Smart Network Interfaces for Advanced Automotive Applications

Shanker Shreejith and Suhaib A. Fahmy
University of Warwick, UK

Computing in vehicles has increased dramatically, with electronic control units (ECUs) communicating over increasingly complex and heterogeneous networks and presenting challenges in scalability, validation, and security. In this article, we describe the concept of smart network interfaces incorporating programmable computation at the network layer to enable hardware-level fault tolerance, application consolidation with sufficient isolation, and system-level security.

In automotive networks, compute and communication are often considered distinctly, though they affect each other significantly. Overall system validation is a process that involves understanding the computational and communication delays and how these impact the higher layer applications implemented on these networks. Within this context, network interfaces serve simply to move data between the processors in the ECUs and the network, abiding by the specifications of the adopted protocol.

Computation in vehicular systems is organized in domains (such as the body domain, powertrain, and infotainment), with each domain being served by a network protocol that satisfies its specific requirements in terms of bandwidth, reliability, and other properties.[1] The ECU systems for each domain integrate the respective network interfaces as an integrated peripheral on the same die (as a subsystem on the microcontroller itself) or through a separate ASIC interfaced externally. In either case, the network interface is an implementation of the defined protocol, such as the Bosch e-Ray in the case of FlexRay, and the protocol is adhered to closely.

Extensions to standard protocol-layers allow unique features to be implemented; for example, a Controller Area Network (CAN) protocol called CAN+ offers 16× higher bandwidth than standard CAN, while maintaining backwards-compatibility with traditional CAN devices.[2] We have proposed the idea of network-layer data-processing extensions as a way to support additional features. We designed an extended FlexRay network interface[3] on an FPGA platform to show that a layer of configurable extensions offers additional capabilities (such as synchronous timestamps and on-the-fly message monitoring) that can be leveraged for unique platform-level capabilities (such as network security).[4] These network layer extensions can also enable capabili-

Published by the IEEE Computer Society
0272-1732/18/$33.00 ©2018 IEEE

ties such as security through direct traffic monitoring on the network,[5] as well as support light-weight authentication schemes.[6] Such network layer extensions can also enable deterministic routing of messages between different domains. In commercial Ethernet switches, FPGAs have been employed in line-rate switching systems for high-speed Ethernet, in-network traffic analyzers, and intrusion detection.[7] In the case of deterministic Ethernet standards such as time-sensitive networks (TSNs), extensions within the network layer enable efficient implementation of fault-tolerance strategies, such as seamless redundancy, by managing low-level tasks such as packet-level retransmission and removal at the network layer.[8] In the automotive domain, such applications are typically built using processors with multiple interfaces, incurring significant latency when moving data between Ethernet and legacy automotive networks.[9] A smart FlexRay controller can also be incorporated in a gateway on a hybrid FPGA platform to enable deterministic interconnection of legacy automotive network standards and Ethernet, with data-path extensions at the network interface layers to ensure low-latency switching performance and efficient message mapping for priority messages even with high network loads.[10]

While computation in the automotive domain has predominantly used automotive-grade processors and microcontrollers, the idea of using FPGAs has gained some traction. Within the automotive domain, FPGAs have been proposed as a compute platform for accelerating real-time vision-based driver assistance systems.[11] FPGA-based architectures can also enable architecture-level fault-tolerance through physical-level reconfiguration.[12]

In this article, we show how programmable data paths in network interfaces can enable unique capabilities to support the increasing demands placed on automotive networks, and we validate these ideas on FPGAs.

GENERAL CONCEPT

In a traditional ECU setup, all computation is done in software that runs on an off-the-shelf automotive-grade microcontroller unit that integrates the network interface and other peripherals (see Figure 1). The application receives information from the sensors over the network and processes it to determine what control outputs need to be fed to an actuator block or passed to another ECU. These individual tasks are invoked periodically based on a predefined schedule. The network interface only manages functions related to the protocol, passing data between the network and the ECU processor, and not offering any additional capability that would require computational ability. As a result, any enhancement, such as monitoring health data or timestamping individual messages, must be managed through additional software on the ECU at the application layer.

