Area efficient cryptographic ciphers for resource constrained devices

T. Blesslin Sheeba[1], Dr. P. Rangarajan[2]

[1.] Department of ECE, Sathyabama University, Chennai-600087, India
[2.] Department of EEE, RMD Engineering College, Chennai-600087, India
blesslinsheebarmk@gmail.com

Abstract: The upcoming area of pervasive computing will be characterized by many smart devices that have very limited resources in terms of memory, computing power and battery supply. In information technology, Ubiquitous which is widely believed to be the next paradigm .The mass deployment of pervasive devices promises on the one hand many benefits, but on the other hand, many foreseen applications are security sensitive. In order to provide security on resource constrained devices lightweight cryptographic algorithms have been developed. In this paper we propose lightweight cryptography for FPGAs by introducing block cipher independent optimization techniques for Altera Cyclone III FPGAs and applying them to the lightweight cryptographic algorithms HIGHT and Present. Both are less than half the size of the AES implementation without using block RAMs.
[T. Blesslin Sheeba, P. Rangarajan. **Area efficient cryptographic ciphers for resource constrained devices.** *Life Sci J* 2013;10(3):1107-1114] (ISSN: 1097-8135). http://www.lifesciencesite.com. 161

Keywords: AES, Block cipher, Camellia, FPGAs, Lightweight cryptography.

1. Introduction

Ubiquitous computing represents the third area of computing devices after mainframes and personal computer for first and second eras. Radio frequency identification (RFID) tags and wireless sensor network (WSN) nodes are a few examples which are being used for automated electronic toll systems, identification tags for food products, pets, clothing and so on. This brings us close to the threshold of pervasive computing. The mass deployment of this device brings serious concerns for security and privacy[18]&[19]. The traditional cryptographic algorithms may not be suitable for these devices as they have limited memory and computational power along with serious power constraints. This led to development of new branch of cryptography called lightweight cryptography. HIGHT and Present were developed specifically for lightweight cryptography AES and Camellia, though not considered lightweight, and are also being used on these devices. Until now, lightweight cryptography is targeted towards application specific integrated circuits (ASICs). ASICs involve high non-recurring engineering cost and long time to market where as Field Programmable Gate Arrays (FPGAs) involve low non-recurring engineering cost and less time to market.

The dominant factor favorable to ASICs is their lower power consumption, which is of primary concern for lightweight cryptographic devices and their lower cost in large volumes. With the advent of low-cost and low-power FPGAs, we expect them to become popular for battery powered applications such as WSN nodes. Hence, they are a targeted for lightweight cryptographic applications. Reconfigurability of FPGAs allows the system to be upgraded if ever the need arises which is not possible with ASICs. Furthermore, lightweight crypto implementations lead to area saving over traditional implementations. This enables a designer to add crypto to an existing design at a minimal cost or to reduce the overall area consumption which might lead to cost saving as the design might now fit into a smaller, cheaper FPGAs. The ciphers considered are of full strength security i.e. 128-bit key length, even though traditional lightweight cryptography considers 80-bit key length to be sufficiently secure.

2. Materials and Methods

Advanced Encryption Standard

The Advanced Encryption Standard (AES) specifies FIPS-approved cryptographic algorithm that can be used to protect electronic data. The algorithm AES is a symmetric block cipher that can encrypt (encipher) and decrypt (decipher) information. In cipher text encryption converts data to an un-intelligible form, decrypting the cipher text converts orginal form of data called plaintext. A cryptographic key of 128, 192, and 256 bits to encrypt and decrypt data in blocks of 128 bits is possible in AES algorithm. The specified standard algorithm may be implemented in software, firmware or hardware. The specific implementation may depend on several factors such as the application, the environment and technology.

