Design Wave設計コンテスト関連企画

数学いらずのAES暗号SubBytes設計ガイド

――「組み合わせ回路を使って1クロックで計算」にチャレンジ！

Dr. S. Morioka

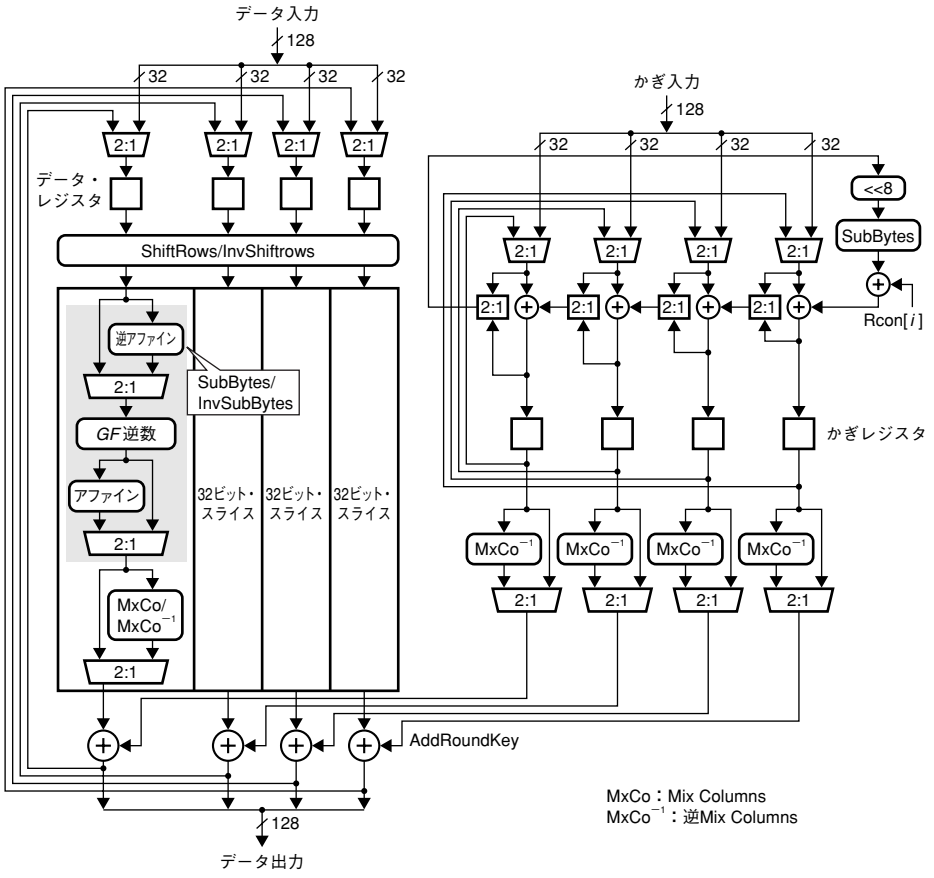
デバイスの記事　ビギナーズ

毎年恒例のDesign Wave設計コンテストですが，今回はAES（Advanced Encryption Standard）暗号回路に使われているSubBytes（S-Box）演算回路が設計課題です．本稿では，SubBytesとは何かを簡単に説明し，実用回路では通常どのように設計されているかを紹介したうえで，コンテストでチャレンジしてみるとおもしろいテーマを示します．　（著者）

“SubBytes”というのは聞き慣れないことばで，何か難しいもののように思われるかもしれません．数学的な話を持ち出すとたしかに難しくなるのですが，「回路を作る」という視点で見れば，ごく単純でわかりやすいものです．

●入出力の表に従ってデータ変換を行うだけ

共通かぎ暗号AES（Advanced Encryption Standard）の回路ブロック図を**図1**に示します．AESでは128ビットの入力データに対して，ラウンド関数と呼ばれる変換を繰



