IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 14, 2020, accepted December 2, 2020, date of publication December 10, 2020,
date of current version December 22, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3043750

Towards Dynamic and Partial Reconfigurable Hardware Architectures for Cryptographic Algorithms on Embedded Devices

ARKAN ALKAMIL, AND DARSHIKA G. PERERA, (Senior Member, IEEE)
Electrical and Computer Engineering Department, University of Colorado at Colorado Springs, Colorado Springs, CO 80918, USA
Corresponding author: Darshika G. Perera (darshika.perera@uccs.edu)

ABSTRACT In the era of IoT, embedded systems are becoming the cornerstone of many IoT related applications, such as smart cars and wearable devices. However, embedded devices have numerous constraints and requirements, including stringent area and power, reduced cost and time-to-market, and increased speedup. Furthermore, these applications are becoming increasingly compute/data-intensive requiring more processing power. Also, especially for IoT related applications, security is another major issue in resource-constrained embedded devices. Although cryptographic algorithms are widely used to ensure the security of these applications, commonly used ones, such as AES, are unsuitable for highly constrained embedded devices, due to their sheer complexity. Hence, several lightweight cryptographic algorithms were proposed in the literature that might be better suited for embedded devices. From these, SPECK and SIMON, introduced by NSA, are the two most popular ones. Another important challenge is how to incorporate the cryptographic algorithms in to embedded devices, efficiently and effectively, without compromising the integrity of the compute/data-intensive applications running on these small-footprint devices. Our previous analysis demonstrated that FPGAs are currently the best avenue to support compute/data-intensive applications running on resource-constrained embedded devices, due to FPGA's many attractive traits, including, post-fabrication reprogrammability, dynamic and partial reconfiguration capabilities, and reduced time-to-market. Also, FPGAs can be utilized to provide several advantages/features required for the embedded device's security, such as cryptographic algorithm agility, algorithm upload, algorithm modification, and resource efficiency. In this research work, we introduce novel, unique, and efficient dynamic and partial reconfigurable hardware architectures for the most popular SPECK and SIMON algorithms on embedded devices, considering the constraints associated with these devices and the requirements of the applications running on embedded devices. We also introduce unique system-level architectures for our proposed designs. To the best of our knowledge, no similar work exists in the literature that provides dynamic and partial reconfigurable hardware for SPECK and SIMON, and also provides system-level architecture. Our dynamic and partial reconfigurable hardware designs achieve 28% space saving compared to its static reconfigurable hardware, and 59 times speedup compared to its software counterpart.

INDEX TERMS FPGAs, reconfigurable hardware, dynamic and partial reconfiguration, embedded systems, embedded hardware, cryptographic algorithms, SPECK and SIMON algorithms.

I. INTRODUCTION

With the advent of Internet of Things (IoT) era, embedded systems are becoming the cornerstone of many IoT related applications, such as intelligent transportation systems, implantable and wearable medical devices, smart grids, and smart homes [1], [2]. The continuous proliferation of embedded devices into these applications is mainly due to the advancements in embedded hardware and software technologies, which enable creating complex but efficient embedded systems for a given application [3]. Conversely, embedded devices have various constraints and challenges, including stringent area and power limitations, reduced cost and time-to-market requirements, and

The associate editor coordinating the review of this manuscript and approving it for publication was Remigiusz Wisniewski.

increased speed-performance requirements [4], [5]. Furthermore, the IoT related applications running on these devices are becoming increasingly complex (compute/data-intensive), requiring more processing power [4], [6]. In addition, as the complexity and utilization of embedded devices in these applications expands, security is becoming another major issue in these resource-constrained embedded devices [3], [7].

In general, cryptographic algorithms are widely used to ensure the security of IoT related applications [8]. However, the cryptographic algorithms utilized for resource-constrained embedded devices must differ from that of the commonly used ones, since typical cryptographic algorithms require heavy computation load and large memory requirements [2], [9]. For instance, as stated in [10], [11], one of the most popular cryptographic algorithms, Advanced Encryption Standard (AES), is considered to be unsuitable or infeasible for highly constrained embedded devices, due to its sheer complexity. This has opened up the lightweight cryptography domain [9], [11]. Several lightweight cryptographic algorithms were proposed in the literature [9], [10], [12], that could potentially lead to more compact designs on embedded devices than that of the AES cryptographic algorithm. To facilitate this endeavor, the National Security Agency (NSA) also introduced a new family of lightweight cryptographic algorithms, specifically SPECK and SIMON [11]. As stated in [11], both the SPECK and SIMON algorithms could potentially be designed in such a way to have different configurations to process different block sizes and different key sizes; whereas, it was observed in [13], [14] that most of the other existing lightweight cryptographic algorithms were often designed for one security configuration at a time. As a result, the SPECK and SIMON algorithms could be utilized for a variety of applications with diverse security requirements, and could potentially be designed in such a way to integrate into the embedded devices while satisfying the associated constraints.

Then another important challenge is how to incorporate the cryptographic algorithms to embedded devices [2], [3], efficiently and effectively, without compromising the integrity of the compute/data-intensive applications running on highly constrained embedded devices, especially with their stringent area limitations.

Our previous work [15], [16], [17] and analyses [18] illustrated that Field Programmable Gate Array (FPGA) based systems are currently the best avenue to support compute/data-intensive applications/algorithms running on resource-constrained embedded devices. This is mainly because FPGAs comprise many attractive features that are beneficial to support applications/algorithms on embedded devices. For instance, FPGAs provide higher flexibility compared to the Application-Specific-Integrated-Circuits (ASICs) and higher performance compared to the equivalent software running on processors [18], [19]. Unlike ASICs, FPGA's post-fabrication reprogrammability allows post-design optimizations and upgrades in applications. This feature also enables reusing the same chip/FPGA to execute numerous tasks/algorithms, by reconfiguring the on-chip hardware from one task to another as needed. Furthermore, with the dynamic and partial reconfiguration capabilities, parts of the chip/FPGA can be modified, while other parts are still operational. This in turn leads to significant space savings on chip for complex embedded applications [20], [21]. In addition, with FPGAs, time-to-market is reduced, since FPGAs are pre-fabricated, hence, immediately available. Due to these attractive traits, there is a dramatic increase in utilization of FPGAs to support and accelerate many real-time compute/data-intensive applications [15], [22], [23], [57], specifically on resource-constrained embedded devices.

As stated in [7], [24], [25], apart from supporting and accelerating compute/data-intensive applications, FPGAs can also be utilized to improve the security within the embedded systems. As in [26]–[29], the advantages of using FPGAs for the embedded device's security mainly include, algorithm agility, algorithm upload, algorithm modification, and resource efficiency. For instance, embedded systems often require processing multiple and diverse security protocols and standards [7], [24]. This algorithm agility feature [24], [26], which enables switching the cryptographic algorithms during the run-time life of the application, can be provided with dynamic and partial reconfiguration capabilities of FPGAs. Furthermore, FPGA's post-fabrication reprogrammbility and dynamic and partial reconfiguration capabilities can provide [26], [27]: the algorithm upload feature, which allows updating the cryptographic algorithm with a new one at any time, without interrupting the system's operations; and the algorithm modification feature, which enables modifying the cryptographic algorithm or changing the configurations as needed. In addition, most of the embedded security systems use different cryptographic algorithms for different scenarios but not necessarily at the same time [24], [26]. With dynamic and partial reconfiguration capabilities of FPGAs, different cryptographic algorithms can be loaded and utilized as needed, thus saving valuable area on chip of these embedded devices. The above facts illustrate that FPGA is indeed a promising avenue to support cryptographic algorithms (or security mechanisms/primitives), specifically on resource-constrained embedded devices.

Our main objective is to create novel, unique, and efficient FPGA-based dynamic and partial reconfigurable hardware architectures to support cryptographic algorithms on embedded devices, considering the constraints associated with these devices as well as the requirements of the applications running on embedded devices. In this research work, we focus on dynamic and partial reconfigurable hardware architectures for SPECK and SIMON lightweight cryptographic algorithms [13], currently, the most popular algorithms in the lightweight cryptographic domain. In this article, we make the following contributions:

- We introduce novel, unique, and efficient FPGA-based reconfigurable hardware architecture/structure for SPECK. Our reconfigurable hardware structure for

SPECK is created in such a way to be generic, parameterized, and scalable; thus, without changing the internal architecture, our hardware design can be reconfigured to any one the 20 different configurations, in order to perform the encryption and decryption, as well as to process different plaintext/ciphertext blocks with varying sizes and different keys with varying sizes. In this case, the reconfiguration can be done on-the-fly (i.e., dynamically), without interrupting the system's operation and without human intervention. Similarly, in previous work [13], we introduced a unique FPGA-based reconfigurable hardware architecture/structure for SIMON, which can also be dynamically reconfigured to 20 different configurations.

- Next, we introduce novel and unique dynamic and partial reconfigurable hardware architecture for SPECK and SIMON algorithms. Our reconfigurable hardware architecture is created in such a way, so that after processing one cryptographic algorithm (e.g., SIMON), the specific region of the chip is reconfigured dynamically (on-the-fly) and partially to the next cryptographic algorithm (e.g. SPECK) and that algorithm is processed. As a result, the SIMON and SPECK algorithms (with any configurations) can be processed any number of times as needed, without interrupting the system's operations and often without human interventions.
- We also introduce unique and efficient system-level architectures for our aforementioned reconfigurable hardware architectures for SPECK and SIMON algorithms. With the system-level architecture, we create and incorporate unique pre-fetching techniques to reduce the memory access latency of our proposed reconfigurable hardware architectures for SPECK and SIMON algorithms.
- We perform experiments on SPECK and SIMON (with 20 different configurations each) as individual entities. We also perform experiments on dynamic and partial reconfigurable architecture for SPECK and SIMON together. We analyze the execution times, reconfiguration time overhead, resource utilization, reconfiguration space overhead, and speedup. In addition, we investigate and analyze the existing works on dynamic and partial reconfigurable hardware architectures for cryptographic algorithms, and the existing works on FPGA-based hardware architectures for SPECK and SIMON, in the published literature.

From our investigations on the existing works (presented in Sections V.C and V.D), and to the best of our knowledge, no similar work exists in the literature that provides dynamic and partial reconfigurable hardware architectures for SPECK and SIMON lightweight cryptographic algorithms. We also could not find any similar work in the literature that provides reconfigurable hardware architectures for SPECK, which can also be dynamically reconfigured to 20 different configurations, without changing the internal architecture. Previously, in [13], we introduced similar reconfigurable hardware architecture for SIMON with 20 configurations. None of the existing works on SPECK and SIMON proposed system-level architectures, which is imperative for embedded applications in real-world scenarios.

This article is organized as follows: In Section II, we discuss and present the structure and the functionality of SPECK and SIMON lightweight cryptographic algorithms. Our design approach and development platform are discussed and presented in Section III. In this section, we also discuss and present our proposed system-level architecture, our proposed top-level architecture, and dynamic and partial reconfiguration process on Virtex-6 FPGA. Our proposed novel and unique embedded and dynamic reconfigurable hardware architectures for SPECK and SIMON are discussed and presented in Section IV. In this section, we present our novel, unique, customized, and optimized internal architectures for SPECK, which include the key generation function, round function, and encryption and decryption functions. Next, we present the top-level architecture for dynamic and partial reconfigurable hardware for SPECK and SIMON. Our experimental results and analysis in terms of resource utilization, reconfiguration space overhead, reconfiguration time overhead, execution times, and speedups are reported and discussed in Section V. Our analysis on existing works on dynamic and partial reconfigurable hardware architectures for cryptographic algorithms, and our analysis on existing works on FPGA-based hardware architectures for SPECK and SIMON, are also presented in Section V. In Section VI, we summarize our work, conclude, and discuss the future work.

II. BACKGROUND – SPECK AND SIMON ALGORITHMS

The SPECK and SIMON algorithms come from the realm of lightweight cryptography family [11]. Both these algorithms are based on the symmetrical block ciphers, since symmetric keys are employed to encrypt the blocks of data.

Similar to the SIMON algorithm [11], [13], SPECK algorithm can also support ten different configurations based on the block size ($2n$) and the key size (mn). This leads to 20 configurations, 10 for the encryption and 10 for the decryption. In this article, the block size and the plaintext size are used interchangeable. Table 1 presents varying parameter selections for the SPECK and SIMON algorithms. As illustrated in Table 1, both these algorithms can process five different block sizes (in column 1) and each block size comprises a set of keys (in column 2). In this case, the block size (or the plaintext size) is denoted by $2n$, where n is the word size (in column 3), which varies from 16, 24, 32, 48, and 64. The key size is denoted by mn, where m is the key words (in column 4), which varies from 2, 3, and 4.

A. SPECK – STRUCTURE AND FUNCTIONALITY

The SPECK round function for encryption and decryption is denoted by R, as depicted in equations (1) and (2), respectively. As shown, for both the encryption (equation (1)) and decryption (equation (2)), the SPECK round function mainly

TABLE 1. Parameters of SPECK and SIMON algorithms.

block size (2n)	key size (mn)	word size (n)	key words (m)	SPECK Parameters			SIMON Parameters	
				Rot (α)	Rot (β)	Rounds (T)	Constant Seq. (Zj)	Rounds (T)
32	64	16	4	7	2	22	Z_0	32
48	72	24	3	8	3	22	Z_0	36
	96		4			23	Z_1	36
64	96	32	3	8	3	26	Z_2	42
	128		4			27	Z_3	44
96	96	48	2	8	3	28	Z_2	52
	144		3			29	Z_3	54
128	128	64	2	8	3	32	Z_2	68
	192		3			33	Z_3	69
	256		4			34	Z_4	72

uses XOR (⊕) logic operators, addition and subtraction operators, and left-shift (S^j) and right-shift (S^{-j}) circular operators, where j is the number of bits being shifted. For both the equations, the rotation amounts of β and α are 2 and 7 respectively, for the block size equal to 32; and for all other block sizes, these two rotation amounts are 3 and 8, respectively.

$$R(x, y) = ((S^{-\alpha}x + y) \oplus k, S^{\beta}y \oplus (S^{-\alpha}x + y) \oplus k) \quad (1)$$

$$R^{-1}(x, y) = (S^{\alpha}((x \oplus k) - S^{-\beta}(x \oplus y)), S^{-\beta}(x \oplus y)) \quad (2)$$

Figure 1 illustrates the typical structure of the SPECK round function [11]. As shown in Figure 1 as well as from equations (1) and (2), x is the leftmost block (X_i), y is the rightmost block (Y_i), k is one of many round keys (k_0, k_1, …, k_{T-1}), and T is the number of rounds (in column 7). From our previous work [13] and from equations (1) and (2), it is observed that the encryption and the decryption have similar structures/flows with respect to the direction of the shift; however, there are minor differences with respect to the value k_i, which varies in each round.

