LPS-SHA1: Low Power and Simple Implementation of Secure Hashing Algorithm (SHA1) using VHDL Implemented on FPGA

Choi Tim Antony Yung, Laurice Sattouf, Dimitri Garcia, **Mohamed El-Hadedy**   
*Department of Electrical and Computer Engineering*  
*College of Engineering, California Polytechnic State University, Pomona*  
Pomona, California  
**Email:** {choiyung, lsattouf, dimitrig, **mealy**}@cpp.edu

*Abstract*—A SHA1 core was designed and implemented to reduce FPGA power consumption and resource usage by integrating byte to word conversion circuitry and eliminating usage of division circuit by separating message schedule counter into a 2-bit and 5-bit counters. A companion system was designed and implemented to handle serial byte stream message input via UART and output the resulting SHA1 digest to a VGA display.

Keywords—VHDL, cryptography, SHA1, FPGA, VGA, UART, message digest

# Introduction

The Internet becomes more and more a major economic parameter of the world's financial and thus whole new applications are being created that presuppose authentication services. One recent example is the Public Key Infrastructure (PKI) that increases citizen’s trust to public networks and thus empowers applications such as on-line banking, B2B applications, electronic payments, stock trading etc. The PKI that is considered as a must-have mechanism for the burst of e-commerce worldwide involves the use of the SHA-1 hash function. However, the implementations that will be used in the PKI should have a much higher throughput compared to the present implementations to be able to correspond to all requests for digital certificates. it is important to mention that SHA1 is already broken, but it is still used as part of other hash-based cryptography schemes, so it is important to find the simplest and more convenient implementation in terms of performance and power consumption. This paper is organized as follows: In section II previous implementations of the SHA-1 are presented. In section III the math behind SHA1 algorithm explained in detail. In section IV the proposed design approach is detailed. In section V power issues concerning the SHA-1 are presented. Throughput and area results of the proposed SHA-1 are offered in section VI and it is compared to the other implementations. Finally, conclusions are offered in section VI.

# Related work

Various techniques have been proposed to minimize the SHA-1 implementation size. The most common techniques are operation rolling loop and/or re-configuration. On the other hand, alternative design approaches have been proposed to increase throughput for a variety of hash function families. The most common techniques are pipeline and parallelism. Design approaches that meet both constraints of high-performance and small-size were presented in [1] and [2], where SHA-1 was implemented applying simultaneously the re-use and pipeline techniques. The SHA-1 implementation of [2], presented the highest throughput. In this paper the SHA-1 hash function is explored in depth and various implementations that have been proposed in the international literature are considered. Design aspects of performance and power dissipation are considered to explore and compare current implementations.

# Math behind SHA1 Algorithm

SHA1 algorithm can be described in two stages: pre-processing and hash computation. Pre-processing involves padding a message, parsing the padded message into m-bit blocks, and setting initialization values to be used in the hash computation. The hash computation generates a message schedule from the padded message and uses that schedule, along with functions, constants, and word operations to iteratively generate a series of hash values. The final hash value generated by the hash computation is used to determine the message digest [3].

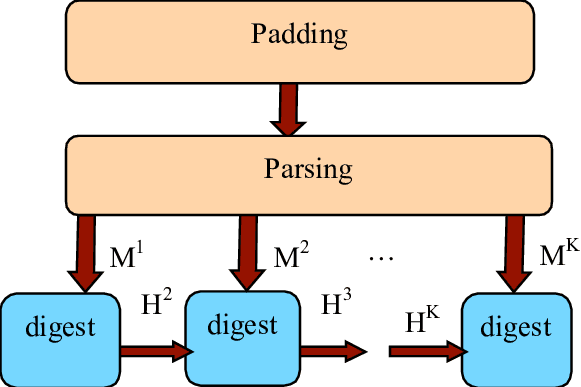


Figure 1. Simplified Architecture of SHA1 Algorithm

The algorithms differ in terms of the size of the blocks and words of data that are used during hashing or message digest sizes. Figure 1 presents the basic properties of these hash algorithms [3].

