A Multi-Core AES Cryptoprocessor for Multi-Channel SDR

Michael Grand[1], Lilian Bossuet[1], Bertrand Le Gal[1], Dominique Dallet[1] and Guy Goniat[2]
[1] Laboratoire IMS, University of Bordeaux, first_name.last_name@ims-bordeaux.fr
[2] Laboratoire LabSTICC, University of Bretagne Sud, guy.gogniat@univ-ubs.fr

Abstract

To answer to the security needs of modern military communication systems, this paper presents a multi-core architecture for cryptographic processors. This architecture is especially designed for use in multi-channel Software Defined Radio devices. It provides support for GCM, CCM, CTR and other block cipher modes applied to AES algorithm. It can reach a maximum throughput of nearly 2 Gbps.

Keywords: Software Defined Radio, Cryptoprocessor, Multi-core.

1 Introduction

Modern wireless devices communicate through GSM, UMTS, Bluetooth and WIFI. Moreover, communication standards evolve continually. As a consequence, flexibility and interoperability are major concerns when designing a communication device. The *Software Defined Radio* (SDR) is a part of the answer to this problem. However, the use of software components as signal processing components tends to limit the maximum throughput and in some cases hardware accelerators are required to meet performance constraints. For military context, communication systems have to be secure and red/black isolation concept is used to achieve this goal. In such architecture a *Cryptographic Sub System* is used as secure boundary between red and black radio parts.

However, cryptographic algorithms need a lot of mathematical computations. General Purpose Processors (GPP) and Digital Signal Processors (DSP) embedded in conventional SDR are not especially fitted to execute them. Then, dedicated hardware like cryptoprocessors must be used. Commonly, cryptoprocessors are based on a GPP linked to one or more algorithm specific cryptographic cores. Each cryptographic core is algorithm dependant and resources cannot be easily shared between them. In this paper, we describe a compact multi-core *Advanced Encryption Standard* (AES) cryptoprocessor which provides several block cipher operation modes and a maximum throughput around 2 Gbps.

Contribution background is presented in the first part of this paper. In a second part, the multi-channel issue on cryptoprocessor is introduced. The following part deals with the AES algorithms and its operation modes. Then our cryptoprocessor architecture is detailed. To conclude, results and future works are presented.

2 Contribution Background

2.1 The Software Communication Architecture Framework

In 1997, the U.S. government launched the *Joint Tactical Radio Software Program* (JTRS). The main goal of this program was to standardize new software architecture for SDR in order to improve software component portability. This architecture, named *Software Communication Architecture* (SCA) [1], is an open set of object interfaces compliant with the *Common Object Request Broker Architecture* (CORBA) which is software architecture for distributed computing. SCA security concerns are handled by the *Security Supplement to the SCA* provided by the JTRS.

Secure SDR are divided in at least two areas, the red side of the radio and the black side of the radio. The red side processes classified data, while the black side processes unclassified or encrypted data. Data crossing the borderline between the red and black sides (and vice versa) are processed by the *Cryptographic Sub System* (CSS).

2.2 The SCA Cryptographic Sub System

This contribution completes a previous work [2] on an open *Crypto Sub System* targeting FPGA device. Previous paper describes the CSS as a secure gateway which provides data encryption and data authentication. It also provides bypass mechanisms which permit to transmit plain text data between red and black sides. These data are carefully checked by the CSS in order to prevent leakage of sensitive data. Figure 1 illustrates the link between the CSS and the radio. CSS, illustrated in Figure 2, is divided in two blocks, a *Control Block* and a *Communication Block*. The Control Block manages the CSS and its configuration. This block receives order from the radio through red/black control I/O. Keys and certificates can be loaded into the CSS using the fill I/O. The Communication Block processes data passing through the CSS. Data can be encrypted, authenticated and filtered. It also provides cryptographic services (e.g. file encryption for data storage) for red and black radio parts.

