Journal of Circuits, Systems, and Computers
Vol. 14, No. 2 (2005) 217–231
© World Scientific Publishing Company

AN FPGA IMPLEMENTATION OF THE GPRS ENCRYPTION ALGORITHM 3 (GEA3)

P. KITSOS*, M. D. GALANIS and O. KOUFOPAVLOU
VLSI Design Laboratory,
Electrical and Computer Engineering Department,
University of Patras, Patras, Greece
**pkitsos@ee.upatras.gr*

Received 21 May 2003
Revised 10 May 2004

The General Packet Radio Service (GPRS) uses the GPRS Encryption Algorithm 3 (GEA3) for data encryption. In this paper, alternative hardware implementations of the GEA3 algorithm are described. GEA3 algorithm is based on the KASUMI block cipher. Various KASUMI block cipher hardware implementations have been examined in order to provide information about the required silicon area and throughput. In order to achieve a significant performance improvement, Double Edge Triggered pipeline technique is used. The S-BOXes, which are fundamental elements of the KASUMI cipher, have been implemented by using combinational logic and ROM memories. The proposed GEA3 algorithm hardware implementation achieves throughput up to 837 Mbps, which is much faster comparing to the previous designs. The whole system is implemented and evaluated by using Field Programmable Gate Array (FPGA) devices.

Keywords: GPRS security; GEA3; KASUMI; stream cipher; block cipher; S-BOX; double edge triggered (DET) pipeline.

1. Introduction

The General Packet Radio Service (GPRS) is a new nonvoice value-added service that allows information to be sent and received across a mobile telephone network.[1] It supplements today's Circuit Switched Data and Short Message Service. Theoretical maximum speeds of up to 171.2 kilobits per second (kbps) can be achieved. This is about three times as fast as the data transmission speeds possible over today's fixed telecommunications networks and ten times as fast as current Circuit Switched Data services on GSM networks. By allowing information to be transmitted more quickly, immediately and efficiently across the mobile network, GPRS may well be a relatively less costly mobile data service compared to SMS and Circuit Switched Data.

GPRS does not itself offer end-to-end security but there are many proprietary techniques and other standards that cover methods of end-to-end encryption, such as the Internet security protocols covered by IPsec. Care needs to be taken in the design of the network to ensure that sensitive elements (such as encryption keys, and identities) cannot be compromised. There may also be legal constraints, such as data privacy or lawful interception requirements that affect the inclusion of encryption, access control or interception within the infrastructure. The air interface ciphering in GPRS is at the same level as in an ordinary GSM network without GPRS. The encryption algorithm in GPRS is GPRS Encryption Algorithm 3 (GEA3).[2]

The GEA3 algorithm is based on KASUMI[3] block cipher. KASUMI uses S-BOXes as fundamental elements. In this work, two different design techniques are proposed. The first one reduces the required silicon area and the second one achieves high-speed performance. The first one uses combinational logic for the S-BOX implementation and the second one uses ROM memory. In order to improve the overall system performance, Double Edge Triggered (DET) pipeline design technique is used. The throughput of the proposed KASUMI block cipher implementations is much higher than previous similar designs.[4–7]

The paper is organized as follows: in Sec. 2, the GEA3 algorithm is introduced. In Sec. 3, the KASUMI block cipher, which is the basic module for the GEA3 algorithm, is briefly presented. The proposed architecture and implementation are analyzed in detail in Sec. 4. Synthesis results for the Field Programmable Gate Array (FPGA) implementations are shown in Sec. 5. Finally, in Sec. 6, conclusions and observations are presented.

2. GEA3 Algorithm Architecture

The GEA3 Algorithm[2] is a stream cipher that encrypts/decrypts blocks of data, between 1 to M bytes (the maximum value of M is 65 536 bytes) in length, by using a confidentiality key K_C. The GEA3 keystream generator is based on a block cipher KASUMI in a form of Output Feedback Mode (OFB),[8] and generates the output keystream in multiples of 64-bits (Fig. 1).

The GEA3 algorithm is almost the same as the UMTS Confidentiality algorithm $f8$.[9] There are only two operation differences. First, GEA3 algorithm has different initialization vector to $f8$, and second, it may operate more times than $f8$, because it may encrypt/decrypt more data blocks.

At the initialization phase, the system parameters CA, CB, INPUT, CE, and DIRECTION[2] are padded in order to make a 64-bit input. With the use of a modified version of the confidentiality key K_C, this input is encrypted. The modified version of the confidentiality key is derived by bitwise XOR of the original confidentiality key with a predefined variable (KM).[2] The output of this process is a 64-bit value A. This value A and the block counter (BLKCNT) are the input in a number of KASUMI cipher units (in OFB operation). Each of these units produces a Keystream (KS) block (Fig. 1). At the last stage, the encryption/decryption operations are identical and the cipher text is produced by the XOR of the plaintext with the generated KS block.

