High Performance Encryption Cores for 3G Networks

Tomás Balderas-Contreras
Instituto Nacional de Astrofísica, Óptica y Electrónica
Luis Enrique Erro 1, 72840
Tonantzintla, Puebla. MEXICO
balderas@inaoep.mx

René Cumplido
Instituto Nacional de Astrofísica, Óptica y Electrónica
Luis Enrique Erro 1, 72840
Tonantzintla, Puebla. MEXICO
rcumplido@inaoep.mx

ABSTRACT

This paper presents two novel and high performance hardware architectures, implemented in FPGA technology, for the KASUMI block cipher; this algorithm lies at the core of the confidentiality and integrity algorithms defined for the Universal Mobile Telecommunication System (UMTS) standard. The first proposal is a pipelined design and the second implements an iterative approach. The throughput for these architectures turn out to be higher than the throughput achieved by other proposals.

Categories and Subject Descriptors

E.3 [**Data Encryption**]: standards; C.3 [**Special-Purpose and Application-Based Systems**]: real-time and embedded systems

General Terms

Algorithms, Performance, Design, Security

Keywords

3G, UMTS Security Architecture, KASUMI, FPGA

1. INTRODUCTION

The KASUMI block cipher was adopted by the 3rd Generation Partnership Program (3GPP) as the cornerstone of the UMTS *f8* confidentiality algorithm and *f9* integrity algorithm. The GSM (Global System for Mobile communications) A5/3 and the GPRS (General Packet Radio Service) GEA3 encryption/decryption algorithms rely on KASUMI as well

KASUMI has a Feistel structure comprising eight rounds, operates on 64-bit data blocks, its processing is controlled by a 128-bit encryption key K, and has the following additional features derived from its Feistel nature [1]: a plaintext block is the input to the first round, the ciphertext is the last round's output, the encryption key K is used to generate a set of round keys ($\{KL_i, KO_i, KI_i\}$) for each round i, each round computes a different function as long as the round keys are different and the same algorithm is used both for encryption and decryption.

Figure 1 shows the structure and components of the KASUMI block cipher. For odd rounds the round-function is computed by applying the FL function followed by the FO function. For even rounds the FO function is applied before FL. FL, shown in figure 1(d), is a 32-bit function made up of simple AND, OR, XOR and left rotation operations. FO, illustrated in figure 1(b), is also a 32-bit function having a three-round Feistel organization which contains one FI block per round. FI, see figure 1(c), is a non-linear 16-bit function having itself a four-round Feistel structure; it is made up of two nine-bit substitution boxes (S-boxes) and two seven-bit S-boxes. Figure 1(c) shows that data in the FI function flow along two different paths: a nine-bit long path (thick lines) and a seven-bit path (thin lines). Notice that in Feistel structures, such as the one used in this algorithm, each round's output is twisted before being applied as input to the following round. After completing eight rounds KASUMI produces a 64-bit long ciphertext block corresponding to the plaintext input block.

The rest of this document is organized as follows: section 2 describes the sequence of steps carried out to design each of the two architectures, section 3 describes the issues concerning the implementation of the designs in FPGA technology, provides the performance and area complexity results obtained and a comparison with related works. Finally, section 4 concludes.

2. DESIGN PRINCIPLES

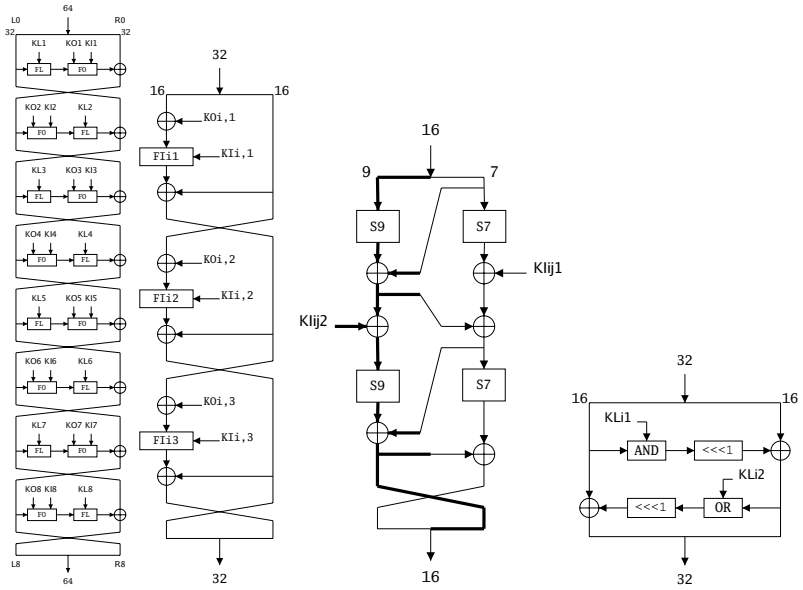
This section describes the main techniques conceived to design the two architectures.

