Design Abstraction for Autonomous Adaptive Hardware Systems on FPGAs

Suhaib A. Fahmy
School of Engineering
University of Warwick
Coventry, United Kingdom
s.fahmy@warwick.ac.uk

Abstract—**Adaptive hardware is gaining importance with the emergence of more autonomous systems that must process large volumes of sensor data and react within tight deadlines. To support such computation within the constraints of embedded deployments, a blend of high throughput hardware processing and adaptive control is required. FPGAs offer an ideal platform for implementing such systems by virtue of their hardware flexibility and sensor interfacing capabilities. FPGA SoCs are specifically well suited offering capable embedded processors that are tightly coupled with a flexible high performance FPGA fabric. This paper explores existing work on adaptive hardware systems before proposing a general model and implementation approach tailored towards these modern FPGA architectures, concluding with pointers for research in this emerging field.**

I. Introduction

Autonomous adaptive systems modify their behaviour based on the operating environment, applying different algorithms in evolving contexts. Example applications include advanced driver assistance [1], cognitive radio [2], and space [3] applications. For complete flexibility, software offers an easy way to combine and interleave the computational and decision making aspects of an application to build an adaptive system. Programming languages support ad hoc approaches to building such systems using control constructs, while also offering features such as polymorphism and object orientation that enable a more systematic approach. More formal software frameworks for describing and implementing adaptive systems also exist [4]. However, in the case of cyber physical systems (CPSs), where applications demand complex sensor data to be processed and acted on within strict deadlines, and potentially power and size constraints, software running on a processor is often insufficient. This has driven interest in the design and implementation of adaptive systems, where a significant portion of the processing is implemented in hardware.

Hardware systems use a compute datapath tailored to the algorithm, exploiting dataflow parallelism to achieve high performance and efficiency. To support flexibility, additional hardware for each of the required operating modes must be included and a decision made at runtime on where to route data for the current conditions. This can be problematic when there are a large number of different operating modes, since only a subset of hardware is active at any point in time, resulting in an area and power overhead.

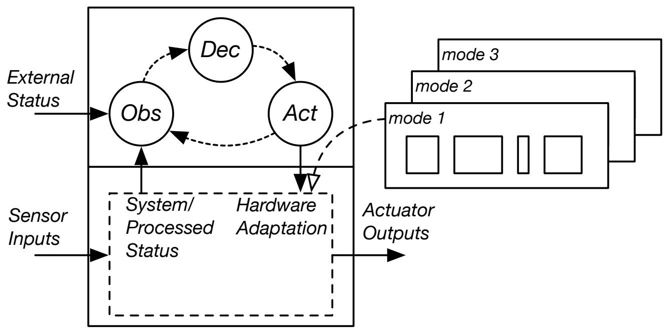

Fig. 1. Abstract representation of an adaptive hardware system showing the Observe-Decide-Act loop that is able to use external status information as well as processed information from the hardware to reconfigure the hardware among a set of different modes, in the case of an FPGA by loading the relevant bitstream.

FPGAs offer the unique capability to modify hardware at runtime through dynamic reconfiguration, where a new bitstream is loaded to replace hardware instantiated on chip. Partial reconfiguration allows just the necessary portions of the hardware design to be modified when needed. Hence, FPGAs represent an ideal platform for deploying self-contained autonomous adaptive hardware systems.

Fig. 1 shows an abstract representation of such a system. The hardware processes sensor inputs and drives actuator outputs in a cyber-physical system arrangement. The typical Observe-Decide-Act loop runs separately to the hardware. Observations come either from external sources, or as status information generated by the hardware system based on sensor inputs. Based on these inputs, the loop determines whether to modify the hardware to perform different functions, and does so by loading one of the other hardware modes that have been determined at design time. The hardware adaptation loop is managed independently of the hardware processing to prevent it from impacting the high sensor-actuator processing rates required to meet application requirements.

