Efficient advanced encryption standard implementation using lookup and normal basis

Published in IET Computers & Digital Techniques
Received on 30th May 2008
Revised on 24th October 2008
doi: 10.1049/iet-cdt.2008.0049

ISSN 1751-8601

Efficient advanced encryption standard implementation using lookup and normal basis

F. Burns J. Murphy A. Koelmans A. Yakovlev
School of Electrical and Computer Engineering, Merz Court, University of Newcastle Upon Tyne, Newcastle Upon Tyne NE1 7RU, UK
E-mail: f.p.burns@ncl.ac.uk

Abstract: A new type of advanced encryption standard (AES) implementation using a normal basis is presented. The method is based on a lookup technique that 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 the lookup size is based on grouping sets of inverses into conjugate sets 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 area and is suitable for security applications. The results of the implementation are competitive in throughput and area compared with the corresponding solutions in a polynomial basis.

1 Introduction

The current NIST advanced encryption standard (AES) 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 centred on Galois arithmetic making use of transformations based on inversion and multipication.

A variety of ways have been attempted to implement the AES standard efficiently. These range from implementations that aim to achieve high throughput [2] to implementations that achieve low area [3]. In [4], various FPGA architectures are presented to improve the throughput of the AES. In [5–7] 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 the 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 optimisation of the S-box [14] is important in its own right.

The architecture here compacts the size of the inverse operation used in the S-box but using the normal basis rather than the polynomial basis. Several attempts have been made in the 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 (RFs). These are used to contain the intermediate AES state. This reduces the overall level of interconnect as it obviates the need for an explicit ShiftRow 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. Power-balancing benefits from the regular architecture, lookup tables and registers, which are regular and more easily power-balanced.

Section 2 introduces some normal basis preliminaries. Section 3 gives a brief overview of the AES algorithm. Section 4 describes the AES architecture together with the inversion model. Section 5 gives comparisons and Section 6 provides some conclusions.

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

A normal basis of F_{q^n} over F_q is a basis of the form $N = \{\alpha, \alpha^q, \ldots, \alpha^{q^{m-1}}\}$, that is, 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 $\alpha, \alpha^q, \ldots, \alpha^{q^{m-1}}$ are linearly independent over F_q. For the normal basis $\{\alpha_0, \alpha_1, \ldots, \alpha_{n-1}\}$, it is assumed that $\alpha_i = \alpha^{q^i}$ for $\alpha \in F_{q^n}$ with $i = 0, 1, \ldots, n-1$.

Assume a base element $\alpha_0 = 010$ taken from the trinomial $x^3 + x^2 + 1$ over $\mathrm{GF}(2^3)$. 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 are linearly dependent on these elements and can be formed from a linear combination of them. For example, in the normal basis the element 110 equates to $111 + 100 = 011$ in the polynomial basis.

Table 1 Squaring operation in polynomial and normal basis

Polynomial basis	Normal basis
010	001
100	010
111	100

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

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 $\alpha(x)$ stands for the polynomial representation of the field element α, the multiplicative inverse of a field polynomial value $\alpha(x)$ denoted $\alpha^{-1}(x)$ is found from the following equation $\alpha(x) \cdot \alpha^{-1}(x) \bmod m(x) = 1$, 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 may be expressed using the following equation

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

Because the squaring operation in the normal basis is cyclic (i.e. rotation of bits); it forms a cyclic set. Because of the commutativity implied by (1), for each set of squares that a value belongs to there must also be a corresponding inverse value that exhibits similar behaviour under squaring. This can be seen in Table 2 where consecutive rows in both columns are squared and the inverse of each row on the left appears to the right. We refer to each of the cyclic sequences appearing in each column as a conjugate set.

Table 2 Squaring operation and inverse in normal basis

Normal basis	Normal basis inverse
001	011
010	110
100	101

Examination of these values for the whole field leads to the following useful observations.

Lemma 1: If an element α in conjugate set A has an inverse in another conjugate set B, then each of the remaining values in conjugate set A formed from α by the squaring operation must have an inverse value in conjugate set B.

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

Lemma 2: If an element α in conjugate set A has an inverse in the same conjugate set A, then each of the remaining values in conjugate set A formed from α by the squaring operation must have an inverse value in conjugate set A also.