2.1 The pipelined datapath

At every clock cycle the architecture receives a plaintext block as input and process it as it goes through the stages that make up the pipeline to produce a ciphertext block. The design can process different blocks simultaneously in the different stages of its datapath. At every clock cycle a ciphertext block leaves the pipeline from the last stage and the first stage receives a new input plaintext block. This architecture reaches the best performance owing to the exploitation of temporal and spatial parallelism when processing plaintext blocks.

Consider two instances of the FI block in figure 1(c), replace each pair of S9 S-boxes located in the same position in

DAC 2005, June 13–17, 2005, Anaheim, California, USA.



(a) Feistel (b) FO (c) FI (d) FL

Figure 1: The KASUMI block cipher.

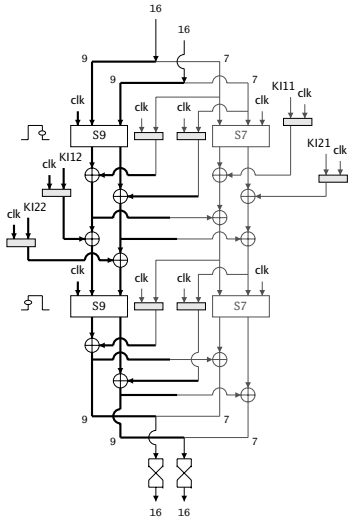
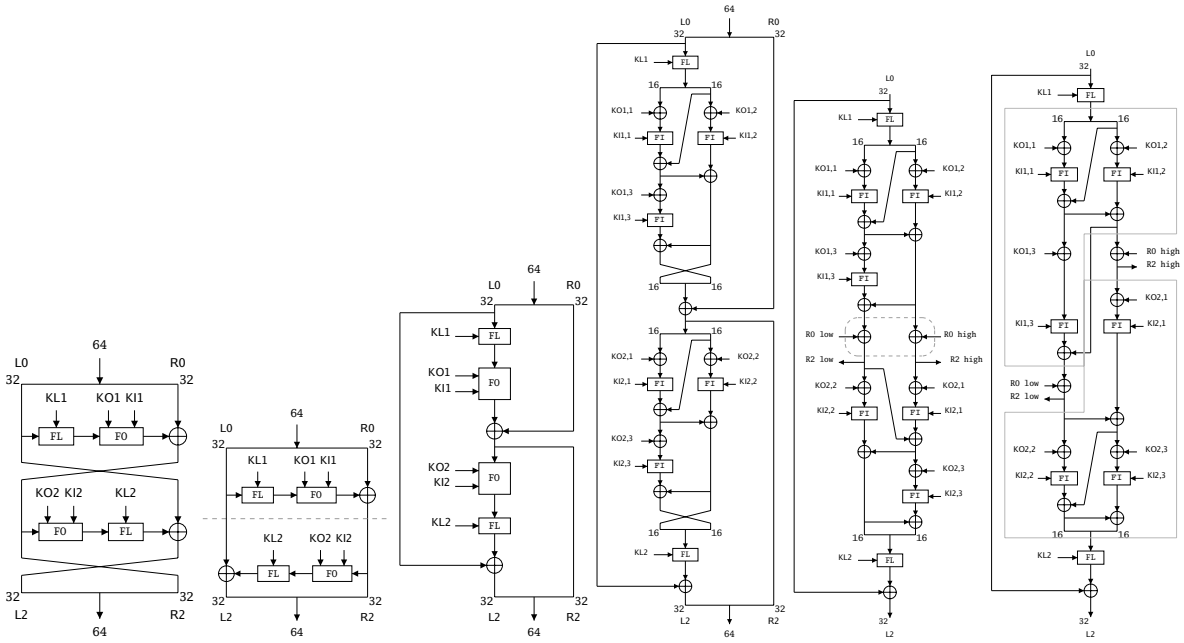


Figure 2: Datapath implementing the dual-port FI function.

both FI blocks by a single dual-port S-box, and repeat this procedure with the rest of the pairs of S7 S-boxes. The result is the datapath in figure 2, that only contains two dual-port S9 S-boxes and two dual-port S7 S-boxes and combines two FI functions in one. During implementation these four dual-port S-boxes are mapped to dual-port embedded memory blocks inside the FPGA (SelectRAM blocks). Since the embedded memory blocks are synchronous, and the dual-port FI datapath is required to provide its results after one clock cycle, the upper S-boxes are designed to be negative edge-triggered, whereas the lower S-boxes are designed to be positive edge-triggered. The first phase of the design process is complete.

