情報処理学会論文誌　Vol. 50　No. 11　2618–2632 (Nov. 2009)

ハッシュ関数 Whirlpool の高スケーラブル回路アーキテクチャ

菅原　健[†1]　本間尚文[†1]
青木孝文[†1]　佐藤証[†2]

512 ビットハッシュ関数 Whirlpool の ASIC 実装向け高スケーラブル回路アーキテクチャを提案する．提案するアーキテクチャは，データパスのループ構造とワード長を選択することで，要求性能に応じた多様な性能を実現できる．本稿では，まず，Whirlpool の回路アーキテクチャについて，データパスのループ構造およびワード長の両面より検討する．その後，回路アーキテクチャに応じた演算コンポーネントの効率的な設計法について述べる．さらに，提案アーキテクチャの具体的な実装例として，性能のトレードオフに応じた 9 種類のデータパスを示し，90 nm CMOS 標準セルライブラリにより性能評価を行う．この結果，スループット，回路面積，および回路効率の性能指標において高いスケーラビリティと，優れたピーク性能（最小の回路規模は 13.6 Kgates，最大スループット 28.0 Gbps，および最大の回路効率 372.3 Kbps/gate）が得られることを示すとともに，従来回路や他のハッシュアルゴリズムとの性能比較を行う．

High Scalable Circuit Architectures of the Hash Function Whirlpool

Takeshi Sugawara,[†1] Naofumi Homma,[†1]
Takafumi Aoki[†1] and Akashi Satoh[†2]

High scalable circuit architectures for the 512-bit hash function Whirlpool are presented. The proposed architectures enable a variety of performances in accordance with a design requirement. In this paper, the efficient circuit architectures are discussed from the view point of loop structure and word length, followed by implementation methods of each circuit components. Total nine datapaths are implemented as examples of the proposed architectures and evaluated using 90-nm CMOS standard-cell library. The results showed that the proposed architectures can acheive high scalability as well as significant peak performances in each performance indices (the smallest circuit area of 13.6 Kgates, the highest throughput of 28.0 Gbps, and the highest efficiency of 372.3 Kbps/gate). The performances are also compared with conventional Whirlpool circuits and SHA-256/512 circuits.

1. はじめに

ハッシュ関数は，データの正真性の保証や，ディジタル署名をはじめとする様々な暗号プロトコルで利用される．応用例は広範に及ぶが，特に，インターネットの認証局（CA）のように高スループットを要する場合や，スマートカードや無線タグのように演算リソースが乏しい場合には，専用 LSI（ASIC）による実装が行われる．このように，応用によって求められる性能は異なる．そのため，ハッシュ関数の ASIC 実装では，速度や回路面積のピーク性能が優れていることに加え，性能要求に応じてスケーラブルな回路が設計できることが求められる．

ハッシュ関数 Whirlpool[1),2)] は NESSIE（New European Schemes for Signatures, Integrity and Encryption）プロジェクト[3)] に提案され，その後 ISO/IEC 10118-3 標準アルゴリズム[4)] として標準化された．Whirlpool は，これまで広く使われてきたハッシュ関数の MD5[5)] や RIPEMD-160[6)]，SHS（SHA-1，SHA-224/-256/-384/-512)[7),8)] と，構造が大きく異なる．MD5，RIPEMD-160，および SHS では，シフトレジスタと簡単なロジック（加減算，ビット演算，巡回シフト等）でメッセージを拡大した後，所定のビットに圧縮する方式が採用されていた．このような方式は，ソフトウェア実装において高い性能を発揮する一方，ハードウェア実装において性能のスケーラビリティが低い．これに対し Whirlpool は，Miyaguchi-Preneel 型のハッシュ関数であり，圧縮関数[9)] にブロック暗号 AES[10)] に似た 512 ビット関数 W を用いる．そのため，AES と同様に，ハードウェア実装において特に高いスケーラビリティが期待できる．しかし，演算単位が 512 ビットと，AES の 4 倍であり，単純な実装手法では非常に大きな回路リソースが必要で，動作周波数の低下も懸念される．そのため，効率的な実装方法の検討が必要である．

†1 東北大学大学院情報科学研究科
Graduate School of Information Sciences, Tohoku University
†2 独立行政法人産業技術総合研究所情報セキュリティ研究センター
National Institute of Advanced Industrial Science and Technology, Research Center for Information Security

Whirlpool の回路実装はこれまでにたくさんの先行研究がある[11)–18)]．しかし，筆者らによる ASIC 実装[17),18)] を除き，すべて FPGA を対象としたものである．FPGA 実装では，その特殊な回路構造を反映した設計が行われるため，そのまま ASIC に実装しても性能が改善されるとは限らない．また，前述のように高いスケーラビリティが必要であるとの立場からは，個々の優れた設計例だけではなく，回路アーキテクチャレベルでの検討が求められる．

これに対し，本稿では，ハッシュ関数 Whirlpool について，ASIC 向けに高いスケーラビリティを有する回路アーキテクチャを提案する．本稿は以下のように構成する．2 章では，Whirlpool のアルゴリズムについて述べる．続く 3 章では，Whirlpool の回路アーキテクチャの設計について，ループ構造とワード長の面から検討する．また，回路アーキテクチャに応じた演算コンポーネントの効率的な設計法について 4 章で述べる．5 章では，提案アーキテクチャの具体的な実装例として，性能のトレードオフに応じた 9 種類のデータパスを示す．6 章では，設計したデータパスの性能評価を，90 nm CMOS スタンダード・セルライブラリを用いて行う．この結果，提案手法により多様な性能が実現可能であることを示す．また，これらの性能を SHA-256/512 回路[19)] や FPGA 実装と性能比較し，ピーク性能においても優れた値が得られることを示す．7 章はまとめである．

2. Whirlpool のアルゴリズム

Whirlpool のアルゴリズムの概要を**図 1** に示す．ハッシュ値の計算においては，入力の 512 ビットメッセージブロック m_i に対して，次の式に従い圧縮を行う．

$$H_i = m_i \oplus H_{i-1} \oplus W[H_{i-1}](m_i) \quad (1)$$

$$H_0 = 0 \quad (2)$$

このとき，H_i はメッセージ $m_1 \cdots m_i$ に対応するハッシュ値，$W[H_{i-1}](m_i)$ は圧縮関数である．圧縮関数 W は，図 1 のように，2 入力 1 出力の 512 ビット関数 $\rho[k]$ が 10 個連なったパスを 2 つ有する．一方は，計算済みのハッシュ値 H_{i-1} と 512 ビット定数 $c^1 \cdots c^{10}$ からラウンド鍵 $K^1 \cdots K^{10}$ を生成するパスである．また，もう一方は，生成した $K^1 \cdots K^{10}$ を用いて現在のメッセージブロック m_i を攪拌するパスである．関数 $\rho[k]$ は 4 つの 512 ビット関数 γ，π，θ および $\sigma[k]$ で構成される．各関数は 512 ビットブロックをバイト（8 ビット）単位で区切った 8 × 8 バイト行列への操作という形で処理を行う．

関数 γ は 8 ビット入出力のテーブル（S-box）による非線形変換である．S-box の幅は 8 ビットであるため，512 ビット幅の実現には 64 個の S-box が必要である．Whirlpool の S-box は，**図 2** に示すような内部構造を有しており，より小さなコンポーネントの組合せ

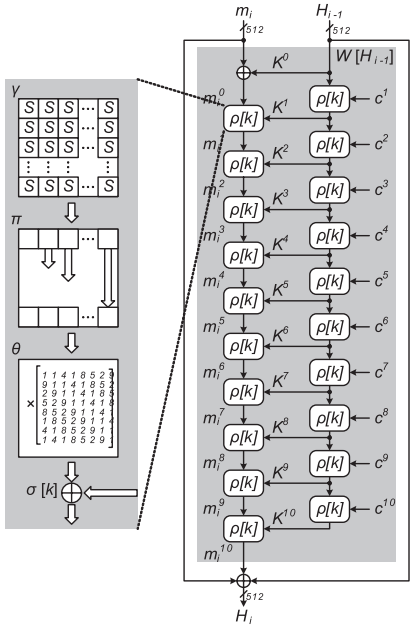

図 1　Whirlpool のアルゴリズム
Fig. 1　Whirlpool algorithm.

