Design of IP Theft Detection Using Scan Side Channel by SHA-512 ALGORITHM

Damera. Kamalakar[1], R.Seetha[2]
1 P.G Student, Department of Electronics and Communication Engineering
2 Assistant professor, J.B. institute of engineering and technology

***Abstract*—** This paper describes a preliminary performance evaluation of the implementation of Secure Hash Algorithm (SHA-512) building blocks on a cell-FPGA-like hybrid CMOS/nano device architecture. Such circuits will combine a semiconductor- transistor (CMOS) stack and a two-level nanowire crossbar with nanoscale two-terminal nanodevices (programmable diodes) formed at each crosspoint. The new design is based on two-cell fabric CMOL FPGA which can be used for mapping any arbitrary circuit. In addition, using a custom set of design automation tools quasi-optimium gate placing, placing, routing and rerouting are provided for SHA-512 fundamental building blocks. It is shown that such a design results in a circuit which is defect tolerant, much faster and strikingly denser than its CMOS counterpart.

***Keywords*—** *Secure Hash Algorithm, Nanoelectronic, CMOS/Nanodevice Architecture, CMOS FPGA, CMOL FPGA.*

I. Introduction

The purpose of a hash function is to produce a “fingerprint” of a file, message, or other block of data. The Secure Hash Algorithm (SHA) was developed by the U. S. National Institute of Standard and Technology (NIST) and published as a federal information processing standard (FIPS-180) in 1993; This version was reviewed and issued as FIPS 180-1 in 1995 and is generally referred to as SHA-1. In 2002, NIST produced a new revised version, FIPS 180-2, which defined three new versions of SHA, with hash value lengths of 256, 384, and 512 bits which have the same underlying structure and logical binary operations [1], [2]. Due to the essential need for security in Internet Protocol Security (IPSec) and Virtual Private Networks (VPN), an efficient and small-sized HMAC implementation, to authenticate both the source of a message and its integrity, is very important. The in inherent advantages of using VLSI chips for encryption and authentication are speed and more physical security. Software encryption has other features like portability and flexibility but is slow and suffers from insecurity in several aspects of key management and program manipulation. Commercial IP vendors introduced to the market high-speed FPGA implementation, which were area demanding and costly.

So far, several architectures for efficient VLSI implementation of SHA-512 have been proposed and their performance evaluated using ASIC libraries and FPGAs. Efficient hardware realization of SHA is still a motivational and challenging subject [3-10]. This paper follows a different approach, compared to the alternative solutions from academia and IP market, and presents a novel nanoelectronic-based solution for high-speed and very compact implementation of the algorithm. The new hybrid technology paradigm will certainly require rethinking of the current circuit architectures. Our proposed method is based on the |recently proposed digital nanoelectronic cell-FPGA-like hybrid semiconductor nanowire-nanodevice structure which combines a CMOS transistor stack and two levels of parallel nanowires. The basic idea for hybrid CMOS/nanodevice circuits is to combine the advantages of CMOS technology (flexibility, high fabrication yield and gain) with nanometer scale devices, which are self-assembled on a pre-fabricated nanowire fabric, enabling very high digital function density at modest fabrication cost. In addition, since nanoelectronic structures may never reach 100% yield, it is appropriate to use a cell-based FPGA type architecture in which one can reconfigure the circuit such that defective connections are rerouted and functioning nanodevices are used in the circuit. This new design is based on a two-cell CMOL FPGA fabric, which is a generalization of the single cell structure. This fabric is a uniform mesh of square shaped “tiles”, while each tile consists of 12 four transistor basic cells and one latch cell [11-15]. To evaluate the potential performance of the design we used a completely custom design automation tools and successfully mapped our circuits on CMOL FPGA and estimated their performance. The results have shown that, in addition to high defect tolerance, such implementation may have extremely high density (more than two orders of magnitude higher than that of usual CMOS FPGA with the same CMOS design rules) while operating at higher speed at acceptable power consumption

II.EXISTING WORK- SHA-256 ALGORITHM

SHA-2 is a widely used family of cryptographic hash functions. The family comprises six members distinguished by the size of the hash value. In this paper, we examine one member of the family, namely, SHA-256. The SHA-256 algorithm receives a message of an arbitrary length and produces a 256-bit-long digest [Fig. 2(a)]. At the first stage, the original message is padded, which makes its length an integer number of 512-bit chunks. The subsequent processing runs for each chunk sequentially.

