リコンフィギャラブル・コンピューティングに関する研究

2002年3月

熊本大学大学院自然科学研究科
飯田全広

目次

表目次

図目次

第 1 章

序論

本論文は，アプリケーションに合わせて自分自身のハードウェア構成を変更する計算処理方式であるリコンフィギャラブル・コンピューティングの構成方法と，その応用について行った研究をまとめたものである．

1.1 研究の背景

フォン・ノイマンが確立した計算機のアーキテクチャには，逐次処理方式やプログラム内臓方式が採用された．このアーキテクチャは当時のハードウェア技術，すなわち，利用可能な素子（真空管）や実装技術，そしてコスト的な強い制約を受けた結果である．実際，ノイマンは，単に万能チューリングマシンとして計算機の実現を夢想していたにすぎず，他の方式を否定しているわけではない．しかしながら，現在のいわゆるノイマン型計算機もその特徴を強く受け継いでおり，汎用ハードウェアとプログラム内臓制御が最大の特徴となっている．これはハードウェアとソフトウェアの分離を意味し，ノイマン以降，半世紀にわたって，それぞれがお互いに影響を受けながらも別個に議論され，また独自の発展を遂げてきている．例えば，ハードウェアは，レジスタ修飾や割り込み，仮想記憶，キャッシュメモリなどによって，より使いやすく，高性能を目指した拡張が施されてきている．また，ソフトウェアは，OS，コンパイラをはじめ，分散・並列処理の枠組み，プロセス，スレッドの発明など，こちらも同様に改良が加えられている．しかしながら，いずれの改良・拡張も，ノイマンのプログラム内臓制御の大枠内であるため，プログラムとデータの同居により，メモリ・バスの慢性的なスループット不足，いわゆるノイマン・ボトルネックの根本的な解消は原理的にできない．また，毎回命令を解読する無駄，すなわ

ちノイマン・オーバヘッドも同様に完全には解消できない.[*1]

元来，ノイマンはハードウェア・コストの削減のためにプログラム制御を考案したのあって，はじめからプログラム制御を目指していたわけではない．ノイマンの目指した計算機の基本概念は，チューリングに負う面が大きく，万能チューリング・マシンは，決してハードウェアとソフトウェアの分離を前提にしていない．したがって，1 チップに数億トランジスタを集積できるほど実装技術が発達した現在は，ノイマンの時代とは前提が異なり，ハードウェア・コスト削減のためにプログラム制御を採用することが合理的な理由ではなくなりつつある．

一方，計算機の進歩に伴って，それに使用されるデバイスの進歩も著しい．トランジスタの集積度や動作速度の改善のみならず，1980 年代には回路構成を変更できるプログラマブル・ロジックデバイス（PLD:Programmable Logic Device）が登場し，さらに 1990 年代以降，何回でも回路の書換えが可能な FPGA （Field Programmable Gate Array）が開発されてきた．これらのプログラマブル・デバイスは，主として LSI の試作や評価用に使用されているが，計算機アーキテクチャにもインパクトを与えつつある．ノイマン型計算機は処理内容をプログラムとして実装し，逐次的に命令解釈しながら実行するのに対して，処理内容をプログラマブル・ロジックデバイス上の回路として実装し，並列的に直接動作する計算方式が実現可能になってきた．また，プログラマブル・ロジックデバイスは，従来の固定的な LSI とは異なり回路を変更できることから，ノイマン型の計算機と同等の柔軟性も持つことができる．したがって，このようなハードウェアの変更可能な計算方式（再構成可能な計算方式），すなわち，リコンフィギャラブル・コンピューティング（Reconfigurable Computing）の登場によって，ノイマン型の固定的な汎用ハードウェアとプログラム制御という枠組みから解放される機運が高まってきている．

1.2 本研究の目的

リコンフィギャラブル・コンピューティングとは，対象とするアプリケーションに合わせて自分自身のハードウェア構成を変更することで処理を行う計算パラダイムの総称である．アプリケーションをソフトウェアによって実現する場合は，実装対象となる固定的なハードウェア資源によって，演算の並列性やデータ幅，データ転送などが制約を受ける．ゆえに，アプリケーションはその制約内で実現可能なアルゴリズムを採用せざるを得ず，

[*1] 計算機の歴史については，文献 [1] が詳しい．

結果として，性能はハードウェア上の制約によって支配される．一方，アプリケーションを直接ハードウェア化した場合は，ハードウェア上の制約は少なく，アルゴリズム自身の能力が性能に直結する傾向が強い．また，アプリケーションの変更に対しても，ソフトウェアと同程度の柔軟性を有している．すなわち，リコンフィギャラブル・コンピュータの特徴は，専用ハードウェアの持つ高い性能とノイマン型の計算機と同等な柔軟性を併せ持つところにある．

しかしながら，現在のリコンフィギャラブル・コンピュータには，ノイマン型計算機が持つコンパイラなどのアプリケーション開発環境や，OS などの実行環境に相当する部分が欠けており，現実問題としてノイマン型計算機と共存しなければ存在し得ない．実際，アプリケーション開発は LSI の開発環境を流用しており，実装デバイスは，主として試作や小ロット製品に使われる FPGA などの既存デバイスである．

本研究は，このような現実を踏まえ，リコンフィギャラブル・コンピューティングの実現方法を議論し，ノイマン型計算機と対峙するのではなく，補完することを目的にしたリコンフィギャラブル・コンピューティングの在り方を明らかにする．また，リコンフィギャラブル・コンピューティングに適した応用を議論すると共に，この計算方式の特徴であるアルゴリズムとアーキテクチャの最適化を実験によって検証する．一方，リコンフィギャラブル・コンピューティングで使用するデバイスのアーキテクチャは別途議論されるべき重要な問題である．本研究では，従来のプログラマブル・デバイスに追加すべき機能や改善すべき機能を明確にし，新しいリコンフィギャラブル・コンピューティング向きのデバイス，すなわち，リコンフィギャラブル・ロジックを提案し，その評価とともに有効性を確認する．

1.3 本論文の構成

本論文は 6 章からなり，第 2 章以降の各章の構成は以下の通りである．

第 2 章「リコンフィギャラブル・コンピューティング」では，リコンフィギャラブル・コンピューティングの概念について明確にし，用語定義を行う．その上で，これまで開発されてきたリコンフィギャラブル・コンピュータの構成方式による分類と，リコンフィギャラブル・コンピューティング用に開発されたデバイスのアーキテクチャについて概説するとともに，これまでの開発事例について述べる．そして，現在リコンフィギャラブル・コンピューティングが抱えている課題について整理する．

第 3 章「リコンフィギャラブル・コプロセッサ方式」では，リコンフィギャラブル・コ

ンピューティングの一方式であるリコンフィギャラブル・コプロセッサ方式（プロセッサ，プログラマブル・ロジック混載方式）の研究成果について報告する．本研究では，スレッドを並列実行単位としたオンチップ・マルチプロセッサのスレッド制御部にリコンフィギャラブル・コンピューティング技術を適用した方式を議論し，制御方式，性能評価について述べる．

第4章「アタッチド・プロセッサ方式」では，リコンフィギャラブル・コンピューティングの一方式であるアタッチド・プロセッサ方式の研究成果について報告する．本研究では，計算集約的な処理に対してリコンフィギャラブル・コンピューティングが有効であることを示し，その応用として暗号解析とレーダ信号処理での適用事例について報告する．また，これらの応用からアルゴリズムとアーキテクチャの最適化の効果を示す．

第5章「リコンフィギャラブル・ロジックの提案と評価」では，リコンフィギャラブル・コンピューティングに使用するデバイスについて議論し，FPGA や PLD に変わる新しいリコンフィギャラブル・ロジックの論理ブロック構成を提案し，シミュレーションによる評価などについて述べる．

そして，第6章「結論」では，本研究のまとめを行うとともに，今後の課題について述べる．

第 2 章

リコンフィギャラブル・コンピューティング

本章では，まず，リコンフィギャラブル・コンピューティングの概念を明確にし，本研究で使用される用語について定義する．さらに，リコンフィギャラブル・コンピューティングの利用形態およびデバイス構造による分類と開発事例について概観する．そして，現在のリコンフィギャラブル・コンピューティングシステムの課題を整理した上で，本研究の位置付けを明らかにする．

2.1 リコンフィギャラブル・コンピューティングの概念

現在では何らかの機械的な情報処理を必要とする場合，専用 LSI を用いた装置を開発するより，計算機上のソフトウェアで実現する方が一般的となっている．これは手軽で短時間にでき，かかるコストも低いというのが主な理由である．しかし，実現性の検討をしてみると，それでは速度などの要求性能が満たせないケースも少なくない．そのような場合には，時間とコストをかけてもアクセラレータや専用計算機を開発することになる．

見方を変えると，前者の方法はアーキテクチャ（ハードウェア）を固定してアルゴリズム（ソフトウェア）を変更する方法と言える．一方，後者の方法はアルゴリズム（ソフトウェア）を固定してアーキテクチャ（ハードウェア）を変えることで要求性能を満たそうとする方法と言い換えることができる．こうした試行錯誤は日常的に行われるが最良の方法とは言い難い．できることなら，最初から最適なアーキテクチャとアルゴリズムを一度に実装できることが望ましいが，これはアプリケーション毎に専用計算機を開発し続けることと等価で，現在の計算機の枠組みでは時間とコストの面から難しい．

ところが，PLD や FPGA というプログラマブル・デバイスの登場によってハードウェア（回路）自身も容易に変更できるようになった．最初から最適なアーキテクチャとアル

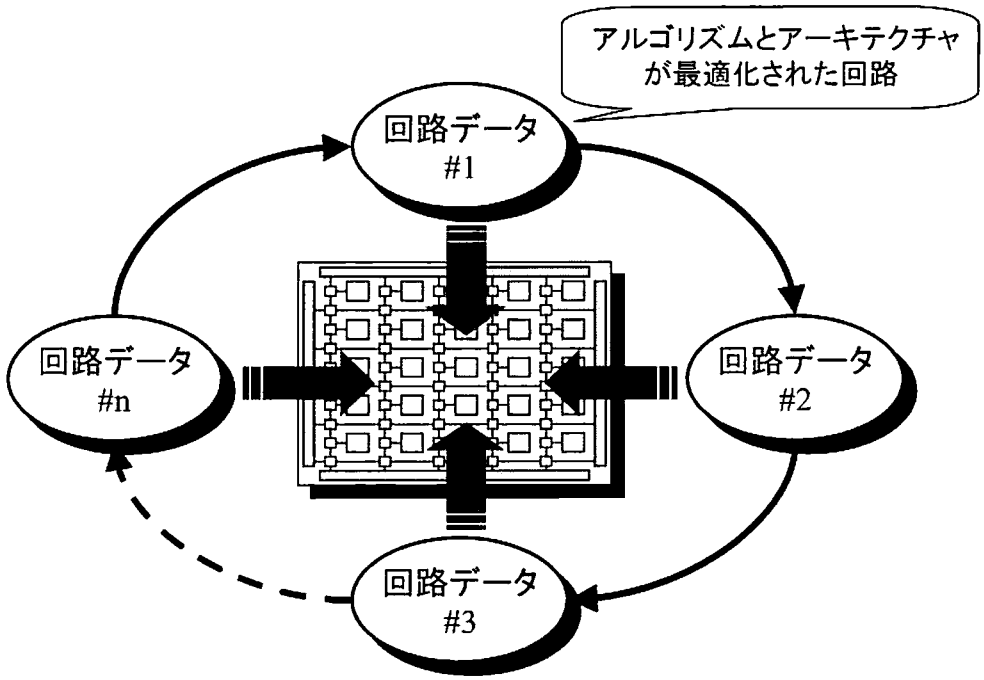

図 2.1: リコンフィギャラブル・コンピューティングの概念

ゴリズムを一度に実装できる可能性がでてきた訳である．リコンフィギャラブル・コンピューティングとは，まさにこのプログラマブル・ロジックの再構成性を最大限活用する．図 2.1 にリコンフィギャラブル・コンピューティングの概念を示す．図中の個々の回路データは，実現しようとしているアプリケーションに最適化したアーキテクチャとアルゴリズムを持つ部分回路であり，それを次々と書換えることでアプリケーションが動作する [2].

このようなハードウェア構成を変更するコンピュータというアイディアは，1960 年代の Gerald Estrin の提案に見られるように決して新しいものではない．しかし，当時のハードウェア技術では実現が難しく，80 年代に PLD が登場して以降，とりわけ何回でも書換えできる FPGA が出回り始めた 90 年代初頭から急速に活発な研究開発が行われるようになった.

2.1.1 リコンフィギャラブル・コンピューティングの効果

リコンフィギャラブル・コンピューティングの特徴は，ハードウェアの持つ高い性能とソフトウェアの持つ高い柔軟性を合わせ持つところにある．すべての処理をプログラマブ

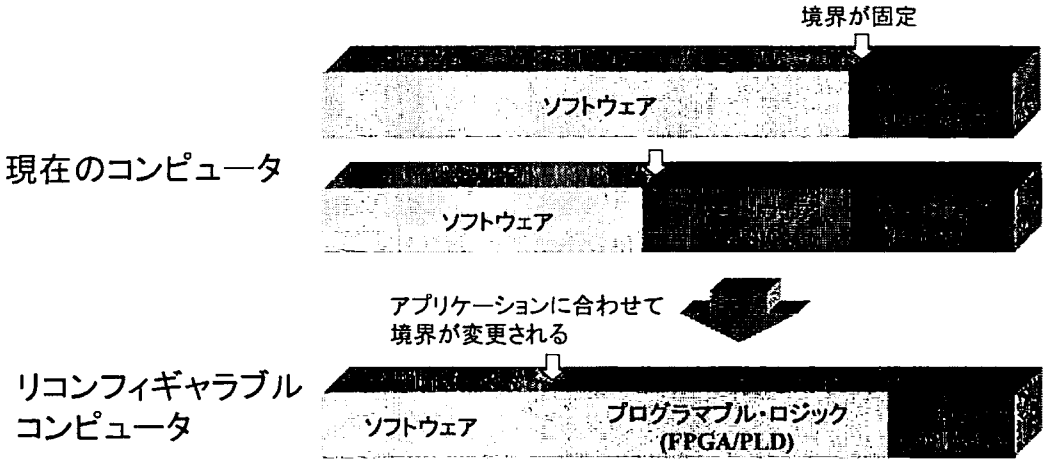

図 2.2: ソフトウェアとハードウェアの境界の変化

ル・ロジックで行うことが理想であり，その場合，ノイマン・ボトルネック等の現在の計算機が抱える根本的な問題の解消が期待できる．しかしながら，前章で述べたように様々な理由から実際的ではない．したがって，図 2.2 に示すように固定的なハードウェアを必要最小限にし，ソフトウェアとプログラマブル・ロジックの境界を変更することで最も効率的な処理を行うことが現実的である [3]．その場合に期待される効果を以下に示す．

1) 高性能化
   従来の汎用プロセッサは逐次的な命令実行を基本とし，データ幅（語長）ならびに演算器構成が固定であるため，メディア処理や通信処理が不得手である．一方，リコンフィギャラブル・コンピューティングはアプリケーション毎にハードウェア構成を最適化できるため，並列的なデータ処理，可変データ幅，多様な演算器構成，ならびにデータの流れの最適化が可能となり，高性能化が図れる．
2) 低コスト化，低消費電力化
   複数の処理を高速化したい場合にそれらすべてを ASIC に作り込むと，コストや消費電力などの点で要求仕様を満たせないことがある．しかし，それらすべてが同時に動作しないのなら，リコンフィギャラブル・コンピューティングでは再構成性という特徴を活かし必要時に実装することでハードウェア量を低減でき，低コスト化や低消費電力化が見込める．
3) 耐故障性の向上
   汎用プロセッサや ASIC の場合，一箇所でも故障すれば全体が機能不全に陥る可能性が高い．しかし，リコンフィギャラブル・コンピューティングでは再構成によって故

障箇所を回避することができ，それによって耐故障性の向上が図れる．

2.1.2 用語定義

本論文で使用する用語について以下に定義する．

- プログラマブル・ロジック：Programmable Logic
  本論文では使用者が書換え可能なロジックデバイス全体を指す．書換え頻度は 1 回以上，書換え単位はデバイス全体である．従来の FPGA や CPLD はこの範疇に入る．
- リコンフィギャラブル・ロジック：Reconfigurable Logic
  本論文ではリコンフィギャラブル・コンピューティングを主ターゲットとしたプログラマブル・ロジックを指す．プログラマブルロジックとの違いは，リコンフィギャラブル・コンピューティングに必要な機能や構造を備えている点である．
- コンフィギュレーション：Configuration
  回路を書換えること．コンフィギュレーション・データは，書換える回路データを意味する．プログラマブル・ロジックでは，回路の書換えは電源投入時に行われるか，または実装されている回路を停止している状態で行われる．回路の停止時の書換えでは，完了後にリセットされる．
- 部分再構成：Partical Reconfiguration
  デバイス全体ではなく，回路の一部分を書換えること．
- 実行時再構成：Run-Time Reconfiguration（RTR）
  回路の動作中に異なる回路に書換えること．書換え単位はデバイス全体，もしくは一部の回路．この場合は部分再構成機能が含まれる．
- 動的再構成：Dynamically Reconfiguration
  実行時再構成とほぼ同義．
- コンテキスト：Context
  回路情報全体，もしくはその一部分のこと．
- マルチコンテキスト：Multi-context
  複数の回路情報を意味する．マルチコンテキスト・デバイスというと，複数の回路情報を内部に保持し，それらを切り替えながら動作するリコンフィギャラブル・ロジック・デバイスの意味となる．
- 論理ブロック：Logic Block

プログラマブル・ロジックの構成要素の一つ．他にはコネクションブロック，配線，I/O などがある．論理ブロックには実装される回路の論理情報が格納される．

- ロジック・エレメント：Logic Element
  論理ブロック内で任意の関数を実装できる部分．FPGA では LUT やマルチプレクサ等がこれにあたる．論理ブロック内には，他にフリップフロック，リセット／クリア回路，カスケードチェーンなど，固定的な回路が含まれる．
- 粒度：Guranularity
  論理ブロックを構成するロジック・エレメント（LUT やマルチプレクサ）に実装可能な回路の規模を示す尺度．例えば，4 入力 1 出力の LUT の場合は，入力 4 ビットの任意の論理関数を構築できる．
- 粗粒度アプローチ：Coarse-grained Approache
  リコンフィギャラブルロジックを構築する上で，論理ブロックに比較的大きなロジック・エレメント（ALU など）を採用する方法．
- 細粒度アプローチ：Fine-grained Approache
  リコンフィギャラブル・ロジックを構築する上で，論理ブロックに比較的小さなロジック・エレメント（LUT やマルチプレクサなど）を採用する方法．
- クラスタ化：Clustering
  一つの論理ブロック内に複数の LUT などのロジック・エレメントを配置する手法．
- マッピング：Mapping
  回路情報をロジック・エレメントに割り当てること．

2.2 リコンフィギャラブル・コンピューティング方式の分類

2.2.1 利用形態による分類

リコンフィギャラブル・コンピューティングの利用形態は，プログラマブル・ロジックを計算処理のどの部分に適用し，どのような役割を持たせているかによって以下のように分類できる [4]．

(1) アタッチド・プロセッサ方式（汎用エンジン方式）
汎用コンピュータのベクトルファシリティのように，ワークステーション等のホスト計算機に付加して計算集約的な処理やプロセッサの不得手な処理を任せる方式で

ある．この方式はホストコンピュータに対して独立性が高く，いわゆるサーバ・クライアントモデルで運用する場合が多い．代表的なシステムには米 SRC の Splash[5], Splash2 がある．これについては後節のシステム開発事例で詳しく述べる．

(2) リコンフィギャラブル・コプロセッサ方式
汎用マイクロプロセッサと組み合せて利用する点に特徴があり，アプリケーションに適した命令の実装や入出力処理などをハードウェア化することで高性能化を図る方式である．この方式は，あくまで処理主体はマイクロプロセッサにあり，プログラマブル・ロジックはそれを補完することが役割となる．最近はマイクロプロセッサとプログラマブル・ロジックを 1 チップに収めた製品も発表されている．また，この方式は，ハードウェアとソフトウェアのトレードオフを考慮しながら最適設計を行うハードウェア／ソフトウェア・コデザインのためのプラットフォームとしても利用されている．

(3) リコンフィギャラブル・プロセッサ方式
プログラマブル・デバイスが主体に処理を行う方式である．処理方式としては，アプリケーションに特化した命令を適応的に実装して処理を行う方式，アプリケーションを完全に回路として実装する方式，さらには内部に RISC プロセッサや DSP などの回路を必要に応じて読み込む方式などがある．

2.2.2 デバイス構造による分類

リコンフィギャラブル・コンピューティングに使用されるデバイスは，細粒度リコンフィギャラブル・アーキテクチャと粗粒度リコンフィギャラブル・アーキテクチャに分類される [2]．以下にそれぞれの特徴について述べる．

(1) 細粒度リコンフィギャラブル・アーキテクチャ
細粒度リコンフィギュラブル・アーキテクチャは，従来の FPGA の構成要素と同じ Look Up Table (LUT) や Multiplexer を用いており，任意の回路が実装できるが，ディジタル信号処理でよく使われる ALU や MAC 演算などの演算回路を LUT で構成した場合，実装面積が大きく，動作速度も遅い．

(2) 粗粒度リコンフィギャラブル・アーキテクチャ
粗粒度リコンフィギュラブル・アーキテクチャは，ALU などの演算回路を最小構成要素としているため，演算処理の多いアプリケーションには向いているが，演算器間

表 2.1: 他のデバイスとの比較

デバイス	性能	コスト	消費電力	柔軟性
ASIC	高	高	高	低
DSP(Digtal Signal Processor)	中	中	中	中
MPU	低	低	中	高
細粒度リコンフィギャラブル・ロジック	中	中	低	高
粗粒度リコンフィギャラブル・ロジック	高	中	中	中

のグルー・ロジック（glue logic）でオーバーヘッドが生じるといった特徴がある．

他のデバイスとの比較

図 2.3 に他のデバイスとの動作の違いを示す．MPU や DSP はノイマン型のプロセッシングデバイスである．これらは処理内容を決定する命令列も処理対象のデータもメモリ上にあり，それらを毎回読み込んで処理を進める．ASCI は処理内容がデバイス上に回路として固定されており，データのみを毎回読み込んで処理する．一方，リコンフィギャラブル・ロジックは，処理内容をその処理の開始時に読み込み，ASIC と同様にデータのみを毎回読み込んで処理する．ただし，処理内容を変更したい場合は，ASIC と異なり回路の書換えによって，処理内容の変更が可能という特徴をもつ．これにより，柔軟性と性能のトレードオフを制御できる．

また，表 2.1 にリコンフィギャラブルロジックと ASIC，DSP，MPU との比較を示す．比較項目は性能，コスト，消費電力，柔軟性である．ASCI などの専用デバイスは性能，コストに優れるが，柔軟性が乏しい．逆に MPU は柔軟性には富んでいるが，性能やコストは低い．一方，リコンフィギャラブルロジックはその中間的な位置付けになる．細粒度リコンフィギャラブルロジックは比較的 MPU よりの特性を持ち，粗粒度リコンフィギャラブルロジックは専用デバイスよりの特性を持つ．

2.2.3 システム開発事例

本節では 2.2.1 節の利用形態による分類に基づいて，それぞれのカテゴリで代表的なシステムの開発事例について述べる．

まず，アタッチド・プロセッサ方式（汎用エンジン方式）の代表例として，米 Supercom-

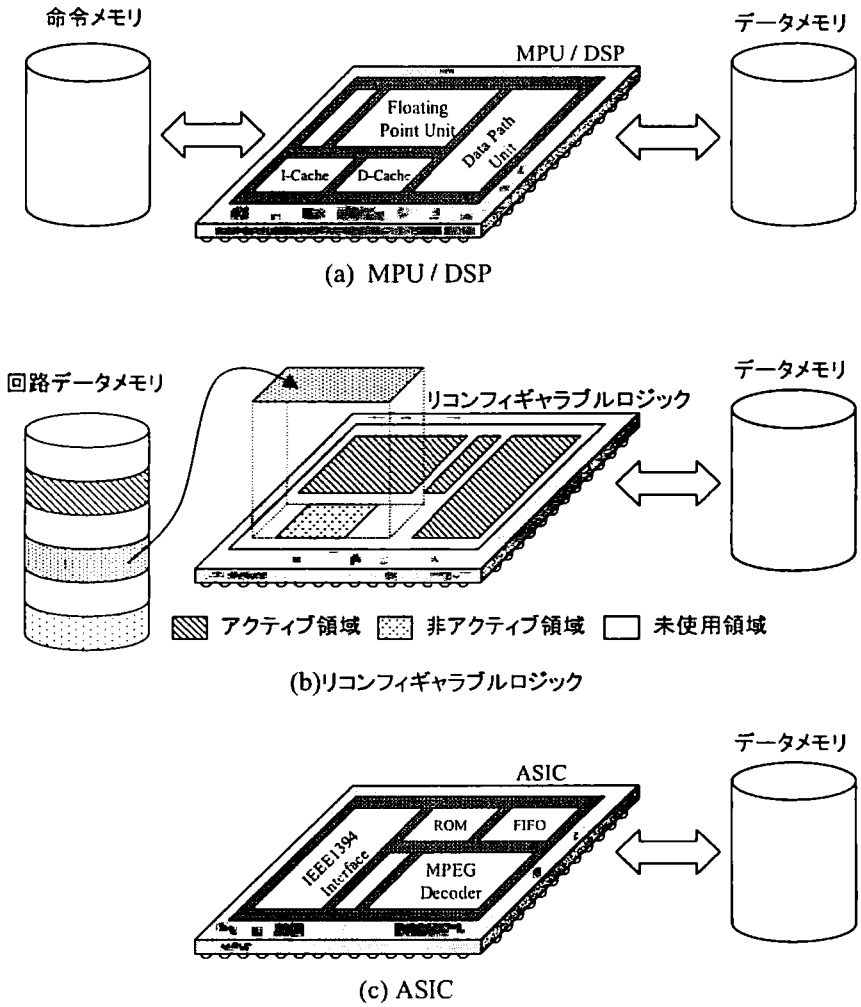

図 2.3: デバイス動作の違い

puting Research Center（SRC）で開発された Splash[5]，Splash2 が挙げられる．Splash は図 2.4 に示したように FPGA とメモリの組み合わせを直線状に並べた構造をしており，これにデータを流すことで，パターンマッチングのような汎用マイクロプロセッサが苦手な処理を高速に行うことができる．特に，開発の背景にはヒト・ゲノム・プロジェクトに必要な DNA パターンマッチング処理の高速化という動機があった．実際に，性能比較ではスーパーミニコン（VAX11/785）の 2,700 倍，スーパーコンピュータ（CRAY-2）の 325 倍，パターンマッチング専用マシン（P-NAC）の 45 倍の性能を達成し，一躍注目を集めた．高性能なスーパーコンピュータや専用マシンを性能で上回るだけではなく，類似問題の処理にも再構成性を活かすことで対処できることから，リコンフィギャラブル・コンピューティングの最初の成功例と言える．その後，SRC は Splash の I/O 性能の改善やよ

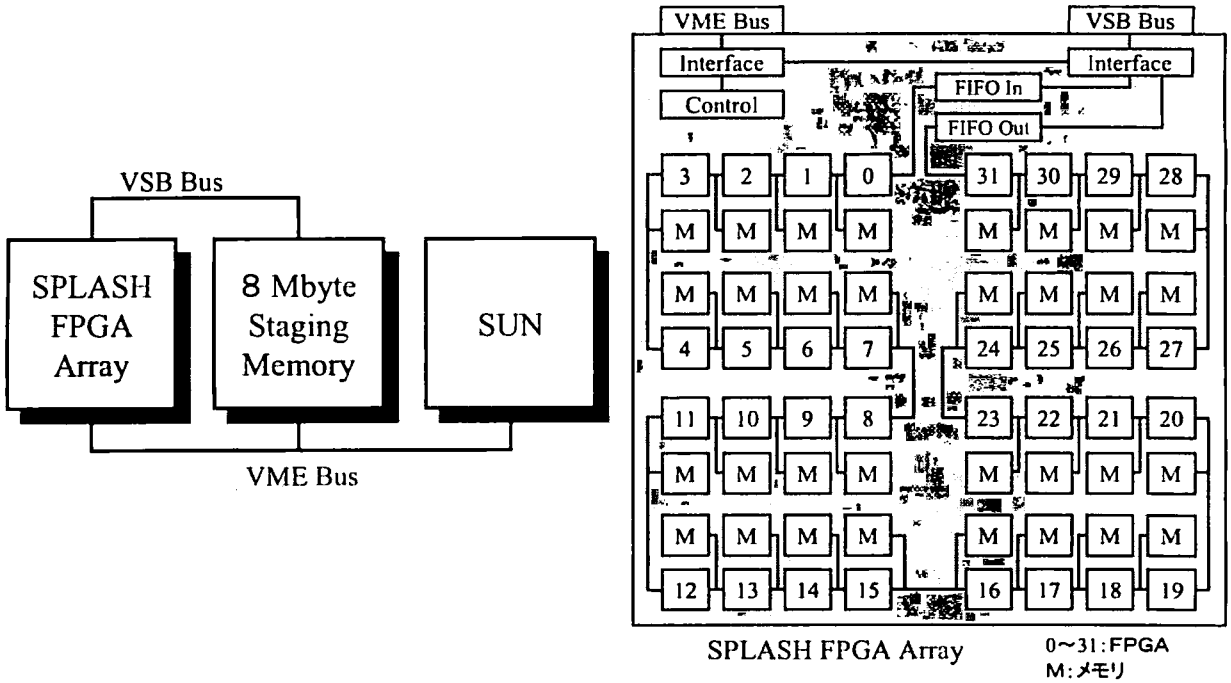

図 2.4: Splash の構成

り高い柔軟性を確保するために各 FPGA 間にクロスバ・スイッチを採用するなどの改良を施した Splash2 を開発した．これは米 Annapolis Micro Systems 社から WILDFIRE[6] という名で商用マシンとして発売されている．

しかしながら，SPLASH，SPLASH2 共に開発環境は，従来の FPGA 開発環境を流用しているため，アプリケーション開発は非常に困難である．また，従来のプログラマブル・デバイスを流用しているため，処理内容のサイズが使用デバイスの論理容量を上回った場合には，性能低下などが問題となる．

次に，リコンフィギャラブル・コプロセッサ方式の代表例としては，Brown University の PRISM-II[7] がある．図 2.5 に PRISM-II の構成を示す．PRISM-II は 32 bit の RISC プロセッサと既存の FPGA を用いたリコンフィギャラブル・アレイ，メモリ，そして周辺回路からなる．リコンフィギャラブル・アレイは，プロセッサに向かない処理に対して，アプリケーションに特化したコプロセッサとして動作する．図 2.6 に PRISM-II のソフトウェア開発環境を示す．PRISM-II は，まずはじめに C 言語で作成されたアプリケーションを解析する．そして，その中から抜粋した関数を回路として合成し，プログラマブル・ロジック上で実行する．この様な PRISM-II のハードウェア／ソフトウェア・コデザイン的なアプローチは，リコンフィギャラブル・コンピューティングの設計環境を考える上で

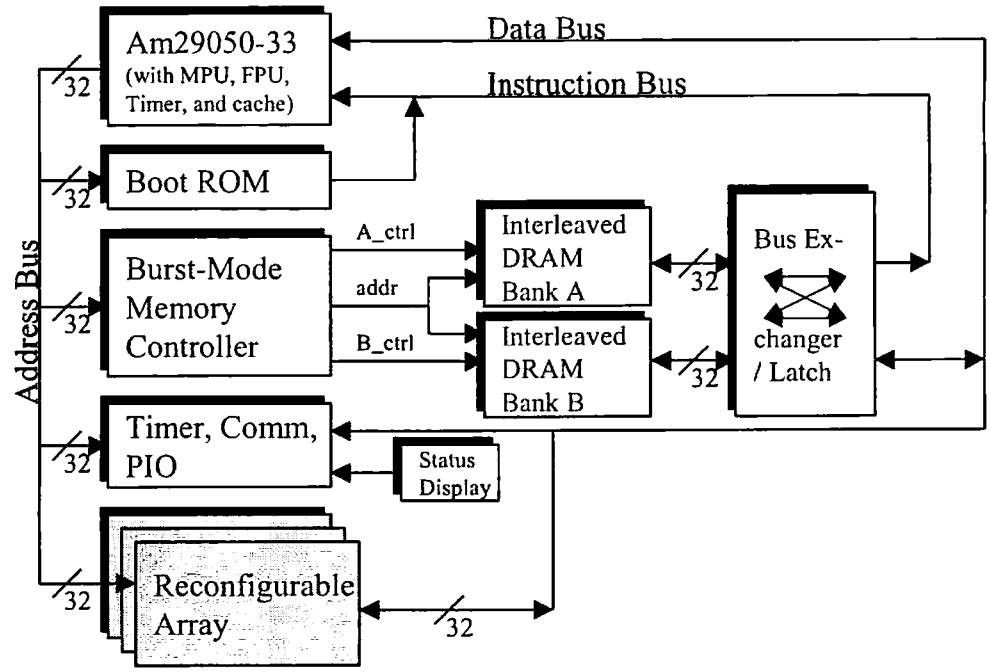

図 2.5: PRISM-II の構成

重要な研究といえる．また，線形近似関数やハミング距離を求める関数などを用いて評価した結果，メインのプロセッサ（Am29050, 33MHz）で実行するよりも，線形近似関数で約 9 倍，ハミング距離を求める関数で約 23 倍の性能向上が報告されている．

しかしながら，この性能評価には回路の再構成時間が考慮されていない．処理内容のサイズが使用デバイスの論理容量を上回った場合の対策が採られていないなどの問題も見られる．また，この方式ではプロセッサとプログラマブル・ロジックが協調して動作する必要から，各々の動作速度が異なる場合の議論も重要である．

そして，最後のリコンフィギャラブル・プロセッサ方式では，米 Brigham Young 大の DISC（Dynamic Instruction Set Computer）[8] が興味深い研究である．DISC プロセッサは，National Semiconductor 社の CLAy31 という部分再構成可能な FPGA を中心に構成され，図 2.7 に示したようにプロセッサがアプリケーションを実行している間に必要となるカスタム命令を動的に再構成するという非常に先進的なアプローチである．

DISC プロセッサは無限の命令セットをハードウェアに実装可能なことにより，適応性に優れている．ただし，DISC も動作原理はストアド・プログラム方式に基づくノイマン型プロセッサである．性能評価では，画像の平均化フィルタという応用に対してクロック換算で比較すると，汎用の命令モジュールで実行した場合と応用指向のカスタム命令での

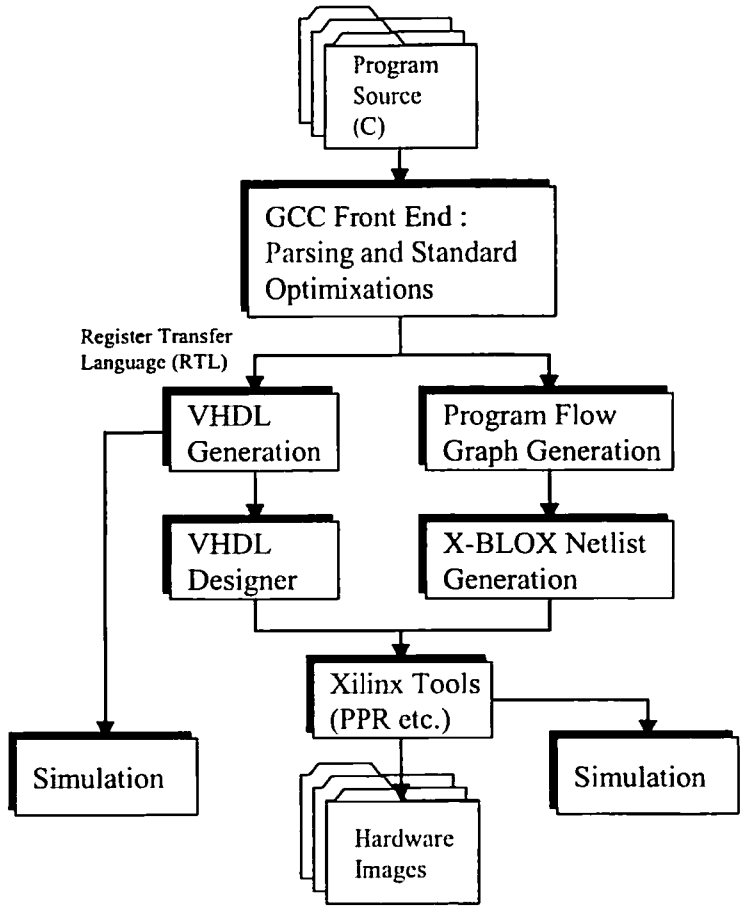

図 2.6: PRISM-IIC のソフトウェア開発環境

実行した場合の性能比は約 80 倍に達している．再構成に要する時間を考慮しても約 23 倍と優位性が報告されている．

しかし，動作速度が 7.5MHz と低いため，Pentium などの現実的なコモディティ・プロセッサとの性能比較では有効性に疑問が残る．

2.2.4 デバイス開発事例

本節ではリコンフィギャラブル・コンピューティング向きデバイスの研究事例についてまとめる．

2.2.2 節で述べたように，リコンフィギャラブル・コンピューティング向きデバイスのアーキテクチャは，細粒度リコンフィギャラブル・アーキテクチャと粗粒度リコンフィギャラブル・アーキテクチャに分類される．

