社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2007－ＳＬＤＭ－128（17）
2007／1／18

動的リコンフィギャラブルプロセッサMuCCRAの実装

中村 拓郎† 長谷川揚平† 堤 聡† 松谷 宏紀†
Vasutan Tunbunheng† Adepu Parimala† 西村 隆† 加東 勝†
斎藤正太郎† 佐野 徹† 関 直臣† 平井啓一郎†
毛 凱毅† 天野 英晴†

† 慶應義塾大学理工学部
〒223–8522 神奈川県横浜市港北区日吉 3-14-1
E-mail: †muccra@am.ics.keio.ac.jp

あらまし MuCCRA(Multi-Core Configurable Reconfigurable Architecture) プロジェクトは、コンフィギャラブルな低電力マルチコア動的リコンフィギャラブルプロセッサに関するアーキテクチャ技術をチップレベルから提案、開発、解析することを目的としている。MuCCRA-1 はこの最初のプロトタイプであり、ローム社の 0.18μm プロセスを用いて 5mm 角のダイ上に 4x4 の 24bit PE アレイ、乗算器 4、分散メモリ 4 を実装している。単一コアの小規模な構成であるが、コンフィギュレーションの高速化と仮想ハードウェア機構を備えており、コンテキスト数の制約を気にしないでプログラムすることが可能である。PE アレイは典型的なアイランドスタイルの構造を持ち、性能とコストのトレードオフ、電力モデルの構築等に用いることができる。

Implementation of Dynamically Reconfigurable Processor MuCCRA

Takuro NAKAMURA†, Yohei HASEGAWA†, Satoshi TSUTSUMI†, Hiroki MATSUTANI†, Vasutan TUNBUNHENG†, Adepu PARIMALA†, Takashi NISHIMURA†, Masaru KATO†, Shotaro SAITO†, Toru SANO†, Naomi SEKI†, Keiichiro HIRAI†, Mao KAIYI†, and Hideharu AMANO†

† Faculty of Science and Technology, Keio University
3-14-1, Hiyoshi, Kohokuku, Yokohama, 223–8522, Japan
E-mail: †muccra@am.ics.keio.ac.jp

Abstract MuCCRA(Multi-Core Configurable Reconfigurable Architecture) project aims to establish architectural techniques to develop low-power multi-core configurable dynamically reconfigurable processors. MuCCRA-1, the first prototype chip in the project is implemented with Rohm's 0.18μm CMOS technology. On the 5mm-square die, 4x4 24bits-PE array, 4 multipliers and 4 distributed shared memory modules are mounted. Although it is a small single core, it provides a high speed configuration mechanism and virtual hardware mechanism to enable programming without taking care of the limitation of context number. PE array structure is a typical island-style architecture on which the trade-off between performance, power consumption and cost can be analyzed.

1. はじめに

動的リコンフィギャラブルプロセッサ [1] の本格的な利用が始まって数年を経過し、各種プロセッサが出揃うと共に、実用的なアプリケーションに基づくアーキテクチャの検討が進んでいる。我々も NEC エレクトロニクスにより開発された DRP-1 [2] を用いてそのハードウェア量、性能、消費電力に関して様々な解析を行ってきた [3] [4]。その結果、以下の点が明らかになっ

てきた。

- PE の内部構造、乗算器の構成、分散メモリの量、PE アレイの接続法はアプリケーションによって適不適がある。このため、アプリケーションの性質に適合した PE アレイを利用するのが望ましい。すなわち、一般的なプロセッサ同様、動的リコンフィギャラブルプロセッサも SoC に組み込む際にコンフィギャラブルであることが望ましい。このため、様々な PE アレイの構成について、そのトレードオフを解析する必要がある。
- PE アレイのサイズは、やや小さめにして、実行できるコンテキスト数を多くした方が多くの問題に対して効率が良い。高い性能が要求される場合は、複数の PE アレイをパイプライン的に用いるかスレッドレベルの並列性を利用する方法が優れている。すなわち、マルチコア構成に関してより検討する必要がある。
- 動的リコンフィギャラブルプロセッサの消費電力に関して、デバイス、アーキテクチャ、CAD を含めた解析と、それに基づく省電力技術の開発が必要である。これには、回路レベルに近い技術が必要になる。
- 多数のコンテキストをなるべく小さなオーバーヘッドで制御する現実的な方法を検討する必要がある。一方で、デバイスレベルに近い、ブレークスルーとなる新たな技術が必要である。