Much of the work reported in the literature is designed in an ad hoc manner that tightly integrates the application with the target architecture and low-level hardware features. As such the adoption of FPGAs by adaptive systems designers remains limited due to present design complexity and the requirement for detailed FPGA knowledge.

In this paper, we investigate recent work on adaptive hardware systems on FPGAs, before proposing an approach for abstracting the design of such systems that would make systematic design possible. We detail how partial reconfiguration can be automated within such a design framework to offer a truly abstract view of adaptive systems on FPGAs at both design time and runtime. Finally, open research questions relating to such abstractions are presented.

II. Background and Related Work

Adaptive hardware has been discussed widely in the literature, and so it is important to understand the varied definitions of this term. In this work, we are concerned with systems that adapt the type of computation in reaction to changing environmental conditions.

A. Types of Adaptive System

A wide body of work concerns itself with adaptation in terms of the methods of computation, but not the application itself; we can refer to these as adaptive computational frameworks. Examples include performing specific parts of a computation in software or hardware, using fewer or more accelerator blocks, swapping a high energy implementation for a more efficient block, or similar. The key consideration in such work is that there exist trade-offs in alternative ways of achieving the overall desired computation, and these are selected depending on the real time characteristics of the system [5]. These could be power constraints, execution deadlines, or other metrics driven by the computational environment. In such systems, the application itself is not adapting, but rather the way the computations are performed is adapting to maintain the correctness of the application in light of changing computational constraints [6], [7]. A subset of such work deals with adaptation to resist harsh environmental conditions such as in space, where single-even upsets (SEUs) can cause catastrophic failure. In such cases, adaptation is used to ensure the hardware continues to function correctly after encountering errors [8], [9].

An extension of the above work is adaptation of a platform to multiple competing workloads. Here, the focus is on a set of applications that each have their own execution constraints, and a hardware resource that must be shared effectively between them. Again, the applications themselves are not fundamentally changing, but their presence, absence, deadlines, or interactions may be changing, and the aim is to ensure that the hardware is able to service these needs to meet system requirements. Such work often extends existing work on multi-tasking to include awareness and management of hardware resources. Constraints such as priorities, mixed hard and soft deadlines, and overall performance and power metrics also come into play in such systems [10]–[13].

We are concerned in this paper with a separate but related class of adaptive hardware systems, wherein the application is adaptive, modifying its computations based on environmental conditions. The systems under consideration are more application specific than general purpose, and typically incorporate sensors and actuators, hence the need for hardware processing to satisfy processing requirements. Examples include a radio selecting a different baseband modulation/demodulation chain depending on spectrum occupancy [14] or an adaptive image processing [15] or smart camera vision system loading different algorithms based on an initial analysis of a scene [16]. A class of such systems are those that evolve in response to a changing environment by modifying computation to states not determined a priori. While such work has been discussed in the past [17], present hardware design flows do not support such online evolution save in the case of simple hardware, and so we will not discuss evolvable hardware in this paper.

B. Adaptive Systems on FPGAs

The hardware adaptation in an FPGA is typically implemented using partial reconfiguration [18]. This requires spatial portions of the FPGA to be allocated at design time to host different hardware modules at runtime. These partially reconfigurable regions (PRRs) should be sized to accommodate the largest hardware module they might need to contain at runtime. Partial bitstreams are prepared at design time, consisting of a configuration for a PRR for a specific hardware module. The remainder of the FPGA is the static region, containing all other parts of the system that do not change at runtime, and must include the reconfiguration controller that manages the loading of partial bitstreams into PRRs alongside other basic infrastructure like a processor, memory controller, and similar. In modern FPGA SoCs, the processor subsystem (PS) can serve the full role of the static region, and the entire programmable logic (PL) can be allocated to one or more PRRs. At runtime, the partial bitstreams for each region must be made available to the reconfiguration controller, which when triggered to initiate a reconfiguration, loads the requisite bitstream over one of the FPGAs configuration interfaces, thereby instantiating the module in the PRR [19].

