


# 高效率的整合 AES 加密器與解密器之電路設計

學生：施忠宏

指導教授：紀翔峰 博士

國立交通大學電機資訊學院 電信學程（研究所）碩士班

## 摘要



前瞻加密標準分成加密演算法與解密演算法兩大部分，本篇論文提出一個新的演算法，用來簡化前瞻加密標準中的混行模組，並設計出一套架構整合加解密演算法，降低積體電路的複雜度進而節省硬體成本。此架構共分成四個模組。位元組替換模組：在轉換與反轉換的過程中，共用同一個乘法反元素轉換表。混行模組：一次考量四位元組來計算陣列，共用轉換與反轉換兩個陣列相同的部分。移列模組：透過硬體接線。密鑰擴充模組：利用即時運算架構來替換以記憶體為基礎的架構。本篇論文利用此架構，設計出資料路徑為 128 位元以及 32 位元兩種硬體電路，以符合不同的應用需求。前者的特性在於較高的資料總輸出量，而後者的特性在於較低的硬體總閘數。此兩種硬體電路以 Verilog HDL 來描述，並使用 TSMC 0.18um CMOS 標準元件庫來合成，對於資料路徑為 128 位元的硬體電路，其工作頻率為 200 MHz，平均資料總輸出量為 1.773 Gbits/s，硬體總閘數為 29.6K；資料路徑為 32 位元的硬體電路，其工作頻率為 270 MHz，平均資料總輸出量為 598 Mbits/s，硬體總閘數為 12.8 K。

**關鍵詞：**前瞻加密標準，乘法反元素

# Circuit Design of Highly Efficient AES Integrated Encryptor and Decryptor

Student : Chung-Hung Shih    Advisor : Dr. Hsiang-Feng Chi

Degree Program of Electrical Engineering Computer Science  
National Chiao Tung University

## Abstract

The algorithm of Advanced Encryption Standard (AES) is divided into encryption and decryption. This paper proposes a new algorithm to simplify MixColumn and Inverse MixColumn of AES and designs hardware architecture to integrate the algorithm of encryption and decryption. This architecture can reduce the hardware complexity and save the cost consumption. It is separated by four modules. SubBytes module—SubBytes and Inverse SubBytes employ one common look-up table only, i.e., multiplicative inverse table. MixColumns module—This module calculates four bytes array per time and shares their operations between transform and inverse transform. ShiftRows module—This module is realized by the hard-wired method. Key Expansion Module—The on-the-fly operation is used to replace the memory-based. Hence, base on the architecture, this paper designs two kinds of hardware, 128-bit datapath and 32-bit datapath, for a specified application. The former is for the purpose of higher data throughput and the latter is for the purpose of lower hardware gate count. Both circuits of hardware were described using Verilog HDL, synthesized by Synopsys with a 0.18um TSMC standard cell library. The simulation results show that a 128-bit datapath of AES circuit can operate at 200MHz and a 32-bit datapath of AES circuit can operate at 270MHz. The former and the latter can be done at a target average throughput of 1.773 Gbits/s and 598 Mbits/s individually. The hardware cost of the former and the latter have the gate count of 29.6 K and 12.8 K individually.

**Keyword: AES, Multiplicative Inverse**

# 誌 謝

首先感謝指導老師紀翔峰博士這兩年以來在本論文整個研究過程中的悉心指導，紀老師總是能在我遭遇瓶頸時給我一記當頭棒喝，引導出解決方向，使我能順利完成學業。感謝曾坤源同學在論文寫作上的指導與支援，並感謝鐘廷章和陳仁賢在設計合成工具的使用上給予莫大的協助。

最後感謝我的家人，尤其是我的賢內助倩萍，她總是能在我最失意時，給予我最大的支持與鼓勵，使我能專注並致力於研究上，而順利完成論文。也要感謝我的小兒翔瀚，每次觀察其言行舉止，他一些意想不到的表情、動作及發聲，總是能抒解我的壓力並且觸發我的研究靈感。

對於整個電信研究所學程，總是覺得自己得之於人者太多，出之於己者少。對於所有曾經指導過我的老師，同學及朋友們致上我無限的感謝。



# 目 錄

中文提要	.....	i
英文提要	.....	ii
誌謝	.....	iii
目錄	.....	iv
表目錄	.....	vi
圖目錄	.....	vii
一、	緒論.....	1
1.1	研究動機.....	1
1.2	研究方向.....	1
1.3	章節概要.....	2
二、	AES 加密及解密演算法.....	4
2.1	位元輸入輸出順序與中間狀態的產生.....	4
2.2	數學基礎.....	5
2.2.1	有限場的加法.....	6
2.2.2	有限場的乘法.....	6
2.3	加密及解密演算法.....	7
2.3.1	位元組替換之轉換與反轉換.....	9
2.3.2	移列之轉換與反轉換.....	11
2.3.3	混行之轉換與反轉換.....	12
2.3.4	每回合密鑰的相加.....	14
2.4	密鑰的排程.....	15
2.4.1	每回合密鑰的擴充.....	15

2.4.2	每回合密鑰的選擇.....	18
2.5	加密及解密之程序.....	19
三、	AES 硬體架構及電路設計.....	23
3.1	整合位元組替換轉換與反轉換模組的架構.....	23
3.2	整合移列轉換與反轉換模組的架構.....	25
3.3	整合混行轉換與反轉換模組的架構.....	26
3.4	密鑰擴充模組的架構.....	35
四、	電路設計的結果與分析.....	41
4.1	整合後之 AES 硬體架構.....	41
4.1.1	資料路徑為 128 位元之架構.....	42
4.1.2	資料路徑為 32 位元之架構.....	43
4.2	設計結果的比較與分析.....	45
五、	結論與未來展望.....	48
5.1	結論.....	48
5.2	未來展望.....	48
參考文獻	.....	49
附錄一	.....	51
自傳	.....	52

## 表 目 錄

表 2-1	Nk-Nb-Nr 的組合.....	8
表 2-2	加密與解密過程每個回合所使用到的密鑰.....	16
表 3-1	整合 1 位元組的 MixColumns/InvMixColumns 運算步驟.....	31
表 3-2	MC_InvMC_3 與 MC_InvMC_Ours 運算次數之比較.....	34
表 3-3	加密與解密過程每個回合 Rcon 的輸出值.....	38
表 4-1	資料路徑 128 位元的 AES-128 硬體架構所使用到的硬體模組數目..	41
表 4-2	資料路徑 32 位元的 AES-128 硬體架構所使用到的硬體模組數目..	42
表 4-3	三種整合 MixColumns 及 InvMixColumns 的硬體電路閘數比較表..	45
表 4-4	不同 AES 的硬體設計之比較.....	46



## 圖 目 錄

圖 2-1	狀態陣列的輸入與輸出.....	5
圖 2-2	加密過程的虛擬碼.....	8
圖 2-3	解密過程的虛擬碼.....	9
圖 2-4	SubBytes 和 InvSubBytes 運算對狀態陣列的影響.....	10
圖 2-5	ShiftRows 運算對狀態陣列的影響.....	11
圖 2-6	InvShiftRows 運算對狀態陣列的影響.....	12
圖 2-7	MixColumns 和 InvMixColumns 運算對狀態陣列的影響.....	12
圖 2-8	AddRoundKey 的運算.....	14
圖 2-9	Key Expansion 程序之虛擬碼.....	17
圖 2-10	RotWord 之運算.....	17
圖 2-11	加密程序之方塊圖.....	19
圖 2-12	解密程序之方塊圖.....	20
圖 2-13	等效解密程序之方塊圖.....	21
圖 3-1	整合 1 位元組的 S-box 和 Inverse S-box 硬體架構圖.....	25
圖 3-2	整合 16 位元組的 S-box 和 Inverse S-box 硬體方塊圖.....	26
圖 3-3	整合 16 位元組的 ShiftRows 和 InvShiftRows 硬體架構圖.....	27
圖 3-4	整合 4 位元組的 MixColumns 和 InvMixColumns 硬體架構圖....	34
圖 3-5	整合 16 位元組的 MixColumns 和 InvMixColumns 硬體方塊圖...	35
圖 3-6	SubWord 運算之硬體方塊圖.....	36
圖 3-7	RotWord 運算之硬體方塊圖.....	36
圖 3-8	用邏輯函數實現 Rcon 的方塊圖.....	37
圖 3-9	用查表方式實現 Rcon 的方塊圖.....	37

圖 3-10	AES-128 Key Expansion 模組硬體架構圖.....	39
圖 3-11	使用Equivalent Inverse Cipher時，Key Expansion模組每個輸出端的 硬體架構圖.....	40
圖 4-1	資料路徑為 128 位元的 AES-128 硬體架構圖.....	43
圖 4-2	資料路徑為 32 位元的 AES-128 硬體架構圖.....	44





# 第一章 緒論

## 1.1 研究動機

不論無線通訊網路或者是有線通訊網路，在最近幾十年來，隨著半導體技術不斷提昇，電腦網路皆被大量並廣泛應用在實際的生活中，並且藉由網路縮短人與人之間的距離，淡化國與國之間的界限。但是伴隨而來的問題是，如何提昇流通在網路上資訊的安全性則是當務之急。

欲提昇資料的安全，則必須在傳送端對原始的資料(plaintext)加密成密文(ciphertext)，而在接收端，對已加密過的資料解密。加密及解密資料的過程中，需要有一個快速並可靠的演算法，對資料做運算。美國標準和科技機構(the National Institute Standard and Technology, NIST)在 2001 年發表前瞻加密標準(Advanced Encryption Standard, AES)[1]，用來制定資料加密及解密的演算法標準，其前身為 Rijndael 加密及解密的演算法[2]。前瞻加密標準是為了用來取代傳統的資料加密標準(Data Encryption Standard, DES)[3]，因此，本論文採用前瞻加密標準，來實現對資料的加密與解密的演算法。

## 1.2 研究方向

實現一個 AES 資料加解密演算法，可從撰寫軟體原始碼[2,4]或者從設計硬體電路來達成，其中實現硬體電路又可分為是以 FPGA[5,6]或者是 ASIC[4,7,8,12]方式來實現，撰寫軟體原始碼方式的優點在於當演算法更改時，軟體的原始碼可以很容易的同步更改，因此彈性較大。缺點是其資料輸出量(throughput)非常低。對於只能以硬體來實現或者需要較高的資料輸出量的應用，如:智慧卡(smart card)及各種的資料儲存卡(flash

card)，會是一個很大的問題。

相對於撰寫軟體原始碼方式而言，以硬體電路來達成資料加解密會有很高的資料輸出量，適合高速的應用。缺點是需要有硬體電路的製造成本，以及當演算法的標準改變或增加時，會有硬體電路不容易被更改的問題。

本論文以硬體電路來實現前瞻加密標準，原因是前瞻加密標準已是一個全球公認的加密標準，其演算法不致於有大幅度更改，即使有更改或增加功能，通常標準制定者會考量往上相容的因素。因此，本篇論文研究方向在於如何採用前瞻加密標準中的演算法以簡化設計硬體電路，並且共用硬體資源以降低設計硬體電路的複雜度及提昇資料輸出量。



### 1.3 章節概要

本論文共分為五章，茲將各章概要說明如下：

#### 第一章 緒論：

說明本篇論文的研究動機、研究方向及章節概要。

#### 第二章 AES 加密及解密演算法：

詳細介紹 AES 標準，並且補充其不足之處，包括所需用到的數學，密鑰的排程，加解密演算法中的位元組替代轉換與反轉換，以及整個加解密程序，此章用來建立第三章硬體架構之基礎。

#### 第三章 AES 硬體架構及電路設計：

描述本論文如何以硬體電路來實現 AES，整個 AES 硬體架構分成四個模組：密鑰擴充模組、位元替換模組、移列模組及混行模組，此章也會討論到每個模組中，在轉換與反轉換的過程中，如何共用硬體資源。

#### 第四章 電路設計的結果與分析：

整合 AES 各個子模組所產生的硬體架構，本論文實現兩種硬體架構，資料路徑為 128 位元及 32 位元之架構，並且將本論文的 AES 硬體設計結果與其它的 AES 硬體設計結果做比較與分析。

#### 第五章 結論與未來展望：

對本篇論文所提出的方法下結論，並說明未來改進的方向。



## 第二章 AES 加密及解密演算法

AES 是一個經由輸入資料區塊後，對這些資料區塊做反覆運算，最後產生輸出資料區塊的加解密演算法，在它規範的標準中[1]，明確地定義出資料區塊的長度為 128 位元及密鑰的長度可以是 128，192，或者是 256 位元，對於這三種組合，AES 各付予它們為”AES-128”，“AES-192”，“AES-256”的名稱。

### 2.1 位元輸入輸出順序與中間狀態的產生

位元輸入輸出順序：

首先，對 128 位元的資料區塊做輸入，其位元輸入順序以及相對應的位元組，其關係式如下所示：

$$in_0 = \{ b_{127}, b_{126}, \dots, b_{120} \}$$

$$in_1 = \{ b_{119}, b_{118}, \dots, b_{112} \}$$

.....

$$in_{15} = \{ b_7, b_6, \dots, b_0 \}$$

其中  $in_m$  表示第  $m$  位元組的輸入資料， $bn$  表示第  $n$  位元的輸入資料。

同理，輸出 128 位元的資料區塊，其位元輸出順序以及相對應的位元組，會有如下的關係式：

$$out_0 = \{ b_{127}, b_{126}, \dots, b_{120} \}$$

$$out_1 = \{ b_{119}, b_{118}, \dots, b_{112} \}$$

.....

$$out_{15} = \{ b_7, b_6, \dots, b_0 \}$$

其中  $out_p$  表示第  $p$  位元組的輸出資料， $bn$  表示第  $n$  位元的輸出資料。

狀態的產生：

AES 演算法在運算過程中所產生的中間值，稱為狀態(state)。它是一個以位元組為單位所構成的二維陣列，此二維陣列稱為狀態陣列(state array)。此陣列大小為  $4 * Nb$  或  $4 * Nk$ ， $Nb$  為區塊長度除以 32 位元求得，而  $Nk$  為密鑰長度除以 32 位元求得。例如：當一筆 128 位元資料輸入時，即 16 位元組，在運算過程中會轉換成  $4 * 4$  的陣列，組成一個狀態。如圖 2-1。

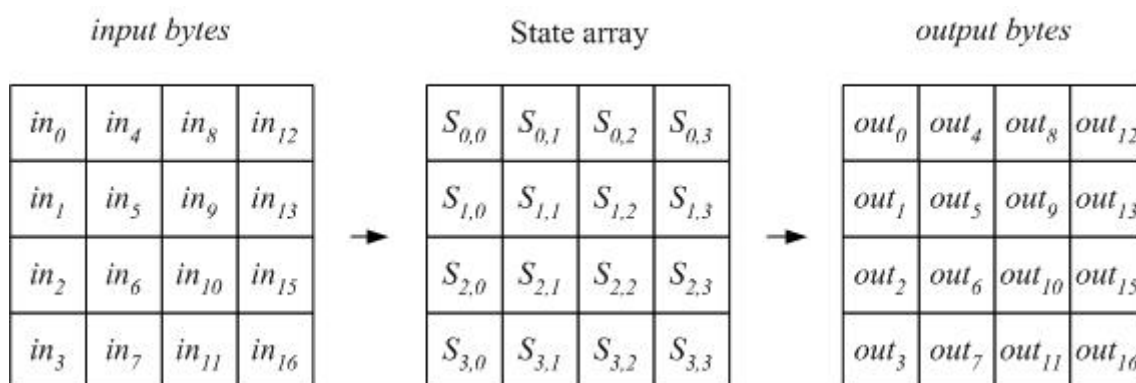


圖 2-1 狀態陣列的輸入與輸出

## 2.2 數學基礎

蓋羅瓦理論(Galois Theory)[9]中的蓋羅瓦場(Galois Field, GF)奠定 AES 標準的數學基礎。並且 AES 演算法中的加法和乘法是以  $GF(2^8)$  為基礎架構，我們稱一個蓋羅瓦場為一個有限場(final field)。

