Design and Comparative Evaluation of GPGPU- and FPGA-based MPSoC ECU Architectures for Secure, Dependable, and Real-Time Automotive CPS

Bikash Poudel*, Naresh Kumar Giri†, and Arslan Munir‡
* Department of Computer Science and Engineering, University of Nevada Reno
†‡Department of Computer Science, Kansas State University
Email: *bpoudel@nevada.unr.edu, †ngiri@ksu.edu, and ‡ amunir@ksu.edu

***Abstract*—In this paper, we propose and implement two electronic control unit (ECU) architectures for real-time automotive cyber-physical systems that incorporate security and dependability primitives with low resources and energy overhead. These ECUs architectures follow the multiprocessor system-on-chip (MPSoC) design paradigm wherein the ECUs have multiple heterogeneous processing engines with specific functionalities. The first architecture, GED, leverages an ARM-based application processor and a GPGPU-based co-processor. The second architecture, RED, integrates an ARM based application processor with a FPGA-based co-processor. We quantify and compare temporal performance, energy, and error resilience of our proposed architectures for a steer-by-wire case study over CAN, CAN FD, and FlexRay in-vehicle networks. Hardware implementation results reveal that RED and GED can attain a speedup of $31.7\times$ and $1.8\times$, respectively, while consuming $1.75\times$ and $2\times$ less energy, respectively, than contemporary ECU architectures.**

***Index Terms*—Automotive, cyber-physical systems, GPGPU, FPGA, steer-by-wire, security, dependability**

I. Introduction and Motivation

Contemporary automobiles integrate a multitude of heterogeneous digital processors (also called electronic control units (ECUs)), radio interfaces, in-vehicle networks and protocols, and hundreds of megabytes of complex embedded software. Next generation of automobiles (also known as cybercars) will further escalate the profusion of novel distributed control applications. Emergence of x-by-wire systems, where electronic controllers replace traditional mechanical and/or hydraulic subsystems, is a prominent example of recent modernization in the automotive industry. However, x-by-wire systems (e.g., steer-by-wire, brake-by-wire, etc.) have stringent real-time performance and reliability requirements, which pose significant challenges for implementation over traditional, bandwidth limited controller area network (CAN). Since CAN is the most prevalent protocol for in-vehicle communication and most of the car manufacturers are reluctant to adopt a completely new protocol, CAN with flexible data rate (CAN FD) is a viable replacement of CAN for x-by-wire applications. Furthermore, FlexRay is another recent protocol that is well suited for x-by-wire applications as the protocol offers high speed data transfer and fault tolerance features.

As electronic components permeate into safety-critical automotive functions, integration of security and dependability in cybercar design becomes imperative. The continuously escalating complexity of automotive systems and increasing integration with wireless entities (e.g., smart phones) exacerbate the security vulnerabilities of cybercars ([10]). Furthermore, harsh operational environment combined with external noise and radiation render ECUs vulnerable to permanent and transient faults. Hence, in order to make cybercars robust to faults and security vulnerabilities, cybercars must incorporate dependability and security features. When retrofitting the in-vehicle architectures with security and dependability mechanisms, a prime challenge is to ensure that hard real-time constraints of the automotive cyber-physical applications are not violated.

The evolving nature of cyber-attacks presents another challenge in integrating security primitives in automotive cyber-physical systems (CPS). The advancements in cryptanalysis and attack technologies might render various security mechanisms ineffective. Most of the global initiatives on future automotive CPS focus on design of dedicated security solutions that could be embedded in future automotive ECUs. These security solutions are based on one of the following standards [15]: secure hardware extension, hardware security module, and trusted platform module, all of which are non-fault-tolerant (NFT) and are dedicated inflexible hardware designs. Hence, a successful attack on the security mechanisms embodied in these standards would require a complete replacement of the ECUs leveraging these security standards, which would be very costly or in worst case, infeasible. Unfortunately, the problem is not only limited to non-fault tolerance and inflexibility of these security mechanism. The constant craving of automotive industry to accommodate new cybercar applications (e.g., voice recognition and object detection for autonomous car-maneuvering system, etc.) requires ECUs with high computational power. These novel cybercar applications may not be effectively handled by contemporary microcontroller-based ECUs.

To address the above mentioned security, dependability, and performance challenges in automotive CPS design, we devise novel multiprocessor system-on-chip (MPSoC) based ECU architectures which are secure, dependable, high-performance, energy-efficient, and flexible. We consider the interplay between temporal performance and dependability in our ECU designs. We emphasize that although temporal performance (measuring timing constraints) is a quality of service (QoS) measure, the temporal performance must also be considered as a dependability measure beyond a certain *critical threshold* as the driver can totally lose control of his/her vehicle if the response time exceeds that critical threshold.

To demonstrate temporal performance, energy efficiency, and error resilience of our proposed architectures, we consider steer-by-wire (SBW) as a case study. We further compare the performance and energy efficiency of our proposed ECU architectures with a baseline ECU design (BED) that embodies the security and dependability features for future cybercars.

Our main contributions are:

- We propose two novel secure and fault-tolerant MPSoC-based ECU architectures: a general-purpose graphics processing unit (GPGPU)-based ECU design (GED) and a reconfigurable ECU design (RED), which are able to effectively meet security, dependability, and real-time requirements of automotive CPS in an energy-efficient manner.
- We implement our proposed GED architecture in NVIDIA's Jetson TK1 board, and our proposed RED architecture in Xilinx Automotive Spartan-6 field-programmable gate array (FPGA) board. Furthermore, we consider a baseline ECU design (BED) architecture, which is implemented in NXP's automotive-grade iMX6Q SABRE board (more in Section III-B).
- We model a SBW subsystem and quantify and compare the temporal performance, energy, and error resilience of our proposed ECU architectures.
- We compare response times of a SBW subsystem leveraging our proposed ECU architectures over three in-vehicle networks: CAN, CAN FD, and FlexRay.

