Received December 26, 2019, accepted January 7, 2020, date of publication February 7, 2020, date of current version February 17, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2972265

SHA-2 Acceleration Meeting the Needs of Emerging Applications: A Comparative Survey

RAFFAELE MARTINO, (Graduate Student Member, IEEE), AND ALESSANDRO CILARDO, (Senior Member, IEEE)
Department of Electrical Engineering and Information Technologies, University of Naples Federico II, 80125 Naples, Italy
Corresponding author: Alessandro Cilardo (acilardo@unina.it)

This work was supported by the SPHERE project under the Italian Ministry of Education, University and Research (MIUR) through the Research Projects of National Relevance (PRIN) Programme.

ABSTRACT While SHA-2 is a ubiquitous cryptographic hashing primitive, its role in emerging application domains, e.g. blockchains or trusted IoT components, has made the acceleration of SHA-2 very challenging due to new stringent classes of requirements, especially implementation cost and energy efficiency. The survey discusses these emerging applications and their fundamental requirements. Then, the work presents a comprehensive review of the different design techniques available in the literature for SHA-2 acceleration. The main focus of the presentation is placed on the impact of each design technique on the area, energy, power, and performance of the resulting accelerator, guiding the designer through the identification of the appropriate technique mixes which meet the constraints of any given application.

INDEX TERMS Accelerators, Hash functions, SHA-2.

I. INTRODUCTION

Cryptographic hash functions have been employed for decades as a fundamental building block of information security. Their properties ensure that message integrity as well as the sender's identity in a communication can be securely verified, provided that a secret has been shared between the communicating parties. Because of serious vulnerabilities identified in popular hash algorithms used in the past, namely MD5 and SHA-1, SHA-2 has now become crucial in a range of applications, being today the most commonly used hash function in practice. The traditional application domain of hash functions has been network security, usually over the Internet. In this context, client devices are usually sufficiently powerful to perform the relatively limited number of hash computations required by the security protocols, while servers can possibly benefit from hardware acceleration of the hash operation [1]. The hardware accelerator is designed to deliver the maximum possible throughput, usually traded off for increased area and power consumption.

However, the particular properties of cryptographic hash functions have paved the way for new application domains, beyond mere network security, with different sets of requirements. The most significant example is provided by blockchain applications, brought to the fore by the explosion of the Bitcoin cryptocurrency. The Bitcoin mining process, which heavily relies on the SHA-2 hash function, is extremely demanding in terms of energy efficiency, even making its profitability uncertain from the miner's standpoint. The development of new domains such as the Internet-of-Things (IoT), with its low-cost battery-powered devices needing secure communication, has also contributed to an increased demand for area- and energy-efficient accelerators to be paired with the resource-constrained main processor.

Pushed by the new classes of requirements posed by emerging applications, a number of different techniques have been introduced for the design of SHA-2 accelerators, geared towards the optimisation of area as well as energy or power consumption. While many of these area- and power-efficient techniques reach reasonable throughput levels, a few of them even sacrifice throughput in order to achieve substantial area or power savings. A designer confronted with the task of designing a SHA-2 hardware accelerator has therefore several alternatives to choose from, each with its own strengths and weaknesses. This paper comprehensively surveys the available techniques, discussing their impact on the different evaluation metrics of relevance for the designer: throughput, area occupation, and energy or power consumption. The discussion takes also advantage of the authors' previous work [2], which presented a SHA-2 evaluation framework used to analyze the implementation requirements of existing solutions. This analysis is combined here with an extensive review of the

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

state of the art to systematically show how relevant evaluation metrics are affected by architectural choices. The applications of SHA-2 are also surveyed in order to identify their key requirements and hence the most effective techniques adopted to meet them.

The rest of the paper is organised as follows. Section II reviews the SHA-2 algorithm family, while Section III surveys the fields of application for the hash functions. Section IV presents the spectrum of techniques available for the acceleration of SHA-2. Section V focuses on the dedicated SHA-2 hardware solutions proposed in the literature. In Section VI the impact of different implementation techniques on circuit-level metrics and the needs of end applications are discussed. Section VII concludes the paper with a few final remarks.

II. THE SECURE HASH ALGORITHM 2 (SHA-2)

The *Secure Hash Algorithm* (SHA) is a family of cryptographic hash functions defined by the National Institute of Standards and Technology (NIST) and published as the Federal Information Processing Standard (FIPS) 180, Secure Hash Standard (SHS) [3]. In the first version of the SHS, FIPS 180-0, published in 1993, only one hash function was described. This algorithm, now known as SHA-0, has been soon after replaced by SHA-1 in the revised version of the standard, FIPS 180-1, published in 1995. While the only difference between SHA-0 and SHA-1 is a single bitwise rotation, SHA-0 turns out to be considerably weaker than SHA-1 [4].

SHA-2 was firstly introduced in 2001, when FIPS 180-2 defined its two main variants, SHA-256 and SHA-512, respectively the 32- and 64-bit versions of the same hash algorithm. Additionally, the same revision described a truncated variant of SHA-512, SHA-384. Subsequent updates to the SHS added other truncated variants to the family. Namely, FIPS 180-3 in 2004 added SHA-224, which is a variant of SHA-256, whilst FIPS 180-4 added in 2012 two variants of SHA-512, SHA-512/224 and SHA-512/256, along with the general specification of a t-wide hash function called SHA-512/t.

In summary, SHA-2 is the set of the cryptographic hash algorithms defined in the SHS, excluding SHA-1. The following description will focus on SHA-256, while Section II-B5 will provide the detail of the other variants. Despite the recent introduction of SHA-3, which has been published separately as FIPS 202 [5], SHA-2 variants are currently widely in use as secure hash algorithms [6] and their relevance for a large variety of applications remains crucial.

A. PROPERTIES OF CRYPTOGRAPHIC HASH ALGORITHMS

A hash algorithm can be used for providing security services only if it ensures a set of properties, which are not necessarily guaranteed by general-purpose hash functions. These properties make algorithms like SHA-2 what we define a *cryptographic* hash function.

The *one-way* or *preimage resistance* property of cryptographic hash functions implies that it is computationally infeasible to compute the message M given its hash $DM(M)$.[1] The *second preimage resistance* property means that, given a hash value $DM(M)$, it is computationally infeasible to find a different message $M' \neq M$ that yields the same hash value. The *pseudo-randomness* property means that the hash value of a message must expose statistical randomness. Finally, the *collision resistance* property means that it is computationally infeasible to find a pair of messages M_1 and M_2 which produce the same hash value.

Each application has different requirements in terms of these properties for its underlying hash function [7]. In Section III a wide range of applications of SHA-2 are presented, not limited to security services, and their specific requirements in terms of cryptographic hash properties are discussed.

The security of a hash algorithm is a measure of the computational infeasibility of breaking its properties, especially its collision resistance. It is measured in terms of the number of operations required to break the algorithm, expressed as a power of 2. More specifically, a hash function is said to have x *security bits* if 2^x operations are required to break it. The name stems from the fact that, for an unbroken hash algorithm, the only feasible attack relies on brute force, which implies a mean number of attempts exponential in the length of the hash value $L(DM(M))$. However, due to the birthday paradox, the number of attempts required on average to find two different messages hashing to the same value, i.e. to break the collision resistance property in its broadest sense, is $2^{L(DM(M))/2}$, making the number of security bits a half of the hash size.

The hash sizes of the members of the SHA-2 family have been chosen to match the security bits provided by symmetric encryption algorithms. For these cryptographic algorithms, the number of security bits coincides with the key length, hence SHA-2 hash sizes are twice the key length of a symmetric encryption algorithm. Specifically, a hash size of 224 implies the same number of security bits of Triple DES [8], while 256, 384, and 512 are twice the key sizes supported by AES [9].

B. ALGORITHM STRUCTURE

SHA-2 is a block-based hash algorithm, meaning that it operates on blocks of fixed size l to produce a fixed-size hash value $DM(M)$ for a given input message M. For SHA-256, $l = 512$ bit and the hash size is 256 bit. Nevertheless, they are capable to process messages of practically any arbitrary length, up to a limit fixed by the padding. Padding is performed to ensure that the variable length $L(M)$ of the message M to be hashed is always a multiple of l.

Message blocks are processed *sequentially*, as the output of each block is combined with the current partial hash value to produce the new partial hash value, that is also fed as input

[1] If $DM(M)$ is the hash value of M, M is called the *preimage* of $DM(M)$.

to the hashing of the next message block. This inherently sequential structure prevents aggressive parallelization of the single computation.

1) PADDING STEP

The message is padded with a single 1 bit, followed by as many 0 bits as needed to reach a length congruent to 448 mod 512, so as to leave the last 64 bits for encoding a representation of the original length $L(M)$. This means that the length of the message to be hashed must not exceed 2^{64} bit. Padding is always performed, even when the length of the message does not strictly require it. Therefore, the number of message blocks required to hash a message can be written as $\lfloor L(M)/l \rfloor + 1$, which can also be written as a function of the padded message M^* in the form $L(M^*)/l$. Moreover, the actual maximum length of a message to be hashed is limited to $2^{64} - 65$ bit.

To recall that padding has to be performed prior to any hash computation, each message block is usually referred to as the *padded data block* (PDB).

2) DEFINITION OF VARIABLES

The current hash value is used for the hashing of the j-th PDB M_j, and is stored in eight 32-bit accumulator variables DM_k, $k \in [0, 7]$. Their initial values are the first 32 bits of the fractional parts of the square root of the first eight prime numbers, although they are also hard-coded in the standard [3] as follows:

$$\begin{aligned} DM_0(0) &= \texttt{0x6a09e667} & DM_4(0) &= \texttt{0x510e527f} \\ DM_1(0) &= \texttt{0xbb67ae85} & DM_5(0) &= \texttt{0x9b05688c} \\ DM_2(0) &= \texttt{0x3c6ef372} & DM_6(0) &= \texttt{0x1f83d9ab} \\ DM_3(0) &= \texttt{0xa54ff53a} & DM_7(0) &= \texttt{0x5be0cd19} \end{aligned} \tag{II.1}$$

For each PDB, the algorithm performs $R = 64$ iterations. At each iteration t, the value of the internal state variable Z is updated. The internal state variable is constituted by eight 32-bit working variables called A to H. To refer to the state variables as a whole, the following definition is used in this paper:

$$\begin{aligned} Z(0)_t &= A_t & Z(4)_t &= E_t \\ Z(1)_t &= B_t & Z(5)_t &= F_t \\ Z(2)_t &= C_t & Z(6)_t &= G_t \\ Z(3)_t &= D_t & Z(7)_t &= H_t \end{aligned} \tag{II.2}$$

At the beginning of the processing of each PDB, the value of the working variables matches the value of the accumulators:

$$\begin{aligned} A_0 &= DM_0(j) & E_0 &= DM_4(j) \\ B_0 &= DM_1(j) & F_0 &= DM_5(j) \\ C_0 &= DM_2(j) & G_0 &= DM_6(j) \\ D_0 &= DM_3(j) & H_0 &= DM_7(j) \end{aligned} \tag{II.3}$$

The message to be hashed determines the value of the variable W_t. Namely, this is a 32-bit variable whose value changes at every iteration as follows:

$$W_t = \begin{cases} M_j[32 \cdot t + 31 : 32 \cdot t] \\ \qquad t < 16 \\ \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16} \\ \qquad t \geq 16 \end{cases} \tag{II.4}$$

In other words, for the first 16 iterations, W_t is simply the t-th 32-bit word of M_j. Note also that all the additions involved in the SHA-2 family are performed modulo the variable size, which is 32 for SHA-256. Whenever needed, the $+$ symbol will denote the appropriate modular addition throughout the text. The two functions $\sigma_0(x)$ and $\sigma_1(x)$ are defined as follows.

$$\begin{aligned} \sigma_0(x) &= x \ggg_r 7 \oplus x \ggg_r 18 \oplus x \ggg 3 \\ \sigma_1(x) &= x \ggg_r 17 \oplus x \ggg_r 19 \oplus x \ggg 10 \end{aligned} \tag{II.5}$$

where $x \ggg_r n$ denotes the circular right shift of n bits and $x \ggg n$ denotes the logical right shift of n bits.

Last, the algorithm employs R constants K_t, whose values are hard-coded in the standard [3] as the first 32 bits of the fractional parts of the cube root of the first 64 prime numbers.

3) MESSAGE BLOCK PROCESSING

At each iteration, the following step functions are computed:

$$\begin{aligned} T^1{}_t &= H_t + \Sigma_1(E_t) + Ch(E_t, F_t, G_t) + K_t + W_t \\ &\quad \forall t \in [0, R-1] \end{aligned} \tag{II.6}$$

$$\begin{aligned} T^2{}_t &= \Sigma_0(A_t) + Maj(A_t, B_t, C_t) \\ &\quad \forall t \in [0, R-1] \end{aligned} \tag{II.7}$$

where the *Choose*[2] and *Majority* functions are defined as

$$Ch(x, y, z) = (x \wedge z) \oplus (\neg x \wedge y) \tag{II.8}$$

$$Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \tag{II.9}$$

whilst the $\Sigma_0(x)$ and $\Sigma_1(x)$ functions are defined as follows.

$$\begin{aligned} \Sigma_0(x) &= x \ggg_r 2 \oplus x \ggg_r 13 \oplus x \ggg_r 22 \\ \Sigma_1(x) &= x \ggg_r 6 \oplus x \ggg_r 11 \oplus x \ggg_r 25 \end{aligned} \tag{II.10}$$

The working variables are updated according to the following scheme:

$$\begin{aligned} A_{t+1} &= T^1{}_t + T^2{}_t & \forall t \in [0, R-1] \\ B_{t+1} &= A_t & \forall t \in [0, R-1] \\ C_{t+1} &= B_t & \forall t \in [0, R-1] \\ D_{t+1} &= C_t & \forall t \in [0, R-1] \\ E_{t+1} &= D_t + T^1{}_t & \forall t \in [0, R-1] \\ F_{t+1} &= E_t & \forall t \in [0, R-1] \\ G_{t+1} &= F_t & \forall t \in [0, R-1] \\ H_{t+1} &= G_t & \forall t \in [0, R-1] \end{aligned} \tag{II.11}$$

[2]The value of x chooses whether y or z is propagated to the output.

It is worth noting that only two working variables out of eight are actually updated at each iteration, whereas the other six take on the value of the preceding variables. In fact, for all variables but A_{t+1} and E_{t+1}, Equation (II.11) can be rewritten as:

$$Z(k)_{t+1} = Z(k-1)_t \quad \forall k \in [1,3] \cup [5,7] \tag{II.12}$$

4) MESSAGE BLOCK CHAINING

At the end of the 64 iterations, the current hash value is updated as follows:

$$\begin{aligned}
DM_0(j+1) &= DM_0(j) + A_R \\
DM_1(j+1) &= DM_1(j) + B_R \\
DM_2(j+1) &= DM_2(j) + C_R \\
DM_3(j+1) &= DM_3(j) + D_R \\
DM_4(j+1) &= DM_4(j) + E_R \\
DM_5(j+1) &= DM_5(j) + F_R \\
DM_6(j+1) &= DM_6(j) + G_R \\
DM_7(j+1) &= DM_7(j) + H_R
\end{aligned} \tag{II.13}$$

If $j < L(M^*)/l$, the algorithm continues with the processing of M_{j+1}; otherwise, there are no message blocks left and the message digest is $DM_0 \mid\mid DM_1 \mid\mid DM_2 \mid\mid DM_3 \mid\mid DM_4 \mid\mid DM_5 \mid\mid DM_6 \mid\mid DM_7$, where the $\mid\mid$ symbol denotes the concatenation operator.