〔図1〕
共通かぎ暗号回路AESの基本構成
共通かぎ暗号回路は，「ラウンド関数」と呼ばれるデータ変換部（図の左半分）と，データ変換に使うラウンドかぎを生成する部分（図の右半分）からなる．ラウンド関数やラウンドかぎ生成部にSubBytes（図の灰色で囲まれた部分）が用いられる．

〔表1〕SubBytesのテーブル

横方向は入力の下位4ビットを，縦方向は上位4ビットを示す．SubBytesは8ビット入力，8ビット出力の1対1写像であり，この表に示された変換を行う．

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
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

〔表2〕InvSubBytesのテーブル

InvSubBytesはSubBytesの逆関数であり，データを復号するときに用いる．なお，表の横方向は入力の下位4ビットを，縦方向は上位4ビットを示す．

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
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

り返し適用します[1]．**図1**のうち，データ・レジスタを出てから戻ってくるまでの組み合わせ回路部分が，このラウンド関数に相当します．

ラウンド関数は，主に次の二つの操作を行います．

- バイト単位でデータを置換する
- 128ビット・データ中のビットの順番を入れ替える

このうち，前者の演算回路がSubBytesです．**図1**では，灰色で囲まれた部分がSubBytesにあたります．

なお，暗号化データを復号するときは上記のラウンド関数の逆関数を用います．ここでは，「InvSubBytes」と呼ばれるSubBytesの逆関数が使われます．

SubBytesの定義は「ガロア体」と呼ばれる数学理論に基づいているのですが，単に回路を作るだけなら，この理論を理解している必要はあまりありません．SubBytesとInvSubBytesはともに8ビット入出力の1対1写像であり，それぞれ**表1**，**表2**に示すデータ変換を行います．これだけのことなのです．

1．実用回路はどのように作られているのか？

それでは，SubBytesやInvSubBytesが実用のAES回路でどのように作られているのかを紹介します．

●真理値表からの合成で組み合わせ回路を得るのが普通

表1，**表2**に示した真理値表（テーブル）を直接Verilog HDLやVHDLのcase文で記述して論理合成ツールにかけると，必要な組み合わせ回路を得ることができます（**リスト1**）．こう言うと，「何のおもしろ味もない！」とみなさんは思われるかもしれません．しかし，多くの実用AES回路ではこの作りかたを採用しています．その理由としては，以下の点が挙げられます．

- AES回路としては，最少クロック（11クロック）でCBC（Cipher Block Chaining）モード暗号化処理[注1]を行える

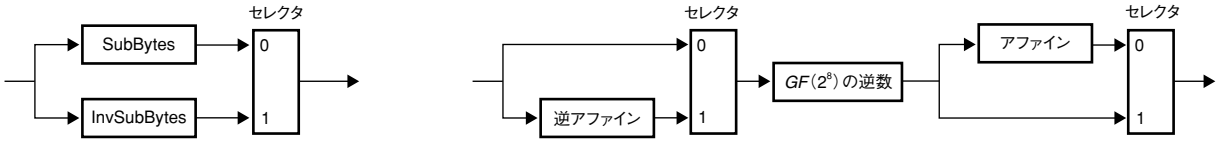
〔リスト1〕もっとも標準的なSubBytes/InvSubBytesのRTL記述

SubBytes/InvSubBytesの入出力幅は8ビットなので，真理値表（テーブル）が書ける程度である．実用のAES回路でもっとも普通に行われている作りかたが，テーブルからの論理合成である．

```
module subbytes(x, y);
    input     [7:0]   x;
    output    [7:0]   y;
    reg       [7:0]   y;

    always @(x) begin
        case (x)
        8'b00000000:    y <= 8'b01100011;   // 00 63
        8'b00000001:    y <= 8'b01111100;   // 01 7c
            :
        8'b11111101:    y <= 8'b01010100;   // fd 54
        8'b11111110:    y <= 8'b10111011;   // fe bb
        default:        y <= 8'b00010110;   // ff 16
        endcase
    end
endmodule
```

