暗号アルゴリズム「FEAL-NX」
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

FEAL-NXは、1988年にNTTより提案された秘密鍵暗号FEAL（the Fast Data Encipherment Algorithm）の１つのバージョンであり、ブロック長64ビットの暗号アルゴリズムである。FEALはオプションとして、鍵の長さ、段数，鍵パリティの3つがある。鍵の長さは、６４ビットか、128ビットであり，段数Nは，データランダマイズ部の段数を指定し，鍵パリティオプションは，鍵ブロックにおけるパリティビットを使うか、使わないかを選択する。

FEAL-NXは、FEALの鍵パリティビットなし、鍵サイズ128ビット、N回転限定版と位置付けられている。本報告では､FEALが差分解読法に対して、31段まで解読可能であることから、段数を32段として､FEAL-NXの評価を行う。

２．アルゴリズムFEAL-NXの説明

２．１　データランダマイズ部

図 1 は、FEAL-NX のデータランダマイズ部を表している。一般的な Feistel 構造であり、暗号化と復号化は同一の構造によって処理可能である。

暗号化処理は、図 1 のように、入力 64 ビットを 64 ビットの拡大鍵と排他的論理和し、その結果を上位 32 ビットと下位 32 ビットに分割する。分割された上位 32 ビットと下位 32 ビットの排他的論理和したものを、下位 32 ビットとする。これまでの処理が前処理と呼ばれる部分である。

次に、下位 32 ビットを段関数である F 関数で処理行ったものを上位 32 ビットと排他的論理和を行い、上位 32 ビットと下位 32 ビットをスワップさせる。この処理が 1 段の処理であり､32 段の FEAL-NX の場合には､32 回繰り返し実行する。この部分の処理が反復計算と呼ばれる部分である。

最後に 32 段目の場合のみ、上位 32 ビットと下位 32 ビットのスワップを実行せずに、上位 32 ビットと下位 32 ビットを排他的論理和して下位 32 ビットとする。上位と下位を連接した 64 ビットに 64 ビットの拡大鍵と排他的論理和した結果を 64 ビットの出力データとする。この部分の処理が後処理と呼ばれる部分である。

図 2 は、反復計算の F 関数を表しており､32 ビット入力 32 ビット出力の関数である。また、段毎に異なる 16 ビットの拡大鍵が入力される。

F 関数の処理は､まず、入力された 32 ビットを 8 ビット毎４つに分割し、α_0、α_1、α_2、α_3とする。α_1とα_2にそれぞれ 8 ビットの拡大鍵とα_0とα_3の 2 回の排他的論理和を行う。その排他的論理和されたα_1とα_2を入力としたS_1関数で処理を行い､その結果をα_1とする。次に、その処理されたα_1とα_2を入力とした S_0関数で処理を行い、その結果をα_2とする。

その次に、α_0とα_1をS_0関数で処理してα_0、α_2とα_3をS_1関数で処理してα_3とする。この8ビット毎の処理結果α_0、α_1、α_2、α_3を連接し32ビットとして、F関数の出力データとする。このF関数処理で用いられるS_0関数、S_1関数は、図2に示された式のように、256を法とした加算とローテーションシフトにより実現されている。

復号化処理に関しても、暗号化処理と同様の処理を実行することで実現可能である。ただし、各段において処理される F 関数に入力される拡大鍵の順番、前処理と後処理で処理される拡大鍵が逆になっていることに注意する必要がある。

