Cross-Chip Partial Reconfiguration for the Initialisation of Modular and Scalable Heterogeneous Systems

Marvin Fuchs, Hendrik Krause, Timo Muscheid, Lukas Scheller, Luis E. Ardila-Perez, Oliver Sander

***Abstract*—The almost unlimited possibilities to customize the logic in an FPGA are one of the main reasons for the versatility of these devices. Partial reconfiguration (PR) exploits this capability even further by allowing to replace logic in predefined FPGA regions at runtime. This is especially relevant in heterogeneous SoCs, combining FPGA fabric with conventional processors on a single die. Tight integration and supporting frameworks like the FPGA subsystem in Linux facilitate use, for example, to dynamically load custom hardware accelerators.**

Heterogeneous SoCs from AMD Xilinx are widely used in data acquisition systems. Qubit control instruments, for instance, are often built on a single RFSoC. However, this restricts the number of analogue channels, which is why only a small number of qubits can be interfaced. To scale to hundreds or even thousands of qubits, it is a promising option to expand the resources of the RFSoC by connecting auxiliary FPGAs. The AXI Chip2Chip (C2C) cross-chip bus allows to easily connect the logic in the devices. However, partial reconfiguration on peripheral FPGAs utilising the same channel is not officially supported. To overcome this restriction, we propose a method using an AXI C2C connection in combination with an AXI ICAP controller and custom Linux drivers. As a result, FPGA firmware updates can be applied at runtime, and peripheral FPGAs can be added and removed during operation.

***Index Terms*—Partial Reconfiguration, Dynamic Function Exchange, DFX, AXI, MPSoC, System-on-Chip, Zynq UltraScale+**

I. Introduction

DATA acquisition (DAQ) systems for physics experiments must push the limits of the latest available technology to fully exploit the potential of novel detectors. At the same time, these systems must be adapted to meet the specific requirements of the particular experiment. Heterogeneous SoCs from AMD Xilinx are often a good platform to meet these requirements, because the included processors can handle complex control schemes and allow to seamlessly integrate the device into existing Ethernet networks, whereas the programmable logic (PL) allows to implement application-specific, fast real-time modules and custom interfaces. As the following examples show, these advantages are beneficial from small single-device systems up to large, distributed setups with multiple hundreds of SoCs and FPGAs.

Various custom electronics cards are being developed on the basis of the advanced telecommunications computing architecture (ATCA) specification for the next electronics upgrade of both general-purpose large-scale experiments ATLAS and CMS at CERN [1], [2], [3], [4], [5]. Most of them will use the versatility of heterogeneous multiprocessor system-on-chips (MPSoCs) from AMD Xilinx and combine them with one or more large FPGAs on an ATCA card to form a unit with more resources. However, to meet the requirements of the experiments, hundreds of these cards will be combined to form a massive DAQ system.

This contrasts, for instance, with systems to interface cryogenic sensors, based on a single MPSoC or RFSoC [6], [7], [8]. One such control and DAQ system is the so-called "QiController" developed at the Karlsruhe Institute of Technology (KIT). It is a platform primarily dedicated to the characterization of superconducting qubits. Currently, the QiController is based on a single RFSoC from AMD Xilinx with sixteen analogue-to-digital converter (ADC) and sixteen digital-to-analogue converter (DAC) channels. To extend the range of applications and increase the number of addressable qubits, the system is currently being expanded to a hardware platform utilising the ATCA form factor and composed of multiple RFSoCs that are interconnected to several peripheral FPGAs and to one another. The proposed architecture is depicted in Figure 1. In this configuration, the analogue channels of the RFSoCs are directly used to interface with the qubits, while the peripheral FPGAs are used to drive further DACs that increase the number of analogue channels and interface the qubit coupling devices. Depending on the qubit device architecture, it is expected to have about 4 couplers per qubit. The qubits typically operate between 4 to 8 GHz while the couplers require signals between DC and 1 GHz.

Expanding the single-chip-based QiController to a multi SoC and multi FPGA platform introduces a variety of new challenges, including the proper partitioning of the system into several electronic cards, a multi-device initialisation sequence that takes into account the specific requirements of each individual component, and an intuitive and easy-to-use method for updating the software and FPGA firmware on the system. Additionally, the ATCA backplane limits the number of connections between the individual cards to only four bi-directional lanes. One of these is used to connect the

Manuscript submitted April 8, 2024. This work was funded by the Federal Ministry of Education and Research (BMBF) within the framework programme "Quantum technologies – from basic research to market" (Project QSolid, Grant No. 13N16151 and Project QBriqs, Grant No. 13N15950). This research acknowledges the support by the Doctoral School *"Karlsruhe School of Elementary and Astroparticle Physics: Science and Technology"*.
M. Fuchs (corresponding author, email: marvin.fuchs at kit.edu), H. Krause, T. Muscheid, L. Scheller, L. E. Ardila-Perez and O. Sander are with the Institute for Data Processing and Electronics (IPE) of the Karlsruhe Institute of Technology, Hermann-von-Helmholtz-Platz 1, D-76344 Eggenstein-Leopoldshafen, Germany

