18-743 Project: Low Power AES Implementations

Nail Etkin Can Akkaya, Raymond Carley
Electrical and Computer Engineering Department
Carnegie Mellon University
Pittsburgh PA 15213, USA
Email: {nakkaya, rcarley} @ece.cmu.edu

Abstract—This report presents low power hardware implementations of the advanced encryption standard (AES). The low power circuit level designs will be implemented in a 65nm CMOS process from STMicroelectronics. There will be a synthesized low power base design for comparison purposes using general purpose standard $V_T$ (GPSVT) gate libraries. For comparison, the gate libraries of the base design will be changed to general purpose high $V_T$ (GPHVT), low power standard $V_T$ (LPSVT) and low power high $V_T$ (LPHVT) gate libraries. A burst mode design will also be implemented and compared. If time allows, further designs will be investigated as well. The low power implementations will be either synthesized using the Synopsys Design Compiler tool or will be designed fully custom on the Cadence Virtuoso tool, using GPSVT, GPHVT, LPSVT or LPHVT gate libraries. The focus of the project will be on low power AES-128 encryption. The keys required for the encryption will be supplied on the fly. After the synthesized and custom implementations will be either synthesized using the Synopsys Design Compiler tool or will be designed fully custom on the Cadence Virtuoso tool, using GPSVT, GPHVT, LPSVT or LPHVT gate libraries. The focus of the project will be on low power AES-128 encryption. The keys required for the encryption will be supplied on the fly. After the synthesized and custom implementations will be compared to the base design implementation with respect to their performance and energy. The comparison of different low power circuit level implementations will be useful for future low power AES research and low power AES hardware implementations.

I. INTRODUCTION

The Rijndael Algorithm was chosen in 2001 as the new Advanced Encryption Standard (AES) by the The National Institute of Standards and Technology [1]. AES is one of the most trusted and widespread encryption algorithms. It is a symmetric block cipher whereby the key needed for encryption and decryption are the same. The algorithm is able to adapt for three different key sizes, which are 128, 192, and 256 bits. The operations in AES are carried out byte-wise; therefore one major advantage is that AES can be efficiently implemented on various platforms, such as 8-bit microprocessors or common 32 processors [2]. Some common uses of AES are as follows:

- Communications LAN IEEE 802.11i, TSL, IPsec [2]
- NSA 128-bit = Secret; 192/256-bit = Top Secret [3]
- Smart Cards
- Smart Phones

AES implementations are flexible. They can be adapted for high performance devices to achieve high throughput. Conversely a low power or low energy implementation can be achieved for passively powered or battery powered devices, respectively.

This following sections are covered in this report. Related work is analyzed, especially low power/energy implementations. The technical details of AES and some low power circuit methods are laid out. Finally a timeline and deliverables for the project are given.

II. RELATED WORK

Since 2001, when the Rijndael Algorithm was chosen as the AES algorithm, there has been a lot of development of hardware and software implementations of that algorithm. The technical details of the algorithm will be shown in Section III. It will be shown that although the algorithm contains strict input and output requirements, the actual implementation of each internal stage is open to different interpretations at the architectural and circuit level. As stated previously, the aim of this project is to exploit circuit level techniques to realize a number of low power AES implementations; therefore a study was done on related work with similar low power goals.

Many implementations take advantage of the flexible nature of the AES algorithm and they implement a number of techniques to increase performance or reduce power & area. As will be further explained in Section III, all bytes are interpreted in the finite field with all arithmetic carried out $GF(2^8)$. At a very basic level it should be clear that this arithmetic can be broken down from $GF(2^8)$ into $GF(((2^2)^2)^2)$ to give the same result, but with smaller logic blocks. This leads to a reduction in power and can be seen in [4]–[7] Parallelism is also used to save power. In [8] the AES core is duplicated to create a parallel architecture. In this way the clock speed can be cut in half, saving 42% of the power, but maintaining the same throughput. In addition, scaling $V_{dd}$ from 1 V to 0.7 V saves 62% of the total power. This does of course more than double the overall area requirement. An example of pipelining is seen in [7].

