Can Real-Time Systems Benefit from Dynamic Partial Reconfiguration?

Pezzarossa, Luca; Kristensen, Andreas Toftegaard; Schoeberl, Martin; Sparsø, Jens

Published in:
Proceedings of the IEEE NorCAS 2017

Link to article, DOI:
10.1109/NORCHIP.2017.8124984

Publication date:
2017

Document Version
Peer reviewed version

Can Real-Time Systems Benefit from Dynamic Partial Reconfiguration?

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

***Abstract*—In real-time systems, a solution where hardware accelerators are used to implement computationally intensive tasks can be easier to analyze, in terms of worst-case execution time (WCET), than a pure software solution. However, when using FPGAs, the amount and the complexity of the hardware accelerators are limited by the resources available. Dynamic partial reconfiguration (DPR) of FPGAs can be used to overcome this limitation by replacing the accelerators that are only required for limited amounts of time with new ones. This paper investigates the potential benefits of using DPR to implement hardware accelerators in real-time systems and presents an experimental analysis of the trade-offs between hardware utilization and WCET increase due to the reconfiguration time overhead of DPR. We also investigate the trade-off between the use of multiple specialized accelerators combined with DPR instead of the use of a more general accelerator. The results show that, for computationally intensive tasks, the use of DPR can lead to a more efficient use of the FPGA, while maintaining comparable computational performance.**

I. Introduction

In general-purpose systems, hardware accelerators (HwAs) are mainly used to speed up average-case execution time of computationally intensive 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. Execution time analysis of hardware used to implement software-equivalent tasks is often easier to perform than analysis of a pure software solution. From a real-time perspective, moving functionality from software into hardware can lead to reduction of the WCET and time-analysis simplification [1], [2].

FPGAs can today implement complete systems-on-chip, composed of soft-core processors and HwAs. For real-time systems, where a low production volume often makes ASIC implementations prohibitively expensive, FPGAs are particularly attractive. One of the drawbacks in using FPGAs, however, is that the complexity of the HwAs is limited by the available resources, especially considering that the FPGA cost is very sensitive to its size.

This limitation can be overcome, to some extent, by using the dynamic partial reconfiguration (DPR) feature offered by modern FPGAs [3]. DPR allows for dynamic reconfiguration of selected regions on the FPGA, while the remaining parts of the FPGA remain unaffected by the reconfiguration. This allows for a more efficient utilization of FPGA resources, since HwAs that are only required for limited amounts of time can be replaced when the functionality implemented in these regions is no longer required. From a real-time perspective, this translates into the possibility to use HwAs to simplify the WCET analysis of selected software tasks. However, the required reconfiguration time introduces an overhead that needs to be considered every time an HwA is reconfigured.

In this paper, we try to experimentally answer the question: *"Can real-time systems benefit from dynamic partial reconfiguration?"*. In order to do this, we investigate the potential benefits of using DPR to implement HwAs in real-time systems and we present an experimental WCET analysis and hardware resources utilization for four test cases. More specifically, we compare a static approach, in which non-reconfigurable HwAs are used to implement software tasks, with a reconfigurable approach, in which DPR is used to switch between different HwAs. We therefore analyze the trade-offs between hardware-resource utilization and the computational performance loss due to the reconfiguration time overhead of DPR, which directly affects the overall WCET.

For one of the test cases, we also investigate whether using DPR to switch between multiple specialized HwAs could provide a lower WCET bound with respect to the use of a more general HwA.

The experiments are carried out targeting the Patmos processor [4] and using HwAs generated from four selected real-time TACLe suite benchmarks [5] using the Xilinx high-level synthesis (HLS) tool Vivado HLS [6]. The Patmos WCET analysis tool-chain is used to perform the WCET analysis of the accelerated tasks and of the reconfiguration process.

This paper makes two contributions: 1) it investigates the benefits of using DPR to implement HwAs in real-time systems analyzing the hardware utilization vs. computational performance trade-offs (in terms of WCET), and 2) it presents an experimental evaluation using HLS-generated HwAs from four selected real-time TACLe benchmarks.

