Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 120 (2023) 601–606

www.elsevier.com/locate/procedia

56th CIRP Conference on Manufacturing Systems, CIRP CMS '23, South Africa

Modular Hardware/Software Architecture for Edge Units in Highly Flexible Manufacturing Systems

Florian Schade[a,*], Marius Kreutzer[b], Edgar Mühlbeier[c], Eduard Gerlitz[c], Philipp Gönnheimer[b], Jürgen Fleischer[c], Jürgen Becker[a,c]

[a] *Institut fuer Technik der Informationsverarbeitung (ITIV), Karlsruhe Institute of Technology, Kaiserstraße 12, 76131 Karlsruhe, Germany*
[b] *FZI Research Center for Information Technology, Haid-und-Neu-Str. 10–14, 76131 Karlsruhe, Germany*
[c] *wbk Institute of Production Science, Karlsruhe Institute of Technology, Kaiserstraße 12, 76131 Karlsruhe, Germany*

Abstract

With the increasing demand for individualized production and swift adaptability to changing market needs, new manufacturing system concepts emerge. Aiming for high flexibility and scalability, they rely on universal reconfigurable machines that can be used in various production steps by automatically changing tools and sensors. Consequently, the data processing infrastructure is required to support the frequent exchange of process-specific software applications, ranging from control and monitoring tasks to complex sensor data processing. This can be a major challenge when computation-intensive software is supported by hardware acceleration to achieve the desired performance or latency requirements. In this paper, we present an edge unit architecture that allows for hosting such applications while meeting the needs for modularity and reconfigurability. It builds upon a hypervisor to partition fixed processing resources among the applications. We present a mechanism that extends this partitioning to field-programmable gate arrays (FPGA) by using dynamic partial reconfiguration. This provides the option to deploy application-specific hardware accelerators with computation-intensive applications while maintaining modularity and on-line reconfigurability. We conclude the paper by pointing out the potential of the proposed architecture based on a use case for automated disassembly in a reconfigurable robotic cell.

Keywords: Flexible manufacturing system (FMS); Reconfiguration; Distributed control

1. Introduction

Facing an increasing demand for individualized production and mass customization, future production systems need to fulfill high flexibility and scalability requirements while maintaining high productivity and efficiency. Since production line planning and deployment are time- and cost-intensive processes, rigid processing systems become increasingly unattractive where volatile market needs need to be met. Instead, production systems need to be designed for different product variants or even different product families. The Wertstromkinematik (WSK, Value Stream Kinematics) concept [1] aims to solve this issue by implementing a robot-based production system concept that avoids the use of specialized machines by covering all production steps using unified, flexible, robot-like kinematics. Using exchangeable end effectors comprising the tools and sensors needed for production, monitoring, and quality control, these robots can implement various handling, processing, and sensing tasks. Where complex manufacturing processes require process forces and accuracy exceeding the capabilities of today's industrial robots, the WSK concept envisions the coupling of multiple kinematics to increase the stiffness.

Frequent automatic reconfiguration is a central characteristic of a WSK production system. This includes coupling and decoupling of robots as well as automatic changeover between tools by changing end effectors. This flexibility needs to be matched by the control and data processing architecture of the production system. Control and data processing units need to be automatically reconfigurable to match the data processing needs of the processes performed in the production system. Depending on the nature of these processes, the corresponding data processing poses different requirements towards the data processing infrastructure. While process monitoring tasks re-

* Corresponding author. Tel.: +49-721-608-41972. *E-mail address:* florian.schade@kit.edu

2212-8271
10.1016/j.procir.2023.09.045

quire low-latency data processing, predictive maintenance tasks have comparably relaxed timing constraints. Similarly, processing complex sensor data, such as camera feeds, can be computationally expensive, while object position checking using touch probes requires little computing power. Where computationally expensive data processing is subject to low latency requirements, hardware-accelerated data processing may be required. This includes the use of graphics processing units (GPU) or application-specific accelerators.

