C-019

ユビキタスプロセッサチップの開発

Development of a Ubiquitous Processor Chip

内海 晴信† 石原 拓美† 三村 直道† 高木 竜哉† 成田 一貴†
Harunobu Uchiumi Takumi Ishihara Naomichi Mimura Tatsuya Takaki Kazuki Narita

深瀬 政秋† 佐藤 友暁‡
Masa-aki Fukase Tomoaki sato

† 弘前大学大学院理工学研究科理工学専攻電子情報工学コース

‡ 弘前大学総合情報処理センター

1. まえがき

近年のユビキタスネットワークの急速な発展に伴い、情報セキュリティのリスクが増大している。そのため信頼性の高い情報セキュリティ対策が重要となってくる。一方、限られた電力を有効に使用するために省電力高速化が求められている。モバイル機器の省電力化は、バッテリ消耗の緩和のみならず昨今の環境保護からグリーン IT への傾向により益々重要な課題となった。このため、ディスプレイとプロセッサの電力消費の抑制が必要である。両者の消費電力は似たようなレベルであるが、ディスプレイは必要な時だけ稼働させるのに対して、プロセッサは常時待受状態でディスプレイの制御もカバーする。モバイル機器の省電力化はプロセッサが負わされている。クロックスピードの追求から並列化による高性能省電力の追求でマルチコア化にシフトした PC プロセッサの流れは、モバイル系のプロセッサにも拡がっている。

以上のような背景にマッチするプロセッサとして、我々はユビキタスプロセッサ HCgorilla の設計とチップ開発を進めてきた[1]。しかし、いくつかの課題も見えてきた。まず、HCgorilla はハードウェア暗号組込み型で、暗号方式はランダムアドレッシングによる転置暗号である。この方式は SIMD 命令によるストリーム暗号であり、高効率であるが、暗号強度が必ずしも十分ではなかった。次に、HCgorilla は PC プロセッサにおける並列化の傾向をいち早く踏襲したが、省電力化はアーキテクチャレベルのみならず、プログラムレベル、セルレベル、トランジスタレベルと多重化することで、より効果を発揮する。実際、これまで開発した HCgorilla でも、マイクロアーキテクチャレベルの高速省電力化策としてウェーブ化を併用している。しかし、CAD ツールは通常型パイプラインの設計に特化されていることから、HCgorilla のウェーブ化はマニアルチューニングの対応がなされ、ウェーブ化の実践は必ずしも十分ではなかった。

本研究では、暗号強度の更なる強化を目指してハードウェア乱数発生器（random number generator, RNG）を 2 つに増やし、2 重暗号方式を採用する。また、多重の省電力化を一層推進するために、ゲーテッドクロックも採用する。但し、ゲーテッドクロックはマイクロアーキテクチャレベルの省電力化の常套手段として通常型パイプラインに対しては確立しているが、ウェーブ化がなされている HCgorilla に直接導入することはできない。クロックスキームに関わるゲーテッドクロックとウェーブ化の融合が必要である。本論文では以上のような課題に取組み、改良を施した HCgorilla のチップ設計と基本的な評価について述べる。

2. プロセッサアーキテクチャ

HCgorilla は、ユビキタス機能の実現と性能追求の両面から図 1 に示すような並列性を備えている。HCgorilla は双方向通信のために 2 つの独立した CORE を持ち、各 CORE はサイファーパイプと Java 対応のメディアパイプで構成される。サイファーパイプは 1 個の RNG を有する。これらのパイプラインと独立して、レジスタファイルとデータキャッシュを備える。CORE1 側の領域は、レジスタファイルとデータキャッシュのそれぞれ下位バイトに割り当てる。同様に、CORE2 側はそれぞれの上位バイトに対応させる。