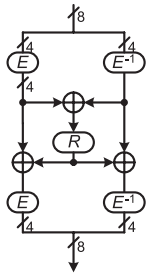

図 2　S-box の内部構造
Fig. 2　Inner structure of S-box.

$$\begin{bmatrix} 0 & 8 & 16 & 24 & 32 & 40 & 48 & 56 \\ 1 & 9 & 17 & 25 & 33 & 41 & 49 & 57 \\ 2 & 10 & 18 & 26 & 34 & 42 & 50 & 58 \\ 3 & 11 & 19 & 27 & 35 & 43 & 51 & 59 \\ 4 & 12 & 20 & 28 & 36 & 44 & 52 & 60 \\ 5 & 13 & 21 & 29 & 37 & 45 & 53 & 61 \\ 6 & 14 & 22 & 30 & 38 & 46 & 54 & 62 \\ 7 & 15 & 23 & 31 & 39 & 47 & 55 & 63 \end{bmatrix} \xrightarrow{\pi} \begin{bmatrix} 0 & 15 & 22 & 29 & 36 & 43 & 50 & 57 \\ 1 & 8 & 23 & 30 & 37 & 44 & 51 & 58 \\ 2 & 9 & 16 & 31 & 38 & 45 & 52 & 59 \\ 3 & 10 & 17 & 24 & 39 & 46 & 53 & 60 \\ 4 & 11 & 18 & 25 & 32 & 47 & 54 & 61 \\ 5 & 12 & 19 & 26 & 33 & 40 & 55 & 62 \\ 6 & 13 & 20 & 27 & 34 & 41 & 48 & 63 \\ 7 & 14 & 21 & 28 & 35 & 42 & 49 & 56 \end{bmatrix}$$

図 3　関数 π の入出力関係
Fig. 3　Example I/O of the function π.

で表現できる．図において，E，E^{-1}，R は 4 ビット入出力のテーブル（mini-table）である．これら mini-table を，XOR を介して組み合わせることで，S-box と等価な入出力関数を作成することができる．関数 π は，行列を列方向に $0 \cdots 7$ バイト巡回シフトさせる操作である．これは，AES の *ShiftRows* に対応するが，シフト方向が行ではなく列方向である点が異なる．関数 π の入出力関係の例を**図 3** に示す．行列 r 列目について下向きにシフト量 $r-1$ で要素の巡回シフトが行われる．関数 θ では，図 1 に示す定数行列との乗算を行う．このとき，行列の要素を拡大体 $GF(2^8)$ 上の要素と見なして計算する．そのため，加法，乗法は $GF(2^8)$ 上の演算規則に従う．8×8 バイト入力を a_{ij}（$0 < i,\ j < 7$），出力を b_{ij} とすると，例として列 $j = 0$ の 8 バイト出力 b_{i0} は次式で表される．

$$b_{i0} = a_{i0} \oplus (9 \otimes a_{i1}) \oplus (2 \otimes a_{i2}) \oplus (5 \otimes a_{i3}) \\ \oplus (8 \otimes a_{i4}) \oplus a_{i5} \oplus (4 \otimes a_{i6}) \oplus a_{i7} \quad (3)$$

$GF(2^8)$ 上の演算規則であることを示すために，加算と乗法を，$\oplus$ および $\otimes$ で示した．なお，$GF(2^8)$ 上の乗算では，次の規約多項式 $p_8(x)$ を使用する．

$$p_8(x) = x^8 + x^4 + x^3 + x^2 + 1 \quad (4)$$

最後の関数 $\sigma[k]$ は，鍵 k の鍵加算である．k の値は，鍵スケジュールのパスでは定数 $c^1 \cdots c^{10}$，データ撹拌のパスではラウンド鍵 $K^1 \cdots K^{10}$ である．

3. 回路アーキテクチャ

本章では，提案する Whirlpool の回路アーキテクチャについて述べる．なお，回路アーキテクチャとは，レジスタと演算コンポーネントの接続関係のことを表すものとする．提案アーキテクチャは，(i) ループ構造と，(ii) ワード長の 2 点を選択することで，性能のスケーラビリティを実現する．以降では，まず，(i) と (ii) の 2 点について分類を行い，採用する方式に応じて，性能が変化することを示す．その後，提案手法の利点および，従来法との差異について述べる．

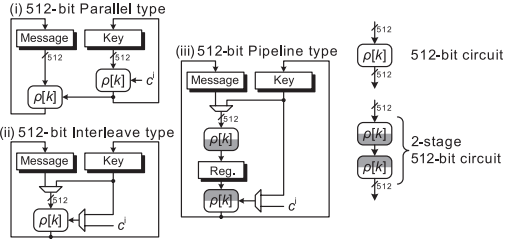

図 4　3 種類のループ構造
Fig. 4　Three types of loop structures.

3.1 ループ構造

回路実装における最も単純な実装手法は，図 1 に示したアルゴリズム仕様を，そのまま組合せ回路として実現し，1 サイクルで処理するものである．しかし，このような実装方式にはきわめて大きな回路リソースが必要となるため実用的ではない．そのため，アルゴリズムから繰返し構造（ループ構造）を抽出して，複数サイクルかけて処理を行うのが一般的である．この場合，1 度のループにおいて実行する処理単位を Whirlpool のアルゴリズムから抽出する必要がある．Whirlpool は，図 1 に示すように，関数 $\rho[k]$ を処理単位とするため，これをループにおける処理単位とするのが妥当である．

2 章で示したように，関数 $\rho[k]$ は，鍵スケジュールとデータ撹拌の両方で使用される．この 2 つのパスをどのように実現するかによってアーキテクチャを分類できる．具体的には，**図 4** に示すような，(i) パラレル型，(ii) インタリーブ型，(iii) パイプライン型の 3 種類のループ構造を考えることができる．パラレル型では，データ撹拌と鍵スケジュールにそれぞれ独立な関数 $\rho[k]$ の演算器を割り当てる．これに対し，インタリーブ型では，関数 $\rho[k]$ の演算器を 1 つだけ用意し，これをデータ撹拌と鍵スケジュールで交互に使用する．最後のパイプライン型では，演算器を 2 つに分割し，その間にレジスタを挿入することでパイプラインを構成する．

各ループ構造の性能を比較する．パラレル型は，少ないサイクル数での処理が可能なため処理速度の面で有利な半面，演算器が 2 つ必要なことから回路面積は大きい．インタリーブ型は，パラレル型と比較して 2 倍のサイクル数が必要となる．しかし，関数 $\rho[k]$ の演算器は 1 つのみで構成できるため，回路面積が小型である．パイプライン型は，インタリーブ型と同等のサイクル数であり，インタリーブ型にパイプラインレジスタを加えた程度の面積となる．ただし，関数 $\rho[k]$ の演算器を分割したことで動作周波数が向上し，その結果として

インタリーブ型よりも高い処理速度が得られる．したがって，パイプライン型は，回路効率（面積あたりのスループット）の面で他のアーキテクチャよりも有力であると考えられる．

上述したループ構造は，データ攪拌と同時に鍵スケジュールを行う方式であり，オンザフライ方式と呼ばれる．これに対し，データ攪拌前にすべてのラウンド鍵を事前計算することもできる．これは，インタリーブ型において演算シーケンスを変更したものと解釈できる．FPGA 実装ではこのような実装が行われることが多い[11),13),14),16)]．しかし Whirlpool では，直前のハッシュ値 H_{i-1} が，次のハッシュ値 H_i の計算にフィードバックされるため，512 ビットメッセージ m_i の入力ごとに鍵スケジュールをやり直す必要がある．そのため，事前計算により計算量が削減されることはなく，両方式のスループットは同等である．これに対し，事前計算方式ではラウンド鍵の格納に 640 バイト（512 ビット× 10 ラウンド）のメモリ容量が追加で必要となる．このような理由から，本稿では事前計算を利用する構造を検討していない．

3.2　ワード長の分割

3.1 節では，関数 $\rho[k]$ を演算単位としてループ構造を構成する方法について述べた．しかし，関数 $\rho[k]$ の内部も繰返し構造を有しており，さらに分割することが可能である．以降では，ループ 1 巡（1 クロックサイクル）で処理するデータ長をワード長と呼ぶ．関数 $\rho[k]$ の分割をまったく行わない場合，ワード長は 512 ビットとなる．これに対し，ワード長を 256，128，64 ビットと分割し，サイクル数を増やして処理すれば，面積を小型化できる．これに対応し，図 4 に示すループ構造を，様々なワード長に拡張できる．

Whirlpool を構成するサブ関数は行列への操作によって定義されているため，分割方法には，行もしくは列方向の 2 通りが考えられる．関数 γ および $\sigma[k]$ は，バイトおよびビット単位で独立な処理であるため，いずれの方向にも分割できる．これに対し，関数 π，θ はそれぞれ行，列単位にのみ独立である．そのため，行方向に分割するのであれば関数 π が，列方向に分割するのであれば関数 θ がそれぞれ依存性を満たせなくなる．そのため，ワード長を 256，128，64 と変化させるには，π か θ いずれかの関数の依存性を解決する機構を導入する必要がある．本稿では，文献 14) と同様に，行方向の分割を採用する．文献 14) では，ワード長 64 ビットのアーキテクチャにおいて，Data manager と呼ばれる演算コンポーネントを使用することで，関数 π の依存性を解決していた．本稿では，Data manager を 128 および 256 ビットに拡張することで，ワード長 128 および 256 ビットのアーキテクチャにおいても，効率的に依存性を解決する手法を提案する．

データパスの分割は，3.1 節で示したいずれのループ構造にも適用可能である．**図 5** に，

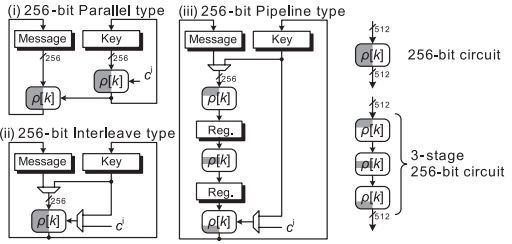

図 5　ワード長 256 ビットのループ構造
Fig. 5　Loop structures with word length of 256 bits.