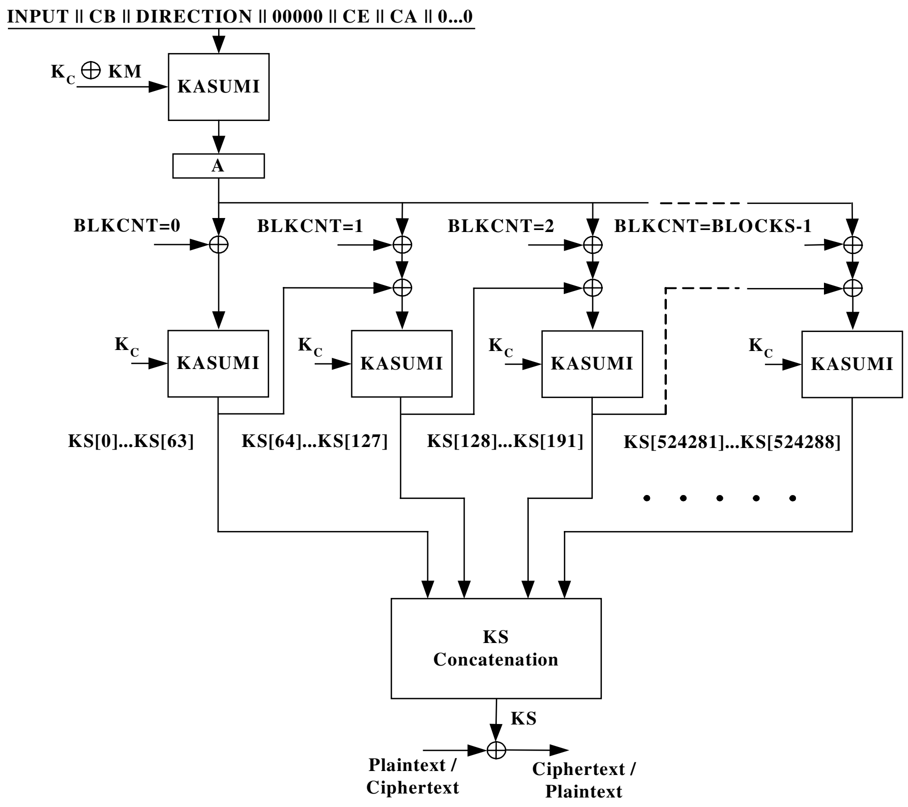


Fig. 1. The GEA3 algorithm architecture.

The keystream generator produces bits in blocks of 64-bit, but the number of output bits may not be a multiple of 64. Depending on the total number of bits, specified by M, 0 to 63 of the least significant bits are discarded from the last block.

3. KASUMI Block Cipher

3.1. *Round function*

The KASUMI[3] block cipher is an enhancement version of MISTY[10] block cipher. KASUMI cipher is designed in order to provide security against differential and linear cryptanalysis.[11,12]

KASUMI cipher comprises eight rounds of Feistel[13] networks. By using a 128-bit ciphering key K, it modifies a 64-bit Plaintext to a 64-bit Ciphertext. The 64-bit input is divided into two 32-bit strings L_0 and R_0. The outputs of each round are produced according to the following equation:

$$R_i = L_{i-1}\,, \quad L_i = R_{i-1} \oplus f_i(L_{i-1}, RK_i)\,, \qquad 1 \leq i \leq 8\,, \tag{1}$$

where f denotes the round function with L_{i-1} and round key RK_i as inputs, and $\oplus$ the exclusive-OR logical operation. The subscript i denotes the number of cipher round.

The round key RK_i comprises of the subkey triplet (KL_i, KO_i, KI_i). At the end of each round, the left and right 32-bit blocks are also swapped. The produced Ciphertext is the 64-bit string derived from the concatenation of the L_8 and the R_8 which are produced at the end of the eighth round. The f itself is constructed from two subfunctions; FL and FO with associated subkeys KL_i (used with FL) and subkeys KO_i and KI_i (used with FO), followed by a bitwise XOR operation with the previous branch. This function has two different forms depending on whether it is an even round or an odd round. In the odd round, the FL function is performed first and then the FO function follows. In the even round, the order of the functions is reversed. The FO function consists of a 3-round network with a 16-bit nonlinear function FI. The FI function consists of a 4-round network with two S-BOXes, S9 and S7. The two S-BOXes perform operations in Galois field (GF). In details, the S9 performs the x^5 in GF(2^9) operation and the S7 performs the x^{81} in GF(2^7) operation.

The FL function transforms the 32-bit data with two 16-bit subkeys $KL_{i,1}$ and $KL_{i,2}$, using AND, OR, XOR, and 1-bit cyclic shift ($\lll 1$) operations. The FO function divides the 32-bit input data into two 16-bit blocks. Then, the left block is XORed with the 16-bit subkey $KO_{i,j}$, transformed by the FI function with a 16-bit subkey $KI_{i,j}$, and XORed with the right block. This routine is iterated three times with swaps of the left and right blocks. A 16-bit data block entering the FI function is divided into two smaller blocks for S-BOX transformations. The leftmost 9 bits become one block, and the rightmost 7 bits become another block. Then, they are transformed twice using the 9-bit S-BOX S9 and the 7-bit S-BOX S7, respectively. The two data blocks are XORed with each other. But, because the bit length is different, zero-extension (ZE) is done to the 7-bit blocks by adding two "0"s, and the two most significant bits of the 9-bit blocks are truncated (TR). In the middle of the 4-round network, an XOR operation is done with the 16-bit subkey $KI_{i,j}$ (where $KI_{i,j,1}$ is 7 bits and $KI_{i,j,2}$ is 9 bits). For more details, see Fig. 2.