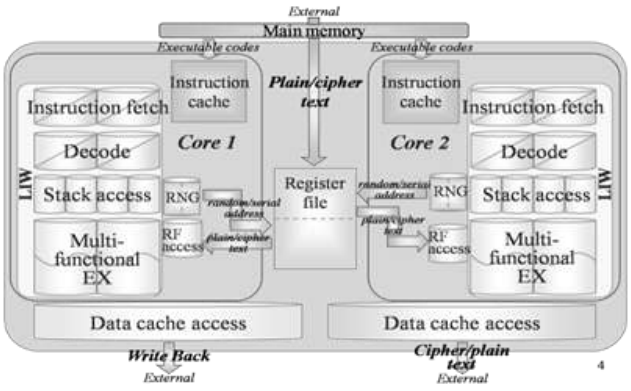


図 1 従来の HCgorilla のアーキテクチャ

以上のような基本構成を基に、本研究ではサイファーパイプに対して図 2 に示すようなマイクロアーキテクチャレベルの機構を採用する。RNG には LFSR (linear feedback shift register)を用いて、一本のサイファーパイプに 2 つ用意する。これにより、従来の転置暗号と平文そのものに対する換字暗号を組み合わせて 2 重暗号方式をサイファーパイプに実装する。

本研究は東京大学大規模集積システム設計教育研究センターを通し、シノプシス社の協力で行われたものである。

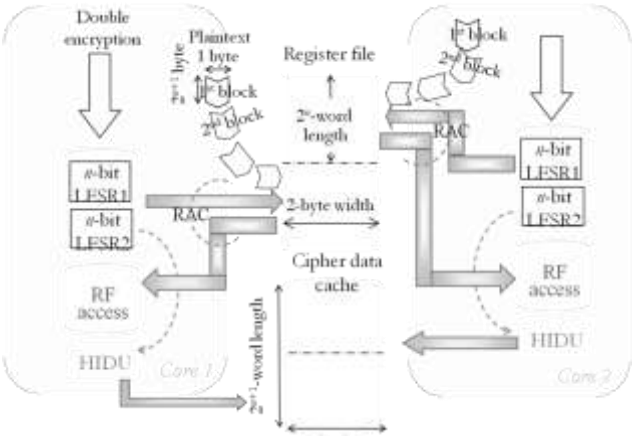


図 2　2 重暗号方式

省電力高性能化のため、メディアパイプの実行段のウェーブ化に加えて、メディアパイプの実行段とスタックにゲーテッドクロックを採用する。ウェーブパイプラインとは回路の最小遅延を最大遅延に近づけることで高速化する [2]。遅延の少ないパスにバッファを挿入していくことで、パス全体の遅延を大きくしていく。従来パイプラインは 1 クロックあたりの時間を最大遅延より遅らせなければいけないが、最大遅延と次の命令の最小遅延がぶつからなければ問題なく動作することができるため、ウェーブパイプラインは 1 クロックあたりの時間を最大遅延より速くすることができる。このことから従来パイプラインのクロックスピードより高速に動作することが可能となる。またパイプラインレジスタを取り除くことで省電力化も期待される。但し、CAD ツールは通常型パイプラインの設計に特化されていることから、HCgorilla の実行段のウェーブ化は、マニアルチューニングの対応がなされてきた。

ゲーテッドクロックはマイクロアーキテクチャレベルの省電力化の常套手段で、意味のある動作をしていない部分にクロックの供給を止めてダイナミック電力を消費しないようにするものであるが、この確立した技術を HCgorilla に直接導入することはできない。何故なら、HCgorilla はウェーブ化がなされているためである。CAD ツールがサポートするゲーテッドクロックは、HCgorilla には適用できない。本研究では、ゲーテッドクロックとウェーブ化のクロックスキームを融合し、使われていないパイプラインには電力を供給しないものとする。命令は Decode から各パイプラインに送られるため、Decode の信号とクロックを制御することにより、使われないパイプラインにクロックを供給しない。全てのパイプラインはクロック同期なためクロックがなくなれば動作を全くしないことになり、省電力化することができる。

本研究ではさらに、テスト容易化設計としてスキャン回路を導入する。これにより、チップ化した際にパイプライン毎にテストをすることができる。動作不具合が発生した際に、どこのパイプラインに問題があるか確認することが可能となる。

