システムＬＳＩ設計技術　104－3
(2002. 1. 23)

マルチコンテキストFPGAを用いたネットワークコントローラ

天野　英晴†　金子　直人†　渡邊幸之介†　宇野　正樹†　土屋潤一郎†

† 慶應義塾大学理工学研究科
〒223-8522 神奈川県横浜市港北区日吉 3-14-1

E-mail: †{hunga,kaneko,nosuke,uno,tutijun}@am.ics.keio.ac.jp

あらまし　マルチコンテキスト FPGA は、構成情報をチップ内に複数持ち、マルチプレクサにより切り替えることにより、瞬時に構成を変更する機能を持つ FPGA である。本報告では、PC クラスタ用ネットワークインタフェースコントローラ Martini のパケット送受信コア部をマルチコンテキスト FPGA 上に実装する手法を示す。評価の結果、マルチコンテキスト化により元の回路の面積の 30%から 40%で動作し、かつ元の回路では困難だった複数チャネルからの並行受信も可能となった。

キーワード　リコンフィギャブルシステム、マルチコンテキスト FPGA、ネットワークインタフェース

Design of a network interface with a multicontext FPGA

Hideharu AMANO†, Naoto KANEKO†, Konosuke WATANABE†, Masaki UNO†, and Jun'ichiro TSUCHIYA†

† Faculty of Science and Technology, Keio University
3-14-1, Hiyoshi, Kohoku-ku, Yokohama, 223-8522, Japan

E-mail: †{hunga,kaneko,nosuke,uno,tutijun}@am.ics.keio.ac.jp

Abstract　Multi-context FPGA is a reconfigurable device which provides multiple sets of configuration memory and changes its logic quickly by replacing them. Here, a design of core packet handlers used in a PC-cluster network interface controller Martini is changed for implementation on a multi-context FPGA. From logic synthesis results, required area is reduced to 30% - 40% of that for original design. A concurrent receiving from multiple channels which was difficult in the original design can be easily realized with the multi-context environment.

Key words　Reconfigurable Systems, Multi-context FPGA, Network Interface

1. はじめに

FPGA 上に複数の Configuration RAM を持ち、これを高速に切り替えるマルチコンテキスト FPGA は、NEC による DRL [4] の開発によりいよいよ本格的な実用化時代を迎えようとしている。我々は、1992 年よりマルチコンテキスト FPGA を利用して、仮想ハードウェア機構を実現するためのシステム WASMII [3], HOSMII [5] の研究を行ってきた。これらの研究では、マルチコンテキスト FPGA の Configuration RAM を制御する手段としてデータ駆動型の原理を利用した。この方法は、「演算を目的とするハードウェア」に対しては有効であり、かなり広い応用分野で、少量のハードウェアで大規模な問題を処理することができることを示した。しかし、実際の応用分野では、純粋に演算目的の Reconfigurable Architecture は、汎用 CPU や DSP との厳しい競争にさらされることから、さほど広く用いられているわけではない。

複雑で巨大な Reconfigurable Architecture は、むしろ、CPU や DSP では性能を上げることの難しい演算以外の分野への利用が期待されており、ネットワークインタフェースは、その一つの応用分野である。本研究は、より汎用の仮想ハードウェア実現の第一歩として、PC クラスタ用に開発されたネットワークインタフェース Martini [19] をマルチコンテキスト FPGA で実現する場合の構成法について検討する。

2. マルチコンテキスト型 FPGA

2.1 制御モデル

マルチコンテキスト型 FPGA は、図 1 に示すように、チップ内に構成情報メモリを複数セット持たせて、これをマルチプレクサにより切り替えることにより瞬時に構成を変更する機能を持った FPGA である。富士通より 1989 年に ROM を用いたマルチコンテキスト型の MPLD が提案され、我々はこれを RAM 型にした構成を基に、コンテキストの切り替えにデータ駆動型制御を利用し、外部 RAM との入れ替え機構を装備した WASMII [3] を提案した。1990 年代後半に入り、Xilinx 社からも同様の構成が提案 [12] され、1998 年には、NEC より実チップである DRL [4] が開発され、いよいよ実現の時代を迎えた。一方、構成メモリを DRAM 型にしたチップ [10]、[11] も試作されると共に、構成情報のキャッシングの研究 [7]、論理ブロック構成法の研究 [16] なども進んでいる。