MuCCRA(Multi-Core Configurable Reconfigurable Architecture) プロジェクトは、上記についての解析と問題点の解決を目指した新たな方法の開拓を目的とし、実チップを開発して、これを基に様々な解析および実証を行う。

この MuCCRA プロジェクトの第一歩として、我々は今回、プロトタイプチップ MuCCRA-1 を、ローム社の 0.18μm CMOS プロセスで実装した。本論文では MuCCRA-1 のアーキテクチャと、実装結果を報告する。

2. MuCCRA

2.1 MuCCRA の概観

MuCCRA は、図 1 に示すように、複数の PE アレイ (動的リコンフィギャラブルプロセッサコア) が NoC(Network-on-Chip) で接続されている構成を持つ。

MuCCRA の PE アレイは、アプリケーションに適合した構成をコンフィギャラブルに生成する。すなわち、コアの個数、PE アレイのサイズ、構成等をアプリケーションおよび要求性能、電力、半導体面積に依存して様々な構成を取ることを可能とする。一方で、それぞれのコアの I/O を介したデータ転送制御と、コンフィギュレーションの制御、これに関連するコンテキスト制御は、すべての PE アレイで共通化する。

2.2 プロトタイプチップ MuCCRA-1

我々は、MuCCRA プロジェクトの第一歩として、前節で述べた MuCCRA の PE アレイの最も小さなサイズを、プロトタイプチップ MuCCRA-1 として実チップ上に実装した。このプロトタイプチップ MuCCRA-1 の開発の具体的な目的は以下の通りである。

- MuCCRA コアの共通となる機構、すなわち I/O メカニズム、コンテキスト制御機構、コンテキスト配送、タスク制御機構が現実的であることを実証し、必要なコストを評価する。

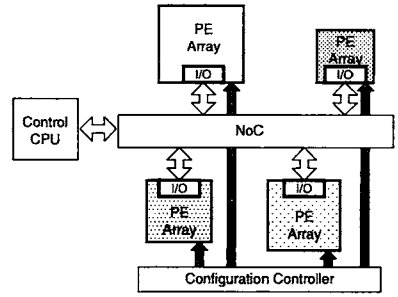

図 1 MuCCRA の概観

- PE アレイの面積に関して実チップに基づいたデータを得る。これに基づきコンテキストメモリのサイズ、PE のサイズ、結合網とスイッチのオーバヘッド等を分析する。
- PE アレイ自体、結合網部分、コンテキスト制御機構のそれぞれで消費する電力を分析する。

本稿では、MuCCRA-1 のアーキテクチャを紹介し、実装結果に基づいた面積評価と、信号処理アプリケーションの実装による性能の予備評価を示す。

3. MuCCRA-1 のアーキテクチャ

3.1 入出力インタフェース

MuCCRA-1 の入出力インタフェースは、文献 [5] で提案した、独立した I/O コントローラとダブルバッファを用いる方法に基づく。この手法では、分散メモリモジュールを 2 種類に分けてダブルバッファとして利用し、PE アレイと独立に動作する I/O コントローラにより制御する。

PE アレイはまず、ダブルバッファの一方を用いて演算を行い、この間、I/O コントローラはもう一方に対して次のデータストリームの入出力を行う。演算が終了したら、分散メモリモジュールの接続を入れ替え、再び入出力と演算を並行して行う。

