IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received March 28, 2022, accepted April 13, 2022, date of publication April 18, 2022, date of current version April 28, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3168686

AGILER: An Adaptive Heterogeneous Tile-Based Many-Core Architecture for RISC-V Processors

AHMED KAMALELDIN, (Graduate Student Member, IEEE), AND DIANA GÖHRINGER, (Member, IEEE)
Technische Universität Dresden, Chair of Adaptive Dynamic Systems, 01062 Dresden, Germany

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

This work was supported in part by the German Research Foundation (“Deutsche Forschungsgemeinschaft”) (DFG) under Project 287022738 TRR 196 for project S05, and in part by the Open Access Funding by Saxon State and University Library Dresden (SLUB).

ABSTRACT Tile-based many-core architectures are extensively used in modern system-on-chip designs to achieve scalable computing performance with adequate energy efficiency. Heterogeneity is the key element to boost computing performance and keep energy consumption under certain limits for several application domains. However, the steady increase of using many custom heterogeneous tiles leads to an expansion in design and integration cost with limited tiles re-usability. The recent widespread of open-source RISC-V ISA provides the potential to develop modular compute units that can be used for many application domains with high reduction in non-recurring engineering costs. The motivation of this work is to bring design modularity and adaptability features for heterogeneous tile-based many-core architectures by increasing their flexibility to realize different many-core configurations with less design time and costs. In this work, AGILER is proposed as an adaptive tile-base many-core architecture for heterogeneous RISC-V based processors. The proposed architecture consists of modular and adaptable heterogeneous multi-/single-core compute tiles that supports 32-/64-bit RISC-V ISAs with different memory hierarchies. Inter-tile communication is developed based on a scalable network-on-chip architecture to achieve a high degree of system scalability. AGILER supports run-time adaptation through a custom internal reconfiguration manager for dynamic and partial reconfiguration over Xilinx FPGAs. Evaluation results demonstrate that the proposed architecture features a scalable computing performance up to 685 MOPS for 8×32-bit tiles and 316 MOPS for 8×64-bit tiles with a scalable memory bandwidth up to 7.4 GB/s. AGILER is evaluated on Xilinx Virtex Ultrascale+ FPGA with a maximum reconfiguration time of 38.1 ms for a single compute tile.

INDEX TERMS Many-core architecture, parallel computing, RISC-V, network-on-chip (NoC), field programmable gate array (FPGA), reconfigurable computing.

I. INTRODUCTION

Machine learning and data-centric applications constitute the main driving forces for computing’s rapid evolution. Over the past decade, several computing paradigms have been introduced seeking to increase computing performance scaling and energy efficiency in order to cope with the emergence of new application classes with massive and irregular data sets [1]. Among those computing paradigms are compute-centric architectures which are still leveraged in the mainstream multi-/many-cores System-on-Chip (SoCs) developed by industry and academia for several application domains [2], [3]. Compute-centric systems went through a tremendous evolution from multi-core homogeneous architectures to highly heterogeneous architectures with big, little cores (e.g. ARM Big.Little [2]) and application-specific accelerators (e.g. Google TPU [4]).

Despite the high-performance gain of heterogeneous architectures, the increasing numbers of heterogeneous elements are limited by the system interconnects scalability and therefore the degree of compute performance scalability [5]. This obstacle of compute-performance scaling is referred to as the scalability wall. Therefore, tile-based architectures are developed for highly scalable many-core systems with

The associate editor coordinating the review of this manuscript and approving it for publication was Bidyadhar Subudhi.

a growing capacity of heterogeneous compute elements. The degree of scalability for tile-based architectures relies on the inter-tile communication fabric which is on recent many-core approaches depends on scalable network-on-chip (NoC) variant topologies. However, the design and development of tile-based many-core architectures is a cumbersome process in terms of development time and costs. Especially if target application domains require a high capacity of heterogeneous compute tiles as is the case in recent computing devices that support a wide range of application domains [6], [7].

Consequently, several research approaches have been proposed for adaptive and self-aware many-core systems to allow the re-usability and reconfigurability of many-core architectures to be adjusted according to multiple requirements for different application domains [8], [9]. As a result, an expected reduction in development time and cost can be achieved by the adoption of adaptive many-core approaches. However, adaptive many-core approaches require a sort of modularity of hardware components to ensure proper integration and communication between them after the adaptation process. In tile-based many-core architectures, modularity can be achieved first by using a unified communication method between heterogeneous compute tiles through NoC, or advanced bus-based architectures. In addition, a heterogeneous set of compute tiles that share the same inter-tile interfaces and apply the same communication protocol over the many-core communication medium as well as a unified parallel programming model.

Therefore, modularity and adaptability are keys to reducing design and integration time and promoting the commodity of many-core architectures for emerging application domains. Recently, the proliferation of the open-source instruction set architecture (ISA) by RISC-V [10] contributes to increasing the level of modularity and openness for compute units or processing elements that can be used by tile-based many-core architectures. Various types of general purpose or specialized RISC-V based processors [11] can be selected to create several types of heterogeneous compute tiles with different computing specifications for variant applications requirements.

Existing research platforms do not fully meet modularity and self-adaptability requirements for heterogeneous RISC-V based multi-/many-core systems. In our previous work [12], we proposed a modular architecture for homogeneous RISC-V based many-core architectures with one type of compute tile that can be configured during run-time through an external device (i.e. PC). Additionally, the proposed compute tile supports one type of RISC-V ISA (RV32IMC) as well as the need for an external soft-core manager (i.e. Xilinx Microblaze) to manage data transfer between the RISC-V-based many-core system and external peripherals (i.e. DDR, and UART). Accordingly, the previous homogeneous platform is restricted to a single RISC-V ISA that can not meet the demands of modern heterogeneous computing architectures.

Therefore, in this work, we propose AGILER as a novel self-adaptive tile-based many-core architecture for heterogeneous RISC-V based many-core configurations targeting FPGAs without the need of an external device/manager for reconfiguration and data transfer. The proposed architecture satisfies the requirements of self-adaptability and modularity for highly scalable heterogeneous RISC-V based many-core systems on FPGAs. Our contributions in this work rely on the following:

- Implementation of modular heterogeneous RISC-V based processing elements supporting RV32/RV64-ISAs with tightly coupled parameterized scratchpad memory and unified AXI interfaces to be used as modular PEs within compute tiles.
- We provide a heterogeneous set of compute tiles for multiple RISC-V ISAs. Compute tiles feature a multi-/single-core architecture using heterogeneous RISC-V based PEs for 32-/64-bit ISAs. In addition, a main/primary processing tile is developed using multi 64-bit RISC-V processors as the main and permanent compute tile for the proposed many-core system.
- Development of a highly scalable tile-based many-core architecture using a generic NoC architecture as the main interconnect medium for inter-tile communication. The required communication model and programming method are provided to support parallel tasks execution and interaction over heterogeneous compute tiles.
- A run-time reconfiguration manager is developed for multi-core RISC-V compute tiles in order to support the adaptability feature for the proposed architecture by self-managing the dynamic partial reconfiguration (DPR) process from the main processing tile at run-time.

As a result, the proposed architecture satisfies the compute performance scalability using a scalable mesh-based NoC topology for inter-tile communication with a variant set of heterogeneous compute tiles. Each compute tile features a configurable multi-/single-core architecture that can be configured with variant number and types of RISC-V based PEs. Moreover, shared and local memory hierarchies with parameterized sizes are supported per each compute tile. Therefore, the proposed architecture provides the flexibility for tailoring several many-core configurations for compute or memory bound applications. Furthermore, a unified communication model and a bare metal programming method are supported to facilitate the parallel execution of several application tasks over many compute tiles. In this work, the implementation and performance evaluation of the proposed architecture are conducted targeting a Xilinx Ultrascale+ FPGA. The tile-based many-core architecture is evaluated based on compute performance scalability, achievable memory bandwidth and resources utilization for different heterogeneous configurations using several benchmarks. Furthermore, the many-core architecture supports run-time adaptation through an internal reconfiguration manager using dynamic and partial reconfiguration technology on Xilinx FPGAs. Finally, the modularity

and adaptability features of the proposed architecture allow the flexibility to be ported to other Xilinx FPGA series. Accordingly and to the best of our knowledge, our proposed architecture is the first heterogeneous tile-based many-core architecture for multiple RISC-V ISAs that supports self-adaptation using internal run-time DPR manager for several heterogeneous many-core configurations on FPGAs.

The rest of the article is structured as follows: Section II discusses background and related work. AGILER architecture and its design approach and components are introduced in Section III. Evaluation and experimental results are presented and analyzed in Section IV. Finally, Section V draws a conclusion about this work and provides further directions for future work.

II. BACKGROUND AND RELATED WORK

During the past decade, several research works have proven the feasibility of tile-based many-core architectures to provide scalable compute performance in parallel with energy efficiency achievement [13]. Many sorts of homogeneous and heterogeneous tile-based architectures have been proposed [14]-[16] targeting pre-defined application domains prior to the design process. Therefore, the baseline tile-based many-core architecture consists of custom set of heterogeneous or homogeneous compute tiles with specialized communication fabric and memory hierarchies which make them highly customized for specific application domains. Accordingly, rapid evolution of application workloads (e.g. machine learning, 5G, 6G algorithms) increases the difficulties for baseline many-core systems to be quickly adapted to changing workloads requirements. This insufficiency necessitates a movement toward modular and configurable tile-based many-core systems, where compute tiles are configurable and re-usable to seamlessly create several many-core implementations using modular and re-usable tiles.

As a result, the development of many-core systems will become inexpensive and affordable for small scale companies and academic research, and as seamless as developing software applications. Meanwhile, the emergence of agile hardware design methodologies and open-source RISC-V ISAs [17], [18] alleviate design costs and efforts (e.g. validation, integration, verification) which can contribute to the proliferation of modular and configurable tile-based many-core frameworks. However, there are limited number of studies that seek to provide modular and configurable tile-based frameworks that can be adapted either at design or run-time to fit with different application requirements. We provide here a review of related work for tile-based many-core architectures using open-source compute units based on RISC-V ISAs targeting both FPGA and ASIC platforms.

OpenPiton [19] has been proposed as an open architecture framework to realize large scale many-core architectures reducing traditional difficulties for many-core systems development. The architecture consists of regular homogeneous compute tiles, multiple tiles are grouped into many-core chips which are connected together through a coherent multi-layer NoC. Each compute tile features a single RV64GC Ariane core [20] with private instruction and data caches. OpenPiton is only configurable at design time with the flexibility to select the number of tiles/cores, memory size configuration, and cores extension with floating point unit (FPU) or stream processing unit. Despite the high level of scalability, the architecture supports a single core compute tile which increases the size and complexities for inter-tile connections by increasing number of cores as well as the required programming model. Furthermore, Kamaleldin *et al.* [12] provides a scalable FPGA based RISC-V many-core platform using a lightweight NoC to realize several RISC-V based many-core taxonomies supporting external run-time configuration through an external device. It features a homogeneous multi-core compute tile based on RV32 ISA, the cores are connected through an AXI interconnect within the tile with shared and local memory. Similarly, Andromeda framework [21] is proposed for early stage applications explorations using homogeneous RV64 based tiles connected with a single mesh NoC. However, the latest framework lacks the run-time configuration feature supported by [12]. To enhance inter-tile communication, MemPool [22] provides a scalable RISC-V based many-core architecture with low latency inter-tile interconnect. The architecture consists of multiple tiles groups to form a cluster of homogeneous computing tiles. Each tile hosts four RV32IMC cores with L1-shared data memory and shared instruction cache. Compute tiles are connected through full-crossbars interconnects to ensure low data transfer latency between tiles as well as between several tiles groups. The architecture features a regular design of compute tiles with configurable L1-shared memory size during design time. Despite the high scalability and design regularity of the aforementioned frameworks and architectures, they do not support heterogeneous compute tiles with multiple ISAs or custom hardware accelerators.

Therefore, Tapasco [23] has been proposed as an open-source framework to generate multi-/many-core SoCs with custom PEs. The framework allows seamless integration of any HLS-based accelerator within a scalable architecture by auto-generating required wrapping logic and interfaces for interconnection and communication within the architecture. The compute tile consists of multiple types of PEs, RV32 based PE or HLS-based PE are supported within the tile. The architecture uses multiple cascaded crossbars to connect multiple PEs within the tile as well as the connection between several tiles. However, the architecture could suffer from low scalability due to the limited bandwidth of crossbars. Accordingly, Savas *et al.* [24] proposed a framework to design a highly scalable domain-specific heterogeneous many-core architecture from application data flow graphs. The framework is based on a heterogeneous tile-based architecture, each tile consists of a 64-bit RISC-V core with a tightly coupled HLS-based accelerator and private local data memory. To ensure high scalability, the framework supports

TABLE 1. Comparison of state-of-the-art approaches for RISC-V based many-core architectures.

