Smart Network Interfaces for Advanced Automotive Applications

Shanker Shreejith *Member, IEEE* and Suhaib A. Fahmy *Senior Member, IEEE*

Abstract

The computing integrated in modern vehicles has increased dramatically over the last decade, with many cars having over 50 compute units controlling critical and non-critical functions. These ECUs communicate over increasingly complex and heterogeneous networks, and these systems combined present challenges in terms of scalability, validation, and security. In this article, we present the concept of smart network interfaces that incorporate programmable computation in the datapath to enable more features at the network layer, thereby offloading auxiliary tasks from the ECU processor. System-level capabilities such as hardware-level fault tolerance, application consolidation with sufficient isolation, and system-level security at each compute node become possible without disturbing the core computational functions of the ECUs. We demonstrate this approach with practical prototyping in FPGAs.

I. Introduction and Background

In automotive networks, compute and communication are often considered distinctly, though they impact 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 organised in domains (such as the body domain, power-train, infotainment, and others) 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 sub-system on the microcontroller itself) or through a separate ASIC interfaced externally. In either case, the network interface is an implementation of the defined protocol, like 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, CAN+ offers 16$\times$ 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 a field programmable gate array (FPGA) platform to show that a layer of configurable extensions offers additional capabilities like synchronous timestamps and on-the-fly message monitoring that can be leveraged for unique platform-level capabilities like network security [4]. These network layer extensions can also enable capabilities like security through direct traffic monitoring [5] on the network, and support lightweight authentication (like the scheme in [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 analysers, and intrusion detection [7]. In the case of deterministic Ethernet standards like Time Sensitive Networks (TSN), extensions within the network layer enable efficient implementation of fault-tolerance strategies like *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 datapath 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 micro-controllers, 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

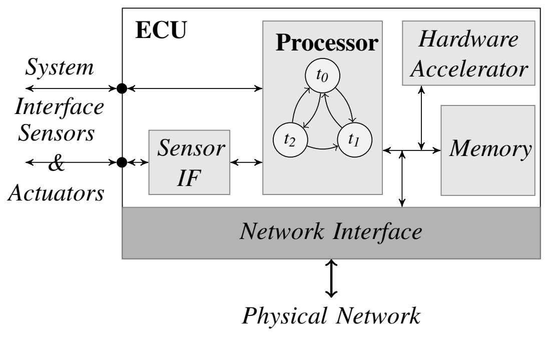

Fig. 1: Typical ECU architecture incorporating one or more processing cores, memory elements, sensor interfaces, network interface, and hardware accelerators.

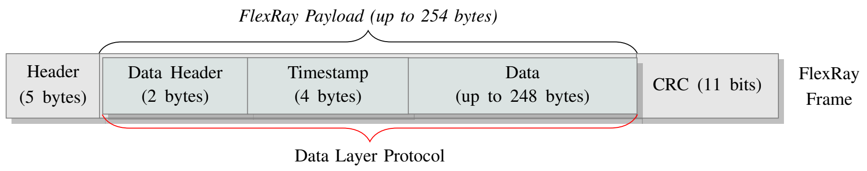

Fig. 2: Data layer headers embedded by the intelligent network controller.

architectures can also enable architecture-level fault-tolerance through physical level reconfiguration [12]. In this article, we show how programmable datapaths in network interfaces can enable unique capabilities to support the increasing demands placed on automotive networks, and we validate these ideas on FPGAs.

II. General Concept

In a traditional ECU setup, all computation is done in software which runs on an off-the-shelf automotive grade microcontroller unit that integrates the network interface and other peripherals (see Fig. 1). The application receives information from the different 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 like monitoring health data or timestamping individual messages must be managed through additional software on the ECU at the application layer.

Incorporating such system-level and low-level tasks in software can be challenging. Firstly, network and software tasks are not always synchronised within an ECU, and software tasks are not synchronised 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 synchronised timestamp across ECUs can be challenging since ECUs may 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 the communication data to add information about system state, maintain a synchronised timebase across all ECUs, or apply other processing on network data. This works similar 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 the application data is passed on to the higher layer software tasks. Such a protocol architecture for FlexRay is shown in Fig. 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 others), the data format (for packing sequences of data together), and flags that indicate the presence of a special network layer message.

