Grenze International Journal of Engineering and Technology, Jul Issue

High Throughput on-the-fly Implementation for AES

P.V.Sriniwas Shastry[1] and Mukul S. Sutaone[2]
[1]Cummins College of Engineering for Women, Pune, India
Email: shastry.pvs@gmail.com
[2]College of Engineering, Electronics and Telecommunication Department, Pune, India
Email: mssutaone@gmail.com

***Abstract*—This paper presents the outcome of our implementation of On-The-Fly based Rolled architecture for all key size AES on a TSMC 0.18µm technology. The architecture performs Encryption and Decryption both. We have employed Composite Field arithmetic for computations under SubstituteByte operations with design decisions for selection of appropriate polynomials to achieve least critical path delays. We have achieved a throughput of 2.33Gbps and one of the best throughput/area of 192 Mbps/K-gates. The whole design could fit into 0.19mm^2 area with power consumption of 19.263mW. The key expansion unit is designed for 128, 192 as well as 256-bit round keys. Critical path delay analysis for the whole design was performed to achieve minimum inter path delay difference so as to reduce unnecessary transitions (Dynamic Hazards) in the combinational circuit paths, which helped us in clocking the design at 182MHz.**

***Index Terms*— AES, On The Fly key expansion, All key size, OTF SubstituteByte and MixColumn, 180um ASIC implementation.**

I. INTRODUCTION

Cryptography plays an important role in the security of data transmission. To replace the old Data Encryption Standard (DES), the National Institute of Standards and Technology (NIST) invited proposals for the Advanced Encryption Standard (AES) in January 1997[1]. 15 proposals submitted to NIST around 1998. Out of fifteen algorithms, five were finalized (MARS, RC6 , Rijndael , Serpent and Twofish) Rijndael algorithm developed by Joan Daemen and Vincent Rijmen was chosen as the AES in 2001 after an open process since it was having the best performance on both hardware and software platforms. Also it has the shortest encryption/decryption time and thus it was voted extensively for the AES by the best cryptographers in the world. It is also known to be resistant to all known linear and differential cryptanalysis.

The hardware implementation of AES can be optimized for speed, size and power consumption. Two major targeted platforms for AES implementations are Field Programmable Gate Arrays (FPGA) and Application Specific Integrated Circuits (ASIC). There are many possible architectural options to the hardware design of on these platforms that includes internal pipelining, external pipelining, rolling and loop unrolled. The selection of these options depends on requirement of different speed/area trade-offs for different applications of AES algorithm. The AES algorithm involves few computations while performing each stage in encryption/decryption and key expansion, which is dealt in detail in next section. These computations can be performed on the fly (OTF) every time or, a Look-up-Table (LUT), consisting of pre-computed values for each of the sub-processes, can be employed. The OTF computational strategy is more or less a memoryless implementation but with higher

Grenze ID: 01.GIJET.7.2.39

complexity. Whereas the LUT based design requires lots of memory to store the pre-computed values.
In this paper we have presented AES architecture with OTF computational strategy for all operations under encryption/decryption both and round key generation for all key sizes. The substitute byte operation is also performed using combinational circuit and hence does not require the memory elements. The design uses limited resources with merely one 256 bit register, for all key sizes. The proposed design performs both encryption and decryption. The rest of the paper is organized in the following manner, Section II describes the computations involved in AES, Section III includes our proposed design and architecture, Section IV gives the results and compares with that of others and lastly in Section V, conclusion of this work.

II. AES Algorithm With Computations

The AES algorithm is an iterative cipher, which applies four different transformations on the data block of 128-bits, through predetermined N_r number of rounds. These four transformations are performed in a sequential manner in each round, and the final cipher is produced in the last round. Out of these four transformations, SubstituteByte and MixColumn involve complex computations, which may be critical for implementation and optimization for speed / area. The other two includes ShiftRow and AddRoundKey. The SubstituteByte transformation is also used in Key Expansion unit, which generates the round keys from the input cipher key.

