Improvement of Speed in Lightweight Cryptographic Algorithm-Spongent Family

V. Gangapriyadharshini[1] M. Nisha Angeline[2] Dr. S. Valarmathi[3]
[1]PG Scholar [2]Assistant Professor [3]Professor and Head
[1,2,3]Department of Electronics Communication Engineering
[1,2,3]Velalar College of Engineering and Technology, Erode, Tamilnadu, India.

Abstract— Security is a big issue for all networks in today's enterprise environment. In order to provide security, hash functions with various algorithms are used. The lightweight hash functions designed based on the sponge constructions. The result of hash function is called SPONGENT. Light weight hash functions which takes an variable length of input and produces fixed length of output. Various parameters of SPONGENT is referred as n/c/r for different hash sizes n, capacities c, and rates r. It has different hash sizes of n={88,128,160,224,256}, covering many security applications. In this project, pipelining and parallel processing techniques are used to construct high speed VLSI architecture for the SPONGENT family devices and to improve the performance of the system.

Key words: Hash Function, Lightweight Cryptography, Sponge Construction, Present, SPONGENT

I. INTRODUCTION

Security requirements are the greatest issue in communication. In order to provide security, hash functions with various algorithms are used. Secret should be kept confidential. The hash functions are used to maintain secure informations. The hash value checks whether the information from the transmitter to receiver is original or not. The secure hash algorithm (SHA) is a family of cryptographic hash functions published by "National Institute of Standards And Technology"(NIST) as a U.S. Federal Information Processing Standard(FIPS). SHA includes the following algorithm with the help of different digest length such as SHA-1, SHA-2, SHA-3.

The SPONGENT lightweight cryptographic algorithm comes under the SHA-3 family. SPONGENT is a sponge construction based on a PRESENT type permutation. It takes a variable length of input and produce a fixed length of output. This paper focus on to construct the high speed VLSI architecture for the SPONGENT family to improve the speed by using pipelining and parallel processing techniques which can be applied in Linear Feedback Shift Register (LFSR).

II. EXISTING METHOD

The sponge based lightweight hash functions SPONGENT having a smaller foot print than the another lightweight hash functions PRESENT and QUARK. Area requirement is a challenging task in the comparison of lightweight hash functions.

The design of PHOTON chooses for a higher rate and larger area occupation, while SPONGENT used for a lower rate and lower area requirements. One can either make the resulting design smaller and slower or larger and faster by slightly changing the rate of a sponge-based hash function

SPONGENT Lightweight hash functions are designed by instantiate the sponge construction with a PRESENT-type permutation. The various parameters are referred as SPONGENT-n/c/r for different hash sizes n, capacities c, rates r, for five different hash sizes of n = {88, 128,160, 224, 256}, It is a hermetic sponge, covering many security applications that is Full preimage and second-preimage security, Reduced second-preimage security, Reduced preimage and second-preimage security. spongent offers a lot of flexibility and speed in terms of serialization[2].

1) Full preimage and second-preimage security
2) Reduced second-preimage security
3) Reduced preimage and second-preimage security

III. THE DESIGN OF SPONGENT

The SPONGENT is based on sponge construction and PRESENT-type permutation. It a simple iterated design that takes a variable length input and can produce an output of an fixed length. The size of the internal state bit is b=r+c>=n, where b is called width, n is the hash sizes, r is the rate and c is the capacity. The sponge constructions are in three phases[1].

A. *Initialization Phase*

The message is added by a single bit 1 followed by a required no. of 0 bits to a multiple of r bits. Then it is cut into blocks of r bits.

B. *Absorbing Phase*

The first r-bits of the state is XOR-ed into the r-bit input message blocks.

C. *Squeezing Phase*

The first r-bits of the state are returned as output, interleaved with application of the permutation Πb.