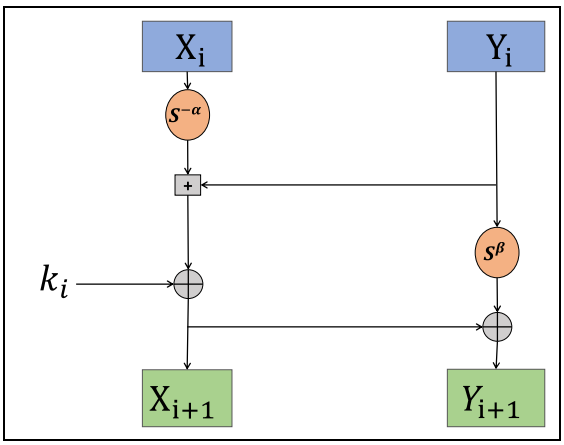

FIGURE 1. Structure of SPECK round function [11].

Figure 2 demonstrates the structure of the SPECK key generation function [11], which generates the round keys from the original key K. The key generation function uses the aforementioned round function to produce the round key (k_i) in each round. In this case, $K = (l_{m-2}, \ldots, l_o, k_0)$ and

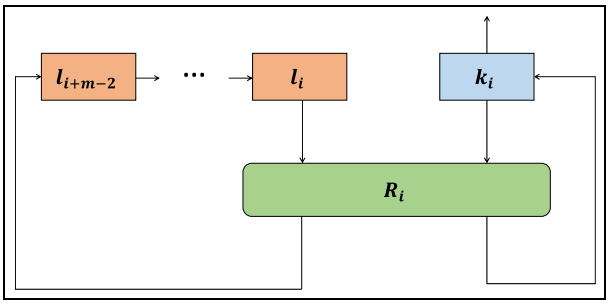

FIGURE 2. Structure of SPECK key generation function [11].

$m = \{2,3,4\}$, then the key generation function can be represented with equation (3), where k_i is the i^{th} round key, for $0 < i < T$:

$$l_{i+m-1} = ((k_i + S^{-\alpha}l_i) \oplus i), \quad k_{i+1} = S^{\beta}k_i \oplus l_{i+m-1} \quad (3)$$

B. SIMON – STRUCTURE AND FUNCTIONALITY

Similar to SPECK, R is considered as the SIMON round function for encryption and decryption as in equations (4) and (5), respectively. The SIMON round function utilizes XOR (⊕) and AND (&) logic operators, and left-shift (S^j) circular operators, where j is the number of bits being shifted.

$$R(x, y, k) = ((S^1(x)\&S^8(x)) \oplus S^2(x) \oplus y \oplus k, x) \quad (4)$$

$$R^{-1}(x, y, k) = ((y, S^1(y)\&S^8(y)) \oplus S^2(y) \oplus x \oplus k) \quad (5)$$

Figure 3 illustrates the structure of the SIMON round function [11]. From equations (4) and (5) as well as from Figure 3, x is the leftmost block (X_{i+1}), y is the rightmost block (X_i), k is one of many round keys ($k_0, k_1, \ldots, k_{T-1}$), and T is the number of rounds (in column 9). Similar to SPECK, for SIMON algorithm, the encryption and decryption have similar structures/flows with respect to the direction of the shift; however, there are minor variations with respect to the value k_i, which varies in each round. In this case, as shown in Figure 3, in encryption, the input block (i.e., plaintext) is

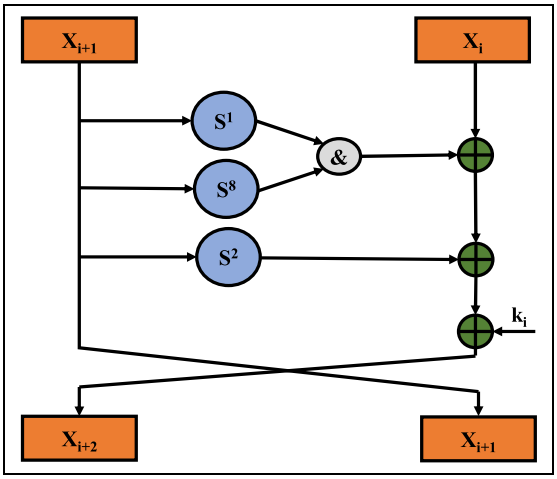

FIGURE 3. Structure of SIMON round function [11].

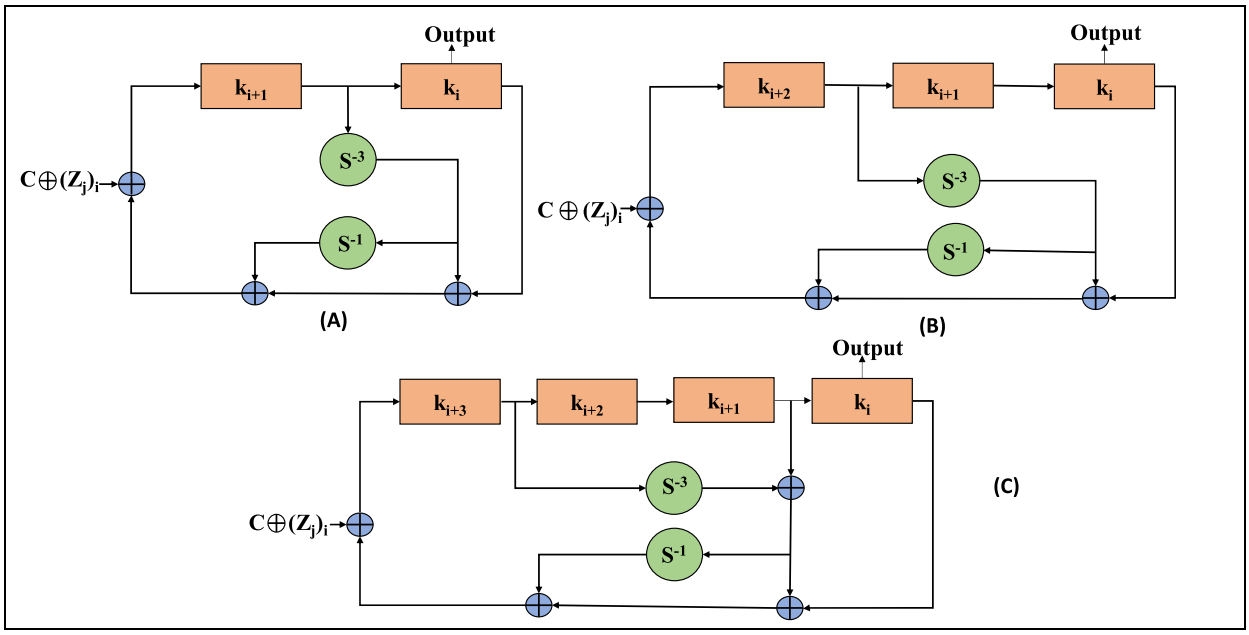

FIGURE 4. Structures of SIMON key expansion function [11].

initially divided into two sub-blocks: leftmost block (X_{i+1}) and rightmost block (X_i). Next, the X_i block goes through 3 XOR-operations to reach the next round as the X_{i+2} block. In decryption, the input block (i.e., ciphertext) is also divided into two sub-blocks; however, unlike the encryption, the leftmost block is X_i and the rightmost block is X_{i+1}, while the rest of the process is similar to the encryption.

Figure 4 demonstrates the structure of the SIMON key expansion function [11], which generates the round keys from the original key K. The key expansion function utilizes the above SIMON round function to produce the round key (k_i) in each round. Although the key expansion process is the same for each round, the structure of the key expansion function increases with the increasing key words (m). As shown in Figures 4(a), 4(b), and 4(c), the structures for key expansion function vary for 2-word ($m = 2$), 3-word ($m = 3$), and 4-word ($m = 4$) keys, respectively. These structures are based on the equation (6). As in equation (6), the key expansion function comprises a parameter known as constant sequence z_j. As in Table 1, there are 5 constant sequences (z_0, z_1,...,z_4), which vary among ten different configurations of SIMON algorithm. For instance, the configurations 48/72 and 48/96 (row 2) have two constant sequences z_0 and z_1. In this case, for $0 < i < (T\text{-}m)$ and for constant sequence (z_i), where j = 0,1,...,4; and for parameters c = *2n-4*, block size = $2n$, round key k, key words m; the round keys (k_{m-1},..., k_1, k_0) can be represented with equation (6).

$$
\begin{aligned}
K_{i+m} &= C \oplus (Z_j)_i \oplus k_i \oplus (1 \oplus S^{-1})S^{-3}k_{i+1}, \quad if\, m = 2,\\
K_{i+m} &= C \oplus (Z_{ji}) \oplus k_i \oplus (1 \oplus S^{-1})S^{-3}k_{i+2}, \quad if\, m = 3,\\
K_{i+m} &= C \oplus (Z_j)_i \oplus k_i \oplus (1 \oplus S^{-1})(S^{-3}k_{i+3} \oplus k_{i+1}), \quad if\, m = 4
\end{aligned}
\tag{6}
$$

As shown in Figures 4(a), 4(b), and 4(c), the key expansion function increments with the key word (m). Let's consider Figure 4(c) for the key expansion function for 4-word key: in this case, the input is divided into m words (k_m *to* k_0) of n bits each. In each round/iteration, the rightmost word k_i moves towards the output, and replaces the leftmost word k_{i+m-1} after going through 3 XOR-operations. In this case, each word moves to the right and replaces the adjacent word on the right. Also, a new key is generated in each round. This process continues until the total number of rounds (T) is completed.

III. DESIGN APPROACH AND DEVELOPMENT PLATFORM

In this research work, we introduce novel, unique, and efficient dynamic and partial reconfigurable hardware architectures for SPECK and SIMON lightweight cryptographic algorithms for embedded devices. We create our unique dynamically reconfigurable hardware architectures as two versions: static reconfigurable hardware (SRH) for SPECK as a separate entity, which can also be dynamically reconfigured to 20 different configurations, without changing the internal architecture; and dynamic reconfigurable hardware (DRH) for SPECK and SIMON together, which can be dynamically and partially reconfigured from SIMON to SPECK and vice versa, by reconfiguring the on-chip hardware from one cryptographic algorithm to another. Although the former can be dynamically reconfigured, we call it SRH, mainly because the partial reconfiguration is not utilized. Both these versions are discussed and distinguished in Section IV. We also create novel and unique embedded software architectures for the SPECK and SIMON algorithms in order to evaluate our embedded reconfigurable hardware architectures.

In our designs, both the software and hardware versions of various operations are implemented using a hierarchical

platform-based design approach to facilitate component reuse at different levels of abstraction, where higher-level functions utilize lower-level sub-functions and operators. Furthermore, we introduce novel, unique, and efficient system-level architectures for our proposed embedded and dynamic reconfigurable hardware designs for SPECK and SIMON. Our system-level architectures are created in such a way to enhance the efficiency of the overall system.

A. EXPERIMENTAL PLATFORM

All our embedded hardware and software experiments are carried out on the Xilinx ML605 development platform [30], which utilizes a Virtex-6 XC6VLX240T-FF1156 FPGA device, built on 40nm CMOS process technology. This development platform includes large on-chip logic resources (37680 slices), 2-MB on-chip block random access memory (BRAM), and 512-MB DDR3-SDRAM external memory to hold large volume of data/results. It enables instantiating MicroBlaze soft processors on chip, and provides onboard configuration circuitry for development purpose. The ML605 board has several external non-volatile memories such as 128-MB Platform Flash XL, 32-MB BPI Linear Flash, and 2-GB Compact Flash, which can be used to hold the configuration bitstreams. It should be noted that we utilized ML605 with Virtex-6 FPGA as a prototyping platform to design, develop, test, and verify our proposed architectures. However, in a real-world scenario, our intention is to execute our DRH design on low-cost small-footprint FPGAs.

Both the static reconfigurable hardware (SRH) and dynamic reconfigurable hardware (DRH) modules are designed in mixed VHDL and Verilog. They are executed on the FPGA (running at 100MHz) to verify their correctness and performance. Xilinx ISE 14.7 and XPS 14.7 are used for the SRH designs. Xilinx ISE 14.7, XPS 14.7, and PlanAhead 14.7 (with partial reconfiguration features) are used for the DRH designs. ModelSim SE and Xilinx ChipscopePro 14.7 are used to verify the results and functionalities of the designs. The software modules are written in C and executed on the 32-bit RISC MicroBlaze soft processor (running at 100MHz) on the same FPGA. Xilinx XPS 14.7 and SDK 14.7 are used to design and verify the software modules. The execution times presented in this article are obtained using the Advanced Extensible Interface (AXI) Timer running at 100MHz [33]. The performance-gain (i.e., speedup) is evaluated using the baseline execution times of software over the improved execution times of hardware.

It should be noted that all our designs, including SRH, DRH, and embedded software on MicroBlaze, are actually implemented on the FPGA, and the real dynamic and partial reconfiguration is performed for DRH design. Furthermore, hardware verification is performed, while these designs are actually running on the chip, which is detailed in Section III.B and Section IV.D. In addition, all the experimental results presented in this article, are obtained while our proposed SRH and DRH designs, and software design are actually running (in real-time) on the FPGA.

B. OUR SYSTEM-LEVEL ARCHITECTURE

In this sub-section, we discuss and present our unique and efficient system-level architectures for both the SRH and DRH architectures for the SPECK and SIMON lightweight cryptographic algorithms. As detailed in [13], in a real-world scenario, the system-level architectures are imperative for the lightweight cryptographic algorithms, especially on embedded devices, since these devices often communicate with the external world; thus, the cryptographic algorithms need to secure these devices from the threats/hackers from the external environments. Customized and optimized system-level architectures provide the necessary peripherals/modules to facilitate this process and also to provide the necessary hardware-software interfaces for both the SRH and DRH architectures for the SPECK and SIMON algorithms. To the best of our knowledge, no similar work exists in the literature that provides system-level architectures for the embedded hardware designs for either SPECK or SIMON algorithms.

Figure 5 demonstrates how our user-designed hardware modules interface with the rest of the system. In this case, as detailed in Section III.C, our user-designed hardware modules consist of either SRH and DRH for SPECK and SIMON algorithms. With the system-level design, for both the SRH and DRH architectures, we incorporate the on-chip BRAM to store the necessary data required to process the SPECK and SIMON algorithms. These on-chip BRAMs support both the single and burst read/write transactions [31]. As illustrated, our user-designed hardware modules (i.e., SRH and DRH for SPECK and SIMON algorithms) communicate with the MicroBlaze soft processor and the on-chip BRAM via the AXI bus [32]. In this case, the AXI bus acts as the glue logic for the system. Our user-designed hardware modules act as the Master when communicating with the BRAM.

With this system-level interfacing, our user-designed hardware modules (both SRH and DRH) typically receive a start signal from the MicroBlaze processor via the AXI bus to: select and execute either the encryption or the decryption process of the cryptographic algorithm, read/write data/results from/to the on-chip BRAM; and hardware modules send a stop signal to the MicroBlaze processor after completing the execution. After sending a start signal, the MicroBlaze processor can be utilized to perform other tasks, until it receives a stop signal from the user-designed hardware module; thus, creating a multi-processor system. After completing the execution, the final results, stored in the on-chip BRAM, are brought to the external HyperTerminal window [42] of a desktop computer via the MicroBlaze processor and RS232 UART (in Figure 5). These final results are compared with the software results for SIMON and SPECK to verify the correctness and functionalities of our proposed hardware designs.

