卒業論文

Misty1 暗号回路の設計
とハード/ソフト最適分割の検討

氏　　名：和田 智行
学籍番号：2210030453-1
指導教員：山崎 勝弘　教授
提 出 日：2007 年 2 月 19 日

立命館大学理工学部情報学科

内容梗概

本論文では、ハードウェア記述言語 Verilog-HDL を用いたハードウェア設計とハード/ソフト協調設計におけるハード/ソフト分割の最適化の検討について述べる。対象アプリケーションとして、Misty1 暗号をとりあげた。

Misty1 暗号を機能ブロックごとにモジュール分割し、それぞれソフトウェアおよびハードウェアモジュールとして設計した。ハードウェア設計ではパイプラインの並列設計を行い、その有用性を検証した。Misty1 をハードウェア化することで、ソフトウェアの約 743 倍の速度向上が得られることを確認した。また、ハードウェアをパイプライン設計，並列設計することでデータのスループットの向上が得られることも確認した。

ハード/ソフト最適分割の検討では、ソフトウェアの処理の負荷をモジュール毎に計測し、その結果からハード/ソフトの分割パターンを決定し、各パターンにおける検討と考察を行った。また、段数（ループ回数）毎にハード/ソフト分割を行うといった分割パターンについても検討，考察を行った。

目次

図目次

表目次

1. はじめに

近年の半導体集積技術の向上は著しく、1チップに集積可能な論理回路の規模は飛躍的に向上している。集積回路の出現より約30年、当初はごくわずかなトランジスタが集積されただけであったものが、現在では10億トランジスタを集積するまでになり、1チップ上にシステム全体を搭載するSoC（System on a Chip）という回路が登場するまでに至った。

また、携帯電話、PDA、AV機器など電子機器がマルチモーダル化、ネットワーク対応されることにより、機器に搭載される電子回路も多機能化・複雑化・大規模化が進み、SoCに対する需要や期待は大きなものとなってきた。

これに伴い、設計技術も進歩を重ねている。ハードウェアの設計ではHDL（ハードウェア記述言語）の普及により、設計の効率化、検証精度の向上が図られた。シミュレーションではソフトウェア・FPGA・ASICなどの開発により、作業の効率化、コストの低減かがもたらされた。しかし、LSI集積度が増大する割合が、設計生産性が向上する割合を上回っているのが現状である。

その設計生産性の問題の解決策として「ハード/ソフト協調設計」が挙げられる。コストダウンと生産速度の向上を図るためには、要求される個々の機能を、ハードウェアとソフトウェアのどちらで実現するかを的確に設計する必要がある。ハード/ソフト協調設計は、システム設計においてハードウェアとソフトウェアのトレードオフを考慮し設計する手法である。

以上のような背景を踏まえ、本研究ではHDLによる回路の設計を行い、回路におけるハード/ソフト最適分割の検討を行う。回路の設計を行い、パイプライン化・並列化を行うことで、その有用性と実用性がどれほどのものかを検証する。また、ハード/ソフト最適分割のパターンを導き出し、それに対し検討を行い、有用性・実用性がどれほどのものかを検討することを目的とする。

研究の対象とするアプリケーションとしてMisty1を使用する。Misty1とは1995年に三菱電機が開発し、2005年にISO（国際標準化機構）/IEC（国際電気標準会議）にて国際標準規格に採用された秘密暗号鍵方式である。暗号技術は、インターネットをはじめ、コンピュータや携帯電話、映像・画像・音声などのデジタルコンテンツの保護、またデジタル家電にいたるまで、日常の様々な場面で利用することができる。そのため、電子機器に対して暗号回路を搭載しようとする要求は必然である。暗号技術であるMisty1はハードウェア化を前提に設計されているため、高速かつ小規模なハードウェア設計が可能である。また、Misty1はデータ変換の作業において、段数が可変という特徴を持つので、ハード/ソフト分割パターンは数通り考えることができる。そこでMisty1を対象として、ハードウェアの設計・ハード/ソフト最適分割の検討を行う。

本研究では、まず、Misty1アルゴリズムを三菱電機が公開している仕様書に基づきソフトウェアによって実現する。その後、ハード/ソフト最適分割を考慮した上で、Misty1暗号

回路の設計を行う。設計においては、パイプラインの並列設計までを行った。その後、ソフト・ハードにおいて各関数・モジュール毎に、性能や規模などを計測し比較することで、Misty1 におけるハード/ソフト最適分割の検討を行う。また、パイプライン化・並列化を行わなかった回路と行った回路の比較も行い、それらを考慮した上ハード/ソフト最適分割の検討を行う。

本論文では、本章で研究全体の背景と目的を明らかにし、第 2 章で Misty1 の概要とアルゴリズムについて述べる。第 3 章では、ソフトウェアによる設計と機能ブロック毎のハードウェア化の検討を述べる。第 4 章では、Misty1 全体を通したハードウェア設計と、パイプライン・並列設計について述べる。第 5 章では、計測したデータを元に Misty1 におけるハード/ソフト最適分割の検討・考察を行う。

2. Misty1 暗号とハード/ソフト協調設計の検討

2.1 Misty1 の概要

Misty1 とは、1995 年に三菱電機株式会社が開発し、2005 年に ISO/IEC にて国際標準規格に採用された 128 ビットの暗号化鍵をもつ 64 ビットブロック暗号アルゴリズムである。十分な安全性と、ハードウェア・ソフトウェアを問わず、あらゆるプラットフォームでの高速性を両立させたアルゴリズムである。

Misty1 は次の 3 つの設計基準を元に設計されている。[6]

1. 安全性に対する何らかの数値的な証明をもつこと。
2. プロセッサの種類によらず、ソフトウェアで実用的な性能を達成すること。
3. ハードウェア上で十分な高速性を実現すること

ブロック暗号の汎用的な解読法として、現在最も強力とされているものは、差分解読法と線形解読法であるが、Misty1 ではこれらの解読法に対する証明可能安全性を実現することにより、これら解読法に対する安全性を数値的に保証した上で設計されている。また、ソフトウェアでの性能について、Misty1 はあらゆるプロセッサ上で適度な高速性と小型化を実現することを重要視し、マルチプラットフォームに対応できるように設計されている。さらに、最近提案される殆どの暗号アルゴリズムではソフトウェアでの実現を前提とし、ハードウェアでは規模が非常に大きくなってしまうなど、ソフトウェアに比べ速度向上があまり期待できないものが多いが、Misty1 ではハードウェアでの実装を念頭に置き、ハードウェアで性能を発揮できるように最適化されたアルゴリズムとなっている。

また、Misty1 は並列処理構造を強く意識して設計されており、ハードウェア・ソフトウェア共に、並列処理を施すことで高い処理を発揮する。

2.2　Misty1 アルゴリズム

Misty1 は、128 ビットの暗号化鍵を持つ 64 ビットブロック暗号である。暗号処理を行うデータランダマイズ部と、秘密鍵から拡大鍵を導きデータランダマイズ部での処理に必要な鍵の割り当てを行う鍵スケジュール部の 2 つに分けて説明していく。

2.2.1　データランダマイズ部

Misty1 の暗号化処理におけるデータランダマイズの流れを図 2.1に示す。

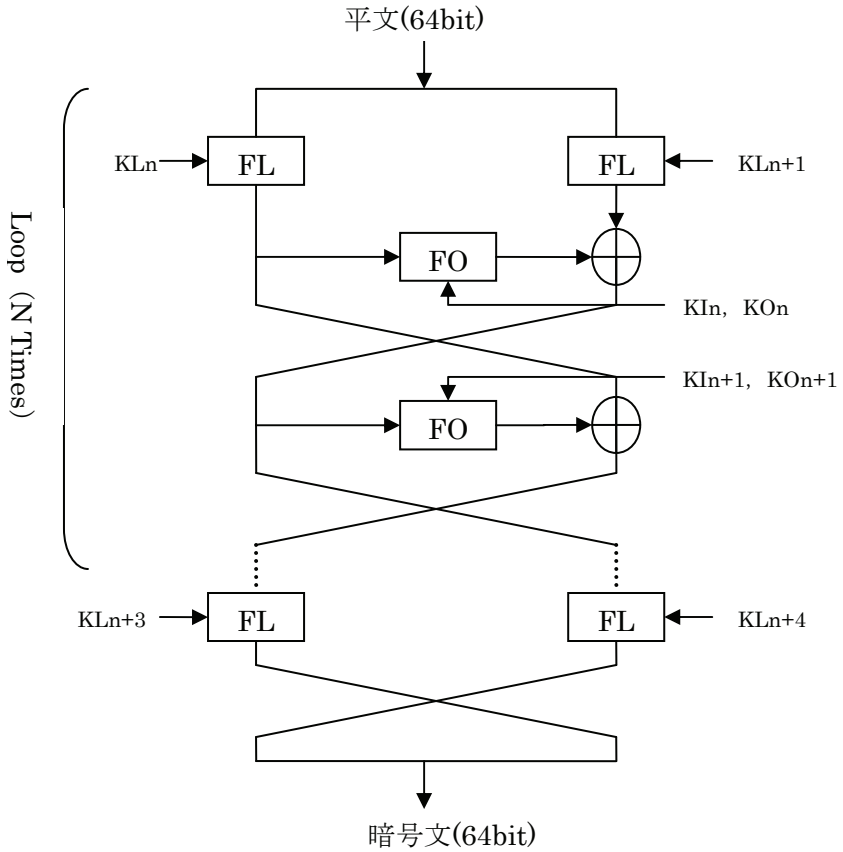

図 2.1　Misty1 暗号化処理データランダマイズ

段数は 4 の倍数をとる限り可変であり、推奨値は 8 段である。なお、図 2.1ではループ 1 回につき 2 段分の処理を表している。入力データ 64 ビットを 32 ビット毎に二分割し、排他的論理和と副関数FL、FOによって変換を行う。FLでは 32 ビットの拡大鍵KLが使用され、FOでは 64 ビットの拡大鍵KOと 48 ビットの拡大鍵KIが使用される。なお、鍵の生成、割り付けについては次節で述べる。次に副関数FL、FOについて説明する。