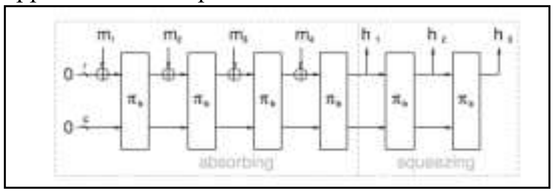


Fig. 1: Sponge Construction

IV. PARAMETERS

The STATE can be evolved by the substitution box layer and permutation layer. The number of rounds depends on block size b. ICounterb(i) is the state of an LFSR dependent on b and is added to the rightmost bits of STATE. counterIb(i) is the value of ICounterb(i) with its bits in reversed order and is added to the leftmost bits of STATE. The following blocks are generalizations of the PRESENT structure.

A. Substitution Box Layer

SubBytes transform is a simple transform which converts 4bit data to other 4 bit data. which is applied b/4 times in parallel. Important is that different 4 bit data are always transformed into different 4 bit data. If this condition is not satisfied, the transformed data can not be recovered. The 4-bit S-box is the major block of function logic used in a serial low-area implementation of SPONGENT. It fulfills the differential and linear properties.[5]. This aims to restrict the linear hull effect discovered in round-reduced PRESENT. The action of the S-box in hexadecimal notation is given

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	E	D	B	0	2	1	4	F	7	A	8	5	9	C	3	6

Table 1: The Action of S-Box.

B. pLayer:

This is extension of the PRESENT bit-permutation and moves bit j of STATE to bit position Pb(j). The function of the bit permutation pLayer is to provide good diffusion, by acting together with the S-box, while having a limited impact on the area requirements. This is its main goal, while a bit permutation may occupy additional space in silicon

$$P_b(j) = \begin{cases} j \cdot b/4 \mod b-1, & \text{if } j \in \{0,\dots,b-2\} \\ b-1, & \text{if } j = b-1, \end{cases}$$

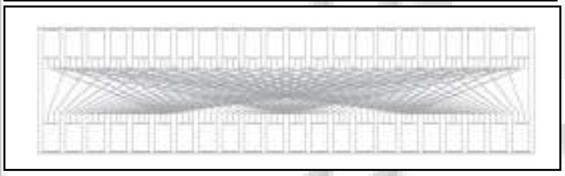


Fig. 2: The Bit Permutation Layer of SPONGENT-88 at the Examples of pLayer

C. Counter:

This is one of the four-bit LFSRs. The LFSR is clocked once every time its state has used and its final value is all ones. If ζ is the root of unity in corresponding binary field, the n-bit LFSR is defined by the polynomials given below are used for the SPONGENT variants. The counters are mainly aimed to prevent sliding properties.

LFSR size (bit)	Primitive Polynomial
6	$\zeta^6 + \zeta^5 + 1$
7	$\zeta^7 + \zeta^6 + 1$
8	$\zeta^8 + \zeta^4 + \zeta^3 + \zeta^2 + 1$
9	$\zeta^9 + \zeta^4 + 1$

Table 2: Sizes and Initial Values of All the LFSRs.

The Linear Feedback Shift Register is based on linear xor feedback logic in which the initial value of the shift register, shift register taps, and feedback logic determines the output sequences.

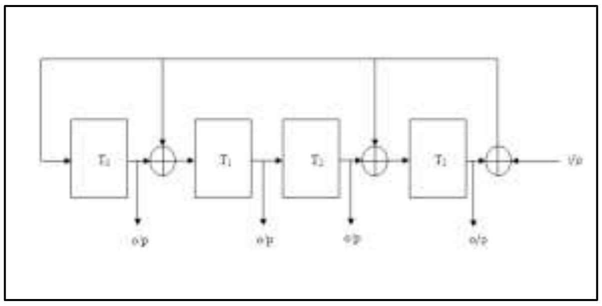


Fig. 3: LFSR

Hardware figures of SPONGENT implemented using open core 45nm library (NANGATE). To further stress on the later issue, we present the performance of SPONGENT using four different CMOS technologies. The comparison is only possible once the designs are implemented on the same hardware platform, using the same standard cell library and the same synthesis tool.

