System Simulation and Optimization using Reconfigurable Hardware

Martin Lukasiewycz[1], Shanker Shreejith[1,2], Suhaib A. Fahmy[2]

[1] TUM CREATE, Singapore, Email: martin.lukasiewycz@tum-create.edu.sg
[2] School of Computer Engineering, Nanyang Technological University, Email: shreejit1@e.ntu.edu.sg

***Abstract*—This paper presents a methodology for simulating and automatically optimizing distributed cyber-physical systems using reconfigurable hardware. By mapping an entire system of distributed devices including the buses onto a single Field Programmable Gate Array (FPGA), it becomes possible to make changes to the architecture, including the topology, using reconfiguration. This approach enables accurate rapid prototyping of distributed architectures, while also closing the gap between early simulation results and the final design, leading to a more robust optimization. Furthermore, the system can be simulated and optimized within a Hardware-in-the-Loop (HIL) setup due to its cycle- and bit-accurate execution in real-time. We introduce the general concept and building blocks that enable a faster and more accurate simulation and optimization: (1) The details of our approach for mapping devices, network interfaces, and buses onto an FPGA are presented. (2) An optimization model is proposed that encodes the topology, task distribution, and communication in a very efficient representation. Finally, the implementation and integration of the methodology is presented and discussed.**

I. Introduction

Designing cyber-physical systems under stringent cost and safety constraints is challenging in many domains. In the automotive domain, for instance, the reduction of cost is a major objective while safety, on the other hand, cannot be compromised. The complexity of cyber-physical systems is growing rapidly while time-to-market is constantly reducing. This leads to the need for novel simulation and optimization methods for distributed cyber-physical systems that are efficient, fast, and accurate.

Presently, simulations in the early stages of design are carried out at the system-level, with several implementation details masked in order to reduce simulation complexity and time. This, however, may mean important details of the implementation are not factored in and emerge only after the integration and testing of these systems. This leads to a gap between simulation results and behavior of the system in the actual implementation. In a worst-case scenario, this mismatch might require a redesign of one or more aspects of the distributed system.

Besides simulation and optimization, validation of cyber-physical systems in safety-critical domains is essential using HIL testing. Here, the distributed cyber-physical architecture is tested either with a real plant or a real-time simulator (of the physical environment). However, simulating a distributed cyber-physical system accurately in real-time is often impossible due to the complex computation involved, prohibiting testing in a HIL setup. Therefore, design flaws are typically only determined at a late stage, once a prototype has been integrated. This is in itself a tedious task involving the correct configuration and wiring of the distributed devices. In such a scenario, a structural or topological change of the distributed architecture becomes a very time-consuming task. As a result, investigations are typically limited to a few possible arrangements out of the many possible.

This work was financially supported by the Singapore National Research Foundation under its Campus for Research Excellence And Technological Enterprise (CREATE) programme.

As a remedy, we propose a methodology that enables an efficient, fast, and accurate design of distributed cyber-physical systems. Simulation, optimization, and HIL testing benefit from the proposed approach that maps the entire system onto an FPGA.

Related Work. FPGAs are widely used for prototyping in the embedded systems domain. This is particularly the case for System-on-Chip (SoC) and Application-Specific Integrated Circuit (ASIC) design, see [1], [2], [3], [4]. Here, FPGAs are used to validate and test the designed hardware/software systems. On the other hand, FPGAs are also used for compute-intensive designs where the systems benefit from dynamic/run-time reconfiguration, such as in specific video processing applications, see [5]. In this paper, we aim at combining the advantages of FPGAs in the field of prototyping of *networked* embedded systems, modeling a set of interconnected devices instead of SoCs or ASICs. At the same time, we take advantage of the reconfigurability of FPGAs in our optimization process.

