International Journal of Networking and Computing – www.ijnc.org
ISSN 2185-2839 (print) ISSN 2185-2847 (online)
Volume 2, Number 1, pages 56–78, January 2012

GALS-based LPSP: Performance Analysis of a Novel Architecture for Low Power High Performance Security Processors

HALA A. FAROUK
Computer Engineering, Arab Academy for Science and Technology
Alexandria, Egypt

MAHMOUD T. EL-HADIDI
Electronics and Electrical Communications, Cairo University
Giza, Cairo, Egypt

AHMED ABOU EL-FARAG
Computer Engineering, Arab Academy for Science and Technology
Alexandria, Egypt

Received: July 23, 2011
Revised: October 31, 2011
Accepted: December 15, 2011
Communicated by Akihiro Fujiwara

Abstract

The past two decades have witnessed a revolution in the use of electronic devices in our daily activities. Increasingly, such activities involve the exchange of personal and sensitive data by means of portable and light weight devices. This implied the use of security applications in devices with tight processing capability and low power budget. Current architectures for processors that run security applications are optimized for either high-performance or low energy consumption. We propose an implementation for an architecture that not only provides high performance and low energy consumption but also mitigates security attacks on the cryptographic algorithms which are running on it. The proposed architecture of the Globally-Asynchronous Locally-Synchronous-based Low Power Security Processor (GALS-based LPSP) inherits the scheduling freedom and high performance from the dataflow architectures and the low energy consumption and flexibility from the GALS systems. In this paper, a prototype of the GALS-based LPSP is implemented as a soft core on the Virtex-5 (xc5-vlx155t) FPGA. The architectural features that allow the processor to mitigate Side-Channel attacks are explained in detail and tested on the current encryption standard, the AES. The performance analysis reveals that the GALS-based LPSP achieves two times higher throughput with one and a half times less energy consumption than the currently used embedded processors.

Keywords: Security Processor, Globally-Asynchronous Locally-Synchronous (GALS), High Performance Processor, Low Power Processor, DPA Countermeasure

1 Introduction

Cryptographic algorithms can be regarded as a series of transform functions that acquire an input data block and produce an output data block that have been operated on by some function. As a result, dataflow graphs lend themselves to the description of such transforms. Hence it is expected that dataflow architectures would better serve the performance of cryptographic algorithms than control-oriented architectures.

Transport-Triggered Architecture (TTA) is one of the dataflow architectural paradigms that are a mirrored paradigm of the von Neumann model [4]. Instead of an operation-triggered architecture, the TTA proposes a transport-triggered architecture in which the operation occurs as a side effect of data transfer. This paradigm decreases the complexity of the control circuits in the processor and can therefore allow for shorter cycle times [5]. This prospect of higher performance is proven in the work done by Hamalainen et al. [13], although it is unfortunate that energy consumption is not studied in that work. The energy consumption of the TTA is studied in [16] where some of the internal function units have been implemented in asynchronous fashion. The asynchrony in [16] has been used just for power reduction and not to mitigate side channel attacks. An example for the use of Globally Asynchronous Locally Synchronous in a cryptographic circuit as side channel attack countermeasure is given by [12]. However, the circuit in [12] is not for a programmable processor but for the implementation of the Advanced Encryption Standard (AES) [6] algorithm only.

We have combined all the advantageous features of the above-discussed work into the GALS-based LPSP. An emulator for this processor has been developed in [10] to verify and test its performance. However, in this paper, a hardware implementation on an FPGA is discussed and additional architectural features are added to secure the processor against Side-Channel attacks; particularly the Differential Power Analysis (DPA) attack. Section 2 provides a description of the architecture and the additional features that make it secure. Section 3 explains how the additional features in the novel architecture countermeasures the DPA attack and simulated DPA attack on the processor while running the AES is performed and explained. Section 4 analyzes the performance of the processor while executing the AES, RC6, TwoFish, RSA, and ECC algorithms. This section also shows the concurrency between the function units' execution and explains the relation between the asynchrony and power dissipation. An excerpt of the implementation report and a floorplan view of the processor on the Virtex-5 (xc5-vlx155t) FPGA is given in Section 5. The processor is compared to other soft cores in Section 6. Finally a summary and our conclusions come into view in Section 7.

2 GALS-based LPSP Architecture

The GALS-based LPSP is composed of six regions as shown in Figure 1. Each of these regions is governed by a different clock source and therefore a different clock frequency. The six regions communicate therefore internally in a synchronous fashion but communicate among themselves in an asynchronous fashion. Region 2, 3, 4, and 5 combine the Function Units (FUs) that are customized to serve cryptographic algorithms. These regions are called the execution regions. The execution regions receive instructions from Region 1 in the form of commands. The format of the command is shown in Figure 2. These commands are called move-commands. The move-command specifies the function unit at which the operation should be performed (Source Information) and the function units to which the result should be directed (Destination Information). The destination function unit upon receiving the required number of operands and also the corresponding move-command is triggered to perform its operation and in turn sends its result to the next destination function units. A single command can specify up to two destination function units. The move-command, not only specifies the destination function units, but also the tag of the result. The tag is used by the function unit to match the move-commands arriving from Region 1 and corresponding operands arriving from other function units. The tags allow for a fully distributed control and freedom in the order of arrival of the commands and operands. Each function unit owns a buffer in which data and commands are stored and matched according to their tags. The size of each of these buffers is critical to a deadlock-free operation and therefore prior to the architecture implementation an

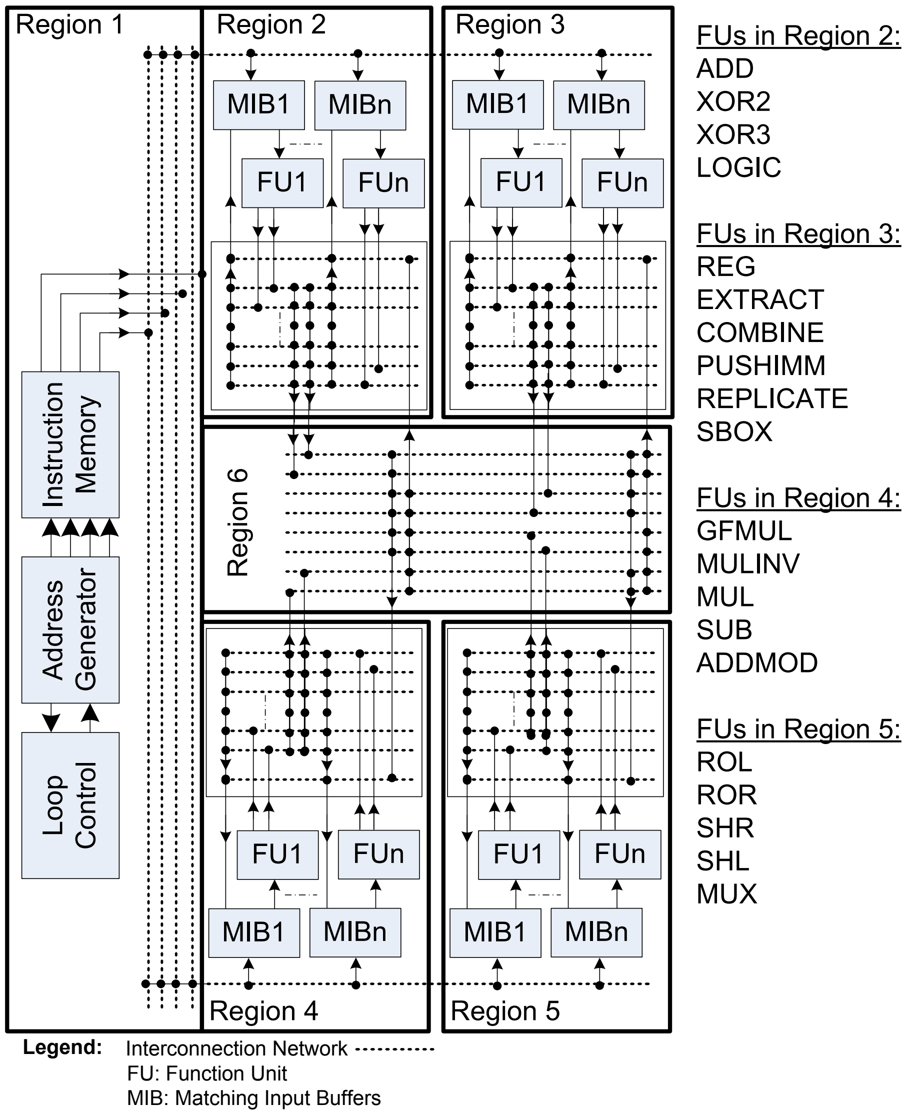

Figure 1: Block diagram of the proposed architecture.

emulator has been developed in [10] to determine the number of these buffers and the size of the tag fields.

In [10] and [9] the architecture of the GALS-based LPSP is given in details. However, for the completeness of this paper, we summarize the function units encompassed in the GALS-based LPSP.

2.1 GALS-based LPSP Function Units

We have decided on the set of FUs shown in Table 1 by studying the AES candidates, Rijndael, Serpent, Twofish, RC6, and MARS, and some older algorithms, the IDEA, and DES, and the SHA-1 as a hash algorithm, and the RSA and ECC representing asymmetric algorithms, to extract the common operations required.

The GALS-based LPSP command set can be logically grouped into seven groups of operations as shown in Table 1.