（1）FL 関数

関数 FL の処理の流れを図 2.2 に示す。

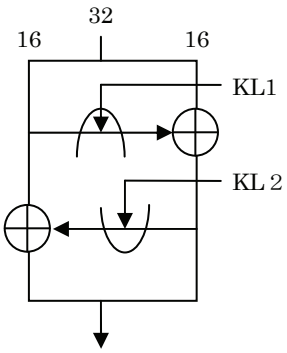

図 2.2　FL 関数

関数 FL は、入力データ 32 ビットを 16 ビット毎に二分割し、排他的論理和と論理積、論理和によって変換を行う。鍵は与えられた鍵 KL の左から i 番目の 16 ビットデータを KLi(i=1、2)として扱い使用する。

（2）FO 関数

関数FOの処理の流れを図 2.3に、FO内で使用される副関数FIを図 **2.4**に示す。

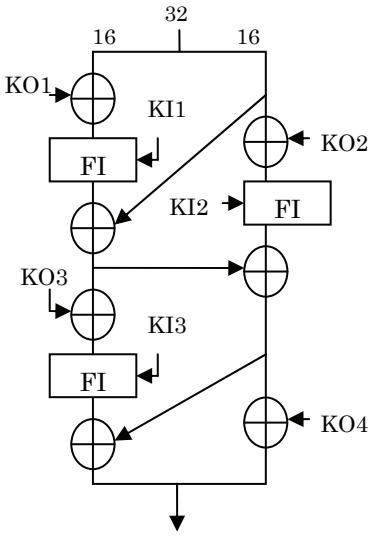

図 2.3　FO 関数

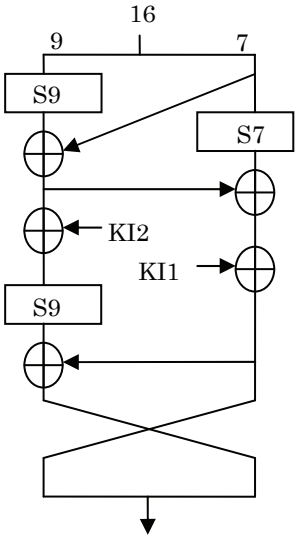

図 2.4　FL 関数

関数 FO は、入力データ 32 ビットを 16 ビット毎に二分割し、排他的論理和と副関数 FI によって変換を行う。鍵は与えられた鍵 KI、KO の左から i 番目の 16 ビットデータをそれぞれ KI（1≦i≦3)、KO（1≦i≦4）として扱い使用する。

関数 FI は、入力 16 ビットを左 9 ビット、右 7 ビットのデータに分割し、排他的論理和

と置換表 S7、S9 によって変換を行う。1 番目と 3 番目の排他的論理和では、7 ビットデータを上位 2 ビットにゼロを付加して 9 ビットデータとして演算を行う。2 番目の排他的論理和では、9 ビットデータの上位 2 ビットを切り捨てて 7 ビットデータとして演算を行う。S7、S9 でのデータ変換では、あらかじめ決められた置換表に従い、入力データに対応した数値データを出力する。鍵は与えられた鍵 KI の左 7 ビットデータを KI1、右 9 ビットデータを KI2 として扱う。なお、S7、S9 におけるデータの置換を表 2.1、2.2 に示す。

2.2.2　鍵スケジュール部

鍵スケジュール部における、秘密鍵からの拡大鍵の生成の流れを図 2.5に示す。

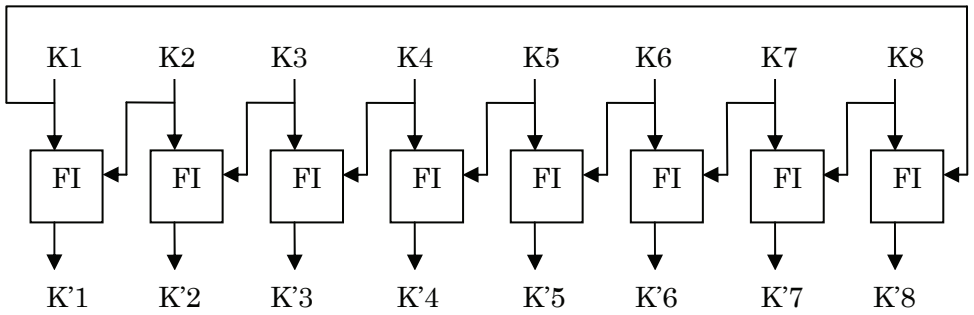

図 2.5　拡大鍵生成アルゴリズム

拡大鍵の生成は、秘密鍵の左からi番目の 16 ビットデータをそれぞれKi (1≦i≦8) とし、図 2.5に示すように、FI関数でデータ変換を行うことで拡大鍵K'i(1≦i≦8)を生成する。このとき、Ki+1 を鍵とすることでFL関数によるデータ変換を行う。

データランダマイズ部への鍵の割り付けでは、秘密鍵と拡大鍵を組み合わせ、KO、KI、KLといった鍵を生成し、鍵を割り付ける。KO、KI、KLと実際の鍵との関係は表 2.1のようになっている。なお、iが 8 を超える場合にはKiとK'iはそれぞれKi-8 とK'i-8 を意味する。

表 2.1　KO、KI、KL と実際の鍵の対応

	KOi1	KOi2	KOi3	KOi3	KIi1	KIi2	KIi3	KLi1	KLi2
Key	K_i	K_i	K_{i+7}	K_{i+4}	K'_{i+5}	K'_{i+1}	K'_{i+3}	(odd)$K_{(i+1)/2}$ (even)$K'_{i/2+2}$	(odd)$K'_{(i+1)/2+6}$ (even)$K_{i/2+4}$

2.3　Misty1 を用いたハード/ソフト協調設計の検討

ハード/ソフト協調設計において、ハードウェアとソフトウェアの最適な設計空間を導き出すための分割探索の流れを図 2.6に示す。

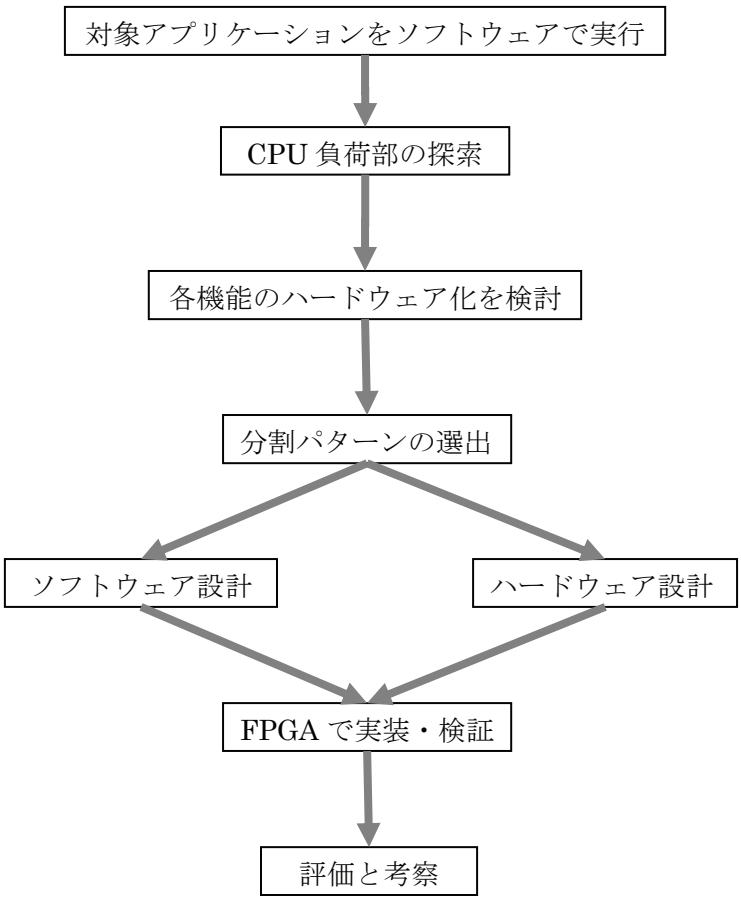

図 2.6　ハード/ソフト分割探索フロー

まず対象アプリケーションを C 言語で記述してアルゴリズムの理解や正しい結果の確認などを行う。次に、ソフトウェア実行環境上において、各モジュールやブロック毎に時間計測し、CPU 負荷を計測し、どのモジュールやブロックの負荷が大きいかを測る。その後、各モジュールのハードウェア化について検討してから、性能や規模について考慮した上で、分割パターンを選び出す。そして実際にソフトウェアとハードウェアの設計を行った上で、FPGA ボードに実装し、検証を行う。