II. Related Work

Various previous works have studied security of automotive systems. Koscher et al. [10] have examined multitude of internal and external attack surfaces of a modern automobile through which an adversary could infiltrate in-vehicle networks to control automotive subsystems (e.g., brakes, steering wheel) while ignoring the driver's input. This work is followed by a scores of studies on embedding security primitives in in-vehicle networks. A number of prior works including [12] have examined integration of message authentication codes (MACs) in CAN data frames to secure in-vehicle data interaction. Furthermore, in order to defend against masquerade and replay attacks, [14] have proposed MACs with counters. Although these methods provides message authentication, these approaches do not provide confidentiality of in-vehicle data. Sojo et al. [15] have surveyed various specifications, standards, and guidelines encompassing secure hardware extensions, hardware security modules, and trusted platform modules, etc., that could provide security for in-vehicle communications. These studies only consider security aspects but fail to analyze the simultaneous integration of security and dependability in real-time automotive CPS which is cardinal to safety and security of modern automobiles.

The dependability for automotive embedded systems has been explored by a number of earlier works. Beckschulze et al. [4] have investigated FT approaches based on dual-core microcontrollers. Baleani et al. [3] have discussed various FT architectures for automotive applications, such as lock-step dual processor architecture, loosely-synchronized dual processor architecture, and triple modular redundant architecture. However, these works have not considered the interplay of performance, dependability, and security for modern automotive CPS. Munir et al. [13] is the first work that has proposed a multicore ECU based design for secure and dependable cybercars. Yet, this work has not implemented the proposed approach on an automotive-grade processor. Furthermore, this ECU design approach may not scale to incorporate computationally intensive cybercar applications under strict real-time constraints.

III. Secure and Dependable Approach for Cybercar Design

This section first elaborates on security threat model of automotive CPS. We then elucidate a secure and dependable approach for cybercar design which we refer to as baseline ECU design (BED).

A. Security Threat Model

With a large number of ECUs operating inside cybercars, there are plenty of security attack surfaces that impact most of the in-vehicle systems. This situation is further exacerbated by the connection to increasingly wide range of external networks, from Wi-Fi, cellular networks, and the Internet to service garages, toll roads, drive-through windows, gas stations, and a rapidly growing list of automotive after-market applications. In order to elucidate the need for incorporating security primitives in ECUs, we discuss various methods in the following by which an adversary could penetrate the in-vehicle networks (e.g., CAN, FlexRay) to accomplish various security attacks.

Need for confidentiality: In-vehicle networks cart a mix of operational and personally identifiable information, such as current location, previous destinations, navigation history, call history, microphone recordings, and financial transactions, etc. An adversary invading an in-vehicle network could perform passive eavesdropping and traffic analysis, thus, obtaining critical information about the driver and the vehicle. In addition, for the x-by-wire systems, if an adversary knows the initial location of the vehicle, then, by eavesdropping on the steering angle, accelerator value, and braking value, the adversary could track the car which might put the driver and passengers at risk. Hence, the confidentiality of messages and data over in-vehicle networks is critical for operational security, privacy, and consumer trust.

Need for authentication and integrity: An attacker invading an in-vehicle network may attach his/her own device or compromise a valid user device (e.g., a cell phone attached to the infotainment system) in order to send fraudulent (or malicious) requests (commands, codes, or messages) into the system. Furthermore, the attacker's device may impersonate a valid ECU or gateway for malicious activities that may jeopardize safety of the driver and the vehicle. Additionally, the adversary may perform spoofing attacks by actively injecting and/or modifying messages in the in-vehicle network. Thus, entity authentication and message integrity verification are required in in-vehicle networks to defend against these vulnerabilities.

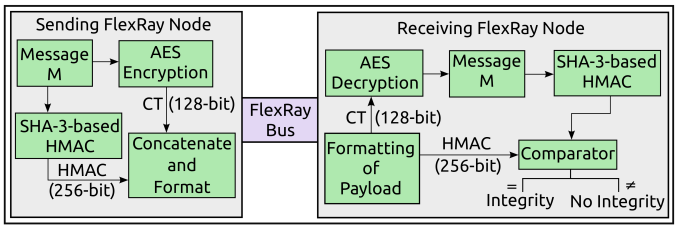

Fig. 1: *BED Architecture.*

B. Baseline ECU Design Approach

We leverage our BED approach in an enhanced version of a prior secure and dependable automotive design [13]. The BED approach is implemented in NXP's iMX6Q SABRE automotive development board. The BED represents the contemporary microcontrollers-based design used in conventional automotive ECUs. The basic security and dependability features used in our proposed ECUs are incorporated in BED. Then, the GED and the RED architectures are compared with BED architecture to access the gains in performance and energy of our proposed designs.

Security: Our BED approach leverages AES-based encryption and SHA-3-based HMAC to integrate confidentiality, integrity, and authentication. Fig. 1 shows our security approach used in sender and receiver FlexRay nodes. The sender node uses "encrypt-and-MAC" security approach, where the 128-bit message (64-bit ECU message plus 64-bit counter) is encrypted in parallel with HMAC computation using separate 128-bit secret keys. The counter is concatenated with the *plaintext* message to defend against replay attacks. The AES encryption module generates 128-bit *ciphertext* (CT) and HMAC module generates 256-bit MAC (*message digest*). The CT and SHA-3 based HMAC are concatenated and sent to receiver node via FlexRay bus. At the receiver node, the CT is decrypted by the AES decryption module which recovers the original plaintext message. This plaintext is used to compute a *local* HMAC by the SHA-3-based HMAC module at the receiver. The local HMAC is compared with the received HMAC to confirm the integrity of the received message. If the received message has lost its integrity, then the message is retransmitted.