References	Heterogeneity Level		Many-Core Architecture		Configurability	Memory Hierarchy per Tile	Target
	Custom HW Accelerator	Multiple ISAs	Compute Tile Architecture	Communication			
OpenPiton+Ariane [20]	-	Only RV64	Single-Core	NoC^{+}	Design-Time	Local	FPGA
Kamaleldin et al.[12]	RTL/HLS-based	Only RV32	Multi-Core	Bus system*/NoC^{+}	External Run-Time	Local/Shared	FPGA
Andromeda [21]	-	Only RV64	Multi-Core	Bus System*/NoC^{+}	Design-Time	Shared	FPGA
MemPool [22]	-	Only RV32	Multi-Core	Cascaded Crossbars^{+}	Design-Time	Shared	ASIC
TaPaSCo [23]	HLS-based	Only RV32	Multi-Core	Cascaded Crossbars^{+}	Design-Time	Local	FPGA
Savas et al. [24]	HLS-based	Only RV64	Single-Core	NoC^{+}	Design-Time	Local	FPGA
GRVAPhalanx [25]	RTL-based	Only RV32	Multi-Core	Crossbar*/NoC^{+}	Design-Time	Shared	FPGA
Blackparrot [26]	RTL-based	Only RV64	Single-Core	NoC^{+}	Design-Time	Local	ASIC
Manticore [27]	-	RV64/RV32	Multi-Core	Cascaded Crossbars^{+}	Design-Time	Shared	ASIC
ESP [28]	HLS-based	Only RV64	Single-Core	NoC^{+}	Design-Time	Local	FPGA
This Work *(AGILER)*	RTL/HLS-based	RV64/RV32	Multi-Core/ Single-Core	Bus system*/NoC^{+}	Internal Run-Time (Self-Configurable)	Local/Shared	FPGA

*Bus systems or crossbars are used in those architectures to connect different PEs/cores and other related tile components inside a single compute tile.
$^{+}$Inter-tile communication method.

inter-tile communication through a lightweight NoC. Similarly, the GRVI Phalanx architecture [25] is proposed for extreme scalability for FPGA-based many-core architectures. It efficiently uses the FPGA resources to place hundreds of simple 32-bit RISC-V PEs within multi-core compute tiles. An FPGA-based NoC architecture is used for inter-tile communication for a high degree of scalability. However, the GRVI architecture lacks an efficient programming model for parallel execution and data sharing over compute tiles.

Therefore, Blackparrot [26] has been proposed as the first heterogeneous and Linux capable open-source multi-core platform. The platform features a coherent NoC-based architecture as the inter-tile communication fabric, three NoCs are used for internal data, shared memory, and external peripherals. Three heterogeneous types of compute tiles are provided for general purpose (GPP) computing based on RV64, coherent, and streaming custom hardware accelerators. The platform supports several programming models based on application requirements. Furthermore, multiple RISC-V ISAs are supported by several many-core platforms [27], [28] for more compute resources and efficiency to support several workloads requirements. Manticore [27] has been proposed for high performance computing workloads, it consists of hundreds of 32-/64-bit RISC-V cores grouped into multiple tiles. Each tile hosts eight RV32 cores with L1-shared memory and shared instruction cache. Multiple tiles are creating a cluster, each cluster contains quad-core RV64 for controlling and management. Cascaded crossbars are used for inter-tile and inter-cluster communication. On the other hand, ESP framework [28] has been proposed to realize different many-core configurations for embedded platforms. ESP features a scalable coherent NoC-based architecture consisting of multiple heterogeneous compute tiles of RISC-V based processors and custom hardware accelerators. Each compute tile hosts a single compute element. RV64 ISA is supported inside a general purpose tile as well as a compute tile for HLS-based accelerator. Similar to Tapasco platform [23], ESP also auto generates the required wrapping logic and interfaces to seamless integrate custom hardware accelerators into the accelerator tile. Also, several programming models are supported through bare metal or Linux operating system.

Accordingly, our proposed architecture AGILER differentiates from those above-mentioned work as shown in Table 1 by providing more design space to create several heterogeneous tile-base many-core configurations. The proposed architecture provides a configurable set of compute tiles that can support two RISC-V ISAs (RV32, RV64). Moreover, compute tiles can be configured to support single-core and multi-core architectures with the flexibility to support several memory hierarchies. Each compute tile features a private address space which allows the communication between all PEs and shared tile peripherals through shared data memory via an AXI interconnect. For inter-tile communication, a synchronous scalable NoC architecture ARTNoC [29] is used with a message-based communication model to manage data transfer between compute tiles. Furthermore, the proposed architecture is run-time adaptable which provides the flexibility to change types and number of active compute tiles during run-time. Thanks to the internal run-time reconfiguration manager that manages and controls the DPR process on FPGAs.

III. AGILER ARCHITECTURE

AGILER is proposed as an adaptive heterogeneous tile-based many-core architecture targeting FPGA devices. The main motivation is to provide a modular and configurable many-core architecture (either at design or run-time) that can be tailored to support general-purpose and domain-specific computing for several application domains. The architecture consists of two main parts: ⓐ a heterogeneous set of modular and configurable compute tiles that support multiple RISC-V ISAs in single and multi-core tiles architectures with the option to add application-specific hardware accelerators, and ⓑ scalable 2-D mesh NoC with an associated communication model for inter-tile communication. In the following, Section III. A will focus on compute tiles architecture, Section III. B on heterogeneous RISC-V based PEs inside

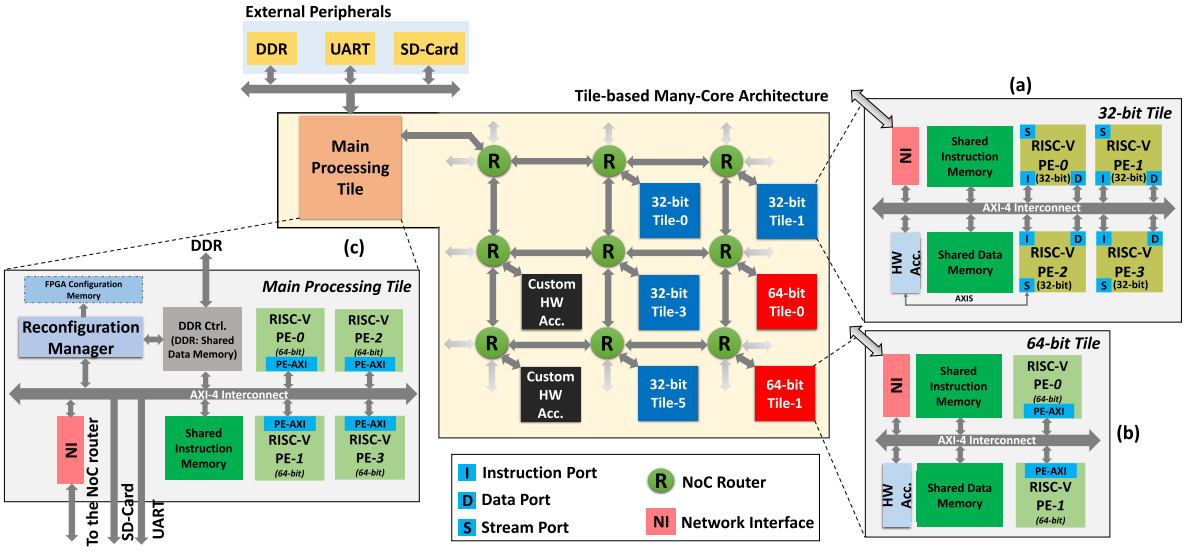


FIGURE 1. Overview of the proposed AGILER architecture with a 3 × 3 tile-based many-core configuration including: (a) 4 × 32-bit compute tiles, (b) 2 × 64-bit compute tiles, (c) the main/primary processing tile, and optional custom hardware accelerator tiles.

compute tiles, Section III. C on scalability and inter-tile communication, Section III. D on run-time reconfiguration management and Section III. E describes the programming method for AGILER architecture.

A. MODULAR AND CONFIGURABLE COMPUTE TILES

Compute tiles are the core of AGILER architecture, they represent the computing nodes for the proposed many-core system. As shown in Figure 1, the many-core architecture consists of three types of heterogeneous tiles that support multiple RISC-V ISAs in addition to custom hardware accelerators. All compute tiles have a regular design pattern based on a bus-based architecture. The compute tile tightly couples single or multiple RISC-V based PEs with shared instruction and data scratchpad memories using a shared AXI interconnect. Therefore, all PEs in a single tile share a common private address space which allows the communication between them and accessing tile's shared memories and memory-mapped peripherals via AXI interconnect. To enhance the memory bandwidth, shared instruction and data memories are implemented as dual-ported BRAM/URAM blocks. Therefore, two memory read/write (R/W) channels can be established across AXI interconnect to handle two memory requests simultaneously. Shared instruction memory is implemented as read-only BRAM memory which is used as a boot memory during the memory initialization stage to load the compiled binary file for execution. Each compute tile implements a uniform memory access (UMA) architecture, where each PE can access shared data and instruction scratchpad memories connected to the AXI interconnect as a slave memory-mapped peripheral [30]. In the UMA architecture, each PE experiences the same bandwidth and access latency to the memory. However, the overall memory bandwidth is divided between the number of PEs per tile. Therefore, the growing number of PEs connected to the AXI interconnect leads to an increase in memory access latency per each PE and increases the probability of memory congestion due to limited crossbar bandwidth. In order to reduce memory congestion per tile, we use an open-source high-performance coherent AXI interconnect implementation [31], [32]. The AXI interconnect is based on a fully-connected crossbar where each slave port has a dedicated connection to each master port. The crossbar supports up to five independent data transaction channels for R/W and applies a round robin arbitration scheme. However, the memory bandwidth scalability is limited and starts to saturate after a certain number of PEs. Therefore, each tile supports a maximum number of four PEs to ensure a congestion free tile implementation. As shown in Figure 1 (a), the first compute tile is a 32-bit processing node with four RV32 PEs. Each PE consists of a single RV32IMC core with scratchpad local memory and the PE is compatible with AXI 32-bit standards to seamlessly connect it to the tile AXI interconnect. Second compute tile is a 64-bit processing node that can be configured to support single or dual RV64 based PEs as shown in Figure 1 (b). Similar to RV32 PE, the RV64 PE consists of a single RV64IMAC core with local scratchpad memory that can be seamlessly connected to the tile interconnect via AXI 64-bit interfaces. The third type of compute tile is shown in Figure 1 (c), it is the main and permanent processing tile of the AGILER many-core architecture. The main processing tile is based on a 64-bit quad-core architecture with shared instruction and data memory. The off-chip DDR memory is used as shared data memory for large capacity of data sharing between the four RV64 PEs. While the shared instruction memory uses on-chip BRAM blocks similar to other compute tiles. Moreover, the main processing tile control and manage external many-core peripherals (i.e. SD-card, UART) and it can be extended to

support other types of off-chip peripherals. Furthermore, the reconfiguration manger unit and its associated components (i.e. direct memory access (DMA), internal configuration access port (ICAP) controller) are hosted and managed inside the main processing tile. Therefore, the main process tile is the static part of our AGILER many-core architecture which is primarily responsible for many-core management and configuration as well as taking part of computational workload with other tiles. Finally, all compute tiles are equipped with a 32-bit generic network-interface (NI) to connect them to the NoC routers for inter-tile communication.

B. HETEROGENEOUS RISC-V BASED PROCESSING ELEMENTS

PEs are the main computing units for the proposed tile-based architecture. The 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. In this section, two RISC-V based PEs based on RV32/RV64 ISAs for 32-/64-bit compute tiles are described as follow.

