COMMUNICATIONS IN SCIENCE AND TECHNOLOGY
Homepage: cst.kipmi.or.id

Block cipher four implementation on field programmable gate array

Yusuf Kurniawan[a,*], Muhammad Adli Rizqulloh[b]

[a]*Electrical Engineering, School of Electrical Engineering and Informatics, Institut Teknologi Bandung, Bandung City, 40132, Indonesia*
[b]*Department of Electrical Engineering Education, Universitas Pendidikan Indonesia, Bandung City, 40154, Indonesia*

Article history:
Received: 9 June 2020 / Received in revised form: 12 August 2020 / Accepted: 23 August 2020

Abstract

Block ciphers are used to protect data in information systems from being leaked to unauthorized people. One of many block cipher algorithms developed by Indonesian researchers is the BCF (Block Cipher-Four) - a block cipher with 128-bit input/output that can accept 128-bit, 192-bit, or 256-bit keys. The BCF algorithm can be used in embedded systems that require fast BCF implementation. In this study, the design and implementation of the BCF engine were carried out on the FPGA DE2. It is the first research on BCF implementation in FPGA. The operations of the BCF machine were controlled by Nios II as the host processor. Our experiments showed that the BCF engine could compute 2,847 times faster than a BFC implementation using only Nios II / e. Our contribution presents the description of new block cipher BCF and the first implementation of it on FPGA using an efficient method.

Keywords: BCF; FPGA; NIOS II; Cryptography

1. Introduction

Block cipher is one of the cryptographic components used to protect information. Information can be in the internet network, financial system, military, and IoT (internet of things). IoT is a network of interconnected objects in various forms such as wireless sensor networks, electrical, electronic, mechanical devices, and their interaction with computer data via the internet [5]. In the IoT period, embedded devices were connected to the internet. The advent of IoT has put telecommunications and embedded systems at risk [6]. BCF is an encryption algorithm based on AES [13], Camellia [14], TwoFish [15], and Khazad [12]. It has 128 bits of input /output and 128, 192, and 256 bits keys. BCF is an encryption algorithm designed by Indonesian researchers [1]. This algorithm has an advantage over AES: The key schedule in BCF is more secure than AES because the main key is very difficult to find even when all sub-keys of BCF have been found. The SBox from BCF changes dependent on the key, while the SBox from AES does not change. Thus, BCF is safer than AES.

There are two types of BCF keys: master key and sub-keys. A master key is processed by key schedule function becoming the sub-keys. Every sub-key is used to encrypt or decrypt partial data in every round. Encryption is a process to convert plaintext to be cipher text and decryption converts cipher text to be plaintext.

* Corresponding author.
Email: yusufk@stei.itb.ac.id

Cryptanalysis is used to crack the key of a block cipher in an unusual way or test the security of a cryptographic algorithm that has been created. Correlation power analysis, for instance, tries to find all of the sub-keys using the correlation between the hamming weights and the power used in the embedded device when calculating the encryption algorithm [7].

The hardware implementation is very important in terms of a performance and security, especially as a countermeasure against timing attacks [8] in particular and as side-channel attacks in general. This paper aims to introduce the BCF algorithm implemented in FPGA with an efficient method. This paper proposes a hardware architecture of the BCF algorithm as a co-host processor (encryption engine accelerator). This architecture was written in Verilog and tested on the Altera Cyclone IV EP4CE115F29 [9] using NIOS as the host processor. We compared the results with AES, Camellia, and TDEA data taken from SASEBO [10]. Moreover, we compared the BCF hardware accelerator with software implementation enabling us to measure how fast the BCF encryption engine accelerator computed, compared to software.

2. Materials and Methods

2.1. BCF Algorithm

BCF uses the Feistel structure [11], in contrast to AES which uses the SPN structure. The SPN structure requires fewer rounds than does Feistel to achieve the same diffusion rate. The advantage of using the Feistel structure over SPN is related to the use of the same structure for the encryption and decryption

processes so that it will require few memories in the implementation. SPN requires two different algorithms for encryption and decryption.

The BCF algorithm has two main components: scheduling part and randomization part. Key Schedule is performed to generate sub keys and randomization is performed to encrypt or decrypt data using sub keys generated by key scheduling. The number of rounds at the randomization stage depends on the length of the key in which 128-bit keys are used in the randomization of 15 rounds, 192-bit keys require 16 rounds and 256-bit keys for 18 rounds. In each round, the F0 function is applied. This function uses sub keys to manipulate the input data for each round.

The main features of the BCF algorithm are:

1. The input and output data are 128 bits (plain text and cipher text) respectively.
2. The length of the master key has 3 variants: 128, 192 and 256 bits.
3. Key scheduling is done in 8 rounds using the F0 function.
4. The number of rounds at the randomizing stage (for encryption or decryption) depends on the length of the key.

The key schedule stage is carried out at the beginning to generate sub-keys for the randomizing stage, but in this paper we will begin by explaining the randomizing stage.

2.2. BCF Encryption

BCF uses the Feistel structure as in the Twofish algorithm, so it can use the same algorithm for encryption and decryption. BCF has 128-bit input / output. The pseudo code of the BCF encryption algorithm is presented as follows.

```
XL = XL ^ KW1
XR = XR ^ KW2
XL = XR
XR = XL ^ F0(K1,XR)
XL = XR
XR = XL ^ F0(K2,XR)
XL = XR
XR = XL ^ F0(K3,XR)
XL = XR
XR = XL ^ F0(K4,XR)
XL = XR
XR = XL ^ F0(K5,XR)
XL = XR
XR = XL ^ F0(K6,XR)
XL = XR
XR = XL ^ F0(K7,XR)
XL = XR
XR = XL ^ F0(K8,XR)
XL = XR
XR = XL ^ F0(K9,XR)
XL = XR
XR = XL ^ F0(K10,XR)
XL = XR
XR = XL ^ F0(K11,XR)
XL = XR
XR = XL ^ F0(K12,XR)
XL = XR
XR = XL ^ F0(K13,XR)
XL = XR
XR = XL ^ F0(K14,XR)
XL = XR
XR = XL ^ F0(K15,XR)
XL = XR
XR = XL ^ F0(K16,XR)
XL = XR
XR = XL ^ F0(K17,XR)
XR = XR
XL = XL ^ F0(K18,XR)
XL = XL ^ KW3
XR = XR ^ KW4
```

Figure 1. shows the BCF structure that can be used for both encryption and decryption.

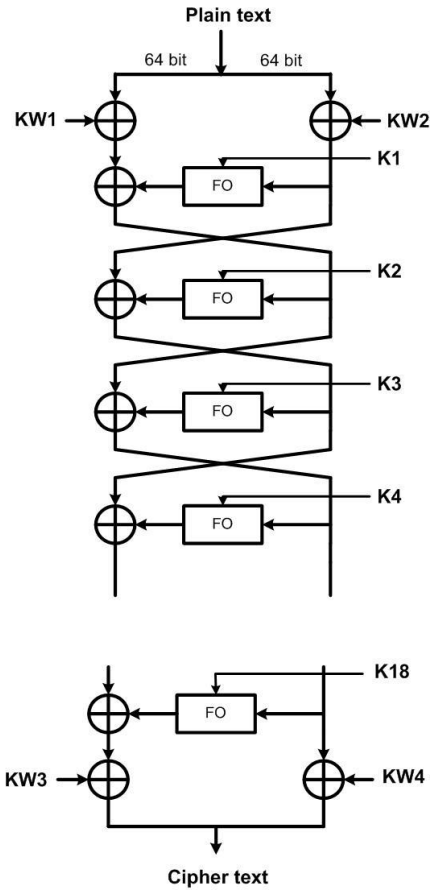

Fig. 1 BCF encryption

In the BCF algorithm, there is an FO function that has an input of two data words x and two words of the k sub-key and produces two output words of y, where 1 word is 32 bits. This function is the heart of BCF encryption/decryption. For a note, 1 word is 32 bits.

$$y = FO(x,k) = P(S_i(x)) \oplus k \quad (1)$$

SBox has 1 byte input / output. Because each x consists of 8 bytes, there are 8 SBox operations for each input $x\{ x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7\}$. There are 4 SBox-es used in BCF. The Substitution Boxes (in hexadecimal) are shown in (Tables 1-4).

