Interfacing Hardware Accelerators to a Time-Division Multiplexing Network-on-Chip

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

***Abstract*—This paper addresses the integration of stateless hardware accelerators into time-predictable multi-core platforms based on time-division multiplexing networks-on-chip. Stateless hardware accelerators, like floating-point units, are typically attached as co-processors to individual processors in the platform. Our design takes a different approach and connects the hardware accelerators to the network-on-chip in the same way as processor cores. Each processor that uses a hardware accelerator is assigned a virtual channel for sending instructions to the hardware accelerator and a virtual channel for receiving results. This allows a stateless and possibly pipelined hardware accelerator to be shared in an interleaved fashion without any form of reservation, and this opens for interesting area-performance trade-offs. The design is developed with a focus on time-predictability, area-efficiency, and FPGA implementation. The design evaluation is carried out using the open source T-CREST multi-core platform implemented on an Altera Cyclone IV FPGA. The size of the proposed design, including a floating-point accelerator, is about two-thirds of a processor.**

I. Introduction

Hardware platforms for hard real time systems must exhibit a time-predictable behavior such that guarantees on worst-case execution time can be provided. In multi-core platforms this means that the networks-on-chip (NOCs) must support guaranteed service connections. One approach to this is time-division multiplexing (TDM), and examples of TDM-based NOCs are Æthereal and aelite [1], Nostrum [2], and Argo [3]. Our work specifically targets the latter, but our design can in principle be used with any TDM-based NOC.

The purpose of including hardware accelerators (HWAs) in a general purpose design is speed-up. In a real-time system, this results in a twofold speed-up: firstly the raw speed-up originating from using a HWA and secondly because it is possible to give a precise, and therefore tight, bound on execution time. Something that is generally very difficult or even impossible if the same function is implemented in software.

The limited production volume of hard real-time systems typically cannot amortize the development cost of an ASIC, and FPGA implementations are often preferred. The usage of the FPGA technology offers also more flexibility towards including relevant HWAs in the platform.

While the use of HWAs is a cost-efficient way to increase the performance, it also brings new challenges: from a software point of view, related to synchronization, programming models and scheduling; and from a hardware point of view, related to how the HWA is integrated into the rest of the system. The latter is even more challenging if the HWA is shared [4].

978-1-4673-6576-5/15/$31.00

In this paper we consider only stateless HWAs. HWAs of this class implement pure functions, where the result is computed entirely from the input – a set of operands and indication of what function to perform. Examples are floating-point, fast Fourier transform (FFT), discrete cosine transform, encryption/decryption, coding/encoding, etc. Such HWAs are typically attached as co-processors to individual processors cores in the platform, but due to the stateless property they can easily be shared among several processors.

This paper contributes a novel technique for integrating and sharing stateless HWAs in multi-core platforms based on a time-division multiplexing NOC for time-predictable applications. In our approach the HWAs are connected to the NOC in the same way as processor cores, as shown in Figure 1(a). Each processor that uses a HWA is assigned a virtual channel for sending instructions to the HWA and a virtual channel for receiving results. This allows a stateless and possibly pipelined HWA to be shared between cores in an interleaved fashion without any form of reservation and opens for interesting area-performance trade-offs.

We evaluate the performance and the hardware cost of our HWA integration technique using the T-CREST platform on an Altera Cyclone IV FPGA. The evaluation is carried out with an application that calculates a double-precision floating-point complex-to-complex FFT using a floating-point arithmetic accelerator. We provide speed-up results for a solution using a HWA compared to a pure software solution.

This paper is organized as follows: Section II provides general background on HWAs and presents related works. Section III gives an overview of the T-CREST platform in which we have tested our design. Section IV describes the overall design of the HWA-adapter and its implementation. Section V evaluates the design using the T-CREST platform and discusses the results and some properties of the integration technique. Section VI concludes the paper.