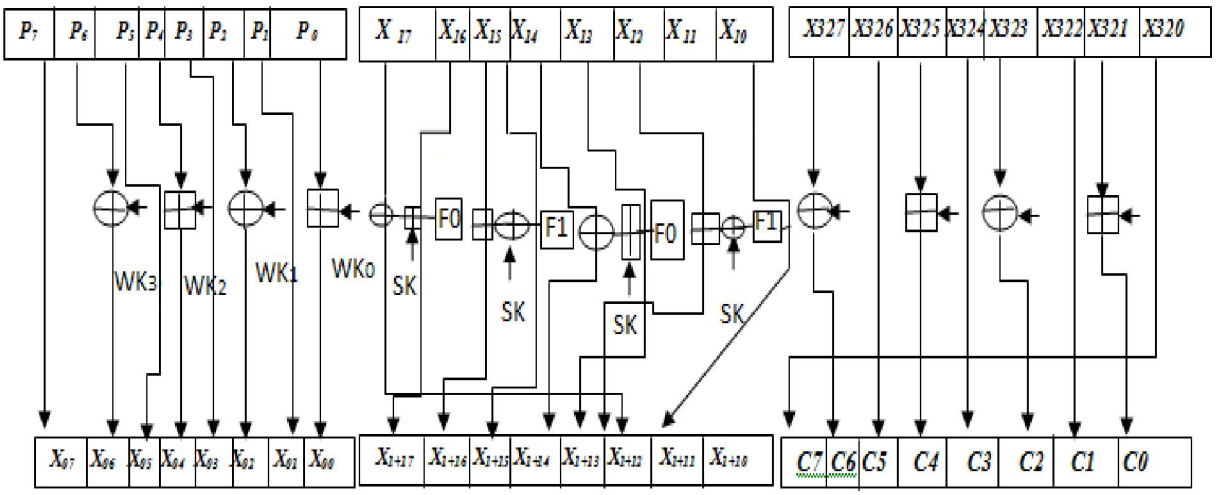

Figure 1.HIGHT

The algorithm shall be used in conjunction with a FIPS approved or NIST recommended mode of operation. Object Identifiers (OIDs) and any associated parameters for AES used in these modes are available at the Computer Security Objects Register (CSOR).

HIGHT

HIGHT (HIGH security and light weight) is a 64-bit block cipher with 128-bit key length. It uses generalized Feistel structure with 32-rounds containing simple operations such as XOR, modular addition in the group of 28 elements, and bitwise rotation. The absence of traditional substitution layer, its Feistel structure and byte oriented operations make it suitable for low-cost, low-power and lightweight implementations. The HIGHT algorithm was modifieded to reduce the critical path in the key schedules which also reduces the area to 2608 gates. The initial security analysis presented in [2] & [4] showed that HIGHT only 19 rounds is secure. It provides low-resource hard-ware implementation, which is proper to ubiquitous computing device such as a sensor in USW or a RFID. HIGHT does not only consist of simple operations to be ultra-light but also has enough security as a good encryption algorithm.

Table 1. Comparision of AES and HIGHT

Algorithm	Technoloy (µm)	Area (GEs)	Throughput (Mbps)	Max frequency(MHz)
AES	0.35	3400	9.9	80
HIGHT	0.25	3048	150.6	80

Present

Present is an ultra-lightweight block cipher an it is a 31-round Substitution-Permutation (SP) network with a block size of 64-bit and 80-bit or 128-bit key lengths. In this paper a 128-bit key length is considered. Present was designed by incorporating some features of Serpent and Data Encryption Standard (DES). The nonlinear substitution layer, i.e. S-box in Present is similar to that of Serpent and the linear permutation layer to that of DES. The original Present proposal provides a basic security analysis. With the establishment of the AES the need for new block ciphers has been greatly diminished; for almost all block cipher applications the AES is an excellent and preferred choice. However, despite recent implementation advances, the AES is not suitable for extremely constrained environments such as RFID tags and sensor networks [3]. In this paper ultra-lightweight block cipher is described. During design of cipher , both security and hardware efficiency have been equally important [5].

Block ciphers

The entire block of plaintext can be encrypted with the same key [14]. This means that the encryption of any plaintext bit in a given block depends on every other plaintext bit in the same block[15]. In practice, the vast majority of block ciphers either have a block length of 128 bits (16 bytes) such as the advanced encryption standard (AES), or a block length of 64 bits (8 bytes) such as the Data Encrypted Standard (DES) or triple DES (3DES) algorithm.