注1：あるデータの暗号化を行うとき，直前のデータの暗号化結果を使用することで暗号強度を強くする方法のこと．CBCでは，回路内部処理のパイプライン化によって処理のスループットを向上させることが原理的に難しい．スループットを上げるには，SubBytesを含むラウンド関数全体を1クロックで処理するとともに，ゲート遅延を減らす必要がある[2), 4)]．



(a) SubBytesとInvSubBytesを別々に作る　　(b) 逆元演算回路を共有する

〔図2〕SubBytesとInvSubBytesの回路共有

(a)のように二つの回路を別々に作って出力を切り替える方法を採ると，回路規模は大きくなる．ここで，SubBytesとInvSubBytesは，それぞれガロア体の逆元演算と，ある演算(アファイン/逆アファイン変換)の組み合わせとして定義されている．そのため，(b)のように逆元演算回路を共有すれば，回路規模をかなり小さくできる．ただし，回路遅延が増え，フォールス・パスもできてしまう．速度の点では(a)がほぼ最速．

〔表3〕$GF(2^8)$の逆元演算のテーブル

ガロア体の逆元演算もSubBytesと同じく，8ビット入出力の1対1写像である．論理合成が可能． なお，AESで使われている$GF(2^8)$では，既約多項式$x^8+x^4+x^3+x+1$で多項式基底である．

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	01	8D	F6	CB	52	7B	D1	E8	4F	29	C0	B0	E1	E5	C7
1	74	B4	AA	4B	99	2B	60	5F	58	3F	FD	CC	FF	40	EE	B2
2	3A	6E	5A	F1	55	4D	A8	C9	C1	0A	98	15	30	44	A2	C2
3	2C	45	92	6C	F3	39	66	42	F2	35	20	6F	77	BB	59	19
4	1D	FE	37	67	2D	31	F5	69	A7	64	AB	13	54	25	E9	09
5	ED	5C	05	CA	4C	24	87	BF	18	3E	22	F0	51	EC	61	17
6	16	5E	AF	D3	49	A6	36	43	F4	47	91	DF	33	93	21	3B
7	79	B7	97	85	10	B5	BA	3C	B6	70	D0	06	A1	FA	81	82
8	83	7E	7F	80	96	73	BE	56	9B	9E	95	D9	F7	02	B9	A4
9	DE	6A	32	6D	D8	8A	84	72	2A	14	9F	88	F9	DC	89	9A
A	FB	7C	2E	C3	8F	B8	65	48	26	C8	12	4A	CE	E7	D2	62
B	0C	E0	1F	EF	11	75	78	71	A5	8E	76	3D	BD	BC	86	57
C	0B	28	2F	A3	DA	D4	E4	0F	A9	27	53	04	1B	FC	AC	E6
D	7A	07	AE	63	C5	DB	E2	EA	94	8B	C4	D5	9D	F8	90	6B
E	B1	0D	D6	EB	C6	0E	CF	AD	08	4E	D7	E3	5D	50	1E	B3
F	5B	23	38	34	68	46	03	8C	DD	9C	7D	A0	CD	1A	41	1C

〔リスト2〕アファイン変換回路と逆アファイン変換回路

これらは，8×8のXORマトリックスである．すでに十分に小規模な回路ではあるが，出力ビット間の共通項をまとめると，さらなる規模縮小も可能．

```
module affine(x, y);
    input   [7:0]   x;
    output  [7:0]   y;
    assign  y[7] =   (x[3] ^ x[4] ^ x[5] ^ x[6] ^ x[7]);
    assign  y[6] =  ~(x[2] ^ x[3] ^ x[4] ^ x[5] ^ x[6]);
    assign  y[5] =  ~(x[1] ^ x[2] ^ x[3] ^ x[4] ^ x[5]);
    assign  y[4] =   (x[0] ^ x[1] ^ x[2] ^ x[3] ^ x[4]);
    assign  y[3] =   (x[0] ^ x[1] ^ x[2] ^ x[3] ^ x[7]);
    assign  y[2] =   (x[0] ^ x[1] ^ x[2] ^ x[6] ^ x[7]);
    assign  y[1] =  ~(x[0] ^ x[1] ^ x[5] ^ x[6] ^ x[7]);
    assign  y[0] =  ~(x[0] ^ x[4] ^ x[5] ^ x[6] ^ x[7]);
endmodule

module invaffine(x, y);
    input   [7:0]   x;
    output  [7:0]   y;
    assign  y[7] =   (x[1] ^ x[4] ^ x[6]);
    assign  y[6] =   (x[0] ^ x[3] ^ x[5]);
    assign  y[5] =   (x[2] ^ x[4] ^ x[7]);
    assign  y[4] =   (x[1] ^ x[3] ^ x[6]);
    assign  y[3] =   (x[0] ^ x[2] ^ x[5]);
    assign  y[2] =  ~(x[1] ^ x[4] ^ x[7]);
    assign  y[1] =   (x[0] ^ x[3] ^ x[6]);
    assign  y[0] =  ~(x[2] ^ x[5] ^ x[7]);
endmodule
```

