A Power-efficient Implementation of SHA-256 Hash Function for Embedded Applications

Binh Kieu-Do-Nguyen[1,2], Trong-Thuc Hoang[2], Cong-Kha Pham[2], Cuong Pham-Quoc[1,3]
[1]Ho Chi Minh City University of Technology (HCMUT), Vietnam
[2]The University of Electro-Communication, Tokyo, Japan
[3]Vietnam National University - Ho Chi Minh City, Vietnam
Email: {binh,thuc}@vlsilab.ee.uec.ac.jp; phamck@uec.ac.jp; cuongpham@hcmut.edu.vn

***Abstract*—SHA-256 is a well-known algorithm widely used in many security applications. The algorithm provides a sufficient level of safety and can be performed efficiently by FPGA devices due to its high parallelism level. This paper presents a high-throughput, low hardware resources usage, and power-efficiency architecture of the SHA-256 algorithm targeting FPGA-based embedded platforms. The SHA-256 computing core takes advantage of the specific architecture of FPGA to achieve high performance. We implement the SHA-256 computing core with hardware description languages so that the computing core is technology-independent. Therefore, the computing core is suitable for building applications with various FPGA-based platforms. We conduct several experiments with both simulation and SoC boards. The experimental results show that the core achieves the same functionality, performance, and power consumption when implemented on different FPGA families. The implemented system with our SHA-256 computing core can function at 139.04 MHz, achieving a bandwidth of up to 1.04 Gbps. The SHA-256 computing core is power-efficient when consuming only 0.072 W with the minimum configuration.**

I. Introduction

Nowadays, hash functions play an essential role in many cryptography algorithms [1], such as the Digital Signature Standard (DSS) [2], Transport Layer Security (TLS) [3], Internet Protocol Security (IPSec) [4], etc. With embedded applications such as (Wireless Sensor Network [5], Trusted Mobile Platform [6], etc.), they help the communication among different parts of the system become more confident and reliable. As an instance of well-known hash functions, Secure Hash Algorithm (SHA) is a Secure Hash Standard [7] defined by the National Institute of Standard and Technology (NIST). Among the SHA-2 standards, SHA-256 is the most common and suitable for embedded applications. The result of the SHA-256 function is a 256-bits digest. The digest and the original message are then transferred from a source to a destination. Both are checked at the destination side to validate the original message. The SHA-256 algorithm offers an excellent opportunity to be accelerated by FPGA-based devices due to a combination of great numbers of basic logic operations, such as addition, rotation, and shift. Consequently, FPGA devices can perform them in parallel to improve performance.

This paper targets an SHA-256 computing core architecture and implementation to deploy on FPGA platforms. Our proposed architecture is technology-independent so that the computing core can function well in both Intel Altera- and Xilinx-based platforms with high-throughput, power-efficiency, and low hardware resource usage. To reach the goal, we apply many optimization techniques in the hardware description as follows.

1) Use a shift register and one-hot encoding to reduce the resource usage and power consumption of the Finite State Machine;
2) Use 3-input Carry Save Adders (CSAs) Adders to reduce the combinational logic delay of the adders circuits;
3) Apply the pipeline architecture for computing stages of the SHA-256 process;
4) Design the hardware to support the pipeline processing of the two consecutive input messages;
5) Promote the parallelism of the system to perform operations with a high parallel level;
6) Keep unused registers stable during the calculation process to reduce the dynamic power of the system.

Input and output signals of our proposed SHA-256 computing core prefer the data streaming form due to a limited number of control signals offered. In addition, the computing core is compatible with both the AXI bus of Xilinx FPGA chips or the Avalon bus of Intel Altera FPGA chips. The design is optimized and implemented without using any vendor-based IP cores. Therefore, it can be synthesized for various FPGA platforms without any modification.

Along with conducting several experiments to demonstrate the correctness and soundness of the proposed computing core, we also evaluate the power consumption and bandwidth. The SHA-256 computing core can work with multiple frequencies to reveal the trade-off between power usage and performance. The computing core achieves a speed-up of $14.45\times$ when compared with an ARM-based General Purpose Processor and $1.26\times$ when compared with a high-performance x86 CPU.

The rest of the paper is organized as follows. Section II presents the background of the SHA-256 algorithm and the previous work on the topic in the literature. Section III reveals the optimized architecture of our SHA-256 computing core. We evaluate the proposed computing core under different aspects, including working frequency, bandwidth, resource utilization, power consumption, and performance in Section IV.

*Corresponding authors: Cuong Pham-Quoc & Binh Kieu-Do-Nguyen

We also compare our design with other proposals in this section. Finally, Section V concludes the paper.

II. Background and Related Work

In this section, we first present the background of the SHA-256 algorithm. Based on the algorithm, we design the FPGA-based computing core. We also introduce a literature review on the research topic in this section.