Misty1 における分割パターンの探索にはいくつかの考慮点がある。まず、Misty1 にはモジュールの入れ子構造が多いことに加え、何度も同一のモジュールを使用することである。これを全てハードウェア記述し構成すると、いくつもの下位モジュールを生成することになり、回路規模は莫大になる。またそういった部分的なハードウェア化を行うことは非常に時間を要する作業になり、設計時間などを考慮すると難しいものとなる。次に Misty1 はループ回数が一定ではなく可変だということである。可変である場合、ループ回数によって、ハード/ソフトの分割パターンが変わってくる可能性が出てくる。また、一定回数はソフトウェアのみによる実行で、一定回数はハードウェアのみによる実行といった分割パタ

ーンも考えることができる。今回の研究ではこれらを考慮に入れ分割パターンの選出を行い、検討を行うものとする。

3. Misty1 暗号回路の設計

3.1　C 言語による設計

2 章で述べた Misty1 暗号アルゴリズムに基づき、C 言語を用いてソフトウェア実装を行った。Windows 上の Cygwin 環境において実装と検証を行った。今回設計したソフトウェアは Misty1 における段数を推奨値の 8 段に固定して実装を行った。

設計したソフトウェアを利用してモジュール毎のCPU負荷を計測した。ソフトウェア実行環境は、Core2Duo 2.66GHz、DRAM 3.0GB、WindowsXP、Cygwin上である。各モジュールの実行時間を計測し、ソフトウェア実行にかかったクロック数を算出し負荷部の計測を行った。結果は図 3.1の通りである。

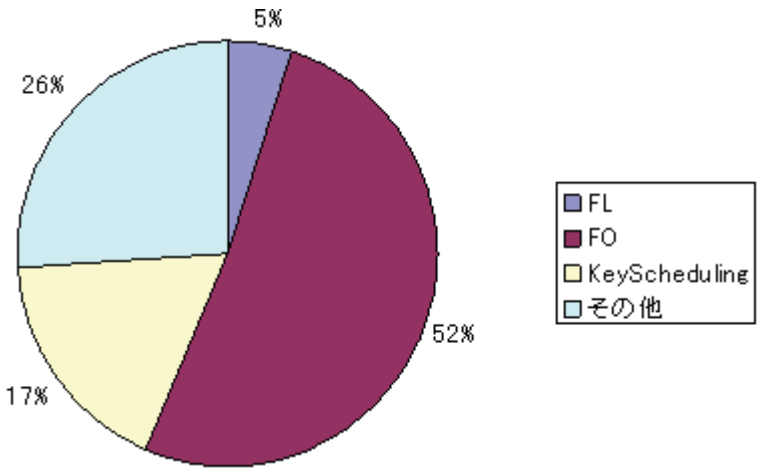

図 3.1　Misty1 暗号システムの各モジュールの CPU 負荷

結果に FI が含まれないのは、FI が FO、KeyScheduling に含まれるからである。FI の FO 中の負荷の割合は約 68%、KeyScheduling 中の負荷の割合は約 66%、ソフトウェア全体を通しての負荷の割合は約 46%である。また、その他はモジュールになってない箇所で主に制御の部分になる。

図 3.1からFL以外のモジュールの負荷が大きいことがわかる。各モジュール、また全体でのFIの負荷の割合が高いことからFIの負荷が高いためFO、KeySchedulingも負荷が高くなっているということが考えられる。これはFI内のテーブル参照部の負荷が高いものと考えられる。また制御部のモジュールが大きな割合を占めていることがわかる。

今回の検証は段数を 8 段に限定しての負荷部探索を行ったが段数が変われば負荷の割合も変わると考えられる。KeyScheduling、制御は段数の変化による影響はない。FL、FO、FI は段数が増えるほど、モジュールの参照回数が多くなるため、負荷が高くなると考えられる。

これらの結果・考察から、FL、FO、FI、またテーブル参照モジュールなどの負荷や使用

頻度の高い部分を優先してハードウェア化することで処理することで、全体の性能を効果的に向上することができると予測する。

3.2　Misty1 暗号回路の設計

Misty1 暗号アルゴリズムをハードウェアモジュールとして実現した。Misty1 暗号アルゴリズムはループ部分とループでない部分に分かれるので、その２つを分けて構成した。ループでない部分というのは厳密にいえばループをする部分の途中までの処理と同一なためループをする部分に含まれるが、今回は後にパイプライン化、並列化をすること、その際の回路規模を考慮してモジュール分割することにした。分割方法を図 3.2に示す。