The efficient optimization of distributed systems needs a model that is capable of capturing resource allocation, task distribution, and communication, see [6]. In [7], a basic model that follows the Y-chart approach was introduced: An application is mapped to a target architecture where specific resources can be allocated. This model and the underlying optimization suffer from two shortcomings which are the restriction to single-hop communication as well as an optimization that relies on Evolutionary Algorithms that might deliver many infeasible solutions. To overcome these drawbacks, [8] extended the model towards multi-hop communication and translated the model into a set of linear constraints that are interpreted by a Pseudo-Boolean (PB) solver that ensures that all iteratively determined solutions are feasible. In this paper, we extend the model of [8] significantly by taking links in the architecture template into account, leading to a higher expressiveness of the model. At the same time, we provide a more compact representation, relying on significantly fewer variables.

Contributions of the paper. We present a methodology for the simulation and optimization of distributed cyber-physical systems using reconfigurable hardware. For this purpose, the entire system is mapped onto an FPGA, leveraging the flexibility and computational power of reconfigurable hardware. This approach has major advantages in various domains: (1) Simulations can be significantly improved in terms of both accuracy and runtime. It is possible to emulate an entire system comprised of multiple devices with different clock frequencies that communicate via shared buses in a cycle- and bit-accurate manner. The concurrent execution of system nodes on a single hardware device also accurately mirrors a real implementation. Due to the inherently parallel architecture of FPGAs, simulations can be carried out in less time and might be further accelerated by using multiple devices in parallel that evaluate different scenarios, investigating the same system. (2) It is possible to use the FPGA within a HIL test since it is now feasible to simulate the system in real-time (if the node frequencies do not exceed the device limitations, which is often the case for many distributed systems). Furthermore, it is possible to model field buses like FlexRay on the fabric of an FPGA. As a result, HIL tests can be conducted very efficiently, avoiding the need to set up the entire distributed system before a valid configuration is found. (3) The ability to change the distributed architecture through reconfiguration makes it possible to implement an efficient and accurate optimization. Hence, various designs can be evaluated in sequence or in parallel on multiple FPGAs. At the same time, it becomes possible to employ automatic optimization within a HIL setup, since even changes to the topology of the system are possible through a reconfiguration of the hardware.

In this paper, we introduce the general concept how simulation, HIL testing, and optimization of distributed systems can be improved using FPGAs. We discuss in detail our prototypical mapping framework of Electronic Control Units (ECUs) and buses (using FlexRay) onto reconfigurable hardware in Section II. In Section III, we propose a very compact system model that enables the efficient optimization of distributed systems in terms of resource allocation, task distribution, and communication. Finally, we present our integration approach for the proposed methodology in Section IV.

II. System on FPGA Mapping

In order to ensure cycle- and bit-accurate emulation of a networked cyber-physical system on an FPGA, it is necessary to faithfully represent the computational modules (ECUs) as well as the communication infrastructure (buses) of the distributed system.
Computational Modules – ECUs. Within the system, each ECU performs computational tasks on a predefined hardware architecture. It comprises one or more processing cores that execute the software-based algorithms, compute accelerators that are used to speed up complex computations, and the network interfaces used to communicate with other elements of the system. In our implementation, the processing core is implemented using one or more instances of the Microblaze soft processor from Xilinx. Supporting logic, like accelerators, memories, sensor interfaces and the network interface are integrated with the processing core using standard high performance bus protocols like Advanced Microcontroller Bus Architecture (AMBA) Advanced eXtensible Interface (AXI). The application software/real-time Operating System (OS) interacts with the supporting components and interfaces, using standard function calls provided by the OS layer, which are taken care of by the driver functions generated by the Xilinx Embedded Development Kit (EDK) tool chain.

To provide seamless migration in a HIL test environment and to interface with a wide range of possible input/output modules, we have designed a generic Sensor/Actuator Interface Module (SAIM) that uses a burst-capable 32-bit communication protocol to the external sensors and actuators, modeled around the AXI specification. Clock domain crossing circuits are built into the SAIM so that the sensors/actuators can be clocked independently of the processing logic. Streaming interfaces are used at the processor end to facilitate high throughput communication and drag-and-drop functionality during the design phase.