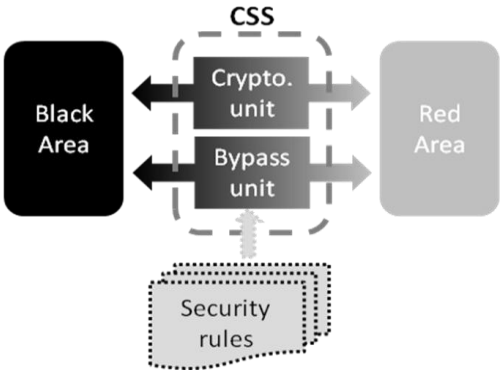

Figure 1 : CSS as Red/Black gateway

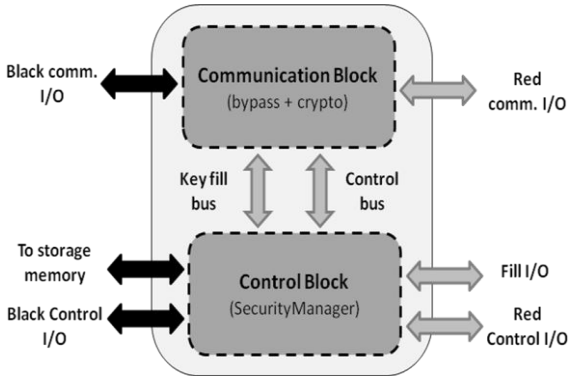

Figure 2 : CSS General Architecture

Figure 3 shows the architecture of the Communication Block. The 32 bits GPP provides IO interfaces, packet packing and unpacking and packet filtering. Cryptographic services are provided by the *Dynamically Reconfigurable Cryptographic Accelerator* (DRCA) which is described in the next part of this paper. Messages are received trough the red and black communication buses, then they are sent to the

DRCA. A channel ID is embedded in each message. It enables the DRCA to apply the right cryptographic protocol on the data stream.

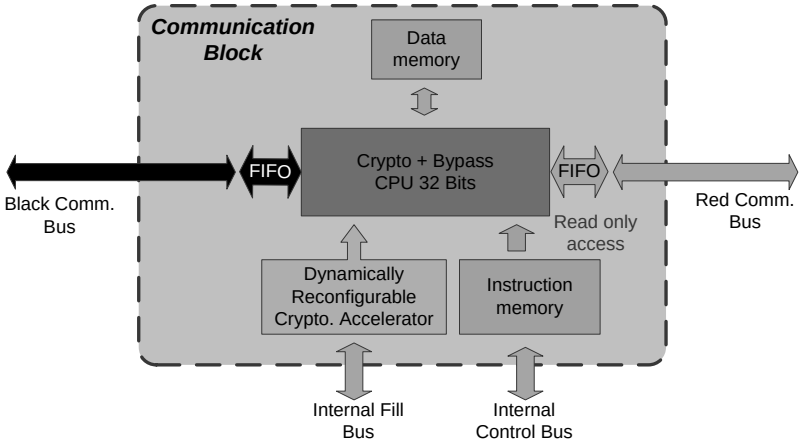

Figure 3 : CSS Communication Block

2.3 The Dynamically Reconfigurable Cryptographic Accelerator

Because SDR devices must be as flexible as possible, FPGA platform becomes an interesting way of implementation. Our CSS architecture targets reconfigurable devices such as SRAM FPGA. Moreover, such FPGA can be partially reconfigured for few years [17]. Partial reconfiguration enables designers to specify a piece of component which will be reconfigured according to radio needs. In consequence, several IP cores may share same hardware resources like several softwares sharing the same processor. While improving flexibility, this approach also reduces the needed silicon area. DRCA is the only area which is defined as partially reconfigurable in the CSS.

The multi-core cryptoprocessor described in this work may be configured into this DRCA. In the next step of our work, the cryptoprocessor data path itself will be dynamically reconfigurable. By this way, the cryptoprocessor will not be limited to the AES algorithm and reconfiguration times of the DRCA will decrease. The following part of this paper describes AES modes implemented in this cryptoprocessor.

3 The AES Algorithm and Block Cipher Modes

3.1 The Advanced Encryption Standard

The AES standard has been chosen by the NIST in 2000 to replace the old *Data Encryption Standard* (DES). Based on the Rijndael algorithm, AES is a block cipher algorithm with 128-bit block size. It allows the use of 128, 192 and 256-bit encryption keys. According to the key size, AES is composed by 10, 12 or 14 almost identical rounds. Each round is divided in several linear and non-linear transformations. More details about the AES open standard are available in the FIPS-197 publication [3].

3.2 The Counter Mode with CBC-MAC (CCM)

CCM mode [4] is one of the block cipher mode available on the NIST web site. It is used in the IEEE 802.11i standard. This mode provides both confidentiality and authentication. It is based on the *Cipher Block Chaining-MAC* (CBC) and the *Counter Mode* (CTR) modes. Figure 4 details the CBC-MAC processing mode which provides

authentication. Output of CBC-MAC is a 128 bits vector which most significant bits are used as an authentication tag. The CBC-MAC *Initialization Vector* (IV) is null when used in CCM mode. CTR mode is used to provide encryption, a unique counter value is encrypted using a key K and the result is xored with the plain text to produce the cipher text. Figure 5 shows the CTR mode process. CCM mode does not allow stream encryption due to data dependencies in CBC-MAC mode.

