卒業論文

AES 暗号化回路の設計と FPGA ボードへの実装

氏　　名：橋爪　啓介
学籍番号：2260060081-1
指導教員：山崎 勝弘 教授
提 出 日：2010 年 2 月 18 日

立命館大学　理工学部　電子情報デザイン学科

内容梗概

本論文では、本研究室が開発を進めている AES 暗号アルゴリズムをガロア体（2^8）の知識を用いて VerilogHDL による独自ハードウェア設計を行い、設計した独自ハードウェアについて HDL シミュレーションを行い、FPGA ボードへ実装して動作検証を行ったことを述べてある。さらに多項式計算（例 $56_{16} \times 12_{16}$）も同様に行ったことも述べてある。

ガロア体（2^8）の知識を用いて AES 暗号化回路の設計を行う上で、ガロア体（2^8）を用いた演算方法と AES 暗号アルゴリズムを学習し、FPGA ボードへ実装して、どのようにすれば高速化できるかを検証することを本研究の目的とする。

目次

図目次

表目次

１　はじめに

AES 暗号は DES 暗号よりも世界的に安全性が認められた共通鍵暗号方式である。この２つの暗号方式の歴史は 1960 年代からの DES 暗号の開発、普及が進んでいったことから始まったとされている。DES 暗号は当時の予想を上回るインターネットの普及と解読技術の進歩と計算機そのものの演算スピードの向上によって、暗号としての寿命を縮めることになってしまった。DES 暗号が使えなくなり特に、松井充（三菱電機）の線形解読法が発表されたあたりから、次世代の標準暗号の必要性が求められるようになり、1997 年米国商務省標準局（NIST）により、AES(Advanced Encryption Standard)の仕様が発表され全世界から次世代暗号の候補が公募され、1999 年の時点で 15 の候補から 5 つの候補に絞られた。最終的に 2000 年 8 月ベルギーの二人の青年、ヨハン・デーセン（J. Daemen）とヴィンセント・ライマン(V. Rijnmen)が提案したラインデール（Rijndael)が AES 暗号として採用され、現在に至る。AES 暗号と DES 暗号の違いはデータ変換部で異なっているが、S-box と呼ばれる数表を用いたデータ生成法の鍵スケジュール部においては DES 暗号と AES 暗号はよく似ているとされている。また、Rijndael はデータのブロックサイズ、鍵サイズともに 128 ビット、192 ビット、256 ビットの 3 種類ずつ独立に選択できるようになっている。ちなみに FIPS 197 の規格としては 192、256 ビットのブロック長は削除されている。このように AES 暗号は DES 暗号より世界的に安全性の面で認められていて、さらなる安全性が求められる。

以上のような背景より、本研究は有限体ガロアフィールド体（2^8）の知識を用いて、多項式計算、AES 暗号アルゴリズムの中の主に MixColumns 変換、ShiftRows 変換の VerilogHDL による独自ハードウェア設計を行い、設計した独自プロセッサについて HDL シミュレーションを行い結果が正しいかどうか確かめた。最後に設計したプログラム（多項式計算、MixColumns 変換）を FPGA ボード上に実装して、動作を確認することによりどのようにすれば高速化できるかを検証し、さらなる高速化するためにはどのようにすればよいかをこの研究の目的としている。

また実際に Verilog-HDL を用いたトップダウン設計を行うことで、HDL と高位合成ツールやシミュレーターの習得、プロセッサとその周辺モジュールのアーキテクチャの理解を目的とする。さらに、実際にプロセッサ検証システムとして FPGA ボード上へ実装することで、プロセッサを含むデジタルシステム全体のアーキテクチャの理解を深め、FPGA ボードの利用方法の理解に努める。

FPGA(Field Programmable Gate Array)ボードとは内容（内部ロジック）を書き換えられる LSI である。基本素子は SRAM で、論理ブロックをアレイ状に敷き詰めてあり、ユーザの自作したシステムや回路のデータをダウンロードすることによって回路構成を構築・変更することができる。この書き換え可能な LSI を使うことで実機に近い検証を手軽に行うことができる。CPLD という EEPROM をベースとしたプログラマブル・ロジック・デバイスもあるがゲート規模が小さく、現在では FPGA ボードにほとんどのシェアを奪われて

いる。最近は FPGA ボードの低価格化や大容量化も進んでおり 1000 万ゲートを越えるものも出てきていて、実際に製品に搭載されることもあるなど、その重要性はますます高まっていっている。

本論文では、第２章でガロア体（2^8）と AES 暗号アルゴリズムの各々について説明する。第三章では設計した多項式計算、MixColumns 変換、ShiftRows 変換の設計について説明する。第４章では FPGA ボードへの実装で得られた結果の評価について述べる。

２　暗号アルゴリズム

２．１　有限体 GF（２^{8}）

GF(2)は１ビットの演算で、加算を排他論理和（XOR）で行う。これによって GF(2)は 0～1 の集合になる。GF(2^8)は GF(2)を拡張したもので、ほぼ８ビット（１バイト）のビット演算である。ただし、適切な既約多項式 p(x)を決めてあって、ビットがあふれてしまったときの処理を定めてある。GF(2^8)の規約多項式 p(x)は　ｐ（ｘ）＝ｘ^{8}＋ｘ^{4}＋ｘ^{3}＋ｘ^{2}＋１である。ガロア体の定義は p(x) = 0 なので、以下の関係が成り立つ。

ｐ（ｘ）＝ｘ^{8}＋ｘ^{4}＋ｘ^{3}＋ｘ^{2}＋１＝０より

ｘ^{8}＝－ｘ^{4}－ｘ^{3}－ｘ^{2}－１

１＋１＝０　より

１＝－１　よって

ｘ^{8}＝ｘ^{4}＋ｘ^{3}＋ｘ^{2}＋１

つまり演算によってビットがあふれ、ビット８が１になってしまったとき、その値は下位の８ビット（$x^4 + x^3 + x^2 + 1$）に置き換えられることを意味する。そして下位８ビットとの加算によって８ビット幅の値を得られる。ここでの加算は排他的論理和で行うので、演算結果が８ビット幅を超えることはない。

演算方法　多項式計算

多項式計算の演算方法は AES 暗号化回路を設計するときに用いられる重要な演算方法である。多項式ｆ（ｘ）で変数は ｘ しかないので、式の係数は $a_7,a_6,a_5,a_4,a_3,a_2,a_1,a_0$ と表す。係数は０と１しかない値を持ち、ｎビット（２^{8}なら８ビット）のデータになる。GF(2^n)である多項式すべては、ｎビットのデータで表すことができる。

加法：GF(2^n)で行う加法は XOR 演算となる。

乗法：多項式 f(x)×ｘに注目する。分かりやすく GF（２^{8}）での乗算と既約多項式ｐ（ｘ）＝ｘ^{8}＋ｘ^{4}＋ｘ^{3}＋ｘ^{2}＋１を利用して、図１に示す。