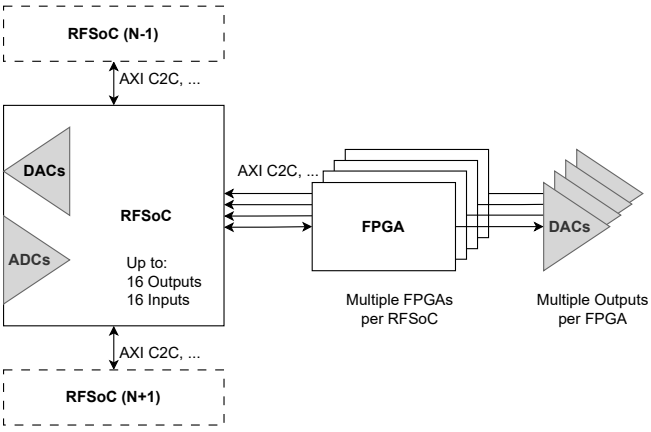

Fig. 1. Proposed modular and scalable multi-device QiController architecture.

PL of an RFSoC to an peripheral FPGA via AXI C2C in order to link the logic in both devices. We propose to use precisely this connection to perform PR, which allows the FPGA firmware on the peripheral FPGA to be initialised, updated, and dynamically exchanged from the processors in the RFSoC, efficiently using the available connections between devices. PR is a feature of FPGAs that enables exchanging the configuration of one part of the FPGA at runtime without affecting the operation of the remaining part.

AXI C2C is a method provided by AMD Xilinx to connect the internal AXI busses of two devices via gigabit transceivers (GTs). Using this feature is not only planned for the multi-device QiController; it is also already implemented in the aforementioned ATCA cards developed at CERN. Figure 2 shows how it can be used to extend the logic and interface resources of an MPSoC by connecting an FPGA. Since the connection is established between the PL and the FPGA, it does not make a difference for the processing system (PS) of the MPSoC if the resources are in the PL of the MPSoC itself or in the external FPGA. All resources are accessible from the same PS interface, and the connection is transparent.

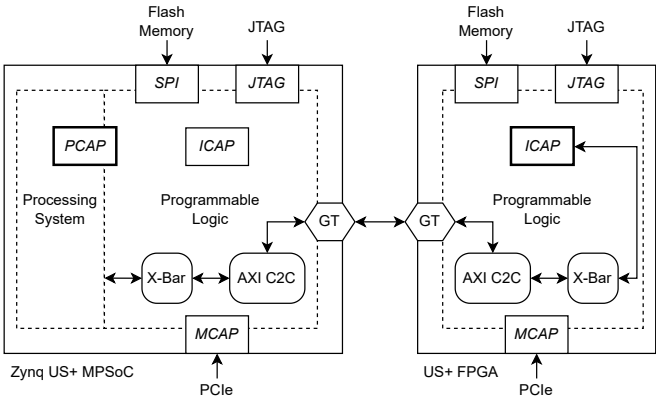

Fig. 2. PL extension of an MPSoC using a peripheral FPGA connected via AXI C2C. The most important configuration interfaces for both the MPSoC and for the FPGA are shown. It can be seen that without external loop backs, only the ICAP interface of the FPGA is accessible from the MPSoC via AXI C2C. The configuration interfaces that are to be used in the multi-device QiController are highlighted (PCAP on the MPSoC, ICAP on the peripheral FPGA).

Figure 2 also shows the most important configuration interfaces of the MPSoC's PL and of an FPGA from AMD Xilinx. It can be seen that the ICAP interface is the only one of the FPGA that is accessible from the PS of the MPSoC via AXI C2C and without additional connections between the devices or loopbacks on the FPGA. The ICAP interface is part of the FPGA and allows PR from within the fabric. It must always be combined with a suitable IP core that provides control logic. All FPGAs of the UltraScale and UltraScale+ family from AMD Xilinx offer this interface, as do all their MPSoC devices. According to the manufacturer, the ICAP interface is also the fastest option for PR [14]. This is especially relevant as the configuration interface is usually the most crucial element when it comes to rapid PR. The theoretical maximum throughput of the ICAP interface is 800 MB/s, which can also be achieved in practice under certain conditions and if the interface is overclocked from 100 MHz to 200 MHz [16], [18], [19]. This is much faster than other interfaces, such as the media configuration access port (MCAP) targeting Peripheral Component Interconnect Express (PCIe) with a bandwidth of typically 3 to 6 MB/s or the PCAP intended for configuring the PL from the PS in an MPSoC and achieving a maximum throughput of 256 MB/s [19]. However, these values refer purely to the performance of the hardware and the FPGA firmware and do not include any software overhead, such as control from an operating system.