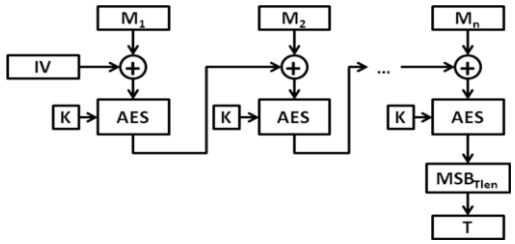

Figure 4 : CBC-MAC mode

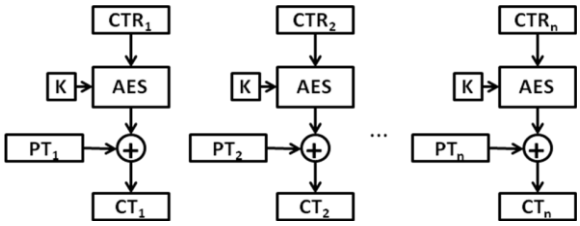

Figure 5 : CTR mode

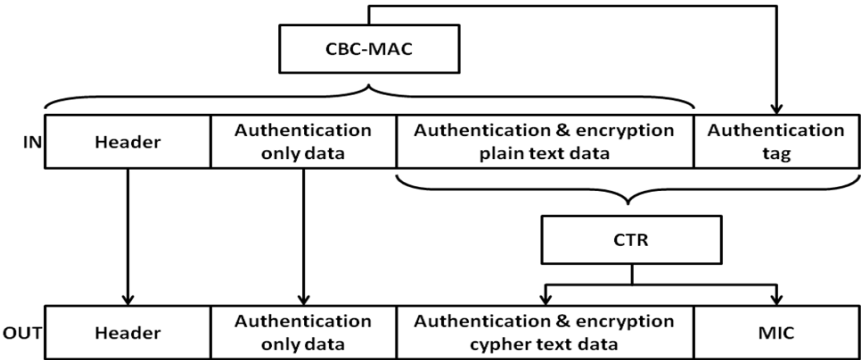

Figure 6 : CCM encryption mode

Figure 6 shows the CCM mode overall process. Incoming data are divided into 128-bit blocks. Size of each data field is used to generate the 128-bit message header defined by the CCM standard. In addition, the *Message Integrity Code* (MIC) is calculated xoring the authentication tag and AES(CTR_1, K) (see Figure 5). Both authentication and encryption/decryption can be computed in parallel.

3.3 The Galois/Counter Mode

GCM mode [5] was developed to overcome CCM mode drawbacks. It does not suffer data dependencies, hence, GCM mode permits to achieve better throughput using unrolled AES cores. It uses the CTR mode for encryption and a multiplication in the Galois field GF(2^{128}) with the following polynomial reduction $x^{129}+x^7+x^2+x+1$.

Figure 7 shows the GCM overall encryption process. IV is coded on 96 bits and is padded with thirty one 0 and one 1 to form J_0. GCTR corresponds to CTR mode, while GHASH corresponds to the following algorithm:

$$Y_{i+1} = (Y_i \text{ xor } X_{i+1})*H$$

Where Y_i is the i^{th} 128-bit hash, X_i a 128-bit block which must be hashed and $Y_{-1}=0$. A and C fields are padded with 0 to make their size be a multiple of 128.

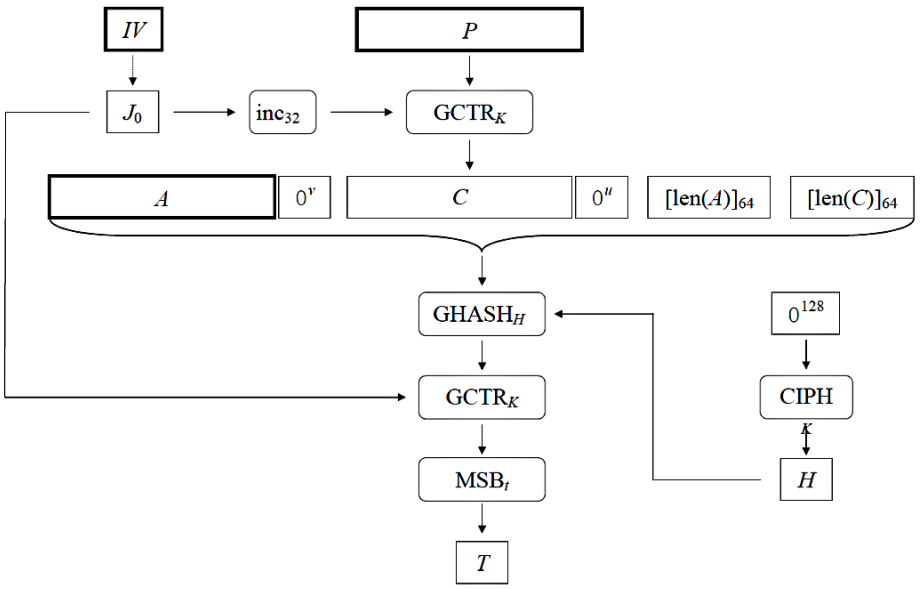

Figure 7 : GCM mode [5]

4 Multi-Channel on Cryptoprocessors

4.1 Multi-Channel Issue

A SCA compliant radio may handle one or more communication channels at the same time according to manufacturer specifications. In consequence, our cryptoprocessor must be able to process concurrent channels in order to meet throughput constraints for each channel. A channel conveys packets which are tagged with a channel ID and a packet number. When a packet is processed by the cryptoprocessor, channel configuration (keys, algorithms) can be found using the channel ID. Basically, there are two ways to implement concurrent channels on a cryptoprocessor. All channels may be processed by the same core or they may be processed by several cores. The followings two parts describe these approaches.