表 1　ワード長と回路性能（理想値）
Table 1　Ideal performances over word lengths.

Datapath width	Loop architecture	Circuit area	Critical path	Clock cycle	Operating time	Efficiency (Circuit area × Operating time)
512	Interleave	A	D	C	D×C	A×D×C
512/N	Parallel	2×A/N	D	C×N/2	D×C×N/2	A×D×C
512/N	Interleave	A/N	D	C×N	D×C×N	A×D×C
512/N	Pipeline	A/N	D/(N+1)	C×N	D×C×N/(N+1)	A×D×C×N

例として，ワード長を 256 ビット（2 分割）とした場合の各ループ構造を示す．また，**表 1** に，データパスの分割数を N とした際の回路性能を，512 ビットインタリーブアーキテクチャを基準として示す．なおこれらの性能は，データパスの分割によるオーバヘッドは生じないとした場合の理想値である．N 分割のパラレルおよびインタリーブ型では，分割数 N が増えるほど面積が小型化されることが分かる．ただし，分割に応じて処理サイクル数が増加するため，結果として回路効率（回路面積あたりの処理速度）は，分割数によらず一定の $D \times C \times A$ に保たれる．このように，パラレルおよびインタリーブ型では，分割数に応じて，回路面積と実行時間のトレードオフが存在する．また，N 分割のパラレルアーキテクチャは，$2N$ 分割のインタリーブアーキテクチャと同等の回路性能となる．ただし，データパスの分割は面積の小型化を目的とするため，面積が大型になるパラレル型のアーキテクチャのデータパスを分割する意味はあまりない．パイプライン型のアーキテクチャも同様に，分割数 N を増やすと面積が小型化するとともにサイクル数が増大するというトレードオフを有する．ただし，パイプライン型では，分割数に応じてパイプラインのステージ数を増やせるという特徴がある．具体的には，分割数 N に対し，$N+1$ ステージまでならば，

データの流れを止めることなく分割できる．この結果，理想的には分割数に比例して動作周波数が向上することになる．このようにして向上した動作周波数が，増大するサイクル数と相殺されるため，処理速度を落とすことなく小型化が実現できる．

本節では，議論を簡単にするためデータパスの分割によるオーバヘッドを考慮しなかった．実際には，分割した回路コンポーネントの共有に要するセレクタが，回路面積と動作速度に悪影響を与える．また，パイプライン型のアーキテクチャにおいては，各ステージのクリティカルパスを等しくできないと，動作周波数が思ったほど向上しなくなる．このような，回路の設計段階でないと明確にできない要因については，6 章の性能評価において議論を行う．

3.3　提案アーキテクチャ

提案アーキテクチャは，上述した，(i) ループ構造と，(ii) ワード長の 2 点に基づいて構成する．この 2 つの組合せにより，多様な性能を有するアーキテクチャが実現できる．これにより，回路設計者は，(i) ループ構造および，(ii) ワード長を選択することで，所望の性能を有する回路を設計できる．以下では，提案アーキテクチャと既存のアーキテクチャを比較するとともに，その利点について論じる．

パラレルおよびインタリーブ型アーキテクチャは，既存のアーキテクチャにおいて採用されているため，提案アーキテクチャは，既存アーキテクチャを包含する．具体的には，文献 12)，13) のアーキテクチャは，512 ビット幅のパラレル型，文献 11) は 512 ビット幅のインタリーブ型，文献 14)，15) は 64 ビット幅のインタリーブ型に対応する．これに対し本稿では，インタリーブ型およびパラレル型を 256，128，64 ビットと，複数のワード長へと拡張する．一方，パイプライン型のアーキテクチャについては，文献 11) で触れられているものの，これは，互いに独立な複数のメッセージを並列処理する方式であった．そのため，最大のスループットを実現するには，複数組の独立したメッセージを入力し続ける必要があり，実用性が低い．これに対し，提案するパイプライン型アーキテクチャは，1 ラウンド内でステージを分割したインナーパイプラインである．本アーキテクチャでは，2 つに分割した演算器で，データ撹拌と鍵スケジュールが並列的に動作する．この結果，入力メッセージが 1 組のみでも最大のスループットを実現できる．

上述のとおり，提案アーキテクチャの利点は，性能が高いスケーラビリティを有する点にある．これに対し，性能のスケーラビリティは，文献 14)–16) に示されるような小型な基本回路を複数並べても実現できる．以下では，このような実装を「並列実装」，並列実装を構成する回路を「基本回路」と呼ぶことにする．並列実装は，基本回路を 1 つだけ設計し，その複製を作るだけでスケーラビリティが実現できるため，設計が容易である．一方，性能の面において，提案アーキテクチャに劣る．以下では，提案アーキテクチャと並列実装の差を，(a) 速度と (b) 回路面積の 2 点より比較する．

(a)　速度

メッセージ $m_1 \cdots m_i$ の処理は依存性を持つため，並列実装で分散処理できない．そのため，1 組しかメッセージがない場合は，1 つの基本回路しか動作しない．並列実装で最大のスループットを得るには，独立した複数のメッセージを同時に入力する必要がある．これは，前述した文献 11) のパイプラインアーキテクチャと同じ欠点をかかえており，入力するメッセージが少ない場合，スループットが低下する．これに対し，提案アーキテクチャは，入力するメッセージが 1 つだけであっても，最大のスループットを得ることができる．また，レイテンシについても差がある．並列実装では，回路のレイテンシは，基本回路のレイテンシに制約される．小型な基本回路は低速であるため結果として並列実装のレイテンシが悪化する．

(b)　回路面積

並列実装では，個々の基本回路が，メッセージ，鍵，およびハッシュ値を格納するためのレジスタを必要とする．それぞれのレジスタは 512 ビットの容量を持つため，並列実装を n 並列にした場合は，$512 \times 3 \times n$ ビット分のレジスタが必要となる．これに対し，提案アーキテクチャを用いる場合は，512×3 ビットのレジスタだけで十分である．このレジスタのオーバヘッド分，並列実装の回路面積は増大することになる．

また，もし並列実装を行う場合でも，本稿の提案アーキテクチャを基本回路に使用することで性能の改善が望める．並列実装では，所望の回路面積とスループットを並列度によって容易に調整できる．そのため，基本回路にとっては，回路効率（面積あたりのスループット）が最も重要な性能指標である．提案アーキテクチャは性能に高いスケーラビリティを有するため，その中で回路効率が高いものを基本回路に採用すれば，並列実装の性能を改善することができる．

4.　演算コンポーネント

本章では，Whirlpool を構成する演算コンポーネントの設計について述べる．具体的には，2 章で示した Whirlpool のサブ関数 γ，π，θ について，それぞれ効率的な実装手法を示す．

4.1　関数 γ：S-box

S-box の回路実装では，(i) ROM によるルックアップテーブルおよび，(ii) 組合せ回路を

用いるのが主な手法である．前者は，8 ビット入出力で容量が 256 バイト（1×2^8）の ROM に S-box の内容を格納するもので，ソフトウェア実装でも一般的である．しかし，ASIC では ROM が比較的高価なため，S-box のように小さな入出力データの実装では，組合せ回路を用いるのが一般的である．そのため，本稿もこれに従って議論を進める．

S-box がランダムテーブルである場合，8 ビットの入出力関係を仕様として与え，市販の論理合成ツールにより組合せ回路を生成するのが一般的であると考えられる．しかし，2 章で示したように，Whirlpool の S-box は mini-box による階層的な構造を有するため，この階層を回路構造にマッピングすれば，回路規模の小型化が可能となる[1),2)]．これらの手法による回路規模の違いをメモリ容量によって見積もる．このとき，8 ビット入出力のテーブルでは 256（$= 2^8$）バイトのメモリに対応する組合せ回路が必要である．一方，mini-box を用いた方法では，80（$= 5 \times 2^4$）バイトのメモリに対応する組合せ回路で十分である．mini-box による S-box 実装のもう 1 つの利点は，パイプラインの分割が容易な点である．3.2 節で述べたように，パイプラインのステージ数を増やすことで動作周波数を向上させていくと，S-box の処理がクリティカルパスとなることが懸念される．この際，mini-box の間にパイプラインレジスタを挿入することで容易にクリティカルパスを分割でき，速度の向上が可能となる．