The design process for partially reconfigurable systems remains cumbersome, requiring a significant amount of FPGA expertise, and is illustrated in Fig. 2. Both Xilinx and Altera support PR through their standard design tools. The designer must first decide on the number of PRRs and their location on the FPGA. Traditionally, designers have chosen to use a single PRR for simplicity, or as many PRRs as there are modules in any one configuration, to allow each module to be configured independently. It is important to note that this choice significantly impacts area overhead and reconfiguration time. The time taken to reconfigure a PRR is proportional to its size. A single PRR means that each valid configuration can be optimised into the smallest possible area by the synthesis tools, resulting in lower area usage. But a single PRR must then be reconfigured each time there is a change, even if small. Multiple PRRs offer the benefit of reconfiguring only the required modules, but result in more area consumption as they much each be sized to accommodate the largest individual modules, and synthesis cannot optimise across modules. Previous work has shown that this selection can be automated to optimise area and reconfiguration time [20]. Modules that intrinsically

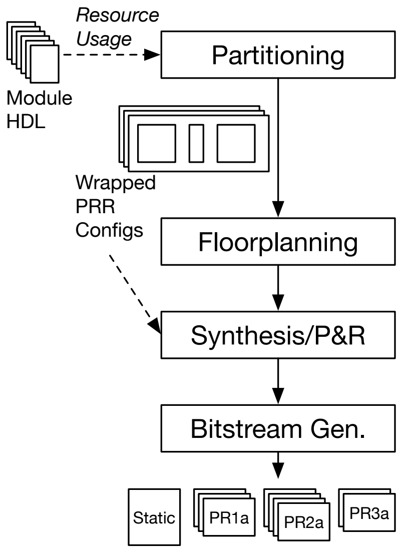

Fig. 2. An outline of the partial reconfiguration design flow, showing modules being partitioned into PRRs, wrapping of modules, followed by floorplanning based on resource requirements, then bitstream generation.

support parametric changes, such as the length of an FFT, or the the coefficient set for a filter, add further complexity as these do not require reconfiguration of a region but must have their configuration inputs exposed.

Once the number of PRRs is finalised and the allocation of modules is confirmed, wrapper modules must be generated for the valid configurations for each PRR. While this is not a complex process, it can be time-consuming and is not currently automated by the tools. After partitioning, the overall resource requirements for each PRR are known by taking the most resource intensive configuration for each PRRs. The PRRs must now be floorplanned on the FPGA. This is another step that requires significant FPGA design expertise as bounding boxes must be determined based on various physical architecture constraints while also offering sufficient slack for the tools to succeed at generating bitstreams. Some automation has been proposed in the literature [21], [22], but remains a distinct operation not integrated into vendor design tools.

The wrapped PRR configurations can now by synthesised to generate the partial bitstreams using the constraints generated in the floorplanning step. As part of this process, the bitstreams for the static region are also generated. The result of this whole process is a set of bitstreams: a static bitstream for the fixed part of the design, and multiple partial bitstreams, one for each configuration for each PRR.

Managing such a system at runtime is usually done in an ad hoc manner. The static region is configured onto the FPGA and the individual partial bitstreams are loaded into memory. The reconfiguration controller instantiated in the static region is responsible for exposing the configuration interface from within the FPGA. Software is written to explicitly load bitstream data from its memory location into the controller, and this is interleaved with the complex adaptation software. As such, the adaptation control loop cannot be written without knowledge of the detailed hardware implementation and bitstream information, resulting in poor productivity, abstraction, and portability. As a result of this design process, it is difficult for those with application knowledge but without FPGA expertise to implement autonomous adaptive systems on FPGAs. A more abstract design framework is necessary if this reconfiguration capability of FPGAs is to be more widely adopted.