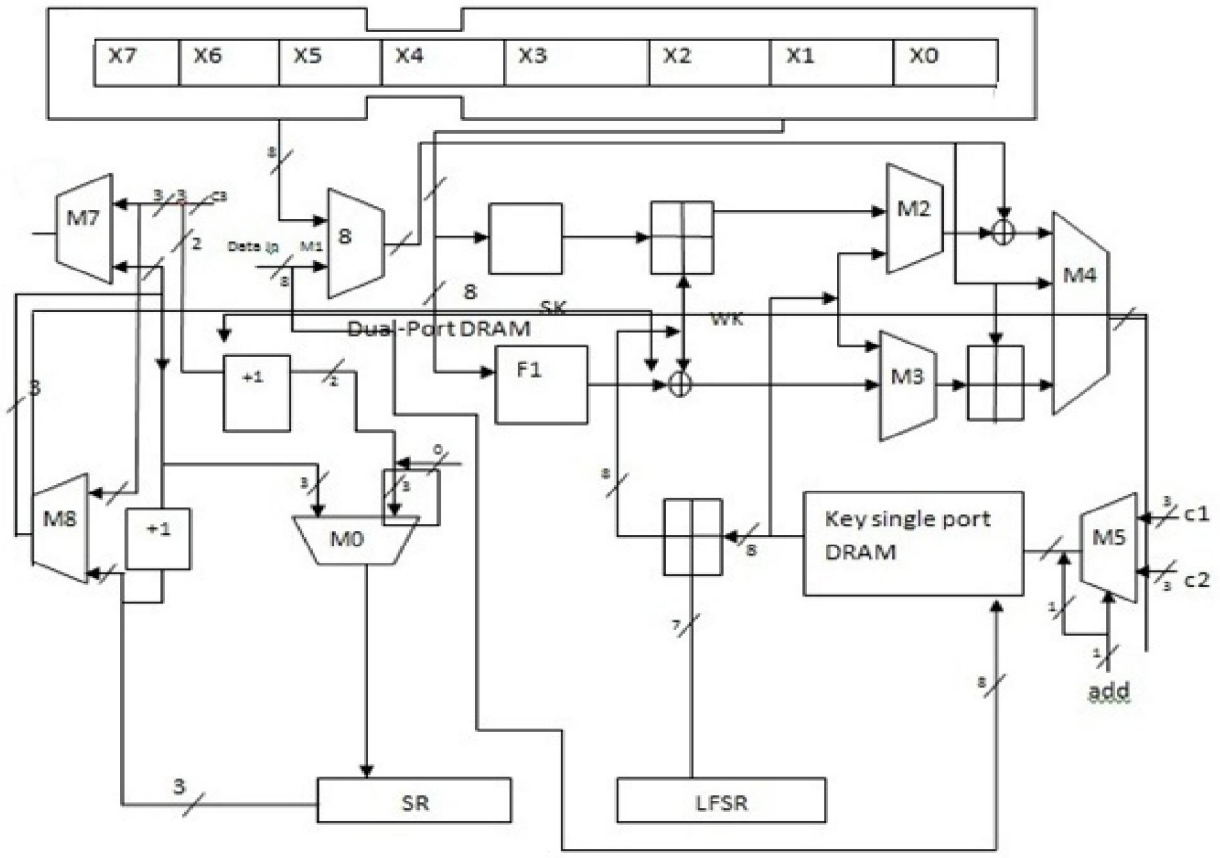

Figure 2.Top level block diagram of HIGHT

2.1. Optimization techniques

Designing compact architectures in FPGAs depends on effective use of architectural features provided in the targeted FPGAs. FPGAs have features such as Look Up Table (LUT) based 16-bit shift register (SRL16) and Distributed Random Access Memory (DRAM) which can be employed to improve the performance and decrease the area of a given design by an order of magnitude.

2.2. FPGA Architecture

FPGA Structure

The fundamental logic unit in FPGAs is a slice, which contains mainly two four input LUTs and two flipflops. Half the slices of a chip are called SLICEMs. Their LUTs can be configured as 16 bit shift registers (SRL16) or as 16-bit distributed RAMs

Called DRAMs.

Shift Register

The number of slices required for implementing a shift register depends on the number of bits to be stored and the number of taps. Taps are positions of a shift register where data can be written to or read from. Each tap is configured as a flipflop.