4.2　関数 π：Data manager

関数 π はバイト単位のシフトであるため，回路としては配線の入れ替えのみで実装できる．しかし，これはワード長が 512 ビットの実装の場合だけであり，ワード長が 256，128，64 ビットの場合は，付加回路が必要となる．具体的には，関数 π のシフトを行う配線の入れ替えに加えて，入出力データのスケジュールのためのセレクタが必要となる．**図 6** に，単純に実装した関数 π の演算コンポーネントを示す．影つきの四角はメッセージまたは鍵を格納するレジスタを表しており，各出力は後段の演算コンポーネントへ接続されることを想定している．この図から，データの入出力に多くのセレクタが必要となることが分かる．

Data manager は，文献 14) において提案された回路コンポーネントであり，関数 π による列方向のシフトを，シフトレジスタとセレクタを組み合わせて実装する手法である．**図 7** に示す 64 ビット Data manager は 8 段のシフトレジスタから構成され，中間値を表す 512 ビット行列を 1 行（8 バイト）ごとに 8 サイクルかけて入力する．図において，影付きの四角は 8 ビットレジスタを表しており，レジスタ中に示した値は，図 3 に示す値を入力した後のレジスタの内部状態を示したものである．セレクタの制御を適切に行うことで，8 サイクル後に関数 π と等価な転置が行われたデータがシフトレジスタに格納される．Data

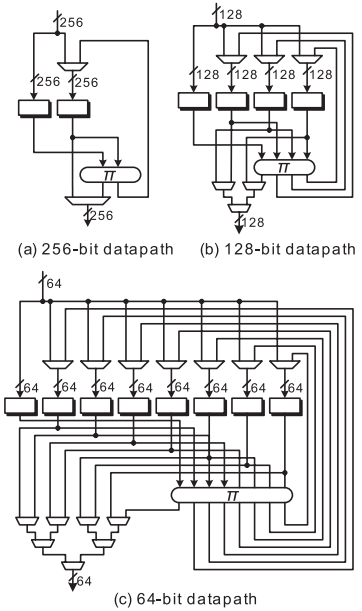

図 6　関数 π のストレートな実装
Fig. 6　Straight forward implementations of the function π.

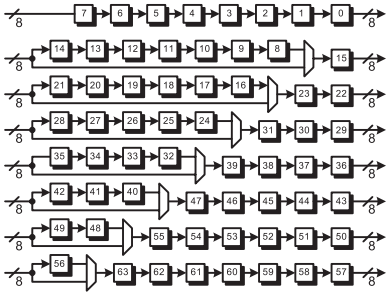

図 7　64 ビット Data manager
Fig. 7　64-bit Data manager.

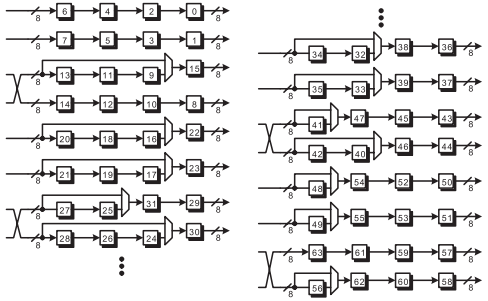

図 8　128 ビット Data manager
Fig. 8　128-bit Data manager.

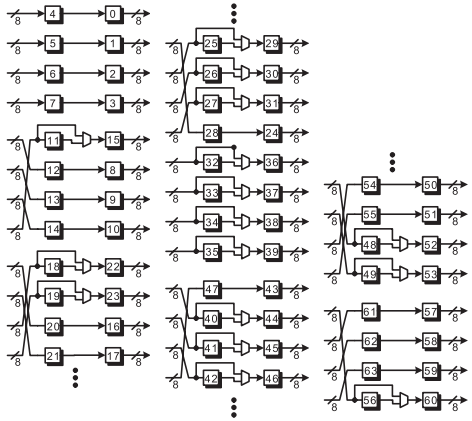

図 9　256 ビット Data manager
Fig. 9　256-bit Data manager.

manager は，図 6 の単純な手法と比較して，セレクタの個数を削減できる．また，セレクタツリーが不要なためクリティカルパスも短縮できる．文献 14) ではワード長 64 ビット用の Data manager のみが提案されているが，本稿では，これを 256，128 ビットに拡張する．256 ビットおよび 128 ビット Data manager を**図 8**，**図 9** にそれぞれ示す．以下では，図 8 の 128 ビット Data manager の動作を中心に説明を行う．128 ビット Data manager では，Whirlpool の内部状態を表す 8 × 8 バイト行列のうち 2 行に対応する 16 バイトが毎サイクル入力され，対応する出力が 16 バイトごとに出力される．図 3 の入出力関係では，1 サイクル目の入力が {0, 1, 8, 9, 16, 17, 24, 25, 32, 33, 40, 41, 48, 49, 56, 57} であり，対応する出力 {0, 1, 15, 8, 22, 23, 29, 30, 36, 37, 43, 44, 50, 51, 57, 58} が 4 サイクル後に出力される．図 9 の 256 ビット Data manager の動作も同様である．

表 2　関数 π の実装に必要なセレクタ数と遅延時間
（2 入力セレクタで正規化）
Table 2　Required number of selectors and delay for the function π (Normalized by a 2-way selector).

Implementation	Word length	# of selectors	Delay
Straightforward	256	512	1
	128	768	2
	64	896	3
Data manager	256	128	1
	128	96	1
	64	56	1

図 6 に示す単純な実装と，図 7〜9 に示す Data manager による実装の比較を**表 2** に示す．表には，回路面積と遅延時間の理論値を 2 入力セレクタの回路面積と遅延時間により正規化した値を表示している．この結果より，Data manager を用いることで，いずれのワード長においても，回路面積と遅延時間が大きく改善されることが分かる．

4.3 関　数　θ

2 章で述べたように，関数 θ は拡大体 $GF(2^8)$ 上の定数行列の乗算として定義されるため，$GF(2^8)$ 上の加算器および定数乗算器として実装できる．関数 θ の定数行列には，$\{1, 2, 4, 5, 8, 9\} \in GF(2^8)$ が含まれる．それぞれの定数乗算器を作成することもできるが，$5 = 4 + 1$，$9 = 8 + 1$ と，2 のべき乗項に分解可能したほうが効率的である．このとき，式 (3) は，次のように変形される．

$$b_{i0} = a_{i0} \oplus (8 \otimes a_{i1} \oplus a_{i1}) \oplus (2 \otimes a_{i2}) \oplus (4 \otimes a_{i3} \oplus a_{i3}) \oplus (8 \otimes a_{i4}) \oplus a_{i5} \oplus (4 \otimes a_{i6}) \oplus a_{i7} \quad (5)$$

さらに，2 をくくり出して式を変形することで，対応する回路の定数乗算器の個数を削減できる．

$$b_{i0} = a_{i0} \oplus a_{i1} \oplus a_{i3} \oplus a_{i5} \oplus a_{i7} \oplus 2 \otimes (a_{i2} \oplus 2 \otimes (a_{i3} \oplus a_{i6} \oplus 2 \otimes (a_{i1} \oplus a_{i4}))) \quad (6)$$

この式に基づく，関数 θ の演算器を**図 10** に示す．上述の式変形の結果，3 つの 2 倍算回路を各定数乗算で共有することが可能となり，回路が小型化できる．これは Whirlpool の

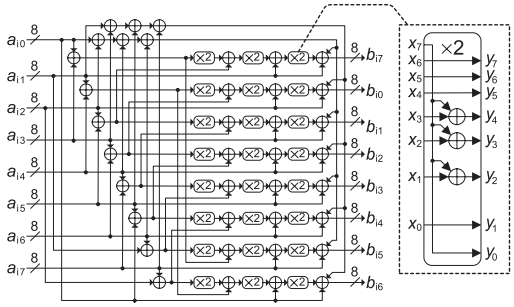

図 10　関数 θ の小型実装
Fig. 10　Compact implementation of the function θ.

仕様書[1] で 8 ビットプロセッサ用アルゴリズムとして示されている式を回路に置き換えたものである．ここまでは，小型実装について検討したが，以下では，高速実装について考える．図 10 より，関数 θ の演算器は，XOR のみで構成されることが分かる．そのため，論理関数を積和型に展開すると，

$$b_{ij} = \bigoplus_{j} d_{ij} a_{ij}, (0 \le i \le 7, 0 \le j \le 7) \tag{7}$$

$$d_{ij} \in \{0, 1\} \tag{8}$$

と記述できる．このような論理関数を最小段数の論理ゲートで構成した場合が，最も高速となる．このとき，論理ゲートは XOR ゲートのツリーとなり，ツリーの深さが，クリティカルパスとなる．本稿の実装では，すべての実装において図 10 に示す回路コンポーネントを用い，論理合成ツールにより展開を行っている．

5. データパス

3，4 章で述べた演算コンポーネントを用いて**表 3** に示す 9 種類のデータパスの設計を行った．高速に向くパラレル型のアーキテクチャは同じく高速に向くワード長 512 ビットのみを設計した．一方，インタリーブ型とパイプライン型のアーキテクチャは，ワード長 512，256，128，64 ビットの 4 種類のについて設計を行った．以降では，ワード長 512 ビットおよび 128 ビットのアーキテクチャを例として，設計したデータパスとその動作を示す．256 および 64 ビットのワード長におけるデータパスは，付録を参照されたい．

表 3　設計したデータパス
Table 3　Designed datapaths.