4.2 Mono-Core Approach

This approach uses the *channel interleaving* mechanism which means that packets from different channels are processed in a sequential way by the same processor. Therefore, total cryptoprocessor throughput is shared by channels. Commonly, channel interleaving is used on unrolled cores. In such cores, AES loop is completely unrolled and pipeline registers are inserted between each round. Consequently, a cipher text is generated at each clock cycle for a 128-bit wide data path core. By this way, throughput of tens of gigabits per second can be reached. However, this kind of architecture has several drawbacks.

Because each packet uses a different key, delays are added between packet processing. These delays may correspond to key expansion steps or pipeline flushing. In addition, due to technical considerations, AES full enrolling is used in most of cases, leading to high resource consumption. To finish, data dependencies in some block cipher operation modes prevent the use of unrolled core. GCM mode is especially designed to take profit of unrolled cores. There are several works dealing with hardware

implementations of AES-GCM core. [6], [7], [8] are few examples of such implementations.

As it is previously explained, unrolled cores are not well suited for iterative mode like CBC-MAC. Iterative cores are better suited because they provide the maximum throughput for the minimum cost. A 128-bit wide iterative core can provide an encrypted text block every ten cycles. Also, CTR mode and CBC-MAC mode of the CCM mode can be computed in parallel, enhancing maximum throughput by a factor two. [9], [10], [16] are illustrations of CCM implementation using two AES cores. They can reach throughput more than 600 Mbps. However, maximum throughput is still limited if compared to unrolled core maximum throughput. The following part explains why multi-core architectures may provide a good tradeoff between performance and resource utilization whatever cipher mode is used.

4.3 Multi-Core Approach

A multi-core cryptoprocessor is composed by several cryptographic cores and at least one controller which act as a task scheduler. Multi-core approach for cryptoprocessors has the same advantages and drawbacks as multi-core general purpose processor approach.

There are, already, few cryptoprocessor implementations which use multi-core approach. Cryptonite [12] is a programmable cryptoprocessor built around two *clusters*. Each cluster provides cryptographic functions useful for block cipher algorithms. This implementation targets ASIC platform and reaches a throughput of 2.25 Gbps for AES-ECB algorithm. Celator [13] is composed by several *Processing Elements* which are connected together to form a grid like a block cipher state. According to PE configuration, cryptographic functions are applied on the grid at each clock cycle. Celator is able to compute AES, DES or SHA algorithms, providing for example a throughput of 47 Mbps when computing AES-CBC algorithm. To finish, Cryptomaniac [14] is a multi-core processor intended to enforce communication security. Its architecture is based on a scheduler core which dispatches incoming packet to several simple cryptoprocessors. It achieves throughput of 512 Mbps on ASIC platform at 360 MHz.

In contrast, our architecture targets FPGA platform. It is built around one task scheduler and four AES cores which handle several block cipher modes. AES block cipher can be easily dissociated from other functions during placement step. In consequence, AES core could be replaced by another 128-bit block cipher such as Twofish using partial reconfiguration functionality of modern FPGAs. The following part describes our multi-core architecture.

5 Multi Core Architecture

5.1 General Architecture

Our multi core cryptoprocessor embeds four single core cryptoprocessors and all data are exchanged on a 32-bit data path, Figure 8 details its architecture. Each fifo port is linked to the data port through the *Crossbar*, control port is linked to the *Task Scheduler* (TS). The TS is used to dispatch cryptographic tasks on all cores and embeds a small RAM to store channel configurations. It manages the *Key Scheduler* and core

computations and selects the right configuration for the *Crossbar:* an input fifo of one core can be selected while an output fifo of another core is already selected.