Distributed RAM (DRAM)

DRAMs offer fast and localized memory. They can be cascaded for realizing deeper memories with minimal penalty on timing. Distributed RAM supports two types of memories: single-port RAM and dual-port RAM. Both have synchronous write and either synchronous or asynchronous read.

2.3. Plain text and key storage

The most area consuming components of cryptographic algorithms are data and key storage. DRAM and shift register are two options for efficient memory implementation. In order to develop lightweight architectures, the algorithm implementations are scaled down to use either an 8-bit or a 16-bit datapath. Key and data are loaded either 8-bits or 16-bits depending on the implementation. Loading into shift register is simpler as the number of control bits needed is less as compared to DRAM which needs addressing. The size of the address increases with the number of

words to be stored in DRAM. On the other hand, the control bits of shift register are independent of size of the data it stores. However, some ciphers require intermediate data values. In case of HIGHT, plaintext is stored in DRAM due to its generalized Feistel structure. The key scheduling in ciphers Camellia, HIGHT and Present involve shifting of key which make use of shift register more apt as in[4]&[5]. However, the key in HIGHT is need in a different order during initial and final transformations compared to round operations which are difficult to accomplish with a shift register. In this case, a DRAM is used to store the key.

2.4. Control logic

Finite State Machines (FSM) are used for realizing the control logic of complex systems. Traditionally, FSMs are implemented using flipflops and combinational logic. However, this type of FSM implementation is complex and not efficient. The use of RAM blocks for sequential logic led to ROM-based FSM implementations. The control signals for each operation are combined to one control word. These control words are stored in a memory location which can be accessed by an address. DROMs are inferred by holding the write signal low for DRAMs. ROM- based FSMs have additional advantages. The maximum frequency at which a ROM-based FSM operate is independent of the complexity of the circuit. This method is also proved to save power. For our HIGHT and Present architectures, the control signals are generated by a counter and some additional logic. The size of the control word is reduced by removing any control signals which repeat a short sequence of values many times, such as control signals for round operations, from the main controller and assigning them to a sub-controller. We use this technique for camellia where the control signals for round function f are generated by a sub-controller called F-controller.

3. Discussion

The HIGHT algorithm consists of 32-rounds with initial and final transformations before the first and after the last rounds respectively. The plaintext P and cipher text C are split into eight 8-bit blocks P7…P0 and C7…C0 and the original key K into sixteen 8-bit blocks K15….K0.The initial transformation uses the four whitening key bytes W K0, W K1 , W K2 , and W K3 to transform a plain text P into the input of first round function, X0 = X0,7 X0,0. In the final transformation, the data is shifted towards right and transforms X32 = X32,7 X32,0 into cipher text C by with the four whitening keys W K4 , W K5 , W K6 and W K7. Both transformations perform a XOR or modular addition as shown in Figure 1.

3.1 Lightweight architecture of HIGHT

The reduction of area for HIGHT is achieved by scaling the 64-bit algorithm to 8-bit. This reduction does not come at the cost of temporary storage or multiplexers.

Data Storage

The round function involves shifting which suggest that a shift register is the most efficient solution for data storage. However, the generalized Feistel structure of HIGHT leads to a misalignment of data when a shift register is used. Realignment requires additional clock cycles. This reduces the throughput and the more complex logic increases the area consumption. Therefore, we use a DRAM which is more efficient both in terms of area and latency. A dual-port DRAM is used as the round function requires two 8-bit blocks of data for computing one 8-bit block. The shifting involved in round function is accomplished by addressing. The addresses needed for all operations are generated by using three 3-bit multiplexers M6, M7, and M8, 2-bit and 3-bit adders, 12-bit shift register SR and a 3-bit counter C3as shown in figure 3. The address from M8 is used for both reading and writing while M7 is used only for reading. The control signals for all operations are generated by using a 5-bit counter and some logic functions.

Round Function,Initial and Final Transformation

The addresses for round function are generated by shift register (SR) and a 3-bit adder. The initial values of SR required for first round are computed by using 2 LSB bits of C3. The addresses for the next round are generated by loading the result of 3-bit adder into SR. This way of generating the addresses reduces the complexity of the control logic and avoids extra clock cycles needed for shifting of data. Each round operation requires 4 clock cycles. The initial and the final transformations are performed by using the data path of the round function with use of two extra multiplexers M2 and M3. The addresses for both transformations are generated by C3. The initial transformation is performed while the data is being loaded into the shift register which save clock cycles.

