High-level synthesis for reduction of WCET in real-time systems

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

Published in:
Proceedings of the 2017 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC)

Link to article, DOI:
10.1109/NORCHIP.2017.8124945

Publication date:
2017

Document Version
Publisher's PDF, also known as Version of record

Link back to DTU Orbit

Citation (APA):
Kristensen, A. T., Pezzarossa, L., & Sparsø, J. (2017). High-level synthesis for reduction of WCET in real-time systems. In Proceedings of the 2017 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC) (pp. 1-6). IEEE. DOI: 10.1109/NORCHIP.2017.8124945

High-Level Synthesis for Reduction of WCET in Real-Time Systems

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

***Abstract*—The increasing design complexity of systems-on-chip (SoCs) requires designers to work at higher levels of abstraction. High-level synthesis (HLS) is one approach towards this. It allows designers to synthesize hardware directly from code written in a high-level programming language and to more quickly explore alternative implementations by re-running the synthesis with different optimization parameters and pragmas. HLS is particularly interesting for FPGA circuits, where different hardware implementations can easily be loaded into the target device. Another perspective on HLS is performance. Compared to executing the high-level language code on a processor, HLS can be used to create hardware that accelerates critical parts of the code. When discussing performance in the context or real-time systems, it is the worst-case execution time (WCET) of a task that matters. WCET obviously benefits from hardware acceleration, but it may also benefit from a tighter bound on the WCET. This paper explores the use of and integration of accelerators generated using HLS into a time-predictable processor intended for real-time systems. The high-level design tool, Vivado HLS, is used to generate hardware accelerators from benchmark code, and the system using the generated hardware accelerators is evaluated against the WCET of the original code. The design evaluation is carried out using the Patmos processor from the open-source T-CREST platform and implemented on a Xilinx Artix 7 FPGA. The WCET speed-up achieved is between a factor of 5 and 70.**

I. Introduction

The increasing design complexity of systems-on-a-chip (SoCs) has encouraged the design community to seek design abstractions with a higher productivity than register-transfer level (RTL). High-level synthesis (HLS) plays a role in this, by enabling the automatic synthesis of high-level specifications, in languages such as C or SystemC, to a low-level cycle-accurate RTL specification for implementation in application-specific integrated circuits (ASICs) or field-programmable gate arrays (FPGAs). Additionally, HLS allows for an efficient reuse of intellectual property (IP) cores specified in a high-level language. These IPs can more quickly be re-targeted for other systems, compared to RTL IPs, which have a more fixed architecture. HLS thus allows designers to better and faster take advantage of FPGAs, where different hardware implementations can easily be refined and replaced in the device.

In general, when systems utilize hardware accelerators (HwAs), the benefit comes from the speed-up of the average-case execution time for specific tasks. Hardware platforms for hard real-time systems are characterized by strict constraints on the execution time of tasks, and the worst-case execution time (WCET) therefore becomes the more important measure of performance. These systems must therefore exhibit a time-predictable behavior, which can be analyzed to guarantee worst-case execution times can be met [1].

When analyzing software, complexity arises from the many possible branches taken in the code, cache misses and the time needed for these actions [2]. Using hardware instead of software may simplify this analysis, particularly if the integration method allows the HwA to operate in isolation from other (peripheral) hardware devices. The latency of the HwA can then be obtained using the HLS tool and an input data set. Furthermore, the software required to use the HwA is often relatively simple, mainly involving data transfer. The purpose of utilizing HwAs in a real-time system is thus the WCET speed-up due to the hardware acceleration itself and a possible reduction in the pessimism of the estimations.

This paper explores the integration of HwAs with the time predictable Patmos CPU of the T-CREST platform [3]. The HwAs are based on benchmark code and synthesized to RTL code in a hardware description language (HDL) using Vivado HLS [4]. HwAs presented in the paper are for matrix multiplication, filter operations and compression.

We evaluate the speed-up achieved for the WCET when utilizing the HwAs and the resource cost of the HwA integration on the Artix 7 FPGA. The WCET evaluation is carried out using the WCET analysis tool *platin* [5]. We provide speed-up results for solutions using the HwA compared to pure software solutions, both for WCET and average-case execution time (ACET). This allows us to explore whether HLS has matured for real-time applications since HwAs for real-time systems are usually designed at RTL.

