VLIW Cryptoprocessor: Architecture and Performance in FPGAs

Fábio Dacêncio Pereira,[1,2] ***, Edward David Moreno Ordonez*** [1,2]***, and Rodolfo Barros Chiaramonte*** [2]

[1] **University of São Paulo-USP/POLI, SP, Brazil,**
[2]**University Center Euripides of Marília-UNIVEM, SP, Brazil**

Summary
This work is intended for presenting the proposal and implementation of a VLIW cryptoprocessor, as well as discussing architecture details and specifying its instruction set. The cryptoprocessor was designed to execute symmetric cryptography algorithms preferentially. To do so, special modules was described in order to increase the performance and simplify the source program. The cryptoprocessor was described using VHDL language, and a prototype was synthesized and implemented in a FPGA Virtex II Pro generating occupation statistic data and temporary performance, both presented in this paper.

This cryptoprocessor supports a number of symmetric algorithms including current ones which uses 128-bit keys or more. It is important to stress that the special modules which makes the cryptoprocessor different are not specific to a certain cryptography algorithm, and they were projected in order to be preconfigured according to the characteristics of the algorithm to be executed

Key words:
VLIW Architecture, cryptoprocessor, FPGAs and performance statistics.

1. Introduction

Security to the transmission of information is becoming as much a vital issue and requires the development of new techniques and algorithms of information cryptography in order to promote a safe and fast transmission environment. One of the tools to obtain such level of reliable security is the cryptography, a process in which a legible text is converted into a senseless writing by means of a key or password, making it possible to recover original information from these senseless text and appropriate key. Basically there are to types of cryptography: symmetric and asymmetric.

The former uses the same key on the ciphering and deciphering processes, and the latter contains two keys (public and private), when generally speaking every text ciphered with the public key can only be deciphered with the private key.

In this work it is presented the proposal and implementation of a VLIW cryptoprocessor dedicated to the execution of symmetric cryptographic algorithms using a FPGA technology. It is also presented the results obtained in the implementation in FPGA discussing architectural details and specifying its instructions set
It was carried a detailed study on the algorithms RC6 [10], Serpent[13], Cast-128 [21], MARS[12], Twofish [14], Magenta [11], Frog [15], BlowFish [16] and IDEA [20] highlighting the algorithms DES [7], AES [8] and RC5[9]. The major goal of this study was identifying and quantifying the most frequently performed operations and the specific operations of each algorithm.

In the end of this research, it is shown the factors which influence the cryptoprocessor architecture, defining the proposed characteristics such as registers numbers, logical and arithmetical unit operations, buffers, special modules, among others. Besides, there is the significative occurrence of operations such as dislocation, rotation, permutation, bit substitution and the logical operation XOR (or exclusive).

From this study, important characteristic of the cryptoprocessor such as modules and special instructions were projected and implemented in FPGA. The main goal was projecting architecture to support the most number of symmetric algorithms and reach a great general performance.

It supports a number of symmetric algorithms including current ones which uses 128-bit keys or more. It is important to stress that the special modules which makes the cryptoprocessor different are not specific to a certain cryptography algorithm, and they were projected in order to be preconfigured according to the characteristics of the algorithm to be executed. Such characteristic highlights the first quality of the cryptoprocessor, that is the quantity of algorithms supported.

2. Reasons for VLIW Architecture

VLIW architecture can be defined from two concepts: (i) the horizontal microcode and (ii) the superscalar processing. An average VLIW machine contains words with hundreds of bits while the VLIW cryptoprocessor herein proposed contains 160 bits per word. The

Manuscript received August 5, 2006.
Manuscript revised August 25, 2006.

operations to be simultaneously executed are stored in one VLIW word. The horizontal microcode of each VLIW word is constituted by opcodes and data which specify the executions to be executed in different functional units.

The VLIW architecture tracks a word down a memory address, and each functional unit executes one of the different operations in the VLIW word. The main advantage of a VLIW architecture are:

- Architecture highly regular and compiler exposed with little restriction as to the access to the processor resources.
- The compiler knows in advance all of the operation effects upon the architecture.
- Multiple operation dispatch.
- It maintains the simple control hardware, theoretically allowing a smaller clock cycle.

Echeloning the instructions in compilation level makes the structure easier for the parallelism is not defined in hardware (execution time), and such simplification theoretically reduces the clock cycle time, optimizing the process as a whole. Two of the VLIW model main advantages are:

- The incorrect prevision of the route taken in conditional pathways may affect its performance considerably. Once the prevision is performed statically, important information available in execution time is completely neglected. In this work we highlight the execution of symmetric cryptographic algorithms. Thus, such advantage does not have any effect upon performance for the algorithms are static codes not changed on execution time.
- Inefficiency in programs with large data dependency. In this case, most part of the instructions cannot execute in parallel, causing harm to the system performance. It is possible to identify when there is not much data dependency by verifying the occurrence of many NOPs on the program. Such disadvantage has an effect on our system performance, and the data dependency varies from algorithm to algorithm.

