A Compact Piplined Hardware Implementation of the AES-128 Cipher

Nadia Nedjah, Luiza de Macedo Mourelle , Marco Paulo Cardoso

Abstract— **Advanced Encryption Standard – AES is the new encryption standard. In this paper, we propose a very efficient pipelined hardware implementation of AES-128 cipher. It has a competitive throughput of more than 2Gbits per second. Besides, improving the encryption throughput, the pipeline can be taken advantage of if the number of rounds (currently 10) must increase for security reasons.**

I. Introduction

The Advanced Encryption System – AES is a block cipher, adopted as the new encryption standard in substitution to its predecessor Data Encryption Standard – DES [4]. AES main scrambling computation is performed on a fixed block size of 128 bits with a key size of 128, 192 or 256 bits. This core computation is iterated for many rounds. The number of rounds depends on the key size. Currently, it is set to 10, 12 and 14 for the cited keys sizes respectively. The resistance of AES against breaking attacks depends entirely on the number of rounds used. So far, the best known attacks are on 7 rounds for 128-bit keys, 8 rounds for 192-bit keys, and 9 rounds for 256-bit keys [2]. The small margin between these round numbers and the actual ones is very worrying for the cryptographer's community.

In this paper, we propose a novel hardware implementation of AES-128. The architecture allows one to perform the core computation of the algorithm is a pipelined manner. The throughput of the cryptographic hardware is more than 2Gbits/s. The piplined execution of the AES algorithm allows an increase of the number of rounds without much loss of efficiency. Increasing the number of rounds applied, improves the resistance of the AES algorithm to cryptanalysis attacks.

The rest of this paper is organised in 4 subsequent sections. First, in Section II, we give a brief description of the AES encryption and decryption algorithms as well as modified version of these two algorithms, which are the basis of the proposed hardware architecture. Thereafter, in Section III, we describe in a structured manner, the pipelined hardware architecture of AES-128 for encryption and decryption. Subsequently, in Section IV, we present some experimental result and compare our implementation to existing ones. Last but not least, in Section V, we draw some conclusions and introduce some directions for future work.

II. Advanced Encryption Standard

AES is an elegant and a so-far-secure cipher. Encryption using AES proceeds as described in Algorithm 1, wherein functions *SubBytes*, *ShiftRows*, *MixColumns* and *AddroundKey* are defined as follows:

- Function *SubBytes* yields a new state simply by substituting each of the 16 bytes of $state$ using a substitution box. The four most significant bits of the byte in question is used as the S-box row index while the remaining four bits are used as the S-box column index.
- Function *ShiftRows* obtains a new state by cyclically shifting the state rows. The bytes of row i are shifted i times, where $0 \leq i \leq 4$.
- Function *MixColumns* operates on the states columns. The bytes of a given column are used as coefficients of a polynomial over GF(2^8). The formed polynomial is multiplied by a fixed polynomial $P(x)$ *modulo* x^4+1, wherein $P(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$. The details of the multiplication operation can be found in [5], [1].
- Function *AddRoundKey* computes the new state using a XOR of the columns bytes and the key schedule of the current round.

Before the cipher operation takes place, a key schedule is generated. Four subkeys are required for each round of the cipher algorithm. The subkeys for the first round are the private cipher key. For a given round, the first subkey is obtained by first rotating once the last subkey form the previous round, then substituting each of byte using the S-box used by function *subBytes*, thereafter XORing the result with a given constant and finally XORing the result with first subkey of the previous round. The subsequent subkeys of the current round are computed using a XOR of the previous key in the current round and the one inversely respective from the previous round.

```
Algorithm 1.AES-Cipher
input:  Byte $T[4 \times nb]$, Word $K[nb \times (nr+1)]$;
output:  Byte $C[4 \times nb]$,
  Byte $state[4, nb]$; $state$ := $T$;
  AddRoundKey($state$, $K[0, nb-1]$;
  for $round$ := 1 to $nr-1$ do
    SubBytes($state$); ShiftRows($state$);
    MixColumns($state$);
    AddRoundKey($state$, $K[round \times nb, nb(round+1)-1]$);
  SubBytes($state$); ShiftRows($state$);
  AddRoundKey($state$, $K[nr \times nb, nb(nr+1)-1]$);
  $C$ := $state$;
  return $C$;
end
```