上記の構成により、PE アレイ本体の演算とデータ入出力は、完全に同時動作するため、I/O 時間が演算時間よりも大きくなければ、これを完全に隠蔽することが可能である。

3.2 PE アレイ

図 2 に示すように，4 × 4 の PE(EXPE) の左端および下端にそれぞれ乗算器 4 個、分散メモリ 4 個を接続した構造を持っている。それぞれの PE は 24bit の ALU、DMU、レジスタファイルから構成される。分散メモリは、24bit × 256 エントリで、前述の通りダブルバッファとして 2 セット実装されている。このため、片方のメモリで演算を行っている間に、もう片方のメモリでチップ外部との入出力を行うことができる。チップ外部とのやりとりは、入力、出力それぞれ 64bit 幅である。乗算器は 24bit 同士を演算し、結果は下位の 24bit のみを有効としている。

PE アレイ間の結合網は、FPGA 同様の典型的なアイランドスタイルを採用した。すなわち、配線領域間にスイッチ (EXSW) を設けて縦横のネットワークを構成し、インターコネクトモジュールによって、PE の入力と出力を接続する。スイッチおよびインターコネクトは、FPGA 同様、構成情報によってスタティックに接続が決まる。したがって、それぞれのコンテ

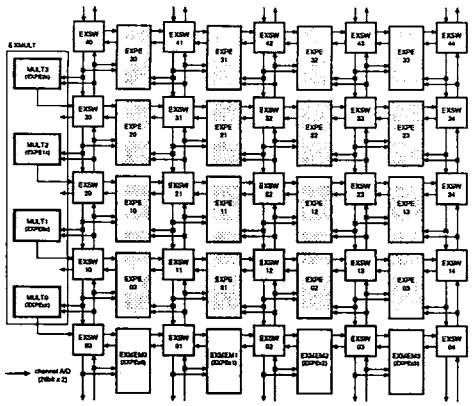

図 2　MuCCRA-1 の PE アレイ構成

キストの最大遅延時間は PE 間の接続経路に影響を受ける。MuCCRA-1 では 24bit のリンクを左右、上下両方向に対して 2 セットずつ設けている。図中では省略されているが、分散メモリへの書き込みデータは、最上位のスイッチからのフィードバックラインによって送られる。

3.3　PE の構成

図 3 に PE の構成を示す。PE は 24bit 構成で、DMU は主としてシフト、マスク、定数値供給の機能を持ち、ALU は加減算、比較、論理演算を行う。また、24bit を 12bit × 2 のデータとして扱い、2 つのデータを同時に計算する half-word 演算を行うことができる。レジスタファイルは 8 個のレジスタを持つ 2 ポート構成で、A ポートは読み書き、B ポートは読み出し専用である。

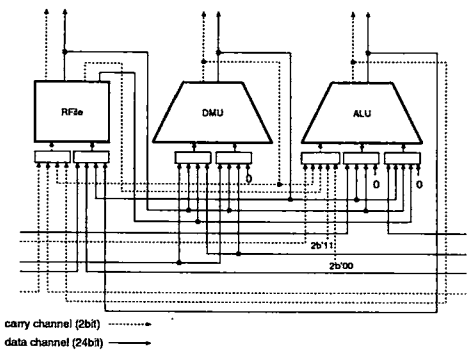

図 3　MuCCRA-1 の PE 構成

3.4　PE と結合網との接続

図 4 に PE(EXPE) と結合網の接続を示す。PE は PICKIN モジュールにより、2 系統の結合網の縦方向の両側から信号を取り入れる。結合網内の組み合わせループを避けるため、上から下へ向うデータ線から信号を取り入れる場合、必ずレジスタファイルにデータを格納する必要がある。すなわち、大規模な組み合わせ回路の演算アレイを構成する場合、MuCCRA では最も下に配置された分散メモリからデータを取り出し、これを上方向に流していき、中間結果を各 PE の rfile(レジスタファイル) に格納したり、最終結果を最上位からフィードバックラインを経由して分散メモリに格納したりする。一方、出力は PICKOUT モジュールにより横方向の線に対して行う。ALU、DMU、rfile すべてのモジュールの出力を左右どちらの方向にも取り出すことができる。