II. Related Work

Since a few years, large heterogeneous systems utilising various computational engines, including FPGAs, central processing units (CPUs), graphics processing units (GPUs), and artificial intelligence (AI) specific processors, are frequently deployed in data centres of major providers like Alibaba, Amazon, Baidu, Huawei, and Microsoft [9]. This trend is motivated by the expectation that specialised processing units offer the biggest potential for hardware-based computational performance gains. Mainly for this reason, large heterogeneous systems with multiple FPGAs are the subject of current research [10], [11], [12]. The automatic and fast (re-)configuration of FPGAs is crucial, as this is a special requirement of these devices that other computational engines do not have, making it a new challenges for a data centre scenario. One example is that during (re-)configuration the FPGA cannot be used, which means a non-negligible dead time. Furthermore, unwanted loading of configuration data can lead to potential security risks. However, solutions for initialising and reconfiguring FPGAs in data centres, which are commonly based on PCIe [13], [15] or Ethernet [19], are tailored to the special requirements of that field and cannot be universally applied to DAQ systems. There are various reasons for this, most of which are a result of the different requirements made on the systems. Data centers aim to achieve best possible computational performance, whereas DAQ systems have real time requirements and must ensure integrity of the measurement data. This can also be seen in the fact that heterogeneous systems in data centres are usually administered with many

layers of software, allowing for automated management and steering of the multi-device setup process and data flow. In DAQ systems, it is typically preferred to avoid this level of overhead and uncertainty by instead using a leaner, customized solution. Nonetheless, there are mechanisms implemented in data centres that can be transferred to the DAQ domain, such as FPGA programming via PR as used by Amazon Web Services (AWS) [13]. This concept includes a small static framework that is automatically loaded on the FPGA at power-up to initialise various of its interfaces, which can later be used to load the payload logic using PR. The mechanism presented here also employs this method, but in combination with the AXI protocol, which is a frequently used option for intercommunication in heterogeneous system consisting of MPSoCs and FPGAs.

III. Partial Reconfiguration

During the design process, the FPGA is divided into a static region and one or more reconfigurable partitions (RPs). One or more modules can be implemented for each RP, which can later be exchanged dynamically at runtime. This corresponds to a time-division multiplexing of the hardware resources within an RP. To prevent signals from being unintentionally emitted or received by the RP during PR, the interfaces between the static design and the RPs should be disconnected in an orderly manner for the duration of the reconfiguration. This is typically accomplished using so-called decoupling logic.

Both major FPGA manufacturers Intel Altera and AMD Xilinx offer devices with PR features [21], [22]. However, the QiController is based on an RFSoC device from AMD Xilinx, so this contribution only targets their UltraScale+ family of FPGA and MPSoC devices. For the sake of clarity, the term PR is used throughout this document, even though AMD Xilinx calls this feature Dynamic Function eXchange (DFX).

The FPGA subsystem in Linux provides a vendor agnostic way for full and partial reconfiguration of FPGAs [23]. It comprises two kinds of Kernel drivers: FPGA managers and FPGA bridges. FPGA managers implement one specific method to configure an FPGA. To accomplish this, they control all required hardware and FPGA firmware. FPGA bridges control the decoupling logic to disconnect a RP from its surroundings. One FPGA manager and any number of FPGA bridges are grouped in FPGA regions, which embody either one RP or a full FPGA. FPGA managers, FPGA bridges, FPGA regions, and their dependencies must be declared in the Linux device tree, which is a standardised form of describing the hardware components of an embedded device or computer. The device tree is evaluated by the Kernel at boot time and later on to initialize, manage and use these components. Device tree overlays provide a method to patch the device tree at runtime. They can be used to notify the Kernel about newly connected hardware or to unregister hardware before it is physically removed. In the same way they can also be used to inform the Kernel that logic is loaded or unloaded from a RP or from the entire FPGA. Device tree overlays are even capable to actively load and unload logic when they are applied or removed.

IV. The Proposed Approach

In the architecture described in Figure 1 one RFSoC acts as the central point for user interaction and updates via the network interface. This is referred to as the central RFSoC. However, to optimise resource utilisation and maintain system modularity and expandability, each RFSoC is responsible for managing, initialising, and updating the FPGAs connected to it. As updates are merely reinitialisations that are carried out at runtime, and therefore do no differ from initialisations, the term initialisation is used for both operations in the following. The ATCA standard prescribes a sophisticated hardware platform management (HPM) infrastructure on every ATCA card which includes a low-bandwidth reconfiguration ability, however it is not suitable for fast real-time PR [24], [25]. The limited backplane connections prevent the implementation of a dedicated configuration connection for each associated FPGA board in the crate. Since the PCAP interface is solely available on MPSoCs, the only remaining option to enable the RFSoCs to manage the peripheral FPGAs is to access their ICAP interface through the AXI C2C connection. This eliminates the need for a dedicated physical configuration connection. If the reconfiguration capability is used primarily for initialisation and not for rapidly multiplexing the FPGA fabric, this is also a particularly efficient solution, as the AXI C2C connection is not required for other data transmission during initialisation. To enable the AXI C2C connection and access to the ICAP interface before initialisation, the FPGA automatically loads a small static design from a local flash memory at power-up. This static design can be kept very generic, which is why it is reasonable to assume that it will rarely need to be updated.