N. Nedjah, M. Mourelle and M. Cardoso are with the State University of Rio de Janeiro.

For hardware efficiency reasons, we modified the AES cipher algorithm as in Algorithm 2. Note that Algorithm 1 and Algorithm 2 are equivalent and yield the same output.

```
**Algorithm 2.**Modified-AES-Cipher
**input:** Byte $T[4 \times nb]$, Word $K[nb \times (nr+1)]$;
**output:** Byte $C[4 \times nb]$,
  Byte $state[4, nb]$; $state$ := $T$;
  for $round$ := 0 to $nr - 1$ do
    AddRoundKey($state$, $K[round \times nb, nb(round+1)-1]$);
    SubBytes($state$); ShiftRows($state$);
    if $round < nr - 1$ then MixColumns($state$);
  AddRoundKey($state$, $K[nr \times nb, nb(nr+1)-1]$);
  $C$ := $state$;
  return $C$;
**end**
```

III. Pipelined Hardware Implementation of AES

The overall architecture of the AES hardware mirrors the structure of Algorithm 2. It is a synchronous implementation of both the processes of cipher. It uses four 128-registers. Every clock transition, these registers are loaded, except *Register*$_3$, which is loaded when an input state is completely ciphered. During the encryption process, *Register*$_0$ is loaded with the input data or the partially encrypted text; *Register*$_1$ with the result of the *AddRoundKey* component; *Register*$_2$ with the state after applying functions *SubBytes* and subsequently *ShiftRows*. In The block architecture of the AES cipher hardware is shown in Fig. 1.

The component that implements function *AddRoundKey* is simply a net of XOR gates that adds in $GF(2^8)$ the key schedule to the current state. The component implementing function *SubBytes* uses 16 S-boxes stored in a Read-Only Memory (ROM). The obtained state is row-shifted before its storage in *Register*$_2$. The component architecture is given in Fig. 2.

Function *MixColumns* is implemented by a massively parallel component that computes all the bytes of the new state in a single clock. It uses four components of the same architecture. This basic component produces one column os the new state. Its architecture is described in Fig. 3, wherein component *mult* yields the a special product of a given byte from the state times {01}, {02} or {03} (see [5], [1] for details on the operation). The architecture of component *mult* is presented in Fig. 4. Component *xtime* computes the *xtime* operation as defined in [5] and shown in Fig. 5.

For component synchronisation purposes, the architecture includes a controller. Among other actions, the controller determines when to reset the cipher hardware, accept input data, to register output results. As the execution of function *MixColumn* is conditional (see Algorithm 2), the controller decides when the result obtained by component *MixColumn* can be used or must be ignored.

The controller is structured as in Fig. 6. The included combinational logic permits the conversion of the 5-bit count to a single bit that triggers state transition. The sate machine includes six states. As long as control signal *keyExpand* is set, the current state is kept unchanged in S_0. As soon as this signal is reset by the *keyExpansion* component, which means that the step of key schedule generation is complete, the machine transits to state S_1, wherein it stays for 3 clock cycles, which is the required time to complete the processing of one 128-bit input state. Also, during this period of time, the data input signal is active, which allows the hardware to accept the 3 input states that will be ciphered in pipelined manner. Synchronously with the fourth clock transition, the machine transits to state S_2 allowing to deactivate the data input signal and waits for the three accepted input states to be almost processed, i.e. only the last *AddRoundKey* is yet to be performed to complete the encryption process. At the

