CoreVA-MPSoC: A Many-core Architecture with Tightly Coupled Shared and Local Data Memories

Johannes Ax, Gregor Sievers, Julian Daberkow, Martin Flasskamp, Marten Vohrmann, Thorsten Jungeblut, Wayne Kelly, Mario Porrmann and Ulrich Rückert

***Abstract*—MPSoCs with hierarchical communication infrastructures are promising architectures for low power embedded systems. Multiple CPU clusters are coupled using an Network-on-Chip (NoC). Our CoreVA-MPSoC targets streaming applications in embedded systems, like signal and video processing. In this work we introduce a tightly coupled shared data memory to each CPU cluster, which can be accessed by all CPUs of a cluster and the NoC with low latency. The main focus is the comparison of different memory architectures and their connection to the NoC. We analyze memory architectures with local data memory only, shared data memory only, and a hybrid architecture integrating both. Implementation results are presented for a 28 nm FD-SOI standard cell technology. A CPU cluster with shared memory shows similar area requirements compared to the local memory architecture. We use post place and route simulations for precise analysis of energy consumption on both cluster and NoC level using the different memory architectures. An architecture with shared data memory shows best performance results in combination with a high resource efficiency. On average, the use of shared memory shows a 17.2% higher throughput for a benchmark suite of 10 applications compared to the use of local memory only.**

***Index Terms*—C.1.4 Parallel Architectures; C.1.4.e Multi-core/single-chip multiprocessors; C.1.4.g On-chip interconnection networks; C.1.5.e Memory hierarchy**

I. Introduction

Modern application areas, like the Internet of Things (IoT) or mobile robotics, require high computational power combined with low energy and manufacturing costs in embedded systems. For this reason, we have developed a resource-efficient Multiprocessor System-on-Chip (MPSoC) with an appropriate trade-off between performance, energy consumption and chip area. An MPSoC architecture that meets these requirements is called a Many-Core architecture. Many cores are defined by a very high number of small-sized CPU cores, which are able to run applications with high resource efficiency. The integration of more and more processing cores on a single chip requires an efficient on-chip communication infrastructure. Due to the limited scalability of bus-based communication, state of the art MPSoCs with dozens of cores introduce communication networks, i.e., Network-on-Chips (NoCs). Hand in hand with the communication infrastructure goes the on-chip memory architecture, which also has a huge impact on performance and energy efficiency.

J. Ax, J. Daberkow, M. Flasskamp, M. Vohrmann, T. Jungeblut, M.Porrmann, and U. Rückert are with Center of Excellence Cognitive Interaction Technology (CITEC), Bielefeld University, Cognitronics and Sensor Systems, Bielefeld, Germany (e-mail: jax@cit-ec.uni-bielefeld.de).

G. Sievers is a former group member and works now at dSPACE GmbH, Paderborn, Germany

W. Kelly is with Queensland University of Technology, Science and Engineering Faculty, Brisbane, Australia

Manuscript received March 29, 2017; revised December 5, 2017.

The main focus of this paper is the comparison of different memory architectures and their interaction with the NoC for many core systems. Compared to traditional processor systems, lots of many cores feature a different memory management, which changes the requirements on memory and NoC infrastructure. Traditional processor systems use a memory hierarchy with several (private and shared) on-chip caches, external DRAM, and a unified address space. This allows for easy programming, but results in unpredictable memory access times. Additionally, the cache logic and the coherence handling require a high amount of chip area and power. Therefore, a lot of Many-Core systems omit data caches and use software-managed scratchpad memories instead, which provide a resource-efficient alternative [1]. For performance reasons, the scratchpad memories are tightly attached to each CPU and communication between CPUs is initiated by software. In [2] we showed that area and power consumption of a single CoreVA CPU's data memory increases by 10%, when using a cache instead of scratchpad memory. Due to cache coherence issues it can be expected that these values will even increase for a cache-based many core system. Additionally, software-managed scratchpad memories gives full control of data communication to the programmer or an automatic partitioning tool (cf. Section III-E) and allows for a more accurate performance estimation.

The many core architecture considered in this work is our CoreVA-MPSoC, which targets streaming applications in embedded and energy-limited systems. Examples for streaming applications are signal processing, video processing, or all kind of tasks processing an incoming data stream (e.g. encryption). To couple hundreds or thousands of CPUs, the CoreVA-MPSoC features a hierarchical interconnect architecture with a NoC interconnect that couples several CPU clusters. Each CPU cluster tightly couples several VLIW (Very Long Instruction Word) CPU cores via a bus-based interconnect using a common address space. This hierarchical interconnect allows for different memory architectures to tightly couple scratchpad memories to CPUs.

The first memory architecture is a local scratchpad memory tightly coupled to a single CPU with a low latency memory access time of one (write access) or two (read access) clock cycles. Within a cluster, CPUs can access each others local scratchpad memory in a Non-Uniform Memory Access (NUMA) fashion via a bus interconnect, which increases the access latency. In [3] we compared different interconnects for use in a CPU cluster. We analyzed different topologies (shared bus, crossbar) and bus standards (AMBA AXI, OpenCores Wishbone) for 8 to 32 CPU cores.

A second memory architecture is presented in [4], which shows the integration of a tightly coupled shared data scratchpad memory into a CPU cluster of our CoreVA-MPSoC. The tightly coupled shared data memory features a low latency of one or two clock cycles as well, when it gets accessed by any CPU within a cluster. Due to the uniform memory access (UMA) within a CPU cluster, copying data can be omitted, which results in high performance and high resource efficiency. However, the tight coupling of CPUs and shared memory requires a low latency, high performance interconnect. This limits the maximum clock frequency of the system for a high number of CPUs within a single cluster. Furthermore, multiple physical memory banks are needed to minimize congestions during concurrent memory accesses of multiple CPUs.

The contribution of this paper is the comparison of different memory architectures and their connection to the NoC via a novel network interface (NI) presented in [5]. This work extents our previous work in [4] and [5] by using the tightly coupled shared data memory within CPU clusters for NoC communication. Therefore, the NI is tightly coupled to the shared data memory of the CPU clusters to allow for a very efficient data transfer between different clusters. Furthermore, it supports sending requests from several CPUs in parallel and reduces the CPU load in a DMA controller like fashion. We analyze memory architectures with local data memory only, shared data memory only, and a hybrid architecture integrating both. Additionally, different MPSoC configurations with varying number of CPUs per cluster are analyzed.

This paper is organized as follows. At first Section II presents the related work and current state-of-the-art MPSoCs with comparable architectures. Section III gives a detailed overview of the CoreVA-MPSoC architecture including the different levels of the hardware hierarchy: CPU core, cluster, and NoC. Additionally, our software communication model and the compiler toolchain are presented. This includes our compiler for streaming applications, which performs an automatic partitioning of applications to hundreds of CPUs [6]. In Section IV we present the architecture of the tightly coupled shared data memory and its connection to the NoC. Implementation results using a 28 nm FD-SOI standard-cell technology are shown in Section V. Section VI presents energy results for low-level data transfers using different memory architectures. Performance results for low-level data transfers and different streaming application benchmarks are shown in Section VII. Finally, this work is concluded in Section VIII.

II. Related Work

Several energy efficient MPSoC architectures have been developed in both research and industry. However, there is not much research into the analysis of different memory architectures in combination with NoC-based MPSoCs.

Kalray presents the hierarchical MPSoC MPPA-256 [7]. It provides no global address-space on NoC level but uses a shared address-space within each CPU cluster. A CPU cluster composes multiple VLIW Cores with local cache memories working on a shared L2 memory within each cluster. CPUs of different clusters communicate via inter-process communication across the NoC. This cache-based approach allows for easier programming but decreases resource efficiency [1].

Adapteva's Epiphany [8] is another commercial MPSoC and a typical example for a many core architecture. The 64-Core chip E64G401 uses 32-Bit-RISC CPUs with 32 kB local level-1 (L1) scratchpad memory each. A global address-space allows the CPUs to directly and randomly access all memories of the MPSoC. The Epiphany does not introduce a cluster-level hierarchy but solely relies on NoC communication. Adapteva announced a new chip with 1024 64-Bit-CPUs [18] for summer 2017 with a very similar interconnect and memory architecture.

Apart from these MPSoCs there are other MPSoC architectures that feature shared L1 memories on cluster level. Rahimi et al. [9] connect several CPU cores with multi-banked tightly coupled data memory using a synthesizable logarithmic interconnect. Different CPU target frequencies and cluster configurations are compared. A CPU read request has a delay of a single clock cycle. This is achieved by using a shifted clock for the memories which complicates clock management and timing closure of the overall system. As the VLIW CPU used in this article features a memory read delay of 2 clock cycles, we can omit a shifted memory clock in our architecture. Besides synthesis results using a 65 nm technology, Rahimi et al. present architectures with up to 32 CPUs and 64 memory banks. The work of Rahimi et al. is extended in [19] by a controllable pipeline stage between the CPUs and memory banks to be more reliable and variation-tolerant. In [10] a shared L1 data cache is presented. Using the logarithmic interconnect network proposed by Rahimi et al., the best-case read latency is one clock cycle. The cache shows an area and power overhead compared to a shared L1 memory (5 % to 30 %), but allows for easier programming of the system. Gautschi et al. [11] present a cluster with four OpenRISC CPUs and an 8-banked shared L1 memory. Using 28 nm FD-SOI technology, the cluster operates at a near-threshold voltage of 0.6 V. The architecture of the 3-stage pipeline OpenRISC is improved to increase the energy efficiency by 50 %. Dogan et al. [12] present a multiprocessor system for biomedical applications with 8 cores, a shared L1 data memory with 16 banks and a shared instruction memory with 8 banks. The shared data memory can be configured to have banks that are accessed by a single CPU exclusively ("private banks"). For this, a memory management unit (MMU) is used to segment private and shared memory banks. To allow for power gating of unused memory banks, the MMU can be configured to access active banks only. The Plurality HyperCore architecture [13] integrates 64 RISC-like CPU cores, a hardware scheduler, and shared L1 data and instruction caches. All CPUs and the two caches are connected via a "smart interconnect". No detailed information about the architecture of the interconnect is provided, but Plurality states that multicasting the same data to several CPUs is supported. The proposed instruction cache has a size of 128 kB and the data cache a size of 2 MB.