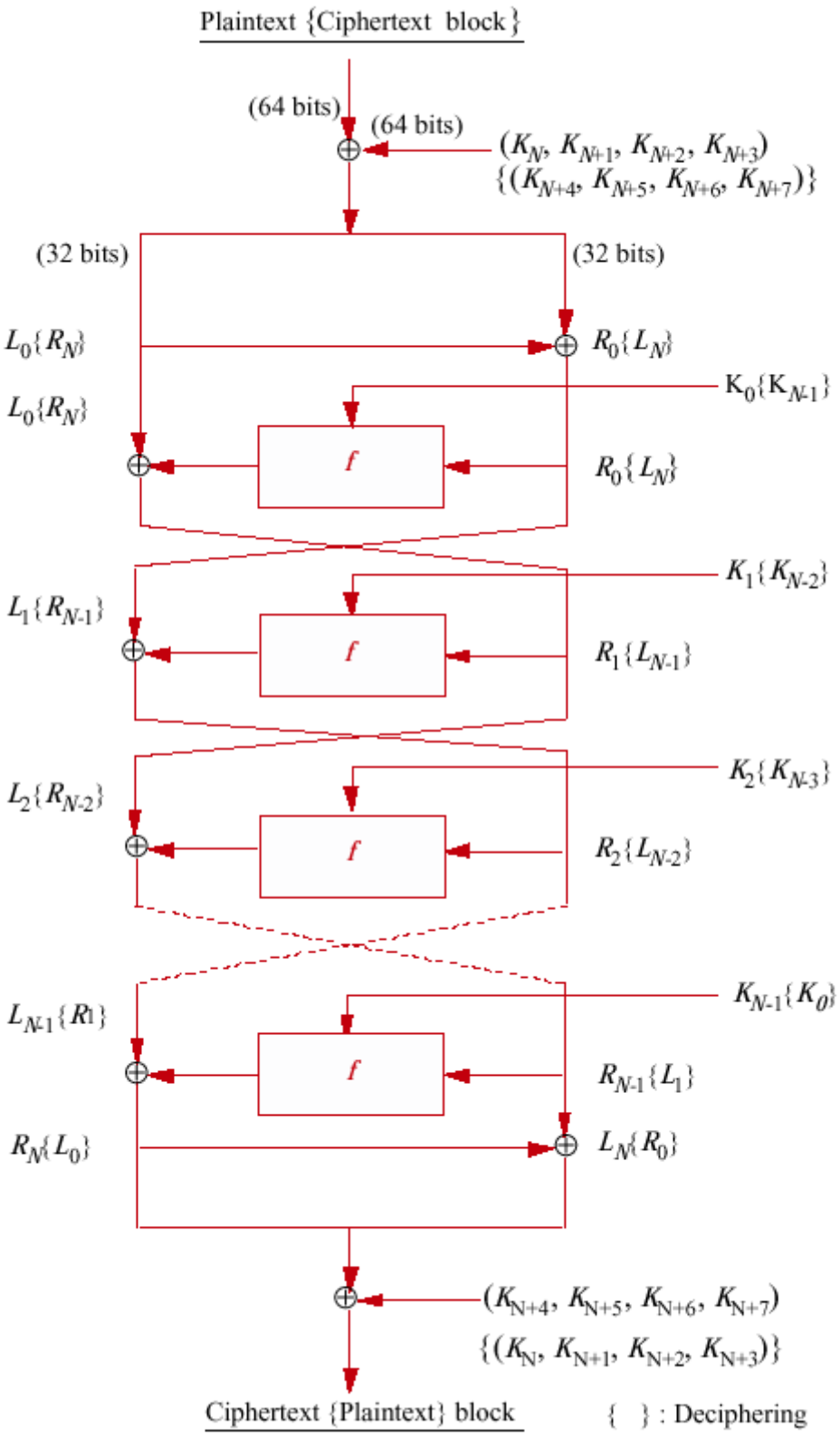
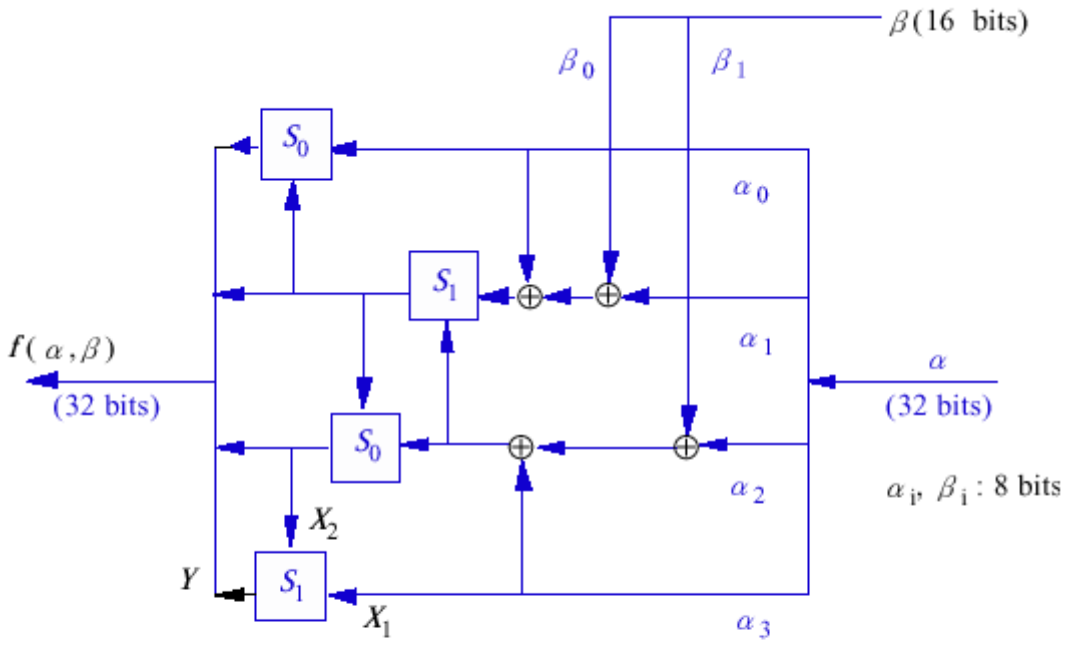