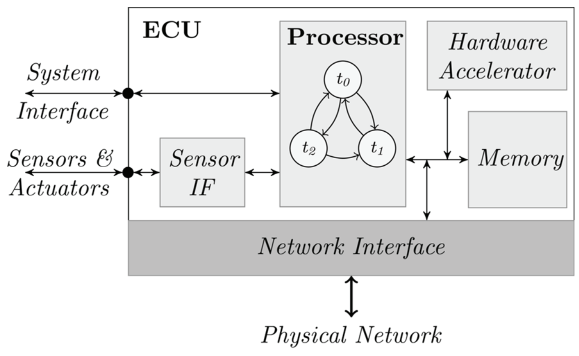

Figure 1. Typical ECU architecture incorporating one or more processing cores, memory elements, sensor interfaces, network interfaces, and hardware accelerators.

Incorporating such system-level and low-level tasks in software can be challenging. Firstly, network and software tasks are not always synchronized within an ECU, and software tasks are not synchronized across ECUs. Such capabilities at the application layer require additional tasks to be added, increasing processor load and potentially requiring extensive rescheduling and revalidation of ECU functionality. Secondly, establishing a synchronized time-base across ECUs can be challenging because ECUs might not be active at all times. Finally, tasks on ECU processors are susceptible to delays resulting from interrupts and priority tasks with strict deadlines. All this can result in non-deterministic responses to events on the network or triggers from other ECUs.

Introducing computational capability in the network layer can overcome these challenges. The capability can transparently augment network data to add information about system state, maintain a synchronized time-base across all ECUs, or apply other processing. This works similarly to layered protocol encapsulation in networks like Ethernet—the network interface adds a set of headers and timestamp information before embedding application data. At the receiving end, these headers are handled by the network layer, while application data is passed on to the higher-layer software tasks. Such a protocol architecture for FlexRay is shown in Figure 2, with a 2-byte header and a 4-byte synchronous timestamp being added as data-layer headers by the network interface. The header can embed information about the state of the ECU (for diagnosis, fault-tolerance measures, or other purposes), the data format (for packing sequences of data together), and flags that indicate the presence of special network-layer messages.

Introducing computational capability in the network layer can overcome [many] challenges.

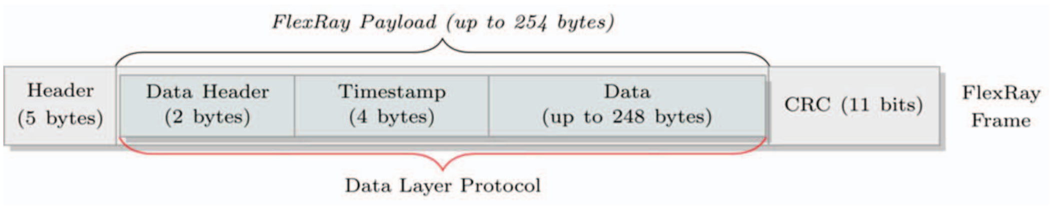

Figure 2. Data-layer headers embedded by the intelligent network controller.

To achieve this transparent and deterministic functionality, a processing path parallel to the regular dataflow in the network interface is integrated, as shown in Figure 3, for the receive path for a FlexRay network interface. This parallel path allows regular messages to be forwarded to the software processor along the traditional path, while special messages (network-level or adaptation schemes) can be processed deterministically within the interface itself, eliminating the need for software changes to handle such enhancements. In the receive direction, the extensions receive the decoded bytes from the network and use pattern detectors to extract the data-layer headers or other information from the payload segment. The timestamp logic leverages the network protocol synchronization scheme and extends this to offer a synchronized view of time across all participating ECUs.

The extensions can trigger specific actions to enhance capabilities; for example, a mode adaptation message for an ECU that supports multiple operating modes (like a terrain response system) could be triggered directly from the extension, mitigating the non-deterministic factors caused by interrupts and software delays. Cipher primitives integrated within the data path (as shown in Figure 3) perform on-the-fly decryption on the protocol headers and application data without incurring additional latency. In the transmit direction, the extensions operate in the reverse order. They use information about system state (periodically updated by the application) and the timestamp logic to form a data-layer header, which is fed to the encryption block (if integrated) that takes 8-byte blocks (starting from the protocol header) and obfuscates them before encoding them to the bit-level format for transmission on the network.

