IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier

Towards a Modular RISC-V based Many-Core Architecture for FPGA Accelerators

AHMED KAMALELDIN[1], (Graduate Student Member, IEEE), SALMA HESHAM[1], (Member, IEEE) AND DIANA GÖHRINGER[1, 2], (Member, IEEE)
[1]Adaptive Dynamic Systems, Technische Universität Dresden, Germany
[2]Center for Tactile Internet with Human-in-the-Loop (CeTI), Technische Universität Dresden, Germany

Corresponding author: Ahmed Kamaleldin (e-mail: ahmed.kamal@tu-dresden.de)

This work was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID 287022738 – TRR196.

ABSTRACT Multi-/Many-core architectures are emerging as scalable, high-performance and energy-efficient computing platforms suitable for a variety of application domains from edge to cloud computing. Recently, the appearance of RISC-V open-source ISA creates new possibilities to develop customized computing platforms with high savings in the non-recurring engineering costs. Moreover, the current trends toward open-source hardware frameworks are aimed to reduce design time and cost for complex system-on-chip architectures. Therefore, modularity and re-usability of hardware components are major challenges for flexible hardware architectures. The motivation behind this work is to introduce a modular cluster-based many-core architecture for FPGA accelerators that is re-usable and flexible tailored to implement different many-core taxonomies with less design time and costs by using regular and replicated sets of computing, memory, and interconnection blocks. The proposed many-core architecture is built using multiple processing clusters coupled with a NoC for communication which allows a high degree of design scalability. The processing cluster inside features a configurable multi-core architecture consisting of multiple RISC-V processing elements (PE) tightly coupled with a bus-based interconnection for intra-cluster communication using parameterized scratchpad shared memory. Each PE features a single RISC-V core with a tightly coupled parameterized scratchpad local memory and generic AXI interface. Evaluation results demonstrate that the proposed architecture features a scalable computing performance of 501 MOp/s for 4 clusters and 878 MOp/s for 8 clusters. Moreover, a scalable memory bandwidth up to 4.3 GB/s is achieved for 9 clusters with a power consumption of 1.4 W per cluster utilizing 7.7% of on-chip memory resources. The many-core architecture is implemented and evaluated on Xilinx Virtex Ultrascale+ with the feature of changing the architecture configurations during run-time using dynamic and partial reconfiguration which provides more flexibility and re-usability.

INDEX TERMS Many-Core Architecture, Parallel Computing, RISC-V, Network-on-Chip (NoC), Field Programmable Gate Array (FPGA), Reconfigurable Computing.

I. INTRODUCTION

CURRENT and future applications in domains like deep neural network or next-generation cellular standards like 5G impose high demands on a novel approach for hardware platforms that can cope with high computational complexity and memory requirements with low energy consumption [1, 2]. Therefore, multi-/many-core architectures have emerged as adequate scalable hardware platforms to address the ever-increasing computation demands while maintaining a sort of energy-efficiency. Whereas, the ending of Dennard's scaling and the inability to achieve energy efficiency with high computing density by a single complex processor drives hardware architects to explore new approaches for novel architectures in order to increase the performance thereby maximize the energy-efficiency. Hence, current many-core architectures designs are following the path of integrating multiple processing nodes using the same silicon area required by a complex processing core. Whereas the processing nodes are either simple general-purpose processors or application-specific hardware accelerators. In the last decade, several multi-/many-core architectures have been developed as application-specific or application-oriented hardware platforms either with homogeneous [3] or heterogeneous [4, 5] processing cores. The goal here is to provide hardware solutions with high performance/watt for specific application domains (e.g. software-defined-radio). However, the integration of many processing cores requires an efficient communication infrastructure (e.g. network-on-chip, advanced bus-based architectures) and memory hierarchies that can cope with the high scalability and performance requirements besides the associated programmability challenges.

As a result, integrating more components and different architectural units on a complete system-on-chip increases design efforts (e.g. verification, validation, integration) and therefore the rising of development time and costs. Moreover, the design specifications could vary due to different application requirements which lead to the necessity of a new design process for each new application requirement [2]. Resultantly increasing the design effort and therefore time to market with a continuous inflation in non-recurring engineering costs. Recently, the agility and re-usability of a new scalable/configurable computing platform have attracted the attention of the computer architecture community [6] driven by the proliferation of the open-source instruction set architecture (ISA) by RISC-V and also the tendency towards a new ecosystem for open-source hardware frameworks similar to the software counterpart.

In this context, this paper introduces a modular cluster-based many-core architecture for FPGAs. The focus of this work is to propose a novel modular implementation for many-core architectures based on RISC-V open-source-hardware processors with cluster-granularity customization for FPGA platforms. The proposed design has re-usable and flexible architectural units that can be tailored to implement different heterogeneous and homogeneous many-core taxonomies using regular building blocks for computation (e.g. PEs, processing clusters), with several memory hierarchies and generic communication interconnections. Thus, our goal is also to analyze the effects of different architecture configurations regarding memory types, communication/network interconnections and the number of processing cores on the system performance.

The foremost contributions of this work are summarized as follows:

- Designing a modular RISC-V based PE with tightly coupled local data/instruction memory and AXI compatible interfaces to communicate with several memory mapped peripherals and custom HW accelerators.
- Implementing a configurable processing clusters that hosts configurable numbers of PEs with configurable size shared memory system connected through a shared bus architecture.
- Developing a scalable RISC-V based many-core architecture using configurable number of processing clusters connected through a generic NoC architecture applying a message-based communication model.
- Allowing the flexibility of run-time configuration for several many-core configurations through dynamic partial reconfiguration techniques.

As a result, the architecture maintains a high degree of scalability using a scalable NoC topology and the design regularity manner offers the flexibility to scale up the number of processing clusters with less design effort and cost. Furthermore, a message-based communication model is adopted to support data transfer over the NoC between processing clusters. In addition, a bare metal programming method is introduced on the level of processing clusters for parallel programming over the RISC-V PEs using shared and local data memories on the cluster level. Moreover, the proposed many-core architecture is evaluated based on different architecture configurations covering different types of memory hierarchies/sizes, communication interconnections, and number of cores/clusters per system to explore several design choices and their effects on the system performance. In this work, for real hardware implementation and evaluation, the proposed architecture is implemented and evaluated on a Xilinx Virtex Ultrascale+ FPGA. Furthermore, the architecture can be flexibly portable to other Xilinx FPGA series without the need to re-design the architecture's building block components which make it feasible for FPGA migration. Finally, the architecture supports run-time adaptation of many-core configurations regarding memory-type and number of processing cores per cluster using dynamic and partial reconfiguration without the need to re-synthesize the whole architecture. Accordingly and to the best of our knowledge, our proposed architecture is the first RISC-V based many-core architecture that supports run-time adaptation of several architectural configurations.

The rest of the article is structured as follows: Section II discusses background and related work. Section III gives a detailed overview of the proposed modular design approach for the proposed RISC-V based many-core architecture. The evaluation and experimental results are presented in Section IV. Finally, Section V summarizes this work and gives an outlook for future work.

II. RELATED WORK AND BACKGROUND

Several high-performance and energy-efficient multi-/many-core architectures have been developed in both academia and industry. However, there are a limited number of studies discussing system scalability and design regularity in the form of replicated building blocks or tile-based architectures for homogeneous and heterogeneous many-core implementations. Besides, common multi-/many-core architectures are mass-produced in the form of ASIC which make them highly customized platforms for specific applications with dedicated processing cores. Therefore, design time and manufacturing are significant obstacles with high non-recurring engineering costs which are not affordable for small-size companies or academic research. We provide here a review of related

TABLE 1. **Comparison of different state-of-the-art multi/many-core platforms with respect to architecture specifications.**

	RISC-V	Topology		Memory System		Support Custom	FPGA	ASIC	Run-time
	based PE	NoC	Cluster/Tile	Local	Shared^{SH}	HW Accelerators	based	based	configuration
HERO [7]	✓	-	✓	-	✓	-	✓	-	-
OpenPiton+Ariane ^{T} [8]	✓	✓	✓	✓	-	-	✓	-	-
ESP ^{T} [9, 10]	✓	✓	✓	✓	-	✓	✓	-	-
Savas et al. ^{T} [11, 12]	✓	✓	✓	✓	-	✓	✓	-	-
RVNoC [13]	✓	✓	-	✓	-	-	✓	-	-
Memphis [14]	-	✓	-	✓	-	✓	✓	-	-
Vestias et al. [15, 16]	-	-	-	-	✓	-	✓	-	-
GRVI Phalanx [17]	✓	✓	✓	✓	✓	-	✓	-	-
BlackParrot ^{T} [18]	✓	✓	✓	✓	-	✓	-	✓	-
CoreVA-MPSoC [19]	-	✓	✓	✓	✓	-	-	✓	-
P2012 [20]	-	✓	✓	✓	✓	✓	-	✓	-
Epiphany [21]	-	✓	-	✓	-	-	-	✓	-
Kalray MPPA256 [22]	-	✓	✓	✓	-	-	-	✓	-
This Work	✓	✓	✓	✓	✓	✓	✓	-	✓

^{T} Tile based architecture. ^{SH} Shared memory between multiple PEs/cores inside one cluster/tile.

work for homogeneous and heterogeneous multi-/many-core architectures and frameworks for FPGA and ASIC platforms.