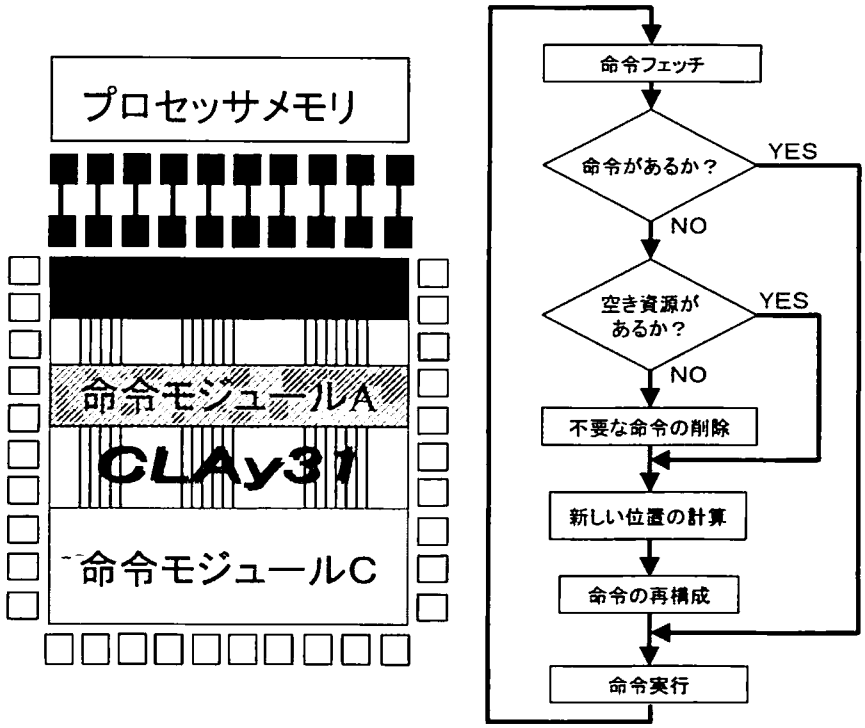

図 2.7: DISC の構成

細粒度リコンフィギャラブル・アーキテクチャの多くは，従来のプログラマブル・ロジックに対して，複数の回路を同一デバイス上に保持するマルチコンテキスト化を図ることで，集積度と再構成オーバーヘッドを改善している．MIT の DPGA（Dynamically Programmable Gate Arrays）[9] は，図 2.8 に示したようにロジックを構築する LUT とは別に回路データを複数保持するメモリを搭載し，それらの回路情報を処理内容が変わる毎に切り替えることで高集積を実現する．また，NEC も DRLE（Dynamically Reconfigurable Logic Engine）[10] と呼ばれる同じ考えの Reconfigurable Logic を提案し，試作している．しかしながら，回路の実装密度の観点からは従来の FPGA より改善が期待できるが，ASIC 等とは比較にならない上，動作速度もさらなる向上が必要である．

粗粒度リコンフィギャラブル・アーキテクチャは，多数の ALU などの演算器を任意の配線で接続することで，高いデータ並列や柔軟なパイプラインを構成可能なアーキテクチャである．CMU の PipeRench[11] は，Linear Arrays Architectures を採用し，パイプライン・アプリケーションを効率良く実行する．図 2.9 に PipeRench のアーキテクチャを示す．アプリケーションのパイプライン段数がデバイス内の段数より多い場合は，動的に再構成しながら動作することができる．また，動的再構成はサイクル毎に可能である．性能

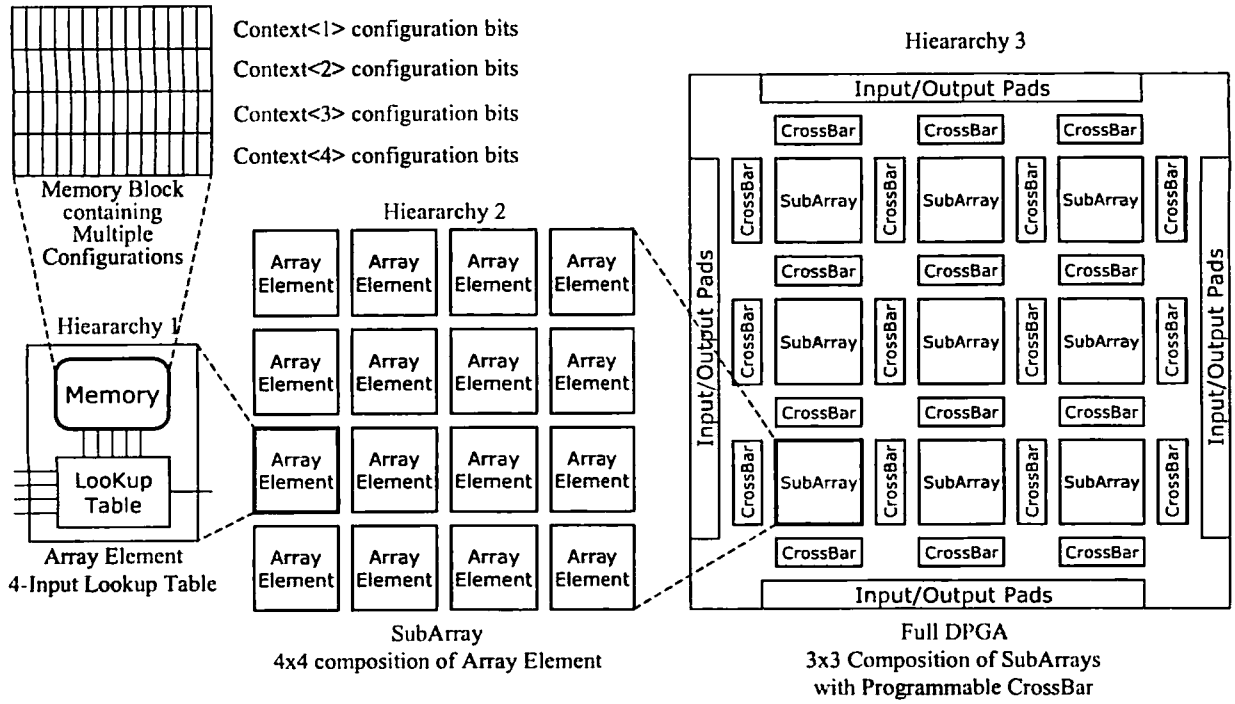

図 2.8: DPGA のアーキテクチャ

評価では，各 PE は 8 ビットの ALU と 8 本のレジスタを持ち，各 Stripe は 16 PE で構成した場合，ATR（Auto Target Recognition），Cordic，DCT，FIR などのアプリケーションで 300MHz の UltraSparc-II と比較して，10 倍以上（最大 189 倍）の速度向上されたと報告されている．しかし，アプリケーションが限定されるため，汎用プロセッシング用途としては使いにくい．

CHESS[12] は，Hexagonal Array 構造を持つ粗粒度リコンフィギャラブル・アーキテクチャである．図 2.10 に CHESS のアーキテクチャを示す．4 ビットの ALU と 2 つの 4 ビットレジスタを持つスイッチボックスと組み込みメモリから構成される．ALU は 16 種類の演算を行い，スイッチボックスは，パイプライン用のレジスタと隣接する ALU 間の接続を提供するためのメモリを持つ．0.35μm プロセスで 512 個の ALU を持つチップは 30mm^2 と報告されている．また，CHESS の制御は，各 ALU が他の ALU の機能を変更できることから，サイクル毎の再構成が可能である．ただし，部分再構成はサポートしていない．

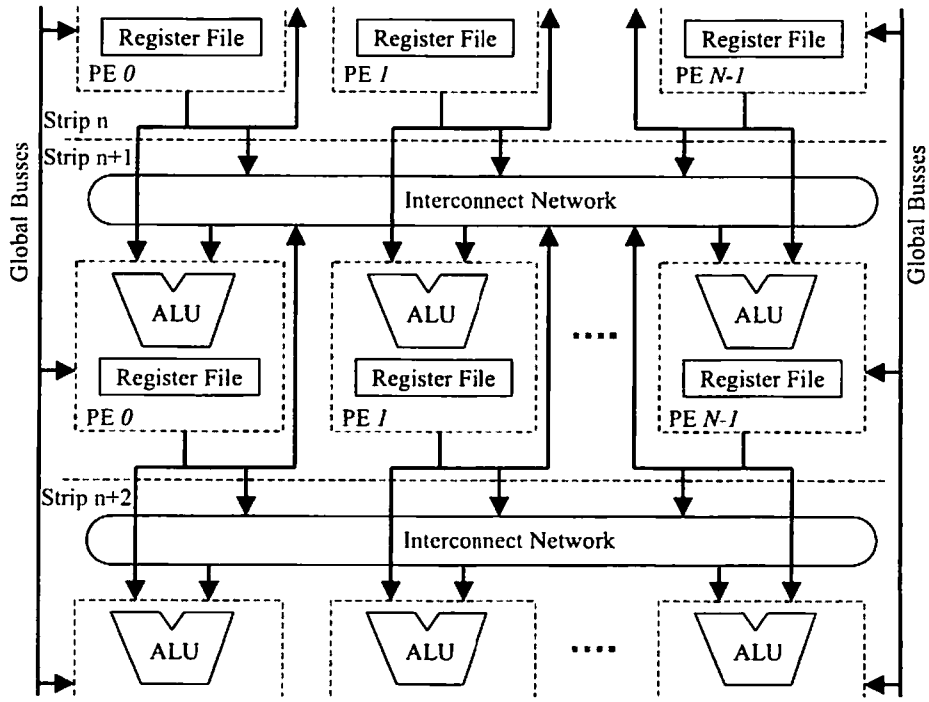

図 2.9: PipeRench のアーキテクチャ

2.3 リコンフィギャラブル・コンピューティングの課題

前章で述べたリコンフィギャラブル・コンピューティング・システムの研究開発事例から，現状のリコンフィギャラブル・コンピューティング・システムの抱える課題が明らかになってきている．これらの課題を集積回路分野ならびに計算機工学分野の視点からまとめる [4].

集積回路分野からの視点

(1) プログラマブル・ロジックは動作周波数が低い．とりわけ，SRAM 方式の FPGA はオン抵抗が大きく通常の LSI に比べて 1 桁は遅い.

(2) FPGA は通常の LSI に比べて面積が 10 倍から 100 倍も大きくなる．これが，適応性を求められながらもシステム LSI 内に FPGA を搭載するのを躊躇する理由にもなっている.

(3) 歩留まりがよくない.

(4) 発熱が大きい.

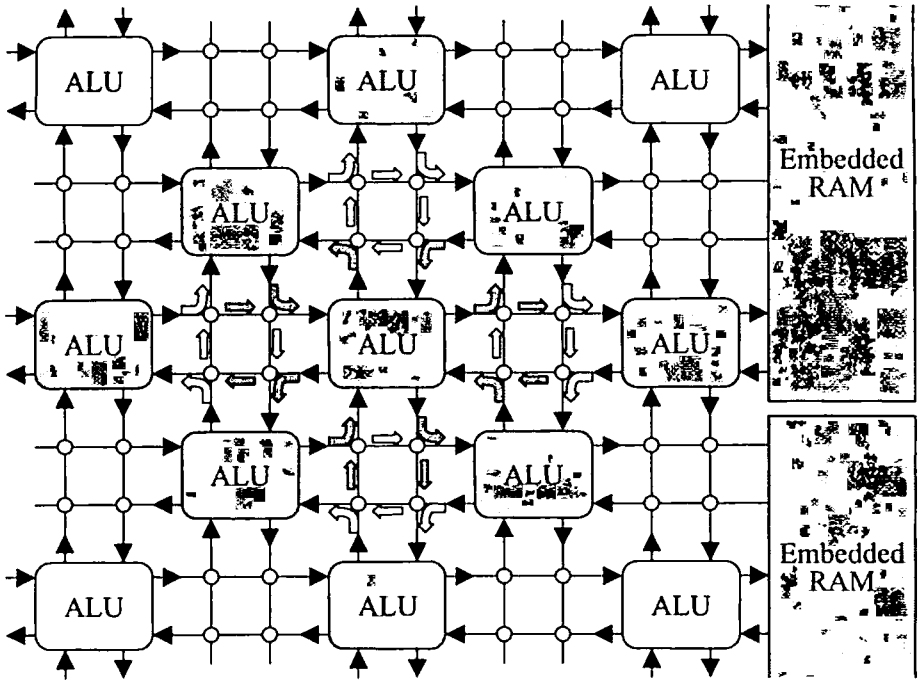

図 2.10: CHESS のアーキテクチャ

(5) LSI 設計においては，クリティカルパスが LSI 全体の動作周波数を決定する．同様に，再構成の過程で複数の回路モジュールを実装する場合には，最も動作周波数の遅いものがリコンフィギャラブル・コンピューティング・システムの性能を決めることになる．前節のシステム事例で示した DISC プロセッサでも，複数の命令モジュールを実装する場合に最も動作周波数の違いものが性能を決定した．

計算機工学分野からの拠点

(6) アプリケーションの開発環境が複雑過ぎる．現在は，ASIC の設計環境を流用しているのが実状であり，アプリケーションをハードウェア記述言語によって直接記述するものも多い．

(7) 性能評価では実行時間だけを評価しており，適正とはいえない．少なくとも，再構成時間は考慮に入れて評価すべきである．

(8) 現行のリコンフィギャラブル・コンピューティングシステムに使用されているプログラマブル・ロジックは容量も小さく，ハードウェア量に制限がある．問題サイズがシステムに搭載されているプログラマブルロジックの容量内に納まるならば効率よいが，納まらない場合は処理そのものが実行できなかったり，極端に性能が低下する．

(9) プログラマブル・ロジックのセットアップに時間がかかる．つまり，再構成レートが低く，リコンフィギャラブル・コンピューティングシステムの本領である書換えの頻度が高い応用では性能に大きな影響を及ぼす．

以上の議論を踏まえ，本研究では，第3章でリコンフィギャラブル・コプロセッサ方式を扱い，課題の（1），（2）および（9）を回避する方式を提示する．また，第4章でアタッチド・プロセッサ方式のシステムを示し，適用事例として暗号解析処理とレーダ信号処理を取り上げる．ここでは課題（7）に対しての一つの解を示す．そして，第5章で細粒度リコンフィギャラブル・アーキテクチャの問題点を整理し，その上で新しいアーキテクチャの提案と評価を述べる．ここでは課題（8）および（9）への対策について報告する．

第 3 章

リコンフィギャラブル・コプロセッサ方式

本章では，リコンフィギャラブル・コンピューティングの一方式であるプロセッサ，リコンフィギャラブル・コプロセッサ方式（プロセッサ，プログラマブル・ロジック混載方式）の研究結果について報告する．

3.1 概要

集積回路技術の進歩により，今まで複数の LSI で構成されていたシステム自体を一つの LSI に詰込むシステム・オン・チップ（以下 SoC と略す）が実現可能になってきた．しかし，その反面，大規模な LSI の開発には，肥大化する機能を効率良く設計するための設計手法の問題や，プロセスの微細化による寄生容量等の新しい問題点も明確になってきた．そこで，これらの問題を解決する手法として，CPU やメモリ等の従来の LSI クラスを組み込みコアとして利用するエンベデド・コア・ベースの設計手法が主流になると考えられ，現在，VSI アライアンス等でその標準化が進められている [13]．この場合，CPU 等のコアをブラックボックスとして捉え，フロアプラン上の負荷を減らすことで，SoC の設計コストを削減することが可能となる．

また，現在の計算機は，プロセッサの単体性能の改善と並行して，複数のプロセッサを使用した並列計算機が研究され実用化されている．ところが，現在の並列計算機は設計・製造コストが高く，それが普及の妨げの一つになっている．そこで，現在の計算機と同程度のコストで，複数のプロセッサを利用して性能の向上を実現する方式として， SoC 技術を使ったオンチップ・マルチプロセッサの研究開発が行われている．

しかし，オンチップ・マルチプロセッサは，単に複数のプロセッサを同一シリコン上に集積するだけでは，従来の並列計算機と同じ問題を持つ．すなわち，複数のプロセッサの

同期制御やコヒーレンス制御をソフトウェアで実行した場合の性能低下や，専用プロセッサを用いる場合は，そのプロセッサの設計コストの増加，互換性確保の困難さなどの問題である．

そこで本研究では，上記の SoC 技術によって実現されるオンチップ・マルチプロセッサの問題点を解決する方法として，プログラマブルロジックを同一シリコン上に配し，それをプロセッサ間の同期制御等に利用することで，既存のプロセッサを無変更で利用する方式の検討を行っている．これによりマルチプロセッサの設計コストや互換性の問題を回避し性能向上を実現する [14].

3.2 マルチプロセッサ制御への適用

3.2.1 オンチップ・マルチプロセッサの利点

社団法人電子情報技術産業協会（JEITA）の半導体技術ロードマップ専門委員会（STRJ）は，1999 年度版の報告書で西暦 2008 年には約 1 億トランジスタ以上が集積されると予測している [15]．また，同報告書では，1999 年時点でハイエンドの SoC をスクラッチから設計した場合 3,400 人月かかり，集積度が 1.4 倍／年で向上するのに対し，設計効率は 21%／年でしか向上しないため，設計工数およびコストの大幅な上昇が危惧されている．したがって，SoC では設計効率を上げるために組み込みコアの利用が必須と言える．また，プロセッサの性能向上要求も強いため，SoC とマルチプロセッサ技術の融合したオンチップ・マルチプロセッサは時代のニーズに沿った選択である．

さて，オンチップ・マルチプロセッサに実装可能なプロセッサ数は，プロセッサ・コアのトランジスタ数が百万〜1 千万 Tr.[15] と考えると，同一 LSI 上に数個程度である．したがって，オンチップ・マルチプロセッサは小規模並列計算機に近い構成になると考えられ，この様な制限下ではバス結合型マルチプロセッサが有望である．この構成は実装が容易かつ安価で，複数の LSI で構成されるシステムは既に商用機で実現されている．この構成の多くは主記憶共有型の UMA（Uniform Memory Access）モデルである．これは既存の計算機の延長として捉えることができ，OS やアプリケーションの移植等が容易なためである [16]．しかし，この構成はメモリアクセス時間の増大を招くといった問題があり，各プロセッサにキャッシュを搭載するなど，メモリのアクセス・レイテンシの隠蔽が不可欠である．さらに，共有メモリを用いた通信には同期機構が必須であり，この同期コストも性能に直接影響する．以上から，バス結合型マルチプロセッサ構成を採る場合の要

点は，

(1) メモリアクセス・レイテンシの削減
(2) 同期コストの削減

であるといえる．

また，オンチップ・マルチプロセッサを実現する場合，考慮すべき項目の一つに個々のプロセッサに割り当てる単位をどうするかという問題がある．例えば，複数あるプロセッサの各々に対して，

(a) プロセスを単位として割り当てる方式
(b) スレッドを単位として割り当てる方式
(c) 機械命令を単位として割り当てる方式

などの方式が考えられる．これらは並列計算機における並列性の粒度の問題と本質的に同じである．

ここでオンチップ・マルチプロセッサがバス結合型マルチプロセッサ構成を採用する場合を考えてみる．まず，(a) のプロセス単位で割り当てる方式では，システム全体で性能は向上するが，個々のプロセスのスループットは向上しない．さらに，各プロセッサ内のキャッシュのコヒーレンス制御の複雑さがコストの増加を招く可能性がある．また，(c) の命令を単位としてプロセッサに割り当てるのは専用プロセッサにならざるをえず，コアとして既存のプロセッサの利用が困難になる．

そこで本研究では，(b) のスレッドをプロセッサの処理単位として検討を進める．

次にスレッドを使用した時のメリットについてまとめる [17].

- スループット
  複数のプロセッサを持つコンピュータ・システムは同時に複数の実行点を提供する．それを利用できるマルチスレッドは，アプリケーション開発者にとってハードウェアの並列性を利用するための効果的な方法といえる．マルチプロセッサ上の並列性を活かしたマルチスレッドプログラムは，確実にプロセスのスループットを上げることができる．また，ユニプロセッサシステムの場合でも，マルチスレッド化によってプロセスは複数のブロックしているシステムコールを重複して処理することができる．したがって，マルチスレッドプログラムは，非スレッド（またはシングルスレッド）プログラムより，プロセッサのアイドル状態の時間を減少させ，結果的にスループットが向上が期待できる．

- 応答性
  非スレッド（またはシングルスレッド）プログラムにとって，プロセスの一部分をブロックすることは，プロセス全体をブロックすることにほかならない．その結果，その操作以外の作業が停止する．このようなアプリケーションは，マルチスレッド化によって，長い操作を必要とする部分を独立したスレッドに実行させることでアプリケーションをアクティブにすることが可能になり，ユーザにとっての応答性がよくなる．
- システムリソース
  複数の処理で共通のリソースを扱う場合，プロセスを単位とした処理は，スレッドを単位とした処理に比べて通信や同期にかかる管理コストが高い．これはプロセス間の独立性に起因する．また，プロセス間の通信やそれらの同期処理は，仮想記憶やカーネル状態，プロセス構造体全体を維持しなければならないため，プログラマにとって大きな負担を要求することになる．スレッドの場合は，同一プロセス内であることから，システムのリソースの必要量もプログラマの負担も少ない．
- 互換性
  マルチスレッドプログラムにおけるスレッドは，計算機に搭載されているプロセッサ数に無関係である．それゆえに，ユニプロセッサでもマルチプロセッサでも同一のプログラムを実行することができる．
- プログラム構造
  複数の処理を同時に実行しようとする従来のプログラムは，これらの処理間の調整をするために多くの複雑なコードを埋め込む必要がある．それに対して，スレッドプログラムは，比較的少ない単純なコード（スレッド）を複数用いることで処理を行う．このようなコードの単純化によってプログラムのリーダビリティは向上する．

一方，バス結合型マルチプロセッサを採る場合の要点である同期制御は，ハードウェアで実現した方が性能向上を見込める．しかし，専用のハードウェアを作り込んでしまっては，対応する制御部の変更に追従する柔軟性に欠ける．

そこで，最近，再書換え可能なデバイスとして注目を集めているプログラマブル・ロジックを用い，その上にスレッド制御機構を実装することで，効率の良いスレッド切替を実現する．本研究が想定しているオンチップ・マルチプロセッサのイメージは図 3.1 に示した通りである．各プロセッサ・コアにはキャッシュは搭載せず外部に共用の MMU およびキャッシュを持ち，データ・アクセスに関してはカスタムハードウェア領域（プログ

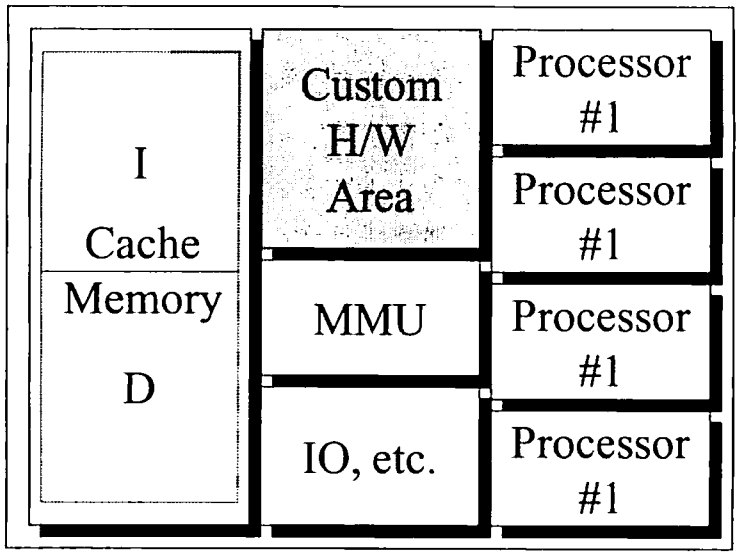

図 3.1: オンチップ・マルチプロセッサのイメージ

ラマブル・ロジック領域）とプロセッサ・コアとは同等であることを想定している．

スレッド制御機構をハードウェア化しプログラマブル・ロジックに実装することによる利点は，

- プロセッサはスレッドスケジューリング等のスレッド管理を行わないため，スレッドの切替え時間の削減が可能になる．
- スレッド制御はプロセッサと並列に実行するため，プロセッサの使用効率を高めることができる．
- ソフトウェアのように制御の命令やデータでキャッシュを汚さないため，キャッシュの利用率を上げ，キャッシュミスによる速度低下の軽減ができる．

などが挙げられる．これら性能に関する利点の他に，ソフトウェアのようなバージョンアップが可能なことや，プログラマブルロジックが搭載されていないプロセッサ上では，ソフトウェアのライブラリを利用することで同一プログラムが実行可能なこと（互換性の確保）が実現できる．

3.2.2 リコンフィギャラブル・コンピューティングの問題点

しかし，プログラマブル・ロジックを制御に用いた場合，従来からリコンフィギャラブル・コンピューティングが抱えていた代表的な 3 つの課題について検討する必要がある．

動作周波数の問題　リコンフィギャラブル・コンピューティングは，プロセッサに対して一桁程度遅いプログラマブル・ロジックで処理を行うため，処理をハードウェア化しても必ずしも高速化につながらないという問題がある．今回のスレッド制御機構では，アプリケーションの処理はプロセッサが行い，プログラマブル・ロジックでは各スレッドの管理とスケジューリングをプロセッサの処理と並列で動作する．したがって，動作中のスレッドが切替わるまでに完了していれば十分であることから，処理速度の問題はさほど影響しないと考えられる．

再構成時間の問題　最近のプログラマブル・ロジックは再構成時間が短縮してきているが，それでもまだ処理時間に対して十分高速だとはいえない．スレッド制御機構は，プログラマブル・ロジックの部分再構成機能を用いることで，システムの立ち上げ時に基本的な部分を再構成し，プログラムの実行時には固有の部分だけに限定することで再構成時間を最小化する．したがって，再構成時間の問題を軽減させることができる．ただし，プログラマブル・ロジックには部分再構成機能が必須となる．

ハードウェア制限の問題　プログラマブル・ロジックに載せる単位処理の論理規模が予測不可能な場合，リコンフィギャラブル・コンピューティングでは実行不能に成り得る．それに対して，本方式は個々制御回路の論理規模が確定しており，全体の論理規模も予測可能である．したがって，アプリケーションは，それらのどの機能をどのくらい使用するかという問題に帰着し，論理規模が超過した場合においては，機能モジュール単位の部分再構成で十分対処可能である．

以上の議論から，本研究で提案する方式は，従来からリコンフィギャラブル・コンピューティングが抱えていた問題による性能低下等の影響が少ない方式と考えられる．

3.3　システム構成

3.3.1　スレッド制御関数および同期制御関数の概要

本研究で対象としたスレッド制御機構は，POSIX 1003.4a / D6[18] に準拠して作成された文献 [19]，[20] を参考にし，機能を限定して必要不可欠な部分のみを実装する．これは本研究の目的がハードウェアによるスレッド制御の基本構造の確立とその論理量の調査を目的にしたためである．また，ソフトウェア版との互換を確保するためにライブラリ形式で呼び出せるように実装した．実装対象とした関数を表 3.1 に示す．

これらはスレッドの生成／終了／切替えに必要な基本関数と同期制御に使用される

表 3.1: プロトタイプ実装したスレッド制御関数

関数名	機能
pthread_init()	スレッドの初期化
pthread_create()	スレッドの生成
pthread_exit()	スレッドの終了
pthread_yield()	スレッド実行権の譲渡
pthread_detach()	スレッドの解放
pthread_join()	スレッドの結合
pthread_terminate()	スレッドの強制終了
pthread_self()	自スレッド ID の取得
pthread_mutex_init()	mutex の生成
pthread_mutex_lock()	mutex のロック
pthread_mutex_unlock()	mutex のアンロック
pthread_mutex_trylock()	mutex のトライロック

mutex オブジェクト操作関数である.

また，スケジューリング・アルゴリズムは直接ハードウェア化するため，実行するプログラム内で固定とする．したがって，ソフトウェア・インタフェース上にはスケジューリング操作関数は存在しない．そのためスケジューリング・アルゴリズムを動的に変更するプログラムには対応していない．アトリビュート機能，プロセス制御，シグナル，スレッドのキャンセル等に関しては未実装である.

3.3.2 スレッド制御関数の詳細

ここでは前小節で概要を述べたマルチスレッド制御関数について，各関数のインタフェースおよび動作の詳細を述べる．実装方法については，動作の先頭の記号が．○のものはS／Wによる実装，◎は一部H／W，●はH／Wによる実装である.

スレッドの初期化　void pthread_init(pthread_t *th)

○スレッドカーネル構造体の初期化

○スレッド 0（main）の状態書込み

●スレッドのスケジューリング

スレッドの生成　int pthread_create(pthread_t *th, pthread_attr_t *attr,

pthread_func_t func, int ac, any_t av)

○スレッド構造体の初期化
○スレッドの状態書込み
●スレッドのスケジューリング

スレッドの終了　void pthread_exit(any_t stat)
○自スレッド ID の取得：pthread_self() コール
○スレッドの結果書込み
○スレッドの強制終了：pthread_terminate() コール
◎コンテキストのセーブ
●スレッドのスケジューリング
◎コンテキストのリストア

スレッドの強制終了　void pthread_terminate()
○自スレッド ID の取得：pthread_self() コール
○スレッドの状態書込み

自スレッド ID の取得　pthread_t *pthread_self()
○ pthread_kern.k_pthread_self の読込み

スレッド実行権の譲渡　void pthread_yield(any_t arg)
◎コンテキストのセーブ
●次スレッド ID の取得
●スレッドのスケジューリング
◎コンテキストのリストア

スレッドの分離　int pthread_detach(pthread_t th)
○スレッドの状態書込み
○リソースの解放処理

スレッドの結合　int pthread_join(pthread_t *th, any_t stat)
○スレッドの状態読み込み
○スレッドの状態のチェック
○スレッド実行権の譲渡：pthread_yield() コール
○リソースの解放処理

3.3.3 同期制御関数の詳細

ここでは前小節で概要を述べた同期制御関数について，各関数のインタフェースおよび動作の詳細を述べる．実装方法については，動作の先頭の記号が．○のものはS／Wによる実装，●はH／Wによる実装である．

mutex のロック　int pthread_mutex_lock(pthread_mutex_t *mx)

● mutex のロック

○自スレッド ID の取得：pthread_self() コール

○スレッドの状態書き込み

○スレッド実行権の譲渡：pthread_yield() コール

●スレッドキューの更新

mutex のアンロック　int pthread_mutex_unlock(pthread_mutex_t *mx)

○自スレッド ID の取得：pthread_self() コール

○ mutex のオーナーチェック

○スレッドの状態書き込み

● mutex のアンロック

●スレッドキューの更新

mutex の試行ロック　int pthread_mutex_trylock(pthread_mutex_t *mx)

● mutex のロック

○自スレッド ID の取得：pthread_self() コール

○スレッドの状態書き込み

3.3.4 データ構造体

次に本研究で設計したマルチスレッドライブラリ用のデータ構造体について述べる．データ構造体は，スレッドカーネル構造体（kernel_t），スレッド構造体（pthread_t），mutex 構造体（pthread_mutex_t），およびスレッドキュー構造体（pthread_queue_t）からなり，それぞれハードウェア（プログラマブル・ロジック）上に実装される部分とソフトウェア（メモリ）上に実装される部分とを持つ．

スレッドカーネル構造体（kernel_t）の内訳を表 3.2 に示す．

スレッドカーネル構造体はすべてのスレッド共有される情報を保持する構造体であるた

表 3.2: スレッドカーネル構造体（kernel_t）

Offset	メンバ	コメント	Loc.
0x00	pthread_t *k_pthread_self;	currently running thread	H/W
0x04	char *func_pth_create;	pth_create function address	H/W
0x08	char *func_pth_yield;	pth_yield function address	H/W
0x0c	char *func_pth_exit;	pth_exit function address	H/W
0x10	char *func_pth_detach;	pth_detach function address	H/W
0x14	char *func_pth_join;	pth_join function address	H/W
0x18	char *init_stack_base;	initial stack base	S/W

め，プログラム（プロセス）中で唯一である．本研究で実装したライブラリでは，メンバのほとんどをプログラマブル・ロジック上で実装している．この構造体のメンバは，実行中のスレッドを表す k_pthread_self と各スレッド制御関数へのポインタ，そしてスレッド用のスタックポインタの初期値を示す init_stack_base からなる．

次にスレッド構造体（pthread_t）の内訳を表 3.3 に示す．

表 3.3: スレッド構造体（pthread_t）

Offset	メンバ	コメント	Loc.
0x00	int errno;	thread-specific errno	H/W
0x04	int ret;	return value (EINTR → -1)	H/W
0x08	char *stack_base;	bottom of run-time stack	H/W
0x0c	int state;	thread state	H/W
0x10	pthread_func_t func;	actual function to call upon activation	H/W
0x14	any_t result;	return value of above function	H/W
0x18	int base_prio;	base priority of thread	H/W
0x1c〜0x28	context_t context;	context save area	H/W
0x2c	any_t arg;	argument list to above function	S/W
0x30	(Reserved)	(Reserved)	S/W
0x34	int max_ceiling_prio;	Max of ceiling prio among locked mutexes	S/W
0x38	int new_prio;	New Priority	S/W
0x3c	struct pthread *next;	Next pointer for mutex queue	S/W

スレッド構造体はスレッド固有のデータを格納する．スレッド固有データとは，一つのスレッドが実行するすべての関数に対してグローバルな変数で，他のスレッドに対してそ

れらはの同じ変数とは独立である変数である．例えば，システムコール時のエラーが二つのスレッドで起こった場合，エラー変数はそれぞれのスレッドでどのようなエラーが起こったかを識別するために，それぞれのスレッドで独立している必要がある．したがって，スレッド構造体はスレッド毎に持つ．この構造体メンバは，エラー番号（errno）やリターン値（ret）などのようなスレッド実行に必要な変数と，スタックベースアドレス（stack_base）やスレッドの先頭アドレス（func）などのようなスレッドの初期設定が格納される変数，そして，スレッドコンテキストの退避領域からなる．また，実行中のスレッドに関しては，この構造体のメンバの多くをプログラマブル・ロジック上に配置している．休眠中のスレッドの場合は，メモリ上に退避している．

また，mutex 構造体（pthread_mutex_t）の内訳は表 3.4 の通りである．

表 3.4: mutex 構造体（pthread_mutex_t）

Offset	メンバ	コメント	Loc.
0x00	int lock;	lock valiable	H/W
0x04	int trylock;	trylock (= lock valiable)	H/W
0x08	int flags;	queue status flag	H/W
0x0c	(Reserved)	(Reserved)	H/W
0x10	struct pthread *owner;	lock owner pth	S/W
0x14	struct pthread_queue queue;	pthread address of queue head	S/W
0x18	(Reserved)	(Reserved)	S/W

mutex 構造体のメンバは，ロック変数（lock）と状態フラグ（flag），ロックしたスレッドを保持するオーナー（owner），そしてロックに失敗したスレッドが格納されるスレッドキューへのポインタ（queue）からなる．trylock 変数は，lock 変数と同じであるが，mutex_trylock() 関数からの呼出しと mutex_lock() 関数からの呼出しを区別するために，今回作成したスレッド制御ライブラリに追加した．純粋にソフトウェアだけからなるライブラリには不要なものである．

mutex 構造体内で使用するスレッドキュー構造体（pthread_queue_t）を表 3.5 に示す．

表 3.5: スレッドキュー構造体（pthread_queue_t）

Offset	メンバ	コメント	Loc.
0x00	struct pthread *head:	head of pthread queue	S/W
0x04	struct pthread *tail;	tail of pthread queue	S/W

