# A High-performance and Area-efficient VLSI Architecture for the PRESENT Lightweight Cipher

Jai Gopal Pandey, Tarun Goel*, Abhijit Karmakar
CSIR - Central Electronics Engineering Research Institute (CEERI)
* Academy of Scientific & Innovative Research (AcSIR), CSIR-CEERI Campus
Pilani-333031, India
jai@ceeri.res.in

*Abstract*— **Security and privacy are the prime concern in the emerging internet of things (IoT) and cyber physical systems (CPS) based applications. Lightweight cryptography plays an essential role for securing the data in this emerging pervasive computing environments. In this paper, we propose a high-performance and area-efficient VLSI architecture with 64-bit datapath for the PRESENT block cipher. The proposed architecture performs an integrated encryption/decryption operation for both 80-bit and 128-bit key lengths. The architecture is synthesized for the Virtex-5 XC5VLX110T FPGA device, available on the Xilinx ML-505 platform. It has been observed that the proposed architecture utilizes 0.73% and 0.87% of FPGA slices for 80-bit and 128-bit key lengths respectively. A throughput of 410 Mbps and power consumption is about 16 mW for both the key lengths.**

***Keywords— Lightweight cryptography; PRESENT block cipher; Integrated encryption/decryption; VLSI architecture; FPGAs.***

## I. Introduction

The rapidly-growing area of internet of things (IoT) and cyber physical systems (CPS) is based on an ecosystem which eventually relies on billions of tiny interconnected computing-devices [1], [2]. The ability of selective computing, sensing, control and communication, makes these ever-ready devices effective, efficient and intelligent. Some of the day-to-day applications around these devices include car-locks, e-cash cards, electronic gadgets, digital lockers, secure communication, and many more. These tiny devices and their network create a wide-spread pervasive-computing infrastructure in emerging applications. Ever-increasing applications of these devices create an extensive demand of smart computing system and their energy-efficient field deployment. Besides design goals, security and privacy are the prime aspects of this IoT-based CPS infrastructure. Here, the field of cryptography and related ciphers provide a mechanism by which data can be efficiently secured. To secure the transmitted data through any electronic system, a variety of ciphers are used for years. The deployment trend of ciphers in electronic systems is shown in Fig. 1.

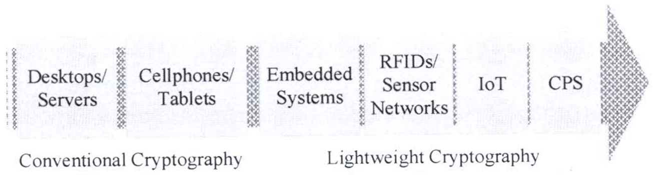

Fig. 1. Deployment trend of ciphers in electronic systems.

This research work has been carried out under the *SMDP-C2SD* project, sponsored by *MeitY, Govt. of India.*

As shown in the Fig. 1, majority of the conventional cryptographic algorithms has been developed around desktop/server centric environments. Therefore, many of these cryptographic algorithms are generally unsuitable for implementation in constrained devices which are used in the modern-age applications. In many conventional cryptographic standards, the trade-offs between security, performance and resource requirements were optimized for desktop and server environments. This makes the implementation of conventional ciphers difficult in resource-constrained applications, and their performance may not be acceptable. The shift from desktop based applications to small-devices centric applications bring a wide range of security and privacy concerns. Lightweight cryptography provides solution tailored for resource-constrained devices and their efficient VLSI implementations [3].

Recently national institute of standards and technology (NIST) provided a report containing an overview of lightweight cryptography and an outline of NIST's plan for standardizing the lightweight cryptographic algorithms [4]. Further, a detailed taxonomy of the lightweight block ciphers can be found in [3] and [5]. Systematic survey of lightweight-cryptography ciphers and their software and hardware implementations with detailed description and related discussions can be found in [3], [5] and [6]. Here, it has been emphasized that efficient implementation of the ciphers are closely dependent on the selection of appropriate architecture, as they result in low implementation complexity and high-performance in actual realizations. To propose a new architecture for the lightweight cryptography, there is always trade-offs between the three prime objectives i.e. security, cost and performance, which is shown in Fig. 2 [7].

