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

2004－ＳＬＤＭ－113　（９）
2004／1／22

DRP上での仮想ハードウェア機構の検討

天野　英晴†　　犬尾　　武††　　紙　　弘和†††

† 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
†† NECシリコンシステム研究所 〒229–1198 神奈川県相模原市下九沢 1120
††† NECマルチメディア研究所 〒211–8668 川崎市中原区下沼部1753
E-mail: †hunga@am.ics.keio.ac.jp, ††inuo@mel.cl.nec.co.jp, †††kami@ccm.cl.nec.co.jp

あらまし　仮想ハードウェア機構をNEC社のDRP(Dynamically Reconfigurable Processor)上に実装する場合の方式および問題点に関して検討する。ループ構造がチップ内のコンテキストに入りきれない場合でも、部分変更を用いることができれば、コンテキストの選択的な実行、割り付けアルゴリズムと読み込みアルゴリズムの工夫により、読み込み時間を隠蔽することが可能である。離散シミュレータの実装例では、56クロックの損失を28クロックまで低減することが可能である。

キーワード　リコンフィギャラブルシステム、仮想ハードウェア

A Virtual Hardware mechanism on DRP

Hideharu AMANO†, Takeshi INUO††, and Hirokazu KAMI†††

† Department of Information and Computer Science, Keio University　3-14-1, Hiyoshi, Kohokuku, Yokohama, 223–8522, Japan
†† NEC Silicons Systems Research Laboratories, 1120 Shimokusawa, Sagamihara, Kawasaki, 229–1198, Japan
††† NEC Multimedia Research Laboratories, 1753 Shimonumabe, Nakahara, Kawasaki, 211–8668, Japan
E-mail: †hunga@am.ics.keio.ac.jp, ††inuo@mel.cl.nec.co.jp, †††kami@ccm.cl.nec.co.jp

Abstract　Implementation methods for a virtual hardware mechanism on NEC's reconfigurable device DRP are proposed and discussed. Even for the case that the loop structure cannot be implemented on contexts inside the chip, the overhead caused by loading of the configuration data can be hidden with a combination of partial configuration technique, dynamic context selection, the context assignment policy and the context reading policy. In an example implementation, the overhead is reduced from 56 clocks to 28 clocks.

Key words　Reconfigurable Systems, Virtual Hardware

1. はじめに

NECのDRP[1]、IPFlex　DAP/DNA[4]、Quicksilver社ACM[3]など、最近開発されたダイナミックリコンフィギャラブルプロセッサは、以下の二つの機能を持っている。

- かなりの数のコンテキストをチップ内に保持するマルチコンテキスト機能
- 実行動作を妨げずに、利用していないコンテキストに対して外部から構成情報を設定する機能

これらの特徴を利用すれば、チップ内部で実現可能なハードウェア量を越える回路や、多くの機能を実現する仮想ハードウェア[2]の実現が可能となる。仮想ハードウェア(VHw)の動機は主として以下の二つである。

- リコンフィギャラブルデバイスの持つコンテキスト数の全てを用いても実現できない巨大な回路の動作を可能とする。
- 対象アプリケーションのサイズはマルチコンテキストデバイスのコンテキスト数に納まるが、複数のアプリケーションを実行する必要があり、それら全てを単一チップに格納しておくことはできない場合。この場合は、要求に応じて複数アプリケーションを切り替えて用いる必要がある。

本報告では、前者の単一ジョブVHwのみを扱う。なお、本報告の内容は、VHwの先行概念であるWASMIIおよびDRPの構成に対する依存が大きいが、紙面の関係で充分記述することができない。これらの点については文献[2][1]を参考されたい。

2. 単一ジョブVHwの基本事項の検討

ここでは、「コンテキスト」を特定のハードウェアを実現するためのConfiguration Dataと定義する。コンテキストの読み込み時間とは、このConfiguration Dataをチップ外(あるいはチップ内部の補助的なメモリ)から読み込んで設定するまでの時間を指す。また、ある設計において一つのコンテキストが実行を開始してから切り替わるまでのクロック数の平均をコンテ