III. Adaptive Hardware Model

A rich body of work has explored modelling of the adaptation process for adaptive systems primarily in the context of what has been referred to as *autonomic computing*. In contrast to traditional systems where the designer determines how best to implement computation based on predictions about the future state of the system, adaptive/autonomic/self-aware systems monitor the runtime conditions, make decisions about how best to achieve goals, and apply the determined adaptation. Numerous models offer a distributed approach to composing multiple adaptive systems.

The fundamental decomposition of this closed-loop process is the Observe-Decide-Act (ODA) loop [23]. This is loosely based on Boyd's Observe-Orient-Decide-Act loop proposed for military planning [24]. The MAPE-K loop was proposed by IBM, further decomposing the loop into four phases: Monitor, Analyze, Plan, and Execute, all interacting with shared Knowledge [25]. Multiple MAPE-K *autonomic managers* can interact with each other to adapt interacting systems. The autonomic manager interacts with the system through *sensor* and *effector* interfaces. This loose definition led to more work on formalising such models for software systems. FOCALE elaborates these processes further into inner and outer control loops, combining a subset of Observe-Normalise-Compare-Learn/Reason-Decide-Act stages. A deliberative process incorporates all these stages as we have seen for the above models, but additional reactive and reflective processes use a subset of these stages to build a longer term model of adaptation [26]. This allows multiple loops to interact to arrive at the autonomous functionality, while preserving the differing requirements of long-term learning and short term reactions. While a detailed analysis of such control models is out of the context of this paper, it is clear that an effective adaptive hardware systems framework must offer an interface to such controllers through an abstracted view of system state and a clear method for reconfiguration.

Hence, the key to such a framework is a clean separation between the control and data planes. This enables maximum throughput for the processing-intensive data plane, while allowing the control plane to adopt one of the above models using abstractions that are easier for an adaptive systems designer to work with. We see a similar separation in software-defined networking (SDN) systems, where packet processing operations must be performed at the high rates required to sustain network performance, while the software control aspects

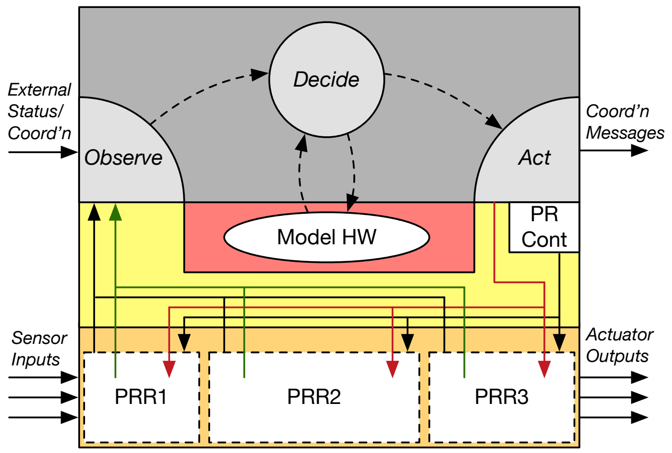

Fig. 3. The proposed model for adaptive hardware systems, showing the hardware portion containing PRRs, the PR controller, and signals for status and control, alongside the control plane in software executing the Observe-Decide-Act loop.

interact through an abstracted interface to enable flexibility in the data plane [27], [28].

We adopt a similar abstraction for adaptive hardware systems: the data plane is responsible for the high throughput computation demanded by the application, while the control loop runs primarily in software on an embedded processor. In most cases, the data plane will be concerned with processing sensor data or similar intensive computations (such as wireless baseband modulation). Hence, as is the case for the above mentioned models, it is assumed that all sensing and actuation is managed by the data plane. The data plane exposes control inputs to the control plane, while also offering the required diagnostic outputs to the control plane. It is worth noting that a complete ODA loop can be implemented in hardware, typically offering a much tighter loop and faster response times [29], however, this would limit the flexibility of the control loop. It is possible to consider a hierarchy of loops that include a fast reacting hardware loop and a more intelligent loop in software, which would be supported by some of the above models. In the case of an adaptive system on an FPGA, the control inputs must also enable reconfiguration of hardware, while the control plane must also be able to query the currently active configuration.

