Design and Implementation of a SHA-256 Co-Processor

Aditi Prakash
Department of Electrical Engineering
Hochschule Darmstadt
Schöfferstr 3, 64295 Darmstadt, Germany
stadprak@stud.h-da.de

Abstract— **Exponential rise in technological developments in the recent era has introduced a dire necessity of security for ensuring the authenticity of the information. Accordingly, the demand for cryptographic hash functions have increased. A hash function is a cryptographic algorithm without a key such as MD5, SHA-1 (Secure Hash Algorithm) and SHA-256. Different types of cryptographic algorithms have been developed in order to improve the performance of these information-protecting procedures. In this paper, software design and hardware design of SHA-256 is implemented and its performance is compared with the co-processor designed using NIOS II CPU. SHA-256 is a one-way hash function characterized by being highly secure and fast while having a high collision resistance. This paper gives an overview about cryptography and discusses in detail the design and implementation of SHA-256 on software and hardware, and as a co-processor. Further, the application of SHA-256 in blockchain and cryptocurrency mining is discussed.**

I. Introduction

Cryptography is the process of converting ordinary plain information into unintelligible text or vice-versa. It is a method of storing and transmitting data in a particular form so that only those for whom it is intended can read and process it. It not only protects data from theft or alteration but can also be used for user authentication. Data can be protected either by encryption or by hashing. While Encryption is a two-way function; Information can be retrieved if the key used to encrypt the data is known, hashing is one-way function; the input cannot be retrieved after hashing is performed.

Hashing is the practice of using an algorithm to map data of any size to a fixed length. This fixed length output is called hash-value. Such one-way functions have typical applications in Pseudo Random Generators, Message Authentication Codes, Public Key Cryptography, IFF (Identification, Friend or Foe Detection) etc...

Hash functions are used to protect data, verify a digital signature, or authenticate a wide range of online processes and transactions, using complex logical operations. The hash functions have unique properties making them highly secure. They are one-way functions, meaning for a given hash output h, it is computationally infeasible to retrieve the input message x, where $H(x) = h$. They also have a very high collision resistance, that is it is computationally infeasible to have two inputs with the same hashed output ($x \neq y$ and $H(x) \neq H(y)$).

Besides, hash functions are highly sensitive. A slight change in the input message will completely change the output hash.

Although the input to a hash function can be of any length, the length of the output is finite. This means that at some point of time when huge amount of data is being hashed, the output combination ought to repeat for a different input data as well. Albeit, collisions exist and cannot be completely avoided.

This paper explores the design and implementation of SHA-256 algorithm. Initially a pure software approach of SHA-256 is explored by designing the algorithm in C language. Later, it is implemented with System Verilog with an appropriate testbench. Simulation results are analyzed with Model Sim Altera tool. Lastly, a wrapper is designed for the SHA-256 kernel to interface it with a NIOS II CPU and is modelled to behave as a co-processor/accelerator when connected with the hardware.

This paper is organized as follows: In section 2, the principle and working of SHA 256 is introduced; design of SHA-256 is discussed in section 3; and finally results of the design and application of SHA256 in currency mining are presented in section 4.

II. PRINCIPLE OF SHA-256

A. SHA-256 Functions and constants

SHA-256 algorithm uses a series of Rotate and Shift operations throughout the computation in order to calculate the hash function. These operations are combined together in different combinations to obtain six functions that are used in the algorithm at different stages.

