School of Electrical, Electronic and Computer Engineering

Efficient AES Lookup Implementation using Normal Basis

F. Burns, J. Murphy, A. Koelmans and A. Yakovlev

Technical Report Series

NCL-EECE-MSD-TR-2008-128

May 2008

Contact:
f.p.burns@ncl.ac.uk
j.p.murphy@newcastle.ac.uk
albert.koelmans@newcastle.ac.uk
Alex.Yakovlev@ncl.ac.uk

Supported by: EPSRC grants GR/S81421 and EP/F016786/1

NCL-EECE-MSD-TR-2008-128

School of Electrical, Electronic and Computer Engineering,
Merz Court,
University of Newcastle Upon Tyne,
Newcastle Upon Tyne, NE1 7RU, UK

http://async.org.uk/

Efficient AES Lookup Implementation using Normal Basis

F. Burns, J. Murphy, A. Koelmans and A. Yakovlev

Abstract

This paper presents a new type of AES implementation using normal basis. The method is based on a lookup technique which makes use of inversion and shift registers which leads to a smaller size of lookup for the S-box than its corresponding implementations. The reduction in lookup size is based on grouping sets of inverses into cosets which in turn leads to a reduction in the number of lookup values. The above technique is implemented in a regular AES architecture using register files which requires less interconnect and is suitable for security applications. The results for the implementation are competitive in throughput and area compared with the corresponding solutions in polynomial basis.

1 Introduction

The current NIST advanced encryption standard is the symmetric block cipher Rijndael [1]. The AES is the preferred algorithm for implementations of cryptographic protocols that are based on a symmetric cipher. It is currently designed to process data blocks of 128 bits, using keys of lengths 128, 192 and 256 bits. The mathematics behind the AES is centered on Galois arithmetic making use of transformations based on inversion and multipication.

A variety of ways have been attempted for implementing the AES standard efficiently. These range from implementations which aim to achieve high throughput [2] to implementations which achieve low area [3]. In [4] various FPGA architectures are presented to improve the throughput of the AES. In [5], [6] and [7] various efficient area implementations are described of the AES in ASIC covering architectures of differing bit-width. Our approach targets the latter low area level implementation domain.

This paper describes a novel architecture for the AES based on normal basis rather than polynomial basis. It makes use specifically of the inverse calculation in GF. Normal basis is frequently used in cryptographic applications for providing efficient implemenations [8]. Hardware implementations using normal basis arithmetic typically have less power consumption than other bases. This is particularly true in the case of the squaring operation in normal basis which requires only a rotation operation.

Efficient software and hardware implementations of the basic arithmetic operations (addition, multiplication and inversion) in the Galois field $GF(2^m)$ are desired in cryptography. This is particularly relevant in the case of the AES. Many AES architectures attempt compaction of the S-box [9][10] or the inverse function [11] to improve the overall performance. The S-box operation is the largest device and requires more area in general than the other operations. Different approaches have been attempted for S-box compaction varying from the use of subfields [12] to the use of lookup techniques. Recent approaches have made use of lookup for the S-box [13] but this tends to consume large amounts of area. For this reason research into optimization of the S-box [14] is important in its own right.

In this paper we focus on reducing the size of the inverse operation used in the S-box but using normal basis rather than polynomial basis. Several attempts have been made in normal basis to find an efficient implemention for the inverse function [15][16]. However, these make use of a multiplicative approach, incorporating several multipliers, which in turn leads to more area consumption. The architecture presented here makes unique use of the squaring operation and lookup to provide for a more efficient architecture.

The lookup technique presented here is incorporated into a regular architecture which makes use of register files. These are used for containing the intermediate AES State. This reduces the overall level of interconnect as it obviates the need for an explicit ShiftRows operation later, thereby reducing the area. In addition, the design here is based on a regular architecture, as it is considered an important aspect of design for security, which facilitates security implementation.

2 Normal Basis Preliminaries

The basis chosen here for implementing the AES is the normal basis. First we introduce some basic theory. In the following it is assumed that p is a prime number, q is a power of p and that F_q denotes a finite field of q elements. The characteristic of F_q is p. The field F_{q^n} is always considered as an n-dimensional extension of F_q and is thus a vector space of dimension n over F_q. The Galois group of F_{q^n} over F_q is cyclic and is generated by the Frobenius mapping $\sigma(\alpha) = \alpha^q, \alpha \in F_{q^n}$.

The polynomial basis is a basis for finite extensions of finite fields. Let $\alpha \in GF(p^m)$ be a root of a primitive polynomial of degree m over $GF(p)$. The polynomial basis of $GF(p^m)$ is then $\{1, \alpha, ..., \alpha^{m-1}\}$.

