Asynchronous AES Crypto-Processor Including Secured and Optimized Blocks

F. Bouesse[1], M. Renaudin[2], and F. Germain[3]

[1,2] F. Bouesse, and M. Renaudin, TIMA laboratory, 26 Av. Félix Viallet, 38031 Grenoble, France,
e-mail : fraidy.bouesse@imag.fr
[3] F. Germain, SGDN/DCISS, 51 bd. De la Tour Maubourg, 75700 Paris, France,
e-mail : fabien.germain@sgdn.pm.gouv.fr

Abstract—**This paper presents the first study of an asynchronous AES architecture compliant with the NIST standard. It exploits the fundamental properties of quasi delay insensitive asynchronous circuits. First, 1 to N encoding is extensively used in order to minimize hardware cost, thus optimizing area and speed. Most importantly, it is shown how the quasi delay insensitive logic style gives the opportunity to design balanced architectures, particularly well suited to improve differential power analysis resistance. Indeed, the proposed design methodology enables the generation of logic circuits which always involve a constant number of logical transitions, independently of data values processed by the circuit. Based on a 32-bit data-path, a balanced and optimized QDI asynchronous architecture of the AES is described. In addition, several architecture trade-offs are considered, and their area and speed estimated. Simulation results show that with the proposed design approach, throughputs ranging from 36 Mbit/s to more than 569 Mbit/s can be achieved, well suited to target smart-card applications.**

Key words—**advanced encryption standard (AES), differential power analysis (DPA), smart card, asynchronous circuits, quasi delay insensitive circuits (QDI).**

I. INTRODUCTION AND MOTIVATIONS

The criteria used by the NIST to evaluate and select RIJNDAEL as the new advanced encryption standard algorithm (AES) included security, flexibility and both hardware and software implementation efficiency. From the implementation point of view, two extreme architectures were studied to evaluate the candidate algorithms. One based on an iterative formulation of the algorithm that minimizes the implementation cost, and the other one fully parallel that maximizes the computation throughput [11]. Representative standard architectures were designed using FPGAs, for the purpose of comparing between AES finalist candidates [12, 13].

Since its adoption, several AES hardware implementations were proposed and published, all of them focused on high throughputs. Among them, Henry Kuo presented in [1] an optimized architecture reaching 1.82 Gbits/sec, using a 0.18 μm CMOS process and requiring 173 Kgates. The data path, 256-bit wide, is ciphering text using data and keys of length 128, 192 or 256 bits. A.K. Lutz proposed in [6] a 2-Gbits/s architecture based on two parallel data paths processing texts of 128 bits using a 128-bit key (ciphering and deciphering both supported). Such architectures are not suited for low-power and low-cost applications. Moreover, from a security point of view, since the side-channel attacks were discovered, the implementations of the cryptographic algorithms are particularly vulnerable against Differential Power Analysis [22]. In fact, secrets information are removed from device by observing and monitoring the electrical activity of a device and performing advanced statistical methods. This method exploits the fact that the power consumption of a chip is correlated to the data processed. Among all this hardware countermeasures proposed for resisting DPA [23-24], asynchronous logic has been presented as a new alternative design solution. The results obtained on [4-25] by using asynchronous logic, demonstrate the increase of security.

The objective of this paper is to target smart-card applications by designing a low power/low cost and highly secured AES architecture. The properties of asynchronous technology are exploited to achieve this goal. More specifically, quasi delay insensitive (QDI) circuits using 1 of N encoding and four-phase handshake protocol are used.

Section II introduces asynchronous technology, especially N-rail quasi delay insensitive asynchronous logic. Section III then investigates various architectural choices of AES critical blocks and proposed an optimised implementation of the Sbox. Section IV describes the key scheduling block. Simulation results are reported in section V, while section VI considers alternative architectural trade-offs. Section VII concludes the paper and gives some prospects.

II. ASYNCHRONOUS LOGIC

Most integrated circuits are today synchronous, which means that they are controlled by a global clock which triggers at the same time the memorization of the complete state of the circuit.

Asynchronous circuits represent a class of circuits which are not controlled by a global clock but by the data themselves. In fact, an asynchronous circuit is composed of individual modules which communicate to each other by means of point-to-point communication channels. Therefore, a given module becomes active when it senses the presence of incoming data. It then computes them and sends the result to the output channels. Communications through channels are governed by a protocol which requires a bi-directional signaling between senders and receivers (request and acknowledge). They are called Handshaking protocols.

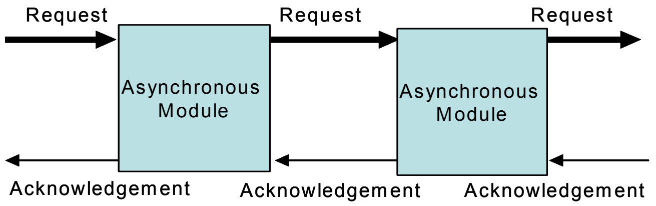


Fig. 1: Handshake based communication between modules. An asynchronous module can be of any complexity.

The communication protocol is the basis of the sequencing rules of asynchronous circuits. There are two main classes of handshaking protocols: two-phase protocol and four-phase protocols. In this work, only the four-phase protocol is considered and described. It is the most widely used when designing integrated circuits because its transistor implementation is more efficient [7].

- Four-phase protocol

This protocol requires a return to zero phase for both data/requests and acknowledges (figure 2).

- Signalling

As presented above, the implementation of a four-phase handshaking protocol requires sensing the presence of data in phase 1, and setting the acknowledgement when incoming data can be released in phase 2. It then requires sensing that data are back to invalid in phase 3, and resetting the acknowledgement in phase 4. In order to do so, dedicated logic and special encoding are necessary for sensing data validity/invalidity and for generating the acknowledgement signal. Detecting that data is valid is referred to as a request for computation. In the same manner, generating an acknowledgement means that the computation is completed and the communication channel can be released. Hence, an individual module is made of a computation unit associated to input and output channels controllers.

- Data/ Request encoding

The invalid state is encoded with data themselves. Considering that one bit has to be transferred through a channel using the four phase protocol, one has to encode three different values: invalid, valid at '1', valid at'0'. Two bits or wires (A0, A1) are then required to encode the three states. This technique is called dual-rail encoding (table1).