3.2. *Key scheduling*

KASUMI[3,11,12,14] cipher uses a 128-bit key K. Each round requires a distinct 128-bit key which is derived from K. The 128-bit key is subdivided into eight 16-bit values K1, ..., K8 where $\mathrm{K} = \mathrm{K1}\|\mathrm{K2}\|\mathrm{K3}\|\cdots\|\mathrm{K8}$. For each integer j ($1 \leq j \leq 8$), a second array of subkeys, K'_j are derived ($K'_j = K_j \oplus C_j$). C_j is a predefined constant. The procedure of the key scheduling consists of circular rotation and XOR operation with the C_j values. Finally, a total 40 16-bit values subkeys are generated. For these subkeys, the round keys are created.

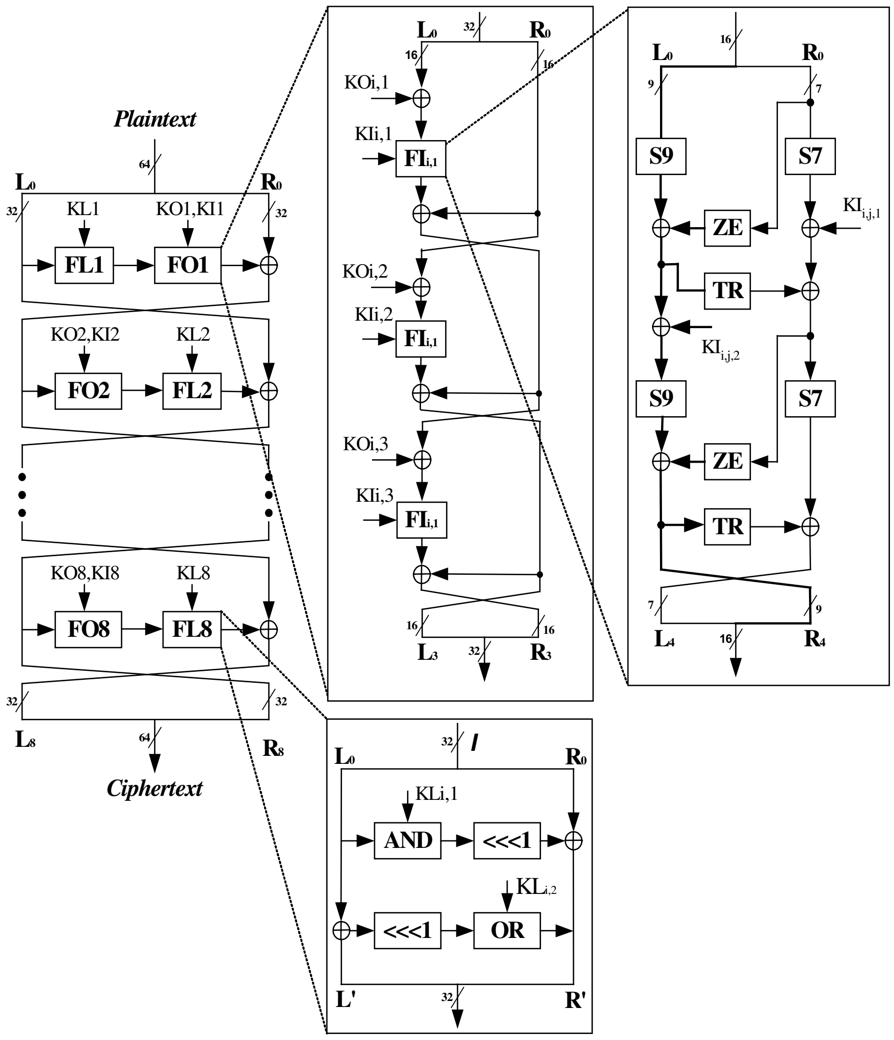


Fig. 2. The KASUMI encryption data path.

4. Hardware Implementation

4.1. *GEA3 keystream cipher implementation*

The proposed implementation consists of three basic subunits. The KASUMI block cipher, the GEA3 data mapping, and a control unit (Fig. 3). The GEA3 data mapping pads the KASUMI initial value and set the value of the counter BLKCNT. The control unit synchronizes the correct operation of the whole algorithm. The CA, CB, and CE parameters are standard and fixed. So, they are stored in the data mapping subunit.

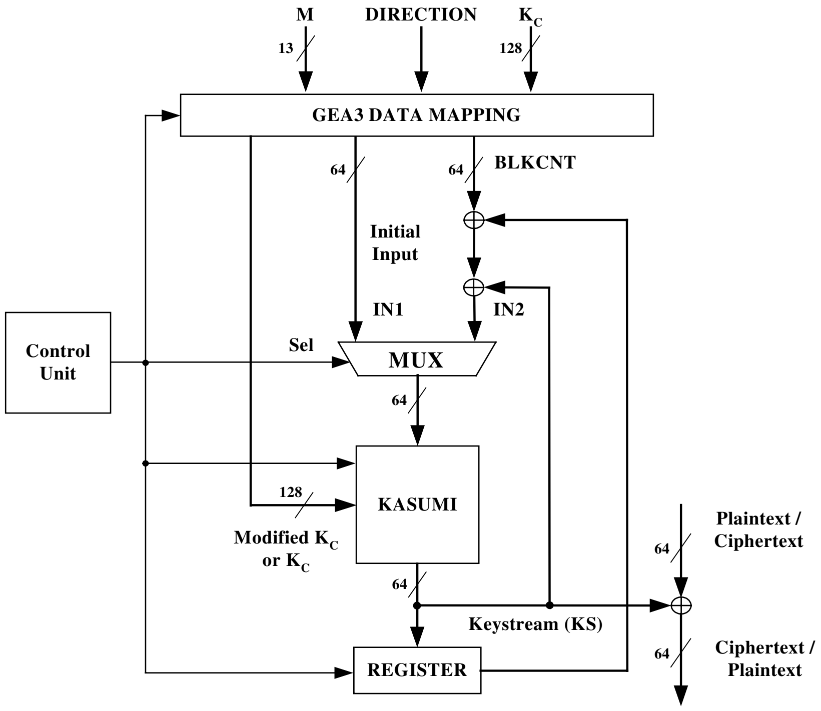


Fig. 3. The GEA3 algorithm implementation.