We have used advanced encryption system (AES-128) to provide confidentiality, and secure hash algorithm-3 (SHA-3) based message authentication code to provide entity and message authentication. The basis of AES security is its robustness to brute force attacks as the key space of AES-128 is 3.4×10^{38} keys. Even at a sustained rate of 1 terakeys/second, it would take 1019 years to exhaust this key space. SHA-3 provides 128-bit security level for collision attacks and 256-bit security strength for pre-image and second pre-image attacks. Additionally, to strengthen the security primitives, the secret keys for AES and HMAC are stored in secure tamper resistant memory [9] and are refreshed deterministically over time by participating ECUs. Hence, our approach is resistant to security attacks described in Section III-A.

Furthermore, the dependability feature incorporated into the ECU makes our ECU robust against the fault injection attacks like differential fault analysis attack [5]. This is due to the resilience of our ECU architecture against the soft-errors caused by the transient faults and the fault injection attacks injects the transient faults that causes the soft-error in the ECU.

Dependability: The dependability requirements as stipulated in ISO 26262 [15] require that at least one critical fault must be tolerated by automotive applications without loss of functionality. The BED leverages FT using redundant multi-threading (FT-RMT) to provide dependability. Redundant multi-threading (RMT) uses two threads–a master thread and a slave thread, which execute same operations with the same data set. In FT-RMT, the results of master and slave threads' operations are compared at the end of computation by the master thread. If there is an error, then recomputation is carried out on both the threads. The rationale behind using recomputation after error is that recomputation rectifies soft errors caused by transient faults. FT-RMT can tolerate one permanent fault and multiple soft errors, and therefore adheres to the dependability requirements specified in ISO 26262 standard.

IV. Proposed Secure and Dependable Multiprocessor System-on-Chip based ECU Architectures

Cybercars integrate a multitude of microcomputer units as ECUs to implement different automotive functions. As discussed in Section I, the security features provided by various contemporary standards [15] are often implemented in dedicated inflexible hardware and are not adaptive to evolving nature of security attacks. Additionally, future automotive ECUs may require high computational power to integrate newly emerging cybercar applications and services. To overcome these limitations in prior ECU designs, we propose flexible and scalable ECU architectures that simultaneously integrate security and dependability primitives with low resources and energy overhead.

A. Generalized Version of the Proposed MPSoC-based ECU Architecture

Fig. 2 shows the generalized internal architecture of the proposed MPSoC-based ECU. The ECU consists of an ARM (Advanced RISC Machine)-based application processor as the main processor and one or more application-specific co-processors. This application processor provides interface to the in-vehicle networks (e.g., CAN, CAN FD, FlexRay, LIN, MOST, etc.), external sensors, other ECUs, and gateways. Furthermore, this application processor executes control algorithms, performs data aggregation from various sensors, and outsources computationally intensive applications to the application-specific co-processors. The application-specific co-processor performs compute-intensive applications like image, audio, and video processing. The application-specific co-processor could be a digital signal processor (DSP), or crypto processor that carries out asymmetric and symmetric cryptographic operations, or Viterbi processor for the realization of maximum-likelihood decoding of convolution codes, etc. The application processor and co-processor communicates via advanced system bus (ASB) or advanced high-

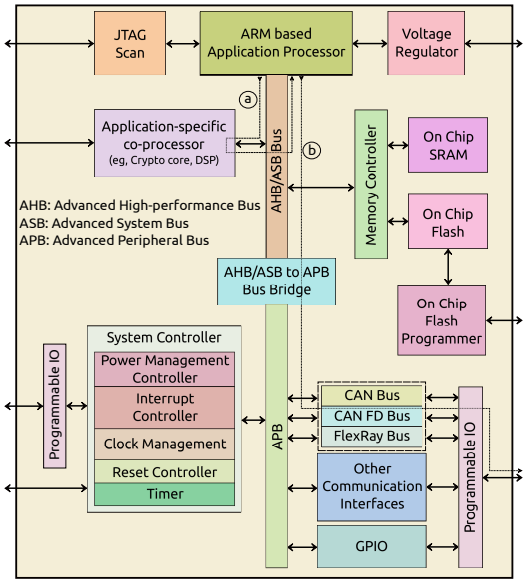

Fig. 2: *A basic MPSoC-based ECU architecture.*

performance bus (AHB) [2]. The application processor of one ECU can communicate with the application processor of another ECU through in-vehicle networks (e.g., CAN, CAN FD, or FlexRay).

During normal operation, the application processor collects data from the sensors. If this data is to be sent to another ECU, the application processor first sends the data to the co-processor via high-speed AHB. The co-processor embeds the security primitives into the data thus creating a *secure data* and sends it back to the application processor. This is represented by path **"a"** in Fig. 2. The *secure data* is send to another ECU via path **"b"**.

In our work, we have used two types of application-specific co-processor viz., GPGPU and programmable logic array. Based on the type of co-processor used, we have named the architecture as GED (refer Section IV-B) or RED (refer Section IV-C). These processors are responsible to provide the cryptographic services, such as confidentiality, message authentication, and message integrity. Our work does not focus on the ECU authentication service. SRAM-based physical unclonable functions [6] is one solution which can be deployed for ECU authentication.

