A Configurable AES Processor for Enhanced Security

Chih-Pin Su, Chia-Lung Horng, Chih-Tsun Huang* and Cheng-Wen Wu
Department of Electrical Engineering
National Tsing Hua University
Hsinchu, Taiwan 30013

*Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan 30013

Abstract—We propose a configurable AES processor for extended-security communication. The proposed architecture can provide up to 2^{19} different AES block cipher schemes within a reasonable hardware cost. Data can be encrypted not only with secret keys and initial vectors, but also by different block ciphers during the communication. A novel on-the-fly key expansion design is also proposed for 128-, 192-, and 256-bit keys. Our unified hardware can run both the original AES algorithm and the extended AES algorithm. The proposed processor design has been fabricated by a 0.25μm CMOS process, with a silicon area of 6.93mm^2—about 200.5K equivalent gates. Under a 66MHz clock, the throughput rate for both the ECB and CBC operation modes are 844.8Mbps, 704Mbps, and 603.4Mbps for 128-bit, 192-bit, and 256-bit keys, respectively.

I. Introduction

The symmetric block cipher Rijndael, standardized by NIST as the Advanced Encryption Standard (AES) [1], has become a popular encryption standard for protecting sensitive data. Today, AES algorithm is used in a wide range of application in Internet and wireless communication [2–5]. In this paper, we propose the design of AESTHETIC (**A**dvanced **E**ncryption **S**tandard with **T**sing **H**ua **E**x**T**ended and **I**mplicit **C**onfigurability), an AES processor. It supports the original AES algorithm and also provides the flexibility of configuring the parameters of each transform defined in AES algorithm. The encryption and decryption procedures are architecturally the same as AES algorithm, however, the data is manipulated in a different way, since the parameters are changed. Following the communication protocol that specifies the secret key and the configuration parameters, the user can select randomly among these extended AES ciphers each time an encryption or decryption is requested. Such a dynamic configurability can further enhance the security of data communication.

We also proposed an novel key scheduler that generates the round key on-the-fly. On-the-fly key generators have been proposed in [3, 6, 7], which generate the round keys concurrently during the encryption or decryption procedure without extra memory to store the subkeys. In [6], the proposed architecture was designed to support 128-, 192-, and 256-bit keys. The key generators only for 128-bit keys were proposed in [3, 7]. Our on-the-fly key generator supports all the key sizes and generates a 128-bit round key per clock cycle, which is suitable both for the original and extended AES algorithms.

II. AES with Extended Configurability

The extended AES algorithm uses the same encryption/decryption procedure and key expansion as the original AES algorithm [1]. It consists of four transforms operating on bytes, rows and columns of the 4×4 byte array, called the *State*, that represents the data block. Four basic transformations of the AES algorithm are described briefly as follows:

SubBytes() transform (S-Box), is a non-linear byte substitution that operates independently on each byte of the State. Given an element of the State array, a_{ij}, $0 \leq i, j \leq 3$, that is treated as the element in $GF(2^8)$ with the irreducible polynomial $p(x)$. An inverse mapping of a_{ij} is performed first followed by an affine transform. The affine transform can be expressed as $b(y) = \text{const}(x) + y \cdot \text{Affine}(x) \mod (x^8 + 1)$, where y is the inverse of a_{ij} and Affine(x), const(x) are two polynomial with the degree less than 8.

ShiftRows() transform is simply a circular shifting operation on the rows of the State with different numbers of bytes (offsets).

MixColumns() transform is the operation that mixes the bytes in each column by the multiplication of the State with a fixed polynomial $c(x)$ modulo $(x^4 + 1)$ with its coefficient in $GF(2^8)$.

AddRoundKey() transform is simply an XOR operation that adds a round key to the State in each iteration, where the round keys are generated by the key expansion procedure.

The extended AES algorithm is that $p(x)$, Affine(x), const(x) and the coefficients of $c(x)$ are all configurable. We can obtain a new encryption/decryption algorithm by arbitrarily selecting the value of these parameters. However, not all the combinations can generate secure block ciphers against existing attacks. Several design criteria must be satisfied to ensure the selected $\{p(x), \text{Affine}(x), \text{const}(x)\}$ tuple can generate strong S-Boxes. The cryptanalysis of this part is contributed by the works of Prof. Chi-Sung Lai to guide the selection. We have selected 16 irreducible polynomial of degree 8 for $p(x)$. Each $p(x)$ has 256 pairs of $\{\text{Affine}(x), \text{const}(x)\}$, which are selected to provide sufficient algebraic complexity of the S-box. The matrix coefficients of MixColumns() transform are designed to provide strong diffusion power and guarantee that the relative inverse matrix exists. Thus 128 alternatives for $c(x)$ are selected with the restricted coefficients in the range of $\{\{01\}_x, \{02\}_x, \cdots \{0F\}_x\}$. The key expansion procedure is the same as the original AES algorithm with the