During the initialization process (first loop execution), the MUX subunit selects the IN1 (Initial Input) and the KASUMI module produces the initial Keystream (KS) by using the modified K_C. This initial KS is stored in the REGISTER and it is used for the next iterations. In all the next iterations, after the initial, MUX selects the second input (IN2) and the K_C is used by the KASUMI module. The Block Count (BLKCNT) counter is set initially to 0, and after each iteration, it is increased by one. The maximum value of the counter is $(8M/64)$ rounded up to the nearest integer, which is the number of iterations. The input M defines the plaintext/ciphertext length (# of bits).

4.2. *KASUMI block cipher implementation*

The proposed KASUMI cipher architecture consists of two main components. The Key Expansion Unit, which is responsible for the round keys generation, and the KASUMI Core, which executes the basic encryption procedure. Two different KASUMI Core architectures have been examined in order to provide useful information regarding the silicon area and throughput. In order to increase the system throughput, the first one has eight pipeline stages. The second reduces the silicon hardware resources by using only two pipeline stages. As it is previously mentioned, the even round of KASUMI cipher has different structure of the odd round. The odd rounds are denoted as Odd Round Cell (ORC) and the even rounds are denoted as Even Round Cell (ERC). In Fig. 4, the implementations of the ERC and ORC are shown.

For both implementations, a data block cipher execution needs eight clock cycles and the Key Expansion Unit is the same. The subkeys are produced and stored in a register file.

A Double Edge Trigger (DET) pipeline is used in each round. In a conventional Single Edge Triggered register design, the data are transferred between two successive registers in one clock period. In a DET pipeline, both rising and falling edges of the clock signal are used; so double transfer data are achieved. The use of DET pipeline technique has recently been proposed for low-power and high-speed designs.[15,16] In Fig. 4, the FO subfunction DET pipeline design is illustrated. The added register was inserted almost in the middle of the data path round.