A. SubstituteByte/InvSubstituteByte

This substitution transformation is non-linear and it occupies much of the total AES encryption area and consume around three fourth of its power [1]. This transformation is operating on each byte of the State elements. SubstituteByte composes of Multiplicative Inverse (MI) in finite field GF(2^8) and then Affine Transformation (AT) over GF(2) [2]. The MI can be found by extended Euclidean algorithm, such that the inverse of $b(x)$ denoted as $b^{-1}(x)$ can be found as shown in (3.10). The selection of polynomials $a(x)$ and $c(x)$ can be done based on Euclidean algorithm, such that $b(x).a(x) + m(x)\ c(x) = 1$. Hence, $a(x).b(x) \bmod m(x) = 1$, and consequently $b^{-1}(x) = a(x) \bmod m(x)$. The computations in GF(2^8) could be complex and if the polynomial $b(x)$ is converted into GF(2^4) rather than keeping in GF(2^8), the MI computation would be simpler. If the polynomial $b(x)$ is represented as $b(x) = b.x + c$, then the MI can be computed based on (1), where b and c are in GF(2^4). All multiplication if performed modulo irreducible polynomial as $x^2 + A.x + B$ [50], then

$$(bx + c)^{-1} = b(b^2B + bcA + c^2)^{-1}x + (c + bA)(b^2B + bcA + c^2)^{-1} \quad (1)$$

After finding the MI of each byte of State elements, an affine transformation is found, as shown in (2). For $0 \le i < 8$, where b_i is the i^{th} bit of the byte and c_i is the i^{th} bit of the c with the value $\{63\}_h$ or $\{01100011\}_2$.

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

The InvSubstituteByte is the inverse of byte substitution transformation. In this transformation inverse of the AT is obtained followed by taking the MI in GF(2^8). The MI computation is same that of SubstituteByte. The inverse AT is found, based on the expression shown in (3).

$$b'_i = b_{(i+2) mod\ 8} \oplus b_{(i+5) mod\ 8} \oplus b_{(i+7) mod\ 8} \oplus c'_i \quad (3)$$

B. MixColumn/InvMixColumn

The MixColumn operation involves byte multiplications and XOR of the products, on each column of the State. While implementing the AES on ASIC, normally memory based designs are discouraged because they require large amount of memory, which is considered to be costlier resource to implement. On-the-fly (OTF) computations of SubstituteByte & MixColumn transformations and Key Expansion reduce the memory usages and the hardware cost [3]. The OTF computations considers the columns of the State as polynomials over GF(2^8) and uses composite field arithmetic, reducing the space complexities. Although different irreducible polynomials can be used to construct GF(2^8), the irreducible polynomial used in all of the computations of AES is $p(x)$, mentioned in (4).

$$p(x) = x^8 + x^4 + x^3 + x + 1 \quad (4)$$

The advantage of OTF style of implementation is that the encryption and decryption can share some of the hardware functional components and achieve even low area implementation. Another advantage of using OTF computations of Key Expansion is that, all key sizes can expanded with a very small incremental amount of hardware resources.

C. ShiftRow/InvShiftRow

The ShiftRow transformation cyclically shifts each individual row of the State with a certain offset as shown in Fig. 4. The cyclic shifting is towards left in case of encryption and it is towards right in case of decryption data path. The offset of shift for each row of the State is denoted by (5), for $0 \le c < N_b$ and $0 \le r < 4$.

$$S_{r,c} = S_{r,(c+h[r,N_b]) \bmod N_b} \quad (5)$$

D. AddRoundKey

The AddRoundKey transformation involves bitwise modulo 2 additions (XOR) of Round Keys and the State. The Round Keys are generated in the Key Expansion unit. The AddRoundKey is same for encryption and decryption, since it only involves an application of XOR operation and it is its own inverse.