A. The SHA-256 algorithm

SHA-256 is the most well-known hash function of the SHA-2 family. The SHA-256 algorithm satisfies all features of the FIPS PUB 180-4 standard. The produced hash code is unique for an original message, i.e., there does not exist any mathematic model to find the original data from the generated hash value. There do not exist two different messages that produce the same hash code. Any change in the original message leads to a significant update in the hash code.

The SHA-2 family includes four members. Those are SHA-224, SHA-256, SHA-384, and SHA-512. They are classified according to the length of the input messages, the number of rounds for the compression step, the initialization vectors, and the size of the final digest (also known as hash code). The SHA-256 algorithm accepts original messages with lengths of up to 2^{64} bits to produce the last 256-bit digest. Before hashing, an original message needs to be split into multiple 512-bit chunks. When the length of the message is not a multiple of 512 bits, the final chunk is padded. In the SHA-256 hash function, each chunk is processed within 64 rounds. Each round is a composition of various logical functions operating on 32-bit input values. Six essential logical functions are described in Equations 1-6 as follows.

$$\Sigma_0(x) = \text{rotr}^2(x) \oplus \text{rotr}^{13}(x) \oplus \text{rotr}^{22}(x) \quad (1)$$

$$\Sigma_1(x) = \text{rotr}^6(x) \oplus \text{rotr}^{11}(x) \oplus \text{rotr}^{25}(x) \quad (2)$$

$$\sigma_0(x) = \text{rotr}^7(x) \oplus \text{rotr}^{18}(x) \oplus \text{shr}^3(x) \quad (3)$$

$$\sigma_1(x) = \text{rotr}^{17}(x) \oplus \text{rotr}^{19}(x) \oplus \text{shr}^{10}(x) \quad (4)$$

$$Ch(x,y,z) = (x\&y) \oplus (\bar{x}\&z) \quad (5)$$

$$Maj(x,y,z) = (x\&y) \oplus (x\&z) \oplus (y\&z) \quad (6)$$

where rotr^n is the n-bit rotation right operator, shr^n is the n-bit shift right, and $\oplus$ is the bit-wise XOR operator.

Each 512-bits chunk is firstly extended to 64×32 bits values in a so-called extension stage as depicted in Algorithm 1. Consequently, the compression stage processes the extended values. Algorithm 2 reveals the compression algorithm for the second stage. In this algorithm, the initial step only computes the first chunk of the input data set (line 3). The temporal variables (a, b, c, d, e, f, g, and h) are initialized by the initial digest value. Each of the following chunks is processed based on the digest value of the previous chunk. The compression step executes in 64 iterations (lines 8 - 18) to calculate the current hash values (variables a to h). After every 64 rounds, the 256-bit digest value is accumulated from the current hash values. When the final chunk is processed, the algorithm produces the final 256-bit digest value.

The SHA-256 hash function offers a great opportunity to accelerate. To calculate each $W[i]$ value, the extension stage performs three additions, four rotations, two-shift operations, and four XOR operations. Meanwhile, the compression stage executes seven additions, six rotations, six shift operations, six XOR operations, and four AND operations before producing the final digest. While a general-purpose processor (GPP) performs these operations sequentially, an FPGA-based accelerator can process them in parallel.

Algorithm 1: The extension algorithm

```
   Input: 32-bit messages M[0..15]
   Result: W[0..63]                    // Extended array
1  Var: I
2  Initialize I ← 0
3  for I ← 0 to 15 by 1 do
4      W[I] ← M[I];
5      I ← I + 1
6  for I ← 16 to 63 by 1 do
7      W[I] ← W[I − 16] + σ0(W[I − 15])
8            +W[I − 7] + σ0(W[I − 2]);
9      I ← I + 1
```

Algorithm 2: The compression algorithm

```
   Input: W[0..63] K[0..63]
   Result: digest;                     // 256-bit hash code
1  Var: I, T1, T2, a, b, c, d, e, f, g, h
2  if It is the first chunk then
3      Initialize digest;
4  Initialize hash values a, b, c, d, e, f, g, h;
5  for I ← 0 to 63 by 1 do
6      T1 ← h + Σ1(e) + Ch(e, f) + K[I] + W[I];
7      T2 ← Σ0(a) + Maj(a, b, c);
8      h ← g;
9      g ← f;
10     f ← e;
11     e ← d + T1;
12     d ← c;
13     c ← b;
14     b ← a;
15     a ← T1 + T2;
16 digest[31..0] ← digest[31..0] + a;
17 digest[63..32] ← digest[63..32] + b;
18 digest[95..64] ← digest[95..64] + c;
19 digest[127..96] ← digest[127..96] + d;
20 digest[159..128] ← digest[159..128] + e;
21 digest[191..160] ← digest[191..160] + f;
22 digest[223..192] ← digest[223..192] + g;
23 digest[255..224] ← digest[255..224] + h;
```