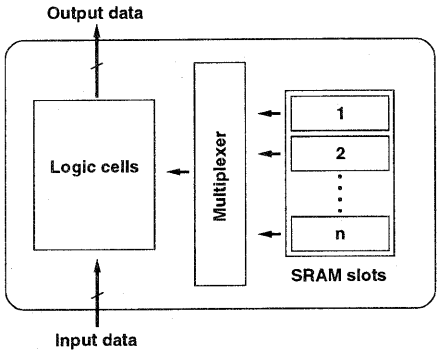

図 1　マルチコンテキスト FPGA

マルチコンテキスト型 FPGA を用いる場合の本質的な問題点は、(1) コンテキストの分割をどのように行うか (2) コンテキストの切り替えとコンテキスト間のデータ転送をいかに制御するか、という点である。(1) の問題は、複数 FPGA への回路分割と類似した点があるが、分割された回路は互いに同時に動作することはない、という点で異なっている。(1) の問題は、具体的な取り組み [8] が既に行われているが、どちらかというと設計手法の分野の問題である。アーキテクチャ上の最も重要な点は (2) の制御法であり、この問題への取り組み方は以下のように分類される。

a) アプリケーション毎のアドホックな解法:

アプリケーションに応じて、回路を分割し、制御法を設計する。DRL による DES 復号処理 [15] などがあり、最大の性能を得ることが可能だが、設計者の負担が大きい。構成情報間のデータの引き渡しも、設計者が管理する必要がある。

b) 決定順序型:

コンテキスト利用の順序を決めておき、これに従って回路構成を切り替えて行く方法。身次によるロータリー型コンピュータ [2] がこれに相当する。この方法では、コンテキスト間にレジスタを設けて、実行したコンテキストの結果を次のコンテキストに入力する。一般的には、コンテキストの処理が終わったことを何らかの方法で検出して、コンテキスト切り替えを行うが、回路エミュレーション用のマルチコンテキスト型 FPGA ではクロック毎に構成情報を切り替える。最も簡単かつ現実的な方法だが、利用される分野が制限される。

c) データフロー制御:

コンテキストの切り替えを自然に行う手法として我々が早い時期に WASMII として提案した手法 [3]

で既にDRLチップ上で実現されている[14]。この方法では、コンテキストの処理の終了は、トークンが全て出力されたことにより検出し、次のコンテキストの起動はトークンが揃ったことを検出して行われる。このため、外部にトークン到着を検出するレジスタ群と、トークンをルーティングするスイッチを必要とする。様々なアプリケーションによる評価の結果、レジスタへの同時書き込み数、レジスタからの同時読み出し数が、性能を決定することが明らかになりつつある[8]。

d) プログラム格納型制御との組み合わせ:

プログラム格納型計算機に組み込んだ形で、コンテキストを制御する方法。この場合、マルチコンテキストFPGA部は、Reconfigurable Function Unit (RFU) として、プログラム格納型計算機のデータパスの一部として用いられる。プロセッサがRFU用の命令RFUOPをフェッチすると、所定のコンテキストが起動され、汎用レジスタ経由でデータが渡されて、コンテキスト内で一連の処理が実行され、結果は汎用レジスタに戻される。RFUOPは、コンパイラにより抽出されたループ等である。明示的にマルチコンテキスト型を想定していないが、Garp[9]やCHIMAERA[13]で用いられている方法がこれに相当し、マルチコンテキスト型でも適用可能と考えられる。この場合、データの受け渡しを行うレジスタが性能上の問題点となり、GarpではShadow registerの利用が提案されている。

e) 自律制御型:

プログラム格納型計算機との組み合わせの形を採らず、コンテキストカウンタのみを用いて自律制御を行う方法。この方法では、命令フェッチを行わず、コンテキストカウンタに従ってコンテキストフェッチを行い、基本的には順番にコンテキストの実行を進めていき、コンテキスト制御系回路を含むコンテキストの出力によって、コンテキストカウンタを変更することで、実行順序を変更する。この方法は、ちょうどVLIW型の計算機が長大な命令をフェッチして実行するのと同様、コンテキストの構成情報をフェッチして実行していくことになる。興味深い方法だが、コンテキストを数多く必要とすることから、実現はやや先のこととなろう。

f) 一般回路のマルチコンテキスト型FPGA制御:

ここまでの方式のうち決定制御型のクロック単位での切り替え以外のほとんどでは、テーブル参照や、パタンマッチなどを含む広い意味での「演算処理」を行うことを想定していた。しかし、実際の応用分野では、純粋に演算目的のReconfigurable Architectureは、汎用CPUやDSPとの厳しい競争にさらされる。そこで、より一般的な、通信や制御の分野における、制御と演算処理を組み合わせた処理をマルチコンテキスト型FPGAで行う方法の提案を行った[18]。この場合、今まで述べた方法に比べてコンテキストの制御が格段に困難になる。これは、(1)コンテキスト内には制御に必要な自律的な順序回路を複数含む可能性がある点 (2) コンテキストの切り替え後も矛盾無く制御を継続する必要がある点 (3) 外部からのイベントに従って速やかにコンテキストを切り替える必要がある点。文献[18]では、これらを実現するために、ハードウェア処理の流れに従い、コンテキスト内外にスケジューラを設け、限定した条件の下でコンテキストの切り替えを許す一般的な方法を提案した。本報告では、この方法を複雑な構成のネットワークインタフェースMartini[19]に適用し、その効果を評価する。

3. Martiniのマルチコンテキスト FPGA上での実装

3.1 マルチコンテキスト設計

文献[18]で提案した方法では，マルチコンテキスト設計の枠組みとして図2に示す階層的な概念的構成を想定する。

設計のトップレベルは常に回路化されている部分と，必要に応じて回路化される部分から構成される．前者をFixed Region，後者をShared Regionと呼ぶ．Shared Regionではコンテキストスイッチによりハードウェア構成が切り替わる．ここで，あるハードウェアコンテキストの回路情報が，実際の回路として用いられることを，コンテキストがアクティベイトされると呼ぶ．

Shared Regionはさらに Context Group，Context，Moduleの3階層を形成する．Shared Regionは1つ以上のContext Groupから，Context Groupは1つ以上のContextから，Contextは1つ以上のModuleからそれぞれ構成される．本報告の実装例では、簡単のため1 Context = 1 Moduleとしている。コンテキストのスケジューリングおよびスイッチングはContext Group単位で行われる．すなわち，各Context Groupで，ある瞬間にアクティベイトされているContextは常に一つのみである．

各Contextには Context Managerが，各Context Groupには Context Schedulerがそれぞれ付随しコンテキストスイッチングを実現するための各種制御を

行う．Context Managerの役割は以下の5つである．

- 担当するContextの状態遷移の管理
- スケジューリングに必要な情報の保持
- Moduleの動作制御
- 隣接Contextとの通信
- Context Schedulerとの通信

また，Context Schedulerの役割は以下の4つである．

- 担当するContext Groupの状態遷移の管理
- Contextのスケジューリング
- 隣接Context Groupとの通信
- Context Managerとの通信

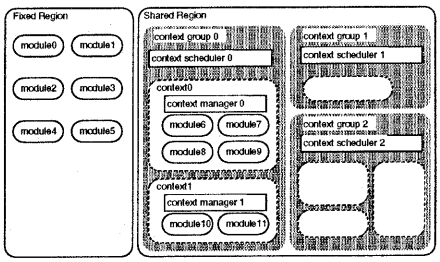

図2　設計の構成

上記の構成は，DRLなど部分的にコンテキストスイッチを行う機能を持つFPGAにより実現することが可能である．

3.2　Martiniの全体構成

新情報処理開発機構のプロジェクトで開発されているRHiNET [21]およびMEMOnet [22]は、机上のPC間を光ネットワークと高速スイッチで接続することで、LAN並の接続距離でSAN並の信頼性と高速性を実現して並列分散処理を行うネットワークコンピューティング環境である。

Martiniはこのプロジェクトのコアとなる0.14μmプロセスのASICであり、コアプロセッサ、内蔵メモリ、強力なハードウェア転送機構、ホストとネットワーク双方に対する多様なインタフェースを備えたネットワークインタフェースコントローラLSIである。単純なリモートメモリのライト機構(PUSHプリミティブ)とリード機構(PULLプリミティブ)のみをハードウェアで高速処理し、それ以外の複雑な機構や生起率の低いイベントに対してはハードコアがコアプロセッサに割込みをかけ、処理の代行を要求する。[17]