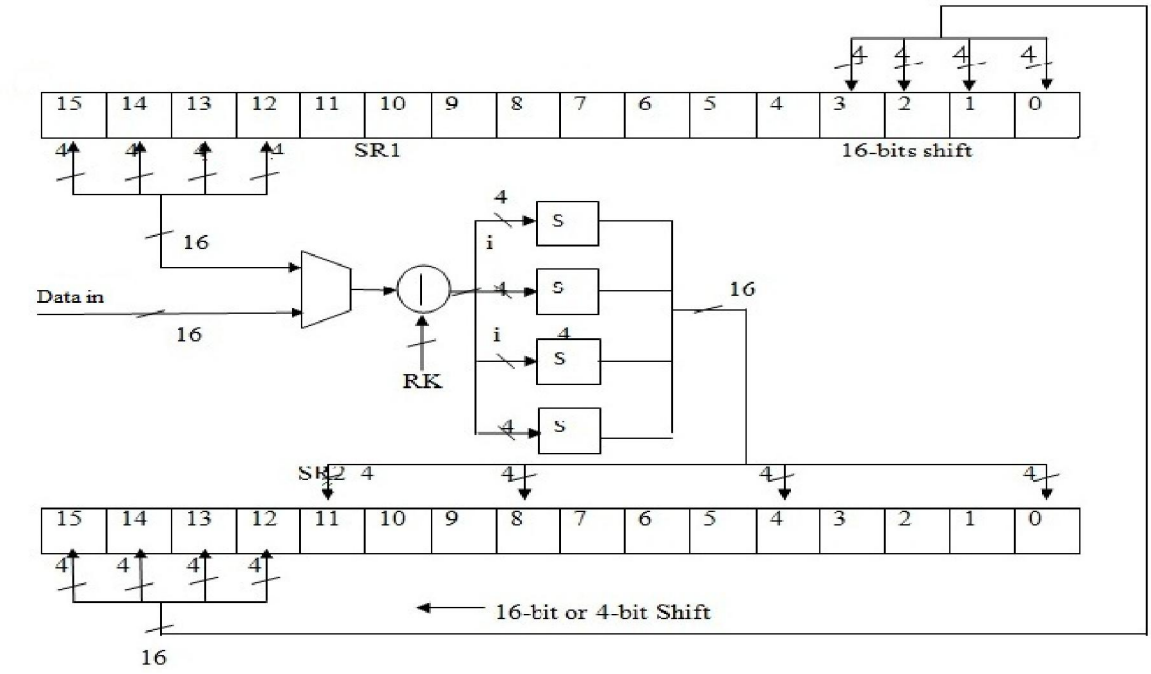

Figure 3. 16-bit datapath of Present

Key Storage and Scheduling

The 128-bit key is stored in a single-port DRAM. The sub-keys and whitening keys are generated by using two 3-bit counters C1 and C2 and a 2x1 multiplexer M5. The MSB bit of the key address which is also used as selection bit for M5 is generated from the output of the 5-bit counter used in control logic. The two 3-bit counters are used for addressing instead of one 4-bit counter to accomplish shifting involved in key scheduling.

3.2. Lightweight architecture of PRESENT

The area is reduced by scaling the 64-bit implementation to a 16-bit implementation and by applying the optimization techniques. Scaling the implementation to 8-bit would decrease the throughput drastically and yield a very small area reduction due to the complexity of the permutation operation. Our implementations of wider data path led to a significant increase in area consumption.

Data storage

The state b63 _ _ _ b0 is stored in the shift register SR1 for the reason specified in III. It performs a 16-bit circular left-shift per clock cycle. We consider the 64-bit shift register as a combination of sixteen 4-bit block 15,_ _ _ ,0. The 16 MSB are tapped out of SR1 for the round operation.