Kurth et. al [7] presented a heterogeneous many-core research platform on FPGA (HERO). It explores the integration of RISC-V core into a shared memory cluster-based programmable many-core accelerator (PMCA). The PMCA is connected with an ARM Cortex A53 as a host CPU through an AXI coherent interconnect which provides coherent access to the shared external memory with the host caches memory. Thus, the system allows shared virtual memory which eases the system programmability. Moreover, the PMCA is configurable in terms of the number of clusters and cores supported during design time. Meanwhile, shared hardware accelerators can be coupled with the RISC-V cores within the cluster. However, the PMCA clusters are connected through a custom shared bus architecture which limits the overall scalability to 8 clusters. Therefore, to increase the degree of scalability, OpenPiton [8] platform is proposed as an open-source scalable architecture for many-core system prototyping. OpenPiton is a tile-based architecture that supports different NoC topologies for tiles interconnection. Each tile contains an Ariane 64-bit RISC-V core with private cache memory connected to three NoC routers. Further, multiple tiles are integrated into a chip and multiple chips are connected together with a NoC for a scalable architecture. Moreover, each tile is configurable at design time to be extended with a tightly coupled floating point or a stream processing unit. Moreover, OpenPiton supports cache coherency protocol enabling shared memory across multiple chips. In the same context, the embedded scalable platform (ESP) proposed by Carloni et. al [9, 10] is aimed to address the complexity of design regularity for heterogeneous many-core architecture with a special focus on HW/SW interaction between RISC-V cores and hardware accelerators. Therefore, the ESP tile-based architecture contains coherent socket interface and direct memory access (DMA) engine for communication and data sharing through a NoC. The accelerators are hosted by the tile connected to a communication socket as a loosly coupled model. Moreover, ESP socket interface supports the integration of high level synthesis (HLS) based accelerator.

Similarly, Savas et. al [11, 12] proposed a framework to design a domain-specific heterogeneous many-core architecture from application data flow graphs. The framework is based on a heterogeneous tile-based architecture consisting of a simple RISC-V core, memory and accelerator tiles connected through a NoC. Also, the framework allows the integration of HLS based accelerators to the architecture. RVNoC [13] framework is a design time configurable RISC-V NoC-based MPSoC to integrate many RISC-V cores using a reconfigurable NoC architecture to allow large system scalability in term of computing elements. However, it leaks the flexibility manner of cluster/tile based architectures to host multiple RISC-V PEs or hardware accelerators on a single processing unit with a shared or local memory system which gives a second level of design scalability inside the cluster/tile node. Furthermore, Memphis [14] framework is proposed for scalable heterogeneous many-core SoCs. It supports the generation and integration of homogeneous PEs with hardware accelerators as shared peripherals to all PEs. However, it does not support the tightly coupled integration of HW accelerators directly with PEs like the case of cluster/tile based architectures. Moreover, the generated architectures [13, 14] lacking fine configurations regarding memory types/sizes and processing cores as proposed by our modular architecture. In contrast, Vestias et al. [15, 16] proposed a configurable FPGA-based many-core overlay for applications acceleration. It works as a co-processor for high performance embedded systems with configurable local memory size, core counts, and supported arithmetic operations per processing core. As well, the GRVI Phalanx overlay [17] is proposed for extreme scalability for FPGA-based many-core accelerators. It efficiently uses the FPGA resources to place hundreds of RV32I base processing clusters with a scalable NoC architecture.

In addition to FPGA based platforms for rapid prototyping and evaluation, several many-core architectures that target ASIC platforms for low power and energy consumption requirements have been developed. In this context, BlackParrot [18] is proposed as modular low power RISC-V based multicore architecture. The architecture is specified as a heterogeneous tile-based architecture similar to the ESP platform proposed by [9, 10] which also maintain data coherency between the RISC-V and accelerators tiles. Hence, BlackParrot uses a cache-coherent NoC for communication between all tiles. Moreover, it supports the extension with second level caches between multiple tiles and external DRAM and user peripherals. Besides, Ax et al. [19] proposed the Core-VA as a NoC-based many-core architecture with a hierarchical communication and cluster-based structure. The architecture features global asynchronous and locally synchronous (GALS) NoC architecture. Moreover, each cluster has a tightly-coupled shared memory between the cores for low memory latency and to reduce energy consumption. However, integration of hardware accelerators is not supported. Similarly, P2012 [20] many-core GALS architecture is built as a modular cluster-based architecture with tightly coupled shared memory. It features a configurable number of processing cores and memory size. Moreover, besides the NoC interface, the cluster has a stream interface which allows communication to multiple hardware accelerators. In the same context, Epiphany [21] is a commercial energy-efficient many-core with a global memory address space over NoC. Therefore, each core is allowed to access other cores memory coherently. However, the architecture does not introduce a cluster level and implements synchronous NoC architecture. Kalray MPPA-256 [22] is another commercial many-core architecture that supports a cluster-based architecture. Each cluster owns a private address space with local caches memories. Moreover, the architecture does not support a global address space on the NoC level compared to [18-20].

Accordingly, our proposed architecture differentiates from these above-mentioned work as shown in Table 1 by providing more design configurability and flexibility related to memory types/sizes and number of cores per cluster in order to achieve high design scalability and regularity features for a modular platform that supports multiple computing and memory choices to be tailored for different applications. Hence, the proposed many-core is implemented as a cluster-based architecture. The clusters are connected through a synchronous NoC architecture ARTNoC [23] using stream network interfaces. Each cluster tightly couple multiple RISC-V PEs with shared instruction/data memories using a shared AXI interconnect. The cluster features a private address space which allows the communication between all PEs and shared peripherals. Moreover, each PE features a scratchpad local memory for low memory latency access. Besides, each PE is extended with a stream interface for communication with hardware accelerators. In addition, a message-based communication model is developed to manage the data transfer between the clusters over the NoC. Furthermore, the cluster is configurable based on the number of cores and memory sizes and could be reconfigured during run-time using dynamic partial reconfiguration.

III. MODULAR MANY-CORE ARCHITECTURE

The proposed many-core architecture features a modular and hierarchical interconnect design which targets domain-specific and general-purpose applications for FPGA accelerators. Moreover, the proposed many-core architecture can be considered as a model for rapid prototyping of different many-core taxonomies with homogeneous or heterogeneous computing elements (by adding optional application-specific hardware accelerator cores) and supports different styles of interconnect topologies. The proposed architecture is a cluster-based many-core architecture which consists of a scalable number of processing clusters connected by a network-on-chip interconnect as shown in Figure 1. Within each cluster, several processing elements with shared data and instruction memories are tightly coupled via a bus-based interconnect. In this section, the proposed many-core architecture and its programming method are described.

A. PROCESSING ELEMENT

The Processing Element (PE) is the main computing unit inside the proposed many-core architecture. The proposed design modularity of the PE allows the execution of general-purpose applications across different domains e.g. (signal or image processing) with different computing requirements and memory footprints. The PE consists of a single open-source RI5CY soft-core processor [24] and a local tightly coupled memory (TCM) subsystem for data and instructions as shown in Figure 1 (b). The RI5CY core is a 32-bit 4-stage pipeline in-order processor. The core implements a simple RV32IMC ISA with main arithmetic-logic-unit (ALU) and dedicated units for multiplication, division and multiply-accumulate (MAC). Moreover, the RI5CY core can be extended to support RV32IMFC with an optional single-precision floating-point unit to increase the computing capabilities. Moreover, the majority of the instructions have a latency of one clock cycles except for the load/store (LD/ST) and the dedicated arithmetic instructions which have a minimum latency of 2 clock cycles [25].

In addition, like typical Harvard architecture, the PE features separated local instruction and data memories tightly coupled with the RI5CY core. The local TCMs feature a

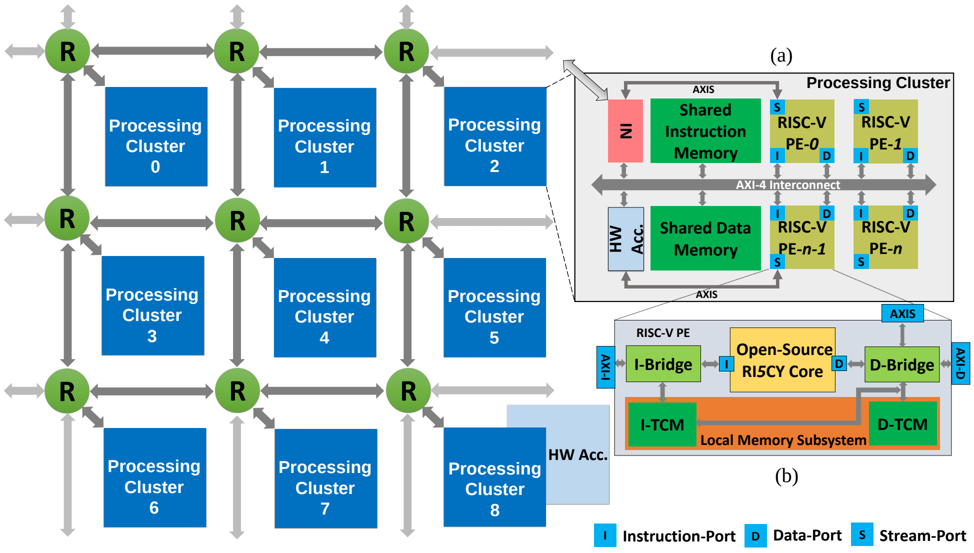
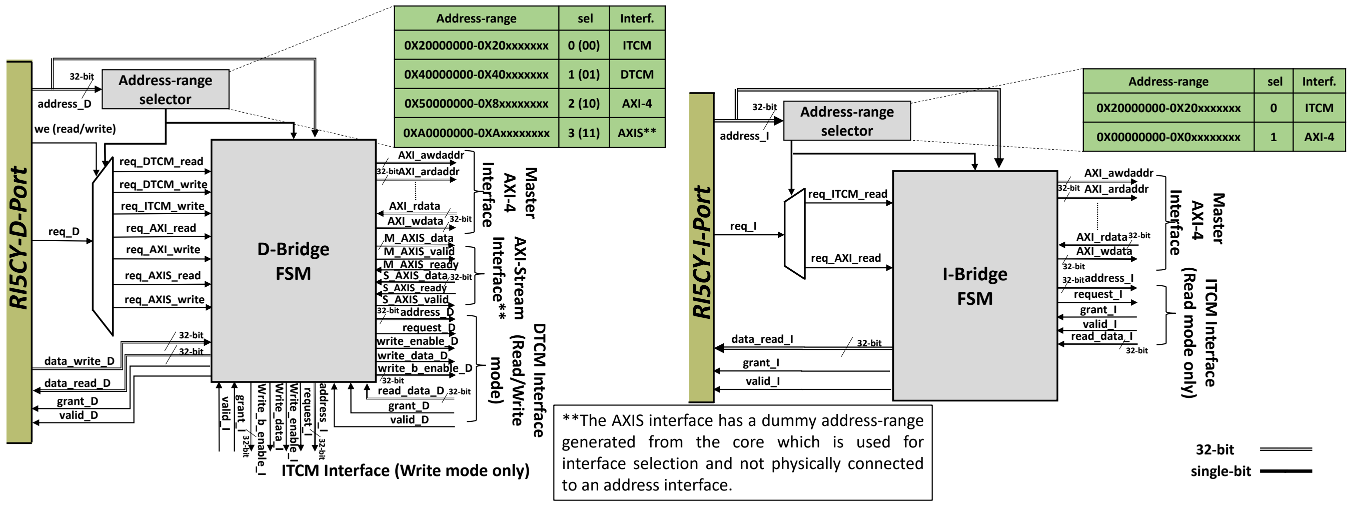


