© 2019 IJSRSET | Volume 6 | Issue 3 | Print ISSN: 2395-1990 | Online ISSN : 2394-4099
Themed Section : Engineering and Technology
DOI : https://doi.org/10.32628/IJSRSET196318

RTL Design, Verification and Synthesis of Secure Hash Algorithm to implement on an ASIC Processor

Akhilesh S Narayan[1], Ashish J[2], Noor Afreen[3], Lithesh V S[4], Sandeep R[5]
[1-4]Department of ECE, VVCE, Mysore, Karnataka, India
[5]Associate Professor, Department of ECE, VVCE, Mysore, Karnataka, India

ABSTRACT

In this project we are comparing different architectures and adding the features that increases the efficiency of our architecture. Few of them are including multiplexers in the message digester, using different adder architectures in the required places, reducing the critical path by breaking the longest path and making them to operate parallelly. Use of multiplexers reduces the number of registers required in the message expander. It simply transfers the output of expander to compressor block in every clock cycle. Whenever the number of cycle is greater than 16, the multiplexer switches the select line so that the computed message digest to send as output to the compressor. Using of a carry save adder and adder array takes lesser time to perform addition than a pair of adders array. Finally we all know that reducing the critical path reduces the overall operation time and hence increases the efficiency. Considering all these factors in the design we are designing the microarchitecture for SHA-256 algorithm and obtain the RTL code for that architecture. We have also verified the design by designing a test-bench, and finally synthesized the design.

Keywords: SHA 256, Bit Coin, Block chain, Microarchitecture, RTL

I. INTRODUCTION

SHA-256 is a member of the SHA-2 cryptographical hash functions designed by the United States intelligence agency. SHA stands for Secure Hash Algorithm. Cryptographic hash functions are mathematical operations that run on digital information; by examination of the computed hash (the output from execution of the algorithm) to a better-known and expected hash value, a person can determine the data's integrity. A unidirectional hash will be generated from any piece of information, but the data cannot be generated from the hash.

SHA-256 is employed in many completely different components of the Bit coin network:

a) Mining uses SHA-256 as the proof of work (A proof of work is a piece of data which is difficult, costly and time-consuming to produce but easy for others to verify and which satisfies certain requirements) algorithm.
b) SHA-256 is used in the creation of bit coin address to improve security and privacy.

BITCOIN AND BITCOIN MINING

Consumers tend to trust printed currencies, at least in the United States. That's because the U.S. dollar is backed by a central bank called the Federal Reserve. In addition to a host of other responsibilities, the Federal Reserve regulates the production of new money and prosecutes the use of counterfeit currency even digital payments using the U.S. dollar are backed by a central

authority. When you make an online purchase using your debit or credit card, for example, that transaction is processed by a payment processing company such as MasterCard or Visa. In addition to recording your transaction history, those companies verify that transactions are not fraudulent, which is one reason your debit or credit card may be suspended while traveling.

Bitcoin, on the other hand, is not regulated by a central authority. Instead, Bitcoin is backed by millions of computers across the world called miners. This network of computers performs the same function as the Federal Reserve, Visa, and MasterCard, but with a few key differences. Like the Federal Reserve, Visa, and MasterCard, Bitcoin miners record transactions and check their accuracy. Unlike those central authorities, however, Bitcoin miners are spread out across the world and record transaction data in a public list that can be accessed by anyone, even you. When someone makes a purchase or sale using Bitcoin, we call that a transaction. Transactions made in-store and online are documented by banks, point-of-sale systems, and physical receipts. Bitcoin miners achieve the same effect without these institutions by their back.

II. LITERATURE SURVEY

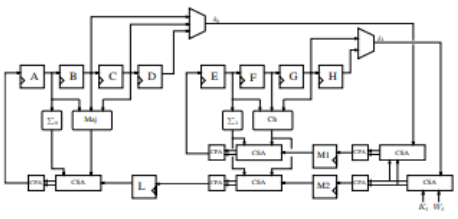

Figure 2: Dadda et al.'s SHA 2 Core [1]

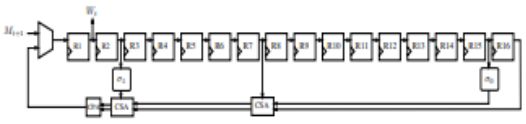

Figure 1: Basic SHA - 2 Message Expander [1]

Of these designs, it was found that the scheme illustrated in Figure 1 gave the shortest critical path and, therefore, could be operated at the highest frequency. In Figure 2, architecture for the message expander was presented which uses CSAs to reduce the number of required adders. Since the critical path in this design (see Figure 2) is shorter than that of the core, it is the core that determines how fast the overall hash algorithm can be executed. The authors also present a design employing delay balancing to further shorten the critical path in the message expander. However, this scheme is unsuitable for unrolling.

