32 位元低面積先進加密晶片實作

張肇軒
國立高雄第一科技大學
碩士
e-mail：
u9654805@ccms.nkfust.edu.tw

陳銘志
國立高雄第一科技大學
助理教授
e-mail：
mjchen@ccms.nkfust.edu.tw

摘要

在這篇論文中，我們提出一個有效率的共同子表示消去演算法(簡稱 CSE)來縮減 AES 轉換子函數中面積的耗費，而一個新的 CSE 演算法是應用在實現 AES 轉換子函數中可以化成位元階層的表示法上，我們利用 CSE 演算法實作出一顆 cell-based 的低面積 AES 晶片，和先前的設計比較在面積的改善上有明顯降低許多。

關鍵詞：AES,CSE,VLSI,Chip

1.前言

Rijndael 區塊加密演算法被選擇當作新一代的加密標準(AES)取代先前的資料加密標準(DES)，此演算法為對稱區塊加密而處理資料區塊為 128 位元由 4x4 位元組矩陣組成，叫做 state，一個 state 的運算會有 10、12、14 個回合依據不同的金鑰長度 128、192、256 位元。加密過程包含四個轉換 SubBytes(SB)，ShiftRows(SR)，MixColumns(MC)，和 AddRoundKey(ARK)，另外需要 KeyExpansion 用來產生每一回合所需要的回合金鑰給 ARK，圖.1(a)為 AES 演算法加密結構，首先初始金鑰先跟明文 XOR，接著一序列的轉換。當產生最後一回合的結果後即為密文。直接解密演算法如圖.1(b)產生顛倒的資料程序且其四個轉換為原本的反相，InvSubBytes(ISB)，InvShiftRows(ISR)，InvMixColumns(IMC)，和 AddRoundKey(ARK)。

藉由利用 CSE 演算法在不同的階層上(如結構階層、位元組階層或是位元階層)，實現 AES 的硬體複雜度可以大大地縮減。和先前的設計比較起來，此篇論文提出一個新的位元階層 CSE 演算法來實現更有效率的 AES 架構。

此篇論文第二部份介紹我們所採用的 32 位元 AES 的架構，第三部份介紹我們所提出的一個新的位元階層子結構分享方法，第四部份是實作晶片結果的比較，第五部份則是結論。

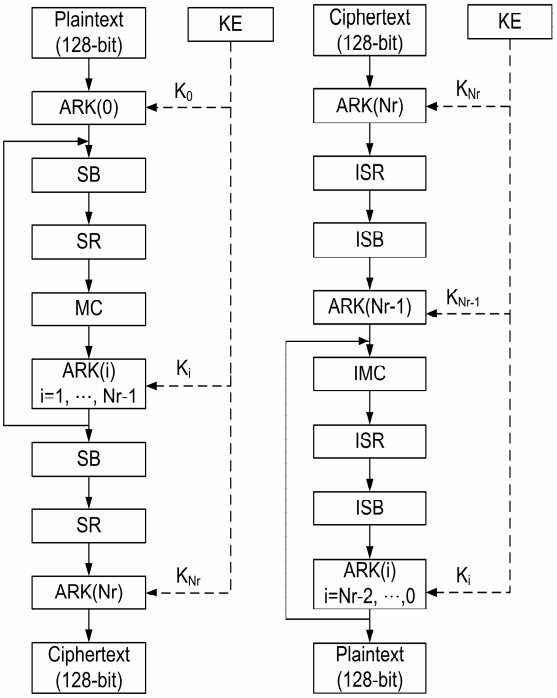

圖.1 AES (a) 加密和 (b) 直接解密

2. 32 位元 AES 各架構說明

2.1 AES 整體架構

在整個 AES 部份我們實現了兩種架構，分別是 AES(加密&直接解密)如圖.2 和 AES(加密&改良式解密)，經由 Synopsys 公司推出的 Design Compiler 可以看出實際合成的電路面積，而經過我們比較之後，AES(加密&直接解密)的面積會比 AES(加密&改良式解密)來的小，因此在最後選擇 AES(加密&直接解密)。

