修士論文

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

氏　　名	:	梅原 直人
学 籍 番 号	:	6124050039-5
指 導 教 員	:	山崎 勝弘 教授
提 出 日	:	2007 年 2 月 16 日

立命館大学大学院　理工学研究科　情報システム学専攻

内容梗概

本論文ではハードウェア／ソフトウェア協調設計におけるシステムの分割時に発生する問題を解決する最適分割手法を主題とした。我々は実際に回路を設計する前に、ハードウェアとソフトウェアの設計空間をユーザの要求を反映しつつ分割する、最適分割システムを考案した。本論文ではその中でも核となるC言語プログラムの解析システムツールを設計し、MISTY1暗号とAES暗号という実用的なアプリケーションに対して適用し、検証を行った。

実装環境はMemec社（現Avnet Japan社）のMicroBlaze評価用FPGAボード「DS-KIT-4 VLX60MB-EURO」を使用し、ソフトウェア実行や通信制御などにソフトコアMicroBlazeを利用した。この評価用ボードにはXilinx社のFPGAチップ「Virtex-4」が搭載されており、59,904ロジックセル、500MHzシステムクロッキングと高性能なチップとなっている。

最適分割システムを適用して実験する対象アプリケーションとして秘密鍵方式ブロック暗号”MISTY1”を採用した。MISTY1暗号は繰り返し同じ処理を行うことで暗号化を実現するものであり、これを利用して意図的にループを展開して粒度を操作することで、任意の大きさのハードウェア／ソフトウェア・モジュールが得られる。その内容を解析することで、実測値と比較して最適分割システムの評価を行った。また、AES暗号アプリケーションに対しても同様の実験を行い、4種類の全く異なる内容のモジュールを用いることで実験の内容をより充実させた。

本研究からアプリケーションのソフトウェアプロトタイプの段階から協調設計システムにおける設計空間領域の探索を行う手法を構築し、検討した。また、協調設計の実機検証において簡便にユーザのカスタムハードウェアをFPGAに搭載してCPUと協調して動作させることが出来る環境を整えた。そして解析と実機での実験結果から相関関係を見つけ出し、提案した最適分割システムの有効性について述べた。

目次

図目次

表目次

はじめに

現在、LSIは微細化が進んでナノスケールオーダでの製造が行われており、小型化・高性能化が進み続けている。それによって機器の小型化などが促され、その結果生まれた代表的な製品として携帯電話などの発展やユビキタスな情報家電などが見られる。LSIは生活の、ひいては社会の基盤となってありとあらゆる場所で用いられている。その発展の著しいLSIにおいてSoC、またはシステムLSIと呼ばれるあるシステムをワンチップに集約する分野は、ドッグイヤー（通常の 7 倍速い）と呼ばれるほど発展の速い箇所であり、それに伴って非常に厳しい制約条件もついてまわる。製品の開発サイクルが短縮しているにも係わらず小型化、高機能化、高速化、低消費電力化などと、その要求は多岐に渡る。SoCの設計では厳しい要求仕様を実現するために様々な手法を用いており、その中にハードウェア／ソフトウェア協調設計がある[15][16]。ハードウェア／ソフトウェア協調システムはハードウェアの処理の高速性や周波数の関係による消費電力、ソフトウェアの再利用性と柔軟性といった両者の長所を取ってシステムの高性能化を実現するものであり、SoCなどの制約の厳しいシステムを開発するためには必須となっている。ソフトウェアの柔軟性と設計の容易さ、ハードウェアの高速性といった利点を取り合わせることが理想となるが、反面、システムのハードウェアとソフトウェアのバランスを誤ると仕様を満たせなくなり、重大な問題である設計の手戻りが発生する可能性が高まる。また、そのバランスを取るためにはハードウェアとソフトウェアについて数深い理解が必要であり、さらに実際の設計経験がなければ実現が難しいものとなる。

我々は以前の研究において最適な分割を行うにはどの様にしたら良いかを網羅的に探索して、計測した数値を独自の数式などを作成して比較した[1][3]。その結果を実際の設計に反映させるためのシステムが本研究で対象とするハードウェア／ソフトウェア最適分割システムの構築である。

本研究ではハードウェアとソフトウェアの最適な分割を実現するために、対象となる問題のC/C++言語のソフトウェアプロトタイプが完成した段階でプロトタイプを解析して設計の早期段階からハードウェアとソフトウェアの最適な分割を見つけ出すことを目的とする。

最適分割システムは C ソースコードをターゲットとして解析を行い、モジュール間の相対的な性能を導出する。C ソースコードをターゲットとする理由は、ソフトウェア・ハードウェア双方のプログラマの大多数が扱えるプログラミング言語であり、現在でも組み込み機器のプロトタイププログラムに利用されているからである。C ソースコードからモジュール毎の CDFG など最適分割ツールのパラメータを作成して、そこにターゲットのシステム固有の特徴量を考慮することで各モジュールの相対性能を導き出す。相対的な性能を基本としているのは、絶対的な性能は評価としてはユーザ要求に応える確実性があり、分割領域探索の結果も一意に決まる利点があるが、現在の大規模モジュール化しているシステムにおいてそのような数値計算を行うと莫大な時間が掛かり、電力解析やタイミング解析といった単一項目では利用されているが、全てを見渡して計算すると実用できるとは考えられないからである。

相対性能が出てきたならば、そこに速度重視や回路規模重視、もしくは複数の項目を重視するといったユーザの重視項目の重みをつけることで、要求に沿った最適なシステム構成案を提示する。

ここで言う最適なシステム構成とはソースコードの関数をモジュールに対応させて、全てのモジュールに対してハードウェアが良いかソフトウェアが良いかという回答であり、関数（モジュール）をツールが自動で分割したり統合したり、あるいは改変したりすることはない。

ツールを作成後、それらの手法をISOの標準暗号にも採択されている三菱電機株式会社が編み出した国産暗号のMISTY1 アプリケーションに適用して、本システムの有効性を評価する[6]。本論文ではFPGAベンダであるXilinx社のEDAツール「EDK」を使用したソフトコアCPU「MicroBlaze」を中心とした検証環境を構築し、MISTY1 暗号の論理合成結果や実装してのクロックサイクル数などを計測した。EDKなどを利用した理由はI/Oインタフェースなどの実験の本質とは関わらない部分を提供してくれる環境を簡単に入手でき、FPGAの普及に伴ってソフトコアも浸透し、完成度もかなり高まっていると考えられたからである。またEDKはツール自体が比較的に安価で手に入りやすく、対象となるボードも安価なものから高性能なものまで幅広く扱えるからである。

上記の検証環境を用いて最適分割システムの実験を行い、MISTY1 暗号アプリケーションでの評価を行った。本論文での評価対象項目は処理速度・回路規模・使用メモリ量の 3 点がメインとなる。消費電力などは構想段階であり、今後の課題となる。システムの検証にはクロックサイクル数などの実測値とツールから合成される予測値を使って利用している。これにより現実と最適分割システムが出す答えがどの程度乖離しているかを数値的に、または経験則からも検証する。

1. ハード／ソフト最適分割システム

1.1　システム構成

ハードウェア／ソフトウェア最適分割システムとはCソースコードを解析してシステムを分割するものであり、その詳細な流れを図 1に示す。

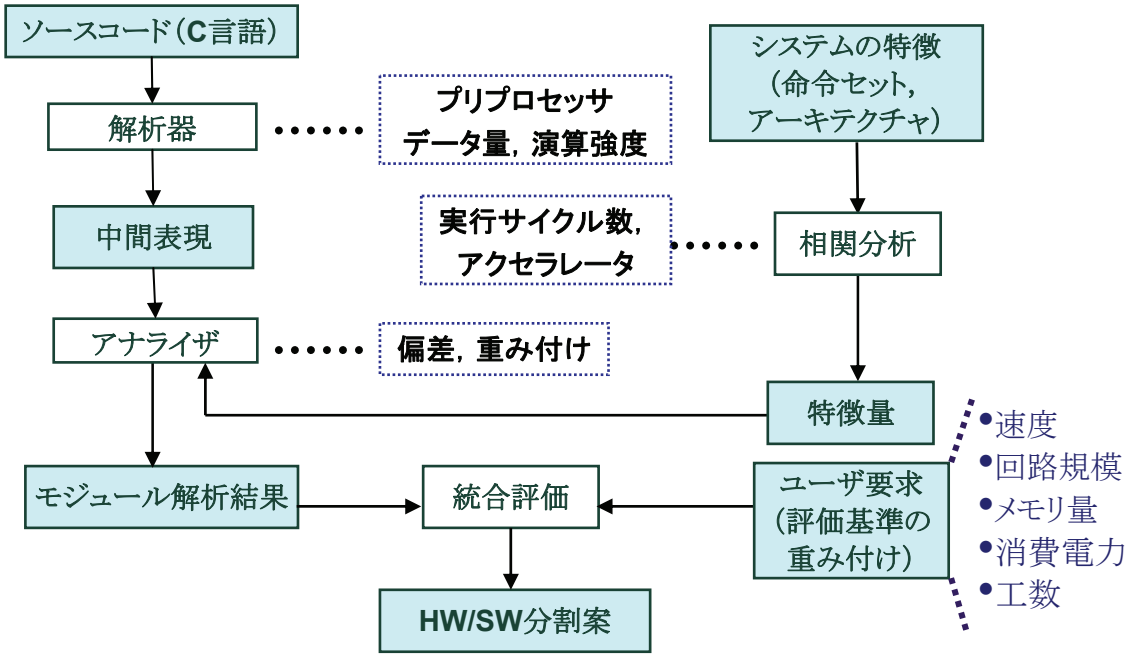


図 1　最適分割システムの全体フロー

本システムの入力はソースコード・システムの特徴・ユーザの要求であり、これらを総称して「設計仕様」と本研究では呼んでいる。これら設計仕様を解析することで、最終的にはハードウェア／ソフトウェアの分割案を提案する。またこれらを全て自動で行うツールもシステムとして含まれている。
図 1において、システム外部仕様の入力は、C言語によるソースコード・設計対象システムの特徴（量）・ユーザ要求となる。また、出力はハードウェア／ソフトウェア分割案のみとなる。内部の処理を順に追っていくと、まず図 1の左側で、ソースコードを解析してモジュール切り出しや変数の探査を行い、モジュール毎の内容を中間表現に変換する流れがある。次に図 1右側においてシステムの特徴量を導く。システム特徴量は、CPUアーキテクチャなどから求められる。モジュール解析結果に対して、システムの特徴量を掛け合わせることで正式な各モジュールの項目別性能を算出する。最後に、ユーザの速度優先といった要求からどの性能項目を重視するかなどを決めてやり、その統合した結果から最適分割案を導く。

1.2　ハード／ソフト最適分割システムにおける着目点

本システムで分割するために着目する協調システムでの性能項目などについて述べる。

分割するために重視する要素として以下の 5 つに着目する。

(1) 処理速度

処理速度は最も単純にして重要な要素であり、ソフトウェアプロトタイプと比較して如何に速くなるか、スループットが向上するかに着目する。この比重を大きくすると速く処理できるほど優秀である。注意すべき点は CPU やシステムの動作周波数は関係しない点である。

数値の単位はクロックサイクル数とほぼ同義だが、基準となるものは排他的論理和"EXOR"1 回を実行するのに必要なサイクルとなる。

(2) 回路規模

回路規模は最終的な製造コストに掛かってくる問題であり、これも分かりやすく且つ重要な目的である。ここにはハードウェア側のメモリに関しても含まれることになる。

単位は EXOR ゲート 1 個分を基準として相対的に表される。

(3) 使用メモリ量

使用メモリ量はソフトウェアで使っているメモリをどれだけ減らせるかを見る。巨大な配列などをハードウェア側に追い出すことでメモリ量削減が実現できるが、その場合には回路規模の方が大きくなる点がネックとなる。

単位は Byte である。

(4) 消費電力

消費電力は近年の SoC における最も大きなトレンドである。LSI の微細化による性能向上とは反比例する形でリーク電流などの問題が浮き彫りになった。また単純に携帯電話など限られた大きさでも長時間駆動させることを目標とすると消費電力の解析を無視するわけにはいかなくなる。本研究では消費電力についても目的として組み込むことで時流に対応する。ただし電力解析は研究のテーマではないので簡単化する。以下に電力の近似式をあげる。

$$P \propto \sum \alpha_i \times C_i \times F \times V^2 \qquad \cdots式(1)$$

式(1)の P が総合消費電力、α はゲーティッドクロック機構を採用した場合の機能モジュール毎の生起確率、C はキャパシタンス、F は周波数、V は電圧である。周波数と電圧は基本的に(特に FPGA 上)システム全体では変更することはないので無視する。つまり機能モジュールの生起確率とキャパシタンスが重要となる。生起確率は静的に計算することで求められ、キャパシタンスはゲート数などから比率を取ることができる。このように簡単化した式を利用して消費電力の計算を行う。

