *SPM-stream* OPERATING MODE.

Bitstream	Computed (CC)				Measured (CC)
	Uncompr.	RLE	Xilinx	RLE+Xilinx	RLE+Xilinx
Mult. & Add.	47311	48745	37384	38994	38999
Adder	47311	49480	33782	36105	36109
Blank	47311	48655	21175	22527	22529

defined as the size of the uncompressed bitstream divided the by the size of the compressed one.

To produce these results, we used a reconfigurable region size of 880 slices for all the experiments. The uncompressed bitstream size, which depends only on the size of the reconfigurable region, is 184.8 KB. A double-precision floating-point multiplier and adder generated with FloPoCo [31] (2406 FFs, 2524 LUTs, 12 DSPs) is implemented in the reconfigurable region to obtain the utilization value of 91 %. A double-precision floating-point adder (1665 FFs, 1499 LUTs) is used to produce utilization of 55 %. Leaving the reconfigurable region without any hardware implementation produces a blank bitstream with utilization of 0 %.

Table II shows the ideal compression ratio and the ratios obtained by our RLE compression applied on a bitstream, by the Xilinx tools compression only, and by our RLE compression applied on a bitstream already compressed by the Xilinx tools. For the latter, Table II reports also the compressed bitstream size. The ideal compression ratio specifies a theoretical upper bound and it is derived from the 32-bit-based information entropy of the entire bitstream. All the compression ratios are referred to the uncompressed bitstream size.

We can observe that the RLE compression introduces a significant reduction of the bitstream size and that it performs better than the native compression offered by the Xilinx tools. Moreover, it can further decrease the size of the bitstream already compressed by the Xilinx tools because the granularity of our compression is considerably smaller than a frame. However, considering the reconfiguration time, we can observe that a bitstream compressed with both the RLE and the Xilinx native compression has a considerably shorter reconfiguration time (about 50 % on average) than the one compressed with RLE only, since the Xilinx decompression logic writes identical configuration frames concurrently to multiple addresses. Therefore, the combination between the RLE and the Xilinx native compression, even if not as good as the pure RLE in terms of compression ratio, can be considered as the best trade-off between a high compression ratio and a low reconfiguration time.

Table III presents the reconfiguration time computed with the *convbitstream* tool and expressed in clock cycles, for the uncompressed bitstreams and for the bitstreams compressed using the techniques evaluated in Table II, for the *SPM-stream* operating mode.

For the bitstreams compressed with both the RLE and the Xilinx native compression, Table III also reports the measured reconfiguration time. With this measurement we verify our computed WCET of the reconfiguration time. The measurement is executed using the RT-ICAP controller connected to a Patmos processor [15] and implemented on the Xilinx Virtex-6 FPGA. The time interval is measured by the software running on the processor. Since the end of the reconfiguration process is determined by polling the *status* register of the RT-ICAP controller in a loop, a small measurement overhead is observed. This overhead is in the order of few clock cycles and, when executing the WCET analysis of a task that uses reconfiguration, it is taken into account by the analysis tool. The results show that the computed reconfiguration time is correct. By removing the measurement overhead, it is possible to notice that there is no pessimism in the estimation of the total reconfiguration time.

C. Application Example

The scope of this simple application example is to provide a further evaluation of the RT-ICAP controller in the context of an operation mode change. The architecture used for this application example consists of a Patmos processor [15], a

TABLE IV

COMPUTED WCET AND MEASURED EXECUTION TIME T_{exe} FOR THE SW, HW_{std}, AND HW_{dpr} TEST-CASES FOR 10-BY-10 MATRICES, EXPRESSED IN CLOCK CYCLES (CC).

Test-case	WCET (CC)	T_{exe} (CC)
SW	8197134	481661
HW_{std}	107892	104918
HW_{dpr}	206953	203979

hardware accelerator, and the reconfiguration infrastructure presented in this paper used in *SPM-stream* operating mode. The hardware accelerator is a double-precision floating-point unit, generated with FloPoCo [31], that performs addition and multiplication. This example aims to show the reconfiguration features of the controller and how DPR can lead to a more efficient usage of FPGA resources without significantly affecting the computational performances. Moreover, it provides computed WCET numbers and measured execution time for the entire application.