Table 1: Dual rail encoding of the three states required to communicate 1 bit.

Channel data	A0	A1
0	1	0
1	0	1
Invalid	0	0
Unused	1	1

This encoding is easily extended to N rails. It is called 1 of N encoding.

- Acknowledge / Completion signal generation

A very common technique used to generate the acknowledgement signal is to take advantage of the data-encoding. Let's consider a module which has been designed to process three-state encoded data and which respects the four-phase communication protocol. Such a module produces dual-rail encoded outputs which state can easily be sensed by means of a simple Nor gate as depicted in figure 4. When several bits are used, the acknowledgement signal is obtained by combining the partial acknowledgements with a rendezvous cell.

- Rendezvous cell

The design of asynchronous circuits requires a rendezvous cell which is commonly named Muller C-element [10]. The Muller C-element is used to synchronize asynchronous signals which eventually occur. In other words, this gate generates an up-transition when up-transitions occur at all the inputs, and generates a down-transition when down-transitions occur at all the inputs. The Muller C-element's truth table and symbol are given in Figure 3.

II.1 QUASI DELAY INSENSITIVE (QDI) CIRCUITS

Because of the handshake signalling used by the modules to communicate, asynchronous circuits may have a very interesting property: delay insensitivity. Delay insensitivity means that the functional correctness of the circuit does not depend on the delays of its constituents. Because delay insensitivity is not free, both in terms of hardware and latency, researchers have worked on the trade-offs between delay-insensitivity and hardware-cost or speed. In this work, QDI circuits are considered because of their potentialities in terms of energy, speed and security [19].

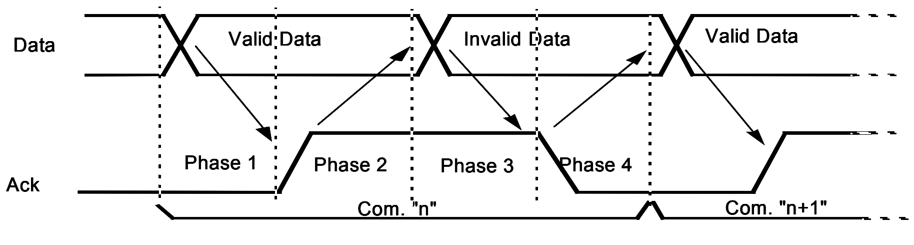
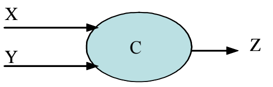


Fig. 2: Four-phase handshaking protocol.

Phase 1: Data detection (invalid Data to valid Data)
Phase 2: Acknowledgement is set to one
Phase 3: Data are re-initialized (valid Data to invalid Data, return to zero phase)
Phase 4: Acknowledgement is reset (return to zero phase)

A QDI circuit is functionally correct without any assumption on the wire and gate delays, except for some forks, called "isochronic forks" [17-18]. An isochronic fork is a fork which branches have to have equal delays to guarantee a correct behaviour of the circuit, whatever might be the delays in the other elements [17]. This asynchronous circuit style is the most robust with respect to delay variations. It has been proved that the "isochronic fork" is the weakest assumption to respect, in order to be able to design any kind of function [15-17] using single-output gates. Moreover the logic used to implement such QDI circuits requires being "hazard free" which is one of the major difficulties to cope with.

Symbol



Equation

$$Z = XY + Z(X+Y)$$

Truth table

X	Y	Z
0	0	0
0	1	no change
1	0	no change
1	1	1

Fig. 3: Muller gate or (C-element).

II.2 QDI CIRCUITS AND SECURITY

As suggested in [8], asynchronous circuits can improve chip security in many ways. In fact, the measurements performed on the Mica microcontroller [21] and reported in [20] proved that QDI circuits are indeed improving DPA resistance. Therefore, this work is focused on technics and methods to design DPA resistant chips by using QDI asynchronous circuits based on 1 of N encodings and a four-phase handshake protocol.

Although difficult, hazard free logic design is the right technology to improve hardware security against DPA, because it gives the designer the opportunity to precisely control the number of electrical transitions involved in a given computation. In fact, because logic is hazard free, spurious transitions are avoided and the number of transitions required to perform a given computation is perfectly known in advance and fixed. Moreover, it can be shown that this number of transitions can even be independent of processed data [19].

Contrary to synchronous circuits where the power consumption depends on the previous states and data values, QDI asynchronous logic using a four-phase protocol re-initializes all previously activated nodes before processing a new data [7]. Therefore, there is no effect of the previous computation on current data processing. Hence, because logical transitions are the source of the current consumed by CMOS circuits, DPA resistant circuits can theoretically be designed. However, it is well known that transistor sizes and wire lengths are also influencing the power consumption profile. Known solutions exist to tackle this problem and it is not addressed in this paper which focuses on logical level design for security. As an example, consider the xor function which is of prime interest in symmetrical cryptographic systems, because it directly handles the keys. Figure 4 shows a dual-rail xor gate implementation. Every computations of this dual-rail xor gate involve a fixed and constant number of transitions regardless of the data values. Hence, its power consumption is data independent, i.e. not correlated to the processed data, which is exactly the goal to achieve.

However, the QDI implementation of a function is not always balanced, as it is for example the case for a dual rail AND gate (Figure 5). The number of transitions remains fixed and perfectly known, but it is data dependent. When “ai” or “bi” is zero, the circuit consumes two transitions in a C-element and in the OR gate. When “ai” and “bi” are ones, the circuit only consumes one transition in the C-element gate. In such cases, the gate structure is modified to ensure that all data paths and control paths are balanced and do involve a constant number of transitions [8]. A balanced dual-rail AND gate is proposed in Figure 5.