With more intensive control loops, incorporating long-term adaptation and learning, such as discussed in FOCALE, there may also be a need to accelerate some parts of the control loop. This requires an additional interface into hardware, separate from the data plane, to allow these functions to be accelerated without affecting the operation of the data plane.

A. Hardware Model

Fig. 3 shows the proposed model for adaptive hardware systems on FPGAs. The control plane (in grey) operates in software, while the data plane (in yellow) operates in hardware. A single or multiple PRRs are instantiated in the data plane, and can be reconfigured via the PR controller. The current status of each PRR is provided as an input to the control plane. Sensor inputs are processed in hardware to produce the required results which are passed to the control plane alongside the parameter values currently set. These are read by the **observe** stage in the control plane, and integrated with external status inputs that may be available in the case of a larger coordinated set of autonomous systems. The observe stage can be decomposed into sub-stages as in the case of FOCALE, where it incorporates normalisation and comparison against a target state.

The **decide** stage implements the cognitive function, and as in the case of the aforementioned models, can be complex. Hence, support is provided for the decide stage to interact with a hardware accelerator that is distinct from the data plane of the adaptive system. This could be a neural network or other more complex control algorithm requiring hardware implementation to achieve acceptable performance. By separating its interface from the control-data plane interface, it can be designed independently and does not impact data plane performance.

The **act** stage applies the determined adaptations through reconfiguration of PRRs using the PR controller, setting of parameters within existing PRRs, and/or sending messages to other interacting systems. In the case of PRR configuration, a single desired state change may require multiple PRR configurations and parameter settings, so this is abstracted in the act stage. The resulting hardware changes are enacted alongside any processing required to apply actuator outputs.

B. Model Implementation

The above hardware model can be implemented on an FPGA by designing the static region housing a soft processor to run the software for the control loop alongside the required interfaces to the PRRs and external sensors, actuators, or peripherals. Modern FPGA SoCs such as the Xilinx Zynq [30] and Intel Arria 10 SoC offer a tightly coupled ARM Cortex-A9 embedded processor and flexible FPGA fabric that can be reconfigured from the processor. Newer devices such as the Xilinx Zynq UltraScale+ and Intel Stratix 10 SoC offer even more capable ARM Cortex-A53 cores alongside additional hardware cores for functions such as real-time processing, security features, and high bandwidth memory (HBM). These architectures hence offer an ideal platform for a wide range of adaptive systems, offering sufficient computational capacity in software, alongside the high performance processing capabilities of the FPGA fabric. We consider these ideal for the implementation of autonomous adaptive systems as they allow the tight coupling necessary to implement the above model, with flexible hardware capabilities through partial reconfiguration.

A full detailed hardware architecture is beyond the scope of this paper, however, we briefly discuss the model's realisation in an FPGA SoC. Table I shows how each aspect of the model is implemented in the context of a Xilinx Zynq SoC. The architectural features of the Zynq, specifically the ample interfacing between the processor subsystem (PS) and programmable

TABLE I
MODEL IMPLEMENTATION DETAILS ON A XILINX ZYNQ SOC.

Model Feature	Hardware Implementation
Control loop execution	Software running on embedded ARM, atop an OS or bare metal
External status/coord	Network interface for message exchange
PR controller	PCAP or custom controller such as ZyCAP
PR bitstream storage	Separate DMA interface via a single HP port interfaced with PR controller
Hardware observe interface	ACP port
Parameter interface	AXI-Lite over GP port
Sensor/actuator interface	Direct connection or AXI-Stream
Inter-PRR interconnect	AXI-Stream or AXI Interconnect configured as required by the application
Optional decision accelerator	DMA over HP port