AES(加密&改良式解密)因為加密和解密的流程是一樣的，在 SubBytes 和 MixColumns 有部份電路可以共用藉此來降低面積花費，但是在 KeyExpansion 的部份必須再加入一份 InvMixColumns 的硬體面積，而 IMC 這部份面積是很大的，因此比較起來，AES(加密&直接解密)的面積會比較小，在此篇論文當中所討論和實現的結構都以 AES(加密&直接解密)為主。

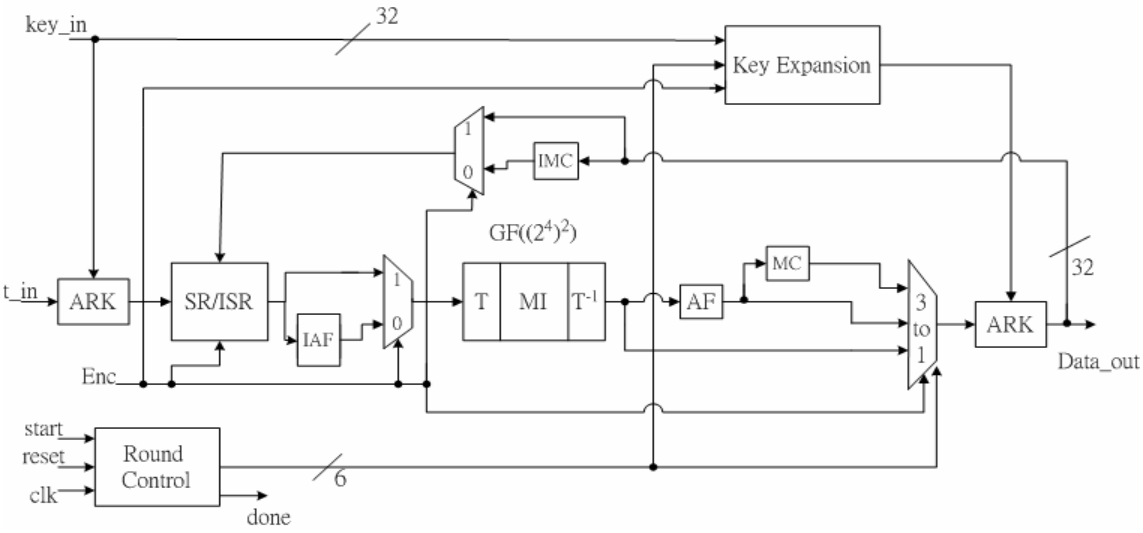

圖.2 AES 整體架構(加密&直接解密)