1. The arithmetic group includes the integer addition (ADD) and integer subtraction (SUB), and integer multiplication (MUL). All of these operations can be performed on 4-bit, 8-bit, 16-bit

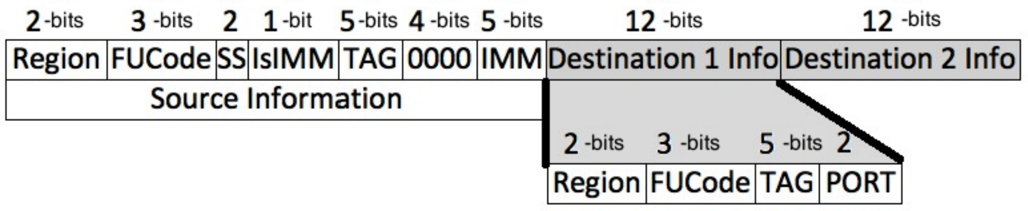

Figure 2: Move-command format of GALS-based LPSP.

Table 1: GALS-based LPSP Command Set

Arithmetic			
ADD	SUB	MUL	
Finite-Field Arithmetic			
ADDMOD	MULINV	GFMUL	WRITEPPOLY
Logic			
XOR2	XOR3	NAND	OR
Shift and Rotation			
ROR	ROL	SHR	SHL
Permutation			
EXTRACT	COMBINE		
Forwarding			
MUX	REPLICATE	PUSHIMM	
Reading and Writing			
READREG	WRITEREG	READSBOX	

and 32-bit operands except for the MUL that can manipulate up to 16-bits operands. The most-significant bit of the operand is reserved for the carry or borrow bit. The most-significant bit of a 4-bit operand is the fourth bit and for an 8-bit operand it is the eighth bit and so on. The most-significant bit changes its location according to the operand size since the operand size deactivates the unused parts of the bus to reduce the consumed power.

2. The second group includes the operations on the finite field. Abstractly a finite field or Galois field, named in honor of Evariste Galois, consists of a finite set of objects called field elements together with the description of two operations - addition and multiplication - that can be performed on pairs of field elements [17]. Each finite field is based on a primitive or irreducible polynomial $p(x)$ of degree n that exists in $\mathbb{F}_q$. Dividing the field $\mathbb{F}_q$ by this irreducible polynomial produces a finite field of size q, where $q = p^n$. The primitive polynomial is incorporated in every operation over the finite field (Galois field).

   - The ADDMOD is an addition modulo x. This type of addition can be used over the prime finite fields with x being the odd prime finite field generator.
   - The MULINV-FU performs the multiplicative inverse over the characteristic 2 finite fields.
   - The GFMUL performs a multiplication over the characteristic 2 finite fields. The primitive polynomial used for these operations is located in an internal register in the same region

as the FUs, namely Region 4.

- The register, which holds the primitive polynomial, is loaded during configuration time and can be overwritten using the WRITEPPOLY move-command. The hardware design of this group is adopted from [3], [23], [11].

3. The Logic group of move-commands includes operations that perform the basic logic operations.

    - The XOR2 performs a logical exclusive-OR between two operands.
    - The XOR3 performs the same operation but between three operands instead of two.
    - The NAND denotes an inverted-AND, which is a universal gate from which all other logic functions can be created.
    - The OR is included to simplify the multiple-precision algorithms [20].

4. The Shift and Rotation group includes the ROR, ROL, SHR and SHL commands.

    - The ROR denotes a rotation to the right.
    - The ROL denotes a rotation to the left.
    - The SHR denotes a logical shift to the right.
    - The SHL denotes a logical shift to the left.

    The shifting and rotation operations are performed in a single cycle using barrel shifters. Simple permutations can be done using the shift and rotation commands, however, there are other commands that perform more complex permutations and therefore are grouped into the Permutation group.

5. The Permutation group includes the EXTRACT and COMBINE FUs.

    - The EXTRACT FU extracts a part, 4 bits, 8 bits, or 16 bits, of the incoming data and sends it to the destination FUs. The location of the part being extracted is indicated by the second operand of this operation.
    - The COMBINE FU performs the opposite by combining data-parts, 4 bits, 8 bits, or 16 bits, and sending the result to the destination FUs.

    These EXTRACT and COMBINE move-commands together with the shift and rotate commands are used to perform complex permutations.

6. The MUX, REPLICATE and PUSHIMM commands are grouped into the Forwarding group since they do not change the input value but they connect function units together.

    - PUSHIMM allows the programmer to pass an up to 32-bit constant (immediate) value with the move-command to the destination-FU. This command is important since the immediate part in any other move-command is limited to five bits. Therefore, in the case that the programmer needs to pass a constant value to a certain function unit that is greater than five bits then the PUSHIMM command should be issued.
    - REPLICATE FU moves the incoming data to the intended destination. A single move-command can indicate up to two movements from the same source FU as shown in Figure 2. The REPLICATE unit is useful in the case that a value is required to be routed to more than two destinations. Instead of repeating the operation, data is moved to the REPLICATE FU and there it can be routed to two different FUs including the REPLICATE itself. This self-reference and the possibility to route to two destinations make it possible to route data to any number of destinations.

- The MUX FU is the only function unit with which conditional branching can be performed. A destination FU is reached by either one of the two input operands of the MUX FU based on the value of the third operand to the MUX FU. If the third operand is all zeros then the first operand is routed to the destination FU; otherwise the second operand is the one routed to the destination FU. The MUX is not needed to implement any of the AES candidates but it is necessary for the implementation of multiple-precision algorithms for the asymmetric cryptographic algorithms [20].

  If-conditions are program sections that pose security vulnerabilities with respect to side-channel attacks on control-oriented processors [1]. The vulnerability is in the fact that the execution time of each of the if-condition branches is not equal, therefore the execution time can reveal the value of the condition. In the GALS-based LPSP, the MUX FU performs the if-conditions in a constant execution time regardless of the value of the condition to countermeasure this security timing attack.

7. The last group of operations includes the reading and writing operations. Every cryptographic algorithm has some parameters that should be saved in a certain store unit. The Block Memory in Region 3 serves this purpose. This unit can be utilized to store the input data block that is required to be encrypted/decrypted and also sub-keys and any other parameters. The cryptographic algorithm uses this Block Memory to save the encrypted/decrypted block at the end of execution. The reading from the Block Memory is performed by the READREG command and the writing is performed by the WRITEREG command.

   An additional store unit is added to the processor to store S-Boxes, which are the substitution boxes. These values are required by certain cryptographic algorithms and are sometimes addressed in nibbles, like in the Serpent cryptographic algorithm. The reading from this store unit is performed using the READSBOX command. The memory for the S-boxes is separated from the Block Memory for two reasons; first because it can be addressed in nibbles and the Block Memory is never addressed in nibbles and second because it is optional and not all algorithm utilize it. This is one of the design parameters taken into consideration for future implementation of dynamic reconfiguration at runtime. The dynamic reconfiguration can reconfigure the FPGA area to include only those processor blocks required for the implementation of a certain algorithm.

2.2 GALS Wrappers

The six regions inside the GALS-based LPSP communicate asynchronously using GALS-wrappers utilizing a four-phase handshaking protocol. The design of these wrappers on an FPGA is not a straight-forward solution since timing constraints on particular paths should be taken into consideration for the correctness and stability of the circuit operation. The details of the design of the wrappers are discussed in [8].

3 Secure Execution of the AES on the GALS-based LPSP

The AES [6] has been chosen as a case study in this paper since it is the current encryption standard and since the Differential Power Analysis (DPA) attack forms a serious threat on its execution. In the DPA attack, a theoretical model of the circuit of the processor executing the AES algorithm or any other cryptographic algorithm is created. The power consumption of the theoretical model is then estimated for all possible values of a part of the private key. This part of the private key can be the eight most significant bits (MSB) as will be shown in the following subsections. On the other hand, the power dissipated from the circuit or the processor under attack is measured in realtime while it is performing the encryption algorithm. This measured power dissipation profile is affected by the whole private key and not just part of it. Finally, a correlation between the measured power dissipation and the estimated power dissipation is supposed to result in a single global peak at the value of the part of the key that matches the same part of the key that was used in the realtime encryption.

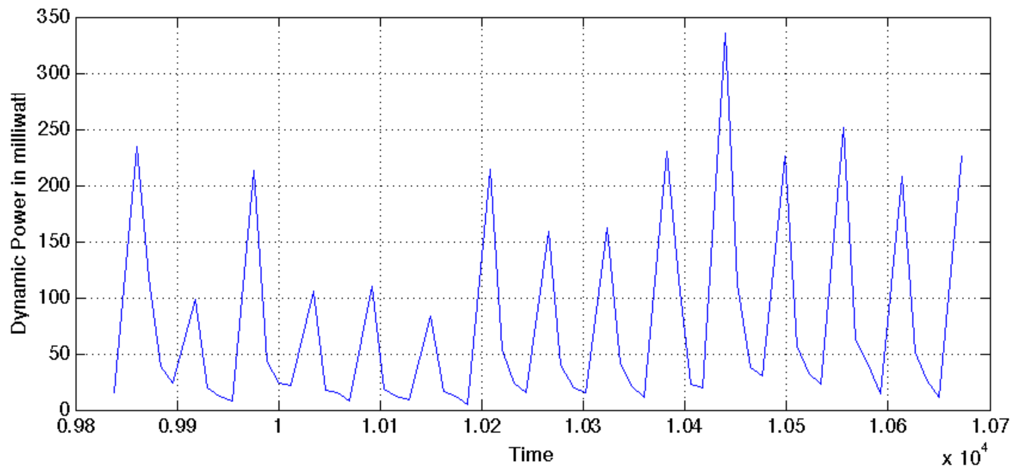

Figure 3: Part of the power dissipated during the execution of the AES on the (synchronous) version of the GALS-based LPSP.

In order to mitigate the DPA attack, it is required to reduce or eliminate the correlation between the measured power dissipation and the estimated power dissipation. Therefore, we have equipped the processor with two main features that increase its resistance to the DPA attacks. The first feature is the employment of the GALS systems and the second is the unpredictable operation reordering at the software and hardware layer.