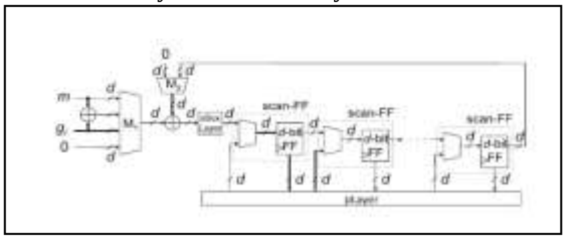


Fig. 4: Serial Datapath

It provides considerably small circuit size. we first focus on serialized designs. We explore four different data path sizes (d) for each of the SPONGENT variants and we focus on d={4,8,b/2,b} These four implementations are clearly marked in the figure by the four marker points for each different SPONGENT version. An architecture representing our serialized data path is depicted in Fig. 3. The control logic consists of a single counter for the cycle count and some extra combinational logic to drive the selection signals of the multiplexers[4]. Further reduce the area use scan flip-flops, which act as a combination of two input multiplexer and an ordinary D flip-flop. Instead of providing a reset signal to each flip-flop separately, use two zero inputs at the multiplexers M1 and M2 to correctly initialize all the flip-flops. This additionally reduces hardware resources, as the scan flip-flops with a reset input approximately require an additional GE per bit of storage. With gi we denote the value of lCounterb(i) in round i. lCounterb(i) is implemented as an LFSR. The pLayer module requires no additional logic except some extra wiring. Using the most serialized implementation, the smallest variant of the SPONGENT family, SPONGENT-88/80/8, can be implemented using only 738 GE. Though some of this advantage is at the expense of a performance reduction, also less serialized implementations result in area requirements significantly lower than 10 kGE. To demonstrate this, implement all the SPONGENT variants in Fig. 4.

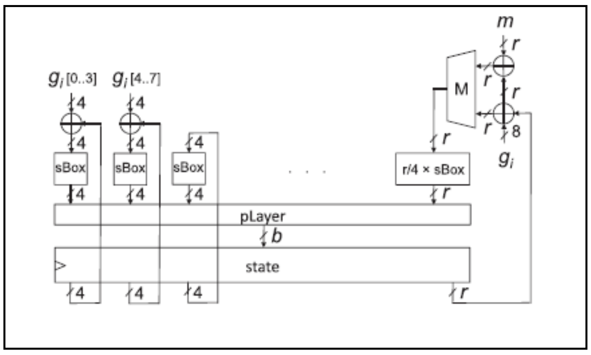


Fig. 5: Parallel Datapath

Every round now requires a single clock cycle, therefore resulting in faster, yet rather compact designs.