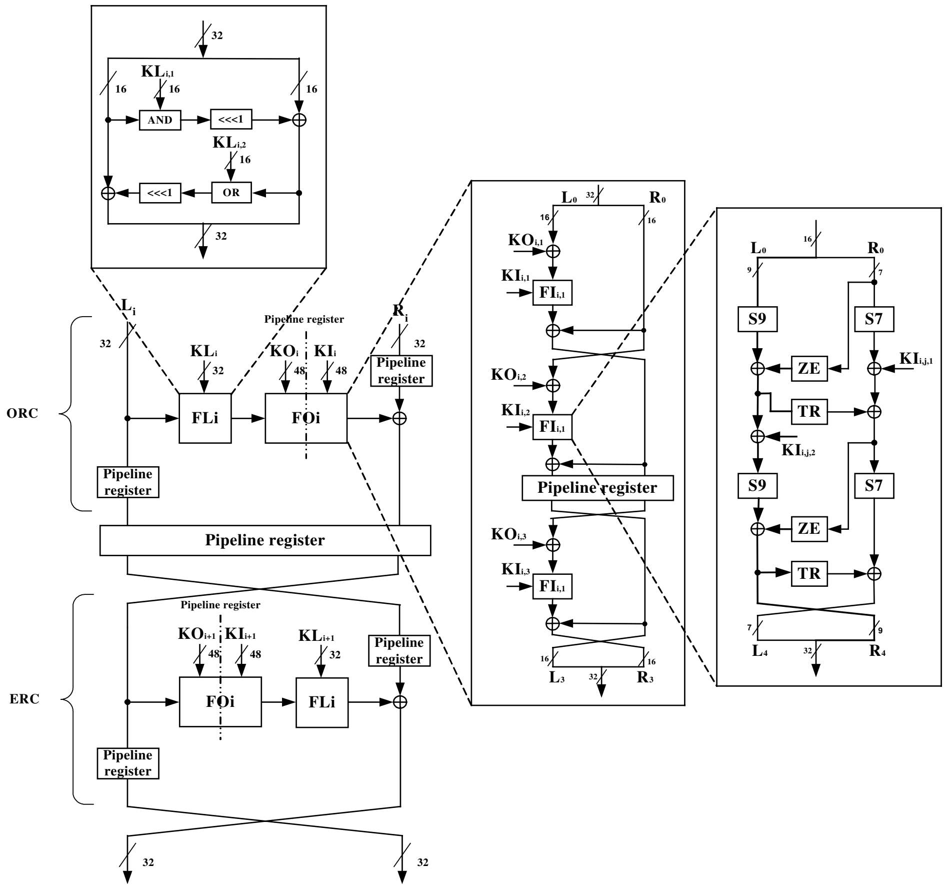


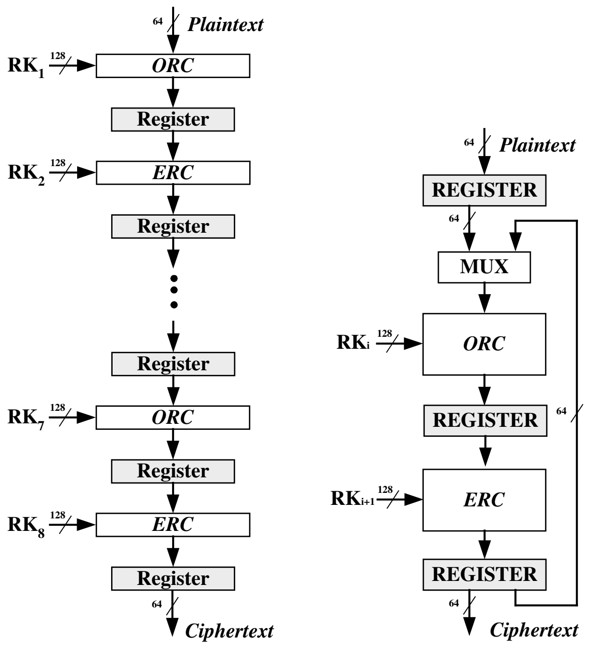
Fig. 4. The KASUMI round implementation.

The execution time of each round is one clock cycle. In order to synchronize the processing data, one similar register is inserted in the left and right branches (L_i and R_i) of each round. As a result, the clock period is reduced roughly to half. The major advantage of the DET pipeline is that the actual number of data in a pipeline is doubled without the increment of the total latency time. A small area penalty is paid due to the usage of the three extra pipeline registers.

When the clock frequency is determined by the system specifications, the usage of DET register can reduce the clock frequency roughly by half, for the same data throughput. As a result, the power consumption is reduced, making this technique desirable for low power applications.

In Fig. 5 the two KASUMI Core architectures are illustrated. The first architecture has eight pipeline stages between rounds. Pipeline registers are inserted between the rounds in order to temporally store the output of each round. With the introduced pipeline technique, the time performance is improved drastically. The cipher has the possibility to process 16 data block in one clock cycle with simultaneously increment of the cipher throughput. So, the latency of this architecture is eight clock stage cycles (without counting the input register delay). The main advantage of this architecture is the high processing throughput.

In order to reduce the silicon area, a two pipeline stage can be used. It is the concatenation of two basic cells, ORC and ERC, in an iterative scheme with an



(a) 8R full pipelining (b) The 2R loop unrolling

Fig. 5. KASUMI core architecture: (a) 8R Pipeline and (b) 2R Pipeline.

additional pipeline register between the two cells. This architecture requires a quarter hardware resources of the 8-round full pipeline architecture, at the expense of throughput. One more register is needed for the input blocks. Also, one multiplexer is necessary in order to combine the input block data with the previous block. Two data blocks can process simultaneously with the conventional pipeline, while four data blocks can process with the DET pipeline. The key scheduler forces one odd number round key and one even number round key at every cycle. The KASUMI cipher execution for one block is completed in four iterations.

The DET technique is essential for the ECB (Electronic Codebook) mode of operation where high data rate is needed. But, the OFB (Output Feedback) mode, which is required for the GEA3 algorithm, cannot be supported by the pipeline technique. So, in the GEA3 architecture, the two proposed KASUMI architectures are used but one data block was processed at every eight-clock cycles.

The proposed Key Expansion Unit architecture is illustrated in Fig. 6. In this implementation, shift registers are used in order to produce a number of subkeys. The symbol "$\lll n$" denotes n bits left rotation. The rest of the subkeys are generated by bit-wise XOR operations with the constants C_j. These constants are stored in the 8×16 bits ROM memory. In total, 40 16-bit subkeys are generated. With the appropriate concatenations of the subkeys, the round keys are generated.

Many of the subkeys are used more than one time. The round keys are computed and stored in a register file. As storage unit, a 52×16 bits register file is used. The appropriate subkeys are produced directly when the "Latch" signal is activate.

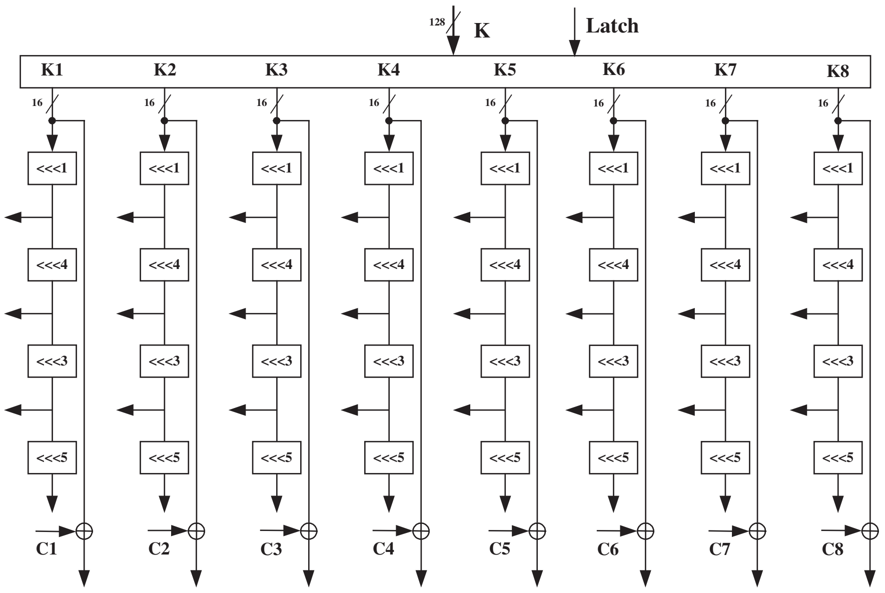


Fig. 6. The KASUMI key expansion unit architecture.