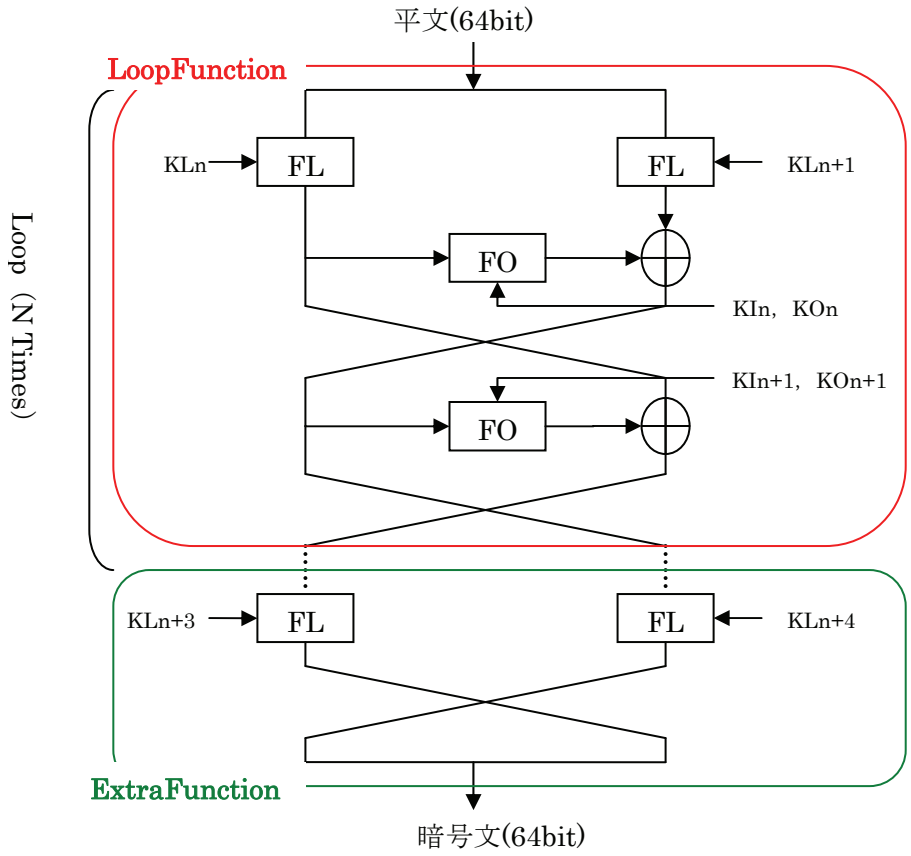

図 3.2　Misty1 のモジュール分割方法

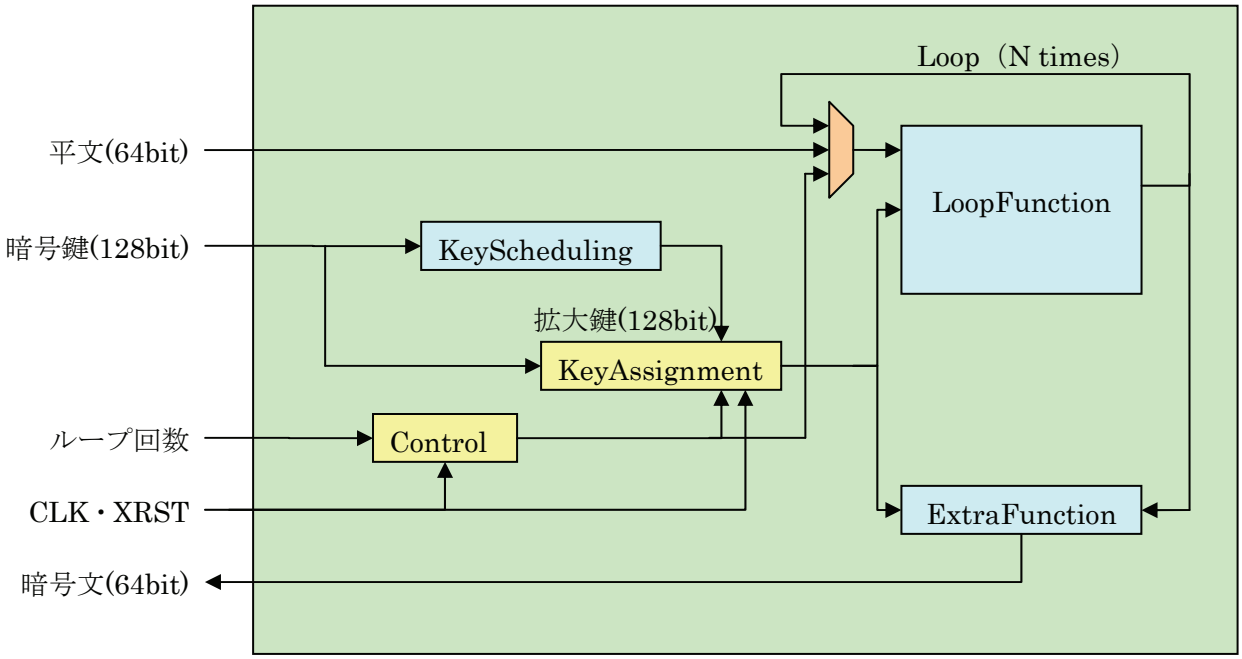

図 3.3　Misty1 暗号回路の構成

ループ部分を LoopFunction、ループをしない部分を ExtraFunction と呼ぶこととする。実際にMisty1 暗号回路の全体的な設計を行った。Misty1 暗号回路の構成を図 3.3に示す。データの流れとしては、被変換データとなる平文がLoopFunctionに渡され、規定回数のデータ変換を行ったのち、ExtraFunctionにてデータ変換を行い暗号文が出力されるといった流れになる。ループをするとき、入力の平文か次にLoopFunctionで参照するデータかを判断するためマルチプレクサを使用して、データの選別を行う。

データランダマイズ処理を行うモジュールに渡す鍵の割付け制御は KeyAssignment で行われる。KeyScheduling 部によって求められた拡大鍵と暗号鍵を KeyAssignment に入力することでデータランダマイズ部に渡す鍵を出力する。データランダマイズに渡す鍵はループするごとに違ったものになるため、KeyAssignment は Control から出力された信号から判断し、データランダマイズ部に渡す鍵を生成する。Control は制御部にあたり、ループ数の状態を管理している。

また、今回の設計では LoopFunction モジュールは Misty1 暗号アルゴリズムの段数構成に照らし合わせると 2 段構成になっているため、ループ回数は Misty1 暗号回路内でのループ回数であって、ループ回数は段数ではない。例えば 8 段の暗号処理を行いたい場合はループ回数への入力は 4 となる。

3.3　各モジュールの説明

3.3.1　LoopFunction モジュール

図 3.4にLoopFunctionモジュールの入出力インタフェースを、信号線の説明を表 3.1に示す。また図 3.5にモジュールの内部構成を示す。

LoopFunction モジュールは組み合わせ回路として設計している。また、このLoopFunction モジュールのみで、2 段構成になっている。Misty1 暗号回路においてループ部分の動作をする。被変換データである入力データ、出力（変換後データ）共にデータ幅は 64 ビットで、データ分割を行い左右のデータを交差させながら、FL モジュール、FO モジュールによってデータ変換を行った後データの結合で出力を得る。

入力の鍵は LoopFunction 内の下位モジュールで使用される全ての鍵をセットで渡している。LoopFunction 内では 32 ビット鍵と 48 ビット鍵を 2 組ずつ生成するため、4 分割されサブモジュールに受け渡され、使用される。

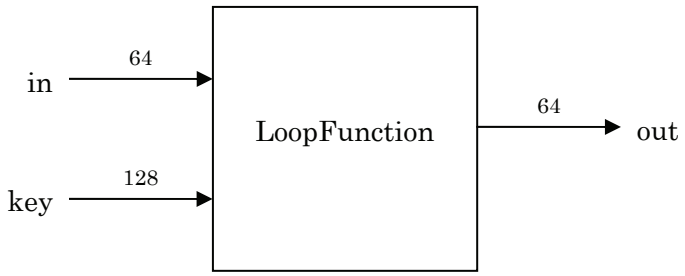

図 3.4　LoopFunction モジュールのインタフェース

表 3.1　LoopFunction モジュールの入出力信号

信号名	方向	幅（bit）	詳細
out	Output	64	変換後データ
in	Input	64	入力データ
key	Input	128	鍵データ

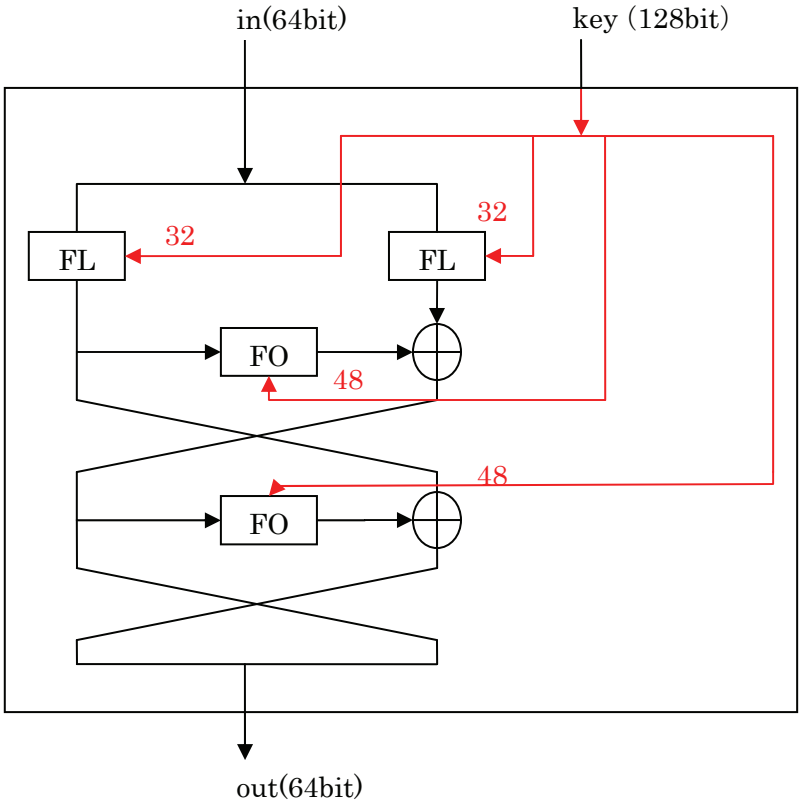

図 3.5　LoopFunction モジュール内部構成

3.3.2　ExtraFunction モジュール

ExtraFunctionモジュールの入出力インタフェースを図 3.6に、信号線の説明を表 3.2に示す。また、図 3.7にExtraFunctionモジュールの内部構成を示す。

ExtraFunction モジュールは組み合わせ回路として設計している。Misty1 暗号回路におけるループでない部分の動作をする。被変換データである入力データ、出力（変換後データ）共にデータ幅は 64 ビットで、データ分割を行い FO モジュールによってデータ変換を行った後、左右のデータを交差させ結合することで出力を得る。

入力の鍵はサブモジュールである FL モジュールで使用される。FL に渡すために 32 ビットに 2 分割して使用される。

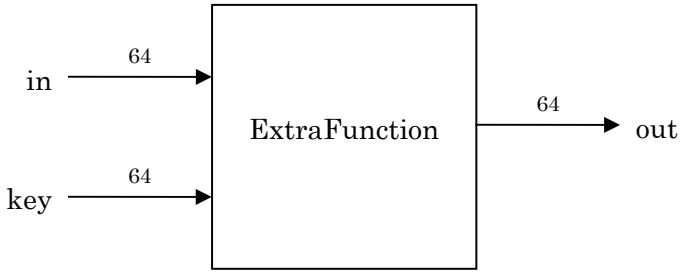

図 3.6　ExtraFunction モジュールのインタフェース

表 3.2　ExtraFunction モジュールの入出力信号

信号名	方向	幅（bit）	詳細
out	Output	64	変換後データ
in	Input	64	入力データ
key	Input	64	鍵データ

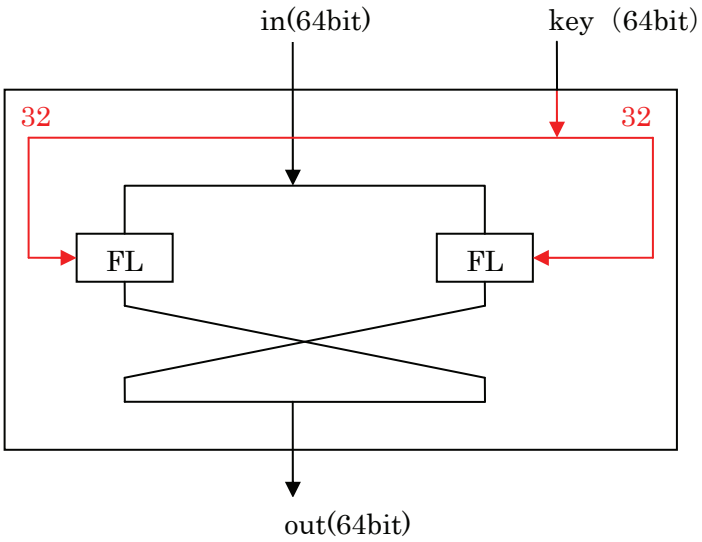

図 3.7　ExtraFunction モジュール内部構成

3.3.3　FL モジュール

FLモジュールの入出力インタフェースを図 3.8に、信号線の説明を表 3.3に示す。FLモジュールでは、入力である被変換データ 32 ビットを、左右 16 ビットずつにデータ分割し、論理和（AND演算）、論理積（OR演算）、排他的論理和（XOR演算）を行いデータ変換が行われる。入力の 32 ビット鍵は左右 16 ビットに分けられ、データ変換の際に使用される。