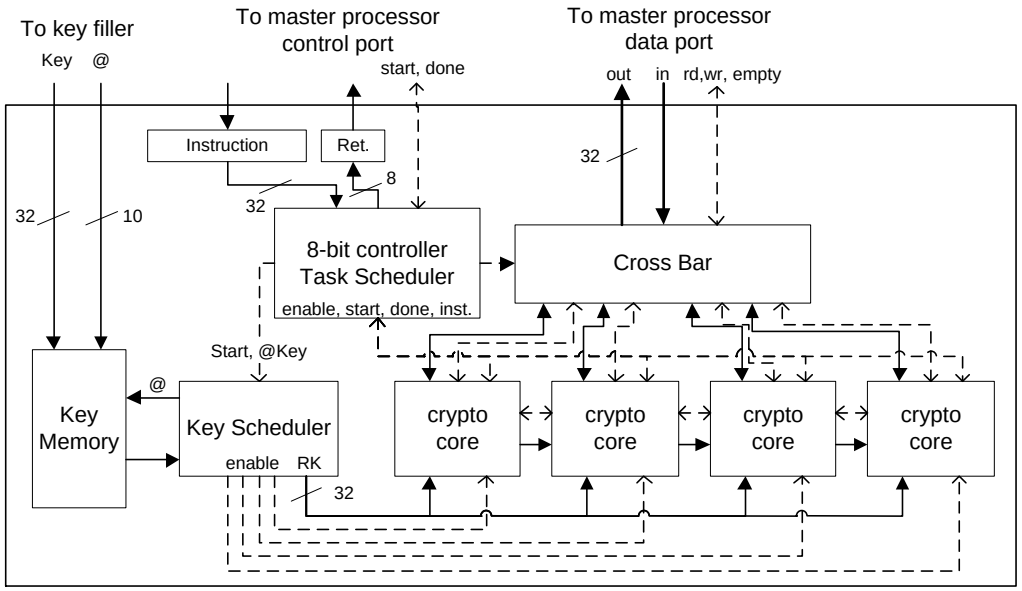

Figure 8 : Multi Core Architecture

5.2 Multi Core Control Protocol

The master processor sends instructions to the task scheduler through the control port, several instructions are available:

- *OPEN Channel_ID, Key_ID, Algorithm, [Tag size]*: This command is used to open a new channel on the cryptoprocessor.
- *CLOSE Channel_ID:* It is used to close a channel.
- *ENCRYPT Channel_ID, Data_Size:* It requests resources for encryption. TS tries to find some available resources before sending a *done* signal.
- *DECRYPT Channel_ID, Data_Size:* It requests resources for decryption and authentication. TS tries to find some available resources before sending a *done* signal.
- *TRANSFER_DONE* : This command informs the TS that the master processor has uploaded/downloaded all data to/from a fifo.

These instructions return an OK flag or some error flags according to the request processing result. DECRYPT instruction returns an *AUTH_FAIL* flag when message authentication fails. Other error flags are not described here.

5.3 Operation Modes

The multi-core cryptoprocessor can execute GCM, CCM, CTR and any other block cipher mode which only uses AES encryption and XOR operators. Available rules for operation modes are described below:

- Packets from a same channel can be concurrently processed on different cores.
- Packets from different channels can be concurrently processed on different cores.
- Any single packet can be processed one any single core.

- Any single CCM packet can be processed on two cores.

Right operation mode is selected by the TS according to the requested channel algorithm and available resources. To be used in an efficient way, smartly designed task scheduler software must be implemented. Also, a channel priority mechanism may be implemented in order to provide a wider bandwidth for specific channels.

6 Cryptographic Core Architecture

6.1 General Architecture

Figure 9 shows the architecture of a single cryptographic core, which communicates with the crossbar and other cores through two fifos (512x32 bits) and one shift registers (SR) (4x32 bits). Cryptographic functions are implemented in the cryptographic coprocessor and round keys are pre-computed and stored in the key cache. These functions can be used through the coprocessor *Instruction Set Architecture* (ISA). An 8-bits controller generates the instruction flow according to the selected cryptographic algorithm.