Table 5. shows the algorithms used to select BCF SBox before encryption/decryption.

Table 1. BCF substitution box 1

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	b4	b7	67	f3	0b	94	52	68	37	3e	ef	d7	eb	8f	55	93
1	2b	a5	02	be	21	20	98	69	8d	05	73	af	a7	bb	dc	ac
2	54	70	95	77	c1	06	22	44	b0	c9	76	b5	2f	27	2d	32
3	72	5b	64	c0	3a	3c	49	de	57	28	3d	0a	f4	e7	71	7a
4	d1	8e	f0	43	c8	23	7f	aa	df	16	ba	5e	18	1a	1d	7c
5	bd	d5	85	ab	56	8c	9f	fd	4c	ca	9d	c7	ed	b9	25	cc
6	75	ee	ae	fc	4b	03	3f	a3	e3	7d	e5	d3	d4	10	e0	36
7	34	d8	3b	82	6e	89	35	01	91	79	d0	da	5c	4	cd	fe
8	84	6b	e4	1b	6c	9b	81	Cf	2c	46	a8	c5	07	26	e9	51
9	c4	d9	0f	33	4d	41	bf	61	4a	a0	a6	b1	c2	ea	66	c6
a	83	f6	e2	40	1f	5d	7b	dd	f9	b3	a4	15	8b	6a	45	09
b	e1	4e	11	50	b6	58	db	08	cb	7e	a2	5a	f1	ad	87	ff
c	53	47	13	80	86	c3	1c	5f	a9	59	63	e6	fa	30	42	f5
d	0d	38	2a	9a	f7	90	65	ce	78	d2	9c	bc	1e	0c	17	ec
e	88	d6	6f	62	0e	6d	99	9e	8a	31	48	19	4f	00	74	fb
f	29	b8	2e	a1	39	60	96	12	97	f8	b2	24	e8	92	f2	14

Table 2. BCF substitution box 2

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	40	a5	3e	ee	28	1e	51	21	60	db	4c	59	3c	c8	8f	77
1	5a	43	2a	d2	5b	ce	a1	e9	b1	47	e3	8a	46	e1	e7	89
2	fd	bc	f9	c0	a0	f4	09	3d	52	5d	fe	a6	67	cb	ec	97
3	18	4b	6a	61	b7	c1	9e	24	4f	e5	01	03	29	08	b9	06
4	f0	a7	e6	cc	39	1d	7f	15	57	f3	82	99	70	6e	9c	58
5	d6	2c	d9	cd	a3	4d	75	48	74	2f	e2	6d	d7	12	b0	37
6	80	7e	86	79	0e	71	d0	34	0a	ac	42	94	b8	aa	56	dd
7	84	55	38	fb	1c	dc	33	de	6c	c2	a2	d3	e4	66	ae	a4
8	9f	26	22	7	7c	f8	3b	f2	f6	1f	96	f5	85	c9	c6	ab
9	8b	45	7b	e0	2e	50	9b	d8	d1	5f	c5	65	0d	88	14	27
a	d5	20	c7	fc	44	fa	3f	62	35	a9	63	2d	49	69	19	0f
b	13	95	90	72	ad	00	31	17	0c	64	11	b4	16	53	9a	04
c	8d	cf	91	bf	d4	a8	ea	8c	c3	54	af	93	3a	ba	1a	eb
d	ef	7d	8e	c4	6b	0b	5c	25	bd	4a	1b	68	87	b6	df	6f
e	83	b3	be	f7	78	81	ed	4e	bb	2b	36	b5	76	30	9d	f1
f	7a	92	e8	23	da	ca	41	5e	10	98	ff	32	b2	73	05	02

Table 3. BCF substitution box 3

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	47	ec	a1	80	49	10	d1	0b	94	50	5e	a7	23	45	b0	38
1	de	f5	6e	d5	54	c0	8e	34	0e	ba	16	44	e2	72	aa	e5
2	26	cb	fe	60	3	bf	dd	56	57	e4	91	8c	19	65	3b	1
3	90	7b	f9	24	0c	c5	61	b7	f7	e1	37	fd	85	7e	9	2c
4	c9	ac	66	d6	40	a9	42	4	5a	6b	1c	ce	fa	af	db	b5
5	83	b3	a6	e7	7a	e0	cf	27	6a	ef	1e	b8	6	18	2f	63
6	b9	82	76	28	f8	ed	71	fb	70	5	c8	88	e6	4e	e3	cc
7	fc	c4	67	95	78	13	9c	2e	74	68	84	31	f2	58	ae	3d
8	df	87	7c	2	ff	79	86	e9	2d	c2	52	5f	ad	30	8a	99
9	25	41	22	8d	1d	20	8f	97	14	77	c7	9a	f0	2b	1a	3e
a	89	9e	7	8	3a	5b	4a	ca	9f	12	c1	59	0a	55	81	a8
b	21	96	73	46	f1	c6	bd	33	0	62	1f	32	b1	7f	bc	4d
c	d2	6f	d9	d3	b2	f6	36	cd	75	dc	64	7d	6c	93	bb	a4
d	a5	f4	53	1b	35	5d	a0	a2	4b	4f	43	51	48	e8	0f	5c
e	c3	b4	da	ee	2a	d0	39	9d	98	ea	d7	a3	ab	29	0d	eb
f	b6	17	3c	4c	f3	3f	6d	15	be	92	8b	9b	11	d4	69	d8

Table 4. BCF substitution box 4

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	2f	5a	c2	d4	9e	0b	eb	b4	6	43	d5	50	59	df	65	4b
1	3e	99	5f	27	10	cb	42	a8	cd	c3	2c	95	e6	24	dc	b5
2	6a	aa	a6	46	c0	19	a7	73	84	ff	fa	9f	31	7f	3f	e9
3	b0	52	1b	93	3a	b3	9a	d8	fb	88	4d	d3	f3	b7	91	5
4	13	5b	d7	22	40	f4	61	75	e3	48	74	55	e5	d0	47	2d
5	11	f0	33	29	ae	9c	e4	90	68	e8	d1	ef	c5	32	6c	20
6	8b	63	d2	4f	3	d9	72	76	1d	bb	2b	1	4	ab	1e	a0
7	96	66	80	25	85	78	34	6f	e0	c9	4a	8d	7e	c6	b6	e2
8	62	0e	9d	64	82	fd	5e	71	54	cc	a4	f8	b8	94	53	30
9	35	de	ce	2a	3c	21	fc	4e	be	12	fe	dd	db	6b	c4	2
a	51	ad	f7	26	8c	15	14	17	af	0f	bf	7b	39	a3	e1	6d
b	f2	c8	cf	f9	1c	23	b2	7c	87	44	18	f1	0a	b9	79	ac
c	f6	0d	ee	98	9b	97	36	da	1f	d6	81	ca	58	7d	8a	83
d	a9	c1	f5	bc	5d	89	77	6e	2e	b1	5c	8e	0c	28	9	1a
e	7	ea	e7	56	37	7a	41	70	57	a5	4c	67	bd	c7	60	3d
f	0	38	92	a2	69	8f	ed	ba	45	3b	a1	8	ec	16	49	86

