International Conference on Electrical and Electronics Engineering (ICEEE) and X Conference on Electrical Engineering (CIE 2004)
Acapulco, Guerrero; Mexico. September 8-10, 2004

An Efficient Reuse-Based Approach to Implement the 3GPP KASUMI Block Cipher

Tomás Balderas-Contreras René A. Cumplido-Parra
Computer Science Department
Instituto Nacional de Astrofísica, Óptica y Electrónica
Tonantzintla, Puebla. Postal Code: 72840, MEXICO

***Abstract*— The KASUMI block cipher is found at the core of both the *f8* data confidentiality algorithm and the *f9* data integrity algorithm, which play an important role in the security architecture of modern third generation (3G) cellular communications networks. This paper describes a technique to design and implement the KASUMI block cipher using a principle based on the iteration over a minimum number of hardware components. The features of the technique include strategies to reduce and simplify KASUMI's regular Feistel structure, the use of frequency division and the exploitation of the facilities provided by the implementation technology: Field Programmable Logic Array (FPGA). The result is a digital system that reaches a higher throughput with fewer resources.**

I. INTRODUCTION

The recent advances in the field of cellular communications have led to the appearance of the third generation of cellular communications technology (3G). 3G is concerned with the transmission of both data and voice at data rates never seen before in any other cellular system and the provision of sophisticated and advanced services. In addition, 3G also allows cellular networks to access IP-based networks like Virtual Private Networks and the Internet. The most successful and promising kind of 3G network is called Universal Mobile Telecommunications System (UMTS) and has been in use for some years. UMTS' security architecture specifies that both its data confidentiality algorithm and its data integrity algorithm be based on a block cipher called KASUMI [2]. This paper presents an efficient FPGA implementation of this cipher that follows the principle of reusing components.

A. The KASUMI block cipher

KASUMI's specifications were developed by the Third Generation Partnership Program (3GPP) consortium [1] based on previous work carried out for MISTY [6], an algorithm that has proven its security against the most advanced cryptanalysis techniques and is suitable for hardware implementation. KASUMI has a Feistel structure comprising eight rounds, operates on 64-bit data blocks, and the processing is controlled by a 128-bit encryption key K. Additionally, KASUMI has the following features derived from its Feistel nature: input plaintext is the input to the first round, ciphertext is the last round's output, the encryption key 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.

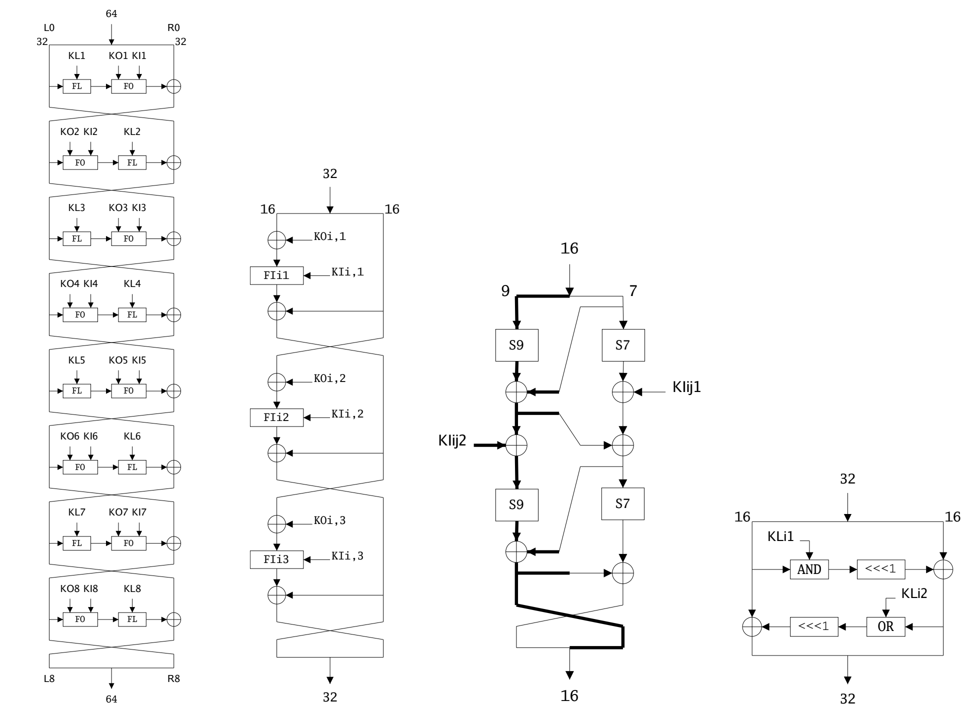
Figure 1 shows the structure and components of the KASUMI block cipher. For odd rounds the round-function is computed by applying the FL function followed by the FO function. For even rounds the FO function is applied before FL. FL, shown in figure 1*d*, is a 32-bit function made up of simple AND, OR, XOR and left rotation operations. FO, depicted 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 ones 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.