3.1 GALS as a Countermeasure

In GALS systems, the decoupled regions operate at different clocks and therefore remove the global power dissipation peaks of ordinary synchronous systems. The GALS-based LPSP is implemented using a global clock source to imitate a synchronous architecture. This implementation has been created in order to show the regular clock peaks in the synchronous version which do not appear in the GALS version. Part of the power consumption of the synchronous version of the GALS-based LPSP is shown in Figure 3. The power consumption for the same period of time but from the GALS-based LPSP in GALS standard mode is measured and shown in Figure 4. It is clear how the (synchronous) version shows periodical power dissipation peaks and that these peaks do not appear in the (standard) version.

These clock peaks are usually used by DPA attackers to synchronize their theoretical model to the real device under attack and thereby enhancing the correlation outcome. Therefore removing these clock peaks complicate the DPA attack [18].

3.2 Randomization as a Countermeasure

In regard to the second feature that increases the processor's resistance to DPA attack, we have mentioned above that the tags and the buffers at each function unit give freedom to the order of arrival of operands and move-commands. Therefore, a different order of arrival does not affect the program semantic but affects the order of execution only. This feature is exploited to obscure the power dissipated from the GALS-based LPSP. The reordering of the move-commands can be done at the software layer by shuffling parts of the program and can be done at the hardware layer by sending the results and the move-commands in different order.

On the hardware layer, results are sent to each function unit through interconnection networks. There are two types of interconnection networks, one connecting the regions and is therefore called the Global Interconnection Network (GIN), and another one connecting the function units inside a single region and is therefore called the Local Interconnection Network (LIN). The GIN is shown in Figure 1 in Region 6 and the LIN is shown in Region 2, 3, 4, and 5 as the lines connecting to

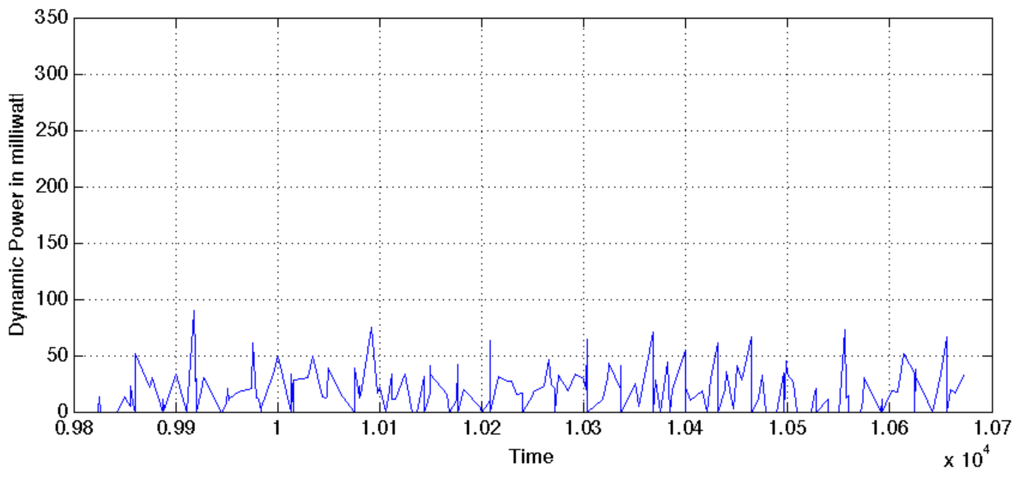

Figure 4: Part of the power dissipated during the execution of the AES on the (standard) GALS-based LPSP.

Region 6. These networks are not fully connected networks and therefore there is an arbiter inside each interconnection network that arbitrates between contending requests. Each arbiter contains a register, called Priority Register, with number of bits equal to the maximum possible requests.

The Priority Register holds a single logic-1 at a random location. The random location is decided upon at the beginning of the algorithm execution, using the random number generator as in [15]. The location of the logic-1 denotes the request with the highest priority. The contents of this priority register is rotated after each decision in order to allow for equal chances for the contending requests. As a result, with every new execution, the priority registers are changed randomly and therefore the arbitration results changes and consequently the order of execution changes.

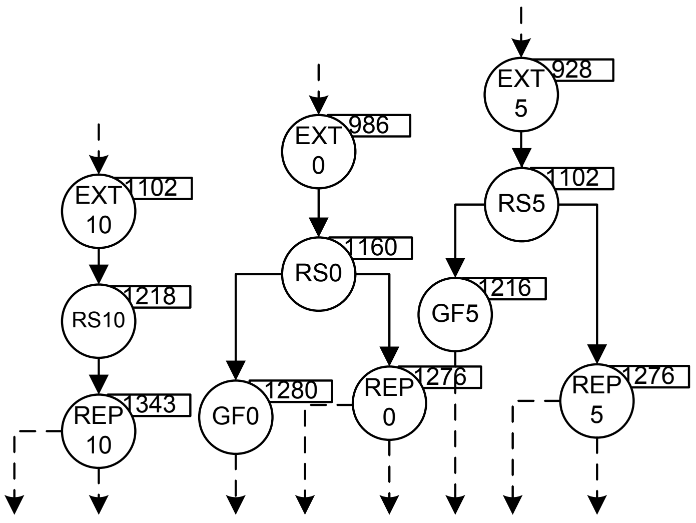

Figure 5: An excerpt of the AES dataflow graph with the request by EXT10 given a higher priority than the request by RS5.

In Figure 5 an excerpt of the AES algorithm dataflow graph is shown. Each bubble carries the name of the FU and a number signifying its tag. Connected to each bubble there is a box with the start time of the operation in the bubble in nanoseconds. The timing data in this dataflow graph is extracted from the Post-Place and Route simulation on the Xilinx hardware designing tool. The

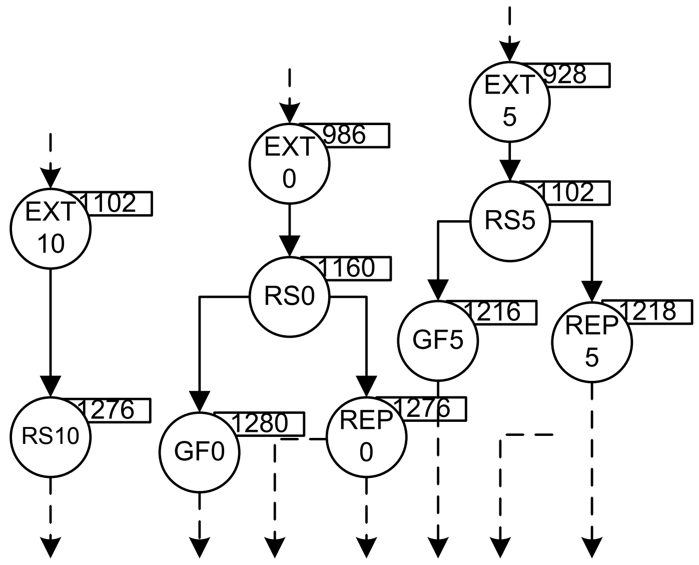

Figure 6: An excerpt of the AES dataflow graph with the request by RS5 given a higher priority than the request by EXT10.

EXT stands for the EXTRACT unit, the RS stands for the READSBOX unit, the REP stands for the REPLICATE unit and the GF stands for the GFMUL unit. The number following the FU's name stands for the tag value. The EXT10 and the RS5 start execution at the same time and since they are located in the same region, they are expected to finish execution and request their results to be routed at the same time. Their results are routed to function units inside the same region and their requests will contend at the local interconnection network of this region. In Figure 5 the request has been granted to EXT10, however in Figure 6 the request has been granted to RS5. The different granting of requests results in different execution sequence and therefore results in different power dissipation although the input data and the private key have not changed.

The AES algorithm constitutes of four major macro-operations, the AddRoundKey, the SubBytes, the ShiftRows and the MixColumns. The first macro-operation poses the highest vulnerability to the execution of the AES against DPA attacks. In this operation the private key is Ex-Ored with the input data that is required to be encrypted. Therefore, any power dissipated during this operation can carry traces from the private key.

We have conducted the DPA attack on the GALS-based LPSP two times in two different modes. The first time, the Priority Register is loaded each time an encryption is performed with the same value and the program is never shuffled at the software layer. We call this mode, the GALS-based LPSP with No Randomization. The second time, the Priority Register is loaded each time an encryption is performed with a value that includes a single logic-1 but that is located randomly each time in the Priority Register. This mode is called the GALS-based LPSP with Randomization.

We have performed the encryption for $10,000$ different plaintexts with the same private key and acquired the power dissipation readings from the Xilinx XPower tool. We have averaged the acquired power dissipation over the period of time when the AddRoundKey is executing and for each of the plaintexts to acquire a Global Prediction Vector (GPV) with $10,000$ values.

The hardware model for the GALS-based LPSP has been used to calculate the number of transitions in the eight MSBs of all registers inside the processor during the period of the AddRoundKey process. In these calculations, we have controlled the eight MSBs of the private key by changing them from 0 to 255. Hence, we have performed the $10,000$ encryptions with the same $10,000$ plaintexts previously mentioned but 256 times for all possible eight MSB of the private key. These calculations resulted in Selected Prediction Matrix (SPM) with $10,000$ rows and 256 columns.