B. Related work

After published in 2002, many FPGA-based implementations for SHA have been proposed in the literature. The authors of papers in [8], [9], [10], [11] proposed a solution of using Carry-Save Adder to minimize the delay caused by the additions chain of the SHA-256 algorithm. The designs in [11] and [12] used an unrolled architecture to reduce the number of cycles needed to calculate all rounds of the algorithms. This unrolled method can reduce up to half of the total cycles. However, it required a complex controller and dramatically increased the amount of hardware resource usage and the computing core power consumption. Research presented in [13] used Block RAM (BRAM) to store constant values needed for the compression stage. The authors in [1] combined the advantage of the above papers and applied the pipeline architecture to achieve higher performance. This paper also suggested the use of a counter to implement the controller for an SHA-256 system. The design in [14] and its improvement in [15] offered a very high throughput. However, they required so many hardware resources that they could not deploy FPGA families for embedded applications.

The most relevant result to our work was proposed in [16]. In this paper, the authors provided a shallow power consumption design to deploy on Mobile Platforms. They applied most techniques of the previous work and tried to reduce the number of computational logic elements by using multiple data selectors. This design resues some modules several times. Although the proposal reduced the design's power consumption, data selectors and their control signals used more hardware resources. Finally, the system consumed more power. Besides, it also increased the combinational logic delay of each pipeline stage and reduced parallelism. The design offered a low bandwidth. Thus, it was not suitable for all embedded applications.

III. Optimized Architecture

In this section, we propose the optimized architecture for the FPGA-based SHA-256 computing core. At first, we present the generic architecture of the computing core. We then explain in detail the functionalities and structures of each module inside.

A. SHA-256 Core

The SHA-256 computing core is designed according to the Datapath-Controller model. The proposed FPGA-based computing core consists of four main modules as follows.

- The **Controller** module receives control signals from outside through the Communication module and statuses of both the Extension and Compression modules to generate the appropriate control signals. The Controller module maintains the soundness and correctness of the other modules.
- The **Communication** module receives original messages and control signals for processing of the computing core. The module also sends the 256-bit digest results out when the hash process completes. The Communication module is designed to target the streaming data communication form.
- The **Extension** module performs the expansion stage. The module then forwards W data generated after the extension stage to the Compression module for processing in the compression stage.
- The **Compression** module compresses the W data generated by the Extension to produce the digest. After 64 iterations of the compression process completed, the Communication module sends out the final digest. Other units of the system further process the final result of the SHA-256 module.

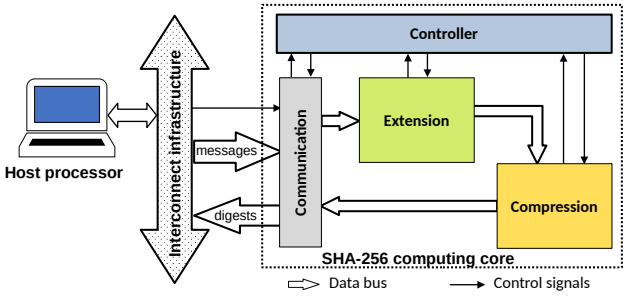

Fig. 1. The overview architecture of the proposed FPGA-based SHA-256 computing core

Figure 1 illustrates the architecture of the FPGA-based SHA-256 computing core. The pipeline model is also preferred in the design. Two levels of the pipeline are applied as follows.

- **The pipeline of computation stages**: the hashing process consists of four steps. The SHA-256 computing core needs 16 cycles during the first stage to fetch a 512-bit chunk from an interconnect infrastructure (e.g., a bus, shared memory, etc.) via the communication module. The Extension module takes one cycle to extend data input from 512 bits to 64×32 bits during the second stage. After that, the Compression module compresses the output of the Extension module in one cycle. The Extension and Compression modules have to execute in 64 rounds for one chunk. Therefore, the pipeline model can be applied to the execution of the Extension and Compression modules. In total, the two modules take 64 cycles to complete hashing a chunk. When the final digest is produced, the computing core takes 8 cycles to push the digest out to the interconnect infrastructure. Therefore, the computing core needs 88 cycles to complete the entire hash process of a chunk.
- **The pipeline of chunks**: along with the pipeline of computation stages, our design offers a pipeline of the input chunks. In this model, hashing of the next piece does not need to wait for the finish of the previous one. It can starts when the previous hashing has been processed for 65 cycles.

B. Controller module

The Controller module maintains the soundness and correctness of the SHA-256 computing core. It receives input signals from the backbone bus and feedback from the other modules to determine the computing core's consequent operations. A Finite State Machine (FSM) is used to implement the Controller. There are 80 states in total. In other studies in the literature, authors prefer a counter to implement a similar FSM for the Controller module. Using a counter allows the design to reduce the number of registers, but it requires more combinational logic elements for the adder and decoder, as depicted in Figure 2.