3. チップ設計

HCgorilla の開発環境を表 1 に示す。論理合成は、Synopsys 社の Design Compiler、動作検証には Synopsys 社の VCS を用いる。配置配線には Synopsys 社の IC Compiler で設計し、各種検証には Mentor 社の Calibre を用いている。ライブラリは、京都大学より提供されている ROHM 0.18-μm CMOS Standard Cell Library を使用する。但し、このライブラリは本研究で試作する 5.0mm×7.5mm のチップサイズには対応していないため、自作で 5.0mm×7.5mm 用の環境を構築した。

表 1　チップ開発環境

Software	
OS	Red Hat Linux 4/SentOS 5.4
Synthesis tool	Synopsys - Design Compiler D-2010.03
Simulation tool	Synopsys - VCS version Y-2006.06-SP1
Physical Implementation tool	Synopsys - IC Compiler C-2009.06
verification tool	Mentor - Calibre v2010.02_13.12
Equivalent verification tool	Synopsys – Formality B-2008.09-SP5
Static Timing analysis tool	Synopsys – Primetime pts,vA-2007.12-SP3
Simulation tool	Synopsys - hspice simif D-2010.03 or hsim_vD-2010.03
Language	
Synthesis	VHDL
Simulation	Verilog - HDL
Technology	
Kyotouniv ROHM 0.18μm CMOS Standard Cell Library	

チップ試作開発の設計フローを図 3 に示す。チップ開発にはデジタル回路を設計していくデジタル設計とデジタル設計で作成した回路を実際のチップ上に配置していくレイアウト設計の大きく 2 つに分かれる。デジタル設計ではサイファーパイプ側における回路設計の詳細について述べ、レイアウト設計ではサイファーパイプとメディアパイプを統合したものをチップ上に配置していき、最終的にチップデータを提出する。

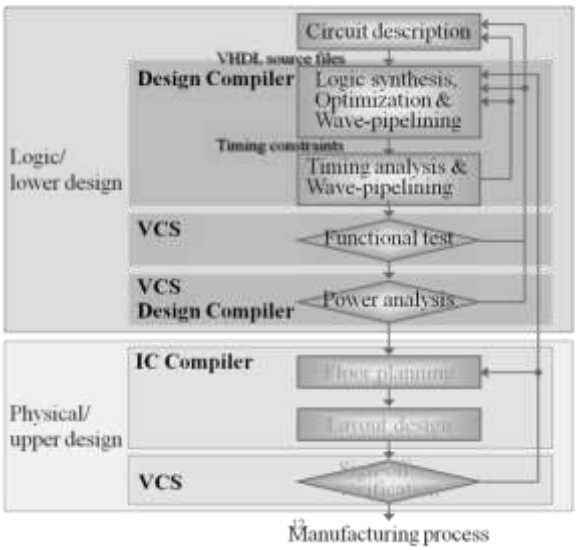
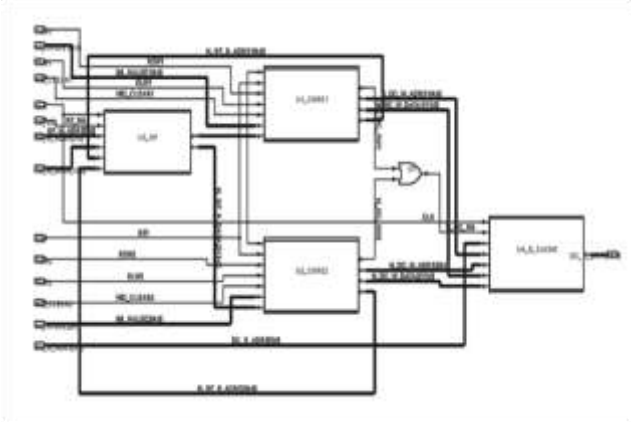