As shown in Figure 1, all peripheral FPGAs in our system serve the same purpose, which is to interface DACs and increase the number of available analogue channels. Therefore, the same configuration file should be used on all of them to reduce the building and maintenance effort and to simplify scaling the system. However, it is not easily possible to reuse the same file for every FPGA, as AXI is an address-based bus that requires a unique address per node. To use the same configuration file on all peripheral FPGAs, the AXI address space of the overall system can be subdivided. The most significant byte addresses an FPGA or the PL of an RFSoC, while the less significant four bytes are used for internal device addressing. This allows to truncate the device-specific part of the address before it is transferred via AXI C2C to a peripheral FPGA [26]. Thus, the RFSoCs have 40-bit addressing, while the FPGAs only have 32-bit addressing.

The initialisation of the entire system is controlled by the Linux operating system on the RFSoCs. For this purpose, the capabilities of the FPGA subsystem in Linux are used in combination with device tree overlays, as described in III. In a three-stage procedure, device tree overlays are first applied on each RFSoC independently to initialise the respective PL. After that, device tree overlays are loaded to reveal the RFSoCs to each other as well as the static part of the peripheral FPGAs to the respective Kernels on the RFSoCs. Finally, the RPs of the peripheral FPGAs are actively configured and introduced to the responsible Kernel with the third layer of

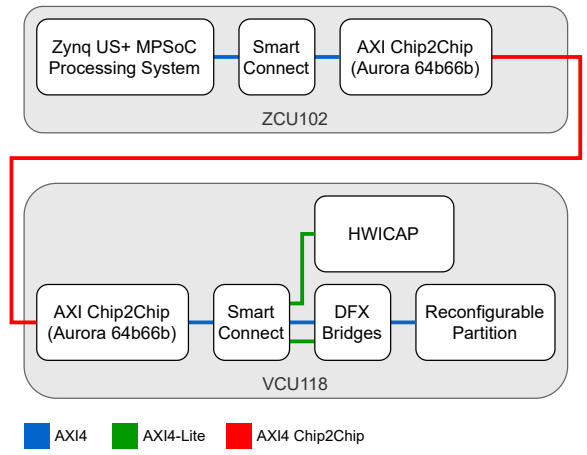

Fig. 3. Setup with one AMD Xilinx ZCU102 and one AMD Xilinx VCU118 to test cross-chip PR with a basic FPGA firmware architecture.

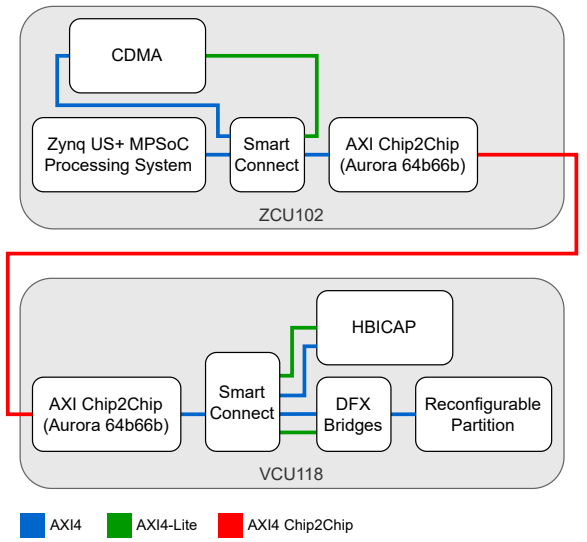

Fig. 4. Setup with one AMD Xilinx ZCU102 and one AMD Xilinx VCU118 to test the performance of cross-chip PR via a full AXI4 interface. The HBICAP IP core used features an AXI4-Lite control interface and an AXI4 data interface.

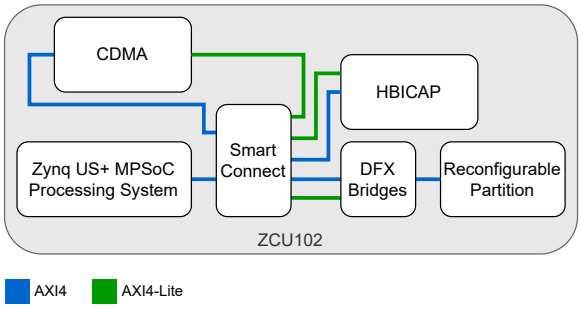

Fig. 5. Setup with one AMD Xilinx ZCU102 to test the performance of the combination of HBICAP and CDMA without AXI C2C.

device tree overlays. This procedure enables the system to be scaled by adding more RFSoCs and FPGAs at runtime. The concrete hardware implementation of the overall system, composed of several chips, can thereby be incorporated in FPGA manager and FPGA bridge drivers.