This paper has two main contributions: (1) it presents an integration method for HwAs generated using Vivado HLS with the Patmos CPU, and (2) it investigates and evaluates the use of HLS to minimize the WCET of a hard real-time system.

This paper is organized into six sections: Section II presents related work. Section III presents the background on real-time systems, HLS and the Patmos architecture. Section IV details the integration of the HwAs with Patmos, the general tool flow used to obtain the results for the paper and the benchmark programs. Section V presents the results for the paper and evaluates these. Finally, we conclude the paper in section VI.

978-1-5386-2844-7/17/$31.00

II. Related Work

A large body of literature exists addressing the development of accelerators in HDL and the co-operation among CPUs and HwAs. In [6], an FPGA accelerator for a convolutional neural network is evaluated against a hard-core CPU, with a reported speed-up of a factor of 17.42. The authors in [7] evaluate the co-operation of an ARM CPU and HwAs on the Xilinx Zynq for the average-case speedup and the energy efficiency for a filtering task. In [8], a hardware accelerator for 2D-DCT computations is evaluated as a co-processor for the Xilinx MicroBlaze soft-core processor, providing an overall speed-up of around 20%.

Accelerators designed using HLS are also common in the literature. In [9], vision and navigation applications such as SLAM are accelerated on an FPGA using HLS to generate the HwAs. The interconnect to a 2 GHz Intel Xeon processor is modeled and speed-ups ranging from 4.4 to 15 times are reported. In [10], the authors use the HLS tool LegUp and explore speed-ups when varying amounts of code from the CHStone benchmark suite [11] are moved into hardware, instead of executing on a soft-core MIPS processor. In [12], the average-case speed-up achieved using HwAs generated with LegUp is evaluated for the computation of the Mandelbrot set for an image, presenting speed-ups ranging from 1.11 to 5.70 times.

Accelerators have also been used to reduce the WCET. [13] presents a hardware scheduler architecture for real-time systems and the WCET for using either the hardware or the software implementation is provided. In [14], an implementation of time-consuming RTOS parts in hardware is proposed as reconfigurable co-processors. One of these, a hardware task-status manager (HTSM), is prototyped on an FPGA for hardware task-status manager and speed-ups ranging between 1.8 to 13.3% are reported. In [15], the authors report a 60% performance increase when context switching and software interrupt processing operations in an RTOS are accelerated as FPGA-based co-processors.

While the literature on the average-case speed-ups achieved when using either HDL or HLS are plentiful, this paper considers the speed-up achieved with the co-operation between a CPU and HwA for the WCET, where the HwAs are generated using an HLS tool.

III. Background

This section provides a brief overview of real-time systems, HLS, the Patmos architecture and how the HwAs have been integrated with Patmos.

A. Real-Time Systems

Real-time systems are a class of systems characterized by timing and reliability requirements, beyond the required functional correctness of the operations [1]. Real-time systems can be classified according to the consequences of not meeting a deadline. For hard real-time systems, missing a deadline could mean total system failure, some examples include nuclear power systems and the anti-lock braking system in cars.

Since time is critical for real-time systems, it should be possible to determine whether all timing constraints will be met, by computing the upper bound of the WCET for a piece of code [16].

In WCET analysis, the computed upper bound is dependent on the hardware information, meaning that the same piece of code could have different WCET bounds depending on the hardware that the software will run on. The WCET analysis determines the CPU time reserved for different tasks during task scheduling. A pessimistic WCET result, the pessimism being the difference between the upper bound and the true WCET, may require some compensation in the form of hand-crafted modifications or faster hardware, in order to meet timing constraints.

B. High-Level Synthesis and Accelerators

High-level synthesis (HLS) is an automated design process, where the functionality expressed in high-level language code, such as C or SystemC, is synthesized into a hardware description. This synthesis is guided by constraints and directives provided by the designer, in order to meet design specifications for clock frequency, resource utilization and performance. For this paper, the Vivado HLS tool from Xilinx has been utilized to generate HwAs which co-operate with the Patmos CPU [4].

Vivado HLS provides a number of optimization options controlled by applying directives. Loop operations can be pipelined and loops can be unrolled by any given factor to construct a parallelized version of a circuit. By analyzing the code, the bit-widths can be optimized manually, using custom bit-widths, to create circuits which better utilize FPGA resources such as DSP blocks. Arrays in the code can be mapped into shared or individual memory blocks to reduce BRAM utilization or improve memory bandwidth respectively. The tool also performs a large number of automatic optimizations, such as if-conversion, dead-code elimination and many of the standard optimizations used by modern software compilers.