5) SHA-2 VARIANTS

SHA-512 is the 64-bit variant of SHA-256. All the sizes are therefore doubled, namely the hash size is 512 bit, the block length l is 1024 bit, and the accumulator variables are 64-bit wise. Similarly to SHA-256, their initial values are the first 64 bits of the fractional parts of the square root of the first eight prime numbers, and are hard-coded in the standard [3].

SHA-512 performs $R = 80$ iterations to hash a single PDB. This means that 80 values for the K_t constants are required. These values, which are hard-coded in the standard, have been determined analogously to SHA-256, as the first 64 bits of the fractional parts of the cube root of the first 80 prime numbers.

The padding step reflects the doubled size, since the message is padded with a single 1 followed by as many 0 bits as needed to reach a length congruent to 896 mod 1024, so as to leave the last 128 bits for encoding the original length $L(M)$. This implies that the limit on the length of the message to be hashed is increased up to $2^{128} - 129$ bit.

The only remaining difference between SHA-512 and SHA-256 involves Equations (II.5) and (II.10), which become

$$\begin{aligned}
\sigma_0(x) &= x \ggg_r 1 \oplus x \ggg_r 8 \oplus x \ggg 7 \\
\sigma_1(x) &= x \ggg_r 19 \oplus x \ggg_r 61 \oplus x \ggg 6 \\
\Sigma_0(x) &= x \ggg_r 28 \oplus x \ggg_r 34 \oplus x \ggg_r 39 \\
\Sigma_1(x) &= x \ggg_r 14 \oplus x \ggg_r 18 \oplus x \ggg_r 41
\end{aligned} \tag{II.14}$$

The other four hash functions are in fact variants of SHA-256 and SHA-512. The specifications of these variants are exactly the same as their base algorithms, but the output is truncated to a lower number of bytes and different initial values $DM_0(0)$ to $DM_7(0)$ are employed. Namely:

- *SHA-224* is the same function as SHA-256, except that the output is truncated to 224 bit and different initial values $DM_0(0)$ to $DM_7(0)$ are used;
- *SHA-384* is the same function as SHA-512, except that the output is truncated to 384 bit and different initial values $DM_0(0)$ to $DM_7(0)$ are used;
- *SHA-512/224* is the same function as SHA-512, except that the output is truncated to 224 bit and different initial values $DM_0(0)$ to $DM_7(0)$ are used;
- *SHA-512/256* is the same function as SHA-512, except that the output is truncated to 256 bit and different initial values $DM_0(0)$ to $DM_7(0)$ are used.

The reader is referred to the standard [3] for the selection of the initial values specific to each variant. The different hash functions of the SHA-2 family are compared in Table 1. The first part of the table presents some functional characteristics of the hash algorithms. The comparison highlights that the different variants of SHA-2 offer different combinations of security levels and block size. A symmetric encryption algorithm with equivalent security level is also listed. The second part of the table compares some internal parameters, which depend on the algorithm being derived from the SHA-256 or SHA-512 main variant.

III. APPLICATIONS OF SHA-2

The properties of cryptographic hash functions make them one of the most versatile cryptographic tools, used in a variety of security services [7].

Digital signature schemes based on asymmetric encryption, such as *Digital Signature Algorithm* (DSA) [10], employ hash functions to reduce their computational complexity while retaining the nonrepudiation property. Due to the processing time of asymmetric cryptography, it would be impractical to directly process a large file to obtain its digital signature, so only the hash of the file is signed. Due to the collision resistance property, it is computationally infeasible to find a second file with the same hash, and hence the same signature, of the original file, thereby preserving the nonrepudiation property.

Hash-Based Message Authentication Codes (HMACs) [11], [12] exploit the collision resistance property to provide message authentication, which ensures the integrity of the transmitted data. The message to be transmitted, along with a shared secret, is hashed to produce a message authentication code (MAC) which can be used at the receiver side to verify that the message has not been altered. The inclusion of the shared secret, in fact, prevents an attacker from simply replacing the whole message-MAC pair, since the latter cannot be computed without knowing the shared secret. It has been proved that HMAC is directly and formally related with the

TABLE 1. Characteristics of the members of the SHA-2 family of algorithms.

Parameter	SHA-256	SHA-512	SHA-224	SHA-384	SHA-512/224	SHA-512/256
Hash size (bit)	256	512	224	384	224	256
Block size (bit)	512	1024	512	1024	1024	1024
Message size (bit)	$< 2^{64}$	$< 2^{128}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$	$< 2^{128}$
Security bits	128	256	112	192	112	128
Corresponding symmetric cipher	AES-128	AES-256	TDES	AES-192	TDES	AES-128
Word size (bit)	32	64	32	64	64	64
Number of rounds	64	80	64	80	80	80
Publication year	2001	2001	2004	2001	2012	2012

security of the hash function employed [11]. HMACs are employed by the IPSec protocol [13], both in the Authentication Header (AH) [14] and in the Encapsulated Security Payload (ESP) [15] modes.

Pseudo-Random Number Generators (PRNGs) based on *Deterministic Random Bit Generation* (DRBG) [16] produce pseudo-random sequences by hashing a linearly increasing seed. In addition to cryptographic hash functions, PRNGs can also be built from HMACs [16].

Most of the security protocols listed above are standardized by NIST [10], [12], [16], which requires the underlying hash function to be one of its approved hash functions. However, vulnerabilities have been found in SHA-1 [17], leading eventually to its breaking [18]. Given that alternative hash functions such as MD5 [19] were already known to be broken [20], [21], SHA-2 has been the only viable alternative for many years. Although a new hash function, SHA-3, has been standardised by NIST in 2015 [5], it has not reached widespread diffusion yet. Considering the slow process that has taken place for SHA-2 to fully replace SHA-1 [22], along with the fact that there are no significant vulnerabilities to SHA-2, the new SHA-3 function is not expected to replace SHA-2 in the near future. Moreover, SHA-3 is based on a completely different mathematical construction than SHA-2. While the different nature of the function makes it difficult to re-use the body of knowledge in SHA-2 and SHA-1 cryptanalysis, more study is still required to increase confidence in the cryptographic strength of the new SHA-3 function [23]. On the other hand, SHA-2 has undergone intense cryptanalysis [24] for more than 15 years, and no weaknesses have been found yet, making SHA-2 attractive for emerging applications where long-term collision resistance is required.

A. BLOCKCHAINS

The blockchain technology employs hash functions to ensure the integrity of a distributed *ledger*. In essence, a blockchain is a distributed database growing in time within a peer-to-peer network. Transactions are grouped into blocks, which are concatenated by including the hash of the last block into the header of the next block. New blocks are added to the blockchain by running a distributed consensus algorithm, ensuring a consistent view of the database. Once the block sequence is agreed upon, the collision resistance property of the hash functions implies that it is infeasible to modify a transaction without being detected, since any change in the block would result in a different hash.

A critical role in ensuring the integrity of the distributed database is played by the distributed consensus algorithm. In fact, if attackers are able to subvert the consensus protocol, they may force the network to agree on their own version of the blockchain, with a content of their own choice. Specifically, consensus protocols running on peer-to-peer networks must avoid *Sybil attacks* [25], i.e. attacks based on the capability of the attacker to present multiple identities, gaining an apparent majority which can drive the consensus. The Bitcoin cryptocurrency [26], which has introduced the blockchain technology, bases its own distributed consensus protocol on SHA-256. It avoids Sybil attacks by forcing peers to perform a computationally-intensive task in order to participate in the consensus algorithm [23], [27].

The Bitcoin consensus protocol is also called *Nakamoto consensus* [28]. It mandates that a new block can be added to the network only if its hash, computed by applying twice the SHA-256 hash function, is below a specific threshold, called *target*. To this end, the header contains a *nonce* which can be changed by peers in order to alter the hash value of the block. The Nakamoto consensus relies on the one-way property of SHA-256: were the SHA-256 function to be invertible, it would be possible to compute the value of the nonce leading to the required hash value by simply inverting the SHA-256 function [27]. Instead, a brute-force approach is required, with minor possible improvements stemming from details of the Nakamoto consensus protocol [29], [30].

The first node in the network capable of finding a new valid block announces it to the network, and gets the reward associated to the block, which consists of the sum of the fees of the transaction included in the new block, plus a pre-defined quantity of newly mined coins, hence the name "mining" given to the process. The increased interest in the Bitcoin cryptocurrency, especially during the speculative bubble of late 2017 and early 2018, during which the value of a Bitcoin almost topped 20 000 $, has made the mining process extremely competitive, paving the way for an entire industry of dedicated miner accelerators [31], [32] with extremely high demands on the underlying SHA-256 circuitry. Not only must such circuits be fast enough to compete profitably within the peer-to-peer network, but they must also be power efficient, in order not to have energy costs exceed mining revenues [33], [34].

But, even more importantly, the success of Bitcoin has raised interest in the potential of the underlying blockchain technology and its applications beyond digital cryptocurrencies. In fact, investigating new blockchain applications is currently a flourishing research and development field [27]. The Bitcoin blockchain itself is rather limited in its applications, mainly because of its transaction script language which is by design Turing-incomplete [23]. Furthermore, the Nakamoto consensus protocol has raised criticism for the large amount of energy it dissipates in a computation yielding no actual result [35]. Therefore, several alternative blockchains have been proposed, and most of them have been implemented. However, only a few of these alternatives are actually completely independent blockchains, or *altchains*, the best known of which being Ethereum [36]. On the contrary, most of the so-called *second-generation blockchains* actually rely on the Bitcoin blockchain itself in some way, mainly in order to avoid destructive attacks from the powerful network of Bitcoin miners [28]. This means that the Bitcoin blockchain, with its reliance on hardware implementations of the SHA-256 algorithm, is today fundamental for the blockchain industry as a whole.

B. IoT

The Internet of Things (IoT) field requires carefully designed hardware accelerators for SHA-2 in order to meet security requirements within its manifold constraints. The IoT is based on infrastructures of low-end devices able to autonomously communicate across the Internet. It is an enabling paradigm for a number of innovative applications across different fields, each with its own security requirements [27]. Some cases, such as the biomedical sector [37], present strict security requirements due to the involvement of sensitive data, but almost any IoT application must face the threat of external attacks [27].

For instance, SHA-2 has been used in different ways to secure IoT devices and infrastructures for biomedical applications. One example is given in [38], where the pseudo-randomness of SHA-256 is exploited to produce a strong cryptographic key for the encryption of medical images which are to be sent over the cloud. It is worth mentioning that SHA-1 is still approved for use as PRNG [6]; however, authors of [38] opt for SHA-256 in order to obtain a longer, hence more secure, key. Another application is presented in [39], where the issue of black-hole routing attacks [40] is considered. The work includes source node authentication with an HMAC based on the SHA-256 hash function in the routing algorithm, based on the ideas presented in [41], in order to prevent unauthorized nodes from sending malicious routing packets.

An example of IoT application where sensitive data are not necessarily involved, but security is relevant anyway, is provided by Radio Frequency Identification (RFID) systems, which are increasingly used in supply chain management to intelligently track parts along the supply chains and manage inventories [42]. A small transponder called *tag* is attached to an item, storing a unique serial number for the item which can be sent to an RFID reader to track the object, or to start more complex interactions between the reader and the tag. The reading of RFID tags must be secured in order to preserve the privacy of both customers and companies [43]. Hence, a number of proposals for RFID access control have been put forward, mainly relying on hash functions and their one-way property. The scheme proposed in [43] avoids tracking by using a random number r in the tag response, which therefore consist of $(r||DM\ (ID||r))$. An improvement is proposed in [44], where transaction numbers are used to avoid replay attacks, and the tag ID is updated at every successful transaction in order to prevent traceability. The tag ID is updated also in the scheme of [42], where MACs are used to authenticate both the tag and the reader.

In addition, since the IoT paradigm implies a peer-to-peer network, and a blockchain is essentially a database distributed over a peer-to-peer network, there is an increasing interest in applying the blockchain technology to the IoT field [27], [45]–[47]. IoT applications can benefit from the use of a blockchain providing integrity and nonrepudiation of communications between the nodes [27], [45]. Furthermore, innovative IoT applications can be built taking advantage of *smart contracts*, which are applications run on a suitable blockchain capable of self-enforcing some business rules [46], or enabling the trading of IoT services directly by the nodes of the IoT network [48].

IoT-related security presents unique challenges due to the characteristics of the devices involved. In fact, a paramount issue in IoT applications is the energy consumption, since devices are usually battery-backed, and the battery life often determines the very life time of the device. Furthermore, passively-powered devices like RFID tags are limited in how much electrical *power* they can receive from the reader. This represents an interesting positioning in the design space, as the limiting factor is not posed by the mere energy budget but, inherently, by the instantaneous power that can be supplied. In a sense, this situation resembles the power cap issue incurred by high-end processors used in server settings, albeit scaled down to the deeply embedded realm. As a consequence, to meet their energy requirements, IoT devices are often limited in their computing capabilities [49]. On the other hand, this conflicts with the computational requirements of cryptographic primitives, like SHA-2, which are very demanding and place a significant overhead on IoT devices [43], [50], [51]. This trade-off reaches its extreme with blockchain-based applications [47], preventing normal IoT nodes from participating in the consensus protocol [46]. Clearly, only carefully designed application-specific accelerators can provide enough flexibility to address the conflicting constraints described above in an effective way.

C. TRUSTED COMPUTING

Hashing functions also have a central role in establishing some form of trustworthiness in dedicated computing platforms, from embedded devices up to servers and cloud systems relying on heterogeneous accelerators. In the trusted computing jargon [52], hashing is used to compute the so-called *measurement* of both hardware and software configurations, checking the full integrity of the system. These measurements can have various incarnations in real applications, beyond the strict definition given by trusted computing standards. Take as an example the emergence of FPGAs in the cloud. In the last few years, there has been an emerging interest in including reconfigurable hardware in cloud systems, as major cloud providers have started having FPGAs in their facilities [53]. This trend stems from the recognition that reconfigurable hardware implementations of commonly used algorithms can achieve better performance and improved power efficiency compared to classical CPU-based implementations [54]. Among the various challenges posed by the employment of reconfigurable hardware in cloud settings, there is the need to secure the provided bitstream. In fact, when the reconfigurable hardware is not under their direct control, users need to be guaranteed that the hardware accelerator synthesized on the remote FPGA is the one they submitted, and has not being tampered with, for example by adding backdoors for leaking user's data. This is particularly relevant when sensitive data is involved [55]. Moreover, it is also necessary to avoid that a single malicious FPGA brings down the entire facility [53]. Securing the provided bitstream implies that the accelerator running on the FPGA can be *trusted*, and therefore can operate with sensitive data with the guarantee that such data is properly handled. For example, [55] proposes an architecture for trusted FPGA in the cloud where a hardware SHA-2 accelerator is employed to authenticate the bitstream supplied to the FPGA. A similar system is employed in [56], where the use of trusted accelerators on reconfigurable hardware is proposed for preventing software running on the CPU from accessing sensitive data by offloading them to the FPGA, although in the latter work the components of the secure system are synthesized on the programmable part of the FPGA along with the user's design, where they are included through a dedicated toolchain.