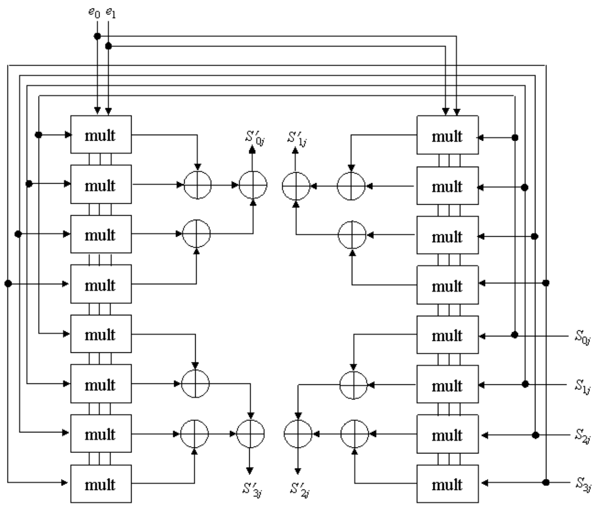


Fig. 3. Basic component in *MixColumns* component

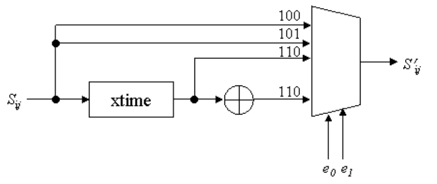


Fig. 4. Architecture of the *mult* component

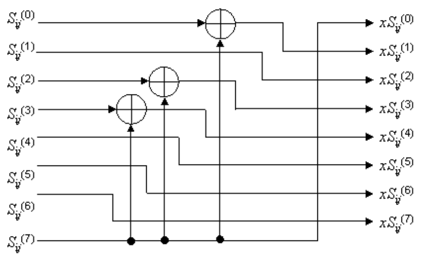


Fig. 5. Description of operation *xtime*

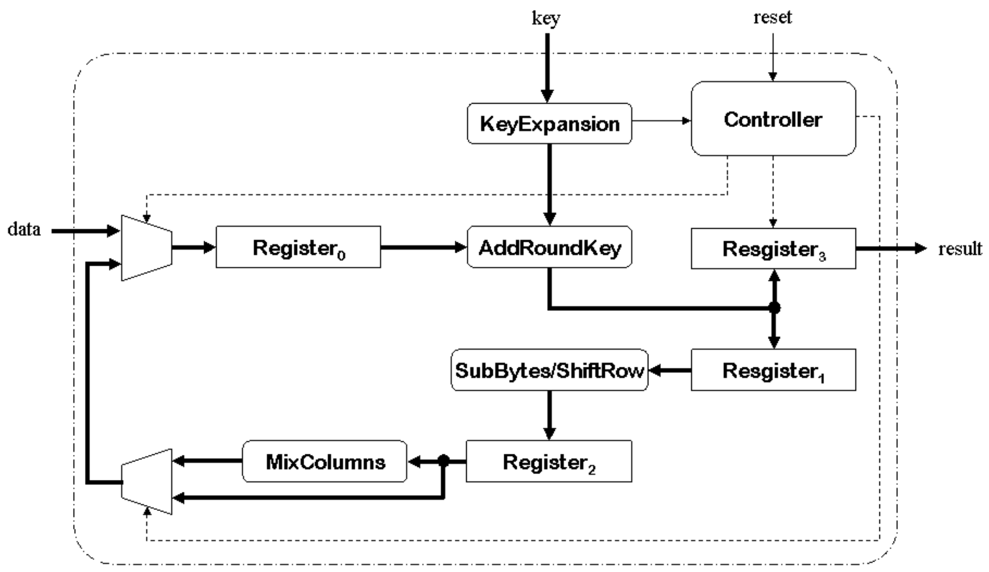
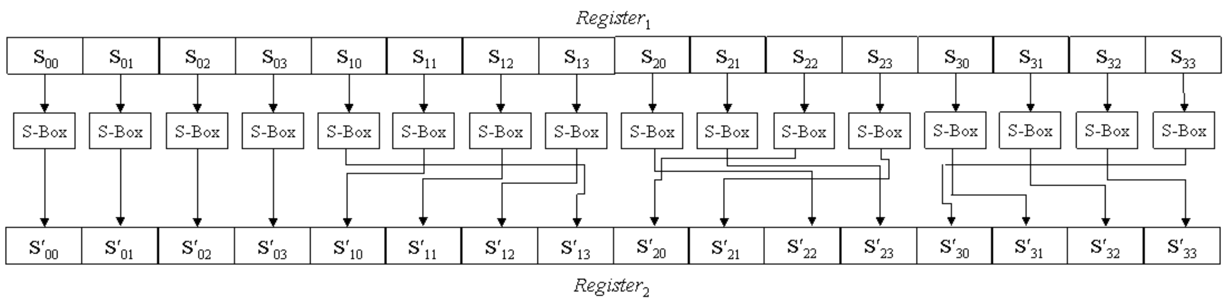