This paper is organized in six sections: Section II presents the related work. Section III provides the general background related to DPR, the reconfiguration controller RT-ICAP, and the Patmos processor infrastructure. Section IV describes the experimental setup used to produce the results, which consists of the hardware platform and the set of HwAs from the TACLe benchmark suite. Section V presents and discusses the experimental results. Finally, section VI concludes the paper.

II. Related Work

To our knowledge, the use of DPR in real-time systems represents a novel and unexplored field of research and the related work that addresses the use of DPR from a real-time perspective is very limited. Therefore, we also include some related work regarding the use of DPR in general-purpose systems.

In the following, we list some relevant hardware/software frameworks especially developed to support reconfiguration and some works addressing scheduling-related problems. These works are representative of the benefits and the challenges experienced from using DPR.

A complete survey of the most relevant hardware aspects of reconfigurable computing can be found in [7]. The work explores the challenges of runtime hardware reconfigurable architectures, addressing both single-chip and multi-chip architectures.

The work presented in [8] proposes a software framework that exploits HwAs combined with DPR in the development of safety-critical real-time systems. It presents a model used to derive the response-time analysis to verify the schedulability of a real-time task set under given constraints and assumptions.

In [9], the author studies the dynamic behavior of reconfigurable architectures, especially focusing on the use of DPR. The work 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.

Another framework is the one presented in [10], called ReCoBus-builder, that addresses component-based, reconfigurable, non-real-time systems. It uses DPR to generate dynamically reconfigurable systems providing one or more runtime reconfigurable areas.

The work presented in [11] provides an overview of the hardware-software partitioning, scheduling, and placement issues and proposes an exact and a heuristic approach for hardware-software partitioning in a system that uses DPR. The paper takes into account key factors such as placement implications and configuration pre-fetching for minimizing the schedule length.

In [12], the authors present the PaRA-Sched automated design methodology. This 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.

Finally, we present some of our previous work that relates to this paper. In [1], we explore the use of DPR in real-time systems to implement mode changes in the context of the T-CREST multi-core platform and in [2] we present a hardware/software infrastructure to support time-predictable reconfiguration. In this paper, by means of experiments we explore some of the concepts that were presented from only a purely theoretical perspective in these two papers.

III. Background

In this section, we present the general background related to DPR, the reconfiguration controller RT-ICAP, and the Patmos processor infrastructure.

A. Dynamic Partial Reconfiguration

DPR is a feature of modern FPGAs that allows runtime modification of an operating FPGA [3]. Partial bit-streams 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 two main parts: a non-reconfigurable static part configured at boot-time with a full bit-stream, and a runtime reconfigurable part, which may consist of many independent reconfigurable regions. Each reconfigurable region can be reconfigured multiple times during runtime with different partial bit-streams without interfering with the functionality of the static part.

For Xilinx FPGAs, DPR is performed at runtime by loading a partial bit-stream through one of the FPGA configuration interfaces. For this work, the ICAP (internal configuration access port) is used to provide access to the configuration memory and partially reconfigure the FPGA after its initial configuration. Modifying the content of the configuration memory corresponds to a change in the hardware implemented on the FPGA.

The time needed to perform a reconfiguration is proportional to the size of the partial bit-stream to be transferred over the ICAP interface, which depends on the size of the region to be reconfigured. For example, assuming the widest possible interface (32 bits) and the fastest possible clock (100 MHz) for the ICAP interface, the reconfiguration time of a reconfigurable region of 500 slices (which can accommodate a double-precision floating-point adder) is $300\,\mu$s [1].

The reconfiguration time is a very important parameter since it introduces a time overhead of DPR that needs to be considered every time an HwA is reconfigured, leading to a computational performance loss when compared to a fully static approach (not using DPR), since it increases the WCET.

B. The RT-ICAP Reconfiguration Controller

The ICAP interface needs to be interfaced with a controller that manages the reconfiguration. For this purpose, we have developed the time-predictable reconfiguration controller RT-ICAP, presented in [2]. The RT-ICAP controller is a time-predictable lightweight DPR controller that enables a processor to perform DPR.