III. IMPLEMENTED METHODOLOGY

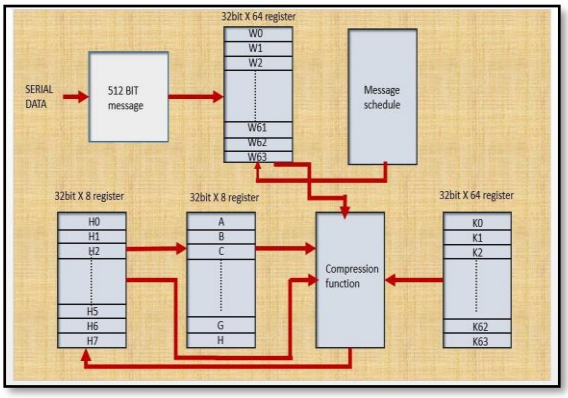

Figure 3 : Top level architecture

The top-level design of SHA-256 is as shown within the figure. It consists of various varieties of memory banks, adders, barrel shifters and completely different bit-wise logical operators. All the operations that's performed here are 32-bit operations and thus 32-bit operators area unit used, to start with, the information is split into chunks of 448 bits for purpose of processing. The message bit is padded with binary '1' and the end of message. The length of the message is padded at the end of 512 bits. i.e. the last 3 bytes of padded message should represent the length of the message. Padding is completed as given below: Let 'M' be the message to be hashed. The message 'M' is cushioned in order that its length (in bits) is adequate 448 modulo 512, i.e. the cushioned message is 64 bits but a multiple of 512. The cushioning consists of information followed by a binary '1' followed by enough zeros to pad the message to the specified length. All the padding operations are done in the I/O block of the ASIC. Once the data is ready for the operation they are loaded on the shift

registers as 32 bit blocks. Using this Message data, extended message block (W) is calculated in the message schedule.

Each extended message block is 32bits. Initially, W_0 to W_{15} is loaded with the message blocks i.e. for i=0 to 15: $W_i = M_i$.

The remaining W_{16} to W_{63} is calculated using previous W values as shown below.

For i = 16 to 63:

$$W_i = \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15}) + W_{i-16}$$

Once all the values of W_i is calculated in message schedule is calculated it is stored in the register bank as shown in the block diagram.

To calculate the hash for given data, some initial hash value is computed by taking the initial hash value (H_0 to H_7) and modifying it using extended message blocks.

i) The initial hash value is obtained by taking the square root of first 8 prime numbers.

These values are stored permanently in a register. A copy of these initial values are copied to a temporary variable a, b, c…h and a lot of mixing operations takes place iteratively in the compression function. The operation that takes place in compression function is as shown in Fig 3.

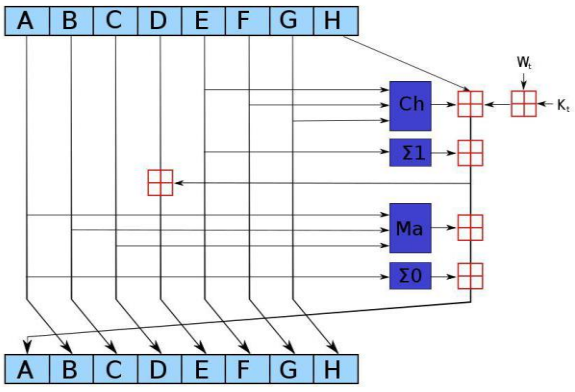

Figure 4 : Compression Function [8]

a) Ma (x, y, z) = [(x & y) ^ (x & z) ^ (y & z)]
b) $\sum_0(x)$ = [(x >> 2) ^ (x >>13) ^ (x >> 22)]
c) $\sum_1(x)$ = [(x >> 6) ^ (x >> 11) ^ (x >> 25)]
d) Ch(x, y, z) = [(x & y) ^ ((~x) & z)]
e) $\sigma_0(x) = [S^7 \oplus S^{18} \oplus R^7]$
f) $\sigma_1(x) = [S^{17} \oplus S^{19} \oplus R^{10}]$

This process iterates for 64 times, for every iteration the A, B, C….D value gets updated. After 64 iterations, these values are added with the initial hash values to obtain a new hash value. The process of updating the hash value is as shown in Figure 4. The new hash value is taken as initial hash value and the process is repeated for next chunk of 512 bit message data. This process repeats until every 512 bit message blocks are used, and the final hash is the hash value which is updated in the register.