The synthetic test application executes a multiply-accumulate operation $A \leftarrow A + (B * C)$ (very often used in digital signal processing), where the accumulator A and inputs B and C are n-by-n square matrices. The elements of the matrices are double-precision floating-point values. Therefore, in terms of floating-point operations, the application consists of a series of n^3 multiplications followed by a series of n^3 additions.

For this example, we consider the series of multiplications and the series of additions as two modes of operation, M_1 and M_2 respectively. This emulates a behavior that can be found in real-case applications. For instance, in the calculation of fast Fourier transforms, where the transformed array is computed with a series of additions and multiplications, followed by a series of divisions for normalization.

This example compares three test-cases, SW, HW_{std}, and HW_{dpr}, in terms of hardware resource utilization and execution time. The SW test-case executes the test application completely in software using the compiler support libraries for floating-point operations [16]. The HW_{std} and HW_{dpr} test-cases run the application using the addition and multiplication capability offered by the floating-point hardware accelerator. The TC_{std} test-case executes the test application on a platform that does not support DPR. Therefore, the adder and the multiplier need to be both simultaneously implemented on the FPGA. The TC_{dpr} test-case executes the test application on platform that supports reconfiguration, and the reconfigurable region alternatively implements the multiplier during the mode M_1 and the adder during the mode M_2.

Table IV shows the WCET of the test application computed with the WCET analysis tools platin [18] and the measured execution time T_{exe} obtained with the Patmos simulator for the three test cases for 10-by-10 matrices. The WCET for the HW_{dpr} test-case also includes the reconfiguration time computed by the *convbitstream* tool. We can notice that the WCET for the SW test-case is very high compared to the measured result. This is due to two effects: (1) not triggering the worst-case path in the measurement and (2) overestimation within the WCET analysis. The floating-point operations are performed in software and we assume that the worst-case path is in handling of corner cases, such as denormalized numbers. The overestimation comes from the fact that not all floating-point library code fits into the method cache [32] of Patmos. In that case our method cache analysis [33] assumes more misses than actually happen at runtime.

TABLE V

HARDWARE RESOURCES UTILIZATION FOR THE PATMOS PROCESSOR AND FOR THE TWO TEST CASES HW_{std} AND HW_{dpr}.

Entity	Slices	FF	LUT	DSP
Patmos proc.	2693	3693	5468	4
FP Adder	506	1665	1499	0
FP Multiplier	482	1218	1456	12
HW_{std} total	888	2883	2955	12
Our controller	91	83	191	0
Recon. region	560	4480 [1]	2240 [1]	32 [1]
HW_{dpr} total	651	83+R [2]	191+R [2]	0+R [2]

[1] Maximum available resources in the reconfigurable region (600 slices).
[2] R can be replaced with the FP Adder and the FP Multiplier values, depending on the current configuration.

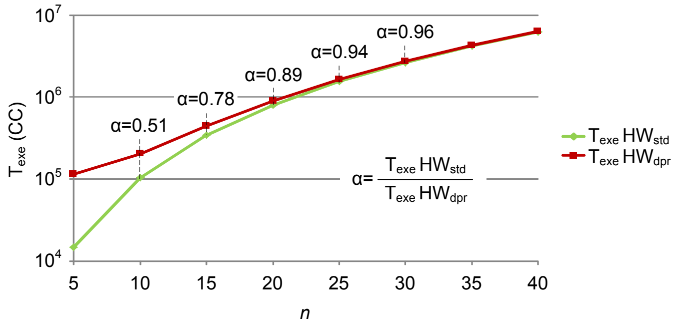

Fig. 4. Execution times, expressed in CC, for the two test-cases and their ratio for different values of n (size of the matrices).

Overall, we can observe that for the HW_{std} and the HW_{dpr} test-cases, using hardware to implement software-equivalent tasks leads to a simplification of the WCET analysis and therefore to a reduction of its pessimism. For this example, this directly translates into a considerable speed-up in the WCET of the tasks executed in hardware.