We try to meet three goals in our ECU design. First design goal is to provide security services to the ECU. In order to provide security services to ECU, we implement the cryptographic algorithms in the co-processor. Second design goal is to provide dependability (or fault tolerance). We leverage FT based on redundant multi-threading (FT-RMT) for GED and dual modular redundancy based FT technique for RED. Finally, to assess whether our ECU performs its tasks without violating the real-time deadline of the tasks, we create a timing model of a SBW automotive subsystem that uses our ECU to provide steering functionality to the vehicle. This timing model is discussed in detail in Section V.

In subsequent subsections, we will discuss the internal architecture of the GED and RED.

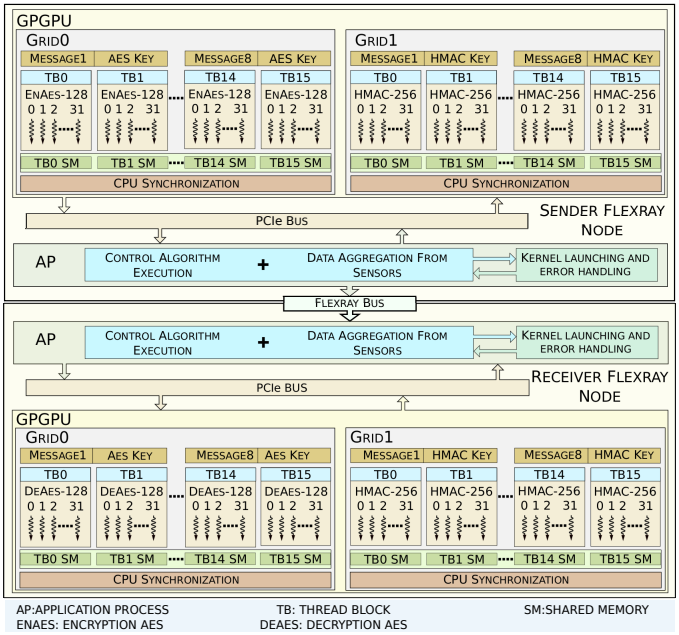

Fig. 3: *GPGPU-based ECU design architecture.*

B. GPGPU based ECU Architecture

Fig. 3 depicts the detailed GED architecture which has the same high level architecture as in the generalized version discussed in Section IV-A. Here, an ARM-based application processor (AP) is the master processor and a GPGPU-based processor is the slave processor (or co-processor) that communicates with AP via a high speed internal bus like AHB. The AP provides interface to the in-vehicle networks, external sensors, other ECUs, and gateways. Furthermore, the AP executes control algorithms, performs data aggregation from various sensors, and outsource computationally intensive applications (e.g., image, audio and video processing, cryptographic operations, etc.) to the GPGPU-based co-processor. We implement FT symmetric cryptographic primitives in GPGPU-based co-processor to provide security and dependability feature to the ECU. In addition, we have used this fault-tolerant (FT) cryptographic primitives implementation as an example use case for comparing the temporal performance and energy consumption of our proposed ECUs, viz., BED, GED, and RED.

Security: The cryptographic module with AES-128 encryption and SHA-3-based HMAC are implemented in GPGPU-based co-processor. The functional architecture of GED is identical to that of BED (refer Section III). However, the GED processes a batch of eight 128-bit messages at once unlike the BED which processes a single 128-bit message at a time. We adopt this batch processing mechanism to utilize the massive computational power of GPGPU and to enhance the throughput of the cryptographic module. Furthermore, the implementation in GPGPU exploits the largely byte-parallel operations of AES-128 and lane-parallel operations of SHA-3. Our implementation adopts parallel granularity of byte-per-thread for AES-128. Each byte of AES-128 is mapped to a thread in GPGPU. A thread-block with number of threads

equals to warp size of 32 is used to compute a complete AES-128 encryption/decryption. All threads in one warp are executed in a single instruction multiple data (SIMD) fashion. For SHA-3, parallel granularity of 8-byte (a *lane*) per thread is used. A thread-block with 32 threads is used to compute complete SHA-3-based HMAC. Furthermore, frequently accessed S-boxes, round constants, and other index constants used in AES-128 and SHA-3 are stored on the on-chip shared memory of GPGPU to ensure fast memory access.

At the sender FlexRay node, eight plaintexts (ECU messages) are processed at once. For NFT mode of operation, the processing is carried out in GPGPU by launching 16 thread-blocks. Eight of these thread-blocks are used for AES-128 computation and eight are used for HMAC computation. Each thread-block processes one ECU message. AES-128 and HMAC are computed in parallel thread-blocks because they are independent operations. However, at the receiver FlexRay node, HMAC computation requires the output of AES computation. Hence, at first, eight thread-blocks are launched to compute AES-128 (decryption) to recover plaintexts of the eight ciphertexts received from the sender node. Then, eight new thread-blocks are launched for HMAC computation of the recovered plaintexts.

Dependability: The design leverages FT-RMT (refer Section III) to provide resilience against soft errors occurring due to transient faults. The FT-RMT is achieved by using redundant thread-blocks. Each thread-block, and its redundant counterpart, performs a complete AES (or HMAC) computation. In the FT-RMT mode (Fig. 3), eight ECU messages are processed using 32 GPGPU thread-blocks: sixteen of these thread-blocks are used for AES-128 and sixteen are used for HMAC computation. In each group of 16 thread-blocks, eight are master thread-blocks and eight are redundant thread-blocks. Each master and redundant thread-block pair processes one ECU message. The results obtained from the master and its redundant thread-block are compared to detect computational errors. The thread-blocks must be synchronized to compare the results. The synchronization is carried out by employing a CPU synchronization technique. If computational errors (soft errors) are detected, then recomputation is conducted.