This implies that the ordered conjugate set A in this case contains all of the inverses of each of its elements. The above observations are used to define the core inversion unit for the AES which is described in the following sections.

3 AES algorithm

The AES is a round-based, symmetric block cipher. The symmetric block cipher Rijndael was standardised by the National Institute of Standards and Technology in November 2001. It is defined for a block size of 128 bits and key lengths of 128, 192 and 256 bits. A block diagram of the AES subsystem is shown in Fig. 1.

The datapath of the AES is based on four different transformations that are performed repeatedly in a certain sequence over the state. The state consists of four rows of bytes, each containing N bytes, where N is the block length divided by 32. Each of the transformations, which are described in the following, maps a 128-bit input state to a 128-bit output state.

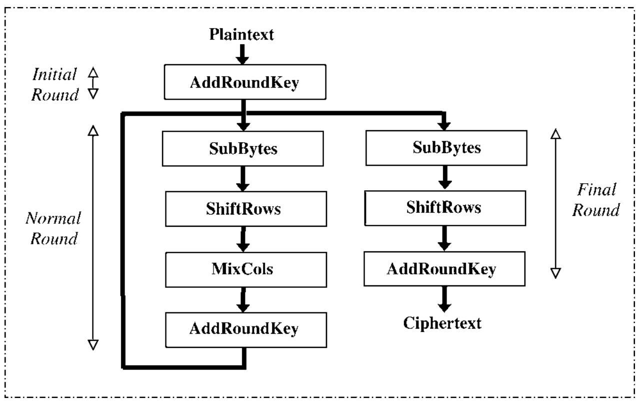

Figure 1 *AES subsystem*

SubBytes: The SubBytes transformation is a nonlinear substitution operation that works on bytes. Each byte of the input state is replaced using the same substitution function (called S-box). The S-box is defined as the multiplicative inverse in the Galois field GF(2^8) with the irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$ followed by an affine transformation.

ShiftRows: In the ShiftRows transformation, the bytes in the last three rows of the state are cyclically shifted over the different numbers of bytes (offsets). This is done according to the equation $S_{r,c} = S_{r,(c+\mathrm{shift}(r,\ N))\mathrm{mod}\ \mathrm{N}}$. The first row is not shifted.

MixColumns: The MixColumns transformation maps each column of the input state to a new column in the output state. Each input column is considered as a polynomial over GF(2^8) and multiplied with the constant polynomial $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$ mod $x^4 + 1$.

AddRoundKey: The AddRoundKey transformation is self-inverting. It maps a 128-bit input state to a 128-bit output state by XORing the input state with a 128-bit round key.

These transformations are applied to a 128-bit input block in a certain sequence to perform an AES encrytion. The transformations are grouped into so-called rounds. There are three different types of rounds, namely, the initial round, the normal round and the final round. The transformations of the different rounds and the sequence of the rounds are shown in Fig. 1. The initial round performs an AddRoundKey operation only. A normal round performs all of the operations. The final round performs all of the operations apart from the MixColumns operation. The number of rounds depends on the key size. For a 128-bit key, the number of rounds is 10.

4 Basic architecure

The proposed architecture for the AES is based on a pipelined lookup architecture that uses multiple clocks. The architecture shown has a blocksize of 128-bits and a 32-bit width datapath. The diagram for the architecture is shown in Fig. 2.

In Fig. 2 each inversion unit loads a value from one of the four RFs shown at the bottom, which contains the AES state. It is assumed that the AES state is already represented in the normal basis and no conversion from the polynomial basis is required. At the top of Fig. 2, the SubBytes operation requires inversion followed by an affine transformation. Four inversion units are employed. Each inversion unit includes a lookup table with 34 8-bit entries based on the theory outlined in Section 4.1. Each inversion unit after inversion subsequently outputs a value to its own register. The outputs of the inversion units are then passed to a normal basis affine block and each affine result is