B. Related work

Several principles to implement a Feistel block cipher in hardware are described in [3]. The first choice is to implement only a small number N of rounds and then iterate over them, feeding back the output of the Nth round to the input of the first round until the required number of rounds have been executed. Improvements to this technique include the addition of inner- and outer-round pipeline registers. The second scheme consists in unrolling the whole number of rounds and adding inner- and outer-round pipeline stages to the design. While the first strategy is aimed to be used when area restrictions are strong, the second strategy is used to reach the maximum throughput possible without space restrictions.

The following proposals implement KASUMI using FPGAs and the reuse approach. The highest performance proposal in [4] implements a FO module having a four-stage inner-round pipeline and the rest of the logic needed to perform only one round. Eight iterations over these components, as well as 32 cycles, are needed in order to complete the block ciphering process. The system may process four blocks each iteration due to its pipelined design. The proposal in [5] that uses the reuse approach implements the logic to perform an odd round followed by an even round and

IEEE Catalog Number: 04EX865
ISBN: 0-7803-8532-2 ©2004 IEEE



a. Feistel structure *b*. FO function *c*. FI function *d*. FL function

Fig. 1. The KASUMI block cipher.

has registers in between the rounds to implement an outer-round pipeline scheme. The mapping of S-boxes to FPGA's embedded memory blocks with registered outputs introduces additional inner-round pipeline stages and thus increases the latency to 40 cycles. Two architectures that exploit the iteration and reuse principle are described in [7]. The first architecture includes logic to implement a single round with very few components, in such a way that 7 cycles are needed to perform a round and 56 cycles to complete the ciphering process for one block. The second architecture requires 4 cycles per round and 32 cycles to complete the processing for one block.

