Design and development of a recongurable cryptographic co-processor

Daniele Fronte

N° d'ordre: Publique

Thèse

présentée devant

l'Université de Provence

pour obtenir

le grade de DOCTEUR DE L'UNIVERSITÉ DE PROVENCE
Mention SCIENCES POUR L'INGÉNIEUR: MÉCANIQUE, PHYSIQUE, MICRO ET NANOÉLECTRONIQUE

par

Daniele FRONTE

Titre de la thèse :

Design and development of a reconfigurable cryptographic co-processor

Soutenue le 8 juillet 2008 devant la commission d'examen :

MM. :	Michele ELIA	Politecnico di Torino, Rapporteur
	Lionel TORRES	Université de Montpellier II, Rapporteur
Mme :	Dominique BORRIONE	Université J. Fourier de Grenoble, Examinateur
MM. :	Jean-Michel PORTAL	Université d'Aix–Marseille I, Examinateur
	Eric PAYRAT	Société Atmel Rousset, Superviseur industriel de thèse
Mme :	Annie PEREZ	Université d'Aix–Marseille I, Directeur de thèse
M. :	Luc JEANNEROT	Société Atmel Rousset, Invité

Contents

Glossary

ADS ARM Developer Suite

AES Advanced Encryption Standard

AHB Advanced High-performance Bus

ALU Arithmetic Logic Unit

APB Advanced Peripheral Bus

ASIC Application Specific Integrated Circuit

ATM Automatic Teller Machine

Bit Binary digit

CBC Cipher Block Chaining

CLA Carry Look-ahead Adders

CLB Configurable Logic Block

CMOS Complementary Metal Oxide Semiconductor

CPU Central Processing Unit

CRAM The Memory included in the Crypto Engine

CRT Cathode Ray Tube

CSA Carry Save Adders

DES Data Encryption Standard

DMA Direct Memory Access

DPA Differential Power Analysis

DRP Dynamically Reconfigurable Processor

EAL Evaluation Assurance Level

ECB Electronic Codebook

EMV Europay, MasterCard and Visa

et al. et alii (and others)

FF Flip Flop

FIFO First-In-First-Out

FIPS Federal Information Processing Standard

FPGA Field Programmable Gate Array

FSM Finite State Machine

GF Galois Fields

GPP General Purpose Processor

GPRS Global Packet Radio Service

GSM Global System for Mobile communications

HDL Hardware Description Language

HMEM Horizontally Memory

IBM International Business Machine Corporation

IC Identity Card

ID Identity Document

IT Information Technology

LCD Liquid Crystal Display

LSB Least Significant Bit

ME Modular Exponentiation

MIMD Multiple Instruction Multiple Data

MSB Most Significant Bit

NIST National Institute of Standards and Technology

NSA National Security Agency

PE Processing Element

PIN Personal Identification Number

PIO Parallel Input/Output

POS Point Of Sale

PP Protection Profile

RAM Random Access Memory

RPA Refined Power Analysis

RSA Rivest Shamir Adleman

RTL Register Transfer Level

SC Smart Card

SCA Side Channel Attacks

SHA Secure Hash Algorithm

SIM Subscriber Identity Module

SIMD Single Instruction Multiple Data

SPA Simple Power Analysis

SRAM Static RAM

STC State Transition Controller

TPS Télévision par Satellite

UMTS Universal Mobile Telecommunication System

VHDL Very-high-speed integrated circuit Hardware Description Language

VMEM Vertically Memory

WURM autonomous Wearable Unit with Reconfigurable Modules

WWII World War II

ZPA Zero Power Analysis

A Francesco e Andrea

Ad Maiora

Abstract

Nowadays hi-tech secure products need more services and more security. Furthermore the corresponding market is now oriented towards more flexibility. In this thesis we propose as novel solution a Multi-algorithm Cryptographic Co-processor called Celator. Celator is able to encrypt or decrypt data blocks using private key encryption algorithms such as Advanced Encryption Standard (AES) [1] or Data Encryption Standard (DES) [2]. Moreover Celator allows condensing data using the Secure Hash Algorithms (SHA) [3]. These algorithms are frequently implemented in hi-tech secure products in software or in hardware mode. Celator belongs to the class of the flexible hardware implementations, and allows an user implementing its own cryptographic algorithm under specific conditions.

Celator architecture is based on a 4x4 Processing Elements (PE) systolic array, a Controller with a Finite State Machine (FSM) and a local memory. Data are encrypted or decrypted by the PE array.

This thesis presents Celator architecture, as well as its AES, DES, and SHA basic operations. Celator performances are then given and compared to other security circuits.

1

Introduction

The trend of the hi-tech secure products market is to offer more services and more security to users. More services and security require flexible functions. It can happen that an electronic device needs to execute further algorithms than those it was designed for; therefore such devices must be flexible and reconfigurable.

Our challenge is to implement a multi-algorithm Crypto-Co-Processor, called Celator. Celator is conceived to be integrated in an ASIC for embedded circuits with Atmel Standard Cells. More precisely, Celator can be included in Smart Cards and in Smart Card Readers.

By using simple logic and arithmetic operations, Celator can perform the following algorithms:

- Advanced Encryption Standard (AES-128, AES-196 and AES-256), [1]
- Data Encryption Standard (DES), [2]
- Secure Hashing Algorithm (SHA-256), [3]

The adopted solution for Celator is based on a 4x4 Processing Elements (PE) systolic array, which seems a good solution to compute all matrix format data. PE array's data path is reconfigurable, just as the Finite State Machine which controls the PE array is too. Because of these two reconfigurable elements, Celator can be reconfigured. Results show that Celator is a good trade off with respect to the execution cycles, to the area and to the flexibility, between a dedicated hardware macro, and a multi-purpose processor.

This thesis presents my published works [4], [5], [6], [7], [8], [9] and the contribution of [10].

The rest of the chapter is organized as follows. Section 1.1 presents a brief history of the security techniques. Section 1.2 introduces the Celator work environment. In section 1.3 several side channel attacks are disclosed. Some conclusions of this chapter are given in section 1.4.

1.1 Security and insecurity

1.1.1 From Herodotus to cryptographic processors

Cryptography, deriving from Greek "kryptós" hidden, and the verb "gráfo" write, is the study of message secrecy.

Encryption attempts to ensure secrecy in communications, such as those of spies, military leaders, and diplomats, but it has also had religious applications. For instance, early Christians used cryptography to obfuscate some aspects of their religious writings to avoid the near certain persecution.

Before the modern era, cryptography was concerned solely with message confidentiality (i.e. encryption). In recent decades, the field has expanded beyond confidentiality concerns to include techniques for message integrity checking, sender/receiver identity authentication, digital signatures, interactive proofs, and secure computation, amongst others.

The earliest forms of secret writing required little more than local pen and paper analogs, as most people could not read. More literacy, or opponent literacy, required actual cryptography. The main classical cipher types were transposition ciphers and substitution ciphers, which systematically replace letters or groups of letters with other letters or groups of letters. An early substitution cipher was the Caesar cipher, in which each letter in the plaintext was replaced by a letter some fixed number of positions further down the alphabet. It was named after *Julius Caesar* who is reported to have used it, with a shift of 3, to communicate with his generals during his military campaigns.

Steganography (i.e. hiding even the existence of a message so as to keep it confidential) was also first developed in ancient times. An early example, from *Herodotus*, concealed a message – a tattoo on a slave's shaved head – under the regrown hair. More modern examples of steganography include the use of invisible ink, microdots, and digital watermarks to conceal information.

Various physical devices and aids have been used to assist with ciphers. One of the earliest may have been the scytale of ancient Greece [11], the cipher grille in medieval times, the Alberti's own cipher disk for the polyalphabetic ciphers (circa 1460), the Johannes Trithemius' tabula recta scheme, and Thomas Jefferson's multi-cylinder (invented independently by Bazeries around 1900). Early in the 20th century, several mechanical encryption/decryption devices were invented, and many patented, including rotor machines – most famously the Enigma machine used by Germany in World War (WW) II [12].

The development of digital computers and electronics after WWII made possible much more complex ciphers. Today digital computers allows a large use of the cryptography not only for government necessity but also for secrecy in ordinary communications, e-commerce secure transaction, trusted ID etc.

1.1.2 The Evaluation Assurance Level

A standard security measure for cryptographic algorithms or for their hardware and software implementations does not exist yet. Nevertheless, there are some criteria to evaluate their security level.

One of these criteria is the evaluation of the difficulty to break a given cryptographic algorithm, with respect to a well-known mathematical problem. Id est, the security level of many cryptographic algorithms can be associated to the difficulty of certain computational problems, such as the integer *factoring* problem or the *discrete logarithms in a finite field* problem [13, chapter 11]. There are proofs that cryptographic techniques are secure if a certain computational problem cannot be solved efficiently [14]. These proofs are contingent, and thus not definitive, but are currently the best available for cryptographic algorithms and protocols. Example given, as the main operation in RSA based algorithms is the modular exponentiation in $\mathrm{GF}(2^n)$, thus the security level of RSA based algorithms is associated to the discrete logarithms in a finite field problem.

Since 1999 the international standards for security evaluation are defined by the Common Criteria [15] as Evaluation Assurance Level (EAL1 through EAL7) of an Information Technology product or system [16]. The EAL is a numerical grade assigned following the completion of a Common Criteria security evaluation. The increasing assurance levels reflect added assurance requirements that must be met to achieve Common Criteria certification. The intent of the higher *levels* is to provide higher *confidence* that the system's principal security features are reliably implemented. The EAL level does not measure the security of the system itself; it simply states at what level the system was tested to see if it meets all the requirements of its Security Target. The Security Target is a document that describes the assets to protect in the system, the threats that are identified on these assets, and the security objectives that are to be achieved by the system security. Then, it describes the system security by listing the security requirements, that are written using the Common Criteria restricted syntax.

A Security Target is generally written to be compliant to a Protection Profile. The PP is a document, typically created by a user or user community, which identifies security requirements relevant to that user for a particular purpose. A PP effectively defines a class of security devices; for example, Smart Cards used to provide digital signatures, or network firewalls.

A PP is associated to a product, and it is dedicated to a particular EAL.

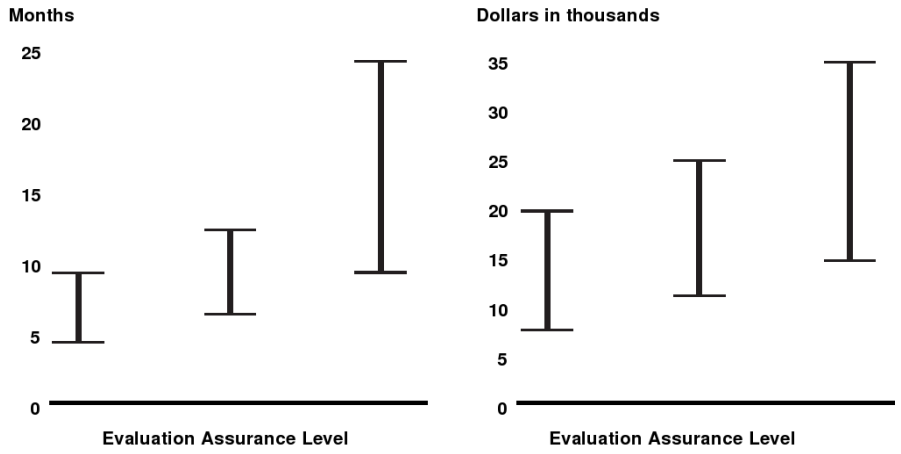
EAL5+ has become the standard in the SC business, with the PP called SC IC Platform Protection Profile, under the reference BSI-PP-0002 [17]. The EAL5+ provides more security confidence than the EAL5 does, but less than the EAL6 does. The BSI-PP-0002 specifies the security target for SC. The perimeter of the Target Of Evaluation (TOE) can be one or more assets to be protected, and one or more threats. For instance, perimeter of the BSI-PP-0002 can be the protection:

- of the user data (asset)
- from the fault injection attack that would try to divulge or modify the asset (threat)

To achieve a particular EAL, the computer system must meet specific assurance requirements. Most of these requirements involve design documentation, design analysis, functional testing, or penetration testing. The higher EALs involve more detailed documentation, analysis, and testing than the lower ones. Achieving a higher EAL certification generally costs more money and takes more time than achieving a lower one. The EAL number assigned to a certified system indicates that the system completed all requirements for that level. A higher EAL means nothing more, or less, than the evaluation completed a more stringent set of quality assurance requirements. It is often assumed that a system that achieves a higher EAL will provide its security features more reliably, but there is little or no published evidence to support that assumption. The required third-party analysis and testing performed by security experts is reasonably evidence in this direction. An evaluation example of an operating system is presented in [18].

In 2006, the US Government Accountability Office published (Figure 1.1) a report on Common Criteria evaluations that summarized a range of costs and schedules reported for evaluations performed at levels EAL2 through EAL4.

Figure 1.1 – ***Range of completion times and costs for Common Criteria evaluations at EAL2 through EAL4.*** *Source [19]*



Receiving a particular EAL takes a lot of time and a lot of money. Celator has not been evaluated by the Common Criteria yet. A next step of the research shall achieve this evaluation.

1.2 From the Smart-Cards to the secure products

1.2.1 Smart Cards

A Smart Card (SC) is defined as a pocket-sized card with embedded integrated circuits which can process information. This implies that the SC can receive input which is processed – by way of the SC applications – and delivered as an output. There are two broad categories of SC:

- Memory SC: they contain only non-volatile memory storage components, and some specific security logic. First generation of SC were typically memory SC.

They can offer a light security.

- Microprocessor based SC: they contain microprocessor, memory medium (volatile and/or non volatile) and other peripheral components (net-card, sound card etc.). This kind of SC offer a stronger security than memory SC.

The microprocessor based SC are used to ensure confidentiality and information integrity, for private data like bank transitions. Therefore the cards need to be protected, and be robust to attacks. Several security levels must be reached, from the support protection to the card protection, from the embedded operating system to the applications that run on them (see section 1.3).

Even if the SC was invented at the beginning of the seventies (first patent about SC was granted to the German scientist Helmut Gröttrup and Jürgen Dethloff [20] in 1972, and to the French inventor Roland Moreno [21] in 1974), the first mass use of the cards was pre-payed cards for French public phones, starting in 1983 (Télécarte).

The major boom in SC use came in the nineteens in Europe, with the introduction of the smart-card-based SIM used in Global System for Mobile communication (GSM) mobile phone equipments. With the ubiquity of mobile phones in Europe, SC have become very common.

1.2.2 A secure Environment

Smart cards have a small gold chip measuring about 1cm by 1cm on the front (Figure 1.2). When inserted into a reader, the chip makes contact with electrical connectors that can read information from the chip and write information back.

The ISO/IEC 7816 and ISO/IEC 7810 series of standards define:

- the positions and shapes of the electrical connectors
- the electrical characteristics
- the communication protocols

- the format of the commands sent to the card and the responses returned by the card
- robustness of the card
- the functionality

Figure 1.2 – ***Smart Card overview.*** *Each Smart Card has 8 Input/Output pins.*

C1: VCC	C5: GRD
C2: RST	C6: no connect
C3: CLK	C7: I/O
C4: D- (USB)	C8: D+ (USB)

The cards do not contain batteries; energy is supplied by the card reader. SC Readers are used as a communication bridge between the SC and a host, e.g. a computer, a Point Of Sale (POS) terminal, or a mobile telephone.

Most advanced SC are equipped with dedicated cryptographic hardware. Today's cryptographic SC are also able to generate key pairs on board, to avoid the risk of having more than one copy of the key. The reconfigurable cryptographic coprocessor we present here can be included in a SC, and the user will be able to select among several cryptographic algorithms.

Such SC are mainly used for digital signature and secure identification trough authentication mechanism. The most widely used cryptographic algorithms in SC (excluding the GSM so-called crypto algorithm) are Data Encryption Standard (DES), or its improved version 3DES (Triple DES), and RSA. The key set is usually loaded (for DES) or generated (for RSA) on the card at the personalization stage. Even if nowadays the Advanced Encryption Standard (AES) is preferred to the DES based algorithms because of its stronger security, both AES and DES based algorithms are

largely used in many IT products, e.g. in e-passports. Another common algorithm in IT products is the Secure Hash Algorithm (SHA), used to sign a clear or encrypted message. Our cryptographic coprocessor will be able to perform AES, DES and SHA.

1.2.3 The Smart Cards market trend

The hardware and software architecture for a SC depends on its targets, which are strictly correlated to the SC market. There are four main SC markets (Figure 1.3):

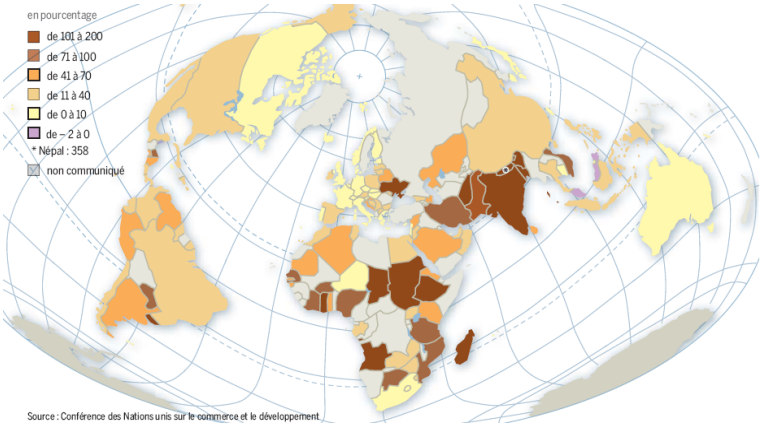
1. radio-mobile telephony (Figure 1.4)
2. banking (Figure 1.6)
3. multimedia (TV on demand, satellite etc., Figure 1.7)
4. ID (e-passports, health cards etc., Figure 1.8)

Figure 1.3 – ***Example of SC uses: radio-mobile telephony, multimedia ID etc ...***

Figure 1.4 – ***Sim cards.*** *They include a Smart Card*

In Europe, all GSM, Global Packet Radio Service (GPRS) and Universal Mobile Telecommunication System (UMTS) mobile phones are equipped with a SIM card (Figure 1.4). It is required to secure the identification process of of phone service users. The SIM cards are sold to the users by the phone operators, and can offer many services to the customers, e.g. they allow to access to the Internet. The radio-mobile telephony market is big. It started in the nineties but now it is almost saturated, especially in Europe: almost everyone owns a mobile phone, which is equipped with a SIM card, and does not change it very often ([22], Figure 1.5)! In this way, the current European radio-mobile telephony market is powered by people who buy a new mobile line, loose its mobile phone with the SIM card etc.

Figure 1.5 – ***Growth in the number of mobile telephone subscribers worldwide, 2005–2006.***



Mobile manufacturers usually do not appreciate SC, typically because they do not produce SC, and also because they consider the SIM cards like an hardware intrusion

of the phone operators in their mobile devices.

Figure 1.6 – ***Credit cards.*** *They include a Smart Card*

The international payment brands Europay, MasterCard and Visa (EMV) agreed in 1993 to work together to develop the specifications for the SC in payment cards used as either a debit or a credit card with the EMV. For the banks interested in introducing SC in credit cards (Figure 1.6), the only quantifiable benefit is the ability to forecast a significant reduction in fraud, in particular counterfeit, loss and stealing. The current level of fraud experienced by a country determines if there is a business case for the financial institutions.

Figure 1.7 – ***Télévision Par Satellite cards.*** *They include a Smart Card*

The development of the multimedia market relies on the diffusion of the satellite TV decoders, and the services like the TV on demand (Figure 1.7). The banking and multimedia markets are smaller than the radio-mobile telephony one. Nevertheless, a banking dedicated SC is typically more expensive (high added value SC) than a radio-mobile telephony one, because the banking identifying process requires more sophisticated security levels.

The ID market is a new market and its expansion is in progress everywhere. Indeed, SC are also being introduced in personal identification and entitlement schemes at regional, national, and international levels. Citizen cards, drivers' licenses, and patient card schemes are becoming more prevalent, and contactless SC are being integrated

into biometric passports to enhance security for international travels. These markets are currently growing. In France, a leader country in developing and using the SC, the *Carte Vitale* (Figure 1.8) includes a SC which secures the identification of patients. Next generation of this health card will also store the history of cares and the details of the last medicines taken by the card owner.

Figure 1.8 – ***A French Carte Vitale.*** *It includes a Smart Card*

1.2.4 Smart Card Readers

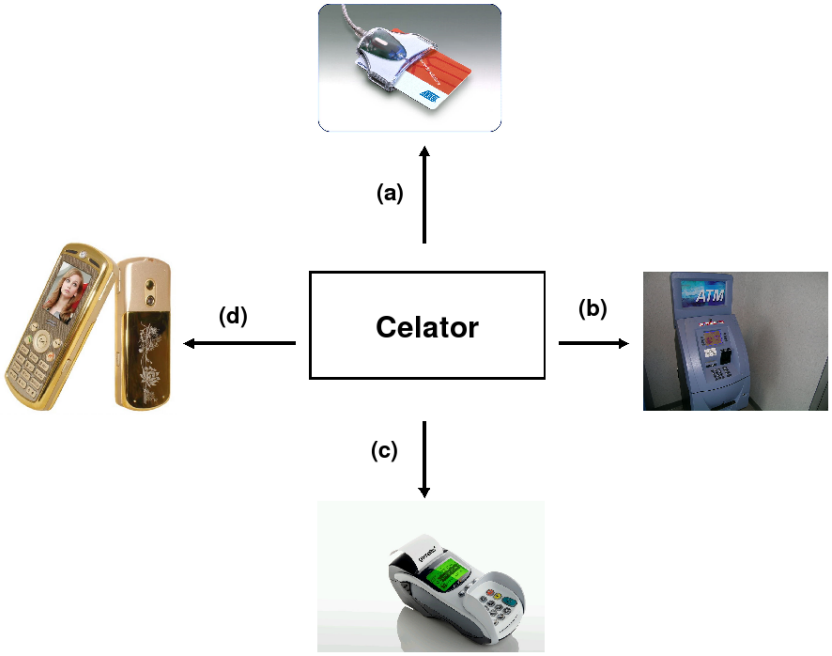
A Smart Card Reader reads the data off a SC. SC Readers are used as a communication device between the SC and a host, e.g. a Personal Computer link SC Reader, a POS terminal, an Automatic Teller Machine (ATM) or a mobile telephone (Figure 1.9). Celator can be included in a SC Reader.

