Implementation of GOST 28147-89 Encryption and Decryption Algorithm on FPGA

H. AKTAŞ[1]

[1] Akdeniz University, Antalya/Turkey, haktas@akdeniz.edu.tr

***Abstract* – GOST(Gosudartsvennyi Standart) Algorithms are state security algorithms developed by Russian Federation (formerly Soviet Union). The first of these algorithms is GOST 28147-89 encryption and decryption algorithm developed in 1987. Other algorithms are GOST 34.11-94 hash function algorithm and GOST 34.10-2001 digital signature algorithm. GOST 28147-89 encryption algorithm is a 64-bit block cipher main algorithm and this main algorithm is used in GOST 34.11-94 hash function algorithm and GOST 34.10-2001 digital signature algorithm. Since the computational time of encryption algorithms are very high, to make a real time and fast encryption algorithms FPGAs(Field Programmable Gate Arrays) are the best platforms to implement these algorithms except ASICs(Application-Specific Integrated Circuits). In this study GOST 28147-89 encryption and decryption algorithm will be implemented with verilog and the algorithm speed will be tested for real time applications.**

***Keywords* - GOST 28147-89, Block Cipher, Encryption Algorithms, FPGA, Verilog, Real Time Applications**

I. INTRODUCTION SYMETRIC KEY ENCRYPTION

Symmetric key block ciphers are the most common encryption methods in cryptographic systems[1]. These ciphers are used as main blocks for the Random Number Generators, Hash Functions and Message Authentication Codes(MAC)[2]. In block cipher systems, n is the length of block and the data is divided into the n lengths of datas and these n lengths datas are encrypted one by one. The well known block cipher algorithms are: DES(Data Encryption Standart)[3] and AES(Advanced Encryption Standart)[4]. Besides, Lucifer[5], GOST[6] and Blowfish[7] are the other main algorithms. However there are many other block cipher algorithms in literature, most of them are not used commonly. Because most of these algorithms are not tested against cryptographic attacks[1].

In block cipher algorithms key length can be different according to the algorithms. S-boxes are very important parts of the algorithms. Block cipher algorithms are different from each others but in many algorithms there is main *f* function and key generation. S-boxes are located in these main *f* functions and S-boxes are the only nonlinear structure in the whole algorithm. This specialty of S-boxes makes the algorithm stronger against linear attacks. One another main structure is fiestal network structure which was created by Horst Feistal[8] in 1970s. In this structure n is the length of the block, n length block is divided into the L and R blocks which's length is n/2. This L and R blocks used in the next round and this makes the algorithm an iterative structure[8]. This can be shown with the equation 1 and 2.

$$L_i = R_{i-1} \tag{1}$$

$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i) \tag{2}$$

K_i , is the sub key for the $i.$ round and *f* is the main function of the algorithm. This structure can be seen in DES, GOST, Lucifer and Blowfish algorithms. The main advantage of this structure is that it makes the algorithm reversible, means that encryption and decryption is the same function. In equation 2, Xor is used and this makes the algorithm reversible, decryption algorithm is showed as in equation 3.

$$L_{i-1} \oplus f(R_{i-1}, K_i) \oplus f(R_{i-1}, K_i) = L_{i-1} \tag{3}$$

In this structure to decrypt the data, *f* function is not important. *f* function only makes the algorithm stronger against attacks.

II. GOST 28147-89

GOST 28147-89 is a block cipher algorithm which was developed by Soviet Union in 1989[6]. It's structure is so similar to DES, algorithm encrypts the 64 bits blocks with the 256 bits key. It has a fiestel network structure and data encrypted with an iterative way in 32 rounds. The algorithm works in one round as shown in figure 1.

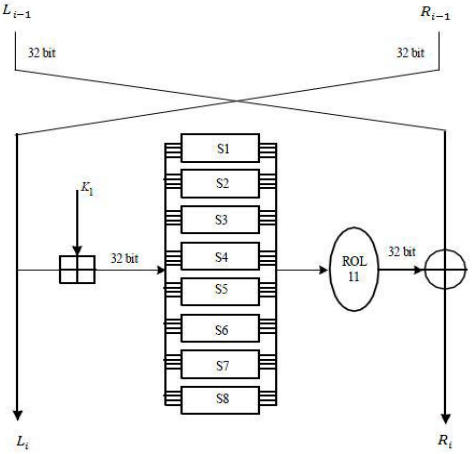

Figure 1 : Encryption in one round.

Key length is 256 bits and sub keys are 32 bits. Since GOST is 32 rounds and there are 8 subkeys, every subkeys is used for 4 times in 32 rounds. Subkey sequence can be seen in table 1.