ここでは、このMartiniのプリミティブ処理用コアをマルチコンテキストFPGAで実装することを考える。

プリミティブ処理部は図3に示すように、ホストからの要求により動作する送信部と、ネットワークからのパケットにより起動を受ける受信部、およびアドレス変換を行うTLB(PATLB)から構成される。ここで、PATLBは論理アドレス-実アドレス変換を行う部分であり、送信部、受信部で共有されるため、Fixed Regionに置く。これ以外の送信部、受信部をそれぞれマルチコンテキスト化して設計を行った。本稿はページの関係上Martiniの構造および動作については文献[20]に譲る。

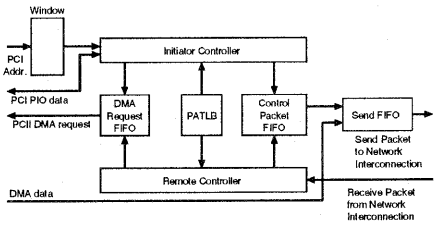

図3　プリミティブ処理部

3.3　受信部のマルチコンテキスト化

3.3.1　基本方針

現在のMartiniの受信部(Remote Controller)は、パケットを受信してヘッダを解析し、受信用のDMA転送要求を出すと共に、応答パケット作成要求を送信部に送る役割を持つ。

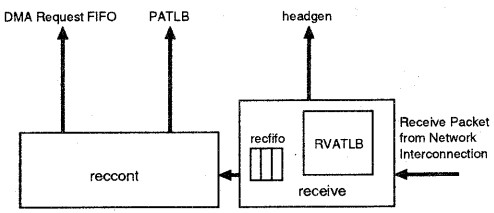

図4　Remote Controllerの構造

図4に構造を示すように、以下の二つのモジュールにより設計されている。

- receive：受信部フロントエンド。パケットを受信し、ヘッダを解析し、受信用のTLB(RVATLB)によりアドレス変換を行い、抽出した情報をバックエンドへのFIFOに格納する。Martiniは受信チャネルを複数持つが、現在のMartiniでは基本的にパケットは一つずつ順番に扱う。応答パケットを必要とする場合は、受信用の要求起動用バッファにヘッダから取り込んだ情報を書き込み、送信部を起動する。receiveは、ヘッダがハードウェアで処理できないものであった場合やRVATLBがミスした場合等に、コアプロセッサに対して割込をかけ、代行処理を要求する。

- reccont: FIFO(図4中のrecfifo)を経由してフロントエンドから送られた情報に応じて、DMA転送の起動を行う。この際、PATLBを引いてアドレス変換を行う。DMA転送サイズが大きく、アドレスがページ境界にまたがる場合は、引き直しの制御を行う。reccontは、PATLBがミスした場合に、コアプロセッサに対して割込をかけ、代行処理を要求する。

Martiniは、日立の0.14μmCMOS Embedded Arrayで実装されているが、今回はマルチコンテキストFPGAで実現するものと考える。ただし、実際の利用可能なマルチコンテキストFPGAであるNECのDRLは、現時点ではHDLからの自動合成が不可能であるため、今回は、面積評価にAltera社FLEX10Kで実現した場合のデータ(Synopsys社FPGA Compiler IIにより合成)を用いた。すなわち、今回の評価では、FLEXがDRL同様にマルチコンテキスト化されたと考える。この場合のLogic Block数を評価すると表1に示すようになる。

表1　受信部のLogic Block数

receive	1513
reccont	981
コアプロセッサ	3046

ここで、現在のreccontは、基本的には複数の受信チャネルから到着するパケットを並行処理せず、reccontによるDMAを起動して転送するのを待ってから次のチャネルの処理を行っている。ここをマルチコンテキスト化することにより、複数の受信チャネルからのパケットを並行して解析可能にする。Martiniでは、ハードウェアで扱うパケットは、チャネル0とチャネル2から到着することになっているので、それぞれを扱うreceive0とreceive2の二つのContextに分離する。分離することによって、それぞれのハードウェア量は現在のreceiveよりも減らすことができる。receiveとreccontの処理は、基本的にはシーケンシャルなので、ここも別のContextとして考え、マルチコンテキスト化の対象とする。さらに、core CPUは、テーブル参照ミス、未定義パケットの到着等の場合にのみ動作するので、これもContextとして定義する。なおRVATLBは容量が大きいため、PATLB同様、Fixed Regionに置くこととした。