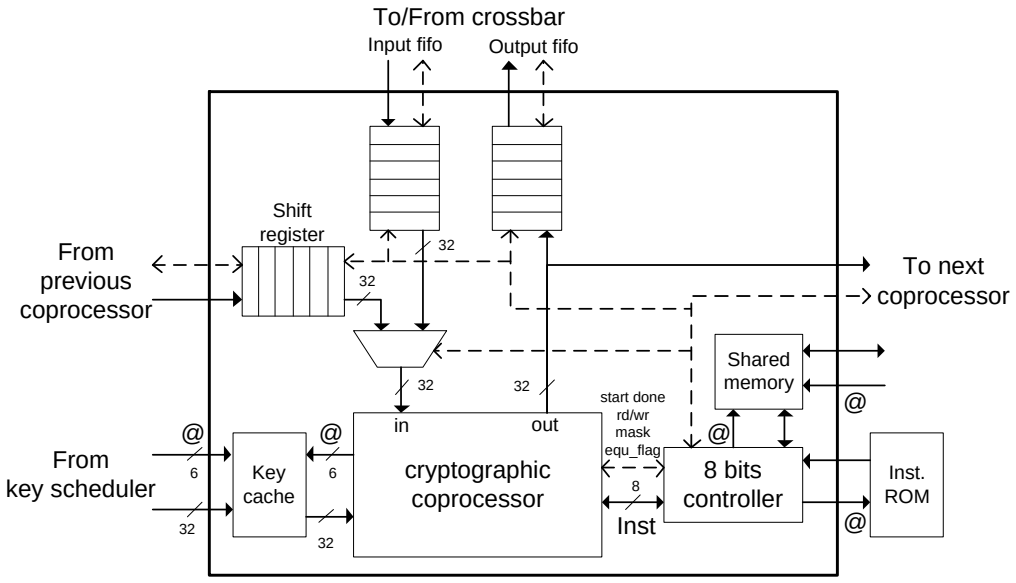

Figure 9 : Cryptoprocessor architecture

Incoming packets are processed in the following way. In a first time, task scheduler sends an instruction to the 8-bit controller through the shared memory and triggers a start signal. Then, controller starts pre-computations needed for the selected algorithm and loads data from input fifo once there are available. Data are processed by blocks of 128 bits and filled into the output fifo. When all data have been processed, the controller sends a done signal to the task scheduler. In order to protect master processor from attacks, output fifo is re-initialized if plain text data does not match the authentication tag. By this way, corrupted data are never seen by the master processor.

6.2 Cryptographic Coprocessor Architecture

The coprocessor, described in Figure 10, provides cryptographic functionalities and works sequentially on 128-bit words over a 32-bit data path. The coprocessor embeds a 32-bit AES core, a GHASH core and some arithmetic and logic operators

which are controlled by the decoder. They work on a dual port bank register which can store up to four 128-bit words.

When the 8-bit controller executes a coprocessor instruction, it writes on the instruction port and set the "*S"* register. Then, the combinatorial decoder enables the right processing core and selects the right address for the multiplexer. Processing core enabling is made thanks to *"start"* signals. There is one start signal per processing core functionality (e.g. *ghash_start_init* and *ghash_start_finalize),* they enable the right state of state machines embedded into each processing core (AES, GHASH, inc, xor/equ). When a processing core acknowledges a start signal, it triggers an *ack* signal in order to reset the "*S"* register. To finish a *done* signal is sent when processing is done. Each processing core also triggers *Rd* and *Wr* signals to read or write data into the bank register. When *Rd* or *Wr* signals are asserted the 2-bit counter is enabled, by this way one of the four 128-bit word of the bank register can be read or written.