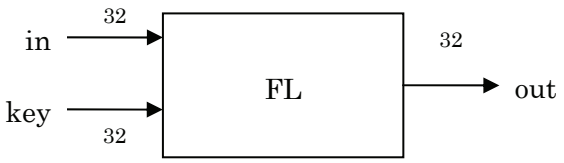

図 3.8　FL モジュールのインタフェース

表 3.3　FL モジュールの入出力信号

信号名	方向	幅（bit）	詳細
out	Output	32	変換後のデータ
in	Input	32	入力データ
key	Input	32	鍵データ

3.3.4　FO モジュール

FOモジュールの入出力インタフェースを図 3.9に、信号線の説明を表 3.4に示す。

FO モジュールでは、入力である被変換データ 32 ビットを、左右 16 ビットずつにデータ分割し、左右に分かれたデータの交差を行いながら、FI モジュールによるデータ変換、排他的論理和（XOR）を行い、データ変換を行う。入力の鍵は 2 種類あり、一方は、下位のモジュールに当たる FI モジュールで使用するためのものであり、FI モジュールに渡される。もう一方の鍵は FO モジュール内でデータ変換の際に使用される。FO 内で使用される 32 ビットの鍵は左右 16 ビットに分けて使用される。

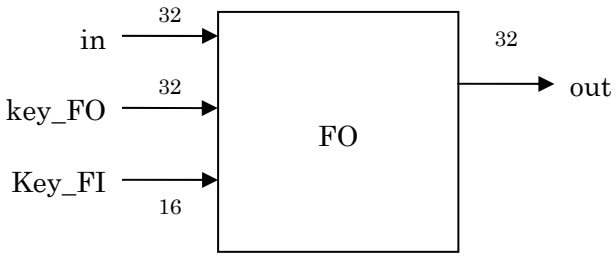

図 3.9　FO モジュールのインタフェース

表 3.4　FO モジュールの入出力信号

信号名	方向	幅（bit）	詳細
out	output	32	変換後のデータ
in	input	32	入力データ
Key_FO	input	32	データ変換に使用される鍵データ
key_FI	input	16	FI に渡される鍵データ

3.3.5　FI モジュール

FIモジュールの入出力インタフェースを図 3.10に、信号線の説明を表 3.5に示す。

FI モジュールでは、入力である被変換データ 16 ビットを左 9 ビットと右 7 ビットにデータ分割し、S7、S9 によるテーブル参照モジュールによるデータ変換、排他的論理和（XOR）や左右に分かれたデータの交差を行うことでデータ変換を行う。入力の鍵は左 9 ビット、右 7 ビットにデータ分割され、データ変換の際に使用される。

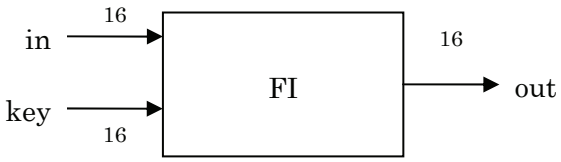

図 3.10　FI モジュールのインタフェース

表 3.5　FI モジュールの入出力信号

信号名	方向	幅（bit）	詳細
out	Output	16	FI による変換後のデータ
in	Input	16	入力データ
key	input	16	データ変換に使用される鍵データ

3.3.6　S7 モジュール/S9 モジュール

S7 モジュールの入出力インタフェースを図 3.11に、信号線の説明を表 3.6に示す。また、S9 モジュールの入出力インタフェースを図 3.12に、信号線の説明を表 3.7に示す。

S7 モジュール、S9 モジュールはテーブル参照モジュールである。本来この部分はメモリを参照するのが妥当であるが、今回は全てシミュレーション環境の関係上、HDL により記述をする。必要なテーブルの要素だけあらかじめレジストリに用意しておき、そこから参照することとする。

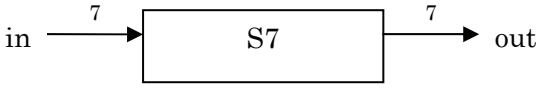

図 3.11　S7 モジュールのインタフェース

表 3.6　S7 モジュールの入出力信号

信号名	方向	幅（bit）	詳細
out	Output	7	S7 テーブル参照後データ
in	Input	7	入力データ

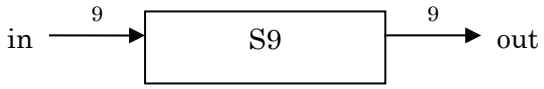

図 3.12　S9 モジュールのインタフェース

表 3.7　S9 モジュールの入出力信号

信号名	方向	幅（bit）	詳細
Out	output	9	S9 テーブル参照後データ
in	input	9	入力データ

3.3.7　KeyScheduling モジュール

KeyScheduling モジュールは、秘密鍵から拡大鍵を生成するモジュールである。入力（秘密鍵)、出力（拡大鍵）共に 128 ビットで、内部に複数の FI モジュールを構成し、データを 8 つに分割し、FI モジュールによりデータ変換を行った後結合し、128 ビットで出力する。

KeySchedulingモジュールの入出力インタフェースを図 3.13に、信号線の説明を表 3.8に示す。

図 3.13　KeyScheduling モジュールのインタフェース

表 3.8　KeyScheduling モジュールの入出力信号

信号名	方向	幅（bit）	詳細
ExtendKey	output	128	拡大鍵
ScretKey	input	128	秘密鍵

3.3.8　その他―Control、KeyAssignment

Control モジュールはループを制御するモジュールである。レジスタとカウンタのみによる単純なモジュールになっている。KeyAssignment モジュールは、LoopFunction、ExtraFunction などのデータランダマイズ部に当たるモジュールに鍵の割り当てを行うモ

ジュールである。Control からの制御信号・秘密鍵・拡大鍵を入力とし、Control からの信号を元に秘密鍵・拡大鍵よりデータランダマイズ部に渡す鍵を生成する。鍵の生成の仕方は 2.2 節で示した鍵の対応表より鍵の並べ換えを行うものである。

KeyAssignmentモジュールの入出力インタフェースを図 3.14に、信号線の説明を表 3.9に示す。

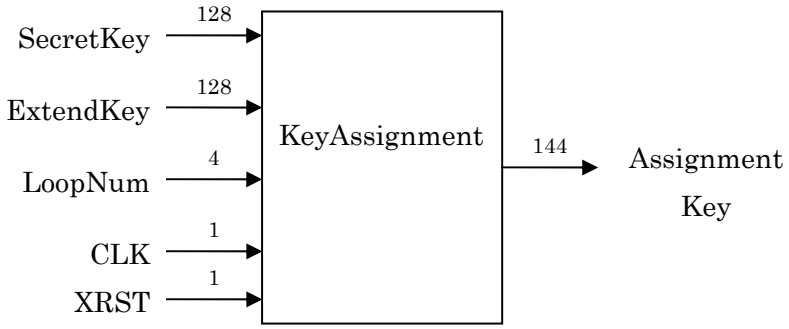

図 3.14　KeyAssignment モジュールのインタフェース

表 3.9　KeyAssignment モジュールの入出力信号

信号名	方向	幅（bit）	詳細
Assignment Key	output	144	データランダマイズ部に割り当てる鍵データ
ExtendKey	input	128	拡大鍵
ScretKey	input	128	秘密鍵
LoopNum	input	4	制御信号（現在のループ回数）
CLK	input	1	クロック
XRST	input	1	リセット信号

4. Misty1 暗号回路の並列化設計

4.1 パイプライン設計

3 章で設計したMisty1 暗号回路にパイプライン処理したものを設計した。これはデータのスループットをあげることによる性能向上を期待しての設計である。パイプライン設計の構成を図 4.1に示す。