The choice of the VLIW architecture model is justified by the need of reaching a distinguished performance with a hardware composed by independents modules which together can be easily controlled and coordinated to the parallel execution of instructions. Unlike other architectures with these characteristics, the VLIW architecture does not need a sophisticated hardware to control the information flow, generating a relatively simplified hardware when compared to other architectures such as superscalar and superpipeline.

The symmetric cryptographic algorithms are constituted of a sequency of static operations. It reduces the occurrence of random events in execution time which may cause harm to the cryptoprocessor performance that is not prepared to deal with this kind of execution. It does not mean that there will be an error, but will only cause harm to the execution performance.

3. Cryptoprocessor VLIW Architecture

Some characteristics of the cryptoprocessor are shown as follows. They will al be discussed in the next sections.
VLIW architecture constituted by the Harvard+Pipeline model as a RISC instruction set.

- 160-bit VLIW word
- Data and instruction cache (Harvard)
- Eight 128-bit functional units (UFCs)
- Up to 4 instructions executing in an only cycle
- 16 permutations per cycle
- 3 global stage-pipeline
- 25 instructions
- 24 registers

The 160-bit VLIW word stores up to 4 instructions of 40 bits each, all to be executed in parallel. The parallelism over 4 instructions makes the hardware more complex, increasing the machine cycle time. Besides, the occurrence over 4 instructions executed in parallel would not be common, since the data dependency would inhibit this possibility as well as jeopardizing the cryptoprocessor's general performance.

Some instruction with more frequent occurrences such as shifting, rotating, and bits permutation and substitution make important the creation of dedicated and distinguished modules in order to reach a better performance.

The bits permutation and substitution module can act in one only bit or in a 32-bit block, reducing the number of instructions it takes to execute these operations. The permutation and substitution function units were designed to reach a satisfactory performance and support the most number of algorithms. The creation of independents LOAD/STORE modules and movement among registers and pathways is important in processors which adopt the Harvard model and its different memories for instructions and data.

Figure 1 illustrates the VLIW cryptoprocessor top-level architecture, which is constituted by four parts: (i) instruction dispatcher, (ii) functional units, (iii) register bank and (iv) control unit. It is possible to note two cache memories: instruction (I-CACHE) and data (D-CACHE). The former stores the cryptography algorithms described in assembly, while the latter stores information such as the S-BOXES content of a certain algorithm, as well as the clear text and the results of the operation performed by the cryptoprocessor.

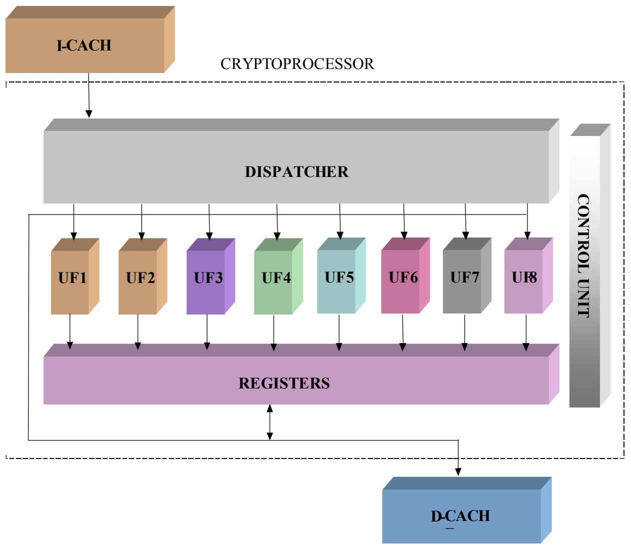

Figure 1 - VLIW Cryptoprocessor top-level architecture

I-CACHE size is 16x160, storing little more than 1 Mbyte of information and allowing the storage of several cryptographic algorithms. D-CACHE size is 16x128, storing 1 Mbyte. In the figure 1, the memories are considered an external part of the cryptoprocessor for in this prototype the memories are not implemented in FPGA. It would not be feasible to describe memories of such dimensions in these devices.

The instruction dispatcher module is responsible for the directing of each instruction in a VLIW word for the execution. In principle, this function would require a sophisticated hardware in order to make decisions, but in the VLIW architecture the dispatcher does not comprise functions of execution time decision-making. That is, the definition of which instructions will be executed and which will be the unit function is total responsibility of the compiler. This makes the hardware more simple and increase the execution speed.