Table 5. The algorithm for selecting SBox

Round	Key	Ramdomize
1-4	K_A	$S[i] = ((K >> (2i + 8(r-1))) \& 03_x$
5-8	K_B	$S[i] = ((K >> (2i + 8(r-5))) \& 03_x$
9-12	K_A	$S[i] = ((K >> (2i + 8(r-9))) \& 03_x$
13-16	K_B	$S[i] = ((K >> (2i + 8(r-13))) \& 03_x$
17-18	K_A	$S[i] = ((K >> (2i + 8(r-17))) \& 03_x$

P is the product between the matrix M and the input x with an aim to obtain optimal diffusion.

$$b = P(x) = M \cdot x \quad (2)$$

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} = \begin{bmatrix} 03_x & 01_x & 05_x & 04_x & 0B_x & 07_x & 06_x & 08_x \\ 01_x & 03_x & 04_x & 05_x & 07_x & 0B_x & 08_x & 06_x \\ 05_x & 04_x & 03_x & 01_x & 06_x & 08_x & 0B_x & 07_x \\ 04_x & 05_x & 01_x & 03_x & 08_x & 06_x & 07_x & 0B_x \\ 0B_x & 07_x & 06_x & 08_x & 03_x & 01_x & 05_x & 04_x \\ 07_x & 0B_x & 08_x & 06_x & 01_x & 03_x & 04_x & 05_x \\ 06_x & 08_x & 0B_x & 07_x & 05_x & 04_x & 03_x & 01_x \\ 08_x & 06_x & 07_x & 0B_x & 04_x & 05_x & 01_x & 03_x \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix}$$

The input is x = { x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7} and the output is P(x) = b = {b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7}. BCF uses the same irreducible polynomials used by AES, m(x).

$$m(x) = x^8 + x^4 + x^3 + x + 1 \quad (3)$$

2.3. BCF Key Expansion

The BCF keyschedule (key expansion) algorithm has a key input of 128 bits (16 bytes or 4 words), and performs a Key Expansion to generate some sub-keys. The Key Expansion produces a total of 17 sub-keys, 15 sub-keys for the regular round (K 0 , K 1 , ..., K 14) and 4 sub-keys for whitening keys (KW 1, KW 2, KW 3, KW 4). If the primary key is 192 bits or 256 bits, then we perform XOR operation between the left side and the right side of the primary key so that it still generates a key of 128 bits to be included in the key schedule.

At the beginning of the key schedule, the intermediate keys:K_A, K_B, ..., K_G are generated. From these intermediate keys, all sub-keys required for the encryption and decryption processes are generated. Figure 2 shows the beginning of the key expansion to generate K_A, K_B, K_C and K_D.

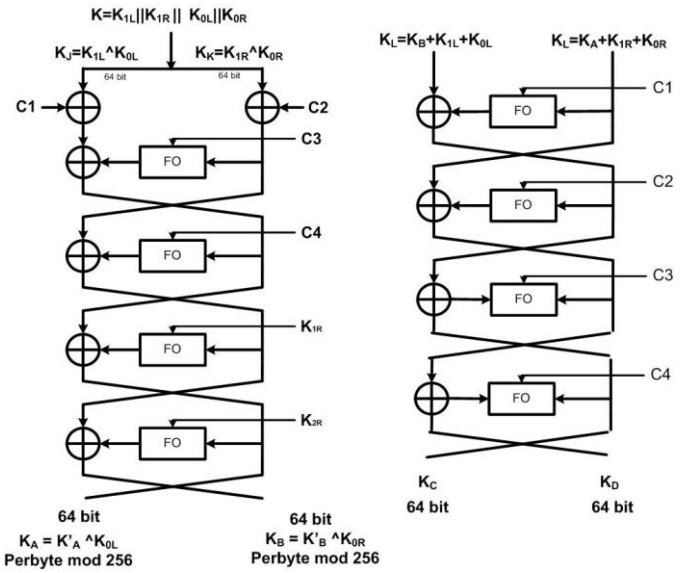

Fig. 2 BCF key expansion

Table 6 describes the complete key expansion process. The table is connected to the figure 2 as key expansion process.

Table 6. BCF key expansion

Key	Key Function
K_E	$(K_A \cap K_B) \oplus K_C$
K_F	$(K_A \cup K_C) \oplus K_D$
K_G	$(K_E \cup K_F) \oplus K_A \oplus K_B$
K_{w1}	$(K_E \cap K_F) \oplus K_C$
K_{w2}	$K_C \oplus K_D \oplus K_E$
K_1	$K_{w1} \oplus K_{w2} \oplus$ 0x00007a0000000000
K_2	$K_1 \oplus K_C \oplus K_D$
K_3	$(K_2 \cup$ 0xff00000000000000) $\oplus K_E$
K_4	$K_3 \oplus K_C \oplus K_D$
K_5	$(K_4 \cup K_F) \oplus K_G$
K_6	$(C_1 \cap K_E) \oplus K_5$
K_7	$(K_G \cup K_6) \oplus K_F$
K_8	$(K_7 \cap K_6) \oplus K_G$
K_9	$(K_8 \cap K_F) \oplus K_G$
K_{10}	$(K_9 \cup K_C) \oplus K_8$
K_{11}	$K_8 \oplus K_9 \oplus K_{10}$
K_{12}	$(K_{10} + K_{11})$ | 8 bit
K_{13}	$(K_{11} + K_{12})$ | 32 bit
K_{14}	$(K13_{32l} <<<1)$ | | $(K13_{32r} <<<1) \oplus K11$
K_{15}	$(K_{12} \cap K_{14}) \oplus K_{13}$
K_{16}	$(K_{14} \cup K_{15}) \oplus K_{12}$
K_{17}	$(K_{14} \cup K_{15})$
K_{18}	$(K_{15} \cup K_{16}) \oplus K_{17}$
KW_3	$K_{13} \oplus K_{14}$

2.4. BCF Decrytpion

The BCF decryption (Figure 3) procedure can be performed in the same way as encryption, but with the sub-key order reversed. More details are shown in the following pseudo code.