A normal basis of F_{q^n} over F_q is a basis of the from $N = \{\alpha, \alpha^q, ..., \alpha^{q^{m-1}}\}$, i.e. a basis consisting of all the algebraic conjugates of a fixed element. We say that α generates the normal basis N, or α is a normal element of F_{q^n} over F_q. In either case we are referring to the fact that the elements $\alpha, \alpha^q, ..., \alpha^{q^{m-1}}$ are linearly independent over F_q. For the normal basis $\{\alpha_0, \alpha_1, ..., \alpha_{n-1}\}$, it is assumed that $\alpha_i = \alpha^{q^i}$ for $\alpha \in F_{q^n}$ with $i = 0, 1, ..., n-1$.

Assume a base element $\alpha_0 = 010$ taken from the trinomial $x^3 + x^2 + 1$ over $GF(2^3)$. The following values can be derived from α_0 by consecutive squaring $\alpha_1 = 100$, $\alpha_2 = 111$. Because these three values are linearly independent they may be used in the formation of a normal basis. All other elements being linearly dependent on these elements can be formed from a linear combination of them. For example, in normal basis the element 110 equates to $111 + 100 = 011$ in polynomial basis.

Squaring or raising to a power of 2^n in normal basis equates to a simple rotation of the bits. The following table shows a comparison between the squaring operation for the trinomial $x^3 + x^2 + 1$ in polynomial basis and in the normal basis starting from α=010. It can be seen that a simple rotation is needed in normal basis to square its equivalent value in polynomial basis.

Table 1: Squaring Operation and Inverse in Polynomial and Normal Basis.

Polynomial Basis (Inverse)	Normal Basis (Inverse)
010 (110)	001 (011)
100 (011)	010 (110)
111 (101)	100 (101)

The multiplicative inverse of a value α, denoted $1/\alpha$ or α^{-1}, is the number which, when multiplied by α, yields 1. Assuming that α(x) stands for the polynomial representation of the field element α, the multiplicative inverse of a field polynomial value $\alpha(x)$ is found from the following equation $\alpha^{-1}(x) \cdot \alpha(x) \, mod \, m(x) = 1$ where $m(x)$ is the irreducible polynomial used for generating the field.

We can combine the operations for inverse and squaring or raising to a power of n. The sequence of these two operations does not affect the result. This is expressed using the following equation.

$$(\alpha^{-1})^n = (\alpha^n)^{-1} \tag{1}$$

Because the squaring operation in normal basis is cyclic (i.e. rotation of bits) it forms a cyclic set. Because of the commutativity implied by Eq. 1, for each set of squares that a value belongs to there must

also be a corresponding inverse value which exhibits similar behaviour under squaring. This can be seen in Table 1 where consecutive rows are squared and the inverse of each value appears in brackets to the right. We refer to each of the cyclic sequences appearing in each column as a coset. Examination of these values for the whole field leads to the following useful observations.

Lemma 1 *In Normal Basis, if an element α in coset A has an inverse in another coset B, then each of the remaining values in coset A formed from α by the squaring operation must have an inverse value in coset B.*

This implies that the ordered coset A in this case has a corresponding ordered coset B containing all the inverses of coset A.

Lemma 2 *In Normal Basis, if an element α in coset A has an inverse in the same coset A, then each of the remaining values in coset A formed from α by the squaring operation must have an inverse value in coset A also.*

This implies that the ordered coset A in this case contains all of the inverses of each of its elements.

The above observations are used for defining the core inversion unit for the AES which is described in the following section.

3 Basic Architecure

The proposed architecture for the AES is based on a mixed asynchronous/synchronous pipelined lookup architecture. The architecture has a blocksize of 128-bits and a 32-bit width datapath. The diagram for the architecture is shown in Fig. 1.

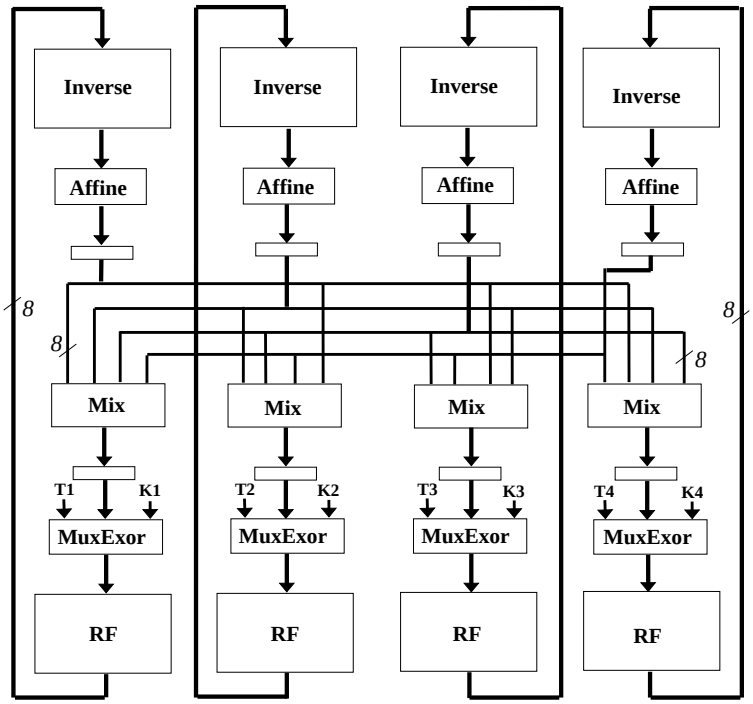

Figure 1: AES architecture.

