國立臺灣師範大學工業教育學系

碩士論文

指導教授：黃奇武 副教授

張吉正 教授

8 位元進階加密器 FPGA 設計

8-bit AES FPGA Design

研究生：戴宏運 撰

中華民國九十六年七月

謝誌

感謝指導教授張吉正老師、黃奇武老師，在我碩班兩年來的求學生涯中，扮演著亦師亦友的角色。在遇到挫折、困難的時候，老師們則像是一盞明燈，指引著前進的方向；在求學過程中，老師們更是一起討論的好對象，協助學生披荊斬棘，達到目標，最終完成學業。再次謝謝老師們的教導，不只在學業上，更使學生在人生目標及生活態度上，有了正確的觀念，讓學生滿載而歸，受益匪淺。

再來要感謝炘彥學長、凱婷學姊，登貴學長、文治學長在我還是新生的時候多方面的提攜與幫助。還要感謝實驗室的所有夥伴，茂元、東毅、國煌，因為你們的幫助，使的研究室的研究氣氛，充滿合諧融洽，擁有樂觀進取、面對挑戰的勇氣，豐富了兩年的碩士生活，同時也希望學弟們能一同營造、維持下去，LAB514 YES。

最後要感謝我的父親戴榮興先生及母親陳美花女士，以及女友鈺茹小姐和所有關心我的、數不盡的、感激不完的……所有所有，在我求學和人生的路途上給我勉勵、給我打氣。

Abstract

On 2000, the National Institute of Standards and Technology (NIST) announced that the Rijndael encryption algorithm was chosen as the Advanced Encryption Standard (AES), which would be the next generation of encryption standard to replace the Data Encryption Standard (DES), and became the federal information encryption standard the next year.

In our research, which is differ from other AES algorithm in data-path width of 128 bit or 32 bit that would probably pipelined to achieve high throughput like tens Giga Bit Per Second (GBPS). In fact, an 8 bit width data-path AES algorithm should be enough in some consuming electronic applications such as Radio Frequency Identification (RFID) which needs only a slower data transfer rate.

In this thesis, we implemented an 8 bit AES circuits on Field Programmable Gate Array (FPGA) and expected that it could be used in many different applications by its advantages of small area and more high through. More, the AES circuits were written in VHDL code by the designing tool of Xilinx ISE and verified and simulated by ModelSim. Moreover, by the way of using Block RAMs could reduce area (here is Slices utilization) effectively and provide a good throughput.

Keywords：AES, DES, FPGA

中文摘要

2000年10月美國政府機構NIST正式宣布選用Rijndael演算法作為AES、且於2001年成為美國聯邦資訊處理加密標準，逐步取代Data Encryption Standard（DES）成為新一代的加密標準。

本研究有別於128-bit、32-bit AES之資料路徑（Datapath），使用管線結構（Pipeline），可以達到每秒數十億位元（GBPS）之高產量（throughput）。在一些消費性電子如行動通訊、RFID上並不需要較大的資料傳輸速率，因此8-bit之資料路徑是個不錯的選擇。

在本論文中，使用FPGA來實現8-bit AES之硬體電路，以達到小面積及較高產率（throughput）之優點，以利於不同應用上。

本研究利用VHDL、Xilinx ISE 7.1、ModelSim來驗證與模擬。且使用不同硬體架構來實現並加以比較。其中使用Block RAM可以有效節省面積（本論文中指Slice之使用量）且可以提供不錯的產率（throughput）。

關鍵字：AES、DES、FPGA

總目錄

謝誌 I

Abstract III

中文摘要 IV

總目錄 V

表目錄 VII

圖目錄 VIII

第一章　緒論 1

　第一節　研究背景 1

　第二節　研究動機 2

　第三節　研究目的 3

　第四節　研究步驟 4

第二章　AES 介紹－Rijndael 演算法 5

　第一節　Rijndael 加解密之架構 6

　第二節　數學背景 13

　第三節　位元組取代轉換（SubByte / InvSubByte） 17

　第四節　移列轉換(ShiftRow/InvShiftRow) 18

　第五節　混行轉換(MixColumn/ InvMixColumn) 20

　第六節　金鑰加法運算(AddRoundkey) 21

　第七節　金鑰擴展(Key expansion) 22

第三章　相關研究 24

第四章　8-bit AES 架構設計 30

　第一節　8-bitAES 位元組取代轉換（SubByte / InvSubByte） 30

　第二節　8-bit AES 移列轉換(ShiftRow/InvShiftRow) 31

第三節　8-bit AES 混行轉換(MixColumn/ InvMixColumn).....33
第四節　8-bit AES 金鑰擴展(Key expansion).....37
第五節　8-bit AES 加密架構 40
第五章　研究成果 43
第一節　金鑰擴展設計與測試結果 43
第二節　混行運算設計與測試結果 47
第三節　8-bit AES 加密器設計與測試結果.....48
第六章 結論及未來工作 52
參考文獻.....53

表目錄

表 2-2　S-box 替換表 18
表 2-3　Inv S-Box 替換表 18
表 2-4　總金鑰長度以及金鑰產生之回合數比較表 22
表 4-1　⊗{02}、⊗{04}、⊗{08}、⊗{12}涵式表 37
表 5-1　金鑰擴展資料表 44
表 5-2　金鑰擴展架構比較表 45
表 5-3　MIX 架構比較表 48
表 5-4　8-bit AES 合成結果 49
表 5-5　8-bit AES 比較表 49
表 5-6　加密資料結果對照表 50

圖目錄

圖 1-1 研究步驟流程圖 4

圖 2-1 狀態陣列 6

圖 2-2 AES 加密流程圖 9

圖 2-3 AES 解密流程圖 12

圖 2-4 SubBytes transformation 17

圖 2-5 移列轉換(ShiftRow)圖 19

圖 2-6 反移列轉換(InvShiftRow)圖 19

圖 2-7 MixColumn transformation 20

圖 2-8 金鑰加法運算（AddRoundkey） 21

圖 2-9 Nk=8 之金鑰擴展 23

圖 3-1 AES partial Encryption and decryption round 24

圖 3-2 Coarse grain column computation using BRAM 25

圖 3-3 Block RAM basedimplementation of *SubBytes* and *InvSubBytes* 25

圖 3-4 Implementation of *MixColumns* and *InvMixColumns* 26

圖 3-5 Data arrangement in the folded architecture 27

圖 3-6 ASIP architecture 29

圖 4-1 Dual-port Block RAM Memory 30

圖 4-2 8-bit AES ShiftRow(1) 31

圖 4-3 8-bit AES InvShiftRow(1) 32

圖 4-4 8-bit AES ShiftRow /InvShiftRow(1) 32

圖 4-5 8-bit AES ShiftRow/InvShiftRow(2) 33

圖 4-6 8-bit AES Mixcolumn(1) 34

圖 4-7 8-bit AES InvMixcolumn(2) 35

圖 4-8　MIX 架構圖一 36
圖 4-9　MIX 架構圖二 36
圖 4-10　金鑰擴展之 RotWord/ SubWord/ Rcon/ XOR 38
圖 4-11　金鑰擴展之 w[i]=w[i-4] xor w[i-16] 38
圖 4-12　金鑰擴展-移位暫存器架構 39
圖 4-13　金鑰擴展-使用 Dual port BRAM 40
圖 4-14　8-bit AES 加密架構一 41
圖 4-15　8-bit AES 加密架構二 41
圖 4-16　8-bit AES 加密架構三 42
圖 5-1　金鑰擴展測試結果(1) 46
圖 5-2　金鑰擴展測試結果(2) 47
圖 5-3　MIX 測試結果 48
圖 5-5　8-bit AES 加密測試結果(1) 50
圖 5-6　8-bit AES 加密測試結果(2) 51

第一章　緒論

第一節　研究背景

隨著網路技術的快速發展，網路通訊的安全性及隱密性隨之增加，所以近代密碼學技術的主要研究，在於將電腦的數位訊號，經由數學演算法的轉換後，得到另一筆完全異於原本文件的數位資料，而接收端則需利用特定金鑰和特定演算法才可還原文件。利用數學演法處理資料，可達到資訊保密及身份驗證等功能。而在IC卡的應用上更是一種趨勢，如IC金融卡、健保IC卡等各種智慧卡（smart card）。

近代密碼學[11]主要功能可歸類為下列四項:

一、秘密性（Secrecy or Privacy）：防止非法的接收著發現明文。

二、鑑定性（Authenticity）：確定資訊來源的合法性，亦即此資訊確實是由發送方所傳送。而非別人伪造，或利用以前的訊息來重送。

三、完整性（Integrity）：確定資訊沒有被有意或無意的更改，及被部分取代、加入或刪除等等。

四、不可否認性（Norepudiation）：發送方在事後，不可否認其傳送過之資訊。

對稱式（symmetric）加密演算法可能是密碼學裏最基本最重要的一環。它被大量的使用在各種不同的應用中:包括ATM、e-mail的信件私密保護、影像傳送、資料的儲存和在網路上的數位文件的安全等。

目前的對稱式加密演算法的標準為Data Encryption Standard

（DES）加密演算法，雖然DES在過去的二十幾年來已經提供了一種相當安全的加密演算法；但隨著電子科技的發展與電腦運算速度的提升，1999年已有報導DES可在一天內被破解，而且因為DES對於區塊長度和密鑰長度的限制（區塊為64 bits，密鑰為56 bits），因此我們希望能夠發展一個新的對稱式加密演算法－Advanced Encryption Standard（AES）以適用於未來的二十幾年。

AES 密碼系統是美國近年評選出來的新一代密碼標準、而 2000 年 10 月美國政府機構 National Institute of Standard and Technology（NIST）正式宣布選用 Rijndael 演算法[1]作為 AES、且於 2001 年成為美國聯邦資訊處理加密標準，逐步取代 DES。

第二節　研究動機

AES 演算法為一區塊加密演算法（block cipher），其加密區塊與金鑰長度允許變動。在目前公佈的文件中，已公佈AES演算法可使用128位元、192位元或256位元長度的金鑰，對於128位元、192位元或256位元的加密區塊進行加密。在AES演算法的運作架構中，輸入的明文、每回合運算的結果、輸出的密文等，統稱為狀態（State）。這些狀態資料及所使用的金鑰，皆以陣列的型態表示。在本研究中加密區塊與金鑰長度都是採用128位元（AES-128）。

AES演算法的優點

一、AES演算法可在個人電腦上快速地執行。

二、AES演算法可在智慧卡上（Smart Card）有效率地執行。

三、AES演算法的設計允許明文區塊及金鑰大小由128位元開始，以32 位元為單位，變化至256位元。

四、雖然在文件中，AES演算法的運算回合數是固定的，但也可視需求加以變化。

基於上述之優點，且為了讓AES更普遍於利用，遂有AES cipher（或AES ASIC）之研究及生產。其發展方向可分為小面積[2][3][6]或高產率[5][8][10]兩部份，以滿足於不同應用。在一些消費性電子如行動通訊、RFID、IC卡上並不需要較大的資料傳輸速率，因此8-bit之資料路徑是個不錯的選擇。而Chodowiec [2] and Rouvroy[3]， Tim Good[6]之AES FPGA設計，正擁有小面積之特色。

第三節　研究目的

本論文研究8-bit AES之架構，資料以及金鑰長度為128位元（AES-128），於FPGA上設計及模擬，其小面積及較高產率之設計，以適合不同應用。

在本研究中為了實現以及比較8-bit AES之硬體架構，實驗了三種硬體架構：

一、使用多工器（Mux）

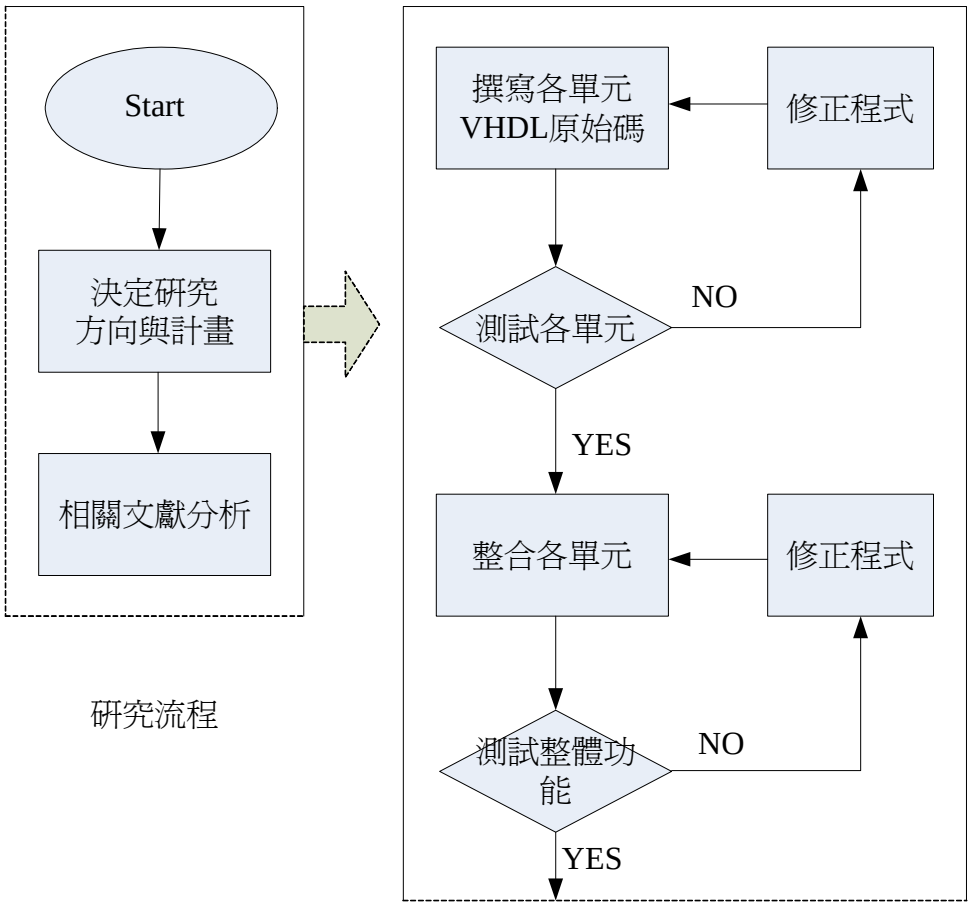
二、使用移位暫存器（Shift Register）

三、使用Xilinx FPGA之Block RAM（簡稱BRAM）

來比較其面積及產率大小；Xilinx FPGAs之架構是以Slice為基本單位，所以我們以使用之Slice來比較面積之大小；而產率則為每秒產出百萬位元為單位（Mbps）。