FIGURE 1. Overview of the proposed RISC-V based many-core architecture with a 3x3 mesh NoC including: (a) Processing cluster block diagram with shared memories, network interface and optional hardware accelerator, (b) RISC-V PE block diagram consisting of a RV32IMC core (RI5CY) with TCMs.



Address-range	sel	Interf.
0X20000000-0X20xxxxxxx	0 (00)	ITCM
0X40000000-0X40xxxxxxx	1 (01)	DTCM
0X50000000-0X8xxxxxxxx	2 (10)	AXI-4
0XA0000000-0XAxxxxxxxx	3 (11)	AXIS**

Address-range	sel	Interf.
0X20000000-0X20xxxxxxx	0	ITCM
0X00000000-0X0xxxxxxxx	1	AXI-4

**The AXIS interface has a dummy address-range generated from the core which is used for interface selection and not physically connected to an address interface.

(a) D-Bridge block diagram (b) I-Bridge block diagram

FIGURE 2. Schematic of the data and instruction bridges inside the RISC-V PE to connect the RISC-V core to the D/I TCM and the memory mapped AXI peripherals.

low memory latency of one clock cycle for read/write operations for private computation within a single PE. Moreover, using a local memory per each PE reduces the probability of memory interference between multiple PEs compared to the uniform memory access (UMA) in shared memory hierarchies. To emphasise the design modularity, the memory sizes of D/I-TCM are configurable during the design time based on the target applications memory requirement. Also, all the memory blocks have a fixed word width of 32-bit compatible with the RV32 ISA. As shown in Figure 1 (b), the ITCM in the local memory subsystem is implemented as a dual-ported BRAM with a read-only interface to the RI5CY core instruction port (I-Port) for instruction fetching to supply one instruction to the decode stage every clock cycle. In addition, a write-only interface to the data port (D-Port) allows the transfer of specific instructions from the shared instruction memory to the ITCM during the memory initialization stage. In contrast, the DTCM is implemented as a single port BRAM with read/write interfaces to the RI5CY core D-Port. The DTCM is only accessed directly via its coupled PE. Therefore, accessing the DTCM directly by other PEs is prevented and the local data memory has to be transferred to the shared data memory to allow data sharing between several PEs in the same processing cluster. In addition to the local TCMs, the RI5CY core I/D-Ports interfaces are extended by implementing data and instruction bridges (D, I-Bridges) to provide compatible interfaces to the AXI-4 and AXI-Stream standard interfaces which allows a direct connection to RI5CY core to communicate with the AXI memory-mapped/stream components inside the processing cluster. Since the RI5CY core or the PE is the master unit on the proposed system. The supported AXI-4 interface is a master interface which permits a connection to any AXI-4 slave peripherals inside the cluster. Figure 2 gives an abstract schematic of the D, I-Bridges internal implementation. The D-Bridge handles the RI5CY read/write memory requests (*req_D*) and the write-enable (*we*) signals from the D-Port interface by rerouting them based on the memory-mapped address range to the corresponding memory-mapped component as shown in Figure 2 (a). Hence, a finite state machine is implemented with 7 states covering the read/write states to the (AXI-4, AXIS, ITCM_write and DTCM) memory-mapped interfaces. According to the state and the address-range input, the D-Port interfaces (*data_write/read_D, valid_D, grant_D*) are reconnected to the corresponding memory-mapped interfaces and a connection is established between the core and the corresponding memory-mapped peripheral. Moreover, inside the FSM a custom AXI-4/AXIS protocol converter is implemented to convert the D-Port interfaces to a compatible AXI-4/AXIS-interfaces. Similar to the D-Bridge, the I-Bridge is implemented as shown in Figure 2 (b) with a 2 states FSM for only reading from the ITCM or the shared instruction memory attached to the AXI-4 interconnect.

B. PROCESSING CLUSTER

The processing cluster tightly couples multiple PEs with shared instruction and data memories using a shared bus interconnect. Therefore, the PEs share a common address space inside the processing cluster which allows the communication between them and accessing shared memories and shared memory-mapped peripherals via the bus interconnect as shown in Figure 1 (a). In this work for purposes of modularity and compatibility, the AXI-4 interconnect standard with a 32-bit width is used as the shared bus interconnect. The AXI-4 interconnect uses separate channels for address and data. In addition, separate read/write channels can be established simultaneously which allows parallel data transaction across the shared bus. Furthermore, the AXI-4 interconnect applies a round-robin arbitration scheduling scheme for multiple requests to the same shared peripheral. This fact increases the data transfer bandwidth across the bus and reduces the probability of bus congestion.

The processing cluster implements a UMA architecture, where each PE can access shared data and instruction scratchpad memories connected to the bus as a slave memory-mapped peripheral. Therefore, the shared data memory is used for communication and synchronization between the PEs inside a single cluster. While the shared instruction memory is implemented as read-only memory which is used as a boot memory during the memory initialization stage [26]. Also, it is considered the main execution memory inside the cluster to store the common instructions running on all PEs. In the UMA architecture, each PE experiences the same bandwidth and latency to the memory. However, the overall memory bandwidth is divided between the number of PEs, since all of the memory read/write request and data transaction are conducted across the AXI interconnect. The growing number of PEs connected to the bus leads to a decrease in the total memory bandwidth for a single cluster. In order to enhance the memory bandwidth, the shared data and instruction memories are implemented as dual-ported BRAM blocks. Therefore, two memory read/write channels can be established across the shared bus to handle two memory requests simultaneously. However, the memory bandwidth scalability is limited and starts to saturate after a certain number of PEs. The shared data and instruction memories are size configurable at design time. Moreover, the cluster supports the integration of loosely coupled hardware accelerator as a memory-mapped peripheral to the PE connected to the shared bus or tightly coupled to a specific PE via the AXI-stream interfaces as shown in Figure 1 (a).

C. NETWORK-ON-CHIP

A Network-on-Chip (NoC) is used on large scale Multi-Processor System-on-Chip (MPSoC) or many-core architectures to connect dozens to hundreds of PEs or processing clusters together, providing on-chip end-to-end communication paradigm and increasing the system scalability. In this work, the ARTNoC [23] real-time NoC architecture is used for inter-cluster communication in the proposed many-core architecture. The NoC provides guaranteed quality of service (QoS) in terms of bandwidth and end-to-end latency. In addition, the router architecture is highly modular and parametrizable. It supports different I/O ports configurations, switching controls, buffering sizes and routing schemes. The ARTNoC circuit-switched-based version is used in the implementation as it features a low area overhead compared to a packet-switched based NoC. The NoC is based on a 2-D mesh topology with an XY-routing algorithm with configurable size and I/O data widths at design time. Furthermore, the NoC internal architecture consists of ① a 5 ports circuit-switched router including a control path circuitry and arbiters for path reservation, ② a crossbar to switch between the I/O ports and using round-robin arbitration scheme, ③ synchronous network links for communication between the routers as shown in Figure 1 (a). The circuit-switched NoC reserves a static transmission path between the source and destination. This is performed by sending a single-flit setup packet from the source containing the X-Y coordinate of the destination node. Moreover, the NoC can transmit a single packet flit every one clock cycle with a 32-bit payload data.

In addition, a network interface (NI) is implemented to allow communication between the processing cluster and the NoC. The used ARTNoC I/O interfaces are compatible with the AXI-stream interface. Therefore, the proposed NI architecture is based on a flit-based streaming approach. Hence, the NI links between the address-based shared bus used by the cluster and the AXI-stream interface of the NoC. Moreover, the NI allows the transmitting or receiving of data directly from/to the PEs via the PE stream port without passing by the shared bus interconnect which provides a

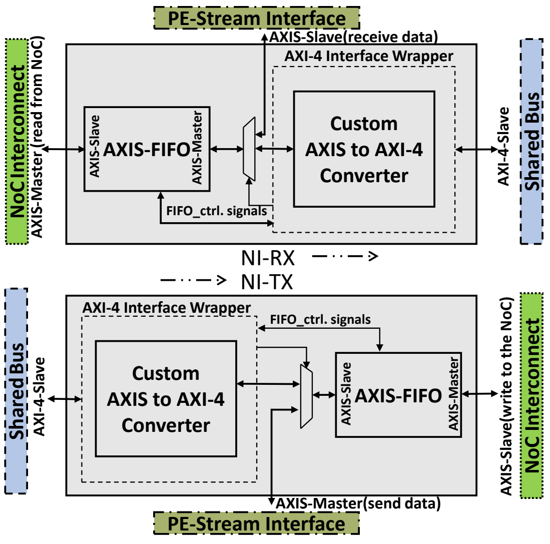


FIGURE 3. Network Interface (NI) block diagram.