一個在  $GF(2^8)$  集合內的任何一個元素，可以表示成一個 7 次多項式：

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0$$

其中係數  $b_7 \sim b_0$  不是 0 就是 1

例如： $(57)_{16} = (01010111)_2$  以多項式表示為  $x^6 + x^4 + x^2 + x + 1$

## 2.2.1 有限場的加法

$GF(2^8)$ 集合內任意兩元素轉換成兩多項式的加法，其定義為相同指數係數總合再做模運算 2 (modulo 2)，簡單地說，就是將兩元素的每個位元互相做互斥或(eXclusive OR，XOR)運算，因此，在硬體設計上，用互斥或閘(XOR gate)即可達成有限場的加法。

假設在  $GF(2^8)$ 集合內的兩元素分別表示兩位元組的資料 Data (A)和 Data (B)

其中  $Data (A) = \{a_7, a_6, \dots, a_0\}$

$Data (B) = \{b_7, b_6, \dots, b_0\}$

則  $Data (A) + Data (B) = \{a_7 \oplus b_7, a_6 \oplus b_6, \dots, a_0 \oplus b_0\}$

例如： $(57)_{16} + (83)_{16} = (01010111)_2 \oplus (10000011)_2 = (11010100)_2 = (D4)_{16}$  或者以多項式表示為  $(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2$



## 2.2.2 有限場的乘法

對於  $GF(2^8)$ 的定義，在  $GF(2^8)$ 中的兩元素或兩多項式相乘的結果，也必須在  $GF(2^8)$ 的集合內，但是兩元素或兩多項式相乘會產生溢位的問題。因此，為了解決溢位問題，則必須將相乘的結果再做模運算一個不可分解的多項式  $m(x)$ 。在 AES 演算法中此多項式如(2-1)

$$m(x) = x^8 + x^4 + x^3 + x + 1 \quad (2-1)$$

或表示成 $(11B)_{16}$ 。例如， $(57)_{16} \cdot (13)_{16} = (FE)_{16}$  其計算方式如下：

$$(57)_{16} = (x^6 + x^4 + x^2 + x + 1)$$

$$(13)_{16} = (x^4 + x + 1)$$

$$\text{則 } (x^6 + x^4 + x^2 + x + 1) \cdot (x^4 + x + 1) = x^{10} + x^8 + x^7 + x^3 + 1$$

因為其相乘結果已經大於 7 次方，產生溢位，所以必須將其相乘結果再模運算  $m(x)$

$$(x^{10} + x^8 + x^7 + x^3 + 1) \text{ modulo } (x^8 + x^4 + x^3 + x + 1)$$

$$= x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x$$

$$= (\text{FE})_{16}$$

有另一種方法可以得出兩元素相乘的結果，稱之為 xtime。說明如下：

若把  $b(x)$  乘上  $x$ ，即乘上  $(02)_{16}$ ，換句話說， $b(x)$  多一倍，得到  $b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x$ 。若  $b_7=0$ ，不會發生溢位問題，答案即是正確的；若  $b_7=1$ ，發生溢位問題，必須減去  $m(x)$ 。我們可以把這種運算表示為  $\text{xtime}(x)$ ，其運算方式是將  $b(x)$  左移(left shift)，若發生溢位則和  $(1B)_{16}$  做互斥或運算。再舉  $(57)_{16} \cdot (13)_{16} = (\text{FE})_{16}$  的例子作說明：

$$(57)_{16} \cdot (02)_{16} = \text{xtime}(57) = (\text{AE})_{16}$$

$$(57)_{16} \cdot (04)_{16} = \text{xtime}(\text{AE}) = (47)_{16}$$

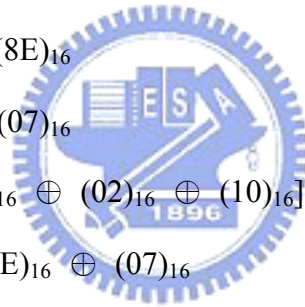
$$(57)_{16} \cdot (08)_{16} = \text{xtime}(47) = (8\text{E})_{16}$$

$$(57)_{16} \cdot (10)_{16} = \text{xtime}(8\text{E}) = (07)_{16}$$

$$(57)_{16} \cdot (13)_{16} = (57)_{16} \cdot [(01)_{16} \oplus (02)_{16} \oplus (10)_{16}]$$

$$= (57)_{16} \oplus (\text{AE})_{16} \oplus (07)_{16}$$

$$= (\text{FE})_{16}$$



## 2.3 加密及解密演算法

由於 AES 是一個不斷反覆運算的加解密演算法，因此，運算的次數透過回合總數 (Number of Round, Nr) 來決定。AES-128, AES-192 以及 AES-256 其密鑰數目 (Number of Key, Nk)，區塊數目 (Number of Block, Nb)，和回合總數 (Nr) 之間的關係，如表 2-1 所示。



	Nk	Nb	Nr
<i>AES-128</i>	4	4	10
<i>AES-192</i>	6	4	12
<i>AES-256</i>	8	4	14

表 2-1 Nk-Nb-Nr 的組合

其中 Nr 可由(2-2)表示：

$$Nr = \max(Nb, Nk) + 6 \quad (2-2)$$

不論加密或解密的演算法，都是由一個初始回合(initial round)將區塊資料與密鑰相加，經過 Nr-1 個回合運算，以及最後回合(final round)運算所組成。除了初始回合，每個回合都包括位元組替代(SubBytes)，移列(ShiftRows)，混行(MixColumns)以及與回合密鑰相加(AddRoundKey)共四個運算，但是最後一回合並不包含混行運算。整個加密和解密過程的虛擬碼(pseudo code)如圖 2-2 和 2-3 所示。

```

Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
  byte state[4,Nb]

  state = in

  AddRoundKey(state, w[0, Nb-1])

  for round = 1 step 1 to Nr-1
    SubBytes(state)
    ShiftRows(state)
    MixColumns(state)
    AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
  end for

  SubBytes(state)
  ShiftRows(state)
  AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

  out = state
end

```

圖 2-2 加密過程的虛擬碼



### 2.3.1 位元組替換之轉換與反轉換

位元組替換之轉換(SubBytes)與反轉換(InvSubBytes)的主要目的是將狀態陣列中的每一個元素透過一個替換表(substitution table)替換成另一個元素，AES 稱轉換的替換表為 S-box，反轉換的替換表為 Inverse S-box，而且替換表是一對一對應的關係。此位元組轉換是一個以位元組為單位的非線性替換運算，並且是可逆的。

```
InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
  byte state[4,Nb]

  state = in

  AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

  for round = Nr-1 step -1 downto 1
    InvShiftRows(state)
    InvSubBytes(state)
    AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
    InvMixColumns(state)
  end for

  InvShiftRows(state)
  InvSubBytes(state)
  AddRoundKey(state, w[0, Nb-1])

  out = state
end
```

圖 2-3 解密過程的虛擬碼

建立轉換的替換表(S-box)必須經過兩個步驟的運算過程：

第一個步驟是先找出每個位元組在  $GF(2^8)$  中的乘法反元素(附錄一)；

第二個步驟是經過一個仿射(affine)轉換運算，定義如(2-3)。其中仿射轉換運算是指一個轉換先經過陣列的乘法運算，再做加法運算。

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \quad (2-3)$$

位元組替換(SubBytes)運算對狀態的影響，如圖 2-4 所示：

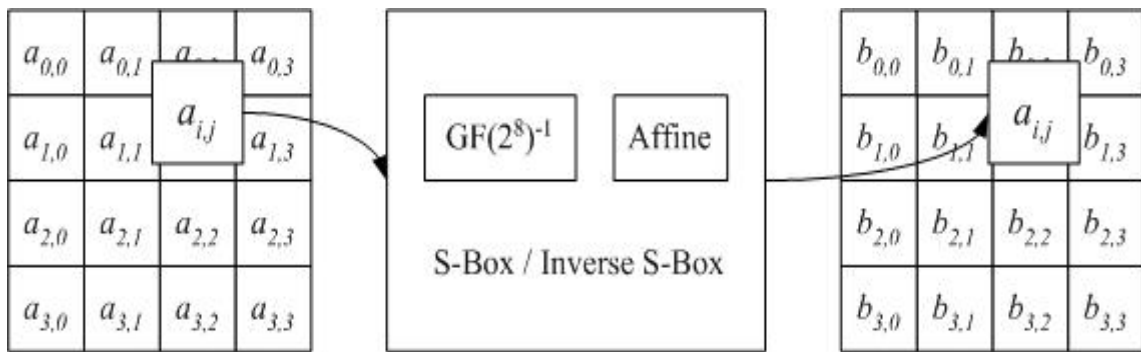


圖 2-4 SubBytes 和 InvSubBytes 運算對狀態陣列的影響

同理，建立反轉換的替換表(Inverse S-box)也必須經過兩個步驟的運算過程，它是建立轉換的替換表(S-box)的反運算：

第一個步驟是經過一個仿射轉換反運算，定義如(2-4)。

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \left( \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \right) \quad (2-4)$$

第二個步驟是找出每個位元組在  $GF(2^8)$  中的乘法反元素(附錄一)。

轉換與反轉換各有一個替換表。當得出位元組替換之轉換表後，也可以由它求出其反轉換表。例如， $(55)_{16}$  對應到位元組替換之轉換表的  $(FC)_{16}$ ，則  $(FC)_{16}$  對應到位元組替換之反轉換表的  $(55)_{16}$ ，其餘類推。

### 2.3.2 移列之轉換與反轉換

這個轉換屬於狀態陣列中的列運算。狀態陣列內的每一列以不同的偏移量做環狀位移，第 0 列不動，第一列左環狀位移 1 個位元組，第二列左環狀位移 2 個位元組，第三列左環狀位移 3 個位元組。因為 AES-128，AES-192 和 AES-256 的區塊的數目(Nb)皆為 4，所以它們的位移的偏移量皆相同。移列轉換(ShiftRows)運算對於狀態的影響，如圖 2-5 所示。



圖 2-5 ShiftRows 運算對狀態陣列的影響

移列反轉換(InvShiftRows)為移列轉換的反運算。第 0 列不動，第一列右環狀位移 1 個位元組，第二列右環狀位移 2 個位元組，第三列右環狀位移 3 個位元組。因為 AES-128，AES-192 和 AES-256 的區塊的數目(Nb)皆為 4，所以它們的位移的偏移量皆相同。移列反轉換運算對於狀態的影響，如圖 2-6 所示。

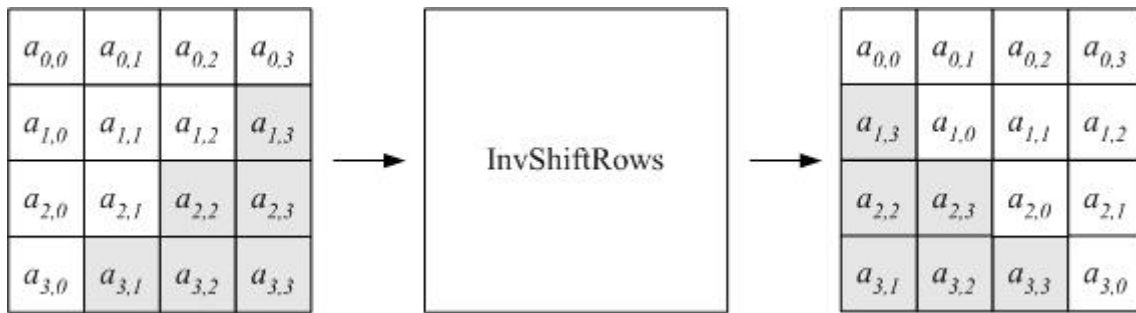


圖 2-6 InvShiftRows 運算對狀態陣列的影響

### 2.3.3 混行之轉換與反轉換

這個轉換屬於狀態陣列中的行運算，一次處理 4 位元組。狀態經過混行之轉換 (MixColumns) 與反轉換 (InvMixColumns) 運算之後的變化如下：

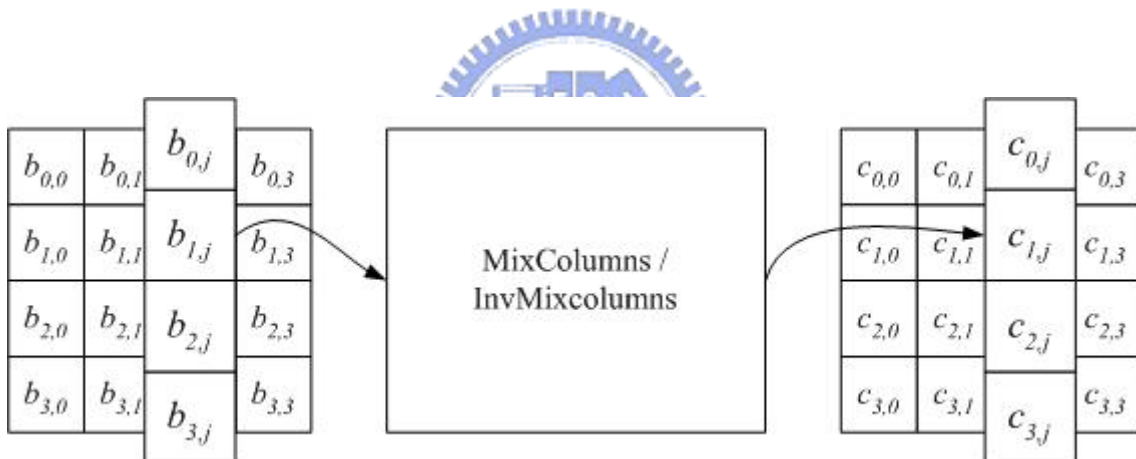


圖 2-7 MixColumns 和 InvMixColumns 運算對狀態陣列的影響

混行轉換運算把狀態陣列的每一行當作一個存在  $GF(2^8)$  中的三次多項式，如(2-5)，並且對一個固定的三次多項式  $c(x)$  作乘法，如(2-6)，兩個三次多項式相乘會產生一個六次的多項式(2-7)，如下所示：

$$a(x) = a_3x^3 + a_2x^2 + a_1x^1 + a_0 \quad (2-5)$$

$$b(x) = b_3x^3 + b_2x^2 + b_1x^1 + b_0 \quad (2-6)$$

$$c(x) = a(x) b(x) = c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x^1 + c_0 \quad (2-7)$$

其中  $c_0 = a_0 \cdot b_0$

$$c_1 = a_1 \cdot b_0 \oplus a_0 \cdot b_1$$

$$c_2 = a_2 \cdot b_0 \oplus a_1 \cdot b_1 \oplus a_0 \cdot b_2$$

$$c_3 = a_3 \cdot b_0 \oplus a_2 \cdot b_1 \oplus a_1 \cdot b_2 \oplus a_0 \cdot b_3$$

$$c_4 = a_3 \cdot b_1 \oplus a_2 \cdot b_2 \oplus a_1 \cdot b_3$$

$$c_5 = a_3 \cdot b_2 \oplus a_2 \cdot b_3$$

$$c_6 = a_3 \cdot b_3$$

為了維持兩個三次多項式相乘，其結果仍然為三次多項式，則必須將其結果再做模運算  $x^4 + 1$ ，並且在 AES 中可知道  $x^i \bmod (x^4 + 1) = x^{i \bmod 4}$ ，所以重寫(2-7)，得出(2-8)。

$$c(x) = a(x) b(x) = c_4 x^4 + c_3 x^3 + c_2 x^2 + c_1 x^1 + c_0 \quad (2-8)$$

其中  $c_0 = a_0 \cdot b_0 \oplus a_3 \cdot b_1 \oplus a_2 \cdot b_2 \oplus a_1 \cdot b_3$

$$c_1 = a_1 \cdot b_0 \oplus a_0 \cdot b_1 \oplus a_3 \cdot b_2 \oplus a_2 \cdot b_3$$

$$c_2 = a_2 \cdot b_0 \oplus a_1 \cdot b_1 \oplus a_0 \cdot b_2 \oplus a_3 \cdot b_3$$

$$c_3 = a_3 \cdot b_0 \oplus a_2 \cdot b_1 \oplus a_1 \cdot b_2 \oplus a_0 \cdot b_3$$