Now consider a sequence of an odd round followed by an even round, as illustrated in figure 3(a). Figures 3(b) and 3(c) show two equivalent ways of representing the same two-round sequence. In figure 3(d) each of the FO boxes is unrolled and drawn in a way that highlights the parallel structure of the FO function. Figure 3(e) shows the result of splitting the 32-bit XOR gate located between the two FO function blocks into two 16-bit XOR gates and "unfolding" the datapath comprising the upper FO function block's output, the two 16-bit XOR gates and the lower FO function block in figure 3(d). Notice that either the 32-bit R0 input and the 32-bit R2 output are now split into two 16-bit lines and that the components to the left of the lower FO function in figure 3(d) appear to the right in figure 3(e) as a consequence of the unfolding action. Figure 3(f) shows the result of joining the two FO blocks; this highlights the parallelism between each pair of FI function blocks. These actions make up the second phase of the design process.

From the two-round datapath in figure 3(f) it is possible to derive the pipelined datapath in figure 4. At first each pair of parallel FI function blocks is replaced by a single dual-port FI block; in this case three replacements are needed. Next, a pair of registers, a negative edge-triggered register followed by a positive edge-triggered register, is added in every line surrounding the dual-port FI block to synchronize the corresponding data with the two values provided by the dual-port FI block. The resulting pipeline has four stages, meaning that it needs four cycles to perform two rounds of the ciphering process. Since the whole encryption process is carried out by the concatenation of four pipelined two-round datapath modules, identical to the one in figure 4, the architecture's initial latency is 16 clock cycles.



(a) Step 1 (b) Step 2 (c) Step 3 (d) Step 4 (e) Step 5 (f) Step 6

Figure 3: Sequence of steps to design a parallel datapath for a sequence of two rounds.

The key scheduler for the two-round architecture just described is also a four-stage pipelined design and provides each stage of the ciphering datapath with the round keys it needs and no more. Four instances of this key scheduling datapath must be connected in series to carry out the whole ciphering process.

2.2 The iterative approach

The basic block for this architecture is the datapath in figure 4, although not used in a pipelined fashion. A multiplexer is placed at each of the L0 and R0 input ports; these multiplexers select the input to the datapath out of two options: a new input plaintext block and the output value L2||R2, which is fed back to the datapath's input. A plaintext block travels alone through the datapath every clock cycle, requiring four clock cycles to reach the end of the two-round datapath and 16 cycles to complete the iterative eight-round ciphering process.

This architecture requires the design of a new key scheduler that issues and maintains during two cycles the set of round keys for the first round ($\{KL_1, KO_1, KI_1\}$) along with the 16 most significant bits of the KO_2 and KI_2 round keys. During the next two cycles it must issue and maintain the rest of the round keys for the second round, i.e. the 32 least significant bits of the KO_2 and KI_2 round keys and the KL_2 round key. The key scheduler designed to meet these requirements contains two left-rotate registers: one register stores the array of eight 16-bit subkeys ($K_i, i = 1, \ldots, 8$) that make up the encryption key K and the second register stores the array of fixed constants ($C_i, i = 1, \ldots, 8$) used to generate the round keys. Both of these registers are synchronized with a divide-by-two clock divider that allows the contents of the registers to be available during two clock cycles before being shifted. The registers must be pre-loaded with the encryption key K and the array of constants before any ciphering process is carried out.

3. IMPLEMENTATION

The design flow consisted on describing the architectures using VHDL, validating then using testbenches in the specifications and implementing then using Xilinx Synthesis Technology (XST) tools for Virtex-E FPGAs to make fair comparisions with other proposals.

3.1 Synthesis results

Table 1 shows the results provided by the XST after synthesizing the pipelined architecture for the XCV1000E-8-BG560 device and the same information for the iterative design in the XCV300E-8-BG432 device. The total number of slices in the reconfigurable fabric assigned to the designs does not surpass 31% in the pipeline case and the 20% in the iterative case. This indicates that the designs use the FPGA's hardware resources in an efficient manner.