Interestingly, reconfigurable hardware is also used to implement trusted execution environments (TEEs) similar to Intel SGX [57], mainly aiming to avoid the need for the manufacturer, e.g. Intel, to directly verify the code running in the TEE [58]. An example of such approaches is proposed in [59], where the case is made for hardware SHA-2 circuits into the trusted processor architecture in order to minimize the security overhead. Another proposal is outlined in [58], where a SHA-512 accelerator is employed as a PRNG.

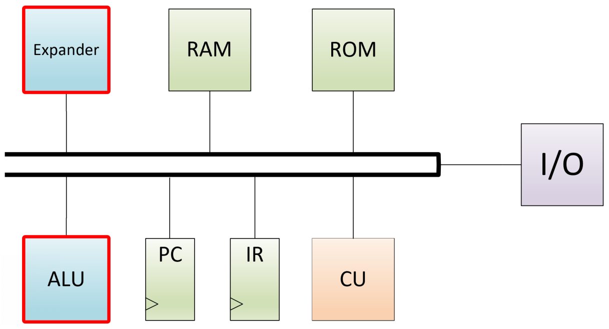

FIGURE 1. General architecture of a SHA-2 processor core. Custom designed components are highlighted.

IV. APPROACHES TO SHA-2 ACCELERATION

While software implementations of SHA-2 may be sufficient for scenarios with limited constraints, emerging applications pose stringent set of requirements that are best met by hardware acceleration. In fact, not only are hardware implementations more efficient in terms of performance and energy consumption compared to software implementations, but they also guarantee inherently better security. Table 2 presents a synoptic overview of the different approaches proposed in the literature to design SHA-2 hardware accelerators. At the highest level, hardware implementations of SHA-2 can be classified according to two design approaches [60]:

- *Programmable processor architectures*. These architectures resemble a general-purpose processor, with one or more central buses, an ALU, an instruction memory, and an ad-hoc microcode; but all these components are designed to perform only the computation of the SHA-2 hash algorithm.
- *Accelerator architectures*. Also referred to as *coprocessor* architectures, these implementations follow the approach of performing the entire computation directly in hardware.

A further classification is based on the implementation strategy for the computational core of the accelerator. Since there are many different strategies, and several optimisation techniques which are appropriate for different strategies, the whole Section V will focus on reviewing the different SHA-2 accelerator architectures proposed in the literature. The remainder of this section, on the other hand, will survey programmable processor architectures and describe their general components along with various optimisation techniques.

A. PROGRAMMABLE PROCESSOR ARCHITECTURES

The datapath of a SHA-2 processor implementation is shown in Figure 1. It includes the components commonly used in a processor design, such as an I/O module, a Control Unit, a Program Counter, a RAM memory used to hold the working variables and the message M to be hashed, and a ROM memory for the K_t constants and the initialisation values $DM_k(0)$. These components are arranged around a central bus, along with components designed specifically for the SHA-2 computation, such as a computation unit dedicated

TABLE 2. Synoptic overview of SHA-2 acceleration solutions.

Approach	Class	Optimisations	Literature Proposals
Programmable processor (Section IV-A)	Base (Section IV-A)	Base	[61]
		Component Improvement (Section IV-C1)	[60]
	Unrolled (Section IV-A)	Loop Unrolling (Section IV-C3)	[62]
Accelerator (Section IV-B)	Base (Section V-A)	Base	[63]–[66]
		Component Improvement (Section IV-C1)	[67]
		Loop Unrolling (Section IV-C3)	[68]–[71]
		Loop Folding (Section IV-C4)	[72], [73]
	Shift Register (Section V-B)	Base	[74]
		Component Improvement (Section IV-C1)	[75], [76]
		Loop Folding (Section IV-C4)	[77]–[80]
	Precomputation-Based (Section V-C)	Variable Precomputation (Section IV-C2)	[81]–[86]
		Component Improvement (Section IV-C1)	[87]
		Loop Unrolling (Section IV-C3)	[88], [89]
	Reordering-Based (Section V-D)	Spatial Reordering (Section IV-C5)	[90]
		Loop Unrolling (Section IV-C3)	[91]
		Component Improvement (Section IV-C1)	[1]
	Quasi-Pipelined (Section V-E)	Quasi-Pipelining (Section IV-C6)	[92]–[94]
		Loop Unrolling (Section IV-C3)	[95]

to the expansion of M to produce the W_t words, and one or more computation units for implementing the compressor function. A detailed description of the datapath of a processor architecture can be found in [61], where other elements commonly used in general-purpose processors, such as Memory Address Registers and Memory Data Registers, are also used.

To be executed by this kind of implementations, the SHA-2 algorithm must be described in terms of *micro-operations*, which are the operations directly executable by the architecture in a clock cycle. Since one iteration of the SHA-2 algorithm usually requires more than one micro-operation, processor architectures require multiple clock cycles to hash a single message, compared to accelerator architectures which take one cycle per SHA-2 iteration.

Improved processor architectures may rely on the optimisation of some of its components, particularly the arithmetic units. For example, in [60] a four-input ALU is used to compute Equations (II.6) and (II.7). A different strategy relies on multiple instances of the same unit, so as to parallelise the execution of different micro-operations. In [62] the number of cycles needed by a SHA-2 iteration is reduced by executing two micro-operations of the same iteration simultaneously, taking advantage of multiple arithmetic units.

B. ACCELERATOR ARCHITECTURES

Accelerator architectures are built around a combinatorial block, hereafter referred to as the *transformation round core*, which performs the SHA-2 step function as described in Section II-B3. The overall equation implemented by a transformation round core can be obtained by combining Equations (II.6), (II.7) and (II.11):

$$\begin{aligned}
A_{t+1} &= \Sigma_1(E_t) + Ch(E_t, F_t, G_t) + \Sigma_0(A_t) \\
&\quad + Maj(A_t, B_t, C_t) + H_t + K_t + W_t \\
B_{t+1} &= A_t \\
C_{t+1} &= B_t \\
D_{t+1} &= C_t \\
E_{t+1} &= \Sigma_1(E_t) + Ch(E_t, F_t, G_t) + D_t + H_t + K_t + W_t \\
F_{t+1} &= E_t \\
G_{t+1} &= F_t \\
H_{t+1} &= G_t
\end{aligned} \tag{IV.1}$$

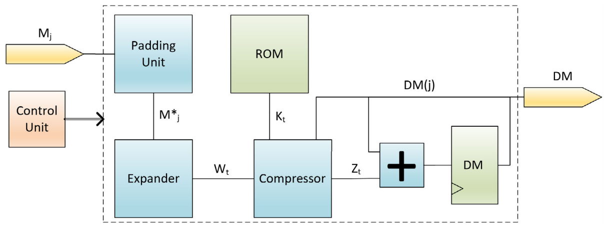

FIGURE 2. General architecture of a SHA-2 accelerator core.

1) GENERAL ARCHITECTURE

The architecture of a SHA-2 accelerator is directly derived from the structure of the algorithm, as described in Section II, and it is shown at a high level in Figure 2.

Prior to hashing, the incoming message needs to be padded. Therefore, a Padding Unit is responsible for producing the PDBs. It is worth noting that the Padding Unit modifies only the last block of a message, leaving the others untouched. Since padding can be performed in software quite efficiently, without affecting the overall security of the system, many implementations [1], [76], [79], [80], [84], [85], [92], [93], [96]–[99] do not include the Padding Unit. These implementations take in already formatted PDBs. On the other hand, other implementations [63]–[66], [71], [72], [74], [75], [82], [95] choose to include a Padding Unit directly in hardware. In this case, the original messages must be provided to the accelerator, while the PDBs will be built internally. When comparing reported results, especially for the area occupation metric, it must be taken into account whether a Padding Unit is included in the hardware design.

The PDB enters the Expander, which outputs the words W_t according to Equation (II.4). The implementation of the Expander is quite straightforward, and is based on a 16-position shift register to store the required words with the necessary delay. The input of the shift register chain is the result of the computation of Equation (II.4). Since the W_t value is not read from the input of the shift register, but from one of the internal delay registers, the Expander is decoupled from the critical path of the Compressor [85]. Most implementations adopt this architecture for the Expander, which is explicitly described in [65], [74], [75], [82], [96], [100] and shown in Figure 3. Possible improvements rely on special devices for implementing the shift register, such as block RAMs (BRAMs) or first-in, first-out memory queue (FIFO)s [85], or the shift register mode of the configurable logic blocks (CLBs) of Xilinx® FPGAs [81]. A folded implementation of the Expander, employing only one adder, is proposed instead in [78], [79]. The unrolled implementation of the Expander is presented in [91], [95]. In [93], an architectural optimisation of the Expander is proposed, based on the *delay balancing* technique. This technique aims to shorten the critical path by placing more combinatorial logic on noncritical paths. The effect of this reordering is that noncritical paths, which do not determine the worst-case delay of the circuit, are stretched, while the critical path is shortened, resulting in a reduction of the circuit delay.

A ROM unit is employed to store and supply the K_t constants. Multi-mode architectures [65], [82] take advantage of the fact that the values for SHA-256 are the most significative halves of the values for SHA-512, therefore the ROM for the SHA-256 constants can be used also in the SHA-512 datapath.

The current hash value needs to be retained until the end of the hashing of the current PDB, since it must be added to the value of the working state Z. In a straightforward implementation, eight adders are required to perform the final addition of Equation (II.13) simultaneously. However, in [84], [85] an efficient alternative technique is proposed to compute the new intermediate hash value without adding any latency, since it is calculated during the last stages.

The actual hash computation is performed within the Compressor, which is also where the critical path is located. The transformation round core is responsible for the computation of Equation (IV.1), and its straightforward implementation is shown in Figure 4. However, Equation (IV.1) must be computed R times for each PDB. There are different techniques that can be used to perform the whole computation from the transformation round core, which will be described next.

2) LOOP ROLLING

The *loop rolling* technique consists of implementing an iterative algorithm by using the same component to perform iteratively the same computation. A feedback loop is employed to forward the output of the component to its input, so as to re-apply the function performed by the component. Architectures employing this technique require $R + 1$ clock cycles to produce a hash value, since an additional clock cycle is employed for performing the last addition described by Equation (II.13).

3) PIPELINING

Hardware pipelining as used in many dedicated SHA-2 architectures consists of instantiating S blocks each performing a fraction of the total number of iterations. In other words, such architectures distribute the R rounds required by a single hash calculation onto S pipeline stages, each handling R/S iterations. However, within each pipeline stage, loop rolling is still employed to perform the R/S iterations. Relying on hardware replication, pipelined SHA-2 architectures increase the steady-state throughput, since they are capable of outputting a new hash value every R/S cycles, or even less if loop unrolling is also used: compared with the nonpipelined implementation employing a single transformation round core, the throughput improvement reaches a factor S. Furthermore, each stage can be optimized for the specific subset of rounds that it handles.

On the other hand, pipelined implementations incur an area occupation increase by a factor S, and an increase in power consumption due to the presence of more register elements. In addition, due to the fact that the computation of the intermediate hash value of the j-th PDB $DM(j)$ requires knowing the previous intermediate hash value $DM(j-1)$ according

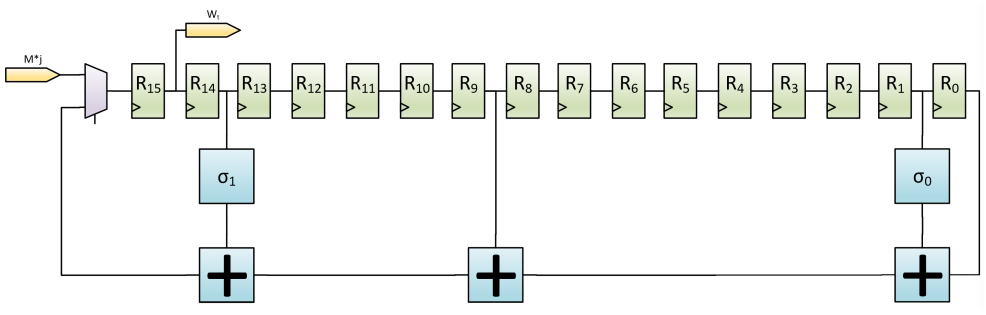

FIGURE 3. Straightforward architecture of the Expander.

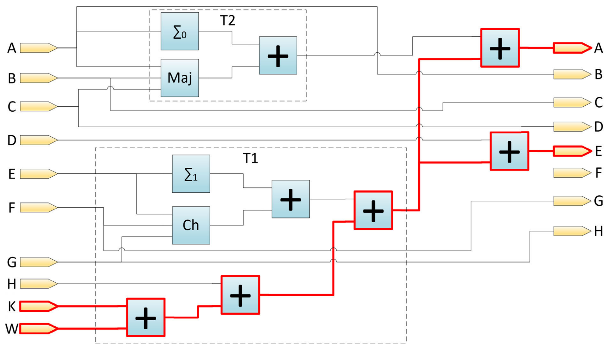

FIGURE 4. A straightforward implementation of the transformation round. The critical path is highlighted.

to Equation (II.3), and taking into account the fact that the latency of the computation of a single PDB is unaffected by the pipelining technique, which works on multiple PDBs in parallel, pipelined architectures cannot speed up the computation of a single message. For this reason, several proposals [101] choose to instantiate multiple loop-rolled SHA-2 cores instead of a single pipelined implementation. Nevertheless, the vast majority of recent and optimised SHA-2 architectures choose to employ pipelining, relying on the opportunity of processing different messages at once.

In pipelined SHA-2 architectures, the output of each stage, which is supplied as input to the following stage, is stored into the pipeline registers. This does not happen if the transformation round employs spatial reordering, since the pipeline register is moved across the architecture and it stores intermediate values rather than the output of the stage. In these architectures, the input to the following stage is provided directly by the combinatorial part of the previous stage.

C. OPTIMISATION TECHNIQUES

The implementation of the transformation round core can exploit a number of optimisation approaches. Usually, the architectures proposed in the literature rely on a mix of techniques taking advantage of their combined effect. In fact, some techniques are not particularly impactful by themselves, but turn out to be indispensable for the application of other techniques leading to the desired improvement. For example, loop unrolling and spatial reordering are often employed with the purpose of creating opportunities for the application of variable precomputation or component improvements.

