High Throughput Area Efficient Architecture for Light Weight Cryptography

Vanitha M and Subha

School of Information and Technology, VIT University, Vellore - 632014, Tamilnadu, India

Abstract
Hummingbird algorithm is one of the recently proposed light weight cryptographic algorithms targeted for resource constrained devices like RFID (radio frequency identification), smart cards and majority of wireless sensor nodes. The main advantage of this algorithm is that it provides adequate security with smaller block size. As per the previous works on this algorithm, area and performance are two main design tradeoff of this algorithm. Performance is increased by loop unrolling and area is optimized by looping. So optimization in both area and performance is a big challenge. This work, proposes efficient hardware architecture for the hummingbird algorithm using partial loop unrolling which concerns both the area and performance of hardware implementation. The overall architecture was modeled using verilog HDL and synthesized using cadence RTL compiler with $45nm$ Technology from TSMC. Proposed design also implemented in low cost Spartan 3 FPGA board and the results are compared with the existing implementations. Results show that there is an area reduction of around 6% and throughput almost get doubles.
Keywords Hummingbird; Lightweight cryptography; RFID.

1 Introduction

The importance of low cost devices like RFID (Radio frequency identification), smart cards and various wireless sensor devices is increasing in our present day life. So the security of such devices is very important. Since these devices are extremely resource constrained in terms of computing power, battery power supply and memory, the standardized cryptographic algorithms such as AES(Advanced encryption standard) , DES(Data encryption standard), which are well focused on software implementation rather than hardware, cannot be used as it is in these devices. So a certain class of cryptographic algorithms known as light weight cryptography is evolved. The design metrics of light weight cryptography are security, cost, and performance. Practically it is difficult to optimize all the three design goals.

1.1 Related Works

Many lightweight cryptographic algorithms are presented until now, a light weight algorithm named HIGHT is proposed with 3048 gate equivalents (GE) which is much faster than AES[1]. Scalable Encryption Algorithm (SEA), with a block

size and key size of 96 bits, and word size of 8 bits and 93 rounds operation can encrypt one data block within 1428 clock cycles and 3758 GE[2].Slight Modifications is done on Classical Block Cipher lightweight DES variant called DESL (DES Lightweight)[3]. ASIC implementation of the same requires 1848 GE and it can encrypt one 64-bit data block in 144 clock cycles. The implementation of DESL has approximately 20% smaller chip size than DES. Key whitening technique can be useful to improve the security of cipher in DESXL algorithm. For encrypting the plaintext, it requires 2,170 GEs and 144 clock cycles. PRESENT algorithm is a lightweight substitution, permutation based block cipher, which operates on 32 rounds, key size of 80 or 128 bits and 64 bit block size. PRESENT serial version can be implemented with 1000GEs[4].

The Hummingbird algorithm is the one of the recently presented ultra-light weight cryptographic algorithm[5]. The size of the key and the internal state of Hummingbird provides adequate security level for many embedded applications. The present researches are going on the development and different implementation of the same because of the simplicity in the architecture. Until now the hummingbird has been implemented on different target platform, software as well as in hardware and shows good efficiency in both. Xinxin fan implemented the algorithm on 16 bit as well as 4-bit microcontroller.[6] Daniel engels implemented the architecture on 8-bit microcontroller.[7] Xinxin fan implemented both area oriented and throughput oriented design of hummingbird on low cost FPGA.[6,8] Biao min proposed another efficient hardware implementation of the same FPGA.[9] Ismail san proposed yet another implementation on FPGA using coprocessor approach.[10] All these implementation shows that this algorithm works well on different target platforms.

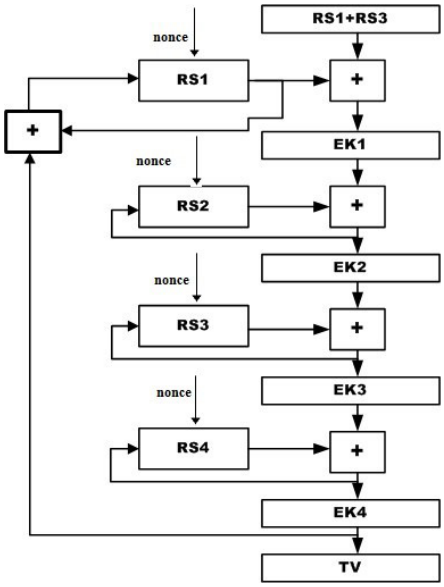
As per the previous works, Hummingbird can be implemented in different platforms but each of which is focused on either optimizing area or optimizing the speed. Here we are presenting an efficient architecture for the hummingbird algorithm focusing to optimize the area as well as the speed. Comparison is made with the existing implementations. The synthesized result shows the increase in throughput by 111% and reduction in area by 6.6%.