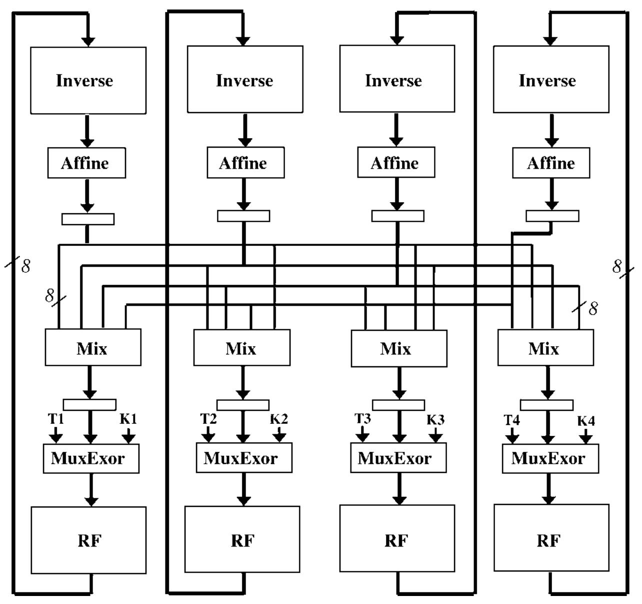

Figure 2 *AES architecture*

subsequently stored in an 8-bit register. The output from this forms the input to the MixColumns operation below. There are four 8-bit identical mix units that are used to carry out an appropriate mix operation (described in Section 4.4). 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 is not shown here as this is implemented implicitly by the appropriate selection of values from the RFs which is explained in Section 4.5.

4.1 Normal basis inversion

The theory in Section 2 can be used to derive the basic inversion architecture. It is based on the correspondence between the different conjugate sets specified in Section 2. 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 conjugate set and its corresponding inverse either from another or the same set as follows.

1. If a conjugate set value contains its inverse in another conjugate set, then both the conjugate set value and the corresponding inverse value may be used as lookup values.

2. If a conjugate set value has its inverse in its own set, then only this conjugate set value is used as a lookup value.

If a value is chosen as a lookup value using (1) or (2), it is referred to as a conjugate set leader.

A diagram showing the basic principles of the inversion architecture is shown in Fig. 3.

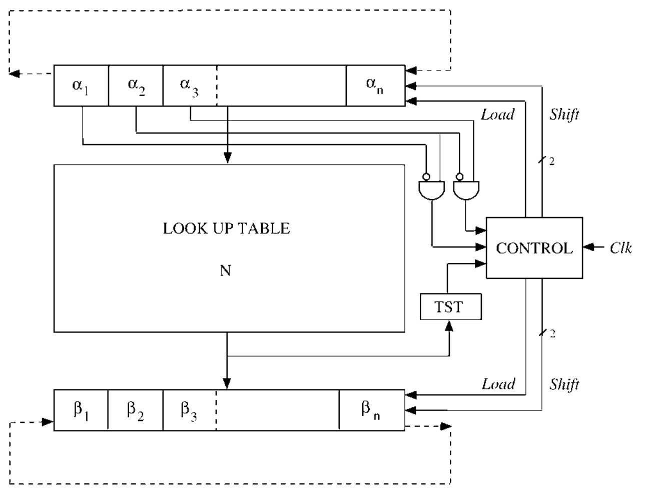

Figure 3 *Inversion architecture*

At the top of Fig. 3 is a register. This can operate using the rotation operation RL as defined below.

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

An equivalence relation can be defined in terms of this operator by the stipulation that two values are equivalent if one can be transformed into the other by possibly repeated applications of the RL operation.

At the bottom is a rotate right register. This can operate in a similar manner using the rotate operation RR as defined below.

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

The architecture loads a normal basis field element into the top register, and this is rotated repeatedly until the conjugate set leader from its conjugate set is found. The top register is governed by an asynchronous control signal which uses a test, TST (OR tree 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 the normal basis is contained in the bottom register.

At the centre of Fig. 3 is the lookup table, which is formed from a group of conjugate set leaders. As an example of how a conjugate set lookup table is formed, consider the polynomial $x^4 + x + 1$ in GF(2^4). The inverse table for this field poynomial is shown in Table 3.

The polynomial basis to the normal basis conversion table for this field polynomial is shown in Table 4.