The network interface (using FlexRay in this case) is an FPGA-optimized implementation with an enhanced set of features that enable tight coupling with the processing core with minimal resource utilization for a compact ECU implementation on the FPGA fabric [9]. Configurable features like time-stamping, data-layer header insertion, and processing capabilities built into the interface logic are used to debug and monitor communication within the test setup.
Communication architecture and buses. The communication architecture is built using multiple dual-channel FlexRay buses within the FPGA fabric. Individual ECUs are then integrated onto the respective bus segment(s), as defined in the system architecture. Each ECU is treated as an isolated computing unit, with independent control signals and isolated clocks (HC_x and IC_x). A scheme for a single bus architecture is shown in Fig. 1.

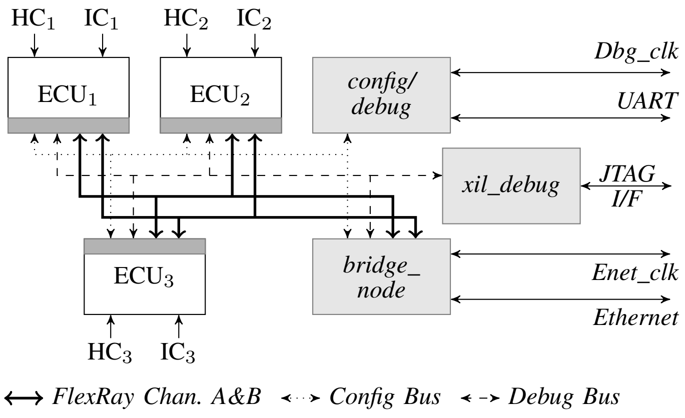

Fig. 1. Communication infrastructure on the FPGA. Both the communication bus (FlexRay) as well as auxiliary buses for configuration and debugging are included.

Bus driver modules are excluded since the network is completely contained within the FPGA logic. Over this shared medium, the ECUs post/subscribe to communications and synchronize with each other, as defined by the FlexRay protocol. Because of the reliable closed environment, we are able to take advantage of the 8-bit serial redundancy of the FlexRay standard by using a byte-wide bus structure to achieve lower sampling and transmission frequencies for the peak data rates supported by FlexRay, enabling a higher acceleration ratio. Multiple FlexRay bus segments are built as modular blocks, allowing them to be configured and connected in any fashion dynamically, without requiring a reconfiguration of the FPGA. This modular mechanism enables us to model multiple topologies and explore a large configuration space within the optimization framework.

In addition to the computational modules, management and debug modules are also integrated into the system for controlling and monitoring the performance of the ECUs, network and the hardware platform. The *bridge_node* is a debug ECU which monitors bus transactions and mirrors them on a debug console in a workstation over a high speed Ethernet interface in real-time. It also features configurable error injection capabilities to introduce common system errors like network faults, timing errors and others, which can be used to test system robustness. The *xil_debug* interface provides access to individual ECUs for debugging purposes using the Xilinx debugger tools, while *config/debug* is used to configure and control the entire platform and the fault-injection capabilities on the *bridge_node*. The platform features are configured in the global register space of the platform, which can be altered at runtime. We have also designed high level software Application Programming Interfaces (APIs) to communicate with these interfaces from a workstation. The APIs modify the behavior of the individual ECUs or the platform by modifying ECU configurations or the global register space that configures the platform features. These APIs are also indirectly used by the optimization framework to control, observe, and evaluate the performance of the ECU/system architecture and communication schedules along with information from the control plants. Further details of the platform can be found in [10].