$X_R = X_R \wedge KW_4$
$X_L = X_L \wedge KW_3$
$X_L = X_L \wedge F0(K_{18}, X_R)$
$X_R = X_L$
$X_R = X_L \wedge F0(K_{17}, X_R)$
$X_L = X_R$
$X_R = X_L \wedge F0(K_{16}, X_R)$
$X_L = X_R$
$X_R = X_L \wedge F0(K_{15}, X_R)$
$X_L = X_R$
$X_R = X_L \wedge F0(K_{14}, X_R)$
$X_L = X_R$
$X_R = X_L \wedge F0(K_{13}, X_R)$
$X_L = X_R$
$X_R = X_L \wedge F0(K_{12}, X_R)$
$X_L = X_R$
$X_R = X_L \wedge F0(K_{11}, X_R)$
$X_L = X_R$
$X_R = X_L \wedge F0(K_{10}, X_R)$
$X_L = X_R$
$X_R = X_L \wedge F0(K_9, X_R)$
$X_L = X_R$
$X_R = X_L \wedge F0(K_8, X_R)$
$X_L = X_R$

$X_R = X_L \wedge F0(K_7, X_R)$
$X_L = X_R$
$X_R = X_L \wedge F0(K_6, X_R)$
$X_L = X_R$
$X_R = X_L \wedge F0(K_5, X_R)$
$X_L = X_R$
$X_R = X_L \wedge F0(K_4, X_R)$
$X_L = X_R$
$X_R = X_L \wedge F0(K_3, X_R)$
$X_L = X_R$
$X_R = X_L \wedge F0(K_2, X_R)$
$X_L = X_R$
$X_R = X_L \wedge F0(K_1, X_R)$
$X_L = X_R$
$X_R = X_R \wedge KW_2$
$X_L = X_L \wedge KW_1$

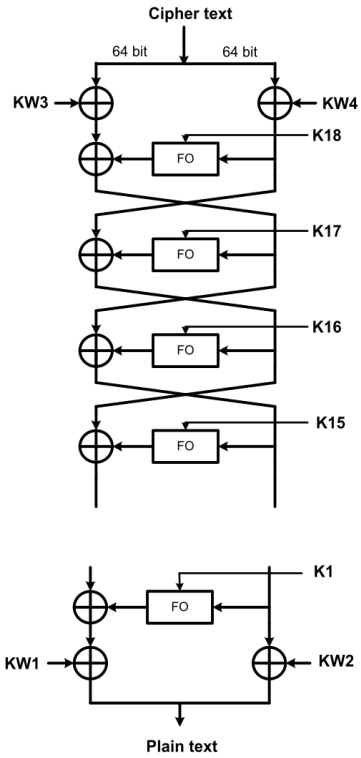

Fig. 3 BCF decryption

2.5. BCF Core IP Design

IP BCF core design is implemented in Verilog HDL language, top down method. The design begins by defining the system, making the architecture first, and designing the supporting modules. The IP BCF core symbol and pin out are shown in Figure 4. Table 7 describes the function of each pin.

Figure 5 shows the general architecture of BCF. Key_len pin out functions to set the number of rounds and the number of rounds depends on the size of the key. For a key with a size of 128 bits, 192 bits, 256 bits, it takes 15, 16, and 18 rounds, respectively. The decrypt pin out determines whether BCF_Core will encrypt or decrypt. The core of the BCF Engine contains encryption, decryption and keyschedule. Figure 6 illustrates the core algorithm for the BCF engine.

Table 7. Core pinout

Port Name	Port Width (bit)	Direction	Description
clk	1	Input	System clock
rst	1	Input	Reset, active high
sel	1	Input	0: Reset BCF Core 1: Active BCF Core
decrypt	1	Input	0: Decryption 1: Encryption
key_len	2	Input	00 : 128 bits 01 : 192 bits 10 : 256 bits
key	256	Input	BCF key
din	128	Input	Data input
dout	128	Output	Data output
done	1	output	0 : BCF Process Done 1 : BCF Processing

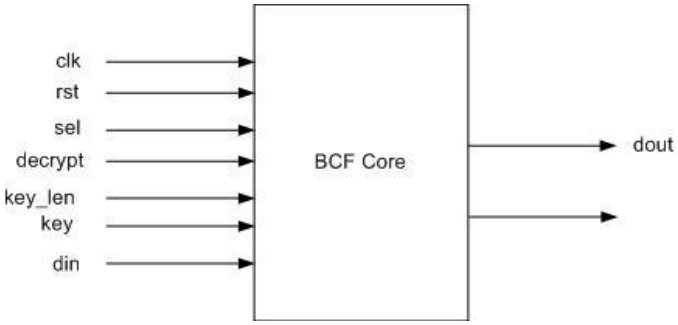

Fig. 4 Core Pin out

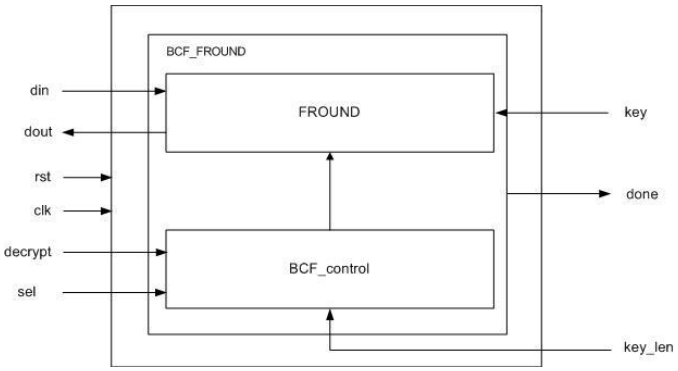

Fig. 5 BCF general architecture

This core architecture was used interchangeably for randomizing and key schedule, resulting in large latency. For the encryption and decryption process, the BCF engine required 316 clocks.

Figure 7 shows the BCF timing diagram. The system was controlled by clk.

To implement BCF, we needed a FSM (Finite State Machine). The BCF FSM is shown in Figure 8.

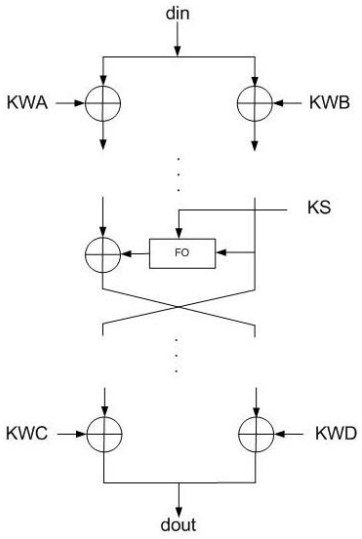

Fig. 6 BCF engine core

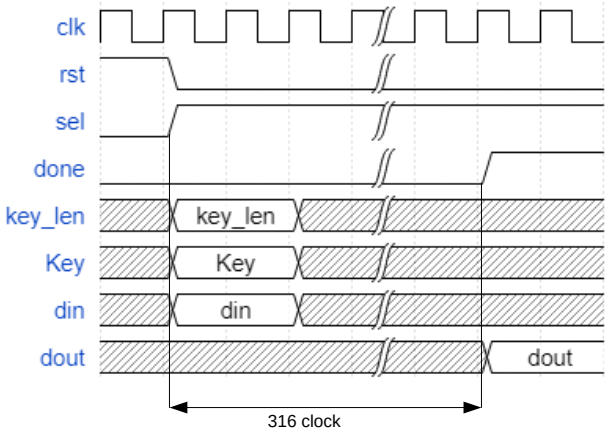

Fig. 7 BCF Timing diagram

Figure 9 and Figure 10 illustrate the keys schedule (key expansion) architecture and the FO module, respectively. One Core FO was used interchangeably in encryption, decryption, and key schedule. The advantage of this design is to use a small area but it has the disadvantage of having a large latency.

The further explanation of FSM in Figure 8 and FO Core in Figure 9 is described in more detailed in Table 14 and Table 15 (appendix).