アファイン変換回路　逆アファイン変換回路

回路のニーズがいちばん高い．そのようなAES回路ではSubBytesを組み合わせ回路として組む必要がある．

- 論理合成ツールを使うと，クリティカル・パスの遅延が最小に近い回路を得ることができる(後述の**表4**を参照)．

論理合成時のタイミング制約や使用するテクノロジ・ライブラリにもよりますが，この方法で得られるSubBytes回路の規模はおよそ1,500～2,500ゲートとやや大きくなります．

FPGAで実装する場合に，**表1**，**表2**のテーブルをそのまま内蔵RAMへマッピングすると，高速でしかもリソース使用量の少ない回路を得ることができます．

●SubBytesとInvSubBytesを両方使う場合のくふう

暗号化/復号化の両方の処理を行う回路では，SubBytesとInvSubBytesを実装する必要があります．この回路のもっとも単純な実装方法は，SubBytesとInvSubBytesをそれぞれ別の真理値表で記述し，**図2(a)**のように組み合わせるというものです．この場合，回路規模は両方合わせて約3,000～5,000ゲートとなります．回路速度は，この構成がほぼ最速です．

さて，ここでSubBytesの数学的定義をほんの少し利用するだけで，SubBytesとInvSubBytesの回路の大半を共有できます．SubBytesとInvSubBytesは，それぞれガロア体の逆元演算と，ある演算(アファイン変換または逆アファイン変換)の組み合わせによって定義されています．このため，両者の間でガロア体の逆元演算を**図2(b)**のように共有できます．**図2(b)**の上半分のパスがSubBytes，下半

分のパスがInvSubBytesです．図中，「$GF(2^8)$の逆数」と書いてある部分が逆元演算回路です．

SubBytesで使われているガロア体の逆元演算は，**表3**に示す8ビット入出力表で定義されます．これもまた，HDLのcase文で**表3**のテーブルを記述して論理合成すれば，回路を得ることができます．また，アファイン変換や逆アファイン変換は，**リスト2**に示す回路で作ることができます．

ここまでの範囲であれば，ガロア体やアファイン変換がいったい何であるのかを知らなくても，回路を作ることは可能です．

●数学的な知識で回路の小規模化を目指す

図2(b)から予想できるとおり，SubBytes/InvSubBytes回路を作る際に，性能上もっともネックになるのがガロア体の逆元演算です．本稿では詳しく説明しませんが，逆元演算を高性能な組み合わせ回路として作る方法はいろいろと研究されています．

実用回路でよく利用されているのは，合成体(composite field)という数学的な知識を使って小規模な逆元演算回路を組む方法です[3]．また参考文献4)では，高速化についてフォーマル・ベリフィケーションなどでよく使われているBDD(2分決定グラフ)を利用する方法が示されています．低消費電力化については，合成体の手法とXOR回路最適化手法を組み合わせた方法があります[5]．これらについては，本誌2003年7月号[2]で回路の作りかたが示されていますので，そちらも参照してみてください．