第四節　研究步驟

本研究的步驟可以分為一、文獻探討部份；二、撰寫 VHDL 程式碼；三、測試實驗成果三部份，如圖 1-1 所示。



研究流程

圖 1-1　研究步驟流程圖

第二章　AES 介紹－Rijndael 演算法

西元一九九七年由 NIST（National Institute of Standard and Technology ）公開徵求的選拔中，從最初的十五個演算法，到十個、五個，逐步篩選出適合用來作為下一代加密演算法的標準。Rijndael 在經過了重重篩選後。直至西元二千年十月二日，Rijndael 才脫穎而出。

Rijndael演算法[1]（Rijndael Algorithm）由Joan Daemen和Vincent Rijmen共同發明，經過美國國家標準與技術協會的驗證與篩選後脫穎而出，成為 AES 演算法的標準，因此之後本研究中即統一用 AES 代表 Rijndael 演算法。

AES演算法為一區塊加密演算法（block cipher），其加密區塊與金鑰長度允許變動。在目前公佈的文件中，已公佈AES演算法可使用128位元、192位元或256位元長度的金鑰，以及128位元的加密區塊進行加密。AES演算法亦考慮到利用軟硬體實作的相關問題，因此AES演算法能夠有效地利用不同的軟硬體加以實作。

在 AES 演算法的運作架構中，輸入的明文、每回合運算的結果、輸出的密文等，統稱為狀態（State），也就是每一個位元組的資料（Byte）。這些狀態資料及所使用的金鑰，皆以陣列的型態表示如下圖 2-1。

在以下各節將會分別介紹 Rijndael 加解密之架構、數學背景、位元組取代轉換（SubByte）/反位元組取代轉換（InvSubByte）、移列轉換（ShiftRow）/反移列轉換（InvShiftRow）、混行轉換（MixColumn）/反混行轉換（InvMixColumn）、金鑰加法函數（AddRoundkey）、金鑰擴展(Key expansion)。

Input ：00112233445566778899aabbccddeeff

Output：000102030405060708090a0b0c0d0e0f

Input

00	*44*	*88*	*cc*
11	*55*	*99*	*dd*
22	*66*	*aa*	*ee*
33	*77*	*bb*	*ff*

Output

00	*04*	*08*	*0c*
01	*05*	*09*	*0d*
02	*06*	*0a*	*0e*
03	*07*	*0b*	*0f*

圖 2-1　狀態陣列

第一節　Rijndael 加解密之架構

Rijndale 加密演算法需 SubByte、ShiftRow、MixColumn、及 AddRoundKey 四步驟。而 AES 解密演算法則需 InvByteSub、InvShiftRow、InvMixColumn、及 AddRoundKey 四步驟；基本上解密過程為加密過程之顛倒，以及些許的改變。AES 演算法加解密流程圖如圖 2-2、圖 2-3 所示，各步驟及差異會在之後各節詳細說明。

Rijndale 演算法可使用 128 位元、192 位元或 256 位元長度的金鑰，以及 128 位元的明文進行加密。根據金鑰長度以及明文長度決定加解密所要進行的回合數（Nr）如表 2-1 所示；Nr、Nb、Nk 之定義如下：

Nr：加解密所要進行之回合數。

Nb：明文之資料長度（Bit）除以 32 所得之數字。

Nk：金鑰之資料長度（Bit）除以 32 所得之數字。

表 2-1 加解密回合數關係表

	金鑰資料長度(Nk)	明文資料長度(Nb)	回合數目(Nr)
AES-128	**4**	**4**	**10**
AES-192	**6**	**4**	**12**
AES-256	**8**	**4**	**14**

一、Rijndale 加密演算法

Rijndale 加密演算法需 SubByte、ShiftRow、MixColumn、及 AddRoundKey 四步驟。Rijndale 加密演算法虛擬碼[1]如下所示，經由一開始的金鑰相加，接者（Ｎｒ-1）次的回合運算，最後在加上終回合運算而結束，圖 2-2 為 AES 加密流程圖。其中加密之金鑰相加、回合運算、終回合運算分別表示如下：

金鑰相加：AddRoundKey(state, w[0, Nb-1])

回合運算：SubBytes(state) → ShiftRows(state) →
MixColumns(state) →
AddRoundKey(state, w[round*Nb,
(round+1)*Nb-1])

終回合運算：SubBytes(state) → ShiftRows(state) →
AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

```
    //Rijndale加密演算法
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])        //一開始的金鑰相
加
for round = 1 step 1 to Nr–1                    //（Ｎｒ-1）次的回
合運算
        SubBytes(state)
        ShiftRows(state)
        MixColumns(state)
        AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
end for
        SubBytes(state)                         // 終回合運算
        ShiftRows(state)
        AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
out = state
end
```

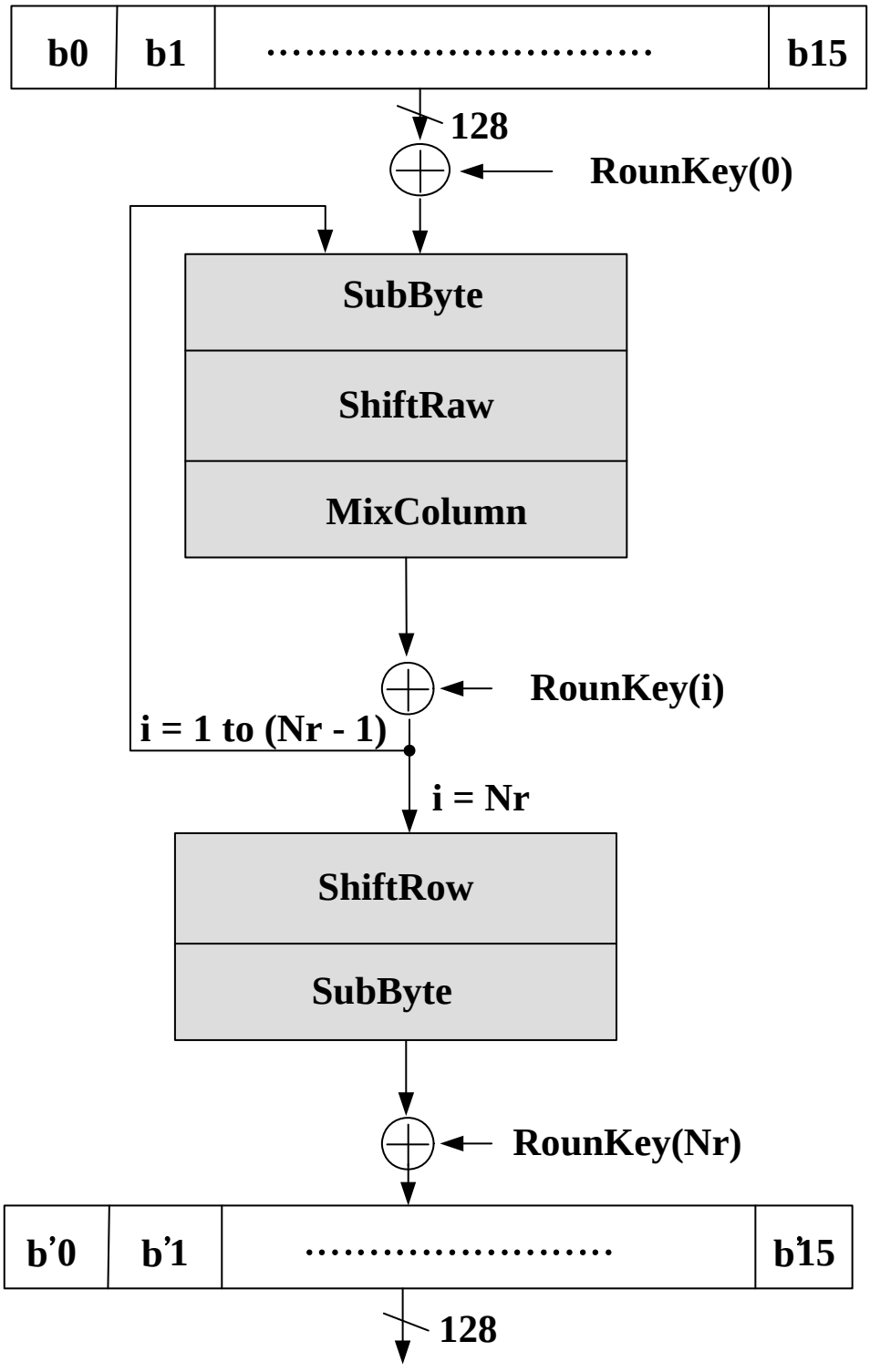


圖 2-2　AES 加密流程圖

二、Rijndale 解密演算法

Rijndale 解密演算法則需 InvByteSub、InvShiftRow、InvMixColumn、及 AddRoundKey 四步驟；基本上解密過程為加密過程之顛倒，而且金鑰順序則是跟加密時的順序顛倒。Rijndale 解密演算法虛擬碼[1]如下所示，經由一開始的金鑰相加，接者（Nr-1）次的回合運算，最後在加上終回合運算而結束，圖 2-3 為 AES 解密流程圖。其中解密之金鑰相加、回合運算、終回合運算分別表示如下：

金鑰相加：AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

回合運算：InvShiftRows(state) → InvSubBytes(state) → AddRoundKey(state, w[round*Nb, (round+1)*Nb-1]) → InvMixColumns(state)

終回合運算：InvShiftRows(state) → InvSubBytes(state) → AddRoundKey(state, w[0, Nb-1])

////Rijndale解密演算法

```
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]) //一開始的金鑰相加
for round = Nr-1 step -1 downto 1                  //（Ｎｒ-1）次的回合運算
        InvShiftRows(state)
        InvSubBytes(state)
        AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
        InvMixColumns(state)
end for
        InvShiftRows(state)                        // 終回合運算
        InvSubBytes(state)
        AddRoundKey(state, w[0, Nb-1])
out = state
end
```

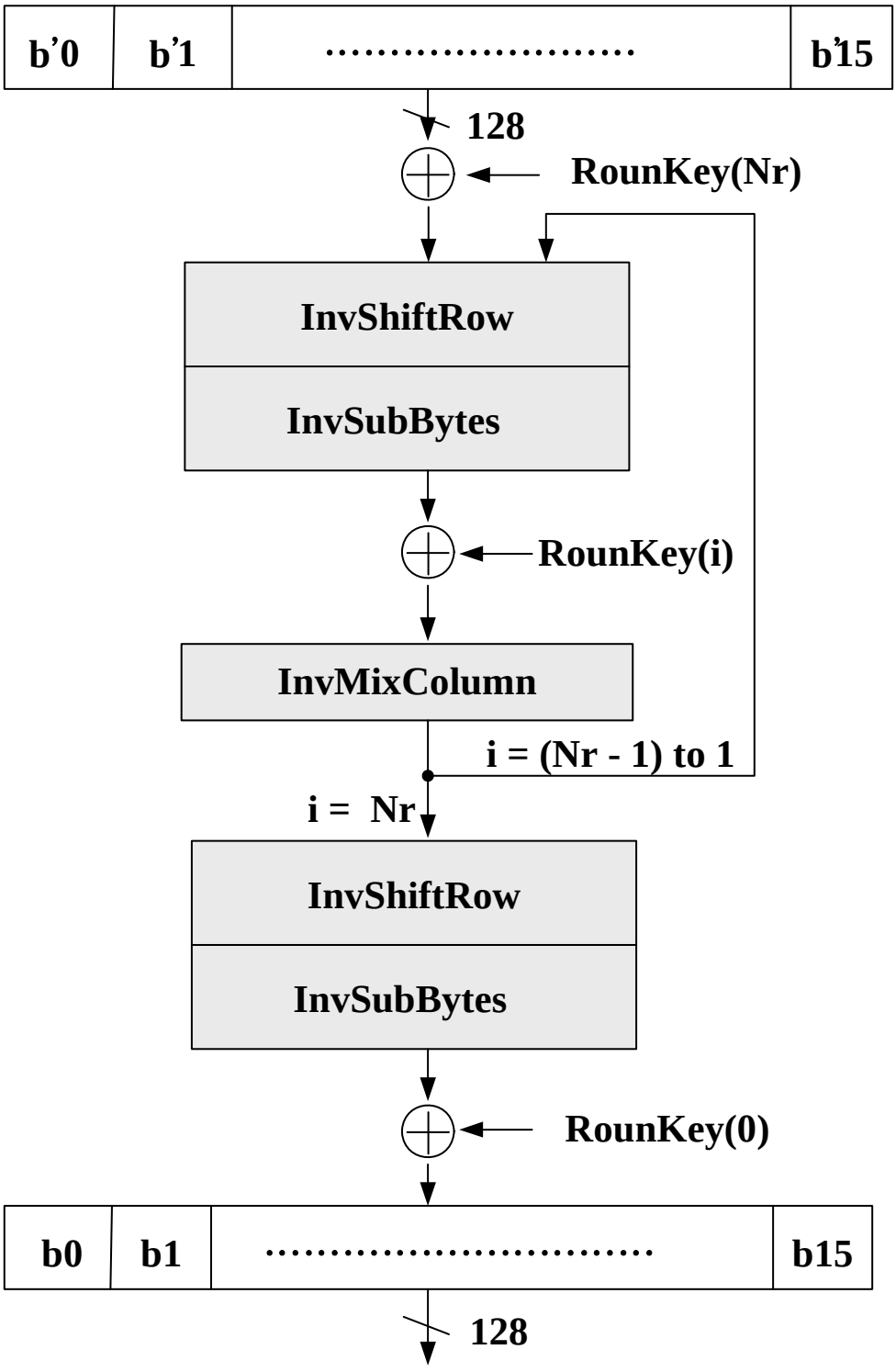


圖 2-3　AES 解密流程圖

第二節　數學背景

Rijndale 演算法中有幾個運算是以位元組（byte）為計算單位，並且是在有限場（finite field）GF（2^8）中做運算（例如AddRoundkey）。而一個有限場中的元素可以被表示成很多種不同的型態，在此使用多項式表示法和二位元表示法。其中常用為有限場之加法（同減法）、乘法…等。

一個位元組（byte）a 是由 8 個 bits 所組成的，其二位元表示法為 $\{a_7\ a_6\ a_5\ a_4\ a_3\ a_2\ a_1\ a_0\}$ 我們可以將它視為一個係數為 0 或 1 的多項式如下:

$$a(x) = a_7 x^7 + a_6 x^6 + a_5 x^5 + a_4 x^4 + a_3 x^3 + a_2 x^2 + a_1 x^1 + a_0 x^0$$