At the top of Fig. 1 four inversion units are used followed by an affine transformation. Each inversion unit loads a value from one of the four Register Files (RF) shown at the bottom which contains the AES State. Each inversion unit includes a 34×8-bit lookup table based on the theory outlined in subsection 3.1. After inversion the outputs of the inversion units are passed to a normal basis affine block and each affine result is subsequently stored in an 8-bit register. The output from this forms the input to the mix operation below. There are four 8-bit identical mix units that are used to carry out an appropriate mix operation (described in subsection 3.3). Once the mix operation has occured the output word formed from concatenating the output of the mix units is then XORed with the key and written to the RFs.

The ShiftRows operation [1] is not shown here as this is implemented implicitly by the appropriate selection of values from the RFs which is explained in subsection 3.4.

3.1 Normal Basis Inversion

The theory in section 2 can be used for deriving the basic inversion architecture. It is based on the correspondence between the different cosets. This correspondence makes it possible to define an inversion architecture based on register rotation and lookup values. It makes use of one lookup value from a coset and its corresponding inverse either from another or the same set as follows:

(1) If a coset value contains its inverse in another coset then both the coset value and the corresponding inverse value may be used as lookup values.

(2) If a coset value has its inverse in its own set then only this coset value is used as a lookup value.

Values chosen using (1) or (2) are referred to as coset leaders.

A diagram showing the basic inversion architecture is shown in Fig. 2.

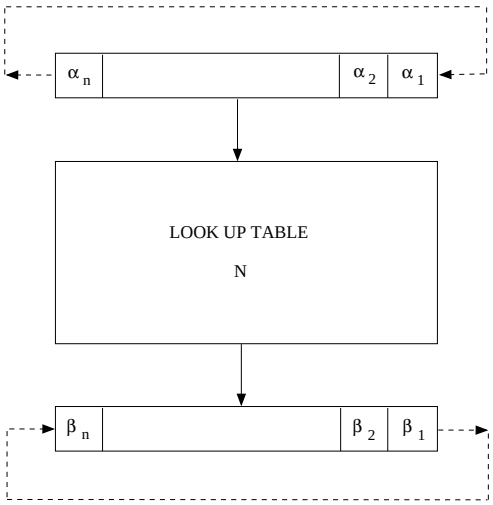

Figure 2: Lookup Architecture.

At the top of Fig.2. is a register. This operates using the rotation operation RL as defined below.

Definition 1 $RL(\alpha_n, \alpha_{n-1}, \alpha_{n-2}, ..., \alpha_2, \alpha_1) = (\alpha_{n-1}, \alpha_{n-2}, ..., \alpha_2, \alpha_1, \alpha_n)$.

At the bottom is a rotate right register. This operates in a similar way using the rotate operation RR.

Definition 2 $RR(\alpha_n, \alpha_{n-1}, ..., \alpha_3, \alpha_2, \alpha_1) = (\alpha_1, \alpha_n, \alpha_{n-1}, ..., \alpha_3, \alpha_2)$.

The architecture loads a normal basis field element into the top register and this is rotated repeatedly until the coset leader from its coset is found. The top register is governed by an asynchronous control signal which uses a test to see if a lookup value has been found. When this value is found the bottom register is loaded with the lookup value and the top register is loaded with a new value. The bottom register is rotated right the same number of times until the inverse of the original value in normal basis is contained in the bottom register.

The lookup table is formed from a set of coset leaders. To find a set of coset leaders the following algorithm can be used, where FL contains the set of values from $GF(2^n)$. CL contains the set of coset leaders which are cumulatively selected from FL.

```
CL=φ
While(FL<>φ)
  search new coset value c ∈ FL
  if c does not have coset leader cl ∈ CL
    CL = CL + c;
    FL = FL - c;
  else FL = FL - c;
```

As an example of how a coset lookup table is formed consider the polynomial $x^4 + x + 1$ in $GF(2^4)$. By applying the above algorithm to this field the following group of coset leaders in normal basis can be found 01(4), 0β(4), β1(4), 11(2), $\beta\beta$(1), 00(1). The number in brackets gives the number of values in each corresponding coset.