Although our main focus of this article is to introduce static reconfigurable hardware (SRH) and dynamic reconfigurable

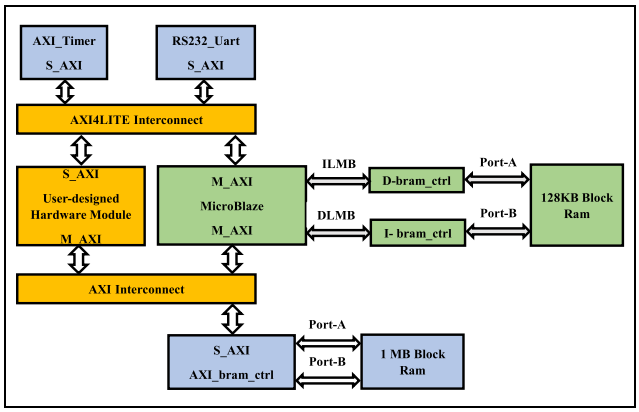

FIGURE 5. System-level architecture for our proposed embedded and dynamic reconfigurable hardware cryptographic algorithms.

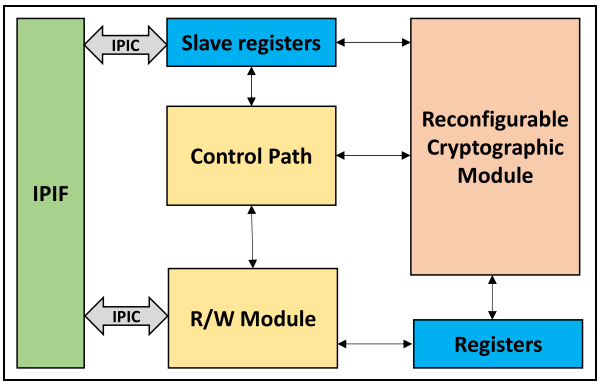

FIGURE 6. Top-level architecture of our proposed user-designed hardware modules.

hardware (DRH) architectures for lightweight cryptographic algorithms, we also create embedded software architectures for the SPECK and SIMON algorithms, mainly to evaluate our proposed reconfigurable hardware designs. As in [13], during our initial embedded software design phase, it was observed that our embedded software architectures for the SMION algorithm could not be executed on the MicroBlaze process due to the limitations of the cache memory, although we utilized the maximum available cache memory of 128KB for the MicroBlaze on ML605 [34]. This is also true for the SPECK algorithm. In this case, our embedded software designs (for both the SPECK and SIMON) need to access several large data arrays (with data elements of varying sizes), which require more memory resources than that of the maximum available cache memory. As a result, we integrate the on-chip BRAM to overcome these memory constraints, while striving to reduce the memory access latency. Furthermore, the integration of the on-chip BRAMs, at the system-level, enhances the efficiency and the flexibility of the embedded software designs at the internal architecture-level as detailed in [13]. Similar outcomes are observed for the embedded and reconfigurable hardware designs, as detailed in Section V.

C. OUR TOP-LEVEL ARCHITECTURE OF PROPOSED RECONFIGURABLE HARDWARE DESIGN

The top-level architecture of our user-designed hardware modules (for SRH and DRH designs) is demonstrated in Figure 6. As depicted, the top-level architecture comprises: the reconfigurable cryptographic module (i.e., the data path designed for the SPECK or SIMON algorithm), the control path module, slave registers, extra internal registers, and Read/Write (R/W) module. In order to simplify the design and routing complexity of the control path module, we design and integrate a unique R/W module to our user-designed hardware. The control path module uses the R/W module to assign the addresses and other control signals required for the R/W operations from/to the on-chip BRAM, whereas the data path module receives/sends data/results from/to the on-chip BRAM using the R/W module. In addition, extra Registers are utilized to buffer the data/results to/from the reconfigurable cryptographic module to avoid any timing and metastability issues, as well as data loss.

As illustrated in Figure 6, the slave registers (also known as software accessible registers) are incorporated to the top-level architecture, in order to establish communication between the user-designed hardware and the MicroBlaze processor, through the AXI Intellectual Property Interface (IPIF), using a set of ports called the Intellectual Property Interconnect (IPIC). In this case, the MicroBlaze processor as well as the user-designed hardware can send/receive certain signals/instructions (such as start/stop signals) via the slave registers and the control path module. Based on these signals, the user-designed hardware module can be configured to perform any one of these tasks at a time: key schedule, encryption, and decryption. The control path module (in Figure 6) monitors and controls the proper operations of the aforementioned tasks and also within suitable timelines.

For both the SRH and DRH architectures, during the encryption process, firstly, the control path module sends the read request signal and the first read data address to the R/W module. Next, the R/W module asserts the essential IPIC port signals to read the data from the on-chip BRAM via the IPIF interface. Then the read data (i.e., the plaintext), is fetched from the BRAM in a single read data transaction mode, and is buffered to the registers via the R/W module. Secondly, the reconfigurable cryptographic module performs the encryption process on the read input data (i.e., the plaintext) and transforms it to the ciphertext (i.e., the write output data). Thirdly, the control path module sends the write request signal and the write data address to the BRAM via the R/W module and the IPIF interface. Next, the write data (i.e., the ciphertext) is buffered to the registers, and then written to the BRAM in a single write data transaction mode. Once the ciphertext is written to the on-chip BRAM, the control path module triggers an encryption complete signal. During the decryption process, similar steps are followed as for the encryption process; however, in this scenario, the read input data is the ciphertext, and the write output data is the plaintext.

D. DYNAMIC AND PARTIAL RECONFIGURATION PROCESS ON VIRTEX-6 FPGA

Typically, FPGA-based reconfigurable hardware designs [35], [36], written in Verilog and/or VHDL, have to undergo a series of steps to fit into the FPGA's available logic, including synthesis, technology mapping, placement and routing, and the final step, bitstream generation, which creates a ''configuration bitstream'' for programming the FPGA.

We have been investigating different FPGA-based reconfiguration methods for embedded devices [37]–[40]. As illustrated [37], [38], [40], [56], the reconfigurable hardware can be divided into two types: static and dynamic. With static reconfigurable hardware (SRH), a full configuration bitstream of an application/algorithm is downloaded to the FPGA at the system start-up, then the chip is configured only once, and often never changed during the run-time life of the application/algorithm. Most of the traditional FPGA-based designs are SRH designs, which typically utilize single context reconfiguration method [35], [36]. Especially with this reconfiguration method [35], [36], [38], in order to execute a different application/algorithm, the corresponding full bitstream has to be downloaded and the entire FPGA has to be reconfigured, which typically requires interrupting the system's operations.

With dynamic reconfigurable hardware (DRH), initially, a full bitstream of an application/algorithm is downloaded to the FPGA, and the on-chip hardware is configured, but is often allowed to change during the run-time life of the application/algorithm, without interrupting the system's operations and also without human interventions. In this case, the reconfiguration can be done autonomously and dynamically (on-the-fly) based on certain stimuli (or parameters) by the system, without any assistance or involvement from a human (or person). With dynamic reconfiguration techniques, we can modify either parts of the chip or the whole chip as needed on-the-fly, can execute numerous applications/algorithms on a single chip by reconfiguring the hardware on chip from one application to another, and can execute large and complex applications on a smaller FPGA, regardless whether these applications fit into the chip or not, by decomposing these into smaller sub-circuits and executing the sub-circuits at different times.

With Virtex-6 FPGA, we can use two different reconfiguration methods [37], [38]: (1) MultiBoot method [41], which allows full bitstream reconfiguration; (2) Partial Reconfiguration method [42], [43], which allows partial bitstream reconfiguration. For our dynamic reconfigurable hardware (DRH) designs, we utilize partial reconfiguration method, since our intention is to reconfigure only some parts of the design/chip, while other parts are still operational.

As stated in [37], [38], [40], the dynamic partial reconfiguration method enables us to reconfigure parts of the chip (or the design) that require modification, while interfacing with the rest of the system that remains operational [42], [43]. This is facilitated by the non-glitching feature of Virtex-6 FPGAs [43], [44]. Figure 7 demonstrates the basic premise of partial reconfiguration method [42], [43]. As illustrated, during the design phase, the logic in the reconfigurable hardware is partitioned into reconfigurable parts versus static parts. With this method, firstly, the FPGA is fully configured with an initial full bitstream for the entire chip. Secondly, the specific parts of the chip/design that require modifications are reprogrammed with new functionalities by loading the corresponding partial bitstreams and reconfiguring those specific parts. In this case, as in Figure 7, the tasks/functions realized in the reconfigurable modules/parts are replaced by the contents of the partial bitstreams, ''without compromising the integrity'' of the application/system running on the rest of the chip [42], [43].

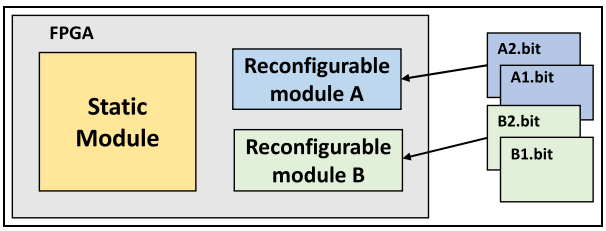

FIGURE 7. Basic premise of partial reconfiguration [42], [43].

For the dynamic and partial reconfiguration, typically, the full and partial bitstreams are stored in the external non-volatile memory [43], and the configuration controller manages the loading of the bitstreams and reconfiguring the chip as needed. These bitstreams can also be stored in an external device such as a desktop computer [58]. The configuration controller can be either a microprocessor or routines (simple finite state machine [42]) programmed into the FPGA. As stated in [42], [43], [59], partial reconfiguration can be done using a wide variety of techniques, one of which is illustrated in Figure 12 (in Section IV.D). Figure 12 (modified from [42], [43]) demonstrates our top-level architecture and system-level setup for the dynamic and partial reconfiguration process on Virtex-6 FPGA. As depicted in Figure 12, in our designs, the full and partial bitstreams are stored in the compact flash (CF) non-volatile memory. For our design, in order to facilitate the in-circuit reconfiguration, the AXI hardware internal configuration access port (ICAP) [45] is instantiated and controlled through the software running on the MicroBlaze processor. In this case, the ICAP module is used to load the partial bitstreams to the FPGA. During the run-time life of the application, the partial bitstreams are fetched from the CF via the MicroBlaze to the ICAP to accomplish the dynamic and partial reconfiguration process.

IV. EMBEDDED AND DYNAMIC RECONFIGURABLE HARDWARE ARCHITECTURES FOR SPECK AND SIMON CRYPTOGRAPHIC ALGORITHMS

In this section, we discuss and present our unique embedded and dynamic reconfigurable hardware architectures for the SPECK and SIMON lightweight cryptographic algorithms.

As briefly mentioned in Section III, we create our unique dynamically reconfigurable hardware architectures as two versions: static reconfigurable hardware (SRH) for SPECK and SIMON as separate entities, and dynamic reconfigurable hardware (DRH) for SPECK and SIMON together.

For the SRH, we introduce unique, customized, and optimized reconfigurable hardware architecture for SPECK in such a way that without changing the internal architecture, our SPECK hardware design can be reconfigured to 20 different configurations, in order to perform encryption and decryption, as well as to process varying block sizes and varying key sizes. In this case, the reconfiguration is performed dynamically (on-the-fly) without interrupting the system's operation and without human intervention, but not utilizing partial reconfiguration. As previous work, in [13], we already introduced similar SRH design for SIMON cryptographic algorithm, which can also be reconfigured to 20 different configurations. Although this version can be dynamically reconfigured, this is called SRH, mainly because the partial reconfiguration is not employed.

For the DRH, we introduce novel and unique dynamic and partial reconfigurable hardware architectures for both SPECK and SIMON in such a way that after processing one cryptographic algorithm (e.g., SIMON), the specific region (i.e., the reconfigurable part) of the chip consisting of the cryptographic algorithm is dynamically (on-the-fly) and partially reconfigured to another cryptographic algorithm (e.g., SPECK), and that algorithm is processed. Then the chip can be dynamically and partially reconfigured back to SIMON (or SPECK) and so on. Hence, using partial reconfiguration, the SIMON and SPECK algorithms (with any configurations) can be processed any number of times as needed, without interrupting the system's operations and often without human interventions.

For both versions, in order to introduce internal architectures for SPECK, first, we investigate and partition the SPECK algorithm into two sub-tasks: round function and the key generation function. Then we create customized and optimized internal hardware architectures for these sub-tasks in such a way that our proposed hardware designs are generic, parameterized, and scalable, as well as highly flexible and reconfigurable. This is analogous to the previously proposed internal architectures of our SIMON algorithm in [13].

For our embedded hardware architectures for SPECK algorithm, considering the two inputs, i.e., the block/plaintext size ($2n$) and the key size (mn), we compute the following parameters: word size (n), key word (m), alph shift (α), beta shift (β), and number of rounds (T), as in Table 1. These parameters play a significant role in determining a specific SPECK configuration. For the SIMON, except the α and β parameters, all the above parameters played a crucial role in determining a specific SIMON configuration [13]. For instance, for both SPECK and SIMON, the number of rounds (T) forms the encryption and the decryption models, and also determines the level of security. In this case, the T increases with the size of the inputs (i.e., block sizes and key sizes), and the security increases with T. For SIMON only, the constant sequence ($Z[j]$) distinguishes the configurations that comprise same block/plaintext size (column 1) but different key seizes (column 2).

In addition to computing the aforementioned parameters, for our embedded hardware architectures for SPECK, we create two other functions that can operate as lookup tables. These two functions have the same inputs, i.e., the key size and block/plaintext size. The first function is created to select an appropriate number of rounds (T). The second function is created to select a suitable number of shifts, i.e., for alph shift (α) and for beta shift (β). These functions and parameters are essential to initialize the SPECK functionality.

In the following sub-sections, we discuss and present the internal architectures for the two sub-tasks of SPECK, i.e., the round function and the key generation function.

A. INTERNAL ARCHITECTURE FOR SPECK KEY GENERATION FUNCTION

Figure 8 illustrates our proposed internal architecture of the computation data path for the embedded hardware SPECK key generation function. From Table 1 (column 4), there are three different key words, where the key word m varies from 2, 3, to 4. This in turn leads to three different hardware structures for the SPECK key generation function. However, in this research work, we create only one hardware structure/design for the SPECK key generation function. Our unique hardware structure is created in such a way to be generic, parameterized, and scalable; thus, our hardware design can be reconfigured to process any key word (i.e., $m = 2, 3, 4$), without changing the internal architecture of this computation data path (in Figure 8).

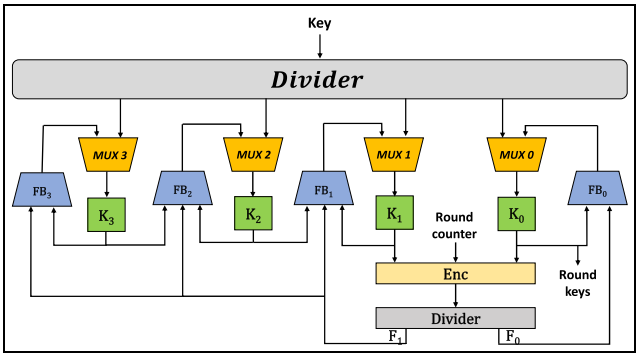

FIGURE 8. Computation data path for SPECK key generation function.