キスト平均スイッチ間隔と呼ぶ。

2.1 コンテキストサイズとループ構造

単一ジョブVHwは、対象とするデータフローグラフにおけるループ構造と実行するコンテキストのサイズとの関係によりその性質が定まることがわかる。

a) (ケースA) 一つのコンテキスト中にループ構造がすっかり入ってしまう場合：

ループの回数が充分大きければ、コンテキストのスイッチ間隔が、コンテキストの読み込み時間より大きくなる。この場合は次に実行する可能性のあるコンテキストを先行して読み込むことにより、読み込み時間を隠蔽することができる。単一VHwの実現が最も容易なケースである。

b) (ケースB) ループ構造を実現するのに要する複数コンテキストが、チップ内にすっかり入ってしまう場合:

コンテキストのスイッチ間隔自体は短いが、ループを終了するまで、次に実行するコンテキストを外部から読み込む必要が生じない。このため、ループ実行中に次に実行する可能性のあるコンテキストを先行して読み込むことにより、読み込み時間を隠蔽することができる。ただし、次に実行するループ構造全体をチップ内の残りのコンテキスト中に先行読み込みできないと、ケースC同様に読み込み時間がボトルネックとなる。

c) (ケースC) ループ構造全体をチップ内に格納することができない場合:

コンテキストセットの読み込み時間が、平均スイッチ間隔を越えると、設定に要する時間が全体を支配する。この状態が続くと、チップ内のコンテキスト数は2より大きくても性能とは関係なくなり、システムはひたすらコンテキストセットを入れ替えつつ実行を行なう論理エミュレータとなる。

さて、DRP-1に現在までに実装されたアプリケーションにおけるコンテキストの設定時間と平均スイッチ間隔について検討する。

現在のDRP-1は、粗粒度なので、通常のFPGAよりもはるかに構成情報の量は少ない。とはいえ、アプリケーションによって異なるものの、1コンテキスト当たり30000bit以上になることが多い。これを、32bitのPCIバスから読み込むためには、これもアドレスとデータの組にする際のロス等で、1000 clock程度は必要である。

コンテキストセットの設定時間について以下の改善を行なうことができる。

- ピンを増やす。また、読み込み周波数を動作周波数に比べて高速にする。それぞれ倍程度が限度と考えられる。
- Configuration Dataの圧縮による高速化。今までの検討[5]によると半分程度には圧縮可能。
- 内部に大規模なバックアップ用メモリ(Configuration Cache)を置き、必要時にConfiguration Dataの高速大容量転送を行なう。

全てを施しても数十から100 clock程度までの短縮が限界と考えられる。

これに対してコンテキストの平均スイッチ間隔は、後に述べる離散系シミュレータが5, 省電力版Viterbi [8]が4.3clock、MPEGのMDCT [7]が3.5clock(入出力を除く)などであり、オーダが1桁から2桁異なる。

したがって、現状のDRP-1のサイズではケースAはやや非現実的であり、ケースBまたはケースCについての検討が必要であることがわかる。本稿では、このうち最も条件の悪いケースCについて重点的に考察するが、この結果はケースBにも応用可能である。ケースCのような場合に読み出し時間を隠蔽ないし大幅に短縮するためには、以下のような特殊な状況が必要であろう。(1) 動的適応型ハードウェア[6]のように、読み込み時に、「つなぎ」の処理をやってくれる回路を用意できる場合。(2) チップ内のコンテキストを用いて、読み出し時間を隠蔽するために投機的な実行を行なうことができる場合。(3) 部分変更可能な場合。

これらのうち、現状で、現実可能性が高いのは部分変更可能な場合である。すなわち、基本となる回路構成がチップ内に存在し、これを部分的に書き換えることで、他の構成に変更可能な場合は、構成情報の設定時間を大幅に節約可能である。この場合でもコンテキストの平均スイッチ間隔が数クロックであれば、対象回路で取り扱うデータセット自体の入れ替えを行なうことは現実的ではない。このため、ケースCにおいて多くのアプリケーションで単一ジョブVHwが現実的なのは、(1) 構成情報は部分変更可能、(2) 利用するデータセットについては全体がチップ内に納まる。本稿では上記を仮定してDRP-1上での実装を検討する。