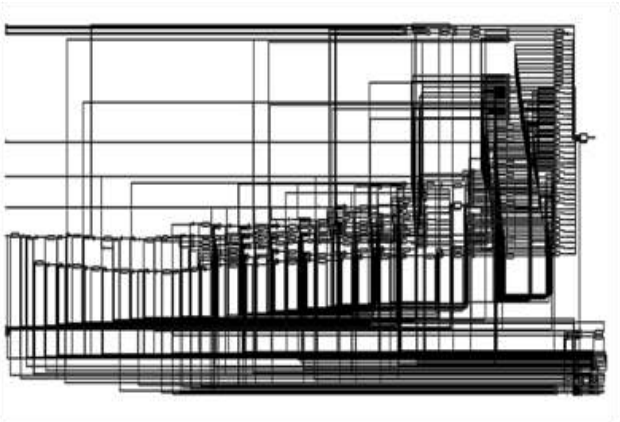
図 3　設計フロー

3.1　デジタル設計

サイファーパイプの場合について説明する。デジタル設計では求められている仕様を実現するために VHDL 言語で機能記述や階層構造記述をした後、論理合成していく。従来の HCgorilla はデジタル設計段階で階層構造を考慮してトップレベルまで論理合成し、それをレイアウト設計していたが、本研究では各ユニットをデジタル設計したものをレイアウト設計でトップレベルまで合成するため、各ユニットで論理合成しレイアウト設計をしていく。また従来は各ユニットをグループ化してまとめていたが、本研究ではグループ化してしまうとレイアウト設計時に入出力の関係で不具合が起こってしまう可能性があるのでグループ化をせずに各セルが見えている状態で論理合成していく。図 4 に論理合成を示す。従来はユニット間で配線が行われているが本研究ではセル間で配線が行われている。論理合成後の動作検証結果を図 5 に示す。各パイプラインの結果から各ステージの結果が正しく出ており、また Register File と Data Cache の入出力データから暗号化と復号化が正しく動作していることがわかる。



(a)



(b)

図 4　論理合成　(a)従来　(b)本研究

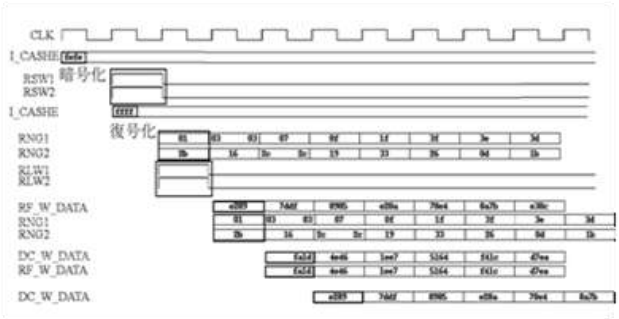


図 5　シミュレーション結果

3.2　レイアウト設計

従来は図 6 に示すようにフラットなレイアウトを採用していたが、配線遅延が増える可能性があった。本研究では、図 7 に示すようにフロアプランを施した。マクロとしてユニットがまとまっているので、配線遅延も小さくなると考えられる。