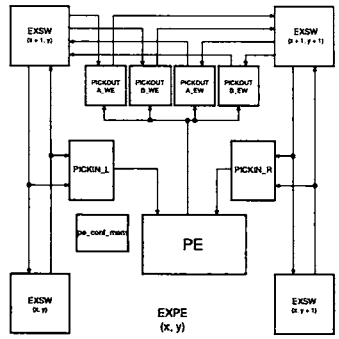

図 4　PE、SW の接続

3.5　スイッチの構成

図 5 にスイッチの構成を示す。スイッチは各方向からの入力をマルチプレクサで選択して出力するという単純な構成をとっている。マルチプレクサがどの入力を出力するかは、スイッチのコンフィギュレーションで決定する。South、East、West からの入力はそのまま出力されるが、North からの入力で East、West に向かう場合には、一度内部レジスタに格納され、次のクロックで出力される。これは、結合網中で組合せ回路のループ構造が発生するのを防ぐためである。

MuCCRA-1 はデータ用の配線を 2 系統持っているため、図 2 の EXSW はこのスイッチを内部に 2 つもつ。また、2 系統それぞれについて、同様の構成の carry 用スイッチを持もつ。

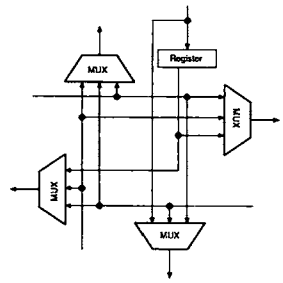

図 5　MuCCRA-1 のスイッチの構成

4.　MuCCRA-1 の制御機構

4.1　構成情報

MuCCRA-1 では PE の各モジュールの機能、入出力の選択、結合網との接続を 64bit の構成情報にまとめている。後述の RoMultiC 機構を用いてマルチキャストするためのビットマップ、コンテキスト番号等を入れて 88bit の形で集中コンフィギュレーションメモリに蓄えられる。これに対してスイッチモジュールは、3 個を並列に情報を設定するようになっている。メモリと乗算器およびコンテキストコントローラは 1 つのコードにより設定する。

4.2 MuCCRA のコンテキスト制御

MuCCRA-1 は、カウンタベースのコントローラを用いてコンテキストの制御を行っている。この方法は、図 6 に示すように、現在実行中のコンテキストを示すコンテキストカウンタ CC(64 コンテキスト：6bit) を用意し、これから次のコンテキスト番号を生成して PE アレイに対してブロードキャストする。CC の値は通常は単純にインクリメントされるが、分岐が成立した場合には分岐先のコンテキスト番号を計算する。

MuCCRA-1 におけるコンテキスト分岐は、コントローラのコンテキストメモリに格納されている Branch Enable と、特定の PE が出力する分岐アドレス badr によって成否が決定する。分岐が可能なコンテキスト (Branch Enable が 1 のコンテキスト) では、次のコンテキスト番号として CC+badr+1 がブロードキャストされる。したがって、badr が 0 であれば、分岐が成立しなかったことになる。badr はアレイの右側の PE(図 2 の EXPE03、13、23、33) のレジスタファイルが出力する。コンテキストごとに、この中のどの PE が badr を出力するかは、コンフィギュレーションデータで指定する。