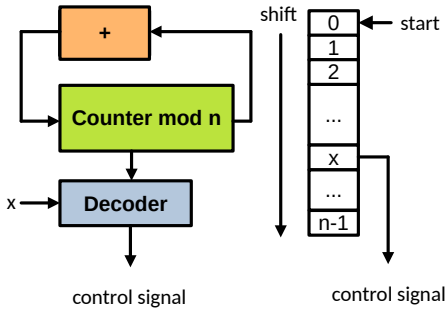

Fig. 2. Comparison of the counter and and our shift register method

Additionally, input data usually change at a high frequency during the counting process. Our design encodes the 80 states of the hash process by the one-hot method and uses shift registers to perform stage transitions for the FSM. Our approach offers two main benefits compared to the counter method: (i) Our approach does not need to implement combinational circuits to calculate the next state for state transition and decode the current state. Therefore, we can reduce hardware resource usage and total power consumption. (ii) One-hot code allows values of each bit in the shift register to change only twice during 80 cycles. Therefore, we can reduce the dynamic power of the design. Figure 2 compares the counter method and our shift register method.

C. Extension module

The Extension module expands 16 $\times$ 32-bit input chunks to 64 $\times$ 32 bits W values. Algorithm 1 shows that the extension stage needs a 64 $\times$ 32-bit buffer to store all values of W. However, only values of $W[I-16]$, $W[I-15]$, $W[I-7]$, and $W[I-2]$ are required to calculate the value of $W[I]$. In other words, the buffer only needs to keep 16 previous values. Therefore, our design uses 16 $\times$ 32-bit shift registers to store required data to calculate values of W in one particular round. When processing new data, it is pushed toward the tail of the shift chain. At the same time, the data element stored at the head of the shift register is released and forwarded to the Compression unit. In our design, Carry Save Adder (CSA) performs three-operand additions. CSA is the most effective adder for the addition of 3 or more operands to implement a fast computation of arithmetic of register-transfer level (RTL) design [17] [18]. Figure 3 illustrates the architecture of the Extension module.

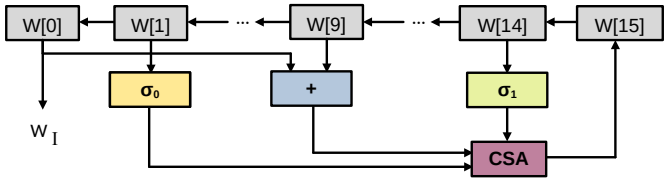

Fig. 3. The micro-architecture of the Extension module

Algorithm 1 also reveals that there are 13 operations in each round of the Extension stage. The Extension module can execute them in parallel due to data independence. Therefore, the FPGA-based SHA-256 computing core can introduce a considerable speed-up when compared to GPPs.

D. Compression unit

The Compression module compresses 64 values of W generated by the Extension module to produce a 256-bit digest. Based on Algorithm 2, eight different 32-bit temporary variables are used for the compression process. We name them as a, b, c, d, e, f, g, and h. When the compression process begins, the temporal variables are initialized by the current value of the digest. After 64 rounds of compression, they are added to the digest to produce the new value of the hash code. Although there are eight variables, only the a and e variables need to be re-calculated in each round. Meanwhile, the variables b, c, d, f, g, and h receives values from their previous neighbors. Therefore, in our design, two shift registers are used to store the variables. The first one stores values of a, b, c, and d while the second one keeps values of e, f, g, and h. Figure 4 reveals the shift direction of the two shift registers and the calculation process of the Compression module. The input of the 256-bit digest register is kept stable during the compression process. It is updated if and only if the `init data` signal is active or the compression process completes. Otherwise, it is disabled to reduce the dynamic power consumption.

According to Algorithm 2, there are 23 operations in each round of the Compression stage. Since the pipeline mechanism is applied, at least 36 operations can be performed in parallel.

E. Communication agent

The Communication module provides an interface so that the SHA-256 computing core receives input data and writes output digest to the interconnect infrastructure. In our design, the Communication module is designed and implemented to support the data stream form. The stream interface and the on-chip memory interface (buffers) are compatible for transferring data. Our designed stream interface has the following features: (i) requiring a minimum of input/output signals; (ii) following the stream protocol to be compatible with both the Intel Avalon bus [19] and Xilinx AXI bus [20]; and (iii) being compatible with the existing IP Core of Intel and Altera. The stream interface includes a sink interface to receive messages from

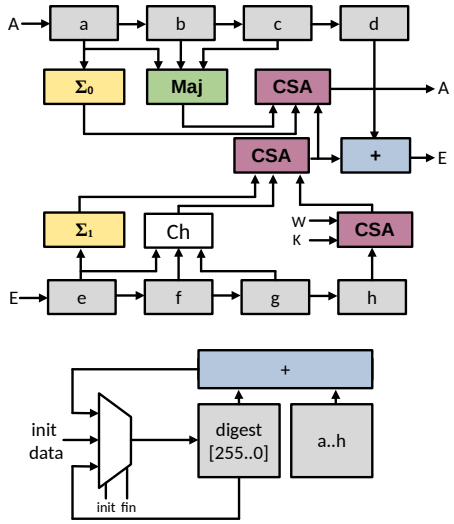

Fig. 4. The micro-architecture of the Compression module

source devices and a source interface to send the final digest to sink devices. Figure 5 illustrates the input/output ports of the FPGA-based SHA-256 computing core following the stream protocol and how they connect with other stream devices.

