修士論文

動的再構成可能プロセッサを用いたハッシュ関数の高速化

氏　　名：亀井 雄介
学籍番号：6162080036-7
指導教員：山崎 勝弘　教授
提 出 日：2010 年 2 月 12 日

立命館大学理工学　理工学研究科　創造理工学専攻

内容梗概

本論文では、動的再構成可能プロセッサ DAPDNA-2 上にハッシュ関数である SHA-1 と SHA-256 の回路を、動的再構成を行わない方法での実装と評価を行っている。また、SHA-1 を動的再構成を行う方法で実装する方法についての提案を行っている。

動的再構成可能プロセッサは、動的に回路構成を変化させて処理を行うことが可能で、処理を行いながら最短 1 クロックで回路構成を変更することが可能なデバイスである。

動的再構成可能プロセッサ DAPDNA-2 上に SHA-1, SHA-256 のプログラムを実装し、それぞれの実行時間を計測する。回路構成情報である DNA コンフィギュレーションは DAPDNA-2 用の開発ツール DNA Designer（GUI ツール）を用いて作成し、DNA 上へ実装を行う。全体の制御を行う DAP プログラムは C ソースコードで記述し、DAP 上へ実装する。2 コアプロセッサ（IntelCore2Duo）と 4 コアプロセッサ（IntelCore2Quad）に同様の処理を実行させ、実行時間を比較し、考察を行う。SHA-1 において、動作周波数が約 15 倍である 2 コア、4 コアプロセッサに比べて約 27 倍の速度向上が得られた。SHA-256 において、動作周波数が約 15 倍である 2 コア、4 コアプロセッサに比べて約 36 倍の速度向上が得られた。また、SHA-1 を動的再構成を用いて実装する方法の提案において、使用ハードウェア量、実行クロック数、消費電力を削減できることを検討している。

目次

図目次

表目次

1. はじめに

半導体の微細化技術の進歩によって、半導体集積回路に搭載できる回路規模が増大し、LSI に対して高度で多様な機能の実現をもたらした。現在では、大規模計算機やパーソナルコンピュータなどの計算機だけでなく、人々の身近なものにまで LSI は用いられている。AV 機器、情報家電、携帯電話、ネットワーク制御などの情報通信分野における様々な製品には、SoC（Syetem-on-a-Chip）と呼ばれる集積回路が用いられている。これらの製品は MPEG のコード圧縮や復号、暗号化されたコードの取り扱いなど、強力な演算能力を必要とされる処理を含んでいる。SoC はこれらの処理を行うために、基本的な機能を持った組込み CPU、演算能力を要する数種類の専用ハードウェア、及びメモリや I/O を同一チップ上に混載している。組込み CPU、メモリ、I/O は標準化が行われており、IP 利用が進んでいる。このため、SoC の性能と面積は専用ハードウェアに依存する傾向が強まっている。しかし、このような専用ハードウェアは特定の用途に特化して設計されるため、それぞれの用途別に様々な構成のハードウェアを混載した ASIC 開発が必要となる。また同一の用途でも、新たな機能が次々に登場し、これに合わせて次々に新たな専用ハードウェアを混載する必要が生じている。開発過程も複雑になっており、開発期間の長期化、仕様の変更に即応不可で、設計リスクが高くなっている。これらの状況から、専用ハードウェアを搭載した新しい SoC を次々に市場に投入するための負担は、半導体製造ベンダにとって大きなものとなっている。

こうした問題を解決するために、専用ハードウェアに代わる高い柔軟性を持ったハードウェアがあれば、一種類のチップを様々な用途に用いることができ、新しい機能も簡単に取り入れることが可能となる。開発コストを削減するとともに、単一のチップを大量に生産すればよいことからチップ自体のコストの削減も期待できる。柔軟性を持ったハードウェア自体の構造が決まっていれば、新しいプロセスにも対応することができ、場合によっては製品の配布後もハードウェアの構造を変更することで、発生した問題点や新たな機能付加に対処することができる[1]。

このような柔軟性を持ったハードウェアとして、回路構成を動的に変更することができる動的再構成可能プロセッサが挙げられる。再構成可能デバイスに対して、動的に回路構成を変化させて処理を行うことが可能な動的再構成可能プロセッサは、処理を行いながら最短 1 クロックで回路構成を変更することが可能である。汎用 CPU のような高い柔軟性を持つ一方で、専用ハードウェア並みの高い性能、面積効率を得ることが可能である。動的再構成可能プロセッサは、現在様々な製品に搭載され始めている。代表的なものでは、プリンタなどの複合機に、複合機向け画像処理アクセラレータとして搭載されている。また、放送用ビデオカメラや、携帯機器に消費電力の低い動的再構成可能プロセッサが搭載されている[4]。

一方、現在インターネット上では、ネットワークを通じた文書や画像などのデジタルデ

ータをやり取りする際に，様々な方式で暗号化を行い，情報セキュリティを確保することが重要となっている。ネットワーク上で、通信の暗号化の補助や、ユーザ認証、デジタル署名を行うのが一般的であり、ハッシュ関数はこれらの用途に多く用いられている。ハッシュ関数の中でも代表的なものが、SHA である。

本研究では、大きな計算量を持つ処理としてハッシュ関数である SHA-1, SHA-256 を対象に、動的再構成可能プロセッサを用いて処理の高速化を行えるよう設計・実装を行い、評価と考察を行うことを目的とする。

SHA-1,SHA-256 回路の実装は IP FLEX 社の動的再構成可能プロセッサ DAPDNA-2 を対象に行う。DAP は全体の制御を行うプロセッサであり、制御プログラムを実装する。DNA は回路を再構成可能なハードウェア部分であり、作成した SHA-1, SHA-256 回路を実装する。SHA-1, SHA-256 回路の設計は、最小限のクロック数で行えるように設計を行う。また、配置配線時にレイテンシ調整やデータ受け渡しに関する問題が生じたが、遅延要素を組み込むことで解決した。本研究では、SHA-1 と SHA-256 の回路を、動的再構成を行わない方法で DAPDNA-2 上に実装し、実行時間を計測して、2 コア、4 コアのプロセッサ上での実行時間と比較する。また、SHA-1 を動的再構成を行う方法で実装する方法について提案する。

本論文では、第 2 章において動的再構成可能プロセッサの概要、本研究で用いる動的再構成可能プロセッサ DAPDNA-2 の概要を示す。第 3 章ではハッシュ関数 SHA-1, SHA-256 の概要とアルゴリズムを説明する。第 4 章では SHA-1, SHA-256 の DAPDNA-2 上への DAP、DNA それぞれに対する実装方法を示す。第 5 章で実験方法と実験結果を示し、考察を行う。

2. 動的再構成可能プロセッサ DAPDNA-2

2.1 動的再構成可能プロセッサとは

動的再構成とは，システムの動作中にチップのプログラム内容（回路構成）を変更することで，１チップに複数の異なる処理をさせようとするものである。一般的な LSI は一度構成を決めて製造するとその機能を後から変更することはできない。FPGA（Field Programmable Gate Array）のように機能を変更（再構成）できるデバイスもあるが、その変更は動作時には行えず、回路構成情報を一度ロードしたらリセットするまで回路を変更できない。機能の変更である再構成には、数十 m[sec]の時間が必要となる。再構成デバイスは、再構成によって図１のように汎用 CPU に近い高汎用性と、ASIC に近い高性能を両立することができる。

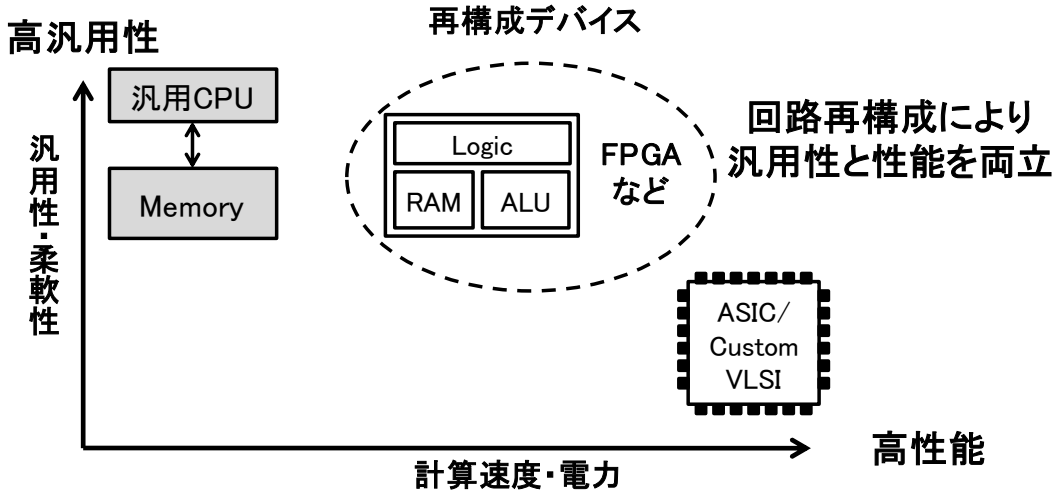

図 1：再構成デバイスの位置づけ

FPGA に対し、動的再構成可能プロセッサ（DRP：Dynamic Reconfigurable Processor）は複数の回路構成情報を用意しておくことで、プログラムの稼働中に回路をダイナミックに再構成可能となり、その際の再構成は最短で１クロックで行うことができる。

動的再構成可能プロセッサは一種類のチップを様々な回路として使用可能なので、回路を実現する場合に回路面積を削減することが可能となる。例えば、複数の機能を実装し、システムの稼働中にそれらの動作モードを切り替える。または、ある単一機能を限られた回路面積の LSI 上に処理のステップ毎に効率的に実装することができる[1][4]。

動的再構成可能プロセッサは汎用 FPGA の発展系というよりは、図 2 のように SoC（System-on-a-chip）におけるアプリケーション専用ハードウェア部の代替製品として、より広い分野で利用されることが期待されている[1]。

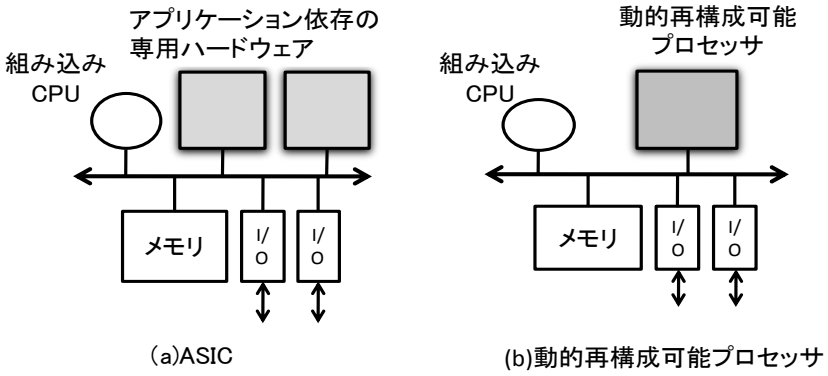

図 2：ASIC と動的再構成可能プロセッサ

FPGA や CPLD は細粒度構成、動的再構成可能プロセッサは粗粒度構成を取っている。FPGA や CPLD は細粒度構成を取っているため構成データの量が多く、また、チップ外のメモリから転送されるため、再構成に要する時間が大きい。動的再構成可能プロセッサは粗粒度で構成されており、これは動的再構成可能プロセッサの特徴の一つである。

FPGA や CPLD といった再構成デバイスは、主に 4 入力 2 出力程度の LUT（Look-up Table）、または AND-OR 接続のプロダクトアーム方式にフリップフロップを設けた基本構成要素を用いていており、基本構成要素のハードウェア量が小さいことから細粒度の再構成デバイスと呼ばれている。細粒度の構成は、演算回路、制御回路など様々な種類の論理回路を無駄なく実現できる点で柔軟性は高いが、メディア処理などで用いられる演算器を実現した場合、専用ハードウェアの数倍程度の面積を要し、動作速度の点で不利である、という問題点が挙げられる。

動的再構成可能プロセッサがとる粗粒度構成方式は、細粒度に比べて柔軟性は低いが、メディア処理などで高速化が必要な演算処理を実行する場合に、面積効率、実行速度の点で有利である。一般に 4 ビットや 8 ビットなどビット幅が小さいほど柔軟性は高いが、演算速度の点で不利である。逆に、32 ビットなどの大きなビット幅を用いると、対象アプリケーションのビット幅とうまく適合すれば高速かつ面積効率が良く、構成データ量も少なくて済む。しかし、対象となる問題のデータが小さい場合には無駄が大きくなる。

粗粒度構の構成要素は、要素プロセッサあるいは PE（Processing Element）と呼ばれ、演算用の PE がアレイ上に配置され、その両側にメモリ用、I/O 用の PE が設けられている[1]。

本論文ではアイピーフレックス社製の DAPDNA-2 という動的再構成可能プロセッサを用いる。DAPDNA-2 は、32 ビットの専用 RISC プロセッサである DAP（Digital Application Processor）と、DNA（Distributed Network Architecture）と呼ばれる並列処理エンジンの大きく 2 つから構成される（図 3）。DAP は基本的には一般的な構造や機能を持つ RISC プロセッサである。DNA は大規模演算を高速に行うための 376 個の PE（Processing