As demonstrated in Figure 8, the computation data path generates the round keys for the SPECK round function. In this case, a different round key is generated for each round. This computation data path comprises dividers, several general multiplexers (denoted by MUX_i), several feedback multiplexers (denoted by FB_i), several registers (denoted by K_i), and encryption module (denoted by Enc). As depicted, the process starts with the division operation. In this case, the input key, which is the original key, is divided

into m number of equal-sized blocks, known as the key words (as in Table 1). Based on the value of the key words (m), we can enable or disable certain K_i registers via the multiplexers to change the architecture and routing of the design. For an example, if the key word (m) is four (i.e., $m = 4$), then all four registers and all the general and feedback multiplexers are enabled, and the internal architecture is reconfigured to process 4 key words. Analogously, if the key word (m) is three (i.e., $m = 3$), then only one general multiplexer, one feedback multiplexer, and one register, (in this case, MUX_3, FB_3 and K_3) are disabled, and the internal architecture is reconfigured to process 3 key words. Furthermore, if the key word (m) is two (i.e., $m = 2$), then the general multiplexers, feedback multiplexers, and registers annotated with numbers 2 and 3 (in Figure 8) are disabled, and the internal architecture is reconfigured to process 2 key words.

Figure 8 also shows the data flow from the most significant register (either K_3 or K_2) to the least significant register (K_1), and then to the encryption module (*Enc*). In this case, the data is forwarded from one register to another in each round. Furthermore, the most significant register (MSR) varies based on the key word (m); for instance, if $m = 4$, the MSR is K_3, else if $m = 3$, the MSR is K_2. The K_0 register always produces the round key in each round, then forwards this newly generated round key as an input to the encryption module (*Enc*), as well as to the FB_0 multiplexer. As illustrated in Figure 8, the encryption (*Enc*) module receives three inputs: (1) one input from the K_0 register, which is the round key; (2) another input from the data flow via register K_1 if $m = 2$, or via registers K_1 and K_2 if $m = 3$, or via K_1, K_2, and K_3 if $m = 4$; (3) third input is the round counter value, which depends on the block/plaintext size (as in Table 1). The result of the *Enc* module is divided into two equal-sized blocks/lines (known as F_0 and F_1). The *Enc* module encompasses the encryption function. The internal architecture of the *Enc* module is detailed in Section IV.C.

The final results (or outputs) of the key generation function are the round keys, which are utilized as inputs for the SPECK round function in Section IV.B. As mentioned before, a new round key is generated in each round.

B. INTERNAL ARCHITECTURE FOR SPECK ROUND FUNCTION

Figure 9 demonstrates our proposed internal architecture of the computation data path for the embedded hardware SPECK round function. As detailed in Section II, the SPECK algorithm is based on the symmetric block ciphers, since symmetric keys are utilized to encrypt the blocks of data. As a result, this computation data path (in Figure 9) is created in such a way to perform the symmetric process.

As illustrated in Figure 9, this computation data path comprises a divider, general multiplexers (MUX_A and MUX_B), buffer module, and encryption and decryption module (denoted by *Enc/Dec*). The symmetric process of the SPECK round function starts with the division operation, where the input data block (either plaintext or ciphertext) is divided into

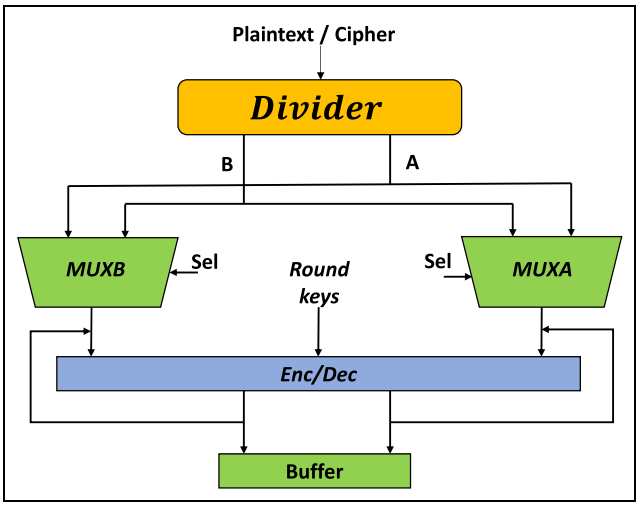

FIGURE 9. Computation data path for SPECK round function.

two equal-sized blocks. For instance, if the input data block size is 64-bits, then it is divided into two 32-bit data blocks. The outputs of the division operation, i.e., these two data blocks, are represented as A and B lines/blocks in Figure 9.

Depending on a specific function, i.e., either encryption or decryption, the two multiplexers swap the positions of these two lines using the associated "Sel" signals. For instance, for the encryption function, both "Sel" signals are de-asserted (set to logic 0), and MUX_A forwards A line/block, while MUX_B forwards B line/block. Conversely, for the decryption function, both "Sel" signals are asserted (set to logic 1), and MUX_A forwards B line/block, while MUX_B forwards A line/block.

Apart from the plaintext/ciphertext, another input to the SPECK round function is the round key produced from the key generation function in Section IV.A. A newly generated round key (in each round) and the A and B lines/blocks are the inputs to the *Enc/Dec* module, which encompasses the encryption and decryption functions. The internal architectures of the encryption and decryption functions are detailed in Section IV.C. As shown in Figure 9, the two outputs of this *Enc/Dec* module are stored in the buffer, and simultaneously forwarded as the new inputs to the *Enc/Dec* module in the subsequent round. In each round, the inputs of the *Enc/Dec* module alternate between the A and B blocks, and the *Enc/Dec* results. This symmetric process continues, and the final ciphertext/plaintext result is formed after the total number of rounds (T) is completed.

C. INTERNAL ARCHITECTURE FOR SPECK ENCRYPTION AND DECRYPTION FUNCTIONS

Figures 10 and 11 illustrate our internal architectures of the computation data path for the SPECK encryption function and SPECK decryption function, respectively. These two are created based on the equations (1) and (2) for encryption and decryption, respectively. The *Enc* module in Figure 8 encompasses the encryption function, whereas the *Enc/Dec* module

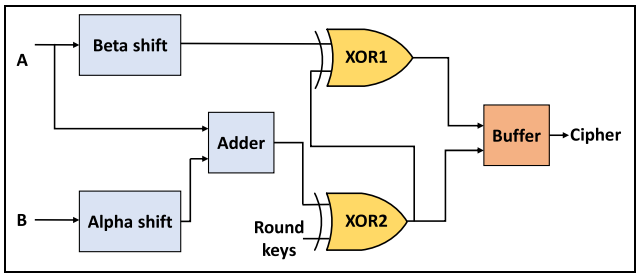

FIGURE 10. Computation data path for SPECK encryption function.

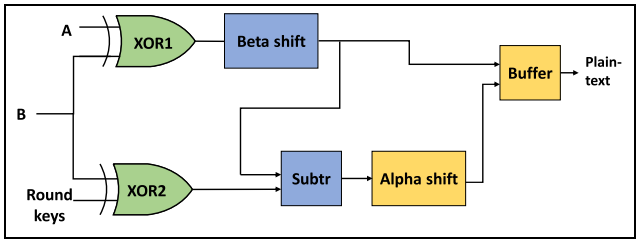

FIGURE 11. Computation data path for SPECK decryption function.

in Figure 9 encompasses both the encryption and decryption functions.

As demonstrated in Figure 10, the encryption function consists of alpha shift and beta shift operators, addition operators, XOR logic-operators, and a buffer. As shown, the A and B lines, and the round keys are the inputs to the SPECK encryption function. In this case, the A and B lines/blocks are the same ones (i.e., two equal-sized blocks) used in the symmetric process in Figure 9. For the encryption function, out of these two equal-sized blocks, the block with the most significant bits (MSBs) is assigned to A and the block with the least significant bits (LSBs) is assigned to B. As depicted in Figure 10, the B line/block goes through the alpha shift (from Table 1), followed by the addition operation with the A line/block. The result of the addition operation goes through the second XOR operation (*XOR2*) with the round key. The result of *XOR2* operation is stored in the buffer. Conversely, the A line/block goes through the beta shift (in Table 1). The beta shift result goes through the first XOR operation (*XOR1*) with the result of *XOR2* operation. The result of *XOR1* is also stored in the same buffer as the result of *XOR2* operation. The outputs of the *XOR1* and *XOR2* are utilized to form the ciphertext after completing the total number of rounds (T).

As depicted in Figure 11, the decryption function consists of XOR logic-operators, alpha shift and beta shift operators, subtraction operators, and a buffer. Similar to the encryption function, the A and B lines, and round keys are the inputs to the SPECK decryption function. In this case also, the A and B lines/blocks are the same ones (i.e., two equal-sized blocks) used in the symmetric process in Figure 9. For the decryption function, unlike the encryption, out of these two equal-sized blocks, the block with the most significant bits (MSBs) is assigned to B and the block with the least significant bits (LSBs) is assigned to A. As shown in Figure 11, the A and B lines/blocks go through the first XOR operation (*XOR1*). Then the result of *XOR1* goes through the beta shift (from Table 1). The beta shift result is stored in the buffer. Conversely, the B line/block goes through second XOR operation (*XOR2*) with the round key. The result of this *XOR2* is subtracted from the result of the beta shift. The result of the subtraction operation goes through the alpha shift. The alpha shift result is also stored in the same buffer as the beta shift result. In this case, the outputs of the beta shift and alpha shift are used to form the plaintext after completing the total number of rounds (T).

D. OUR TOP-LEVEL ARCHITECTURE FOR DYNAMIC AND PARTIAL RECONFIGURABLE HARDWARE FOR SPECK AND SIMON

In this sub-section, we discuss and present our top-level architecture and system-level setup for our dynamic and partial reconfigurable hardware for SPECK and SIMON lightweight cryptographic algorithms.

The dynamic and partial reconfiguration process of the aforementioned two lightweight cryptographic algorithms is as follows. Initially, the full configuration bitstream that comprises the reconfigurable module (RM) of the SIMON is downloaded to the FPGA, then the FPGA is configured to its appropriate hardware circuitry, and the SIMON algorithm is performed. Once the SIMON algorithm is executed, the RM for SIMON sends an ''execution complete'' signal to the processor or to the configuration controller. Next, the configuration controller (or the processor) downloads the partial bitstream for the RM for SPECK, then the RM is modified from SIMON to SPECK, and the SPECK algorithm is performed. After executing both the SIMON and SPECK, the final DRH results for SIMON and SPECK, stored in the on-chip BRAM, are brought to the external HyperTerminal window [42] of a desktop computer via the MicroBlaze processor and RS232 UART, in order to verify that the DRH design operates correctly, and the dynamic and partial reconfiguration is performed correctly. Additional signals are utilized to further verify the latter. In our design, the loading of the partial bitstreams to the reconfigurable parts of the FPGA (i.e., to the RM) and modifying the functionalities of the RMs are done without interrupting the operations of the remaining parts of the FPGA and typically without human intervention.

Our top-level architecture for the dynamic and partial reconfiguration process is shown in Figure 12 (modified from [42], [43]). As detailed in Section III.D, the full and partial bitstreams are stored in the external non-volatile memory, and the configuration controller manages the loading of the bitstreams and reconfiguring the FPGA as needed. As illustrated in Figure 12, for our design, we employ the compact flash (CF) memory to store the required full and partial bitstreams. In order to facilitate the dynamic and partial reconfiguration process as well as to perform the in-circuit reconfiguration, we integrate the internal

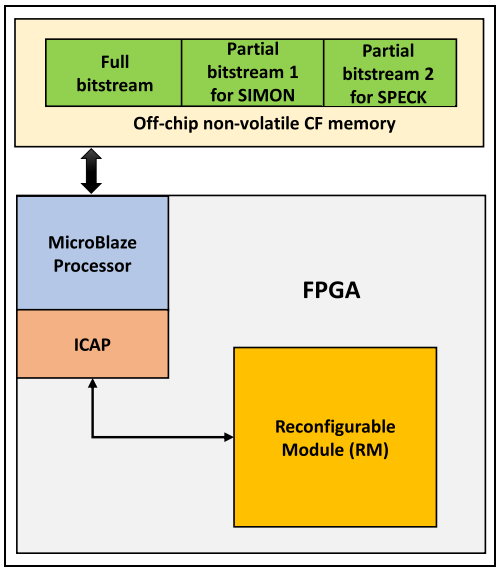

FIGURE 12. Top-level architecture for our dynamic and partial reconfigurable hardware for SPECK and SIMON.

configuration access port (ICAP) [46], [47]. Furthermore, we utilize the MicroBlaze processor as our configuration controller. The ICAP is also controlled by the MicroBlaze processor, and is used to load the partial bitstreams to the FPGA. In this case, the partial bitstreams are fetched from the CF via the MicroBlaze to the ICAP, and downloaded to the region of the RM as needed. The specific region of the RM is typically determined and selected during the design phase, based on the required resource utilization of the algorithm.

After executing both the lightweight cryptographic algorithms, i.e., SIMON and SPECK, the MicroBlaze processor can dynamically and partially reconfigure the chip/FPGA back to SIMON, without downloading the full bitstream. In this way, the SIMON and SPECK algorithms (with any one of the 20 different configurations) can be processed any number of times as needed, without interrupting the system's operations and often without human interventions.

V. EXPERIMENTAL RESULTS AND ANALYSIS

We perform experiments to evaluate and illustrate the feasibility and efficiency of our proposed dynamic and partial reconfigurable hardware architectures for SPECK and SIMON lightweight cryptographic algorithms. Experiments are also performed to evaluate the internal architectures of our embedded and reconfigurable hardware for these cryptographic algorithms.

As distinguished in Section IV, we design and implement our dynamically reconfigurable hardware architectures as two versions: static reconfigurable hardware (SRH) for SPECK and SIMON as separate entities, and dynamic reconfigurable hardware (DRH) for SPECK and SIMON together. The former can be dynamically reconfigured to 20 different configurations, without changing the internal architecture; and the latter can be dynamically and partially reconfigured from SIMON to SPECK and vice versa, by reconfiguring the on-chip hardware from one cryptographic algorithm to another.

A. SPACE AND TIME ANALYSIS

To investigate the feasibility and efficiency of our proposed dynamic and partial reconfigurable hardware architectures for the SPECK and SIMON lightweight cryptographic algorithms, cost analysis on space and time is carried out for static reconfigurable hardware (SRH) and dynamic reconfigurable hardware (DRH) designs.

1) ANALYSIS ON SPACE SAVINGS

The space (or area) is one of the main criteria for performance analysis, since area is a major constraint, especially for small-footprint portable and embedded devices. As stated in [13], this performance metric directly impacts not only the cost associated with the final product but also the feasibility of implementing cryptographic algorithms on a specific embedded platform.

The hardware resource utilization (or the occupied area) on chip for our proposed SRH and DRH for SIMON and SPECK algorithms is presented in Table 2. From this table, significant resource utilization parameters are the number of occupied slices, number of DSPs, and number of BRAMs, whereas the number of occupied slices typically contains the slice registers, slice LUTs, and flip-flops.

TABLE 2. Space statistics for SPECK and SIMON: SRH vs. DRH.