図1．データランダマイズ部



$Y = S_0(X_1, X_2) = \mathrm{Rot2}((X_1 + X_2) \bmod 256)$
$Y = S_1(X_1, X_2) = \mathrm{Rot2}((X_1 + X_2 + 1) \bmod 256)$
Y : output (8 bits), X_1/ X_2 : inputs (8 bits),
Rot2 (Y) : a 2-bit left rotation on 8-bit data Y

図2.F関数

２．２　鍵スケジュール部

FEAL-NXの鍵スケジュール部は、図3のような変形的ではあるがFeistel構造であり、暗号化と復号化は同一の構造によって処理可能である。

鍵スケジュール部は、図 3 のように 128 ビットの暗号化鍵 K を上位 64 ビットと下位 64 ビットに分割する。下位 64 ビットを再び 32 ビット毎に分割して、K_{R1} と K_{R2} とする。また、K_{R1} と K_{R2} の排他的論理和したものを K_{R1}＋K_{R2} と定義する。この K_{R1}＋K_{R2} と K_{R1}、K_{R2} を図 3 のように、順次 1 段目の Q_R、2 段目の Q_R、3 段目の Q_R、･･･のようにして、3 段毎に鍵スケジュール処理に使用する。

最初に分割した上位 64 ビットは、32 ビット毎に分割し、上位 32 ビットと下位 32 ビットに Q_R を排他的論理和したものを鍵スケジュール部用の段関数 Fk 関数に入力し処理する。その結果を、上位 16 ビット、下位 16 ビットに分割して、拡大鍵（K_0,K_1）とする。Fk 関数の出力 32 ビットと下位 32 ビットをスワップし、鍵スケジュール部の 1 段の処理が完了する。

以降、1 段目の処理と同様にして、拡大鍵（K_2,K_3）、拡大鍵（K_4,K_5）、･･･のように、FEAL-NX が 32 段の場合は、20 段の鍵スケジュール処理を実行する。