2.2 コンテキストの部分変更

多くのアプリケーションでは、コンテキストを類似構造を持ついくつかの種類に分けることができる。後述する離散系シミュレータでは、PUモジュール、スイッチモジュール、メモリモジュールの3種類に大きく分けられる。また、MPEGのMDCT [7]ではcontext4,5,11,12,13,14は類似構造を持ち、context9, context10も類似構造を持つ。類似構造を持つコンテキスト間の相違点は、(1)VMEMなどの記憶素子や入出力との接続線 (2)演算器の機能 (3)定数 などが多い。

2.2.1 部分変更 vs. 単一回路の機能切り替え

コンテキストの部分変更は、Configurationの読み込み時間を劇的に減らすことが可能であるが、その分変更可能な範囲が狭くなってしまう。類似構造を持つコンテキストの共通部分をA、個別に変更する部分をa,a'とし、A+aをA+a'に部分変更する場合を考える。この時、a,a'がAに比べて非常に小さいと、A+a+a'の構造を持ち、aとa'を切替える機能を持ったハードウェアを構成した方が有利となってしまう。

すなわち、a,a'は、コンテキストの平均スイッチ間隔と同レベルで読み込むことができる程度に小さくなければならないが、A+a+a'すなわち、aとa'を切替える機能を持たせる設計に対してハードウェア要求量および遅延時間の点で一定のメリットを持っていることが必要である。

部分変更を用いて単一ジョブVHwを実現する場合、この条件を満足することのできる設計を見出すことができるかどうかが問題となる。後述する離散系シミュレータのPUモジュールにおいては以下のようになる。それぞれ別のContextとして設

計されたPU0, PU1と、それらを共に実現し、外部からの入力によりこの機能を切り替えて使えるモジュールPUallを比較した結果を表1に示す。

表1 部分書き換えと機能切り替え

回路	ハードウェア要求量	クリティカルパス
PUall	55 PEs 22 Vmem	27457psec
PU0/PU1	47 PEs 19 Vmem	22555psec

結果を見ると、ある程度のメリットはあると見做すことができる。

一方で、PU0/PU1相互間を変換するのに必要な構成情報の設定時間は、10clock(後述する変更を行なえばさらに減らすことが可能)程度である。この数値は、大きいものではないが、PUモジュールの実行時間が4clockであることを考えると、設定機構の拡張を行なってはじめて部分変更が有効になるレベルと言える。

3. 単一ジョブVHwの実現

3.1 単一ジョブVHwの実現に要する条件

部分変更を基本とした単一ジョブVHwを効率的に実現するためには、DRP-1に対して以下の拡張が必要である。

- 量的拡張：

– 現在のDRP-1は論理状態番号が64であるが、VHwではこれを論理コンテキスト番号として用いるため、この数の限界が単一ジョブ実行時に扱える最大論理コンテキスト数となる。64より大きい論理コンテキスト番号を利用する必要があれば、拡張が必要である。

– 現在のDRP-1は内部メモリVMEM(8bit、256エントリ)を80セット持つ。先に述べたように、本稿では単一ジョブVHwでのデータセットの入れ換えは考えないので、論理コンテキスト間の交信、状態保持を含め、全ての情報をVMEMに保持する。このため、エントリ数が不足すれば拡張の必要がある。

- 機能的拡張の必要性：

– コンテキストの先行読み込みには、設定(読み込み)指示を外部に出す機能が必要である。

– 現在のDRP-1のSTCによるコンテキストの制御を、コンテキストがチップ内に存在するかどうかに依存した動的な制御が可能なように拡張すると性能が改善される可能性がある。