ここで、コアプロセッサが問題となる。Martiniでは本格的なMIPS互換CPUを制御に用いているため、3046Logic Blockを必要とする。しかし、プロセッサは、ここではTLBミスの解決等単純な処理のみを行えばよいため、簡単な16bitプロセッサで充分である。この場合、後に評価するように995 Logic Block程度の面積消費で実現可能である。したがって、receiveの必要ハードウェア量が分離により減ると考えて、約1000 Logic Block程度の面積のマルチコンテキストブロックを用意すれば、receive0, receive2, reccontと同様に、コアプロセッサをも一つのコンテキストとして、他の3つのモジュールの並行処理を実現することができる。

3.3.2　コンテキストの切り替えとスケジューラ

今回の実装では文献[18]の一般化されたスケジューラを簡単化し、以下の構成のスケジューラを用いている。このスケジューラでは、コンテキストの切り替えは、外部からのイベントであるForeign Request(タイマーからのタイムアウトを含む)と、内部発生的なイベントに分られる。派生し得るイベントはそれぞれのContextについて以下の通りである。

- receive0,2：チャネル0,2からの受信パケットの到着
- reccont: receiveからのデータの到着
- コアプロセッサ: チャネル1,3からの受信パケットの到着、receive0,2,reccontからのTLBミスによるプロセッサの処理要求

そこで、各Contextは、

- 動作中で基本的には切り替え不可
- サスペンデッド
- Idle状態
- 例外発生CPUに切り替え要求

の四つの状態から成るContext Managerを持つ。Context Group上のContext SchedulerはContext Managerの状態に基づき現在動作中のコンテキストの状態と、イベントからコンテキストのスイッチを決める。また、コンテキストの切り替えを行うためには、receive0,2,reccontなどのコンテキストスイッチの対象となるハードウェアモジュールの状態遷移を受信部以外のモジュールに影響を与えることなく停止させることが必要である。一般のステートマシンではこれは困難であるが、Martiniのステートマシンは、コアプロセッサの「代行処理機構」の実現のため、特定の状態で、周囲のモジュールに影響を与えることなく停止させることができるように設計されている[17]ため、問題が起きることはない。

3.4　送信部のマルチコンテキスト化

3.4.1　基本方針

ホストPCまたはコアプロセッサは、windowに必要データを書き込んだ後、特定アドレスへの書き込む操作(キック)により、パケット転送処理を起動す

る。Martiniの送信部(Initiater Controller)は、この起動を受けて、パケットを送信するモジュールから構成される。アドレス変換用のPATLBのアクセスの他に、内部にPGID TableやCONT Table等のテーブルを持つが、これらは、プロセス間の通信保護と、転送の終了をDMAによりホストに知らせるための変換用テーブルである。送信部は、デッドロックを防ぐために以下の構成に分かれている。

- window : 送信部フロントエンド。ホスト、Core CPU、受信部からの起動を受け付ける部分で、受け付けた要求は、FIFOに格納した後、チャネル0に対する要求はwcontiで、チャネル2に対する要求はwcontrで処理する。
- wconti: ホスト、Core CPUからのチャネル0に対するpush/pull要求の処理を行う。この際、PGID Table, Cont table, PATLBなどのテーブルを参照し、アドレス変換を行う。転送要求が大きい場合は所定のパケットに切り分ける作業も含み、さらにアドレスがページ境界にまたがる場合は、引き直しの制御も自動的に行う。wcontiは、起動された要求が未定義だったり、PATLBがミスした場合に、コアプロセッサに対して割込をかけ、代行処理を要求する。
- wcontr: 受信部からの要求に対する応答パケットをチャネル2に対する送信する。この際、PATLBを引いてアドレス変換を行う。pull要求の応答時に、転送要求が大きい場合は所定のパケットに切り分ける作業も含み、さらにアドレスがページ境界にまたがる場合は、引き直しの制御も自動的に行う。wconti同様、PATLBがミスした場合に、コアプロセッサに対して割込をかけ、代行処理を要求する。
- headgen0, headgen2: それぞれwconti, wcontrからの要求に基づきパケットヘッダを生成する。