The inverse table for the field poynomial is shown in Table 2. In Table 2 the following compact representation is used $\{0 = 00, 1 = 01, \alpha = 10, \beta = 11\}$.

Table 2: Normal Basis Inverse Table for $x^4 + x + 1$.

$x \setminus y$	0	1	α	β
0	00	10	α0	β1
1	01	$\alpha\alpha$	$\alpha\beta$	β0
α	0α	$\beta\alpha$	11	1α
β	1β	0β	α1	$\beta\beta$

For the above example five lookup table values are required. Zero is not included as this is the default value which is used if no lookup value can be found. The derived lookup table is shown in Table 3.

Table 3: Coset Leaders and Inverses in Normal Basis.

Coset leader	Inverse	Coset leader	Inverse
01	10	11	$\alpha\alpha$
0β	β1	$\beta\beta$	$\beta\beta$
β1	0β	-	-

As an example, suppose we wish to use the lookup table to find the inverse of α1 (normal basis) for $x^4 + x + 1$. From Table 2, the inverse of α1 in normal basis is $\beta\alpha$. The value α1 has to be rotated left once to find its coset leader 0β which appears in the left of Table 3. The inverse of the coset leader in Table 3 is β1. Rotating this right once gives $\beta\alpha$ which is the required inverse.

Table 4 shows the type and number of lookup values for polynomials of different orders. The first two columns give the order and the polynomial. The terms(entries) column shows the number of cosets together

with their size in brackets. The final two columns show the total coset size and the normal size of the polynomial field in terms of its entries.

Table 4: Lookup Size.

Order	Polynomial	Terms(entries)	Lookup size	Field size
$n=3$	x^3+x^2+1	2(3),1(1)	3	8
$n=4$	x^4+x^3+1	3(4),1(2),1(1)	5	16
$n=5$	$x^5+x^4+x^2+x^1+1$	6(5),1(1)	7	32
$n=6$	x^6+x^5+1	9(6),2(3),1(2),1(1)	13	64
$n=7$	x^7+x^6+1	18(7),1(1)	19	128
$n=8$	$x^8+x^4+x^3+x^1+1$	30(8),3(4),1(2),1(1)	35	256

We need to make use of the following fact when calculating the lookup size. If the sequence of terms $\alpha_1, \alpha_2, \alpha_3, ..., \alpha_n$ are arranged in a circle then any one of the rotated sequences $\alpha_2, \alpha_3, ...\alpha_n, \alpha_1, \alpha_3, ..., \alpha_n, \alpha_1, \alpha_2$ etc. may be thought of as defining the same circular sequence. But not all n linear sequences corresponding to the same circle need be different. If for a divisor d of n, the sequence $\alpha_1, \alpha_2, ..., \alpha_n$ consists of a sequence of d terms $\alpha_1, \alpha_2, ..., \alpha_d$ repeated n/d times, the linear sequences repeat after the first d. With each circular sequence of length n we may associate a unique minimum period such that the circular sequence consists of n/d repetitions of a sequence of d terms.

Assuming different numbers of 0 and 1 digits in the sequence $\alpha_1, \alpha_2, ..., \alpha_n$, we can use Moreau's necklace-counting function to calculate the non-periodic sequences of length n. In combinatorial mathematics Moreau's necklace-counting function is defined by the following equation.

$$M(\alpha, n) = \frac{1}{n} \sum_{d|n} \mu\left(\frac{n}{d}\right) \alpha^d \tag{2}$$

where $d|n$ implies the positive divisors d of n, α represents the number of different digits $\{0, 1\}$ and μ is the classic Mobius function defined below.

Definition 3 *The Mobius function $\mu(n)$ is equal to one of the following:*

- $\mu(n) = 1$ *if n is a square-free +ve integer with an even number of distinct prime factors.*
- $\mu(n) = -1$ *if n is a square-free +ve integer with an odd number of distinct prime factors.*
- $\mu(n) = 0$ *if n is not square-free.*

Thus $\mu(1) = 1$ and $\mu(prime) = -1$. A square-free integer is one divisible by no perfect square, except 1. The first integers that are not square-free are $\{4, 8, 9, 12, 16, 18, 20\}$. Consider the non-repetitive case where $n = 6$. Equation 2 equates to $\frac{1}{6}\left\{\mu(6) \cdot 2^1 + \mu(3) \cdot 2^2 + \mu(2) \cdot 2^3 + \mu(1) \cdot 2^6\right\} = \frac{1}{6}\left\{2 - 2^2 - 2^3 + 2^6\right\} = \underline{9}$. In Table 4 in the terms(entries) column the first term shows the number of cosets of period n which are non-repetitive within this period i.e. 2, 3, 6, $\underline{9}$, 18 and 30.