Below is a short description of the functions:

$$\sigma0(x) = ROTR7\ (x)\ \wedge\ ROTR18\ (x)\ \wedge\ SHR3\ (x) \quad (1)$$
$$\sigma1(x) = ROTR17\ (x)\ \wedge\ ROTR19\ (x)\ \wedge\ SHR10\ (x) \quad (2)$$
$$\Sigma0(x) = ROTR2\ (x)\ \wedge\ ROTR13\ (x)\ \wedge\ ROTR22\ (x) \quad (3)$$
$$\Sigma1(x) = ROTR6\ (x)\ \wedge\ ROTR11\ (x)\ \wedge\ ROTR25\ (x) \quad (4)$$

Choice (x, y, z) → Decides the second input based on the first input (e.g., if x = 1, choose y and if x = 0 then choose z)

Majority (x, y, z) → The result is the majority of the three bits

The constant Kt in SHA-256 is used to mix the message that is put into hash function. They are cube roots of prime numbers. This is done for the first 64 prime numbers, i.e., $0 \leq t \leq 63$ for Kt. The initial hash value for the first round of compression is stored into 8 registers a to h. They are initialized with the square root of first 8 prime numbers.
Two temporary hash values are required which are calculated as below:

$$T_i = \Sigma 1(e) + \text{Choice}(e, f, g) + h + K_i + Wi \quad (5)$$

$$T_i = \Sigma 0(a) + \text{Maj}(a, b, c) \quad (6)$$

B. *Message Padding*

Before starting the actual computation of the hash value, data has to be padded since the hash function hashes the data in chunks of 512 bits at a time. Therefore, input message of length 'L' is padded with a '1' that indicates the termination of message input or acts as a separator, followed by 'k' zero bits where k is the smallest non-negative solution to L+1+k = 448 mod 512, then the length of the message 'L' is expressed in binary in the last 64 bits.

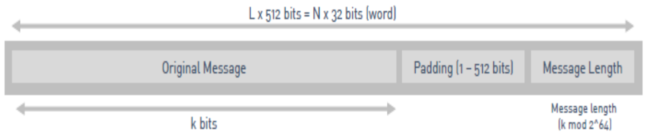

L x 512 bits = N x 32 bits (word)
Original Message
Padding (1 – 512 bits)
Message Length
k bits
Message length (k mod 2^64)

Fig 1: Structure of message after padding

If the message is longer than 512 bits then it will be padded to the next least multiple of 512 i.e., 1024. Later the padded message is divided into multiple blocks of 512 bits.

C. *Creation of Message Schedule*

A message schedule, Wt needs to be created from each message block. It is required that the schedule is 64 words long.
For $0 \leq t \leq 15$, a message schedule is directly created from the input message while for $16 \leq t \leq 63$, a message schedule is calculated by using the following expression:

$$Wt = \sigma 1 [W(t - 2)] + W(t - 7) + \sigma 0 [W(t - 15)] + W(t - 16) \quad (7)$$

D. *Compression*

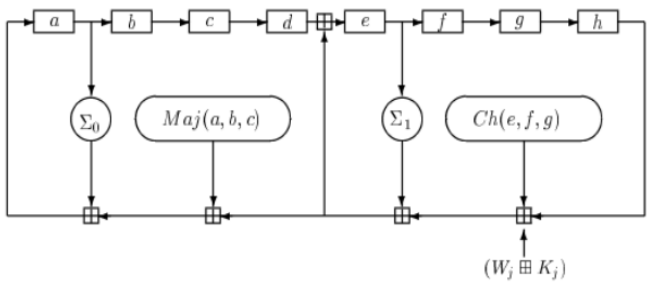


Fig 2: Compression in SHA-256

Compression uses four functions (Equation (3), (4), Choice and Majority) for its computation. The initial hash value is assigned to 8 variables and 64 iterative operations are performed along with constants (Kt) and message schedule (Wt) to obtain the final value of these 8 variables.