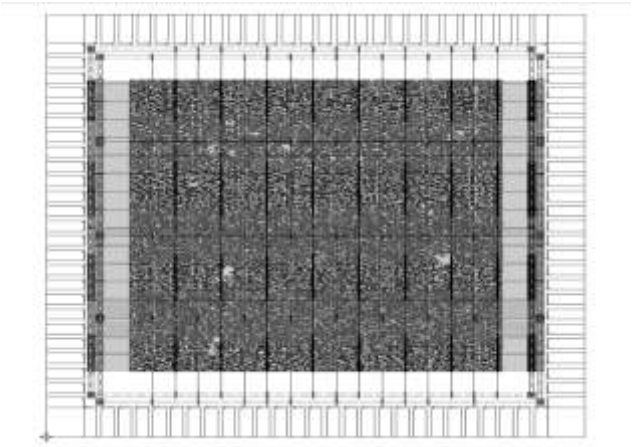


図 6　従来のチップレイアウト手法

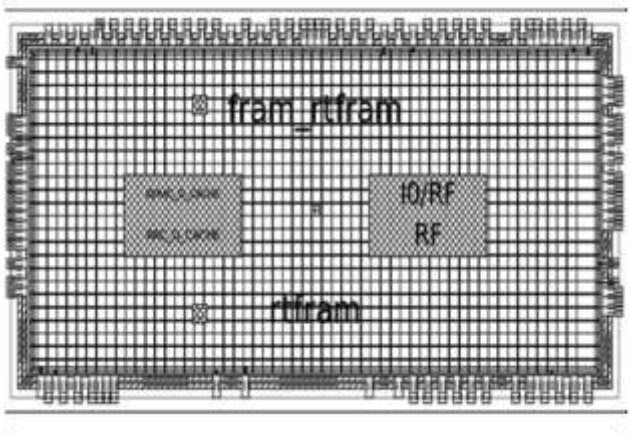


図 7　本チップレイアウト手法

レイアウト設計には、それぞれの機能を実現するマクロを作成していくマクロレベル設計と、そのマクロを実際のチップに配置していくチップレベル設計の大きく 2 つに分かれる。しかし、構造や機能が複雑化している中でより分かりやすくまとめるために、マクロレベル設計で機能を実現するマクロと階層構造を実現するマクロの 2 パターンを作成していく。階層構造を実現するマクロの用途として、CORE という大きなマクロの中に各パイプラインのマクロを配置して CORE として機能するマクロを作成していった。CORE は図 1 のような従来のものとは別で、本研究ではメディアパイプとサイファーパイプのそれぞれで CORE をもっており、メディアパイプの CORE でこの手法を採用している。これらの 3 種類の設計について以下に述べる。

3.2.1 マクロレベル機能設計

デジタル設計で生成した回路情報を記載している verilog ファイルや面積情報を基に、マクロを作成する。まず始めに各ユニットの入出力とマクロの入出力を同期するためにピン位置ファイルを作成し、デジタル設計時の回路面積や詰め込み具合を示す充填率を決め、最終的なマクロの形を決めていく。その後、CAD ツールの IC Compiler でマクロの中に回路を配置配線していく。図 8 に示すのは、通常クロックを入力としてゲーテッドクロックを出力する回路のマクロである。図 9 は、ウェーブ化実行段のマクロである。

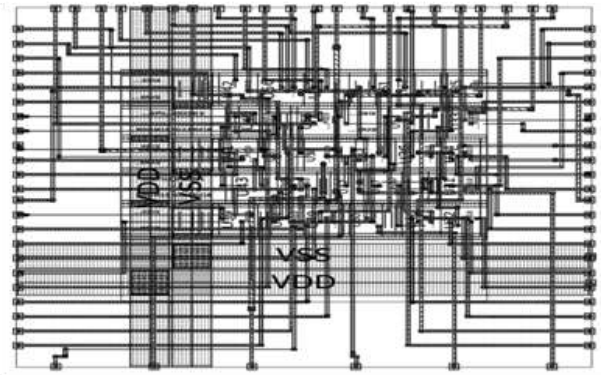


図 8　ゲーテッドクロック回路のマクロ

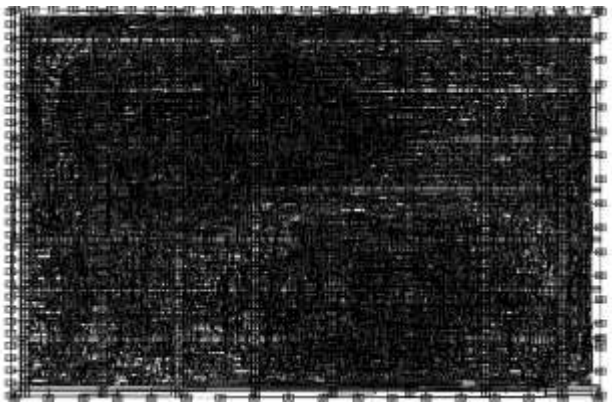


図 9　ウェーブ化実行段のマクロ

マクロレベル機能設計の検証では、DRC (Design Rule Check), LVS (Logic Versus Schematic), アンテナルールチェック, グリッドチェック, 等価性検証を行う。セルの規模が大きくなると、エラーが出やすくなるが、ピン配置を変更したり、充填率を小さくすることにより、セルを詰めすぎないようにしてエラーが出ないように注意しながら、すべての機能マクロを作成する。各種検証でエラーがでなかった場合、この段階でのチップデータ提出には問題がないため、次の工程である全ての機能マクロを 1 つのマクロに集結する構造マクロを設計していく。