Using equation 2 it is possible to derive equation 3, which is used to find the number of repetitive cosets with period $d < n$ (mid terms in the terms(entries) column).

$$rep\_cos = \sum_{d|n} \{M(\alpha, d)\} \tag{3}$$

Here each non-periodic sequence of length d forms a unique non-repetitive sequence in repetitive sequence n. The total lookup size can be found using equations 2 and 3.

The irreducible polynomial generator used for the inverse for the AES is $x^8 + x^4 + x^3 + x + 1$ in $GF(2^8)$ which appears in the last row of Table 4. For the AES the lookup table generated has a size of 35. The corresponding group of coset leaders for the AES in normal basis is shown in Appendix A. As before the number in brackets gives the number of values in the corresponding coset. The normal basis inverse table generated for the AES field polynomial, formed using $\alpha = 33$, is shown in Appendix B.

3.2 Reduced Lookup and Timing

The lookup table for the AES table can be modified by rotating the values to derive new values whilst maintaining the inverse relation between the input and output. The logic for the lookup can be reduced by rotating values in the coset leader table so that matching bits coincide in specific columns. This is arranged here so that the first two bits are set to the values 0 and 1 in each row. By rotating the coset leaders of Appendix A, a new group of coset leaders can be found. The table of new rotated lookup values with inverses is shown in Appendix C. Each new coset leader in Appendix C now has bits 7 and 6 set to 0 and 1. The value FF is removed from the table, reducing it to 34 values. The FF value is detectable if no match is found. The gate equivalent (GE) value, in terms of basic gates, for the rotated lookup after logic reduction is 134.

The timing for lookup makes use of a clocking scheme that is based on two clocks: the main clock Tc and a faster clock that is half its period τc. The clocks work in combination with the asynchronous control signals. The faster clock τc is used for clocking the shift registers. These are shifted 2 bits at a time in a τc cycle and the first two relevant pairs of bits appearing in the first three bits from left to right are tested. This test is based on whether the first two bits of either of the first pair of values have been set to 0 and 1. If either of the pairs is set, a lookup is made on the value which has this setting. If either combination is not apparent for the first two pairs, then a double shift is made which requires no lookup. The actual lookup is executed using the slower clock Tc. The critical path for the lookup is 8 gates. Out of 256 values there are 64 bytes which have the 0 and 1 combination which require lookup, and 34 of these are coset leaders. This reduces the overall lookup effort considerably.

3.3 Normal Basis Affine and Mix

An affine transformation must be applied to the output of each inversion unit (Fig. 2). The original specification requires the following affine transformation (over $GF(2^8)$):

$$b_i = b_i \oplus b_{(i+4)mod8} \oplus b_{(i+5)mod8} \oplus b_{(i+6)mod8} \oplus b_{(i+7)mod8} \oplus c_i \tag{4}$$

for $0 \leq i < 8$, where b_i is the i^{th} bit of the byte, and c_i is the i^{th} bit of a byte c with the value $\{63\}$. This is depicted in matrix form in [1]. The affine transformation has an equivalent representation in normal basis. This can be found by substituting all polynomial basis values in the original matrix with normal basis values.

For the AES inversion there are 128 different normal basis representations. Each one of these will reduce to a different affine solution depending on the value of α that is chosen in the formation of the normal basis. Table 5 shows different affine solutions for different values of α. The resulting normal basis equations generated for the affine transformation for $\alpha = 33$ results in 14 gates. This represents a reduction in logic over the polynomial basis.

The mix operation operates on the state column-by-column. The columns are considered as four-term as polynomials over $GF(2^8)$ and are multiplied by a fixed polynomial [1]. As a result of the multiplication, the four bytes in a column are replaced using equations of the following form:

$$\begin{aligned}
s'_{0,c} &= 02 \bullet s_{0,c} \oplus 03 \bullet s_{1,c} \oplus s_{2,c} \oplus s_{3,c} \\
s'_{1,c} &= s_{0,c} \oplus 02 \bullet s_{1,c} \oplus 03 \bullet s_{2,c} \oplus s_{3,c} \\
s'_{2,c} &= s_{0,c} \oplus s_{1,c} \oplus 02 \bullet s_{2,c} \oplus 03 \bullet s_{3,c} \\
s'_{3,c} &= 03 \bullet s_{0,c} \oplus s_{1,c} \oplus s_{2,c} \oplus 02 \bullet s_{3,c}
\end{aligned}$$

Table 5: Affine & Multiplication solutions for different values of α.

α	Affine x-or gates	Multiply x-or gates
65	20	21
61	18	20
36	16	18
63	16	18
33	14	16
$2E$	13	17
26	13	19

The mix operation requires multiplication by 02 and multiplication by 03. The right hand side of Table 5 shows different solutions for multiplication by 02 for different values of α. Multiplication by 02 for $\alpha = 33$ results in 16 gates. Multiplication by 03 is derived from XORing by multiplication by 02.