III. System Model for Optimization

To enable an efficient optimization that leverages the proposed mapping of a system onto FPGAs, it becomes necessary to introduce a model that captures all aspects that need to be optimized, i.e., the topology, task distribution, and communication. Corresponding to [8], the proposed approach uses a graph-based model which is transformed into a set of constraints that are used within an iterative optimization framework. However, in contrast to [8],

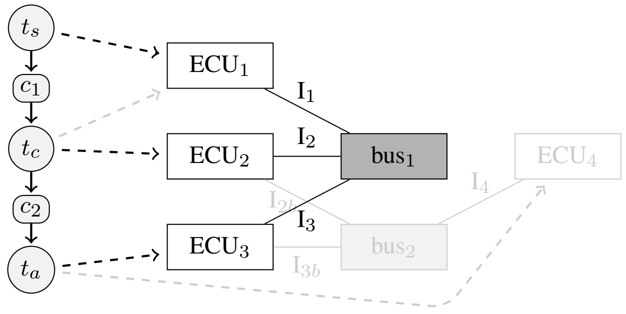

Fig. 2. Illustration of a graph representation of a distributed architecture where an application is mapped to an architecture. The routing of messages (c_1 and c_2) is carried out over bus$_1$.

our model also considers architecture links (network interfaces) explicitly, resulting in a significantly more compact representation. In summary, the advantages are (1) the higher expressiveness and flexibility of the model by taking architecture links into account as well as (2) a more efficient encoding with less variables. We present our model and the encoding scheme below, and make it available at [11].

Model. The model relies on an architecture graph, applications, and the mapping edges between these.

- The architecture graph $G_R(R, L)$ contains the components R and communication links L. The components might be ECUs or buses, while network interfaces are represented by links.
- The application graph $G_A(T, D)$ consists of tasks $T = P \cup C$ and data-dependencies D where the tasks might be either processes P or messages C.
- The mapping edges E_M with $m = (p, r) \in E_M$ and $p \in P$, $r \in R$ indicate on which resource a certain process task can be implemented.
- For each message $c \in C$, the subgraph of the architecture $G_c(R_c, L_c)$ indicates on which resources and links it can be routed.

To determine one feasible implementation, a resource allocation, mapping, and routing have to be determined.

- The allocated architecture graph $G_\alpha(R_\alpha, L\alpha)$ contains the subset of components $R_\alpha \subseteq R$ and communication links $L_\alpha \subseteq L$ that are part of the implementation.
- The mapping $E_\beta \subseteq E_M$ contains the actual implementation of each process task where each process task is mapped to exactly one allocated resource.
- The routing $G_{\gamma,c}(R_{\gamma,c}, L_{\gamma,c})$ is a graph on the allocated architecture graph that determines the routing of the respective message $c \in C$. The graph $G_{\gamma,c}$ is a tree that starts at the mapping target of the preceding process task and contains all target resources of the succeeding process tasks. This ensures that all data-dependencies are fulfilled.

An example of the graph-based optimization model is given in Fig. 2.

Encoding. In the following, an encoding of the model into linear constraints with binary variables is given. This encoding ensures that each feasible solution of the constraints also represents a feasible implementation of a system. The encoding relies on the following set of variables.

$\mathbf{r}$	resource r is allocated in R_α (1) or not (0)
$\mathbf{l}$	link l is allocated in L_α (1) or not (0)
$\mathbf{m}$	mapping m is used in E_β (1) or not (0)
$\mathbf{c_r}$	message c is routed on resource r in $R_{\gamma,c}$ (1) or not (0)
$\mathbf{c_{l=(r,\tilde{r})}}$	message c is routed on link l in the direction from resource r to $\tilde{r}$ in $L_{\gamma,c}$ (1) or not (0)
$\mathbf{c^p_{l=(r,\tilde{r})}}$	message c is routed exclusively to process p on link l in the direction from resource r to $\tilde{r}$

The constraints that determine a feasible implementation are defined in the following. We specify all constraints that deal with mapping of process tasks.