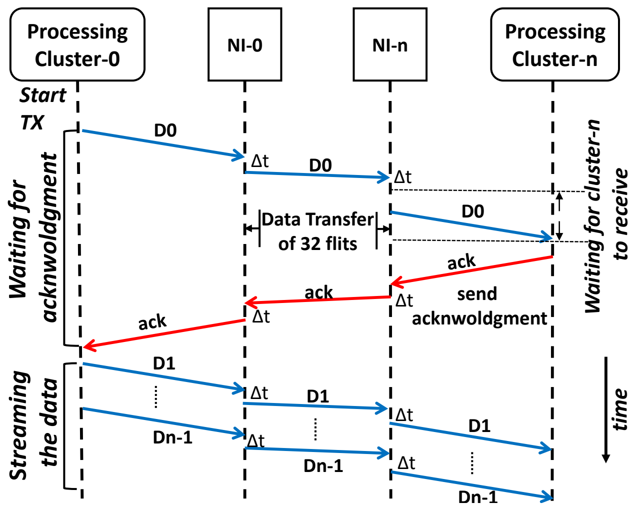
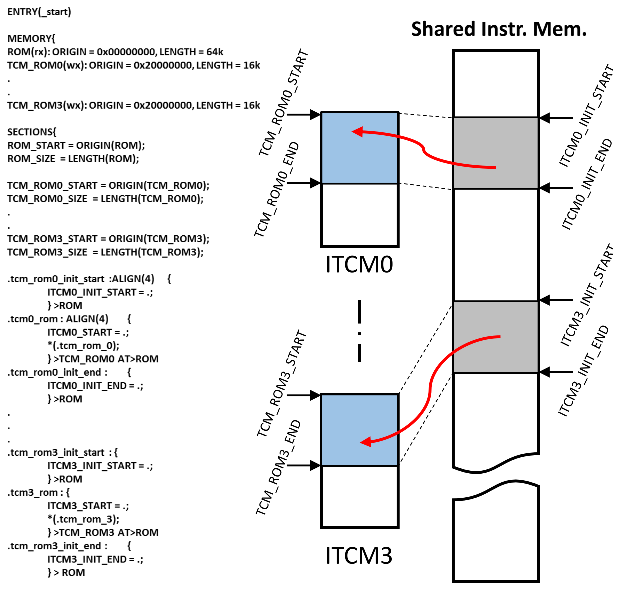


FIGURE 4. Sequence diagram of the Synchronous message-based communication model for NoC data transfer between the clusters.



```
ENTRY(_start)

MEMORY{
ROM(rx): ORIGIN = 0x00000000, LENGTH = 64k
TCM_ROM0(wx): ORIGIN = 0x20000000, LENGTH = 16k
.
.
TCM_ROM3(wx): ORIGIN = 0x20000000, LENGTH = 16k

SECTIONS{
ROM_START = ORIGIN(ROM);
ROM_SIZE  = LENGTH(ROM);

TCM_ROM0_START = ORIGIN(TCM_ROM0);
TCM_ROM0_SIZE  = LENGTH(TCM_ROM0);
.
.
TCM_ROM3_START = ORIGIN(TCM_ROM3);
TCM_ROM3_SIZE  = LENGTH(TCM_ROM3);

.tcm_rom0_init_start  :ALIGN(4)    {
            ITCM0_INIT_START = .;
            } >ROM
.tcm0_rom : ALIGN(4)      {
            ITCM0_START = .;
            *(.tcm_rom_0);
            } >TCM_ROM0 AT>ROM
.tcm_rom0_init_end :      {
            ITCM0_INIT_END = .;
            } >ROM
.
.
.
.tcm_rom3_init_start  : {
            ITCM3_INIT_START = .;
            } >ROM
.tcm3_rom : {
            ITCM3_START = .;
            *(.tcm_rom_3);
            } >TCM_ROM3 AT>ROM
.tcm_rom3_init_end :      {
            ITCM3_INIT_END = .;
            } > ROM
```

FIGURE 5. Instruction memory mapping and its relevant linker script (*.ld*) for a single processing cluster.

tightly coupled connection between the PE and the NoC. An overview of the NI internal architecture is shown in Figure 3. The NI has two separated channels for sending and receiving data. It is connected to the shared bus interconnect as AXI-slave memory-mapped peripheral that can be accessed by all PEs. Moreover, the NI supports the streaming of data directly to a single PE via the stream port in order to reduce the data interference between multiple PEs for hard real-time applications. The NI internal architecture consists of ① an AXI-stream FIFO of size 64 locations to store the transmitted or received data to/from the NoC, ② a custom AXI-stream to AXI-4 converter to connect the FIFO and its control signals to the cluster AXI-interconnect. A certain PE can access the NI by setting a synchronization flag (for either sending or receiving) in the shared memory indicating that the NI is used by this PE to prevent NoC deadlocks and data interference between several NI requests from different PEs. The data flow between a PE to a NI for NoC sending is performed by setting a pointer to the shared data memory or PE DTCM to transfer a specified size of data. The data is transmitted by a form of a group of 32 packet-flits to the NI-TX FIFO either by the PE stream port or the AXI-interconnect based on the NI control signals. Similarly, in the receiving direction, the received packet-flits are stored in the NI-RX FIFO until a reading request comes from a certain PE to transfer the packets to the shared data memory or DTCM. Due to the separated read/write channels of the AXI-interconnect, transmitting and receiving of data can be done concurrently.

D. COMMUNICATION MODEL

Typical many-core and MPSoC architectures are considered as a suitable platform to run multi-tasks applications. Each task is mapped to one or more PEs or processing clusters based on the computation requirements. The tasks are connected via a directed data flow graph that defines the data flow and the execution period of each task for a specific application. In this work, a communication model between the clusters with unidirectional RX/TX channels is developed based on the NoC and NI architectures described in the previous subsection. The communication model applies a message-based communication approach initiated by the transmitting cluster and ending by the receiving cluster. Figure 4 shows a sequence diagram of synchronous message-based communication between two processing clusters. The proposed communication model provides a synchronization mechanism between the sender and receiver clusters to avoid NoC deadlocks and prevents packet losses during the transmission. As shown in Figure 4, the transmission is initiated by any PE in the sender cluster. The first transmitted packet contains the X-Y coordinate of the destination followed by 32 packet-flits containing the first 32 payload packets. The sending PE is blocked until it receives an acknowledgement (*ack*) packet from the receiving cluster to indicate a successful establishment of a communication channel. Afterwards, the sending cluster starts to stream the following data packet-flits and the receiving PE in the receiving cluster is blocked until successful receiving of the complete size of data. However, the software latency cost is higher than the physical data streaming latency of the NoC. Due to the reading and writing processes from/to the shared or TCM data memories on both processing clusters. Listing 1 shows the *C* application programming interfaces (APIs) for the proposed communication model depicted in Figure 4 for sending (*send_data*) and receiving (*rec_data*) over the NoC architecture using the NI. The communication APIs are executed from the shared instruction memory of the cluster to be accessable to all of the PEs in a cluster.

E. PROGRAMMING METHOD AND SOFTWARE EXECUTION

Programming many-core architectures or MPSoCs is a challenging task for the programmer to effectively uses their computation and communication resources. For this reason, a bare-metal programming method is developed for the proposed many-core architecture to generate multiple binary files from multi-tasks application source codes e.g. (*c* codes) corresponding to the number of used processing clusters. Whereas, each processing cluster executes a single binary file for its mapped task from the application data flow graph. Task mapping and partitioning processes are done statically by the programmer at design time. Therefore, the proposed programming method does not support mapping or partitioning methods during runtime.

The *PULP-RISC-V GNU* toolchain [27] is used to compile the *C* source codes for the RV32IMC architecture. Afterwards, the generated (*.elf*) file is converted to a Verilog memory file using the *objcopy-tool* of the toolchain. The Verilog memory file contains the generated binary file or the complete instruction set for a single cluster. Finally, a BRAM coefficient file (*.coe*) is generated from the Verilog memory file to be loaded on the shared instruction memory during design time. In-order to programme each PE inside the processing cluster a memory initialization stage is required to load the ITCM of each PE by the corresponding instruction sets of a specific task running on this PE. Therefore, a linker script (*.ld*) is developed as shown in Figure 5 for instruction memory mapping. The linker script defines the instruction memory partitions based on the memory address space for the complete processing cluster. Hence, during the memory initialization, each PE starts to load its instruction set from the shared instruction memory based on the address mapping to its ITCM as shown in Listing 2. In the application C code each function which has to be executed from a local ITCM has to be preceded with a memory section attribute e.g. (*__attribute__*((*section*(".*tcm_rom*0")))) which defines its executable ITCM as shown in Listing 2.

IV. IMPLEMENTATION RESULTS AND PERFORMANCE EVALUATION

Physical hardware implementation, system scalability/reconfiguration and performance analysis results of different design configurations are discussed and presented in this section. The Xilinx Virtex Ultrascale+ XCVU9P FPGA is used for implementation and prototyping of the proposed RISC-V based many-core architecture.