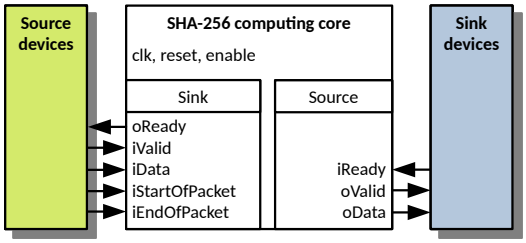

Fig. 5. The Communication interface for the SHA-256 computing core

The stream protocol requires that sink devices issue and keep ready signals active to notice that they can receive data. When source devices receive ready status from sink devices and data are ready to send, the source devices issue and keep valid signals active. Figure 6 reveals how our SHA-256 computing core communicates with source and sink devices. To start communication with a source, the computing core activates the `oReady` signal. The source module then acknowledges by the high level of the `iValid` signal (please note that a source device only activates a valid status when a connected sink device is ready before). After this handshaking, data arrive at the computing core via the `iData` port with the `iValid` signal active. When receiving all 16 valid data segments, the computing core de-activates the `oReady` output port to alert the connected source. The source device takes one cycle to de-activate its valid signal. The `oReady` can be asserted again after 65 clock cycles for computation of the core. The same rules are used for communication with a sink device to deliver a hash code to the sink. When the `iReady` input is active (i.e., the sink is ready to receive data), the computing core asserts the `oValid` signal along with data segments at the `oData` port. When the bit-width of the interconnect infrastructure is 32, it takes eight clock cycles to transfer a 256-bit digest to the sink.

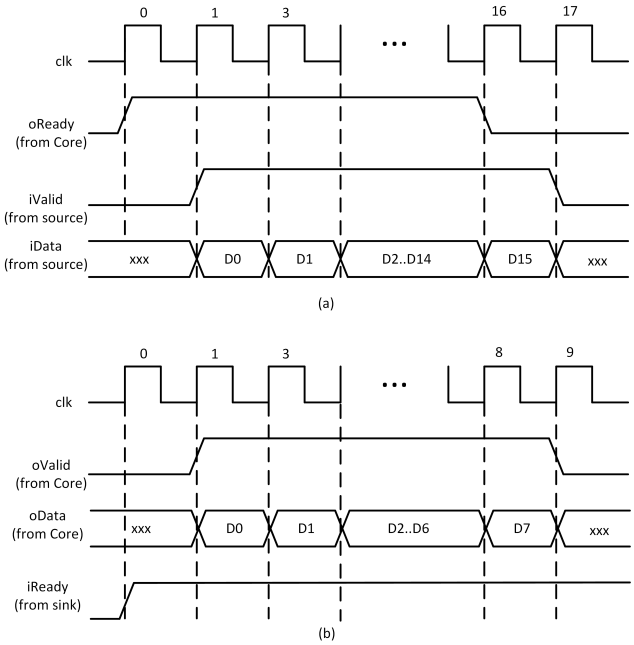

Fig. 6. Waveform of (a) communication with a source; and (b) communication with a sink

IV. Experiments

In this section, we present our experiments for testing and validating the core. At first, we introduce our setup for conducting experiments. We then discuss the experimental results and comparisons with other studies in the literature.

A. Experimental setup

We perform the experiments in the SoCKit platform. SoCKit is an FPGA development kit based on the Intel Altera Cyclone V SoC FPGA. The experimental platform, widely used for embedded applications, includes a Cyclone V SoC 5CSXFC6DF32 FPGA chip with 110K Programmable Logic Elements and Dual-Core ARM Cortex-A9. The Cyclone V family targets performance efficiency and low-cost embedded systems. In our test scenarios, we execute the SHA-256 hash function with a 1000 MB input message. We collect and compare the execution time between FPGA-based and GPP-based implementation. The GPP is the ARM Cortex-A9 processor inside the FPGA SoC functioning at 925 MHz [21].

Meanwhile, the working frequency of our FPGA-based core is 100 Mhz. We also execute the hash function on the AMD Ryzen 7 4800H processor functioning at 4.2 GHz. Nevertheless, we would like to emphasize that our computing core targets embedded applications, which prefer less power consumption and hardware resources usage instead of high

frequency or bandwidth. We also compare hardware resource usage and the maximum frequency for the computing core when synthesized with various FPGA families from Xilinx, including Spartan, Artix, and Zynq. These families also target efficient cost and power consumption.