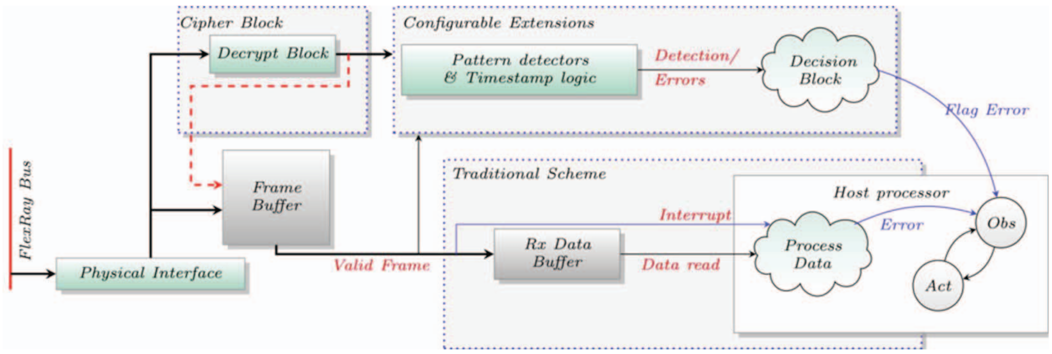

Figure 3. Configurable extensions embedded within the receive path of the intelligent FlexRay network interface.

Hence, integrating such intelligence at the network interface in a transparent manner enables unique capabilities without incurring valuable processor time to manage such low-level tasks. While we validate these ideas in the next section using FPGAs, this concept can be easily applied within new ASIC or SoC network interfaces by integrating a small programmable logic path to implement these extensions in the transmission and reception chains.

CASE STUDIES

To quantify the advantages of embedding computation into the network interface, we look at three case studies that relate to enhancements discussed in the previous section. First, we look at how integrating data packing and unpacking in the network interface can reduce processor overhead. The second case study explores hardware-level adaptability by coupling a reconfiguration management system to the data-path extension in the network interface, improving the determinism over a software-driven approach. The final example shows how cipher primitives can be embedded within the data path of the network interface to offer both network and data security with no latency overhead on network transmission or impact on software applications. For all experiments, we use the Xilinx Zynq hybrid FPGA device, because it reasonably approximates a typical automotive embedded architecture (with its dual-core ARM Cortex-A9 processor), while also allowing us to integrate the proposed hardware extensions on the same platform. For software evaluations, the application is run on top of the Standalone operating system from Xilinx, a very lightweight OS that abstracts some hardware details. Details of the FlexRay network interface design are discussed in another work.[3] We use FlexRay and Ethernet as the network standards for experiments, but the same principles can be extended to other legacy automotive networks such as CAN, or newer standards such as TSN.

Handling Volume Data at Interfaces

In this case study, we consider the case of transmitting messages from a conventional FlexRay ECU to an Ethernet backbone network. We use an 8-byte message for this experiment (on FlexRay), as other work has shown that the 8-byte message size represents more than 70 percent of traffic on FlexRay-based vehicular systems. Multiple such messages are packed together to form a valid Ethernet payload of 64 bytes. With a software-based gateway, the processor has a fetch-and-pack task that is activated whenever an 8-byte FlexRay frame is received at the network interface (using an interrupt). The task reads the message into the Ethernet buffer and sets the “done” flag if the packet is ready to be transmitted (when 64 bytes have been filled), otherwise it executes other tasks and waits for the next interrupt. Each of these actions incurs some latency, as shown in Table 1, with a best-case interrupt latency of 2.96 μs. As shown, the fetch-and-pack task is executed multiple times every Ethernet frame, consuming considerable processor cycles in context switch and data movement (total latency of 26.08 μs).

Table 1. Data repacking for multi-cycle data transfers (64-byte data).

Mode	Latency Components		Total Time	Change
	Interrupt	Data Movement		
Software	2.96 µs x 8	0.3 µs x 8	26.08 µs	
Extension	2.96 µs x 1	0.3 µs x 8	5.36 µs	–79 %

Embedding this capability into the network layer allows the interface to pack multiple messages into an Ethernet payload, which can be read with a simpler fetch task, reducing latency by around 80 percent. It should also be noted that many tasks in an automotive system are non-preemptible to ensure strict deadlines, which could increase performance gains further. Finally, a fully hardware-based packing and switching system that does not rely on software tasks further cuts down the latency to 3.3 µs, including the transmission latency over the Ethernet link (through hardware-based packing and forwarding measured on actual hardware), and is a more viable solution for high-performance automotive gateways (see VEGa[10]). Such packing also applies to ECUs that deal with data-dense sensors, such as radar or cameras.