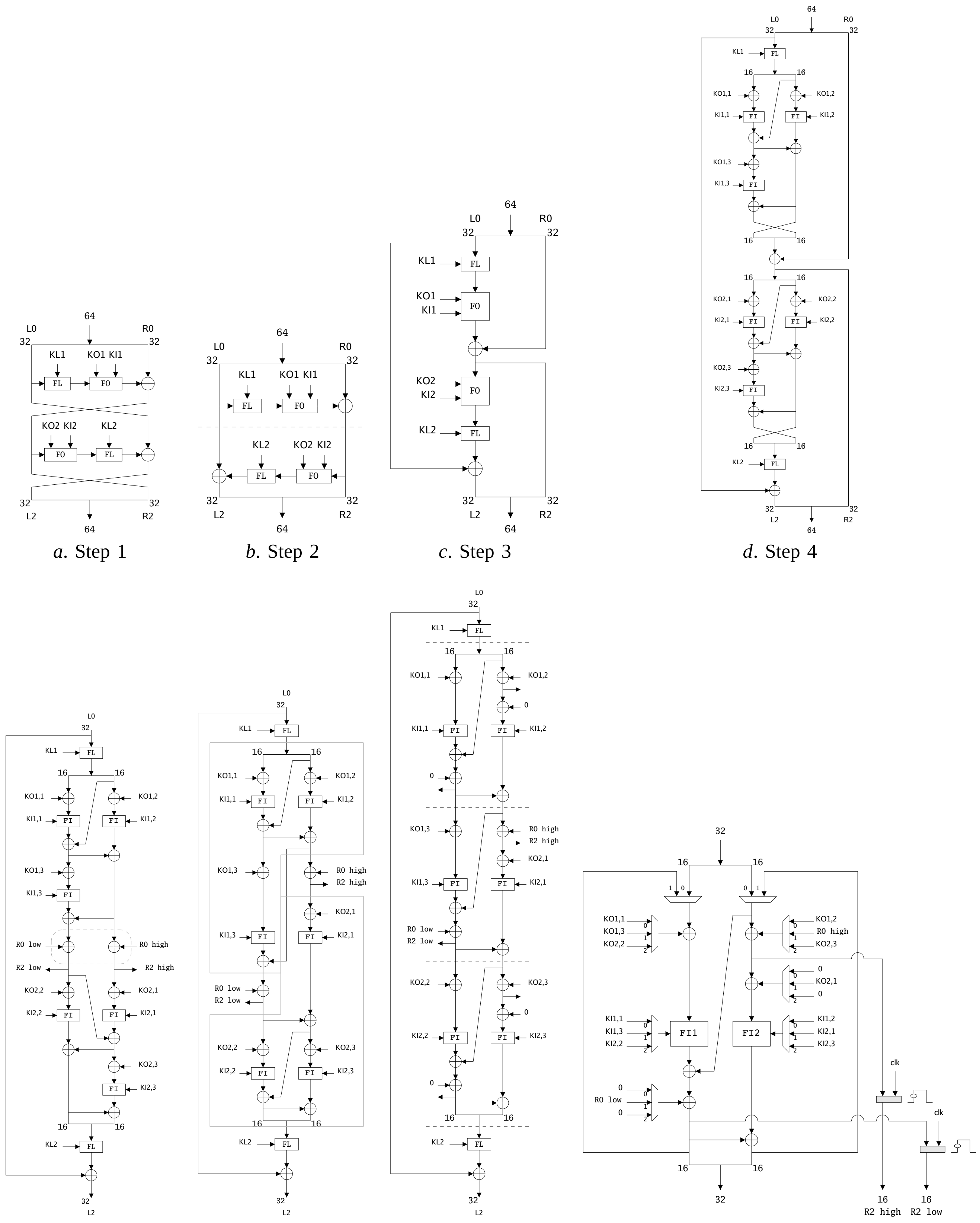
The goal of the design described in this document is to reach a good balance between high performance and small area by using the strategy of iteration over a simplified round logic. The rest of this paper is organized as follows: section II describes the architecture proposed and the techniques used in its design; section III provides information concerning the implementation in the FPGA platform and a comparison of the results obtained with those reported for the other proposals; section IV discusses some future work directions; finally, section V concludes.

II. DESIGN STRATEGIES

This section describes the main techniques used to design the system's architecture, and shows the resulting datapaths for each component.

A. Joining two FO function components

The first strategy considers a pair of consecutive rounds, an odd round followed by an even round, manipulates the structure of this pair without altering its effects, adds components that balance the structure and discovers a design pattern that repeats so often. This pattern then turns into the basic building block that is implemented once and then reused until completion of the ciphering process. The development of this strategy is depicted, step by step, in figure 2. Figures 2*a*–2*c* show exactly the same two-round sequence in three different ways. In figure 2*d* both FO black boxes are replaced by a parallel description for FO function, which is equivalent to that shown in figure 1*b*. Figure 2*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 2*d*. Notice that both 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 2*d* appear to the right in figure 2*e* as a consequence of the unfolding action. Figure 2*f* shows the result of doing a joint of the two FO function blocks, in order to highlight the parallelism between each pair of FI function blocks. Some 16-bit XOR gates with one zero input are added along the datapath in certain places so that the datapath can be divided in three structurally similar sections. Figure 2g shows the



a. Step 1 *b*. Step 2 *c*. Step 3 *d*. Step 4

e. Step 5 *f*. Step 6 *g*. Step 7 *h*. Step 8

Fig. 2. Sequence of steps to design a reusable datapath for the FO function.

result of this step: the sections between dashed lines are structurally identical to each other due to the additional XOR gates, which do not modify the datapath's behavior. The sections just described contain the design pattern to be used as the basic component for the system, this component takes three cycles to perform the operations corresponding to two consecutive FO functions, one for the odd round and the other for the even round. Figure 2*h* shows the basic FO module, called superFO, and the surrounding logic needed to provide the appropriate inputs each cycle. A three-state finite state machine issues the signals that control the multiplexors, and the two positive edge-triggered registers delay the R2 output one cycle, this is required because R2 is computed during the second cycle.