This section discusses in isolation each of the techniques applied in the different hardware implementations of SHA-2. Section V will then illustrate how these techniques are combined in each full design proposal.

1) COMPONENT IMPROVEMENT

Performance can be improved by optimizing the single components which perform the basic operations of the algorithm. For the SHA-2 algorithms, this technique can be applied to the adders. Equation (IV.1) suggests that two-input adders, which usually are implemented by using fast carry-propagation schemes, can be replaced by three-input Carry Save Adders (CSA) which have slightly the same latency of a single two-input adder. However, this replacement is not always profitable, depending on other optimisations being in place, which may change the order of the performed operations [1].

2) VARIABLE PRECOMPUTATION

Some values can be computed earlier than strictly needed, if the inputs on which they depend are already available. If the computation is on the critical path, this can directly improve the frequency and hence the throughput. Precomputation can be applied to SHA-2 at two distinct scales.

Looking at the SHA-2 algorithm as a whole, we highlight that the message schedule described by Equation (II.4) does not depend on any intermediate result, and hence can be precomputed so as to make the proper message block available when needed. In fact, the first 16 values of W_t are available from the very beginning, and the time needed to compute a W_t value is usually less than the time required by Equation (IV.1). Moreover, due to the fact that the values of the constant K_t are known from the beginning of the computation, the precomputation of W_t allows for the precomputation of the sums $W_t + K_t$, as done in [1].

Precomputation can also be exploited within the transformation round core, by computing in the current step some values that are not immediately used, but will be consumed by some of the following iterations. This type of precomputation is also favoured by Equation (II.12), which implies that the values of the accumulator variables, except those that are computed in the current round, are available at least one round earlier. By repeatedly applying Equation (II.12), it turns out that some accumulators are actually available two or even three rounds in advance.

Some examples of this kind of precomputation, also called *operation rescheduling* [84], are illustrated in Section V-C; precomputation within the round also underpins the quasi-pipelining approach, discussed in Section IV-C6.

3) LOOP UNROLLING

The combinatorial block can perform more than one iteration of the algorithm in the same clock cycle. A transformation round core employing loop unrolling by a factor u computes u subsequent iterations in the same clock cycle, hence reducing the total number of iterations to R/u.

To achieve loop unrolling by a factor u, Equation (IV.1) must be rewritten to combine the results of u subsequent iterations $(t, t-1, \ldots, t-(u-1))$. Consider for instance an unrolling factor $u = 2$, meaning that the iterations t and $t-1$ are to be combined. Taking into account Equations (II.6) to (II.7), Equation (II.11) for step t can be written as

$$\begin{aligned}
A_{t+1} &= {T^1}_t\,(E_t, F_t, G_t, H_t, K_t, W_t) + {T^2}_t\,(A_t, B_t, C_t)\\
B_{t+1} &= A_t\\
C_{t+1} &= B_t\\
D_{t+1} &= C_t\\
E_{t+1} &= D_t + {T^1}_t\,(E_t, F_t, G_t, H_t, K_t, W_t)\\
F_{t+1} &= E_t\\
G_{t+1} &= F_t\\
H_{t+1} &= G_t
\end{aligned} \tag{IV.2}$$

and for step $t-1$ it can be written as

$$\begin{aligned}
A_t &= {T^1}_{t-1}\,(E_{t-1}, F_{t-1}, G_{t-1}, H_{t-1}, K_{t-1}, W_{t-1})\\
&\quad + {T^2}_{t-1}\,(A_{t-1}, B_{t-1}, C_{t-1})\\
B_t &= A_{t-1}\\
C_t &= B_{t-1}\\
D_t &= C_{t-1}\\
E_t &= D_{t-1}\\
&\quad + {T^1}_{t-1}\,(E_{t-1}, F_{t-1}, G_{t-1}, H_{t-1}, K_{t-1}, W_{t-1})\\
F_t &= E_{t-1}\\
G_t &= F_{t-1}\\
H_t &= G_{t-1}
\end{aligned} \tag{IV.3}$$

Combining Equation (IV.2) and Equation (IV.3) yields the equation expressing the value of the accumulators at iteration $t+1$ as functions of the value of the accumulators at the iteration $t-1$, which is the function performed by a transformation round core unrolled by a factor $u = 2$.

$$\begin{aligned}
A_{t+1} &= {T^1}_t(D_{t-1} + {T^1}_{t-1}(E_{t-1}, F_{t-1}, G_{t-1},\\
&\quad H_{t-1}, K_{t-1}, W_{t-1}), E_{t-1}, F_{t-1}, H_t, K_t, W_t)\\
&\quad \times {T^2}_t({T^1}_{t-1}(E_{t-1}, F_{t-1}, G_{t-1}, H_{t-1}, K_{t-1}, W_{t-1})\\
&\quad + {T^2}_{t-1}(A_{t-1}, B_{t-1}, C_{t-1}), A_{t-1}, B_{t-1})\\
B_{t+1} &= {T^1}_{t-1}(E_{t-1}, F_{t-1}, G_{t-1}, H_{t-1}, K_{t-1}, W_{t-1})\\
&\quad {T^2}_{t-1}(A_{t-1}, B_{t-1}, C_{t-1})\\
C_{t+1} &= A_{t-1}\\
D_{t+1} &= B_{t-1}\\
E_{t+1} &= C_{t-1} + {T^1}_t(D_{t-1} + {T^1}_{t-1}(E_{t-1}, F_{t-1}, G_{t-1},\\
&\quad H_{t-1}, K_{t-1}, W_{t-1}), E_{t-1}, F_{t-1}, H_t, K_t, W_t)\\
F_{t+1} &= D_{t-1} + {T^1}_{t-1}(E_{t-1}, F_{t-1}, G_{t-1}, H_{t-1}, K_{t-1}, W_{t-1})\\
G_{t+1} &= E_{t-1}\\
H_{t+1} &= F_{t-1}
\end{aligned} \tag{IV.4}$$

As shown by Equation (IV.4), loop unrolling increases the critical path, due to the addition of another level of functions T^1 followed by another sum in the computation of A_{t+1}. On the other hand, the number of iterations is reduced by a factor equal to the unrolling factor.

Furthermore, the unrolled loop may expose more opportunities for applying other optimisations that can reduce the critical path, further improving performance. For example, loop unrolling may expose the fact that some values are computed before the time when they are actually needed, and this circumstance enables the application of temporal precomputation [1].

4) LOOP FOLDING

Loop folding is the opposite transformation of loop unrolling, since it consists of splitting the execution of one iteration in multiple clock cycles. The advantage of doing so is the possibility of reusing the same functional block to perform different operations in the same iteration, hence reducing the total area occupation.

Usually, the architectures employing loop folding incur an increase in latency, due to the steep rise in the number of clock cycles required to perform the whole computation [78]. Nevertheless, [72] proposes a rescheduling of operations which avoids any increase in latency. This rescheduling takes into account the data dependencies in the SHA-2 algorithm, which prevent the simultaneous execution of all the additions.

5) SPATIAL REORDERING

In the architecture of the transformation round core, pipeline registers are located at the beginning of the computation to hold the inputs or, more frequently, at the end of the computation to store the outputs. The *spatial reordering* technique, firstly introduced in [102] for SHA-1, consists of moving the pipeline register in the middle of the round, more specifically in the best position so as to obtain a reduction in the critical path by splitting the round itself into balanced, parallel halves.

This optimisation, essentially a form of variable precomputation, is in fact also referred to as the *spatial precomputation* technique. It has the key advantage of avoiding any latency penalty, since there are no additional registers.

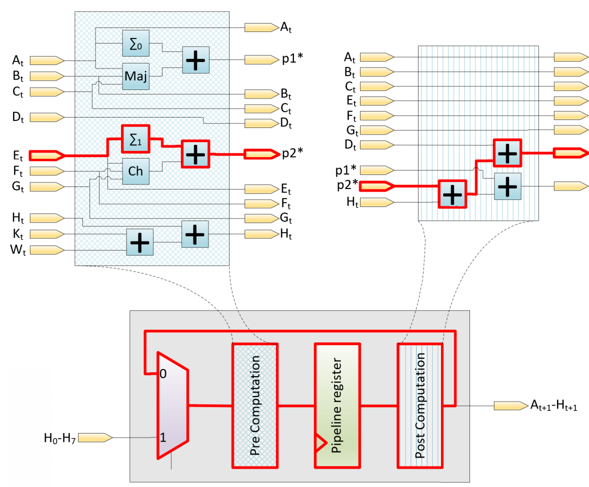

FIGURE 5. The architecture with spatial reordering proposed in [90]. The critical path is highlighted.

6) QUASI-PIPELINING

Quasi-pipelining is a technique aimed to introduce pipelining within the transformation round core, taking advantage of Equation (II.12) to perform data forwarding. The model has been formalised in [103] and can be potentially applied to any circuit which can be modeled as:

- a shift register chain of n positions R_i, $i \in [1, n]$;
- a number of combinatorial logic functions ϕ_i, including the identity, each of which taking as input one or more register values;
- a chain of combining operations, each of which being a commutative and associative binary operator to combine the results of the ϕ_i functions, feeding the shift register chain with the result.

Two additional registers are placed at the end of the chain for the K constant and the W expanded word, with index $n+1$ and $n+2$ respectively. The latency of the combining operators, which for SHA-2 always coincide with the modulo addition, is assumed greater than the latency of the ϕ_i functions, therefore the critical path runs from the R_{n+2} register through the combining operators chain, ending to the input of R_1.

In order to break this critical path, it may be first necessary to reorder the chain of combining operators, which is possible thanks to their commutativity and associativity properties. To this end, define the ϕ_i block as the couple of each ϕ_i operation and its associated combining operator.[3] Each ϕ_i block is associated with an index I_i constituted by the list of indices of the registers R_j which feed the ϕ_i function. The chain of combining operators can therefore be reordered by sorting the ϕ_i blocks according to the lexicographic order of their indices.[4] For SHA-2, the chain is already well-ordered. The critical path can now be broken into so-called *quasi-pipeline sections* Q_j, again according to the index I_i of the ϕ_i blocks. Namely, a quasi-pipeline section includes all the ϕ_i blocks sharing the same first number in their index I_i. Quasi-pipelined sections are finally separated by registers.

For SHA-2, the application of the quasi-pipelining technique requires the circuit to be split in two halves due to the feedback in the middle of the chain required to compute E_t. The quasi-pipelining technique is then applied as graphically shown in Figure 6 and described below:

$$\begin{aligned}
&\phi_1 = \Sigma_0(R_1) && \Rightarrow I_1 = \{1\} \\
&\phi_2 = Maj(R_1, R_2, R_3) && \Rightarrow I_2 = \{1, 2, 3\} \\
&\phi_3 = \Sigma_1(R_5) && \Rightarrow I_3 = \{5\} \\
&\phi_4 = Ch(R_5, R_6, R_7) && \Rightarrow I_4 = \{5, 6, 7\} \quad \Rightarrow \\
&\phi_5 = R_8 && \Rightarrow I_5 = \{8\} \\
&\phi_6 = R_9 && \Rightarrow I_6 = \{9\} \\
&\phi_7 = R_{10} && \Rightarrow I_7 = \{10\}
\end{aligned}$$

$$\Rightarrow \begin{aligned}
Q_1 &= \{\phi_1, \phi_2\} \\
Q_2 &= \{\phi_3, \phi_4\} \\
Q_3 &= \{\phi_5\} \\
Q_4 &= \{\phi_6\} \\
Q_5 &= \{\phi_7\}
\end{aligned} \tag{IV.5}$$

The quasi-pipeline sections can be optimised by employing the delay balancing technique, letting paths shorter than the critical one be stretched without incurring any performance penalty. For example, there is no need to separate Q_3 and Q_4, since both of them contain one modular addition, while Q_1 and Q_2 contain two modular additions. Similarly, there is no point in having Q_5 as a separate quasi-pipeline section, since it does not include any modular addition. The resulting quasi-pipeline sections are hence:

$$\begin{aligned}
Q_1 &= \{\phi_1, \phi_2\} \\
Q_2 &= \{\phi_3, \phi_4\} \\
Q_3 &= \{\phi_5, \phi_6, \phi_7\}
\end{aligned} \tag{IV.6}$$

Taking into account that the quasi-pipeline sections are filled in descending order, at each iteration t the quasi-pipeline section Q_j computes its part of round $t - q + j$, where q is the total number of quasi-pipeline sections. If $t < q - j$, quasi-pipeline section Q_j is not active, and its registers R_i are not clocked in order to fill the pipeline. However, unlike usual pipelining, all the quasi-pipeline sections operate on the same input registers R_i. This creates the need for data forwarding, which is allowed by the shift register configuration and the fact that the chain is lexicographically ordered. An array of *selecting functions* σ_i, i.e. multiplexers, is therefore added to perform data forwarding and completing the circuit.

V. SHA-2 ACCELERATOR ARCHITECTURES

The vast majority of hardware implementations of SHA-2 follow the coprocessor architecture approach. Despite the fact that it requires a higher design effort compared to the processor architecture approach, a coprocessor implementation can

[3]The ϕ_{n+2} block does not include a combining operator.

[4]$I_i \prec I_j$ if and only if I_i is a prefix of I_j or, possibly after a common prefix, the first differing number of I_i is less than the corresponding number of I_j

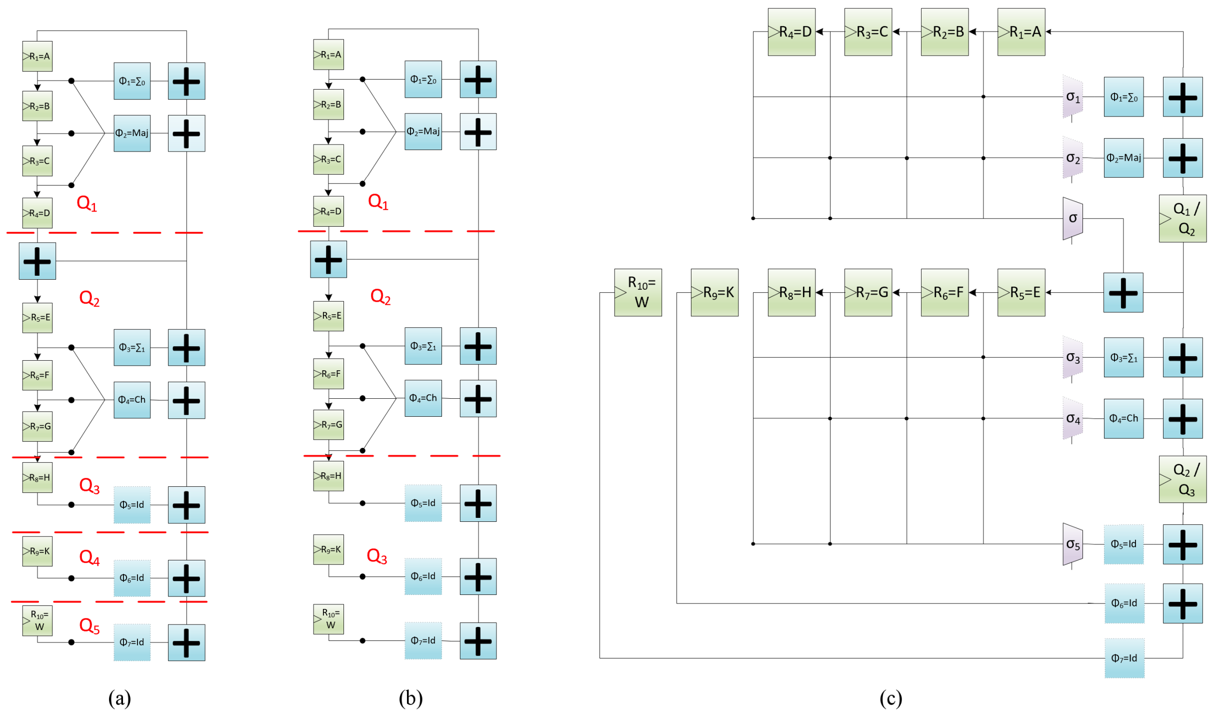

FIGURE 6. Application of Quasi-pipelining in SHA-2: (a) Quasi-pipelined model of SHA-2 with quasi-pipelined sections. (b) Final Quasi-pipelined Sections. (c) Complete Quasi-pipelined SHA-2 circuit. Light border denotes identity functions, which do not correspond to any actual circuits. Note that there are only two actual multiplexers in the final circuit.

achieve improved gains in terms of raw performance and area efficiency [96].