During the first clock cycle, the first round subkeys are set directly from the Key Expansion Unit to the data path, while the rest of the subkeys are stored in the register file. With this, the writing process of the register file does not add any additional delay.

The KASUMI specifications do not describe the decryption process because the GEA3 algorithm uses the KASUMI only in encryption mode. However, a general-purpose KASUMI implementation is proposed that supports both encryption and decryption. The only difference between the two modes is the order of the round keys. The first key in the encryption is the last key in the decryption. When decryption must be performed, the round keys are read by the register file in reverse order.

5. Results and Evaluation

The whole system (Fig. 3) was captured by using VHDL with structural description logic. All the VHDL code was simulated and verified by using the official test vectors, provided by the 3GPP standard.[17,18] The design was synthesized, placed, and routed using FPGA devices of XILINX.[19]

As it has been previously mentioned, the KASUMI specifications do not describe the decryption process because the GEA3 algorithm uses the KASUMI only in encryption mode. However, in this paper, a general-purpose KASUMI implementation is proposed that supports both encryption and decryption. This can be useful for bulk encryption cipher usage, in other applications beside GEA3 algorithm. The KASUMI key scheduling architecture is simple and faster than the data path. So, the encryption has the same throughput with the decryption operation. Finally, the proposed KASUMI architectures have the same throughput as the encryption mode.

Two alternative approaches are proposed for the KASUMI cipher S-BOXes implementation: (1) the ROM-based design, which guarantees high-speed performance and (2) the Combinational Logic (CL) design which minimizes the silicon area but with a performance delay penalty. For the last case, simple logical gates (ANDs and XORs) are used.

The synthesis results of the proposed KASUMI block cipher implementations for the two architectures 8R and 2R, and the two approaches CL and ROM are shown in Table 1.

Table 1. KASUMI FPGA synthesis results.

Architecture	CLBs	FGs	DFFs	ROM (Bytes)	F (MHz)	Device
8R_Comb.	4738	9479	2564	—	61	XCV400E-8BG432
8R_ROM	1574	2997	2562	2752	116	XCV200E-8FG456
2R_Comb.	1726	3452	1571	—	61	XCV200E-8FG456
2R_ROM	910	1819	1570	688	115	XCV200E-8FG456

For the S-BOXes implementation, methods based on the Look-Up-Table (LUT) synthesis[20,21] can be applied. In this paper, the two above mentioned methods are used.

Throughput comparisons between the proposed KASUMI implementations and previous implementations[4–7] are given in Table 2.

In Ref. 4, two architectures were proposed, one with eight-stage pipeline and the second with two rounds. In these designs, for the implementation of the KASUMI S-BOXes, Combinational logic and Lookup Table (LUT) are used. Similar architectures as in Ref. 4 were proposed in Ref. 5, and additionally another area efficient architecture for the $f8$ implementation was suggested. In Ref. 6, two implementation versions were proposed. The first implementation (Type1) achieved reduction of the power dissipation and the second (Type2) targeted high performance, with the usage of four-stage pipeline. Finally, in Ref. 7, a hardware implementation that reduces the hardware resources was presented. For this implementation, two different synthesis results were given (Synth1 and Synth2).

The proposed KASUMI architectures outperform all of the above previous implementations in terms of time performance. The synthesis results for the proposed GEA3 algorithm are given in Table 3. The clock frequency of the GEA3 algorithm implementation is roughly the same as the KASUMI cipher implementation. This is due to the function interface does not include heavy mathematical operations. In the GEA3 algorithm, the KASUMI works in OFB mode. One data block is processed after eight cycles.

Table 2. KASUMI time performance comparisons.

Architecture	F (MHz)	Throughput	Device
8R_Comb.[4]	20.86	1.34 Gbps	XCV400E-6BG432
8R_LUT.[4]	37.72	2.42 Gbps	XCV1000E-6BG560
2R_Comb.[4,5]	20.88	167.04 Mbps	XCV300E-6BG432
2R_LUT.[4,5]	35.35	70.70 Mbps	XCV200E-6FG456
Type1[6]	20	110 Mbps	—
Type2[6]	60	410 Mbps	—
Synth1[7]	33.14	265.12 Mbps	XCV300E-8BG432
Synth2[7]	28.38	227.04 Mbps	XCV300E-6BG432
Proposed 8R_Comb.	55.5	7.1 Gbps	XCV300E-8BG432
Proposed 8R_ROM	106	13.6 Gbps	XCV300E-8BG432
Proposed 2R_Comb.	54	1.73 Gbps	XCV300E-8BG432
Proposed 2R_ROM	105	3.36 Gbps	XCV300E-8BG432

Table 3. GEA3 FPGA synthesis results.

