IEICE Electronics Express, Vol.xx, No.xx, xx-xx

LETTER

A Full Matrix Joint Optimization Method for Hardware Implementation of AES MixColumns/InvMixColumns

Xiaoqiang ZHANG[1, 2, 3a)], Fan YANG[1, 2)], Xinxing ZHENG[4], Xinggan ZHANG[3)], Ning WU[5]

Abstract Among Advanced Encryption Standard (AES) operations, MixColumns/InvMixColumns is the second most computationally complex operation after S-box. It occupies a large hardware resources and critical path delay (CPD) in AES hardware implementations. To reduce the hardware complexity of the MixColumns/InvMixColumns, a whole matrix joint optimization method is proposed in this paper. All coefficient multiplications in MixColumns/InvMixColumns are combined into a single matrix multiplication in the proposed method, and larger number of common subexpressions can be shared in the combined matrix. Therefore, the area can be drastically reduced in implementations. The validity of our whole matrix joint optimization is verified by theoretical analyses and synthesis tools. Both analyses results and synthesized results indicate that, compared with column joint optimization and row joint optimization, the optimization efficiency is improved greatly in the whole matrix joint optimization. Compared with previous works, our implementations have wider area-delay tradeoff, from less delay to minimal area cost.

key words: MixColumns, joint optimization, common subexpression eliminations, critical path delay
Classification: Integrated circuits

1. Introduction

Since the Advanced Encryption Standard (AES) is the latest block cipher standard published by the National Institute of Standards and Technology (NIST) in 2001, it is widely used in the systems of information security [1]. In the encryption process of AES, there are four operations in a round transform, *i.e.*, SubBytes, ShiftRows, MixColumns, and AddRoundKey. The decryption process of AES performs the reverse data flow of encryption process, and the round transforms in decryption process perform four inverse operation of encryption process, *i.e.*, InvSubBytes, InvShiftRows, InvMixColumns, and AddRoundKey.

Among these four operations, ShiftRows/ InvShiftRows is free in hardware implementations, and AddRoundKey requires only one layer XOR operations, there is no space to be further optimized for both of them in hardware implementations. Therefore, the optimization of AES mainly focus on the optimizations of SubBytes/InvSubBytes and MixColumns/ InvMixColumns. SubBytes/InvSubBytes is only one nonlinear operation among the four operations, so it causes wide concern in hardware implementations of AES [2, 3]. Although the concern of MixColumns/ InvMixColumns is smaller than SubBytes/InvSubBytes in hardware implementations, it still has optimization space to be further developed [4].

The MixColumns/InvMixColumns mainly consists of coefficient multiplications over $GF(2^8)$. In hardware implementations, the optimizations of MixColumns/ InvMixColumns are mainly focused on gate counts reduction. And resource sharing is a most commonly method to reduce gate counts in hardware implementations. There are two levels resource sharing in hardware implementations of MixColumns/ InvMixColumns, byte level sharing and bit level sharing.

In byte level sharing, the complex coefficient multiplications are decomposed into simple coefficient multiplications. The gate counts can be reduced by sharing common coefficient multiplications between outputs. The most common coefficient multiplication is {02}× operation, which is often called *xtime* function [1]. Different *xtime* block sharing strategies are proposed in [5, 6, 7, 8, 9, 10, 11, 12, 13, 14].

Besides these byte level sharing methods, a bit level sharing method is proposed in [4]. Compared with byte level sharing, more fine common operation units can be found in bit level sharing. In bit sharing level, the coefficient multiplications over $GF(2^8)$ are further expressed as bit level expressions [4]. Gate counts will be shared in hardware implementations through common subexpressions sharing in bit level expressions. The common subexpressions can be found out effectively by

[1]Key Laboratory of Advanced Perception and Intelligent Control of High-end Equipment, Ministry of Education, Wuhu 241000, China
[2]College of Electrical Engineering, Anhui Polytechnic University, Wuhu 241000, China
[3]School of Electronic Science and Engineering, Nanjing University, Nanjing 210023, China
[4]College of Information Engineering, Wuhu Institute of Technology, Wuhu 241006, China
[5]College of Electronic and Information Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China
a)zhangxiaoqiang@ahpu.edu.cn

DOI: 10.1587/elex.17.20200391
Received November 18, 2020
Accepted November 27, 2020
Publicized December 07, 2020

common subexpressions elimination (CSE) algorithms, which are widely used in a variety of complex computing circuits, such as digital signal processing circuit [15, 16, 17, 18, 19], cryptographic circuits [20, 21, 22], and codec circuits [23, 24, 25, 26].