Element）の配列である。DAP と DNA の関係は、SoC における組込み CPU と、それを補佐する高速処理用ハードウェアという関係である。汎用コンピュータにおける CPU とそれを補佐する大規模演算用コプロセッサの関係にも例えられる。PE マトリックスの各 PE の機能を決定する設定値と PE 間の接続情報のセットを DNA コンフィギュレーションと呼び、現在の構成情報の他に 3 セット分の構成情報をオンチップの DNA コンフィギュレーションメモリに保存しておくことができる[1][2]。

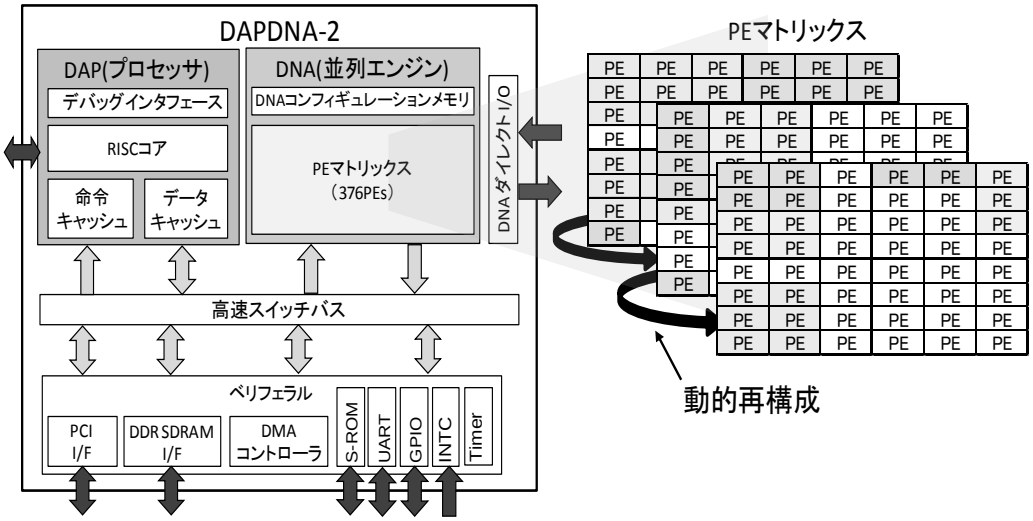

図 3：DAPDNA-2 の構造

DAPDNA-2 は動的再構成の実現にマルチコンテキスト方式を採用している。マルチコンテキスト方式は構成データを記憶する構成データメモリを複数セット持ち、これらをマルチプレクサで切り替え、ロードすることで、一瞬で再構成デバイスの構成を変更する。実行中に次のタスクの回路構成情報を設定することにより、タスク間の待ち時間なしのコンテキストスイッチが可能である。このため、命令/構成データ転送方式で問題であったタスク間の切換えオーバーヘッドをなくすことができる。

図 4 に DNA コンフィギュレーションメモリの構成を示す。DNA コンフィギュレーションメモリはバックグラウンドメモリが 3 バンク、フォアグラウンドメモリが 1 バンクの計 4 バンクからなる。4 つのバンクに 4 種類の DNA コンフィギュレーションを保持できるが、実際に有効になるのはフォアグラウンドメモリにある DNA コンフィギュレーションである。DNA コンフィギュレーションは、命令レジスタを書き換えることで切り替える。

回路構成情報（DNA コンフィギュレーション）は DAP が DNA にロードを行い、DNA が回路を実現する。動的再構成を行う方法は大きく 2 種類ある。1 つ目は、DAP が DNA へ割り込みを行い、DNA コンフィギュレーションをロードする方法である。2 つ目は、DAP が割り込みを行わず、DNA が自律的に DNA コンフィギュレーションを切り替え、ロードする方法である。

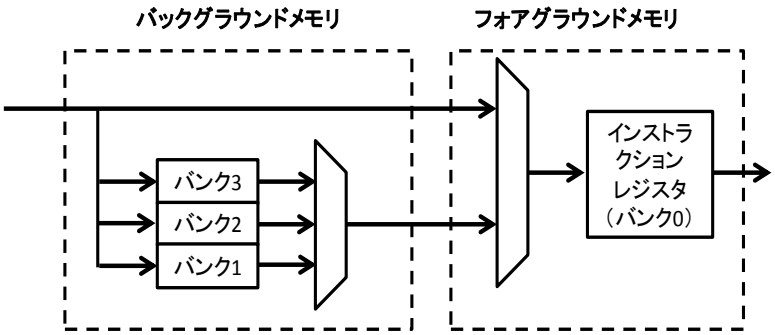

図 4：DNA コンフィギュレーションメモリの構成

2.2 プロセッサ DAP[1]

DAP は 128 種類の命令セットから構成されており、1 命令を 1 クロックで実行することができる 5 段パイプラインステージによるマイクロアーキテクチャを採用している。これは富士通製の 0.13μm CMOS プロセスのスタンダードセルで設計製造し、166MHz の動作周波数を実現している。一般的な ASIC では、高いデータ処理性能を必要とする場合、専用のハードウェアモジュールを集積化して、マイクロプロセッサにより制御する方式を採用する場合が多い。近年では、専用のデータパスを搭載して、専用命令を組み込むことで相当の性能向上をさせるような方法もある。しかしいずれの場合も、開発期間とコストの問題があり、数量が少ない製品や仕様変更の多い製品への採用は難しい。FPGA や PLD などのプログラマブルなハードウェアとの組み合わせが、このような場合の 1 つの選択肢になっている。アイピーフレックス社は、本来の RISC が持つプログラミング言語との親和性を犠牲にせず、高いデータ処理性能を必要とする場合に、直接的に DNA を呼び出すことでこの問題を解決している。

DAPDNA-2 の内部構成を図 5 に示す。図中の矢印はデータの経路を、破線矢印はコンフィギュレーションの経路を示す。

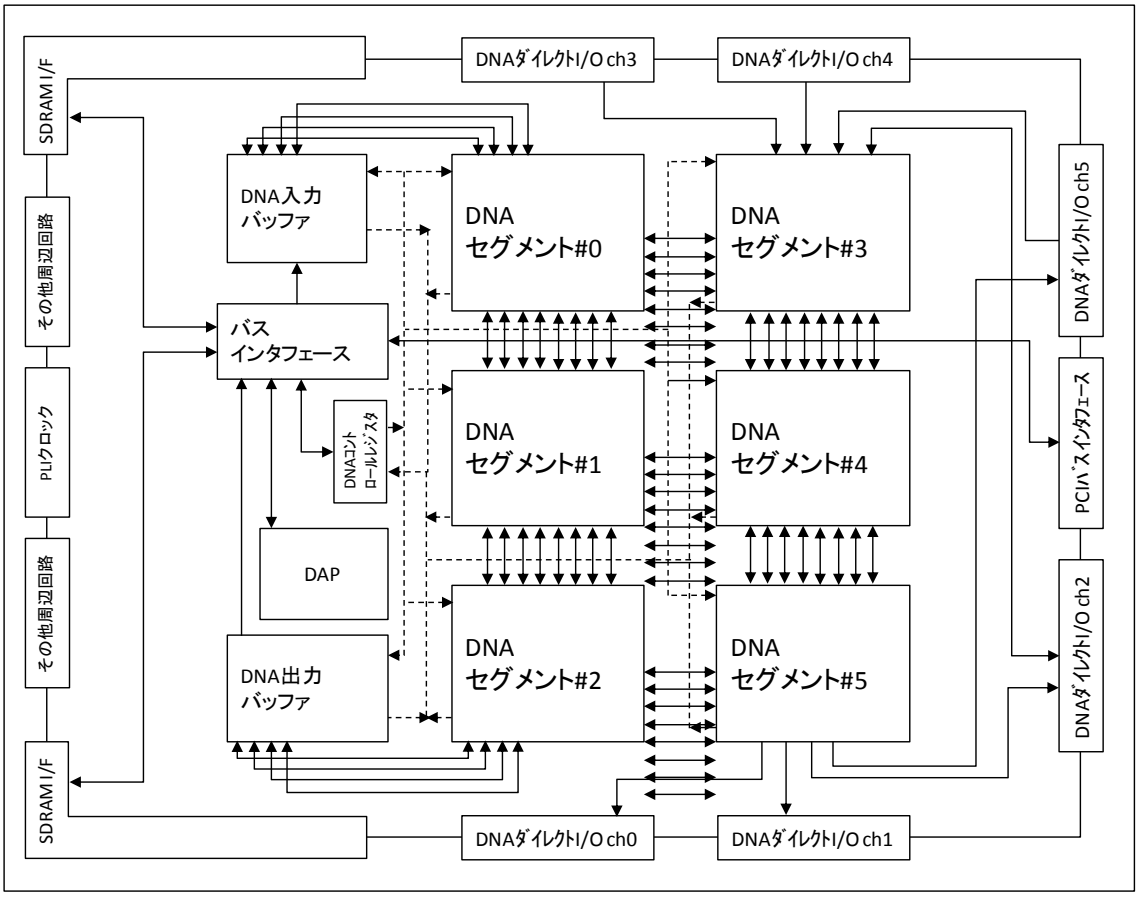

図 5：DAPDNA-2 内部ブロックダイアグラム

DAP は 8K バイトの命令キャッシュと 8K バイトのデータキャッシュ（2way セットアソシアティブ方式）を内蔵している（図 6）。各キャッシュとも 1 サイクルでのアクセスが可能で、キャッシュヒット時は原則としてペナルティ（サイクルロス）は発生せず、キャッシュミス時に外部 DDR-SDRAM メモリアクセス時間と内部のバスアービトレーション分として数サイクル追加される形でペナルティ（ロス）が発生する。全体としては、外部アクセス時間が支配的で、メモリアクセス時間がペナルティとなる。way 数に関しては、競合を回避し、ヒット率を上げるために 4way や 8way の方式を採用するものが多いが、タグ領域の検索時間によるアクセススピードの低下や、面積的にもタグ領域が増加し、特にスタンダードセルと汎用 SRAM による設計上の制約も考慮し、2way 方式を選択した。組込みの分野では、そのペナルティ差はコストとのトレードオフで十分引き合うとの判断である。内部キャッシュは高速バッファとして、内部アクセス専用に使うように設定することができる。これにより、あまり大規模でないアプリケーションでは、チップ内蔵メモリを利用した高速アクセスが可能となる。

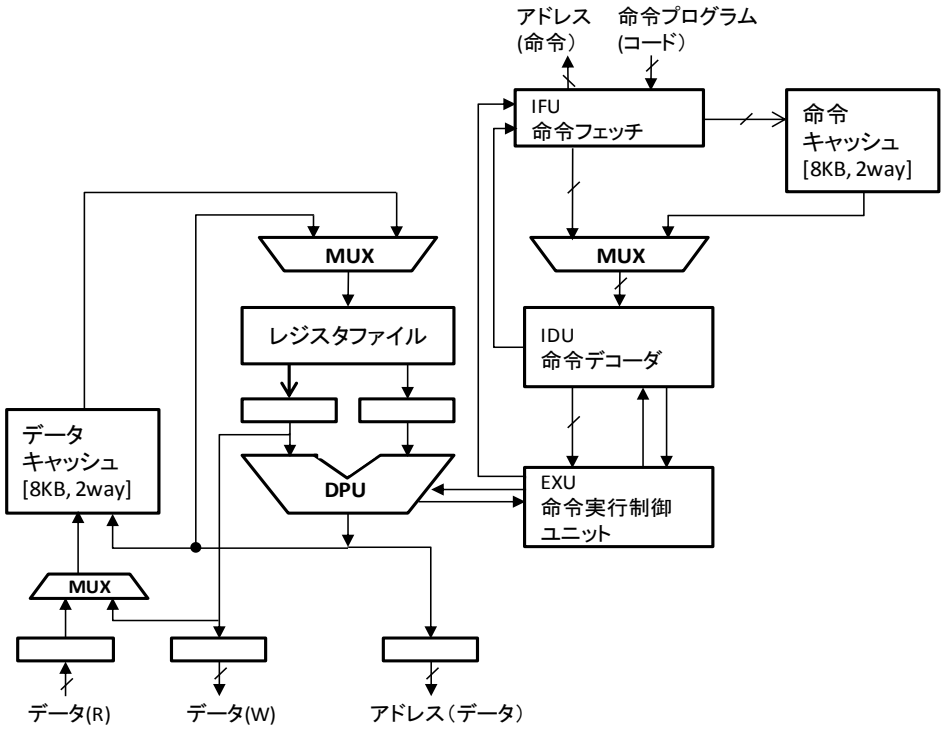

図 6：DAP ブロックダイアグラム

DAP と DNA とのインターフェースは、割り込み処理とステータスレジスタをチェックする、いわゆるフラグセンス方式である。また、DNA コンフィギュレーション（構成データ）の設定や PE の部分的な変更を DAP 側から可能としている。DNA 側の演算結果や中間データなどをやり取りするオーバーヘッドが専用プロセッサという意味で予想以上に大きいという指摘があり、複数の構成情報の変更が連続するようなアプリケーションに対しても対応可能なアーキテクチャへの改良を今後施していく必要がある。

DAP は DAP コンパイラでコンパイルされたプログラム通りに動作する。図 7 に DAPDNA-2 プログラムのデザインフローを示す。プロジェクトを作成する際は、まず仕様とアルゴリズムを検討し、DAPDNA-2 全体の制御プログラムと、DNA 上で回路を構成する DNA プログラムの 2 つに別れて設計を行う。DAP プログラムは主に DAPDNA-2 全体の制御プログラムから成る。DNA への割り込みなども実行可能であり、DNA へ割り込みを行うことで、回路の動的再構成を行う。DAP プログラムは C 言語で記述を行う。DAP コンパイラは DAPDNA-2 用に移植された GCC(GNU Compiler Collection)である。DAP プログラムと DNA プログラムから生成されるプロジェクトのソースコードは DAP コンパイラによってコンパイルされ、DAP ライブラリとリンクされ、実行ファイルが生成される。DNA プログラムは、DNA コンフィギュレーションと呼ばれる回路構成情報である。DNA コンフィギュレーションは DAPDNA-2 専用の DNA Designer という GUI ツールを用いて設計を行い、DNA コンパイラによって物理位置に配置配線を行う。