Architecture	CLBs	FGs	DFFs	F (MHz)	Device
8R_Comb.	4864	9735	2874	55	XCV400E-8BG432
8R_ROM	1708	3243	2874	104	XCV200E-8FG456
2R_Comb.	1858	3690	1789	52.5	XCV200E-8FG456
2R_ROM	1042	1813	1789	103	XCV200E-8FG456

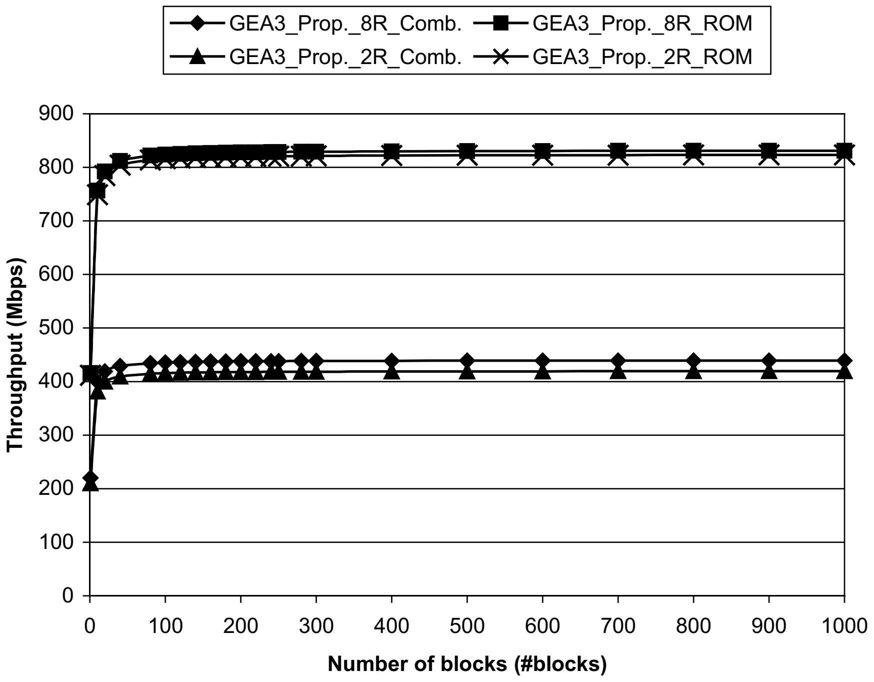


Fig. 7. GEA3 implementation throughput measurements.

In Fig. 7, the throughputs of the proposed GEA3 implementations are shown for different number of blocks.

As it is mentioned above, the GEA3 algorithm is almost the same as the UMTS Confidentiality algorithm $f8$. Because no other previous GEA3 implementations are referred, comparisons with the previous mentioned $f8$ implementation are made. Throughput comparisons between the proposed GEA3 implementations and the previous $f8$ implementations[4–7] are given in Table 4.

In Ref. 4, the 2-round and 8-round KASUMI versions were proposed. In the 8-round version, the $f8$ algorithm was implemented only by using combinational S-BOXes logic. The amount of the embedded memory that is required in order to implement the S-BOXes in LUTs exceeded the capacity of even the biggest part of the target FPGA device family. Our proposed GEA3 implementations outperform all the previous implementations.[4–7]

The GPRS performance[22] requirements demand 50 MHz mobile device clock speed. From Table 4, it is obvious that all the proposed GEA3 implementations meet this requirement.

6. Conclusions

In this paper, hardware implementations of the GEA3 algorithm are presented. This algorithm has been standardized by the 3GPP for the GPRS security. Different

Table 4. GEA3 time performance comparisons.

Architecture	F (MHz)	Throughput (Mbps)	Device
$f8$_8R_Comb.[4]	20.01	158	XCV1000E-6BG432
$f8$_2R_Comb.[4]	20.52	162.1	XCV300E-6BG432
$f8$_2R_LUT.[4]	33.14	261.8	XCV600E-6BG432
$f8$_2R_Comb.[5]	16.93	135	XCV300E-6BG432
$f8$_2R_LUT.[5]	46.56	372	XCV600E-6FG432
$f8$_Type1[6]	19.5	154	—
$f8$_Type2[6]	52	411	—
$f8$_Synth1[7]	30.12	238	XCV300E-8BG432
$f8$_Synth2[7]	25.80	204	XCV300E-6BG432
Proposed GEA3_8R_Comb.	55	434.5	XCV300E-8BG432
Proposed GEA3_8R_ROM	106	837	XCV300E-8BG432
Proposed GEA3_2R_Comb.	53.5	423	XCV300E-8BG432
Proposed GEA3_2R_ROM	104	822	XCV300E-8BG432

VLSI KASUMI implementations have been examined, providing useful information regarding the silicon area and throughput. By using the DET pipelining technique, significant throughput improvement is achieved. The S-BOXes have been implemented with combinational logic and ROM memories. The system was synthesized, placed, and routed by using FPGA devices. The proposed 8-Round KASUMI cipher implementation achieves throughput up to 13.6 Gbps and the 2-Round up to 3.36 Gbps. So, the GEA3 algorithm implementation throughput reaches the 822 Mbps (2-Round KASUMI version) and the 837 Mbps (8-Round KASUMI version). Since many GEA3 architectures have been proposed in this paper, each specific application can choose the appropriate speed-area trade-off architecture. The performance of the proposed hardware implementations is much higher than the 3GPP specifications demands. So, the proposed implementations are efficient solutions for devices, which provide GPRS services.