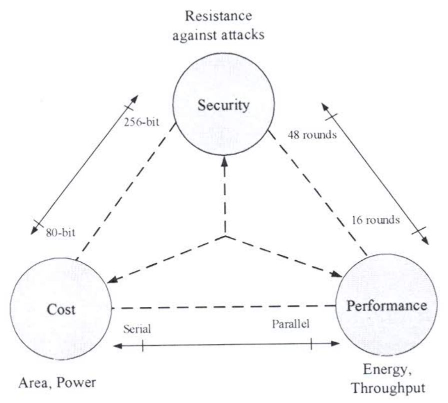

Fig. 2. Architectural trade-offs between security, cost and performance. Adapted from [7].

In the context of lightweight cryptography, ISO/IEC 29192-2 has standardized a symmetric block cipher algorithm in the year 2012, which is known as the PRESENT cipher [8]. The algorithm provides an adequate level of data security alongwith hardware-oriented performance attributes, which makes it a prominent choice for developing most of the secure and lightweight applications [5] and [9].

In this paper we propose a high-performance and area-efficient VLSI architecture for the PRESENT block cipher that completely integrates both encryption and decryption engines. The architecture has been implemented in the Xilinx Virtex-5 XC5VLX110T FPGA device [10]. The experimental results of the implementation show that the proposed architecture consumes a number of 126 slices for the 80-bit key and 150 slices for the 128-bit key lengths. The architecture is capable of running at a clock frequency of more than 210 MHz. The dynamic power dissipation is about 16 mW when the architecture has been operated with around two thousand random input vectors.

Rest of this paper is organized as follows: In Section II, an overview of the PRESENT algorithm is given. Section III is used to discuss the existing implementations of the PRESENT block cipher. An integrated architecture for the PRESENT block cipher alongwith a detailed description of various basic building blocks are proposed in Section IV. Section V is used to provide experimental results alongwith a comparison with an existing architecture from literature. Finally, conclusions are drawn in Section VI.

## II. THE PRESENT ALGORITHM

The PRESENT algorithm works on a block size of 64-bit and supports two key length variants of 80-bit and 128-bit. The principle of PRESENT cipher is based on the concept of substitution and permutation network (SPN). There are total 31 rounds and each round consists of an XOR operation, which is required to introduce a round key $K_i$, for $0 \le i \le 31$, where $K_{31}$ is used for post-whitening operation [9]. A non-linear substitution layer operation is performed in each round and this layer consists of a 4-bit S-box which is applied 16-times in parallel. In addition to that there is a linear bitwise permutation layer. These operations are described below:

### A. The Key Schedule

The cipher requires a unique round key ($K_i$) in each round, where the input key is stored in a key register $K(k_{79}k_{78}...k_0)$ for 80-bit key or $K(k_{127}k_{126}...k_0)$ for 128-bit key length.

### B. Add Roundkey Operation (AddRoundKey)

With current state $b_{63}...b_0$ and for the given leftmost 64-bit of the round key $K_i = k^i_{63} \dots k^i_0$ (or $K_i = k^i_{127} \dots k^i_{64}$); $0 \le i \le 31$, the *AddRoundKey* operation is defined as $b_j = b_j \oplus k^i_j$ for $0 \le j \le 63$ [9].

### C. The S-box and Inverse S-box

The PRESENT algorithm requires a 4-bit-to-4-bit S-box (S) as $\mathbb{F}_2^4 \to \mathbb{F}_2^4$ [9]. The 64-bit current state $b_{63}...b_0$ is taken as sixteen 4-bit words $w_{15}...w_0$, where $w_i = b_{4\times i+3} \,||\, b_{4\times i+2} \,||\, b_{4\times i+1} \,||\, b_{4\times i}$ for $0 \le i \le 15$. The output $S[w_i]$ provides the updated sta values as per [9]. The S-boxes are used in each of the rounds and in the key scheduling operation. For inverse S-box, the S-box does not satisfy $S(S(x)) = x, x \in \mathbb{F}_2^4$; thus, same S-box cannot be used in encryption and decryption both. For the PRESENT cipher, a relation between the S-box and inverse S-box is given by expression, $S^{-1}(S(x)) = x, x \in \mathbb{F}_2^4$.