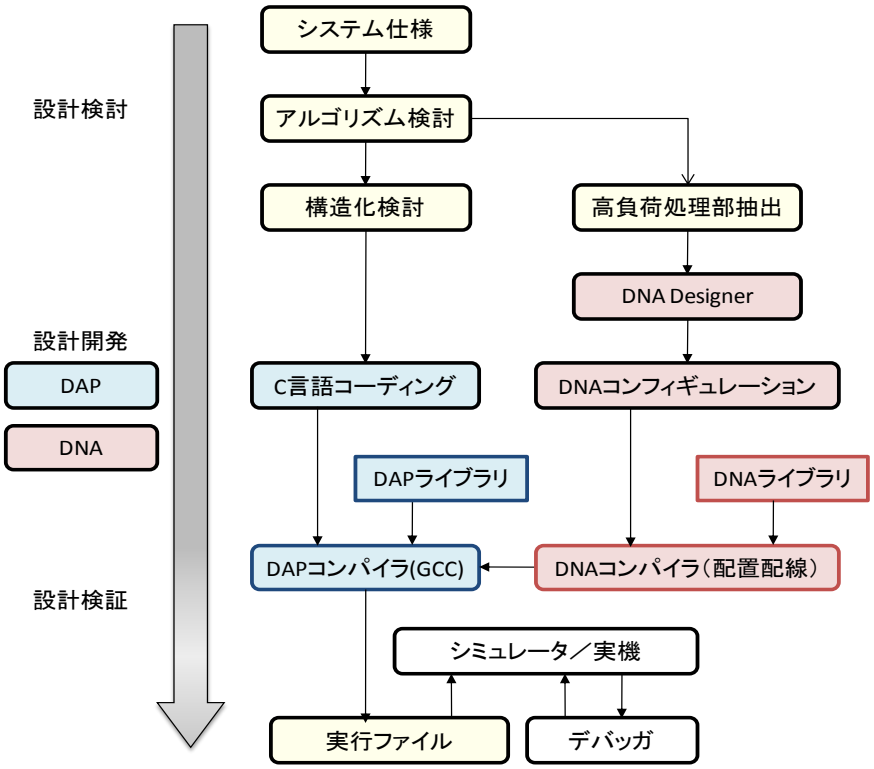

図 7：DAPDNA-2 アプリケーション・プログラム　デザインフロー

2.3　並列エンジン DNA[1]

DNA は構成を動的に変更可能な PE マトリックスとその構成情報を記憶する DNA コンフィギュレーションメモリから構成される。DAPDNA-2 の PE マトリックスは 376 個のプロセッシングエレメント(PE)から構成され、それぞれ接続指定と機能定義が可能である。データフロー型の処理をパイプライン処理と同時に、メインメモリからのデータの読み出し（入力）や格納（出力)、内部に持つ複数の高速メモリのアクセスが可能である。実行中の構成情報のほかに、バックグラウンドとして 3 つの再構成情報を DNA コンフィギュレーションメモリに保存可能である。この再構成情報のフォアグラウンドへのロードは、1 クロックサイクルで実現することができる。データ転送のバンド幅に余裕があれば、データ処理を一切停止せずに将来必要となる再構成データを内部にダウンロードできる。また、DAPDNA-2 はデバイス間で最大 32Gbps の広帯域でのデータ転送が可能である。1 つの DAPDNA-2 での実装が難しい場合、時分割せずに複数のデバイスへ分割実装が可能である。アプリケーションの並列度にもよるが、166MHz 動作の DAPDNA-2 は数十 GHz 以上のクロック周波数で動作するマイクロプロセッサや DSP に相当する性能の実現に成功している。

2.3.1 PE マトリックス

PE マトリックスの内部構成を図 8 に示す。PE マトリックスには主にデータ処理にしようされる Execution エレメント（EXM/EXC/EXF/EXR/EXS）、Delay エレメント（DLE/DLV/DLX/DLH）、RAM エレメントと、データ入出力用のエレメント（LDB/STB/, LDX/STX、C32/C16）から構成される（表 1）。

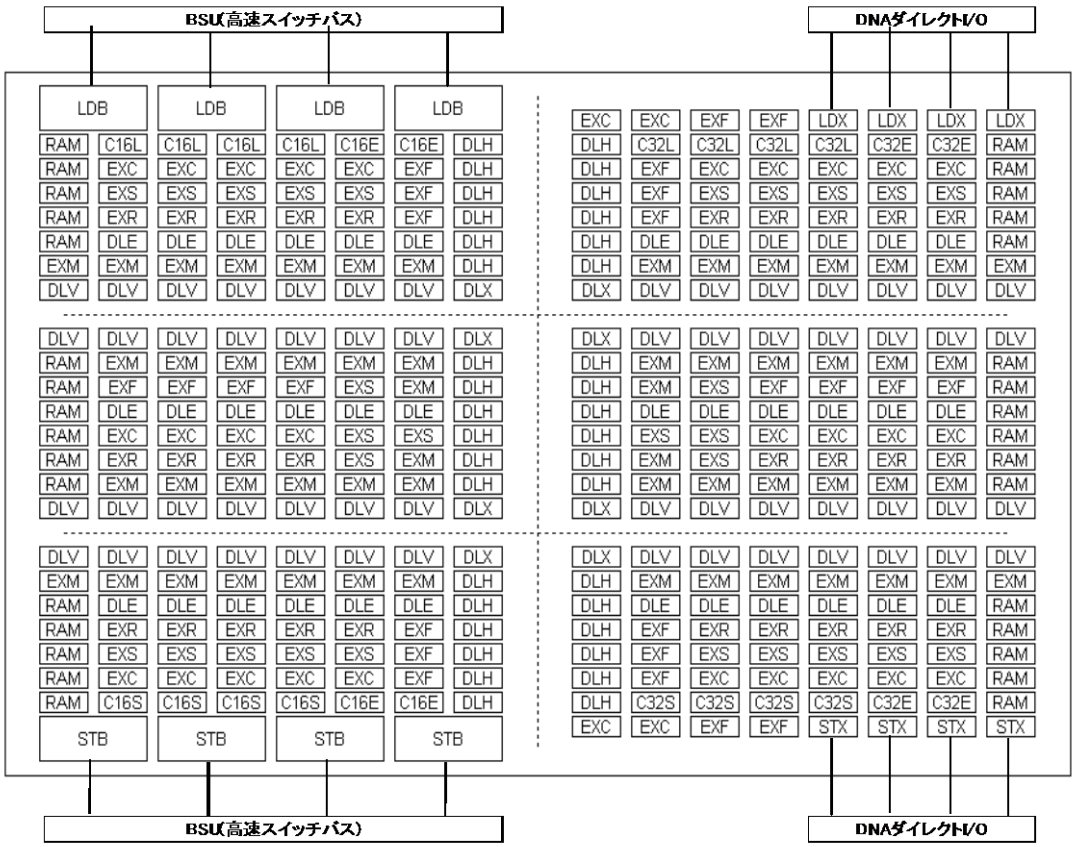

図 8 : PE マトリックス

表 1 : PE マトリックスの機能

データ処理部のバス幅	32bit データ + 2bi トークン + 2bit キャリー
全 PE 数	376 個
演算系 PE 数	168 個
乗算可能 PE 数(EXM)	56 個（16bit×16bit　32bit 出力）
メモリ PE 数(RAM)	32 個（32bit×4Kword）
ロードバッファ(LDB)	4 個（32bit×2Kword+2bank）
ストアバッファ(STB)	4 個（32bit×2Kword+2bank）
DNA ダイレクト I/O	32bit×6ch

PE マトリックスは 32 ビットのデータ処理用の演算器とメモリを中心に配置された 2 次元配列型のデータパスである。データ処理は並列処理が可能となる、アプリケーションのアクセルレータとして機能する。このPEマトリックスは6セグメントから構成されており、全体として 376 個の PE から成る。1 つのセグメントは 64 個（8x8）の PE から構成されており、セグメント内部では各 PE は自由に接続することができる。しかし、セグメント間の接続は必ずセグメント境界の PE を介して次のセグメントにデータを渡すという制限がある。したがって、複数のセグメントにまたがって PE を接続するような回路では、セグメント境界において発生するレイテンシを吸収するための同期調整が必要である。また、6 セグメント内にアルゴリズムが実装できないほど大きなアプリケーションを扱う場合は、空間方向もしくは時間方向に分割する必要がある。すなわち、複数の DAPDNA によるマルチチップシステムで並列パイプライン処理実行にするか、シングルチップでマルチ DNA コンフィギュレーションによる時分割多重実行にするか、コストと性能をトレードオフしたうえでアルゴリズムを分割する。コンパイラは、セグメント間の同期調整を含めて 6 セグメントへの配置敗戦を自動的に行う。

PE マトリックスの各 PE の機能を決定する設定値と PE 間の接続情報のセットを DNA コンフィギュレーションと呼び、現在の構成データの外に 3 セット分の構成データをオンチップのコンフィギュレーションメモリに保存しておくことができる。内部に保存されている構成データは、1 クロックサイクルで呼び出し可能であるが、新しい構成データを内部にロードする場合、マイクロ秒オーダの時間が必要となる。したがって、アプリケーション設計ではデータ処理実行時にバックグラウンドで構成データを事前にロードしておくことで、性能ロスの防止が可能となる。構成データは暗号化可能なパケット構成としており、開発環境から暗号化するかどうかの指定が可能となる。

PE マトリックスには BSU と呼ばれる高速バススイッチを経由してメインメモリ（DDR-SDRAM）へアクセスするインターフェースと、DNA ダイレクト I/O を通して、直接外部の DAPDNA-2 デバイスやその他、高速デバイスとの接続が可能な DNA ダイレクト I/O インターフェースとが装備されている。この DNA ダイレクト I/O は、1 チャネル 32 ビットの高速入出力で、入力専用 2 チャネル、出力専用 2 チャネル、入出力定義可能チャネル 2 チャネルの合計 6 チャネルを実装している。データ転送バンド幅はトータルで 32Gbps となり、ダイナミックリコンフィギャラブルプロセッサとしては他に類を見ない転送能力を持つ。

2.3.2 エレメントの構成

（1）EXE（Execution Element）

Execution エレメントはデータ処理を専用に実行する PE で、データ、キャリーそれぞれ 2 入力、1 出力のポートを持ち、25 種類の算術演算、論理演算、および入力データに対するシフトやマスクなどのビット演算機能を持っている。入力データにビット演算等

の前処理を行うプリプロセッシングステージと、それらのデータの演算を主に行う。ALU ステージが直列に連結されてパイプラインを構成しており、入力から出力までのレイテンシは 2 から 4 クロックと可変である。

Execution エレメントは、EXM, EXC, EXF, EXR, EXS の 5 種類に分類され、各々が固有の演算機能を持っている。

EXM：乗算命令（下位 16 ビットを乗算し、32 ビットの結果を出力する）

EXC：バイト比較（対応する各バイトごとに比較を行う）

EXF：ファインド ファースト 1 命令（入力を上位ビットから検索し、最初に“1”を見つけたビット位置を出力する）

EXR：ビットリバース命令（入力データをビット単位で上位と下位を入れ替える）

EXS：バイトスワップ命令（入力データをバイト単位で上位と下位を入れ替える）

（2）DLE（Delay Element）

Delay エレメントは、データとキャリーそれぞれ 2 入力、2 出力の計 4 ポート構成で、2 つの入力に対して独立に遅延量を設定可能である。おもにレイテンシの調整とセグメント間のデータ受け渡しのために使用される PE で、DLE, DLH,DLV, DLX の 4 種類がある。

DLE：セグメント内部の PE 間の同期調整を行う。遅延量は 1-7 クロック可変。

DLH：横隣のセグメント間の接続を行う。遅延量は 2 クロック固定。

DLV：縦隣のセグメント間の接続を行う。遅延量は 2-8 クロック可変。

DLX：DLV とほぼ同じ。ただし、チェーン接続は横隣のセグメントの DLX と行う。

（3）RAM

RAM エレメントは PE マトリックス内のデータを蓄積するためのメモリで、16K バイトの SRAM で構成される。アドレス入力からリードデータの出力までのレイテンシは 3 クロックである。PE マトリックスは 32 個の RAM エレメントを持っている。RAM エレメントはワーキングバッファ/FIFO/ルックアップテーブルなどの用途で使用される。

（4）入出力用エレメント

PE マトリックスのデータ入出力は専用の入出力エレメントを介して行われる。PE マトリックスへデータを入力するには、メインメモリから高速バススイッチユニットを介して LDB にデータを入力する方法と、DNA ダイレクト I/O を接続された外部デバイスから LDX にデータを入力する方法の 2 種類がある。同様に、PE マトリックスからデータを出力する方法には、STB から高速バススイッチユニットを介してメインメモリにデータを出力する方法と、STX から DNA ダイレクト I/O と接続された外部デバイスデータを出力する方法がある。