For a more thorough discussion of the theoretical and practical aspects of HLS, the reader is directed to [17]. A survey and evaluation of different FPGA HLS tools is also provided in [18].

C. The Patmos processor

The HwAs have been integrated with the Patmos CPU of the open-source T-CREST platform [3], as shown in Figure 1 and explained later in subsection IV-A

Patmos is designed for real-time systems and optimized for time predictability. It uses special instruction and data cache memories to ease WCET analysis, and it contains a local private scratchpad memory for instructions and data. The processor does not stall, except for explicit instructions that wait for data from the memory controller. All instruction delays are thus explicitly visible at the ISA-level.

The platform is supported by a compiler, also developed with a focus on WCET [19]. In the T-CREST platform, the *platin* tool-kit is used for WCET analysis [5]. This tool has been

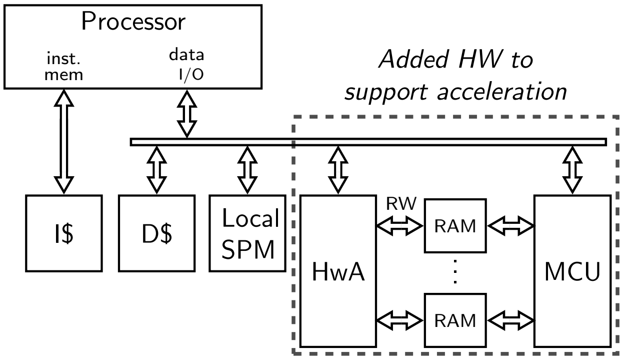


Fig. 1. An overview of the system on the CPU bus. The HwA is controlled by Patmos and the MCU is used to distribute data into different memory banks.

integrated with the LLVM-based T-CREST compiler, and the tool is also able to interface with several research and industrial strength tools. The platin tool has dedicated support for the specific architecture of Patmos, such as the method caching.

IV. Method and evaluation setup

A. Integration of processor and accelerator

The HwA is attached to the internal bus of the Patmos CPU, as shown in Figure 1, using an interface that implements a subset of the open-core protocol (OCP) [20]. As shown, the HwA has its own memory bank(s) (the RAM blocks in Figure 1). These private memory bank(s) allows for the separation of the HwA operation from the operation of the processor, which is necessary to ease the WCET analysis. Without this separation, the interference between the processor and the HwA would be very challenging to analyze and results could well be overly pessimistic.

For performance reasons, the HwA typically uses several memory banks in parallel, allowing it to access multiple operands in a single clock-cycle. This requires the code used for HLS to be structured in such a way that the available memory bandwidth is optimally utilized [21]. Vivado HLS supports several data allocation schemes which ensures that multiple memory banks are used whenever possible.

The role of the memory control unit (MCU) is to present the memory banks as a single address space towards the processor and enable hardware support for some of the data allocation schemes supported by Vivado HLS. This requires some address transformation and this transformation is controlled/configured from the processor before the HwA is used. When this is in place, the processor may write operands into the private memory of the HwA, start it, and at some later point read back the results. The processor polls the HwA, allowing for the processor and the HwA to operate concurrently.

B. Synthesis and Design Flow

Figure 2 presents an overview of the tool flow used to obtain the results of the paper. We will now go through this process.

First, an application is selected and code regions which can benefit from acceleration are identified. For evaluation, we use a set of relatively small benchmark programs which are described in Section IV-C. The code regions selected for acceleration are simply the main computational kernels. For real programs, the WCET analysis tool may be used to identify code regions which contribute the most to the WCET. We then partition the code and new .c files are written for HLS, we also write a new main .c file to use the generated HwA with Patmos. The new .c files for HLS are written explicitly for synthesis, with directives and constraints added to guide the process. The region of code to be accelerated is also passed through the WCET analysis tool, along with parameters characterizing the Patmos CPU to estimate the WCET results for the pure software.

The code for hardware acceleration is synthesized using Vivado HLS to generate the HwAs, each of which can be integrated as shown in Figure 1. Vivado HLS can then output the latency of the HwA given an input data set, thereby providing a measurement of the WCET for the HwA. This is used together with the code required to run the HwA and hardware information, to compute the WCET of the system with the HwA integrated. The optimal solution is obtained by testing the effect of different constraints and directives on the latency and resource utilization of the HwAs and selecting the solution which minimizes the latency for a reasonable resource utilization. This includes varying the number of memory banks available, unrolling and pipelining loops, rewriting code, constraining resource utilization etc.