AES 定義混行轉換運算的  $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$ ，且與  $x^4 + 1$  互質。一個循環矩陣(circulant matrix)乘法可用來表示  $c(x) = a(x) \otimes b(x)$ ，如(2-9)。

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \quad (2-9)$$

混行反轉換為混行轉換的反運算。同理，將每行乘上一個三次多項式  $a^{-1}(x)$ 。即  $c(x) = a^{-1}(x) b(x)$ ， $a^{-1}(x)$  與  $a(x)$  之關係如(2-10)。

$$a^{-1}(x) \otimes a(x) = \{01\} \quad (2-10)$$

其中  $a^{-1}(x) = \{0B\}x^3 + \{0D\}x^2 + \{09\}x + \{0E\}$ ，若以一個循環矩陣(circulant matrix)

乘法來表示  $c(x) = a^{-1}(x) \otimes b(x)$ ，則如(2-11)。

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \quad (2-11)$$

### 2.3.4 每回合密鑰的相加

此小節在加密及解密的虛擬碼(表二及表三)為 AddRoundKey。其中每個回合密鑰(round key)的長度為 Nb 字組。這個運算主要是把每一個回合密鑰透過簡單的 bitwise XOR 加入到每一個狀態中，如圖 2-8 所示。

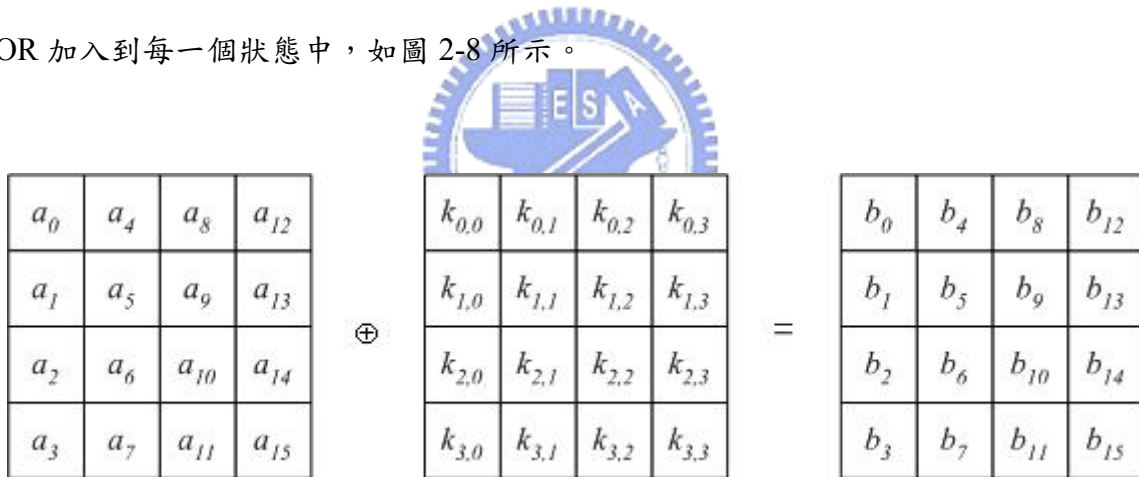


圖 2-8 AddRoundKey 的運算

不論是加密或解密過程中，都需要執行  $Nr + 1$  次的 AddRoundKey 運算，唯一不同的是加密與解密過程中每回合都會產生不同的密鑰，至於每回合的密鑰如何產生，則由下一節來說明之。

## 2.4 密鑰的排程

每個回合密鑰是從初始密鑰(initial key ; cipher key)經過運算產生出來的。密鑰排程(key schedule)是由密鑰擴充(key expansion)及回合密鑰的選擇(round key selection)組成的，基本的理論如下：

所有回合密鑰的總位元數是把區塊長度(block length)乘上回合數加1，所有回合數由第一次的初始輸入回合(round 0)，加上 $N_r - 1$ 個回合，以及第  $N_r$  次的終止回合(final round)所組成，例如，對於AES-128，128個位元的區塊長度經過10個回合運算，所需要用的所有回合密鑰的總位元數為 $128 * (10 + 1) = 1408$ 個位元。

擴充密鑰是由初始輸入的加密密鑰所擴充出來的。回合密鑰是從擴充密鑰中選出來的，選擇的方式為第一個回合密鑰由前 $N_b$ 個字組組成，第二個回合密鑰由接下來的 $N_b$ 個字組組成，餘此類推。



### 2.4.1 每回合密鑰的擴充

每個回合所擴充出來的密鑰，主要是用來提供給 AddRoundKey 使用。它屬於陣列中行的運算。

擴充後的密鑰是一個4個位元組的線性陣列，表示為 $w[i]$ ，其中  $0 \leq i < N_b(N_r + 1)$ 。所以每一個  $w[i]$  為一個字組，共有  $N_b(N_r + 1)$  個  $w[i]$ 。前  $N_k$  個字組包含了所輸入的加密密鑰。例如：對於AES-128而言，則  $N_r = 10$ ，密鑰長度為128位元，則  $N_k = 4$ ， $w[0]$ 至 $w[3]$ 字組為所輸入的密鑰，區塊長度為128位元，則  $N_b = 4$ ，全部密鑰的大小為

$w[4(10 + 1)] = w[44]$ 個字組，因此要在 $w[0] \sim w[43]$ 選擇每個回合要提供給AddRoundKey函數所使用密鑰。

解密過程中，每回合所使用的擴充密鑰，為反向提取加密過程中所產生的擴充密鑰，例如對於AES-128， $Nr = 10$ ，其解密過程中初始輸入密鑰為加密過程中的第十回合的擴充密鑰，如表2-2所示。

Round number	Round Key, encryption	Round Key, decryption
0(initial round)	w[0]~w[3]	w[40]~w[43]
1	w[4]~w[7]	w[36]~w[39]
2	w[8]~w[11]	w[32]~w[35]
3	w[12]~w[15]	w[28]~w[31]
4	w[16]~w[19]	w[24]~w[27]
5	w[20]~w[23]	w[20]~w[23]
6	w[24]~w[27]	w[16]~w[19]
7	w[28]~w[31]	w[12]~w[15]
8	w[32]~w[35]	w[8]~w[11]
9	w[36]~w[39]	w[4]~w[7]
10(final round)	w[40]~w[43]	w[0]~w[3]

表 2-2 加密與解密過程每個回合所使用到的密鑰

由於密鑰擴充演算法與加密和解密的演算法分開，因此會有一個獨立的虛擬碼，稱之為KeyExpansion，如圖 2-9 所示。



在圖 2-9 的副程式中，主要包含有三個運算方式：SubWord、RotWord 和 Rcon。分別敘述如下：

SubWord：輸入一個 4-byte 的字組，之後這個輸入的字組經過 S-box 的轉換產生相對應的字組，最後，傳回一個 4-byte 的字組。

RotWord：輸入一個 4-byte 的字組，之後這個輸入的字組經過經過一個位元組的旋轉運算，最後，傳回一個 4-byte 的字組。如圖 2-10 所示：

```

KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
  word temp

  i = 0

  while (i < Nk)
    w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
    i = i+1
  end while

  i = Nk

  while (i < Nb * (Nr+1))
    temp = w[i-1]
    if (i mod Nk = 0)
      temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
    else if (Nk > 6 and i mod Nk = 4)
      temp = SubWord(temp)
    end if
    w[i] = w[i-Nk] xor temp
    i = i + 1
  end while
end

```

圖 2-9 Key Expansion 程序之虛擬碼

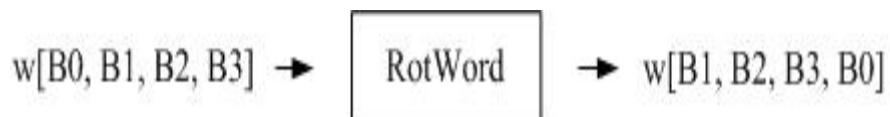


圖 2-10 RotWord 之運算

Rcon：輸入該回合的回合數，之後根據這個回合數計算出該回合的回合常數，最後傳回一個4-byte的字組。

上述回合常數定義如下：

$$\text{Rcon}[i/\text{Nk}] = (x^{(i/\text{Nk})-1}, \{00\}, \{00\}, \{00\}) \quad (2-12)$$

其中  $i/\text{Nk}$  為回合數，而  $x$  在  $\text{GF}(2^8)$  為 02，因此可將(2-12)重新表示成(2-13)

$$\text{Rcon}[i/\text{Nk}] = (\{02\}^{(i/\text{Nk})-1}, \{00\}, \{00\}, \{00\}) \quad (2-13)$$

例如，對於 AES-128 而言，其  $\text{Nr} = 10$ ，則加密過程中每個回合  $\text{Rcon}[i/\text{Nk}]$  產生的字組如下：

$$\text{Rcon}[1] = (01000000)_{16}$$

$$\text{Rcon}[2] = (02000000)_{16}$$

$$\text{Rcon}[3] = (04000000)_{16}$$

$$\text{Rcon}[4] = (08000000)_{16}$$

$$\text{Rcon}[5] = (10000000)_{16}$$

$$\text{Rcon}[6] = (20000000)_{16}$$

$$\text{Rcon}[7] = (40000000)_{16}$$

$$\text{Rcon}[8] = (80000000)_{16}$$

$$\text{Rcon}[9] = (1b000000)_{16}$$

$$\text{Rcon}[10] = (36000000)_{16}$$



## 2.4.2 每回合密鑰的選擇

第  $i$  個回合密鑰是指在存在回合密鑰緩衝區的字組  $W[\text{Nb} * i]$  到  $W[\text{Nb} * (i + 1) - 1]$ ，如表 2-2 所示。

## 2.5 加密及解密之程序

整個 AES 的加密及解密的程序可以由方塊圖 2-11 及圖 2-12 來描述之。

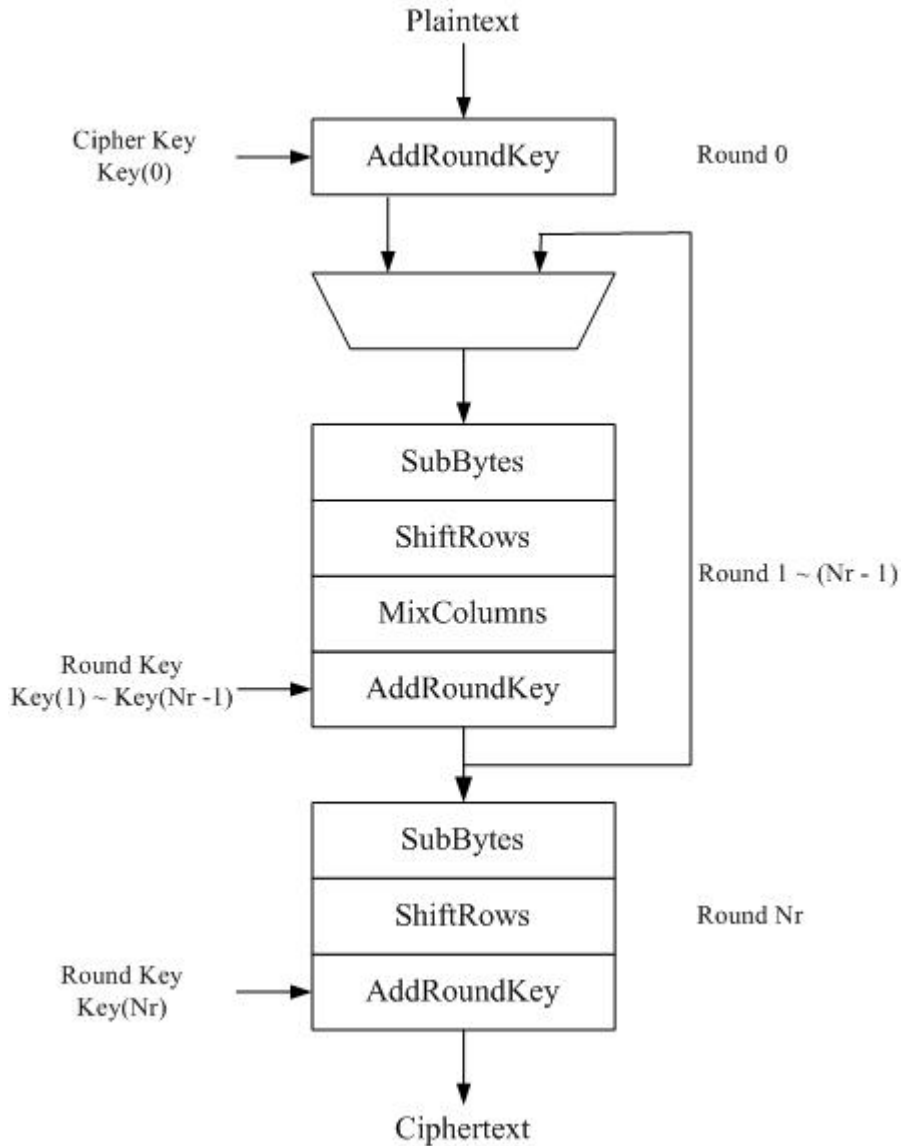


圖 2-11 加密程序之方塊圖

由圖 2-11 及圖 2-12 可以很明顯地看出，加密及解密的程序在 1~Nr 回合有很大的不同，這將提高硬體設計的困難度。所幸 AES 提供了另一個解密的程序，稱為等效解密

(equivalent inverse cipher)的程序，此等效解密程序與原先加密程序完全相同。但是，需要多增加一個運算。即在等效解密程序中每個回合所使用的擴充密鑰，必須經過混行反轉換的運算才能輸出給 AddRoundKey 做運算。等效解密的程序其方塊圖，如圖 2-13 所示。