B. Optimizations in the FI function

Figure 2*h* shows that two FI blocks are required to carry out each section of the superFO module. As can be noticed in figure 1*c*, an FI function contains two seven-bit S-boxes and two nine-bit S-boxes. Therefore, this datapath that takes three cycles to complete the FO functions for two rounds requires a total number of eight S-boxes. It is possible to map the S-boxes to four 128×7 ROMs and four 512×9 ROMs. However, a decrease by two in the number of ROMs required is achieved by implementing S-boxes as dual-port ROMs. The use of this technique exploits the principle of reuse even further because the same S-box is now able to meet two requests at the same time.

Consider two instances of the FI block depicted in figure 1*c*; next replace each pair of S9 S-boxes located in the same position in both FI blocks by a single dual-port S-box, and repeat this procedure with the rest of the pairs of S9 S-boxes and the pairs of S7 S-boxes. The result is the datapath illustrated in figure 3, which only contains two dual-port S9 S-boxes and two dual-port S7 S-boxes, and combines two FI functions into one. As before, the thick lines highlight the flow of nine-bit long signals and the thin lines indicate the flow of seven-bit long signals. There are several notes that must be pointed out concerning this design. First, the four dual-port ROMs used to implement the S-boxes are intended to be mapped to embedded memory blocks inside FPGAs during the implementation phase. Second, the common situation is that these embedded memory blocks are synchronous, and since this dual-port FI datapath is required to provide its results within the range of one clock cycle, the upper S-boxes are designed to be negative edge-triggered, and the lower S-boxes are designed to be positive edge-triggered, as indicated in figure 3. Third, there are plenty of registers throughout the design depicted in figure 3, they are colored in grey and their purpose is to synchronize input data with the values provided by the upper and lower S-boxes.

C. Assembling the components

An improved superFO module which includes the dual-port FI component is depicted in figure 4. The dual-port FI module outputs its results at every positive clock edge, so the additional registers are needed to delay data, synchronizing them with dual-port FI's outputs. Figure 5 shows the complete datapath that performs the operations corresponding to two consecutive rounds. This architecture takes three cycles to complete two rounds, performs the block ciphering process in 12 cycles, and requires that two sets of round keys ($\{KL_1, KO_1, KI_1\}$ and $\{KL_2, KO_2, KI_2\}$), corresponding to two rounds, be available during three cycles. The control for this datapath consists of a 12-state finite state machine that sets the multiplexors' select inputs properly.

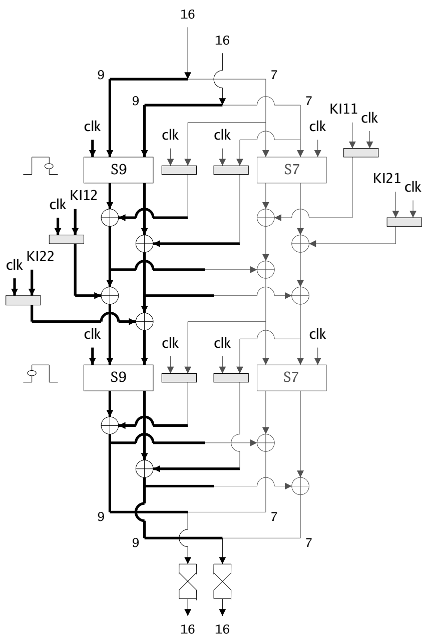


Fig. 3. Datapath implementing the dual-port FI function.

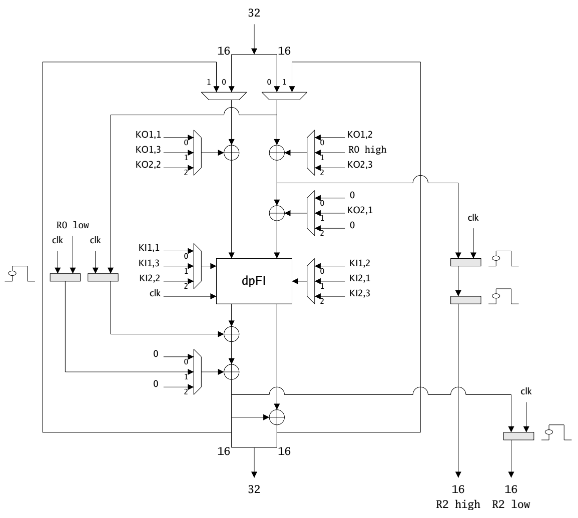


Fig. 4. Datapath for superFO including a dual-port FI module.

