ISSN: 2278 – 1323
International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)
Volume 2, Issue 7, July 2013

Low Power And Area Efficiency of SHA-1 and SHA-2 Hash Algorithm

N.Swetha
E.C.E., Ganapathy Engineering College.
Warangal, JNTUH,
Andhra Pradesh, INDIA

K. Raveena, Asst.Prof.
E.C.E., Ganapathy Engineering College.
Warangal, JNTUH,
Andhra Pradesh, INDIA

Abstract—**This paper summarizes about the design of a Low Power and Efficiency of SHA-1 and SHA-2 Hash Functions capable of performing all members of the Secure Hash Algorithm (SHA) group of Hash Functions. The need for high-speed cryptography is introduced, as well as the SHA-1 and SHA-2 Hash Functions and their operation. Work performed at other institutions to improve throughput and power consumption is presented with advantages and disadvantages discussed. The ASIC design is then discussed, with comparisons made to previously publish ASIC and FPGA implementations. The possibility of using this ASIC architecture for the SHA-3 candidates, as well as the Message Digest (MD) families of Hash Functions is suggested as an area of future work as it is shown the ASIC Architecture designed would be capable of this with only program modifications required.**

Keywords- Cryptography, Hash Function, Secure Hash Algorithm.

I. INTRODUCTION

The growth rate for e-commerce has been double-digit over the last decade, with an estimated $301 billion expected online retail sales in 2012. This extreme increase in online trading has lead to a rise in online attacks to obtain money through deception or other illegal means. Due to this, companies and consumers have become more aware of Security risks exchanging information over such an open medium, leading to several third parties setting up secure areas for credit card and bank account details to be shared with Minimal risk of the numbers being obtained and used fraudulently. Major credit companies such as Visa and MasterCard have set up subsidiaries such as Verified by Visa to give consumers confidence that the sites they are buying from are safe. These "security seals" are becoming more common on commercial sites, as customers have been found to avoid purchasing from companies who do not use them. When shopping on The Internet, a connection is set up between the computer being used and the company server using a "Challenge and Response" through the Transport Layer Security (TLS), or its predecessor Secure Sockets Layer (SSL). The typical operation for this can be seen in Fig 1, using hash functions at both the Client and Server end to ensure a connection is secure without the need for cryptographic key exchange. A connection is only accepted if the received and calculated values of cr and sr match.

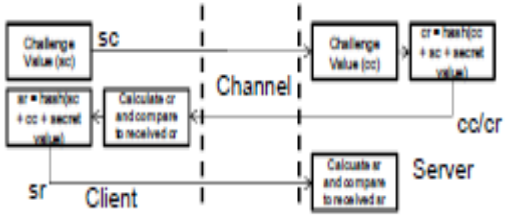

Fig 1. A typical Challenge and Response Scenario.

Secure Hash Algorithm (SHA) is the most widely used Hash Function in the world. It was developed by the National Institute of Standards and Technology (NIST) in the United States and first published in 1993. The original version (SHA- 0) was found to have a serious security flaw and replaced in 1995 with SHA-1.As computing power has increased, this too is found to have weaknesses and the possibility of collisions has been identified. To further enhance security, SHA-2 was released in 2001 containing improvements in the message computation and hash output size. To date, no weaknesses have been identified in SHA-2. Like all Hash Functions, SHA outputs a fixed length digest of a message with arbitrary input length. The original message is converted to blocks of a fixed size, which are sequentially reduced. This is repeated for the entire file, giving a message digest. Due to the Preimage resistance of SHA (the inability to find the message from the hash function), the method of operation can be made public. Would-be attackers gain no benefit from knowledge of the SHA Algorithm, as no key is used to create the message digest. Both published variants of SHA work in a similar way, taking an arbitrary length input, sectioning this to 512-bit blocks (with padding where necessary to ensure the message length is a multiple of 512) and processing these 512- bit blocks through rounds of addition, shifting and logical operations to produce a hash output of either 160 bits (SHA-1) or 224/256/384/512 bits (SHA-2).