The remaining portion of this paper is organized as follows. Section 2 presents the standard Hummingbird algorithm, with its initialization, encryption and decryption steps. Next, Section 3 discuss about the proposed architecture for increasing the throughput and reducing the area. Section 4 presents the simulation results, synthesis outcome in FPGA and ASIC platform and their comparison with the existing architecture section 5 concludes this work.

2 Standard Hummingbird algorithm

Hummingbird is the recently proposed ultra-light weight cryptographic algorithm, which is the combination of block cipher and a stream cipher. The design of hummingbird consists of 16-bit block size, 256-bit key size, and 80-bit internal state. The main advantage of this algorithm is, it is having smaller block size compared to other algorithms and provide sufficient security even though the block size is small.

The overall structure of the hummingbird cryptographic algorithm includes four 16-bit block ciphers EKi(i=1,2,3,4), four 16-bit internal state registers RSi (i= 1,2,3,4), and a 16-bit LFSR (linear feedback shift register). The 256-bit key with 4 internal registers and LFSR provides adequate security level. For each block EKi., 256-bit secret key is divided into four 64-bit sub keys Ki(i=1,2,3,4).



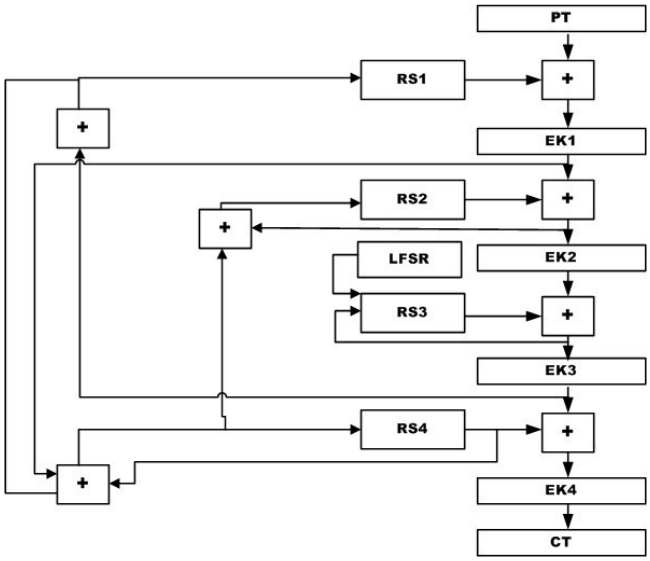
Notations Used	
RS1-RS4	Internal State registers to hold nonce value
EK1-EK4	State registers to hold the modulo addition values of 16 bit register RS with EK
+	Modulo addition
TV	Register to store the final output after 4 rounds of initialization

Fig. 1 Initialization

2.1 Initialization

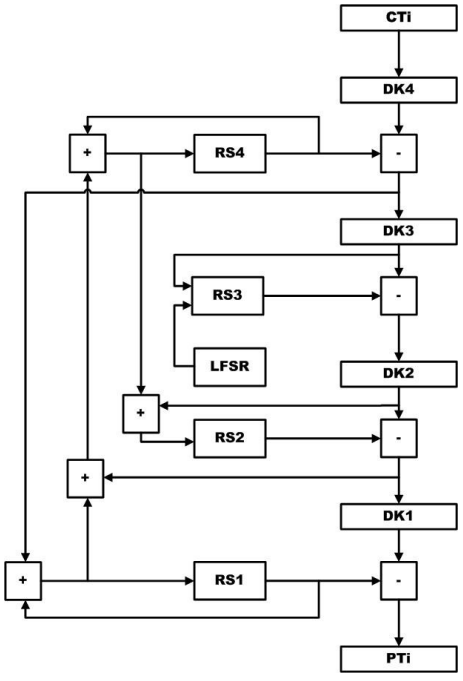
The initialization process shown in Fig.1 initialize the four internal states registers E_{K1} to E_{K4} and get the LFSR initial value before encryption starts. The four internal state registers are first loaded with four 16-bit random NONCE values. Taking RS1 + RS3 as input data, four block ciphers are consecutively executed four times and the states are updated accordingly as shown in Fig.1. The final output after four iteration is shown in the register TV, which is used to get the initial value of LFSR and used to update the state RS3 in encryption process.

The LFSR is used not only for RS3 updating but also to ensure that period of the internal states are at least 2^{16}.