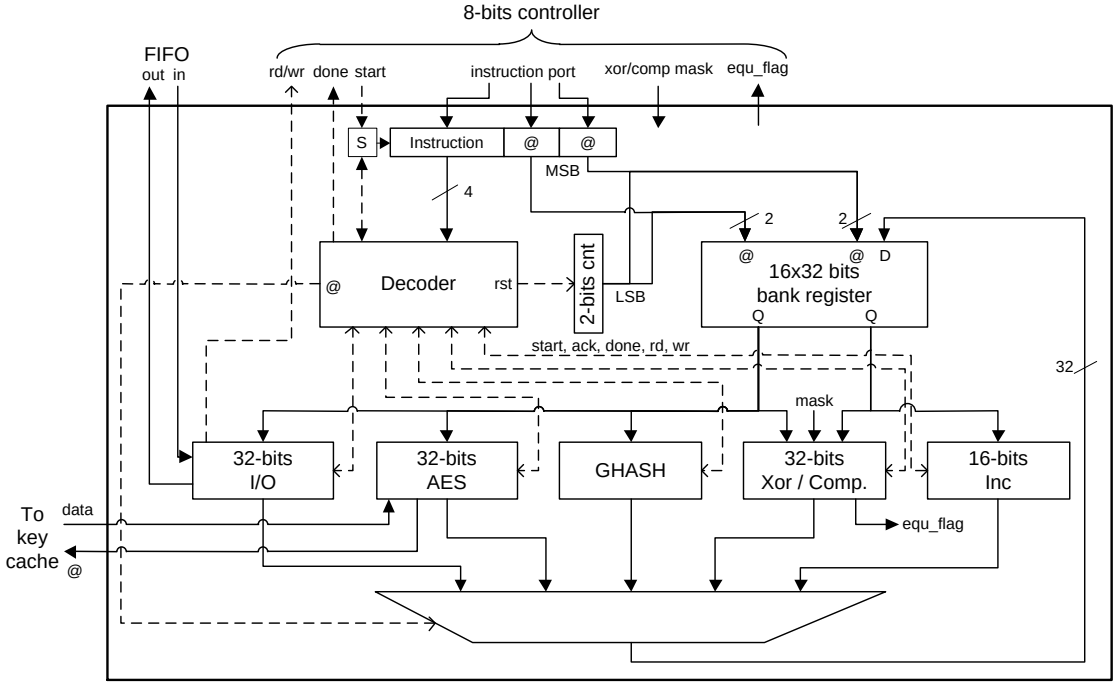

Figure 10 : Coprocessor Architecture

6.3 AES and GHASH cores

AES and GHASH cores are both compact cores presented in some previously published works. The AES core was developed using P. Chodowiec and K. Gaj works [15] on a compact AES core for FPGA devices. Because CCM and GCM modes only use encryption mode, the AES decryption algorithm was not implemented. AES core is implemented using an iterative architecture and *SubBytes* transformation [3] is based on look up tables. Iterative architecture implies that AES computation time is key size dependant. Computation of one 128-bit block takes 44, 52 or 60 cycles when using respectively 128-bit, 192-bit or 256-bit key sizes. GHASH core is based on the digit-serial architecture described in [6]. Digit-serial multiplication is made using 3-bit digits and takes 43 clock cycles to be computed.

It is noticeable that any 128-bit block cipher can be used instead of AES cipher. In consequence partial reconfiguration may be used to dynamically reconfigure the

coprocessor block cipher. In this case, partial reconfiguration may again improve flexibility of our core.

6.4 Coprocessor Instruction Set Architecture

The coprocessor is controlled thanks to a specific ISA. Each instruction is executed in exactly four cycles. ISA instructions are detailed below:

- *LOAD @A*: Loads a 128-bit word into the A register.
- *WRITE @A*: Writes a 128-bit word from the A register to the out port.
- *LOADH @A*: Loads the computed H constant into the GHASH core.
- *SGFM @A*: Computes one iteration of the GHASH algorithm.
- *FGFM @A*: Stores the result of the GHASH algorithm into the A register.
- *SAES @A*: Encrypts the value stored in the A register.
- *FAES @A*: Stores the results of the SAES computation into the A register.
- *INC @A*: Increments the 16 less significant bits of the A register.
- *XOR @A, @B*: Computes *B = (A XOR B) AND mask.*
- *EQU @A, @B*: Sets the *equ_flag* to 1 if A = B and 0 else.

It must be noticed that SGFM and SAES are non-blocking instructions, while FAES and FGFM are synchronizing instructions. In consequence encryption and GHASH computation can be made at the same time. When the result of one of these two operations is needed FGFM and FAES are used to wait for computation ending and result storing. Synchronization mechanism is obtained with the use of the *ack* signals and the "*S"* register. For example, when a FAES instruction is executed while the AES core is busy, "*S"* register is still set to 1 until core comes into an idle state and *ack* signal is triggered.

6.5 8-bit Controller Architecture

Our processor handles several block cipher operation modes leading to complex control state machines. A more flexible approach is to use a general purpose controller to generate instruction flow executed by the coprocessor. Use of such architecture allows us to simplify execution of loop conditions used for packet encryption/decryption. Because this controller does not perform heavy computations a simple processor may be used. In our case, we use an 8-bit controller providing a simple ISA. This processor communicates with the task scheduler to receive orders and some algorithm parameters like packet size, then instruction flow is generated and to finish instruction results are sent back to the task scheduler.

To the prototyping step, a modified 8-bit Xilinx PicoBlaze controller [16] has been used. Two instructions are needed to fetch and start executing a coprocessor instruction. A custom HALT instruction is used to put the controller into a sleep mode when a *start* signal is sent. The controller wakes up when coprocessor triggers the *done* signal.

7 Results

Our multi-core cryptoprocessor has been developed in VHDL and synthesized on a Xilinx Virtex 4 SX25-11 FPGA. On this device, our cryptoprocessor is able to reach a frequency of 190 Mhz after place and route. After mapping, 3984 slices and 26 BRAMs are used. The following table summarizes resource consumption of cryptoprocessor

main parts. Actually, due to slice packing, each core uses fewer slices than what is written in the table when they are put together.

Core	LUTs	Flip Flop registers	Slices	18kb BRAMs
AES block cipher	250	46	156	2
GHASH	637	395	324	0
Coprocessor	1150	463	646	2
One crypto core	1506	738	911	5
Key scheduler	215	83	131	2

Tableau 1 : Resource Consumption

Overall throughput is limited by the AES algorithm execution time which depends on AES key size. In addition, SAES and FAES instructions take two clock cycles to be fetched into the coprocessor. Tableau 2 details critical execution path for GCM and CCM modes. Where T_{XOR} and T_{FAES} are equal to six clock cycles and T_{SAES} is equal to 42, 50 or 58 cycles for 128-bit, 192-bit or 256-bit keys. Cryptoprocessor latency is mainly due to key scheduling before packet processing and algorithm initializations. However, our core is built to handle packet size up to 2KB, then latency time introduced by key scheduling and others computations is negligible compared to encryption time. Tableau 3 summarizes average maximum throughputs according to channel configuration.