The STMicroelectronics STHORM MPSoC [14] connects

TABLE I
COMPARISON OF DIFFERENT STATE-OF-THE-ART MPSOCS WITH RESPECT TO THE DATA MEMORY ARCHITECTURE

	Topology		L1 Memory			L2 Memory		DMA
	NoC	Cluster	Local	Shared	Cache	Shared	Cache	
MPPA-256 [7]	✓	✓	✓	-	✓	✓	-	✓
Epiphany [8]	✓	-	✓	-	-	-	-	✓
Rahimi et al. [9]	-	✓	-	✓	-	-	-	-
Kakoee et al. [10]	-	✓	-	✓	✓	-	-	-
Gautschi et al. [11]	-	✓	-	✓	-	-	-	-
Dogan et al. [12]	-	✓	-	✓	-	-	-	-
Plurality HyperCore [13]	-	✓	-	✓	✓	-	-	-
STHORM [14]	✓	✓	-	✓	-	-	-	✓
Rossi et al. [15]	-	✓	-	✓	-	-	-	✓
Loi and Benini [16]	-	✓	-	✓	-	✓	✓	✓
NVIDIA's Pascal GPU [17]	-	✓	-	✓	✓	✓	✓	-
CoreVA-MPSoC (this work)	✓	✓	✓	✓	-	-	-	✓

several CPU clusters via a Globally-Asynchronous Locally-Synchronous (GALS)-based NoC. Within a cluster, 16 CPUs can access a tightly coupled shared 256 kB 32-banked data memory via a logarithmic interconnect. The CPU cores have a private instruction cache. Each cluster integrates a 2-channel DMA controller that can access the shared memory and the NoC interconnect via an additional bus interconnect. The MPSoC is programmable via OpenCL or a proprietary native programming model.

Rossi et al. [15] optimize a similar architecture for ultra-low power operations. The PULP platform features one CPU cluster with four 32 bit CPUs, eight 2 kB L1 memory banks, and 16 kB L2 memory. In this architecture the DMA controller is tightly coupled to the shared L1 memory with two additional ports to allow access from an external 64 bit bus.

Loi and Benini [16] present a L2-Cache, which connects multiple CPU clusters with shared L1 memories to an external DRAM memory. Because of the uniqueness of the L2-Cache in the system no coherence handling is required. A configurable STBus interface is used to connect the CPU clusters with a latency of 9 to 16 clock cycles. The cache has a maximum clock frequency of 1 GHz in a 28 nm FD-SOI technology and requires 20 % to 30 % more area compared to a scratchpad memory with the same memory size.

In addition to MPSoC architectures many general purpose graphical processing units (GPUs) integrate a shared L1 memory. NVIDIA's Pascal GPU family features several partitions including one Pascal shader core with 32 shader ALUs. Two of these partitions share a dedicated 32-banked L1 data memory with 64 kB [17]. Additionally, these two partitions share a L1 texture cache and L1 instruction cache. The high-end Tesla P100 GPU contains 56 Pascal cores with a performance of 21.1 TFlops (half-precision) and consumes 300 W.

Table I classifies several state-of-the-art memory architectures and our CoreVA-MPSoC with respect to the data memory architectures. The CoreVA-MPSoC targets streaming applications in energy-limited systems. Streaming applications are characterized by continuous processing of a data stream via many different tasks. Due to the static data-flow between these tasks, the CoreVA-MPSoC uses software-managed scratchpad memories instead of caches, as they are used in [7], [10], [13], [16], and [17]. In contrast to the Epiphany [8], our CoreVA-MPSoC features a hierarchical communication infrastructure. Tightly coupled tasks benefit from the low access latency within a CPU cluster and the NoC allows to scale the number of these CPU clusters.

The CPU cluster of our CoreVA-MPSoC uses a tightly coupled shared memory architecture like [9] and [11]. Additionally, we tightly couple the NoC to this shared memory, in a similar manner to the concept of Rossi et al. [15]. As a new concept, our CoreVA-MPSoC allows the coexistence of tightly coupled local and shared data memories. The CoreVA-MPSoC can be configured with local data memory only, shared data memory only, or a hybrid architecture integrating both. These different memory architectures are analyzed in Sections V, VI and VII relating to their resource efficiency for streaming applications.

III. THE COREVA-MPSOC ARCHITECTURE

The CoreVA-MPSoC features a hierarchical interconnect architecture and targets applications for embedded and energy-limited systems. Our CoreVA CPUs can be easily extended by application specific instructions and dedicated hardware accelerators, as discussed in [20]. Generally, a CoreVA-CPU can be used for general purpose applications, so that all kinds of application domains can be addressed. However, currently our CoreVA-MPSoC and especially our automatic partitioning tool (cf. Section III-E) focuses on streaming applications, like signal processing, video processing, or all kind of tasks processing an incoming data stream (e.g. encryption). In [20] the CoreVA-MPSoC is presented as a platform for Software-Defined-Radio applications.

Its hierarchical infrastructure consists of a Network-on-Chip (NoC) interconnect that couples several CPU clusters (cf. Figure 1). Within each CPU cluster several VLIW CPU cores are tightly coupled via a bus-based interconnect.

A. CPU Core

The CPU core used in our MPSoC is the 32 bit VLIW processor architecture CoreVA, which is designed to provide a high resource efficiency [21]. It features a six-stage pipeline. VLIW-architectures omit complex hardware schedulers and leave the scheduling task to the compiler [22]. The CoreVA architecture allows to configure the number of VLIW slots,

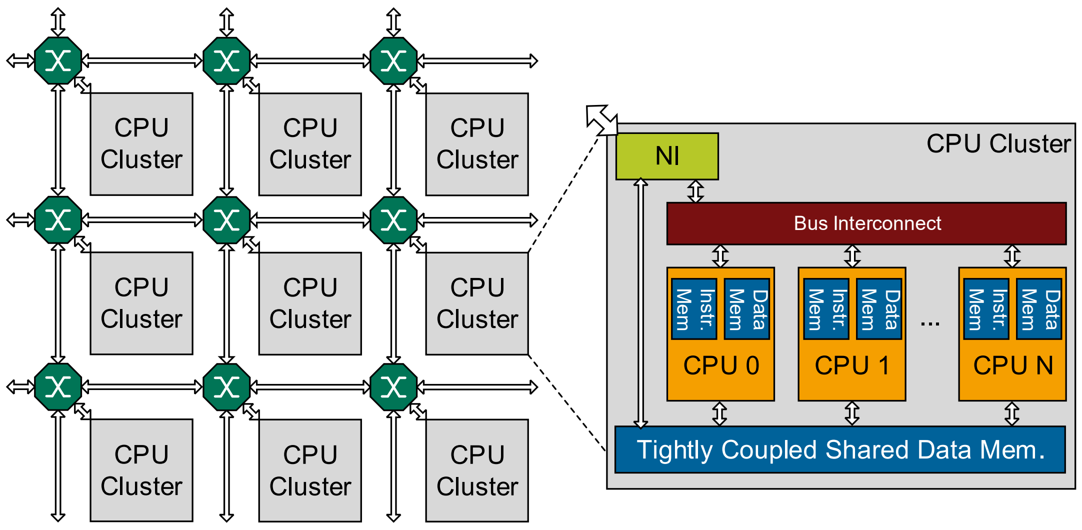


Fig. 1. Hierarchical CoreVA-MPSoC with a 3x3 mesh NoC.

their functional units, as well as the number of load/store units (LD/ST) at design-time. Functional units are arithmetic-logic-units (ALUs), dedicated units for multiply-accumulate (MAC) and division (DIV). Almost all instructions have a latency of one cycle, excluding memory loads, multiplications and MAC operations, which have a latency of two cycles. Additionally, both, ALU and MAC-units, support a 16 bit single instruction multiple data (SIMD) mode. Due to the highly configurable architecture it is possible to tailor the processor's performance to match the needs of a given application domain. As a typical Harvard architecture, the CPU features separated instruction and data memories (cf. Figure 1). In this work, we use a CoreVA CPU configuration with 16 kB instruction memory, two VLIW slots with one ALU each, as well as one MAC, DIV, and LD/ST unit. The local data memory is optional if a shared data memory is present on cluster level (cf. Section IV) and vice versa. In addition to the memory, the CoreVA CPU can be extended via a generic interface to connect to additional components via memory-mapped-IO (MMIO). These components are several dedicated hardware accelerators and the bus master interface to access the cluster interconnect. To avoid CPU stalls due to bus congestion, a small first in first out (FIFO) buffer is used to decouple CPU bus writes from the bus master interface. Additionally, the CPU integrates a bus slave interface to enable access to the memories from the bus.

To verify our physical implementation flow, performance and power models, two chip prototypes based on the CoreVA CPU architecture have been manufactured in a 65 nm process. The CoreVA-VLIW chip contains four ALU, two MAC, two DIV and two LD/ST units and is based on a conventional low power standard-cell library from ST Microelectronics [23]. This chip consumes about 100 mW at an operating clock frequency of 300 MHz. Additionally, a RISC-like ultra low power CPU, the CoreVA-ULP, is build using a custom standard cell library optimized for sub-threshold operation [24], [25]. Operation voltage range from 1.2 V down to 200 mV and frequency from 94 MHz down to 10 kHz. The CPUs lowest energy consumption per clock cycle of 9.94 pJ is observed at 325 mV and a clock frequency of 133 kHz. At this point the CPU core consumes only 1.3 µW.