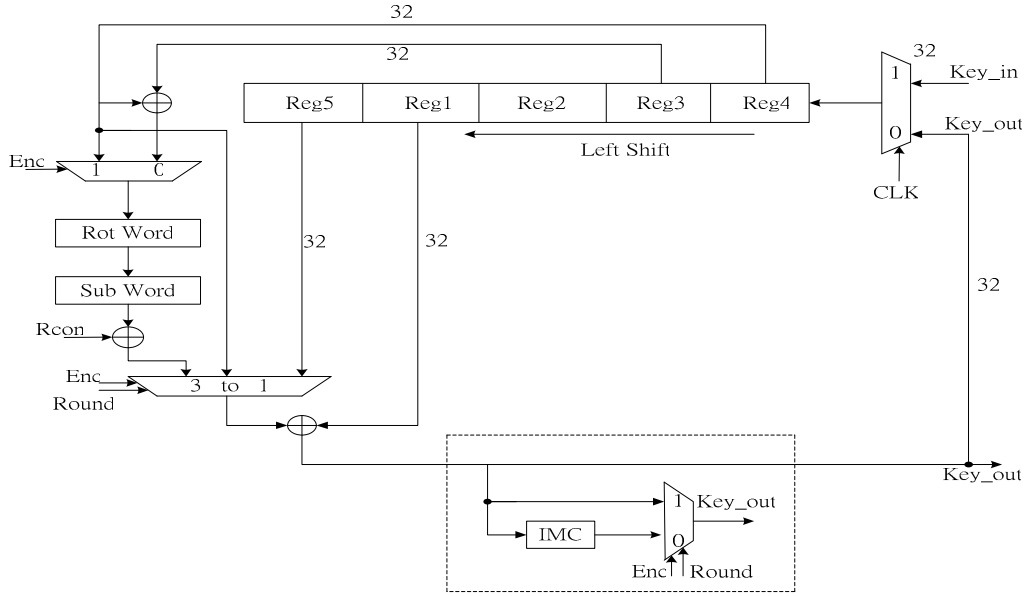

圖.3 KeyExpansion 架構

2.2 KeyExpansion

KeyExpansion 是藉由一把初始金鑰來產生 AES 演算法每一個回合所需使用到的回合金鑰，如圖.3 所示，此架構為 32 位元 Key Expansion on-the-fly 的電路，可以用來產生加密和解密的回合金鑰，目的是因為我們要作一個低面積的設計，因此不能使用 RAM 把每一回合的回合金鑰存起來以提供加解密使用，所以必須各別的回合產生此回合所需的加密或解密的回合金鑰，圖中框框的部份是假如使用的 AES 演算法為加密&改良式解密時必須加入的電路，假如使用的 AES 演算法為加密&直接解密，那麼就不必加入此段電路，右上角輸入的部份一開始連續 4 個 clock cycle 輸入四筆 32 位元的資料向左位移，之後產生的每回合的回合金鑰便會拉回輸入以產生下一回合的回合金鑰，總共產生 44 個回合金鑰(包含一開始的初始金鑰)。

2.3 ShiftRows/InvShiftRows 轉換

在 128 位元 AES 的架構上 ShiftRows 是不需耗費任何硬體的，只是單純的位移，但是在 32 位元架構上，ShiftRows 必須靠幾個移位暫存器來完成，R0,R1,R2,B0,B1,B2,B3 都是 32 位元的移位暫存器，一開始會連續輸入四筆 32 位元的資料到 B0~B3 暫存器中，接下來每個正緣 clock 便會向左位移一次，並由所選擇的 D0,D1,D2,D3(各別為 8 位元)當成此回合

的 32 位元輸出，再送至下一個 block 運作，每產生四筆輸出時 B0~B3 暫存器內的值便會被新的值取代。

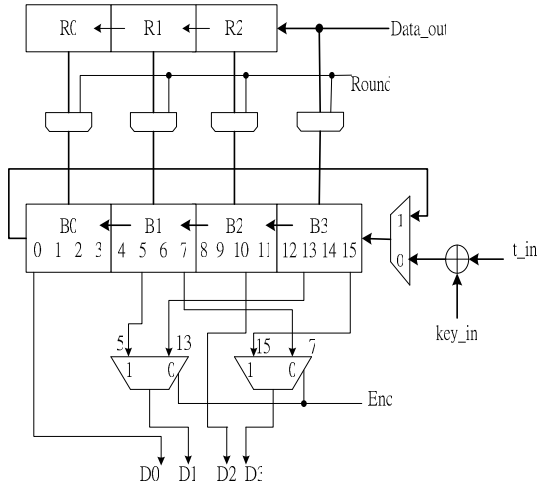

圖.4 ShiftRows/InvShiftRows 架構

2.4 AddRoundKey

單純的將 state 和回合金鑰作 XOR。

2.5 SubBytes/InvSubBytes 轉換

由於是作低面積設計，這部份不使用一般常見的查表方法，因查表法必須使用到 RAM 會花費大量的面積，這部份我們以組合邏輯來實現。

此轉換主要含兩部份，首先輸入值會先經由乘法反元素區塊，使輸入的值 S 變成 S^{-1}，接著緊隨著 Affine transformation，先乘一個 8x8 的 M 矩陣再 XOR 一個常數值 C 即可得到 SubBytes 轉換的輸出，乘法反元素的部份使用混合場算術，我們將 $GF(2^8)$降為 $GF(2^4)^2$ 藉此來縮減硬體複雜度[6]，一個元素$(S_hx + S_l)$的乘法反元素可以藉由擴展的歐幾里得演算法表示成 $(S_hx + S_l)^{-1}=S_h\Theta x + (S_h+S_l)\ \Theta$，$S_h$、$S_l\varepsilon GF(2^4)$，而 $\Theta= (S_h^2\lambda+S_hS_l+S_l^2)^{-1}$，藉此表示式將電路結構表示如下圖:

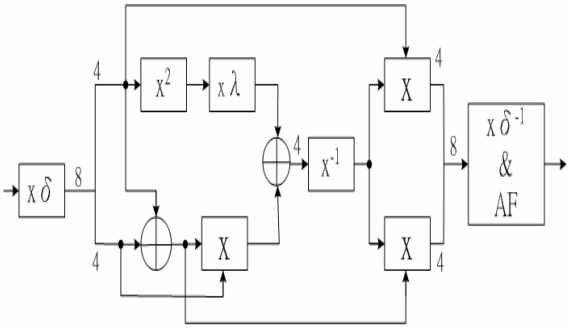

圖 5.乘法反元素電路

δ 和 δ^{-1} 分別都是 8x8 的矩陣，因為我們將 $GF(2^8)$降為 $GF(2^4)^2$ 必須先乘以一個轉換矩陣 δ 後，才能進去乘法反元素裡面運算，運算完之後再乘以轉換矩陣 δ^{-1} 將我們的值從 $GF(2^4)^2$ 轉回 $GF(2^8)$，可參考文獻[9]。

$$T=\begin{bmatrix}1&0&1&0&0&0&0&0\\0&0&0&0&1&1&0&0\\0&1&1&1&0&0&1&0\\1&0&1&0&1&1&1&0\\0&0&0&1&0&1&0&0\\1&0&0&1&0&1&1&0\\0&0&0&1&0&0&1&0\\1&1&1&0&0&0&0&1\end{bmatrix}\quad T^{-1}=\begin{bmatrix}1&1&0&1&1&1&0&0\\0&1&1&1&1&1&1&0\\0&1&0&1&1&1&0&0\\1&1&0&1&0&0&1&0\\1&0&0&1&1&0&1&0\\1&1&0&1&1&0&1&0\\1&1&0&1&0&0&0&0\\1&1&1&1&1&1&1&1\end{bmatrix}$$

圖.6 T & T^{-1} 矩陣

電路中的 δ、δ^{-1}、$GF(2^4)$乘法、x^{-1} 等區塊都可以化成位元階層的形式利用 CSE 演算法作化簡，藉此降低所需面積，ISB 的流程則是先經由 Affine transformation 再緊隨著乘法反元素。

2.5.1 混合場算術

混合場算術可以用來縮減硬體複雜度，更進一步可以使用管線的方法提昇效率，在此不採用管線，兩個多項式定義為 $Q(y)=y^n+\Sigma_{i=0}^{n-1}q_iy^i, q_i\varepsilon GF(2), P(x)=x^m+\Sigma_{i=0}^{m-1}p_ix^i$, $p_i\varepsilon GF(2^n)$，Q(y)用來建構 GF(2)成 $GF(2^n)$，P(x)用來建構 $GF(2^n)$成 $GF((2^n)^m)$，一個混合場就定義成 $GF((2^n)^m)$，而此混合場和原本的 $GF(2^k)$是同形的對於 k=nm，舉例來說，$GF(2^8)$的混合場可以藉由從 GF(2)反覆的建造使用如下的 irreducible 多項式:

$GF(2)=>GF(2^4)$: $P_0(x)=x^4+x+1$

{ $GF(2^4)=>GF((2^4)^2)$: $P_1(x)=x^2+x+\lambda$ }

其中 $\lambda=\{1001\}_2$。

2.5.2 轉換實現的比較

表.1 為實現 32 位元 SB/ISB 共用的 MI(包含 δ 和 δ^{-1})部份和參考文獻[6][7][10]分別在結構上和合成後作面積的比較，表中 Area 欄中(X; Y*)為面積的資訊，X 表示為結構上換算為相等的 gate 數(合成前)，Y*表示為實際合成(使用 Synopsys Design Compiler)後的 gate 數，使用的製程為 TSMC 0.18um 標準元件函式庫且使用相同的條件限制，我們合成的策略盡可能達到低面積的需求，利用 Synopsys Design Compiler 內所有面積相關的指令來產生面積最佳化，另外表中 Delay 欄中(D**)資訊代表實際合成後閘的延遲時間，舉例來說，如 Lu[10]

的設計中，在結構上估計花費 2956 閘來實現 SB/ISB 轉換中 MI 的部份，而實際合成後的面積花費則為 2796 閘而最長路徑延遲為 9.04 ns。

表中方法(1)是以 ROM 來實現，因此跟(2)和(3)的方法比起來會花費更大的面積，主要的面積花費在 4 個 256 位元組的 ROM 上來實現 4 個位元組的並行轉換。方法(2)為利用混合場算術將 $GF((2^4)^2)$ 有著較小的面積和延遲時間，應用我們所提出的 CSE 演算法來縮減 δ、δ^{-1}、MI 部份和[6]作比較，在結構上和合成後大概達到 7.3%的面積節省，方法(3)經由 CSE 化簡後為 1910 閘[7]比較約節省了 30.4%的面積，經由合成後仍然省了 8.1%的面積。