Figure 1 shows the eight functional units capable of executing a set of operations selected by the instruction opcodes:

- UF1 e UF2 – ALU: Arithmetical and Logical Unit
- UF3 – 1,2,3, 8 and 32-bit logical shifter
- UF4 - 1,2,4, 8 and 32-bit rotator
- UF5 - Permutation (P-BOX)
- UF6 - Substitution (S-BOX)
- UF7 – Load/Store (responsible for the memory search and writing
- UF8 – MOV/Branches

The functional units are responsible for the execution of instructions conducted by the dispatcher module. Each unit is able to decoding the instruction to be executed. The execution among the functional units is parallel. In a certain execution cycle, up to four functional units can be in execution, once the maximum number of instructions in a VLIW word is of four instructions. Table 1 presents the functional units and the main registers they use.

Table 1 – Main registers used by the functional units

Functional Unit	Main Registers
ALU 1	A1 e B1
ALU 2	A2 e B2
Shifter	A3
Rotator	A4
Permutation	A5, B5, X
Substitution	A6, B6, SPC
Load/Store	DPC, IPC
MOV/Branches	JPC

The 128-bit A, B, and X registers store operands and functional units results, while 128-bit registers and operators optimize the symmetric algorithms descriptions which operate with 128-bit text blocks and keys or more. Thus, it is possible to perform 128-bit operations in only one cycle and store them on the appropriate register, optimizing the execution of the implemented algorithm.

The 16-bit counter registers (SPC, DPC, JPC and IPC) are responsible for the addressing of the D-CACHE and I-CACHE memories. For all operations, the result is stored on the A register. Table 2 shows all the registers organized according to the function.

Table 2 – Registers organized according to the function

Function	Registers
General Registers	X, A1, B1, A2, B2, A3, A4, A5, B5, A6, B6
Counters Registers	PERAC, AC1, AC2, SPC, DPC, IPC, JPC
Configuration Registers	SBOXEND, SBOXCOL, SBOXQ, TBO, TBD, B

Table 2 shows the register bank totalizing 24 registers, each functional unit making use of one or more different registers. The whole architecture is controlled by the control unit, which organizes the execution flow, including the three-stage pipeline.

4. ISA of VLIW Cryptoprocessor

Our cryptoprocessor instruction set follows the RISC model in which, with only 25 instructions, it is possible to describe most part of the symmetric cryptography algorithms, as shown in table 3. The instructions can be divided into four classes:

- Logical and Arithmetical (128 bits): AND, OR, XOR, ADD, SUB, SHL, SHR, ROL, ROR, INC, DEC, NOT, CLR, NOP.
- Movement: LOAD, STORE, MOV.
- Pathways: JMP, JZ, JL, JG.
- Special: PERINIC, PERBIT, SBOXINIC, SBOX.

The special instructions allow the access to the cryptoprocessor special module. Each instruction group is executed by a specific functional unit. It is important to stress that, from the eight functional units, two are designed to execute special instructions such as permutation and substitution. There are also functional units suitable for instructions which deserve some attention from the cryptoprocessor project such as the bits rotation frequent in symmetric cryptography algorithms. The relation between functional unit and instructions is described in table 3.

Table 3 – Relation between functional unit and instructions

Functional Unit	Instructions
ULA 1 e 2	AND,OR, XOR, ADD, SUB, INC, DEC, NOT, CLR, NOP
Shifter	SHL, SHR
Rotator	ROR, ROL
Permutation	PERINIC, PERBIT
Substitution	SBOXINIC, SBOX
Load/Store	LOAD, STORE
MOV/Branch	MOV, JMP, JZ , JG, JL

In general, each VLIW word is stores in one only memory position, i.e., accessing a word takes only one memory address. In this cryptoprocessor, each word has 160 bits. The VLIW word contains four 40-bit instructions simultaneously executed. Some of them occupy all the VLIW word, that is the case of SBOXINIC and PERBIT. Figure 2 shows the VLIW word format.

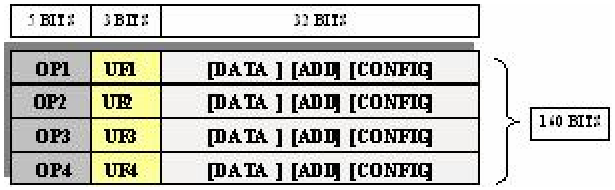

Figure 2 - VLIW word format.

Some instructions are exclusive, i.e., they cannot occur in the same VLIW word. More details as follows.

4.1. Exclusive Instructions

The exclusive instructions access the data cache and data registers. In order to maintain the coherence and integrity to the access of these devices, and it is not possible to execute more than one exclusive instruction in the same VLIW word. They are:

- LOAD – Loads the data of a specific memory address (D-CACHE)
- STORE – Stores memory data (D-CACHE)
- SBOX – Replaces data (D-CACHE).

These are exclusive instructions because in this case it is not physically possible to access the same memory simultaneously (D-CACHE).