Hardware-Level Adaptation

This case study explores the benefits of coupling device-level capabilities such as dynamic reconfiguration with the data-path extensions in the network interface. Consider an ECU system that can adapt its control algorithm in response to changes in environmental conditions or user settings, like an adaptive terrain response system that is common in off-road-capable vehicles. Because these different modes of operation are mutually exclusive, it is sensible to have them swap in and out as required to save area and power. The Zynq platform enables hardware blocks to be selectively modified to adapt the processing logic through a processor-based PCAP interface. In this scenario, a software task that monitors information from sensors or user inputs (over the network) triggers a reconfiguration through the processor, keeping the processor occupied with a non-preemptible task until reconfiguration is completed.

[There are benefits to] coupling device-level capabilities such as dynamic reconfiguration with the data-path extensions in the network interface.

Alternatively, by interfacing the low-level reconfiguration primitives with the network extensions, the reconfiguration process can be fully handled by the interface, while the processor carries out its regular tasks. The custom reconfiguration system determines the mode to be chosen, fetches the new hardware configuration (through DMA) and configures the hardware block without processor intervention. The time consumed for the adaptation process (from message reception to adaptation) in both cases is shown in Table 2. The software technique keeps the processor occupied for 2.26 ms for the reconfiguration of a small hardware block (3 percent of device resources), delaying other tasks significantly. By handling the reconfiguration through the network interface, the processor continues to execute its tasks normally; this approach also offers improved reconfiguration performance (reduced by 66 percent), allowing for a faster switch to the new mode. For more complex hardware blocks that incur more resources, the processor-driven reconfiguration can result in the processor being busy for tens of milliseconds, and it might not be a viable option in critical systems.

Table 2. Comparison of adaptation times when handled through software or through the hardware extension within the smart network interface.

Mode	Latency Components			Total Time	Change
	Interrupt	Data Movement	Reconfiguration		
Software (PCAP)	2.96 µs	0.3 µs	2,257.9 µs	2,261.1 µs	
Hardware intelligence with custom ICAP	N/A	N/A	759.4 µs	759.4 µs	–66 %

Network Security

This case study shows how a security architecture can be integrated seamlessly as an extension of the network interface with zero latency overhead. Our prior work showed that security primitives within the network interface can authenticate application code and protect the network from unauthorized access.[4] However, the key challenge is to integrate this complex security architecture in a manner that introduces minimal overheads in latency (for the network or application) and without affecting protocol guarantees. For security managed through software, the encrypted message received from the network must be read and decrypted using the current configuration of the cipher primitives before the information can be used by the application. As shown in Table 3, this results in considerable overheads (41.5 µs) per 8 bytes of sensor data, for a lightweight symmetric cipher, PRESENT, at a minimal security setting of 32 rounds (meaning each block of data is encrypted and decrypted over the entire cycle 32 times). Increasing the security level (more rounds) increases the latency super-linearly due to the complexity associated with managing the cipher operations (such as memory requirements and computation of intermediate stage keys). For comparison, the slot width on a standard 5-ms FlexRay cycle that supports 64 (static) slots is around 65 µs, and the increased security level results in a lost window for transmission. Moreover, the software tasks are not synchronized to network timing, while the self-adaptive nature of networks such as FlexRay causes the application and network to drift out of sync, causing further errors due to missed transmissions.

Table 3. Latency introduced by the PRESENT cipher on a Xilinx Zynq ARM core per 8 bytes of data, compared to the smart controller that embeds the same cipher block in its data path.

Mode	Rounds	Latency Components			Total Delay
Encryption		**TS Read**	**Encrypt**	**Writeback**	
Software	32	0.3 µs	40.9 µs	0.3 µs	41.5 µs
	64	0.3 µs	82.6 µs	0.3 µs	83.2 µs
Extension	Up to 470	N/A	0 µs overlaps with txn	0.3 µs	0.3 µs
Decryption		**Data Read**	**TS Read**	**Decrypt**	
Software	32	0.6 µs	0.3 µs	42.1 µs	43.0 µs
	64	0.6 µs	0.3 µs	85.2 µs	86.1 µs
Extension	Up to 470	0.3 µs	N/A	0 µs overlaps with rxn	0.3 µs