V. Evaluation Setups

Three different setups based on commercial evaluation cards were used to evaluate the functionality and performance of the approach. Each setup contains one AMD Xilinx ZCU102 to emulate an RFSoC of the architecture in Figure 1. Even though the ZCU102's MPSoC is missing ADCs and DACs, it is based on the same architecture as an RFSoC and offers all relevant features for PR. The peripheral FPGAs in Figure 1 are emulated with AMD Xilinx VCU118 evaluation cards. One lane of an QSFP to SFP+ cable is used to realise the AXI C2C connection between the boards.

To test the concept of PR via AXI C2C, the setup with one ZCU102 and one VCU118 shown in Figure 3 was used [26]. The FPGA on the VCU118 hosts one instance of the AXI hardware internal configuration access port (HWICAP) IP core to access the ICAP configuration interface of the FPGA and one instance each of an DFX Decoupler and an DFX AXI Shutdown Manager, which are FPGA bridges provided by the manufacturer. All of this connects to the PS of the ZynqMP on the ZCU102 via AXI C2C Bridge and Aurora 64B66B IP Cores on both devices. All IP cores used in this setup are provided by AMD Xilinx. To integrate the setup with the Linux FPGA subsystem, a custom FPGA Manager driver for the HWICAP was developed, based on the example character device driver for MicroBlaze [28].

The HWICAP IP core is a light weight and relatively easy to use way to access the ICAP configuration interface. However, it only provides an AXI4-Lite data interface for transferring configuration data, which offers lower performance compared to a full AXI4 interface. To access the ICAP interface with a full AXI4 connection, AMD Xilinx provides the more powerful and flexible AXI HBICAP IP core. Figure 4 shows the adapted architecture that was used to explore the advantages of this IP core. The architecture also comprises a CDMA IP core on the MPSoC to reduce processor load and exploit the full potential of the AXI4 interface. To control both the HBICAP and the CDMA IP core in this distributed configuration, a custom FPGA manager driver had to be developed. Finally, this setup was also tested with two VCU118 evaluation cards to better represent the architecture described in Figure 1.

To evaluate the impact of the AXI C2C connection on the PR performance, an equivalent setup without AXI C2C was created as well. The layout is shown in Figure 5. This setup is also more comparable to setups commonly used for the development and benchmarking of custom high-performance ICAP solutions such as ZyCAP [17]. This in turn makes it easier to compare the combination of HBICAP and CDMA with these high-performance solutions to assess whether they should also be tested in an AXI C2C configuration.

To support the use of the HBICAP and HWICAP IP cores in the community, we have released our implementation of the corresponding FPGA manager drivers to the public [29].

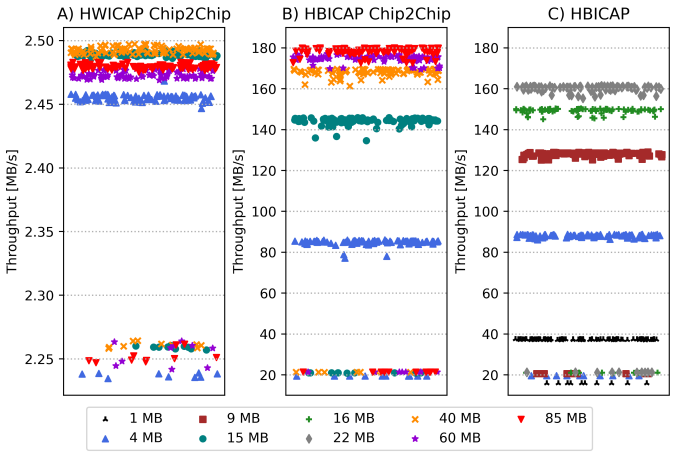

Fig. 6. PR throughput for a range of different RP sizes in the setup with: A) the HWICAP + AXI C2C (Figure 3 A), B) the setup with a HBICAP + AXI C2C (Figure 4 B), and C) in the setup with a HBICAP on a single MPSoC (Figure 5 C).

VI. Analysis

Several tests were conducted to quantify the suitability of the proposed approach for the initialisation of peripheral FPGAs in the multi-device QiController. All measurements were performed using a system clock of 100 MHz, and the ICAP interface was not overclocked. Additionally, the line rate for the AXI C2C connection was set to 10.3125 Gbps. CPU intensive startup tasks in the Linux userspace, such as the jitter-based initialisation of *rng-tools*, were disabled, and tests were automatically executed once the AXI C2C link was up, or, in the setup without AXI C2C, once Linux was fully booted. The reconfiguration time was determined in the FPGA fabric using a counter incremented with the system clock while the FPGA bridges were disconnected, measuring the time during which the RP was not usable due to reconfiguration. The DFX AXI Shutdown Manager initiated and stopped the counter. Each measurement was repeated ten times, separated by a one-second pause. After this, the system was hard rebooted and the entire procedure was repeated ten times, resulting in a total of 100 data points.