Finally, to obtain the resource utilization, the synthesized design, along with constraint files and the HDL code for the Patmos CPU are synthesized for the FPGA using Vivado. The different designs have also been verified on the FPGA.

C. Benchmarks

TABLE I
An overview of the TACLe [22] and CHStone [11] benchmarks used in the paper with the amount of bytes written and read for the HwAs.

Accelerator	Data type	Lines of code	Input data	Output data
ADPCM dec.	Int	293	16	12
ADPCM enc.	Int	316	12	16
Filterbank	Float	75	1 024	1 024
fir2dim	Float	75	208	64
Matrix mult.	Int	28	8 192	4 096

Table I presents the benchmark programs used from the TACLe [22] and CHStone [11] benchmark suites. The amount of data transferred are also presented in Table I. Since we are performing WCET analysis, the TACLe benchmark suite is used for the benchmarking process. The CHStone benchmark code is only used to synthesize the ADPCM HwA, since this has already been prepared for synthesis, and the code of the two benchmarks have the same functionality.

For this paper, all the benchmarks have been changed to 32-bit integer data-types. Patmos does not contain a floating-

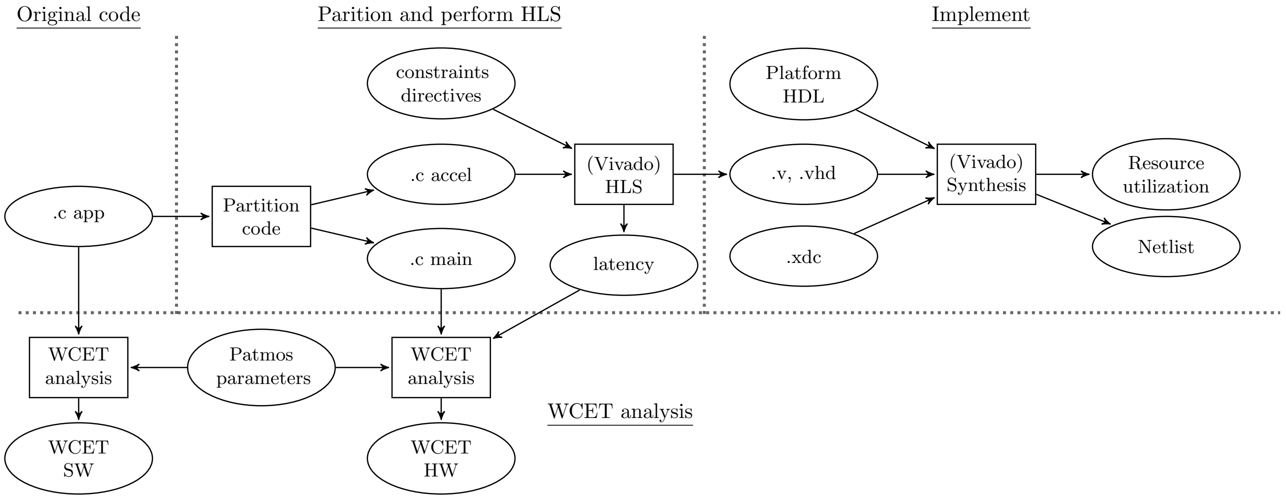


Fig. 2. An overview of the tool flow used to obtain the results of the paper.

point unit, and any WCET analysis involving floating-point operations are very pessimistic since the compiler generates code to handle floating-point operations. The matrix multiplication benchmark has been changed from a 10×10 matrix to a 32×32 matrix since the original size is deemed too small to show the benefit of hardware acceleration.

For all TACLe benchmark programs running on the Patmos CPU, the code has been optimized to utilize the local SPM for the main data structures. These data structures are the arrays of the programs which contain the input operands and the intermediate and final results. This reduces the pessimism of the WCET results for the software benchmarks, since the WCET tool is less pessimistic about the SPM than the caches for data. Whenever the HwA is used, the required data is written from the local SPM to the HwA's memory banks through the MCU, and the results are read back to the local SPM when it has finished.

We now briefly describe the benchmark programs.

ADPCM encoder/decoder: ADPCM (Adaptive Differential Pulse Code Modulation) implements the CCITT G.722 ADPCM algorithm for voice compression. It includes both encoding and decoding functions, selected before using the ADPCM HwA.