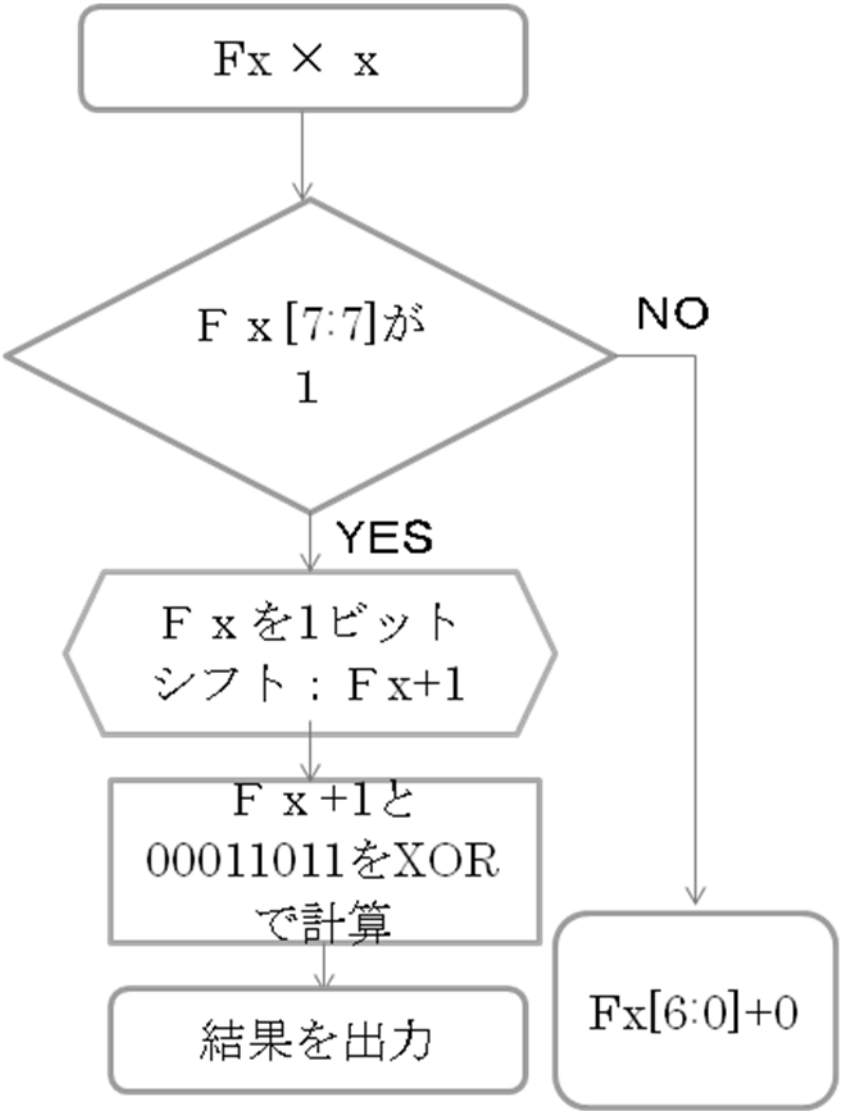

図１　f(x)×x のフローチャート

多項式計算例 $56_{16}\times12_{16}$ の演算方法を以下の手順に示す。

①$12_{16}$ を２進数に変換する・・・00010010_2

②それを X の式に変換する・・・X^4+X

③$56_{16}$ に X^4+X をかける・・・$56\ X^4+56X$

④$56_{16}$ を２進数に変換しておく・・・01010110_2

⑤まず図１を用いて 56X を計算する

⑥$56X\times X, 56X^2\times X, 56X^3\times X$ を同様の方法で計算する

⑦56X と $56X^4$ を XOR する・・・BB_{16}

この演算方法は MixColumns 変換の列演算（$S_{00}\times02_{16}$ など）で用いられる

２．２　AES 暗号アルゴリズム

図２は AES の暗号化の流れを示しており 128 ビットのデータに対応している。AES の入力データはバイトの行列になり、状態の行列に対応して処理する。ラウンドは１から 10 までありラウンド１からラウンド９までは同様に S-BOX 変換（SubBytes 変換）、ShiftRows 変換、MixColumns 変換、addkey 変換（AddRoundKey 変換）の４つの状態の行列で処理をして、最後のラウンド１０では S-BOX 変換（SubBytes 変換）、ShiftRows 変換、MixColumns 変換の３つの最後の状態行列から暗号文を出力する。

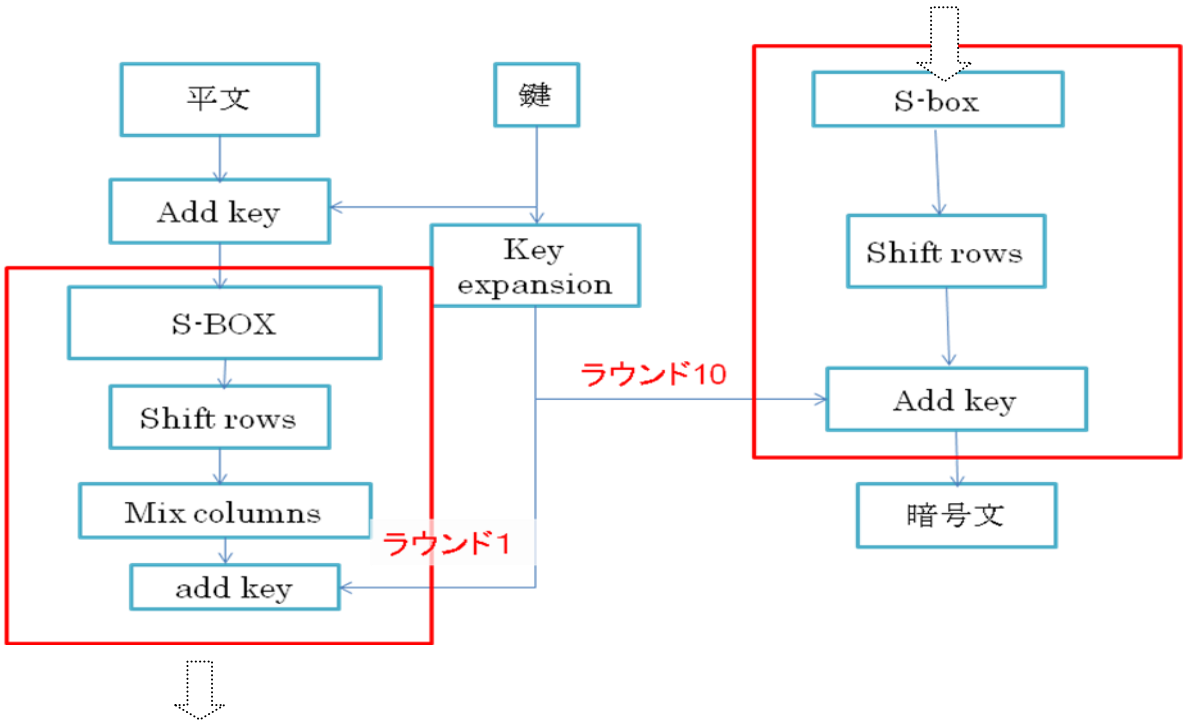

図２　AES の暗号化

（1） KeyExpansion

鍵を拡張する KeyExpansion は処理の手順が一番煩雑である。本研究では鍵長を 128 ビットに限定しているので、それを中心に話を進める。

鍵データの扱い方は 1 バイトを 1 ブロックとして列単位で見ている。ラウンドごとに 128 ビット拡張するので、128 ビットごとに鍵を RoundKey として分ける。図 3 は鍵拡張の例を示す。