（5）LDB と STB

LDB エレメントはメインメモリから PE マトリックスへのデータの読み出しを行うための入力バッファで、1 ポート RAM でメインメモリからの読み出しと PE マトリックスへの読み出しを同時に実行可能とするダブルバッファ構成を採用している。一方、STB エレメントは PE マトリックスの演算結果をメインメモリへ書き込みを行うための出力バッファで、LDB と同様にダブルバッファ構成を採用し、連続的にメインメモリへデータの転送を可能としている。

（6）LDX と STX

LDX エレメントは DNA ダイレクト I/O に接続された外部デバイスや DAPDNA-2 から PE マトリックスにデータの連続読み込みをするための入力バッファである。一方、STX エレメントは PE マトリックスから DNA ダイレクト I/O 経由で直接外部デバイスや DAPDNA-2 へデータ出力を行う。

（7）C32（C32L/C32S/C32E）, C16（C16L, C16S, C16E）

C32 エレメントは 32 ビットのカウンタ 2 個を組み合わせたカウンタエレメントである。その中でも特に C32L は、メインメモリのリードアドレス、C32S はメインメモリのライトアドレスを生成するアドレスジェネレータとして動作する。C16 エレメントは 16 ビットのカウンタ 4 個を組み合わせたカウンタエレメントである。その中でも特に C16L は、LDB に対するリードアドレス、SDB や STB に対するライトアドレスを生成するアドレスジェネレータをして動作する。

3. ハッシュ関数のアルゴリズム

3.1 ハッシュ関数とは

ハッシュ関数とは与えられた原文から固定長の擬似乱数を生成する演算手法であり、生成した値はハッシュ値、または要約関数、メッセージダイジェストと呼ばれる。通信回数を通じてデータを送受信する際に、往路の両端でデータのハッシュ値を求めて両者を比較すれば、データが通信途中で改ざんされていないか調べることができる。不可逆な一方向関数を含むため、ハッシュ値から原文を再現することはできず、また同じハッシュ値を持つ異なるデータを作成することは極めて困難である。ハッシュ関数は、通信の暗号化の補助や、ユーザ認証、デジタル署名などに使われている。現在広く用いられているのは出力するハッシュ値が 160bits の SHA-1（Secure Hash Algorithm 1）である。SHA-1 は Draft FIPS 180-1[10]で提案された。

Draft FIPS-180-2[11]では、224bits, 256bits, 384bits, 512bits のハッシュ値を出力するハッシュ関数 SHA-224, SHA-256, SHA-384, SHA-512 が提案されている。

各 SHA は、衝突困難かつ一方向性であるように設計されている。また、同時に入力メッセージの少しの変化が出力のハッシュ値に大きく影響するように設計されている。SHA-224, SHA-256, SHA-384, SHA-512 をデジタル署名、鍵付きハッシュによるメッセージ認証、疑似乱数生成器として使用するためには、これらの設計目標が達成されていなければならない。これらの設計目標が達成されていないという証拠はないが、逆に設計目標が達成されているという数学的な証明もないという現状に注意する必要がある。

各 SHA の構造は Merkle-Damgård が提案した衝突困難なハッシュ関数の構成法に準じている。図 9 のようにメッセージをいくつかのブロックに分割し、各ブロックを圧縮関数 f により繰り返し処理をする構造になっている。この構成法の特徴は、ハッシュ関数全体の衝突困難性が圧縮関数 fの衝突困難性に帰着されることである。

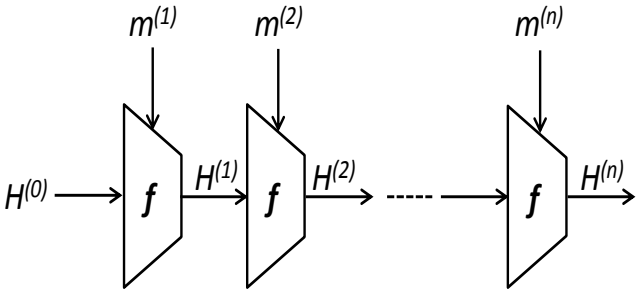

図 9：Merkle-Damgård のハッシュ関数の構成法

表 2 に 5 つのアルゴリズムの基本的な機能を示す。5 つのアルゴリズムは、ハッシュ生成に使うブロックサイズとワードサイズ、出力ハッシュサイズが異なる。

表 2：SHA の機能

Algorithm	メッセージサイズ (bits)	ブロックサイズ (bits)	ワードサイズ (bits)	出力ハッシュサイズ (bits)
SHA-1	$<2^{64}$	512	32	160
SHA-224	$<2^{64}$	512	32	224
SHA-256	$<2^{64}$	512	32	256
SHA-384	$<2^{128}$	1024	64	384
SHA-512	$<2^{128}$	1024	64	512

3.2 SHA-1

SHA-1 は、1993 年に設計された SHA-0 の規格修正版として、1995 年に米国家安全保障局(NSA)が再設計し、NIST が規格化した米国政府標準ハッシュ関数（FIPS 180-1［10］）である。ハッシュ長は 160 ビットである。1995 年に米国標準技術局（NIST）によってアメリカ政府の標準ハッシュ関数をして採用された。インターネット上で安全に通信を行うための IPSec などに応用されている。

2 の 64 乗ビット以下の原文から 160 ビットのハッシュ値を生成し、通信経路の両端で比較することで、通信途中で原文が改ざんされていないかを検出することができる。計算方法には不可逆な一方向関数を含むため、ハッシュ値は擬似的な乱数のような値をとり、これをもとに原文を再現することは出来ない。また、同じハッシュ値を生成する別のメッセージを作成することも極めて困難である。図 10 は SHA-1 の処理全体の流れである。以下に SHA-1 のアルゴリズムを述べる。

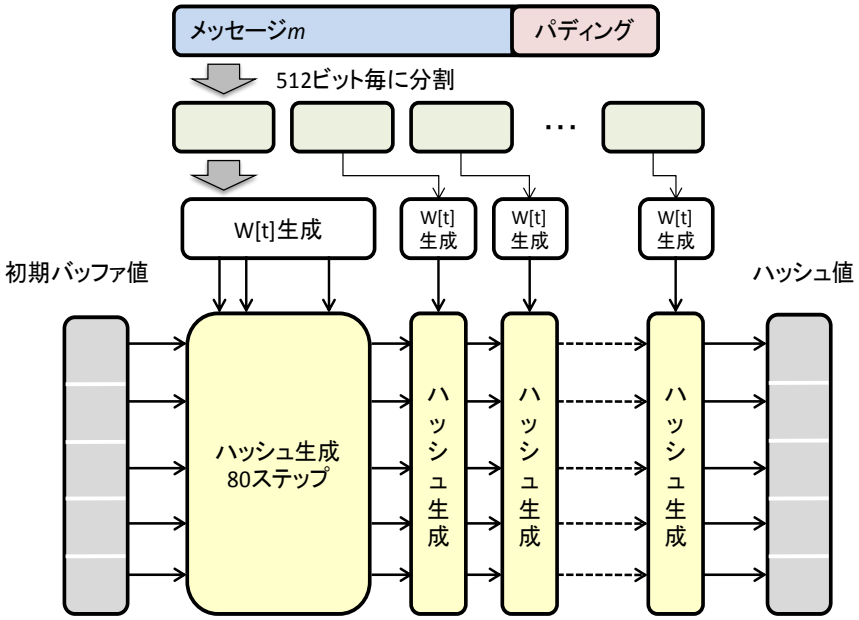

図 10：SHA-1 の処理全体フロー

（1）使用する関数の定義

関数を次のように定義する。変数のビット長は 32 ビットである。

$$ft(x, y, z) = \begin{cases} Ch(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z) & 0 \le t \le 19 \\ Parity(x, y, z) = x \oplus y \oplus z & 20 \le t \le 39 \\ Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) & 40 \le t \le 59 \\ Parity(x, y, z) = x \oplus y \oplus z & 60 \le t \le 79 \end{cases}$$

$ROTL^n(x)$ ： x を n bits 左へ巡回シフト

（2）前処理

ハッシュ生成を行う前にメッセージmをパディングする。パディングとは入力されるメッセージをある数の倍数の長さに変換することであり、SHA-1 の場合 512 の倍数の長さのビット系列に変換する。表 2 おけるブロックサイズのビット長の倍数の長さ、512 ビットの倍数長にメッセージ長を整える。

SHA-1 はメッセージ要約値を計算する際に 512 ビットのブロックごとに順番に計算する。 以下に、このパディングの処理内容を述べる。

（ｉ）パディング

メッセージmは512を法として448と等しいビット長になるようパディングされる(長さ＝448 mod 512)。つまり、パディングが施されたメッセージの長さは、512 ビットの整数倍と言うよりは 64 ビットの倍数となる。たとえメッセージmがすでに望ましい長さを持っている場合であっても、常にパディングは行われる。パディングに用いられるビット数の範囲は 1 から 512 である。パディングは 1 つの 1 のビットと、その後に続く必要な数の 0 のビットからなる。

例えば、図 11 のように 8 ビット ASCII 文字のメッセージ m“abc”の長さが 8*3=24 の場合、まず 1 ビットの“1”を付加する。次に、448 – (24 +1) = 423 により 423 個の“0”を付加する。

（ⅱ）長さを付加する

64 ビットのブロックがメッセージに付け加えられる。このブロックは符号なしの 64 ビットの整数として扱われ、(最上位のバイトが最初にくる）パディングが施される前の元のメッセージの長さを表す。

図 11 で言うと、メッセージmの長さ 24 を下位 64 ビットに付加する。これで、合計の長さが 512 ビットとなる[11]。

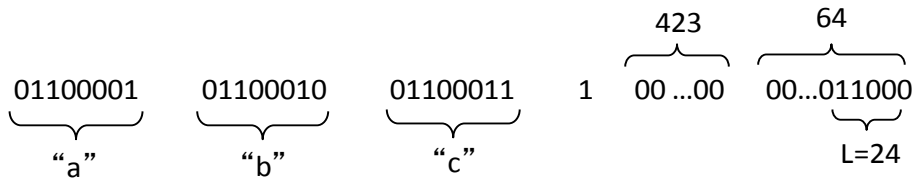

図 11：前処理

要約すれば、64 ビットの整数、m 個の "0"、1 個の "1" が 512n 長のメッセージにするためにその末尾に追加される。 64 ビットの整数は l、元のメッセージの長さである。 パディングされたメッセージはこの後に n 個の 512 ビットブロックとして SHA-1 によって処理される。

（3）ワード W[t]の生成

m(i)を 32bits 毎に分割し、それらを m[t]（0≦t≦15）と書く。そして、W[t]を以下のように計算する。

$$W[t] = \begin{cases} m[t] & 0 \le t \le 15 \\ ROTL^{1}(W[t-3] \oplus W[t-8] \oplus W[t-14] \oplus W[t-16]) & 16 \le t \le 79 \end{cases}$$

（4）定数 K[t]の準備

SHA-1 は 80 個のワードで、ハッシュ生成の入力定数となる K[0], K[1], …, K[79]を使用する。これらは 32 ビットの 16 進数で次の定数である。

$$K[t] = \begin{cases} 5a827999 & 0 \le t \le 19 \\ 6ed9eba1 & 20 \le t \le 39 \\ 8f1bbcdc & 40 \le t \le 59 \\ ca62c1d6 & 60 \le t \le 79 \end{cases}$$

（5）初期ハッシュ値を用意する。数値は 32bit の 16 進数である。

$H_0^{(0)} = 67452301$

$H_1^{(0)} = efcdab89$

$H_2^{(0)} = 98badcfe$

$H_3^{(0)} = 10325476$

$H_4^{(0)} = c3d2e1f0$

（６）変数を初期化する。

a[0] = $H_0^{(0)}$

b[0] = $H_1^{(0)}$

c[0] = $H_2^{(0)}$

d[0] = $H_3^{(0)}$

e[0] = $H_4^{(0)}$

（7）ハッシュ生成

t=0, 1, ... , 79 に対して以下の計算を繰り返す。1 つの t に対する以下の処理をまとめて、1 ステップと呼ぶ。図 12 にハッシュ生成を示す。

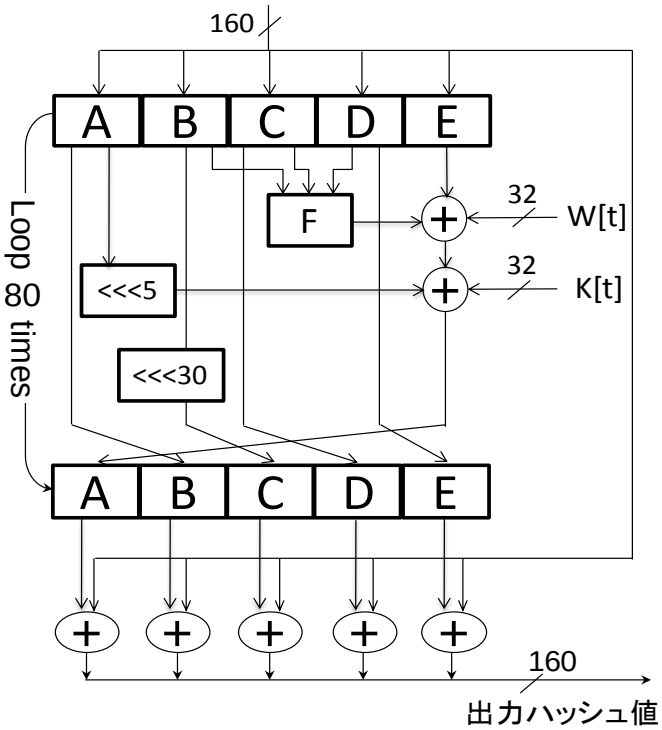

図 12：SHA-1 のハッシュ生成部

TEMP = ROTL ^{5}(a[t])) + ft(b[t], c[t], d[t]) + e[t] + W[t] + K[t]

a[t+1] = TEMP

b[t+1] = a[t]

c[t+1] = ROTL ^{30} (b[t])

d[t+1] = c[t]

e[t+1] = d[t]