IV. COMPUTATIONAL ANALYSIS AND RESULTS

Within one iteration the order of additives in SHA2 doesn't have an effect on the results, there square measure many doable DFGs. as an example, ((a + b) + c) and ((b + c) + a) square measure equivalent in arithmetic however can have completely different DFGs. As a place to begin, the DFG having the minimum iteration sure should be chosen, transformations square measure then performed to seek out the design that achieves this sure. In SHA2 mechanical device, there square measure solely seven adders, finding a DFG having the minimum iteration sure isn't troublesome as long as we tend to perceive a way to calculate the iteration sure.

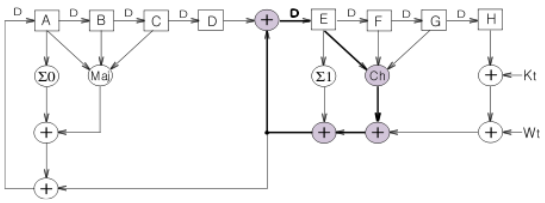

Figure 5 : Compressor Architecture [4]

The Verilog implementation was done for the architecture is shown in the figure 5. Different blocks of the compressor the compressors are created as

different modules and are instantiated to form a compressor module.

Few of the modules used in the compressor block are shown below.

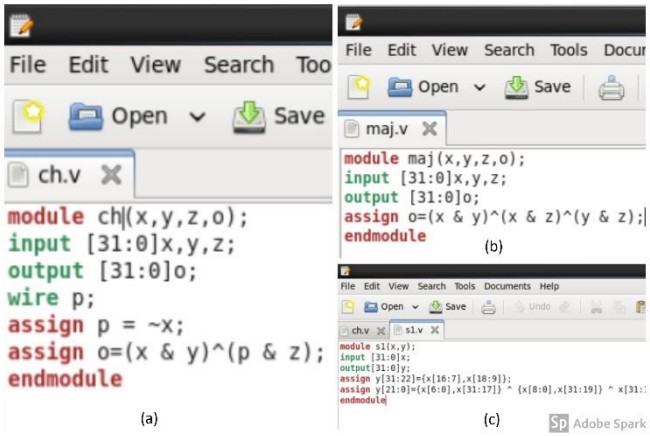

Figure 6 : Different modules used in compressor

Figure 6 (b) represents MAJ module, it performs and - xor operations with 3 operands. One thing we need to observe is that the operands are 32 bits. Similarly Figure 6 (a) represents the 3 operand operation which is described as shown in it. Figure 6 (c) contains an RTL code of 2 operand operation used in the compressor.

Similarly all the other blocks shown in the figure 5 is coded as a different module and later instantiated to form a single module. The part of the main compressor module is as shown in figure 7.

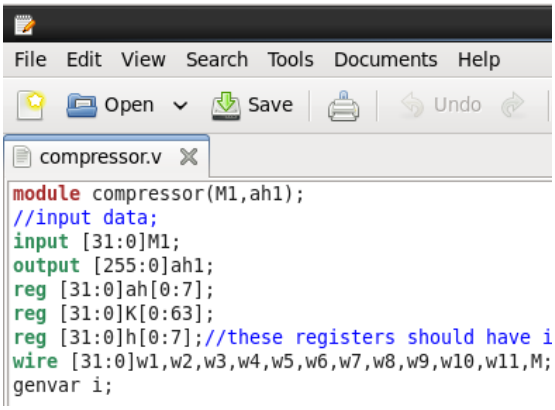

Figure 7: Compressor Module

Coming to the expander part, we have used a multiplexer in order to send the message digest into the compressor block. The 512 bit message is broken into 16 32bit blocks. The remaining message blocks are computed using the previous message blocks. Here the role of the multiplexer is to send the message and the computed message to compressor at the proper time. Figure 8 is the main expander block where we have used multiplexer for routing the message.