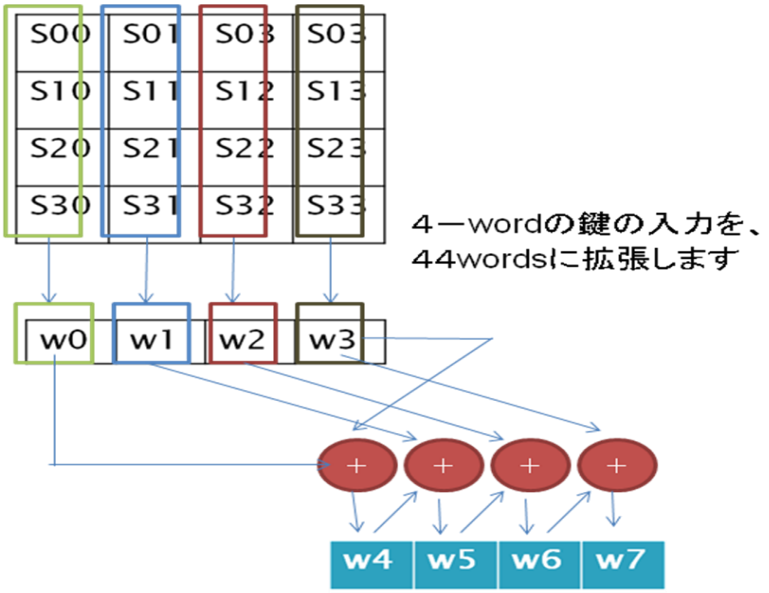

図 3　KeyExpansion 変換

（2）SubBytes 変換

SubBytes 変換は図 4 に示している通り入力されてきた被変換データを表 1 の Sbox というテーブルを参照して変換するものである。この処理は列単位で行う。

表 1 の Sbox テーブルは各ブロックにおいて、値を上位 4 ビットと下位 4 ビットに分解し、Sbox[上位 4 ビット ，下位 4 ビット]としてテーブルとして参照する。

例えば、16 進数の {95} を SubBytes 変換したら、表 1 の Sbox にある行 x ＝9 と列 y ＝5 の要素の値は {2a} になるので、 {95} の SubBytes 変換の結果は {2a} になる。

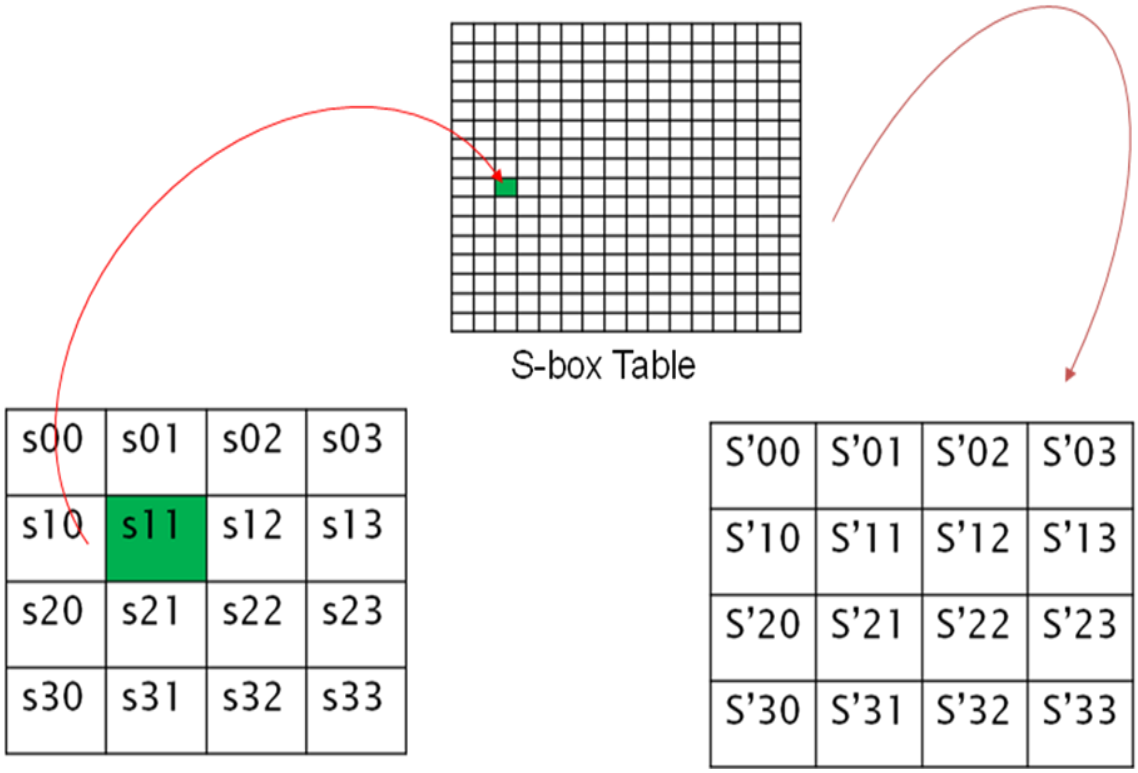

図 4　SubBytes変換

表 1 : S-box table

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

（3）Mixcolumns 変換

MixColumns 変換は入力されてきた被変換データを有限体理論に基づく演算によって変換する。処理単位は列である。

ここでは有限体理論を用いることになるが、MixColumns 変換を使う場合には注意する点は絞られる。まず、MixColumns 変換は次の行列をかけて計算することで結果を導ける。式内の右辺にある列ベクトルは被変換データで左辺にある列ベクトルは変換後データの列（4 ブロック）、行列の数値は 16 進数である。

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}$$

上記の行列を用いて乗算を行えば一列分の変換データを導出できるが、そのための積和演算が有限体理論に準じているので注意する必要がある。有限体理論での積和演算は以下の項目についてのみ考慮すればよい。

・ フィールドでの加算は排他的論理和である。
・ 乗算は図５の通り

(1)S20x01,S30 x01 との乗算は被乗数の値のままである。
(2) S00x02 との乗算は、被乗数が S00x08 未満なら被乗数を 1bit 左シフトした値で、被乗数が S00x08 以上なら 1bit 左シフトした値に S00x1b を EXOR 演算した値。
(3) S10x03 との乗算は、(被乗数×01+被乗数×02)に分配できる。

この有限体理論で注目するのは乗算部分での、S20x01, S30x01 と S00x02, S10x03 の 2 通りの処理パターンがあることに注意する。この 2 通りの処理はモジュールでは場合分けに相当する。この乗算を行った後に総和を求める。図６に MixColumns 変換の例を示す。