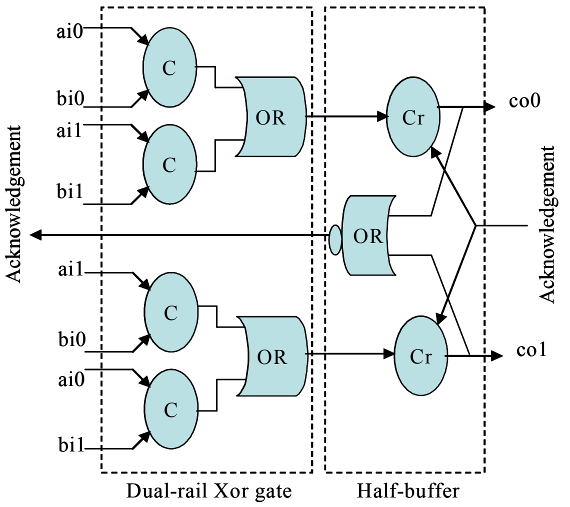


Fig. 4: Dual-rail xor gate with an output half-buffer (four-phase handshake protocol).
Dual rail “co” outputs the xor function performed between dual rail inputs “ai” and “bi”, (Cr is a Muller gate with a reset signal).

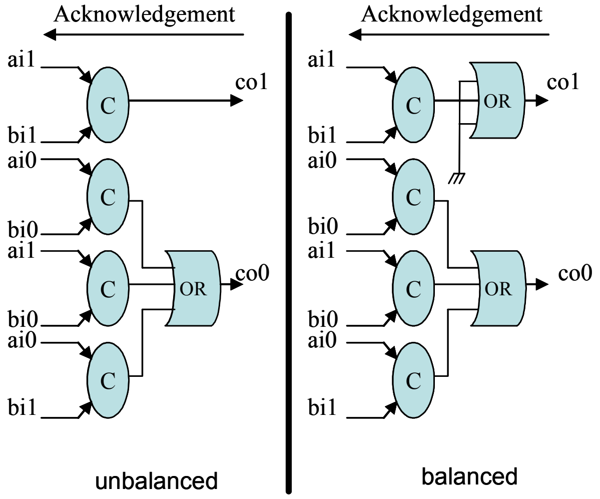


Fig. 5: dual – rail AND gate, unbalanced and secure versions (no output half-buffer).

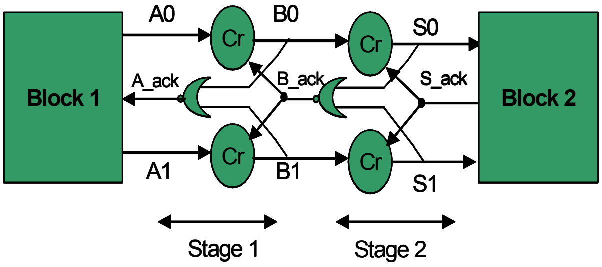


Fig. 5-b: A four-phase Dual-rail Buffer used as a memory.

We have proposed [19] an approach to design balanced registers, data-paths and finite state machine structures which involves a fixed constant number of transitions to compute.

This technique relies on the adoption of a flexible structure only based on balanced computational blocks such as explained here-before, and balanced memory element called half-buffer and full buffer such as the ones illustrated in figure 5-b.

Moreover, this design approach is integrated in a design framework, called TAST (Tima Asynchronous Synthesis Tools), which enables an automatic generation of such balanced circuits. Circuits are modelled using a high level hardware description language called CHP (Communicating Hardware Processes) [15] [16]. The TAST CHP enables the designer to use multi-rail data types implemented using 1 of N encoding [14]. After synthesis, the tool formally verifies that the structure of the circuit is well balanced, and that the number of transitions involved in the computation is data independent. Thus, at this logical or gate level the circuits are formally proven to be DPA resistant.

III ASYNCHRONOUS AES ARCHITECTURE

In this section we apply the design approach presented in section II to the design of an AES crypto-processor with secure asynchronous blocks. The proposed architecture is compliant with the NIST AES standard: 128 bit data blocks and 128, 192 or 256 bit keys [4].

In order to easily interface the asynchronous AES crypto processor to standard synchronous processors or ASICs, a synchronous register-file and specific synchronous/ asynchronous and asynchronous/synchronous interfaces are designed (Figure 6).

Apart from these interfaces, the asynchronous core of the circuit is built of two main blocks: cipher block (AES_core) and the key scheduling block (AES_key) (Figure 6).

III.1 REGISTER FILE

The register-file is composed of a Mode register and 3 register-sets respectively storing the plain text, the keys, and the ciphered text.

- 1 register of 4 bits for the Mode,
- 8 registers of 16 bits for the plain text,
- 16 registers of 16 bits for the key,
- 8 registers of 16 bits for the ciphered text.

The Mode register allows the user to configure the crypto processor (key length), and on request to start the computation. It also includes a flag which is set when the text is ciphered and ready for up-loading.

III.2 INTERFACES

This block implements the conversion functions required for a synchronous environment to communicate and synchronize with an asynchronous block. The synchronous/asynchronous interface converts standard binary data to N-rail data. The asynchronous/synchronous interface converts N-rail data to standard binary data.

As this work objective is to evaluate the asynchronous technology potentials in terms of DPA resistance, these interfaces and register-file are not designed for security and will not be operating when the asynchronous AES is computing.

III.3 CIPHER BLOCK (AES_core)

This block implements the four main functions of the Rijndael algorithm, namely: Addkey, Bytesubs, Shiftrows and Mixcolumns. The data path of one round is described in Figure 7. During the first round, only Addkey is used. During the last round Mixcolumns is not used [9].

Given this data path, three architectures can be considered according to the number of bytes processed (1, 4 or 16 bytes). In the first and the second cases (1 or 4 bytes), the use of registers is necessary: a minimum of 15 registers for the 1-byte data path, and 12 registers for the 4-byte data path. With a key length of 128 bits, the computation of the 128-bit text requires 160 and 40 iterations respectively. The fully parallel architecture processing the 16-byte in parallel would require a lot of hardware resources, especially for the substitution function (16 Bytesubs, only for the cipher block). Given the targeted applications and the estimated hardware costs, the best speed/area trade-off is the 4-byte data-path. Its architecture is depicted in Figure 8 whereas the other architectures are discussed in section VI.

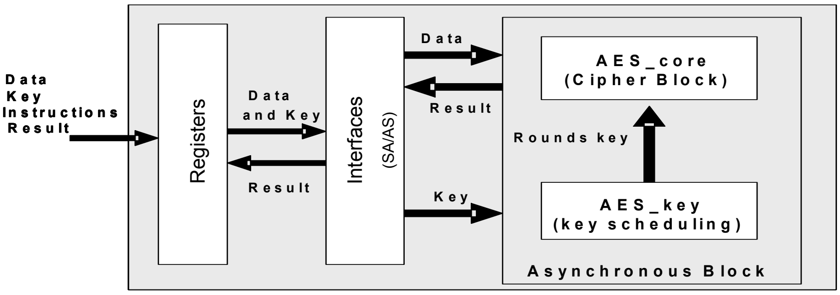


Fig. 6: AES component.