The last step in the DPA attack is to correlate each of the columns in the SPM with the GPV. This correlation results in 256 correlation coefficients. We have conducted this DPA attack twice,

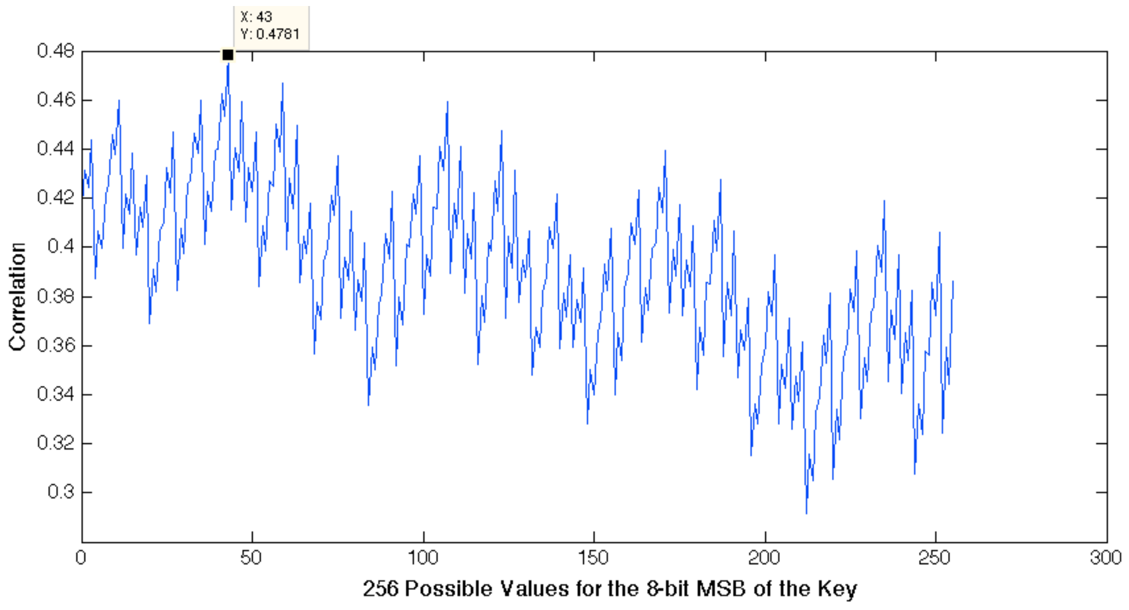

Figure 7: The 256 correlation coefficients for the 256 possible eight MSBs of the private key for the 128-bit AES encryption on the GALS-based LPSP in No Randomization mode.

as has been mentioned above. Figure 7 shows the Pearson correlation coefficients for the different 256 values of the eight MSBs of the private key while the processor is running in No Randomization mode. There is a single peak at the value 43, which is pointed out in the graph in Figure 7. The value 43 is the correct value of the eight MSBs of the private key that has been used in the encryption that resulted in the first GPV. Therefore, the attack was successful and eight bits of the private key have been recovered by the DPA attack. However, the GALS-based LPSP has been running in No Randomization mode.

Figure 8, on the other hand, shows the Pearson correlation coefficients for the different 256 values of the eight MSBs of the private key while the processor is running in Randomization mode. There is no single peak and the correct part of the key, which is 43, has a low correlation coefficient as pointed out in Figure 8.

The power dissipation changes during each encryption even if the plaintext and key have not changed and this is due to the randomization introduced by every Priority Register in every Interconnection Network inside the processor. Figure 9 shows four different power dissipation profiles for the same plaintext and key. It is for this reason that there is no correlation between the GPV and SPM.

4 GALS-based LPSP Performance Analysis

The GALS-based LPSP shows great resistance to DPA attacks. Despite the fact that this is an essential feature it cannot compensate for low performance. Therefore, in the following paragraphs we show how we take high performance as the other important design dimension in our architectural design. The three performance metrics considered in this paper are the throughput, the energy consumption and the performance density.

The function units in the GALS-based LPSP are grouped into the execution regions according to their execution latency as shown in Figure 10. Hence, the clock frequency of each regions is almost close to the performance of every function units inside the region. This decision has been taken to reduce the amount of idle time per each instruction and increase the concurrency between the function units. Moreover, if a certain algorithm does not utilize a certain region then this region is

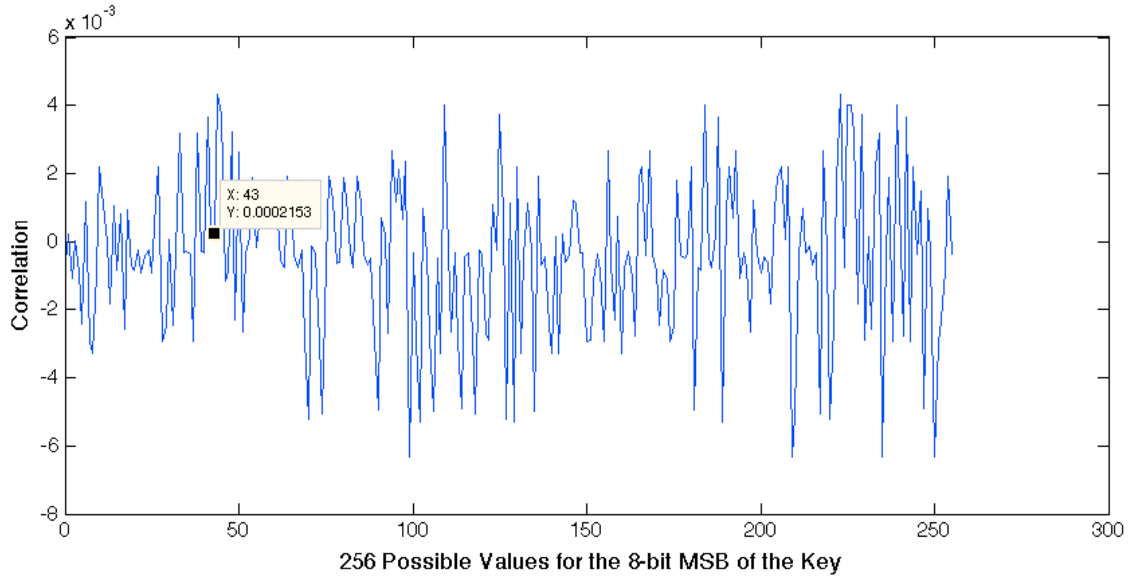

Figure 8: The 256 correlation coefficients for the 256 possible eight MSBs of the private key for the 128-bit AES encryption on the GALS-based LPSP in Randomization mode.

shut down to reduce the total consumed power. The dataflow architectural model of the GALS-based LPSP allows shutting down regions without affecting the operation of the processor.

There is couple of FUs that are grouped into regions that do not match their latency, such as the EXTRACT FU. This decision has been taken because the EXTRACT FU communicates often with the COMBINE, REG, and REPLICATE FUs. As a result, if these FUs were not combined in the same regions then the Interconnection Network in Region 6 would have been greatly overloaded. Consequently, the intercommunication between function units has been also partly considered in the decision of grouping these FUs into regions.

In order to view the effect of the function unit grouping over the performance of the cryptographic algorithms, we illustrate the interaction between the function units during the execution of five cryptographic algorithms in Figure 11, 12, 13, 14, and 15.

Function units can communicate in an asynchronous fashion if they are located in two different regions. Such interactions are denoted by dashed lines in Figure 11 through Figure 15.The dashed lines implicitly mean the interaction of the Region 6 to move results from one region to another.

In the case that a function unit acts as a source of results to another function unit but never receives results from the other function unit, then a blue arrow is drawn from the first to the second function unit. However, if the first function unit receives also results from the second function unit, then a black bidirectional arrow is drawn in between them.

Hence, figures from 11 to 15 show the opened channels (unidirectional and bidirectional) between function units and the active function units for each algorithm execution. Executions that require less number of FUs consume less power and execution that do not require all regions also save power since the clock source to the idle region is disabled. Consequently, these figures give a rough idea about the expected relative power consumption. However, they do not show which function units operate in parallel and how many function units operate in parallel during the execution of the cryptographic algorithm.

Figures 16 to 20 show the amount of FUs that operate in parallel every nanosecond. The more function units operating in parallel, the more power is consumed at this unit time but the faster the algorithm is executed. In Figure 16, for instance, there are five function units operating in parallel several times during the AES execution unlike the RC6 execution (shown in Figure 17). However, the RC6 execution has four function units operating in parallel for more percentage of the time than in the AES execution. These differences can be inspected visually from the Figures 16-20, but in

