暗号アルゴリズム「MARS」
詳細評価（HW実装評価）レポート

２００１年1月１９日

目次

１．アルゴリズム概要

MARSは Carolynn Burwick、Don Coppersmith、Edward D'Avignon、Rosario Gennaro、Shai Halevi、Charanjit Jutla、Stephen M. Matyas Jr.、Luke O'Connor、Mohammad Peyravian、David Safford、Nevenko Zunic により1997年に設計された128ビットブロック暗号であり、AESの最終審査対象5候補のひとつに選定されている。鍵サイズは128ビット以上の可変長をサポートしている。

MARS は比較的複雑なアルゴリズム構成をとっている。データの暗号化の流れは、鍵の whitening に引き続いて、8 段の unkeyed forward mixing、8 段の keyed forward core、8 段の keyed backward core、8 段の unkeyed backward mixing、鍵の whitening となっている。またこれらの中で利用されるコンポーネントもさまざまである。unkeyed mixing rounds では、8 ビット入力 32 ビット出力の S-boxes と算術加算と排他的論理和演算が用いられている。keyed core rounds ではこれに加えて、鍵との 32 ビット乗算、鍵との 32 ビット加算ならびにデータ依存回転シフト演算が用いられる。

これらの演算は 32 ビット単位のソフトウエア実装に適している。一方 32 ビット乗算やデータ依存回転シフトの利用は、プラットフォームによっては速度の低下をまねくことがある。特にハードウェアでは、MARS のスループット（平文入力から暗号文出力までの時間）は非常に低速であり、しかも実装面積が大きくなる。このように MARS は結果的に AES の標準プラットフォームであった 32 ビットプロセッサにターゲットを絞った暗号であるということができる。

なお、MARS は AES 投稿後に一度アルゴリズムの変更をおこなっている。これは変更前の版に等価鍵が存在したこと、また IC カード実装には向かないことに対応したもので、例えば必要 RAM サイズを削減する工夫がなされた。これから見ても MARS はその設計時には IC カードでの実装が意図されていなかったことがわかる。さらに最近になって、MARS の S-box は設計者が設定した設計基準を満たしていないことが発見されている。この事実が MARS のセキュリティに与える影響は少ないようであるが、設計ミスであったことは否定できない。

２．アルゴリズムMARSの説明

２．１　データランダマイズ部

図１は、MARS のデータランダマイズ部を表している。一般的な Feistel 構造の場合は、暗号化と復号化において、完全に共通のデータランダマイズ部を利用することが可能であるが、MARS の場合は、若干異なる構成となっている。ただし、SPN 構造のように暗号化と復号化の処理構成が大きく異なる構成ではないため、異なる論理の部分だけを選択する方法で対処可能である。

データランダマイズ部は、図１のように、最初に拡大鍵との 32 ビット毎の加算を行い、Forward Mixing を 2 回繰り返し、Forward transformation(keyed)を 4 回繰り返し、Backward transformation(keyed) を 4 回繰り返し、Backward Mixing を 2 回繰り返し実行して、最後に拡大鍵との 32 ビット毎の減算を行うアルゴリズムである。この処理は、鍵サイズが 128 ビット、192 ビット、256 ビットいずれの場合においても同様である。

データランダマイズ部で使用される Forward Mixing は、図 2 のように、128 ビットの入力データを 32 ビット毎 4 つに分割して上位から D[3],D[2],D[1],D[0]とする。まず、D[0]の下位 8 ビットを関数 S0 に入力し、その結果を D[1]に排他的論理和する。D[0]を 8 ビット右ローテーションシフトした結果の下位 8 ビットを関数 S1 に入力し、その結果を D[1]に加算する。D[0]を再度 8 ビット右ローテーションシフトした結果の下位 8 ビットを関数 S0 に入力し、その結果を D[2]に加算する。D[0]を再度 8 ビット右ローテーションシフトした結果の下位 8 ビットを関数 S1 に入力し、その結果を D[3]に排他的論理和する。最後に、D[3]を D[0]に加算を行う。ここまでが Forward Mixing の処理の 1/4 である。次に、上記の処理の D[0]を D[1]、D[1]を D[2]、D[2]を D[3]、D[3]を D[0]に置き換えて実行する。その後、もう 2 回同様に D[0]を D[2]に、D[0]を D[3]のように１つずつ処理するデータをずらして実行する。ただし、最後の 2 回は D[3]を D[0]に加算するものに対応する処理が存在しないことに注意する。最後に処理された D[3],D[2],D[1],D[0]を連接し、Forward Mixing の 128 ビット出力データとなる。

データランダマイズ部で使用される Forward transformation(keyed)は、図 3 のように、128 ビットの入力データを 32 ビット毎 4 つに分割して上位から D[3],D[2],D[1],D[0]とする。まず、D[0]を関数 E に入力し、その結果の Out1 を D[1]に加算し、Out2 を D[2]に加算し、Out3 を D[3]に排他的論理和する。その後、D[0]は左 13 ビットローテーションシフトを行い D[3]へ、その他は D[1]は D[0]、D[2]は D[1]、D[3]は D[2]に置き換えを行う。ここまでが Forward transformation(keyed)の処理の半分であり、もう一度、同じ処理を繰り返す。

データランダマイズ部で使用される Backward transformation(keyed)は、図 4 のように、128 ビットの入力データを 32 ビット毎 4 つに分割して上位から D[3],D[2],D[1],D[0]とする。まず、D[0]を関数 E に入力し、その結果の Out1 を D[3]に加算し、Out2 を D[2]に加算

し、Out3 を D[1]に排他的論理和する。その後、D[0]は左 13 ビットローテーションシフトを行い D[3]へ、その他は D[1]は D[0]、D[2]は D[1]、D[3]は D[2]に置き換えを行う。ここまでが Forward transformation(keyed)の処理の半分であり、もう一度、同じ処理を繰り返す。

データランダマイズ部で使用される Backward Mixing は、図 7 のように、128 ビットの入力データを 32 ビット毎 4 つに分割して上位から D[3],D[2],D[1],D[0]とする。まず、D[0]の下位 8 ビットを関数 S1 に入力し、その結果を D[1]に排他的論理和する。D[0]を 8 ビット左ローテーションシフトした結果の下位 8 ビットを関数 S0 に入力し、その結果を D[2]から減算する。D[0]を再度 8 ビット左ローテーションシフトした結果の下位 8 ビットを関数 S1 に入力し、その結果を D[3]から減算する。D[0]を再度 8 ビット左ローテーションシフトした結果の下位 8 ビットを関数 S0 に入力し、その結果を D[3]に排他的論理和する。ここまでが Backward Mixing の処理の 1/4 である。次に、上記の処理の D[0]を D[1]、D[1]を D[2]、D[2]を D[3]、D[3]を D[0]に置き換えて実行し、最後に D[2]から D[1]を減算する。その後、もう 2 回同様に D[0]を D[2]に、D[0]を D[3]のように 1 つずつ処理するデータをずらしてずらして実行する。ただし、最後の 2 回のうち、D[0]を D[2]に置き換えた時は、D[0]を D[1]に置き換えた場合と同様に、D[3]から D[0]を減算する処理が存在することに注意する。最後に処理された D[3],D[2],D[1],D[0]を連接し、Forward Mixing の 128 ビット出力データとなる。