The RT-ICAP controller, integrated in the hardware architecture used in this paper, is shown in Figure 1. The controller is interfaced to the processor, the bit-stream scratch-pad memory (SPM), and the ICAP interface. Once the bit-stream is stored in the SPM, the processor sets up the RT-ICAP controller to initiate a reconfiguration, which autonomously fetches the bit-stream from the bit-stream-SPM and loads it into the FPGA configuration memory through the ICAP interface.

The RT-ICAP controller is supported by a software tool, named *convbitstream*. For each configuration, the *convbitstream* tool computes, at compile-time, the reconfiguration time.

The reconfiguration time is from the moment the processor initiates the reconfiguration and until the partial bit-stream is completely written into the FPGA's configuration memory and the reconfigurable region is ready to be used. This time interval is needed to perform WCET analysis of an application that uses the reconfiguration feature.

C. The Patmos Processor and the 'platin' Time-Analysis Tool

The processor used in this work is Patmos [4]. Patmos is a time predictable, dual-issue, RISC processor used in the T-CREST [13] multi-core platform and it has been developed specifically for use in real-time applications. It contains special instruction and data caches, and local private SPMs for instructions and data.

In real-time systems, the calculation of the WCET is fundamental to determine the system ability to respond in time. For this reason, several commercial tools and research prototypes have been developed to satisfy this need [14]. Patmos is supported by an LLVM-based compiler, also developed with focus on WCET [15] and by the WCET analysis tool *platin* (portable LLVM-based annotation and timing analysis integration) [16], which allows static derivation of tight WCET bounds. The tool computes the WCET by analyzing the software without executing it on hardware. It works both at the intermediate representation of the LLVM-based compiler to determine the structure of the program and at the machine code level, taking into account the hardware timing and architecture specific information of the utilized hardware platform.

The *platin* tool, together with the *convbitstream* tool and the HwA execution time information, are used in this work to compute the WCET reported in the results section.

IV. Experimental Setup

This section describes the experimental setup used to produce the results. More specifically, we describe the hardware platform and the set of HwAs from the TACLe benchmark suite that we use.

A. Hardware Platform

The hardware platform used in our experiments is shown in Figure 1. The I/O devices are connected to the Patmos processor using a bus that implements a subset of the open-core protocol (OCP) [17]. The HwA is connected to a shared memory (HwA-SPM) for data exchange with Patmos and to a controller (HwA-ctrl) used to manage the HwA and provide the current status to the processor. The HwA uses the *ap_ctrl_hs* interface protocol to communicate with the HwA-ctrl, as defined in [6, p. 89].

The HwA-SPM is divided into a certain number of banks decided at synthesis time. The HwA-SPM appears as a single address space to the processor, but the banks can be accessed in parallel by the HwA to increase the memory bandwidth towards it. Note that the HwA-SPM is not the local SPM of Patmos,

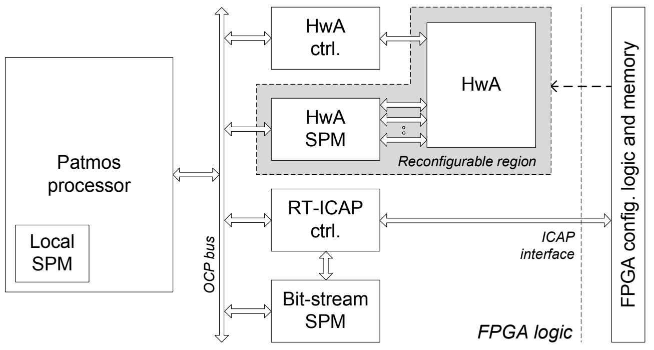

Fig. 1. An overview of the hardware platform used for the experiments. All the I/O devices are connected to the processor using the OCP bus.

which is used by the processor to store easily accessible data and instructions.

Both the HwA and the HwA-SPM reside in the reconfigurable region, since the number of memory banks used is dependent on the HwA and, therefore, it needs to be reconfigured together with the HwA.