To find a group of conjugate set leaders, the following algorithm (Fig. 4) can be used, where FL contains the set

Table 3 Polynomial inverse table for $x^4 + x + 1$

$x \backslash y$	00	01	10	11
00	0000	0001	1001	1110
01	1101	1011	0111	0110
10	1111	0010	1100	0101
11	1010	0100	0011	1000

Table 4 Polynomial to normal basis conversion table for $x^4 + x + 1$

$x \backslash y$	00	01	10	11
00	0000	1111	1001	0110
01	0011	1100	1010	0101
10	0001	1110	1000	0111
11	0010	1101	1011	0100

of values from GF(2^n); CL contains the set of conjugate set leaders which are cumulatively selected from FL.

By applying the algorithm to the above field, using Tables 3 and 4 to find the inverses, the following group of conjugate set leaders in the normal basis can be found 0001(4), 0011(4), 1101(4), 0101(2), 1111(1), 0000(1). The number in brackets gives the number of values in each corresponding conjugate set. For this example, five lookup table values are required. Zero is not included as this is the default value that is used if no lookup value can be found. The lookup table is shown in Table 5.

As an example, suppose we wish to use the lookup table to find the inverse of 0010 (polynomial basis) for $x^4 + x + 1$ in GF(2^4). The inverse of 0010 in the polynomial basis is 1001, which can be found using $\alpha(x) \times \alpha^{-1}(x) \bmod m(x) = 1$. Using Table 4 the normal basis of 0010 is 1001. This has to be rotated left once to find its conjugate set leader 0011 in the left of Table 5. The inverse of the conjugate set leader in Table 5 is 1101. Rotating this right once gives 1110. The polynomial basis value for this can be seen to be 1001 which is the required inverse.

```
CL=$\phi$
While(FL<>$\phi$)
  search new conjugate set value c $\in$ FL
  if c does not have conjugate set leader cl $\in$ CL
    CL = CL + c;
    FL = FL - c;
  else FL = FL - c;
```

Figure 4 *Conjugate set leaders*

Table 5 Conjugate set leaders and inverses in normal basis

Conjugate set leader	Inverse
0001	0100
0011	1101
1101	0011
0101	1010
1111	1111

Table 6 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 conjugate sets together with their size in brackets. The final two columns show the total lookup size and the normal size of the polynomial field in terms of its entries.

The example that follows is based on the inverse used for the AES encryption standard. 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 6. Table 7 shows the normal basis inverse table for this field polynomial formed using $\alpha = 33$.

By applying the selection algorithm described earlier in this section to the above example, the following group of conjugate set leaders in the normal basis can be found: 97(8), *B*8(8), *C*9(8), 2*B*(8), *EE*(4), *E*6(8), 04(8), 75(8), *C*1(8), *B*0(8), 52(8), *D*9(8), 5*A*(8), 9*F*(8), 7*D*(8), 0*C*(8), 14(8), 33(4), *F*6(8), 87(8), 42(8), 55(2), 23(8), *A*8(8), 6*D*(8), 8*F*(8), *FE*(8), 7*A*(8), 98(8), *D*1(8), *C*6(8), 4*D*(8), 88(4), 34(8) and *FF*(1). As before, the number in brackets gives the number of values in the corresponding conjugate set. For this example, 35 lookup values are required.

4.2 Reduced lookup and timing

The lookup table for the AES table can be modified by rotating the values to derive new values while maintaining the inverse relation between the input and output. The logic for the lookup can be reduced by rotating values in the conjugate set leader table so that matching bits coincide in specific columns. This is arranged such that bits 7 and 6 are set to the values 0 and 1 in each row. By rotating the conjugate set leaders of the previous section, the following group of conjugate set leaders can be found: 5*E*(8), 71(8), 4*E*(8), 56(8), 77(4), 6*E*(8), 40(8), 75(8), 70(8), 61(8), 52(8), 67(8), 5*A*(8), 7*E*(8), 7*D*(8), 60(8), 50(8), 66(4), 6*F*(8), 78(8), 42(8), 55(2), 46(8), 51(8), 6*D*(8), 7*C*(8), 7*F*(8), 7*A*(8), 62(8), 47(8), 6*C*(8), 4*D*(8), 44(4) and 68(8). Each new conjugate set leader now has bits 7 and 6 set to 0 and 1. The table of rotated lookup values with inverses is shown in Table 8. This indicates that the value FF is removed from the table, reducing it to 34 values. The FF

Table 6 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

Table 7 Hex normal basis inverse table for $x^8 + x^4 + x^3 + x + 1$

x\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

value is detectable if no match is found. The gate equivalent (GE) value, in terms of basic gates, for the rotated lookup after logic minimisation is 134.