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm | Message Size (bits) | Block Size  (bits) | Word Size (bits) | Message Digest size (bits) |
| SHA1 | < 264 | 512 | 32 | 160 |

Table1. Secure Hash Algorithm Properties

Taking into consideration the properties in Table 1, the SHA1 Algorithm has two stages specified in detail as follow:

## Pre-processing:

Includes three different steps:

### Padding the message M, to ensure that the padded message is a multiple of 512.

### Parsing the message into message blocks: the message and its padding are parsed into N 512-bit blocks,M(1), M(2), … , M(N)

### Setting the initial hash value, H(0)

the initial hash value, H(0) shall consist of the following five 32-bit words, in hex:

## Hash Computation:

### SHA1 Functions:

SHA-1 uses a sequence of logical functions, f0, f1…f79. Each function ft, where , operates on three 32-bit words, x, y, and z, and produces a 32-bit word as output. The function ft(x, y, z) is defined as follows:

### SHA1 Constants:

SHA-1 uses a sequence of eighty constant 32-bit words, K0, K1, … , K79, which are given by

### Using these functions and constants, each message block, M(1), M(2), … , M(N), is processed in order, using the following steps for i = 0 to N:

#### Preparing the Message schedule:

#### Initialize the five working variables, a, b, c, d, and e, with the (i-1)st hash value:

#### For t=0 to 79:

{

}

#### Compute the ith intermediate hash value H(i):

The steps will be repeated N time results a160-bit message digest of message M as follow:

# Proposed Design

To test the functionality of the SHA1 module in a real-life setting, a system was designed to feed message as byte stream from a computer via UART to a Digilent Nexys A7 development board and display resulting SHA1 digest by VGA output. The system was implemented on the on-board Xilinx Artix-7 FPGA.

## UART Controller

To facilitate use of the SHA-1 core, a UART controller is developed. The UART controller takes a variable input from a keyboard and sends it to the SHA-1 core to be interpreted as the message via terminal. As previously mentioned, the message size of SHA-1 is up to 2^64 bits. Traditional inputs such as switches and buttons are impractical when dealing with large message sizes. With UART, it is possible to quickly send out large amounts of data either manually or with scripting.

## VGA Controller

A VGA controller is employed to display the output digest of the SHA-1 core onto a computer monitor. Since the output digest of SHA-1 is a lengthy 160 bits, displaying via LED’s or seven-segment displays is unwieldy. This VGA core displays all 160 bits as 40 hex digits. There are three components to displaying the proper digest values as shown in the figure below:

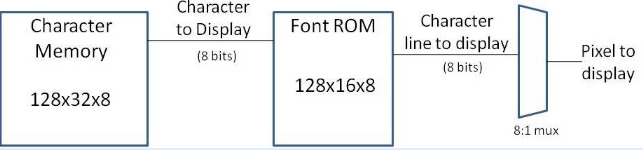


Figure 2: VGA Text Generation Block Diagram [4]

1. *A character memory to create a grid-like map of the monitor and assign each location a value.*
2. *A font ROM to assign each letter/number a binary value.*
3. *A multiplexer to determine whether a given location should contain a character, and if so, assign the proper value at each pixel.*

In addition to these three components, there is also a top-level driver for synchronization and timing according to the VGA standards. An example of the VGA output is shown in the figure below:

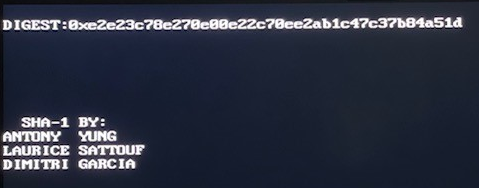


Figure 3: VGA Monitor Output

## SHA1 Core Implementation

As it is common to transmit and store data in multiple of bytes, this implementation integrated the circuitry of byte to word conversion to optimize specifically intake of byte stream. a 5-bit counter and a 2-bit counter was used to count 80 iterations for each message block such that division circuit can be waived i.e., simple 4x1 multiplexers can handle control of ft and Kt using 2-bit counter as select. The message schedule was generated and stored as needed, i.e., only the past 16 Wt processed was stored in a 16-word shift register, among them four word was used as input to XOR gate and rotated left by one by rearranging wires from the XOR gate output to provide Wt when ­.