0-7803-8736-8/05/$20.00 ©2005 IEEE.

exception that S-Boxes are identical to those used in the encryption and decryption. Thus, including the original AES algorithm, there are totally $16 \cdot 256 \cdot 128 = 2^{19}$ types of block cipher which can be selected as the extended AES algorithm.

III. Design Considerations

Based on the discussion in the previous section, we have to design a hardware of SubBytes() and MixColumns() transform with extended configurability and consider both the hardware overhead and the performance impact. The S-Box implementation of the original AES algorithm has been proposed in various works. Most of them either use the table look-up approach [2, 6, 8–12] that stores the value of the S-Box in SRAM/ROM, or the arithmetic computing [3–5, 13, 14] that utilizes arithmetic components to perform a SubBytes() transform. The area and speed tradeoff of various S-Box designs have been evaluated in [15]. Using the ROM-based table look-up approach to support as much as 2^{12} kinds of S-Boxes will consume unacceptable area. Another way is to use a 256×8-bit SRAM to store the S-Box values. Any change on S-Box's parameters requires a dedicate hardware to recompute all the values, which will consume a long configuration time. Arithmetic computing approach, which implements the finite field inverter, the field multiplier and the matrix multiplier, has smaller hardware overhead and is easy to configure. We select this approach to implement the data path of the extended AES algorithm. The arithmetic computing approach will induce longer critical path than the table look-up approach. This drawback can be alleviated by the proposed architecture depicted in Fig. 1, which implements the field multiplier or inverter with a dedicate irreducible polynomial. The data path can be optimized for this polynomial instead of using a general inverter or multiplier. In addition, altering the irreducible polynomial $p(x)$ can be easily achieved by converting the data to or from the field constructed by the selected one.

A. Composite Field Arithmetic

All the computation logic related to the extended AES algorithm is implemented using the composite field arithmetic in $GF((2^4)^2)$ for better space-time trade-off. Basic computing elements include the field inverse, multiplier, and adder in $GF((2^4)^2)$. Let the elements in $GF(2^8)$ are represented as polynomials of degree one, i.e., $(a_1 x + a_0)$, where $a_0, a_1 \in GF(2^4)$. The inverse operation in the S-box can then be computed by taking the inverse of the polynomial $(a_1 x + a_0)$, with the irreducible polynomial, $P(x) = x^2 + Ax + B$, where $A, B \in GF(2^4)$. The multiplicative inverse of $(a_1 x + a_0)$ can be obtained by the following equation:

$$C(x) = (a_1 x + a_0)^{-1} = a_1 (a_1^2 B + a_1 a_0 A + a_0^2)^{-1} x + (a_0 + a_1 A)(a_1^2 B + a_1 a_0 A + a_0^2)^{-1}$$

To simplify the complexity, A is set to be the unit element in $GF(2^4)$. Multiplication of two field elements for MixColumns() transform results in

$$\begin{aligned} C(x) = A(x) \cdot B(x) &= (a_1 x + a_0)(b_1 x + b_0) \\ &= a_1 b_1 x^2 + ((a_1 + a_0)(b_1 + b_0) + a_0 b_0 + a_1 b_1) x + a_0 b_0. \end{aligned}$$

Let A be the unit element in $GF(2^4)$, we have

$$C(x) \mod P(x) = ((a_1 + a_0)(b_1 + b_0) + a_0 b_0) x + (a_0 b_0 + a_1 b_1 B).$$

Thus all the computing elements include field multipliers, squarers and adders in $GF(2^4)$. We select irreducible polynomial $p(x) = x^8 + x^4 + x^3 + x + ^2 + 1$ (11D as the hexadecimal representation) to construct $GF(2^8)$, and select the polynomial $P(x) = x^2 + x + w^{14}$ for $GF((2^4)^2)$ and the irreducible polynomial $Q(x) = x^4 + x + 1$ for $GF(2^4)$ based on the works in [16], which reported the optimal implementation of $GF(2^8)$ multiplier using the composite field arithmetic.

