RECONFIGURABLE HARDWARE (FPGA) IMPLEMENTATION OF CRYPTOGRAPHIC ALGORITHMS - AES DECRYPTION

Paul BURCIU¹, Ionuț Mihai SIMA²
University of Pitești, Electronics Communications & Computer Faculty
E-mail: ¹pburciu@yahoo.com, ²mihaibb2000@yahoo.com

Keywords: hardware implementation, cryptographic algorithm, FPGA platform, cryptographic module, ASIC chip, AES, VHDL code, hardware simulation.

Abstract: The hardware implementation of cryptographic algorithms is a timely method, providing efficient security solutions, both regarding the processing speed and the consumed power. The present-day FPGA platforms, which are the physical groundwork for such implementations, however they are not new engineering solutions, assert as the most efficient way to practically transpose the cryptographic algorithms, resulting optimized (concerning diversified aspects) cryptographic modules, respectively in the end, unmatched (concerning performance) ASIC chips. This paper presents a new hardware implementation for the deciphering block of the AES (Advanced Encryption Standard) symmetric cryptographic algorithm, using VHDL programming language and a hardware simulation of the resulted deciphering module.

1. INTRODUCTION

The cryptographic algorithms became the main proceeding for protection of very important data, the security objective called confidentiality being the one taken into account by their hardware implementation and by their integration into the present-day communication systems.

Among the diverse cryptographic algorithms, the symmetric algorithms may be considered as the most susceptible of being hardware implemented, because the mathematical mechanisms used by them contain arithmetical operations which can be executed by logical combinational or sequential circuits, namely both by ordinary logical gates and by Finite State Machines (FSM), according to Church-Turing Thesis [1], in other words by VLSI digital integrated circuits.

The international contest organized by the National Institute of Standards and Technology (NIST) in 1997 for selection of the new symmetric cryptographic algorithm, which was intended to replace the old DES, successfully attacked and proved to be insecure, imposed in 2000 the Belgian algorithm RIJINDAEL as the winner and designated as the American cryptographic standard, under the name of Advanced Encryption Standard (AES), through the FIPS PUB 197 in 2001 [2].

The hardware implementation of cryptographic algorithms is a timely method, providing efficient security solutions, both regarding the processing speed and the consumed power. The present-day FPGA platforms, which are the physical groundwork for such implementations, however they are not new engineering solutions, assert as the most efficient way to practically transpose the cryptographic algorithms, resulting optimized (concerning diversified aspects) cryptographic modules, respectively in the end, unmatched (concerning performance) ASIC chips.

This paper presents a new hardware implementation for the deciphering block of the AES (Advanced Encryption Standard) symmetric cryptographic algorithm, using VHDL programming language and a hardware simulation of the resulted deciphering module.
2. THE BASIC CRYPTOGRAPHIC ARCHITECTURE

The 2 architectures of the AES block deciphering algorithm contain, as described in Figure 1 and Figure 2, an initial round where at its input a 128 bits ciphertext is applied, consisting of 4 columns with 4 bytes each, or in other words, 4 words with 32 bits each.

The ciphertext is XORed with the initial key, also represented as a data block consisting of 4 words with 32 bits each.

The inverse algorithm, like the encryption algorithm, has 3 possible versions [2], respectively \( N_k = 4, 6, 8 \) (the number of 4 bytes columns, or 32 bits words, of the key) corresponding to AES-128, AES-192 and AES-256 versions, also uniquely \( N_b = 4 \) (the number of 4 bytes columns, or 32 bits words, of the plaintext), namely 128 bits.

The architectures presented by Figure 1 and Figure 2 refer to the AES-128 version.

For the 3 constructive versions we have 3 different round numbers, as shown by the presented table in Figure 3.

<table>
<thead>
<tr>
<th>Name</th>
<th>Key Length ((N_k\text{ words}))</th>
<th>Block Size ((N_b\text{ words}))</th>
<th>Number of Rounds ((N_r))</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES-128</td>
<td>4</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>AES-192</td>
<td>6</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>AES-256</td>
<td>8</td>
<td>4</td>
<td>14</td>
</tr>
</tbody>
</table>

Figure 3 - AES versions characteristics.

After the initial round, AES-128 contains 9 decryption rounds. Each of these rounds has a corresponding round key supplied by the key schedule mechanism which starts the key generation process with the initial secret key. Every round consists of 4 main operations: Inv Shift Rows (every row of the State is right shifted by a specific number of positions, which is the inverse transposition of those made in the encryption case), Inv Sub Bytes (the substitution of every byte of the State using an inverse substitution table), Inv Mix Columns (which produces the two variants of implementation) and the Add Round Key (the round key is XORed with the resulted State of the previous operation). This sequence of operations is
iterated 9 times.

We know that the column mixing operations (Mix Columns() and Inv Mix Columns()) are linear with respect to the column input [2], which means:

$$\text{InvMixColumns(State} \oplus \text{Round Key) = InvMixColumns(State) \oplus \text{InvMixColumns(Round Key)}}$$

We will obtain two variants for the decryption algorithm: we firstly calculate the Inv Mix Columns of the State and of the Round Key and finally XOR the two results, or, secondly, we XOR the State with the Round Key, and after it we'll apply the Inv Mix Columns function to the result.