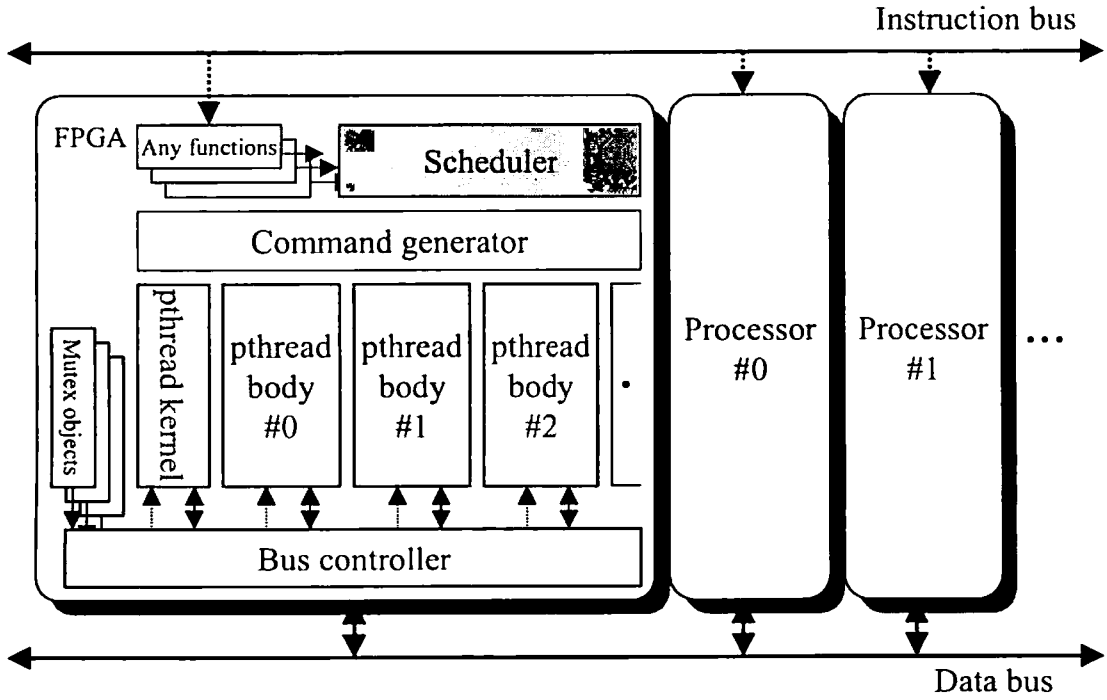

図 3.2: スレッド制御機構のハードウェア構成

スレッドキュー構造体は，mutex のロックに失敗したスレッドのキューを作成するためのデータ構造体である．メンバはキューの先頭のスレッドを示す head とキューの末尾を表す tail からなる．

3.3.5 ハードウェア構成

実際のハードウェア化は以下の 3 方針で実施する．

(1) プロセッサと並列実行可能な処理
(2) ハードウェア化により実行速度が向上する処理
(3) ハードウェア量が極端に増加しないリソース

スレッド制御機構のハードウェア構成を図 3.2 に示す．ここでは 2 プロセッサの場合を示しているが，プロセッサ数が増えても基本的な構成は変わらない．

スレッド制御機構のハードウェア構成は，基本的に 3 つの部分から成り立っている．一つ目はスレッド制御に必要不可欠なモジュール，pthread kernel（スレッドのカーネル情報の保持とその制御），pthread body（実行中スレッドのスレッド情報の保持とその制

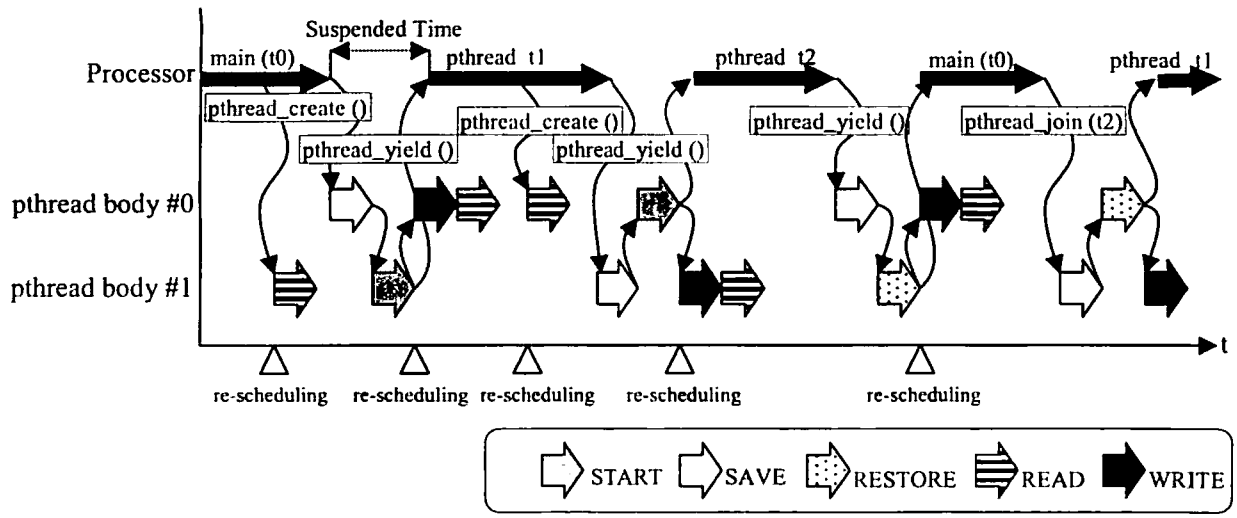

図 3.3: マルチスレッド動作例（1 Processor, 3 threads）

御），command generator（制御コマンドの発行），bus controller（バス制御）の 4 種 6 モジュールである．これらはアプリケーションに依存しない構造であるため，起動時に一度ロードすれば良い．ただし，pthread body はプロセッサ数＋ 1 個以上必要である．これはプロセッサの動作の裏で次に実行されるスレッドの準備を行うためである．二つ目はアプリケーションで使用されているスレッド制御関数の実行を検出するモジュール（any functions，スレッド関数の実行検出）である．これはアプリケーション毎に再ロードする必要はないが，以前のアプリケーションで使用してない場合はロードする必要があるモジュールである．三つ目は完全にアプリケーションに依存し，起動するスレッド数や使用するオブジェクト数によって変化するモジュールである．すなわち，scheduler（スレッドのスケジューリング）と mutex objects（mutex データ構造体の制御）である．これはアプリケーション毎に再ロードが必要となる．

次に各モジュールの動作の詳細について述べる [21].

any functions　これは実際には各スレッド制御関数のコールを検出するモジュールの集合である．これは pthread kernel 内に保存された各スレッド制御関数の命令アドレスをモニタすることでプロセッサでコールされたことを検出する．検出された情報は，command generator と scheduler に報告される．

command generator　これは any functions からの情報と pthread body から出力される現在実行中のスレッド状態，および scheduler が指定したスレッドから pthread body へのコ

マンド送出を行う．コマンドは START, SAVE, RESTORE, WRITE, READ の 5 種類である．START はスレッドの起動，SAVE はスレッドの切替時に現在プロセッサが実行しているスレッドコンテキストの pthread body への一時保存を行う．RESTORE は pthread body に準備されているコンテキストのプロセッサへのリストア，WRITE は pthread body 内のスレッドコンテキストのメモリへの書出し，READ はスレッドコンテキストの pthread body 内レジスタへの読込みを行う．

pthread body　これは内部にスレッドコンテキスト用レジスタを持ち，command generator のコマンドに従ってプロセッサおよびメモリとのアクセスを行う．コンテキスト用レジスタは，プロセッサの要求に従って読み書き可能である．プロセッサ内の汎用レジスタや PC の書込みは，プロセッサ停止信号をアサートしたあと専用の信号線で行う．また，pthread body は，実行中のスレッド状態を command generator に出力する．

pthread kernel　これはスレッドに共通の情報を格納するカーネル構造体の一部をレジスタで持つ．このレジスタはプロセッサからメモリと等価に扱われる．また，pthread kernel は，スレッド制御関数の命令アドレスを各関数検出モジュールに出力する．

scheduler　これは各スレッド制御関数検出モジュールからの信号を受け取り，その信号の指示にしたがってスケジューリング・エントリに追加や削除，または再スケジューリングを実行する．内部には各スレッドの状態を保持するレジスタを持ち，それにしたがってスケジューリングを行う．

mutex objects　これはロックと解除の 2 状態しか持たないセマフォアである．このモジュールは mutex の値を格納するレジスタとロック，アンロック等の制御回路を持つ．このレジスタはプロセッサからメモリと等価に扱われる．mutex は，プロセッサから読出されるだけでロックされ，ロックしたスレッドが書込むことでアンロックされる．また，mutex ロック時のアクセスによるスレッドキューの追加や，アンロック時のキューからの削除も，このモジュールで制御して行う．

bus controller　これは各モジュールからのリクエストのアービトレーションを行う．

3.3.6　スレッド制御機構の動作

次に簡単な動作フローを図 3.3 に示す．この例は 1 つのプロセッサ上で 2 つのスレッドが pthread_create() 関数で生成され，各スレッド実行中に pthread_yield() 関数および pthread_join() 関数によって明示的にスレッドを切替えながら実行する例である．

図 3.3 から分かるように，スレッドのスケジューリング，次スレッドの準備はプロセッ

サ動作に影響されずに，プロセッサの処理のバックグランドで行う．この時，プロセッサとメモリアクセスの競合が発生する可能性があるが，その場合はプロセッサを優先させている．これはスレッド制御ハードウェアはプロセッサの処理を妨げないようにするためである．また，プロセッサが停止する時間は，現スレッドコンテキストの保存と次スレッドコンテキストの復帰間にとどめた．これによりソフトウェアで実行するスレッド切替え制御に比べて処理時間の短縮が期待できる．

3.4 評価結果

3.4.1 動作検証

3.3 節で述べたスレッド制御機構は，すべて論理合成可能なハードウェア記述言語 Verilog–HDL で記述した．動作確認に必要なプロセッサ・コアは，DLX アーキテクチャ [22] を採用した DLX–FPGA[23] の仕様に基づいて Verilog–HDL で記述した．これを採用した理由は DLX アーキテクチャがシンプルでかつ標準的な RISC アーキテクチャであるためである．また，スレッド制御機構のソフトウエアインタフェースであるマルチスレッドライブラリもこの DLX–FPGA 用の C コンパイラ（dlxcc），およびアセンブラ（dlxasm）を用いて作成した．dlxcc と dlxasm は文献 [24] を元に，DLX–FPGA 仕様に変更したものである．

また，実際のシミュレーションではプログラマブル・ロジック領域もプロセッサと同列に扱い，シミュレータ上で実行するプログラムに必要なモジュールは，あらかじめ組み込んであるものとして，プログラマブル・ロジック領域のセットアップ時間の扱いは省略した．動作検証は，全てのスレッド制御機構が動作する簡単なテストプログラムを数本用意し，Verilog-XL シミュレータを用いて行った．その結果，期待通りに動作していることを確認した．

表 3.6 は，スレッド制御の各処理に要する時間をクロック数で示したものである．表 3.7 は，mutex 制御の各処理に要する時間をクロック数で示したものである．コメントで “プロセッサ停止” となっている処理は，プロセッサの動作を停止して処理を行うことを意味する．“バス競合” となっている処理は，プロセッサの動作と並列で実行される処理である．表中には最短クロック数を示した．実際は競合するため，それ以上の処理時間がかかる．また，ここで示したクロック数は全てプログラマブル・ロジック部に供給されるクロックである．したがって，プロセッサのクロックと同一でない場合，プロセッサの停

表 3.6: スレッド制御の処理時間

処理名	クロック数	コメント
START	2	プロセッサ停止
SAVE	4	プロセッサ停止
RESTORE	3	プロセッサ停止
WRITE	11～	バス競合
READ	11～	バス競合

表 3.7: mutex 制御の処理時間

処理名	クロック数	コメント
LOCK（Suc.）	2～	バス競合
LOCK（Fail）	3～	バス競合
UNLOCK	4～	バス競合

止期間はその比率倍される.

スレッド操作の各処理の内訳は以下の通りである.

- **START:** thread body からプログラム・カウンタとスタック・ポインタを汎用レジスタへ書込み.
- **SAVE:** 汎用レジスタからプログラム・カウンタとスタック・ポインタ，フレーム・ポインタを thread body 内の専用レジスタへの読込み.
- **RESTORE:** thread body からプログラム・カウンタとスタック・ポインタ，フレーム・ポインタの汎用レジスタへの書込み.
- **WRITE:** thread body 内のスレッド構造体レジスタへの書込み.
- **READ:** メモリから thread body 内のスレッド構造体レジスタへの読込み.

上記以外のスレッド切替えに伴うプロセッサのレジスタの退避は，すべてソフトウェアで行っている.

mutex 操作の各処理の内訳は以下の通りである.

- **LOCK（Suc.）:** mutex 値のメモリへの書込み.
- **LOCK（Fail）:** mutex 値のメモリへの書込みと，スレッド・キューへの追加.
- **UNLOCK:** mutex 値のメモリへの書込みと，スレッド・キューからの削除とリンクの張り替え.

表 3.8: スレッド制御機構の回路規模

モジュール名	CLBs	FFs	Cells
any functions	11	2	11
command generator	53	18	53
pthread body	621	352	653
pthread kernel	92	160	92
scheduler	36	16	124
mutex object	86	54	150

全体的に処理時間が少ないのは，ハードウェア化にあたっての方針として，“ソフトウェアで実行してもハードウェアで実行してもさほど処理時間が変わらないものはソフトウェアで行う”という理由からである．実際にハードウェアで処理しているスレッドコンテキストの切替えは，全スレッド切替処理の一部分であり，残りはソフトウェアで処理している．

完全にハードウェアで行っている処理は，スレッドのスケジューリング，mutex のロック，アンロックに付随するスレッドキューの制御である．これらの処理は，プロセッサと完全に並列で実行される．

なお，メモリとのアクセスは，同一チップ上にあることを仮定して，1 データの書込み，読出しは，全て 1 クロックで行えるものとして計測した．

3.4.2 回路規模

表 3.8 にスレッド制御機構の各モジュール毎の回路規模を示した．本来，スレッド制御機構は SoC 上に配した部分書換え可能なプログラマブル・ロジック部に実装することから，特定の FPGA 製品をターゲットにしてはいない．しかし，回路規模を見積もる必要から，今回はターゲットのプログラマブル・ロジック部を Xilinx 社の XC4000e シリーズ [25] とした．回路規模は，論理セル（CLB）数とフリップ・フロップ（FF）数および使用セル（Cell）数で示した．

また，any funtions はスレッド制御関数を検出するモジュール一つ分とした．実際には使用しているスレッド制御関数の数だけ必要となる．pthread body は 1 つ分の回路規模である．先に述べたようにプロセッサの個数より多く必要であるため，実際には最少でもこの回路規模の（プロセッサ数＋ 1）倍になる．mutex object と scheduler は，使用する

mutex 数やスレッド数，スケジューリング・アルゴリズムが異なるアプリケーションでは回路規模が変化するため，ここでは参考値として，mutex object は一つの mutex オブジェクトだけを含む場合を，scheduler は 4 スレッドをラウンドロビンでスケジューリングするものを示した．

論理合成の制約条件は，全てのモジュールの動作周波数を 33MHz とした．論理合成ツールは，Synopsys 社の FPGA Compiler v3.3b を用いた．また，表 3.8 で示した結果は，全てのモジュールが制約条件を満たして論理合成を完了している．表 3.8 から分かるように，pthread body モジュールの回路規模が際だって大きい．これは内部にスレッドコンテキストのデータをフリップ・フロップで持っているためである．

mutex の使用数に対する mutex object モジュールの回路規模の変化を図 3.4 に示した．図から分かるように，FF 数は mutex の個数に比例，CLB 数および CELL 数は，論理合成の最適化のばらつきはあるものの，使用個数が多くなるにしたがって mutex の個数に概ね比例する傾向がある．スケジューリングする pthread 数に対する scheduler モジュールの回路規模の変化を図 3.5 に示した．これも mutex の場合とほぼ同じ傾向を示している．

以上の結果から，今回プロトタイプ実装したスレッド制御機構は，プロセッサ数，およびプログラムが使用するスレッド数，mutex 数，使用したスレッド制御関数の種類の数から，その実行に必要な回路規模が式（3.1）から推定可能である．

$$
\begin{aligned}
\text{総 } CLB \text{ 数} = {} & 11(any\ functions) \times (\text{検出する関数の数}) \\
& +621(pthread\ body) \times (PE \text{ 数} + 1) \\
& +53(command\ generator) \\
& +92(pthread\ kernel) \\
& +\alpha\ (scheduler) \\
& +\beta\ (mutex\ objects)
\end{aligned}
\tag{3.1}
$$

α および β はグラフ（図 3.5. 図 3.4）より抽出する．

3.4.3 性能比較

次に純粋にソフトウェアだけで作成されたライブラリと性能比較した結果について述べる．比較した関数は，スレッド制御を行う pthread_init(), pthread_create(), pthread_exit(), pthread_yield() の 4 関数と，mutex 制御を行う pthread_mutex_lock(), pthread_mutex_unlock() の 2 関数である．これらの関数はいずれも内部に今回ハード

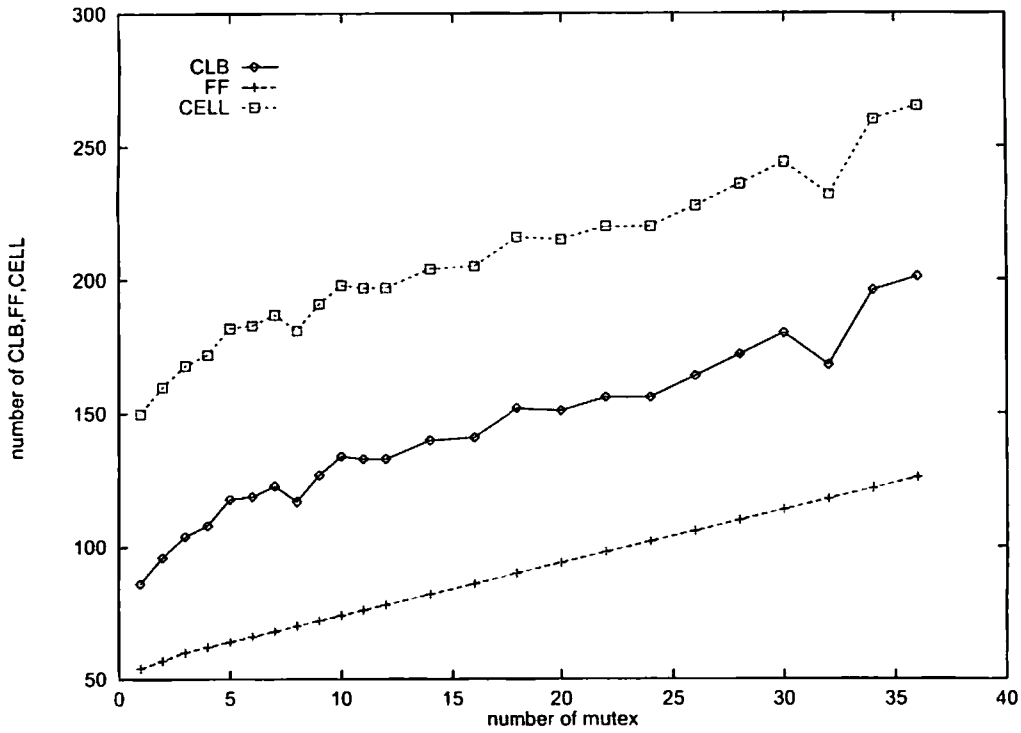

図 3.4: mutex の個数に対する回路規模の変化

ウェア化した機能を持っている．比較はプロセッサ部とプログラマブルロジック部が同一クロックであるという条件下で計測し，結果は実行時間をクロック数で示した．また，pthread_yield() 関数は，処理時間の異なる Start 処理（スレッドの初期割当て：表 3.6 における SAVE+START）と Cont. 処理（スレッドの切替え：表 3.6 における SAVE+RESTORE）の二つの場合について，pthread_mutec_lock() 関数は，ロックの成功，ロックの最初の失敗，2 回目以降の失敗の 3 種類について比較している．その結果，ハードウェア化した部分の比率によって性能改善率は異なるが，図 3.6 および図 3.7 に示したように約 17%～60%の性能改善が見られた．

次にプログラマブル・ロジック部とプロセッサ部とで動作クロックが異なる場合に性能がどう変化するかを示す．一般にプログラマブルロジックの動作速度はプロセッサのそれよりも数倍遅い．しかし，今回のマルチスレッド制御ライブラリは，基本的にプロセッサと並列して実行するため速度差の影響は少ない．唯一，pthread_yield() 関数だけは，プロセッサを止めて実行する期間（表 3.6 における START, SAVE, RESTORE 処理）を持っているため，プロセッサとの動作速度の差が全体性能に影響する．したがって，ここでは pthread_yield() 関数の性能推移について示す．図 3.8 は，プロセッサ部とプログラマブ

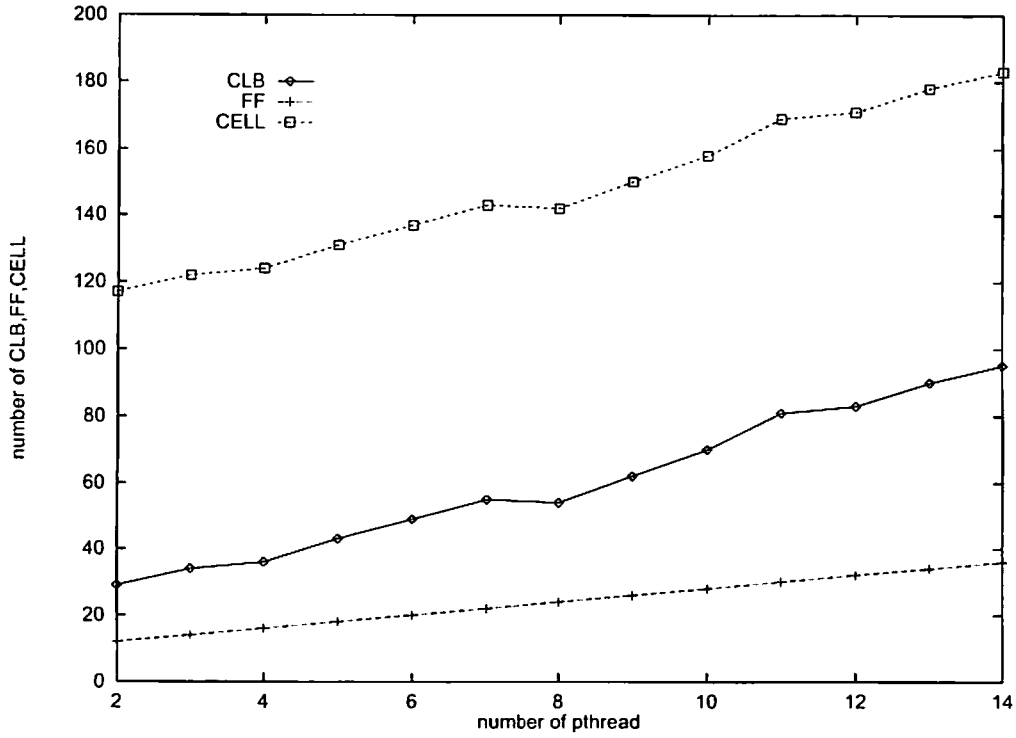

図 3.5: pthread 数に対する scheduler の回路規模変化

ル・ロジック部のクロック比を変化させた時の pthread_yield() 関数の Start 処理と Cont. 処理の実行クロックを，ソフトウェア版と本論文のハードウェア化したものとの比較した結果である．横軸はプログラマブル・ロジック部のクロックをプロセッサの倍数で示し，縦軸はプロセッサのクロック数を示した．図から判るように，Start 処理，Cont. 処理共にプロセッサのクロックが FPGA のクロックの約 11 倍の時にソフトウェア版とハードウェア化した処理がほぼ同じになる．したがって，スレッド制御機構のハードウェア化は，プロセッサとのクロック比が約 10 倍程度までは性能的な優位を保っていることがわかる．

3.5 考察

3.5.1 考察

本章では，リコンフィギャラブル・コプロセッサ方式としてプロセッサ，プログラマブル・ロジック混載方式について議論し，オンチップ・マルチプロセッサの同期制御やコヒーレンス制御にリコンフィギャラブル・コンピューティングを適用した．ここでは性

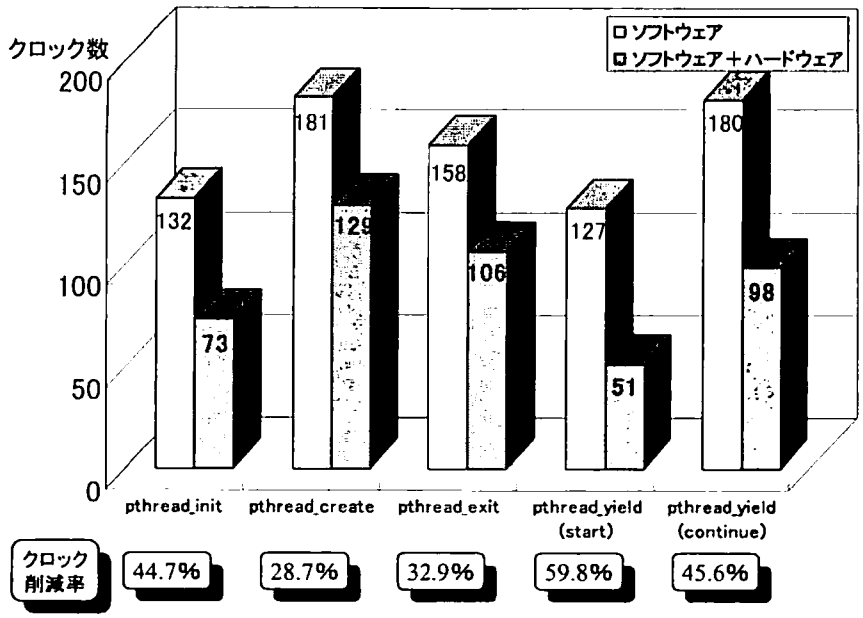

図 3.6: 性能比較（スレッド制御）

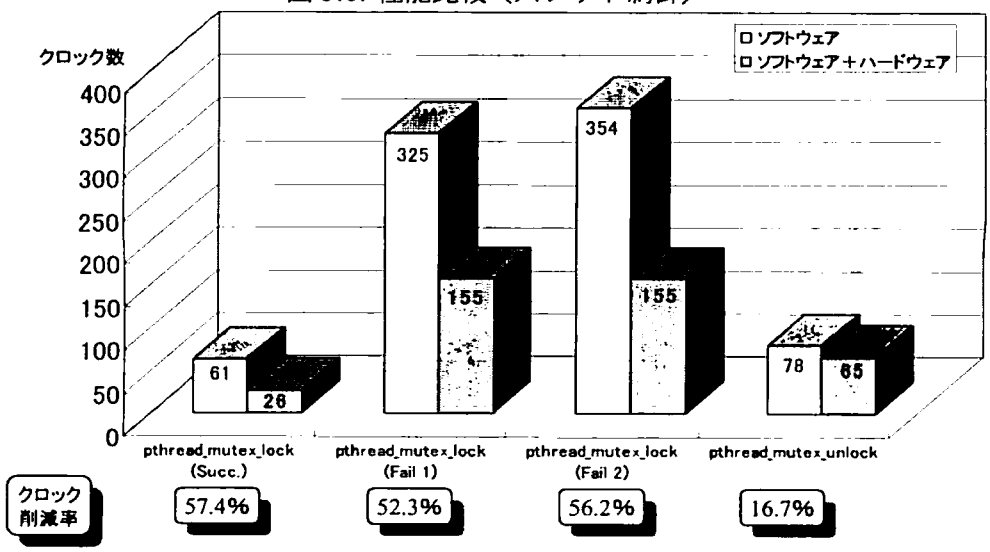

図 3.7: 性能比較（Mutex 制御）

能，回路規模および構成に関する考察を行う．

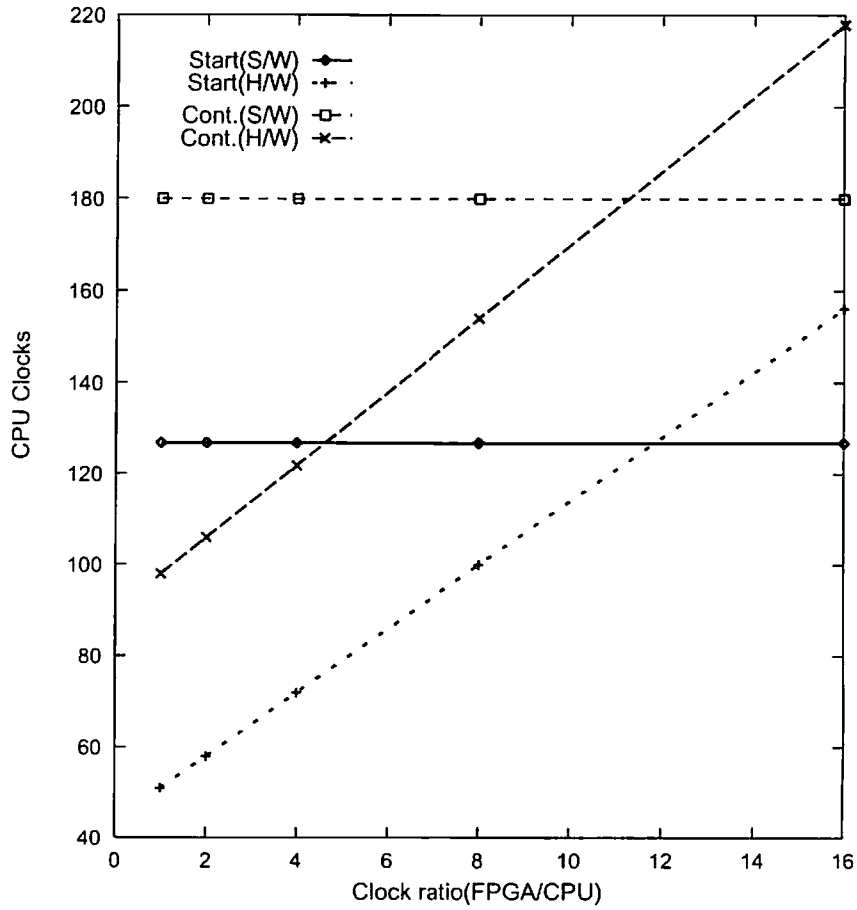

図 3.8: クロック比率の変化による pthread_yield() 関数の性能比較

性能に関する考察

本研究ではスレッド制御関数をハードウェアとソフトウェアの両方を用いて実装した．その結果，プロセッサ部とプログラマブル・ロジック部が同一クロック周波数で動作する場合，純粋にソフトウェアだけで実装した場合と比較して，約 17%～60%の性能改善が確認できた．また，プロセッサ部とプログラマブル・ロジック部が異なるクロック周波数である場合，そのクロック比が 10 倍以内であれば，性能が向上することを示した．この結果は，リコンフィギャラブル・コプロセッサ方式の目的である“プロセッサの稼動効率を高め性能を引き出すこと”が成功した事例といえる．

回路規模に関する考察

3.4 節では，本研究で作成したスレッド制御機構の各モジュール毎の回路規模について述べた．これを基に以下の条件のオンチップ・マルチプロセッサを実現する場合のプログラマブル・ロジック部の容量を試算すると，その回路規模は約 3,500CLBs になる．

- プロセッサ数は 4
- スケジューリングアルゴリズムはラウンドロビン
- スレッド数は 10 程度
- mutex 数も 10 程度

Xilinx 社の FPGA である XC4025（約 25K ゲート相当）の CLB 数が 1,024 個であることから，これはその約 3.5 倍必要であり，ゲート換算すると約 90K ゲートとなる．現在，利用可能な FPGA のゲート数が既に約 1,000 万システムゲート程度 [26] であり，プロセッサとの混載を考慮しても十分に実現可能な範囲といえる．上記の条件は，比較的小規模のプログラムを実行するものであるから，実用的な大規模のプログラムの実行にはさらに多くの領域が必要である．また，本研究で実装したスレッド制御関数は最低限のものに止めているため，現在のスレッド制御機構の未実装機能を追加した場合，必要となる領域はさらに拡大する．

構成に関する考察

本方式では既存のプロセッサを無変更で使用することを目指したが，プログラマブル・ロジック部からプロセッサを制御する必要から，今回は専用信号線を数本追加した．具体的には，パイプライン制御用信号 3 本，内部レジスタ制御用信号 1 本，アドレス信号 5 本の計 9 本（ビット）である．しかし，これはプロセッサ外部から内部リソースへのアクセスを可能にするインタフェースを追加するために必要な変更であり，プロセッサの構成を大きく変更してしまうものではないため，コア・プロセッサを新規に設計する場合には負担にならない．また，専用信号線を追加せず既存のプロセッサ・コアを使用する場合は，スレッド切替制御をハードウェアからソフトウェアに変更することで対応が可能である．この場合，VSI に準拠したインタフェースでプロセッサ・コアを利用することで，さらに

実用的なシステムが構築できると考えている.

3.5.2 今後の課題

本研究ではスレッド制御関数単体の性能比較しか行えなかったが，より実用的なアプリケーションによる評価が必要である．また，プログラマブルロジック部の代りにスレッド制御専用プロセッサを搭載した場合との性能比較なども必要であろう.

一方，3.4.2 節の図 3.4 および図 3.5 で示したように，今回設計したスレッド制御機構は，アプリケーションによって回路規模が大きく変わる要素を持っている．これでは搭載されているプログラマブル・ロジック部の容量から，プログラムの作成に制限を加えることになり，コアとして既存のプロセッサを用いても，その制限からプログラムの変更する必要が出てくる．したがって，使用されている mutex 数やスレッド数に影響されずに，回路規模を一定にする構成を検討する必要がある．また，本研究の回路すべてをプログラマブル・ロジック部に実装するのは，チップ面積の無駄になりかねない．それゆえに，プログラマブル・ロジック部と専用回路部の切り口に対するコデザイン的な評価も必要である.

また，図 3.1 に示したオンチップ・マルチプロセッサ上のカスタムハードウェア領域（プログラマブル・ロジック領域）は，本研究で扱ったスレッド制御関数専用の領域ではない．この領域にはアプリケーションに特化した演算器や外部との入出力処理をサポートする回路等の搭載も可能である．さらに，リコンフィギャラブル・コンピューティングの特徴である再構成性を発揮すれば，それらの機能を必要に応じて実現できより柔軟なオンチップ・マルチプロセッサが構築可能と考えられる．したがって，スレッド制御以外の応用についての評価も今後の課題である.