表.1 實現 32 位元 SB/ISB 方法比較

Methods \ Results		Area (gate)	Delay (ns)
(1) MI in $GF(2^8)$	Lu [10]	$192A_{XOR}+64A_{MUX}+24A_{INV}+4A_{ROM}$	$3T_{XOR}+2T_{MUX}+1T_{INV}+1T_{ROM}$
		(2956; 2796)*	*(9.04**)*
	Our CSE	$120A_{XOR}+64A_{MUX}+12A_{INV}+4A_{ROM}$	$4T_{XOR}+2T_{MUX}+1T_{INV}+1T_{ROM}$
		(2752; 2564)*	*(8.50**)*
(2) MI in $GF((2^4)^2)$	Zhang [6]	$616A_{XOR}+64A_{MUX}+144A_{AND}$	$21T_{XOR}+2T_{MUX}+4T_{AND}$
		(2026; 1923)*	*(7.65**)*
	Our CSE	$516A_{XOR}+64A_{MUX}+224A_{AND}+12A_{INV}$	$19T_{XOR}+2T_{MUX}+4T_{AND}+2T_{INV}$
		(1878; 1783)*	*(7.76**)*
(3) MI in $GF(((2^2)^2)^2)$	Satoh [7]	$764A_{XOR}+64A_{MUX}+360A_{AND}+36A_{INV}$	$23T_{XOR}+2T_{MUX}+4T_{AND}+1T_{INV}$
		(2744; 2001)*	*(8.67**)*
	Our CSE	$520A_{XOR}+64A_{MUX}+240A_{AND}+12A_{INV}$	$23T_{XOR}+2T_{MUX}+4T_{AND}+1T_{INV}$
		*(1910; **1839***)*	*(9.15**)*

2.6 MixColumns/InvMixColumns 轉換

一次一行四個 bytes 的轉換，藉由 4x4 的矩陣乘法，如下所示:

$$MC:\begin{bmatrix}A''\\B''\\D''\\E''\end{bmatrix}=\begin{bmatrix}02&03&01&01\\01&02&03&01\\01&01&02&03\\03&01&01&02\end{bmatrix}\bullet\begin{bmatrix}A'\\B'\\D'\\E'\end{bmatrix}$$

$$IMC:\begin{bmatrix}A''_{inv}\\B''_{inv}\\D''_{inv}\\E''_{inv}\end{bmatrix}=\begin{bmatrix}0e&0b&0d&09\\09&0e&0b&0d\\0d&09&0e&0b\\0b&0d&09&0e\end{bmatrix}\bullet\begin{bmatrix}A_{inv}\\B_{inv}\\D_{inv}\\E_{inv}\end{bmatrix}$$

A、B、D、E 各為 8 位元而轉換矩陣為十六進制表示法，舉例來說，MC 轉換後的 A"=({02}‧A')+({03}‧B')+({01}‧D')+({01}‧E')

{0X}‧A' 表示在 $GF(2^8)$ 中的乘法運算，irreducible 多項式為 $x^8+x^4+x^3+x^1+1$。

MC 和 IMC 在 $GF(2^8)$ 中的運算，可以使用子結構分享的方式有效率的實現[7][10-12]，使用函數區塊 XTime 來產生不變的乘法，如 {02}‧A'可以表示為 XTime(A')。MC 和 IMC 的轉換可以藉由共同的子結構分享來達到面積的縮減，IMC 可以被分解為加法和乘法表示，如 IMC=MC+G、IMC=MC x Z。

表.2 比較一些分開實現 MC 和 IMC 轉換的方法產生 4 個位元組輸出，每個方法有兩列分別為上列是結構上的資訊，下列為實際合成後的資訊，MC 和 IMC 的面積花費使用我們所提出的 CSE 演算法可以得到一個最佳化的結果在結構上分別為 $108A_{XOR}$ 和 $167A_{XOR}$，和 Direct realization 比較起來在結構上省了 28.9%(MC)和 62%(IMC)的面積花費，而經過合成後則是分別省了 11.4%和 44.8%。