図 4 は、鍵スケジュール部の段関数 Fk 関数を表しており、64 ビット(32 ビット×2)入力 32 ビット出力の関数である。

Fk 関数の処理は、まず、入力された 32 ビットαを 8 ビット毎４つに分割し、α_0、α_1、α_2、α_3とする。また、入力された 32 ビットβを 8 ビット毎４つに分割し、β_0、β_1、β_2、β_3とする。α_0とα_1、α_2とα_3をそれぞれ排他的論理和し、α_1、α_2とする。その排他的論理和されたα_1とα_2にβ_0を排他的論理和したものを入力とした S_1 関数で処理を行い、その結果をα_1とする。次に、その処理されたα_1にβ_1を排他的論理和したものとα_2を入力とした S_0 関数で処理を行い、その結果をα_2とする。その次に、α_0とα_1にβ_2を排他的論理和したものを S_0 関数で処理してα_0、α_2にβ_3を排他的論理和したものとα_3を S_1 関数で処理してα_3とする。この 8 ビット毎の処理結果α_0、α_1、α_2、α_3を連接し 32 ビットとして、Fk 関数の出力データとする。この Fk 関数処理で用いられる S_0 関数、S_1 関数は、データランダマイズ部で使用されている F 関数と同等の 256 を法とした加算とローテーションシフトにより実現されている。

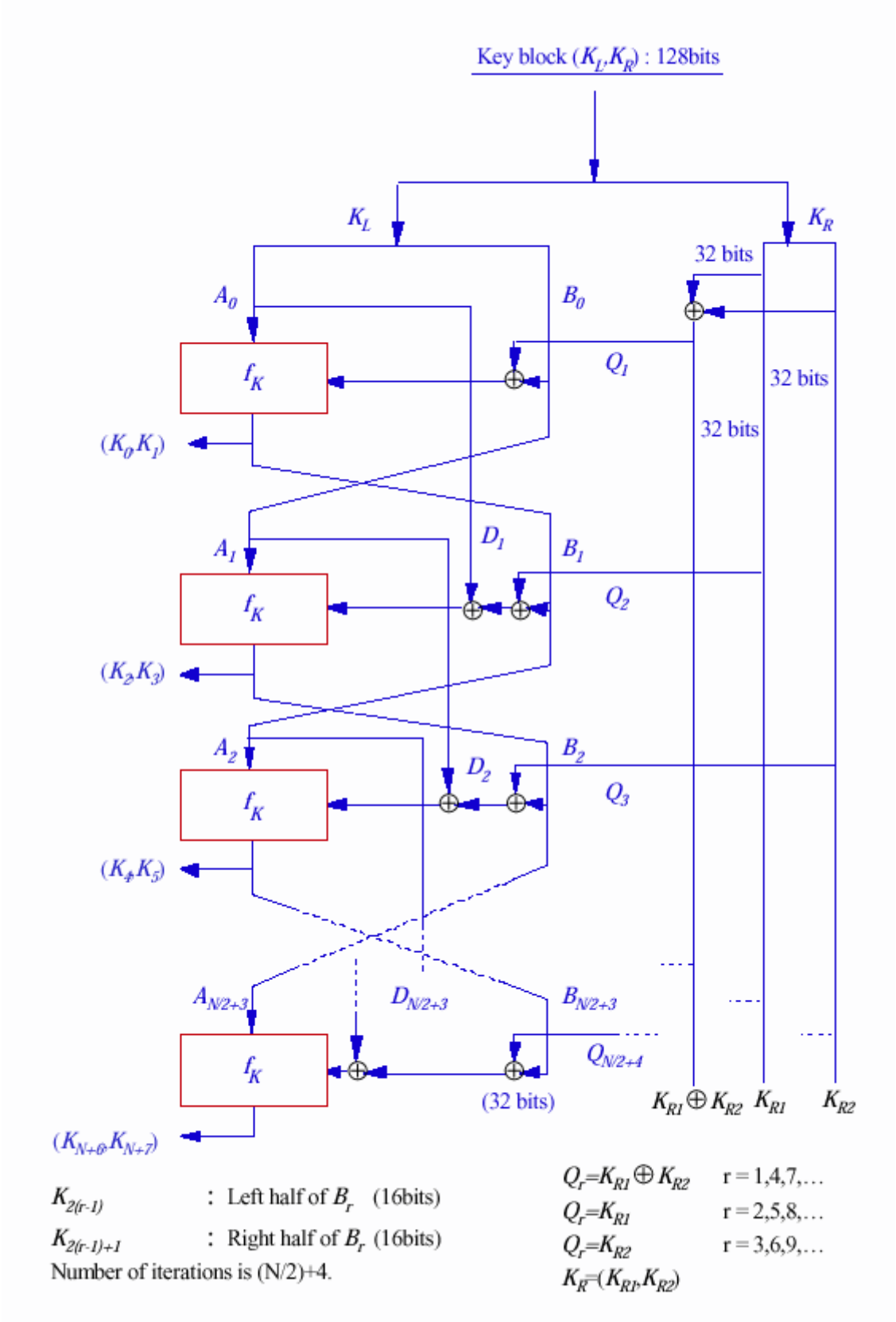
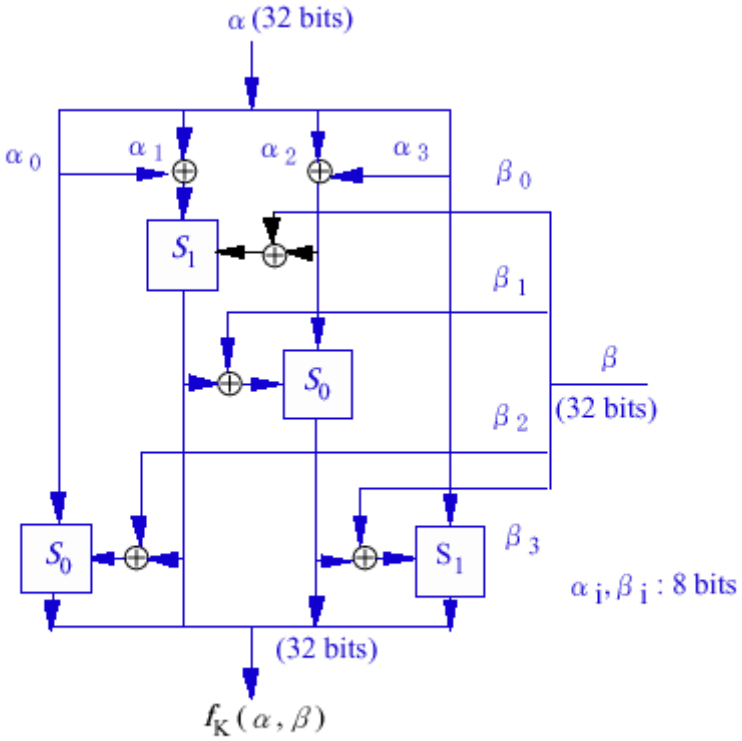