This section surveys the various SHA-2 architectural designs proposed in the technical literature, showing how each of them makes use of the techniques discussed in Section IV to achieve different optimization objectives.

A. BASIC ARCHITECTURES

The work in [63] proposes a hardware implementation of Equations (II.6) to (II.11) with dedicated logic within a transformation round unit. Since the equations employ only bitwise logic operations, shift operations and modulo additions, corresponding logic gates, bit reordering and modular adders are used to build up the transformation round unit. This is surrounded by a ROM, which provides it with the K_t constant values, and a number of support units. The Constants Unit supplies the transformation round core with the initialisation values DM. The initial values DM (0) are hard-coded in the FPGA LUTs, while for the subsequent PDBs the initialisation values DM provided by the Constants Unit are updated by the Last Transformation Unit. This unit includes an array of adders which perform Equation (II.13) to update the intermediate message digest. The W_t values are provided by a W_t - unit, which in turn is fed by the Padding Unit. A very similar architecture is implemented in [64] on a Xilinx Virtex-5 FPGA board, while [67] proposes the use of parallel adders to implement the additions leading to the computation of A_{t+1} and E_{t+1}.

The work in [65] implements a multi-mode variant of [63], where a Control Unit is in charge of reconfiguring the circuit to perform one of the different SHA-2 variants, according to the user's specifications. The most important aspect of [65] is the management of the different word widths of SHA-256 and SHA-512. This is tackled by clearing the least significant 32 bits of the datapath when SHA-256 is selected. The multi-mode architecture of [66] takes advantage of the similarities between MD5, SHA-1, and SHA-256 to support all of them. On the other hand, it does not support SHA-512, which is the case in [65]. This exclusion is due to the fact that SHA-512 works on 64-bit words, whereas [66] is a $32-$bit architecture.

A straightforward implementation of loop unrolling is presented in [68], where multiple instances of the transformation logic round are placed between registers. This allows for evaluating different values of the unrolling factor.

Loop unrolling by a factor 2 is also exploited in [69] with the aim of decreasing power consumption while increasing parallelism in the round function computation. This architecture is further optimized in [70] where multi-operand additions are compressed by using CSAs. The same optimisation techniques are exploited in [71], where a multi-mode architecture is proposed. This architecture is further optimised in [72] by observing that, due to data dependencies, the whole round function can be computed using only two CSAs without incurring any performance penalty, by properly scheduling the various additions.

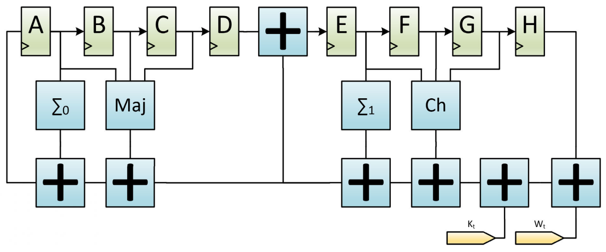

FIGURE 7. **Straightforward shift register architecture implementation.**

In [86], a reordering within the transformation round is proposed. The following variable is defined

$$\tau_t = H_t + K_t + W_t + D_t \tag{V.1}$$

and substituted into Equation (IV.1), leading to

$$\begin{aligned} A_{t+1} &= \Sigma_0(A_t) + Maj(A_t, B_t, C_t) \\ &\quad + \Sigma_1(E_t) + Ch(E_t, F_t, G_t) + \tau_t - D_t \\ E_{t+1} &= \Sigma_1(E_t) + Ch(E_t, F_t, G_t) + \tau_t \end{aligned} \tag{V.2}$$

This architecture is further improved in [87] with the addition of CSAs, while in [104] the application of loop unrolling is explored, with the unrolling factor 4 yielding the best results.

In order to achieve low power consumption, [73] reduces the number of adders and the simultaneously clocked register by employing the loop folding technique. Only one adder is used to perform all the operations of the Compressor and the Expander, which therefore do not operate in parallel and can be disconnected from the clock network accordingly.

B. SHIFT REGISTER ARCHITECTURES

The architecture proposed in [74] exploits Equation (II.12) in the implementation of Equation (IV.1). In fact, Equation (II.12) implies that the accumulators can be chained in a shift register configuration, where for E_{t+1} the incoming value of D_t is added with the output of the T_t^1 function. The shift register chain is supplied with $T_t^1 + T_t^2$ as input, which is the value of A_{t+1}. The coprocessor architecture proposed in [96] also follows the shift register approach, both for the SHA-256 and the full HMAC-SHA-256 implementations. The multi-mode variant of the shift register approach is presented in [98]. The shift register architecture can be optimised to utilise a single adder to perform the final sum. This is illustrated in [77], where an adder from the datapath is reused to this end.

The shift register approach is also adopted in [78], which is an architecture specifically tailored to low-power, area-constrained applications. The round function is implemented by reducing the number of operator blocks at the minimum, reusing the same operator block to perform multiple computations. The shift register architecture of the Compressor is therefore modified to work with a single adder, which subsequently adds different operands to compute the round function across several clock cycles, while the H_t register is used as the accumulator for the addition. Interestingly, a similar architecture is adopted also for the Expander, which requires four clock cycles to compute a word, compared with the seven clock cycles of the Compressor. The whole circuit requires 490 clock cycles to fully compute a hash, hence effectively trading throughput for area and power consumption. The multi-mode extension of this architecture is presented in [79].

Another proposal aimed at reducing area occupation, following the shift register architecture, is [100]. In this case, the area reduction is obtained by reducing the word size of the SHA-512 hash function, which is normally 64 bit, to a lower value, taking advantage of the fact that most of the operations involved in the hash algorithm can be computed in a bit-wise fashion. Implementations with the word size reduced to 32 bit, 16 bit and 8 bit are considered. Interestingly, the 32-bit variant achieves 72% of the throughput of the full-word-size implementation, meaning that it is more area efficient than the full-word-size counterpart. The architecture described in [99] also reduces the word width of the circuit to 8 bit, exploiting also loop folding to achieve further area reduction.

A different way to improve the shift register architecture involves the use of parallel adders for the adder chain. In [75] the adder chain is implemented with 5-to-3 parallel adders, while [76] employs a single 7-3-2 parallel adder to compute A_{t+1}. The architecture proposed in [80] combines the use of parallel adders with the loop folding technique, in order to reduce area occupation limiting the throughput penalty, therefore maximizing the area efficiency.

C. ARCHITECTURES WITH PRECOMPUTATION

The architecture proposed in [81] precomputes the sum $K_t + W_t$ outside the main operational block, in order to shorten the critical path. Moreover, it employs a 5-to-3 parallel adder to compute A_{t+1}. The architecture is further optimised in [88], where loop unrolling by a factor 5 is used. Values required later in the unrolled chain of operations are precomputed as soon as possible.

Variable precomputation is employed in [84] to shorten the critical path at the iteration t. Taking into account that $H_t = G_{t-1}$ due to Equation (IV.1), and that the values K_t and W_t are available in advance, the quantity

$$\delta_t = H_t + K_t + W_t = G_{t-1} + K_t + W_t \tag{V.3}$$

can be precomputed during round $t - 1$, leading to the following computation during round t:

$$\begin{aligned} A_{t+1} &= \Sigma_0(A_t) + Maj(A_t, B_t, C_t) \\ &\quad + \Sigma_1(E_t) + Ch(E_t, F_t, G_t) + \delta_t \\ E_{t+1} &= D_t + \Sigma_1(E_t) + Ch(E_t, F_t, G_t) + \delta_t \end{aligned} \tag{V.4}$$

This architecture is further improved in [85] with two optimised variants for the Expander, relying upon BRAMs and FIFOs respectively

The work in [86] pushes this approach further, proposing another precomputation based on the fact that $D_t = C_{t-1}$ again due to Equation (IV.1). The quantity τ_t defined in

Equation (V.1) can be precomputed during round $t-1$ as[5]

$$\tau_t = \delta_t + D_k = \delta_t + C_{t-1} \tag{V.5}$$

The computation of round t can therefore be reduced to

$$\begin{aligned} A_{t+1} &= \Sigma_0(A_t) + Maj(A_t, B_t, C_t) \\ &= \Sigma_1(E_t) + Ch(E_t, F_t, G_t) + \delta_t \\ E_{t+1} &= \Sigma_1(E_t) + Ch(E_t, F_t, G_t) + \tau_t \end{aligned} \tag{V.6}$$

In [86] three architectures are compared against the straightforward implementation of Equations (II.6) to (II.11) with a common platform, which employs a rolling loop to perform the whole SHA-256 computation. The three architectures considered in [86] include the scheme resulting from Equations (V.1) and (V.2) originally proposed in [84], the one resulting from Equations (V.3) and (V.4), and the one resulting from Equations (V.5) and (V.6).

The architectures are further optimised in [87] by employing Carry-Save Adders. Moreover, the common evaluation platform is improved by inserting two registers to break the critical path without requiring any additional clock cycle. These additional registers are located after the K constant ROM memory and before the DM feedback loop multiplexer.

The multi-mode architecture proposed by [82] also exploits the precomputation of δ_t and τ_t, computing these two in parallel as

$$\begin{aligned} \delta_t &= H_t + K_t + W_t \\ &= G_{t-1} + K_t + W_t \\ \tau_t &= H_t + K_t + W_t + D_t \\ &= G_{t-1} + K_t + W_t + C_{t-1} \end{aligned} \tag{V.7}$$

While the first equation is the same as Equation (V.3), the second equation is implemented by computing $G_{t-1} + C_{t-1}$ in parallel with the sum $K_t + W_t$, and the result of the latter is added, in parallel, to both the former and G_{t-1}.

The architecture presented in [83] exploits variable precomputation even further, in order to introduce a form of pipelining within the transformation round. In the first stage, Equation (V.3) is computed and the output is stored into a register, while the second stage computes Equation (II.6). A similarly aggressive precomputation is performed in [89] in the context of a two-unrolled architecture.

D. ARCHITECTURES WITH SPATIAL REORDERING

[90] introduces the use of spatial reordering within the design of the transformation round core for SHA-2. The computation of

$$\begin{aligned} P1^*_{t+1} &= \Sigma_0(A_t) + Maj(A_t, B_t, C_t) \\ P2^*_{t+1} &= \Sigma_1(E_t) + Ch(E_t, F_t, G_t) \\ H^*_{t+1} &= H_t + K_t + W_t \end{aligned} \tag{V.8}$$

[5]To stress the fact that Equation (V.5) is computed at round $t-1$ while Equation (V.1) is computed at round t, [86] calls the latter δ'_t instead of τ_t

is performed before the pipeline registers, while the computation of

$$\begin{aligned} A_{t+1} &= P1^*_t + P2^*_t + H^*_t \\ E_{t+1} &= D_t + P1^*_t \end{aligned} \tag{V.9}$$

is performed after the register, along with Equation (II.12). Because of this reordering, the critical path includes the two adders used for computing of A_{t+1} and the following adder for $P1^*_t$, along with the Maj function.

The same authors propose in [1] a methodology for the optimisation of the block responsible for the computation of the round function, building on the spatial reordering technique. This methodology takes also advantage of loop unrolling, component improvements and variable precomputation, and leads to the following computation ahead of the pipeline registers:

$$\begin{aligned} p1_{t+1} &= \Sigma_0(A_{t-1}) + Maj(A_{t-1}, B_{t-1}, C_{t-1}) \\ p2_{t+1} &= \big(D_{t-1} + H_{t-1} + (K+W)_{t-1}\big) \\ &\quad + \Sigma_1(E_{t-1}) + Ch(E_{t-1}, F_{t-1}, G_{t-1}) \\ p3_{t+1} &= D_{t-1} + (K+W)_{t-1} + \Sigma_1(E_{t-1}) \\ &= Ch(E_{t-1}, F_{t-1}, G_{t-1}) \\ p4_{t+1} &= (K+W)_t + G_{t-1} \\ p5_{t+1} &= p4_{t-1} + C_{t-1} \\ p6_{t+1} &= E_{t-1} + Ch(p2_{t+1}, E_{t-1}, F_{t-1}) \end{aligned} \tag{V.10}$$

In the above equation, $(K+W)$ denotes that the sum is precomputed outside the operational block, due to data prefetching. Note also that $p2_{t+1}$ is not computed from $p3_{t+1}$, due to the fact that the sum $D_{t-1}+H_{t-1}+(K+W)_{t-1}$ is performed by a Carry-Save Adder. The results of Equation (V.10) are stored in the pipeline registers along with A_{t-1}, B_{t-1}, E_{t-1}, and F_{t-1}, allowing for the following computation after the pipeline registers:

$$\begin{aligned} A_{t+1} &= \Sigma_0(B_{t+1}) + Maj(B_{t+1}, A_{t-1}, B_{t-1}) \\ &\quad + (p4_{t+1} + p6_{t+1} + \Sigma_1(p2_{t+1})) \\ E_{t+1} &= p6_{t+1} + \Sigma_1(p2_{t+1}) + p5_{t+1} \\ B_{t+1} &= p3_{t+1} + p1_{t+1} \qquad F_{t+1} = p2_{t+1} \\ C_{t+1} &= A_{t-1} \qquad G_{t+1} = E_{t-1} \\ D_{t+1} &= B_{t-1} \qquad H_{t+1} = F_{t-1} \end{aligned} \tag{V.11}$$

While the methodology in [1] proposes a sequence of techniques to be applied for obtaining an optimised hash core, their application is not always straightforward and requires a careful analysis of the circuit by the designer. The approach is further improved in [97], most notably by adding recursion, obtaining an even improved hash core. The latter version is evaluated against different FPGA platforms in [91], where the corresponding architecture of the Expander is also described. Finally, [105] presents a multi-mode hash accelerator based on the same techniques.