– その他、構成情報設定機能の強化および構成情報のクリア機能が必要であり、以下の章に詳述する。

3.1.1 構成情報設定機能の強化

前節に検討したように、現在のDRP-1におけるコンテキストの平均スイッチ間隔は数クロックである場合が多く、現在の構成情報設定機能では、部分変更を考えた場合ですら変更時間がボトルネックとなってしまう。しかし、後に示すように、部分変更に要する情報量の変更自体は大した量ではない(数十バイト)ため、以下の工夫により、この点を回避することは可能である。

- 部分変更は、様々な構成情報メモリのアドレスに対して、多くの場合数ビットの変更が必要である。ここで、同一コンテキストが対象なので、アドレスの上位は固定される。したがって、bit数の多いワード単位で書き込む構成は有利ではなく、(16bitアドレス+バイト)の組をいくつか作って、同時に書き込む構成が有利である。この組み合わせを4つ設けて、4箇所同時変更が可能にする場合、96bitの入力ピンが必要である。さらに、書き換えの周波数をアプリケーションの動作周波数の2倍にすることができれば、平均スイッチ間隔と同程度の変更時間で部分変更が可能になる。

- 部分変更に要する情報量自体は大したことはないため、差分構成情報用メモリをチップ内に設ける。この方法は前者の方法に比べると、コスト、消費電力共に有利であり、現実的である。前述の並列設定も、4より多くのモジュールに対して行なうことができる。ただし、部分変更で必要な情報量は、個々の設計で相当の差があるため、差分メモリの構成を工夫しないと、無駄な領域が多くなる可能性がある。

いずれの方式も構成情報メモリをある程度のブロックで並列に書き込むことのできる機能が必要である。また、変更用の構成情報は、同一のブロックに同時に書き込まないように、あらかじめ編成しておく必要がある。

3.1.2 構成情報クリア機能

部分変更により、A+aをA+a'に変更するには、aをa'に変更するのに必要な構成情報を書き込めば良いのだが、一般に部分変更可能なグループ間で書き込みの構成情報の範囲は異なる。このため、aをa'に変更するのと、aをa"に変更する場合、さらにはa'をa"に変更するのでは、それぞれ異なった差分構成情報が必要となる。これでは、入れ替えの柔軟性を高めるためには、様々な組み合わせの差分構成情報が必要になってしまう。

この点を解決するには、Aに対する差分構成情報のみを用意し、A+aからA+a'への切り替えには、(1) aに対応するアドレスに全て0にセット。(2) a'を設定。という2段設定が必要となる。すなわち、自動的に前コンテキストの差分構成情報を参照して、これを消去し、次のコンテキストの差分構成情報を設定する機構が望ましい。この方法が適用可能なのは、チップ内部に差分構成情報メモリを設ける方法に限られる。

3.2 コンテキスト非依存な設計

VHwの対象となるアプリケーションは、現在実行中のコンテキストに依存しない形で実行できなければならない。このため、物理コンテキスト番号と論理コンテキスト番号を分離する。論理コンテキスト番号は、DRPにおける論理状態番号とし、現在のDRP-1では64となる(不足の場合は拡張の必要がある)。物理コンテキスト番号は、実際にそのコンテキストがチップ上で割り当てられるコンテキスト番号であり、DRP-1では16である。この割り当ては、チップ内のアドレス変換テーブルにより管理される。

さて、各コンテキストの動作は、全て論理コンテキスト番号を基に設計する必要がある。このために、コンテキスト番号により指定する必要のあるRFUは、コンテキスト滞在時の中間データ格納にのみ利用する。また、FFUは全コンテキストで常に利用する特殊な共有データにのみ利用する。すなわち、コンテキストの状態を含む、全てのデータはVMEMに格納する必

要がある。ここでは、256エントリのVMEMを順にそれぞれの論理コンテキストに割り付ける。したがって各コンテキストは4エントリ、80セットを占有して利用可能となる(これも不足する場合は拡張の必要がある)。あるコンテキストは、どの物理コンテキストで実行されても、論理コンテキスト番号でアドレスするVMEM内のデータを利用することから、物理コンテキスト番号に依存しない動作が可能である。