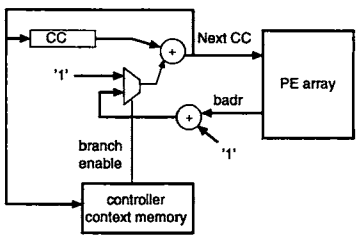

図 6 コンテキストコントローラ

4.3 RoMultiC

MuCCRA-1 は、各 PE、スイッチに小規模のコンテキストメモリを分散して持つ。さらにチップ上には、集中コンテキストメモリを持ち、立ち上げ時に、チップ外部よりこの集中コンテキストメモリに対して、全ての PE およびスイッチで利用するコンフィギュレーションデータを転送する。実行開始後は必要に応じてこの集中コンテキストメモリからそれぞれの PE、スイッチのコンテキストメモリにコンフィギュレーションデータを転送する。集中コンテキストメモリから、それぞれの PE、スイッチのコンテキストメモリへのデータ転送は基本的にはバスを用いて順番に行うが、ここで、RoMultiC [6] を利用することでマルチキャストによる高速化を実現する。

RoMultiC は、PE アレイ上の PE およびスイッチにシーケンシャルな番号を与え、これを指定してコンフィギュレーションデータを送るのではなく、図 7 に示すように縦横のビットマップを用いる方法である。共通バス上のコンフィギュレーションデータは縦横のビットマップが共に 1 である PE あるいはスイッチのコンテキストメモリに送られる。図中では、PE00,01,10,11 の 4 つに対して同時に同じコンフィギュレーションデータが送られる。

この手法はマルチキャストにより、コンフィギュレーションに要する時間を削減することが可能である。また、マルチキャストするデータは共有されるため、集中共有メモリに格納されるコンフィギュレーションデータの量は、全てのコンフィギュレーションデータを持たせる通常の方法に比べて少なくなる。

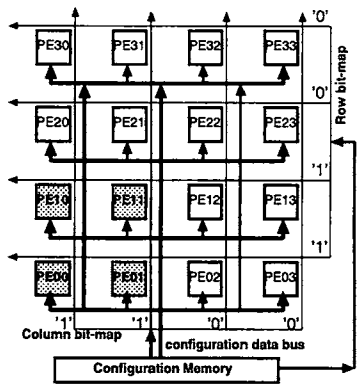

図 7 RoMultiC マルチキャスト法

4.4 MuCCRA のタスク制御

MuCCRA-1 は、各 PE が 64 コンテキストを保持可能なコンテキストメモリを持っている。これは他の動的リコンフィギャラブルプロセッサに比べかなり大きい (DRP-1 は 16 [2],ADRES は 32 [7], DAPDNA-2 は 4 [8]) が、MPEG や JPEG などといったアプリケーション全体を 64 コンテキスト以内で実装することは難しい。そこで、このようなアプリケーションを実装するために、アプリケーションを複数のタスクに分割し、タスク単位で PE アレイにコンフィギュレーションをロードし、実行する必要がある。

MuCCRA-1 は PE アレイ上の各ユニットが持つコンテキストメモリの他に、512 エントリの集中コンテキストメモリを持ち、ここから次に必要なコンテキストを PE アレイにロードする。各タスクのロードにかかる時間を隠蔽するため、現在のタスクの実行と次に実行するタスクのコンテキストロードは同時に行うことができる。

図 8 に、MuCCRA で用いるタスク制御テーブルを示す。このテーブルは各タスクについて以下のエントリを持つ。

- start: 集中コンテキストメモリ上のそのタスクに対応するコンフィギュレーションデータの先頭番地
- size: そのタスクのコンフィギュレーションデータのエントリ数
- context size: そのタスクで利用するコンテキスト数
- branch task: タスクが分岐する場合、分岐先タスク番号
- default task: 分岐しない場合のタスク番号
- jobend: このタスクで当該ジョブの終了を行うかどうか

MuCCRA-1 では、あるタスクの実行中に、タスク間制御テーブルの default task で示された番号のタスクが PE アレイ内の各コンテキストメモリにロードされる。図 8 に示した例では、まず Task0 の実行中に Task1 のコンフィギュレーションをロードする。Task1 のロードが終了した時点における各 PE のコンテキストメモリの様子を図 9(a) に示す。コンテキストメモリのエントリ数 64 に対して、この時点でロードされたコンテキストは Task0 と Task1 の和で 46 コンテキストなので、メモリには空きがあるが、さらに次のコンテキストをロードすることはしない。すなわち、あるタスクの実行中には、タスク制御テー