II. Background and Related Work

A HWA implements an operation in dedicated hardware as an alternative to a software routine. HWAs can be stateless where the result is computed from the input only, or stateful where the result depends from the input and some state information

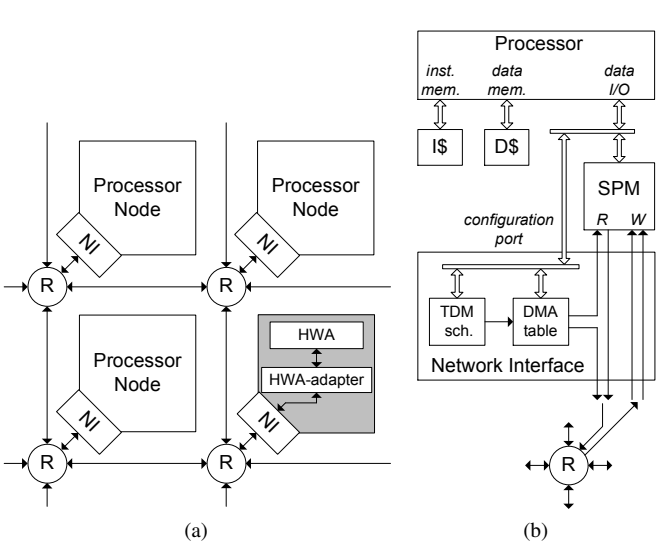

Fig. 1. (a) A 2×2 node fragment of the Argo NOC showing 3 processor nodes and a node containing a HWA. (b) Details of a network interface and a processor node.

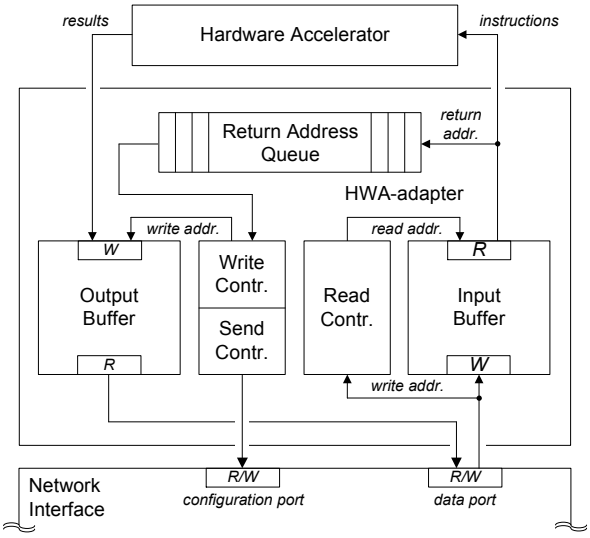

Fig. 2. The block diagram showing the HWA-adapter design and its interfaces towards the NOC and the HWA itself.

stored in the HWA. In this paper we consider only the former class. Orthogonal to this classification is the question of how the HWAs are integrated. Here is a large variety of solutions but most fall into one of the three categories described below.

In the first category the HWA is an extra execution unit that is attached directly to the CPU pipeline and it operates on variables in CPU's register file. This method is particularly advantageous for fine-grain HWAs with low latency. An example is the SuperH processor [5] where the FPU is part of the CPU pipeline and shares the CPU register file.

In the second category the HWA is connected to one or more CPUs using a dedicated bus (or NOC) for the HWA traffic. Examples are video/audio HWAs in the Philips Viper Nexperia and the STMicroelectronics Nomadik multi-core processors [6]–[8]. This approach is suitable for coarse or medium-grain HWAs with medium to high latency. Although advantageous from the speed-up point of view, this approach uses a dedicated bus/network leading to a significant increase of the system complexity and hardware cost.

In the third category the HWA is attached as a node to the already existent inter-processor NOC and it is considered as a peer of the other processor cores in the platform. This method is suitable for coarse or medium-grain HWAs with high-medium latency. An example is the Intel IXP2855 multi-core processor where cryptography HWAs are attaching to the NOC of the system [6], [9]. The method we consider in this paper falls in this category.

III. The T-CREST Platform

Our HWA-adapter has been designed for and evaluated using the opens-source T-CREST multi-core platform, but our design can in principle be used with any TDM-based NOC. The T-CREST platform [10] integrates time-predictable resources; such as processors, memories, and NOCs, all designed to reduce the complexity of worst case execution time (WCET) analysis. The T-CREST platform uses two NOCs: a NOC connecting all cores to a shared memory and a TDM-based NOC called Argo that supports message passing between cores [3].