他のコンテキストとのデータの交換は、他の論理コンテキストに割り当てられたVMEMからデータを読み出すことによって行なう。自分に割り当てられた領域にのみ書き込むことを許し、他からは読み出しのみとする。

この方法では処理は以下のように進む。(1)他の論理コンテキストに割り当てられたVMEMよりデータを読み出す。同時に自分のVMEM領域からそのコンテキストの現在の状態を読み出す(状態を持たない場合は不要) (2)演算などの処理を行なう(3)結果および状態をVMEMの自分の領域に書き込む。(4)コンテキスト切り替え。

この手法では中間データは保存されないので、コンテキスト切り替えは、必ず結果および状態を自分のVMEM領域に書き込んでから行なう必要があり、実行途中の切り替えは禁止されている。

3.3 Context Switching Policy

現在のDRP-1のコンテキスト制御は、STCを用いた状態遷移で行なっている。各論理コンテキストは、現在実行中のコンテキストに関するイベントの有無によって、あらかじめ設定した論理コンテキストのどれかに遷移する。イベントの設定によって動的な要素を採り入れることは可能だが、論理コンテキストがチップ内に存在するかどうかなどに依存する動的な制御を行なうことはできない。

一方、WASMIIで提案されたデータ駆動型の制御[2]は、条件が揃ったコンテキストのどれかを選んで実行する点で動的要素が強い方法である。この方法の問題点は、条件が揃ったことを検出するコストに比べて、メリットが見いだせないアプリケーションが多いことである。ただし、VHwを実現する場合、コンテキストがチップ内に存在するものから順に実行する等、ある程度の動的制御を許すことが有効であるかもしれない。

そこで、限定的な動的制御を以下の形で採り入れたい。まず、一定の論理コンテキストのグループを定義する。このグループは部分書き換え可能なもの同士から形成されることを想定する。このグループに属するコンテキストに関して、

- 特定の条件に合うコンテキストを任意の順番で一回実行する。(選択的実行)
- グループの全コンテキストを任意の順番で一度だけ実行する。(バリア自由実行)

の二つの方法について検討を行なう。図1においては、選択的実行を行なう場合、A,B,C,Dの条件を満足するものが一回ずつ実行され、ブロック的実行を行なう場合は、A,B,C,Dが任意の順番で実行される。

ここで、選択的実行の条件は、設計者が判定用のコンテキストを用いて設定することとする。今回の実現方法では、それぞれのコンテキストが実行可能かどうかは、VMEM中のデータセットをチェックしなければ判定できない。このデータセットの判定には、VMEMを順に読み出して判断する必要がある。VMEMは80セットに分離されており、問題によって判定の方法は異なるため、判定操作を、固定したハードウェアで行なうのは非効率的である。そこで、このような判定を行なう場合は、設計者が判定用のコンテキストを別に設けることとする。上記の動的判定機構と組み合わせることにより、多様なContext Switching Policyが実装可能となる。離散系シミュレーションを例にこの方法を後述する。

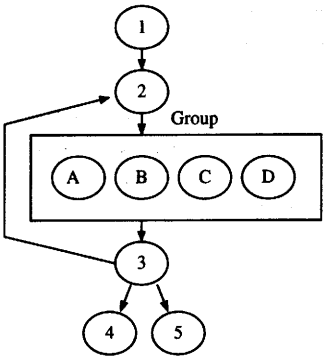

図1 拡張した状態遷移

3.4 Context Loading Policy

Context Loading Policyは、論理Contextを物理Contextのどこに割り付けるかを定めるContext Allocation Policyと、いつ、どのContextを選んでLoadingするかを定めるLoading Context Selection Policyに分けられる。

3.4.1 Context Allocation Policy

64の論理コンテキストから16の物理コンテキストへの割り付けを決めれば良いので、キャッシュのライン割り付けなどに比べて少数である点が特徴である。通常、単一ジョブVHwにおいては、Context Allocationは設計者が制御した方が効率が良い。ここでは、(1)部分書き換えでないと構成情報の設定時間がボトルネックになる。(2)部分書き換えであっても、平均スイッチ間隔と構成情報の設定時間が近いこと。の二つを考慮して以下の方法を提案する。