例如：

16 進位表示法｛57｝的 2 進位表示法為｛01010111｝，它所對應的多項式則為：

$$x^6 + x^4 + x^2 + x + 1$$

以下將介紹有限場之運算基本原理：

一、有限場之加法

在有限場中的加法是被定義為多項式的二進位加法，也就是在模 2（modulo 2）運算下做係數相加，相當於數位邏輯中的 XOR 運算（由'⊕'表示）。

例如：

$$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2$$

{01010111} + {10000011} = {11010100}

{57} ⊕ {83} = {d4}

由上面的結果我們可以得到知在有限場中的加法是相當於互斥或運算；因此有限場之減法和加法結果是一樣的。

二、有限場之乘法及乘法反元素

在有限場 GF（2^8）中的乘法（由'●'表示）是被定義為多項式的二進位乘法，其差別為，若其積之階數等於或比 8 大時，必須模（modulo）一個不可分解的八次多項式（irreducible polynomial）。對於 Rijndael 演算法，我們給定此多項式為 **$m(x)= x^8+ x^4+ x^3+ x+1$**，或者以 16 進位表示法｛11B｝來表示。其結果一定是一個次方低於八次的二進位多項式。

例如：

{57} ● {83} = {D5}

$(x^6+ x^4+ x^2+ x+1)(x^7+ x+1) = x^{13} + x^{11} + x^9 + x^8 + x^7 + x^7 + x^5 + x^3 + x^2 + x + x^6 + x^4 + x^2 + x + 1$

$= x^{13}+ x^{11}+ x^9+ x^8+ x^6+ x^5+ x^4+ x^3+1$

$(x^{13}+ x^{11}+ x^9+ x^8+ x^6+ x^5+ x^4+ x^3+1)$ modulo $(x^8+ x^4+ x^3+ x+1)$

$= x^7 + x^6 + x^4 + x^3 +1$

對任意次方低於八次的二進位多項式 b(x)而言，且 $m(x)= x^8+ x^4+ x^3+ x+1$ 為不可分解之多項式（irreducible polynomial），所以利用歐基里德演算法可以求出多項式 a(x)、c(x)使得

$$b(x)a(x) + m(x)c(x) = 1$$

因此

$a(x)b(x)=1 \pmod{m(x)} \quad <=> \quad b^{-1}(x) = a(x) \pmod{m(x)}$

其中 $b^{-1}(x)$稱為其乘法反元素。

例如：

{57} ● {bf} = 1 (mod m(x))

{bf}為{57}之乘法反元素；{57}亦為{bf}之乘法反元素。

三、係數為有限場之多項式

多項式的係數可以被定義在有限場 GF（2^8）上。經由這個方法，一個 4-byte 的向量可以被表示為一個次方低於四次的多項式。多項式的加法可以透過簡單的係數相加來達成。因為在有限場 GF（2^8）上的加法是做 XOR 的運算，因此兩個向量的加法就是簡單的 XOR 運算。

乘法的運算就比較複雜了。假設在有限體 GF（2^8）上我們有兩個多項式：

$$a(x) = a_3x^3 + a_2x^2 + a_1x + a_0 \quad \{a_3、a_2、a_1、a_0\} \in GF(2^8)$$

$$b(x) = b_3x^3 + b_2x^2 + b_1x + b_0 \quad \{b_3、b_2、b_1、b_0\} \in GF(2^8)$$

它們的加法 a(x)+b(x)運算如下：

$$a(x) + b(x) = (a_3 \oplus b_3)x^3 + (a_2 \oplus b_2)x^2 + (a_1 \oplus b_1)x^1 + (a_0 \oplus b_0)$$

它們的乘積 c(x) = a(x)b(x)運算如下：

$$c(x) = c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0$$

$$\{c_6、c_5、c_4、c_3、c_2、c_1、c_0\} \in GF(2^8)$$

其中

$$c_0 = a_0 \bullet b_0$$

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

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

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

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

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

$$c_6 = a_3 \bullet b_3$$

我們清楚的知道 c(x)已經超過 4-byte 的向量表示範圍，因此藉由模（modulo）一個四次多項式的運算，我們就可以將 c(x)的

次方降到四次以下。以 Rijndael 演算法來說，我們選定的四次多項式 $M(x)=x^4+1$(因為 $x^j \bmod x^4+1=x^{j \bmod 4}$)。我們定義兩個多項式 a(x)、b(x)的模乘法運算 $d(x)=a(x)\otimes b(x)$為

$$d(x) = d_3x^3 + d_2x^2 + d_1x + d_0$$

其中

$$d_0=(a_0\bullet b_0)\oplus(a_3\bullet b_1)\oplus(a_2\bullet b_2)\oplus(a_1\bullet b_3)$$

$$d_1=(a_1\bullet b_0)\oplus(a_0\bullet b_1)\oplus(a_3\bullet b_2)\oplus(a_2\bullet b_3)$$

$$d_2=(a_2\bullet b_0)\oplus(a_1\bullet b_1)\oplus(a_0\bullet b_2)\oplus(a_3\bullet b_3)$$

$$d_3=(a_3\bullet b_0)\oplus(a_2\bullet b_1)\oplus(a_1\bullet b_2)\oplus(a_0\bullet b_3)$$

這一個運算可以使用矩陣的乘法來表示如下:

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

對於 a(x)而言，因為 $M(x)=x^4+1$ 並非為不可分解之多項式（irreducible polynomial）所以並不保證其乘法反元素 $a^{-1}(x)$存在。在 Rijndael 演算法中使用一組乘法反元素如下，在混行轉換（MixColumn）及反混行轉換（InvMixColumn）中使用。

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$$

$$a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}$$

第三節　位元組取代轉換（SubByte / InvSubByte）

位元組轉換（SubBytes）是一個以位元組為單位的非線性替換運算，替換表（S-box）是經過兩個運算過程而建立，並且是可逆的。

首先找出每個位元組在 GF（2^8）中的乘法反元素｛$b_7\,b_6\,b_5\,b_4\,b_3\,b_2\,b_1\,b_0$｝，接著經過一個仿射（Affine）轉換運算而得｛$b_7'\,b_6'\,b_5'\,b_4'\,b_3'\,b_2'\,b_1'\,b_0'$｝，其結果如表 2-2 所示，計算公式如下：

$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

下圖 2-4 是 SubBytes 所運算的結果示意圖：

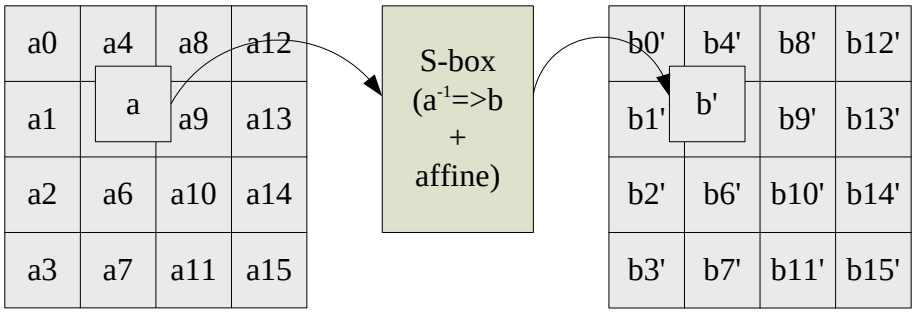


圖 2-4　SubBytes transformation

位元組轉換(SubBytes)除了可以藉由求出其乘法反元素來得到外也可以利用下表表 2-2 S-Box 替換表來達到相同的目的。

表 2-2　S-box 替換表

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

反位元組轉換（InvSubBytes）也是一個以位元組為單位的非線性替換運算，替換結果如表 2-3 所示。

表 2-3　Inv S-Box 替換表

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

第四節　移列轉換(ShiftRow/InvShiftRow)

在這個轉換中，狀態（State）的每一列以不同的偏移量做環狀位移，第 0 列不動，第一列位移一個位元組，第二列位移二個位元組，第三列位移三個位元組。

移列轉換（ShiftRow）運算對於狀態（State）的影響，如圖 2-5 所示：第二第三及第四列做 1,2,3 個位元組的左旋即可。

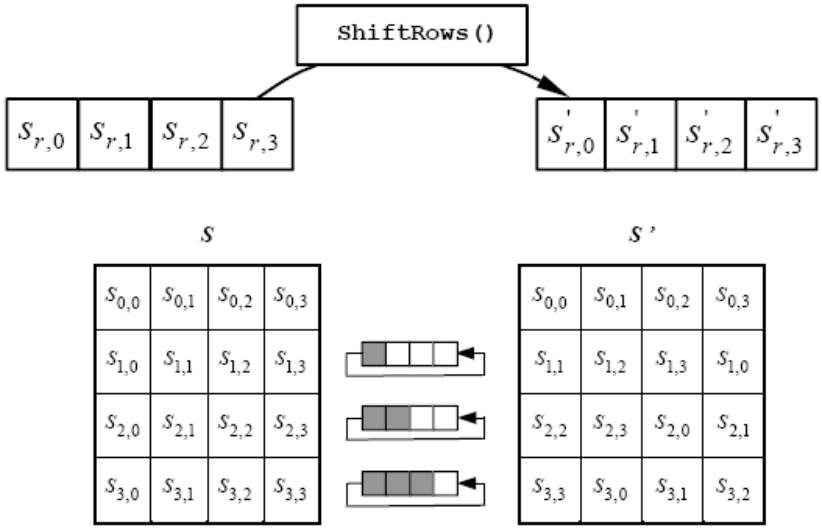


圖 2-5　移列轉換(ShiftRow)圖

反移列轉換（InvShiftRow）運算對於狀態（State）的影響，如圖 2-6 所示；和移列轉換（ShiftRow）運算差別只在於旋轉的方向（右旋）而已。

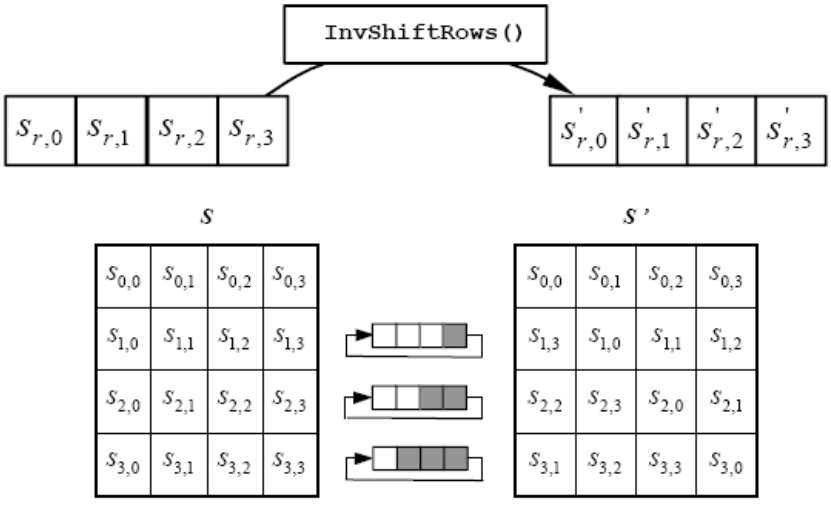


圖 2-6　反移列轉換(InvShiftRow)圖

第五節　混行轉換(MixColumn/ InvMixColumn)

混行轉換（MixColumn）運算中，狀態表的每一行（Column）是被視為在 GF（2^8）中的多項式，並且乘上一個固定的多項式 C(x)後再同餘於多項式（x^4+1），如第二節係數為有限場之多項式定義，而其中

$$c(x)= \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$$

我們可將其表示矩陣乘法。令 $b(x) = c(x) \otimes a(x)$

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

我們把上面這一個運算的過程以 MixColumn（State）表示之。在下圖 2-7 中將把一個狀態表經過運算後的結果表示出來。運算結果也可看成另一種替換方式。

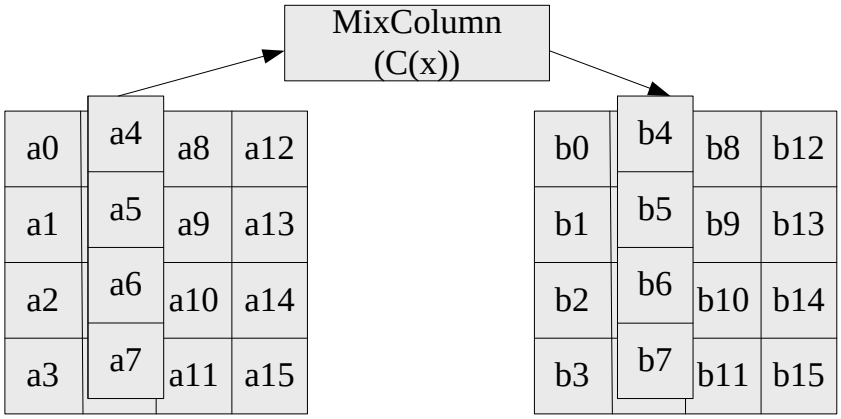


圖 2-7　MixColumn transformation

反混行轉換（InvMixColumn）運算中，狀態表的每一行是被視為在 GF(2^8)中的多項式，並且乘上一個固定的多項式 $C^{-1}(x)$後

再同餘於多項式（x^4+1），而其中

$$c^{-1}(x)= \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}$$

我們可將其表示矩陣乘法。令 $b(x) = c^{-1}(x) \otimes a(x)$

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

混行轉換（MixColumn）運算和反混行轉換（InvMixColumn）運算之差異為乘上兩不同多項式 c(x)及其乘法反元素 $c^{-1}(x)$。

第六節　金鑰加法運算(AddRoundkey)

金鑰加法運算（AddRoundkey）將會把狀態值與子金鑰做互斥或（XOR）的運算。而這一個子金鑰是由密鑰經過（key expansion）所產生出來的，其長度等於明文長度（N_b）；金鑰擴展將再下一節介紹。而這整個步驟如圖 2-8 所示。

a0	a4	a8	a12
a1	a5	a9	a13
a2	a6	a10	a14
a3	a7	a11	a15

⊕

b0	b4	b8	b12
b1	b5	b9	b13
b2	b6	b10	b14
b3	b7	b11	b15

=

c0	c4	c8	c12
c1	c5	c9	c13
c2	c6	c10	c14
c3	c7	c11	c15

圖 2-8　金鑰加法運算（AddRoundkey）

第七節　金鑰擴展(Key expansion)