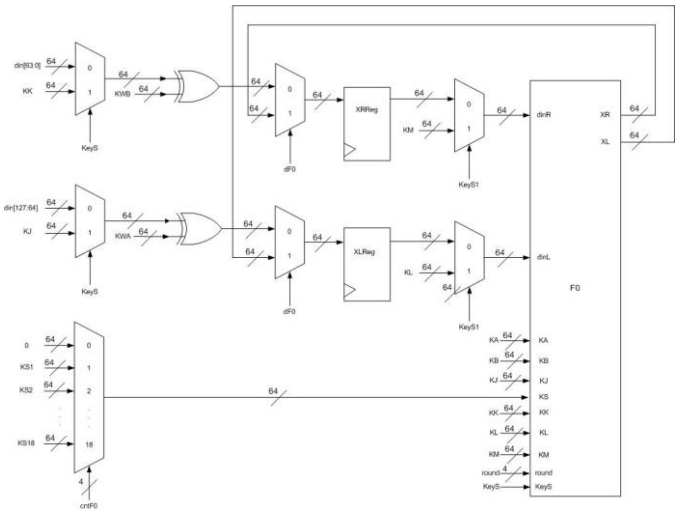

Fig. 9 FO core

Subbyte BCF operations were implemented using LUTs for the ease of design and minimization of critical paths [4].

The description of FO module pins is described in Table 16 (appendix).

The search for the substitution box number was obtained from this equation :

$$si = ((K >> ((2*i) + (8*r))) \& 8'h03) \wedge ((((K >> (8*r + 2)) \& 8'h03) \wedge ((K >> (8*r + 4)) \& 8'h03) \wedge ((K >> (8*r + 6)) \& 8'h03)) \& zero) \quad (4)$$

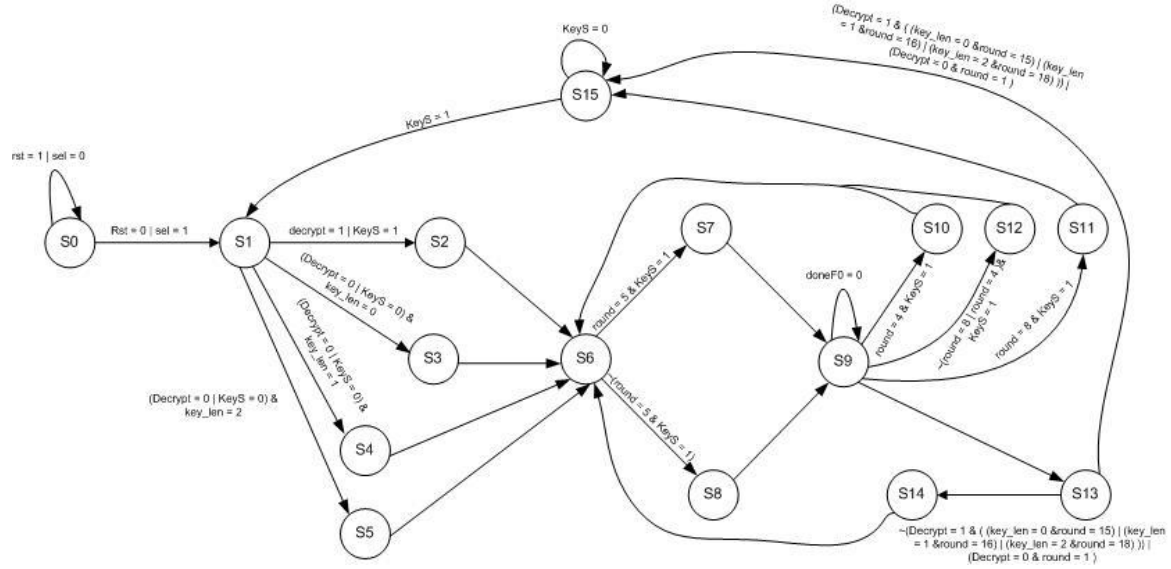

Fig. 8 BCF FSM

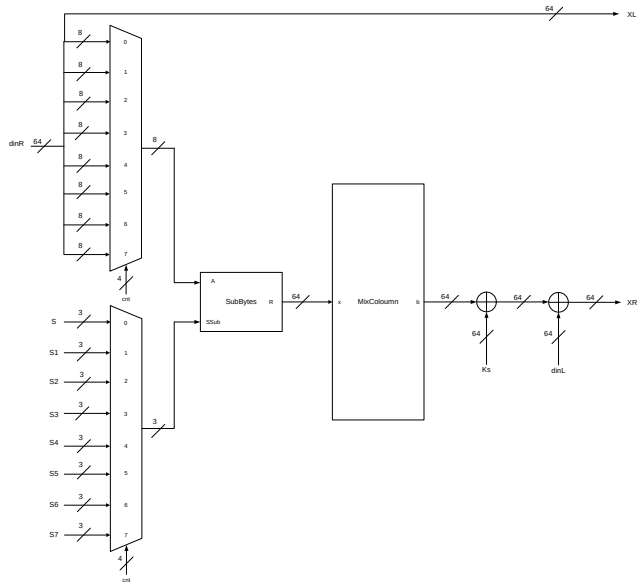

Fig 10 F0 module

The MixColumn operation used a systolic array architecture (Figure 11). The architecture used only eight processing elements in MixColumn and one processing element in Subbyte processing. The tradeoff of this architecture was the latency from one clock to eight clocks.

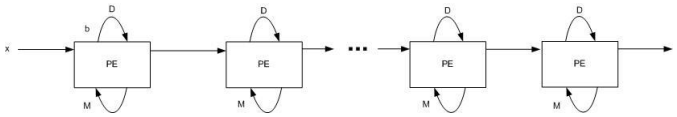

Fig. 11 Systolic array diagram block edge map

Figure 12 shows the Processing Element Design. If Input data is x and multiplier number in mixcolumn is M, then the result of polynomial multiplication between x and M is b.

The design of processing elements is implemented using the architecture as shown in Figure 13.

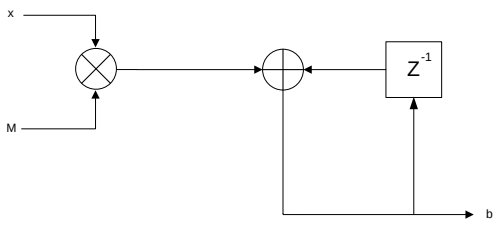

Fig. 12 Processing element design

Table 8. Processing element pinout description

Port Name	Data Length (bit)	Direction	Description
A	8	Input	Data input for polynomial multiplication in mixcolumn.
B	8	Input	Data input for polynomial multiplication in mixcolumn.
R	8	Output	Data output for polynomial multiplication in mixcolumn A * B.

Table 8 shows some descriptions of the Processing Element pinouts. Pinout R contains the result of polynomial multiplication between data from pinout A and B. Xtime algorithm was used for polynomial multiplication in mixcolumn operation. For efficiency in Mixcolumn operation, shift and xor operation was applied [3]. Figure 14 depicts the architecture of xtime algorithm implemented in mixcolumn.

Table 9 shows some descriptions of the XTime pinouts. Xtime architecture used in this design had one input data and eight output data. It was purposely to enable the polynomial multiplication to be completed in one clock.