単位は電力などに関するものではなく、厳密には定義できないがクロック数に係数を掛けること

で表す。

(5) 工数

最後に工数であるが、これが一番本研究における特異な部分となる。工数はあるシステムを設計・検証する上で必要となる人員の人数とそれに掛かる期間であり単位は「人月」である。重要なファクタは人間の（設計）能力とモジュールの設計難易度である。大きな問題となるのが、人間という極めてファジーな要素が関わってくることである。個々人の能力を調査する方法は IQ テストなどが有名ではあるが、それにしても絶対ではない。ましてや LSI 設計能力といったものは測ることなど不可能に等しい。そこで最適分割システムにおいては個人の能力に関しては棚上げし、モジュールの設計難易度にのみ着目する。これには主に制御に必要となる状態遷移図の複雑さやデータストリーム系列数が要点となる。

単位は通常、工数を表すために使う人月ではなく、独自の複雑性を表すものを使用する。

ユーザ要求もこれらをどう扱うかという情報を保持することになる。これらの項目を採用した理由は、処理速度や回路規模、使用メモリ量、消費電力などはシステムの性能を決める上で明らかに直接的に関わる要素であり、これらを無視することはできないので取り上げている。工数については以前に我々が行った協調設計の研究から、回路の複雑さとその労力に対する性能向上が得られないパターンがあったことから、開発におけるメインコストの一つである工数と結びつけて算出することを考案した。

1.3 ハード／ソフト分割手法

考案した最適分割システムの処理内容などについて段階的に説明する。

まず C ソースコードからは、一旦解析しやすい独自の中間表現に変換してそこから各モジュールの相対的性能値を解析する。またその解析の際には、何 bit 幅の処理系を持っているか・乗算器やシフタを備えているかといったシステムの特徴量を抽出して解析に反映させることで環境にあった結果を出すことを可能にしている。

性能比較などに用いる中間出力の数値などは何らかの絶対的な数値ではなく、相対的な数値となる。例えば、演算強度の算出には排他的論理和を 1 回実行する速度を基準としており、これの何倍になるかをモジュールの演算強度として導く。このような性能評価をいくつか行い、性能評価結果を単独で、あるいは組み合わせることでモジュール毎の総合的な性能を出す。

性能の一覧が出されたならば、次にユーザの目的にあわせて性能値に係数を掛けて統合することで、最終的な解析結果を求める。解析結果は各モジュールに対応する関数をハードウェアとソフトウェアのどちらにするべきかを示すものであり、関数内部を解析してモジュールの粒度を自動的に調整までするものではない。

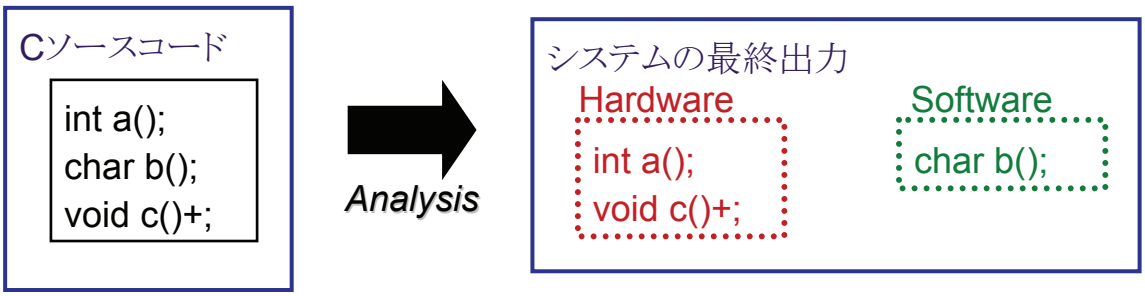


図 2　最適分割の入力と結果

1.4　設計仕様解析

設計仕様解析はソースコードとシステム特徴量、ユーザ要求から構成されるが、この中で核となるのはソースコードとシステム特徴量の 2 点である。これら 2 点を解析するフローを図 3に示す。

まず入力としての C ソースコードの条件やサポート範囲について示す。これらを満たさなければ仕様対象外となる。

- コンパイラ GCC をエラーなく正常に通ること
- 各関数がハードウェアに対応するように設計されている
- モジュールの入力と出力が関数の引数と戻り値に対応
- システムの状態遷移をコントロールするモジュールは関数 1 つにまとめること
- for 文は評価式内部を 3 カ所とも埋めること
- while ループ中の break はサポートする

次にシステムのサポート対象外となる主な項目について示す。

- 文字列操作
- ポインタ変数
- malloc 関数, goto 関数

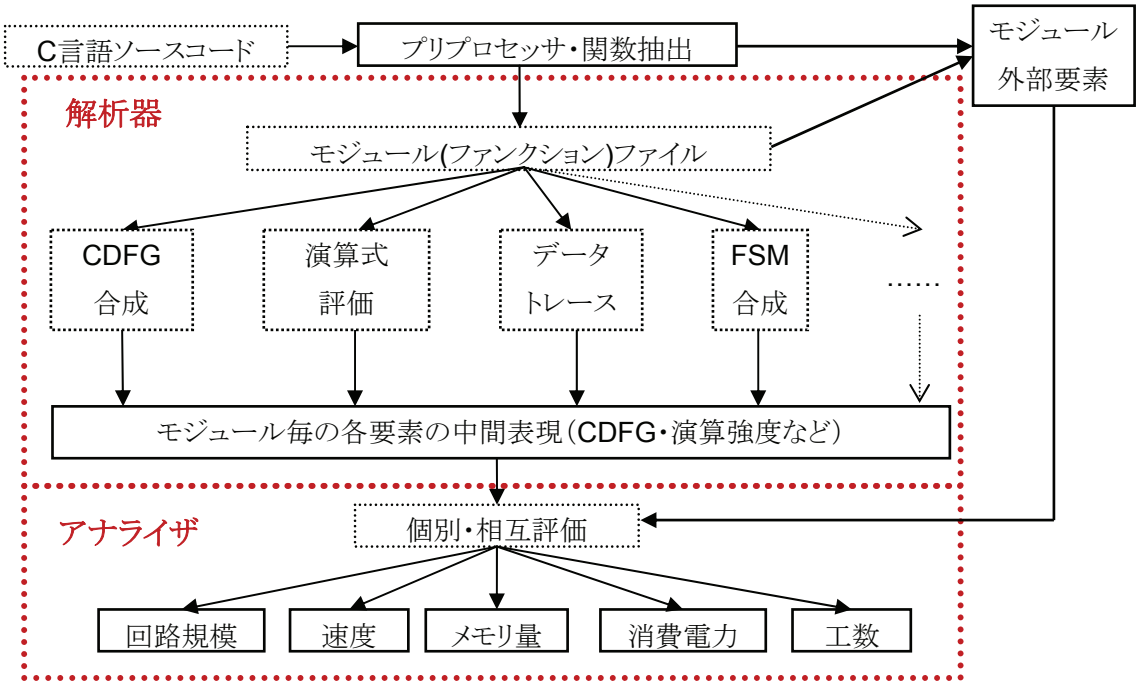


図 3　設計仕様の解析手法

少々厳しい制約かもしれないがハードウェアのリファレンスとして設計されているとするならば、これらの制約も無理難題ではないと考えられる。これらの条件を満たしたソースコードに対して以下のような手順で処理を行っていく。

1. 前処理としてグローバル変数やプリプロセッサ、関数部分の特定などを行って抽出する
2. モジュールに対して CDFG の合成や演算式の評価を行い、中間表現にまとめる
3. それらに環境などの外部要素を組み込んで調整し、相対性能を導く

上記の処理を行うために、解析器とアナライザの 2 つのツールを作成した。

解析器はモジュール毎に CDFG などを生成するものであり、性能評価の元となる部分を担当する。現段階では CDFG 生成・演算式評価・データストリーム検出・有限状態機械の作成を機能として見込んでいるが、モジュールファイルは独立しているので、さらに評価のための要素(例えばタイミング解析)を追加することも可能となっている。

2. 設計仕様解析システム

2.1 概要

本研究ではソースコードや実装環境、ユーザの目的などをひとまとめにして「設計仕様」と呼んでいる。これらを分析することで性能値や最終評価を導き出す。図 4に最適分割システムの解析システムのUMLクラス図を利用した表現を示す。ただし、図 4において"CodeParser"などに向かっている黒塗り四角の矢印はUMLにおいてはコンポジションを意味するが、本論文ではアグリゲーションを意味する。†

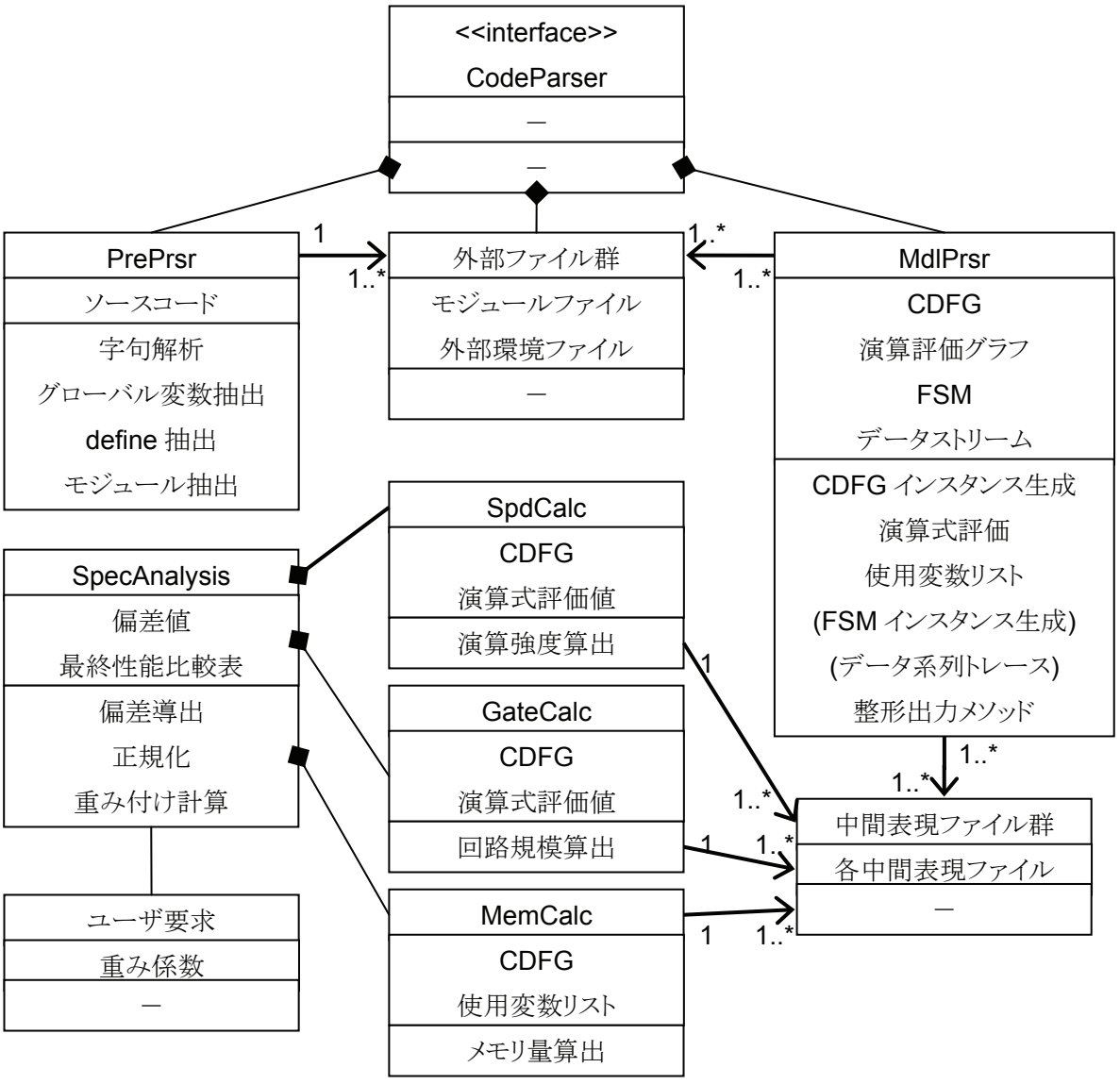


図 4　解析システムの UML クラス表現

---

†本来 UML のアグリゲーションは白抜き四角矢印で表されるが、これは Microsoft Office Word における図の表現で白抜きが表現しづらいので簡単化のためにこうなっているので注意されたい。

図 4の各クラスの説明を以下に箇条書きする。クラス名がアルファベットで記載されているものがRubyオブジェクトに相当し、実際のツールの核と呼べる。日本語で書かれているものは実際にデータとして存在しうるファイル群である。ファイル群はディレクトリに格納される予定であり、実際のオブジェクトとは少々異なるが無いとUMLクラス図が表現できないので作った。

- CodeParser：解析器の最上位オブジェクトインタフェース．PrePrsr,外部ファイル,MdlPrsrのオブジェクトを包含する
- PrePrsr：前処理部．関数箇所の切り出しや、定義文などの抽出・独自表現への変換を行う
- 外部ファイル群：モジュールファイルと外部環境ファイルからなるファイルグループ．モジュールファイルは単なる関数部分を切り出したファイルであり，外部環境ファイルは独自の表現方法で記述されている
- MdlPrsr：モジュール解析部．CDFG などを合成する部分であり，実際にはさらにクラス内部でモジュール化される．収まりきらないので省略している
- 中間表現ファイル群：CDFG などを中間表現に変換したファイルグループ
- SpdCalc：処理速度クラス．中間表現から適当なファイルを参照して導出する
- GateCalc：回路規模クラス．中間表現から適当なファイルを参照して導出する
- MemCalc：使用メモリ量クラス．中間表現から適当なファイルを参照して導出する
- SpecAnalysis：総合評価クラス．偏差を求め正規化してから，重み付け計算を行う
- ユーザ要求：速度・回路規模・メモリ量・消費電力・工数のそれぞれの評価比重を示すファイル．それぞれ 0.0～1.0 を範囲とする．現在は消費電力と工数はシステムとして考慮していない