The reconfiguration is managed by the RT-ICAP controller presented in subsection III-B. The RT-ICAP controller can modify the content of the FPGA configuration memory through the ICAP interface. Therefore, by writing a partial bit-stream (stored in the bit-stream SPM) into this memory, the content of the reconfigurable region is dynamically modified. The dashed arrow in Figure 1 characterizes this dependency.

Assuming that the bit-streams are available in the bit-stream SPM, the full operational flow of the system to use the HwA is as follows:

1) Patmos requests the RT-ICAP controller to reconfigure the needed HwA;
2) Patmos moves the data to be processed into the HWA SPM;
3) Patmos activates the HwA by interacting with the HWA controller;
4) When the HwA has finished, Patmos can read the processed data from the HWA SPM.

The step 1) can be skipped if the reconfigurable area does not need to be reconfigured. During step 3), when the HwA is running, the processor is free to execute other operations.

B. Benchmarks and Hardware Accelerators

The HwAs used in this paper are based on code from four benchmarks from the TACLe suite, which is a collection of open-source C programs, for timing analysis and real-time related research [5]. For the selected benchmarks, the computationally intensive part of the program is identified and moved into hardware. The benchmark is then modified to interact with the HwA to perform the section of the program that was moved into hardware. When DPR is used, the reconfiguration of the dynamic region with the needed HwA is performed before using the HwA.

The HwAs used in this paper are generated using Xilinx Vivado HLS, which is an automated design process able to transform high-level language code, such as C, into functionally equivalent synthesizable RTL HDL code [6]. In our case, it is used to transform C code from the TACLe real-time benchmarks into VHDL.

The benchmarks have been chosen to be representative of HwAs working on large data sets, small data sets, and data streams. In the following, we provide a brief description of the functionality of each benchmark and of the characteristics of the associated HwA. The data type used in all the benchmarks is single-precision floating-point.

Matrix multiplication: This benchmark executes the matrix multiplication between two square matrices. For this benchmark, we have generated different specialized HwAs for matrices of size 4×4, 16×16, and 32×32. Moreover, we have generated a generic HwA for matrix multiplication which can take any given matrix size up to 32×32. This is used to analyze how the specialized HwAs combined with partial reconfiguration perform in comparison with a generic HwA.

Matrix inversion: This benchmark computes the matrix inversion operation on a square matrix. For this benchmark, we have also generated HwAs for matrices of size 4×4, 16×16, and 32×32.

2D FIR filter: This benchmark performs a bi-dimensional FIR filtering on a matrix of size M×N using a 3×3 coefficient mask. This kind of filtering is commonly used for smoothing or sharpening bi-dimensional data sets. More specifically, the benchmark computes the cross-correlation between a 4×4 area surrounding each value and the coefficient mask. Zero-padding is performed at the matrix edges to satisfy the filter conditions. For this benchmark, we have generated an HwA for a matrix of size 3×3, corresponding to the minimum input matrix size.

Filterbank: This benchmark implements a filter-bank with FIR filters for multi-rate signal processing. The input signal is passed into eight different FIR filters. The filtered signals are then down-sampled and up-sampled again. The up-sampled signal is passed through a second set of FIR filters and finally the outputs are summed together. Normally, some data processing is performed between the down-sampling and the up-sampling. In our benchmark, we do not perform any processing. For this benchmark, we have generated an HwA where the input data and the two sets of filter coefficients are passed as arguments.

V. Results and Discussion

This section presents the experimental evaluation of the use of DPR in real-time systems, in terms of reconfiguration overhead, hardware utilization, and trade-off between specialized and generic HwAs.

All the results of our architecture, presented in this section, are produced using Xilinx Vivado and HLS (v16.4) and targeting the Xilinx Artix-7 FPGA (model XC7A100T-1CSG324C). Equivalent results can be obtained when targeting a different FPGA model. The size of the reconfigurable region used in all the experiments is 1500 slices, which contains 6000 look-up tables (LUTs), 12000 flip-flops (FFs), 30 block-RAMs (BRAMs), and 40 digital signal processing (DSP) elements.