$\forall p \in P$:

$$\sum_{m=(p,r)\in E_M} \mathbf{m} = 1 \tag{1}$$

$\forall m = (p, r) \in E_M$:

$$\mathbf{r} - \mathbf{m} \geq 0 \tag{2}$$

$\forall p, \tilde{p} \in P, (p, \tilde{p}) \in D, m = (p, r) \in E_M, \tilde{m} = (\tilde{p}, \tilde{r}) \in E_M, l = (r, \tilde{r}) \in L$:

$$\mathbf{l} - \mathbf{m} - \tilde{\mathbf{m}} \geq -1 \tag{3}$$

$\forall p, \tilde{p} \in P, (p, \tilde{p}) \in D, m = (p, r) \in E_M, \tilde{m} = (\tilde{p}, \tilde{r}) \in E_M, (r, \tilde{r}) \notin L, r \neq \tilde{r}$:

$$\mathbf{m} + \tilde{\mathbf{m}} \leq 1 \tag{4}$$

$\forall l = (r, \tilde{r}) \in L$:

$$-2 \cdot \mathbf{l} + \mathbf{r} + \tilde{\mathbf{r}} \geq 0 \tag{5}$$

Constraint (1) states that for each process task exactly one mapping is active. Constraint (2) ensures that a process task is only mapped to allocated resources. For each pair of directly communicating process tasks that are implemented on neighbor resources, the link between the resources has to be allocated as stated in Constraint (3). If resources are not neighbors or equal as stated in Constraint (4), the mappings of these process tasks onto these resource is prohibited. Constraint (5) ensures that the allocation of a link requires the allocation of the source and target resource.

In the following, all constraints that concern general message routing are specified.

$\forall c \in C, r \in R_c$:

$$\mathbf{r} - \mathbf{c_r} \geq 0 \tag{6}$$

$\forall c \in C, l = (r, \tilde{r}) \in L_c$:

$$\mathbf{l} - \mathbf{c_{l=(r,\tilde{r})}} \geq 0 \tag{7}$$

$$-2 \cdot \mathbf{c_{l=(r,\tilde{r})}} + \mathbf{c_r} + \mathbf{c_{\tilde{r}}} \geq 0 \tag{8}$$

$\forall c \in C, p \in P, (p, c) \in D \vee (c, p) \in D, m = (p, r) \in E_M, r \in R_c$:

$$\mathbf{c_r} - \mathbf{m} \geq 0 \tag{9}$$

$\forall c \in C, p \in P, (p, c) \in D \vee (c, p) \in D, m = (p, r) \in E_M, r \notin R_c$:

$$\mathbf{m} = 0 \tag{10}$$

$\forall c \in C, (p, c) \in D, m = (p, r) \in E_M, l = (\tilde{r}, r) \in L_c$:

$$\mathbf{m} + \mathbf{c_{l=(\tilde{r},r)}} \leq 1 \tag{11}$$

$\forall c \in C, r \in R_c$:

$$\sum_{l=(\tilde{r},r)\in L_c} \mathbf{c_{l=(\tilde{r},r)}} \leq 1 \tag{12}$$

$$-\mathbf{c_r} + \sum_{(c,p)\in D, m=(p,r)\in E_M} \mathbf{m} + \sum_{l=(r,\tilde{r})\in L_c} \mathbf{c_{l=(r,\tilde{r})}} \geq 0 \tag{13}$$

$$-\mathbf{c_r} + \sum_{(p,c)\in D, m=(p,r)\in E_M} \mathbf{m} + \sum_{l=(\tilde{r},r)\in L_c} \mathbf{c_{l=(\tilde{r},r)}} \geq 0 \tag{14}$$

Constraint (6) and (7) ensure that a message can only be routed on allocated resources and links. As stated in Constraint (8), a message that is routed on a link from r to $\tilde{r}$ also is routed on these two respective resources. Constraint (9) specifies that input and output messages of a process task have to be routed on the same target resource as the process is mapped to. Correspondingly, if the routing of one of these messages is not possible on the specific target resource, the mapping is disabled as stated in Constraint (10).