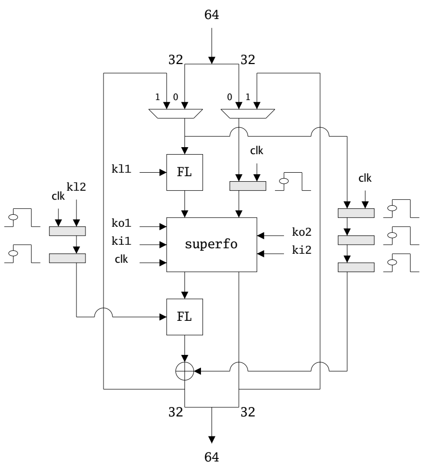


Fig. 5. Reusable datapath implementing operations for two rounds.

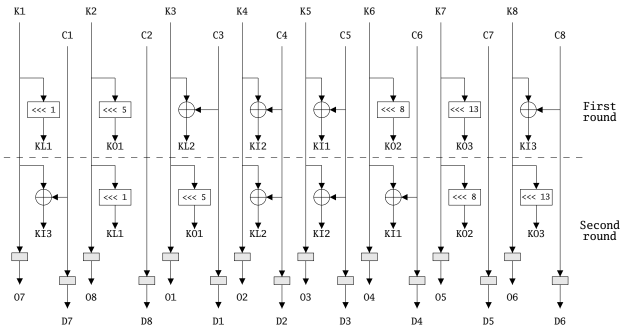


Fig. 6. The key scheduler.

D. The key scheduler

The key scheduler has the responsibility of providing each round with the set of round keys it needs to perform its task. The key scheduler for this project must provide the reusable datapath with two sets of round keys, one for each round, and maintain these sets of keys during three cycles. The module depicted in figure 6 meets the requirement of providing two sets of round keys since it contains two replicas of the components used to generate a set of round keys. Adding logic to keep the output values during three cycles is too much expensive; therefore, an alternative solution is used. The key scheduler is synchronized with a clock signal whose frequency is one third the frequency of the overall system's clock. A divide-by-three frequency divider, implemented as a three-state finite state machine, generates the appropriate clock signal for this module. The key scheduler receives the encryption key K as an array of eight 16-bit input values $(K_1, K_2, \ldots, K_8)$, which are used to generate the two sets of round keys, and issues these same values rotated to the left twice every positive clock edge. The key scheduler module is reused in this project by feeding the rotated outputs back to the inputs.

TABLE I

SYNTHESIS RESULTS CONCERNING AREA COMPLEXITY.

Category	Amount of elements used	Total amount of elements available	Percentage of use
Slices	566	3072	18%
Slice Flip Flops	546	6144	8%
4-input LUTs	1014	6144	16%
SRAMs	6	32	18%
GCLKs	1	4	25%

III. IMPLEMENTATION

FPGA is the chosen implementation platform due to its proven advantages, such as fast prototyping and advanced reconfigurability. The resources used to implement the design just described are: the VHDL hardware description language, an FPGA platform from Xilinx and the Xilinx Synthesis Technology (XST) software synthesis tools. The device of choice is the XCV300E-8-BG432 belonging to the Virtex-E family of devices. The main reason for choosing this family is that the designs reported in the related papers are implemented on Virtex-E devices, so in order to make a fair comparison the design reported here is implemented on a Virtex-E device as well.

Virtex-E devices introduce large blocks of SelectRAM (SRAM) memories. Each of these blocks is a fully synchronous dual-port (True Dual Port) 4096-bit RAM with independent control signals for each port. The data widths of the two ports can be configured in an independent fashion. The device considered in this document has 32 SelectRAM blocks, which provide a total amount of 131072 bits of embedded memory to hold whatever is convenient.

Table I shows the results of the synthesis process, carried out with the goal of optimizing speed. Table II summarizes the most important information concerning the published works and our own design.

A. Synthesis results

Notice in table I that the number of resources belonging to the reconfigurable fabric, i.e. slices, flip-flops and LUTs, that are occupied by the design does not surpass 20% in each category. While the fully unrolled KASUMI block cipher requires a total number of 96 S-boxes the design described in this document requires only four dual-port S-boxes, which is a significant saving no matter if the S-boxes are implemented as combinational logic or as embedded block memories. Table I reports that 6 SelectRAM blocks are used instead of four because two memory blocks are needed to hold each S9 S-box. A S9 S-box requires storage for 4608 bits, which can not be provided by a single 4096-bit SelectRAM memory block. XST reports an estimated operational clock frequency of 41.625 MHz for the whole design. From this information it is possible to compute the throughput, which turns out to be 222 Mbps.