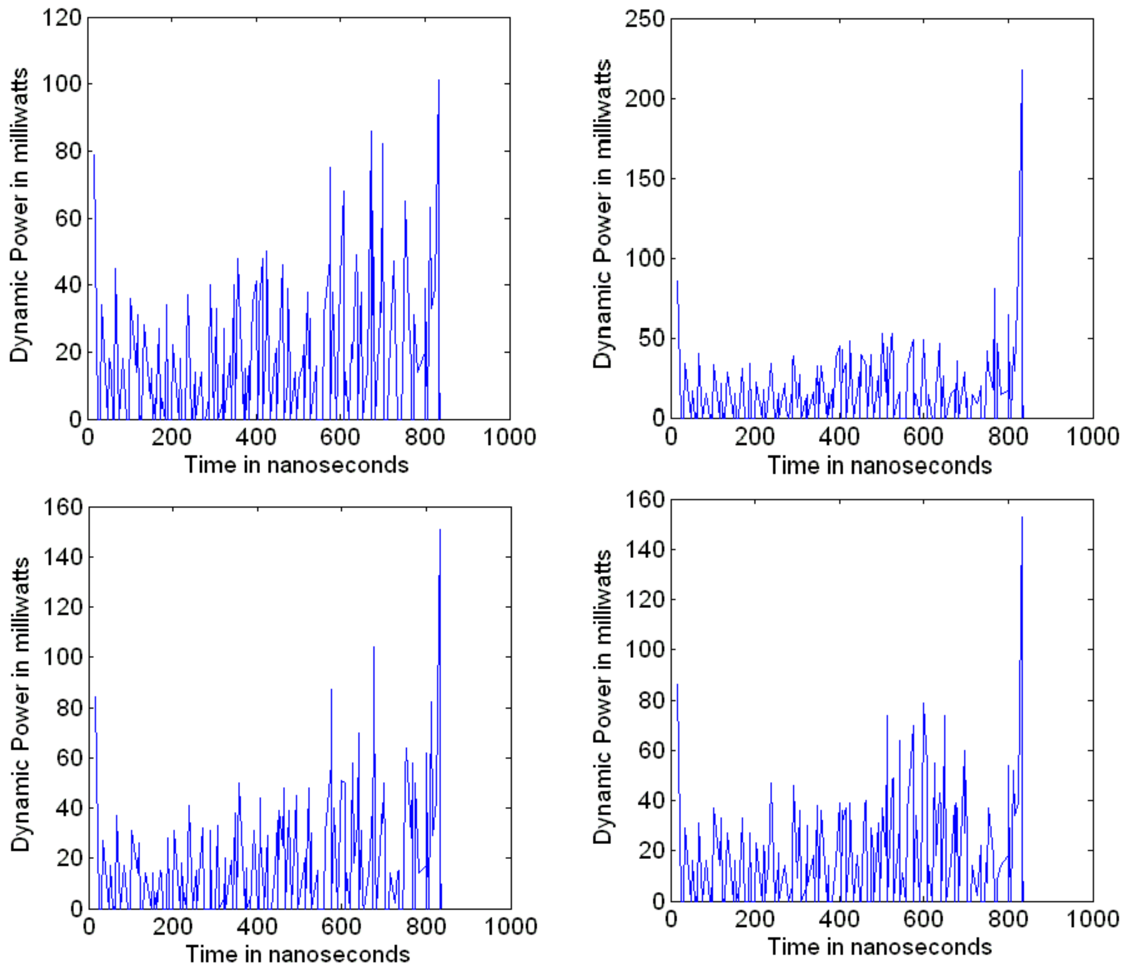

Figure 9: Power dissipated during the first 800 nanoseconds for four runs with the same algorithm, same key, and same input data-block.

order to compare the parallel executions numerically, the average parallelism is calculated.

$$\text{Avg.Parallelism} = \frac{\sum_{t=0}^{EndofExecution} \text{ParallelFU(t)}}{\text{Total Execution Time}} \tag{1}$$

The average parallelism has been calculated using Equation 1 to summarize the information given in the graphs showing the amount of parallelism exploited by each algorithm.

The average parallelism, execution time and the consumed dynamic power are given for each algorithm in Table 2. According to the average parallelism factor, the TwoFish algorithm is the algorithm that most exploits the current layout of the architecture. The RC6 is the algorithm that consumes more power than any of the other illustrated algorithms, although it requires less FUs to be executing in parallel than the Twofish. However, comparing the amount of used asynchronous communication in Figure 12 and Figure 13, it can be seen that the RC6 algorithm requires more asynchronous communications than the Twofish. The asynchronous communication is denoted by the dashed lines and the bidirectional arrows are counted twice.

The amount of asynchronous communication between the function units affects the dissipated power since it activates the Global Interconnection Network in Region 6 accordingly. The Elliptic Curve Cryptography (ECC) is the other algorithm that is affected greatly by the amount of asynchronous communications. The ECC has an average parallelism less than the three symmetric encryption algorithms and even though it requires more power than two of them. This is due to

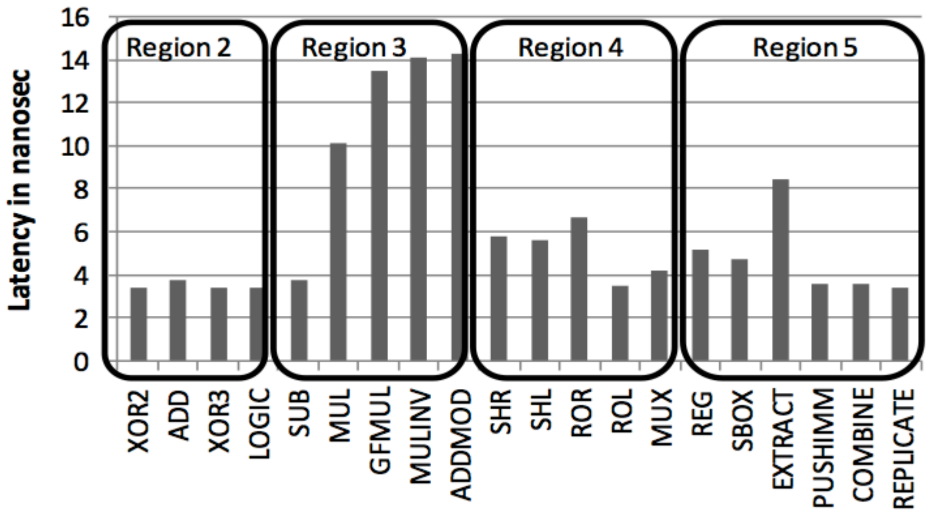

Figure 10: Distribution of FUs over the execution regions.

the fact that the ECC has the largest amount of asynchronous communication among all these algorithms. Again, the amount of asynchronous communication can be reduced by grouping the most communicating function units into a single region although this will affect the execution time since they would have to follow the same clock frequency.

Dissipated power can be reduced even more by allowing dynamic reconfiguration according to the algorithm resource requirements. However, this feature is not employed in this implementation. Nonetheless, it is worth noting that the architecture is flexible enough to allow the implementation of such features.

5 Implementation Results

The Xilinx Integrated Software Environment is used to design the GALS-based LPSP starting from the design entry using the hardware description language (VHDL) through to the device programming using iMPACT tool. The entire implementation process has been performed on a 2.66 GHz Intel CoreTM2 Quad Q6700 processor with 8 MB L2 Cache and 4 GB RAM.

The GALS-based LPSP occupies almost 70% of the Xilinx Virtex5 chip [24] (XC5VLX155T) area as shown in Figure 21. Region 1 and Region 6 occupy the least areas while Region 3 and Region 4 occupy the greatest part of the area. Region 3 and Region 4 include the permutation and Galois field operations which are complex and require a great amount of FPGA slices. The following report states the area utilized by the processor and the equivalent gate count.

6 Comparisons

The predominant soft cores that have been adopted as a co-processor for security applications or their internal architecture has been modified to accommodate new instructions for cryptographic algorithms are the LEON2[22], LEON3[11,12], Xtensa LX2[21], Xtensa T1040[19], and CoreMP7[7].

In Table 4 the throughput of the AES encryption on 128-bit block with 128-bit key is given in Mbps and normalized to 50MHz clock frequency. Results on soft cores that are flexible enough to be programmed for any other cryptographic algorithm are taken. Results for optimized soft cores with a special circuit that performs the whole AES algorithm or a whole round in the AES algorithm are not taken since they do not reflect the performance of the soft core in the execution of any cryptographic algorithm other than the AES.

The GALS-based LPSP performs at 10.13 Mbps and the next best throughput is 5.21 Mbps by

Table 2: Performance of different algorithms.

	AES 128-bit Encryption	RC6 128-bit Encryption	Twofish 128-bit Encryption	RSA 1024-bit Signature Generation	ECC 163-bit Encryption
Execution Time [μsec]	12.64	3.01	14.66	774,000	413,560
Avg. Dynamic Power [mW]	17.2	36.01	19.33	9.37	29.25
Avg. Parallelism	1.68	1.75	1.91	1.06	1.47

Table 3: Design Summary.

Slice Logic Utilization:				
Number of Slice Registers:	30,544	out of	97,280	31%
Number used as Flip Flops:	30,532			
Number used as Latches:	12			
Number of Slice LUTs:	41,986	out of	97,280	43%
Slice Logic Distribution:				
Number of occupied Slices:	17,273	out of	24,320	71%
Number of LUT Flip Flop pairs used:	57,068			
Number of fully used LUT-FF pairs:	15,462	out of	57,068	27%
Total equivalent gate count for design:	1,530,820			

Table 4: Throughput Comparison for the AES Encryption.

	Cycles	**Throughput** [MB/s]
GALS-based LPSP	632	10.13
Baseline LEON2[22]	1673	3.83
Baseline LEON3[2]	40358	0.14
LEON3 with SPx[2]	30430	0.21
LEON3 with HWAccel[14]	1228	5.21
Baseline Xtensa LX2[21]	283000	0.02
Baseline Xtensa T1040[19]	24419	0.26
Xtensa with SPx[21]	2800	2.29
Xtensa ASIP[19]	1400	4.57
Baseline CoreMP7[7]	8400	0.76

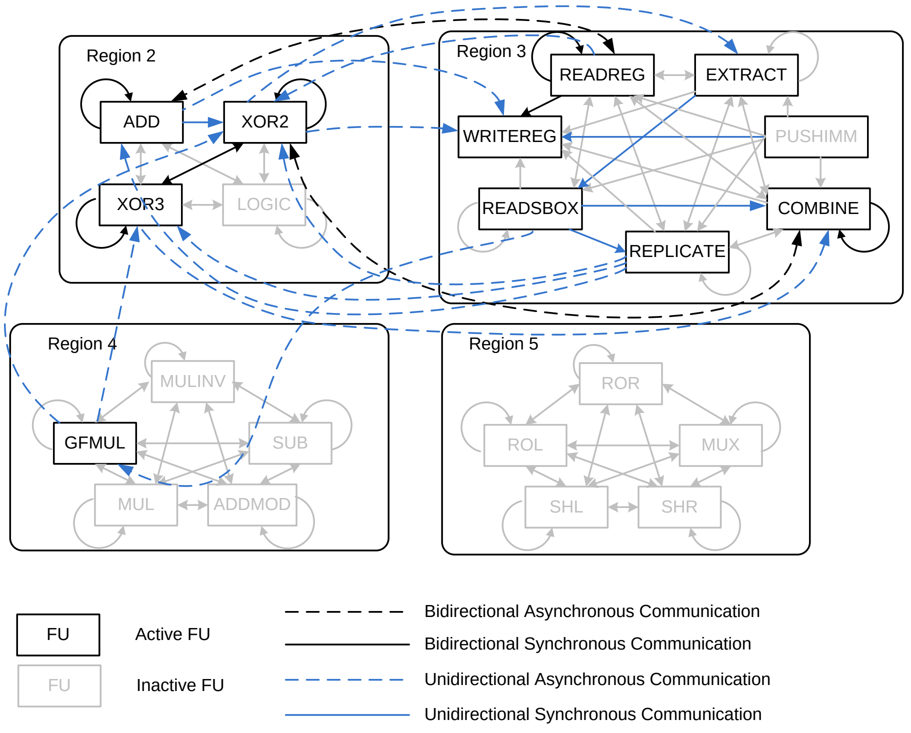

Figure 11: Interaction between function units during the AES execution.

the LEON3 soft-core with a hardware accelerator for the AES special functions. The effect of the resonance between the AES cryptographic algorithm and the dataflow nature of the GALS-based LPSP is evident by the high throughput.