In order to check and ensure the customer identity and authenticity, a SC Reader can ask the user to enter its Personal Identification Number (PIN), and then performs one or more cryptographic algorithms. The algorithms to be used depend on the service required by the user. For instance, the AES can be used to encrypt a message, while the SHA can be used to sign a message. Of course, a SC Reader that is able to execute several cryptographic algorithms, can offer more services to users than mono algorithm ones.

1.3 Side channel attacks

Using some cryptographic algorithms can not be enough to ensure the identification and the access to confidential data like bank account, because SC and SC Readers can

Figure 1.9 – ***Celator can be included in a SC Reader like a PC link SC reader (a), an ATM (b), a POS (c), a mobile phone (d)***



leak information if they are not protected from attacks. They have to be preserved against attacks.

Several kinds of attacks exist:

1. social attacks against the people who develop or use the SC
2. static physical attacks (power is not supplied)
3. dynamic physical attacks (power is supplied)
4. passive logical attacks (the hacker tries to obtain information from encrypted data)
5. active logical attacks (the hacker is able to manipulate encrypted data)

These kinds of attacks cannot be achieved at once. Several studies, hardware and software developments at different levels are required as countermeasures. We will detail the above type 3 attack, and more particularly the side channel attack.

In cryptography, a side channel attack is any attack based on information gained from the physical implementation of a crypto-system, rather than theoretical weaknesses in the algorithms, which is the aim of cryptanalysis. For instance, examples of side channel attacks are the following ones:

- timing analysis attack based on the measure of the time execution for certain arithmetic or logical operations;
- power analysis attack based on the power analysis during the execution of a given algorithm;
- TEMPEST (also known as *van Eck*) attack based on the analysis of the Electro-Magnetic radiation emissions;
- acoustic analysis based on the measures of the noise emitted by the SC during a given operation.

In all cases, the underlying principle is that physical effects caused by some operations of a crypto-system (*on the side*) can provide useful extra information about secrets in the system, for example, the cryptographic key, partial state information, full or partial plaintexts and so forth.

Next sections will detail the various side channel attacks.

1.3.1 Timing analysis

A *timing attack* watches I/O data movement of the CPU and of the memory, while one algorithm is running. Simply by observing how long it takes to transfer key information, it is sometimes possible to determine how long the key is. Internal operational stages in many cipher implementations provide information (typically partial) about the plaintext, key values and so on, and some of this information can be inferred from observed timings. Alternatively, a timing attack may simply watch for the time a cryptographic algorithm requires.

One of possible countermeasure is to employ the same time to perform all supported algorithms. For instance, the encryption and the decryption must have the same execution time. If one operation is faster than the other one, some random operations which do not modify the final result (masking data as shown in [23], no-operations etc.) can be added.

1.3.2 Power dissipation analysis: SPA, DPA

A *power monitoring attack* can provide similar information by observing the power lines to the hardware, especially the CPU. As with a timing attack, considerable information is inferable for some algorithm implementations under some circumstances.

Among these attacks, first to be developed was the Simple Power Analysis (SPA). The current power samples are analysed in order to obtain information. The following operations are considered as leaks and can be attacked by SPA:

- writing "1" or "0" into the storage mediums (RAM, ROM, registers etc.): the transition current from the p-plan to the n-plan (and vice versa) of CMOS transistors are different, therefore writing an "1" is different than writing a "0";
- comparing data value stored in memory (e.g. in the conditional branching) can cause a variation of the power consumption;
- the execution of certain operations like the power elevation, in which there is an high correlation between the time during (and then the power consumption) of the operation itself and the power exponent.

In 1999 the SPA attacks could be performed easily and they cost 400$ only, as it is detailed in [24].

Another power analysis based attack more efficient than the SPA is the Differential Power Analysis (DPA) attack, which works even on small signals [25]. In order to perform a DPA, first an attacker must be able to precisely measure the power consumption. Second, the attacker needs to know what algorithm is computed, and third

an attacker needs the plaintexts or ciphertexts. The strategy of the attacker is to make a lot of measurements, and then divide them with the aid of some oracle into two or more different sets. Then, statistical methods are used to verify the oracle. If and only if the oracle was right, one can see noticeable peaks in the statistics.

A direct countermeasure against SPA and DPA is to parallelize all computations. In this way the electrical noise produced can make the power analysis stronger to be performed. The coprocessor's architecture we present here allows to parallelize the computations. Furthermore, Atmel technology we used, allows to *secure* write "1" and "0" into the memory. Therefore we will consider the writing operations as trusted ones.

1.3.3 Electromagnetic analysis

As a fundamental and inevitable fact of electrical life, current fluctuations generate radio waves, which are the currents subject – at least in principle – to a TEMPEST or *van Eck* attack. If the currents concerned are patterned in distinguishable ways, which is typically the case, the radiation can be recorded and used to infer information about the operation of the associated hardware.

If the relevant currents are those associated with a display device (i.e. highly patterned and intended to produce human readable images), the task is greatly eased. Cathode Ray Tube (CRT) displays use substantial currents to steer their electron beams and they have been 'snooped' in real time with minimum cost hardware from considerable distances (hundreds of meters have been demonstrated). Liquid Crystal Displays (LCDs) require and use smaller currents and are less vulnerable than CRT displays – which is not to say they are invulnerable.

As we said in the previous section, a parallel architecture allows a good protection even against TEMPEST attack, because the computing data are dispatched in several components working concurrently. Celator can exploit this protection against TEMPEST attack thanks to its parallel structure.

1.3.4 Acoustic analysis

As an inescapable fact of electrical life in actual circuits, flowing currents heat the materials through which they flow. These materials also continually transmit heat to the environment due to other equally fundamental facts of thermodynamic existence, so there is a continually changing thermally induced mechanical stress as a result of these heating and cooling effects. That stress appears to be the most significant contributor to low level acoustic (i.e. *noise*) emissions from operating CPUs. Recent research by Shamir *et al.* [26] has demonstrated that information about the operation of cryptosystems and algorithms can be obtained in this way by the so-called *acoustic attack*. This kind of attack is easy to perform hardware machines which include big CPU and hard disk.

1.4 Conclusions

In this chapter we have presented how the security techniques have changed from old Greeks to nowadays. Smart cards are used to secure confidential data, ensure the privacy, provide the authenticity and the integrity of an information message. SC and SC Readers include cryptographic algorithms. Moreover they have to be side channel attack resistant.

The rest of the thesis is organised as follows. The Chapter 2 describes some algorithms implemented in Celator, i.e. AES, DES and SHA. The state of the art of the hardware and software cryptographic implementations is disclosed in Chapter 3. The Celator hardware architecture is detailed in Chapter 4. The Celator software programming is shown in Chapter 5. Finally some conclusions are given in Chapter 6.

2

Three cryptographic algorithms

This Chapter briefly introduces three algorithms that have been implemented into Celator: the AES, the DES and the SHA. The reader can find the complete description of them in [1, 2, 3]. The AES, the DES and the SHA are presented in sections 2.1, 2.2 and 2.3, respectively.

Table 2.1 – ***The 4x4 byte matrix for an AES-128 data block.***

d_{11}	d_{12}	d_{13}	d_{14}
d_{21}	d_{22}	d_{23}	d_{24}
d_{31}	d_{32}	d_{33}	d_{34}
d_{41}	d_{42}	d_{43}	d_{44}

2.1 The AES algorithm

The Advanced Encryption Standard (AES) specifies a Federal Information Processing Standards (FIPS) [1] approved cryptographic algorithm that can be used to protect electronic data. The AES algorithm is a symmetric block cipher that can encrypt or decrypt information. The AES algorithm is capable of using cryptographic keys of 128, 192 and 256 bits. These different versions are called AES-128, AES-192 and AES-256 respectively, and all versions can be performed by Celator. In this work, we focus on the AES-128. The plain text consists of 128-bit data blocks. Each block can be managed as a matrix of 4x4 bytes (Table 2.1).

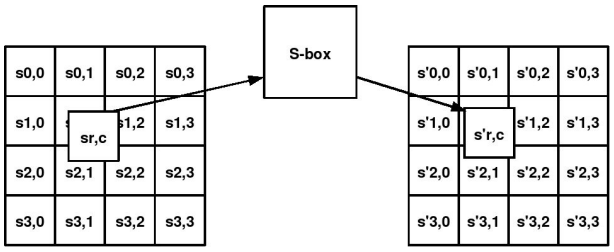
The AES encryption process includes 10 rounds. Each round (excepting the first one and the last one) involves the following transformations:

1. Sub-Bytes transformation
2. Shift-Rows transformation
3. Mix-Columns transformation
4. Add-Round-Key transformation

Sub-bytes transformation A round of AES starts with the Sub-Bytes transformation (Figure 2.1). All data of the array are substituted by using Sbox tables [1].

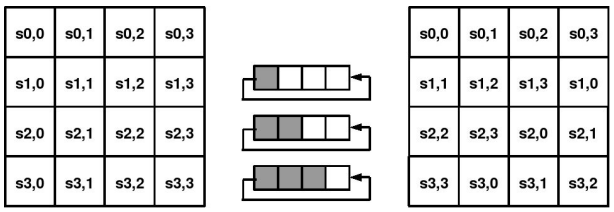
Shift-Rows transformation The second transformation of a round is the Shift-Rows (Figure 2.2): each row of the matrix is left shifted by 1, 2, or 3 positions respec-

Figure 2.1 – ***S-Box transformation***



tively for row 1, 2 or 3.

Figure 2.2 – ***Shift-Rows transformation.*** *This transformation cyclically shifts the last three rows in the state* $S(x)$.



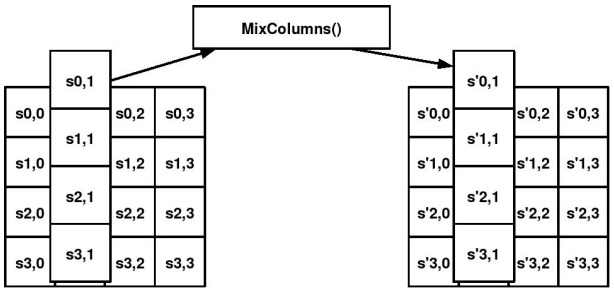
Mix-Columns transformation After the Shift-Rows, the following AES transformation to be executed is the Mix-Columns (Figure 2.3). This transformation linearly combines all the data in each whole column. More precisely, 4 vectors are applied to linearly transform the 4 columns. Celator must perform the following matrix multiplication:

$$S'(x) = A(x) * S(x) \tag{2.1}$$

with

$$A(x) = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \tag{2.2}$$

Figure 2.3 – ***Mix-Columns transformation.*** *This transformation operates on the state column-by-column.*



where $S(x)$ is the present state of the data, and $A(x)$ is the matrix made of multiplicative vectors, as it is explained in [1, pag. 18].

$$\begin{pmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \bullet \begin{pmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{pmatrix} \qquad \text{for } 0 \leq c < 4 \tag{2.3}$$

Celator could perform the Mix-Columns transformation in two ways. In the first one, the logarithmic tables [27] and some lookup tables are used; in the second way, the *xtime* function is used [1].

Mix-Columns with lookup tables. Assume that a and b are two bytes from the state matrix and multiplicative vectors matrix, respectively. The equation (2.1) can be transformed in an addition by using the following logarithmic property:

$$c = a * b \tag{2.4}$$

That is equivalent to:

$$c = \log^{-1}\left[(\log a) + (\log b)\right] \tag{2.5}$$

where $\log(a)$ or $\log(b)$ means a lookup from a logarithmic table (see [27]), providing

the power representation of a and b. Then $\log(a)+\log(b)$ should be modulated by 255. Another lookup, from the inverse logarithmic table, is needed to obtain the polynomial basis representation of the result c, which is a byte from the next state matrix.

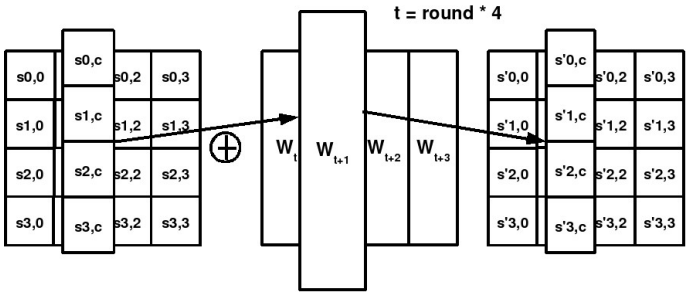
For each byte of the state matrix $S(x)$, Celator uses the logarithm properties in order to perform (2.5) under the Sequencer control.

Mix-Columns with *xtime* function. In order to compute the multiplication (2.1), each byte of the state (i.e. $S(x)$) is multiplied by the polynomial vector (i.e. $A(x)$, as described in [1]) by the *xtime* and *xor* functions.

In section 5.1.1 we will detail the adopted choice for the Mix-Columns transformation, and its implementation in Celator based on the *xtime* function.

Add-Round-Key transformation The last transformation Add-Round-Key of the encryption round combines the key values W_t related to the current *round*, with the present state S of the data through an exclusive XOR (Figure 2.4).

Figure 2.4 – ***Add-Round-Key transformation.*** *This transformation xors each column of the state with a word from the key schedule.*



2.2 The DES algorithm

The Data Encryption Standard (DES) algorithm was developed at International Business Machine Corporation (IBM), as a modification of an earlier system known as

LUCIFER. DES was first published in the US Federal Register in 1975. After a considerable amount of public discussion, DES was adopted by the NIST as a standard in 1977, and has become one of the most widely used cryptosystem in the world [2]. Nowadays DES is still very used, especially the enhanced DES, so-called 3DES.

DES is a block cipher which operates on plaintext 64-bit blocks and returns ciphertext blocks of the same size, using a key which is a bitstring of length 56 bits. The 3DES operates on plaintext 64-bit blocks using a 168-bit key.

The algorithm proceeds in three stages (Figure 2.5a):

1. Given a plaintext x, the bitstring x_0 is constructed by permuting the bits of x according to a (fixed) Initial Permutation (IP). We write $x_0 = \text{IP}(x) = L_0 R_0$ where L_0 comprises the MSB 32 bits of x_0 and R_0 the LSB 32 bits:

$$x_0 = \{ \underbrace{MSB(x_0)}_{R} , \underbrace{LSB(x_0)}_{L} \} \tag{2.6}$$

2. for $i = 1$ to 16

$$\begin{aligned} L_i &= R_{i-1} \\ R_i &= L_I \oplus f(R_{i-1}, K_i) \end{aligned} \tag{2.7}$$

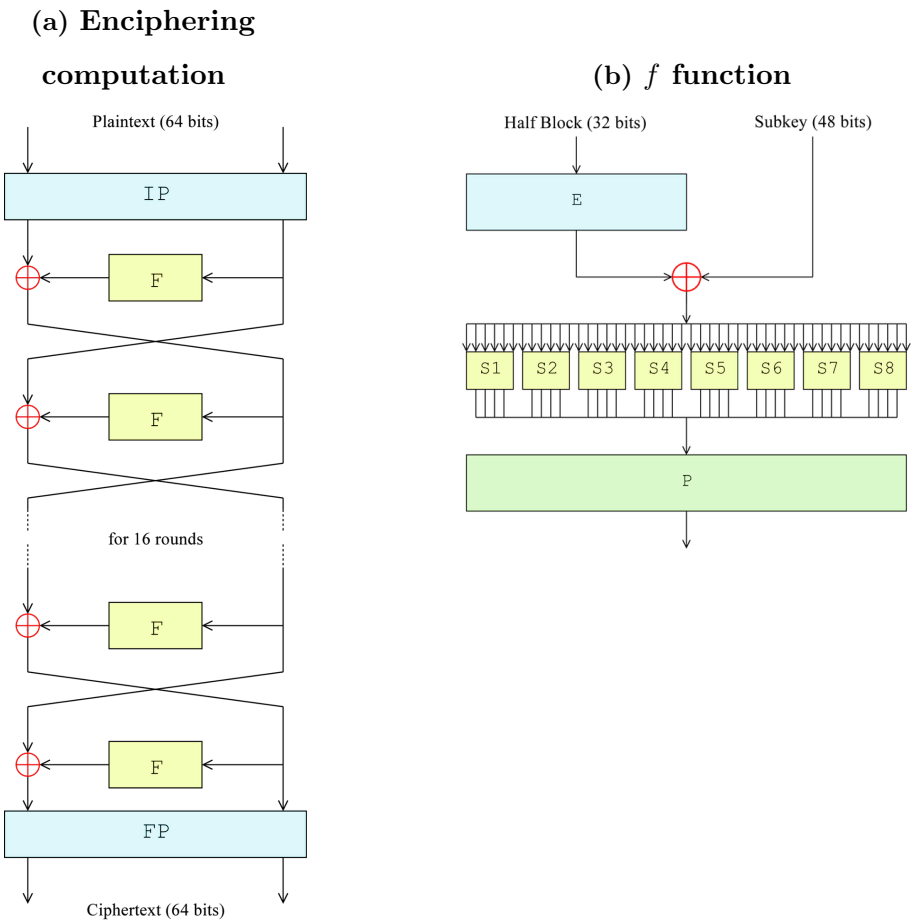
   where $\oplus$ denotes the exclusive-or of two bitstrings. f is a function that we will describe later, and K_1, $K_1 \ldots K_{16}$ are each bitstrings of length 48 computed as a function of the key K_i.

3. Apply the inverse permutation IP^{-1} to the bitstring $R_{16}L_{16}$ obtaining the ciphertext y. That is, $y = IP^{-1}(R_{16}L_{16})$. Note the inverted order of R_{16} and L_{16}.

The f function is depicted in Figure 2.5b. Basically, it consists of

- the E expansion, from 32 to 48 bits, using the E table (Table 2.2).
- the xoring with a 48-bit key word
- the substitution using an S-box

Figure 2.5 – ***DES algorithm.*** *Source: Wikimedia Commons.*



- the permutation P

Figure 2.6 illustrates the DES key schedule:

1. The 56-bit key K is expansed to 64-bits by adding 8 zeros. Then the 64-bit key K is permuted according to the permutation PC_1. We will write $PC_1(K) = C_0D_0$, where C_0 comprises the MSB 28 bits of $PC_1(K)$ and D_O the LSB 28 bits.

2. for $i = 1$ to 16

$$\begin{aligned} C_i &= LS_i(C_{i-1}) \\ D_i &= LS_i(D_{i-1}) \end{aligned} \tag{2.8}$$

and $K_i = PC_2(C_iD_i)$. LS_i represents a cyclic shift (to the left) of either one or

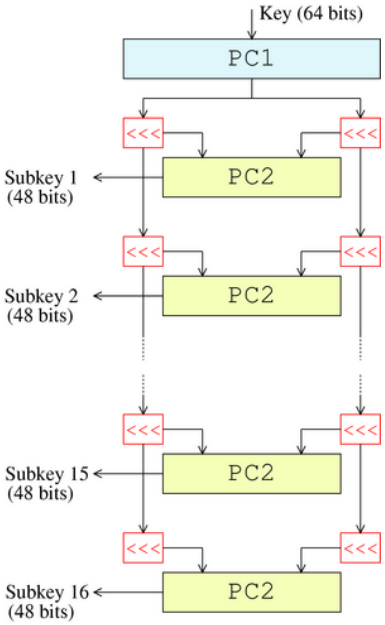
Table 2.2 – ***E function.*** *The first three bits of $E(R)$ are the bits of R in positions 32, 1 and 2 respectively, while the last 2 bits of $E(R)$ are the bits of R in positions 32 and 1 respectively.*

E bit-selection table

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

two positions, depending on the value of i: shift one position if $i = 1$, 2, 9 or 16, and shift two positions otherwise. PC_2 is another fixed permutation.

Figure 2.6 – ***Calculation of*** $f(R, K)$. *Source: Wikimedia Commons.*



To resume the DES operations, in order to execute DES, Celator must perform the

xor and *shift* operation, the E expansion as well as the following permutations: IP (Table 2.3), IP^{-1}, P, PC_1, PC_2. All expansion and permutation tables are given in annexes C.1.

Table 2.3 – ***Initial Permutation.*** *The permuted input has bit 58 of the input as its first bit, bit 50 as its second bit, and so on to bit 7 as its last bit.*

IP							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Unlike AES transformations, DES permutations are bitwise permutations and thus their execution in Celator requires more time than a byte permutation. Their implementation in Celator, which work on 32-bit wide words, shows how Celator can also perform bit format data operations.

2.3 The SHA

The SHA-1 and the SHA-2 (i.e. SHA-256, SHA-384, and SHA-512) are four secure hash algorithms specified by [3]. These algorithms are iterative, one-way hash functions that can process a message to produce a condensed representation called a *message digest*.

Each algorithm can be described in two steps: *preprocessing* and *hash computation*. Preprocessing involves padding a message, parsing the padded message into m-bit blocks (m= 512 for SHA-1 and SHA-256, m= 1024 for SHA-384 and SHA-512), and setting initialization values to be used in the hash computation. The hash computation generates a message schedule from the padded message and uses that schedule, along

with functions, constants, and word operations to iteratively generate a series of hash values. The final hash value generated by the hash computation is used to determine the message digest. In this work, we focus on the SHA-256.

From [3], eight 32-bit intermediate variables (a, b, c, ..., g, h) are required by SHA. These variables are initialized by eight 32-bit constants given by the standard $(H_1, H_2, \ldots, H_8)$.

