A Multi-Core AES Cryptoprocessor for Multi-Channel SDR

Michael Grand¹, Lilian Bossuet¹, Bertrand Le Gal¹, Dominique Dallet¹ and Guy Goniat²
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

This paper presents a multi-core architecture for cryptographic processors. This architecture is especially designed for use in multi-channel Software Defined Radio Device. It provides support for GCM, CCM, CTR and other block cipher modes applied to AES algorithm. It can reach a maximum throughput around 2 Gbps. This paper also presents how partial reconfiguration could be used to improve flexibility of multi-core cryptoprocessors.

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

1 Introduction

Modern cell phones 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.

Cryptographic protections used to secure wireless transmissions need a lot of mathematical computations and general purpose processors and DSPs embedded in conventional SDR are not especially fitted to execute them. Then, dedicated hardware like cryptoprocessors must be used. Commonly, cryptoprocessor 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 these cores. In this paper, we describe a compact multi-core 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. The security supplement does not give implementation details to developers. A secure SDR must include features like:

- **Standard cryptographic services**
- **Security policy enforcement**
- **Keys and certificates management**
- **Access control**
- **Configuration management**
- **Alarms and auditing**
- **Memory management**

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)**.

![CSS as Red/Black gateway](image1.png)

**Figure 1 : CSS as Red/Black gateway**

![CSS General Architecture](image2.png)

**Figure 2 : CSS General Architecture**

### 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 3 shows the architecture of the Communication Block. The 32 bits GPP provides I/O 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.
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 target reconfigurable devices like FPGA. Moreover, FPGA can be partially reconfigured for few years. Partial reconfiguration enables designers to specify a piece of component which will be reconfigured according 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.

The multi-core cryptoprocessor described in this paper is an example of a core which can be configured into the DRCA. In the future, the data path of the cryptoprocessor 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 next part of this article 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 DES standard. It is 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 standard are available in the FIPS-197 standard [3].

3.2 The Counter Mode with CBC-MAC (CCM)

CCM mode [4] is one of cipher block 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. CBC-MAC provides data authentication, Figure 4 details the CBC-MAC processing mode. Output of CBC-MAC is a 128 bits vector which most significant bits are used as authentication tag. 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](image)

![Figure 5: CTR mode](image)

Figure 6 shows the CCM mode overall process. Incoming data are divided in 128 bits block, size of each data field is used to generate the 128 bits message header. Header format is defined by the CCM standard. In addition, the Message Integrity Code (MIC) is calculated xorring the authentication tag and AES(CTR, K). Message is decrypted using header data and CTR mode, then message can be authenticated. Both authentication and encryption/decryption can be made 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. GCM mode uses the CTR mode for encryption and multiplication in the Galois field GF(2^{128}) with the following polynomial reduction \( x^{128}+x^7+x^2+x+1 \).

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:

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

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.
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. Each packet is tagged with a channel ID and a packet number and a cryptographic key and a cryptographic protocol are attributed to channels. When a packet is processed by the cryptoprocessor, processing parameters can be found using the channel ID. Basically, there is 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.

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 unrolled 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 gigabit per second can be reached and channel interleaving becomes possible. 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, partial unrolling of the AES loop is not easy. Full enrolling is used in most of cases, leading to high resource consumption. To finish, data dependencies in some block cipher operation modes prevent use of unrolled core. GCM mode is especially designed to take profit of unrolled cores. There are several papers dealing with implementations of AES-GCM core on ASIC or FPGA platform. [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 one. Iterative cores are better suited for iterative modes, 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 when a 128 bits key is used. 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 still limited if compared to unrolled core maximum throughput.

The next part explains why multi-core architectures may provide us a good tradeoff between performance and resource utilization whatever cipher mode is used.

4.3 Multi-Core Approach

A multi-core cryptoprocessor is a hardware core 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 reach a throughput of 2.25 Gbps for AES-ECB algorithm. Celator [13] is composed by several Processing Elements which are put 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 target FPGA platform. It is built around one task scheduler and four AES cores which handle several block cipher mode. 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 next part of this paper describes architecture of one AES cores and then multi-core architecture is detailed.

5 Single-core Architecture

5.1 General Architecture

Figure 8 shows architecture of a single cryptographic core. This core communicates with the master processor and others core through fifos (512x32 bits) and shift registers (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.
Incoming packets are processed in the following way. In a first time, task scheduler sends an instruction to 8-bits 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.

### 5.2 Cryptographic Coprocessor Architecture

The coprocessor, described in Figure 9, 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 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 on start signal per processing core functionality (e.g. ghash_start_init and ghash_start_finalize), it enable the right state of state machines embedded into each processing core. 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 can also trigger 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.
5.3 AES and GHASH cores

AES and GHASH cores are both compact cores presented in some previously published papers. The AES core was developed using P. Chodowiec and K. Gaj paper [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 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 and 60 cycles when using respectively 128-bit, 192-bit and 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. Hence, computation time is equal to 43 clock cycles.

It is noticeable that any 128-bit block cipher can be used instead AES core. In consequence partial reconfiguration may be used on this core to replace it by another cryptographic core. In this case, partial reconfiguration may again improve flexibility of our core.

5.4 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**: Allows loading of a 128-bit word into the A register.
- **WRITE @A**: Allows writing of a 128-bit word from the A register on 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 \( B = (A \text{ XOR } B) \text{ AND mask} \).
• *EQU @A, @B*: Set 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 instruction. 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 AES core is busy, “\( S \)” register still set to 1 until core come into an idle state and *ack* is triggered.

5.5 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 used 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.

At this moment, a modified 8-bit Xilinx PicoBlaze controller [16] is used. All instructions are executed in two clock cycles. Two instructions are needed to fetch and start executing a coprocessor instruction. Firstly, instructions are loaded into a register then, theyr are dumped on the output port. A custom HALT instruction is used to put the controller into a sleep mode when a start signal is sent, controller wakes up when coprocessor triggered the done signal.

Next part explains how single cores are used in a parallel way to make a multi-core cryptoprocessor.

6 Multi Core Architecture

6.1 General Architecture

Our multi core cryptoprocessor embeds four single core cryptoprocessors and all data are exchanged on a 32-bit data path. Each fifo port is linked to the data port through the *Cross Bar*, control port is linked to the *Task Scheduler* (TS). The TS is used to dispatch cryptographic tasks on all cores, it embeds a small RAM to store channel configurations. It manages the *Key Scheduler* and core computations and selects the right configuration for the *Cross Bar*: an input fifo of one core can be select while an output fifo of another core is already selected. Figure 10 details the multi-core architecture.
6.2 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 to send a done signal.
- **DECRYPT Channel_ID, Data_Size**: It requests resources for decryption and authentication. TS tries to find some available resources before to send 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 flag according to the request processing result. DECRYPT instruction returns an AUTH_FAIL flag when message authentication fails. Other error flags are not described here.

6.3 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, a smartly designed task scheduler software must be implemented in the controller core. Also, a channel priority mechanism may be implemented in order to provide a wider bandwidth for specific channels.

7 Results

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 also can 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, to go beyond just the very small AES processing core (about 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