To facilitate the efficient execution of data processing tasks with varying computing requirements in highly-flexible production systems, we present an edge unit architecture concept that allows for the deployment of data processing applications in combination with highly-customized, application-specific hardware accelerators, while maintaining the required modularity and reconfigurability. It supports both CPU-triggered accelerators as well as *active accelerator components* that can be used to implement sensor readout and actuator control without CPU intervention. The edge unit software bases on a hypervisor to partition processing resources among the applications, which we complement by a mechanism that allows for the concurrent deployment of multiple application-specific hardware accelerators to a field-programmable gate array (FPGA). We employ the dynamic partial reconfiguration feature of today's FPGAs in combination with a partitioning shell to enable on-line reconfiguration of hardware accelerators while enforcing access constraints to processing system resources. We conclude the paper with a discussion of the potential of the proposed architecture with respect to a use case for automated disassembly in a reconfigurable robotic cell.

2. Related work

Hypervisors are a widely-used technology to isolate applications and implement resource sharing in processing platforms. Regarding edge units, they can be used to co-host applications with varying timing requirements, e.g., to run PLC software in parallel to less time-critical software on a shared platform [2].

The integration of FPGA-based accelerators with hypervisors has been subject to extensive research in the recent years [3, 4, 5, 6]. While a main focus has been on virtualization in datacenters, their use in the context of embedded systems has gained more attention lately [7]. In the following, we will focus on System-on-Chip-based hardware platforms that tightly integrate CPUs and FPGA units with an emphasis on approaches that allow for accelerators that actively initiate accesses to the system memory (direct memory access, DMA). They allow for high flexibility with respect to accelerator functionality and high performance when processing complex data structures.

Aiming at multiplexing FPGA hardware in virtualized cloud environments, Ma et al. present the Optimus hypervisor [8]. According to the authors, it is the first hypervisor to support scalable shared-memory FPGA virtualization, which is the basis for active accelerator components. The hypervisor provides both spatial FPGA multiplexing, i.e., partitioning of the FPGA into multiple accelerators, and temporal FPGA multiplexing, i.e., sequential sharing of one accelerator between multiple hypervisor guest applications. To ensure that each accelerator faces the same memory layout as the corresponding hypervisor guest software, it uses the hardware platform's input/output memory management unit (IOMMU) using page table slicing. Optimus does not use partial FPGA reconfiguration and thus requires a reset of all accelerators when one accelerator is replaced.

In contrast, the Ker-ONE hypervisor [9] makes use of dynamic partial reconfiguration to load accelerator designs onto the FPGA in a way that is transparent to the application. It thereby provides both spatial and temporal FPGA multiplexing. While the authors briefly mention that accelerators can be programmed to perform DMA transfers, it does not become clear to which extent memory isolation and a coherent memory layout between partition and accelerator is achieved.

In contrast to both Optimus and KER-ONE, our proposed architecture avoids temporal FPGA multiplexing. This simplifies the accelerator interface and allows for continuous operation of the accelerator even when the hypervisor partition is not scheduled, thus facilitating accelerator-based low-latency control implementations. To ensure both memory isolation and a unified memory layout between accelerator and guest application, our approach utilizes the hardware platform's IOMMU. In contrast to Optimus, our partitioning shell enables the use of different IOMMU address spaces instead of page table slicing, thus simplifying memory management. Finally, our approach employs dynamic partial reconfiguration to achieve the modularity and reconfigurability required in a flexible production environment.

3. Data processing in highly flexible processing systems

Flexible manufacturing system concepts, such as WSK, exhibit several features to achieve efficient adaptability to changing product requirements. These include the use of universal manufacturing machines, flexible machine positioning resulting in a flexible layout, control devices that are not assigned to single machines but can be dynamically assigned processing tasks as required, and autonomous plant planning and management systems. As in conventional production systems, various data acquisition and processing tasks are present. These include process monitoring, quality control, and data management tasks, ranging from simple object detection checks to quality inspection and from simple data logging to data compression or holistic digital representations of machines and products. Besides, process and robot control tasks are present, which can be subject to stringent latency and safety requirements.

When such a processing system is reconfigured, the number of active kinematics, sensors, actuators, and peripheral devices in the robot's workspace may change. This affects the data processing infrastructure, where communication network load, data volumes to be processed, as well as the set of required data processing applications change. This leads to the need for online reconfiguration planning as well as a reconfigurable edge and network infrastructure.