The required steps to apply the SHA to a 512-bit message are the following ones:

- for $j = 16 \ \ to \ \ 63$
  - $W_j = \sigma_1(W_{j-2}) + W_{j-7} + \sigma_0(W_{j-15}) + W_{j-16}$
- for $j = 0 \ldots \ 63$
  - $h \Leftarrow g$
  - $g \Leftarrow f$
  - $f \Leftarrow e$
  - $e \Leftarrow d + T_1$
  - $d \Leftarrow c$
  - $c \Leftarrow b$
  - $b \Leftarrow a$
  - $a \Leftarrow T_1 + T_2$
  - $T_1 \Leftarrow h + \Sigma_1(e) + Ch(e, f, g) + K_j + W_j$
  - $T_2 \Leftarrow \Sigma_0(a) + Maj(a, b, c)$
  - $Ch \Leftarrow (a \ and \ b) \ xor \ ((not \ a) \ and \ c)$
  - $Maj \Leftarrow (a \ and \ b) \ xor \ (a \ and \ c) \ xor \ (b \ and \ c)$
  - $\Sigma_0(x) = ROTR^2(x) \quad xor \quad ROTR^{13}(x) \quad xor \quad ROTR^{22}(x)$
  - $\Sigma_1(x) = ROTR^6(x) \quad xor \quad ROTR^{11}(x) \quad xor \quad ROTR^{25}(x)$

- $\sigma_0(x) = ROTR^7(x) \quad xor \quad ROTR^{18}(x) \quad xor \quad SHR^3(x)$
- $\sigma_1(x) = ROTR^{17}(x) \quad xor \quad ROTR^{19}(x) \quad xor \quad SHR^{10}(x)$

- The result is $H_N = (H_{n1}, H_{n2}, \ldots, H_{n8})$, i.e. :
  - $H_1 \Leftarrow a + H_1$
  - $H_2 \Leftarrow b + H_2$
  - $\ldots$
  - $H_8 \Leftarrow h + H_8$

2.4 Conclusions

Three algorithms have been presented: the AES, the DES and the SHA. Celator must be able to execute these three algorithms. The AES is a ciphertext that can encrypt and decrypt a 128-bit data block. The AES data format is the byte. The DES is a ciphertext that can encrypt and decrypt a 64-bit data block. The DES data format is the bit. The SHA is a hashing function that can hash a 512 data block, resulting in to a 256-bit digest. The SHA data format is 32-bits.

This list of already-implemented algorithms is not closed. Given the reconfigurable architecture of Celator (chapter 4), Celator can be reconfigured in order to perform more algorithms. Before detailing in chapter 5 the implementation and the execution of these algorithms in Celator, we will draw up the state of the art in chapter 3 and we will describe the Celator architecture in chapter 4.

3

Hardware and software implementations of cryptographic algorithms: state of the art

Cryptographic algorithms can be performed on General Purpose Processors (GPP), or on hardware special purpose devices, like hardwired macros. GPP usually have lower performances than hardwired macros, but GPP are 100% reconfigurable. Hardwired macros give higher performances than GPP, but unlike GPP the hardwired macros are not reconfigurable.

Next sections present several implementations of cryptographic algorithms by GPP (section 3.1) and by hardwired macros(section 3.2).

3.1 General Purpose Processors

Hereafter three reconfigurable cryptographic coprocessors, are presented. We will consider the AES as the benchmark algorithm. Some of the techniques presented in these sections can be viewed in Celator, too.

3.1.1 The NEC DRP

The systolic technology described in [28] allows to parallelize the computations and shows a way to easily reconfigure such Dynamically Reconfigurable Processor (DRP, Figure 3.1). The DRP enables switching among different encryption algorithms.

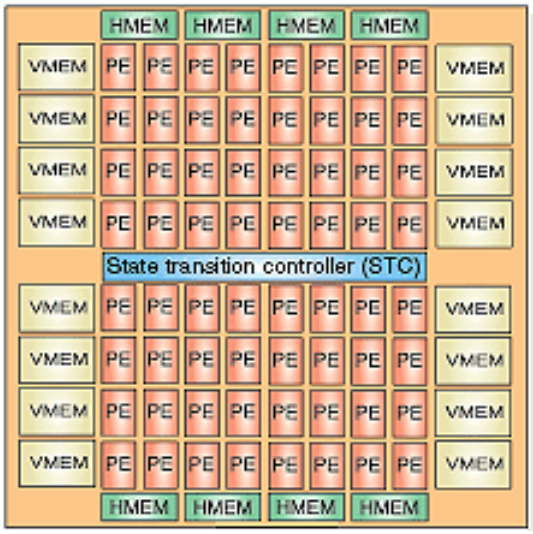
The DRP core is a two dimensional array, and it is made of:

- Several Processing Elements (PE) arranged in a systolic fashion. A PE is able to receive instructions and compute data. A PE is made of:
  - an Arithmetic and Logic Unit (ALU)
  - several registers (including a register file)
- A State Transition Controller (STC) which contains the control programs
- Vertically and Horizontally Memory units (VMEM and HMEM)

Before performing a computation, all data paths and control programs have to be saved. In order to save them, a C-language code is used: first it has to be compiled, then synthesized and downloaded into the DPR.

This systolic architecture seems to be well suited to parallel compute matrix format data. In Celator we will also see that several processing elements work simultaneously to parallelize some operations. Nevertheless in Celator, contrarily to the DRP architecture, a dedicated instruction set is defined, in order to avoid the timing loss of the C compilation phase.

Figure 3.1 – ***The DRP architecture.*** *The DRP is made of several Processing Elements, a State Transition Controller and several memory blocks.*



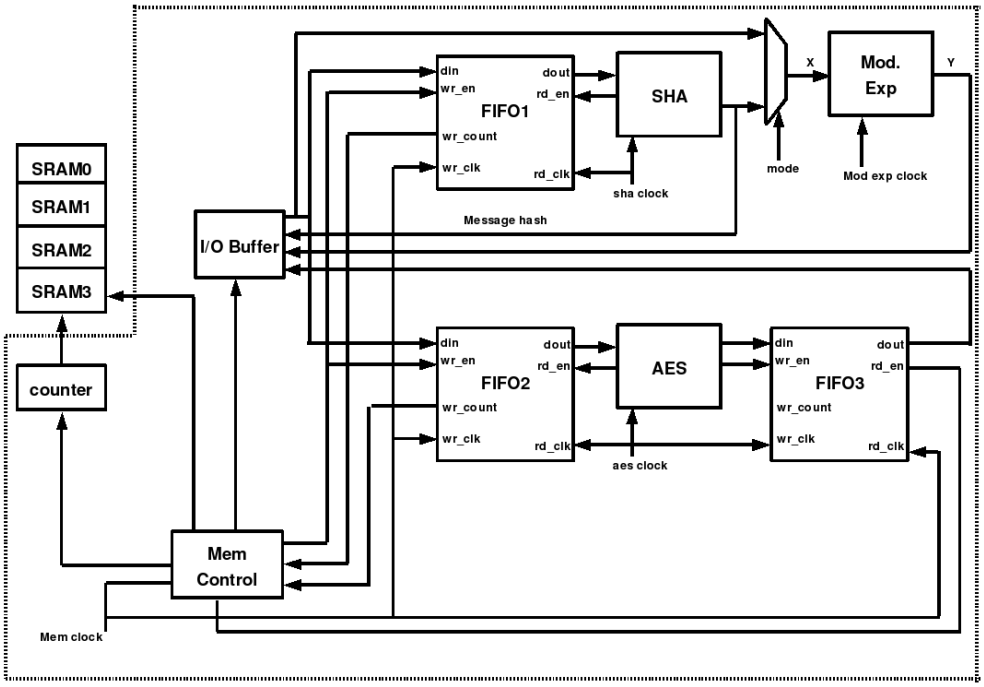
3.1.2 The Crow FPGA Implementation

The flexibility and high speed capability of FPGAs make the hardware accelerators a suitable platform for cryptographic applications, as it is shown in [29]. Crow *et al.* built a Field Programmable Gate Array (FPGA) coprocessor to perform the AES in Cipher Block Chaining (CBC) mode [30], the SHA-512 [3] and a Modular Exponentiation (ME).

The FPGA cryptographic co-processor includes a First-In-First-Out (FIFO) memory block, a Static RAM (SRAM), a memory control block, an AES block, a SHA block and a ME block (Figure 3.2). Data are stored in the FIFO. The SHA input data are stored in FIFO1, the AES input data are stored in FIFO2. The AES output data are stored in FIFO3, which is linked to the FIFO1 via an I/O buffer. In this way, after encrypting data by the AES module, data can be signed by the SHA module. The memory control block manages data transfers and provide commands to the AES, SHA and ME blocks which perform the crypto operations.

This co-processor was tested on Xilinx VirtexII-pro FPGA. The SHA is achieved

Figure 3.2 – ***An FPGA cryptographic co-processor***



with a throughput of 570 Mbps when operated at 20 MHz; the SHA block requires 2304 CLB slices. The AES execution is performed with a throughput of 468 Mbps when operated at 40 MHz (there is a multi-clock environment); the AES block requires 2688 CLB slices. For the ME block, the throughput is $\frac{51.84}{\#Expon.bits}$ Mbps when operated at 50 MHz; the ME block requires 8064 CLB slices large.

The performances of the combined architecture are impressive. Nevertheless, its implementation requires a big amount of CLB slices.

3.1.3 The Zippy Project

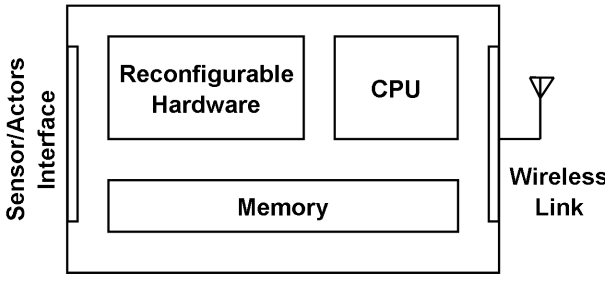
Enzler *et al.* present the Zippy design methodology in [31]. Zippy's mission is to develop reconfigurable processor for the domain of handled and wearable computing. Figure 3.3 shows the body computing system. Enzler *et al.* built a so called autonomous Wearable Unit with Reconfigurable Modules (WURM). Each WURM node consists of a CPU, a FPGA based reconfigurable hardware unit, memory, a set of I/O interfaces

that connect sensors and actors, and a wireless interface for communications with other WURM nodes (Figure 3.4).

Figure 3.3 – ***The body area computing system for wearable computers***

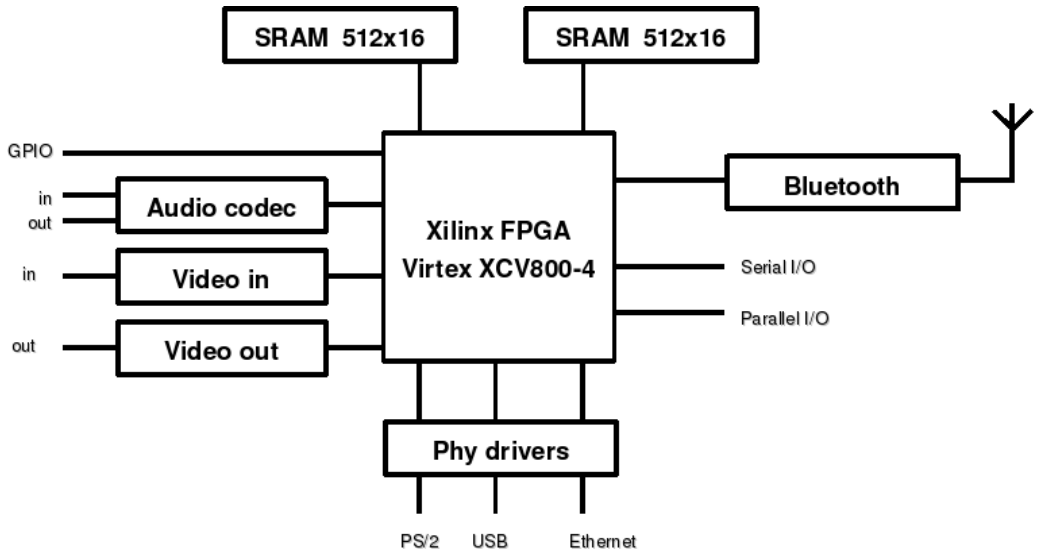
While the CPU is configured into the FPGA at power-on, the hardware tasks are dynamically configured on demand. For instance, when the WURM receives a compressed audio stream for playback, the CPU recognizes the encoding format in use and initiates the dynamic configuration of suitable audio decoding coprocessor into the reconfigurable hardware unit. When the audio coprocessor is in place, it receives the encoded audio stream from the CPU. The audio coprocessor decodes the audio stream and sends the raw audio data for playback to the digital-to-analog converter.

Figure 3.4 – ***WURM hardware architecture***



Results show how the reconfigurable hardware unit computes runtime intensive functions more efficiently than the CPU. Runtime functions are found in applications from cryptography, multimedia and communication. Enzler *et al.* did not report WURM's performances and size for encrypting or decrypting functions. Nevertheless, they state that the WURM's soft CPU core requires 3865 Virtex slices, which is 41% of the FPGA resources, and 14 dedicated block RAM, which equals 50% of the FPGA memory resources. The CPU runs at 25MHz (Figure 3.5 shows the overhead of the WURM).

Figure 3.5 – ***Block diagram of the WURM prototyping platform***



In our opinion the WURM solution can require too many area resources for embedded systems. Moreover, as we said in the previous section, the FPGA based solutions require many FPGA resources for connections. Therefore, our reconfigurable crypto co-processor will not include any FPGA technology, even if one of our goals is to implement a multi-algorithm circuit.

3.2 Hardwired macros

Hereafter eight cryptographic hardwired macros are detailed. These macros are dedicated to AES, DES and SHA.

3.2.1 The Sharma macro

In [32] a systolic architecture is used for constructing high-speed AES special-purpose devices, which includes a computer memory and a processing element array. Data flow from the computer memory in a rhythmic fashion, passing through several processing elements before it returns to memory. The architecture has improved the hardware complexity and the rate of encryption/decryption. Similarities of encryption and decryption are used to provide a high level of performance while keeping the chip size small.

The four AES transformations are achieved in the following way:

1. The Sub-Bytes transformation is performed by ASIC AES boxes (Figure 3.6). With few connections, the same Galois Field (2^8) inverter can be used both in encryption and decryption.

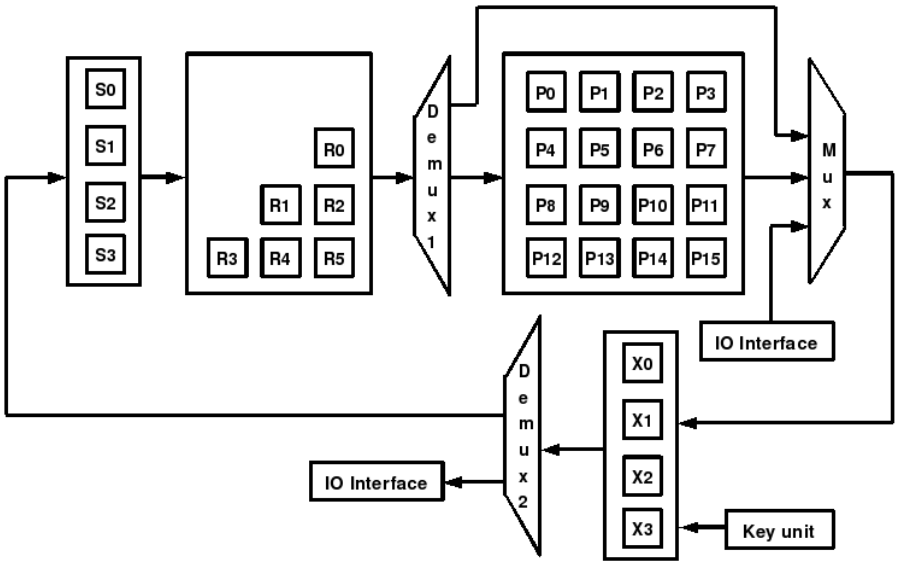
2. In Shift-Rows transformation no separate design have been implemented. Data are shifted by using registers R0–R5.

3. In Mix-Columns transformation systolic architecture is used for matrix-matrix multiplication of the 4x4 array. In each processing element a combinational logic is used for multiplication in GF(2^8) and for addition. Moreover, each cell has one register which contains the values required by the encryption and decryption process and one register is used to store the computation result.

4. In Add-Round-Key transformation, a combination of XOR gates is used.

The implementation of this hardware systolic architecture requires 40 clock cycles to perform an AES encryption for 128-bits, with a theoretical throughput of 3.2 bit

Figure 3.6 – ***Design for AES Data Unit***



per cycle (the authors did not provide the test clock frequency). Considering that an AES round includes 4 transformations and that a round is repeated 10 times for an encryption, each transformation is performed in one clock cycle only. This is a high performance result.

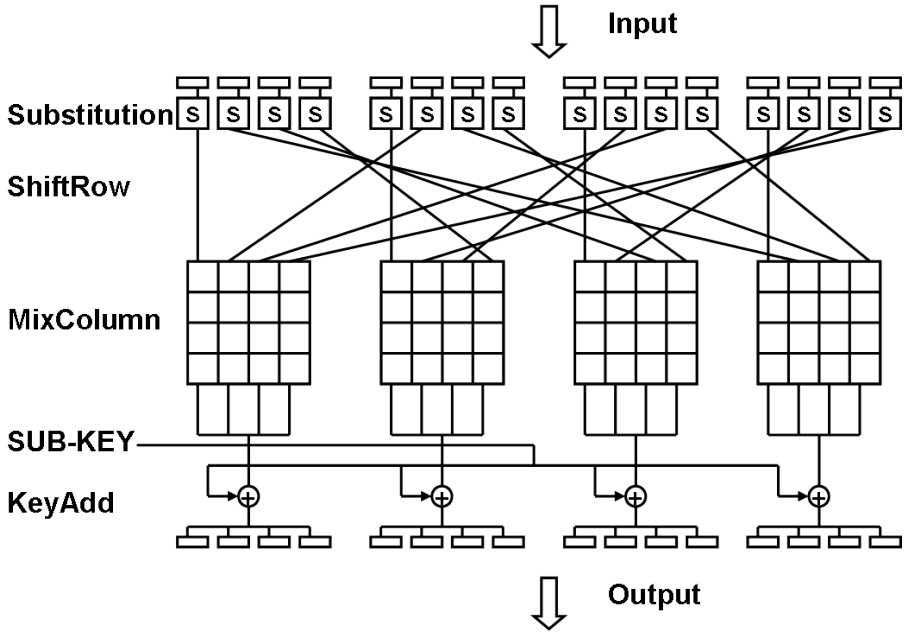
The drawback of these macros is the **non reconfigurability**: they are able to perform an algorithm only, i.e. the AES. Nevertheless, their systolic architecture seems to be a good idea to encrypt/decrypt matrix format data. Our solution shall improve this idea in order to obtain a reprogrammable and multi-algorithms integrated circuit to compute matrix format data.

3.2.2 The G-Plus AES implementation

In [33] a special purpose architecture which performs the whole AES encryption in 1 clock cycle only is patented. Its implementation requires a maximum parallel encryption module (Figure 3.7).

High throughput is achieved, even when the AES algorithm is employed with one of the feedback operation modes [30]. Hardware is provided for one encryption round,

Figure 3.7 – *The G-plus AES implementation*



which is re-used as needed to complete the encryption process. This allows feedback modes to be used without adversely affecting AES throughput.

The controller is part of a hierarchical distributed control scheme comprising some Finite State Machines (FSMs). It controls the operation of the encryption and key scheduling modules such that one round of the AES algorithm is completed per clock cycle.

The main advantage of this solution is the possibility to encrypt/decrypt on the fly, which is useful in products dedicated to the multimedia market. A drawback of this solution is the big amount of logic gates, which can be difficult to implement in embedded systems. Furthermore, as in the device described previously [32], this circuit is not reconfigurable.

3.2.3 The Trichina Coprocessor

In [23] an immune to first order DPA AES coprocessor is described. Masking all input and intermediate data values appears to be useful to decorrelate any information leaked

through Side Channels Attacks (SCA). This countermeasure is one of the most powerful countermeasures against SCA [34], [35].

The data masking idea is simple: the message and the key are masked with some random values at the beginning of the computation process, and thereafter everything is almost as usual. Of course, the value of the mask at the end of some fixed steps must be known in oder to re-establish the expected data value at the end of the execution. A traditional XOR operation is used for data masking. The operation is compatible with the AES structure except for the inversion in Sub-Bytes, which is the only non-linear transformation. Trichina *et al.* show how to manipulate masked data. They build a masked multiplier in GF(2^n) from standard multipliers, and they achieve the Sub-Bytes transformation by a Masked S-box.

Results show that this architecture is able to secure encrypt an AES data block with a throughput of 4 Mbps when operated at 5 MHz. The total gate count for the circuit is 16K. By a simple multiplication, we can estimate that the encryption is achieved in 16 rounds only. In our solution, we shall adopt the masking data as one possible countermeasure.

3.2.4 The Eli Biham DES implementation

In [36] a DES implementation is presented by Eli Biham. Its solution uses a 64-bit processors as SIMD parallel computer which can compute a 64-bit operations simultaneously. It achieves speeds of about 137 Mbps to encrypt a 64-bit data block. The DES operations are executed as follows:

- XOR operation: the XOR operation of the processor computes 64 one-bit XORs;
- expansion and permutations: these operations do not cost any operation, since instead of changing the order of words, the required word can be addressed directly by only changing the naming of the registers;
- S-box operation: the S-boxes are computed by their logical gate circuit, using

XOR, AND, OR and NOT operations. Typically they are represented in about 100 gates.

This implementation is attractive to ciphers that manipulate bit format data block, whose operations are simple (no multiplication for example, except the trivial one with factor 2), use only small S-boxes (thus their gate complexity is small), or use small register sizes. With difficulty Celator can be a Single Instruction Multiple Data (SIMD) processor. A Multiple Instruction, Multiple Data (MIMD) processor would be more suitable to ensure many possible computations. The dedicated S-boxes seem to be a good solution to speed up these substitutions. Unfortunately, dedicated S-boxes can be exploited by one algorithm only. Thus, we will try to develop a generic look up table for every kind of substitutions in Celator.