TABLE II

COMPARISON WITH OTHER IMPLEMENTATIONS.

Proposal	Latency (cycles)	Area (slices)	Frequency (MHz)	Throughput (Mbps)	Hardware efficiency (kbps/slice)	Number of S-boxes	Number of SRAMs
Work in [7]	56	368	68.13	77.86	211.58	2	N/A †
	32	370	58.06	116.12	313.84	4	N/A †
Work in [5]	40	749	35.35	70.70	94.39	24	24
Work in [4]	pipeline	1100	33	234	212.73	12	N/A †
This work	12	566	41.625	222	392.22	4	6

†: N/A = Information not available

B. Comparison

The use of SelectRAM blocks removes complexity from the reconfigurable fabric. Among the two designs that use memory blocks, our architecture uses the least number of them. Table II shows that the design with the highest throughput in [4] is the one that consumes the greatest number of resources, being almost twice more expensive than our architecture in terms of area complexity, measured in number of slices. In addition, the proposal in [4] has a throughput which is slightly higher than our design's throughput, which in turns is much higher than the throughput reported for the rest of the three proposals. The two architectures described in [7] are the cheapest ones in terms of area because they reuse small basic components. However, in spite of their high clock frequencies, they do not achieve higher performances due to their large latencies. The higher the number of cycles needed for completion the lower the throughput, as indicated by the following expression:

$$throughput = \frac{block\ size \times clock\ frequency}{latency}.$$

Our simplified architecture has the lowest latency, only 12 cycles, and, at the same time, a short critical path which contrasts with those for the design in [4], which implements a complete FO function, and the architecture in [5], which implements two fully unrolled rounds.

In [4], an additional design using the reuse approach with a performance of 110 Mbps fitting in 560 slices is proposed. The pipelined architecture, issuing a ciphertext block each clock cycle, is reported here to show that our simplification strategy produces a highly competitive alternative.

IV. FUTURE WORK

Future work includes the integration of the architecture as a coprocessor in a CPU that is used within UMTS components such as mobile stations and Radio Network Controllers (RNC).

V. CONCLUSIONS

An efficient and novel hardware architecture for the KASUMI block cipher that exploits reutilization of components was described. The design strategies used for this design are: the simplification of the logic for a two-round sequence, the use of dual-port embedded memory blocks to implement S-boxes and the division of clock frequency to meet the requirements imposed to the key scheduler and keep it simple. The results of the implementation process show that the architecture's design is a good balance between high performance and low area complexity.

VI. ACKNOWLEDGMENTS

This work was sponsored by the scholarship number 171498 granted by CONACyT.

VII. REFERENCES

[1] 3rd Generation Partnership Program. 3GPP Home Page. http://www.3gpp.org

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

[3] P. Chodowiec, P. Khuon and K. Gaj, "Fast Implementation of Secret-Key Block Cipher Using Mixed Inner- and Outer-Round Pipelining", *in Proc. of the ACM/SIGDA 9th International Symposium on Field Programmable Gate Arrays*, Monterey, CA, 2001, pp. 94–102.

[4] H. Kim, Y. Choi, M. Kim and H. Ryu. "Hardware Implementation of the 3GPP KASUMI Crypto Algorithm", *in Proc. of the 2002 International Technical Conference on Circuits/Systems, Computers and Communications*, Phuket, Thailand, 2002, pp. 317–320.

[5] K. Marinis, N. K. Moshopoulos, F. Karoubalis and K. Z. Pekmestzi, "On the Hardware Implementation of the 3GPP Confidentiality and Integrity Algorithms", *in Proc. of the 4th International Conference on Information Security*, Málaga, Spain, 2001, pp. 248–265.

[6] M. Matsui, "New Block Encryption Algorithm MISTY", *in Proc. of the 4th International Fast Software Encryption Workshop*, Haifa, Israel, 1997, pp. 54–68.

[7] A. Satoh and S. Morioka, "Small and High-Speed Hardware Architectures for the 3GPP Standard Cipher KASUMI", *in Proc. of the 5th International Conference on Information Security*, Sao Paulo, Brazil, 2002, pp. 48–62.