As part of the WSK research project, a prototype of such a production system has been developed and built. In the follow-

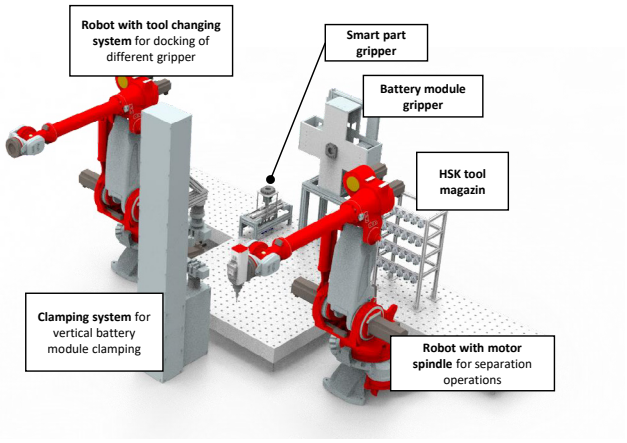

Fig. 1. Flexible robot cell for battery module disassembly

ing, its use for the flexible disassembly of battery modules is described as an exemplary use case to point out the impact on data processing and the requirements towards the edge units.

Due to the high diversity of battery variants in the electric vehicle market, a battery dismantling system that is not only capable of handling various battery models but also adapts to varying battery dimensions resulting from battery expansion is needed. Such a system is currently being developed based on the prototypical WSK production cell as illustrated in Figure 1. The process chain for disassembly includes the following sequential steps: First, the battery is gripped by a robot and inserted it into a clamping system. Then, its exact topography is determined using a 3D camera. Once the exact dimensions have been determined and trajectories for dismantling have been calculated, the welded connections of the battery housing are loosened using a milling spindle. Then, the housing components are removed using grippers. During the process, a thermographic camera continuously monitors the surface temperature and sends an alarm signal in case of thermal runaway. These process steps are executed by two robots using varying end effectors. The corresponding data processing tasks can be categorized into subclasses depending on their requirements towards the data processing infrastructure as listed in Table 1. Note that disassembly planning and code generation is not considered as they are performed on a dedicated computing unit.

Table 1. Operation classes and requirements for battery disassembly operations

Operation class	**Exemplary end effector**	**Real-time requirements**	**Data rate**	**Processing complexity**
Open-loop control	Pneumatic gripper	no	low	low
Closed-loop control	Milling spindle	yes	low	low
Safety monitoring	Thermal camera	yes	low/high	low/high
Machine Vision	3D camera	no	high	high

4. Reconfigurable edge unit architecture

To match the requirements of highly flexible production systems, the following main properties were considered during edge architecture design.

Modularity. Since data processing tasks in production systems vary greatly concerning computational effort and typically do not require all resources offered by the edge unit, multiple applications shall be hosted in parallel on a single unit to make efficient use of the unit's processing resources. In the context of the WSK concept, we expect one edge unit to implement data processing functionality related to multiple robots. Since these robots can be reconfigured independently, the corresponding applications need to be handled in a modular way. Edge units have to support adding and removing applications at runtime without affecting other applications running in parallel.

Resource management and isolation. To ensure that critical applications can meet their deadlines, resource starvation needs to be avoided. Therefore, the edge platform needs to support explicit assignment of processing resources to applications and needs to enforce resource usage limits among applications, preventing unintended interaction between applications. Resources of interest include CPU processing time, memory, and access to peripherals.

Support for flexible hardware acceleration. To allow for efficient and low-latency processing of computation-intensive workloads, the edge platform shall support the integration of hardware accelerators. To achieve functional flexibility and future extensibility, the accelerator integration mechanism shall allow for the integration of various accelerator designs. Specifically, active accelerators shall be supported. Thus, the architecture shall allow for accelerators that initiate memory and peripheral accesses to enable the implementation of control tasks by directly interfacing peripherals or to simplify accessing complex data structures without further CPU interaction. At the same time, memory isolation between different applications must be upheld as well as modularity.