3.2.5 The Saqib implementation of DES

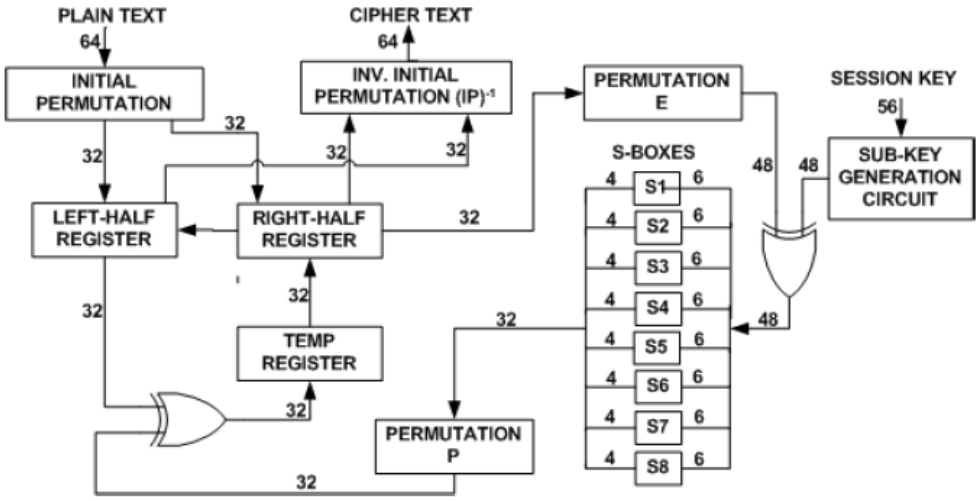
In [37] an efficient and compact DES architecture especially designed for reconfigurable hardware platforms is detailed. This DES implementation makes use of an eight DES S-Boxes parallel structure (Figure 3.8), resulting in a significant reduction of the critical path for encryption/decryption. Its DES round design achieves a data encryption/decryption rate of 274 Mbps.

DES makes use of 8 S-Boxes (each of 64x4) occupying a total of 2 Kbits. This relatively small amount of memory is implemented by using the distributed memory resources in FPGAs. Fixed permutations in fact occupy a low percentage of FPGA interconnection lines because they are hardwired.

FPGA implementation of DES algorithm was accomplished on a VirtexE device XCV400e-8-bg560. The design achieves a frequency of 68.05 MHz. It takes 16 clock cycles to encrypt one data block (64-bits). Therefore, the achieved throughput is 274 Mbps.

Reconfigurable devices are attractive since the time and costs of VLSI design and manufacturing can be reduced. The S-Boxes can be stored in a dedicated memory,

Figure 3.8 – ***DES Algorithm implemented on a FPGA***



but including a memory per S-box is expensive in terms of area. We will see how in Celator two look up tables can be accessed concurrently by using a double port memory to store the S-boxes, and how the execution time can be saved.

3.2.6 The Ahmad hardware implementation of SHA

In [38] an architecture to implement the hash algorithms of the secure hash standards SHA-256 on Altera FPGA is explored. A ROM is used to store the SHA initial hash value. A high throughput is achieved 335 Mbps. Multi-operand adders are implemented by carry save adders (CSAs), referred to as redundant adders but which are faster and have a smaller area than Carry look-ahead Adders (CLAs).

3.2.7 The Chavez hardware implementations of SHA

The hardware implementations of SHA-256 can be improved by the techniques presented in [39]. The most relevant techniques are:

- parallel counters or well balanced Carry Save Adders, in order to improve the partial additions;

- techniques (see below the example of the SHA variable computation) that optimize the data dependency, and parallelize operations;
- improved addition units;
- embedded memories to store the required constant variables.

In this way, the SHA is executed in 65 cycles (i.e. one cycle per SHA round) with a throughput of 1.4 Gbps.

The usage of SHA, CSA would not be so beneficial in Celator, because among the PE (its reconfigurable devices) there is a dedicated connection paths for addition. Embedded memories can speed up all data transfers between registers and we shall use this technique.

Parallelizing computations is difficult in the SHA-256, because the data dependency is very complex. Nonetheless, SHA algorithm can be divided into two parts: the variables B, C, D, F, G, and H (section 5.3.1) are obtained directly from the variables of the round, without any computation, while the variables A and E require some computations, and depend on all other SHA variables. Thus the computations of all SHA variables can be split in two stages: a stage to get the values of B, C, D, F, G, and H, and a stage to get the values of A and E.

We will see how Celator split the computations of the SHA variables in two stages, too.

3.2.8 The Cadence Hashing Algorithm Generator SHA-256

Cadence implemented an hardware SHA-256 [40] suitable for use in random number generators. High speed operation with throughput at low gate count were achieved:

- 133 MHz operation at 0.18 μm CMOS technology
- 2100 NAND-2 equivalent ports
- 971 Mbps throughput

Hardwired macros like this has the advantage of a high performances, but is not reconfigurable. Unlike Celator shall be.

3.3 Conclusions

We have seen the hardware and software implementations of AES, DES and SHA algorithms. We can resume the following statements:

- The systolic architectures seem to be well suited for parallel compute matrix format data [28].
- The performances of the combined architecture are impressive. Nevertheless, its implementation requires a big amount of CLB slices [29].
- FPGA based solutions require many FPGA resources for connections [31].
- The hardwired macros are not reconfigurable.
- The hardwired macros can achieve a high bit-rate, and they can encrypt/decrypt on the fly, which is useful in products dedicated to the multimedia market [33].
- Masking data can be an useful countermeasure against SCA [23].
- The dedicated S-boxes speed up a software implementation [36].
- The time and costs of VLSI design and fabrication can be reduced on reconfigurable devices [37].
- In SHA the data dependency is very complex, therefore parallelizing computations is difficult [39].

The reconfigurability grows up from a hardwired implementation (0% reconfigurable) to an implementation on GPP (100% reconfigurable), while the throughput and the area decrease. For us, the good trade off choice is a solution “in the middle”.

In this way, the developed Celator has a x degree of reconfigurability comprised between 0% and 100%, faster than GPP and smaller than hardwired macros. In next chapter the Celator architecture is disclosed.

4

Proposing a reconfigurable cryptographic coprocessor: Celator

This chapter describes the hardware implementation of Celator processor.

Our challenge is to make Celator reconfigurable. If you do a Google search for "reconfigurable processor" you find more than 40000 results including FPGA technologies (Avril 2008). But FPGA based solutions are not adequate to implement Celator because:

- too many FPGA resources are needed for connections; the embedded FPGA can be another solution, and this technique is currently under study
- too many time is lost in *compilation time* when the FPGA cards are reconfigured; the dynamically configurable FPGA can be another solution, but again, to many area is required to interface the FPGA card with other devices.

Therefore we aimed to find a reconfigurable technology without FPGA cards. We found the solution in a systolic processor based on a Proposing Element array. In Celator, the data-path of the PE array can be reconfigured at each clock cycle. Therefore Celator is a reconfigurable processor.

The AES operates on 8-bit data word, while the DES operates on 1-bit data word, and the SHA operates on 32-bit data word. In this way we show that Celator can efficiently operate on 8-bit, 1-bit and 32-bit data words.

Three cryptographic algorithms have been implemented on Celator. Therefore Celator is also a cryptographic multi-algorithm processor.

Section 4.1 shows the computer system that Celator is conceived for. Celator architecture is detailed in section 4.2. Finally, in section 4.3 some considerations are given about the choices we made in Celator hardware development.

4.1 The system: CPU, Memory, peripherals, bus

Celator is designed to be integrated in a computer system which includes at least the following devices:

- a Main processor or CPU
- a RAM
- Celator
- an Interface (IF) unit
- Other peripherals (i.e. Parallel Input/Output (PIO), buses, Direct Memory Access (DMA) controller, net-card, etc..)

A Smart Card or a Card Reader are an example of such a computer system (Figure 4.1). In the validation system the CPU is the ARM 7 TDMI core (Figure 4.2, [41]). The CPU will be the master of both the bus and the Main Memory.

There are 2 buses: the Advanced High-performance Bus (AHB) and the Advanced Peripheral Bus (APB). The first works at a higher frequency than the second one. Celator will be linked to the CPU and to the RAM via the AHB and via the IF unit.

Figure 4.1 – ***Smart Card architecture.*** *Celator can be included in a SC*

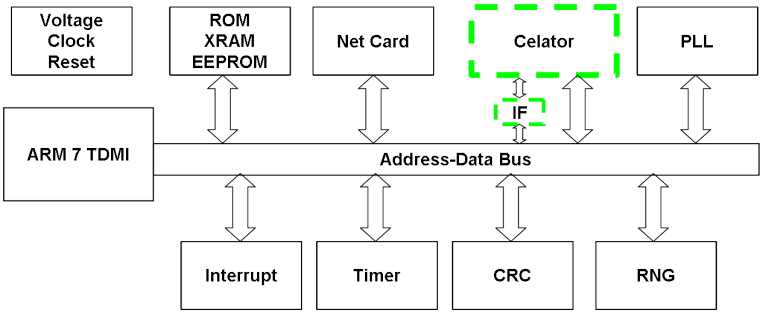
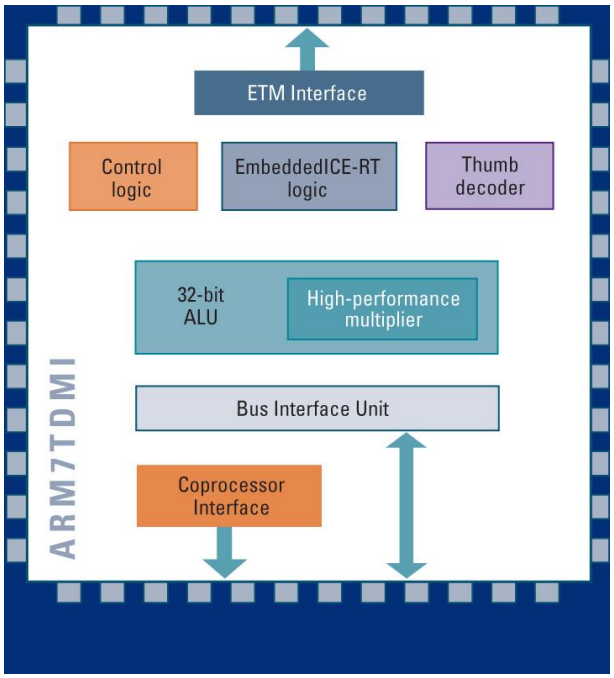


Figure 4.2 – ***The ARM 7 TDMI architecture***

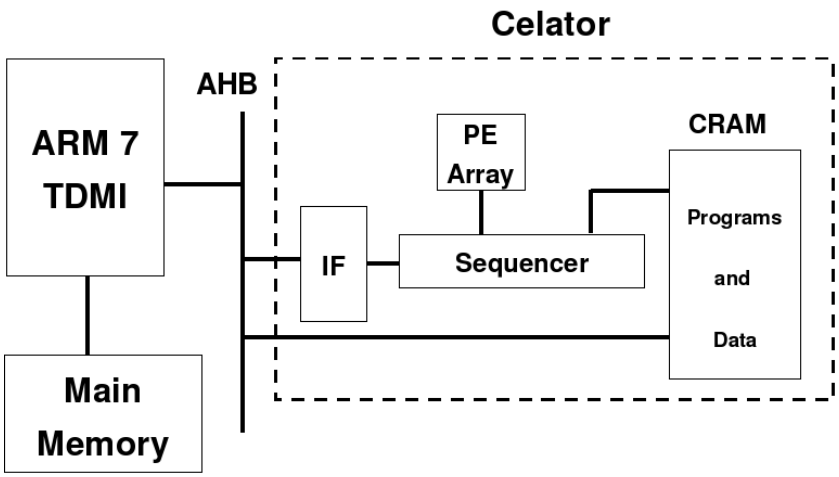


The IF unit manages the AHB (section 4.2.5) in order to let the CPU communicate to Celator, and vice-versa. For instance, the CPU can program Celator to encrypt a data block. When the encryption is achieved, Celator communicates the results to the CPU.

4.2 Celator hardware architecture

This section details the hardware development of Celator and the architecture of the whole computer system around Celator. The whole computer system (Figure 4.3) includes the ARM 7 TDMI core, a Main Memory, the IF unit, Celator and the AHB. Celator includes a PE array, a Controller and the Celator Memory called CRAM. Celator can have a different clock from the CPU, and the IF unit allows Celator and the CPU to communicate even when they work at a different frequency. These modules are detailed in next sections.

Figure 4.3 – ***Whole computer system architecture***

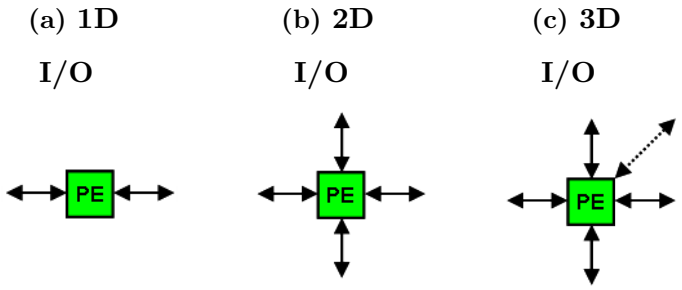


4.2.1 The Processing Element Array

The PE array is the *data path* of Celator, and consists of a systolic array of 4x4 Processing Elements. All PE are identical. The PE array is controlled by the Controller (section 4.2.3). Systolic array processors [42] can have mono-dimensional, two-dimensional or three-dimensional I/O connections. The 3D I/O connections need more area than 1D I/O. For Celator we chose the 2D I/O, which seems to be a good trade off with respect to the connection (Figure 4.4) and allows Celator to execute several algorithms.

Thus, by the 2D I/O, the whole PE array has (Figure 4.5):

Figure 4.4 – ***Three types of I/O connections***



- four 32-bit data inputs/outputs, i.e. one per cardinal direction;
- four multiplexers (MUX_N, MUX_E, MUX_W, MUX_S) which the input/output of the PE array allow to be switched among: North, East, West and South input/outputs. The multiplexers can also make PE array linked in a toric fashion, which can be useful to perform operations such as rotations around the PE array (Figures 4.5 and 4.6).

All PE array multiplexers are configured by the Controller, and the configuration can be modified at each clock cycle. Therefore, the PE array data path is reconfigurable.

4.2.2 The Processing Element – Confidential

This section is Atmel confidential.

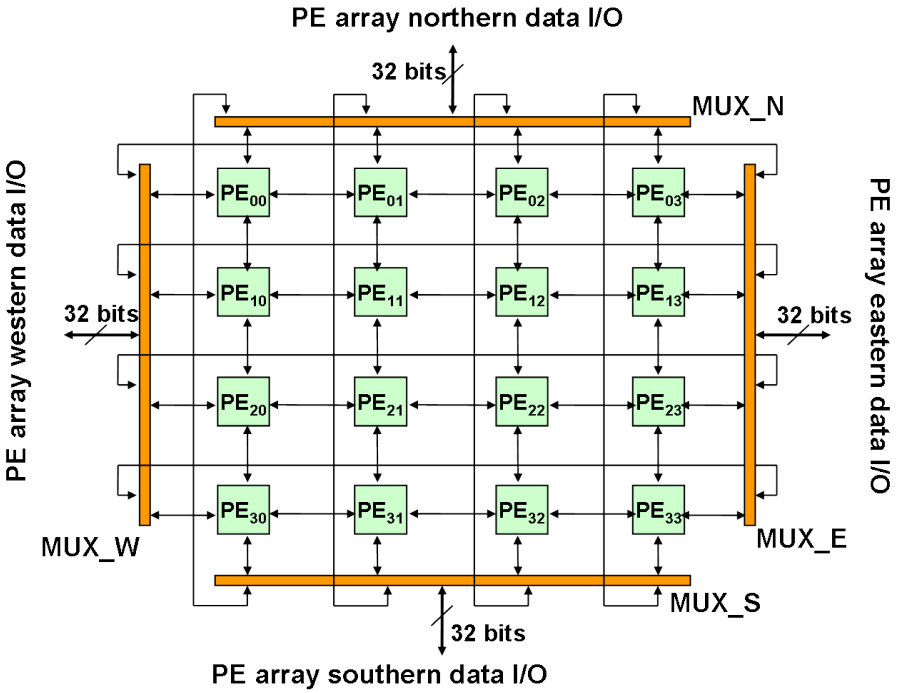
4.2.3 The Controller – Confidential

This section is Atmel confidential.

4.2.4 CRAM

The Celator RAM, or CRAM, can store 32-bit data and instruction words (Figure 4.7). Data include the encrypting or decrypting input/output data, as well as all data needed by the encrypting/decrypting process, e.g. the Sbox for the AES. Programs stored in CRAM include the micro-instructions for the Controller. The CRAM is a double port

Figure 4.5 – *The architecture of the Processing Element array*



RAM, therefore both CPU and Controller can access to the CRAM in read/write mode. The CRAM that we use is 4KB large, 32-bit word size and its area is 1 mm^2 with Atmel CMOS 130 nm technology.

4.2.5 The Interface unit

The Interface (IF) unit links the CPU and Celator via the AHB bus. It lets the CPU communicate with Celator. The AHB is made of the following buses:

HADDR : 32-bit input address bus

HWRITE : 1-bit input write enable signal

HSEL_RAM : 1-bit input RAM select signal. Celator includes a RAM (section 4.2.4), and this signal enables the writing and the reading operations into the Celator RAM.

HSEL_REG : 1-bit input register select signal. The IF includes several registers, and this signal enables the writing and reading into the IF unit registers

Figure 4.6 – ***Details of the 1st column of the PE Array***

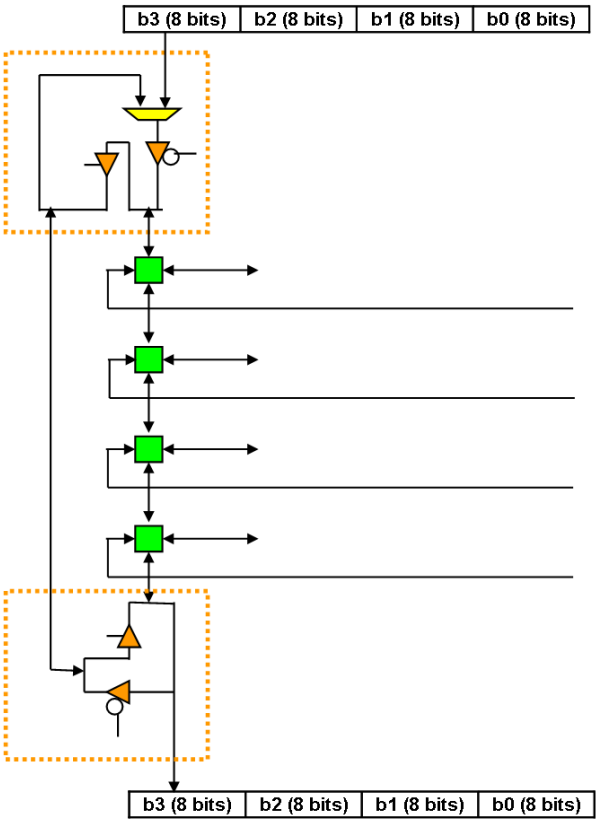
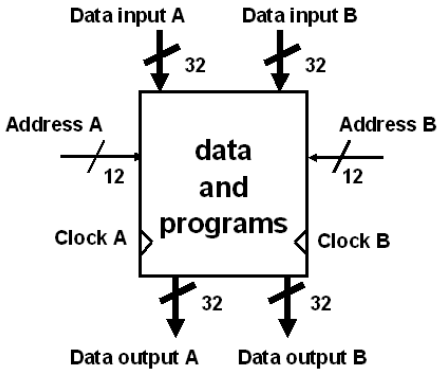


Figure 4.7 – ***The CRAM architecture.*** *It is a double port RAM.*



HWDATA : 32-bit output data bus

HRDATA : 32-bit input data bus

The IF unit includes (Figure 4.8):

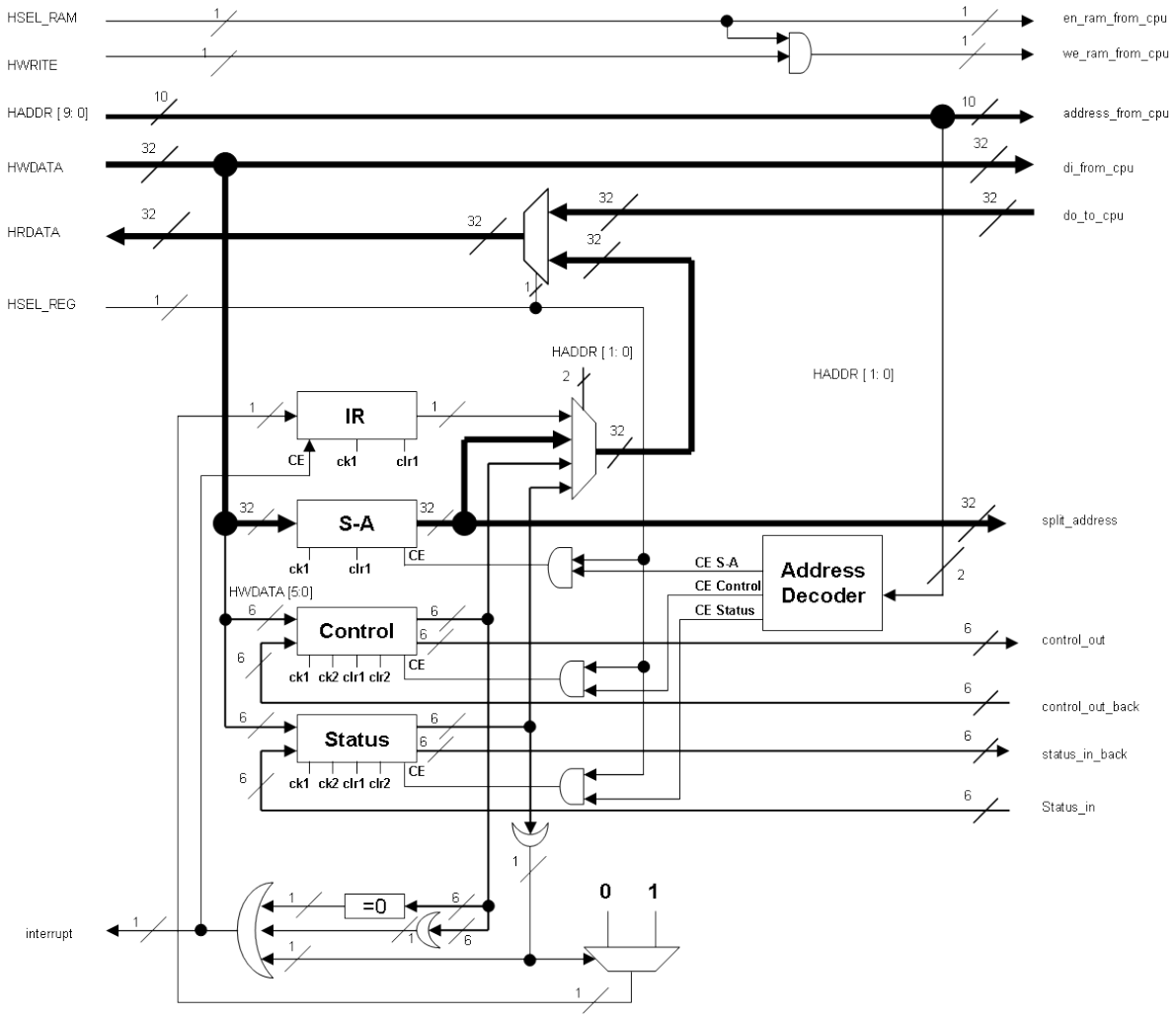
- Control register: used by the CPU to start and reset Celator (2 bits)
- Status register: used by Celator to indicate to the CPU that the current algorithm is done (1 bit)
- Interrupt register: to indicate the CPU when the current algorithm is done (1 bit)
- Split-Address (S-A) register: the S-A register can be written by the CPU only, and stores a RAM address. This address is comprised between 0 and the CRAM highest address (1023), that we call `Max_CRAM_Address`. The S-A register value is used to disable the CPU writing mode into the CRAM: the CPU may write into the CRAM addresses comprised between the S-A value and the `Max_CRAM_Address`, only. For instance, before filling the CRAM with a program, the CPU sets the S-A value to `0`. Before starting Celator, the CPU can set the S-A value to `Max_CRAM_Address - 16 bytes`. If so, the CPU may write into the last 16 bytes of the CRAM only: that memory space can be used as data buffer between Celator and the CPU. In this way, no malicious applications can write into CRAM, and modify the program (or data) stored at the address comprised between `0` and in the `Max_CRAM_Address`. The S-A size is 12 bits.