The processing comprises a message schedule and 64 stages, called compression stages. The message schedule takes the 512-bit input and prepares 64 32-bit words, one for every compression stage. The first 16 words are a copy of the input chunk, and for the remaining 48 words, the schedule operation involves bit permutations, XOR operations, and a four-input 32-bit adder.

The compression stage receives an 8 × 32 bit hash value and produces an input to the next stage, in which six out of the eight words are a mere permutation of the input, and the remaining two words are the result of a five-element and a seven-element adder, respectively. The inputs to the adder are the words from the input of the stage, while some of them pass additional transformations, which include permutations, XOR, selectors, and a majority function. For the case of IP theft detection, we assume that the exact function of majority of combinational building blocks are known, and the objective is to learn how they are combined, what the structure of the pipeline is, and what the differences from the original function are, if they exist. Hence, the learning method is built around recognition of the known structures. The SHA-256 algorithm can be seen as an acyclic data flow graph with many repetitive stages along the way. the circuit is known, and the target is to discover the implementation details. Namely, each includes 6 × 32 pass-through connections, two 32-bit adders, one five-element, and one seven-element. In addition, the compression stage includes permutations, selectors, and majority functions.

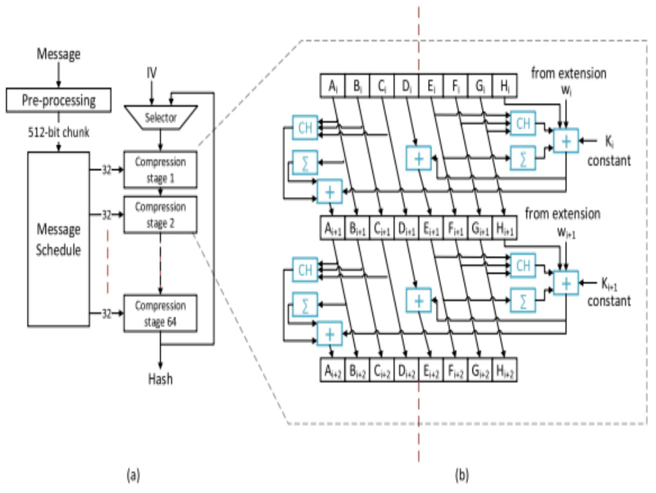


Fig. 1. SHA-256 algorithm block diagram. (a) SHA-256 execution flow, including preprocessing stage, message schedule, which outputs 64 × 32 bit words, and 64 compression stages. (b) Detailed diagram of two 256-bit-wide compression stages.

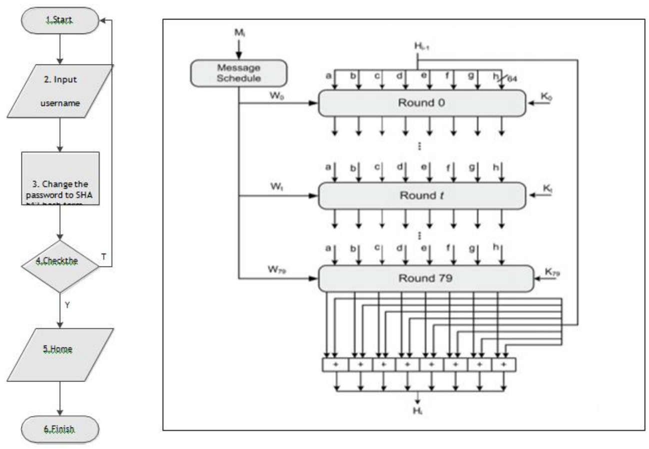
```
Algorithm 1 Probe(Circuit S and Vector v)
1: r || i := v                    ▷ Set registers and inputs state to v
2: o_{t−1} := o                   ▷ Sample outputs of S
3: Capture
4: return r||o_{t−1}              ▷ New register values and outputs
```

The implementer can decide on a number of pipeline stages by dividing the stages of the algorithm. Fig. 1(b) shows two stages of the SHA-256 inner loop. If the implementation dedicates one pipeline stage for one compression stage, the combinational logic between the corresponding flip-flops will include six 32-bit pass-through connections, and two 32-bit arithmetic sums: one of seven and the other of five elements. However, if two compression stages of the algorithm comprise one pipeline stage, the combinational logic for one pipeline stage will include four 32-bit pass-through paths, and four 32-bit arithmetic sums: of 5, 7, 11, and 17 elements.