4.1. Edge unit architecture overview

To meet the requirements defined above, the proposed edge architecture extends the partitioning concept implemented by hypervisors to reconfigurable FPGA regions and thereby allows for the deployment of *application modules* to partitions. An application module comprises application software for deployment on the CPU and an optional hardware accelerator component for deployment on the FPGA. As depicted in Figure 2, each module is deployed to a separate partition that comprises both the hypervisor-provided partition and a reconfigurable FPGA region. A partition is mainly characterized by the processing resources that are assigned to it. For the software component, these resources include the number of CPU cores, the share of CPU time on each core, as well as access to virtualized components, such as virtual network interfaces. For accelerator components, these resources include the FPGA region, i.e., a subset

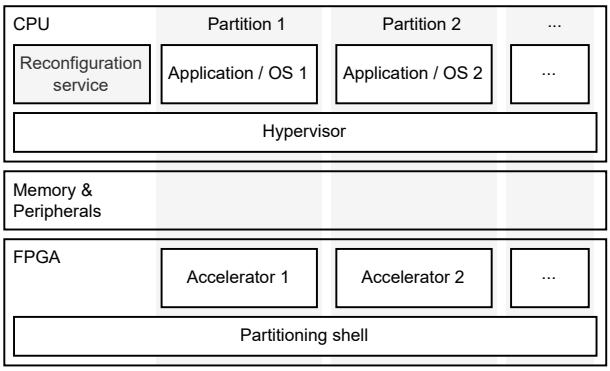

Fig. 2. Edge unit resource partitioning

of FPGA hardware resources that can be used to implement the accelerator. Access to system memory and peripherals of the hardware platform is controlled for both software and accelerator components.

To implement this partitioning, a hypervisor is running on the CPU. It realizes CPU scheduling as well as partitioning of memory resources, peripherals, and virtualized components. Besides, it provides means for inter-partition communication. Using a hypervisor to implement partitioning leads to high flexibility concerning the deployable software, since both operating-system-level and application-level software can be deployed. This can be helpful where latency-critical applications that are implemented based on real-time operating systems (RTOS) are to be deployed parallel to less time-critical software that often profits from the rich functionality provided by general-purpose operating system such as Linux.

To achieve the desired modularity with respect to accelerator components, the hypervisor is complemented by a *partitioning shell*. The partitioning shell is a static FPGA design that divides the FPGA into multiple reconfigurable regions, into which accelerator components can be deployed using dynamic partial reconfiguration (DPR). This enables modular loading and unloading of accelerator components as application modules are deployed and removed. Besides, the shell comprises static logic that implements the accelerator interfaces and signal routing required for interaction between the software component and the accelerator as well as the accelerator and system memory or peripherals. To meet the resource management and isolation requirements, the shell implements a mechanism that allows for enforcing the same memory and peripheral access restrictions for the accelerator as imposed by the hypervisor for the corresponding software components. Moreover, this mechanism is used to realize a homogeneous memory mapping for both software and accelerator. This ensures that within a partition both the software component and the accelerator component use the same memory layout and are both subject to the memory access permissions configured on the partition level. This hardware accelerator integration is described in more detail in Section 4.2.

To allow plant management systems to trigger a reconfiguration of the edge unit, a *reconfiguration service* is implemented as a privileged software component. It implements resource management by tracking the available resources on the platform while executing reconfiguration commands triggering the loading or unloading of single application modules. Upon a load command it first checks resource availability and then obtains the partition configuration, software image, and, if needed, the accelerator bitstream for the requested reconfigurable region from a central repository. It then deploys the accelerator to the FPGA region and creates the hypervisor partition using the provided configuration. When unloading an application module, it destroys the hypervisor partition and clears the reconfigurable FPGA region to destroy the corresponding accelerator.

4.2. Hardware accelerator integration

This work targets System-on-Chip (SoC) components that combine a hardwired *processing system* (PS) comprising a CPU, memory, and peripherals with an FPGA as the underlying hardware platform. These platforms allow for a tight integration of software applications and accelerators by directly connecting the FPGA to the PS memory infrastructure, interrupt lines, etc. This allows for high-bandwidth data exchange and direct access to the SoC's peripherals and memory from both subsystems.