3.2.2 マクロレベル構造設計

構造マクロは、機能マクロ設計とチップレベル設計の両方の特徴を持っている。まず始めにマクロレベル機能設計と同様にピン位置ファイルを作成する。次にマクロの形を決めていくが、構造マクロ設計ではマクロの形を自分で決めていかなければいけない。全ての機能マクロを 1 つのマクロにいれることができ、かつマクロ間の配線などの面積も考慮しながら大きさを決めなければいけないため、余裕のある大きさにしていく。さらに各機能マクロをどこに置くか決めるマクロ配置位置ファイルを作成していく。その後、CAD ツールの IC Compiler でマクロの中にマクロを配置配線していく。メディアパイプとサーファーパイプのフロアプランを、それぞれ図 10、図 11 に示す。

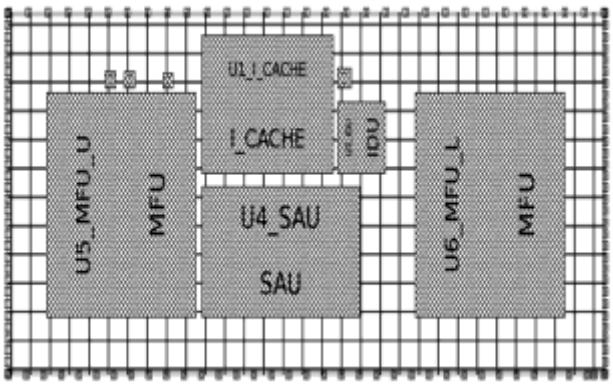


図 10　メディアパイプ

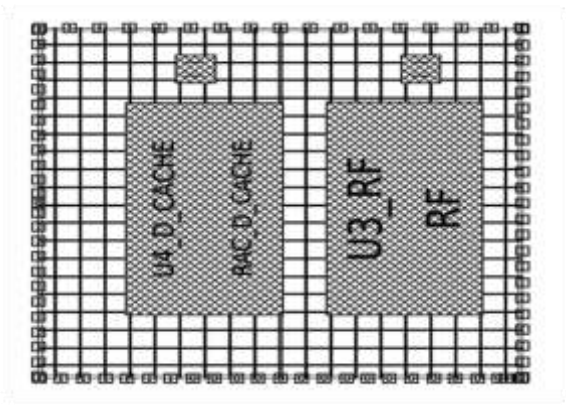


図 11　サイファーパイプ

マクロレベル構造設計の検証は、マクロレベル機能設計の場合と同様である。マクロの配置位置によって配線が密になりすぎるなどのエラーも出てくるので、マクロ間の関係を考えながらマクロ配置位置を決定していく必要がある。

3.2.3 チップレベル設計

この工程は、構造マクロを実チップに配置する。チップレベル設計ではチップ面積が重要になってくる。試作チップの 5.0mm×7.5mm 用のスクリプトを自作する。チップの最大の縦幅、横幅、IO ピンの位置、読み取るフレーム名などを変更し、同じように電源リング、電源ストラップのスクリプトも変更し、設計環境を構築してからチップレベル設計を行っていく。始めにトップレベル構造記述ファイルに IO セルを付加し、それに伴い IO ピンの位置を指定するファイルも作成する。チップレベルの入出力はチップにより電源、グラウンド専用 IO ピンの位置が決まっているため、注意しながらピン配置をしなければいけない。次に各構造マクロをどこに置くか決めるマクロ配置位置ファイルを作成していく。その後、CAD ツールの IC Compiler でチップの中にマクロを配置配線していく。最終的なチップレイアウトを図 12 に示す。パッケージの概観を図 13 に示す。