Besides, Xilinx Vivado Design Suite HLx 2017.4 is used for RTL synthesis, simulation, verification, FPGA place and routing as well as bitstream generation. In this section, the many-core architecture is evaluated based on :

1) The hardware resources utilization of the different building blocks described in Section III.
2) The system scalability in terms of the number of PEs (RV32 cores) inside the processing cluster and its

```c
uint32_t*const NI_TX = (uint32_t)*0xA0000000;
uint32_t*const NI_RX = (uint32_t)*0xA000F000;
void send_data(uint_32t A[data_size], uint_32t data_size,
      uint_32t x_y_dest){
uint32_t t = 0; uint32_t ack = 0;
    while(t < data_size/32){
        NI_TX->data = x_y_dest;//start TX
        for(int m = 0; m < 32; m++)
            NI_TX->data = A[m+(32*t)];
        if(t == 0){//waiting for ack from RX node
            while(ack != 0x00000001){
              while(NI_RX->FIFO_data_count < 32);
                    for(int n = 0; n < 32; n++)
                        ack = NI_RX->data;
            }}}
return;}
void rec_data(uint_32t B[data_size], uint_32t data_size,
     uint_32t x_y_source){
    for(int t = 0; t < data_size/32; t++){
         while(NI_RX->FIFO_data_count < 32);
         for(int j = 0; j < 32; j++)
            B[j+(32*t)] = NI_RX->data;
        if(t == 0){//send ack to the TX node
            NI_TX->data = x_y_source;
            for(int j = 0; j < 32; j++)
             NI_TX->data = 0x00000001;//ack to the TX
                  node
         }}
return;}
```

Listing 1. ***C* API functions for sending/receiving data to/from the NI.**

```c
uint32_t*const tcm0_instr=(uint32_t*)&TCM_ROM0_START;
uint32_t*const tcm_rom0_init=(uint32_t*)&ITCM0_INIT_START
      ;
void __main(){//exec. from the shared instr.mem on all
      PEs
    if(read_csr(0xF14U) == 0){//copy instr. from shared
        //instr. mem. to ITCM0 of PE-0 (thread-ID = 0)
        for(int i = 0; i <= &ITCM0_INIT_END - &
             ITCM0_INIT_START; ++i)
            tcm0_instr[i] = tcm_rom0_init[i];
    }
    if(read_csr(0xF14U) == 0) main_0();// execute on PE-0
}
void main_0(void){load_0(src[size],dest[size],size);}
__attribute__((section(".tcm0_rom")))//map to to ITCM0
uint32_t load_0(uint32_t* src, uint32_t* dest, int size)
      {}
```

Listing 2. **Memory initialization *C* code for a single processing cluster.**

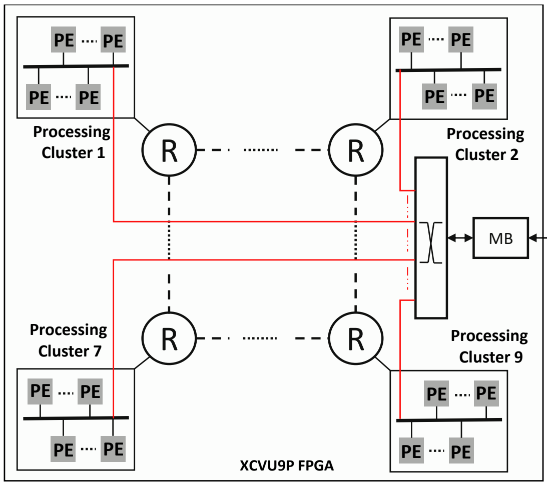


FIGURE 6. Complete system and testing setup for the 36 RV32 PEs many-core implementation, the MicroBlaze is used to dump the clusters data memories.

impact on the overall memory bandwidth of a single cluster.

3) The performance of NoC communication is measured in term of data transfer latency and the maximum achievable data rate between the clusters.

All benchmarks and test cases used for evaluation are written in software (C codes) and compiled using the *PULP-RISC-V GNU* toolchain [27] as described in the previous subsection (programming method and software execution) to generate the corresponding (.elf) files and coefficient files (.coe) to be loaded into the shared instruction memory of each cluster during the synthesis phase. The numbers of execution cycles in this section are software measured by using the performance counter register of the RI5CY core (PCCR) [25]. The number of cycles measured by the PCCR can be read using *read_csr* assembly function call in the application software and stored back in the cluster shared data memory to be retrieved back during the simulation (on Vivado simulator) and testing on FPGA. In addition, dynamic and partial reconfiguration (DPR) is applied to change the processing cluster configuration in terms of number of cores and memory sizes during runtime without re-synthesizing the complete architecture.

A. HARDWARE IMPLEMENTATION AND PROTOTYPING

The proposed many-core architecture has been implemented in a modular and hierarchical design process by creating each module as an intellectual-property (IP) block and integrating them inside the cluster module. The PE module contains the RI5CY core integrated with the I, D-Bridges with parameterized size D/I-TCMs. Also, NIs for transmitting and receiving are implemented as separate modules containing AXI-stream FIFOs of 32-bit data width and depth of 64 locations plus the required protocol converters. Afterwards, the processing cluster module integrates multiple PE modules connected to the AXI-interconnect via the AXI-I, D interfaces.

TABLE 2. Hardware Resources Utilization and Power Consumption for 36 PEs (9 clusters) Many-Core Platform on Xilinx XCVU9P.

	Resources Utilization				
Units	LUT	LUT RAM	FF	BRAM	DSP
Complete Setup	395308	257	145288	1537	219
MicroBlaze & Peripherals	1580	203	696	32	0
NoC (3×3)	64127 **5.42%**	0 **0%**	4752 **0.2%**	0 **0%**	0 **0%**
Single Processing Cluster†	36579 **3.1%**	6 **∼0%**	15532 **1.3%**	167 **7.73%**	24 **0.35%**
Shared Data Mem.[1]	127	2	14	64	0
Shared Inst. Mem.[2]	51	2	13	15	0
AXI interconnect	597	0	68	0	0
NIs	845	0	1937	2	0
RISC-V PE	7878 **0.66%**	0 **0%**	1944 **0.08%**	20 **0.92%**	6 **0.09%**
RISC-V Core	7626	0	1908	0	6
ITCM[3]	5	0	0	4	0
DTCM[4]	17	0	0	16	0
I, D Bridges	231	0	2	0	0
Total (%)* (complete setup)	**33.4%**	**∼0%**	**12.28%**	**71.15%**	**3.2%**
Max. Freq.	**120 MHz**				
Total Power Estimated (complete setup)	**13.869 W**				
Power Estimated (one cluster†)	**1.443 W**				
Power Estimated (NoC (3×3))	**1.91 W**				

†4 PEs, Mem. size:[1]256,[2]64,[3]16,[4]64KB. *% of total chip resources.

Besides, AXI-BRAM controllers with parameterized size shared instruction/data dual-ported BRAMs and NI modules are connected to the AXI-interconnect. In addition, the NoC (ARTNoC) is implemented as a single parameterized module including the circuit-switched routers based on the mesh topology size and the network links. The NoC parameters are mesh size, flit size and the maximum number of packet-flits for single-stream transmission over the NoC. Figure 6 shows the complete implementation and testing setup for 9 processing clusters with 36 RI5CY cores synthesized and placed on the Xilinx XCVU9P FPGA. A single MicroBlaze soft-core processor is connected to each cluster via the AXI-interconnect for system monitoring and to dump the shared data memory of each cluster. Hence, the shared data memory in each cluster is considered as a memory-mapped peripheral to the MicroBlaze during the monitoring stage to extract the memories contents for results and operations checking.

Table 2 shows the hardware resources utilization for the complete system depicted in Figure 5 of the many-core architecture with 3x3 NoC size. The processing cluster is configured with 4 PEs, 256 KB shared data memory and 64KB shared instruction memory. Each PE is configured with 16KB ITCM and 64 KB DTCM. As shown in Table 2 (third

TABLE 3. Comparison between different processing cluster/tile with respect to architecture specifications and hardware resources utilization.

Platform	RISC-V ISA	# PE	Memory Size [KB]	Resources per (Cluster/Tile) LUT	FF	BRAM	DSP	FPGA Device [Xilinx]	Freq. [MHz]	Power [W]
HERO [7]	RV32IMC	8	4160	128K	43K	384	48	Zynq-7000 (XC7Z045)	57	NA*
OpenPiton+Ariane ^{T} [8]	RV64GC	1	384	90K	81K	88	19	Virtex-Ultra.+ (XCVU9P)	100	NA*
Savas et al. ^{T} [12]	RV64G	1	1024	28241	9042	257.5	15	Virtex-Ultra. (XCVU095)	113	NA*
GRVIPhalanx [17]	RV32I	8	36	4800	NA*	12	NA*	Virtex-Ultra.+ (XCVU9P)	150	0.34
ESP ^{T} [10, 28]	RV64GC	1	1024	50290	38481	36‡	27	Virtex-Ultra.+ (XCVU9P)	250	1.484†
This Work	RV32IMC	4	1344	36579	15532	164	24	Virtex-Ultra.+ (XCVU9P)	120	1.443†

^{T} Tile based architecture without HW accelerators, * (Not available), † estimated power, ‡ reduction in BRAM blocks is due to using LUTRAMs instead of BRAMs during synthesis (5598 LUTRAMs is used from total 50290 LUTs).

row), a processing cluster consumes ∼3% of the total amount of LUT on the FPGA. While the complete NoC consumes ∼6% of the FPGA LUTs. Therefore, combining several cores in a cluster is more resource efficient than connect single cores directly to the NoC which leads to a high resources utilization and high power consumption by a large size NoC to achieve large scale many-core architectures. Furthermore, the power consumption of the complete system is estimated by Xilinx Power Estimator (XPE) at a clock frequency of 120 MHz as shown in Table 2 (last row).

