Hardware/Software Co-Design of a Lightweight Crypto Algorithm BORON on an FPGA

Burak Acar and Berna Ors

Istanbul Technical University, Turkey
burakacaritu@gmail.com, Siddika.Ors@itu.edu.tr

Abstract

New areas such as Internet of Things (IoT), smart home technologies and wearable technologies have brought security problems together. In order for these technologies to be implemented in the future, attention should be paid to the confidentiality of the produced data. The best way to achieve this is to use cryptography. This article is about hardware implementation of BORON, which is an energy efficient crypto algorithm with small footprint, on a Field Programmable Gate Array (FPGA) using hardware description language. The Diffie-Hellman key exchange protocol for secret key sharing is implemented using the Montgomery Modular Multiplier written in VHDL language. After testing that the hardware is fully functional, the software is designed with the C language on the ARM processor to control the entire system. Finally, the output of the hardware and software-designed Boron code on FPGA is printed on the screen via serial communication protocol (UART).

Keywords: IoT, Lightweight Cryptography, BORON, FPGA

1. Introduction

The emerging fields such as Internet of Things (IOT), smart home technology and wearable technology will play a very important role in the future to facilitate human life [1]. Thanks to these new technologies, people will be constantly connected devices they use regularly, too much data will be collected from the sensor networks and analysis of this data will be made to solve problems in daily life. At the same time, this situation means that each individual's personal information must be in the same environment and it can be accessed by everyone in certain ways. To make every device smart and connect them to internet causes external threats like information leakage, personal data capture and Distributed Denial of Service (DDoS) attacks [2]. The security measures of the devices against all possible types of attacks must be taken. In order for these technologies to be implemented in the future, attention should be paid to the confidentiality of data. In the past, many passwords have been designed to secure devices and protect personal data. AES [3] and Triple DES [4] are commonly used ciphers. As of now, there is no attack that could be successful against these ciphers in a reasonable amount of time.

These old ciphers have large hardware occupancy and high power consumption. It has become impossible to implement them in embedded systems with limited resources. All of these constraints led to the emergence of the area of Lightweight Cryptography [5]. Lightweight Cryptography aims to design encryption algorithms that have a robust design, takes up less space and have low power consumption with fewer than 2200 gate equivalent (GE) numbers [6].

In this project, we aim to design a cryptographic device with minimum space and minimum power consumption that can be used in low-scale embedded systems. When the lightweight crypto algorithms in the literature such as XTEA, SEA, SIMON, PRESENT and PRINT are examined, BORON is selected as most appropriate algorithm. Then the BORON crypto algorithm is implemented on a FPGA using Verilog language. The Diffie-Hellman key exchange protocol for secret key sharing is implemented using the Montgomery Modular Multiplier written in VHDL language. After testing that the hardware is fully functional, the software is designed with C language on ARM microprocessor to control the entire system. Finally, hardware and software-designed Boron cryptosystem is implemented on the Field Programmable Gate Arrays (FPGAs) and outputs are printed on the screen via serial communication protocol (UART).

2. The Lightweight Crypto Algorithm BORON

In this section, the properties of the BORON algorithm and data encryption algorithms are explained.

2.1. Properties

BORON cryptographic algorithm is a lightweight block cipher which have substitution-permutation network (SPN) structure [7]. It runs faster than other Feistel based ciphers. BORON consists of a round function that repeats 25 times and this feature of BORON makes it cryptographically strong. Having a non-linear layer in the substitution-permutation network (SPN) has resulted in good results in the active S_box count, making it secure against standard cryptographic attacks such as linear and differential cryptanalysis.

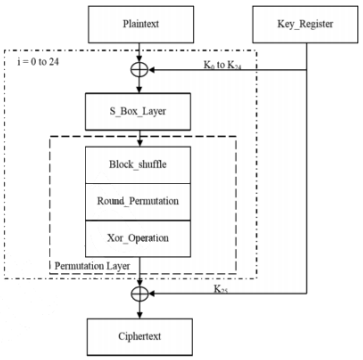

Fig. 1. Block diagram of a BORON cipher [7]

Key generation is based on key generation of PRESENT encryption algorithm [8]. The BORON encryption algorithm supports an 80-bit or 128-bit encryption text length, with a 64-bit plain text or encrypted text block length. An overview of the encryption algorithm is given in Figure 1. More detailed specifications will be given in the following sections.