第 4 章

アタッチド・プロセッサ方式

本章では，リコンフィギャラブル・コンピューティングの一方式であるアタッチド・プロセッサ方式の研究結果について報告する．

4.1 概要

SPLASH[5] に代表されるアタッチド・プロセッサ方式のリコンフィギャラブル・システムは，汎用計算機のベクトルファシリティのように，計算集約的な処理やホスト計算機の不得手な処理を実行する．この方式は，ホスト計算機に対して独立性が高く，いわゆるサーバ・クライアントモデルで運用する場合が多い．実際，SPLASH は DNA パターンマッチング処理をスーパーミニコン（VAX11/785）の 2,700 倍，スーパーコンピュータ（CRAY-2）の 325 倍の性能を達成している．しかしながら，リコンフィギャラブル・システムとして見ると，SPLASH はパターンマッチング専用マシンに近く，類似問題も処理できるとは言え，決して汎用的でもスケーラブルでもない．

そこで本研究では，より汎用性とスケーラビリティを持つアタッチド・プロセッサ方式のリコンフィギャラブル・システム RASH（Reconfigurable Architecture based on Scalable Hardware）を開発した [27]．

RASH はアルゴリズムをハードウェアに直接実装し，並列かつ高速にデータ処理を行う．プロセッサに対する位置づけは，コプロセッサ的なものではなく，自ら処理の主体となる汎用の計算エンジンとして動作可能な装置である．

また，RASH のメモリモデルは，NORA（NO Remote memory Access model）/ NUMA（Non-Uniform Memory Access model）を基本としている．各処理要素（PE:Processing Element）に対してはメッセージによって交信する．したがって，各処理要素毎に付随し

ているメモリはローカルメモリとなり，他の処理要素からは直接参照できない．NORA の採用は同一筐体内の処理要素以外に外部ネットワークで接続された他の筐体内の処理要素ともシームレスな連携を可能にするためである．また，NUMA は FPGA 内に制御論理を追加することによって実現している．RASH のネットワークモデルは，実装方式には多段バスモデルを基本としているが，各 PE 間の通信は，Store & Forward 方式のパケット転送を採用している．

RASH の特徴を以下に示す．

- ハードウェアによる処理の高速化
- 高い並列処理能力
- FPGA による処理の柔軟性確保
- 大容量メモリのサポート可（ドータボードによる）

本章では，まず RASH のアーキテクチャおよび実機の構成について述べる．そして，アプリケーションとして暗号解析処理 [28, 29] および SAR （Synthetic Aperture Rader）の画像再生処理 [30] を用いて性能評価を行う．

4.2 システム構成

RASH は，複数演算ボードを搭載した CompactPCI (Peripheral Component Interconnect) ユニットをネットワークで接続することでスケーラビリティを確保している．本節では RASH を構成する各要素の構成について述べる．

4.2.1 ユニット構成

RASH では，1 つの CompactPCI ユニットからなる構成を基本構成（1 ユニット）としている．図 4.1 に示したように，基本構成では CompactPCI バス上に最大 6 枚の演算ボード（EXE（EXEcution）ボード）とそれらを制御するための 1 枚の汎用プロセッサボード（CPU ボード）が接続される．また，基本構成には，CPU ボード経由で接続される磁気ディスクやネットワークインタフェースも含まれている．図 4.2 に RASH の実機を示す．

ネットワークはイーサネットとし，これを介して FEP（Front-End Processor）として PC（Personal Computer）や WS（Work Station）などが接続される．また，複数ユニット間もネットワーク接続される．これにより，多数のユニットを接続してより大きなシステムを

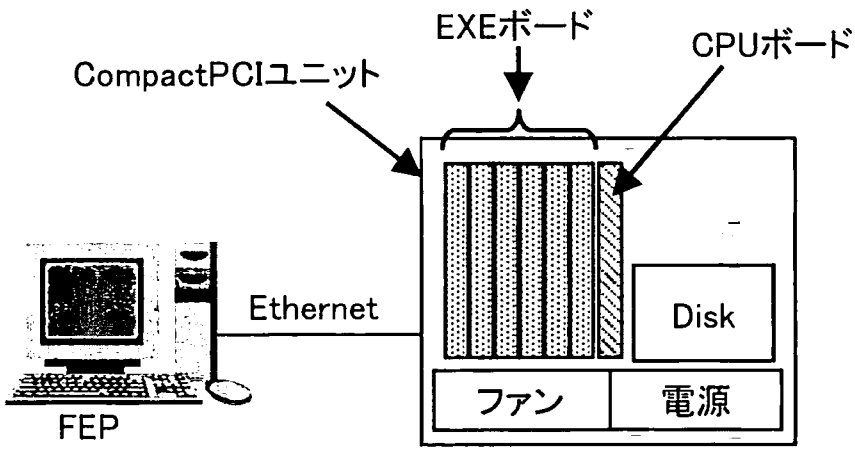

図 4.1: RASH のユニット構成

構成することが可能である．

4.2.2 EXE ボードの構成

RASH の基本構成要素は，CompactPCI 基板上に 1 チップ 10 万ゲート規模相当の SRAM タイプの FPGA を 8 個搭載した EXE ボードである．FPGA には ALTERA 社の FLEX10K100A-1（240 ピン QFP）を使用した．図 4.3 に EXE ボードの構成を示し，図 4.4 に EXE ボードの実ボードを示す．EXE ボードには PCI バスインタフェース回路と 2MB の SRAM からなるローカルメモリが搭載されコントローラに接続されている．また，コントローラは各 FPGA とバス接続（32bit）されている．FPGA の回路情報はローカルメモリを経由してロードされる．ローカルメモリ上に複数種類の回路情報を保持することができ，1 つの FPGA 当り 190ms 程度で再構成が可能である．また，同一回路なら全数の 8 個までの任意数の FPGA を同時に再構成が可能である．

FPGA 間はバスとは別に 32bit の信号線でメッシュ接続されている．これにより，実現したい機能を 2 チップ以上を使って搭載するような場合や，機能ブロック間のデータをパイプライン的に流すような処理も可能となる．後者のような用途を考慮し，各 FPGA には個別に周波数を設定可能なローカルクロックの他に，全 FPGA で共通のグローバルクロックが供給される．グローバルクロックおよびローカルクロックは表 4.1 のように約 4.9MHz から 60MHz の 16 種類から選択可能である．また，回路の再構成やクロックの

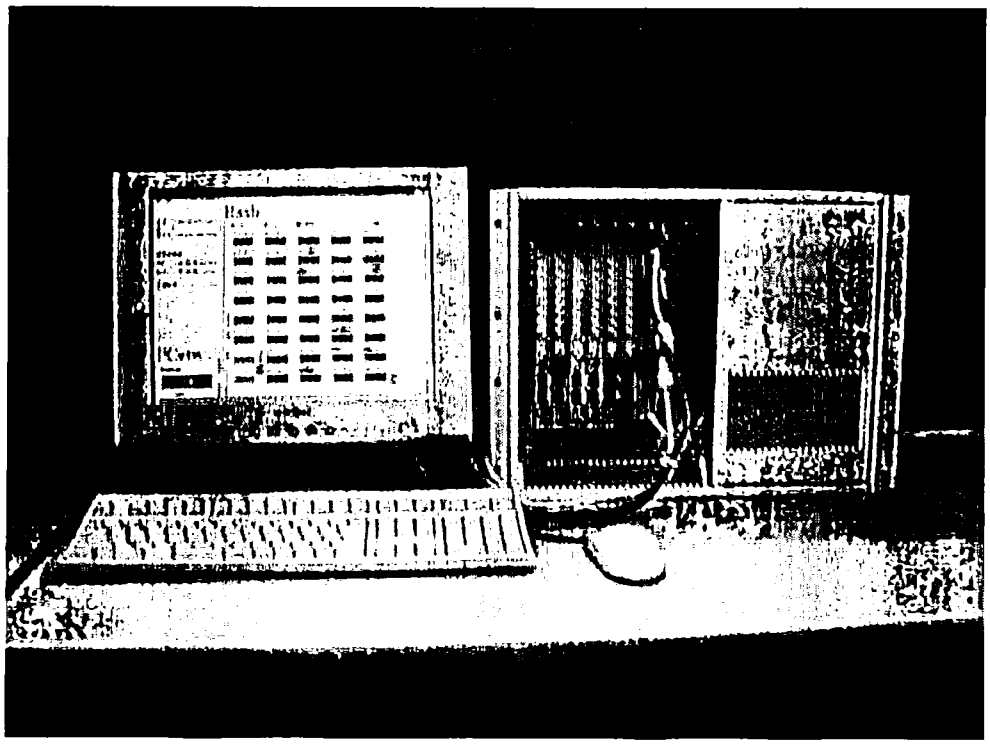

図 4.2: RASH システム（1 ユニット構成）の概観

表 4.1: 使用可能なクロック周波数

4.92	24.58	36.00	48.22
9.68	28.64	39.50	50.00
14.32	30.00	42.00	55.00
19.35	33.15	45.00	60.00

単位：MHz

設定などは，すべて FEP 上のソフトウェアから制御できる．

4.2.3 拡張ボード

各 FPGA からは直接 40bit ずつの信号線が拡張ボードコネクタに接続されている．FPGA での実現が容量および速度の面で非効率な場合や，PCI バス経由では入出力のスループットが不足する場合には，拡張ボードをドータボードとして搭載させる．例えばメモリや I/O デバイスコントローラ等をドータボード上に実現すれば良い．このような実装形態を取ることにより，EXE ボード上でのアーキテクチャ上の制約の最小化と用途別の性

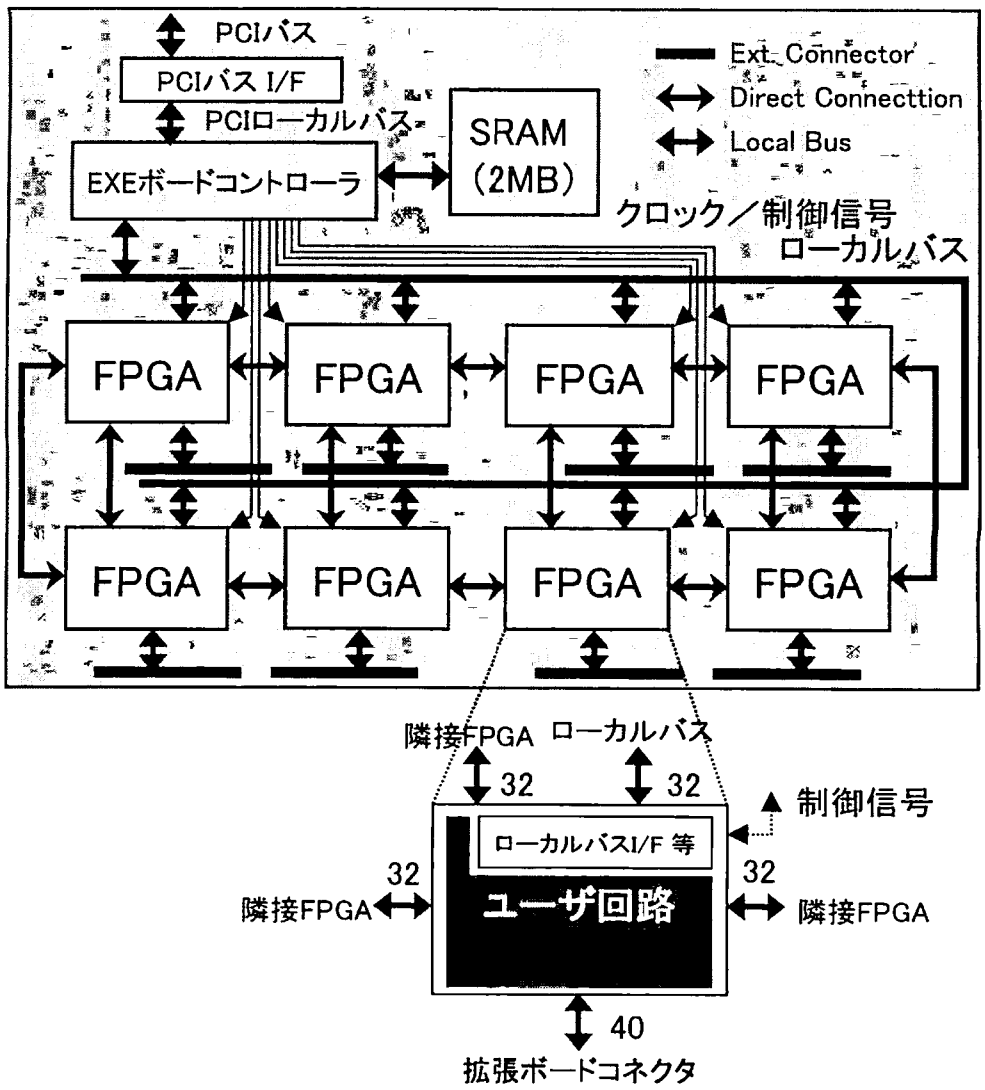

図 4.3: EXE ボードの構成

能最大化の両立を図れる．また，EXE ボードに RASH 用メモリ搭載ドータボード（RDM : Rash Daughter Memory board）を搭載する場合には，CompactPCI のスロットの制約から EXE ボード 3 枚，RDM6 枚（各 EXE ボードに RDM2 枚搭載）が最大の構成になる．

図 4.5，図 4.6 に RDM の構成と外観を示す．RDM には，APEX20K200 を使用した SDRAM コントローラが 2 個搭載されており，それぞれに 128MByte の SDRAM モジュール（S.O.DIMM）が接続されている．SDRAM コントローラには EXE ボード上の 2 つの FPGA から 40bit の信号が拡張コネクタを通して接続され，SDRAM モジュールをそれぞれ 2 つずつの FPGA で共有する構成になっている．RDM 1 枚で 4 個の FPGA に対応す

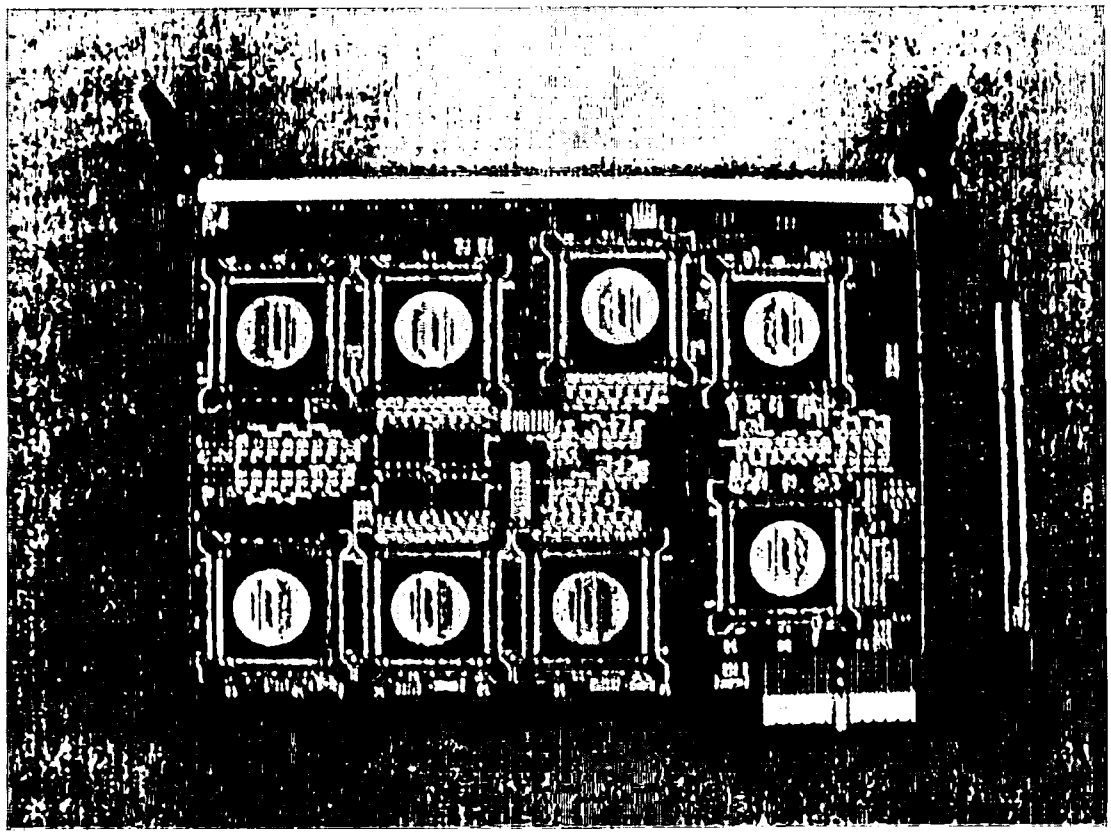

図 4.4: EXE ボードの概観

るため，1 枚の EXE ボードには 2 枚まで RDM を搭載できる．

また，各々の SDRAM コントローラには USB コントローラが接続されており，ドータボード上の USB コネクタを介して外部とのデータ交換ができる構成になっている．

4.3 暗号解析処理への適用

RASH の一応用分野として暗号解析（暗号鍵探索）が挙げられる．FPGA に暗号回路を実装することにより，汎用のマイクロプロセッサよりも高速な処理が行える．また，FPGA は専用 LSI に比べて処理性能は劣るが，解析する暗号を容易に変更できる．さらに，鍵探索処理は完全な並列性を持つため，FPGA の並列化により容易に高速化できる．このため，本研究では DES（Data Encryption Standard）の鍵探索処理を RASH で実装し，性能評価を行う．

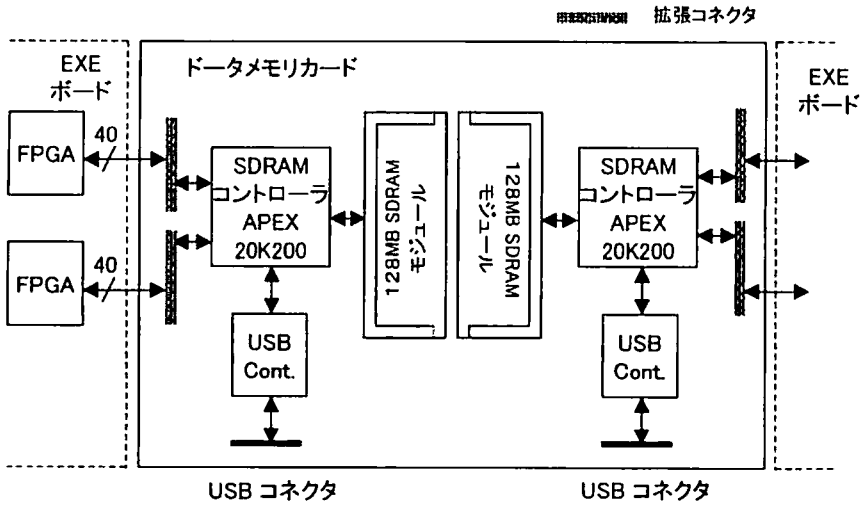

図 4.5: メモリドータボードの構成

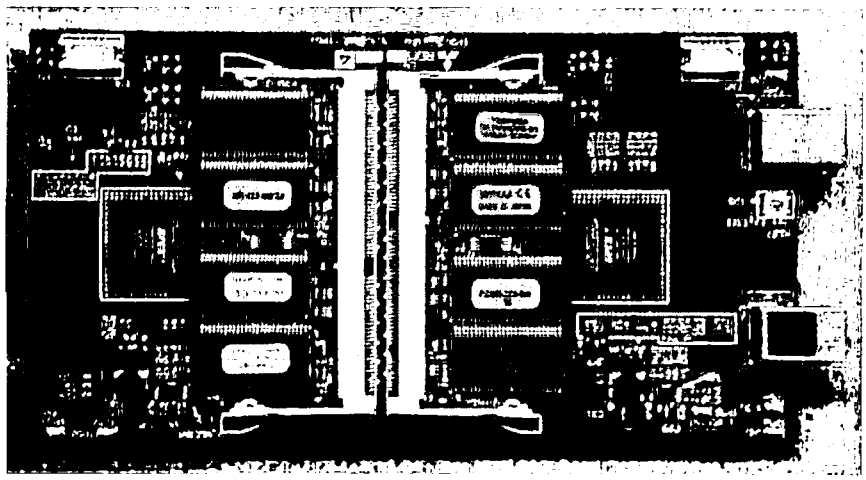

図 4.6: メモリドータボードの概観

4.3.1 DES 鍵探索処理アルゴリズム

DES は 56bit 鍵の秘密鍵暗号であり，米国を中心として広く使われている．以下に DES のアルゴリズムについて説明する．

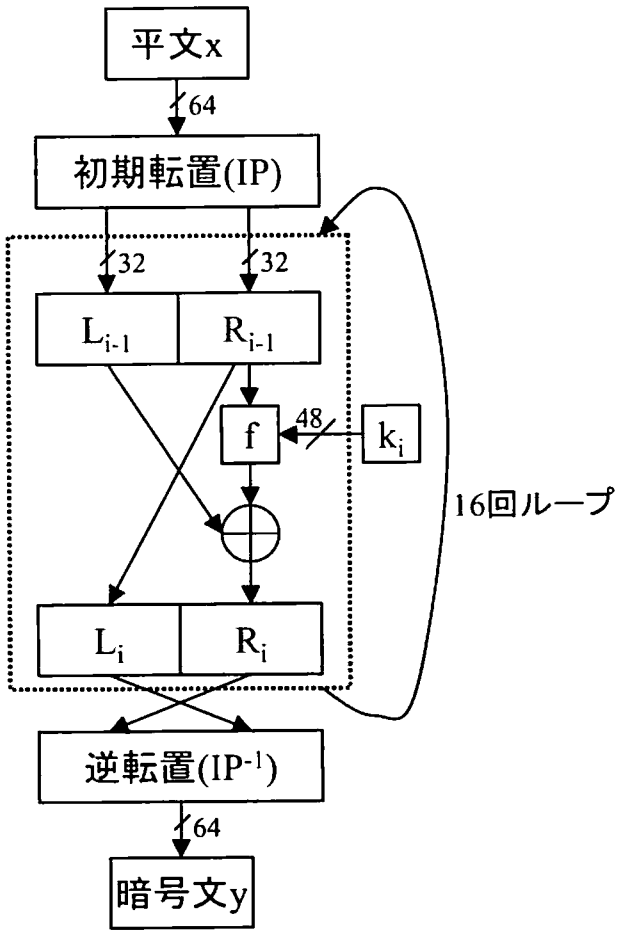

図 4.7: DES の基本アルゴリズム

暗号化アルゴリズム

DES は，長さ 64bit の平文ビット列 x を長さ 56bit の鍵ビット列 K で暗号化し，長さ 64bit の暗号文ビット列 y を出力する．図 4.7 に DES 暗号の基本アルゴリズムを示す．

(1) 与えられた平文を初期転置 IP（Initial Permutation）により変換し，64bit のビット列 x_0 を得る．ここで，x_0 の上位 32bit を L_0，下位 32bit を R_0 とする．

(2) 次の演算を 1 段として，これを 16 回繰り返し，L_i と R_i を計算する．

$$L_i = R_{i-1} \tag{4.1}$$

$$R_i = L_{i-1} \oplus f(R_{i-1}, k_i) \tag{4.2}$$

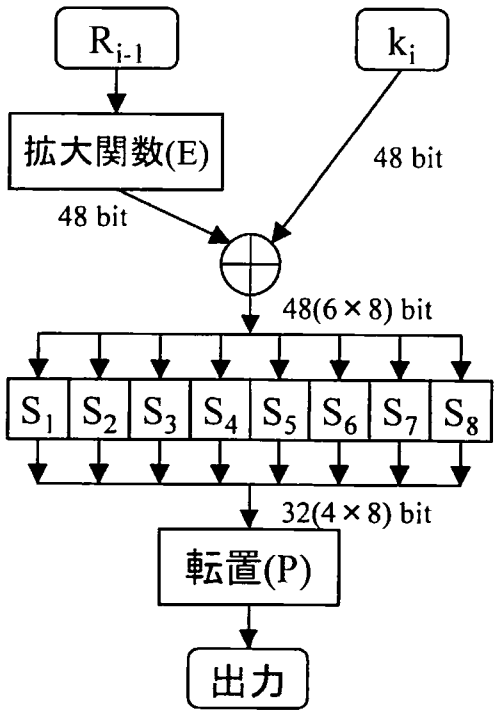

図 4.8: f 関数

（$\oplus$ は排他的論理和，$1 \leq i \leq 16$）

k_1～k_{16} はそれぞれ長さが 48bit の副鍵で，鍵スケジュールにより鍵 K から導き出される．鍵スケジュールと関数 f については後述する．

(3) R_{16} を上位 32bit，L_{16} を下位 32bit とするビット列に逆転置 IP^{-1} を行い，暗号文 y を得る．

関数 f は 32bit のビット列 R_{i-1} と 48bit のビット列 k_i を入力とし，32bit のビット列を出力する．図 4.8 のこの関数の演算の構成を示す．

(1) 32bit の入力データ R_{i-1} を決められた拡大関数 E（Expansion function）により 48bit に拡大する．

(2) $E(R_{i-1}) \oplus k_i$ を計算し，結果を 6bit 単位の 8 個のビット列 B_1～B_8 に分ける．

(3) 各ビット列 B_1～B_8 を 8 個の S-Box S_1～S_8 で処理し，4bit のビット列 C_1～C_8 を得る．S-Box は 6bit の入力に対して決められた表をもとに 4bit を出力する関数である．

(4) C_1～C_8 を 1 つの 32bit のビット列として決められた転置 P により並び替える．これにより，得られたビット列が関数 f の出力になる．

鍵スケジューリング

鍵スケジュールでは，56bit の鍵 K から 48bit の副鍵を 16 個生成し（k_1〜k_{16}），上述の暗号化のアルゴリズムでの各段に供給する．DES の鍵スケジュールアルゴリズムについて以下で示す．なお，以下で $c_i d_i$ $(0 \leqq i \leqq 16)$ は 56bit のビット列であり，c_i はそのビット列の上位 28bit を表し，d_i はそのビット列の下位 28bit を表すものとする．

(1) 56bit の鍵 K を置換し 56bit のビット列 $c_0 d_0$ を得る．$PC-1$ はあらかじめ定められた 56bit から 56bit へのビットの置換である．

$$c_0 d_0 = PC-1(K) \tag{4.3}$$

(2) 次の演算を 1 段として，これを 16 回繰り返し，c_i と d_i を計算する（$1 \leq i \leq 16$）．ls_i は i の値によってあらかじめ定められた 1bit もしくは 2bit の左への巡回シフトである．

$$c_i = ls_i(c_{i-1}) \tag{4.4}$$
$$d_i = ls_i(d_{i-1}) \tag{4.5}$$

(3) 56bit のビット列 $c_i d_i$ を置換し 48bit の副鍵 k_i を生成する．$PC-2$ はあらかじめ定められた 56bit から 48bit へのビットの選択的置換である．

$$k_i = PC-2(c_i d_i) \tag{4.6}$$

4.3.2 従来の DES 暗号の実装方法

RASH における FPGA（FLEX10K100A-1）への DES 暗号回路の従来の実装 [31] について図 4.9 に示す．

従来の回路は，3 個の並列動作可能な DES コア，DES コアの制御回路およびバスインタフェース回路からなる．DES コアは 56bit の候補鍵を生成する鍵生成回路（バイナリカウンタ）と，1 個の f 関数を含む回路（図 4.7 の破線部分，以降 f 関数 1 段回路と称する），56bit の鍵から 48bit の副鍵を作成する鍵スケジュール回路で構成されている．DES コアでは f 関数 1 段回路と，鍵スケジュール回路で演算を 16 回繰り返すことにより 1 回の暗号結果を得る．

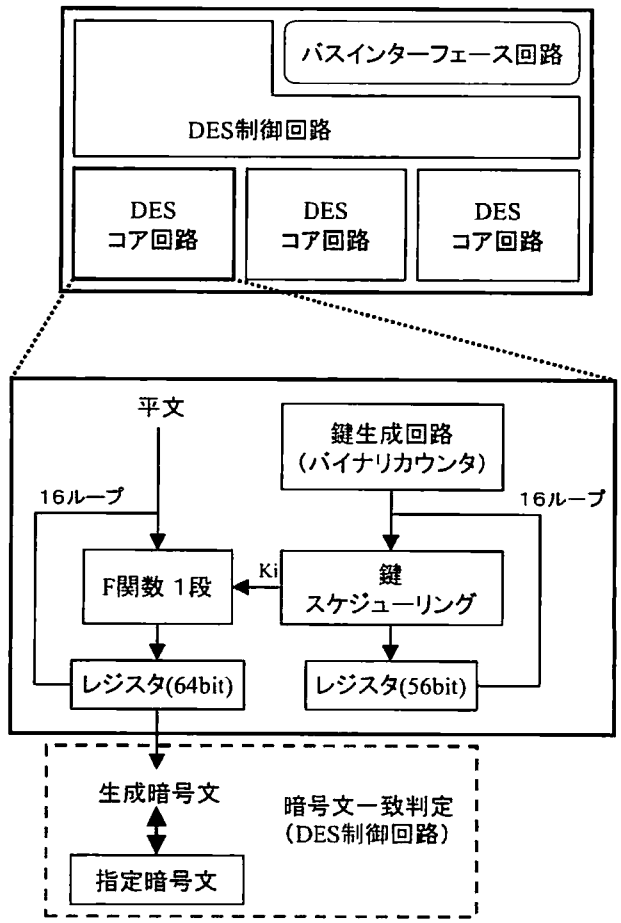

図 4.9: 従来の DES 暗号の実装方式

RASH での実行時には，CPU ボードから，各 FPGA 上の上述の回路に鍵の探索範囲，即ち初期鍵と探索数が与えられる．また，鍵探索を行う一組の指定平文と指定暗号文も与えられる．鍵生成回路は，初期鍵を順次カウントアップして鍵を作成し，DES コアで暗号化を行う．生成された暗号文は指定した暗号文と一致判定を行い，一致する場合は正しい鍵が発見されたと判断する．鍵が発見されず鍵生成回路が探索範囲の鍵生成を終えた場合は，鍵探索終了報告を CPU ボードに行い，CPU ボードから新たな探索範囲が与えられ正しい鍵が発見されるまで処理が継続される．従来の実装方式では，各 DES コアで独立して鍵探索を行えるようにするために，DES コアそれぞれに鍵生成用回路を設けている．

上述の回路を Verilog-HDL で記述し MAX+plusII で合成を行い，RASH 上で実際の性能を測定した．この時の各回路の LE（Logic Element）の使用数と使用率を MAX+plusII 上のフロアプランから見積もったのが表 4.2 である．表 4.3 にこの時の暗号化性能を示

表 4.2: 従来の DES 暗号の実装結果

回路	使用 LE 数	LE 使用率
バスインターフェース回路	89	1.8%
DES 制御回路	1,357	27.2%
f 関数 1 段回路 3 個	2,814	56.4%
(f 関数 1 段回路 1 個)	(938)	(18.8%)
全回路	4,227	84%

表 4.3: 従来の DES 暗号の性能

項目	内容
DES コア回路	f 関数 1 段の 16 ループ
f 関数の個数	3 個／ FPGA
動作周波数	39.5MHz
鍵探索性能	7.41M 鍵／秒／ FPGA
LE 使用率	84%

す．表 4.2 から f 関数 1 段回路 1 個の LE 使用率が 20%程度，それ以外のバスインターフェース回路＋ DES 制御回路の LE 使用率が 30%程度である．そのため，1 個の FPGA に搭載する DES コア回路は 3 個となった．

4.3.3 回路構成の改良

DES 暗号の FLEX10K100 デバイスへの実装に関しては， AHDL の記述により f 関数 16 段のパイプラインを構成し（LE 使用率 86%），周波数 25MHz で動作させたという報告が Hamer らによりなされている [32].

彼らは主に，S-Box と鍵生成回路を改良することにより回路規模を縮小し，FPGA 上に 1 組の f 関数 16 段パイプラインを構成した．また，従来の DES 回路は 39.5MHz で動作している．このため，本研究では 16 段パイプラインよりも少ない段数でパイプラインを構成しデバイスに余裕を持たせることにより動作クロックを向上できるのではないかと考えた．そのため，彼らの改良のアイデアを採り入れ，更に f 関数 N (<16) 段パイプラインの構成を検討し，性能評価を行うことにした．従来実装と同様に Verilog-HDL で回路を記述し MAX+plusII で合成を行った．

本研究での改良の主なものを次に挙げる．これらについて以下で説明する．なお，(1) と (3) は Hamer らの行った改良点である．

(1) **S-Box の最適化**

S-Box の構成をデバイス（FLEX10K100）に適したものにする．

(2) **パイプラインの最適化**

f 関数 N（<16）段パイプラインを検討し，デバイスに適した段数構成にする．

(3) **鍵生成回路の単純化**

鍵生成回路を単純化する．

(4) **副鍵の供給の最適化**

副鍵の供給方法をパイプラインの段数に合わせて最適化する．

S-Box の最適化

S-Box は f 関数 1 段回路の回路構成のほとんどを占めている．このため，S-Box をデバイス（FLEX10K100）に適した構成にして，f 関数 1 段回路の使用 LE 数を縮小できる．

S-Box は 4.3.1 節でも述べたように 8 個の 6 入力 4 出力の Look-Up Table（LUT）で構成されている．これは 32 個の 6 入力 1 出力の LUT（6-LUT）とみなせる．これに対し，FLEX10K100 では，1 つの LE に 4 入力 1 出力の LUT（4-LUT）が 1 つある．

このため，図 4.10 のように 7 個の 4-LUT を使い，6-LUT を構成するように，Verilog-HDL の記述を明示的に変更する．この 6-LUT では，6 入力のうちの 4 入力を 1 段目の各 LUT に入力する．また，2 段目と 3 段目の LUT で 4 入力 1 出力のセレクタを構成し，6 入力のうちの残りの 2 入力をセレクト信号として，1 段目の出力をセレクタで選択する．また，4.3.1 節にあるように S-Box での出力は L_i と排他的論理和をとる．このため，3 段目の LUT を使ってこの演算も行う．

これにより，1 つの S-Box を 224LE（$=7\times32$）で構成する．

ちなみに，Hamer らは AHDL 記述により最終段の LUT に代えて，AND カスケードチェインを使いセレクタを構成しているが，Verilog-HDL 記述ではこの指定は必ずしもうまく行かない．

パイプラインの最適化

f 関数 N（<16）段パイプラインを構成することを考える．$N=2^h$（h：自然数）ならば N 段パイプラインでの処理を $16/N$ 回繰り返すことにより DES の処理が行えるので構成

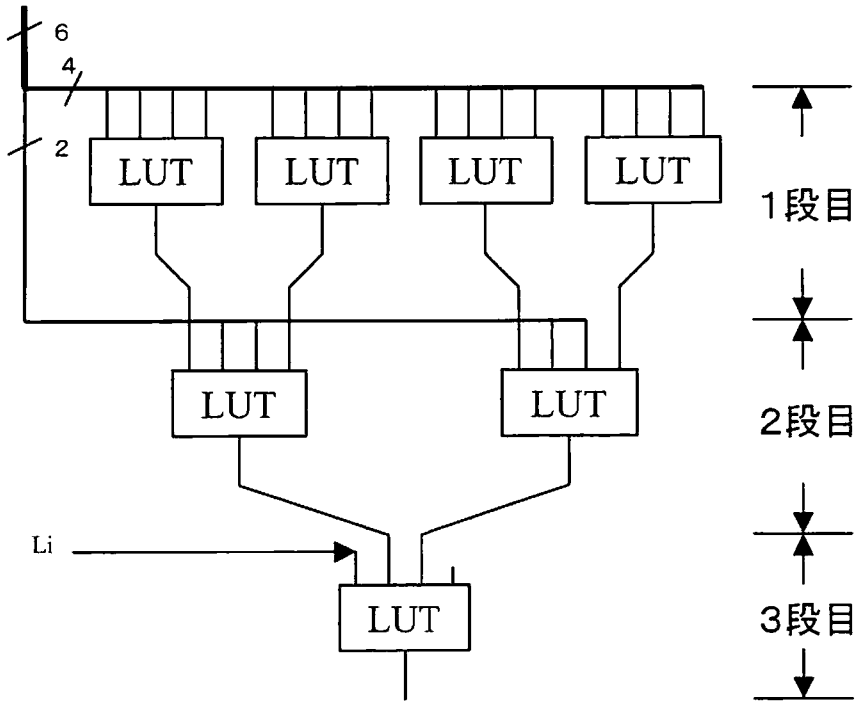

図 4.10: S-Box の最適化

は容易である．$N \neq 2^h$ の場合は図 4.11 のようにセレクタと n（$= 2^k$，$N = \alpha n$）段パイプラインを組み合わせることにより構成できる．

図 4.11 には一例として 12 段パイプラインを示した．12 段パイプラインは 4 段パイプラインを 3 個直列に接続することにより構成できる．この場合，処理は 16 クロックで 1 サイクルであり，タイミング 1〜12 で平文と鍵が入力され，次のサイクルのタイミング 1〜12 で暗号文が出力される．また，タイミング 13〜16 で各 4 段パイプラインでの出力をセレクタを使い入力に戻す．これにより，タイミング 1〜4 で入力されたデータは 4 段パイプライン C での処理を 2 回，5〜8 で入力されたデータは 4 段パイプライン B を 2 回，9〜12 で入力されたデータは 4 段パイプライン A を 2 回繰り返す．これにより，各データに対して f 関数 16 回の処理がおこなわれる．

FLEX10K100 では LE 数の制約から構成できる 4 段パイプラインは 3 個程度である．これを図 4.12 のように単純に並列に構成した場合に比べて 12 段パイプラインを構成した方が，鍵の探索処理を行う場合の鍵の供給や暗号文の一致判定を行う回路が 1 つにできるため，回路の規模を縮小できる．また，以下の 4.3.3 節に示す副鍵の供給の最適化を適用する場合，4 段パイプラインでは各段に 4 入力 1 出力のセレクタが必要となり 3 並列のた