Table 3 shows a comparison between our proposed cluster and several state-of-the-art cluster- /tile-based architectures specifications. The comparison is based on single cluster/tile specifications and hardware resources utilization without adding hardware accelerators. Our proposed cluster architecture supports a more complex RV32 core with M (multiplication and division) extension with more memory resources in comparison with the cluster architecture of GRVIPhalanx [17]. Furthermore, in comparison with HERO [7] with similar RV32 extensions for PE, the hardware resources utilization of (LUTs and FFs) are less than ∼30% of HERO's cluster resources utilization while using 4 PEs (half of PEs number used by [7]). In contrast, the tile-based architectures of [8, 10, 12] support a single core RV64G/C with floating point execution unit and multiple levels of caches subsystems which increase the resources utilization compared to the proposed cluster-based architecture with scratchpad memories and less complex RISC-V cores.

B. DESIGN SCALABILITY AND COMPUTING PERFORMANCE

Design scalability determines the capability and the flexibility of a parallel computing architecture to meet the required computing resources, memory bandwidth and communication data rate for parallel algorithms with growing complexity. Moreover, scalability is used to predict the performance of many-core architectures from the measured performance of single cores. In this work, the proposed many-core architecture is evaluated based on the maximum achievable memory bandwidth with respect to the number of PEs per cluster. Also, the computing performance is evaluated by the achievable number of operations per second (Op/s) and the maximum data transfer latency between the clusters over

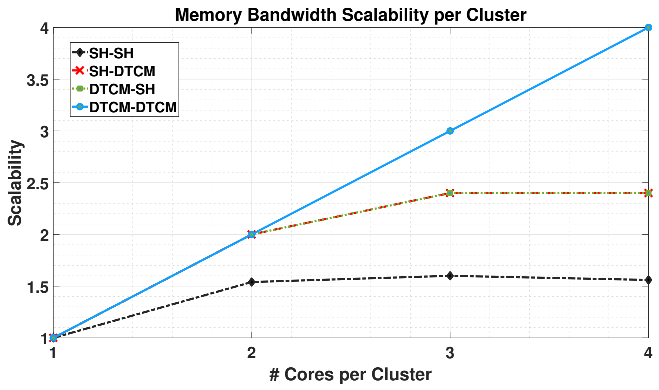


FIGURE 7. Memory bandwidth scalability for a single processing cluster with respect to the number of RV32 cores per cluster.

the NoC. The memory bandwidth is measured by a parallel executing of a *copy* function on all PEs to copy data of size of 4 KB through 3 evaluation scenarios ① shared data memory to shared data memory (SH-SH), ② from shared data memory to the DTCM (SH-DTCM) or vice versa (DTCM-SH) and ③ from DTCM to DTCM (DTCM-DTCM). Figure 7 shows the memory bandwidth scalability for one processing cluster. As a result, the data transfer bandwidth in case of shared to shared data memory is scaled by 1.5x using two PEs compared to one PE. However, the dual-port data memory is used, memory bandwidth is not scaled by the same factor due to the waiting cycles consumed for address collision mitigation if two PEs write or read from the same address at the same time. In contrast, splitting the memory write destinations by using DTCM in (SH-DTCM) scenario exploits the dual-ported memory feature by increasing the scalability to 2x in case of using 2 PEs.

Moreover, in case of using shared data memory for reading or writing, increasing the number of PEs over 2 will not increase the memory bandwidth scalability in proportional to the number of PEs due to the traffic contention through the AXI-interconnect. On the other hand, as shown in Figure 7, memory bandwidth is proportionally scalable with the increasing number of PEs in case of using DTCMs for writing and reading in a non-uniform memory access (NUMA) mode. The total memory bandwidth for the complete many-

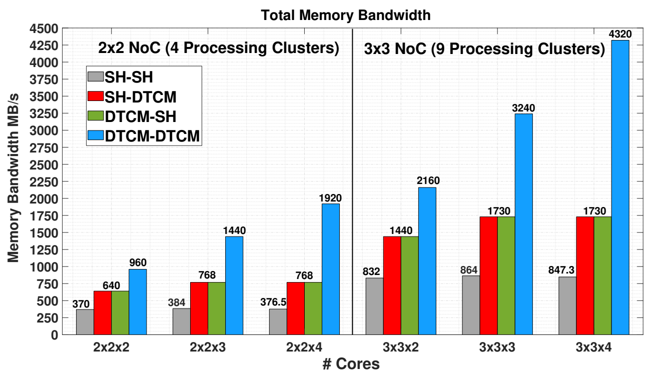


FIGURE 8. **Total memory bandwidth of the many-core platform with respect to the total number of RV32 cores and the used data memories inside the processing cluster.**

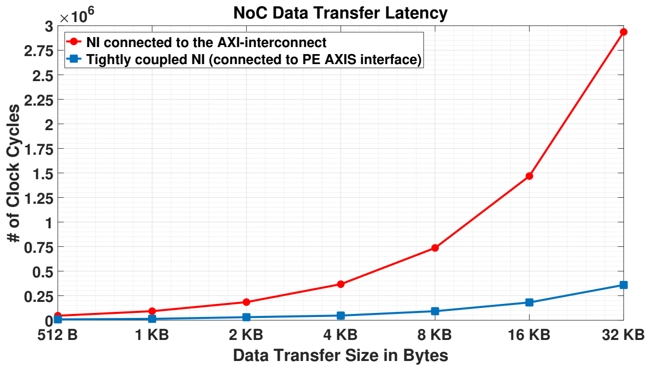


FIGURE 9. **Total latency in number of cycles for the NoC data transfer in cases of connecting the NI to the AXI-interconnect or directly to the PE AXIS interface.**

core architecture is calculated as follows:

$$Memory\ BW = \frac{freq. \times n_{PE} \times 2 \times data\_size}{n_{cycles}} \times n_{cluster} \quad (1)$$

Where, n_{PE} is the number of PEs per cluster, $data\_size$ is the data transfer size from memory source to destination per bytes within one cluster, n_{cycles} is the measured number of clock cycles, and $n_{cluster}$ is the total number of processing cluster in the many-core architecture. Figure 8 shows the total memory bandwidth for different many-core sizes and memory types configurations. A maximum memory bandwidth of 4.3 GB/s is achieved by a 3x3x4 many-core configuration using only DTCMs which is $\sim$5x the maximum memory bandwidth achieved with the same many-core configuration using only shared data memories.

In order to measure the data transfer latency for a single NoC transfer. A variant set of data sizes are transferred between two processing clusters by using two NI configuration modes. In the first configuration the NI is connected as a slave peripheral to the bus and the data and control signals are sent/received through the AXI-interconnect to the PEs. While in the second NI configuration the control signals only are transferred via the AXI-interconnect and the data signals are directly connected to the PE via the AXIS interface. Figure 9 shows the measured data latency for both NI configurations. Therefore, connecting the NI data port (RX/TX FIFOs) directly to the PE decreases the data transfer latency by $\sim$10x compared to connect it to the AXI-interconnect. However, connecting the NI to a single PE prevents the other PEs in a cluster to share it and to connect to the NoC.

A fixed point parallel square matrix multiplication benchmark is implemented in order to evaluate the computing performance of the proposed many-core architecture. The parallel block matrix algorithm is used to partition the A matrix into sub-matrices equals to the number of processing clusters. While the B matrix is partitioned into sub-matrices equals to the number of PE per cluster e.g. ($A\_size = 32 \times 32$, $A_{sub\_size} = 32/n_{cluster} \times 32$; $B\_size = 32 \times 32$, $B_{sub\_size} = 32 \times 32/n_{PE}$). Each processing cluster is responsible for an A sub-matrix and each PE inside a cluster compute the multiplication of a sub-matrix A with a sub-matrix B. For evaluation, two many-core configurations with 4 and 8 clusters including 16/32 RV32 cores are used for computation. In addition, a ninth cluster is used for block matrix generation, data transfer and results collection from each cluster. Table 4 shows the measured computing latency of the parallel matrix multiplication with different fixed point sizes using only the shared data memory or the DTCMs of each processing cluster for LD/ST the sub-matrices elements. Furthermore, the computing performance at a clock frequency = 120 MHz is calculated as follows:

$$Performance\ (Op/s) = \frac{2 \times n^3}{n_{cycles}} \times freq. \quad (2)$$

Where n^3 is the computing complexity of square matrix multiplication of size (n×n), n_{cycles} is the computing latency per clock cycles and the multiplication by 2 is the number of multiply and accumulate (MAC) operations. As shown in Table 4 (DTCM column), The computing performance is increased by $\sim$1.75x in case of using DTCMs for LD/ST operations in comparison of only using the shared data memories. Besides, the computing scalability is doubled in case of using 32 cores compared to 16 cores for local and shared data memory configuration. Also, from Table 4 (performance results), it can be observed that increasing the matrix sizes increases the computing performance. Whereas, the total number of LD/ST operations is proportional to the square size of the matrix while the computation is $\mathcal{O}(n^3)$. Therefore, a decreasing in the percentage of LD/ST cycles from/to the memory to the total computing cycles n_{cycles} has occurred which increases the computing performance based on equation (2). Similarly, reducing the fixed-point data size from 32-bit to 16- or 8-bit decreases LD/ST cycles from/to the data memory. Therefore, the total computing latency is reduced and the computing performance is increased. As a result, a maximum performance of 878.4 FP-MOp/s can be achieved in case of 8-bit (64×64) matrix size over 32 cores with DTCM as shown in Table 4. Moreover, Figure 10 shows the overall speedup (acceleration) achieved when parallelizing different matrix multiplication sizes over 16 cores (4 clusters)

TABLE 4. Computing performance and data transfer latency for matrix multiplication benchmark using 16/32 cores at clock frequency = 120 MHz.