2.2. Round Function

BORON cipher has a round function that repeats itself 25 times. Round Function has key addition, nonlinear permutation and linear layer as shown in Figure 2. The key addition layer performs bit insertion at the bit level. The non-linear layer is the S_box layer. It consists of 16 parallel-operated s_boxes with 4-bit input and output. The linear layer consists of block shuffle, round permutation and XOR operations, respectively.

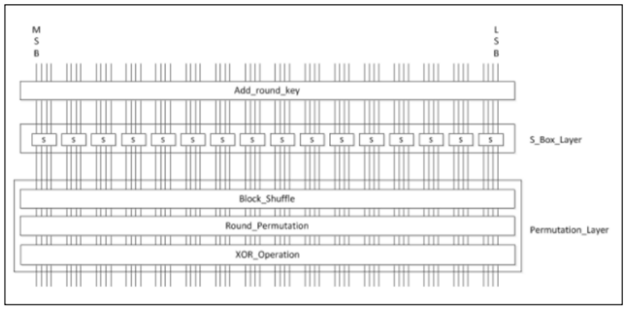

Fig. 2. Round Function Structure [9]

2.2.1. Add_Round_key Layer

This layer performs a simple XOR operation between the current state and the least significant 64 bits of the round key. The key changes on each turn. Detailed information on key production is given in Section 2.3.

2.2.2. S_Box Layer

The S_Box layer consists of a total of 16 s_boxes running in parallel with each other producing a 4 bit output from the 4 bit input. The output of each value of the S_Box layer of the BORON algorithm is given in Table 1 as hexadecimal.

Table 1. S_box table

X	0	1	2	3	4	5	6	7
S(x)	E	4	B	1	7	9	C	A
X	8	9	A	B	C	D	E	F
S(x)	D	2	0	F	8	5	3	6

2.2.3. Permutation_Layer

The Permutation_Layer has 3 intermediate layers.

2.2.3.1 Block_Shuffle

The Block_Shuffle layer cyclically shifts the 16-bit input by 8 bits to produce 16-bit output. Block blending is shown in Table 2. The 16-bit input is divided into 4-bit blocks first. The least significant 4-bit block ($j = 0$) replaces the third least significant block by shifting 8 bits to the left. In the same way, the second least significant block ($j = 1$) shifts 8 bits to the left, replacing the most significant block.

Table 2. 8 bit left cyclic shift in 16 bit blocks

j	0	1	2	3
B(j)	2	3	0	1

2.2.3.2 Round_Permutation

The 64-bit input of the Round Permutation is first divided into four 16-bit blocks, and cyclic shifting to the left is applied to these 4 blocks in the direction of Table 3.

Table 3. Round_Permutation left cyclic shift values

j	0	1	2	3
r(j)	1	4	7	9

2.2.3.3 XOR_Operation Layer

The XOR Operation Layer performs a simple XOR operation between 16-bit blocks. According to the following equation, four 16-bit output (W'3 W'2 W'1 W'0) is a layer that produces a total of 64 bits of output.

$$W'3 = (W3 \hat{} W2 \hat{} W0) \quad (1)$$
$$W'2 = (W2 \hat{} W0) \quad (2)$$
$$W'1 = (W3 \hat{} W1) \quad (3)$$
$$W'0 = (W3 \hat{} W1 \hat{} W0) \quad (4)$$

" ' " represents outputs, "W" represents 16-bit inputs, and " ^ " indicates XOR operation.

2.3. Key_Schedule (Key Generation)

BORON key generation is very similar to PRESENT. An attack against PRESENT key production has not been published until now. It uses 26 64-bit intermediate keys for encryption. The generation of these keys is performed by inserting a 128 bit or 80 bit user defined master key into the key generation function and extracting the least significant 64 bits of the output. The generated keys are included in the round function, respectively. It should be noted here that the first round key is not produced and the least significant 64 bits of the user-defined master key are used. Therefore, the first produced key of the key generation function will be included in the second round. As a result, the BORON satisfies the first key (K0) of the 26 keys (K0-K25) necessary for encryption from the last 64 bits of the master key and other 25 keys from the key generation function. In this study, key generation function is used from only 80 bit master key.