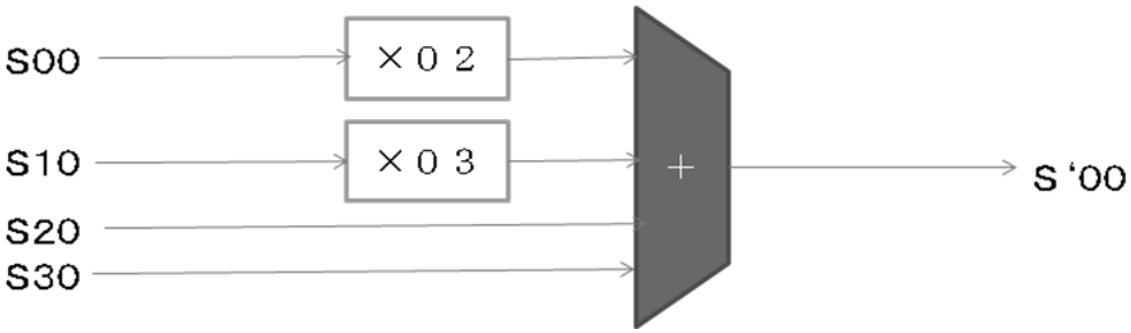

図５　MixColumuns 変換

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

47	40	A3	4C
37	D4	70	9F
94	E4	3A	42
ED	A5	A6	BC

図６　MixColumns変換の例

（４）ShiftRows 変換

ShiftRows 変換は入力されてきた被変換データを図７の通り行ごとにシフトする。注意する点は今まで列ごとにデータを扱ってきたが、ShiftRows 変換は行単位でデータを扱うことと、行によってシフトする量が異なることである。シフト量は n 行目では(n-1)ブロック（バイト）分だけであり、シフト方向は左である。図８に ShiftRows 変換の例を示す。

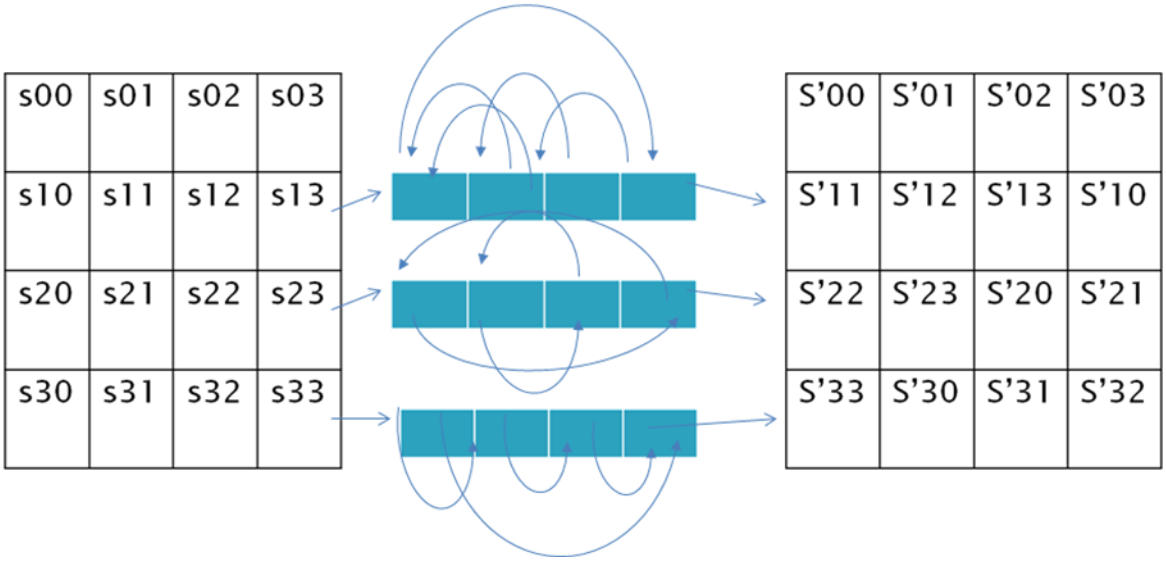

図７　ShiftRows 変換

87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

→

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

図８　ShiftRows 変換の例

（５）Add Round Key 変換

AddRoundKey では鍵拡張部で拡張された鍵の中から１ラウンド分の鍵を持ってきて入力の statemt と排他的論理和(XOR)を出力する。図９にそれらを示す。

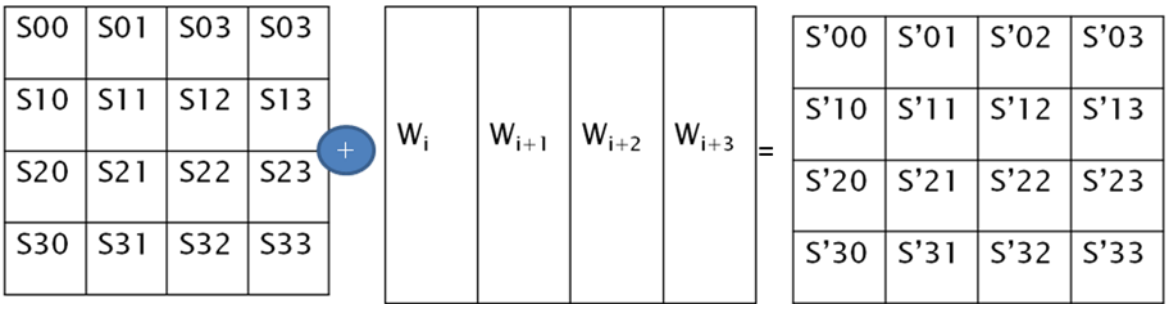

図９　Add Round Key 変換

（６）その他－AES_Loop, AES_fullHW, AES_ss

AES_Loop、AES_fullHW は AES 暗号システムを全てハードウェア化する際に使用する。AES_fullHW はトップモジュールである。ここではデータ入力直後にある AddRoundKey 変換と次にある AES_Loop 部分を接続・管理する役割を担っている。

AES_Loop は図２のように AES 暗号化の SubBytes→ShiftRows→MixColumns→AddRoundKey のループ部分を制御するモジュールに相当する。そのための制御部も含んでいる。また、ループ脱出後にあるラウンド１０の SubBytes→ShiftRows→AddRoundKey という変換も兼ねている。

AES_ss は上記 2 つのモジュールとは別のものであり、128 ビットの暗号化を最小でシンプルな回路として実現できるかを目的としたシステムである。AES_ss の”ss”は”simplest system”を表す。