Data Memory →		Shared Data Mem.				DTCM				Data Transfer Latency[2] [Cycle]	
Fixed Point Size	Matrix Size	Computing Latency[1] [Cycle]		Performance[1] [FP-MOp/s]		Computing Latency[1] [Cycle]		Performance[1] [FP-MOp/s]			
		16 Cores	*32 Cores*	*16 Cores*	*32 Cores*	*16 Cores*	*32 Cores*	*16 Cores*	*32 Cores*	*16 Cores*	*32 Cores*
8-bit	*16×16*	3695	1691	265.2	600	2119	1279	463.2	765.6	17040	28440
	32×32	23439	14216	336	553.2	15995	9216	480	852	68160	113760
	64×64	180960	91304	**348**	**688.8**	125155	71624	**501.6**	**878.4**	273096	459078
16-bit	*16×16*	5603	3024	175.2	320.4	2375	1559	392.4	630	33720	56280
	32×32	42333	21834	180	360	18043	10579	411.6	744	135276	225360
	64×64	333578	169825	**188.4**	**360**	141539	83086	**420**	**757.2**	541200	901860
32-bit	*16×16*	10584	4009	92.88	240	2800	1762	348	547.2	64320	112728
	32×32	83210	31392	94.44	240	21300	11952	360	660	270480	450840
	64×64	660497	248339	**96**	**253.2**	166876	91836	**376.8**	**684**	1082160	1803720

[1] Data transfer latency over the NoC for matrix elements sending and output collection is excluded from the computing latency and performance. [2] Total data transfer latency between the clusters over the NoC. Numbers in bold represent the highest performance for each fixed point size.

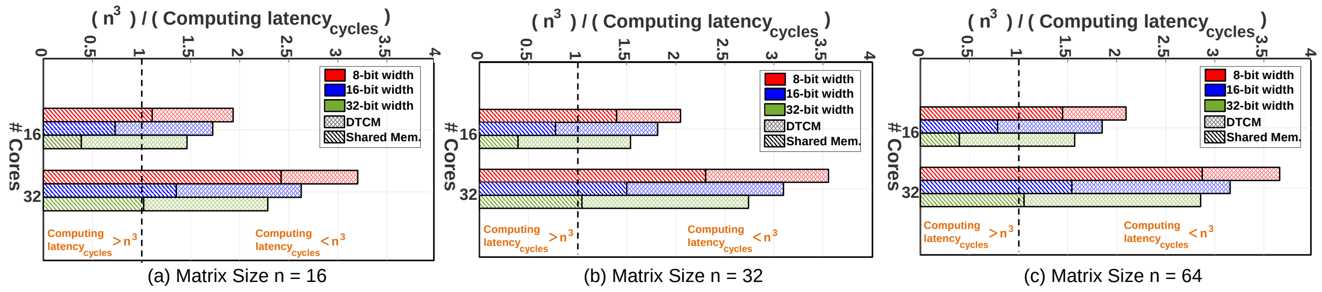


FIGURE 10. Overall matrix multiplication speedup over 16/32 cores for different matrix sizes n with several bit-widths and memory configurations.

and 32 cores (8 clusters) using different bit-width operations and memory types (shared data or DTCM memory). The speedup is calculated by $(n^3)/Computing\,latency_{cycles}$. Where $Computing\,latency_{cycles}$ is the execution time per clock cycles for multiplication and n^3 is the computing complexity of the matrix multiplication. The (red bars) in Figure 10 represents 8-bit width matrix multiplication. It shows a scalable performance of 1.8x by using 8 clusters compared to 4 clusters. While in case of 32-bit width (green bars) multiplication the performance scalability is 1.7x as the numbers of memory operations increased compared to the 8-bit width multiplication case. Moreover, using DTCM instead of shared data memory increases the multiplication speedup by at least the double (>2x) in cases of 32, 16-bit width multiplication as shown in Figure 10.

C. MATRIX LU DECOMPOSITION (USE CASE)

LU decomposition is a key function for linear algebra calculations required by signal/image processing applications. LU decomposition factors a square matrix of size (n×n) as a product of a lower (L) and upper (U) triangular matrices.

TABLE 5. Performance evaluation of the LU decomposition test case.

Matrix Size [32-bit]	Computing Latency [Cycle]		Data Transfer Latency [Cycle]
	16 Cores	*32 Cores*	
32×32	9360	3880	270562
64×64	51172	25128	541200

It is performed by a sequence of Gaussian eliminations to form A=LU. LU decomposition can be performed for a non-singular matrix A in a column-oriented method as shown in Algorithm 1. The algorithm is parallelized over the many-core architecture as follows. Matrix A is divided column-wise over the number of processing clusters. Each cluster receiving a sub-matrix of A and computes a sub-matrix for

Algorithm 1 LU Decomposition

```
1:  $n : rows[A]$
2:  Loop-1: $A_{cols}$ are divided by # clusters = 4 or 8
3:  for $k \leftarrow 1$ to $n$ do
4:      $u_{kk} \leftarrow a_{kk}$
5:      Loop-2: $A_{sub\_cols}$ are divided by # cores = 4
6:      for $i \leftarrow k+1$ to $n$ do
7:          $l_{ik} \leftarrow a_{ik}/u_{kk}$
8:          $u_{ki} \leftarrow a_{ki}$
9:      end for
10:     Loop-3: $A_{sub\_cols}$ are divided by # cores = 4
11:     for $i \leftarrow k+1$ to $n$ do
12:         for $j \leftarrow k+1$ to $n$ do
13:             $a_{ij} \leftarrow a_{ij} - l_{ik} \times u_{kj}$
14:         end for
15:     end for
16: end for
17: return $L$ and $U$
```

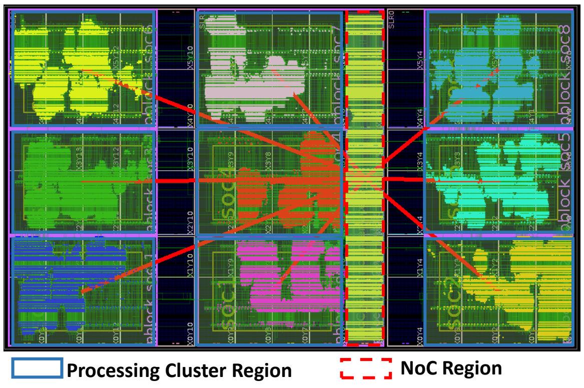


FIGURE 11. Placement of 9 reconfigurable processing clusters and a static 3x3 mesh NoC on the Xilinx Virtex Ultrascale+ XCVU9P floorplan.

L and U as shown by Algorithm 1 (Loop-1) then store the results in its shared data memory. Inside each cluster, the A sub-matrix is divided again column-wise over the number of cores (PEs) per cluster as shown by Algorithm 1 (Loop-2, Loop-3). Therefore, the computation is conducted in parallel over the number of PE as each PE is responsible to produce a sub-matrix for L and U. The LU calculation requires MAC and reciprocal operations with a total computing complexity of $\sim \mathcal{O}(\frac{2}{3}n^3)$. For performance evaluation, 16 and 32 cores many-core configurations are used to compute the LU decomposition for two square matrix sizes of (32×32, 64×64). In addition, an extra cluster is used for matrix generation and results collection. Table 5 shows the computing and data transfer latency for the different many-core and matrix sizes configurations. Whereas the data transfer latency is higher than the computing latency due to the software cost of data transfer latency over the NoC (described in Section III) plus the memory read/write overhead by each PE.

D. RUN-TIME RECONFIGURATION

In order to provide high flexibility and configurability for the proposed many-core architecture. Xilinx dynamic partial reconfiguration (DPR) technique is supported to change the configuration of the many-core architecture during run-time without the need to synthesize the complete architecture for every configuration changes. For the many-core implementation setup depicted in Figure 6, the FPGA floorplan is divided into 9 reconfigurable regions to host a reconfigurable processing cluster each and one static region including a 3x3 NoC, a single MicroBlaze, and testing peripherals as shown in Figure 11. Each processing cluster can host a dynamic number of PEs depends on the computation requirements of the target applications. In addition, the shared memory sizes can be changed at run-time by uploading a new partial bitstream with a new memory configuration.

Moreover, DPR provides the feature to change the running application over a cluster at run-time by changing the contents of the shared instruction memory using a new partial bitstream. For experimental analysis, an external reconfiguration technique using the JTAG interface is developed to load the partial bitstreams from an external device (e.g. PC) to the Virtex Ultrascale+ FPGA configuration memory. Table 6 shows the resources utilization of the reconfiguration region assigned to a single cluster and the percentage of usage by a different number of PEs. Besides, the measured reconfiguration time for a single reconfiguration region is equal to ∼0.9s and it could be reduced by using SelectMap or PCIe interfaces.

TABLE 6. DPR resources utilization and reconfiguration time.