Both Control and Status registers are set-reset registers (Figure 4.9). These registers allow the communication between 2 devices (i.e. CPU and Celator), even if these devices work at different frequencies. For each bit of a such set-reset register, 4 Flip-Flops with asynchronous clear and one "or" logic port are needed.

The communication between CPU and Celator works in the following way. Let the Control register have the following structure:

- LSB (i.e. bit number "0"): reset signal
- MSB (i.e. bit number "1"): start signal

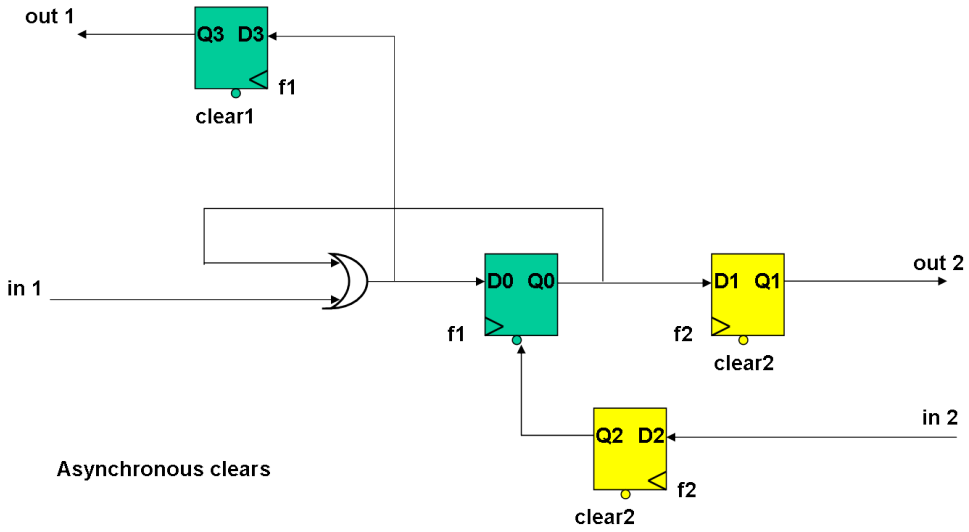
Figure 4.8 – *Interface unit.*



In order to reset Celator, the CPU has to write 01_b into the Control register. In order to start Celator, the CPU has to write 10_b into the Control register.

Consider one bit of the Control register, e.g. the start signal. Four flip-flops (FF) and one or logic gate are needed to transmit the start information. Let `in 1` and `out 1` be the start I/O with the CPU. Let f1 be the CPU frequency. Let `in 2` and `out 2` be the start I/O with Celator. Let f2 be the Celator frequency. In this way, when the CPU starts Celator, `in 1`, D0 and D3 are set to 1. After 1 clock cycle of f1, D1 is set to 1; `in 1` switches from 1 to 0, D0 and D3 hold 1. The set information (start) is

Figure 4.9 – ***Architecture of the set-reset register.***



ready to pass from f1 world to f2 world (D1 is sampled on f2). In the meanwhile, D3 is also set to 1. By now:

- after 1 clock cycle of f1, Q3 and `out 1` are set to 1. The CPU knows that Celator is receiving the start signal
- after 1 clock cycle of f2, Q1 and `out 2` are set to 1. Celator receives the start signal

Once Celator is started, Celator sets `in 2` to 1. After 1 clock cycle of f2, Q2 is set to 1, and Q0 is asynchronously reset. D1, D0, and D3 are immediately set to 0. The reset information (start) is ready to pass from f2 world to f1 world (D3 is sampled on f1) By now:

- after 1 clock cycle of f2, Q1 and `out 2` are set to 0. Celator knows that the CPU is knowing that Celator started
- after 1 clock cycle of f1, Q3 and `out 1` pass from 1 to 0. The CPU knows that Celator started

In this way, the CPU frequency modules are decoupled from Celator, which can work at a different frequency.

4.3 Considerations about Celator hardware architecture

The adopted solution Celator is based on a 4x4 Processing Element (PE) systolic array [42], which seems a good solution to compute all matrix format data.

PE array's data path is reconfigurable, just as the Finite State Machine which controls the PE array is, too. Because of these two reconfigurable elements, Celator can be reconfigured. Results show that Celator is a good trade off with respect to the execution cycles, between a dedicated hardware macro and a general purpose processor. The CRAM stores the instructions for the FSM, and the user data.

The PE array is the Celator *operating center*. It is made of 4x4 1 byte PEs, which can operate arithmetic and logic computations on 8-bit data words. In order to operate on data words larger than 8-bits, more PE must be concatenated each others.

Three algorithms have been implemented into Celator: AES [1], DES [2] and SHA [3]. In the next sections we show how Celator may be programmed in order to execute these algorithms.

5

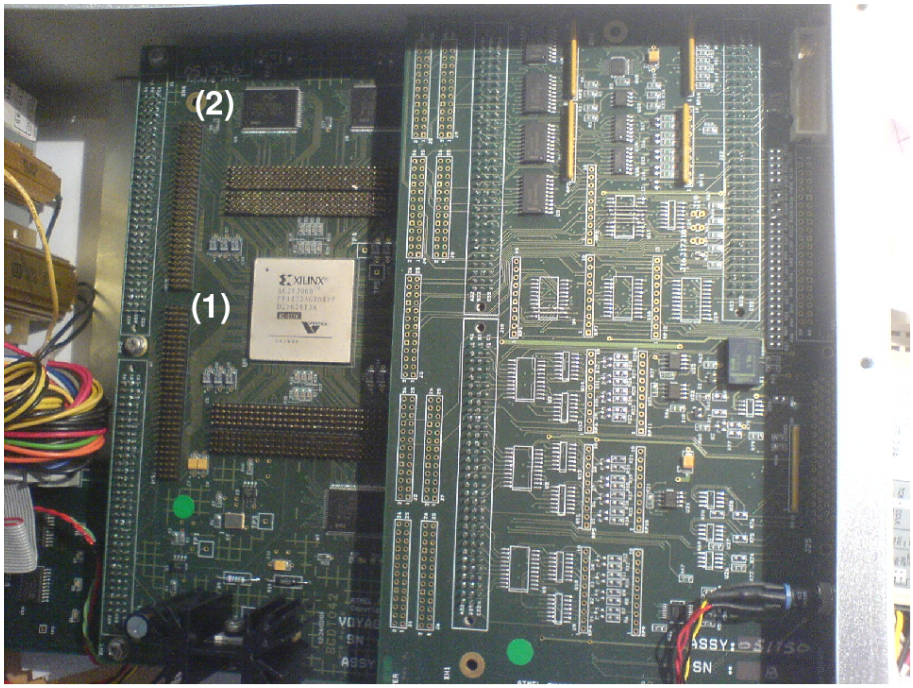
Validating Celator on FPGA

Before developing the ASIC, Celator was prototyped and validated on FPGA. In this way, Celator was developed in Verilog Hardware Description Language (Verilog HDL) at the Register Transfer Level (RTL). Therefore Celator was prototyped by Mentor Precision RTL (synthesis) and Xilinx ISE (place and route) and validated on a Xilinx Virtex XC2V3000 FPGA. We used the Atmel Voyager Platform (Figure 5.1) which include the FPGA card and an external memory, the SRAM. Simulations were executed by means of the ARM Developer Suite (ADS).

Two images were encrypted, decrypted and signed by Celator: Lena color image (Figure 5.2a) and the Penguin gray image (Figure 5.2b).

The Celator test on FPGA requires several hardware components and software applications. These components include the CPU (ARM 7 TDMI), the Interrupt Control (IC), Celator and the Interface (IF). The algorithm to be tested, for instance the AES, must be written first in Celator software instructions, and then the Celator software instructions must be coded in hexadecimal format, according to the instruction types shown in section 4.2.3. In order to automatize the assembler to hexadecimal format translation, we developed a C program called `celator_compiler`. This program is called by the following command:

Figure 5.1 – ***The Atmel Voyager Platform used to prototype Celator.*** *The Xilinx Virtex XC2V3000 FPGA (1) and an external SRAM (2) are included in Atmel Voyager Platform.*




```
/>  celator_compiler src_file dst_file
```

For the AES Celator dedicated code, see annexes B.2. For the `celator_compiler` source C code, see annexes A.1.

The FPGA hardware implementation is performed in the following way (Figure 5.3):

1. Celator, IF, CPU and IC Verilog codes (RTL) are synthesized by Mentor Precision RTL for the Xilinx Virtex XC2V3000 FPGA
2. the synthesis netlist is placed and routed by Xilinx ISE
3. a bit-file is generated by Xilinx ISE
4. the bit-file is downloaded into the Virtex XC2V3000 FPGA by Xilinx Impact software

Figure 5.2 – ***AES encrypting a colour image in ECB and CBC modes***

(a) (b)

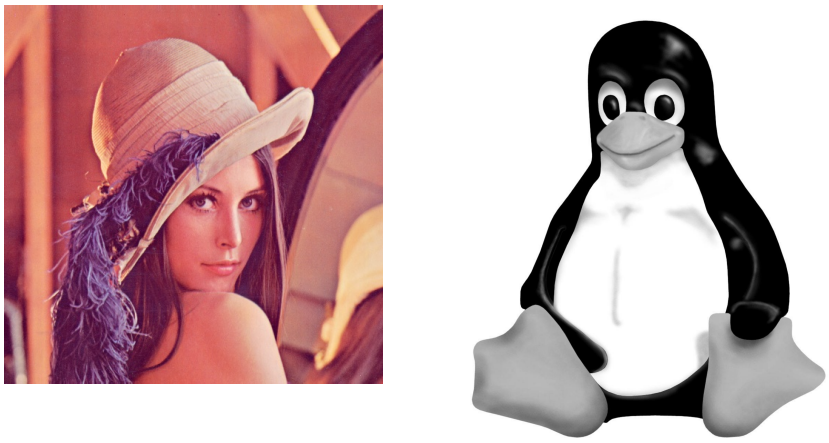
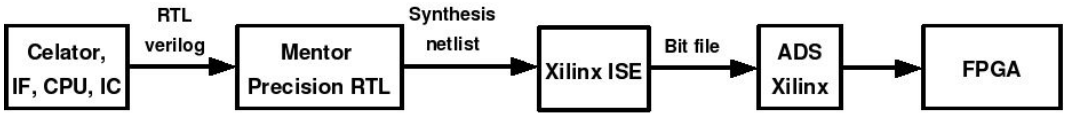


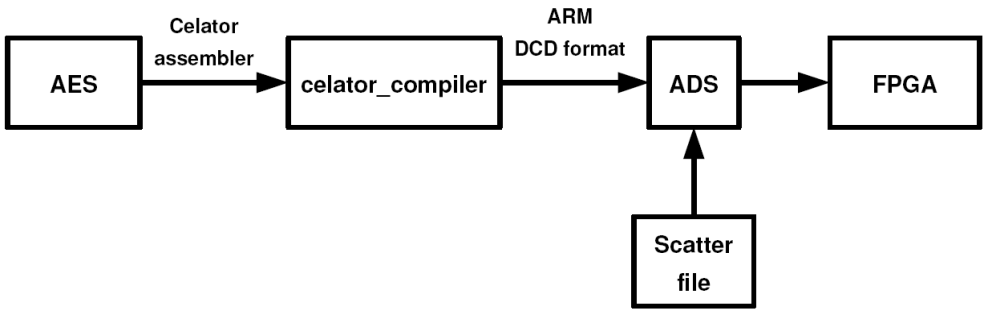
Figure 5.3 – ***The Celator FPGA implementation steps***



The Celator software programming is performed in the following way (Figure 5.4):

1. the AES is written in Celator assembler dedicated format
2. the assembler format file is translated into hexadecimal format file by `celator_compiler`. Each instruction is coded as a 32-bit word
3. each 32-bit word is written into ARM assembler format file, the ARM DCD constants
4. a scatter file is used to map the AES instructions (in DCD format) into the SRAM
5. all ARM assembler files are compiled by ADS
6. an object file is generated and downloaded into the FPGA

Figure 5.4 – ***The Celator software programming steps***



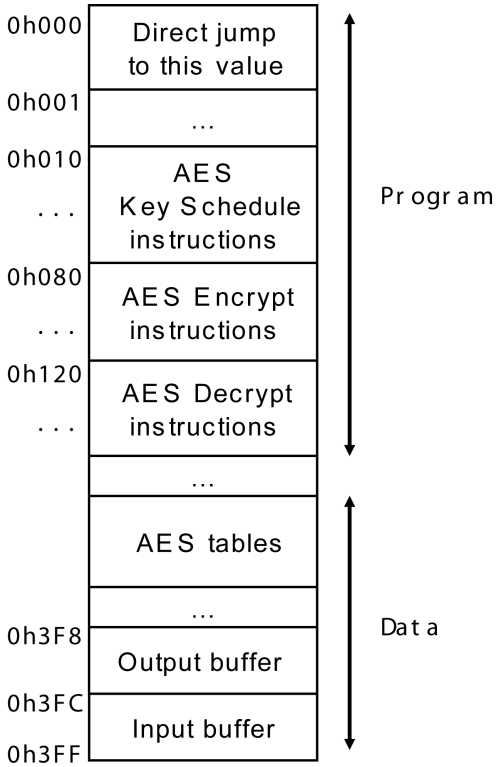
The assembler files compiled by ADS include the AES instructions in DCD format, and also other files that let Celator work, e.g. the IC handler, a vector table, a test image etcetera; for more details, see annexes B.4. Data and program words are stored by the ADS into the SRAM, at the addresses specified in the scatter file.

After the hardware and the software set up, Celator can be controlled by the IF registers. The ADS allows to write/read into all memory space. Thus, Celator can be manually reseted and started. The CPU resets Celator, fills the CRAM with AES instructions and data (Figure 5.5), then starts the AES. A software C function, called `CFunction`, was programmed to execute an algorithm on Celator (annexes B.3).

For instance, in order to encrypt (AES) an image stored in the SRAM, the `CFunction` does the following:

1. copy 128-bit data block from the image in the SRAM into the CRAM;
2. write the CRAM address of the encrypting program (e.g. `0h10`) at the address `0` of the CRAM;
3. start the AES encryption; once the AES finished;
4. copy the AES output 128-bit data block from the CRAM to the SRAM;
5. copy a new 128 bit block from the image in the SRAM into the CRAM;
6. allow to start the AES encryption, and go to 1.

Figure 5.5 – ***The CRAM.*** *In this example, the CRAM is filled with the AES instructions and data by the CPU.*

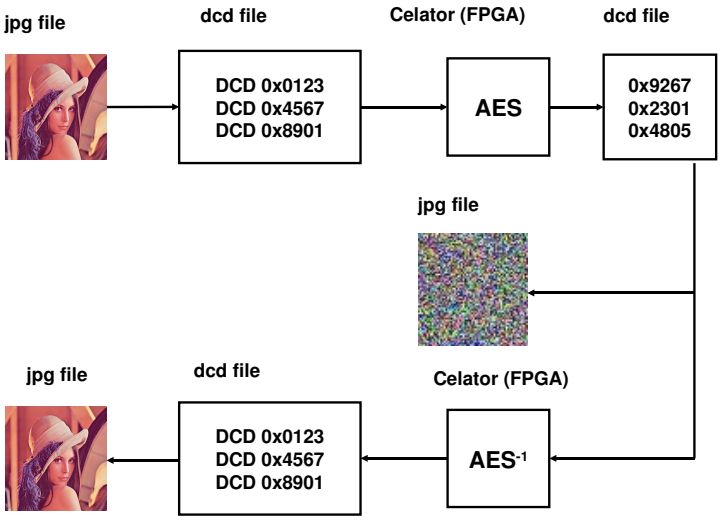


If the Celator user would decrypt a block data instead of encrypt it, he can use the same above `CFunction` by simply modifying the value that the `CFunction` stores at address `0` of the CRAM. Celator directly jumps to the address written at the address `0` of the CRAM. Thus, to execute an AES decrypting, the `CFunction` has to write the address of the AES decrypting instructions (e.g. `0h120`) at address `0` of the CRAM (Figure 5.5). To execute the AES key schedule, the `CFunction` has to write the address of the AES key schedule (e.g. `0h010`) at address `0` of the CRAM.

A whole encrypting/decrypting cycle schema is shown in Figure 5.6.

After the Celator hardware and software set up, the CPU starts Celator by writing the start signal into the IF Control register (section 4.2.5). Then Celator works autonomously. In order to compute an algorithm, the instructions stored in the CRAM are fetched by the FSM and then are executed by the PE array. When an algorithm

Figure 5.6 – ***An encrypting/decrypting cycle schema***



is achieved, Celator transfers the results into the CRAM and generates an interrupt dedicated to the CPU.

Next sections will detail how Celator can be programmed in order to perform AES (section 5.1), DES (section 5.2) and SHA (section 5.3).

5.1 AES

An AES data block is made of 128-bits. In Celator data are computed by the 4x4 PE array and each PE works on 8-bit data. Each element of the data matrix (Figure 5.7a) can be mapped into a Processing Element device (Figure 5.7b). As a consequence, each byte to be encrypted or to be decrypted is mapped into a single PE, and each AES transformation can be mapped into the 4x4 PE array.

In Figure 5.7a the data block is arranged in a 4x4 matrix, just like the 4x4 PE array does in Figure 5.7b.

Celator can perform all 4 AES transformations that are required by AES encryption, decryption and key scheduling. The Controller can execute a whole AES by combining these transformations.

Figure 5.7 – ***Data Mapping to a PE array.***

(a) The 4x4 byte data matrix for an AES data block.

d_{11}	d_{12}	d_{13}	d_{14}
d_{21}	d_{22}	d_{23}	d_{24}
d_{31}	d_{32}	d_{33}	d_{34}
d_{41}	d_{42}	d_{43}	d_{44}

(b) The 4x4 8-bit PE array of Celator.

PE_{11}	PE_{12}	PE_{13}	PE_{14}
PE_{21}	PE_{22}	PE_{23}	PE_{24}
PE_{31}	PE_{32}	PE_{33}	PE_{34}
PE_{41}	PE_{42}	PE_{43}	PE_{44}

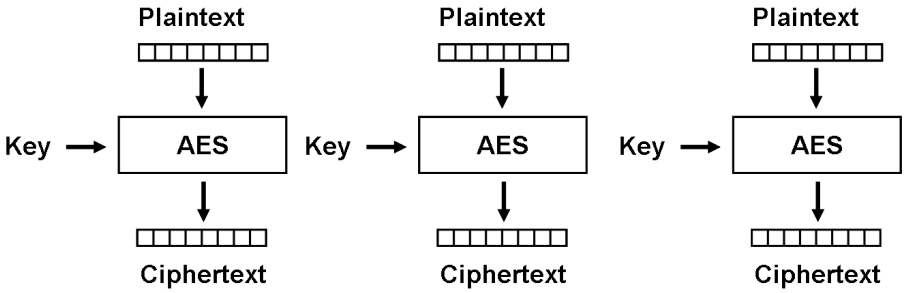
5.1.1 Implementation of the AES into a PE Array – Confidential

This section is Atmel confidential.

5.1.2 FPGA results

Celator is able to encrypt and decrypt an AES data block both in Electronic Codebook (ECB, Figure 5.8) mode and in Cipher Block Chaining (CBC, Figure 5.9) mode [30]. The Celator dedicate assembler code for AES is annexed (section B). Figures 5.10 show the difference between encrypting a gray scale image (Figure 5.10a) in ECB (Figure 5.10b) and CBC (Figure 5.10c) modes.

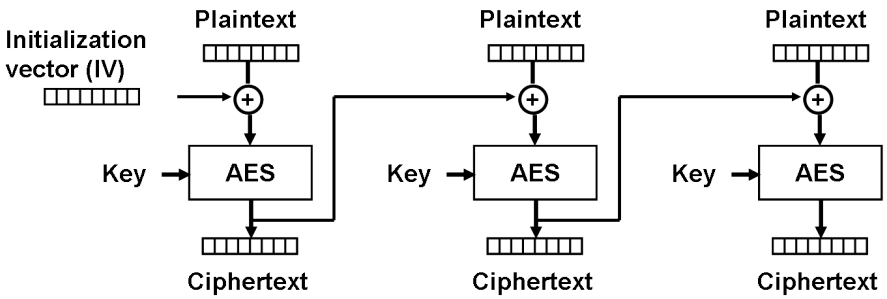
Figure 5.8 – ***Electronic Codebook (ECB) encryption mode***



In ECB mode, the plain image is first cut in several 128-bit data blocks separately, and

then the blocks are independently encrypted. If the input data blocks of the encrypting process have the same values, i.e. if there are some repeated pixels in the image, then the output data blocks of the encrypting process will be the same, i.e. several pixels will have the same values, since the keys are the same for all data blocks. That is why we can identify the *penguin's* profile in Figure 5.10b.