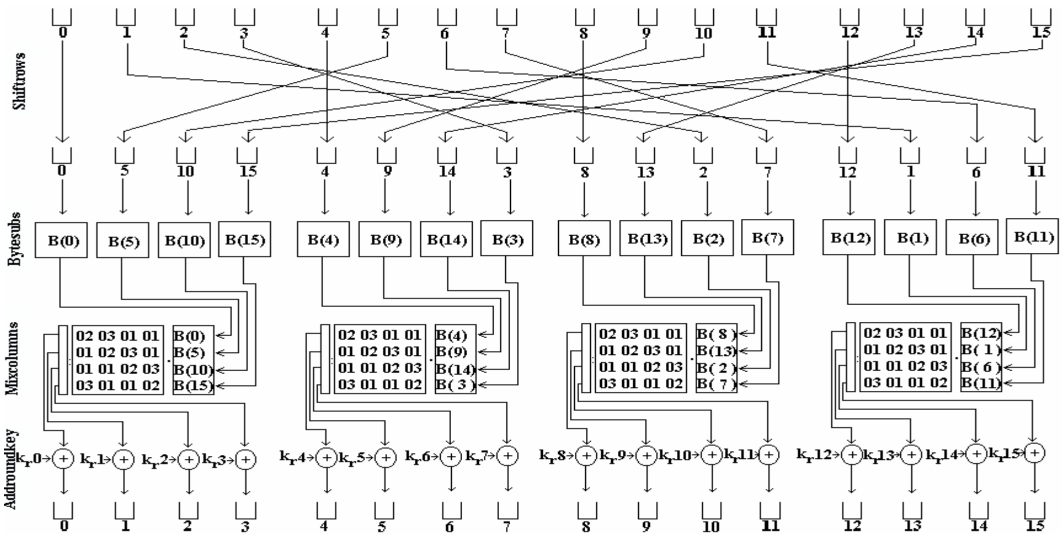

Shiftrows
Bytesubs
Mixcolumns
Addroundkey
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0 5 10 15 4 9 14 3 8 13 2 7 12 1 6 11
B(0) B(5) B(10) B(15) B(4) B(9) B(14) B(3) B(8) B(13) B(2) B(7) B(12) B(1) B(6) B(11)
02 03 01 01
01 02 03 01
01 01 02 03
03 01 01 02
k_r0 k_r1 k_r2 k_r3 k_r4 k_r5 k_r6 k_r7 k_r8 k_r9 k_r10 k_r11 k_r12 k_r13 k_r14 k_r15
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Fig. 7: Rijndael Round operation

III.3.1 ADDKEY FUNCTION IMPLEMENTATION

It is an Exclusive-Or between the four bytes of the State and the four bytes of the key. The Addkey0 block deals with the initial key, the Addroundkey block takes sub-keys as inputs, whereas the Addlaskey block computes the last ciphering operation with the last sub-key. As presented in figure 4, the use of Dual-rail xor gates ensures a well balanced architecture.

III.3.2 BYTESUBS FUNCTION

Its equation is given by:

$$B(z)=[(1F).(z)^{-1}+(63)]\mathrm{mod}(x^8+1)$$

It is constructed by the composition of two transformations: an affine transformation and an inverse multiplication.

- The affine transformation is defined by:

$$B_{aff}(z)=[(1F)(A(x))+(63)]\mathrm{mod}(x^8+1)$$

The expansion of this equation is given by:

$$\begin{aligned}
b_7 &= a_3+a_4+a_5+a_6+a_7\\
b_6 &= a_2+a_3+a_4+a_5+a_6+1\\
b_5 &= a_1+a_2+a_3+a_4+a_5+1\\
b_4 &= a_0+a_1+a_2+a_3+a_4\\
b_3 &= a_0+a_1+a_2+a_3+a_7\\
b_2 &= a_0+a_1+a_2+a_6+a_7\\
b_1 &= a_0+a_1+a_5+a_6+a_7+1\\
b_0 &= a_0+a_4+a_5+a_6+a_7+1
\end{aligned}$$

After factorizing redundant terms, this function is implemented with 17 dual-rail xor gates. Because “a xor 1 = not (a)”, this operation does not require any hardware. In fact, the logical “not” of a dual-rail coded bit is simply obtained by exchanging the two rails.

- The inverse multiplication in GF(2^8) is defined by:

$$B_{inv}(z)=(z)^{-1}\ \mathrm{mod}(x^8+x^4+x^3+x+1)$$

This transformation is implemented by using the architecture defined in [2] [3] [5]. This architecture is based on changing the representation from the GF(2^8) Galois field into GF(2^4) Galois field, performing the inverse multiplication into GF(2^4) and finally converting the result back to GF(2^8). That is made possible because the finite Galois field GF(2^8) is isomorphic to the finite Galois field (GF(2^4))². GF(2^8) is considered as an extension of GF(2^4). It is formalized by: $a = a_h\ x + a_l$ with $a \in$ GF(2^8) and a_h, $a_l \in$ GF(2^4). All mathematical operations done in GF(2^8) remains possible in GF(2^4). The irreducible polynomial needed for modular reduction is given by:

$$n(x) = x^2 + \{1\}x + \{e\}$$

Coefficients {1} and {e} are written in hexadecimal.
Hence, the inverse multiplication is expressed as specified in equation (E1) below:

$$\begin{gathered}
(a_hx + a_l) * (a_hx + a_l)^{-1} = 1 \bmod n(x) \text{ with } a_h, a_l \in GF(2^4)\\
(a_hx + a_l)^{-1} = (a_h * d)x + (a_h + a_l) * d\\
\text{with } d = ((a_h^2 * \{e\}) + (a_h * a_l) + a_l^2)^{-1} \qquad (E1)
\end{gathered}$$

Figure 9 describes the architecture of this computation (E1).

