Fast and Efficient Implementation of Lightweight Crypto Algorithm PRESENT on FPGA through Processor Instruction Set Extension

Abdullah Varici
EE Dept.
Ozyegin University
Istanbul, Turkey
abdullah.varici@ozu.edu.tr

Gurol Saglam
CS Dept.
Ozyegin University
Istanbul, Turkey
gurol.saglam@ozu.edu.tr

Seckin Ipek
EE Dept.
Ozyegin University
Istanbul, Turkey
seckin.ipek@ozu.edu.tr

Abdullah Yildiz
CSE Dept.
Yeditepe University
Istanbul, Turkey
ayildiz@cse.yeditepe.edu.tr

Sezer Gören
CSE Dept.
Yeditepe University
Istanbul, Turkey
sgoren@cse.yeditepe.edu.tr

Aydin Aysu
ECE Dept.
North Carolina State University
North Carolina, United States
aaysu@ncsu.edu

Deniz Iskender
CS Dept.
Ozyegin University
Istanbul, Turkey
deniz.iskender@ozu.edu.tr

T. Baris Aktemur
CS Dept.
Ozyegin University
Istanbul, Turkey
aktemur@gmail.com

H. Fatih Ugurdag
EE Dept.
Ozyegin University
Istanbul, Turkey
fatih.ugurdag@ozyegin.edu.tr

***Abstract*—As Internet of Things (IoT) technology becomes widespread, the importance of information security increases. PRESENT algorithm is a major lightweight symmetric-key encryption algorithm for IoT devices. Compared to the Advanced Encryption Standard (AES), PRESENT uses a lower amount of resources while achieving the same level of security. In this paper, we implement PRESENT with different design methodologies including hand-coded RTL, Vivado HLS, PicoBlaze, VerySimpleCPU (VSCPU) based microcontrollers, and a customized VSCPU. The customized VSCPU design is based on optimizing the instruction set architecture for the algorithm specifics of PRESENT. Our results show that the customized VSCPU design metholodogy can be more efficient than HLS and PicoBlaze while providing the flexibility compared to RTL designs.**

Keywords—Internet of Things, Information Security, Cryptography, PRESENT Algorithm, FPGA, VerySimpleCPU

I. INTRODUCTION

The Internet of Things (IoT) is a platform where smart home appliances, various sensors, wearable and many other devices communicate with each other over the Internet. The smart devices surrounding us gather information from the environment and they share the information by using processors and communication units with other smart devices over IoT platform to produce solutions to daily life problems. These smart devices are becoming more and more widely used in various fields such as agriculture, health, safety, education, and urbanism [8]. Such a situation makes information security in these devices covering many areas of our lives an important issue.

Data encryption before transfer and storage is a typical method to prevent unwanted access to confidential information. Encryption algorithms such as Advanced Encryption Standard (AES) [1] and Data Encryption Standard (DES) [2] are widely used for this purpose. However, since these algorithms require a large amount of hardware resources and high power consumption, it is not possible to use them in edge/IoT devices with limited resources. The Lightweight Cryptography field [9] has emerged as a solution to this problem by decreasing processing complexity and the area usage of the chips while providing a reasonable amount of security.

One of the most important components of modern electronic systems are the processors and the software enables the processors to perform the desired operations in order. The instruction sets of many commercially available processors are non-modifiable. This leads to the fact that the instructions, which the applications sometimes do not use at all, consume unnecessary hardware resources. In addition, much needed application specific instructions are not in the instruction set causing serious decreases in performance or the application cannot perform the desired operation at all. To avoid these problems, there is a need for application-specific hardware design and changing the instruction set of processors.

In this work, we discuss a methodology by extending the instruction set of the VerySimpleCPU (VSCPU) [5] for efficient design of any application-specific algorithm on the devices with limited resources. To that end, we implement and compare the results obtained by designing the desired algorithm with pure RTL coding, Vivado High Level Synthesis (HLS), PicoBlaze, and the unmodified VSCPU. We use

978-1-7281-1003-5/19/$31.00 ©2019 IEEE

FPGAs as the target platform for our implementations and choose PRESENT [3], the popular lightweight cryptography algorithm, as the example algorithm in this work.

Our results shows that customizing the VSCPU core can lead to a very desirable solution by combining the best of both worlds. On the one hand, it can provide the flexibility of a software-based solution. On the other hand, it can lead to a more efficient design than HLS and Picoblaze microcontrollers through hardware customization. This work therefore shows an exciting opportunity for IP design of next-generation cryptography standards.