Especial case: the **MOV** instruction moves data between registers and one cycle. Depending on the context, this instruction can generate incoherence and jeopardize the data and execution flow integrity. In this manner, the MOV instruction can be considered exclusive in some cases:

Example: VLIW word [ADD A1, B1 | INC A2 | MOV A1,A3 | NOP]

This word contains four instructions (ADD, INC, MOV e NOP) not executed simultaneously in one cycle. There is incoherence, once the A1 value may not be required after the execution. That is because the ADD and MOV instructions modify the A1 content, and the latter predominates. Therefore, in the end of the execution, A1 stores the value included in A3 (MOV A1, A3). In this context, the MOV instructions can be considered exclusive, and its execution is indicated in cycle prior or after the ADD instructions, according to the result one wants to obtain.

4.2. Special Instructions

Special instructions fit the characteristic pertaining to symmetric cryptography algorithms, such as bit permutation and substitution. They are special because they are in most part of the symmetric algorithms and holds functional units dedicated to execute them in the VLIW cryptoprocessor, although they are not in the GPP. Each special instruction can be considered a macroinstruction, that is, they are simple but execute a complex task constituted by several operations in one machine cycle.

In order to support the most number of algorithms it was necessary to define initializing parameters which configure the execution mode of the special instructions, as they are:

Permutation Instructions – PERINIC – Permutation instruction starting / PERBIT – Bits permutation (16 e permutations per cycle)

Substitution Instructions – SBOXINIC – Substitution instruction starting / SBOX – bits substitution.
SBOXINIC and PERBIT are 160-bit instructions

The permutation operation performs the permutation of bits predefined in a static permutation table by means of a specific functional unit (UF5). The detailed functioning of the substitution instructions is presented in the following section.

5. Substitution Operation (S-Box)

The substitution box (S-BOX) is a data set which constitutes a vector or bidimensional matrix. Every S-BOX is constituted by a bits input (*Input S-BOX*) which determines the substitution data location, and by a bits output *(Output S-BOX)* which returns the substitution data value. The substitution data will superscribe the bits used in the S-BOX input.

The SBOX instruction performs the substitution of predetermined bits. By means of a search by line and column, it is possible to identify the address where the substitution data is stored, and then retake and store it in the appropriate position in the resulting register, performing the substitution.

This process is executed based on some configurable parameters necessary to calculate the memory address where the substitution data is located. Some parameters may vary according to the cryptographic algorithm described, therefore the need of configurable parameters. Table 4 shows the result of a study carried on this project on how the S-BOX can range from algorithm to algorithm.

Preliminary studies showed that the following parameters are vital to implement any symmetric cryptographic algorithms S-BOX:

- The number of SBOXES of a certain algorithm
- The number of S-BOX input bits
- The number of S-BOX output bits
- The bits which determine the S-BOX line and column

This variation regarding the size of the S-BOXES, the input and output bits, and the quantity of S-BOXES makes the hardware project relatively complex. A hardware dedicated to each S-BOX described in table 4 may add speed to the circuit, but it is necessary a great number of components to implement it.

Table 4 – Comparison between different cryptography algorithms SBOXES.

Algorithm	NSBOX	Input	Output	LIN	COL
DES	8	6	4	1 e 6	2,3,4,5
AES	1	8	8	1,2,3,4	5,6,7,8
Serpent	8	4	4	--	1,2,3,4
Cast-128	4	8	32	--	1 - 8
MARS	2	8	32	--	1 - 8
Twofish	8	4	4	--	1 - 4
Magenta	1	8	8	--	1 - 8
Frog	1	8	8	--	1 - 8
BlowFish	4	8	32	--	1 - 8
LOKI97	2	14	8	--	1 - 14
RC5	--	--	--	--	--
RC6	--	--	--	--	--
IDEA	--	--	--	--	--

*NSBOX – number of SBOXES
*Input – number of S-BOX input bits
*Output – number of S-BOX output bits
*LIN, COL – bits which determine the S-BOX line and column

Developing a general S-BOX which is fast and serves most part of the symmetric algorithms is a challenging skill. In this project we propose a solution with the aid of configuration parameters by softwares, i.e., by means of an instruction, the programmer is able to configure the size of the S-BOX, input and output bits, bits which determine line and column, as well as other parameters necessary to configure the S-BOX required.

One of these future solutions may be the reconfiguration in which the hardware decides if it is necessary a new S-BOX reconfigured in execution time *(RTR – Run Time Reconfiguration)* [23].

The configuration parameters are implemented by means of the SBOXINIC instruction, and the necessary parameters and its description is presented next:

- SBOXEND – Memory address where the S-BOX is located. The S-BOX data is stored in the D-CACHE memory. It is only necessary to indicate from which address the S-BOX is started, independently from the number of S-BOXES used by the algorithm.
- SBOXCOL – Number of columns of an S-BOX.
- SBOXQ – Quantity of elements in an S-BOX.
- TBO – size of the origin block. It represents the quantity in bits of a substitution block. In the end, the origin block is replaced by the destination block, even if they are different in size. The destination block stores the substitution result.
- LIN - Origin block bits and bytes which determine the S-BOX address line. Each algorithm has a specific form to establish which bits or bytes determine the line and column of an S-BOX, where the substitution data is located. This parameter determines the bits or bytes for such line.