３　AES 暗号アルゴリズムの設計

３．１多項式計算の設計

多項式計算の設計は前節で述べたように有限体ガロアフィールド体（2^8）の知識を用いて、２．１の手順に従って設計する。

ｈｘ＝fx×gx を計算する回路を図 10 に示す。また Verilog-HDL コードを図 11 に示す。

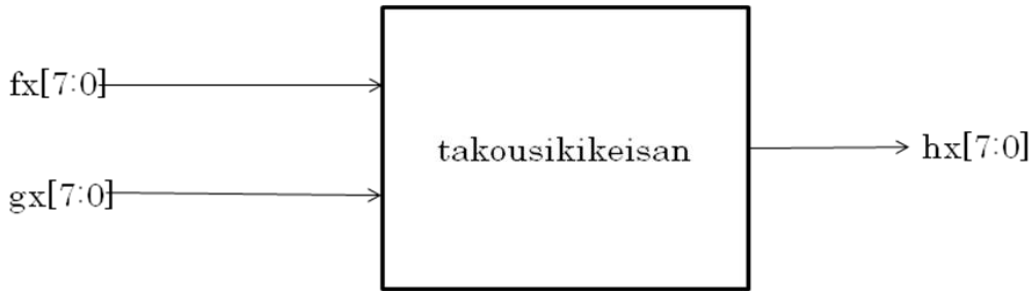

図１０　多項式計算の入出力仕様

```
module takousikikeisan(fx,gx,hx);
input [7:0] fx;
input [7:0] gx;
output [7:0] hx;
wire [7:0] f0;
wire [7:0] f1;
wire [7:0] f2;
wire [7:0] f3;
wire [7:0] f4;
wire [7:0] f5;
wire [7:0] f6;
wire [7:0] f7;

assign f0 = fx;
assign f1 = (f0[7:7]) ? {f0[6:0],1'b0}^8'h1B : {f0[6:0],1'b0};
assign f2 = (f1[7:7]) ? {f1[6:0],1'b0}^8'h1B : {f1[6:0],1'b0};
assign f3 = (f2[7:7]) ? {f2[6:0],1'b0}^8'h1B : {f2[6:0],1'b0};
assign f4 = (f3[7:7]) ? {f3[6:0],1'b0}^8'h1B : {f3[6:0],1'b0};
```

```
  assign f5 = (f4[7:7]) ? {f4[6:0],1'b0}^8'h1B : {f4[6:0],1'b0};
  assign f6 = (f5[7:7]) ? {f5[6:0],1'b0}^8'h1B : {f5[6:0],1'b0};
  assign f7 = (f6[7:7]) ? {f6[6:0],1'b0}^8'h1B : {f6[6:0],1'b0};

  assign hx = (f7 & {gx[7:7],gx[7:7],gx[7:7],gx[7:7],gx[7:7],gx[7:7],gx[7:7],gx[7:7]}) ^
              (f6 & {gx[6:6],gx[6:6],gx[6:6],gx[6:6],gx[6:6],gx[6:6],gx[6:6],gx[6:6]}) ^
              (f5 & {gx[5:5],gx[5:5],gx[5:5],gx[5:5],gx[5:5],gx[5:5],gx[5:5],gx[5:5]}) ^
         (f4 & {gx[4:4],gx[4:4],gx[4:4],gx[4:4],gx[4:4],gx[4:4],gx[4:4],gx[4:4]}) ^

       (f3 & {gx[3:3],gx[3:3],gx[3:3],gx[3:3],gx[3:3],gx[3:3],gx[3:3],gx[3:3]}) ^
       (f2 & {gx[2:2],gx[2:2],gx[2:2],gx[2:2],gx[2:2],gx[2:2],gx[2:2],gx[2:2]}) ^
              (f1 & {gx[1:1],gx[1:1],gx[1:1],gx[1:1],gx[1:1],gx[1:1],gx[1:1],gx[1:1]})
^
              (f0 & {gx[0:0],gx[0:0],gx[0:0],gx[0:0],gx[0:0],gx[0:0],gx[0:0],gx[0:0]});



  endmodule
```

図１１　多項式計算の Verilog-HDL 記述

図 11 の 13 行目から 20 行目は図 1 のフローチャート通りに記述した。それぞれ 8 ビットのｆ 0 からｆ 7 を計算するようにプログラミングした。また 23 行目から 31 行目では演算結果 hx を求めた f0 から f7 をそれぞれ gx で掛け算して最後にそれら全てを XOR するようにプログラミングした。

３．２　MixColumuns の設計

MixColumuns の設計は前節で述べたように有限体ガロアフィールド体（2^8）の知識と２．２（３）での考え方を用いて設計する。以下の行列演算式の入力値 S00~S33 から出力値 S'00~S'33 をもとめる回路を Verilog-HDL で設計する。

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S00 & S01 & S02 & S03 \\ S10 & S11 & S12 & S13 \\ S20 & S21 & S22 & S23 \\ S30 & S31 & S32 & S33 \end{bmatrix} = \begin{bmatrix} S'00 & S'01 & S'02 & S'03 \\ S'10 & S'11 & S'12 & S'13 \\ S'20 & S'21 & S'22 & S'23 \\ S'30 & S'31 & S'32 & S'33 \end{bmatrix}$$

式１

図１２にＭｉｘＣｏｌｕｍｎｓ変換の入出力仕様を示す。またＭｉｘＣｏｌｕｍｎｓ変換の Verilog-HDL コードを p26 の付録に示す。

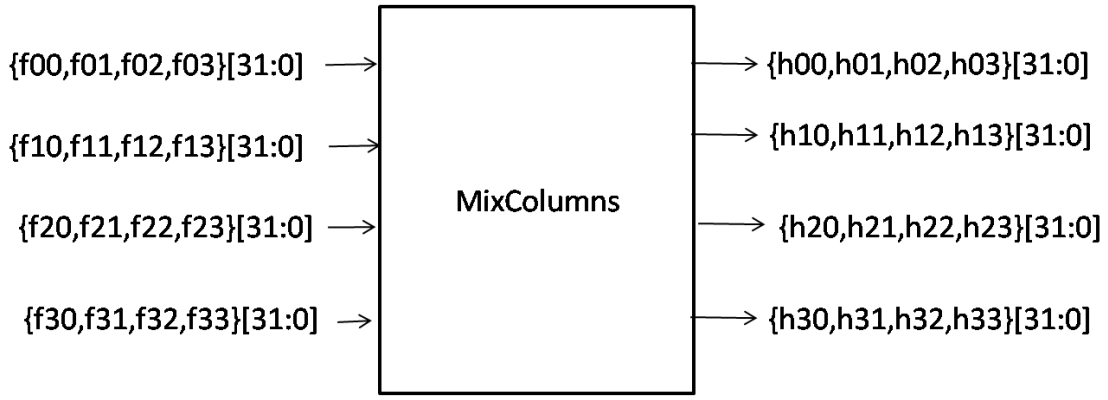

図１２　ＭｉｘＣｏｌｕｍｎｓ変換の入出力仕様

（１） 設計で工夫した点

S00×02 や S10×03 などを計算する場合、前節で述べたｆｘ×ｘの演算方法を用いて設計すると、どうしても回路が大きくなってしまうのでガロア体 GF(2^8)の逆数計算を用いて計算をするようプログラミングした。

（２）ガロア体 GF(2^8)の逆数計算

ある値　y　の逆数 y^{-1} を求めるとき、GF(2^8)のガロア体の要素 y には

$$y^{2^8-1} = y^{255} = 1$$

なる性質がある。したがって、

$$y^{-1} = y^{-1} \cdot y^{255} = y^{254}$$

より、y^{254} を求めれば逆数が求まることになる。したがって、ガロア体の乗算器があればそれを繰り返し用いることで y^{254} を計算することができる。乗算器を図１３に示す。