E. Key Expansion

The Key Expansion Unit of AES takes the Cipher Key, *K*, and generates a key schedule. A total of $Nb(N_r+1)$ words are generated in Key Expansion operation [2]. Each round key generated from these words are of 128-bit length. The resulting key schedule consists of a linear array of 4-byte words and are denoted by $[W_i]$, where *i* is in the range $0 \le i < Nb(N_r+1)$. The size of input Cipher Key *K*, can be 128, 192 or 256 bits. This also means that the Key Expansion unit generates 10 Round Keys in 10 iterations for 128-bit input *K*, 12 Round keys in 8 iterations for 192-bit input *K* and 14 Round Keys in 7 iterations if the input *K* is 256 bits. The SubWord( ) function which substitutes a four-byte input word and applies the SubstituteByte operation and produces a four-byte output word that has been operated on the RotWord( )function. RotWord( ) is a cyclic permutation function which shifts the input four-byte word circularly by one byte to left and returns a four-byte word. Rcon[i] is a round constant array containing values given by (6), where the value of *x* is $\{02\}_h$ in GF(2^8) and value of '*i*' defined within $1 \le i < N_b*(N_r+1)$.

$$Rcon[i] = [x^{i-1}, \{00\}, \{00\}, \{00\}] \quad (6)$$

Every following word, w[i] is equal to the XOR of the previous word, w[i-1] and the word N_k position earlier, w[i-N_k]. The words which are at positions that are multiples of N_k, are applied with transformations, prior to XOR, with SubWord(), RotWord ( ) and Rcon[i]. In case of 256-bit cipher keys where N_k=8, it is slightly different than for 128-bit and 192-bit cipher keys. The words, which are at the positions N_k=8 and at *i* such that (*i*-4) is a multiple of N_k, SubWord ( ) is applied to W[i-1] prior to XOR.

III. Design Consideration

While implementing the AES on ASIC, normally memory based designs are discouraged because they require large amount of memory, which is considered to be costlier resource to implement. On-the-fly (OTF) computations of SubstituteByte & MixColumn transformations and Key Expansion reduce the memory usages and the hardware cost. The OTF computations considers the columns of the State as polynomials over GF(2^8) and uses composite field arithmetic, reducing the space complexities. The advantage of OTF style of implementation is that the encryption and decryption can share some of the hardware functional components and achieve even low area implementation. Another advantage of using OTF computations of Key Expansion is that, all key sizes can expanded with a very small incremental amount of hardware resources. In the following subsections, implementation of SubstituteByte, MixColumn and all key size Key Expansion operations using composite fields are explained.

A. Computation of SubstituteByte/InvSubstituteByte

Of all the transformations in AES, the SubstituteByte transformation is the most computationally heavy [4]. This transformation involves computation of multiplicative inverse (MI) in GF(2^8), which is followed by an affine transformation (AT). In case of InvSubstituteByte, inverse affine transformation is performed first, followed by multiplicative inverse [2]. Both of these transformations are performed in GF(2^8), which is constructed by using $p(x)$ as the irreducible polynomial mentioned in (4). The MI computation is same for SubstituteByte and InvSubstituteByte, thus enabling the encryption and decryption data path to share the hardware resources. Composite field arithmetic can be employed to reduce the hardware complexity [4]. If $k = n.m$, then GF($(2^n)^m$) is isomorphic to the field GF(2^k). This property can be used to express polynomials over GF(2^8) in GF(2^4). The conversion of GF(2^8) in corresponding composite fields can be accomplished using conversions of (7), where φ can be $\{10\}_2$ or $\{11\}_2$ and λ may take values $\{1\ b_2\ b_1\ b_0\}_2$; b_i can be either '0' or '1'.

$$GF(2^2) \rightarrow GF(2) \qquad : \ x^2 + x + 1$$

$$GF((2^2)^2) \rightarrow GF(2^2) \quad : \; x^2 + x + \varphi$$
$$GF(((2^2)^2)^2) \rightarrow GF((2^2)^2) \quad : \; x^2 + x + \lambda \tag{7}$$

The MI computation can be done using (1), where A and B can be replaced using (7). Choosing the irreducible polynomial x^2+Ax+B and values of $A = 1$ and $B = \lambda$, the MI in GF(2^4) will take a form of equation in (8)

$$(bx + c)^{-1} = b(b^2\lambda + bc + c^2)^{-1}x + (c + b)(b^2\lambda + bc + c^2)^{-1} \tag{8}$$

The MI operation is performed after the Isomorphic mapping, represented as 'δ' in the Fig.1 on the input Byte. Post MI again the Byte are remapped using Inverse Isomorphic transformation, represented as 'δ^{-1}'. The hardware Implementation of these two transformations are done employing XOR gates. Care has been taken that the fan-in and fan-out is kept under three gates, so as to reduce the dynamic hazards in the combinational circuits. This has resulted in additional XOR gates, however, we achieved reduced dynamic power consumption and inter path delay differences [5].