The larger scale the bit level expressions are, the larger number of common subexpressions will be found among them, and the more gates will be reduced in hardware implementations [23, 25]. In [4], to improve the reduction rate, the coefficients on the same column are jointed to expand search scope of the common subexpressions. In this paper, a whole matrix joint optimization is proposed. Both column jointed optimization and row jointed optimization are used in the proposed whole matrix joint optimization, therefore, more gates counts are reduced in hardware implementations.

As the hardware complexities are usually evaluated by area and delay [27, 28, 29], all implementations are constructed by the shortest critical path structures in this paper. But area and delay are often mutually restricted in hardware implementation, it has been proven that sharing common subexpressions will increase critical path delay (CPD) of implementations [26]. To control the CPD in implementations, a delay-aware CSE (DACSE) algorithm proposed in [22] is employed to find out the common subexpressions under delay constraints. Two delay constraints, a tighter constraint to achieve the shortest feasible CPD and a looser constraint to achieve the smallest area, are provided for each implementation.

2. MixColumns/InvMixColumns

AES is a symmetric block cipher that process data blocks of 128 bits, and the data blocks can be regarded as 4×4 bytes state matrices. The MixColumns in AES transformation operates on the state matrix column-by-column. It treats each column as a four-term polynomial over $GF(2^8)$, and the polynomial is multiplied by another fixed polynomial modulo x^4+1 [1]. As a result, The MixColumns can be expressed as

$\mathbf{S}' = \mathbf{A} \cdot \mathbf{S}$:

$$\begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} \quad (1)$$

where $\mathbf{S}$, $\mathbf{S}'$, and $\mathbf{A}$ are input state matrix, output state matrix, and coefficient matrix of MixColumns, respectively. As a result of this multiplication, the four bytes in a column are replaced by the following.

$$\begin{cases} s'_{0,c} = \{02\}\, s_{0,c} + \{03\}\, s_{1,c} + \{01\}\, s_{2,c} + \{01\}\, s_{3,c} \\ s'_{1,c} = \{01\}\, s_{0,c} + \{02\}\, s_{1,c} + \{03\}\, s_{2,c} + \{01\}\, s_{3,c} \\ s'_{2,c} = \{01\}\, s_{0,c} + \{01\}\, s_{1,c} + \{02\}\, s_{2,c} + \{03\}\, s_{3,c} \\ s'_{3,c} = \{03\}\, s_{0,c} + \{01\}\, s_{1,c} + \{01\}\, s_{2,c} + \{02\}\, s_{3,c} \end{cases}, \; c = \{0,1,2,3\} \quad (2)$$

The corresponding hardware architecture of (2) is shown as Fig. 1

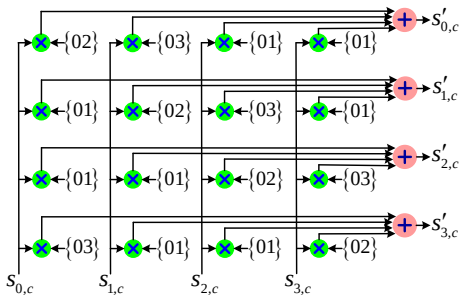

Fig. 1. Hardware architecture of MixColumns

The width of circuit shown in Fig. 1 is 32 bits, and only one column of state matrix is performed at once time. For the whole state matrix, it requires four iterations by using the circuit or four same circuits operating parallelly.

InvMixColumns is the inverse of the MixColumns transformation, it can be expressed as

$\tilde{\mathbf{S}}' = \tilde{\mathbf{A}} \cdot \tilde{\mathbf{S}}$:

$$\begin{bmatrix} \tilde{s}'_{0,0} & \tilde{s}'_{0,1} & \tilde{s}'_{0,2} & \tilde{s}'_{0,3} \\ \tilde{s}'_{1,0} & \tilde{s}'_{1,1} & \tilde{s}'_{1,2} & \tilde{s}'_{1,3} \\ \tilde{s}'_{2,0} & \tilde{s}'_{2,1} & \tilde{s}'_{2,2} & \tilde{s}'_{2,3} \\ \tilde{s}'_{3,0} & \tilde{s}'_{3,1} & \tilde{s}'_{3,2} & \tilde{s}'_{3,3} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} \tilde{s}_{0,0} & \tilde{s}_{0,1} & \tilde{s}_{0,2} & \tilde{s}_{0,3} \\ \tilde{s}_{1,0} & \tilde{s}_{1,1} & \tilde{s}_{1,2} & \tilde{s}_{1,3} \\ \tilde{s}_{2,0} & \tilde{s}_{2,1} & \tilde{s}_{2,2} & \tilde{s}_{2,3} \\ \tilde{s}_{3,0} & \tilde{s}_{3,1} & \tilde{s}_{3,2} & \tilde{s}_{3,3} \end{bmatrix} \quad (3)$$