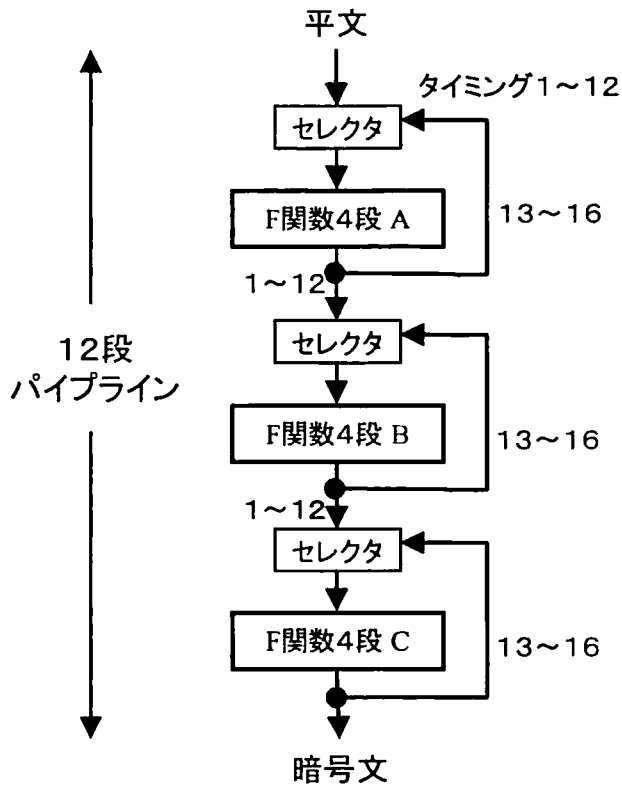

図 4.11: 12 段パイプラインの構成

め処理は複雑になる．これに対し，12 段パイプラインでは 2 入力 1 出力のセレクタで単純に処理を行うことができるため回路規模を縮小できる．

鍵生成回路の単純化

RASH では CPU ボードからのソフトウェア的な制御により，鍵 K の上位 32bit 程度を固定して各 FPGA に供給する．従って，上位 32bit 程度はレジスタで保持すれば良いが，下位 24bit 程度で候補鍵を順次生成しなくてはならない．従来実装ではこれをバイナリカウンタで行っていた．パイプラインを構成した場合，常に段数分の鍵を保持する必要があるため，更に多くのフリップフロップが必要になる．

このフリップフロップを削減するために，鍵生成にバイナリカウンタではなく，LFSR (Linear-Feedback Shift Register，線形フィードバックシフトレジスタ) を使用する．LFSR は図 4.13 のような構成であり，M 系列の乱数発生器として使用できる．L bit の LFSR は最大 $2^L - 1$ の周期をもち，単純な機能追加により 2^L の周期にすることができる．した

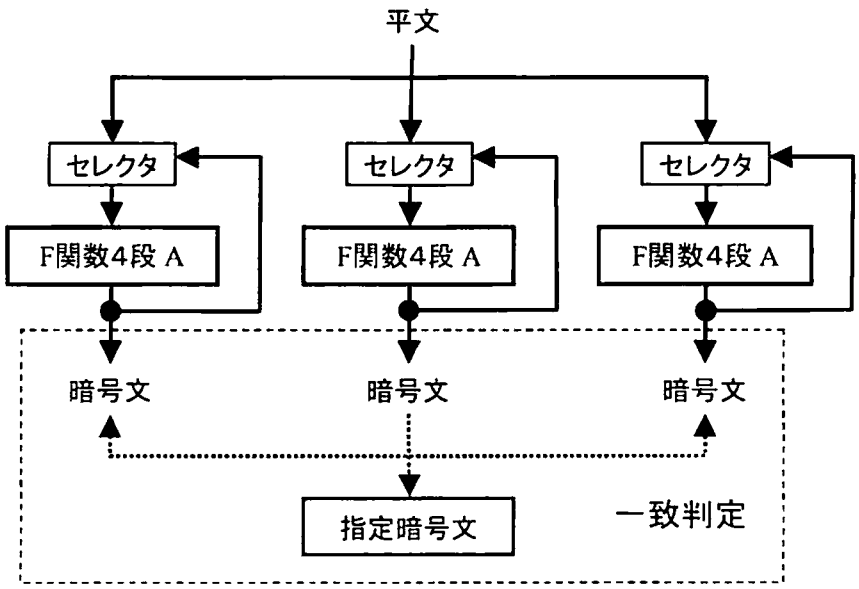

図 4.12: 4 段パイプライン 3 並列の構成

がって，探索範囲の大きさと同じ周期の LFSR をバイナリカウンタの代わりに使うことで，指定された探索範囲のすべての鍵を生成することができる．また，図 4.13 において LFSR の左端から L bit 目までを m 番目の鍵として使用した場合，1 つ前の $m-1$ 番目の鍵は 2bit 目から $(L+1)$bit 目までとなる．したがって，$N-1$ bit のシフトレジスタを連結することで，現在 LFSR で生成した鍵から $N-1$ 個前までの鍵，すなわち N 段パイプラインに必要なすべての鍵を保持することが可能である．これにより，パイプラインで処理する場合のフリップフロップ数を大幅に抑制できる．

副鍵の供給の最適化

従来の実装では，鍵スケジュールでの副鍵 k_i の生成では 4.3.1 節に示したビットシフトと選択的置換による演算を行っていた．この k_i を得るための演算は，ハードウェア的には鍵 K のビットを入れ替えるだけの処理である．そのため，f 関数の 16 段パイプラインを構成した場合，各段の f 関数に鍵 K からのビットの選択だけで全ての副鍵を供給することにより，LE の消費を抑制できる．さらに，4.3.3 節の LFSR を用いた鍵生成回路を組み合わせることにより副鍵の供給を単純に行うことができる．

なお，12 段パイプライン等の場合，副鍵の供給方法が多少複雑になる．これは，4.3.3 節で示したようにタイミング 13〜16 で各 4 段パイプラインでの出力が入力に戻されるた

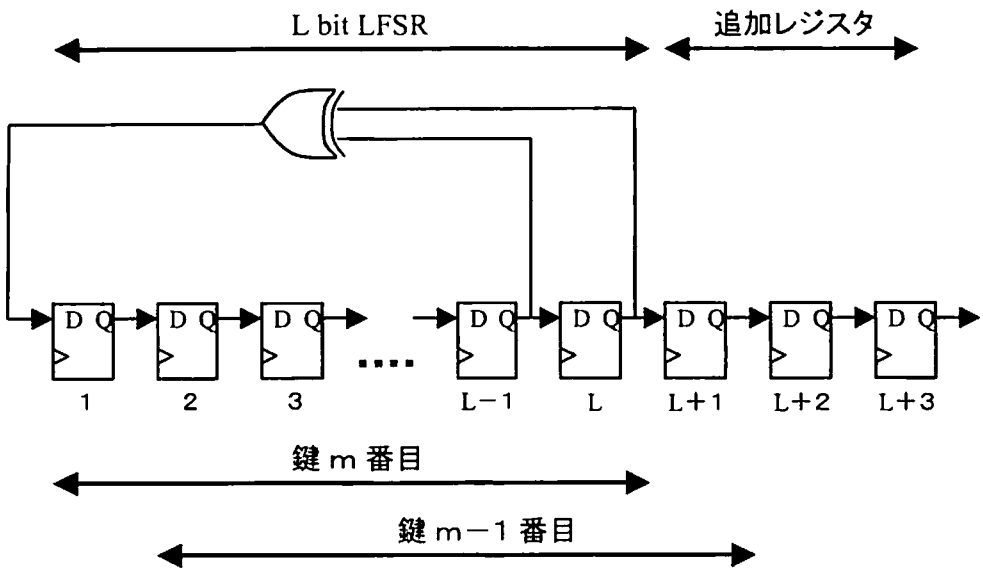

図 4.13: LFSR の構成

め，ある f 関数において入力に戻される前のデータと戻された後のデータでは供給する副鍵が異なるからである．このため，図 4.14 のように LFSR に保持された鍵から作られる副鍵を各 4 段パイプラインの出力を入力に戻す前と後で切り替えて各段の f 関数に供給する．セレクタには LE を使用するが，全体としては従来の実装方法に比べ少ない LE で鍵スケジュールが行えるようになる．

4.3.4 回路構成と性能評価

回路構成

4.3.3 節の S-Box の最適化により f 関数 1 段回路の回路規模を従来に比べて 450LE 程度削減できた．また，4.3.3〜4.3.3 節の改良手法を組み合わせることにより，f 関数 1 段回路の回路規模を 100LE 程度，DES 制御回路の回路規模を 500LE 程度削減できた．その結果，f 関数 1 段回路の回路規模は 350LE 程度に，DES 制御回路の回路規模は 800LE 程度になった．これをもとにして，DES 暗号回路のパイプライン化を行い，RASH 上での性能評価を行った．パイプラインは f 関数 1 段（従来方式），8 段，12 段の構成を作成し比較した．表 4.4 に各構成での性能を，表 4.5 に LE の使用数と使用率を示す．また，参考のため 1 チップに収まりきらなかった 16 段パイプラインの MAX+plusII での評価結果も合わせて表 4.5 に示す．

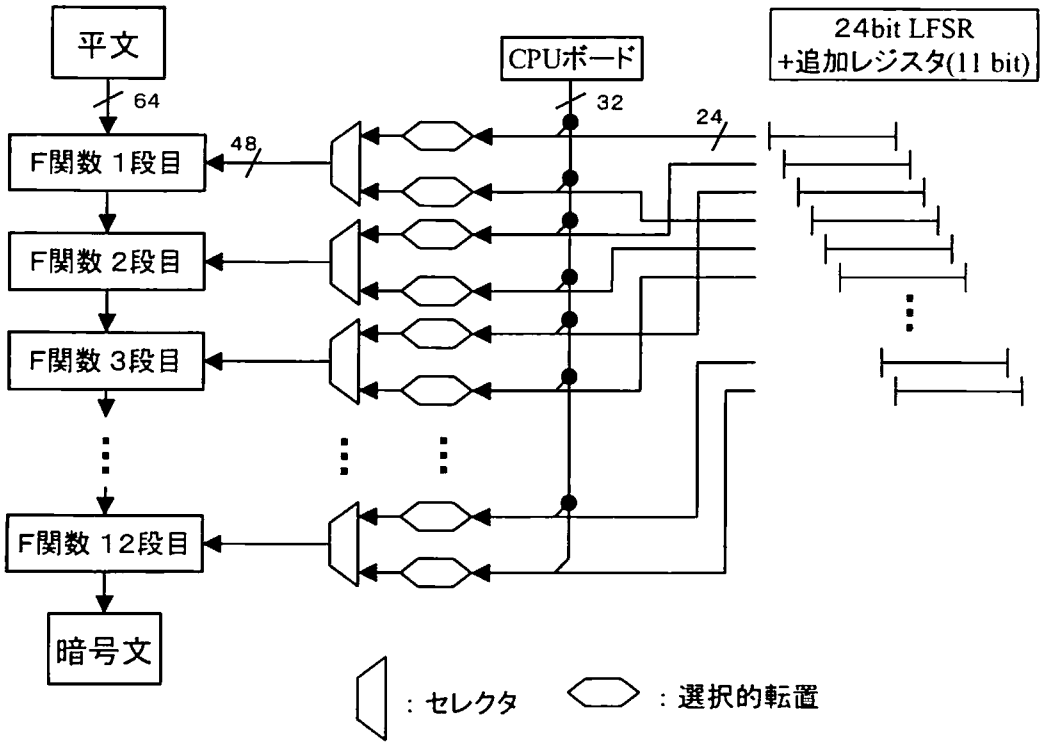

図 4.14: 副鍵の供給

表 4.4: 各構成での性能

パイプ段数	コア回路数	動作周波数	鍵探索性能（FPGA 当り）
1 段	5	42MHz	13.1M 鍵／秒
8 段	1	48.22MHz	24.1M 鍵／秒
12 段	1	39.5MHz	29.6M 鍵／秒

性能評価

上述の FPGA での単体性能と，汎用マイクロプロセッサ [33, 34] や DES 暗号専用 LSI[35] 上で DES 暗号化を行った場合との性能比較を表 4.6 に示す．本研究の改良により，FPGA の単体性能で専用 LSI の 2 分の 1 程度まで性能を向上させることができた．

また，本研究では $N \neq 2^h$ の場合のパイプライン構成を示した．例えば，12 段パイプラインを構成した場合，4.3.3 節に示す副鍵の供給手法と組み合わせることにより，4 段パイプラインを 3 個並列に構成する場合よりも回路規模をおさえることができる．このような

表 4.5: 各構成での LE の使用

パイプ段数	コア回路数	LE 使用数	LE 使用率
1 段	5	3,946	79%
8 段	1	3,943	78%
12 段	1	4.506	90%
16 段	1	5,447	109%

表 4.6: DES 暗号の性能比較

対象	動作周波数	ゲート数	単体性能
FPGA（RASH:12 段，1 回路）	39.5MHz	100K	29.6MKey/s
Intel Pentium[33]	300MHz	—	0.83MKey/s
DEC α チップ [34]	300MHz	—	2.14MKey/s
FPGA（TM-2a:16 段，1 回路）[32]	25MHz	100K	25.0MKey/s
DES 暗号 LSI（16 段，2 回路）[35]	33MHz	150K	66.0MKey/s
FPGA（RASH 従来回路:1 段，3 回路）	39.5MHz	100K	7.41MKey/s

パイプライン構成は，回路規模と動作クロックのトレードオフをとりつつ適切な段数を選択できるため，機能拡張や FPGA の容量変更に対する柔軟性も高く，メリットは大きい．

なお，Hamer らは 16 段パイプラインを 1 チップに収めることができたが，これは Altera 社の FPGA 専用の HDL である AHDL で記述したためであろう．我々の Verilog-HDL による記述では 12 段パイプラインで彼らの 16 段パイプラインの LE 数を超えた．ひとつには 4 段パイプラインごとに挿入したセレクタ等に LE を消費したためであろうが，ファンアウト調整などにおいても論理合成の差が大きく出たのではないかと推察される．ちなみに，Hamer らの回路の動作周波数は本研究のそれに比べ 63%程度と低い．彼らの論理合成結果と詳細な比較をしない限り断定できないが，クリティカルパスの短縮と使用 LE 数の縮小とが FPGA においてもトレードオフの関係になった可能性がある．

4.4 SAR 画像再生処理への適用

他の応用として合成開口レーダ（SAR, Synthetic Aperture Radar）の画像再生処理を用いて評価した．SAR はマイクロ波を使用した映像レーダであり，レーダで受信したデータに SAR 画像再生と呼ばれる処理を行い画像を作成する．SAR 画像再生処理の大部分は FFT（Fast Fourier Transformation：高速フーリエ変換）による処理であり，画像サイズが大きくなると演算量が膨大になる．このため，従来は汎用大型計算機かワークステーショ

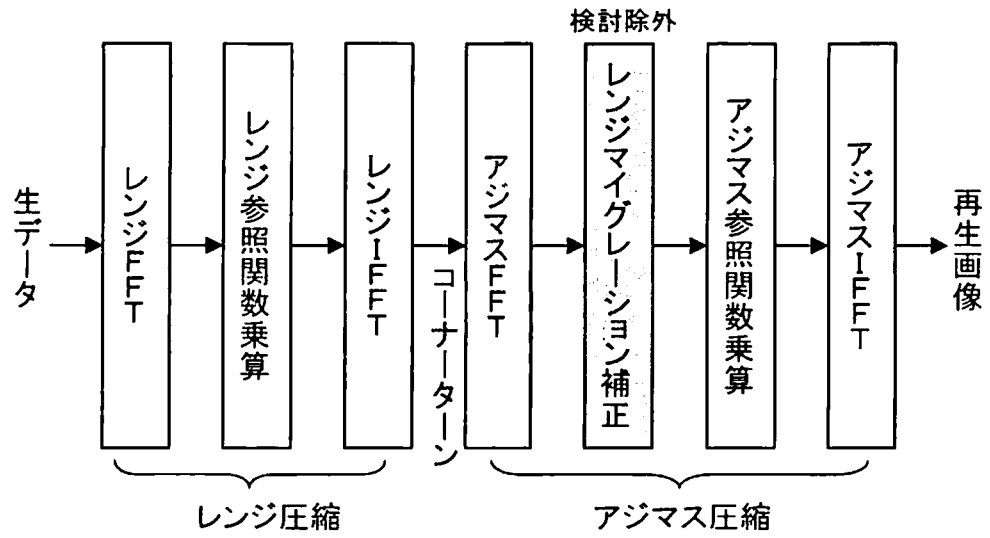

図 4.15: SAR 画像再生処理の流れ

ン，もしくは専用ハードウェア等を用いて処理を行っていた．

4.4.1 SAR 画像再生処理アルゴリズム

SAR は，雲霧等の天候に左右されず，高い分解能で地表を撮像することができるセンサである [36]．SAR では，飛行機等のプラットフォームからマイクロ波を送信し，反射波が返ってくるまでの時間・強度から対象物体までの距離情報を測定する．測定したデータを SAR 画像再生と呼ばれる処理で重ね合わせることにより画像を再生する．

SAR 画像再生のアルゴリズムは種々提案されているが，ここでは最も一般的に使用されている，レンジ - ドップラーアルゴリズム [37] を対象とする．このアルゴリズムの基本的な処理の流れを図 4.15 に示す．レンジ - ドップラーアルゴリズムは，アジマス方向（プラットフォーム進行方向）とレンジ方向（アジマス方向に対して垂直な方向）の受信データそれぞれに，参照関数と呼ばれる送信波を表すデータとの相互相関をとることにより 2 次元ホログラムを得る処理である．

相互相関処理には単純に乗算を行う方法（時間領域処理と呼ぶ）と FFT を用いる方法（周波数領域と呼ぶ）があるが，図 4.15 のように FFT を用いて周波数領域で行う方が一般的であり演算量が少なくすむ．しかし，時間領域処理の方はデータの分割が容易であり，並列処理を行う場合にノード間のデータ転送量を少なくできる．

表 4.7: FPGA 内部のコア回路の性能

回路	動作周波数	使用 LE 数
バタフライ演算器	40MHz	2,993 LE
8bit 乗算器	35MHz	135 LE

4.4.2 SAR 画像再生処理の実装方法

SAR 画像再生処理の時間領域処理，周波数領域処理の両方の手法について，RASH の各構成での処理方式と演算時間の検討を行う．また，今回の検討では，レンジマイグレーション補正は対象から除外した．レンジマイグレーション補正は SAR 画像再生処理を行う上では重要な処理であるが，本研究では，装置の規模や処理時間の試算を目的としているため，最も演算量に影響する相互相関処理のみを対象とすることにした．また，以下の条件で検討を行い，各接続の転送速度は RASH での実測値から次のものを使用した．FPGA 内部の回路はシミュレーションによる見積りから表 4.7 の値を使用した．

(1) SAR の生データのサイズはレンジ方向 8K ポイント，アジマス方向 8K ポイント
(2) 参照関数のデータは 1K ポイント
(3) 各ポイントは実数部 8bit，虚数部 8bit
(4) CompactPCI のデータ転送速度：128Mbps
(5) ローカルバスの転送速度： 128Mbps

次に実装方法について述べる．SAR 画像再生処理を実装する方法として以下の 3 方式の性能を評価する．

(1) **RASH 単体での実装方法**

ドータボードを搭載せず，基本構成だけの構成である．この構成ではメモリ不足により時間領域処理が不可能であるため，周波数領域で処理を行う．レンジ方向の処理（レンジ圧縮処理）を EXE ボード上の FPGA で行った後，CPU ボード上のメインメモリでコーナーターンと呼ばれるデータの転地処理を行い，再び EXE ボードでアジマス方向の処理（アジマス圧縮処理）を行う．1 個の FPGA で 8K ポイントの FFT を行う場合，FPGA 内部のメモリ不足により処理が困難である．このため，図 4.16 のように複数の FPGA に処理を分けレンジ／アジマス圧縮を行う．FPGA にメモリが接続されていないため，FFT の中間結果は EXE ボード上のローカルメモリに格納する．

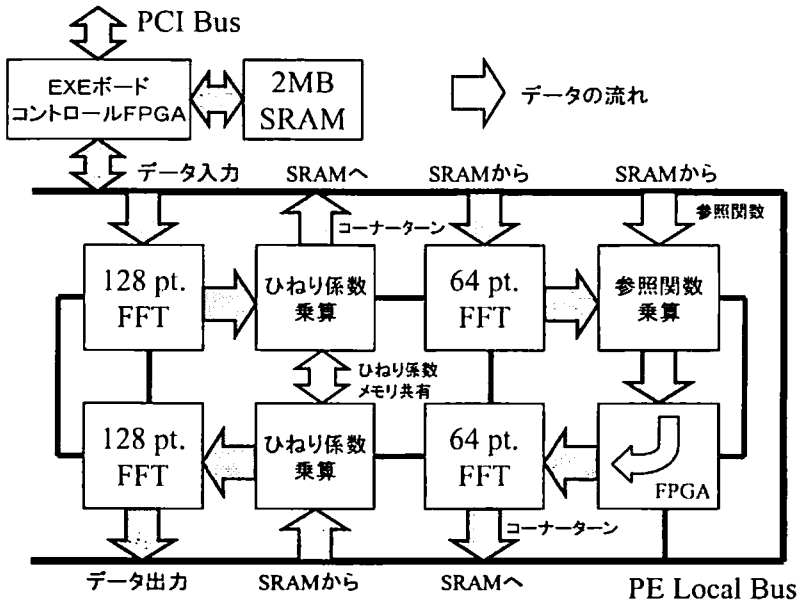

図 4.16: 周波数領域処理の実装方法

(2) **RASH ＋ドータボード（メモリ）での実装方法**

EXE ボードにメモリドータボード（RDM）を搭載してメモリを補強した構成である．これにより，EXE ボードでのメモリ不足が解消され，時間領域による処理が可能になる．この場合，図 4.17 のように，EXE ボード上の半分の FPGA に乗算器を構成することにより，レンジ圧縮を時間領域処理で行い，残りの FPGA でアジマス圧縮を周波数領域で行うことで，レンジ圧縮とアジマス圧縮をパイプラインで処理する．これにより，レンジ圧縮処理での演算量が増えるが，コーナーターンによる EXE ボード間のデータ転送が無くなるため処理時間が短縮される．

(3) **RASH ＋ドータボード（メモリ＋ I/O）での実装方法**

EXE ボード間の通信ボトルネックを補うため，ドータボードに通信ポートを追加した構成を想定した．ドータボード上には 100Mbps の通信ポートが 2 個あると仮定した．これは，今後 RDM 上に USB2.0（最大性能 480Mbps）の機能を実装できるならば，実現可能な値である．この場合，図 4.16 のような構成で，アジマス／レンジ圧縮共に周波数領域で処理を行う．ただし，(1) の RASH 単体の場合と異なり，FFT の中間結果は各 FPGA に直接接続されたドータボード上のメモリへ格納する．

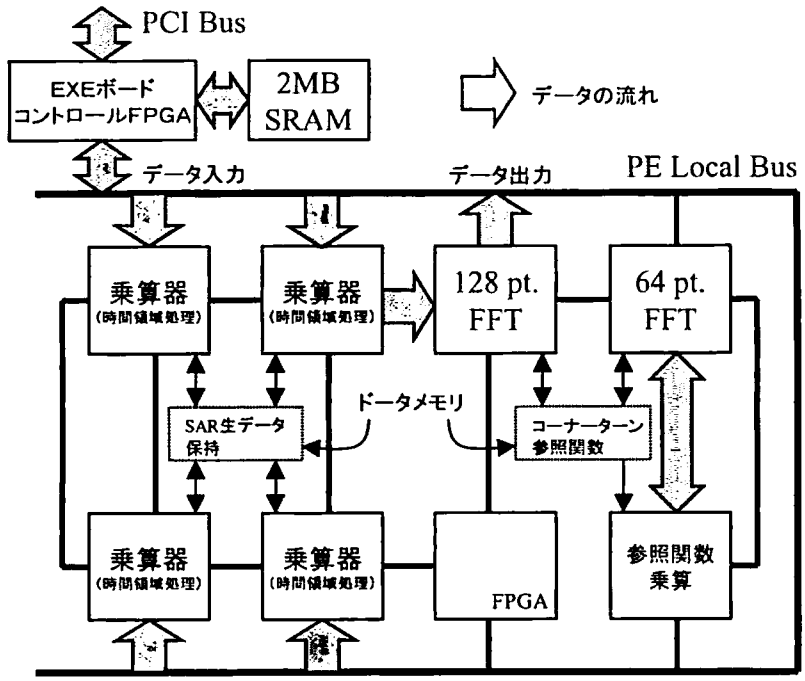

図 4.17: 時間領域処理の実装方法

4.4.3 評価結果

4.4.2 節の構成・条件で，EXE ボード上の回路動作を想定して，1 行（8K ポイント）のデータの処理にかかる演算時間，データ転送時間等を試算し，RASH で SAR 画像再生を行った場合の総処理時間を見積もった．その結果を表 4.8 に示す．構成は全て 1 ユニットであり，表には各構成において最良となる処理時間のみを記載した．また，市販 DSP システムでの検討結果 [38] も合わせて示す．DSP での検討は，Analog Device 社製の DSP である SHARC（ADSP-21060,40MHz）を搭載した Mercury Computer Systems 社の DSP ボードで行った．

SAR 画像再生の処理時間は，各演算ボードへのデータ転送時間と，各演算ボードでの演算時間のうち，より大きい方の値になる．データ転送時間と演算時間の影響を示すため，表 4.8 には外部からのデータの入出力時間を考慮しない処理時間を “演算時間” として示し，1 行のデータの処理時間を記載した．なお，表にある 1 行のデータの演算処理時間は，使用デバイス数（FPGA,DSP）で割った仮想的な値である．これらの値から，各構成における処理時間は，データ転送時間が演算時間と同じかそれより大きいことが分かる．

表 4.8: SAR 回路の実装法による性能比較

項目	RASH			市販 DSP システム
	基本構成	ドータボード拡張		
		メモリ拡張 (RDM)	メモリ & I/O 拡張	
処理時間	49 秒	24 秒	8 秒	10 秒
演算時間	36 秒	24 秒	8 秒	7 秒
1 行分のデータの演算時間	2.28 ms	3.32 ms	0.48 ms	0.045 ms
1 行分のデータの転送時間	3.7 ms	3.7 ms	0.38 ms	0.52 ms
処理方式	周波数領域処理	時間領域処理	周波数領域処理	周波数領域処理
演算ボード数	6 枚	3 枚	3 枚	8 枚
ドータボード数	-	6 枚	6 枚	-
FPGA/DSP 数	48 個	24 個	24 個	96 個

また，DSP と比較して，半分のチップ数でほぼ同程度の性能が得られることが分かる．

4.5 考察

本章では，アタッチド・プロセッサ方式のリコンフィギャラブル・コンピューティングシステムである RASH の構成と応用事例について報告した．RASH は従来のシステムよりスケーラビリティを向上し，より適応範囲の広いリコンフィギャラブル・コンピューティング・システムを目指して研究・開発してきた．そして，本研究では計算集約的な応用として暗号解析処理と SAR 画像再生処理を用いて評価した．

暗号解析処理では DES 暗号の鍵探索を実行する回路を実装した．性能比較では Intel 社の Pentium プロセッサ (300 MHz) に対して，ALTERA 社の FPGA である FLEX10K100A-1 単体（39.5 MHz）で約 35 倍の性能を示し，マシンレベルでは 1,700 倍以上を示した．これはアルゴリズムのハードウェア化によって，ノイマン・オーバヘッドが解消され，処理対象に内在する並列性を十分に引き出すことができた結果と考えられる．また，RASH 上の DES 暗号の鍵探索回路も 2 種類の実装方法を試行し，実装アーキテクチャの改善によって性能向上が図られた事を確認した．これはリコンフィギャラブル・コンピューティングの特徴であるアルゴリズムとアーキテクチャの最適化を実現した事例として注目に値

する．

また，SAR 画像再生処理では，周波数領域処理と時間領域処理の 2 つのアルゴリズムを検討し，処理時間とアーキテクチャのトレードオフを図れることを示した．しかしながら，この応用では全体の処理時間に対しデータ転送時間の比率が高く，これが性能抑制要因になりえることが分かった．リコンフィギャラブル・コンピューティングシステムでは，命令フェッチが無いためノイマンボトルネックの解消が期待できるが，一方でアタッチド・プロセッサ方式ではデータに対しては従来の並列計算機と同様なデータ配置問題等が存在する．今後，アタッチド・プロセッサ方式のリコンフィギャラブル・コンピューティングシステムを研究する上で十分に検討をする必要がある．

次に，FPGA への再構成時間と処理時間の関係を考察する．暗号解析処理で示したように，DES 暗号の鍵探索範囲は 2^{56} と非常に広いため，FPGA 単体の鍵探索性能 29.6*M* 鍵/秒 程度の性能では 77 年を必要とする．並列化を推し進めて，仮に 6 枚の EXE ボードを搭載した RASH ユニットを 30 台並列で動作させたとしても，FPGA 数は高々 1,440 個（30*unit* × 6 ボード × 8 個）であり，全数探索に約 20 日を要する．一方，FPGA の再構成時間は 190*ms* 程度であり，処理時間に比べたら十分に無視できる．したがって，このように再構成時間と処理時間とが極端に差がある場合は，リコンフィギャラブルコンピューティング・システムが持つ再構成時間の問題は発生しない．

以上から，リコンフィギャラブル・コンピューティングの応用としては，再構成間隔，再構成時間，処理時間の関係が重要であることが分かる．特に，従来の FPGA を用いたリコンフィギャラブル・コンピューティングシステムの場合では，再構成時間が長いため適応範囲が制限される．したがって，リコンフィギャラブル・コンピューティングの適応範囲を広げるためには，再構成時間が処理時間に対して十分に短いかまたは，処理時間に隠蔽されるようなデバイスが必要である．

第 5 章

リコンフィギャラブル・ロジックの提案と評価

本章では，リコンフィギャラブル・コンピューティングに使用するデバイスについて議論し，FPGA や PLD に変わる新しいリコンフィギャラブル・ロジックの論理ブロックの構成について研究した結果を報告する．

5.1 概要

これまで既存の FPGA アーキテクチャをシステム LSI に組み込む方式や，既存の FPGA デバイスを多数用いたリコンフィギャラブル・コンピューティングシステムを議論してきた．

しかしながら，現在市販されている FPGA をリコンフィギャラブル・コンピューティングに利用する場合，第 2 章で述べたように多くの問題点が見つかっており，それらの改善なしでは限定的な用途でしか十分な性能を発揮することが難しいことが判ってきた [3]．問題点の多くは，市場が求めている FPGA の能力とリコンフィギャラブル・コンピューティングが求めている能力とが異なることが原因と考えられる．したがって，リコンフィギャラブル・コンピューティングを実現するためには，開発試作に使われている FPGA の流用ではなく，リコンフィギャラブル・コンピューティングに適したリコンフィギャラブル・ロジックの開発が必要になってきている．

本章ではリコンフィギャラブル・ロジックの基本要素である LUT（Look Up Table）の粒度（論理規模）について議論し，構成データキャッシュと高機能化した LUT を持つリコンフィギャラブル・ロジックデバイス向き論理ブロックを提案する．そして，実用的な評価回路を用いた評価結果について述べる．[39]．

5.2 リコンフィギャラブル・ロジックの要件

従来の FPGA を用いたリコンフィギャラブル・コンピューティングは，ハードウェア化による性能向上を再構成時間のオーバヘッドが相殺することが多い．また，実装回路規模はデバイスに依存し，それ以上の処理をしたいときには処理ができないか，または極端な性能低下が見られた．これらの問題のためリコンフィギャラブル・コンピューティングの用途は限定され，DNA パターンマッチング [5] や前章で述べた暗号解析処理 [28] 等の処理内容の変更頻度が少ないアプリケーションでしか十分な効果が得られていないのが現状である．

元来 FPGA は ASIC 等と比べて実装密度が低く，動作速度も遅いと言う欠点がある．リコンフィギャラブル・コンピューティングでは処理するデータや内容に応じてデータ幅や演算器構成などを最適化することで欠点を補い，一層の性能向上を図るものだが，その最適化に時間が掛かっては元も子もない．また，従来の ASIC では全機能を LSI 上に実現しており，ある時点で使われていない部分も電力を消費しているのに対し，リコンフィギャラブル・コンピューティングではある時点で必要な処理のみを実装することで，低コスト化や低消費電力化が期待できる．しかし，これも再構成時間が長かったり，回路規模が固定されているようだと実現が難しい．

したがって，リコンフィギャラブル・ロジックに求められる要件は，

(1) 実装密度と動作速度を向上すること
(2) 再構成時間が短いこと
(3) 回路の仮想化実現すること

であるといえる．

（1）の実装密度と動作速度が低い原因は，LUT による実装が同じ論理を構成するゲートより大きいという主にテクノロジの問題であるが，それ以外にも回路を実装するための LUT の粒度というアーキテクチャの問題でもある．（2）の再構成時間の問題と（3）回路の仮想化実現はデバイス・アーキテクチャの問題である．これらの問題の解決には次に示す機能の実現が望まれている．

(a) 部分再構成機能
(b) マルチコンテキスト機構

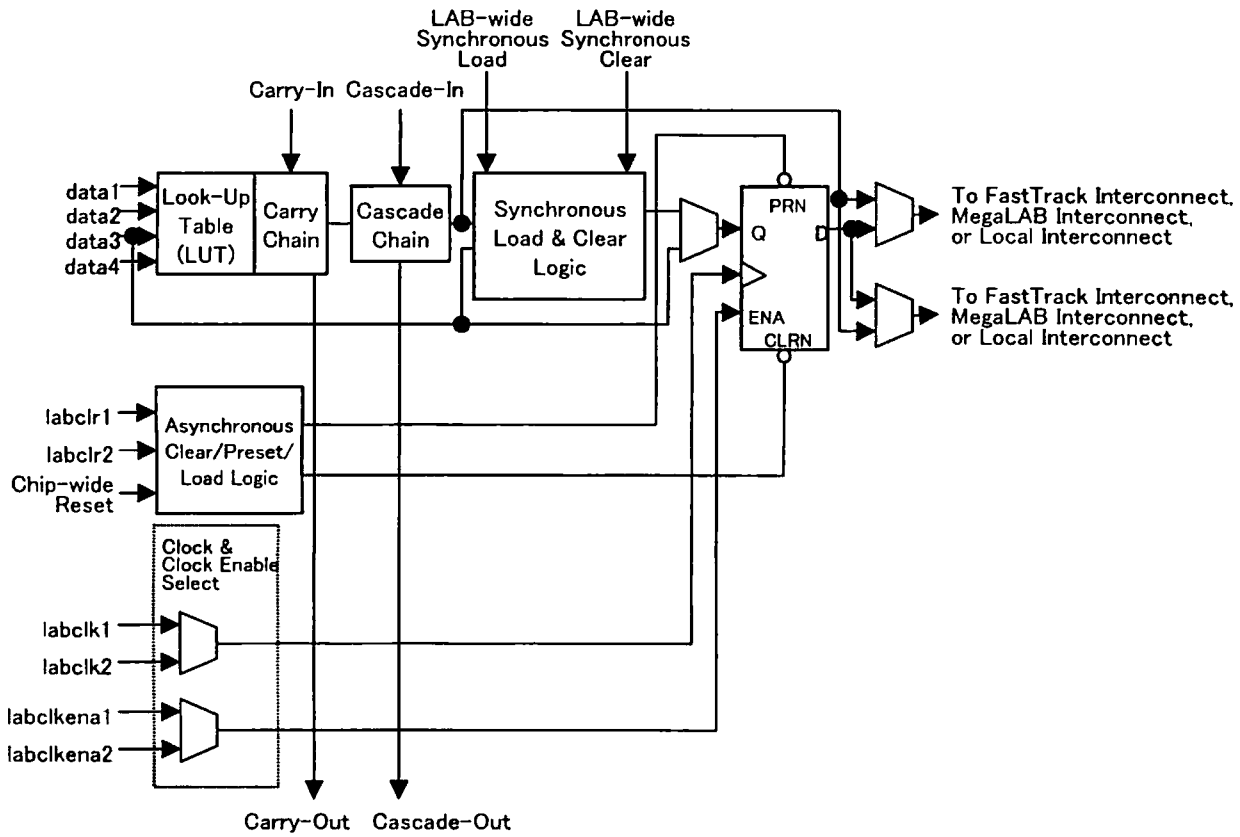

図 5.1: APEX20K の論理ブロック構成

これら機能を新たに論理ブロック内に追加する必要がある.

図 5.1 は Altera 社の APEX20K シリーズ [46] の論理ブロックである. 図からも分かるように現行の FPGA の典型的な論理ブロックには, FlipFlop (FF), クロックとリセットのセレクタ, カスケードチェーン用ロジックなど多くの固定回路が含まれている. ここに先に述べた機能を追加することになる. したがって, リコンフィギャラブル・ロジックは, 従来の FPGA より多くの固定回路を持ち, 相対的に LUT が占める面積の比率が下がることを意味する.