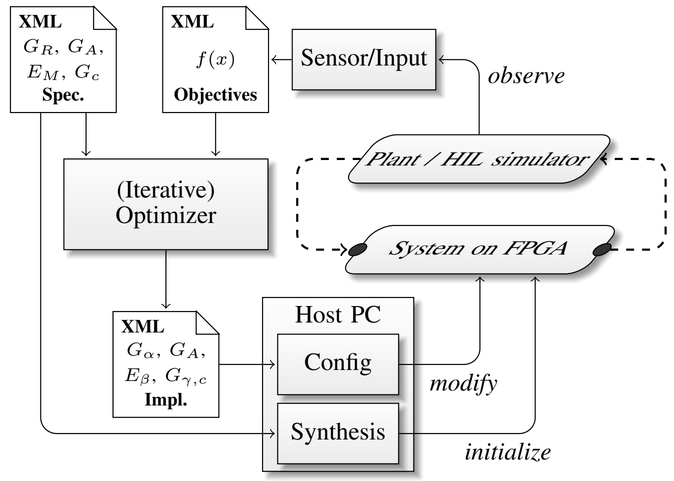

Fig. 3. Illustration of the integration of proposed approach using a HIL. (In case of a simulation without HIL, the system on the FPGA is directly observed and evaluated.)

Constraint (11) specifies the source of the message routing as the resource of the sending process task. To ensure that the routing is a tree, the degree of incoming message flows of a resource cannot be more than 1 as stated in Constraint (12). Constraint (13) specifies that a message is either received at a specific resource or forwarded to the next one. Constraint (14) states that if a message passes a resource, this resource either has to be the sender or there has to exist an incoming message flow.

In case of a unicast communication (a single receiver of a message), the following constraints are defined.

$\forall c \in C, r \in R_c :$

$$\sum_{l=(r,\tilde{r}) \in L_c} \mathbf{c_{l=(r,\tilde{r})}} \leq 1 \tag{15}$$

$\forall c \in C, (p,c) \in D, (c,\tilde{p}) \in D, r \in R_c :$

$$\sum_{l=(r,\tilde{r}) \in L_c} \mathbf{c_{l=(r,\tilde{r})}} - \sum_{l=(\tilde{r},r) \in L_c} \mathbf{c_{l=(\tilde{r},r)}} = \sum_{m=(p,r) \in E_M} \mathbf{m} - \sum_{m=(\tilde{p},r) \in E_M} \mathbf{m} \tag{16}$$

Constraint (15) states that a message cannot be forwarded to more than one receiver. Constraint (16) ensures the message flow conservation such that the flow starts at the target resource of the predecessor task and ends the target resource of the successor task. In all other cases, the incoming flow of a resource equals the outgoing flow.

In case of a multicast communication (multiple receivers of a message), the variables $\mathbf{c^P_{l=(r,\tilde{r})}}$ to determine the flow for each receiver task and then map the communication onto all sub-flows. Alternatively, the hop-based encoding from [8] might be applied.

IV. Integration

In the following, we give an overview of the entire optimization flow of our methodology and discuss the scalability.

Optimization flow. Fig. 3 illustrates the integration of the proposed approach, coupling our methodology of mapping a system on FPGAs (Section II) with the optimization method (Section III). The optimization is carried out in an iterative process such that the system implementation is constantly modified, mapped to the FPGA, and observed. Using the Opt4J framework [12], we implemented our optimization model within a meta-heuristic algorithm (Evolutionary Algorithm) such that it is capable of handling multiple and non-linear objectives. To interface the optimizer, *XML* representations of the system specification, system implementations, and observed objectives are used. The objectives might be determined by observing the simulator or plant and deducing for instance control quality measures. At the same time, it is possible to directly observe the system on the FPGA to evaluate specific objectives for the optimization such as worst-case communication delays, etc.