### D. The Bit Permutation (p-layer) and Inverse Bit Permutation (inv-p-layer) Operations

The bit permutation layer is used to move bit $i$ of the state to bit position $P(i)$ [9] and it is given by the following expression,

$$P(i) = \begin{Bmatrix} i.16 \bmod 63, \ i \ \hat{I} \ \{0,...,62\} \\ 63, \ i = 63 \end{Bmatrix} \qquad (1)$$

Similarly, the inverse bit permutation operation is defined by the below equation,

$$P^{-1} \begin{Bmatrix} i.16 \bmod 63, \ i \in \{0,...,62\} \\ 63, \ i = 63 \end{Bmatrix} = i \qquad (2)$$

In the following section, some of the work related to the implementations of the PRESENT cipher is provided.

## III. RELATED WORK

An architectural design space exploration for encryption and decryption operations can be found in [11]. In this Spartan-III FPGA device based implementation, the encryption and decryption operations require a total of 373 slices and 423 slices for the 80-bit and 128-bit key lengths respectively [11]. In another implementation, a cryptographic co-processor with encryption and decryption capabilities has been provided by [12]. In this paper the architectural exploration for serial, iterative and parallel variants of the cipher has been provided.

For encryption operation, an FPGA-based implementation of the PRESENT cipher that uses 117 slices of the Xilinx Spartan-3 XC3S50 FPGA device has been reported in [13]. Here, a throughput of 28.46 Mbps at a maximum frequency of 114 MHz has been obtained. Two different RAM-based implementations of PRESENT cipher have been provided in [14]. In the first implementation, the substitution box of the cipher has been realized into the FPGA slices. In the second implementation it has been integrated into the RAM. In these implementations, the first design occupies 83 slices and the second design consumes 85 slices of the Xilinx Spartan XC3S50 device. These realizations produce a throughput of 6.03 Kbps and 5.13 Kbps at 100 KHz system clock respectively.

Related to the encryption-only operation, one such implementation of the PRESENT cipher with 8-bit datapath has been given in [15]. This design utilizes 62 slices of the Xilinx Virtex-5 XC5VLX50 device and provide a latency of 295 clock cycles with a throughput of 51.32 Mbps at the maximum frequency of 236.574 MHz. Another implementation using 64-bit datapath has been reported in [16] that consumes 74 slices of the Xilinx Spartan-6 XC6SLX16-3CSG324C FPGA device. At maximum clock frequency of 221.63 MHz and with 33 clock latency, a throughput of 221.63 Mbps has been obtained. Similar to this a 64-bit datapath based architecture has been synthesized on 87 slices of the Xilinx Virtex-5 XC5VLX50 FPGA device [17]. Here a latency of 47 clocks, maximum frequency of 221.64 MHz and a throughput of 341.64 Mbps have been reported.

As evident from the above, most of the authors have provided architecture for encryption-only operation and they have assumed that the decryption operation works opposite to that of the encryption operation and hence would require roughly same logic complexities and hardware resources. However, in our opinion the decryption operation is a bit complex in comparison to the encryption operation. This is due to the fact that to start the decryption operation, first we need the last round key that is generated from the key scheduling operation. In connection to this, one of the implementations in which both encryption and decryption operations have been tackled [11], assumes that last round key is available at the starting of the decryption operation. They have considered that the key is static in nature throughout all of the encryption and decryption operations for all the input blocks. In another implementation by [17], for PRESENT cipher context it has been given that the decryption operation requires almost same area as of encryption when implemented separately. In the following section, an integrated encryption and decryption architecture has been proposed and described in detail.

## IV. An Integrated VLSI Architecture for the PRESENT Lightweight Block Cipher

The proposed architecture for an integrated encryption and decryption of PRESENT block cipher is shown in Fig. 3. Here, an iterative type of architecture is considered for saving the resources and computation time. To implement the PRESENT block cipher a 64-bit datapath is chosen, mainly to implement the permutation operation efficiently.

In the proposed architecture there are three main components which are encryption/decryption engine, key scheduling unit and controller. There is a 1-bit input signal '*enc_dec*' which is used to select the encryption or decryption operations. If '*enc_dec*' is at logic '1' level then encryption operation is performed, else the decryption operation is executed. An up-down counter facilitates the integrated encryption/decryption operation. The main building blocks of the proposed architecture have been arranged in different subsections which are described below.