Filterbank: This benchmark implements a filterbank with several FIR filters for the processing of multiple frequency sub-bands.

fir2dim: This program performs 2-dimensional filtering, using the cross-correlation operation.

Matrix mult.: Matrix multiplication is one of the most fundamental operations in linear algebra, and it is therefore interesting to have this included in the paper.

V. Results

This section evaluates the HwAs and the associated hardware in terms of hardware cost and the ACET/WCET speed-up. All the results of our architecture were produced using Xilinx Vivado (v16.4) targeting the Xilinx Artix-7 FPGA (model XC7A100T-1CSG324C). The clock frequency of the system is 80 MHz, limited by the Patmos CPU. All synthesis properties were set to their defaults.

TABLE II
The utilization of the different hardware modules in terms of Flip-Flops (FFs), look-up tables (LUTs), DSP blocks (DPSs) and block-RAMs (BRAMs).

Entity	LUT	FF	BRAM	DSP
Patmos	4 931	3 602	8.5	4
MCU	215	215	0	0
ADPCM enc/dec.	5 165	4 798	1 (3)†	26
Filterbank	5 358	4 849	3.5 (4)†	12
fir2dim	2 038	2 108	0 (2)†	6
Matrix mult.	1 439	3 159	0 (9)†	12

†Number in parenthesis denote required number of memory banks.

A. Utilization

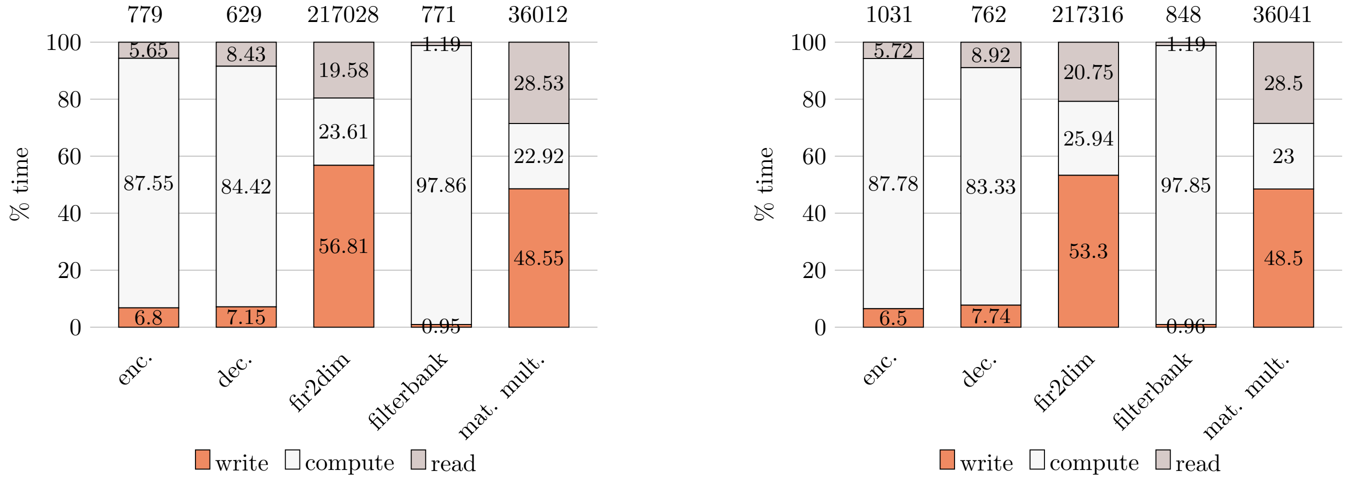
Utilization results are presented in Table II, providing the FPGA resource usage. The different HwAs each have their own requirements for the number of memory banks, and the BRAM utilization is thus shown in parenthesis to distinguish between the requirements for the HwAs themselves, and the amount of memory banks utilized outside the HwA.

We note that the MCU is very small compared to the HwAs and the CPU, adding little resource overhead to the system while providing the benefit of handling data distribution in hardware, which simplifies the WCET analysis.

Regarding the HwAs, the ADPCM and filterbank HwAs are comparable to the Patmos CPU itself, which is expected given the code complexity of the ADPCM and the utilized parallelization for the filterbank.