なお，ガロア体$GF(2^8)$では，任意の値Xについてその逆元$X^{-1} = X^{254}$という性質があるので，乗算器をいくつか組み合わせて逆元演算回路を作ることも可能です[2),6),7)]．しかし，組み合わせ回路としてSubBytesを組む場合，乗算器を組み合わせるアプローチは，速度，回路規模，消費電力のいずれの点でもあまり得策ではありません．

●SubBytesについては数学的知識が有効とは限らない

さまざまな回路の組みかたで作られたSubBytesの性能を**表4**に示します(本表は参考文献5)から引用した)．もちろん，絶対的な性能値は使用するテクノロジ・ライブラリによって変わりますが，相対的な性能差はあまり変わりません．

表4からも明らかなとおり，速度や回路規模，消費電力は回路の組みかたによっていずれも大きく変化します．演算回路としても，これほど変化があるものは珍しいと思われます．筆者の経験では，前述の合成体のような数学知識は，回路規模の縮小には役立つのですが，高速化には意外と役に立ちません．高速化では，論理自動合成の知識のほうがずっと重要になるようです．消費電力削減には，数学知識と論理合成の知識の組み合わせが有効なようです．

一般の演算回路の設計であれば，数学知識を利用することは有効であるとほぼ決まっています．しかし，SubBytesでは「数学知識」にこだわらないほうが効果的に最適化を行えるかもしれないというおもしろさがあります．これがガロア体を知らなくても，ぜひこのテーマにチャレンジしていただきたい理由です．自由な発想で回路を作ってみると，とてもおもしろい結果が得られるかもしれません．

〔表4〕組み合わせ回路で作られたいろいろなSubBytesの性能比較

論理構成を変更することで大幅な性能調整が可能である．特に，合成体を用いた場合の規模縮小効果は際立っている．SubBytesでは，論理合成でもかなり良好な回路が得られる．ガロア体の数学知識は回路規模の縮小には役立つが，高速化にはあまり役に立たない．

アルゴリズム	出力遅延 (ns)	回路規模 (ゲート)	パワー (μW@100MHz，1.5V)
伊東・辻井[6),7)]	2.79	1,771	21,000
合成体[3)]	2.19	354	1,360
3段AND-XOR[5)]	1.43	712	290
1段AND-XOR	1.14	2,241	3,430
積和形	0.69	1,650	950
BDD	0.69	1,399	2,750
自動合成	0.68	2,623	1,440
Twisted BDD[4)]	0.43	2,818	−

(0.13 μmプロセスのスタンダード・セル)