References

1. ETSI TS 148 018. Digital cellular telecommunications system (Phase 2+), General Packet Radio Service (GPRS); Base Station System (BSS) — Serving GPRS Support Node (SGSN), BSS GPRS Protocol, May 2002, http://webapp.etsi.org/action/PU/20020611/ts_148018v050300p.pdf.
2. ETSI/SAGE, Specification of the A5/3 Encryption Algorithms for GSM and EDGE, and GEA3 Encryption Algorithm for GPRS, Document 1: A5/3 and GEA3 Specifications, ETSI/SAGE, May 2002, http://www.3gpp.org/ftp/Specs/archive/55_series/55.216/.
3. ETSI/SAGE, Specification of the 3GPP Confidentiality and Integrity Algorithms, Document 2: KASUMI Specification, ETSI/SAGE, December 1999, http://www.3gpp.org/ftp/Specs/archive/35_series/35.202/.
4. K. Marinis, N. K. Moshopoulos, F. Karoubalis and K. Z. Pekmestzi, On the hardware implementation of the 3GPP confidentiality and integrity algorithms, *Proc. 4th Int. Conf. Information Security*, *ISC 2001*, Malaga, Spain, 1–3 October 2001, pp. 248–265.

5. K. Marinis, N. K. Moshopoulos, F. Karoubalis and K. Z. Pekmestzi, An area optimized hardware implementation of the 3GPP confidentiality and integrity algorithms, *Proc. 8th Conf. Optimization of Electrical and Electronic Equipment, OPTIM 2002*, Brasov, Romania, 16–17 May 2002.
6. H. Kim, Y. Choi, M. Kim and H. Ryu, Hardware implementation of 3GPP KASUMI crypto algorithm, *2002 Int. Technical Conf. Circuits/Systems, Computers and Communications* (*ITC-CSCC*), Vol. 1, 16–19 July 2002, Phuket, Thailand, pp. 317–320.
7. A. Satoh and S. Morioka, Small and high-speed hardware architectures for the 3GPP standard cipher KASUMI, *Proc. 5th Int. Conf. Information Security, ISC 2002*, Sao Paulo, Brazil, 30 September–2 October 2002, Lecture Notes in Computer Science, Vol. 2433 (2002).
8. Recommendation for block cipher modes of operation, methods and techniques, National Institute of Standards and Technology (NIST), Technology Administration, U.S. Department of Commerce, Special Publication, http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf.
9. ETSI/SAGE, Specification of the 3GPP Confidentiality and Integrity Algorithms Document 1: $f8$ and $f9$ specification, ETSI/SAGE, September 2000, http://www.3gpp.org/ftp/Specs/archive/35_series/35.201/.
10. M. Matsui, New block encryption algorithm MISTY, *Fast Software Encryption '97*, Vol. 1267 of LNCS (Springer-Verlag, 1997), pp. 54–68.
11. J. Wallén, Design principles of the KASUMI block cipher, HUT TML 2000, *Tik-110.501 Seminar on Network Security*, http://www.niksula.cs.hut.fi/∼jwallen/kasumi/kasumi.html.
12. 3rd Generation Partnership Project; Security Algorithms Group of Experts (SAGE); Report on the Evaluation of 3GPP Standard Confidentiality and Integrity Algorithms, 3GPP KASUMI Evaluation Report, http://www.3gpp.org/tb/other/algorithms/KASUMI_Eval_rep_v20.pdf.
13. B. Schneier, *Applied Cryptography-Protocols and Source Code in C*, 2nd edn. (John Wiley and Sons, New York, 1996).
14. Mitsubishi Electric Corporation's, Information Security Technology, MISTY, KASUMI, Camellia & Security Solutions, http://www.security.melco.co.jp/SecWWW/index/02e/melsecE.htm.
15. M. Afghahi and J. Yuan, Double edge-triggered D flip-flops for high speed CMOS circuits, *IEEE J. Solid-State Circuits* **26** (1991) 1168–1170.
16. W. Chung, T. Lo and M. Sachdev, A comparative analysis of low-power low-voltage dual-edge-triggered flip-flops, *IEEE Trans. Very Large Scale Integration* (*VLSI*) **10** (2002) 913–918.
17. ETSI/SAGE, Specification of the A5/3 Encryption Algorithms for GSM and EDGE, and the GEA3 Encryption Algorithm for GPRS Document 2: Implementors' Test Data, ETSI/SAGE, May 2002.
18. ETSI/SAGE, Specification of the A5/3 Encryption Algorithms for GSM and EDGE, and the GEA3 Encryption Algorithm for GPRS Document 3: Design Conformance Test Data, ETSI/SAGE, May 2002.
19. Xilinx Inc., San Jose, California, Virtex, 2.5 V Field Programmable Gate Arrays, 2003, www.xilinx.com.
20. J. Cong and Y.-Y. Hwang, Boolean matching for LUT-based logic blocks with applications to architecture evaluation and technology mapping, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems* **20** (2001) 1077–1090.

21. M. Rawski, L. Jozwiak and T. Luba, Functional decomposition with an efficient input support selection for sub-functions based on information relationship measures, *Elsevier J. Syst. Architectures* **47** (2001) 137–155.
22. ETSI/SAGE, Specification of the A5/3 Encryption Algorithms for GSM and EDGE, and GEA3 Encryption Algorithm for GPRS, Document 4: Design and Evaluation Report, ETSI/SAGE, May 2002, http://www.3gpp.org/ftp/Specs/archive/55_series/55.919/.