Table 1: Subkey sequence for 32 rounds [9].

Round	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Subkey	1	2	3	4	5	6	7	8
Round	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Subkey	1	2	3	4	5	6	7	8
Round	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**
Subkey	1	2	3	4	5	6	7	8
Round	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**
Subkey	8	7	6	5	4	3	2	1

After *Li* , *Ri* and subkeys are generated the first operation is modulo sum of *Ri* and subkeys. The results of this modulo sum will be the input of 8 S-boxes. MSB bits are going to be an input for S1 and LSB bits are input for S8[9]. The usage of S-boxes can be seen in table 2.

Table 2: S-Boxes[9].

S-Box1															
4	10	9	2	13	8	0	14	6	11	1	12	7	15	5	3
S-Box2															
14	11	4	12	6	13	15	10	2	3	8	1	0	7	5	9
S-Box3															
5	8	1	13	10	3	4	2	14	15	12	7	6	0	9	11
S-Box4															
7	13	10	1	0	8	9	15	14	4	6	12	11	2	5	3
S-Box5															
6	12	7	1	5	15	13	8	4	10	9	14	0	3	11	2
S-Box6															
4	11	10	0	7	2	1	13	3	6	8	5	9	12	15	14
S-Box7															
13	11	4	1	3	15	5	9	0	10	14	7	6	8	2	12
S-Box8															
1	15	13	0	5	7	10	4	9	2	3	14	6	11	8	12

S-boxes has 4 bits inputs and 4 bits outputs. For example if S-Box1 input is 0xA then output of the S-Box1 is going to be 0x1. Outputs of these 8 S-boxes will be concatenated and 32 bits data will be generated. In next step this 32 bits data will be 11 bits shift left rolled. And in final step of this rolled data and *Li* bits is going to be Xored . Decryption has the same structure with the encryption. 64 bits blocks divided into the 32 bits blocks. The first 32 bits blocks of decryption can be shown as in equation 4 and 5.

$$L_1 = R_{32} = L_{31} \oplus f(R_{31}, K_1) \quad (4)$$

$$R_1 = L_{32} = R_{31} \quad (5)$$

Decryption done in 32 rounds and at the end of 32. round plain text is generated. GOST algorithms is developed alternatively for DES algorithm by Soviet Union. Comparison of two algorithm is :

- Subkey generation in DES is much more complicated then GOST subkey generation.
- DES has 56 bits key and GOST has 256 bits key length
- DES S-boxes has 6 bits inputs and 4 bits outputs but GOST S-boxes has 4 bits inputs and outputs.
- Confusion is done by permutation in DES, in GOST this operation is done with 11 bits rolling shift left.

III. FPGA IMPLEMENTATION OF GOST 28147-89

GOST 28147-89 has an iterative network structure and these kind structures are very compatible for FPGAs[10]. The encryption done in 32 rounds. And every 32 rounds there are four operations which are: modulo sum, S-boxes, Rol 11 and Xor function. This means that, every round takes 4 clocks if any pipeline structure didn't done. Figure 2 shows the one round encryption f function for the GOST 28147-89. And this function takes 4 clock cycles.

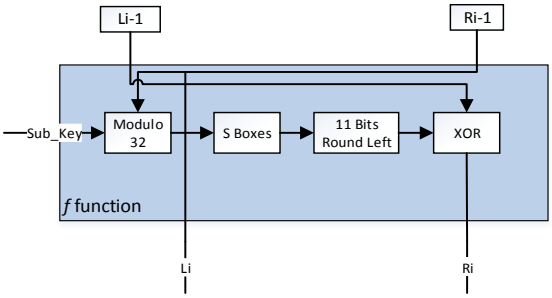

Figure 2 : *f* function of GOST 28147-89

Using this *f* function module in every 32 rounds means that outputs of *f* function will be the input of *f* function in next step. This can be shown in figure 3.