Many papers focus on optimizing just one or two parts of the algorithm. The MixColumns step burns a lot of power, as it includes a lot of XORs and finite field multiplication. Optimizations of this step can be seen in [9], [10]. The SubBytes or S-Box step typically burns even more power than MixColumns and therefore there is a lot of focus on reducing this power through using pass transistor XOR gates [4], Galois field transformations [11], or by implementing the S-Box as a look up table in memory [5]. The AES operations are carried out on groups of bytes; therefore it is possible to do these operations in a serial manner and thus share hardware resources across steps. This can be seen in [12]–[15]. Two papers [11], [15], which have been cited previously for using certain low power methods, are of particular interest as they carry out these low power methods to so that they could meet the power budget of RFID smart cards (a few $\mu W$).
III. TECHNICAL DESCRIPTION

A. AES Details

The complete description of the AES algorithm is given in the NIST fips-197 document [1]; however for complete clarity an abridged description is given in this section. This will also include a brief explanation of some of the low power methods from the related work.

AES is a symmetric block cipher. It encrypts and decrypts data using a cryptographic key of length 128, 192 or 256 (thus giving the names AES-128, AES-192, and AES-256 depending on the key length). Unencrypted or decrypted data is called plaintext and encrypted data is called ciphertext. The symmetric nature of the cipher means that the same key must be used for encryption and decryption. The algorithm acts on the data in blocks of 128 bits. These 128 bits are further broken down into 16 8-bit words, labelled 0-15 and arranged in a 4x4 matrix called the State, as shown in Fig. 1. This arrangement is important for how the operations are carried out.

AES consists of four main steps:

1) SubBytes
2) ShiftRows
3) MixColumns
4) AddRoundKeys

The sequence of carrying out each of these steps is called a round. These rounds are repeated a set number of times, the only exception being that on the last round the MixColumns operation is omitted, as shown in Fig. 2. There are 10, 12, or 14 rounds for 128, 192, and 256 bit AES, respectively. Arithmetic in the AES algorithm is all carried out using the Galois Field, $GF(2^8)$, and thus all bytes are interpreted as finite field elements. This is important as it simplifies the hardware required for some of the steps.

1) SubBytes: The SubBytes step is very important as it typically takes up the majority of the area and power in the algorithm [13]. It is a straight substitution of every byte. To perform this an 8-bit S-Box is used. The S-Box can either be implemented as logic gates or as a look up table in ROM. The substitution is the multiplicative inverse in $GF(2^8)$ followed by an affine transform, as show in Fig. 3 [1]. The ROM method is easier to understand as it is simply an 8 input memory with pre-fixed values. The SubBytes step is nonlinear.

$$\begin{bmatrix}
b_0 \\
b_1 \\
b_2 \\
b_3 \\
b_4 \\
b_5 \\
b_6 \\
b_7 \\
\end{bmatrix} = \begin{bmatrix}
1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
\end{bmatrix} \begin{bmatrix}
b_0 \\
b_1 \\
b_2 \\
b_3 \\
b_4 \\
b_5 \\
b_6 \\
b_7 \\
\end{bmatrix} + \begin{bmatrix}
1 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
\end{bmatrix}$$

Fig. 3. SubBytes step. The 8x8 matrix performs the the multiplicative inverse in $GF(2^8)$ and the matrix added afterwards is the affine transform [1].

2) ShiftRows: The shift rows step is just a simple reordering of the bytes, as shown in Fig. 1. For each row of the State numbered row 0 to row 3, the bytes are shifted right according to the row number, i.e. bytes in row 0 are shifted right 0 places, row 1 are shifted right 1 place etc. These shifts wrap around.

3) MixColumns: In this step each column of the State is multiplied by a 4x4 matrix, as shown in Fig. 4. This can be implemented in hardware using a series of XOR gates thanks to the multiplication being carried out $GF(2^8)$. For example multiplying by 3 can be broken down to multiplying by 2 and then XORing the result with 1, since $3 = 2 \oplus 1$, $GF(2^8)$.

4) AddRoundKeys: In the AddRoundKeys step, each word in the state is added to the Round Key using a bitwise XOR. The Round Key is generated using the Key Scheduler.

5) Key Scheduler: The Key Scheduler takes the Cipher Key (128, 192, or 256 bits) as input and expands it out into
Fig. 4. MixColumn step. Each column of the State is multiplied by a 4x4 matrix as shown [1].

\[
\begin{bmatrix}
1 & 2 & 3 & 4 \\
5 & 6 & 7 & 8 \\
9 & 10 & 11 & 12 \\
13 & 14 & 15 & 16
\end{bmatrix}
\]

A key schedule. The key schedule has some similarities to the previous steps. For a given four words of round keys, \(w_i, w_{i+1}, w_{i+2}, w_{i+3}\), the following four words are calculated as follows:

\[
\begin{align*}
w_{i+4} &= w_i \oplus g(i + 3) \\
w_{i+5} &= w_{i+4} \oplus w_{i+1} \\
w_{i+6} &= w_{i+5} \oplus w_{i+2} \\
w_{i+7} &= w_{i+6} \oplus w_{i+3}
\end{align*}
\]

The calculation for \(w_{i+4}\) is a little more complicated, as it includes the \(g()\) function. This function has the following three steps:

1) Shift each byte one to the left
2) Use an S-Box on each byte returned from the previous step
3) XOR these bytes with the round constant.