ブルの default task で示された番号のタスクのみをロードし、ロードを終えたらタスクコントローラはサスペンドする。

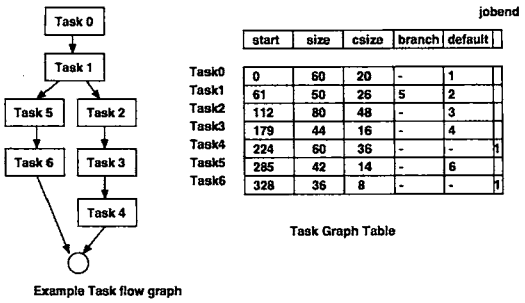

図 8 タスク制御テーブルの例

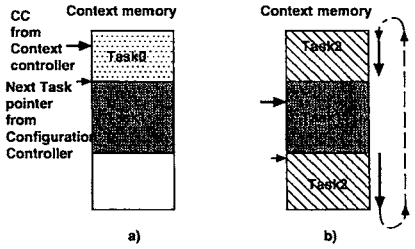

図 9 タスクのコンフィギュレーションの実行例

Task0 の実行が終了すると、PE アレイからタスクコントローラにタスク終了信号が送られてくる。このとき、Task1 のロードが終わっていれば即座に Task1 の実行を開始すると共に、Task1 の default task である Task2 の先行ロードを開始する。ロードする番地は、Task1 の最後のコンテキストの直後からで、すなわち、64 コンテキストは図 9(b) に示すように、サイクリックバッファ状に使われることになる。ここで、Task1(26) と Task2(48) の必要コンテキスト数を合わせると 64 を越えてしまうため、Task2 のコンテキストを全てロードすることができない。この場合は、Task2 を 38 コンテキスト分だけロードし、タスクコントローラはサスペンドする。

Task1 の実行終了時に、PE アレイからタスク分岐信号が送られてこなかった場合には、Task2 を実行するため、先行ロードできなかった残りの 10 コンテキストをロードする。この間、PE アレイはサスペンドする。Task1 の実行終了時にタスク分岐信号が送られてきた場合には、次はタスク制御テーブルの branch task に示された Task5 が実行されなければならない。この場合、PE アレイはサスペンドし、タスクコントローラは Task5 のコンテキストを必要無くなった Task2 のコンテキストに上書きする形でロードする。なお、タスク分岐信号はコンテキスト分岐と同様、PE アレイの右側の PE のレジスタ出力から取得する。

この手法では PE アレイ上の各ユニットが持つコンテキストメモリのエントリ数を超えるアプリケーションの実装が可能であり、さらにコンテキストロードのレイテンシを大幅に隠蔽できる。ただし、以下の 3 つのケースでストールが発生する。

（1） 現在のタスクと先行ロードするタスクのコンテキスト数の和が、コンテキストメモリのエントリ数 64 を超えたとき
（2） タスク分岐が発生したとき
（3） 先行ロードが終了する前に、現在のタスクの実行が終了したとき

また、個々のタスクはコンテキストメモリのエントリ数 64 に収まる必要がある。したがって、大きなタスクは 64 コンテキストに収まるように分割する必要がある。

5. 実　　装

MuCCRA-1 の設計には Verilog-HDL を用い，VDEC でサポートされるシノプシス社 Design Compiler 2006.06-SP2 を用いて論理合成を行った。また，レイアウト、フロアプラン、配置配線には、ケイデンス社の SoC Encounter 5.2 を 用いた。図 10 にコア中心部のフロアプラン図を示す。図 2 のアレイ構成をほぼそのままの形で配置し、中心部に集中コンテキストメモリを置いている。