B. Field Conversion

When utilizing the data path with a fixed polynomial to perform an extended AES algorithm operated in different polynomials, it requires data conversion at the input and output of the data path. The multiplication and addition defined in the extended AES algorithm can be easily done by the composite-field data path with the help of a data conversion hardware. However, the affine transform is operated on the prime field $GF(2)$. It does not have an isomorphic operation defined in $GF((2^4)^2)$. Additional computation of the affine transform's parameters are required to work with the composite-field data path. We derive the isomorphic relation of the S-box with polynomial $p'(x)$ and $p(x)$ as follows.

Let T_α^γ be the basis transfer matrix from the $GF(2^8)$ with the polynomial $p(x)$ (11D) to the $GF((2^4)^2)$, and T_β^α be the basis transfer matrix from the field $GF(2^8)$ with $p'(x)$ to the field with $p(x)$. Thus $T_\beta^\gamma = T_\alpha^\gamma \cdot T_\beta^\alpha$ represents the basis transfer matrix from $GF(2^8)$ with $p'(x)$ to $GF((2^4)^2)$. Given the field element $x_\beta \in GF(2^8)$, we can find its composite field representation $x_\gamma = T_\beta^\gamma x_\beta$. Assume that T_A and c_A is the selected polynomial and constant of the affine transform. The relation of the S-Box with polynomial $p'(x)$ and that in the composite field can be expressed as

$$\begin{aligned} \text{SBox}_\beta(x_\beta) &= T_A \cdot \text{Inv}_\beta(x_\beta) + c_A \\ &= T_A \cdot ((T_\beta^\gamma)^{-1} \cdot \text{Inv}_\gamma(T_\beta^\gamma x_\beta)) + c_A. \end{aligned}$$

where Inv_β is the inverse operation with polynomial $p'(x)$. It can be substituted by the operation in the composite field with data conversion. From the above equation, we have

$$\begin{aligned} \text{SBox}_\gamma(x_\gamma) &= T_\beta^\gamma \cdot \text{SBox}_\beta(x_\beta) \\ &= T_\beta^\gamma T_A (T_\beta^\gamma)^{-1} \cdot \text{Inv}_\gamma(x_\gamma) + T_\beta^\gamma c_A. \end{aligned}$$

The inverse of the S-Box transform can be derived in the similar way, i.e.,

$$\begin{aligned} \text{SBox}_\gamma^{-1}(x_\gamma) &= \text{Inv}_\gamma(T_\beta^\gamma \cdot T_A^{-1}(x_\beta + c_A)) \\ &= \text{Inv}_\gamma(T_\beta^\gamma T_A^{-1} (T_\beta^\gamma)^{-1} x_\gamma + T_\beta^\gamma T_A^{-1} c_A). \end{aligned}$$

Thus $T_\beta^\gamma T_A (T_\beta^\gamma)^{-1}$, $T_\beta^\gamma c_A$, $T_\beta^\gamma T_A^{-1} (T_\beta^\gamma)^{-1}$ and $T_\beta^\gamma T_A^{-1} c_A$ must be computed beforehand. In our design where only 16 irreducible polynomials are used, T_β^γ and $(T_\beta^\gamma)^{-1}$ are precomputed and stored in a small ROM. The parameters of the affine transform can be obtained within a short configuration time by using an 8-by-8 bitwise matrix multiplier.

IV. Hardware Architecture

The top-level view of the AESTHETIC processor is shown in Fig. 1. It provides up to 2^{19} kinds of extended AES algorithm with the key size of 128, 192 and 256 bits. The original AES algorithm is also included as well. It supports both the Electronic Code Book (ECB) and Cipher Block Chaining (CBC) operation modes. The round keys for encryption and decryption are generated on the fly without any internal memory. Based on the discussion in previous sections, the time of the reconfiguration between two extended AES algorithm can be achieved rapidly within three clock cycles. All the data access operations are manipulated by the *I/O interface*. The 32-to-128-bit input FIFO buffer caches the 32-bit input data from the I/O interface to form a block of the 128-bit data, while the output FIFO buffer is used to cache the 128-bit output block from the *AESTHETIC engine*. The I/O interface also consists of the Initial Key (IK) register, the Initial Vector (IV) register and a control register for configuring the AESTHETIC processor. The *Main Controller* is designed to enable the configuration procedure, start or halt the encryption/decryption procedure of the *AESTHETIC engine*, and also manage the data flow between the input buffer, data path and the output buffer.