TABLE I. CRITICAL PATH DELAYS AND STATISTICAL ANALYSIS OF DATA PATH DELAYS WHEN IMPLEMENTED ON ASIC

Values of φ and λ	Synthesized results			
	# XOR	Critical path delay(ns)	Mean Delay (ns)	Standard Deviation of path delay
{10},{1000}	90	3.291	3.035	0.102
{10},{1100}	89	3.297	3.133	0.136
{10},{1111}	90	3.289	3.045	0.126
{11},{1001}	90	3.300	2.997	0.168
{11},{1111}	92	3.294	2.967	0.152
After rearranging XOR gates {11},{1111}	95	3.063	2.916	0.081

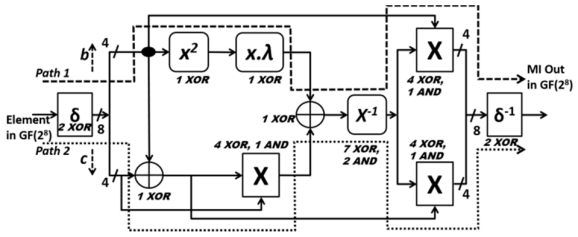

Figure 1. Combinational Circuit for Multiplicative Inverse and Critical Paths

The selection of the constants λ and φ, is one of the design consideration as the effect of these values on the area and power consumption of their hardware realizations depends of their values. Xinmiao *et. al* [6] concluded that $\varphi=\{10\}_2$ and $\lambda=\{1100\}_2$ are the optimum values of φ and λ giving minimum delays and consequently high speed. On the contrary we have used overall five best combination values of λ and φ. $\varphi = \{10\}_2$; $\lambda = \{1000\}_2,\{1100\}_2$ and $\{1111\}_2$ and $\varphi = \{11\}_2$; $\lambda = \{1001\}_2$ and $\{1111\}_2$ to evaluate the performance. The criteria for choosing these five combinations is to achieve reduction in number of terms in the expression, fan-in & fan-out requirement and inter path delay difference Δt_{pd} . This also helped us in reducing the unnecessary transitions and consequently reducing dynamic power consumption. Table I reveals that $\varphi = \{11\}_2$ and $\lambda = \{1111\}_2$, the critical path delay observed post place and route, is 3.063ns whereas the average delay is 2.967ns. The standard deviation of all net delays (σ) for the gate rearranged implementation is 0.081. One of the very important outcome of our experimentation is about the critical path from 0^{th} bit (LSB) of input net to 0^{th} bit (LSB) of output net and localizing the cause and techniques to minimize it. There are in all 64 possible data paths from 8-bit input to 8-bit output of the SubstituteByte operation including isomorphic mapping, MI and AT. This critical path delay analysis was conducted while implementing the design on Virtex-4, only to decide the values of φ and λ in conjunction with rearranging XOR gates.

The MI is the common operation in SubstituteByte and InvSubstituteByte operations. In SubstituteByte operation the Isomorphic transformation is followed by MI, AT and Inverse Isomorphic transform in order. Whereas in InvSubstituteByte operation, Isomorphic transform, AT^{-1} and MI, followed by Inverse Isomorphic transform. The hardware employed to realize Isomorphic transform, MI and Inverse Isomorphic transform can be shared between SubstituteByte and InvSubstituteByte operations.