Forward transformation(keyed)と Backward transformation(keyed)で使用される関数 E は、図 5 のように、32 ビット入力 96 ビット出力の拡張関数である。入力 32 ビットを 13 ビット左ローテーションシフトし、拡大鍵と 32 ビット乗算したものを Out3 とする。入力 32 ビットと拡大鍵の加算したものを Out2 とする。Out2 の下位 9 ビットを関数 S に入力し、その結果を Out1 とする。Out3 を 5 ビット左ローテーションシフトし、Out1 に排他的論理和する。Out2 を Out3 の下位 5 ビットの値ビットだけ左ローテーションシフトを行う。再度、Out3 を 5 ビット左ローテーションシフトし、Out1 に排他的論理和する。最後に、Out1 を Out3 の下位 5 ビットの値ビットだけ左ローテーションシフトを行う。以上の処理によって生成された Out1、Out2、Out3 が関数 E の出力となる。なお、関数 E で使用される関数 S は、図 6 のように、関数 S0 と関数 S1 を併せた構成となっている。

暗号化処理と復号化処理の違いに関しては、Forward transformation(keyed)と Backward transformation(keyed)で処理される 32 ビット加算が 32 ビット減算に置き換わるのみである。

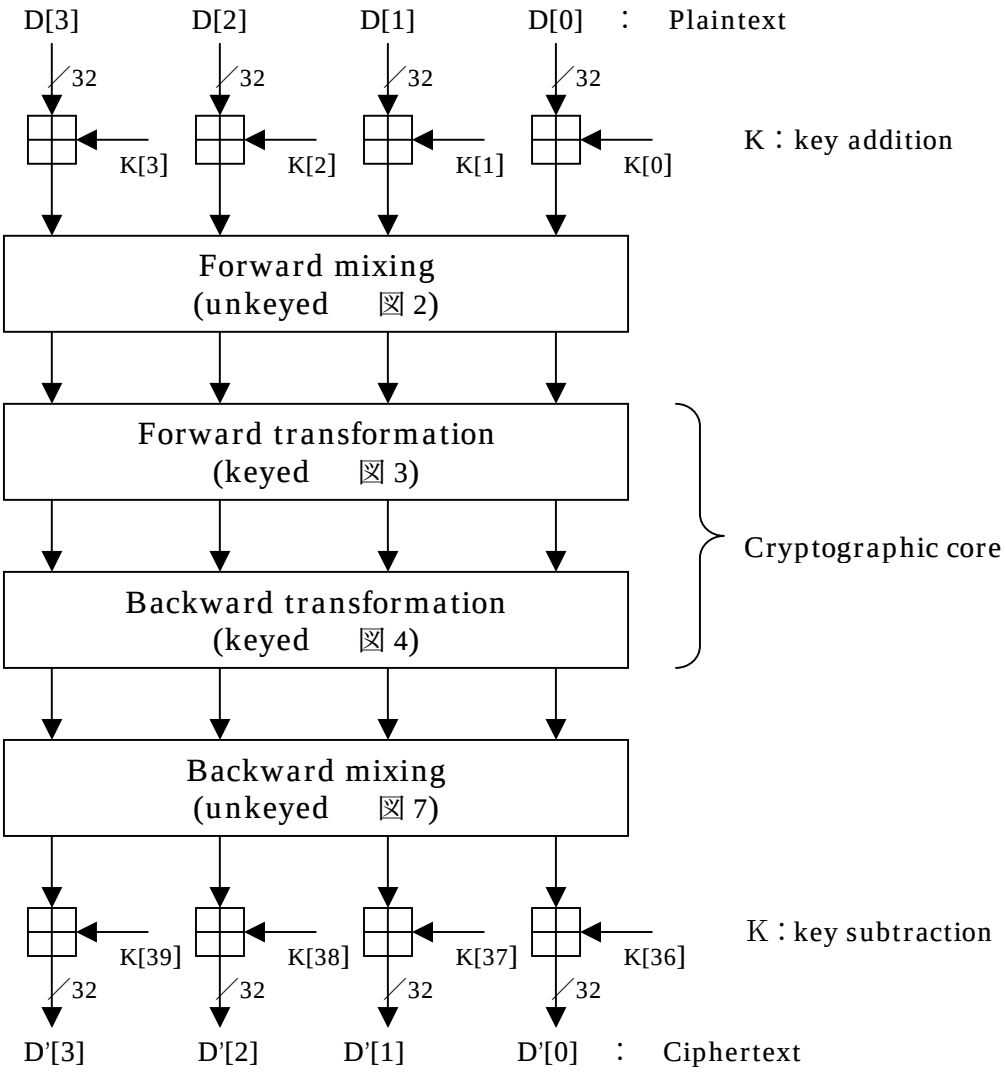


図 1. データランダマイズ部

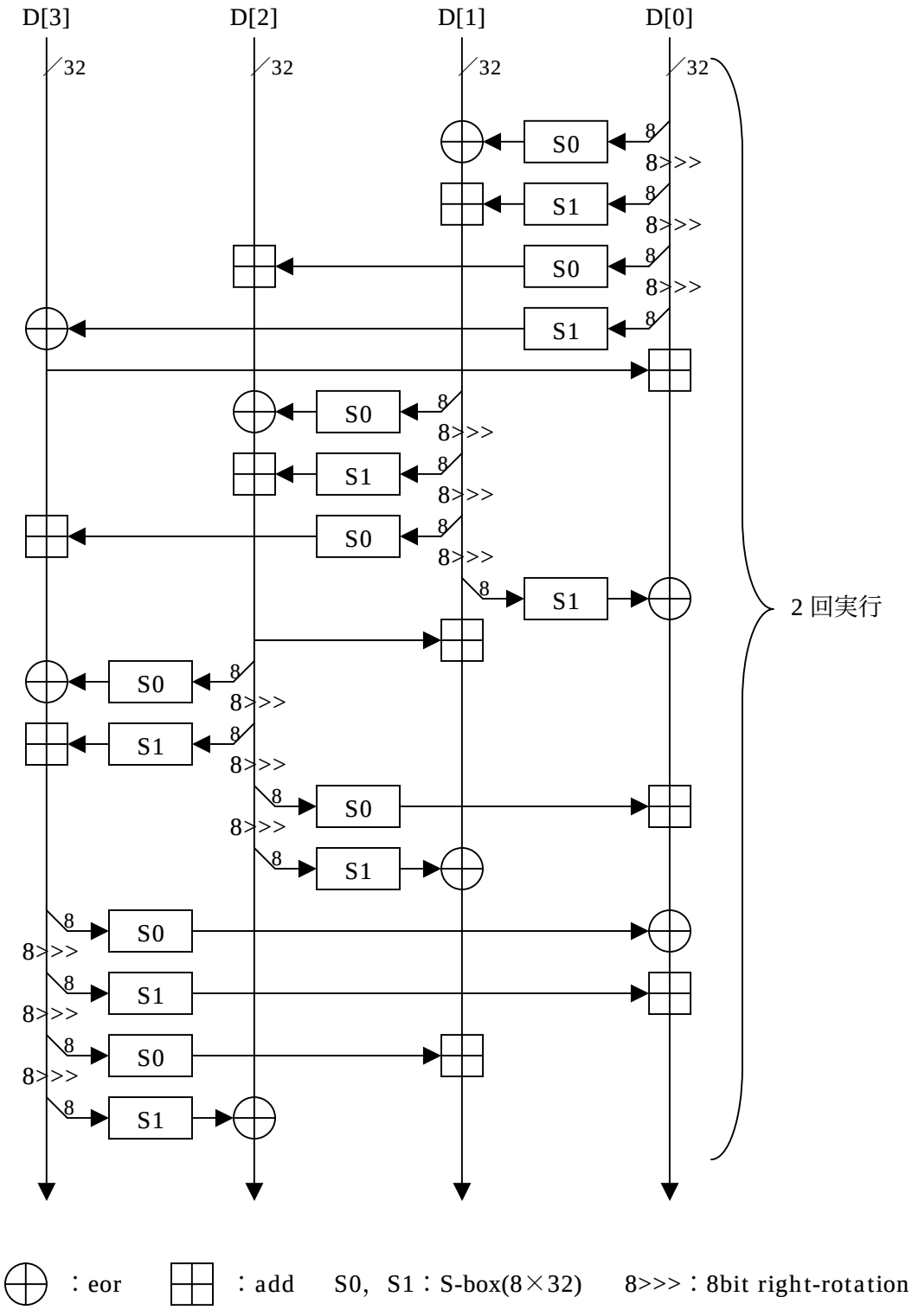
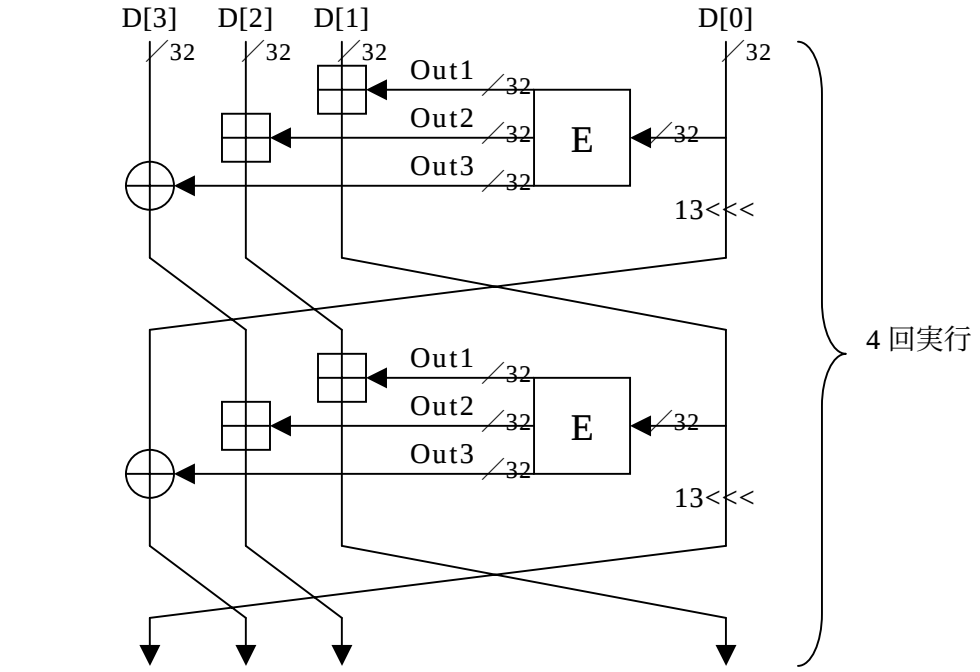
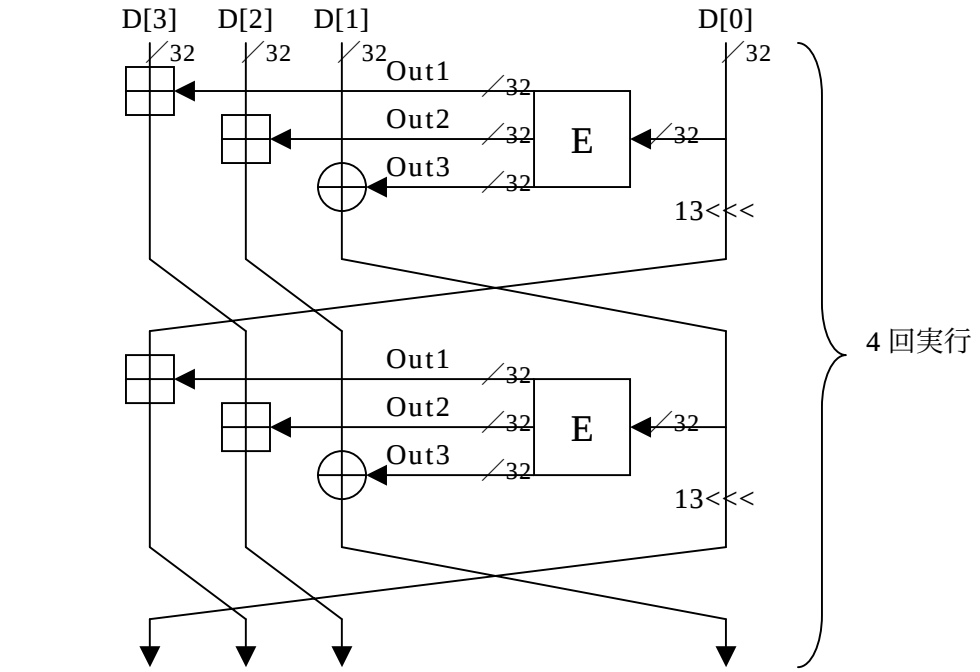


図 2. Forward mixing(unkeyed)



⊕ ：eor　⊞ ：add　E：Expansion function　13<<<：13bit left-rotation

図 3. Forward transformation(keyed)



⊕ ：eor　⊞ ：add　E：Expansion function　13<<<：13bit left-rotation

図 4. Backward transformation(keyed)