（８） SHA-1 のハッシュ値を計算する。

80 ステップ後の値と初期ハッシュ値を加算して、最終的なハッシュ値を求める。

$$H_0^{(i)} = a[79] + H_0^{(i-1)}$$
$$H_1^{(i)} = b[79] + H_1^{(i-1)}$$
$$H_2^{(i)} = c[79] + H_2^{(i-1)}$$
$$H_3^{(i)} = d[79] + H_3^{(i-1)}$$
$$H_4^{(i)} = e[79] + H_4^{(i-1)}$$

3.3 SHA-256

SHA-256、SHA-384、SHA-512 は 2002 年に、SHA-224 は 2004 年に米国家安全保障局(NSA)が設計し、NIST が規格化した米国政府標準ハッシュ関数（FIPS 180-2［11］）である。ハッシュ長は順に 256 ビット、384 ビット、512 ビット、224 ビットであり、MD5/SHA-1 同様、メッセージ拡張関数と圧縮関数とからなる構造をしている。

この 4 つのハッシュ関数を区分する場合、ハッシュ長による違いというよりも利用環境での CPU 基本演算系の違いという側面の方が大きい。つまり、32 ビット演算系 CPU での利用を前提とする場合には SHA-224 と SHA-256 を、64 ビット演算系 CPU での利用を前提とする場合には SHA-384 と SHA-512 を使うことが想定されている。

また、SHA-224 と SHA-384 は、（初期ベクトル値は異なるものの）それぞれ SHA-256 または SHA-512 と同様の演算をした後、それらのハッシュ値の一部（下位 32 ビットまたは 128 ビット）を切り捨てる形でハッシュ長を短くしているだけである。従って、処理性能では、SHA-224 と SHA-256、SHA-384 と SHA-512 はほとんど変わらない。図 13 は SHA-256 の是処理全体の流れである。

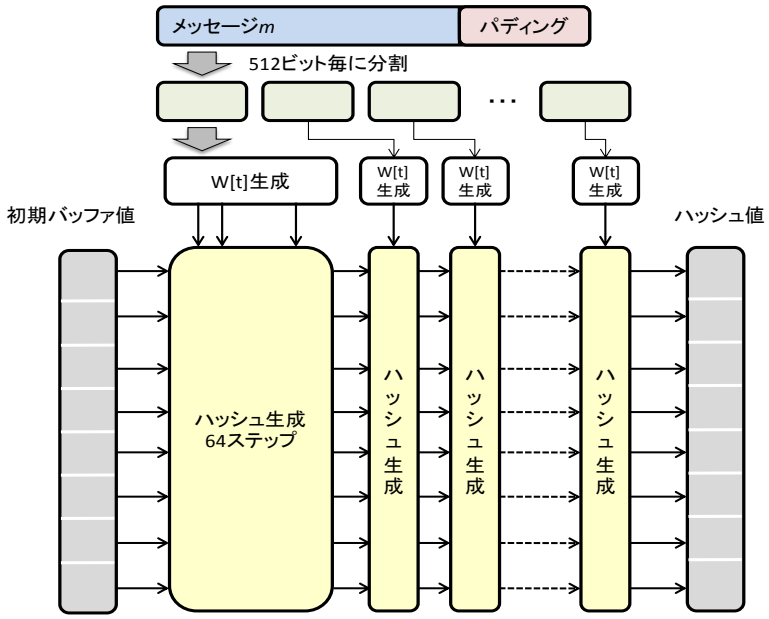

図 13：SHA-256 の処理全体フロー

以下に SHA-256 のアルゴリズムを述べる

（1）使用する関数の定義

関数を次のように定義する。変数のビット長は 32bits である。

$SHR^n(x)$ ： x を n bits 右へ論理シフト

$ROTR^n(x)$ ： x を n bits 右へ巡回シフト

（2）前処理

SHA-256 の前処理は SHA-1 の前処理（3.2（２））と同様である。パディングとビット長の付加を行う。

（3）ワード W[t]の生成

m(i)を 32bits 毎に分割し、それらを m[t]（0≦t≦15）と書く。そして、W[t]を以下のように計算する。

$$W[t] = \begin{cases} m[t] & 0 \le t \le 15 \\ \sigma_1^{\{256\}}(W[t-2]) + W[t-7] + \sigma_0^{\{256\}}(W[t-15]) + W[t-16] & 16 \le t \le 63 \end{cases}$$

ここで、

$$\sigma_0^{\{256\}}(x) = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x)$$

$\sigma_1^{\{256\}}(x)$ = $ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x)$

である。

（4）定数 K[t]の準備

SHA-256 は 64 個の 32bit のワード、K[0], K[1], ..., K[63] を使用する。これらは 16 進数で次の定数である。

```
428a2f98 71374491 b5c0fbcf e9b5dba5 3956c25b 59f111f1 923f82a4 ab1c5ed5
d807aa98 12835b01 243185be 550c7dc3 72be5d74 80deb1fe 9bdc06a7 c19bf174
e49b69c1 efbe4786 0fc19dc6 240ca1cc 2de92c6f 4a7484aa 5cb0a9dc 76f988da
983e5152 a831c66d b00327c8 bf597fc7 c6e00bf3 d5a79147 06ca6351 14292967
27b70a85 2e1b2138 4d2c6dfc 53380d13 650a7354 766a0abb 81c2c92e 92722c85
a2bfe8a1 a81a664b c24b8b70 c76c51a3 d192e819 d6990624 f40e3585 106aa070
19a4c116 1e376c08 2748774c 34b0bcb5 391c0cb3 4ed8aa4a 5b9cca4f 682e6ff3
748f82ee 78a5636f 84c87814 8cc70208 90befffa a4506ceb bef9a3f7 c67178f2
```

（5）初期ハッシュ値を用意する。数値は 32bit の 16 進数である。

$H_0^{(0)}$ = 6a09e667 , $H_1^{(0)}$ = bb67ae85

$H_2^{(0)}$ = 3c6ef372 , $H_3^{(0)}$ = a54ff53a

$H_4^{(0)}$ = 510e527f , $H_5^{(0)}$ = 9b05688c

$H_6^{(0)}$ = 1f83d9ab , $H_7^{(0)}$ = 5be0cd19

（6）変数を初期化する。

$a[0] = H_0^{(0)}$, $b[0] = H_1^{(0)}$

$c[0] = H_2^{(0)}$, $d[0] = H_3^{(0)}$

$e[0] = H_4^{(0)}$, $f[0] = H_5^{(0)}$

$g[0] = H_6^{(0)}$, $h[0] = H_7^{(0)}$

（7）ハッシュ生成

t=0, 1, ... , 63 に対して以下の計算を繰り返す。1 つの t に対する以下の処理をまとめて、1 ステップと呼ぶ。図 14 にハッシュ生成を示す。

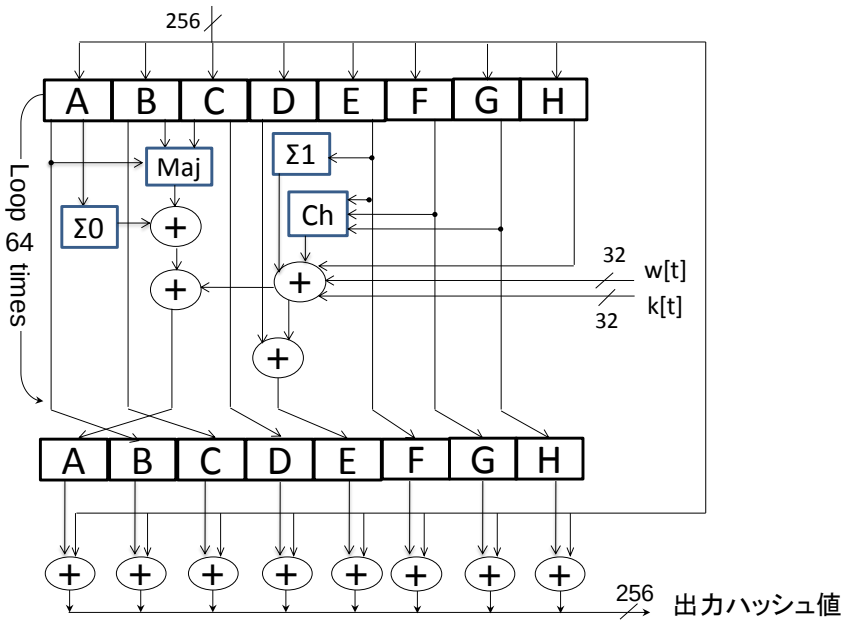

図 14：SHA-256 のハッシュ生成部

$T_1[t] = h[t] + \sum_1^{\{256\}}(e[t]) + Ch(e[t], f[t], g[t]) + K[t] + W[t]$

$T_2[t] = \sum_0^{\{256\}}(a[t]) + Maj(a[t], b[t], c[t])$

$a[t+1] = T1[t] + T2[t]$

$b[t+1] = a[t]$

$c[t+1] = b[t]$

$d[t+1] = c[t]$

$e[t+1] = d[t] + T1[t]$

$f[t+1] = e[t]$

$g[t+1] = f[t]$

$h[t+1] = g[t]$

ここで、

$Ch(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z)$

$Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$

$\sum_0^{\{256\}}(x) = ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x)$

$\sum_1^{\{256\}}(x) = ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x)$

である。

(8)　SHA-256 のハッシュ値を計算する。

64 ステップ後の値と初期ハッシュ値を加算して、最終的なハッシュ値を求める。

$$H_0^{(i)} = a[63] + H_0^{(i-1)} \quad , \quad H_1^{(i)} = b[63] + H_1^{(i-1)}$$

$$H_2^{(i)} = c[63] + H_2^{(i-1)} \quad , \quad H_3^{(i)} = d[63] + H_3^{(i-1)}$$

$$H_4^{(i)} = e[63] + H_4^{(i-1)} \quad , \quad H_5^{(i)} = f[63] + H_5^{(i-1)}$$

$$H_6^{(i)} = g[63] + H_6^{(i-1)} \quad , \quad H_7^{(i)} = h[63] + H_7^{(i-1)}$$

4. ハッシュ関数の DAPDNA-2 上での実装

SHA-1,SHA-256 の処理を行う回路（DNA コンフィギュレーション）を作成し、DAPDNA-2 上に実装した。DNA 上に実装する DNA コンフィギュレーションは、PE のパラメータ情報、接続情報を GUI ソフトウェアで設計し、それを DNA コンパイラに通して生成する。

GUI での設計方法は、GUI で PE を配置し、それらの機能を設定、接続を行うという手順である。また、必要に応じてレイテンシの調節と、セグメント間のデータの受け渡し用 PE の配置を意図的に行う必要がある。

次に、この章で用いる用語を定義する。

セグメント：2.3.1 の通り、DNA 内の PE マトリックスは、一定の動作周波数を保つため、図 5, 図 8 のように 6 つの領域（セグメント）に分割されている。1 つの領域のことをセグメントと呼び、セグメント 0 からセグメント 5 までの 6 つのセグメントが存在する。1 つのセグメントは 64（8×8）個の PE から構成されており、全体で 376 個の PE を持つ。

インスタンス：DNA 上に実装する回路構成情報は、DAPDNA-2 の専用 GUI で図 15 や図 16 のように記述を行う。基本的に 1 つのブロックが 1 つの PE を示すが、記述方法によっては 1 つのブロック内に複数の PE が含まれているものが存在する。これは、複数の PE を接続した回路を 1 つのグループとして、1 つのブロックで表現しているものである。このように、グループ化している 1 つのブロックをインスタンスと呼ぶ。

レイテンシの調節が必要な場合の例を挙げる。1 クロック毎に変化する入力データを a, b, c、出力データを d として d=a+b+c の加算を行う場合、演算系 PE は 2 入力 1 出力より、一度に 1 つの加算のみ行うことができる。よって専用 GUI でこの加算を実現する際には、図 15 のようにまず a+b を行い、次に(a+b)+c のように、加算を 2 度行って d を求める。図中において、接続された 1 つの四角形が 1 つの PE を現している。また、加算には 2 クロックかかるので、加算②の入力 c は、入力(a+b)のタイミングを合わせるために 2 クロックの遅延を与えなければならない。このように、PE への入力データのタイミングはディレイ PE を追加で配置して合わせる必要がある。