Notations Used	
RS1-RS4	Internal State registers
EK1-EK4	16 bit register to hold the block cipher value after modulo addition
+	Modulo addition
PT	Plain Text
CT	Cipher Text

Fig. 2 Encryption



Notations Used	
RS1-RS4	Internal State registers
DK1-DK4	16 bit register to hold the value after modulo subtraction
-	Modulo subtraction
+	Modulo addition
PTi	Plain Text
CTi	Cipher Text
LFSR	Linear Feedback Shift Register

Fig. 3 Decryption

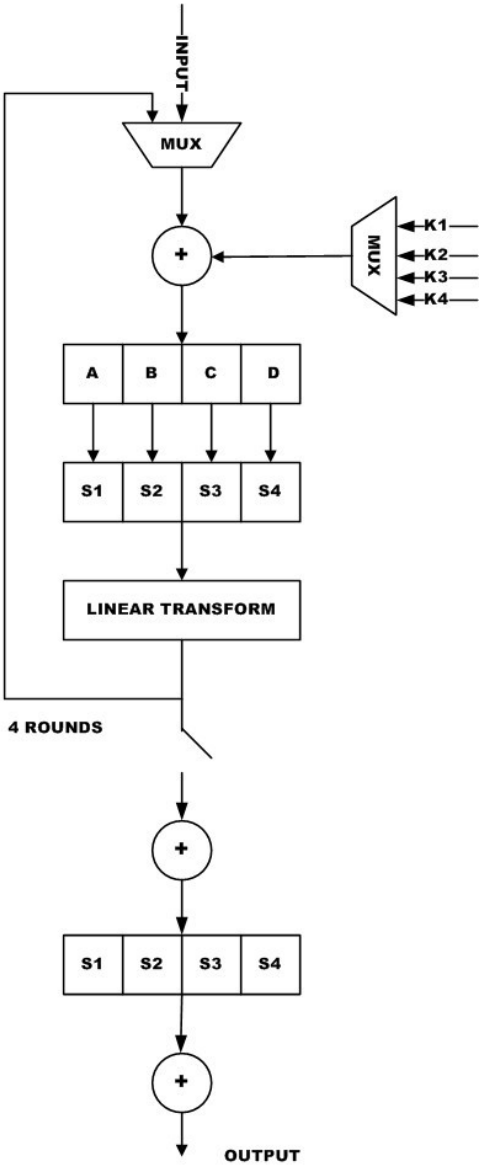


Fig. 4 Block cipher

2.2 Encryption/Decryption

After the initialization process, encryption starts by taking the input as the Plain Text(PT) followed by the modulo 2^{16} addition (shown in fig.2 as +) of plain text with internal state register RS1 and the result is applied to the block cipher EK1. The whole process of encryption is shown in Fig.2.

This process is repeated for four times and produces the cipher text. When all the four block ciphers are completed, the RSi state register is updated accordingly. Decryption process is just reverse operation of the encryption as shown in Fig.3.

2.3 Block cipher

Block cipher used in encryption process as shown in Fig.4. It consists of four rounds of operation and a final round. One regular round operation consists of a key mixing step, a substitution step and a linear transformation step. The sub-key of 64-bit is separated into four 16-bit round keys which will be used in the next corresponding rounds respectively. In the key mixing process, plaintext block uses an exclusive-OR with the round-key. The S-box produces the results step by step. The substitution round uses 4 Serpent-type S-boxes with the input and output of 4-bits. Table 1 shows the substitution value of 4bits used for substitution process. Here each 4-bit input is substituted with another 4-bit to make confusion in output. For the final step it uses one substitute step. There is no linear transformation step in the final round. The linear transformation step is defined in equation (1) where an XOR operation is performed between Din and 6 times right shifted value of Din and 10 times right shifted value of Din.

$$Dout = Din^{\wedge}(Din << 6)^{\wedge}(Din << 10) \tag{1}$$

Table 1 Substitution S box values

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
s1	8	6	5	f	1	c	a	9	e	b	2	4	7	0	d	3
s2	0	7	e	1	5	b	8	2	3	a	d	6	f	6	2	9
s3	2	e	f	5	c	1	9	a	b	4	6	8	0	7	3	d
s4	0	7	3	4	c	1	a	f	d	e	6	b	2	8	9	5