B. CPU Cluster

A CPU cluster tightly couples several CoreVA CPUs which share a common address space within the cluster. In its basic configuration, each CPU can read from and write to the local data memories of the other CPUs via a bus-based interconnect (cf. Figure 1). CPUs are attached to the cluster's interconnect via a FIFO buffer to prevent penalty cycles on interconnect congestion for write operations. Bus standard (AXI, Wishbone), data bus width, and the topology (partial or full crossbar, shared bus) can be configured at design time. All of these different interconnect fabrics are evaluated in [3]. For this work, we use the AMBA AXI4 interconnect standard with a 32 bit or 64 bit data bus width. AXI defines separate channels for address- and data transfers. In addition, read and write (R/W) channels are separated and allow for parallel R/W requests even for a shared bus. This fact is a big advantage for our NoC and is picked up in the next Section III-C.

Register stages can be added to the interconnect to simplify place and route timing closure and to increase the maximum clock frequency of the MPSoC. Outstanding read requests are not supported as our VLIW architecture does not implement out-of-order execution. The minimum CPU to CPU read latency without register stages is 4 clock cycles. The shared bus implementation requires five arbiters in total (1 per channel), the crossbar interconnect two arbiters per slave (R/W address channel). Our interconnect does not support outstanding transactions and therefore we can omit extra arbitration for the data channels. We use a round robin arbitration scheme in this work. The extension of a CPU cluster with a tightly coupled shared data memory is presented in Section IV.

C. Network-on-Chip

To allow for large-scale MPSoCs with hundreds of CPU cores, a Network-on-Chip (NoC) can be used to connect multiple CPU clusters. The NoC features packet switching and wormhole routing. Packets are segmented into small flits each containing 64 bit payload data. Routers forward the flits along a path of network links. Each router has a configurable number of ports to allow for the implementation of different network topologies at design-time. In this work, a 2D-mesh topology (cf. Figure 1) in combination with a static XY-routing algorithm is used. A crossbar within each router switches between the ports and uses a round robin arbitration if multiple input ports need access to the same output link. Each input port is registered and uses a FIFO for buffering, which results in a latency of two clock cycles per router. To increase the average throughput, virtual channels can be introduced to each router, which implies dedicated FIFO buffers in each port.

In [26] we presented an asynchronous router design, which indicates a lower area and power consumption compared to the synchronous NoC. Additionally, this asynchronous NoC allows for a Globally-Asynchronous Locally-Synchronous (GALS) chip design.

Communication between two CPUs across the NoC is done via the Network Interface (NI, cf. Figure 1) presented in [5]. Each CPU cluster is addressable via a unique X and Y coordinate in the 2D-mesh topology of the NoC. Within a cluster a shared address space is used for all memories and components of the CPU cluster. The NI bridges between the address-based communication within a CPU cluster and the packet-based communication used by the routers in the network. Therefore, it provides an efficient flow control between CPU cores, while minimizing the CPU's execution time for this communication. For this, packet data is directly stored to and read from each CPU's local data memory. Hence, the CPUs can benefit from the low access latency of its local memory. Additionally, the NI acts like a DMA controller by sending and receiving data concurrent to CPU processing. Within the basic configuration, the NI is connected to the cluster interconnect via an AXI master and an AXI slave port. Due to the separated AXI channels for R/W transactions, sending and receiving of packets can be done concurrently by the NI's AXI master port. The AXI slave port is used by the CPUs to configure the NI. As a new approach the NI can be directly connected to the shared data memory within a cluster (cf. Section IV).

D. Communication Model

The CoreVA-MPSoC platform particularly targets streaming applications. Streaming applications consist of many different tasks which are connected via a directed data flow graph. An efficient communication model is required to allow communication between the tasks executed on different CPUs. Within the CoreVA-MPSoC a communication model with unidirectional communication channels is used. This approach promises more scalability and efficiency compared to shared memory concepts where the memory access can become the performance bottleneck [27].

In general, a task will read from one or more input channels and write to one or more output channels. Each channel manages one or more R/W data buffers. An application can request a buffer to write to (getWriteBuf) and a buffer to read from (getReadBuf). The synchronization is managed at the granularity of the buffer size. Requesting a buffer from a channel blocks the CPU if sufficient data has not yet arrived or if there is no free buffer available to write to. However, once a buffer is received from the channel, the application is free to randomly access any memory location within that buffer. Apart from this, no further synchronization is required. When a task has finished writing to or reading from a buffer, it must inform the task it interacts with, so that the buffer can be read by the reader (setWriteBuf) or reused by the writer (setReadBuf).

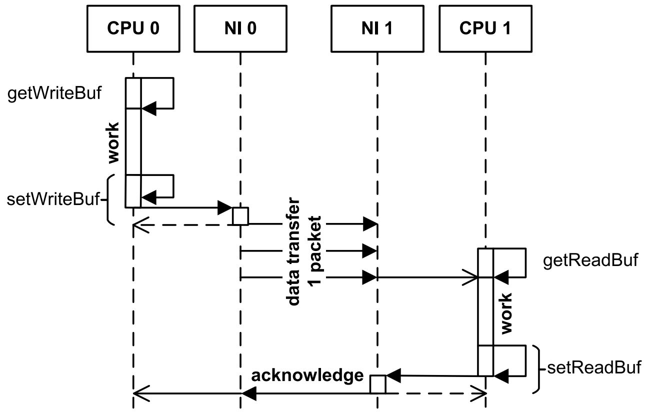


Fig. 2. Sequence diagram of the communication model for a NoC transfer.

There are three different types of channels: A (i) MemoryChannel for communication between tasks mapped to the same CPU. A (ii) ClusterChannel handles the communication between CPUs of the same cluster. A (iii) NoCChannel allows for communication between CPUs of different clusters via the NoC. Inter-CPU channels (ClusterChannel and NoCChannel) maintain at least two separate buffers so that latency can be hidden. This is called double buffering—one buffer can be filled while a previous buffer is still being read.

In case of using local data memory, the data buffers are allocated in the receiving CPU's memory to avoid high read latency of the bus. However, we prefer the shared memory for data buffers if it is available in the cluster, to minimize write latencies as well. For synchronization, one mutex pair per buffer is used. One mutex is used for getWriteBuf and setWriteBuf while the other one is used for getReadBuf and setReadBuf. The programming interface remains the same for a NoCChannel. The sequence of the communication model for a NoCChannel is shown in Figure 2. For a NoCChannel, data buffers are allocated at the sending and at the receiving cluster. A NoCChannel is implemented as a pair of ClusterChannels, one on the sending cluster and one on the receiving cluster. The NI in the sending cluster acts as a consumer and the NI in the receiving cluster acts as a producer.

E. Compiler Infrastructure

We developed an LLVM backend for the CoreVA architecture supporting VLIW and SIMD vectorization. Each CoreVA CPU can be programmed in C individually. However, programming a complex MPSoC and making effective use of its resources is a challenging task for the programmer. For this reason we have developed a parallelizing compiler for streaming applications to assist in programming the CoreVA-MPSoC [6], [28]. This compiler, called CoreVA-MPSoC compiler, processes applications written in the StreamIt language [29]. An application is described independently from the target architecture and can be represented by a structured data flow graph of its tasks. Our CoreVA-MPSoC compiler considers applications with a steady state and a static schedule at runtime, without the need of dynamic task scheduling. The efficient mapping of tasks to the processing elements of an MPSoC is a challenging task. A typical optimization goal is to find a placement for the application's tasks which maximizes the throughput of an application. Typically, optimization algorithms like simulated annealing or evolutionary algorithms are used to explore the large solution space for this issue [6], [30], [31].

Our CoreVA-MPSoC compiler utilizes an approach based on simulated annealing. During the partitioning process the compiler exploits three degrees of freedom to handle an application's data flow graph. Firstly, stateless tasks can be cloned to exploit data parallelism. Secondly, the compiler decides to which CPU each task is mapped to. Thirdly, the granularity of work done in each iteration can be increased to reduce the overhead of communication. The changes resulting from these decisions are called mutations and lead to a huge search space for the partitioning algorithm. All candidate partitions are analyzed regarding the expected performance but they are also checked for exceeded hardware limits. These can be upper limits for, e.g., memory consumption per CPU or the packet size in the NoC. Partitions that violate these restrictions are marked as invalid but they are not immediately rejected. This behavior enables the partitioning algorithm to explore more parts of the search space. Finally, the best partition is used to generate an individual C code file for each CPU of the MPSoC. For each task the computational part and the synchronization with other tasks are considered separately.

An important and time-consuming step of the partitioning algorithm is the accurate evaluation of thousands of different partitions of the application. The effectiveness of the algorithm is based on two conflicting goals: Firstly, the *faster* the evaluation of a single partition can be done the more different partitions can be analyzed. Accordingly, the chance is higher to derive a partition with optimal performance from the search space of possible partitions. Secondly, the more *accurate* a partition is evaluated the better is the compiler's chance to correctly select between solutions. Thus, the challenge is to balance *computation time* against *accuracy* and find that trade-off that leads to the best overall result independent of the optimization algorithm.

The performance evaluation of an application can be performed on various levels of abstraction at different levels of accuracy and speed. A simple analytical model might be evaluated faster compared to a post place and route simulation but achieves a lower accuracy. Our approach is called Simulation Based Estimation (SBE) and combines a single execution-based simulation and an analytic approach [28]. The execution time of each tasks is measured only once and independent of MPSoC configuration and task placement, because all CPUs in our MPSoC have the same properties like, e.g., clock frequency and number of VLIW slots (cf. Section III-A). However, our asynchronous NoC presented in [26] allows us to scale the frequency of individual CPU clusters in the future, to save energy and to handle on-chip variations. This will lead to different task runtime depending on the used CPU core. Methods presented, e.g., in [32] can help to identify the task runtime of specific CPUs due to on-chip variations. Future work on our compiler will include the handling of varying task runtime, by scheduling critical tasks to fast CPUs.

MPSoC configurations with the same number of CPUs can differ in many ways like topology, communication infrastructure and memory architecture. All these characteristics must be modeled for accurate estimations.