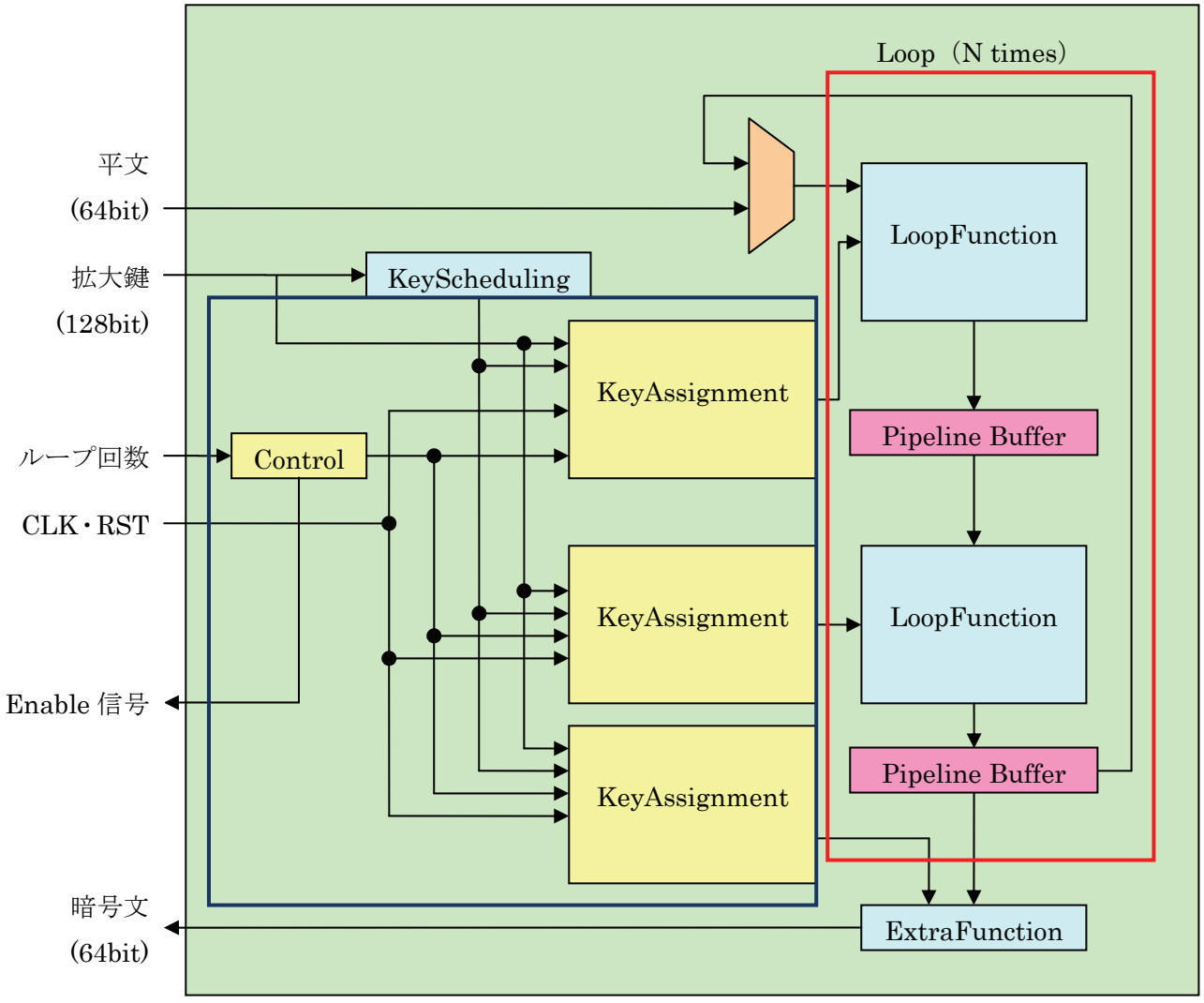

図 4.1 パイプライン処理

赤線で囲まれた部分がデータパスにおけるパイプライン処理、青線で囲まれた部分が制御におけるパイプライン処理となっている。

今回はループをする部分をパイプラインで記述した。LoopFunction を複数用意し、その間にレジスタを挟むことで、直前のクロックで得られたデータを保持し、次のクロックで

次の処理に移行することができるといった過程でのパイプライン処理がなされている。

パイプラインでの制御は、パイプライン処理を行う前における Control モジュール内の状態保持を行う、レジスタ・カウンタはそれぞれ 1 個であったが、パイプラインの段数だけレジスタ・カウンタを用意することで、複数の状態を保持することを可能とした。また、平文データの入力が可能であるか否かを判断するため、Control には Enable 信号を出力する機能も付け加えた。鍵の割付けにおいても KeyAssignment を複数個用意することで、各ループ数における鍵をデータランダマイズ部に渡せるように設計を行った。

なお、今回の設計ではパイプラインは 2 段構成にした。これは、Misty1 に使用できる段数は 4 の倍数に限り可変であるためである。LoopFunction の段数は 2 段構成になっている。そのためパイプラインを 2 段で構成すれば、Misty1 に使用できる段数の最低値をクリアすることになる。これ以上増やすのは 4 段で使用する場合においてのみだが冗長であると判断し、パイプラインを 2 段構成にした。

4.2　並列設計

4.1 節でパイプライン設計した回路に、さらに並列処理をしたものを設計した。これはパイプライン同様、データのスループットをあげることによる性能向上を期待しての設計である。並列処理設計したものの構成を図 4.2に示す。

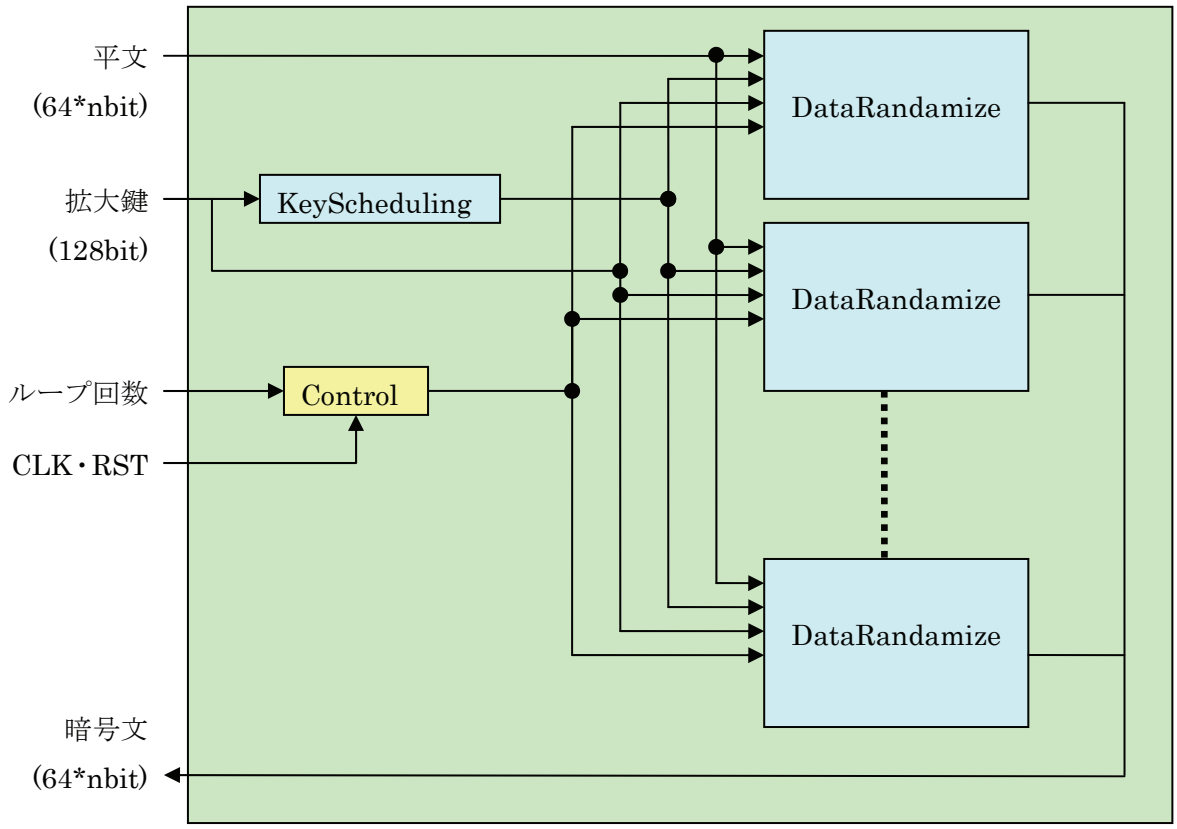

図 4.2　パイプライン＆並列化

図 4.2はパイプライン化したMisty1 暗号回路をさらに並列化した回路の構成である。ここで新しくDataRandamizeというブロックが出てきているが、これは、図を簡略化するためのもので、図 4.1におけるLoopFunction、ExtraFunction、KeyAssigment、をまとめたものである。図 4.2のようにモジュールを並べることで並列処理を行わせる。今回の設計では並列化は 2 段構成の並列化を行った。2 段以上で行うことに問題はないが、スループットの向上を測る目的での並列化なので、2 段でとどめることとした。

4.3 検証結果

4.3.1 ソフトウェアとハードウェアの性能比較検証

作成したソフトウェアで実行した場合の実行時間と設計したハードウェアで実行した場合の実行時間を各モジュール毎に比較した。尚、今回の検証ではMisty1 の段数を 8 段に固定して計測している。ソフトウェアの実行環境は、Core2Duo2.66GHz、DRAM3.0Gbyte、WindowsXP、Cygwin上でプログラムを実行し、動作時間を計測し、クロック数の算出を行った。ソフトウェアの実行時間は、複数の値で 10 回ずつ実行した場合の平均時間を出している。表 4.1にソフトウェアとハードウェアにおける各モジュール単体でのクロック数の比較を示す。表 4.1の全体実行以外の各モジュールのクロック数はデータパス部なので実際にはクロックは使用されていないが、ソフトウェアと対比するため 1 とする。

表 4.1 ソフトウェアとハードウェアの性能比較

モジュール名	All	FL	FO	FI	KeyScheduling
クロック数（ソフトウェア）	2973	15	191	43	520
クロック数（ハードウェア）	4	1	1	1	1