Virtex-E devices contain large blocks of SelectRAM memory. Each of these blocks is a fully synchronous dual-port (True Dual Port) 4096-bit RAM with independent control signals for each port. Therefore, one SelectRAM block is enough to store an 896-bit dual-port S7 S-box, but the designs require two SelectRAM blocks to store a 4608-bit dual-port S9 S-box.

3.2 Comparison of results

Table 2 shows that the pipelined architecture described in this document has the best performance among all of the designs proposed and reaches the highest clock frequency

Table 2: Comparison of the architectures with other proposals.

Proposal	Initial latency (clock cycles)	Area (slices)	Frequency (MHz)	Throughput (Mbps)	Number of S-boxes	Number of SelectRAM blocks
Pipelined						
Work in [2]	32	1100	33	234	12	N/A †
Work in [4]	40	2213	37.72	2414	96	96
Work in [3]	8	9476	56	3584	96	N/A †
This work	16	3928	83.139	5321	48	72
Iterative						
Work in [2]	8	650	20	110	12	N/A †
Work in [4]	40	749	35.35	71	24	24
	56	368	68.13	78	2	N/A †
Work in [5]	32	370	58.06	117	4	N/A †
	8	588	33.14	266	12	N/A †
This work	16	625	79.453	318	12	18

†: N/A = Not applicable

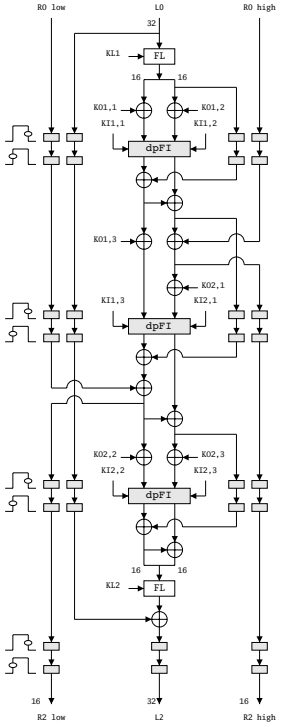


Figure 4: Pipelined datapath for the two-round sequence.

owing to its short critical path. Not only does it have a higher throughput than the design in [3], but is 2.4 times less expensive. The iterative architecture achieves higher throughput than the rest of the examined iterative proposals, this time due to its high clock frequency and its low latency. The table also shows that the least expensive designs in terms of hardware resources are the ones with the lowest performance.

4. CONCLUSION

This document described two novel proposals to implement the KASUMI block cipher in hardware: the first implements a pipeline technique that achieves the highest performance; the second implements an iterative strategy that reduces the number of hardware resources needed and has a high throughput. The techniques used during the design phase are the mapping of S-boxes to dual-port embedded memory blocks, the manipulation of the datapath to reveal and exploit parallelism, and the use of clock frequency dividers to simplify the key scheduler. The architectures are suitable for implementation in ASIC technology and integration into mobile terminals and network equipment to improve the level of security in the network. The proposed designs can also be integrated into complex processing elements as functional units.

Table 1: Synthesis results concerning area complexity for the architectures.

Category	Number of elements used	Total number of elements available	Percentage of use
Pipelined			
Slices	3928	12288	31%
Slice Flip Flops	6596	24576	26%
4-input LUTs	2146	24576	8%
SelectRAM blocks	72	96	75%
Iterative			
Slices	625	3072	20%
Slice Flip Flops	1018	6144	16%
4-input LUTs	649	6144	10%
SelectRAM blocks	18	32	56%

5. REFERENCES

[1] 3rd Generation Partnership Program. Document 2: KASUMI Specification. Technical Specification 35.202. Release 5. Version 5.0.0.

[2] H. Kim, Y. Choi, M. Kim and H. Ryu. Hardware Implementation of the 3GPP KASUMI Crypto Algorithm. In *ITC-CSCC-2002 Conference Proceedings*. 2002.

[3] P. Kitsos, M. D. Galanis and O. Koufopavlou. High-Speed Hardware Implementations of the KASUMI Block Cipher. In *ISCAS'04 Conference Proceedings*. 2004.

[4] K. Marinis, N. K. Moshopoulos, F. Karoubalis and K. Z. Pekmestzi. On the Hardware Implementation of the 3GPP Confidentiality and Integrity Algorithms, In *4th ISC 2001 Conference Proceedings*. 2001.

[5] A. Satoh and S. Morioka. Small and High-Speed Hardware Architectures for the 3GPP Standard Cipher KASUMI. In *5th ISC 2002 Conference Proceedings*. 2002.