logic (PL) offers sufficient distinct interfaces to support the above model. High Performance (HP) ports offer multiple high throughput PL master interfaces to the PS. General Purpose (GP) ports offer a mixture of master and slave interfaces between the PS and PL. The Accelerator Coherency Port (ACP) offers a cache-coherent high throughput interface for PS-PL data movement. These interfaces all support Advanced Extensible Interface (AXI) signalling. While the Zynq supports native configuration using the Processor Configuration Access Port (PCAP) and standard software driver, this is a blocking operation which is not ideal for the closed loop nature of this above model. Hence an alternative configuration approach as in ZyCAP [31] is preferred as it is non-blocking and also faster.

The above description is based on the assumption that the system being controlled is primarily hardware based, with the processor managing the adaptation control loop. In the case where part of the processing is software based, it is possible to use an additional processor core for these functions, or instantiate a soft processor in the programmable logic. On newer FPGA SoCs with different core types, additional cores may be suited to application computation and could be integrated over the required interconnect.

IV. ADAPTIVE HARDWARE DESIGN FLOW

While the above framework offers a more abstract method for implementing autonomous systems, integration with an automated design flow remains necessary for wider adoption. The following processes would need to be considered.

The definition of valid system **configurations** is an important part of the process as it would drive generation of the required bitstreams. Due to limitations in the present FPGA PR design flow, the designer must determine and prepare the configurations in advance. While this suggests an inability of the system to evolve beyond a predefined set of states, the parametric configurations of individual PRR modules does provide some additional flexibility. These parameters can be exposed to allow the correct control interface to be built as required by the framework. Modules that offer information used in the control loop also have these outputs exposed for creation of the monitoring interface. With advances in the PR flows, it may be possible that the requirement for configuration definition in advance be relaxed to enable evolutionary adaptation. Transformations can also be made at this point to eliminate unneeded states or find equivalences to reduce the state space [32].

The tool flow must then decide how to **partition** and allocate modules to PRRs. A detailed discussion of the considerations is presented in [20]: A single region offers simpler reconfiguration management, but wastes resources and time. A large number of regions offers better configuration granularity but complicates the hardware design process and introduces significant constraints on PRR sizes. Meanwhile, a combination of PR for modules that require significant hardware changes, and parametric reconfiguration for modules with minor changes (such as the coefficients of a filter), offers a balance that minimises reconfiguration time [33]. This decision can be automated, resulting in the generation of all the required bitstreams as in [19]. The tool elaborates all supported configurations into a set of symbols that have parameter values exposed as properties, offering an abstract view of configurations rather than PRRs and bitstreams.

The **runtime system** supports the operation of the adaptive control loop. It offers an API that is used to write the adaptation software, referencing the symbols produced in the previous phase. A set of standard methods for each of the observe-decide-act processes are provided and can be extended to support more complex models. If hardware acceleration is required, an accelerator can be interfaced separated to the data plane as shown in Fig. 3. At runtime, the various processes operate independently with message passing enabling communication between them to close the loop and enable adaptation.

V. TOPICS FOR INVESTIGATION

The following research topics would enable this proposed hardware model to be realised on modern architectures:

- A PR generation tool flow that generates bitstreams and abstract symbols from a configuration specification, respecting the limitations of vendor design flows to be supported on emerging devices.
- Research relating to PRRs and bitstream relocation could greatly simplify the bitstream generation process while also making the addition of new modes to a deployed system possible.
- A open software runtime and architectural template that builds on the adaptive system model to automate interface generation to configuration management and module parameters, with abstracted runtime management.
- A module interface specification that supports composition in such systems with clear demarcation between data plane and control plane I/O.

- Libraries of domain specific hardware modules conforming to the above specification, allowing designers to focus on the adaptive design rather than low level hardware.