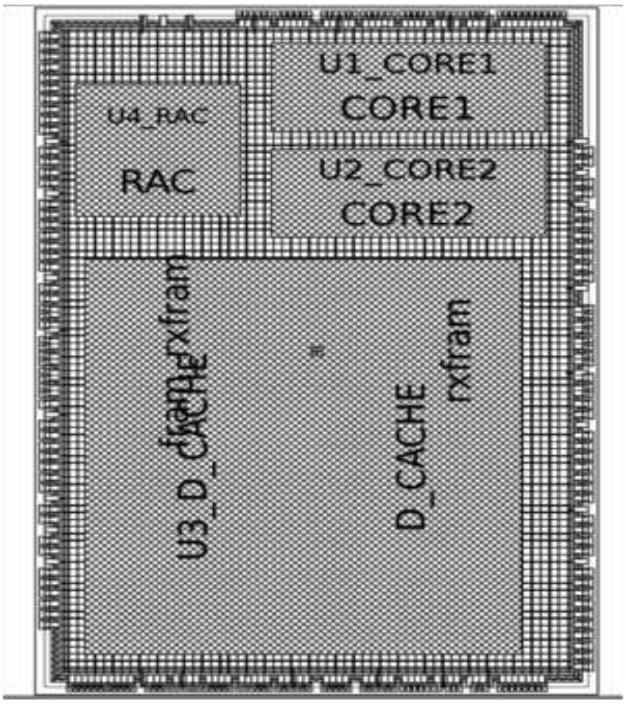


図 12　チップレイアウト

図 13　パッケージ外観

4. HCgorilla の評価

デジタル設計段階のサイファーパイプについて、面積、消費電力、実行時間、スループット、暗号強度の RNG 数依存性を評価する。RNG が 1 個の場合は従来 HCgorilla の転置暗号方式で、2 個の場合が本研究の 2 重暗号方式である。面積、消費電力に関しては RNG1 つあたりのセル数は 100 以下であり HCgorilla 全体でのセル数は 10 万以上なので 1 つ増設しただけでは、変化はないと考えてよい。また実行時間、スループットはクロック周波数とパイプライン処理数から導出され、それらに変化は全くないので、実行時間、スループットも変化はない。面積、消費電力の比較を図 14 に示し、実行時間、スループットの比較を図 15 に示す。