TABLE III
ACET AND WCET RESULTS (CLOCK-CYCLES) FOR SOFTWARE, HARDWARE AND THE CALCULATED SPEED-UPS WHEN USING THE HWAS.

Acc.	ACET (CC)			WCET (CC)		
	SW	HwA	Speed-up	SW	HwA	Speed-up
ADPCM enc.	9 010	779	11.57	12 426	1031	12.05
ADPCM dec.	7 714	629	12.26	10 083	762	13.23
Filterbank	6 253 891	217 028	28.81	15 279 271	217 316	70.31
fir2dim	3 695	781	4.73	4 155	848	4.90
Matrix mult.	508 653	36 012	14.12	530 534	36 041	14.72



(a) An overview of the distribution of the contributions to the ACET with the total execution time at the top.

(b) An overview of the distribution of the contributions to the WCET with the total execution time at the top.

Fig. 3. The contribution of the writes, computation and reads in percentages to the total execution time of the HwAs for the WCET and ACET cases.

B. Performance

WCET and ACET results for the HwAs are shown in Table III. The speed-ups are calculated by dividing the software measurement with the total measurement for the system using the HwAs. The ACET and WCET measurements have been divided into write, compute and read components, and the contribution of each of these factors to the total execution time is represented in Figure 3 as bar charts. For each entry in the bar chart, we have normalized to the total execution time for each HwA, as given in Table III and presented at the top of the bars.

For all benchmarks, speed-ups are observed for the ACET and WCET, but the filterbank benchmark stands out with a speed-up of around 70.31 times for the WCET. This is also the benchmark with the largest WCET for software, which is expected since this is the most computationally demanding benchmark. The 2D FIR-filter has the smallest speed-up overall, this is also the benchmark with the least computationally demanding code. The remaining benchmarks show speed-ups around 12-14 times both for ACET and WCET.

The difference between the total ACET and WCET for the HwAs is relatively small, around a couple of hundred cycles. The WCET results for the HwAs thus show very little pessimism. The SW ACET and WCET measurements have a much larger difference, in the range of 400 cycles for the 2D FIR-filter and up to 9 million cycles for the filterbank. The large difference between the ACET and WCET measurements for the filterbank leads to a larger speed-up for the WCET compared to the ACET, since we do not only see a computational speed-up but also a reduction in the pessimism.

For the remaining benchmarks, the WCET speed-up is always larger than the ACET speed-up. However, since the WCET speed-up is only slightly larger, there is room for measurement errors affecting any conclusions which can be drawn from these results.

We can also look at what contributes to the WCET and ACET when the HwAs are utilized, with the time to perform writes, reads and computations independently presented in Figure 3. This provides us with some knowledge of how the data transfer to the memory banks affects the total speed-up. The ratio between the three types of measurements is approximately the same for the ACET and WCET.

For the ADPCM and filterbank benchmarks, the computations are the main contributor to the WCET, and we would therefore see little benefit if the local SPM of Patmos could be used for the HwAs as well, since data transfer is relatively insignificant. For these benchmarks, greater acceleration may be possible by exploring other designs, since the computation is the main bottleneck.

For the matrix multiplication and 2D FIR-filter, the data transfer is the largest contributor, which reduces the speed-ups observed. For the 2D FIR-filter and matrix multiplication benchmarks, we could expect a larger speed-up if the local SPM was to be utilized since the data transfer accounts for the largest contributor to the total execution time. The local SPM would then have to be able to support memory banking, and we would have to consider its support for instructions, which take up a portion of the address-space of the local SPM besides the data.

For other applications, we may also just write directly from memory to the memory banks for the HwA, such that the local SPM would not be used to write from. This would, of course, result in a more pessimistic WCET results due to the involvement of the memory and cache.

VI. Conclusion

This paper presented a method for integrating HwAs generated using Vivado HLS with the Patmos CPU of the T-CREST platform. We have also presented some considerations for the functionality of associated hardware modules to improve the WCET performance of the system with the HwAs integrated.

The system with the different HwAs integrated was evaluated in terms of hardware cost and the ACET and WCET speed-ups. HwAs generated using HLS proved to give a speed-up of the WCET, showing the benefit of HLS in a real-time system implemented on a reconfigurable platform.

Source Access

The source code used for synthesis are 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_paper*. 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] K. Shin and P. Ramanathan, "Real-time computing: a new discipline of computer science and engineering," *Proceedings of the IEEE*, vol. 82, no. 1, pp. 6–24, 1994.