The benefit of this alternative algorithm is the reduction of the hardware cost. In fact, because operations in GF(2^4) process 16 elements of GF(2^4), the adoption of 1 of 16 encoding for the GF(2^4) elements leads to an efficient hardware implementation. In this case every GF(2^4) element is represented by one rail. As an illustration of the reduction of hardware complexity, let’s consider the squaring block (Square_MR16) described in Figure 10 which now does not require any gate. It is simply implemented by wire exchanges.

Like the Square_MR16 block, the multiplication by constant {e} performed in block Mult_E_MR16 and the inverse computation performed in block Inverse_MR16 are simplified and implemented with wires only, thanks to the 1 of N encoding.

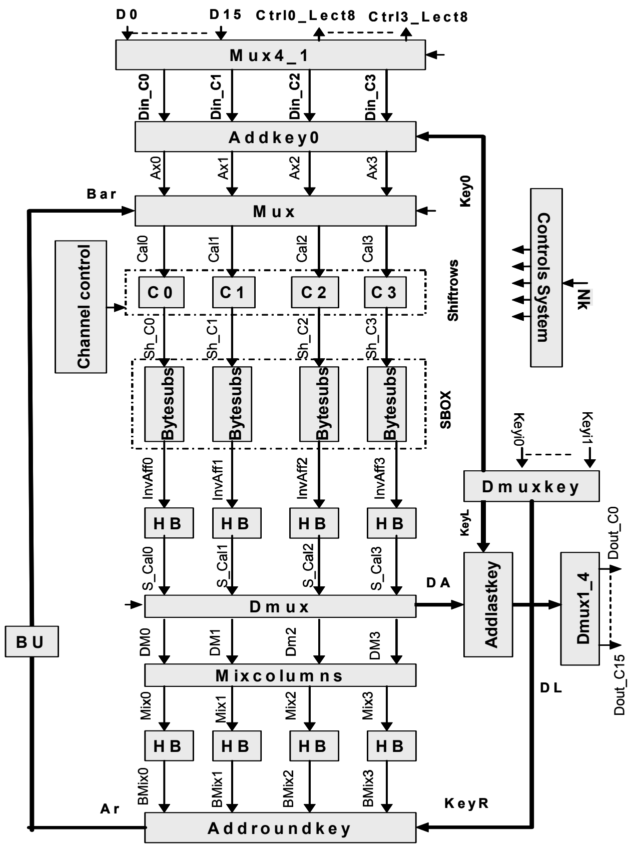


Fig. 8: Cipher block architecture (AES_core).

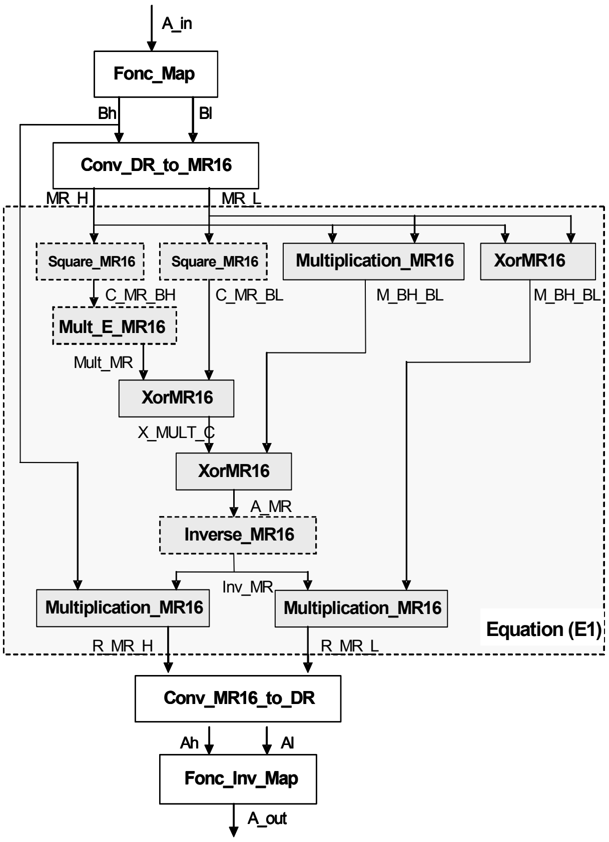


Fig. 9: Inverse function in GF(2^8).

The Xor_MR16 and the Multiplication_MR16 blocks are xor and multiplication operations also performed on 1 of 16 encoded data. Both blocks have very simple implementations involving C-elements and OR gates. If performing the computation into GF(2^4) using 1 of 16 encoded data is simplifying the computation, it requires some extra conversion functions. These functions are denoted Fonc_Map, Fonc_Inv_Map, Conv_DR_to_MR16 and Conv_MR16_to_DR in Figure 9. However, the extra hardware cost of these conversion blocks is negligible when compared to hardware savings realized on the computation blocks.

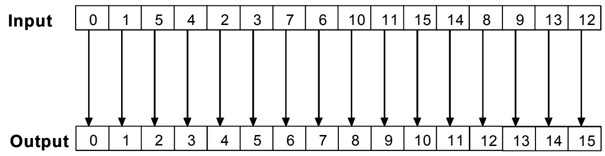


Fig. 10: Square_MR16 implementation.

To conclude, the choice of this algorithm together with the choice of the 1 of 16 encoding bring the following advantages.

- The power consumption is very low, because only 1 wire is activated (1 to 16 encoding data) when processing a GF(2^4) element.
- The architecture is balanced at the price of a very low overhead in order for the computation to involve a constant number of logical transitions regardless of the data values.
- The speed is high because the optimization proposed reduces the number of blocks on the critical path, and then reduces its latency.

III.3.3 MIXCOLUMNS FUNCTION

The Mixcolumns transformation consists in multiplying a column of 4 bytes by the square matrix defined by:

$$\begin{bmatrix} M_0 \\ M_1 \\ M_2 \\ M_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} . \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

It can be expanded to the following equations:

$$M_1 = 02\,(a + b) + b + c + d$$
$$M_2 = 02\,(b + c) + c + a + d$$
$$M_3 = 02\,(d + c) + a + b + d$$
$$M_4 = 02\,(a + d) + b + c + a$$

The implementation diagram is described in Figure 11.