B. Computation of MixColumn/InvMixColumn

The MixColumn involves byte multiplication by $\{02\}_h$ and $\{03\}_h$ over GF(2^8). The polynomial representation of $\{02\}_h$ is *x* and its multiplication with any GF(2^8) and the hardware realization of this multiplication can be done as shown in Fig. 2. The InvMixColumn involves byte multiplication by $\{09\}_h$,$\{0b\}_h$, $\{0d\}_h$ and $\{0e\}_h$. The polynomial representation of these multipliers in GF(2^8) are as shown below (9). The multiplication of an element with $\{08\}_h$ and $\{04\}_h$ can be realized merely by bit shifting and the multiplication with $\{02\}_h$ and $\{03\}_h$ over GF(2^8) can be as shown in Fig. 2.

$$\{09\}_h = (x^3 + 1) = (x^3) + 1 \quad = \{08\}_h + \{01\}_h$$

$$\{0b\}_h = (x^3 + x + 1) = (x^3) + (x + 1) \quad = \{08\}_h + \{03\}_h$$
$$\{0d\}_h = (x^3 + x^2 + 1) = (x^3) + (x^2) + 1 \quad = \{08\}_h + \{04\}_h + \{01\}_h$$
$$\{0e\}_h = (x^3 + x^2 + x) = (x^3) + (x^2) + x \quad = \{08\}_h + \{04\}_h + \{02\}_h \qquad (9)$$

C. Computation of Round Keys for all Key sizes

The key expansion unit has to generate round keys every clock cycle and provide them to AddRoundKey operation of the encryption or decryption data block. The synchronization of the key expansion unit and encryption/decryption data path is very critical as both may have different computational delays and latencies. The Key expansion algorithm can be drawn as a data flow graph (DFG), shown in Fig. 3. Each data path is of one word width (32-bits) and the process node 'O' is initialized with the input cipher key. The process node 'X' performs XOR, 'T' represents the 'temp' that holds the intermediate values of round keys being generated, 'S' performs SubWord ( ), RotWord ( ) and XOR with RCON, while rest of process nodes represent null processes and are added for the convenience of applying unfolding algorithm. The SubWord ( ) includes SubstituteByte operation. The process node 'S' is different for 256-bit key expansion. With an assumption that one round key needed every clock cycle, where each round key has to be 128-bits, the data flow graph for the key expansion can be unfolded [7] with a factor of *J*, that is same as the value of N_k .

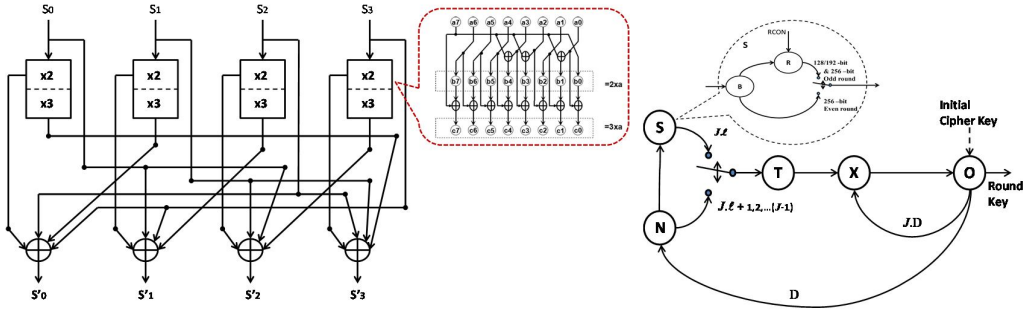

Figure 2. Hardware realization of MixColumn transformation

The J-unfolded DFG takes J-words in each clock cycle and generates next set of expanded keys. Out of the J-words generated, only 4 words (128-bits) are taken as Round keys and passed on to AddRoundKey operation of encryption/decryption data path and the remaining words in the subsequent cycle. The further expansion of process node 'S' is shown in Fig. 3. The process node 'B' performs the SubWord ( ) and RotWord ( ), while process node 'R' performs XOR with RCON (a round constant as mentioned in AES document [2]). The internal switching is to select either 128-bit and 192-bit or 256-bit key expansion. The register view of the DFG can be shown as in Fig. 4. In the DFG, due to incompatibility of the clock cycle for 128-bit, 192-bit and 256-bit key expansion, the registers are read such that only one 128-bit key material is produced in each time frame. The shaded words are the ones which include the additional operations of rotate word, substitute byte and XOR with RCON. The round keys generated by KE are *#RK_n*, where *n* is encryption or decryption round number. Each of the iteration of KE unit generates *#KEm_i*, where *m* is 128, 192 or 256 and *i* is the KE iteration number. The low level architecture for OTF key expansion for all key sizes is shown in Fig. 5.