金鑰擴展（Key expansion）是將主金鑰使用線性位移轉換（RotWord）、狀態值替換（SubWord）、加上回合常數（Rcon）、以及互斥或（XOR）等運算，共產生（Nb（Nr+1）*4）個位元組（Byte）子金鑰。而回合常數（Rcon），是根據 Nk 值而有不同的回合數所決定，下表 2-4 為不同金鑰長度（Nk）、總金鑰長度以及所進行之回合數比較。

表 2-4　總金鑰長度以及金鑰產生之回合數比較表

	Nr	*總金鑰長度(Bytes)*	*金鑰產生之回合數*
Nk=4 (AES-128)	*10*	*44*4*	*10*
Nk=6 (AES-196)	*12*	*52*4*	*8*
Nk=8 (AES-256)	*14*	*60*4*	*7*

當 Nk=8 時，其金鑰產生又會有不同方式。如圖 2-9 我們可知 Nk=8，當 i=n*Nk 時，要進行線性位移轉換（RotWord）、狀態值替換（SubWord）、加上回合常數（Rcon）、以及互斥或（XOR）等運算；不同的地方為，當 i=12+n*Nk 時要多進行一次狀態值替換（SubWord）運算；也就是說 Nk=4 和 Nk=6 不必多進行一次狀態值替換（SubWord）運算。詳細步驟可參考 NIST 之說明。

Cipher Key = 60 3d eb 10 15 ca 71 be 2b 73 ae f0 85 7d 77 81
1f 35 2c 07 3b 61 08 d7 2d 98 10 a3 09 14 df f4

for $Nk = 8$, which results in

w_0 = 603deb10 w_1 = 15ca71be w_2 = 2b73aef0 w_3 = 857d7781

w_4 = 1f352c07 w_5 = 3b6108d7 w_6 = 2d9810a3 w_7 = 0914dff4

i (dec)	temp	After RotWord()	After SubWord()	Rcon[i/Nk]	After XOR with Rcon	w[i-Nk]	w[i]= temp XOR w[i-Nk]
8	0914dff4	14dff409	fa9ebf01	01000000	fb9ebf01	603deb10	9ba35411
9	9ba35411					15ca71be	8e6925af
10	8e6925af					2b73aef0	a51a8b5f
11	a51a8b5f					857d7781	2067fcde
12	2067fcde		b785b01d			1f352c07	a8b09c1a
13	a8b09c1a					3b6108d7	93d194cd
14	93d194cd					2d9810a3	be49846e
15	be49846e					0914dff4	b75d5b9a
16	b75d5b9a	5d5b9ab7	4c39b8a9	02000000	4e39b8a9	9ba35411	d59aecb8
17	d59aecb8					8e6925af	5bf3c917
18	5bf3c917					a51a8b5f	fee94248
19	fee94248					2067fcde	de8ebe96
20	de8ebe96		1d19ae90			a8b09c1a	b5a9328a
21	b5a9328a					93d194cd	2678a647
22	2678a647					be49846e	98312229

圖 2-9　Nk=8 之金鑰擴展

第三章　相關研究

在本章中我們探討幾位學者之研究成果，其研究方向也是在縮小面積上，而其貢獻對於本論文之影響深遠，也提供不同方向之思考模式；以下將逐一介紹其研究以及特色。

一、Ricardo Chaves[4]架構

Ricardo Chaves 架構主要是使用 memory-bases，將一些複雜計算，預先產生，然後儲存於記憶體內（BRAM），用以減少運算量；同時使用記憶體除了可以增快頻率（frequency），也可減少 slices 之使用量，提升整體效能。

其 memory-bases 如下圖 3-1 所示，將混行轉換（MixColumn）及反混行轉換（InvMixColumn）所需之 S’ ⊗ {03 01 01 02}及 S’ ⊗ {0b 0d 09 0e}之結果同時儲存在同一個記憶體（BRAM），以減少運算量，更加快了工作頻率。

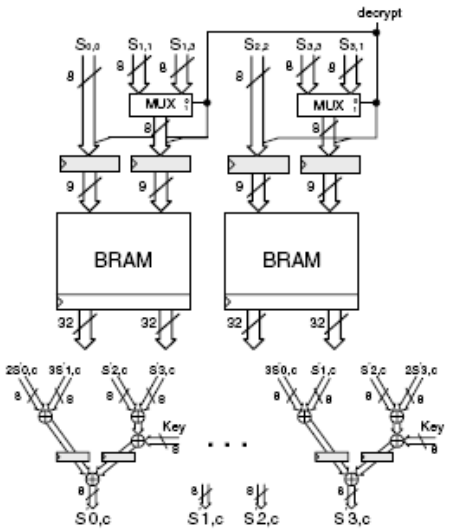


圖 3-1　AES partial Encryption and decryption round

Ricardo Chaves 架構還有一重要特色為：使用雙埠記憶體（Dual port BRAM）如圖 3-2 所示可以將記憶體之使用數量減少

一半，以節省資源；而其位元組轉換（SubBytes）和混行轉換（MixColumn）是分兩階段進行。值得一提的是金鑰擴展（Key expansion）這一步驟，在 Ricardo Chaves 架構中並沒有加以硬體實現及探討。

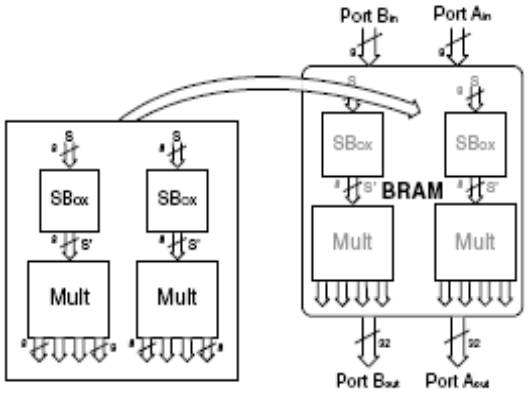


圖 3-2　Coarse grain column computation using BRAM

二、Pawel Chodowiec [2]架構

Pawel Chodowiec 之架構也是使用 FPGA 之 Dual port BRAM，如下圖 3-3 所示，位元組轉換（SubBytes）及反位元組轉換（InvSubBytes）都放在同一個 Dual port BRAM，用 ADDR[8] 來控制選擇加密還是解密，以達到節省面積之目的。

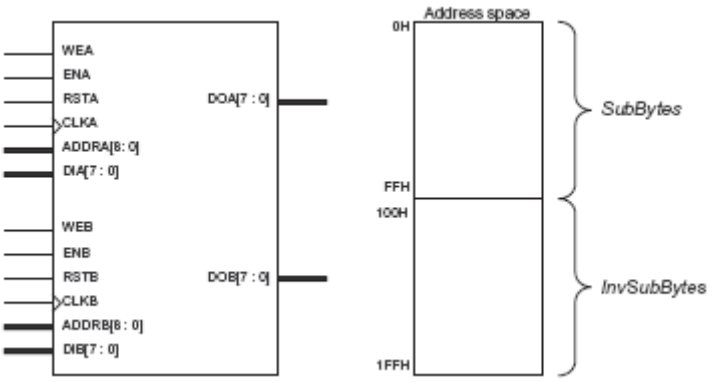


圖 3-3　Block RAM basedimplementation of *SubBytes* and *InvSubBytes*

Pawel Chodowiec and Kris Gaj 之架構在混行轉換（MixColumn）及反混行轉換（InvMixColumn）上採用不同之資源共享方式如下圖 3-4；將混行轉換（MixColumn）及反混行轉換（InvMixColumn）所要使用之多項式 c(x)及 $c^{-1}(x)$整理推導如下：

$$c(x)= \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$$

$$d(x) = c^{-1}(x)= \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}$$

因為

$$c(x) \otimes d(x) =\{01\}$$

所以

$$c(x) \otimes d^2(x) =d(x)$$

且

$$d^2(x) = \{04\}x^2 + \{05\}$$

所以在進行混行轉換（MixColumn）時乘以多項式 c(x)，而進行反混行轉換（InvMixColumn）則要選擇 $c(x) \otimes d^2(x)$代替原本之多項式 d(x)。

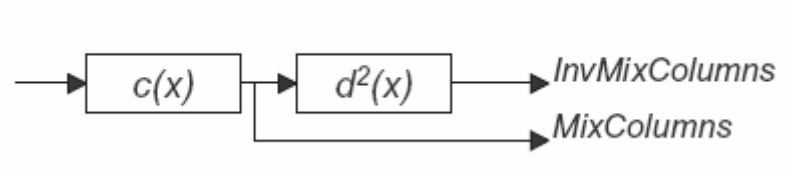


圖 3-4　Implementation of *MixColumns* and *InvMixColumns*

Pawel Chodowiec 之 AES 運算架構如下圖 3-5 所示，作者將輸入（Input）以及輸出（Output）兩個部份之暫存器以記憶體（Dual port BRAM）取代，用以減少 Slice 之使用，達到小面積、低成本之要求。

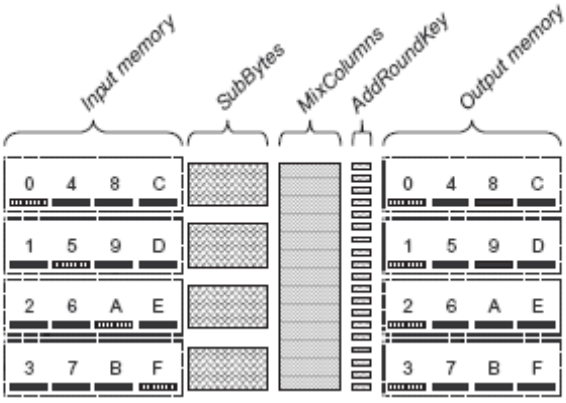


圖 3-5　Data arrangement in the folded architecture

三、Gael Rouvroy[3]架構

Gael Rouvroy提出之架構，使用Xilinx Spartan-3和Virtex-II FPGAs晶片來設計AES演算法，其目的是使用18-Kbit之大容量之Dual port BRAM，利用查表法（Look Up Tables），將位元組轉換（SubBytes）及混行轉換（MixColumn）都整合存放在同一記憶體內，其整合方式如下。

將原本要分開執行之位元組轉換（SubBytes）：$SB(a_0)$、$SB(a_1)$、$SB(a_2)$、$SB(a_3)$；和之後進行的混行轉換（MixColumn），經由查表$T_0(a_0)$、$T_1(a_1)$、$T_2(a_2)$、$T_3(a_3)$作互斥或（XOR）運算即可得。

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} SB(a_0) \\ SB(a_1) \\ SB(a_2) \\ SB(a_3) \end{bmatrix}$$

$$T_0(a) = \begin{bmatrix} 02 \otimes SB(a) \\ SB(a) \\ SB(a) \\ 03 \otimes SB(a) \end{bmatrix} \qquad T_1(a) = \begin{bmatrix} 03 \otimes SB(a) \\ 02 \otimes SB(a) \\ SB(a) \\ SB(a) \end{bmatrix}$$

$$T_2(a)=\begin{bmatrix} SB(a) \\ 03\otimes SB(a) \\ 02\otimes SB(a) \\ SB(a) \end{bmatrix} \qquad T_3(a)=\begin{bmatrix} SB(a) \\ SB(a) \\ 03\otimes SB(a) \\ 02\otimes SB(a) \end{bmatrix}$$

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = T_0(a_0)\oplus T_1(a_1)\oplus T_2(a_2)\oplus T_3(a_3)$$

同理反位元組轉換（InvSubBytes）及反混行轉換（InvMixColumn）也可依上述公式求得；且加密之 $T_0(a_0)$、$T_1(a_1)$、$T_2(a_2)$、$T_3(a_3)$各表大小為 8Kbit，所以解密 $IT_0(a_0)$、$IT_1(a_1)$、$IT_2(a_2)$、$IT_3(a_3)$之大小也相同。因此（$T_0(a_0)$| $IT_0(a_0)$）、（$T_1(a_1)$| $IT_1(a_1)$）、（$T_2(a_2)$| $IT_2(a_2)$）、（$T_3(a_3)$| $IT_3(a_3)$）之大小則為 16Kbit，可置於 Xilinx Spartan-3 和 Virtex-II FPGAs 之 BRAM。

四、Tim Good[6]架構

Tim Good此篇論文主要在陳述一非常小之AES特殊應用指令處理器（application-specific instruction processor，ASIP），它結合了軟體跟硬體（co-design），其重心是放在小面積上，產率（throughput）只是第二考量因素。Tim Good的處理器架構，在執行AES工作時，指令的數目高達30個以上，而總共花費的時間也超過3000個時脈（CLK），相對的產率也跟著下降，圖3-6為Tim Good的處理器架構。

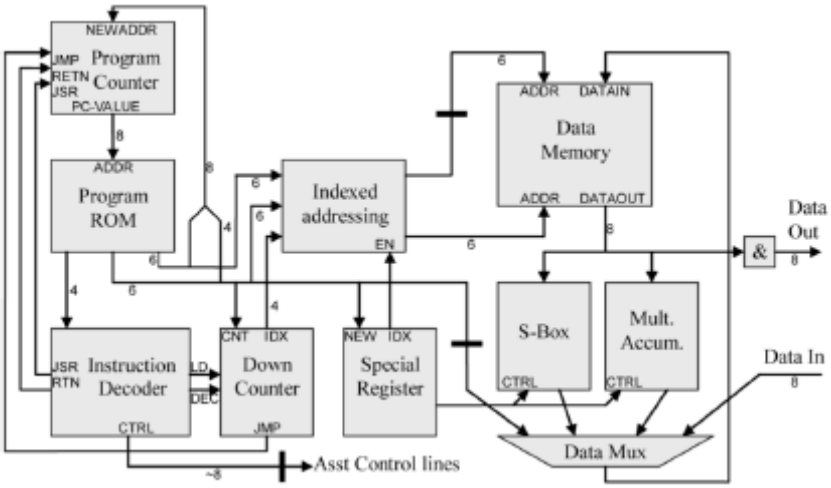


圖 3-6　ASIP architecture

第四章　8-bit AES 架構設計

利用軟體提供加解密的技術，已經無法滿足資料在網路的傳輸速度需求。利用 Xilinx 的 FPGA 平台，實現 AES 加解密演算法，提高處理速度，可以隨時配合不同環境所需的架構。本研究在 Xilinx FPGA Spartan2（xc2s15）上設計 AES-128 架構，此晶片共包含 194 Slices 以及四個 BRAM 可供使用。