Word length	Loop structure		
	Parallel	Interleave	Pipeline
512	○	○	○
256	×	○	○
128	×	○	○
64	×	○	○

5.1　512 ビットアーキテクチャ

図 11 に，ワード長 512 ビットにおける 3 種類のアーキテクチャを示す．図中のアーキテクチャは，それぞれ (a) パラレル型，(b) インタリーブ型，および (c) パイプライン型のループ構造を採用したものである．

図 11 (a) のパラレル型アーキテクチャは，3 章で述べたように，関数 $\rho[k]$ の演算器を 2 つ有し，1 サイクルで関数 $\rho[k]$ の処理を完了する．そのため，512 ビットメッセージブロックの圧縮には，10 サイクルを要する．一方，図 11 (b) のインタリーブ型アーキテクチャは，1 つの関数ブロック $\rho[k]$ をデータ攪拌と鍵スケジュール部で共有するので，処理サイクル数はパラレル型の 2 倍の 21 サイクルとなる．ラウンド鍵レジスタ $RegK$ の前に 512 ビットのセレクタを挿入することで 20 サイクルに短縮することも可能であるが，動作周波数の低下と回路規模の増加が生じるため，21 サイクル実装の方が回路効率の点では有利である．図 11 (c) のパイプライン型は，データ圧縮部と鍵スケジュール部でインタリーブして使用していた (b) の関数ブロック $\rho[k]$ を 2 分割し，その間に 512 ビットレジスタを置くことでパイプライン処理を行う．本データパスでは，パイプラインレジスタを S-box の内部に挿入した．パイプラインでは，各ステージの遅延がバランスするようにした際に最も効率が良くなる．しかし，分割箇所に応じて，要するパイプラインレジスタの個数が変化するため，遅延とレジスタの個数の両方を考慮してレジスタの挿入箇所を決定する必要がある．本データパスでは，各ステージの遅延をバランスさせるためには S-box の内部にパイプラインレジスタを挿入する必要があった．この際，mini-box E，E^{-1} の直前にパイプラインレジスタを挿入すれば個数が 512 ビット数で済むことから，そこを分割箇所とした．もし，stage 1 のクリティカルパスが大きい場合は，mini-box R と次段の XOR の間にパイプラインレジスタを挿入することで，高速化が行える．しかし，このような場合，パイプラインレジスタの個数は 768 ビットに増大する．

ハッシュ値の更新は，式 (1) に示すように，H_{i-1}，m_i，および $W[H_{i-1}](m_i)$ の 3 項の XOR によって行われる．文献 11)–14) では，式 (1) の加算を専用の XOR でそのまま行っ

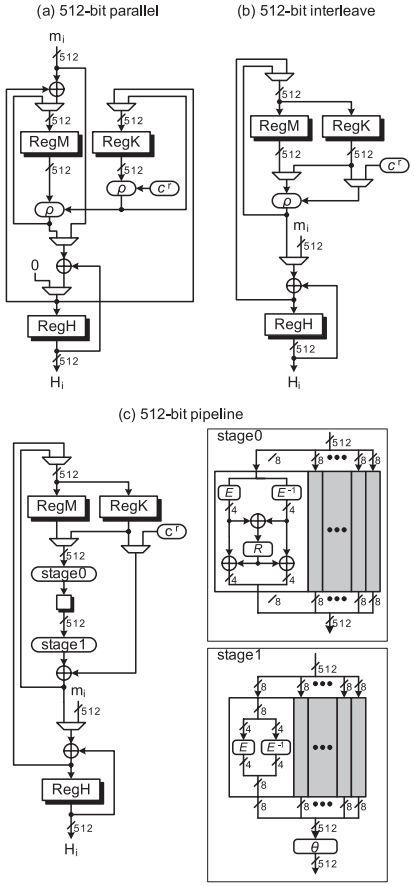

図 11　512 ビットアーキテクチャ
Fig. 11　512-bit architectures.

ている．それに対し，図 11 (a) では，

$$RegH \leftarrow m_i \oplus H_{i-1} \tag{9}$$

$$RegH \leftarrow RegH \oplus W[H_{i-1}](m_i) \tag{10}$$

と，2 ステップに分けて実行することで XOR ゲートを共有している．1 つ目の XOR は m_i の入力時，2 つ目の XOR は関数 $\rho[k]$ の完了時に行う．これにより，サイクル数を増やすことなく，XOR の数を削減できる．このような実装方法は文献 11)，12) でも採用されていたが，図 11 (b) では，さらなる小型化のために図 1 の Whirlpool 処理を変形して，K^0 の加算用の XOR と下部のハッシュ値積算の XOR を次のように共有している．すなわち，

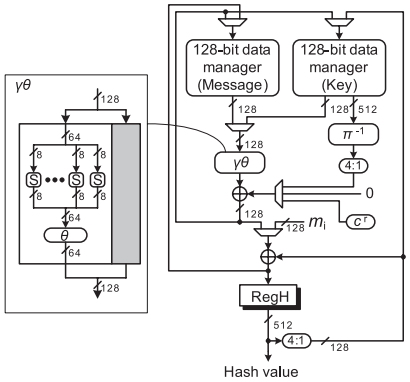

図 12　128 ビットインタリーブアーキテクチャ
Fig. 12　128-bit interleave architecture.

$$RegH \leftarrow m_i \oplus H_{i-1} \tag{11}$$

$$RegM \leftarrow RegH \tag{12}$$

として，$m_i \oplus H_{i-1}$ を図 11 (a) と同様にレジスタ $RegH$ に書き込んだ後に再利用している．図 11 (a) ではこの値を関数 W の演算に再利用していないが，それは 1 クロック分増加することで高速実装に不利となるためである．

なお，パラレル型およびインタリーブ型のアーキテクチャでは，S-box を (i) 8 ビット入出力のテーブルに基づく組合せ回路および，(ii) mini-box の 2 通りの方法で実装した．以降では，これらを (i) 8-bit Table，(ii) 4-bit Table として参照する．一方，パイプライン型のアーキテクチャでは，S-box を内部でパイプラインステージに分割する必要があるため，4-bit Table のみを使用した．このような S-box の実装方針については，他のワード長のデータパスにおいても同様である．

5.2　128 ビットアーキテクチャ

図 12 に 128 ビットインタリーブアーキテクチャのデータパスを示す．本アーキテクチャは，データ攪拌と鍵スケジュールにおいて，ワード長 128 ビットの演算コンポーネントを交互に使用する．メッセージおよび鍵のデータは 2 つの Data manager 内のシフトレジス

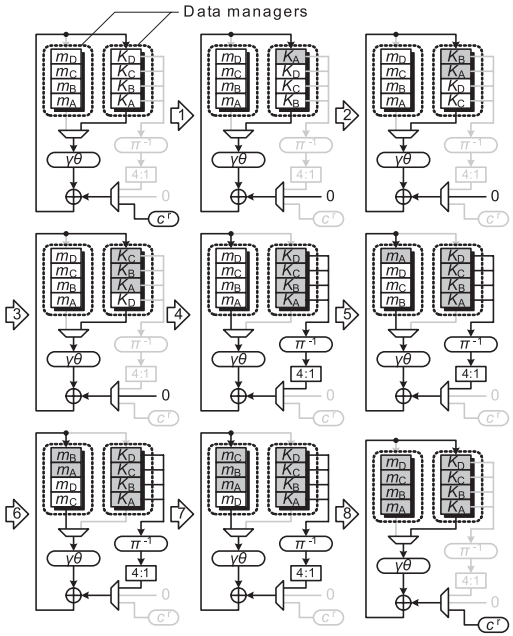

図 13　128 ビットインタリーブアーキテクチャの動作例
Fig. 13　Example behavior of the 128-bit interleave architecture.