E. *Final Hash Creation*

The computed hash values are added with the initial hash value and is fed as an input to the next block. The final block provides the hash value of 256 bits.
The below image provides an overall picture of the SHA-256 computation. C here in the image indicates the compression block

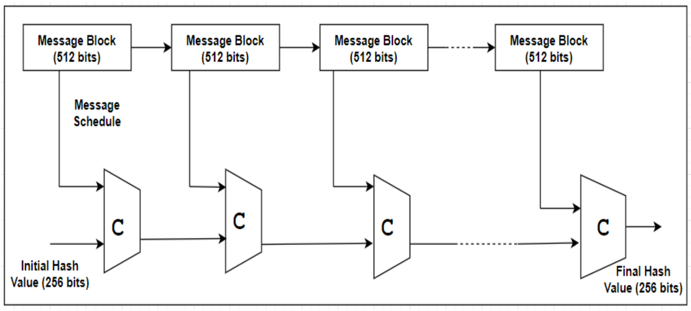


Fig 3: Block Diagram of SHA-256 operation

III. DESIGN AND IMPLEMENTATION

A. *Software Design*

The Software Design of SHA-256 is a simple C program that reads the input string from the user and provides the hash values as output.
The below image shows the flow of the C code.

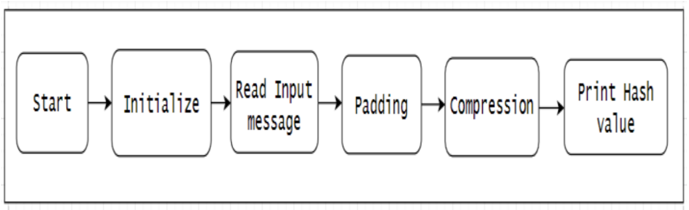


Fig 4: Softwware Block Diagram

B. *Hardware Design*

Hardware Description Language System Verilog is used for the Hardware design. The computation of SHA-256 is designed using a State Machine. The SHA-256 kernel is clock synchronous with a clock of 50Mhz and performs computations on every positive edge. The processing of SHA-256 takes 176 cycles. The behaviour is verified by writing an appropriate test bench for the implementation.

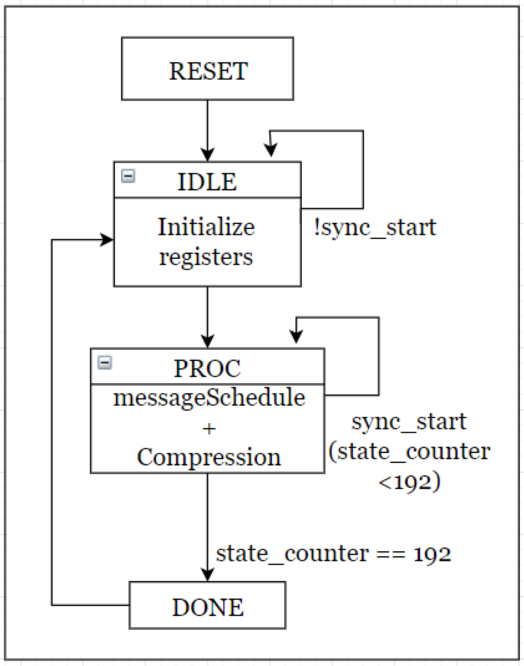


Fig 5: State machine for SHA-256

Below is a table that represents the number of clock cycles consumed by each stage in the processing state.

	No. of Clock cycles	Time (in ns)
Calculation of Message Schedule	48	960ns
Compression	128	2560
Total	**176**	**3520**

Table 1: Table depicting clock cycles

C. *Hardware Software Co-Design*

The goal is to design an accelerator that offloads the CPU and performs the SHA-256 computation independent of the CPU. This reduces CPU load and speeds up the processor. For this, a NIOS II CPU is taken as a reference for the implementation.

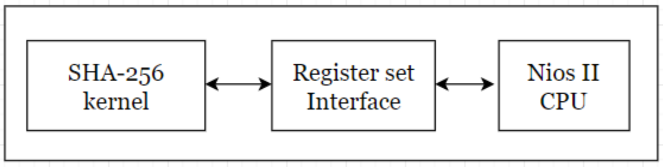
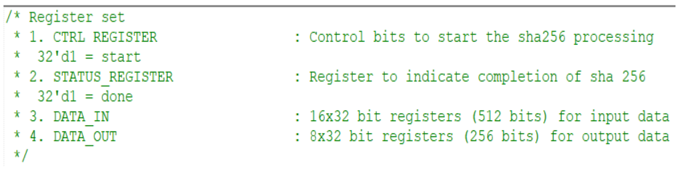


Fig 6: Block diagram of Accelerator interfaced with CPU

A testbench is designed in a such a way that it emulates the CPU behavior. A register module is used as an interface between the SHA-256 kernel and the CPU to provide the necessary interface.