In the proposed architecture, hardware accelerators are deployed to reconfigurable regions of the FPGA that are defined by the partitioning shell. Prioritizing flexibility, accelerator developers can make direct use of the FPGA fabric within a reconfigurable region without being restricted to a specific overlay architecture. The accelerators, however, need to adhere to a defined interface at the border of the corresponding reconfigurable region. To allow for various accelerators, this interface comprises both a slave port and a master port towards the system memory bus as well as a defined number of interrupt lines. The slave port can be used to transfer data from the CPU

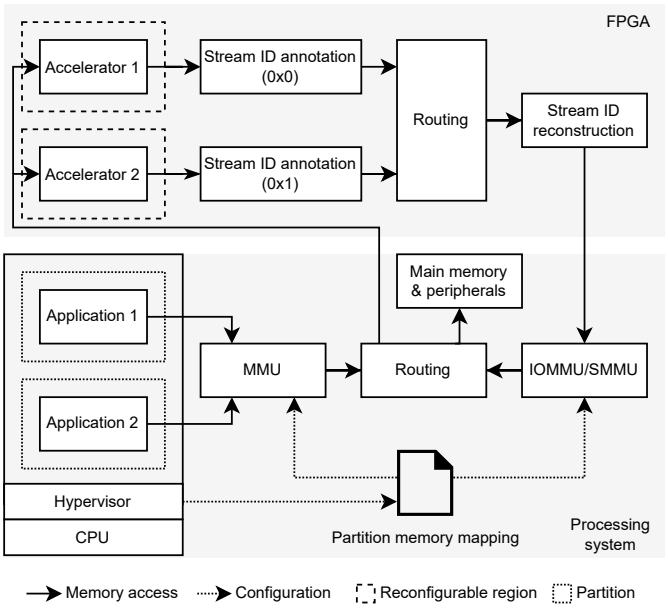

Fig. 3. Memory infrastructure for coherent addressing for both CPU-based hypervisor partition software and FPGA-based accelerators

or other master components to the accelerator. The master port can be used by the accelerator to initiate memory transactions (DMA) itself, allowing for memory and peripheral accesses in memory-mapped architectures. Interrupt lines can be used to signal events to the corresponding software components. The shell provides functionality to decouple the reconfigurable region during reconfiguration and an optional safety module that ensures that faulty accelerator implementations do not block the memory bus when incorrectly handling memory transactions.

To integrate accelerator interrupts with hypervisor partitions, the partitioning shell forwards them to the PS where they are handled by the platform's interrupt controller and thus can be mapped to the corresponding partition by hypervisor configuration. The integration of the memory interfaces is more complex, since hypervisors typically employ virtual memory addressing to provide a continuous virtual memory address space to partition software while the data can be scattered in physical memory. The required address translation is implemented by the CPU's memory management unit (MMU) based on a mapping table provided by the hypervisor. To achieve a unified memory layout for accelerators and software components in a partition, the same memory mapping needs to be implemented for memory transactions initiated by the accelerator. This can be achieved by using the platform's IOMMU (also referred to as system MMU, SMMU) as depicted in Figure 3. It can be used to apply memory address translation and enforce access restrictions for memory transactions that are initiated by other memory bus masters in the system. Since it requires information on the master interface from which a transaction originates, the partitioning shell is extended by logic to annotate unique IDs to memory transactions originating from each accelerator's master port before forwarding the transaction to the processing system. By configuring the hypervisor to create corresponding IOMMU mapping tables, the IOMMU can be configured to apply the same translations to accelerator-based memory accesses as are applied for CPU-based memory accesses.

4.3. Prototypical implementation

A prototypical implementation of the proposed architecture was created based on the Xilinx Zynq UltraScale+ MPSoC. The software architecture builds upon the XEN hypervisor [10], since it provides the required resource management capabilities, various scheduling mechanisms, including real-time schedulers, and support for the platform's SMMU. The partitioning shell was implemented as described in Section 4.2. Since the interconnect used for routing memory accesses on the FPGA discards the master interface IDs, the ID was inserted into a persistent field and later restored and inserted into the correct field before routing the transaction to the processing system, thus ensuring correct memory mapping and memory access permissions. The reconfiguration service was implemented as an application in XEN's management partition *dom0*. It provides a REST interface by which it accepts reconfiguration commands.

5. Development tool

To simplify the generation of the partitioning shell design for a given set of accelerators and the generation of the corresponding accelerator bitstreams for deployment, a development tool was created. It requires an input model that defines the location and size of each reconfigurable region, references the accelerators to be implemented, and specifies accelerator-region mappings that shall be supported. In addition, the instantiation of safety modules and the number of interrupt signal lines has to be configured for each partition. Based on this information, it generates a partitioning shell design that implements the configured regions and accelerator mappings. It outputs design artifacts which can directly be used with the FPGA vendor's synthesis tool to generate the bitstreams for deployment.