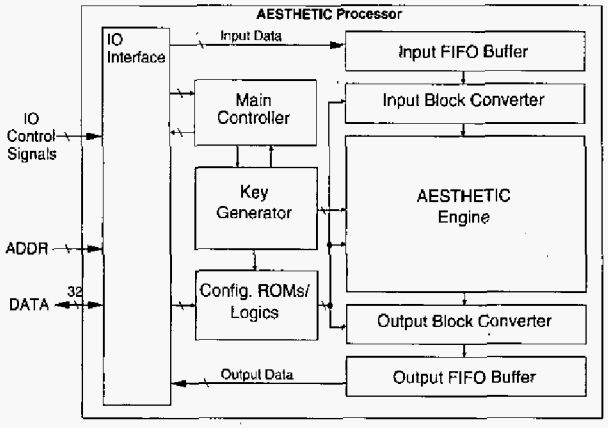

Fig. 1. Block diagram of the AESTHETIC processor.

A. Input/Output Block Converter

Whenever the encryption or decryption procedure is enabled by the Main Controller, the input data block will be retrieved from the input buffer and converted to the composite field representation in the *Input Block Converter*. This converter provides the following functions: 1) mapping the initial key to $GF((2^4)^2)$, 2) mapping the sum of the input data block and IV to $GF((2^4)^2)$ when encrypting the first block of the CBC data stream, and 3) mapping the input data block to $GF((2^4)^2)$ in the ECB mode.

Contrary to the Input Block Converter, the *Output Block Converter* converts the data in $GF((2^4)^2)$ back to $GF(2^8)$. When the processor is operated in the CBC decryption mode, the Output Block Converter will convert the sum of current output data block and previous output data block back to $GF(2^8)$, otherwise only the current output data block is converted. The converted data will either be XOR-ed with the initial vector when in the first block of the CBC mode decryption, or directly stored in the Output FIFO Buffer.

B. Configuration ROMs/Logic

The *Configuration ROMs/Logic module* contains various ROMs and computation logic to generate the necessary coefficients for the AESTHETIC engine. Figure 2 shows the block diagram of this module. There are 16 256 × 16-bit ROMs in the *Affine_ROM* module. Each ROM stores the Affine(x) and const(x) parameters for a specified polynomial. These parameters can be selected by control signals (Poly_select and Affine_select) specified in the control register. The *InvAffine Polynomial ROM* is used to store the inverse polynomial of each Affine(x). Either Affine(x) or InvAffine(x) will be selected as output according to the state of En_De signal

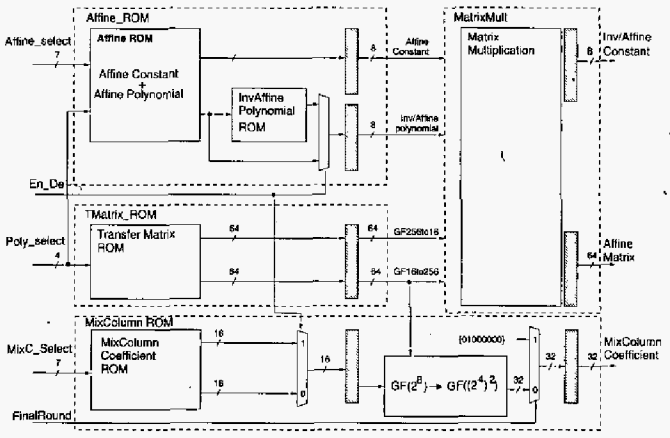

Fig. 2. Block diagram of configuration logic including the ROMs that store the coefficient of polynomials, affine transform and MixColumn coefficient, and a matrix multiplication circuit. The shaded boxes represent the pipeline registers.

TMatrix_ROM stores the basis transfer matrix T_α^β and its inverse matrix $(T_\alpha^\beta)^{-1}$ as discussed in Sec. III. The output signals GF256to16 and GF16to256 represent the value of T_β^α and T_α^β matrix respectively. These signals are used by *MatrixMult* module to compute either $T_\beta^\gamma T_A (T_\beta^\gamma)^{-1}$ and $T_\beta^\gamma c_A$ or $T_\beta^\gamma T_A^{-1} (T_\beta^\gamma)^{-1}$ and $T_\beta^\gamma T_A^{-1} c_A$.