Embedding the security primitive within the network interface allows the cipher operations to be synchronized with the network timing, ensuring guaranteed transmission at all times. Within the data path, prefetching and extensive pipelining allow the transmission/reception of the data segments to be overlapped with the encryption/decryption process. An abstract timing diagram of the process is shown in Figure 4. The frame headers are prefetched at the start of the transmission slot and are encrypted (along with the frame timestamp t_n, labelled TS) using the pre-shared key (PSK) before the start of frame sequence and the flag bits have been transmitted. Subsequently, the transmission of the frame header is overlapped with the encryption of the first 8 bytes of data and so on. The timestamp-based key technique (PSK + t_n) ensures that the encrypted data varies in every slot even if the actual application data is static, which is common in many automotive applications. Also, the overlap allows higher levels of security (up to 470 rounds) per 8-byte data block before the slightest violation of timing boundaries, as shown in Table 3 for both transmission and reception. Furthermore, the network extensions can also manage a security adaptation frame (a special frame for adapting security specifics) without intervention from the application, allowing the security scheme to be fully transparent to the application.

Embedding the security primitive within the network interface allows the cipher operations to be synchronized with the network timing, ensuring guaranteed transmission at all times.

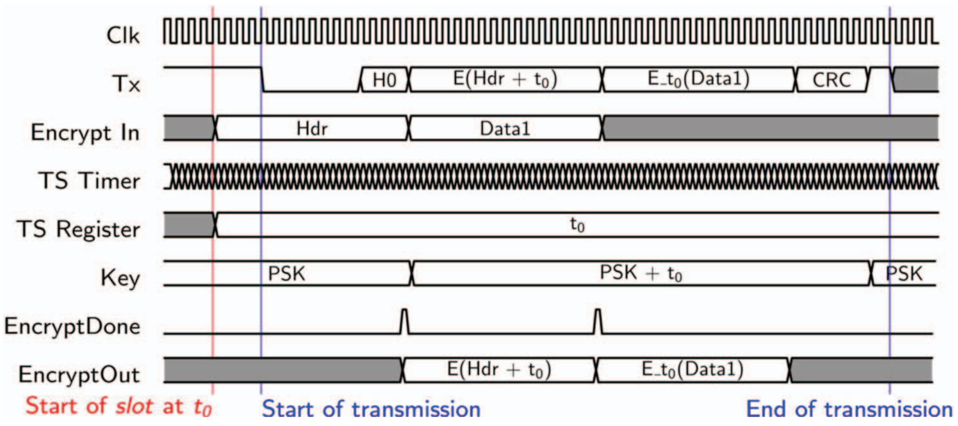

Figure 4. Timing diagram showing the overlapping of transmission with the encryption process to effectively hide the encryption latency for the header and data segment of the communication. The start of slot timestamp t_0 registered in the TS Register is used to improve the entropy of the header (by encrypting Header + t_0 using the PSK, labelled E(Hdr+t_0)) and for randomizing the data segment (by using a PSK + t_0 as key, labelled E_t_0(Data1)).

Overheads

While embedding smart capabilities into the network interface improves the overall determinism and flexibility of the system, it does incur some cost in terms of hardware resources and power consumption, as shown in Table 4, when implemented on a small Xilinx Zynq Z-7020 device. The simple data-path extensions (pattern detectors and timestamp logic) on an otherwise standard FlexRay network interface increase resource consumption by 28.9 percent (for registers, with dual-channel mode), with a negligible increase in power consumption. Interfacing the reconfiguration management increases resource consumption of the intelligent network interface by 11.8

percent (for registers), with no appreciable increase in power consumption. However, incorporating network security within the interface for both channels on a FlexRay network incurs an additional 98.7 percent of resources (for registers) and increases the overall power consumption of the network interface by 24 percent (36 mW). Similarly, incorporating the data-segment protocol discussed previously reduces the payload capacity of the FlexRay frame to 248 bytes. Despite these minor overheads (compared to the available resource on the chip, with the highest occupancy being 6 percent of LUTs), the smarter network interface offers unique ways to enhance the system's performance and capabilities, some of which are impossible to achieve using a software-based implementation.

Table 4. Area and power overheads on a Xilinx Zynq Z-7020 device.

Implementation	**Normalized Resource Consumption**			**Peak Resource**	**Power Consumption**
	Reg	**LUTs**	**BRAMs**		
FlexRay with data-path extensions	1.29 x	1.20 x	1.00 x	21.0 % (LUTs)	1.02 x
Intelligent network interface	1.42 x	1.27 x	1.06 x	22.4 % (LUTs)	1.02 x
Secure FlexRay interface	2.27 x	1.51 x	1.63 x	26.7 % (LUTs)	1.26 x

We must state once more that while these experiments were validated on FPGAs, the approach could equally be applied in the design of new network interface ASICs, where a programmable data-path segment could be integrated.