- COL – Destination block bits or bytes which determine the S-BOX address column in the same way as LIN.
- B – Bit or byte type, when B=0 (zero), the LIN and COL values are considered bits. When B=1, the LIN and COL values are considered bytes.

All parameters are stored in dedicated registers, and there is a physical limit of bits to the respective representation of each parameter which determines the algorithm supported by the cryptoprocessor. Table 5 shows the maximum representations of each parameter.

Table 5 – Representation limits of the substitution operation parameters

Parameter	Bits	Description
SBOXEND	16	Up to 65535 positions of memory
SBOXCOL	16	S-boxes of up to 65535 columns
SBOXQ	16	S-boxes of up to 65535 elements
TBO	6	configure blocks origin with up to 64 elements
TBD	6	configure blocks destination with up to 64 elements
LIN	32	Configure up to 2 bytes for row
COL	32	Configure up to 2 bytes for colum

The definition of the bit number of each parameter was based on table 4, where is possible to identify and quantify the bits necessary to represent the maximum number of the S-BOX columns, the maximum size of the origin and destination block, as well as other parameters. The syntax of the S-BOX start and configuration instruction is:

SBOXINIC, SBOXEND, SBOXCOL, SBOXQ, TBO, TBD, LIN, COL, B

After the initiation and configuration by meand of the SBOXINIC instruction, the operation and substitution itself may be performed. The SBOX instruction performs calculus necessary to localize the substitution data stored in the D-CACHE memory, based on configurable parameters.

The NSBOX parameter used by the instruction SBOX identifies which S-box will be used in the substitution. When the cryptography algorithm makes use of only one S-BOX, the NSBOX value is zero. In order to identify an element in the memory, it was developed a mathematic formula which is capable of identifying the required address by means of some configurable parameters (see the next expression).

$$\mathbf{END} = \underbrace{\mathbf{SBOXEND}}_{\text{Initial address}} + \underbrace{\mathbf{(SBOXC\ x\ LIN)}}_{\text{address the row of the}} + \overbrace{\mathbf{COL}}^{\text{address the column of the element}} + \underbrace{\mathbf{(SBOXQ\ x\ NSBOX)}}_{\text{address the SBOX of the element}}$$

The memory is a vector of N positions. In some algorithms, the S-BOXES are represented by matrixes (DES and AES), while in other algorithms they are represented by vectors (Serpent, Magenta and others). Independently from the S-BOX type, the formula 1 has to be used In order to identify the position of the correspondent memory. The value in END after the execution represents the memory address where the substitution data is stored. To follow, the sequency of steps necessary to execute an S-BOX instruction:

1º step: determine the S-BOX input bits

2º step: determine the line (LIN) and column (COL) value of the required substitution.

3º step: make use of the formula 1, generate the physical address of the substitution data (D-CACHE)

4º step: determine the destination block and perform the substitution

The SBOX substitution operation uses the following registers:

- A6 – stores the final result of the substitution
- B6 – stores bits to be substituted
- Configurable parameters registers: SBOXEND, SBOXCOL, SBOXQ, TBO,TBD, LIN, COL.
- AC1 – Accumulator 1, points to the origin block to be substituted
- AC2 – Accumulator 2, points to the destination block to be substituted

Every substitution, the AC1 is increased with the TBO (*tamanho do bloco de origem* - origin block size) register value, pointing to the next origin block to be replaced. The same applies to the AC2, but this is increased with the TBD value, pointing to the next substitution destination block. The SBOX initiation and configuration annul the AC1 and AC2 value.

6. VLIW Cryptoprocessor Pipeline

The VLIW cryptoprocessor holds a 3- global-stage pipeline. Each of them is executed in one machine cycle, including the instructions execution stage, and controlled by a state finite machine (control unit) synchronized by the global clock. These stages are:

- **Stage 1** – Search and dispatch of instructions. Searches a 160-bits VLIW word, received by the instruction dispatcher which directs the predefined functional units. This stage is performed in one machine cycle.

- **Stage 2** – Decoding and execution. The functional units decoding the instructions to be executed, and the required operations are accomplished.
- **Stage 3** – Writing in memory or registers. The results are stored in the appropriate registers or in the D-CACHE memory. The Harvard architecture allows the simultaneous reading and writing in different memories.

In this manner, the pipeline is filled in three machine cycles. The pipeline is never interrupted for the treatment of special instructions or any other one. Since all instructions are executed in only one cycle, the description of the hardware which controls the transition among global stages is much easier.

7. Implementation in FPGA

This section presents the performance statistics of the VLIW cryptoprocessor implemented in a FPGA Virtex II Pro. The statistics can be visualized in table 6.