Alternatively, if the main constraint is power or silicon real estate, even a single compression stage can be divided, and the same erformance-hungry applications will use deep pipelines, and latency-oriented designs will strive to combine as many calculations as possible in a single pipeline stage. Despite the countless configurations, clearly distinguishable structures can be found in most of them. For example, even without knowing the exact configuration, such as the number of inputs or additional logic, multiple bit adder structures have a distinct pattern of dependencies between input and result bits. Adders constitute the majority of SHA-256 complex building blocks; therefore, detecting adder-like structures is helpful for both partitioning the data into hierarchical structures and learning the exact function of these blocks.

III. PROPOSED SHA-512 ALGORITHUM

After Based on the flowchart presented in Figure 2, process no. 3 that was previously encrypted using MD5 is changed using SHA 512 method. So in that process, the data transmission in the form of input from password will be changed to SHA 512 hash form which has the hash value much longer than MD5 therefore, user data will be more secure from a vulnerability that can occur when using MD5 as described in Figure2.



The explanation of the conceptual image is as follows.

1. Users access the application and login to login to the application. The login process is done by sending data in the form of username and password. The process of sending data is done by changing the password data in the form of plaintext into SHA 512 cipher text hash.

2. The application server receives the data in the form of the hash value of the password and then forward it to the database. This process is performed to verify the hash sent by the user whether it is the same as the password hash stored in the database (hash function for storing password).

3. If the data is suitable then the user can enter and access the application.

Renewal is done by changing the existing hash method into SHA 512 hash method combined with the addition of SALT secret key. Implementation done at this stage is encoding by creating a patch that will be used to call a hash function during login. The plot of the calling process and the data changes for the username and password is first made before the encoding is done, so it can be known where the calling of the hash function calling can change the password to the ciphertext hash value. This process generates a flowchart. To implement our method we use two SHA-512 modules in parallel. A message is passed to SHA-512 modules in 64-bit chunks, alternatively. For a given message, there will be two 512-bit temporary signatures. There is a permutation unit that permutes two temporary signatures with the given secret key which is stored in a 1024×10-bit Block ROM. Each word of ROM indicates the position of corresponding bit in output signature. After permutation of the 1024-bit temporary signatures, the final signature becomes ready.

SHA-512 LOGIC

The algorithm takes as input a message with a maximum length of 2128 bit and produces as output a 512-bit message digest. The input is processed in 1024-bit blocks. The processing consists of the following steps [16].

Step1. Append padding bits. The message is padded so that its length is congruent to 896 modulo 1024.

Step2. Append length. A bloc ck of 128 bits is appended to the message. This bloc ck is treated as an unsigned 128-bit integer and contain ns the length of the original message before padding.

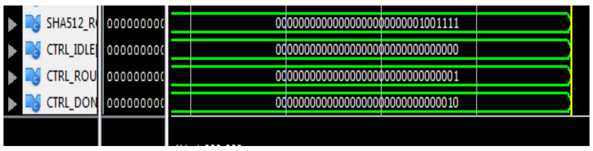
Step3. Initialize hash buffer. A 512-bit buffer is used to hold the intermediate and final results of hash function. The buffer can be represented as eight 64-bit registers (a, b, c, d, e, f, g, h). T These registers are initialized with some 64-bit hexadecimal values.

Step4. Process message in 102 24-bit blocks. The hearth of the algorithm is a module that consists of 80 rounds; the logic is illustrated in F Fig. 1. Each round takes as input the 512-bit buffer va alue abcdefgh, and updates the intermediate hash value , Hi-1. Each round t makes use of a 64-bit value Wt, derived from the current 1024 bit block being processed (Mi).

Step5: After all N 1024-bit b blocks have been processed; the output from the Nth s stage is the 512-bit message digest.

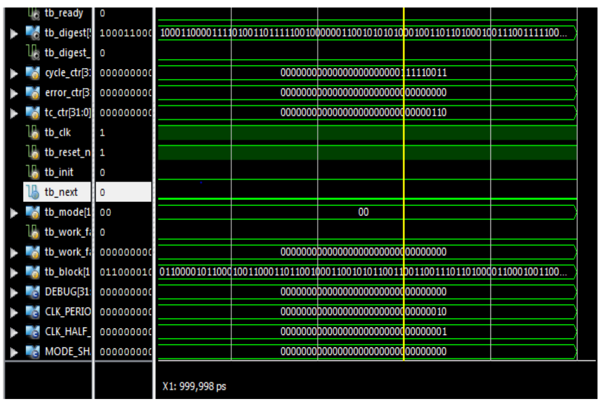
IV. SIMULATION RESULTS

Input:



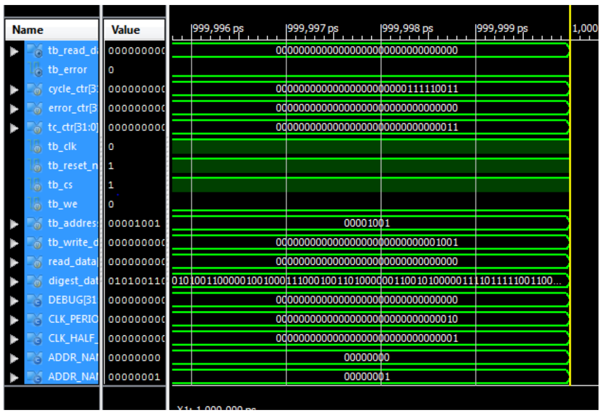
```
SHA512_R(   000000000   00000000000000000000000001001111
CTRL_IDLE[  000000000   00000000000000000000000000000000
CTRL_ROU    000000000   00000000000000000000000000000001
CTRL_DON    000000000   00000000000000000000000000000010
```

Output:



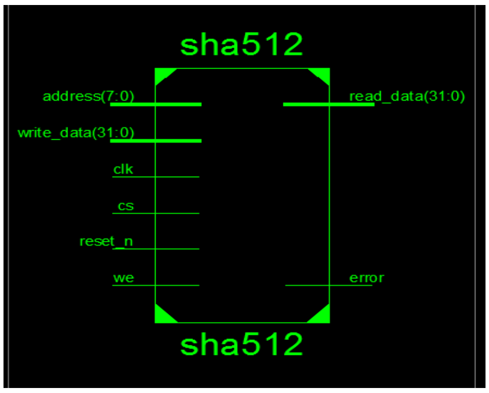
```
tb_ready      0
tb_digest[!   100011000   10001100001111010011011111001000001100101010100100110110100010011100111100...
tb_digest_    0
cycle_ctr[3   000000000   00000000000000000000000111110011
error_ctr[3   000000000   00000000000000000000000000000000
tc_ctr[31:0]  000000000   00000000000000000000000000000110
tb_clk        1
tb_reset_n    1
tb_init       0
tb_next       0
tb_mode[1     00          00
tb_work_fa    0
tb_work_fa    000000000   00000000000000000000000000000000
tb_block[1    011000010   01100001011000100110001101100100011001010110011001100111011010000110001001100...
DEBUG[31      000000000   00000000000000000000000000000000
CLK_PERIO     000000000   00000000000000000000000000000010
CLK_HALF_     000000000   00000000000000000000000000000001
MODE_SH.      000000000   00000000000000000000000000000000

X1: 999,998 ps
```

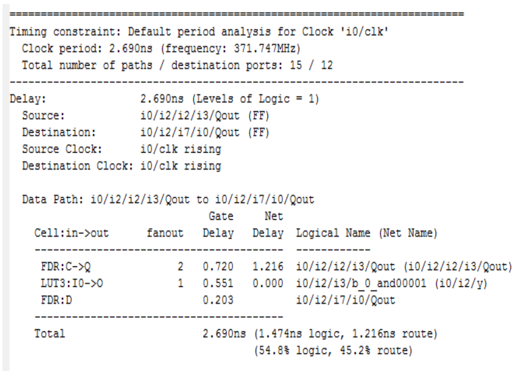


```
Name          Value       |999,996 ps    |999,997 ps    |999,998 ps    |999,999 ps    1,000
tb_read_da    000000000   00000000000000000000000000000000
tb_error      0
cycle_ctr[3   000000000   00000000000000000000000111110011
error_ctr[3   000000000   00000000000000000000000000000000
tc_ctr[31:0]  000000000   00000000000000000000000000000011
tb_clk        0
tb_reset_n    1
tb_cs         1
tb_we         0
tb_addres     00001001    00001001
tb_write_d    000000000   00000000000000000000000000001001
read_data[    000000000   00000000000000000000000000000000
digest_dat    010100110   0101001100000100100011100010011010000001100101000001111011111001100...
DEBUG[31      000000000   00000000000000000000000000000000
CLK_PERIO     000000000   00000000000000000000000000000010
CLK_HALF_     000000000   00000000000000000000000000000001
ADDR_NAI      00000000    00000000
ADDR_NAI      00000001    00000001
```

RTL SCHEMATIC:



TIMING REPORT:



```
================================================================================
Timing constraint: Default period analysis for Clock 'i0/clk'
  Clock period: 2.690ns (frequency: 371.747MHz)
  Total number of paths / destination ports: 15 / 12
--------------------------------------------------------------------------------
Delay:               2.690ns (Levels of Logic = 1)
  Source:            i0/i2/i2/i3/Qout (FF)
  Destination:       i0/i2/i7/i0/Qout (FF)
  Source Clock:      i0/clk rising
  Destination Clock: i0/clk rising

  Data Path: i0/i2/i2/i3/Qout to i0/i2/i7/i0/Qout
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDR:C->Q              2   0.720   1.216  i0/i2/i2/i3/Qout (i0/i2/i2/i3/Qout)
     LUT3:I0->O            1   0.551   0.000  i0/i2/i3/b_0_and00001 (i0/i2/y)
     FDR:D                     0.203          i0/i2/i7/i0/Qout
    ----------------------------------------
    Total                      2.690ns (1.474ns logic, 1.216ns route)
                                       (54.8% logic, 45.2% route)
```

V.CONCLUSION

The security of hash functions is determined by the size of their outputs, referred to as hash values, n. The best known attack against these functions, the "birthday attack", can find a pair of messages having the same hash value with a work factor of approximately 2n/2. Therefore, for a SHA-512 module, complexity of the best attack is 2256. For our method each SHA-512 has the same attack complexity, but our permutation process increases the attack complexity in proportion to the size of the secret key. If we use a 1024-bit secret key, permutation complexity becomes which yields the total complexity of 2 !210 10! × 2256 and it's much more secure than the SHA-512 algorithm.

VI.FUTURE WORK

Future research related to this paper can study harnessing the flow for additional applications. One important application that we are exploring is the detection of deviation of the design from the original function, which may indicate the presence of Trojan hardware.

REFERENCES

[1] M.Pecht and S. Tiku, "Bogus!" IEEE Spectrum, vol. 43, no. 5, pp. 37–46, May 2006.
[2] G. Qu and M. Potkonjak, Intellectual Property Protection in VLSI Designs. Boston, MA, USA: Kluwer, 2004.
[3] W. P. Griffin, A. Raghunathan, and K. Roy, "CLIP: Circuit level IC protection through direct injection of process variations," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 20, no. 5, pp. 791–803, May 2012.
[4] F. Koushanfar and G. Qu, "Hardware metering," in Proc. Design Autom. Conf., 2001, pp. 490–493.
[5] F. Koushanfar, "Provably secure active IC metering techniques for piracy avoidance and digital rights management," IEEE Trans. Inf. Forensics Security, vol. 7, no. 1, pp. 51–63, Feb. 2012.
[6] T. Guneysu, B. Moller, and C. Paar, "New protection mechanisms for intellectual property in reconfigurable logic," in Proc. 15th Annu. IEEE Symp. Field-Programm. Custom Comput. Mach. (FCCM), Apr. 2007, pp. 287–288.
[7] I. Torunoglu and E. Charbon, "Watermarking-based copyright protection of sequential functions," IEEE J. Solid-State Circuits, vol. 35, no. 3, pp. 434–440, Mar. 2000.
[8] M. Lewandowski, R. Meana, M. Morrison, and S. Katkoori, "A novel method for watermarking sequential circuits," in Proc. IEEE Int. Symp. Hardware-Oriented Security Trust, Jun. 2012, pp. 21–24.
[9] E. Charbon, "Hierarchical watermarking in IC design," in Proc. IEEE Custom Integr. Circuits Conf., May 1998, pp. 295–298. [10] G. T. Becker, M. Kasper, A. Moradi, and C. Paar, "Side-channel based watermarks for integrated circuits," in Proc. IEEE Int. Symp. HardwareOriented Security Trust (HOST), Jun. 2010, pp. 30–35.
[11] Y.-C. Fan, "Testing-based watermarking techniques for intellectualproperty identification in SoC design," IEEE Trans. Instrum. Meas., vol. 57, no. 3, pp. 467–479, Mar. 2008.
[12] J. L. Wong, D. Kirovski, and M. Potkonjak, "Computational forensic techniques for intellectual property protection," IEEE Trans. Comput.- Aided Design Integr. Circuits Syst., vol. 23, no. 6, pp. 987–994, Jun. 2004.
[13] S. Guilley, L. Sauvage, J. Micolod, D. Réal, and F. Valette, "Defeating any secret cryptography with SCARE attacks," in Progress in Cryptology—LATINCRYPT. Berlin, Germany: Springer, 2010, pp. 273–293. [14] T. M. Mitchell, Machine Learning. New York, NY, USA: McGraw-Hill, Mar. 1997.