CONCLUSION

This article presents the concept of integrating a programmable computation layer within automotive network interfaces. This offers unique ways to address emerging challenges in vehicular systems, namely security, deterministic performance, and hardware-level adaptation. We demonstrated the approach using a prototype implementation of a smart FlexRay network interface on FPGA and evaluated the benefits, as well as overheads, associated with the approach. Our evaluation demonstrates that smart network interfaces offer significant improvements in terms of processing and response times over traditional software approaches.

REFERENCES

1. N. Navet and F. Simonot-Lion, *In-vehicle communication networks - a historical perspective and review*, University of Luxembourg, 2013; http://orbilu.uni.lu/handle/10993/5540.
2. T. Ziermann, S. Wildermann, and J. Teich, "CAN+: A new backward-compatible Controller Area Network (CAN) protocol with up to 16x higher data rates," *Proceedings of the Design Automation and Test in Europe Conference* (DATE), 2009; https://dl.acm.org/citation.cfm?id=1874885.
3. S. Shreejith and S.A. Fahmy, "Extensible FlexRay Communication Controller for FPGA-Based Automotive Systems," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 2, 2015, pp. 453–465; http://ieeexplore.ieee.org/document/6816104/.
4. S. Shreejith and S.A. Fahmy, "Security Aware Network Controllers for Next Generation Automotive Embedded Systems," *Proceedings of the Design Automation Conference* (DAC), 2015, p. 39:1; http://ieeexplore.ieee.org/document/7167223/.

5. P. Waszecki et al., “Automotive electrical/electronic architecture security via distributed in-vehicle traffic monitoring,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 11, 2017, pp. 1790–1803; http://ieeexplore.ieee.org/document/7849145/.
6. P. Mundhenk et al., “Security in automotive networks: Lightweight authentication and authorization,” *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 22, no. 2, 2017, p. 25:1; https://dl.acm.org/citation.cfm?id=2960407.
7. G. Carvajal et al., “Atacama: An Open FPGA-Based Platform for Mixed-Criticality Communication in Multi-segmented Ethernet Networks,” *Proceedings of the International Symposium on Field-Programmable Custom Computing Machines* (FCCM), 2013, pp. 121–128; http://ieeexplore.ieee.org/document/6546006/.
8. F. Groß et al., “A hardware/software co-design approach for ethernet controllers to support time-triggered traffic in the upcoming IEEE TSN standards,” *Proceedings of the International Conference on Consumer Electronics Berlin* (ICCE-Berlin), 2014; http://ieeexplore.ieee.org/document/7034229/.
9. J.H. Kim et al., “Gateway Framework for In-Vehicle Networks based on CAN, FlexRay and Ethernet,” *IEEE Transactions on Vehicular Technology (TVT)*, vol. 64, no. 10, 2015, pp. 4472–4486; http://ieeexplore.ieee.org/document/6960111/.
10. S. Shreejith et al., “VEGa: A high performance vehicular Ethernet gateway on hybrid FPGA,” *IEEE Transactions on Computers*, vol. 66, no. 10, 2017, pp. 1790–1803; http://ieeexplore.ieee.org/document/7917319/.
11. C. Claus et al., “Towards rapid dynamic partial reconfiguration in video-based driver assistance systems,” *Proceedings of the International Symposium on Applied Reconfigurable Computing* (ARC), 2010; https://link.springer.com/chapter/10.1007%2F978-3-642-12133-3_8.
12. S. Shreejith et al., “An Approach for Redundancy in FlexRay Networks Using FPGA Partial Reconfiguration,” *Proceedings of the Design, Automation and Test in Europe Conference* (DATE), 2013, pp. 721–724; http://ieeexplore.ieee.org/document/6513601/.

ABOUT THE AUTHORS

Shanker Shreejith is a teaching fellow at the School of Engineering, University of Warwick, UK. His research interests include distributed computing architectures and reconfigurable systems. He has a PhD in computer science and engineering from Nanyang Technological University, Singapore. Contact him at s.shanker@warwick.ac.uk.

Suhaib A. Fahmy is an associate professor at the School of Engineering, University of Warwick, UK, where he leads the Connected Systems Research Group. His research interests include reconfigurable computing, embedded systems, and hardware acceleration. He has a PhD in electrical and electronic engineering from Imperial College London, UK. Contact him at s.fahmy@warwick.ac.uk.