6. Discussion

The proposed edge architecture concept allows for the integration of hardware accelerators with deployed software applications in a modular way by mapping them to individual reconfigurable regions. When defining the dimensions of these regions, developers need to consider that accelerators can only be mapped to regions that provide sufficient resources for the accelerator. Thus, if multiple complex accelerators have to be deployed in parallel on the same edge unit, the partitioning shell may mainly comprise large regions. This can lead to underusage of the FPGA when low-footprint accelerators are mapped to these regions, since the remaining FPGA resources of the respective regions cannot be used by other partitions. To avoid this underusage, reconfigurable region sizes can be adjusted to the current needs of the production system at runtime by alternating between different partitioning shell designs. This, however, comes at the cost of resetting all active accelerators, since the FPGA has to be reprogrammed completely.

In contrast to conventional approaches, the proposed architecture is intended to support hosting *active accelerators* that go beyond the acceleration of single algorithms in data processing applications. As accelerators are not subject to the hypervisor scheduler, they can permanently be active as long as the corresponding application module is deployed. At the same time, the architecture allows accelerators to actively access memory and peripherals without CPU interaction. This allows for a dedicated hardware implementation of sensor readout, data processing, and actuator control, which can be helpful in fulfilling low-latency functionality beyond the reaction time achievable by CPU-based implementations. Depending on the peripherals involved, however, this comes at the cost of more complex accelerator design and challenges concerning shared usage of peripherals between accelerators and software.

Where hard real-time requirements need to be fulfilled, the suitability of direct peripheral accesses by accelerators depends on the real-time properties of the processing platform's memory subsystem. Since the shared use of interconnects within the memory subsystem may cause unpredictable delays due to parallel accesses and potentially unknown arbitration mechanisms,

further measures are necessary. One way to help control parallel accesses by hardware accelerators would be the implementation of memory bandwidth reservation as presented in [11]. Alternatively, the FPGA could be used to directly interface sensors and actuators. By realizing communication logic in the FPGA as described in [12], both processing and communication of the hard real-time functionality is completely decoupled from other, potentially interfering applications on the SoC.

6.1. Suitability for the exemplary use case

The battery dismantling process described in Section 3 comprises several sequential processing steps. Consequently, several end effector reconfigurations are necessary to implement the process in the prototype production cell, leading to the need for reconfiguration of the applications running on the edge unit.

The data processing operations required during different processing steps pose varying requirements towards the data processing infrastructure. Firstly, they vary significantly with respect to the data rate to be processed. Simple open-loop control tasks such as gripper control require little signalling. Due to infrequent gripper state changes, a low update rate is acceptable, leading to a data rate in the order of few bits per second. In contrast, machine vision tasks require the processing of streaming camera data, resulting in a significantly higher data rates in the order of hundreds of Mbit/s. Secondly, the computing complexity of the tasks varies significantly. Open-loop control tasks such as gripper control are mainly based on state machines that transition based on defined conditions. This can be handled efficiently by a CPU-based implementation. On the other hand, image processing tasks, such as interpolation or feature detection require the computation of uniform operations on large amounts of data. Thus, such applications can profit significantly from parallel processing in accelerators, especially where subsequent operations need to be applied in a pipelined fashion.

Since the dismantling process requires continuous temperature monitoring, continuous operation of the monitoring application must be ensured during reconfiguration of the edge unit. In case that the thermal monitoring application makes use of the FPGA, this is achieved as long as the partitioning shell is not replaced. If it is implemented in software only, the partitioning shell could be exchanged if necessary, e.g., to adapt to changes in other sections of the production system.

7. Conclusion

Targeting highly flexible production systems such as the WSK concept [1], we presented an edge unit hardware/software architecture that allows for the modular deployment of applications comprising both software components and application-specific hardware accelerators on a SoC-based hardware platform. A hypervisor-based software stack is complemented by a partitioning shell that divides the SoC's FPGA for reconfigurable accelerator deployment. The architecture enforces memory access controls both for software components and hardware accelerators and allows for accelerator components that actively access memory and peripherals without software intervention. In addition to the CPU-triggered acceleration of computation-intensive workload, this can be used to realize critical control functionality in dedicated hardware, where the latency of CPU-based implementations is not acceptable. We discussed the potential of the proposed architecture and pointed out strengths and limitations both for its general application and specifically in regard to an adaptive battery dismantling system that is under development using the WSK approach.