Argo provides a set of virtual point-to-point channels to an application programmer. Through these channels blocks of data can be transferred from the local scratch pad memory (SMP) in a processor node into the SPM of a remote node. This data transfer is driven by DMA controllers, and each virtual channel has its own logical DMA controller in the sender end of the channel. A unique feature of Argo is the way in which the DMA-controllers are integrated with the TDM-scheduling in the NIs, as illustrated in Figure 1(b). Packets consist of two 32-bit data-words and a header word containing the route and the target address in the destination SPM. Incoming packets write directly into the destination SPM.

IV. Design and Implementation

Our key idea is to attach HWAs to the Argo NOC in the same way as a processor core, reusing the NI and replacing the processor and the SPM with a HWA-adapter and the HWA itself, as illustrated in the shaded tile in Figure 1(a). The HWA is typically a pre-existing standard stateless accelerator core. Details of the architecture and implementation of the HWA-adapter is explained below and illustrated in Figure 2.

Stateless HWAs are typically pipelined and have simple interfaces: an input port where they receive instructions (operands and indication of what operation to perform) and an output port where they deliver results. As the NOC serialize the arrival of packets, a stateless HWA can be shared by several processors without interference or need for reservation. For every processor that uses the HWA, the NOC is configured to provide a pair of (virtual) channels; one used for transmitting instructions from the processor to the HWA and one for transmitting results from the HWA back to the processor.

Our design piggybacks on the DMA-driven data transfer functionality of the NOC. This means that buffer space for instructions and result must be provided in the SPMs of the processors that use the HWA and in some memory-space in the HWA-adapter. It also means that the HWA-adaptor must be provided with a return-address for the result.

A block of data corresponding to an instruction or a result is typically larger than the payload of a NOC-packet, and a block of data is transmitted using multiple packets. As the TDM schedule interleaves timeslots for different virtual channels, it follows that the HWA-adapter receives packets sent from multiple processors in an interleaved fashion. Therefore, the HWA-adapter must be able to assemble and buffer instructions before they can be passed on to the HWA.

Below we briefly elaborate on the design of the different components in the HWA-adapter, shown in Figure 2:

The *input buffer* is a dual-port memory with write port size of 64 bits that corresponds to the payload size in a NOC-packet. The width of the read port corresponds to an entire instruction.

The *read control block* monitors the writing of data to the input buffer. When it detects that the last word of an instruction is received it reads the instruction and the return-address, and issues them respectively to HWA and to the return-address queue.

The *return-address queue* receives and holds the return-addresses from the input buffer and issues DMA set-up requests to the send controller. It is a FIFO-queue implemented using a dual-port memory and a read and a write pointer.

The *output buffer* receives results produced by the HWA and holds these until the word-by-word transmission through the NOC is complete. Like the input buffer, the output buffer is also a simple dual-port memory with different port sizes. The write port has the size of a result and the read port has the size of 64 bits.

The *write controller* is based on a simple incrementing counter and it manages the writing of result and return-address tuples into the output buffer. At the same time the *send controller*, implemented as a small Mealy state machine, is triggered to set up the relevant DMA in the NI. The DMA controller then sends the result back to the processor that requested the HWA operation.

V. Evaluation and Results

The presented technique is tested and evaluated in the T-CREST multi-core platform implemented on the Altera Cyclone IV FPGA (EP4CE115) on the Terasic DE2-115 development board. The configuration used in the test consists of a 2-by-2 bi-torus platform with four nodes. Three of them host Patmos processors, the last hosts the HWA. The HWA used in the test is a 15 stages pipelined double-precision FPU generated with FloPoCo [11] that executes addition, subtraction, and multiplication operations, selected with a dedicated input operand. The instruction size is four 64-bit words: one for the return-address, one for the operation selector (including some not utilized fields for future extensions) and two for the operands. Therefore, the memory-space in the HWA-adaper (input buffer) is divided in three segments of four 64-bit words; each segment belongs to a processor served by the HWA.

TABLE I
Hardware resource utilization for the Altera Cyclone IV FPGA implementation of the test case platform configuration.