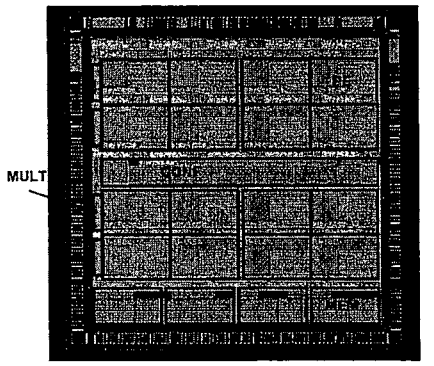

図 10 MuCCRA-1 のフロアプラン

表 1 に、配置配線後のセル数および面積を示す。PE アレイの総面積に対して、メモリの総面積 (データメモリ、コンテキストメモリ、集中コンテキストメモリを含む) は約 67%となっている。なお、メモリ面積のうち約 71%(コア面積の約 46%) は、各ユニットが持つコンテキストメモリの面積で占められている。本実装では各ユニットが 64 コンテキストを保持可能であるが、これはローム社のメモリライブラリがサポートするもっとも小さいエントリ数が 64 であったことによる。

表 1 MuCCRA-1 使用セル数および面積

使用セル数	121305
アレイ面積 (mm^2)	10.89
メモリ総面積 (mm^2)	7.33

図 11 に、PE アレイ全体の面積における各ユニットが占める面積の内訳を示す。面積は、各ユニットが持つコンテキストメモリの面積をそれぞれ含む値である。図中の Controller は、コンテキストコントローラとタスクコントローラ、および集中コンテキストメモリを含む。MEM はダブルバッファ方式のデータメモリ、MULT は乗算器である。

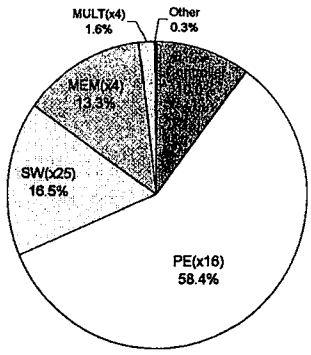

図 11 各ユニットの面積

がわかる。また、PE 1 つあたりの面積のうち、コンテキストメモリの占める割合は約 55.7% となった。これより、64 コンテキスト分のコンテキストメモリの面積が、1 PE のロジックのみの面積とほぼ等しいということがわかる。また、1 PE あたりのロジックの面積に対する 1 コンテキスト分のコンテキストメモリの面積の割合は 0.019 となった。IMEC 社の動的リコンフィギャラブルプロセッサ ADRES ではこの値は 0.031 であり [7]、MuCCRA-1 の方がコンテキストメモリの占める割合が小さいことがわかった。

また、Controller の面積のうち、集中コンテキストメモリの占める割合は約 86.8% (PE アレイ全体の 8.1%) である。これより、コンテキストコントローラ、タスクコントローラの占める面積はごくわずかで，少ない面積オーバヘッドで実現可能であることがわかった。

ただし，今回の実装では PE の面積が比較的大きく，更に最適化の余地があると考えられる．このため，実際にはコンテキストメモリやコントローラのオーバヘッドの割合は更に大きくなると考えられる．今後は，アプリケーションによる評価を進め，PE の命令の利用率などを評価し，より詳細な面積コストの解析が必要である．

6. アプリケーションによる予備評価

実装した MuCCRA-1 の配置配線後のネットリストを用いて、JPEG デコーダで用いられる DCT(Discrete Cosine Transform) を実装し、予備評価を行った。

アプリケーションの開発には，MuCCRA 向けに開発した開発環境である MuCCRA-editor を使用している．ツール上でマウス操作により，PE の機能や，スイッチ間の接続を手動で選択することができる．また，Verilog シミュレーションで用いるコンフィギュレーションデータファイルの生成を行うことができる．また，この際に RoMulTiC によるコンフィギュレーションのサイクル数を削減するようにスケジューリングを行う．