II. PRESENT ALGORITHM

The PRESENT [3] encryption algorithm is a lightweight symmetric-key block cipher algorithm. It has the Substitution Permutation Network (SPN) structure and has a 64-bit block length along with an 80-bit or 128-bit key length. As shown in Fig. 1, the algorithm consists of 31 rounds and the last round of key addition. A round consists of the following 3 functions: add round key, substitution box layer, and permutation layer.

Adding the round key is simply the XOR operation of the state data block with the round key. The nonlinear substitution layer uses identical Substitution Box (SBox) that converts a 4-bit input to a 4-bit output. Fig. 2 shows the input-output relationship of a single Sbox. In the permutation layer, the bit value in the index *m* is moved to the index *n = P (m)* for all indexes. Fig. 3 provides the construction of the P function.

PRESENT uses distinct round-keys at each round. Fig. 4 formulates the key schedule operation, which first performs a 61-bit left rotation. Then, the left-most 4-bit for 80-bit keys and the left-most 8-bit for 128-bit keys are passed through the Sboxes. Finally, the round counter value is subjected to XOR operation with the bits of 19:15 indexes of the previous result. The most significant 64-bit of the value forms the round key.

III. THE IMPLEMENTATION OF THE PRESENT ALGORITHM

To implement the PRESENT algorithm, we use Xilinx's ISE 14.7 software for synthesis, simulation, and placement and routing, and Vivado HLS 2017.1 software for high-level synthesis (HLS). We select the Xilinx XC7A100T-1CSG324 FPGA and use the 80-bit key version of PRESENT. Tables I and II summarize the results.

A. Implementation with Hand-Coded RTL

Designing a custom chip is the first option that comes to mind when designing hardware with low resource consumption and high-performance. However, because of the high cost per product of chip design for small quantities of products, Field Programmable Gate Arrays (FPGA) are preferred. FPGAs are also used to ensure shorter design time and prototyping of ASICs. FPGA design is made using a Hardware Description Language (HDL). The two most commonly used languages are Verilog and VHDL. FPGAs are programmed after the design's synthesis and implementation stages with selected FPGA software.

generateRoundKeys()
for $i = 1$ to 31 **do**
  addRoundKey(STATE,K_i)
  sBoxLayer(STATE)
  pLayer(STATE)
end for
addRoundKey(STATE,K_{32})

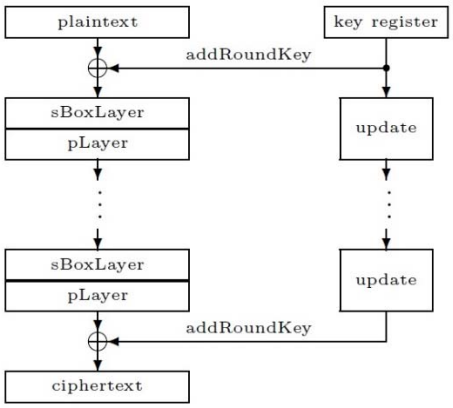

Fig. 1. PRESENT encryption algorithm [3]

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

Fig. 2. Substitution box layer [3]

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P(i)$	0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P(i)$	4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P(i)$	8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$P(i)$	12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

Fig. 3. Permutation layer [3]

1. $[k_{79}k_{78}\ldots k_1k_0] = [k_{18}k_{17}\ldots k_{20}k_{19}]$
2. $[k_{79}k_{78}k_{77}k_{76}] = S[k_{79}k_{78}k_{77}k_{76}]$
3. $[k_{19}k_{18}k_{17}k_{16}k_{15}] = [k_{19}k_{18}k_{17}k_{16}k_{15}] \oplus \texttt{round\_counter}$

Fig. 4. Key schedule [3]

Before carrying out other implementations, we wanted to find out hardware resource consumption and typical performance of the PRESENT algorithm with hard-coded, i.e., pure RTL design on FPGA. For this purpose, we reviewed the RTL design shared on GitHub [4]. We also simulated, synthesized and implemented the design on the selected FPGA without the need for optimization. The design consumed 213 LUTs and 213 FFs from FPGA resources after implementation. It also can work with a clock frequency up to 368 MHz, and it can compute a result in 32 cycles (latency).