1) 32-BIT PE

The 32-bit PE consists of a single open-source RI5CY soft-core processor [33] with an implemented local tightly coupled memory subsystem for data and instructions as shown in Figure 2. RI5CY core is a 32-bit 4-stage pipeline in-order processor. The core implements a simple RV32IMC ISA with a main arithmetic-logic unit (ALU) and dedicated units for multiplication, division and multiply-accumulate (MAC). Average base instructions loading latency from instruction memory is one clock cycle except for load/store (LD/ST) instructions and other custom instructions which have a minimum latency of 2 clock cycles [34]. The PE features 2 separate tightly coupled memory blocks implemented using on-chip BRAM/URAM for instruction and data as shown in Figure 2 (b). I/D-TCM offer low memory latency of one clock cycle for R/W operations, it also increases data locality for memory-bound applications. All memory blocks have a fixed word size of 32-bit compatible with RV32 ISA. As shown in Figure 2 (b), ITCM is implemented as a dual-ported on-chip BRAM/URAM with a read-only interface to the RI5CY core instruction port (I-Port) for instruction fetching every one 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 on-chip BRAM/URAM with R/W interface to the RI5CY core D-Port. The DTCM is only accessed via its own coupled PE. Therefore, accessing local memory directly by other PEs is prevented and the local data memory has to be transferred to the shared data memory to be accessible by other PEs in compute tile. To allow seamless integration of PE in 32-bit compute tile, the I/D-Ports of the RI5CY core are extended to be compatible with AXI-4 and AXI-Stream standards by implementing (D, I-Bridges) as shown in Figure 2 (b). D, I-Bridges allows the communication between the RI5CY core and tile's memory-mapped peripherals through the shared AXI interconnect. Since the RI5CY core or the PE is the master unit on the proposed system. The PE AXI interfaces are master interfaces that permit a connection to any AXI slave peripherals inside the tile. As shown in Figure 2, 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 the table in Figure 2). Hence, a finite state machine is implemented with seven states covering the read/write states to the (AXI interconnect, AXIS, ITCM_write and DTCM) interfaces. According to the state and the address-range input, the D-Port interfaces (*data_write/read_D*, *valid_D*, *grant_D*) are re-connected to the corresponding interfaces. Similar to the D-Bridge, the I-Bridge is implemented as shown in Figure 2 (b) with a two states FSM for only reading from the ITCM or the shared instruction memory.

2) 64-BIT PE

The 64-bit PE consists of a single open-source Ariana (CVA6) soft-core processor [35] with an implemented local tightly coupled memory subsystem for data and instructions as shown in Figure 3. Ariane core is a 64-bit 6-stage pipeline in-order processor. The used core version in this work is configured to fully implement RV64IMAC [36]. Similar to the 32-bit PE, the tightly coupled memory subsystem is implemented using on-chip BRAM/URAM blocks as shown in Figure 3 (c). All memory blocks have a fixed word size of 64-bit compatible with RV64 ISA. ITCM is implemented as a dual-ported memory with a read-only interface connected directly to the tile main AXI-interconnect. On the other hand, DTCM is implemented with R/W interfaces using also dual-ported memory. In contrast to the 32-bit PE, the used open-source Ariane core is equipped already with interfaces that are compatible with AXI 64-bit standard interfaces (*AXI_resp*, *AXI_req*) to access instruction and data memories as shown in Figure 2 (a). Therefore, the design of 64-bit PE is quite simple and does not require implementing extra I/D-bridges or converters to make native core I/D-Ports compatible with AXI standards. However, an AXI-Master connect is implemented to re-route AXI request and response signals to different memory-mapped slaves peripherals (i.e. I/D-TCM, shared memory, and tile peripherals) as shown in Figure 3. Accordingly, D/I-TCM are directly connected and accessed through the main tile AXI-interconnect in order to reduce the number of crossbars interconnects inside the PE and also memory access latency. Similarly to 32-bit PE, the local memory subsystem per PE is only accessed by its core. Therefore for each PE, I/D-TCM_offsets are inserted to core's *AXI-resp/-req* R/W addresses signals (as shown in Figure 2 (b)) to modify the AXI addresses sent to the shared AXI-interconnect so that each core can access its local memory. As mentioned in the right tables in Figure 3, I/D-TCMs

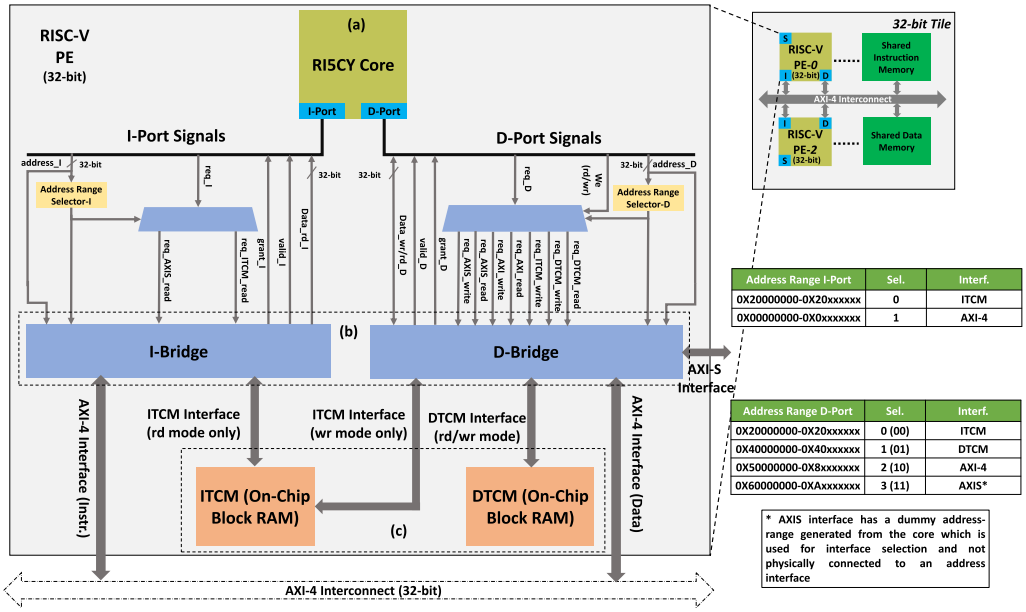


FIGURE 2. Schematic of 32-bit RISC-V based PE showing: (a) open-source RV32IMC (RI5CY) core, (b) instruction and data bridges for converting native I/D signals to AXI-4 interfaces, (c) on-chip I/D TCM and their connection to the RI5CY core through I/D bridges.

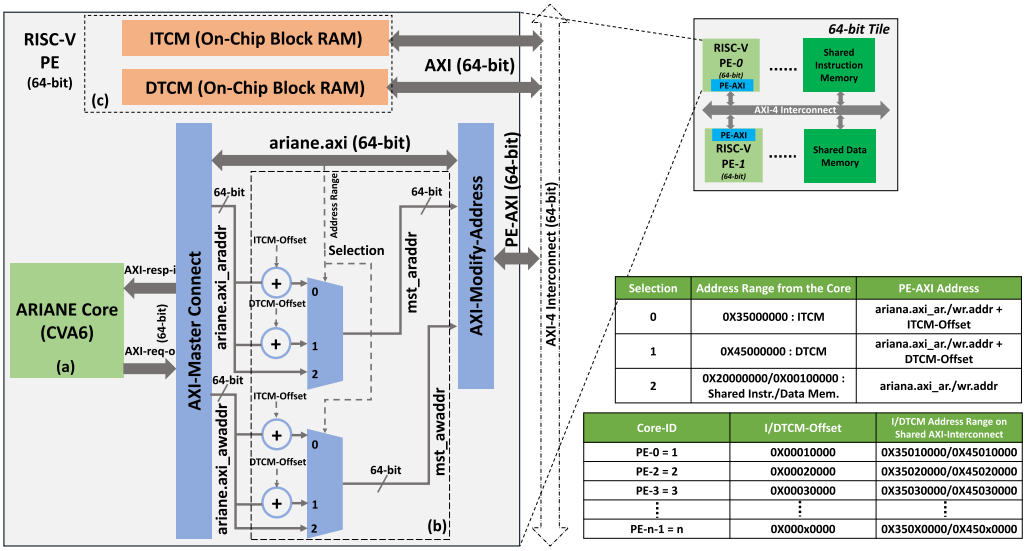


FIGURE 3. Schematic of 64-bit RISC-V based PE showing: (a) open-source RV64IMAC (CVA6/ARIANE) core, (b) address converter to access I/D TCM through main AXI-4 interconnect, (c) on-chip I/D TCM and their connection to the CVA6 core through main AXI-4 interconnect.

for all PEs have the same address range for all cores, but they have unique address ranges on the shared AXI-interconnect. Therefore, addresses modifications are required for each PE/core to access its corresponding I/D-TCM. Accordingly, based on that implementation, we reduced the number of crossbar interconnects to be only the shared AXI interconnects which leads to a decrease in memory access latency (4 clock cycles) to local and shared memory.

C. SCALABILITY AND COMMUNICATION MODEL

In this work, the real-time NoC architecture ARTNoC [29] is used for inter-tile communication and to provide the required high scalability for the proposed many-core architecture. The used NoC provides guaranteed quality of service (QoS) in terms of bandwidth and end-to-end latency. In addition, the NoC router architecture is highly modular and parameterized in terms of I/O ports configurations, switching controls, buffering sizes and routing schemes. A circuit switching based version with an XY-routing algorithm is used in the implementation for less resource utilization compared to a packet-switched NoC. The NoC architecture consists of: a 5 ports circuit-switched router including a control path circuitry and arbiters for path reservation, a crossbar to switch between the I/O ports using a round-robin arbitration scheme,

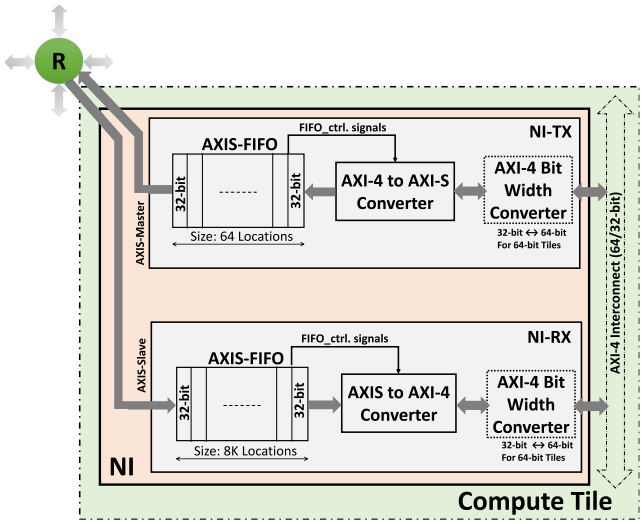


FIGURE 4. A unified network interface (NI) block diagram for many-core compute tiles.

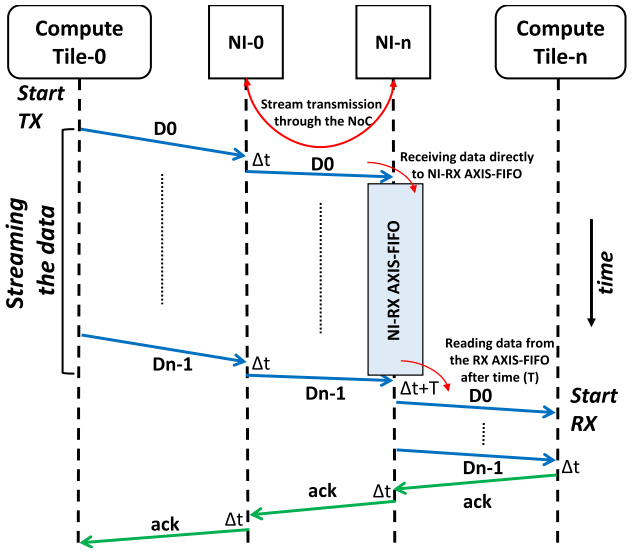


FIGURE 5. Sequence diagram of the message-based communication model between computing tiles over the NoC.

synchronous network links to connect between routers. The circuit-switched NoC reserves a static transmission path between the source and destination routers. This is performed by sending a single flit of 32-bit from the source router containing the X-Y coordinate of the destination router. The router can transmit a single flit every one clock cycle as the I/O router ports are stream interfaces with 32-bit data width.

A generic network interface (NI) is implemented to allow communication between compute tiles and NoC routers. The router I/O interfaces are compatible with the AXI-stream standard. Therefore, the proposed NI architecture is based on a flit-based streaming approach. The NI links between the compute tile crossbar and the AXI-stream interfaces of the router. An overview of the NI internal architecture is shown in Figure 4. The NI has two separated channels for sending and receiving data (NI-TX, NI-RX). It is connected to the AXI interconnect as AXI-slave memory-mapped peripheral that can be accessed by all compute tile PEs. The NI internal architecture consists of: an AXI-stream FIFO of size 64 locations in case of NI-TX and 8K locations in case of NI-RX to store/receive the transmitted and received packet flits, an AXI-stream to AXI-4 converter to connect the AXIS-FIFO and its control signals to the tile AXI-interconnect. As the NoC is based on a 32-bit architecture, an AXI-4 bit width converter is required for 64-bit compute tiles. A single PE can access the NI by setting a synchronization flag (for either sending or receiving) in the shared data memory indicating that the NI is blocked by this PE to prevent data interference by several NI requests from different PEs. The data flow between a PE to a NI is performed by setting a pointer to the data source address in the shared data memory to transmit a specified size of data. The data is transmitted in a form of a group of 32 packet-flits to the NI-TX FIFO. Similarly, in the receiving direction, the received packet-flits are stored in the NI-RX AXIS-FIFO until a reading request comes from a certain PE to start data storing in the shared data memory. Transmitting and receiving of data through NI can be done concurrently as the NI has two separate read/write channels to the AXI-interconnect.