これらのクラスをまとめることでモジュール解析システムが構成される。CodeParser クラスとSpecAnalysis クラスがユーザから直接的に見ること、及び操作することのできるクラスとなる。ファイル群も基本的にディレクトリ管理されているだけなので見ることは可能であるが度外視する。

これらツールのモジュールはオブジェクト指向スクリプト言語であるRubyで記述されている[9]。Rubyは字句解析器を作成する上で便利かつ強力であり、記述量が少なく手済む。またC++を意識して洗練されたオブジェクト指向の文法となっているのでメンテナンスの上でも大きな利点がある。以上のような理由から本研究のツール設計のプログラミング言語として採用している。

2.2 性能評価項目

最終的なユーザの目標として、処理速度・回路規模・メモリ量・消費電力・設計工数を対象としている。これらを見積もるためにモジュール毎に性能を測り、性能から上記のユーザ要求項目にする。そして要求に応えるために必要かつ十分と思われる性能評価項目をリストアップした。

(1) コントロール・データ・フロー・グラフ（CDFG）

(2) 演算式評価
(3) 状態遷移図(FSM)
(4) データストリーム系列

これらから単独で、あるいは複数を組み合わせることで処理速度などを見積もることができると考えられる。例えば本研究では、処理速度は演算強度のみから導き出すが、そのためにはCDFGと演算式評価のそれぞれの結果を組み合わせることで表現する。これらは図 3の解析器の結果に相当している。モジュール性能は独自の中間表現(言語)として表している。グラフや図と書かれているが、実際には単なるオブジェクトあるいは文字列であり、見やすいように可視化などはしない。必要ならば人間が手作業でする必要がある。

これらの解析には拡張性を十分残してあるので、上記 4 種類の性能だけでは精度などが不十分であったり、ユーザ要求にもっと他の要素を増やしたい時に性能項目を追加することもできる。例えば、PC 上での処理にかかる実時間などを追加することも可能である。

2.3 CDFG

コントロール・データ・フロー・グラフ(CDFG、またはCFG)はデータの流れをその制御(イベント)とともにグラフ化したものである。システムの構造化分析においてよく出てくる手法である。代表的な例を図 5に示す。

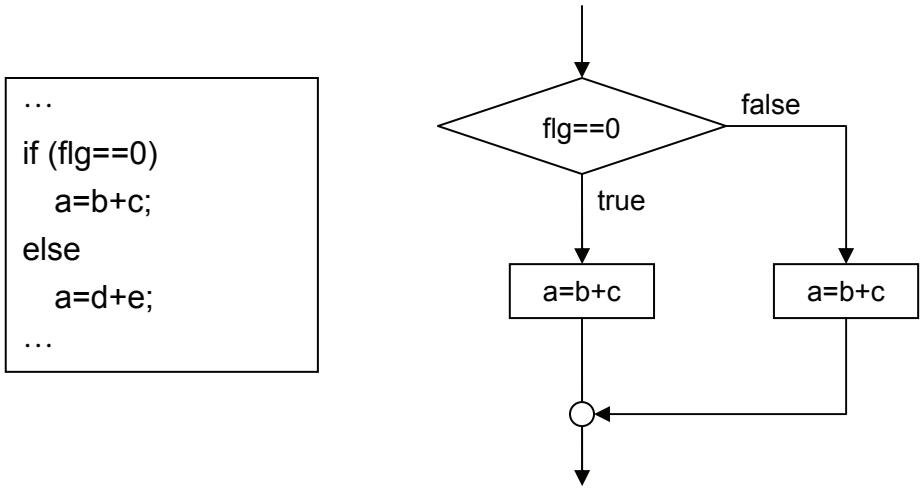


図 5 C 言語における CDFG 例

本研究においては図 5のように処理と分岐だけではなく、その命令の詳しい内容やプログラムの構造の深度、他との組み合わせのためのタグナンバといった独自の情報も付随しているので拡張CDFGと呼ぶべきものである。その一例を図 6に示す。関数毎に切り出されたファイルから右のCDFGインスタンスに変換する。拡張CDFGにおいてループ制御などを含む場合にその深度を付加して多重ループなどで構成されていてもループ回数などを分かるようにしていることが最大の特

徴となっている。

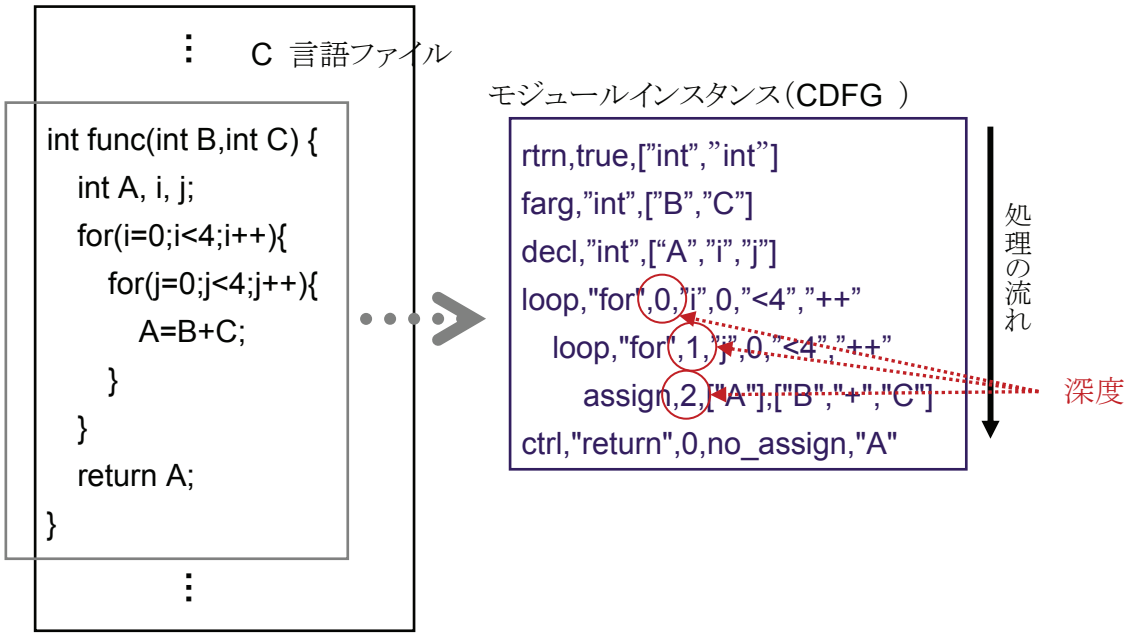


図 6　本システムにおける CDFG

2.4　演算式評価

演算式評価は非常に単純なものであり、単純にCDFGなどで記述される演算命令に重みを付けるだけである。しかし、CPU アーキテクチャが変わればその重み付けも大きく変更される。例えば、乗算器が無ければ乗算命令を実行するために何度もループしたシフトと加算をする必要があり、シフトでもバレルシフタ機能が有るかどうかで数十倍の効率の差が出てしまう。

使用するCPUのアーキテクチャに沿った重み付けをする必要がある。表 1に本研究で使用している演算に対する重み付けの対応表を示す。ただし、今回のものはソフトコアMicroBlazeに特化しており、MicroBlazeはMIPSに準拠したアーキテクチャとなっている。コンパイラに関してもGCCのNoOptimizeレベルを想定している。CPUなどによってこの対応表は変動する。

2.5　状態遷移

状態遷移はシステムの制御としてある関数にまとめられている場合にのみ見る。これは解析を容易にするための工夫である。状態遷移図にはミーリ型オートマトンを参考として作成する。なおここで使用する有限状態機械は全て決定性を持つ。

表 **1**　演算命令に対する重み

演算	重み	演算の意味・備考
EXOR	1	排他的論理和．重みの基礎
ADD	1	加算
SUB	1	減算
MULTI	乗数	乗数によって変動
DIV	除数×5	乗算の 5 倍の重さ
RSH	1	シフト量によって重みは比例する
LSH	1	シフト量によって重みは比例する
ASSIGN	1	代入命令
LP	―	ループ．これ自体に重みはない
BRNC	2	分岐命令．ジャンプを含む
RTN	1	戻り値代入．

ソースコードサンプル

```
func_A(); //関数呼び出し

if (flg_a==0)
   func_B();
else
   for(i=0;i<100;i++)
      func_C();

if (flg_b==0)
   x=y+z;
else
   func_D();

return EXIT_SUCCESS;
```

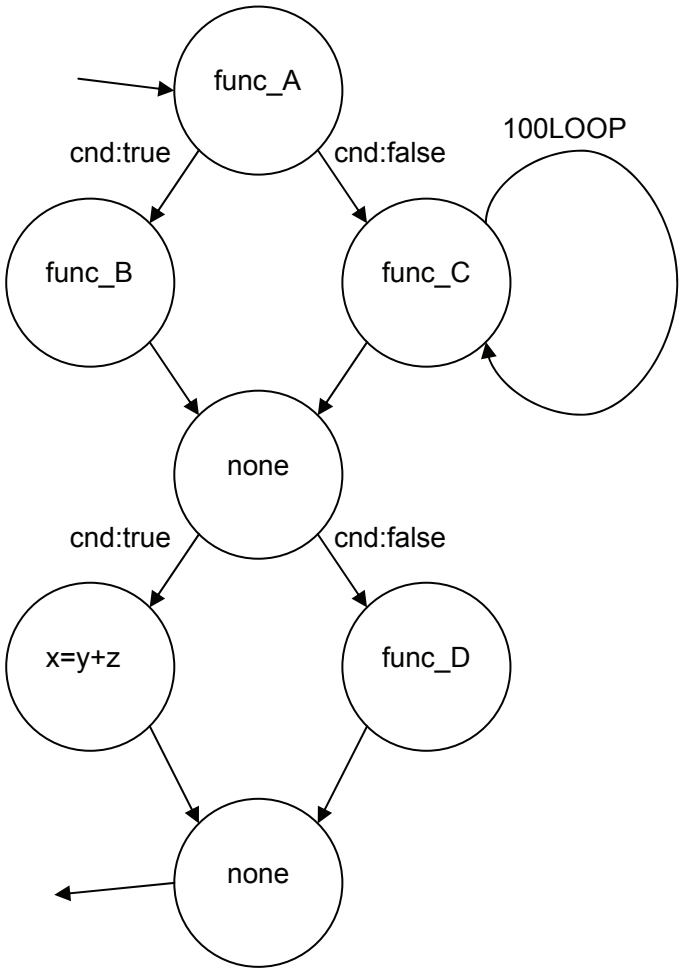


図 **7**　状態遷移図作成例（**Mealy** オートマトン）

図 7にC言語からの状態遷移図作成例をあげる。図 7中で始点側にステートがない遷移矢は初期入力、逆に終点側にステートがなければ最終出力となっている。図 7左のソースコードで”func_*”と記述されている部分が関数呼出である。関数が起動している時、または演算式を実行している時を状態遷移図のステートとし、遷移は基本的にif文のみを想定している。これは条件分岐後にループなどが入ることを想定しているからである。また状態遷移を表現するのに便利であるswitch-case文の形式もサポートする。分岐途中でのgoto命令などによる制御の離脱は当然ながら認めていない。状態遷移においてif分岐から抜ける際には無条件遷移を利用している（図 7の”none”とその入力矢）。

2.6 データストリーム系列

C 言語などのプログラミング言語は大概、シーケンシャルに動作する。しかしハードウェアは物理的にデータが分けられている場合もあり、データや処理の流れが複数、それも並列に動作する場合が多々ある。この流れが全く独立しているならばソフトウェアプログラムにおいても分離して書くことが普通だと思われるので、ハードウェアとの対応も簡単である。しかし 2 つ以上あるデータの流れが特定の箇所で相互に作用することも当然あり得る。そのような場合にはハードウェア回路を設計する難易度が格段に上がると考えられる。そこでこのデータや処理の流れ、本論文ではデータストリーム系列と呼んでいるものを追い、入力から出力まで数量と変化を観測することで設計の難易度に反映させる。

3. MISTY1 暗号のハード／ソフト最適分割

3.1 MISTY1 概要

ハードウェア／ソフトウェア最適分割システムの実験対象とするアプリケーションに MISTY1 暗号を選んだ。MISTY1 暗号は 64bit ブロック共通鍵暗号で、鍵長は 128bit である。暗号強度は変換モジュールに対するループ回数を増減させることによって実現する。ブロック長と鍵長はどのような場合でも変化しない。MISTY1 暗号化処理の基本はループ(N=n×4 {n:1,2,3,…})、データの bit 分割、シフト、さらに入れ子構造にすることである。2005 年には標準化団体 ISO の国際標準暗号の 1 つに指定されている。