我們可以觀察出有先使用 CSE 演算法再經過 Synopsys DC 所產生的結果比沒有使用 CSE 演算法的結果好很多，因此我們所提出的 CSE 演算法確實可以有效率的縮減電路面積。

表.2 實現 32 位元 MC 和 IMC 的比較

Area/Delay / Methods	MC		IMC	
	Area (A_{XOR})	Delay (T_{XOR})	Area (A_{XOR})	Delay (T_{XOR})
	Area (gates)	Delay (ns)	Area (gates)	Delay (ns)
(1) XTime-based	140	4	356	6
Kuo [11]	*310*	*0.62*	*758*	*1.70*
(2)Byte-level sharing	132	4	464	7
Sklavos [12]	*310*	*1.06*	*558*	*2.84*
(3)Direct realization	152	3	440	6
	325	*0.73*	*815*	*1.37*
(4)Bit-level sharing	136	4	264	6
Satoh [7]	*336*	*0.67*	*623*	*1.80*
(5)Our CSE method	108	3	167	7
	288	*0.83*	*450*	*2.14*

3. CSE 演算法

AES 演算法主要的面積花費在 SB、ISB、MC 和 IMC 上，且都可以表示成由 XOR/AND 組成的函式之位元階層表示法，我們所提出新的 CSE 演算法主要就是可以縮減能表示成這種函式的電路面積。首先第一步為先提取 AND 的項目，盡可能的達到最大次數的提取後，第二步則為提取 XOR 項目，根據:

Rule1:找出最大共同項的 pair。

Rule2:找出最小相關性最小的 pair。

Rule3:找出出現在函式中最少次的 pair。

Rule4:比較在下一回合中哪一個還能提出最多共同項的。

舉例來說，我們要化簡此 IMC 內部一項乘法展開後由 XOR 所組成的函式:

$$D''_{inv} = \{0d\}A_{inv} + \{09\}B_{inv} + \{0e\}D_{inv} + \{0b\}E_{inv}$$

$$= \begin{bmatrix} a_4+a_5+a_7+b_4+b_7+d_4+d_5+d_6+e_4+e_6+e_7 \\ a_3+a_4+a_6+a_7+b_3+b_6+b_7+d_3+d_4+d_5+d_7+e_3+e_5+e_6+e_7 \\ a_2+a_3+a_5+a_6+b_2+b_5+b_6+b_7+d_2+d_3+d_4+d_6+e_2+e_4+e_5+e_6+e_7 \\ a_1+a_2+a_4+a_5+a_7+b_1+b_4+b_5+b_6+d_1+d_2+d_3+d_5+e_1+e_3+e_4+e_5+e_6+e_7 \\ a_0+a_1+a_3+a_5+a_6+a_7+b_0+b_3+b_5+b_7+d_0+d_1+d_2+d_5+d_6+e_0+e_2+e_3+e_5 \\ a_0+a_2+a_6+b_2+b_6+b_7+d_0+d_1+d_6+e_1+e_2+e_6+e_7 \\ a_1+a_5+a_7+b_1+b_5+b_6+d_0+d_5+e_0+e_1+e_5+e_6+e_7 \\ a_0+a_5+a_6+b_0+b_5+d_5+d_6+d_7+e_0+e_5+e_7 \end{bmatrix}$$

圖.7 舉例說明:所要化簡的函式

我們可以提取出以下的共同項並且以變數取代所提取出的 pair

$w_{32}=e_6+e_7$	$w_{33}=b_6+w_{32}$	$w_{34}=a_7+d_5$	$w_{35}=a_5+b_5$
$w_{36}=d_5+w_{35}$	$w_{37}=a_6+d_6$	$w_{38}=a_0+w_{37}$	$w_{39}=a_3+b_7$
$w_{40}=a_2+w_{33}$	$w_{41}=a_4+w_{34}$	$w_{42}=e_0+w_{36}$	$w_{43}=b_0+w_{38}$
$w_{44}=w_{42}+w_{43}$	$w_{45}=b_1+d_1$	$w_{46}=d_0+w_{34}$	$w_{47}=a_1+w_{45}$
$w_{48}=b_2+e_2$	$w_{49}=b_3+w_{39}$	$w_{50}=e_3+w_{49}$	$w_{51}=b_4+w_{41}$
$w_{52}=e_4+w_{51}$	$w_{53}=d_3+d_4$	$w_{54}=d_2+w_{36}$	$w_{55}=d_1+w_{40}$