2. 見込みのありそうな最適化アプローチは？

それでは，設計コンテストに向けて，どのような回路最適化のアプローチがあるのか，いくつか考えてみましょう．

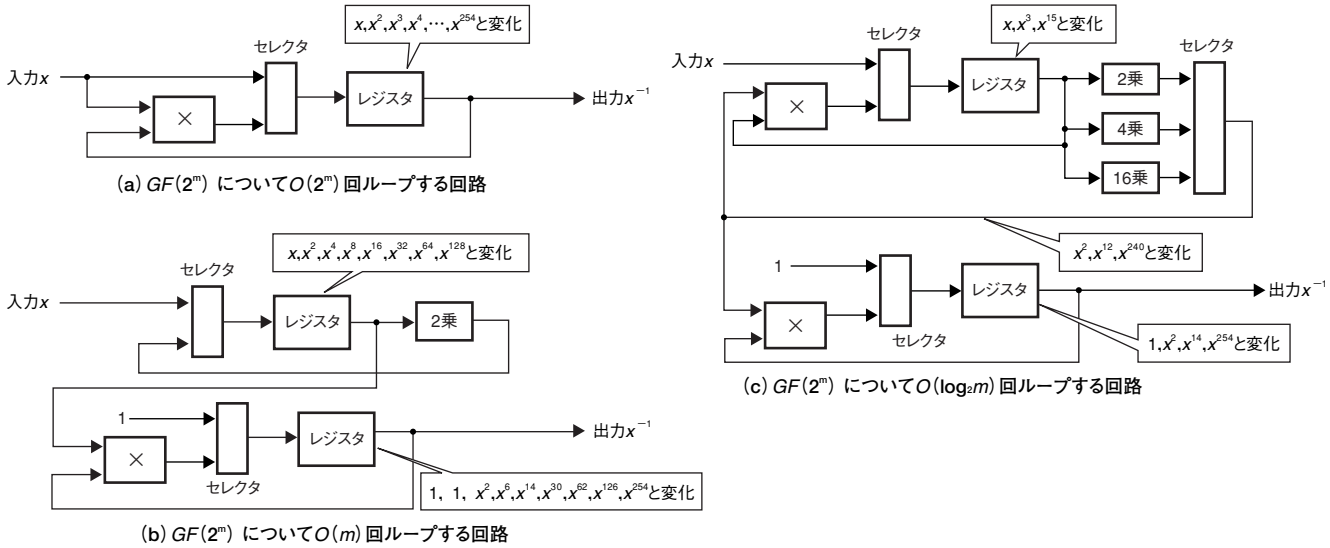
●順序回路で複数クロックかけて計算する

ここまでの説明は，すべてSubBytesを組み合わせ回路で組むということが前提でした．しかし，普通のハードウェア設計でもよく行われるような，順序回路化による回路規模の縮小や動作周波数の向上はもちろん可能です．

順序回路化が有効な部分は，主に逆元演算($GF(2^8)$の逆数)であると考えられます．逆元演算を順序回路化する場合，次のようなアプローチがあります．

1) 組み合わせ回路で作った逆元演算回路のパイプライン化

こうすれば，動作クロック周波数を上げることができま



〔図3〕順序回路による逆元演算回路(8ビット入力)の例

SubBytesで用いるガロア体$GF(2^8)$では，入力を254乗することで逆元を計算できるが，使う回路リソースを少し増やしていくだけで劇的にループ回数(クロック・サイクル数)を減らすことが可能．(a)は単純に入力を254回乗じるもので，回路は小さいが遅い．(b)はx^2，x^4，x^8，…を掛け合わせて8(+1)クロックで計算する回路．(c)は伊東・辻井のアルゴリズム[6]という，3(+1)クロックで計算する回路である．(c)は，現在知られている限りではほぼ最速の方法である．また，この例とは別に，ビット・シリアル入力の逆元演算器や乗算器を構成する手法もある．

す．一般には回路規模がかなり増加しますが，合成体を使って作ったSubBytesなら，パイプライン化してもわりと小規模な回路のままですむでしょう．

2) X^{254}を計算する際のループ回数

前述したとおり，Xの逆元は乗算器を使ってX^{254}を計算することで求められます．単純にXを254回掛け合わせる回路(**図3(a)**)，レジスタを二つ使い$X^{254}=X^2\times X^4\times X^8\times\cdots\times X^{128}$という性質のもとで8(+1)回のループで計算する回路(**図3(b)**)，乗算器を二つ使い$X^{254}=X^2\times X^{12}\times X^{240}$という性質のもとで3(+1)回のループで計算する回路(**図3(c)**)などがあります．このうち**図3(c)**の回路は，現在知られている限りもっともループ回数が少なくてすむ逆元演算回路です[6]．

3) 上記2)について乗算を合成体上で行う

このアプローチでは回路規模が劇的に減るでしょう．合成体上の乗算器については参考文献2)，8)を参照してください．

4) 使っている乗算器，または逆元演算器全体をビット・シリアル入力にする

この場合，処理クロック数は増えますが，回路規模をさらに減らせます．また，シストリック・アレイにしてみてもおもしろいかもしれません．

●組み合わせ回路にして1クロックで計算する

実用AES回路では，SubBytesを組み合わせ回路で作るのが普通であるとすでに紹介しましたが，この最適化手法はかなり煮詰まってきているのかもしれません．

しかし，実用的価値も学術的価値もいちばん高いので，組み合わせ回路SubBytesの最適化にぜひチャレンジしてください！！！[注2]．

注2：筆者としては，文献2)，3)，4)，5)などの回路を追い抜かれてしまうと，正直悔しい．と同時に，とてもうれしく思う．そこにどのようなアイデアが使われるのか，ぜひ見てみたい！