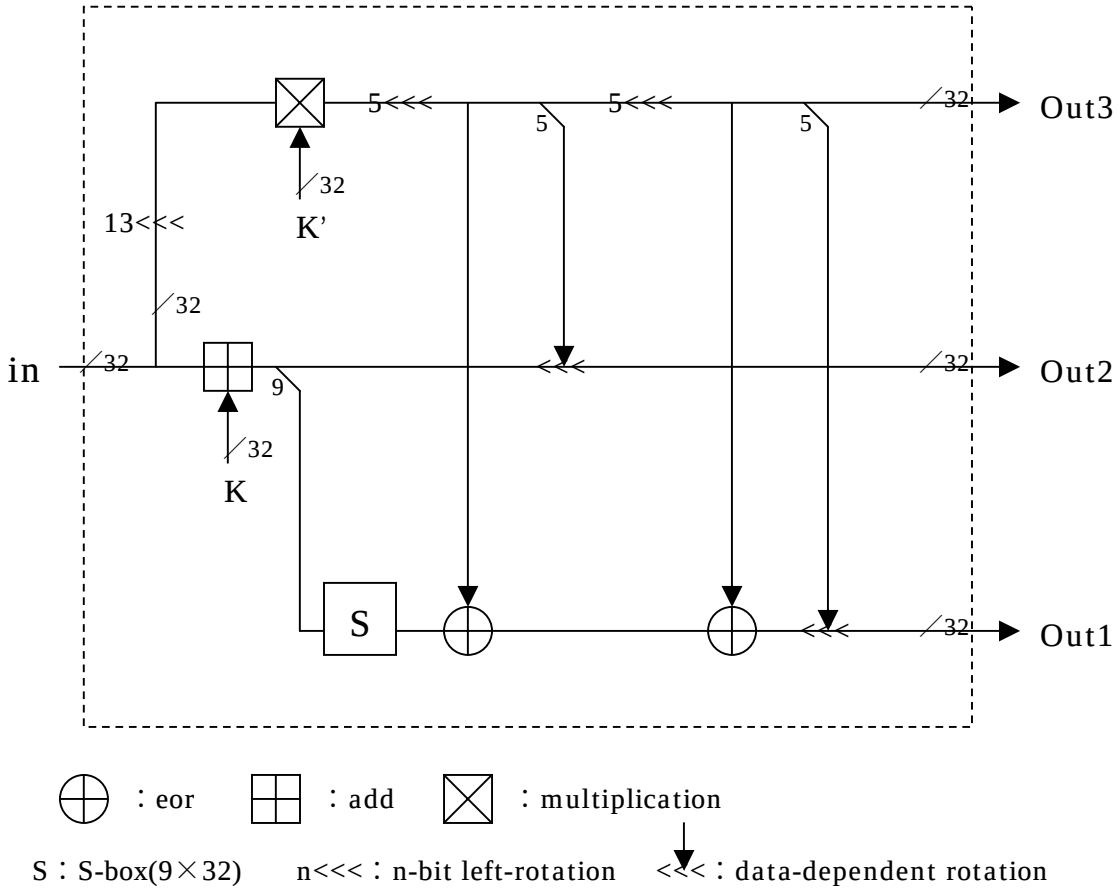
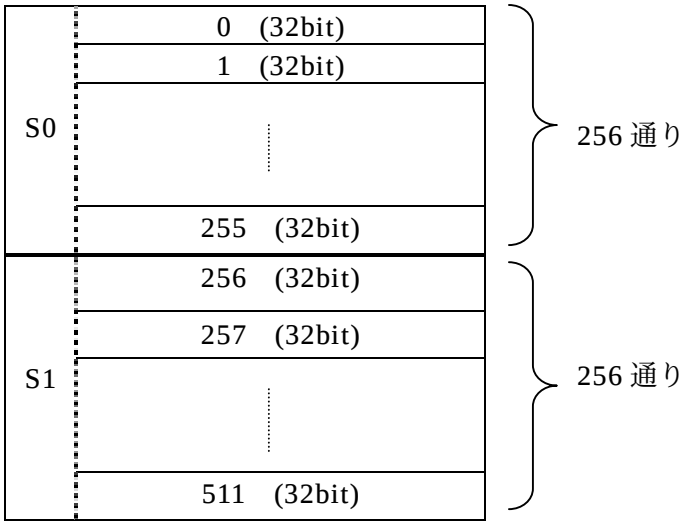


図 5. 関数 E



※関数 S は、9 ビット入力 32 ビット出力であり、512 通りの 32 ビットデータ。 512 通りの最初の 256 通りが関数 S0 で、残りの 256 通りが関数 S1 により、構成されている。

図 6. 関数 S.

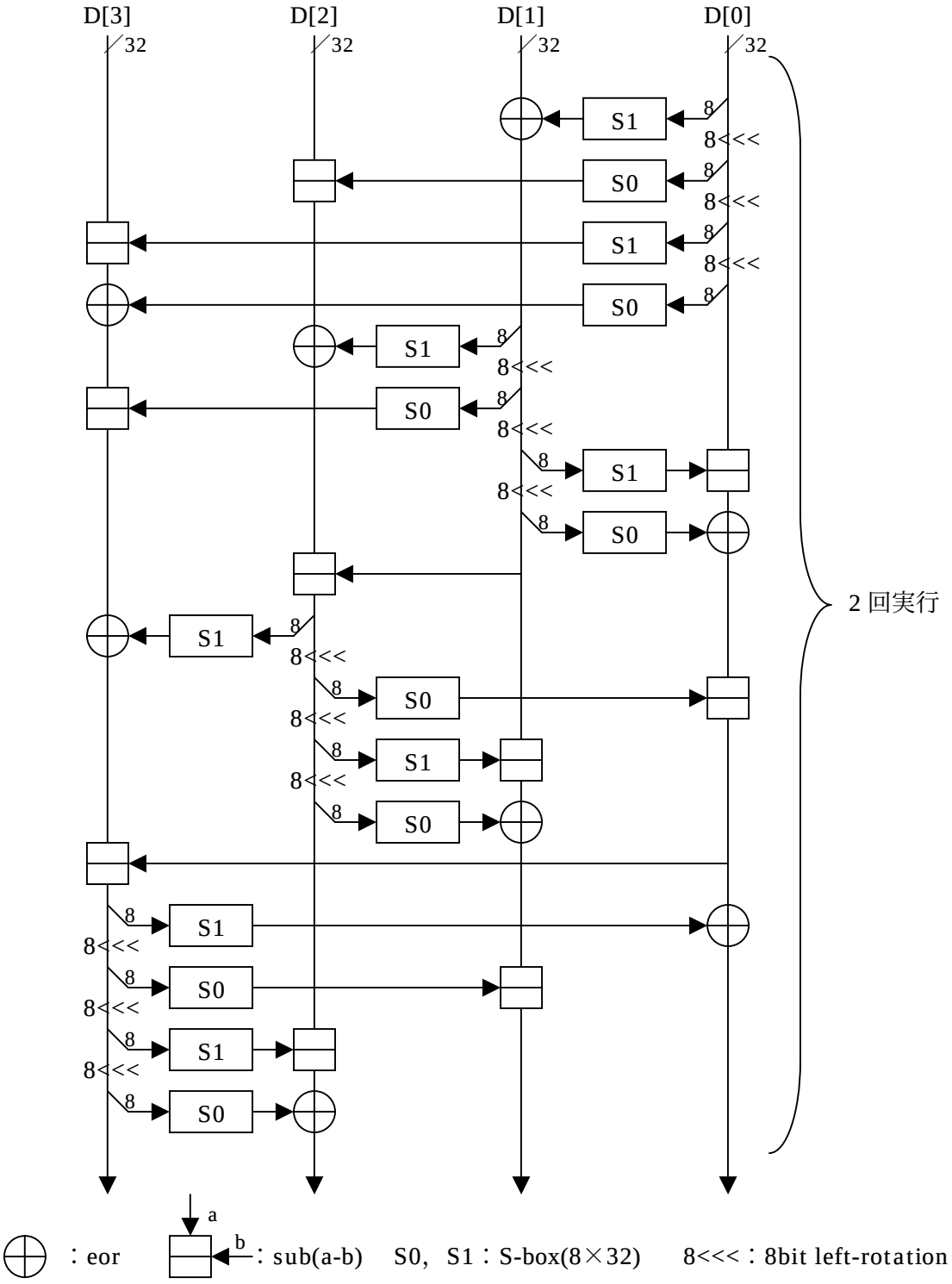


図 7. Backward mixing(unkeyed)