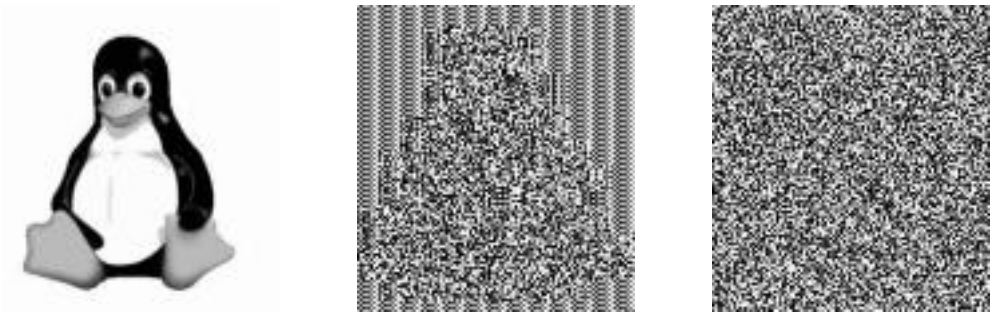
Figure 5.9 – ***Cipher Block Chaining (CBC) encryption mode***



In CBC mode, the plain image is first cut in several 128-bit data blocks and, unlike the ECB mode, each block is xored with the previous encrypted data block, and then each block is encrypted (first data block of the picture is xored with an initialisation vector). In this way, all output data blocks are different, and the output image is unrecognizable (Figure 5.10c).

Figure 5.10 – ***AES encrypting a gray scale image in ECB and CBC modes***

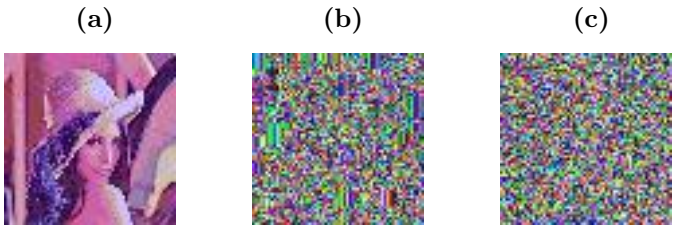
(a) **(b)** **(c)**



Again, Figures 5.11 show difference between encrypting the colour image of *Lena* in

ECB and CBC modes. Like for the penguin, *Lena* in ECB mode is recognisable, while in CBC mode the image is not.

Figure 5.11 – ***AES encrypting a colour image in ECB and CBC modes***



The images of *Lena* and of the *penguin* used in this test are smaller than the original ones (Figure 5.2a and 5.2b). They were resized because the adopted SRAM was size limited, and only images up to 12 MB size could be loaded.

5.1.3 ASIC results

Celator performances for the AES algorithm are given thereafter and compared with the performances of other cryptographic structures. We chose to compare Celator with GPPs and AES dedicated hardwired macros. The GPPs are ARM7TDMI (32-bits) [41], ARM9 (32-bits) [43] and AVR (8-bits). The AES dedicated hardware macros are the Atmel SOMA AES (AT91S0100 [44]), the Helion Standard AES cores [45], and the architectures presented in [32], [46], [47]. Moreover, Celator is also compared with a dynamically reconfigurable architecture called PiCoGa [48].

The whole computer system presented in Figure 4.3, i.e. Celator, the ARM7TDMI and some other peripherals, were synthesized by Synopsys Design Compiler with Atmel Standard Cells resources and the 130 nm technology. These same tools were also used to determine the ASIC areas of SOMA and of AVR microprocessors. Results are shown and commented below.

The version of Celator presented here needs 570 clock cycles and performs at 43 Mbps to encrypt an AES 128-bit data block in ECB mode. In CBC mode an AES 128-bit data block is encrypted in 580 clock cycles and performed at 42 Mbps.

Even though a dedicated hardware macro is faster than our solution (the AES hardware macro of Atmel SOMA takes 40 cycles, i.e. a cycle for each AES transformation), it needs a larger area. GPPs such as the ARM7TDMI and ARM9 processors are larger and slower than Celator. The GPP AVR has the same size but it is slower than Celator. Comparisons between Celator and hardwired macros and GPP are given in Table 5.1. This table does not show the size of RAM and caches. The CRAM area is $1mm^2$ in $130nm$ Atmel technology.

Table 5.1 – ***Comparisons of areas and performances for encrypting an AES 128-bit data block in CBC mode.*** *Note that RAM and Cache sizes are not taken into account.*

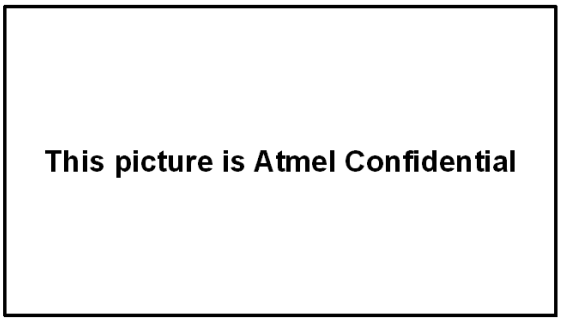
	cycles (# of)	**Max Freq.** (MHz)	**Area** (mm^2)	**Bit-rate** (Mbps)	**Techno.** (nm)
Atmel SOMA [44]	40	150	Atmel	480.0	130
Celator	580	190	Confidential	42.0	130
μ **PiCoGa**[48]	285	200	Atmel	90.0	90
μ **AVR**	5000	40	Confidential	1.0	130
μ **ARM7TDMI** [41]	3600	65	Atmel	2.3	130
μ **ARM9** [43]	830	200	Confidential	31.0	130

Dedicated hardware macros (Tables 5.1 and 5.2) such as Mangard, Satoh, Sharma and Atmel SOMA AES have a higher bit-rate (bit per second) than Celator but unlike Celator they are not reconfigurable. A dynamically reconfigurable architecture such as PiCoGa [48] achieves a higher bit-rate than Celator but, even if PiCoGa is reconfigurable, it is far larger than Celator.

Table 5.2 – ***Throughput performances for encrypting an AES 128-bit data block in ECB mode***

	Throughput (bits per cycle)
μ **AVR**	0.024
μ **ARM7TDMI** [41]	0.036
Celator	0.224
Mangard Macro [47]	2.0
Satoh Macro [46]	2.37
Sharma Macro [32]	3.2
AES Macro of Atmel SOMA [44]	3.2

Figure 5.12 – ***Areas and Throughputs for AES-128 in CBC mode encrypting of a special purpose macro (Atmel), of multi-algorithms processors (Celator and PiCoGa) and of general purpose processors (AVR and ARM).***



To conclude this result section, Celator has a good percentage of reconfigurability given by the portfolio of its instructions and proved by the implementation of other algorithms (sections 5.2 and 5.3): the PE array data-path can be modified and up to eight ALU operations can be selected. It also results clear that Celator represents a good trade off among dedicated hardware macros (which are not reconfigurable), the dynamically reconfigurable architecture as PiCoGa (which has a good percentage of reconfigurability) and GPPs (which are fully programmable) in terms of areas and throughput (Figure 5.12).

5.2 DES

Before detailing every step on how Celator can do the DES permutations, we must consider that Celator is able to compute 32-bit word data (Figure 5.13). In fact, as a single PE computes a 8-bit data (Figure 5.13a), four PE can be concatenated in order to compute a 32-bit data. In this way, the PE array can be considered as four 32-bit data rows (Figure 5.13b). Moreover, as each PE includes two 8-bit registers, the whole PE array can be considered as eight 32-bit registers, four Reg A and four Reg B (Figure 5.13c).

5.2.1 Implementation of the DES into a PE Array – Confidential

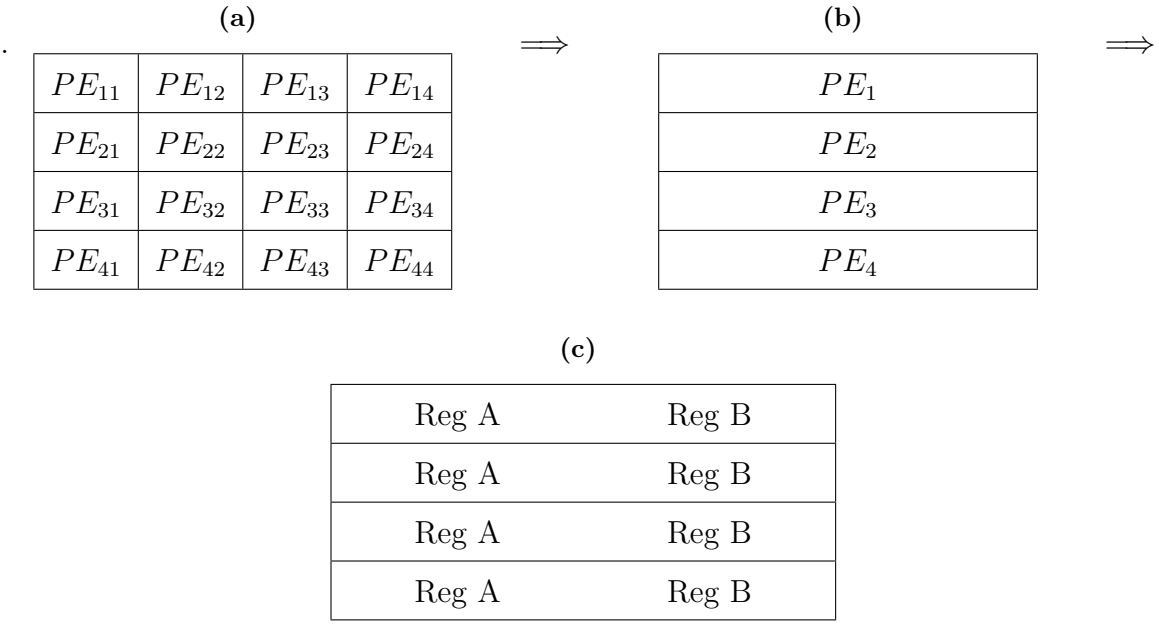
This section is Atmel confidential.

5.2.2 FPGA results

Figures 5.14 show the image of *Lena* before (Figures 5.14a) and after DES encrypting (Figures 5.14b) in ECB mode. The Celator dedicate assembler code for DES is annexed (section C). Like in the Figure 5.8, the *Lena's* profile is recognisable when encrypted in ECB mode.

The schema followed for encrypting in ECB mode is the same used in Figure 5.8, replacing the AES with the DES algorithm.

Figure 5.13 – ***Considering the PE as eight 32-bit PE rows.*** *In each PE row, four PE can be concatenated and operate on 32-bit data; as each PE include two registers (Reg A and Reg B), four PE can work on two 32-bit data*



5.2.3 ASIC results

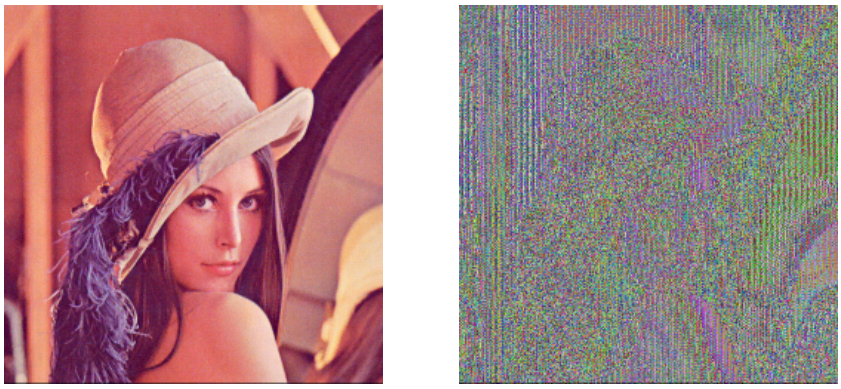
Celator performances for the DES algorithm are given thereafter and compared with the performances of other cryptographic structures (Table 5.3). We chose to compare Celator with a GPP and a DES dedicated hardwired macro. The GPP is the Alpha processor [36]. The DES dedicated hardware macro is the Atmel DES macro. Moreover, Celator is also compared with an efficient FPGA implementation [37].

Since Celator is reconfigurable, we do not need to modify its architecture in order to encrypt a DES data block. Therefore, the same timing results presented in section 5.1.3 will be used here. Celator was synthesized by Synopsys Design Compiler with Atmel Standard Cells resources and the 130 nm technology.

The version of Celator presented here needs 590 clock cycles to encrypt a DES 64-bit data block. It achieves 22 Mbps throughput when operated at 190 MHz. This throughput is smaller than other circuits performances. The slowest operation in DES

Figure 5.14 – ***DES: encrypting a colour image in ECB***

(a) **(b)**



Celator implementation is the permutation. In DES there are up to 5 permutations, and 3 of them are executed 16 times. A PE array like Celator seems to not be fully suitable to execute these permutations, that are bitwise operations, and leaves most of the PEs unused.

The software solution proposed in [36] operates on a 64-bit processor, which is very suitable to manage 64-bit data block. These S-boxes are hardwired. Therefore this solution is not fully reconfigurable, unlike Celator is.

The Atmel hardware macro needs 16 clock cycles to perform a DES encryption, i.e. a cycle per DES round. This is one of the best throughput suitable for embedded systems. Again, this solution is not reconfigurable, unlike Celator is.

The FPGA based solution is more then 12 times faster than Celator, thanks especially to its parallelized S-boxes.

Celator has a good percentage of reconfigurability given by the portfolio of its instructions and proved by the implementation of other algorithms (sections 5.1 and 5.3): the PE array data-path can be modified and up to eight ALU operations can be selected. Like AES execution, DES execution is slower on Celator than on dedicated hardware macros. Unlike AES execution, DES execution is slower on Celator than on hardware implementations in FPGA or on GPP.

Table 5.3 – ***Comparisons of areas and performances for encrypting a DES 64-bit data block in EBC mode.***

	Type	Cycles (# of)	Frequency (MHz)	Bit-rate (Mbps)
Atmel	HW	16	100	400
Saqib [37]	FPGA	–	–	274
Ebiham 1 [36]	SW	140	300	167
Ebiham 2 [36]	SW	417	300	46
Celator	HW/SW	590	190	22

5.3 SHA

We implemented the SHA-256. The input block for a such hashing algorithm is 512 bits, while the output digest is over 256 bits. In the SHA-256 main loop eight 32-bit variables are used. As the whole PE array can compute up to four 32-bit values at once, Celator compute the eight SHA variables in 2 times.

5.3.1 Implementation of the SHA into a PE Array – Confidential

This section is Atmel confidential.

5.3.2 FPGA results

Celator was programmed to operate a SHA on *Lena* image (Figure 5.15). The Celator dedicate assembler code for SHA is annexed (section D).

The SHA output of Figure 5.15 is the following eight 32-bit words (hexadecimal format):

```
D0E309A7  88BE2E1B  255BEE42  B18B0675  174E1E05  69063F30  D748EEF4  F236D21D
```

After changing a single pixel on the original image, which is unrecognisable by the naked human eye (Figure 5.15b), the SHA output is:

Figure 5.15 – ***The versions of* Lena *image.*** *The original image is (a), in (b) one pixel is changed*

(a) **(b)**

`38F26C9A B2DC15A3 845E6AAD 6B94495C 9747FE14 86E513D1 D2FD2CE7 BDA331C3`

The digest of Figure 5.15a is totally different of the 5.15b.

5.3.3 ASIC results

Celator performances for the SHA algorithm are given thereafter and compared with the performances of other cryptographic structures (Table 5.4). We chose to compare Celator with some SHA dedicated hardwired macros. The SHA dedicated hardware macros are the macros presented in [39], [38] and [40].

Since Celator is reconfigurable, we do not need to modify Celator architecture in order to encrypt a SHA data block. Moreover, the same timing results presented in section 5.1.3 will be used here. Celator was synthesized by Synopsys Design Compiler with Atmel Standard Cells resources and the 130 nm technology.

The version of Celator presented here needs 6000 clock cycles to hash a SHA 256-bit data block. It achieves 16 Mbps throughput when operated at 190 MHz. With some slight hardware modifications, which are already under study, Celator can hash a SHA 256-bit data block with an estimated 46 Mbps throughput. This throughput is smaller than other circuits performances. The slowest operations in SHA Celator implementation are the data transfers among all registers that store the SHA values (SHA values are eight 32-bit registers). In order to speed up the data transfers, thus the SHA execution, some local memory blocks must be included in the Celator implementation (other than the CRAM).

Table 5.4 – ***Comparisons of areas and performances for hashing 512-bit data block, according to SHA-256.***

	Type	**Cycles** (# of)	**Frequency** (MHz)	**Bit-rate** (Mbps)
Rchaves [39]	HW	65	–	1400
Iahmad [38]	HW	–	–	1000
Cadence datasheet [40]	HW	70	133	971
Celator	HW/SW	6000	190	16

Like for AES and DES execution, the SHA execution is slower in Celator than in dedicated hardware macros. But dedicated macros are not reconfigurable, unlike Celator is. Celator has a good percentage of reconfigurability given by the portfolio of its instructions and proved by the implementation of other algorithms (sections 5.2 and 5.3): the PE array data-path can be modified and up to eight ALU operations can be selected.

6

Conclusions and Further Work

This thesis describes an original implementation of a multi-algorithm Crypto-Co-Processor called Celator. Celator can perform the AES, DES and SHA algorithms. Celator is aimed to be integrated in an ASIC for embedded circuits by using Atmel Standard Cells, and it can be included in a Smart Card or in a Smart Card Reader. We propose the systolic architecture to make Celator reconfigurable, which is a novelty, at our knowledge, for cryptographic applications. Usually FPGA techniques are adopted to make a device reconfigurable [49]. However, our reconfigurable solution is FPGA less. The developed Celator processor has a degree of reconfigurability. We found a reconfigurable technology by using a systolic processor based on a Processing Element (PE) array.

In chapter 1 we have presented a brief history of the security. Confidential data have to be hidden, or encrypted, in order to ensure the people privacy (ID), and to allow the people to access to a required service (communications, electronic payment, TV on demand et cetera). Thus we need secure products. But these products must be protected against side channel attacks, otherwise the protection they are supposed to offer is leaked. Side channel attacks is one example of threats that must be countered.

Another big family is the fault injection, which can be very powerful for an attacker too.

In chapter 2 we have seen three algorithms that we have executed on Celator: AES, DES, and SHA algorithm. AES and DES are used to encrypt/decrypt a message by private keys, while SHA is used to hash a message. These algorithms are largely used in most of the nowadays secure products.

In chapter 3 we have described some hardware and software implementations of cryptographic algorithms. The execution on GPP is slower than on hardwired macros, but hardwired macros are larger in terms of equivalent gates. Hardwired macros are not reconfigurable unlike GPP. For us, the good trade off choice is a solution "in the middle", between GPP and hardwire macros. Celator performances are situated among those of dedicated hardware macros and those of GPPs.

In chapter 4 we have presented Celator. Celator is made of 16 Processing Elements arranged in a systolic fashion (PE array), of a local RAM (the CRAM) and of a Controller, which includes an FSM. The FSM reads the instructions in the CRAM, and the PE array executes them under the control of the FSM. Celator can be reconfigured by the CPU. Given its structure based on a PE array, all matrix format data can be easily computed.

In chapter 5 we have described how Celator can be reprogrammed in order to execute AES, DES and SHA algorithm. We have shown how Celator can operate on 32-bit, 8-bit and 1-bit data words. We have prototyped Celator on a Xilinx FPGA card to test it. Celator has been synthesized and placed and routed for an ASIC with Atmel Standard Cells. Results show how Celator is a good trade off choice of crypto-processor with respect to the number of gates, the number of execution cycles and the reconfigurability.

In this thesis the security aspects are not fully analysed. We consider "writing" or "reading" operations into/from a register or the CRAM as trusted operations. Given the structure of Celator, during the encrypting or decrypting operations, data can be masked; for instance data can be xored once or several times with random data in order

to become more difficult to be intercepted by hackers through Side Channel Attacks (SCA) [50], [51]. Furthermore, several `nop` operations can be added to certain Celator programs, in order to have the same execution time for all algorithms performed by Celator. In this way, Celator would be timing attack resistant.

Celator will be included in the next generation of Atmel Smart Card Readers. Recent studies show how Celator performances can be improved with a slight modification of its architecture. Celine Huynh Van Thieng [52] is currently developing a combinatorial block in the PE array, which is dedicated to the bit permutations on 32-bit words based based on omega–flip networks [53]. She is also allowing the FSM to access the CRAM by both ports. In this way, the execution of the AES, DES and SHA in Celator is speeded up. The cost of these modifications is to add 10% of the equivalent gates only. Thus, the AES execution can be reduced to 510 clock cycles only. The DES execution can be reduced to 470 clock cycles only: the DES permutations benefit from the permutation-dedicated block. The SHA execution can be reduced to 3700 clock cycles only: in the legacy SHA implementation there are many data transfers from/to the CRAM, which are speeded up by the new access to the CRAM.

As further work on Celator, we will improve the security against all SCA, and we will study the implementation of other algorithms in Celator. Thanks to its generic structure and its generic instruction set, Celator can be multi-algorithms. According to the Atmel strategy, Celator can be required to obtain an EAL evaluation. If so, a Protection Profile will be written in order to fit the EAL5+ specific requirements. Another main point of the further work will be to study and develop a trusted platform to secure exchanges of data between the CPU and Celator, as well as between the CPU and an external data source.

7

Résumé en langue française de la thèse intitulée "Design and development of a reconfigurable cryptographic co-processor" par Daniele Fronte

Ce chapitre résume ma thèse en langue française.