where $\tilde{\mathbf{S}}'$, $\tilde{\mathbf{S}}$, and $\tilde{\mathbf{A}}$ are output state matrix, input state matrix, and coefficient matrix of InvMixColumns, respectively. In the same way, a 32bits wide circuit for InvMixColumns can be constructed.

As shown in Fig.1, MixColumns/InvMixColumns consists of coefficient multiplications and adders over $GF(2^8)$. Both of them are linear operations over Galois field, which contain XOR operations only. Therefore, the area of their hardware implementations can be measured by total used XOR gate counts, and the CPD can also be measured by XOR gate counts on critical path. In this paper, the hardware complexities are measured by A_{XOR} and T_{XOR}, where A_{XOR} and T_{XOR} denote area and delay of a XOR gate, respectively. And the hardware complexity comparison means area comparison at the same CPD in this paper.

3. The Proposed Joint Optimized Implementations

3.1 Shortest CPD structures

The coefficient multiplications over $GF(2^8)$ in MixColumns/InvMixColumns can be expressed as bit level expressions [4], and these expressions can be further expressed as a 8×8 bits constant matrix multiplication [22]. To achieve the shortest CPD, the coefficient multiplications are constructed by *Delay-Driven-Binary-Tree* (DDBT) structure in implementations, as the DDBT structure has the shortest CPD for the circuits consisting of single two-input gates

[30].

Suppose input delays can be ignored, the coefficient multiplications are also constructed by *Fastest-Binary-Tree* (FBT) structure [15, 31], which is a special case of the DDBT structure and it is only suitable for the circuits with same input delays [22]. The hardware complexities of these coefficient multiplications are listed on Table I. The direct implementations of coefficient multiplications can be further optimized by DACSE algorithm proposed in [22], the optimized results are also listed on Table I. The optimized implementations are also constructed by DDBT structures to achieve the shortest CPD.

Table I. Hardware Complexities of Coefficient Multiplications ($A_{XOR}@T_{XOR}$)

Methods	MixColumns			InvMixColumns			
	{01}×	{02}×	{03}×	{09}×	{0*b*}×	{0*d*}×	{0*e*}×
Direct	0@0	3@1	11@2	17@2	26@3	23@3	20@3
Optimized	0@0	3@1	9@2	12@2 11@3	17@3 16@4	15@3	16@3

As shown on Table I, coefficient multiplication {01}× requires no hardware resources. And there is no common subexpression can be shared in coefficient multiplication {02}×. Coefficient multiplications {03}×, {0*d*}×, and {0*e*}× can achieve minimal area under the shortest CPD constraints. Coefficient multiplications {09}× and {0*b*}× achieve minimal area at the cost of $1T_{XOR}$ CPD increase.

Besides the coefficient multiplications, the adders should also be constructed by DDBT structure to achieve the shortest CPD. But they cannot be constructed by FBT structure, as input delays of the adders are different. We take the adders for output $s'_{0,c}$ to illustrate the point in Fig. 2.

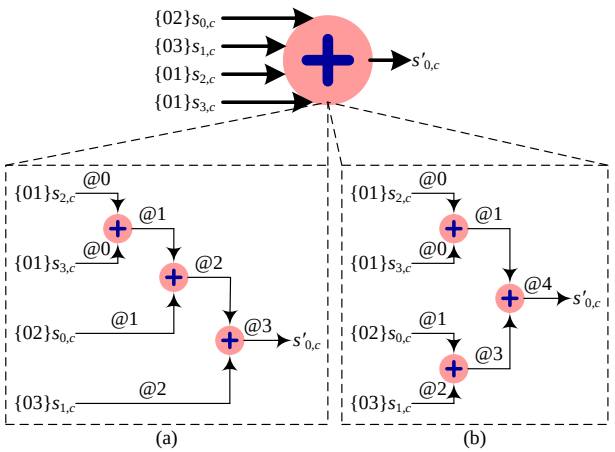

Fig. 2. Structures of the adders: (a) DDBT structure; (b) FBT structure

It requires $3T_{XOR}$ in the implementation if the adders are constructed by DDBT structure, but it requires $4T_{XOR}$ if the adders are constructed by FBT structure.