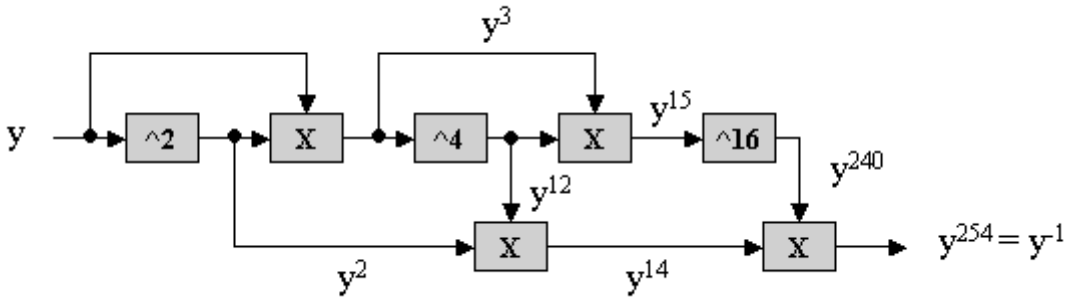

図１３　逆元計算回路

ガロア体の乗算回路を繰り返し用いることで、逆元が計算できる。以下に、ガロア体の乗算回路の１つの実装例を示す。

２つの８ビットの数 A= (a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0)、B=(b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0)　の乗算を考えるとすると、２つの値は以下のような多項式で表すことができる。

$$\begin{cases} A(x) = a_7 \cdot x^7 + a_6 \cdot x^6 + a_5 \cdot x^5 + a_4 \cdot x^4 + a_3 \cdot x^3 + a_2 \cdot x^2 + a_1 \cdot x^1 + a_0 \\ B(x) = b_7 \cdot x^7 + b_6 \cdot x^6 + b_5 \cdot x^5 + b_4 \cdot x^4 + b_3 \cdot x^3 + b_2 \cdot x^2 + b_1 \cdot x^1 + b_0 \end{cases}$$ ・・・式２

この多項式の乗算を行うと、

$$C(x) = c_{14} \cdot x^{14} + c_{13} \cdot x^{13} + c_{12} \cdot x^{12} + c_{11} \cdot x^{11} + c_{10} \cdot x^{10} + c_9 \cdot x^9 + c_8 \cdot x^8 + c_7 \cdot x^7 + c_6 \cdot x^6 + c_5 \cdot x^5 + c_4 \cdot x^4 + c_3 \cdot x^3 + c_2 \cdot x^2 + c_1 \cdot x^1 + c_0$$

$$\begin{cases}
c_{14} = a_7 \cdot b_7 \\
c_{13} = a_7 \cdot b_6 \oplus a_6 \cdot b_7 \\
c_{12} = a_7 \cdot b_5 \oplus a_6 \cdot b_6 \oplus a_5 \cdot b_7 \\
c_{11} = a_7 \cdot b_4 \oplus a_6 \cdot b_5 \oplus a_5 \cdot b_6 \oplus a_4 \cdot b_7 \\
c_{10} = a_7 \cdot b_3 \oplus a_6 \cdot b_4 \oplus a_5 \cdot b_5 \oplus a_4 \cdot b_6 \oplus a_3 \cdot b_7 \\
c_9 = a_7 \cdot b_2 \oplus a_6 \cdot b_3 \oplus a_5 \cdot b_4 \oplus a_4 \cdot b_5 \oplus a_3 \cdot b_6 \oplus a_2 \cdot b_7 \\
c_8 = a_7 \cdot b_1 \oplus a_6 \cdot b_2 \oplus a_5 \cdot b_3 \oplus a_4 \cdot b_4 \oplus a_3 \cdot b_5 \oplus a_2 \cdot b_6 \oplus a_1 \cdot b_7 \\
c_7 = a_7 \cdot b_0 \oplus a_6 \cdot b_1 \oplus a_5 \cdot b_2 \oplus a_4 \cdot b_3 \oplus a_3 \cdot b_4 \oplus a_2 \cdot b_5 \oplus a_1 \cdot b_6 \oplus a_0 \cdot b_7 \\
c_6 = a_6 \cdot b_0 \oplus a_5 \cdot b_1 \oplus a_4 \cdot b_2 \oplus a_3 \cdot b_3 \oplus a_2 \cdot b_4 \oplus a_1 \cdot b_5 \oplus a_0 \cdot b_6 \\
c_5 = a_5 \cdot b_0 \oplus a_4 \cdot b_1 \oplus a_3 \cdot b_2 \oplus a_2 \cdot b_3 \oplus a_1 \cdot b_4 \oplus a_0 \cdot b_5 \\
c_4 = a_4 \cdot b_0 \oplus a_3 \cdot b_1 \oplus a_2 \cdot b_2 \oplus a_1 \cdot b_3 \oplus a_0 \cdot b_4 \\
c_3 = a_3 \cdot b_0 \oplus a_2 \cdot b_1 \oplus a_1 \cdot b_2 \oplus a_0 \cdot b_3 \\
c_2 = a_2 \cdot b_0 \oplus a_1 \cdot b_1 \oplus a_0 \cdot b_2 \\
c_1 = a_1 \cdot b_0 \oplus a_0 \cdot b_1 \\
c_0 = a_0 \cdot b_0
\end{cases} \quad \cdots \text{式 3}$$

で与えられる１４次の多項式となる。また各係数は GF(2)のガロア体の演算に従うので、乗算は AND 演算、加算は排他的論理和 EXOR 演算となる。
ここで既約多項式＝０とし、GF(2)では加算と減算は同じであるので、

$$
\begin{cases}
m(x) = x^8 + x^4 + x^3 + x + 1 = 0 \\
x^8 = x^4 + x^3 + x + 1 \\
x^9 = x^5 + x^4 + x^2 + x \\
x^{10} = x^6 + x^5 + x^3 + x^2 \\
x^{11} = x^7 + x^6 + x^4 + x^3 \\
x^{12} = x^8 + x^7 + x^5 + x^4 = (x^4 + x^3 + x + 1) + x^7 + x^5 + x^4 = x^7 + x^5 + x^3 + x + 1 \\
x^{13} = x^8 + x^6 + x^4 + x^2 + x = (x^4 + x^3 + x + 1) + x^6 + x^4 + x^2 + x = x^6 + x^3 + x^2 + 1 \\
x^{14} = x^7 + x^4 + x^3 + x
\end{cases}
$$

・・・式 4

なる関係式を用いると上記 C(X)は 7 次以下の多項式 D(X)に変換することができる。

$$
\begin{cases}
D(x) = d_7 \cdot x^7 + d_6 \cdot x^6 + d_5 \cdot x^5 + d_4 \cdot x^4 + d_3 \cdot x^3 + d_2 \cdot x^2 + d_1 \cdot x^1 + d_0 \\
\\
d_7 = c_7 \oplus c_{11} \oplus c_{12} \oplus c_{14} \\
d_6 = c_6 \oplus c_{10} \oplus c_{11} \oplus c_{13} \\
d_5 = c_5 \oplus c_9 \oplus c_{10} \oplus c_{12} \\
d_4 = c_4 \oplus c_8 \oplus c_9 \oplus c_{11} \oplus c_{14} \\
d_3 = c_3 \oplus c_8 \oplus c_{10} \oplus c_{11} \oplus c_{12} \oplus c_{13} \oplus c_{14} \\
d_2 = c_2 \oplus c_9 \oplus c_{10} \oplus c_{13} \\
d_1 = c_1 \oplus c_8 \oplus c_9 \oplus c_{12} \oplus c_{14} \\
d_0 = c_0 \oplus c_8 \oplus c_{12} \oplus c_{13}
\end{cases}
$$

・・・式 5

以上の計算により、２つの８ビットの数 A= (a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0)、B=(b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0) の乗算結果 D=(d_7, d_6, d_5, d_4, d_3, d_2, d_1, d_0) が得られる
すなわち、実際の実装では多数の AND と EXOR を用いることにより乗算器を実現することができる。