In this work, a message-based communication model is developed over the NoC to control the communication between compute tiles. In other words, the communication model is considered as the network transport layer over the NoC hardware architecture. Figure 5 shows a sequence diagram of the proposed communication model between two compute tiles. The transmission is initiated by any PE in the sender tile. The first transmitted packet contains the X-Y coordinate of the tile destination followed by a stream of payload data packets, each packet contains 32-flits of payload data. The transmitted data will be directly stored in the NI-RX AXIS-FIFO of the received tiles. Therefore, the receiving tile will not be blocked while the data is transmitted from the sending tile and it can load the received data to shared data memory after a certain amount of time. However, the maximum amount of data that can be transmitted without blocking is equal to the size of NI-RX AXIS-FIFO which is set to 8K locations (32 KiB) for (8K $\times$ 32-bit) data flits. In case the data is larger than 32 KiB, the sending tile has to wait for an acknowledgement (ack) signal from the receiving tile that the previous 8K data flits have been stored on the shared data memory and it is permitted to start sending another block of (8K $\times$ 32-bit) data flits. Accordingly, the proposed communication model guarantees no data loss during the transmission process. Table 2 gives a detailed description of used communication model functions for data sending and receiving between compute tiles. Based on the proposed many-core architecture, two sending/receiving cases are applicable. The first case is sending/receiving by a 32-bit compute tile. In this case, 32-bit data are loaded from sender shared data memory to the NI and then

TABLE 2. Communication model functions for sending and receiving data between compute tiles.

void send_data (uint_32t dest_tile, uint_32t data_size, uint_32t packet_size, uint_32t *data_addr‡ , uint_32t *NI_TX_addr‡)	
dest_tile	Defines which tile the data shall be sent to, it is a 32-bit value (e.g 0x8000_0000)
data_size	Defines the size of data to be sent in term of number of packet (# of data packets)
packet_size	Defines the packet size in term of number of data flits. The packet size is a constant value of 32 flits, each flit is 32-bit
data_addr‡	Defines the source address of the data which will be sent from the compute tile shared memory.
NI_TX_addr‡	Defines the memory mapped address of the NI_TX of the compute tile, it is a constant value for each compute tile.
void receive_data (uint_32t data_size, uint_32t packet_size, uint_32t *data_addr‡ , uint_32t *NI_RX_addr‡)	
data_size	Defines the size of data to be received in term of number of packet (# of data packets)
packet_size	Defines the packet size in term of number of data flit. The packet size is a constant value of 32 flits, each flit is 32-bit
data_addr‡	Defines the destination address in the compute tile shared memory to store the received data.
NI_RX_addr‡	Defines the memory mapped address of the NI_RX of the compute tile, it is a constant value for each compute tile.

‡ Addresses data_type is uint_64t for 64-bit compute tiles.
For 64-bit tiles, 64-bit data width is split into 2x32-bit data flits to be transmitted over the 32-bit NoC.

transmitted over the 32-bit NoC architecture. The second case is sending/receiving by a 64-bit compute tile. In this case, a 64-bit data flit has to be split into 2 × 32-bit data flits by the corresponding PE before transmission to the NI. For receiving by a 64-bit compute tile, each received two data flits have to be concatenated again into one 64-bit data width before writing in shared data memory.

D. INTERNAL RUN-TIME RECONFIGURATION MANAGEMENT

AGILER architecture differentiates from other state-of-the-art RISC-V based many-core architectures by the ability to support different many-core configurations through the run-time adaptation feature. As shown in Figure 1 (c), a run-time reconfiguration manager unit (suitable for multi-core RISC-V based compute tile) is implemented based on a single RISC-V core approach [37] inside the main processing tile to control and manage the reconfiguration process through the DPR technique for Xilinx FPGA. The proposed many-core architecture consists of a static partition region and several reconfigurable partition (RP) regions to be reconfigured according to the selected many-core configuration. The static region hosts the main processing tile and the NoC architecture. While the reconfigurable regions host different configurations for 64-/32-bit compute tiles (e.g. number of PEs, memory type and size) through a set of compute tiles reconfigurable modules (RMs). All compute tiles RMs share unified interfaces to the NoC routers through NI with single domain clock and reset signals. The DPR process is conducted internally through internal access configuration port (ICAP) primitive to allow AGILER architecture to self-manage the configuration process without any external controlling peripherals (e.g. a PC through a JTAG) [37]. A block diagram for the proposed reconfiguration manager is shown in Figure 6. The proposed implementation provides a high data throughput rate to the FPGA configuration memory via the ICAP primitive. The reconfiguration manager has three interfaces as shown in Figure 6: one control interface connected to the main tile AXI interconnect to control the Xilinx DMA controller [38], a data interface for

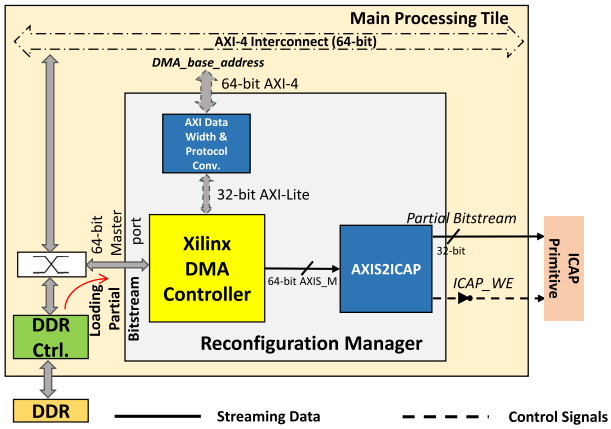


FIGURE 6. Run-time reconfiguration manager block diagram including a DMA controller and an internal ICAP controller for DPR process.

transferring partial bitstream from the external DDR to the ICAP primitive, and the ICAP primitive interface to the FPGA configuration memory. The DMA controller is connected to the main tile DDR controller through an additional crossbar as a master component to the DDR controller. Therefore, the DDR can be accessed either by PEs as a shared data memory or by the reconfiguration manager to load partial bitstream to FPGA configuration memory. Hence, partial bitstreams are stored in different address ranges than shared data memory address section of PEs. As the reconfiguration manager is implemented inside the 64-bit main tile, the DMA is configured to transfer a 64-bit data word from the DDR. Therefore, as mentioned in Figure 6, an AXIS2ICAP block is implemented to split a 64-bit data word into 2 × 32-bit data words to be compatible with the 32-bit data interface of the ICAP primitive. Besides, the valid stream signal is inverted and connected to the ICAP data port as the write enable signal (*ICAP_WE*). Also, an AXI data width and protocol converters are used to convert between the 64-bit AXI-interconnect and the 32-bit AXI-Lite control interface of Xilinx DMA component. The reconfiguration process takes place in two steps to successfully load a new partial bitstream with a new RM into the RP. All reconfiguration steps are

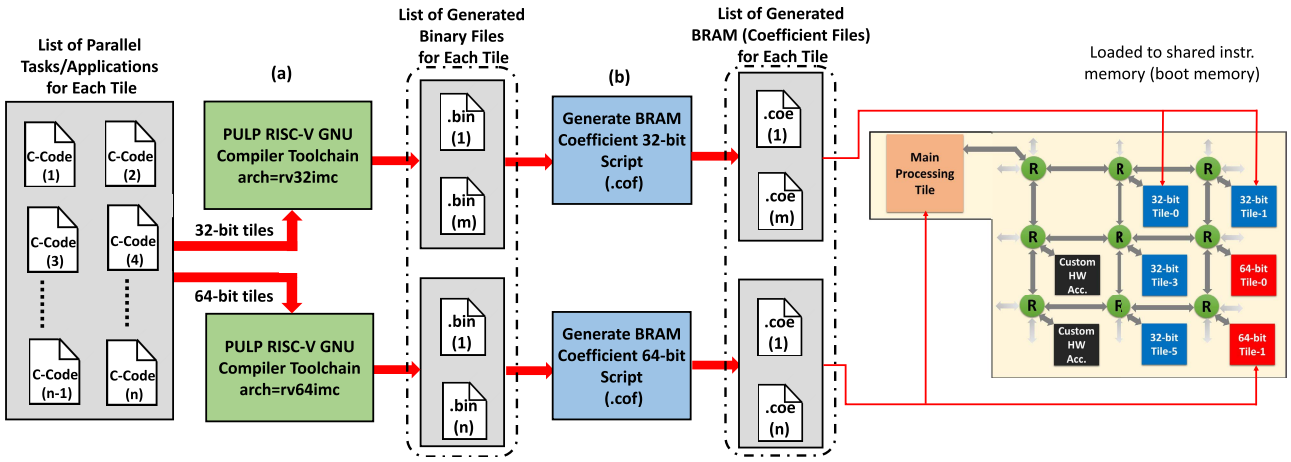


FIGURE 7. Schematic of many-core programming flow including: (a) building application tasks source codes targeting 32-/64-bit ISA, (b) generation of BRAM coefficient files to be stored on shared instruction memory (boot memory) of target compute tiles.

```
   //initialize the start address for RMs in the DDR and
        RMs location on the SD-card prior to
        reconfiguration
1  init_RModules(*reconfig_module, uint_64t RM_number,
                                    *pbit_fat_partition);
2  void init_reconfig_process() {
   // load a partial bitstream from the sd-card to
        rm_start address on the DDR
3   load_rm_sd (RM_number, pbit_fat_partition,
     reconfig_module->start_address);
    // start the reconfiguration process
4   reconfigure_RP(reconfig_module->start_address,
                   reconfig_module->pbit_size);
6  }
7  void reconfigure_RP (uint_64t *rm_address, uint_64t
     pbit_size){
    // starting Xilinx DMA
8   dma_start();
    // dma start reading from the DDR by setting the
         source address and length of RM partial bitstream
          on Xilinx DMA control registers
9   dma_write_stream(*rm_address, pbit_size);
10 }
```

Listing 1. Internal dynamic partial reconfiguration process executing on the 64-bit main processing tile.

managed by one 64-bit PE through a set of software functions as mentioned in Listing 1. The first step initializes RMs by reading the bit size (*pbit_size*) of their partial bitstream files stored on the external sd-card and loading them to defined destination addresses in the DDR memory. The first step is performed by the FAT32 I/O file system software modules running on one PE to load selected RMs partial bitstreams from FAT partitions on sd-card and store them on specified DDR destinations. Second, the reconfigure process step is executed by starting the DMA read channel to read the selected RM partial bitstream from the DDR and transfer it to the AXIS2ICAP block. After the DMA completes transferring the entire partial bitstream payload, a control signal is sent to the PE to indicate completion of the reconfiguration process, and the RM module of the corresponding compute tile is ready to communicate with the rest of the many-core architecture.

E. PROGRAMMING METHOD AND APPLICATIONS EXECUTION

In this work, a bare-metal parallel programming method is developed for the proposed many-core architecture to generate multiple binary files from multi-tasks applications to be executed on many-core compute tiles. Each compute tile executes a separate binary file for its mapped task, for this work we consider static task mapping that is done by the programmer prior to application execution. The PULP-RISC-V GNU toolchain [39] is used to compile C source codes for RV32/RV64 compute tiles architecture. As shown in Figure 7, a list of generated binary files (.bin) for 32-/64-bit tiles are the output of the compilation process, each compute tile has a single and separate binary file that will be executed on the shared instruction memory (boot memory) of its corresponding tile. Afterwards, the generated (.bin) files are converted to verilog memory files that contain the set of instructions for each tile. Then BRAM coefficient files (.coe) are generated to be loaded on shared instruction BRAM block for each tile as shown in Figure 7 (b). The BRAM coefficient files can be loaded to shared instruction memory during design tile prior to synthesizing process or after the generation of bitstreams by using update memory tool from Xilinx to only update BRAM contents of generated bitstreams.