S-box Implementation and Permutation Layer

The round operation starts by XOR the incoming data with the round key RKi and applying the result to four S-boxes. Present's 4-bit to 4-bit S-boxes are implemented in a single LUT each. Our architecture uses four S-boxes for round operations and two for key scheduling. The Permutation function is implemented by using the shift register SR2 which performs a shift by 4 bits during round operation and by 16 bits after each round when copying its content into SR1. Furthermore, the 16-bit output from the S-boxes is given as input to the blocks 12, 8, 4, and 0 of SR2. During first clock cycle of the round operation the 4-bits blocks 15 11, 7 and 3 are computed from the 16 MSB of SR1 and placed in position 12, 8, 4, and 0 of SR2. In the subsequent clock cycle SR2 is shifted by 4 bits and blocks 14, 10, 6 and 2 are computed and placed in the now empty positions 12, 8, 4, and 0 of SR2. These operations repeat for another two clock cycles to complete the round function. This results in a total of 8 clock cycles for each round operation.

Key Storage and Scheduling

The key is stored in a 128-bit shift register which performs a 16-bit circular left shift. The first round key RK1 is obtained during the first four clock cycles by tapping the 16 MSB from the key and passing them to the RKGen function. However, during these four clock cycles the key was shifted by 64-bit. Subsequent round keys require only a shift by 61 bits which is not possible with a 16-bit shift. We overcome this problem by placing three extra taps on the shift register and using two 3-bit registers A and

B along with several multiplexers . The 3 bits that were "lost" during generation of the first round key are stored in register A. For subsequent round keys, the value from register A and the 13 MSB from the key are passed to RKGen. However, when generating the round key RK2 we are facing the same problem of shifting by 64-bit again and compensate by storing three bits in register B. From now on, RKgen gets the value from registers BjjAjj11 MSB from key. When we write the round keys back into the shift register we take the 3-bit difference into account. Therefore, no additional registers are required for subsequent rounds. The RKgen function consists of two S-boxes for S-box operation, a 5-bit XOR to compute the XOR with the round counter, and multiplexers to choose the appropriate bits for round key generation. The output of the RKgen function is the round key as shown in figure 4.

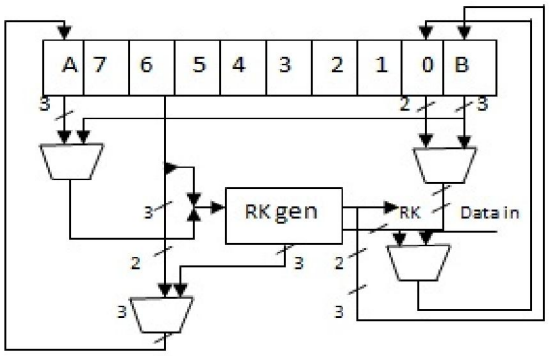

Figure 4. Key scheduling of present

4. Results

The designs of HIGHT and Present were described in Verilog HDL, synthesized for the Altera Cyclone III EP3C16F484C6 device using Altera Quartus II and Simulated with Modelsim. All results are after place and route. Table I shows the detailed implementations of HIGHT and Present. The results for AES were obtained by using the VerilogHDL code for the ASIC implementation and synthesizing it in FPGA. Camellia and AES encrypt blocks of 128-bit data, whereas the other algorithms operate on 64-bit data blocks. Therefore, AES and Camellia implementations require more storage. Furthermore, AES and Camellia have 8x8 S-boxes which occupy 64 slices or 16% and 20% respectively of the total design area in our implementations. Present's 4x4 Sbox occupies only 2 slices Present AES use registers (i.e. flipflops) for data and key storage. Even though the total number of flipflops needed is far smaller than the number of LUTs used, the addressing logic contributes to the area consumption. The Camellia implementation uses 88 SRL-16 elements, which would be capable of storing a maximum of 1,408 bits, to store its two 128-bit keys. Unfortunately, the round key generation shifts the key by 15 and 17 bits. This irregular shift requires many additional tapings causing the high number of SRL-16 elements. Implementing shifts in multiples of 8 require less area.