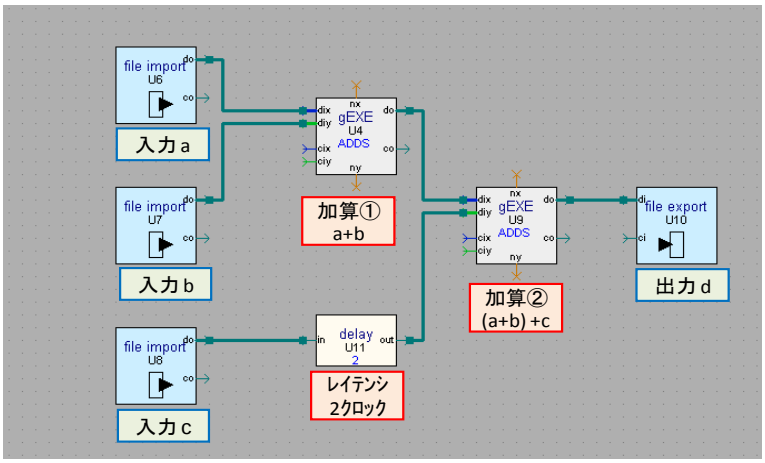

図 15：3 加算の方法

次に、セグメント間のデータの受け渡し用 PE の配置が必要な例を挙げる。設計した回路を DNA コンパイラでコンパイルを行うと、自動で配置配線を行い、セグメント間のデータ受け渡し用の PE も回路の演算内容が変わらないように自動で挿入する。しかし、使用する PE 数が多い、またはループ構造が含まれるなど回路が複雑になると、コンパイル時に回路作成者の意図しない箇所にディレイ PE が自動で挿入されてしまう場合がある。その場合、配置配線された回路はコンパイル前の回路とは異なった演算結果を出力してしまう。よって、SHA-1, SHA-256 のようなループ構造を含む回路を設計する場合、予め手動でセグメント間のデータ受け渡し用のディレイ PE を配置することでこの問題を解決できる。しかし、設計者はセグメント毎の使用 PE 数に常に注意を払う必要がある。

次に、実装した SHA-1, SHA-256 回路の DNA, DAP それぞれでの設計方法と実装方法について述べる。

4.1 SHA-1 の実装

4.1.1 演算部分の DNA 上での実装

DNA に実装した SHA-1 回路の設計・実装方法について述べる。DNA 上に 3.2(4)から 3.2(8)の SHA-1 の演算部分を実装した。図 16 は DAPDNA-2 用の GUI で設計した SHA-1 回路図である。例外を除いて、1 つのブロックが 1 つの PE を表している。f[t]の計算部分はインスタンス化して設計を行った。

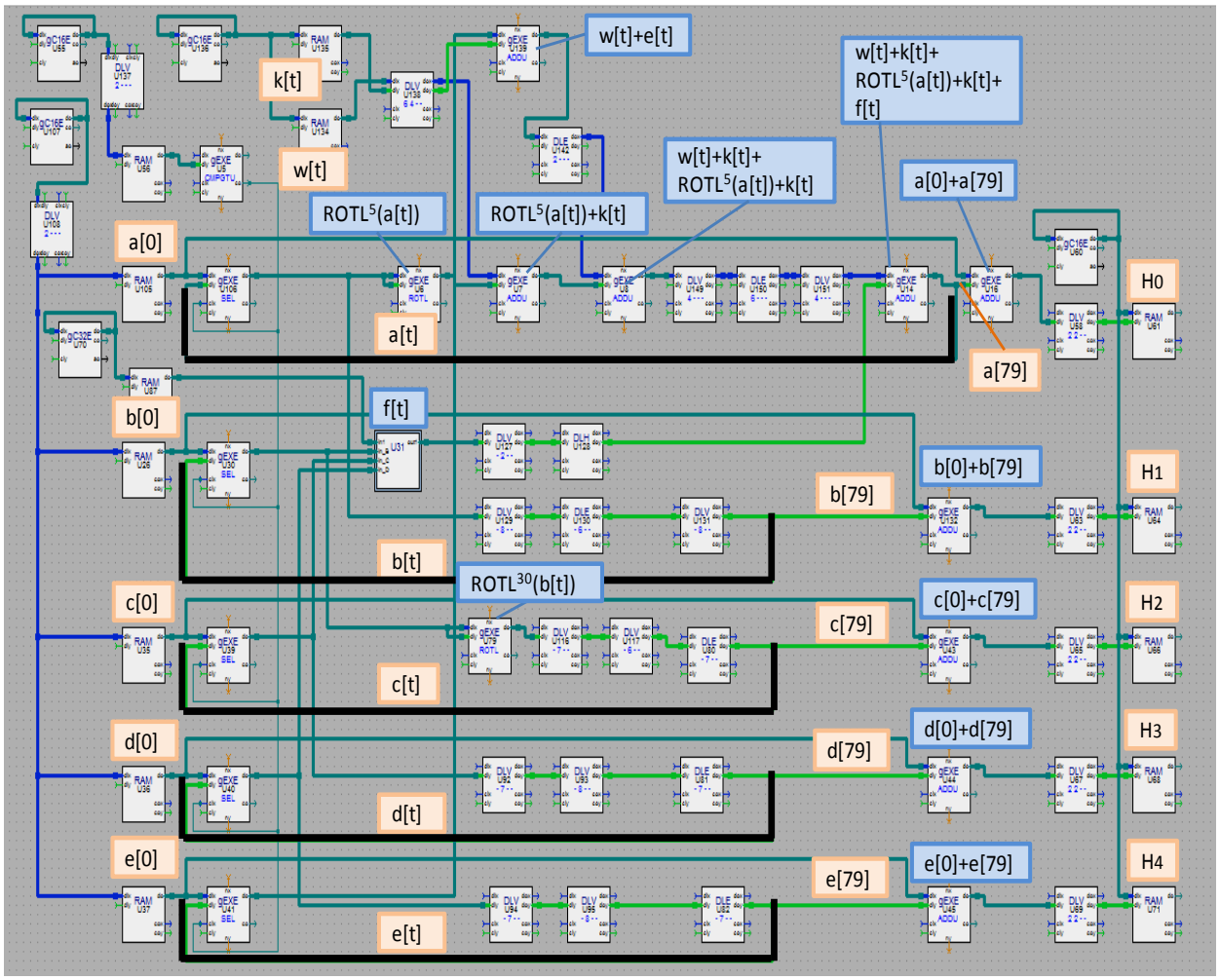

図 16：DNA に実装した SHA-1 回路

SHA-1 回路のハッシュ演算部分、主にハッシュ生成部は a[t], b[t], c[t], d[t], e[t]の 5 つの変数と、K[t], W[t]の 2 つの定数を用いて論理計算などを行う。ハッシュ生成部は a[0], b[0], c[0], d[0], e[0]を元に a[79], b[79], c[79], d[79], e[79]まで 80 回のループ計算を行うので、データ依存性のある計算である。よって図 16 に太線で示している a[t], b[t], c[t], d[t], e[t]はループ構造をとっている。また、入力は PE マトリックスの RAM エレメントに入力/出力するようになっている。

SHA-1 回路の作成にあたり、前述のレイテンシの調節と、セグメント間のデータの受け渡し用ディレイ PE の配置を手動で行う必要がある。はじめに、これらの配置を無視して作成した回路のブロック図を図 17 に示す。これは計算部分のみを考慮して作成した回路のブロック図である。しかしこの回路では、レイテンシの調節ができておらず、2 入力の PE のそれぞれの入力に、望んでいるタイミングとは数クロックずれたタイミングでデータが入力され、正しい計算が行われず、ハッシュ値を得ることはできない。そこに、ディレイ PE の配置を行ったのが図 18 のブロック図である。回路の適切な位置に、レイテンシの調整の

ディレイ PE、セグメント間のデータ受け渡し用のディレイ PE を追加することで、DNA コンパイラで配置配線しても、自動で PE の挿入がされなくなる。これにより、正しいタイミングでデータが流れ、正しいハッシュ値を得ることができる。なお、ディレイ PE 配置前ではハッシュ生成の 1 ステップに 12 クロック要するが、配置後では 22 クロック要する。

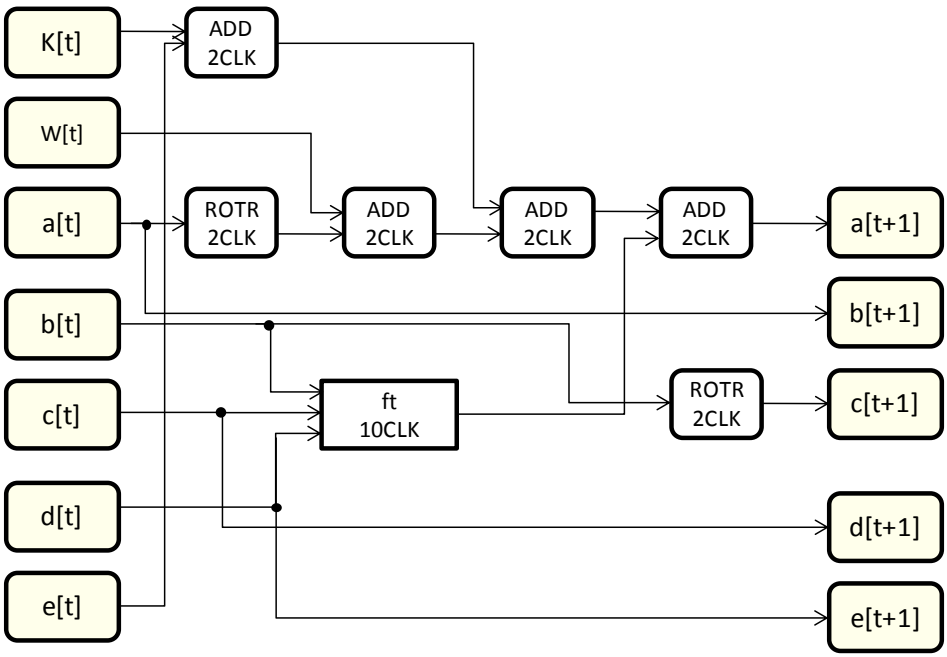

図 17 : ディレイ PE 配置前の SHA-1 回路ブロック図

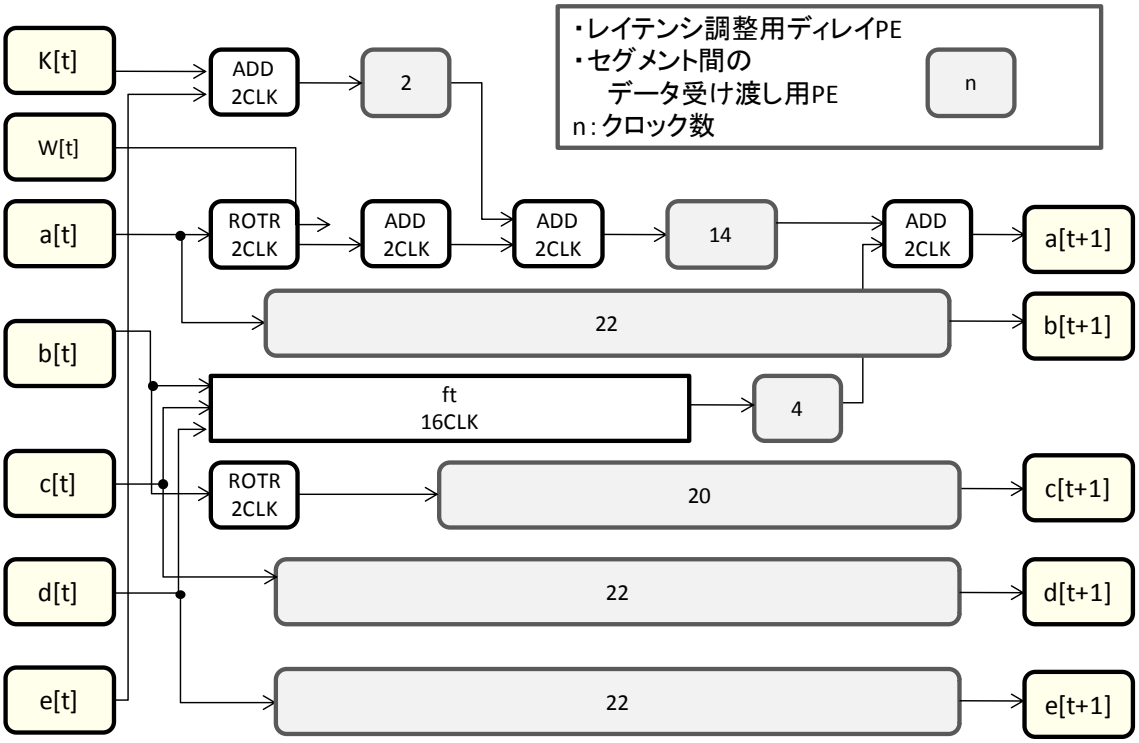

図 18 : ディレイ PE 配置後の SHA-1 回路ブロック図

SHA-1 回路の PE 使用数を表 3 に示す。

表 3 : SHA-1 回路の使用 PE 数

機能		使用数
演算 PE	EXS, EXC, EXM EXR, EXF	39
RAM	RAM	14
入出力	LDB, LDX, STB, STX	0
遅延	DLE	11
エレメント間受け渡し	DLV	20
	DLH	4
	DLX	0
カウンタ	C16L, C16E, C16S C32L, C32E, C32S	5
合計		93

4.1.2　制御部分の DAP 上での実装

DAP プログラムは DAPDNA-2 全体の制御を行い、主に DNA の制御を行う。SHA-1 回路の DNA 制御プログラムは以下の様な流れで動作するように作成した。

1. 開始
2. DNA コンフィギュレーションの切り替え条件と割り込み条件の取得
3. DNA コンフィギュレーションを DNA コンフィギュレーションメモリのバックグラウンドメモリへロード
4. DNA コンフィギュレーションメモリのバックグラウンドメモリからフォアグラウンドメモリへロード
5. DNA コンフィギュレーションの終了条件を取得する
6. DNA の実行を開始
7. 終了条件を満たすまで終了を待つ
8. DNA の実行を停止
9. 終了