Figure 7 depicts the experimental system used to measure the performance of our computing core. The **Data generator** module generates a 1000 MB message for the hash process. The **Performance counter** starts when receiving the `Start` signal and completes when receiving the `Finish` signal from the SHA-256 computing core. The host computer reads the performance records through USB Blaster [22] to report measured values.

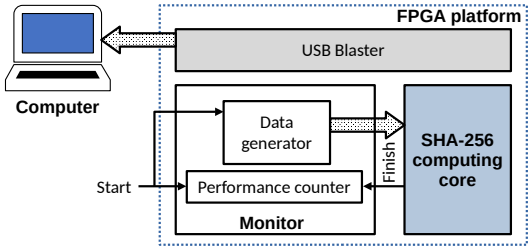

Fig. 7. The experimental system

B. Experimental results

Table I presents the execution time and bandwidth of our FPGA-based SHA256 computing core, the ARM Cortex-A9 processor, and the AMD processor when processing 1000 MB message hashing. The SHA-256 computing core archives speed-ups by up to $14.45\times$ and $1.26\times$ when compared to ARM Cortex-A9 and AMD Ryzen 7 4800H, respectively. Please note that the working frequency of the FPGA-based SHA-256 Core is about 11,11% of the ARM Processor working frequency and 2,38% of the AMD CPU frequency.

TABLE I
EXECUTION TIME, WORKING FREQUENCY, AND BANDWIDTH COMPARISON

Platform	Frequency	Execution Time	Bandwidth
Our FPGA-based core	100 MHz	10.65s	751 Mbps
ARM	925 MHz	153.63s	52 Mbps
AMD CPU	4200 MHz	13.42s	596 Mbps

Table II shows the proposed FPGA-based SHA-256 computing core synthesis results with various FPGA families from Intel Altera and Xilinx vendors. The FPGA chips that we select for our experiments are the most popular when deploying embedded applications. They are Intel Cyclone V (5CSXFC6), Xilinx Artix-7 (xc7a200t), Xilinx SoC Zynq-7000 (xc7z020), Xilinx MPSoC Zynq Ultrascale+, Xilinx Spartan-7 (xc7s50). The resource usage occupies 2% on Cyclone V, 0.97% on Artix-7, 2.46% on Zynq-7000, 1.85% on Zynq Ultrascale+, and 4% on Spartan-7. We manually develop and optimize the core with Verilog-HDL without using any IP Core to obtain the technology-independent computing core. Therefore, our SHA-256 computing core can be easily synthesized and built on FPGA devices from either Intel or Xilinx vendors without any modification. Besides, the working frequency is similar for different FPGA platforms. The maximum working frequency is nearly 140 MHz on most platforms. Furthermore, the maximum bandwidth is 1.04 Gbps, sufficient for all communication protocols used in embedded applications.

TABLE II
HARDWARE RESOURCES USAGE FOR VARIOUS FPGA DEVICES

Device	5CSXFC6	xc7a200t	xc7z020	zu3eg	xc7s50
LUTs	878	1310	1310	1310	1310
Total	41910	134600	53200	70560	32600
Registers	1159	881	881	881	881
Slices	N/A	327	327	327	327
FMax (MHz)	139.04	139.02	139.02	141.84	123.78

We also perform experiments to evaluate the power consumption of our SHA-256 computing core. We use the Spartan-7 FPGA chip from Xilinx for this evaluation, which offers an impressive power optimization. Data are collected when the core functions at 100 MHz, 50 MHz, 25 MHz, and 10 MHz. Figure 8 shows that the Spartan-7 family consumes less power than Artix-7 with the same frequency. With the Spartan-7, the static power consumption is almost stable at $\sim$68 mW with different frequencies. The dynamic power consumption, meanwhile, reduces accordingly to working frequency. It significantly decreases with frequencies lower than 50 MHz when consuming only 4 mW at 10 MHz. In almost all scenarios, the power for clock signal occupies 50% of the dynamic power consumption, while the proportion of signals power and logic power is 25% each.

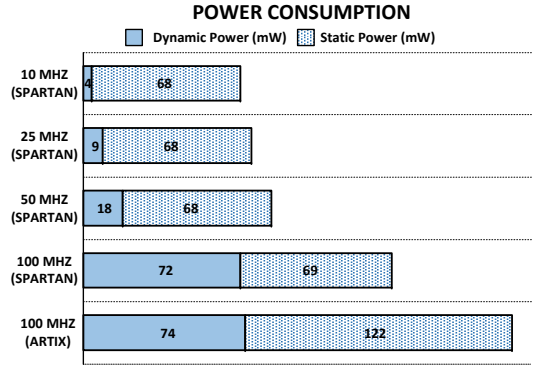

Fig. 8. Power consumption

Figure 9 analyzes the relationship of bandwidth of the computing core, working frequency, and power consumption. In this picture, the performance is measured by Megabits per Watt (Mb/W). The SHA-256 computing core achieves 5202.77 Mb/W at 100 MHz, 4579.65 Mb/W at 50 MHz, 2381.69 Mb/W at 25 MHz, and 1094.03 Mb/W at 10 MHz. The chart shows that the performance significantly decreases when executed under 50 MHz. However, the bandwidth of the core working at 10 MHz (which is 78.77 Mbps) still satisfies the requirements of almost all embedded applications.