Acknowledgment

The authors would like to express their appreciation to Karlsruhe Institute of Technology for its Future Fields Funding program supporting the "Wertstromkinematik" project.

References

[1] E. Mühlbeier, P. Gönnheimer, L. Hausmann, J. Fleischer, Value Stream Kinematics, in: B.-A. Behrens, A. Brosius, W. Hintze, S. Ihlenfeldt, J. P. Wulfsberg (Eds.), Production at the leading edge of technology, Springer Berlin Heidelberg, Berlin, Heidelberg, 2021, pp. 409–418. doi:10.1007/978-3-662-62138-7_41.

[2] Linux Foundation's ACRN hypervisor achieves first commercial product integration with TTTech Industrial, https://www.tttech-industrial.com/press/first-commercial-product-integration-of-linux-acrn-hypervisor, accessed: 2023-01-04 (Dec. 2020).

[3] A. Vaishnav, K. D. Pham, D. Koch, A Survey on FPGA Virtualization, in: 2018 28th International Conference on Field Programmable Logic and Applications (FPL), 2018, pp. 131–138. doi:10.1109/FPL.2018.00031.

[4] M. H. Quraishi, E. B. Tavakoli, F. Ren, A Survey of System Architectures and Techniques for FPGA Virtualization, IEEE Transactions on Parallel and Distributed Systems 32 (9) (2021) 2216–2230. doi:10.1109/TPDS.2021.3063670.

[5] R. Skhiri, V. Fresse, J. P. Jamont, B. Suffran, J. Malek, M. Margala, From FPGA to Support Cloud to Cloud of FPGA: State of the Art, Int. J. Reconfig. Comput. 2019 (jan 2019). doi:10.1155/2019/8085461.

[6] Q. Ijaz, E.-B. Bourennane, A. K. Bashir, H. Asghar, Revisiting the High-Performance Reconfigurable Computing for Future Datacenters, Future Internet 12 (4) (2020). doi:10.3390/fi12040064.

[7] C. Wulf, M. Willig, D. Göhringer, A Survey on Hypervisor-based Virtualization of Embedded Reconfigurable Systems, in: 2021 31st International Conference on Field-Programmable Logic and Applications (FPL), 2021, pp. 249–256. doi:10.1109/FPL53798.2021.00047.

[8] J. Ma, G. Zuo, K. Loughlin, X. Cheng, Y. Liu, A. M. Eneyew, Z. Qi, B. Kasikci, A Hypervisor for Shared-Memory FPGA Platforms, ASPLOS '20, Association for Computing Machinery, New York, NY, USA, 2020, p. 827–844. doi:10.1145/3373376.3378482.

[9] T. Xia, Y. Tian, J.-C. Prévotet, F. Nouvel, Ker-ONE: A New Hypervisor Managing FPGA Reconfigurable Accelerators, J. Syst. Archit. 98 (C) (2019) 453–467. doi:10.1016/j.sysarc.2019.05.003.

[10] XEN project, https://xenproject.org, accessed: 2023-01-04.

[11] F. Restuccia, A. Biondi, M. Marinoni, G. Cicero, G. Buttazzo, AXI HyperConnect: A Predictable, Hypervisor-level Interconnect for Hardware Accelerators in FPGA SoC, in: 2020 57th ACM/IEEE Design Automation Conference (DAC), 2020. doi:10.1109/DAC18072.2020.9218652.

[12] F. Schade, C. Karle, E. Mühlbeier, P. Gönnheimer, J. Fleischer, J. Becker, Dynamic Partial Reconfiguration for Adaptive Sensor Integration in Highly Flexible Manufacturing Systems, Procedia CIRP 107 (2022) 1311–1316, leading manufacturing systems transformation – Proceedings of the 55th CIRP Conference on Manufacturing Systems 2022. doi:10.1016/j.procir.2022.05.150.