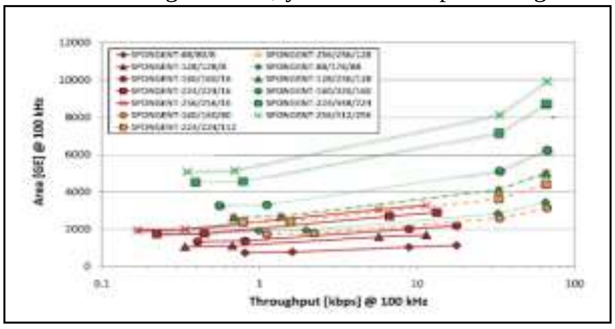
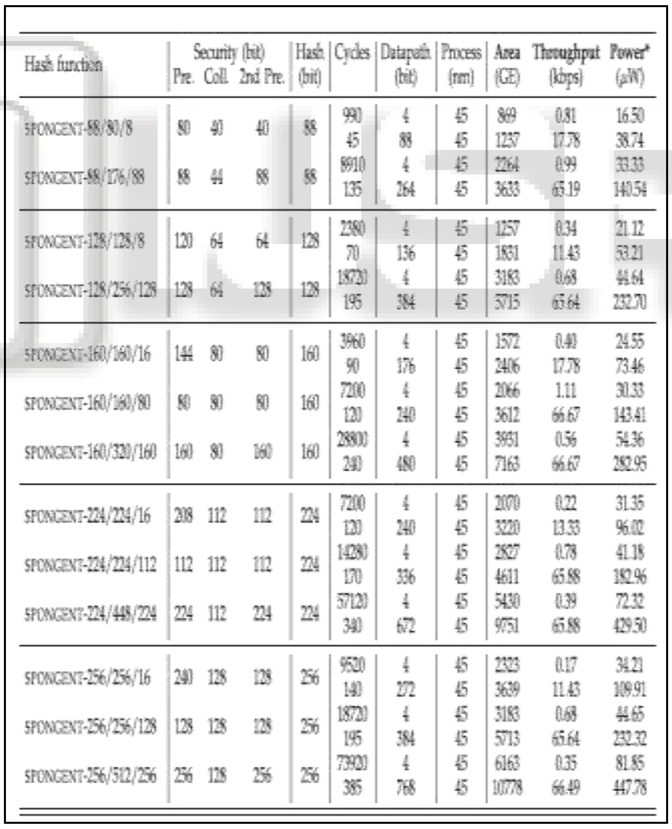


Fig. 6: Area versus Throughput Tradeoff of the SPONGENT Family

The main cause of the present variance is a difference in relative cell sizes, which is directly related to the library type. Note that the serial implementation of our designs consists of more than 90 percent of sequential logic. A single scan flip-flop consumes at least 7.67 GE in NANGATE45. The same cell consumes 6.25 and 6.67 GE in UMC130 and UMC180, respectively. The NXP90 library has significantly smaller flip-flops that are the main area consumers in the case of SPONGENT family, and thus provides a considerably better hardware performance. Present the obtained hardware figures for all of the SPONGENT variants. The smallest implementation consuming only 738 GE, which achieve more than 66 kbps at 100 kHz[4].



Hash function	Security (bit) Pre.	Coll.	2nd Pre.	Hash (bit)	Cycles	Datapath (bit)	Process (nm)	Area (GE)	Throughput (kbps)	Power* (µW)
SPONGENT-88/80/8	80	40	40	88	990	4	45	869	0.81	16.50
					45	88	45	1237	17.78	38.74
SPONGENT-88/176/88	88	44	88	88	8910	4	45	2264	0.99	33.33
					135	264	45	3633	65.19	140.54
SPONGENT-128/128/8	120	64	64	128	2380	4	45	1257	0.34	21.12
					70	136	45	1831	11.43	53.21
SPONGENT-128/256/128	128	64	128	128	18720	4	45	3183	0.68	44.64
					195	384	45	5715	65.64	232.70
SPONGENT-160/160/16	144	80	80	160	3960	4	45	1572	0.40	24.55
					90	176	45	2406	17.78	73.46
SPONGENT-160/160/80	80	80	80	160	7200	4	45	2066	1.11	30.33
					120	240	45	3612	66.67	143.41
SPONGENT-160/320/160	160	80	160	160	28800	4	45	3931	0.56	54.36
					240	480	45	7163	66.67	282.95
SPONGENT-224/224/16	208	112	112	224	7200	4	45	2070	0.22	31.35
					120	240	45	3220	13.33	96.02
SPONGENT-224/224/112	112	112	112	224	14280	4	45	2827	0.78	41.18
					170	336	45	4611	65.88	182.96
SPONGENT-224/448/224	224	112	224	224	57120	4	45	5430	0.39	72.32
					340	672	45	9751	65.88	429.50
SPONGENT-256/256/16	240	128	128	256	9520	4	45	2323	0.17	34.21
					140	272	45	3639	11.43	109.91
SPONGENT-256/256/128	128	128	128	256	18720	4	45	3183	0.68	44.65
					195	384	45	5713	65.64	232.32
SPONGENT-256/512/256	256	128	256	256	73920	4	45	6163	0.35	81.85
					385	768	45	10778	66.49	447.78

Table 3: Hardware Performance of SPONGENT Family

V. Proposed System

Parallel processing and Pipelining techniques are used in proposed system.