Table 6 - performance statistics in FPGA

Device	Slices	LUTs	FF	PT(ns)	Freq.(MHz)
Virtex II Pro	1315	2484	669	8,618	116,036

*PT- Propagation time

The cryptoprocessor delivered a great time performance, the circuit propagation time (PT) was 8.667ns, reaching a maximum frequency of 116 Mhz, and the cryptoprocessor mapped on the FPGA device used 1315 Slices and 669 Flip-flops.

Tabela 7 - comparison among general-purpose processors

Proc.	Freq.	Memory	File	Time
P4	1,6 Ghz	128 MBytes	1 MByte	3,250s
P3	1.0 Ghz	256 Mbytes	1 MByte	4,256s
P3	800 Mhz	128 MBytes	1 Mbyte	5,307s
P3	500 Mhz	512 Mbytes	1 MByte	5,875s
VLIW	116 Mhz	1 Mbyte	1 MByte	0,36 s

The table 7 shows the comparison among general-purpose processors (P3 and P4, Pentium III and IV) when executing the DES algorithm. Such performance statistics show that the VLIW cryptoprocessor may be 16 times faster than a Pentium III 500 Mhz.

7.1. DES algorithm implementation

Unlike the sequential program, in VLIW processors the instructions are represented by horizontal microcode. An example described below presents a part of the horizontal microcode of the DES algorithm according to the cryptoprocessor assembly.

```
// expansion exchange
PERBIT  31,00,01,02,03,04,03,04,05,06,07,08,07,08,09,10
PERBIT  11,12,11,12,13,14,15,16,15,16,17,18,19,20,19,20
PERBIT  21,22,23,24,23,24,25,26,27,28,27,28,29,30,31,00

//Xor with the Ki key

MOV  B1,A3 LOAD A1,[K1] XOR  A1,B1 NOP

// S-BOX  operation start

SBOXINIC  SBOXEND,SBOXCOL,SBOXQ,TBO,TBD,LIN,COL,B
SBOX 0 NOP NOP NOP
```

An important piece of information is the utilization of the VLIW cryptoprocessor processing capacity, which indicates the utilization rate of all hardware resource available for a certain program.

To calculate this value, it is necessary to quantify the utilization percentage of each algorithm VLIW word. In our situation, one VLIW word can store 4 instructions to be executed in parallel, and the calculus consists of quantifying the instructions per VLIW word executed. In this case, the DES algorithm had a utilization of 61.18% of the cryptoprocessor capacity. This result can be considered acceptable, but not ideal – which would be 100% hypothetically. However, it is known that such utilization rate is not common. Each symmetric cryptography algorithm reaches different rates, likewise it is possible to find different descriptions for the same algorithm.

7.2. Data Dependency Impact Minimization – Loop Pipelining

Horizontal architectures such as VLIW are characterized by the possibility of controlling the several functional units independently and directly, usually by means of an only control flow, allowing the exploration of only the thinnest level parallelism. Making use of a high number of functional units is useful when there are data dependency or data control.

For the same total of functional units, the performance level can be higher if such units are divided into groups and there are parallelism techniques with effect upon the destination architecture. The previous studies of several symmetric algorithms showed that most part of them execute a high-degree-of-dependency loop among the operations called iteration.

The number of iterations ranges from algorithm to algorithm, once they are performed in a main loop. In case of symmetric cryptographic algorithms, the iterations act upon a text block of permanent size in order to cipher or decipher the required information. For instance: the DES algorithm act upon 64-bit blocks, and in this context, it is possible to apply the Loop Pipelining technique efficiently, in which the operations of different iterations can be performed in a same VLIW state or word. In this manner, the technique allows a better use of the resources.

In a first implementation of the DES algorithm, the following steps were performed: (i) capture the 64-bit block, (ii) process the 16 iterations of DES, and (iii) store the ciphered text. Such sequency reached 61,18% of effective hardware utilization.

The Loop Pipelining technique makes the execution flow more complex. By the other hand, the utilization of the available hardware resources is better, and the architecture general performance is optimized. Figure 3 shows the technique application in order to achieve a medium grain parallelism.

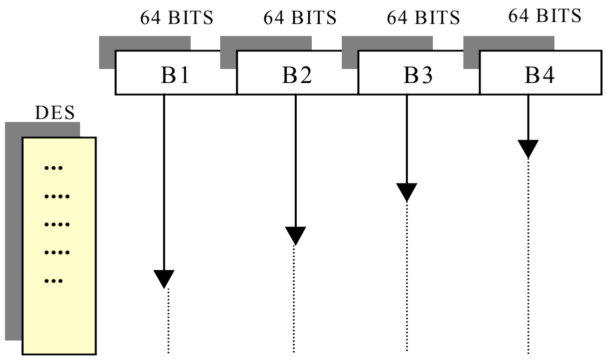

Figure 3 – Loop Pipelining Illustration

It is important to note on Figure 3 that the DES algorithm can act upon four text blocks in parallel. By using a loop pipelining, operation of different iterations and text blocks can be executed in a same state. The beginning of each block processing is required in different moments (as indicated by the arrows on Figure 3), avoiding competitiveness between two or more instructions by the same functional unit. With this technique, the hardware utilization rate rose significantly from 61.18% to 82,75%. The impact can be observed in the data shown as follows.