一方, FPGA の LUT の入力数と面積の関係についての Rose らの研究 [40] によると, いくつかの回路を異なる LUT 粒度に実装した場合, 平均的には LUT の入力数が 4 の時に最小面積になり, 2-LUT では 4-LUT の 1.5 倍, 7-LUT では 4-LUT の 2 倍以上面積になるという結論を得ている. すなわち, LUT への入力数の増加は必要な LUT 数を減少させるが, 同時に LUT あたりの面積は増加させ, LUT 数× LUT 面積を最小化するポイントが 4-LUT ということを示している. また, Rose らの速度と LUT 粒度の関係について

の研究 [41] では速度性能が最適なのは 5-LUT という結論を得ている．各 FPGA ベンダが論理ブロック内の LUT を 4-LUT や 5-LUT を採用している理由がここにある．

しかしながら，Rose らの研究対象は従来の FPGA である．リコンフィギャラブル・ロジックでは，先に述べたように論理ブロック内に様々な機能が追加され，Rose らの評価モデルとは前提が異なってきている．そのため，彼らの研究成果をそのままリコンフィギャラブル・ロジックに適用することができない．したがって，新たにリコンフィギャラブル・ロジックに適した LUT 粒度を面積と速度から再評価する必要性が出てきたといえる．

5.3 リコンフィギャラブル・ロジックの提案

本節では本研究で提案するリコンフィギャラブル・ロジックの構成について述べる．

5.3.1 論理ブロックの構成

本研究で提案するリコンフィギャラブル・ロジック向き論理ブロックは，複数の LUT 構成を持つマルチコンテキスト・マルチグレイン LUT（Multi-Context, Multi-Grain LUT, 以下 MCMG-LUT と略す）と複数のコンテキストを格納するためのメモリである構成データキャッシュ（Configuration Data Cache, 以下 CDC と略す）を搭載する．

図 5.2 の構成例では，6 入力 3 出力 LUT を 6 つのモード（(a) ～ (f)）で使用する．図中の n-LUT は n 入力 LUT を示す．前半の 3 モード (a) ～ (c) は，LUT をクラスタ化している．(a) は 2 入力 LUT を 3 個として機能するモードであり，(b) は 3 入力 LUT を 2 個として機能するモードである．(c) は 6 入力 LUT である．後半の 3 モード (d) ～ (f) は，LUT をマルチコンテキスト化するモードで、複数のコンテキストを切替えながら動作が可能である．これらのモードではコンテキスト数と LUT 粒度のバランスを選択できる．

また，CDC には，MCMG-LUT へ書き込まれるコンテキスト（回路データ）と MCMG-LUT の構成モードを決定するモード変更用ビット等が複数セット格納され，LUT 部が動作中であっても，専用のデータ線によって CDC の内容を書換え可能な構成をとる．そのため CDC への書き込み動作は，MCMG-LUT の動作には影響しない．

以上から，MCMG-LUT は従来の LUT より実装効率を改善と，回路毎の LUT 最適化，構成データ量の抑制等の効果が期待できる．一方，CDC は搭載可能な回路容量の増加，すなわち実装密度向上が期待でき，さらに再構成時間の隠蔽等，従来の FPGA に欠如していた機能を論理ブロックに加える．

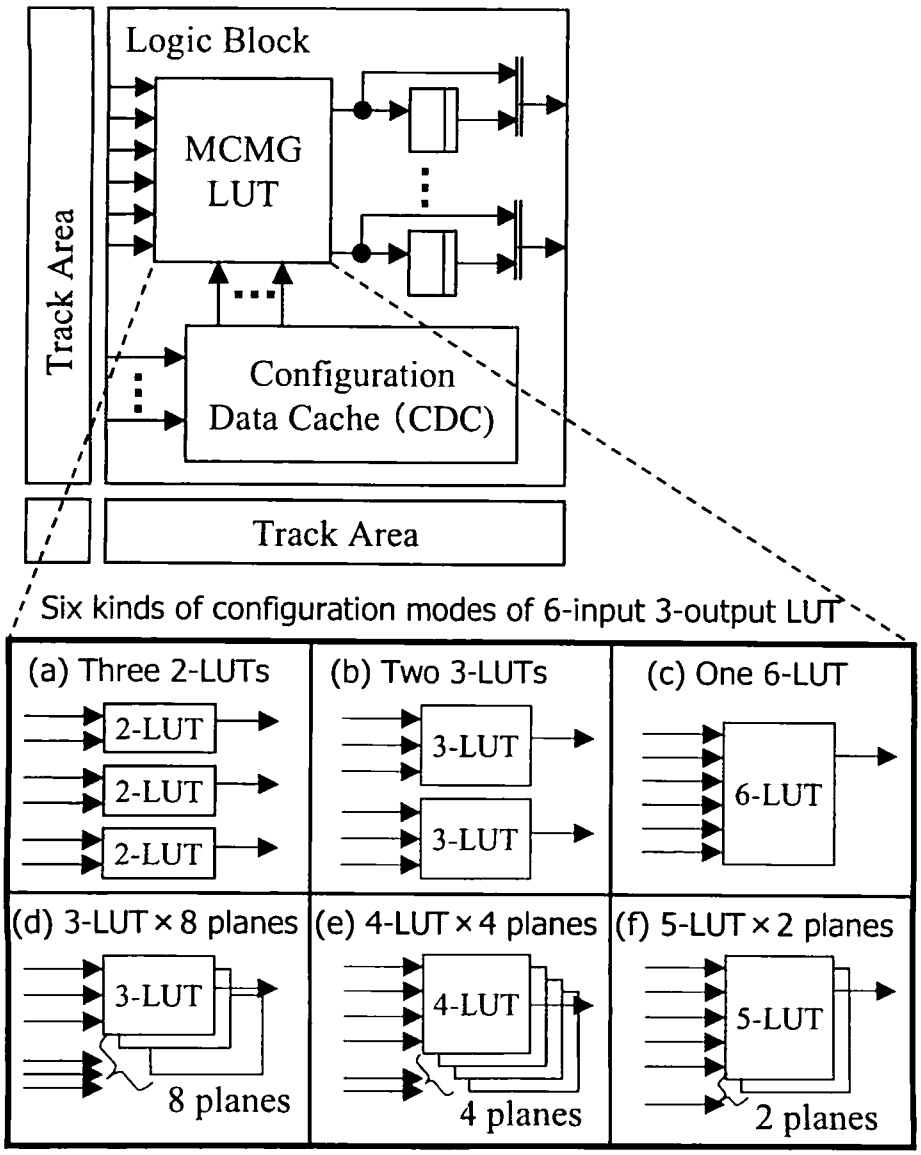

図 5.2: リコンフィギャラブル・ロジック向き論理ブロックの構成

5.3.2 MCMG-LUT の構成例

図 5.3 に前述の MCMG-LUT の内部構成例を示す．A～F が入力，Q, V, W が出力である．また，X, Y, Z はモード変更用メモリを示し，LUT を構成するメモリは 3 入力 2 出力のメモリアレイ 4 個で構成している．F0, F1 は入力セレクタの切り替え信号であり，モード変更用メモリ X, Y, Z によって決められる．F2 は出力 V のセレクタ切り替え信号であ

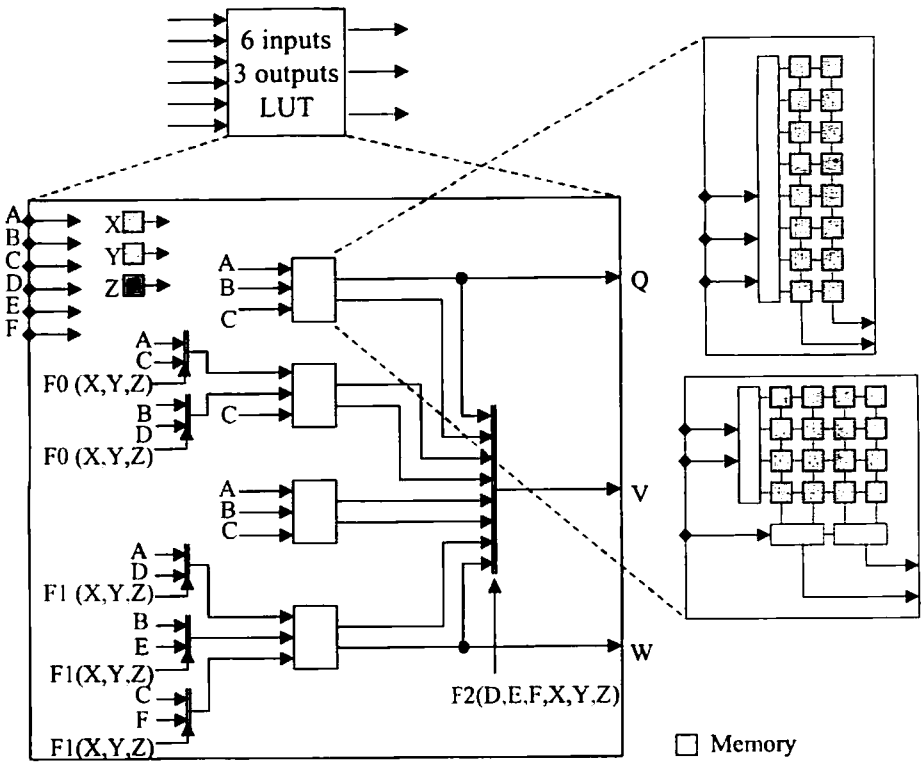

図 5.3: MCMG-LUT の構成例

り，入力信号 D, E, F およびモード変更用メモリ X, Y, Z によって決められる．

5.4 評価モデルおよび計算機実験環境

本節では，従来の FPGA の特性評価および提案するリコンフィギャラブル・ロジックの評価に用いる面積評価モデル，遅延評価モデルを示し，実装効率，実装密度，および構成データ量の算出方法を定義する．また，評価を行うための計算機実験環境と評価回路についても言及する．

5.4.1 面積評価モデル

図 5.4 に示す面積評価モデルは，規則的に配置された独立した論理ブロック部と，水平・垂直方向の配線チャネル部からなる．ただし，階層配線構造等は持たない．各トラックの幅はビット面積 BA（Bit Area）の平方根とし，論理ブロックは正方形と仮定する．また，評価モデルにおける単位面積 A_{LB} は，論理ブロックと隣接する水平・垂直方向の配線

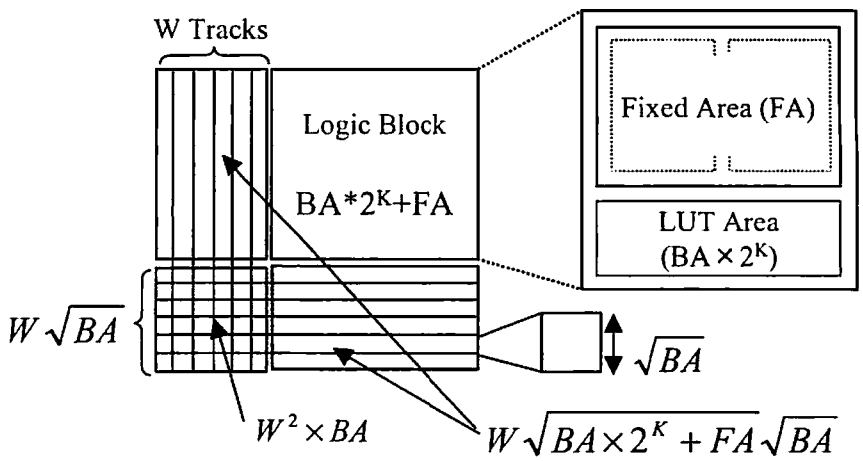

図 5.4: 面積評価モデル

領域の面積の合計である（式（5.1））．W は配線トラック数，FA（Fixed Area）は論理ブロック内の固定領域の総面積である．表 5.1 に各パラメータの定義を示す．

$$A_{LB} = (W^2 \times BA + 2W\sqrt{BA \times 2^K + FA}\sqrt{BA}) + (BA \times 2^K + FA) \tag{5.1}$$

このモデルは，トロント大の Rose らのモデル [40] から W と FA の二箇所の変更を加えたモデルである．本論文では，W を式（5.2）のように LUT への入力信号数 K の関数とする．a は LUT への入力信号数 K に対してトラックで必要とする割合を示す使用率係数である．b はクロック，リセット等の固定配線数である．一方，FA は，FF 等の固定領域の面積 FMA（Fixed Miscellaneous Area）と CDC のメモリ領域，MCMG-LUT のモード変更用メモリおよび制御回路領域の合計面積であり，式（5.3）のように定義する．BN は CDC を構成するメモリのビット数，R_a は BA に対するコンテキスト用メモリの面積比，CLA（Control Logic Area）は MCMG-LUT 内のモード変更用メモリおよび制御回路などの面積である．従来の論理ブロックは，CLA および BN をゼロとした場合と同じである．また，CDC 用メモリのテクノロジが LUT のメモリと異なる場合は，R_a を用いて調整する．

$$W = a \times K + b \tag{5.2}$$

$$FA = FMA + BN \times R_a \times BA + CLA \tag{5.3}$$

表 5.1: 面積評価モデルのパラメーター一覧

パラメータ	説明
IA	実装回路面積
N_{LB}	論理ブロック数
A_{LB}	論理ブロックの単位面積（配線領域を含む）
K	LUT への入力信号数（LUT の粒度）
BA	LUT を構成するメモリの単位面積
W	配線トラック数
a	K のトラック上の使用率係数
b	リセットやクロックなどの固定配線数
FA	論理ブロック内の固定領域の総面積
FMA	FA 内の FF などの固定領域の面積
BN	CDC を構成するメモリのビット数
R_a	BA に対するコンテキスト用メモリの面積比
CLA	MCMG-LUT のモード制御回路の面積

以上から，評価回路を実装した時の実装回路面積 IA（Implementation Area）は，実装に要した論理ブロック数 N_{LB} と単位面積 A_{LB} の積と定義する（式（5.4））．

$$IA = N_{LB} \times A_{LB} \tag{5.4}$$

次に，本評価で用いる各パラメータの値を示す．文献 [40] から 1.25μm CMOS プロセスの時，BA は 400μm^2，FMA（文献 [40] の FA に相当）は 5,100μm^2 となっており，4-LUT 論理ブロックの単位面積 A_{LB4} は 182,133μm^2 である．本論文もこの値を用いる．また，CLA は 2,250μm^2 と仮定した．これはモード変更用メモリの 3 ビット（1,200$\mu m^2 = BA \times 3$）と制御回路（1,050μm^2）の合計である．制御回路は文献 [40] の FF を含まない FA 値（2,100μm^2）の半分程度の面積と見積もっている．また，K の使用率係数 a は 1 とし，b はリセット等の専用配線とグローバル配線を考慮して 12 本とする．コンテキスト用メモリの BA 面積比は LUT と同じと仮定し，R_a は 1.0 とする．

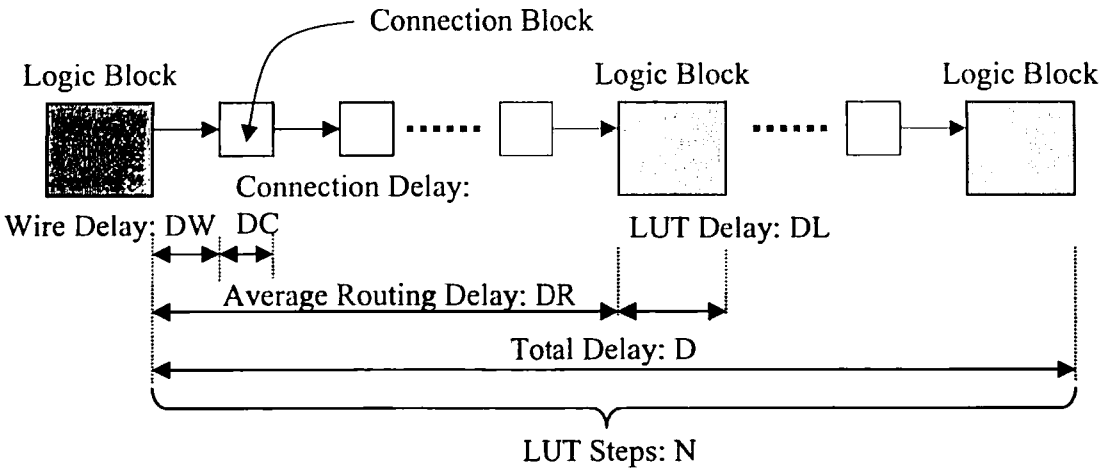

図 5.5: 遅延評価モデル

表 5.2: 遅延評価モデルのパラメータ一覧

パラメータ	説明
D	FF 間の総遅延（クリティカルパス遅延）
DW	ワイヤ遅延
DC	コネクションブロック遅延
N	FF 間の論理ブロック段数
R_d	DW-DC 遅延分配係数
DR	平均配線遅延
DL	論理ブロック遅延

5.4.2 遅延評価モデル

遅延評価モデルは，Rose らのモデル [41] をより詳細にしたモデルを採用する．本研究では，図 5.5 に示すように，平均配線遅延 DR をワイヤ遅延 DW とコネクションブロック遅延 DC に分解する．総遅延 D は，論理ブロック段数 N，平均配線遅延 DR および論理ブロック遅延 DL から式（5.5）を用いて算出している．各パラメータは表 5.2 の通りである．

$$D = (DR + DL) \times N \tag{5.5}$$

$$DR = \Sigma(DW) + \Sigma(DC) \tag{5.6}$$

一方，配線構造が不明の場合，$\Sigma(DW)$，$\Sigma(DC)$ は DW-DC 遅延分配係数 R_d と DR を用いて式（5.7），式（5.8）と定義する．

$$\Sigma(DW) = DR \times R_d \tag{5.7}$$
$$\Sigma(DC) = DR \times (1 - R_d) \tag{5.8}$$

一般に，LUT の粒度が大きくなると，(a) 論理ブロック遅延 DL が大きくなる，(b) 論理ブロックの面積増によって論理ブロック間のワイヤ遅延 DW が大きくなる，(c) LUT あたりの論理量が増えるために N が小さくなる，等の変化が起こる．

本研究では，論理ブロック間の配線長は，論理ブロックの辺長に比例すると仮定する．したがって，本論理ブロック間の平均配線遅延 DR_6 は，本論理ブロックの単位面積 A_{LB6} の平方根と 4-LUT 論理ブロックの単位面積 A_{LB4} の平方根との比と，4-LUT での平均配線遅延 DR_4 との積として求められる（式（5.9））．

$$DR_6 = \frac{\sqrt{A_{LB6}}}{\sqrt{A_{LB4}}} \times DR_4 \times R_d + DR_4 \times (1 - R_d) \tag{5.9}$$

次に，遅延評価に用いる各パラメータの値を示す．文献 [41] から 1.25μm CMOS プロセスを採用した場合，4-LUT 論理ブロック遅延 DL_4 は 1.71 ns，平均配線遅延 DR_4 は 4.0ns，6-LUT 論理ブロック遅延 DL_6 は 2.38 ns である．従来の LUT に対してはこれらの値を用いる．MCMG-LUT の論理ブロック遅延（DL_{n6}）は，従来の 6-LUT よりも 10%程度遅延が増加すると見積り 2.62 ns とする．また，R_d は 0.1 とする．

5.4.3 実装効率の定義

ここでは，実装効率 IE（Implementation Efficiency）を定義するために，占有ビット数 M_o と使用ビット数 M_u を導入する．M_o は総論理ブロック数 TN_{LB} 内に含まれる全 LUT のメモリビット数を示し，式（5.11）によって算出する．一方，M_u は，各論理ブロックの入力信号線数によって，LUT で使用されるメモリのビット数が異なることから，各入力の LUT 数 N_{LBn} を用いて，式（5.12）によって算出する．n は LUT への実際の入力数である．以上から，実装効率 IE は，M_o に対する M_u の割合と定義する（式（5.10））．

例えば，4 入力 LUT の論理ブロックを 2 つ使用している回路の場合，占有ビット数 N_{ob} は $2^4 \times 2 = 32bits$ であるが，2 つの LUT 共に 2 入力しか使用しなければ，使用ビット数 N_{ub} は $2^2 \times 2 = 8bits$ となる．したがって，実装効率 IE は，$(8/32) \times 100 = 25\%$ である．

$$IE = M_u / M_o \times 100 \tag{5.10}$$

$$M_o = TN_{LB} \times 2^K \tag{5.11}$$

$$M_u = \sum_{n=1}^{K} (N_{LBn} \times 2^n) \tag{5.12}$$

5.4.4 実装密度の定義

実装密度は単位面積あたりの実装論理量と定義されるが，ここでは従来の FPGA との比較のために，4-LUT を持つ論理ブロックに対する相対論理量を定義し，それを用いて実装密度を再定義する．

相対論理量 RLC（Relative Logic Capacity）は 4-LUT 論理ブロックに対して同じ回路を実装するのに要する対象論理ブロック数の比と定義する．例えば，10 個の 4-LUT 論理ブロックを必要とした回路が，8 個の 5-LUT の論理ブロックで実装できるなら，5-LUT の論理ブロックは RLC が $10/8 = 1.25$ となり，4-LUT の論理ブロックより 1.25 倍の論理量を持つことになる．

次に，4-LUT の論理ブロックに対する対象論理ブロックの相対的な実装密度 RID（Relative Implementation Density）を式（5.13）と定義する．NC はコンテキスト数，AR（Area Ratio）は面積比である．また，NC と RLC の積を相対的な回路容量とする．例えば，4-LUT を 2 個持つ論理ブロックが 4-LUT を 1 個の論理ブロックに対して 1.5 倍の面積であると仮定すると，4-LUT を 2 個持つ論理ブロックの実装密度は，$NC = 2$，$RLC = 1$，$AR = 1.5$ から $RID = 2 \times 1/1.5 = 1.33$ となり，4-LUT を 1 個の論理ブロックの 1.33 倍である．

$$RID = NC \times RLC / AR \tag{5.13}$$

5.4.5 構成データ量の算出方法

構成データは LUT に格納されるコンテキストと配線上のコネクションブロックのデータからなる．コンテキスト分の構成データ量は，LUT 使用数と LUT のビット数の積である．コネクションブロック分は，配線数（ネット数），論理ブロック間の配線の平均接続点およびコネクションブロックのビット数の積として見積もる．本研究では平均接続点数を 4 箇所と仮定し，各コネクションブロックは，1 接続点あたり 1 bit のメモリを要し，トラック数分のメモリで構成される．さらに，本方式の論理ブロックでは 3 ビットのモード変更用メモリも構成データに加味する．

従来の LUT の構成データ量（CD_c）の算出式を式（5.14）に，本方式の論理ブロックの構成データ量（CD_n）を式（5.15）に示す．N_{LUT} は LUT 数，N_{NT} はネット数，N_{LB} は論理ブロック数である．

$$CD_c = N_{LUT} \times 2^K + N_{NT} \times 4 \tag{5.14}$$

$$CD_n = N_{LUT} \times 2^K + N_{NT} \times 4 + N_{LB} \times 3 \tag{5.15}$$

5.4.6 計算機実験環境

図 5.6 に，評価回路の実装フローを示す．Verilog-HDL で記述された評価回路は，ALTERA 社の MAX+PLUS II を用いて EDIF ネットリストを生成し，さらに，トロント大で開発された EDIF2BLIF[42] を用いて BLIF フォーマットへ変換する．そして，BLIF ネットリストを UCB の SIS[43] で 3-LUT から 7-LUT にマッピングし，同じくトロント大で開発された TV-Pack[44] を用いて FF と LUT のパッキングを行うことで，最終的なネットリストを得ている．

5.4.7 評価回路

評価回路の選定は重要な問題である．偏った評価にならないように回路を選定する必要があるが，現実には明確な基準を設けることは難しい．CDC とクラスタ化の評価では，MCNC ベンチマーク回路 [45] から表 5.3 に示した比較的大きな回路 10 種類と，表 5.4 の Veriog-HDL で記述された 6 種類の実用回路を用いる．

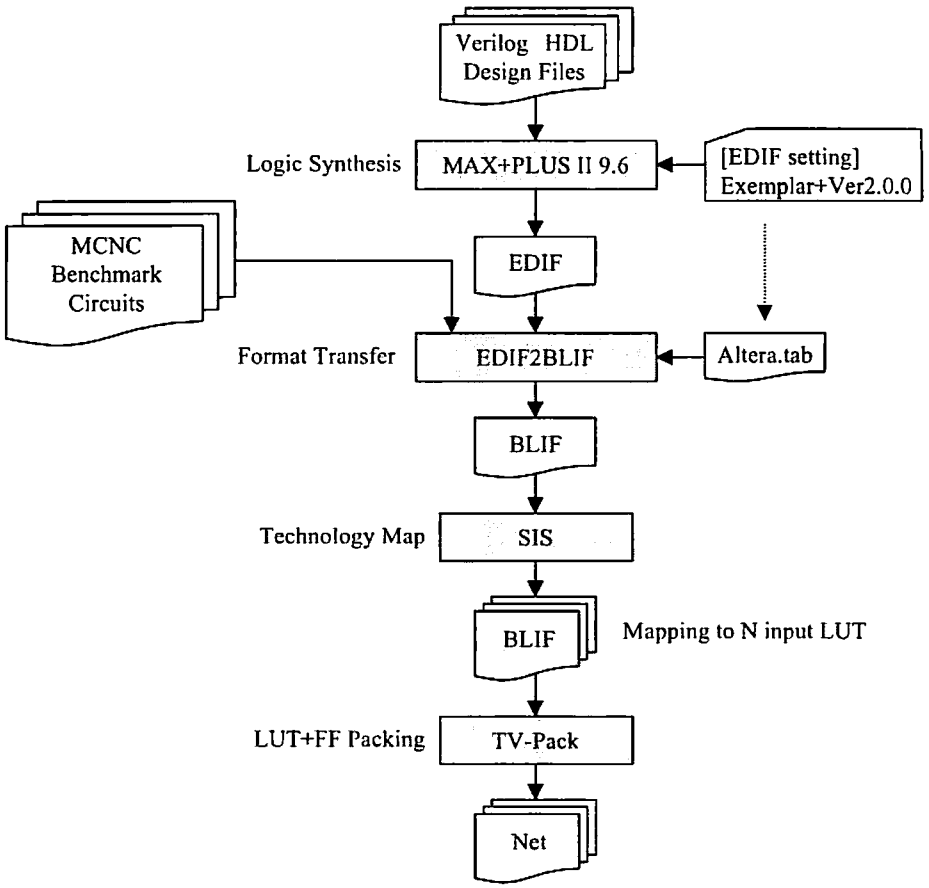

図 5.6: 評価回路の実装フロー

5.5 評価結果

5.5.1 使用論理ブロック数

表 5.5 は，全評価回路を前述の方法で，3〜7 入力の LUT にマッピングした時の論理ブロック数である．表中の n-LUT は n 入力 LUT を示す．表中の太字は最小論理ブロック

表 5.3: MCNC ベンチマーク回路一覧

回路名	
alu4	ex1010
apex2	ex5p
des	frisc
dsip	seq
elliptic	tseng

表 5.4: 追加評価回路一覧

回路名	説明
ACS4	ビタビ復号法の ACS 回路（4bit 精度）
DIV8	パイプライン割算器（8bit 精度.4 段）
FFT6	8 ポイント FFT 回路（6bit 精度）
DES16	16 ループ版 DES 暗号回路
MA8I	複素積和演算回路（8bit 精度）
SCU	DLX プロセッサ（命令デコーダ部）

数である．

最小論理ブロック数は，ほとんどの回路で 7-LUT が最小数になった．また，表中の対 4-LUT 比は，4-LUT の論理ブロック数に対する各粒度の論理ブロック数の割合を示し，その逆数を 4-LUT に対する相対論理量としている．例えば 6-LUT の場合，同じ回路を 4-LUT の約 70%の論理ブロック数で実装できるため，6-LUT は 4-LUT の約 1.45 倍の論理量を持つと言える．

5.5.2 実装面積

表 5.6 は，前節と同様に全評価回路を 3〜7 入力の LUT にマッピングした時の実装面積である．単位は mm^2 である．太字は最小面積を示す．

表から判るように，最小面積になる LUT 粒度は回路毎に異なっている．前節で示した論理ブロック数は，ほとんどが 7-LUT の時に最少数になったが，実装面積では，必ずしも LUT 粒度が大きいほうが有利とはならない．傾向としては，フリップフロップ間の論理量が多い回路は，LUT 粒度が大きい方が有利になり，論理量が少ない回路は LUT 粒度が小さい方が実装面積を小さくする傾向がある．

5.5.3 論理ブロック段数比

表 5.7 に各評価回路のクリティカルパスの論理ブロック段数を示し，表 5.8 に追加評価回路における各 LUT 粒度の 4-LUT に対する論理ブロック段数比を示す．また，MCNC ベンチマーク回路における 4-LUT と 6-LUT の論理ブロック段数比 N_6/N_4 は，文献 [41] に 0.7 程度であることが示されている．一方，追加評価回路では，表 5.8 に示したように 0.68〜0.84 で，平均 0.76 である．

表 5.5: 評価回路における使用論理ブロック数

回路名	LUT 粒度				
	3-LUT	4-LUT	5-LUT	6-LUT	7-LUT
alu4	1,854	1,323	1,067	922	**795**
apex2	2,249	1,589	1,377	1,269	**1,131**
des	2,127	1,539	1,312	1,180	**1,154**
dsip	1,852	1,371	1,169	**690**	1,279
elliptic	3,892	2,916	2,500	2,043	**1,988**
ex1010	5,229	4,039	3,427	3,202	**3,017**
ex5p	348	245	137	115	**104**
frisc	4,644	2,973	2,535	2,252	**2,191**
seq	2,101	1,501	1,290	1,177	**1,057**
tseng	1,185	957	874	784	**698**
ACS4	267	179	97	100	**83**
DIV8	459	391	313	259	**209**
FFT6	3,982	2,640	1,813	1,622	**1,159**
DES16	2,194	1,931	950	**726**	981
MA8I	1,000	647	421	406	**364**
SCU	1,055	839	625	**531**	534
合計	34,434	25,080	19,907	17,278	16,744
対 4-LUT 比	1.37	1.00	0.79	0.69	0.67
相対論理量	0.73	1.00	1.26	1.47	1.50

5.5.4 各 LUT への入力信号数

全評価回路をマッピングした結果として，実際に各粒度の LUT へ入力されている信号線数の割合を表 5.9 に示す．太字部分は，各 LUT 粒度の最大入力数へマッピングされた LUT の割合である．LUT 粒度が大きくなるにしたがって減少していくことがわかる．これは，マッピング・アルゴリズムにも依存するが，LUT 粒度を大きくするにしたがって，すべての入力ポートに信号を割り当てることができない LUT の割合が増えていくことを意味する．

表 5.6: 評価回路における実装面積

回路名	LUT 粒度				
	3-LUT	4-LUT	5-LUT	6-LUT	7-LUT
alu4	283.6	241.5	**239.5**	264.1	302.9
apex2	344.0	**290.0**	309.1	363.5	430.9
des	325.4	**280.9**	294.5	338.0	439.7
dsip	283.3	250.3	262.4	**197.7**	487.3
elliptic	595.3	**532.3**	561.2	585.2	757.5
ex1010	799.8	**737.2**	769.3	917.2	1,149.6
ex5p	53.2	44.7	**30.8**	32.9	39.6
frisc	710.4	**542.7**	569.1	645.1	834.8
seq	321.4	**274.0**	289.6	337.2	402.7
tseng	181.3	**174.7**	196.2	224.6	266.0
ACS4	40.8	32.7	**21.8**	28.6	31.6
DIV8	81.8	**71.4**	87.3	74.2	108.6
FFT6	633.6	511.1	**443.8**	465.8	443.1
DES16	345.4	364.2	213.3	**208.0**	398.2
MA8I	153.0	126.9	**105.3**	116.3	138.7
SCU	161.4	153.1	**140.3**	152.1	203.5

5.6 考察

5.6.1 構成データキャッシュの効果

ここでは，CDC の効果を見るために，本方式の CDC を含む 6-LUT 論理ブロック（MCMG-LUT はモード（c）を使用）と従来の FPGA で使われている 4-LUT 論理ブロッ

表 5.7: 最大 LUT 段数

回路名	LUT 粒度				
	3-LUT	4-LUT	5-LUT	6-LUT	7-LUT
ACS4	36	26	19	18	13
DIV	32	25	22	19	19
FFT6	25	19	15	16	15
DES16	23	18	14	14	10
MA8I	23	22	13	15	13
SCU	21	16	14	13	14

表 5.8: 4-LUT に対する論理ブロック段数比

回路名	各 LUT 粒度の論理ブロック段数比（対 4-LUT 比）				
	3-LUT	4-LUT	5-LUT	6-LUT	7-LUT
ACS4	1.38	1.00	0.73	0.69	0.50
DIV	1.28	1.00	0.88	0.76	0.76
FFT6	1.32	1.00	0.79	0.84	0.79
DES16	1.28	1.00	0.78	0.78	0.56
MA8I	1.05	1.00	0.59	0.68	0.59
SCU	1.31	1.00	0.88	0.81	0.88
平均	1.27	1.00	0.77	0.76	0.68

表 5.9: 各 LUT 粒度における入力数の割合

入力信号数	各 LUT 粒度における入力信号数毎の割合（%）				
	3-LUT	4-LUT	5-LUT	6-LUT	7-LUT
1 入力	0.91	1.11	1.47	0.08	0.90
2 入力	17.59	13.38	7.59	9.21	6.55
3 入力	**81.50**	22.58	19.87	7.23	8.12
4 入力	-	**62.94**	24.07	16.16	18.19
5 入力	-	-	**47.01**	25.62	12.29
6 入力	-	-	-	**41.70**	17.96
7 入力	-	-	-	-	**35.98**

クとを比較し，4-LUT 論理ブロックに対する相対面積と実装密度を求める．

4-LUT 論理ブロックに対する相対的な回路容量は，5.4.4 節で述べたように，5.5.1 節で求めた相対論理量と，MCMG-LUT および CDC 内に搭載できるコンテキスト数の積である．例えば，CDC 容量が 64 ビットの場合，本方式の論理ブロック内には，MCMG-LUT 分を含めて 2 つのコンテキストが保持できる．したがって，4-LUT に対する回路容量は，$2 \times 1.47 = 2.93$ となり，1 個の 4-LUT を持つ論理ブロックに対して，本方式の論理ブロックは 2.93 倍の論理回路が搭載できる．また，実装密度は式（5.13）から 4-LUT 論理ブロックに対する実装可能な回路容量に面積比を掛けた値である．

表 5.10: BN に対する 4-LUT 面積当たりの回路容量

CDC ビット数 BN (bit)	4-LUT に対する 面積比	回路容量	4-LUT 面積 当たりの実装密度
64	4.66	2.93	**0.63**
128	5.16	4.40	**0.85**
256	6.02	7.33	**1.22**
512	7.47	13.20	**1.77**
1,024	9.91	24.94	**2.52**
2,048	14.08	48.41	**3.44**
4,096	21.32	95.35	**4.47**
8,192	34.22	189.24	**5.53**

表 5.10 に，CDC のビット数 BN の変化に対する 4-LUT との面積比および回路容量，そして 4-LUT 面積当たりの回路容量を示す．表 5.10 から分かるように，4-LUT 面積当たりの回路容量は，CDC の容量が 128 ビットまでは低いが 256 ビット辺りで逆転し，それ以上では増加する．512 ビットでは 13.20 倍の回路容量を持つが，面積も 7.47 倍になるため，4-LUT 面積当たりの回路容量は約 1.77 倍になる．

次に，CDC の回路遅延への影響を考察する．表 5.11 は，CDC 内のビット数 BN ならびに 4-LUT と 6-LUT の論理ブロック段数比 N_6/N_4 が変化したときの，4-LUT に対する遅延比である．なお，N_6/N_4 の変化は 5.5.3 節から 0.7〜0.9 としている．表における太字は，4-LUT から 6-LUT への変更による論理ブロック段数の減少が CDC による配線遅延の増加を上回り，その結果として総遅延 D が小さくなるケースである．

また，テクノロジの変化によって R_d の比率が変わった場合を図 5.7 に示す．なお，N_6/N_4 は 0.75 としている．一般に，R_d はプロセス・テクノロジに依存し，プロセスが微細化するとスイッチング遅延に対して配線遅延が相対的に大きくなる（R_d は大きくなる）．図 5.7 から，R_d が 0.1 で CDC 容量が 2,048 ビットまでと，R_d が 0.2 で CDC 容量が 128 ビットまでとは，4-LUT より遅延値が小さい（$RD < 1.0$）．しかし，それ以外の CDC 容量では 4-LUT より遅くなる．

なお，4-LUT より遅延を大きくしないと言う条件下では，論理ブロック段数比 0.75，DW-DC 遅延分配係数 R_d が 0.1 の時，表 5.11 から CDC 容量が 1,024 ビットまで搭載でき，このとき実装密度は表 5.10 から約 2.5 倍に引き揚げることが可能であることがわ