図 8にMISTY1 暗号・復号、そして鍵拡張の構成を示す。図 8の(a)基本構成がループの 1 単位であり、これを一定回数繰り返すことで暗号化しており、その中のモジュールは更に細かく処理を分けられる。図 8(a)の処理内容は、被暗号文 64bitブロックをまず 2 分割し 32bitずつに分け、それぞれをFL関数で処理する。FL関数は単純に被暗号文と鍵を同じように分割してから排他的論理和や論和を行う。さらに、その結果の上位 32bitを図 8(b)FO関数を用いて変換し、下位 32bit に対して排他的論理和を行う。図 8の(b)FO関数も同様に 32bit入力を 16bitに分割して鍵と排他的論理和を行うが、さらに内部に図 8(c)FI関数を入れ子にしている。FI関数もbit分割をするが、上位 9bit、下位 7bitと非対称にするのが特徴の一つである。FI関数では"S9"と"S7"というテーブルを持っており、これを参照して変換する。"S9"は 512 個の 9bitデータ、"S7"は 128 個の 7bitデータを要素として持つテーブルである。またFIは上位・下位・鍵のbit列を排他的論理和するときに、上位切り捨てや上位ゼロパディングを行うことも特色である。最終的にそれぞれの関数が分けたbitを全て結合して出力し、64bitまで再結合する。

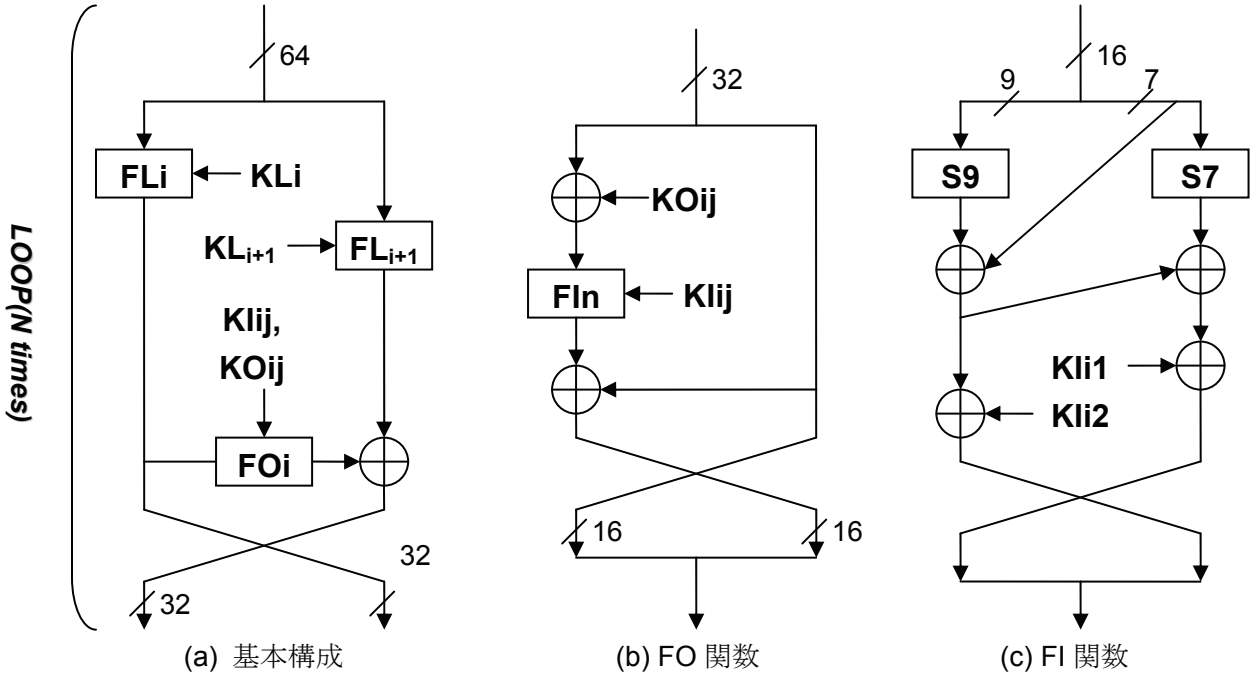


図 8 MISTY1 暗号化

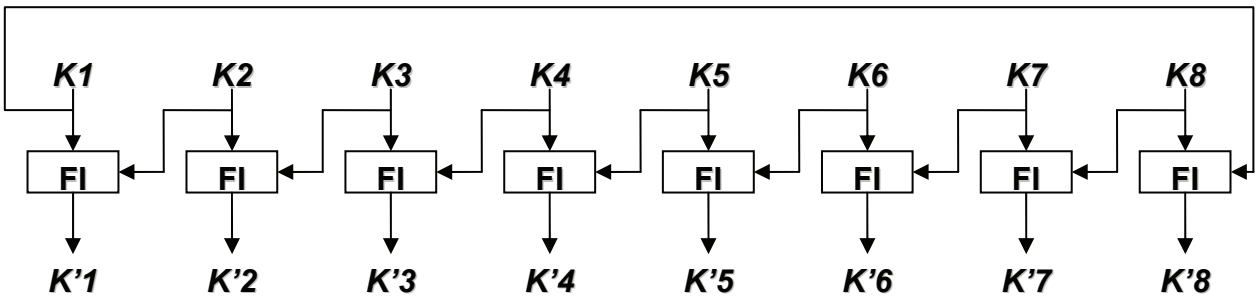
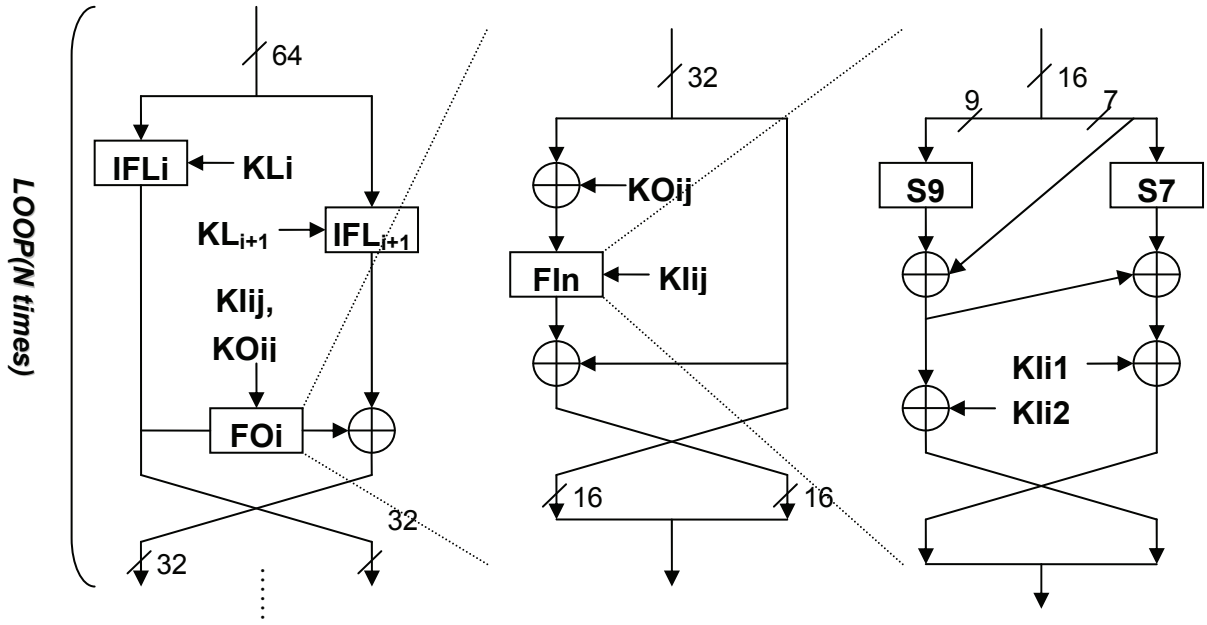


図 9　鍵拡張（KeyScheduling）



(a) 基本構成　　(b) FO 関数　　(c) FI 関数

図 10　MISTY1 復号

表 2　秘密鍵・拡張鍵と"K[L,O,I]"の対応(1≦i≦8,i>8→i:=i%8)

	KO_{i1}	KO_{i2}	KO_{i3}	KO_{i4}	KI_{i1}	KI_{i2}	KI_{i3}	KL_{i1}	KL_{i2}
Key	K_i	K_{i+2}	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) $K_{i/2+4}$

※K:秘密鍵、K':拡張鍵

図 8の他に拡張鍵を生成するKeySchedulingモジュールがあり、構成を図 9に示す。また、復号化は暗号化の逆処理であり、演算強度などは全く同じとなる（図 10）。

図 8について少し詳しく述べると、MISTY1 は基本的にデータのビット単位での分割と結合、排他的論理和、テーブル参照、そしてループが基本となっており比較的、典型的なな形になっている。

表 2に秘密鍵と拡張鍵からどのように各関数に鍵を切り出すかを示す。基本的に秘密鍵か拡張鍵の一部が切り出されてそのまま利用されている。一方、KLはループ回数によって可変となっている

ので、一見するとシステム仕様としてループ回数を可変にすると巨大なメモリが必要に見える。しかし、実際には工夫することによって秘密鍵と拡張鍵のそれぞれ 128bit分の格納領域さえあれば、適宜必要となる分の鍵をそれらからセレクタなどで切り出すことによって実現できる。鍵のためのアドレス計算もシフトなどで簡単に実装可能である。

鍵の対応は少々難解なのでここで例をあげて説明する。秘密鍵と拡張鍵をメモリテーブルに格納しているとして考える。KOとKIはそれぞれインデックスに対応する箇所を切り出してくるだけである。KL も基本的に同じ手法で対応する箇所を切り出すだけなのだが、その"対応する箇所"を探す際にループが奇数番か偶数番かによってパターンが分かれ、さらにアドレス計算をしなければならないことが難点となる。

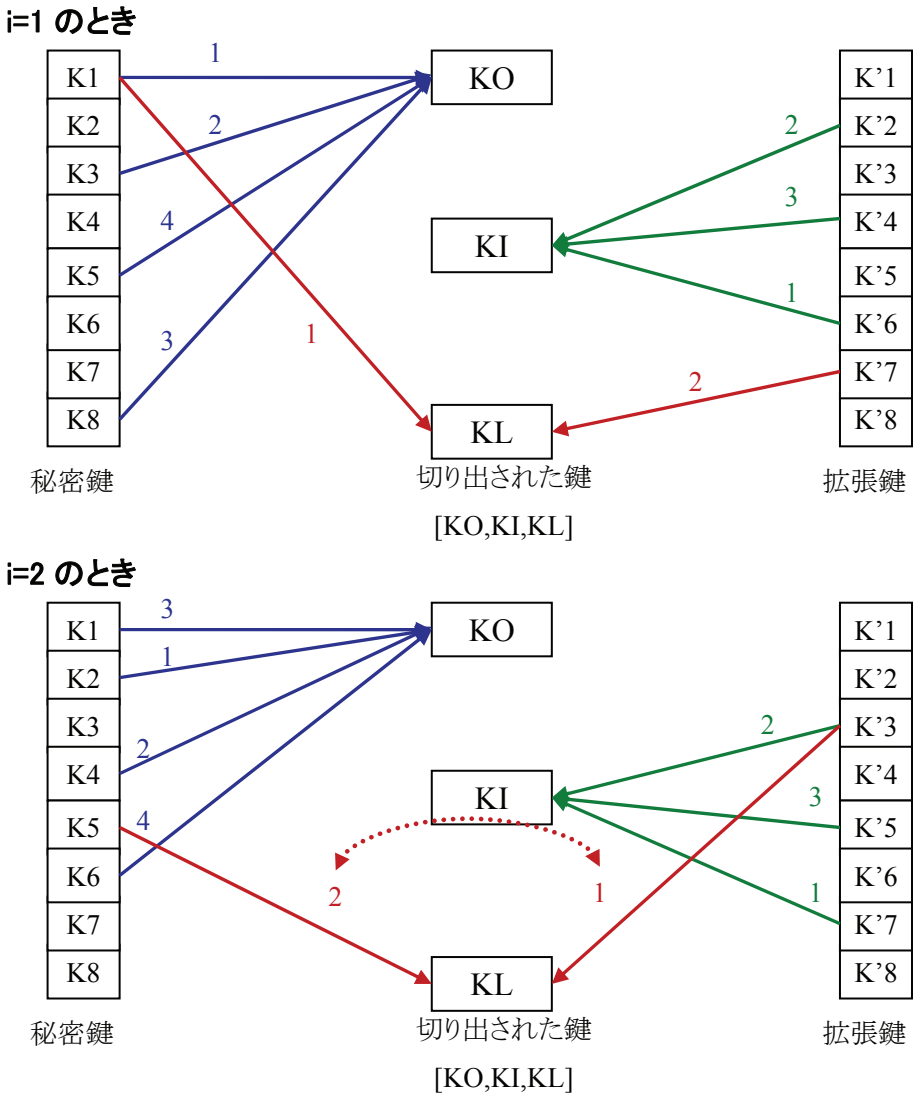


図 11　秘密鍵・拡張鍵の切り出し例

MISTY1 暗号・復号の内部処理は、ブロックをさらに（場合によっては非対称的に）細分化して、

それを入れ子の関数や鍵との排他的論理和、ランダムな表引きによって暗号化を実現する。このような比較的単純な理論を用いることで暗号化を行うが、MISTY1 暗号は線形合同法や差分解読法といった暗号破りに力を発揮する理論に対する耐性・強度は数学的に証明されている。