In-order to execute software kernels using local memory of each PE inside a compute tile a memory initialization stage is required to load local instructions to the ITCM and local data initialization to DTCM for each PE as shown in Figure 8 (a). Therefore, a linker script (.ld) is developed as shown in Figure 8 (b) for instruction/data memory mapping. The linker script defines memory partitions and sectors for PEs shared and local instructions/data memory for a single compute tile. Initially and before application execution, all shared and local instructions are stored on shared instruction memory as shown in Figure 8 (a). Also, shared and local data initial values are stored on shared data memory. Therefore, during the memory initialization, each PE starts to load its

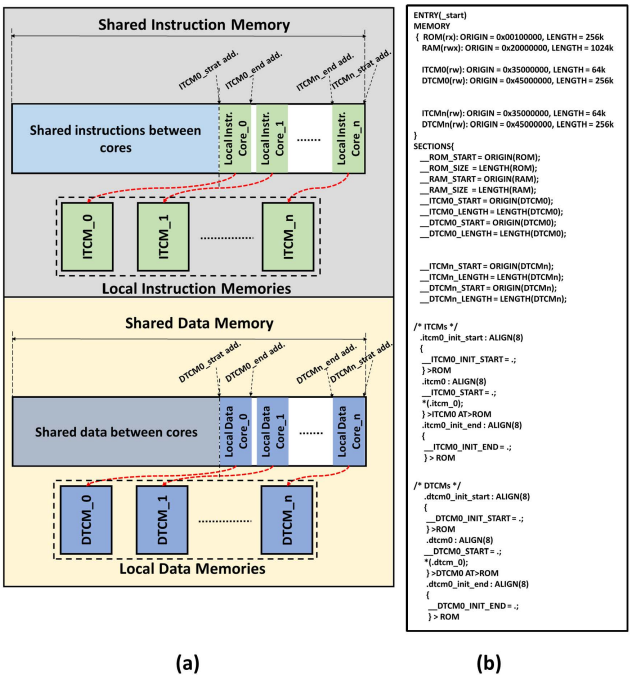


FIGURE 8. Memory mapping and sectors for a single compute tile: (a) instruction and data memory mapping for compute tile shared and local memories, (b) the corresponding linker script file (.ld) defines the different memory sectors for software building and compilation.

```
1void __main(){//exec. from the shared instr.mem on all
    cores/PEs
2  uint_32t core_id = read_csr(0xF14u)
3  if(core_id = x){ // copy local instr./data from
4                   // shared mem. to core_x local mem.
       // transfer data to local data mem.
5      uint_32t *dtcm_x = (uint_32t*)&_DTCM_x_START;
6      uint_32t *dtcm_x_init = (uint_32t*)&
           _DTCM_x_INIT_START;
7      for(uint_32t i = 0; i <= &__DTCMx_INIT_END - &
           __DTCM_x_INIT_START; ++i){
8          dtcmx[i] = dtcmx_init;
9      }
       // transfer instr. to local instr. mem
10     uint_32t *itcm_x = (uint_32t*)&_ITCM_x_START;
11     uint_32t *itcm_x_init = (uint_32t*)&
           _ITCM_x_INIT_START;
12     for(uint_32t i = 0; i <= &__ITCMx_INIT_END - &
           __ITCM_x_INIT_START; ++i){
13         itcmx[i] = itcmx_init;
14     }
15 }
16 if (core_id = y) {// copy local instr./data from
                      // shared mem. to core_y local mem.
                     }
17 // _DTCM_x_START, _DTCM_x_INIT_START, _ITCM_x_START,
   //  _ITCM_x_INIT_START, __DTCMx_INIT_END,
       __ITCMx_INIT_END
   // are defined in the linker script (Figure 8 (b))
18}// addresses data_type is uint_64t for 64-bit tiles
```

Listing 2. Memory initialization for each compute tile prior to kernels/tasks execution.

local instructions and initial data values from shared instruction/data memory to I/D-TCM based on memory address ranges defined on linker script for each PE. Listing 2 shows a snippet C code of memory initialization process for a single tile. In the C code kernel each function which has to be executed from a local ITCM has to be preceded with a memory section attribute (__attribute__(section(".itcm_0"))) which defines its executable ITCM for a specific PE. Similarly, local data variables have to be preceded with memory section attribute (__attribute__(section(".dtcm_0"))) which defines its executable DTCM for a specific PE.

IV. HARDWARE AND PERFORMANCE EVALUATION

Physical hardware implementation, system scalability, run-time reconfiguration and performance analysis results for the proposed AGILER architecture are discussed and presented in this section. Xilinx Virtex Ultrascale+ XCVU9P is the target FPGA for implementation and prototyping of the proposed tile-based many-core architecture. Also, Vivado Design Suite HLx 2019.1 is used for RTL synthesis, simulation, place and routing, and full and partial bitstream generation.

In this section, AGILER architecture is evaluated based on:

1) Hardware resources utilization and power consumption for different compute tiles and heterogeneous PEs described in Section III.
2) Run-time reconfiguration based on hardware resource utilization of reconfigurable partitions and reconfiguration time for different compute tiles configurations.
3) System scalability and computing performance with respect to different number and types of compute tiles in terms of achievable memory bandwidth, inter-tile data transfer latency, and computing operations per second (Op/s).

Benchmarks and test cases used for evaluation are written as software kernels over corresponding compute tiles using C programming language and compiled using the PULP-RISC-V GNU toolchain [36] as described in the previous section to generate the corresponding binary (.bin) files and coefficient files (.coe) to be loaded into the shared instruction memory of each compute tile. The execution cycles used in this section are measured by the performance counter register (PCCR) of RV32/RV64 cores. The number of cycles measured by the PCCR can be read using read_csr assembly function called in the corresponding benchmark or test case kernels and stored back in the compute tile shared data memory. As shown in Figure 1 (Section III), sd-card and UART peripherals are only accessed by the main processing tile which manages external data transfer between compute tiles and external peripherals. Therefore, for purposes of testing and evaluation each compute tile transmits evaluation results to the main processing tile to be transmitted to UART peripheral.

A. HARDWARE EVALUATION

AGILER architecture has been developed and implemented using a modular and hierarchical design approach. Where architectural modules (i.e. RISC-V cores, PEs, memory blocks, interconnects, etc.) are implemented as intellectual property (IP) components to be integrated together to

TABLE 3. Hardware resource utilization and power consumption of different compute tiles for the proposed many-core architecture targeting a Xilinx Virtex Ultrascale+ (XCVU9P) FPGA.

Compute Tiles and Modules			Resource Utilization				Power*
			LUT	BRAM	URAM	DSP	
RV32 Tile (4-PEs)	Total (4-PEs)		30717 **(2.6%)**	36 **(1.6%)**	12 **(1.25%)**	24 **(0.35%)**	**0.562 W**
	Shared Instr. Memory (64 KiB)		87	16	0	0	
	Shared Data Memory (256 KiB)		334	0	8	0	
	AXI-Bus		1033	0	0	0	
	RV32 PE	RI5CY Core	6902 (0.58 %)	0	0	6 (∼ 0.09%)	
		Local ITCM (4 KiB)	49	1	0	0	
		Local DTCM (16 KiB)	134	0	1	0	
	NI-RX**		563	15	0	0	
	NI-TX**		525	1	0	0	
RV64 Tile (1-PE)	Total (1-PE)		46311 **(3.9%)**	76 **(3.5%)**	8 **(0.83%)**	27 **(0.4%)**	**0.819 W**
	Instr. Memory (64 KiB)		87	16	0	0	
	Data Memory (256 KiB)		334	0	8	0	
	AXI-Bus		2893	0	0		
	ARIANE Core		39693 (3.35%)	44 (∼ 2%)	0	27 (0.4 %)	
	NI-RX		1647	15	0	0	
	NI-TX		1609	1	0	0	
RV64 Tile (2-PEs)	Total (2-PEs)		95636 **(8%)**	168 **(7.7%)**	8 **(0.83%)**	54 **(0.79%)**	**1.423 W**
	Shared Instr. Memory (64 KiB)		87	16	0	0	
	Shared Data Memory (256 KiB)		334	0	8	0	
	AXI-Bus		5411	0	0	0	
	RV64 PE	ARIANE Core	39693 (3.35%)	44 (∼ 2%)	0	27 (0.4 %)	
		Local ITCM (32 KiB)	335	8	0	0	
		Local DTCM (64 KiB)	367	16	0	0	
	NI-RX		1575	15	0	0	
	NI-TX		1644	1	0	0	
Main Processing Tile (4-PEs)	Total (4-PEs)		218773 **(18.5%)**	348 **(16.1%)**	0 **(0%)**	114 **(1.6%)**	**5.881 W**
	Shared Instr. Memory (64 KiB)		95	16	0	0	
	AXI-Bus		6428	0	0	0	
	RV64 PE	ARIANE Core	39693 (3.35%)	44 (∼ 2%)	0	27 (0.4 %)	
		Local ITCM (32 KiB)	335	8	0	0	
		Local DTCM (64 KiB)	367	16	0	0	
	NI-RX		1575	15	0	0	
	NI-TX		1644	1	0	0	
	Reconfiguration Manager		2317	6	0	0	
	DDR Controller		12365	26	0	6	
NoC (2x7) Configuration			70777 **(6%)**	0 **(0%)**	0 **(0%)**	0 **(0%)**	**1.9 W**
NoC (2x4) Configuration			45154 **(3.8%)**	0 **(0%)**	0 **(0%)**	0 **(0%)**	**0.8 W**

*Estimated dynamic power consumption for a complete compute tile including PEs, memory, and peripherals.
**NI for 32-bit tiles does not include AXI data width converter.

build heterogeneous compute tile modules for several many-core configurations. Inter-tile communication is implemented using ARTNoC framework [27] to generate multiple 2-D mesh NoC dimensions based on selected many-core configurations. The NoC module is implemented as a single parameterized module including routers and network links. The NoC is configured only during design-time based on 2-D mesh size, and number of flits per packet. In addition, dual-ported BRAM/URAM blocks used for instruction and shared memory inside compute tiles are implemented using Xilinx BRAM/URAM memory generator blocks with AXI-BRAM controllers. All many-core configurations and their required modules are synthesized and implemented targeting Xilinx Virtex Ultrascale+ XCVU9P FPGA. The proposed tile-based architecture is running using a single clock domain of 120 MHz for all implemented components and modules.

Table 3 shows the hardware resource utilization for different heterogeneous compute tiles used for the proposed tile-based many-core architecture as depicted in Figure 1. Three compute tile and a main processing tile modules are implemented to support different many-core configurations with two NoC configurations. All compute tiles are configured

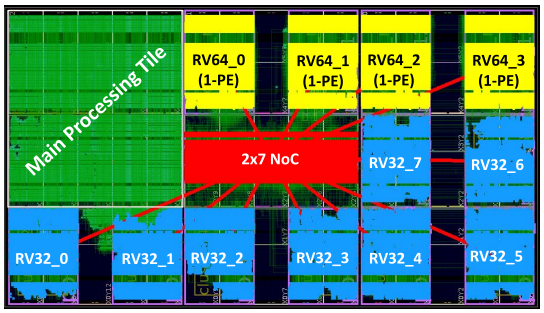
during design-time with a 64 KiB shared instruction memory, and 256 KiB shared data memory except the main processing tile which uses external DDR as shared data memory. As shown in Table 3, the RV32 tile consists of four PEs with 4KiB ITCM and 16 KiB DTCM. Each 32-bit PE consumes (∼0.6%) of total LUT, the 32-bit tile consumes (2.6%) of LUT mostly consumed by the 4 PEs. On-chip memory usage is distributed between BRAM and URAM blocks with a total percentage utilization of (∼3%) for a complete tile. As shared and local data memories are implemented using URAM blocks while all instruction memories are implemented using BRAM blocks for balanced on-chip memory utilization. Similarly, RV64 tiles (w/single-PE, and w/2-PEs) consume the same on-chip memory resources as the RV32 tile. In contrast, one 64-bit PE is 6x the size of a 32-bit PE in terms of resource utilization. Therefore, the maximum number of PEs per 64-bit tile is two to keep resource utilization under a certain limit for RV64 tiles. As shown in Table 3, the RV64 (w/single-PE) tile is 1.5x and the RV64 tile with two PEs is 3x the size of the RV32 tile respectively. On the other hand, the main processing tile consumes (18.5%) of total LUT and (∼16%) of on-chip memory as it is configured with four 64-bit PE. However, the main processing tile can be configured with less PEs in order to reduce resource utilization for smaller FPGAs or more design space. In addition, all compute tiles are equipped with two NI channels for transmitting and receiving over the NoC, for NI-RX 15 BRAM blocks are used to implement the (8K×32-bit) AXIS-FIFO and single BRAM block for NI-TX AXIS-FIFO. Two NoC configurations are realized in this work for two different many-core sizes, the first one support the communication up to 14 compute tiles including one main processing tile with total resource utilization of (6%) of total LUT. The second NoC configuration consumes less resource utilization for smaller many-core size up to 8 compute tiles. Furthermore, the power consumption of each type of compute tile and NoC configurations are estimated by Vivado power estimation tool at a clock frequency of 120 MHz as shown in Table 3.