Configuration	On-chip resource utilization				
	Number of occupied slices	Number of DSP48E1s	Number of BRAMs	Slice registers	Slice LUTs
SRH for SIMON (hw-v1a)	4,003	3	85	8,515	10,224
SRH for SPECK (hw-v1b)	3,851	3	85	7,618	8,859
DRH with largest RM (hw-v2)	5,719	3	85	10,129	11,865

As illustrated in Table 2, the total number of occupied slices, the total number of DSP slices, and the total number of BRAMs required for SRH with SIMON (hw-v1a) and SRH with SPECK (hw-v1b) are 7854 (=4003+3851), 6 (=3+3), and 170 (=85+85), respectively. Conversely, the total number of occupied slices, the total number of DSP slices, and the total number of BRAMs required for DRH (hw-v2) are 5719, 3, and 85, respectively. For the DRH area analysis, we utilize the DRH design for the SIMON algorithm, which consists of the largest RM of the two algorithms (SPECK vs. SIMON).

From these results and analyses, considering the total number of occupied slices on chip, the space saving using partial reconfiguration is 28%. Furthermore, considering the total number of DSP slices and total number of BRAMs, the space savings using partial reconfiguration are 50% and 50%, respectively. This significant space saving is mainly because the same area of the chip is being reused for both the SPECK and SIMON lightweight cryptographic algorithms in DRH design. In this case, the reconfigurable parts (i.e., the RM in Figure 12) on the chip are being reconfigured and

reused from one algorithm to another (either from SIMON to SPECK or vice versa), which in turn lead to dramatic space savings on chip/FPGA. Also, with dynamic and partial reconfiguration, we can integrate other lightweight cryptographic algorithms as needed, in order to be executed on the same area of the chip as SPECK and SMION. This will enable us to incorporate cryptographic algorithms to embedded devices efficiently and effectively, without compromising the integrity of the compute/data-intensive applications running on the remainder of these devices. This is indeed imperative for portable and embedded devices with their limited hardware footprint.

Apart from this significant space saving from our proposed dynamic partial reconfiguration architecture, our unique internal hardware structures for SPECK and SIMON also lead to major space savings, since each structure is created in such a way to encompass 20 configurations in one design. As illustrated Table 2, the resource utilization remains the same for all 20 SPECK configurations as well as for all 20 SIMON configurations.

2) ANALYSIS ON RECONFIGURATION SPACE OVERHEAD

The reconfiguration space overhead is the extra hardware required on chip for reconfiguration [37], [38]. As stated in [38], for some reconfiguration methods, reconfiguration space overhead is unavoidable, and could potentially occupy valuable real estate of the chip. Hence, it is imperative to analyze the reconfiguration space overhead for our dynamic and partial reconfigurable hardware (DRH) designs, especially for resource-constrained embedded devices.

As detailed in Sections III.D and IV.D, we integrate the AXI hardware ICAP (internal configuration access port) [45] to facilitate the in-circuit reconfiguration on the FPGA. Furthermore, we use the external non-volatile memory, specifically SystemACE compact flash (CF) [48] to store the full and partial bitstreams of our designs. In this case, the on-chip AXI SystemACE interface controller [48], known as the AXI SYSACE, acts as the interface between the AXI bus and the SystemACE CF external memory on the board. Then, we utilize the MicroBlaze and ICAP to fetch the full and partial bitstreams from and CF, and to download and reconfigure the FPGA at run time, without interrupting the system's operations. As a result, the ICAP module and the SystemACE interface controller are the only extra hardware required on chip for dynamic and partial reconfiguration. Based on the user guides for ML605 [30], the number of occupied slices for ICAP is 436, whereas the number of occupied slices for SystemACE interface controller is 46. Hence, the total number of occupied slices is 482. This value is indeed an approximation, since area often varies based on how these modules interface with the rest of the system. Considering the total number of slices (i.e., 37680 slices) on Virtex-6 FPGA, the reconfiguration space overhead (or the extra hardware required on chip) for reconfiguration is about 1.28% of the chip, and is constant. Also, considering the total number of occupied slices (i.e., 5,719, from Table 2) for the DRH design, the extra hardware (i.e., 482) occupies only 8.43% of our DRH design. This percentage of reconfiguration space overhead would be amortized and decreased, if we integrate other lightweight cryptographic algorithms to our DRH design.

3) ANALYSIS ON RECONFIGURATION TIME OVERHEAD

The reconfiguration time overhead is the time required to load and change the configuration from one algorithm (or task) to another [37], [38]. As stated in [38], this has to be done every time, we change the application or the functionality of the hardware on chip.

From our experimental results presented in Table 5 and 6, the reconfiguration time overhead is approximately 749 milliseconds for our DRH design, while our design is actually running (in real-time) on the FPGA at 100MHz.

Next, we investigate and analyze our experimental results obtained in order to gain further insight into the reconfiguration time overhead. It is observed that the partial bitstream created (with Xilinx PlanAhead tools) for the reconfigurable module (RM) (in Figure 12), is 9,232,561 bytes, or 73,860,488 bits. As stated in [38], [42], when utilizing the ICAP running at 100MHz and 3.2Gbps, the aforementioned partial bitstream can be loaded in: 73,860,488 bits/ 3.2Gbps = 23 millisecond. This theoretical value of 23 milliseconds for reconfiguration time overhead is much less than the actual experimental value of 749 milliseconds.

For the theoretical value, it is assumed that the ICAP is continuously enabled at 100MHz, and the configuration utilizes the full bandwidth of 3.2Gbps. However, in a practical scenario, the partial bitstreams are stored in the external non-volatile CF and the MicroBlaze processor fetches and executes data/instructions sequentially, which often leads to higher reconfiguration time overhead. This difference was further investigated, analyzed, and detailed in [21], [38]. From our previous work [20], [21], it was observed that the partial reconfiguration time overhead is in the range of milliseconds for the bit files of similar sizes. All these facts and our previous work [20], [21], [40], [49] illustrate that this difference between theoretical and experimental values for reconfiguration time overhead is quite normal.

There are several existing works [50], [51] that propose techniques to reduce the reconfiguration time overhead. Although, this is beyond the scope of this article, these techniques will be investigated as future work, in order to further enhance our DRH designs.

B. ANALYSIS OF EXECUTION TIMES AND SPEEDUPS FOR SRH AND DRH DESIGNS

The execution time, which directly relates to the speedup, is another main criteria for performance analysis for our proposed embedded and reconfigurable hardware designs. Hence, in this sub-section, we discuss and analyze the experimental results obtained for each configuration in terms of execution times and the speedup.

As detailed in Section II and III, both the SPECK and SIMON algorithms have 20 different configuration options (i.e., 10 for encryption and 10 for decryption), based on the varying key sizes and varying block (plaintext/ciphertext) sizes. However, in this research work, we create only one hardware architecture/structure for the SPECK algorithm, in such a way to be generic, parameterized, and scalable; thus, without changing the internal architecture, our hardware design can be reconfigured to any one of the 20 configurations. Similarly, in [13], we introduced a unique hardware structure for SIMON, which encompasses 20 configurations in one design.

In this article, we perform experiments for all 20 configurations for both the SPECK and SIMON by reconfiguring the embedded and reconfigurable hardware designs from one configuration to another as needed on-the-fly. We obtain the execution times for each configuration for both the SPECK and SIMON algorithms.

1) ANALYSIS ON EXECUTION TIMES FOR SRH

In order to evaluate our proposed dynamic and partial reconfigurable hardware (DRH – hw-v2) architecture for SPECK and SIMON lightweight cryptographic algorithms, we design and implement static reconfigurable hardware (SRH) designs for SIMON (hw-v1a) and SPECK (hw-v1b) algorithms separately. Our proposed SRH and DRH designs are detailed and distinguished in Section IV.

With our SRH designs for cryptographic algorithms, firstly, a full configuration bitstream comprising the SIMON algorithm is downloaded to the FPGA and the FPGA is reconfigured to its appropriate hardware circuity, only once. After the SIMON is executed, in order to execute the SPECK algorithm, a full bitstream consisting of the SPECK is downloaded to the FPGA, and the entire FPGA is reconfigured again. This process can continue from SIMON to SPECK and vice versa. For SRH designs, the system's operation is interrupted for every download and reconfiguration process.

The experiments are performed on SRH designs for SIMON (hw-v1a) and SPECK (hw-v1b) for 20 different configurations, i.e., for encryption and decryption with varying block sizes and with varying key sizes. Then the execution times are obtained separately for each configuration and presented in Tables 3 and 4 for encryption and decryption, respectively. The execution time for each configuration is measured 10 times and the average is presented. In this case, the total execution times (presented in the column 4 in Tables 3 and 4) do not include the download and the reconfiguration times between the two SRH designs.

The execution times for SRH designs are obtained using the AXI Timer [33], while our designs are actually running (in real-time) on the FPGA at 100MHz. The execution time is measured in clock cycles, which is a standard unit; hence could potentially be used to estimate the time/speedup of our proposed designs on different platforms.

Visually, from Figure 13 and also from Table 3, the execution times for SRH with SIMON (hw-v1a) and SRH with SPECK (hw-v1b) for encryption remain almost the same, i.e., around 13.57μs -13.95μs (column 2) and 72.73μs - 79.26μs (column 3), respectively for all the configurations regardless of the block sizes and the key sizes. As illustrated from Table 4, the execution times for SRH for decryption show similar behaviors. This is mainly because our efficient and generic architectures for SPECK and SIMON, including the system-level and internal architectures, are created in such a way that the execution times are not affected by the input data sizes.

Furthermore, the execution times for SPECK (column 3) is much higher than the execution times for SIMON (column 2). This is mainly due to the higher design complexity of the SPECK algorithm compared to that of SIMON.

TABLE 3. Separate execution times for SRH for SIMON and SPECK for encryption.

Configurations (Plaintext/Key)	Execution times in AXI_clk_cycles for encryption		
	SIMON	SPECK	Total
32/64	1365	7289	8654
48/72	1357	7273	8630
48/96	1359	7437	8796
64/96	1357	7593	8950
64/128	1366	7645	9011
96/96	1359	7718	9077
96/144	1359	7762	9121
128/128	1361	7814	9175
128/192	1363	7814	9177
128/256	1395	7926	9321

TABLE 4. Separate execution times for SRH for SIMON and SPECK for decryption.

Configurations (Plaintext/Key)	Execution time in AXI_clk_cycles for decryption		
	SIMON	SPECK	Total
32/64	1357	7164	8521
48/72	1356	7228	8584
48/96	1371	7268	8639
64/96	1367	7328	8695
64/128	1378	7541	8919
96/96	1359	7593	8952
96/144	1362	7593	8955
128/128	1363	7666	9029
128/192	1358	7770	9128
128/256	1397	7822	9219

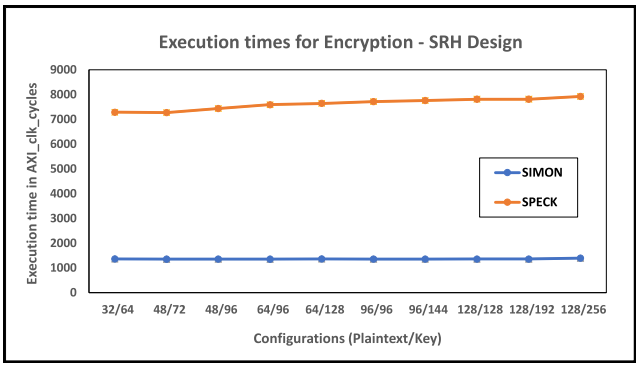

FIGURE 13. Graph for SRH for SIMON and SPECK encryption: execution times vs. varying configurations.

2) ANALYSIS ON EXECUTION TIMES FOR DRH

With our DRH designs for cryptographic algorithms, initially, a full configuration bitstream comprising the reconfigurable

module (RM) of the SIMON algorithm is downloaded, and the FPGA is reconfigured to its appropriate hardware circuity, and the SIMON algorithm is performed. After the execution of SIMON, the partial bitstream for the RM of the SPECK algorithm is downloaded to the specific region (i.e., the reconfigurable part) of the chip consisting of the SIMON RM, and that region is reconfigured to the SPECK algorithm. Then the SPECK algorithm is performed. Since both SIMON and SPECK comprise 20 different configurations each, in order to process varying configurations, the hardware is again reconfigured, partially and dynamically, back to SIMON without downloading the full bitstream or without interrupting the system's operation.

The experiments are performed on DRH designs (hw-v2) for 20 different configurations, i.e., encryption and decryption with varying block sizes and with varying key sizes. Unlike our SRH designs, for our DRH designs, the execution times (for each configuration) are measured consecutively from one cryptographic algorithm to another (i.e., SIMON → SPECK), without interrupting the remaining parts of the system, and without human intervention. Then the execution times are obtained separately for each configuration and presented in Tables 5 and 6 for encryption and decryption, respectively. The execution time for each configuration is measured 10 times and the average is presented. The execution times for DRH designs are also obtained using the AXI Timer [33], while our designs are actually running (in real-time) on the FPGA at 100MHz, and are measured in clock cycles.

In Tables 5 and 6, the total execution times (presented in column 5) include the reconfiguration time overhead for the DRH designs. The reconfiguration time overhead from one cryptographic algorithm to another, in our case, from SIMON to SPECK, are presented in column 3. Considering the values in column 3, the reconfiguration times slightly vary from 749 to 756 milliseconds. As detailed in Section V.A.3, in an ideal scenario, the reconfiguration time often depends on the size of the partial bitstream, i.e., the area of the reconfigurable module (RM). However, in a practical scenario, other factors, for instance, storing the partial bitstreams in off-chip CF and MicroBlaze processor fetching/executing data/instructions sequentially, can lead to slight variations in reconfiguration time.

In Tables 5 and 6, for DRH designs, columns 2 and 4 illustrate the individual execution times for SIMON and SPECK, respectively, for each configuration. Visually, from Figure 14 and also from Table 5, the individual execution times for DRH with SIMON and with SPECK for encryption remain almost the same, i.e., around 13.57μs - 14.05μs (column 2) and 72.9μs - 79.33μs (column 4), respectively for all the configurations regardless of the block sizes and the key sizes. This is analogous to the SRH designs for SIMON and SRH designs for SPECK. As illustrated from Table 5, the individual execution times for DRH for decryption show similar behaviors. This is also because our efficient and generic architectures for SPECK and SIMON, including the system-level and internal architectures, are created in such a way that the execution times are not impacted by the input data sizes.

From Tables 3–6, the individual execution times for DRH designs for SIMON (in column 2) and SPECK (in column 4) are also very close to the execution times for SRH designs for SIMON (column 2) and SPECK (column 3) for each configuration. As detailed in Section IV and as in [13], the internal hardware architectures for the SIMON and SPECK are the same for both the DRH and SRH designs, leading to almost similar individual execution times for each configuration.

From our previous work on dynamic and partial reconfigurable hardware architectures for data mining/analytics applications [20], [21], [40], [49] on embedded devices, it was observed that the percentage of reconfiguration time

TABLE 5. Separate execution times for DRH for SIMON and SPECK for encryption.