この動作をする DAP プログラムを作成し、DAP コンパイラでコンパイルし、DNA コンフィギュレーションを合わせて動作させる。図 19 にこの DAP プログラムと、1 から 9 の対応を示す。

```
1   int main(void)
    {
    /*////////////////////////////////////////////////////
                              省略
    ////////////////////////////////////////////////////*/
2           /* DNA コンフィギュレーションの終了条件を取得する */
            dna_get_event(matrix, &Switch, &Completion);

            /* RAM エレメントを初期化する*/
            dna_init_ram(1, matrix, 0);

3           /* DNA コンフィギュレーションをDNAコンフィギュレーションメモリの
            バックグラウンドメモリへロードする*/
            dna_cfgram_load3(1, matrix);

            /* DNAコンフィギュレーションメモリのバックグラウンドバンクの
4           DNA コンフィギュレーションをフォアグラウンドバンクへロードする */
            dna_config(0, 1);

            /* DNA のステータスをクリアする */
            dapdna_write(DNAICLR, Completion);

5           /* DNA コンフィギュレーションの終了条件を設定する */
            dna_set_event(matrix);

            /* DNA を実行する */
6           dna_run(0);

7           /* 終了を待つ */
            while ((dapdna_read(DNAISTR) & Completion) != Completion);

            /* DNA を停止する */
8           dna_stop(0);

9           return 0;
    }
    /*////////////////////////////////////////////////////
                              省略
    ////////////////////////////////////////////////////*/
```

図 19：DAP プログラム

4.2 SHA-256 の実装

4.2.1 演算部分の DNA 上での実装

DNA に実装した SHA-256 回路の設計・実装方法について述べる。DNA 上に 3.3 (4)から 3.3(8)の SHA-256 の演算部分を実装した。図 20 は DAPDNA-2 用の GUI で設計した SHA-256 回路図である。Ch, Maj, Σ0, Σ1 の計算部分はインスタンス化して設計を行った。

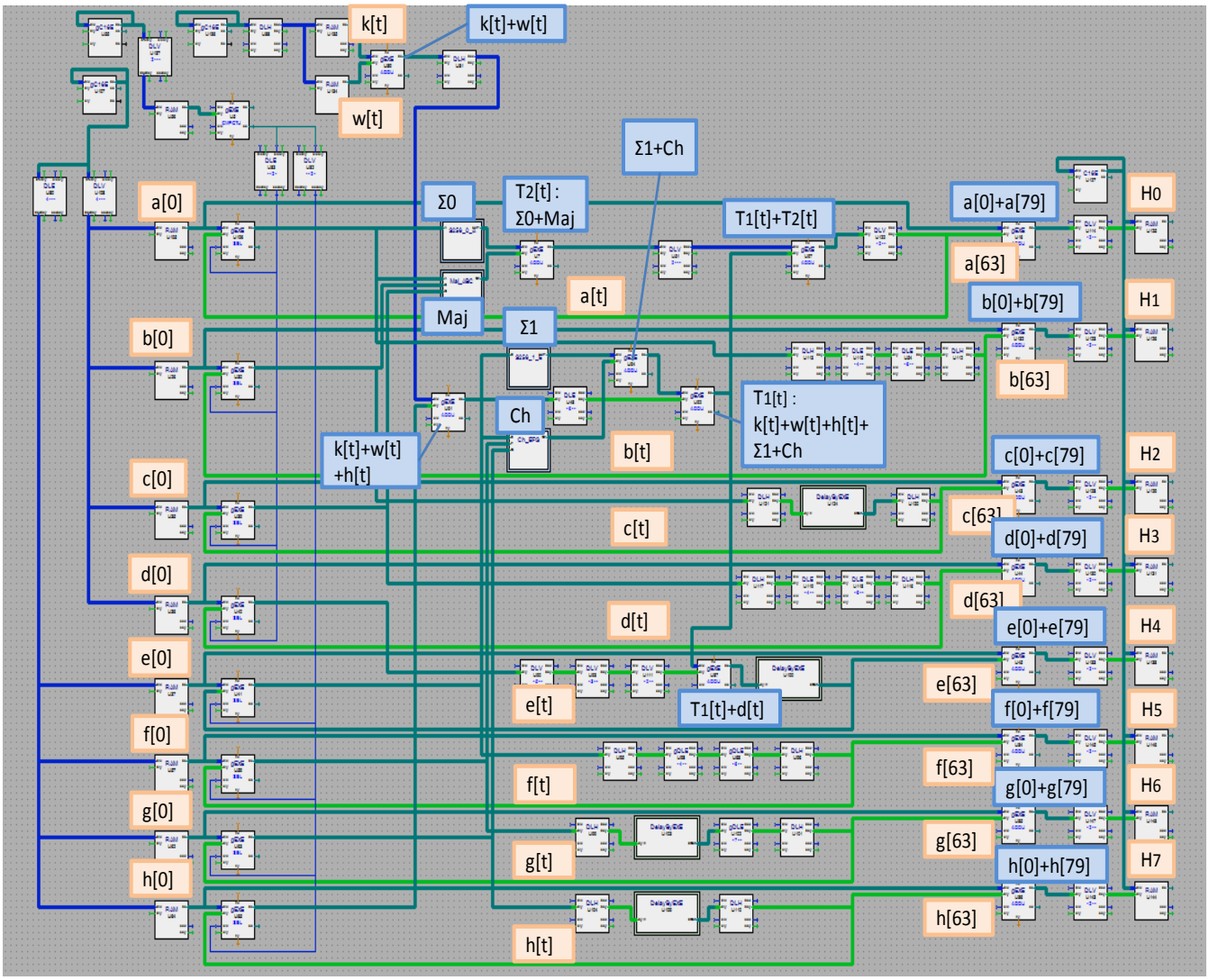

図 20：DNA に実装した SHA-256 回路

SHA-256 回路のハッシュ演算部分、主にハッシュ生成部は a[t], b[t], c[t], d[t], e[t], f[t], g[t], h[t]の 8 つの変数と、K[t], W[t]の 2 つの定数を用いて論理計算などを行う。ハッシュ生成部は a[0], b[0], c[0], d[0], e[0], f[0], g[0], h[0]を元に、a[63], b[63], c[63], d[63], e[63], f[63], g[63], h[63]まで 64 回のループ計算を行うので、データ依存性のある計算である。よって作成した図 20 の回路においても a[t], b[t], c[t], d[t], e[t] , f[t], g[t], h[t]はループ構造をとっている。

SHA-256 回路の作成にあたり、SHA-1 と同様にディレイ PE の配置を手動で行う必要がある。これら 2 つの調節を無視して作成した回路のブロック図を図 21 に示す。これは計算部分のみを考慮して作成した回路のブロック図である。これに各ディレイ PE の配置を行ったものが図 22 のブロック図である。ディレイ PE 配置前ではハッシュ生成の 1 ステップに 12 クロック要するが、配置後では 14 クロック要する。

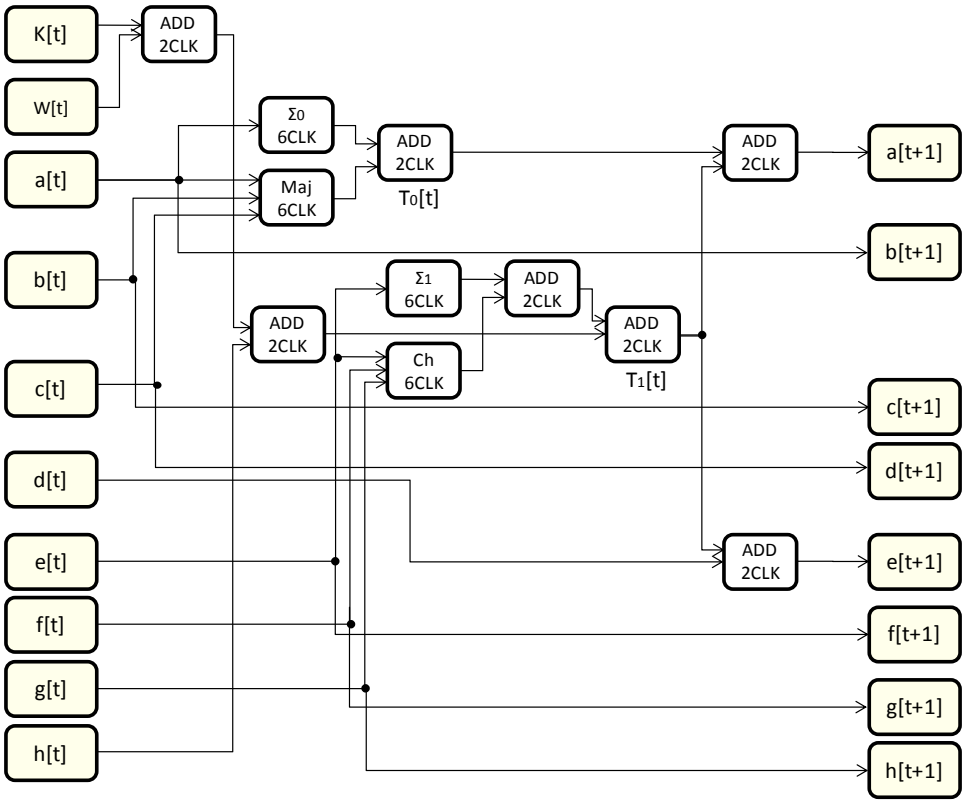

図 21：ディレイ PE 配置前の SHA-256 回路ブロック図

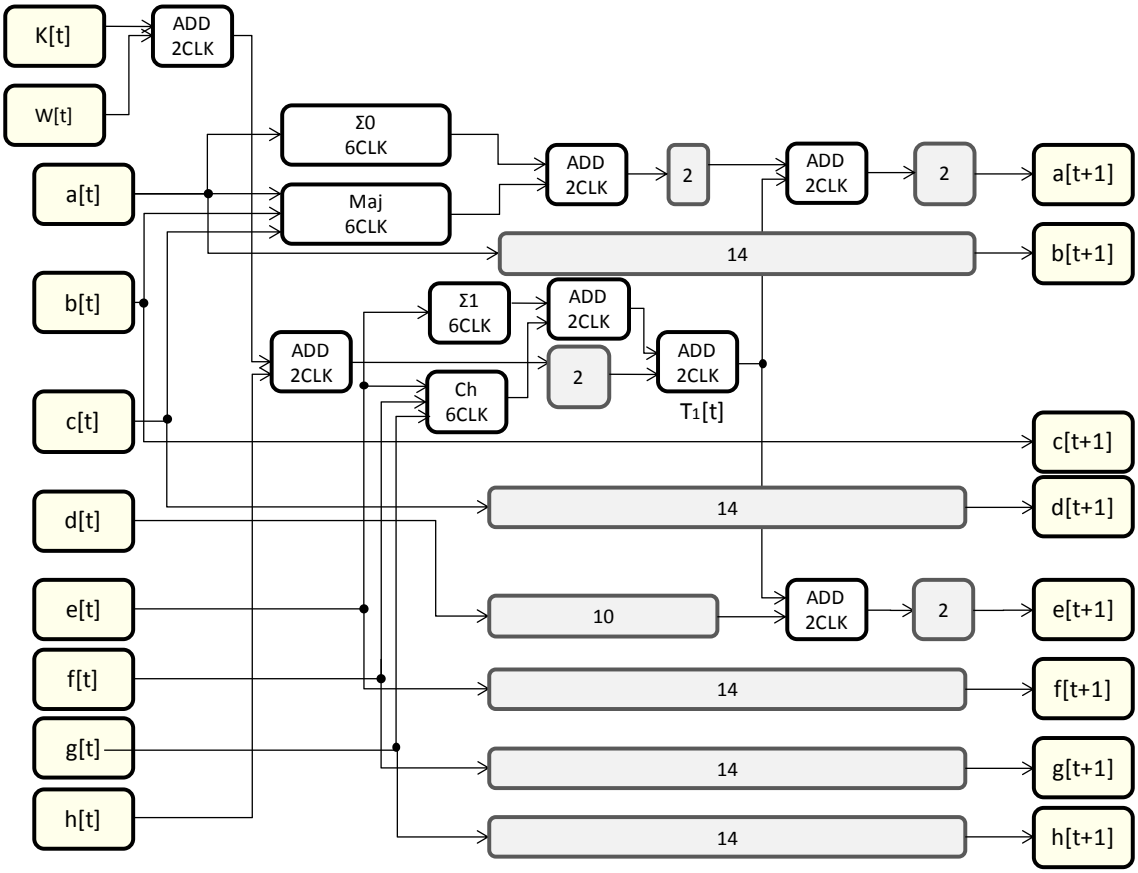

図 22：ディレイ PE 配置後の SHA-256 回路ブロック図

ブロック図内の Ch, Maj, Σ0, Σ1 のブロック は回路のインスタンスである。また、SHA-256 回路の使用 PE 数を表 4 に示す。

表 4：SHA-256 回路の使用 PE 数

機能		使用数
演算 PE	EXS, EXC, EXM EXR, EXF	53
RAM	RAM	19
入出力	LDB, LDX, STB, STX	0
遅延	DLE	15
エレメント間受け渡し	DLV	9
	DLH	14
	DLX	0
カウンタ	C16L, C16E, C16S C32L, C32E, C32S	7
合計		117

4.2.2 制御部分の DAP 上での実装

DAP プログラムは DAPDNA-2 全体の制御を行い、主に DNA の制御を行う。SHA-256 回路の DNA 制御プログラムは以下の様な流れで動作するように作成した。