VI. Summary

We have discussed autonomous adaptive systems in the context of FPGAs, proposing a hardware model based on existing abstract Observe-Decide-Act models. We have discussed its implementation on modern FPGA SoCs, showing how the architectural features of such platforms can support the needs of such a framework. While a significant body of work has dealt with individual aspects of the proposed approach, we feel that there remains a need for these to be brought together into a usable framework, combining hardware and software, to enhance the usability of FPGAs in this important area. With the increased focus on autonomy in systems and the growing reliance on data-intensive sensors, such a framework would enable a more systematic approach to design and deployment than has been demonstrated to date.

References

[1] N. Harb, S. Niar, M. A. Saghir, Y. El Hillali, and R. B. Atitallah, "Dynamically reconfigurable architecture for a driver assistant system," in *Proceedings of the IEEE Symposium on Application Specific Processors (SASP)*, 2011, pp. 62–65.

[2] S. Shreejith, B. Banarjee, K. Vipin, and S. A. Fahmy, "Dynamic cognitive radios on the Xilinx Zynq hybrid FPGA," in *Proceedings of the International Conference on Cognitive Radio Oriented Wireless Networks (CROWNCOM)*, 2015, pp. 427–437.

[3] N. Montealegre, D. Merodio, A. Fernández, and P. Armbruster, "In-flight reconfigurable FPGA-based space systems," in *Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2015.

[4] R. De Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel *et al.*, "Software engineering for self-adaptive systems: A second research roadmap," in *Software Engineering for Self-Adaptive Systems II.* Springer, 2013.

[5] X. An, E. Rutten, J.-P. Diguet, N. Le Griguer, and A. Gamatié, "Autonomic management of dynamically partially reconfigurable FPGA architectures using discrete control," in *Proceedings of the International Conference on Autonomic Computing (ICAC)*, 2013, pp. 59–63.

[6] F. Sironi, M. Triverio, H. Hoffmann, M. Maggio, and M. D. Santambrogio, "Self-aware adaptation in FPGA-based systems," in *Proceedings of International Conference on Field Programmable Logic and Applications (FPL)*, 2010, pp. 187–192.

[7] M. Happe, E. Lübbers, and M. Platzner, "A self-adaptive heterogeneous multi-core architecture for embedded real-time video object tracking," *Journal of Real-Time Image Processing*, vol. 8, no. 1, pp. 95–110, 2013.

[8] A. Jacobs, G. Cieslewski, A. D. George, A. Gordon-Ross, and H. Lam, "Reconfigurable fault tolerance: A comprehensive framework for reliable and adaptive FPGA-based space computing," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 5, no. 4, pp. 21:1–21:30, 2012.

[9] R. Glein, B. Schmidt, F. Rittner, J. Teich, and D. Ziener, "A self-adaptive SEU mitigation system for FPGAs with an internal block RAM radiation particle sensor," in *Proceedings of the IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2014, pp. 251–258.

[10] M. Ullmann, M. Hübner, B. Grimm, and J. Becker, "On-demand FPGA run-time system for dynamical reconfiguration with adaptive priorities," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, 2004, pp. 454–463.

[11] J. Becker, M. Hubner, G. Hettich, R. Constapel, J. Eisenmann, and J. Luka, "Dynamic and partial FPGA exploitation," *Proceedings of the IEEE*, vol. 95, no. 2, pp. 438–452, 2007.

[12] D. Gohringer, M. Hübner, V. Schatz, and J. Becker, "Runtime adaptive multi-processor system-on-chip: RAMPSoC," in *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, 2008.

[13] R. Bitirgen, E. Ipek, and J. F. Martinez, "Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach," in *Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2008, pp. 318–329.

[14] J. Lotze, S. A. Fahmy, J. Noguera, B. Ozgul, L. Doyle, and R. Esser, "Development framework for implementing FPGA-based cognitive network nodes," in *Proceedings of the Global Telecommunications Conference (GLOBECOM)*, 2009.

[15] R. Salvador, A. Otero, J. Mora, E. de la Torre, T. Riesgo, and L. Sekanina, "Self-reconfigurable evolvable hardware system for adaptive image processing," *IEEE Transactions on Computers*, vol. 62, no. 8, pp. 1481–1493, 2013.