TABLE 3. Impact of SHA-2 optimisation techniques on evaluation metrics. Stronger impacts are highlighted.

Architectural Technique	Impact on Throughput	Impact on Area	Impact on Power	Complexity	Employed by
Pipelining	**positive**	**negative**	**negative**	**low**	[1], [70], [90], [91], [97]
Variables Precomputation	**positive**	negative	negative	**low**	[81], [84]–[86], [88] [82], [87], [89]
Loop Unrolling	positive	**negative**	**positive**	**low**	[1], [68], [89], [91], [97] [71], [72], [88]
Loop Folding	**negative**	**positive**	negative	**high**	[72], [73], [75], [78], [80], [99]
Spatial Reordering	**positive**	negative	negative	**high**	[1], [90], [91], [97]
Quasi - Pipelining	**positive**	negative	negative	**low**	[92]–[94], [103]

E. ARCHITECTURES WITH QUASI-PIPELINING

The first quasi-pipelined architecture for SHA-2 is introduced in [92], together with an improved variant employing delay-balancing. A slightly different version is presented in [93], where the Expander is also optimised with the delay balancing technique. An independent theoretical analysis carried out in [106] confirms that this design is optimal, at the architectural level, with respect to throughput.

Unrolling by factors 2 and 4 of the quasi-pipelining architecture is presented in [95], obtaining an improvement in throughput only for SHA-512 with unrolling factor 2.

Although it does not apply explicitly the quasi-pipelined formulation described in Section IV-C6, the work in [94] exploits the same basic idea of splitting the adder chain with registers and exploiting Equation (II.12) for data forwarding, here for the E_t accumulator. This work also employs a form of precomputation by summing the value K_t with the word W_t directly in the Expander, so as to remove this sum from the critical path.

VI. MATCHING DESIGN TECHNIQUES TO APPLICATION METRICS

Table 3 compares the various techniques described in Section IV in terms of their effects on performance, area occupation and energy efficiency, and the implementation complexity of each of them. The table also lists the most representative works employing each technique. As described in Section V, each design usually exploits more than one technique in order to meet the stated objectives.

Most design techniques are primarily aimed to performance, at the expense of increased area occupation and energy consumption. These approaches are best evaluated in terms of *area efficiency* or *area-delay product*, and *power efficiency* or *power-delay product*, in order to assess whether the price in terms of area occupation and energy consumption actually pays off.

A. PERFORMANCE

The most used metric to assess performance of hash circuits is the *throughput*, defined as the number of bits delivered per unit of time. If the hash circuit is capable of outputting a new hash value every N_{clk} clock cycles, and the clock period is τ_{clk}, the throughput can be written as

$$Q = \frac{L(DM(M))}{\tau_{clk} \cdot N_{clk}} = \frac{L(DM(M)) \cdot f_{clk}}{N_{clk}} \qquad \text{(VI.1)}$$

The number of bits of the output depends on the selected hash function and does not offer any degree of freedom for improving throughput, unless the designer is in the position of choosing which hash function to employ. For this reason, when comparing designs for the SHA-2 function with different hash sizes, it is preferable to refer to the *hash rate*, defined as

$$F = \frac{1}{\tau_{clk} \cdot N_{clk}} = \frac{f_{clk}}{N_{clk}} \qquad \text{(VI.2)}$$

The other two terms of Equation (VI.1) are instead fully dependent on architectural design decisions. The clock period is lower bounded by the critical path delay, hence it is directly influenced by the architectural techniques which impact the critical path. Variable precomputation, spatial reordering and quasi-pipelining all reduce the critical path, therefore they are beneficial for the throughput metric.

The number of clock cycles between two consecutive outputs is the latency required to compute the hash of a single PDB. For architectures capable to perform one iteration per clock cycle, N_{clk} is $R + 1$ unless the architecture can handle concurrently the final sum and the execution of the last iteration, such as [84], [85], or includes the adder in the critical path, such as [1], [97]; in such cases N_{clk} is R. Architectures with loop folding, on the other hand, require multiple clock cycles per iteration, making this technique negative for throughput.

Hardware pipelining, relying on S pipelined blocks each handling R/S iterations, does not modify the number of clock cycles required to produce a single hash message. Instead, it reduces N_{clk} by processing more PDBs simultaneously based on the availability of multiple units. The throughput is consequently increased by a factor equal to the number of pipeline stages.

Loop unrolling has a twofold effect on performance. On one hand, it directly reduces the number of cycles required to hash a message by computing multiple iterations in a single clock cycle. On the other hand, it substantially increases τ_{clk} due to the increased number of combinatorial levels required to compute the different iterations. The overall effect of loop unrolling hence depends on whether the reduction of the number of clock cycles compensates for the increase in the critical path. This does not happen in [88], while it happens in [1]. This discussion does not take into account the fact that loop unrolling can enable further transformations which can reduce the critical path, as mentioned in Section IV-C3.

Moreover, while the reduction of N_{clk} is platform-independent, the increase in τ_{clk} does depend on the underlying technology. This implies that the overall effect of loop unrolling on throughput is technology-dependent. In [2] loop unrolling turns out to be beneficial for throughput on fairly recent FPGAs.

B. AREA OCCUPATION

Area occupation refers to the size of the circuit when implemented in ASIC technology. On reconfigurable technologies such as FPGAs, area occupation refers to the utilisation of device resources [107]. It is a relevant factor for the final cost of the design, since a larger design requires a larger device for the physical implementation [108].

Usually, area occupation is traded off for performance. Techniques like pipelining and loop unrolling, which are aimed to increase throughput, have a severely adverse impact on area occupation, since the circuit must be replicated as many times as the number of pipelined blocks or the unrolling factor. However, unlike the case of multiple instances of the same circuit, only the datapath needs to be replicated. Conversely, loop folding is aimed to reduce area occupation, at the expense of degraded throughput.

Other techniques, such as variable precomputation, spatial reordering and quasi-pipelining, have a limited impact on the area occupation of SHA-2 accelerators since the additional resources utilised by these architectures are just a handful of registers.

C. POWER AND ENERGY CONSUMPTION

As with any digital system, energy consumption represents the main operational cost for SHA-2 hardware accelerators. However, interestingly, operational costs might not be the only factor of interest, as instantaneous power consumption may sometimes have strong implications on the physical design of the accelerator. In fact, the energy absorbed per unit of time by a circuit is dissipated as heat, which must be driven away from the hardware to avoid damaging the circuit by overheating [109]. The cost of cooling clearly increases with the amount of power to be driven away [110], while thermal requirements can even place an upper bound on the amount of functionality that can be integrated in a chip [111], no matter what the cost constraints are, leading to the so called power cap issues. Other applications, including passively-powered circuits, are limited in the amount of power they can draw. The constraint on the SHA-2 functionality placed by power supplying limitations is even tougher than the one posed by power dissipation, since there are no measures that can be used to mitigate the problem, unlike the case of cooling. In such occurrences, the input power constraint can only be met by means of architectural optimisations [43].

If N_M is the number of clock cycles required to hash a single message M by a hardware component consuming power P, the energy consumption of hashing a message can be written as

$$J_M = P \cdot \tau_{clk} \cdot N_M = \frac{P \cdot N_M}{f_{clk}} \tag{VI.3}$$

since $\tau_{clk} \cdot N_M$ is the time required to hash the message M. For architectures processing one message at a time, $N_M = N_{clk}$, while for architectures processing more than one message simultaneously, the identity $N_M = N_{clk}$ holds on average over multiple messages. Therefore, the average energy consumption per message hash can be written as

$$J = P \cdot \tau_{clk} \cdot N_{clk} = \frac{P}{F} \tag{VI.4}$$

with F being the hash rate, defined by Equation (VI.2).

The equation above suggests that the energy consumption can be reduced by either increasing the hash rate, or decreasing the power consumption. The hash rate has been analyzed in Section VI-A, so the remainder of this section will focus on power consumption. The power consumption of a circuit is usually divided into a static and dynamic component. The static component depends mainly on technological aspects, such as the power supply V, or the threshold voltage of the transistors, amongst others [112]. On the other hand, the dynamic component can be influenced by architectural decisions [113] and therefore by the particular techniques employed in designing the SHA-2 circuit. Energy is absorbed from the supply by a CMOS gate during a transition from 0 to V, and is dissipated as heat during the subsequent transition from V to 0. Therefore, only the former transition leads to energy consumption [110]. For this reason, the *switching activity* α is often defined as the probability of a transition from 0 to V within a clock cycle, avoiding a $1/2$ factor throughout the power consumption formulae.

At the gate level, the dynamic power consumption can be written as [110]:

$$P_d = \alpha \cdot f_{clk} \cdot C \cdot V^2 \tag{VI.5}$$

where C is the capacitive load of the gate. For FPGAs, the power consumption can be rewritten to take into account the utilisation U of each resource in the whole device after programming [114]:

$$P_d = f_{clk} \cdot V^2 \cdot \sum_i \alpha_i \cdot C_i \cdot U_i \tag{VI.6}$$

These formulae show the impact of the clock frequency on power consumption. Techniques that optimise throughput of the SHA-2 accelerator through decreasing the critical path end up increasing the power consumption, due to the corresponding increase of f_{clk}. This is the case of variable precomputation, spatial reordering, and quasi-pipelining.

As mentioned above, in massively parallel systems, increasing f_{clk} puts more pressure on the cooling system, whose dissipation capabilities may limit the maximum clock frequency, or poses an inherent limitation for passively-powered devices. On the other hand, this power increase does not lead to an energy-per-hash consumption increase, since f_{clk} also appears on the denominator of Equations (VI.3) and (VI.4). It is worth noting that if the optimisation of throughput is not due to a decrease in τ_{clk} but to a decrease in N_{clk}, this argument does not apply. In such cases, there is no power consumption penalty, and there is an energy-per-message reduction due to the throughput increase, as shown by Equation (VI.4). Techniques that increase throughput by decreasing N_{clk} include pipelining and loop unrolling.

Another factor impacting power consumption is area occupation [115], since more low-level resources need to be powered, and physical data and clock nets are longer [116]. This is shown also by Equation (VI.6). Therefore, designs which optimise resource utilisation, such as loop folding, also improve power efficiency. On the other hand, techniques leading to increased area occupation, such as pipelining and loop unrolling, also face increased power consumption. A more direct effect of architectural choices on power consumption is linked with the number of register operations [70], [117]. Since each register is read and written once per clock cycle, reducing the number of clock cycles needed to compute a hash also reduces the number of operations performed by registers, hence the dynamic power dissipation due to register operations. From this point of view, loop unrolling turns out to be beneficial in terms of power savings.

The overall effect to be expected on power consumption due to each design technique is summarized in Table 3. Pipelining has an adverse effect on power consumption, due to its increase in area and register operations. On the other hand, loop unrolling can be expected to be beneficial, despite the area increase, due to its frequency and register operation reduction. The experimental evaluation of [2] confirms the impact of loop unrolling, while other techniques appear not to have significant effects on power consumption. Loop folding has in fact twofold implications on power consumption. It leads to power savings due to the area reduction, but this also comes at the cost of increased register operations. The predominant effect ultimately depends on the underlying technology employed for the SHA-2 accelerator.

D. IMPLEMENTATION COMPLEXITY

This comparison criterium refers to the effort required for the designer to apply the architectural technique to a circuit, which is especially relevant in cases where a customized system is to be built. Pipelining and loop unrolling are structured techniques, that can also be applied automatically by modern CAD tools. Variable precomputation implies for the designer to break the critical path, which for SHA-2 is clearly located in the computation of A_{t+1}. Computations that do not depend on A_t and E_t can be moved ahead of the previous round. Quasi-pipelining is a more sophisticated technique, but it can be applied without too much effort as it is clearly documented in [103].

On the other hand, loop folding and spatial reordering require a significant intervention by the designer. For the former, the designer must identify which components can be shared, establish the schedule of operations, and then implement it. For the latter, it is up to the designer to balance the paths between the two halves resulting from reordering.

E. IMPACT ON APPLICATIONS

Different applications place different constraints on the underlying SHA-2 circuitry. Table 4 lists the constraints incurred by the applications discussed in Section III in order to suggest the optimisation techniques best suited for each application.

High-performance Web servers, providing security services relying on SHA-2, are mainly concerned with throughput, since they need to scale in the number of concurrent users they can serve. For this application, switching from software implementations to a dedicated crypto-accelerator may potentially bring substantial savings in terms of energy, i.e. operational costs. The need of processing different messages simultaneously, and the relatively large budget for circuit area and cost, indicate that pipelining, among other throughput-focused techniques, can be greatly beneficial for this class of applications.

Similarly, special applications like trusted computing support are loosely constrained, in that SHA-2 is performed rather infrequently and does not pose strict throughput or energy constraints, while the main concern is typically the area occupation, since the security scheme reduces the area available for the user logic. This can be addressed by using the loop folding technique.

Bitcoin mining rigs have far more stringent constraints, especially on throughput and energy consumption, and both of them must be fulfilled in order to design a profitable miner. Area requirements must also be kept under control, both for allowing parallel instances of the miner to be instantiated within the same device, and to contain the cost of the rig. The same goes for power, which can become a limiting factor in the design of a massively parallel architecture. This set of requirements can be addressed by using loop unrolling combined with throughput-oriented techniques such as variable precomputation or spatial reordering. In this way, the loop unrolling expands the applicability of the other techniques while delivering energy savings.

IoT applications typically feature heavily constrained low-cost devices. Area occupation of the SHA-2 circuit must therefore be contained for these applications. Most importantly, energy consumption must be kept as low as possible, in order not to waste the limited energy budget of these devices. Of course, porting mining within an IoT environment would add the requirement for high throughput and is normally performed by the most powerful elements in the network. In this context, techniques that deliver increased

TABLE 4. Requirements of surveyed applications relying on SHA-2 and recommended optimisations. The most important requirements, listed as critical, must be addressed by the designer with the utmost care, since even the sheer functionality of the application relies on satisfying such requirements. Slightly less critical requirements, still capable of influencing the quality of the overall solution, are listed as major. On the other hand, moderate requirements provide additional advantages to the application but are not essential, while less relevant or negligible requirements are listed as minor.

Application	Requirements				Recommended
	Throughput	Area	Energy	Power	Optimisations
Web server	moderate	minor	minor	minor	Pipelining Loop Unrolling Precomputation Spatial Reordering Quasi-Pipelining
Bitcoin mining	critical	major	critical	major	Loop Unrolling Precomputation Spatial Reordering Quasi-Pipelining
IoT (cryptography)	moderate	critical	critical	minor	Precomputation Spatial Reordering Quasi-Pipelining
RFID	minor	critical	minor	critical	Loop Folding
IoT (mining)	critical	critical	critical	major	Precomputation Spatial Reordering Quasi-Pipelining
Trusted computing	minor	moderate	minor	minor	Loop Folding

throughput of the accelerator with limited impact on area and throughput, such as variable precomputation, spatial reordering, and quasi-pipelining can be utilised.