The timing for the lookup makes use of a clocking scheme that is based on two clocks: a main clock Tc and a faster clock that is half its period, τc. The clocks work in combination with the 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 pair of consecutive values from left to right is tested. This test is based on whether the most significant 2 bits of the first value (bits 7 and 6) or the most significant 2 bits of the next value (bits 6 and 5) have been set to 0 and 1, respectively. If either one of the pair is set, a lookup is made on the corresponding value that has this setting. If either combination is not apparent for the first pair of values, then a double shift is made, which requires no lookup. A preliminary shift is made upon loading the top register based on the above test. 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 the lookup, and 34 of these are conjugate set leaders. This reduces the overall lookup effort considerably.

The inversion unit is pipelined to make both shift registers work simultaneously. The inversion architecture is

Table 8 Rotated conjugate set leaders and inverses in normal basis

Conjugate set leader	Inverse	Conjugate set leader	Inverse	Conjugate set leader	Inverse
5*E*	02	5*A*	81	6*D*	6*A*
71	65	7*E*	E7	7*C*	7*A*
4*E*	07	7*D*	51	7*F*	*CD*
56	17	60	96	7A	7C
77	*CC*	50	*B*7	62	26
6*E*	*FB*	66	*BB*	47	52
40	*CB*	6*F*	*A*0	6*C*	*C*2
75	12	78	*CE*	4*D*	*AD*
70	*E*4	42	*AE*	44	11
61	36	55	*AA*	68	32
52	47	46	0*D*		
67	3*C*	51	7*D*		

made to run efficiently so that it only uses up the necessary number of clocks required for each rotation. The clock signal for the top register is governed by a control signal which uses a test to see if a lookup value has been found. The test makes use of an OR tree to test that all the bits are not zero. If a zero is returned, a further test for the most significant bit is made to distinguish 00 found from the FF found. When a lookup value is found, the bottom register is loaded with the lookup value and the top register is loaded with a new value.

4.3 Normal basis affine

An affine transformation must be applied to the output of each inversion unit. One affine unit is used by each of the inversion units. The original specification requires the following affine transformation (over GF(2^8))

$$b_i = b_i \oplus b_{(i+4)\bmod 8} \oplus b_{(i+5)\bmod 8} \oplus b_{(i+6)\bmod 8} \oplus b_{(i+7)\bmod 8} \oplus c_i \quad (2)$$

for $0 \le i < 8$, where b_i is the ith bit of the byte, and c_i is the ith bit of a byte c with the value {63}. This is depicted in matrix form in [1].

The polynomial basis transformation has an equivalent representation in the normal basis. This can be found using a similarity transformation to the matrix based on the change of basis. The resulting normal basis equations generated for the affine transformation are given below for $\alpha = 33$

$$\begin{aligned}
q(0) &= i[6] \oplus i[2] \oplus i[1] \oplus i[0] \oplus 1\\
q(1) &= i[6] \oplus i[5] \oplus i[2]\\
q(2) &= i[5] \oplus i[4] \oplus i[3] \oplus i[1]\\
q(3) &= i[6] \oplus i[4] \oplus 1\\
q(4) &= i[4]\\
q(5) &= i[5] \oplus i[4]\\
q(6) &= i[7] \oplus i[6] \oplus i[5]\\
q(7) &= i[6] \oplus i[5] \oplus i[3] \oplus 1
\end{aligned}$$

This represents a reduction in logic over the polynomial basis. It is worth pointing out that 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. A reduction is possible by exploring the alternative values of α. Table 9 shows different affine solutions for different values of α. The XOR numbers can be reduced by factoring the matrix or finding common subexpressions (e.g. $i[6] \oplus i[5]$ in the example).