Fig. 1. Overall hardware architecture for the AES-128 cipher



Register1
S00 S01 S02 S03 S10 S11 S12 S13 S20 S21 S22 S23 S30 S31 S32 S33
S-Box
S'00 S'01 S'02 S'03 S'10 S'11 S'12 S'13 S'20 S'21 S'22 S'23 S'30 S'31 S'32 S'33
Register2

Fig. 2. Component implementing both functions *SubBytes* and *ShiftRows*

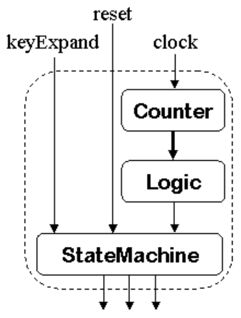


Fig. 6. Controller architecture

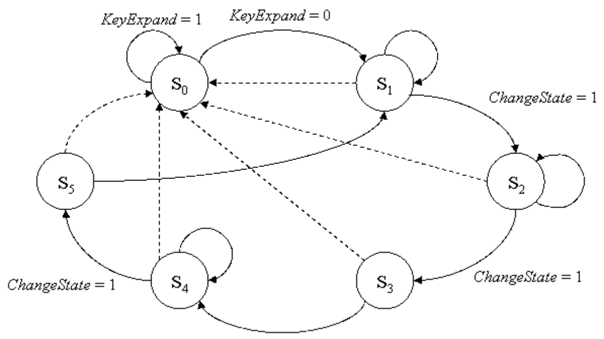


Fig. 7. State machine transition diagram

30th. clock transition, the machine state changes to S_3 to activate output result signal, which is maintained for the two subsequent clock periods. A the 33rd. clock transition, the encryption of the 3 accepted input states is completed and therefore, the control is returned to state S_1, where in data input signal is reactivated to allow more date to be entered and processed. The state machine transition diagram is shown in Fig. 7.

IV. Experimental Results

The pipelined execution of the AES cipher using the architecture of Fig. 1 is illustrated in Fig. 8. We implemented the hardware described throughout this paper using reconfigurable hardware. The FPGA family used is VIRTEX-II. Component *KeyExpansion* introduces a delay of 62.9ns. The clock cycle