B. RUN-TIME RECONFIGURATION

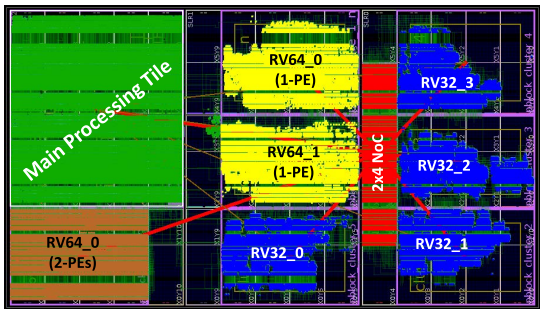
Xilinx dynamic partial reconfiguration (DPR) technique is supported to change many-core configurations during runtime without the need to synthesize or implement the complete architecture every time a new configuration is applied. The generation of partial bitstreams for many-core tile configurations is conducted through Xilinx partial reconfiguration (PR) design flow [40]. AGILER architecture is equipped with an internal reconfiguration manager located inside the main processing tile to manage and control the reconfiguration process through the FPGA ICAP primitive. For evaluation, two NoC configurations are used to support several many-core sizes regarding the number and types of compute tiles. As mentioned in the previous section, the main processing tile and NoC are hosted by the static partition region of the architecture, while the other three types of compute tiles are swapped over reconfigurable partition regions during runtime. Figure 9 (a, b) show two FPGA floorplans based on two different NoC configurations: first configuration with 2 × 7 NoC, and second configuration with 2 × 4 NoC. The first many-core size shown in Figure 9 (a) consists of 12 RPs that can host two types of compute tiles (RV32, and RV64 (w/single-PE) tiles) as RMs. All 12 RPs have the same size on the FPGA floorplan with the same number and types of hardware resources. A single RP size is specified to host the largest compute tile used in the first many-core size. Table 4 shows hardware resource utilization for a single RP and percentage resource utilization for each RM from the total RP size. The maximum utilization percentage is achieved by the RV64 (w/single-PE) tile with (87.5%) LUT and (91.6%) on-chip memory utilization. On the other hand, the second many-core size with 2 × 4 NoC supports larger RP size to support the largest configurable compute tile module (RV64 (w/2-PE) tile) for AGILER architecture. Figure 9 (b) shows the FPGA floorplan of the second many-core size with 7 RPs and three types of RMs for all compute tiles. However, the number of compute tiles is limited to 7 tiles to fit with the target FPGA total size. Similar to first many-core, all RPs have the same size that can fit with the largest compute tile which is in this case the RV64 tile with two PEs. Table 4 shows resource utilization for second many-core and its RMs. The RP size is double the size used for first many-core with a maximum utilization of (92.2%) for LUT and (87.5%) for on-chip memory hosting RV64 (w/2-PEs) tile modules. In contrast, RV32 tile modules consume less than (30%) of RP resources. Therefore, the second many-core can be used efficiently to support more RV64 tiles in comparison with the first many-core.

Moreover, Table 5 shows total resources utilization for the two many-core sizes shown in Figure 9. The resource utilization reports the total hardware resources required by all used RPs with the main processing tile for the two many-core sizes. Accordingly, the first many-core is more suitable for heterogeneous configurations that support more RV32 tiles than RV64 tiles to increase the efficiency of resources usability. On the other hand, the second many-core is more suitable for RV64 based configurations. In addition to that, several many-core sizes can be realized by using different NoC sizes to support different number of homogeneous or heterogeneous compute tiles based on target application requirements.

Furthermore, reconfiguration time is an important aspect to evaluate the performance of the internal reconfiguration manager (presented in Section III). In this work, the reconfiguration time is measured through the RISC-V core PCCR to measure the required number of clock cycles from loading a partial bitstream from the DDR memory until fully transferring the partial bitstream to the ICAP primitive. The reconfiguration time includes all software and hardware overhead required by the reconfiguration manager to successfully loaded one partial bitstream to the FPGA configuration memory through the ICAP. As shown in Table 4 (last column),



(a) First many-core size with 2x7 NoC configured by 8xRV32 (w/4-PEs), and 4xRV64 (w/single-PE) tiles.



(b) Second many-core size with 2x4 NoC configured by 4xRV32 (w/4-PEs), 2xRV64 (w/single-PE),,and 1xRV64 (w/2-PEs) tiles.

FIGURE 9. FPGA floorplan and resources utilization percentage for two different many-core sizes and configurations targeting a Xilinx Virtex Ultrascale+ (XCVU9P) FPGA.

TABLE 4. DPR resource utilization and reconfiguration time for two many-core sizes.

Many-core Size			Resource Utilization				Reconfiguration Time**
			LUT	BRAM	URAM	DSP	
First Many-Core Size (12 RPs)	Single RP Size (single Tile)		52880	96	32	384	18.8 ms
	RMs*	RV32-Tile (4-PEs)	30717 (58%)	35 (36.5%)	12 (37.5%)	24 (6.25%)	
		RV64-Tile (Single-PE)	46311 (87.5%)	88 (91.6%)	8 (0.25%)	27 (7%)	
Second Many-Core Size (7 RPs)	Single RP Size (Single Tile)		103680	192	64	768	38.1 ms
	RMs*	RV32-Tile (4-PEs)	30717 (29.6%)	35 (18.2%)	12 (18.7%)	24 (3.1%)	
		RV64-Tile (Single-PE)	46311 (44.6%)	88 (45.8%)	8 (12.5%)	27 (3.5%)	
		RV64-Tile (2-PEs)	95636 (92.2%)	168 (87.5%)	8 (12.5%)	54 (7%)	

*Percentage utilization of the total RP size.
**Reconfiguration time of a single partition for a single compute tile module.

TABLE 5. Total hardware resource utilization for the two different many-core sizes shown in Figure 9.

Many-Core Size	Resource Utilization			
	LUT	BRAM	URAM	DSP
First Size (2x7 NoC, 12 RPs)	853333 (72.1%)	1500 (69.5%)	384 (40%)	4722 (69 %)
Second Size (2x4 NoC, 7 RPs)	944533 (80%)	1692 (78.3%)	448 (46.7%)	5490 (80.3%)

Resource utilization is calculated based on total number of used RPs.

the total reconfiguration time of a single RP for first configuration is 18.8 ms. For second configuration, as the RP size increased, the required reconfiguration time is 38.1 ms. Accordingly, the proposed reconfiguration manager supports a high speed reconfiguration process as it uses a separate data stream channel to transfer partial bitstream to the ICAP primitive through a Xilinx DMA.

C. COMPUTING PERFORMANCE AND SCALABILITY

In this work, AGILER architecture is evaluated based on inter-tile data transfer latency, achievable memory bandwidth, and computing performance in terms of integer operations per second (Op/s). The evaluation is conducted over different numbers and types of compute tiles supported by the proposed many-core architecture. In order to measure inter-tile data transfer latency through the NoC. Two data transmission scenarios are evaluated for RV32 and RV64 tiles. Data transfer latency is measured from the transmitter tile (tile-TX) by measuring the time delay of transmitting a variant set of data sizes to the NI-RX of the receiving tile (tile-RX). The transmission time includes ① the time overhead of loading the data from tile-TX shared data memory to NI-TX using the communication model software function running on a PE inside tile-TX, and ② the time overhead of transmitting the data from source to destination routers over the NoC. Figure 10 shows the measured data transfer latency for both RV32 and RV64 tiles scenarios up to 32KiB of data size. Since RV64 tiles support 64-bit memory transfer from shared data memory to NI-TX compared to 32-bit memory transfer for RV32 tiles, the data transfer latency from the RV64 tile is ($\sim$ 2x) faster than the RV32 tile. The transmission time overhead over the NoC is similar for both RV32 and RV64 tiles as the NoC supports stream data transmission of 32-bit. Therefore, transmission time between NoC source and destination routers is negligible compared to the time overhead required to load the data from shared data memory to the NI-TX inside the transmitting tile.

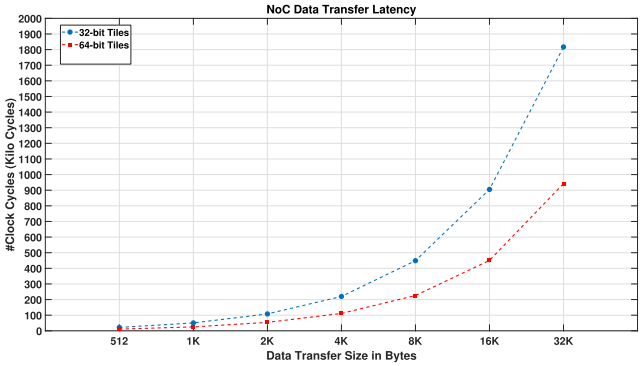


FIGURE 10. Data transfer latency over NoC between heterogeneous 32-/64-bit compute tiles (lower is better).

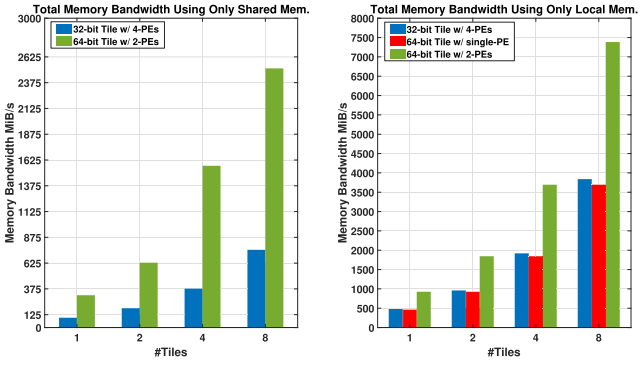


FIGURE 11. Achievable memory bandwidth with respect to numbers and types of many-core computing tiles using shared or local data memories (higher is better).

The memory bandwidth is measured by a parallel execution of a copy function on all PEs inside a tile to copy a data size of 4 KiB through two evaluation scenarios ① by using only shared data memory for multi-core supported compute tiles, ② by using local data memory (DTCM) for all type of compute tiles. Figure 11 shows the memory bandwidth scalability up to 8 compute tiles using different types of tiles for shared and local memory scenarios. Memory bandwidth is approximately proportionally scalable with the increasing number of tiles for all types of supported compute tiles. For shared memory scenario as shown in Figure 11, using RV64 tiles (2-PEs per tile) achieve higher memory bandwidth (∼3.5x) compared to RV32 tiles (4-PEs per tile). Therefore, supporting 64-bit memory transfer over 64-bit AXI interconnect for RV64 tiles provides more memory bandwidth per tile and thus the total many-core configuration. Moreover, for RV32 tiles memory bandwidth scalability is not increased proportionally by increasing number of PEs due to the traffic contention through the AXI-interconnect. As a result, for shared memory scenario, using 8xRV64 (w/2-PEs) tiles achieves a maximum memory bandwidth of (∼2.5 GB/s) while 8xRV32 (w/4-PEs) achieve (∼0.75 GB/s). On the other hand, for local memory scenario as shown in Figure 11, the overall memory bandwidth scalability for all compute tiles is improved by (∼4x) for RV32 tiles and by (∼2.5x) for RV64 (w/2-PEs) tiles compared to shared memory scenario. Moreover, the memory bandwidth achieved by using RV64 (w/single-PE) tiles is approximately the same achieved by RV32 (w/4-PEs) tiles. Therefore, memory access latency is less in RV64 tiles compared to RV32 tiles due to fewer interconnects and data bridges usage inside 64-bit PEs between RISC-V core and DTCM. Consequently, RV64 (w/2-PEs) tiles achieve (∼2x) memory bandwidth compared to RV64 (w/single-PE) tiles. As a result, the maximum achievable memory bandwidth using 8xRV64 (w/2-PEs) tiles is (7.4 GB/s) and (3.8 GB/s) for 8xRV32 tiles.

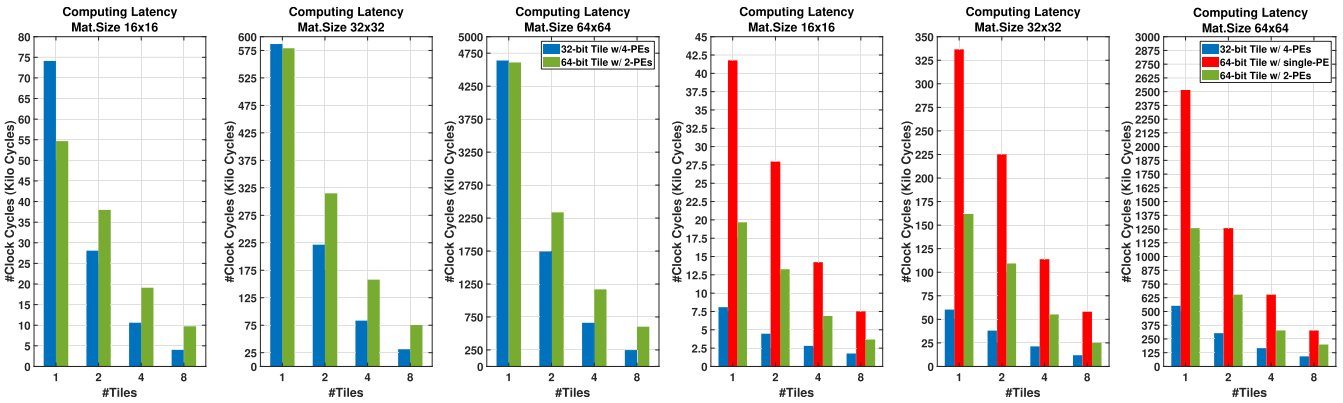
A parallel block matrix multiplication benchmark is implemented in order to evaluate the computing performance of different compute tiles for several many-core configurations.