4.4 Normal basis mix

The MixColumns operation operates on the state column-by-column, treating each column as a four-term polynomial. The columns are considered as polynomials over GF(2^8) and are multiplied by $x^4 + 1$ with a fixed polynomial *a(x)*, given by

$$a(x) = 03 \bullet x^3 \oplus 01 \bullet x^2 \oplus 01 \bullet x \oplus 02$$

Table 9 Affine solutions for different values of α

α	Affine x-or gates
65	20
61	18
36	16
63	16
33	14
2*E*	13
26	13

Table 10 Multiplication solutions for different values of α

α	Multiplication x-or gates
27	21
29	19
71	18
36	18
24	17
3*A*	16
33	16

As a result of the multiplication, the four bytes in a column are replaced using the following equations

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

The MixColumns unit in the AES implementation has been designed to operate in parallel. This implies that the operations in the equations above are arranged to operate on separate units. A diagram showing the mix operation is shown in Fig. 5. The mix operation inputs four 8-bit values and the appropriate multiplication operations are executed according to the relative positioning of the inputs (Fig. 2).

The mix requires multiplication by 02 and multiplication by 03. The multiplication complexity can be reduced by searching among 128 possible normal basis solutions (these vary depending on the value of α). After searching, a reduction is possible down to only a few gates. It is also possible to reduce the multiplication complexity in conjunction with the affine solution. Table 10 shows different multiplication solutions for different values of α.

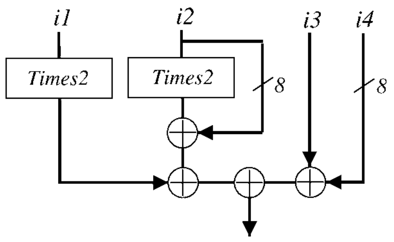

Figure 5 *Mix operation*

The normal basis equations for multiplication by 2 for $\alpha = 33$ are given below

$$q(0) = i[7] \oplus i[6] \oplus i[4] \oplus i[2] \oplus i[1]$$
$$q(1) = i[6] \oplus i[5] \oplus i[4] \oplus i[1]$$
$$q(2) = i[7] \oplus i[4] \oplus i[0]$$
$$q(3) = i[4] \oplus i[2] \oplus i[0]$$
$$q(4) = i[6] \oplus i[5] \oplus i[2]$$
$$q(5) = i[7] \oplus i[6] \oplus i[2] \oplus i[1]$$
$$q(6) = i[6] \oplus i[5] \oplus i[4] \oplus i[3] \oplus i[2] \oplus i[0]$$
$$q(7) = i[6] \oplus i[4] \oplus i[2] \oplus i[1]$$

Multiplication by 03 can be derived from XORing the multiplication by 02. Sharing of multipliers is possible between units. The output from the four mixes in Fig. 2 after concatenation represent a 32-bit subword of the text value prior to the XOR-key operation.

4.5 Register files

The RF layout is shown in Fig. 6. Four RFs are used, RF1. . .RF4, each containing four, 8-bit registers, which are used to contain the representation of the state.

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: $1/\{S_{00}, S_{10}, S_{20}, S_{30}\}$, $2/\{S_{01}, S_{11}, S_{21}, S_{31}\}$,

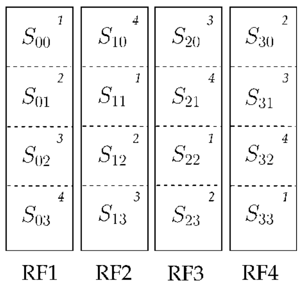

Figure 6 *RF layout*

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 RFs 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 indicates that the values to be mixed are sent in groups to the inversion unit directly from the RFs. This reduces the level of interconnect and area considerably.

4.6 Key generation

The same datapath as for the main algorithm may be used for key generation. This is made possible by extending the RFs to create storage for the key. S-box operations are required for key expansion. Since the data unit does not perform an S-box operation when the last Affine, 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.