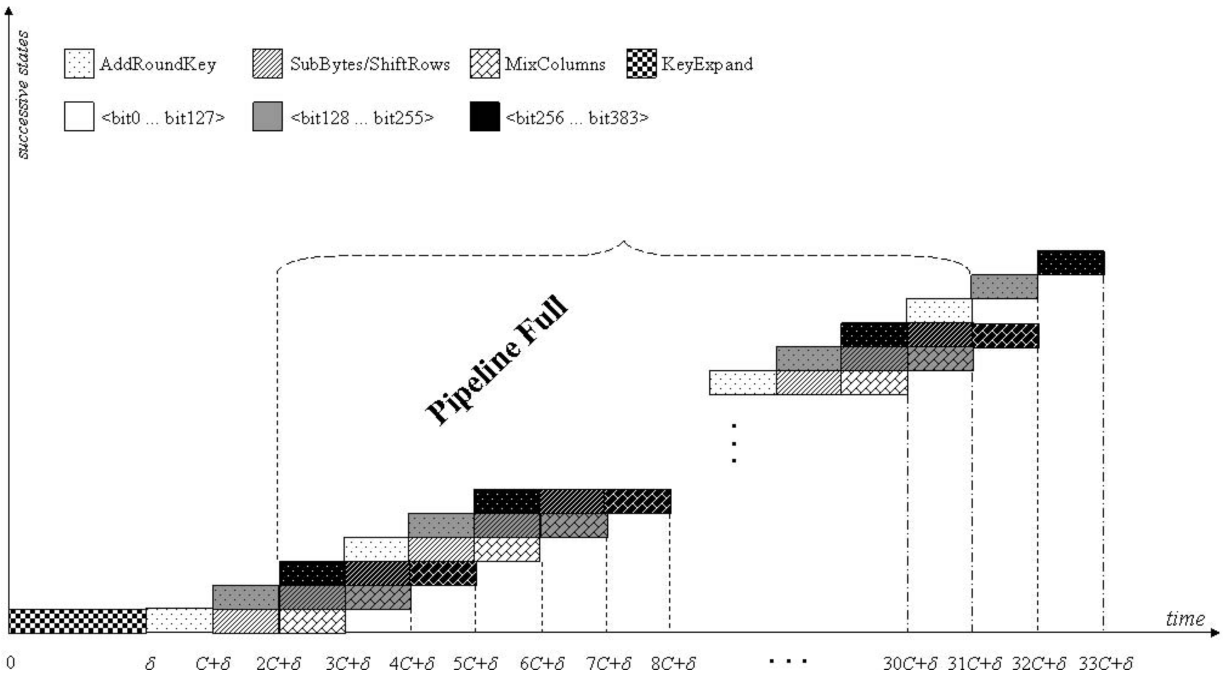


Fig. 8. Pipelined execution of the AES algorithm using the hardware described in Fig. 1

TABLE I
PERFORMANCE COMPARISON

Implementation	Throughput	Area	CLB/Mbs
Our's	2118	1937	915
[3]	1911	8767	4588
[7]	1450	542	374

is 5.24ns. Every 33 clock cycles, the hardware can yield an encrypted datastream of 3×128 bits. The throughput, say tp can then be calculated as in (1). The throughput is a little more than 2Gbps.

$$Tp = \frac{3 \times 128}{33 \times clockcylcle} = \frac{128}{11 \times 5.24} = 2117.8Mbs \quad (1)$$

We compared our implementation to the ones from [3] and [7]. The throughput, expressed in Mbps, as well as the hardware area required, expressed in number of CLBs, are given in Table I.

Recall that the resistance of AES-based encryption against cryptanalysis attacks depends entirely on the number of rounds used. The pipelined implementation we propose throughout this paper can be easily adapted to a higher round number. The chart of Fig. 9 show that this can be done without much loss in efficiency. To be able to increase the number of round, component *KeyExpansion* needs to generate more key schedules and therefore the delay introduced by it increases with the number of rounds. The throughput, say tp, can be expressed in terms of the round number, say rn, is as in (2).

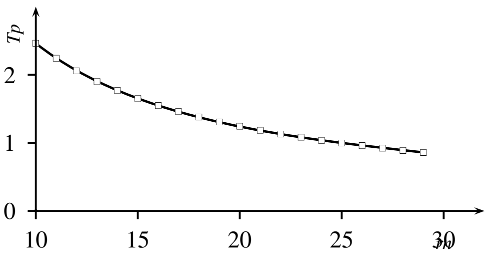


Fig. 9. The impact of increase in the round number

$$Tp(rn) = \frac{128}{(rn+1) \times clockcycle} \quad (2)$$

V. CONCLUSION

In this paper, we proposed a novel hardware implementation of AES-128. The architecture allows one to perform the core computation of the algorithm is a pipelined manner. The throughput of the cryptographic hardware is more than 2Gbits per seconds. The pipelined execution of the AES algorithm allows an increase of the number of rounds without much loss of efficiency. Increasing the number of rounds applied, improves the resistance of the AES algorithm to cryptanalysis attacks.

In future work, we intend to generalise the proposed hardware to allow decryption as well. This must be done without loss of efficiency and also without much increase of hardware area.

References

[1] J. Daemen and V. Rijmen, *The Design of Rijndael: AES – The Advanced Encryption Standard*, Springer-Verlag, 2002.

[2] N. Ferguson, J. Kelsey, S. Lucks, B. Schneier, M. Stay, D. Wagner and D. Whiting, *Improved Cryptanalysis of Rijndael*, Proceedings of FSE 2000, pp. 213-230, 2000.

[3] A. Labbe, Annie Perez, *AES Implementation on FPGA: Time and Flexibility Tradeoff*, in Proceedings of FPL, pp. 836-844, 2002.

[4] National Institute of Standard and Technology, *Data Encryption Standard*, Federal Information Processing Standards 46, November 1977.

[5] National Institute of Standard and Technology, *Advanced Encryption Standard*, Federal Information Processing Standards 197, November 2001.

[6] Nicolas Courtois, Josef Pieprzyk, *Cryptanalysis of Block Ciphers with Overdefined Systems of Equations*, Proceedings of ASIACRYPT 2002, pp 267-287, 2002.

[7] F. Standaert, G. Rouvroy, J. Quisquater, J. Legat, *A Methodology to Implement Block Ciphers in Reconfigurable Hardware and its Application to Fast and Compact AES RIJNDAEL*, in Proceedings of FPGA, 2003.