- 繰り返し実行されるコンテキストと一回のみ実行するコンテキストを分け、一回のみ実行するコンテキストに対して2つの物理コンテキストを割り付ける。この二つの物理コンテキストをMiscコンテキストと呼ぶ。
- 繰り返し実行されるコンテキストを、部分書き換えを行なうグループを形成するものとしないものに分ける。形成する場合、各グループに対して2つの物理コンテキストを割り付ける。
- グループを形成しないコンテキストに対して物理コンテキストを静的に割り付ける。この場合、コンテキストスイッチ間隔に対して構成情報の設定時間が大きいものを優先する。
  - 物理コンテキストが不足した場合、割り付けられなかっ

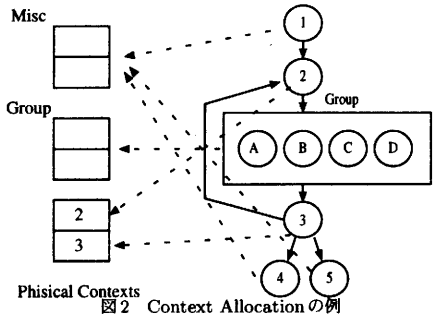

図2 Context Allocationの例

た論理コンテキストは、Miscコンテキストに割り当てる。

– 物理コンテキストが余ったら、繰り返し実行されるコンテキストのグループに分配し、適当な論理コンテキストを静的に割り当てる。これはケースBのように複数コンテキストで実行されるループ構造に対処するためである。

図1に対応するコンテキスト割り付けの例を図2に示す。1,4,5はMiscコンテキストに、A-Dは対応するグループコンテキストに割り付けられ、ループ中の2,3はそれぞれ独自のコンテキストに静的に割り付けられている。

この手法は、「コンテキストの実行ループが単純であり、コンテキストスイッチ間隔と構成情報設定時間が近い場合は、二枚以上コンテキストがあっても役に立たない」という考え方に基づいている。しかし、部分書き換えか全面書き換えかの差は大きいことから、チップ内の物理コンテキストは、それぞれのグループに対して2枚分コンテキストを供給することに対して優先的に利用する。この方法では、あるコンテキストが実際に設定される物理コンテキスト番号は、あらかじめ決まった番号であるか、割り当てられた2つの物理コンテキストのうちのどちらかである。二つの可能性がある場合、片方が実行中の場合はそうでない方を選び、実行中でなければランダムに選んで良い。

3.4.2 Loading Context Selection Policy

実行中にどのようなタイミングでどのコンテキストの構成情報を設定するかがLoading Context Selection Policyである。もっとも単純なのは、必要なコンテキストをオンデマンドで読み込んで設定する方法であるが、実行と構成情報設定のオーバラップができなくなるので、最後の手段とすべきである。

基本的な手法は以下の通りである。

- 実行中の論理コンテキストの次に実行するコンテキスト、あるいはコンテキストのグループが決まっている場合は、Context Allocation Policyに従って構成情報の設定を行なう。
- 次に実行するコンテキストの可能性が複数存在する場合、同一グループ内のコンテキストを優先して選択して構成情報の設定を行なう。

この方法は、平均スイッチ間隔がコンテキストの構成情報設定時間と同等であることを想定しており、一つのコンテキストの実行中に複数のコンテキストを読み込み可能な場合、あるいは複数コンテキストで構成されるループを複数もつ場合などは、別の方法を用いるべきであろう。

4. 離散系シミュレータの実装

4.1 離散系シミュレータ

単一ジョブVHwの実装例として、離散系シミュレータによる並列計算機のシミュレーションをDRP-1上に実装した。離散系シミュレータは、呼の発生源(この場合PU)、呼を受け取って処理する伝送系(この場合スイッチ)、呼を消費するサーバ(この場合メモリ)から構成される系のシミュレーションを行なうプログラムで、結合網や交通などのシミュレーションに用いられる。ここでは、MIN(Multistage Interconnection Network)で接続された並列計算機をシミュレーション対象とした。このシステムは図3で示す構成で、以下の要素から構成される。