Table 2. Lightweight implementation results for Altera Cyclone III

Block cipher	Flip Flops	LUTs		SRL-16s	Slices	Implementation Cipher for Data	Key
Present HIGHT	114 25	1595 1332	0 24	16 3	117 91	SRL DRAM	SRL DRAM
Camellia Tiny XTEA-3 AES	164 226 338	4200 4245 531	160 000	88 0 0	318 254 393	DRAM FF FF	SRL FF FF

The same can be observed in Present's key schedule as its involves 61-bit shifts. HIGHT makes extensive use of DRAM elements for both, data and key storage and uses SRL-16s in its control logic. Present uses SRL-16s for both. DRAM and SRL-1 elements are an ideal choice for storing data and key provided that the algorithm is regular which leads to a simple control logic. Camellia is an example for an algorithm with high irregularity, therefore DRAM and SRL-16 elements cannot be used to full effect. Implementing permutation functions that span more than 8 or 16 bits also increases the area consumption and latency for lightweight implementations in FPGAs. Table 3.compares our implementations with Camellia, TinyXTEA-3, AES and the eSTREAM portfolio ciphers. The stream ciphers outperform all block cipher implementations with respect to the throughput/area metric. However, they are defined for 80-bit keys and only MICKEY and Grain offer 128-bit versions. Stream ciphers are still considered immature and only recently the stream cipher F-FCSR-H was removed from the portfolio. AES has the highest throughput of the block ciphers followed by HIGHT. However, HIGHT has a better throughput to area ratio and consumes only half the size and no block rams. For the throughput to area ratio computation 300 slices are added to the area of AES and 140 to AES to compensate for the block ram usage.

Table 3. Results for Present and HIGHT compared to other block Ciphers and the eSTREAM for TFOLIO Ciphers on FPGA

Design	Max. Delay	Clock Cycles per block	Block Size (bits)	Key Size (bits)	Area (Slices)	Block RAMs	Throughput (Mbps) at (fmax)	Throughput area (Mbps/ Slices)	Device
Present	8.78	256	64	128	117	0	28.46	0.24	XC3S50-5
HIGHT	6.12	160	64	128	91	0	65.48	0.72	XC3S50-5
Camellia	7.95	875	128	128	318	0	18.41	0.06	XC3S50-5
AES	14.21	534	128	128	393	0	16.86	0.04	XC3S50-5
AES 8-bit	14.93	3900	128	128	124	2	2.2	0.01	XC2S15-6
AES Tiny	20.00	46	128	128	222	3	139	0.27	XC2S30-5
XTEA-3-	15.97	112	64	128	254	0	35.78	0.14	XC3S50-5
Grain v1	5.10	1	1	80	44	0	196	4.45	XC3S50-5
Grain 128	5.10	1	1	128	50	0	196	3.92	XC3S50-5
MICKEY v2	4.29	1	1	80	115	0	233	2.03	XC3S50-5
MICKEY 128	4.48	1	1	128	176	0	223	1.27	XC3S50-5
Trivium	4.17	1	1	80	50	0	240	4.80	XC3S50-5
Trivium (x64)	4.74	1	64	80	344	0	13.504	39.26	XC3S400-5

Corresponding Author:
T.Blesslin Sheeba
Department of ECE, Sathyabama University,
Chennai-600087, India.
Email: blesslinsheebarmk@gmail.com

5. Conclusion

Lightweight implementations of cryptographic algorithms for FPGAs are going to become an important research area due to the introduction of FPGAs for battery powered devices. In this paper we introduced the first lightweight implementations of the block ciphers HIGHT and Present on FPGAs. Our implementation of HIGHT consumes less than 100 slices, encrypts data at 65 Mbps and has a better throughput over area ratio than the previously published lightweight implementation of AES. Furthermore, we introduced optimization techniques for lightweight implementations that can also be applied to other algorithms. Investigating the robustness of lightweight implementations against side channel analysis and implementing lightweight asymmetric cryptosystems is future work.