圖.8 所有被提取出的最大共同項

$s^{(0)}_{max}=6$	$s^{(1)}_{max}=5$	$s^{(2)}_{max}=5$	$s^{(3)}_{max}=5$
$s^{(4)}_{max}=5$	$s^{(5)}_{max}=4$	$s^{(6)}_{max}=3$	$s^{(7)}_{max}=3$
$s^{(8)}_{max}=3$	$s^{(9)}_{max}=3$	$s^{(10)}_{max}=3$	$s^{(11)}_{max}=2$
$s^{(12)}_{max}=2$	$s^{(13)}_{max}=2$	$s^{(14)}_{max}=2$	$s^{(15)}_{max}=2$
$s^{(16)}_{max}=2$	$s^{(17)}_{max}=2$	$s^{(18)}_{max}=2$	$s^{(19)}_{max}=2$
$s^{(20)}_{max}=2$	$s^{(21)}_{max}=2$	$s^{(22)}_{max}=2$	$s^{(23)}_{max}=2$

圖.9 對應圖.8 最大共同項所發生的次數

$$D''_{inv} = \begin{bmatrix} a_5+b_7+d_4+d_6+w_{32}+w_{52} \\ a_6+d_7+e_5+w_{33}+w_{41}+w_{50}+w_{53} \\ a_4+w_{37}+w_{39}+w_{40}+w_{48}+w_{53}+w_{55} \\ d_3+e_4+w_{47}+w_{52}+w_{54}+w_{55} \\ a_1+d_1+d_2+e_2+w_{44}+w_{46}+w_{50} \\ b_7+d_0+e_1+w_{38}+w_{48}+w_{55} \\ w_{33}+w_{42}+w_{46}+w_{48} \\ d_5+d_7+e_7+w_{44} \end{bmatrix}$$

圖.10 提取完且以變數取代後的結果

原本的函式需要 $110A_{XOR}$ 經由使用我們所提出的 CSE 演算法後，變成只需要 $63A_{XOR}$ 可以看出面積有明顯的改善。

4.晶片結果比較

表.3 晶片結果比較

	Tech. (um)	Freq. (MHz)	Thr. (Mb/s)	Gate Equivalent (k-gates)
Hua [5]	0.35	85	247	9.843
Mangard[13]	0.6	64	128	10.799
Ours	0.18	77	116	8.206

我們所實現的 AES 設計，核心部份達到 8206 閘的結果且最高操作頻率為 77MHz。由於晶片下線有腳位的限制，因此最後的晶片包含一層 8 位元的介面將輸入輸出包覆成 8 位元傳輸，因為多了此介面，在最後晶片的面積變為 13962 閘，操作頻率則是降為 40MHz。在此我們只和[5][13]比較，皆為 32 位元資料路徑，可以看出我們所使用的 CSE 演算法確實可以達到一個低面積的結果，在金鑰產生的單元上我們的設計和[5]一樣都是使用 32 位元資料路徑，[13]則是採用 128 位元資料路徑。

5.結論

在這篇論文當中，我們提出一個新的CSE演算法最佳化有效率的縮減 AES 中的硬體面積，主要針對 SB/ISB、MC/IMC 轉換可以表示成位元階層表示法由 XOR/AND 組成的函式來使用，即使 Synopsys DC 對於所有的組合邏輯皆可以產生邏輯最小化，我們觀察 SB/ISB、MC/IMC 有先利用 CSE 演算法化簡額外分別可以多省 9.3%和 35%的面積花費，總共的面積縮減效率可以達到 12.2%使用 TSMC 0.18um cell library，由晶片比較結果顯

示我們的結果確實比別人先前的設計在面積上要好很多。

表. 4 晶片資訊

使用製程	TSMC 0.18um cell library
封裝	40 S/B
晶片大小	1.447 x 1.445 mm^2
閘數	13692
錯誤涵蓋率	99.86 % with 70 test patterns
功率消耗	16.24mW@40 MHz