The performance of the application is limited by at least one bottleneck, which can be a CPU or a communication link with the highest load. A CPU's work consists of the computational work and synchronization costs of the executed tasks as well as additional waiting cycles if the communication infrastructure is overloaded. In addition, the non-bottleneck CPUs are blocked while waiting for the bottleneck processor(s) to complete its task. Computational work can be calculated for each CPU by summing up the predetermined runtime of all tasks partitioned onto this CPU. The runtime of all tasks can be determined in advance for a specific type of CPU by measuring the execution on a single CPU. Beside the placement of a task, the communication between two tasks can influence the performance as well. Depending on the used communication infrastructure, there are additional software costs (in terms of CPU cycles) for synchronizing and handling of communication channels in software. These costs differ if a channel connects two tasks on the same CPU or across a CPU cluster or the NoC. The communication overhead is estimated with respect to the target MPSoC, including the topology, communication bandwidth, and latency of the MPSoC as well as software costs for CPU-to-CPU synchronization. The software costs can automatically be determined in advance on our cycle accurate simulator.

Multiple tasks communicating concurrently on a communication link (e.g., a NoC link or a bus interconnect) may lead to a load that is exceeding the link's bandwidth. This can cause a stall of the sending or receiving CPU and therefore results in additional cycles. Our SBE models the communication infrastructure of the MPSoC to determine the load of all network links for a certain partition. The load of a network link is the sum of all communication channels that communicate via this link [33].

The available data memory is a critical resource in embedded multiprocessor systems since its size is typically limited due to chip costs and energy constraints. The type of communication channel between two tasks influences the amount of required memory as well. For instance, a channel inside a cluster allocates data buffers at the receiving CPU's memory whereas a channel across the NoC allocates data buffers also at the memory of the sending CPU (cf. Section III-D). Our SBE model estimates the consumption of data memory and enables the partitioning algorithm to reject partitions exceeding the physically available memory.

The load of all CPUs and communication links is estimated individually as described above. This combination enables our compiler to determine a fast but still accurate performance estimation of an application's partitions. Changing the placement of tasks or the MPSoC configuration requires only a simple reevaluation of the SBE model without executing or simulating the application. Our partitioning algorithm utilizes this estimation for balancing the load due to computation and communication across the MPSoC. The SBE model shows a speedup of 2848 and an average estimation error of 2.6 % compared to cycle accurate instruction set simulation [28].

Currently our SBE model is being extended to estimate the expected energy consumption of a partition.

More details about the programming model and the strategy of our CoreVA-MPSoC compiler are presented in [6].

IV. Tightly Coupled Shared Memory Architecture

In order to increase flexibility and bandwidth in terms of CPU-to-CPU communication, a multi-banked shared data memory is integrated as an addition to the CPUs local data memory. This memory is connected to every CPU in a single cluster and can be accessed easily via the CPU's MMIO interface. Additionally, the NI is connected to the shared memory via multiple ports in order to allow for simultaneous R/W accesses.

A shared memory conflict occurs if two consumers (CPU or NI) are claiming access to the same memory bank during the same clock cycle. In this case, a round robin arbitration is used and the losing consumer gets stalled until the next cycle. To minimize the amount of shared memory conflicts, the ratio between consumers and memory banks (the banking factor) needs to be high enough. Even though it has to be considered that a higher banking factor leads to tighter requirements regarding the shared memory's interconnect.

State of the art tightly coupled shared memory implementations do not integrate any local data memory and frequently choose a banking factor of 2, which has been shown to be a good choice [11], [12], [14]. We were already able to show, that the use of local data memory decreases the need for a high banking factor significantly [4]. In order to take this into account, we choose a factor of 2 as well, while considering only the amount of CPUs and ignoring the additional ports that belong to the NI. To decrease the probability of shared memory conflicts even further, we achieve a fine-grained interleaving of accesses by selecting the bank according to the least significant address bits.

A. Interconnect

Since our CoreVA CPU reads from the local data memory within two clock cycles, the same requirements are used for the shared memory in order to prevent additional CPU pipeline stages or stall cycles. Therefore, a fully combinational interconnect is required to be able to route a read request to the memory in the first and return the data to the CPU in the second cycle. In the case of a memory conflict, the stall signal needs to get routed back to the requesting CPU in the same cycle, which represents a reasonably long path. Thus, the shared memory interconnect becomes the critical path in our design in terms of timing requirements and needs to be considered in particular.

In [4] we integrated two different topologies regarding this interconnect. We consider a full crossbar implementation, where a single arbiter is used per memory bank and connected to every consumer (cf. Figure 3a). The arbiter stores the ID of the requesting consumer in order to route the answer of a read request to its correct destination.

The second interconnect is based upon a logarithmic Mesh-of-Trees (MoT) topology as proposed by [9]. This interconnect is constructed by two opposing logarithmic trees (cf. Figure 3b). The routing tree is used to route the request from the CPU to the corresponding memory bank. The request gets arbitrated by the second tree. In case of a read request, the routing nodes store the path of their last request to backtrack the data towards the correct consumer. The number of consumers and memory ports of both interconnects can be configured individually at design time.

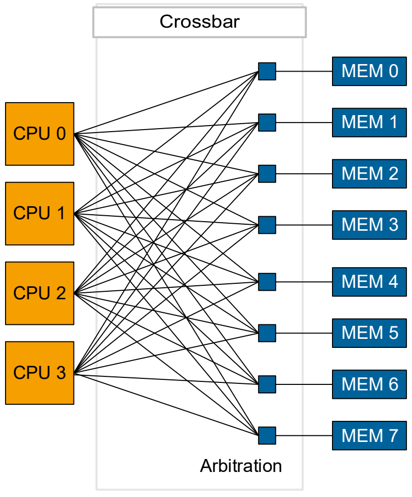
B. NI Connection

To allow for inter-cluster communication that involves utilization of the tightly coupled shared memory, a connection is not only established to the CPUs, but also to the NI (cf. Figure 4). Thereby, our NI extends the approach of additional ports to the shared memory's interconnect like the DMA controller in [15]. This allows the NI to perform R/W accesses directly to the shared memory which enables fast NoC communication. In contrast to [15] our hybrid memory architecture may also include local data memories which can be accessed by the NI via the bus interconnect as well. Whether an access needs to be addressed towards the shared memory or the bus interconnect is determined by the address of the specific memory access.

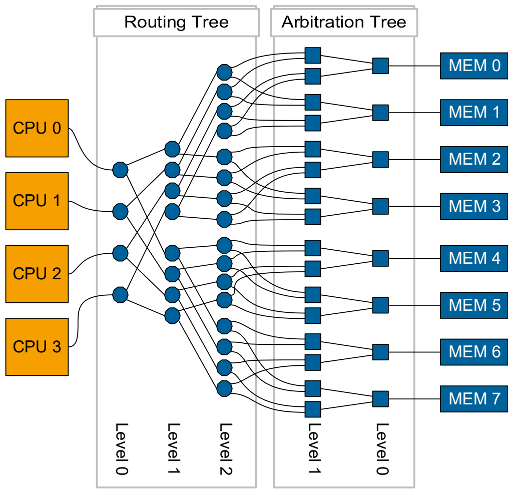
The NI is capable of sending and receiving data simultaneously. To retain this parallelism when accessing the shared memory, R/W accesses are performed via distinct ports. Additionally, the data width of our NoC-based communication is 64 bits. The shared memory on the other hand, is using a data width of only 32 bits in order to maintain homogeneity regarding the local data memory of the CPUs. To still achieve 64 bit accesses at the shared memory during a single clock cycle, the number of R/W ports are both doubled. This leads to a total quantity of 4 shared memory ports that are utilized by the NI.

V. Implementation Results

Physical implementation results of different hardware configurations are discussed in this section. Varying the hardware architecture leads not only to different area requirements but also to different maximum clock frequencies. For this reason, Section V-A discusses the maximum clock frequency of different shared memory configurations. Section V-B analyzes the area requirements of different CoreVA-MPSoC configurations by varying the number of CPUs per cluster and the data memory architecture. Both analysis, maximum clock frequency and



(a) Crossbar interconnect.



(b) Mesh-of-Trees interconnect [9].

Fig. 3. Shared memory interconnect topologies for 4 CPU cores and 8 memory banks.

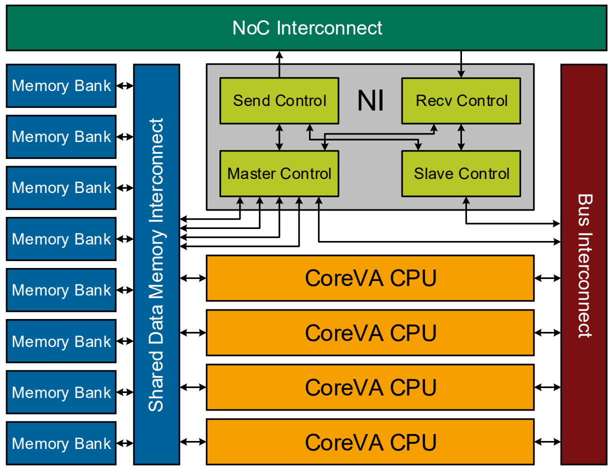


Fig. 4. Four ports connect the NI to the tightly coupled shared memory interconnect.

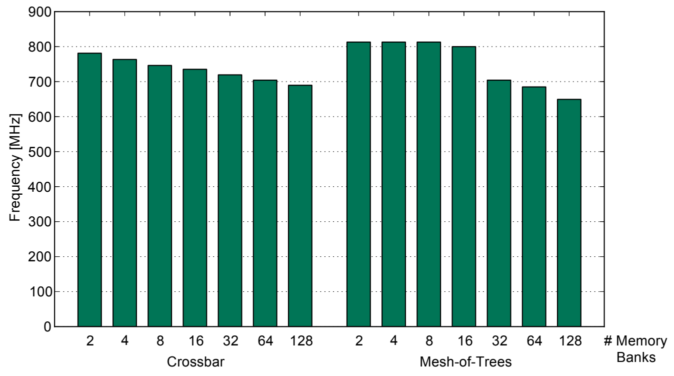


Fig. 5. Maximum clock frequencies for different memory interconnects, number of memory banks, and 16 kB memory per bank.

area requirements, are done on synthesis results of all different hardware configurations. Post place and route results for a specific MPSoC architecture are presented in Section V-C.