The Xor8_SB blocks calculate 8 bit XORs. The “Xtime” block is defined by: “02.(a + b)”. It requires a shift and a reduction when the MSB’s value is one. To guarantee that the architecture is balanced, the reduction operation is always performed, regardless of the MSB’s value. The Xtime schematic, based on Muller C-elements and OR gates is described in Figure 12.

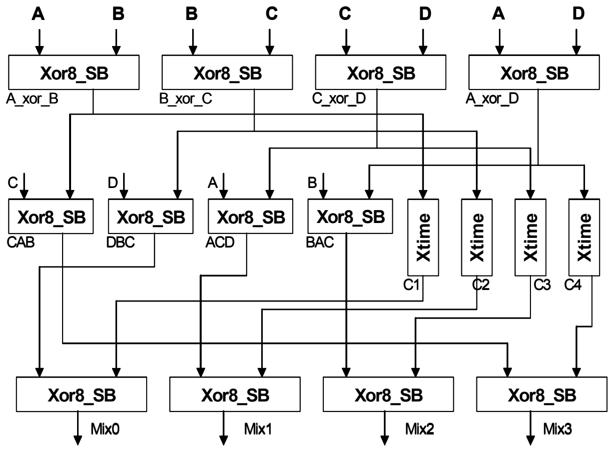


Fig. 11: Mixcolumns function.

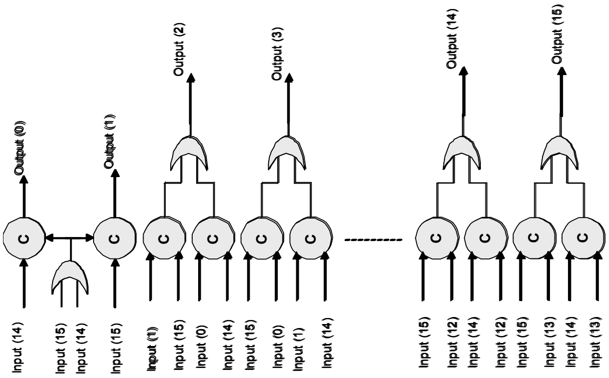


Fig. 12: Xtime function gate implementation.

III.3.4 SHIFTROWS FUNCTION

With the 4-byte data-path described in Figure 8, Shiftrows is built of four different blocks called C0 to C3 performing bytes reordering and rescheduling. C0 to C3 receive four 4-byte-packets in sequence: packet1 includes bytes 0 to 3, packet2 includes bytes 4 to 7, packet3 includes bytes 8 to 11, and packet4 includes bytes 12 to 15. At the output, C0 to C3 produce four 4-byte-packets in sequence as follows: packet1 includes bytes 0, 4, 8, 12, packet2 includes bytes 1, 5, 9, 13, packet3 includes bytes 2, 6, 10, 14, and packet4 includes bytes 3, 7, 11, 15 (Figure 13).

C0 to C3 are designed so as to minimize the memory resources. The minimum number of bytes stored in the structure is 12, equally distributed in the Ci blocks. A finite state machine is added to each Ci block in order to implement bytes rescheduling.

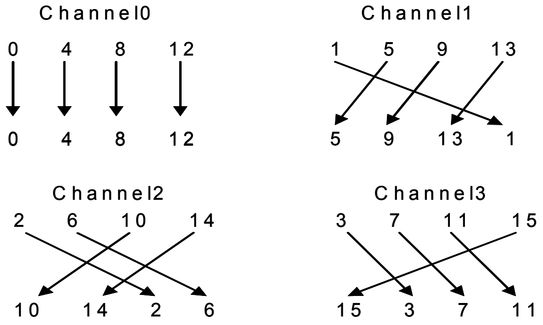


Fig. 13: Shiftrows C0 to C3 blocks specification.

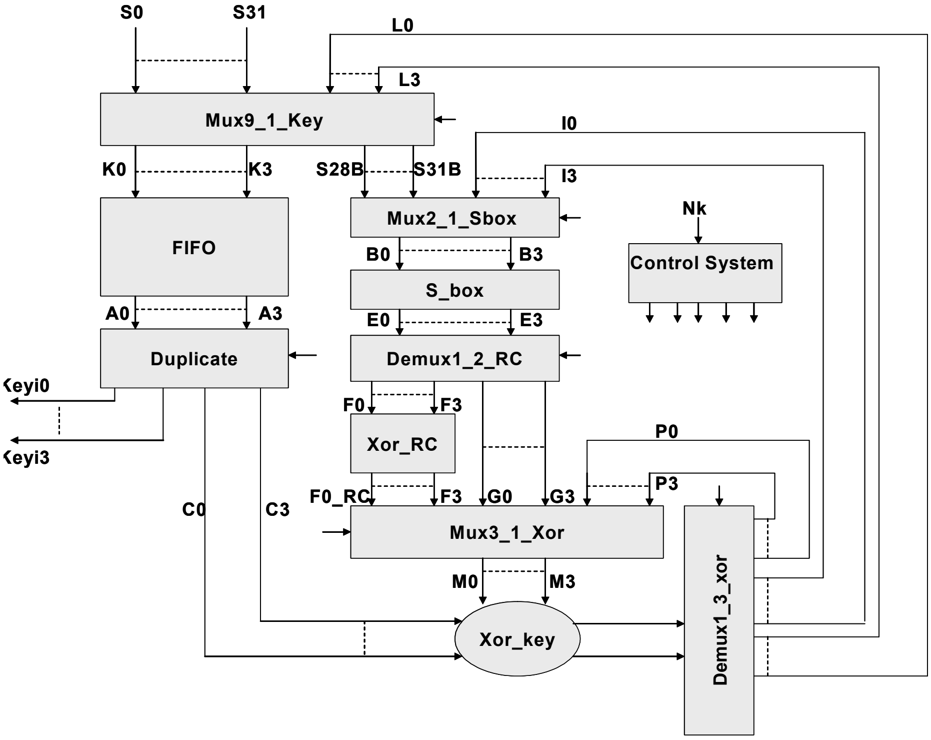


Fig. 15: AES_Key architecture.