References

[1] J.-P. Kaps, G. Gaubatz, and B. Sunar, "Cryptography on a speck of dust," *Computer*, vol. 40, no. 2, Feb 2007, pp. 38– 44.
[2] D. Hong *et al.*, "HIGHT: A new block cipher suitable for low-resource device," in *CHES 2006*, ser. LNCS, L.Goubin and M. Matsui, Eds., vol. 4249. Springer, 2006, pp. 46–59.
[3] A. Bogdanov *et al.*, "PRESENT: An ultra-lightweight block cipher," in *CHES 2007*, ser. LNCS, vol. 4727. Springer,2007, pp. 450–466.
[4] L. Young-Il, L. Je-Hoon, Y. Younggap, and C. Kyoung-Rok, "Implementation of HIGHT cryptic circuit for RFID tag,"*IEICE Electronics Express*, vol. 6, no. 4, 2009,pp. 180–186.
[5] O. Ozen *et al.*, "Lightweight block cipher revisisted: Cryptanalysis of reduced round PRESENT and HIGHT," in *ACISP*, ser. LNCS, vol. 5594. Springer, 2009, pp. 90–107.
[6] M. Rawski, H. Selvaraj, and T. Luba, "An application of functional decomposition in ROM-based FSM implementation in FPGA devices," *J. Syst. Archit.*, vol. 51, no. 6-7, 2005, pp. 424–434.
[7] E. Biham, R. Anderson, and L. Knudsen, "Serpent: A new block cipher proposal," in *FSE*

1998, ser. LNCS, vol. 1372. Springer, January 1998, pp. 222–223.

[8] B. Collard and F.-X. Standaert, "A statistical saturation attack against the block cipher PRESENT," in *CT-RSA*, ser. LNCS, vol. 5473. Springer, 2009, pp. 195–210.

[9] A. Antipa, D. Brown, R. Gallant, R. Lambert, R. Struik, and S. Vanstone, \Accelerated Veri_caiton of ECDSA Signatures", Selected Areas in Cryptography – SAC 2005, LNCS 3897, B. Preneel, S. Tavares (eds.), Berlin, Germany: Springer-Verlag, 2006, pp. 307-318.

[10] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe, \PRESENT: An Ultra-Lightweight Block Cipher", The 9th International Workshop on Cryptographic Hardware and Embedded Systems - CHES 2007, LNCS 4727, P. Paillier, I. Verbauwhede (eds.), Berlin, Germany: Springer-Verlag, 2007, pp. 450-466.

[11] T. G uneysu, T. Kasper, M. Novotn_y, C. Paar and A. Rupp. Cryptanalysis with COPACOBANA. In IEEE Transactions on Computers, 2008, vol. 57, no. 11, pages 1498-1513.

[12] H. Kim, J. Kim, and S. Chee. HIGHT: A new block cipher suitable for low-resource device. Cryptographic Hardware and Embedded Systems (CHES), volume LNCS 4249, Springer, 2006, pp. 46–59.

[13] G. Leander, C. Paar, A. Poschmann, and K. Schramm.New lightweight DES variants.In A. Biryukov, editor, Fast Software Encryption 2007 (FSE), volume LNCS 4593, Springer, 2007, pp. 196–210.

[14] Lars R. Knudsen, Gregor Leander, Axel Poschmann, and MatthewJ. B. Robshaw. PRINTcipher: A block cipher for IC-printing. CHES,LNCS 6225, 2010, pp. 16–32.

[15] Chae Hoon Lim and Tymur Korkishko. mCrypton - a lightweight block cipher for security of low-cost RFID tags and sensors. WISA2005 , volume 3786 of LNCS, Springer, 2005, pp. 243–258.

[16] K. Shibutani, T. Isobe, H. Hiwatari, A. Mitsuda, T. Akishita, and T. Shirai. Piccolo: An Ultra-Lightweight Blockcipher. In CHES, LNCS 6917, 2011, pp. 342–357.

[17] Matthias Krause and Matthias Hamann. The Cryptographic Power of Random Selection. ECRYPT Workshop on Lightweight Cryptography. 2011, pp. 122–146.

[18] Ari Juels. Yoking-Proofs for RFID Tags. In the Proceedings of First International Workshop on Pervasive Computing and Communication Security, IEEE Press, 2004, pp.138–143.

[19] Ari Juels, Yoking-Proofs for RFID Tags, In the Proceedings of First International Workshop on Pervasive Computing and Communication Security, IEEE Press, (2004).,pp.138–143

[20] Junichiro Saitoh and Kouichi Sakurai, Grouping Proofs for RFID Tags, In the Proceedings of AINA International Conference, IEEE Computer Society, (2005), pp. 621–624.

[21] Selwyn Piramuthu, On Existence Proofs for Multiple RFID Tags, In the Proceedings of ACS/IEEE International Conference on Pervasive Services, IEEE Computer Society, (2006),pp. 317–320.

6/25/2013