表 4.1より、全体的な比較ではハードウェアで実行した場合はソフトウェアで実行した場合に比べて約 743 倍の速度向上が得られている。

4.3.2 各ハードウェアにおける性能評価と考察

Misty1 暗号回路、パイプライン設計後の回路、パイプラインの並列設計後の回路の性能の評価・比較を行った。表 4.2に各回路の回路規模と性能を、回路をパイプライン化・並列化したときのスライス数と最大スループットの伸び率のグラフを図 4.3示す。スライス数および最高動作周波数は、論理合成ツールISE8.2iによる配置配線レポートから算出している。FPGAはVirtex4 を対象としている。

表 4.2におけるNormalとはパイプライン設計も並列設計もしていない回路である。また、今回の検証に使った回路はパイプライン、並列化、共に 2 段構成である。

表 4.2　Misty1 暗号回路と並列設計された回路の評価

	Normal	パイプライン	パイプライン＆並列化
スライス数	4,943	7,124	11,273
Max Delay(ns)	2,922	2,859	3,018
最大動作周波数(MHz)	342	349	331
最大スループット(Mbps)	5,472	11,168	21,184

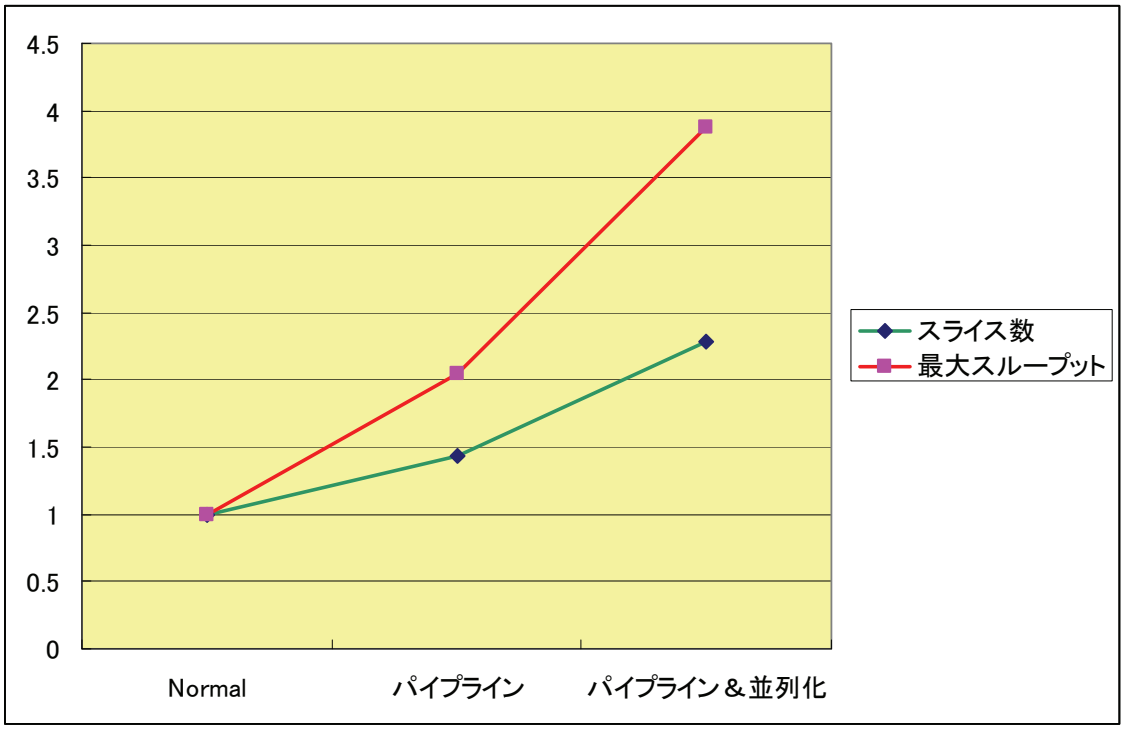

図 4.3　スライス数・スループットの伸び率

表 4.2、図 4.3より、パイプライン設計、並列設計をしたことによる性能の向上がわかる。スライス数の増加率よりも最大スループットの増加率の方が大きい。そして、最大動作周波数の変化は少ない。これは並列化設計の目的通り、スループットの増加による性能向上が図られたといえる。パイプラインから並列化をしたときのスライス数と最大スループットの増加率はそれぞれ、1.58 倍、1.90 倍と、並列設計による性能の恩恵はパイプライン設計による性能の恩恵よりも少ないが、今回のようなパイプライン設計では、段数が増えるほど冗長な回路も増える可能性が高くなるので、並列化による性能の向上は充分であると考えられる。Misty1 で使用する段数に合わせたパイプラインの段数設計を行い、並列設計をした回路が最適なものになると考えられる。

次に表 4.3にモジュールの単体での回路規模を示す。

表 4.3　各モジュール単体での回路規模

モジュール名	ALL	FL	FO	FI	S 9	S7	KeyScheduling
スライス数	4，943	16	955	282	120	29	2,489

表 4.3から、テーブル参照モジュールが使用されているモジュールの回路規模が大きいことがわかる。これは本来メモリを使用するべきところを回路記述しているからであると考えられる。テーブル参照モジュールを改良すれば、より規模の小さい回路が実現できると考えられる。

5. ハード/ソフト最適分割の検討

5.1 Misty1 暗号における検討

本節では、Misty1 暗号におけるハード/ソフト最適分割の検討を行っていく。2.3 節で述べた手法に基づきモジュール毎での分割パターンを決定し検討する。しかし、Misty1 は入れ子構造の多いモジュール構成になっているため、モジュールをハード/ソフト分割するには、非常に難しく設計時間が大きくなる可能性が高い。そこで、Misty1 の段数（ループ回数）が可変であることを利用して、ハードウェアとソフトウェアで段数による分割も検討する。

5.1.1 モジュール分割

ソフトウェアとハードウェアのそれぞれで設計した、Misty1 暗号に対して、ハード/ソフト協調システムとして実装する分割パターンを選択する。表 5.1に分割パターンを決定する上で考慮すべき、各モジュールが回路全体で占める回路規模とSW実行時の負荷の割合を示す。なお、SW実行時の負荷割合はMisty1 における段数が推奨値である 8 のときのものになっている。

表 5.1　各モジュールが全体に占める回路規模と SW 実行時の負荷割合

	スライス数	SW 実行時間の負荷割合
ALL	4,943	—
FL	64	5
FO	1,910	51
FI	3,948	46
KeyScheduling	2,489	17
その他（制御等）	480	28

表 5.1および、3.1 節で示したソフトウェアの負荷探索における考察結果と、4.3 節で示したハードウェアモジュールの評価から、表 5.2に示すような 7 種類の分割のパターンを決定した。

表 5.2　Misty1 暗号システムのハード/ソフト分割パターン

分割パターン	ハードウェア処理	ソフトウェア処理
A		FI、FO、KeyScheduling、FL、制御
B	FI	FO、KeyScheduling、FL、制御
C	FI、FO	KeyScheduling、FL、制御
D	FI、KeyScheduling	FO、FL、制御
E	FI、FO、KeyScheduling	FL、制御
F	FI、FO、FL	KeyScheduling、制御
G	FI、FO、KeyScheduling、FL	制御

FI は負荷の高い FO、KeyScheduling の下位モジュールであるため、必然的にハードウェアで実現する形になる。FI の負荷は全体の約 1/2 を占めており、ハードウェア化の効果が高いと見込まれる。しかし、至るところで下位モジュールとして使用されているため、システムの設計は複雑になる可能性が高い。KeyScheduling は、段数 8 段では負荷は高いが、段数の変化による影響がないため、KeyScheduling のみをソフトウェア部とするパターンも分割パターンに組み込んだ。

5.1.2　段数による分割

モジュール分割を行わず、ソフトウェア設計したもの、ハードウェア設計したもの（制御、KeyShceluling をソフトウェアで分割したもの。表 5.2でFにあたるもの）を用意し、ソフト・ハードで一定回数ずつ処理を行わせる形で分割した。例えば、100 段（100 ループ）を行う暗号化システムに対し、32 回分のループをソフトウェア実行し、68 回分のループをハードウェア実行するなどの形で分割した。多段の処理を行わせ実験することで、段数における、ソフト：ハードでの割合、またはそれぞれに適した回数を導き出し最適分割設計を行う。分割の方法は、設計したハードウェアの仕様の都合上、4 の倍数ずつに分割されることになる。図 5.1に分割方法を示す。