Although SHA-2 is increasing in popularity, SHA-1 is still significantly used, not least for its incorporation into the Trusted Platform Module (TPM) ASIC. However, TCG are currently investigating the use of SHA-2 in later module. If

this takes place, the TPM ASIC will also have to contain backwards-compatibility to SHA-1 for communication with older modules. Therefore, it is important that an ASIC can manage creation of SHA-1 and SHA-2 message digests at a speed where delay would not be noticed on a high-speed internet connection.

II. PREVIOUSLY PUBLISHED SHA-1 AND SHA-2 DESIGNS

Most published work regarding FPGA or ASIC implementation of SHA concentrates on either SHA-1 or SHA-2 with no work found on a fully integrated ASIC. Chaves proposes core layouts for both SHA-1 and 2, but does not integrate these into a common core. The implementations found are also only capable of performing one hash function. Due to the high number of different hash functions available, a fully flexible implementation would be of great use, even if this flexibility comes with a throughput penalty.

A. Throughput Improvements

Zeghid and Wanzhong both identify reducing the critical path for calculation through the substitution of Carry Save Adders (CSA) for slower Carry Look-Ahead Adders (CLA). CSAs are capable of performing addition of three numbers, rather than the two that CLA or Full Adders (FA) can perform Xia, in however, uses Full Adders, placing them into a Wallace Tree to reduce the critical path.

Both methods show speed benefits, with giving a maximum throughput of 2073 Mbit/sec for SHA-256 and 950Mbit/sec for SHA-1 in [12] with throughput calculated Using

$$\text{Throughput} = \frac{\text{Block size}}{\text{Clock period} + \text{Latency}}$$

The reduction in Critical Path for the longest calculation allows the remainder of the SHA operation to be unrolled and therefore completed in one, rather than multiple, clock cycles. It is found in that unfolding the design for two operations (so performing two operations in one clock cycle) gives a factor-of-two speed improvement for the same increase in area. Unfolding is also performed in [14], increasing throughput to 76Mbit/sec in conjunction with pipelining. Changing the architecture to a pipeline allows for simultaneous processing of blocks while not affecting the SHA operation. Using the pipeline also gives a level of delay balancing to the circuit, preventing incorrect signal propagation through a circuit and causing an incorrect message digest. Use of these improvement techniques comes at a gate penalty and therefore would not be ideal for applications where power consumption is critical.

B. Power Consumption Improvements

Power Consumption is a key factor in all modern integrated circuit design. Due to the increase in wireless and mobile internet, systems are more commonly running from battery power rather than a mains supply. Therefore, reduction in both dynamic and static power consumption must be considered.

The main consumer of dynamic power is the clock signal, so reducing either its speed or proliferation through the circuit would have a noticeable effect. Reduction of the clock speed is not recommended, as this will reduce the number of computations that can be performed per second and therefore the throughput of the circuit. Due to this, methods of reducing its presence in the circuit take priority. In both Locally Explicit and Bus Specific Clock Enabling are suggested as methods to reduce power. These take the clock signal and only allow it to propagate into areas of the circuit when a signal is applied to the sub-sections input. By doing this, the number of transistors the clock is applied to is reduced and therefore the capacitance it must charge/discharge in a clock cycle, thus reducing the dynamic power consumed on each clock event.

As

$$\text{Dynamicpower} = \text{Frequency} + \text{Capacitency} + \text{Voltage}$$

This technique is found to save up to 65% of the dynamic power, without altering any gate propagation times or overall operational speed.