[2] P. Puschner, "Is worst-case execution-time analysis a non-problem? Towards new software and hardware architectures," *Proceedings of the second Euromicro International Workshop on WCET Analysis*, pp. 89–92, 2002.

[3] 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, Oct. 2015.

[4] Xilinx Inc., "Vivado high-level synthesis," 2017.

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

[6] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA '15*. New York, New York, USA: ACM Press, 2015, pp. 161–170.

[7] M. Sadri, C. Weis, N. Wehn, and L. Benini, "Energy and performance exploration of accelerator coherency port using Xilinx Zynq," in *Proceedings of the 10th FPGA world conference*. ACM Press, 2013, pp. 1–8.

[8] A. Tumeo, M. Monchiero, G. Palermo, F. Ferrandi, and D. Sciuto, "A pipelined fast 2D-DCT accelerator for FPGA-based SoCs," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2007, pp. 331–336.

[9] J. Cong, B. Grigorian, G. Reinman, and M. Vitanza, "Accelerating vision and navigation applications on a customizable platform," in *22nd IEEE International Conference on Application-specific Systems, Architectures and Processors*. ASAP, Sep. 2011, pp. 25–32.

[10] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, J. H. Anderson, S. Brown, and T. Czajkowski, "LegUp: High-level synthesis for FPGA-based processor/accelerator systems," *ACM/SIGDA international symposium on Field programmable gate arrays*, pp. 33–36, 2011.

[11] Yuko Hara, Hiroyuki Tomiyama, Shinya Honda, Hiroaki Takada, and Katsuya Ishii, "CHStone: A benchmark program suite for practical C-based high-level synthesis," in *2008 IEEE International Symposium on Circuits and Systems*. IEEE, 2008, pp. 1192–1195.

[12] B. Fort, A. Canis, J. Choi, N. Calagar, R. Lian, S. Hadjis, Y. T. Chen, M. Hall, B. Syrowik, T. Czajkowski, S. Brown, and J. Anderson, "Automating the design of processor/accelerator embedded systems with LegUp high-level synthesis," in *International Conference on Embedded and Ubiquitous Computing*. IEEE, Aug. 2014, pp. 120–129.

[13] C. K. Ng, S. Vyas, J. A. Shidal, R. Cytron, C. Gill, J. Zambreno, and P. H. Jones, "Improving system predictability and performance via hardware accelerated data structures," *Procedia Computer Science*, vol. 9, pp. 1197–1205, 2012.

[14] P. G. Zaykov, G. Kuzmanov, A. Molnos, and K. Goossens, "RTOS acceleration in an MPSoC with reconfigurable hardware," *Computers & Electrical Engineering*, vol. 53, pp. 89–105, Jul. 2016.

[15] M. Song, S. H. Hong, and Y. Chung, "Reducing the Overhead of Real-Time Operating System through Reconfigurable Hardware," in *10th Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD 2007)*, no. 8. IEEE, aug 2007, pp. 311–316.

[16] P. Puschner and A. Burns, "A review of worst-case execution-time analysis," *The International Journal of Time-Critical Computing Systems*, vol. 128, no. 18, pp. 115–128, 2000.

[17] P. Coussy and A. Morawiec, Eds., *High-Level Synthesis*. Springer Netherlands, 2008.

[18] R. Nane, V.-M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, F. Ferrandi, J. Anderson, and K. Bertels, "A survey and evaluation of FPGA high-level synthesis tools," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 10, pp. 1591–1604, Oct. 2016.

[19] P. Puschner, R. Kirner, B. Huber, and D. Prokesch, "Compiling for time predictability," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2012, vol. 7613 LNCS, no. 288008, pp. 382–391.

[20] "OCP official website." [Online]. Available: http://www.accellera.org/downloads/standards/ocp/

[21] C. Alias, A. Darte, and A. Plesco, "Optimizing remote accesses for offloaded kernels," *ACM SIGPLAN Notices*, vol. 47, no. 8, p. 285, Sep. 2012.

[22] H. Falk, S. Altmeyer, P. Hellinckx, B. Lisper, W. Puffitsch, C. Rochange, M. Schoeberl, R. B. Sorensen, P. Waegemann, and S. Wegener, "TACLeBench: A benchmark collection to support worst-case execution time research," *16th International Workshop on Worst-Case Execution Time Analysis (WCET 2016)*, vol. i, no. 2, p. 10, 2016.