In addition to error resilience, the fault tolerance provides resistance against the fault attacks that tries to inject soft-errors in the operating ECUs.

C. Reconfigurable ECU Architecture

The high-level architecture of RED is similar to that of generalized MPSoC architecture discussed in Section IV-A. The ARM-based AP has same set of functionalities for both RED and GED. In RED, the AP outsources compute-intensive applications to an FPGA-based co-processor. Fig. 4 shows the internal architecture of RED.

Security: The cryptographic module is implemented in an FPGA-based co-processor and provides three security services: confidentiality, message integrity, and authentication. The FPGA-based co-processor computes AES encryption and

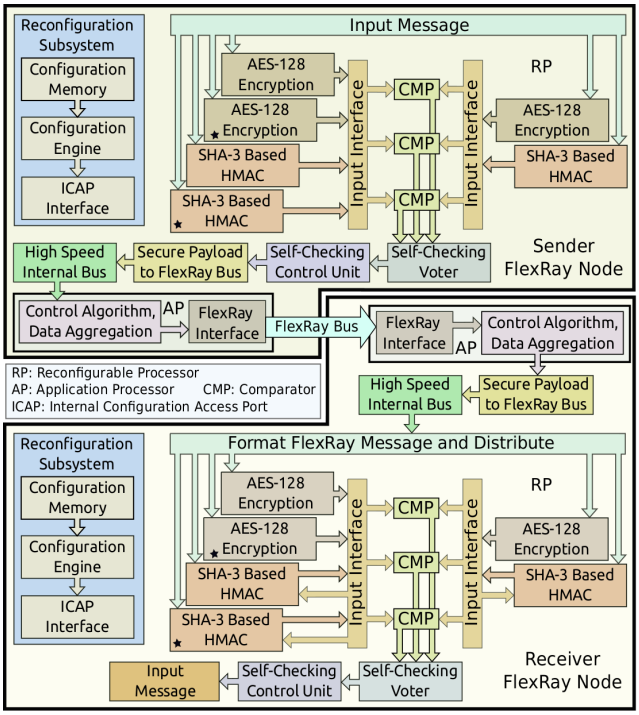

Fig. 4: *Reconfigurable ECU design architecture.*

HMAC of the ECU message at the sender node. The co-processor then relays the concatenation of the CT and the *message digest* to the AP which then transmits it to the receiver node via FlexRay bus. The receiver AP then relays the received concatenation to its FPGA-based co-processor. The co-processor, first, decrypts the CT to recover the original plaintext and then computes HMAC of the plaintext to generate *local message digest*. The local message digest is compared with the received message digest to check the integrity of the received message. If the message has lost its integrity, then the message is retransmitted.

Dependability: The cryptographic module implemented in FPGA is resilient to multiple transient faults and one permanent faults. FT is provided by a combination of three methods: dual modular redundancy (DMR) with an extra spare module (marked by * in Fig. 4), Berger code based totally self-checking (TSC) combinatorial circuits [11], and dynamic partial reconfigurability of Xilinx Automotive Spartan-6 FPGA [7]. This FT mechanism is named as FT using self-reconfiguration in dual modular redundant system (FT-SR-DMR). DMR is used to detect errors in computation; TSC design method is employed to design a combinatorial circuit that flags itself as erroneous in case of faults; and dynamic partial reconfiguration is exploited to heal faulty modules. At both sender and receiver nodes, AES and HMAC modules in DMR computes CT and message digest, respectively, from the ECU message. The results of redundant modules are compared by comparators in triple modular redundancy (TMR). Comparators are implemented in TMR because the comparator circuit is simple and has a smaller footprint. Furthermore, TMR helps to localize the faulty comparator whenever there is fault in the comparator logic. The results of

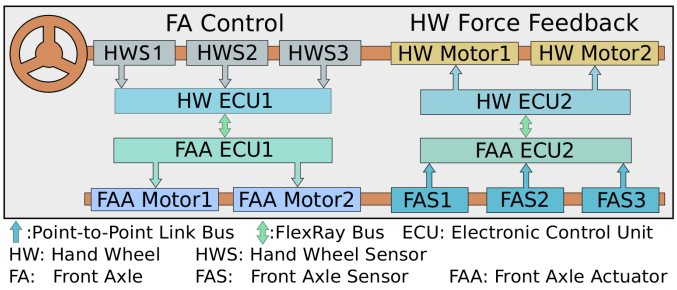

Fig. 5: *SBW architecture.*

comparators are checked by a Berger code based TSC voter. The voter sends its voting result to a self-checking control unit (SCCU). The SCCU generates the control signals for the cryptographic module and stores the result of recent AES and HMAC in a buffer memory. Whenever there is error in AES (or HMAC) computation, the SCCU activates the spare AES (or HMAC) module. The spare module(s) then computes the results corresponding to the input for which there was an error. The results of the spare module is then compared with the results of regular AES (or HMAC) module that were stored in buffer to localize the faulty module. The SCCU then activates the *reconfiguration subsystem* which performs partial reconfiguration of the faulty module. The cryptographic module operates with the spare module(s) during the reconfiguration period. The rationale for using the spare module during reconfiguration is that the reconfiguration takes longer time (in terms of tens of millisecond) and the cryptographic module must be functional during this period to fulfil the security and dependability requirements of automotive CPS.

V. Modelling and Analysis of a Steer-by-Wire Subsystem

This section expands on the timing model of a SBW subsystem that leverages our purposed ECUs to incorporate security and dependability. We use this timing model to compute the quality of service (QoS) and behavioral reliability of the SBW subsystem.