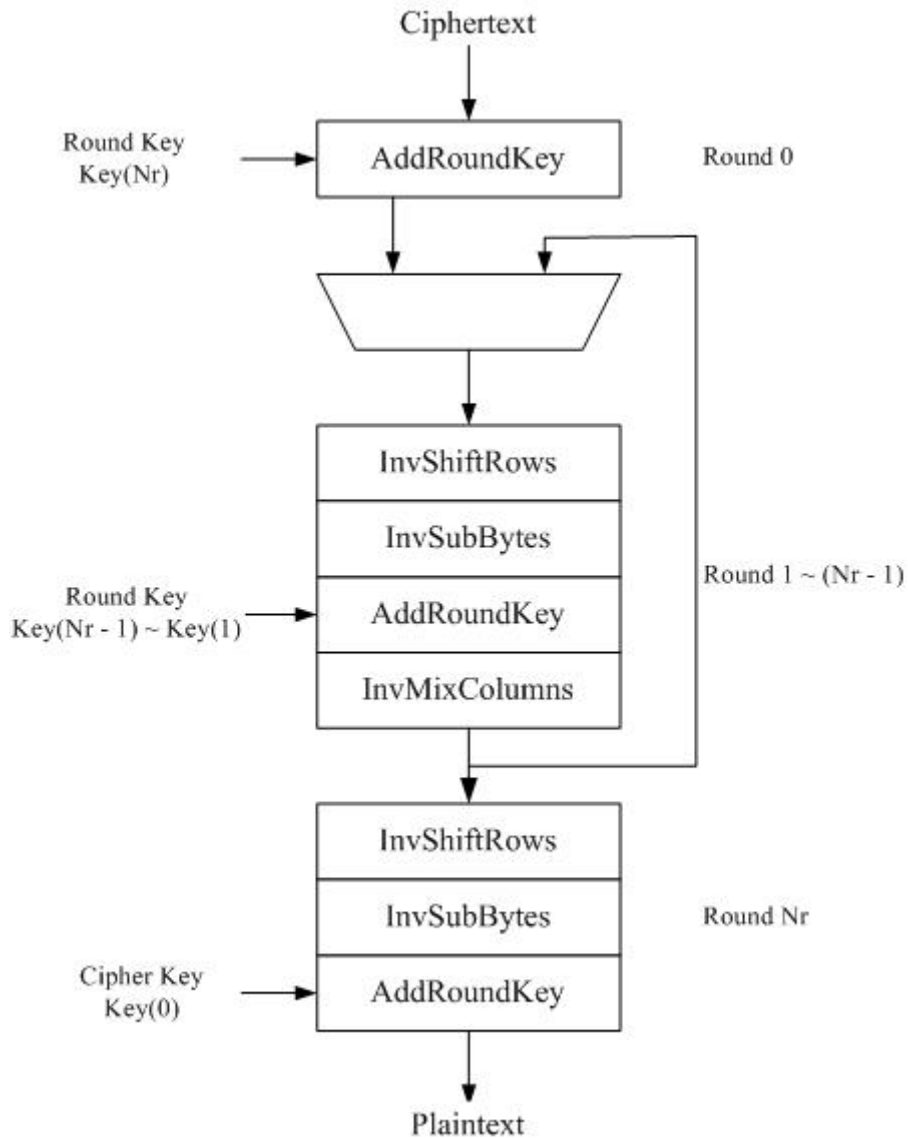


圖 2-12 解密程序之方塊圖

以下兩點將解釋為何可以改變原先解密的順序：

第一點是加密程序與解密程序中的 SubBytes 和 InvSubBytes 運算皆屬於單一位元組

的運算，所以可以將 SubBytes 和 InvSubBytes 運算與 ShiftRows 和 InvShiftRows 運算的處理順序對調。

第二點是 MixColumns 與 InvMixColumns 皆屬於線性運算，所以圖 2-12 的解密過程中先處理 AddRoundKey 的運算再處理 InvMixColumns 的運算對調成先處理 InvMixColumns 的運算再處理 AddRoundKey 的運算可以由(2-14)證明之：

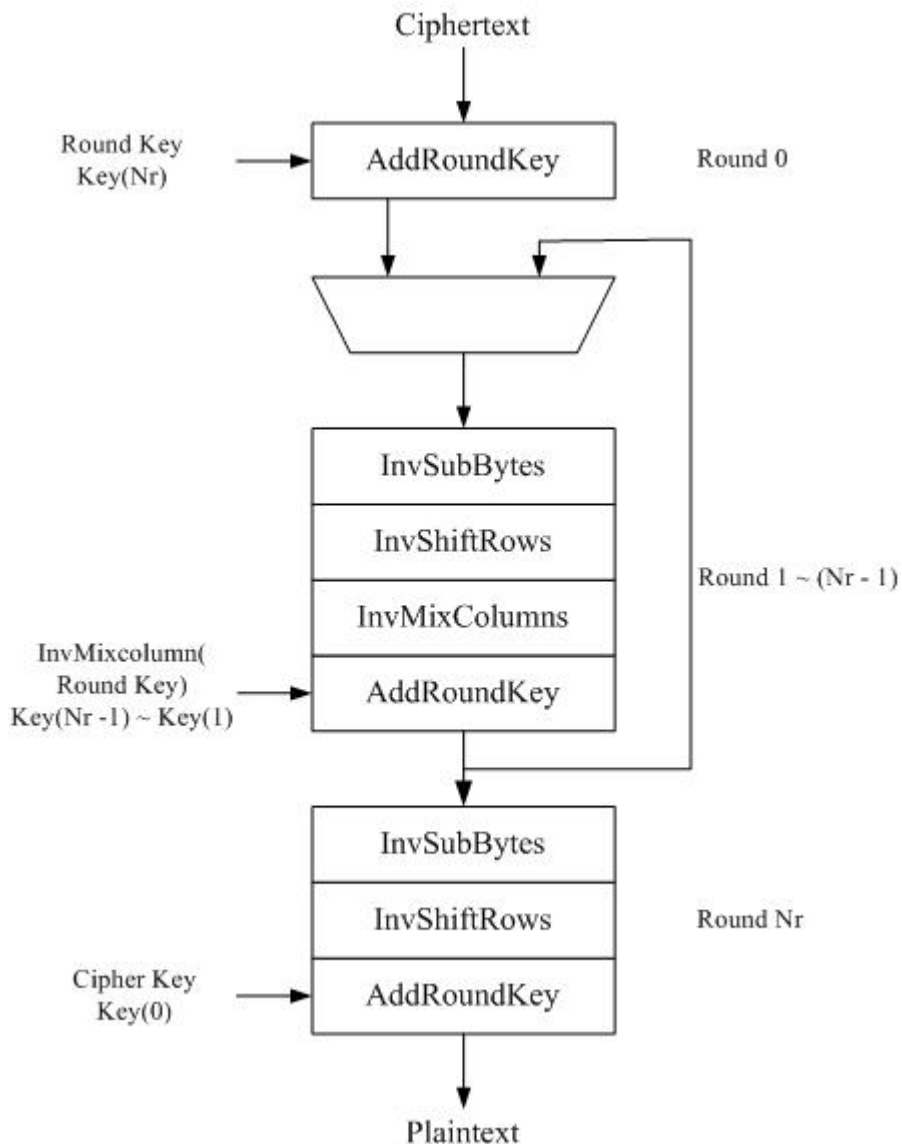


圖 2-13 等效解密程序之方塊圖

$$\begin{aligned}
& (\text{State}(i) \oplus \text{RoundKey}(i)) \otimes a^{-1}(x) \\
&= (\text{State}(i) \otimes a^{-1}(x)) \oplus (\text{RoundKey}(i) \otimes a^{-1}(x)) \\
&= \text{InvMixColumns}(i) \oplus \text{InvMixColumns}(\text{RoundKey}(i)) \tag{2-14}
\end{aligned}$$

其中  $1 \leq i \leq (Nr-1)$ ， $a^{-1}(x)$  為 2.3.3 節所定義的多項式。

因此，可以由原先在解密過程中先 AddRoundKey 再 InvMixColumns 對調成先 InvMixColumns 再與執行過 InvMixColumns 運算的該回合密鑰相加。



## 第三章 AES 硬體架構及電路設計

實現 AES 加密及解密演算法的硬體設計方法有二：

第一個方法是將加密及解密的部分各自分開設計成單一模組，最後再由多工器切換是加密模組或是解密模組的資料輸出[4]，這個方法在硬體設計的直覺上比較直接，但是，因為無法共用彼此的硬體資源，因此設計出來的硬體面積會較大。

第二個方法是將在加密程序的 SubBytes，ShiftRows 以及 MixColumns 與解密程序的 InvSubBytes，InvShiftRows 以及 InvMixColumns，還有 Key Expansion 彼此共用其相關的硬體，以達到縮小硬體面積的目的。例如：對於 SubBytes 和 InvSubBytes 如何減少 S-box 表格的使用量[7]或是不用建 S-box 表格的方式而是採用邏輯函數來實現 S-box 表格[10]。對於 MixColumns 和 InvMixColumns 是要利用固定係數的乘法器(fixed coefficient multiplier)來作矩陣的運算與反運算[11]，或者是有其他的方法可以簡化它們，因為矩陣的運算與反運算會比較複雜。對於 Key Expansion 是要在加密過程中對於每回合所產生的密鑰存入記憶體，之後提供給解密過程中每個回合的密鑰使用，或者是不使用密鑰存入記憶體的方式，而是在解密過程中，利用加密最後回合的密鑰來即時(on-the-fly)計算出解密的每個回合的密鑰，以節省硬體資源。

以下四節將描述各模組，如何有效地共用各自的轉換運算與反轉換運算產生的硬體，以達到硬體資源共享的目的。

### 3.1 整合位元組替換轉換與反轉換模組的架構

採用查表的方式來實現 SubBytes 及 InvSubBytes 運算，就執行速度而言，會比用邏

輯函數計算出 S-box 或者 Inverse S-box 內的替換值要來快很多[4]，唯一要解決的是，如何減少在硬體內 S-box 及 Inverse S-box 表格的數目。因為，若採用平行架構，對於 AES-128 而言，加密過程必須同時對 16 個位元組的資料區塊做查表的動作，因此需要使用 16 個 S-box，同理，解密過程也需要使用到 16 個 Inverse S-box，不論加密或解密過程，Key Expansion 使用 4 個 S-box，所以總共 36 個表格儲存在硬體中，再者每一個表格為 256 位元組，因此，總共需要 9216 位元組的記憶體來儲存。這會增加不少硬體的成本，以下敘述如何解決這個問題。

求出 S-box 需經過兩個步驟，用以下數學式來表示之：

$$y = M * \text{multiplicative\_inverse}(x) \oplus C \quad (3-1)$$

其中

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad C = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

(3-1)的反運算為：

$$x = \text{multiplicative\_inverse}^{-1}(M^{-1} * (y \oplus C)) \quad (3-2)$$

根據[7]可知  $\text{multiplicative\_inverse}^{-1}()$ 相等於  $\text{multiplicative\_inverse}()$ ，並且將  $M^{-1}$  用  $M'$  代替，故(3-2)可以表示成(3-3)

$$x = \text{multiplicative\_inverse}(M' * (y \oplus C)) \quad (3-3)$$

其中



$$M' = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

由(3-1)及(3-3)可知，SubBytes 以及 InvSubBytes 模組可以共用同一個 GF(2<sup>8</sup>)乘法反元素(附錄一)，因此，若採用平行架構，在加密及解密過程中，所使用的表格(S-box 和 Inverse S-box)就可以由 36 降為 20，硬體成本約少了 44%，此模組之架構圖如圖 3-1 所示，以及採用平行架構其方塊圖如圖 3-2 所示。

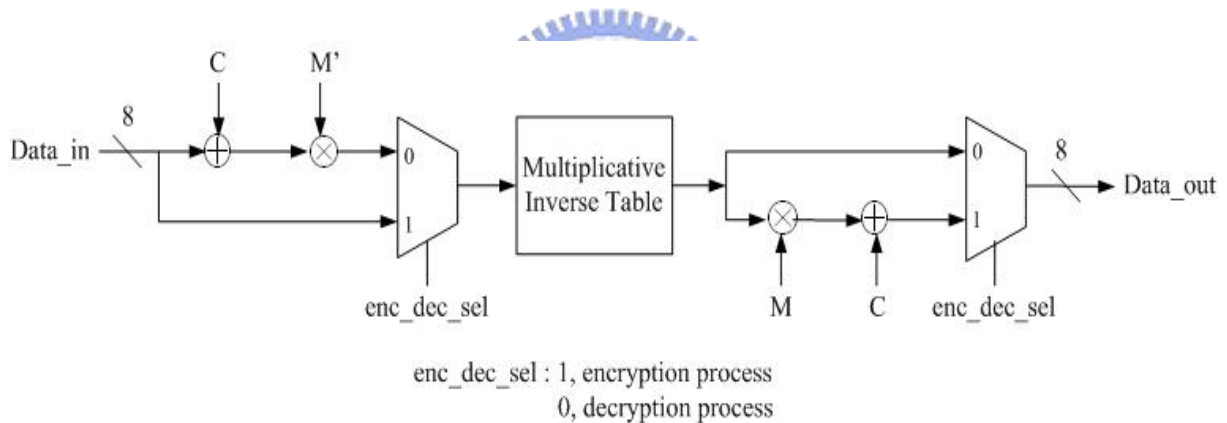


圖 3-1 整合1位元組的 S-box 和 Inverse S-box 硬體架構圖

### 3.2 整合移列轉換與反轉換模組的架構

用硬體電路實作移列轉換與反轉換時，並不需要用位移器(shifter)實際做列位移的運算，而是只要用硬體多工器及連線的方式來改變狀態陣列內元素的位置，即可輕易達成位移的功能。

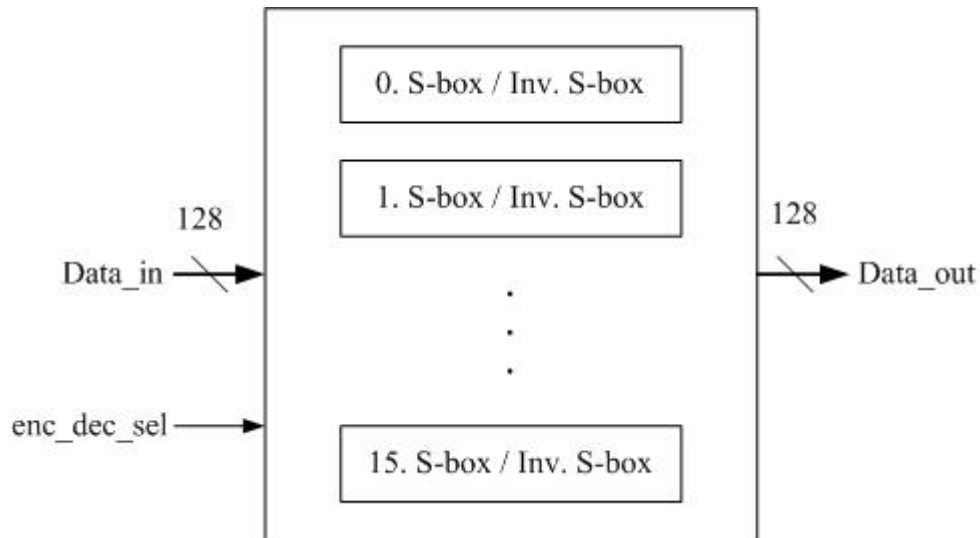


圖 3-2 整合16位元組的 S-box 和 Inverse S-box 硬體方塊圖

倘若我們先將 ShiftRows 及 InvShiftRows 分開設計，之後再整合，總共需要 12 組多工器。因為第 0 列位置不變，第一列，第二列和第三列都有改變位置，所以有 12 個位元組的資料需要做切換的動作。為了能達到 ShiftRows 及 InvShiftRows 能彼此共享硬體資源，不難看出圖 2-5 及 2-6，兩張圖彼此之間的關係。其中狀態陣列中的第 0 列以及第二列在狀態陣列中的每個資料，在經過 ShiftRows 及 InvShiftRows 運算後是相同的，因此可直接用硬體連線來實作此部分的電路。而另外狀態陣列中的第一列以及第三列，用 8 組多工器來作 ShiftRows 及 InvShiftRows 資料的切換。因此，可以由原來的 12 組多工器減少為 8 組多工器，省下了 33% 的硬體資源。此模組的硬體架構圖，如圖 3-3 所示。

### 3.3 整合混行轉換與反轉換模組的架構

由 2.3.3 節可知，對於 InvMixcolumns 所需要在矩陣內運算的係數  $(0B)_{16}$ ,  $(0D)_{16}$ ,  $(09)_{16}$  和  $(0E)_{16}$  會比在 MixColumns 所需要在矩陣內運算的係數 02, 03, 01 和 01 要複雜許多。換句話說，若各自分開做硬體設計，InvMixcolumns 模組的硬體面積會比 MixColumns 模組的硬體面積大上約 2.2 倍[12]。