Table V shows the FPGA hardware resource utilization for the Patmos processor and for the acceleration infrastructure for the HW_{std} and the HW_{dpr} test-cases, targeting the Virtex-6 FPGA. The total resources needed to implement the reconfigurable region and the reconfiguration infrastructure for the HW_{dpr} test-case are roughly 73 % of the resources needed to implement the static accelerator for the HW_{std} test-case.

Figure 4 shows the execution times T_{exe} of the test application for the two test cases for different sizes n of the A, B, and C matrices. The $\mathrm{T}_{\mathrm{exe}}$ for the $\mathrm{HW}_{\mathrm{dpr}}$ test-case also takes into account the reconfiguration time overhead. In the Figure 4 it is shown that, for a matrix size $n > 25$, the ratio α between the execution times of the two test-cases becomes very close to 1, showing that the reconfiguration overhead becomes negligible with respect to the duration of the computation interval. In this case, taking into account the hardware resources utilization results presented in Table V, we can observe that DPR leads to a more efficient usage of FPGA resources, while maintaining comparable computational performance.

VI. Conclusion

This paper presented a lightweight controller especially developed to support DPR in real-time systems. The controller offers two different operating modes (*SPM-stream* and *CPU-stream*) to perform the reconfiguration in a bounded and predictable amount of time. We also presented a software tool for conversion and compression of the bitstreams, and for reconfiguration time analysis.

The DPR controller was evaluated in terms of hardware cost, operating frequency, reconfiguration speed, bitstream compression ratio, and worst-case reconfiguration time. We also presented a simple application example of the reconfiguration features offered by the presented architecture, including WCET numbers, hardware resources, and speed results.

Source Access

The source code of the RT-ICAP controller and the *convbit-stream* tool are available at *https://github.com/t-crest/reconfig/*. The full T-CREST platform is available at *https://github.com/t-crest/*. The entire work is open-source under the terms of the simplified BSD license.

References

[1] L. Pezzarossa, M. Schoeberl, and J. Sparsø, "Reconfiguration in FPGA-based multi-core platforms for hard real-time applications," in *Proc. of the International Symposium on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*. IEEE, 2016, pp. 1–8.

[2] M. Schoeberl et al., "T-CREST: Time-predictable multi-core architecture for embedded systems," *Journal of Systems Architecture*, vol. 61, no. 9, pp. 449–471, 2015.

[3] XILINX, "DS586: LogiCORE IP XPS HWICAP product specifications (v5.01a)," Tech. Rep., 2011, online (last accessed: Jan. 2016).

[4] ——, "DS586: Processor Local Bus (v4.6)," Tech. Rep., 2019, online (last accessed: Jan. 2016).

[5] ——, "PG134: AXI HWICAP LogiCORE IP product guide (v3.0)," Tech. Rep., 2015, online (last accessed: Jan. 2016).

[6] ——, "UG761: LogiCORE IP XPS HWICAP product specifications (v13.1)," Tech. Rep., 2011, online (last accessed: Jan. 2016).

[7] ——, "UG081: MicroBlaze processor reference guide (v10.3)," Tech. Rep., 2009, online (last accessed: Jan. 2016).

[8] ——, "UG011: PowerPC processor reference guide (v1.3)," Tech. Rep., 2010, online (last accessed: Jan. 2016).

[9] M. Liu, W. Kuehn, Z. Lu, and A. Jantsch, "Run-time partial reconfiguration speed investigation and architectural design space exploration," in *Proc. of the International Conference on Field Programmable Logic and Applications (FPL)*. IEEE Computer Society, 2009, pp. 498–502.

[10] 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, Sept 2014.

[11] XILINX, "PG139: LogiCORE IP PRC product guide (v1.0)," Tech. Rep., 2015, online (last accessed: Jan. 2017).

[12] J. Tarrillo, F. A. Escobar, F. L. Kastensmidt, and C. Valderrama, "Dynamic partial reconfiguration manager," in *Proc. of the Latin American Symposium on Circuits and Systems (LASCAS)*. IEEE, 2014, pp. 1–4.