### A. *Datapath of the Integrated PRESENT Architecture*

As shown in the Fig. 3, the architecture consists of a set of registers, multiplexers and XOR gates. The bit permutation (1) and inverse bit permutation operations (2) are simple bit-transposition, which require only simple wirings. There is one 64-bit multiplexer which is required to switch the state between the load phase (Input) and the intermediate state. After that the multiplexer passes the state to a 64-bit state register. This register is used to store the intermediate state and passes it to the 64-bit XOR gate. This gate performs the XOR operation of intermediate state coming from the state register with 64-bit round key coming from the key scheduling unit. In the architecture, both the S-box ($S$) and inverse S-box ($S^{-1}$) are realized by the area-optimized combinational logic implementation. To differentiate between the encryption and decryption operations, two 64-bit multiplexers are deployed in the datapath. In the proposed architecture, the inputs and output are registered. The output register is added to synchronize the output with the last round.

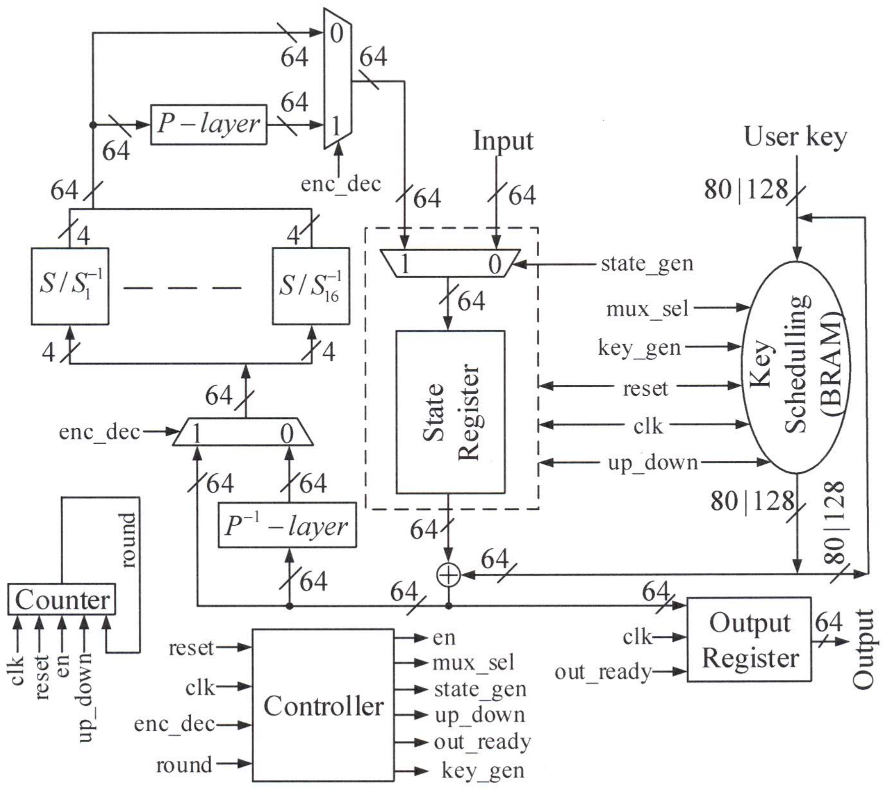

Fig. 3. A VLSI architecture of the PRESENT block cipher for an integrated encryption/decryption operation.

As per the Fig. 3, a total of 33 clock cycles are required for the encryption operation to get the ciphertext. Out of 33 clock cycles, the first cycle is used to load the user key and plaintext. In next 31 clock cycles, 31 intermediate states are computed for each of the rounds and finally in the last clock cycle output is available at the output register. Whereas, in the decryption operation, total 64 clock cycles are required to obtain the first block of decrypted output. Out of these 64 clock cycles, the 31 clock cycles are required to compute the last round key and 33 clock cycles are required to perform the decryption operation. Here, the computed keys have been simultaneously stored in a block RAM (BRAM) so that there is no need to compute the last round key for other blocks of input. Thus, only 33 clock cycles are required to decrypt the remaining blocks of ciphertext. The advantage of using the integrated architecture is that there are some resources which can be used in both encryption and decryption operations. The detailed description of the key scheduling operation is detailed below.

## B. *Architecture for the Key Scheduling Operation*

Key scheduling unit works on the storage mode. Here, the computation of the round keys is performed only for the first block of data and the computed round keys are stored simultaneously in the BRAM. This computation mode offers reduced number of clock cycles for the decryption operation. The key storage mode is also beneficial for processing a large chunk of data which contains multiple blocks that have to be encrypted or decrypted with the same key. A detailed description of the round key computation for both 80-bit and 128-bit key lengths is given in [9]. The key scheduling unit is shown in the datapath of Fig. 3. Detailed architectures of the key scheduling unit for the 80-bit and 128-bit key lengths are given in Fig. 4.