TABLE I. MEMORY RELATED INFORMATION ON PROCESSOR BASED IMPLEMENTATIONS

Processor / Memory	*PicoBlaze*	*VSCPU compiled code*	*VSCPU-80*	*VSCPU-16*	*[13]*
Instruction Bit Length	18	32	10	12	8
Number of Instructions in the Program	509	2210	16	45	156
Data Bit Length	8	32	80	16	8
Number of Data in the Program	-	58	8	16	48

TABLE II. HARDWARE RESOURCE CONSUMPTION AND PERFORMANCE INFORMATION OF IMPLEMENTATIONS ON FPGA

Implementation / Consumption and Performance Information	*Pure RTL*	*Vivado HLS*	*PicoBlaze*	*VSCPU compiled code*	*VSCPU-80*	*VSCPU-16*	*[11]*	*[12]*	*[13]*
Look-Up Tables (LUT)	213	466	292	8966	1023	270	222	226	377
Flip Flops (FF)	213	765	104	3203	507	291	201	89	N/A
Max. Clock Frequency in MHz	368	125	185	67	141	153	236	172	542
Latency in Cycles	32	29k	360k	558k	502	1402	295	132	20k
Latency	87 ns	233 us	2 ms	8 ms	4 us	9 us	1 us	767 ns	37 us
Latency x (LUTs + FFs)	37k	286M	768M	101G	5.4M	5.1M	528k	242k	14M

With the pure RTL design, the best results were obtained in terms of performance and area consumption and these results were used for comparison with other implementations. The design to be done in this way has disadvantages such that the design is coded and validated manually so that the design time is too long and that the design does not offer any flexibility to the lack of programmability.

B. Implementation with Vivado HLS

Vivado is a complete software that allows performing RTL design, simulation, synthesis, implementation and Xilinx FPGA programming. To make FPGA programming simpler and reduce the design time, Xilinx has developed the Vivado HLS. Thanks to this program, a C-based code can be transformed into a Verilog or VHDL design which allows synthesis and implementation on FPGAs.

We used the C-based design of the PRESENT algorithm from FELIX project [10] on Vivado HLS 2017 and confirmed that the design works correctly. Afterward, synthesis and implementation of the obtained RTL design from HLS were performed. The resulting design consumes 466 LUTs and 765 FFs, achieves a clock frequency of up to 125 MHz, and takes 29124 cycles to complete an encryption operation.

Designing with HLS significantly reduces design time. However, when the obtained results are examined, it is seen that resource consumption increases approximately twice as compared to the pure RTL implementation. In addition, an encryption operation results in 32 cycles in the pure RTL implementation, while 29124 cycles in the Vivado HLS implementation. Considering the maximum clock frequencies that the designs can operate, it is seen that there is a decrease of approximately 2681 times in performance. Moreover, as with the pure RTL design, the design produced by Vivado HLS does not include a processor, so it does not allow software development.

C. Implementation with PicoBlaze

PicoBlaze [7] is an 8-bit processor with low area consumption and low performance for Xilinx's own FPGAs. This processor goes through the synthesis and implementation stages and works on the resources in Xilinx FPGAs. There are a total of 70 commands in PicoBlaze and all commands are executed in 2 cycles.

In order to implement PRESENT algorithm with PicoBlaze, the algorithm was written using Assembly language and PicoBlaze instruction set. Initially, the Assembly program consisted of 1446 instructions, and the design's program memory would consume a significant amount of resources. To optimize the code, we review it and reduce the number of assembly instructions to 509. Since FFs are used as the memory element in all implementations and in order to make the comparison under equal conditions, the memory elements of PicoBlaze are changed to FFs and LUTs from Block RAMs. Afterwards, we performed synthesis and implementation using the memory file from Assembly code and PicoBlaze's source code for Xilinx ISE. Eventually, the design working with a clock frequency up to 185 MHz consumed 292 LUTs and 104 FFs of FPGA resources and the latency for an encryption was 359943 cycles.

The results show us that, by using PicoBlaze, it is possible to create a low resource consuming design, as in pure RTL design. In addition, due to it's processor based architecture, software development on this implementation is possible. However, when the clock frequencies and the number of cycles required for an encryption are considered, PicoBlaze

implementation is approximately 22324 times slower than the pure RTL design.

D. Implementation with VSCPU

VSCPU [5] is a simple and customizable 32-bit processor that can be implemented on FPGAs, with an instruction set simulator, Assembly code generator, and C compiler. VSCPU has 16 instructions in its instruction set and supports unsigned integer arithmetic operations. The RTL codes of the processor are written in Verilog language and the structure can be changed completely as required.

We compiled the C code of the PRESENT algorithm with the C compiler of the VSCPU and extracted the Assembly code. As a result, 2210 32-bit instruction memory, which is stored in a ROM, and 58 32-bit data memory, which stored in a RAM, were needed. The result of encryption operation is computed at the end of 557997 cycles and the design can work with clock frequency of up to 67 MHz using 8966 LUTs and 3203 FFs.