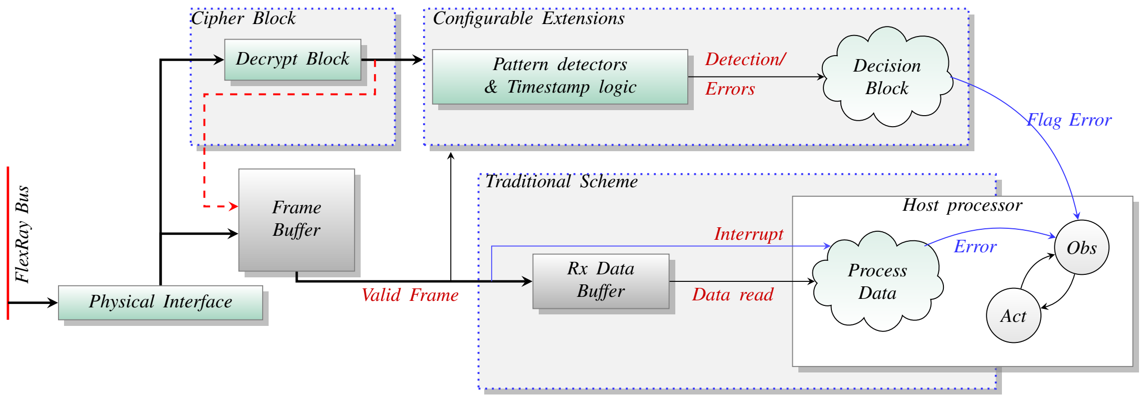

Fig. 3: Configurable extensions embedded within the receive path of the intelligent FlexRay network interface.

To achieve this transparent and deterministic functionality, a processing path parallel to the regular dataflow in the network interface is integrated into the interface, as shown in Fig. 3 for the receive path for a FlexRay network interface. The parallel path allows regular messages to be forwarded to the software processor along the traditional path, while special message (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 synchronisation scheme and extends this to offer a synchronised 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 datapath (as shown in the Fig. 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 the 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.

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.

III. 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 Sec. II. 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 datapath 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 datapath 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 as it reasonably approximates a typical automotive embedded architecture (with its dual-core ARM Cortex A9 processors), 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 can be found in [3]. We use FlexRay and Ethernet as the network standards for experiments, but the same principles can be extended to other legacy automotive networks like CAN, or newer standards.

TABLE I: Data re-packing for multi-cycle data transfers (64-byte data).

Mode	Latency components		Total time	Change
	Interrupt	Data Movement		
Software	2.96 µs × 8	0.3 µs × 8	26.08 µs	
Extension	2.96 µs × 1	0.3 µs × 8	5.36 µs	−79%

TABLE II: 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	Reconfig.		
Software (PCAP)	2.96 µs	0.3 µs	2257.9 µs	2261.1 µs	
H/W intelligence with custom ICAP	NA	NA	759.4 µs	759.4 µs	−66%

A. 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 over 70% 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 (i.e., 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 I, 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).

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%. It should also be noted that many tasks in an automotive system are non-preemptive 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.

B. Hardware-Level Adaptation

This case study explores the benefits of coupling device-level capabilities like dynamic reconfiguration with the datapath 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. Since 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 the 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-preemptive task until reconfiguration is completed.

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 II. The software technique keeps

TABLE III: Latency introduced by the PRESENT cipher on an Zynq ARM core per 8 bytes of data, compared to the smart controller that embeds the same cipher block in its datapath.

Task	Rounds	Latency components			Total delay
Encyption		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	NA	0 µs *Overlaps with txn*	0.3 µs	0.3 µs
Decyption		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	0.3µs	NA	0 µs *Overlaps with rxn*	0.3 µs

the processor occupied for 2.26 ms for the reconfiguration of a small hardware block (3 % of device resources), delaying other tasks by a significant period of time. By handling the reconfiguration through the network interface, the processor continues to execute its tasks normally; this approach also offers much improved reconfiguration performance (reduced by 66%), allowing 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 may not be a viable option in critical systems.

C. *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 unauthorised access (see [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 III, this results in considerable overheads (41.5 µs) per 8-bytes of sensor data, for a lightweight symmetric cipher, PRESENT, at a minimum security setting of 32 rounds (i.e., 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 (memory requirements, 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 synchronised to the network timing while the self adaptive nature of networks like FlexRay causes the application and network to drift out of sync, causing further errors due to missed transmissions.

Embedding the security primitive within the network interface allows the cipher operations to be synchronised with the network timing ensuring guaranteed transmission at all times. Within the datapath, 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 Fig. 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 time-stamp 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 III 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.

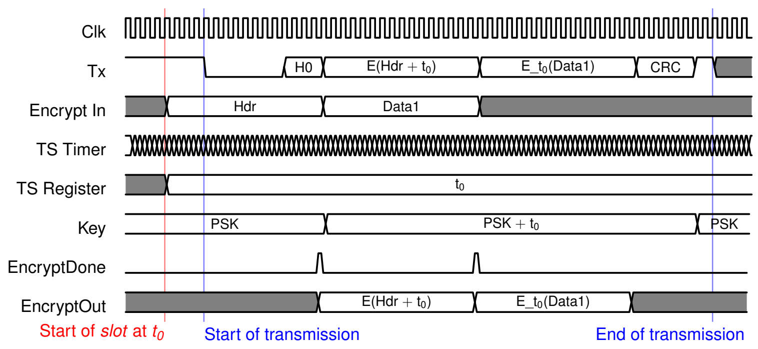

Fig. 4: Timing diagram showing the overlapping of transmission with the encryption process to effectively hide the encryption latency for the header and data segment of the communication. The start of slot timestamp t_0 registered in the TS Register is used to improve the entropy of the header (by encrypting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

TABLE IV: 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 extns	1.29 ×	1.20 ×	1 (×)	21.0% (LUTs)	1.02 ×
Intelligent network interface	1.42 ×	1.27 ×	1.06 (×)	22.4% (LUTs)	1.02 ×
Secure FlexRay interface	2.27 ×	1.51 ×	1.63 (×)	26.7% (LUTs)	1.26 ×

D. *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 IV when implemented on a small Xilinx Zynq Z-7020 device. The simple datapath extensions (pattern detectors, timestamp logic) on an otherwise standard FlexRay network interface increase resource consumption by 28.9% (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% (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% resources (for registers) and increases the overall power consumption of the network interface by 24% (36 mW). Similarly, incorporating the data-segment protocol discussed in Sec. II 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 being 6% 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.