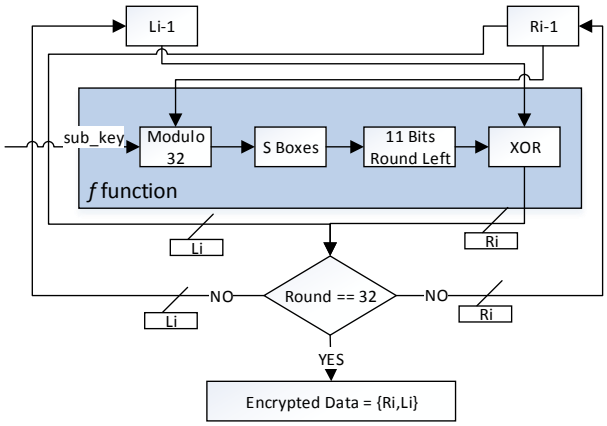

Figure 3 : Full encryption with only one *f* function

In this work Virtex-5 XC5VLX110T FPGA used for the implementation. The FPGA chip has 100MHz clock frequency. If the algorithm implemented with only one *f* module without any pipeline structure; this means that the inputs for next round is generated in 4 clocks. To encrypt 64 bit blocks FPGA implementation will takes: 4 clocks * 32 rounds = 128 clocks. Since our FPGA clock is 100MHz then 64 bit block encryption takes: 128 * 10ns = 1280ns = 1280 ns. In this work, FPGA implementation of GOST 28147-89 Algorithm is implemented with Verilog. As can be seen from figure 4 one 64 bits block encryption takes 1280ns.

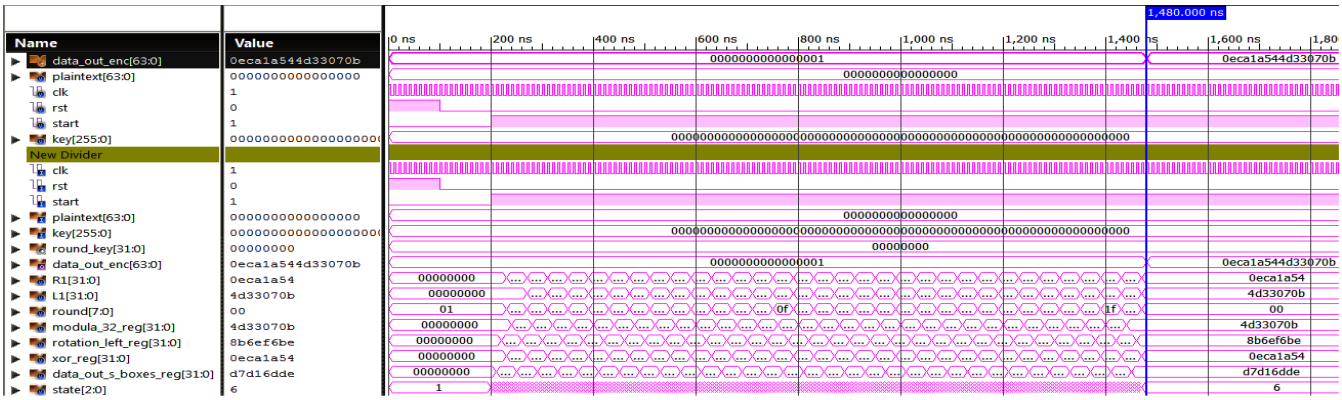

Figure 4 : Verilog implementation of algorithm without pipeline structure

This calculation time is quite fast for the implementation but if this implementation is used in a real time system and if an image wanted to be encrypted then this duration can be critical for the implementation. Table 3 shows the encryption duration of one frame for different resolution and fps of gray scale camera outputs.

Table 3: Encryption duration for different types of camera outputs

Camera Type	Resolution	fps	Number of Pixels in Bits	One frame Encryption
VGA	640*480	20	2457600	49.15 ms
VGA	640*480	50	2457600	49.15 ms
720p	1280*720	20	7372800	147.45 ms
720p	1280*720	50	7372800	147.45ms
1080p	1920*1080	20	16588800	331.77ms
1080p	1920*1080	50	16588800	331.77ms

From table 3 it is obviously seen that only first row (VGA, 640*480, 20fps) can be achieved. Since the camera is 20 fps encryption of one frame must be done in 50ms. If the VGA resolution camera is 50 fps then it is not possible to encrypt it in 20ms because, the algorithm takes 49.53 ms. To make a real time encryption core and use it for 1080p 50 fps camera, one frame encryption must be done in 20ms. Since the algorithm takes 334.368ms, one frame encryption algorithm must be accelerated minimum 334.368 / 20 = 16.718 times.

IV. ACCERELATION OF ALGORITHM WITH PIPELINE STRUCTURE

FPGAs are very effective and efficient platforms for cryptography applications such as encryption[11]. Since the algorithm has 32 rounds, if the main *f* function is used 32 times in a pipeline structure then the algorithm can be accelerated 32*4 = 128 times. Figure 5 shows how this structure can be done. Every single *f* functions calculate the result in 4 clocks and last clock calculates the final outputs of *f* function. Therefor, between two *f* function using 4 registers can make pipeline structure possible. With the first clock Li-1 goes to Reg5, Ri-1 goes to Modulo 32 and Reg1, after 4 clocks Ri-1 goes to Li and Output of Xor module goes to Ri. Data transitions with every one clock can be fallowed with the coloured registers. Same colours shows the synchronous data transitions.