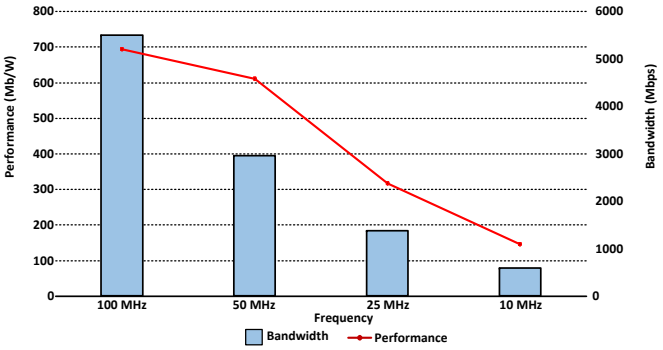

Fig. 9. Bandwidth and Performance

C. Comparison

Table III shows a comparison between our proposed design and the previous work of the SHA-256 computing core design. Our computing core needs a limited amount of hardware resource usage and provides a higher bandwidth when compared with almost all proposals in the literature.

TABLE III
COMPARISON OF OUR SHA-256 COMPUTING CORE AND THE OTHERS PROPOSED IN THE LITERATURE

Ref	Platform	Resources (slices)	FMax (Mhz)	Bandwidth (Mbps)
This work	xc7a200t	327	139.02	1020
	xc7z020	327	139.02	1020
	zu3eg	327	141.84	1040
	xc7s50	327	123.78	910
[16]	xc2v2000	779	71.50	75
[23]	xcv300E	1261	88.00	617
[24]	xcv200	1060	83.00	326
[25]	xcv300e	2008	42.90	56
[26]	VirtexII-6	849	87.00	685
[1]	xc2v200	1373	133.06	1009
[27]	xc2v200	797	150.00	1184
[28]	xcv200	1306	77.00	597
[29]	xc2v2000	1938	81.00	1296
[30]	v200pq240	2120	74.00	582

V. CONCLUSION

This paper presents a compact design of the SHA-256 hash function on an FPGA-based device. The proposed architecture achieves multiple targets: high-throughput, limited resource utilization, optimized and technology-independent, and minimal power consumption. The presented architecture can work at 141.84 Mhz, offers a bandwidth of 1.04 Gbps, and consume only 0.072 W with minimal configuration. The paper also shows a detailed evaluation of the provided SHA-256 computing core on different aspects. It can be helpful to select the appropriate configuration for specific applications. In summary, the SHA-256 implementation in this paper can adapt to a wide range of embedded applications, from low-power to high-performance requirements.

REFERENCES

[1] R. McEvoy, F. Crowe, C. Murphy, and W. Marnane, "Optimisation of the sha-2 family of hash functions on fpgas," in *IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures (ISVLSI'06)*, 2006, pp. 6 pp.–.

[2] "Digital signature standard (dss)," NIST, Tech. Rep. FIPS PUB 186-3, Nov. 2008.

[3] T. Dierks and E. Rescorla, "The transport layer security (dss) protocol," IETF Network Working Group, Tech. Rep. RFC 5246, Aug. 2008.

[4] S. Kent, "Security architecture for the internet protocol," IETF Network Working Group, Tech. Rep. RFC 4301, Dec. 2005.

[5] H. Nunoo-mensah, K. O. Boateng, and J. D. Gadze, "Article: Comparative analysis of energy usage of hash functions in secured wireless sensor networks," *International Journal of Computer Applications*, vol. 109, no. 11, pp. 20–23, January 2015.

[6] M. Kim, D. G. Lee, and J. Ryou, "Compact and unified hardware architecture for sha-1 and sha-256 of trusted mobile computing," *Personal and Ubiquitous Computing*, vol. 17, no. 5, pp. 921–932, Jun 2013. [Online]. Available: https://doi.org/10.1007/s00779-012-0543-0

[7] "Secure hash standard (shs)," NIST, Tech. Rep. FIPS PUB 180-2, 2008.