The implementation without the loop pipelining can execute one 64-bit block in 380 clock cycles. Thus, the relation cycles per ciphered bits is 5.94, while the implementation with loop pipelining can execute up to four 64-bit blocks in a total of 256 bits in 564 cycles. The cycle rates per bit is 2,2 cycles/bit, that is, it is necessary less cycles in order to cipher on bit in relation to the previous implementation.

The impact of such optimization in the cryptoprocessor performance, measured in ciphered text megabits per second, was significant. The maximum ciphered was 19MBits/s, and now it is 51,23MBits/s. It is important to highlight that there were not changes on the VLIW cryptoprocessor architecture.

The final implementation of the VLIW Cryptoprocessor can be visualized in figure 4 through floorplanne and figure 5 with the real system.

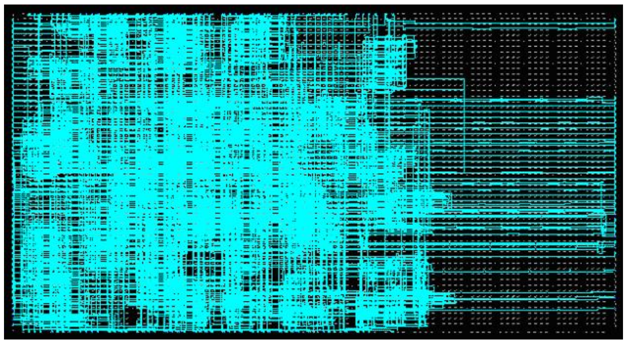

Figure 4 – Floorplanner of VLIW Cryptoprocessor

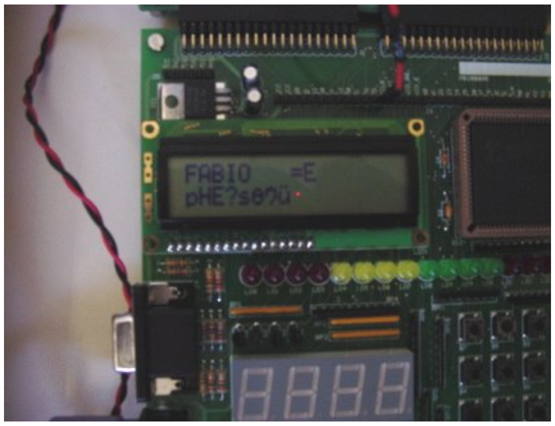

Figure 5 – Real tests with the VLIW Cryptoprocessor

8. Conclusion

In this paper, we have showed and discussed a new VLIW-based architecture, it is specifically to symmetric cryptography algorithms. This approach was project for using a FPGA and the preliminary results for executing the DES algorithm are acceptable. Thus, the FPGA cryptoprocessor implementation had a great time performance. One of the most relevant characteristics in this project is the number of algorithms supported by the cryptoprocessor. Every cryptographic algorithm in table 4 can be implemented in this architecture. The next activities of the project would be:

- Optimize the architecture, adding or removing elements such as registers and functional registers
- Study the possibility of architecture reconfiguration
- Implement a simulator and assembler for the cryptoprocessor
- Implement a high level compiler for the cryptoprocessor

9. References

[1] WEAVER, Chris, WU, Lisa, AUSTIN, Todd. CryptoManiac: A Fast Flexible Architecture for Secure Communication. Advanced Computer Architecture Laboratory, Univ. of Michigan, U.S.A, In Proc. of ISCA-2001 (Intl. Conf. On Computer Architecture), Goteborg, Sweden.

[2] BURKE, Jerome, MCDONALD, John, AUSTIN, Todd. Architectural Support for Fast Symmetric-Key Cryptography..Advanced Computer Architecture Laboratory, Univ. of Michigan, U.S.A, In Proc. of ASPLOS IX, Cambridge, 2000.

[3] PAAR, Christof, CHETWYND, Brebdon, CONNOR, Thomas, DENG, Sheng Y., MARCHANT, Steve, An Algorithm-Agile Cryptographic Coprocessor Based on FPGAs. ECE Dep Worcester Polytechnic Institute, Worcester, U.S.A, The SPIE´s Symposium on Voice and Data Communications, Sep. 1999, Bosto, U.S.A

[4] XIE, Haiyong, ZHOU, Li, BHUYAN, Laxmi. An Architectural Analysis of Cryptographic Applications for Network Processors. Department of Computer Science and Engineering, Univ. of California, Riverside. IEEE First Workshop on Network Processors, with HPCA-8, Boston, U.S.A., Feb., 2002.