7.1 Résumé

Les circuits à haute technologie d'aujourd'hui requièrent toujours plus de services et de sécurité. Le marché correspondant est orienté vers de la reconfigurabilité. Dans cette thèse je propose une nouvelle solution de coprocesseur cryptographique multi-algorithmes, appelé Celator. Celator est capable de crypter et décrypter des blocs de données en utilisant des algorithmes cryptographiques à clé symétrique tel que l'Advanced Encryption Standard (AES) [1] ou le Data Encryption Standard (DES) [2]. De plus, Celator permet de hacher des données en utilisant le Secure Hash Algorithm (SHA) [3]. Ces algorithmes sont implémentés de façon matérielle ou logicielle dans les produits sécurisés. Celator appartient à la classe des implémentations matérielles flexibles, et permet à son utilisateur d'exécuter des algorithmes cryptographiques standards ou propriétaires.

L'architecture de Celator est basée sur un réseau systolique de 4x4 Processing Elements, nommé réseau de PE, commandé par un Contrôleur réalisé par avec une Machine d'États Finis (FSM) et une mémoire locale.

Cette thèse présente l'architecture de Celator, ainsi que les opérations de base nécessaires pour qu'il exécute AES, DES et SHA. Les performances de Celator sont également présentées, et comparées à celles d'autres circuits sécurisés.

7.2 Introduction

La tendance du marché des produits sécurisés est d'offrir plus de services et sécurité aux utilisateurs. Il se peut qu'un dispositif électronique doive exécuter des algorithmes pour lesquels il n'avait pas été conçu. Notre challenge est l'implémentation d'un coprocesseur cryptographique multi-algorithmes, appelé Celator. Celator est conçu pour être intégré dans un ASIC pour circuits embarqués, réalisés par des cellules standards de la société Atmel. Plus précisément, Celator doit être inclus dans des cartes à puce et dans des lecteurs de cartes à puce.

En utilisant des opérations logiques et arithmétiques de base, Celator est capable d'exécuter les algorithmes suivants:

- Advanced Encryption Standard (AES-128, AES-196 et AES-256), [1]
- Data Encryption Standard (DES), [2]
- Secure Hashing Algorithm (SHA-256), [3]

La solution que nous avons adoptée pour Celator est basée sur un réseau systolique à maille carrée de 4x4 Processing Elements, nommé réseau de PE. Le chemin de données du réseau de PE ainsi que la Machine d'Etats Finis sont reconfigurables.

Le reste de ce chapitre est organisé comme suit. Le paragraphe 3 introduit les algorithmes exécutés par Celator. Le paragraphe 4 présente l'architecture matérielle de Celator. Le paragraphe 5 montre les résultats de validation sur FPGA, et également les résultats de validation ASIC de Celator. Dans le paragraphe 5 je donnerai des conclusions sur Celator.

7.3 Trois algorithmes cryptographiques

Ce paragraphe introduit brièvement les trois algorithmes qui ont été implémentés dans Celator: AES, DES et SHA. Le lecteur peut trouver une description complète de ces algorithmes dans [1, 2, 3].

7.3.1 L'algorithme AES

L'Advanced Encryption Standard (AES) est un algorithme de cryptage et décryptage à clé privée, approuvé comme standard par le Federal Information Processing Standards (FIPS). C'est le successeur de DES décrit dans le paragraphe suivant. Cependant, DES est encore de nos jours très demandé par les utilisateurs. L'AES existe en trois versions AES-128, AES-196 et AES-256, dans lesquelles la clés initiale est codée sur 128, 192 et

256 bits. Dans les trois versions, l'AES crypte ou décrypte des messages par blocs de 128 bits. Dans cette thèse, nos études se sont portés sur l'AES-128

Chaque bloc de données peut être géré comme une matrice de 4x4 octets (Table 7.1).

Table 7.1 – ***La matrice de 4x4 octets pour un bloc de données AES-128.***

d_{11}	d_{12}	d_{13}	d_{14}
d_{21}	d_{22}	d_{23}	d_{24}
d_{31}	d_{32}	d_{33}	d_{34}
d_{41}	d_{42}	d_{43}	d_{44}

Le procédé de cryptage AES inclut 10 boucles. Chaque boucle (à l'exception de la première et de la dernière) comporte les transformations suivantes :

1. Sub-Bytes
2. Shift-Rows
3. Mix-Columns
4. Add-Round-Key

7.3.2 L'algorithme DES

L'algorithme Data Encryption Standard est un algorithme de cryptage et de décryptage à clé privée, constitué d'une permutation initiale IP, de 16 boucles de transformations, et d'une permutation finale IP^{-1}. Dans chaque boucle, une opération "ou exclusif", une fonction f, ainsi que deux permutations sont exécutées : PC_1, PC_2. La fonction f inclut l'expansion E, la permutation P et huit substitutions S. Contrairement à AES, les permutations de DES se font bit à bit. Leur exécution par Celator, qui travaille sur des mots d'un octet, montre comment Celator peut travailler avec un format de données d'un bit.

7.3.3 L'algorithme SHA

Le SHA-1 et le SHA-2 (c'est-à-dire SHA-256, SHA-384, et SHA-512) sont quatre Secure Hash Algorithms spécifiés par [3]. Ces algorithmes sont itératifs, représentent des fonctions de hachage à une direction (one-way hash functions) qui peuvent traiter un message pour produire un condensé appelé haché (ou digest en anglais).

Chaque algorithme peut être décrit en deux étapes : d'abord le prétraitement, puis le calcul du condensé. Le prétraitement implique le remplissage d'un message (padding), la décomposition de ce message formaté en plusieurs blocs de m bits (m= 512 pour SHA-1 et SHA-256, m= 1024 pour SHA-384 et SHA-512), et l'initialisation de valeurs qui seront ensuite utilisées dans le calcul du condensé. Le calcul du condensé génère une liste de messages à partir du message rempli et utilise cette liste, à travers des fonctions, constantes, et opérations sur des mots pour générer itérativement une série de valeurs hachées. La valeur finale du condensé généré par ce calcul est utilisée pour déterminer le message haché.

Comme décrit dans [3], 8 variables intermédiaires de 32 bits (a, b, c, ..., g, h) sont nécessaires pour le calcul de SHA. Ces variables sont initialisées par 8 constantes 32 bits données par le standard $(H_1, H_2, \ldots, H_8)$.

7.4 Implémentations matérielles et logicielles d'algorithmes cryptographiques : état de l'art

Les algorithmes cryptographiques peuvent être exécutés sur des processeurs à usage générique (GPP, de l'anglais General Purpose Processor), ou sur des dispositifs dédiés, comme les macros matérielles. Généralement, les macros matérielles ont de meilleures performances que les GPP, mais les GPP sont reconfigurables à 100%, alors que les macros sont dédiés et donc pas reconfigurables.

7.4.1 Le NEC DRP

Un processeur dynamiquement reconfigurable (DRP) a été décrit par NEC dans [54]. Le DRP est composé d'un réseau à 2 dimensions fait de plusieurs Processeurs Élémentaires (PE), une machine d'états et d'éléments de mémorisation. Un PE contient un banc de registres, et une ALU. Avant de commencer un calcul, les chemins de données et les programmes de contrôle doivent être sauvegardés. A cette fin, un programme écrit en langage C est d'abord compilé, puis synthétisé et ensuite téléchargé dans le DRP.

Une telle architecture systolique se montre bien adaptée pour des calculs en parallèle sur des données sous format matriciel. Son point négatif est le temps de compilation-synthèse-téléchargement du programme de contrôle. Si une architecture similaire est présente dans Celator, le temps de contrôle devra être minimisé.

7.4.2 La macro SHARMA

Dans [32] une architecture systolique est utilisée pour construire un dispositif à hautes performances dédié à AES, incluant une mémoire et un réseau de PE. Les données circulent de la mémoire vers les PE pour ensuite retourner vers la mémoire.

Les quatre opérations d'AES sont réalisées de la façon suivante :

1. Les transformations Sub-Bytes sont réalisées à l'aide d'un bloc ASIC. Avec quelques connexions, la même structure peut être modifiée et utilisée tant en cryptage qu'en décryptage.

2. Les transformations Shift-Rows ne demandent pas d'implémentations séparées. Les données sont décalées à l'aide des registres R0–R5.

3. Pour les transformations Mix-Columns, une architecture systolique est utilisée pour effectuer la multiplication matricielle du réseau 4x4

4. Dans les transformations Add-Round-Key, des portes combinatoires XOR sont utilisées.

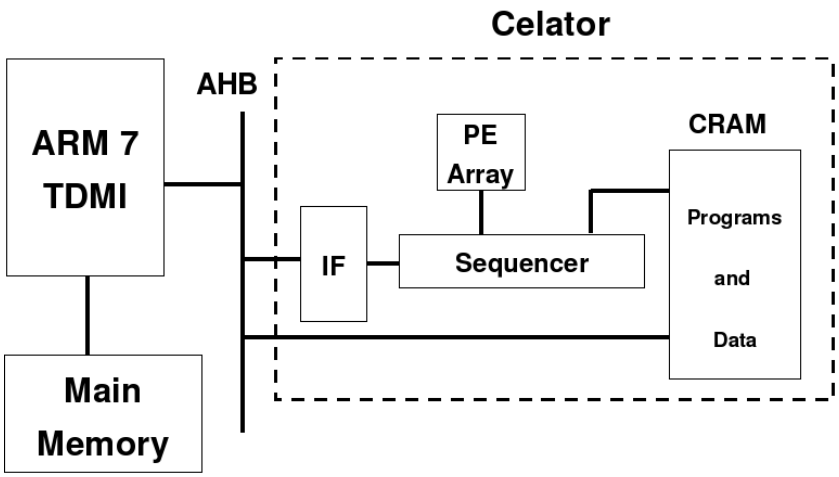
Une telle implémentation systolique requiert 40 cycles d'horloge pour crypter un bloc de 128 bits avec AES, atteignant une vitesse théorique d'exécution de 3,2 bits par cycle. Sachant qu'une boucle AES comprend 4 transformations et qu'une boucle est répétée 10 fois par cryptage, alors chaque transformation est exécutée en un cycle d'horloge seulement. Cela représente un résultat excellent.

Le désavantage d'une telle macro est la non reconfigurabilité : elle est capable d'exécuter un seul algorithme, AES dans ce cas. Néanmoins, l'approche systolique se montre un bon candidat pour crypter/décrypter des données en format matriciel. Notre solution de coprocesseur doit améliorer cette idée afin d'obtenir un circuit intégré reprogrammable et multi-algorithmes.

7.5 L'architecture matérielle de Celator

Dans cette section nous allons illustrer le développement de Celator et l'architecture du système dans lequel s'insère Celator. Le système considéré (Fig. 7.1) comprend une CPU basée sur un processeur ARM, une mémoire centrale, Celator et d'autres périphériques.

Figure 7.1 – ***Système complet incluant Celator***



Celator comprend un réseau de PE, un contrôleur, et une mémoire propre à Celator, appelée CRAM. Les Processeurs Élémentaires sont les blocs de base à partir desquels le réseau de PE est construit. Celator peut avoir une horloge de fonctionnement différente du CPU. L'unité IF assure la communication entre Celator et le CPU.

7.5.1 Le réseau de PE

Le réseau de PE (Fig. 7.1) est composé d'un réseau 4x4 de Processeurs Élémentaires (PE). Tous les PE sont identiques. Le réseau de PE est contrôlé par le Séquenceur.

Le réseau de PE a une structure d'entrée/sortie à deux dimensions (2D I/O). De cette façon, chaque PE (Fig. 7.3) a 4 entrées données, c'est-à-dire une par chaque direction cardinale : Nord, Sud, Ouest et Est. Chaque PE possède une sortie donnée, connectée à l'entrée donnée de ses 4 PE voisins les plus proches. Dans un PE, toutes les entrées et sorties données sont de 1 octet. Un PE inclut un registre 8 bit et une ALU.

Figure 7.2 – ***L'architecture du réseau de PE***

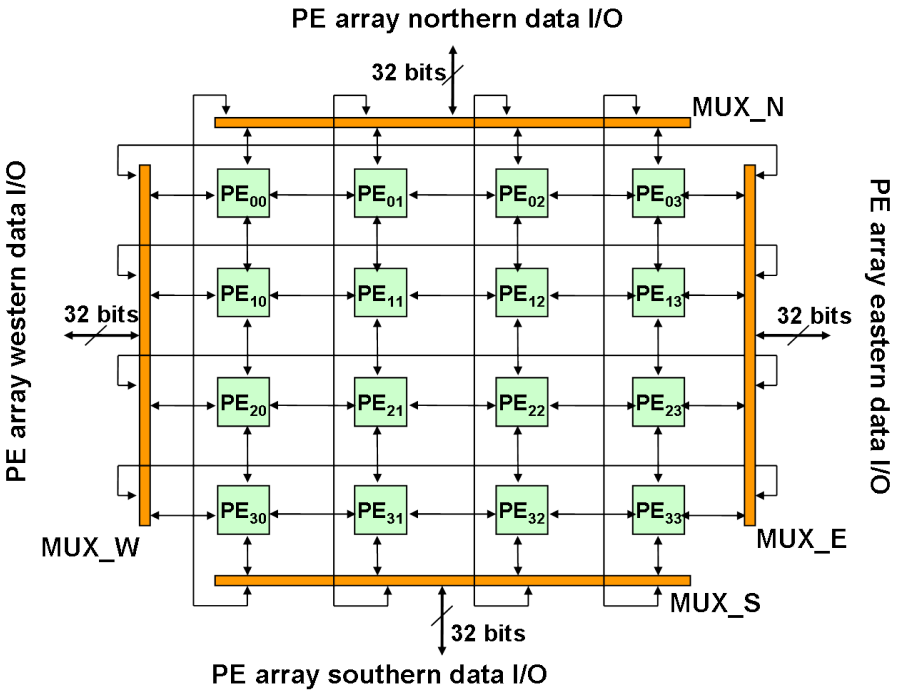
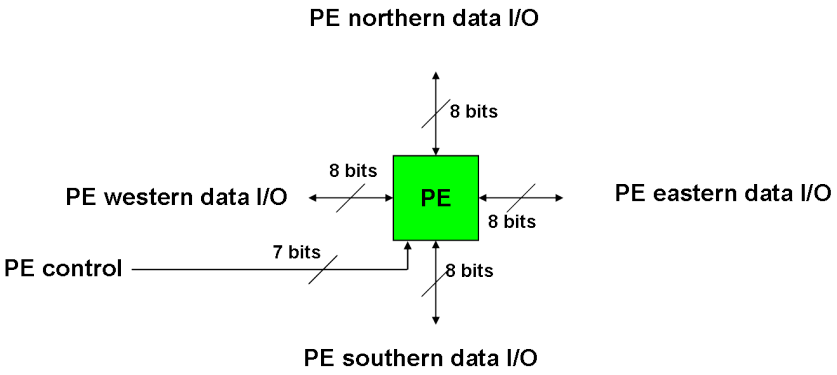


Figure 7.3 – ***Les entrées/sorties du PU***



Chaque PE reçoit des signaux de contrôle du Séquenceur afin de traiter les données. Les signaux de contrôle peuvent :

1. charger les données du PE nord, sud, ouest, est, la CRAM, le CPU, ou une source extérieure ;

2. choisir quelle donnée enregistrer entre la donnée qui vient d'être calculée et la donnée en entrée du PE ;

3. choisir la sortie donnée du PE entre la donnée précédemment enregistrée et la donnée qui vient d'être élaborée ;

4. choisir une des opérations de l'ALU ;

5. lire/écrire à partir du PE ouest/nord et écrire/lire dans les PE est/sud.

Un PE peut exécuter seulement une instruction par cycle d'horloge.

L'ALU de chaque PE peut exécuter les opérations arithmétiques et logique sur des opérandes de 8 bits : *addition modulo 256, addition modulo 255, ou, ou exclusif, xtime, et, inversion* et *non-opération.*

Étant donné que les multiplexeurs des PE sont configurés par la machine d'états (FSM) du Séquenceur, et que cette configuration peut être modifiée à chaque cycle

d'horloge, le chemin des données du réseau de PE est donc *reconfigurable*, tout comme la FSM.

Nous avons choisi une granularité de PE sur 8 bits. De cette façon, les registres et l'ALU de chaque PE permettent de manipuler très aisément des octets. Comme nous le montrerons dans le paragraphe 6, le choix de ce format de données pour le PE permettra l'exécution d'algorithmes n'ayant pas le même format de données, tels que DES et SHA.

Par exemple, comme chaque PE traite des mots de données de 8 bits, alors une ligne entière de PE peut traiter des mots de données de 32 bits. Ainsi, le réseau de PE peut être considéré comme travaillant sur quatre mots de données de 32 bits. Cette propriété peut être exploitée afin de résoudre des algorithmes tels que SHA-256, qui opère sur huit mots de données de 32 bits. Les calculs du SHA-256 peuvent être exécutés par Celator en deux temps.

De plus, ce type d'architecture peut être aussi utile pour exécuter des algorithmes tels que DES, parce que DES opère sur des mots de données de 64 bit, qui est la taille de 2 colonnes PE.

7.5.2 Le Séquenceur

Le Séquenceur génère les signaux de contrôle pour le réseau de PE et les instructions pour le CPU. Aussi, il contrôle les entrées et sorties données du réseau de PE. Le Séquenceur est composé d'une FSM et d'éléments de stockage.

Les instructions de la FSM sont simples, et ne sont pas dédiées à AES. Il s'agit d'instructions génériques qui peuvent être réutilisées pour implémenter d'autres algorithmes. Par exemple, avec une seule instruction la FSM peut copier des données de la CRAM vers le réseau de PE et vice-versa, avec un offset spécifique ou pas, ou peut effectuer une rotation de données dans le réseau de PE. Les instructions de la FSM sont enregistrées dans la CRAM.

7.5.3 La CRAM

La mémoire RAM de Celator, ou CRAM, enregistre des mots de données et d'instructions sur 32 bits. Parmi les données, on compte les données d'entrées/sorties qui doivent être cryptées ou décryptées, ou bien les données nécessaires au procédé de cryptage/décryptage, comme les Sbox pour AES. Parmi les programmes enregistrés dans la CRAM, on compte les instructions pour le Séquenceur. La CRAM est une RAM double port, ainsi le CPU et le Séquenceur peuvent accéder simultanément à la CRAM en mode lecture/écriture.

7.6 Comment Celator exécute les algorithmes cryptographiques

7.6.1 Les transformations d'AES

Celator peut exécuter les 4 transformations demandées par un cryptage, un décryptage et la génération des clefs d'AES.

Le réseau de PE travaille comme un processeur systolique de type Complex Instruction Multiple Data (CIMD) [42]. La CRAM contient les instructions pour la FSM. Ces instructions sont chargées par la FSM et ensuite sont exécutées par le réseau de PE. Le CPU charge d'abord l'algorithme cryptographique, par exemple l'AES, dans la CRAM et ensuite le CPU démarre Celator. Le CPU contrôle Celator via l'unité IF et peut modifier quelques instructions de la séquence d'instructions de l'algorithme cryptographique, ou bien sélectionner un algorithme différent. Donc, le CPU peut reconfigurer Celator.

Sub-Bytes : la table Sbox est enregistrée dans la CRAM. La FSM du Contrôleur fournit les signaux pour exécuter une "look-up table" de chaque octet de donnée enregistré dans chaque PE. Un registre de la FSM est utilisé.

Shift-Rows : chaque ligne du réseau de PE est décalé de 1, 2, ou 3 positions respectivement pour les colonnes 1, 2 or 3.

Mix-Columns : l'opération fondamentale de la transformation Mix-Columns est

la multiplication $S'(x) = A(x) * S(x)$.

Chaque octet de l'état S(x) est multiplié par le vecteur polynomial A(x), comme décrit dans [1], par l'opération xtime et ensuite xoré par colonnes. Les deux opérateurs xtime et xor sont présents dans l'ALU.

Add-Round-Key : les clefs sont chargées à partir de la CRAM dans le réseau de PE et ensuite "xorées" avec les données du réseau.

7.6.2 Les transformations de DES

Les permutations de DES peuvent être exécutées de la façon suivante. Le mot à permuter $A = a_1 \dots a_{64}$ est multiplié par une matrice identité. Du moment que les ai sont des éléments binaires, le résultat de la multiplication sera constitué de 64 lignes avec un seul "1" et soixante "0". Ensuite ces mots sont rotées, et xorés entre eux.

Comme un seul PE traite des données de 8 bits, quatre PE peuvent être concaténés afin de traiter des données sur 32 bits. Ainsi, le réseau de PE peut être considéré comme un ensemble de quatre lignes de PE de 32 bits chacune. Afin d'exécuter DES, 2 lignes de 4 PE chacune peuvent être utilisées pour traiter un bloc de données de 64 bits.

Les substitutions S de DES sont exécutées comme les transformations Sub-Bytes de AES.

7.6.3 Les transformations de SHA-256

Nous supposons que le prétraitement des données en entrée est réalisé par le CPU, et que Celator traite le calcul du condensé seulement. Nous allons détailler l'implémentation du calcul du condensé de SHA-256.

Comme décrit dans [3], les variables requises pour l'exécution de SHA sont les suivantes :

1. 8 variables intermédiaires de 32 bits (a, b, c ... g, h);
2. 48 variables 32 bits W_j;

3. 8 variables 32 bits : $T_1, T_2, Ch, Maj, \Sigma_0, \Sigma_1, \sigma_0, \sigma_1, ROTR$;

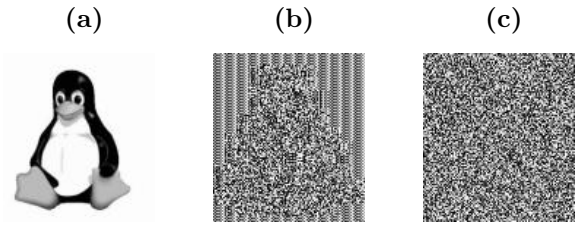
4. Le résultat est $H_N = (H_{n1}, H_{n2}, \ldots, H_{n8})$, c'est-à-dire 8 variables 32 bits;

Comme pour DES, plusieurs PE sont utilisés pour traiter les variables du calcul du condensé. Dans le cas de SHA-256, quatre PE sont concaténés pour traiter des mots de 32 bits.