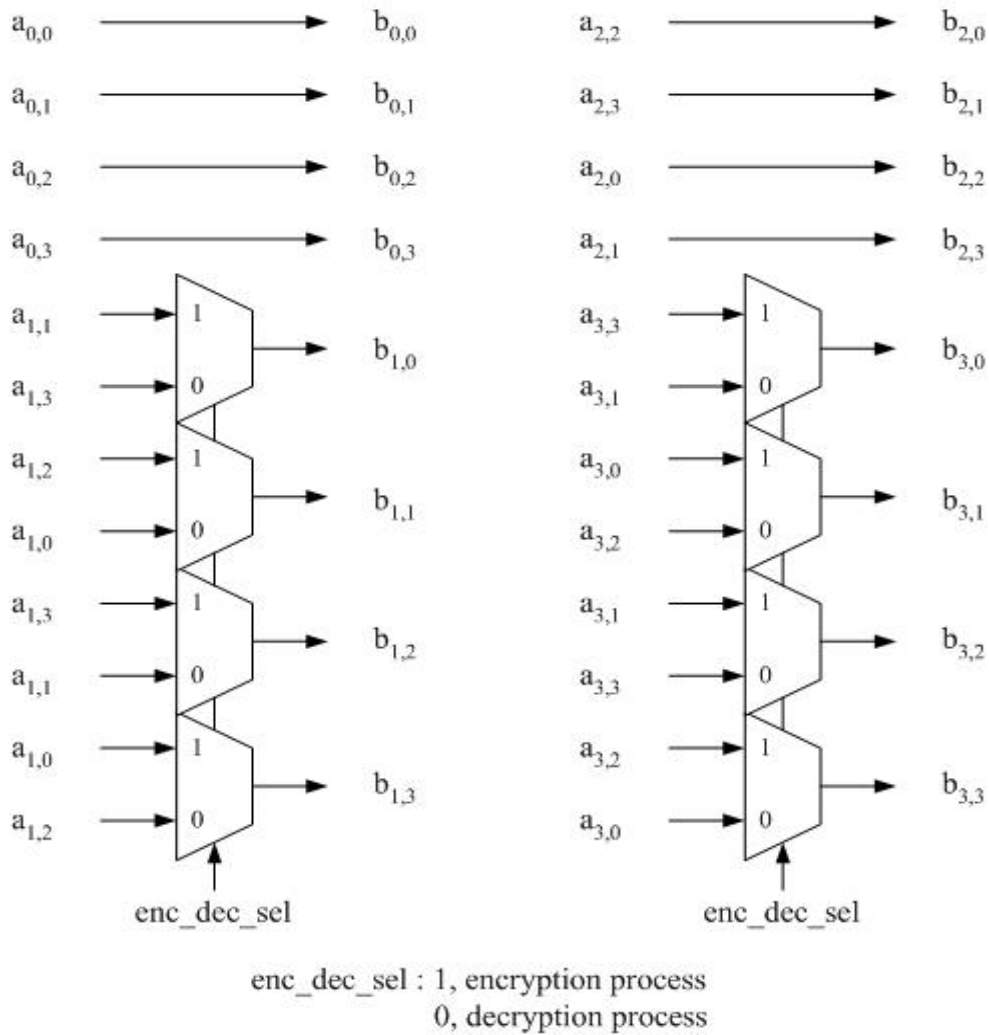


圖 3-3 整合16位元組的 ShiftRows 和 InvShiftRows 硬體架構圖

有三種方式可以用來實現整合混行轉換與反轉換模組的硬體電路：

第一種方式採用固定係數的乘法器(fixed-coefficient multiplier)來實現此模組的硬體電路[11]，此模組共有六個係數需要用硬體方式計算出來，之後本論文稱此種方式為 MC\_InvMC\_1。描述如下：

將  $B(x)$  表示成多項式

$$B(x) = b_0 + b_1x + b_2x^2 + b_3x^3 + b_4x^4 + b_5x^5 + b_6x^6 + b_7x^7$$

其中  $b_k \in (0, 1), 0 \leq k \leq 7$

則  $B(x)$  與六個係數相乘後，其結果如下：

$$\begin{aligned} \{02\} \cdot B(x) &= b_7 + (b_0 \oplus b_7) x + b_1 x^2 + (b_2 \oplus b_7) x^3 \\ &\quad + (b_3 \oplus b_7) x^4 + b_4 x^5 + b_5 x^6 + b_6 x^7 \end{aligned} \quad (3-4)$$

$$\begin{aligned} \{03\} \cdot B(x) &= (b_0 \oplus b_7) + (b_0 \oplus b_1 \oplus b_7) x + (b_1 \oplus b_2) x^2 + (b_2 \oplus b_3 \oplus b_7) x^3 \\ &\quad + (b_3 \oplus b_4 \oplus b_7) x^4 + (b_4 \oplus b_5) x^5 + (b_5 \oplus b_6) x^6 + (b_6 \oplus b_7) x^7 \end{aligned} \quad (3-5)$$

$$\begin{aligned} \{0D\} \cdot B(x) &= (b_0 \oplus b_5 \oplus b_6) + (b_1 \oplus b_5 \oplus b_7) x + (b_0 \oplus b_2 \oplus b_6) x^2 \\ &\quad + (b_0 \oplus b_1 \oplus b_3 \oplus b_5 \oplus b_6 \oplus b_7) x^3 + (b_1 \oplus b_2 \oplus b_4 \oplus b_5 \oplus b_7) x^4 \\ &\quad + (b_2 \oplus b_3 \oplus b_5 \oplus b_6) x^5 + (b_3 \oplus b_4 \oplus b_6 \oplus b_7) x^6 \\ &\quad + (b_4 \oplus b_5 \oplus b_7) x^7 \end{aligned} \quad (3-6)$$

$$\begin{aligned} \{0E\} \cdot B(x) &= (b_5 \oplus b_6 \oplus b_7) + (b_0 \oplus b_5) x + (b_0 \oplus b_1 \oplus b_6) x^2 \\ &\quad + (b_0 \oplus b_1 \oplus b_2 \oplus b_5 \oplus b_6) x^3 + (b_1 \oplus b_2 \oplus b_3 \oplus b_5) x^4 \\ &\quad + (b_2 \oplus b_3 \oplus b_4 \oplus b_6) x^5 + (b_3 \oplus b_4 \oplus b_5 \oplus b_7) x^6 \\ &\quad + (b_4 \oplus b_5 \oplus b_6) x^7 \end{aligned} \quad (3-7)$$

$$\begin{aligned} \{0B\} \cdot B(x) &= (b_0 \oplus b_5 \oplus b_7) + (b_0 \oplus b_1 \oplus b_5 \oplus b_6 \oplus b_7) x \\ &\quad + (b_1 \oplus b_2 \oplus b_6 \oplus b_7) x^2 + (b_0 \oplus b_2 \oplus b_3 \oplus b_5) x^3 \\ &\quad + (b_1 \oplus b_3 \oplus b_4 \oplus b_5 \oplus b_6 \oplus b_7) x^4 + (b_2 \oplus b_4 \oplus b_5 \oplus b_6 \oplus b_7) x^5 \\ &\quad + (b_3 \oplus b_5 \oplus b_6 \oplus b_7) x^6 + (b_4 \oplus b_6 \oplus b_7) x^7 \end{aligned} \quad (3-8)$$

$$\begin{aligned} \{09\} \cdot B(x) &= (b_0 \oplus b_5) + (b_1 \oplus b_5 \oplus b_6) x + (b_2 \oplus b_6 \oplus b_7) x^2 \\ &\quad + (b_0 \oplus b_3 \oplus b_5 \oplus b_7) x^3 + (b_1 \oplus b_4 \oplus b_5 \oplus b_6) x^4 \\ &\quad + (b_2 \oplus b_5 \oplus b_6 \oplus b_7) x^5 + (b_3 \oplus b_6 \oplus b_7) x^6 + (b_4 \oplus b_7) x^7 \end{aligned} \quad (3-9)$$

由(3-4)到(3-9)可知，{0D}，{0E}，{0B}和{09}四個固定係數的乘法會比較複雜，

倘若使用平行架構必須複製 16 個六個固定係數的乘法器來實現硬體電路，將會耗費不少硬體資源。

第二種方式是六個固定係數使用  $xtime$  的運算方式[4]，來取代固定係數的乘法器，之後本論文稱此種方式為  $MC\_InvMC\_2$ ，描述如下：

$$\{02\} \cdot B(x) = xtime(B(x)) \quad (3-10)$$

$$\begin{aligned} \{03\} \cdot B(x) &= (\{01\} \oplus \{02\}) \cdot B(x) \\ &= B(x) \oplus xtime(B(x)) \end{aligned} \quad (3-11)$$

$$\begin{aligned} \{0D\} \cdot B(x) &= (\{01\} \oplus \{04\} \oplus \{08\}) \cdot B(x) \\ &= B(x) \oplus xtime(xtime(B(x))) \oplus xtime(xtime(xtime(B(x)))) \end{aligned} \quad (3-12)$$

$$\begin{aligned} \{0E\} \cdot B(x) &= (\{02\} \oplus \{04\} \oplus \{08\}) \cdot B(x) \\ &= xtime(B(x)) \oplus xtime(xtime(B(x))) \oplus xtime(xtime(xtime(B(x)))) \end{aligned} \quad (3-13)$$

$$\begin{aligned} \{0B\} \cdot B(x) &= (\{01\} \oplus \{02\} \oplus \{08\}) \cdot B(x) \\ &= B(x) \oplus xtime(B(x)) \oplus xtime(xtime(xtime(B(x)))) \end{aligned} \quad (3-14)$$

$$\begin{aligned} \{09\} \cdot B(x) &= (\{01\} \oplus \{08\}) \cdot B(x) \\ &= B(x) \oplus xtime(xtime(xtime(B(x)))) \end{aligned} \quad (3-15)$$

此種方式同時處理  $MixColumns$  和  $InvMixColumns$  的 1 個位元組需要使用 14 個  $xtime$  及 14 個 XOR 運算。要實現(3-10)到(3-15)的硬體遠比實現(3-4)到(3-9)的硬體簡單許多，但是，倘若使用平行架構仍然必須複製 16 個硬體電路，也會耗費不少硬體資源。

第三種方式是將 MixColumns 和 InvMixColumns 兩矩陣運算結果中的 1 個位元組做整合[7,12]，之後本論文稱此種方式為 MC\_InvMC\_3。大致描述如下：

(3-16)及(3-17)各自可得出 MixColumns 和 InvMixColumns 矩陣運算結果中的 1 個位元組。

$$mc\_out = [2\ 3\ 1\ 1][a\ b\ c\ d]^T = 2a \oplus 3b \oplus c \oplus d \quad (3-16)$$

因此，MixColumns 處理 1 個位元組需要使用 2 個 xtime 及 4 個 XOR 運算，這也是 MC\_InvMC\_2 的 MixColumns 所需要用到的運算量。

$$inv\_mc\_out = [E\ B\ D\ 9][a\ b\ c\ d]^T = 14a \oplus 11b \oplus 13c \oplus 9d \quad (3-17)$$

因此，InvMixColumns 處理 1 個位元組需要使用 12 個 xtime 及 10 個 XOR 運算，這也是 MC\_InvMC\_2 的 InvMixColumns 所需要用到的運算量。

分解(3-16)得到(3-18)，分解(3-17)得到(3-19)

$$mc\_out = 2(a \oplus b) \oplus b \oplus (c \oplus d) \quad (3-18)$$

$$imc\_out = 4(2(a \oplus b) \oplus 2(c \oplus d) \oplus (a \oplus c)) \oplus 2(a \oplus b) \oplus b \oplus (c \oplus d) \quad (3-19)$$

整合(3-18)及(3-19)，實現其硬體之步驟[7]如表 3-1。

Step	Operation
1	$w1 = a \oplus b$
2	$w2 = a \oplus c$
3	$w3 = c \oplus d$
4	$w4 = \text{xtime}(w1)$
5	$w5 = \text{xtime}(w3)$
6	$w6 = w2 \oplus w4 \oplus w5$
7	$w7 = \text{xtime}(w6)$
8	$w8 = \text{xtime}(w7)$
9	$\text{mc\_out} = b \oplus w3 \oplus w4$
10	$\text{imc\_out} = w8 \oplus \text{mc\_out}$

表3-1 整合1位元組的 MixColumns/InvMixColumns 運算步驟

由表 3-1 可知，在步驟 10 的  $\text{imc\_out}$  使用到步驟 9 的  $\text{mc\_out}$ ，此即為硬體資源共享的特性。因此，當整合(3-18)及(3-19)，同時處理 MixColumns 和 InvMixColumns 的 1 個位元組只需使用 4 個  $\text{xtime}$  及 8 個 XOR 運算，與 MC\_InvMC\_2 的 14 個  $\text{xtime}$  及 14 個 XOR 運算比較起來少了許多。但是，如果採用平行架構，則仍然要複製 16 個此模組，共花費 64 個  $\text{xtime}$  及 128 個 XOR 運算。

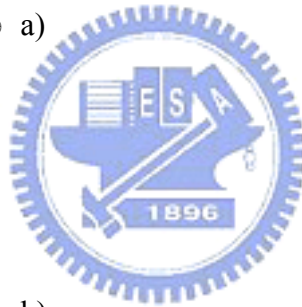
本論文提出另一種方法可以將 MixColumns 和 InvMixColumns 的整合做最佳化，就是使(3-20)及(3-25)以 4 個位元組來整合其運算式相同的部分，之後本論文稱此種方式為 MC\_InvMC\_Ours，描述如下：

對於 MixColumns 而言，其運算式如下：

$$\begin{bmatrix} W \\ X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \quad (3-20)$$

$$\begin{aligned} W &= 2a \oplus 3b \oplus c \oplus d \\ &= 2(a \oplus b) \oplus b \oplus (c \oplus d) \\ &= g_0 \oplus b \oplus f_2 \end{aligned} \quad (3-21)$$

$$\begin{aligned} X &= a \oplus 2b \oplus 3c \oplus d \\ &= 2(b \oplus c) \oplus c \oplus (d \oplus a) \\ &= g_1 \oplus c \oplus f_3 \end{aligned} \quad (3-22)$$



$$\begin{aligned} Y &= a \oplus b \oplus 2c \oplus 3d \\ &= 2(c \oplus d) \oplus d \oplus (a \oplus b) \\ &= g_2 \oplus d \oplus f_0 \end{aligned} \quad (3-23)$$

$$\begin{aligned} Z &= 3a \oplus b \oplus c \oplus 2d \\ &= 2(d \oplus a) \oplus a \oplus (b \oplus c) \\ &= g_3 \oplus a \oplus f_1 \end{aligned} \quad (3-24)$$

其中

$$f_0 = a \oplus b \quad g_0 = 2(a \oplus b) = \text{xtime}(f_0)$$

$$f_1 = b \oplus c \quad g_1 = 2(b \oplus c) = \text{xtime}(f_1)$$

$$f_2 = c \oplus d \quad g_2 = 2(c \oplus d) = \text{xtime}(f_2)$$

$$f_3 = d \oplus a \quad g_3 = 2(d \oplus a) = \text{xtime}(f_3)$$



對於 InvMixColumns 而言，其運算式如下：

$$\begin{bmatrix} S \\ T \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \quad (3-25)$$

$$\begin{aligned} S &= 14a \oplus 11b \oplus 13c \oplus 9d \\ &= 4(2(a \oplus b) \oplus 2(c \oplus d) \oplus (a \oplus c)) \oplus 2(a \oplus b) \oplus b \oplus (c \oplus d) \\ &= i_0 \oplus W \end{aligned} \quad (3-26)$$

$$\begin{aligned} T &= 9a \oplus 14b \oplus 11c \oplus 13d \\ &= 4(2(b \oplus c) \oplus 2(d \oplus a) \oplus (b \oplus d)) \oplus 2(b \oplus c) \oplus c \oplus (d \oplus a) \\ &= i_1 \oplus X \end{aligned} \quad (3-27)$$

$$\begin{aligned} U &= 13a \oplus 9b \oplus 14c \oplus 11d \\ &= 4(2(c \oplus d) \oplus 2(a \oplus b) \oplus (c \oplus a)) \oplus 2(c \oplus d) \oplus d \oplus (a \oplus b) \\ &= i_0 \oplus Y \end{aligned} \quad (3-28)$$

$$\begin{aligned} V &= 11a \oplus 13b \oplus 9c \oplus 14d \\ &= 4(2(d \oplus a) \oplus 2(b \oplus c) \oplus (d \oplus b)) \oplus 2(d \oplus a) \oplus a \oplus (b \oplus c) \\ &= i_1 \oplus Z \end{aligned} \quad (3-29)$$