[5] PAAR2, C., Reconfigurable Hardware in Modern Cryptography. Technical Report, Cryptography and Information Security Group, Electrical and Computer Engineering Department. Worcester Polytechnic Institute,2000

[6] YEOM, Dong B., LEE, Jong S., PARK, Jong S., KIM, Sang T. The Simulation and Implementation of Next Generation Encryption Algorithm RIJNDAEL. In Proc. of MIC-2002.

[7] FIPS46-2, Federal Information Processing Standards Publication 46-2, DATA ENCRYPTION STANDARD, Dec. 1993

[8] FIPS197, Federal Information Processing Standards Publication 97, ADVANCED ENCRYPTION STANDARD (AES), Nov. 2001.

[9] RIVEST, Ronald L.., The RC5 encryption algorithm, Fast Software Encryption, 2nd. International Workshop, Lec. Note in Comp. Sci. 1008, pp 86-96, Springer-Verlag, 1995).

[10] RIVEST2, Ronald L., RC6 Block Cipher, RSA Security Inc, USA, 2000.

[11] JACOBSON, M.J., HUBERY, K., The MAGENTA Block Cipher Algorithm,GERMANY, 1998.

[12] BURWICK, Carolynn, COPPERSMITH, Don, et al, MARS - a candidate cipher for AES, IBM Corporation, USA, 1999.

[13] ANDERSON, Ross, BIHAM, Eli, KNUDSEN, Lars, Serpent: A Proposal for the Advanced Encryption Standard, England, 1998.

[14] SCHNEIER, Bruce, KELSEY, John, et al Twofish: A 128-Bit Block Cipher,USA, 1998

[15] GEORGOUDIS, Dianelos, LEROUX, Damian, et al, Specification of the Algorithm THE "FROG" ENCRYPTION ALGORITHM,USA, 1998.

[16] SCHNEIER2, B., Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish), USA, 1994.

[17] ORDONEZ, E. D. M, PEREIRA F. D. et al. Algoritmos de criptografia em hardware software. III Escola Regional de Informática, RJ/ES,2003

[18] ORDONEZ2, E. D. M, PEREIRA F. D. et al. Projeto, Desempenho e Aplicações de Sistemas Digitais em Circuitos Programáveis (FPGAs), Bless Gráfica e Editora, Marília/SP, 2003.

[19] PEREIRA F. D., Ordonez E. D. M, Otimização em VHDL e Desempenho em FPGAs do Algoritmo de Criptografia DES. Quarto Workshop em Sistemas Computacionais de Alto Desempenho (WSCAD'2003), São Paulo, Nov. 2003.

[20] SCHNEIER3, B, "The IDEA Encryption Algorithm", Dr. Dobb's Journal, December 1993.

[21] ADAMS, Carlisle, "The CAST-128 Encryption Algorithm,",*RFC 2144*, May 1997.

[22] CANRIGHT, D., A Very Compact S-box for AES, CHES'2005 - Workshop on Cryptographic Hardware and Embedded Systems ,August, Edinburgh (Scotland), UK, 2005.

[23] HAGEMEYER, J., KETTELHOIT, B., PORRMANN, M., Dedicated Module Access in Dynamically Reconfigurable Systems, RAW'2006 - 13th Reconfigurable Architectures Workshop, Rhodes Island,Greece, 2006.

[24] PATERSON, K. G., YAU, A.K.L., Cryptography in Theory and Practice: The Case of Encryption in IPSec,Eurocrypt'2006 - 25th International Cryptology Conference, Saint Petersburg, Russia, 2006.

[25] SEDCOLE, P., BLODGET, B., BECKER, T., ANDERSON J., LYSAGHT, P., Modular PartialReconfiguration in Virtex FPGAs, International Conference on Field Programmable Logic and Applications, Finland, 2005.

Fabio Dacêncio Pereira. Received the BS and MSc degrees in Computing Sciences from UNIVEM – Euripides Foundation of Marilia – Brazil, in 2002 and 2004. He is pursuing the PhD at the University of Sao Paulo, working in strategies on hardware security. He now is professor at the IST – UNIVEM.
He is interested in reconfigurable computing, hardware security and digital systems.

Edward David Moreno. Received the MSc. and PhD degrees in Electrical Engineering from University of Sao Paulo - Brazil, in 1994 and 1998. During 1996 and 1997 he stayed as invited researcher at University of Toronto, Canada, and Chalmers University of Technology, Sweden. The research areas are: computer architecture, reconfigurable computing, embedded systems and hardware security.

Rodolfo Barros Chiaramonte. Received the BS and MSc degrees in Computing Sciences from UNIVEM – Euripides Foundation of Marilia – Brazil, in 2004 and 2006. He now is professor at the IST – UNIVEM. He is interested in information security, distributed systems and reconfigurable computing.