第一節　8-bitAES 位元組取代轉換（SubByte / InvSubByte）

在本論文中，位元組轉換（SubBytes）一樣採用查表法，和 Pawel Chodowiec and Kris Gaj 所使用之架構相同，將雙埠記憶體（Dual port BRAM）切成 S-box 及 InvS-box 兩部份，大小共為 512-Byte，PortA 被金鑰擴展（Key expansion）時使用，PortB 則在進行加密時使用，其示意圖如下。

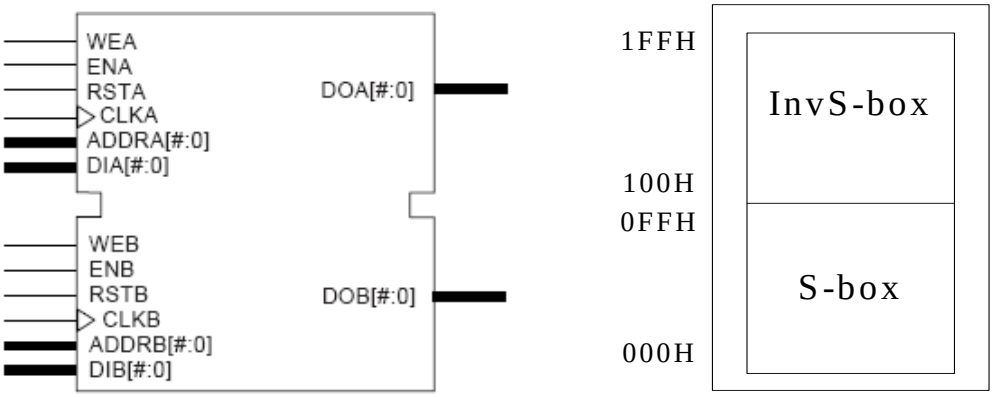


圖 4-1　Dual-port Block RAM Memory

第二節　8-bit AES 移列轉換
(ShiftRow/InvShiftRow)

在進行移列轉換（ShiftRow）時，如第二章第四節所示。本研究中使用暫存器、以及 BRAM 來完成移列轉換以及反移列轉換；下圖 4-2 為移列轉換之架構一，使用暫存器轉移之方法；下圖 4-3 為反移列轉換，暫存器轉移之架構。使用暫存器轉移，將會使用 32 個暫存器（register），而且在連線上稍嫌複雜。圖 4-4 則是將移列轉換和反移列轉換整合在一起之架構，多出了八個 8-bit 二對一多工器，使用 EN_DE 來做選擇。

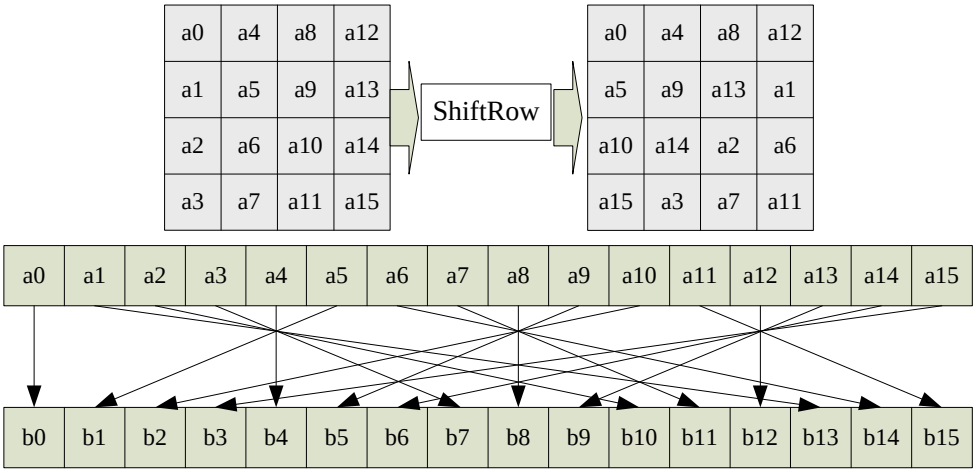


圖 4-2　8-bit AES ShiftRow(1)

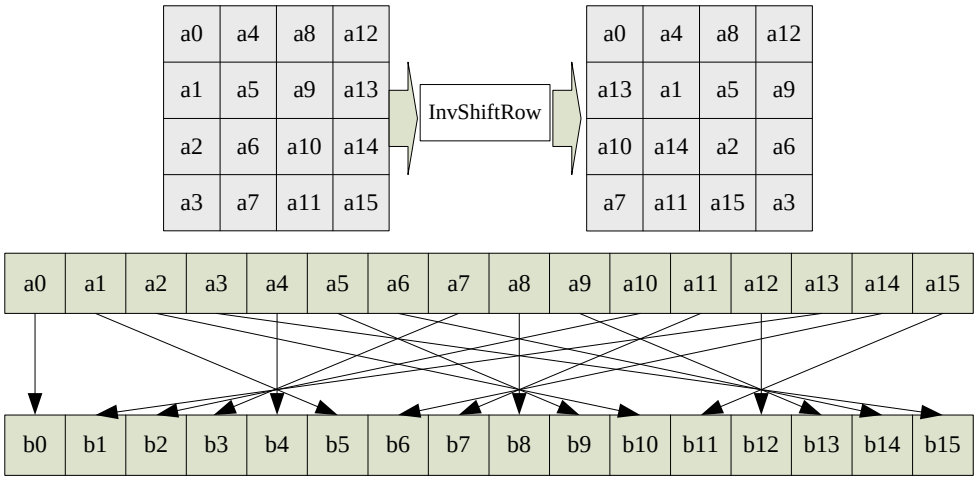


圖 4-3　8-bit AES InvShiftRow(1)

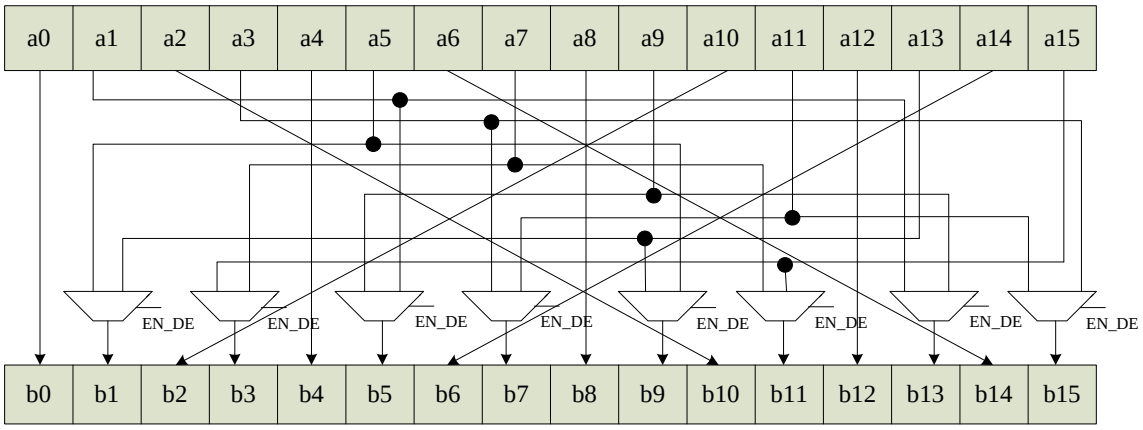


圖 4-4　8-bit AES ShiftRow /InvShiftRow(1)

第二種方法為使用 Dual Port BRAM，使用 Dual Port BRAM 的好處可以減少 Slice 的使用，而在控制上只要設定記憶體位址（Address）就可以達到移列轉換以及反移列轉換的目的，如圖 4-4 所示；在進行移列轉換時，只要由 EN_DE 來選擇加 5 計數器來輸出{a0 a5 a10 a15}…{a12 a1 a6 a11}；而在進行反移列轉換時則選擇加 13（或減 3）之計數器。

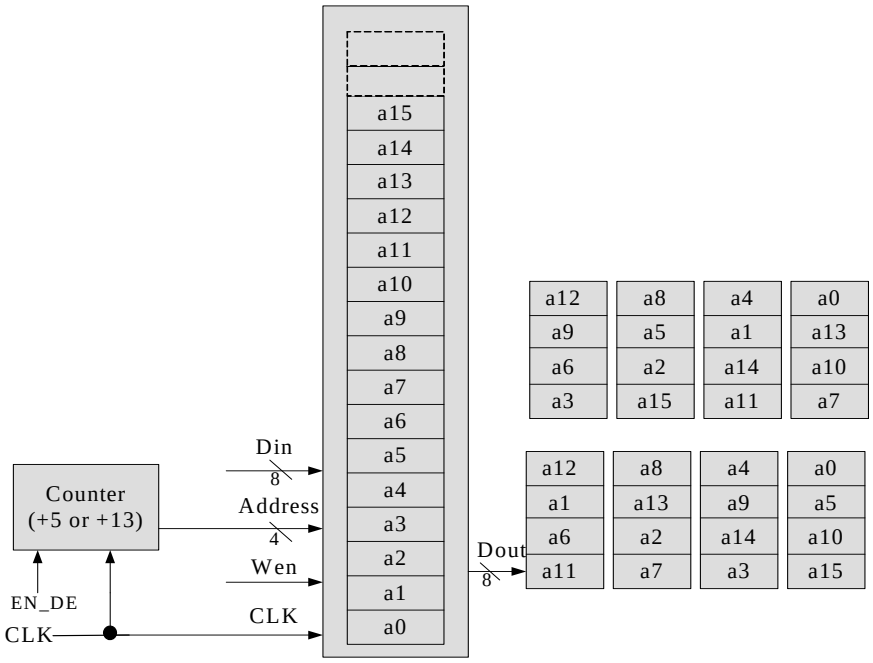


圖 4-5　8-bit AES ShiftRow/InvShiftRow(2)

第三節　8-bit AES 混行轉換(MixColumn/ InvMixColumn)

8-bit AES 混行轉換可由 $b(x) = c(x) \otimes a(x)$（或 $b(x) = c^{-1}(x) \otimes a(x)$）推導而來，其中 $c(x)= \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$（或 $c^{-1}(x)= \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}$），經由以下推導可求得 b_0、b_1、b_2、b_3，為了節省四分之三的計算，我們只要設計一個⊗[02 03 01 01]（和⊗[0e 0b 0d 09]）就可達到目的。以下本研究實驗兩個方法來比較面積的大小、Slice 的使用量。

第一種如圖 4-5 所示，使用多工器（multiplexer），總共用四個四對一 8-bit 多工器；第二種則使用左旋移位暫存器（rotate left register），如圖 4-6 所示，兩種方式都可以達到移位的目的。

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

$$b_0 = [02 \;\; 03 \;\; 01 \;\; 01] \otimes [a_0 \;\; a_1 \;\; a_2 \;\; a_3]^T$$
$$b_1 = [02 \;\; 03 \;\; 01 \;\; 01] \otimes [a_1 \;\; a_2 \;\; a_3 \;\; a_0]^T$$
$$b_2 = [02 \;\; 03 \;\; 01 \;\; 01] \otimes [a_2 \;\; a_3 \;\; a_0 \;\; a_1]^T$$
$$b_3 = [02 \;\; 03 \;\; 01 \;\; 01] \otimes [a_3 \;\; a_0 \;\; a_1 \;\; a_2]^T$$

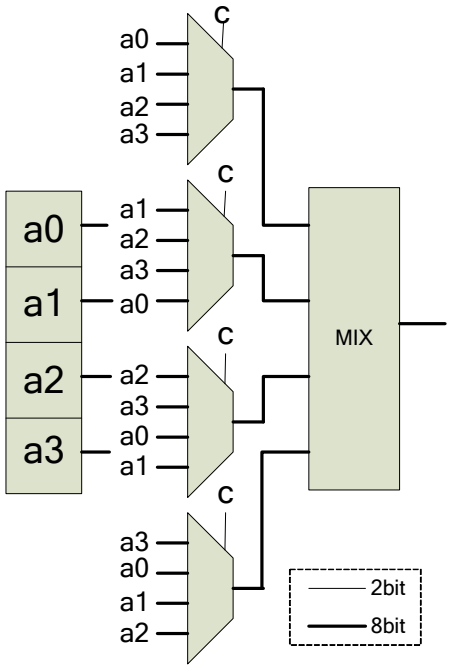


圖 4-6　8-bit AES Mixcolumn(1)

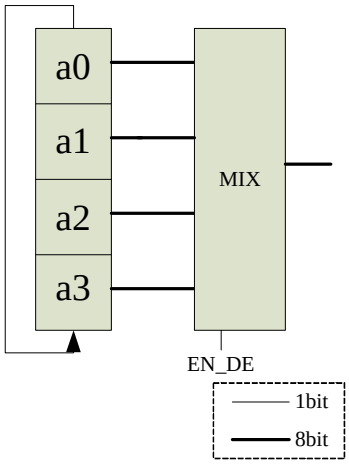


圖 4-7　8-bit AES InvMixcolumn(2)

接下來討論 MIX 的部份，由於進行混行轉換和反混行轉換時，可以運用資源共用的方式，達到節省資源的目的，我們可以得到以下推導，由 EN_DE 信號來控制，是要做加密（⊗ [02 03 01 01]），還是再加上（[08 08 08 08] + [04 00 04 00]）做解密的動作。

$$[0e \quad 0b \quad 0d \quad 09]=[02 \quad 03 \quad 01 \quad 01]+[08 \quad 08 \quad 08 \quad 08]+[04 \quad 00 \quad 04 \quad 00]$$

$$[0e \quad 0b \quad 0d \quad 09]=[02 \quad 03 \quad 01 \quad 01]+[12 \quad 08 \quad 12 \quad 08]$$

$$[02 \quad 03 \quad 01 \quad 01]=[00 \quad 01 \quad 01 \quad 01]+[02 \quad 02 \quad 00 \quad 00]$$

我們令輸入為$[a_0\ a_1\ a_2\ a_3]^T$，進行混行轉換時輸出為 b_0，在進行反混行轉換時之輸出為 b_0'，我們整理兩種化減方式如下。圖 4-7、4-8 為此兩種方式之架構圖。

$b_0 = \{02\} \otimes (a_0 + a_1) + (a_1 + a_2 + a_3)$

$b_0' = b_0 + \{08\} \otimes (a_0 + a_1 + a_2 + a_3) + \{04\} \otimes (a_0 + a_2)$