次にソフトウェアプログラムの構成を示す。最初にデータ構造（変数名、型、意味、グローバル/ローカル）、次に各関数について箇条書きにして述べる。

[1] データ構造（主要部分）

- **unsigned char S7[128] － グローバル**

  128 要素のランダムな表。1 次元配列、アドレスとインデックスが互換
- **unsigned short S9[512] － グローバル**

  512 要素のランダムな表。1 次元配列、アドレスとインデックスが互換
- **int plain0**

  平文。上位 32bit
- **int plain1**

  平文。下位 32bit
- **int K[8] － グローバル**

  秘密鍵。使用するのは 128bit。エンディアンに注意
- **int Kd[8] － グローバル**

  拡張鍵。使用するのは 128bit。エンディアンに注意

[2] 関数

- **int KeyScheduling**

  鍵拡張関数。モード不変。FI 関数を使用。環構造となっているが、非常に単純で ”軽い”
- **int FI**

  9bit と 7bit に分割してテーブル参照と XOR を行う。構成の最小単位。鍵拡張にも用いられる
- **int FO**

  16bit ずつに分割しての FI 関数処理と XOR を行う
- **int FL**

  入力 32bit を 16bit ずつに分割して、鍵と XOR 後、結合
- **int IFL**

  FL 関数の逆処理。これだけは復号で独自に必要となる
- **int main**

  全体の流れを制御。将来的には制御は外部に追い出す可能性大。MicroBlaze での動作結果確認用の出力関数もここに含まれている。この内部で暗号化と復号化の両フローを行っている

MISTY1 暗号・復号についてはこのようになっている。本論文において復号化部分については暗号化の正しさを見るために利用しているが、性能計測などには一切使っていない。これは復号化は暗号化と全く同じ回路構成になると言うことが自明であり、考慮する意味がないためである。

3.2 MISTY1 への最適分割手法の適用

MISTY1 暗号は元々ハードウェアでの設計も意識されており、鍵の扱いを除けば非常に簡単なアルゴリズムと制御から成り立つ。そして推奨ループ回数の 8 回で 64bit 平文を暗号化する程度ではどのような環境でも一瞬で終わり、かつ回路としても簡単にすぎるので、MISTY1 暗号処理に恣意的に負荷をかけることで実験の観測などをやりやすくする。そのためには 2 つのアプローチがある。

1. 64bit 平文ではなくもっと大量の平文を 8 回ループで暗号化する。
2. 64bit 平文を多数回ループさせることで暗号化する

本論文では 2 番目のループ回数を操作する方法をとっている。その理由は大量のデータを扱うとメモリがそれだけ必要になってくるからである。FPGAは再構成可能なLSIであり汎用性も高いが、メモリ使用に関しては性能的な面や、技術的な面で厳しい部分がある。従って、実験の方針として簡単に行うことが可能である 2 番目のループ回数制御で対応する。これならば図 12のようなインライン展開による巨大モジュールの合成も容易に行える。

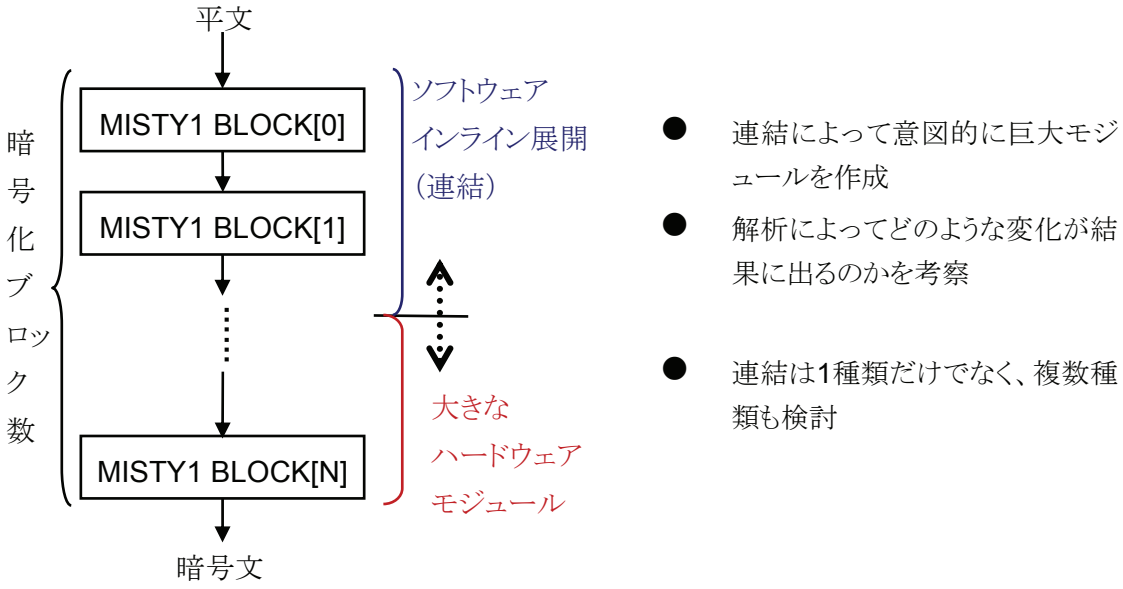


図 12 ループ回数調整による実験

3.3 MISTY1 暗号回路設計

MISTY1 暗号をハードウェア回路としてVerilog-HDLで設計した[17]。その回路構成を図 13に示す。構成は図 8(a)のループ 1 回分を基本としており、それを制御して任意の暗号強度（ループ回数）を達成する。秘密鍵や拡張鍵などについては、これらはモジュールの性能比較の対象外となっているので予めハードウェアモジュール内部にテーブルとして値を保持しておき、セレクタなどで適時、暗号化処理コアに対して適切な鍵を供給する。これには実装時におけるCPUとのインタフェースの簡易化が大きな目的として含まれている。鍵をループする毎に生成して送ることになると実装上、大変な手間がかかる。

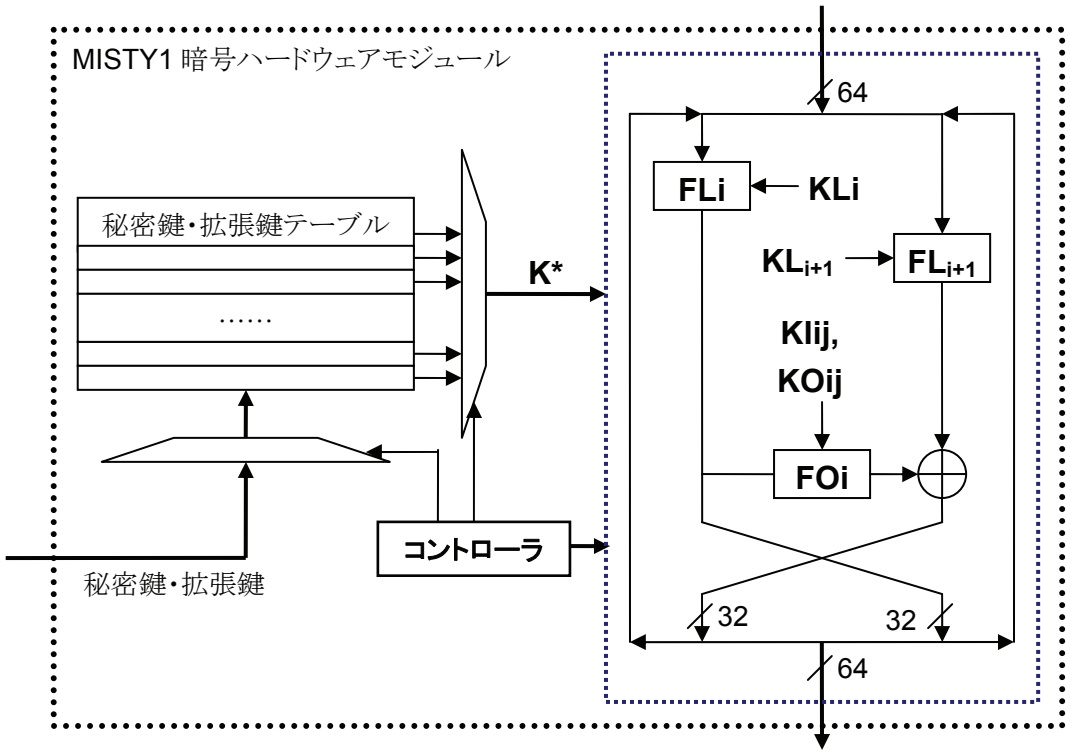


図 13　MISTY1 暗号モジュール

3.4 検証システム

Xilinx 社のソフトコアプロセッサ”MicroBlaze”を中心として使用する。MicroBlaze は MIPS に準拠した 32bit-CPU アーキテクチャになっている。これは元々PowerPC プロセッサと互換で設計されているからである。そのためにデータの扱いがビッグエンディアンになっているのでソフトウェアでのバイト操作では多少注意が必要となる。また、浮動小数点演算器などはオプションとして指定できるが通常は付属しない。

MicroBlazeは統合環境ツールEDKと論理合成ツールISEと言うツールと密接に関係しており、これらのツールでパラメータ設定やコーディングをすることでC/C++プログラムとハードウェアペリフェラ

ルとの協調動作が可能になる。図 14に本研究で使用した検証システムの構成を示す。MicroBlazeプロセッサにBlockRAMを接続し、これに命令とデータを格納する。BlockRAMのサイズはチップにもよるが、最大で 64KBの領域を確保できる。ユーザが作成したシステムのハードウェア処理に当たるモジュールや評価用のクロックカウンタなどは、それぞれのモジュールをVerilog-HDLで設計し、何らかのバスを通じてCPUとコミュニケーションを取ることになる。以下に、その通信手段について記載する。

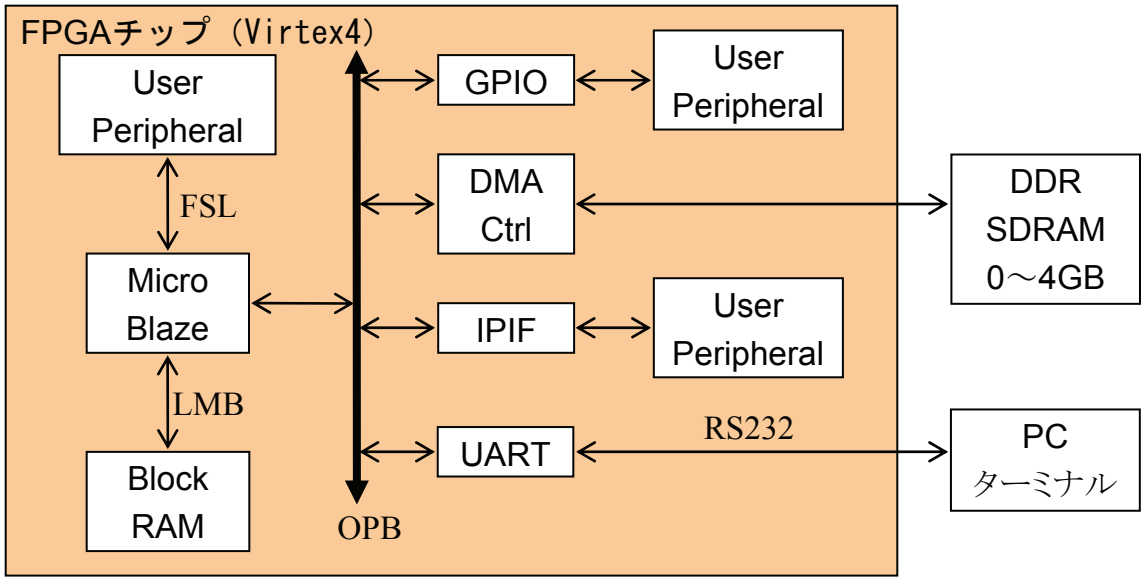


図 14　検証システムの構成

1. On-chip Peripheral Bus 経由

On-chip Peripheral Bus（OPB）は汎用的なバスアーキテクチャであり、多くの機能が標準で搭載されている場所でもある。シリアル転送用 UART や外部メモリへのアクセスなどは OPB を経由して連結される。ユーザペリフェラルも、ネゴシエーション機能付きバッファ「GPIO」やユーザが独自に仕様を定義できる「IPIF」を介して MicroBlaze と繋げることができる。利点は汎用的で扱いやすく使用できる IP も SDRAM などの DMA コントローラなど豊富にあるが、MicroBlaze との通信には少々レイテンシがあり、読み出しや書き込み動作 1 度につき 10 サイクル以上は最低でもかかると考えられる。

2. Fast Simplex Link 経由

Fast Simplex Link（FSL）は MicroBlaze コアと直結したバスであり、FIFO で構成される。入力出力ともに最大 32bit 幅で、FSL1 ラインで入出力がセットになっている。FSL バスは 8 本まで MicroBlaze と繋げることができる。簡単なネゴシエーション機能によって通信し、最大でも 2 クロックサイクルのレイテンシでデータの送信/受信を行える。非常に簡単にかつ高速にデータ転送ができる機構である。ただし簡便な分、インタフェースやプロトコルが厳密に規定されてい