Figure 6 provides an overview of the acquired measurement data. The overall highest reconfiguration throughput at 178 MB/s was achieved with the largest RP and the AXI HBICAP. This corresponds to configuring 90% of the resources of an AMD Xilinx XCVU9P in 0.5 s and is 45% of the maximum throughput that can be achieved using the 32-bit AXI4 bus at a clock rate of 100 MHz. In contrast, the highest reconfiguration throughput with the AXI HWICAP is about two orders of magnitude lower at 2.5 MB/s. The three diagrams in Figure 6 show minor fluctuations in the measurements and that the reconfiguration speed clearly depends on the size of the RP. This is particularly visible in subfigures B and C, showing results for the HBICAP. One exception are the clusters with differently sized RPs at the lower end of all three subfigures. The clusters consist of measurements carried out immediately after power-up, indicating that in this case the reconfiguration throughput does not depend on the size of the RP. Figure 7 shows that all reconfigurations carried out directly after the start of the system are affected. But only these, from the second reconfiguration onwards, the throughput is significantly higher and approximately constant across all following reconfiguration runs. The effect could also not be observed if only the VCU118 was restarted while the ZCU102 waited in idle operation.

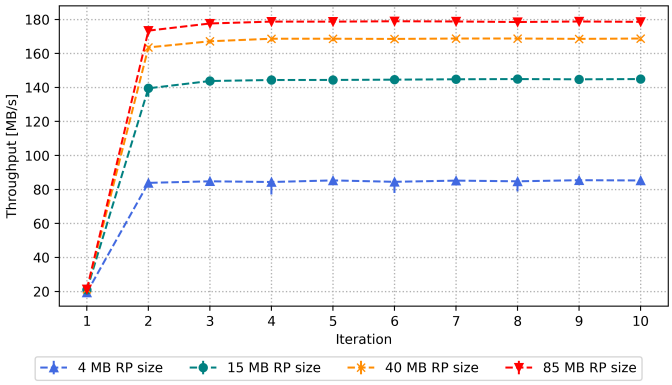

Fig. 7. Arithmetically averaged PR throughput curves over several consecutive measurements performed directly after system startup and for different RP sizes. The consecutive measurements are separated by a one-second pause. Deviations are indicated by error bars. The presented data shows the behaviour of the setup with a HBICAP + AXI C2C (Figure 4). However, the other two setups exhibit similar behaviour.

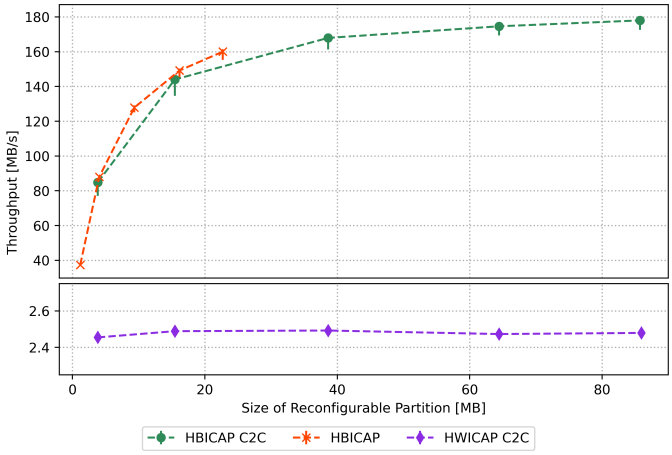

Fig. 8. Arithmetically averaged PR throughput in relation to the RP size. Deviations are indicated by error bars.

Figure 8 illustrates how RP size affects throughput during reconfiguration. This graph only contains values from the second reconfiguration onward to omit the above-mentioned effect of slow initial reconfiguration. The large gap of more than an order of magnitude between HBICAP and HWICAP based setups clearly shows that AXI4-Lite severely limits performance. This is mainly because AXI is a burst-based protocol and the maximum burst length for AXI4-Lite is limited to one data transfer per read and write request [30]. In contrast, a full AXI4 interface supports bursts of up to 256 data transfers per request. This difference has an even greater impact on performance when an AXI C2C connection is involved, since each request requires a handshake procedure,

which takes longer when two independent chips are involved. The measurement results of the HBICAP-based systems show that in this case the size of the RP has a significant influence on the throughput during reconfiguration. This observation is independent of whether AXI C2C is used or not. Thus, the setup with AXI C2C achieves even a higher maximum throughput than the single-board implementation on the ZCU102, because the FPGA on the VCU118 has significantly more resources than the PL on the ZCU102, allowing for a larger RP. In case of the ZCU102, more than 75% of all resources where assigned to one RP and with the VCU118 even more than 90% were possible without any issues. In case of the HWICAP based setup, there is no dependence of the reconfiguration throughput on the partition size.

In all measurement setups, the content of the RP and the order in which the configurations were applied had no effect on the reconfiguration speed. Therefore, this will not be discussed further.