IV KEY SCHEDULING (AES_key)

The AES_key block is in charge of generating, on fly, all necessary sub-keys for the ciphering block AES_Core. Its architecture is also based on a 4-byte data-path (Figure15). Most of the blocks have similar structures than the blocks used in the ciphering data-path, and are all balanced to involve a constant number of logical transitions.
Figure 15 describes the AES-Key block architecture. The FIFO obtained by cascading half-buffers, is required to temporally store the sub-keys within the computation loop. The AES Rcon(i) function is implemented by the XOR_RC block which includes a permutation of the inputs (Rotbytes function [9]).

V. RESULTS

All the test vectors provided by the NIST were used to validate the circuit. Validation was performed by simulating the CHP specification as well as the VHDL gate netlist. To design the asynchronous AES crypto-processor we used the 0,13 µm CMOS technology from STMicroelectronics.

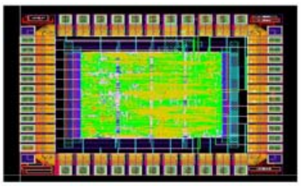


Fig. 16: QDI AES crypto-processor layout

Following the synthesis procedure presented in section II, we obtained a VHDL gate netlist of the whole architecture. This netlist is instantiating standard cells drawn from the library provided by STMicroelectronics only. No dedicated cells were used [10]. Place-and-route steps were performed using Silicon Ensemble tools from Cadence. The gate Netlist back annotated with wire and gate delays has been simulated. Layout area, speed and power figures are reported in table 2.

Table 2: Area, speed and power of AES crypto-processor (Standard Cells - Gate Netlist simulation with wire and gate delays back annotation).

Key length	Vdd	Area Without pads	Area With pads	Ciphering time	Power (average)	Throughput
128 bit key	1,2 V	0,490 mm²	0,169 mm²	850 ns	8,6 mA	150 Mbits/s
192 bit key				1030 ns		124 Mbits/s
256 bit key				1210 ns		106 Mbits/s
128 bit key	0,6 V			3920 ns	0,8 mA	33 Mbits/s
192 bit key				4720 ns		27 Mbits/s
256 bit key				5520 ns		23 Mbits/s

The ciphering time is the time elapsed from the writing of the start bit of the Mode register to the setting of the completion-flag. The larger the key, the longer the ciphering time. The computation of a single round requires about 90 ns when the circuit is powered at 1.2 volt. Therefore, the inner computational loop is performed within about 22.5 ns. The circuit still process with power supply of 0.6 volt by reducing current down to a facto ten and increasing time up to a factor four. According to the target applications, this flexibility makes possible to dramatically reduce the current in spite of time.

We have estimated the benefits of using dedicated cells [10] such as C-elements in terms of area and speed. Table 3 presents this estimation performed on the AES crypto-processor. The area is divided by two whereas the latency is reduced by about 30 %.

Table 3: Area and Speed estimations using dedicated cells.

Key length	Vdd	Area without pads	Ciphering time	Throughput
128 bit key	1,2 v	0,24 mm²	595 ns	215 Mbits/s
192 bit key			721 ns	178 Mbits/s
256 bit key			892 ns	143 Mbits/s

VI. ARCHITECTURE TRADE-OFFS

As stated in section II, the AES architecture is very modular and then offers several options for choosing the data-path wideness according to the applications and throughput targeted. Starting from the architecture presented in section III (Figure 8), we were able to evaluate two other data-path architectures: the 128-bit wide data-path for high throughput, and the 8-bit wide data-path for low speed / low area. Figure 17 gives the synopses of these architectures for the AES_Core. 8-bit and 128-bit AES_Key data-paths are easily obtained but are not represented.

Table 4 reports the comparison of the three architectures in terms of estimated area and speed when the key length is 128 bits by using Standard cells library. Even though table 4 concerns the AES_Core data-path only, the same scaling factors apply to the AES_Key data-paths.
Table 4 shows that the 8-bit data-path is twice smaller than the 32-bit data-path but is about 3.5 times slower. On the contrary, the 128-bit data-path is about 2.2 times larger, but is more four times faster than the 32-bit data-path.

Table 4: Area/speed trade-offs for the AES_Core data-path.

Data path	Vdd	Area	Ciphering time	Throughput
8 bits	1,2 v	0,088 mm²	3600 ns	36 Mbits/s
32 bits		0,192 mm²	850 ns	150 Mbits/s
128 bits		0,427 mm²	225 ns	569 Mbits/s

Moreover, applying aggressive pipelining techniques and using dedicated library cells would even increase the throughput of the 128-bit data-path.

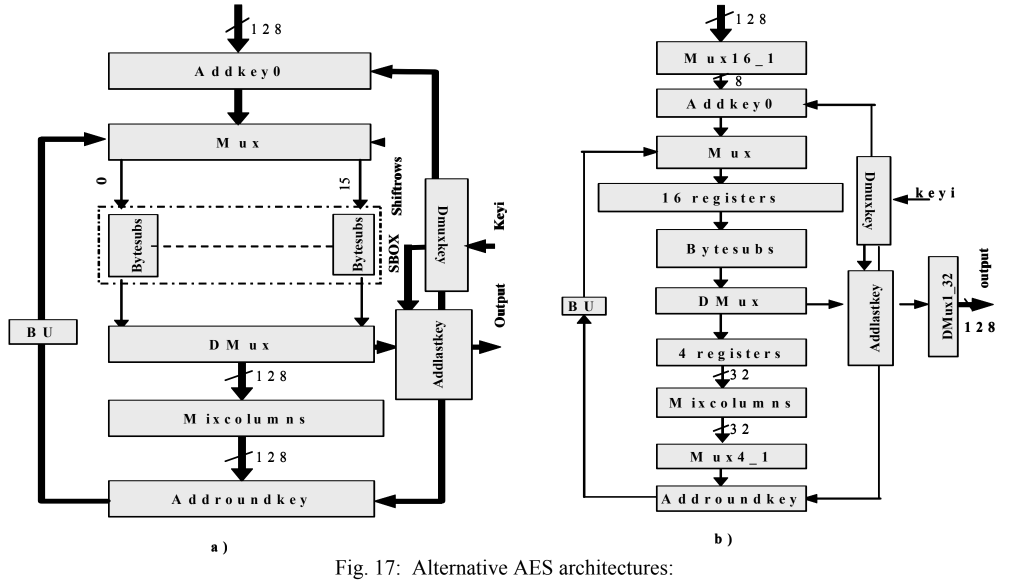


Fig. 17: Alternative AES architectures:
a) 128-bit data-path, b) 8-bit data-path.