We utilize a 28 nm FD-SOI standard cell technology[1] with regular-VT standard cells for the analysis in this work. Our hierarchical standard-cell design-flow is based on Cadence Innovus Implementation System (formerly Encounter). Basic blocks of the analysis are hard macros of our CoreVA CPU in a 2 VLIW slot configuration, 16 kB instruction memory, and 32 kB, 16 kB, or no local data memory respectively. The macro with 16 kB and the macro without data memory feature a port to the shared memory.

A. Maximum Clock Frequency

In this Section we investigate the maximum clock frequency of different shared memory configurations within a CPU cluster. The CPU cluster integrates 8 CPU cores without local data memories. The number of memory banks varies from 2 to 128 and each shared memory bank has a size of 16 kB. For the MoT interconnect and configurations with up to 8 banks, the clock frequency is not limited by the shared memory subsystem but by the AXI interconnect (813 MHz, cf. Figure 5). The critical paths of all other considered configurations traverse the shared memory interconnect. In detail, the path goes from one CPU through the memory interconnect and the memory arbiter back to another CPU. The memory hard macros are not part of this critical path so area efficient high density memory blocks can be used instead of high speed blocks. The MoT with 16 banks has a slightly reduced maxium clock frequency of 800 MHz whereas the MoT configurations with 32 to 128 banks achieve only 704 MHz to 649 MHz. However, the crossbar interconnect shows a linear decrease of the maximum clock frequency from 781 MHz (2 banks) to 689 MHz (128 banks). For 2 to 16 banks, the MoT interconnect shows an advantage compared to crossbar, but for more than 16 banks the crossbar is faster. Cluster configurations with 4 and 16 CPUs benefit from the crossbar interconnect for higher bank counts as well.

[1]STMicroelectronics, 10 metal layer, Worst Case Corner: 1.0 V, 125°C

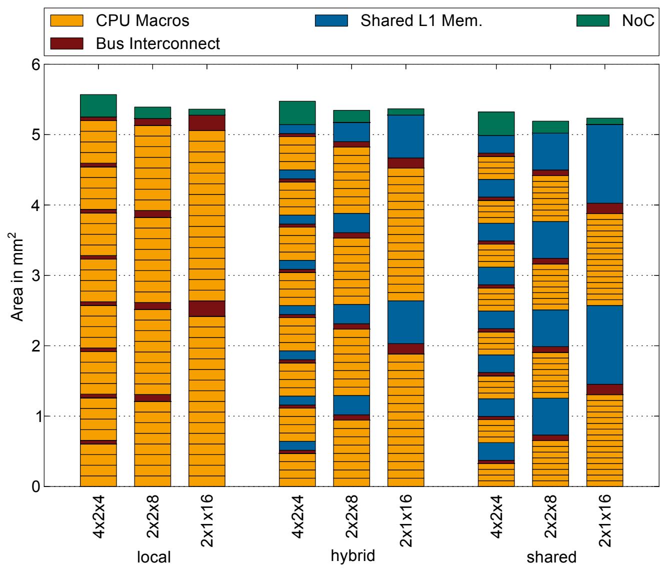


Fig. 6. Area requirements of different CoreVA-MPSoC configurations, with 32 CPUs and 256 kB data memory in total, and 650 MHz.

B. Area Requirements

This section analyzes different CoreVA-MPSoC configurations. For comparison of the different configurations the total MPSoC layout is set to 32 CPUs and 1 MB shared data memory. We vary the number of CPUs per cluster and the data memory architecture. The number of CPUs per cluster is set to 4, 8 or 16 which are connected via a 64 bit AXI crossbar in each configuration. These different MPSoC configurations are termed 4x2x4, 2x2x8 and 2x1x16, in which the NoC topology is indicated by the first two digits (e.g. 4x2 mesh) and the number of CPUs per cluster by the third digit.

To allow for a MPSoC with 32 CPUs, multiple of these CPU clusters are connected via the NoC. Additionally, we compare three different data memory architectures: 1) *local* memory attached to each CPU, 2) *shared* memory and 3) a *hybrid* memory architecture that integrates both. The number of memory banks for the shared memory is set to twice the number of CPUs, e.g. 32 memory banks for the configuration with 16 CPUs per cluster.

As shown in the previous Section V-A, the maximum clock frequency is limited by the shared memory interconnect for more than 8 memory banks. The AXI interconnect integrates two register stages so it does not limit the maximum clock frequency of our design, likewise the NoC. For better comparison of the area requirements, the target clock frequency is slightly reduced to 650 MHz for all analyzed MPSoC configurations. This allows the synthesis tool to use more power and area efficient cells, because timing is no longer critical. We use a crossbar memory interconnect for the following results. However, the memory interconnect of crossbar and MoT does not differ significantly in area requirements [4]. Area requirements of a CoreVA CPU macro are 0.15 mm^2 (32 kB local data memory), 0.12 mm^2 (16 kB local data memory), or 0.08 mm^2 (no local data memory).

The bars of *local* in Figure 6 show the MPSoC configurations without shared data memory and 32 kB local data memory per CPU. CPU macros contain the local data and instruction memories. This *local* MPSoC configuration features a full AXI crossbar (0.22 mm^2 for 16 CPUs) as bus interconnect to allow for concurrent communication of multiple CPUs. All other configurations use a shared AXI bus (0.15 mm^2 for 16 CPUs), because the shared data memory is used for data transfers between the CPUs. In this case the AXI bus is used for initialization, synchronization, and control of the CPUs and the NI only. Especially for a larger CPU count per cluster the area overhead of an AXI crossbar is significantly larger compared to a shared bus [3]. The area for the NoC components decreases from 0.33 mm^2 (4 CPUs per cluster) to 0.09 mm^2 (16 CPUs per cluster) with a decreasing number of CPU clusters. In contrast, the total bus area increases only slightly from the 4 CPU cluster to the 16 CPU cluster (both 0.44 mm^2).

All *hybrid* MPSoC configurations use the 16 kB CPU macro and 512 kB shared memory. With 512 kB local data memory in total, the CPUs require an area of 3.77 mm^2. The shared data memories vary from 8 to 32 memory banks with 8 kB each, depending on the CPU count per cluster. Total area requirements for all shared memory blocks and their interconnects increase from 1.02 mm^2 (4 CPUs per cluster) to 1.22 mm^2 (16 CPUs per cluster). The area requirement of the NoC is similar to the *Local Memory* configuration. Router area remains constant, only the NI area slightly increases from 0.20 mm^2 to 0.22 mm^2 due to the additional logic and ports for the shared memory.

The *shared* MPSoC configurations use the CPU macro without local data memory. To achieve larger shared data memories we increase the size of a bank to 16 kB. The area of the shared data memory within a single cluster varies from 0.25 mm^2 (4 CPUs per cluster) to 1.12 mm^2 (16 CPUs per cluster). An increasing amount of memory banks and hence a larger interconnect is the reason for this.

The lowest area requirements are achieved by the MPSoC configuration with 8 CPUs per cluster and shared data memory only. However, altogether there are only minor differences in area requirements between all MPSoC configurations. Thus it is required to consider the energy requirements (cf. Section VI) and performance (cf. Section VII) of the different MPSoC configurations.

C. Place and Route/Physical Implementation

Our hierarchical standard-cell design-flow includes three stages to build a full MPSoC (cf. Figure 7). The basic block for the place and route (P&R) layout used in this work is a 2 VLIW slot CPU macro with 16 kB instruction and 16 kB local data memory. Furthermore, we achieve the next level of hierarchy by combining several CPU macros to a cluster node macro. Multiple of these cluster node macros can be strung together on a top level P&R step to build a scalable MPSoC with a 2D-Mesh NoC. Due to the mesochronous NoC design (cf. Section III-C) the additional area and power consumption for the global clock tree is negligible on the top level P&R step.

Within this section post P&R results are shown for a cluster node layout including 4 CPUs using the hybrid data memory

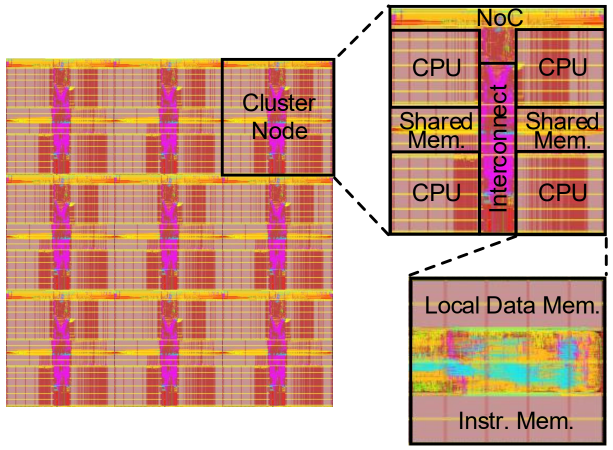


Fig. 7. 3x3 2D-Mesh MPSoC layout using the hierarchical design flow with cpu and cluster node macros

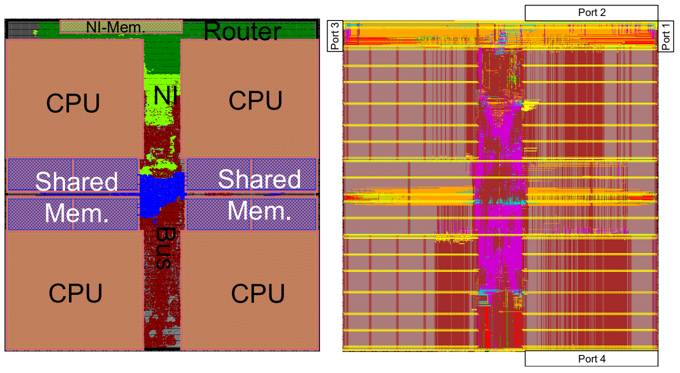


Fig. 8. Physical layout of a CPU cluster node with 4 CPU macros (16 kB local data memory each), 64 kB shared data memory and NoC components. Area requirements are 0.817 mm^2. Left: Placed design (shared memory and its interconnect is marked blue) Right: Routed design with NoC ports marked

architecture (cf. Figure 8). For the shared data memory the crossbar interconnect with 8 memory banks is used. The layout corresponds with the block diagram in Figure 10. Additionally, NoC components are included, like the NI and a router with connection ports to all four directions. More detailed P&R results of the CoreVA-MPSoC CPU clusters without the NoC but more CPUs have been published in [4].