Static power reduction is more difficult to implement, but equally important as leakage power loss can account for up to 50% of power consumed by an ASIC. Little work can be found on reducing this in cryptographic ASICs, but techniques such as Logic Gating could be used to deactivate areas when not in use. However this would mean any data in these areas has to be stored before deactivation, increasing the register count for the ASIC. Due to the time constraints of the project, reductions in static power were not investigated further. A final method for increasing throughput suggested in and tested in [18] is the use of Very Long Instruction Word (VLIW) to increase parallelism. These papers suggest that having multiple sets of instruction commands to set logic prior to data arriving would improve throughput. Qualitative improvements for this have been noted and this was deemed aviable extension item to the base ASIC design.

III. PROJECT GOALS AND DESIGN

A. Goals

This project lead to the design of ASIC containing functional blocks for performing SHA-1 and SHA-2. This design was intended to be flexible, allowing other Hash Functions to be executed through changing the program within the ROM. Due to time constraints, this chip was not physically

manufactured; instead it was simulated and run using a constructed test bench in VHDL. Since VHDL is standardized by the IEEE, it was felt to be more suitable for use than other languages such as Verilog. For these simulations, timing Considerations were not taken into account.

B. Design

To allow a design capable of performing all variants of the SHA family, a modular approach to the design was taken. This can be seen in Fig 2. The ASIC has one common 32-bit bus, allowing movement of data between sections of the architecture based on the command extracted from ROM and interpreted at the Instruction Register. Differences between SHA-1 and 2 such as the value of initialization vectors are stored in ROM and extracted by requesting their memory address within the program. The ASIC uses tri-state buffers to store data, reducing read/write operations to RAM and eliminating the need for a cache, as well as ensuring the bus can be shared by all components without corruption through the use of their high impedance state.

The ASIC has 36 data pins; with 32 connected to the I/O block, plus a clock pin, a "more message" pin, a selector for SHA-1 or SHA-2 and an asynchronous reset. The reset will place the FSM back into its initial state whenever it is set, as well as clear all register and RAM content. Initialization Values (IV's) and calculation constants (K) will not be affected by the reset, as these are stored in ROM.

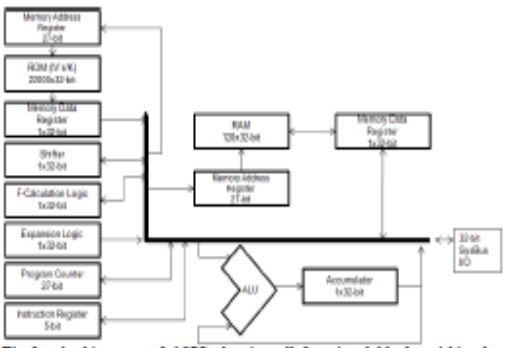

Fig 2. Architecture of ASIC showing all functional blocks within the ASIC connected by a common 32-bit bus

1) Execution of Programs

The program to execute the hash function in stored in ROM and called by incrementing the Program Counter (PC). This is a 27-bit binary counter, incremented by setting an increment flag. The PC may also be reset to allow repeated looping of the program for messages larger than 512-bits, or jumped to a value stored in ROM to allow different programs to be executed. This allows multiple programs to be stored in ROM, such as SHA-1, variants of SHA-2 or Message Digest (MD) hash functions, giving the ASIC a great deal of flexibility. The program stored in ROM consists of a 32-bit number containing an 18-bit operations code (opcode) and 14-bit RAM address for this to be executed upon. To allow execution of SHA-1 and 2, approximately 22000 lines of code were required, with another 81 locations in ROM required from initialization vectors and constants. The opcode is decoded by the Instruction Register and are broadly split into four types; load; store; pass and execute. Load commands place a value from the specified RAM address into a block's tri-state registers. Store commands take the value stored in a specific blocks tri-state registers and place it at the address specified in RAM. Pass allows movement between tri-state registers of different blocks without the use of RAM to increase throughput. The execute command will run a logic block and update its tri-state register with the new result. To execute an opcode, a fetch section increments the program counter and loads the relevant command from ROM into RAM. This is then placed in the instruction register, which splits the 32-bit value received into the Opcode and the RAM address to be extracted (if required). The Opcode is then set within the ASIC, allowing progression through the controlling state machine. Each Opcode takes between 50 and 70us to execute, depending on its location within the state machine. All opcodes can be seen in Table I.