The key update of the BORON is performed according to the following algorithm. First the Key_Register is shifted cyclically to the left by 13 bits. Then, the most meaningless 4 bits of this key record are updated in the S_box. Finally, 4 bits between the 59th and 63rd bits of this key register are updated by XORing with the 4 bit binary number of that number.

KEY_Register = K79K78...K2K1K0
round_key = K63K62...K2K1K0

1. KEY_Register <<< 13
2. [K3K2K1K0] ←S [K3K2K1K0].
3. [K63K62K61K60K59] ← [K63K62K61K60K59] ^ RCi

<<< n expresses leftward cyclic shift of n bits and RCi expresses the value of the round counter of the i. tour.

3. The Diffie-Hellman Key Exchange Protocol

In this article, Diffie-Hellman Key Exchange protocol is used to let two parties that have no prior knowledge of each other to share secret key over an insecure channel. At the end of the communication both sender and receiver have the same key.

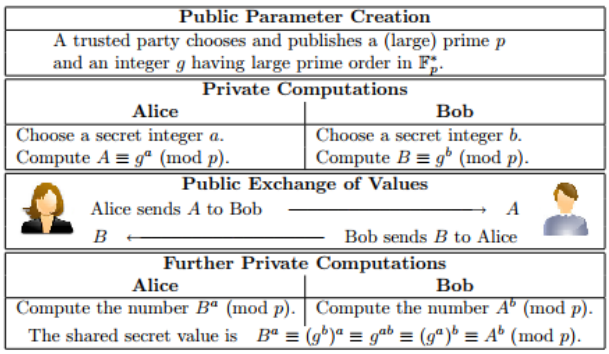

Fig. 3. Diffie–Hellman key exchange [10]

4. Hardware Design

The implementation of the BORON encryption algorithm is first started by designing the hardware in Xilinx Vivado 2017.1 program using the Verilog hardware description language. The hardware in the algorithm are collected under the Boron module. After the design of the cipher hardware, Montgomery modular multiplier is designed in VHDL language to be used in secure key sharing. All the designed hardware are packaged and connected with the ARM processor using AXI bus architecture.

After the hardware design phase is completed, software is designed on the embedded ARM microprocessor in the FPGA to control these hardware and system.

4.1. Boron Module

The Boron module has 64-bit plaintext input. The main input key length is 80-bit selected. This module is designed to perform both encryption and decryption in itself. The 1-bit enc_dec entry specifies whether the module will encrypt or decrypt the input text. The start and done signals used in the Boron module are used to communicate with the microprocessor.

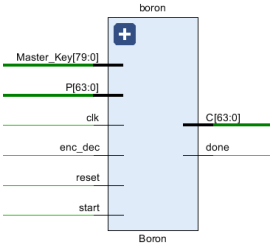

Fig. 4. Boron module RTL schematic

The Boron consists of Round function that repeats itself 25 times. At the same time key production has a repetitive structure. Thanks to the CU finite state machine in it, the Boron module is able to perform its functions in a synchronous manner. It is preferred that all keys be generated and recorded once instead of generating all keys repeatedly in encryption-decryption process in the module. Although this draws the disadvantage of using a lot of space in the first stage, it is thought that the absence of the key generation process repeatedly, when considering a continuously working system, would seriously reduce the total working time, thus minimizing power consumption.