VII. CONCLUSION

This paper presents the first secure QDI asynchronous architecture of the AES. It is shown how QDI asynchronous technology together with 1 of N encoding can be exploited to: i) reduce the hardware complexity and decrease the latency, ii) improve DPA resistance by logical paths balancing. Several architectural tradeoffs are considered, enabling the use of this new design methodology to a wide spectrum of applications ranging from networking to smart-cards.

Current works are focused on the back-end steps (standard cell choice and place and route) in order to preserve the benefits of logical path balancing at the electrical/physical level. Prototypes have been sent for fabrication. They will enable us to perform differential power analysis and measure the approach efficiency.

REFERENCES

[1] Henry Kuo, Ingrid Verbauwhede, “Architecture Optimization for a 1.82 Gbits/sec VLSI implementation of the AES Rijndael Algorithm”, Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings. Springer 2001, ISBN 3-540-42521-7.

[2] Johannes Wolkerstorfer, Elisabeth Oswald, and Mario Lamberger, “An ASIC Implementation of the AES Sboxes”, Wolkerstorfer J., Oswald E., Lamberger M. "An ASIC implementation of the AES S-Boxes", Proceedings of the Cryptographer's Track at the RSA Conference 2002, LNCS 2271, Springer Verlag, Feb. 2002.

[3] Vincent Rijmen, “Efficient Implementation of the Rijndael Sbox”, http://www.esat.ku-leuven.ac.be/~rijmen/rijndael/

[4] NIST, Advanced Encryption Standard (AES), FIPS PUBS 197, National Institute of Standards and Technology, November 2001. http://csrc.nist.gov/CryptoToolkit/aes/

[5] An Optimized S-box Circuit Architecture for Low Power AES Design. http://csrc.nist.gov/CryptoToolkit/aes/

[6] A. K. Lutz, J. Treichler, F. K. Gürkaynak, H. Kaeslin, G. Basler, A. Erni, S. Reichmuth, P. Rommens, S. Oetiker, and W. Fichtner, “2Gbit/s hardware realizations of RIJNDAEL and SERPENT: A comparative analysis”, Cryptographic Hardware and Embedded Systems - CHES 2002, LNCS 2523, pp. 144-158, 2003 Third Springer – Verlag Berlin Heidelberg 2003.

[7] Marc Renaudin, “Asynchronous circuits and systems: a promising design alternative”, Microelectronic Engineering 54 (2000) 133-149.

[8] Simon Moore, Ross Anderson, Paul Cunningham, Robert Mullins, George Taylor, “Improving Smart Card Security using Self-timed Circuits”, Eighth International Symposium on Asynchronous Circuits and systems (ASYNC2002). 8-11 april 2002. Manchester, U.K.

[9] J. Daemen and V. Rijimen, “AES Proposal: Rijndael.” Available at http://csrc.nist.gov/encryption/aes/rijndael/Rijndael.pdf

[10] P. Maurine, J.B. Rigaud, F. Bouesse, G. Sicard, M. Renaudin, “Static Implementation of QDI Asynchronous Primitives”, 13th International Workshop on Power and Timing Modeling, Optimization and Simulations, PATMOS2003.

[11] B. Weeks, M. Bean, T. Rozylowicz, C. Ficke : Hardware Performance Simulations of round 2 Advanced Encryption Standard Algorithms. National Security Agency (NSA)

[12] K Gaj, P. Chodowiec : Hardware performance of the AES finalists – survey and analysis of result. Technical Report, George Mason University, September 2000.

[13] N. Weaver, J. Wawrzynek : A Comparison of the Candidates Amenability to FPGA Implementation. Proceedings of the Third Advanced Encryption Stndard Candidate Conference, New York, April 2000, 28-39.

[14] TAST tutorial, Summer School on Asynchronous Circuit Design”, july 15-19,2002. TIMA Laboratory. Grenoble- France

[15] A. Martin, "Synthesis of Asynchronous VLSI Circuits", Caltech-CS-TR-93-28.

[16] A.J. Martin, "Programming in VLSI: from communicating processes to delay-insensitive circuits", in C.A.R. Hoare, editor, Developments in Concurrency and Communication, UT Year of Programming Series, 1990, Addison-Wesley, p. 1-64.

[17] A.J. Martin, "The limitations to delay-insensitivity in asynchronous circuits", in W.J. Dally, editor, Proceedings of the Sixth MIT Conference on Advanced Research in VLSI, 1990, MIT Press, p. 263-278.

[18] K. Van Berkel, "Beware the isochronic fork", Integration, the VLSI journal, N° 13, pp. 103-128, 1992.

[19] Marc Renaudin, Fraidy Bouesse, “On the design of secure chips”, EPFL conference, June 2002, http://tima.imag.fr/cis .

[20] Fraidy Bouesse, Laurent Fesquet, Marc Renaudin “QDI circuit to Improve Smartcard Security”, 2nd Asynchronous Circuit Design Worshop (ACID2002), Munich; Germany, 28-29 Januray,2002.

[21] A. Abrial, J. Bouvier, M. Renaudin, P Senn and P. Vivet, « A New Contactless Smart Card IC using On-chip Antenna and Asynchronous Microcontroller », Journal of Solid-State Circuits, Vol. 36, 2001, pp. 1101-1107.

[23] A. Shamir, “Protecting Smart Cards from Passive Power Analysis with Detached Power Supplies”, Proceedings of Cryptographic Hardware and Embedded Systems (CHES 2000), 2000, pp.71-77.

[24] K. Tiri, M. Akmal, I. Verbauwhede, "A Dynamic and Differential CMOS Logic with Signal Independent Power Consumption to Withstand Differential Power Analysis on Smart Cards", Proceedings of 28th European Solid-State Circuits Conference (ESSCIRC 2002), 2002, pp 403-406.

[25] J.J.A. Fournier, S. Moore, H. Li, R. Mullins, G. Taylor, "Security Evaluation of Asynchronous Circuits” proceedings of Cryptographic Hardware and Embedded Systems (CHES 2003), 2003, LNCS 2779, pp.137-151.