3.2 Matrix joint optimization

To share larger number of common subexpressions, a column joint optimization is proposed in [4]. In the column joint optimization, the common subexpressions are shared among not only expressions of each coefficient multiplications but also the coefficient multiplications on the same column. The column joint optimization is shown in Fig. 3(a). The coefficient multiplications on the same column are combined into a larger matrix multiplication. As a coefficient multiplication in MixColumns can be expressed as an 8×8 bits matrix multiplication, the scale of combined matrix is 32×8 bits. Compared with individual optimization of each coefficient multiplication, can be found out in the combined matrix multiplication, so the area reduction is improved in the implementations.

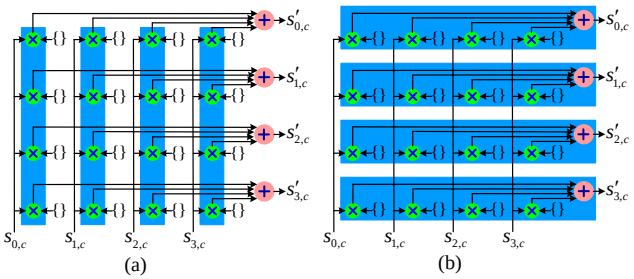

Fig. 3. Sketch of joint optimization: (a) column joint optimization; (b) row joint optimization

Similarly, the coefficient multiplications on the same row can also share common subexpressions jointly. The row joint optimization is shown in Fig. 3(b). Not only the coefficient multiplications but also the adders on the same row are combined into an 8×32 bits matrix multiplication. Therefore, the area reduction can be further improved in the row joint optimization.

Based on column joint optimization and row joint optimization, a whole matrix joint optimization can be deduced easily. In the whole matrix joint optimization, all coefficients in the coefficient matrix are combined into a 32×32 bits matrix. Hence the common subexpressions among all coefficient multiplications will be searched by the DACSE algorithm.

3.3 Hardware complexities analyses

The hardware complexities of MixColumns and InvMixColumns in different implementations are theoretically analyzed on Table II. As the circuit scale of MixColumns is smaller, all implementations can achieve the minimal area at the shortest CPD constraints after optimized by DACSE. As shown on Table II, in hardware implementations of MixColumns, the direct implementation requires $152A_{XOR}@3T_{XOR}$. Only 5.26% area is reduced in individual optimization. Compared with individual optimized implementations, the area reductions are improved greatly in column joint

optimized implementations. The reduction is further improved in row joint optimization, as the adders are also joined into the joint optimization. Both row joint optimization and column joint optimization are used in whole matrix joint optimized implementations, therefore, the area reduction of whole matrix joint optimization is improved more than twice, compared with row joint optimization.

Table II. Hardware Complexities of MixColumns & InvMixColumns

Blocks	Methods	Min CPD		Min Area	
		A_{XOR}{Red.}	T_{XOR}	A_{XOR}{Red.}	T_{XOR}{Inc.}
M.C.	Dir.	152	3	152	3
	Ind.	144{5.26%}	3	144{5.26%}	3
	Col.	136{10.53%}	3	136{10.53%}	3
	Row	132{13.16%}	3	132{13.16%}	3
	Mat.	108{28.95%}	3	108{28.95%}	3
I.M.C.	Dir.	440	5	440	5
	Ind.	332{24.55%}	5	328{25.45%}	6{20%}
	Col.	264{40.00%}	5	260{40. 91%}	6{20%}
	Row	264{40.00%}	5	248{43.64%}	8{60%}
	Mat.	193{56.14%}	5	169{61.59%}	8{60%}

In hardware implementations of InvMixColumns, the complexities of direct implementation are $440A_{XOR}@5T_{XOR}$. After optimized by DACSE, the CPD of all implementations are increased when these implementations achieve the minimal area. At min CPD constraints, the row joint optimization has the same area reduction as column joint optimization, but it has more area reduction at looser delay constraints. By using whole matrix joint optimization, the area reduction is up to 56.14% at min CPD constraint and up to 61.59% at min area constraint. As shown in Table II, in row joint optimization and whole matrix optimization, the implementations achieve the minimal area at cost of $3T_{XOR}$ increased on critical path.