3.4 Register Files

The RF layout is shown in Fig. 3. Four register files are used, RF1 .. RF4, each containing four, 8-bit registers, which are used for containing the representation of the state.

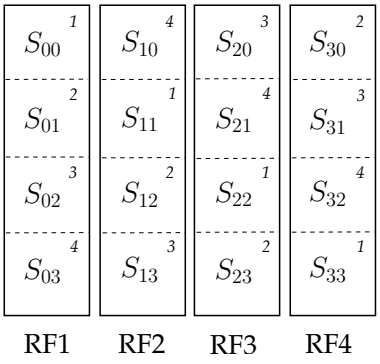

Figure 3: Register file layout.

An element of the state is denoted by S_{xy}. The plaintext is initially XORed with the first key and is input to the RFs as follows: $1/\{S_{00}, S_{10}, S_{20}, S_{30}\}$, $2/\{S_{01}, S_{11}, S_{21}, S_{31}\}$, $3/\{S_{02}, S_{12}, S_{22}, S_{32}\}$, $4/\{S_{03}, S_{13}, S_{23}, S_{33}\}$. In this way each row as specified in [1] is stored in its own RF.

When the RF values are accessed by the inversion units they are not read as original words from left-to-right. The values are grouped and read in the following order $1/\{S_{00}, S_{11}, S_{22}, S_{33}\}$, $2/\{S_{01}, S_{12}, S_{23}, S_{30}\}$, $3/\{S_{02}, S_{13}, S_{20}, S_{31}\}$, $4/\{S_{03}, S_{10}, S_{21}, S_{32}\}$, which refers to the order with which the bytes are read from the register files by the inversion units. Each ordered group accessed by the four inversion units corresponds to the required state elements that need to be mixed together in order to form a new word. This selection implies that the sequence of each row of the state already corresponds to the pre-shifted row of the state. This pre-shifting obviates the need for an explicit ShiftRows operation later on. This lookahead means that the values to be mixed are sent in groups to the inversion unit directly from the RFs. This reduces the level of interconnect considerably.

3.5 Key generation

The same datapath as for the main algorithm may be used for key generation. This is made possible by extending the register files to create storage for the key. S-box operations are required for the key expansion. Since the data unit does not perform an S-box operation when the last MixColumns and AddRoundKey transformations are executed, the S-box operations of the datapath can be used when they become available towards the end of each round. A multiplexor in the MixColumns is used to miss out the Mix operation. The XOR-block is adapted for the remaining key additions to produce the next key.

4 Comparisons

The AES encryption ASIC implementation is compared in this section. For our implementation the lookup tables were reduced to equations and implemented in standard gates. This was executed using our own in-house synthesis tool. The implementation technology used was Cadence ASIC $0.35\mu m$. Most other implementations are implemented using FPGA and therefore only the direct relevant ASIC comparisons are made here excluding FPGA. The other implementations have been implemented in varying process technologies. In addition the implementations that are used for comparison purposes are in polynomial basis. Table 6 shows the results for the implementation.

Table 6: AES Comparison

Implementation	Width [bits]	Area [kgates]	Process [μm]	Frequency [MHz]	Block Clock cycles	Throughput [Mbps]
NB	32	4,100	0.35	132	108	156
[3]	32	8,200	0.6	50	64	128
[3]	32	12,894	0.6	50	34	241
[6]	32	5,400	0.11	131	54	311
[7]	8	3,200	0.13	130	160	104
[7]	8	3,100	0.13	152	160	121
[18]	8	3,400	0.35	na	1032	na

In the table, the datapath widths are shown in terms of the numbers of bits processed in parallel. Two sizes are used for all implementations, either 32-bits or 8-bits. The next column shows the areas in terms of their gate equivalent. The implementation technology for each is shown in the process column which ranges from $0.11\mu m$ to $0.16\mu m$. The next column show the clock frequency in MHz and the next the Block Clock Cycles. The Block Clock Cycles column shows the number of clock cycles to process one plaintext input. Finally the throughput column is shown in Mbps.

Comparing the results, it can be seen that the area for our implementation fares well against comparable implementations with a datapath width of 32-bits. The first set of results for Ref. [3] has approximately twice the area and a throughput which is 18% slower but which uses a slower process technology of $0.6\mu m$. The second set of results for Ref. [3] has a much larger area of approximately three times the value but has a throughput which is approximately 40% as fast. Although the Satoh throughput is high [6] it uses a faster technology and the critical path is known to be longer, meaning an improvement would be apparent in our implementation with an upgrade in technology.

The gate equivalent count for our normal basis implementation comes to 4100. The majority of the area is taken up by the storage, which includes the data and key and which accounts for approximately 45% of the area. The next largest area is taken by the shifting and lookup logic which accounts for approximately 25% of the total area. The control represents about 8% of the total area.