この考え方に基づいて式１の fx×02 を計算してみると

fx×02 = {a_7……a_0}×{00000010} = {d_7……d_0}

$d_7 = a_6$　$d_6 = a_5$　$d_5 = a_4$　$d_4 = a_3 + a_7$　$d_3 = a_2 + a_7$　$d_2 = a_1$　$d_1 = a_0 + a_7$　$d_0 = a_7$

fx×03 も同様に

$d_7 = a_6 + a_7$　$d_6 = a_5 + a_6$　$d_5 = a_4 + a_5$　$d_4 = a_3 + a_4 + a_7$　$d_3 = a_2 + a_3 + a_7$　$d_2 = a_1 + a_2$

$d_1 = a_0 + a_1 + a_7$　$d_0 = a_0 + a_7$ となる

これらの結果をもとに記述した Verilog-HDL コードを図１４、図１５に示す。

```
module MC_mul02(a,d);

input [7:0] a;
output [7:0] d;



assign d[7:7] = a[6:6];
assign d[6:6] = a[5:5];
assign d[5:5] = a[4:4];
assign d[4:4] = a[3:3] ^ a[7:7];
assign d[3:3] = a[2:2] ^ a[7:7];
assign d[2:2] = a[1:1];
assign d[1:1] = a[0:0] ^ a[7:7];
assign d[0:0] = a[7:7];



endmodule
```

図１４　fx×02 の Verilog-HDL 記述

```
module MC_mul03(a,d);

input [7:0] a;
output [7:0] d;



assign d[7:7] = a[6:6] ^ a[7:7];
assign d[6:6] = a[5:5] ^ a[6:6];
assign d[5:5] = a[4:4] ^ a[5:5];
assign d[4:4] = a[3:3] ^ a[4:4] ^ a[7:7];
assign d[3:3] = a[2:2] ^ a[3:3] ^ a[7:7];
assign d[2:2] = a[1:1] ^ a[2:2];
assign d[1:1] = a[0:0] ^   a[1:1] ^ a[7:7];
assign d[0:0] = a[0:0] ^ a[7:7];



endmodule
```

図１５　fx×03 の Verilog-HDL 記述

これらのコードは附録 MixColumns 変換プログラムの HDL 記述の中で使われている。

３．３　ShiftRows の設計

ShiftRows の設計は前節で述べたように有限体ガロアフィールド体（2^8）の知識と２．２（４）の考えを基にして設計した。ShiftRows 変換の入出力仕様を図１６に示す。
また Verilog-HDL コードを図１７に示す。

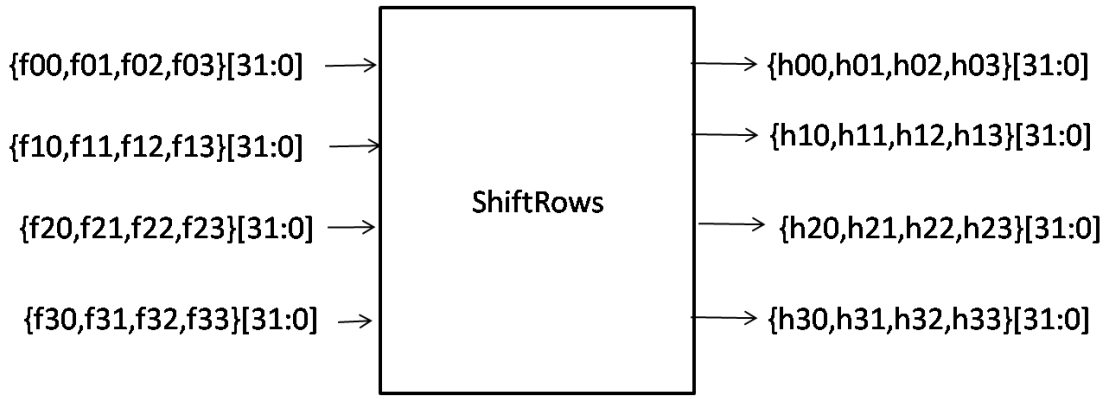

図１６　ShiftRows の入出力仕様

```
module ShiftRows(
        input [7:0] f00, f01, f02, f03,
        input [7:0] f10, f11, f12, f13,
        input [7:0] f20, f21, f22, f23,
        input [7:0] f30, f31, f32, f33,
    output [7:0] h00, h01, h02, h03,
    output [7:0] h10, h11, h12, h13,
    output [7:0] h20, h21, h22, h23,
    output [7:0] h30, h31, h32, h33
);

assign h00 = f00;
assign h01 = f01;
assign h02 = f02;
assign h03 = f03;

assign h10 =  f11;
assign h11 =  f12;
assign h12 =  f13;
assign h13 =  f10;
```

```
assign h20 =  f22;
assign h21 =  f23;
assign h22 =  f20;
assign h23 =  f21;

assign h30 = f33;
assign h31 = f30;
assign h32 = f31;
assign h33 = f32;

endmodule
```

図１７　ShiftRows の Verilog-HDL 記述

図１７の１２行目から 29 行目は 2.2(4)の考えを基に assign 文で記述した。それほど複雑なプログラムにせず素直に作成した。

３．４　考察

多項式計算の設計は hx = fx×gx と回路規模をあまり考えず fx×x の演算方法の考えを基にプログラミングし、ShiftRows も同様にプログラミングしたが、MixColumns 変換の設計に関しては回路規模を意識してプログラミングした。この３つはシミュレーションしても正しい結果が得られたので、これから回路規模を意識した設計が求められるなと感じた。また回路規模を意識した設計だけでなくほかの方法で設計できないかをこれから考えていきたいと考えている。

4　FPGA ボードへの実装と検証

実験環境

システムの環境として用いた環境を表２に示す

表 2：実験環境

ハードウェア動作合成	論理合成ツール	Xilinx ISE 9. 2 i
回路シミュレーション	PC 環境	IntelCore2duo2.4Ghz,Memory2GB
	シミュレーションツール	ModelSim XE III 6.1e

４．１　多項式計算の実装と検証

MixColumuns 変換の列演算（S00×02,S10×03）を前節で述べた多項式計算 fx×x の演算方法を用いて作成した回路を FPGA ボードへ実装してみると表 3 のような結果が得られた。

表 3 :実装結果（多項式計算）

回路面積	39 out of 1920 ・・・　2%
クロックサイクル時間	4.3 ×　10^{-9}秒

処理時間は周波数 f = 231.957MHz の逆数によって得られた結果である

４．２　MixColumuns の実装と検証

MixColumns 変換の列演算（S00×02,S10×03）を前節で述べたガロア体 GF(2^8)の逆数計算を用いて作成した回路を FPGA ボードへ実装してみると表 4 のような結果が得られた。