From Table II, we can get that the area reductions of InvMixColumns implementations are larger than the ones of MixColumns implementations, as InvMixColumns operation is more complicated than MixColumns, and CSE algorithms have more efficiency in larger circuit.

Our works are compared with previous works on Table III. Byte level sharing is used in most previous works, and bit level sharing is used in [4] only. The column joint optimization is also used in [4], but there is no delay constraints in the common subexpressions sharing process. According to Table II and Table III, MixColumns implementation proposed in [4] has the same area reduction as our column joint optimized implementation of MixColumns, but the CPD is larger than ours, as the adders in [4] are not constructed by DDBT structure. The CPD of InvMixColumns implementation is increased $1T_{XOR}$ after sharing common subexpression in [4]. Compared with [4], our column joint optimized implementation of InvMixColumns has lower hardware complexity.

Table III. Hardware complexities Comparisons of Our Implementations with Other Works

Works	Opt. Level	MixColumns ($A_{XOR}@T_{XOR}$)	InvMixColumns ($A_{XOR}@T_{XOR}$)
[4]	Bit	136@4	264@6
[5]	Byte	140@4	364@5
[6]	Byte	142@4	356@6
[7]	Byte	140@4	304@7
[8]	Byte	108@3	193@7
[9]	Byte	132@4	292@7
[10]	Byte	108@3	212@5
[11]	Byte	132@4	192@8
[12]	Byte	140@4	292@7
[13]	Byte	140@4	304@7
[14]	Byte	116@4	198@7
Ours	Bit	108@3	193@5 169@8

The MixColumns implementations in [8] and [10] have the minimal area, and CPD is also kept without increasing. Our MixColumns implementation also achieves the same area at minimal CPD constraints as in [8] and [10]. For the InvMixColumns implementations proposed in previous works, the implementation proposed in [11] achieve the minimal area, but the CPD is also the largest in previous works. Our InvMixColumns implementation has lower hardware complexities, compared with previous works.

3.4 Synthesized results in IC design process

The implementations of MixColumns and InvMixColumns are described by Verilog HDL, and they are synthesized by Synopsys^{TM} DC Tool with SMIC $0.18\mu m$ technology.

In synthesis process of DC, the implementations at min CPD constraints are also constrained by tight delays to achieve min delay, and the implementations at min area constraints are also constrained by loose delays to achieve min area. The synthesized results of DC are list on Table IV.

Table IV. DC Synthesized Results for Different Implementations of MixColumns & InvMixColumns

Blocks	Met.	Min Delay		Min Area	
		Area(*gates*) {Red.}	Delay(*ns*) {Inc.}	Area(*gates*) {Red.}	Delay(*ns*) {Inc.}
M.C.	Dir.	638.67	0.62	384.00	1.05
	Ind.	638.67{0%}	0.62{0%}	384.00{0%}	1.05{0%}
	Col.	638.67{0%}	0.62{0%}	362.67{5.56%}	1.13{7.62%}
	Row	627.33{1.78%}	0.63{1.6%}	352.00{8.33%}	1.14{8.57%}
	Mat.	618.00{3.24%}	0.63{1.6%}	288.00{25%}	1.03{-1.9%}
I.M.C.	Dir.	2029.33	0.97	917.33	1.55
	Ind.	1966.33{3.10%}	0.98{1.03%}	877.33{4.36%}	1.77{14.19%}
	Col.	1792.33{11.68%}	1.03{6.19%}	693.33{24.42%}	2.16{35.39%}
	Row	1555.67{23.34%}	1.06{9.28%}	677.33{26.16%}	1.98{27.74%}
	Mat.	1271.67{37.34%}	1.07{10.31%}	455.33{50.36%}	2.99{92.90%}

Compared with other optimizations, the implementations based on whole matrix joint optimization have minimal area, after synthesized by DC Tool. Some optimization methods are also integrated in DC tool. Therefore, the optimization space is small for implementations of MixColumns at minimal delay constraints, the area reduction of whole matrix joint optimization is only up to 3.24%. For the

implementations of InvMixColumns at minimal delay constraints, the area reduction of whole matrix joint optimization is up to 37.34%, it is also smaller than the one in theoretical analyses.

For implementations constrained by minimal area, the area reduction of whole matrix joint optimization is up to 25% in MixColumns implementation, and up to 50.36% in InvMixColumns implementation. They are closer to the area reductions in theoretical analyses. So the theoretical analyses mentioned in this paper have some guiding significances for actual hardware complexities evaluations.