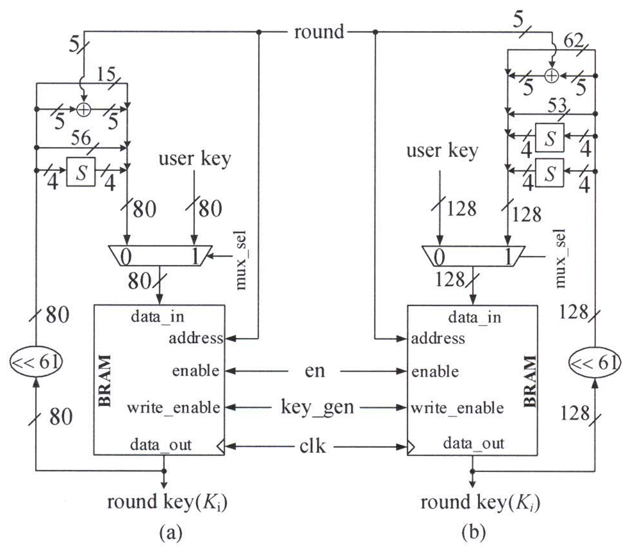

Fig. 4. The key-scheduling process in the PRESENT cipher (a) with 80-bit input key (b) with 128-bit input key.

There are three steps in the key scheduling process. Firstly, the intermediate key is left-rotated by 61 bits. Then, the first leftmost 4 bits are passed through one S-box for 80-bit key length and two S-boxes for 128-bit key length, as shown in the Fig. 4 (a) and (b) respectively. Finally, in the third step, 5 bits of the intermediate key are XORed with the counter value and after that the 80-bit (or 128-bit) key is written to *data_in* port of the BRAM. The intermediate key is read out from the *data_out* port of the BRAM and first leftmost 64 bits of intermediate key i.e. round key is XORed with the intermediate state of that particular round. There are three control signals, '*en*', '*key_gen*' and '*mux_sel*' generated by the controller for monitoring the key scheduling process, which are as shown in the Fig. 4. Alongwith the BRAM, one 80-bit or 128-bit multiplexer is also used. The '*mux_sel*' signal is used to switch between the load phase (user key) and intermediate keys. The '*key_gen*' signal is given to the *write_enable* port of the BRAM and it enables the BRAM for updating its content. The controller which controls the datapath and the key generation unit is described in the following section.

## C. *Controller for the Encryption/Decryption Operations*

The controller for the integrated encryption/decryption operation of the PRESENT cipher is given in Fig. 5. To make the proposed architecture (as in Fig. 3), capable of performing the encryption and decryption operations for the multiple blocks with the same user key, an appropriate controlling mechanism is required. The controller, thus, is designed to provide various control signals for monitoring the key generation unit and for controlling the encryption/decryption engine. There are total nine states in the controller, which are used for performing the encryption and decryption operations. Out of these nine states, two states i.e. Start ($S_0$) and the Reg_out ($S_{OUT}$) are common for both the encryption and decryption. In the first state, $S_0$, the counter is enabled through '*en*' signal and works as an up counter by setting the '*up_down*' signal to logic '1'. In this state, input and the user keys are loaded and the key scheduling unit starts computation through '*key_gen*' signal which is at logic '1'. In the next clock cycle, the state switches to either $S_{1E}$ state or $S_{1D}$ state for *enc_dec*='1' or '0' respectively.

For controlling the encryption operation given in the Fig. 5, in the state $S_{1E}$, multiplexers are switched as '*mux_sel*' signal for the key scheduling unit gets logic '1' and the signal '*state_gen*' for the datapath gets high so the encryption of first block of plaintext begins in this state. Simultaneously, all the generated round keys are stored in the BRAM. The state remains in $S_{1E}$, until the counter value reaches 31. Then, the state switches to $S_{OUT}$ state, where the counter is disabled through *en*='0' and '*out_ready*' signal becomes logic '1'. In this state the key generation unit finishes its computation operation and it is only used for fetching the stored round keys. So, here, the '*key_gen*' signal is kept at logic '0'. In the next clock cycle, first block of ciphertext is available through the output register.