Configurations (Plaintext/Key)	Execution time in AXI_clk_cycles for encryption			
	SIMON	SIMON → SPECK reconfiguration	SPECK	Total
32/64	1366	74877375	7311	74886052
48/72	1366	74854147	7290	74862803
48/96	1368	74865055	7441	74873864
64/96	1357	74820092	7601	74829050
64/128	1371	75628035	7662	75637068
96/96	1367	74809214	7748	74818329
96/144	1362	74906026	7777	74915165
128/128	1365	74949949	7838	74959152
128/192	1370	74776982	7827	74786179
128/256	1405	75255827	7933	75265165

TABLE 6. Separate execution times for DRH for SIMON and SPECK for decryption.

Configurations (Plaintext/Key)	Execution time in AXI_clk_cycles for decryption			
	SIMON	SIMON → SPECK reconfiguration	SPECK	Total
32/64	1362	74902081	7168	74910611
48/72	1360	74888759	7228	74897347
48/96	1382	74844971	7297	74853650
64/96	1383	74924094	7352	74932829
64/128	1386	75190845	7569	75199800
96/96	1366	74865176	7606	74874148
96/144	1370	74862193	7599	74871162
128/128	1373	75488200	7689	75497262
128/192	1371	74926642	7795	74935808
128/256	1410	74962728	7835	74971973

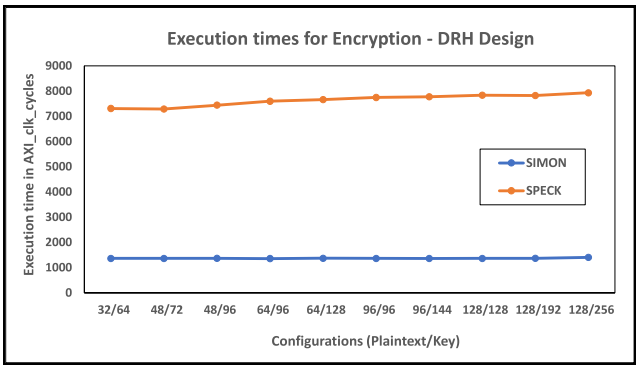

FIGURE 14. Graph for DRH for SIMON and SPECK encryption: execution times vs. varying configurations.

TABLE 7. Speedup for SIMON and SPECK for encryption: SRH and DRH vs. software.

Configurations (Plaintext/Key)	Execution time in AXI_clk_cycles for SIMON encryption			Speedup for SIMON		Execution time in AXI_clk_cycles for SPECK encryption			Speedup For SPECK	
	SRH	DRH	SW on MicroBlaze	SRH vs. SW	DRH vs. SW	SRH	DRH	SW on MicroBlaze	SRH vs. SW	DRH vs. SW
32/64	1365	1366	24156	17.69	17.68	7289	7311	99565	13.66	13.62
48/72	1357	1366	27664	20.38	20.25	7273	7290	106880	14.70	14.66
48/96	1359	1368	29264	21.53	21.39	7437	7441	108449	14.58	14.57
64/96	1357	1357	37886	27.91	27.92	7593	7601	122407	16.12	16.10
64/128	1366	1371	39682	29.04	28.94	7645	7662	127967	16.74	16.70
96/96	1359	1367	54598	40.17	39.94	7718	7748	142815	18.50	18.43
96/144	1359	1362	56639	41.67	41.59	7762	7777	146316	18.85	18.81
128/128	1361	1365	73752	54.18	54.03	7814	7838	177808	22.76	22.69
128/192	1363	1370	75454	55.35	55.08	7814	7827	182926	23.41	23.37
128/256	1395	1405	82110	58.86	58.44	7926	7933	205258	25.90	25.87

TABLE 8. Speedup for SIMON and SPECK for decryption: SRH and DRH vs. software.

Configurations (Plaintext/Key)	Execution time in AXI_clk_cycles for SIMON decryption			Speedup for SIMON		Execution time in AXI_clk_cycles for SPECK decryption			Speedup For SPECK	
	SRH	DRH	SW on MicroBlaze	SRH vs. SW	DRH vs. SW	SRH	DRH	SW on MicroBlaze	SRH vs. SW	DRH vs. SW
32/64	1357	1362	24189	17.83	17.76	7164	7168	102711	14.34	14.33
48/72	1356	1360	27618	20.37	20.31	7228	7228	108903	15.07	15.07
48/96	1371	1382	29208	21.30	21.13	7268	7297	112171	15.43	15.37
64/96	1367	1383	37932	27.75	27.43	7328	7352	118169	16.13	16.07
64/128	1378	1386	39656	28.78	28.61	7541	7569	127965	16.97	16.91
96/96	1359	1366	54686	40.24	40.03	7593	7606	143807	18.94	18.91
96/144	1362	1370	56727	41.65	41.41	7593	7599	144724	19.06	19.05
128/128	1363	1373	73756	54.11	53.72	7666	7689	176417	23.01	22.94
128/192	1358	1371	75458	55.57	55.04	7770	7795	183072	23.56	23.49
128/256	1397	1410	82114	58.78	58.24	7822	7835	198286	25.35	25.31

was amortized and decreased, as the computation complexity (i.e., the number of iterations/computations) increases as well as the size of the data increases; however, the percentage of reconfiguration time was significant for lower number of iterations/computations and smaller data sizes [20], [21]. Conversely, for our DRH designs for cryptographic algorithms, the percentage of reconfiguration time remains almost the same for each configuration. This is mainly because the individual execution times for DRH designs with SIMON and with SPECK for encryption and decryption remain almost the same for all the configurations. These results and analyses (from this research work and from our previous work) demonstrate that the more compute and data-intensive the applications/tasks are, the lesser the impact of the reconfiguration time overhead is.

3) ANALYSIS ON SPEEDUPS: SRH AND DRH VS. SOFTWARE

In order to evaluate our DRH designs as well as our SRH designs, we perform additional software experiments using the embedded MicroBlaze soft processor on the same ML605 development platform. In this case also, the experiments are performed on software designs for 20 different configurations, i.e., encryption and decryption with varying block sizes and with varying key sizes. Similar to the DRH designs, for the software designs, the execution times (for each configuration) are also measured in sequence from one cryptographic algorithm to another (i.e., SIMON → SPECK). Then the execution times are obtained separately for each configuration and presented in Tables 7 and 8 (columns 4 and 9) for encryption and decryption, respectively.

Although the execution times for our DRH and SRH for SPECK and SIMON are quite similar for varying configurations, the execution times for embedded software increase drastically with the increasing block (plaintext/ciphertext) sizes, and with the same key size. For instance, as illustrated in Table 7 (column 9), the execution times are 1.084 *ms* and 1.224 *ms* for the SPECK configurations of 48/96 and 64/96

(plaintext/ciphertext), respectively. Furthermore, the embedded software execution times also increase with the increasing key sizes and with the same block size; however, the incremental rate is not high as the one with the increasing block size. For instance, the execution times are 1.429 *ms* and 1.463 *ms* for the SPECK configurations of 96/96 and 96/144, respectively. The execution times for SIMON embedded software showed similar behaviors in [13]. As presented in Table 1, these differences are mainly because the increase in block size leads to dramatic increase in the number of rounds (or number of iterations), whereas the increase in key size leads to only minor increase in the number of rounds, and the number of rounds in turn impacts the execution times in embedded software designs for both SPECK and SIMON.

For the performance-gain (speedup) comparisons between the DRH and SRH, we focus on the individual execution (processing) times of the DRH and SRH designs with SPECK and SIMON algorithms. Initially, we measure the speedups of our hardware designs (both SRH and DRH) versus the software counterparts on the MicroBlaze processor. These speedups are presented in Tables 7 and 8 for encryption and decryption, respectively. In these tables, the speedups for SRH with SIMON and SPECK are in columns 5 and 10, respectively, whereas the speedups for DRH with SIMON and SPECK are in columns 6 and 11, respectively. As illustrated in Tables 7 and 8, the speedups vary from 18 to 59 for SRH designs with SIMON for varying configurations for both the encryption and decryption, whereas the speedups vary from 18 to 58.5 for DRH designs with SIMON for varying configurations for both the encryption and decryption. Furthermore, the speedups vary from 14 to 26 for SRH designs with SPECK for varying configurations for both the encryption and decryption, whereas the speedups vary from 14 to 26 for DRH designs with SPECK for varying configurations for both the encryption and decryption. This illustrates that DRH designs and SRH designs achieve almost similar speedups, when considering the individual execution times of the SIMON and SPECK algorithms.

Visually, as shown in Figures 15 and 16, the speedups increase exponentially for the DRH and SRH designs with SPECK and SIMON, respectively for encryption, with the increasing block sizes and also with the increasing key sizes. Similar behaviors are observed for the decryption for both SPECK and SIMON. From these results and analysis, it is evident that our DRH and SRH designs achieve much higher speedups compared to the software counterparts on the same embedded platform.

Figure 17 demonstrates the speedups for SPECK and SIMON for both the DRH and SRH designs, during the encryption process. As illustrated, for both the cryptographic algorithms, the speedups (performance) increase, with the increasing block sizes and also with the increasing key sizes. However, the incremental rate of performance improvement is much higher for SIMON (top line) compared to the that of SPECK (bottom line). Similar behaviors are observed for the decryption process. This is mainly due the higher design

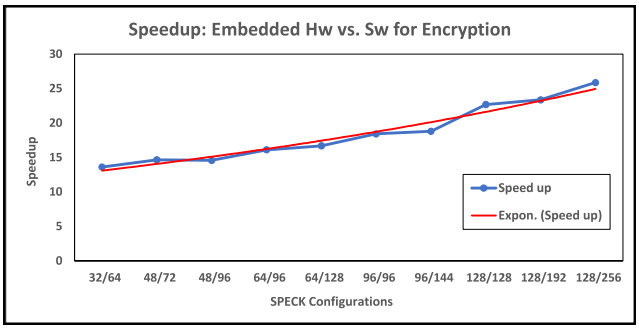

FIGURE 15. Speedup graph for DRH and SRH designs for SPECK encryption: speedup vs. varying configurations.

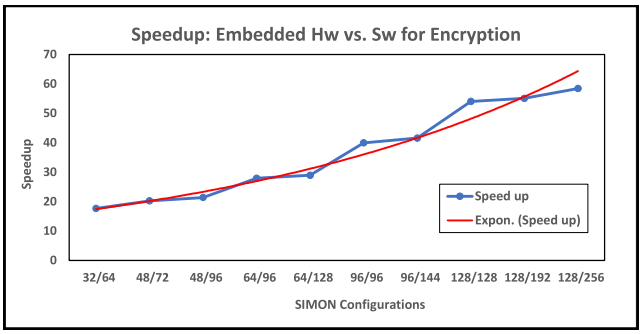

FIGURE 16. Speedup graph for DRH and SRH designs for SIMON encryption: speedup vs. varying configurations.

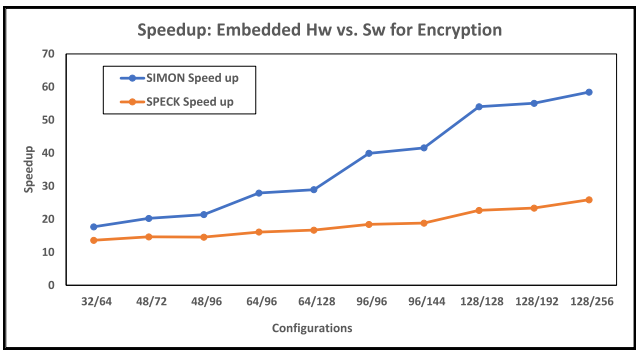

FIGURE 17. Speedup graphs for DRH and SRH designs for encryption: SPECK vs. SIMON.

complexity of SPECK, which in turn leads to higher execution times, thus lesser speedup, compared to that of SIMON hardware designs.

It should be noted that we do not make performance-gain (speedup) comparison between the DRH and SRH designs considering the total execution times. This is mainly because a significant percentage of the total execution time is spent on reconfiguration, thus, our DRH designs would not show much performance-gain. However, as detailed in Section V.A.1, our DRH designs achieve significant space savings compared to our SRH designs, i.e., 28%, 50%, and 50% space savings in terms of total number of occupied slices, number DSP slices, and number of BRAMs, respectively. Hence, it is crucial to

consider these speed-space tradeoffs, especially in portable and embedded devices with their limited hardware footprint.

As mentioned in Section V.A.3, for future work, we are planning to investigate and incorporate techniques to reduce the reconfiguration time overhead to further enhance the speedup of our DRH designs, although this is beyond the scope of this article.

From our previous work on dynamic and partial reconfigurable hardware architectures for data mining/analytics on embedded devices [20], [21], [40], [49], it was observed that the DRH designs and SRH designs achieved similar speedups, when considering the execution times for individual tasks/operations. This behavior is similar to the speedup results of our DRH and SRH for SPECK and SIMON algorithms, presented in this article.

When considering the total execution times in [20], [21], [40], [49], as anticipated, the DRH design achieved lesser speedup than that of the SRH design, i.e., 53 times versus 66 times, respectively. Although this 53 times speedup was significant, it was from the DRH design with the highest computation complexity and with the largest data size, whereas the speedup was much less (or almost non-existence) for the DRH design with low computation complexity and with small data size [20], [21]. This behavior is not similar to the speedup results of our DRH design for SPECK and SIMON algorithms, reported in this article. This is mainly because the hardware execution times for the DRH design for data mining/analytics application in [20], [21] varied with the computation complexity (i.e., number of iterations/computations) and with the data size. Conversely, the hardware execution times for our DRH designs for cryptographic algorithms remain almost the same for varying configurations.

C. ANALYSIS ON EXISTING WORKS ON DYNAMIC PARTIAL RECONFIGURABLE HARDWARE ARCHITECTURES FOR CRYPTOGRAPHIC ALGORITHMS

We investigated the existing works on dynamic and partial reconfigurable hardware architectures for cryptographic algorithms in the published literature. This investigation revealed that there were only few existing works on dynamic and partial reconfigurable hardware designs for cryptographic algorithms [52], [53], [54], [55], however, all of these focused on the AES (advanced encryption standard) algorithm.

A hardware-software co-design architecture was proposed in [52], in order to implement several Rijndael (AES) algorithms on the FPGAs. These AES algorithms were designs and developed on two platforms, Xilinx Spartan-2 and Altera EPEX-2. The dynamic reconfiguration was performed from one key size to another (128, 192, and 256). Although the authors claimed that partial reconfiguration was utilized, no experimental results were presented to validate this claim. Furthermore, only the simulation results were presented, and there was no indication of the actual implementation of the proposed design.

An FPGA-based reconfigurable co-processor was proposed for AES algorithm in [53]. In this case, the AES can be reconfigured from one key size to another (128, 192, and 256) using dynamic and partial reconfiguration. Similar to our DRH designs, Microblaze processor was utilized as a configuration controller. Experiments were performed on two platforms, Xilinx Spartan-2 and Virtex-2. In this article, the resource utilization results were presented separately for each configuration, which is unusual for DRH design. Typically, for DRH design, the resource utilization results are obtained from the DRH configuration that comprises the largest RM. Furthermore, the authors did not report the space savings due to the dynamic and partial reconfiguration of the AES algorithm, nor did they report the time overhead associated with the partial reconfiguration.

Another FPGA-based reconfigurable architecture was proposed for AES algorithm in [54], in which the AES was reconfigured from one key size to another (128, 192, and 256) using dynamic and partial reconfiguration. In this case also, experiments were performed on two platforms, Xilinx Virtex-2 and Virtex-5. Authors did not present any experimental results to validate the claim of utilizing the partial reconfiguration for DRH designs. Similar to [52], only the simulation results were presented, and there was no indication of the actual implementation of the proposed DRH design using partial reconfiguration.