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

IV. Conclusions

This article presented 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 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 a traditional software approach.

References

[1] N. Navet and F. Simonot-Lion, "In-vehicle communication networks-a historical perspective and review," University of Luxembourg, Tech. Rep., 2013.

[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,” in *Proceedings of the Design Automation and Test in Europe (DATE) Conference*, 2009.

[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, pp. 453–465, 2015.

[4] ——, “Security Aware Network Controllers for Next Generation Automotive Embedded Systems,” in *Proceedings of the Design Automation Conference (DAC)*, 2015, pp. 39:1–39:6.

[5] P. Waszecki, P. Mundhenk, S. Steinhorst, M. Lukasiewycz, R. Karri, and S. Chakraborty, “Automotive electrical/electronic architecture security via distributed in-vehicle traffic monitoring,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2017.

[6] P. Mundhenk, A. Paverd, A. Mrowca, S. Steinhorst, M. Lukasiewycz, S. A. Fahmy, and S. Chakraborty, “Security in automotive networks: Lightweight authentication and authorization,” *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 22, no. 2, pp. 25:1–25:27, 2017.

[7] G. Carvajal, M. Figueroa, R. Trausmuth, and S. Fischmeister, “Atacama: An Open FPGA-Based Platform for Mixed-Criticality Communication in Multi-segmented Ethernet Networks,” in *Proceedings of the International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2013, pp. 121–128.

[8] F. Groß, T. Steinbach, F. Korf, T. C. Schmidt, and B. Schwarz, “A hardware/software co-design approach for ethernet controllers to support time-triggered traffic in the upcoming IEEE TSN standards,” in *Proceedings of the International Conference on Consumer Electronics Berlin (ICCE-Berlin)*, 2014.

[9] J.-H. Kim, S. Seo, T. Nguyen, B. Cheon, Y. Lee, and J. Jeon, “Gateway Framework for In-Vehicle Networks based on CAN, FlexRay and Ethernet,” *IEEE Transactions on Vehicular Technology (TVT)*, vol. 64, no. 10, 2015.

[10] S. Shreejith, P. Mundhenk, A. Ettner, S. A. Fahmy, S. Steinhorst, M. Lukasiewycz, and S. Chakraborty, “VEGa: A high performance vehicular Ethernet gateway on hybrid FPGA,” *IEEE Transactions on Computers*, vol. 66, no. 10, pp. 1790–1803, 2017.

[11] C. Claus, R. Ahmed, F. Altenried, and W. Stechele, “Towards rapid dynamic partial reconfiguration in video-based driver assistance systems,” in *Proceedings of the International Symposium on Applied Reconfigurable Computing (ARC)*, 2010.

[12] S. Shreejith, K. Vipin, S. A. Fahmy, and M. Lukasiewycz, “An Approach for Redundancy in FlexRay Networks Using FPGA Partial Reconfiguration,” in *Proceedings of the Design, Automation and Test in Europe (DATE) Conference*, 2013, pp. 721–724.

Shanker Shreejith (S’13,M’17) received the B.Tech degree in electronics and communication engineering from University of Kerala, India and the Ph.D. degree in computer science and engineering from Nanyang Technological University, Singapore, in 2006 and 2016, respectively.

From 2006 to 2008, he worked as an FPGA design and development engineer. From 2008 to 2011, he worked as a scientist at Digital Systems Group, Vikram Sarabhai Space Centre, Trivandrum, under the Indian Space Research Organisation (ISRO). From 2015 to 2016, he was a Research Fellow at the School of Computer Science and Engineering, Nanyang Technological University, Singapore. Since 2017, he is a Teaching Fellow at the School of Engineering, University of Warwick, United Kingdom.

Suhaib A Fahmy (M’01, SM’13) received the M.Eng. degree in information systems engineering and the Ph.D. degree in electrical and electronic engineering from Imperial College London, UK, in 2003 and 2007, respectively.

From 2007 to 2009, he was a Research Fellow with the University of Dublin, Trinity College, and a Visiting Research Engineer with Xilinx Research Labs, Dublin. From 2009 to 2015, he was an Assistant Professor with the School of Computer Engineering, Nanyang Technological University, Singapore. Since 2015, he has been an Associate Professor at the School of Engineering, University of Warwick, UK. His research interests include reconfigurable computing, high-level system design, and computational acceleration of complex algorithms.

Dr Fahmy was a recipient of the Best Paper Award at the IEEE Conference on Field Programmable Technology in 2012, the IBM Faculty Award in 2013, and is a senior member of the ACM.