IV. Implementation and Results

The proposed with OTF computations for AES with all three key sizes was first on Virtex-4 device and simulated using ModelSim. The design was verified by performing timing simulations for 128, 192 and 256 bit

key sizes. Analysis of the implemented design to find number of gates employed and the gates in the critical paths from input to final output for each bit was performed. The SubstituteByte and InvSubstituteByte shares the common hardware blocks and the data paths for encryption and decryption splits for ShiftRow and InvShiftRow operations. The AddRoundKey is performed in the decryption data path after InvShiftRow transformation. The E/D multiplexers select the path for either encryption or decryption. In case of encryption data path the AddRoundKey is performed after MixColumn transformation. The rnd# multiplexers are employed to bypass certain blocks. These multiplexers bypasses the MixColumn transformation in the last round of encryption, InvMixColumn transformation in the first round of decryption and first AddRoundKey operation on input plain/cipher text for rest of the rounds of the algorithm. The ShiftRow and InvShiftRow do not require any hardware resources, as the data lines of the bytes are twisted. The critical paths for encryption and decryption are showed in red and blue color traces on the architecture in Fig. 6. The clock period estimation for encryption and decryption is given in (10) and (11) respectively, where t_{SB}, t_{ISB} are the delays due to SubstituteByte and InvSubstituteByte.

$$T_{CLK_{Rolled}} = (4t_{mux} + t_{SB} + t_{ARK} + t_{MC}) + (t_{reg} + t_{cd} + t_{pp}) \quad (10)$$

$$T_{CLK_{Rolled}} = (4t_{mux} + t_{ISB} + 2t_{ARK} + t_{MC}) + (t_{reg} + t_{cd} + t_{pp}) \quad (11)$$

Even though our implementation requires higher than Quingfu[8], we have restricted the logic gates have fan-in of 3 consequently reduced critical path delay. The design was synthesized using RTL Compiler of Cadence using standard cell libraries of TSMC 180nm technology. A total of 12,134 standard cells utilized. Every clock cycle generated one RoundKey of 128-bit achieving a throughput of 2330Mbps. The physical layout of the design on 180nm was performed using SoC Encounter of Cadence. The total design could fit successfully into 190,676μm^2 area, with a core density of 70%. Table II, gives the brief comparison of our all key size Rolled OTF AES design.

V. CONCLUSIONS

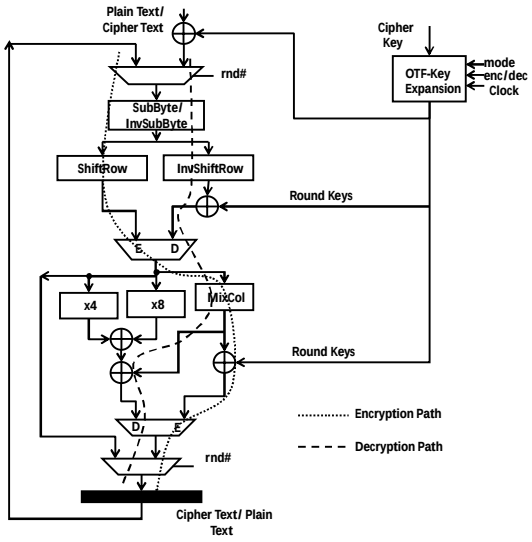

Figure 6. Encryption and Decryption Architecture for AES

The AES algorithm iterative, rolled style of architecture is most obvious and the simplest for low area implementation. Further insight of the sub-processes within each round of AES encryption and decryption, we could exploit the possibilities of applying composite field arithmetic for memory less design and OTF computation techniques. The implementation of multiplicative inverse of a byte in SubstituteByte and InvSubstituteByte operations, using composite field of GF$(2^4)^2$, the choice of φ and λ as $\{11\}_2$ and $\{1111\}_2$ respectively achieved minimum inter delay path difference as against the popular choice of $\{10\}_2$ and $\{1100\}_2$. Further keeping the Fan-in and Fan-out under three for all combinational gates and path delay equalization by mirroring some of these gates resulted in lower critical path delays. A Rolled OTF based AES encryption and

decryption combined architecture for all key sizes employing lowest number of standard cells to achieve 2.33Gbps throughput and higher throughput/area was successfully implemented on TSMC 180nm technology. A pipelined OTF architecture would result into even higher throughput is our suggestion.