	Logic cells	DSP	RAM (bits)
2 x 2 Platform	48 331	96	1 894 693
2 x 2 Argo NOC	7 161	0	0
4 SPMs	148	0	1 122 340
1 Patmos core	8 770	12	211 456
1 Network interface	899	0	0
1 HWA-adapter	2 206	0	180
1 FPU	3 354	48	881

Table I shows the FPGA hardware resource utilization in terms of logic cells, digital signal processing elements (DSP), and bits of memory (RAM) of the entire platform and of some relevant entities of the design. The results in Table I show that the hardware cost of a HWA-adapter is roughly a quarter of a Patmos processor (considering only logic cells), and that a node comprising a FPU-unit and a HWA-adapter is about two-thirds of a Patmos processor.

To evaluate the design in terms of average-case execution times speed-up, we have carried out a comparison between the execution time of two C applications that calculate an N points double-precision floating-point complex fast Fourier transform using the Cooley-Tukey algorithm [12]. The first application executes the algorithm in software utilizing the software floating-point functions (from the LLVM project [13] and Newlib [14]) for the floating-point addition, subtraction and multiplication needed by the algorithm, while the second application uses the FPU. An identical application runs in parallel on each core of the platform.

Table II shows the average-case execution times needed by the two applications to calculate the FFT of N points and the speed-up calculated as ratio between the execution times of the FFT function executed in software and using the FPU. The average on the execution times is calculated on 100 FFTs on randomly generated arrays of values. All the execution times are measured by the processor on the FPGA implementation of the platform.

According to the results reported in Table II, the design shows a speed-up from 7.8 for a 256 points FFT to 10.1 for 4096 points. As a general rule, to benefit from the use of a HWA, the speed-up has to be in the order of tens or hundreds depending on the application [15]. In our case, the usage of the FPU can be considered advantageous not only for the obtained speed-up, but also for the fact that the presented integration technique lead to WCET analysis simplifications, since some operations are executed in hardware with a static and pre-calculated WCET, as explained below.

Regarding the time-predictability of the design, the HWAs integrated with the presented technique inherit an important

TABLE II
AVERAGE-CASE EXECUTION TIMES, EXPRESSED IN 10^6 CLOCK CYCLES, OF AN N POINTS FFT EXECUTED IN SOFTWARE OR USING THE FPU.

N	Software	FPU	Speed-up
256	48.1	6.2	7.8
512	106.1	12.6	8.4
1 024	233.4	25.8	9.0
2 048	510.8	53.1	9.6
4 096	1 103.4	109.2	10.1

property from the TDM-based NOC: the total latency L_{tot} of an operation and the throughput of the HWA, as seen by a processor using it, are static and independent of other processors using the HWA. In other words, each processor sees the same performance, in terms of latency and throughput as it was alone. This property is particularly relevant for hard real-time systems, since it allows a simplification of the WCET analysis of applications running on different cores of the platform and sharing the HWA. The total latency can be calculated as a sum of three main latency contributes, as shown in 1.

$$L_{tot} = T_{NOCf} + T_{HWA\,adapter} + L_{HWA} + T_{NOCb} \quad (1)$$

T_{NOCf} and T_{NOCb} are the latencies needed by an instruction and a result, respectively, to traverse the NOC. These depend by the NOC size and by the TDM schedule [3]. $T_{HWA\,adapter}$ is the latency introduced by the HWA-adapter [16], and L_{HWA} is the latency of the HWA. Since the data path from the processor to the HWA-adapter and back to the processor is heavily pipelined, the latency of a single operation can be hidden by sending operations to the HWA in an interleaved fashion in a way similar to the loop unrolling procedure used in software to speed up loops execution. This means that a user of the HWA can have multiple outstanding computations leading to an increase of the effective throughput. Further explanations regarding the throughput and a formula for its calculation can be found in [16]. Using 1 for the presented hardware configuration, the calculated latency L_{tot} is 224 clock cycles. This value is static and pre-calculated and it can be directly used in the WCET analysis tools leading to a relevant analysis simplification.

Finally, we observe that having a payload size of a single NOC-packet smaller than the instruction size introduces the need for buffers and control logic to gather and assemble full instructions; and this affects the design complexity. Looking forward it may be relevant to consider increasing the number of words in a NOC-packet. If the payload of a single NOC-packet matches the size of an instruction, the input buffer and the associated control logic in the HWA-adapter can be reduced, also leading to a reduction of the total latency and to an increase of the effective HWA throughput.