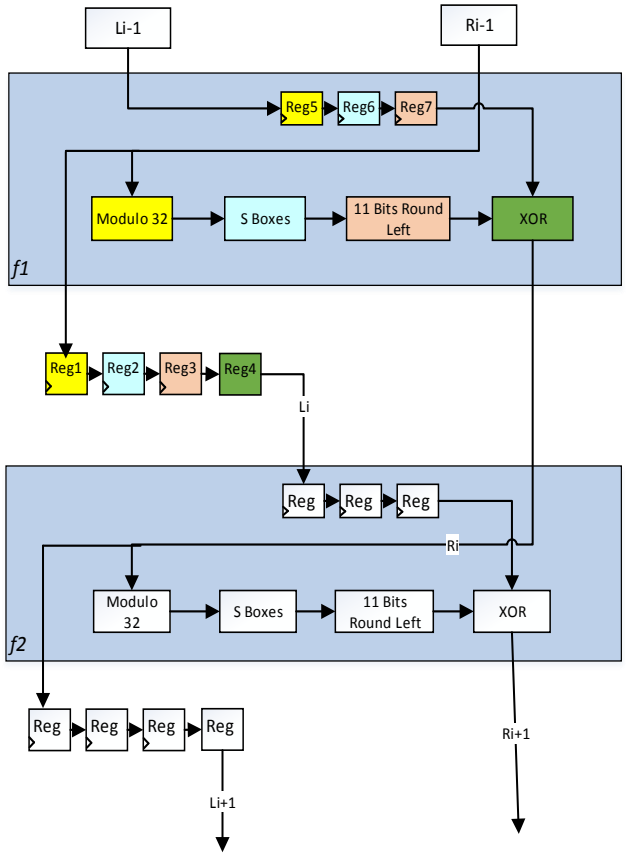

Figure 5 : Pipeline structure of algorithm between two f functions.

The whole pipeline structure is showed in figure 6. Using pipeline structure the first 64 bit blocks again takes 32*4 = 128 clocks. After 128 clocks in every single clock 64 bit blocks can be encrypted because of the pipeline structure. This can be seen in figure 7. This means that one 1080p resolution frame can be encrypted in approximately 331,77 / 128 = 2.59 ms.