[16] J.-P. Diguet, Y. Eustache, and G. Gogniat, "Closed-loop–based self-adaptive hardware/software-embedded systems: Design methodology and smart cam case study," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 10, no. 3, pp. 38:1–38:28, 2011.

[17] G. W. Greenwood and A. M. Tyrrell, *Introduction to evolvable hardware: a practical guide for designing self-adaptive systems*. John Wiley & Sons, 2006, vol. 5.

[18] K. Vipin and S. A. Fahmy, "FPGA dynamic and partial reconfiguration: A survey of architectures, methods, and applications," *ACM Computing Surveys*, vol. 51, no. 4, pp. 72:1–72:39, 2018.

[19] K. Vipin and S. A. Fahmy, "Mapping adaptive hardware systems with partial reconfiguration using CoPR for Zynq," in *Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2015.

[20] K. Vipin and S. A. Fahmy, "Automated partitioning for partial reconfiguration design of adaptive systems," in *Proceedings of the International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW)*, 2013, pp. 172–181.

[21] K. Vipin and S. A. Fahmy, "Architecture-aware reconfiguration-centric floorplanning for partial reconfiguration," in *Proceedings of the International Symposium on Applied Reconfigurable Computing (ARC)*, 2012, pp. 13–25.

[22] M. Rabozzi, G. C. Durelli, A. Miele, J. Lillis, and M. D. Santambrogio, "Floorplanning automation for partial-reconfigurable FPGAs via feasible placements generation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 1, pp. 151–164, 2017.

[23] H. Hoffmann, "SEEC: A framework for self-aware management of goals and constraints in computing systems," Ph.D. dissertation, Massachusetts Institute of Technology, 2013.

[24] J. R. Boyd, "The essence of winning and losing," *Unpublished lecture notes*, vol. 12, no. 23, pp. 123–125, 1996.

[25] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[26] J. Strassner, N. Agoulmine, and E. Lehtihet, "FOCALE: A novel autonomic networking architecture," in *Proceedings of the Latin American Autonomic Computing Symposium (LAACS)*, 2006.

[27] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we ready for SDN? Implementation challenges for software-defined networks," *IEEE Communications Magazine*, vol. 51, no. 7, pp. 36–43, 2013.

[28] S. Kim, J.-M. Kang, S.-s. Seo, and J. W.-K. Hong, "A cognitive model-based approach for autonomic fault management in OpenFlow networks," *International Journal of Network Management*, vol. 23, no. 6, pp. 383–401, 2013.

[29] K. Vipin, S. Shreejith, S. A. Fahmy, and A. Easwaran, "Mapping time-critical safety-critical systems to hybrid FPGAs," in *Proceedings of the IEEE International Conference on Cyber-Physical Systems, Networks, and Applications (CPSNA)*, 2014, pp. 31–36.

[30] L. H. Crockett, R. A. Elliot, M. A. Enderwitz, and R. W. Stewart, *The Zynq Book: Embedded Processing with the ARM Cortex-A9 on the Xilinx Zynq-7000 All Programmable SoC*. Strathclyde Academic Media, 2014.

[31] K. Vipin and S. A. Fahmy, "ZyCAP: Efficient partial reconfiguration management on the Xilinx Zynq," *IEEE Embedded System Letters (ESL)*, vol. 6, no. 3, pp. 41–44, 2014.

[32] J. Lotze, S. A. Fahmy, J. Noguera, and L. E. Doyle, "A model-based approach to cognitive radio design," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 2, pp. 455–468, 2011.

[33] T. H. Pham, S. A. Fahmy, and I. V. McLoughlin, "An end-to-end multi-standard OFDM transceiver architecture using FPGA partial reconfiguration," *IEEE Access*, vol. 5, pp. 21 002–21 015, 2017.