Figure 8 shows the placed design with the memory interconnect and the shared memory macros highlighted in blue. The AXI interconnect is marked in red. All NoC components are highlighted in green whereas the NI is lime-green and the router dark-green. The routed design on the right hand side of Figure 8 shows the connections to all four external ports of the router. To achieve this the P&R tool can route across the CPUs due to the fact that the top two routing layers are not used by the CPU macros.

After synthesis, a CPU cluster with 8 memory banks shows a maximum clock frequency of 750 MHz (cf. Figure 5). However, after P&R the maximum clock frequency of the cluster node macro decreases to 700 MHz. The total area requirements of the node macro are 0.817 mm^2.

Post P&R power estimation results show a total power consumption of about 130 mW for the node macro. A detailed power and energy analysis by using back-annotated simulations is presented in the next section.

VI. Energy Results

Our CoreVA-MPSoC is designed for embedded low power systems. Therefore, we present results about the energy consumption of the different MPSoC configurations. Main focus of this analysis are the energy costs of data transfers between CPUs by comparing different memory architectures (shared vs. local data memory). Section VI-A presents energy results for communication on CPU cluster level regarding simple R/W transactions to the particular memory. Within Section VI-B a synchronized and more complex data transfer is analyzed for cluster and NoC communication.

For most 28 nm and smaller process technologies, the wire load models are skipped during synthesis step. This results in inaccurate energy results especially for interconnects (NoC and AXI) that are the main focus of this work. Hence, it is necessary to use a full P&R design and not only a post synthesis netlist for an accurate power analysis.

For all power analyses the post P&R layout of the node macro presented in the previous Section V-C is taken as a basis. It includes four CPUs with 16 kB local data memory each and a 64 kB shared data memory per cluster. All power analyses are based on a maximum clock frequency of 700 MHz. The dynamic power has been estimated by extracting the traces from a back-annotated simulation on the P&R layout, and performing the power simulation with Cadence Voltus.

All analysis in this section are done on a low-level of communication, which allows us to get very accurate energy results for single data transfers. In future work, these results can be used by our CoreVA-MPSoC compiler to optimize application for energy.

A. Energy Consumption for Read/Write Transactions on Cluster Level

Within a CPU-Cluster CPU-to-CPU communication is realized as a remote memory access. Thus this section presents the energy consumption of different memory accesses.

A synthetic benchmark, written in assembler, is used to access the CPU's local data memory (Local CPU0), the memory of another CPU within the same cluster Local CPU1, and the shared memory (Shared) respectively (cf. Figure 9). The benchmark runs on a single CPU while all other CPUs of the cluster are executing empty operations (NOPs) so as not to interfere with the energy measurement. Figure 9 shows the worst case energy consumption for reading or writing an all-one 32 bit data word (`0xFFFFFFFF`). Using other data words results in slightly lower energy consumption (e.g. 13% less for writing all-zeros). For comparison, Figure 9 also shows the energy consumption of a NOP.

The energy consumption is divided into a static (due to leakage current) and a dynamic portion (e.g., due to transistor switching). Total static power consumption of the CPU-Cluster is 13.2 mW and 4.6 pJ per cycle per CPU. The execution of a NOP requires 27.5 pJ with a dynamic portion of 22.8 pJ. About 20% of the overall energy is spent on the clock network. The

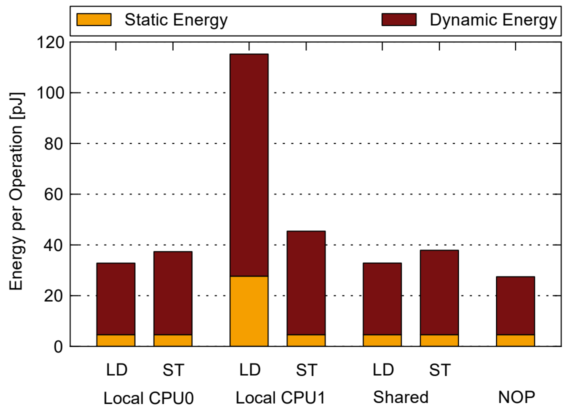


Fig. 9. Energy per memory operation in the CoreVA-MPSoC.

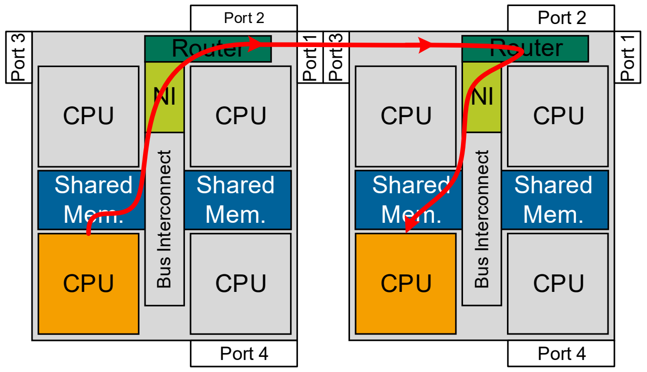


Fig. 10. Block diagram of two connected cluster node macros. Components involved in NoC communication using shared data memory are highlighted.

instruction memory requires about 12% of the overall energy as the benchmark requires the read of a 64 bit instruction every second cycle.

An access to the local memory (Local CPU0) requires 37.1 pJ (write) and 32.6 pJ (read). A write access to a remote memory (Local CPU1) results in 45.2 pJ per operation. This increase of 21.7% is due to the FIFO of the CPU's MMIO-interface and the register stages of the AXI interconnect. A read access lasts six cycles and requires 114.0 pJ. The reading CPU has to be stalled for five cycles till the requested data is read from the memory and send back via the AXI interconnect. These results show that read requests via the AXI interconnect should be avoided. In our CoreVA-MPSoC, we achieve this by using a tailored communication model (cf. Section III).

A request to the shared memory increases the energy consumption only slightly by 0.01% (read) and 1.5% (write) in comparison to a local memory access. In case of a bank conflict a CPU needs to be stalled for at least one clock cycle. However, with 32.63 pJ for a read and 37.68 pJ for a write, the access to the shared memory is more favorable than an access to the local memory of another CPU.

B. Energy Consumption on NoC Level

To determine the energy costs for CPU-to-CPU communication including NoC components we have implemented another low-level synthetic benchmark. This synthetic benchmark includes two simple tasks mapped to two CPUs of different clusters. One task produces random data and sends it via a NoC channel to the consuming task. Sending is done only once to isolate a single NoC transfer. We assume that the sending data is already available in the memory next to the sending CPU. The storing of data to the local memory is usually done during the work process of the specific task and does not need to be considered for communication costs. CPUs are set to sleep mode once they finished their synchronization and communication tasks. The sequence of the communication used in the benchmark is nearly identical to the sequence of Figure 2. It includes costs for synchronization but excludes the costs for the work process. We varied the data volume to be transfered between 8 B and 4 kB. For the back-annotated simulation we use a MPSoC design with two node macros which are attached to each other during the top level P&R step. The scenario is shown in Figure 10.

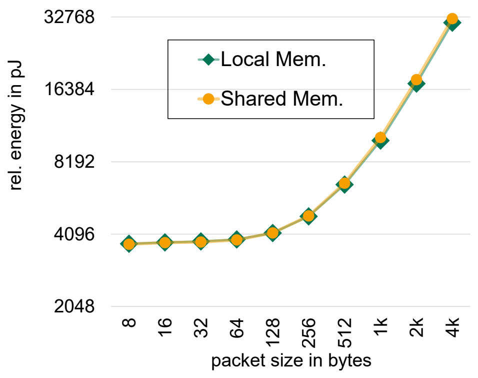
Results for power and energy consumption of NoC communication using the shared data memory are shown in Table II. The table presents the power consumption of the different components that are involved in the communication phase during a data transfer of 1 kB. These components are highlighted in Figure 10. The influence of bus interconnect on the energy consumption is negligible, because it is only used for a single write access to trigger the NI's sending job. As a reference, Table II shows the power consumption during their inactive or idle phase. In idle mode CPUs and memory blocks have a very low power consumption compared to the other components because they can be clock gated. However, components of the communication infrastructure have to be available for communication tasks of other CPUs. During the process of communication tasks the power consumption increases from 4.4 mW to about 26 mW for CPUs and from 1.1 mW to about 14 mW for the shared memory. Power consumption for router and NI increases only slightly from 4.5 mW to about 6 mW and from 2.4 mW to about 5 mW, respectively. The values for the individual components vary marginally, depending on their receiving or sending task (e.g sending NI 3.9 mW and receiving NI 5.5 mW).

Furthermore, Table II presents the energy requirements, considering the active time of the different components during communication. The third row shows the absolute energy requirement for the communication task of the individual components. The relative energy requirement is displayed in the fourth row. It excludes the energy that would have been consumed during the idle mode anyway. In future, these relative energy requirements can be used during the energy optimization process of our compiler (cf. Section III-E). It solely considers the additional energy costs for communication and synchronization while mapping tasks to different cores.