A. Steer-by-Wire Operational Architecture

In an SBW subsystem, heavy mechanical steering column is substituted by electronic systems to reduce vehicle weight. This eliminates the risk of steering column entering into the cockpit in the event of a crash. The SBW subsystem provides the same functionalities as conventional steering column: front axle control (FAC) and hand-wheel (HW) force feedback. The SBW architecture is depicted in Fig. 5. This paper focuses only on the FAC part to compute response time and error resilience of the FT approaches used in our proposed ECUs. Furthermore, the SBW subsystem is made FT by using redundant ECUs, sensors, and actuators. Point-to-point links connect ECUs to sensors and ECUs to actuators. For ECU-to-ECU connection, we experiment on all three commercial automotive buses: CAN, CAN FD, and FlexRay. We evaluate and present a comparison of the SBW *response time* and *error resilience* when using these three buses. The operation of our SBW subsystem is same as in [13], however, our SBW subsystem leverages our proposed ECU architectures.

B. Timing Model of SBW to Compute the QoS and Behavioral Reliability

The *end-to-end delay/response time* (τ_r) is the delay between the driver's request at the HW and the corresponding response at the front axle actuator (FAA). τ_r is regarded as a QoS metric but can also be interpreted as a dependability metric that impacts automotive safety and reliability if it exceeds a critical threshold value τ_r^{max}, which is determined by automotive original equipment manufacturers (OEMs). Furthermore, the probability that the worst-case response time is less than the critical threshold is termed as *behavioral reliability*. In the following, we analytically model the response time for the SBW subsystem and error resilience of our FT approaches.

Response time (τ_r) is modeled as the sum of pure delay (δ_p), mechatronic delay (δ_m), and sensing delay (δ_s), as, $\tau_r = \delta_p + \delta_m + \delta_s$. The mechatronic delay is introduced by the actuators (electric motor in our case). The sensing delay is the delay during the interaction of application processor of ECU with the sensors. The sensing and mechatronic delays are bounded by a constant value of $3.5\,ms$ [8]. For our secure and dependable approach, the pure delay (δ_p) includes ECUs' computational delay for processing the control algorithm (depends on the execution time of application processor), computational delay for processing the incorporated security and dependability primitives (depends on the execution time of the co-processor), and transmission delay including bus arbitration (depends on the type of in-vehicle network used like CAN or CAN FD). Mathematically, pure delay (δ_p) for our FAC function can be written as,

$$\delta_p = rcc1 \cdot \delta_{hw}^{ecu1} + rtc \cdot \delta_{bus} + rcc2 \cdot \delta_{faa}^{ecu1}) \leq \delta_p^{max}, \quad (1)$$

where δ_{hw}^{ecu1} and δ_{faa}^{ecu1} denote the computation time at HW-ECU1 and FAA-ECU1, respectively; δ_{bus} represents the transmission time for a message on an in-vehicle bus (CAN, CAN FD, or FlexRay) from HW-ECU1 to FAA-ECU1; $rcc1$ and $rcc2$ represent the number of recomputations that are needed to be done at HW-ECU1 and FAA-ECU1, respectively, to rectify soft errors; rtc represents the number of retransmissions required for an error-free transmission of a secure message over in-vehicle bus; and δ_p^{max} represents maximum allowable δ_p. According to Wilwert et al. [16], with a minimum tolerable QoS score of 11.13, the critical limit for pure delay δ_p^{max} is $35\ ms$, beyond which the vehicle becomes unstable and risks the driver's safety.

δ_{hw}^{ecu1} and δ_{faa}^{ecu1} are calculated as the sum of execution time of application processor, execution time of co-processor, and bus time of AHB and APB bus (refer path "a" and "b" in Fig. 2). Since the co-processor is executing the computationally intensive cryptographic primitives, the execution time of the co-processor is far greater than the sum of bus times and execution time of application processor. Therefore, we use the execution time of the co-processor as the δ_{hw}^{ecu1} and δ_{faa}^{ecu1}.

VI. Results

In this section, we present our experimental set up and evaluation results comparing timing analysis, energy analysis,

TABLE I: *Performance and energy results for BED, GED, and RED architectures.*

In-vehicle Node	Operational Mode	BED Implementation			GED Implementation			RED Implementation		
		FT Mode	Time (μs)	Energy (μJ)	FT Mode	Time (μs)	Energy (μJ)	FT Mode	Time (μs)	Energy (μJ)
Sender Node	NFT	none	189	9.661	none	99.50	4.674	none	4.90	2.170
	FT	FT-RMT	207	10.581	FT-RMT	112.75	5.297	FT-SR-DMR	6.53	6.040
Receiver Node	NFT	none	184	9.406	none	102.20	4.801	none	9.00	3.996
	FT	FT-RMT	203	10.337	FT-RMT	115.42	5.422	FT-SR-DMR	9.63	9.831

response time, and QoS and behavioral reliability of the SBW subsystem with different in-vehicle buses, viz., CAN, CAN FD, and FlexRay.

A. Experimental Setup

Baseline Design Implementation: We have implemented the BED (Section III) on *NXP quad-core iMX6Q SABRE* development board which has a 32-bit *Cortex-A9* CPU core running *Ubuntu 14.04.4 LTS* at 396 MHz clock speed. The security and dependability primitives are coded in *C*. *OpenMP* is leveraged to provide FT-RMT.