The configurable coefficients of MixColumns() Transform are stored in *MixColumn_ROM*. Since the coeffi-

cients of the polynomial $c(x)$ are restricted to the range $\{\{01\}_x, \{02\}_x, \ldots, \{0F\}_x\}$, a 16-bit data is sufficient to store the coefficients of $c(x)$. All the coefficients of MixColumns() transform and inverse MixColumns() transform are selected by MixC_select and multiplexed by En_De. The selected coefficients are then converted to the composite field domain. Finally, a control signal, FinalRound, will multiplex between the converted coefficients and the value $\{01000000\}_x$. Thus in the MixColumns() Transform of the AESTHETIC engine, the data will either multiply with the MixColumns matrix or a unit matrix to bypass the MixColumns() transform.

C. AESTHETIC Engine

Figure 3 shows the block diagram of the AESTHETIC Engine. It consists of four basic transform of the extended AES algorithm. The architecture is designed to be used for both the encryption and decryption. The *Block S-Box* implements the S-Box transform for one 128-bit data block. The *SubBytes* and *InvSubBytes* transforms are implemented using one $GF((2^4)^2)$ inverse and two affine transform modules to eliminate the combinational loop. Both affine transform and inverse affine transform circuits are identical, however the function is different in the input values of affine matrix and constant. *ShiftRows* and *InvShiftRows* perform the same functions as those defined in the original AES algorithm. Since the shifting operations are performed in byte level. It makes no different in $GF((2^4)^2)$.

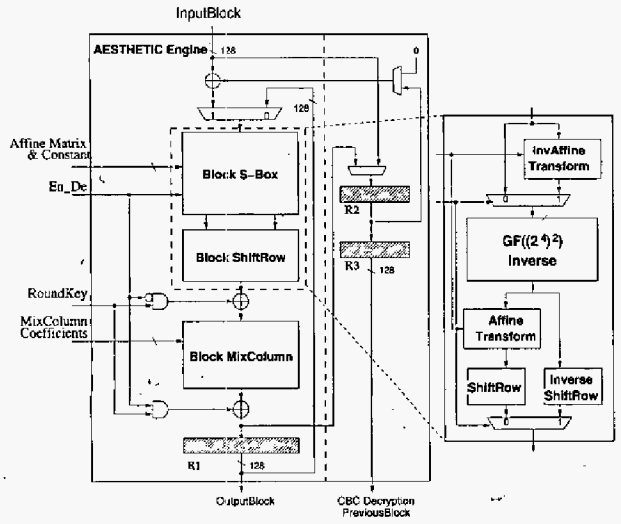

Fig. 3. Block diagram of the AESTHETIC engine.

Since all the coefficients of the MixColumn transform are programmable, it requires a 4-by-4 matrix multiplication. Thus in *Block MixColumn*, we implements 64 $GF((2^4)^2)$ multipliers to process the data block in parallel. The *MixColumn* and *InvMixColumn* transforms can easily share the same hardware by changing the coefficients according to the processing mode. In the final round of the AES algorithm where the MixColumns() transform is omitted, we configure the value of MixColumn coefficients to perform a unit matrix multiplication. Therefore, no extra hardware is required. The bitwise AddRoundKey() transform is implemented by two groups of XOR gates that are placed before and after the *Block MixColumn* module. According to the state of En_De signal, only one group of XOR gates affects the data.

In order to make both the ECB and CBC cipher modes consume the same execution time, our data path is designed to perform one round function per clock cycle. In Fig. 3, register R1 latches the output data of each round. The output will be fed-back to the input of the data path for the next round. Register R2 is used to support CBC cipher mode. In the CBC encryption mode, R2 holds the latest result of the encrypted data block. It will be used to XOR with the next input data block. In the CBC decryption mode, R2 holds the latest input data block. Additional register R3 is used to delay one clock cycle of R2's output, which will be XORed with the current decrypted block in the *Output Block Converter*.

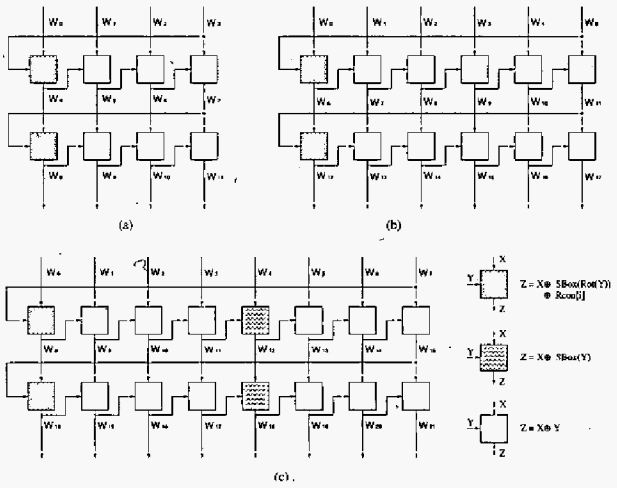

Fig. 4. Data flow graph of key expansion procedure for (a) 128-bit key, (b) 192-bit key and (c) 256-bit key.