Further, for the encryption of other blocks of plaintexts, the state switches to the $S_{2E}$ state. In this state, other blocks of plaintext are loaded to the state register. This state is very similar to the initial state $S_0$, except that now there is no need to compute the round keys as they have already been stored in the BRAM. So, here, '*key_gen*' signal is at logic '0'. After that, the state switches to the $S_{3E}$ state. The state $S_{3E}$ is similar to the state $S_{1E}$ except for the condition when '*key_gen*' signal changes to logic '0'. When the counter value reaches at 31, the state switches to the output state $S_{OUT}$. This cycle continues till all the blocks of plaintexts are encrypted.

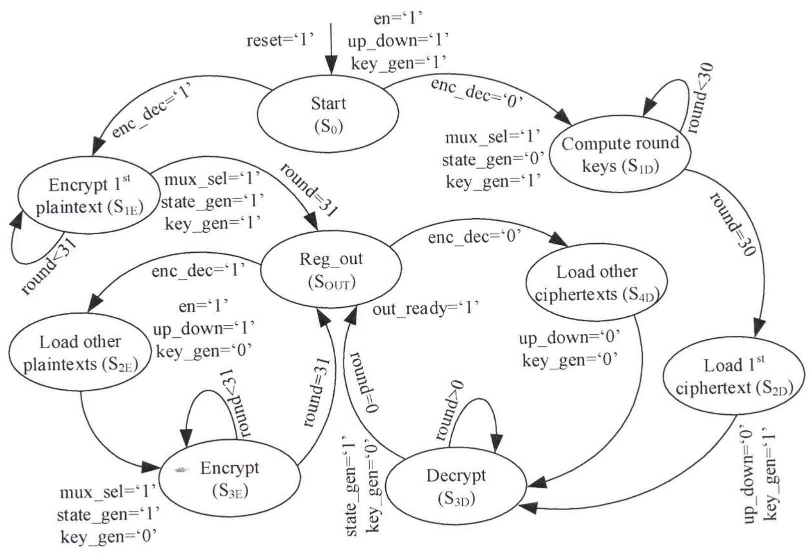

Fig. 5. Controller for the integrated encryption and decryption operations.

In the processing of the decryption operation, which is shown in the Fig. 5, the state switches at $S_{1D}$ from the initial state $S_0$. In the state $S_{1D}$, all the round keys are computed and stored in the BRAM. In this state, the decryption process does not start as the '*state_gen*' signal is at logic '0'. When the counter value reaches to 30, the state is switched at $S_{2D}$ state. Here, first bock of ciphertext is loaded to the state register and now the counter starts to work as down counter by switching the '*up_down*' signal to logic '0'. Further, the state is switched to $S_{3D}$ state, where the decryption process starts by assigning the '*state_gen*' signal to logic '1'. Further, the key generation unit completes its computation operation with the '*key_gen*' signal becoming logic '0'. As counter value becomes 0, the state switches to the final state $S_{OUT}$ and first block of decrypted ciphertext is available through the output register as '*out_ready*' signal becomes logic '1'. Next, to decrypt the remaining blocks of ciphertext the state does not go back to the initial state because pre-computed round keys are available in the BRAM. So, the state moves to $S_{4D}$ state, which is very similar to the state $S_{2D}$ except that '*key_gen*' signal is at logic '0'. In this state, other blocks of ciphertext are loaded into the state register. To perform the decryption operation for the rest blocks, the state switches to the $S_{3D}$ state. This cycle continues until all the blocks of ciphertext are decrypted.

## V. EXPERIMENTAL RESULTS FOR AN FPGA DEVICE

The proposed architecture is implemented in the VHDL language, and synthesized using Xilinx Design Suite 14.7 for the Xilinx Virtex-5 XC5VLX110T-1-FF1136 FPGA device on Xilinx ML-505 platform. The synthesis process for the implementation has been configured with design goal as *balanced* and strategy as *Xilinx default*. The FPGA device utilization summary of the proposed architecture is given in Table I.

As per the table, the architecture with 80-bit key (*PRE_80*) is consuming only 0.73% slices while the proposed architecture with 128-bit key (*PRE_128*) is consuming 0.87% slices. The *PRE_80* architecture consumes $32 \times 80$ bit size block memory, whereas, the *PRE_128* architecture needs $32 \times 128$ bit size of memory to store the intermediate keys.