Table 5: Energy Consumption Comparison for the AES Encryption.

	Energy $[\mu Joule]$
GALS-based LPSP	5.3
Baseline LEON3[2]	504.9
LEON3 with SPx[2]	184.5
LEON3 with HWAccel[14]	15.8
Baseline Xtensa LX2[21]	1681
Xtensa with SPx[21]	18.9
Baseline CoreMP7[7]	8.4

The GALS-based LPSP does not have a single clock generator and therefore a weighted average of the execution frequency in each region is taken. The execution time of the AES encryption is then multiplied by the power dissipated to calculate the energy in micro-Joule. The energy consumption comparisons are given in Table 5. The GALS-based LPSP requires 5.3 μJoule and the next lowest energy consumption is by the CoreMP7 which is 60% more.

Finally, the area occupied by each of the soft core processor is given in Table 6. The GALS-based LPSP occupies 43,627 slices of the Virtex-5 chip which is equivalent to 1530 kilo-Gates as reported by the Xilinx Mapper and shown in Section 5.

$$\text{Performance Density} = \frac{\text{Throughput(Mbps)}}{\text{Area(kGates)}} \tag{2}$$

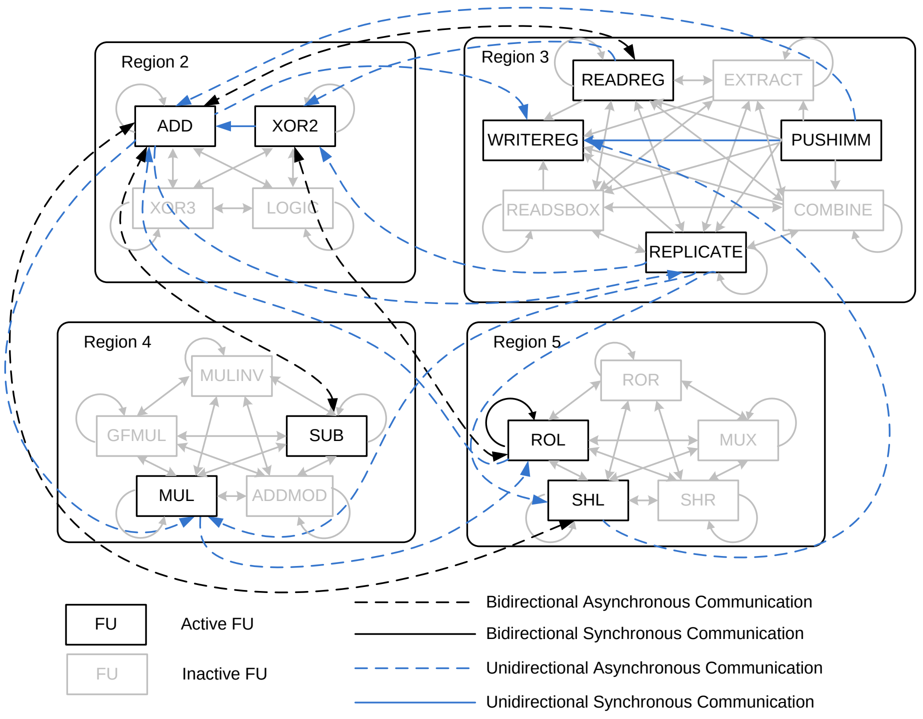

Figure 12: Interaction between function units during the RC6 execution.

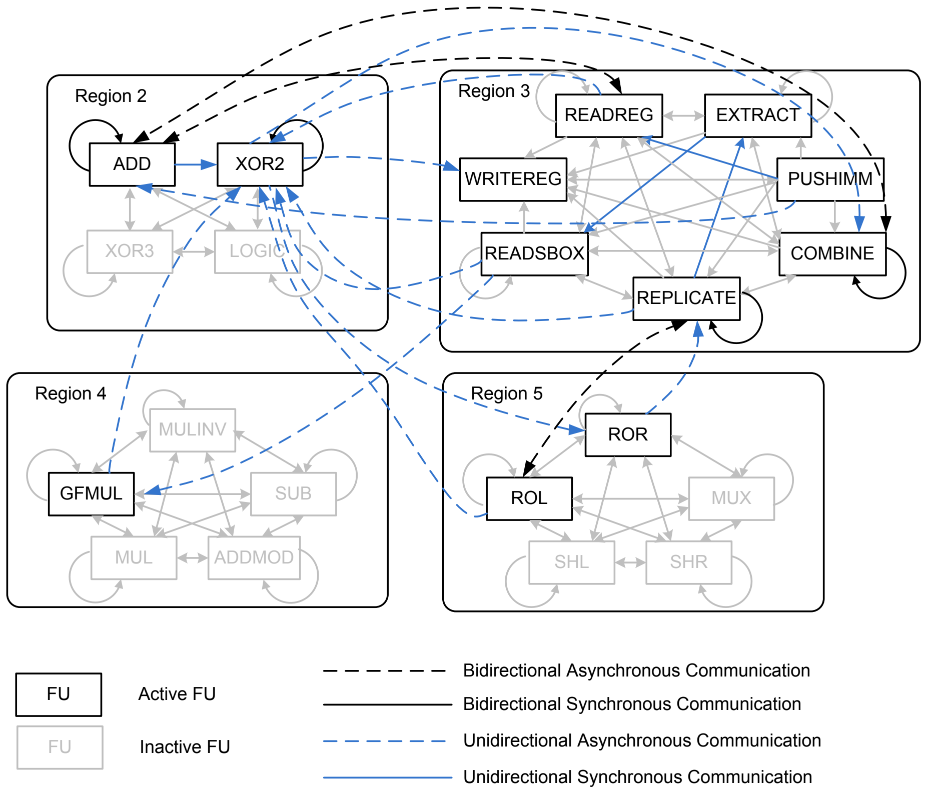

Figure 13: Interaction between function units during the TwoFish execution.

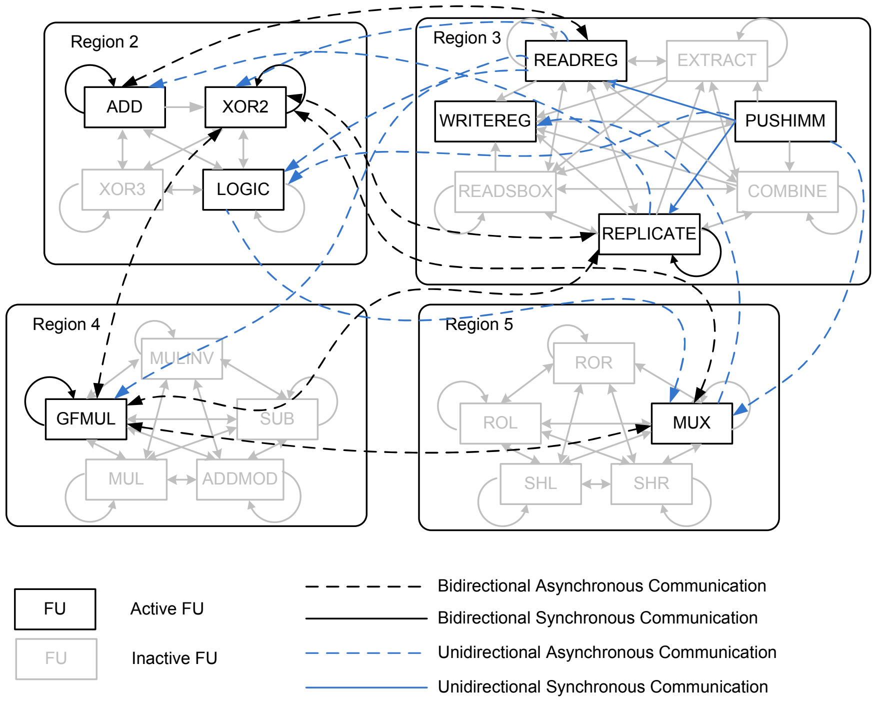

Figure 14: Interaction between function units during the RSA execution.

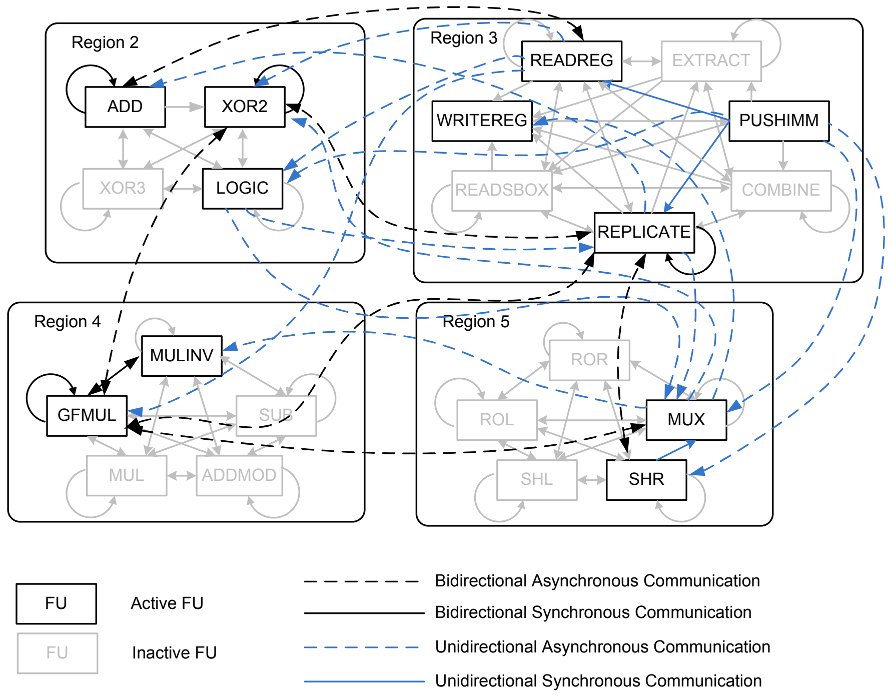

Figure 15: Interaction between function units during the ECC execution.