The MixColumns and InvMixColumns in other works are also implemented by Verilog HDL, and synthesized by DC Tool with the same setting conditions. These synthesized results are listed on Table V.

Table V. The Comparisons of DC Synthesized Results

Works	MixColumns				InvMixColumns			
	Min Delay		Min Area		Min Delay		Min Area	
	Area (*gates*)	Delay (*ns*)	Area (*gates*)	Delay (*ns*)	Area (*gates*)	Delay (*ns*)	Area (*gates*)	Delay (*ns*)
[4]	638.67	0.62	362.67	1.13	1891.33	0.98	704.00	2.17
[5]	652.00	0.65	288.00	1.26	1567.00	1.16	928.00	1.81
[6]	631.00	0.64	306.67	1.07	1100.00	1.19	650.67	2.22
[7]	652.00	0.65	288.00	1.26	1356.67	1.23	800.00	1.94
[8]	508.00	0.68	288.00	1.21	821.00	1.20	514.67	2.00
[9]	656.67	0.68	309.33	1.21	1149.33	1.27	654.67	2.16
[10]	508.00	0.68	288.00	1.21	1125.33	1.16	565.33	1.92
[11]	656.67	0.68	309.33	1.21	1008.00	1.40	469.33	2.45
[12]	652.00	0.65	288.00	1.26	1149.33	1.27	654.67	2.16
[13]	652.00	0.65	288.00	1.26	1356.67	1.23	800.00	1.94
[14]	656.67	0.68	309.33	1.21	843.33	1.23	490.67	2.28
Ours	618.00	0.63	288.00	1.03	1271.67	1.07	455.33	2.99

Compared with previous works, our implementations at minimal delay constraints achieve nearly the minimal delay, and our implementations at minimal area constraints have achieved the minimal area. The results indicate that our designs can provide a wider range of area-delay tradeoff. The MixColumns implementations at minimal area constraints are also achieved the minimal area in many previous works, but the delays of these implementation are larger than ours.

3.5 Synthesized results in FPGA design process

Our designs are also synthesized by Xilinx™ Vivado Tool with Virtex7, respectively. The synthesized results of Vivado are list on Table VI. No constraints are added in synthesized process, as the synthesized results are not affected by constraints in FPGA designs.

Table VI. Vivado Synthesized Results for Different Implementations of MixColumns & InvMixColumns

Methods	MixColumns		InvMixColumns	
	Area(*Slices*) {Red.}	Delay(*ns*) {Inc.}	Area(*Slices*) {Red.}	Delay(*ns*) {Inc.}
Dir.	44	1.24	100	1.75
Ind.	44{0%}	1.24{0%}	96 {4%}	1.77{1.14%}
Col.	44{0%}	1.24{0%}	89 {11%}	1.85{5.71%}
Row	43{2.27%}	1.29{4.03%}	88 {12%}	1.85{5.71%}
Mat.	43{2.27%}	1.29{4.03%}	68 {32%}	2.20{25.71%}

As some optimization methods are also integrated in Vivado, only one slice is reduced by whole matrix joint optimization in MixColumns implementation. In InvMixColumns implementation, the area reduction of whole matrix joint optimization is up to 32%.

The MixColumns and InvMixColumns in other works are also synthesized by Xilinx™ Vivado Tool with Virtex7. These synthesized results are listed on Table VII.

Table VII. The Comparisons of Synthesized Results of InvMixColumns Implementations

Methods	MixColumns		InvMixColumns	
	Area (*Slices*)	Delay (*ns*)	Area (*Slices*)	Delay (*ns*)
[4]	44	1.24	91	1.75
[5]	44	1.24	93	1.74
[6]	44	1.24	89	2.44
[7]	44	1.24	85	1.81
[8]	44	1.24	89	1.75
[9]	40	1.58	97	1.92
[10]	44	1.24	89	1.73
[11]	44	1.24	91	1.75
[12]	44	1.24	97	1.92
[13]	44	1.24	85	1.81
[14]	40	1.58	70	2.24
Ours	43	1.29	68	2.20

As the basic logic element in Virtex7 is 6-input look-up-table (LUT), the optimization effect is limited if the scale of common operation sharing is too small. In implementations of MixColumns, the optimization effect of our design is not obvious due to the circuit scale of MixColumns. In larger scale implementations of InvMixColumns, our design achieves the smallest area.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grants 61976113 and 61904001, in part by the Natural Science Foundation of Anhui Province under Grants 1908085MF179 and 1908085QF272, in part by the Natural Science Foundation of the Anhui Province Higher Education Institutions under Grants KJ2019A0983 and KJ2019A0163, in part by the Natural Science Research Program of Anhui Province Higher Education Promotion Plan under Grant TSKJ2017B23, in part by the Scientific Research Starting Foundation for the Introduction of Talents of Anhui Polytechnic University under Grants 2017YQQ001 and 2018YQQ007.