[13] V. Lai and O. Diessel, "ICAP-I: A reusable interface for the internal reconfiguration of Xilinx FPGAs," in *Proc. of the International Conference on Field-programmable Technology (FTP)*. IEEE Computer Society, 2009, pp. 357–360.

[14] S. D. Carlo, P. Prinetto, P. Trotta, and J. Andersson, "A portable open-source controller for safe dynamic partial reconfiguration on Xilinx FPGAs," in *Proc. of the 25th International Conference on Field Programmable Logic and Applications (FPL)*, 2015, pp. 1–4.

[15] M. Schoeberl et al., "Towards a time-predictable dual-issue microprocessor: The Patmos approach," in *Proc. of the Workshop on Bringing Theory to Practice: Predictability and Performance in Embedded Systems (PPES)*, 2011, pp. 11–20.

[16] 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. Springer, 2012, vol. 7613, pp. 382–391.

[17] R. Heckmann and C. Ferdinand, "Worst-case execution time prediction by static program analysis," AbsInt Angewandte Informatik GmbH, Tech. Rep., online (last accessed: Jan. 2017).

[18] S. Hepp, B. Huber, J. Knoop, D. Prokesch, and P. P. Puschner, "The platin tool kit - The T-CREST approach for compiler and WCET integration," in *Proc. of the 18th Kolloquium Programmiersprachen und Grundlagen der Programmierung (KPS)*, 2015.

[19] E. Kasapaki, M. Schoeberl, R. B. Sørensen, C. Mller, K. Goossens, and J. Sparsø, "Argo: A real-time network-on-chip architecture with an efficient GALS implementation," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 24, no. 2, pp. 479–492, 2016.

[20] J. Garside and N. C. Audsley, "Investigating shared memory tree prefetching within multimedia NoC architectures," in *Proc. of the Memory Architecture and Organisation Workshop*, 2013.

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

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

[23] XILINX, "UG702: Partial reconfiguration user guide (v14.1)," Tech. Rep., 2012, online (last accessed: Jan. 2016).

[24] ——, "UG360: Virtex-6 FPGA configuration user guide (v3.9)," Tech. Rep., 2015, online (last accessed: Jan. 2016).

[25] L. Pezzarossa, R. B. Sørensen, M. Schoeberl, and J. Sparsø, "Interfacing hardware accelerators to a time-division multiplexing network-on-chip," in *Proc. of 1st Nordic Circuits and Systems Conference*. Oslo, Norway: IEEE, October 2015.

[26] OCP-IP Association, "Open core protocol specification 2.1," http://www.ocpip.org/, 2005.

[27] S. Hauck and W. D. Wilson, "Runlength compression techniques for FPGA configurations," in *Proc. of the Annual Ieee Symposium on Field-programmable Custom Computing Machines (FCCM)*. IEEE Computer Society, 1999, pp. 286–7, 286–287.

[28] Z. Li and S. Hauck, "Configuration compression for Virtex FPGAs," in *Proc. of the Annual Ieee Symposium on Field-programmable Custom Computing Machines (FCCM)*. Institute of Electrical and Electronics Engineers Inc., 2001, pp. 147–159.

[29] D. Koch, C. Beckhoff, and J. Teich, "Hardware decompression techniques for FPGA-based embedded systems," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 2, no. 2, pp. 9:1–9:23, Jun. 2009.

[30] L. A. Cardona and C. Ferrer, "AC-ICAP: A flexible high speed ICAP controller," *Intl. Journal of Reconfigurable Computing*, vol. 2015, pp. 1–15, 2015.

[31] 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.

[32] P. Degasperi, S. Hepp, W. Puffitsch, and M. Schoeberl, "A method cache for Patmos," in *Proc. of the 17th IEEE Symposium on Object/Component/Service-oriented Real-time Distributed Computing (ISORC)*, 2014, pp. 100–108.

[33] B. Huber, S. Hepp, and M. Schoeberl, "Scope-based method cache analysis," in *Proc. of the 14th International Workshop on Worst-Case Execution Time Analysis (WCET)*, 2014, pp. 73–82.