VII. Results and Discussion

This section will discuss several inferences drawn from the preceding three diagrams. Figure 6 and Figure 8 show that reconfiguration with the HWICAP IP core via AXI C2C is in some cases nearly two orders of magnitude slower than with the HBICAP IP core. In case of the HWICAP, the reconfiguration throughput is limited by AXI C2C, as can be seen by the RP size independent configuration speed in Figure 8. This is also underlined by the fact that in other comparable studies without AXI C2C, the HWICAP is significantly faster and the performance gap to the HBICAP is considerably smaller [17], [19]. In contrast, reconfiguration with the AXI HBICAP using full AXI4 for data transfer is not limited by AXI C2C, as shown by the RP size-dependent throughput in the same figure. From Figure 8 it can also be seen that the impact of AXI C2C is almost negligible when the HBICAP is used. This is especially visible at the 4 MB data point, where the setup with AXI C2C achieves an average of 84.8 MB/s versus 88.0 MB/s in the setup without AXI C2C, corresponding to a marginal deviation of approximately 3.4%.

The slow first reconfiguration, as seen in Figure 7, was also investigated. This effect is only observed after a hard restart of the MPSoC, and not when the peripheral FPGA is restarted. Thus, it is unlikely that this effect is related to AXI C2C, the ICAP interface controller, the FPGA bridges, or the RP itself. Since the effect occurs with both the HBICAP and HWICAP, it is also not related to the CDMA. Furthermore, the effect also occurred if the first measurement was not started immediately after power-up, but with a delay in the order of minutes. Most likely, this effect is caused by the PS of the MPSoC or by the Linux operating system running on it. However, the exact cause could not be identified yet.

As was previously shown by Vipin and Fahmy, with a custom controller the ICAP interface can reach a maximum throughput of 382 MB/s, without overclocking the configuration interface and utilising a system clock of 100 MHz [17]. This is approximately twice as fast as the highest average reconfiguration throughput we have achieved with the HBICAP from AMD Xilinx at 178 MB/s (see Figure 8). Based on our investigations so far, it is not possible to give a statement about how much of this deviation is caused by the HBICAP IP core, the Linux operating system, or our custom HBICAP FPGA manager. Therefore, we intend to continue our work by investigating the performance of a custom ICAP controller in the C2C configuration. Additionally, further improvements may also be achieved through optimised settings in Linux or improvements on the custom HBICAP FPGA manager. In particular, possibilities to accelerate the first reconfiguration of an RP after system startup should be investigated, because this is crucial if the proposed approach is eventually being used to initialize the system. Nonetheless, the reconfiguration speed achieved in this work, and the proposed method utilising the AXI HBICAP IP core are already suitable for use in the multi-device QiController.

VIII. Conclusion

This work presented an AXI C2C-based approach to initialise modular and scalable heterogeneous DAQ systems composed of multiple MPSoCs and FPGAs from AMD Xilinx. It uses the Linux operating system on the MPSoC to manage the entire process. In particular, the FPGA subsystem of Linux is used together with device tree overlays to perform partial reconfiguration on the peripheral FPGAs. Our implementation of suitable FPGA manager drivers is released to the public [29]. A series of measurements were performed on three different setups using evaluation cards to determine the performance and reliability of the method.

A firmware architecture made entirely of AMD Xilinx IP cores and built around the combination of AXI HBICAP and AXI CDMA proved the feasibility of using this architecture for the multi-device QiController, currently being designed following the ATCA standard. The reconfiguration throughput of 178 MB/s achieved with the current implementation has the potential to even further improve with the use of custom ICAP and CDMA controllers. Therefore, we intend to explore this avenue in the coming months.

References

[1] T. Mehner et al., “Lessons learned while developing the Serenity-S1 ATCA card,” in *JINST 19 C02018*, 2024. doi:10.1088/1748-0221/19/02/C02018.

[2] T. Mehner et al., “ZynqMP-based board-management mezzanines for Serenity ATCA-blades,” in *JINST 17/03 C03009*, 2022. doi:10.1088/1748-0221/17/03/C03009.

[3] A. Albert et al., “The Apollo ATCA design for the CMS track finder and the pixel readout at the HL-LHC,” in *JINST 17 C04033*, 2022. doi:10.1088/1748-0221/17/04/C04033.

[4] S. Tang et al., “Prototype hardware design and testing of the global common module for the global trigger subsystem of the ATLAS phase-II upgrade,” in *JINST 17 C05026*, 2022. doi:10.1088/1748-0221/17/05/C05026.

[5] N. Loukas et al., “The CMS Barrel Calorimeter Processor demonstrator (BCPv1) board evaluation,” in *JINST 17 C08005*, 2022. doi:10.1088/1748-0221/17/08/C08005.

[6] T. Muscheid et al., “DTS-100G — a versatile heterogeneous MPSoC board for cryogenic sensor readout,” in *JINST 18 C02067*, 2023. doi:10.1088/1748-0221/18/02/C02067.

[7] J. P. Smith et al., “Highly-Multiplexed Superconducting Detector Readout: Approachable High-Speed FPGA Design,” in *2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2022. doi:10.1109/FCCM53951.2022.9786140.

[8] R. Gebauer, "A Flexible FPGA-based Control Platform for Superconducting Multi-Qubit Experiments," doctoral thesis, Karlsruhe Institute of Technology (KIT), 2022. doi:10.5445/IR/1000141695.