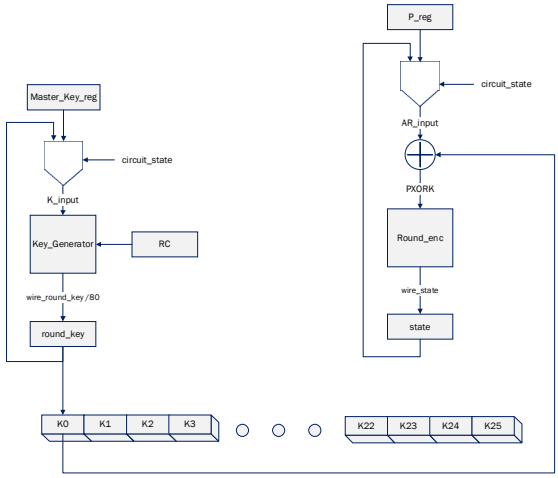

Fig. 5. Boron module working diagram

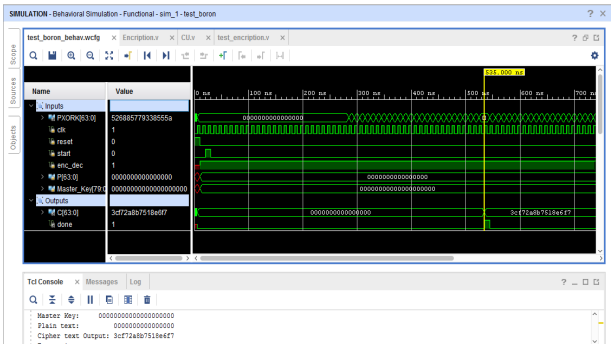

Fig. 6. Boron module behavioural simulation result

4.1.1. CU Module

A finite state machine is designed to manage the Boron module. Five different working states are determined: IDLE, Key_Generation, Enc, Round25, Dec.

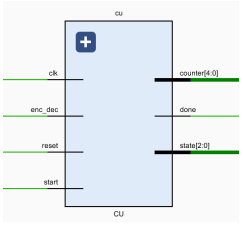

Fig. 7. CU module RTL schematic

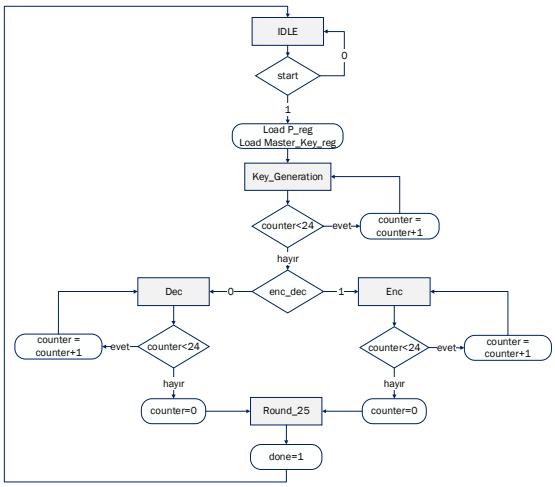

Fig. 8. CU module working diagram

In the state of IDLE, the processor is expected to wait until the start signal. If the start signal arrives, it goes into the Key_Generation state and remains in this state until the counter reaches 24 (starting from 0) to produce the keys needed for our Round function. When the counter is set to 24, Enc or Dec is toggled by looking at enc_dec. Enc status is the encryption process, and Dec is the decryption process. Holding a counter holds 24 rounds in this case, and when the counter is 24, it goes Round25. The Boron module now performs its task and returns to the IDLE state by pulling the done output high. When doing this, the counter is reset and ready for new operations.

4.1.2. Add_round_key Module

The Add_round_key module performs a simple XOR operation between the current state and the least significant 64 bits of the round key.

4.1.3. Round_enc Module

The Boron module has a Round function that repeats itself 25 times. S_Box_layer and Permutation_Layer are implemented in the Round module while the key addition part of the round function is performed in the Add_round_key module.

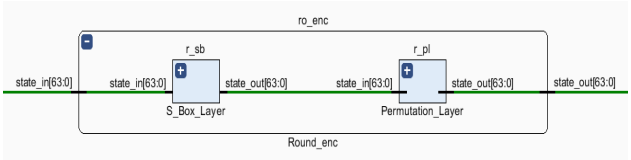

Fig. 9. Round_enc module RTL schematic

4.1.3.1 S_Box_Layer Module

The S_Box module consists of a total of 16 s_boxes operating in parallel with each other producing a 4-bit output from a 4-bit input. The S_box modules perform a non-linear cipher change according to the values given in Table 1. When the module is designed, no reduction is made to the function to be implemented, inputs and outputs are written directly in the switch-case structure.

4.1.3.2 Permutation _Layer Module

This module has 3 intermediate modules connected in series. It combines the functions given in Section 2.2.3.