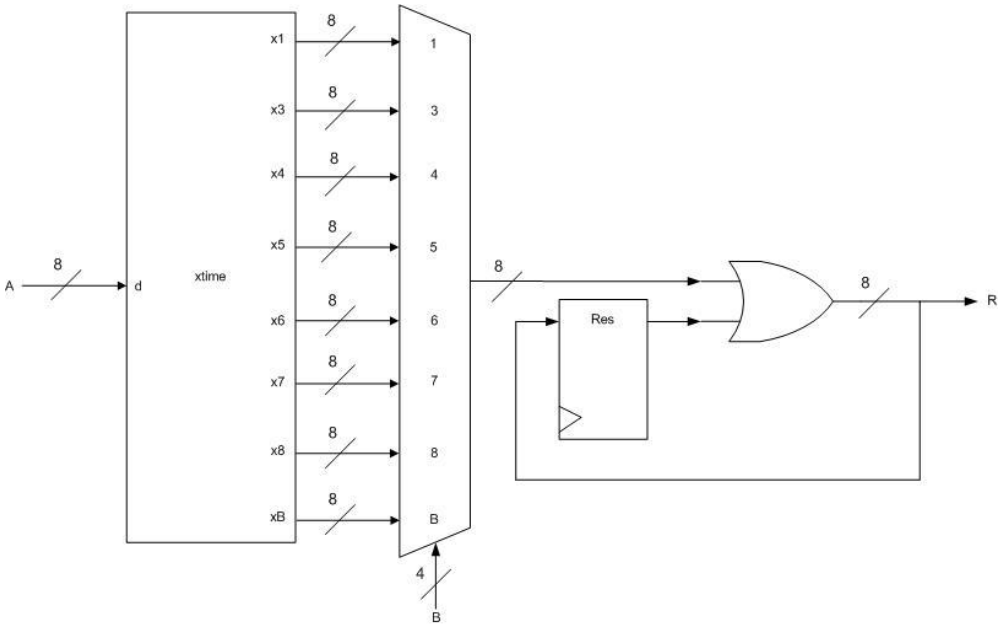

Fig. 13 Processing element architecture

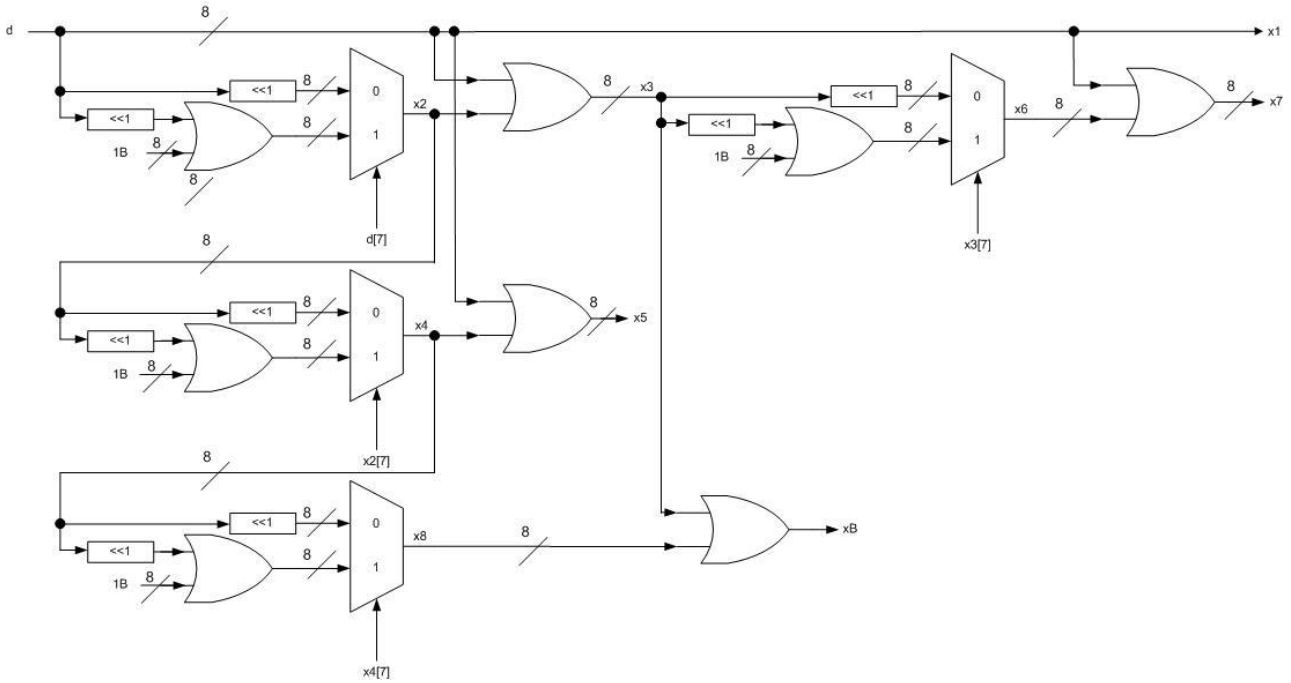

Fig. 14 Xtime architecture

Table 9. Xtime pinout description

Port Name	Data Length (bit)	Direction	Description
d	8	Input	Data input xtime function.
x1	8	Output	Data output for d multiple by 1_x.
x3	8	Output	Data output for d multiple by 3_x.
x4	8	Output	Data output for d multiple by 4_x.
x5	8	Output	Data output for d multiple by 5_x.
x6	8	Output	Data output for d multiple by 6_x.
x7	8	Output	Data output for d multiple by 7_x.
x8	8	Output	Data output for d multiple by 8_x.
xB	8	Output	Data output for d multiple by B_x.

Table 10. Synthesis comparation

Algorithm	Logic Element	Register	FMax (MHz)	Key length (bit)	Throughput (Bps) at 50 MHz
BCFV1	5790	729	54.7	256	7594937
SASEBO AES	5122	396	84.84	128	123076923
SASEBO Camellia	4251	397	59.36	128	55172413
SASEBO TDEA	1017	256	74.4	192	38461538

2.6. BCF Integration to FPGA Atera DE II

DE2-115 is a development board with the main component in the form of Altera Cyclone® IV 4CE115 FPGA. Soft core NIOS processor can be implemented on FPGA. NIOS is a soft core 32-bit RISC Microprocessor. In this paper, we used 50 MHz frequency on BCF. The BCF module was wrapped with the Avalon interconnect interface. Figure 15 shows the block diagram of the NIOS interface.

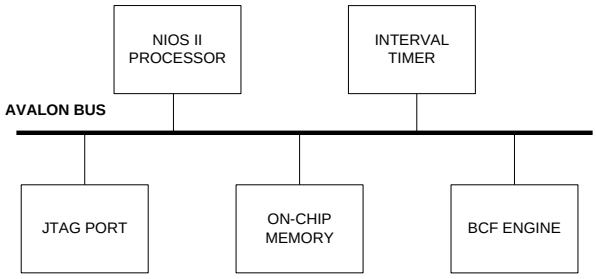

Fig. 15 BCF - NIOS interface

To access IP BCF, NIOS write / read registers in Table 11.

The functionality test of IP BCF was carried out by comparing the computing results of BCF IP with program that run on a computer. Figure 16 and 17 show the result.

Based on the above test, the ciphertext and plaintext values generated by the IP Core BCF were found similar with those generated by the C program running on the computer. This means that the implementation of BCF on BCF has been functionally successful.