表 4 :実装結果　（逆数計算）

回路面積	１７ out of 1920 ・・・　1%
クロックサイクル時間	4.2×10^{-9} 秒

処理時間は周波数 f = 238.132MHz の逆数によって得られた結果である

４．３　考察

これら２つの実装結果を比較して回路面積、クロックサイクル時間が大きく異なっていることが分かる。まず回路面積だが多項式計算 fx× x の計算をして計算結果を出力する回路は逆数計算を用いて計算結果を出力する回路と比較して 2 倍以上の回路規模になっていることがわかった。これは前節で述べた fx× x の計算は逆数計算を用いた回路と比べて非常に手間がかかり、その分、回路規模が大きくなったのではないかと思われる。またクロックサイクル時間のほうだがこれもまた同様に fx×X を用いた回路のほうが逆数計算を用いた回路と比べて非常に大きくかかることがわかった。これもまた同様の理由であると思われる。しかし両方のプログラムを見てまだまだ改善できる点はたくさんあるのでこれら２つの実装結果はさらに最適化ができると思われる。

５　おわりに

本研究では、現在のシステム LSI の設計について考察し、AES 暗号化回路（MixColumns 変換、ShiftRows 変換）を設計した。また Xilinx 社の提供する ISE を用いて、Memec 社の FPGA ボード上に実装し、どのようにすれば高速化できるかを検証した。
今後の課題として、AES 暗号化回路の特に回路規模を意識した設計が必要と思われる。大学院ではさらに FPGA ボードへ実装して、どのように高速化できるかを本研究以外の方法で検証して、システムのさらなる発展を目指そうと考えている。

LSI 回路設計は順調に発展を続けており、今後も拡大し続けるであると思われる。一方で、少し前にあったソフトウェア危機というものと似た現象も LSI の業界では発生しつつあり、LSI の設計が危ぶまれている。当然、その解決策はいくつか挙げられており、主なものにはシステムレベル言語設計（高位合成）の再利用などがある。中でも高位合成などではハードウェアとソフトウェアの双方の十分な知識が必要とされる。今後の LSI 設計に要求される事柄は一面的な知識では解決できない。そういった意味で、電子情報デザイン学科というハードとソフトの両面を学ぼうとする学科に組み込まれる本研究室は最適な環境に近いだろう。本研究室の研究活動がさらに活発になり、将来の技術に関われる研究や、必要とされる研究員が多く輩出されることを願って止まない。

謝辞

本研究の機会を与えてくださり、貴重な助言、ご指導をいただきました山崎勝弘教授に深く感謝いたします。

また、本研究に関して様々な相談に乗っていただき、貴重な助言を頂いた TUAN 氏、及び色々な面で励ましを下さった高性能計算研究室の皆様に心より深く感謝いたします。

参考文献

[1]　三谷政昭：今日から使える工業数学、CQ 出版

[2]　William Stallings　：Cryptography and NetworkSecurity

[3] 深山正幸,北山章夫,秋山純一,鈴木正國:HDL による VLSI 設計,共立出版,1999.

[4] 清家秀律,黒川恭一:AES 暗号用 SubBytes,MixColumns 変換の方式設計,情報処理学会研究報告.CSEC, Vol.2001, No.75, pp119-126, 2001.07.

[5]　梅原 直人 : ハード/ソフト最適分割を考慮した AES 暗号システムと JPEG エンコーダの設計と検証　２００５

[6]　http://mailsrv.nara-edu.ac.jp/~asait/crypto/crypto/crypt.html

[7]　http://www.ie.u-ryukyu.ac.jp/~wada/design04/spec_j.html

[8] 神永正博, 山田聖、渡邊高志共著：Java で作って学ぶ暗号技術： RSA,AES,SHA の基礎から SSL まで　- 東京： 森北出版 , 2008.5.

附録　MixColumns 変換プログラムの HDL 記述

```
module MixColumns(
        input [7:0] f00, f01, f02, f03,
        input [7:0] f10, f11, f12, f13,
        input [7:0] f20, f21, f22, f23,
        input [7:0] f30, f31, f32, f33,
    output [7:0] h00, h01, h02, h03,
    output [7:0] h10, h11, h12, h13,
    output [7:0] h20, h21, h22, h23,
    output [7:0] h30, h31, h32, h33
);

wire   [7:0] w002, w103;
wire   [7:0] w012, w113;
wire   [7:0] w022, w123;
wire   [7:0] w032, w133;

wire   [7:0] w102, w203;
wire   [7:0] w112, w213;
wire   [7:0] w122, w223;
wire   [7:0] w132, w233;

wire   [7:0] w202, w303;
wire   [7:0] w212, w313;
wire   [7:0] w222, w323;
wire   [7:0] w232, w333;

wire   [7:0] w302, w003;
wire   [7:0] w312, w013;
wire   [7:0] w322, w023;
wire   [7:0] w332, w033;
```

```
MC_mul02 i002(.a(f00), .d(w002));
  MC_mul03 i103(.a(f10), .d(w103));
MC_mul02 i012(.a(f01), .d(w012));
  MC_mul03 i113(.a(f11), .d(w113));
MC_mul02 i022(.a(f02), .d(w022));
  MC_mul03 i123(.a(f12), .d(w123));
MC_mul02 i032(.a(f03), .d(w032));
  MC_mul03 i133(.a(f13), .d(w133));

MC_mul02 i102(.a(f10), .d(w102));
  MC_mul03 i203(.a(f20), .d(w203));
MC_mul02 i112(.a(f11), .d(w112));
  MC_mul03 i213(.a(f21), .d(w213));
MC_mul02 i122(.a(f12), .d(w122));
  MC_mul03 i223(.a(f22), .d(w223));
MC_mul02 i132(.a(f13), .d(w132));
  MC_mul03 i233(.a(f23), .d(w233));

MC_mul02 i202(.a(f20), .d(w202));
  MC_mul03 i303(.a(f30), .d(w303));
MC_mul02 i212(.a(f21), .d(w212));
  MC_mul03 i313(.a(f31), .d(w313));
MC_mul02 i222(.a(f22), .d(w222));
  MC_mul03 i323(.a(f32), .d(w323));
MC_mul02 i232(.a(f23), .d(w232));
  MC_mul03 i333(.a(f33), .d(w333));

MC_mul02 i302(.a(f30), .d(w302));
  MC_mul03 i003(.a(f00), .d(w003));
MC_mul02 i312(.a(f31), .d(w312));
  MC_mul03 i013(.a(f01), .d(w013));
MC_mul02 i322(.a(f32), .d(w322));
  MC_mul03 i023(.a(f02), .d(w023));
MC_mul02 i332(.a(f33), .d(w332));
```

```
  MC_mul03 i033(.a(f03), .d(w033));
assign h00 = w002 ^w103 ^f20 ^f30;
assign h01 = w012 ^w113 ^f21 ^f31;
assign h02 = w022 ^w123 ^f22 ^f32;
assign h03 = w032 ^w133 ^f23 ^f33;

assign h10 =    f00 ^w102 ^w203 ^f30;
assign h11 =    f01 ^w112 ^w213 ^f31;
assign h12 =    f02 ^w122 ^w223 ^f32;
assign h13 =    f03 ^w132 ^w233 ^f33;

assign h20 =    f00 ^f10 ^w202 ^w303;
assign h21 =    f01 ^f11 ^w212 ^w313;
assign h22 =    f02 ^f12 ^w222 ^w323;
assign h23 =    f03 ^f13 ^w232 ^w333;

assign h30 = w003 ^f10 ^f20 ^w302;
assign h31 = w013 ^f11 ^f21 ^w312;
assign h32 = w023 ^f12 ^f22 ^w322;
assign h33 = w033 ^f13 ^f23 ^w332;




endmodule
```