Initiater Controllerの構造を、図5に示す。

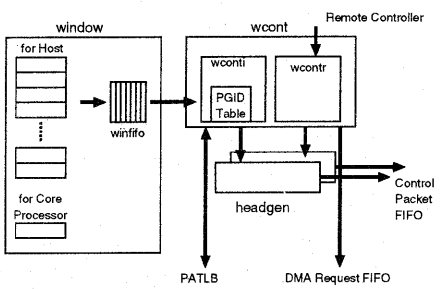

図5 Initiater Controllerの構造

現在のMartini送信部をFLEX上に置く場合、各モジュールの必要LB数は表2の通りである。

表2 送信部のLogic Block数

wconti	1291
wcontr	1004
headgen0/2	362

ここで、基本的にwindow部はほとんどがメモリから出来ており、常に要求を受け付けるため、Fixed Regionに置くことにする。また、pgid, conttableなどのアドレス変換用のテーブルも、かなり容量の大きなレジスタファイルで構成されているため、これもFixed Regionに置く必要がある。

現構成ではwcontiとheadgen0, wcontrとheadgen2まとめて、チャネル0用の要求処理部、チャネル2用の要求処理部をそれぞれコンテキストとし、これにCore CPUのコンテキストを加え以下の3つのコンテキストにまとめる。

- wconti + headgen0 → wconti
- wcontr + headgen2 → wcontr
- core CPU

3.4.2 コンテキストの切り替えとスケジューラ

コンテキストの切り替えは、それぞれのContext Managerによって行われる。Foreign Requestの種類は以下の通りである。

- wconti, wcontr : 対応windowのキック
- core CPU: PATLBミスによるCPU代行処理要求

各Context Managerの状態遷移は受信部のそれと同様である。ここで、送信部はサスペンデッド状態が実際に用いられているのが特徴的である。送信は対応するチャネルのバッファが一杯になると不可能になるので、この場合にそれぞれのコンテキストはサスペンデッド状態になる。サスペンド状態のコンテキストは他のコンテキストが実行可能であれば、切り替わり、バッファフル状態でなくなれば、実行可能状態に戻る(注1)。

3.4.3 コンテキスト間のデータ受け渡しと入出力の切り替え

コンテキスト間のデータ受け渡し用には、レジスタ群が必要になるが、これはマルチコンテキスト部の周辺にコンテキストスイッチを行わないシングルコンテキスト部を用意することにより実現する。今回の受信部の例では、receive0,2からreccontに受け

(注1) : 上記の制御は若干一般的な手法[18]と異なっている

渡すために、それぞれ独立のレジスタを確保する必要がある。しかし、もともとMartiniは、フロントエンドとバックエンドはFIFO(recfifo)を設けているため、今回はこれを利用することで対処している。また、送信部へのデータ受け渡しもバッファを共有している。このため、今回の実装ではデータ転送領域は、ほとんど余分のハードウェアを必要としないが、一般的にはハードウェア量の増大を招く危険性がある。

また、コンテキスト切り替え時の入出力処理も問題となる。アクティブでないコンテキストに対応する出力には適切な値を設定する必要がある。これはreceive0とreceive2のように基本的に入出力を共有する同質のコンテキストの場合は、全く問題がないが、receive0,2とreccontなど全く異質のコンテキストの場合に問題になる。これを解決するためには、周辺シングルコンテキスト部に、出力制御用のジョイントを設け、アクティブでないコンテキストについては、最後の出力を記憶しておき、出力しつづける機構が必要になる。この部分の構成は今後の課題である。

以上述べたの方針に基づき、マルチコンテキスト版Martiniを設計した。現状で、push/pullなどの基本動作はシミュレーションにより確認済みである。しかし、今回のシミュレーション環境はマルチコンテキスト機能までシミュレーションを行っていない。このため、以下のようにしてこれを擬似的に実現している。(1)アクティブでないコンテキストはクロックを止めてしまう。(2)アクティブでないコンテキストの出力は、クロックを止めた状態で固定する。これは、本来マルチコンテキストFPGA上で実装する場合は、外部にレジスタを設けた上で、さらにマルチプレクサで切り替える必要がある。