Considering the resulting resource consumption and performance, we can say that VSCPU implementation has the worst results among other options. The reasons for this may be that the instruction set and architecture of VSCPU are not well suited to the PRESENT algorithm, and that the C code compiler does not produce good results. However, this implementation has the advantages that it provides a much faster design opportunity than a pure RTL implementation and allows for software development due to its processor based structure.

E. Implementation with Instruction Set Modified VSCPU

Due to the lack of XOR operation and algorithm specific commands such as SBOX in the VSCPU's instruction set, the compiled Assembly code consumed a lot of memory and took too long to compute an encryption operation. Therefore, to improve performance and reduce resource consumption, we changed the instruction set of the VSCPU and wrote the Assembly code manually.

Firstly, because the key length is 80-bit, we increased the length of the data bus and the words in the data memory to 80-bit length. We also added four new instructions in the algorithm that implement substitution (SBOX), left rotation (RRL), permutation (PER), and XOR functions. We removed instructions that were not used in the algorithm from the instruction set.

Another change in the VSCPU was the transition from a combined memory unit in the Von Neumann architecture to the separate storage of data and instructions, as in the Harvard architecture. The aim was to take advantage of the fact that the bit length of the instructions is shorter than the bit length of the data and to consume fewer resources.

This design required 16 instruction words of 10-bit length and 8 data words of 80-bit length. The design, which completes an encryption in 502 cycles, can work with up to 141 MHz clock frequency, using 1023 LUTs and 507 FFs. When the design obtained is evaluated in terms of performance, it provides much higher performance than other realizations except pure RTL implementation. However, it gives better results in terms of resource consumption than only VSCPU.

TABLE III. INSTRUCTION SET OF THE VSCPU-16

Instruction	***Functionality***
XOR A B	*A = *A ^ *B
CP A B	*A = *B
ADD A B	*A = *A + *B
BZJ A B	if(*B == 0) jump *A
GETW A B	if(B == 0) W = 0 else *A = W; W = 0
SRW3 A B	*A = (*A >> 3) | W; W = *A << 13
SBOX16 A B	if(*B == 0) *A = { sbox(*A[15:12]) , *A[11:0] } else *A = { sbox(*A[15:12]), sbox(*A[11:8]), sbox(*A[7:4]), sbox(*A[3:0]) }
PER3 A B	W = { *A[15], *A[11], *A[7], *A[3], *B[15], *B[11], *B[7], *B[3], W[15:8] }
PER2 A B	W = { *A[14], *A[10], *A[6], *A[2], *B[14], *B[10], *B[6], *B[2], W[15:8] }
PER1 A B	W = { *A[13], *A[9], *A[5], *A[1], *B[13], *B[9], *B[5], *B[1], W[15:8] }
PER0 A B	W = { *A[12], *A[8], *A[4], *A[0], *B[12], *B[8], *B[4], *B[0], W[15:8] }
BZJi A B	Jump *A+B

To further reduce the resource consumption of the VSCPU with the 80-bit data path (VSCPU-80) design, we designed a VSCPU with the 16-bit data path (VSCPU-16). As shown in Table III, we have made significant changes to the instruction set.

In VSCPU-16 design, an additional register named “W” has been defined to execute the algorithm in a more optimized manner. Since our key value is 80-bit, it is stored in 5 separate data addresses in this 16-bit architecture design. A new instruction called SRW3 was created, which also uses the W register to perform the 3-bit right shift required in the key schedule function. Using this instruction repeatedly, rotation of the key value which is found in different addresses can be made easily.

With the added SBOX16 instruction, the parameter 0 provides the SBOX conversion in the key schedule function only to a 4-bit section, and with the parameter 1, all input values are converted through SBOX operation.

The permutation function is also provided by calling the PER0, PER1, PER2 and PER3 instructions twice in succession. [6] is used to create these instructions.

With the transition from 80-bit architecture to 16-bit, 45 instruction words of 12-bit and 16 data words of 16-bit were needed, and the algorithm computed the result in 1402 cycles. However, the new design can operate with a clock frequency up to 153 MHz and consumes 270 LUTs and 291 FFs. These results have approached pure RTL implementation in terms of FPGA resource consumption, yet it is seen that it consumes 41% more resources than PicoBlaze implementation. However, when the clock frequency and the latency of both

implementations are considered, it is seen that the VSCPU-16's performance is about 211 times better than the PicoBlaze.

F. Other Works

PRESENT is implemented on FPGAs after it's publication by other teams. We examined some of the most recent and promising researches and added their results to Table I and II.