Table 11. Register for software

Address	W/R	Description
0x81080	W	Write to register Key on bit 0 to 31
0x81081	W	Write to register Key on bit 32 to 63
0x81082	W	Write to register Key on bit 64 to 95
0x81083	W	Write to register Key on bit 96 to 127
0x81084	W	Write to register Key on bit 128 to 159
0x81085	W	Write to register Key on bit 160 to 191
0x81086	W	Write to register Key on bit 192 to 223
0x81087	W	Write to register Key on bit 224 to 255
0x81088	W	Write to register din on bit 0 to 31
0x81089	W	Write to register din on bit 32 to 63
0x8108A	W	Write to register din on bit 64 to 95
0x8108B	W	Write to register din on bit 96 to 127
0x8108C	W	Write to register key_len 0: 128 bit 1: 192 bit 2: 256 bit
0x8108E	W	Write to register decrypt 0: Decryption 1: Mode encryption
0x8108F	W	Write to register BCF_start
0x81090	R	Read BCF_Flag 0: BCF not done 1: BCF done
0x81091	R	Read register dout on bit 0 to 31
0x81092	R	Read register dout on bit 32 to 63
0x81093	R	Read register dout on bit 64 to 95
0x81094	R	Read register dout on bit 96 to 127
0x81095	W	Reset BCF Engine

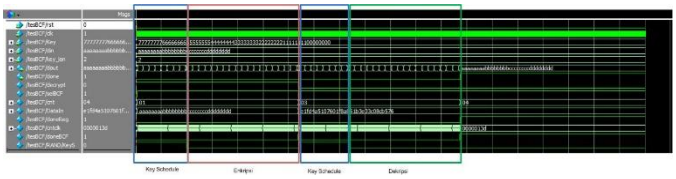

Fig 16. IP BCF result

```
YK's Block Cipher BCF
Key :    77777777 66666666 55555555 44444444 33333333 22222222 11111111 0
Plaintext (128 bit) :   aaaaaaaa        bbbbbbbb        cccccccc        dddddddd
The result of encryption (ciphertext) :         e1fd4a51        07601f8a        861b3e33        c08cb576
The result of decryption (plaintext) :  aaaaaaaa        bbbbbbbb        cccccccc        dddddddd
```

Fig 17. BCF engine computer result

One way to measure the BCF performance is by comparing the hardware and software implementation. If the speed of the hardware far exceeds the speed of the software, the hardware implementation can be stated to be successful. The measurement results are presented in Table 12 showing that the computation time for BCF software implementation depends on the processor architecture. Hardware BCF Engine can speed up BCF compute 488-2847 times compared to software, dependent on processor architecture and BCF key length.

Table 12. Software and hardware running time comparation

NIOS II Version	Average execution time (μS)					
	256 bit		192 bit		128 bit	
	NIOS II	BCF Engine	NIOS II	BCF Engine	NIOS II	BCF Engine
Fast	141664	290	135977	255	127704	239
Standart	172815	273	159609	261	153015	253
Economy	760116	267	701536	267	672323	267

The speed of BCF and AES was measured, and the results of the comparison are shown in Table 13 informing the BCF Engine was 44 times faster than the AES hardware accelerator, where the devices operated at a clock of 50MHz. From this data, BCF Engine is suitable to be implemented in devices with small computing resources such as IoT, where these devices require a low clock for power saving, but require a high level of security for sending data to the internet [17].

Table 13. Comparasion of Block Cipher 128-bit implementation on FPGA

Design	Algorithm	Execution time (uS)
Sideris *et al.*[16]	AES	10414
This work	BCF	239

3. Conclusion

This paper describes the BCF encryption algorithm, the algorithm implementation on the Altera DE2-115 FPGA and its performance. On Altera DE2-115 boards, hardware implementations were found 488-2847x faster than software implementations, dependent on processor architecture and BCF key length. BCF also has a high speed to be implemented on devices with small resources such as IoT. For further research, we will perform a Correlation Power Analysis (CPA) attack on this proposed BCF device. The attack will be based on the previous paper [2].

References

1. Y. Kurniawan. *The BCF block cipher design*, Technical Report, Institut Teknologi Bandung, Indonesia, 2018.
2. Ma'muri, Y. Kurniawan and S. Sutikno, *Implementation of BC3 encryption algorithm on FPGA Zynq-7000, Int. Symp. Electronics Smart Devices, Yogyakarta, Indonesia, 2017, pp. 329-334.*
3. Hua Li and Z. Friggstad, *An efficient architecture for the AES mix columns operation, IEEE Int. Symp. Circuits Syst., Kobe, 2005, pp. 4637-4640.*
4. S. Ghaznavi, C. Gebotys and R. Elbaz, *Efficient technique for the FPGA implementation of the AES mixcolumns transformation, Int. Conf. Reconfigurable Computing FPGAs, Quintana Roo, 2009, pp. 219-224.*
5. A. O. Adebayo, M. S. Chaubey, and L. P. Lumbu, *Industry 4.0: The fourth industrial revolution and how it relates to the application of internet of things(IoT)*, J. Multidisciplinary Eng. Sci. Studies 5 (2019) 2477-2482.
6. I. A. Landge and H. Satopay, *Secured IoT through hashing using MD5, Fourth Int. Conf. Adv. Electric., AEEICB, Chennai, 2018, pp. 1-5.*
7. S. D. Putra, M. Yudhiprawira, Y. Kurniawan, S. Sutikno and A. S. Ahmad, *Security analysis of BC3 algorithm for differential power*

analysis attack, Int. Symp. Electronics Smart Devices (ISESD), Yogyakarta, 2017, pp. 341-345.

8. P. C. Kocher. *Timing attacks on implementations of diffie-hellman, RSA, DSS, and other systems*. In: Koblitz N. (eds) Advances in Cryptology — CRYPTO '96. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 1996.
9. Altera, *DE2-115 cyclone II development board user manual*, Altera, 2010
10. A. Laboratory, *Cryptographic hardware project: IP core*, http://www.aoki.ecei.tohoku.ac.jp/crypto/web/cores.html, 2019.
11. W. Stalling. *Cryptography and network security principles and practices*(4th ed). New Jersey: Prentice Hall, 2005.
12. P. S. L. M. Barreto and V. Rijmen. *The Khazad Legacy-level Block Cipher*, NESSIE, 2001.
13. A. E. Standard., *Federal Information Processing Standards Publication*, FIPS PUB 197, 2001.
14. K. Aoki, T. Ichikawa, M. Kanda, M. Matsui, S. Moriai, J. Nakajima, T. Tokita,*Specification of camellia — a 128-bit block cipher*, NTT and Mitsubishi Electric Corporation 2000-2001, 2001, pp. 1-35.
15. B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, N. Ferguson, *Twofish: a 128-bit block cipher*. *Proceedings first AES cand. conf.*, 1998.
16. A. Sideris, T. Sanida and M. Dasygenis, *Hardware acceleration of the AES algorithm using Nios-II processor*, *Panhellenic Conf. Electronics Telecommunications, Volos, Greece, 2019, pp. 1-5.*
17. R. Mahmoud, T. Yousuf, F. Aloul and I. Zualkernan, *Internet of things (IoT) security: Current status, challenges and prospective measures, 10th Int. Conf. Internet Tech. Secured Transactions, London, 2015, pp. 336-341.*

Appendix

Table 14. FSM state description

State	Description
S0	Reset state KeyS = 1
S1	Pre-Whitening
S2	round = 1
S3	round = 15
S4	round = 16
S5	round = 18
S6	Conditional state
S7	dinR = KM, dinL = KL
S8	dinR = XRreg, dinL = XLreg
S9	Wait until F0 done, then XReg = XR, XLreg = XL
S10	Input data to KAreg and KBreg register
S11	Key Schedule finished
S12	Process(Key Schedule) not finished, go to S6 state, ronde = ronde + 1
S13	Conditional state
S14	Process (encryption or decryption) not finished, go to S6 state, ronde = ronde + 1
S15	Process (Key Schedule or encryption or decryption i) finished, goto S6 state, if Key Schedul finished and KeyS = 1, then KeyS = 0 and go to state S1.