References

[1] National Institute of Standards and Technology (NIST), Advanced Encryption Standard (AES) FIPS Publication 197. http:/csrc.nist.gov/ publications/fips/fips197/fips-197.pdf, Nov 2001.

[2] M. M. Wong, et al.: "Composite field GF(((2^2)^2)^2) Advanced Encryption Standard (AES) S-box with algebraic normal form representation in the subfield inversion," IET Circuits Dev. Syst. 5 (2011) 471 (DOI: 10.1049/iet-cds.2010.0435).

[3] M. M. Wong, et al.: "Construction of optimum composite field

architecture for compact high-throughput AES S-boxes," IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 20 (2012) 1151 (DOI: 10.1109/TVLSI.2011.2141693).

[4] X. Zhang, and K. K. Parhi, "Implementation Approaches for the Advanced Encryption Standard Algorithm," IEEE Circuits and Systems Magazine, Vol.2, Issue.4, pp. 24-46, Fourth Quarter 2002. (DOI: 10.1109/MCAS.2002.1173133).

[5] Y.-K. Lai, L.-C. Chang, L.-F. Chen, C.-C. Chou, C.-W. Chiu, "A novel memoryless AES cipher architecture for networking applications" *in Proc. IEEE Int. Symposium on Circuits and Systems* (ISCAS'04), 2004, pp. IV-333-336. (DOI: 10.1109/ ISCAS.2004.1329008).

[6] H. Kuo and I. Verbauwhede, "Architectural Optimization for a 1.82Gbits/sec VLSI Implementation of the AES Rijndael Algorithm", Proceedings CHES 2001, pp.51–64, Paris, France, May 2001. (DOI: 10.1007/3-540-44709-1_6).

[7] C. C. Lu and S. Y. Tseng, "Integrated Design of AES (Advanced Encryption Standard) Encrypter and Decrypter," the IEEE International Conference on Application-Specific Systems, Architectures and Processors, 2002, pp. 277–285. (DOI: 10.1109/ASAP.2002.1030726).

[8] X. Zhang and K. K. Parhi, "High-Speed VLSI Architectures for the AES Algorithm," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 12, No. 9, pp. 957-966, September 2004. (DOI: 10.1109/TVLSI.2004.832943).

[9] V. Fischer, "Realization of the Round 2 Candidates Using Altera FPGA", The Third AES Conference (AES3), New York, Apr. 2000.

[10] C.-Y. Li, C.-F. Chien, J.-H. Hong, and T.-Y. Chang, "An Efficient Area-Delay Product Design for MixColumns/InvMixColumns in AES," IEEE Computer Society Annual Symposium on VLSI, 2008. pp. 1-4. (DOI: 10.1109/ISVLSI.2008.81).

[11] V. Fischer, M. Drutarovsky, P. Chodowiec, and F. Gramain, "InvMixColumn Decomposition and Multilevel Resource Sharing in AES Implementations," IEEE Transactions on Very Large Scale Integration (VLSI) System, Vol. 13, No. 8, Aug. 2005, pp. 989-992. (DOI: 10.1109/TVLSI.2005.853606)

[12] H. Li and Z. Friggstad, "An Efficient Architecture for the AES Mix columns Operation, Circuits and Systems," IEEE International Symposium on Circuits & Systems. IEEE, 2005, 23-26 May 2005, pp. 4637-4640. (DOI: 10.1109/ISCAS.2005. 1465666).

[13] Z. F. Zhao, D. Y. Yu, L. Li, "A Low-cost and High Efficiency Architecture of AES Crypto-engine," China Communications, Feb. 2008, No. 02, pp. 8-15. (DOI: 10.1109/CHINACOM.2007. 4469389).