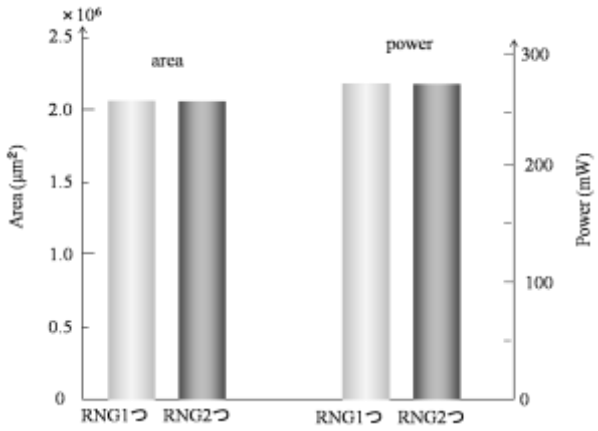


図 14　面積、消費電力の RNG 数依存性

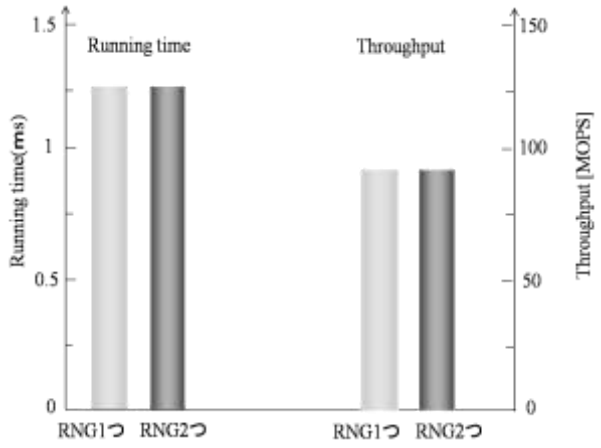


図 15　実行時間、スループットの RNG 数依存性

暗号強度は、図 16 に示す方法で測定する。RNG が 2 つの場合二重ループになっており、両方が正しくないと、正しく解読出来ない。RNG が 1 つの場合ループが 1 つなので、容易に解読できる。レジスタファイルの各サイズでの暗号強度を図 17 に示す。RAC/HIDU は RNG が 1 つの転値暗号で、Double cipher は RNG が 2 つの 2 重暗号である。試作した HCgorilla はレジスタファイルが 128word である。レジスタファイルサイズを増加させていくと暗号強度も増加している。また、RNG の増設により暗号強度は増大する。

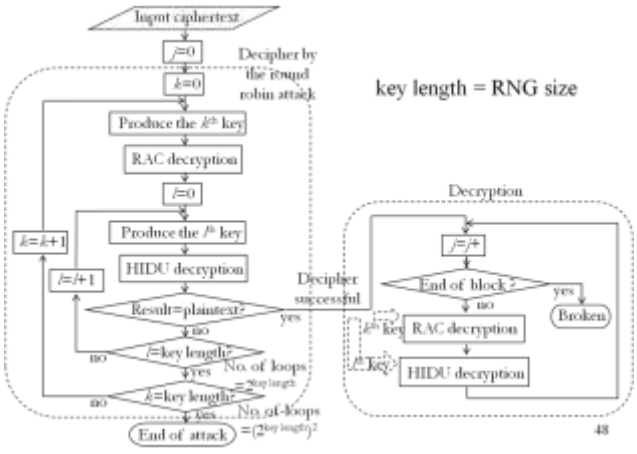


図 16　暗号強度の測定方法

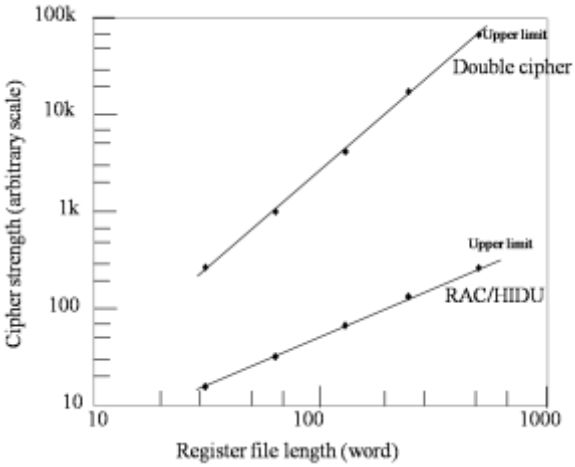


図 17　暗号強度のレジスタファイル長依存性

5. まとめ

HCgorilla のユビキタス仕様からチップ完成までの一連の流れと、試作チップの基本的な評価について述べた。通常の設計工程に独自の技術を加味した。省電力高速化のためのウェーブ化、ゲーテッドクロックと、高機能設計とテストのためのスキャン回路はいずれもクロックスキームに関わる。これらの融合のため、レイアウト設計をマクロ化することにより個々のユニットをかためて配置することにした。サイファーパイプに RNG を増設することにより、暗号強度を強化した。この改良は、消費電力やスループット、処理時間に影響しないことを確認した。

今後の課題としては、

・レイアウト後の電力評価：ゲーテッドクロックの効果、レイアウト設計でマクロ化したことによる配線長の減少効果。
・実チップの妥当性確認：スキャン回路を併用する。

6. 参考文献

[1] M. Fukase, H. Uchiumi, and T. Sato, “Cipher and Media Possibility of a Ubiquitous Processor,” Proc. of ISCIT’09, pp.343-347, Sep 2009.

[2] W. P. Burleson, M. Ciesielski, F. Klass, and W. Liu, “Wave-Pipelining: A Tutorial and Research Survey,” IEEE Trans. on Very Large Scale Integration (VLSI) Systems, Vol. 6, No. 3, pp. 464-474, Sept. 1998.