以下に、図6で説明した関数Sの実際の値を記述する。

関数S =

0x09d0c479,0x28c8ffe0,0x84aa6c39,0x9dad7287,0x7dff9be3,0xd4268361,
0xc96da1d4,0x7974cc93,0x85d0582e,0x2a4b5705,0x1ca16a62,0xc3bd279d,
0x0f1f25e5,0x5160372f,0xc695c1fb,0x4d7ff1e4,0xae5f6bf4,0x0d72ee46,
0xff23de8a,0xb1cf8e83,0xf14902e2,0x3e981e42,0x8bf53eb6,0x7f4bf8ac,
0x83631f83,0x25970205,0x76afe784,0x3a7931d4,0x4f846450,0x5c64c3f6,
0x210a5f18,0xc6986a26,0x28f4e826,0x3a60a81c,0xd340a664,0x7ea820c4,
0x526687c5,0x7eddd12b,0x32a11d1d,0x9c9ef086,0x80f6e831,0xab6f04ad,
0x56fb9b53,0x8b2e095c,0xb68556ae,0xd2250b0d,0x294a7721,0xe21fb253,
0xae136749,0xe82aae86,0x93365104,0x99404a66,0x78a784dc,0xb69ba84b,
0x04046793,0x23db5c1e,0x46cae1d6,0x2fe28134,0x5a223942,0x1863cd5b,
0xc190c6e3,0x07dfb846,0x6eb88816,0x2d0dcc4a,0xa4ccae59,0x3798670d,
0xcbfa9493,0x4f481d45,0xeafc8ca8,0xdb1129d6,0xb0449e20,0x0f5407fb,
0x6167d9a8,0xd1f45763,0x4daa96c3,0x3bec5958,0xababa014,0xb6ccd201,
0x38d6279f,0x02682215,0x8f376cd5,0x092c237e,0xbfc56593,0x32889d2c,
0x854b3e95,0x05bb9b43,0x7dcd5dcd,0xa02e926c,0xfae527e5,0x36a1c330,
0x3412e1ae,0xf257f462,0x3c4f1d71,0x30a2e809,0x68e5f551,0x9c61ba44,
0x5ded0ab8,0x75ce09c8,0x9654f93e,0x698c0cca,0x243cb3e4,0x2b062b97,
0x0f3b8d9e,0x00e050df,0xfc5d6166,0xe35f9288,0xc079550d,0x0591aee8,
0x8e531e74,0x75fe3578,0x2f6d829a,0xf60b21ae,0x95e8eb8d,0x6699486b,
0x901d7d9b,0xfd6d6e31,0x1090acef,0xe0670dd8,0xdab2e692,0xcd6d4365,
0xe5393514,0x3af345f0,0x6241fc4d,0x460da3a3,0x7bcf3729,0x8bf1d1e0,
0x14aac070,0x1587ed55,0x3afd7d3e,0xd2f29e01,0x29a9d1f6,0xefb10c53,
0xcf3b870f,0xb414935c,0x664465ed,0x024acac7,0x59a744c1,0x1d2936a7,
0xdc580aa6,0xcf574ca8,0x040a7a10,0x6cd81807,0x8a98be4c,0xaccea063,
0xc33e92b5,0xd1e0e03d,0xb322517e,0x2092bd13,0x386b2c4a,0x52e8dd58,
0x58656dfb,0x50820371,0x41811896,0xe337ef7e,0xd39fb119,0xc97f0df6,
0x68fea01b,0xa150a6e5,0x55258962,0xeb6ff41b,0xd7c9cd7a,0xa619cd9e,
0xbcf09576,0x2672c073,0xf003fb3c,0x4ab7a50b,0x1484126a,0x487ba9b1,
0xa64fc9c6,0xf6957d49,0x38b06a75,0xdd805fcd,0x63d094cf,0xf51c999e,
0x1aa4d343,0xb8495294,0xce9f8e99,0xbffcd770,0xc7c275cc,0x378453a7,
0x7b21be33,0x397f41bd,0x4e94d131,0x92cc1f98,0x5915ea51,0x99f861b7,
0xc9980a88,0x1d74fd5f,0xb0a495f8,0x614deed0,0xb5778eea,0x5941792d,
0xfa90c1f8,0x33f824b4,0xc4965372,0x3ff6d550,0x4ca5fec0,0x8630e964,

0x5b3fbbd6,0x7da26a48,0xb203231a,0x04297514,0x2d639306,0x2eb13149,
0x16a45272,0x532459a0,0x8e5f4872,0xf966c7d9,0x07128dc0,0x0d44db62,
0xafc8d52d,0x06316131,0xd838e7ce,0x1bc41d00,0x3a2e8c0f,0xea83837e,
0xb984737d,0x13ba4891,0xc4f8b949,0xa6d6acb3,0xa215cdce,0x8359838b,
0x6bd1aa31,0xf579dd52,0x21b93f93,0xf5176781,0x187dfdde,0xe94aeb76,
0x2b38fd54,0x431de1da,0xab394825,0x9ad3048f,0xdfea32aa,0x659473e3,
0x623f7863,0xf3346c59,0xab3ab685,0x3346a90b,0x6b56443e,0xc6de01f8,
0x8d421fc0,0x9b0ed10c,0x88f1a1e9,0x54c1f029,0x7dead57b,0x8d7ba426,
0x4cf5178a,0x551a7cca,0x1a9a5f08,0xfcd651b9,0x25605182,0xe11fc6c3,
0xb6fd9676,0x337b3027,0xb7c8eb14,0x9e5fd030,
0x6b57e354,0xad913cf7,0x7e16688d,0x58872a69,0x2c2fc7df,0xe389ccc6,
0x30738df1,0x0824a734,0xe1797a8b,0xa4a8d57b,0x5b5d193b,0xc8a8309b,
0x73f9a978,0x73398d32,0x0f59573e,0xe9df2b03,0xe8a5b6c8,0x848d0704,
0x98df93c2,0x720a1dc3,0x684f259a,0x943ba848,0xa6370152,0x863b5ea3,
0xd17b978b,0x6d9b58ef,0x0a700dd4,0xa73d36bf,0x8e6a0829,0x8695bc14,
0xe35b3447,0x933ac568,0x8894b022,0x2f511c27,0xddfbcc3c,0x006662b6,
0x117c83fe,0x4e12b414,0xc2bca766,0x3a2fec10,0xf4562420,0x55792e2a,
0x46f5d857,0xceda25ce,0xc3601d3b,0x6c00ab46,0xefac9c28,0xb3c35047,
0x611dfee3,0x257c3207,0xfdd58482,0x3b14d84f,0x23becb64,0xa075f3a3,
0x088f8ead,0x07adf158,0x7796943c,0xfacabf3d,0xc09730cd,0xf7679969,
0xda44e9ed,0x2c854c12,0x35935fa3,0x2f057d9f,0x690624f8,0x1cb0bafd,
0x7b0dbdc6,0x810f23bb,0xfa929a1a,0x6d969a17,0x6742979b,0x74ac7d05,
0x010e65c4,0x86a3d963,0xf907b5a0,0xd0042bd3,0x158d7d03,0x287a8255,
0xbba8366f,0x096edc33,0x21916a7b,0x77b56b86,0x951622f9,0xa6c5e650,
0x8cea17d1,0xcd8c62bc,0xa3d63433,0x358a68fd,0x0f9b9d3c,0xd6aa295b,
0xfe33384a,0xc000738e,0xcd67eb2f,0xe2eb6dc2,0x97338b02,0x06c9f246,
0x419cf1ad,0x2b83c045,0x3723f18a,0xcb5b3089,0x160bead7,0x5d494656,
0x35f8a74b,0x1e4e6c9e,0x000399bd,0x67466880,0xb4174831,0xacf423b2,
0xca815ab3,0x5a6395e7,0x302a67c5,0x8bdb446b,0x108f8fa4,0x10223eda,
0x92b8b48b,0x7f38d0ee,0xab2701d4,0x0262d415,0xaf224a30,0xb3d88aba,
0xf8b2c3af,0xdaf7ef70,0xcc97d3b7,0xe9614b6c,0x2baebff4,0x70f687cf,
0x386c9156,0xce092ee5,0x01e87da6,0x6ce91e6a,0xbb7bcc84,0xc7922c20,
0x9d3b71fd,0x060e41c6,0xd7590f15,0x4e03bb47,0x183c198e,0x63eeb240,
0x2ddbf49a,0x6d5cba54,0x923750af,0xf9e14236,0x7838162b,0x59726c72,
0x81b66760,0xbb2926c1,0x48a0ce0d,0xa6c0496d,0xad43507b,0x718d496a,
0x9df057af,0x44b1bde6,0x054356dc,0xde7ced35,0xd51a138b,0x62088cc9,

```
0x35830311,0xc96efca2,0x686f86ec,0x8e77cb68,0x63e1d6b8,0xc80f9778,
0x79c491fd,0x1b4c67f2,0x72698d7d,0x5e368c31,0xf7d95e2e,0xa1d3493f,
0xdcd9433e,0x896f1552,0x4bc4ca7a,0xa6d1baf4,0xa5a96dcc,0x0bef8b46,
0xa169fda7,0x74df40b7,0x4e208804,0x9a756607,0x038e87c8,0x20211e44,
0x8b7ad4bf,0xc6403f35,0x1848e36d,0x80bdb038,0x1e62891c,0x643d2107,
0xbf04d6f8,0x21092c8c,0xf644f389,0x0778404e,0x7b78adb8,0xa2c52d53,
0x42157abe,0xa2253e2e,0x7bf3f4ae,0x80f594f9,0x953194e7,0x77eb92ed,
0xb3816930,0xda8d9336,0xbf447469,0xf26d9483,0xee6faed5,0x71371235,
0xde425f73,0xb4e59f43,0x7dbe2d4e,0x2d37b185,0x49dc9a63,0x98c39d98,
0x1301c9a2,0x389b1bbf,0x0c18588d,0xa421c1ba,0x7aa3865c,0x71e08558,
0x3c5cfcaa,0x7d239ca4,0x0297d9dd,0xd7dc2830,0x4b37802b,0x7428ab54,
0xaeee0347,0x4b3fbb85,0x692f2f08,0x134e578e,0x36d9e0bf,0xae8b5fcf,
0xedb93ecf,0x2b27248e,0x170eb1ef,0x7dc57fd6,0x1e760f16,0xb1136601,
0x864e1b9b,0xd7ea7319,0x3ab871bd,0xcfa4d76f,0xe31bd782,0x0dbeb469,
0xabb96061,0x5370f85d,0xffb07e37,0xda30d0fb,0xebc977b6,0x0b98b40f,
0x3a4d0fe6,0xdf4fc26b,0x159cf22a,0xc298d6e2,0x2b78ef6a,0x61a94ac0,
0xab561187,0x14eea0f0,0xdf0d4164,0x19af70ee
};
```