For passively-powered devices such RFID tags, a limited amount of energy per unit of time can be delivered to the device, while the overall energy consumption, or even the duration of the operation is less of a concern. Therefore, the dominant constraint is on power rather than energy consumption or throughput. Moreover, the severe cost limitations of RFID applications result in strict constraints on area occupation, which suggests implementations based on the loop folding technique.

VII. CONCLUSION

Modern applications relying on SHA-2 have strict requirements in terms of performance, area, electrical energy or power, often involving multiple of these metrics in conflicting tradeoffs. In many cases, the constraints are so strict that a hardware implementation is the only viable option. Different design techniques have been proposed in the technical literature for SHA-2 hardware acceleration, among which the designer can choose to implement their own customized accelerator. Most of the available techniques are oriented to the optimisation of throughput at the price of increased energy consumption or area occupation, reflecting the fact that classical applications of SHA-2 belong to the field of network security, where interacting parties are traditionally supposed to have sufficient computing resources and electrical energy. This assumption no longer holds for emerging SHA-2 applications, making the resource-efficient acceleration of SHA-2 crucial in a number of contexts. Optimisation techniques that can deliver improvements in throughput without excessive increase in area occupation or energy consumption must be preferred in these cases. Nevertheless, in some situations area or energy constraints even force a reduction in terms of throughput. Sometimes, such as in passive-powered devices, the requirement is not on the energy budget, but on instantaneous power. In these occurrences, reducing the energy by shortening the computation time is not an option. Frequently, as shown by this survey, the requirements of the application in hand can only be met by a combination of multiple techniques. Therefore, it is vital to understand the influence of each design technique and its interplay with the others. As an example, loop unrolling turns out to be an optimisation-enabling technique which can be employed, if its associated area penalty is tolerable, in order to enable the application of other techniques. Having showcased positive and negative impacts of each design choice on the different circuit metrics, and ultimately on the overall application requirements, this article aims to serve as a general survey but also as a designer's guide for the selection of the best technique mixes to use while designing application-specific SHA-2 hardware accelerators.

REFERENCES

[1] H. Michail, A. Kakarountas, A. Milidonis, and C. Goutis, "A top-down design methodology for ultrahigh-performance hashing cores," *IEEE Trans. Dependable Secure Comput.*, vol. 6, no. 4, pp. 255–268, Oct. 2009.

[2] R. Martino and A. Cilardo, "A flexible framework for exploring, evaluating, and comparing SHA-2 designs," *IEEE Access*, vol. 7, pp. 72443–72456, 2019.

[3] *Secure Hash Standard (SHS)*, Standard FIPS 180-4, National Institute of Standards and Technology, U.S. Department of Commerce, Aug. 2015.

[4] X. Wang, H. Yu, and Y. L. Yin, "Efficient collision search attacks on SHA-0," in *Proc. 25th Annu. Int. Cryptol. Conf. (CRYPTO)*, 2005, pp. 1–16.

[5] *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*, Standard FIPS 202, National Institute of Standards and Technology, U.S. Department of Commerce, 2015.

[6] *NIST Policy on Hash Functions*. Accessed: Nov. 12, 2017. [Online]. Available: https://csrc.nist.gov/projects/hash-functions/nist-policy-on-hash-functions
[7] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 7th ed. London, U.K.: Pearson, 2017.
[8] *Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher*, Standard SP 800-67 Rev.2, National Institute of Standards and Technology, U.S. Department of Commerce, Nov. 2017.
[9] *Advanced Encryption Standard (AES)*, Standard FIPS 197, U.S. Department of Commerce, 2011.
[10] *Digital Signature Standard (DSS)*, Standard FIPS 186-4, U.S. Department of Commerce, Jul. 2013.
[11] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication," in *Proc. 16th Annu. Int. Cryptol. Conf. (CRYPTO)*. Berlin, Heidelberg: Springer, 1996, pp. 1–15.
[12] *The Keyed-Hash Message Authentication Code (HMAC)*, Standard FIPS 198-1, National Institute of Standards and Technology, U.S. Department of Commerce, Jul. 2008.
[13] S. Kent and S. Seo, *Security Architecture for the Internet Protocol*, document RFC 4301, Internet Request for Comments, Dec. 2005. [Online]. Available: https://tools.ietf.org/html/rfc4301
[14] S. Kent, *IP Authentication Header*, document RFC 4302, Internet Request for Comments, Dec. 2005. [Online]. Available: https://tools.ietf.org/html/rfc4302
[15] *IP Encapsulating Security Payload (ESP)*, document RFC 4303, Internet Request for Comments, Dec. 2005. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4303.txt
[16] *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*, Standard SP 800-90A Rev. 1, National Institute of Standards and Technology, U.S. Department of Commerce, Jun. 2015.
[17] X. Wang, Y. L. Yin, and H. Yu, "Finding collisions in the full SHA-1," in *Proc. 25th Annu. Int. Cryptol. Conf. (CRYPTO)*. Berlin, Germany: Springer, 2005, pp. 17–36.
[18] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, Y. Markov, A. Petit Bianco, and C. Baisse. (2017). *Announcing the First SHA1 Collision*. [Online]. Available: https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html
[19] R. Rivest, *The MD5 Message-Digest Algorithm*, document RFC 1321, 1992. [Online]. Available: https://www.ietf.org/rfc/rfc1321.txt
[20] X. Wang and H. Yu, "How to break MD5 and other hash functions," in *Proc. 21st Annu. Int. Conf. Theory Appl. Cryptograph. Techn. (EUROCRYPT)*. Berlin, Germany: Springer, 2005, pp. 19–35.
[21] J. Black, M. Cochran, and T. Highland, "A study of the MD5 attacks: Insights and improvements," in *Proc. 12th Int. Conf. Fast Softw. Encryption (FSE)*. Berlin, Germany: Springer, 2006, pp. 262–277.
[22] C. Palmer and R. Sleevi. (2014). Gradually Sunsetting SHA-1. Google. [Online]. Available: https://security.googleblog.com/2014/09/gradually-sunsetting-sha-1.html
[23] F. Tschorsch and B. Scheuermann, "Bitcoin and beyond: A technical survey on decentralized digital currencies," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 3, pp. 2084–2123, 3rd Quart., 2016.
[24] H. Gilbert and H. Handschuh, "Security analysis of SHA-256 and sisters," in *Proc. Int. Workshop Sel. Areas Cryptogr. (SAC)*, 2004, pp. 175–193.
[25] J. R. Douceur, "The sybil attack," in *Proc. 1st Int. Workshop Peer-Peer Syst. (IPTPS)*. Berlin, Germany: Springer-Verlag, 2012.
[26] S. Nakamoto. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*. [Online]. Available: https://bitcoin.org/bitcoin.pdf
[27] D. E. Kouicem, A. Bouabdallah, and H. Lakhlef, "Internet of Things security: A top-down survey," *Comput. Netw.*, vol. 141, pp. 199–221, Aug. 2018.
[28] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, "SoK: Research perspectives and challenges for bitcoin and cryptocurrencies," in *Proc. IEEE Symp. Secur. Privacy*, May 2015, pp. 104–121.
[29] N. T. Courtois, M. Grajek, and R. Naik, "Optimizing SHA256 in bitcoin mining," in *Proc. 3rd Int. Conf. Cryptogr. Secur. Syst. (CSS)* Berlin, Germany: Springer, 2014, pp. 131–144.
[30] M. Vilim, H. Duwe, and R. Kumar, "Approximate bitcoin mining," in *Proc. 53rd Annu. Design Autom. Conf. (DAC)*, 2016, pp. 1–6.
[31] M. B. Taylor, "The evolution of bitcoin hardware," *Computer*, vol. 50, no. 9, pp. 58–66, 2017.
[32] M. B. Taylor, "Bitcoin and the age of bespoke silicon," in *Proc. Int. Conf. Compil., Archit. Synth. Embedded Syst. (CASES)*, Sep. 2013, pp. 1–10.
[33] K. O'Dwyer and D. Malone, "Bitcoin mining and its energy footprint," in *Proc. 25th IET Irish Signals Syst. Conf. China-Ireland Int. Conf. Inf. Communities Technol. (ISSC/CIICT)*, 2014, pp. 280–285.
[34] S. Valfells and J. H. Egilsson, "Minting money with megawatts," *Proc. IEEE*, vol. 104, no. 9, pp. 1674–1678, Sep. 2016.
[35] P. Fairley, "Blockchain world–Feeding the blockchain beast if bitcoin ever does go mainstream, the electricity needed to sustain it will be enormous," *IEEE Spectr.*, vol. 54, no. 10, pp. 36–59, Oct. 2017.
[36] G. Wood. (2014). *Ethereum: A Secure Decentralised Generalised Transaction Ledger*. [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf
[37] S. M. Riazul Islam, D. Kwak, M. Humaun Kabir, M. Hossain, and K.-S. Kwak, "The Internet of Things for health care: A comprehensive survey," *IEEE Access*, vol. 3, pp. 678–708, 2015.
[38] R. Hamza, K. Muhammad, A. Kumar, and G. Ramirez-Gonzalez, "Hash based encryption for keyframes of diagnostic hysteroscopy," *IEEE Access*, vol. 6, pp. 60160–60170, 2018.
[39] A. Mathur, T. Newe, W. Elgenaidi, M. Rao, G. Dooly, and D. Toal, "A secure end-to-end IoT solution," *Sens. Actuators A, Phys.*, vol. 263, pp. 291–299, Aug. 2017.
[40] A. Wood and J. Stankovic, "Denial of service in sensor networks," *Computer*, vol. 35, no. 10, pp. 54–62, Oct. 2002.
[41] A. Mathur, T. Newe, and M. Rao, "Defence against black hole and selective forwarding attacks for medical WSNs in the IoT," *Sensors*, vol. 16, no. 1, p. 118, Jan. 2016.
[42] T. Dimitriou, "A lightweight RFID protocol to protect against traceability and cloning attacks," in *Proc. 1st Int. Conf. Secur. Privacy Emerg. Areas Commun. Netw. (SECURECOMM)*, 2005, pp. 59–66.
[43] S. A. Weis, S. E. Sarma, R. L. Rivest, and D. W. Engels, "Security and privacy of low-cost radio frequency identification systems," in *Proc. 1st Int. Conf. Secur. Pervas. Comput.*, 2003, pp. 201–212.
[44] D. Henrici and P. Müller, "Hash-based enhancement of location privacy for radio-frequency identification devices using varying identifiers," in *Proc. 2nd IEEE Annu. Conf. Pervas. Comput. Commun. Workshops*, Jun. 2004, pp. 149–153.
[45] A. Reyna, C. Martín, J. Chen, E. Soler, and M. Díaz, "On blockchain and its integration with IoT. Challenges and opportunities," *Future Gener. Comput. Syst.*, vol. 88, pp. 173–190, Nov. 2018.
[46] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the Internet of Things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.
[47] M. Conoscenti, A. Vetro, and J. C. De Martin, "Blockchain for the Internet of Things: A systematic literature review," in *Proc. IEEE/ACS 13th Int. Conf. Comput. Syst. Appl. (AICCSA)*, Nov. 2016, pp. 1–6.
[48] Y. Zhang and J. Wen, "The IoT electric business model: Using blockchain technology for the Internet of Things," *Peer-Peer Netw. Appl.*, vol. 10, no. 4, pp. 983–994, Jul. 2017.
[49] F. Samie, L. Bauer, and J. Henkel, "IoT technologies for embedded computing: A survey," in *Proc. 11th IEEE/ACM/IFIP Int. Conf. Hardw./Softw. Codesign Syst. Synthesis (CODES)*, Oct. 2016, pp. 1–10.
[50] L. Malina, J. Hajny, R. Fujdiak, and J. Hosek, "On perspective of security and privacy-preserving solutions in the Internet of Things," *Comput. Netw.*, vol. 102, pp. 83–95, Jun. 2016.
[51] M. El-Haii, M. Chamoun, A. Fadlallah, and A. Serhrouchni, "Analysis of cryptographic algorithms on IoT hardware platforms," in *Proc. 2nd Cyber Secur. Netw. Conf. (CSNet)*, Oct. 2018, pp. 1–5.
[52] T. C. Group, "Trusted platform module library specification, family '2.0,"' Trusted Comput. Group, Beaverton, OR, USA, Tech. Rep. 01.38, Sep. 2016.
[53] F. Chen, Y. Shan, Y. Zhang, Y. Wang, H. Franke, X. Chang, and K. Wang, "Enabling FPGAs in the cloud," in *Proc. 11th ACM Conf. Comput. Frontiers (CF)*, 2014, pp. 1–10.
[54] C. Kachris and D. Soudris, "A survey on reconfigurable accelerators for cloud computing," in *Proc. 26th Int. Conf. Field Program. Logic Appl. (FPL)*, Aug. 2016, pp. 1–10.
[55] K. Eguro and R. Venkatesan, "FPGAs for trusted cloud computing," in *Proc. 22nd Int. Conf. Field Program. Logic Appl. (FPL)*, Aug. 2012, pp. 63–70.
[56] B. Hong, H.-Y. Kim, M. Kim, T. Suh, L. Xu, and W. Shi, "FASTEN: An FPGA-based secure system for big data processing," *IEEE Design Test*, vol. 35, no. 1, pp. 30–38, Feb. 2018.
[57] V. Costan and S. Devadas. (2016). *Intel SGX Explained*. [Online]. Available: https://eprint.iacr.org/2016/086.pdf

[58] A. Coughlin, G. Cusack, J. Wampler, E. Keller, and E. Wustrow, "Breaking the trust dependence on third party processes for reconfigurable secure hardware," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays (FPGA)*, 2019, pp. 282–291.

[59] D. Evtyushkin, J. Elwell, M. Ozsoy, D. Ponomarev, N. A. Ghazaleh, and R. Riley, "Flexible hardware-managed isolated execution: Architecture, software support and applications," *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 3, pp. 437–451, May 2018.

[60] R. García, I. Algredo-Badillo, M. Morales-Sandoval, C. Feregrino-Uribe, and R. Cumplido, "A compact FPGA-based processor for the Secure Hash Algorithm SHA-256," *Comput. Electr. Eng.*, vol. 40, no. 1, pp. 194–202, Jan. 2014.

[61] J. Docherty and A. Koelmans, "A flexible hardware implementation of SHA-1 and SHA-2 hash functions," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2011, pp. 1932–1935.

[62] S. Dominikus, "A hardware implementation of MD4-family hash algorithms," in *Proc. 9th IEEE Int. Conf. Electron., Circuits, Syst. (ICECS)*, vol. 3, Jun. 2002, pp. 1143–1146.