The bottle-neck in terms of time for our implementation comes from the inverse function. The main clock Tc which is used for the lookup operation is set at 132 MHz. The lookup block which has a size of 134 gates and a low critical path of 8 gates is used for determining this. The smaller clock used in our implementation τc is set at half this size. The total inverse time is determined by the rotation time together with lookup time. Because the number of rotations required for each value entering the inversion unit is different, and lookup reduction is also used, an average estimate for time is made. The average time is derived from a simulation of a dataset consisting of several thousand inputs.

The comparisons against implementations with datapath widths of 8 bits show that they are more efficient than our implementation in terms of area but not by too large a percentage. However, this is generally at the expense of a slower throughput, which is much slower, which makes our implementation competitive. The explanation for the slower throughput is straightforward as the bottle-neck for 8-bit wide implementations comes from the mix operation (assuming the S-box can be pipelined) which can only execute 8-bits at a time.

5 Conclusions

We have presented a novel efficient way of implementing the AES algorithm in Normal Basis using an approach which includes shift registers and lookup tables. This makes use of the commutative relationship between inverse and square and lookup values. The results are promising, particularly for the AES which makes use of inversion in the field $GF(2^8)$. As a result, the look up size for inversion has been reduced and minimal lookup tables in terms of their GE are used. This represents a considerable reduction over previous architectures which use lookup.

A low cost implementation of the AES has been presented which targets a minimal number of gates. The gate equivalent size is less than other presented work for datapath widths of a similar size and competitive against those which are not. This means that it is competitive from an area-time perspective after taking into account the critical path and process technology. The number of lookup accesses are reduced thereby improving the latency. The regular RF design means that the interconnect and area is also reduced. The regular architecture is considered an important aspect of design for security, which facilitates security implementation.

Acknowledgments

References

[1] Nat'l Inst. of Standards and Technology, "Federal Information Processing Standard 197, The Advanced Encryption Standard (AES)," *http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf,2001.*

[2] T. Lin, C. Su, C. Huang and C. Wu, "A High-Throughput Low-Cost AES Cipher Chip," *IEEE Proc. of the 3rd Asia-Pacific Conf. on ASICS (AP-ASIC)*, Aug. 2002.

[3] S. Mangard, M. Aigner and S. Dominikus, "A Highly Regular and Scalable AES Hardware Architecture," *IEEE Trans. Comp.,* Vol 52, no. 4, pp. 483-491, Apr. 2003.

[4] A. Elbirt, W. Yip, B. Chetwynd and C. Paar, "An FPGA-Based Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists," *IEEE Trans. VLSI Systems,* Vol 9, no. 4, pp. 545-557, Aug. 2001.

[5] Y. Huang, Y. Lin, K. Hung and K. Lin, "Efficient Implementation of AES IP," *Circuits and Systems 2006, APCCAS IEEE Conference,* pp. 1418-1421, 2006.

[6] A. Satoh, S. Morioka, K. Takano and S. Munetoh, "A Compact Rijndael Hardware Architecture with S-box Optimization," *Proc. Advances in Cryptology - ASIACRYPT 2001,* pp. 239-254, 2001.

[7] P. Hämäläinen, T. Alho, M. Hännikäinen and T. Hämäläinen, "Design and Implementation of Low-area and Low-power AES EncryptionHardware Core," *Proc. 9th EUROMICRO Conference on Digital System Design (DSD'06),* pp. 577-583, 2006.

[8] T. Al-Somani and A. Amin, "Hardware Implementations of GF(2ˆm) Arithmetic Using Normal Basis," *Journalof Applied Sciences,* Vol 6, no 6, pp. 1362-1372, 2006.

[9] N. Yu and H. Heys, "Investigation of Compact Hardware Implementation of the Advanced Encryption Standard," *Proc. IEEE Conf. CCECE.,* Saskatoon, Saskatchewan, pp. 1069-1072, May 2005.

[10] I. Verbauwhede, P. Schaumont and H. Kuo, "Design and Performance Testing of a 2.29 GB/s Rijndael Processor," *IEEE Journal of Solid-State Circuits,* Vol. 38, no. 3, Mar. 2003.

[11] M. Jing, Y. Chen, Y. Chang and C. Hsu, "The Design of a Fast Inverse Module in AES," *Proc. Info-tech and Info-net, Cong. ICII.,* pp. 298-303, 2001.

[12] S. Tillich, M. Feldhofer and J. Großschädl, "Area, Delay, and Power Characteristics of Standard-Cell Implementations of the AES S-box," *Proc. Embedded Computer Systems: Architectures, Modelling, and Simulation,* LNCS 4017, pp. 457-466, Jul. 2006.

[13] M. McLoone and J. McCanny, "Rijndael FPGA Implementation utilizing Look-Up Tables," *Journal of VLSI Signal Processing Systems,* Vol 34, no. 3, Aug. 2003.