7.6.4 Modes ECB et CBC

Celator est capable de crypter et décrypter un bloc de données AES et DES en mode Electronic Codebook (ECB) et en mode Cipher Bloc Chaining (CBC) [30]. La Fig. 7.4 montre la différence entre le cryptage AES d'une image en noir et blanc ([55], Fig. 7.4a) en mode ECB (Fig. 7.4b) et en mode CBC (Fig. 7.4c).

Figure 7.4 – ***Cryptages AES en modes ECB et CBC***



7.7 Résultats et discussions

Dans cette section nous détaillons les performances de Celator et nous les comparons aux performances d'autres dispositifs cryptographiques. Nous avons choisi de comparer Celator à des GPP [56, 41, 57], à des macros matérielles dédiées [44, 37, 36] et aussi à une architecture dynamiquement reconfigurable [48].

Le système complet placé autour de Celator est présenté dans la Fig. 7.1. Ce système a été développé en Verilog au niveau RTL (Register Transfer Level), simulé par Mentor ModelSim synthétisé par Synopsys DC avec des ressources Standard Cells

d'Atmel, avec une technologie de 130 nm (longueur du canal du transistor). Ces mêmes outils ont été utilisés pour déterminer les surfaces ASIC des microprocesseurs Atmel et AVR. Les résultats correspondants sont montrés et commentés plus bas.

La version de Celator présentée ici requiert 514 cycles d'horloge pour crypter un bloc de données par AES de 128 bit, avec un débit de 47 Mbps en mode ECB. En mode CBC, le même bloc est crypté en 524 cycles d'horloge, soit 46 Mbps (Table 7.2). Même si une macro matérielle dédiée est plus rapide que notre solution (la macro matérielle AES d'Atmel prend 40 cycles, c'est-à-dire un cycle pour chaque transformation AES), elle a besoin d'une surface plus importante. Les GPP tels qu'ARM 7 TDMI et ARM 9 occupent une superficie plus importante et sont plus lents que Celator. Le GPP AVR a la même taille mais il est plus lent que Celator.

Table 7.2 – ***Comparaison des surfaces et des débits (AES en mode CBC) entre macros dédiées, macros reconfigurables et GPP***

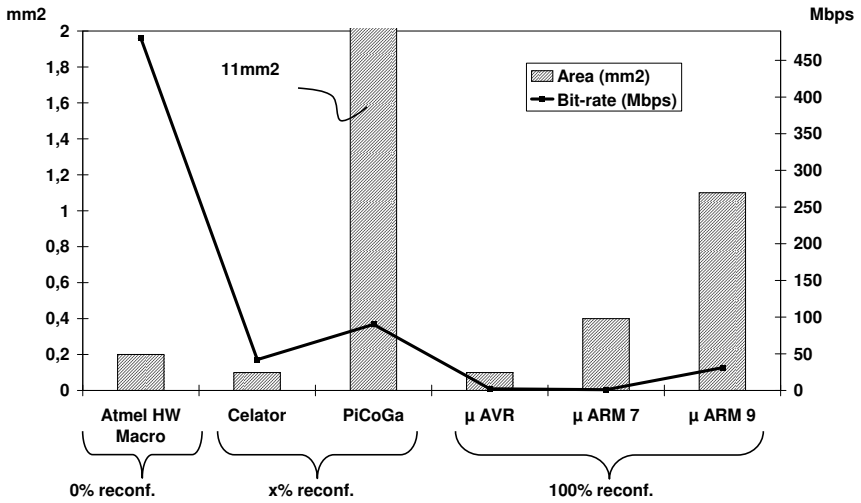
	cycles (# of)	**Max Freq.** (MHz)	**Area** (mm^2)	**Bit-rate** (Mbps)	**Techno.** (nm)
Atmel SOMA [44]	40	150	0.2	480.0	130
Celator	580	190	0.1	42.0	130
μ **PiCoGa**[48]	285	200	11.0	90.0	90
μ **AVR**	5000	40	0.1	1.0	130
μ **ARM7TDMI** [41]	3600	65	0.4	2.3	130
μ **ARM9** [43]	830	200	1.1	31.0	130

Les macros présentées dans [47], et [46, 32] ont des performances (exprimées en bits par cycle) inférieures ou égales à celles des macros comme Atmel [44], c'est-à-dire 3.2 bits par cycle pour crypter un bloc de données AES de 128 bits en mode ECB.

Les macros matérielles dédiées (Table 7.2) ont un débit plus élevé (en bit par seconde) que Celator mais, contrairement à Celator, elles ne sont pas reconfigurables. Une architecture dynamiquement reconfigurable tel que PiCoGa [48] a un débit plus élevé

que Celator mais, même si PiCoGa est reconfigurable, sa surface est beaucoup plus importante.

Figure 7.5 – ***Surface et débits pour des cryptages AES-128 en mode CBC de macros dédiées, de processeurs multi-algorithmes et de GPP***



Étant donné que Celator est reconfigurable, nous n'avons pas besoin de modifier son architecture afin de crypter un bloc de données DES.

Celator a besoin de 476 cycles d'horloge pour crypter un bloc de données DES de 64 bit. Il réalise un débit de 26 Mbps avec une fréquence de fonctionnement de 190 MHz.

Ce débit est plus faible que celui d'autres circuits (Table 7.3). Pour DES, l'opération la plus lente dans l'implémentation de DES dans Celator est la permutation. L'algorithme DES comporte 5 permutations, et 3 d'entre elles sont exécutées 16 fois. Un réseau de PE comme Celator ne semble pas pleinement adéquat à exécuter ces permutations, car il s'agit d'opérations bit à bit qui laissent la plupart des PEs non utilisés.

La solution logicielle proposée dans [36] utilise un processeur 64 bit, qui est très

utile pour manipuler des blocs de données 64 bit. Les tables S-boxes sont câblées en dur, donc cette solution, à la différence de Celator, n'est pas pleinement reconfigurable.

La macro matérielle d'Atmel requiert 16 cycles d'horloge pour exécuter un cryptage DES. Celle-ci est l'un des meilleurs débits dans des cartes à puce. Encore une fois, cette solution, à la différence de Celator, n'est pas reconfigurable.

Table 7.3 – ***Comparaison des performances pour le cryptage d'un bloc de données DES 64 bits en mode EBC***

	Type	**Cycles** (# of)	**Frequency** (MHz)	**Bit-rate** (Mbps)
Atmel	HW	16	100	400
Saqib [37]	FPGA	–	–	274
Ebiham 1 [36]	SW	140	300	167
Ebiham 2 [36]	SW	417	300	46
Celator	HW/SW	590	190	22

La solution basée sur l'utilisation d'une carte FPGA a un débit élevé, grâce surtout à ses S-boxes parallèles.

Celator requiert 2720 cycles d'horloge pour condenser un bloc de données de 512 bits, en utilisant l'algorithme SHA-256. Il atteint un débit de 36 Mbps avec une fréquence de fonctionnement de 190 MHz.

Ce débit est plus petit que les débits des autres circuits (Table 7.4). Les opérations plus lentes dans l'implémentation SHA de Celator sont les transferts de données entre tous les registres qui contiennent les variables utilisées dans SHA (il s'agit de 8 registres 32 bits). Afin d'accélérer ces transferts de données, et par conséquent l'exécution de SHA, quelques éléments de mémoire doivent être inclus dans Celator (outre la CRAM).

Pour conclure, nous pouvons dire que Celator a un bon pourcentage de reconfigurabilité fournie par le portfolio de ses instructions, qui peuvent modifier le chemin des données du réseau de PE et sélectionner jusqu'à 8 opérations de l'ALU.

Table 7.4 – ***Comparaison des performances pour condenser un bloc de données de 512 bits, en utilisant l'algorithme SHA-256***

	Type	**Cycles** (# of)	**Frequency** (MHz)	**Bit-rate** (Mbps)
Rchaves [39]	HW	65	–	1400
Iahmad [38]	HW	–	–	1000
Cadence datasheet [40]	HW	70	133	971
Celator	HW/SW	6000	190	16

Il résulte aussi clairement que Celator représente un bon compromis entre les macros matérielles dédiées (qui ne sont pas programmables), les architectures dynamiquement reconfigurables (qui ont un bon pourcentage de programmibilite) et les GPP (qui sont pleinement programmables) en termes de surfaces, débit et flexibilité (Fig. 7.5).

7.8 Conclusions

Le coprocesseur cryptographique multi-algorithmes nommé Celator est présenté dans cette thèse. Celator peut exécuter les algorithmes AES, DES et SHA-256. Il est composé de 16 Processeurs Élémentaires (PE) disposés en réseau systolique (réseau de PE), d'une mémoire RAM locale (la CRAM) et d'un Contrôleur (qui inclut une FSM). La FSM lit les instructions dans la CRAM, et le réseau de PE les exécute sous le contrôle de la FSM. Celator peut être programmé par le CPU. Grâce à sa structure basée sur un réseau de PE, toutes les données sous format matriciel peuvent être facilement traitées.

Celator est un bon compromis de crypto-coprocesseur par rapport au nombre de portes équivalentes, au nombre de cycles d'exécution et au pourcentage de programmabilité. Les performances de Celator se situent entre celles de macros matérielles dédiées et celles des GPP.

Dans cette thèse, les aspects sécuritaires n'ont pas été analysés. Nous considérons les opérations d'écriture ou de lecture dans les registres ou dans la CRAM comme des opérations sûres. Grâce à la structure de Celator, pendant les procès de cryptage ou décryptage, les données peuvent être masquées : par exemple, les données peuvent être "xorées" une ou plusieurs fois avec des variables aléatoires, afin de les rendre plus difficilement interceptables par des personnes malveillantes, par des attaques de type Side Channel Attacks [50, 51].

Le prochain pas de nos investigations sera l'implémentation d'autres algorithmes dans Celator. L' architecture du réseau de PE sera peut-être modifiée, mais sa surface ne sera pas affectée de manière considérable, grâce surtout à sa structure et à son jeu d'instruction génériques.

Notre objectif à long terme sera l'étude et le développement de plateformes sûres pour sécuriser les échanges de données entre le CPU et Celator, ainsi que entre le CPU et des sources de données externes.

Acknowledgments

This research was sponsored by the Erevna European research project, and was a collaboration with the following organizations:

- Institut Matériaux Microélectronique Nanosciences de Provence (IM2NP), Marseille, France
- Atmel, Rousset, France

I would like to thank Annie Perez and Eric Payrat, without whom this thesis would not have been possible. I would also like to thank my parents, without whom I would not have been possible.

I was fortunate to have a conscientious and supportive thesis committee: thanks to Michele Elia and Lionel Torres for their helpful advices; thank to Dominique Borrione, Jean-Michel Portal and Luc Jeannerot for attending the thesis committee.

I warmly thank all at IM2NP and Atmel, especially Sabine De Molliens, Nicolas Ferrazzi, David Moreira, Alexandre Croguennec, Jean-Charles Lesage and Nadine Fabre: they were generous with thier time in reviewing this thesis.

Thanks to Fanny, who lovely supported me in these years.

A

Annexes – Confidential

A.1 Celator assembler converter

This section is Atmel confidential.

B

Annexes: AES codes – Confidential

B.1 AES Celator assembler code

This section is Atmel confidential.

B.2 AES Filling CRAM code

This section is Atmel confidential.

B.3 C function

This section is Atmel confidential.

B.4 Validating ARM code

This section is Atmel confidential.

C

Annexes: DES codes – Confidential

C.1 DES tables

This section is Atmel confidential.

C.2 DES Celator assembler code

This section is Atmel confidential.

D

Annexes: SHA codes – Confidential

D.1 SHA Celator assembler code

This section is Atmel confidential.

Bibliography

[1] "Specification for the advanced encryption standard (AES)," Federal Information Processing Standards Publication 197, 2001. [Online]. Available: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[2] "Data encryption standard (DES)," Federal Information Processing Standards Publication 46, Tech. Rep., 1999. [Online]. Available: http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf

[3] "Secure hash standard (sha-2) (+ change notice to include sha-224)," Federal Information Processing Standards Publication 180, Tech. Rep., 2002. [Online]. Available: htt://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf

[4] D. Fronte, "Etude pour un crypto processeur reconfigurable," in *Journées Nationales du Réseau Doctoral en Microélectronique*, Ecole Nationale Supérieure de Télécom de Paris, 2005. [Online]. Available: http://jnrdm.free.fr/Conference.php

[5] S. Charbouillot, A. Pérez, and D. Fronte, "A programmable hardware cellular automaton: example of data flow transformation," *VLSI Des.*, vol. 2008, no. 1, pp. 1–7, 2008.

[6] D. Fronte, A. Perez, and E. Payrat, "A multi-algorithm cryptographic co-processor," in *Computing, Communications and Control Technologies, in the context of the International Multi-Conference on Engineering and Technological Innovation*, 2008.

[7] ——, "The aes in a systolic fashion: Implementation and results of celator processor," in *IEEE International Conference on Electronics, Circuits, and Systems*, 2008.

[8] ——, "System and method for encrypting data," U.S. Patent US2 008 062 803, 2008. [Online]. Available: http://v3.espacenet.com/textdoc?DB=EPODOC&IDX=US2008062803&F=0

[9] A. Perez, C. Huynh Van Thieng, and D. Fronte, "Application de la physique statistique à la cryptographie," in *Congrès Général 2007 de la Société Française de Physique*, 2007.

[10] V. Mollet, "Implémentation de fonctions cryptographiques sur fpga et programmation d'un microprocesseur arm," diploma thesis, 2007.

[11] B. Collard, "La cryptographie dans l'antiquité gréco-romaine," *Folia Electronica Classica*, 2004. [Online]. Available: http://bcs.fltr.ucl.ac.be/FE/07/CRYPT/Crypto44-63.html#42047

[12] A. R. Miller, "The cryptographic mathematics of enigma," National Security Agency, Tech. Rep., 2001. [Online]. Available: http://www.nsa.gov/publications/publi00004.cfm

[13] B. Schneier, *Applied Cryptography*, P. Sutherland, Ed. Wiley, 1996.

[14] O. Goldreich, *Foundations of Cryptography*, C. University, Ed. Cambridge University Press, 2001.

[15] Common Criteria, Tech. Rep., 2007. [Online]. Available: http://www.commoncriteriaportal.org/

[16] C. Criteria, *Supporting Document Guidance – Smartcard Evaluation*, DCSSI, Ed., 2006.

[17] "Smartcard ic platform protection profile, bsi-pp-0002," Bundesamt fur Sicherheit in der Informationstechnik (Federal Office for Information Security), Tech. Rep., 2000. [Online]. Available: www.bsi.de/cc/pplist/ssvgpp01.pdf

[18] J. S. Shapiro. (2007) Understanding the windows EAL4 evaluation. [Online]. Available: http://web.archive.org/web/20060527063317/http://eros.cs.jhu.edu/~shap/NT-EAL4.html

[19] Report to the Honorable William Lacy Clay, House of Representatives, "Information assurance, national partnership offers benefits, but faces considerable challenges," United States Government Accountability Office, Tech. Rep., 2006.

[20] J. Dethloff and H. Grottrup, "Identification switch," Patent US3 678 250, 1972. [Online]. Available: http://v3.espacenet.com/textdoc?DB=EPODOC&IDX=US3678250&F=0

[21] M. Roland, "Procédé et dispositif de commande électronique," French Patent 2 266 222, 1974. [Online]. Available: http://v3.espacenet.com/textdoc?DB=EPODOC&IDX=JP51015947&F=0

[22] "Le téléphone mobile, premier pas du dveloppement?" *Le Monde*, 2008.

[23] E. Trichina, T. Korkishko, and K. H. Lee, "Small size, low power, side channel-immune AES coprocessor: Design and synthesis results," *Lecture Notes in Computer Science*, 2005. [Online]. Available: http://www.springerlink.com/

[24] P. Kocher, J. Jaffe, and B. Jun, "Introduction to differential power analysis and related attacks," Cryptography Research, San Frencisco, Tech. Rep., 1999. [Online]. Available: http://www.cryptography.com/

[25] M. Aigner and E. Oswald, "Power analysis tutorial."

[26] A. Shamir and E. Tromer, "Acoustic cryptanalysis, on nosy people and noisy machines." [Online]. Available: http://people.csail.mit.edu/tromer/acoustic

[27] V. Rijmen and J. Daemon, "Rijndael (cryptografie)," Wikipedia, Tech. Rep., 2007. [Online]. Available: http://nl.wikipedia.org/wiki/Rijndael#Mix_Column

[28] Dynamically Reconfigurable Processor, "DRP," NEC, Tech. Rep., 2004. [Online]. Available: http://www.necel.com/en/techhighlights/drp/

[29] F. Crowe, A. Daly, T. Kerins, and W. Marnane, "Single chip FPGA implementation of a cryptographic co-processor," *IEEE International Conference on Field-Programmable Technology*, 2004. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1393279

[30] N. Ferguson and B. Schneier, *Practical Cryptography*, C. A. Long, Ed. Wiley, 2003.

[31] R. Enzler, M. Platzner, C. Plessl, L. Thiele, and G. Troester, "Reconfigurable Processors for Handhelds and Wearables: Application Analysis," in *Proceedings of Reconfigurable Technology: FPGAs and Reconfigurable Processors for Computing and Communications III (ITCom 2001)*. Denver, Colorado, USA: SPIE, August 2001. [Online]. Available: http://www.ethz.ch

[32] S. Sharma and S. B. Sudarshan, "Design of an efficient architecture for advanced encryption standard algorithm using systolic structures," in *International Conference of High Performance Computing (HiPC)*, 2005. [Online]. Available: http://www.hipc.org/hipc2005/posters/systolic.pdf

[33] I. Verbauwhede, "High throughput aes architecture," U.S. Patent 7 221 763, 2007. [Online]. Available: http://v3.espacenet.com/textdoc?DB=EPODOC&IDX=US2003202658&F=0&QPN=US2003202658

[34] K. Tiri, M. Akmal, and I. Werbauwhede, "A dynamic and differential cmos logic with signal independent power consumption to withstand differential power analysis on smart cards," *IEEE 28th European Solid-State Circuit – ESSCIRC'02*, 2002.

[35] S. Moore, R. Anderson, P. Cunningham, R. Mullins, and G. Tylor, "Improving smart card security using self-timed circuits," *8th IEEE International Symposium on Asynchronous Circuits and Systems – Async'02*, 2002.

[36] E. Biham, "A fast mew des implementation in software," 1997. [Online]. Available: http://www.cs.technion.ac.il/~biham/

[37] N. A. Saqib, F. Rodríguez-Henriquez, and A. Díaz-Pérez, "A compact and efficient fpga implementation of the des algorithm," 2004.

[38] I. Ahmad and A. S. Das, "Hardware implementation analysis of sha-256 and sha-512 algorithms on fpgas," *Computers & Electrical Engineering*, vol. 31, no. 6, pp. 345–360, 2005.

[39] R. Chaves, G. Kuzmanov, L. Sousa, and S. Vassiliadis, "Improving sha-2 hardware implementations," in *CHES*, ser. Lecture Notes in Computer Science, L. Goubin and M. Matsui, Eds., vol. 4249. Springer, 2006, pp. 298–310.

[40] "Hashing algorithm generator sha-256, cadence datasheet."

[41] (2007) Arm 7 tdmi. [Online]. Available: http://www.arm.com/products/CPUs/ARM7TDMI.html

[42] P. Frison, E. Gautrin, D. Lavenier, and J. L. Scharbarg, "Réseaux systoliques spécifiques à base du processeur," Institut National de Recherche en Informatique

et en Automatique (INRIA), France, Tech. Rep., 1990. [Online]. Available: http://www.inria.fr/

[43] C. Mucci, L. Vanzolini, F. Campi, and M. Toma, "Interactive presentation: Implementation of aes/rijndael on a dynamically reconfigurable architecture," in *DATE '07: Proceedings of the conference on Design, automation and test in Europe*. San Jose, CA, USA: EDA Consortium, 2007, pp. 355–360.

[44] Atmel Secure Terminals, "AT91SO100," Atmel, Tech. Rep., 2007. [Online]. Available: http://www.atmel.com/dyn/products/product_card.asp?part_id=3810

[45] Standard AES cores, "Helion standard AES," Helion, Tech. Rep., 2007. [Online]. Available: http://heliontech.com/aes_std.htm

[46] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, "A compact rijndael hardware architecture with s-box optimization," in *ASIACRYPT '01: Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security*. London, UK: Springer-Verlag, 2001, pp. 239–254. [Online]. Available: http://www.springerlink.com/index/BC7DVD7YMADU3J8L.pdf

[47] S. Mangard, M. Aigner, and S. Dominikus, "Highly regular and scalable AES hardware architecture," *IEEE Transactions on Computers*, 2003. [Online]. Available: http://ieeexplore.ieee.org/Xplore/login.jsp?url=/iel5/10557/33406/01581498.pdf?arnumber=1581498

[48] D. Thull and R. Sannino, "Performance considerations for an embedded implementation of oma drm 2," in *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 46–51.

[49] N. Valette, L. Torres, G. Sassatelli, and F. Bancel, "Securing embedded programmable gate arrays in secure circuits," *ipdps*, vol. 0, p. 226, 2006.

[50] J. Kelsey, B. Schneier, D. Wagner, and C. Hall, "Side channel cryptanalysis of product ciphers," in *The Journal of Cryptography*, Counterpane, University of Berkley and University of Princeton, 2000. [Online]. Available: http://www.schneier.com/paper-side-channel.html

[51] T. Lash, "A study of power analysis and the AES (recommendation for designing power analysis resistant attack)," in *MS Scholarly Paper*, George Mason University, 2002.

[52] C. Huynh Van Thieng, "Amélioration des performances d'un crypto-processeur," diploma thesis, 2008.

[53] R. B. Lee, Z. Shi, and Y. L. Yin, "Cryptographic properties and implementation complexity of different permutation operations," Princeton University, Tech. Rep., 2002.

[54] NEC, "Dynamically reconfigurable processor (drp)," Tech. Rep., 2004. [Online]. Available: http://www.necel.com/en/techhighlights/drp/

[55] Tux the penguin. [Online]. Available: http://upload.wikimedia.org/wikipedia/commons/a/af/Tux.png

[56] Atmel, "AVR processor 8 bits," Tech. Rep., 2006. [Online]. Available: http://www.atmel.com/products/AVR/

[57] (2007) Arm 9 embedded core. [Online]. Available: http://www.arm.com/products/CPUs/families/ARM9Family.html