２．２　鍵スケジュール部

鍵スケジュール部は、図8のように、非常に複雑な処理で構成されている。最初にInitial expansionを行い、40個の32ビットデータを生成する。生成されたデータに対して、関数Sで処理した結果と隣のデータと32ビット加算し、その結果を9ビット左ローテーションシフトするという演算を合計280回実行する。その結果を、ReorderingとFixing weak key-wordsと呼ばれる処理を実行して、40個の32ビット拡大鍵を生成する。

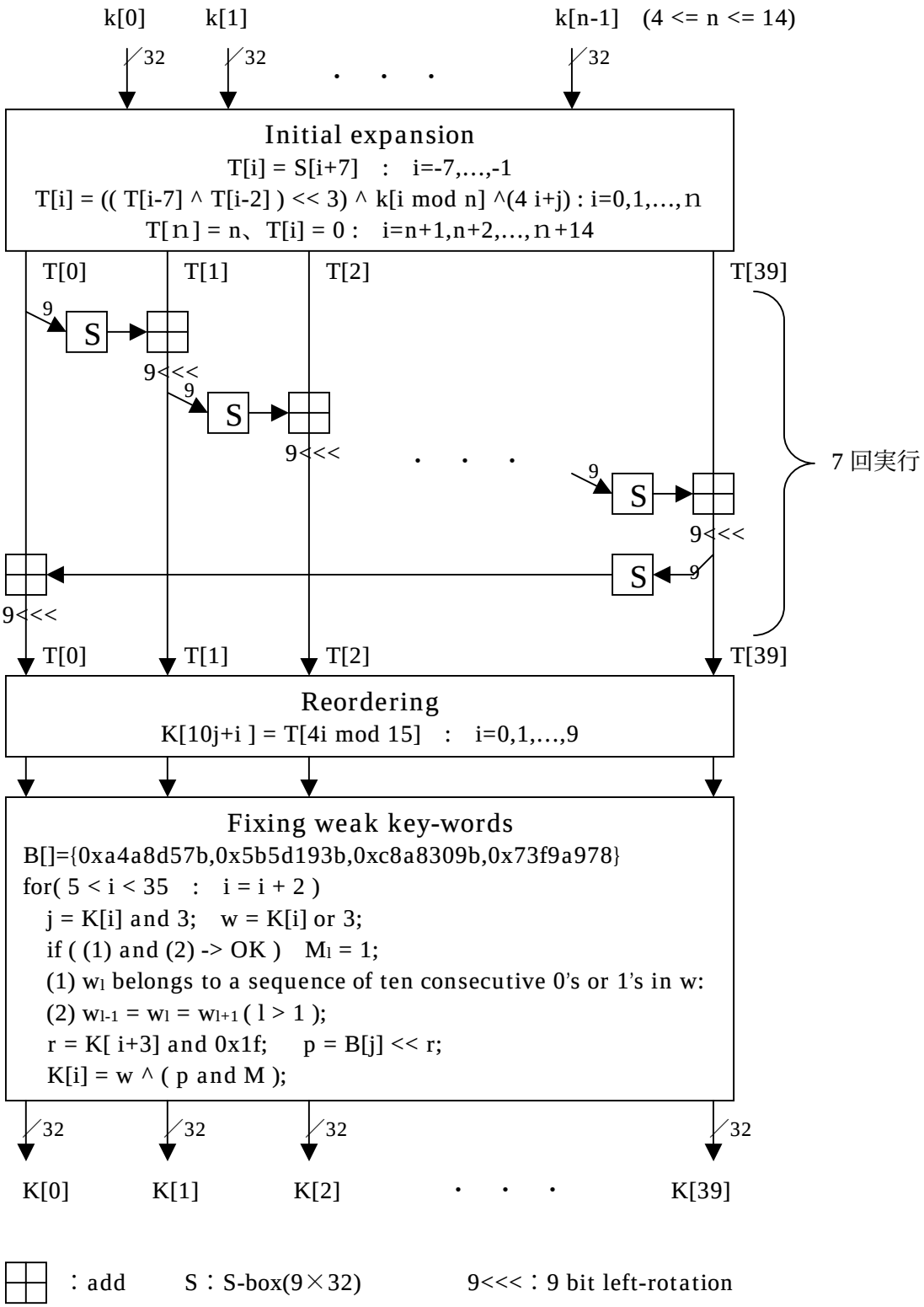

k[0] k[1] k[n-1] (4 <= n <= 14)
32
Initial expansion
T[i] = S[i+7] : i=-7,…,-1
T[i] = ((T[i-7] ^ T[i-2]) << 3) ^ k[i mod n] ^(4 i+j) : i=0,1,…,n
T[n] = n、T[i] = 0 : i=n+1,n+2,…,n+14
T[0] T[1] T[2] T[39]
9
S
9<<<
7 回実行
Reordering
K[10j+i] = T[4i mod 15] : i=0,1,…,9
Fixing weak key-words
B[]={0xa4a8d57b,0x5b5d193b,0xc8a8309b,0x73f9a978}
for(5 < i < 35 : i = i + 2)
j = K[i] and 3; w = K[i] or 3;
if ((1) and (2) -> OK) Ml = 1;
(1) wl belongs to a sequence of ten consecutive 0's or 1's in w:
(2) wl-1 = wl = wl+1 (l > 1);
r = K[i+3] and 0x1f; p = B[j] << r;
K[i] = w ^ (p and M);
K[0] K[1] K[2] K[39]
：add S：S-box(9×32) 9<<<：9 bit left-rotation