3 Proposed Architecture

Until now, hummingbird is implemented across different target platform. Each of that implementation is either focusing area optimization by looping technique or throughput optimizing by loop unrolling of block cipher. The proposed block cipher is similar to the previous models with a modification that cipher loop is partially unrolled. This architecture is a compromise between earlier looped and loop unrolled architecture so that it tries to optimize both area and throughput. It requires 6 XORs, 12 S-Boxes, a linear transform and 2 multiplexers. Here a plain text can be encrypted in 8 clock cycles. So the proposed architecture is an optimization of area and speed. Fig.5 shows the proposed block cipher architecture. The four rounds of the encryption are unrolled and the three operation in a round like substitution, permutation and linear transformation is executed

in sequence. Since the four rounds are executed in parallel it requires 4 times duplication of hardware but at the same time the number of clock cycle required is only 1 and for the last round it require one more extra. This encrypted block of 16 bit data is passed for next 5 rounds in a looped pipelined manner with pipelining register R1 and R2. The final encrypted data is taken in a 64 bit register at every clock cycle so it require 5 more clock cycle so in total it require 7 clock cycle for getting the encrypted data. The Encryption block (EE) is represented from EE1-EE4 which operates in parallel.

Instead of using the four sbox as such in the design, a hardware friendly sbox is selected from the sbox given in the Table 1 and it is repeated four times to improve the area and performance. From the Table 3, it is found that minimum hardware required for S3 implementation. So it is selected as the hardware friendly sbox's and used in the further implementation. The overall encryption core architecture is shown in fig.6. System is made reset before the first encryption starts. Upon reset, the control signals data_sel, rss_el, key_sel,init_encr and the counters reset to zero. Data_sel is used to select corresponding input data to the block cipher, rs_sel is used to select required internal register during operation, key_sel is used to select corresponding sub keys and init_encr is used to differentiate between initialization and encryption.

At first the internal registers are loaded with the 16-bit random nonce values and core starts encrypting the RS1+RS3 for four iterations. Each iteration takes 7 clock cycles as well as internal state updating and same block cipher architecture is reused each time as shown in the fig.5 which ensures maximum architecture reuse and hence better area and performance. After four iteration, the initialization is completed and the control signal init_encr set to 1. Once the initialization is completed, i th plain text is taken as the input and after 7 clock cycles, we get i th cipher text. During the above procedure, the corresponding sub keys, internal register and input data are selected according to the control signals key_sel, rs_sel and data_sel respectively. The control signals are generated based on two counters. Control signals rs_sel and data_sel should be updated after each block cipher operation so it is controlled by a block counter. The init_encr signal is updated after initialization process so it is controlled by a round counter. After each iteration, the internal states are updated accordingly. The LFSR is initialized after initialization process and updated after each encryption. The proposed architecture is not altering the algorithm; it is an optimized hardware implementation of the same.