Table 15. FO core pinout description

Port Name	Data Length (bit)	Direction	Description
din	128	Input	Data input, plain text or cipher text.
dinR	64	Input	Data input right
dinL	64	Input	Data input left
Ks	64	Input	Subkey
XR	64	Output	Data output right
XL	64	Output	Data output left
KS1…KS18	64	Input	Subkey for each round
KS1	64	Input	KeyS = 1, then KS1 = C3. Decrypt = 1, then KS1 = K1. Key_len = 0 & decrypt = 0, then KS1 = K15. Key_len = 1 & decrypt = 0, then KS1 = K16. Key_len = 2 & decrypt = 0, then KS1 = K18.
KS2	64	Input	KeyS = 1, then KS2 = C4. Decrypt = 1, then KS2 = K2. Key_len = 0 & decrypt = 0, then KS2 = K14. Key_len = 1 & decrypt = 0, then KS2 = K15. Key_len = 2 & decrypt = 0, then KS2 = K17.
KS3	64	Input	KeyS = 1, then KS3 = K1R. Decrypt = 1, then KS3 = K3. Key_len = 0 & decrypt = 0, then KS3 = K13. Key_len = 1 & decrypt = 0, then KS3 = K14. Key_len = 2 & decrypt = 0, then KS3 = K16.
KS4	64	Input	KeyS = 1, then KS4 = K1L. Decrypt = 1, then KS4 = K4. Key_len = 0 & decrypt = 0, then KS4 = K12. Key_len = 1 & decrypt = 0, then KS4 = K13. Key_len = 2 & decrypt = 0, then KS4 = K15.
KS5	64	Input	KeyS = 1, then KS5 = C1. Decrypt = 1, then KS5 = K5. Key_len = 0 & decrypt = 0, then KS5 = K11. Key_len = 1 & decrypt = 0, then KS5 = K12. Key_len = 2 & decrypt = 0, then KS5 = K14.
KS6	64	Input	KeyS = 1, then KS6 = C2. Decrypt = 1, then KS6 = K6.

Table 15 "continued". FO core pinout description

			Key_len = 0 & decrypt = 0, then KS6 = K10.S			
			Key_len = 1 & decrypt = 0, then KS6 = K11.			
			Key_len = 2 & decrypt = 0, then KS6 = K13.			
KS7	64	Input	KeyS = 1, then KS7 = C3.			
			Decrypt = 1, then KS7 = K7.			
			Key_len = 0 & decrypt = 0, then KS7 = K9.			
			Key_len = 1 & decrypt = 0, then KS7 = K10.			
			Key_len = 2 & decrypt = 0, then KS7 = K12.			
KS8	64	Input	KeyS = 1, then KS8 = C4.			
			Decrypt = 1, then KS8 = K8.			
			Key_len = 0 & decrypt = 0, then KS8 = K8.			
			Key_len = 1 & decrypt = 0, then KS8 = K9.			
			Key_len = 2 & decrypt = 0, then KS8 = K11.			
KS9	64	Input	KeyS = 1, then KS9 = 0.			
			Decrypt = 1, then KS9 = K9.			
			Key_len = 0 & decrypt = 0, then KS9 = K7.			
			Key_len = 1 & decrypt = 0, then KS9 = K8.			
			Key_len = 2 & decrypt = 0, then KS9 = K10.			
KS10	64	Input	KeyS = 1, then KS10 = 0.			
			Decrypt = 1, then KS10 = K10.			
			Key_len = 0 & decrypt = 0, then KS10 = K6.			
			Key_len = 1 & decrypt = 0, then KS10 = K7.			
			Key_len = 2 & decrypt = 0, then KS10 = K9.			
KS11	64	Input	KeyS = 1, then KS11 = 0.			
			Decrypt = 1, then KS11 = K11.			
			Key_len = 0 & decrypt = 0, then KS11 = K5.			
			Key_len = 1 & decrypt = 0, then KS11 = K6.			
			Key_len = 2 & decrypt = 0, then KS11 = K8.			
KS12	64	Input	KeyS = 1, then KS12 = 0.			
			Decrypt = 1, then KS12 = K12.			
			Key_len = 0 & decrypt = 0, then KS12 = K4.			
			Key_len = 1 & decrypt = 0, then KS12 = K5.			
			Key_len = 2 & decrypt = 0, then KS12 = K7.			
KS13	64	Input	KeyS = 1, then KS13 = 0.			
			Decrypt = 1, then KS13 = K13.			
			Key_len = 0 & decrypt = 0, then KS13 = K3.			
			Key_len = 1 & decrypt = 0, then KS13 = K4.			
			Key_len = 2 & decrypt = 0, then KS13 = K6.			
KS14	64	Input	KeyS = 1, then KS14 = 0.			
			Decrypt = 1, then KS14 = K14.			
			Key_len = 0 & decrypt = 0, then KS14 = K2.			
			Key_len = 1 & decrypt = 0, then KS14 = K3.			
			Key_len = 2 & decrypt = 0, then KS14 = K5.			
KS15	64	Input	KeyS = 1, then KS15 = 0.			
			Decrypt = 1, then KS15 = K15.			
			Key_len = 0 & decrypt = 0, then KS15 = K1.			
			Key_len = 1 & decrypt = 0, then KS15 = K2.			
			Key_len = 2 & decrypt = 0, then KS15 = K4.			
KS16	64	Input	KeyS = 1, then KS16 = 0.			
			Decrypt = 1, then KS16 = K16.			
			Key_len = 1 & decrypt = 0, then KS16 = K1.			
			Key_len = 2 & decrypt = 0, then KS16 = K3.			
KS17	64	Input	KeyS = 1, then KS17 = 0.			
			Decrypt = 1, then KS17 = K17.			
			Key_len = 2 & decrypt = 0, then KS17 = K2.			
KS18	64	Input	KeyS = 1, then KS18 = 0.			
			Decrypt = 1, then KS18 = K18.			
			Key_len = 2 & decrypt = 0, then KS18 = K1.			
KA, KB, KJ, KK, KL, KM	64	Input	Subkey untuk randomizing SBox.			
KWA, KWB	64	Input	Subkey untuk pre-whitening.			
cntF0	4	Input	Data counter ronde BCF.			
dF0	1	Input	0: Ouput Pre-Whittening 1: Ouput F0			
KeyS	1	Input	0: Encrypt/Decrypt 1: Key Schedulling			
KeyS1	1	Input	0: XRREg, XLReg 1: KM, KL			

Table 16. F0 module pinout description

Port Name	Data Length (bit)	Direction	Description
dinR	64	Input	Data input F0 function
dinL	64	Input	Data input F0 function
Ks	64	Input	Subkey
S…S7	3	Input	Sbox number used
cnt	4	Input	Data counter.
XL	64	Output	Data output F0 function
XR	64	Output	Data output F0 function