タにそれぞれ格納される．アルゴリズムの仕様では関数 γ，π，θ の順に処理を行うが，関数 γ と π はいずれもバイトごとに独立した演算なので順序を交換している．これは，データ依存性の解決が必要な関数 π をその他の関数と切り離し，Data manager として実装するためである．なお，Data manager は，レジスタへのデータの格納と同時に関数 π の処理を行うため，関数 $\sigma[k]$ において鍵 K^i を使用する場合，レジスタに格納されている値はすでに π により転置されている．そこで，鍵の値を元に戻したうえで加算するために，逆変換用の演算コンポーネント π^{-1} を用意している．この場合，π^{-1} の実装は配線のみで可能だが，128 ビットの 4：1 セレクタが必要となる．ただし，関数 π^{-1} を利用した場合でも，定数 c^r を選択するセレクタと共有化することによりセレクタ数は削減できるとともに，関数 γ，θ と並列に動作するため，クリティカルパスへのオーバヘッドもない．

本アーキテクチャの動作を**図 13** に示す．図中の影付きの四角がレジスタであり，内部に

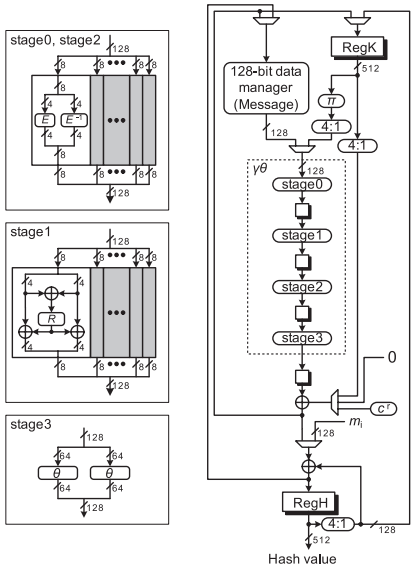

図 14　128 ビットパイプラインアーキテクチャ
Fig. 14　128-bit pipeline architecture.

表示された記号 $m_A \cdots m_D$ および $K_A \cdots K_D$ は，512 ビットの中間値を，128 ビットごとに 4 分割したものである．ここで，初期状態の値を格納するレジスタを白，関数 $\rho[k]$ の出力を格納するレジスタは灰色で表示している．図は，最初の 4 サイクルで $K_A \cdots K_D$ の変換を行い，続く 4 サイクルで $m_A \cdots m_D$ の変換を行う様子を表現している．このような計 8 サイクルにより，関数 $\rho[k]$ 2 個分の処理が完了する．そのため，関数 $\rho[k]$ を 20 個有する関数 W の処理は 80 サイクルで終了する．これに加え，512 ビットのメッセージブロック m_i を 128 ビットずつ入力するのに 4 サイクル必要なため，処理全体では 84 サイクルが必要となる．

図 14 に，128 ビットパイプラインアーキテクチャに基づくデータパスを示す．図 12 のインタリーブアーキテクチャで必要なレジスタはメッセージと鍵を格納する 1024（$= 2 \times 512$）ビットだったのに対し，図 14 では 512 ビットのパイプラインレジスタが追加されている．パイプラインは，表 1 に示した最大段数の 5 段に分割した．この際，512 ビットパイプラインアーキテクチャの際と同様に，ステージ間の遅延のバランスと，パイプラインレジスタの個

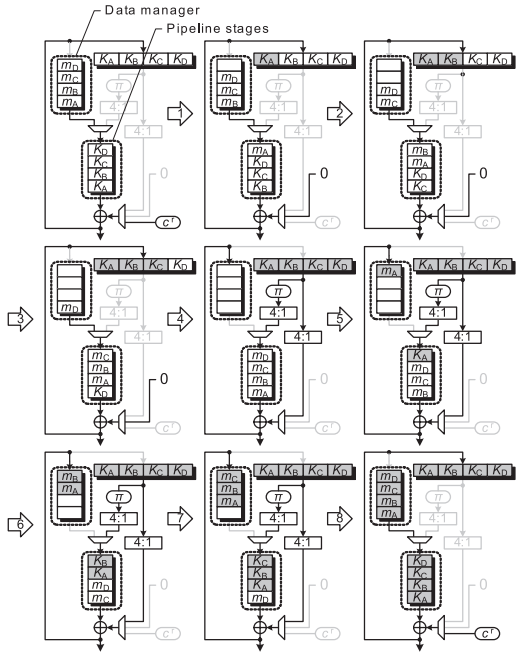

図 15　128 ビットパイプラインアーキテクチャの動作例
Fig. 15　Example behavior of the 128-bit pipeline architecture.

数の両方を考慮し，分割方法を決定した．図中に示すパイプラインステージ $stage0 \sim stage3$ では，3 分割した S-box および関数 θ の演算をそれぞれ行う．また，図中にはステージとして明示していないが，5 段目のステージには，関数 $\sigma[k]$ の XOR ゲートが存在する．

インタリーブ型のアーキテクチャではメッセージと鍵の両方に Data manager を採用していたのに対し，パイプライン型のアーキテクチャではメッセージ側のみに Data manager を用いている．これは，鍵の値を，(i) $stage0$ への入力と，(ii) 関数 $\sigma[k]$ での鍵加算という 2 種類の用途で使用するため，1 つのルールでスケジューリングを行う Data manager ではデータの依存関係を矛盾なく取り扱えないためである．

図 15 に，本データパスの動作例を示す．ここでも，図 13 と同様に，初期状態のレジスタを白，関数 $\rho[k]$ の変換が済んだデータを格納するレジスタを灰色で表示している．初期状態では，$K_A \cdots K_D$ がすでにパイプラインレジスタへ格納されている．最初の 4 サイクルで，$K_A \cdots K_D$ の処理が完了し，鍵レジスタが更新される．これと同時に，メッセージ $m_A \cdots m_D$ が，順次パイプラインレジスタへ格納される．その後，さらに 4 サイクルをかけて，メッセージ $m_A \cdots m_D$ の処理が完了する．ここで，メッセージの先頭 m_A が鍵加算の XOR に到達する直前に鍵 $K_A \cdots K_D$ すべての更新が完了する，パイプラインが滞ることなく処理が完了する．図に示すように，関数 $\rho[k]$ の処理 2 回分を 8 サイクルで完了するため，128 ビットインタリーブアーキテクチャと同様に関数 W の処理を 80 サイクルで完了する．メッセージ入力およびパイプラインを空にするためのレイテンシが加わるため，1 メッセージブロックの処理に合計 88 サイクルを要する．

6. 性能評価

表 3 に示した 9 種類のデータパスについて，ASIC 実装での性能評価を行った．標準セルライブラリに STMicroelectronics 社の 90 nm CMOS（動作電圧 1.2 V 版）[20] を用い，Synopsys 社 Design Compiler によって面積優先と速度優先の 2 種類のオプションで論理合成した．結果を**表 4** および**図 16** に示す．ここで，回路規模は 2 入力 NAND セルを 1 ゲートで換算し，動作速度は最悪条件下の値とした．また，回路効率（Efficiency）を回路面積あたりのスループット（Kbps/gate）として評価しており，この値が大きいほど効率の良い実装といえる．また，レイテンシを，1 K バイトのメッセージを処理するのに必要な処理時間（μs/KB）として表示した．比較のため，SHA-256，SHA-512 の回路[19] を同一条件下で合成したときの性能と Whirlpool の FPGA 実装[11)–16)] による性能も合わせて示した

最も高いスループットを得たのは，512 ビットパラレルアーキテクチャに 8-bit Table を用いた速度優先実装で，28.0 Gbps@546.5 MHz である．しかし，S-box に大きな回路リソースが必要なため，回路規模も 179.0 Kgates と最大であった．最も小型だったのは 64 ビットインタリーブアーキテクチャに 4-bit Table を用いた面積優先の実装であり，13.6 Kgates（817 Mbps@268.1 MHz）で回路効率は 60.02 Kbps/gate であった．回路効率はワード長が狭いほど低くなる傾向にあった．これは，データパスの分割に必要なセレクタなどのオーバヘッドによるものである．また，パイプライン型のアーキテクチャは，他と比較して高い回路効率を実現しており，256 ビットアーアーキテクチャでは全体として最大の回路効率 372.3 Kbps/gate が得られた．

インタリーブ型のアーキテクチャは合成オプションの違いにおける性能差が大きく現れたものの，面積優先の場合はパイプライン型よりも小さくなる傾向にあった．パイプライン型と比較してレジスタの個数が少ないため，この結果は妥当である．S-box の実装手法に関

表 4　性能評価
Table 4　Performance evaluation.