[9] C. Bobda et al., "The Future of FPGA Acceleration in Datacenters and the Cloud," *CHEP 2019*, vol. 15, Art. no. 34, Sept. 2022. doi:10.1145/3506713.

[10] R. Ammendola et al., "The Next Generation of Exascale-Class Systems: The ExaNeSt Project," in *2017 Euromicro Conference on Digital System Design (DSD)*, 2017, pp. 510-515. doi:10.1109/DSD.2017.20.

[11] J. Flich et al., "From a FPGA Prototyping Platform to a Computing Platform: The MANGO Experience," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021, pp. 7-12. doi:10.23919/DATE51398.2021.Flich2021.

[12] A. D. Ioannou et al., "UNILOGIC: A Novel Architecture for Highly Parallel Reconfigurable Systems," in *ACM Trans. Reconfigurable Technol. Syst.*, New York, NY, USA, 2020. doi:Ioannou2020.

[13] "AWS FPGA - Frequently Asked Questions". [Online]. Available: https://github.com/aws/aws-fpga/blob/master/FAQs.md. Accessed on: March 25, 2024.

[14] *Fast Partial Reconfiguration Over PCI Express*, version 1.0, AMD Xilinx. [Online]. Available: https://docs.amd.com/r/en-US/xapp1338-fast-partial-reconfiguration-pci-express/Summary. Accessed on: April 4, 2024.

[15] J. Ruan et al., "Increasing Flexibility of Cloud FPGA Virtualization," in *2022 32nd International Conference on Field-Programmable Logic and Applications (FPL)*, 2022, pp. 350-357. doi:10.1109/FPL57034.2022.00060.

[16] S. Di Carlo et al., "A portable open-source controller for safe Dynamic Partial Reconfiguration on Xilinx FPGAs," in *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, 2015. doi:10.1109/FPL.2015.dicarlo2015.

[17] K. Vipin and S. A. Fahmy, "ZyCAP: Efficient Partial Reconfiguration Management on the Xilinx Zynq," in *IEEE Embedded Systems Letters*, 2014, pp. 41-44. doi:10.1109/LES.2014.2314390.

[18] F. Duhem et al., "FaRM: Fast Reconfiguration Manager for Reducing Reconfiguration Time Overhead on FPGA," in *Reconfigurable Computing: Architectures, Tools and Applications*, 2011, pp. 253-260. doi:10.1007/978-3-642-19475-7.

[19] K. Pham et al., "Moving Compute towards Data in Heterogeneous multi-FPGA Clusters using Partial Reconfiguration and I/O Virtualisation," in *2020 International Conference on Field-Programmable Technology (ICFPT)*, 2020, pp. 221-226. doi:10.1109/ICFPT51103.2020.00038.

[20] E. S. Hazen et al., "The APOLLO ATCA Platform," in *Proceedings of Topical Workshop on Electronics for Particle Physics — PoS(TWEPP2019)*, 2022. doi:10.22323/1.370.0120.

[21] *Intel Quartus Prime Standard Edition User Guide - Partial Reconfiguration*, version 18.1, Intel. [Online]. Available: https://www.intel.com/content/www/us/en/docs/programmable/683499/18-1/design-planning-for-partial-reconfiguration.html. Accessed on: March 26, 2024.

[22] *Vivado Design Suite User Guide - Dynamic Function eXchange*, version 2023.2, AMD Xilinx. [Online]. Available: https://docs.amd.com/r/en-US/ug909-vivado-partial-reconfiguration. Accessed on: March 26, 2024.

[23] "Linux Kernel Documentation - FPGA Subsystem". [Online]. Available: https://www.kernel.org/doc/html/latest/driver-api/fpga/index.html. Accessed on: March 26, 2024.

[24] "PICMG Hardware Platform Management,". [Online]. Available: https://www.picmg.org/product-category/hardware-platform-management/. Accessed on: March 26, 2024.

[25] L. Calligaris et al., "Novel developments on the OpenIPMC project," in *JINST 19/02 C02079*, 2024. doi:10.1088/1748-0221/19/02/C02079.

[26] H. Krause, "Cross-Chip Dynamic Function eXchange for the Initialization of Heterogeneous Multi-FPGA Systems," M.S. thesis, Karlsruhe Institute of Technology (KIT), 2023.

[27] "Linux Kernel Documentation - FPGA Region Device Tree Binding". [Online]. Available: https://www.kernel.org/doc/Documentation/devicetree/bindings/fpga/fpga-region.txt. Accessed on: March 26, 2024.

[28] "HWICAP driver for MicroBlaze". [Online]. Available: https://github.com/Xilinx/linux-xlnx/tree/master/drivers/char/xilinx_hwicap. Accessed on: March 29, 2024.

[29] "Cross-Chip DFX Drivers". [Online]. Available: https://github.com/kit-ipe/Cross-Chip-DFX-Drivers. Accessed on: May 16, 2024.

[30] *AMBA AXI and ACE Protocol Specification*, version H, Arm Limited. [Online]. Available: https://developer.arm.com/documentation/ihi0022/h/?lang=en. Accessed on: March 29, 2024.