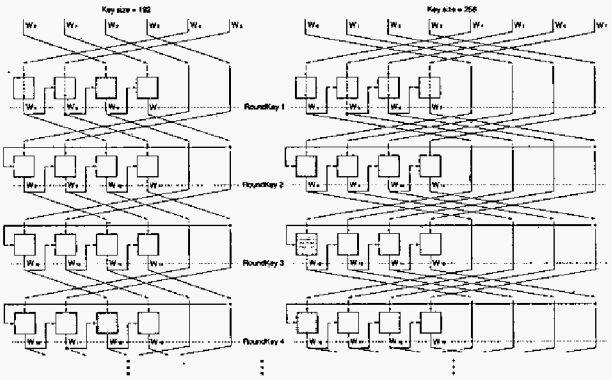

Fig. 5. Rearranged data flow graph of 192 and 256-bit key expansion.

D. Key Generator

We propose the 3-in-1 key generator to cooperate with the AESTHETIC engine. Since the key expansion procedure is the same as the original AES algorithm, our design can also be applied to standard AES cipher. All the round keys are generated on the fly without additional memory to store the subkeys. Our key generator produces one 128-bit round key per clock to cooperate with our data path, despite the change of the key size.

Fig. 4 shows the the data flow graph for three key size based on the key expansion procedure By observing the data flow graph, it can be easily found that generating a 128-bit round key per clock is quite straight-forward in the 128-bit key expansion procedure. For the 192 and 256-bit keys, although we can concurrently compute 192-bit or 256-bit keys per clock, the timing diagram is not compatible with the AESTHETIC engine. Thus we rearrange the data flow graph of the 192-bit and 256-bit keys such that only one 128-bit key is produced for each time frame. By properly shuffling the initial key and the round keys, we can use only four computing elements to generate the 128-bit round key at a time. The result is demonstrated in Fig. 5. The regularity in the data flow graph makes it easy to implement three different key expansion procedures by a unified hardware. For decryption, the key expansion procedure can be also found by easily reversing the computing order.

Figure 6 shows the data path of the *Key Generator* module based on the rearranged data flow graph. $R0, R1, \ldots, R7$ are 32-bit registers to store the shuffled round keys. Initially all the initial keys are converted to $GF((2^4)^2)$ and stored in $R0, R1, \ldots, R7$ registers depending on the key size. Once the *Key Generator* is enabled, the data path will generate the round key by properly controlling the multiplexers in each clock cycle. The data shuffling multiplexer is used to swapping the Ri's data and the generated round key iteratively. For the 192-bit key encryption, the output $RoundKey_i$ will be stored in $R2, R3, R4$ and $R5$ while the data in $R4$ and $R5$ will be stored in $R0$ and $R1$ to produce the next round key (see Fig. 5). Similarly when in the 256-bit key encryption, $RoundKey_i$ will be stored in $R4, R5, R6$ and $R7$, while the data in $R4, R5, R6$ and $R7$ will be swapped to $R0, R1, R2$ and $R3$. The S-Box transform in this data path must behave the same manner as those in the AESTHETIC engine. Thus, the parameters of the affine transform must be provided to configure the S-Box. All the Round Constant, $Rcon_i$, should also be converted to $GF((2^4)^2)$ before entering the key generator. In order to reduce the critical path and eliminate the combinational loop, the output of the S-Box is broadcasted to each W_i instead of connecting to the zero input of mux_00 and mux_12 in Fig. 6. Thus additional XOR gates and multiplexers such as mux_s0, mux_s1, mux_s2 and mux_s3 are used to provide equivalent mathematical relation.

V. Implementation Results

The processor is fabricated in a 0.25-μm CMOS technology with 5 metal layers. Figure 7 shows a micro-photograph of the AESTHETIC whose core contains 88K instances (about

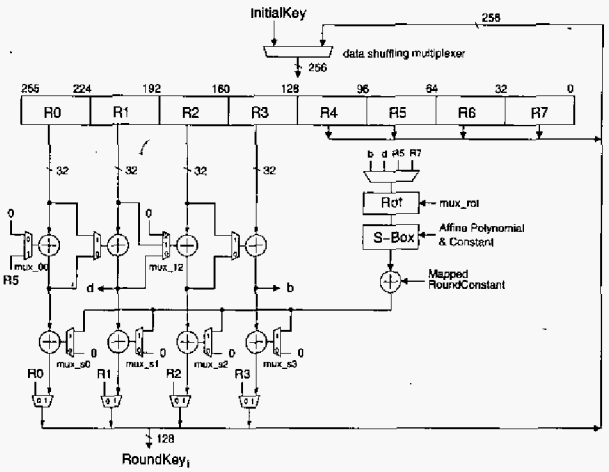

Fig. 6. Data path of *Key Generator* module.