The round constant is calculated as follows:

\[
\begin{align*}
RC[1] &= 0x01 \\
RC[i] &= 0x02 \times RC[i - 1]
\end{align*}
\]

These Round Keys can either be precomputed for each new key before the actual AES encryption is carried out, or the Round Keys can be computed on the fly for each round. Each method has pros and cons. And obvious con to pre computation is the need for additional memory to store the keys, where as on the fly computation can burn power on subsequent plain text blocks where the key has remained constant. In many cases far more than 128 bits of data needs to encrypted or decrypted for each key.

B. Low Power Circuit Methods

In this project, to achieve a low power AES implementation the following low power circuit methods are going to be investigated:

1) Low Power and High Vt Transistors: It is possible to use different gate libraries in the design phase. These libraries are supplied with the technology design kit and each library changes the properties of the implemented design. Instead of general purpose (GP) transistors, using low power (LP) transistors will decrease the total power consumption of the design. In addition, using high \(V_T\) (HVT) transistors decrease the leakage current and power extensively compared to standard \(V_T\) (SVT) transistors.

2) Voltage and Frequency Scaling: Dynamic power is proportional to the square of the operating voltage and frequency is directly proportional to the supply voltage. Therefore, with voltage and frequency scaling a cubic power reduction is possible. However, the delay of the circuit is also dependent on the supply voltage. Thereby, reducing the supply voltage increases the execution time. To achieve power reduction via voltage and frequency scaling, the timing constraints have to be met. According to [16] the most dynamic power efficient way is executing as slow as possible while just meeting the timing constraints.

3) Clock Gating: In synchronous systems, the clock signal is used in the majority of the circuit blocks. If a clocked circuit block is not used all the time, it will end up consuming redundant dynamic power. In addition, the part of the clock tree that supplies the clock to these unused circuit blocks will consume dynamic power with a switching factor of 1. Clock gating is achieved by adjusting the clock network to trigger only the required circuit blocks, which will result in consuming less dynamic power [16].

4) Power Gating: Another method to reduce power is to decrease leakage power. If one cuts the power supply to a certain circuit block, that block’s leakage power will be reduced without changing the type of the transistors in that block. Power gating is turning off the supply power temporarily. Therefore, those blocks will be in low power mode and their leakage power will be reduced [16]. Fig. 5(a) shows the power levels of the circuit block when it is activated, whereas fig. 5(b) shows the power levels when it is deactivated.

5) Burst Mode: Burst mode is a variation of clock gating. While having the same architecture for clock gating, the sleep transistors are HVT transistors to reduce the leakage current and power, and to increase the performance, the circuit block is
designed with low $V_T$ (LVT) transistors. This way the dynamic and leakage power will be reduced, and at the same time the performance will be increased. Fig. 6 shows the design for burst mode.

![Burst mode diagram](image)

Fig. 6. Burst mode [17]

IV. TIME-LINE AND DELIVERABLES

For the first 2 weeks, different low power circuit methods will be explored and the current literature results will be compared. Following 5 to 7 weeks, implementation of the following designs will take place:

1) RTL synthesis of base design with GPSVT
2) RTL synthesis of base design with GPHVT
3) RTL synthesis of base design with LPSVT
4) RTL synthesis of base design with LPHVT
5) Custom schematic design of burst mode

If time allows, further design techniques will be implemented. For the last 3 weeks, testing of the implemented designs, gathering data and comparing results will be done.

V. CONCLUSION

The aim of this project is to implement different low power AES designs and compare the low power circuit techniques which are used in the designs. This will be achieved through synthesis and full custom schematic design. As shown, AES is a flexible, yet secure algorithm, which enables the implementation of these different low power circuit techniques. Since AES is one of the most widespread used security algorithms, comparison of different low power circuit implementations of AES will be useful for future low power AES research and AES hardware implementations.

REFERENCES