其中

$$i_0 = 4h_0 = \text{xtime}(\text{xtime}(h_0)) \quad h_0 = g_0 \oplus g_2 \oplus a \oplus c$$

$$i_1 = 4h_1 = \text{xtime}(\text{xtime}(h_1)) \quad h_1 = g_1 \oplus g_3 \oplus b \oplus d$$

由(3-26)到(3-29)可知，InvMixColumns 的運算結果完成使用到 MixColumns 的運算

結果，並且(3-28)及(3-29)又共用了(3-26)及(3-27)運算結果中的  $i_0$  及  $i_1$ ，此為最佳化的運算結果。此合併的 4 位元組 MixColumns/InvMixColumn 的架構圖如圖 3-4。

由圖 3-4 可以看出來，整合後的 4 位元組 MixColumns/InvMixColumns 模組，總共使用了 8 個 xtime 運算及 22 個 XOR 運算，與 MC\_InvMC\_3 作一比較，如表(3-2)

Operations	MC_InvMC_3	MC_InvMC_Ours	Improvement
xtime	16	8	50%
XOR	32	22	31%

表 3-2 MC\_InvMC\_3 與 MC\_InvMC\_Ours 運算次數之比較

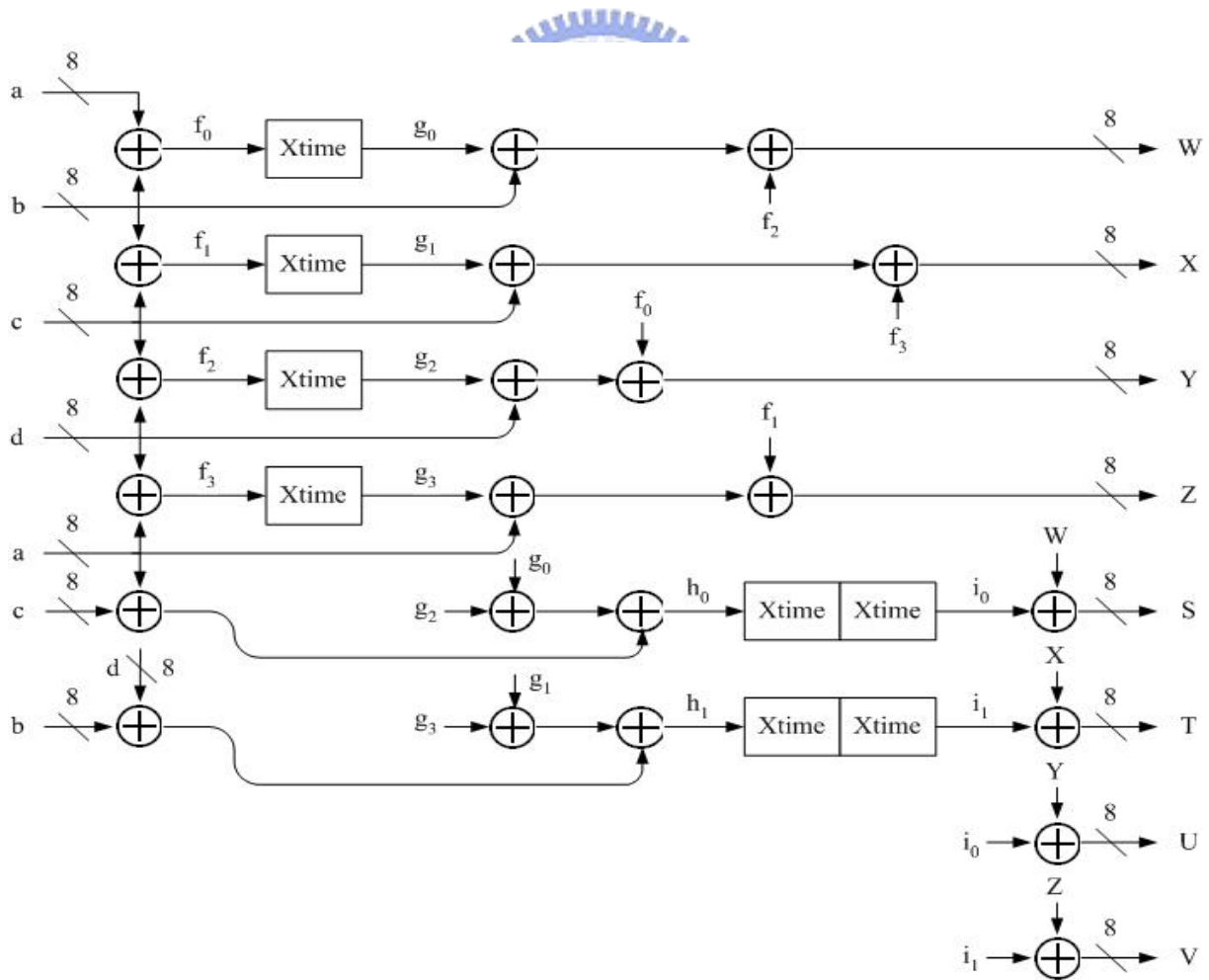


圖 3-4 整合4位元組的 MixColumns 和 InvMixColumns 硬體架構圖

因此，使用 4 個位元組來整合其運算式相同的的部分的方式，將有效降低硬體的複雜度及硬體的成木。對於採用平行架構，只要複製 4 個此模組，如圖 3-5，共使用 32 個 xtime 及 88 個 XOR 運算。

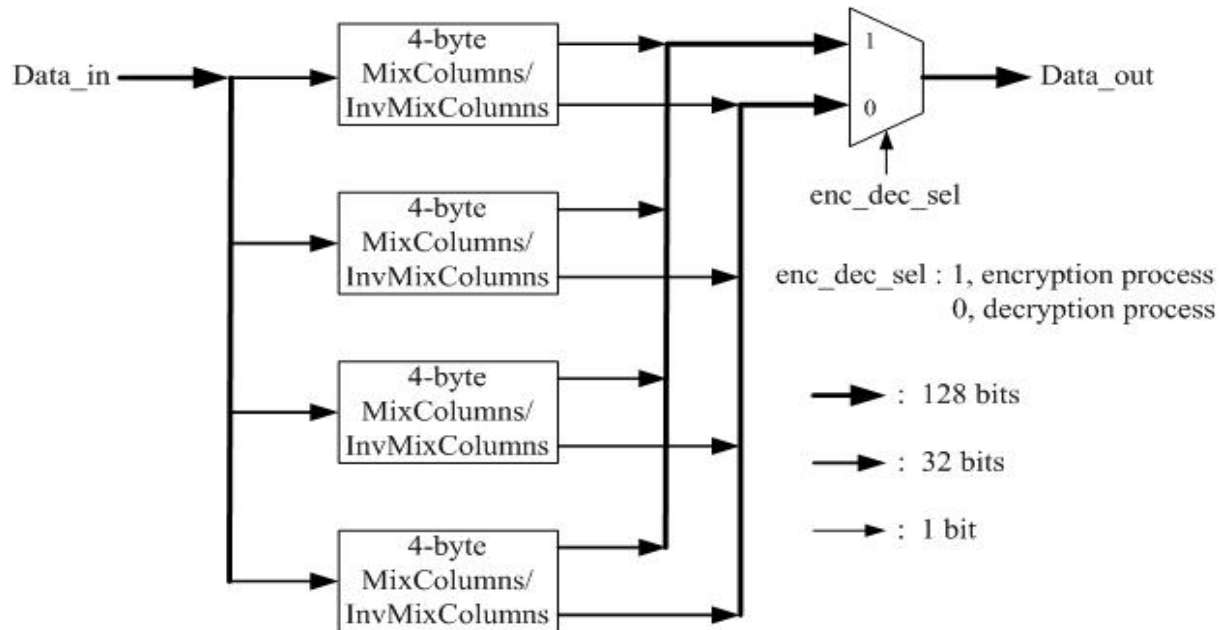


圖 3-5 整合16位元組的 MixColumns 和 InvMixColumns 硬體方塊圖

### 3.4 密鑰擴充模組的架構

Key Expansion 主要由三種運算所組成，以下敘述如何以硬體電路來實現此三種運算：

**Subword**：由於此運算為輸入一個 4-byte 的字組，經 S-box 查表轉換之後，產生一個 4-byte 的字組，所以對於平行架構而言，需要複製 4 個整合的 S-box/Inv. S-box 所描述的硬體架構，並且使用 S-box 查表，如圖 3-6。

**RotWord**：如同 ShiftRows 的硬體電路設計方式，只需要透過硬體接線即可達到位元組位移的功能。如圖 3-7。

**Rcon**：兩種方式可以實現 Rcon 的運算。

第一種方式是用邏輯函數作 xtime 運算，得出每個回合在 Rcon 內中 B0 的數值，例如：第一回合 Rcon 的 B0 為(01)<sub>16</sub>，第二回合 Rcon 的 B0 為(02)<sub>16</sub>，其餘的每個回合類推。其架構如圖 3-8。

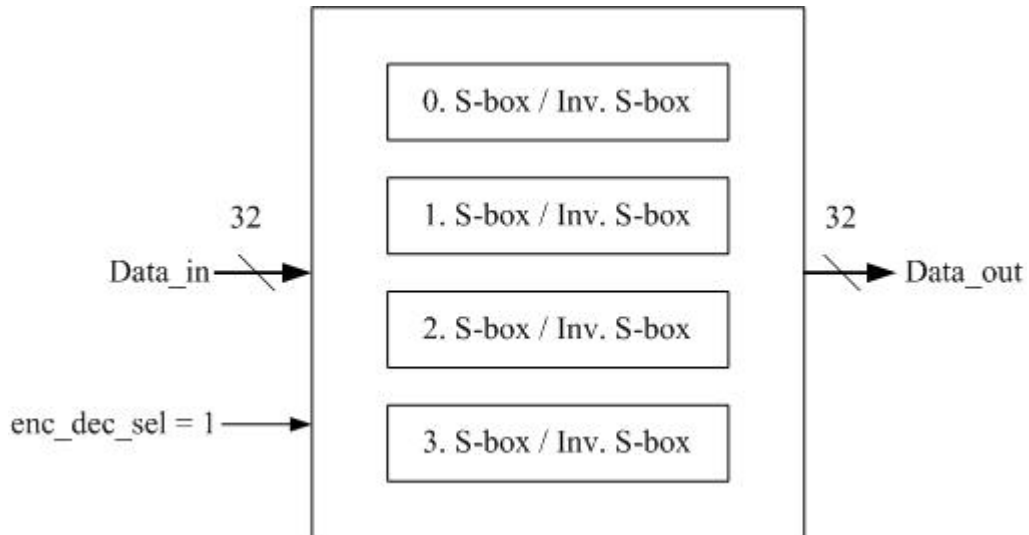


圖 3-6 SubWord 運算之硬體方塊圖

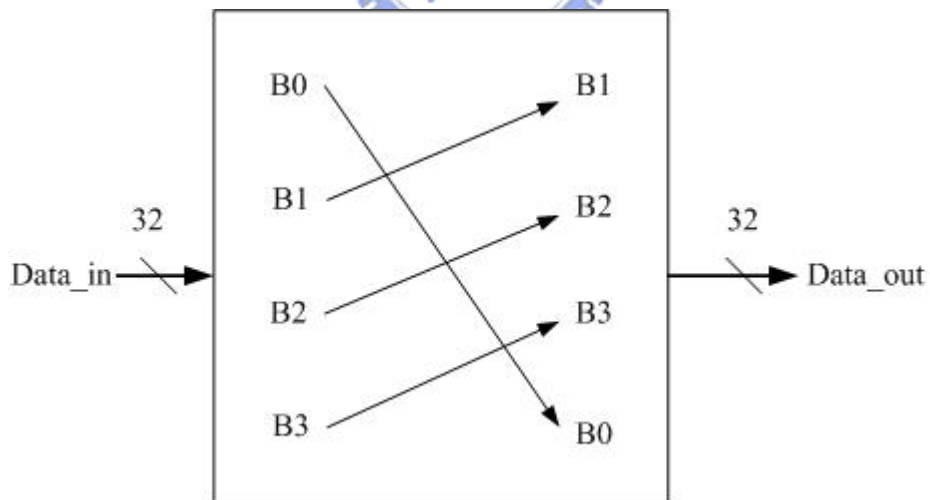


圖 3-7 RotWord 運算之硬體方塊圖

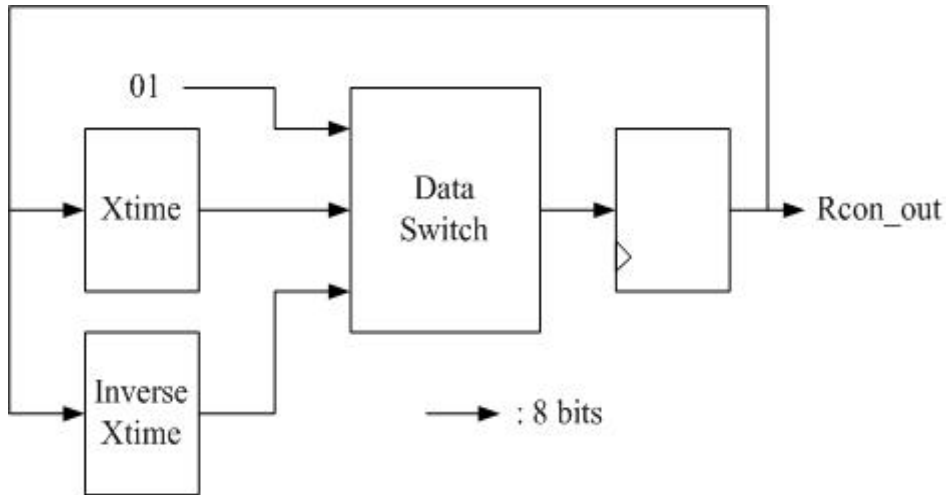


圖 3-8 用邏輯函數實現Rcon的方塊圖

圖 3-8 中，01 設定為初始的 Rcon 輸出值，xtime 的運算如(3-4)，Inverse xtime 的運算如(3-30)。

$$\begin{aligned}
 (\{02\} \cdot B(x))^{-1} &= (b_0 \oplus b_1) + b_2 x + (b_0 \oplus b_3)x^2 + (b_0 \oplus b_4)x^3 \\
 &\quad + b_5 x^4 + b_6 x^5 + b_7 x^6 + b_0 x^7
 \end{aligned}
 \tag{3-30}$$

第二種方式是經由查表方式，將每個回合的 Rcon 的 B0 做輸出的動作，其中 B1，B2 和 B3 因為皆為 0，0 XOR 一個數仍然等於那個數，所以不需要將它們建入表格，增加硬體面積，其架構如圖 3-9。對於 AES-128 而言，此表格只有 20 位元組，包括加密及解密過程中 Rcon 的輸出值，如表 3-3 所示。