4.7 Decryption

The datapath is easily modified to work for decryption. For ShiftRows the existing RFs are used to carry out pre-shifting at no extra cost in hardware but using the appropriate selection of address. For decryption, the same S-boxes are used as for encryption. An inverse-affine function must be added to each S-box, which needs to be executed before the multiplicative inverse. The combined S-box and its inverse is arranged to work in either direction using multiplexing. Finally, the mix-function needs to be augmented with the inverse-mix function and the appropriate multiplexing added to sequence the correct values.

The key generation is also easily modified for decryption. This is made to work on-the-fly, that is, the key unit stores the current RoundKey and is able to calculate the next or preceding key, respectively. Augmenting encryption, key generation with decryption can be executed efficiently using appropriate multiplexing.

5 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 automatic synthesis tool which makes use of a two-level minimisation algorithm. The implementation technology used was Cadence ASIC 0.35 μm. 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 the polynomial basis. Table 11 shows the results for the implementation.

In Table 11, 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 GEs. The implementation technology for each is shown in the process column, which ranges from 0.11 to 0.6 μm. The next column shows 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 and the decryption column is given in terms of the operation mode.

Table 11 AES Comparison

Implementation	Width, bits	Area, gates	Process μm	Frequency, MHz	Block Clock cycles	Throughput, Mbps	Decryption
NB	32	4,800	0.35	132	108	156	yes
[3]	32	8,200	0.6	50	64	128	yes
[3]	32	12,894	0.6	50	34	241	yes
[6]	32	5,400	0.11	131	54	311	yes
[7]	8	3,200	0.13	130	160	104	no
[7]	8	3,100	0.13	152	160	121	no
[17]	8	3,400	0.35	na	1032	na	no

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 [3] has ~70% more area and a throughput which is 18% slower but which uses a slower process technology of 0.6 μm. The second set of results for [3] has a much larger area of approximately three times the value but has a throughput which is ~50% as fast. Although the Satoh throughput is high [6], this is because of the faster technology used and the critical path is known to be longer, meaning an improvement would be apparent in our implementation with an upgrade in technology. Generally, the improvement in area is significant and exhibits a considerable reduction over alternative small area AES solutions of a similar 32-bit width.

The GE count for our normal basis implementation comes to 4800. The majority of the area is taken up by storage, which includes the data and key and which accounts for ~40% of the area. The next largest area is taken by the shifting and lookup logic which accounts for ~20% of the total area. The control represents about 8% of the total area.

The core logic functionality for the normal basis S-box shows a reduction of 25% over implementations using subfield S-boxes including [3, 6, 17], where the gate count is approximately 190 against 134 for our lookup logic. For S-boxes where shift registers and pipeline registers are taken into account, the normal basis technique appears to be roughly on a par basis. For S-boxes using more traditional lookup methods, such as [12], the normal basis implementation shows a significant improvement.

The bottle-neck in terms of time for our implementation comes from the inverse function. The large clock Tc that 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 the 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 of the time is made. The average time is derived from a simulation of a data set 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 on a par basis. However, this is generally at the expense of a slower throughput that is significantly slower, which makes our implementation competitive overall. The explanation for the slower throughput is straightforward as the bottle-neck for 8-bit wide implementations comes from the mix operation (this assumes the S-box can be pipelined) that can only execute 8 bits at a time.

Comparing power with other architectures is difficult because of the different technologies used and the lack of available data. As a basic estimate, the power consumption is estimated to be ~50% higher than that stated in [17]. Based on area and throughput comparisons, it is estimated that the power consumption will be significantly less than that of the other 32-bit architectures.

6 Conclusions

We have presented a novel efficient method of implementing the AES algorithm in the 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 that makes use of inversion in the field GF(2^8). As a result, the lookup size for inversion has been reduced when compared with alternative lookup implementations, and minimal lookup tables in terms of their GE are used.

A low-cost implementation of the AES has been presented, which targets a minimal number of gates. The GE size is less than other presented works 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. The number of lookup accesses are reduced, thereby improving latency. The regular RF design indicates that the interconnect and area are also reduced. The architecture would benefit from an improvement in technology.

The regular architecture is considered an important aspect of design for security, which facilitates security implementation. There are benefits from the regular architecture, lookup tables and registers that are regular and more easily power-balanced.