200.5K gates) and measures $3.24 \times 1.94 mm^2$. The entire AESTHETIC processor is implemented by the standard cells. The gate counts of each module in the AESTHETIC processor are summarize in Table I. The whole chip is tested by the full-scan test. Using the commercial ATPG tool, we obtained 3906 scan test patterns with 99.33% fault coverage. Our chip can run with a 66MHz clock rate under the worse case condition. As a result, the throughput is 844.8Mb/s for 128-bit keys, 704Mb/s for 192-bit keys and 603.4Mb/s for 256-bit keys. The comparison of our implementation with existing AES designs is shown in Table II. Although the throughput to gate count ratio is about 40 times smaller to Satoh's design [4], our design can configure 2^{19} kinds of alternative AES.

On-the-fly key generator has been proposed in [3,6,7]. However, in [6], the proposed architecture was designed to support 128, 192 and 256-bit block of Rijndael algorithm with 3 different key size. Our design provided a more compact hardware dedicated for 128-bit block Rijndael algorithm just as that defined in the AES. The key generators in [3, 7] are designed for the 128-bit key. Our key generator can be used for encryption and decryption of 3 different key sizes.

TABLE I
AREA STATISTICS OF THE AESTHETIC PROCESSOR.

Module Name	Gate Count	%
I/O Interface	6121.3	3.05%
Input FIFO	1539.4	0.77%
Output FIFO	1774.3	0.88%
Main Controller	221.0	0.11%
Input Block Converter	9723.7	4.85%
Output Block Converter	4532.3	2.26%
AESTHETIC Engine	103523.3	51.63%
Key Generator	26639.3	13.3%
Configuration ROMs/Logics	49261.0	24.57%
Total	200501.0	100.00%

TABLE II
COMPARISON OF DIFFERENT AES DESIGNS AND OURS.

	[2]	[3]	[4]	[5]	Ours
Tech.	0.18μm	0.6μm	0.11μm	0.35μm	0.25μm
f	125MHz	64MHz	224.22MHz	200MHz	66MHz
α	1.6			2.381	0.844
	1.33	0.241	2.21	2.008	0.704
	1.14			1.736	0.603
β	173K	15.493K	21.337K	58.430K	200.5K
$\frac{\alpha}{\beta}$	9.25			41.49	4.21
	7.68	15.56	122.28	34.98	3.51
	6.59			30.24	3.01

f: clock rate, α: throughput (Gbps), β: gate count.

VI. CONCLUSIONS

A configurable AES processor has been described. The implicit configurability provides a rapid reconfiguration to switch among 2^{19} AES-extended block ciphers, making it suitable for high-security applications. Our design supports ECB, CBC operation modes with 128-, 192-, and 256-bit keys. Synthesized using a $0.25\mu m$ cell library, the gate count is estimated to be around 200.5K. The maximum throughput is about 844.8Mb/s for 128-bit keys, 704Mb/s for 192-bit keys, and 603.4Mb/s for 256-bit keys under a 66MHz clock. An on-the-fly key generator has also been proposed that produces exactly one 128-bit round key per clock cycle. It can be applied to not only the extended AES cipher but the original AES as well.

Technology	$0.25\mu m$ CMOS Technology with 5 Metal
Core Vdd	2.5V
Package	QFP128 pin
Clock Rate	66MHz
Power	259.1mW
Gate Count	200.5K Gates
Core Area	$3.24 \times 1.94 mm^2$
Baud Rate	844.8Mb/s (128-bit Key)
	704Mb/s (192-bit Key)
	603.4Mb/s (256-bit Key)

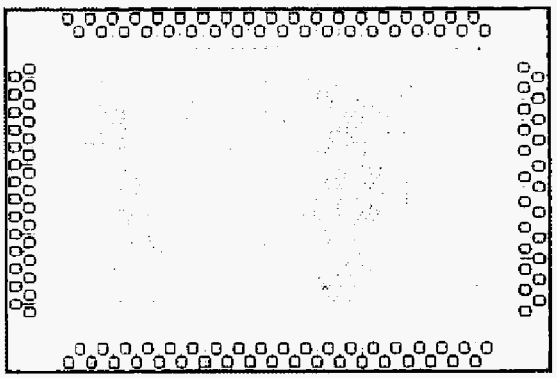

Fig. 7. The test chip characteristics and die photograph of the AESTHETIC processor.