[14] E. G. Ahmed, E. Shaaban, and M. Hashem. "Lightweight mix columns implementation for AES," MMACTEE'09: Proceedings of the 11th WSEAS international conference on Mathematical methods and computational techniques in electrical engineering, September 2009 pp. 48-53.

[15] A. Hosangadi, et al.: "Simultaneous optimization of delay and number of operations in multiplierless implementation of linear systems," 14th International Workshop on Logic and Synthesis-IWLS (2005) 1.

[16] R. Maheshand A. P. Vinod. "New Reconfigurable Architectures for Implementing FIR Filters with Low Complexity," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2010, 29(2): 275–288. (DOI: 10.1109/TCAD.2009. 2035548).

[17] M. Martínez-Peiró, E. I. Boemo, and L. Wanhammar. "Design of High-Speed Multiplierless Filters Using a Nonrecursive Signed Common Subexpression Algorithm," IEEE Transactions on Circuits and Systems II: Express Papers, 2002, 49(3): 196–203. (DOI: 10.1109/ TCSII.2002.1013866).

[18] F. Al-Hasani, M. P. Hayes, and A. Bainbridge-Smith. "A Common Subexpression Elimination Tree Algorithm," IEEE Trans. Circuits Syst. I: Reg. Papers, 2013, 60(9): 2389–2400. (DOI: 10.1109/TCSI. 2013.2244328).

[19] R. Maheshand A. P. Vinod. "A New Common Subexpression Elimination Algorithm for Realizing Low-Complexity Higher Order Digital Filters," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., 2008, 27(2): 217–229. (DOI: 10.1109/TCAD.2007. 907064).

[20] N. Chen and Z. Y. Yan: "High-performance designs of AES transformations," 2009 IEEE International Symposium on Circuits and Systems - ISCAS (2009) 2906 (DOI: 10.1109/ ISCAS.2009.5118410).

[21] M. M. Wong, and M. L. D. Wong. "A New Common Subexpression Elimination Algorithm with Application in Composite Field AES S-box," Tenth International Conference on Information Sciences, Signal Processing and their Applications (ISSPA 2010), 2010: 452–455. (DOI: 10.1109/ISSPA.2010. 5605445).

[22] X. Zhang, *et al.*: "An optimized delay-aware common subexpression elimination algorithm for hardware implementation of binary field linear transform," IEICE Electron. Express 11 (2014) 20140934 (DOI: 10.1587/elex.11.20140934).

[23] C. Paar, Optimized arithmetic for Reed-Solomonen coders, in Proc. IEEE Int. Sym. Information Theory, p. 250, 1997. (DOI: 10.1109/ISIT. 1997.613165).

[24] N. Chen, and Z. Y. Yan. "Cyclotomic FFTs With Reduced Additive Complexities Based on a Novel Common Subexpression Elimination Algorithm," IEEE Trans. Signal Process., 2009, 57(3): 1010–1020. (DOI: 10.1109/TSP.2008. 2009891).

[25] Y. Lee, H. Yoo, and I.-C. Park, "Low-Complexity Parallel Chien Search Structure Using Two-Dimensional Optimization," IEEE Transactions on Circuits and Systems-II: Express Papers, vol.58, no. 8, Aug. 2011, pp. 522-526. (DOI: 10.1109/TCSII.2011. 2158709).

[26] X. Zhang, et al.: "Low-delay parallel Chien search architecture for RS decoder," IEICE Electron. Express 13 (2016) 20160729 (DOI: 10.1587/ elex.13.20160729).

[27] J. L. Imana, et al.: "Bit-parallel finite field multipliers for irreducible trinomials," IEEE Trans. Comput. 55 (2006) 520 (DOI: 10.1109/TC.2006.69).

[28] X. Zhang, et al.: "Optimization of area and delay for implementation of the composite field advanced encryption standard S-box," J. Circuits Syst. Comput. 25 (2016) 1650037 (DOI: 10.1142/S0218126616500377).

[29] X. Zhang, et al.: "A low critical path delay structure for composite field AES S-box based on constant matrices multiplication merging," IEICE Electron. Express 7 (2020) 20200035 (DOI: 10.1587/elex.17.20200035).

[30] N. Petra, et al.: "A novel architecture for Galois Fields $GF(2^m)$ multipliers based on mastrovito scheme," IEEE Trans. Comput. 56 (2007) 1470 (DOI: 10.1109/TC.2007.70741).

[31] A. Chandrakasan, et al.: "Optimizing power using transformations," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. 14 (1995) 12 (DOI: 10.1109/43.363126).