Design	Algo -rithm	Message Block (bits)	Platform	Cycle	Datapath architecture	S-box	Optimize	Area	Operation Frequency (MHz)	Latency [μs/KB]	Through-put (Mbps)	Efficiency (Kbps/gate)
This work	Whirl -pool	512	90 nm CMOS	10	512-bit Parallel	8-bit Table	Area	103,633 gates	211.42	0.76	10,825	104.45
							Speed	179,035 gates	546.45	0.29	27,978	156.27
						4-bit Table	Area	43,726 gates	210.97	0.76	10,802	247.03
							Speed	103,408 gates	523.56	0.31	26,806	259.23
				21	512-bit Interleave	8-bit Table	Area	58,792 gates	210.08	1.60	5,122	87.12
							Speed	97,541 gates	518.13	0.65	12,633	129.51
						4-bit Table	Area	29,577 gates	210.08	1.60	5,122	173.18
							Speed	64,549 gates	500.00	0.67	12,190	188.86
				21	512-bit Pipeline	4-bit Table	Area	30,105 gates	363.64	0.92	8,866	294.50
							Speed	40,330 gates	574.71	0.58	14,012	347.43
				42	256-bit Interleave	8-bit Table	Area	36,201 gates	266.67	2.52	3,251	89.80
							Speed	64,796 gates	568.18	1.18	6,926	106.90
						4-bit Table	Area	21,495 gates	266.67	2.52	3,251	151.23
							Speed	42,972 gates	529.10	1.27	6,450	150.10
				44	256-bit Pipeline	4-bit Table	Area	21,395 gates	558.66	1.26	6,501	303.84
							Speed	35,520 gates	1,136.36	0.62	13,223	372.27
				84	128-bit Interleave	8-bit Table	Area	22,527 gates	269.54	4.99	1,643	72.93
							Speed	37,865 gates	571.43	2.35	3,483	91.98
						4-bit Table	Area	15,891 gates	268.10	5.01	1,634	102.83
							Speed	28,337 gates	537.63	2.50	3,277	115.64
				88	128-bit Pipeline	4-bit Table	Area	16,677 gates	574.71	2.45	3,344	200.50
							Speed	23,230 gates	1,111.11	1.27	6,465	278.29
				168	64-bit Interleave	8-bit Table	Area	16,675 gates	268.10	10.03	817	49.00
							Speed	24,029 gates	568.18	4.73	1,732	72.06
						4-bit Table	Area	13,614 gates	268.10	10.03	817	60.02
							Speed	20,554 gates	546.45	4.92	1,665	81.02
				176	64-bit Pipeline	4-bit Table	Area	14,762 gates	564.97	4.98	1,644	111.34
							Speed	19,500 gates	1,041.67	2.70	3,030	155.40
Satoh et, al. [17]	SHA -256	512	90 nm CMOS	72			Area	9,764 gates	362.32	3.18	2,576	263.88
							Speed	13,624 gates	490.20	2.35	3,486	255.86
	SHA -512	1024		88			Area	17,104 gates	209.64	3.36	2,439	142.63
							Speed	27,239 gates	404.86	1.74	4,711	172.95
Kitsos et. al. [11, 12]	Whirl -pool	512	Virtex-E xcv 1000E (0.18um)	10	512-bit Parallel	8-bit Table		5,585 slices	85.50	1.83	4,480	
				20	512-bit Interleave	8-bit Table		3,751 slices	93.00	3.44	2,380	
McLoone et. al. [13, 14]			Virtex-4 xc4vlx100 (90 nm)	5	512-bit Parallel 2-round	4-bit Table		13,210 slices	47.80	1.67	4,896	
				10	512-bit Parallel	8-bit Table		4,956 slices	93.56	1.71	4,790	
						4-bit Table		7,507 slices	91.37	1.75	4,678	
			xc4vlx15 (90 nm)	160	64-bit Interleave	4-bit Table		709 slices	134.00	19.05	430	
Pramstaller et. al. [15]			Virtex-II Pro xc2vp40 (130 nm)	176	64-bit Interleave	4-bit Table		1,456 slices	131.00	21.45	382	
Alho et. al. [16]				1344	8-bit Interleave Parallel	4-bit Table		376 slices	214.00	100.52	82	

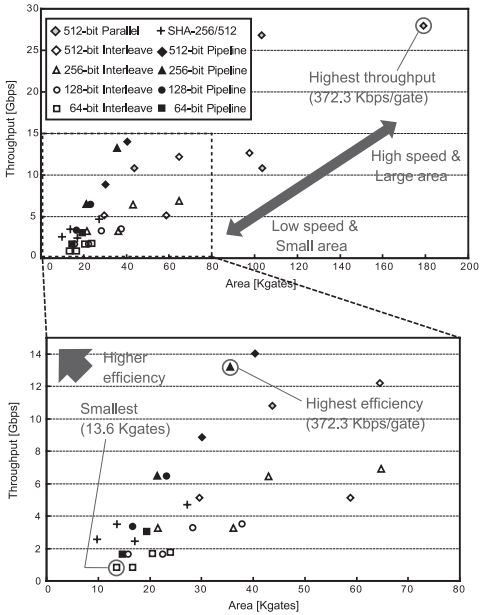

図 16　性能値のグラフ
Fig. 16　Graph of performances.

しては，4-bit Table による実装が，8-bit Table よりも高い回路効率を示した．

パイプライン型のアーキテクチャはスループットと回路効率の両面において高い性能を示した．しかし，分割数の増加にともなって単純に回路効率が向上するという傾向には必ずしもなっていない．これは，ワード長の分割によるオーバヘッドが無視できないとともに，パイプラインの段数を増やし続けると関数 $\rho[k]$ がクリティカルパスでなくなり，動作周波数の向上が止まるためである．具体的には，制御回路におけるデコードやセレクタがボトルネックとなっており，64 ビットパイプラインアーキテクチャにおいてはその影響が顕著であった．付録の図 19 に示す 64 ビットパイプラインアーキテクチャにおいて，多くのステージに演算器が配置されていないのは，そのボトルネックを解消できないためである．本稿では，3.2 節で述べたとおり，分割数 N に対し，最大のパイプライン段数 $N+1$ を使用するという指針で設計を行った．しかし，上述の例のように，ステージ数の増加による動作周波数の向上が止まる場合，パイプラインステージ数を削減することで，パイプラインレジスタの個数を削減した方が結果として高い回路効率を得られると考えられる．

スケーラビリティという観点からは，回路面積で 13.6〜179.0 Kgates，スループットで 0.8〜28.0 Gbps という幅広い性能が得られたことにより，提案アーキテクチャが性能について高いスケーラビリティを持つことが示された．ピーク性能としては，スループットを重

視する場合は 512 ビットパラレルアーキテクチャ，回路面積重視では 64 ビットインタリーブアーキテクチャ，回路効率重視では 256 ビットパイプラインアーキテクチャを採用するのが最適となった．一方，回路効率についても 49.0〜278.3 Kbps/gate と広い範囲の値が得られた．回路効率については，速度や回路面積と異なり，高いほど好ましい値である．そのため，回路設計者は，設計要求を満たす範囲で，回路効率を最大化するアーキテクチャを選択するのが適当である．

同じ ASIC ライブラリを用いた SHA-256/-512 の実装結果は，Whirlpool と比較して小型 · 低速となる傾向となった．特に SHA-256 の面積 9.8 Kgates は，いずれの Whirlpool 実装よりも小型である．しかしこれは SHA-256 のハッシュ値が 256 ビットと Whirlpool の半分であるためで，ハッシュ値の大きさをそろえた場合，Whirlpool はいずれの性能指標においても SHA-512 を上回る．また，SHA-256/512 はハードウェア実装時のアーキテクチャの自由度が低く，Whirlpool のように多様な性能を実現することは困難である．

ASIC 実装と FPGA 実装の結果を比較すると，スループットのピーク値は ASIC 実装が優れていた．FPGA は再構成可能とするための回路構造がオーバヘッドとなるため，これは当然の結果である．なお，前述のとおり，文献 11)–14)，の構成は，本稿で設計した 512 ビットパイプライン，512 ビットインタリーブアーキテクチャ，および 64 ビットインタリーブアーキテクチャとほぼ同等である．評価環境の違いにより単純な比較はできないが，これらの共通項により提案アーキテクチャを FPGA へ移植した際の性能を予測することは可能であろう．なお，特徴的な場合として，文献 16) の実装では，回路規模 376 スライスというきわめて小型な性能を示している．この理由としては，ワード幅が 8 ビットである点，および，ブロック RAM によって実装した演算コンポーネントが，スライス数に計上されていない点があげられる．そのため，文献 16) のアーキテクチャを ASIC へ移植することを考えた場合，ブロック RAM 分の回路面積が追加されることを考慮する必要がある．すなわち，データパス幅を 8 ビットに分割したとしても，RegM，RegK，RegH に対応する 1,536 ビットの記憶容量は最低限必要である．なお，本稿で使用したスタンダードセルライブラリにおいて，1,536 ビットレジスタの回路面積は，およそ 6.5 K ゲートであった．そのため，文献 16) のアーキテクチャを ASIC へ移植しても，レジスタを用いて実装する限りは，回路面積の改善は，提案アーキテクチャの 13.6 K ゲートと比較して最大でも 50%程度にとどまると予測される．

7. ま と め

本稿では，512 ビットハッシュ関数 Whirlpool の高スケーラブル回路アーキテクチャを提案した．提案する回路アーキテクチャは，データパスのループ構造とワード長を変更することで多様な演算性能を実現する．各演算コンポーネントの構成を示すとともに，提案アーキテクチャによる具体的な設計例として 9 種類のデータパスを示した．これらのデータパスを 90 nm CMOS スタンダード・セルライブラリで評価した結果，性能において高いスケーラビリティが得られた．また，ピーク性能においても，従来の実装や他のハッシュ関数の実装と比較して，最小の回路面積 13.6 Kgates（スループット 817 Mbps），最大スループット 30.0 Gbps（回路面積 35.5 Kgates），および最大の回路効率 372.3 Kbps/gate が得られた．