TABLE I. DEVICE UTILIZATION SUMMARY ON THE XILINX VIRTEX-5 XC5VLX110T FPGA DEVICE.

| Elements | Available Resources | Resource Utilization | Resource Utilization |
|---|---|---|---|
| | | *PRE_80* | *PRE_128* |
| LUTs | 69120 | 348 | 396 |
| Registers | 69120 | 137 | 137 |
| Total Slices | 17280 | 126 | 150 |

The performance of the design is evaluated in terms of power dissipation, latency, maximum frequency and throughput. The performance of the proposed architecture is given in Table II.

TABLE II. PERFORMANCE ON THE XILINX VIRTEX-5 XC5VLX110T FPGA DEVICE.

| Elements | Resource Utilization | Resource Utilization |
|---|---|---|
| | *PRE_80* | *PRE_128* |
| Latency | 33 | 33 |
| Max. frequency (MHz) | 215.42 | 212.13 |
| Throughput (Mbps) | 417.79 | 411.41 |
| Efficiency (Mbps/#Slices) | 3.32 | 2.74 |
| Power (mW) | 16.59 | 16.80 |

As given in the Table II, the architecture is consuming around 16 mW power at 215 MHz frequency. Both the operations have a latency of 33 clock cycles. However, the decryption operation requires an additional 33 cycles for round key generation for the first block of ciphertext. Throughput of the design is found to be around 410 Mbps for both the key lengths, that is computed for 64-bit datapath by expression, $throughput = (\max . frequency \times total\ no.\ of\ bits)/latency$.

To compare the proposed design with an existing design available in literature, the selected design metrics are: slice LUTs, registers and total number of consumed slices. To perform a comparison at the architectural-level, the proposed integrated architecture is tuned to match the architectural capability of [11]. Therefore, for the comparison the key scheduling unit is implemented using *on-the-fly* mode rather than storing the computed keys in the BRAM. The following section shows an architectural-level comparison between the proposed design and the design of [11].

## *A. Architectural-level Comparison*

The architecture presented in [11] is one of a few established papers which provides decryption operation for the FPGA. The implementation of the architecture [11] has been done on the Xilinx Spartan-IIIXC3S400 FPGA device. Thus, to perform a fair comparison of utilized device resources, we have targeted the same FPGA device and equal speed grade. Similar to [11], we also kept the synthesis tool project design goals and strategies with synthesis and place and route (PnR) effort properties as *high* and the PnR extra effort at *continue on impossible*. The implementation has been performed for both 80-bit key length (*PRE_80*) and 128-bit key length (*PRE_128*). The synthesis results for both the architectures are compared and is shown in Fig. 6.

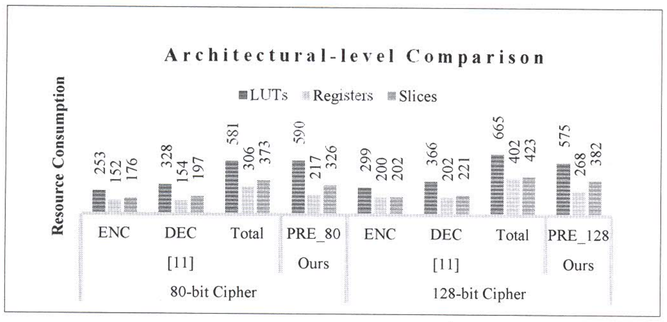

Fig. 6. Architectural-level comparison between architecture of [11] and the proposed architecture using Xilinx Spartan-III XC3S400 FPGA device.

All the data presented in Fig. 6, are from the post place and route (*PnR*) report. It can be observed from the above figure that in comparison to the implementation reported in [11], the proposed architecture with 80-bit key length (*PRE_80*) requires 12.6% lower FPGA slices and with 128-bit key length (*PRE_128*) consumes 9.7% lesser slices. By this we can say that the proposed integrated architecture is capable of performing both the encryption (ENC) and decryption (DEC) by the same set of hardware, which is an essential requirement in any practical lightweight cipher-based system. Also the integrated architecture consumes lesser slices in comparison to two separate modules for performing encryption and decryption. It can be noted that our design requires an extra clock cycle in comparison with [11] to perform the operations as we have considered the registered output.