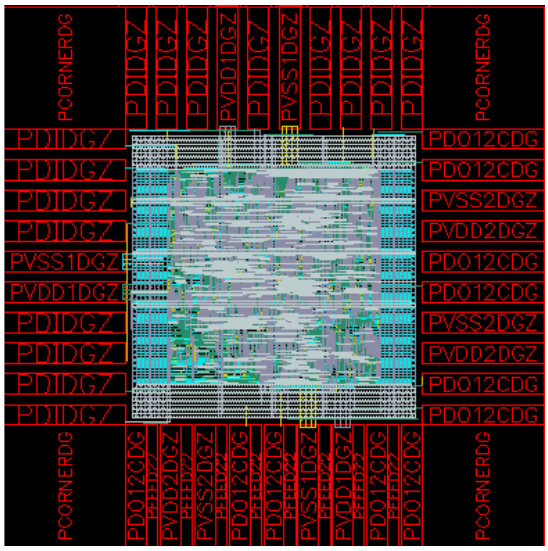

圖.11 32 位元 AES layout

參考文獻

[1] Chih-Pin Su, Tsung-Fu Lin, Chih-Tsun Huang ,and Cheng-Wen Wu,"A High-Throughput Low-Cost AES Processor,"*IEEE Communications Magzine*,Vol. 41, pp. 86-91,Dec. 2003.

[2] A. Rudra, P. K. Dubey, C. S. Jutla, V. Kumar, J. R. Rao, and P. Rohatgi, "Efficient Implementation of Rijndael Encryption with Composite Field Arithmetic," *Proceedings of Cryptographic Hardware and Embedded Systems (CHES2001)*, pp. 171-184, Paris, France, May. 2001.

[3] B. Sunar,E. Savas,Cetin K. Koc,"Constructing Composite Field Representations for Efficient Conversion,"*IEEE Transactions On Computers*,Vol. 52,No.11, pp. 1391-1398, Nov. 2003.

[4] X.Zhang,and K.K.Parhi,"On the Optimum Constructions of Composite Field for the AES Algorithm,"*IEEE Transactions On Circuits and Systems II Express Briefs*,Vol.53,No.10, pp. 1153-1157, Oct. 2006.

[5] H.Li,and J.Li,"A New Compact Architecture for AES with Optimized ShiftRows Operation,"*IEEE International Symposium on Circuits and Systems,2007.ISCAS 2007.* pp.1851-1854, May 2007.

[6] X. Zhang, and K. K. Parhi, "High-Speed VLSI Architectures for AES Algorithm," *IEEE Transactions on Very Large Scale Integration Systems*,Vol.12, pp. 957-967, Sept. 2004.

[7] A.Satoh,S.Morioka,K.Takano,and S.Munetoh,"A Compact Rijndael Hardware Architecture with S-Box Optimization,"C.Boyd(Ed.):ASIACRYPT 2001,LNCS 2248,pp.239-254, 2001.

[8] M.H.Jing,J.H.Chen,and Z.H.Chen,"Diversified MixColumn Transformation of AES,"*Information, Communications & Signal Processing, 2007 6th International Conference on*.pp.1-3, Dec. 2007.

[9] C.Paar,"Efficient VLSI architecture for bit-parallel computations in Galois field,"Ph.D. dissertation, Institute for Experimental Mathemetics, University of Essen, Germany, 1994.

[10] C. C. Lu and S. Y. Tseng, "Integrated Design of AES (Advanced Encryption Standard) Encrypter and Decrypter," *Proceedings of Application-Specific Systems, Architectures and Processors*, pp. 277-285, July 2002.

[11] H. Kuo and I. Verbauwhede, "Architectural Optimization for a 1.82 Gbits/sec VLSI Implementation of the AES Rijndael Algorithm," *Proceedings of Cryptographic Hardware and Embedded Systems*, pp. 51-64, May 2001.

[12] N. Sklavos and O. Koufopavlou, "Architecture and VLSI Implementation of the AES-Proposal Rijndael," *IEEE Transactions on Computers*, Vol. 51, pp. 1454-1459, Dec. 2002.

[13] S. Mangard, M. Aigner, and S. Dominikus,"A highly regular and scalable AES hardware architecture," *IEEE Transactions on Computers*,Vol. 52, pp. 483-491, April 2003.