Both the optimizer and FPGA are initialized once with the system specification that contains the architecture template and application as well as mapping and routing information as specified in Section III. Since each architecture of an implementation (G_α) is a subgraph of the architecture template (G_R), we can avoid recurrent synthesis operations which can be very time consuming. In our setup, the synthesizer invokes the Xilinx tool chain using *tcl* scripts to generate a bitstream corresponding to the seed architecture. Bitstream generation is a time consuming process, typically requiring 30 minutes to a few hours depending on the complexity of the circuit and the chosen FPGA. Therefore, the synthesis path can be thought of as a one-time process, representing the initialization of the FPGA by the *initialize* operation. Once, the system specification is mapped to the FPGA, each implementation can be mapped by modifying (*modify* path) the architecture, task mapping, and communication which does not require any time consuming synthesis. As mentioned in Section II, high-level APIs provide access to these enhancements for software-based control, which is used in the optimization routine. Specifically, the *mod_platform()* API is used to change the bus architecture, alter the clock frequency, and enable/disable specific features on any ECU without requiring a new bitstream to be generated. Similarly, the *init_processor()* API can be used to load new software or task profile to a specific ECU without affecting the others.

Scalability. In our present design, we can map up to six complete Microblaze-based ECUs and four different bus segments on a commercial Xilinx ML605 development board that incorporates a Virtex-6 LX240T device. On a Xilinx VC707 development board that incorporates a Virtex-7 X485T device, we can map up to ten complete ECUs, thus capturing many subsystems in the automotive domain.

From our implementations, we observed that the factors limiting scalability on a single FPGA are the clocking requirements for isolating ECUs as well as the logic/memory capacities of the device. Using larger FPGAs and multi-FPGA boards, it will be possible to integrate even more complex systems with more ECUs.

References

[1] H. Nikolov, M. Thompson, T. Stefanov, A. Pimentel, S. Polstra, R. Bose, C. Zissulescu, and E. Deprettere, "Daedalus: toward composable multimedia MP-SoC design," in *Proc. of DAC*, 2008, pp. 574–579.

[2] R. Ohlendorf, T. Wild, M. Meitinger, H. Rauchfuss, and A. Herkersdorf, "Simulated and measured performance evaluation of RISC-based SoC platforms in network processing applications," *Journal of Systems Architecture*, vol. 53, no. 10, pp. 703–718, 2007.

[3] M. Gschwind, V. Salapura, and D. Maurer, "FPGA prototyping of a RISC processor core for embedded applications," *IEEE Trans. on VLSI Systems*, vol. 9, no. 2, pp. 241–250, 2001.

[4] J. J. Rodriguez-Andina, M. J. Moure, and M. D. Valdes, "Features, design tools, and application domains of FPGAs," *IEEE Trans. on Industrial Electronics*, vol. 54, no. 4, pp. 1810–1823, 2007.

[5] C. Claus, W. Stechele, M. Kovatsch, J. Angermeier, and J. Teich, "A comparison of embedded reconfigurable video-processing architectures," in *Proc. of FPL*, 2008, pp. 587–590.

[6] A. Sangiovanni-Vincentelli and M. Di Natale, "Embedded system design for automotive applications," *Computer*, vol. 40, no. 10, pp. 42–51, 2007.

[7] T. Blickle, J. Teich, and L. Thiele, "System-level synthesis using evolutionary algorithms," *Design Automation for Embedded Systems*, vol. 3, no. 1, pp. 23–58, 1998.

[8] M. Lukasiewycz, M. Streubühr, M. Glaß, C. Haubelt, and J. Teich, "Combined system synthesis and communication architecture exploration for MPSoCs," in *Proc. of the DATE 2009*, 2009, pp. 472–477.

[9] S. Shreejith and S. A. Fahmy, "Extensible FlexRay communication controller for FPGA-based automotive systems," *IEEE Transactions on Vehicular Technology*, vol. To Appear, 2015.

[10] S. Shreejith, S. A. Fahmy, and M. Lukasiewycz, "Accelerating validation of time-triggered automotive systems on FPGAs," in *Proc. of International Conference on Field Programmable Technology*, 2013, pp. 4–11.

[11] OpenDSE, "Open design space exploration framework," http://opendse.sf.net/.

[12] M. Lukasiewycz, M. Glaß, F. Reimann, and J. Teich, "Opt4J: A modular framework for meta-heuristic optimization," in *Proc. of the GECCO 2011*, 2011, pp. 1723–1730.