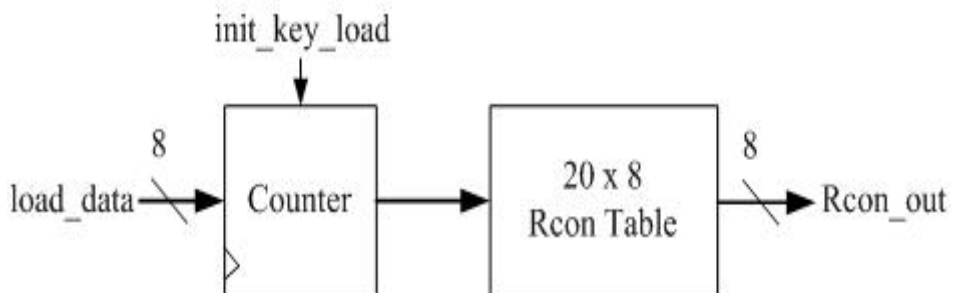


圖 3-9 用查表方式實現Rcon的方塊圖

比較兩種方式的硬體閘數(gate count)，實現第一種方式所需要用到的 gate count 為 143，實現第二種方式所需要用到的 gate count 為 89，所以採用 gate count 較小的查表方式來實現 Rcon 運算。

Round Number	Encryption process Rcon B0 output	Decryption process Rcon B0 output
1	(01) <sub>16</sub>	(36) <sub>16</sub>
2	(02) <sub>16</sub>	(1B) <sub>16</sub>
3	(04) <sub>16</sub>	(80) <sub>16</sub>
4	(08) <sub>16</sub>	(40) <sub>16</sub>
5	(10) <sub>16</sub>	(20) <sub>16</sub>
6	(20) <sub>16</sub>	(10) <sub>16</sub>
7	(40) <sub>16</sub>	(08) <sub>16</sub>
8	(80) <sub>16</sub>	(04) <sub>16</sub>
9	(1B) <sub>16</sub>	(02) <sub>16</sub>
10	(36) <sub>16</sub>	(01) <sub>16</sub>

表 3-3 加密與解密過程每個回合 Rcon 的輸出值

有兩種方法可以用來實現 Key Expansion 硬體模組：

第一種方法是先算出所有回合的密鑰，然後將這些密鑰放在記憶體內，等到要執行 AES 的加密或解密的流程時，再依序輸出個回合的密鑰[4]。此方法適用於不會經常更改初始密鑰的應用。其優點是在加密及解密過程中，可隨時輸出在記憶體內每個回合的密鑰，不需要先執行過加密過程中 Nr 個回合的 Key Expansion 運算；缺點是硬體需要額外提供一塊面積給記憶體使用，對於 AES-128 而言，需要 176\*8 的記憶體。

第二種方法是，即時計算出該回合的密鑰提供給 AddRoundKey 運算使用[7]，所以它不需要使用到記憶體來存放這些密鑰，此方法適用於經常需要變動初始密鑰的應用，其優點是硬體可以省下一塊記憶體面積以及記憶體控制電路。缺點是解密過程輸入的初始密鑰為加密過程最後回合的密鑰，因此必須先計算出加密過程最後回合的密鑰。為了降低硬體使用上的資源，本論文採用第二種方法來實現 AES-128 的 Key Expansion，其硬體架構如圖 3-10。

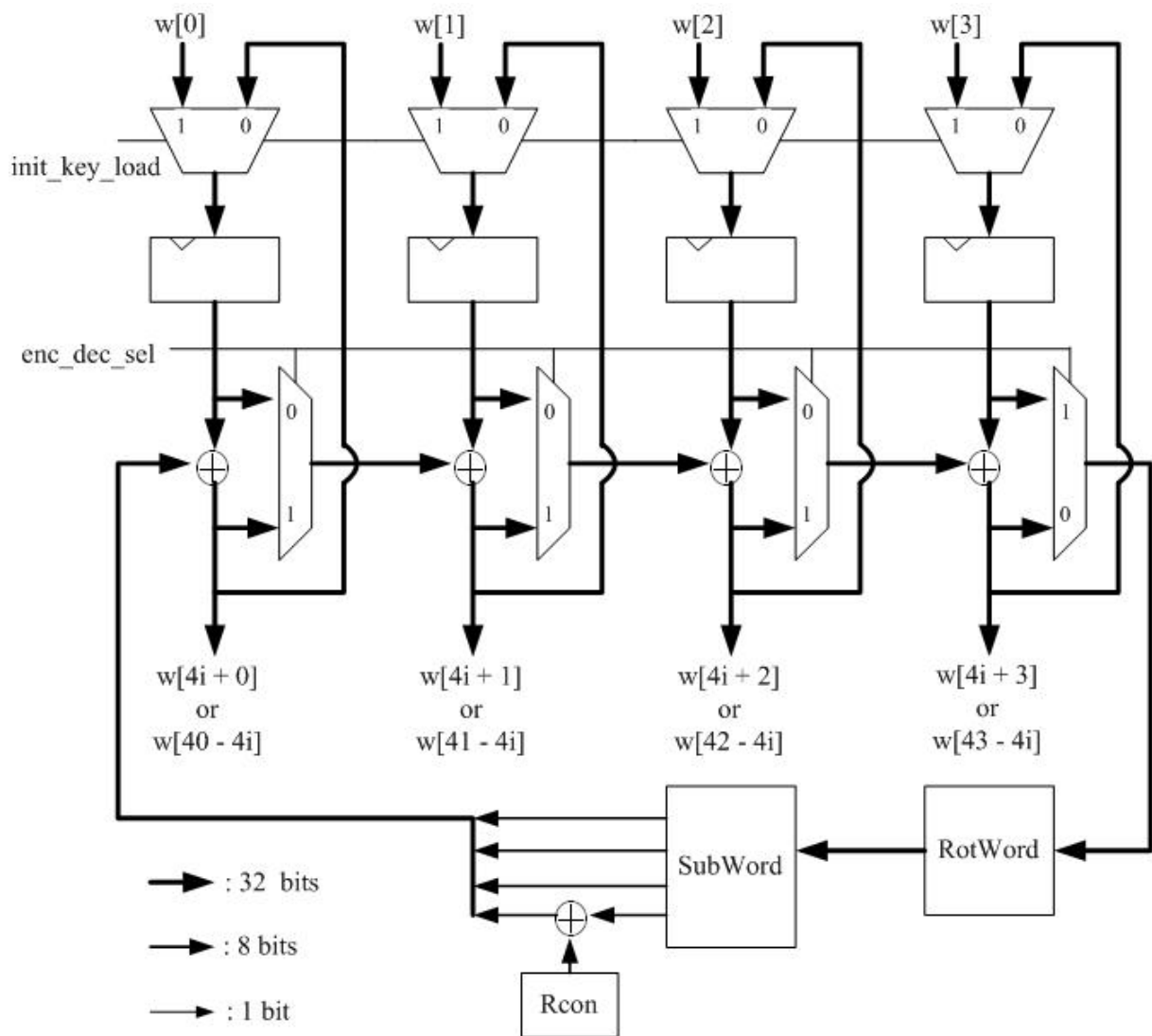


圖 3-10 AES-128 Key Expansion 模組硬體架構圖

在圖 3-10 中的  $\text{init\_key\_load} = 1$  時，為輸入初始回合的密鑰  $w[0] \sim w[3]$ ，之後  $\text{init\_key\_load} = 0$ ，每個時脈產生每個回合的密鑰輸出，對加密程序而言， $\text{enc\_dec\_sel} = 1$ ，輸出  $w[4i + 0]$  到  $w[4i + 3]$ ，對解密程序而言， $\text{enc\_dec\_sel} = 0$ ，輸出  $w[40 - 4i]$  到  $w[43 - 4i]$ ，其中  $i$  為回合數 0 到 10。若解密方式採用 Equivalent Inverse Cipher 則需要增加四個 4-Byte MixColumns/InvMixColumns 模組在 Key Expansion 的每個輸出上，如圖 3-11。

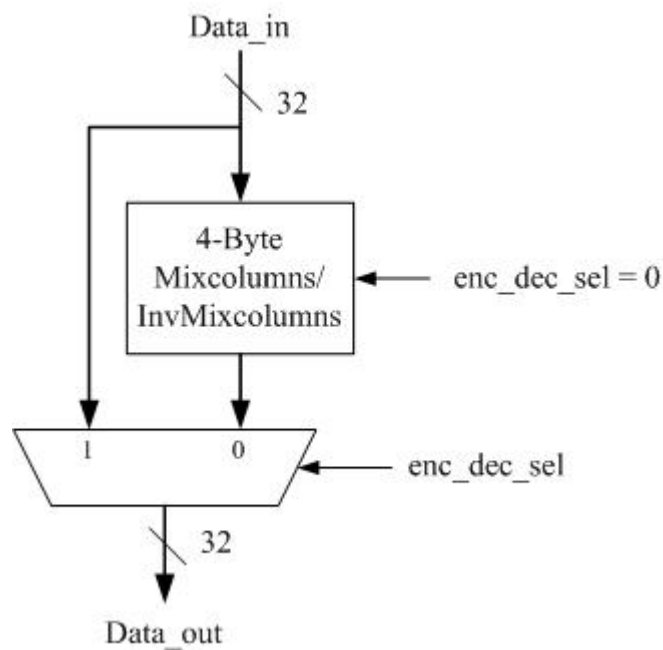


圖 3-11 使用 Equivalent Inverse Cipher 時，Key Expansion 模組每個輸出端的架構圖



## 第四章 電路設計的結果與分析

根據 2.5 節所描述，AES 的解密程序分為兩種，第一種方式是解密程序在 1~Nr 回合與加密程序不同，AES 稱此解密程序為 Inverse Cipher。第二種方式是解密程序在 1~Nr 回合與加密程序相同，AES 稱此解密程序為 Equivalent Inverse Cipher。採用前者，在硬體架構上需要多工器去切換不同的加密及解密程序。採用後者在 Key Expansion 的輸出端需要增加四個 4 位元組的 MixColumns/InvMixColumns 硬體模組。

本論文實現 AES-128 硬體架構分為 128 位元的資料路徑以及 32 位元的資料路徑，以適應不同的應用需求。為了使加密程序與解密程序一致，此兩種不同的資料路徑的硬體架構在解密程序皆使用 Equivalent Inverse Cipher 的方式，以降低硬體設計的複雜度。

### 4.1 整合後之 AES 硬體架構



由於整合的 S-box/InvS-box 及 MixColumns/InvMixColumns 運算較為複雜，因此，此兩種硬體模組所佔整個 AES-128 硬體面積比重較大。對於資料路徑為 128 位元及 32 位元的 AES-128 硬體架構，其 AES-128 加解密主流程的部分以及 Key Expansion 流程的部分，其所使用到此兩個硬體模組數目分別整理如表 4-1 及表 4-2

Module	AES core	Key Expansion core	Total
S-box/Inv S-box	16	4	20
4-byte MixColumns/ InvMixColumns	4	4	8

表 4-1 資料路徑為 128 位元的 AES-128 硬體架構所使用到的硬體模組數目

Module	AES core	Key Expansion core	Total
S-box/Inv S-box	4	4	8
4-byte MixColumns/ InvMixColumns	1	1	2

表 4-2 資料路徑為 32 位元的 AES-128 硬體架構所使用到的硬體模組數目

#### 4.1.1 資料路徑為 128 位元之架構

資料路徑為 128 位元的 AES-128 硬體架構圖，如圖 4-1。每一回合的運算使用了一個時脈。它整合了 2.5 節的圖 2-11 和 2-13 所描述的加密和等效的解密程序。圖 2-11 和 2-13 中各使用了三個 AddRoundKey，但是在圖 4-1 只使用 1 個 AddRoundKey 的硬體模組，以節省硬體資源。MUX0 用來切換初始回合輸入的資料或是第一回合到第十回合的輸入資料。其中第十回合也稱為 AES-128 的最終回合；MUX1 用來切換第一回合到第九回合輸入的資料或是第十回合輸入的資料，這也是為什麼此硬體架構只使用了 1 個 AddRoundKey 硬體模組的原因。

由於此硬體架構的解密方式採用 Equivalent Inverse Cipher，所以輸入給 AddRoundKey 的資料為 Key Expansion 產生的每個回合的密鑰再經過 InvMixColumns 運算後的輸出的資料。

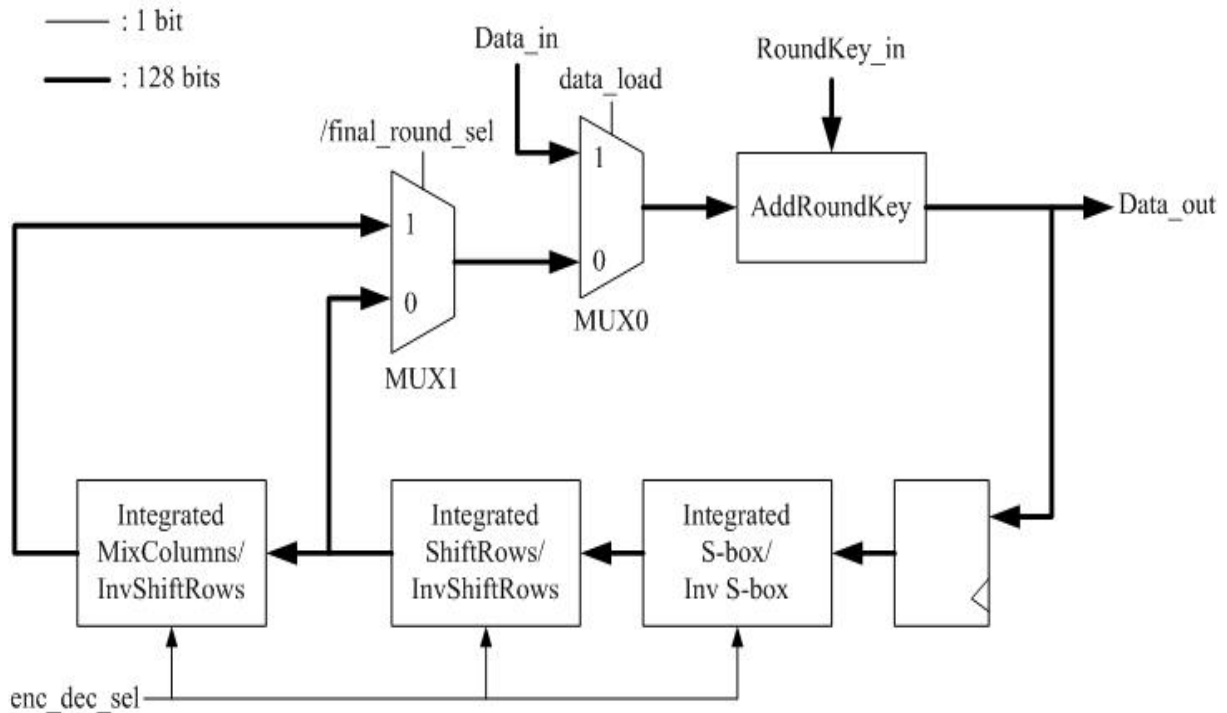


圖 4-1 資料路徑為128位元的AES-128硬體架構圖

#### 4.1.2 資料路徑為 32 位元之架構

資料路徑為 32 位元的 AES-128 硬體架構圖，如圖 4-2。每一回合的資料運算需要使用 4 個時脈，換言之，128 位元的初始資料和 128 位元的初始密鑰要分四次作輸入。它整合了 2.5 節的圖 2-11 和 2-13。同樣的，只有 1 個 AddRoundKey 的硬體模組在架構圖中，以節省硬體資源。

由於資料輸入或輸出順序，對狀態陣列而言屬於每行資料依序輸入或輸出，因此 S-box/InvS-box 和 MixColumns/InvMixColumns 硬體模組屬於行運算，然而 ShiftRows/InvShiftRows 卻屬於列運算，所以需要等到狀態陣列中 16 個位元組的資料經過 S-box 或 Inverse S-box 的完全運算完後， ShiftRows/InvShiftRows 列運算才能完全正確的輸出 16 位元組的資料，之後此 16 位元組的資料每次再分成 4 位元組傳送給