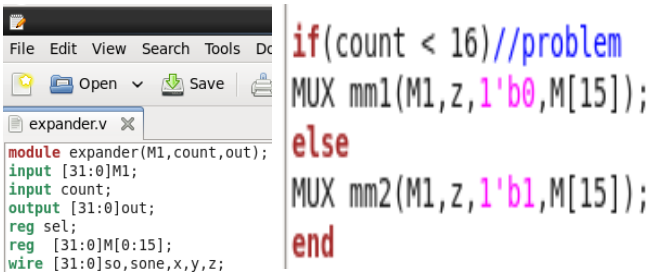

Figure 8 : Expander module and use of multiplexer

Finally the values that has to be stored permanently are stored using case statements. Where each value is associated with a memory. So the values can be accessed by accessing the memory. Figure 9 represents the RTL for storing the constant values.

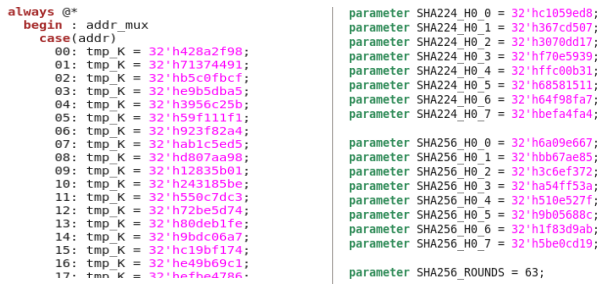

Figure 9 : Initialization of Constant Values

Once the code is being written we checked for errors and simulated the waveforms. Once the code is logically verified we check for synthesis. Basically our target was ASIC. We used RC compiler to compile and synthesis the code. One of the results of the synthesis is as show below in figure 10.

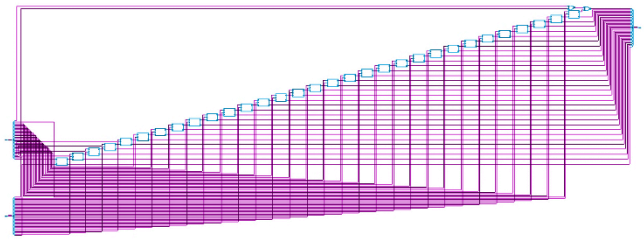

Figure 10 : Truncated synthesis result

V. APPLICATIONS, ADVANTAGES & DISADVANTAGES

It is profusely used in Bit coin IC mining - transactions. Extended applications include implementation in cryptographic protocols such as TLS, SSL, and PGP and so on.

Several govt applications such as protection of classified documents, pprivacy of data transfers, secure electronic transactions (Fund transfers) and digital signature algorithms (Biometric).

SHA-2 may be a cryptologic hash operate, and is often a building block for alternative cryptologic constructs. In satisfying the wants of cryptologic hash, it is a unidirectional operate that's settled, quick to figure, proof against pre-image and second-pre image attacks, and is collision resistant. It's a proof-of-work operate for block chain-based currencies. It seems to be very little quite inelegantly mashing of a bunch of unrelated hash functions along within the naïve hope that which will somehow build it ASIC-resistant and safer. Faster to figure when put next to MD5, SHA-1 algorithms (Lesser machine cycles – sixty four cycles) and is additionally proof against pre-image and second pre-image attacks.

SHA-256 can want considerably further processor time there is an additional 514.5 seconds for SHA-256 to end hashing compared to MD5, or 4.18 further seconds per computer hardware unit. The general measure differed for each environment; the common 30 minutes further processor time required per computer hardware unit remained consistent. there is an incredible discrepancy between amount checksums and conjointly the number of it slow spent in total execution of the method, roughly on the scale of 9 "idle" seconds to at least one processor second. The extended distinction in total runtimes for SHA-256 and MD5 (1,674 seconds on average) is because of factors external to the algorithms. There's to boot a considerable time – memory trade-off. Finally, it's overpriced to implement on ASIC (approx. four integer INR).

VI. CONCLUSION

To conclude, the SHA-256 hashing algorithmic rule is an integral a part of the Bit coin protocol. Its seen implementation in varied aspects of the technology such as: bit coin mining, merkle trees and therefore the creation of Bit coin addresses. Throughput comparisons amongst FPGA-based SHA-2 styles have recently been drawn reportable knowledge turnout for SHA-256 was 693 Mbps and 1034 Mbps for SHA-512, each targeting Virtex-E FPGAs while implementing the algorithm on an ASIC using our architecture and libraries, we obtained a throughput of approximately 1.2 Gbps when compiled at a frequency of 150 MHz and 0.8V supply. Also, there was a significant decrease in power consumption from 9 mW to 3 mW along with the reduction in the number of cells and cell area on the whole. Considering all the evidences declared, we will note that there's a substantial advantage of victimization VLSI hardware implementations to accelerate cryptographical algorithms and protocols.