Register set consists of control register, status register, input and output registers as shown below. A sum total of 26 registers, each of 32 bit is required for the design.


```
/* Register set
 * 1. CTRL REGISTER                : Control bits to start the sha256 processing
 *  32'd1 = start
 * 2. STATUS_REGISTER              : Register to indicate completion of sha 256
 *  32'd1 = done
 * 3. DATA_IN                      : 16x32 bit registers (512 bits) for input data
 * 4. DATA_OUT                     : 8x32 bit registers (256 bits) for output data
 */
```
Fig 7: Register set used for the accelerator design

The testbench always selects address of the appropriate register before reading or writing the data.
Initially, the input message is stored into the registers. Upon receiving start signal from the testbench, SHA kernel starts the processing and stores the hash value in output registers after processing.
The testbench then reads the output hash value from the registers after a certain duration.

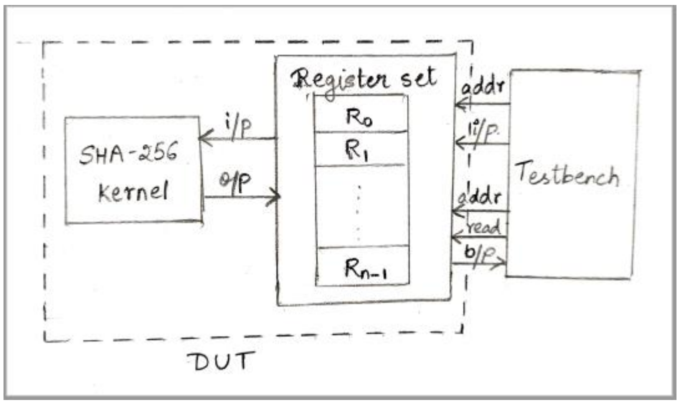
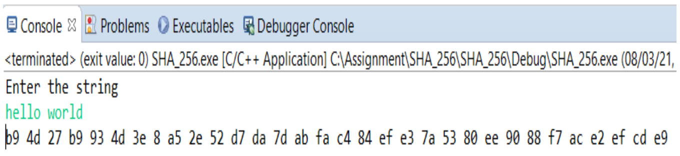


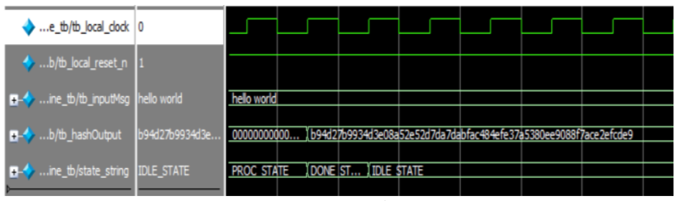
Fig 8: Block diagram of Hardware Software Co-Design

IV. RESULTS AND DISCUSSION

This paper implements SHA-256 in software, hardware and as an accelerator to the NIOS II CPU. All the designs are tested with different inputs and each of them verified for their authenticity.
Below are the results of the tests performed for one of the input messages.



```
<terminated> (exit value: 0) SHA_256.exe [C/C++ Application] C:\Assignment\SHA_256\SHA_256\Debug\SHA_256.exe (08/03/21,
Enter the string
hello world
b9 4d 27 b9 93 4d 3e 8 a5 2e 52 d7 da 7d ab fa c4 84 ef e3 7a 53 80 ee 90 88 f7 ac e2 ef cd e9
```
Fig 9: Hash value output of C program



```
...e_tb/tb_local_clock    0
...b/tb_local_reset_n     1
...ine_tb/tb_inputMsg     hello world          hello world
...b/tb_hashOutput        b94d27b9934d3e...    00000000000... | b94d27b9934d3e08a52e52d7da7dabfac484efe37a5380ee9088f7ace2efcde9
...ine_tb/state_string    IDLE_STATE           PROC STATE | DONE ST... | IDLE STATE
```