In [55], two reconfigurable architectures were proposed for AES algorithm, based on two pipelined versions: modular pipelined for high-speed and simpler pipelined for area-efficiency. In this case also, the AES can be reconfigured from one key size to another (128, 192, and 256) using dynamic and partial reconfiguration. Experiments were performed on the Xilinx Zed board. Similar to [53], authors reported resource utilization separately for each configuration. In this article, the reconfiguration time was theoretically analyzed and presented, but actual reconfiguration time was not measured, while the DRH design was running on the chip.

It should be noted that all the above dynamic reconfigurable hardware (DRH) architectures were proposed for only one cryptographic algorithm, i.e., for AES algorithm, in which the AES was reconfigured to 3 different configurations with varying key sizes. Conversely, our proposed DRH architecture is designed for two cryptographic algorithms, i.e., for SPECK and SIMON, in which our DRH design can be dynamically and partially reconfigured from SIMON to SPECK and vice versa. Furthermore, our proposed SRH architectures for SIMON and SPECK, as separate entities, are created in such a way to be generic, parameterized, and scalable, hence, without changing the internal architectures, our SPECK and SIMON hardware designs can also be dynamically reconfigured to 20 different configurations, in order to perform the encryption and decryption, and to process varying block sizes and varying key sizes. For our SRH designs, partial reconfiguration is not utilized.

Based on the above investigation, we create a performance comparison table (Table 9) for most of the existing DRH

TABLE 9. Performance analysis: existing DRH designs vs. our proposed DRH designs for cryptographic algorithms.

Proposed work	Device or Platform	Configurations	Area of DRH Design (Slices; DSPs; BRAMs)	Area of SRH Designs (Slices; DSPs; BRAMs)	Actual implement. on chip with PR*
This work - DRH design for SIMON & SPECK	Xilinx Virtex-6	All 20 configurations	5719; 3; 85	SIMON (4003; 3; 85) SPECK (3851; 3; 85)	Yes
EI Abidine [53] – DRH design for AES	Xilinx Spartan-2	128,192, & 256	N/A	N/A	No
	Xilinx Virtex-2	128,192, & 256	N/A	AES128 (3565; n/a, n/a) AES192 (3764; n/a, n/a) AES256 (3632; n/a, n/a)	No
Wankhade [54] - DRH design for AES	Xilinx Virtex-2	128,192, & 256	N/A	N/A	No
	Xilinx Virtex-5	128,192, & 256	N/A	N/A	No
Burman [55] – DRH design for AES (a) high speed, (b) area-efficiency	(a) Xilinx Zed board	128,192, & 256	N/A	AES128 (5600; n/a, n/a) AES192 (6560; n/a, n/a) AES256 (7552; n/a, n/a)	No
	(b) Xilinx Zed board	128,192, & 256	N/A	AES128 (1512; n/a, n/a) AES192 (1888; n/a, n/a) AES256 (2136; n/a, n/a)	No

designs for any cryptographic algorithms, since we could not find any similar work for DRH designs specifically for SIMON and SPECK, in the published literature. In Table 9, we do not include the details from [3], since authors did not present enough evidence that partial reconfiguration was considered. Although the values (presented in Table 9) do not provide direct performance comparisons between our proposed DRH designs and the existing DRH designs for cryptographic algorithms, these values can be used as guidelines to enhance the design and development of future DRH designs not only for cryptographic algorithms but also for other similar algorithms.

In Table 9 (column 4), we present the total number of occupied slices, number of DSP slices, and number of BRAMs, for DRH designs. As illustrated, none of the existing works reported the occupied area for the DRH designs. In Table 10 (column 5), we present the total number of occupied slices, number of DSP slices, and number of BRAMs, for SRH designs as separate entities. As shown, existing works also reported the total number of occupied slices for the SRH designs as separate entities, but did not report the number of DSP slices or the number of BRAMs for these designs. It should be noted that our DRH design as well as our SRH designs encompass all 20 configurations in one design, whereas the existing designs were created to comprise only one configuration at a time. Regardless, our DRH design occupy less area on the chip as a single module, compared to the combined areas of the three different configurations for the existing ones in the literature.

In Table 9, the execution times (or speedup) values are not presented, since timing/speedup comparison between DRH designs for completely different cryptographic algorithms is not necessarily fair. Hence, in Table 9, we only present the resource utilization values, since these values are crucial when designing cryptographic algorithms on resource-constrained embedded devices, with their limited hardware footprint.

From the above investigation (Table 9, column 6), it is evident that most of the existing DRH designs for cryptographic algorithms were not fully and actually implemented on the FPGA, since no valid experimental results and analysis were presented to support these claims. Furthermore, the existing works did not report the reconfiguration time overhead, while the DRH designs were actually running on the chip. None of the existing works reported and analyzed the space savings due to the dynamic and partial reconfiguration. In addition, most of the existing works did not design and implement the system-level architectures.

From this investigation and to the best of our knowledge, we could not find any similar work as ours in the literature that provides dynamic and partial reconfigurable hardware architectures, specifically for SPECK and SIMON lightweight cryptographic algorithms, nor could we find any similar work that provides system-level architecture for the proposed DRH designs.

D. ANALYSIS ON EXISTING WORKS ON FPGA-BASED HARDWARE ARCHITECTURES FOR SPECK AND SIMON

We also investigated the existing works on FPGA-based hardware architectures for SPECK and SIMON algorithms in the published literature, since we could not find any existing works on dynamic and partial reconfigurable hardware architectures for these two algorithms.

In [67], an FPGA-based bit-serialized hardware architecture was proposed for only one SIMON configuration, i.e., for 128/128 configuration. Experiments were performed on two platforms: Xilinx Spartan-3 and Spartan-6. The proposed SIMON design was compared with different cryptographic algorithms, including AES, PRESENT, etc., in terms of area, specifically with the number of slices, in order to illustrate that SIMON is an alternative to AES for low-cost FPGA-based systems.

In [68], FPGA-based hardware architectures were proposed for both SIMON and SPECK algorithms for only two configurations, i.e., for 64/128 and 128/128 configurations. These designs were created as separate individual modules for each configuration and for each algorithm. The proposed designs were executed on Xilinx Spartan-3 platform, and compared with existing AES, PRESENT, and SIMON in terms of area (number of slices) and throughput, in order to demonstrate that SIMON and SPECK are more suitable for IoT applications and devices.

Another FPGA-based hardware architecture was proposed in [69], for only one SIMON configuration, i.e., for 32/64 configuration. The proposed design was executed on Xilinx Virtex-5 FPGA, and compared with different cryptographic algorithms, specifically Hummingbird and X-TEA, in terms of area (i.e., IOs, LUTs, registers, and buffers) and maximum frequency, to illustrate that SIMON is more appropriate for embedded applications.

An FPGA-based hardware architecture was proposed in [70], for only one SPECK configuration, i.e., for 128/128 configuration. Experiments were performed on Xilinx Spartan-3 platform. The proposed design was compared with various cryptographic algorithms, including AES, PRESENT, SIMON, etc., in terms of area (number of slices) and throughput, to demonstrate that SPECK is suitable for low-cost FPGAs.

From this investigation, it is evident that most of the aforementioned existing designs for SPECK and SIMON, were not generic, parameterized, or scalable. With these designs, usually, only one configuration was designed and implemented at a time, with only one block size and one key size. Hence, to create a different configuration, the underlying hardware circuitry needs to be changed, and then the design has to go through the whole synthesis and implementation process. Conversely, our SRH designs for SPECK and SIMON are created in such a way to be generic, parameterized, and scalable; thus, without changing the underlying hardware circuity, our design can be reconfigured on-the-fly to any one of the 20 different configurations. Furthermore, most of the existing works did not report the execution time or speedup for the proposed hardware designs. Also, none of the existing works proposed system-level architectures, which is imperative for embedded applications in real-world scenarios. Consequently, the existing works did not report the corresponding system-level area, and did not consider the associated memory access latency while reporting throughput/latency. As a result, we could not make any direct performance comparisons with the existing works on FPGA-based hardware architectures in the published literature.

VI. CONCLUSION AND FUTURE WORK

In this article, we introduced novel, unique, and efficient dynamic and partial reconfigurable hardware architectures for the most popular lightweight cryptographic algorithms: SPECK and SIMON. We created our dynamically reconfigurable hardware as two versions. As the first version, we introduced unique, customized, and optimized FPGA-based reconfigurable hardware architecture for SPECK, which was generic, parameterized, and scalable. As the second version, we introduced novel and unique dynamic and partial reconfigurable hardware architectures for both SPECK and SIMON. The first version can be dynamically reconfigured to 20 different configurations, without changing the internal architecture, which was analogous to our previously proposed SIMON reconfigurable hardware architecture in [13]. The second version can be dynamically and partially reconfigured from SIMON to SPECK and vice versa, by reconfiguring the on-chip hardware from one cryptographic algorithm to another. For both versions, the dynamic reconfiguration can be done without interrupting the system's operations and often without human intervention.

In this article, we distinguished the first and the second versions as the SRH (static reconfigurable hardware) and the DRH (dynamic reconfigurable hardware) designs, respectively. Although the first version can be dynamically reconfigured, we named this version SRH mainly because the partial reconfiguration was not utilized.

Due to the various reconfigurable features, our proposed hardware versions are highly flexible to accommodate different data arrays and data elements with varying data sizes; and the same architectures can be utilized for other embedded applications with diverse security requirements and not limited to IoT-related applications.

We also introduced unique and efficient system-level architectures for our proposed SRH and DRH designs for SPECK and SIMON lightweight cryptographic algorithms. With the system-level architecture, we created and incorporated unique pre-fetching techniques to reduce the memory access latency of our proposed reconfigurable hardware architectures for SPECK and SIMON algorithms. To the best of our knowledge, we could not find any similar work in the published literature that provides dynamic and partial reconfigurable hardware architectures for SPECK and SIMON; nor could we find any similar work that provides reconfigurable hardware architectures for SPECK, which can be dynamically reconfigured to 20 different configurations, without changing internal architectures. Also, we could not find any existing work on SPECK and SIMON that proposed system-level architecture, which is imperative for embedded applications in real-world scenarios.

From our experimental results and analysis, our DRH design showed a significant space savings, since the same

area of the chip/FPGA was being reused by reconfiguring the on-chip hardware circuity from one cryptographic algorithm to another (i.e., from SIMON → SPECK → SIMON →...), which is important for embedded devices with their stringent area constraints. With our DRH design, the space savings were about 28%, 50%, and 50% in terms of number of occupied slices, number of DSP slices, and number of BRAMs, respectively. Furthermore, the reconfiguration space overhead, which is the extra hardware required for reconfiguration, was relatively low compared to the whole chip (i.e., about 1.28%), and remained the same.

Considering the reconfiguration time overhead, we observed that there was a difference between the experimental value (749 milliseconds) and the theoretical value (23 milliseconds). This difference was mainly because, in our experimental setup, we utilized the MicroBlaze processor, with its sequential execution nature, as our configuration controller to bring the partial bitstreams from the off-chip CF. From our previous work [20], [21], it was observed that this difference between theoretical and experimental values for the reconfiguration time overhead is quite normal. Although, this is beyond the scope of this article, as future work, we are planning to investigate and incorporate techniques, such as [50], [51], to reduce the reconfiguration time overhead to further enhance the performance of our DRH designs.

Our current reconfigurable hardware architectures (both SRH and DRH designs) for SIMON and SPECK executed up to 59 times and 26 times, respectively, faster than their software counterparts on the embedded processor. In addition, for both SRH and DRH designs, it was observed that the processing times for SPECK remained almost the same for all 20 configurations. Similar behavior was observed for SIMON in [13]. This was mainly because our efficient and generic architectures for SPECK and SIMON, including the system-level and internal architectures, were created in such a way that the processing times were not affected by the input data sizes.

These experimental results are encouraging and show a great potential in utilizing FPGAs to create and incorporate lightweight cryptographic algorithms, specifically on embedded devices, considering the constraints associated with these devices, as well as the requirements of the applications running on these devices.

Power consumption is another major issue in the resource-constrained embedded devices. It has been demonstrated [65], [66] that FPGA-based reconfigurable hardware often consumes less power than embedded microprocessor-based software-only designs. Furthermore, as stated in [60]–[64], the dynamic and partial reconfiguration could potentially lead to reduction in power consumption. However, as future work, we are planning to investigate sophisticated power analysis tools to measure the power consumption of our reconfigurable hardware designs, since Xilinx Power Analysis tools for Virtex-6 only report estimated power, which does not reflect accurate values.