TABLE 6. Computing performance for different numbers and types of compute tiles using matrix multiplication benchmark.

Number and Types of Tiles		Performance	
		Only Shared Data Mem.	Only Local Data Mem. (DTCM)
1-Tile	RV32 (4-PEs)	14 MOPS	114 MOPS
	RV64 (1-PE)	-	25 MOPS
	RV64 (2-PEs)	14 MOPS	50 MOPS
2-Tiles	RV32 (4-PEs)	36 MOPS	207 MOPS
	RV64 (1-PE)	-	52 MOPS
	RV64 (2-PEs)	27 MOPS	97 MOPS
4-Tiles	RV32 (4-PEs)	96 MOPS	377 MOPS
	RV64 (1-PE)	-	98 MOPS
	RV64 (2-PEs)	54 MOPS	193 MOPS
8-Tiles	RV32 (4-PEs)	254 MOPS	685 MOPS
	RV64 (1-PE)	-	192 MOPS
	RV64 (2-PEs)	105 MOPS	316 MOPS
Main Processing Tile	RV64 (4-PEs)	25 MOPS	96 MOPS

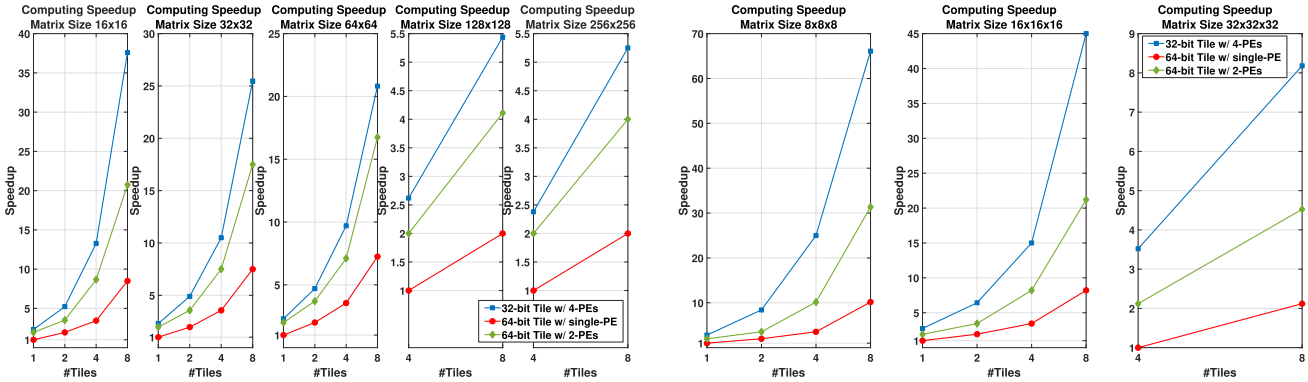
- Performance for RV32 tiles is measured by 32-bit integer Op/s, for RV64 tiles is 64-bit integer Op/s.
- Local instruction memories (ITCMs) are used for all scenarios.

Matrix multiplication benchmark is based on square matrix multiplication dimension for equal matrices partitioning for parallel execution over binary numbers of compute tiles. The parallel block matrix algorithm is used to partition matrix A into sub-matrices equals to the number of compute tiles. While matrix B is partitioned into sub-matrices equals to the number of PEs per compute tile. Each PE inside a compute tile computes the multiplication of a sub-matrix A with a sub-matrix B and store the result in a sub-matrix C in shared data memory. 32-/64-bit integer matrix multiplication



(a) Using only compute tiles shared memory (lower is better). (b) Using only compute tiles local memory (lower is better).

FIGURE 12. Computing latency of matrix multiplication benchmark over different numbers and types of compute tiles.



(a) 2-D convolution speedup, kernel size = (3x3), (higher is better). (b) 3-D convolution speedup, kernel size = (3x3x3), (higher is better).

FIGURE 13. Speedup of 2-D and 3-D convolution kernels for different input matrix sizes over different numbers and types of compute tiles.

algorithms are used over RV32 and RV64 tiles respectively. For evaluation, three many-core configurations are used with 8 compute tiles. Each configuration supports only one type of compute tile (RV32, RV64 (w/single-PE), and RV64 (w/2-PEs)). In addition, two evaluation scenarios are conducted using tiles shared data memory or local data memory to load and store matrices values. Figure 12 shows parallel matrix multiplication computing latency for the three many-core configurations using several types of compute tiles with three square matrix sizes. As a result, computing performance is proportionally scalable with increasing number of compute tiles for RV32 and RV64 tiles using shared or local data memory. As shown in Figure 12 (a), the computing performance of single RV32 tile is approximately the same as RV64 (w/2-PE) tile. Despite the higher number of PEs per RV32 tile which increases computing performance, the RV64 tile has a higher memory bandwidth compared to RV32 tile. Therefore, the required memory access time for loading and storing matrices values is less in case of RV64 tile which improves the overall computing performance with less number of PEs. However, by increasing the number of tiles, many-core configuration with RV32 tiles achieve less computing latency by (∼1.5x) compared to many-core configuration with RV64 tiles. On the other hand, using local data memory will improve the computing performance by (∼6x) for RV32 tiles and (∼3.5x) for RV64 tiles as shown in Figure 12 (b). In local memory scenario, computing performance depends only on number of PEs as each PE has its own data memory. Therefore, for all number of tiles, the RV32 tile achieves less computing latency by (∼5x) in comparison with RV64 (w/single-PEs). Table 6 shows the computing performance in term of number of Op/s based on the matrix multiplication benchmark for the aforementioned many-core configurations and evaluation scenarios. As a result, AGILER architecture achieves a maximum 32-bit computing performance of (685 MOPS) configured with 8xRV32 (w/4-PEs) tiles using only local data memory. For 64-bit integer operations, a maximum computing performance of (316 MOPS) is achieved by 8xRV64 (w/2-PEs) configuration. In addition, the main processing tile achieves a maximum performance of (96 MOPS).

Furthermore, 2-D and 3-D parallel convolution kernels are considered for evaluation as they are commonly used for signal processing and neural network algorithms.

TABLE 7. Comparison between state-of-the-art RISC-V based many-core architectures and the proposed architecture (AGILER) in term of resources utilization and computing performance targeting FPGA platforms.

Architecture (# RV Cores/PEs per Tile, ISA)		Resource Utilization per Tile				Freq. (MHz)	Power per Tile	Perf. (8-Tiles)	FPGA Device
		LUT	BRAM	URAM	DSP				
OpenPiton+Ariane [20] (Single-Core), RV64GC		90K	88	0	19	150	-	-	Virtex-Ultra.+ (XCVU9P)
Andromeda* [21] (4-Cores), RV64GC		131188	48	0	140	50	-	7.5 MOPS*	Synopsys HAPS (Virtex Ultra.)
Savas et al. [24] (Single-Core), RV64G		28241	258	0	15	113	-	-	Virtex-Ultra.+ (XCVU9P)
GRVIPhalanx [25] (8-Cores), RV32I		4800	12	0	0	150	0.34 W	-	Virtex-Ultra.+ (XCVU9P)
ESP [41] (Single-Core), RV64IMAC		50290	36	0	27	250	1.484 W	400 MOPS	Virtex-Ultra.+ (XCVU9P)
This Work *(AGILER)*	RV32-Tile (4-PEs), RV32IMC	30717	35	12	24	120	0.562 W	685 MOPS	Virtex-Ultra.+ (XCVU9P)
	RV64-Tile (Single-PE), RV64IMAC	46311	76	8	27	120	0.819 W	192 MOPS	
	RV64-Tile (2-PEs), RV64IMAC	95636	168	8	54	120	1.423 W	316 MOPS	

* Performance for Andromeda is reported by single precision floating point Op/s, the rest of architectures are measured by 32-/64-bit integer Op/s

Similarly to the matrix multiplication benchmark, parallel convolution kernels are evaluated over the aforementioned three many-core configurations based on RV32 and RV64 tiles. Also, 64-bit and 32-bit integer operations are supported by RV32 and RV64 tiles respectively. Parallel execution is conducted by partitioning the input matrix over the target number of compute tiles. In this experiment, input matrices have square dimensions to be partitioned equally over a binary number of compute tiles. Inside each compute tile the input matrix is partitioned again over the number of PEs per tile. 2-D and 3-D convolution are computed by sliding a kernel size of (3×3, and $3 \times 3 \times 3$) across the assigned input matrix for each PE. Loading and storing operations from/to the memory during convolution are conducted using local data memory for each PE to achieve maximum computing performance. Therefore, due to the limited size of on-chip memory larger sizes of input matrix require multiple compute tiles to be executed. Figure 13 shows the execution speedup of 2-D and 3-D convolution over different numbers and types of compute tiles. Speedup is measured based on the computing latency achieved by a specific many-core configuration in comparison with the computing latency of many-core configuration with RV64 (w/1-PE) tile for the same input matrix size. As a result, the many-core configuration with RV32 tiles achieves the highest speedup compared to other RV64 many-core configurations. As shown in Figure 13 (a), for 2-D convolution a maximum speedup of (∼37x) for (16×16) input matrix size is achieved by 8xRV32 tiles in comparison with a single RV64 (w/single-PE) tile. In case of (256×256) input matrix size, 8xRV32 configuration achieves a speedup of (∼5.5x) in comparison with 4xRV64 (w/single-PE) configuration. For 3-D convolution, as shown in Figure 13 (b), a speedup of (∼65x) is achieved by 8xRV32 configuration for ($8 \times 8 \times 8$) input matrix size. In case of a large input matrix size ($32 \times 32 \times 32$), a speedup of (∼8x) is achieved compared to 4xRV64 (w/single-PE) configuration.

D. STATE OF THE ART COMPARISON

In this work, the goal is to provide a heterogeneous and adaptable tile-based many-core architecture to support seamless integration and communication between multiple RISC-V ISAs for realizing several many-core configurations. Our main contributions rely on the modularity and run-time adaptability of compute tiles to support variant requirements for compute and memory-bound applications. The proposed architecture specifically targets FPGA devices for fast prototyping and evaluation which make it suitable for design space exploration for many application domains. Several RISC-V based many-core architectures are previously proposed in the literature and as open-source platforms. However, design modularity and run-time adaptability are not supported by existing state-of-the-art approaches. Besides, Table 7 shows a comparison between our proposed architecture and several state-of-the-art RISC-V-based many-core architectures targeting FPGA devices. Comparison aims to evaluate hardware specifications and computing performance of our proposed architecture with other state-of-the-art approaches. Proposed architectures by [20], [24], [41] are based on a single application class RISC-V core per tile supporting one ISA which increases the cost of scalability in terms of hardware resources required for interconnection for many-core realizations as well as high clock frequency to increase the compute performance of a single tile (in case of ESP [41]). In contrast, Andromeda architecture [21] combines several cores per tile adding more computing power to a single tile using lower clock frequency and reducing the cost of scalability. However, power consumption is the main bottleneck of using application class RISC-V processors in compute tiles, especially for high scalable many-core systems. Therefore, GRVIPhalanx [25] uses a simple RISC-V ISA (RV32I) to support tens of compute tiles with appropriate overall power consumption. However, computing performance is very limited as it only support RV32I. In comparison to them, our

proposed architecture supports different types of compute tiles with more computing capabilities suitable for several application requirements. The RV32 tile supports highly scalable many-core systems with less resource utilization and power consumption per tile. Also, RV64 tiles feature low resource utilization compared to [20], [21], [41] supporting local and shared memory hierarchies per tile.

V. CONCLUSION

This work proposes AGILER a novel adaptive tile-based many-core architecture for heterogeneous RISC-V processors suitable for FPGA platforms. The proposed architecture features a high degree of design scalability and regularity using heterogeneous RISC-V PEs for multi-/single-core compute tiles. AGILER is based on a modular and configurable set of heterogeneous compute tiles connected through a scalable NoC architecture. Each compute tile supports scratchpad shared and local memory systems. Moreover, the proposed architecture supports design-/run-time configurations to change numbers and types of compute tiles for several many-core configurations. For run-time reconfiguration, a high-speed reconfiguration manager is implemented to manage and control the DPR process internally from AGILER architecture. The proposed architecture aims to ease the development and realization of RISC-V based heterogeneous many-core architectures by reducing the design time and the non-recurrent engineering costs. AGILER architecture is evaluated based on hardware resource utilization and computing performance scalability through several many-core configurations and benchmarks. The results show a high degree of computing scalability using a scalable number of heterogeneous compute tiles.