	Resources Utilization			
	LUT	**FF**	**BRAM**	**DSP**
Reconfigurable Region	103680	207360	192	768
2 PEs (%)	16%	3.4%	62.8%	1.6%
3 PEs (%)	23%	5%	73%	2.34%
4 PEs (%)	30.4%	6.3%	83.6%	3.14%
Reconfiguration Time (s) via JTAG	**0.91 s**			

V. CONCLUSION

This work proposes a novel modular RISC-V based many-core architecture with a high degree of design scalability and regularity for FPGA platforms. The architecture is based on a configurable cluster-based design connected through a scalable generic NoC architecture. The processing cluster features a RISC-V based multicore computing architecture supporting a software managed shared and local memory systems. Moreover, it supports design-/run-time configurable number of PEs and memory sizes based on the target application requirements. Several building blocks for PEs, memory systems and interconnections are developed and designed based on a modular and regular manner and implemented in the form of configurable IP blocks to be integrated together to generate different many-core taxonomies. The proposed approach aims to ease the development and the implementation of homogeneous/heterogeneous many-core architectures by reducing the design time and the non-recurrent engineering costs. The many-core architecture is evaluated based on different architecture configurations to measure the design scalability in terms of growing numbers of processing clusters, the number of PEs per cluster and the achievable memory bandwidth as well the hardware resources utilization of the architecture building blocks. The results show a high degree of computing and memory bandwidth scalability by using local memory blocks per processing cluster for memory intensive applications e.g. (matrix multiplication) compared to using shared memory blocks. Moreover, the used NoC architecture allows the integration of a scalable number of processing clusters for compute intensive applications.

As for the future work, it is planned to integrate custom hardware accelerators for linear algebra functions to the proposed architecture in order to provide a real-time computing architecture for signal-/image processing applications. In addition, an internal DPR controller coupled with a RISC-V core will be implemented inside a static cluster to speedup the reconfiguration time to support real-time requirements.

REFERENCES

[1] X, Xu, et al. "Scaling for edge inference of deep neural networks." Nature Electronics 1.4, 2018, pp. 216-222.

[2] G. P. Fettweis et al., "5G-and-Beyond Scalable Machines," In IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC), Peru, 2019, pp. 105-109.

[3] R. Airoldi, F. Garzia, T. Ahonen, and J. Nurmi, "Ninesilica: A Homogeneous MPSoC Approach for SDR Platforms," In Computing Platforms for Software-Defined Radio. Springer, 2017, pp. 107-119.

[4] S. Gregor et al., "The CoreVA-MPSoC: A multiprocessor platform for software-defined radio." In Computing Platforms for Software-Defined Radio Springer, 2017, pp. 29-59.

[5] S. Haas et al., "A heterogeneous SDR MPSoC in 28nm CMOS for low-latency wireless applications," In 54th IEEE Design Automation Conference (DAC), Austin, 2017, pp. 1-6.

[6] A. Amid et al., "Chipyard: Integrated Design, Simulation, and Implementation Framework for Custom SoCs," In IEEE Micro, vol. 40, no. 4, 2020, pp. 10-21.

[7] A. Kurth, P. Vogel, A. Capotondi, A. Marongui, and L. Benini, "HERO: Heterogeneous Embedded Research Platform for Exploring RISC-V Manycore Accelerators on FPGA," In Proceedings of Computer Architecture Research with RISC-V workshop (CARRV), 2017, pp. 1-7.

[8] J. Balkind et al., "OpenPiton+ Ariane: The First Open-Source, SMP Linux-booting RISC-V System Scaling From One to Many Cores," In 3rd Workshop on Computer Architecture Research with RISC-V (CARRV), 2019, pp. 1-6.

[9] L. P. Carloni, "Invited: The case for Embedded Scalable Platforms," In 53nd IEEE Design Automation Conference (DAC), Austin, 2016, pp. 1-6.

[10] D. Giri, K. Chiu, G. Di Guglielmo, P. Mantovani and L. P. Carloni, "ESP4ML: Platform-Based Design of Systems-on-Chip for Embedded Machine Learning," In Design, Automation & Test in Europe Conference Exhibition (DATE), Grenoble, 2020, pp. 1049-1054.

[11] S. Savas, Z. Ul-abdin, and T. Nordström, "Designing domain-specific heterogeneous architectures from dataflow programs," In Computers 7.2, 2018, pp. 1-28.

[12] S. Savas, Z. Ul-Abdin, and T. Nordström, "A framework to generate domain-specific manycore architectures from dataflow programs," In Microprocessors and microsystems 72 , 2020, pp. 1-18.

[13] M. A. Elmohr et al., "RVNoC: A Framework for Generating RISC-V NoC-Based MPSoC," In 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP), Cambridge, 2018, pp. 617-621.

[14] M. Ruaro, L. Caimi, V. Fochi, and F. Moraes, "Memphis: a framework for heterogeneous many-core SoCs generation and validation," In Design Automation for Embedded Systems, Springer, 2019, pp 103-122.

[15] M. Vestias, and H. Neto, "A Many-Core Overlay for High-Performance Embedded Computing on FPGAs," In First International Workshop on FPGAs for Software Programmers (FSP), 2014, pp. 71-76.

[16] W. José, H. Neto, and M. vestias, "A Many-Core Co-Processor for Embedded Parallel Computing on FPGA," In Euromicro Conference on Digital System Design, 2015, pp. 539-542.

[17] J. Gray, "GRVI Phalanx: A Massively Parallel RISC-V FPGA Accelerator Accelerator," In IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), Washington, 2016, pp. 17-20.

[18] D. Petrisko et al., "BlackParrot: An Agile Open Source RISC-V Multicore for Accelerator SoCs," in IEEE Micro (Early Access), 2020, pp. 1-8.

[19] J. Ax et al., "CoreVA-MPSoC: A Many-Core Architecture with Tightly Coupled Shared and Local Data Memories," In IEEE Transactions on Parallel and Distributed Systems, 2018, pp. 1030-1043.

[20] L. Benini, E. Flamand, D. Fuin, and D. Melpignano, "P2012: Building an ecosystem for a scalable, modular and high-efficiency embedded computing accelerator," In Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, 2012, pp. 983-987.

[21] A. Olofsson, T. Nordström, and Z. Ulabdin, "Kickstarting high-performance energy-efficient manycore architectures with Epiphany," In 48th Asilomar Conference on Signals, Systems and Computers, CA, 2014, pp. 1719-1726.

[22] B. D. de Dinechin et al., "A clustered manycore processor architecture for embedded and accelerated applications," In IEEE High Performance Extreme Computing Conference (HPEC), MA, 2013, pp. 1-6.

[23] S. Hesham, D. Göhringer, and M. A. Elghany,"ARTNoCs: An Evaluation Framework for Hardware Architectures of Real-Time NoCs," In IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Chicago, 2016, pp. 259-264.

[24] P. Davide Schiavone et al., "Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for Internet-of-Things applications," In International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS), 2017, pp. 1-8.

[25] A.Traber et al., "RI5CY: User Manual, " April 2019, [Online]. Available: pulp-platform.org/docs/ri5cy_user_manual.pdf

[26] A. Kamaleldin, M. Ali, P. Rad, M. Gottschalk, and D. Göhringer, "Modular Memory System for RISC-V Based MPSoCs on Xilinx FPGAs," In IEEE 13th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), Singapore, 2019, pp. 68-73.

[27] pulp-riscv-gnu-toolchain [Online]. Available: github.com/pulp-platform/pulp-riscv-gnu-toolchain

[28] ESP: the open-source SoC platform [Online]. Available: https://github.com/sld-columbia/esp

AHMED KAMALELDIN (Graduate Student Member, IEEE) received the B.Sc. and M.Sc. degrees (Hons.) in Electronics and Electrical Communications Engineering from Cairo University, Egypt, in 2012 and 2017 respectively. Currently, he is working as a research assistant at the Adaptive Dynamic Systems (ADS) chair, TU Dresden, Germany. His current research interests include reconfigurable computing, multiprocessor systems-on-chip (MPSoCs), networks-on-chip, hardware-software codesign and runtime systems.

SALMA HESHAM (Member, IEEE) received the M.Sc. degrees in Electronics Engineering from German University in Cairo (GUC), Egypt, in 2012, with DAAD exchange thesis semester in Ulm University, Germany. From 2012-2019 she was working as an Assistant Lecturer and Research Assistant in GUC, Egypt. From 2012 to 2014 she was involved in collaborative research work with the TU-Darmstadt, Germany and from 2015 to 2019 she was a bilateral doctoral researcher between Ruhr-Universität-Bochum (RUB), Germany and German University in Cairo (GUC), Egypt. From 2019 to 2020 she was further part of collaborative research work with the Adaptive Dynamic Systems (ADS) chair, TU-Dresden, Germany. In 2019, she received her PhD (summa cum laude) in Electrical Engineering and Information Technology in a dual degree frame between RUB, Germany and GUC, Egypt. Since September 2019 she is a lecturer at German University in Cairo (Berlin Exchange Campus). Her research interests include hardware design of digital circuits and systems, real-time networks-on-chip, multiprocessor systems-on-chip (MPSoCs), and asynchronous circuits designs.

DIANA GÖHRINGER (Member, IEEE) is professor for Adaptive Dynamic Systems at TU Dresden, Germany. From 2013 to 2017 she was an assistant professor and head of the MCA (application-specific Multi-Core Architectures) research group at Ruhr-University Bochum (RUB), Germany. Before that she was working as the head of the Young Investigator Group CADEMA (Computer Aided Design and Exploration of Multi-Core Architectures) at the Institute for Data Processing and Electronics (IPE) at the Karlsruhe Institute of Technology (KIT). From 2007 to 2012, she was a senior scientist at the Fraunhofer Institute of Optronics, System Technologies and Image Exploitation IOSB in Ettlingen, Germany (formerly called FGAN-FOM). In 2011, she received her PhD (summa cum laude) in Electrical Engineering and Information Technology from the Karlsruhe Institute of Technology (KIT), Germany. She is author and co-author of 1 book, 10 invited book chapters and over 130 publications in international journals, conferences and workshops. Additionally, she serves as technical program committee member in several international conferences and workshops. She is reviewer and guest editor of several international journals. Her research interests include reconfigurable computing, multiprocessor systems-on-chip (MPSoCs), networks-on-chip, simulators/virtual platforms, hardware-software-codesign and runtime systems.