[14] D. Canright, "A Very compact S-box for AES," *Proc. 7th Int. Workshop on Cryptographic Hardware and Embedded Systems (CHES 2005),* LCNS 3659, pp. 441-455, 2005.

[15] N. Takagi, J. Yoshiki and K. Takagi, "A Fast Algorithm for Multiplicative Inversion in GF(2ˆm) Using Normal Basis," *IEEE Trans. Comp.,* Vol 50, no. 5, pp. 394-398, May 2001.

[16] J. Jeng, "Normal Basis Inversion in some Finite Fields," *Fifth International Symposium on Signal Processing and its Applications, ISSPA'99,* Brisbane, Australia, pp. 701-703, Aug. 1999.

[17] M. Feldhofer, J. Wolkerstorfer and V. Rijmen. "AES implementation on a grain of sand," *IEE Proc. Inf. Secur.,* Vol 152, no 1, pp. 13-20, 2005.

Appendix

A Coset values for $x^8 + x^4 + x^3 + x + 1$.

97(8), $B8$(8), $C9$(8), $2B$(8), EE(4), $E6$(8), 04(8), 75(8), $C1$(8), $B0$(8), 52(8), $D9$(8), $5A$(8), $9F$(8), $7D$(8), $0C$(8), 14(8), 33(4), $F6$(8), 87(8), 42(8),55(2), 23(8), $A8$(8), $6D$(8), $8F$(8), FE(8), $7A$(8), 98(8), $D1$(8), $C6$(8), $4D$(8), 88(4), 34(8).

B Normal Basis Inverse Table for $x^8 + x^4 + x^3 + x + 1$.

$x \setminus y$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	$2F$	$5E$	$B4$	BC	$7B$	69	$4E$	79	BA	$F6$	$B1$	$D2$	46	$9C$	$D9$
1	$F2$	44	75	31	ED	$D7$	63	56	$A5$	34	$8C$	$B0$	39	49	$B3$	$9E$
2	$E5$	57	88	86	EA	74	62	83	DB	$A3$	AF	$8B$	$C6$	$C0$	AC	01
3	$4B$	13	68	DD	19	$B6$	61	FD	72	$1C$	92	$E1$	67	$3E$	$3D$	$F3$
4	CB	DE	AE	91	11	$F5$	$0D$	52	$D5$	$1D$	$E8$	30	$C4$	AD	07	$8F$
5	$B7$	$7D$	47	$6B$	$5F$	AA	17	21	$8D$	$5C$	81	$9A$	59	84	02	54
6	96	36	26	16	$D0$	71	BB	$3C$	32	06	$6D$	53	$C2$	$6A$	FB	$A0$
7	$E4$	65	38	DF	25	12	$C3$	CC	CE	08	$7C$	05	$7A$	51	$E7$	CD
8	97	$5A$	BD	27	$5D$	$D8$	23	EC	22	98	EB	$2B$	$1A$	58	$A4$	$4F$
9	AB	43	$3A$	$C1$	$D1$	$C5$	60	80	89	EE	$5B$	FE	$0E$	$F0$	$1F$	$F9$
A	$6F$	$C8$	FA	29	$8E$	18	$D6$	$C7$	BE	$B5$	55	90	$2E$	$4D$	42	$2A$
B	$1B$	$0B$	$B8$	$1E$	03	$A9$	35	50	$B2$	EF	09	66	04	82	$A8$	$E6$
C	$2D$	93	$6C$	76	$4C$	95	$2C$	$A7$	$A1$	$E0$	$E2$	40	77	$7F$	78	FC
D	64	94	$0C$	$E3$	DA	48	$A6$	15	85	$0F$	$D4$	28	$F7$	33	41	73
E	$C9$	$3B$	CA	$D3$	70	20	BF	$7E$	$4A$	$F1$	24	$8A$	87	14	99	$B9$
F	$9D$	$E9$	10	$3F$	$F8$	45	$0A$	DC	$F4$	$9F$	$A2$	$6E$	CF	37	$9B$	FF

C Rotated Coset Leaders and Inverses in Normal Basis.

Coset leader	Inverse	Coset leader	Inverse	Coset Leader	Inverse
$5E$	02	$5A$	81	$6D$	$6A$
71	65	$7E$	$E7$	$7C$	$7A$
$4E$	07	$7D$	51	$7F$	CD
56	17	60	96	$7A$	$7C$
77	CC	50	$B7$	62	26
$6E$	FB	66	BB	47	52
40	CB	$6F$	$A0$	$6C$	$C2$
75	12	78	CE	$4D$	AD
70	$E4$	42	AE	44	11
61	36	55	AA	68	32
52	47	46	$0D$		
67	$3C$	51	$7D$		