参考文献

1) 佐藤証，森岡澄夫，「暗号処理のソフト vs. ハード」，『Design Wave Magazine』，pp.72-79，2003年9月号．
2) 森岡澄夫，「エラー訂正や暗号処理で使われる演算回路を究める」，『Design Wave Magazine』，pp.57-67，2003年7月号．
3) A. Satoh， S. Morioka， K. Takano and S. Munetoh，“A Compact Rijndael Hardware Architecture with S-Box Optimization”，*ASIACRYPT2001*，LNCS Vol.2248，pp.239-254，Dec 2001.
4) S. Morioka and A. Satoh，“A 10Gbps Full-AES Crypto Design with a Twisted-BDD S-Box Architecture”，*Proc. IEEE Intl. Conf. on Computer Design 2002*（*ICCD2002*），pp.98-103，2002.
5) 森岡澄夫，佐藤証，「共通鍵暗号AESの低消費電力論理回路構成法」，『情報処理学会論文誌』，pp.1321-1328，2003年5月号．
http://www.ie.u-ryukyu.ac.jp/~wada/design04/Morioka.pdf

FPGAによる回路実装の結果

p.154の**図2**に示した二つの回路を，本誌2003年10月号付属のFPGA（米国Altera社のEP1C3T100C8）にマッピングしてみました．入出力ピンの遅延を除いて回路速度を測定するため，マッピングした回路の入力と出力を8ビット・レジスタではさみました．SubBytesやInvSubBytesは，テーブルからの論理合成としました．Quartus II Version 3.0を使い，デフォルトのオプション設定のもとで論理合成と配置配線を行いました．

さて結果ですが，本文の**図2(a)**に示したSubBytes/InvSubBytesの並列型では回路規模が424LE（Logic Element）に，遅延時間が6.865ns（145.67MHz）になりました．一方，**図2(b)**の逆元演算共有型では回路規模が237LEに，遅延時間が9.163ns（103.86MHz）になり，本文で述べたとおり速度は前者が，回路規模は後者が有利という結果が出ました．

ただし，後者の共有型は実際にはもう少しだけ速く動作します（おそらく遅延時間は8ns程度）．ツールがレポートしてくる回路遅延最大の信号経路は，**図2(b)**における逆アファイン変換→逆元演算→アファイン変換なのですが，これはフォールス・パス（実際には活性化されない信号経路）だからです．

参考までに，本文中で触れた合成体上の演算回路（参考文献2）などを参照）を利用した場合，共有型の回路規模は106LE，遅延時間は14.519ns（68.88MHz）でした．このように，合成体を使うと論理合成の場合より速度は落ちるものの，回路規模は小さくなります．

6) S. Morioka and Y. Katayama, "$O(\log_2 m)$ Iterative Algorithm for Multiplicative Inverse in GF (2^m)", *IEEE Intl. Symposium on Information Theory* (*ISIT2000*), p.449, 2000.
7) 和田知久,「共通かぎ暗号AES用SubBytes変換回路設計仕様書」,『Design Wave Magazine』, pp.151-155, 2003年11月号. http://www.ie.u-ryukyu.ac.jp/~wada/design04/spec_j.html
8) C. Paar, "A New Architecture for a Parallel Finite Field multiplication with Low Complexity Based on Composite Fields", *IEEE Trans. on Computers*, Vol. 45, No.7, pp.856-861, 1996.
9) 森岡澄夫,『HDLによる高性能ディジタル回路設計』, CQ出版, 2002年.

Dr. S. Morioka
◆著者プロフィール◆
Dr. S. Morioka. 電話関連会社，コンピュータ関連会社を経て，家電メーカで暗号，エラー訂正，画像処理などのIPの研究開発とSOCへの組み込みを行っています．みなさんが日ごろ使っている何かに，私の回路が入っているかもしれませんね．morioka@fb3.so-net.ne.jp