MixColumns/InvMixColumns 硬體模組做運算。圖 4-2 中，MUX0 用來切換初始回合輸入的資料或是第一回合到第十回合的輸入資料，其中第十回合 AES-128 也稱為最終回合；MUX1 用來切換第 1 回合到第九回合輸入的資料或是第十回合輸入的資料，這也是為什麼此架構只使用了 1 個 AddRoundKey 硬體模組的原因。另外，由於 AES-128 的狀態陣列共有四行，所以在資料路徑為 32 位元的 AES-128 硬體架構中多用了一組 2 位元的上數計數器(upper counter)，用來指向目前的運算是在狀態陣列中的哪一行，若 column\_index 等於 0，則指向狀態陣列的第 0 行，並且開始做運算，同理，對於 column\_index 等於 1、2 和 3，則指向狀態陣列的第 1、2 和 3 行，並且開始做運算。column\_index 為狀態陣列中行的運算指標。

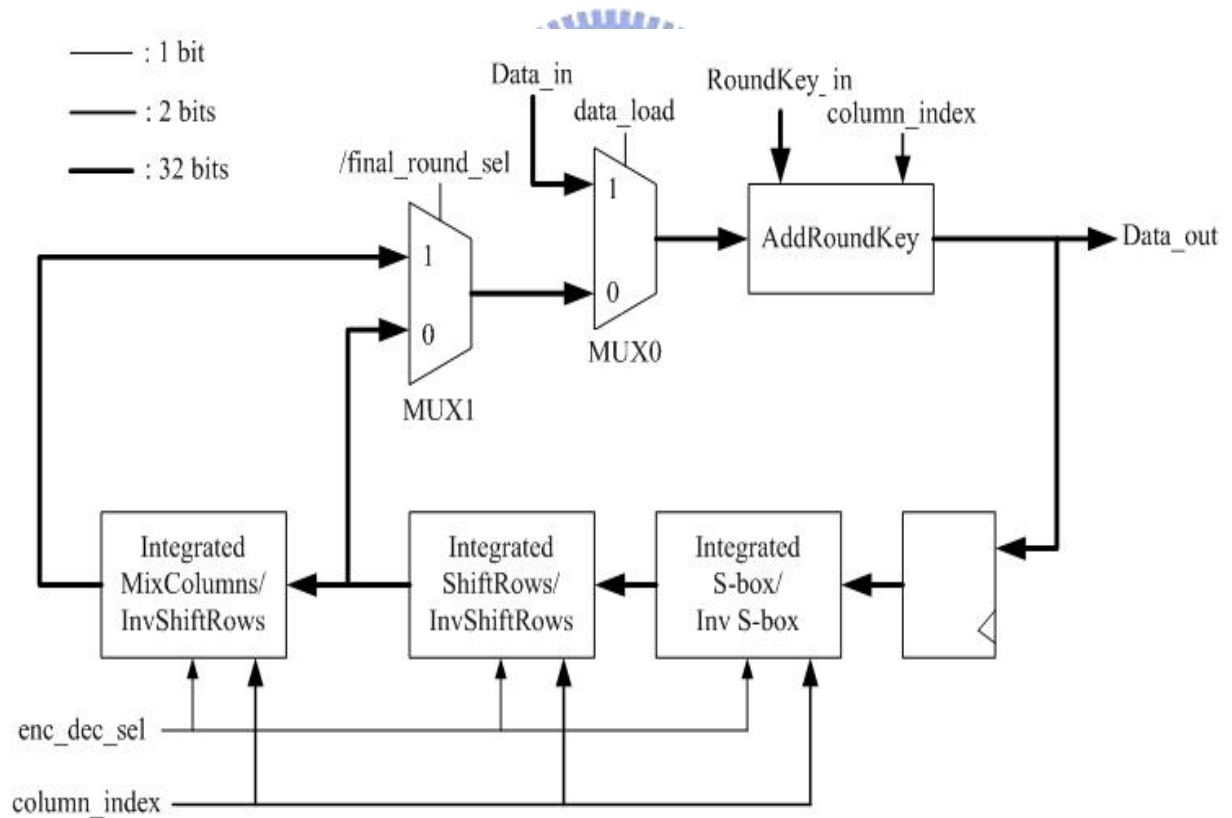
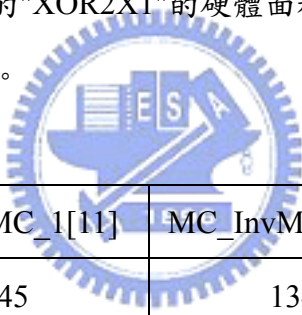


圖 4-2 資料路徑為32位元的AES-128硬體架構圖

## 4.2 設計結果的比較與分析

3.3 節提到有四種方式可以硬體實現 MixColumns 及 InvMixColumns 的整合，分別是 MC\_InvMc\_1, MC\_InvMC\_2, MC\_InvMC\_3 以及 MC\_InvMC\_Ours。其中 MC\_InvMc\_2 其 xtime 和 XOR 的運算量遠高於 MC\_InvMC\_3，因此不列入整合的 MixColumns 及 InvMixColumns 硬體模組比較的範圍，因此，對於平行架構而言，我們將實現 MC\_InvMc1，MC\_InvMc3 及 MC\_InvMc\_Ours 的方式來整合 MixColumns 及 InvMixColumns 硬體電路，並且採 TSMC 0.18um 標準元件庫來做合成，由於 MixColumns 及 InvMixColumns 的演算法大部分屬於 XOR 運算，因此我們將合成出來的硬體面積除以在 TSMC 0.18um 標準元件庫的 "XOR2X1" 的硬體面積  $26.6\mu\text{m}^2$ ，而得出硬體的閘數。此三種電路的硬體閘數，如表 4-3。



	MC_InvMC_1[11]	MC_InvMC_2[7,12]	MC_InvMC_Ours
Gate count	2145	1344	824

表 4-3 三種整合 MixColumns 及 InvMixColumns 的硬體電路閘數比較表

由表 4-3 可知，本論文提出的整合 MixColumns 及 InvMixColumns 的方式，實現出來的硬體成本，與 MC\_InvMC\_1 及 MC\_InvMC\_3 比較起來各少了 160.3% 及 63.1%，因此，有效地節省了硬體資源。

本論文所實現出來兩種不同資料路徑的 AES-128 硬體電路，採用 Verilog HDL 來描述硬體，並且使用 TSMC 0.18um CMOS 標準元件庫在 TYPICAL case 來做合成，對於硬體總閘數為合成後的總硬體面積除以 TSMC 0.18um 標準元件庫的 "NAND2X1" 的硬體面積  $9.98\mu\text{m}^2$ 。

對於資料路徑為 128 位元的硬體電路，其 critical path 為 4.853 ns，所以我們將它操作在 200 MHz，得出硬體總閘數為 29663，加密程序使用了 11 個時脈，初始回合使用 1 個時脈，第一到第十個回合每個回合使用 1 個時脈；解密程序使用 21 個時脈，其中前 11 個時脈輸入資料和初始密鑰以及用來產生解密程序每回合的密鑰，之後解密程序的 10 個回合每個回合使用 1 個時脈。加密程序的資料總輸出量(throughput) 為 2.327 Gbits/sec，解密程序的資料總輸出量為 1.219 Gbits/sec。平均的資料總輸出量為 1.773 Gbits/sec。

對於資料路徑為 32 位元的 AES-128 硬體電路，其 critical path 為 3.633 ns，我們將它操作在 270 MHz，得出硬體總閘數約為 12826，加密程序使用 44 個時脈，解密程序使用 84 個時脈，因此加密程序的資料總輸出量為 785 Mbits/sec，解密程序的資料總輸出量為 411 Mbits/sec。平均的資料總輸出量為 598 Mbits/sec。

對於本論文所實現資料路徑為 128 位元和 32 位元兩種硬體電路與其他不同 AES 的硬體設計之比較如表 4-4。

	NSYSU[4]	ITRI[7]	Kuo[8]	Ours, DP:128	Ours, DP:32
Process(um)	0.35	0.25	0.18	0.18	0.18
Frequency(MHz)	70	100	154	200	270
Throughput(Mbps)	298	609	2290	1773	598
Gate count(K)	74.3	31.9	173	29.6	12.8
Throughput / Gate count (Mbps/K)	4	19.1	13.2	39.1	46.7

表 4-4 不同 AES 的硬體設計之比較

由表 4-4 可知，對於資料輸出量而言，Kuo[8]的 AES 設計有最佳的數據，且高於本論文的 128 位元資料路徑的硬體設計約 1.3 倍，然而它的硬體成本卻是將近本論文的設計的六倍。對於硬體總閘數，我們的 32 位元資料路徑的硬體設計，有最少的硬體成本，同時也有最佳的效能成本比。





## 第五章 結論與未來展望

### 5.1 結論

本篇論文所提出實現整合的 MixColumns/InvMixColumns 演算法的硬體架構，已有效地降低了 160.3%[11]及 63.1% [7,12]此演算法硬體設計的複雜度，並進而提昇整體 AES-128 的性能。對於資料路徑為 128 位元的硬體電路，整個電路的硬體成本為 29663 個閘數，產生的平均資料輸出量為 1.773Gbits/sec，適用在高速傳輸的應用。對於資料路徑為 32 位元的硬體電路，整個電路的硬體成本為 12826 個閘數，產生的平均資料輸出量為 598Mbits/sec，適用在較低硬體成本。

### 5.2 未來展望



在 AES 標準中定義了其資料區塊長度為固定 128 位元，但是密鑰長度可以是 128，192 以及 256 位元。由於本論文設計的硬體電路其資料區塊及密鑰長度皆為 128 位元，因此未來的工作希望將此 AES 硬體電路能將密鑰的長度為 192 及 256 位元的功能一併整合，換句話說，具有 AES-128，AES-192 及 AES-256 三種加密層次，並且要達到此項功能的整合其所需要更動的主要硬體模組為 Key Expansion 以及增加一些密鑰選擇的控制電路。



## 參 考 文 獻

- [1] “Advanced Encryption Standard (AES)”, Federal Information Processing Standards Publication 197, November 26, 2001.
- [2] J. Daemen and V. Rijmen, AES Proposal: Rijndael, Document Version 2, March, 9, 1999.
- [3] National Institute of Standards and Technology (NIST), Data Encryption Standard (DES), FIPS PUB 46-3, October 25. 1999.
- [4] K. H. Chung, “The design and implementation of security and networking co-processors for high performance SoC applications”, Master Thesis, Dept. of Computer Science, National Sun Yat-sen University, Kaohsiung, Taiwan, 2002.
- [5] A. J. Elbirt, et al. “An FPGA-based performance evaluation of the AES block cipher candidate algorithm finalists”, IEEE Trans. on VLSI Systems, 9.4, August 2001, pp. 545-557.
- [6] Alireza Hodjat, Ingrid Verbauwhede, “A 21.54 Gbits/s Fully Pipelined AES Processor on FPGA”, IEEE Symposium on Field-Programmable Custom Computing Machines, April 2004.
- [7] C. C. Lu and S. Y. Tseng, “Integration of AES (Advanced Encryption Standard) encrypter and decrypter”, Proceeding, Application-Specific Systems, Architecture and Processor, 2002, pp. 277-285.
- [8] I. Verbauwhede and Henry Kuo, “Design and Performance Testing of a 2.29Gbits/sec Rijndael Processor”, IEEE Journal of solid state circuits, vol. 38, No. 3, March 2003, pp. 569-572.
- [9] Richard E. Blahut, “Algebraic Codes for Data Transmission”, Cambridge University Press, New York, USA, 2003.

- [10] S. Chantarawong, P. Noo-intara, and S.Choomchuay, “An Arhitecture for S-Box Computation in the AES”, ReCCIT, Phuket, Thailand, 2004.
- [11] P. Noo-intara, S. Chantarawong, and S.Choomchuay, “Arhitectures for MixColumn Transform for the AES”, ReCCIT, Phuket, Thailand, 2004.
- [12] Hsin-Chung Wang, Chih-Hsiu Lin, and A.-Y. Wu, “Design and Implementation of a Cost-efficient AES Cryptographic Engine”, 台大工程學刊, No. 88, June 2003, pp. 51-60.



# 附 錄 一

GF(2<sup>8</sup>) 乘法反元素對照表

0	0	20	3a	40	1d	60	16	80	83	a0	fb	c0	b	e0	b1
1	1	21	6e	41	fe	61	5e	81	7e	a1	7c	c1	28	e1	d
2	8d	22	5a	42	37	62	af	82	7f	a2	2e	c2	2f	e2	d6
3	f6	23	f1	43	67	63	d3	83	80	a3	c3	c3	a3	e3	eb
4	cb	24	55	44	2d	64	49	84	96	a4	8f	c4	da	e4	c6
5	52	25	4d	45	31	65	a6	85	73	a5	b8	c5	d4	e5	e
6	7b	26	a8	46	f5	66	36	86	be	a6	65	c6	e4	e6	cf
7	d1	27	c9	47	69	67	43	87	56	a7	48	c7	f	e7	ad
8	e8	28	c1	48	a7	68	f4	88	9b	a8	26	c8	a9	e8	8
9	4f	29	a	49	64	69	47	89	9e	a9	c8	c9	27	e9	4e
a	29	2a	98	4a	ab	6a	91	8a	95	aa	12	ca	53	ea	d7
b	c0	2b	15	4b	13	6b	df	8b	d9	ab	4a	cb	4	eb	e3
c	b0	2c	30	4c	54	6c	33	8c	f7	ac	ce	cc	1b	ec	5d
d	e1	2d	44	4d	25	6d	93	8d	2	ad	e7	cd	fc	ed	50
e	e5	2e	a2	4e	e9	6e	21	8e	b9	ae	d2	ce	ac	ee	1e
f	c7	2f	c2	4f	9	6f	3b	8f	a4	af	62	cf	e6	ef	b3
10	74	30	2c	50	ed	70	79	90	de	b0	c	d0	7a	f0	5b
11	b4	31	45	51	5c	71	b7	91	6a	b1	e0	d1	7	f1	23
12	aa	32	92	52	5	72	97	92	32	b2	1f	d2	ae	f2	38
13	4b	33	6c	53	ca	73	85	93	6d	b3	ef	d3	63	f3	34
14	99	34	f3	54	4c	74	10	94	d8	b4	11	d4	c5	f4	68
15	2b	35	39	55	24	75	b5	95	8a	b5	75	d5	db	f5	46
16	60	36	66	56	87	76	ba	96	84	b6	78	d6	e2	f6	3
17	5f	37	42	57	bf	77	3c	97	72	b7	71	d7	ea	f7	8c
18	58	38	f2	58	18	78	b6	98	2a	b8	a5	d8	94	f8	dd
19	3f	39	35	59	3e	79	70	99	14	b9	8e	d9	8b	f9	9c
1a	fd	3a	20	5a	22	7a	d0	9a	9f	ba	76	da	c4	fa	7d
1b	cc	3b	6f	5b	f0	7b	6	9b	88	bb	3d	db	d5	fb	a0
1c	ff	3c	77	5c	51	7c	a1	9c	f9	bc	bd	dc	9d	fc	cd
1d	40	3d	bb	5d	ec	7d	fa	9d	dc	bd	bc	dd	f8	fd	1a
1e	ee	3e	59	5e	61	7e	81	9e	89	be	86	de	90	fe	41
1f	b2	3f	19	5f	17	7f	82	9f	9a	bf	57	df	6b	ff	1c

# 自 傳

本論文作者施忠宏在 1971 出生於台北，大學於 1996 年畢業於國立台灣工業技術學院電子工程學系，目前正在台灣新竹的國立交通大學電機資訊學院電信組攻讀碩士學位，研究領域為系統單晶片設計。