TABLE I
Contributions to the WCET in clock-cycles and the number of times ($N_{95\,\%}$) the HwA has to be used to reach a performance that is 95 % of the one of a system that uses a static HwA.

Benchmark	Software (C_{sw})	Hardware (T_{hwa})	Reconfig. (C_{rec})	$N_{95\,\%}$
Matrix mult.				
- 4×4	3 203	42	133 436	781
- 16×16	30 345	1 107	130 786	79
- 32×32	114 816	8 351	133 823	21
Matrix inv.				
- 4×4	2 307	793	130 772	802
- 16×16	21 363	12 168	131 885	75
- 32×32	78 859	55 223	130 071	18
2D FIR filter	3 174	137	132 098	758
Filterbank	46 450	216 264	132 152	10

A. Reconfiguration Overhead

The first set of results is related to the WCET of the benchmarks and aims to characterize the overhead of reconfiguration over actual computation time.

Three factors contribute to the WCET of a benchmark: C_{sw}, C_{hwa}, and C_{rec}. The first contribution C_{sw} is the WCET of the software section of the benchmark produced by the *platin* time-analysis tool. This includes the WCET of the data transfer to and from the HwA-SPM and the WCET of the setup of the HwA. The second contribution C_{hwa} is the time needed by the HwA to perform the computation. This result is produced by the Vivado HLS tool. The third contribution C_{rec} is the reconfiguration time needed to perform the reconfiguration of the reconfiguration region. This time interval is produced by the *convbitstream* tool. All time periods are measured in clock cycles.

Table I presents the values of these three contributions for all the benchmarks considered in this work. The total WCET of a benchmark that uses DPR is denoted C_{tot_dpr} and it can be calculated using equation (1), where N is the number of computations executed by the HwA after a reconfiguration.

$$C_{tot\_dpr} = C_{rec} + N\left(C_{sw} + C_{hwa}\right) \qquad (1)$$

The reconfiguration time, C_{rec}, represents an overhead, and the effect of this overhead reduces the more times the HwA is used. The last column of table Table I with heading $N_{95\,\%}$ shows the number of times the HwA has to be used in order to reach a performance that is 95 % of the performance of a system that uses a static HwA. It is possible to observe that, for the HwAs that perform computationally intensive tasks, such as *Matrix multiplication* and *Matrix inversion* for 32×32 matrices and *Filterbank*, the value of $N_{95\,\%}$ is very low. Low values of

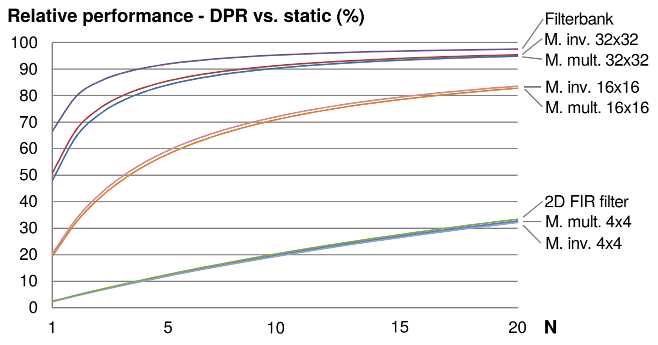

Fig. 2. Plot showing the performance when using DPR relative to the performance when using static HwAs for values of $N \in [1, 20]$.

$N_{95\,\%}$ show that the reconfigurable solution may be particularly beneficial, even if only a small number of computations are required, since the loss of performance is compensated for by lower hardware cost as discussed in the next subsection.

Figure 2 provides additional insight into the relation between reconfiguration overhead and the number of times, N, that the HwA is used. The figure shows the performance when using DPR relative to the performance when using static HwAs for values of $N \in [1, 20]$. In the figure, it is possible to observe how results for similar benchmarks tend to cluster together. In the top of the plot we can find the curves related to *Filterbank* and the benchmarks operating on 32×32 matrices. Right below, in the second group, we can find the curves related to the benchmarks operating on 16×16 matrices. For these two groups, the solution using DPR becomes comparable to the static approach, in terms of computational performance, for low values of N. Finally, the curves related to the *2D FIR filter* and to the benchmarks operating on 4×4 matrices are located in the lower part of the plot, showing that it is unlikely that a real application can benefit from using DPR.