Pipelining leads to a reduction in the critical path, which can increase the sample speed or reduce power consumption at the same speed. Parallel processing techniques require multiple outputs, which are computed in parallel in a clock period. Therefore, the effective sample speed is increased by the level of parallelism. Parallel processing should be applied in the Linear Feedback Shift Register(LFSR) and then Pipelining should be applied in LFSR. Speed get increased by this techniques.

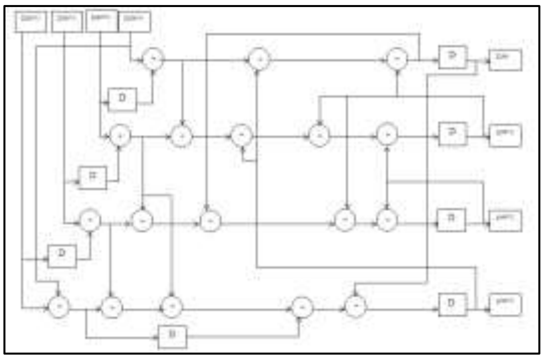


Fig. 7: Proposed Architecture with Parallel Processing

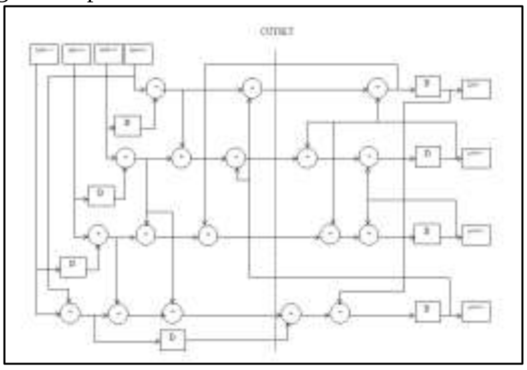


Fig. 8: Proposed Architecture with Pipelining

VI. Result Analysis

Simulation results are shown below:

A. *Output for Spongent Architecture:*

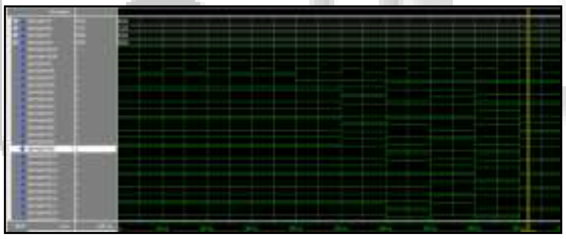


Fig. 9: Spongent Architecture Waveform

B. *Output for Parallel Processing*

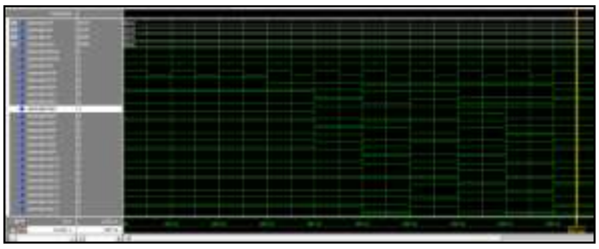


Fig. 10: Parallel Processing Waveform

C. *Output for Pipelining*

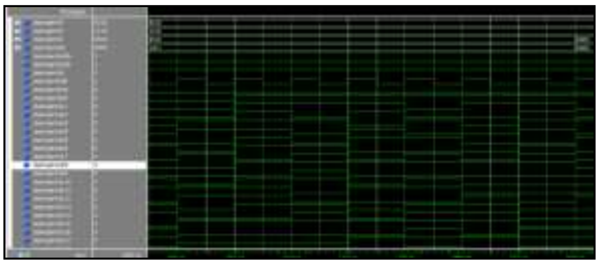
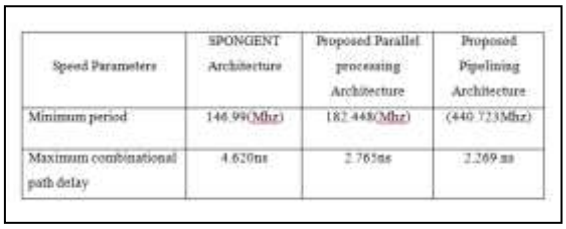