Key block (K_L,K_R) : 128bits
K_L
K_R
32 bits
A_0
B_0
f_K
Q_1
32 bits
32 bits
$(K_0 K_1)$
A_1
D_1
B_1
f_K
Q_2
$(K_2 K_3)$
A_2
B_2
D_2
f_K
Q_3
$(K_4 K_5)$
$A_{N/2+3}$
$D_{N/2+3}$
$B_{N/2+3}$
f_K
$Q_{N/2+4}$
(32 bits)
$K_{R1} \oplus K_{R2}$
K_{R1}
K_{R2}
$(K_{N+6} K_{N+7})$
$K_{2(r-1)}$: Left half of B_r (16bits)
$K_{2(r-1)+1}$: Right half of B_r (16bits)
Number of iterations is (N/2)+4.
$Q_r = K_{R1} \oplus K_{R2}$ r = 1,4,7,…
$Q_r = K_{R1}$ r = 2,5,8,…
$Q_r = K_{R2}$ r = 3,6,9,…
$K_R = (K_{R1}, K_{R2})$

図3．鍵スケジュール部



Note : S_0/S_1 are the same as S_0/S_1 in f-function.

図4．Fk関数

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、FEAL-NX 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に関しては、F 関数が 32 個の 32 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 64 ビット排他的論理和（EXOR）が 2 個、上位 32 ビットと下位 32 ビットの排他的論理和が 2 個、各段の F 関数からの出力と上位 32 ビットの排他的論理和が 32 個存在している。Feistel 構造であるため、復号化処理に関しても同等の構成で処理可能なため、回路規模の大幅な増加はない。

それぞれ、F 関数の内部には、8 ビットの排他的論理和が 4 個と S_0 関数、S_1 関数が 2 個ずつで構成されている。S_0 関数は、8 ビット加算と 2 ビット左ローテートシフト、S_1 関数は、8 ビット加算が 2 個と 2 ビット左ローテートシフトで構成されている。

FEAL-NX のデータランダマイズ部の構成は、大半を排他的論理和と 8 ビットの加算で占めていることが判る。

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	F 関数	32 個	※暗号化･復号とも同じ構成
	32 ビット EXOR	34 個	
	64 ビット EXOR	2 個	

F 関数	8 ビット EXOR	4 個
	S_0 関数	2 個
	S_1 関数	2 個

S_0 関数	8 ビット ADDR	1 個	
	2 ビット LROT	1 個	※2 ビット LROT：2 ビット左ローテートシフト
S_1 関数	8 ビット ADDR	2 個	
	2 ビット LROT	1 個	※2 ビット LROT：2 ビット左ローテートシフト

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、Fk関数が20個の20段構成となっている。その他には、32ビット排他的論理和（EXOR）が40個存在している。

それぞれ、Fk関数の内部には、8ビットの排他的論理和が6個とS_0関数、S_1関数が2個ずつで構成されている。データランダマイズ部のF関数と同様に、S_0関数は8ビット加算と2ビット左ローテートシフト、S_1関数は8ビット加算が2個と2ビット左ローテートシフトで構成されている。

FEAL-NXの鍵スケジュール部の構成は、大半を排他的論理和と8ビットの加算で占めていることが判る。

鍵スケジュール部の場合は、データランダマイズ部と同様に、暗号化処理と復号化処理において、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	Fk関数	20個	※暗号化･復号とも同じ構成
	32ビットEXOR	40個	

FK関数	8ビットEXOR	6個
	S_0関数	2個
	S_1関数	2個

S_0関数	8ビットADDR	1個	
	2ビットLROT	1個	※2ビットLROT：2ビット左ローテートシフト
S_1関数	8ビットADDR	2個	
	2ビットLROT	1個	※2ビットLROT：2ビット左ローテートシフト

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、F 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。

また、F 関数の実装では、F 関数処理後に行う 32 ビット排他的論理和を含んだ構成となっている。Fk 関数の実装でも同様に、Fk 関数処理前に行う 2 回の 32 ビット排他的論理和を含んだ構成となっている。

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]
F 関数	1023	6.94
2to1selector*32bit	226	1.6
64 ビットレジスタ	515	0.80
32bit EXOR	267	1.08

表 4 は、鍵スケジュール部の各 primitive の実装結果である。ここで、Fk 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。

また、Fk 関数の実装では、Fk 関数処理前に行う 2 回の 32 ビット排他的論理和を含んだ構成となっている。

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]
Fk 関数	1259	7.3
2to1selector*32bit	226	1.6
64 ビットレジスタ	515	0.80
32bit EXOR	267	1.08

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。

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]
32 段 F 関数合計	32715	222.08
32bit EXOR	1600	4.32
64 ビットレジスタ	515	0.80
小計（演算部分）	34830	227.20
20 段 Fk 関数合計	25174	146
暗／復号化鍵セレクタ	4520	1.6
64 ビットレジスタ	5147	0.80
小計（鍵生成部分）	34840	148.4
合計	69670	227.2

以上の実装結果および見積もりにより、FEAL-NX 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. FEAL-NX 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]
69670	227.2	281.69

６．まとめ

FEAL-NX については、5 章の評価結果のように、約 281Mbps の処置速度であることから、64 ビット暗号として Triple-DES と比較した場合、低速なアルゴリズムであると言える。これは、クリティカルパス上に並列実行のできない 8 ビット加算が大量に存在するためであると考えられる。回路規模に関しては、Triple-DES と比較した場合、約 70Kgate と小さいアルゴリズムである。実際の回路実装を考慮した場合、F 関数や Fk 関数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、その場合は、約 5Kgate 前後の規模となることが予想される。従って、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にも実用可能なアルゴリズムであると考えられる。

－ 以上 －