るので、それらの改変はできない仕様となっている。

3. Local Memory Bus 経由

Local Memory Bus（LMB）は MicroBlaze の命令メモリなどにアクセスするためのバスである。バス幅は 32bit となっている。このバスは OPB と比較すると高速ではあるが、ユーザが利用すると言うことはメモリへのアクセスを掠め取ることであり、それに応じた工夫が必要となる。

本論文では FSL を通じて接続している。これは EDK 上での設定の簡便さと通信速度の高速性を重視した結果である。

計測結果は OPB にシリアル通信用 UART の IP を搭載して、WindowsOS のハイパーターミナルアプリケーションにシリアル転送することで動作結果を確認した。

4. ハード／ソフト最適分割システムによる実験

4.1 実験方法

最適分割システムを適用した実験には4節で述べたMISTY1暗号回路を検証システムに搭載してクロックサイクル数などを計測する。

その実装概要を図 15に示す。MicroBlazeのFSLを用いた実装を行っているが、FSLは 32bit幅の入出力バスであり、これらを束ねたりI/Oを改変したりすることは不可能である。そのため、必要となるデータをバッファリングしてからMISTY1 コアジュールを起動して、結果を取得する。

使用メモリ量などはツールの合成結果から得ることができるのだが、ハードウェア／ソフトウェアモジュールの処理に必要なクロックサイクル数の計測だけは実機上で動かさないことにはなかなかデータを得ることが難しい。そこで32bitのカウンタを設置し、MicroBlazeからMISTY1暗号モジュールが起動したときにほぼ同時に起動して終了するまでの時間を計測することで処理時間を測る。

なお、システム全体の動作周波数は100MHzに固定となっている。

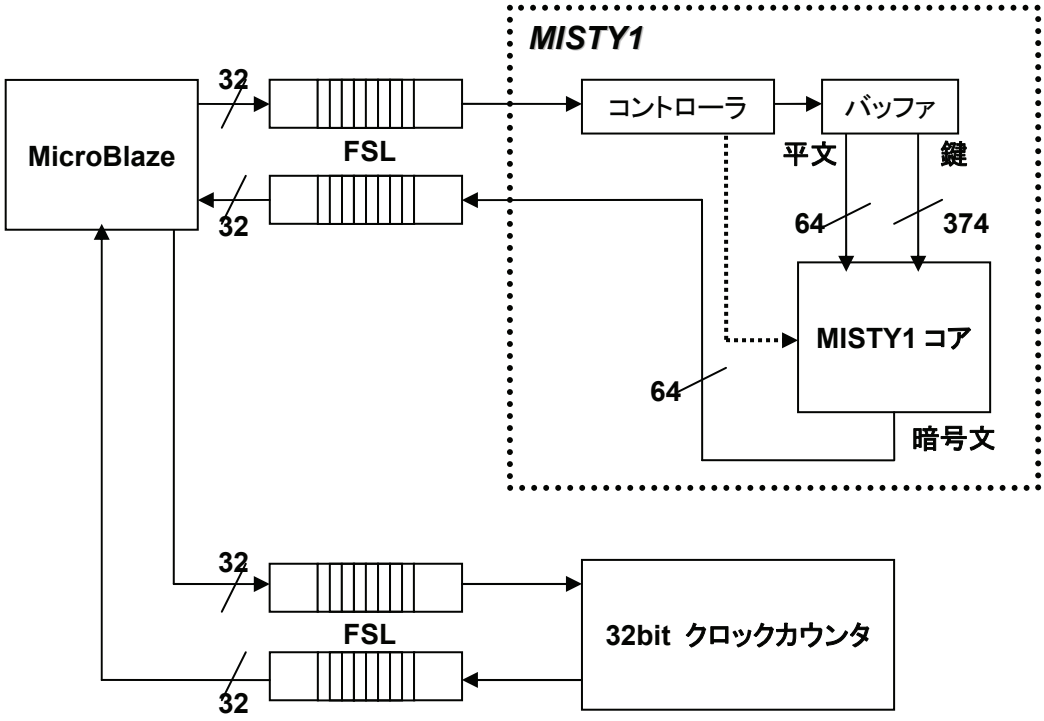


図 15 MicroBlaze 連携 MISTY1 検証システム構成

4.2 MISTY1 実験結果

まずMISTY1 暗号のループ 1 回分の処理コア（図 15）のハードウェアとしての性能について表3に記載する。MISTY1 暗号モジュールにはパイプライン化や並列化などのアーキテクチャは全く使用していない。1ブロックの暗号化を行うのみである。

ターゲットとなるFPGAチップはXilinx社のVirtex-4 LX60 を対象としている。合成結果の取得にはEDKツールとFoundation ISEツールを使用している。表 3の動作周波数はそのまま最大遅延

からの逆算、レイテンシは結果が出力されるまでにかかるクロックサイクル数、回路規模はFPGAロジックセルの使用スライス数、メモリ量はソフトウェアプログラムの使用する命令メモリとデータメモリを合わせた値となる。

表 3　MISTY1 コアモジュール性能

項目	動作周波数	レイテンシ	回路規模	メモリ量
数値	397[MHz]	4[cycles]	4932[slices]	4843[Bytes]

動作周波数に関しては、システムクロック限界が 500MHzとなっているので優秀な性能だと考えられる。特に高速化を意識せずに、これだけの速度性能となっているのは MISTY1 暗号のアルゴリズム自体がハードウェアを意識して作成されていると言うことも関連している。回路規模に関しても、テーブルメモリを含んだ状態でターゲット FPGA チップの 18%程度となっている。

レイテンシなどの周りの要素を含めて理論最大スループットを計算するとおおよそで 4Gbpsとなるので、実際の製品と比較しても遜色は無いと思われる†2。このような性能の回路を図 15のように接続して動作させることで実測を行った。

4.3　MISTY1 暗号実験の評価と考察

5.2 節で述べたような構成で実測などを行うが、現時点での最適分割システムは CDFG 生成、演算式評価、各モジュールメモリ量算出の 3 項目についてそれぞれ導き出すことが出来るようになっており、これにより処理速度・回路規模・メモリ量の評価はできるが残りの消費電力と工数に関してはツールを作成・整備中なのでまだ不十分なところがある。従って本論文では処理速度・回路規模・メモリの点から本システムの有効性評価などを行う。

表 4に本システムでMISTY1 暗号Cソースコードを解析した結果を記載する。

表 4　MISTY1 暗号解析実験結果

HW ループ回数	0 回 （SW のみ）	8 回	32 回	64 回	100 回 （HW のみ）
処理速度	92330	85120	63056	24406	542
回路規模	0	13265	13270	13276	13282
メモリ量	38048	37668	25885	13728	64

表 4において、処理速度は生成されたCDFGと演算式評価結果を統合して、回路規模もCDFGと演算式評価から、メモリ量はグローバル変数やローカル変数を解析した結果から導き出した。それぞれの単位は 2.2 節で述べた通りである。

---

†2 関連研究[17]でパイプライン化と並列化によるスループットの向上を図った結果、最大で 20Gbps の性能を出した。これは現行のどのような通信方式よりも高速である。

表 4はそれぞれの単位で数値を計算されているので、各々の傾向は大体見て取れるが相互の影響を考慮して評価することは困難である。そこで、独自に考えた評価式を導入することで定量的に評価する。その評価式を式に示す。

$$Eval = \sum \left[Wu_{item} \times \frac{Std\left(Val_{item}\right) - StdMin}{StdMax - StdMin} \right]_{pattern} \quad \cdots\cdots式(2)$$

式(2)の左辺"Eval"はそのパターンにおける総合評価結果であり、システムとして該当パターンのプライオリティの高さを表す。数値が大きいほど優秀である。"Wu"はユーザ要求からの重み付けであり、処理速度や回路規模などに応じた比率が記されている。。"Std()"関数は偏差値を算出する関数であり、あるパターンでの処理速度などの偏差値を導く。偏差値は一般的に使用される"(ある値－平均値)*10/標準偏差+50"という式を採用している。そして、"StdMax"と"StdMin"はそれぞれある性能項目における偏差値の最大値と最小値を表す。これらを使用しているのは偏差値を独自に標準化するためである。標準化は 0.0～1.0 の範囲に値を収めるためである。Val はある項目(処理速度など)に関する解析値、もしくは実測値である。

この評価式を使用することで、異なる価値基準の項目を同列にして定量的に、ひいては視覚的に優先度(プライオリティ)を判断することが可能となる。

表 4の結果と式(2)を利用して分割パターンのプライオリティを計算した。その結果を表 5に載せる。表 5の一番上の行がMISTY1 暗号ループ 100 回中のハードウェアの担当回数、その回数の下が優先度になる。また優先度を導く際に使用した処理速度や回路規模に対する重み付けを左側に記載しており、分類するためにA～Fのアルファベットを割り振った。重み付けは(A,B,C,)が処理速度・回路規模・使用メモリ量をそれぞれ極端に重くした場合であり、即ち何か 1 カ所だけを重視して他はほとんど無視するといった構成となっている。(D,E,F)は少しだけ速度など 1 カ所を重視するが、他の項目も無視しているわけではない、という状況を仮定している。

表 5　MISTY1 暗号評価結果

HW ループ回数			0 回	8 回	32 回	64 回	100 回
重み付け (Spd:Gt:Mem)	A	8:1:1	0.10	0.06	0.29	0.66	0.90
	B	1:8:1	0.80	0.06	0.14	0.20	0.00
	C	1:1:8	0.10	0.02	0.29	0.59	0.90
	D	6:2:2	0.20	0.05	0.26	0.57	0.80
	E	2:6:2	0.60	0.02	0.13	0.28	0.40
	F	2:2:6	0.20	0.02	0.26	0.53	0.80

※Spd:処理速度, Gt:回路規模, Mem:メモリ量

表 5をグラフ化したものを以下に示す。(A,B,C)グループは図 16に、(D,E,F)グループは図 17にグラフ化する。

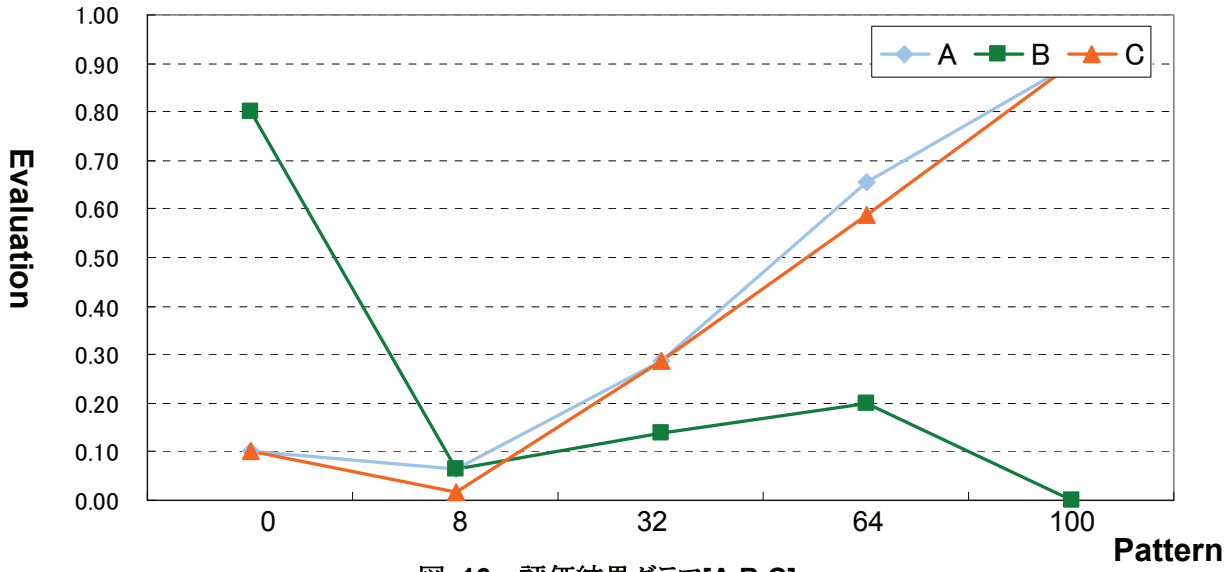


図 16　評価結果グラフ[A,B,C]