図 8. 鍵スケジュール部

３．評価方針

今回の性能評価は、SBOX、乗算器、加算器、ラウンド関数等といった基本的な機能を実現している個々のパーツ(以下、primitive)の評価(回路規模、処理速度等)を実施することによりアルゴリズム全体の性能を見積る方法で評価を行い、また、個々の primitive においては、実際の LSI 作成に則した条件を付加して評価を行うことを前提に行った。なお、本報告で用いる評価環境は、我々が H/W 評価経験のある三菱 0.35μm CMOS ASIC ライブラリを用い、回路記述には Verilog-HDL、Synthesis には Design Compiler を用い、回路規模(ゲート数)およびクリティカルパス長、処理速度等の性能見積を行った。

３.１　回路規模・性能の見積方法

理想的と考えられる評価は、個々のアルゴリズムに対し、公平な評価指標のもとに、最大限の optimize（SBOX の小型化および最速化、および SBOX やラウンド関数のような複数回使用される回路の実装個数と繰り返し回数のトレードオフ等の検討）を試みて回路規模、性能を見積ることである。しかし、この評価方法では、開発時間、開発資源（論理合成ツール等のソフトウェア、コンピュータ等のハードウェアおよび開発者等）が大量に必要となり、本詳細評価期間では、全てのアルゴリズムを公平な指標のもとに optimize することは困難である。そこで特定のアルゴリズムだけを最適化することはせずに、アルゴリズムを全て実装（回路規模は大きくても構わない）し、クリティカルパス長の短縮（処理速度向上）を重視して評価を行った。

そこで今回の評価方法は、以下の通りに行った。

① 原則として、評価対象アルゴリズム全体(鍵スケジュールおよびすべての内部鍵レジスタを含む)をすべて H/W 実装することを前提とする。
② 入力鍵サイズは 128bit とし、性能評価時に、暗号化、復号部分と拡大鍵生成部分の分離を容易にするため、拡大鍵は全てレジスタに格納する。
③ SBOX、乗算器、加算器、ラウンド関数等といった基本的な機能を実現している個々のパーツ(以下、primitive)の評価(回路規模、処理速度等)を実施する。
④ primitive の使用個数の総和を求め回路規模とする。
⑤ クリティカルパス上に存在する primitive の遅延値すべての総和を求めクリティカルパス長とする。

⑥ 「処理速度」を以下のように定義する。

$$\mathrm{Throughput\ [Mbps]} = \frac{\text{ブロックサイズ（128[bit]）}}{\{\text{暗号化（または復号）回路のクリティカルパス [ns]}\}}$$

⑦ テーブル参照型 SBOX の実装は、論理合成ツール（Design Compiler）のみを用い optimize を行う。

⑧ 我々がハードウェア評価経験のあるツールを用いる。

・開発言語：Verilog-HDL

・シミュレータ：Verilog-XL

・デザインライブラリ：三菱 0.35μm CMOS ASIC ライブラリ

・ 論理合成および性能評価：Design Compiler（version 1998.08）

⑨ 評価するときの環境条件は、Worst ケースとする。

※注…Worst ケースとは、ハードウェアが動作する環境が劣悪、かつ、LSI 製造時のバラツキ度合いが悪い方の状態にあることを意味する。一般的に、ハードウェアの性能を議論するときには、議論の対象となっているハードウェアの動作を保証するという意味で、Worst ケースで性能を論じることがほとんどである。よって、これ以降は、何も断りが無い場合は全て Worst ケースで議論を行う。

3.2 設計上の留意点

primitive の回路設計は、Verilog-HDL で記述し、設計する際の細かなブロック分けは、可能な限りアルゴリズム開発者のブロック分けに準じるように留意し行った。

3.3 Synthesis（論理合成）条件

Synthesis(論理合成)ツールは Synopsys 社の Design Compiler(version 1998-08)を用い、すべてのアルゴリズムおよび primitive に同一の条件を付加した。この条件は、速度を最大、かつ、実際の LSI 作成に則するようなものである。以下に条件を示す。

① 加算・減算・乗算回路は、Synopsys の Design Ware Basic Library 内にある最速なライブラリを用いる。

・ 32bit 加算器は Synopsys の Design Ware Basic Library 内にある DW01_add(cla)[Carry look-ahead synthesis model]を用いる。

・ 32bit 減算器は Synopsys の Design Ware Basic Library 内にある

DW01_sub(cla)[Carry look-ahead synthesis model]を用いる。

・ 32bit 乗算器は Synopsys の Design Ware Basic Library 内にある DW02_mult(csa)[Carry-save array synthesis model]を用いる。

② その他の条件は、基本的に default 値を用いる。ただし、fanout に関する条件は以下の理由で付加する。

○fanout 条件を全く付加しないことによる問題点

ある primitive の入力端子に fanout 条件が付けられていないとその入力端子は駆動能力が「∞」のドライバより駆動されていることになり、『primitive レベルでの評価からアルゴリズム全体を試算し評価する結果』と『アルゴリズム全体のレベルでの評価する結果』でかなりの違いが出てきてしまう。

実際の LSI 設計では、各 primitive の負荷状況(fanout 等)やドライバの駆動能力は、キャラクタライズ（characterize）という手法を用い、各入出力ピンの負荷状況を自動的に設定する方法、もしくは、各入出力端子に fanout 条件を付加する方法等を用いて LSI を設計、作成している。そこで、今回は、アルゴリズム全てを実装せずに、primitive レベルから性能を見積る方法を行うため、characterize 手法は使用せず、以下のように入出力端子に fanout 設定を行う。

① primitive の入力ピンの fanin が 1 になるように条件をつける。

② primitive の出力から最大 40 個の primitive が並列に接続されても耐えうるように、primitive の出力ピンの fanout は 40 になるように条件をつける。

４．各 primitive の見積り

評価方法に従って評価を実行するために、MARS の構造を詳細にチェックし、各 primitive の見積りを行った。ここでは、回路規模や速度に関しては考慮せずに、論理的にどのような構成のものがいくつあるかと言うことについて見積もりを行っている。

４．１　データランダマイズ部

表１にあるように、データランダマイズ部では、Forward Mixing と Backward Mixing がそれぞれ 2 個、Forward transformation(keyed)と Backward transformation(keyed) がそれぞれ 4 個の 12 段構成となっている。その他には、最初と最後の部分の拡大鍵との 32 ビット加算と 32 ビット減算が 4 個ずつ存在している。この構成の中には、暗号化処理、復号化処理の両方が含まれている。