VI. CONCLUSION

This paper presented a technique that provides a flexible and scalable interface for integrating stateless HWAs in time-predictable multi-core platforms using a message passing NOC based on time-division multiplexing. The combination of a TDM based NOC and stateless HWAs allows the accelerators to be shared among several processors in an interleaved fashion without any form of reservation. This gives the designer maximum flexibility when deciding on the number of HWAs needed to achieve a certain performance requirement. The design is developed with special focus on time predictability, area-efficiency, and FPGA implementation and it was evaluated in the T-CREST multi-core platform and implemented on an Altera Cyclone IV FPGA.

ACKNOWLEDGMENTS

This work was partially funded by the Danish Council for Independent Research | Technology and Production Sciences, Project no. 12-127600: Hard Real-Time Embedded Multiprocessor Platform (RTEMP).

REFERENCES

[1] K. Goossens and A. Hansson, "The Æthereal network on chip after ten years: goals, evolution, lessons, and future," in *Proc. of the 47th ACM/ IEEE Design Automation Conference (DAC2010)*, 2010, pp. 306 –311.

[2] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch, "The Nostrum backbone - a communication protocol stack for networks-on-chip," in *Proc. of the VLSI Design Conference*, Mumbai, India, Jan. 2004, pp. 693–696.

[3] E. Kasapaki, M. Schoeberl, R. B. Sørensen, C. T. Müller, K. Goossens, and J. Sparsø, "Argo: A real-time network-on-chip architecture with an efficient GALS implementation," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. accepted for publication, 2015.

[4] R. Wilson, "Attaching accelerators in multicore systems," 2014. [Online]. Available: https://www.altera.com/solutions/technology/system-design/articles/_2014/article-accelerators.html (visited on Apr. 9, 2015).

[5] F. Arakawa, T. Okada, T. Hayashi, O. Nishii, and T. Hattori, "An embedded processor core for consumer appliances with 5.6 GFLOPS and 73M polygons/s FPU," *Microprocessors and microsystems*, vol. 33, no. 4, pp. 254–259, 2009.

[6] W. Wolf, A. A. Jerraya, and G. Martin, "Multiprocessor system-on-chip (MPSoC) technology," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 27, no. 10, pp. 1701–1713, 2008.

[7] S. Dutta, A. Rieckmann, and R. Jensen, "Viper: A multiprocessor SOC for advanced set-top box and digital TV systems," *IEEE Design and Test of Computers*, vol. 18, no. 5, pp. 21–31, 2001.

[8] STMicroelectronics, "Nomadik: mobile multimedia application processor," Tech. Rep., 2003. [Online]. Available: http://pdf.datasheetarchive.com/datasheetsmain/Datasheets-46/DSA-22050.pdf (visited on Apr. 9, 2015).

[9] Intel Corporation, "Intel IXP2855 network processor," Tech. Rep., 2005. [Online]. Available: http://pdf.datasheetarchive.com/indexerfiles/Datasheet-081/DASF0021075.pdf (visited on Apr. 9, 2015).

[10] M. Schoeberl et al., "T-CREST: Time-predictable multi-core architecture for embedded systems," *Journal of Systems Architecture*, 2015, accepted for publication.

[11] 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.

[12] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Mathematics of Computation*, vol. 19, pp. 297–301, 1965.

[13] C. Lattner and V. Adve, "The LLVM instruction set and compilation strategy," CS Dept., Univ. of Illinois at Urbana-Champaign, Tech. Rep. UIUCDCS-R-2002-2292, Aug 2002.

[14] "Newlib: a C library intended for use on embedded systems." [Online]. Available: http://www.sourceware.org/newlib/ (visited on Apr. 9, 2015).

[15] Y. Watanabe and B. Moyer, *Chapter 13: Hardware Accelerators - Real World Multicore Embedded Systems*, 1st ed. Newton, MA, USA: Newnes, 2013, pp. 481–515.

[16] L. Pezzarossa, "Hardware Accelerators in Network-on-Chip Based Multi-Core Platforms," Master's thesis, Technical University of Denmark, Dept. of Applied Mathematics and Computer Science, 2014.