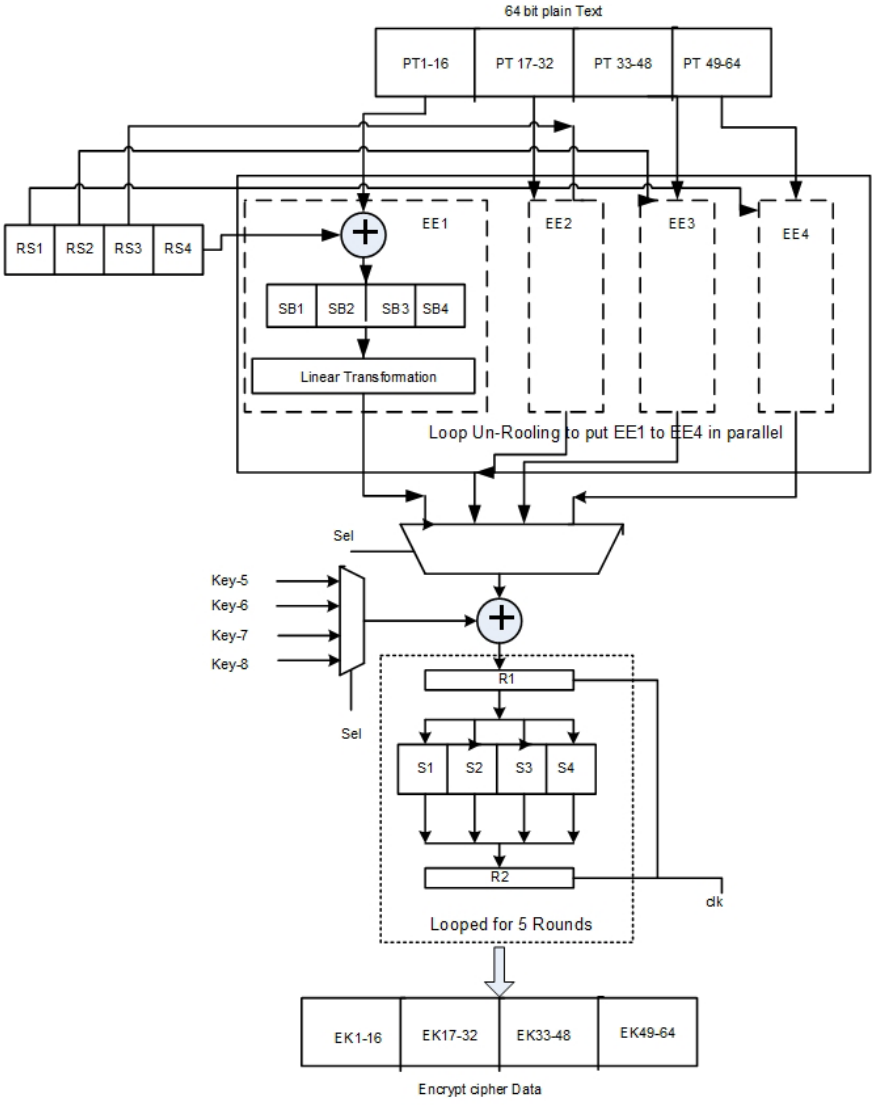


Fig. 5 Proposed block cipher

Table 2 Cipher comparison

	Looped	Loop unrolled	Proposed (partial loop unrolled)
Area	5XOR 8Sbox 1Linear transform 2MUX	8XOR 20Sbox 4linear transform No MUX	6XOR 12SBox 2Linear transform MUX
Clock cycles (encryption)	16	4	7

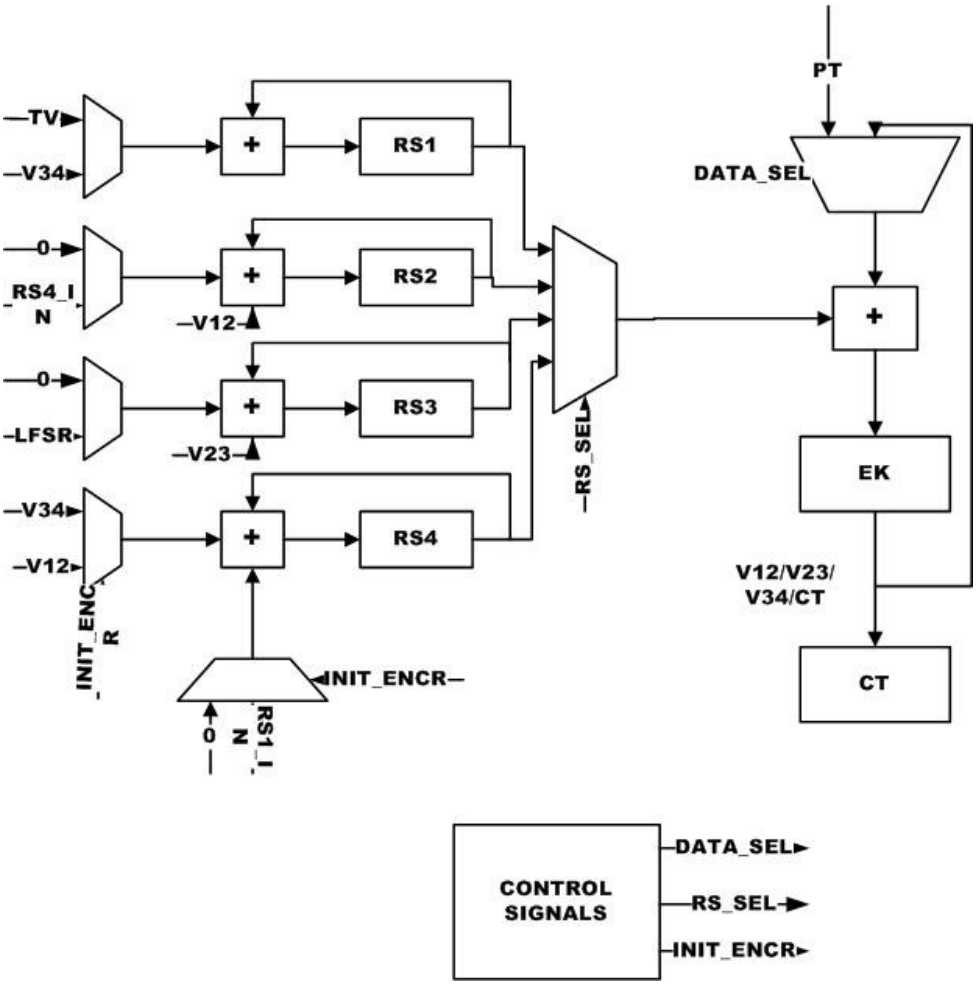


Fig. 6 Overall architecture

Table 3 Area requirement of different sbox implemented on Spartan-3 XCS200 FPGA platform