Forward Mixing は、32 ビット排他的論理和が 8 個、32 ビット加算が 10 個、関数 S0 と関数 S1 が 8 個ずつ、32 ビットローテーションシフトが 12 個で構成されている。Backward Mixing は、32 ビット排他的論理和が 8 個、32 ビット減算が 10 個、関数 S0 と関数 S1 が 8 個ずつ、32 ビットローテーションシフトが 12 個で構成されている。Forward transformation(keyed)と Backward transformation(keyed)は、それぞれ 32 ビット排他的論理和が 2 個、32 ビット加算が 4 個、32 ビット減算が 4 個、関数 E が 2 個、32 ビットローテーションシフトが 2 個で構成されている。

関数 E は、32 ビット排他的論理和が 2 個、32 ビット加算、32 ビット乗算、関数 S、32 ビットローテーションシフトと 32 ビット可変ローテーションシフトがそれぞれ 2 個で構成されている。

表 1. データランダマイズ部の各 primitive の見積り

データランダマイズ部	Forward Mixing	**2 個**
	Forward transformation(keyed)	**4 個**
	Backward transformation(keyed)	**4 個**
	Backward Mixing	**2 個**
	32 ビット加算	**4 個**
	32 ビット減算	**4 個**

Forward Mixing	32 ビット EXOR	8 個
	32 ビット加算	10 個
	関数 S0	8 個
	関数 S1	8 個
	32 ビット LOT	12 個
Forward transformation (keyed)	32 ビット EXOR	2 個
	32 ビット加算	4 個
	32 ビット減算	4 個
	関数 E	2 個
	32 ビット LOT	2 個
関数 E	32 ビット EXOR	2 個
	32 ビット加算	1 個
	32 ビット乗算	1 個
	関数 S	1 個
	32 ビット LOT	2 個
	32 ビット可変 LOT	2 個

Backward Mixing	32 ビット EXOR	8 個
	32 ビット減算	10 個
	関数 S0	8 個
	関数 S1	8 個
	32 ビット LOT	12 個
Backward transformation (keyed)	32 ビット EXOR	2 個
	32 ビット加算	4 個
	32 ビット減算	4 個
	関数 E	2 個
	32 ビット LOT	2 個

４．２　鍵スケジュール部

表 2 にあるように、鍵スケジュール部に関しては、Initial expansion、Reordering、Fixing weak key-words と 32 ビット加算、関数 S、32 ビットローテーションシフトがそれぞれ 280 個ずつで構成されている。なお、Initial expansion、Reordering、Fixing weak key-words の 3 つの処理に関しては、この構成を 1 つの primitive として見積りを行うこととする。

鍵スケジュール部の場合は、データランダマイズ部と異なり、暗号化部と復号化部において、共通の鍵スケジュール部を利用することが可能である。

表 2. 鍵スケジュール部の各 primitive の見積り

鍵スケジュール部	Initial expansion	1 個	※暗号化･復号とも同じ構成
	32 ビット加算	280 個	
	関数 S	280 個	
	32 ビット LOT	280 個	
	Reordering	1 個	
	Fixing weak key-words	1 個	

５．ハードウェア評価結果

各 primitive の見積もり結果をもとにして、評価方針通りに評価を行った。

表 3 は、データランダマイズ部の各 primitive の実装結果である。ここで、Round 関数や $Round^{-1}$ 関数の実装に関しては、下位の primitive 毎に設計して積み上げる方式ではなく、実際に Verilog-HDL で記述し、論理合成を行った結果である。なお、32 ビットローテーションシフトは、ハードウェアとしては、結線のみの構成で実現されるため primitive 構成の見積りとしては含んでいない。

表 3. データランダマイズ部の primitive 実装結果

	回路規模[Gate]	クリティカルパス[ns]
32 ビット乗算	5446	23.5
32 ビット加算	859	3.45
32 ビット減算	1087	3.69
32 ビット EXOR	267	1.08
32 ビット可変ローテート	3336	3.33
関数 S0	4464	3.96
関数 S1	4390	4.01
関数 S	8072	4.68
2to1selector*128bit	581	1.98
128 ビットレジスタ	1,029	0.80

表 4 は、鍵スケジュール部の各 primitive の実装結果である。鍵スケジュール部に関しては、暗号化部と復号化部において、共通に利用することが可能である。なお、データランダマイズ部の場合と同様に、32 ビットローテーションシフトは、ハードウェアとしては、結線のみの構成で実現されるため primitive 構成の見積りとしては含んでいない。

表 4. 鍵スケジュール部の primitive 実装結果

	回路規模[Gate]	クリティカルパス[ns]
32 ビット加算	859	3.45
関数 S	8072	4.68

表 3、表 4 の primitive 実装結果をもとに、データランダマイズ部、鍵スケジュール部をアルゴリズムの構成を再現すると、表 5 のような結果となった。

なお、forward transformation と backward transformation は、内部構成はほとんど同じであり、回路規模は同じサイズとなるが、クリティカルパスには大きな違いがある。forward transformation の場合は、関数 E を 2 回通過するパスは、関数 E の出力 Out1 を通る場合であり、32 ビット加算または 32 ビット減算がクリティカルパスになる。それに対して、backward transformation の場合は、関数 E を 2 回通過するパスは、関数 E の出力 Out3 を通る場合であり、32 ビット排他的論理和がクリティカルパスになることから、違いが生じている。

表 5. データランダマイズ部、鍵スケジュール部の評価結果

	回路規模[Gate]	クリティカルパス[ns]
初期加算	3435	3.45
forward mixing	163094	67.92
関数 E	20722	28.99
forward transformation	201356	277.28
backward transformation	201356	196.64
backward mixing	164454	62.86
最終減算	4347	3.69
32 ビット EXOR	267	1.08
128 ビットレジスタ	1029	0.80
小計 (演算部分)	739069	612.64
Initial expansion → Reordering	2191280	2020.32
Fixing weak key-words	61179	3.12
32 ビットレジスタ*40	41147	0.80
2to1selector*128bit*40	23240	1.98
小計 (鍵生成部分)	2316846	2026.22
合計	3055914	612.64

以上の実装結果および見積もりにより、MARS の H/W 詳細評価は、表 6 のような結果となった。なお、アルゴリズムの処理速度を見積もるため、本報告ではクリティカルパスに鍵スケジュールは含まれていないことに注意する。

表 6. RC6 の H/W 詳細評価結果

回路規模[Gate]	クリティカルパス[ns]	処理速度[Mbps]
3055914	612.64	208.93

６．まとめ

MARSについては、5章の評価結果のように、約208Mbpsの処置速度であることから、128ビット暗号としてAESに採用されたRijndael等と比較した場合、相当低速なアルゴリズムであると言える。また、回路規模に関しては、32ビット乗算や32ビット加算、また大きめな関数Sの回路を多数使用していることから、約3.0Mgateと非常に大きいアルゴリズムである。実際の回路実装を考慮した場合においても、Forward Mixing、Backward Mixing、Forward transformation(keyed)、Backward transformation(keyed)等を全て実装することは無く、1個実装し、ループして使用することが考えられるが、データランダマイズ部の回路規模以上に鍵スケジュール部の回路規模が大きいため、約100Kgate以下の実装をすることは非常に困難であることが予想される。従って、処理速度から、高速デジタル回線等の適用も困難であり、かつ、回路規模から、ICカードや携帯端末などの小型なデバイスを使用するアプリケーションは実装困難なアルゴリズムであると考えられる。

－ 以上 －