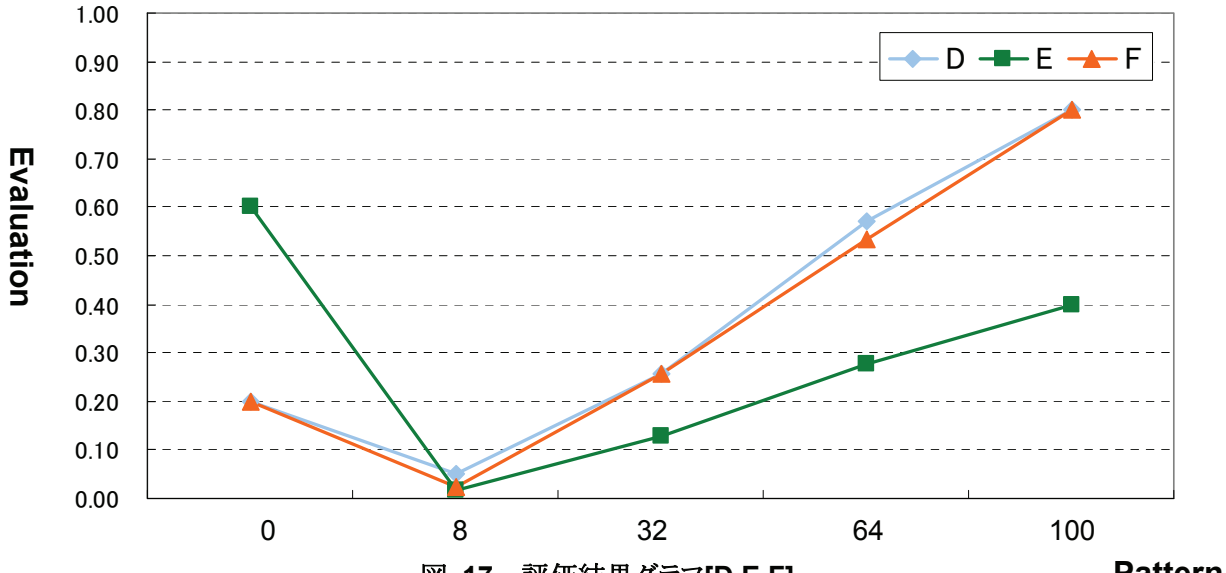


図 17　評価結果グラフ[D,E,F]

図 16と図 17を考察すると、おおよそ似たような傾向になっており、特にハードウェアのループ回数が8回の点で評価値が急激に降下している点などが全パターンで共通している。さらに、ループ8 回以降は評価が線形的に右肩上がりになっているところが見られる。これらのことから、MISTY1 暗号の性能は処理負荷が高ければ高いほど向上していくことが分かる。MISTY1 暗号はアルゴリズム的に見てデータの依存性などがほとんど無く、優秀と言える。

次に着目する点は図 16の"B"のループ 8 回以降があまり性能向上していない点である。"B"は

回路規模を極端に重視して性能を測った場合である。解析結果から見ると、回路規模はループ 8 回以降はあまり増加しない。これは暗号化処理のコアモジュールをソフトウェア制御で何度も再利用しているためであり、回路規模が増加しなければ性能向上は高いと予想されるが、実際にはソフトウェアのみでの動作が回路規模 0 で計算されているので、それと比較して速度などの性能向上にあまりメリットがないことがあげられる。一方で、図 17の回路規模を少々重視したタイプである"E"はハードウェアのループ回数を増やすと順調に性能向上をしている。これは"D"と"F"の性能向上に大きく助けられているおかげである。しかし100回全てをハードウェアにつぎ込んでもソフトウェアのみでの動作の評価より低い点が、MISTY1 暗号において回路規模のみから性能を追う限界を示している。

他の処理速度とメモリ量重視は非常に似たような結果になっており、また評価値についてもループ8回を除けば全てのパターンにおいてハードウェア化した方が勝っている。このことからMISTY1暗号は処理速度とメモリ量の削減ではハードウェア化の利点が非常に大きいと判断できる。
図 16と図 17のソフトウェアのみでの動作において、図 16では(A,C)と(B)の間が極端に離れているが、図 17のハードウェアループ回数は 0 の時はその差が少々短くなっている。これは処理速度とメモリ量、そして回路規模の相互影響が強まっているためであり、ここに最適分割システムのバランスを見るための効果が現れていると言える。

4.4 AES暗号実験結果

MISTY1 暗号はループ回数を調整することで任意の大きさのモジュールを作ることが出来るが、その場合には性能や結果が線形的になりがちであり、複数モジュールに分割するにしてもそれぞれがまた線形的な結果を出すので、評価をするにはもっとバリエーションに富んだアプリケーションの方が自然である。そこで、我々の研究室で過去に AES 暗号を設計・実装して計測を行っているので、これらのデータはすでに十分取得されているので、追加実験を行った。

まずAES暗号の概要と分割方法について述べる。AES暗号はRijndaelアルゴリズムを採用しており[8]、入力データに一定回数のデータ変換を行うことで暗号化する。鍵長は 128,192,256 ビットの 3 種類である。鍵長によって暗号化のループ回数などが変化する。今回、設計する暗号システムは 128 ビットの暗号強度のみに限定している。図 18にAES暗号の処理手順を示す。データ変換にはAddRoundKey変換、SubBytes変換、ShiftRows変換、及びMixColumns変換の 4 種類がある。さらに鍵拡張部(KeyExpansion)を加えることで大きく 5 つの機能ブロックに分けられる。

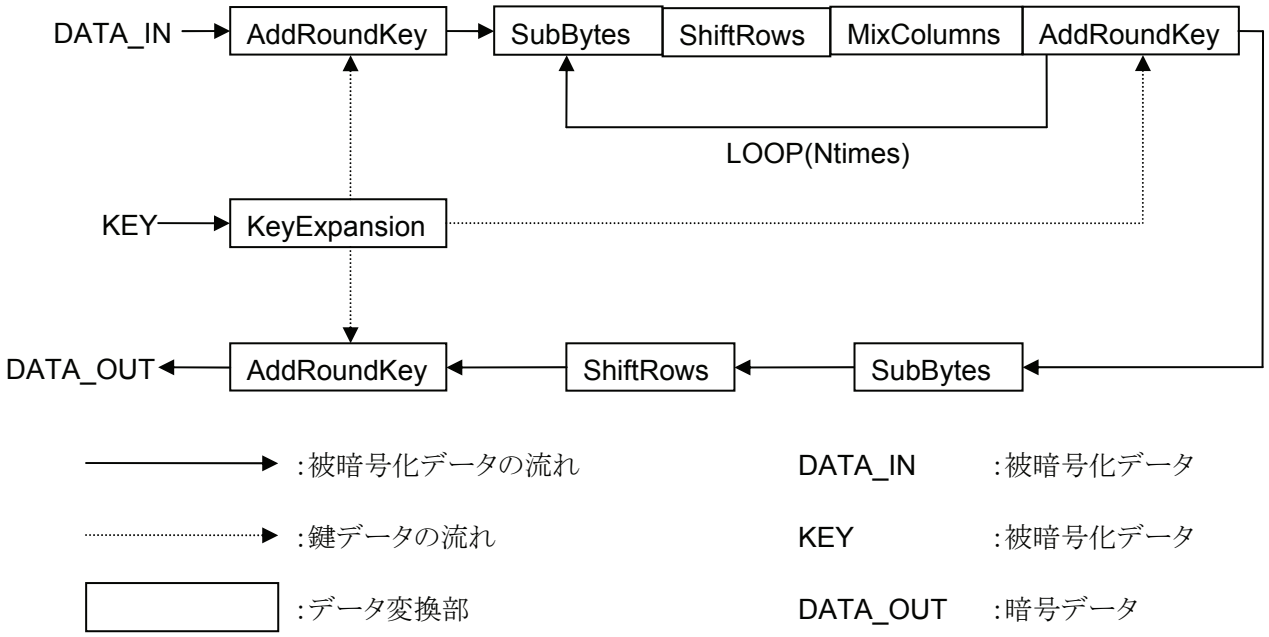


図 18　AES 暗号処理フロー

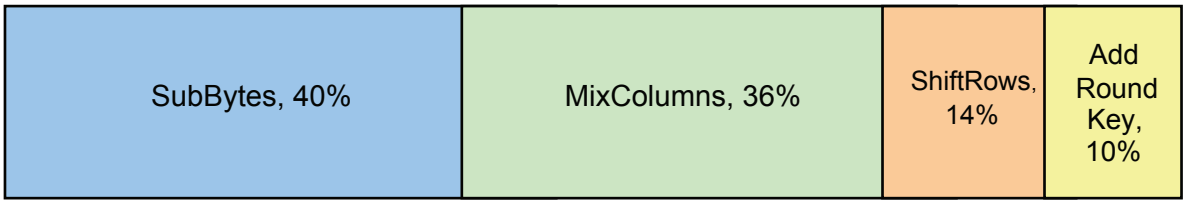


図 19　各モジュールの CPU 処理負荷

KeyExpansion変換は暗号化に使用する鍵を拡張する処理である。この処理はデータの大きさに関わらず一定なので、処理データが巨大になれば全体に占める割合も小さくなる。AddRoundKey変換では入力データと鍵データの排他的論理和をとる。SubBytes変換は本来、数学的で複雑な処理であるが、入力に対して出力が一意に決まるので、テーブル参照で実現している。ShiftRows変換は入力データを 2 次元配列と見て、各行でシフトを行う。ただしシフト量は行によって異なる。最後のMixColumns変換は、数学的なガロアフィールド理論に基づいた行列演算をしなければならない。ガロアフィールド理論に基づく演算は難解であるが、行列による積和演算を行うので加算と乗算についてのみ考慮すればよい。図 18のような 5 つのモジュールでAES暗号は構成されるが、本論文ではKeyExpansionモジュールは除いて検討する。KeyExpansionモジュールは暗号化ストリーム全体を通して 1 度しか起動しないので、長期的に見ると処理負荷が 0 に等しくなるからである。

図 19にAES暗号の 4 つの処理モジュールをMicroBlazeでソフトウェア動作させたときのCPU負荷を示す。ここから、MixColumnsモジュールとSubBytesモジュールが圧倒的にCPU負荷が高く、重たい処理を行っていると分かる。このデータを参考にしてAES暗号の 4 つのモジュールをどの様

にハードウェアとソフトウェアに切り分けるか考案し、そのパターンを表 6に示す。処理負荷の高いモジュールを優先的にハードウェア化して、計測値などの変化を見る。

表 6　AES 暗号分割パターン

分類	ハードウェア処理部	ソフトウェア処理部
ア		SubBytes, MixColumns, ShiftRows, AddRoundKey
イ	MixColumns	SubBytes, ShiftRows, AddRoundKey
ウ	SubBytes	MixColumns, AddRoundKey, ShiftRows
エ	SubBytes, MixColumns	ShiftRows, AddRoundKey
オ	SubBytes, MixColumns, ShiftRows	AddRoundKey
カ	SubBytes, MixColumns, AddRoundKey	ShiftRows
キ	SubBytes, MixColumns, ShiftRows, AddRoundKey	

表 6のパターンで、AES暗号を実機で数値測定した結果を表 7に示す。処理速度の単位はクロックサイクル、回路規模はゲート、メモリ量はバイトである。

表 7　AES 暗号の実機計測結果

	ア	イ	ウ	エ	オ	カ	キ
処理速度（サイクル数）	38075	26959	25859	14743	11313	13979	10549
回路規模（ゲート）	0	324	9441	9765	9765	9957	9957
メモリ量（バイト）	4387	3935	4259	3807	3731	4047	3971

ここまでは以前に我々が研究してきた部分である。次に、新規部分としてAES暗号を本システムに通して解析した結果からの性能予測値を表 8に記載する。表 8の数値は本システムの単位に則っている。

表 8　AES 暗号の最適分割システムによる解析予測結果

	ア	イ	ウ	エ	オ	カ	キ
処理速度	4212	1872	3636	1152	581	576	9
回路規模	0	113184	152208	265392	279216	270288	284112
メモリ量	4392	3312	4032	612	252	360	8

4.5　AES暗号実験の評価と考察

実際の測定値と本研究のシステムでの予測値が出てきたので、比較することで有効性などにつ

いて論じることが出来るが、このままでは数値の基準が異なるので比較しづらい。よって、式(2)を利用してMISTY1 暗号と同様に評価することで基準の統一を行う。その評価結果を図 20と図 21に載せる。表 9が実機から得た性能を評価したものであり、図 20にそれをグラフ化した。表 10が本システムで予測した性能を評価したものであり、同様に図 21にグラフ化した。これらを使って考察する。

表 9　AES 暗号の実測値からの評価結果

HW ループ回数			ア	イ	ウ	エ	オ	カ	キ
重み付け（Spd:Gt:Mem）	A	8:1:1	0.1	0.49	0.38	0.77	0.88	0.75	0.86
	B	1:8:1	0.8	0.88	0.11	0.19	0.21	0.14	0.16
	C	1:1:8	0.1	0.69	0.21	0.79	0.9	0.5	0.61
	D	6:2:2	0.2	0.57	0.32	0.69	0.79	0.63	0.73
	E	2:6:2	0.6	0.8	0.16	0.36	0.41	0.28	0.33
	F	2:2:6	0.2	0.69	0.22	0.7	0.8	0.49	0.58

表 10　AES 暗号の解析予測値からの評価結果

HW ループ回数			ア	イ	ウ	エ	オ	カ	キ
重み付け（Spd:Gt:Mem）	A	8:1:1	0.1	0.53	0.16	0.68	0.79	0.79	0.9
	B	1:8:1	0.8	0.56	0.39	0.21	0.19	0.22	0.2
	C	1:1:8	0.1	0.31	0.13	0.77	0.84	0.83	0.9
	D	6:2:2	0.2	0.50	0.19	0.62	0.71	0.71	0.8
	E	2:6:2	0.6	0.52	0.32	0.36	0.37	0.39	0.4
	F	2:2:6	0.2	0.38	0.17	0.68	0.74	0.73	0.8