[8] L. Dadda, M. Macchetti, and J. Owen, "An asic design for a high speed implementation of the hash function sha-256 (384, 512)," in *Proceedings of the 14th ACM Great Lakes Symposium on VLSI*, ser. GLSVLSI '04. New York, NY, USA: Association for Computing Machinery, 2004, p. 421–425. [Online]. Available: https://doi.org/10.1145/988952.989053

[9] ——, "The design of a high speed asic unit for the hash function sha-256 (384, 512)," in *Proceedings Design, Automation and Test in Europe Conference and Exhibition*, vol. 3, 2004, pp. 70–75 Vol.3.

[10] 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 *Proceedings of the 5th International Conference on Information Security*, ser. ISC '02. Berlin, Heidelberg: Springer-Verlag, 2002, p. 75–89.

[11] R. Lien, T. Grembowski, and K. Gaj, "A 1 gbit/s partially unrolled architecture of hash functions sha-1 and sha-512," in *Topics in Cryptology – CT-RSA 2004*, T. Okamoto, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 324–338.

[12] F. Crowe, A. Daly, T. Kerins, and W. Marnane, "Single-chip fpga implementation of a cryptographic co-processor," in *Proceedings. 2004 IEEE International Conference on Field- Programmable Technology (IEEE Cat. No.04EX921)*, 2004, pp. 279–285.

[13] M. McLoone and J. McCanny, "Efficient single-chip implementation of sha-384 and sha-512," in *2002 IEEE International Conference on Field-Programmable Technology, 2002. (FPT). Proceedings.*, 2002, pp. 311–314.

[14] H. E. Michail, G. Athanasiou, V. I. Kelefouras, G. Theodoridis, T. Stouraitis, and C. E. Goutis, "Area-throughput trade-offs for SHA-1 and SHA-256 hash functions' pipelined designs," *J. Circuits Syst. Comput.*, vol. 25, no. 4, pp. 1 650 032:1–1 650 032:26, 2016. [Online]. Available: https://doi.org/10.1142/S0218126616500328

[15] H. E. Michail, G. Athanasiou, V. I. 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. [Online]. Available: https://doi.org/10.1145/2133352.2133354

[16] M. Kim, J. Ryou, and S. Jun, "Efficient hardware architecture of sha-256 algorithm for trusted mobile computing," in *Information Security and Cryptology*, M. Yung, P. Liu, and D. Lin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 240–252.

[17] T. Kim, W. Jao, and S. Tjiang, "Arithmetic optimization using carry-save-adders," in *Proceedings 1998 Design and Automation Conference. 35th DAC. (Cat. No.98CH36175)*, 1998, pp. 433–438.

[18] M. Zeghid, B. Bouallegue, M. Machhout, and R. Tourki, "Architectural design features of a programmable high throughput reconfigurable sha-2 processor," 2008.

[19] "Avalon interface specification," Intel, Tech. Rep. MNL-AVABUSREF, May 2021.

[20] "Axi reference guide," Xilinx, Inc., Tech. Rep. UG-1037, Jul. 2017.

[21] "Cyclone v device datasheet," Intel, Tech. Rep. CV-51002, Nov. 2019.

[22] "Intel quartus prime standard edition user guide: Debug tools," Intel, Tech. Rep. UG-20182, Sep. 2018.

[23] K. K. Ting, S. C. L. Yuen, K. H. Lee, and P. H. W. Leong, "An fpga based sha-256 processor," in *Field-Programmable Logic and Applications: Reconfigurable Computing Is Going Mainstream*, M. Glesner, P. Zipf, and M. Renovell, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 577–585.

[24] N. Sklavos and O. Koufopavlou, "On the hardware implementations of the sha-2 (256, 384, 512) hash functions," in *Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS '03.*, vol. 5, 2003, pp. V–V.

[25] S. Dominikus, "A hardware implementation of md4-family hash algorithms," in *9th International Conference on Electronics, Circuits and Systems*, vol. 3, 2002, pp. 1143–1146 vol.3.

[26] Helion ip core products, helion technology. [Online]. Available: http://www.heliontech.com/core.htm

[27] R. Chaves, G. Kuzmanov, L. Sousa, and S. Vassiliadis, "Improving sha-2 hardware implementations," in *Cryptographic Hardware and Embedded Systems - CHES 2006*, L. Goubin and M. Matsui, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 298–310.

[28] R. Glabb, L. Imbert, G. Jullien, A. Tisserand, and N. Veyrat-Charvillon, "Multi-mode operator for sha-2 hash functions," *Journal of Systems Architecture*, vol. 53, no. 2, pp. 127–138, 2007, embedded Hardware for Cryptosystems. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1383762106001093

[29] M. Zeghida, B. Bouallegue, A. Baganne, M. Machhout, and R. Tourki, "A reconfigurable implementation of the new secure hash algorithm," in *The Second International Conference on Availability, Reliability and Security (ARES'07)*, 2007, pp. 281–285.

[30] N. Sklavos and O. Koufopavlou, "Implementation of the sha-2 hash family standard using fpgas," *The Journal of Supercomputing*, vol. 31, pp. 227–248, 2005.