Sbox	#LUT	#FlipFlop	Total occupied slices
S1	187	16	101
S2	184	16	99
S3	184	16	98
S4	186	16	99

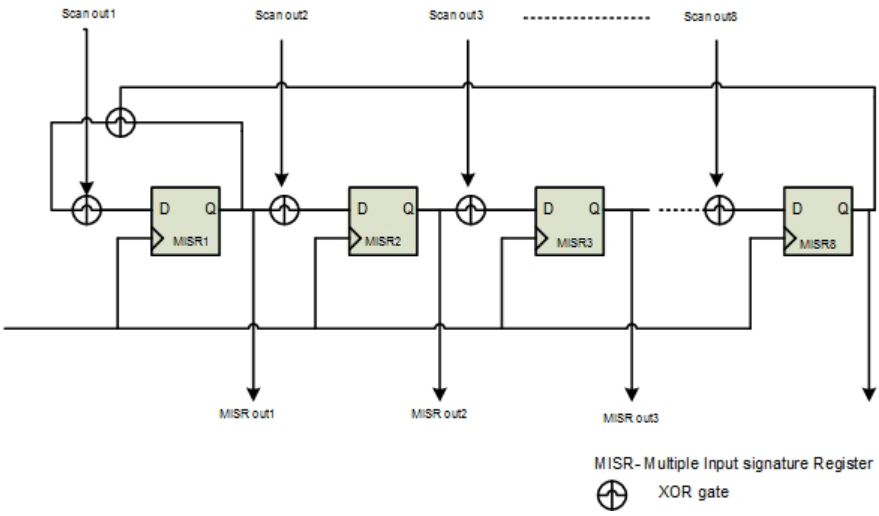


Fig. 7 Proposed architecture against scan attack

3.1 Cryptanalysis of proposed Hummingbird architecture

Cryptographic processors are subjected to side channel attack. If the processor uses the intermediate registers, the probability of easy hacking is more. The Scan chain based Flip-flop (FF) used in the design synthesis is more vulnerable to scan attack. The hackers can easily run the processor for 2 to 3 rounds by operating in NORMAL mode and then switch over to TEST mode which can retrieve all the data stored in the intermediate register made of scan based FF. The hackers can hack the data by running the processor for few clock cycles by applying all possible inputs and outputs vectors in a reversal fashion. Based on the hamming distance between the input pairs the plain text can be hacked. Usage of scan based FF cannot be avoided because it facilitates the easy testability of a ICs and this opens an easy way for the hackers to steal the data.

A response compactor register is used instead of using normal FF to store the intermediate results. The response compactor is a Multiple Input Signature

Register (MISR) which compresses the scan FF intermediate data in a compact register by XOR operation as shown in the Fig.7. This will reduce the area and simultaneously make the architecture resistance to scan attack. We could see this architecture provide two hamming distance (1 and 8) which are at extreme so that we get $40 \approx 2^{5.32}$ possible values for each key byte. Therefore, on average, the final key hypotheses is $(25.32)^{16} = 2^{85.15}$, which cannot be brute-forced easily. Basically this architecture is resistance to attacks like birthday attack, differential attack and linear attack.

3.2 Validation of the proposed architecture

The encryption and decryption block for 16 bit is modeled with Verilog HDL and simulated using Modelsim. The functional verification is shown in Fig.8 and 9. The ASIC implementation is done by synthesizing the design with RTL compiler with 45nm technology file from TSMC. The backend of the design is done using SOC Encounter. The complete chip layout is shown in Fig.10 with design specification. The cryptanalysis on the proposed architecture is performed, since we have used the iterative architecture for compressing the scan chain FF response in a response compactor it shows that it is very difficult for brute-forced attach on our architecture.

4 Results and discussion

The architecture for encryption core as well as decryption core is modeled using Verilog HDL and simulated in modelsim. The architecture is synthesized and implemented using Cadence RTL complier and encounter. For comparison with the previous implementations, the same architecture is also implemented using Xilinx ISE 13.2 by taking low cost FPGA sparten-3 XC3S200 in package FT-256 with a speed grade of - 5.