# Result and Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| ***Implementation*** | ***Power Consumption*** | ***LUT Used*** | ***Flip-Flops Used*** |
| ***Kawazome*** [5] | 0.219 | 1171 | 1034 |
| ***Proposed Design*** | 0.222 | 727 | 816 |

*Table 2. Comparing Results of SHA1 implementation*

The integration of byte to word conversion circuit and specialization of performing SHA1 may contribute to the reduced LUTs and Flip-Flops used in this design comparing to a multifunctional design. However, there are room for improvement on power consumption for this design. As Wt are generated by a combinational circuit, a possible improvement is to limit transition of signal on that logic to reduce power consumption, either by converting it to a sequential circuit or mask the activity of the circuit with an enable signal possibly from states signals to reduce signal transitions during idle states.

# Conclusions

A hardware SHA1 digest core designed to reduce power consumption and resource usage was successfully implemented at register-transfer level by integrating byte to word conversion circuitry and eliminating usage of division circuit by separating message schedule counter into a 2-bit and 5-bit counters. A companion system was designed and implemented to handle serial byte stream message input via UART and output the resulting SHA1 digest to a VGA display. Future work includes further reduction of power consumption possibly by reducing idle state signal transitions.

# References

|  |  |
| --- | --- |
| [1] | N. Sklavos, P. Kitsos, E. Alexopoulos and O. Koufopavlou, "Open Mobile Alliance (OMA) security layer: Architecture, implementation and performance evaluation of the integrity unit," *New Generation Computing,* vol. 23, p. 77–100, 3 2005. |
| [2] | N. Sklavos, E. Alexopoulos and O. Koufopavlou, "O.G.: Networking data integrity: High speed architectures and hardware implementations," *Int. Arab J. Inf. Technol,* 2003. |
| [3] | Q. H. Dang, "Secure Hash Standard," National Institute of Standards and Technology, 2015. |
| [4] | "VGA Text Generator," Brigham Young University, [Online]. Available: https://ece320web.groups.et.byu.net/labs/VGATextGeneration/VGA\_Terminal.html. [Accessed 17 May 2021]. |
| [5] | I. Kawazome, "ikwzm/SECURE\_HASH: SHA-1,SHA-256,SHA-512 Secure Hash Generator written in VHDL(RTL) for FPGA(Xilinx and Altera).," 13 October 2017. [Online]. Available: https://github.com/ikwzm/SECURE\_HASH. [Accessed 17 May 2021]. |
| [6] | N. Sklavos, G. Dimitroulakos and O. Koufopavlou, "An ultra high speed architecture for VLSI implementation of hash functions," in *10th IEEE International Conference on Electronics, Circuits and Systems, 2003. ICECS 2003. Proceedings of the 2003*, 2003. |
| [7] | G. Selimis, N. Sklavos and O. Koufopavlou, "VLSI implementation of the keyed-hash message authentication code for the wireless application protocol," in *10th IEEE International Conference on Electronics, Circuits and Systems, 2003. ICECS 2003. Proceedings of the 2003*, 2003. |
| [8] | S. Dominikus, "A hardware implementation of MD4-family hash algorithms," in *9th International Conference on Electronics, Circuits and Systems*, 2002. |
| [9] | P. P. Chu, "VGA Controller II: Text," in *FPGA Prototyping by VHDL Examples*, John Wiley & Sons, Inc., p. 291–319. |
| [10] | J. M. Diez, S. Bojanić, L. Stanimirovicć, C. Carreras and O. Nieto-Taladriz, "Hash algorithms for cryptographic protocols : FPGA implementations," *10th Telecommunications Forum TELFOR'2002, Nov.,* 2002. |