REFERENCES

[1] M. M. Kermani, M. Zhang, A. Raghunathan, and N. K. Jha, "Emerging frontiers in embedded security," in *Proc. 26th Int. Conf. VLSI Design 12th Int. Conf. Embedded Syst.*, Jan. 2013, pp. 203–208.
[2] S. Surendran, A. Nassef, and B. D. Beheshti, "A survey of cryptographic algorithms for IoT devices," in *Proc. IEEE Long Island Syst., Appl. Technol. Conf. (LISAT)*, May 2018, pp. 1–8.
[3] D. N. Serpanos and A. G. Voyiatzis, "Security challenges in embedded systems," *ACM Trans. Embedded Comput. Syst.*, vol. 12, no. 1s, pp. 1–10, Mar. 2013.
[4] N. B. F. Silva, D. F. Pigatto, P. S. Martins, and K. R. L. J. C. Branco, "Case studies of performance evaluation of cryptographic algorithms for an embedded system and a general purpose computer," *J. Netw. Comput. Appl.*, vol. 60, pp. 130–143, Jan. 2016.
[5] W. Jiang, Z. Guo, Y. Ma, and N. Sang, "Measurement-based research on cryptographic algorithms for embedded real-time systems," *J. Syst. Archit.*, vol. 59, no. 10, pp. 1394–1404, Nov. 2013.
[6] O. Hyncica, P. Kucera, P. Honzik, and P. Fiedler, "Performance evaluation of symmetric cryptography in embedded systems," in *Proc. 6th IEEE Int. Conf. Intell. Data Acquisition Adv. Comput. Syst.*, Sep. 2011, p. 277.
[7] T. Wollinger, J. Guajardo, and C. Paar, "Cryptography in Embedded Systems: An Overview," in *Proc. Embedded World Exhib. Conf.*, Feb. 2003, pp. 735–744.
[8] D. Dinu, Y. L. Corre, D. Khovratovich, L. Perrin, J. Großsschädl, and A. Biryukov, "Triathlon of Lightweight Block Ciphers for the Internet of Things," in *Proc. NIST Lightweight Cryptogr. Workshop*, Jul. 2015, pp. 1–21.
[9] W. J. Buchanan, S. Li, and R. Asif, "Lightweight cryptography methods," *J. Cyber Secur. Technol.*, vol. 1, nos. 3–4, pp. 187–201, Sep. 2017.
[10] M. Appel, A. Bossert, S. Cooper, T. Kußmaul, J. Löffler, C. Pauer, and A. Wiesmaier, "Block ciphers for the IoT–SIMON, Speck, KATAN, LED, TEA, PRESENT, and SEA compared," in *Proc. Appel Block CF*, 2016, pp. 1–37.
[11] R. Beaulieu, D. Shors, and J. Smith, *The Simon and Speck of Lightweight Block Ciphers*. Fort Meade, MD, USA: National Security Agency, Jun. 2013.
[12] P. Yalla and J.-P. Kaps, "Lightweight cryptography for FPGAs," in *Proc. Int. Conf. Reconfigurable Comput. FPGAs*, Dec. 2009, pp. 225–230.
[13] A. Alkamil and D. G. Perera, "Efficient FPGA-based reconfigurable accelerators for SIMON cryptographic algorithm on embedded platforms," in *Proc. Int. Conf. ReConFigurable Comput. FPGAs (ReConFig)*, Dec. 2019, pp. 1–8.
[14] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "The SIMON and SPECK block ciphers on AVR 8-bit microcontrollers," in *Proc. 3rd Int. Workshop Lightweight Cryptogr. Secur. Privacy (LightSec)*, in Lecture Notes in Computer Science, vol. 8898. Springer, 2014, pp. 3–20.
[15] S. N. Shahrouzi and D. G. Perera, "Optimized hardware accelerators for data mining applications on embedded platforms: Case study principal component analysis," *Microprocessors Microsyst.*, vol. 65, pp. 79–96, Mar. 2019.
[16] A. K. Madsen and D. G. Perera, "Efficient embedded architectures for fast-charge model predictive controller for battery cell management in electric vehicles," *EURASIP J. Embedded Syst.*, vol. 2018, no. 1, p. 2, Jul. 2018.
[17] M. A. Mohsin and D. G. Perera, "An FPGA-based hardware accelerator for K-Nearest neighbor classification for machine learning on mobile devices," in *Proc. 9th Int. Symp. Highly-Efficient Accel. Reconfigurable Technol.*, Jun. 2018, pp. 1–7.
[18] D. G. Perera and K. F. Li, "Analysis of single-chip hardware support for mobile and embedded applications," in *Proc. IEEE Pacific Rim Conf. Commun., Comput. Signal Process. (PACRIM)*, Aug. 2013, pp. 369–376.
[19] P. Garcia, K. Compton, M. Schulte, E. Blem, and W. Fu, "An overview of reconfigurable hardware in embedded systems," *EURASIP J. Embedded Syst.*, vol. 2006, pp. 1–19, Dec. 2006.
[20] S. N. Shahrouzi and D. G. Perera, "Dynamic partial reconfigurable hardware architecture for principal component analysis on mobile and embedded devices," *EURASIP J. Embedded Syst.*, vol. 2017, no. 1, p. 25, Feb. 2017.
[21] D. G. Perera and K. Fun Li, "FPGA-based reconfigurable hardware for compute intensive data mining applications," in *Proc. Int. Conf. P2P, Parallel, Grid, Cloud Internet Comput.*, Oct. 2011, pp. 100–108.
[22] D. G. Perera and K. F. Li, "Embedded hardware solution for principal component analysis," in *Proc. IEEE Pacific Rim Conf. Commun., Comput. Signal Process.*, Aug. 2011, pp. 730–735.

[23] A. K. Madsen, M. S. Trimboli, and D. G. Perera, "An optimized FPGA-based hardware accelerator for physics-based EKF for battery cell management," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Oct. 2020, pp. 1–5.

[24] G. Gogniat, T. Wolf, W. Burleson, J.-P. Diguet, L. Bossuet, and R. Vaslin, "Reconfigurable hardware for high-Security/ high-performance embedded systems: The SAFES perspective," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 16, no. 2, pp. 144–155, Feb. 2008.

[25] S. Drimer, "Security for volatile FPGAs," Comput. Lab., Univ. Cambridge, Cambridge, U.K., Tech. Rep. UCAM-CL-TR-763, Nov. 2009.

[26] T. Wollinger and C. Paar, "How secure are FPGA in cryptographic applications? (Long version)," in *Field Programmable Logic and Application. FPL* (Lecture Notes in Computer Science), vol. 2778. Berlin, Germany: Springer, 2003.

[27] B. Badrignans, J.-L. Danger, V. Fischer, G. Gogniat, and L. Torres, *Security Trends for FPGAS, From Secured to Secure Reconfigurable Systems*. New York, NY, USA: Springer, 2011.

[28] T. Wollinger and C. Paar, "How secure are FPGAs in cryptographic applications," in *Proc. 13th Int. Conf. Field-Program. Logic Appl.*, 2003, pp. 91–100.

[29] T. Wollinger and C. Paar, "Security aspects of FPGAs in cryptographic applications," in *New Algorithms, Architectures and Applications for Reconfigurable Computing*, P. Lysaght and W. Rosenstiel, Eds. Dordrecht, The Netherlands: Springer, 2005, pp. 265–278.

[30] Xilinx, Inc. (2011). *ML605 Hardware User Guide*. UG534(v1.5). [Online]. Available: www.xilinx.com/support/documentation/boards_and_kits/ug534.pdf

[31] Xilinx, Inc. (2012). *LogiCORE IP AXI Block RAM Controller, (v1.03a)*. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/axi_bram_ctrl/v1_03_a/ds777_axi_bram_ctrl.pdf

[32] Xilinx, Inc. (Dec. 2012). *LogiCORE IP AXI Interconnect*. DS768 (v1.06.a). [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/axi_interconnect/v1_06_a/ds768_axi_interconnect.pdf

[33] Xilinx, Inc. (Jul. 2012). *LogiCORE IP AXI Timer*. DS764 (v1.03.a). [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/axi_timer/v1_03_a/axi_timer_ds764.pdf

[34] Xilinx, Inc. (2013). *MicroBlaze Processor Reference Guide*. UG081, (v 14.7). [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/mb_ref_guide.pdf

[35] K. Compton and S. Hauck, "Reconfigurable computing: A survey of systems and software," *ACM Comput. Surv.*, vol. 34, no. 2, pp. 171–210, Jun. 2002.

[36] S. Hauck and A. Dehon, *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computing*. San Mateo, CA, USA: Morgan Kaufmann, 2008.

[37] K. F. Li and D. G. Perera, "A design methodology for mobile and embedded applications on FPGA-based dynamic reconfigurable hardware," *Int. J. Embedded Syst.*, vol. 1, no. 1, p. 1, 2019.

[38] D. G. Perera, "Analysis of FPGA-based reconfiguration methods for mobile and embedded applications," in *Proc. 12th FPGAworld Conf. - FPGAworld*, Sep. 2015, pp. 15–20.

[39] D. G. Perera and K. F. Li, "Analysis of computational models and application characteristics suitable for reconfigurable FPGAs," in *Proc. 10th Int. Conf. P2P, Parallel, Grid, Cloud Internet Comput. (3PGCIC)*, Nov. 2015, pp. 244–247.

[40] D. G. Perera, "Chip-level and reconfigurable hardware for data mining applications," Ph.D. dissertation, Dept. Elect. Comput. Eng., Univ. Victoria, Victoria, BC, Canada, Apr. 2012.

[41] J. Hussein and R. Patel, *MultiBoot With Virtex-5 FPGAs and Platform Flash XL*, document XAPP1100 (v1.0), Nov. 2008.

[42] Xilinx, Inc. (2010). *Partial Reconfiguration User Guide UG702 (v12.3)*. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_3/ug702.pdf

[43] D. Dye, *Partial Reconfiguration of Xilinx FPGAs Using ISE Design Suite*, document WP374 (v1.1), Jul. 2011.

[44] Xilinx, Inc. (2010). *Virtex-6 FPGA Configuration User Guide UG360 (v3.2)*. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug360.pdf

[45] Xilinx, Inc. (2012). *LogiCORE IP AXI HWICAP, DS817 (v2.03.a)*. [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/axi_hwicap/v2_03_a/ds817_axi_hwicap.pdf

[46] Xilinx, Inc. (2009). *PlanAhead User Guide, UG632 (v 11.4)*. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/PlanAhead_UserGuide.pdf

[47] V. Eck, P. Kalra, R. LeBlanc, and J. McManus, *In-Circuit Partial Reconfiguration of RocketIO Attributes*, document XAPP662 (v2.4), May 2004.

[48] Xilinx, Inc. (2011). *LogiCORE IP AXI SystemACE Interface Controller, DS789 (v1.01.a)*. [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/ds789_axi_sysace.pdf

[49] S. N. Shahrouzi, "Optimized embedded and reconfigurable hardware architectures and techniques for data mining applications on mobile devices," Ph.D. dissertation, Dept. Elect. Comput. Eng., Univ. Colorado at Colorado Springs, Colorado Springs, CO, USA, Dec. 2018.

[50] M. Hubner, D. Gohringer, J. Noguera, and J. Becker, "Fast dynamic and partial reconfiguration data path with low hardware overhead on xilinx FPGAs," in *Proc. IEEE Int. Symp. Parallel Distrib. Process., Workshops Phd Forum (IPDPSW)*, Apr. 2010, pp. 1–8.

[51] M. Liu, W. Kuehn, Z. Lu, and A. Jantsch, "Run-time partial reconfiguration speed investigation and architectural design space exploration," in *Proc. Int. Conf. Field Program. Log. Appl.*, Aug. 2009, pp. 498–502.

[52] R. Ashruf and G. S. G. Vassiliadis, "Reconfigurable implementation forT-the AES algorithm," in *Proc. ProRISC*, 2002, pp. 169–172.

[53] Z. E. Abidine and A. A. I. Moussa, "Self-partial and dynamic reconfiguration implementation for AES using FPGA," *IJCSI Int. J. Comput. Sci. Issues*, vol. 2, p. 33, Aug. 2009.

[54] S. Wankhade and R. Mahajan, "Dynamic partial reconfiguration implementation of AES algorithm," *Int. J. Comput. Appl.*, vol. 97, no. 3, pp. 15–18, Jul. 2014.

[55] S. Burman, P. Rangababu, and K. Datta, "Development of dynamic reconfiguration implementation of AES on FPGA platform," in *Proc. Devices Integr. Circuit (DevIC)*, Mar. 2017, pp. 247–251.

[56] D. G. Perera and K. F. Li, "Similarity computation using reconfigurable embedded hardware," in *Proc. 8th IEEE Int. Conf. Dependable, Autonomic Secure Comput.*, Dec. 2009, pp. 323–329.

[57] D. Perera and K. Li, "Hardware acceleration for similarity computations of feature vectors," *Can. J. Electr. Comput. Eng.*, vol. 33, no. 1, pp. 21–30, 2008.

[58] R. Wisniewski, "Dynamic partial reconfiguration of concurrent control systems specified by Petri nets and implemented in Xilinx FPGA devices," *IEEE Access*, vol. 6, pp. 32372–32391, 2018.

[59] *Dynamic Function Exchange*, document UG947 (v2019.2), Xilinx, Inc, Mar. 2020.

[60] *Vivado Design Suite User Guide, Partial Reconfiguration*, document UG909 (v2018.1), Xilinx Inc., Apr. 2018.

[61] S. Liu, R. N. Pittman, A. Forin, and J.-L. Gaudiot, "Achieving energy efficiency through runtime partial reconfiguration on reconfigurable systems," *ACM Trans. Embedded Comput. Syst.*, vol. 12, no. 3, p. 72, 2013.

[62] "FPGA run-time reconfiguration two approaches, version 1.0," Altera Corp., San Jose, CA, USA, White Paper, Mar. 2008. [Online]. Available: file:///C:\UsersC:\pdarsC:\DownloadsC:\38-21671%20(1).pdf

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

[64] J. Noguera and I. O. Kennedy, "Power reduction in network equipment through adaptive partial reconfiguration," in *Proc. Int. Conf. Field Program. Log. Appl.*, Aug. 2007, pp. 240–245.

[65] T. J. Todman, G. A. Constantinides, S. J. E. Wilton, O. Mencer, W. Luk, and P. Y. K. Cheung, "Reconfigurable computing: Architectures and design methods," *IEE Proc.-Comput. Digit. Techn.*, vol. 152, no. 2, pp. 193–207, Mar. 2005.

[66] M. Qasaimeh, K. Denolf, J. Lo, K. Vissers, J. Zambreno, and P. H. Jones, "Comparing energy efficiency of CPU, GPU and FPGA implementations for vision kernels," in *Proc. IEEE Int. Conf. Embedded Softw. Syst. (ICESS)*, Jun. 2019, pp. 1–8.

[67] A. Aysu, E. Gulcan, and P. Schaumont, "SIMON says: Break area records of block ciphers on FPGAs," *IEEE Embedded Syst. Lett.*, vol. 6, no. 2, pp. 37–40, Jun. 2014.

[68] R. Beaulieu, "SIMON and SPECK: Block ciphers for The Internet of Things," in *Proc. Lightweight Cryptogr. (NIST) Workshop*, Jul. 2015, p. 585.

[69] S. Feizi, A. Ahmadi, and A. Nemati, "A hardware implementation of simon cryptography algorithm," in *Proc. 4th Int. Conf. Comput. Knowl. Eng. (ICCKE)*, Oct. 2014, pp. 245–250.

[70] A. Nemati, S. Feizi, A. Ahmadi, and V. A.-D. Makki, "A low-cost and flexible FPGA implementation for SPECK block cipher," in *Proc. 12th Int. Iranian Soc. Cryptol. Conf. Inf. Secur. Cryptol. (ISCISC)*, Sep. 2015, pp. 42–47.

ARKAN ALKAMIL received the B.Sc. and M.Sc. degrees in electronics and electrical engineering from Al-Nahrain University, Iraq, in 2004 and 2008, respectively. He is currently pursuing the Ph.D. degree. He is currently working as a Research Assistant under the guidance of Dr. Darshika G. Perera. He is also working as a Lecturer with the Department of Electrical and Computer Engineering, University of Colorado at Colorado Springs. His research interests include hardware accelerators and hardware security.

DARSHIKA G. PERERA (Senior Member, IEEE) received the B.Sc. degree in electrical engineering from the University of Peradeniya, Sri Lanka, the M.Sc. degree in electrical engineering from the Royal Institute of Technology, Sweden, and the Ph.D. degree in electrical and computer engineering from the University of Victoria, Canada. She is currently an Assistant Professor with the Department of Electrical and Computer Engineering, University of Colorado at Colorado Springs (UCCS), USA. Prior to joining UCCS, she has worked as a Senior Engineer and the Group Leader of embedded systems with CMC Microsystems, Canada. Her research interests include reconfigurable computing, mobile and embedded systems, data mining, and digital systems. She is a Senior Member of the IEEE CAS VLSI and Computer Societies and the IEEE Women in Engineering. She received the ''Teacher of the Year–Tenure Track Award'' from the Engineering and Applied Science College, UCCS, in April 2019. She also received the Best Paper Award at the IEEE 3PGCIC 2011 Conference in 2011 and the Best Poster Paper Award at the ACM HEART 2018 Symposium in 2018. She serves on organizing and program committees for several IEEE/ACM conferences and workshops, and a reviewer for several IEEE, Springer, and Elsevier journals. She also serves on the VLSI Systems and Applications Technical Committee (VSATC) of the IEEE Circuits and Systems Society. She also serves as an Associate Editor for the *Microelectronics Journal (MEJ)* (Elsevier).