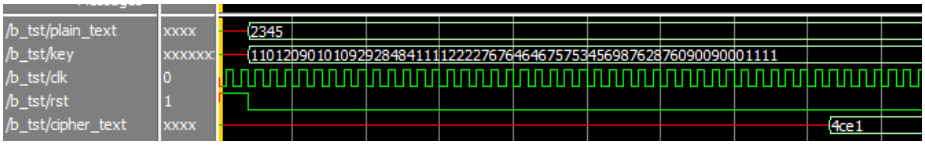


Fig. 8 Simulation result of encryption

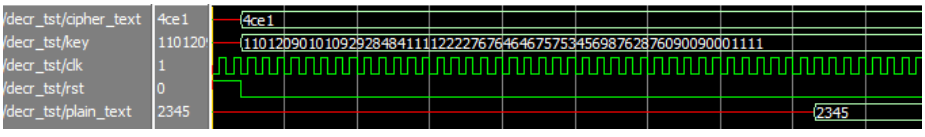


Fig. 9 Simulation result of decryption

Table 4 Cadence synthesis report in 45nm

Item	Area		Power(uW)	Frequency 4900(MHz)
	cell	cell area		
encr_cipher	535	1126	794.09	218.91
decr_cipher	463	1025	735.92	244.61
encryption	1393	3996	845.86	164.17
decryption	1956	5494	1623.29	216.07

Table 5 Performance comparison of FPGA implementations

Design	**Algorithm**	**Key size**	**Block Size**	**FPGA**	**Area (Slices)**	**Memory Blocks**	**Frequency (MHz)**	**Throughput (Mbps)**	**Efficiency (Mbps/ #slice)**
X.Fan et al.[6,8]	Hummingbird (Throughput opt)	256	16	Spartan-3 XC3S200-5	273	0	40.1	160.4	0.59
	Hummingbird (area opt)				253	0	66.1	66.1	0.26
Ismailsan et al.[10]	Hummingbird	256	16	Spartan-3 XC3S200-5	40	2	260.8	55.64	1.38
Biao-min[9]	Hummingbird (Throughput opt)	256	16	Spartan-3 XC3S200-5	230	0	39.8	157.6	0.68
	Hummingbird (area opt)				183	0	72.9	72.9	0.39
Posch-mann et al.[4]	PRESENT	80	64	Spartan-3 XC3S400-5	176	0	258	516	2.93
		128	64		202	0	254	508	2.51
Kaps et al.[11]	STEA	128	64	Spartan-3 XC3S50-5	254	0	62.6	36	0.14
Yalla et al.[12]	HIGHT	128	64	Spartan-3 XC3S50-5	91	0	163.7	65.48	0.72
Mace et al.[13]	SEA	126	126	Virtex-II X-C2V4000	424	0	145	156	0.368
This Work	**Hummingbird**	**256**	**16**	**Spartan-3 XC3S200-5**	**255**	**0**	**69.96**	**140**	**0.54**

Table 4 shows the 45nm synthesis report in Cadence. No comparison is made with this report because it is done for the ASIC implementation of the algorithm. Comparison is done with FPGA implementation of the same with existing implementations. Table 5 shows the comparison of our implementation with other existing implementations as well as other few cryptographic algorithm implementations. X.fan proposed throughput optimized implementation of hummingbird with 273 slices with a throughput of 160.4Mbps,[8] X.fan implemented area op-

timized hummingbird with 253 slices and 66.1Mbps throughput.[6] Comparing with [8] our implementation has better area performance (-6.6%) but throughput is slightly lower. Comparing with [6] our implementation has a good throughput performance (+111%) but area is slightly higher. Strictly speaking, our implementation is a compromise between area optimized design and throughput optimized design.

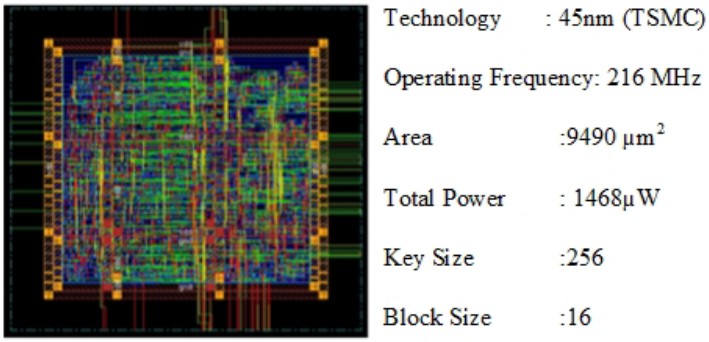


Fig. 10 Simulation result of decryption

This clearly shows that the proposed design is optimized in terms of both area and throughput. Ismail san has very good area performance but it utilizes the embedded memory in the FPGA and it utilizes instruction stored in the embedded memory.[10] So this implementation cannot be assumed as a complete hardware implementation. Biao min throughput oriented design has better performance compared to our design.[9]But the proposed design have better throughput compared to area oriented design of the same [9]. The algorithm PRESENT [4] has better area and throughput performance than hummingbird implementation but it required comparatively larger FPGA XC3S400 which intern increases the cost. xTEA[11] and HIGHT[12] have very less throughput compared to hummingbird implementation and for SEA[13] requirement of area is very high. The Loop unrolling can increase the throughput but simultaneously increase the area so an optimum of around 50% rounds is unrolled to have a compromising solution for area and performance.