GPGPU based ECU Implementation: We have implemented the security and dependability primitives of GED architecture on NVIDIA's Jetson TK1 GPGPU. The NVIDIA's Jetson TK1 GPGPU has *192 CUDA cores*, *1 streaming multi-processor*, CUDA capability of 3.2, and runs at 852 MHz clock speed. The application processor is ARM *Cortex-A15*. The FT cryptographic modules are coded in CUDA 6.5 using C.

Reconfigurable ECU Implementation: We have implemented the security and dependability primitives of RED architecture in *automotive grade Spartan-6* FPGA. The FT cryptographic modules are coded in *Verilog HDL* in *Xilinx ISE 14.7*. We use *ModelSim* for the functional verification of the design. The total power consumption (both static and dynamic) is obtained through *XPower Analyzer* packaged with Xilinx ISE 14.7 suite. We use power and latency to compute the energy consumed by the FT cryptographic module.

Vector CANoe based Setup: We have simulated our SBW subsystem in *Vector CANoe 8.5* for different in-vehicle buses, viz., CAN, CAN FD, and FlexRay. We have used the following settings for in-vehicle buses: CAN settings: baud rate = 1 Mbps, payload size = 8-bytes; CAN FD settings: baud rate = 3 Mbp, payload size = 64-bytes; FlexRay setting: mixed mode of operation, baud rate = 10 Mbps, payload size = 256-byte. We have used CAPL [1] (CAN Access Programming Language) to implement the SBW functions on ECUs.

Operational Parameters: For our SBW subsystem, we assume the steering wheel sensor sampling rate to be fixed at 420 Hz, that is, $\delta_s = 2.38\,ms$. For the BED architecture, the ECU operates at 396 MHz with an operational current of *36 mA* and the operational voltage of *1.42 V*. For the GED architecture, the operating voltage is 0.87 V and the operating current is 54 mA. For the RED architecture, the ECU operates at 50 MHz.

B. Evaluation Results

Timing Analysis: In real-time automotive CPS, system response times must adhere to strict deadlines. Our evaluations demonstrate that the execution times of our proposed ECU architectures are in the range of microseconds, which conform to the real-time constraints of the SBW subsystem (refer Section V-A). Furthermore, results indicate that our proposed ECU architectures attain significant speedup as compared to a conventional ECU architecture (BED architecture). Table I shows execution time and energy consumption profile for processing one ECU message for BED, GED, and RED architectures. We evaluate our architectures with both NFT and FT operational modes. Results reveal that FT features incur 37% overhead, on average, on temporal performance.

The comparison between NFT GED and NFT BED reveals that the NFT GED is $1.84\times$ faster than the NFT BED. Similarly, the FT GED is $1.79\times$ faster than the FT BED. This is because the GED exploits the massive computational power of GPGPU to perform cryptographic operations. However, the RED has better timing performance than the GED. The NFT RED and the FT RED attain a speedup of $15.83\times$ and $14.62\times$ over the NFT GED and the FT GED, respectively. Additionally, the comparison between RED and BED reveals that the NFT RED and the FT RED provide a speedup of $29.5\times$ and $26.4\times$ over the NFT BED and the FT BED, respectively. The RED is able to achieve this high speedup because of its high-performance parallel hardware architecture which is implemented in reconfigurable logic (FPGA). These speedup values are calculated as the average of speedups at the sender and the receiver ECUs. For each of the sender and the receiver ECUs, the measurements are averaged over 100 runs.

Energy Analysis: Energy efficiency is an important metric for automotive CPS as it implies greater fuel-efficiency for combustion engine vehicles and longer battery lifer for hybrid and electric vehicles. All of our proposed ECU architectures are energy-efficient. Table I shows that the GED yields better energy efficiency than the BED. The NFT GED and the FT GED consumes $2\times$ and $1.95\times$ less energy than the NFT BED and the FT BED, respectively. This is because of the energy efficiency of the GPGPU architecture and less execution time for the GED than that of the BED. When comparing RED and BED, we observe that the RED offers $3.4\times$ and $1.5\times$ improvements in energy efficiency over the BED for NFT and FT operational modes, respectively. Results also indicate that the NFT RED is $1.68\times$ more energy-efficient than the NFT GED. This is because the execution time for the NFT RED is $15.83\times$ lower than that of the NFT GED. However, the FT GED is $1.47\times$ Energy-efficient than the FT RED. The FT GED attains this energy savings because the FT technique (FT-SR-DMR) employed in FT RED has high static power consumption while the FT GED has significantly lower static power consumption.

QoS and Behavioral Reliability: We conduct experiments to determine the impact of using different in-vehicle buses on the QoS and behavioral reliability of the SBW subsystem leveraging our proposed ECU architectures. Table II presents the end-to-end delay/response time of our SBW subsystem when using various in-vehicle buses in combination with different ECU architectures. Results manifest that CAN FD and FlexRay are better alternatives to conventional CAN bus as they provide higher bandwidths and lower latencies. In order to deliver the *48-bytes payload*, CAN FD takes $18.79\times$ less time than CAN bus. FlexRay offers even less transport time, that is, the payload transport time of FlexRay is $32.36\times$ lower than that of CAN bus and $1.5\times$ lower than CAN FD bus. Moreover, FlexRay provides redundant communication channels which offers FT communication. When comparing the response time of the SBW subsystem, the system with ECUs integrating RED architecture yields $3.88\times$ and $2.53\times$ better response time than that of the system with ECUs leveraging BED and GED architectures, respectively. This is because the execution time of the FT cryptographic module in RED is much less than that of the BED and the GED architectures.