[63] N. Sklavos and O. Koufopavlou, "On the hardware implementations of the SHA-2 (256, 384, 512) hash functions," in *Proc. Int. Symp. Circuits Syst. (ISCAS)*, Nov. 2003, p. 5.

[64] C. Jeong and Y. Kim, "Implementation of efficient SHA-256 hash algorithm for secure vehicle communication using FPGA," in *Proc. Int. SoC Design Conf. (ISOCC)*, Nov. 2014, pp. 224–225.

[65] N. Sklavos and O. Koufopavlou, "Implementation of the SHA-2 hash family standard using FPGAs," *J. Supercomput.*, vol. 31, no. 3, pp. 227–248, Mar. 2005.

[66] S. Ducloyer, R. Vaslin, G. Gogniat, and E. Wanderley, "Hardware implementation of a multi-mode hash architecture for MD5, SHA-1 and SHA-2," in *Proc. Conf. Design Archit. Signal Image Process. (DASIP)*, 2007, pp. 1–7.

[67] A. Mohamed and A. Nadjia, "SHA-2 hardware core for virtex-5 FPGA," in *Proc. IEEE 12th Int. Multi-Conf. Syst., Signals Devices (SSD)*, Mar. 2015, pp. 1–5.

[68] F. Crowe, A. Daly, T. Kerins, and W. Marnane, "Single-chip FPGA implementation of a cryptographic co-processor," in *Proc. IEEE Int. Conf. Field- Program. Technol.*, Mar. 2005, pp. 279–285.

[69] H. E. Michail, A. P. Karakountas, E. Fotopoulou, and C. E. Goutis, "High-speed and low-power implementation of hash message authentication code through partially unrolled techniques," in *Proc. 5th Int. Conf. Multimedia, Internet Video Technol. (MIV)*, 2005, pp. 130–135.

[70] K. Aisopos, A. Kakarountas, H. Michail, and C. Goutis, "High throughput implementation of the new secure hash algorithm through partial unrolling," in *Proc. IEEE Workshop Signal Process. Syst. Design Implement.*, Jan. 2006, pp. 99–103.

[71] M. Zeghida, B. Bouallegue, A. Baganne, M. Machhout, and R. Tourki, "A reconfigurable implementation of the new secure hash algorithm," in *Proc. 2nd Int. Conf. Availability, Rel. Secur. (ARES)*, Apr. 2007, pp. 281–285.

[72] M. Zeghid, B. Bouallegue, M. Machhout, A. Baganne, and R. Tourki, "Architectural design features of a programmable high throughput reconfigurable SHA-2 Processor," *J. Inf. Assurance Secur.*, vol. 3, pp. 147–158, Jan. 2008.

[73] M. Feldhofer and C. Rechberger, "A case against currently used hash functions in RFID protocols," in *Proc. Move Meaningful Internet Syst. Workshop (OTM)*, 2006, pp. 372–381.

[74] M. Mcloone and J. Mccanny, "Efficient single-chip implementation of SHA-384 and SHA-512," in *Proc. IEEE Int. Conf. Field-Programmable Technol. (FPT)*, Oct. 2003, pp. 311–314.

[75] W. Sun, H. Guo, H. He, and Z. Dai, "Design and optimized implementation of the SHA-2(256, 384, 512) hash algorithms," in *Proc. 7th Int. Conf. ASIC*, Oct. 2007, pp. 858–861.

[76] L. Bai and S. Li, "VLSI implementation of high-speed SHA-256," in *Proc. IEEE 8th Int. Conf. ASIC*, Oct. 2009, pp. 131–134.

[77] A. Satoh and T. Inoue, "ASIC-hardware-focused comparison for hash functions MD5, RIPEMD-160, and SHS," *Integration*, vol. 40, no. 1, pp. 3–10, Jan. 2007.

[78] M. Kim, J. Ryou, and S. Jun, "Efficient hardware architecture of SHA-256 algorithm for trusted mobile computing," in *Proc. Int. Conf. Inf. Secur. Cryptol. (INSCRYPT)*, 2009, pp. 240–252.

[79] M. Kim, D. G. Lee, and J. Ryou, "Compact and unified hardware architecture for SHA-1 and SHA-256 of trusted mobile computing," *Pers. Ubiquitous Comput.*, vol. 17, no. 5, pp. 921–932, Jun. 2013.

[80] M. M. Wong, V. Pudi, and A. Chattopadhyay, "Lightweight and high performance SHA-256 using architectural folding and 4-2 adder compressor," in *Proc. IFIP/IEEE Int. Conf. Very Large Scale Integr. (VLSI-SoC)*, Oct. 2018, pp. 95–100.

[81] T. Grembowski, R. Lien, K. Gaj, N. Nguyen, P. Bellows, J. Flidr, T. Lehman, and B. Schott, "Comparative analysis of the hardware implementations of hash functions SHA-1 and SHA-512," in *Proc. Int. Conf. Inf. Secur. (ISC)*, 2002, pp. 75–89.

[82] R. Glabb, L. Imbert, G. Jullien, A. Tisserand, and N. Veyrat-Charvillon, "Multi-mode operator for SHA-2 hash functions," *J. Syst. Archit.*, vol. 53, nos. 2–3, pp. 127–138, Feb. 2007.

[83] H. A. Tuan, K. Yamazaki, and S. Oyanagi, "Three-stage pipeline implementation for SHA2 using data forwarding," in *Proc. 18th Int. Conf. Field Program. Logic Appl. (FPL)*, 2008, pp. 29–34.

[84] R. Chaves, G. Kuzmanov, L. Sousa, and S. Vassiliadis, "Improving SHA-2 hardware implementations," in *Proc. 8th Workshop Cryptograph. Hardw. Embedded Syst. (CHES)*, 2006, pp. 298–310.

[85] R. Chaves, G. Kuzmanov, L. Sousa, and S. Vassiliadis, "Cost-efficient SHA hardware accelerators," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 16, no. 8, pp. 999–1008, Aug. 2008.

[86] I. Algredo-Badillo, C. Feregrino-Uribe, R. Cumplido, and M. Morales-Sandoval, "Novel hardware architecture for implementing the inner loop of the SHA-2 algorithms," in *Proc. 14th Euromicro Conf. Digit. Syst. Design*, Aug. 2011, pp. 543–549.

[87] I. Algredo-Badillo, C. Feregrino-Uribe, R. Cumplido, and M. Morales-Sandoval, "FPGA-based implementation alternatives for the inner loop of the Secure Hash Algorithm SHA-256," *Microprocessors Microsyst.*, vol. 37, nos. 6–7, pp. 750–757, Aug. 2013.

[88] R. Lien, T. Grembowski, and K. Gaj, "A 1 Gbit/s partially unrolled architecture of hash functions SHA-1 and SHA-512," in *Proc. Cryptographers' Track RSA Conf. (CT-RSA)*, 2004, pp. 324–338.

[89] F. Aisopos, K. Aisopos, D. Schinianakis, H. Michail, and A. Kakarountas, "A novel high–throughput implementation of a partially unrolled SHA-512," in *Proc. IEEE Medit. Electrotech. Conf. (MELECON)*, Aug. 2006, pp. 61–65.

[90] H. Michail, A. Milidonis, A. Kakarountas, and C. Goutis, "Novel high throughput implementation of SHA-256 hash function through pre-computation technique," in *Proc. 12th IEEE Int. Conf. Electron., Circuits Syst.*, Dec. 2005, pp. 1–4.

[91] G. S. Athanasiou, C. E. Goutis, G. Theodoridis, and H. E. Michail, "Optimising the SHA-512 cryptographic hash function on FPGAs," *IET Comput. Digit. Techn.*, vol. 8, no. 2, pp. 70–82, 2014.

[92] L. Dadda, M. Macchetti, and J. Owen, "The design of a high speed ASIC unit for the hash function SHA-256 (384, 512)," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, vol. 3, 2004, pp. 70–75.

[93] L. Dadda, M. Macchetti, and J. Owen, "An ASIC design for a high speed implementation of the hash function SHA-256 (384, 512)," in *Proc. 14th ACM Great Lakes Symp. VLSI (GLSVLSI)*, 2004, pp. 421–425.

[94] K. K. Ting, S. C. L. Yuen, K. H. Lee, and P. H. W. Leong, "An FPGA based SHA-256 processor," in *Proc. 12nd Int. Conf. Field Program. Logic Appl. (FPL)*, 2002, pp. 577–585.

[95] R. Mcevoy, F. Crowe, C. Murphy, and W. Marnane, "Optimisation of the SHA-2 family of hash functions on FPGAs," in *Proc. IEEE Comput. Soc. Annu. Symp. Emerg. VLSI Technol. Archit. (ISVLSI)*, Mar. 2006, pp. 317–322.

[96] M. Juliato and C. Gebotys, "Tailoring a reconfigurable platform to SHA-256 and HMAC through custom instructions and peripherals," in *Proc. Int. Conf. Reconfigurable Comput. FPGAs*, Dec. 2009, pp. 195–200.

[97] H. E. Michail, G. S. Athanasiou, V. Kelefouras, G. Theodoridis, and C. E. Goutis, "On the exploitation of a high-throughput SHA-256 FPGA design for HMAC," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 5, no. 1, pp. 2:1–2:28, 2012.

[98] M. Khalil, M. Nazrin, and Y. Hau, "Implementation of SHA-2 hash function for a digital signature system-on-chip in FPGA," in *Proc. Int. Conf. Electron. Design*, Dec. 2008, pp. 1–6.

[99] X. Cao, L. Lu, and M. O'Neill, "A compact SHA-256 architecture for RFID tags," in *Proc. 22nd IET Irish Signals Syst. Conf. (ISSC)*, 2011, pp. 6–11.

[100] I. Ahmad and A. Shoba Das, "Hardware implementation analysis of SHA-256 and SHA-512 algorithms on FPGAs," *Comput. Electr. Eng.*, vol. 31, no. 6, pp. 345–360, Sep. 2005.

[101] M. Togan, A. Floarea, and G. Budariu, "Design and implementation of cryptographic modules on FPGA," in *Proc. Eur. Conf. Appl. Math. Informat.*, 2010, pp. 149–154.

[102] I. Yiakoumis, M. Papadonikolak, H. Michail, A. Kakarountas, and C. Goutis, "Maximizing the hash function of authentication codes," *IEEE Potentials*, vol. 25, no. 2, pp. 9–12, Mar. 2006.
[103] M. Macchetti and L. Dadda, "Quasi-pipelined hash circuits," in *Proc. 17th IEEE Symp. Comput. Arithmetic (ARITH)*, Jul. 2005, pp. 222–229.
[104] I. Algredo-Badillo, M. Morales-Sandoval, C. Feregrino-Uribe, and R. Cumplido, "Throughput and efficiency analysis of unrolled hardware architectures for the SHA-512 hash algorithm," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, Aug. 2012, pp. 63–68.
[105] H. Michail, G. Athanasiou, G. Theodoridis, and C. Goutis, "On the development of high-throughput and area-efficient multi-mode cryptographic hash designs in FPGAs," *Integration*, vol. 47, no. 4, pp. 387–407, Sep. 2014.
[106] Y. K. Lee, H. Chan, and I. Verbauwhede, "Iteration bound analysis and throughput optimum architecture of SHA-256 (384,512) for hardware implementations," in *Proc. Int. Workshop Inf. Secur. Appl. (WISA)*, 2007, pp. 102–114.
[107] L. Deng, K. Sobti, Y. Zhang, and C. Chakrabarti, "Accurate area, time and power models for FPGA-based implementations," *J. Signal Process. Syst.*, vol. 63, no. 1, pp. 39–50, Apr. 2011.
[108] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 2, pp. 203–215, Feb. 2007.
[109] V. Tiwari, D. Singh, S. Rajgopal, G. Mehta, R. Patel, and F. Baez, "Reducing power in high-performance microprocessors," in *Proc. 35th Design Autom. Conf. (DAC)*, 1998, pp. 732–737.
[110] A. Chandrakasan and R. Brodersen, "Minimizing power consumption in digital CMOS circuits," *Proc. IEEE*, vol. 83, no. 4, pp. 498–523, Apr. 1995.
[111] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low-power CMOS digital design," *IEEE J. Solid-State Circuits*, vol. 27, no. 4, pp. 473–484, Apr. 1992.
[112] A. Amara, F. Amiel, and T. Ea, "FPGA vs. ASIC for low power applications," *Microelectron. J.*, vol. 37, no. 8, pp. 669–677, Aug. 2006.
[113] F. Ge, P. Jain, and K. Choi, "Ultra-low power and high speed design and implementation of AES and SHA1 hardware cores in 65 nanometer CMOS technology," in *Proc. IEEE Int. Conf. Electro/Inf. Technol.*, Jun. 2009, pp. 405–410.
[114] L. Shang, A. S. Kaviani, and K. Bathala, "Dynamic power consumption in Virtex-II FPGA family," in *Proc. 10th ACM/SIGDA Int. Symp. Field Program. Gate Arrays (FPGA)*, 2002, pp. 157–164.
[115] L. Benini and G. De Micheli, "Static assignment for low power dissipation," *IEEE J. Solid-State Circuits*, vol. 30, no. 3, pp. 158–268, Mar. 1995.
[116] L. Deng, K. Sobti, and C. Chakrabarti, "Accurate models for estimating area and power of FPGA implementations," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, Mar. 2008, pp. 1417–1420.
[117] H. Michail, A. Kakarountas, O. Koufopavlou, and C. Goutis, "A low-power and high-throughput implementation of the SHA-1 hash function," in *Proc. IEEE Int. Symp. Circuits Syst.*, Jul. 2005, pp. 4086–4089.

RAFFAELE MARTINO (Graduate Student Member, IEEE) received the B.Sc. and M.Sc. degrees *(magna cum laude)* in computer engineering from the University of Naples Federico II, in 2013 and 2016, respectively, where he is currently pursuing the Ph.D. degree in information technology and electrical engineering with the Department of Electrical Engineering and Information Technologies. He has been involved in a few European projects funded under the Horizon 2020 research and innovation programme, and he currently collaborates with the Department of Agriculture, University of Naples Federico II. His research interests include hardware architecture, compilers, and GPU programming.

ALESSANDRO CILARDO (Senior Member, IEEE) received the degree *(magna cum laude)* in electronics engineering, in 2003, and the Ph.D. degree in computer science, in November 2006. He is currently an Associate Professor with the University of Naples Federico II. He is the single or main author of around 80 peer-reviewed papers published in leading scientific journals and conferences, including various the IEEE and ACM transactions, as well as top conferences like DATE and FPL. His researches focus on digital design methodologies, and the application of programming paradigms and tools from the parallel computing domain to electronic system-level design. A further research activity in the area of computer arithmetic targets the application domain of security and cryptography-related processing. He is a Senior Member of the European Network of Excellence on High Performance and Embedded Architecture and Compilation (HiPEAC). He is involved in a number of funded projects at both the national level and the European level (7FP and H2020 projects).