4.1.4. Round_dec Module

Round_dec designed as the opposite of the Round_enc module. This module will be used for decryption.

4.1.5. key_generator Module

The Boron module needs 26 intermediate keys. In this design, the intermediate keys are stored in the 2080 (26 * 80) - bit Key_Register. After the user-defined 80-bit key is inserted into the key_generator module and a set of mathematical operations are performed, the generated intermediate key is registered to the most significant 80 bits of the Key_Register. In each round, the Key_Register is shifted to the right by 80 bits to ensure that the newly created key remains at the most significant 80 bits. In total, at the end of 26 rounds, the first generated K0 key is placed in the least significant 80 bits of the Key_Register [79:0], and the last generated key K25 is placed in the most significant 80 bits of Key_Register [2079:2000]. If the Add_round_key module is to be used for encryption, it uses those keys that are generated by shifting the Key_Register to 80 bits to the left of each round starting from the least significant 80 bits (K0) of the Key_Register. If it is to be used for decryption, it uses the keys generated by scrolling 80 bits right on each round starting from the most significant 80 bits (K25) of Key_Register.

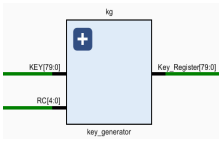

Fig. 10. key_generator module RTL schematic

4.2. Montgomery Modular Multiplier Module

In this project, Montgomery algorithm is preferred to perform modular multiplier operation that will be used in the key exchange protocol. Montgomery Modular Multiplier is designed with VHDL. This module implements $d = a.b.2^{-80} \bmod c$ operation for 80-bit data. The modular product is designed in this way as hardware and then exponentiation is done in ARM microprocessor according to Fig. 13.

4.3. Block Design

A custom AXI IP block is created in Vivado and the Boron module is packaged in Boron IP with integrated Verilog codes. The same process is also done for the Montgomery modular multiplier written in VHDL language. The Zynq-7000 hardware platform is also included in the block design, so the AXI communication protocol connections between the Boron IP, Montgomery IP and the ARM processor have been made automatically by the program. In this project, the Zynq-7000 platform's ARM microprocessor system is used to control the Boron cryptographic hardware along with the C code written for the microprocessor.

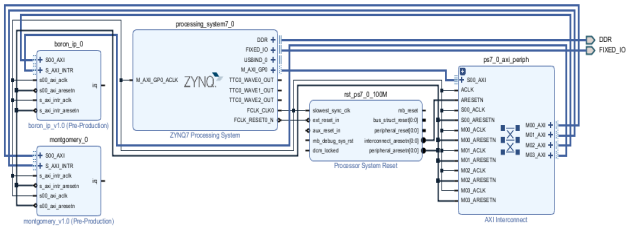

Fig. 11. Block design of hardware design

5. Software Design

A microprocessor is used to control the encryption / decryption equipment of the Boron and to provide inter-hardware data flow. Instead of embedding MicroBlaze provided by Xilinx on the FPGA, the ARM processor, which is already available in hardware, is preferred. This avoids unnecessary space usage on the FPGA for the processor. Text, key and which operation will be done are said to the Boron hardware with the C code written in the SDK interface. Initially reset signal resets the Boron module registers. Inputs with the start sign are transferred to the module. The start sign is pulled to low level after a certain period of time so that the module does not continue its operations repeatedly. When the Boron module fulfills its tasks, it writes the output it produces to the terminal via UART communication.

Firstly, encryption of a text is performed according to the scenario. The result is stored in registers. Then, the encrypted text is transferred the module again and this time the decryption is performed. If the resulting text overlaps the input text at the beginning, it prints to the terminal via the UART where the design is successfully performed.

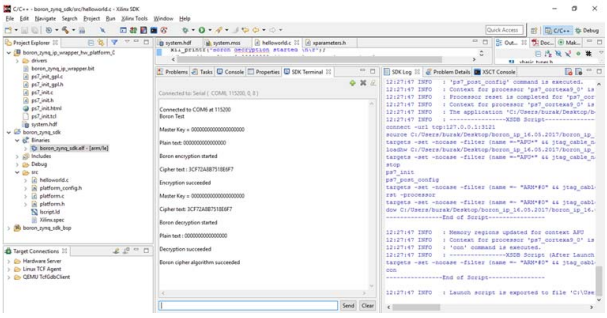

Fig. 12. Outputs of design in SDK interface

After testing that the encryption decryption works correctly, Montgomery custom IP is tested. Finally, Montgomery exponentiation algorithm is used to compute Alice and Bob's public values.