In addition to bus latency and response time analysis, we orchestrate experiments to quantify maximum number of allowable recomputations at SBW ECUs to yield error-free results subject to the critical pure dealy $\delta_p^{max} = 35\,ms$ and δ_{bus} (or bus latency) as shown in Table II. Based on the aforementioned δ_p constraint (refer Eq. 1 in Section V-B), we calculate $(rcc1 - 1) + (rcc2 - 1)$ with $rtc = 2$ which gives the number of tolerable faults at HW-ECU1 and FAA-ECU1. Results reveal that the ECUs with RED architecture can tolerate up to $3,459$ faults with one transmission error while the GED and the BED architectures can tolerate 231 and 127 faults, respectively, with one transmission error. It is apparent that the RED is more robust to soft errors since it can tolerate $27.23\times$ and $14.97\times$ more faults than the BED and the GED, respectively.

TABLE II: *Response time (in ms) of the SBW subsystem.*

In-vehicle Bus	Operational Mode	BED	GED	RED
CAN latency = 0.74 ms	NFT	4.81	4.64	4.45
	FT	4.85	4.66	4.45
CAN FD latency = 0.12 ms	NFT	0.49	0.32	0.13
	FT	0.53	0.34	0.13
FlexRay latency = 0.05 ms	NFT	0.42	0.25	0.06
	FT	0.460	0.27	0.06

VII. Conclusion

In this paper, we have proposed and implemented two novel ECU architectures, viz., GED and RED, for real-time automotive CPS. The salient features of RED and GED are: (1) simultaneous integration of security and dependability primitives while adhering to stringent real-time constraints of automotive CPS; (2) the ability to perform compute-intensive applications, such as cryptography and audio, video, graphics, and image processing, in an energy-efficient manner; (3) the flexibility and scalability rendered by reprogrammability that enable upgrading the architectures to incorporate new applications in future; and (4) the resistance of the ECU against fault injection and analysis attacks.

Furthermore, we have quantified and compared temporal performance, energy, and error resilience of our proposed ECU architectures for a SBW case study over CAN, CAN FD, and FlexRay in-vehicle networks. Hardware implementation results reveal that RED and GED can attain a speedup of $31.7\times$ and $1.8\times$, respectively, while consuming $1.75\times$ and $2\times$ less energy, respectively, than the contemporary ECU architectures. Furthermore, RED and GED can tolerate $27.23\times$ and $1.9\times$ more transient faults, respectively, than the contemporary ECU architectures.

VIII. Acknowledgement

This work was supported by the National Science Foundation (NSF) (NSF-CRII-CPS-1564801). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the NSF.

References

[1] *Programming with CAPL*, Dec 2004.

[2] ARM. Advanced microcontroller bus architecture (amba), Feb 2016.

[3] M. Baleani, A. Ferrari, L. Mangeruca, A. Sangiovanni-Vincentelli, M. Peri, and S. Pezzini. Fault-tolerant platforms for automotive safety-critical applications. In *ACM CASES*, San Jose, California, Nov 2003.

[4] E. Beckschulze, F. Salewski, T. Siegbert, and S. Kowalewski. Fault handling approaches on dual-core microcontrollers in safety-critical automotive applications. In *ISoLA*, Porto Sani, Greece, Oct 2008.

[5] E. Biham and A. Shamir. *Differential fault analysis of secret key cryptosystems*, pages 513–525. Springer Berlin Heidelberg, 1997.

[6] M. Cortez, A. Dargar, S. Hamdioui, and G. J. Schrijen. Modeling sram start-up behavior for physical unclonable functions. In *2012 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–6, Albuquerque, New Mexico, Oct 2012.

[7] J. Emmert, C. Stroud, B. Skaggs, and M. Abramovici. Dynamic fault tolerance in FPGAs via partial reconfiguration. In *IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 165–174, Napa Valley, California, Apr 2000.

[8] K. Klobedanz, C. Kuznik, A. Thuy, and W. Mueller. Timing modeling and analysis for autosar-based software development - a case study. In *2010 Design, Automation Test in Europe Conference Exhibition*, pages 642–645, Dresden, Germany, Mar 2010.

[9] O. Kömmerling and M. G. Kuhn. Design principles for tamper-resistant smartcard processors. In *USENIX Workshop on Smartcard Technology on USENIX Workshop on Smartcard Technology*, page 2, Berkeley, California, 1999.

[10] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage. Experimental security analysis of a modern automobile. In *IEEE Symposium on Security and Privacy*, pages 447–462, Berkeley, California, May 2010.

[11] S. Kundu and S. M. Reddy. Embedded totally self-checking checkers: A practical design. *IEEE Design Test of Computers*, 7(4):5–12, Aug 1990.

[12] C.-W. Lin and A. Sangiovanni-Vincentelli. Cyber-security for the controller area network (CAN) communication protocol. In *International Conference on Cyber Security (CyberSecurity)*, pages 1–7, Washington, DC, Dec 2012.

[13] A. Munir and F. Koushanfar. Design and performance analysis of secure and dependable cybercars: A steer-by-wire case study. In *IEEE Annual Consumer Communications Networking Conference CCNC*, Las Vegas, Nevada, Jan 2016.

[14] D. K. Nilsson, U. E. Larson, and E. Jonsson. Efficient in-vehicle delayed data authentication based on compound message authentication codes. In *IEEE 68th Vehicular Technology Conference*, pages 1–5, Calgary, BC, Sep 2008.

[15] R. Soja. Automotive security: From standards to implementation. Technical report, Freescale, 2014.

[16] C. Wilwert, Y. Song, F. Simonot-Lion, Loria-Trio, and T. Clement. Evaluating quality of service and behavioral reliability of steer-by-wire systems. In *IEEE ETFA*, Lisbon, Portugal, Sep 2003.