4. ハードウェア量の評価

4.1 マルチコンテキストFPGAが有利になるための条件

マルチコンテキストFPGAのコンテキスト一つで実現可能なゲート数をG、このために必要なConfiguration MemoryをCとし、それぞれの面積をar(C), ar(G)とする時、圧縮係数αは、

$$\alpha = ar(G)/ar(C)$$

となる。ここでパーマネント部とスケジューラの面積をそれぞれarea(P)とarea(S)とし、最大コンテキスト数をnとすると、コンテキスト付きハードウェアで実現可能な最大ハードウェア面積area(M)は、

$$ar(M) = ar(P) + n \times ar(G)$$

一方で、このために必要とされる面積ar(R)は、

$$ar(R) = ar(P) + ar(S) + ar(G) + n \times ar(C)$$
$$= ar(P) + ar(S) + ar(G)(1 + n/\alpha)$$

となる。ここで、ar(M)がar(R)より充分大きくなることが、コンテキスト付きハードウェアの成立の条件となる。このためには、αの値が大きくなくてはならない、NECによるDRLでは35%のオーバヘッドで8つのコンテキストを取ることができる[?]。つまり、単純計算すると、コンテキスト1つ当たりオーバーヘッドが4.375すなわちα=22.9となる。これを飛躍的に大きくするには、DRAM混載か、外部RAMとの高速圧縮転送が必要となる。

4.2 必要面積の評価

マルチコンテキスト化の結果、必要Logic Block数は、受信部に関しては、表3のようになった。

表 3 マルチコンテキスト化した受信部のLogic Block数

receive 0,2	806
reccont	683
16 bit core CPU	995
scheduler	22

receiveは、コンテキストに分けたため個々のハードウェア量は減ると共に、合成時の最長遅延パスも約20%減り、動作速度も向上した。αを22、nを4、受信部のみについて先の式を用いると以下のようになる。

ここでは、必要Logic Block数を面積と考える。表1より、マルチコンテキスト化しないFPGAでのLogic Block数の総計は、以下のようになる。ただし、ここでは、16bitの簡単なコアプロセッサを利用しているとする。

$$(1513 + 981 + 995) \times (1 + 1/22) = 3648$$

となる。一方、マルチコンテキスト化した場合のLogic Block数は、

$$(995 + 22) \times (1 + 4/22) = 1200$$

であり、必要面積は1/3以下になっていることがわかる。また、わずかながら遅延パスも短くなったことから、個々のモジュールの性能はアップしていることがわかる。さらに元々の設計では不可能だったチャネル単位の並行処理が可能になっている。

次に、送信部に関しては、表4の結果を得た

表 4 マルチコンテキスト化した送信部の Logic Block 数

wconti	1421
wcontr	1121
16 bit core CPU	995
scheduler	22

受信部同様、表2より、マルチコンテキスト化しないFPGAでの必要Logic Block数は以下のようになる。

$$(1291 + 1004 + 362 + 362 + 995) \times (1 + 1/22) = 4195$$

となる。一方、マルチコンテキスト化した場合の必要Logic Blockは、

$$(1421 + 22) \times (1 + 4/22) = 1702$$

となり、面積の点で約4割となっている。ただし、マルチコンテキスト化しない場合は、受信部と送信部とでCore CPUは共有できるので、この点は差し引いて考える必要がある。また、今回の面積評価は、コンテキスト間のデータ保持用のレジスタの多くは換算されているが、信号切り替え用のマルチプレクサの面積は換算されていない。しかし、これらを考えに入れても、Martiniにおいてマルチコンテキスト化は有効であると考えられる。

5. おわりに

本報告におけるマルチコンテキスト化の制御は、Martiniの動作に依存しており、文献[18]に示した一般的な手法から逸脱している点がある。また、Martiniの構造を利用して、コンテキスト間のデータ保持用のレジスタを省略している部分も多く、設計者による最適化が結果を改善している部分が大きい。このようなことを行わず、完全に一般的な方法により自動的にマルチコンテキスト化された場合の面積は、かなり悪化することが考えられる。今回のケーススタディをいかに自動的なマルチコンテキスト化手法に反映していくかが今後の課題となる。