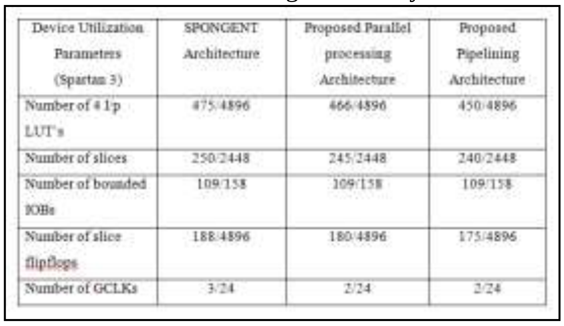
Fig. 11: Pipelining Waveform

– Implementation Results Are Shown Below:



Speed Parameters	SPONGENT Architecture	Proposed Parallel processing Architecture	Proposed Pipelining Architecture
Minimum period	146.99(Mhz)	182.448(Mhz)	(440.723Mhz)
Maximum combinational path delay	4.620ns	2.765ns	2.269 ns

Table 4: Design Summary



Device Utilization Parameters (Spartan 3)	SPONGENT Architecture	Proposed Parallel processing Architecture	Proposed Pipelining Architecture
Number of 4 I/p LUT's	475/4896	466/4896	450/4896
Number of slices	250/2448	245/2448	240/2448
Number of bounded IOBs	109/158	109/158	109/158
Number of slice flipflops	188/4896	180/4896	175/4896
Number of GCLKs	3/24	2/24	2/24

Table5: Comparison of Speed Parameters

VII. Conclusion

Existing method uses the SPONGENT lightweight cryptographic hash function. Though the hardware architecture of SPONGENT is optimized in area, it consist of more number of registers and also it is bit oriented not byte oriented. So the latency is increased. Here parallel processing and pipelining process can be applied in existing LFSR. The hybrid method of pipelining and parallel processing increase the speed of the architecture. The effective sample speed is increased by the level of parallelism.

References

[1] Aumasson., J.P., Henzen, L., Meier, W., Naya-Plasencia, M. (2010), "Quark: A Lightweight Hash", Springer, Vol.6225, pp.1-15.
[2] Bertoni, G., Daemen, j., Peeters, M., Van Assche, G (2008) "On TheIndifferentiability of the Sponge Construction", In smart, N.P. (ed) EUROCRYPT'08. Springer, Vol.4965,pp.181-197.
[3] Bertoni, G., Daemen, j., Peeters, M., Van Assche, G (2010) "Sponge Based Pesudo Number Generators", In: Mangard, S, Standaert, F.X.(ed) CHES. Springer, Vol. 6225,pp.33-47.
[4] Bogdanov, A., Knezevic, M., Leander, G., Toz, D., Varici, K., Verbauwhede, (2011) "SPONGENT: A Lightweight Hash Function", In: Preneel, B., Takagi, T. (eds). LNCS, Springer, vol.6917,pp.312-325.
[5] Bogdanov, A., Knezevic, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, (2007) "PRESENT: An Ultra-Lightweight Block Cipher", In: Paillier, P., Verbauwhede, I.(eds) LNCS, Springer, Vol.4727,pp.450-466.
[6] Cho, J.Y, (2010) "Linear Cryptanalysis of Reduced Round PRESENT" In: Pieprzyk, J. (ed). CT-RSA'10. LNCS, Springer, Vol.5985, pp.302-317.