As for the future work, it is planned to provide more heterogeneous compute tiles with RISC-V ISA extensions for floating point operations (e.g. RV32F, RV64D) and vector instructions to support more applications requirements. In addition, an auxiliary memory tile will be implemented to support off-chip memory access directly through the NoC to be accessed directly by all compute tiles without passing by the main processing tile to access the external memory.

REFERENCES

[1] X. Xu, Y. Ding, S. X. Hu, M. Niemier, J. Cong, Y. Hu, and Y. Shi, "Scaling for edge inference of deep neural networks," *Nature Electron.*, vol. 1, no. 4, pp. 216–222, 2018.

[2] (2017). *Arm DynamIQ*. Accessed: Apr. 15, 2022. [Online]. Available: https://www.arm.com/why-arm/technologies/dynamiq

[3] S. Haas, T. Seifert, B. Nöthen, S. Scholze, S. Höppner, A. Dixius, E. P. Adeva, T. Augustin, F. Pauls, S. Moriam, and M. Hasler, "A heterogeneous SDR MPSoC in 28 nm CMOS for low-latency wireless applications," in *Proc. 54th Annu. Design Autom. Conf.*, Jun. 2017, pp. 1–6.

[4] T. Norrie, N. Patil, D. H. Yoon, G. Kurian, S. Li, J. Laudon, C. Young, N. Jouppi, and D. Patterson, "The design process for Google's training chips: TPUv2 and TPUv3," *IEEE Micro*, vol. 41, no. 2, pp. 56–63, Mar./Apr. 2021.

[5] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[6] G. Fettweis, M. Hassler, R. Wittig, E. Matus, S. Damjancevic, S. Haas, F. Pauls, S. Nam, and N. Grigoryan, "A low-power scalable signal processing chip platform for 5G and Beyond–Kachel," in *Proc. 53rd Asilomar Conf. Signals, Syst., Comput.*, Nov. 2019, pp. 896–900.

[7] R. Airoldi, F. Garzia, T. Ahonen, and J. Nurmi, "Ninesilica: A homogeneous MPSoC approach for SDR platforms," in *Computing Platforms for Software-Defined Radio*. Cham, Switzerland: Springer, 2017, pp. 107–119.

[8] J. Henkel, A. Herkersdorf, L. Bauer, T. Wild, M. Hubner, R. K. Pujari, A. Grudnitsky, J. Heisswolf, A. Zaib, B. Vogel, V. Lari, and S. Kobbe, "Invasive manycore architectures," in *Proc. 17th Asia South Pacific Design Autom. Conf.*, Jan. 2012, pp. 193–200.

[9] B. Janssen, F. Schwiegelshohn, M. Koedam, F. Duhem, L. Masing, S. Werner, C. Huriaux, A. Courtay, E. Wheatley, K. Goossens, F. Lemonnier, P. Millet, J. Becker, O. Sentieys, and M. Hubner, "Designing applications for heterogeneous many-core architectures with the FlexTiles platform," in *Proc. Int. Conf. Embedded Comput. Syst., Archit., Modeling, Simulation (SAMOS)*, Jul. 2015, pp. 254–261.

[10] K. Asanovic, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz, and S. Karandikar, "The rocket chip generator," Dept. EECS, Univ. California, Berkeley, Berkeley, CA, USA, Tech. Rep., UCB/EECS2016-17, Apr. 2016. Accessed: Apr. 15, 2022. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html

[11] C. Heinz, Y. Lavan, J. Hofmann, and A. Koch, "A catalog and in-hardware evaluation of open-source drop-in compatible RISC-V softcore processors," in *Proc. Int. Conf. ReConFigurable Comput. FPGAs (ReConFig)*, Dec. 2019, pp. 1–8.

[12] A. Kamaleldin, S. Hesham, and D. Gohringer, "Towards a modular RISC-V based many-core architecture for FPGA accelerators," *IEEE Access*, vol. 8, pp. 148812–148826, 2020.

[13] S. Rheindt, T. Sabirov, O. Lenke, T. Wild, and A. Herkersdorf, "X-centric: A survey on compute-, memory- and application-centric computer architectures," in *Proc. Int. Symp. Memory Syst.*, Sep. 2020, pp. 178–193.

[14] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C. C. Miao, J. F. Brown, and A. Agarwal, "On-chip interconnection architecture of the tile processor," *IEEE Micro*, vol. 27, no. 5, pp. 15–31, Sep./Oct. 2007.

[15] L. Benini, E. Flamand, D. Fuin, and D. Melpignano, "p2012: Building an ecosystem for a scalable, modular and high-efficiency embedded computing accelerator," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2012, pp. 983–987.

[16] J. Ax, G. Sievers, J. Daberkow, M. Flasskamp, M. Vohrmann, T. Jungeblut, W. Kelly, M. Porrmann, and U. Ruckert, "CoreVA-MPSoC: A many-core architecture with tightly coupled shared and local data memories," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 5, pp. 1030–1043, May 2018.

[17] A. Amid, D. Biancolin, A. Gonzalez, D. Grubb, S. Karandikar, H. Liew, A. Magyar, H. Mao, A. Ou, N. Pemberton, P. Rigge, C. Schmidt, J. Wright, J. Zhao, Y. S. Shao, K. Asanovic, and B. Nikolic, "Chipyard: Integrated design, simulation, and implementation framework for custom SoCs," *IEEE Micro*, vol. 40, no. 4, pp. 10–21, Jul. 2020.

[18] M. V. Arunkumar and G. H. Hayatnagarkar, "RISKA: Towards an open-source RISC-V based domain-specific system-on-chip for SKA data processing," in *Proc. Workshop Comput. Archit. Res. With RISC-V (CARRV) Workshop (ISCA)*, 2021, pp. 1–7.

[19] J. Balkind, M. McKeown, Y. Fu, T. Nguyen, Y. Zhou, A. Lavrov, M. Shahrad, A. Fuchs, S. Payne, X. Liang, and M. Matl, "OpenPiton: An open source manycore research framework," in *Proc. ACM SIGPLAN*, 2016, pp. 217–232.

[20] J. Balkind, K. Lim, F. Gao, J. Tu, D. Wentzlaff, M. Schaffner, F. Zaruba, and L. Benini, "OpenPiton+ Ariane: The first open-source, SMP linux-booting RISC-V system scaling from one to many cores," in *Proc. Workshop Comput. Archit. Res. With RISC-V (CARRV)*, 2019, pp. 1–6.

[21] F. Merchant, D. Sisejkovic, L. M. Reimann, K. Yasotharan, T. Grass, and R. Leupers, "ANDROMEDA: An FPGA based RISC-V MPSoC exploration framework," in *Proc. 34th Int. Conf. VLSI Design 20th Int. Conf. Embedded Syst. (VLSID)*, Feb. 2021, pp. 270–275.

[22] M. Cavalcante, S. Riedel, A. Pullini, and L. Benini, "MemPool: A shared-L1 memory many-core cluster with a low-latency interconnect," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Feb. 2021, pp. 701–706.

[23] C. C. Heinz, J. Hofmann, J. Korinth, L. Sommer, L. Weber, and A. Koch, "The TaPaSCo open-source toolflow," *J. Signal Process. Syst.*, vol. 93, no. 5, pp. 545–563, 2021.

[24] S. Savas, Z. Ul-Abdin, and T. Nordström, "A framework to generate domain-specific manycore architectures from dataflow programs," *Microprocessors Microsystems*, vol. 72, Feb. 2020, Art. no. 102908.
[25] J. Gray, "GRVI phalanx: A massively parallel RISC-V FPGA accelerator accelerator," in *Proc. IEEE 24th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, May 2016, pp. 17–20.
[26] D. Petrisko, F. Gilani, M. Wyse, D. C. Jung, S. Davidson, P. Gao, C. Zhao, Z. Azad, S. Canakci, B. Veluri, and T. Guarino, "BlackParrot: An agile open-source RISC-V multicore for accelerator SoCs," *IEEE Micro*, vol. 40, no. 4, pp. 93–102, Jul./Aug. 2020.
[27] F. Zaruba, F. Schuiki, and L. Benini, "Manticore: A 4096-core RISC-V chiplet architecture for ultraefficient floating-point computing," *IEEE Micro*, vol. 41, no. 2, pp. 36–42, Mar./Apr. 2021.
[28] P. Mantovani, D. Giri, G. D. Guglielmo, L. Piccolboni, J. Zuckerman, E. G. Cota, M. Petracca, C. Pilato, and L. P. Carloni, "Agile SoC development with open ESP," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, Nov. 2020, pp. 1–9.
[29] S. Hesham, D. Gohringer, and M. A. El Ghany, "ARTNoCs: An evaluation framework for hardware architectures of real-time NoCs," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2016, pp. 259–264.
[30] A. Kamaleldin, M. Ali, P. A. Rad, M. Gottschalk, and D. Gohringer, "Modular memory system for RISC-V based MPSoCs on Xilinx FPGAs," in *Proc. IEEE 13th Int. Symp. Embedded Multicore/Many-Core Syst. Chip (MCSoC)*, Oct. 2019, pp. 68–73.
[31] A. Kurth, W. Ronninger, T. Benz, M. Cavalcante, F. Schuiki, F. Zaruba, and L. Benini, "An open-source platform for high-performance non-coherent on-chip communication," *IEEE Trans. Comput.*, early access, Aug. 25, 2021, doi: 10.1109/TC.2021.3107726.
[32] *AXI SystemVerilog Modules for High-Performance on-Chip Communication*. Accessed: Apr. 15, 2022. [Online]. Available: https://github.com/pulp-platform/axi
[33] A. Traber, M. Gautschi, and P. D. Schiavone. (Apr. 2019). *RI5CY: User Manual*. Accessed: Apr. 15, 2022. [Online]. Available: https://pulp-platform.org/docs/ri5cy_user_manual.pdf
[34] P. Davide Schiavone, F. Conti, D. Rossi, M. Gautschi, A. Pullini, E. Flamand, and L. Benini, "Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for internet-of-things applications," in *Proc. 27th Int. Symp. Power Timing Modeling, Optim. Simulation (PATMOS)*, Sep. 2017, pp. 1–8.
[35] F. Zaruba and L. Benini, "The cost of application-class processing: Energy and performance analysis of a linux-ready 1.7-GHz 64-bit RISC-V core in 22-nm FDSOI technology," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 11, pp. 2629–2640, Nov. 2019.
[36] *CVA6 RISC-V CPU*. Accessed: Mar. 13, 2022. [Online]. Available: https://github.com/openhwgroup/cva6
[37] N. Charaf, A. Kamaleldin, M. Thummler, and D. Gohringer, "RV-CAP: Enabling dynamic partial reconfiguration for FPGA-based RISC-V system-on-chip," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, Jun. 2021, pp. 172–179.
[38] Xilinx Inc. *AXI DMA v7.1*. Accessed: Apr. 15, 2022. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/axi_dma/v7_1/pg021_axi_dma.pdf
[39] *PULP RISC-V GNU Compiler Toolchain*. Accessed: Apr. 15, 2022. [Online]. Available: https://github.com/pulp-platform/pulp-riscv-gnutoolchain
[40] Xilinx Inc. (Jan. 2019). *Vivado Design Suite User Guide: Partial Reconfiguration*. Accessed: Apr. 15, 2022. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_1/ug909-vivado-partial-reconfiguration.pdf
[41] *ESP: The Open-Source SoC Platform*. Accessed: Apr. 15, 2022. [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. He is currently working as a Research Assistant at the Chair of Adaptive Dynamic Systems (ADS), Technische Universität Dresden (TU Dresden), Germany. His current research interests include reconfigurable computing, multiprocessor systems-on-chip (MPSoCs), networks-on-chip, hardware-software codesign, and runtime systems.

DIANA GÖHRINGER (Member, IEEE) is a Professor for Adaptive Dynamic Systems at TU Dresden, Germany. From 2013 to 2017, she was an Assistant Professor and the Head of the Application-Specific Multi-Core Architectures (MCA) Research Group at Ruhr-University Bochum (RUB), Germany. Before that, she was working as the Head of the Young Investigator Group Computer Aided Design and Exploration of Multi-Core Architectures (CADEMA) 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 the Ph.D. degree *(summa cum laude)* in electrical engineering and information technology from the Karlsruhe Institute of Technology (KIT), Germany. She is author and coauthor of over 150 publications in international journals, conferences, and workshops. Additionally, she serves as a technical program committee member in several international conferences and workshops. She is a reviewer and a 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.