B. Hardware Utilization

One of the advantages of using DPR is the possibility of reducing the hardware utilization in the case when some of the hardware resources implemented in a system are only utilized for a limited amount of time, since the HwAs can be loaded in the reconfigurable regions when needed.

Table II shows the hardware utilization results for the main components of the hardware platform (shown in Figure 1) and for all the HwAs used in the experiments, in LUTs, FFs, BRAMs, and DSP elements.

Considering a hypothetical application that includes all the functionality of the benchmarks used in this work, it is possible to observe that the minimum size of the reconfigurable region should be enough to contain the largest hardware requirements across all the HwAs. In our case, this corresponds to 5 126 LUTS, 7 411 FFs and 3 BRAMs to fit the *Filterbank* HwA, and 20 DSP elements to fit the *Matrix multiplication* and *Matrix inversion* HwAs requirements. In a static solution, where all the HwAs need to be implemented, the total resource utilization would be roughly equal to the sum of the hardware

TABLE II
HARDWARE UTILISATION RESULTS FOR THE HARDWARE PLATFORM AND FOR ALL THE HWAS USED IN THE EXPERIMENTS.

Entity	LUT	FF	BRAM	DSP
Patmos	4 931	3 602	8.5	4
HwA controller	7	4	0	0
ICAP controller	289	105	0	0
Matrix mult.				
- 4×4	1 270	1 138	0	20
- 16×16	1 979	2 523	0	20
- 32×32	2 763	4 048	0	20
Matrix inv.				
- 4×4	2 051	2 017	0.5	5
- 16×16	3 425	3 725	0.5	20
- 32×32	4 080	4 636	0.5	20
2D FIR filter	1 816	1 987	0	10
Filterbank	5 126	7 411	3	10
Generic matrix mult.	2 912	4 037	0	5

resources of each HwA. This leads to a relevant saving in the hardware cost, since the required minimum size of the reconfigurable region is the 23 % of LUTs and the 27 % of FFs of the estimated hardware cost of a static solution.

Taking the hardware resource utilization results and the performance results presented in the previous subsection into account, we can observe that, for a value of N sufficiently high, DPR leads to a more efficient use of FPGA resources, while maintaining comparable computational performance.

C. Specialized vs. Generic Accelerator

The goal of this experiment is to determine what benefits a very specific HwA combined with DPR may bring compared to a generic one, since the reconfiguration feature can be used to just change the type of specialized HwA based on current requirements.

As previously mentioned, for *Matrix multiplication* we have generated specialized HwAs for matrix sizes of 4×4, 16×16, 32×32, and a generic HwA which can take in input matrices of any size up to 32×32. The idea is to investigate the trade-offs between using DPR to switch between multiple specialized HwAs and using a more general HwA, in terms of WCET and hardware utilization.

Table III shows the WCETs for both the specialized HwAs and the generic one in clock cycles for the *Matrix multiplication* benchmark for the three different matrix sizes. Table III also shows the reconfiguration time for the specialized HwA.

Contrarily to the experiment presented in Subsection V-A, where the solution using reconfiguration is always slower than the static one, in this experiment the solution using reconfiguration can be faster than a static solution, since the specialized HwAs are faster than the generic ones.

TABLE III
WCET results for general and specialized HwAs, reconfiguration time for the specialized HwAs, and the minimum number of operations $N_{100\,\%}$ for which it is convenient to use DPR.

	$4{\times}4$	$16{\times}16$	$32{\times}32$
General HwA	8 028	49 438	152 933
Specialized HwA	3 245	31 452	123 167
Reconfig. time	133 436	130 786	133 823
$N_{100\,\%}$	28	8	5

Therefore, after a certain amount of computation, the overhead introduced by the reconfiguration time will be compensated for by the difference in speed between the specialized and the general HwA. The value $N_{100\,\%}$, shown in the last row of Table III, is the threshold value of N (number of times the HwA is used) in which the general and the specialized HwAs, including reconfiguration, are equivalent in performance. For values of $N > N_{100\,\%}$, the use of the specialized HwA combined with reconfiguration outperform the general HwA.