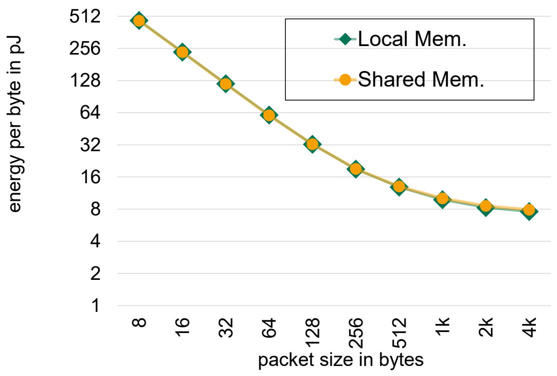
Figure 11 shows the relative energy costs for NoC communication using shared or local data memories transferring different packet sizes. The energy costs for shared and local data memory are very similar. Except for very large packets local memories have a minor advantage. The shared memory consumes 32.2 nJ for the transfer of 4 kB, which is 4 % more

TABLE II
POWER AND ENERGY CONSUMPTION FOR NOC TRANSFERS USING SHARED DATA MEMORY.

		Sending Cluster Node				Receiving Cluster Node			
		CPU	Shared Mem.	NI	Router	Router	NI	Shared Mem.	CPU
Power in mW	Idle	4.4	1.0	2.4	4.4	4.4	2.4	1.0	4.4
	Active	25.5	13.6	3.8	6.0	5.6	5.3	15.4	25.3
Energy in nJ	Absolute	1.7	3.0	0.8	1.3	1.2	1.2	3.4	1.6
	Relative	1.5	2.7	0.3	0.4	0.3	0.6	3.1	1.3



(a) Relative energy costs.



(b) Energy costs per byte.

Fig. 11. Relative energy costs for NoC communication using shared or local data memories transferring different packet sizes.

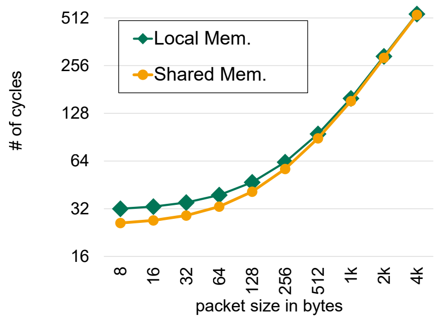


Fig. 12. Total latency in number of cycles for a NoC data transfer using shared or local data memories transferring different packet sizes.

compared to local memory. For small packet sizes the energy is dominated by the static part. This includes the synchronization tasks on the CPUs and the setup time of the NI to initialize the transfer. However, for packets larger than 256 bytes the energy increases nearly linear with the packet size (cf. Figure 11a). This behavior is shown more clearly in Figure 11b which shows the energy per byte decreasing from 466 pJ for 8 bytes to 8 pJ for 4 kB.

VII. PERFORMANCE RESULTS

The performance of NoC communication can be measured on different levels. Section VII-A shows results for the minimum latency of a single NoC data transfer by using the shared or the local data memory. Performance results on application level are presented in Section VII-B.

A. *Minimum Latency on NoC Level*

This section shows minimum latency results for a single NoC transfer. We used the same synthetic benchmark which has already been used for energy measurements (cf. Section VI-B). One task is producing data and sends it via a NoC channel to the consuming task. The measurement of the minimum latency starts when the sending CPU triggers the NI for the sending job and ends when the receiving NI signals the consuming CPU that all data has been arrived.

Figure 12 shows the minimum latency for a single NoC data transfer using shared or local data memories transferring different packet sizes. The shared memory outperforms the local memory for all packet sizes by saving about 6 clock cycles per transfer (e.g. 26 vs. 32 cycles for 8 bytes, cf. Figure 12). For packets larger than 2 kB the effect becomes negligible because of the NI's ability to burst data transfers.

B. *Performance on Application Level*

To determine the impact of shared memory on application performance, we use a set of 10 different streaming applications. These applications are derived from the StreamIt benchmark suite [29]. The benchmark suite features applications from the signal processing domain, sorting algorithms, and encryption. We use our MPSoC compiler (cf. Section III-E) to automatically partition the different tasks of the benchmarks to our CoreVA-MPSoC. In this work, the compiler is forced to optimize the partition of each application to achieve the best data throughput for the benchmarks. However, our compiler can optimize for minimum latency on application level, as well

Three different hierarchical CoreVA-MPSoC configurations with 32 CPUs and 1 MB data memory are considered: 4x2x4, 2x2x8 and 2x1x16. In addition, three different memory configurations are used. The *local* memory configuration features 32 kB local memory per CPU and no shared memory. The *hybrid* memory configuration uses 16 kB local memory per CPU and a balanced amount of shared memory per cluster. Which means 64 kB shared memory per cluster for the configuration with 4 CPUs per cluster (4x2x4), 128 kB for the configuration with 8 CPUs per cluster (2x2x8) and 256 kB for the configuration with 16 CPUs per cluster (2x1x16). For *hybrid*, the heap and the stack of each task are located in the local memory whereas the communication data buffers are stored in the shared memory. The *shared* memory configuration solely uses a balanced amount of shared memory per cluster without any local memories. For this configuration heap, stack, and communication buffers are allocated in the shared memory.

In [4] we have shown that the *hybrid* memory configuration outperforms other configurations within a single cluster. In that context we only considered a single way of partitioning for each application which results in identical memory access patterns. However, within this work the compiler is authorized to consider the particular memory configuration to benefit from the flexibility of a shared memory. Additionally, we analyze larger MPSoCs including NoC communication.

Figure 13 shows the speedup of data throughput of selected benchmarks for the considered MPSoC configurations compared to a single CPU. The memory architecture has almost no effect on algorithms like FFT, which have a very homogeneous structure. Some applications with special memory access pattern benefit from the shared data memory, especially when some CPUs require more memory than others (like LowPassFilter). In this case the compiler is able to allocate a larger amount of memory in the shared memory for the critical CPU, while other CPUs get less. Additionally, these kind of applications do also benefit from clusters including a higher number of CPUs and a larger shared data memory. The column *Average* shows the mean speedup of all 10 applications of our benchmark suite which shows that the use of shared data memory increases the throughput by 17.2 % (*shared* compared to *local*). Due to its higher flexibility the shared only configuration shows slightly better results compared to the hybrid configuration (4 % in average). However, for some special memory access patterns *hybrid* outperforms *shared* due to bank congestions of the shared data memory. In this case the hybrid configuration benefits from fewer accesses to the shared data memory because heap and stack of each CPU are stored in their local memories.

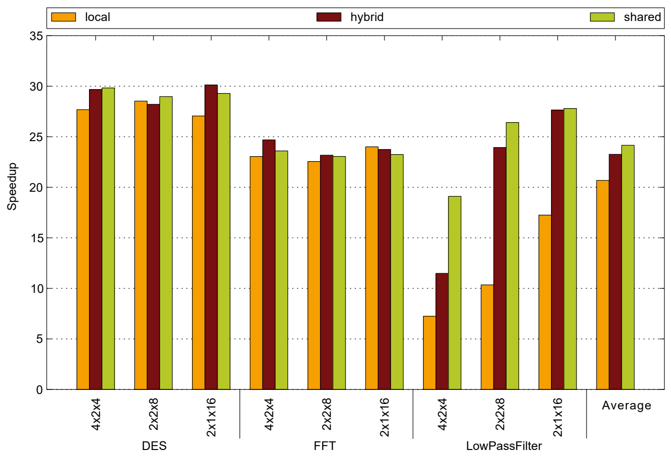


Fig. 13. Speedup of MPSoC configurations with different number of receive channels per NI compared to a single CPU. Average over ten applications and three MPSoC configurations.

VIII. Conclusion

This work investigates different memory architectures for our CoreVA-MPSoC, which targets streaming applications in embedded and energy-limited systems. Due to the static data-flow of streaming applications, the CoreVA-MPSoC uses software-managed scratchpad memories instead of caches. The CoreVA-MPSoC has a hierarchical architecture with a NoC interconnect coupling several CPU clusters. We analyzed different memory topologies for tightly coupled data memory within a CPU cluster. A CPU cluster can integrate local data memories, shared memory, or a hybrid architecture that utilizes both. Particularly this work introduces the efficient coupling of a cluster's shared memory to the network interface (NI) for highly efficient NoC communication.

Physical implementation results utilizing a 28 nm FD-SOI standard cell technology show only minor differences in area and energy requirements between the use of tightly coupled shared and local data memory architectures. However, shared memory outperforms local memories for NoC communication in terms of application throughput. The NoC communication benefits from a lower access latency to the shared memory. Additionally, the applications benefit from more memory allocation flexibility introduced by the shared memories. On average, this results in about 17.2 % higher throughput for a benchmark suite of 10 applications compared to local memory only.

Overall, hierarchical MPSoCs with a software-managed shared data memory on cluster level and a tightly coupled NoC are a best choice architecture for streaming applications in embedded systems. For future work we plan to extend our CoreVA-MPSoC compiler in a way that it optimizes the combined utilization of local and shared memory. If possible heap and/or stack may be allocated in a small local memory or in the shared data memory otherwise.

Acknowledgment

This work was funded as part of the DFG Cluster of Excellence Cognitive Interaction Technology 'CITEC' (EXC 277), Bielefeld University and the BMBF Leading-Edge Cluster "Intelligent Technical Systems OstWestfalenLippe" (it's OWL), managed by Project Management Agency Karlsruhe (PTKA).

References

[1] R. Banakar *et al.*, "Scratchpad Memory: Design Alternative for Cache On-Chip Memory in Embedded Systems," in *Int. Symp. on Hardware/Software Codesign (CODES)*. ACM Press, 2002, pp. 73–78.

[2] T. Jungeblut *et al.*, "Design Space Exploration for Memory Subsystems of VLIW Architectures," in *5th IEEE International Conference on Networking, Architecture, and Storage*, 2010, pp. 377–385.

[3] G. Sievers *et al.*, "Evaluation of Interconnect Fabrics for an Embedded MPSoC in 28nm FD-SOI," in *ISCAS*, 2015.