表 2 DCT のコンテキスト数と実行クロック数

TASK	CNTXT	CONF	EXEC
行方向	13	135	89
転置	15	182	14
列方向	13	135	89

今回の実装では、DCT を行方向演算、行列転置、列方向演算の 3 つのタスクに分割した。それぞれのタスクのコンテキスト数 (CNTXT)、RoMultiC によるコンフィギュレーションクロック数 (CONF)、実行クロック数 (EXEC) を表 2 に示す．動作周波数は約 45MHz である。

DCT の実装では、コンフィギュレーションクロック数が実行クロック数を大きく上回っている。各タスクのコンフィギュレーションオーバーヘッドを完全に隠蔽するためには、RoMultiC の改善、コンフィギュレーションバス幅の拡張などの工夫が必要となる。

7. おわりに

我々は今回、MuCCRA プロジェクトの第一歩として、プロトタイプチップ MuCCRA-1 を実装した。MuCCRA-1 は 4 × 4 の 24 bit PE アレイと、乗算器、ダブルバッファ方式の分散メモリを持つ。制御機構では RoMultiC によるコンフィギュレーション配送、タスク間制御が実装されている。MuCCRA-1 は Rohm 社の 0.18μm CMOS プロセスを用いて、5mm 角のチップに実装した。

今後は実チップを用いてより多くのアプリケーションを実装し、性能や電力の評価を行う。また、90nm CMOS プロセスを用いて省電力技術を導入したチップを開発する予定である。

謝　辞

本研究は，科学技術振興機構「JST」の戦略的創造研究推進事業「CREST」における研究領域「情報システムの超低消費電力化を目指した技術革新と統合化技術」の研究課題「革新的電源制御による次世代超低電力高性能システム LSI の研究」による．

本研究は東京大学大規模集積システム設計教育研究センターを通し，ローム (株)・凸版印刷 (株)・シノプシス株式会社・日本ケイデンス株式会社・メンター株式会社の協力で行なわれたものである．

文　献

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

[2] M. Motomura: “A Dynamically Reconfigurable Processor Architecture”, Microprocessor Forum (Oct. 2002).

[3] H.Amano, S.Abe, Y.Hasegawa, K.Deguchi, M.Suzuki: “Performance and Cost Analysis of Time-multiplexed Execution on the Dynamically Reconfigurable Processor”, *Proc. of the FCCM 2005.*

[4] 長谷川 揚平, 阿部 昌平, 黒瀧 俊輔, ヴ マン トウアン, 天野英晴：“動的リコンフィギャラブルプロセッサにおける時分割多重実行の 評価”, 先進的計算基盤システムシンポジウム (SACSIS2006) 論文集, pp. 135–142 (2006).

[5] H. Amano, S. Abe, K. Deguchi and Y. Hasegawa: “An I/O mechanism on a Dynamically Reconfigurable Processor - Which should be moved: Data or Configuration”, Proceedings of International Conference on Field Programmable Logic and Applications (FPL2005), pp. 347–352 (2005).

[6] V.Tunbunheng, M.Suzuki, H.Amano: “RoMultiC: Fast and Simple Configuration Data Multicasting Scheme for Coarse Grain Reconfigurable Devices”, *Proc. of IEEE FPT*, pp. 129–136.

[7] F.J.Veradas, M.Scheppler, W.Moffat, B.Mei: “Custom Implementation of the Coarse-Grained Reconfigurable ADRES architecture for multimedia Purposes”, *Proc. of International Conference on Field Programmable Logic and Applications (FPL05)*, pp. 106–111.

[8] Y.Sugawara, K.Ide, T.Sato: “Dynamically Reconfigurable Processor Implemented with IPFlex's DAPDNA Technology”, *IEICE Trans. on Inf.&Syst. Vol.E87-D, No.8*, pp. 1997–2003.