1. 開始
2. DNA コンフィギュレーションの切り替え条件と割り込み条件の取得
3. DNA コンフィギュレーションを DNA コンフィギュレーションメモリにロード
4. DNA コンフィギュレーションメモリの指定されたバンク中のデータを PE マトリックスにロード
5. DNA の実行を開始
6. 終了条件を満たすまで終了を待つ
7. DNA の実行を停止
8. 終了

この動作をする DAP プログラムを作成し、DAP コンパイラでコンパイルし、DNA コンフィギュレーションと合わせて動作させる。この DAP プログラムは図 19 の SHA-1 の DAP プログラムと同様である。

4.3　動的再構成を用いた実装

SHA-1 回路を、動的再構成を用いて DAPDNA-2 上に実装を行う実装案について述べる。SHA-1 のハッシュ生成は、80 ステップ中 20 ステップ毎に計算内容が異なり、20 ステップずつ 4 種類の論理計算を行う。4.1 ではこの 4 種類を同一の回路内に設計し、実装を行っている。よって、処理中のステップによって使用されない回路が存在する。回路内に使用されない回路が存在していると、リーク電流、回路規模の増大の原因となる。動的再構成を行うとこれらの点を解決することができる。図 23 に動的再構成可能を用いた SHA-1 の実装方法を示す。SHA-1 の 20 ステップ毎の回路を順に動的再構成により切り替える。動的再構成に要する時間は数クロックで、秒数にすると 0 に近く無視すると考えられ、動的再構成により SHA-1 回路の回路規模を抑えることができ、消費電力も減らすことができる。

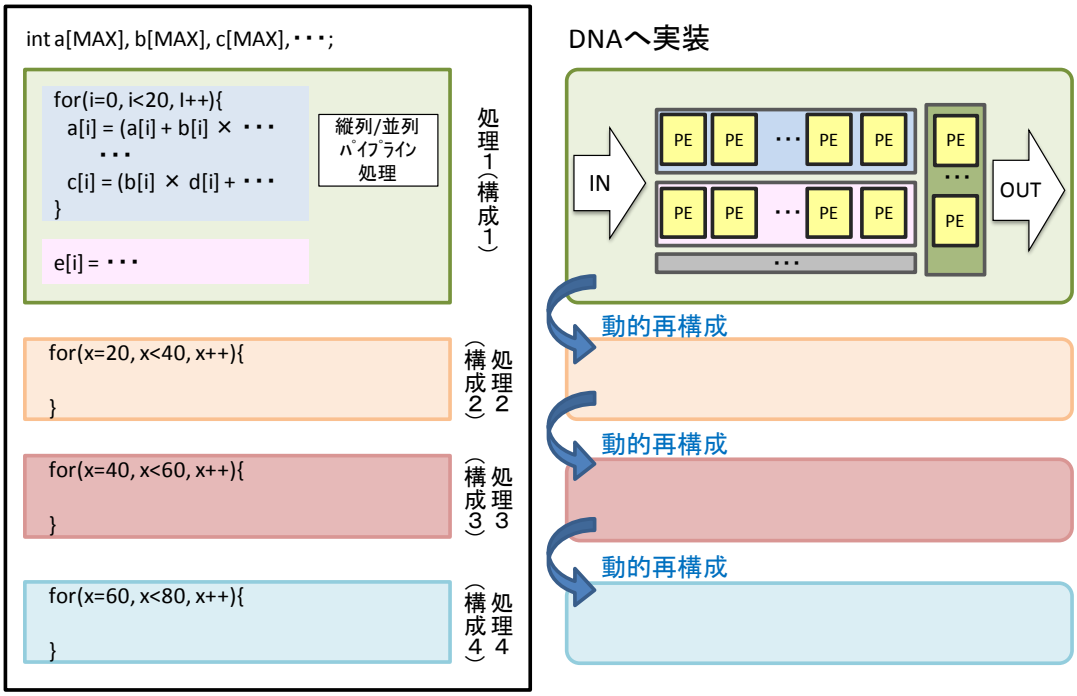

図 23：動的再構成を用いた SHA-1 の実装方法

5. 実験と考察

5.1 実験方法

動的再構成可能プロセッサ DAPDNA-2 上に SHA-1, SHA-256 のプログラムを実装し、それぞれの実行時間を計測する。DNA コンフィギュレーションは DAPDNA-2 用の開発ツール DNA Designer（GUI ツール）を用いて回路を作成し、DNA 上へ実装を行う。全体の制御を行う DAP プログラムは C ソースコードで記述し、DAP 上へ実装する。2 コアプロセッサ（IntelCore2Duo：動作周波数 2.53GHz, メモリ 4GB）と、4 コアプロセッサ（IntelCore2Quad : 動作周波数 2.66GHz, メモリ 4GB）に同様の処理を実行させ、実行時間を比較し、考察を行う。

5.2 実験結果

5.2.1 SHA-1

SHA-1 を実装するための DAP プログラムと DNA プログラムを作成し、DAPDNA-2 上へ実装した。また、DAPDNA-2 上へ実装した SHA-1 と同じアルゴリズムで、SHA-1 の C プログラムを作成し、動作させた。それぞれの実行時間を表 5 に示す。

表 5：SHA-1 実行時間

	DAPDNA-2（166MHz）	Core2Duo（2.53GHz）	Core2Quad(2.66GHz)
全実行時間[μ s]	462	13000	12000
ハッシュ生成(DNA)時間 [μ s]	2	4000	4000

全実行時間は DAP プログラムと DNA プログラムの全ての実行時間である。ハッシュ生成時間は、DAP が DNA の実行を開始する命令を出し、DNA で回路が実行され、DNA の実行を停止するまで、すなわち SHA-1 に必要な初期値を読み込んで、ハッシュ生成を行い、出力を行うまでの実行時間である。Core2Duo と Core2Quad のハッシュ実行時間は、DAPDNA-2 でのハッシュ生成部と同様の処理内容の部分を C プログラムで動作させた実行時間である。動作周波数 166MHz の DAPDNA-2 での実行時間は、動作周波数が約 15 倍である汎用プロセッサに比べて約 27 倍の速度向上と得られた。DAPDNA-2 での実行時間のうちのほとんどは、DNA 制御を行う DAP プログラムに要しており、DNA 上でのハッシュ生成時間はごくわずかである。また、PE マトリックスのみの消費電力は 977[mW]である。消費電力は DAPDNA-2 専用のツールで調べることができる。

5.2.2 SHA-256

SHA-1 と同様に、SHA-256 を DAPDNA-2 上と C プログラムで動作させた。それぞれの実行時間を表 6 に示す。

表 6：SHA-256 実行時間

	DAPDNA-2（166MHz）	Core2Duo（2.53GHz）	Core2Quad(2.66GHz)
実行時間[μ s]	397	15000	14000
ハッシュ生成(DNA)部 実行時間 [μ s]	2	5000	4000

動作周波数 166MHz の DAPDAN-2 での実行時間は、動作周波数が約 15 倍である汎用プロセッサに比べて約 36 倍の速度向上となっている。また、PE マトリックスのみの消費電力は 1263[mW]である。

5.3 考察

SHA-1, SHA-256 回路を動的再構成可能プロセッサ DAPDNA-2 上で動作させた。動作周波数の高い汎用プロセッサに比べ、DAPDNA-2 の動作周波数は低いにも関わらず、大きな速度向上を得ることができた。DAPDNA-2 は専用回路を実装可能であり、入力を行いながら計算の処理を並行して進めていくことができ、処理時間の短縮が可能である。

回路（DNA コンフィギュレーション）設計に用いる専用 GUI ツールは、視覚的に設計ができるため、イメージしている回路をそのまま回路に実現することができる。ユーザにとって作業が行いやすいと感じた。

DNA 内の PE マトリックスは 376 個の PE を持つが、実装した SHA-1 は 93 個の PE を用い、SHA-256 回路は 117 個の PE を用いた。実行時間とハッシュ生成部実行時間より、SHA-1, SHA-256 ともに DAP プログラムが全体の処理時間の 99%以上を占めている。C プログラムを汎用プロセッサで動作させたものは、定数の入力に多くの時間を占めている。しかし、DAPDNA-2 での処理時間は汎用プロセッサに比べて非常に短縮された。

動的再構成を行わない方法と行う方法の使用 PE 数や必要クロック数の比較を表 7 に示す。動的再構成を行なえば、使用する PE 数を大幅に減らすことができる。動的再構成を行わない方法では 93 個の PE を要したが、動的再構成を行う方法では 1 つの回路あたり 53 個の PE で設計を行うことができる。また 1 ステップ毎の必要クロック数は、22 クロックから 8 クロックに減らすことができる。これらのことより、SHA-1 を動的再構成を行う方法で実装すれば、消費電力と実行時間を減らすことができる。実行クロック数において、ハッシュ生成のステップ部分のみに注目すると、22 クロック×80 ステップ＝1760 クロック必要

なものが、8 クロック×80 ステップ＝640 クロックに削減することができる。

表 7：動的再構成を行わない方法と行う方法の比較

	動的再構成を行わない方法	動的再構成を行う方法
使用 PE 数	93	53
1 ステップに要するクロック数	22	8
全クロック数	1760	640
消費電力[mW]	977	742

6. おわりに

本研究では、動的再構成可能プロセッサ DAPDNA-2 上に、ハッシュ関数である SHA-1 と SHA-256 アルゴリズムを実装した。DAP 上には全体の制御を行うプログラムを、DNA 上にはそれぞれのハッシュ生成回路を実装した。また、SHA-256 も同様に実装した。DNA 上に実装した回路は複雑なループ構造をとっているため、レイテンシ調整とセグメント間のデータ受け渡しのためのディレイ PE を意図的に挿入し、設計を行った。実行時間において、動作周波数が約 15 倍の汎用プロセッサに比べて、SHA-1 は約 27 倍、SHA-256 は約 36 倍の速度向上を得られた。DAPDNA-2 での実行時間の 99%以上は DAP プログラムが占めており、さらに複雑な回路を DNA 実装することが可能である。また、DAPDNA-2 上に動的再構成を用いた方法で SHA-1 回路を実装することを検討した。SHA-1 を動的再構成を用いて実装すれば、使用 PE 数を 93 個から 53 個に削減することができる。また、全実行クロック数は 1760 から 640 へ、消費電力は 977mW から 742mW へ削減することができる。

今後の課題として、パイプライン化を用いた SHA の実装が考えられる。また SHA 以外においても、動的再構成可能プロセッサの長所を活かすために、連続したデータを扱うストリーム処理などのアルゴリズムを、動的再構成を用いて実装することが必要である。

謝辞

本研究の機会を与えて下さり、貴重な助言、ご指導を頂きました山崎勝弘教授に深く感謝いたします。 また, 本研究に関して様々な相談に乗って頂き、貴重なご意見を頂きました、泉知論教授、浦口真治氏、及び高性能計算研究室の皆様に深く感謝いたします。

参考文献

[1] 末吉敏則, 天野英晴：リコンフィギャラブルシステム, オーム社, 2005.

[2] 加藤良平：ダイナミックリコンフィギャラブルプロセッサ入門講座, 電波新聞社, 2008.

[3] IPFLEX：http://www.ipflex.com/jp/

[4] 進藤智則:やわらかくなる LSI, 日経エレクトロニクス, 7 月 27 日号, 日経 BP 社, 2009.

[5] 天野英晴：動的リコンフィギャラブルシステムの最近の動向, 信学会誌, Vol.91, No.6, pp.478-483, 2008.

[6] Hoang Anh Tuan, 山崎勝弘, 小柳滋 : Three-stage Pipeline Implementation for SHA2 using Data Forwarding, IEEE FPL2008, MIB-2, 2008.

[7] 渡辺良亮, 阿部公輝 : 画像処理の逐次実行を実装例とした動的再構成可能プロセッサの評価, 電子情報通信学会技術研究報告（信号処理研究会）, Vol.108, No.104, pp.57-62,2008.

[8] 桑門秀典：暗号アルゴリズムの評価, 通信･放送機構情報処理振興事業協会, 2001.

[9] 亀井雄介, 山崎勝弘 : 動的リコンフィギャラブルプロセッサを用いた暗号化アルゴリズムの高速化の検討, 情報処理学会第71回全国大会, 6ZD-8, 2009.

[10] National Institute of Standards and Technology, Secure hash standard, NIST FIPS PUB 180-1, 1993.

[11] National Institute of Standards and Technology, Secure hash standard, NIST FIPS PUB 180-2, 2002.

[12] B. Wilkinson, M. Allen : Parallel Programming : Techniques and Applications Using Networked Workstations and Parallel Computers, 2nd Edition, Pearson Education (US), 2004.

[13] ウィリアム・スターリングス：暗号とネットワークセキュリティ 理論と実際, ピアソン・エデュケーション, 2001.

[14] 神田雅透：これだけは知っておきたいアルゴリズム～ハッシュ関数・公開鍵暗号・デジタル署名編：http://www.atmarkit.co.jp/fsecurity/rensai/crypt04/crypt01.html

[15] Japan Society of the Promotion of Science, “Ultimately Integrated Devices and Systems” The 165 Research Committee : Extended Abstracts of International Symposium on Advances Reconfigurable Systems, 2005.

[16] 山田和夫, 内藤孝雄, 他：DRP技術を活用したリアルタイム画像処理システムの構築, 富士ゼロックステクニカルレポートNo.18, 2008.