Algorithm	Critical execution path (clock cycles)	
	1 core implementation	2 cores implementation
AES-GCM/CTR	$T_{SAES} + T_{FAES}$	---
AES-CCM	$2*(T_{SAES}+T_{FAES}+T_{XOR})$	$T_{SAES}+T_{FAES}+T_{XOR}$

Tableau 2 : Critical Execution Path

Key size (bit)	Average maximum throughputs (Mbps)				
	AES-GCM/CTR		AES-CCM		
	1 core	4x1 cores	1 core	2 cores	2x2 cores
128	506	2026	225	450	900
192	434	1737	196	405	810
256	380	1520	173	347	694

Tableau 3 : Average Maximum Throughput at 190 MHz

8 Conclusion and future works

This paper follows a previous work on the Software Communication Architecture and presents a multi-core cryptoprocessor which may be used in SDR device. This work shows that multi-core architectures provide a good tradeoff between flexibility, performances and resource consumption. This multi-core architecture can reach a maximal throughput of 2 Gbps using parallel packet processing. While single cores can be used in an independent way, they can also be used in a cooperative way when an operation mode uses several AES primitives at the same time.

Moreover, multi-core architecture enables implementation of efficient partial reconfiguration mechanism. Instead of reconfiguring a large unrolled AES core, multi-core architectures enable reconfiguration of a small single core. In our case, just the very small AES processing core (around 150 slices on V4 devices) may be reconfigured

and replaced by another block cipher core as block cipher operation modes are not limited to AES algorithm.

Future works will deal with implementation of a more generic multi-core cryptoprocessor which may implement cryptographic hash algorithms or some Galois field multiplier useful for computations on elliptic curves.

References

[1] JTRS, *"Software Communication Architecture Specification"*, V2.2.2, 2006

[2] M. Grand, L. Bossuet, G. Gogniat, B. Le Gal, D. Dallet, *"A Reconfigurable Crypto Sub System for the Software Communication Architecture"*, in Proceedings of MILCOM 2009

[3] NIST, "*FIPS-197",* 2001

[4] NIST, *"Special Publication 800-38C",* 2007

[5] NIST, *"Special Publication 800-38D",* 2007

[6] S. Lemsitzer, J. Wolkerstorfer, N. Felber, M. Braendi, *"Multi-gigabit GCM-AES Architecture Optimized for FPGAs"*, in Proceedings of CHES'07, Springer-Verlag, 2007, 227-238

[7] S. Akashi, S. Takeshi, A. Takafumi, *"High-Speed Pipelined Hardware Architecture for Galois Counter Mode"*, Lecture Notes in Computer Science, Volume 4479/2007, Springer Berlin / Heidelberg, 2007, 118-129

[8] S. Wang, *"An Architecture for the AES-GCM Security Standard"*, Master Thesis, University of Waterloo, 2006

[9] A. Arshad, I. Nassar, *"An FPGA-based AES-GCM Crypto Core For IEEE 802.11i Architecture"*, in International Journal of Network Security, Vol.5, No.2, 2007, 224-232

[10] K. Vu, D. Zier, *"FPGA Implementation AES for CCM Mode Encryption Using Xilinx Spartan-II"*, ECE 679, Advanced Cryptography, Oregon State University, 2003

[11] E. Lopez-Trejo, F. Rodriguez-Henriquez, A. Diaz-Perez, *" An Efficient FPGA Implementation of CCM mode Using AES ",* ICISC 2005, Lecture Notes in Computer Science, volume 3935, Springer-Verlag, 2005, 208-215

[12] R. Buchty, N. Heintze, D. Oliva, *"Cryptonite – A Programmable Processor Architecture for High-Bandwidth Applications"*, ARCS 2004, LNCS, Volume 2981/2004, Springer Berlin / Heidelberg, 2004, 184-198

[13] D. Fronte, A. Perez, E. Payrat, *"Celator, A Multi-algorithm Cryptographic Co-processor"*, in Proceedings of ReConFig'08, 2008

[14] L. Wu, C. Weaver, T. Austin, *"CryptoManiac: a Fast Flexible Architecture for Secure Communication"*, in Proceedings of ISCA'01 , 2001

[15] P. Chodowiec, K. Gaj, *"Very Compact FPGA Implementation of AES Algorithm"*, in Proceedings of CHES 2003, Springer-Verlag, 2003, 319-333

[16] Xilinx Inc., *"PicoBlaze 8-bit Embedded Microcontroller User Guide"*, http://www.xilinx.com

[17] Xilinx Inc., *"Partial reconfiguration web site"*, http://www.xilinx.com/tools/partial-reconfiguration.htm