或

$b_0 = \{02\} \otimes (a_0 + a_1) + (a_1 + a_3) + a_2$

$b_0' = b_0 + \{12\} \otimes (a_0 + a_2) + \{08\} \otimes (a_1 + a_3)$

其中⊗{02}、⊗{04}、⊗{08}、⊗{12}可由表 4-1 得知。令輸入為一個係數為 0 或 1 的多項式：$a(x) = a_7 x^7 + a_6 x^6 + a_5 x^5 + a_4 x^4 + a_3 x^3 + a_2 x^2 + a_1 x^1 + a_0 x^0$ 。

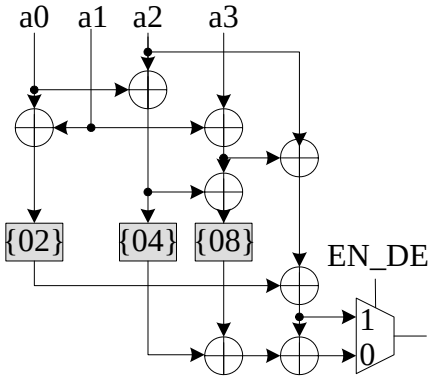


圖 4-8　MIX 架構圖一

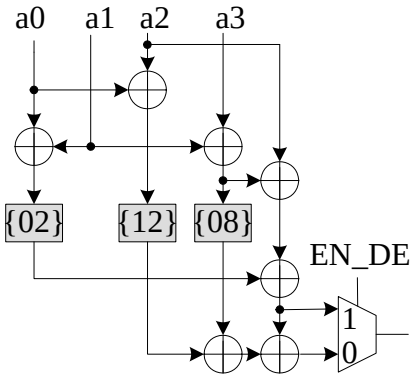


圖 4-9　MIX 架構圖二

表 4-1　⊗{02}、⊗{04}、⊗{08}、⊗{12}涵式表

涵　　式 (Function)	輸出
⊗{02}	$a_6 x^7+ a_5 x^6+ a_4 x^5+ (a_3+a_7) x^4+$ $(a_2+a_7) x^3+ a_1 x^2+ (a_0+a_7) x^1+ a_7 x^0$
⊗{04}	$a_5 x^7+ a_4 x^6+ (a_3+a_7) x^5+ (a_2+a_6+a_7) x^4+$ $(a_1+a_6) x^3+ (a_0+a_7) x^2+ (a_6+a_7) x^1+ a_6 x^0$
⊗{08}	$a_4 x^7+ (a_3+a_7) x^6+ (a_2+a_6+a_7) x^5+ (a_1+a_5+a_6) x^4+$ $(a_0+a_5+a_7) x^3+ (a_6+a_7) x^2+ (a_5+a_6) x^1+ a_5 x^0$
⊗{12}	$(a_4+a_5)x^7+(a_3+a_4+a_7)x^6+(a_2+a_3+a_6)x^5+$ $(a_1+a_2+a_5+a_7)x^4+ (a_0+a_1+a_5+a_6+a_7) x^3+$ $(a_0+a_6) x^2+ (a_5+a_7) x^1+ (a_5+a_6) x^0$
輸入為 $a(x)= a_7 x^7+ a_6 x^6+ a_5 x^5+ a_4 x^4+ a_3 x^3+ a_2 x^2+ a_1 x^1+ a_0 x^0$	

第四節　8-bit AES 金鑰擴展(Key expansion)

金鑰擴展是獨立的一部分，在進行 AES-128 解密時，必須先產生完整的 10 把金鑰，在依相反的順序使用，達到解密的效果。在本研究中，所設計為 8-bit 資料路徑，因此 128-bit 之金鑰，必須經過十六個週期（CLKs）才能產生下一把金鑰；而每一把金鑰的產生，只跟前一把金鑰有密不可分的關係。

金鑰擴展之動作重點說明如下：

步驟一、輸入金鑰為 128-bit(w0~w15 共 16-Bytes)，Nr=10，Nk=4；

下一把金鑰為（w0'~w15'共 16-Bytes）。

步驟二、當輸入為 w12~w15 時，必須經過線性位移轉換

（RotWord）、狀態值替換（SubWord）、加上回合常數（Rcon）、以及互斥或（XOR）等運算，來產生 w0’~w3’，如下圖 4-10 所示。

步驟三、而 w4’~w15’只要 w[i-4]與 w[i-16]（前第 16-Byte 金鑰）進行互斥或（XOR）運算即可，如圖 4-11。

步驟四、當 w0’~w15’都完成時，就可以開始擴展下一把 128-bit 金鑰，直到共 11 把金鑰為止。

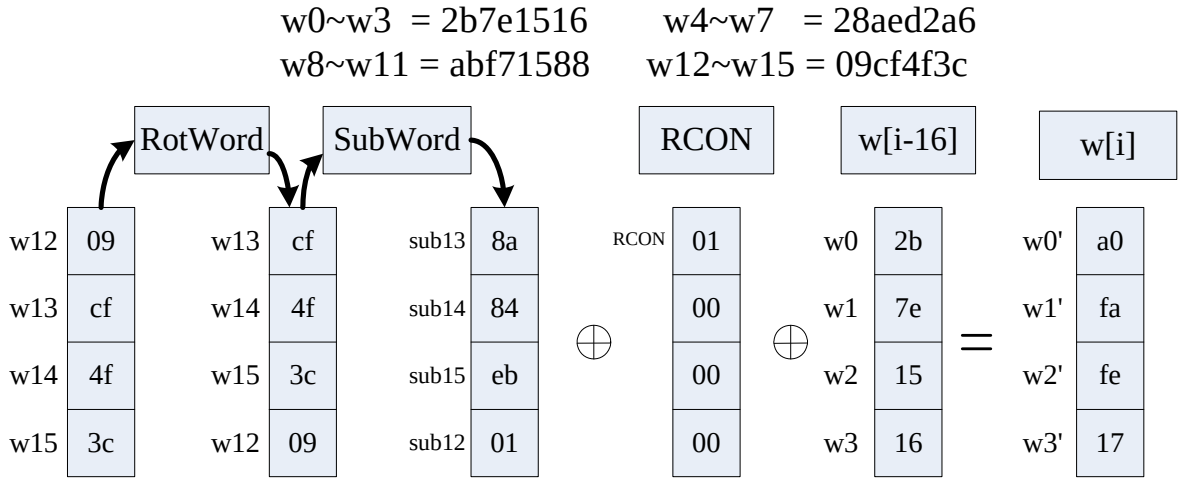


圖 4-10　金鑰擴展之 RotWord/ SubWord/ Rcon/ XOR

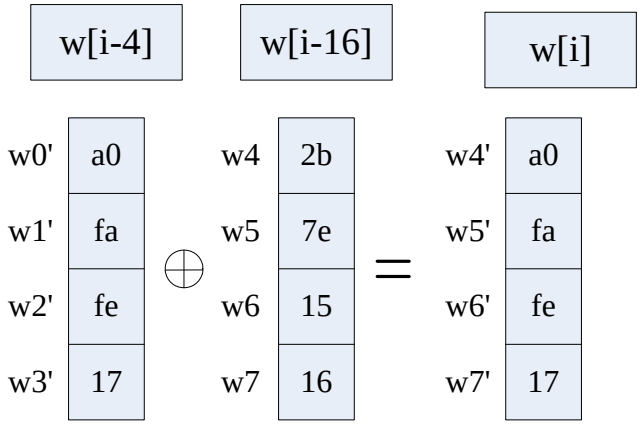


圖 4-11　金鑰擴展之 w[i]=w[i-4] xor w[i-16]

在本研究中，為了實現 8-bit AES 之金鑰擴展，也為了達到小面積之目的，實驗了以下兩種架構，一是使用移位暫存器（圖

4-12)，一是使用 Dual port BRAM（圖 4-13)。

圖 4-12，如上述之金鑰擴展之動作可知，w12~w15 需進行 SubWord 之動作，且 w13 必須進行加上回合常數(Rcon)之運算，如圖 4-12 可知，使用信號線 c1 來控制暫存器（sub12~sub15）之寫入。由於是使用移位暫存器，所以 w[i-4]之資料會在 w12 暫存器上，而 w[i-16]則會在 w0 暫存器上；信號線 c0 則用來選擇要進行金鑰擴展之步驟二或步驟三。

第二種架構為 Dual port BRAM，如圖 4-13 所示，為了減少 Slice 之使用，將第一種架構中之移位暫存器、以及暫存器兩部分，使用 Dual port BRAM 來替換。PortA0、PortA1 用來完成寫入資料的動作，而 PortB0 則用來讀出步驟三 w[i-16]之資料，PortB1 用來讀出 w[i-4]之資料，以及 sub12~sub15。cc、c2 信號線用來選擇寫入 RAM1 之資料包括 sub12~sub15、w0'~w12' 。

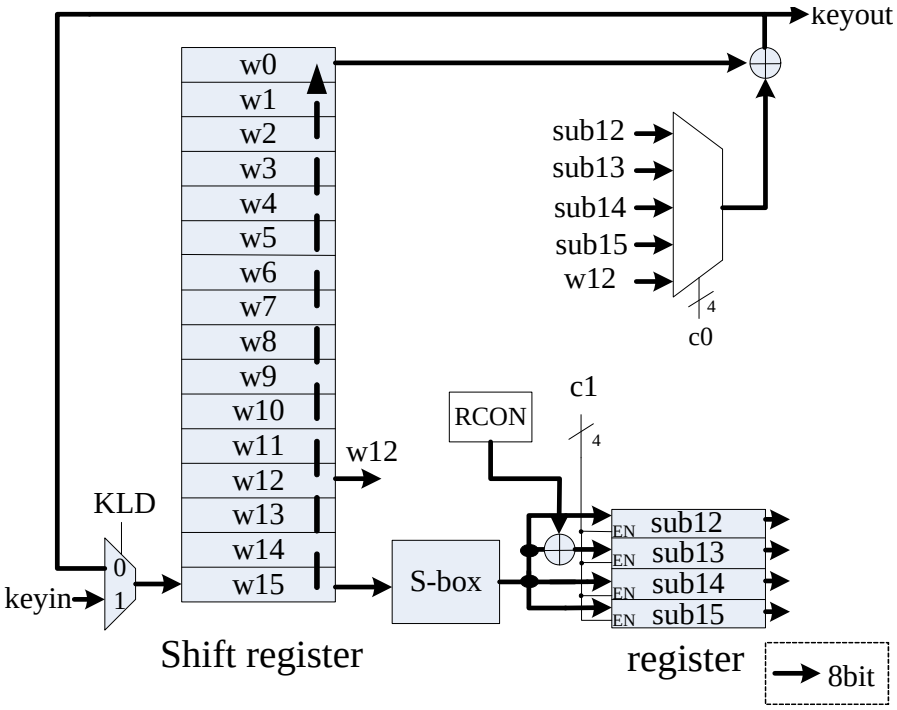


圖 4-12　金鑰擴展-移位暫存器架構

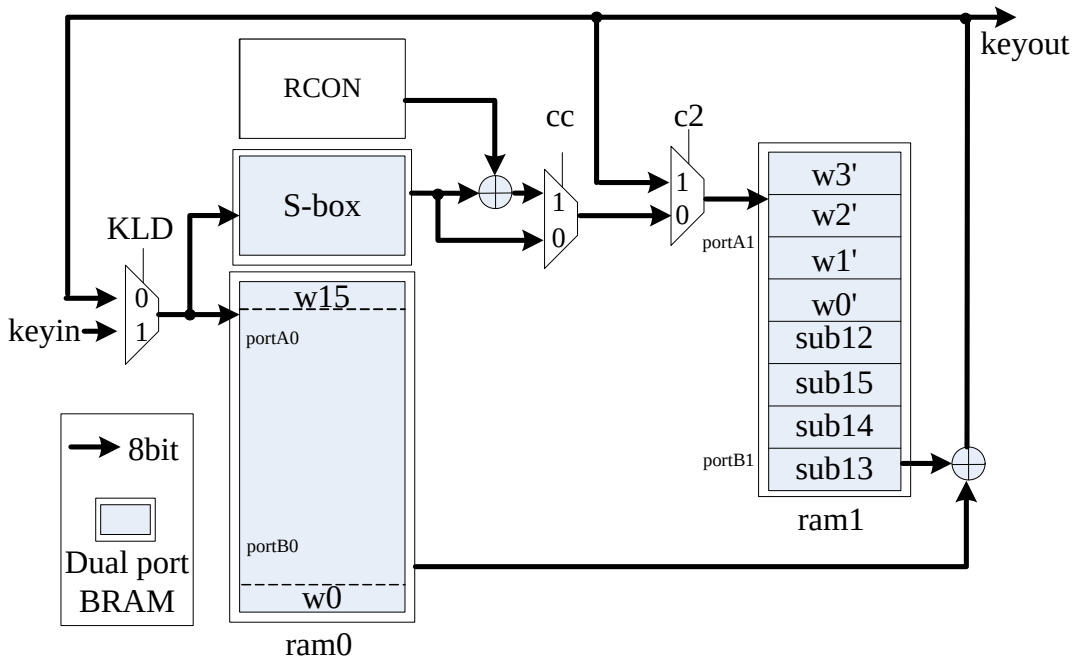


圖 4-13　金鑰擴展-使用 Dual port BRAM

第五節　8-bit AES 加密架構

在上述章節，已經介紹各類運算之方式及架構，在本節將會提出不同架構來完成 8-Aes 加密器。Rijndale 加密演算法需 SubByte、ShiftRow、MixColumn、及 AddRoundKey 四步驟。下圖 4-14 為 8-bit AES 加密架構一，是由多工器來完成 MixColumn 的動作，如第三節所闡述。此架構由信號線 c 來控制資料寫入的位址，c4 則用來決定 ShiftRow 動作。