謝辞　本稿の内容は，科学研究費補助金（挑戦的萌芽研究 No.21650009）の援助により得られた成果の一部である．

参 考 文 献

1) The Whirlpool Hash Function (online). available at http://paginas.terra.com.br/informatica/paulobarreto/WhirlpoolPage.html
2) Barreto, P.S.L.M. and Rijmen, V.: The Whirlpool Hash Function (online). available at http://planeta.terra.om.br/informatica/paulobarreto/whirlpool.zip
3) NESSIE, New European Schemes for Signatures, Integrity, and Encryption (online). available at https://www.cosic.esat.kuleuven.be/nessie/
4) ISO/IEC 10118-3:2004: Information Technology – Security Techniques – Hash-functions – Part 3: Dedicated Hash-functions, Third Edition (2004).
5) Rivest, R.L.: The MD5 Message Digest Algorithm, RFC 1321 (Apr. 1992).
6) The hash function RIPEMD-160 (online). available at http://www.esat.kuleuven.ac.be/˜bosselae/ripemd160.html
7) NIST: Secure Hash Standard (SHS), FIPS PUB 180-2 (Aug. 2002).
8) NIST: FIPS 180-2 Secure Hash Standard Change Notice 1 (Feb. 2004).
9) Menezes, A.J., Oorschot, P.C. and Vanstone, S.A.: *Handbook of Applied Cryptography*, CRC Press (1997).
10) NIST: Advanced Encryption Standard (AES) FIPS Publication 197 (Nov. 2001).
11) Kitsos, P. and Koufopavlou, O.: Efficient Architecture and Hardware Implementation of the Whirlpool Hash Function, *IEEE Trans. Consumer Electronics*, Vol.50, No.1, pp.208–213 (Feb. 2004).
12) Kitsos, P. and Koufopavlou, O.: Whirlpool hash function: Architecture and VLSI implementation, *Proc. 2004 International Symposium on Circuits and Systems,*

2004 (*ISCAS '04*), Vol.2, pp.II-893–6 (23–26 May 2004).
13) McLoone, M., McIvor, C. and Savage, A.: High-speed Hardware Architectures of the Whirlpool Hash Function, *Proc. 2005 IEEE International Conference on Field-Programmable Technology 2005*, pp.147–153 (11–14 Dec. 2005).
14) McLoone, M. and McIvor, C.: High-speed & Low Area Hardware Architectures of the Whirlpool Hash Function, *J. VLSI Signal Processing*, Vol.47, No.1, pp.47–57 (2007).
15) Pramstaller, N., Rechberger, C. and Rijmen, V.: A Compact FPGA Implementation of the Hash Function Whirlpool, *Proc. 2006 ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays*, pp.159–166 (Feb. 2006).
16) Alho, T., Hamalainen, P., Hannikainen, M. and Hamalainen, T.D.: Compact Hardware Design of Whirlpool Hashing Core, *Design, Automation & Test in Europe Conference & Exhibition 2007*, pp.1–6 (16–20 Apr. 2007).
17) 佐藤　証：ハッシュ関数 Whirlpool の ASIC 回路実装，コンピュータセキュリティシンポジウム 2007 予稿集，pp.181–186 (Oct. 2007).
18) 菅原　健，本間尚文，青木孝文，佐藤　証：ハッシュ関数 Whirlpool の小型ハードウェア・アーキテクチャ，2008 年暗号と情報セキュリティシンポジウム，2C3-1 (Jan. 2008).
19) Satoh, A. and Inoue, T.: ASIC Hardware Focused Comparison for Hash Functions MD5, RIPEMD-160 and SHS, *International Conference on Information Technology: Coding and Computing 2005* (*ITCC 2005*), Vol.1, pp.532–537 (4–6 Apr. 2005).
20) Circuits Multi-Projets (CMP), CMOS 90 nm (CMOS090) from STMicroelectronics (online). available at http://cmp.imag.fr/products/ic/?p=STCMOS090

付　　録

本稿では，512/128 ビットアーキテクチャについて，その詳細を述べた．**図 17**，**図 18**，**図 19** では，256 および 64 ビットアーキテクチャについて，データパスを示す．各回路の動作は，5 章に示したものとほぼ同等である．

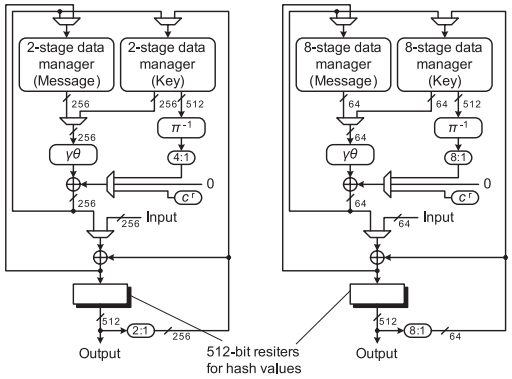

図 17　256 ビットインタリーブアーキテクチャ（左），および 64 ビットインタリーブアーキテクチャ（右）
Fig. 17　256-bit interleave architecture (left) and 64-bit interleave architecture (right).

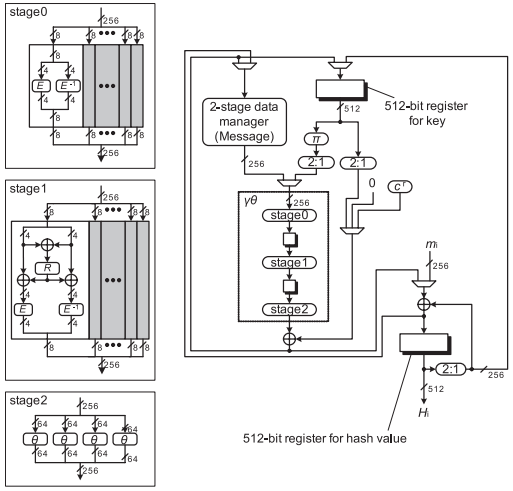

図 18　256 ビットパイプラインアーキテクチャ
Fig. 18　256-bit pipeline architecture.

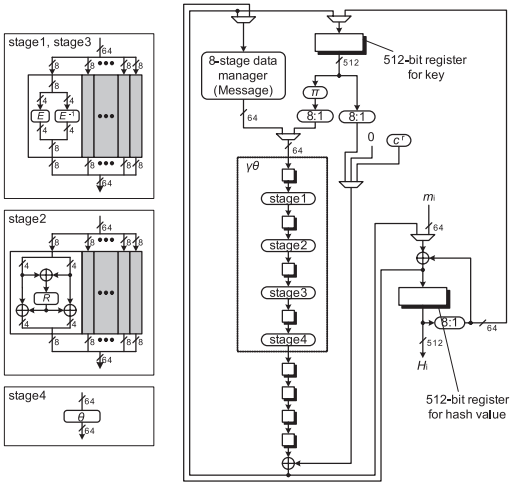

図 19　64 ビットパイプラインアーキテクチャ
Fig. 19　64-bit pipeline architecture.

(平成 21 年 2 月 2 日受付)
(平成 21 年 9 月 11 日採録)

菅原　健

2006 年東北大学工学部情報工学科卒業．2008 年同大学大学院情報科学研究科修士課程修了．同年同博士課程進学．また，同年より日本学術振興会特別研究員を兼任，現在に至る．高性能暗号ハードウェアの設計およびサイドチャネル攻撃に関する研究に従事．IEEE 会員．

本間　尚文（正会員）

1997 年東北大学工学部情報工学科卒業．2001 年同大学大学院情報科学研究科博士課程修了．同年同研究科助手，2007 年同助教．2009 年同准教授．2002〜2006 年科学技術振興機構さきがけ研究者を兼任，現在に至る．ハードウェアアルゴリズム，VLSI 設計技術，ハードウェアセキュリティに関する研究に従事．CRYPTREC 暗号実装委員会委員．IEEE，電子情報通信学会各会員．博士（情報科学）．

青木　孝文（正会員）

1988 年東北大学工学部電子工学科卒業．1992 年同大学大学院工学研究科博士課程修了．同年同大学工学部助手，1994 年同大学大学院情報科学研究科助手，1996 年同助教授，2002 年同教授，現在に至る．超高速ディジタル計算の理論，画像センシング，映像信号処理，バイオメトリクス，VLSI 設計技術，分子コンピューティングに関する研究に従事．英国電気学会フレミング賞およびマウントバッテン賞ほかを受賞．IEEE，計測自動制御学会，電子情報通信学会各会員．博士（工学）．

佐藤　証

1989 年早稲田大学大学院理工学研究科電気工学専攻修士課程修了．同年日本アイ・ビー・エム（株）東京基礎研究所入所．1999 年早稲田大学より博士（工学）授与．2007 年（独）産業技術総合研究所入所，現在に至る．情報セキュリティに関するアルゴリズムおよび，その高性能 VLSI 実装方式に関する研究に従事．博士（工学）．