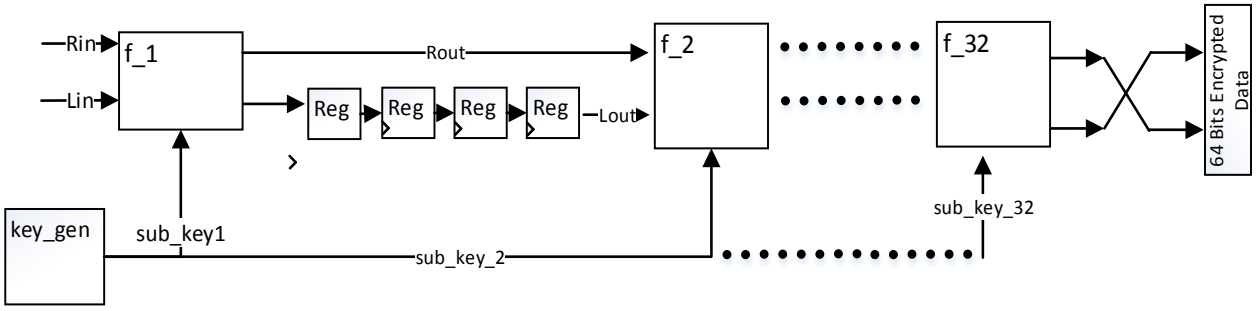

Figure 6: Pipeline structure of whole algorithm

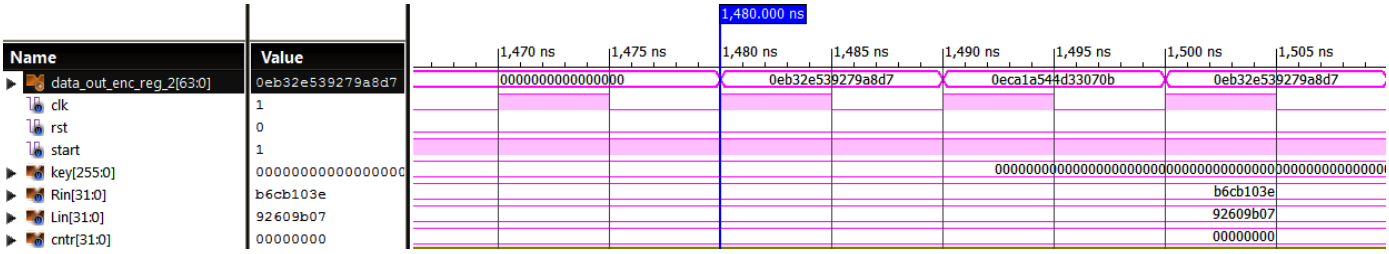

Figure 7 : Verilog implementation of algorithm with pipeline structure

V. CONCLUSION

In this paper GOST 28147-89 Encryption and Decryption Algorithm is implemented with Verilog codes on Virtex-5 XC5VLX110T FPGA. Without pipeline structure the algorithm is not fast enough for real time image encryption systems. To make the algorithm fast enough, pipeline structure is used. The results of the pipeline structure shows that 1080p 50 fps gray scale camera output can be encrypted in real time. For the 1080p resolution encryption takes approximately 2.59ms. If the camera is selected rgb which has 1080p resolution then, one frame encryption will take 2.59*3 = 7.7ms. This means that with this pipeline structure can achieve 1080p, 100fps rgb camera output encryption in real time. The computation performance of the pipeline structure is 5.96 Gbit/s. This result shows that the implementation is compatible for real time implementations.

APPENDIX

This example shows every round results during 32 rounds encryption. Example should be used for the readers for the next studies as a test vector file. Encrypted data output *0eca1a54 4d33070b* can be also seen in figure 7.

plain_text : 00000000 00000000
key : 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

1. round : 00000000 bb268a72
2. round : bb268a72 ca44ee08
3. round : ca44ee08 bed841b7
4. round : bed841b7 b89f4820
5. round : b89f4820 24424fa0
6. round : 24424fa0 ea684495
7. round : ea684495 56ff7562
8. round : 56ff7562 767aa8e8
9. round : 767aa8e8 62663a1b
10. round : 62663a1b 3ebf58ca
11. round : 3ebf58ca fdfd2330
12. round : fdfd2330 ad3fd177
13. round : ad3fd177 6421838e
14. round : 6421838e 473f9147
15. round : 473f9147 f9782558
16. round : f9782558 34a858de
17. round : 34a858de 8ee4646e
18. round : 8ee4646e b2129bf2
19. round : b2129bf2 dbca0a4a
20. round : dbca0a4a 81530479
21. round : 81530479 50f69913
22. round : 50f69913 4c66800c
23. round : 4c66800c 1ad02f91
24. round : 1ad02f91 7793fd4c
25. round : 7793fd4c 93b198c6
26. round : 93b198c6 9a88a8af
27. round : 9a88a8af e72efd01
28. round : e72efd01 33ee527f
29. round : 33ee527f 48fa1c65
30. round : 48fa1c65 85a4ecea
31. round : 85a4ecea 4d33070b
32. round : 4d33070b 0eca1a54

Encrypted : 0eca1a54 4d33070b

REFERENCES

[1] Stinson D.R., *Cryptography Theory and Practice*, Second Edition, Chapman &Hall/CRC, CRC Press Company

[2] ANSI X9.9, *'American National Standart for Financial Institution Message Authentication(Wholesale)'*, American Bankers Association, 1986.

[3] E. Schaefer, *An introduction to cryptography*. Santa Clara University, United States of America, 1998.

[4] National Inst. Of Standards and Technology, *"Federal Information Processing Standard Publication 197, the Advanced Encryption Standard (AES),"* Nov. 2001

[5] Smith J. L., *'The Design of Luficer, A Cryptographic Device for Data Communications,'* IBM Research Report RC3326, 1971.

[6] Schneier B., *The GOST Encryption Algorithm,* Dr. Doob's Journal, v. 20, n. 1, pp. 123-124, 1995.

[7] Schneier B., *'The Blowfish Encryption Algorithm,'* Dr. Dobbs journal, v. 19, n. 4, pp. 38-40, 1994.

[8] Feistel H., *Cryptography and Computer Pricvacy.* Scientific American, v.228, n. 5, pp. 10 – 18. 1973.

[9] Schneier B., *Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2nd edition,* John Wiley & Sons, Inc, 1996.

[10] J.P. Kaps and C. Paar, *"Fast DES Implementations for FPGAs and Its Application to a Universal Key-Search Machine,"* 5th Annual Workshop on selected areas in cryptography, pp.234-247 ,Canada 1998

[11] McLoone, M., and McCanny, A.: '*A high performance implementation of DES'*. Proc. IEEE Workshop on Signal processing systems design and implementation, SiPS2000, Lafayette, LA, USA, October 2000, pp. 374–383