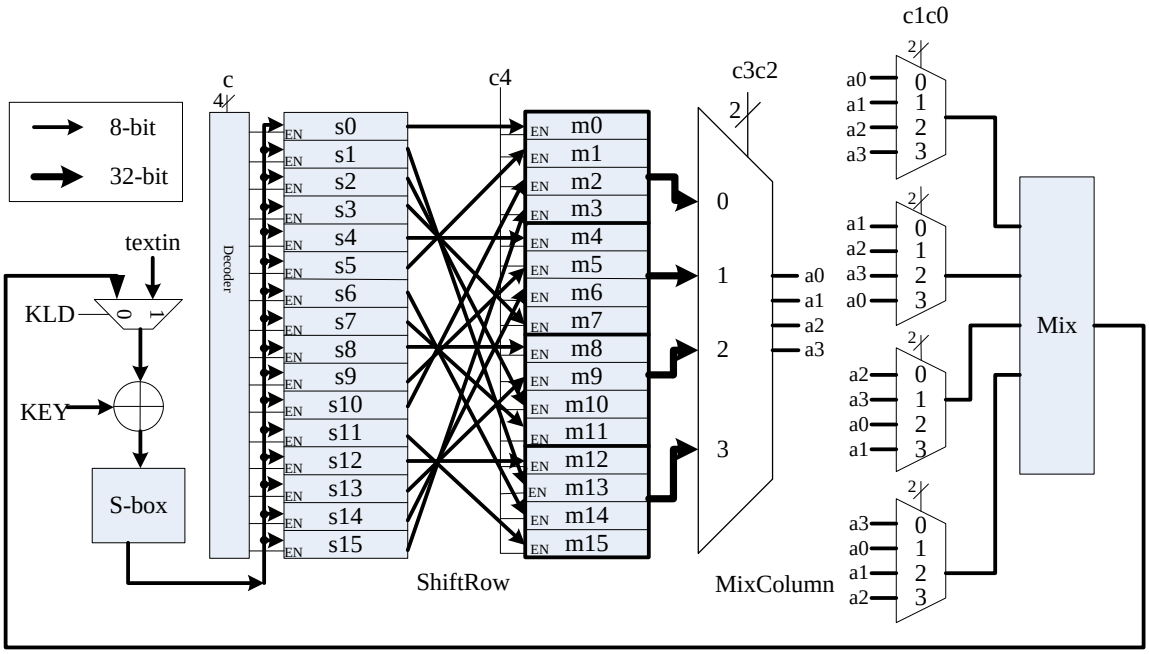


圖 4-14　8-bit AES 加密架構一

8-bit AES 加密架構二，如圖 4-15，用左旋移位暫存器（rotate left register）和移位暫存器（Shift register），來代替多工器以及解碼器（decoder）。信號線 EN 用來決定何時進行移位轉換（ShiftRow），將資料寫入 32-Bit 之 m0、m1、m2、m3；再由 c5 選擇資料寫入左旋移位暫存器（m0）。

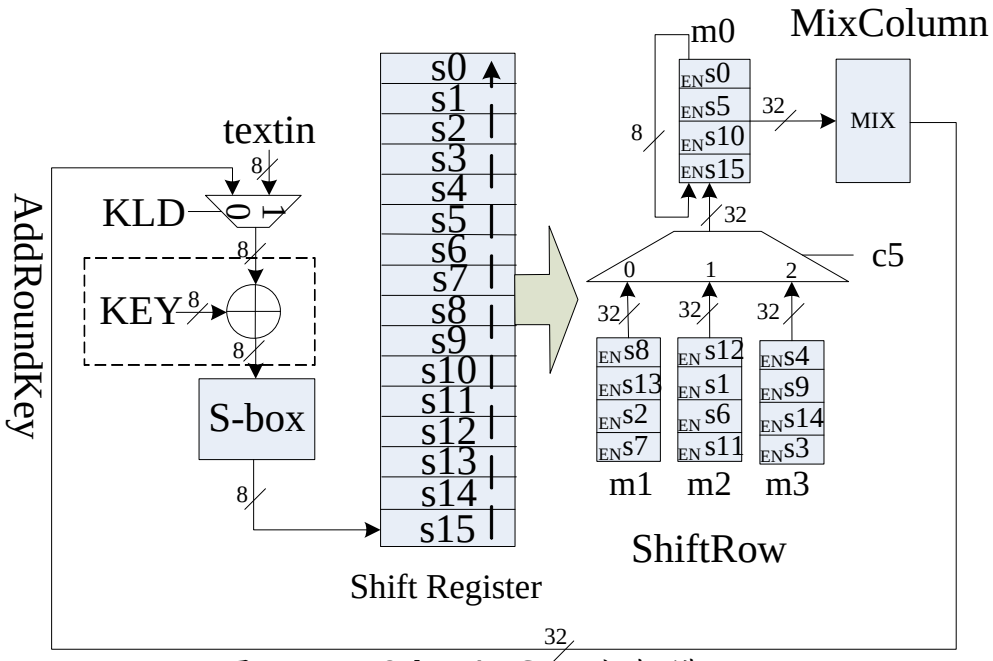


圖 4-15　8-bit AES 加密架構二

8-bit AES 加密架構三，如圖 4-16，使用 Dual port BRAM 來代替架構二之移位暫存器，如第二節圖 4-5 所示；但是為了使混行運算順利進行，必須增加 4-Byte 移位暫存器。PortA 用來寫入資料至 BRAM，PortB 則用來讀出資料至 4-Byte 移位暫存器，來達成移位運算。

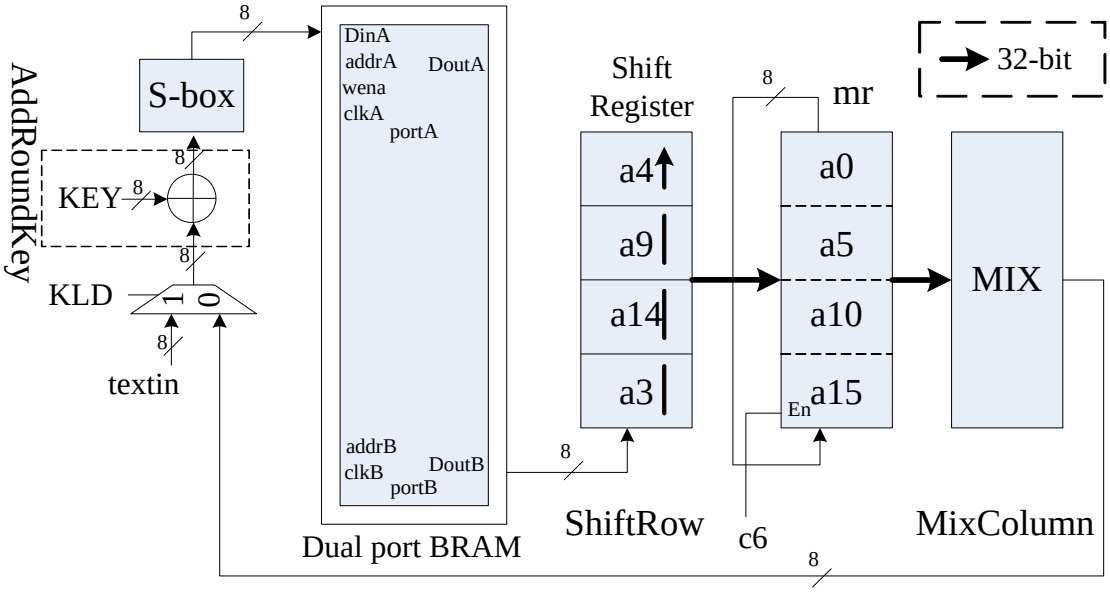


圖 4-16　8-bit AES 加密架構三

第五章　研究成果

將 128-bit 資料路徑轉換成 32-bit 或 8-bit 時，雖然在位元組轉換（SubBytes）及混行轉換（MixColumn）節省了資源，但是在移列轉換（ShiftRow）將產生額外的暫存器（Register）或記憶體（BRAM）來儲存資料，而且增加了控制上的複雜度，相對的面積也會跟者增加一些。

使用雙埠記憶體（Dual port BRAM），可以節省移位轉換（ShiftRow）時多工器（Multiplexer）之使用量，只需改變輸入之記憶體位址，就可以達到移列轉換（ShiftRow）和反移列轉換（InvShiftRow）之需求，達到節省資源之目的。

接下來各節中，將會把實驗結果與數據測試如下。

第一節　金鑰擴展設計與測試結果

在本節中，將測試金鑰擴展單元之功能與結果比較。金鑰擴展的正確數據如下表 5-1，在本研究中測試的金鑰長度為 128-bit。

如下圖 5-1 至圖 5-2，為金鑰擴展之結果，當 kld 為‘1’時，將第一把金鑰讀入；kld 為‘0’時，開始動作。信號線 count0 用來計算金鑰產生的順序，而 douta_k0 則為新產生的金鑰，每一筆輸入資料長度為 8-bit，共 128-bit；因此會產生另外 10 把金鑰，共花費 160 個週期（CLKs）才完成。

輸入金鑰：0102030405060708090a0b0c0d0e0f

由表 5-2 金鑰擴展架構比較表可以發現，移位暫存器架構雖然有較高的 Slice，但是其工作頻率（Frequency）也相對較高。相對於 Dual port BRAM 架構，雖然其擁有較低的 Slice，但工作

頻率減少許多。原因可能出在於 Dual port BRAM 的位址(address)控制上，讓 Slice 的減少並不如預期。

表 5-1　金鑰擴展資料表

金　鑰　序（NO.）	資料
KEY[0]	000102030405060708090a0b0c0d0e0f
KEY[1]	d6aa74fdd2af72fadaa678f1d6ab76fe
KEY[2]	b692cf0b643dbdf1be9bc5006830b3fe
KEY[3]	b6ff744ed2c2c9bf6c590cbf0469bf41
KEY[4]	47f7f7bc95353e03f96c32bcfd058dfd
KEY[5]	3caaa3e8a99f9deb50f3af57adf622aa
KEY[6]	5e390f7df7a69296a7553dc10aa31f6b
KEY[7]	14f9701ae35fe28c440adf4d4ea9c026
KEY[8]	47438735a41c65b9e016baf4aebf7ad2
KEY[9]	549932d1f08557681093ed9cbe2c974e
KEY[10]	13111d7fe3944a17f307a78b4d2b30c5

表 5-2　金鑰擴展架構比較表

	金鑰擴展- 移位暫存器架構	金鑰擴展- 使用 Dual port BRAM
Number of Slices	68	52
Number of BRAMs	1	3
Maximum Frequency (MHz)	119.033	73.795
（包含控制單元）		

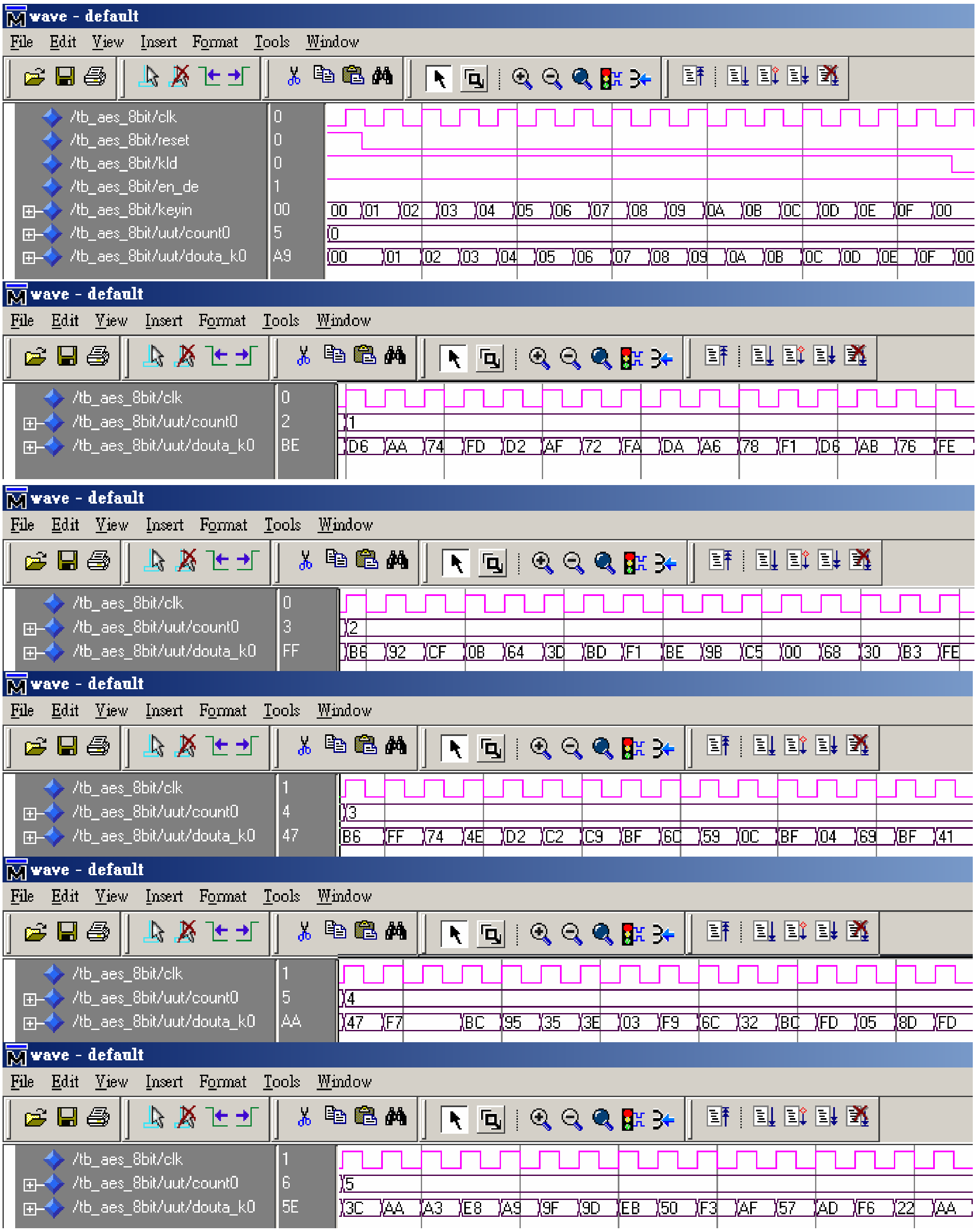


圖 5-1　金鑰擴展測試結果(1)