2) Arithmetic Logic Unit

The ALU in this case is a full adder that performs modulo- 32 additions on the value in its accumulator and the value on the bus. The result is stored in the accumulator for transfer to RAM or another functional block. The capability to perform full addition with a carry out is present in case a future hash function requires it, but unused at this time.

3) Shift Register

To perform the logical shifts present in SHA, a shift register was created. This allows parallel loading through the tri-state registers and will then run a set number of times depending on

the opcode loaded. As an example, SHA-1 uses a 30-place left circular shift within its calculation. For this, an opcode was created that would run the shifter this number of times and allow other logical blocks to execute while this was taking place. Once the shift was complete, a completion flag is set, allowing the value in the tri-state register to be moved to another block when required.

4) Logic Calculation

SHA also uses a great deal of logical calculations within the message digest creation. For

example, in SHA-1 on rounds 40-59, the following logical sum is used

F= (b.c) + (b.d) + (c.d)

Where b, c, d and f are all 32-bit blocks of the 512-bit message. Initially, opcodes capable of performing all basic logic functions were created and these equations performed one factor at a time. To increase throughput, further opcodes were added to perform all aspects of SHA-1 and 2 by loading 32-bit blocks to multiple tri-states within the logic block and executing all items in an unrolled function. This gave a great increase in this blocks throughput for a small size penalty.

5) Memory Management

For accessing the ROM and RAM, two sets of registers were added. A Memory Address Register (MAR) and Memory Data Register (MDR). The MAR was used to access the address specified within the opcode and either place the data from this address into the MDR, or load data from the MDR into this address depending which opcode had been loaded into the instruction register. The use of MDR/MAR prevents memory directly writing to the bus and therefore minimizes the possibility of data corruption should the bus exhibit problems.

TABLE I
OPCODES IN ASIC AND USES WITHIN PROGRAM

Opcode	Use	Opcode	Use
Store ACC	Store Accumulator to MDR	Load ACC	Load Accumulator with MDR
Store Expansion	Store Expansion Logic to MDR	Load Expansion	Load Expansion Logic with MDR
Store Logic	Store Logic to MDR	Load Logic	Load Logic with MDR
Store Shifter	Store Shifter to MDR	Load Shifter	Load Shifter with MDR
Store ROM	Store Value In ROM to MDR	Left Shift	Runs Left Shift Operation
Logic Calculation1	Run AND Logic	Logic Calculation2	Run OR Logic
Logic Calculation3	Run NOT Logic	Logic Calculation4	Run XOR Logic
ALU Add	Performs Modular Addition	More Message	Denotes if message on input to Bus is complete
Message Out	Output Message to IO	Expansion Calculation	Runs Block Expansion
Store IO	Store IO to MDR	Load IO	Load IO with MDR
Shift2ALU	Load ALU with Shifter	ACC2Logic	Load Logic with Accumulator

IV. RESULTS

All components of the ASIC in Fig 2 were tested individually by simulation in VHDL using a set of instructions that gave a known output. The correct output from the blocks was placed into the ASIC through the "assert" statement to compare the outputted result and note any errors in the terminal window of the simulation software. All blocks within the ASIC performed as expected and were integrated into a complete system through the use of packages and component instantiation in VHDL. The test benches were designed to test all aspects of operation and feasible conditions of use to identify any issues in the modules [20]. Through this separate testing, some small coding errors were identified and corrected prior to their integration in to the main ASIC. Most errors identified were omissions of commands such as setting the tristate register to high impedance when not requested to output to the bus. An omission such as this could cause the system bus to be used by two components simultaneously and therefore cause communication collisions. As no protocol to identify this (such as CDMA [21]) was implemented along with its associated error checking, any collisions could cause severe issues to the ASIC operation. Therefore, the correction of this code was critical to allowing successful operation of all modules within the system.