5 Conclusions

This paper presented an efficient VLSI architecture for ultra-light weight cryptographic algorithm named hummingbird. Previous implementation of this algorithm was focusing on optimizing either the throughput or area. But this paper presented a design which is optimized in terms of both in area as well as throughput. The proposed design is a compromise between both area optimized and

throughput optimized schemes. The hummingbird algorithm has good efficiency compared to all other algorithms and also it has smallest block size compared to all. The experimental results shows that the hummingbird algorithm is very much useful for resource constrained embedded devices. The initialization of internal registers is done by loading some random nonce values which is highly influences the security of the algorithm. The generation of these random values on the hardware is done using an LFSR.

References

[1] D. Hong, J. Sung, S. Hong, J. Lim, S. Lee, B. S Koo, C. Lee, D. Chang, J. Lee, K. Jeong, H. Kim, J. Kim and S. Chee. (2006), "HIGHT: a new block cipher suitable for Low-Resource device", *Proceedings of CHES 2006*, volume 4249 of LNCS, pages 46-9, Springer.

[2] F. X Standaert, G. Piret, N. Gershenfeld and J.-J. Quisquater. (2006), "SEA: a scalable encryption algorithm for small embedded application", *Smart Card Research and Applications, Proceedings of CARDIS 2006*, volume 3928 of LNCS, pages 222-236, Springer-Verlag.

[3] G.Leander, C.Paar, A. Poschmann and K. Schramm. (2007), "New LightweightDES Variants", *Fast Software Encryption.*

[4] A.Poschmann, A.bogdanov and L.R Knudsen. (2007), PRESENT: An Ultra-Lightweight Block Cipher, springer.

[5] D. Engels, X. Fan, G. Gong, H. Hu and E. M. Smith. (2010), "Hummingbird:Ultra-Lightweight Cryptography for Resource- Constrained Devices", *toappear in the Proceedings of The 14th International Conference on Financial Cryptography and Data Security - FC 2010*, Berlin, Germany:Springer-Verlag.

[6] X.Fan, G. Gong, K.Lauffenburger and T.Hicks. (2010), Design Space Exploration of Hummingbird Implementations on FPGAs, Techincal Report.

[7] Xinxin Fan, Honggang Hu, Guang Gong1, Eric M. Smith and Daniel Engels. (2009), "Lightweight Implementation of Hummingbird Cryptographic Algorithm on 4-Bit Microcontrollers", *Institute of Electrical and Electronics Engineers.*

[8] Xinxin Fan, Guang Gong, Ken Lauffenburger and Troy Hicks. (2010), "FPGA Implementations of the Hummingbird Cryptographic Algorithm", 978-1-4244-7812-5/10/, IEEE.

[9] Biao Min, Ray C.C. Cheung and Yan Han. (2011), "FPGA-based High-Throughput and Area-Efficient Architectures of the Hummingbird Cryptography", 978-1-61284-972-0/11/, IEEE.

[10] Ismail San and Nuray At. (2011), "Compact Hardware Architecture for Hummingbird Cryptographic Algorithm", *21st International Conference on Field Programmable Logic and Applications*, 978-0-7695-4529-5/11, IEEE.

[11] J.-P. Kaps. (2008), "Chai-tea, cryptographic hardware implementations of xTEA", *INDOCRYPT 2008*, LNCS, vol.5365, pp.363-375, Springer.

[12] P. Yalla and J.P. Kaps. (2009), "Lightweight Cryptography for FPGAs", *International Conference on Re-ConfigurableComputing and FPGAs ReConFig'09.*

[13] F. Mace, F.X. Standaert and J.J. Quisquater. (2007), "FPGAimplementation(s) of a Scalable Encryption Algorithm", *IEEETrans*, Very Large Scale Integ, (VLSI) Syst.Vol.16, No.2, pp.212-216.

Corresponding author
Vanitha M can be contacted at: mvanitha@vit.ac.in