TABLE II. COMPARISON OF ROLLED OTF AES WITH OTHER RELATED DESIGNS

	Our	[8]	[9]	[10]	[3]	[11]
Area(mm^2)	0.19	--	--	3.96	3.96	0.79
Throughput(Mbps)						
128-bit	2330	1160	380	1280	1600	3840
192-bit	1940	990	320	1070	1330	--
256-bit	1664	850	270	910	1140	--
Clock MHz	182	100	120	100	125	330
Standard Cells	12134	19500	21000	173000	173000	79000
Throughput/area Mbps/K-gates	192	59.5	18	7.4	9.25	48.6

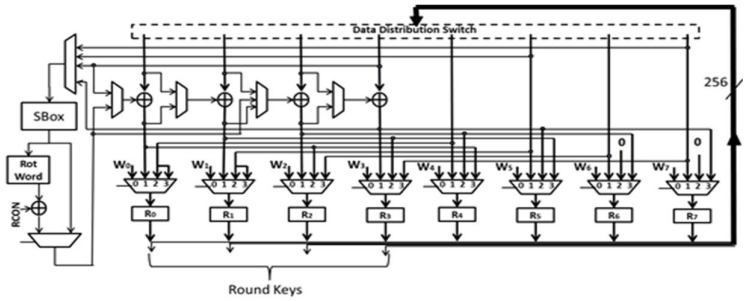

Figure 5. Low level architecture for OTF all key size Key Expansion unit

REFERENCES

[1] Akashi Satoh, Sumio Morioka, Kohji Takano, Seiji Munetoh, "A Compact Rijndael Hardware Architecture with S-Box Optimization", *ASIACRYPT 2001,* Springer-Verlag Berlin Heidelberg, LNCS2248, pp. 239-254.

[2] Federal Information Processing Standards Publication 197, "Announcing the Advanced Encryption Standard (AES)", November 26, 2001.

[3] Ingrid Verbauwhede, Partick Schaumont, Henry Kuo, "Design and performance testing of a 2.29GB/s Rijndael processor", IEEE Journal of Solid State Circuits, Vol. 38, No. 3,March 2003, pp. 569-572.

[4] Xinmiao Zhang, Keshab K. Parhi, "High-speed VLSI architectures for the AES algorithm", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 12, No. 9, September 2004, pp. 957-967.

[5] P.V.S.Shastry, M.S. Sutaone, "Multiplicative Inverse in GF(2^8) using combinational logic circuits", IETE National Journal for Innovation and Research (NJIR), Vol.1, Issue 1, June.

[6] Xinmiao Zhang, Keshab K. Parhi, "On the optimum constructions of composite field for the AES algorithm", IEEE Trans. On Circuits and Systems –II: Express Briefs, Vol 53, No. 10, pp. 1153-1157, 2006.

[7] Keshab K Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*, John Wiley & Sons Inc, 1999.

[8] Qingfu Cao, Shuguo Li, "A high throughput cost effective ASIC implementation of the AES algorithm", Proceedings of 8th IEEE International Conference on ASIC (ASICON'09), pp.805-808.

[9] Monjur Alam, S. Ray, D. Mukhopadhyay, S. Ghosh, D. Roychowdhury, I. Sengupta, "An area optimized reconfigurable encryptor for AES Rijndael", Proceedings of Design, Automation and Test in Europe (DATE 2007), April 2007, France, pp. 1116-1121.

[10] Henry Kuo, Ingrid Verbauwhede, "Architectural optimization for a 1.82Gbits/sec VLSI implementation of the AES Rijndael algorithm", Proceedings of 3rd International CHES 2001, pp. 51-64.

[11] Alizera Hodjat, David D Hwang, Bocheng Lai, Kris Tiri, Ingrid Verbauwhede, "A 3.84Gbits/s AES crypto coprocessor with modes of operation in a 0.18-um CMOS technology", Proceedings of the 15th ACM Great Lakes Symposium on VLSI, Chicago, Illinois, USA, 2005, pp. 60-63.