Fig 10: Simulation output of hardware design

The Bitcoin network usees SHA-256 as a core component to its design. Bitcoin consists of a network of computers connected through the internet called nodes. Anyone with a computer and internet connection can join the network by running a Bitcoin software. Every node on the network contains an identical copy of a distributed ledger; a database containing all previous transaction history on the network. This is also known as the blockchain.

A transaction consists of the transfer of Bitcoin from one or more inputs to one or more outputs. These transactions are given a unique transaction value as a 256-bit hash which is used to uniquely identify and validate the transaction. This hash is calculated by running the transaction code through the SHA-256 algorithm.

A block contains a group of transactions. Similar to the transaction data, the block has a unique 256-bit hash value which is used to identify and validate the block. This value is computed by applying the SHA-256 algorithm to the raw code of the transaction.

Before the block can be added to the blockchain, the mining node responsible for creating the block must prove that it has solved a computationally difficult problem. This is also achieved using SHA-256 hash algorithm.

For each block, miners do not modify the transaction data, but use another random piece of data called a 'nonce' with transaction data to generate final hash. This process is repeated with different nonce until the final output hash confirms with the bitcoin protocol.

Hardware implementations for bitcoin mining varying from using a generic CPU to dedicated ASICs are commercially available, with different optimizations like pipelining, delay balancing, loop unrolling, different type of adders and more.

V. CONCLUSION

In conclusion, SHA-256 designs are implemented and tested successfully. Software model is implemented with Eclipse IDE, hardware model is implemented and tested in ModelSim tool and software for accelerator is designed with Quartus Prime tool and simulation is performed in ModelSim Altera tool.

Lastly the extensive use of SHA-256 in bitcoin mining is also discussed.

REFERENCES

[1] J. Li, Z. He and Y. Qin, "Design of Asynchronous High Throughput SHA-256 Hardware Accelerator in 40nm CMOS," 2019 IEEE 13th International Conference on ASIC (ASICON), Chongqing, China, 2019, pp. 1-4.

[2] M. Sivanesan, A. Chattopadhyay and R. Bajaj, "Accelerating Hash Computations Through Efficient Instruction-Set Customisation," 2018 31st International Conference on VLSI Design and 2018 17th International Conference on Embedded Systems (VLSID), Pune, India, 2018, pp. 362-367.

[3] S. binti Suhaili and T. Watanabe, "Design of high-throughput SHA-256 hash function based on FPGA," 2017 6th International Conference on Electrical Engineering and Informatics (ICEEI), Langkawi, Malaysia, 2017, pp. 1-6.

[4] A. H. Gad, S. E. E. Abdalazeem, O. A. Abdelmegid and H. Mostafa, "Low power and area SHA-256 hardware accelerator on Virtex-7 FPGA," 2020 2nd Novel Intelligent and Leading Emerging Sciences Conference (NILES), Giza, Egypt, 2020, pp. 181-185.

[5] M. Padhi and R. Chaudhari, "An optimized pipelined architecture of SHA-256 hash function," 2017 7th International Symposium on Embedded Computing and System Design (ISED), Durgapur, India, 2017.

[6] What is Cryptography? Definition of Cryptography, Cryptography Meaning - The Economic Times (indiatimes.com)

[7] The difference between Encryption, Hashing and Salting (thesslstore.com)

[8] The Mathematics of Bitcoin — The Blockchain | by Toby Chitty | The Startup | Medium

[9] How Bitcoin mining really works (freecodecamp.org)

[10] Lecture Slides