In terms of hardware, it is possible to observe from Table II that the minimum size of the reconfigurable region should be enough to contain the specialized *Matrix multiplication* HwA for size $32{\times}32$, which is smaller than the hardware resources needed to implement the generic HwA (last row of Table II).

VI. Conclusion

In this paper, we tried to answer the question: *"Can real-time systems benefit from dynamic partial reconfiguration?"*. In order to do this, we have performed an experimental analysis of the WCET and hardware resources utilization for four test cases derived from the real-time TACLe benchmark suite.

Our conclusion is that, in the case where an application performs a sequence of tasks in a way that allows different HwAs to be loaded in the reconfigurable region when needed, the use of DPR can lead to a significant reduction in the hardware cost if the tasks moved into hardware are sufficiently computationally intensive. In our benchmarks, this was observed for computationally intensive tasks, such as the *Filterbank* and the benchmarks operating on $32{\times}32$ matrices.

Similarly, the experiments regarding the trade-offs between specialized and general HwAs suggest that, for computationally intensive tasks, the use of the specialized HwAs combined with reconfiguration is advantageous with respect to the general HwA, both in terms of performance and hardware utilization.

Source Access

The source code used for synthesis is available at *https://github.com/A-T-Kristensen/patmos_HLS/tree/master/hls* and the code required to run on the T-CREST platform is available at *https://github.com/A-T-Kristensen/patmos/tree/patmos_hls*. 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] ——, "A controller for dynamic partial reconfiguration in FPGA-based real-time systems," in *Proc. of the 20th International Symposium on Real-Time Distributed Computing (ISORC)*. IEEE, 2017, pp. 92–100.

[3] XILINX, "UG909: Vivado Design Suite User Guide - Partial Reconfiguration (v2017.1)," Tech. Rep., 2017, online (last accessed: Aug. 2017).

[4] 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 *Proc. of the Workshop on Bringing Theory to Practice: Predictability and Performance in Embedded Systems (PPES)*, 2011, pp. 11–20.

[5] H. Falk, S. Altmeyer, P. Hellinckx, B. Lisper, W. Puffitsch, C. Rochange, M. Schoeberl, R. B. Sørensen, P. Waegemann, and S. Wegener, "TACLeBench: A benchmark collection to support worst-case execution time research," in *Proc. of the 16th International Workshop on Worst-Case Execution Time Analysis (WCET)*, 2016, p. 10.

[6] XILINX, "UG902: Vivado Design Suite User Guide - High-Level Synthesis (v2017.1)," Tech. Rep., 2017, online (last accessed: Aug. 2017).

[7] K. Compton and S. Hauck, "Reconfigurable computing: A survey of systems and software," *ACM Computing Surveys*, vol. 34, no. 2, pp. 171–210, 2002.

[8] A. Biondi, A. Balsini, M. Pagani, E. Rossi, M. Marinoni, and G. Buttazzo, "A framework for supporting real-time applications on dynamic reconfigurable FPGAs," in *Proc. of the Real-Time Systems Symposium (RTSS)*. IEEE, Nov 2016, pp. 1–12.

[9] K. Wu, "Reconfigurable architectures: from physical implementation to dynamic behavoir modelling," Ph.D. dissertation, Dept. of Informatics and Mathematical Modelling, Technical University of Denmark, 2007.

[10] 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.

[11] 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.

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

[13] 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.

[14] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The worst-case execution time problem – overview of methods and survey of tools," *Trans. on Embedded Computing Systems*, vol. 7, no. 3, pp. 1–53, 2008.

[15] 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.

[16] 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.

[17] OCP official website, "Webpage: http://www.accellera.org/downloads/standards/ocp/," (last accessed: Aug. 2017).