謝　辞

Martini の Core CPU の設計を行った山本淳二博士他RWCPつくば研究所の皆様、および今回の評価に協力してくれた慶應大学理工学部の安福健太君に感謝する。

文　献

[1] 吉見昌久, “マルチファンクションプログラマブルロジックデバイス,” 公開特許公報 (A), 平 2-130023, 1990.
[2] 身次、“ロータリー型コンピュータと仮想回路,” 情処全国大会論文集 pp.6-109, 1992.
[3] X.-P. Ling, H. Amano, “WASMII: A Data Driven Computer on a Virtual Hardware” Proc. FCCM '93, pp. 33–42, 1993.
[4] M.Yamashina, M.Motomura, “Reconfigurable Computing: Its concept and practical embodiment using newly developed DRL LSI,” Proc. ASP-DAC 2000, pp.329–332, (2000).
[5] Y.Shibata, H.Miyazaki, X.Ling, H.Amano, “HOSMII: A Virtual Hardware Integrated with DRAM,” Proc. of Parallel and Distributed Processing Workshop, LNCS 1388, pp. 85-90, (1998).
[6] 山品、“再構成可能なマイクロプロセッサ,” STARC シンポジウム 2001 予稿集, (2001.9)
[7] Z.Li,K.Compton,S.Hauck, “Configuration Caching Techniquies for FPGA,” Proc. of FCCM2000, pp.87-96, 2000.
[8] 柴田、高山、岩井、天野、“ データ駆動型仮想ハードウェアにおける自動ページ分割手法,” 情報処理学会論文誌, Vol.42, No.4, (2000.4)
[9] J.Hauser, J.Wawrzynek, “ Garp: A MIPS processor with a reconfigurable coprocessor,” Proc. of FCCM97, pp.12-21, 1997.
[10] M.Motomura, and et. al. “An embedded DRAM-FPGA chip with instantaneous logic reconfiguration,” Proc. of the Symposium on VLSI Circuits, 1997.
[11] D.Kawakami, Y.Shibata, H.Amano, “Design of Multicontext FPGA embedded DRAM for Virtual Hardware HOSMII,” Proc. of ASP-DAC 2001, A1-9, 2001.
[12] S.Trimberger, D.Carberry, A.Johnson, J.Wong, “A Time-Multiplexed FPGA,” Proc. of FCCM, pp.22-28, 1997.
[13] S.Hauck, T.Fry, M.Hosler, J.Kao, “The Chimaera reconfigurable functional unit,” Proc. of FCCM97, pp.389-398, 1997.
[14] Y.Shibata, et. al, “A Virtual Hardware System on DRL,” Proc. of FCCM2000, pp.295-296,2000.
[15] M.Yamashina, M.Motomura, “Reconfigurable Computing: Its Concept and a Practical Embodiment using Newly Developed DRL LSI,” Proc. ASP-DAC 2000, pp.329-332, 2000.
[16] 飯田、末吉, “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,” 信学報 CPSY2001-79 (デザインガイア) (2001.11)
[17] 天野、渡邊、土屋、金子、工藤, “クラスタコンピュータ用ネットワークインタフェースチップ Martini における代行処理機構,” 信学報 CPSY2001-54 (2001.10)
[18] 金子、宇野、渡邊、山本、工藤、天野, “ハードウェア設計のマルチコンテキスト化手法,” 信学報 CPSY2001-78 (デザインガイア) (2001.11)
[19] 山本、渡邊、他 “RHiNET の概要と Martini の設計/実装” 情処研報 2001-ARC-144-7 (SWoPP2001).
[20] 渡邊、山本、他 “RHiNET/MEMOnet ネットワークインタフェース用コントローラチップ Martini” 情処研報 2001-ARC-144-9 (SWoPP2001).
[21] T.Kudoh, et.al., RHiNET: A network for high performance parallel processing us-ing locally distributed computers , IWIA 99 (1999.11)
[22] N.Tanabe, et.al., MEMOnet : Network interface plugged into a memory slot , IEEE International Conference on Cluster Computing (CLUSTER2000), pp.17-26 (2000.11)