表 5.11: LUT 段数比毎の 4-LUT に対する遅延比（R_d=0.1）

CDC ビット数	論理ブロック段数比毎の遅延比				
BN （bit）	0.70	0.75	0.80	0.85	0.90
64 bit	**0.87**	**0.93**	**0.99**	1.05	1.12
128 bit	**0.87**	**0.94**	**1.00**	1.06	1.12
256 bit	**0.88**	**0.95**	1.01	1.07	1.13
512 bit	**0.90**	**0.96**	1.02	1.09	1.15
1,024 bit	**0.92**	**0.98**	1.05	1.11	1.18
2,048 bit	**0.95**	1.01	1.08	1.15	1.22
4,096 bit	**0.99**	1.06	1.13	1.20	1.27
8,192 bit	1.05	1.12	1.20	1.27	1.35

かった．ただし，構成データキャッシュ内の回路は LUT へ書き込んでから実行する必要がある．

5.6.2 LUT のクラスタ化の効果

次に，LUT におけるクラスタ化の効果について考察する．本方式の 6-LUT は，2-LUT × 3 個（モード（a））または 3-LUT × 2 個（モード（b））として使用することができる．一方，表 5.9 に示したように，最大入力数未満の信号線しか入力されない LUT が，ある一定の割合で存在する．例えば 6-LUT の場合，2 信号しか入らない LUT は約 9%あり，3 信号しか入らない LUT は約 7%ある．したがって，これらの LUT をクラスタ化することで，使用論理ブロック数を減少させることが可能である．

図 5.8 は，全評価回路を 3-LUT から 7-LUT と，MCMG-LUT にマッピングした正規化面積と実装効率を示している．MCMG-LUT への実装は，完全にクラスタ化できた場合である．また，クラスタ化の効果を見るために，本方式の論理ブロックは CDC を含まない．実装効率は式（5.10）から求めた．正規化面積は最小値で正規化した値である．その結果，本方式の論理ブロックは，正規化面積は最小値に近づき，実装効率も従来の 6-LUT より約 6%改善されている．

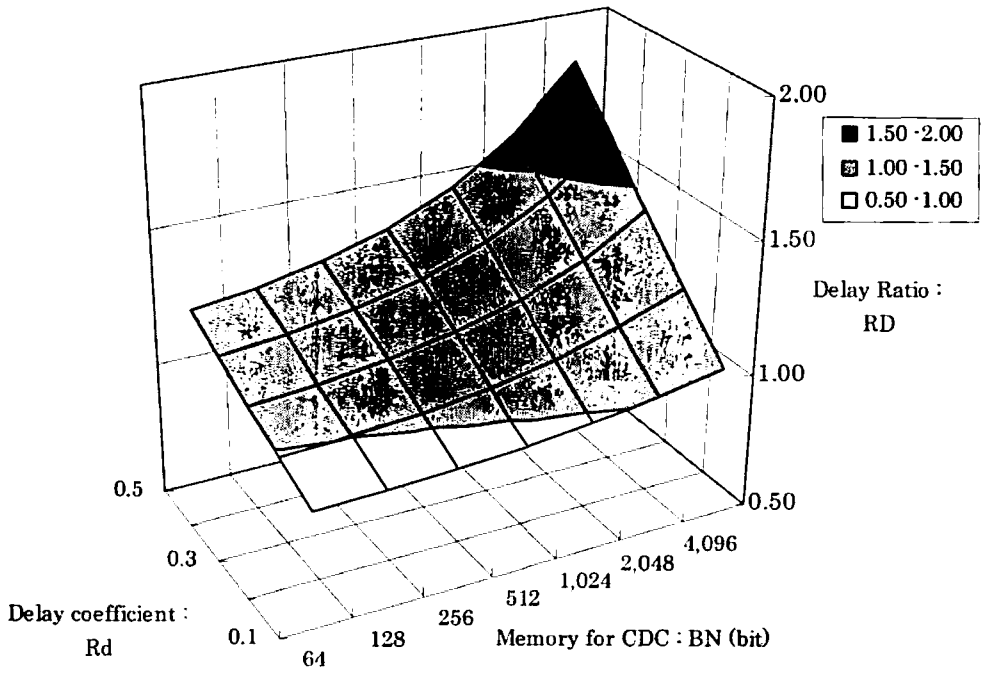

図 5.7: Rd による 4-LUT に対する遅延比の変化（N_6/N_4=0.75）

5.6.3 LUT のマルチコンテキスト化の効果

次に，マルチコンテキスト化の効果について考察する．

図 5.2 に示した本方式のマルチコンテキスト化するモード（d）～（f）を使用する場合，LUT の各コンテキストは同時に動作しないことが前提となるが，上記のマッピング方法では保証できない．そこで，ここでは図 5.9 に示すように，Verilog-HDL の RTL 記述レベルで，明示的に同時動作しないことが分っている case 文または if 文の各条件下の処理を，別のコンテキストにマッピングする方法を採用する．本方式へのマッピングは，オペレーション毎に図 5.6 にしたがって行われ，それらを手作業で統合することで最終的なネットリストを得る．

マルチコンテキスト化の評価では，図 5.10 に示した 16 ビット精度，8 オペレーションの ALU 回路（ALU16）を用いる．ここで，入力 A，B はデータ信号，入力 C はオペレーションを決めるコマンド信号，出力は Q である．

図 5.11 に，5.5.1 節で述べた 7 種類の LUT に実装した時の 4-LUT に対する面積比を示す．比較のために，本方式の論理ブロックは CDC を搭載しない．グラフの横軸は LUT

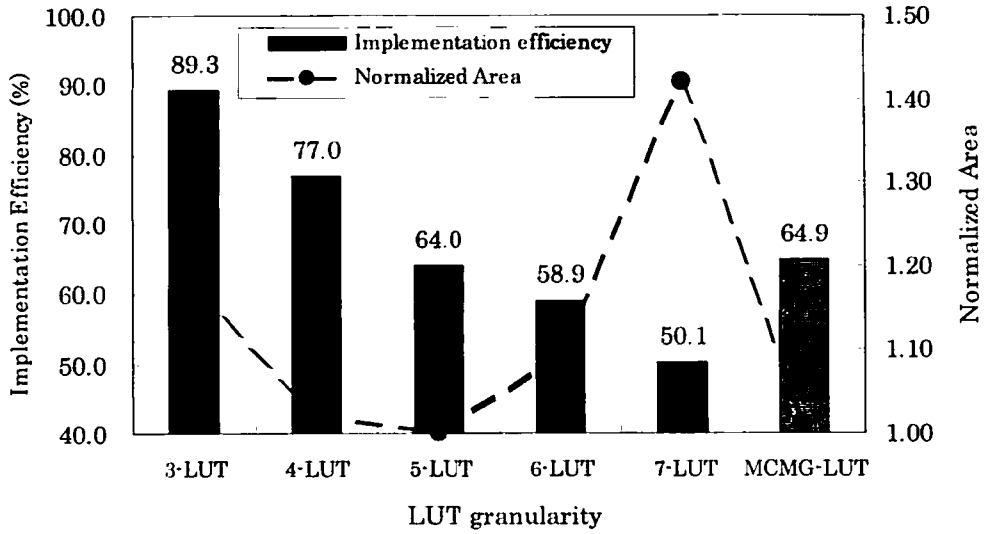

図 5.8: 評価回路における正規化面積と実装効率

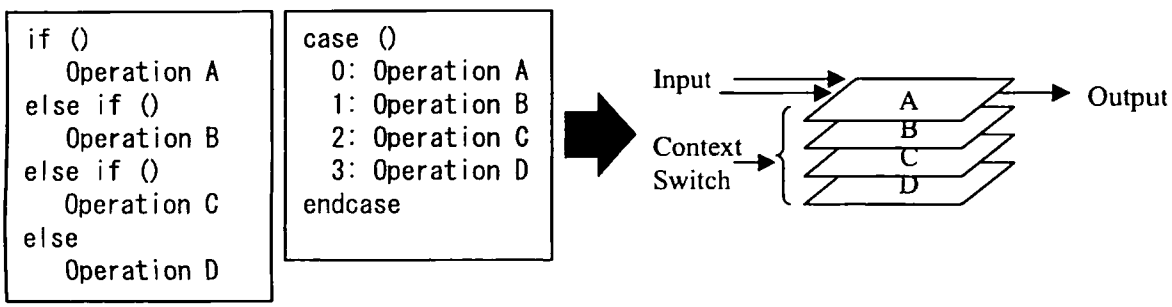

図 5.9: マルチコンテキストのマッピング方法

粒度を示し，縦軸に 4-LUT に対する面積比を示した．その結果，本方式の 3-LUT × 8（モード（d））に実装した場合が最小面積を示し，従来の 4-LUT と比べて約 87%で実装できることが分かる．

また，図 5.12 に 4-LUT の値で正規化した構成データ量を示す．モード（d）およびモード（e）共に，構成データ量は，従来の LUT 中で最小データビット数を示す 4-LUT と同等であることが分かる．

実装面積，構成データ共に削減できたのは，従来の 3-LUT から 7-LUT では出力段のセレクタが LUT を用いて構成されるのに対して，本方式では配線の切り替えによって行われたためと考えられる．

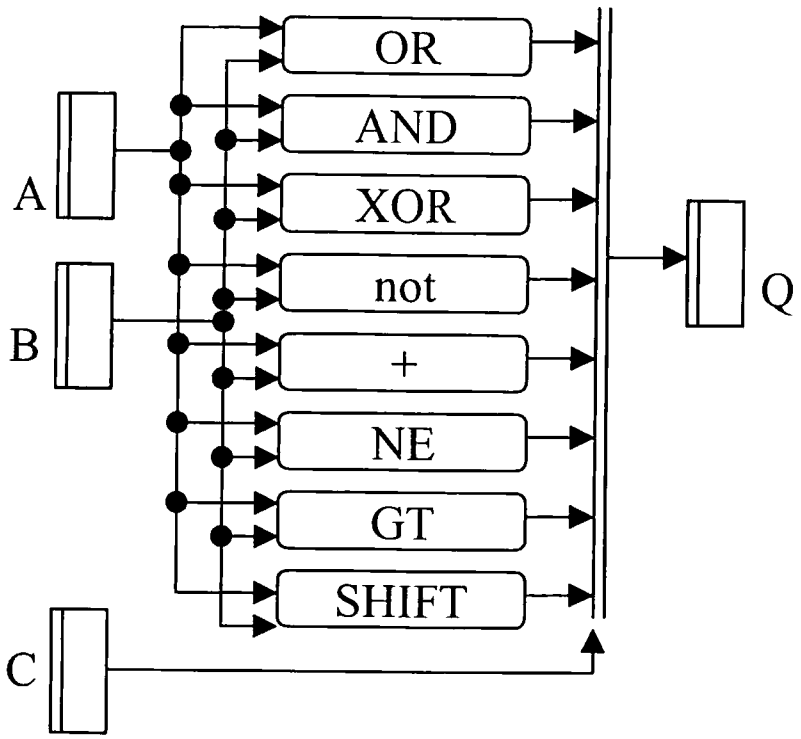

図 5.10: ALU16 回路の構成

5.6.4 関連研究

FPGA のマルチコンテキスト化の研究は広く行われている．MIT の DPGA (Dynamically Programmable Gate Arrays) [9] は，コンテキスト用メモリとして DRAM を持ち，LUT へロードすることで回路を切り替える．また，慶応大学の HOSMII[47] も DRAM をコンテキスト用メモリに使用している点で DPGA と近いが，コンテキストの切り替えをデータ駆動で行うという特徴がある．これらの方式は，LUT そのものは従来の FPGA と同じである．本提案方式では LUT の粒度も可変となり，より自由度が高い．

また，NEC の DRLE (Dynamically Reconfigurable Logic Engine) [10] は，論理ブロックを 4×4 の UC (Unified Cell) と呼ぶ統合セルに置き換え，LUT とクロスバスイッチとして働かせる．UC は 4×4 の MC (Memory Columns) からなり，各 MC を多ビット化することで複数のコンテキストを実装する．コンテキストの切り替えは RC (Reconfiguration Controller) からの制御信号によって行われる．DRLE は LUT が複数のモードを持つ点で本提案方式に近いが，コンテキストデータを LUT のメモリセル内に配置する点が異なる．

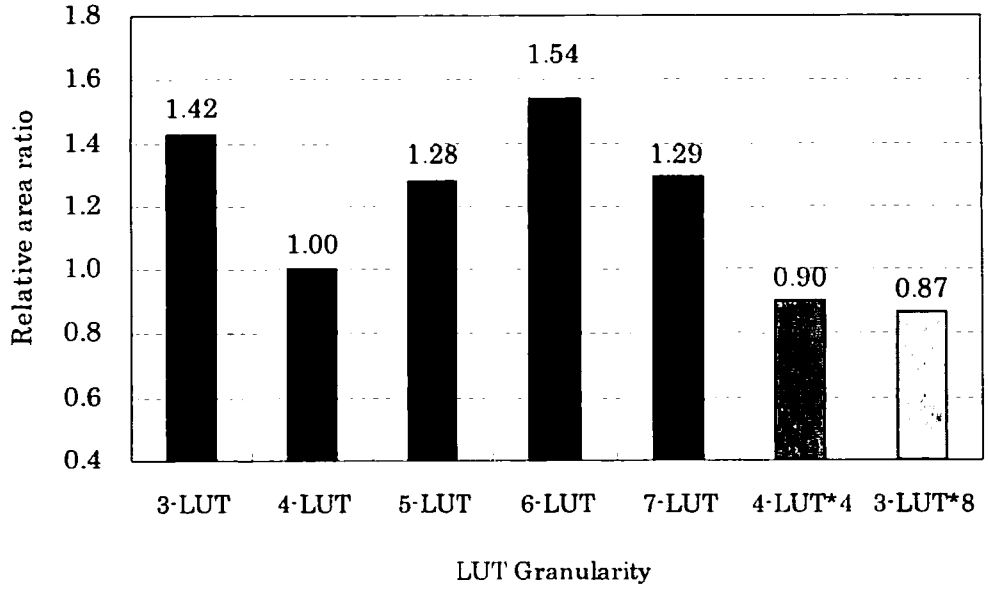

図 5.11: ALU16 の面積比較

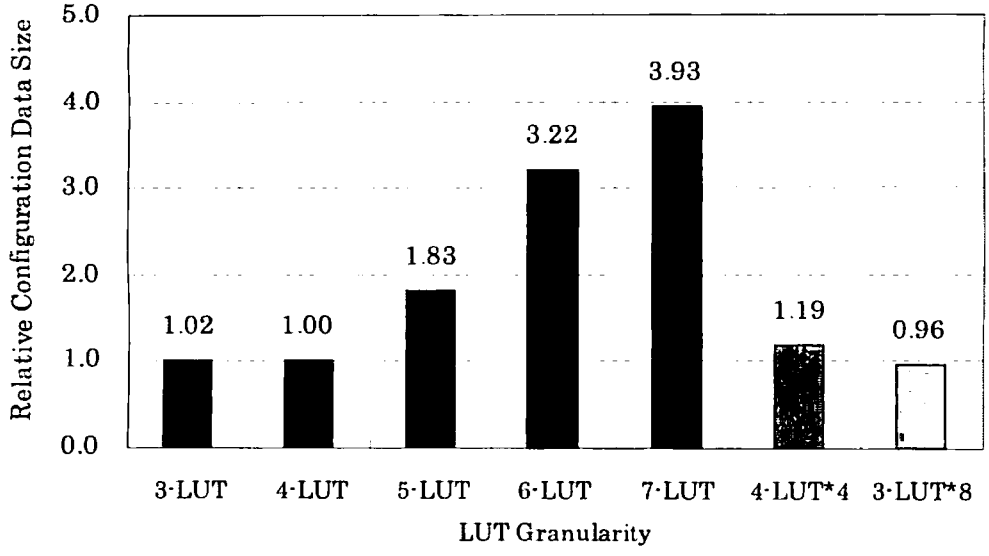

図 5.12: ALU16 の構成データ量比較

本方式では，CDC と LUT の 2 階層でコンテキストを管理する．

5.6.5 まとめ

本章では，LUT のマルチコンテキスト化とクラスタ化の両方を実現した MCMG-LUT と，論理ブロック内に複数のコンテキストを格納するための構成データキャッシュ（CDC）

を搭載したリコンフィギャラブル・コンピューティング向きの論理ブロックを提案した．そして，提案方式の各機能の効果を実装密度，実装面積，実装効率，および構成データ量について評価した．

その結果，*DW-DC* 遅延分配係数 $R_d = 0.1$，論理ブロック段数比 $N_6/N_4 = 0.75$，4-LUT に対して性能を落とさないという条件下で，搭載できる CDC の最大容量は 1,024 bits となった．この時，従来の FPGA（4-LUT）と比較して，実装密度を 2.52 倍に引き揚げることが分かった．ただし，CDC 内の回路は LUT へロードするまで実行できない．また，LUT のクラスタ化によって，実装効率は従来の 6-LUT に対して約 6%改善された．そして，マルチコンテキスト化によって，実装面積は最小値を示し，構成データ量は従来の LUT 中で最小の 4-LUT と同程度であることが分かった．

5.6.6 今後の課題

本章では，リコンフィギャラブル・コンピューティングに適したリコンフィギャラブル・ロジックデバイスを実現する論理ブロックを提案した．しかし，現在の CAE ツールは本提案論理ブロックには対応していない．したがって，今後は提案方式の論理ブロックを有効に活用できる CAE ツールを検討していく必要がある．また，構成データの入れ替え方法，配線構造などの詳細化も課題である．

第 6 章

結論

本研究の目的は，ノイマン型計算機を補完することを目的にしたリコンフィギャラブル・コンピューティングの在り方を明らかにし，新しいリコンフィギャラブル・コンピューティングのシステム構成とデバイス・アーキテクチャを提案することであった．これらの目的は本論文で示したようにほぼ達成できたと考える．以下に本研究の成果についてまとめる．

1. リコンフィギャラブル・コプロセッサ方式

   リコンフィギャラブル・コプロセッサ方式（プロセッサ，プログラマブル・ロジック混載方式）の目的である“プロセッサの稼動効率を高め性能を引き出すこと”を目標にして，オンチップ・マルチプロセッサの性能を向上させるために，プログラマブル・ロジックを搭載する方法を提案した．この複数のプロセッサにはスレッド単位で処理を割り当て，同期制御やコヒーレンス制御を行うスレッド制御ライブラリをハードウェアとソフトウェアの両方を用いて実装した．その結果，プロセッサ部とプログラマブル・ロジック部が同一クロック周波数で動作する場合，純粋にソフトウェアだけで実装した場合と比較して，約 17%～60%の性能改善が確認できた．また，プロセッサ部とプログラマブル・ロジック部が異なるクロック周波数である場合，そのクロック比が 10 倍以内であれば，性能が向上することを示した．また，回路規模に関しては実現可能な範囲であることを示した．

   以上が第 3 章で詳しく論じた第一の成果である．

2. アタッチド・プロセッサ方式

   汎用性とスケーラビリティを持つアタッチド・プロセッサ方式のリコンフィギャラブル・コンピューティングシステム RASH を開発し，計算集約的な処理に対して有効

であることを示した．特にリコンフィギャラブル・コンピューティングの特徴である“アルゴリズムとアーキテクチャの最適化”の効果を実験によって検証した．さらにアルゴリズムのハードウェア化によって，ノイマン・オーバヘッドが解消され，処理対象に内在する並列性を十分に引き出すことができた．DES 暗号の鍵探索処理では，Intel 社の Pentium プロセッサ（300MH z）に対して，ALTERA 社の FPGA である FLEX10K100A 単体（39.5MHz）で約 35 倍の性能を示し，マシンレベルでは 1,700 倍以上を示した．また，SAR 画像再生処理では，DSP と比較して，半分のデバイス数でほぼ同程度の性能が得られることが分かった．

以上が第 4 章で詳しく論じた第二の成果である．

3. リコンフィギャラブル・ロジックの提案

リコンフィギャラブル・ロジックの基本要素である LUT の粒度（論理規模）について実用的な評価回路を用いて面積評価，遅延時間評価および実装密度評価を行い，それに基づいて構成データキャッシュと，マルチコンテキスト化とクラスタ化の両方を実現した LUT とを持つリコンフィギャラブル・ロジックデバイス向き論理ブロックを提案した．そして，評価によって，提案方式の論理ブロックは，従来の論理ブロックより実装密度と実装効率が向上し，構成データ量の削減できることを示した．

以上が第 5 章で詳しく論じた第三の成果である．

さて，本研究のテーマであるリコンフィギャラブル・コンピューティングの研究に関して，今後の展望と課題についてまとめておきたい．

1. システムに関する展望と課題

本研究で開発した RASH は，汎用性と拡張性を重視したリコンフィギャラブル・コンピューティング・システムであるが，現在の FPGA を使う限りノイマン型のプロセッサほどの使い易さは無い．やはり，システムとしては計算集約的な処理を行う専用システムとして用いることが有望である．特に気象学や天文学，分子動力学などの計算科学の分野では大規模なシミュレーションが必要とされる．しかしながら，一般に並列計算機やクラスタ・コンピュータは，価格，占有スペース，消費電力，信頼性のどの側面から考えても専用システムには及ばない．一方，リコンフィギャラブル・コンピューティングシステムは，専用化できる汎用システムと考えることができ，このような用途での活用が期待できる．実際に東京大学の GRAPE プロジェクト [48] では，天文シミュレーション用の専用計算機として，リコンフィギャラブル・コンピューティング技術を利用して成果をあげている．また，小規模のシステムと

しては，プロセッサ・プログラマブルロジック混載の SoC の商品化が始まっている [49, 50]．これらの商品では情報家電や組み込みシステムのコスト削減に効果的であり，今後期待が持てる方式である．

課題としては，大規模なリコンフィギャラブル・コンピューティングシステムでは，並列計算機と同様にデータ入出力処理が性能向上面での一つの障害になる．小規模なシステムでは，ますますプロセッサとの共存が図られることが予想され，それに伴い動作クロックの違いをどう吸収するかが課題である．本論文でも SoC に適用した評価を述べたが，クロック差が 10 倍程度は許容できてもそれ以上に広がった場合にどうするか．また，今後，この差は開くのか縮まるのかの考察をしなければならない．

2. デバイスに関する展望と課題

第 5 章で述べたように，市場が求めている FPGA の能力とリコンフィギャラブル・コンピューティングが求めている能力とが異なるため，開発試作に使われている FPGA の流用ではなく，リコンフィギャラブル・コンピューティングに適したリコンフィギャラブル・ロジックの開発が必要である．本研究では細粒度アーキテクチャの提案を行ったが，粗粒度アーキテクチャの研究も盛んである．特に粗粒度アーキテクチャは，ディジタル信号処理分野での応用が有望であり，SoC 技術と組み合わせて新しい市場の開拓が期待できる．

また，SoC 時代の到来とは多品種小量生産の時代を迎えることを意味しており，少品種大量生産向きの集積回路製造技術は本質的矛盾を抱えることになる．それゆえに，半導体産業界においても “多品種小量となるシステム LSI 開発に少品種大量生産システムでどう乗り切るか” という根本命題に対する解決策の一つとして，この動的な柔軟性をもたらすリコンフィギャラブル・コンピューティングの特質に大きな期待が寄せられている．しかしながら，商業化を目指す上では，特許が障害になりつつある．プログラマブル・ロジックの基本特許は，諸外国に押さえられており，それを覆すためには，新しいデバイス構造や新素材を見つけ，その利用技術を含めた総合的な研究を進めなければならない．特に新素材としては，MRAM や FRAM のような不揮発性の素子を用いたリコンフィギャラブル・ロジックの研究が緊急の課題である．

3. 開発環境に関する展望と課題

2.3 節で述べたアプリケーション開発環境の課題は，未だ解消していない．新しい試みとして C/C++言語または Java を用いたハードウェア設計も行われており，これらがリコンフィギャラブル・コンピューティング向けのアプリケーション開発環境に

進展することを期待したい．

また，本研究ではリコンフィギャラブル・コンピューティング向けのアプリケーション開発環境については，ほとんど議論していない．これはデバイス構造を確定して上で研究を進めたかったためである．しかしながら，逆の発想でアルゴリズムとアーキテクチャを最適化するツールの構築を先に行い，それに即したデバイス構造を研究する手法も考えられる．このような研究手法の検討も，今後の検討課題である．

4. 応用に関する展望と課題

先にも述べたが，大規模なシステムでは計算科学的な応用に期待が持て，SoC のような小規模なシステムではディジタル信号処理への応用が有望である．さらに，新しい応用としてはリコンフィギャラブル・コンピューティングをさらに進展させた進化型システムの研究にも期待できる．進化型システムとは，自立的にハードウェア構成を変化させることで，常に状況に対して最適な性能を達成しようとするシステムである．すでに筋電制御型義手に利用されており，一定の評価を得ている [51]．

今後の応用研究では，効果的なアプリケーションのさらなる開拓と集積が益々重要になってくる．また，それらのアプリケーションの多角的な分析をシステム構成やデバイス・アーキテクチャへ反映させることが待望される．

リコンフィギャラブル・コンピューティングはまだ発展途上であり，実用的な技術としては未熟である．しかしながら，次世代の計算パラダイムとして，その期待に応えるだけの素養は備えていると考えている．近い将来にはソフトウェアとハードウェアという区別がなくなり，どちらもダウンロードされて実行されるだけの普通のオブジェクトと言われる時代が来るかもしれない．

謝辞

本研究をまとめるに際し，御指導，御鞭撻を賜りました熊本大学・末吉敏則教授に深甚なる謝意を表します．また，本論文をまとめるにあたり懇切なる御教示を賜りました熊本大学・中村良三教授，梅野英典教授，井上高宏教授に深く感謝の意を表します．研究の諸所で有益な御討論，数々の御助言を賜りました熊本大学・久我守弘助教授ならびに柴村英智助手に厚く御礼申し上げます．さらに，公私にわたり御世話頂くと共に，大きな御援助を頂きました福岡大学・奥村勝助教授に感謝致します．

また，三菱電機エンジニアリング（株）においては，御指導を賜った常務取締役・鎌倉事業所長・山田武氏，電子技術センタ長・秋田興一郎博士，技術開発課長・中澤俊夫氏，ＳＥ事業部東京マネージメントセンタ長・古川俊二氏，そして，戸根吉彦氏に感謝いたします．特に古川氏と戸根氏には，本研究を行う機会を得るために，暖かい御支援を賜りましたことを厚く御礼申し上げます．また，新留勝広氏をはじめとする技術開発課の諸兄からは，惜しみない御激励・御援助を頂きました．ここに感謝の意を表します．

本研究の重要な成果は，三菱電機（株）の宮田裕行博士，中島克人博士，森伯郎氏，佐藤裕幸氏，高橋勝己氏，浅見廣愛氏の多大なる御支援・御尽力なくしては成し遂げられませんでした．ここに深甚なる謝意を表します．特に，宮田博士には本研究を継続する端緒を戴き，中島博士には研究の諸所で有益な御討論，数々の御教示を賜りましたことを御礼申し上げます．

その他，本研究は，多数の方々の御指導，御協力のもとなされたものです．ここに謹んで御礼申し上げます．

最後に，著者を学問の道へ導いてくれた父 尋彦，母 伊代子，そして，研究を行う間，常に著者の健康に配慮し，また，心の支えとなってくれた妻 景子と，いつも笑顔で励ましてくれた息子 一穂に心より感謝いたします．

参考文献

[1] 星野力, “誰がどうやってコンピュータを創ったか？”, 共立出版, ISBN4-320-02742-6, 1995.

[2] Toshinori Sueyoshi and Masahiro Iida, “Configurable and Reconfigurable Computing for Digital Signal Processing”, IEICE Transactions on Fundamentals of Electronics,Communications and Computer Sciences Vol.85E-A, No.3, pp.591-599, 2002.

[3] 末吉敏則, 飯田全広, “リコンフィギャラブル・コンピューティング”, 情報処理 Vol.40, No.8, pp.777-782, 1999.

[4] 末吉敏則, “Reconfigurable Computing System の現状と課題-Computer Evolution へ向けて-”, 信学技報, Vol.96, No.426, pp.111-118, 1996.

[5] C.Thomas, “Field programmable gate array key to reconfigurable array outperforming supercomputers”, Proc. of the IEEE 1991 Custom Integrated Circuits Conf. (Cat. No.91CH2994-2), 6.6/1-4, 756, 1991.

[6] Annapolis Micro Systems, Inc., http://www. annapmicro.com/

[7] M. Wazlowski, L. Agarwal, T. Lee, A. Smith, E. Lam, P. Athanas, H. Silverman, and S. Ghosh, “PRISM-II compiler and architecture”, Proceedings IEEE Workshop on FPGAs for Custom Computing Machines (Cat. No.93TH0535-5), 9-16, ix+212, 1993.

[8] Michael J. Wirthlin and Brad L. Hutchings, “DISC : The dynamic instruction set computer”, Proc. SPIE - Int. Soc. Opt. Eng. (USA), vol.2607, pp.92–103, 1995.

[9] E. Tau, D. Chen, I. Eslick, J. Brown, and A. DeHon, “A First Generation DPGA implementation”, Proc. 3rd Canadian Workshop on Field-Programmable Devices, pp.138-143, 1995.

[10] T. Fujii, K. Furuta, M. Motomura, M. Nomura, M. Mizuno, K. Anjo, K. Wakabayashi, Y. Hirota, Y. Nakazawa, H. Ito, and M. Yamashina, “A Dynamically Reconfigurable Logic Engine with a Multi-Context / Multi-Mode Unified-Cell Architecture”, IEEE In-

ternational Solid-State Circuits Conference, WA 21.3, 1999.

[11] S.C. Goldstein, H. Schmit et al., "PipeRench: A Coprocessor for Streaming Multimedia Acceleration," in Proceedings, International Symposium on Computer Architecture, Atlanta, GA, pp.28–39, June 1999.

[12] A. Marshall, J. Vuillemin et al., "A Reconfigurable Arithmetic Array for Multimedia Applications," Proceedings of the 1999 ACM/SIGDA International Symposium on Field Programmable Gate Arrays, Monterey (CA), pp.135–143, Feb. 1999.

[13] VSI Alliance, http://www.vsi.org

[14] 飯田全広, 久我守弘, 末吉敏則, "スレッド制御回路を持つオンチップ・マルチプロセッサの構成", 情報処理学会論文誌 Vol.39, No.6, pp.1613-1621, 1998.

[15] 半導体技術ロードマップ専門委員会 (STRJ), "半導体技術動向に関する調査研究報告－半導体産業発展のための技術指針－", 社団法人電子情報技術産業協会, 1999.

[16] 天野英晴, "並列コンピュータ", 昭晃堂, ISBN4-7856-2045-5, 1996.

[17] Bil Lewis and Daniel J. Berg, 訳:岩本信一, "マルチスレッドプログラミング入門", アスキー出版, ISBN4-7561-1682-5, 1996.

[18] IEEE. Threads Extension for Portable Operating Systems (Draft 6), February 1992. P1003.4a/D6.

[19] Frank Mueller, "A Library Implementaion of POSIX Threads under UNIX", In Proceedings of the USENIX Conference, pages 29-41, January 1993.

[20] http://www.informatik.hu-berlin.de/~mueller/projects .html

[21] 飯田全広, 久我守弘, 末吉敏則, "マルチスレッド制御ライブラリのハードウェア化によるオンチップ・マルチプロセッサの構成", 並列処理シンポジウム JSPP'97 論文集, pp.337-344, 1997.

[22] J.L. Hennessy and D.A. Patterson, "Computer Architecture : A Quantitative Approach", Morgan Kaufmann Publishers, Inc., 1990.

[23] 中垣憲一, 井上弘士, 久我守弘, 末吉敏則, "上級コース向き教育用マイクロプロセッサ DLX–FPGA の設計と実装", 信学技報 CPSY94-57, 1994.

[24] ftp://max.stanford.edu/pub/hennessy-patterson.software/dlx.tar.Z

[25] Xilinx, Inc., "The Programmable Logic Data Book", 1996.

[26] Xilinx, Inc., "Virtex-II Platform FPGA Handbook", 2000.

[27] Katsuto Nakajima, Hiroyuki Sato, Hiroai Asami, Masahiro Iida, Katsuhiro Shindome,

Hakuro Mori, Katsumi Takahashi, and Yusuke Mizukami, "FPGA-based Parallel Machine : RASH", Proceedings of AI2000 (Applied Infomatics 2000), pp.269-273, 2000.

[28] 浅見廣愛, 飯田全広, 中島克人, 森伯郎, "FPGA ベース並列マシン RASH での DES 暗号解析処理の改良", 情報処理学会論文誌 ハイパフォーマンスコンピューティングシステム Vol.41, No.SIG 5(HPS 1), pp.50-57, 2000.

[29] Katsumi Takahashi, Masahiro Iida, and Katsuto Nakajima, "Time-Memory Trade-Off Cryptanalysis on FPGA-based Parallel Machine RASH", Proceedings of HPC-Asia2000 (The Fourth International Conference / Exhinbition on High Performance Computing in Asia Pacific Reagion), pp.366-369, 2000.

[30] Hiroai Asami, Masaharu Mizuno, Katsuto Nakajima, Masahiro Iida, and Hakuro Mori, "Application of SAR Image Reconstruction Processing on an FPGA-based Parallel Machine RASH", AI2002 (Applied Informatics 2002) , pp.65-70, 2002.

[31] 飯田全広, 水上雄介, 高橋勝己, 浅見廣愛, 佐藤裕幸, "FPGA による並列暗号解析装置の構成 (1) － DES 暗号等の鍵探索－", 第 58 回情処全国大会, 5N-08, 1999.

[32] I.Hamer and P.Chow, "DES Cracking on the Transmogrifier 2a", *CHES'99 (CHES: Workshop on Cryptographic Hardware and Embedded Systems)*, pp.13–24, 1999.

[33] B. Schneier and D. Whiting, "Fast Soft Encryption: Designing Encryption Algorithms for Optimal Software Speed on the Intel Pentium Processor", *Proceedings of 4th International Workshop FSE97*, Lecture Notes In Computer Science 1267, Springer Verlag, pp.242–259, 1997.

[34] E. Biham, "Fast Software Encryption", *4th International Workshop, FSE'97 Proceedings*, 1997.

[35] 高橋勝己, 飯田全広, 水上雄介, 山崎弘巳, 宮田裕行, 中島克人, 松本勉, "タイムメモリトレードオフ解読法に基づく暗号強度評価装置の実現性について", 情報処理学会論文誌, Vol.40, No.8, pp.3318–3328, 1999.

[36] 藤坂貴彦, 岩本雅史, 原芳久, 江馬浩一, "画像レーダ", 非破壊検査, vol.47, no.12, pp872-877, 1998.

[37] J.C. Curlander and R.N. McDonough, "SYNTHETIC APERTURE RADAR systems and Signal Processing", John Wiley Sons Inc., 1991.

[38] 水野政治, 浅見廣愛, 飯田全広, 中島克人, 森伯郎, "FPGA ベース並列マシン RASH の SAR 画像再生処理への適用検討 (2) －市販 DSP システムとの比較検討－", 第 59 回情処全国大会, 5H-04, 1999.

[39] Masahiro Iida and Toshinori Sueyoshi, "A Novel Programmable Logic Architecture for Reconfigurable Computing", PYIWIT02 (Pan-Yellow-Sea International Workshop on Information Technologies for Network Era), 2002.

[40] J.S. Rose, R. Francis, and D. Lewis, "Architecture of Field-Programmable Gate Arrays : The Effect of Logic Block Function on Area Efficiency", IEEE J.Solid-State Circuits, Vol.25, No.5, pp.1217-1225, 1992.

[41] J.S. Rose and S. Brown, "The Effect of Logic Block Architecture on FPGA Performance", IEEE J.Solid-State Circuits, Vol.27, No.3, pp.281-287, 1992.

[42] Paul Leventis, "Using edif2blif Version 1.0 (Draft)", June 30, 1998. http://www.eecg.toronto.edu/~jayar/software/software.html

[43] E.M. Sentovich et al., "SIS:A System for Sequential Circuit Synthsis", Memorandom No. UCB / ERL M92/41, 1992.
http://www-cad.eecs.berkeley.edu:80/Software/software.html

[44] A. Marquardt, et al., "Using Cluster-Based Logic Blocks and Timing-Driven Packing to Improve FPGA Speed and Density", FPGA 99, 1999.
http://www.eecg.toronto.edu/~jayar/software/software.html

[45] S. Yang, "Logic Synthesis and Optimization Benchmarks, Version 3.0", Tech. Report, Microelectronics Centre of North Carolina, 1991.

[46] Altera Corporation, "APEX20K Programmable Logic Device Family Data Sheet", 1999.