Once the SHA-1 ASIC was deemed to be operating successfully, the addition of SHA-2 to the core was made. Due to the modular design used, minimal additions needed to be made to give this chip SHA-2 capabilities: A larger ROM was declared, to contain the extra IV and K values used by SHA-2 for its round calculations, along with the new program, and the addition of right linear and circular shifters to the shifter logic block.

TABLE II
COMPARISON AGAINST PUBLISHED IMPLEMENTATIONS

Implementation	SHA Variant	Logic Blocks Used	Clock Frequency/ MHz	Throughput/ Mbit/sec
Kakarountas	SHA-1	950	98.7	2526.7
Sklavos	SHA-1	1004	42.9	119
Lee	SHA-1	2894	118	5900
Chaves	SHA-2	565	227	1420
Sklavos [22]	SHA-2	2384	74	291
Khalil [23]	SHA-2	4489	50	644
This Work	SHA-1	6836	100	162
This Work	SHA-2	6836	100	98

Note: The throughput values for this work are estimated based on all code-optimization and parallelism improvements identified being implemented.

Though the size of this work seems large compared to other published ASIC/FPGA implementations, this includes the ROM containing the program. If an external RAM/ROM is used, the size of this ASIC implementation decreases to 284, a factor of two lower than Chaves work in [8].

V. CONCLUSION

The ASIC within this paper allows for a reprogrammable design capable of performing the majority of common hash functions. Although only SHA-1 and 2 have been investigated initially, reprogramming the ROM would allow execution of other commonly used functions such as MD4/5. It is also possible for this ASIC to calculate some of the shortlisted SHA-3 algorithms, though the opcodes would not be optimized for these alternate functions. SHA-1 and 2 were chosen for optimization due to their widespread use and integration into the Trusted Platform Module. Therefore, the flexibility of running other hash functions would lead to a speed trade-off for this added ability. Through the use of VHDL, no limitations have been placed on the manufacturing techniques that can be used to create this ASIC.. The architecture already has been successfully

synthesized in Xilinx for the QPRO Vertex 4 FPGA, and could no-doubt be used on others also. Since most other work has been completed at the 0.13-0.18μm level, it is believed that this ASIC could also be built using these dimensions without major issues occurring.

VI. REFERENCES

[1] C. I. Tang pong, Muhammad; Lertpittayapoom, Nongkran "The Emergence of Business-to-Consumer E-Commerce," Journal of Leadership & Organizational Studies, vol. 16, pp. 131-140, 2009.

[2] R. R. Dube, Hardware Based Computer Security Techniques to Defeat Hackers. Hoboken NJ: John Wiley and Sons, 2008.

[3] B. C.-T. Ho, and Kok-Boon Oh, "An empirical study of the use of security Seals in e-commerce," Online Information Review, vol. 33, pp. 655-671, 2009.

[4] J. Seberry and J. Pieprzyk, Cryptography: an introduction to computer Security. New York; London: Prentice Hall, 1989.

[5] R. A. Mollin, An introduction to cryptography. Boca Raton: Chapman & Hall/CRC, 2001.

[6] P. C. v. O. Alfred J. Menezes, Scott A. Vanstone, Handbook of Applied Cryptography: CRC Press, 1996.

[7] (2009). Trusted Computing Group www.trustedcomputinggroup.org. Available: www.trustedcomputinggroup.org

[8] R. Chaves, et al., "Cost-efficient SHA hardware accelerators," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 16, pp. 999-1008, 2008.

[9] M. Zeghid, et al., "A reconfigurable implementation of the new secure Hash algorithm," in Proceedings - Second International Conference on Availability, Reliability and Security, ARES 2007, 2007, pp. 281-285.

[10] S. Wanzhong, et al., "Design and optimized implementation of the SHA-2(256, 384, 512) hash algorithms," in ASICON 2007 - 2007 7th International Conference on ASIC Proceeding, 2007, pp. 858-861.