- PU: 論理クロックに1に対して、一定の確率で読み出し要求、書き込み要求パケットを発生する。膨大なクロック数のシミュレーションに耐えるため、192bitのM系列の乱数発生器を内蔵し、これにより要求を出すかどうか、要求先メモリを決定する。読み出し要求を発生した場合、メモリがアクセスされて結果が戻るまでは、次のアクセスを発生しない。
- Switch: 2入力2出力のスイッチで、各ステージでShuffle接続してOmega網を構築する。パケットのヘッダを解析してDestination Routingでルーティングを行なう。これらの処理に対して2クロックずつ消費する。また、入力リンクにサイズ4のパケットバッファを持ち、Store-and Forward方式でパケットを転送する。PUからメモリに向うForward Networkと、メモリからPUに向うBackward Networkの2セットを持つ。
- Memory: 入力にサイズ4のパケットバッファを持ち、書き込みは1論理クロック、読み出しは2論理クロックで処理を終了する。読み出しの場合、Backward Networkを経由してパケットをPUに戻す。

論理クロックは、全モジュールのシミュレーションが終わった時点で更新し、バッファの内容を入れ替えてシミュレーションを進めていく。

今回の実装では、図3に示すようにPU, Switch, Memoryそれぞれ1個について1コンテキストを割り当てた。PUやMemoryはハードウェア要求量が少ないため、2モジュール1コンテキストが可能だが、今回は実装の容易さから1モジュール1コンテキストとした。この場合、図3のように4PU-4Memoryのシステムでは、PUで4、Memoryで4、Forward Networkで4、Backward Networkで4、すなわち16コンテキストを必要とする。本来DRP-1は16コンテキストを持っているので全てのコンテキストを収容可能であるが、ここでは、8コンテキストのみ利用可能とし、16論理コンテキストを8物理コンテキストで実行する状況を想定して評価を行なった。

4.2 部分変更

それぞれのコンテキストスイッチの間隔と、部分変更に要するクロック数を表2に示す。ここで、変更ワード数、変更バイト数は、それぞれ部分変更に必要なワード数、バイト数であり、変更クロック数は、4バイト同時変更、倍速設定可能な場合(チップ内に差分構成情報データを持たせて倍のバンド幅で変更可能な場合も含む)を仮定した部分変更に必要なクロック

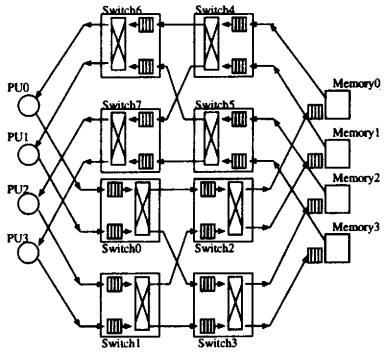

図3　対象となるMIN接続型マルチプロセッサ

数である。( )内は倍速設定はできない場合である。なお、ここでの数値は、A+aをA+a'に直接変更する場合であり、それぞれ変更する構成情報の組み合わせに対する差分構成情報データが必要である。

表2　スイッチ間隔と読み出し時間(clock数)

回路	間隔	変更ワード数	変更バイト数	設定クロック数
PU	4	6	10	2 (3)
Switch	8	45	61	8 (16)
Memory	8	35	52	7 (14)

まず倍速設定可能な場合、それぞれのコンテキストのスイッチ間隔は設定時間以下になるため、それぞれのグループに2割り当てれば、後はどのグループに複数コンテキストを割り当てても良い。ここでは原則に従って、スイッチ間隔と設定時間の差が最も少ないコンテキストであるSwitchに多くのコンテキストを割り当てる。

- 物理コンテキスト0,1: PU
- 物理コンテキスト2,3: Switch0, Switch1
- 物理コンテキスト4,5: 他のSwitch
- 物理コンテキスト6,7: Memory