[47] 柴田裕一郎, 宮崎英倫, 凌暁萍, 天野英晴, "HOSMII : DRAM 混載型 FPGA に基づく仮想ハードウェアシステム", 信学技報 CPSY97-45, 1997.

[48] 牧野淳一郎, "次世代超並列計算機開発 - 多粒子系向け超並列計算機の開発", 計算科学 1999-2000 報告書,pp.21-32, 日本学術振興会, 2000.

[49] Tensilica Inc., "Tensilica: Technology",
http://www.tensilica.com/technology.html

[50] Triscend Corporation, "Configurable System-on-Chip",
http://www.triscend.com/

[51] 樋口哲也, "進化型ハードウェア", 情報処理 Vol.40, No.8, pp.795-800, 1999.

付録 A

Survey : Reconfigurable Computing Systems and Devices

ここでは 2.2 章で紹介した開発事例含めて，これまでに研究開発されてきたリコンフィギャラブル・コンピューティングシステムおよびリコンフィギャラブル・コンピューティング向きのデバイスについてまとめる．

まず，リコンフィギャラブル・コンピューティングシステムとしては，アタッチド・プロセッサ方式のシステムを表 A.1 から表 A.6 までに示し，リコンフィギャラブル・コプロセッサ方式を表 A.7 から表 A.8 に示す．そして，表 A.9 にリコンフィギャラブル・プロセッサ方式のシステムをまとめる．

一方，デバイスの研究開発事例は，表 A.10 から表 A.11 では細粒度アーキテクチャのデバイスを示し，表 A.12 から表 A.14 で粗粒度アーキテクチャのデバイスについてまとめる．

表 A.1: アタッチド・プロセッサ方式の代表的な研究開発システム（その 1）

名称		GANGLION
開発元		IBM
諸元	FPGA（個数）	24 XC3090s
	On-board RAM	24KB PROM
	Interconnect	Fixed
	External bus	VME / Datacube MAXbus
用途 / 応用		ニューラルネットワーク
備考		
関連文献		Cox, C.E., Blanz, W.E., "GANGLION-a fast hardware implementation of a connectionist classifier", Proc. IEEE 1991 Custom Integrated Circuits Conf., p.756, 1991.
名称		SPLASH
開発元		Supercomputing Research Center
諸元	FPGA（個数）	32 XC3090s
	On-board RAM	4MB SRAM
	Interconnect	Linear array
	External bus	VME
用途 / 主な応用		汎用エンジン / DNA パターンマッチング
備考		
関連文献		Thomas C. Waugh, "Field programmable gate array key to reconfigurable array outperforming supercomputers", Proc. IEEE Custom Integrated Circuits Conf., p.756, 1991.
名称		SPLASH 2
開発元		Supercomputing Research Center
諸元	FPGA（個数）	16 XC4010s
	On-board RAM	8MB
	Interconnect	Linear array および crossbar
	External bus	SBus
用途 / 主な応用		汎用エンジン / DNA パターンマッチング，キーワード検索，辞書検索
備考		SPLASH の改良版
関連文献		Arnold, J. M., Buell, D. A., Davis, E. G., "Splash 2 (attached processor board)", SPAA '92. 4th Ann. ACM Symp. Parallel Algorithms and Architectures, pp.316–322, 1992.
名称		Anyboard
開発元		North Carolina State University
諸元	FPGA（個数）	5 XC3042s
	On-board RAM	384KB
	Interconnect	Fixed buses
	External bus	ISA
用途 / 主な応用		プロトタイピング
備考		
関連文献		Van den Bout, D.E., Morris, J.N., Thomae, D., Labrozzi, S., Wingo, S., Hallman, D., "AnyBoard: an FPGA-based, reconfigurable system", IEEE Des. Test Comput. (USA), vol.9, no.3, pp.21–30, Sept. 1992.

表 A.2: アタッチド・プロセッサ方式の代表的な研究開発システム（その２）

名称		ArMen
開発元		Universite de Bretagne Occidentale (France)
諸元	FPGA（個数）	1 XC3090 per node
	On-board RAM	1, 2 or 4Mb/node each board
	Interconnect	cube
	External bus	SBus
用途 / 主な応用		トランスピュータによる MIMD マシン
備考		
関連文献		Raimbault, F., Lavenier, D., Rubini, S., Pottier, B., "Fine grain parallelism on a MIMD machine using FPGAs", Proc. IEEE Workshop on FPGAs for Custom Computing Machines, pp.2–8, 1993.
名称		Rasa
開発元		Carnegie Mellon University
諸元	FPGA（個数）	3 XC4010
	On-board RAM	320K SRAM
	Interconnect	2 Aptix FPICs
	External bus	ISA
用途 / 主な応用		プロトタイピング
備考		
関連文献		Schmit, H., Amstein, L., Thomas, D., Lagnese, E., "Behavioral synthesis for FPGA-based computing", Proc. IEEE Workshop on FPGAs for Custom Computing Machines, pp.125–32, 1994.
名称		CHAMP
開発元		Lockheed Sanders
諸元	FPGA（個数）	16 XC4013s
	On-board RAM	512KB Dual-ported
	Interconnect	Crossbar (using FPGAs)
	External bus	VME
用途 / 主な応用		ハードウエアプロトタイプ / イメージプロセッシング
備考		
関連文献		Box, B. "Field Programmable Gate Array Based Reconfigurable Preprocessor", Proc. IEEE 1994 National Aerospace and Electronics Conf. NAECON 1994, vol.1, pp.427–434, 1994.
名称		DFFC
開発元		University of montreal (Canada)
諸元	FPGA（個数）	512 custom-built FPOAs
	On-board RAM	4MB×2
	Interconnect	3D 8x8x8 array
	External bus	VME
用途 / 主な応用		データフローコンピュータ
備考		
関連文献		"A Reconfigurable Compute Engine for Real-Time Vision Automata Prototyping", IEEE Workshop on FPGAs for Custom Computing Machines, pp.91–100, 1994.

表 A.3: アタッチド・プロセッサ方式の代表的な研究開発システム（その3）

名称		BORG
開発元		University of California Santa Cruz
諸元	FPGA（個数）	2 XC3030s and 2 XC3042s
	On-board RAM	2KB
	Interconnect	Clos network 2 FPGAs can be used as interconnect or logic
	External bus	PC-bus interface in 5th FPGA
用途 / 主な応用		プロトタイピング
備考		
関連文献		Strandberg, R.H., Le Duc, J.C., Zsu-Yao Yang, Bustamante, L.G., Oklobdzija, V.G., Soderstrand, M.A., "Reconfigurable processor for real-time adaptive sample rate notch filtering", Conf. Record of the Twenty-Eighth Asilomar Conf. Signals, Systems and Computers, vol.2, pp.1497–1500, 1994.
名称		BORG-II
開発元		University of California Santa Cruz
諸元	FPGA（個数）	2 XC4003As and 2 XC4002As
	On-board RAM	8KB
	Interconnect	4 FPGAs in a Clos network 2 FPGAs can be used as interconnect or logic
	External bus	PC-bus interface in 5th FPGA
用途 / 主な応用		プロトタイピング
備考		
関連文献		Strandberg, R.H., Le Duc, J.C., Zsu-Yao Yang, Bustamante, L.G., Oklobdzija, V.G., Soderstrand, M.A., "Reconfigurable processor for real-time adaptive sample rate notch filtering", Conf. Record of the Twenty-Eighth Asilomar Conf. Signals, Systems and Computers, vol.2, pp.1497–1500, 1994.
名称		RM-II
開発元		神戸大
諸元	FPGA（個数）	10 XC4005s
	On-board RAM	768KB
	Interconnect	クロスバス・スイッチ
	External bus	不明
用途 / 主な応用		汎用エンジン / 論理エミュレーション
備考		
関連文献		沼 昌宏, "FPGA を利用したアーキテクチャとシステム設計", 情報処理, Vol.35, No.6, pp.511–518, Jun. 1994.
名称		perle-1
開発元		DEC
諸元	FPGA（個数）	24 XC3090s
	On-board RAM	4MB SRAM
	Interconnect	Fixed mesh
	External bus	DEC TURBO channel
用途 / 主な応用		汎用エンジン
備考		
関連文献		Digital Equipment Corporation., http://www.research.digital.com/SRC/pamette/

表 A.4: アタッチド・プロセッサ方式の代表的な研究開発システム（その 4）

名称		RPM
開発元		University of Southern California Los Angeles
諸元	FPGA（個数）	7 XC4013s
	On-board RAM	2MB SRAM(1st), 8MB SRAM(2nd), 96MB DRAM (3rd)
	Interconnect	不明
	External bus	SCSI backplane FutureBus+
用途 / 主な応用		マルチプロセッサ・プロトタイピング
備考		
関連文献		Oner, K., Barroso, L.A., Iman, S., Jeong, J., Ramamurthy, K., Dubois, M., "The design of RPM: an FPGA-based multiprocessor emulator", FPGA '95. 1995 ACM Third Int. Symp. Field-Programmable Gate Arrays, pp.60–66, 1995.
名称		TERAMAC
開発元		Hewlett-Packard
諸元	FPGA（個数）	108 custom FPGAs
	On-board RAM	32MB SRAM / board
	Interconnect	Hardwired design independent 27 chip, 39 layer MCM
	External bus	SCSI
用途 / 主な応用		汎用エンジン / 論理エミュレーション
備考		
関連文献		Amerson, R., Carter, R.J., Culbertson, W.B., Kuekes, P., Snider, G., "Teramac-configurable custom computing", Proc. IEEE Symp. FPGAs for Custom Computing Machines, pp.32–38, 1995.
名称		YARDS
開発元		NTT
諸元	FPGA（個数）	XC4010 and XC3030, Daughter card:XC4010 x4, or MAX9000 x2, or PROTEUS(custom) x2
	On-board RAM	Dual-Port SRAM 32KB x4
	Interconnect	I-CUBE IQ320 x2
	External bus	VME
用途 / 主な応用		伝送処理
備考		
関連文献		筒井章博, 宮崎敏明, "YARDS:ハードウェア・ソフト融合型通信処理システム", 信学会総合大会, March 1996.
名称		RACER
開発元		University of Toronto (Canada)
諸元	FPGA（個数）	PS Block(2 XC4010s) x14, CNT Block(1 XC4010) x1, RAM Block(1 XC4010) x7
	On-board RAM	RAM Block(SRAM 32Kx8 x2) x7
	Interconnect	Ring
	External bus	SBus x2
用途 / 主な応用		マルチプロセッサ / ビタビ・デコーダ
備考		
関連文献		Yeh, D., Feygin, G., and Chow, P., "RACER : A Reconfigurable Constraint-Length 14 Viterbi Decoder," Proc. of IEEE Workshop on FPGAs for Custom Computing Machines, Napa, Ca, Apr. 1996.

表 A.5: アタッチド・プロセッサ方式の代表的な研究開発システム（その５）

名称		HAB2
開発元		電総研，東京大
諸元	FPGA（個数）	10 XC4085s(8 Nodes, Crossbar, Video I/O), 1 XC4025(SCSI Cont.)
	On-board RAM	SRAM 4MB x8, SRAM 6.5MB
	Interconnect	Fixed
	External bus	SCSI x1, Video I/O x2
用途 / 主な応用		アルゴリズム検証用テストベット / 動画像処理
備考		
関連文献		関山守, 野村真義, 平木敬, "Hardware での動画処理における動的再構成の有意性について", 情報処理学会報告 98-ARC-130-23, 1998.
名称		RASH
開発元		三菱電機
諸元	FPGA（個数）	8 EPF10K100As
	On-board RAM	SRAM 2MB
	Interconnect	mesh, ring, local bus
	External bus	PCI
用途 / 主な応用		汎用エンジン / 暗号解析，SAR 画像処理
備考		
関連文献		Nakajima, K., Sato, H., Asami, H., Iida, M., Shindome, K., Mori, H., Takahashi, K., Mizukami, Y., "FPGA-based Parallel Machine : RASH", Proceedings of AI2000 (Applied Informatics 2000), 2000.

表 A.6: リコンフィギャラブル・コプロセッサ方式の代表的な研究開発システム（その 1）

名称		PRISM
開発元		Brown University
諸元	FPGA（個数）	4 XC3090s
	On-board RAM	None
	Interconnect	None
	External bus	16 bit
用途 / 主な応用		コプロセッサ
備考		
関連文献		Athanas, P., and Silverman, H., "Processor reconfiguration through instruction-set metamorphosis : Architecture and compiler", IEEE Computer, March 1993.
名称		PRISM-II
開発元		Brown University
諸元	FPGA（個数）	3 XC4010s
	On-board RAM	128KB×32 / XC4010
	Interconnect	Inverted tree, or none, application selectable
	External bus	64 bit writes, 32 bit reads, on processor bus
用途 / 主な応用		コプロセッサ
備考		C 言語のプログラムからハードウェアを合成.
関連文献		Wazlowski, M., Agarwal, L., Lee, T., Smith, A., Lam, E., Athanas, P., Silverman, H., Ghosh, S., "PRISM-II compiler and architecture", Proc. IEEE Workshop on FPGAs for Custom Computing Machines, pp.9–16, 1993.
名称		RECON
開発元		University of Melbourne (Australia)
諸元	FPGA（個数）	XC4010, XC2064
	On-board RAM	32K
	Interconnect	不明
	External bus	SBus
用途 / 主な応用		コプロセッサ
備考		
関連文献		Wo, D., Forward, K., "Compiling to the gate Level for Reconfigurable Co-Processor", Proc. IEEE Workshop on FPGAs for Custom Computing Machines, pp.147–154, 1994.
名称		TbC-Pamette
開発元		DEC
諸元	FPGA（個数）	4 XC4010s, 1 XC4003H
	On-board RAM	ドータボード
	Interconnect	Fixed mesh 2×2 matrix
	External bus	DEC TURBO channel
用途 / 主な応用		I/O 指向コプロセッサ
備考		
関連文献		Digital Equipment Corporation., http://www.research.digital.com/SRC/pamette/

表 A.7: リコンフィギャラブル・コプロセッサ方式の代表的な研究開発システム（その２）

名称		PCI Pamette V1
開発元		DEC
諸元	FPGA（個数）	4 XC4010Es
	On-board RAM	Two 64k x 16-bitsSRAM banks, 4 angled 72-pin SIMM DRAM connectors
	Interconnect	2x2 matrix of PQ208 footprints
	External bus	PCI
用途 / 主な応用		I/O 指向コプロセッサ
備考		
関連文献		Digital Equipment Corporation., http://www.research.digital.com/SRC/pamette/
名称		GPCP-SS
開発元		奈良先端大
諸元	FPGA（個数）	4 EPF81188ARC240-2
	On-board RAM	128KB
	Interconnect	不明
	External bus	SBus
用途 / 主な応用		コプロセッサ
備考		
関連文献		伊藤 康史, 平尾 誠, 木村 晋二, 渡邉 勝正, "汎用コプロセッサ GPCP-SS の実現と評価", 信学技報, VLD95-100,FTS95-62(1995-10), 1995.
名称		HARP1
開発元		Oxford University (UK)
諸元	FPGA（個数）	1 XC3195
	On-board RAM	64KB SRAM / 4MB DRAM
	Interconnect	FPGA shares RISC processor bus
	External bus	4×20Mbit / sec transputer links + expansion port (spare FPGA pins)
用途 / 主な応用		コプロセッサ
備考		
関連文献		Oxford University, http://www.comlab.ox.ac.uk/oucl/ users/ian.page/harp/harp1.html Page, I., "Reconfigurable Processor", An invited keynote address for the Heathrow PLD Conference, April 1995
名称		OPERA
開発元		名古屋大
諸元	FPGA（個数）	2 ORCA OR2C
	On-board RAM	SRAM
	Interconnect	Fixed
	External bus	PCI
用途 / 主な応用		コプロセッサ
備考		
関連文献		市川周一, 島田俊夫, "PCI バスに付加する再構成可能ボードの試作評価", 信学技報 VLD96-85, CPSY96-97(1996-12), 1996.

表 A.8: リコンフィギャラブル・コプロセッサ方式の代表的な研究開発システム（その３）

名称		MUGE-board
開発元		電通大
諸元	FPGA（個数）	3 XC4010Es
	On-board RAM	SRAM 128KB
	Interconnect	Local Bus
	External bus	SBus
用途 / 主な応用		コプロセッサ / 圧縮伸張
備考		
関連文献		齊藤正伸, 村松寛文, 三浦誠, 中村嘉志, 多田好克, "再構成可能型計算機 MUGE-board を用いたシステムの評価", 信学技報 CPSY98-53(1998.8), 1998.

表 A.9: リコンフィギャラブル・プロセッサ方式の代表的な研究開発システム

名称		PRISC-1
開発元		Cambridge University (UK)
諸元	FPGA（個数）	Custom-built PFU
	On-board RAM	不明
	Interconnect	3 Local Buses
	External bus	不明
用途 / 主な応用		リコンフィギャラブル・プロセッサ
備考		
関連文献		Razdan, R., Brace, K., Smith, D., "A High-Performance Microarchitecture with Hardware-Programmable Functional Units", Proc. of MICRO-27, November 1994.
名称		DISC
開発元		Brigham Young University
諸元	FPGA（個数）	2 CLAy31s
	On-board RAM	不明
	Interconnect	不明
	External bus	ISA
用途 / 主な応用		リコンフィギャラブル・プロセッサ
備考		
関連文献		Michael J. Wirthlin, Brad L. Hutchings, "DISC : The dynamic instruction set computer", Proc. SPIE-Int. Soc. Opt. Eng. (USA), vol.2607, pp.92–103, 1995.

表 A.10: 細粒度アーキテクチャの代表的なデバイス（その 1）

名称		DPGA
開発元		Massachusetts Institute of Technology
諸元	論理ブロック	Context Memory + 4-LUT
	配線構造	4x4 SubArray, 3x3 Composition of SubArrays with Programmable CrossBar
	再構成方式	非アクティブなコンテキストへの書き込み.
	内部メモリ	DRAM 32x4 bit per Array Element
備考		
関連文献		Tau, E., Chen, D., Eslick, I., Brown, J., and DeHon, A., "A First Generation DPGA implementation", Proc. 3rd Canadian Workshop on Field-Programmable Devices, pp.138-143, 1995.
名称		CAM-based FPGA
開発元		Oxford University (UK)
諸元	論理ブロック	CAM-based cell
	配線構造	2D mesh
	再構成方式	不明
	内部メモリ	SRAM/CAM 16 bits per cell (using CAM-based cell)
備考		CAM-base cell を LUT モードと PLA モード，SRAM，CAM，ルーティング・スイッチの 5 種類で使用可能.
関連文献		Stansfield, A., Page, I., "The Design of a New FPGA Architecture", Proc. of FPL95 (LNCS), 1995.
名称		Garp
開発元		University of California Berkeley
諸元	論理ブロック	4-LUT x2
	配線構造	不明
	再構成方式	部分再構成
	内部メモリ	不明
備考		MIPS-II プロセッサ，キャッシュメモリ等と混載
関連文献		Hauser, J.R., Wawrzynek, J., "Garp: A MIPS Processor with a Reconfigurable Coprocessor", Proc. IEEE Symposium on FPGA for Custom Computing Machines, 1997.
名称		PCA
開発元		NTT
諸元	論理ブロック	可変部（4-LUT x4）, 組み込み部（Fixed Logic）
	配線構造	2D mesh
	再構成方式	動的再構成，部分再構成
	内部メモリ	不明
備考		オブジェクト間を非同期のメッセージ通信で制御.
関連文献		伊藤秀之, 小栗清, 永見康一, 小西隆介, 塩澤恒道, "動的再構成可能計算機構のためのプラスティックセルアーキテクチャ", 情報処理学会報告 98-ARC-129-6, 1998.

表 A.11: 細粒度アーキテクチャの代表的なデバイス（その２）

名称		HSRA
開発元		University of Califomia Berkeley
諸元	論理ブロック	Conventional 4-LUT
	配線構造	Full 2-ary hierarchical array with shortcut connections
	再構成方式	不明
	内部メモリ	不明
備考		0.4μm DRAM process. 250MHz operation.
関連文献		Tsu, W., Macy, K., Joshi, A., Huang, R., Walker, N., Tung, T., Rowhani, O., George, V., Wawrzynek, J., DeHon, A., "HSRA: High-Speed, Hierarchical Synchronous Reconfigurable Array", Proc. ACM/SIGDA International Symposium on Field Programmable Gate Arrays. pp.69–78, Feb 1999.
名称		DRLE
開発元		NEC
諸元	論理ブロック	4x4 UC(4x4 4 input multi-contex LUT), RC
	配線構造	2D mesh
	再構成方式	マルチコンテキスト
	内部メモリ	不明
備考		0.25μm CMOS, 2.5v, 動作速度:70MHz. 再構成時間:4.6ns.
関連文献		Fujii, T., Furuta, K., Motomura, M., Nomura, M., Mizuno, M., Anjo, K., Wakabayashi, K., Hirota, Y., Nakazawa, Y., Ito, H., and Yamashina, M., "A Dynamically Reconfigurable Logic Engine with a Multi-Context / Multi-Mode Unified-Cell Architecture", IEEE International Solid-State Circuits Conference, WA 21.3, 1999.
名称		HOSMII
開発元		慶応大
諸元	論理ブロック	3-LUT x3 with Contex Memory
	配線構造	不明
	再構成方式	マルチコンテキスト
	内部メモリ	DRAM 64x16 bits Contex Memory per Logic block
備考		0.35μm Metal3-Poly2, 3.3v, 動作速度:100MHz. 再構成時間:5.0ns. データフロー制御.
関連文献		川上大輔, 柴田裕一郎, 天野英晴, "仮想ハードウェア HOSMII のための DRAM 型マルチコンテキスト FPGA の設計", 情報処理学会報告 SLDM-99-1, 2001.
名称		MCMG-LUT
開発元		熊本大
諸元	論理ブロック	Multi-Context, Multi-Grained LUT
	配線構造	2D mesh
	再構成方式	部分再構成，マルチコンテキスト
	内部メモリ	構成データ用キャッシュ
備考		
関連文献		飯田全広, 末吉敏則, "リコンフィギャラブル・ロジック向き論理ブロックの提案", 信学技報 CPSY2001-79(2001-11), 2001.

表 A.12: 粗粒度アーキテクチャの代表的なデバイス（その 1）

名称		rDPA
開発元		Kaiserslauterm University (Germany)
諸元	論理ブロック	ALU
	配線構造	2D mesh
	再構成方式	不明
	内部メモリ	Control Memory
備考		'As soon as possible' schedule (ASAP), Scheduling Policy : 'As late as possible' shedule (ALAP)
関連文献		Hartenstein, R., Kress, R., Reinig, H., "A New FPGA Architecture for Word-Oriented Datapaths", Field-Programmable Logic: Architectures, Synthesis and Applications, 4th International Workshop on Field-Programmable Logic and Applications, pp.144–155, 1994.
名称		Pleiades
開発元		University of California Berkeley
諸元	論理ブロック	EXU (ALU + booth multiply)
	配線構造	2D mesh, Crossbar
	再構成方式	不明
	内部メモリ	不明
備考		
関連文献		Rabaey, J.M., "Reconfigurable Computing: The Solution to Low Power Programmable DSP", Proc. of 1997 ICASSP Conf., Munich, April 1997.
名称		FPDF
開発元		東北大
諸元	論理ブロック	Configurale Arithmetic Block (CAB) : Signed-Weight 数 4-2 カウンタ
	配線構造	2D mesh
	再構成方式	不明
	内部メモリ	
備考		ディジタルフィルタ用 FPGA.
関連文献		澤田善樹, 青木孝文, 樋口龍雄, "新しい冗長数表現に基づくディジタル信号処理用 FPGA の構成", 情報処理学会報告 98-DA-90-9, 1998.
名称		CHESS
開発元		Hewlett-Packard, Brigham Young University
諸元	論理ブロック	ALU
	配線構造	Hexagon mesh
	再構成方式	部分再構成
	内部メモリ	256 Word x 8 bit per 16 ALUs
備考		0.35μm Process.
関連文献		Marshall, A., Vuillemin, J., et al., "A Reconfigurable Arithmetic Array for Multimedia Applications," Proceedings of the 1999 ACM/SIGDA International Symposium on Field Programmable Gate Arrays, Monterey (CA), pp.135–143, Feb. 1999.

表 A.13: 粗粒度アーキテクチャの代表的なデバイス（その2）

名称		MFAB
開発元		Imperial College (UK)
諸元	論理ブロック	Multiplier
	配線構造	不明
	再構成方式	不明
	内部メモリ	不明
備考		
関連文献		Haynes, S.D., Ferrari, A.B., Cheung, P.Y.K., "Flexible Reconfigurable Multiplier Blocks suitable for Enhacning the Architecture of FPGAs", Proc. of Custom Integrated Circuit Conference, 1999.
名称		FPAccA
開発元		広島市立大
諸元	論理ブロック	8 bit ALU
	配線構造	2D mesh
	再構成方式	不明
	内部メモリ	
備考		0.6μm メタル3層 CMOS プロセス.
関連文献		越智裕之, "FPAccA: フィールドプログラマブルアキュームレータアレイ-FPAccA model 1.0 チップの設計と評価", 情報処理学会論文誌 Vol.40, No.40, 1999.
名称		PipeRench
開発元		Carnegie Mellon University
諸元	論理ブロック	ALU
	配線構造	1D array, interconnection network
	再構成方式	部分再構成
	内部メモリ	不明
備考		0.35μm プロセス. 動作速度:100MHz.
関連文献		Goldstein, S.C., Schmit, H., et al., "PipeRench: A Coprocessor for Streaming Multimedia Acceleration," in Proceedings, International Symposium on Computer Architecture, Atlanta, GA, pp.28–39, June 1999.
名称		Chameleon
開発元		Chameleon Systems
諸元	論理ブロック	DPU (32bit ALU or 16bit Multiplier), 8 命令
	配線構造	2D mesh
	再構成方式	動的再構成
	内部メモリ	8KB per slice
備考		32bit RISC コア混載. Include 4 slices, slice=3 tiles, tile=9 DPU.
関連文献		Tang, X., Aalsma, M., and Jou, R., "A Compiler Directed Approach to Hiding Configuration Loading Latency in Chameleon Processors", Proc. of 10th International Conference on Field Programmable Logic and Applications, 2000.

表 A.14: 粗粒度アーキテクチャの代表的なデバイス（その３）

名称		RHW
開発元		NEC
諸元	論理ブロック	9 bit ALU
	配線構造	2D mesh
	再構成方式	不明
	内部メモリ	不明
備考		0.35μm プロセス. 6x20 ALU. Adder operation:126MHz.
関連文献		Yamauchi, T., Nakaya, S., Inuo, T., and Kajihara, N., "Mapping Algorithms for a Multi-Bit Data Path Processing Reconfigurable Chip RHW", Proc. IEEE Symposium on FPGA for Custom Computing Machines, 2000.
名称		MorphoSys
開発元		University of California Irvine
諸元	論理ブロック	8x8 RA, 4x4 16 bit RC per RA, RC:ALU, multiplier, shifter, register file and 32bit context register
	配線構造	2D mesh
	再構成方式	動的再構成
	内部メモリ	Context Memory : 2x8x16x32 bit
備考		MIPS-like Tiny RISC プロセッサ混載.
関連文献		Singh, H., Lee, M.H., Kurdahi, F.J., Bagherzadeh, N., "MorphoSys: An Intergrated Reconfigurable System for Data-Parallel Computation-Intensive Applications", IEEE Trans. Computers 49(5): pp.465–481, May 2000.

付録 B

業績一覧

論文

1. Toshinori Sueyoshi and Masahiro Iida, "Configurable and Reconfigurable Computing for Digital Signal Processing", IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol.85E-A, No.3, pp.591-599, 2002.
2. 浅見廣愛, 飯田全広, 中島克人, 森伯郎, "FPGA ベース並列マシン RASH での DES 暗号解析処理の改良", 情報処理学会論文誌:ハイパフォーマンスコンピューティングシステム, Vol.41, No.SIG 5(HPS 1), pp.50-57, 2000.
3. 高橋勝己, 飯田全広, 水上雄介, 山崎弘巳, 宮田裕行, 中島克人, 松本勉, "タイムメモリトレードオフ解読法に基づく暗号強度評価装置の実現性について", 情報処理学会論文誌, Vol.40, No.8, pp.3318-3328, 1999.
4. 飯田全広, 久我守弘, 末吉敏則, "スレッド制御回路を持つオンチップ・マルチプロセッサの構成", 情報処理学会論文誌, Vol.39, No.6, pp.1613-1621, 1998.

講演論文

1. Masahiro Iida and Toshinori Sueyoshi, "A Novel Programmable Logic Architecture for Reconfigurable Computing", PYIWIT02 (Pan-Yellow-Sea International Workshop on Information Technologies for Network Era), 2002.
2. Hiroai Asami, Masaharu Mizuno, Katsuto Nakajima, Masahiro Iida, and Hakuro Mori, "Application of SAR Image Reconstruction Processing on an FPGA-based Parallel Machine RASH", AI2002 (Applied Informatics 2002), pp.65-70, 2002.
3. 飯田全広, 末吉敏則, "リコンフィギャラブル・コンピューティング向き PLD の論理ブロック", FPGA/PLD Design Conference 2002, 2002.
4. 飯田全広, 末吉敏則, "リコンフィギャラブル・ロジック向き論理ブロックの提案", 信学技報 CPSY2001-79, pp.25-30, 2001.
5. 浅見廣愛, 水野政治, 中島克人, 飯田全広, 森伯郎, "FPGA ベース並列マシン RASH での SAR 画像再生処理の適用", 情報処理研究会 2001-ARC-144-4 (SWoPP 2001), pp.19-24,

2001.
6. 浅見廣愛, 飯田全広, 中島克人, 森伯郎, 佐藤裕幸, 高橋勝己, "FPGA ベース並列マシン RASH における TMTO 法暗号解析の実装 (1) －実装手法－", 情報処理学会 第 62 回全国大会, 2S-7, 2001.
7. 飯田全広, 浅見廣愛, 中島克人, 森伯郎, "FPGA ベース並列マシン RASH における TMTO 法暗号解析の実装 (2) －性能評価－", 情報処理学会 第 62 回全国大会, 2S-8, 2001.
8. 中島克人, 高橋勝己, 佐藤裕幸, 浅見廣愛, 飯田全広, 森伯郎, "TMTO 法暗号解析の FPGA ベース並列マシン上の実現", 暗号と情報セキュリティシンポジウム, SCIS2001-2A-1, 2001.
9. 飯田全広, 末吉敏則, "リコンフィギャラブルロジックにおける LUT の最適粒度に関する一検討", 信学技報 VLD2000-82, ICD2000-139, FTS2000-47, pp.77-82, 2000.
10. Katsumi Takahashi, Masahiro Iida, and Katsuto Nakajima, "Time-Memory Trade-Off Cryptanalysis on FPGA-based Parallel Machine RASH", Proceedings of HPC-Asia2000 (The Fourth International Conference / Exhibition on High Performance Computing in Asia Pacific Region), pp.366-369, 2000.
11. Katsuto Nakajima, Hiroyuki Sato, Hiroai Asami, Masahiro Iida, Katsuhiro Shindome, Hakuro Mori, Katsumi Takahashi, and Yusuke Mizukami, "FPGA-based Parallel Machine : RASH", Proceedings of AI2000 (Applied Informatics 2000), pp.269-273, 2000.
12. 浅見廣愛, 飯田全広, 中島克人, 森伯郎, "FPGA ベース並列マシン RASH の DES 暗号回路の改良", 信学技報 VLD99-102, CPSY99-111, pp.37-44, 2000.
13. 高橋勝己, 飯田全広, 中島克人, "FPGA ベース並列マシン RASH のタイムメモリトレードオフ解読法への適用", 情報処理学会 第 60 回全国大会, 2J-01, 2000.
14. 浅見廣愛, 水野政治, 中島克人, 飯田全広, 森伯郎, "FPGA ベース並列マシン RASH の SAR 画像再生処理への適用検討 (1) － RASH での SAR 画像再生処理の実現方式－", 情報処理学会 第 59 回全国大会, 5H-03, 1999.
15. 水野政治, 浅見廣愛, 飯田全広, 中島克人, 森伯郎, "FPGA ベース並列マシン RASH の SAR 画像再生処理への適用検討 (2) －市販 DSP システムとの比較検討－", 情報処理学会 第 59 回全国大会, 5H-04, 1999.
16. 中島克人, 森伯郎, 佐藤裕幸, 高橋勝己, 浅見廣愛, 水上雄介, 飯田全広, 新留勝広, "FPGA ベース並列マシン RASH", 並列処理シンポジウム (JSPP'99), p.222, 1999.
17. 飯田全広, 水上雄介, 高橋勝己, 浅見廣愛, 佐藤裕幸, "FPGA による並列暗号解析装置の構成 (1) -DES 暗号等の鍵探索-", 情報処理学会 第 58 回全国大会, 5N-08, 1999.
18. 高橋勝己, 飯田全広, 水上雄介, 中島克人, 宮田裕行, "FPGA による並列暗号解析装置の構成 (2) -ASIC との比較-", 情報処理学会 第 58 回全国大会, 5N-09, 1999.
19. 中島克人, 森伯郎, 佐藤裕幸, 高橋勝己, 浅見廣愛, 水上雄介, 飯田全広, 新留勝広, "FPGA ベース並列マシン RASH の概要", 情報処理学会 第 58 回全国大会, 1H-08, 1999.
20. 浅見廣愛, 佐藤裕幸, 飯田全広, 新留勝広, 中島克人, 森伯郎, "FPGA ベース並列マシン RASH の機能と構成", 情報処理学会 第 58 回全国大会, 1H-09, 1999.
21. Tsutomu Matsumoto, Katsumi Takahashi, Masahiro Iida, Hiroyuki Miyata, and Katsuto

Nakajima, "A Massively Parallel ASIC-based Machine for Time-Memory Trade-Off Cryptanalysis", CRYPTO'98 (Rump Session), 1998.
22. 飯田全広, 高橋勝己, 宮田裕行, 松本勉, "タイムメモリトレードオフ解読法による暗号強度評価装置の実現性検討", 暗号と情報セキュリティシンポジウム, SCIS'98-6.2.C, 1998.
23. 高橋勝巳, 飯田全広, 宮田裕行, 松本勉, "タイムメモリトレードオフ解読法を用いた暗号強度評価装置", 信学技報 ISEC97-39, pp.9-14, 1997.
24. 高橋勝巳, 飯田全広, 宮田裕行, 松本勉, "並列計算機を用いたタイムメモリトレードオフ法の実現", 情報処理学会 第 55 回全国大会, 4F-04, 1997.
25. 飯田全広, 久我守弘, 末吉敏則, "マルチスレッド制御ライブラリのハードウェア化によるオンチップ・マルチプロセッサの構成", 並列処理シンポジウム (JSPP'97), pp.337-344, 1997.
26. 飯田全広, 久我守弘, 末吉敏則, "マルチスレッド制御ライブラリのハードウェア化によるリコンフィギャラブルシステム", 信学技報 VLD96-82, CPSY96-94, pp.135-142, 1996.
27. 井上弘士, 飯田全広, 大内正英, 久我守弘, 末吉敏則, "上級コース向き教育用 32 ビット RISC マイクロプロセッサ DLX-FPGA", 情報処理学会九州支部シンポジウム, pp.60-65, 1996.
28. 井上弘士, 飯田全広, 大内正英, 久我守弘, 末吉敏則, "32 ビット RISC マイクロプロセッサ DLX-FPGA の設計教育フィジビリティ・スタディ", 信学技報 VLD95-118, 情報処理研究会 95-ARC-115-18, DA-78-18, 1995.
29. 飯田全広, 井上弘士, 大内正英, 久我守弘, 末吉敏則, "DLX-FPGA を用いた上級コース設計教育のフィジビリティ・スタディ", 電気関係学会九州支部連合会大会, 1995.
30. 西川浩司, 白井健治, 中野哲, 清水徹, 飯田全広, "テストケース自動生成ツール「Mirage」による 1 チップ CPU プロセッサの機能検証", 情報処理学会 第 46 回全国大会, 5N-06, 1994.

その他

1. 飯田全広, "HDL 入門 -検証の実際-", FPGA/PLD Design Conference 2002 (Electronic Design and Solution Fair 2002), トラック A セッション 5, 2002.
2. 飯田全広, "HDL 入門", FPGA/PLD Design Conference 2001 (Electronic Design and Solution Fair 2001), トラック A セッション 5, 2001.
3. 飯田全広, "論理合成向け Verilog HDL 基本記述スタイルと記述例", FPGA/PLD プレカンファレンス 2000, B-1, 2000.
4. 末吉敏則, 飯田全広, "リコンフィギャラブル・コンピューティング", 情報処理 Vol.40, No.8, pp.777-782, 1999.