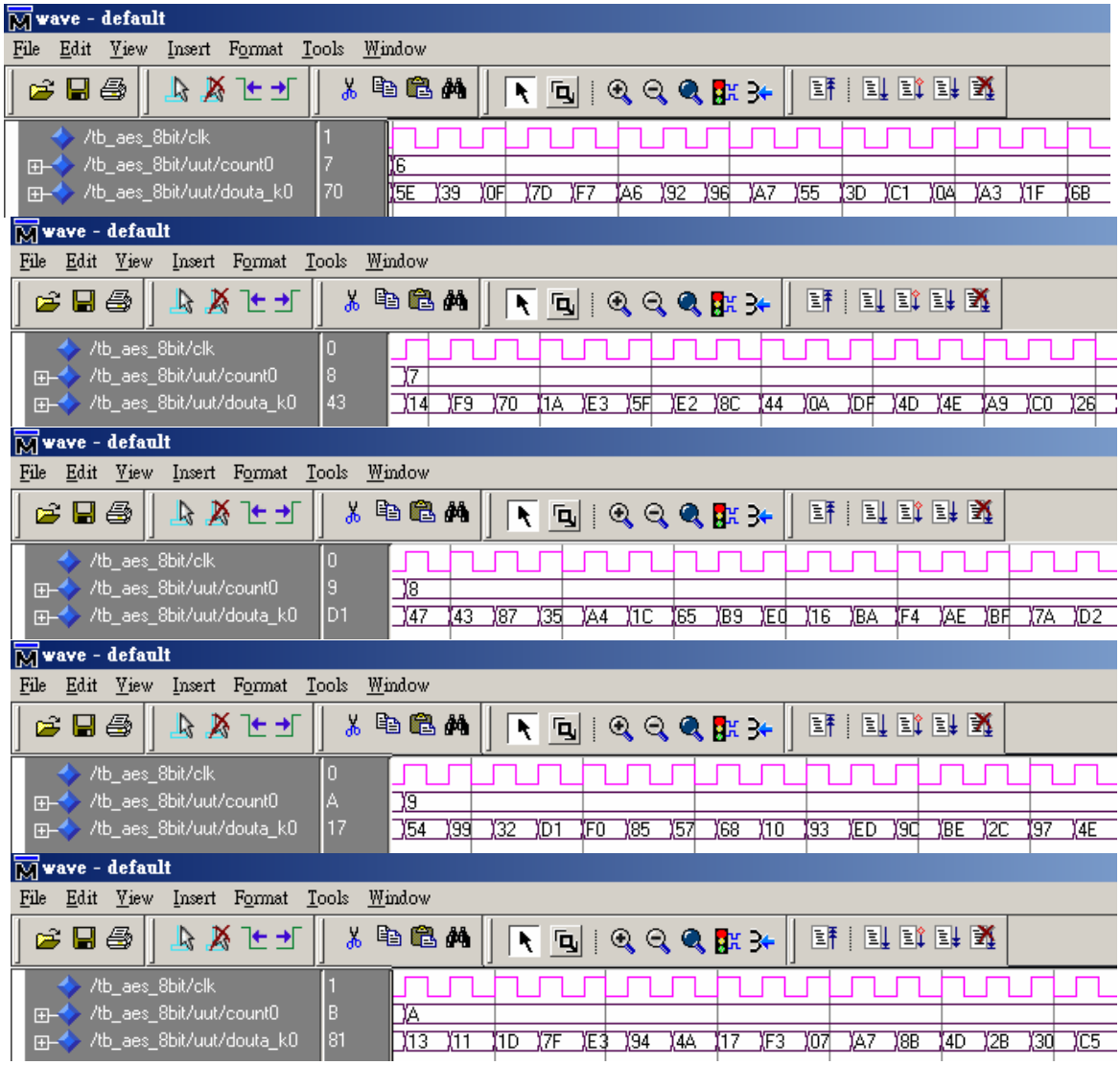


圖 5-2　金鑰擴展測試結果(2)

第二節　混行運算設計與測試結果

本研究在設計混行運算時，主要是使用左旋移位暫存器（rotate left register），來達到小面積之要求。而在 MIX 架構上，由於架構二擁有較低的延遲時間，且 Slice 相差不遠，所以在設計上採用架構二，如表 5-3 所示。圖 5-3 則為測試結果，輸出信號為 mixout。

表 5-3　MIX 架構比較表

	MIX 架構一 (圖 4-8)	MIX 架構二 (圖 4-9)
Number of Slices	24	24
Maximum combinational path delay(ns)	13.814ns	12.248ns

MixColumn（en_de='1'）		InvMixColumn（en_de='0'）	
輸入(a0 a1 a2 a3)	輸出	輸入(a0 a1 a2 a3)	輸出
D4 BF 5D 30	04	E9 F7 4E EC	54
E0 B4 52 AE	E0	02 30 20 F6	6B
B8 41 11 F1	48	1B F2 CC F2	96
1E 27 98 E5	28	35 3C 21 C7	A1

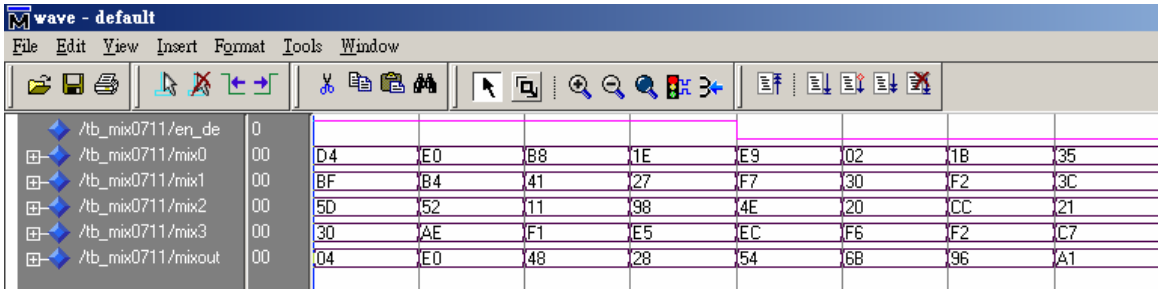


圖 5-3　MIX 測試結果

第三節　8-bit AES 加密器設計與測試結果

在這一節，本研究將比較第四節所提出之 AES 架構，如表 5-4 所示，由於 Xilinx Spartan2 xc2s15 這一塊晶片，只提供 192-Slices 以及四個 BRAM；因此為了實現 8-bit AES 加密器，必須選擇架構三之加密方式，當配合 Dual port BRAM 之金鑰擴展

來即時完成。而表 5-5，則是本研究跟 Tim Good[]之面積與產率（Throughput）比較之結果。在產率上擁有將近 15 倍之優勢。

表 5-6 為本研究之加密資料結果對照表，圖 5-5、圖 5-6 則為測試結果，在本研究中，我們測試兩筆資料，以驗證加密器之功能。

表 5-4　8-bit AES 合成結果

	8-bit AES 加密架構一	8-bit AES 加密架構二	8-bit AES 加密架構三
Number of Slices	234	184	76
Number of RAMs	1	1	2
Maximum Frequency (MHz)	103.125	71.225	47.228
Device	Xilinx Spartan2 xc2s15		

表 5-5　8-bit AES 比較表

	Tim Good[6]	Ours
Number of Slices	122	112
Number of RAMs	2	4
Maximum Frequency (MHz)	72.3	44.362
Throughput(Mbps)	2.18	33.40
Device	Xilinx Spartan2 xc2s15	
	smallest	More high throughput

表 5-6　加密資料結果對照表

第一筆資料	
明文	32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34
金鑰	2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c
祕文	39 25 84 1d 02 dc 09 fb dc 11 85 97 19 6a 0b 32
第二筆資料	
明文	00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff
金鑰	00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
祕文	69 c4 e0 d8 6a 7b 04 30 d8 cd b7 80 70 b4 c5 5a

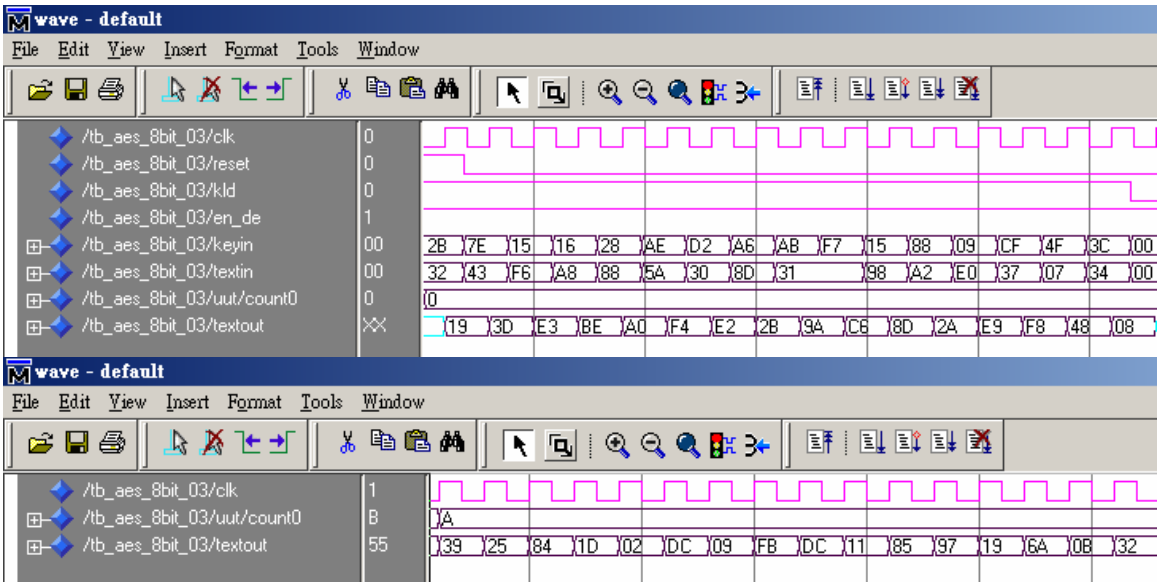


圖 5-5　8-bit AES 加密測試結果(1)

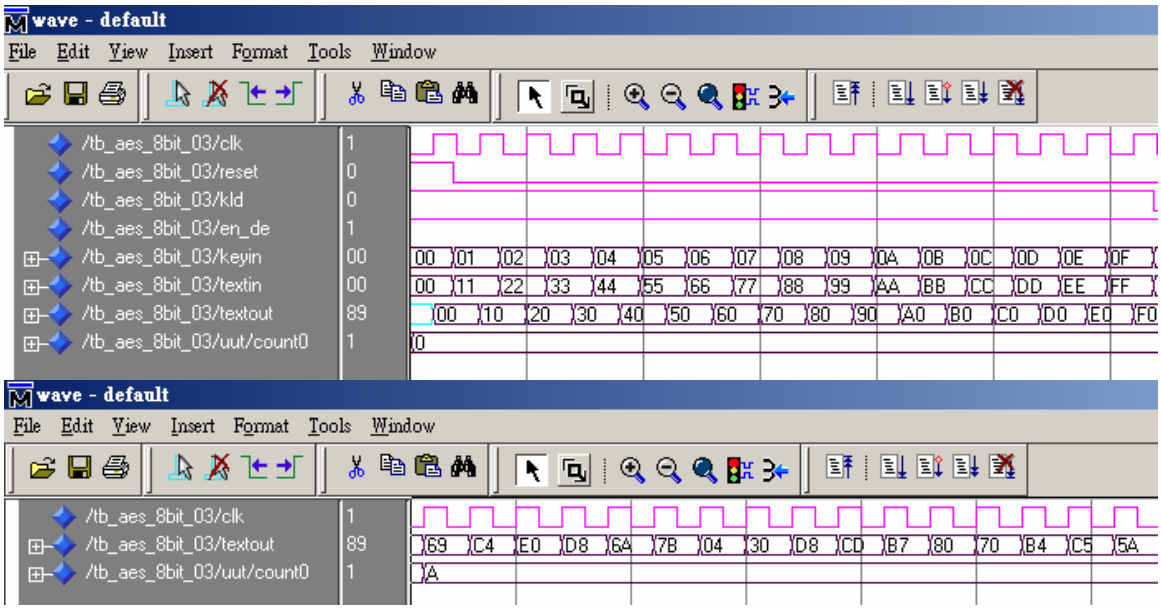


圖 5-6　8-bit AES 加密測試結果(2)

第六章 結論及未來工作

在本研究中，有效利用 FPGA 的特色，使用 BRAM 來減少 Slice 的使用量，可以將 AES 加密器放入 Xilinx Spartan2 (xc2s15) 晶片中。我們所使用的三種架構，當然也可以應用在解密架構中，未來可以將加解密功能，以資源共用的方式整合在一起，以利更廣泛的應用。

電路的功率消耗(power consumption)，在注重環保及省電的現今社會，已是不可避免的問題；尤其在可攜式的設備上，如行動電話以及 PDA…等，小面積以及省電的要求就更為重要。電路的功率消耗將是未來可以討論的地方。

參考文獻

[1]. Advanced Encryption Standard (AES) (in National Institute of Standards and Technology [NIST]), Federal Information Processing Standards (FIPS) Pub. 197, Nov. 2001.

[2]. P. Chodowiec and K. Gaj, “Very Compact FPGA Implementation of the AES Algorithm,” in Proc. LNCS’03, 2003, vol. 2779, pp. 319–333.

[3]. G. Rouvroy, F. X. Standaert, J. J. Quisquater, and J. D. Legat, “Compact and efficient encryption/decryption module for FPGA implementation of the AES Rijndael very well suited for small embedded applications,” in *Proc. ITCC’04*, Apr. 2004, vol. 2, pp. 583–587.

[4]. Ricardo Chaves, Georgi Kuzmanov, Stamatis Vassiliadis, and Leonel Sousa,“Reconfigurable Memory Based AES Co-Processor,” in Proc.IEEE 2006,

[5]. A. Satoh, S. Morioka, K. Takano, and S. Munetoh, “A Compact Rijndael Hardware ArchitectureWith S-Box Optimization,” in *Proc. LNCS ASIACRYPT’01*, Dec. 2001, vol. 2248, pp. 239–254.

[6]. Tim Good, Mohammed Benaissa,“Very Small FPGA Application-Specific Instruction Processor for AES,” IEEE transations on circuits and systems—I: Regular Papers, vol. 53, NO. 7, July 2006

[7]. C. Paar, “Efficient VLSI Architectures for Bit-Parallel Computation in Galois Fields,” Ph.D. dissertation, Inst. for

Experimental Mathematics, Univ. of Essen, Essen, Germany, Jun. 1994.

[8] X. Zhang and K. K. Parhi, “High-speed VLSI architectures for the AES algorithm,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 12,no. 9, pp. 957–967, Sep. 2004.

[9] D. Canright, “A very compact S-box for AES,” in Proc. Cryptographic Hardware and Embedded Syst., Edinburgh, U.K., Sep. 2005, pp. 441–455.

[10] J.Wolkerstorfer, E. Oswald, and M. Lamberger, “An ASIC implementation of the AES S-boxes,” in *Proc. RSA Conf.*, San Jose, CA, Feb. 2002, pp. 67–78.

[11] 瀨溪松、韓亮、張真誠，“近代密碼學及其應用”，旗標出版股份有限公司。