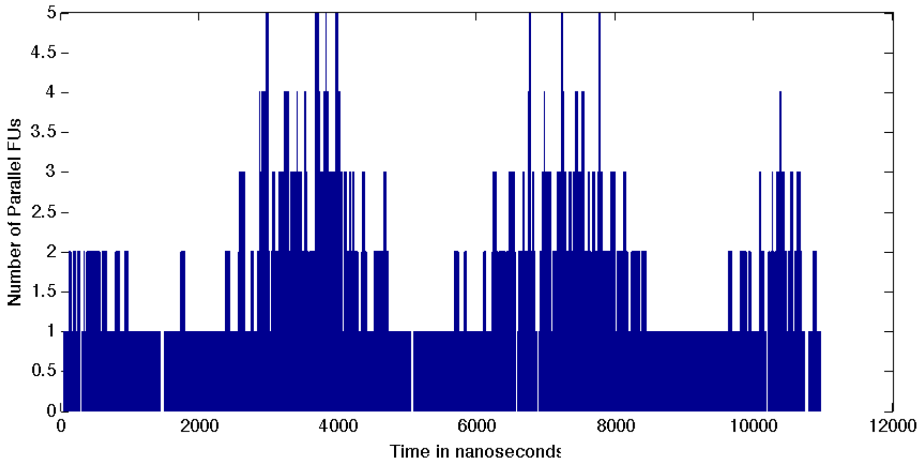

Figure 16: The amount of function units operating in parallel during the AES execution.

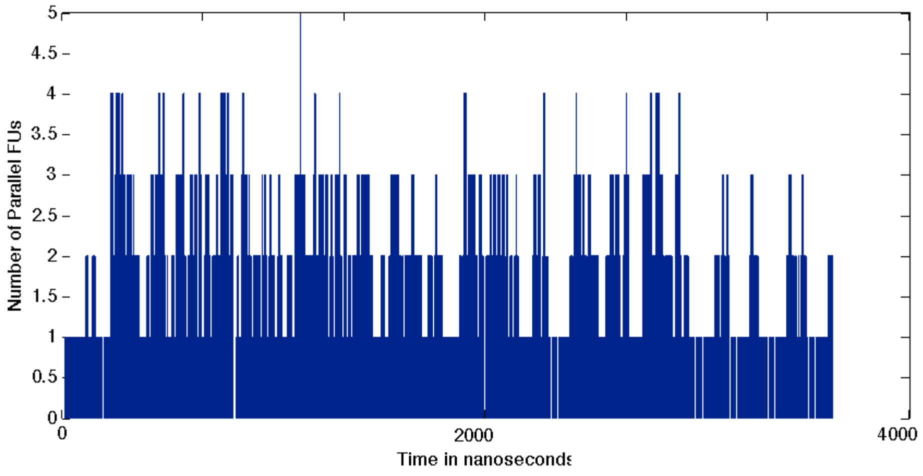

Figure 17: The amount of function units operating in parallel during the RC6 execution.

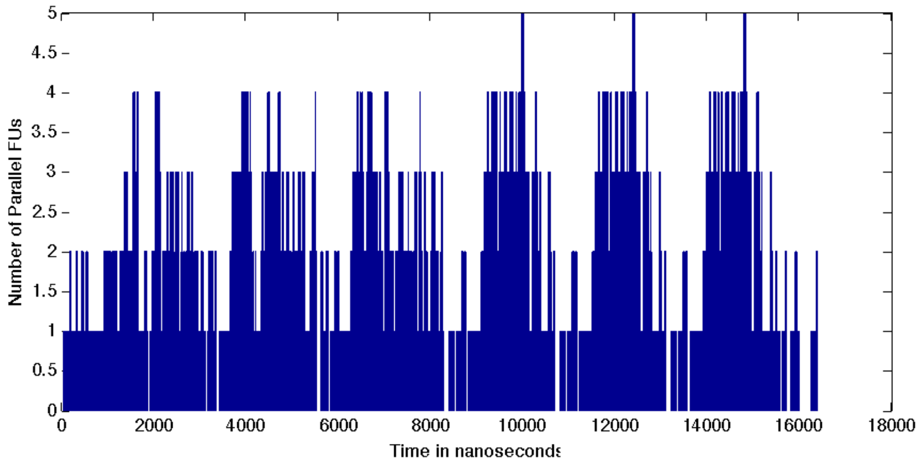

Figure 18: The amount of function units operating in parallel during the TwoFish execution.

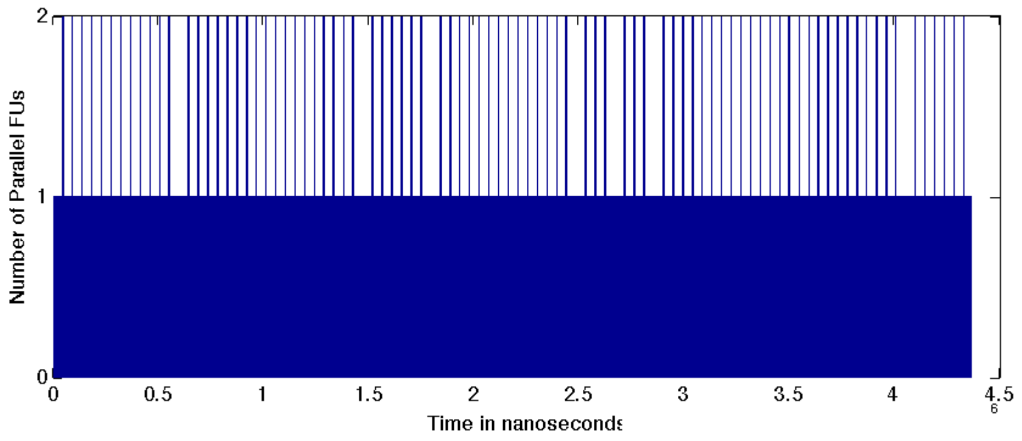

Figure 19: The amount of function units operating in parallel during the RSA execution.

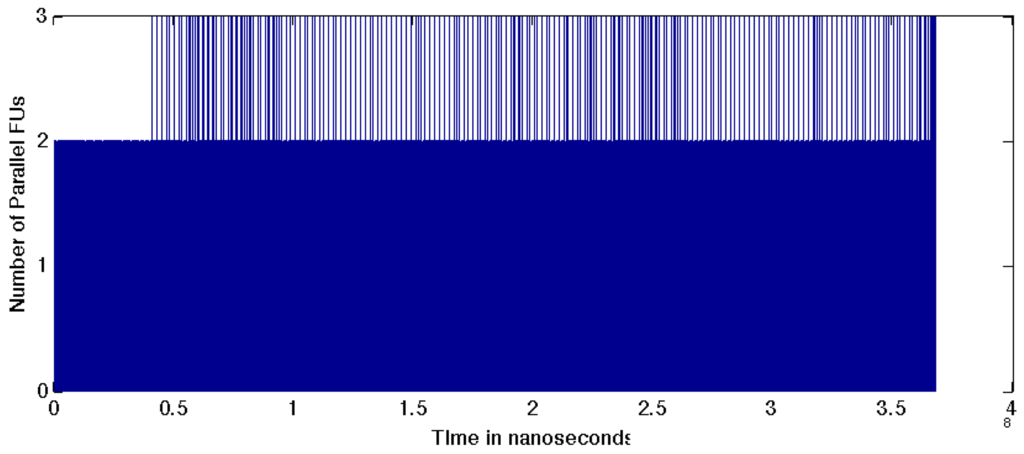

Figure 20: The amount of function units operating in parallel during the ECC execution.

Table 6: Area and Performance Density Comparison for the AES Encryption.