[4] ——, "Comparison of Shared and Private L1 Data Memories for an Embedded MPSoC in 28nm FD-SOI," in *International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*. IEEE, sep 2015.

[5] J. Ax *et al.*, "System-Level Analysis of Network Interfaces for Hierarchical MPSoCs," in *International Workshop on Network on Chip Architectures (NoCArc)*. ACM Press, 2015.

[6] W. Kelly *et al.*, "A Communication Model and Partitioning Algorithm for Streaming Applications for an Embedded MPSoC," in *Int. Symp. on System on Chip (SoC)*. IEEE, 2014.

[7] B. D. de Dinechin *et al.*, "A clustered manycore processor architecture for embedded and accelerated applications," in *2013 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, sep 2013.

[8] Adapteva, "E64G401 Epiphany 64-Core Microprocessor Datasheet," 2014, http://www.adapteva.com/.

[9] A. Rahimi *et al.*, "A Fully-Synthesizable Single-Cycle Interconnection Network for Shared-L1 Processor Clusters," in *DATE*. IEEE, 2011.

[10] M. R. Kakoee *et al.*, "A Multi-Banked Shared-L1 Cache Architecture for Tightly Coupled Processor Clusters," in *Int. Symp. on System on Chip (SoC)*. IEEE, 2012.

[11] M. Gautschi *et al.*, "Customizing an Open Source Processor to Fit in an Ultra-Low Power Cluster with a Shared L1 Memory," in *Great Lakes Symp. on VLSI (GLSVLSI)*. ACM Press, 2014, pp. 87–88.

[12] A. Y. Dogan and others., "Multi-Core Architecture Design for Ultra-Low-Power Wearable Health Monitoring Systems," in *Design, Automation & Test in Europe - DATE*. IEEE, 2012, pp. 988–993.

[13] J. Turley, "Plurality Gets Ambitious with 256 CPUs," *Microprocessor Report*, 2010.

[14] L. Benini *et al.*, "P2012: Building an Ecosystem for a Scalable, Modular and High-Efficiency Embedded Computing Accelerator," in *Design, Automation & Test in Europe (DATE)*. IEEE, 2012, pp. 983–987.

[15] D. Rossi *et al.*, "A 60 GOPS/W, -1.8V to 0.9V Body Bias ULP Cluster in 28nm UTBB FD-SOI Technology," *Solid-State Electronics*, vol. 117, pp. 170–184, mar 2016.

[16] I. Loi and L. Benini, "A Multi Banked - Multi Ported - Non Blocking Shared L2 Cache for MPSoC Platforms," in *Design, Automation & Test in Europe (DATE)*. IEEE, 2014.

[17] D. Kanter, "Nvidia Hits HPC First With Pascal," *Microprocessor Report*, 2016.

[18] A. Olofsson, "Epiphany-V: A 1024 processor 64-bit RISC System-On-Chip," Adapteva, Tech. Rep., 2016.

[19] M. R. Kakoee *et al.*, "A Resilient Architecture for Low Latency Communication in Shared-L1 Processor Clusters," in *DATE*. IEEE, 2012, pp. 887–892.

[20] G. Sievers *et al.*, *The CoreVA-MPSoC: A Multiprocessor Platform for Software-Defined Radio*. Cham: Springer International Publishing, 2017, pp. 29–59. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-49679-5_3

[21] B. Hübener *et al.*, "CoreVA: A Configurable Resource-Efficient VLIW Processor Architecture," in *Int. Conf. on Embedded and Ubiquitous Computing (EUC)*. IEEE, 2014, pp. 9–16.

[22] J. A. Fisher, "Very Long Instruction Word architectures and the ELI-512," in *ISCA'83*. ACM, 1983, pp. 140–150.

[23] G. Sievers *et al.*, "Design-Space Exploration of the Configurable 32 bit VLIW Processor CoreVA for Signal Processing Applications," in *2013 NORCHIP*, 2013.

[24] S. Lütkemeier *et al.*, "A 65 nm 32 b Subthreshold Processor With 9T Multi-Vt SRAM and Adaptive Supply Voltage Control," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 1, pp. 8–19, 2013.

[25] M. Vohrmann *et al.*, "A 65 nm Standard Cell Library for Ultra Low-power Applications," in *ECCTD*. IEEE, 2015.

[26] J. Ax *et al.*, "Comparing synchronous, mesochronous and asynchronous NoCs for GALS based MPSoC," in *IEEE 11th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC-17)*, 2017.

[27] T. Marescaux *et al.*, "The Impact of Higher Communication Layers on NoC Supported MP-SoCs," in *International Symposium on Networks-on-Chip (NOCS)*. IEEE, may 2007, pp. 107–116.

[28] M. Flasskamp *et al.*, "Performance estimation of streaming applications for hierarchical MPSoCs," in *Workshop on Rapid Simulation and Performance Evaluation (RAPIDO)*. ACM Press, 2016.

[29] W. Thies *et al.*, "StreamIt: A Language for Streaming Applications," in *International Conference on Compiler Construction (CC)*. Springer, 2002, pp. 179–196.

[30] Z. Wang and M. F. O'Boyle, "Partitioning Streaming Parallelism for Multi-cores," in *International Conference on Parallel Architectures and Compilation Techniques (PACT)*. ACM Press, sep 2010, p. 307.

[31] L. Thiele *et al.*, "Mapping Applications to Tiled Multiprocessor Embedded Systems," in *Seventh International Conference on Application of Concurrency to System Design (ACSD 2007)*. IEEE, jul 2007, pp. 29–40.

[32] A. Rahimi *et al.*, "Task scheduling strategies to mitigate hardware variability in embedded shared memory clusters," in *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*. IEEE, 2015, pp. 1–6.

[33] J. Ax *et al.*, "An Abstract Model for Performance Estimation of the Embedded Multiprocessor CoreVA-MPSoC (v1.0)," Universität Bielefeld, Tech. Rep., 2015.

Johannes Ax received the Dipl.-Ing. degree (equiv. M.Sc.) in electrical engineering from the University of Paderborn, Germany, in 2011. Since 2011 he worked as a research assistant at the Cognitronics and Sensor Systems group of the Center of Excellence Cognitive Interaction Technology (CITEC), Bielefeld University. His research focuses on the development of multiprocessor-system-on-a-chip (MPSoC) for embedded applications with focus on NoC interconnects.

Gregor Sievers received the Dipl.-Ing. degree (equiv. M.Sc.) in electrical engineering from the University of Paderborn, Germany, in 2009. In 2016 he received the Ph.D. degree in electrical engineering from Bielefeld University, Germany, for his work on the design-space exploration of tightly coupled multiprocessors.

Currently, he works as an application engineer at dSPACE GmbH, Paderborn, Germany.

Julian Daberkow is a masters student in intelligent systems at Bielefeld University, Germany, where he received the B.Sc. in cognitive informatics in 2014. He is part of the research group Cognitronics and Sensor Systems, CITEC, Bielefeld University. His work focuses on the development of embedded multiprocessor-system-on-a-chip (MPSoC) architectures.

Martin Flasskamp received the Dipl.-Ing. degree (equiv. M.Sc.) in electrical engineering from the University of Paderborn, Germany, in 2012. Currently, he is a member of the research group Cognitronics and Sensor Systems, CITEC, Bielefeld University. His research focuses on the development of multiprocessor-system-on-a-chip (MPSoC) for embedded applications.

Marten Vohrmann received the Dipl.-Ing. degree (equiv. M.Sc.) in electrical engineering from the University of Paderborn, Germany, in 2012. Currently, he is a member of the research group Cognitronics and Sensor Systems, CITEC, Bielefeld University. His research focuses on the development of low power circuits for ASIC designs.

Thorsten Jungeblut Dr. Thorsten Jungeblut graduated as Dipl.-Ing. (equiv. M.Sc.) in electrical engineering at the University of Paderborn, Germany, in 2005. In 2011 he received the Ph.D. degree in electrical engineering from Bielefeld University, Germany, for his work on the design-space exploration of resource-efficient VLIW processors. Since 2011, he is head of the team Nanoelectronics in the research group Cognitronics and Sensor Systems, CITEC, Bielefeld University, Germany. His research focuses on the development of resource-efficient multiprocessor architectures.

Wayne Kelly received a B.Sc. Honours degree majoring in Computer Science from the University of Queensland in 1989. In 1996 he completed his Ph.D in Computer Science from the University of Maryland, College Park. He is a Senior Lecturer at the Queensland University of Technology in Australia. His research interests include Static Program Analysis and Reasoning about Parallelism.

Mario Porrmann is Academic Director in the research group Cognitronics and Sensor Systems, CITEC, Bielefeld University. He graduated as Diplom-Ingenieur in Electrical Engineering at the University of Dortmund, Germany, in 1994. In 2001 he received the Ph.D. degree in electrical engineering from the University of Paderborn, Germany. From 2001 to 2009 he was "Akademischer Oberrat" and from 2010 to March 2012 Acting Professor of the research group System and Circuit Technology at the Heinz Nixdorf Institute, University of Paderborn. His main scientific interests are in on-chip multiprocessor systems, dynamically reconfigurable hardware and resourceefficient computer architectures.

Ulrich Rückert received the Diploma degree in computer science and the Dr.-Ing. degree in electrical engineering from the University of Dortmund, Germany, in 1984 and 1989, respectively. From 1985 to 1994 he worked at the Faculty of Electrical Engineering, University of Dortmund, and at the Technical University of Hamburg-Harburg, Germany. In 1995 he joined as a Full Professor the Heinz Nixdorf Institute at the University of Paderborn, Germany, heading the research group System and Circuit Technology and working on massive-parallel and resourceefficient systems. Since 2009 he is a Professor at Bielefeld University, Germany heading the Cognitronics and Sensor Systems group of the CITEC. His main research interests are now bio-inspired architectures for nanotechnologies and cognitive robotics.