In 2015, Tay *et al*. developed 8-bit hardware architecture of PRESENT algorithm on a Virtex-5 FPGA [11]. They managed to reduce the amount of resources for SBoxes due to 8-bit datapath, and Karnaugh mapping and further factorization of SBoxes. The resulting implementation consumes 222 LUTs and 201 FFs; it can work with a clock frequency of up to 236 MHz; and it can compute a result in 295 cycles.

In 2016, Lara-Nino *et al.* created an architecture based on a 16-bit datapath [12]. By doing that, the implementation of PRESENT on Spartan-6 FPGAs consumes only 226 LUTs and 89 FFs and the design works with a clock frequency up to 172 MHz and with a latency of 132 cycles.

Diehl *et al.* presented the implementation of 6 different ciphers including PRESENT using both custom hardware design and software design with 8-bit microprocessor [13]. There are only 30 native instructions on this soft microprocessor, and the data words and instruction words are 8-bit length. The processor is tailored to the algorithms and unrequired functionality is removed before implementation as in our work. PRESENT implementation on Kintex-7 FPGAs using this custom processor consumes 377 LUTs, achieves a clock frequency up to 542 MHz, and takes 20030 cycles to complete an encryption operation. 156 instruction words and 48 data words are used in the implementation.

Our pure RTL implementation gives better result than [12] in every aspect and consumes almost the same amount of FPGA resources with [11]. However, the throughput of our pure RTL implementation is almost 6 times higher than of the [11]. Moreover, our VSCPU design with 16-bit architecture is more efficient and consumes lower resources than [13].

IV. Conclusion

In this work, we aim to make high performance and efficient implementation of any application specific algorithm for devices with low hardware resources in a short time. As an exemplary application, we chose the PRESENT cipher algorithm on IoT. In order to compare the results, this algorithm was implemented on FPGA with pure RTL, Vivado HLS, PicoBlaze, VSCPU and modified VSCPU. In addition, we examined some of the most recent and promising researches. The modified VSCPU implementation, which has been modified by changing the instruction set and architecture, gave us the best results due to its low resource consumption, high performance and the ability to develop software on it.

In future studies, VSCPU's compiler and simulator can work according to the changing instruction set and architecture and rapid prototyping can be realized completely. In this way, it would be possible to make high performance design in a short time.

Acknowledgement

This work is partially supported by a TÜBİTAK (The Scientific and Technological Research Council of Turkey) ARDEB 1001 project (no: 117E090). Prof. Sezer Gören is the PI of the project. While Prof. T. Baris Aktemur is the initial co-PI of the project, Prof. H. Fatih Ugurdag is the current co-PI. Abdullah Yildiz and Deniz Iskender are among the research assistants of the project.

References

[1] J. Daemen and V. Rijmen, "AES proposal: Rijndael", 1999.

[2] "Data Encryption Standard," Federal Information Processing Standards Publication No. 46, National Bureau of Standards, January 15, 1977.

[3] 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", Cryptographic Hardware and Embedded Systems (CHES), Lecture Notes in Computer Science, pp. 450–466, 2007.

[4] "Implementation of the PRESENT lightweight block cipher in VHDL", Github 11.08.2019, https://github.com/huljar/present-vhdl.

[5] A. Yildiz, H.F. Ugurdag, B. Aktemur, D. Iskender, and S. Gören, "CPU design simplified," International Conference on Computer Science and Engineering (UBMK), 2018.

[6] E.B. Kavun and T. Yalcin, "RAM-Based Ultra-Lightweight FPGA Implementation of PRESENT," International Conference on Reconfigurable Computing and FPGAs, 2011.

[7] "PicoBlaze 8-bit Microcontroller", Xilinx, 11.08.2019, https://www.xilinx.com/products/intellectual-property/picoblaze.html.

[8] D. Bandyopadhyay and J. Sen, "Internet of Things: Applications and Challenges in Technology and Standardization," Wireless Personal Communications, vol. 58, no. 1, pp. 49–69, Sep. 2011.

[9] "Report on Lightweight Cryptography", National Institute of Standards and Technology Internal Report 8114, March, 2017.

[10] D. Dinu, A. Biryukov, J. Groszschaedl, D. Khovratovich, Y.L. Corre, L. Perrin, "FELICS - Fair Evaluation of Lightweight Cryptographic Systems", 2015.

[11] 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 2015 6th Asia Symposium on Quality Electronic Design (ASQED), Aug 2015, pp. 144–148.

[12] 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 on Digital System Design, 2016.

[13] W. Diehl, F. Farahmand, P. Yalla, J. Kaps and K. Gaj, "Comparison of Hardware and Software Implementations of Selected Lightweight Block Ciphers", in 2017 27th International Conference on Field Programmable Logic and Applications, 2017.