	Area [kGates]	**Perf. Density** [Bps/Gate]
GALS-based LPSP	1530	6.6
Baseline LEON3[2]	437	0.32
LEON3 with SPx[2]	531	1.14
Baseline CoreMP7[7]	423	1.80

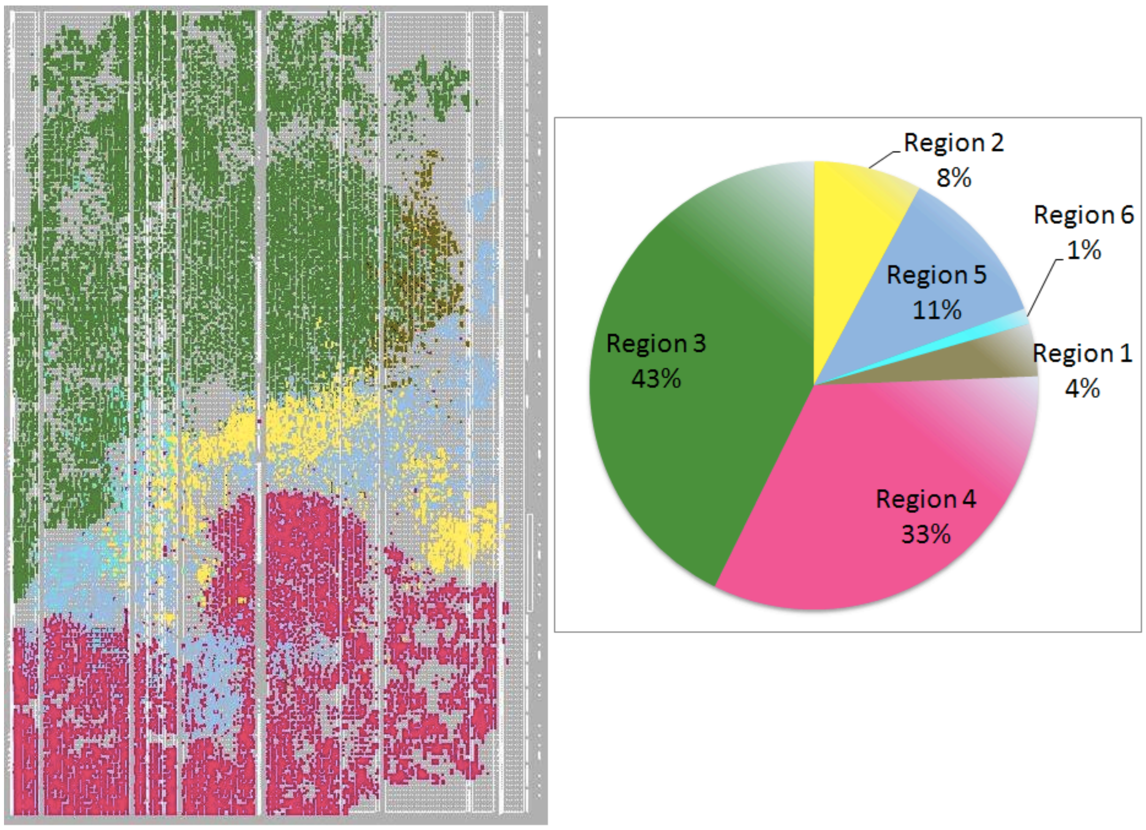

Figure 21: The Floorplan view of the placement on Xilinx Virtex5 (XC5VLX155T).

The GALS-based LPSP occupies a large area due to the large number of buffers attached to each function unit. However, this large area is effectively used and this is revealed by the performance density. The performance density denotes the amount of throughput achieved by a certain area and is given by Equation 2.

7 Conclusions and Future Work

In the presented work, it has been already demonstrated the possibility of implementing a new architecture, the GALS-based LPSP, that enjoys the following outstanding advantages and features:

1. Lower energy consumption when compared with other soft cores that implement the same cryptographic algorithms and this is shown in Table 5. The lower energy is achieved by the implementation of the Globally-Asynchronous Locally-Synchronous interactions that allow the decoupling of the execution regions and thereby disabling the clock sources to the idle regions.
2. Higher throughput when compared with other soft cores implementing cryptographic algorithms. This is shown in Table 4. The high throughput is achieved by the architecture of the GALS-based LPSP that allows any amount of function units to operate in parallel as required by the algorithm. Moreover, the multiple clock sources let the function units operate at their own pace and better utilize their clock period.
3. Immunity to Differential Power Analysis (DPA) attacks due to the implementation of the GALS systems in the internal structure of the processor and the possibility of unpredictable reordering of instructions without affecting the program semantic. The reordering does not only happen on the level of instructions in the program but also on the level of the bypassed results between the function units.

Moreover, the flexibility of the developed architecture allows the implementation of dynamic reconfiguration. The dynamic reconfiguration can be used to rearrange or allocate and deallocate function units either to reduce the energy consumption or to achieve higher throughputs while keeping its immunity to DPA attacks. Realization and performance evaluation of the GALS-based LPSP when using dynamic reconfiguration constitute a very interesting problem for future work.

References

[1] Onur Aciicmez, Cetin Kaya Koc, and Jean-Pierre Seifert. On the power of simple branch prediction analysis. In *ASIACCS 07: Proceedings of the 2nd ACM symposium on Information, computer and communications security*, pages 312–320, New York, NY, USA, March 2007. ACM.

[2] Syed Zahid Ahmed, Julien Eydoux, Laurent Rougé, Jean-Baptiste Cuelle, Gilles Sassatelli, and Lionel Torres. Exploration of power reduction and performance enhancement in leon3 processor with esl reprogrammable efpga in processor pipeline and as a co-processor. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '09, pages 184–189, Nice, France, April 2009.

[3] H. Brunner, A. Curiger, and M. Hofstetter. On computing multiplicative inverses in GF(2^m). *IEEE Transactions on Computers*, 42:1010–1015, August 1993.

[4] H. Corporaal. Design of transport triggered architectures. In *Proceedings of the Fourth Great Lakes Symposium on Design Automation of High Performance VLSI Systems. GLSV '94.*, pages 130 –135, South Bend, Indiana, USA, March 1994.

[5] Henk Corporaal and Hans (J.M.) Mulder. MOVE: a framework for high-performance processor design. In *Proceedings of the 1991 ACM/IEEE conference on Supercomputing*, Supercomputing '91, pages 692–701, New York, NY, USA, November 1991. ACM.

[6] Joan Daemen and Vincent Rijmen. Rijndael for AES. In *Proceedings of the Third Advanced Encryption Standard Candidate Conference (AES3)*, pages 343–348, New York, NY, USA, April 2000.

[7] Milos Drutarovsky and Michal Varchola. Cryptographic system on a chip based on actel ARM7 soft-core with embedded true random number generator. *Design and Diagnostics of Electronic Circuits and Systems*, pages 1–6, 2008.

[8] Hala A. Farouk and Mahmoud T. El-Hadidi. Implementing globally asynchronous locally synchronous processor pipeline on commercial synchronous FPGAs. In *Proceedings of the IEEE 17th International Conference on Telecommunications (ICT)*, pages 989 –994, Doha, Qatar, April 2010.

[9] Hala A. Farouk, Mahmoud T. El-Hadidi, and Ahmed Abou El Farag. GALS-based LPSP: Implementation of a novel architecture for low power high performance security processors. In *Proceedings of the 25th IEEE International Parallel and Distributed Processing Symposium*, pages 537–545, Anchorage, Alaska, USA, May 2011.

[10] Hala A. Farouk, Mahmoud T. El-Hadidi, and Magdy Saeb. LP-GALS-C: a new low-power globally asynchronous locally synchronous architecture for symmetric-key cryptography. In *Proceedings of the 9th International Conference on Systems Theory and Scientific Computation*, pages 173–179, Moscow, Russia, August 2009.

[11] M.A. Fayed, M.W. El-Kharashi, and F. Gebali. A high-speed, low-area processor array architecture for multiplication and squaring over GF(2^m). In *Proceedings of the 2nd International Design and Test Workshop IDT 2007*, pages 226–231, Cairo, Egypt, December 2007.

[12] Frank Gurkaynak, Stephan Oetiker, Hubert Kaeslin, Norbert Felber, and Wolfgang Fichtner. Improving DPA security by using globally-asynchronous locally-synchronous systems. In *Proceedings of the 31st European Solid-State Circuits Conference*, pages 407 – 410, Grenoble, France, September 2005.

[13] Panu Hamalainen, Jari Heikkinen, Marko Hannikainen, and Timo D. Hamalainen. Design of transport triggered architecture processors for wireless encryption. In *Proceedings of the 8th Euromicro Conference on Digital System Design*, pages 144–152, Washington, DC, USA, August 2005. IEEE Computer Society.

[14] Alireza Hodjat. Interfacing a high speed crypto accelerator to an embedded CPU. In *Proceedings of the 38th Asilomar Conference on Signals, Systems, and Computers*, pages 488–492, Pacific Grove, CA, USA, November 2004. IEEE Press.

[15] Samy H.M. Kwok and Edmund Y. Lam. FPGA-based high-speed true random number generator for cryptographic applications. In *IEEE Region 10 Conference TENCON 2006*, pages 1 –4, Fukuoka, Japan, November 2006.

[16] Yong Li, Zhi-ying Wang, and Kui Dai. A low-power application specific instruction set processor using asynchronous function units. In *Proceedings of the 7th IEEE International Conference on Computer and Information Technology*, pages 817–822, Fukushima, Japan, October 2007. IEEE Computer Society.

[17] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography.* CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1996.

[18] Simon Moore, Ross Anderson, Paul Cunningham, Robert Mullins, and George Taylor. Improving smart card security using self-timed circuits. In *Proceedings of the Eighth International Symposium on Asynchronous Circuits and Systems*, pages 211 – 218, Manchester, UK, April 2002.

[19] Srivaths Ravi, Anand Raghunathan, Nachiketh Potlapally, and Murugan Sankaradass. System design methodologies for a wireless security processing platform. In *Proceedings of the 39th annual Design Automation Conference*, DAC '02, pages 777–782, New Orleans, LA, USA, June 2002. ACM.

[20] Erkay Savas, Alexandre F. Tenca, and Çetin Kaya Koç. A scalable and unified multiplier architecture for finite fields GF(p) and GF(2^m). In *Proceedings of the Second International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '00, pages 277–292, Worcester, MA, USA, August 2000. Springer-Verlag.

[21] Tensilica. How to minimize energy consumption while maximizing ASIC and SOC performance. White paper, `www.tensilica.com`, October 2008.

[22] Stefan Tillich and Johann Großschädl. Instruction set extensions for efficient AES implementation on 32-bit processors. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006*, volume 4249 of *Lecture Notes in Computer Science*, pages 270–284. Springer Berlin / Heidelberg, 2006.

[23] Chien-Hsing Wu, Chien-Ming Wu, Ming-Der Shieh, and Yin-Tsung Hwang. High-speed, low-complexity systolic designs of novel iterative division algorithms in gf(2^m). *Computers, IEEE Transactions on*, 53(3):375 – 380, March 2004.

[24] Xilinx. Virtex-5 overview. Data Sheet DS100(V5.0), Xilinx Inc., February 2009.