REFERENCES

[1] National Institute of Standards and Technology (NIST), *Advanced Encryption Standard (AES)*, National Technical Information Service, Springfield, VA 22161, Nov. 2001.

[2] 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, pp. 569–572, Mar. 2003.

[3] S. Mangard, M. Aigner, and S. Dominikus, "A highly regular and scalable AES hardware architecture", *IEEE Trans. Computers*, vol. 52, no. 4, pp. 483–491, Apr. 2003.

[4] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, "A compact Rijndael hardware architecture with S-box optimization", in *ASIACRYPT 2001*. 2001, vol. 2248 of *LNCS*, pp. 239–254, Springer-Verlag.

[5] C.-P. Su, T.-F. Lin, C.-T. Huang, and C.-W. Wu, "A high-throughput low-cost AES processor", *IEEE Communications Magazine*, vol. 41, no. 12, pp. 86–91, Dec. 2003.

[6] H. Kuo and I. Verbauwhede, "Architectural optimization for a 1.82 Gbits/sec VLSI implementation of the AES Rijndael algorithm", in *Cryptographic Hardware and Embedded Systems (CHES) 2001*, Ç. K. Koç, D. Naccache, and C. Paar, Eds. May 2001, vol. 2162 of *LNCS*, Springer-Verlag.

[7] J. H. Shim, D. W. Kim, Y. K. Kang, T. W. Kwon, and J. R. Choi, "A rijndael cryptoprocessor using shared on-the-fly key scheduler", in *Proc. 3rd IEEE Asia-Pacific Conf. ASIC*, Taipei, Aug. 2002, pp. 89–92.

[8] V. Fischer and M. Drutarovsky, "Two methods of Rijndael implementation in reconfigurable hardware", in *Cryptographic Hardware and Embedded Systems (CHES) 2001*. May 2001, vol. 2162 of *LNCS*, pp. 77–92, Springer-Verlag.

[9] S. McMillan and C. Patterson, "JBits implementations of the advanced encryption standard (Rijndael)", in *Proc. 11th Int. Conf. Field-Programmable Logic and Applications (FPL)*. Aug. 2001, vol. 2147 of *LNCS*, pp. 162–171, Springer-Verlag.

[10] P. Chodowiec, K. Gaj, P. Bellows, and B. Schott, "Experimental testing of the Gigabit IPSec-compliant implementations of Rijndael and triple DES using SLAAC-1V FPGA accelerator board", in *Proc. Information Security Conf. (ISC)*. Oct. 2001, vol. 2200 of *LNCS*, pp. 220–234, Springer-Verlag.

[11] S. Morioka and A. Satoh, "A 10Gbps full-AES crypto design with a twisted-BDD S-Box architecture", in *Proc. IEEE Int. Conf. Computer Design (ICCD)*, Freiburg, Germany, Sept. 2002, pp. 98–103.

[12] K. U. Jarvinen, M. T. Tommiska, and J. O. Skytta, "A fully pipelined memoryless 17.8 Gbps AES-128 encryptor", in *Proc. Int. Symp. Field-Programmable Gate Arrays (FPGA)*, Monterey, 2003, pp. 207–215, ACM Press.

[13] J. Wolkerstorfer, E. Oswald, and M. Lamberger, "An ASIC implementation of the AES SBoxes", in *CT-RSA 2002*. 2002, vol. 2271 of *LNCS*, pp. 67–78, Springer-Verlag.

[14] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, "Unified hardware architecture for 128-bit block ciphers AES and Camellia", in *Cryptographic Hardware and Embedded Systems (CHES) 2003*. Aug. 2003, Springer-Verlag.

[15] U. Mayer, C. Oelsner, and T. Kohler, "Evaluation of different Rijndael implementations for high end servers", in *Proc. IEEE Int. Symp. Circuits and Systems (ISCAS)*, May 2002, vol. 2, pp. 348–351.

[16] C. Paar, "A new architecture for a parallel finite field multiplier with low complexity based on composite fields", *IEEE Trans. Computers*, vol. 45, no. 7, pp. 856–861, July 1996.