ここではパターン毎の傾向などを考察するのではなく、実機計測と予測の両者を全体的に見て比較することで、最適分割システムについて考察する。両者を比較すると傾向として似ていると見られる。A～F のそれぞれの動きを比べると似たような形になっており、特に”ウ”の時にプライオリティが急激に落ちるところや、”キ”の方向に遷移するときに”B”と”E”がやや低迷している点などが類似している。このことは考案した本研究のシステムが実際にある程度有効だという証左である。

しかし、実測値からの評価の方が比較的、起伏の激しい折れ線グラフになっているのに対し、予測値からの評価結果は少々滑らかになっている。これは実機での何らかの細かな動きを本システムでは考慮していないため、及び評価方法が完全ではないためだと考えられる。

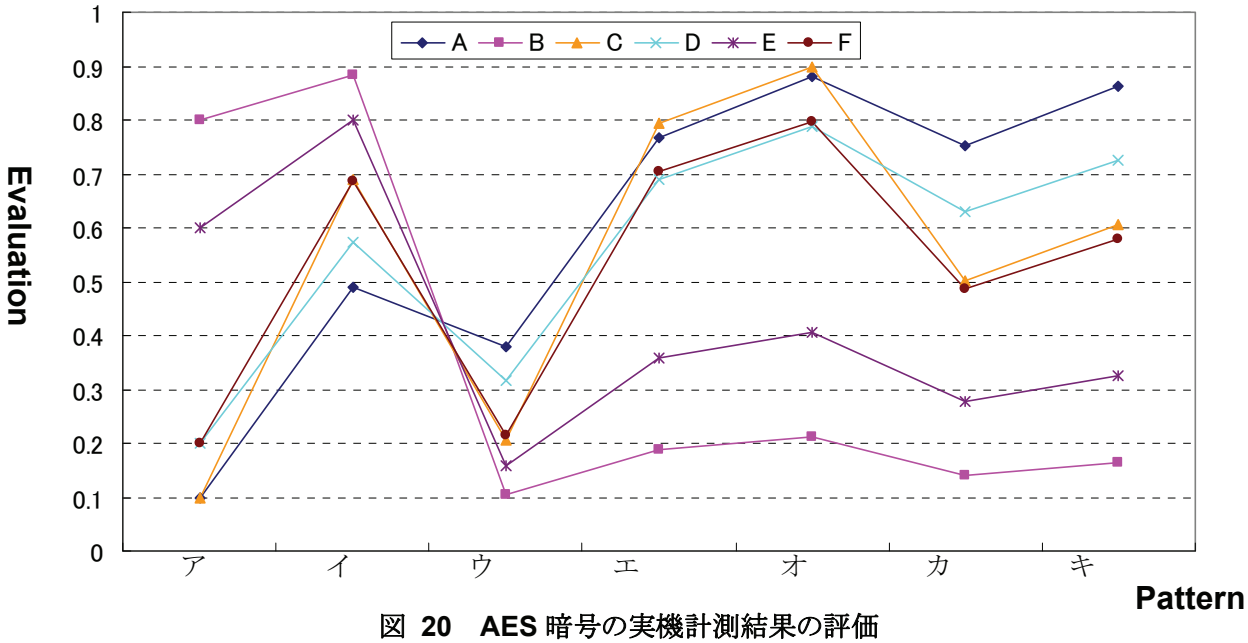


図 20　AES 暗号の実機計測結果の評価

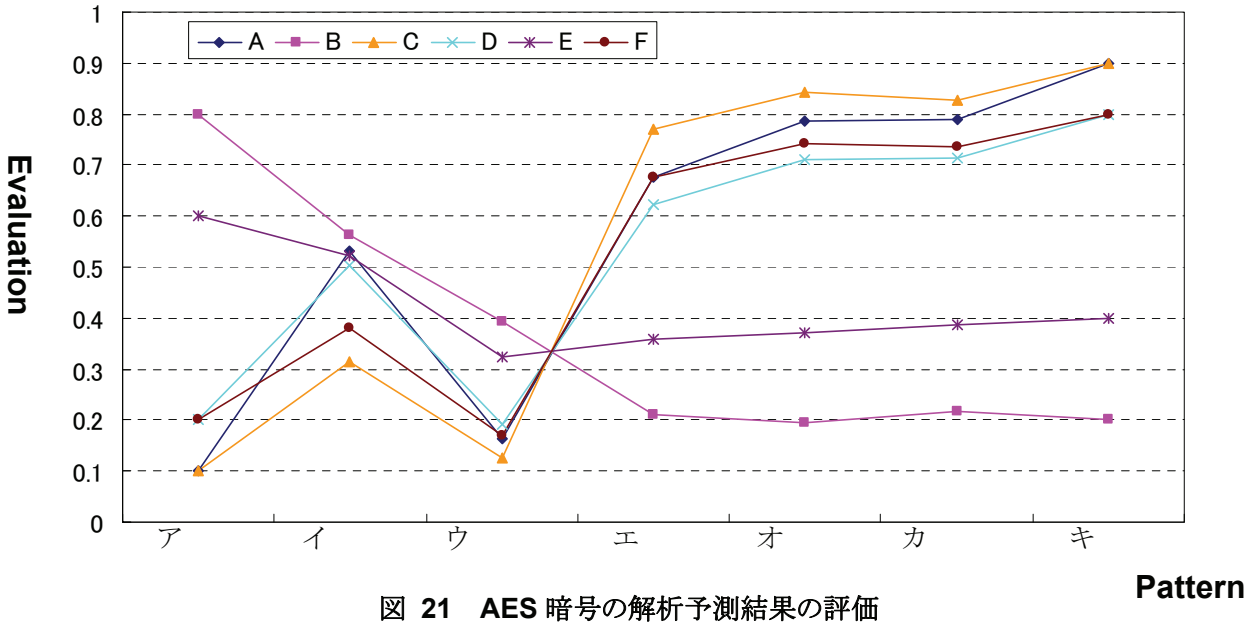


図 21　AES 暗号の解析予測結果の評価

5. おわりに

本論文ではハードウェア／ソフトウェア協調設計における設計空間領域の探索する最適分割システムを提案し、その構築した。設計空間領域を分割するために C ソースコードを利用して早期の協調設計への移行を可能にし、またそれを支援するための C 言語プロトタイプ解析システムをツールとして考案し、作成した。現時点で本システムの完成している範囲を検証するために、検証対象アプリケーションとして MISTY1 暗号を利用し、解析結果から考察を行った。また追加実験として AES 暗号を使用して解析予測結果と実際の性能数値を評価、比較することで現実的に有用であることを示した。

システムの課題として、ツールの完成が第一にあげられる。現時点ではツールは解析器部分がほぼ完成に近い状態であるが、性能分析系において、まだ消費電力や工数の見積もりができていない状態にある。特に工数については人間の能力という曖昧さを含んだ内容となるので困難ではあるが、何としても実現してもらいたい。工数見積もりの例としては設計者に対するアンケートで数段階程度に能力を割り振ることや、アプリケーションの難易度に関しても状態遷移図以外にも単純にモジュール毎のコード行数を勘案することで導き出せると考えている。そして、性能予測手法の妥当性に関しても、より詳細な証明や検証が求められると考える。

次に、本研究の意義としての課題や懸案事項について述べる。ここでは、引き続き研究を行う後進たちのために、私が実際に最適分割システムを構築している際に、重要だと考えた、もしくは実感した点を記している。

＜設計・検証期間の短縮＞

FPGA という画期的な LSI が開発され、目覚ましい発展を遂げており、ハードウェア回路の検証環境は格段に改善された。しかしその影響で設計期間の短縮がさらに進んだことも事実だと考えられる。如何に効率よく設計・検証を行うかはシステム開発にとって至上命題である。IP など過去の、もしくは他所の財産をフルに活用することこそが重要である。自身のみで全てを賄うなど不可能となっているのが、LSI の開発の現状である。

＜仕様策定の重要性＞

設計技術のさらに上位に位置する能力が仕様設計である。仕様を策定すると言うことは開発を管理するということである。ソフトウェア開発プロジェクトでは開発管理はすでに確立された技術だが、ハードウェア開発は長期間にわたることもよくあることなので等閑になりがちである。しかし、長期間の設計開発においてこそ手戻りの発生は避けるべきであり、その意味でもハードウェア開発にこそ有用であると考える。協調設計はハードウェアとソフトウェアの長所を取り合わせようという分野だが、反面、短所としてソフトウェアのやり方にハードウェアが、ハードウェアのやり方にソフトウェアが引き摺られてしまう面もあるように見受けられる。このようなことのないように、曖昧さを排除して線引きをする「仕様を策定する」行為は非常に重要だと考えられる。

＜性能予測の独自性＞

現在、企業でもシステム性能の統合的な解析を推進するツールなどは開発されてきている。

この場合にはタイミング解析などを利用して詳細な性能値を算出する手法が主流である。この点で争うことにも意味はあると思うが、独自性に欠ける研究になってしまうので、もっと大きな視点から研究を進めてもらいたい。例えば、本研究における工数の推定などはおそらく、どのようなツールでも考えてはいないはずである。

我々の研究室ではハードウェアとソフトウェアの相互理解を深めることを 1 つの重要な命題としている。ハードウェア、ソフトウェアをそれぞれ深く理解しようとするときには避けては通れない道であると思われる。本研究での最適分割システムを利用し、理解することで特にソフトウェア専門の技術者に対して、ソフトウェアプログラムと CPU などのハードウェアの深い関連について興味を持ち、把握できることを期待する。

現在の LSI 業界は FPGA によって検証環境は身近になったが、EDA ツールが 1 年間 1 ライセンスで数千万円もかかるような、非常に高価なものであることに変わりはない。ここに一石を投じることができるようになることが理想であるが、ソフトウェアとハードウェアの両方に非常に深い知識と経験がある人間でなければ、そのようなツールは開発できない。

そして、ハードウェア／ソフトウェア・コデザインの分野において、両者を繋ぐ貴重な人材となってくれることを期待する。

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、小柳滋教授に深く感謝いたします。

また、最適分割システムの解析ツールの発想の原点となった C 言語からの高位合成ツール”CCAP”を開発している関西学院大学の石浦菜岐佐教授、財団法人京都高度技術研究所（ASTEM）ソフトウェア研究室室長の神原弘之氏、名古屋大学の冨山宏之助教授、ならびに”CCAP”を開発されている関西学院大学の学生の皆様、MISTY1 暗号のハードウェア回路を作設計した立命館大学 高性能計算研究室の和田氏、本研究に関して貴重なご意見をいただきました高性能計算研究室の皆様に心より深く感謝致します。

参考文献

[1] 梅原直人,古川達久,的場督永,山崎勝弘,小柳滋:ソフト・マクロ CPU を用いた回路設計とハードウェア/ソフトウェア最適分割法の検討,情報処理学会関西支部大会 VLSI システム研究会,C-07,2005.

[2] 梅原直人,古川達久,的場督永,山崎勝弘,小柳滋: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,FIT2005,C-034,2005.

[3] 古川 達久: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,立命館大学大学院理工学研究科,2005.3.

[4] 梅原直人,山崎勝弘:設計仕様の解析によるハードウェア/ソフトウェア最適分割手法の検討,情報処理学会関西支部大会 VLSI システム研究会,C-08,2006.

[5] 松井,小松,藤田:UMLとSpecCを用いたハードウェアの上位設計手法に関する検討,電子情報通信学会技術研究報告 Vol.104,No.708, pp.65-70, 2005.3.

[6] 西原,松本,小松,藤田:FSM への変換に基づく HW/SW 協調設計の形式的検証手法に関する研究,情報処理学会研究報告 Vol.2005, No.27,pp.37-42, 2005.3.

[7] 三菱電機株式会社:暗号技術仕様書 MISTY1,2001.

[8] Announcing the DVANCED ENCRYPTION STANDARD (AES), http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[9] 高橋征義,後藤裕蔵:たのしい Ruby,ソフトバンクパブリッシング,2002.

[10] オブジェクト指向スクリプト言語 Ruby 公式サイトチュートリアル,http://www.ruby-lang.org/ja/

[11] アヴネットジャパン:Xilinx MicroBlaze Development Workshop テキスト,2006.

[12] 西村啓成,石浦菜岐佐,石守祥之,神原弘之,冨山宏之:高位合成システム CCAP におけるハードウェア関数からのソフトウェア関数の呼び出し,情報処理学会関西支部大会 VLSI システム研究会,C-05,2006.

[13] 石守祥之,奥野裕,石浦菜岐佐,西村啓成,神原弘之,冨山宏之:C プログラムと中間表現の実行比較に基づく高位合成システム CCAP のテスト,情報処理学会関西支部大会 VLSI システム研究会,C-06,2006.

[14] 中田育男:コンパイラの構成と最適化,朝倉書店,1999.

[15] J.Axelsson: A hardware/software codesign methodology and workbench for predictable development of hard real-time systems,IEEE Micro Vol.17,pp.62-70,IEEE Real-Time Systems Proceedings,1997.

[16] P.Arato, S.Juhasz, Z.A.Mann, A.Orban, D.Papp:Hardware-software partitioning in embedded system design, IEEE Intelligent Signal Processing,pp.197-202, 2003.

[17] 和田智行:,立命館大学理工学部情報学科,卒業論文,2007.