VII. REFERENCES

[1]. Luigi Dadda, Marco Macchetti, Jeff Owen "An ASIC Design for high speed implementation of Hash function SHA-256", SIGDA – Compendium, March 2004

[2]. Rajeev Sobti, G.Geetha "Cryptographic Hash functions: A review", IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 2, No. 2, March 2012

[3]. Robert P. McEvoy, Francis M. Crowe, Colin C. Murphy and William P. Marnane "Optimization of the SHA-2 Family of Hash Functions on FPGAs", IEEE Computer Society Annual Symposium on VLSI, Germany, March 2006

[4]. James Docherty, Albert Koelmans "Hardware Implementation of SHA-1 and SHA-2 Hash Functions", Newcastle University, 2011

[5]. Dr. Harris E Michael, George Athanasiou, George Theodoridis "Area throughput trade – offs for SHA-1 and SHA-256 hash functions pipelined designs", Journal of Circuits, Systems and Computers, July 2015

[6]. Digital Signature Standard. National Institute of Standards and Technology. Federal Information Processing Standards Publication 186-2. http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf.

[7]. Secure Hash Standard. National Institute of Standards and Technology. Federal Information Processing Standards Publication 180-2, http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf

[8]. https://en.wikipedia.org/wiki/SHA-2

[9]. L. Dadda, M. Macchetti and J. Owen: An ASIC design for a high speed implementation of the hash function SHA-256 (384, 512). ACM Great Lakes Symposium on VLSI. 2004.

[10]. L. Dadda, M. Macchetti and J. Owen: The design of a high speed ASIC unit for the hash func-tion SHA-256 (384, 512). DATE 2004. IEEE Computer Society. 2004.

[11]. M. Macchetti and L. Dadda: Quasi-pipelined hash circuits. Proceedings of the 17th IEEE Symposium on Computer Arithmetic. 2005.

[12]. R. P. McEvoy, F. M. Crowe, C. C. Murphy and W. P. Marnane,: Optimisation of the SHA-2 Family of Hah Functions on FPGAs. Proceedings of the 2006 Emerging VLAI Technologies and Architectures (ISVLSI'06). 2006.

[13]. H. Michail, A.P. Kakarountas, O. Koufopavlou and C.E. Goutis: A Low-Power and High-Throughput Implementation of the SHA-1 Hash Function. IEEE International Symposium on Circuits and Systems, 2005.

[14]. F. Crowe, A. Daly and W. Marnane: Single-chip FPGA implementation of a cryptographic co-processor. In Proceedings of the International Conference on Field Programmable Technology (FPT 2004). 2004

[15]. R. Lien, T. Grembowski and K. Gaj: A 1 Gbit/s partially unrolled architecture of hash functions SHA-1 and SHA-512. CT-RSA 2004. Vol. 2964 of LNCS. Springer. 2004.

[16]. Y. Ming-yan, Z. Tong, W. Jin-xiang and Y. Yi-zheng: An Efficient ASIC Implementation of SHA-1 Engine for TPM. The 2004 IEEE Asia-Pacific Conference on Circuits and Systems. December 6–9. 2004.

[17]. G. T S and T S B Sudarshan: ASIC Implementation of a Unified Hardware Architecture for Non-Key Based Cryptographic Hash Primitives. Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'05). 2005.

[18]. A. Satoh and T. Inoue: ASIC-Hardware-Focused Comparison for Hash Functions MD5, RIPEMD-160, and SHS. Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'05). 2005.

[19]. Helion IP Core Products. Helion Technology. http://heliontech.com/core.htm.

Cite this article as : Akhilesh S Narayan, Ashish J, Noor Afreen, Lithesh V S, Sandeep R, "RTL Design, Verification and Synthesis of Secure Hash Algorithm to implement on an ASIC Processor", International Journal of Scientific Research in Science, Engineering and Technology (IJSRSET), Online ISSN : 2394-4099, Print ISSN : 2395-1990, Volume 6 Issue 3, pp. 70-75, May-June 2019. Available at doi : https://doi.org/10.32628/IJSRSET196318
Journal URL : http://ijsrset.com/IJSRSET196318