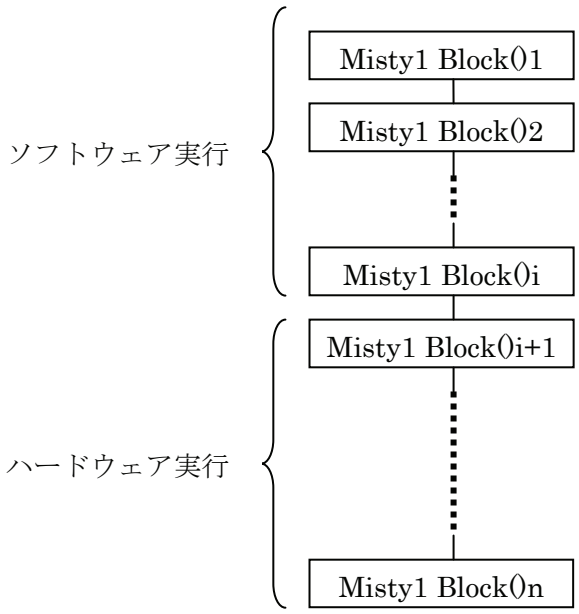

図 5.1　段数によるハード/ソフト分割

処理速度ではハードウェアの方が優れていること、回路規模ではソフトウェアの方が規模は小さく優れていることは明快なので、メモリ使用量、設計工数、電力などの項目に重点を置き評価する。

5.2　考察

5.2.1　各分割方法についての考察

モジュール分割による最適化では 7 つの分割方法を提唱したが、Misty1 は入れ子構造が多く、しかも、複数のモジュールを何度も利用することから、モジュール分割は非常に難しい作業になる。そのため提唱したうちの A~D、F は設計に非常に時間を要する可能性が高い。KeyScheduling 以外のモジュールは段数が増加するにつれソフトウェア実行による負荷が高くなること、またソフトウェア実行のほうが、電力消費が高いであろうことも考慮して、E、F、G による分割パターンが一番バランス良く妥当なのではないかと考えられる。

重要視する箇所を変えれば、優先される分割パターンも変わる。重要視する項目を変えたとき優先される分割パターンを表 5.3に示す。これは実測データを基にしたものでなく、予想されうる範囲でのものとなる。

表 5.3　重みの違いによる分割パターンの優先順位

優先する重み	クロックサイクル	回路規模	電力消費	設計工数	メモリ使用量
分割パターン	G>E>F>D>C	A>B>C>D>E	G>E>F>D>C	A、G >E>F>D	G>E>F>D>C

表 5.3における、クロックサイクル数、回路規模での優先順位は、設計したソフトウェアとハードウェアの検証結果よりこのような順位をつけた。また、電力消費もハードウェア部が多い方が消費電力も少ないと容易に予想されるのでこのような順位をつけることとした。設計工数においては、入れ子構造を持ち、上位モジュールと下位モジュールで、ソフトウェアとハードウェアに分割されている箇所が多いものを、設計困難と見なし、優先順位を決定した。メモリ使用量については、ハードウェア部が多い方が、メモリ使用量が少ないと予想し優先順位を決定した。

表 5.3より、全体的にE、F、Gの優先順位が高くなっていることから、この 3 パターンの分割手法が妥当ではないかと考えられる。

次に、段数による分割について考察する。Misty1 のハードウェア処理速度はソフトウェアによる処理速度の約 743 倍である。制御、KeyScheduling をソフトウェアで実行するとしてもかなりの性能差がある。他の重みの比重を大きくしたとしても、ハードウェアのみでループをまわすほうが、いい結果が得られるであろうと予測される。設計工数などを考慮してもハードウェアのみの設計が最適ではないかと考えられる。

それぞれの分割方法による考察からMisty1は制御以外の部分をハードウェア設計することが最適なのではないかと考えられる。

5.2.2 評価式による考察

共同研究者である梅原直人氏が考案した、評価式を使用し、考察を深めていく。

$$priority_{pattern} = \sum_{Item=C,G,M....} \left(weight_{Item} \frac{Value_{worst} - Value_{this}}{Value_{worst} - Value_{best}} \right)_{Item} \quad \cdots(1)$$

priority ：あるパターンの優先順位度

Item：項目、クロックサイクル数（C)、回路規模（G)、使用メモリ量（M）等

weight ：項目(Item)ごとの重み、ただし全項目の weight の総和は常に 1

Value：実測値、添え字の”worst”は該当項目するパターンの最大値、”best”は該当するパターン項目の最小値、”this”は該当するパターンの実測値

この式は、全パターンにおいて、項目ごとに最悪の値（最大値）と最良の値（最小値）を抽出して、最大値を 1、最小値を 0 として、評価対象となるパターンの実測値の割合を出し、重み付けをして、それらを全て足し合わせることで、そのパターンの優先度を数値として算出する。

この評価式を実行するプログラムを作成した。その実行結果を図 5.2に示す。

```
$ ./a.exe 0.25 0.25 0.25 0.25 misty1.txt
Clock           Gates           Power           Man-Hour
weight
0.250000        0.250000        0.250000        0.250000
value
2973.000000     0.000000        0.000000        1.000000
1597.000000     3948.000000     7896.000000     5.000000
1101.000000     4106.000000     6159.000000     3.000000
1421.000000     2489.000000     4480.200000     4.000000
925.000000      4399.000000     5206.800000     2.000000
951.000000      4230.000000     5499.000000     2.000000
775.000000      4463.000000     4463.000000     1.000000
max_value
2973.000000     4463.000000     7896.000000     5.000000
min_value
775.000000      0.000000        0.000000        1.000000

priority[A] = 0.750000
priority[B] = 0.185354
priority[C] = 0.412915
priority[D] = 0.457750
priority[E] = 0.509168
priority[F] = 0.506426
priority[G] = 0.608694
```

図 5.2 評価式プログラムの実行結果

このプログラムはコマンドライン引数に各項目の重みをクロックサイクル数、回路規模、消費電力、設計工数の順に入力し、最後に実測データの入ったファイルを渡す。結果には全分割パターンのプライオリティが数値として小数点以下 6 桁まで出力される。

今回は、実測データを使用せず、予測データを使用して評価式を実行した。また、今回は、メモリ使用量を評価項目から外し、検討することとした。メモリ使用量の予測を立てることができず、実測しなければ、評価項目に含めることができなかったためである。クロックサイクル数、回路規模は、4.3 節の検証結果を基に、数値を予測した。電力消費はクロックサイクル数、回路規模から予測している。しかし、ハードウェアのみの電力消費の予測のみで、ソフトウェアで使用する電力消費量を予測に含めていない。設計工数は 5 段

階評価で、入れ子構造を持ち、上位モジュールと下位モジュールで、ソフトウェアとハードウェアに分割されている箇所が多いものから、数値の高い評価を与えている。

この予測データを使用して、各項目の重みを全て同等にして、実行を行った結果が図 5.2 の結果である。この結果から優先順位を得ると、A>G>E>F>D>C>Bという優先順位が得られる。ソフトウェアでの電力消費を考慮しなかったため、ソフトウェアのみで設計されるAが最優先となったが、電力消費を考慮すればAの優先順位は大きく下がるものと思われる。やはり、E、F、Gのパターンが優先順位が大きく、Misty1 暗号システムにおいてはハードウェアでの設計が望ましいものと考えられる。

6. おわりに

本論文では、Misty1 のハードウェア化による高速化の実現方法と、パイプライン処理、並列処理によるスループットの向上の実現方法、また、Misty1 におけるハード/ソフト最適分割の検討と考察を述べた。Misty1 のハードウェア化による高速化については、C 言語で実行した場合と比較して、約 743 倍の速度向上が得られた。また、ハードウェアに、パイプライン処理、並列処理を施すことで、スループットの向上を図ることができた。ハード/ソフト最適分割の検討では、モジュール分割において 7 つの分割パターンを選択し、それぞれについて検討、考察をした。また、段数（ループ回数）毎にハード/ソフトを分割する方法についても検討、考察をした。

今後の課題として、考案した分割パターンの実装と検証、評価があげられる。また、現在の評価方法は、クロックサイクル数、回路規模、消費電力、設計工数、メモリ使用量の 5 つだが、この他に、再利用性、コスト、保守性なども評価対象に含め評価する必要もある。

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、本研究の共同研究者である梅原直人氏、及び色々な面で貴重な助言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] David A. Patterson/John L. Hennessy, 成田光彰訳:コンピュータの構成と設計(上), 日経ＢＰ社, 1999.

[2] David A. Patterson/John L. Hennessy, 成田光彰訳:コンピュータの構成と設計(下), 日経ＢＰ社, 1999.

[3] 小林優:ハードウェア記述言語の速習＆実践 入門 Verilog-HDL 記述, CQ 出版社, 2002.

[4] 並木秀明, 前田智美, 宮尾正大：実用入門ディジタル回路と Verilog-HDL, 技術評論社, 2004.

[5] 三菱電機株式会社：暗号技術仕様書 Misty1, 2001.

[6] 松井充：ブロック暗号アルゴリズム MISTY, 信学技報, ISEC96-11, 1996.

[7] 深山正幸, 北側章夫, 秋田純一, 鈴木正國 : HDL による VLSI 設計, 共立出版, 2000.

[8] 梅原直人：設計仕様解析によるハード／ソフト最適分割システムの実現と評価, 立命館大学理工学研究科修士論文, 2007.

[9] 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, 立命館大学理工学部情報学科卒業論文, 2005.

[10] 的場督永 : ハード/ソフト最適分割を考慮したJPEGエンコーダの協調設計, 立命館大学理工学部情報学科卒業論文, 2005.

[11] 古川達也：FPGA上でのソフト・マクロCPUによるハードウェア/ソフトウェア分割手法の研究, 立命館大学理工学研究科修士論文, 2005.