The Inv Mix Columns function is a Galois Field ($2^8$) multiplication between the operand (described above), each of its columns being polynomials over GF($2^8$)), and a fixed polynomial given by:

$$\begin{bmatrix}
0E & 09 & 0D & 0B \\
09 & 0E & 0B & 0D \\
0D & 09 & 0E & 0B \\
0B & 0D & 09 & 0E
\end{bmatrix}$$

This implementation uses a decomposition of the inverse affine matrix, in two matrix terms:

$$\begin{bmatrix}
0E & 0B & 0D & 09 \\
09 & 0E & 0B & 0D \\
0D & 09 & 0E & 0B \\
0B & 0D & 09 & 0E
\end{bmatrix} = \begin{bmatrix}
S_{0,c} \\
S_{1,c} \\
S_{2,c} \\
S_{3,c}
\end{bmatrix}$$

for $0 \leq c < N_b$.

3. FPGA IMPLEMENTATION OF AES

The main goal of this hardware implementation is not speed, but the area & resource limitations of a specific target FPGA device, respectively Xilinx Virtex-4 (model XC4VFX12 - FF668) which is a low resource platform, namely: 5472 slices, 320 I/O buffers, 10944 LUTs with 4 inputs. A summary of the occupied resources is presented in Figure 4 and Figure 5.

The implementation uses the VHDL programming language, which nowadays is a well-established commonly used language for FPGAs. The design & simulation software is
The decryption block is represented in Figure 6, where the main signals used by the implementation are shown.

![Figure 6 - The AES Decryption Block (both variants)](image)

The limitations of this device determined the use of 64 bit inputs, consequently loaded, firstly on the LOW-HIGH transition, and secondly on the HIGH-LOW transition, both in case of the input key and in case of the input ciphertext. The use of 128 bit inputs would easily lead to exceeding of the I/O buffer resources. No buffer limitations are imposed to the output, which is a 128 bit plaintext.

From Figure 4 and 5 we can easily notice that the first implementation variant occupies more than 77 % (4241 slices) of the device’s slices, when the second implementation needs about 68 % (3771 slices) of the slice resources. The number of occupied slice flip flops is 2484 (22 %) in the first case, and 2323 (21 %) in the second case. The number of 4 input LUTs is 6166 (56 %) in the first case, and 5470 (49 %) in the second case. The conclusion is that the second implementation is more efficient than the first one, concerning the number of occupied device resources, but slower, concerning the resulted work frequencies (see the resulted work frequencies from the next section).

In both variants of the decryption implementation, because the decryption uses the same key generation mechanism as the encryption algorithm, the key expansion routine is executed before the decryption itself. This happens because the generation of the same round keys as in the encryption case, synchronously with the specific rounds, but in a reverse order, was not possible as described in [2]. The main signals are: the system clock (CLK), the system reset (RESET), LOAD signal which loads the key and the ciphertext in the initial round, and BEGIN_DEC/END_DEC which starts/ends the decryption process. The loading process, which is described above, will be immediately followed by BEGIN_DEC signal which will start the decryption process. The entire decryption process takes exactly 12 clock periods from the HIGH-LOW transition of the LOAD signal. The signals diagram is represented in the simulation chapter in Figure 7 and 8.

Like in the encryption case, this implementation refers to the case of Electronic Code Book (ECB) mode of operation for the AES decryption.

### 4. Simulation & Security Aspects

The detailed diagrams of the simulation processes for the AES implementation are presented below, in Figure 7 and 8. The total duration of the decryption process is 4600 ns or 4.6 µs (approx. 28 Mb/s), calculated as described in the previous section. The resulted maximum clock frequency is 172.476 MHz \(\approx\) 172.5 MHz, corresponding to a clock period of 5.798 ns, respectively 159.808 MHz \(\approx\) 160 MHz, corresponding to a clock period of 6.258 ns. The simulations used a sequence of 128 ‘0’s, both for the input ciphertext and the secret input key. The output sequence is shown in Figure 7 and 8: ‘140F0F1011B5223D79587717FFD9EC3A’ (in hexadecimal code).

To be remarked that the END_DEC signal announces the final result of the decryption: when END_DEC = ‘1’, the plaintext appears on the output.

The evaluation of the security provided by this implementation must take into consideration the security requirements for cryptographic modules stipulated by [4]. In fact, the security of the cryptographic module stands on the security provided by the cryptographic algorithm itself. In [5], a complex evaluation of the 5 finalists of the NIST’s international contest concludes that using S-boxes as non-linear components, ‘Rijndael appears to have an adequate security margin, but has received some criticism suggesting that its mathematical structure may lead to attacks. On the other hand, the simple structure may have
facilitated its security analysis during the timeframe of the AES development process.
Even if some critiques were formulated (e.g. ‘the key schedule does not have high diffusion’), Rijndael was highly appreciated for the provided level of security. Regarding attacks on implementations, [5] remarks that ‘the operations used by Rijndael are among the easiest to defend against power and timing attacks.’

**Figure 7** - Simulation of the encryption block (1st variant)

**Figure 8** - Simulation of the encryption block (2nd variant)

5. CONCLUSIONS

This paper has presented a brief description of the implementation of the AES block decryption algorithm, underlining the benefits of this modern design concept. An FPGA implementation of a decryption algorithm is a cryptographic module device in which the structure is software implemented.

The FPGA implementations allow us to increase flexibility, lower costs, and reduce time to release enhanced cryptographic equipment, providing a satisfactory level of security for communication applications, or other electronic data transfer processes where security is needed.

**REFERENCES**