7 Acknowledgment

The authors would like to thank everyone who has contributed to this work including the useful suggestions made by the referees. This work was supported by EPSRC grants GR/S81421 (SCREEN) and EP/F016786/1 (SURE).

8 References

[1] Natl 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] LIN T., SU C., HUANG C., WU C.: 'A high-throughput low-cost AES cipher chip'. IEEE Proc. 3rd Asia-Pacific Conf. ASICS (AP-ASIC), August 2002

[3] MANGARD S., AIGNER M., DOMINIKUS S.: 'A highly regular and scalable AES hardware architecture', *IEEE Trans. Comput.*, 2003, **52**, (4), pp. 483–491

[4] ELBIRT A., YIP W., CHETWYND B., PAAR C.: 'An FPGA-based performance evaluation of the AES block cipher candidate algorithm finalists', *IEEE Trans. VLSI Syst.*, 2001, **9**, (4), pp. 545–557

[5] HUANG Y., LIN Y., HUNG K., LIN K.: 'Efficient implementation of AES IP'. Circuits and Systems 2006, APCCAS IEEE Conf., 2006, pp. 1418–1421

[6] SATOH A., MORIOKA S., TAKANO K., MUNETOH S.: 'A compact Rijndael hardware architecture with S-box optimization'. Proc. Advances in Cryptology – ASIACRYPT 2001, 2001, pp. 239–254

[7] HÄMÄLÄINEN P., ALHO T., HÄNNIKÄINEN M., HÄMÄLÄINEN T.: 'Design and implementation of low-area and low-power AES encryptionhardware core'. Proc. 9th EUROMICRO Conf. Digital System Design (DSD′06), 2006, pp. 577–583

[8] AL-SOMANI T., AMIN A.: 'Hardware implementations of GF($\hat{2}$m) arithmetic using normal basis', *J. Appl. Sci.*, 2006, **6**, (6), pp. 1362–1372

[9] YU N., HEYS H.: 'Investigation of compact hardware implementation of the advanced encryption standard'. Proc. IEEE Conf. CCECE, Saskatoon, Saskatchewan, May 2005, pp. 1069–1072

[10] VERBAUWHEDE I., SCHAUMONT P., KUO H.: 'Design and performance testing of a 2.29 GB/s Rijndael processor', *IEEE J. Solid-State Circuits*, 2003, **38**, (3), pp. 569–572

[11] JING M., CHEN Y., CHANG Y., HSU C.: 'The design of a fast inverse module in AES'. Proc. Info-tech and Info-net, Cong. ICII, 2001, pp. 298–303

[12] TILLICH S., FELDHOFER M., GROßSCHÄDL J.: 'Area, delay, and power characteristics of standard-cell implementations of the AES S-box'. Proc. Embedded Computer Systems: Architectures, Modelling, and Simulation, LNCS 4017, July 2006, pp. 457–466

[13] MCLOONE M., MCCANNY J.: 'Rijndael FPGA implementation utilizing look-up tables', *J. VLSI Signal Process. Syst*, 2003, **34**, (3), pp. 261–275

[14] CANRIGHT D.: 'A very compact S-box for AES'. Proc. 7th Int. Workshop on Cryptographic Hardware and Embedded Systems (CHES 2005), LCNS 3659, 2005, pp. 441–455

[15] TAKAGI N., YOSHIKI J., TAKAGI K.: 'A fast algorithm for multiplicative inversion in GF(2∧m) using normal basis', *IEEE Trans. Comp.*, 2001, **50**, (5), pp. 394–398

[16] JENG J.: 'Normal basis inversion in some finite fields'. 5th Int. Symp. Signal Processing and its Applications, ISSPA′99, Brisbane, Australia, August 1999, pp. 701–703

[17] FELDHOFER M., WOLKERSTORFER J., RIJMEN V.: 'AES implementation on a grain of sand', *IEE Proc. Inf. Secur*, 2005, **152**, (1), pp. 13–20

[18] SOKOLOV D., MURPHY J., BYSTROV A., YAKOVLEV A.: 'Design and analysis of dual-rail circuits for security applications', *IEEE Trans. Comp*, 2005, **54**, (4), pp. 449–460

doi: 10.1049/iet-cdt.2008.0049