## VI. Conclusion

An integrated VLSI architecture for PRESENT lightweight block cipher has been presented. The architecture supports both the encryption and decryption operations with 80-bit and 128-bit key lengths. The design has been modeled in the VHDL language and synthesized in the Xilinx Virtex-5 XC5VLX110T-1-FF1136 FPGA device on ML-505 platform. The presented architecture utilizes 0.73% and 0.87% of FPGA slices for 80-bit and 128-bit key length respectively. Throughput of the design is around 410 Mbps and power consumption is around 16 mW for both the key lengths. Thus, the proposed architecture is area-efficient with high-performance capability for providing an adequate level of security under the resource-constraine environment for IoT and CPS applications.

## References

[1] E. A. Lee and S. A. Seshia, Introduction to Embedded Systems, A Cyber-Physical Systems Approach, 2011.

[2] T. Xu, J. B. Wendt and M. Potkonjak, "Security of IoT systems: Design challenges and opportunities," in *IEEE/ACM Int'l Conf. on Computer-Aided Design*, San Jose, California, pp. 417-423, 03 Nov. 2014.

[3] A. Biryukov and L. Perrin, "Lightweight Block Ciphers," [Online]. Available: https://www.cryptolux.org/index.php/Lightweight_Block_Ciphers. [Accessed 06 Jan. 2017].

[4] K. McKay, L. E. Bassham, M. S. Turan and N. W. Mouha, "NISTIR 8114 - Report on Lightweight Cryptography," National Institute of Standards and Technology (NIST), Gaithersburg, March 2017.

[5] B. J. Mohd, T. Hayajneh and A. V. Vasilakos, "A survey on lightweight block ciphers for low-resource devices: Comparative study and open issues," *Journal of Network and Computer Applications,* vol. 58, pp. 73-93, 2015.

[6] T. Eisenbarth, S. Kumar, C. Paar, A. Poschmann and L. Uhsadel, "A survey of lightweight-cryptography implementations," *IEEE Design & Test of Computers,* vol. 24, no. 6, pp. 522-533, 2007.

[7] A. Y. Poschmann, "Lightweight cryptography: cryptographic engineering for a pervasive world," PH. D. Thesis, Ruhr-University, Bochum, Germany, 2009.

[8] "Information technology – Security techniques – Part 2: Block ciphers," Jan. 2012.

[9] 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," in *Int'l Workshop on Cryptographic Hardware and Embedded Systems*, Vienna, Austria, Springer, pp. 450-466, 2007.

[10] Xilinx Inc., "Virtex-5 FPGA user guide UG190 (v5.4)," www.xilinx.com, 2012.

[11] M. Sbeiti, S. Michael, A. Poschmann and C. Paar, "Design space exploration of present implementations for FPGAS," in *5th Southern Conference on Programmable Logic (SPL)*, Sao Carlos, Brazil, pp. 141-145, 1-3 April 2009.

[12] C. Rolfes, A. Poschmann, G. Leander and C. Paar, "Ultra-lightweight implementations for smart devices–security for 1000 gate equivalents," in *Int'l Conf. on Smart Card Research and Advanced Applications*, London, UK, pp. 89-103, 08-11 Sep. 2008.

[13] P. Yalla and J. P. Kaps, "Lightweight cryptography for FPGAs," in *IEEE Int'l Conference on Reconfigurable Computing and FPGAs (ReConFig'09)*, Cancun, Mexico, pp. 225-230, 09 Dec. 2009.

[14] E. B. Kavun and T. Yalcin, "RAM-based ultra-lightweight FPGA implementation of PRESENT," in *Int'l Conf. on Reconfigurable Computing and FPGAs (ReConFig'11)*, Cancum, Mexico, pp. 280-285, 30 Nov-2 Dec 2011.

[15] J. J. Tay, M. L. D. Wong, M. M. Wong, C. Zhang and I. Hijazin, "Compact FPGA implementation of PRESENT with Boolean S-Box," in *6th Asia Symp. on Quality Electronic Design (ASQED)*, Kula Lumpur, Malaysia, pp. 144-148, 04 Aug. 2015.

[16] C. A. Lara-Nino, M. Morales-Sandoval and A. Diaz-Perez, "Novel FPGA-based low-cost hardware architecture for the PRESENT block cipher," in *2016 Euromicro Conference Digital System Design (DSD)*, Limassol, Cyprus, pp. 646-650, 31 Aug. 2016.

[17] N. Hanley and M. O'Neill, "Hardware Comparison of the ISO/IEC 29192-2 Block Ciphers," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Amherst, MA, USA, pp. 57-62, 19-21 Aug. 2012.