今、コンテキストの制御をPU0, PU1, PU2, PU3, Switch0, Switch1,...のように順番に行なって行き、次に実行すべきコンテキストを先行して読み出せば、PU3の実行時に次に実行されるSwitch0は常駐しているため、それぞれの構成情報設定時間を完全に隠蔽することができる。

一方、倍速設定ができない場合、SwitchおよびMemoryは、構成情報設定時間がスイッチ間隔を上回る。このため、Switch1/2, Switch2/3, Switch4/5, Switch6/7の切り替え時にそれぞれ8clock、Switch7/Memory0, Memory0/1, Memory1/2, Memory2/3の切り替え時にそれぞれ6 clock、総計56clockの損失が生じる。

4.3　動的実行の効果

離散系シミュレーションは全モジュールが一巡した段階で論理時間とバッファを更新して実行する。このため、基本的にはモジュールはどこから実行を始めてもかまわない。ここで、倍速設定ができない場合の損失は、Switchグループおよびグループに対してバリア自由実行を施すことで低減できる。この場合、Switch1の終了後、チップ内に存在するSwitchコンテキストに対して処理を先に実行することができるため、2回分のコンテキスト切り替えがオーバヘッドなしで可能になる。同様に、Memoryもチップ内に存在するコンテキストを利用できることから2回分のコンテキスト切り替えの損失を低減できる。このことによって、損失は、28clockとなり半分に低減する効果がある。

倍速設定可能な場合は、今回実装した方式では効果は得られないが、以下のように最適化した場合に、効果が得られる。すなわち、PUは毎回呼を発生するかどうか乱数を発生してチェックする必要があるが、SwitchおよびMemoryは、入力Queueにパケットが存在しなければ何も処理する必要がない。この場合、各モジュールはチェックに必要な2クロックのみで次のコンテキストに切り替えるように最適化可能である。この場合は、倍速設定ができない場合と同様に、バリア自由実行の効果が期待できる。一方、入力Queueにパケットが存在することをチェックして、選択的実行を行なうことができれば、イベントの存在するコンテキストのみ実行可能であり、イベント数が少ない場合には効果が期待できる。

5. おわりに

ここでは、もっとも困難なケースCに対応する方法を主として検討したが、当面有効になるのはケースBの場合と考えられる。より現実的なアプリケーションに対する検討を行なっていく予定である。

謝　辞

DRP及び開発環境を提供して頂き、本研究に対する多くのアドバイスをしてくださったNECマルチメディア研究所の戸井崇雄氏、粟島亨氏、若林一敏氏NECシリコンシステム研究所の梶原伸樹氏、NECエレクトロニクスの古田浩一朗氏、藤井太郎氏、安生健一朗氏、本村真人氏に深く感謝します。

文　献

[1] M.Motomura:"A Dynamically Reconfigurable Processor Architecture," Microprocessor Forum, Oct. 2002.
[2] X.-P. Ling, H. Amano, "WASMII: A Data Driven Computer on a Virtual Hardware" Proc. FCCM, pp. 33–42, (1993).
[3] P.Master: "The Age of Adaptive Computing Is Here," Proc. of FCCM, pp.1-3 (2002).
[4] htpp://www.ipflex.com/.
[5] T.Kitaoka, H.Amano, K.Anjo, "Reducing the configuration loading time of a coarse grain multicontext reconfigurable device," Proc. of FPL2003, to appear
[6] H.Amano, A.Jouraku, K.Anjo, "A dynamically adaptive switching fabric on a multicontext reconfigurable device," Proc. of FPL2003, to appear
[7] Y.Yamada, K.Deguchi, N.Kaneko, H.Amano, "Core Processor/Multicontext Device Co-design," Proc. of Cool chips VI, pp.82, 2003.
[8] 金子、伊澤、後藤、天野"SISOビタビデコーダのマルチコンテキストデバイスDRP上への実装," 第1回リコンフィギャラブルシステム研究会、2003年9月.
[9] E.L.Horta, J.W.Lockwood, D.Partour, "Dynamic Hardware Plugins in an FPGA with Partial Run-time Reconfiguration." Proc. of DAC2002, June (2002).