INPUT: $m = (m_{l-1} \cdots m_0)_b$, $R = b^l$, $m' = -m^{-1} \bmod b$, $e = (e_t \cdots e_0)_2$ with $e_t = 1$, and an integer x, $1 \le x < m$.
OUTPUT: $x^e \bmod m$.

1. $\widetilde{x} \leftarrow \text{Mont}(x, R^2 \bmod m)$, $A \leftarrow R \bmod m$. ($R \bmod m$ and $R^2 \bmod m$ may be provided as inputs.)
2. For i from t down to 0 do the following:
   2.1 $A \leftarrow \text{Mont}(A, A)$.
   2.2 If $e_i = 1$ then $A \leftarrow \text{Mont}(A, \widetilde{x})$.
3. $A \leftarrow \text{Mont}(A, 1)$.
4. Return(A).

Fig. 13. Montgomery exponentiation algorithm [11]

6. Conclusions

In this article, it is aimed to design a lightweight crypto system with low power-area consumption for low-scale embedded systems. Since software-only implementation is very slow and hardware-only implementation consumes too much area, SoC-based implementation is preferred.

The Boron module can encrypt or decrypt a 64-bit text block with 379,822 Mbps throughput (25 clock cycles) on Zedboard. It uses 418 LUTs, 125 mW on the FPGA while Montgomery IP requires 755 LUTs. If we keep the intermediate keys, LUT usage increases to 1723 for Boron IP. Compared to lightweight encryption algorithms, these numbers are little bit more but this is normal as the design has also decryption ability. These results achieve the targeted low floor space and low power consumption in the designed hardware.

It has been determined that improvements can be made to design a more sophisticated system by investigating how precautions can be taken against potential side channel attacks in advanced studies.

Resource	Utilization	Available	Utilization...
LUT	418	53200	0.79
FF	265	106400	0.25

Fig. 14. Utilization report of our design

7. References

[1] GfK. (2017, May 18th). *Tech Trends 2017*. Available: https://cdn2.hubspot.net/hubfs/2405078/Landing_Pages_PDF/Tech%20Trends/Global_201703_Tech_Trends_2017_Report.pdf

[2] A. Joubert. (2017, July 14th). Available: https://www.azuruw.com/articles/2017/07/14/will-the-internet-of-things-become-a-playground-for-cyber-criminals

[3] Advanced Encryption Standard (AES), Federal Information Processing Standards Publication 197, November 2001.

[4] W. C. Barker, "Recommendation for the triple data encryption algorithm (TDEA) block cipher", U.S. Dept. of Commerce, Technology Administration, National Institute of Standards and Technology, 2004.

[5] E. Nikolaos et al, "Lifelogging in smart environments: challenges and security threats." Communications (ICC), 2012 IEEE International Conference on. IEEE, 2012.

[6] K. Finkenzeller, "RFID Handbook: Fundamentals and Applications in Contactless Smart Cards and Identification" Hoboken, NJ, USA: Wiley, 2003.

[7] G. BANSOD, N. PISHAROTY, A. PATIL3, "BORON: an ultra lightweight and low power encryption design for pervasive computing", Springer, 2017.

[8] A. Bogdanov et al., "PRESENT—An ultra-lightweight block cipher", Springer-Verlag, Berlin, Germany, 2007, pp. 450–466.

[9] T. Okabe, "FPGA Implementation and Evaluation of lightweight block cipher – BORON", Tokyo Metropolitan Industrial Technology Research Institute, 2017.

[10] Nan Li, "Research on Diffie-Hellman Key Exchange Protocol", Proceedings: 2nd Computer Engineering and Technology (ICCET), 2010.

[11] A. Menezes et al., "Handbook of Applied Cryptography", CRC Press, 1996.